Tips and Tricks for Computational Savings

Soumadeep Saha

Contents

PART 1 - General Techniques

- Pruning.
- Distillation.
- Quantization.

PART 2 - Transformers

- Dealing with quadratic growth.
- Curse of parallelism - Linear Models.
- Input size issues.

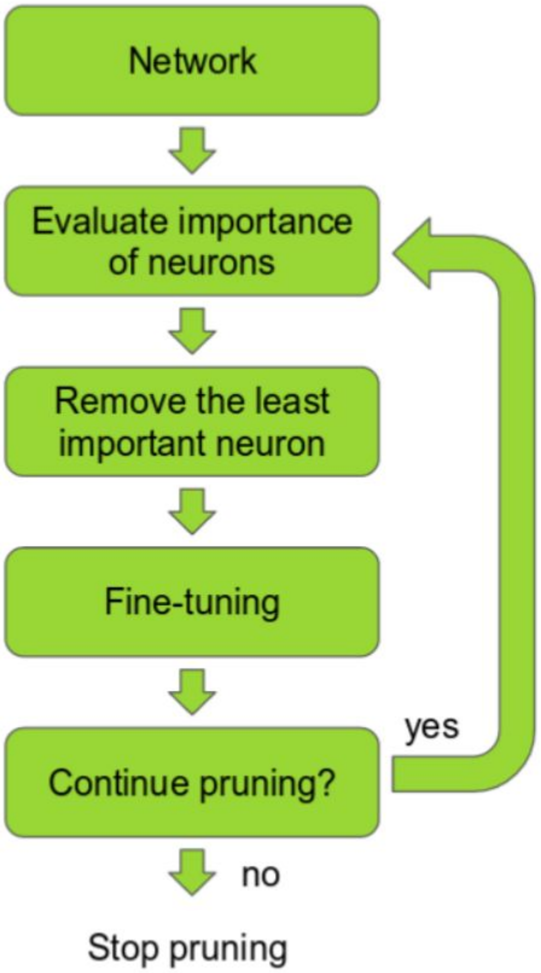
PART 1

General Techniques

Pruning

Are all weights necessary?

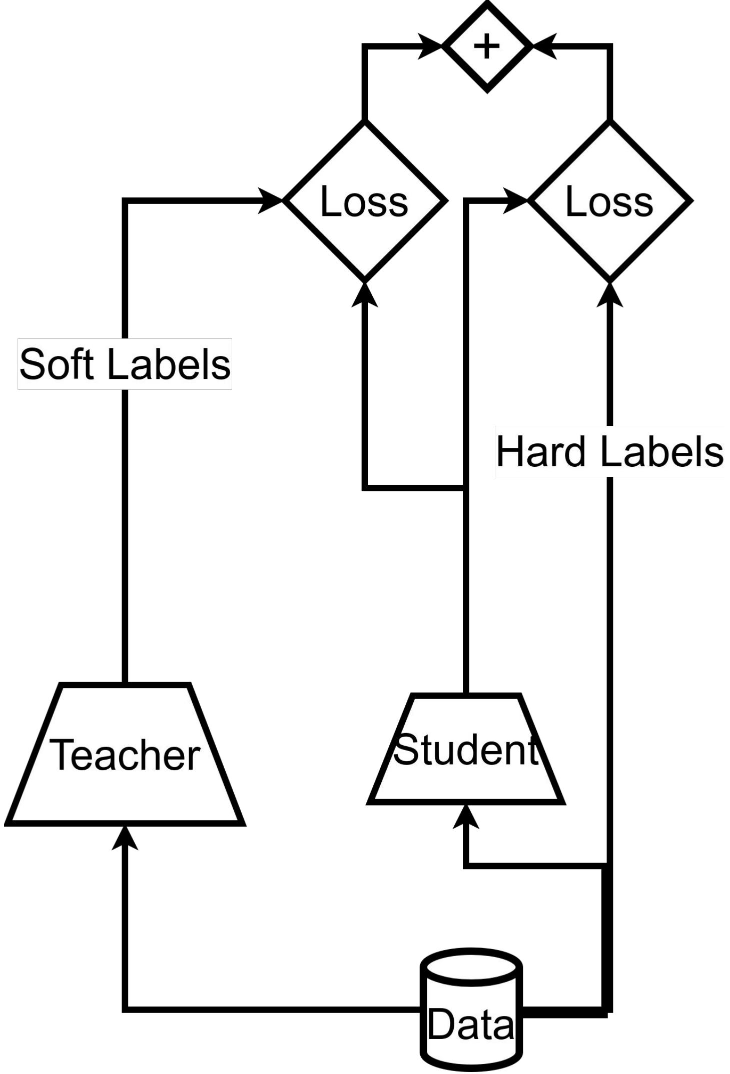
2nd derivatives - Hessians



Distillation

Smaller network trained on outputs from a larger network.

E.g. DistilBert
0.6x size | 0.95x performance



Quantization

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}),$$

Reduce number of bits per parameter - saves memory and compute.

FP32 <-> Int8 [-127,127]

Done in chunks to avoid outliers.

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}}$$

PART 2

Transformers

Quadratic Growth

$$\text{For } n \text{ tokens, we have } (Q_i, K_i, V_i) \quad \forall i \in \{1, \ldots, n\}$$

$$A_i = \underbrace{\sum_j \text{softmax}(Q_i, K_j)}_{\text{PROBLEM!}} \cdot V_j \qquad \rightarrow \mathcal{O}(n^2)$$

PROBLEM!

Quadratic Growth

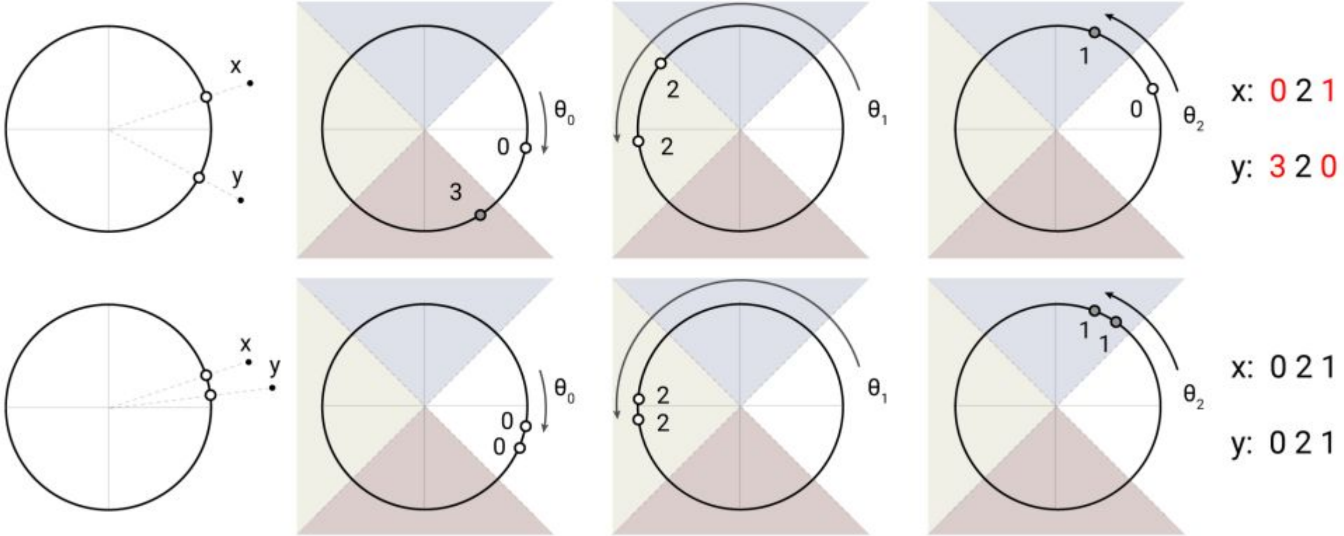
$$A_i = \sum_j \text{softmax}(Q_i, K_j) \cdot V_j$$

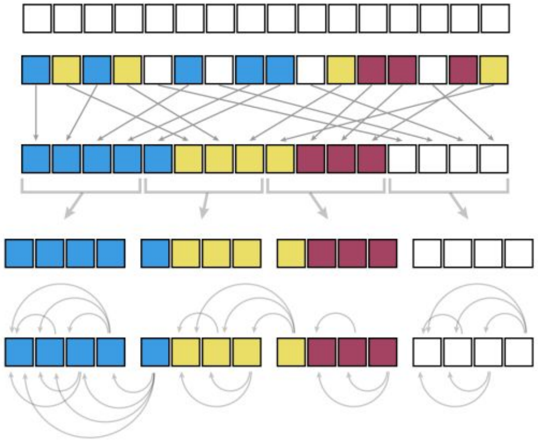
PROBLEM!

Replace softmax of cosine similarity with locality sensitive hashing algorithms.

-> **Reformer architecture.**

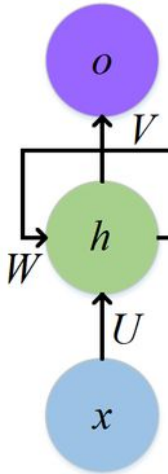
Quadratic Growth



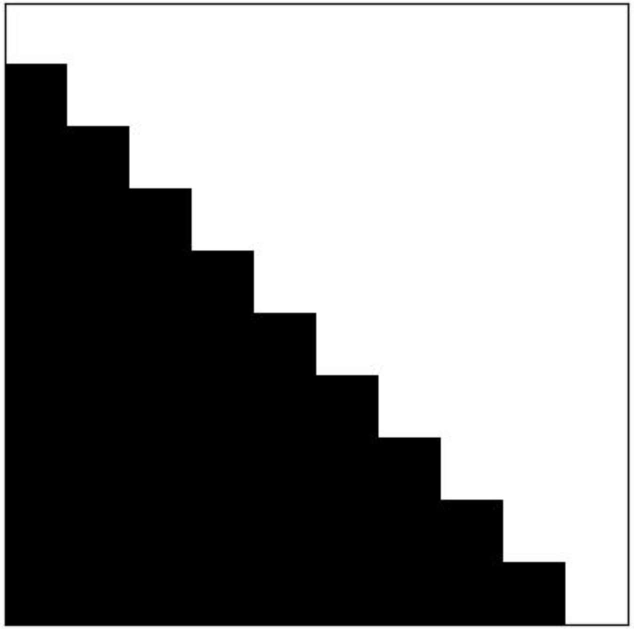


$$\mathcal{O}(n \log(n))$$

Parallelism - Recall...



$$A_i = \sum_j \operatorname{softmax}(Q_i, K_j) \cdot V_j$$



The quick brown fox jumps over the lazy dog.

Parallelism - Softmax begone!

$$A_i = \sum_j \text{softmax}(Q_i, K_j) \cdot V_j$$

$$A_i = \sum_j \text{softmax}(Q_i, K_j) \cdot V_j$$

$$= \sum_j \frac{\exp(Q_i \cdot K_j)}{\sum_l \exp(Q_i \cdot K_l)} \cdot V_j$$

$$= \sum_j \frac{sim(Q_i, K_j)}{\sum_l sim(Q_i \cdot K_l)} \cdot V_j$$

$$= \frac{1}{\sum_l sim(Q_i \cdot K_l)} \sum_j sim(Q_i, K_j) \cdot V_j$$

$$= \frac{1}{n_i} \sum_j \left(\phi(Q_i)^T \phi(K_j) \right) \cdot V_j$$

$$= \frac{1}{n_i} \phi(Q_i)^T \sum_j \phi(K_j) \cdot V_j$$

This can be made recurrent $\rightarrow \mathcal{O}(n)$

But we lose the powerful non-linearity

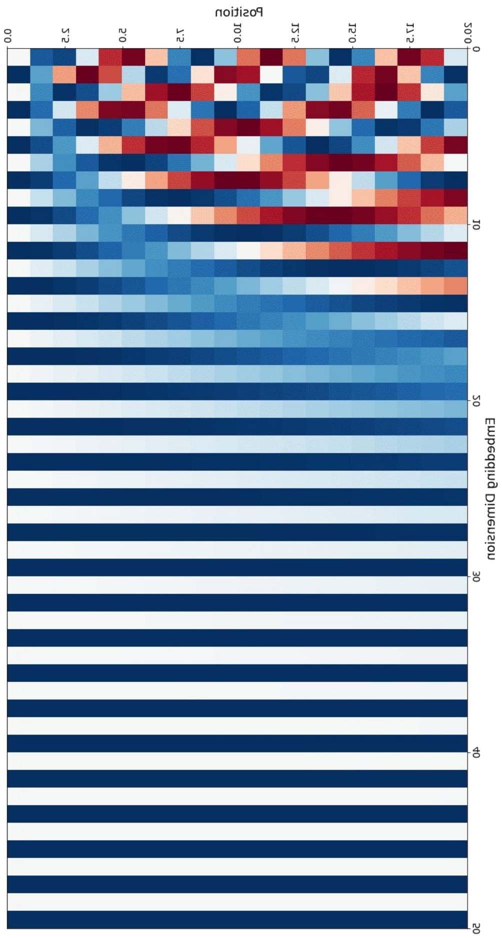
Other input size issues...

Long input sizes take a lot of memory & compute time…

But, they also maybe out of distribution!

Fine-tuning + Linear Interpolation

Recall...



Key takeaways

- General compression techniques like distillation, quantization, etc are helpful in this context.
- Quadratic growth in time - LSH.
- Parallelism requires a lot of memory - Linear attention.
- Position embeddings may be OOD.