Tal Ben-Nun

Reproducible Parallel and Distributed Deep Learning

What we will cover

- **What is Machine Learning and Deep Learning**
- **How can ML/DL be parallelized, distributed, or otherwise optimized**
- **The Deep500 metaframework**
- **Hands-on reproducible DL and caveats**
- **Hands-on distributed DL**
  - Centralized
  - Decentralized
  - Inconsistent

Additional Pointers

- **Demystifying Parallel and Distributed Deep Learning: https://dl.acm.org/citation.cfm?id=3320060**

- **Deep500: https://spcl.inf.ethz.ch/Publications/.pdf/deep500_ipdps19.pdf**

- **The samples in this tutorial:**
  - https://www.github.com/deep500/eurompi19
  - https://nbviewer.jupyter.org/github/deep500/eurompi19/tree/master

What is Deep Learning good for?

Hey Cortana · Hey Siri · Google now · skype Translate · XBOX · amazon alexa · Baidu 百度 · ...

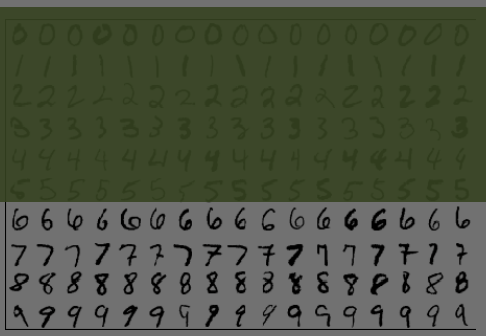
Digit Recognition

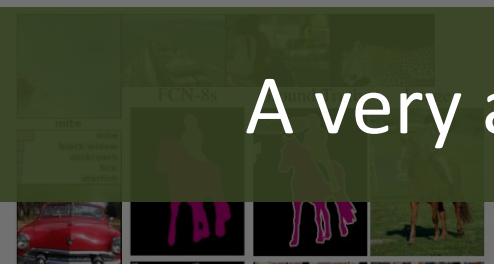
Object Classification
Segmentation

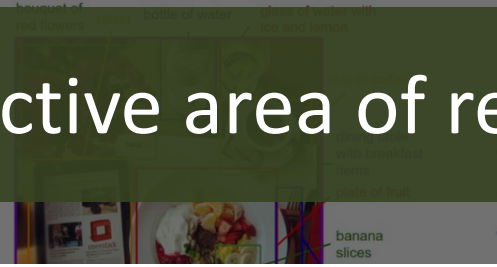
Image Captioning
GANs

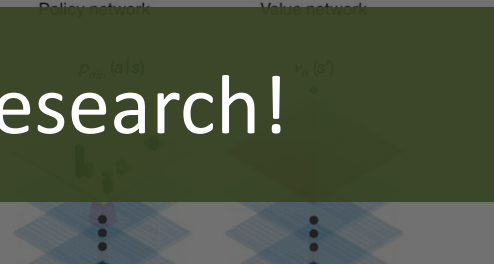
Gameplay AI
Translation

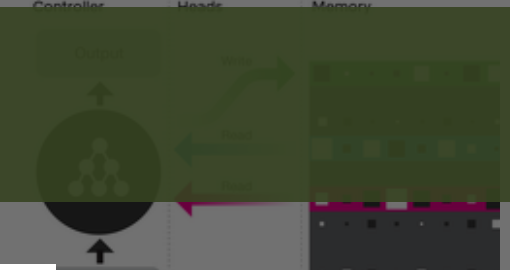
Neural Computers
Text/Voice Synthesis











A very active area of research!

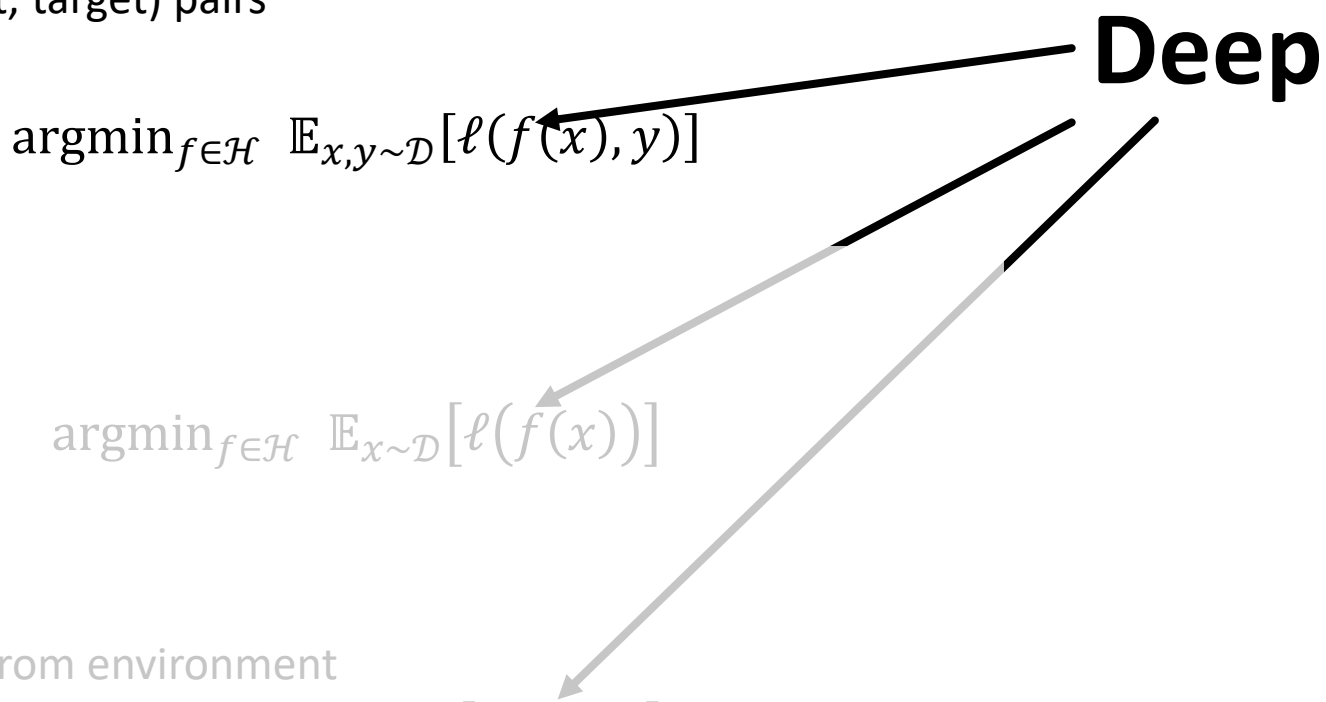
Year	2012	2013	2014	2015	2016	2017	2018
cs.AI	1,081	1,765	1,022	1,105	1,929	2,790	4,251
cs.CV	577	852	1,349	2,261	3,627	5,693	8,583

35 papers per day!

1989 2012 2013 2014 2016 2017 2019

Classes of Machine Learning Problems

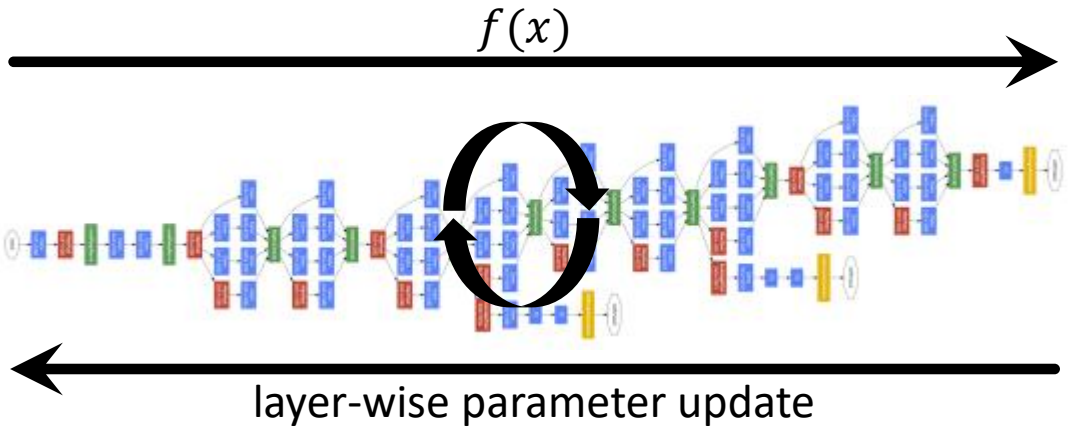
- **Supervised Learning**
  - Learn to predict mapping from (input, target) pairs

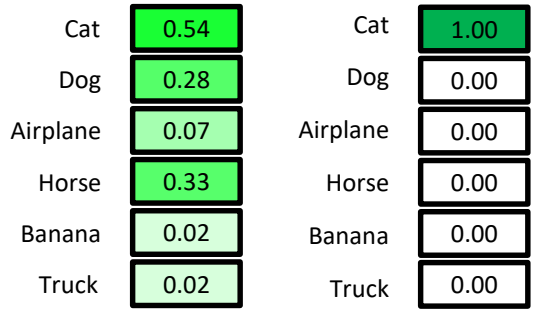


- **Unsupervised Learning**
  - Learn to find patterns from inputs
- **Reinforcement Learning**
  - Learn policy that maximizes reward from environment

A brief theory of supervised deep learning (minibatch SGD)

labeled samples $x \in X \subset \mathcal{D}$





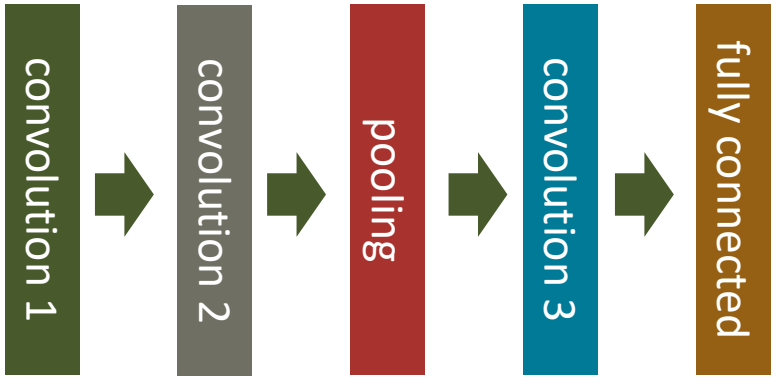
label domain Y true label $l(x)$

$$f(x): X \to Y$$

network structure (fixed) weights w (learned)

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}(\dots f_1(x)\dots)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ ... $f(x)$



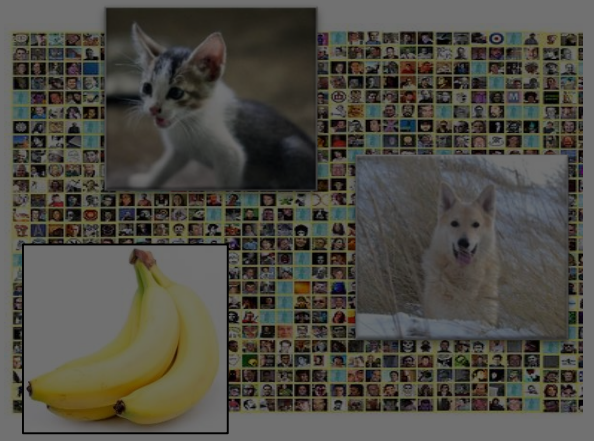
$$\ell_{sq}(w,x) = \big(f(x) - l(x)\big)^2$$

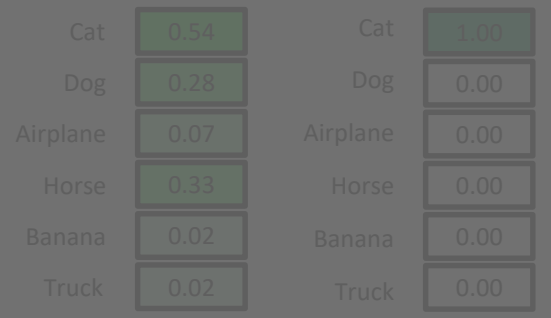
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

A brief theory of supervised deep learning (minibatch SGD)





≥TBs of random access

100MiB-26GiB and beyond

22k-millions

Deep Learning is Supercomputing!

Stochastic Gradient Descent

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

```
for t = 0 to T do                                   ▷ Stopping condition
    x ← Random element from X                       ▷ Sample dataset
    o_1 ← f_1(x)
    for i = 2 to layers do                          ▷ Forward evaluation of ℓ
        o_i ← f_i(o_{i−1})
    end for
    for i = layers−1 to 1 do                        ▷ Compute gradient of ℓ via backpropagation
        ∇o_i ← ∂ℓ/∂o_i (o_{i−1}, o_i, ∇o_{i+1})     ▷ Gradient w.r.t. data
        ∇w_i^(t) ← ∂ℓ/∂w_i (o_{i−1}, o_i, ∇o_{i+1}) ▷ Gradient w.r.t. layer parameters
    end for
    w^(t+1) ← w^(t) + u(∇w^(t), w^(0,...,t), t)     ▷ Update weights with function u
end for
```

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

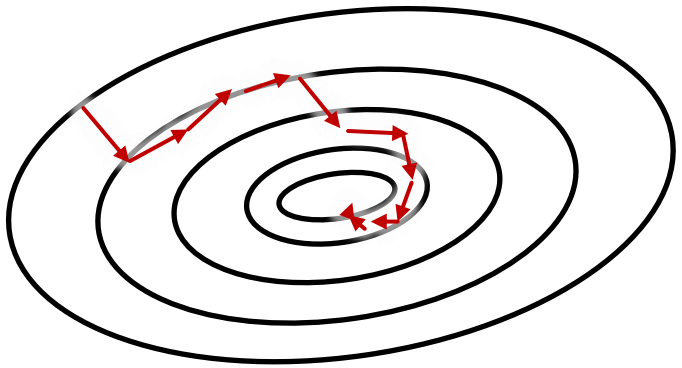
pooling

...

convolution 3

$f(x)$ fully connected

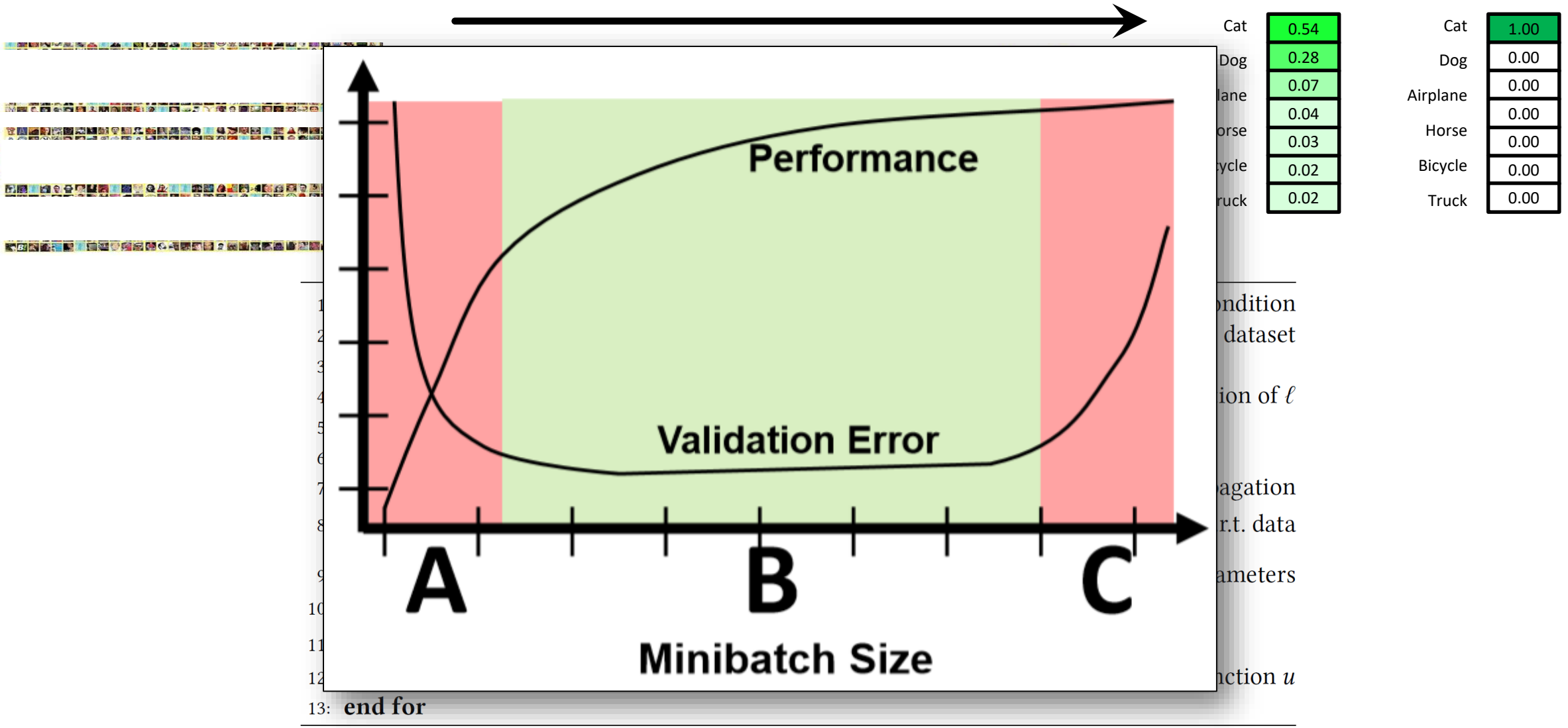
- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla \ell(w^{(t)}, z) \quad = w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$
Nesterov Momentum [Nesterov 1983]	$w^{(t+1)} = w^{(t)} + v_t; \quad v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla \ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [Duchi et al. 2011]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t} + \varepsilon}}; \quad A_{i,t} = \sum_{\tau=0}^{t} \left(\nabla w_i^{(t)}\right)^2$
RMSProp [Hinton 2012]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t} + \varepsilon}}; \quad A'_{i,t} = \beta \cdot A'_{t-1} + (1-\beta)\left(\nabla w_i^{(t)}\right)^2$
Adam [Kingma and Ba 2015]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)} + \varepsilon}}; \quad M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1-\beta_m)\left(\nabla w_i^{(t)}\right)^m}{1-\beta_m^t}$

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

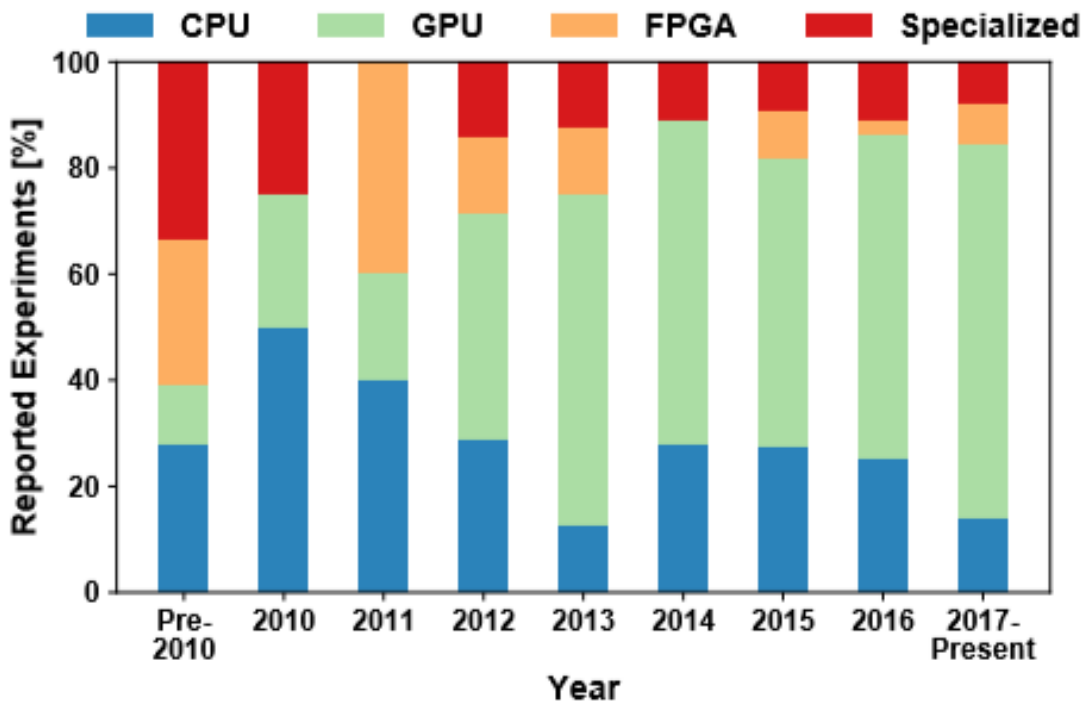
Minibatch Stochastic Gradient Descent (SGD)

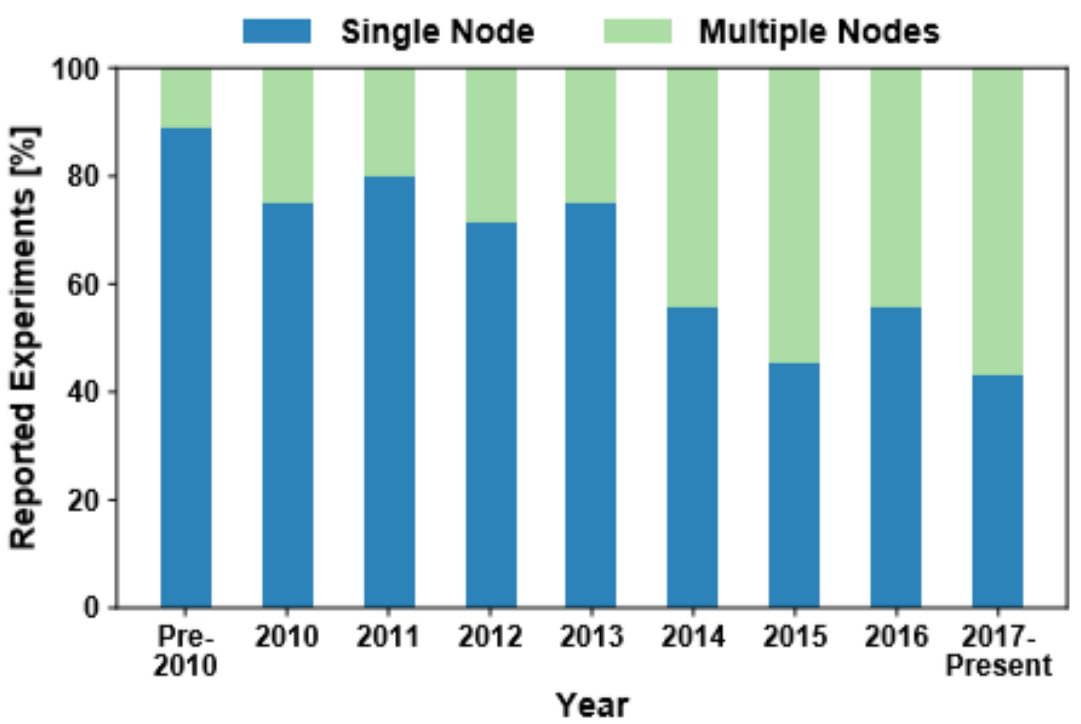


T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



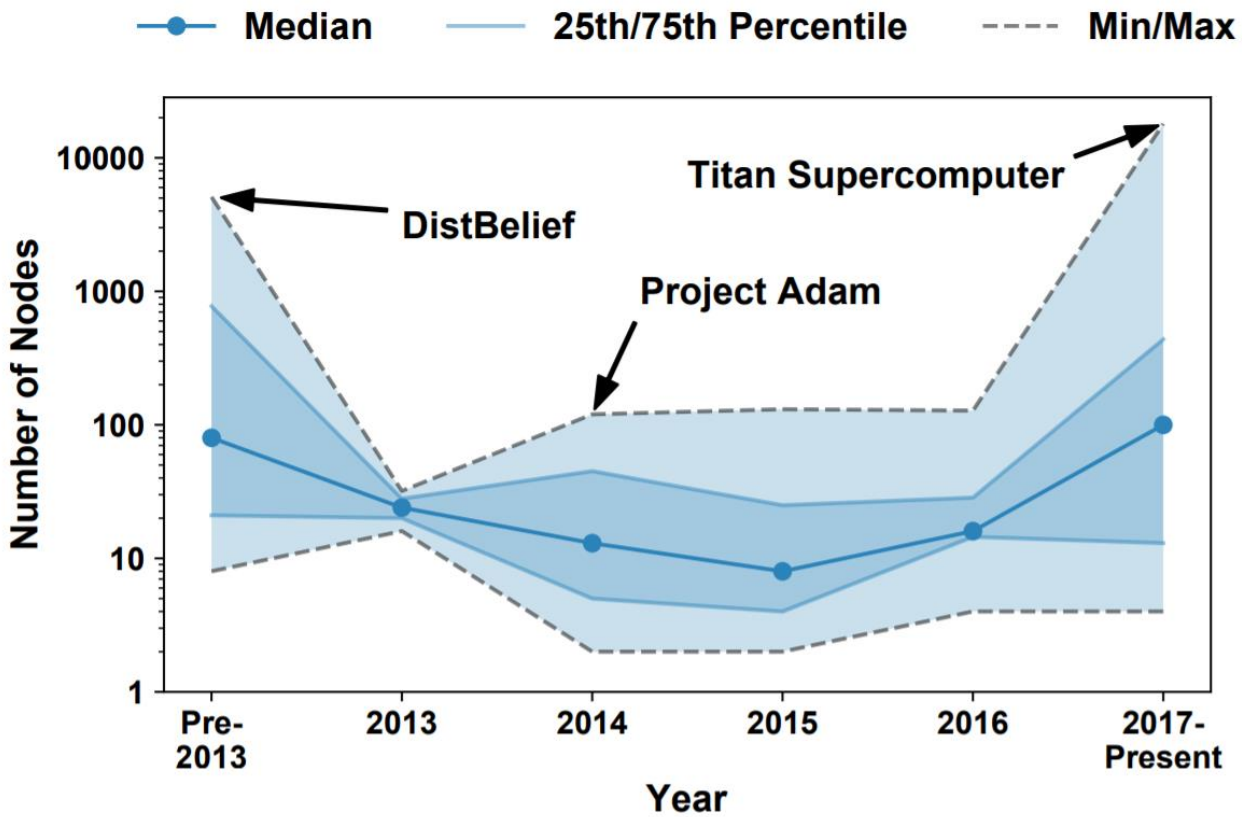


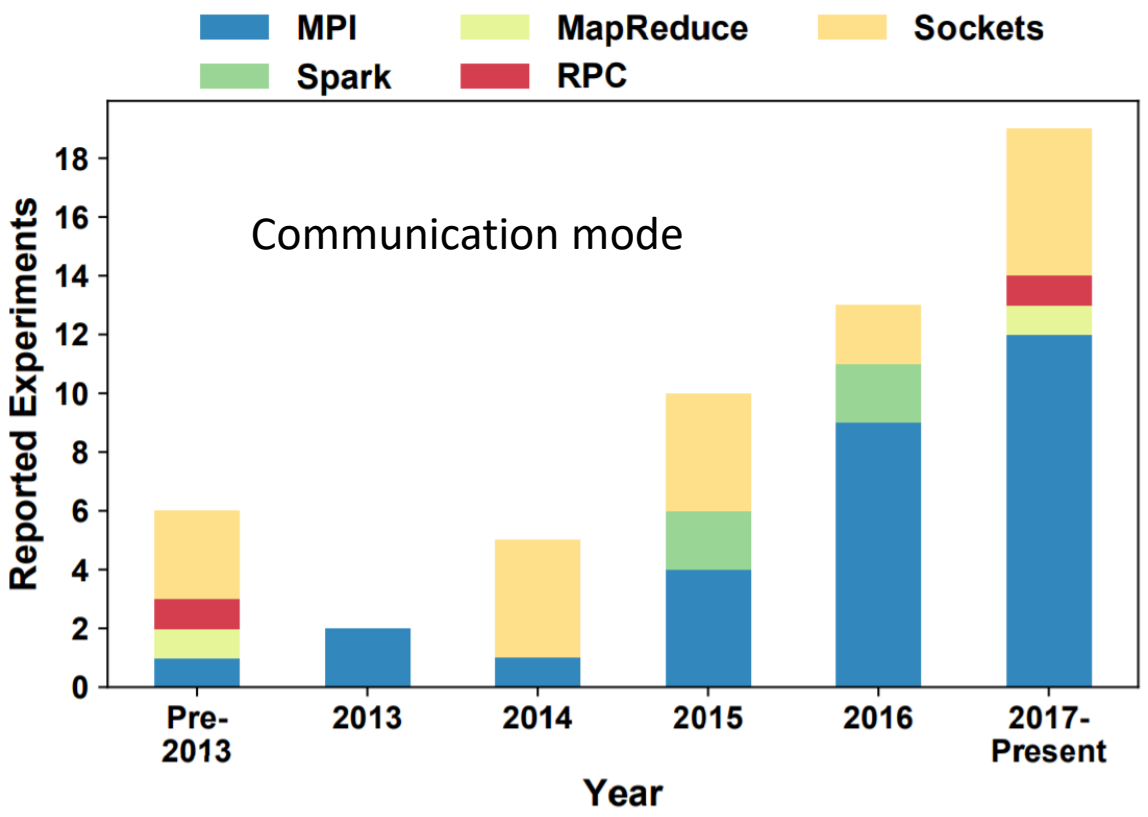
Deep Learning is largely on distributed memory today!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning





Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Reproducing and Benchmarking Deep Learning

- **End result – generalization**

Benchmark	Focus			Metrics									Criteria			Customizability			DL Workloads						Remarks
	Perf	Con	Acc	Tim	Cos	Ene	Util	Mem	Tput	Brk	Sca	Com	TTA	FTA	Lat	Clo	Ope	Inf	Ops	Img	Obj	Spe	Txt	RL	
DeepBench [39]	👍	👎	👎	👍	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👍	👎	👍	👍	👎	👎	👎	👎	👎	Ops: Conv., GEMM, RNN, Allreduce
TBD [47]	👍	👎	👎	👎	👎	👎	👍	👍	👍	👎	👎	👎	👎	👎	👍	👍	👎	👎	👎	👍	👍	👍	👍	👍	+GANs
Fathom [2]	👍	👎	👎	👍	👎	👎	👎	👎	👍	👍	👍	👎	👎	👎	👍	👍	👎	👎	👎	👍	👎	👍	👍	👍	+Auto-encoders
DAWNBench [9]	👍	👍	👎	👍	👍	👎	👎	👎	👎	👎	👎	👎	👍	👎	👍	👎	👍	👎	👎	👍	👎	👎	👍	👎	
Kaggle [21]	👎	👎	👍	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👎	👎	👍	👎	👎	👍	👍	👍	👍	👍	Varying workloads
ImageNet [13]	👎	👎	👍	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👎	👎	👍	👎	👎	👍	👍	👎	👎	👎	
MLPerf [30]	👍	👍	👍	👍	👍	👍	👎	👎	👎	👎	👎	👎	👍	👍	👍	👍	👍	👎	👎	👍	👍	👍	👍	👍	
Deep500	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	👍	

TABLE II: **An overview of available DL benchmarks**, focusing on the offered functionalities. **Perf**: Performance, **Con**: Convergence, **Acc**: Accuracy, **Tim**: Time, **Cos**: Cost, **Ene**: Energy, **Util**: Utilization, **Mem**: Memory Footprint, **Tput**: Throughput (Samples per Second), **Brk**: Timing Breakdown, **Sca**: Strong Scaling, **Com**: Communication and Load Balancing, **TTA**: Time to Accuracy, **FTA**: Final Test Accuracy, **Lat**: Latency (Inference), **Clo**: Closed (Fixed) Model Contests, **Ope**: Open Model Contests, **Inf**: Fixed Infrastructure for Benchmarking, **Ops**: Operator Benchmarks, **Img**: Image Processing, **Obj**: Object Detection and Localization, **Spe**: Speech Recognition, **Txt**: Text Processing and Machine Translation, **RL**: Reinforcement Learning Problems, 👍: A given benchmark does offer the feature. 👍: Planned benchmark feature. 👎: A given benchmark does not offer the feature.

- **Sample throughput**

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Existing Deep Learning Frameworks

System	Operators		Networks				Training			Dist. Training			
	Sta	Cus	Def	Eag	Com	Tra	Dat	Opt	Cus	PS	Dec	Asy	Cus
(L) cuDNN	👍	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎
(L) MKL-DNN	👍	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎
(F) TensorFlow [1]	👍	👍	👍	👍	👍	👍	👍	UR	👍	👍	👍	👍	👎
(F) Caffe, Caffe2† [21]	👍	👍	👍	👎	👎	👎	👎	UR	👎	👎	👍	👎	👎
(F) [Py]Torch† [10, 35]	👍	👍	👎	👍	👎	👎	👎	👍	👍	👎	👍	👎	👎
(F) MXNet [6]	👍	👍	👍	👎	👎	👎	👍	UR	👍	👍	👎	👍	👎
(F) CNTK [48]	👍	👍	👍	👎	👎	👎	👍	UR	👍	👎	👍	👎	👎
(F) Theano [4]	👍	👍	👍	👍	👍	👍	👎	👍	👎	👎	👎	👎	👎
(F) Chainer[MN] [44]	👍	👍	👎	👍	👎	👎	👍	👍	👍	👎	👍	👎	👎
(F) Darknet [38]	👍	👎	👍	👎	👎	👎	👎	👍	👎	👎	👎	👎	👎
(F) DL4j [43]	👍	👍	👍	👎	👎	👎	👍	UR	👍	👍	👍	👍	👎
(F) DSSTNE	👍	👎	👍	👎	👎	👎	👎	UR	👎	👎	👎	👎	👎
(F) PaddlePaddle	👍	👍	👍	👎	👎	👍	👍	UR	👍	👍	👍	👍	👍
(F) TVM [7]	👍	👍	👍	👎	👍	👍	👎	👎	👎	👎	👎	👎	👎
(E) Keras [8]	👍	👎	👎	👎	👎	👎	👎	UR	👍	👎	👎	👎	👎
(E) Horovod [42]	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👍	👍	👍
(E) TensorLayer [14]	👍	👍	👎	👎	👎	👎	👎	UR	👎	👍	👍	👎	👎
(E) Lasagne	👍	👍	👎	👎	👎	👎	👎	UR	👍	👎	👎	👎	👎
(E) TFLearn [11]	👍	👎	👎	👎	👎	👎	👍	👎	👎	👎	👎	👎	👎

- **Customizing operators relies on framework**
- **Network representation**
- **Dataset representation**
- **Training algorithm**
- **Distributed training (e.g., asynchronous SGD)**

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Operator interface

```
tf.layers.conv2d(
  inputs, filters,
  kernel_size, strides, padding,
  data_format, dilation_rate,
  activation, use_bias,
  kernel_initializer, bias_initializer,
  kernel_constraint, bias_constraint,
  kernel_regularizer,
  activity_regularizer,
  trainable, name, reuse
)
```

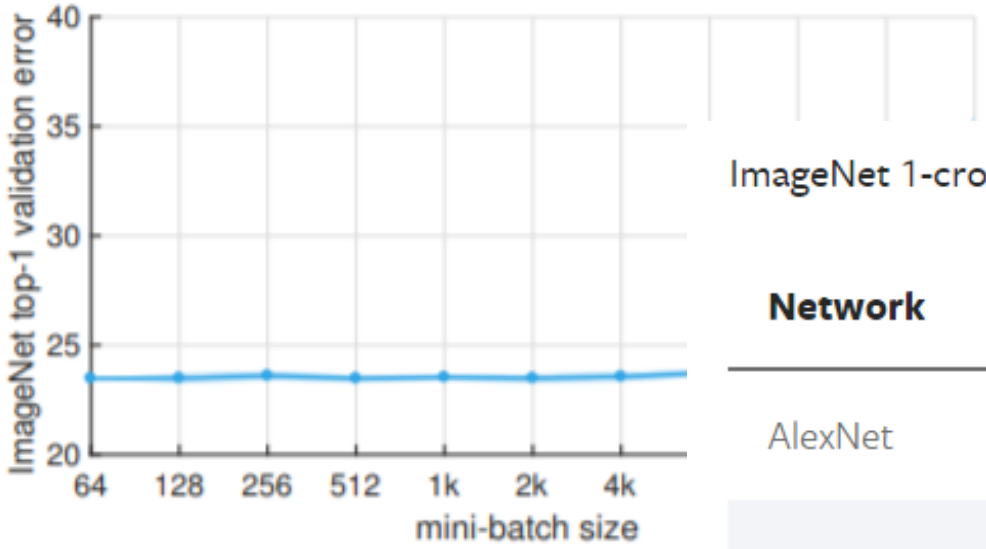
Listing 1: **TensorFlow: 19 parameters** to init 2D convolution.

```
cntk.layers.Convolution2D(
  filter_shape, num_filters,
  activation, init,
  pad, strides, bias, init_bias,
  reduction_rank, name
)
```

Listing 2: **CNTK: 10 parameters** to init 2D convolution.

- **Custom operators – even worse**
  - ATen
  - Tensors
  - Numpy
  - Eigen
  - …

Reproducibility – Networks / Distributed Training



kn	η	top-1 error (%)
256	0.1	23.60 ±0.12
512	0.2	23.48 ±0.09
1k	0.4	23.53 ±0.08
2k	0.8	23.49 ±0.11
4k	1.6	23.56 ±0.12
8k	3.2	23.74 ±0.09

https://pytorch.org/docs/stable/torchvision/models.html

ImageNet 1-crop error rates (224x224)

Network	Top-1 error
AlexNet	43.45
VGG-11	30.98
ResNet-50	23.85

https://github.com/tensorflow/models/tree/r1.11/official/resnet

Pre-trained model

You can download 190 MB pre-trained ImageNet validation set. Simply downl
`--model_dir` flag.

ResNet-50 v2 (Accuracy 76.05%):

- Checkpoint
- SavedModel

ResNet-50 v2 (fp16, Accuracy 75.56%):

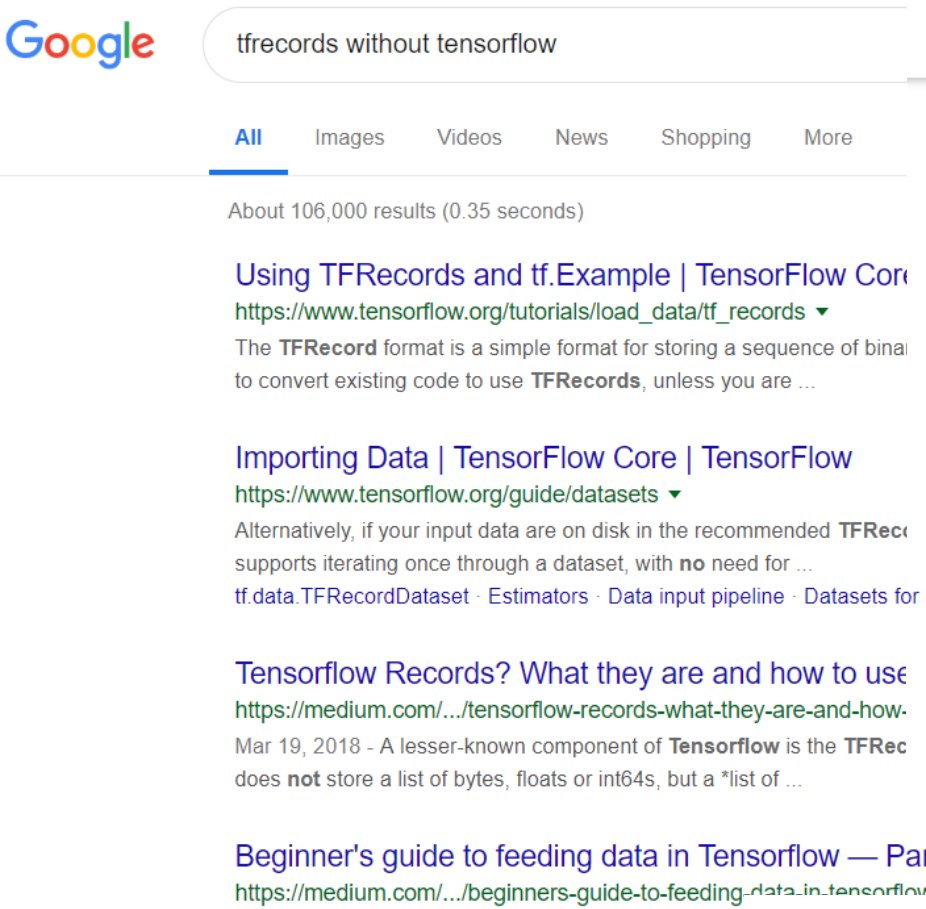
- Checkpoint
- SavedModel

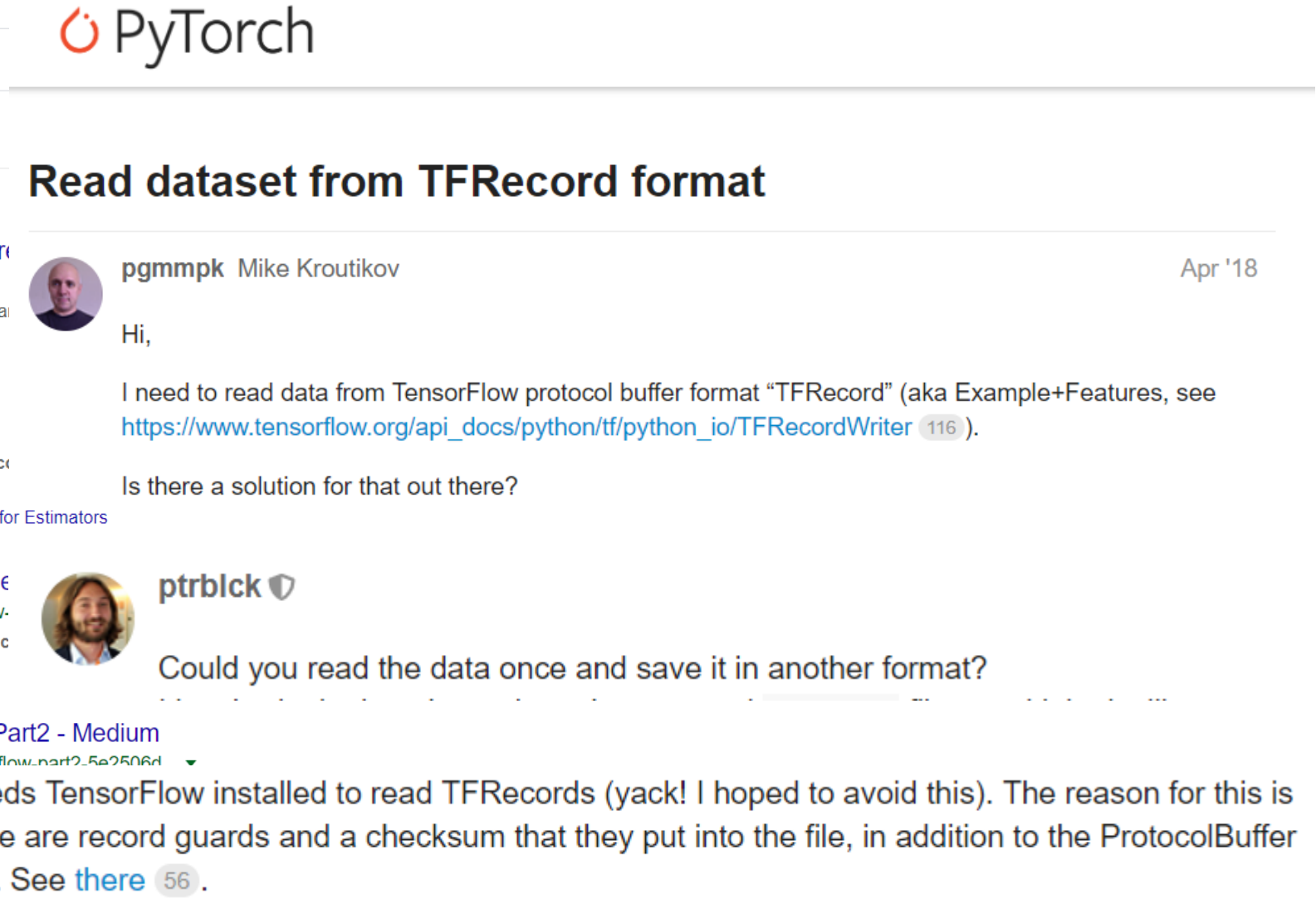
ResNet-50 v1 (Accuracy 75.91%):

24.09%

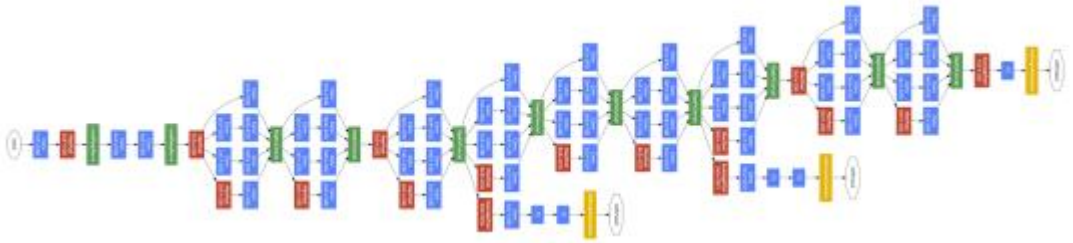
P. Goyal et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour” arXiv 2017.

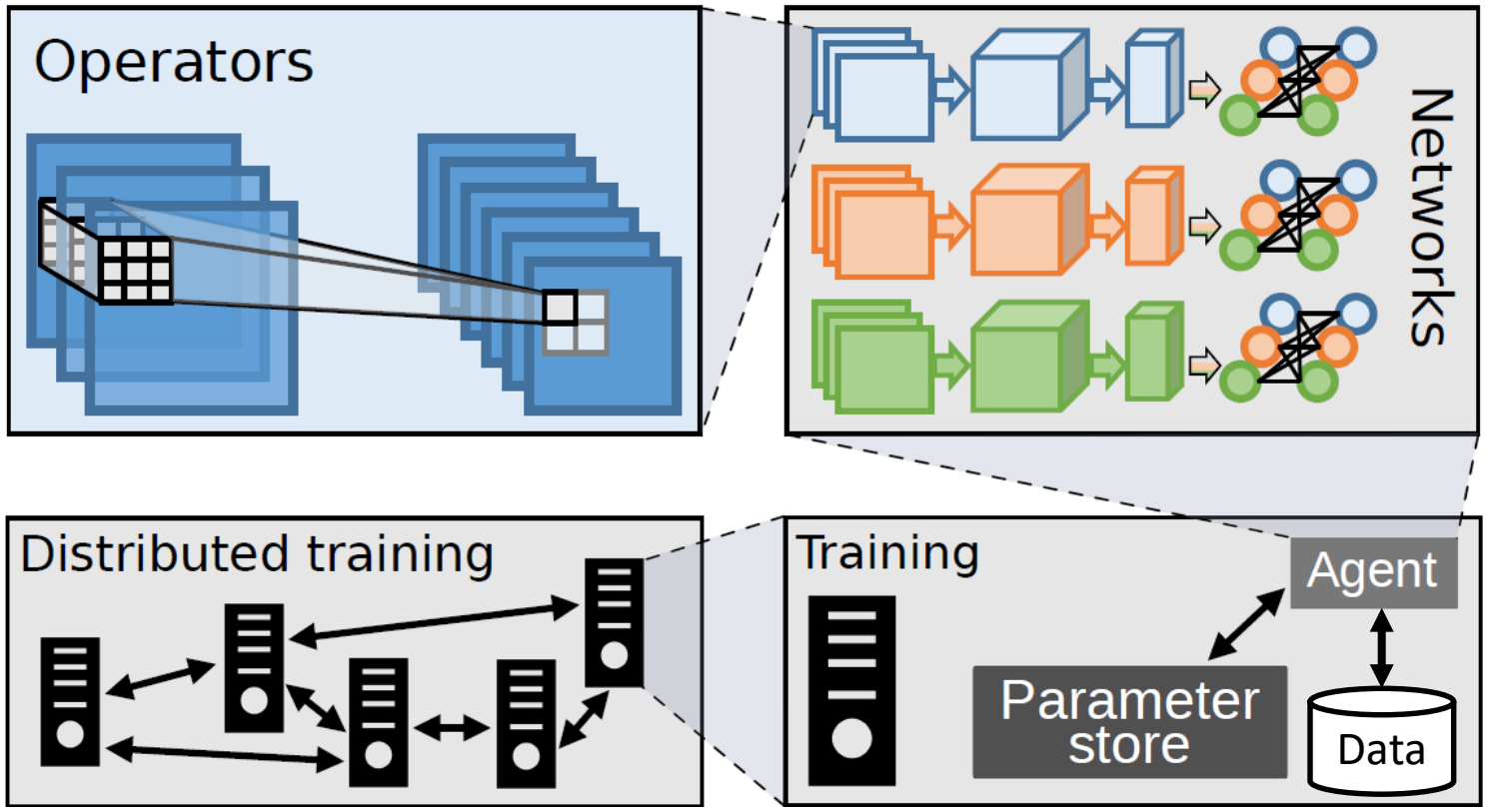
TFRecords format





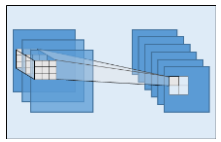
Computational Principles

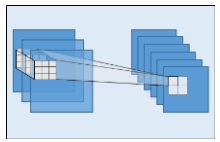




Level 0

```
forward()
backward()
```





CustomOp

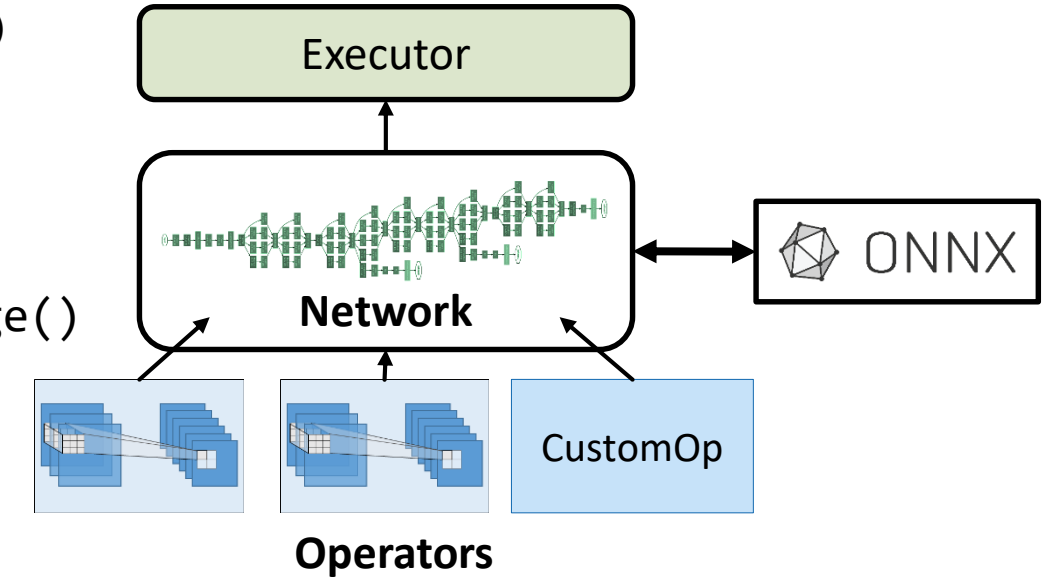
Operators

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

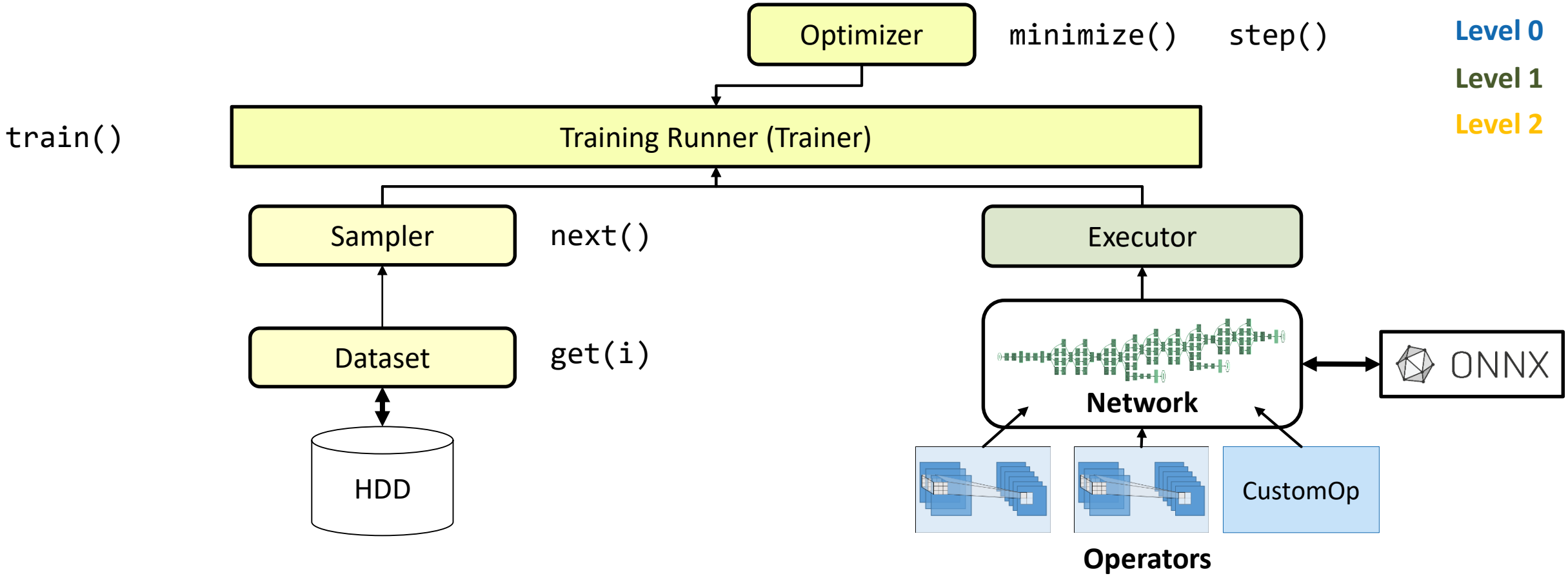
Level 0

Level 1

```
inference()
inference_and_backprop()
fetch/feed_tensor()

        add_node()
        add_edge()
        remove_node/edge()
```



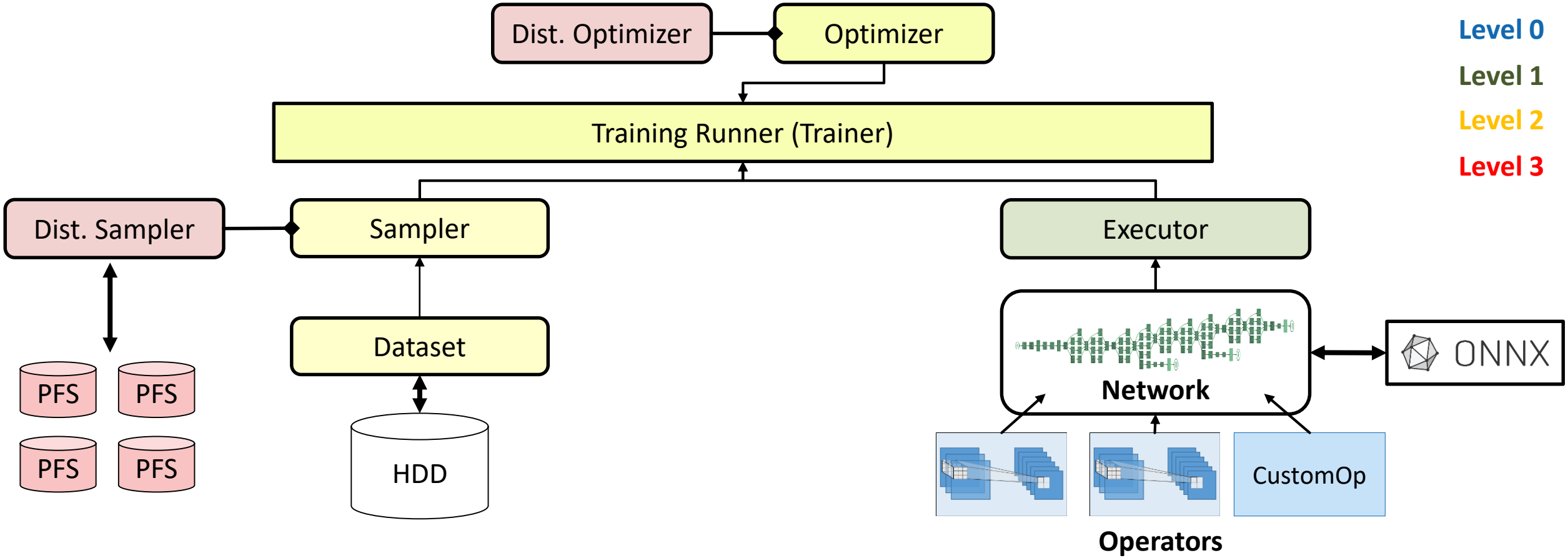
T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.



T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Deep500

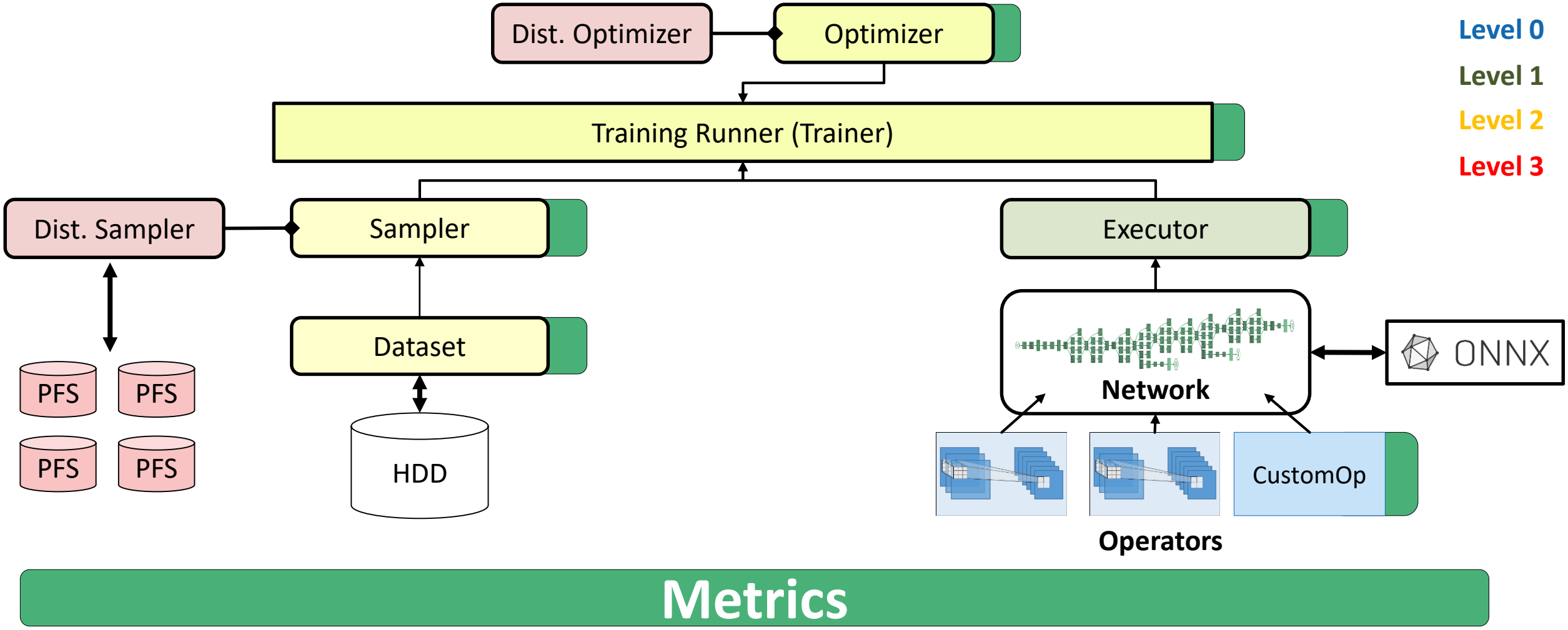
- **Deep learning meta-framework: a framework for frameworks to reside in**



T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Deep500

- **Deep learning meta-framework: a framework for frameworks to reside in**



T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Metrics

- **Latency**
  - Could be used for overhead
- **Throughput**
- **Correctness**
  - Norms
  - Automatic numerical gradient checker
- **Repeatability**
- **Communication volume**
- **Sampling bias**

Metric Examples

- **Norm:**

```
class NormDifference(TestMetric):
    def __init__(self, ord):
    """ Returns the difference between all tensors as a list of norms.
        @param ord Order of the norm. Abides by numpy's definition of norm
         order. See numpy.linalg.norm documentation.
    """
        self.ord = ord

    def measure(self, inputs, outputs, reference_outputs):
        if isinstance(outputs, (list, tuple)):
            assert len(outputs) == len(reference_outputs)
            return [np.linalg.norm((ro - o).flatten(), ord=self.ord) for o, ro
                        in zip(outputs, reference_outputs)]
        else:
            return np.linalg.norm((outputs - reference_outputs).flatten(),
                                  ord=self.ord)
```

- **Wallclock time:**

```
# ....
def begin(self, *args):
    if self._t % self._avg_over == 0:
        self._begintime.append(time.time())

def end(self, *args):
    self._t += 1
    if self._t % self._avg_over == 0:
        self._endtime.append(time.time())

def measure(self, inputs, outputs, reference_outputs):
    return [(e - b) / self._avg_over for b, e in zip(self._begintime, self._endtime)]

def measure_summary(self, inputs, outputs, reference_outputs):
    result = np.median(self.measure(inputs, outputs, reference_outputs))
    # ....
```

For Benchmarking: Recipes

Fixed definitions + mutable definitions + acceptable metric set = Recipe

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

For Benchmarking: Recipes

Fixed definitions + mutable definitions + acceptable metric set = Recipe

```python
""" A recipe for running the CIFAR-10 dataset with ResNet-44 and a momentum
    optimizer, with metrics for final test accuracy. """

import deep500 as d5
from recipes.recipe import run_recipe

# Using PyTorch as the framework
import deep500.frameworks.pytorch as d5fw


# Fixed Components
FIXED = {
    'model': 'resnet',
    'model_kwargs': dict(depth=44),
    'dataset': 'cifar10',
    'train_sampler': d5.ShuffleSampler,
    'epochs': 1
}


# Mutable Components
MUTABLE = {
    'batch_size': 64,
    'executor': d5fw.from_model,
    'executor_kwargs': dict(device=d5.GPUDevice()),
    'optimizer': d5fw.MomentumOptimizer,
    'optimizer_args': (0.1, 0.9),
}

# Acceptable Metrics
METRICS = [
    (d5.TestAccuracy(), 93.0)
]


if __name__ == '__main__':
    run_recipe(FIXED, MUTABLE, METRICS) or exit(1)
```

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

First (Stochastic) Steps

- **Install protobuf using apt:** `sudo apt install protobuf-compiler libprotoc-dev`
  or Anaconda (https://anaconda.org , `conda install protobuf`)

- **Get latest version of Deep500:** `pip install git+https://github.com/deep500/deep500.git`

- **Install an executor**
  - **PyTorch** (Anaconda): `conda install pytorch torchvision cudatoolkit=10.0 -c pytorch`
  - PyTorch (pip): Follow website https://pytorch.org/get-started/locally/

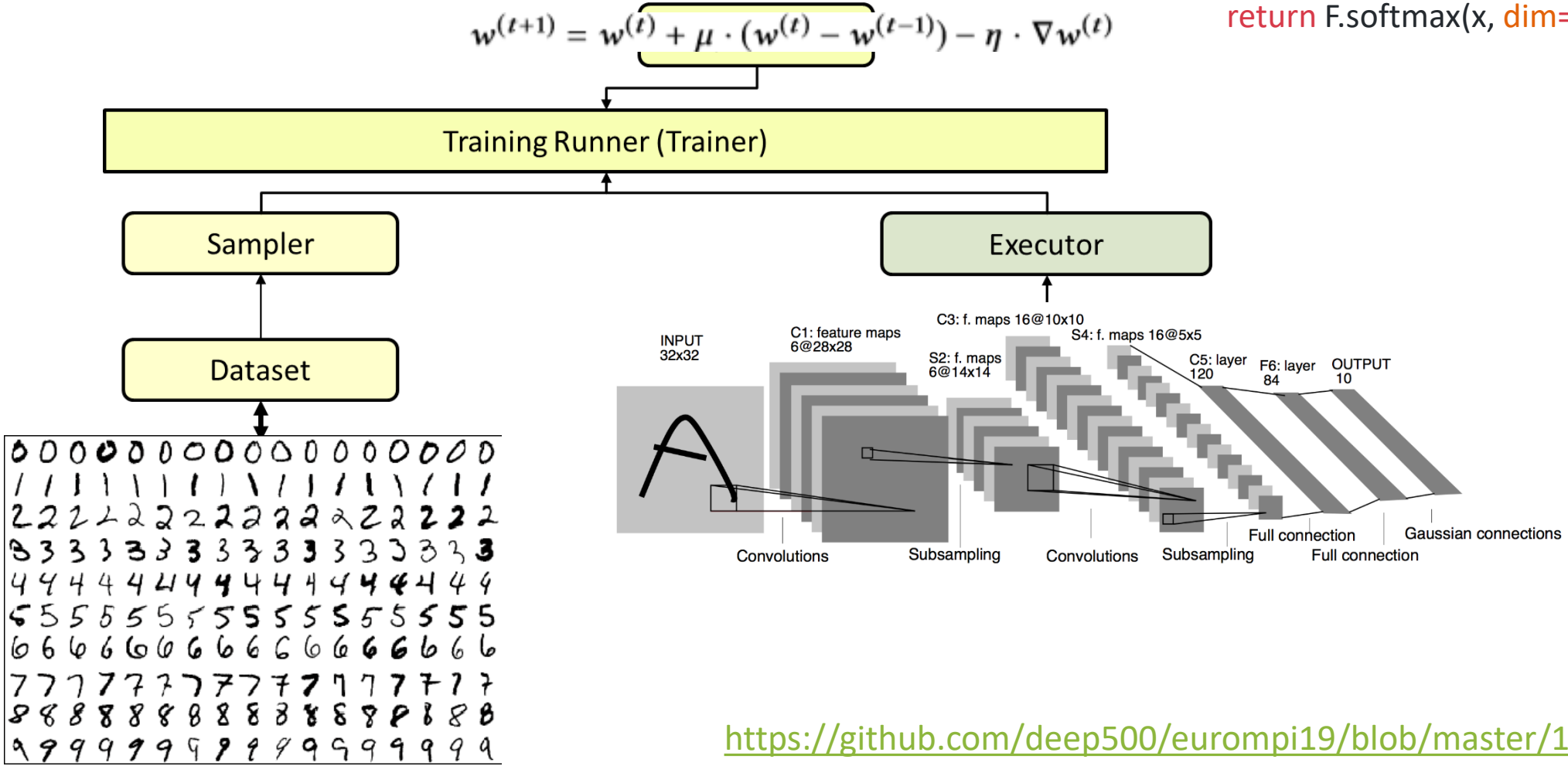
  - TensorFlow (GPU): `pip install tensorflow-gpu`
  - TensorFlow (CPU): `pip install tensorflow`

Notebook #1 – First (Stochastic) Steps

- **We will run the MNIST dataset with a simple CNN (LeNet)**

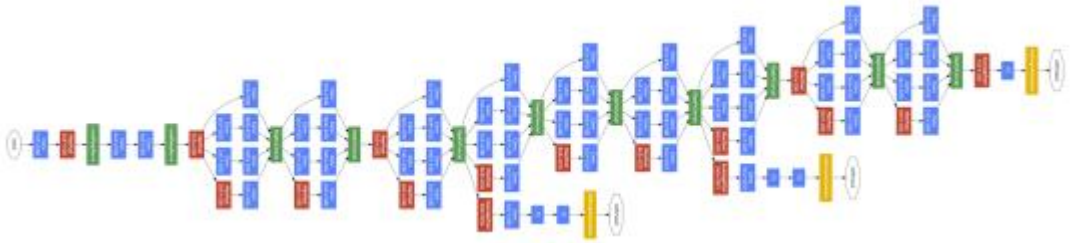
```
def forward(self, x):
  x = F.relu(F.max_pool2d(self.conv1(x), 2))
  x = F.relu(F.max_pool2d(self.conv2(x), 2))
  x = x.view(-1, self.imw*self.imw*20)
  x = F.relu(self.fc1(x))
  x = self.fc2(x)
  return F.softmax(x, dim=1)
```

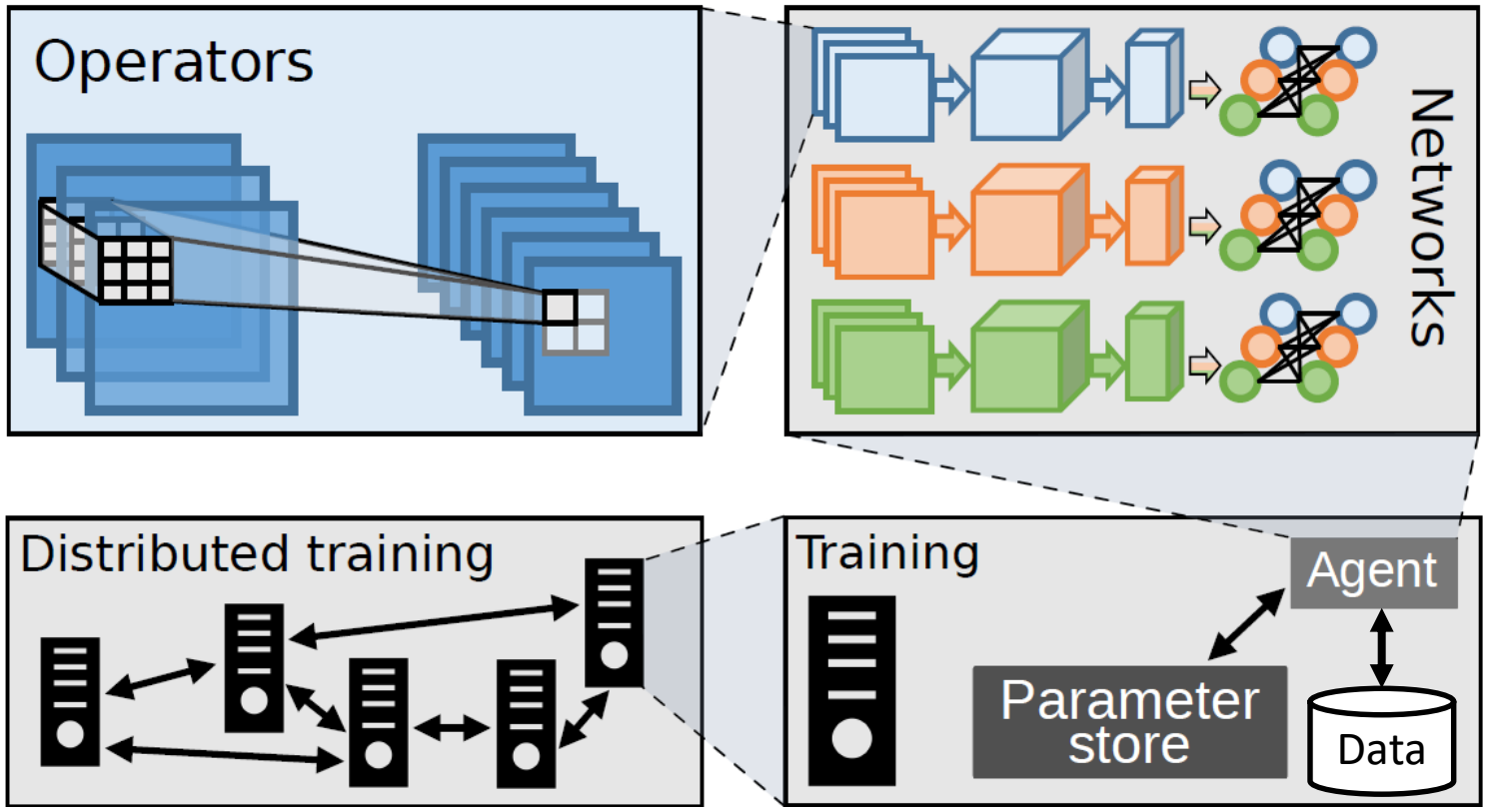
$$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$$



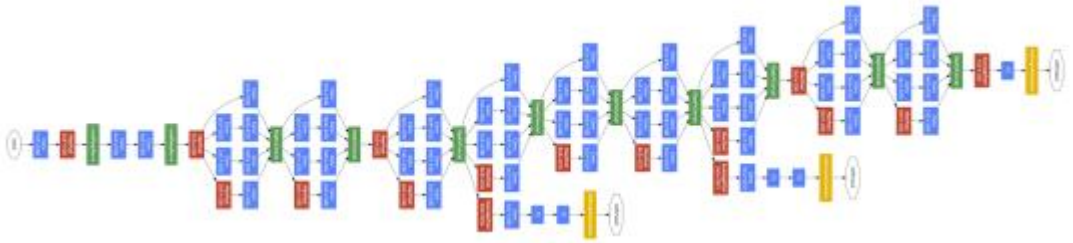
https://github.com/deep500/eurompi19/blob/master/1_mnist.ipynb

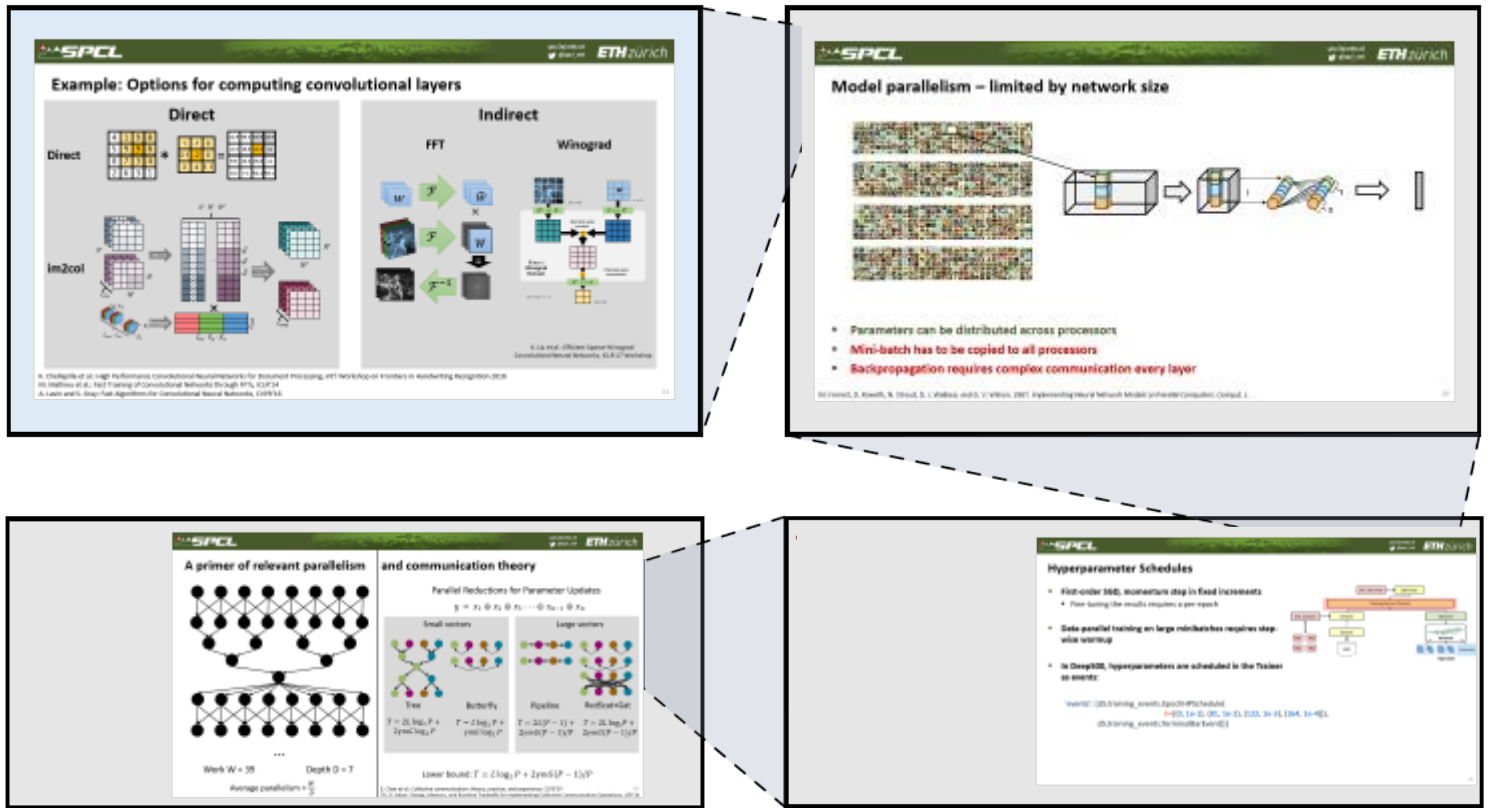
Computational Principles





Computational Principles

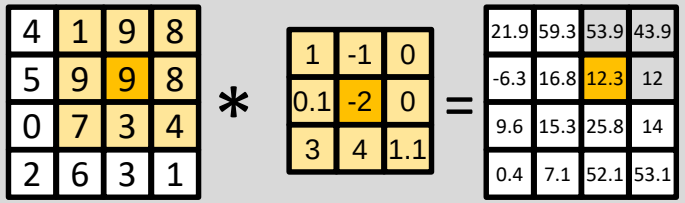




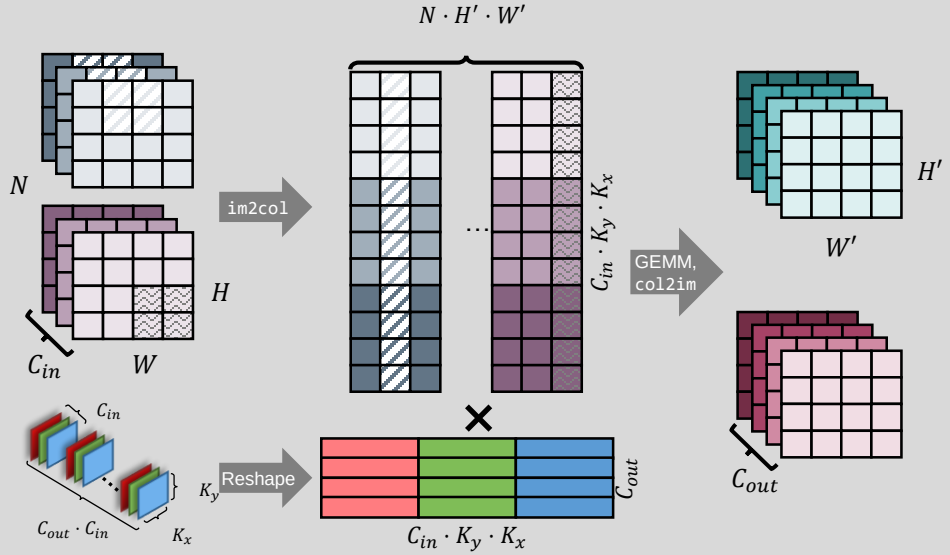
Example: Options for computing convolutional layers

Direct

Direct

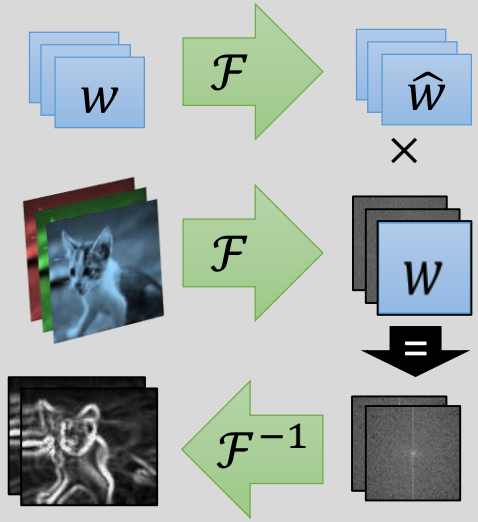


im2col

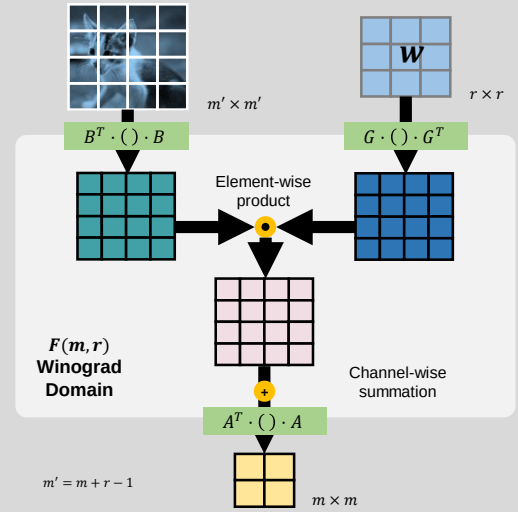


Indirect

FFT



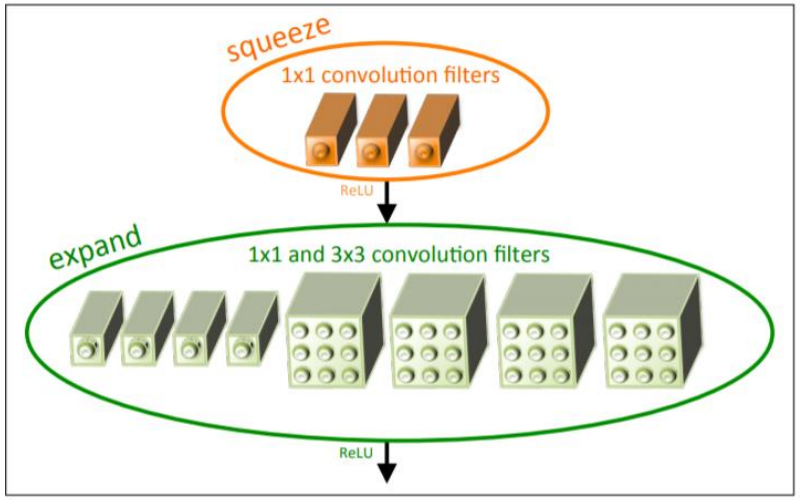
Winograd



X. Liu et al.: Efficient Sparse-Winograd Convolutional Neural Networks, ICLR'17 Workshop

K. Chellapilla et al.: High Performance Convolutional Neural Networks for Document Processing, Int'l Workshop on Frontiers in Handwriting Recognition 2016
M. Mathieu et al.: Fast Training of Convolutional Networks through FFTs, ICLR'14
A. Lavin and S. Gray: Fast Algorithms for Convolutional Neural Networks, CVPR'16

Operator Design



MobileNet 70.6%

3x3 Conv
BN
ReLU

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

1 ← N →

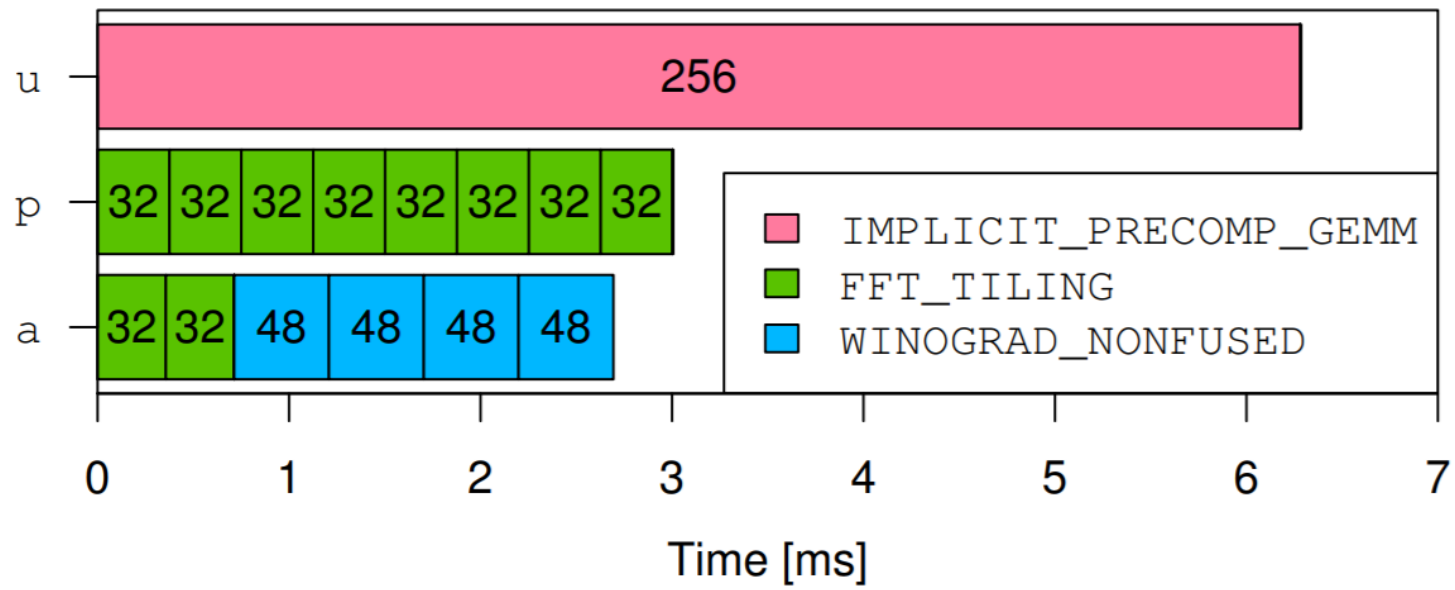
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

A.G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv 2017.
F.N. Iandola et al "Squeezenet: alexnet-level accuracy with 50x fewer parameters and <0.5MB model size," ICLR 2017.

Microbatching (µ-cuDNN) – how to implement layers best in practice?

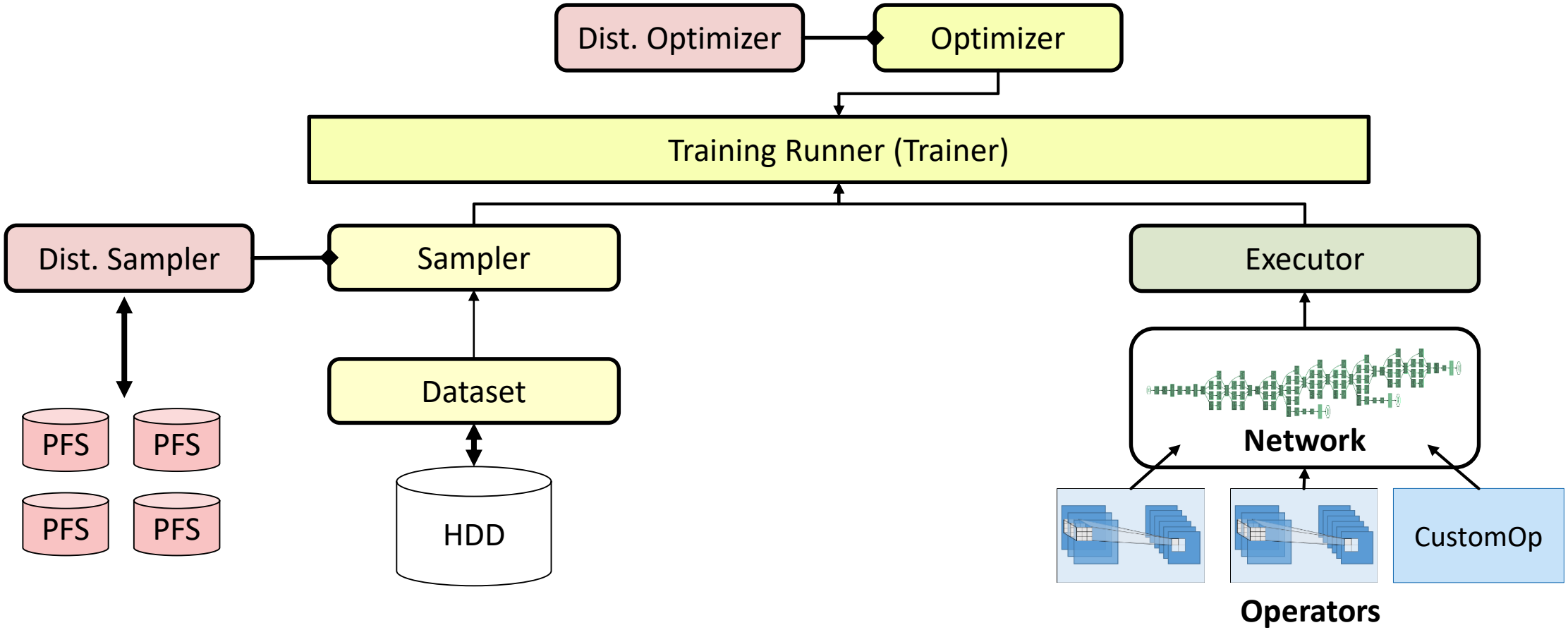
Microbatching Strategy

Fast (up to 4.54x faster on DeepBench)



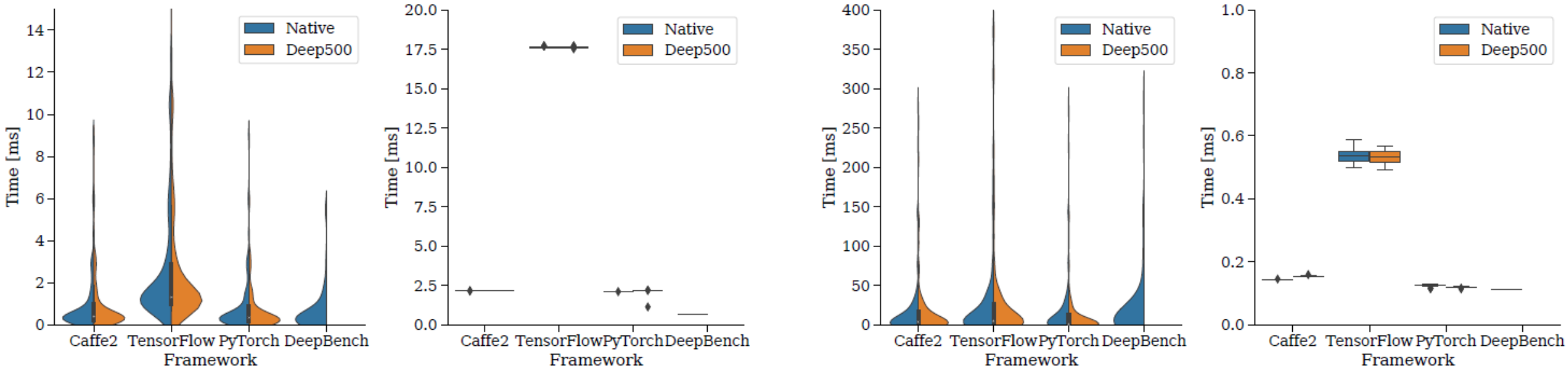
Y. Oyama, T. Ben-Nun, T. Hoefler and S. Matsuoka: µ-cuDNN: Accelerating Deep Learning Frameworks with Micro-Batching, Cluster 2018

Notebook #2 – No Recipes



https://github.com/deep500/eurompi19/blob/master/2_manual.ipynb

Use case: DeepBench



(a) Convolution Performance (left: violin plot of all kernels, right: box plot of size $N = 16$, $C = 3$, $H = W = 224$, filter size 3×3).

(b) Matrix Multiplication Performance (left: violin plot of all kernels, right: box plot of size $M = K = 2560$, $N = 64$).

Fig. 6: **Analysis of Deep500 Level 0:** Performance of operators implemented with Deep500 and selected frameworks, together with the DeepBench baseline.

Deep500: New Operator

```python
class IPowOp(CustomPythonOp):
    def __init__(self, power):
        super(IPowOp, self).__init__()
        self.power = power
        assert int(power) == power # integral

    def forward(self, inputs):
        return inputs[0] ** self.power

    def backward(self, grads, fwd_inputs, fwd_outputs):
        return (grads[0] * self.power *
           (fwd_inputs[0] ** (self.power - 1)))
```

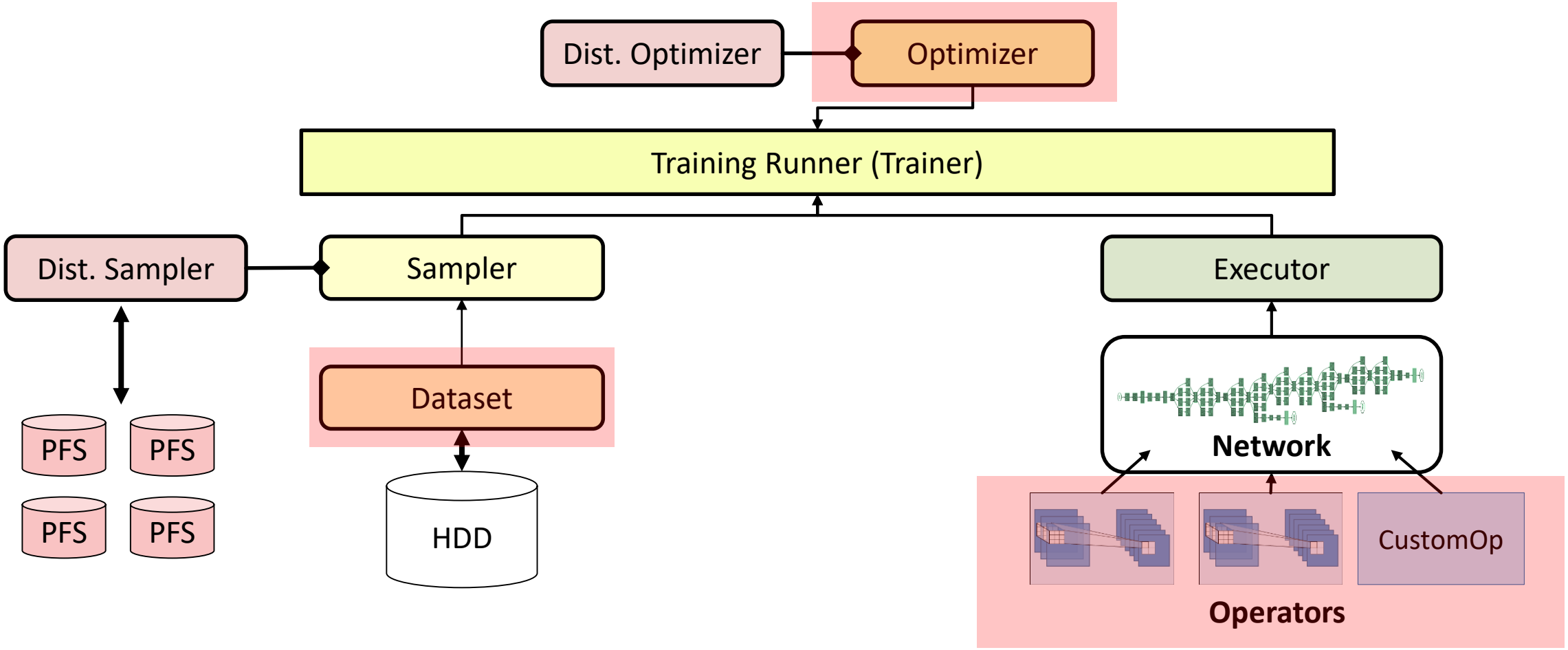
Python

```cpp
template<typename T>
class ipowop : public deep500::CustomOperator {
protected:
    int m_len;
public:
    ipowop(int len) : m_len(len) {}
    virtual ~ipowop() {}

    void forward(const T *input, T *output) {
        #pragma omp parallel for
        for (int i = 0; i < m_len; ++i)
            output[i] = std::pow(input[i], DPOWER);
    }

    void backward(const T *nextop_grad,
                  const T *fwd_input_tensor,
                  const T *fwd_output_tensor,
                  T *input_tensor_grad) {
        #pragma omp parallel for
        for (int i = 0; i < m_len; ++i) {
            input_tensor_grad[i] = nextop_grad[i] * DPOWER *
                std::pow(fwd_input_tensor[i], DPOWER - 1);
        }
    }
};
```

C++

T. Ben-Nun et al., A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning, IPDPS 2019.

Notebook #3 – Customizing Learning



https://github.com/deep500/eurompi19/blob/master/3_custom.ipynb

Deep500: New Optimizer

Algorithm 2 Acceler

Input: #Iterations
Set: $y_0 = z_0 = x_0$
for $t = 0 \dots T$ **do**
Set $\tau_t = 1/\alpha_t$
Update:

end for
Output: $\bar{y}_T \propto \sum_{t=}^{T}$

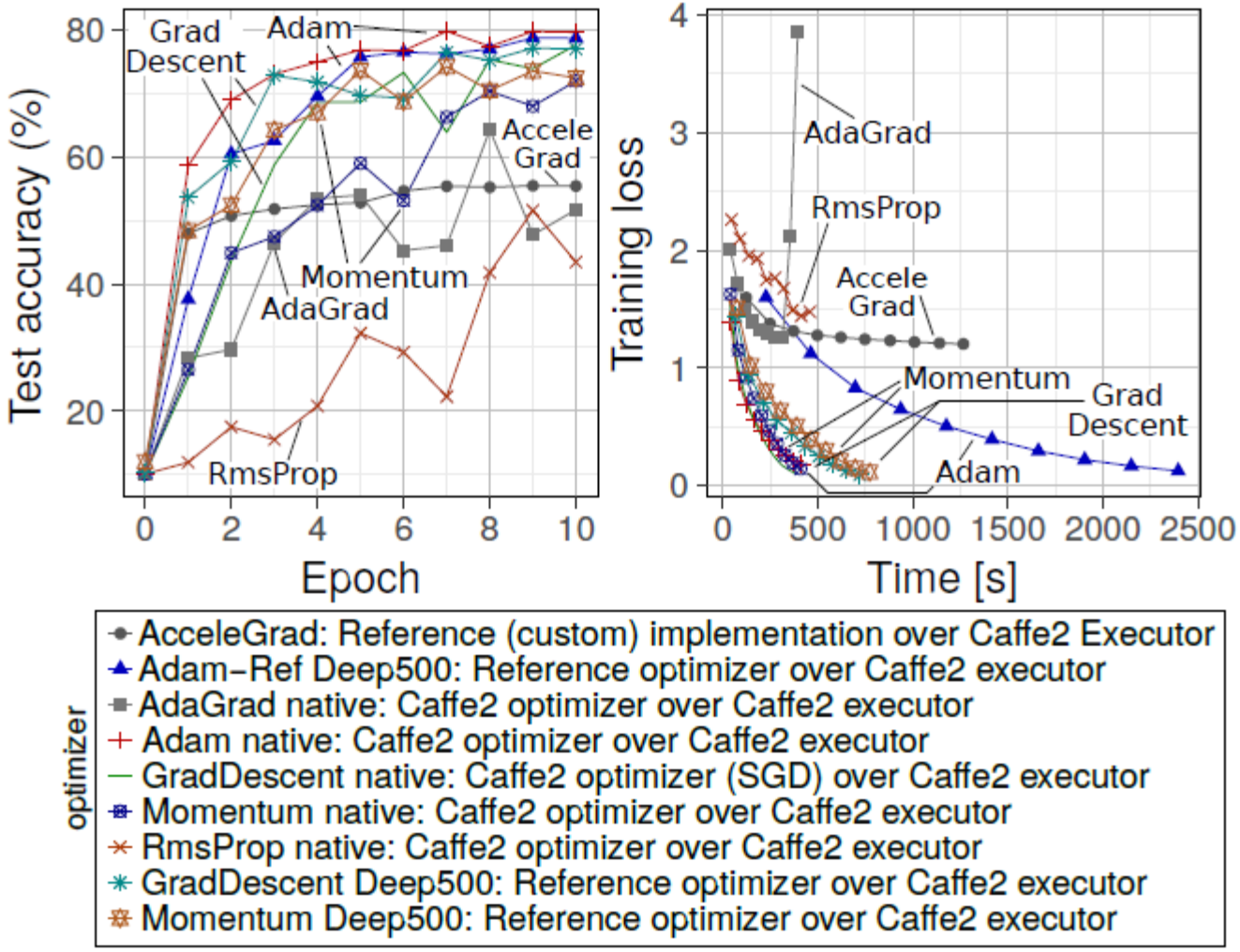
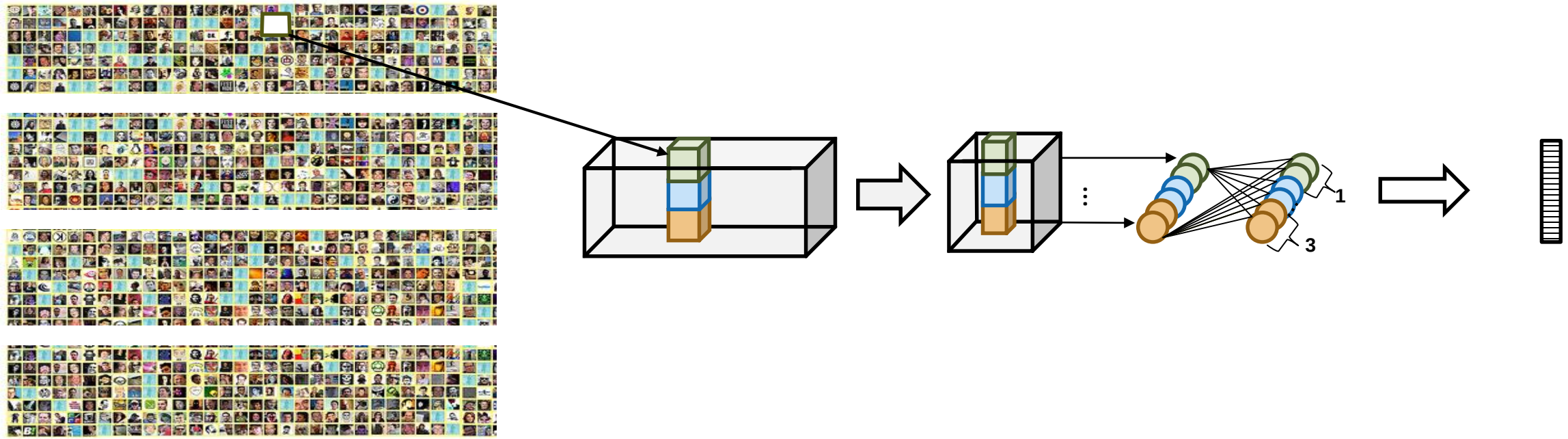


Fig. 9: The analysis of test accuracy vs. epoch number and training loss vs. elapsed time for different optimizers (assuming Caffe2, ResNet-18, CIFAR).

```
imizer):
                                    New input
else 1 / 4 * (self.t + 1)

                                Adjust parameters
ensors([param_name])[0]

f.tau_t) * y
am_name, new_param)

param_name):
e]
p.linalg.norm(grad) ** 2
* 2 + squared_grad)

rad

ad
p.sqrt(squared_grad))

                                    Update rule
```

K. Levy et al. "Online Adaptive Methods, Universality and Acceleration" NeurIPS 2018

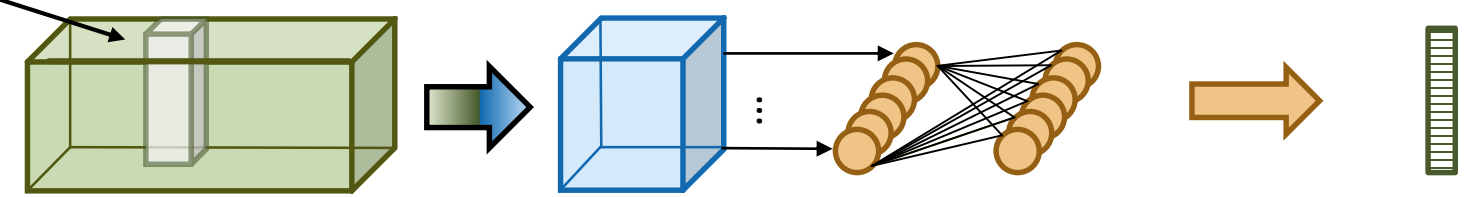
Model parallelism – limited by network size

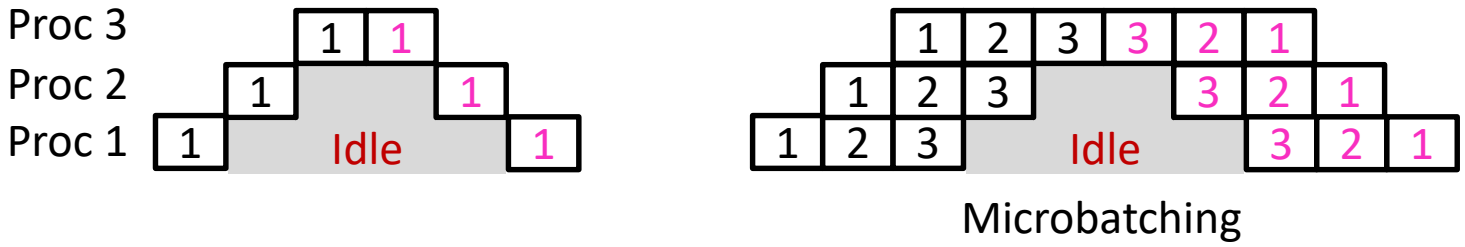


- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

M. Forrest, D. Roweth, N. Stroud, D. J. Wallace, and G. V. Wilson. 1987. Implementing Neural Network Models on Parallel Computers. Comput. J.

Pipeline parallelism – limited by network size

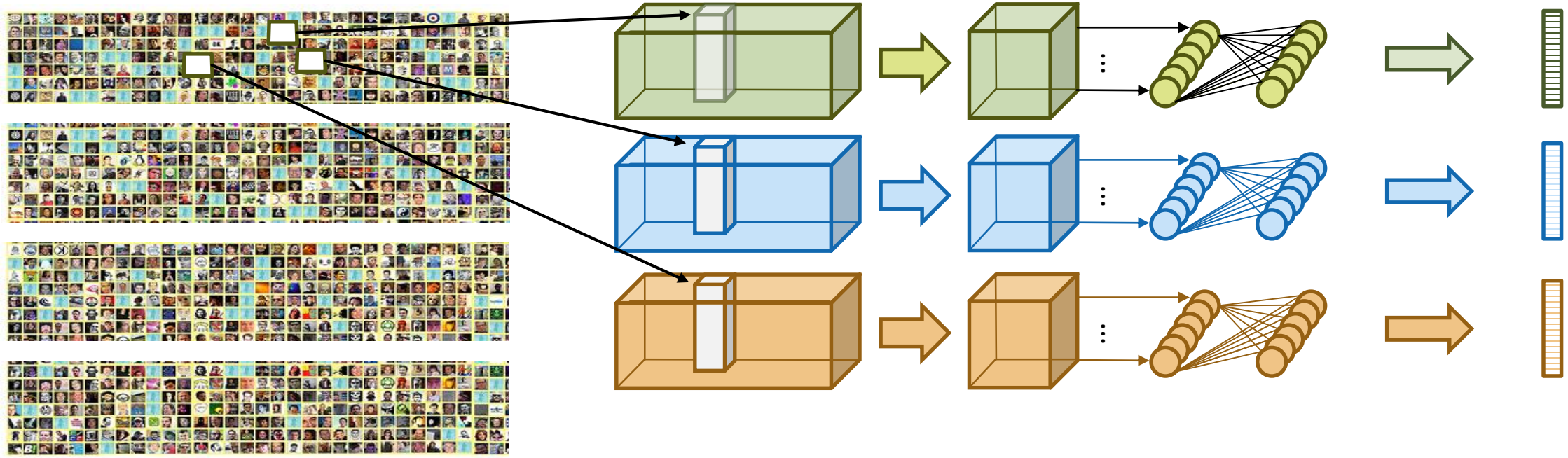




- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “Bubble”

G. Blelloch and C.R. Rosenberg: Network Learning on the Connection Machine, IJCAI’87

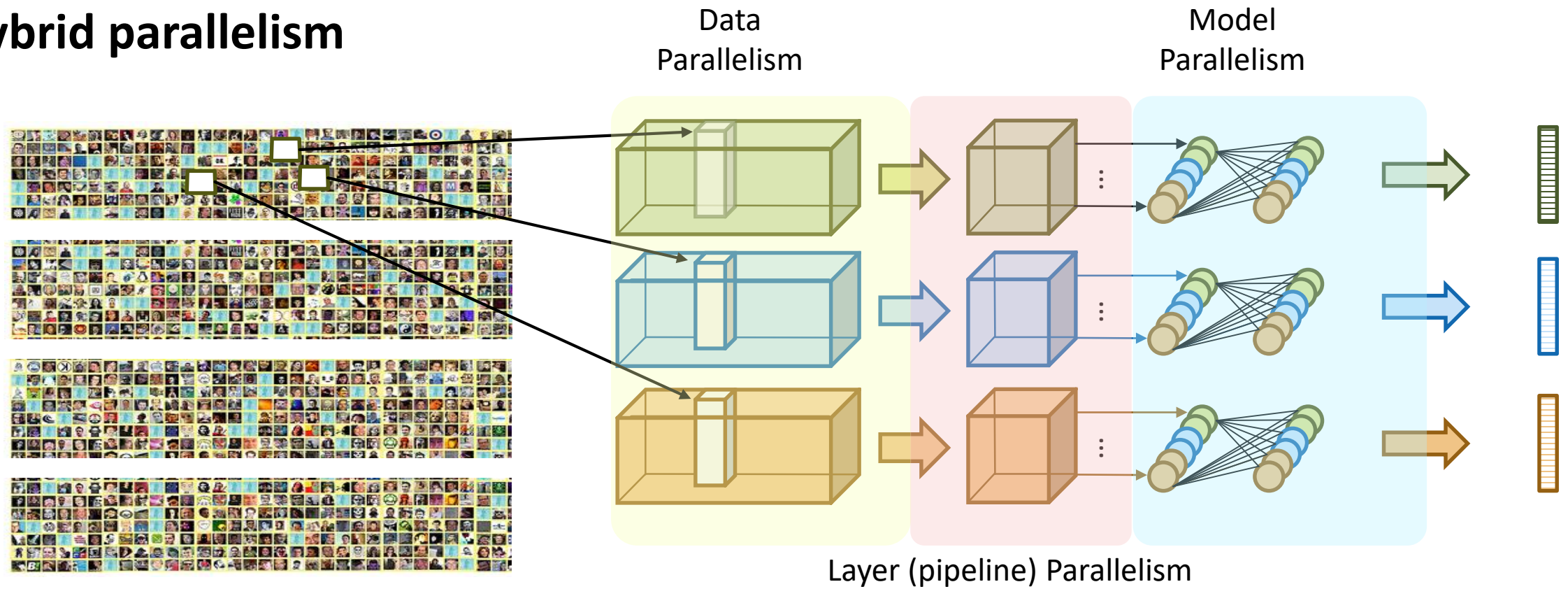
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

X. Zhang et al.: An Efficient Implementation of the Back-propagation Algorithm on the Connection Machine CM-2, NIPS'89

Hybrid parallelism

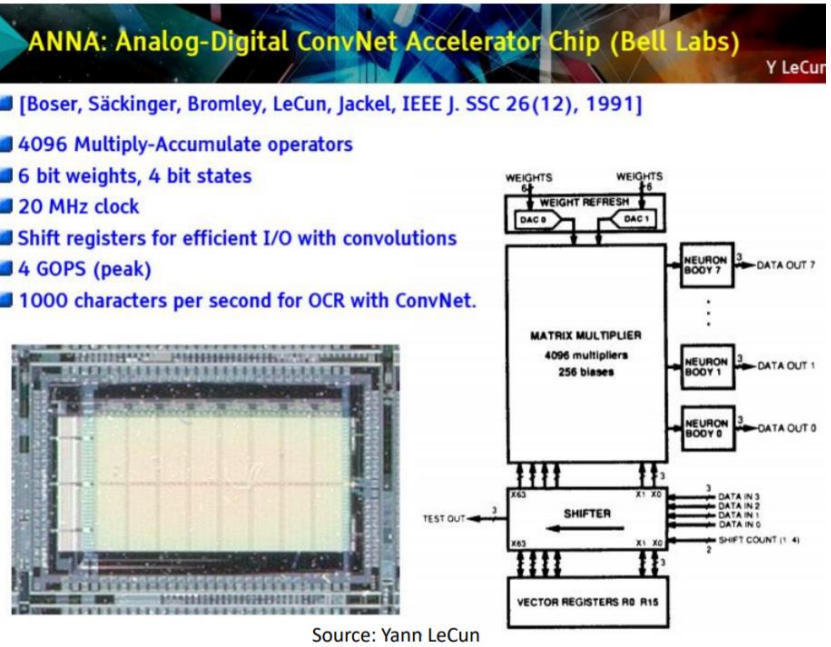


- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

A. Krizhevsky: One weird trick for parallelizing convolutional neural networks, arXiv 2014
J. Dean et al.: Large scale distributed deep networks, *NIPS*'12.
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

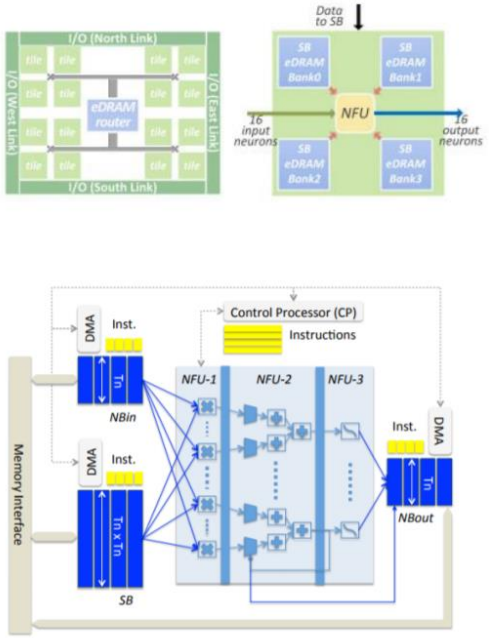
Agent

Specialized Hardware

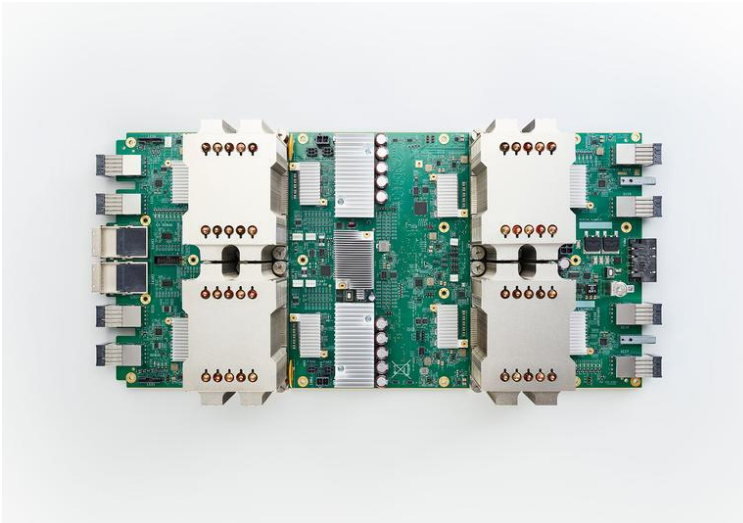


Source: Yann LeCun

ANNA [1991]



DianNao/Cambricon NPU [2014-today]



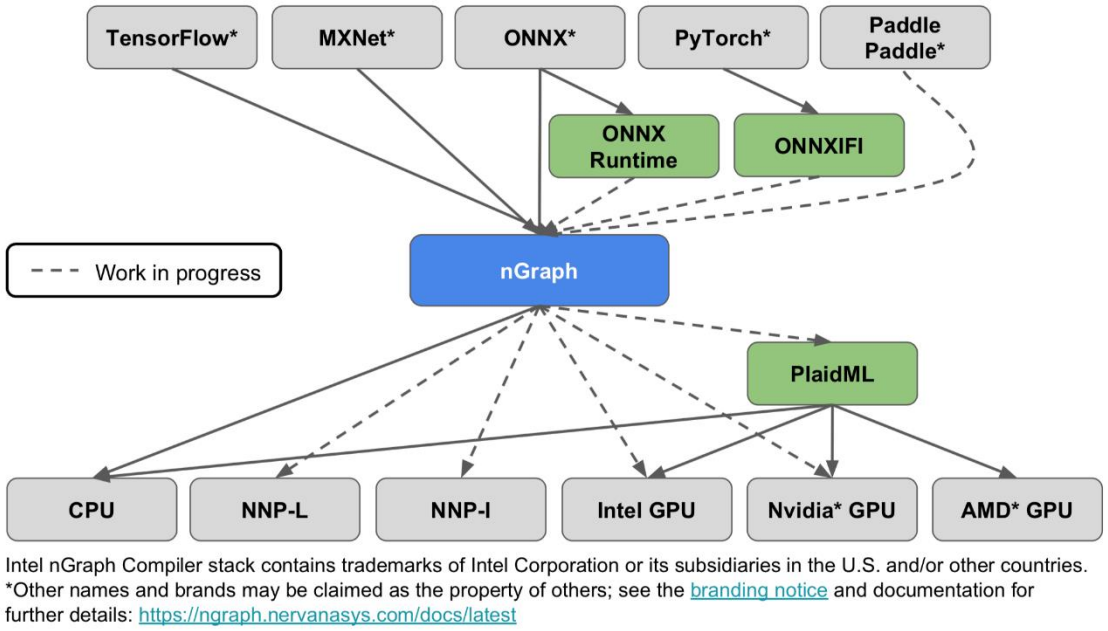
TPU v1(inference); v2-3(training) [2016-today]

EIE ESE NeuroCube YodaNN Minerva ...

Eyeriss Habana Goya PRIME BRein SCNN

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



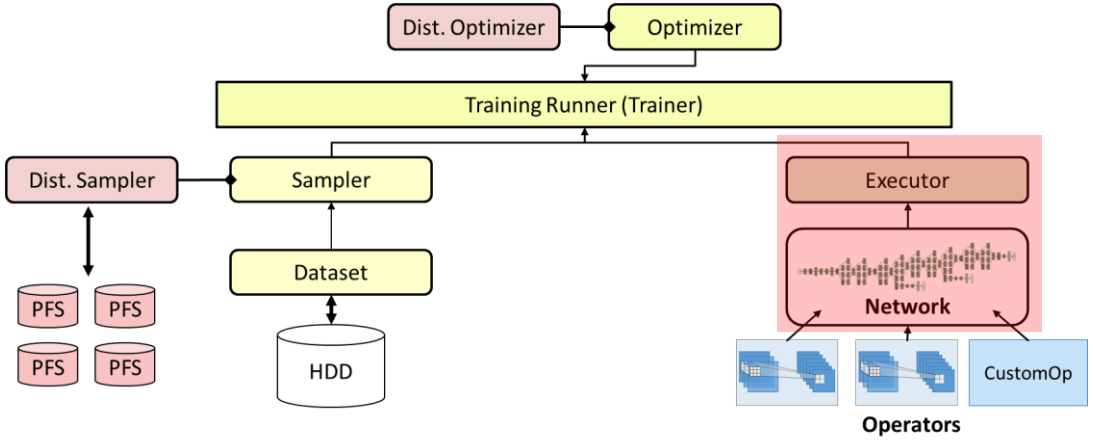
Intel nGraph Compiler stack contains trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others; see the branding notice and documentation for further details: https://ngraph.nervanasys.com/docs/latest

Intel nGraph

TVM Stack

DNN Compilers in Deep500

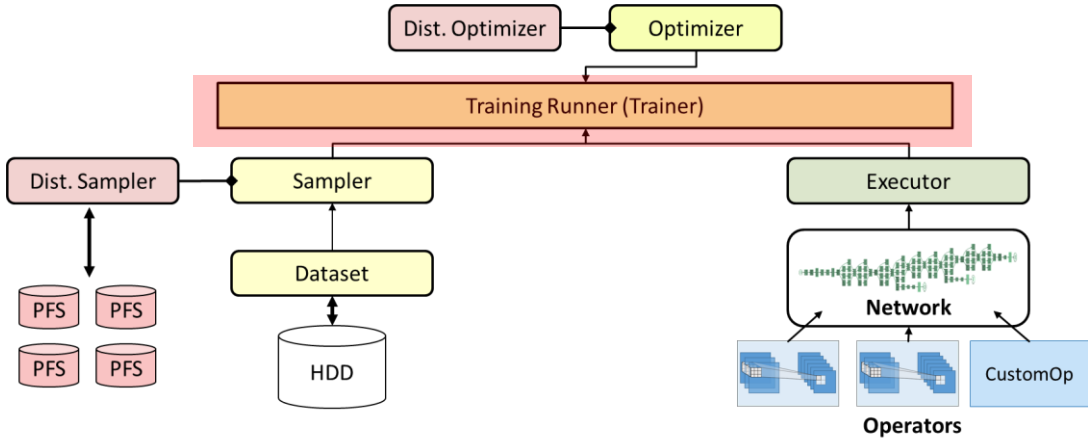
- **DNN compilers can be implemented as executors:**




```
class XLAGraphExecutor(d5.GraphExecutor):
  def __init__(self, model: d5.ops.OnnxModel, device: d5.DeviceType,
               events: List[d5.ExecutorEvent] = []):
    super().__init__(model, events)
    self.compiled_inference = xla.compile(…)
    self.compiled_training = xla.compile(…)
    # ...

  def inference(self, input: Dict[str, np.ndarray]) -> Dict[str, np.ndarray]:
    return self.compiled_inference(input)

  def inference_and_backprop(self, input: Dict[str, np.ndarray], y: str = 'loss'):
    return self.compiled_training(input)
```

Hyperparameter Schedules

- **First-order SGD, momentum step in fixed increments**
  - Fine-tuning the results requires a per-epoch
- **Data-parallel training on large minibatches requires step-wise warmup**
- **In Deep500, hyperparameters are scheduled in the Trainer as events:**




```
'events': [d5.training_events.EpochHPSchedule(
                                    lr=[(0, 1e-1), (81, 1e-2), (122, 1e-3), (164, 1e-4)]),
           d5.training_events.TerminalBarEvent()]
```

Data Augmentation

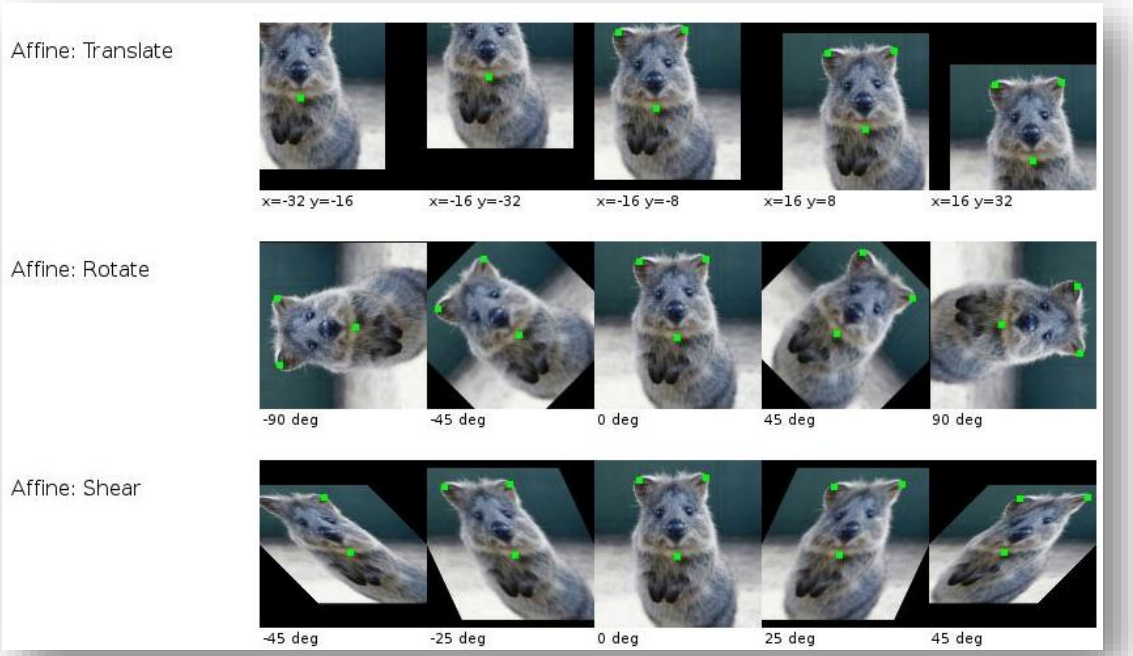
- **It is essential to augment the data during training**

- **Deep500 provides reference transformations**
  - Performed on the **Sampler**
  - Per-sample augmentations:

    Crop, Resize, Normalize, Random-Flip, Cutout
  - Batch Augmentation

- **Example:**

```
# Mutable Components
MUTABLE = {
    'batch_size': 64,
    'executor': d5fw.from_model,
    'executor_kwargs': dict(device=d5.GPUDevice()),
    'train_sampler': d5.ShuffleSampler,
    'train_sampler_kwargs': dict(transformations=[
        d5ref.Crop((32, 32), random_crop=True, padding=(4, 4)),
        d5ref.RandomFlip(),
        d5ref.Cutout(1, 16),
    ]),
```



Source: Google AutoAugment

DeVries, Terrance, and Graham W. Taylor. "Improved regularization of convolutional neural networks with cutout." arXiv preprint arXiv:1708.04552 (2017).
Hoffer, Elad, et al. "Augment your batch: better training with larger batches." arXiv preprint arXiv:1901.09335 (2019).

Hyperparameter and Architecture search

- **Meta-optimization of hyper-parameters (momentum) and DNN architecture**

Model	# Parameters	# Multiply-Adds	Top-1 / Top-5 Accuracy (%)
Incep-ResNet V2 [44]	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 [48]	83.6M	31.5B	80.9 / 95.6
PolyNet [51]	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131 [7]	79.5M	32.0B	81.5 / 95.8
GeNet-2 [47]*	156M	–	72.1 / 90.4
Block-QNN-B [52]*	–	–	75.7 / 92.6
Hierarchical [30]*	64M	–	79.7 / 94.8
NASNet-A [54]	88.9M	23.8B	82.7 / 96.2
PNASNet-5 [29]	86.1M	25.0B	82.9 / 96.2
AmoebaNet-A (N=6, F=190)*	86.7M	23.1B	82.8 / 96.1
AmoebaNet-A (N=6, F=448)*	469M	104B	83.9 / 96.6

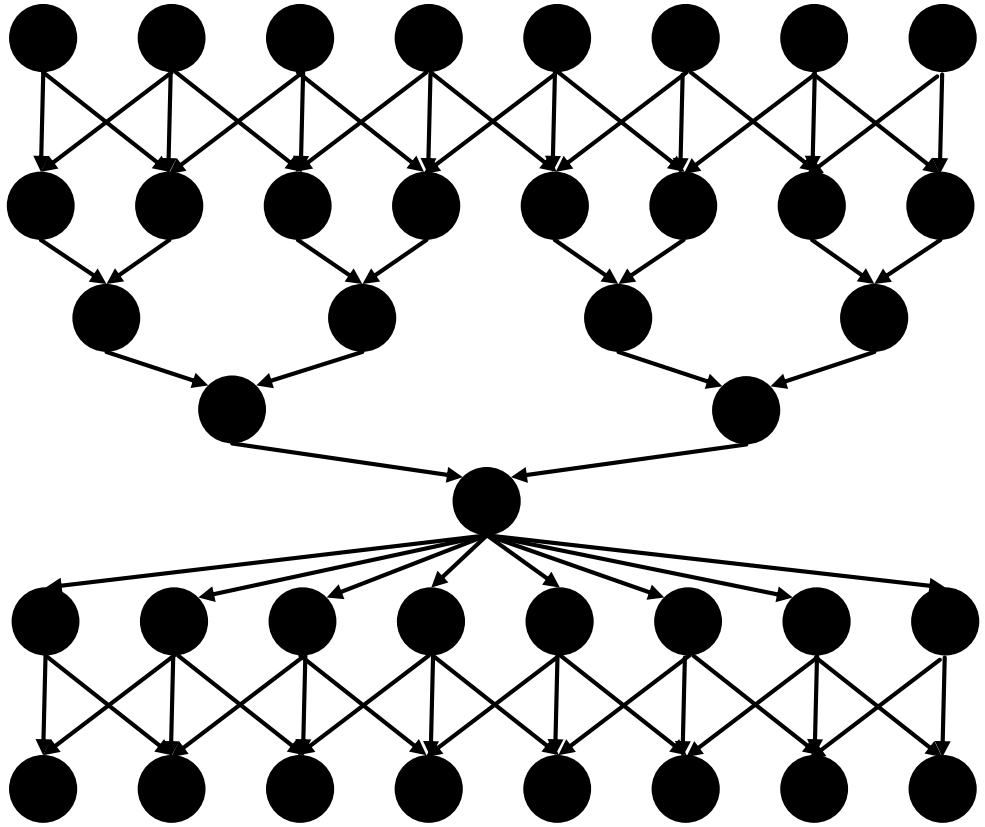
Model	Params	×+	1/5-Acc (%)
MobileNetV1	4.2M	575M	70.6 / 89.5
ShuffleNet (2x)	4.4M	524M	70.9 / 89.8
MobileNetV2 (1.4)	6.9M	585M	74.7 / –
NASNet-A (4, 44)	5.3M	564M	74.0 / 91.3
PNASNet-5 (3, 54)	5.1M	588M	74.2 / 91.9
AmoebaNet-C (4, 44)	**5.06M**	**535M**	**75.1 / 92.1**
AmoebaNet-A (4, 50)	5.08M	555M	74.5 / 92.0
AmoebaNet-A (4, 56)	6.34M	585M	75.5 / 92.5

Reinforcement Learning [1]

Evolutionary Algorithms [4]

[1] M. Jaderberg et al.: Population Based Training of Neural Networks, arXiv 2017
[2] E. Real et al.: Regularized Evolution for Image Classifier Architecture Search, arXiv 2018
[3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO'17
[4] H. Liu et al.: Hierarchical Representations for Efficient Architecture Search, ICLR'18

A primer of relevant parallelism and communication theory



…

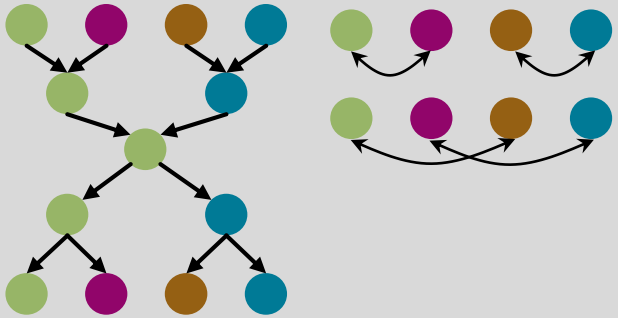
Work W = 39 Depth D = 7

Average parallelism = $\frac{W}{D}$

Parallel Reductions for Parameter Updates

$$y = x_1 \oplus x_2 \oplus x_3 \cdots \oplus x_{n-1} \oplus x_n$$

Small vectors



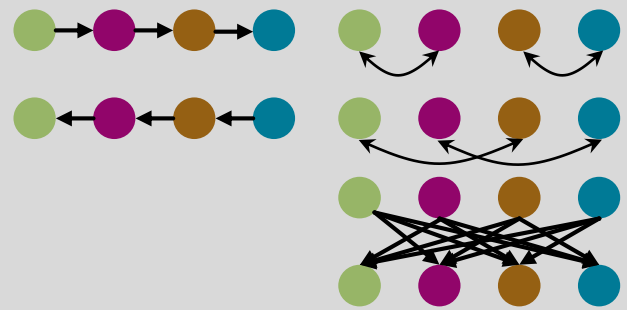
Tree

$T = 2L \log_2 P + 2\gamma mG \log_2 P$

Butterfly

$T = L \log_2 P + \gamma mG \log_2 P$

Large vectors



Pipeline

$T = 2L(P-1) + 2\gamma mG(P-1)/P$

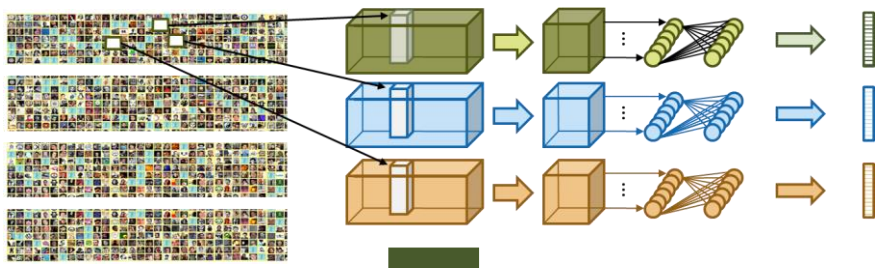
RedScat+Gat

$T = 2L \log_2 P + 2\gamma mG(P-1)/P$

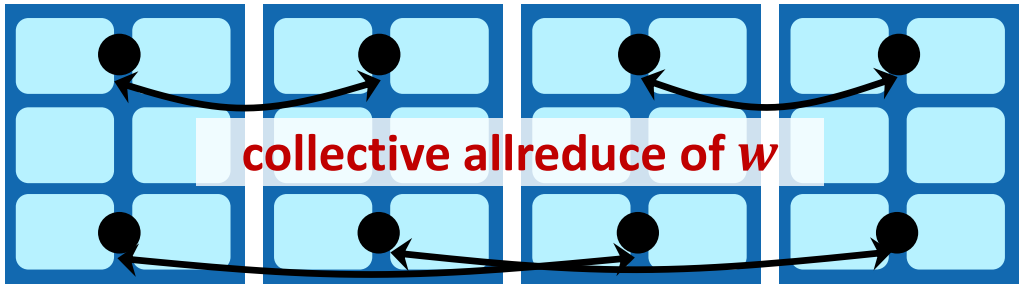
Lower bound: $T \geq L \log_2 P + 2\gamma mG(P-1)/P$

E. Chan et al.: Collective communication: theory, practice, and experience. CCPE'07
TH, D. Moor: Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations, JSFI'14

Updating parameters in distributed data parallelism



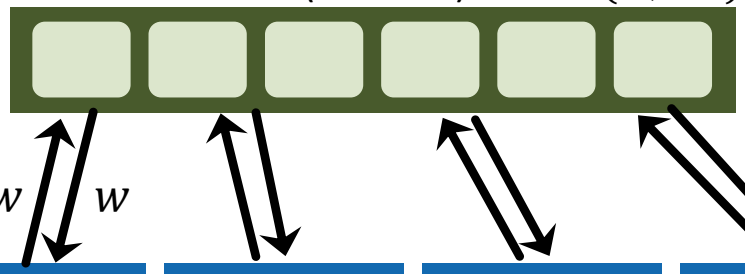
Decentral



Training Agent Training Agent Training Agent Training Agent

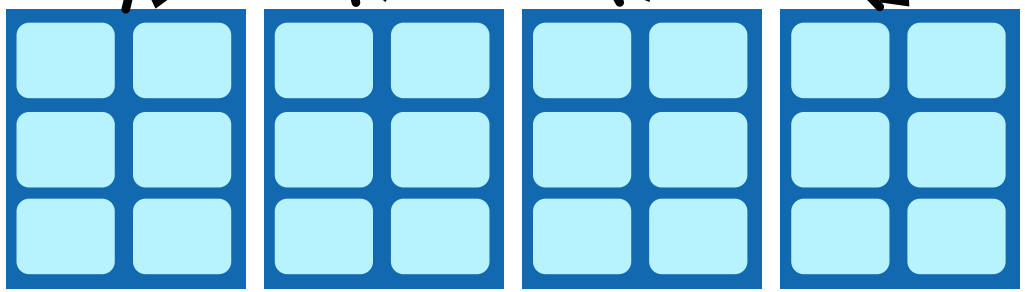
Central

parameter server (sharded) $w' = u(w, \nabla w)$



$T = 2L + 2P\,\gamma m/s\,G$

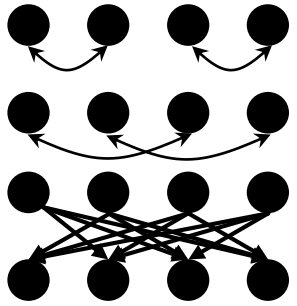
∇w w



Training Agent Training Agent Training Agent Training Agent

MPI

- Collective operations
- Topologies
- Neighborhood collectives
- RMA?



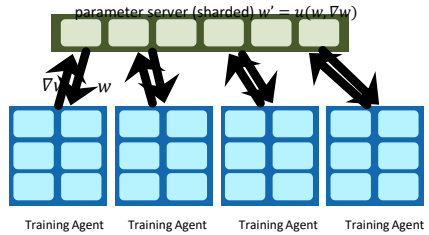
$T = 2L\log_2 P + 2\gamma mG(P-1)/P$

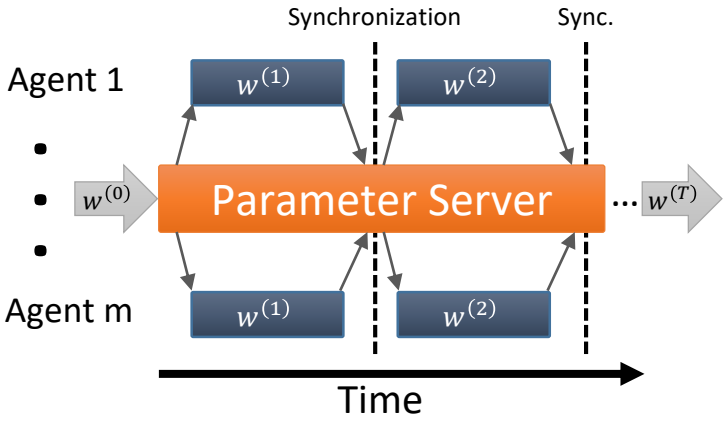
Hierarchical Parameter Server
S. Gupta et al.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. ICDM'16

Adaptive Minibatch Size
S. L. Smith et al.: Don't Decay the Learning Rate, Increase the Batch Size, arXiv 2017

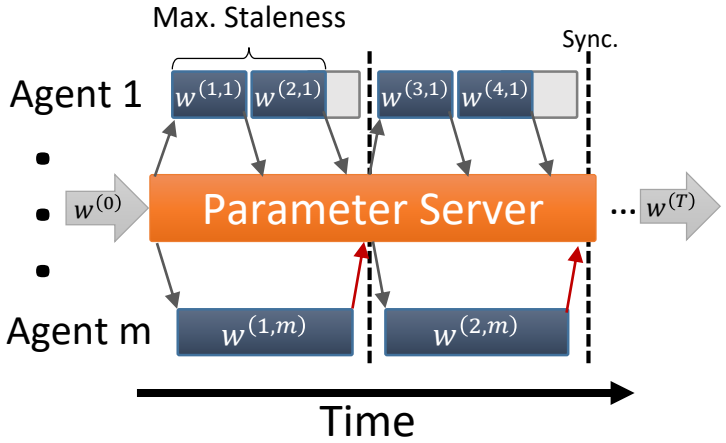
Parameter (and Model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**

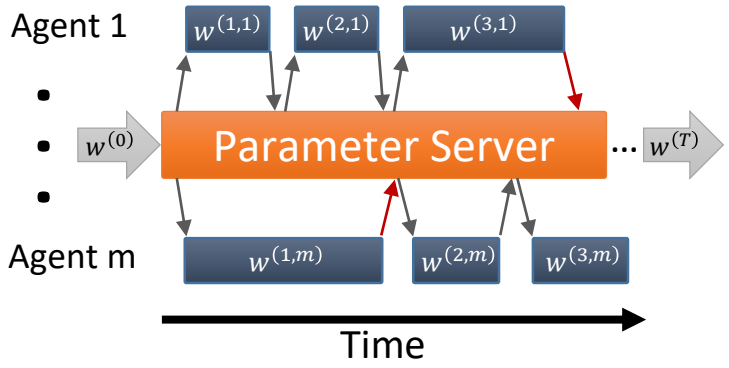




Synchronous

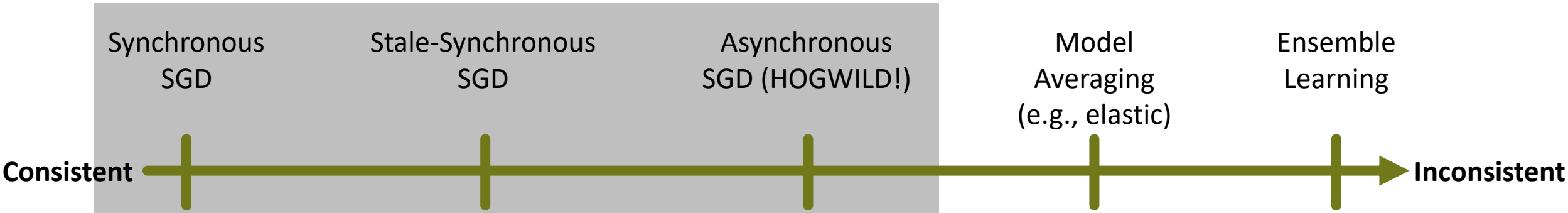


Stale Synchronous / Bounded Asynchronous



Asynchronous

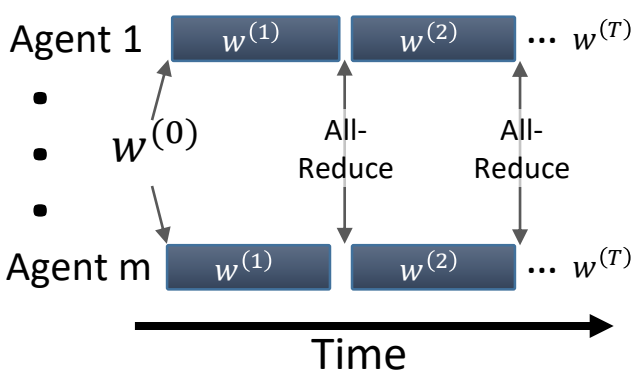
- **Trades off "statistical performance" for "hardware performance"**



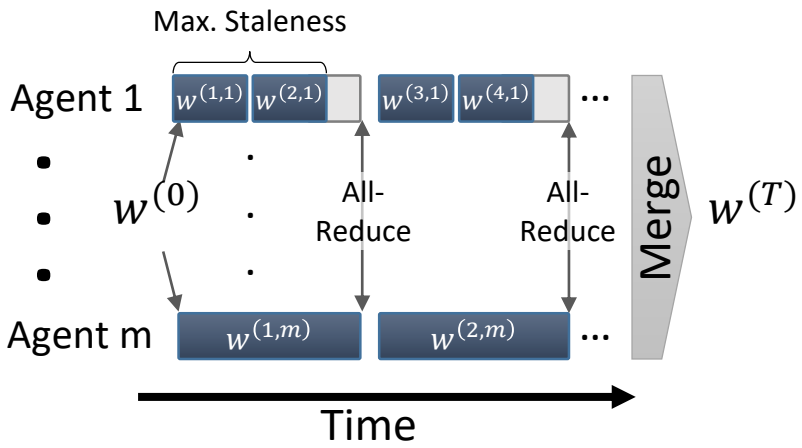
J. Dean et al.: Large scale distributed deep networks, *NIPS*'12.
F. Niu et al.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent, *NIPS*'11.

Parameter (and Model) consistency - decentralized

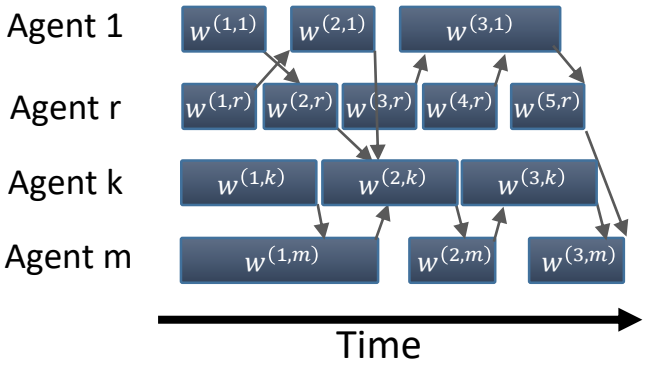
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous

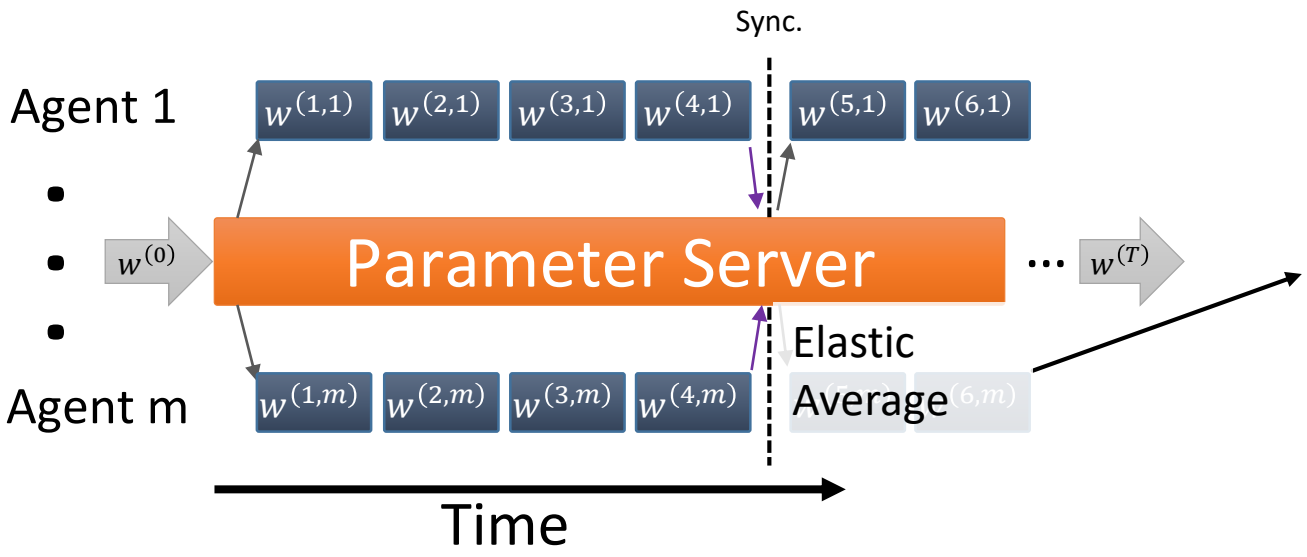


Asynchronous

- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Peter H. Jin et al., "How to scale distributed deep learning?", NIPS MLSystems 2016

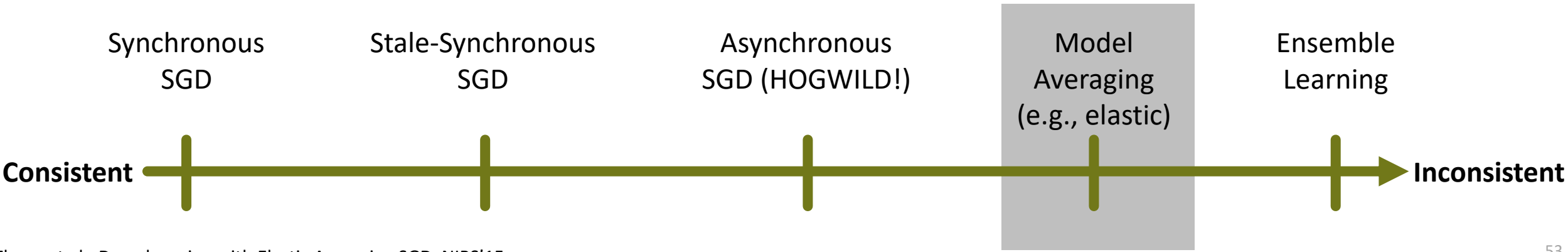
Parameter consistency in deep learning



Using physical forces between different versions of w:

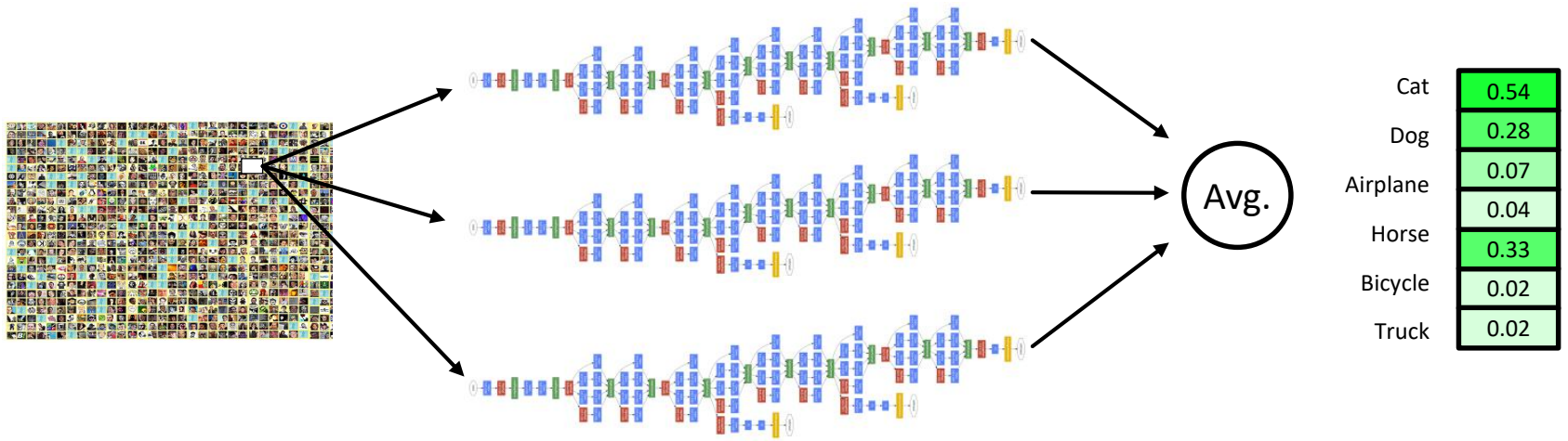
$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \tilde{w}_t\right)$$

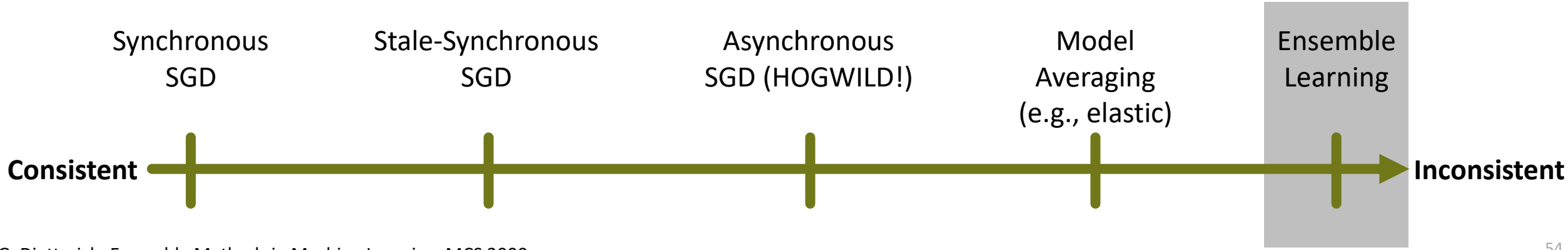
$$\tilde{w}_{t+1} = (1-\beta)\tilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



S. Zhang et al.: Deep learning with Elastic Averaging SGD, NIPS'15

Parameter consistency in deep learning

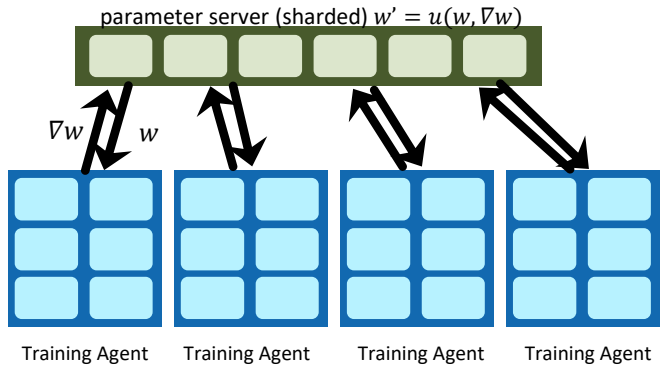


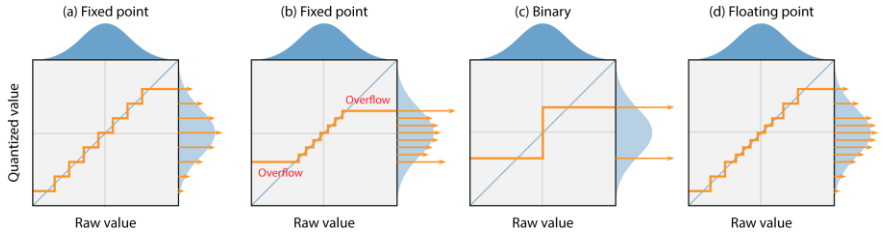


T. G. Dietterich: Ensemble Methods in Machine Learning, MCS 2000

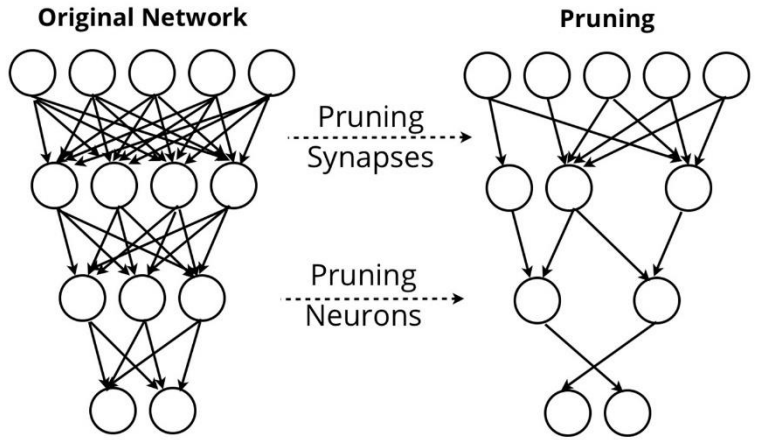
Communication optimizations

- **Different options how to optimize updates**
  - Send ∇w, receive w
  - Send FC factors (o_{l-1}, o_l), compute ∇w on parameter server
    Broadcast factors to not receive full w
  - Use lossy compression when sending, accumulate error locally!
- **Quantization**
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate
    Enables low precision (half, quarter) to become standard
  - TernGrad - ternary weights [2], 1-bit SGD [3], …
- **Sparsification**
  - Do not send small weight updates **or** only send top-k [4]
    Accumulate omitted gradients locally



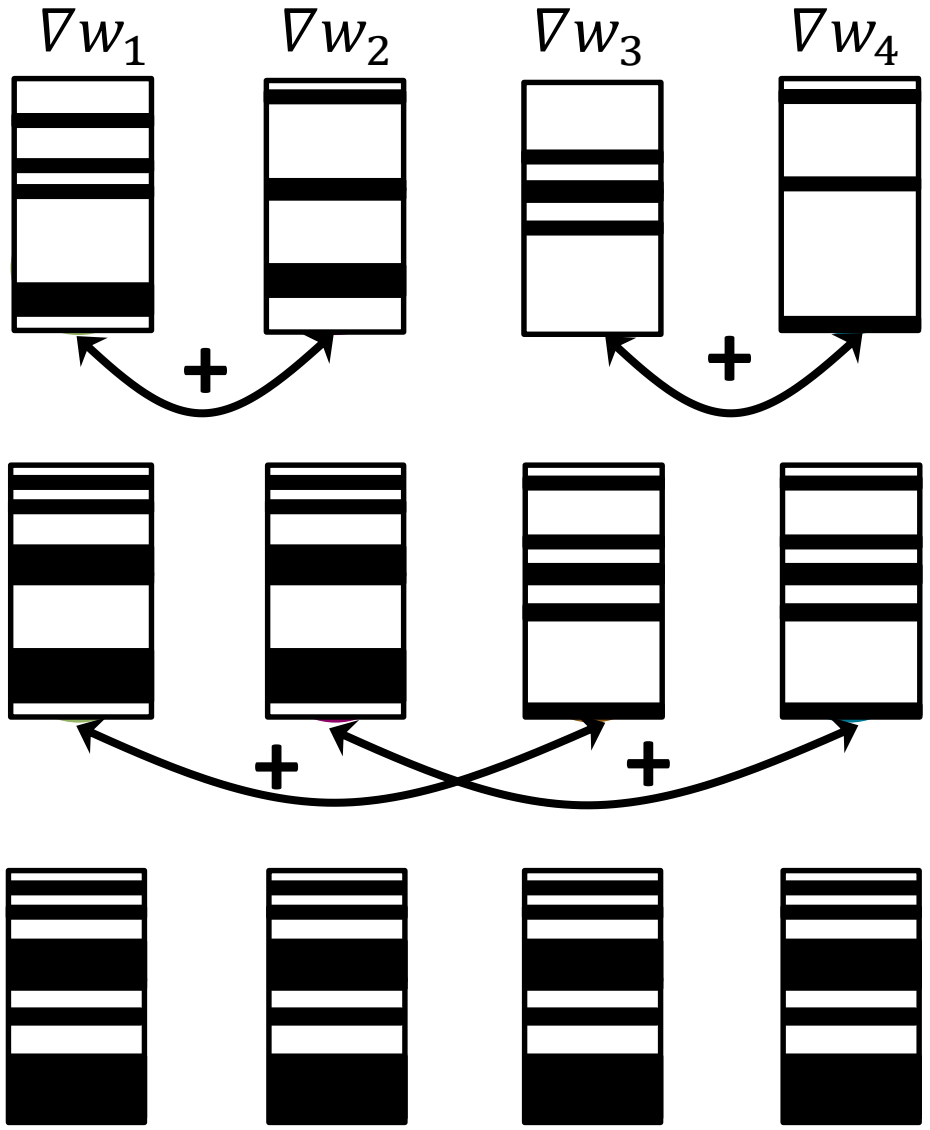


source: ai.intel.com

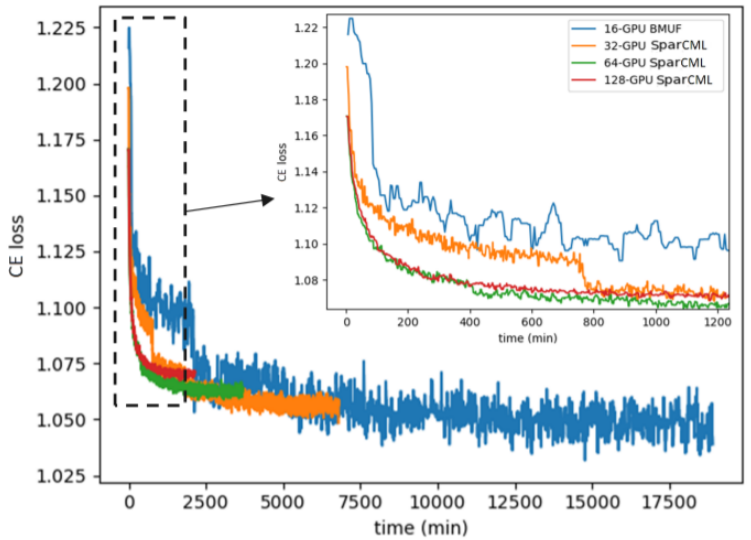


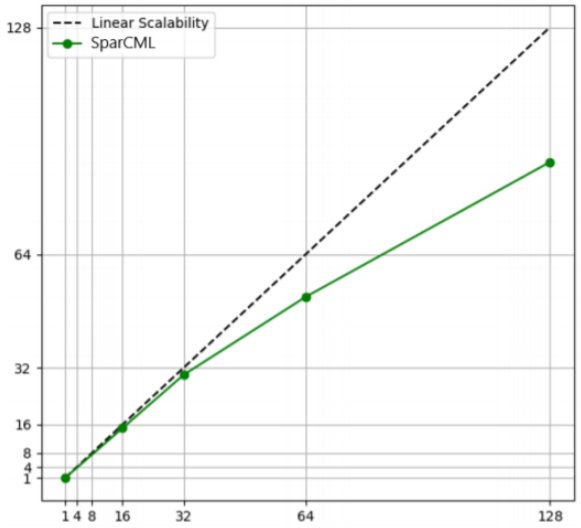
[1] S. Gupta et al. Deep Learning with Limited Numerical Precision, ICML'15
[2] F. Li and B. Liu. Ternary Weight Networks, arXiv 2016
[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014
[4] C. Renggli et al. SparCML: High-Performance Sparse Communication for Machine Learning, arXiv 2018

SparCML – Quantized sparse allreduce for decentral updates



Six epochs, 60 million params





Microsoft Speech Production Workload Results – **2 weeks → 2 days**!

System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	**1.55 (3.31)**
Piz Daint	ImageNet	AlexNet	16	Q4	**1.30 (1.36)**
Piz Daint	MNIST	MLP	8	Top16_Q4	**3.65 (4.53)**
EC2				Top16_Q4	**19.12 (22.97)**

C. Renggli et al. SparCML: High-Performance Sparse Communication for Machine Learning, SC19.

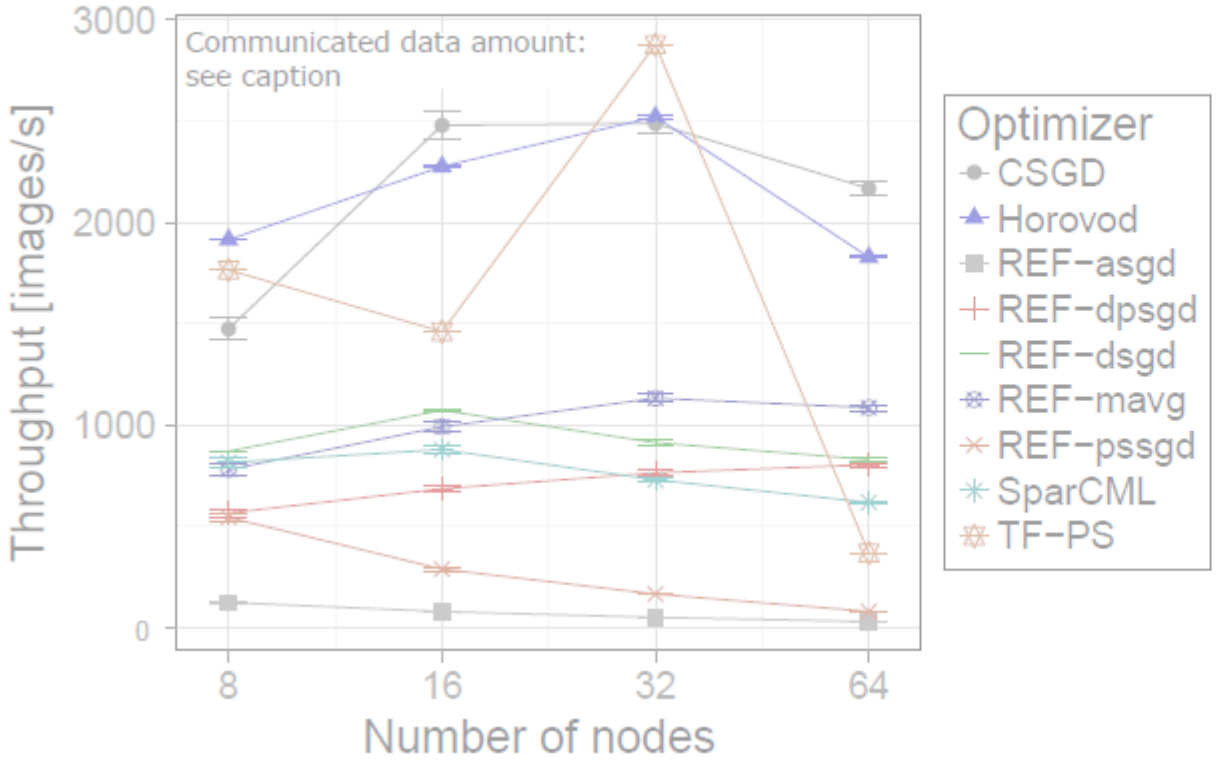
Deep500: Distributed Optimization

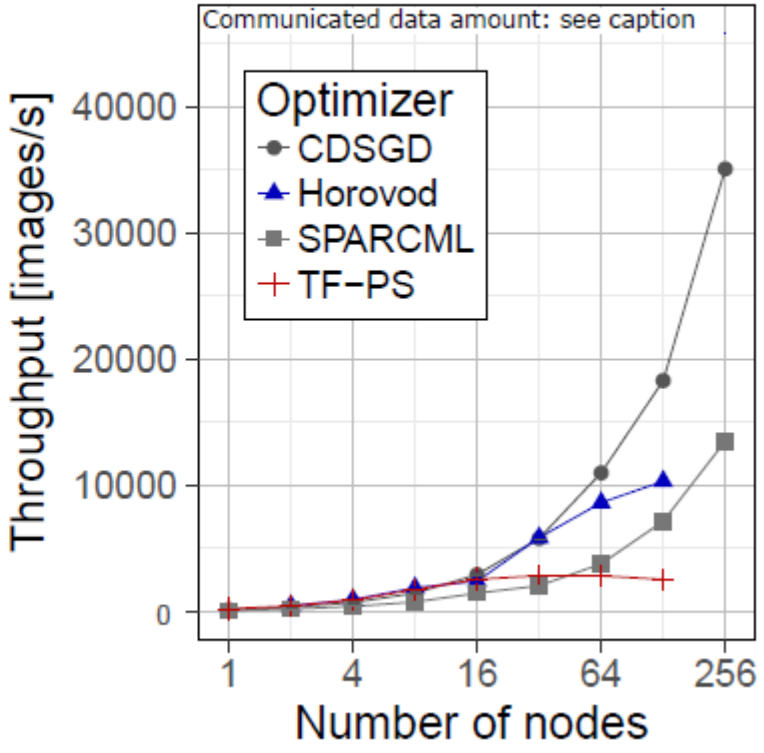
```python
class ConsistentDecentralized(DistributedOptimizer):

    def step(self, inputs):
        #batch_size = list(inputs.values())[0].shape[0]
        self.base_optimizer.new_input()
        for param in self.network.get_params():
            self.base_optimizer.prepare_param(param)
        output = self.executor.inference_and_backprop(inputs, self.base_optimizer.loss)
        gradients = self.network.gradient(self.base_optimizer.loss)
        for param_name, grad_name in gradients:
            param, grad = self.network.fetch_tensors([param_name, grad_name])
            grad = self.communication.sync_all(grad) / self.communication.size
            param = self.base_optimizer.update_rule(grad, param, param_name)
            self.network.feed_tensor(param_name, param)

        return output
```

https://github.com/deep500/deep500/blob/master/deep500/frameworks/reference/distributed_optimizers.py

Deep500: Distributed Optimization

```
#include <deep500/deep500.h>
#include <mpi.h>
```



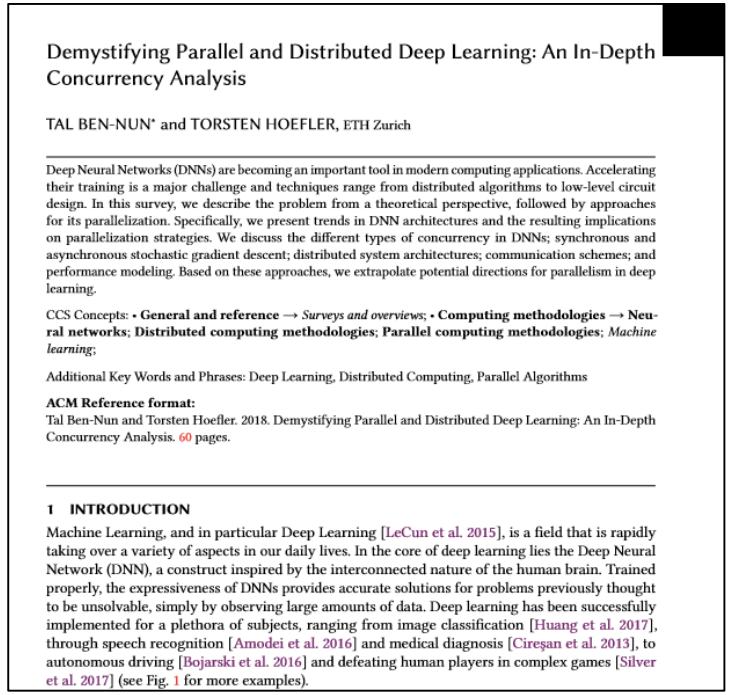



```
                float *) {
        // Do Nothing here
    }
};
```

https://github.com/tbennun/tf-models/blob/r1.11/official/utils/opt.py

HPC for Deep Learning – Summary

- **A supercomputing problem - amenable to established tools and tricks from HPC**
- **Concurrency is easy to attain, hard to program beyond data-parallelism**
- **Main bottleneck in distributed is communication – reduction by using the robustness of SGD**
- **Co-design is prevalent**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Main objective is generalization**
  - Performance-centric view in HPC can be harmful for accuracy



Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

TAL BEN-NUN* and TORSTEN HOEFLER, ETH Zurich

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs; synchronous and asynchronous stochastic gradient descent; distributed system architectures; communication schemes; and performance modeling. Based on these approaches, we extrapolate potential directions for parallelism in deep learning.

CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Computing methodologies** → **Neural networks**; **Distributed computing methodologies**; **Parallel computing methodologies**; *Machine learning*;

Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms

ACM Reference format:
Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. 60 pages.

1 INTRODUCTION

Machine Learning, and in particular Deep Learning [LeCun et al. 2015], is a field that is rapidly taking over a variety of aspects in our daily lives. In the core of deep learning lies the Deep Neural Network (DNN), a construct inspired by the interconnected nature of the human brain. Trained properly, the expressiveness of DNNs provides accurate solutions for problems previously thought to be unsolvable, simply by observing large amounts of data. Deep learning has been successfully implemented for a plethora of subjects, ranging from image classification [Huang et al. 2017], through speech recognition [Amodei et al. 2016] and medical diagnosis [Cireşan et al. 2013], to autonomous driving [Bojarski et al. 2016] and defeating human players in complex games [Silver et al. 2017] (see Fig. 1 for more examples).

https://dl.acm.org/citation.cfm?id=3320060