Distributed Trees

Split Selection Formula

$Split(i) = \arg\max_{s \in S} f(\sum_{\mathbf{x} \in \mathcal{I}} g(\mathbf{x}, s))$

- s: split candidate
- $\mathcal{I}$: data points in node i
- f: aggregate to compute purity of candidate
- g: compute sufficient stats for each point $\mathbf{x}$

Row Partitioning (PLANET)

- Data points distributed across workers
- Communication cost: $O(2^D Bkm)$
- B = # split candidates/feature, k = # features, m = # workers, D = tree depth
- Use for small k/n and small D/n

Column Partitioning (YGGDRASIL)

- Features distributed across workers
- Each worker evaluates subset of features
- Communication cost: $O(2^D m + Dnm)$
- Use for large k/n and large D/n

Neural Networks

- For layers $l \in \{1, \ldots, L\}$:
  - $h_l = W_l^\top o_{l-1}$
  - $o_l = \sigma_l(h_l)$
- Final output: $\phi(x) = o_L$
- Matrix multiplies: $O(nm)$ for $n \times m$ matrix

Gradient Descent Update

- $w_{i+1} = w_i - \alpha_i \nabla f(w_i)$
- α_i: step size at iteration i
- $\nabla f(w_i)$: gradient of loss w.r.t weights

Chain Rule

- $\frac{d}{dx} f(g(h(x))) = f'(g(h(x))) \cdot g'(h(x)) \cdot h'(x)$
- Naive: $O(k^2)$ for k compositions
- With memorization: $O(k)$

Automatic Differentiation

- Forward pass: Compute and store intermediates
- Backward pass: Apply chain rule efficiently
- Key feature of deep learning frameworks

Foundation Models

- Pretraining: Train on large dataset
- Alignment: Tune for specific objectives
- Finetuning: Adapt to downstream task

Full Finetuning

- Start with pretrained weights w_{pre}
- Update all model parameters on task data

Linear Probing

- Freeze weights & only ft final layer
- Can underperform full finetuning

LoRA (Low Rank Adaptation)

- Low rank (r) approx. of weight updates ΔW
- $\Delta W = BA$ where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times d}$

Distributed Optimization

Gradient Descent Update

- Compute on full dataset in parallel
- $O(nk)$ distributed compute, $O(k)$ local storage

Stochastic GD Update

- Sample single point j randomly
- $O(k)$ distributed compute, $O(k)$ local storage

Mini-batch SGD

- $\mathcal{B}_i \subseteq \{1, \ldots, n\}$ random mini-batch
- $O(bk)$ compute for batch size b

One-shot Averaging

- Solve locally: $w_m^* = \arg\min_w f_m(w)$
- Average: $w = \frac{1}{M} \sum_{m=1}^M w_m^*$
- Minimal communication but approximate

CoCoA Framework

- Local subproblems solved to accuracy $\Theta \in [0, 1)$
- $w^{(t+1)} = w^{(t)} + \sum_{k=1}^K \Delta w_k$
- Controls local computation vs communication
- Convergence guarantees for convex problems

Convergence Rates (to ϵ-accuracy)

Condition	GD	SGD
Convex	$O(1/\epsilon^2)$	$O(1/\epsilon^2)$
+ Lipschitz	$O(1/\epsilon)$	$O(1/\epsilon^2)$
+ Strong convex	$O(\log(1/\epsilon))$	$O(1/\epsilon)$

DL Optimization

Adaptive Learning Rates

- AdaGrad: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{G}_t^{-1} \nabla f(\mathbf{w}_t)$
  - $(G_t)_{ii} = \sqrt{\sum_{j=1}^t \nabla f(\mathbf{w}_j)_i^2}$
  - Separate learning rate per parameter
  - Issue: Learning rates decay to zero
- RMSProp: Moving average instead of sum
  - $(g_t)_i = \beta (g_{t-1})_i + (1 - \beta)(\nabla f(\mathbf{w}_t)_i)^2$
  - $(G_t)_{ii} = \sqrt{(g_t)_i}$

Momentum Methods

- Polyak: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f(\mathbf{w}_t) + \beta(\mathbf{w}_t - \mathbf{w}_{t-1})$
- Nesterov: Evaluate gradient at "look-ahead" point
  - $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f(\mathbf{w}_t + \beta(\mathbf{w}_t - \mathbf{w}_{t-1})) + \beta(\mathbf{w}_t - \mathbf{w}_{t-1})$
- Adam: Combines momentum and RMSProp
  - $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{G}_t^{-1} \mathbf{v}_t$
  - $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \nabla f(\mathbf{w}_t)$ (momentum)
  - $(g_t)_i = \beta_1 (g_{t-1})_i + (1 - \beta_1)(\nabla f(\mathbf{w}_t)_i)^2$ (RMSProp)

Other Techniques

- Batch Normalization: Normalize layer inputs
  - $\hat{x}_i = \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$
  - $y_i = \gamma \hat{x}_i + \beta$ (learnable parameters)
  - Use batch stats during training, population stats at test
- Early Stopping: when val accuracy starts decreasing

Distributed Deep Learning

- **Data Parallel**: replicate model, partition data
  - Compute gradients on local data shard
  - Aggregate gradients globally, update model replica
  - DDP: basic data parallel, single GPU per process
  - FSDP: shards model params/gradients for memory
- **Model Parallel**: partition model, replicate data
  - Tensor Parallel: split matrix ops across devices
  - Pipeline Parallel: split sequential layers
  - Communication pattern defines parallelism type
- **Communication Strategies**:
  - Parameter Server: central server aggregates updates
    - Workers: $w' = w - \eta \Delta w$
    - $T_{comm} = O(mk)$ bottleneck at server
    - Good for async, sparse updates
  - Ring All-reduce: peer-to-peer communication
    - Each worker sends/receives from neighbors
    - $T_{comm} = O(k)$, optimal bandwidth use
    - Takes $2(m-1)$ rounds for m workers
- Large batches: more computation/less communication, but: DR & poor generalization
- **Diminishing Returns**:
  - Gradient diversity: $\Delta_D(w) := \frac{\sum_{i=1}^n \|\nabla f_i(w)\|_2^2}{\|\sum_{i=1}^n \nabla f_i(w)\|_2^2}$
  - Lower diversity $\rightarrow$ worse large batch performance
  - Additional gradients provide less value
- **Poor Generalization**:
  - Large batches converge to sharp minima
  - Sharp minima generalize worse than flat minima
  - Solutions: noise injection, progressive batch sizes
- **Gradient Compression**:
  - Quantization: reduce bits per value
  - Sparsification: drop small gradients
  - ATOMO: optimize compression-accuracy tradeoff
  - Must maintain convergence guarantees

Hardware Acceleration

- **Tensor Cores**: Matmul in newer GPUs
  - Multiply 4x4 matrices in one clock cycle (128 FLOP)
  - Mixed prec.: multiply in FP16, accumulate in FP32

Parameter-Efficient Finetuning

- **Specification**: Select subset of parameters to tune
  - Linear probing: Only tune final layer
- **Addition**: Add new trainable parameters
  - Adapters: Insert trainable layers between frozen

pretrained layers
- **Reparameterization**: Transform parameters for efficient training
  - LoRA: Learn low-rank update matrices: $\mathbf{W}_{\text{ft}} = \mathbf{W}_{\text{pt}} + \frac{\alpha}{r}\mathbf{AB}$
  - QLoRA: Quantize pretrained weights + LoRA update

Hyperparameter Tuning

- **Random vs Grid Search:**
  - Grid: Evenly discretize search space (linear/log)
  - Random: Sample HPs from defined ranges
  - Random often better when n (configs) linear in d
- **Successive Halving:**
  - Start with n configs, train for r iterations each
  - Keep top η^{-k} configs after $\eta^k r$ iterations
  - Repeat until single best config remains
- **Hyperband:** Multiple brackets of successive halving
  - Each bracket: different n vs r trade-off
  - Outer loop: s_{max} to 0, inner: successive halving
  - $n_i = \lfloor \frac{n}{s+1} \rfloor$, $r_i = r\eta^i$
  - Optimal bracket exists but unknown a priori
- **Non-Stochastic Best Arm:**
  - Loss sequence: $\lim_{k\to\infty} \ell_{i,k} = \nu_i$
  - Error bound: $|\ell_{i,k} - \nu_i| \leq \gamma_k$
  - Goal: $\min_i \sum T_i$ subject to $\nu_i = \min_i \nu_i$

Inference & Compression

Inference Metrics:
- Accuracy: How well model performs
- Model size: Memory needed for parameters
- Latency: Time per single prediction
- Throughput: Predictions per time unit
- Energy: Power consumption per prediction

Quantization:
- Low-precision arithmetic for inference
- FP32 $\rightarrow$ FP16/INT8/Binary
- Common formats: {Sign, Exponent, Mantissa}
- bfloat16: {1,8,7} matches FP32 range

Pruning:
- Unstructured: Individual weights
- Structured: Entire filters/channels
- Methods: Magnitude, Gradient, Taylor
- Iterative: Prune $\rightarrow$ Retrain cycles

Knowledge Distillation:
- Student learns from teacher's soft targets
- Softmax temperature: $p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$
- Loss: $\mathcal{L} = \alpha H(y, \sigma_s) + \beta H(\sigma_t, \sigma_s)$
- $T \in [1, 20]$, larger T for smaller students

Evaluation:
- Pareto frontier: accuracy vs. cost tradeoff
- No dominated solutions exist on frontier
- Compare: same accuracy, lower cost
- Or: same cost, higher accuracy

Federated Learning

ERM Objective: $\min_w f(w) = \sum_{k=1}^m p_k F_k(w)$ where $F_k(w) = \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$

FedAvg Algorithm:
- Local training: E epochs of SGD on each device
- Average model updates across devices: $w_{t+1} = \sum_k p_k w_k^t$
- Communication cost: $O(mk)$ for m devices, k parameters

FedProx: Adds proximal term to limit impact of heterogeneity
- Local objective: $\min_{w_k} F_k(w_k) + \frac{\mu}{2}\|w_k - w^t\|^2$
- Converges under B-dissimilarity: $\mathbb{E}[\|\nabla F_k(w)\|^2] \leq \|\nabla f(w)\|^2 B^2$
- IID data: $B = 1$, non-IID data: $B > 1$

q-FFL: Fair resource allocation objective
- $\min_w \frac{1}{q+1}(p_1 F_1^{q+1} + p_2 F_2^{q+1} + \cdots + p_N F_N^{q+1})$
- $q \to 0$: standard objective; $q \to \infty$: minimax fairness
- Reduces accuracy variance across devices while maintaining mean

Key Challenges:
- Statistical heterogeneity: non-IID data across devices
- Systems heterogeneity: variable hardware/connectivity
- Communication bottleneck: limited bandwidth, high latency
- Privacy constraints: raw data cannot leave devices

Model Approaches:
- Global: Single shared model, learn from all devices
- Local: Independent models per device, no sharing
- MTL: Personalized models that learn from peers

Foundation Models Training

- Next-token prediction on text, image, video, audio tokens
- Error between learned model $\hat{p}_\theta(y|x)$ and ground-truth $p^*(y|x)$:

$$|\hat{p}_\theta(y|x) - p^*(y|x)| \propto \frac{1}{|\mathcal{D}(y|x)|^\alpha}$$

- Error reduces with more data similar to target x

Synthetic Data

- Generated by model (LLM) trained on expert data
- Key challenges:
  - Verification of correctness
  - Bias amplification/model collapse
  - Need for diversity
- Types:
  - Positive data: Correct answers $\sim \pi$
  - Negative data: Incorrect answers $\sim \pi$

Reinforcement Learning for LLMs

- Sparse reward MDP with terminal reward
- Q-value: Expected future reward under policy $\tilde{\pi}$

$$Q_\pi(x, \hat{y}_{1:i-1}; \hat{y}_i) = \mathbb{E}_{y^{\text{new}}_{i+1:L} \sim \tilde{\pi}(\cdot|x, \hat{y}_{1:i})}[r([\hat{y}_{1:i}, y^{\text{new}}_{i+1:L}], y)]$$

- Advantage: Relative change in value function

$$A_\pi(x, \hat{y}_{1:i-1}; \hat{y}_i) = Q_\pi(x, \hat{y}_{1:i-1}; \hat{y}_i) - Q_\pi(x, \hat{y}_{1:i-2}; \hat{y}_{i-1})$$

Process Advantage Verifiers (PAVs)

- Use advantage as dense reward bonus in RL
- Improvement bound:

$$V^{t+1}(\pi) - V^t(\pi) \succcurlyeq \gamma \cdot \text{var}_{\tilde{\pi}}[A_{\tilde{\pi}}] + \gamma \cdot \langle A_\pi, A_{\tilde{\pi}} \rangle$$

- Sample efficiency: 5-6x over outcome rewards only
- Performance gain: 6-7% improvement