Advanced Topic A11: *The Lottery Ticket Hypothesis*

Data Science with Python – BSc Advanced Lectures

Joerg Osterrieder

10 Minutes

Motivation: Large Models Are Wasteful

Modern neural networks are massively over-parameterised

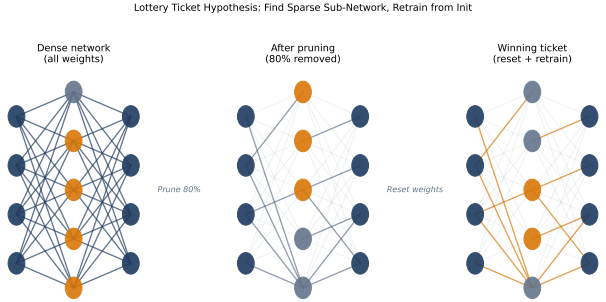
- A ResNet-50 has 25 million parameters; a GPT-3 variant has 175 billion
- At inference: most activations are near zero; most weights contribute little
- Pruning can remove 90%+ of weights with near-zero accuracy loss
- Why train a large model if a sparse sub-model has the same accuracy?
- Finance: model compression reduces inference latency for real-time fraud scoring; smaller models fit on edge devices for local risk assessment

Over-parameterisation aids optimisation but is not needed for inference

The Lottery Ticket Hypothesis

Every large network contains a small, trainable sub-network

- Frankle & Carlin (2019): a randomly initialised dense network contains sparse sub-networks ("winning tickets") that can be trained to full accuracy in isolation
- A winning ticket: a subset of weights with their original initial values that, when trained from those initial values, matches the dense network's accuracy
- The hypothesis implies: the role of large networks is to *find* the winning ticket, not to use all its capacity at inference
- This reframes training as a search problem: find the sparse structure, then train it alone



Frankle & Carlin (ICLR 2019): winner of the ICLR Best Paper award

Magnitude Pruning

Remove the smallest-magnitude weights after training

- Train the full network to convergence
- Prune a fraction p of weights with the smallest absolute value (set to zero)
- The remaining weights form the sparse sub-network
- Reset the surviving weights to their original initial values (not the trained values)
- Retrain the sparse sub-network from those reset initial values
- Key result: the reset sub-network matches or exceeds the dense network's accuracy if the winning ticket initial values are used

Weight resetting is critical: using trained values instead of initial values fails to find winning tickets

Early Bird Tickets and Late Resetting

Winning tickets can be found early in training without full convergence

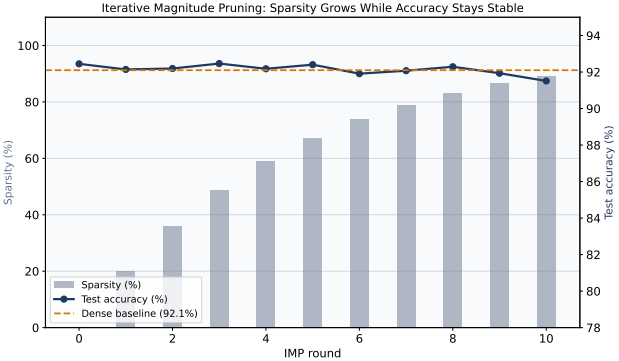
- You & Ding (2020) "Early Bird Tickets": run training for only 10–20% of epochs, apply IMP, and get a winning ticket nearly as good as one from a fully-trained network
- Cost reduction: finding the ticket takes 10–20% of the original training budget
- Frankle et al. (2020): for large networks (ResNet-50+), resetting to the initial values fails – must rewind to an early checkpoint ($\sim$1% of training) instead
- This suggests that the winning ticket structure is *determined early in training* but the initial values may not be the right reset target at scale

Early bird + late rewind: reduces the LTH search cost to 10–20% of full training

Iterative Magnitude Pruning (IMP)

Prune gradually to find sparser and sparser winning tickets

- One-shot pruning to 90% sparsity is unstable: iterative pruning works better
- IMP algorithm: train $\rightarrow$ prune 20% $\rightarrow$ reset $\rightarrow$ retrain $\rightarrow$ prune 20% $\rightarrow$ repeat
- After n rounds at pruning rate p per round: retained fraction is $(1-p)^n$
- IMP finds tickets at 1–5% density that still match the original dense accuracy
- Cost: n full training runs – expensive but the resulting models are small

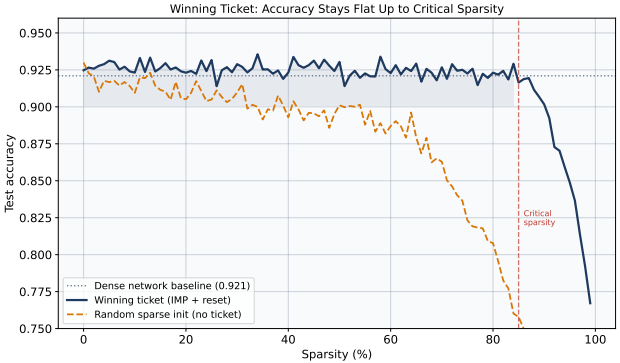


IMP finds denser winning tickets than one-shot pruning at the same target sparsity level

Winning Ticket Accuracy Curve

Sparse sub-networks match or beat dense accuracy up to a critical threshold

- As sparsity increases from 0% to 90%: accuracy stays flat or slightly improves (implicit regularisation from pruning reduces overfitting)
- Beyond a critical sparsity threshold: accuracy drops sharply
- The winning ticket lives in the flat region: typically 80–95% sparsity
- The critical threshold depends on task complexity and architecture depth



The flat accuracy region defines the range of winning tickets for a given architecture

Structured vs Unstructured Pruning

Not all sparsity is equally hardware-friendly

- **Unstructured pruning:** remove individual weights anywhere; achieves highest sparsity; but sparse matrix multiplication is not well-supported on standard GPUs
- **Structured pruning:** remove entire filters, attention heads, or layers; lower sparsity but produces dense sub-matrices that run efficiently on hardware
- **Semi-structured (2:4):** NVIDIA A100 supports 2-out-of-4 sparsity natively; 2x speedup
- In practice: unstructured pruning for edge devices (custom ASIC), structured for GPU deployment
- Finance: edge inference on branch ATMs favours structured pruning; cloud inference favours 2:4

Sparsity without hardware support only reduces model size, not inference latency

Knowledge Distillation as an Alternative

Compress a large teacher into a small student via soft targets

- Hinton et al. (2015): train a small "student" model to match the softmax outputs of a large "teacher" model, not the hard labels
- Soft targets carry more information than one-hot labels: the teacher's probability distribution reveals inter-class similarities
- Distillation loss: $\mathcal{L} = \alpha \mathcal{L}_{\text{CE}} + (1 - \alpha)\mathcal{L}_{\text{KD}}$ where $\mathcal{L}_{\text{KD}}$ matches softened teacher probabilities (temperature $T > 1$)
- Pruning vs distillation: pruning preserves the original architecture; distillation allows a different, smaller architecture; often combined (prune then distil)

DistilBERT (Sanh et al. 2019): 40% smaller, 60% faster, 97% of BERT performance

Neural Architecture Search vs Pruning

Two routes to compact models: search vs shrink

- **NAS**: design the architecture from scratch to be small and accurate; uses reinforcement learning, evolutionary search, or differentiable relaxation (DARTS)
- **Pruning**: start from an over-parameterised architecture; remove redundant capacity
- In practice: NAS-found architectures (EfficientNet, MobileNetV3) outperform pruned dense networks at the same parameter count – but NAS is 100x more expensive to run
- Lottery Ticket insight: NAS can be seen as searching for the winning ticket directly, bypassing the expensive IMP loop
- Finance: for custom architectures, pruning is more practical; use NAS only when latency constraints are extremely tight and training compute is available

EfficientNet found via NAS achieves better accuracy/FLOPs than any pruned ResNet at same size

Quantisation and Mixed Precision

Reducing numerical precision for further compression

- Standard training uses fp32 (4 bytes per weight); inference can use int8 (1 byte) or fp16 (2 bytes)
- Post-training quantisation (PTQ): calibrate scale factors on a small dataset after training
- Quantisation-aware training (QAT): simulate quantisation during training; better accuracy for ¡4-bit
- 4-bit quantisation (GPTQ, AWQ): large language models compressed to 4 bits with minimal perplexity loss
- Pruning + quantisation stacks: a 90% sparse model quantised to int8 is 40x smaller than fp32 dense

Pruning, distillation, and quantisation are complementary and commonly applied together

Zero-Cost Proxies: Finding Tickets Without Training

Can we identify the winning ticket at initialisation?

- IMP requires multiple training runs: expensive at scale
- **SNIP** (Lee et al. 2019): prune weights whose removal causes the largest loss change at initialisation (single forward-backward pass)
- **GraSP**: preserve gradient flow; prune weights that would most reduce the gradient norm
- **Synaptic Flow**: avoids layer collapse; provably preserves trainability
- Zero-cost proxies are 100–1000x cheaper than IMP but find lower-quality tickets; accuracy gap closes for moderate sparsities ($<80\%$)

Zero-cost pruning at init: single forward pass; quality lower than IMP but vastly faster

Limitations and Open Questions

The lottery ticket hypothesis has practical and theoretical limits

- Finding the winning ticket requires multiple full training runs: finding a sparse model that trains fast does not itself train fast – the search is expensive
- Winning tickets are not transferable across architectures or even different random seeds
- At very large scale (transformers): Frankle et al. (2020) found tickets require rewinding to later checkpoints, not the initial values – the original hypothesis weakens
- Structured winning tickets are rare: most identified tickets are unstructured (hardware-unfriendly)
- Open: can we predict the winning ticket without training first? (Synaptic Flow, SNIP, GraSP)

The hypothesis is empirically robust for CNNs; for large transformers the picture is more complex

Pruning for Regulatory Explainability

Sparser models are easier to explain and audit

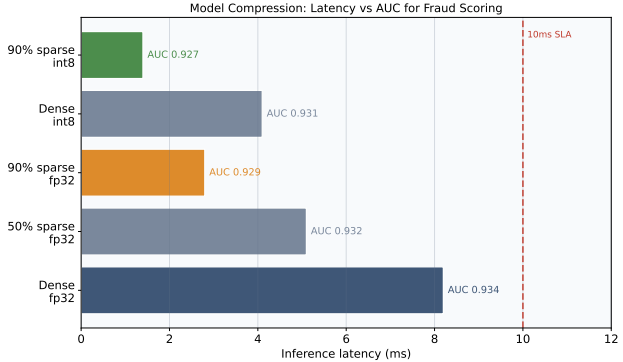
- Dense 200-layer networks: near-impossible to trace which input features drive a prediction
- A 95%-sparse network: most weights are zero; fewer active paths from input to output
- Sparse networks combined with SHAP (A04) produce smaller, more stable SHAP values: fewer non-zero weights means fewer feature interaction terms
- Regulators favour simpler models: SR 11-7 requires models to be explainable and amenable to validation; pruned models reduce validation surface area
- Finance: submit a pruned model to the model risk team alongside its dense version; the pruned model passes validation faster due to reduced complexity

Pruning and explainability reinforce each other: sparser models produce cleaner attributions

Finance Application: Model Compression for Real-Time Scoring

Pruning fraud detection models to meet latency budgets

- Payment fraud detection: scoring must complete in <10ms to avoid blocking the transaction
- A 95%-sparse pruned model can run 5–10x faster with equivalent AUC on the training distribution
- IMP workflow for fraud: train dense XGBoost or MLP → prune → reset → retrain on monthly snapshots of transaction data
- Structured pruning of attention heads in transformer-based NLP compliance models: removes low-utility heads with <2% F1 loss



Transferability of Winning Tickets

Can a ticket found on one task be reused on another?

- Morcos et al. (2019): winning tickets found on large datasets transfer to smaller ones; a ticket from ImageNet transfers to CIFAR-10 with minimal accuracy loss
- The sparse structure generalises: the good initialisation, not just the mask, transfers
- Transferability is stronger between similar domains (natural images) than across modalities
- Finance: a ticket found on general tabular data may transfer to a specific credit dataset, reducing the cost of finding tickets from scratch for each new product line
- Caveat: transfer of the mask without the initial weights does not work; both the mask and the reset initial values are needed

Ticket transfer: IMP once on a large source task, reuse the ticket on smaller target tasks

Code Pointer

Pruning in Python

- **PyTorch pruning**: `torch.nn.utils.prune.l1_unstructured(module, 'weight', amount=0.5)`; built-in support for magnitude, random, and structured pruning; mask-based implementation
- **torch-pruning**: structured pruning library; dependency-aware pruning of full filters
- **SparseML (Neural Magic)**: IMP + one-shot + quantisation; scikit-learn and PyTorch support
- **Distillation**: Hugging Face `DistilBERT` or custom KD loop with `nn.KLDivLoss(reduction='batchmean')`
- Finance: wrap your fraud MLP with PyTorch pruning; benchmark with `time.perf_counter` to verify latency improvement before deployment

Always measure latency after pruning: theoretical FLOPs reduction $\neq$ wall-clock speedup

Summary and Open Questions

Three things to remember

- The Lottery Ticket Hypothesis: every large network contains a sparse sub-network that, trained from its original initial values, matches the dense network – finding it requires iterative magnitude pruning with weight resetting
- Pruning, distillation, and quantisation are complementary compression tools; structured pruning maps to hardware speedup; unstructured pruning maximises sparsity
- Finance: pruned models meet real-time latency budgets for fraud scoring with equivalent AUC to the dense model

Open questions:

1. Can we identify winning tickets before training (zero-cost proxies)?
2. Do winning tickets transfer across tasks or datasets?
3. Is the lottery ticket phenomenon a property of the optimiser or the loss landscape?

Further reading: Frankle & Carlin (2019) LTH; Han et al. (2015) Deep Compression; Hinton et al. (2015) KD