

Dynamic Sparse Training (DST)

A Decade of Sparse Training: Why Do We Still Stick to Dense Training?

AAAI 2026 Tutorial

The Paradox

The Problem

- Neural networks are **over-parameterized**
- Most weights are redundant or unnecessary
- Can we train with **95% fewer parameters?**

The Paradox

- DST is algorithmically mature
- Can perform on par with dense training
- **But most implementations remain dense!**

Why? Let's find out...

What is Sparse Training?

Dense Network:

- All weights active
- 1.3M parameters
- Full memory usage

Sparse Network (95%):

- Only 5% weights active
- ~65K parameters
- 95% memory reduction

95% Sparsity

| Dense | Sparse |
|-------|--------|
| 1.3M | 65K |
| 100% | 5% |

The Promise

Insane 95% parameter reduction! Can we achieve this with minimal performance loss?

Simulated vs Truly Sparse

Simulated Sparsity (Dense+Mask)

- Binary mask over dense weights
- Still stores ALL weights
- Still computes with dense ops
- Easy to implement
- No memory savings

Truly Sparse (CSR)

- Sparse matrix formats (CSR/COO)
- Stores only non-zeros
- Custom sparse kernels
- Complex to implement
- Real memory savings

Key Insight

Most research uses **simulated sparsity** - convenient but doesn't deliver true efficiency!

Network Architecture

Simple CNN for MNIST

- **Conv Layers:** 2 conv layers (16, 32 channels)
- **FC Layers:** 4 fully connected layers
- **Total:** 1.3M parameters
- **Dataset:** MNIST (60K train, 10K test)

Masked Implementation

- `MaskedConv2d` - masked convolutional layers
- `MaskedLinear` - masked fully connected layers
- Binary mask: 1 = active, 0 = pruned
- Mask applied during forward pass

Dynamic Sparse Training (DST)

Core Algorithm

Prune-and-Regrow Strategy:

- 1 **Prune:** Remove small-magnitude weights
- 2 **Regrow:** Add random new connections
- 3 **Maintain:** Constant sparsity level

Magnitude Pruning:

- Remove weights with smallest $|w|$
- Intuition: Small weights contribute less

Applied every 2 epochs during training

Random Regrowth:

- Explore new connectivity
- Reinitialize to small random values
- Alternative: Gradient-based (RigL)

Training Results

MNIST Classification (10 epochs, 95% sparsity)

| | Dense | Dense+Mask |
|-----------------|--------|------------------|
| Test Accuracy | 98.98% | 96.71% |
| Test Loss | 0.0449 | 0.0992 |
| Parameters | 1.3M | 65K (95% sparse) |
| Performance Gap | | -2.27% |

Remarkable Results!

- **Insane 95% parameter reduction** (1.3M \rightarrow 65K)
- **Only 2.27% accuracy drop** - metrics barely affected!
- Achieves **96.71% accuracy** with just 5% of weights
- DST maintains sparsity while dynamically exploring connectivity

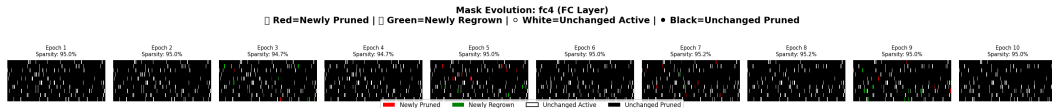
Mask Evolution During Training

Sparsity Maintained

- Initial: 95% sparsity (random)
- After DST: $\sim 95\%$ sparsity maintained
- Topology changes dynamically

Visualization (Last Layer)

Red: Newly pruned **Green:** Newly regrown **White:** Unchanged active **Black:** Unchanged pruned



FC4 Layer: Deletion and regrowth over epochs

Benchmarking: Simulated vs Truly Sparse

Inference Time Comparison (100 samples, 10 runs)

| Method | Time (ms) |
|------------------------|--------------------------------|
| Dense+Mask (GPU) | 0.68 |
| Dense+Mask (CPU) | 25.03 |
| Truly Sparse CSR (CPU) | 1351.50 |
| Truly Sparse CSR (GPU) | 1.04 |
| GPU Comparison | Dense+Mask 1.53x faster |

Critical Finding

- **Trade-off:** Slight slowdown in inference (1.53x)
- **But:** Drastic 95% parameter reduction
- Truly sparse requires specialized implementations

Why Do We Still Stick to Dense Training?

The Barriers

1 Simulated Sparsity

- Convenient but no real-world savings
- Still uses dense operations

2 Truly Sparse Implementation

- Requires sparse formats (CSR/COO)
- Needs custom kernels
- Significant engineering effort

3 System-Level Barriers

- Hardware optimized for dense ops
- Frameworks favor dense operations
- Sparse overhead for low sparsity

Key Takeaways

What We Learned

- 1 **Sparse training works!** Insane 95% parameter reduction with only 2.27% accuracy drop
- 2 **DST adapts topology** dynamically during training (prune + regrow)
- 3 **Simulated vs Truly Sparse:** Big difference in real-world efficiency
- 4 **The paradox:** Algorithm works brilliantly, but implementation remains challenging

The Promise

- **95% parameter reduction** - from 1.3M to just 65K!
- **Metrics barely affected** - only 2.27% accuracy loss
- Significant memory and energy savings
- But requires truly sparse implementations to realize full benefits

Next Steps: Hardware software co design for sparse training

Dynamic Sparse Training:

- Algorithmically mature
- **Insane 95% parameter reduction**
- **Metrics barely affected** (only 2.27% drop)
 - Performs on par with dense

But implementation remains challenging

The gap between research and practice
needs hardware-software co-design