Natural Language Processing

Berkeley

N L P

Efficiency

Kevin Lin – UC Berkeley

April 24, 2023

(many slides credits to EMNLP 2020 High Performance NLP tutorial)

Efficiency

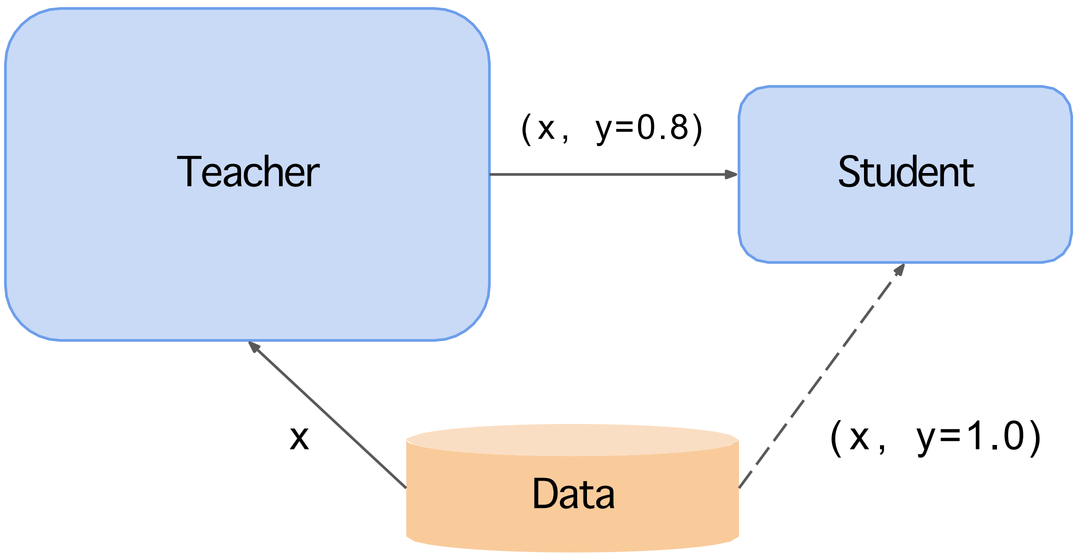
Today

- Knowledge Distillation
- Quantization
- Pruning
- Efficient Attention
- Efficient Architectures

Knowledge Distillation

Hinton et al., 2015

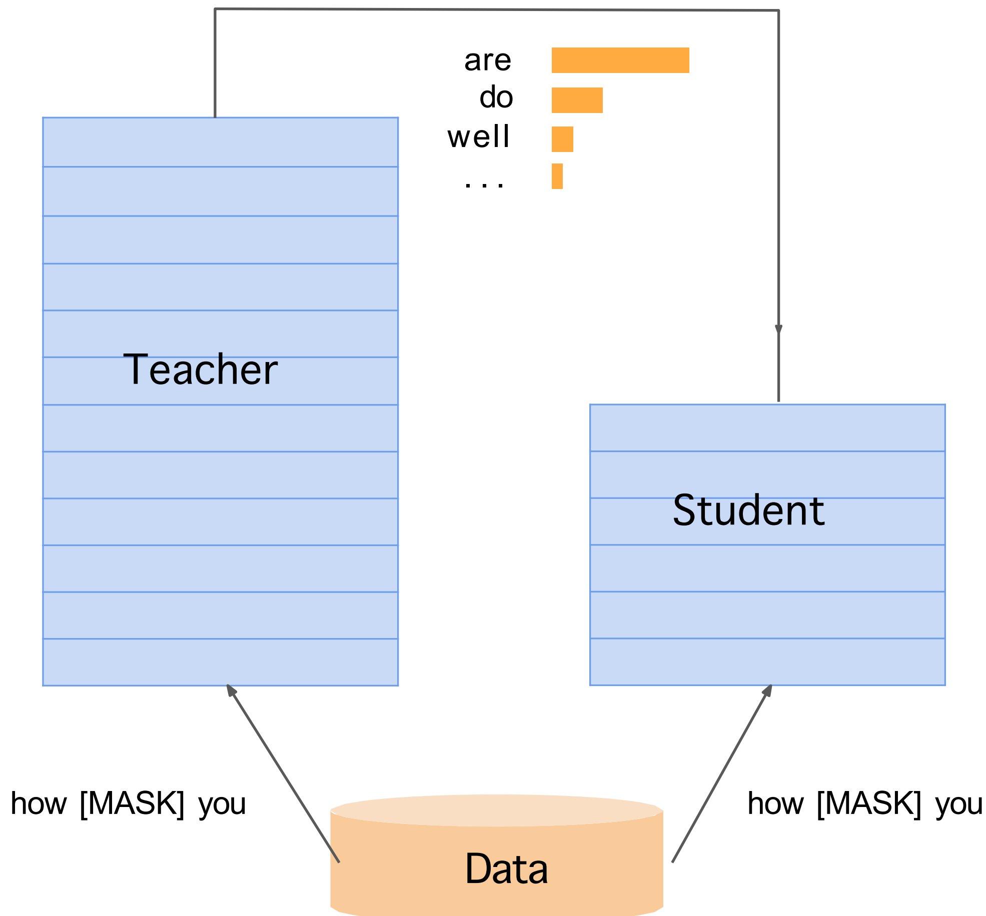
Distilling the Knowledge in a Neural Network



Knowledge Distillation for Pre-training

Sanh et al., 2019

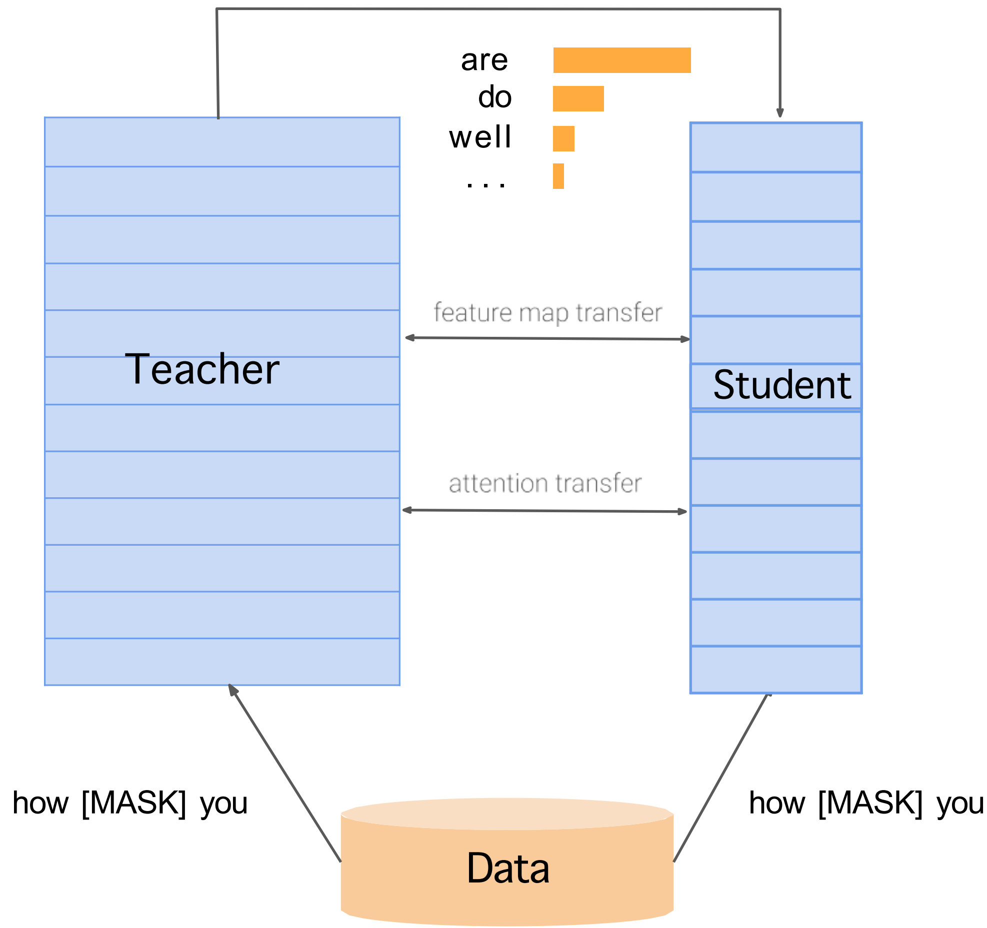
DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter



Knowledge Distillation for Pretraining

Sun et al., 2019

MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices



Knowledge Distillation for Fine-Tuning

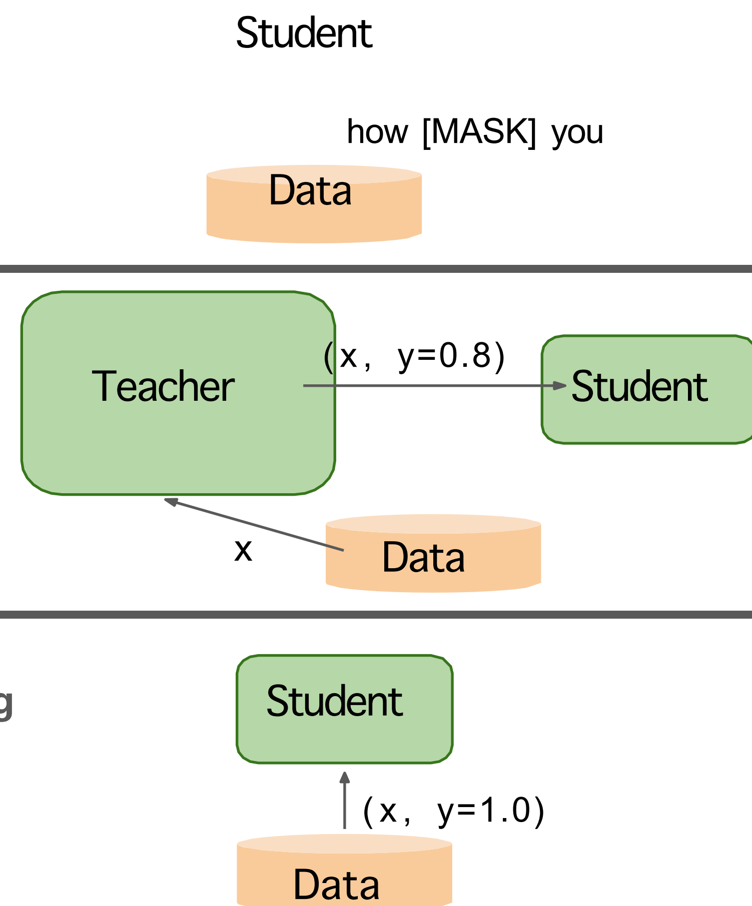
Turc et al., 2019

Well-Read Students Learn Better: On the Importance of Pre-training Compact Models

1 Regular Pre-training

2 Fine-tuning via distillation

3 (Optional) regular fine-tuning



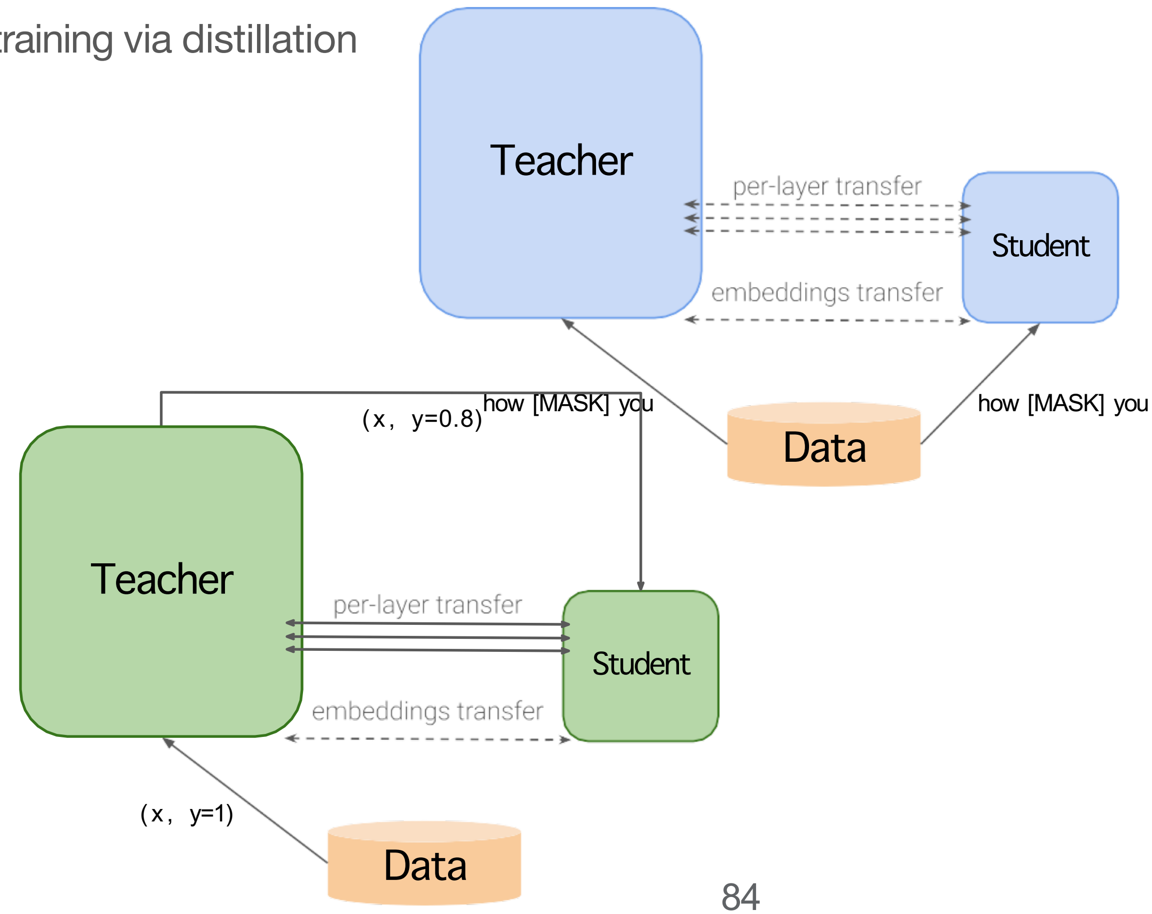
Knowledge Distillation for Fine-Tuning

Jiao et al., 2019

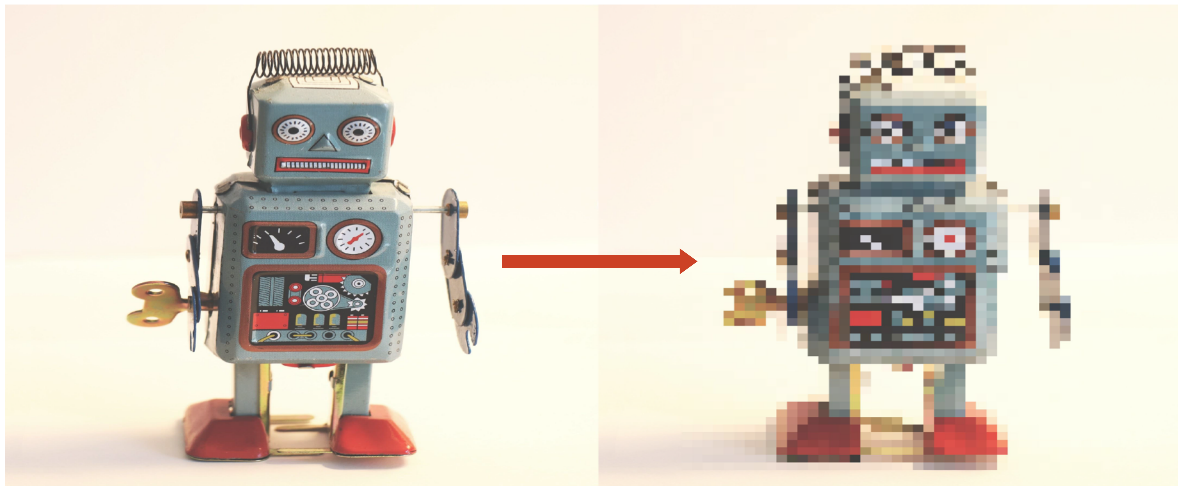
TinyBERT: Distilling BERT for Natural Language Understanding

1 Pre-training via distillation

2 Fine-training via distillation



Quantization



Source: unsplash.com

Quantization

Definition

$$Q(z) = q_j \qquad z \in (t_j, t_{j+1}] \qquad j=0, \ldots, 2^k-1$$

quantization operator

real-valued tensor (activation or weight)

quantization precision

Linear Quantization

$$z = S\,(q_j - Z)$$

scaling factor

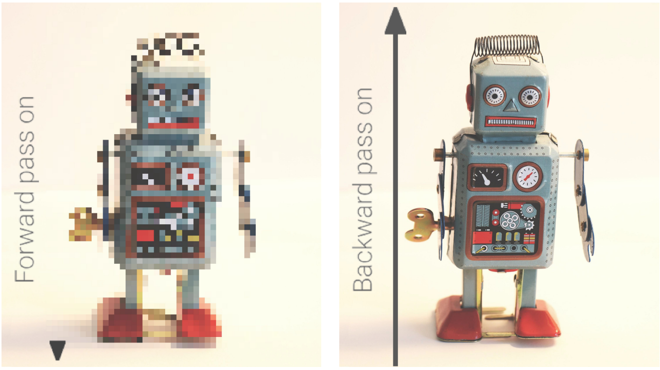
zero point

Quantization

Jacob et al., 2017

Quantization and Training of Neural Networks
for Efficient
Integer-Arithmetic-Only Inference

Quantization-Aware Training



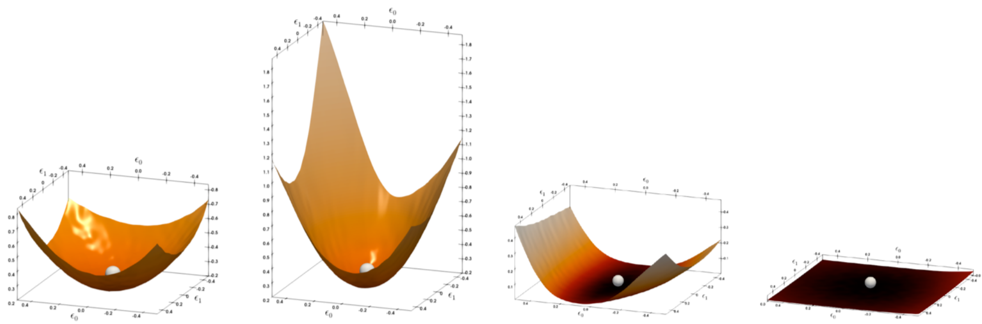
$$\boldsymbol{w}^{t+1} = \text{UpdateParameter}(\boldsymbol{w}^{t}, \frac{\partial L}{\partial \widehat{\boldsymbol{w}}^{t}}, \eta^{t})$$

Quantization

Shen et al., 2019

Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

- **Q-BERT**: uniform quantization to $\{0, \ldots, 2^k\text{-}1\}$ with:
  - mixed precision (higher Hessian spectrum => higher precision for layer)
  - group precision (each matrix W_k W_q W_v W_o is its own group)



(c) MNLI 10^{th} (d) MNLI 4^{th} (e) CoNLL-03 4^{th} (f) CoNLL-03 11^{th}

Quantization with Distillation

Zhang et al., 2020

TernaryBERT: Distillation-aware Ultra-low Bit BERT

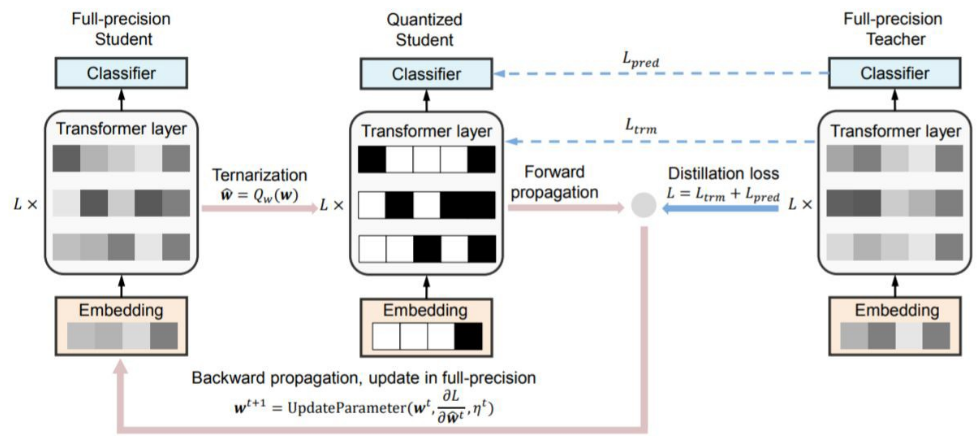


Figure 2: Depiction of the proposed distillation-aware ternarization of BERT model.

Pruning

Definition

Pruning removes "unimportant" weights from a network:

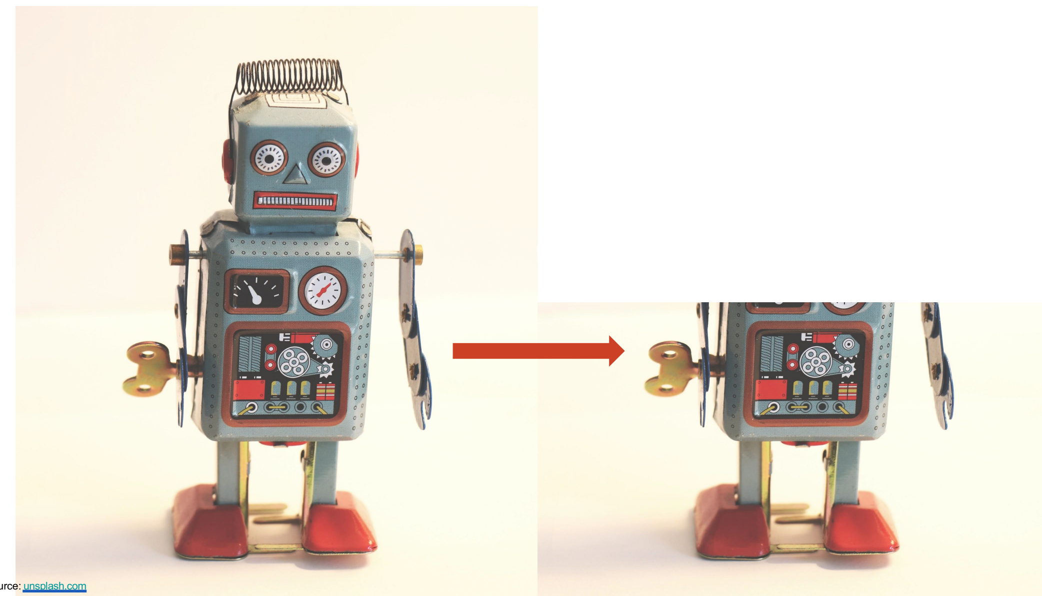
$$a = (W \odot M)\, x$$

input

pruning mask

model weight

activation



Source: unsplash.com

Main Questions (Hassibi and Stork)

- Which weights should be eliminated?
- How should the remaining weights be adjusted?
- How can such network pruning be done in an efficient way?

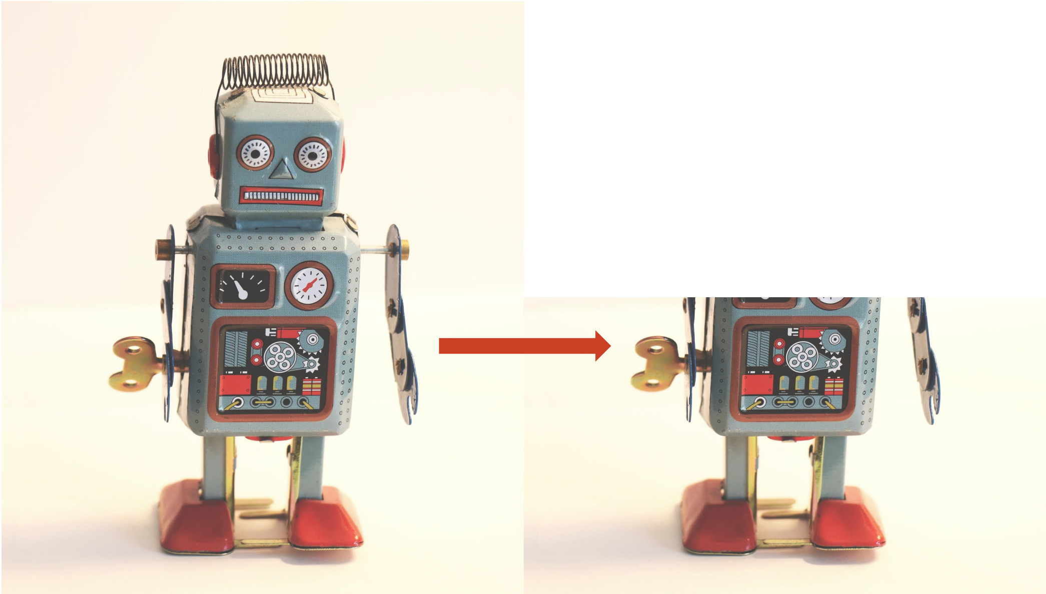
Pruning

LeCun et al., 1990

OBD: Optimal Brain Damage

Hassibi and Stork, 1993

OBS: Second order derivatives for network pruning: Optimal Brain Surgeon



Main idea:

- Start with a "reasonably large" network
- Train it to convergence
- Prune in multiple iterations, based on second-order derivatives:
    - OBD: prune and train
    - OBS: prune and update weights based on second-order statistics

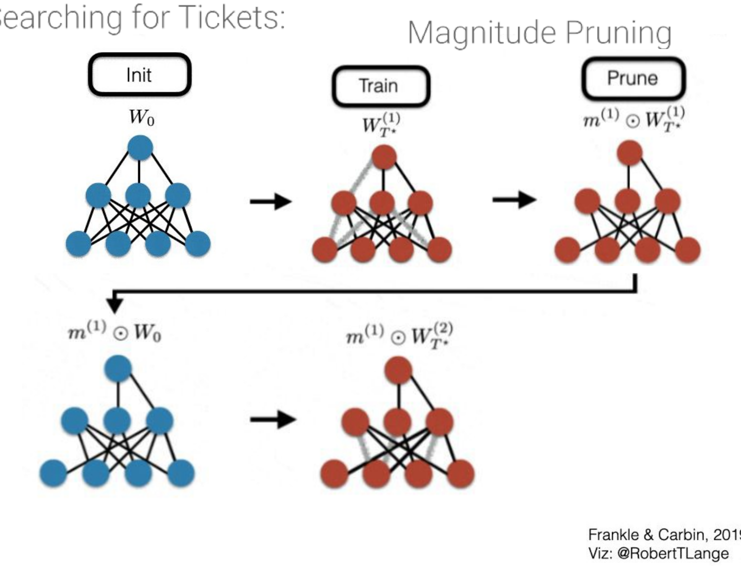
Lottery Ticket Hypothesis

Frankle and Carbin, 2018

[The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks]

The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

Searching for Tickets:



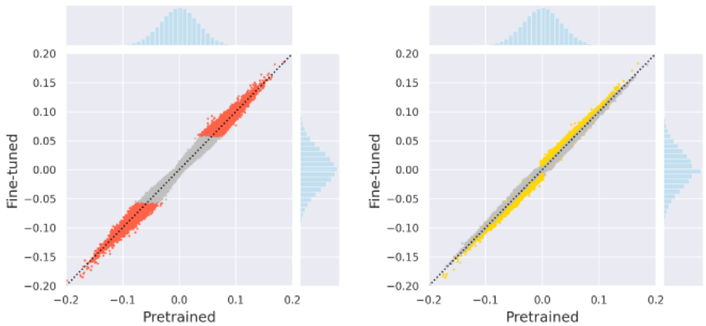
Source: https://roberttlange.github.io/posts/2020/06/lottery-ticket-hypothesis/

Movement Pruning

Sanh et al., 2020

Movement Pruning: Adaptive Sparsity by Fine-Tuning

- **First-order** strategy: "instead of selecting weights that are far from zero, we retain connections that are moving away from zero during the training process"
- The pruning mask $\mathbf{M}$ is learnt together with the model parameters.
    - **hard** version: $\mathbf{M} = \mathrm{Top}_v(\mathbf{S})$, where score $\mathbf{S}$ is learnt and $\mathbf{v}$ is a hyperparameter.
    - **soft** version: $\mathbf{M} = (\mathbf{S} > \tau)$, where score $\mathbf{S}$ is learnt and threshold τ is a hyperparameter.



(a) Magnitude pruning (b) Movement pruning

Movement Pruning

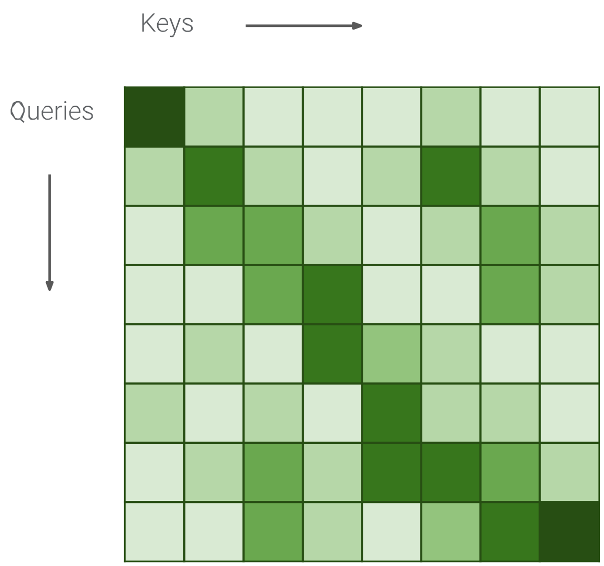
	Unstructured Pruning	Structured Pruning
Storage	✓	✓
Inference	❌	✓
Flexibility	✓	❌

Efficient Attention

Goal:

Approximate the computation of attention

via more efficient operations



Efficient Attention

- Data-Independent
- Data-Dependent
- Kernels
- Recurrence
- I/O Aware-Attention

Data-Independent Patterns

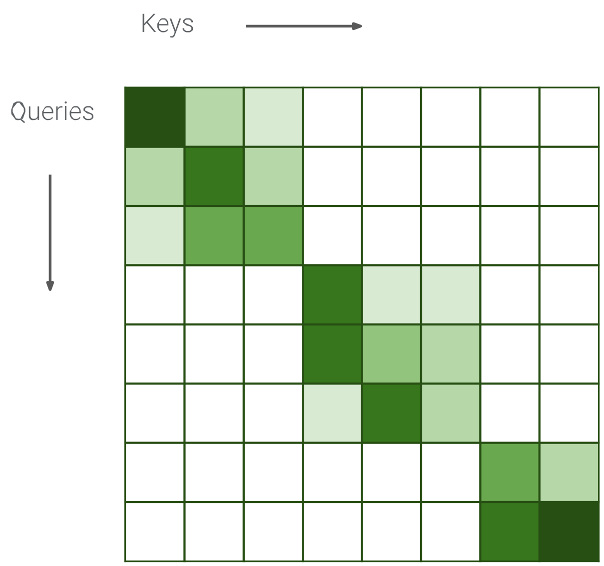
Blockwise Patterns

Divide sequence into local blocks and restrict attention within them

Examples:

Blockwise Transformer (Qiu et al., 2019)

Local Attention (Parmar et al., 2018)



Data-Independent Patterns

Strided Patterns

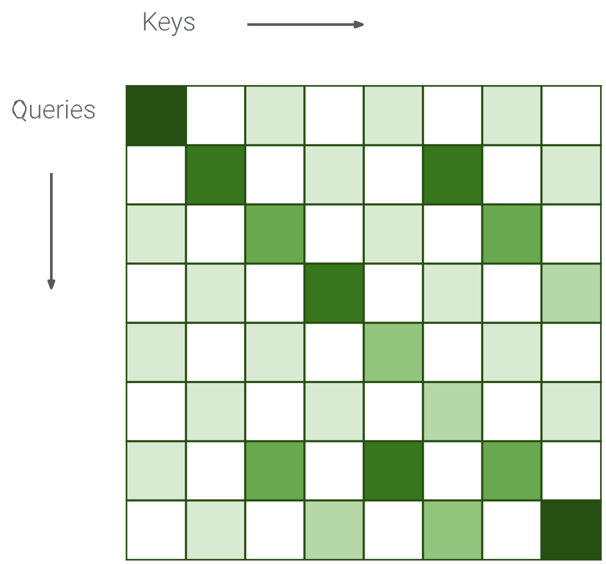
Skip some query/key pairs.

Quadratic in sequence length / stride

Examples:

Sparse Transformer (Child et al., 2019)

Longformer (Beltagy et al, 2020)



Data-Independent Patterns

Diagonal Patterns

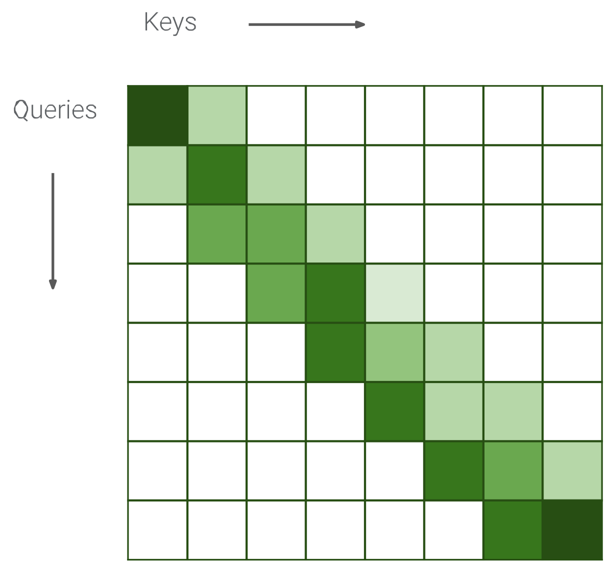
Compute attention over the diagonal.

Linear in sequence length and window size.

Examples:

Longformer (Beltagy et al, 2020)

Big Bird (Zaheer et al., 2020)



Data-Independent Patterns

Global Attention

Applied to one or a few special tokens, often prepended to the sequence.

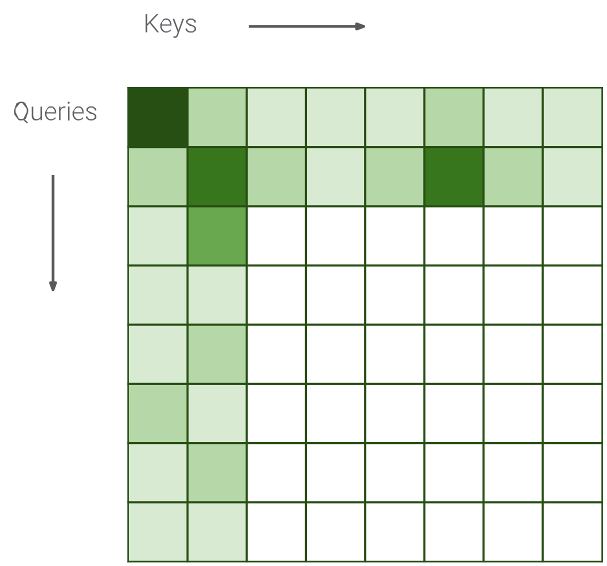
Usually combined with other patterns

Examples:

Big Bird (Zaheer et al., 2020)

Longformer (Beltagy et al., 2020)

ETC (Ainslie et al., 2020)



Data-Dependent Patterns

Buckets

Create buckets/clusters and compute attention within.

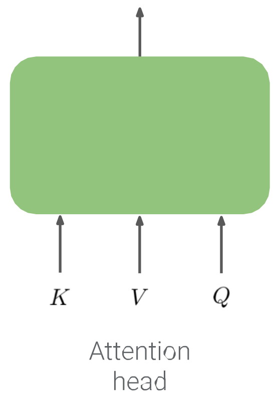
Ideally, buckets should contain the highests attention weights in the matrix

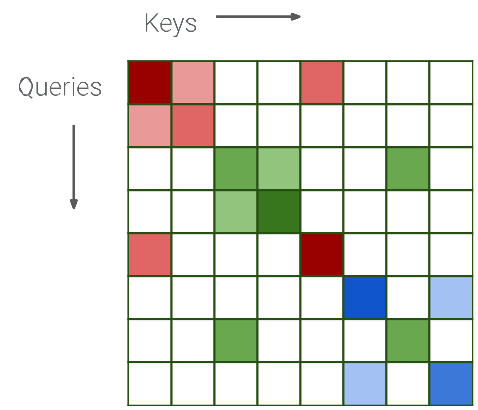
Examples:

Reformer (Kitaev et al., 2020)

Routing Transformer (Roy et al., 2020)

$$O_i = \sum_{j=0}^{l} a_{ij} V_j$$





Data-Dependent Patterns

Buckets: Hashing

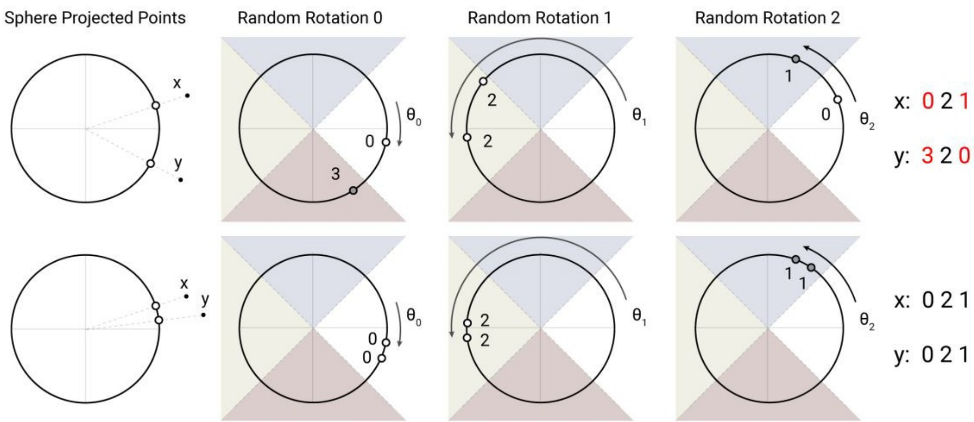
[Locality-Sensitive Hashing (LSH)](#)

Key idea: take a random projection matrix R, compute hash for a vector x through:

$$h(x) = \arg\max([xR; -xR])$$

Examples:

Reformer ([Kitaev et al., 2020](#))



Data-Dependent Patterns

Compression

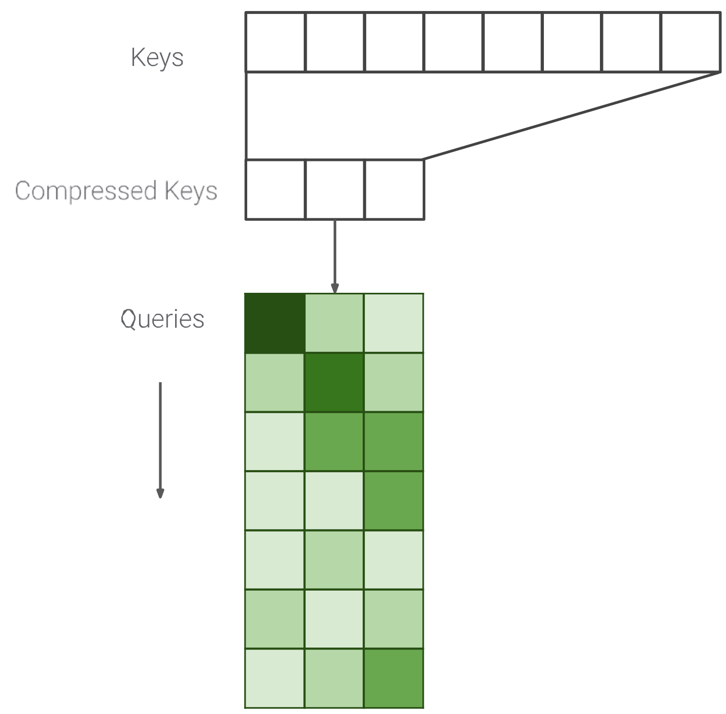
E.g. pooling, strided convolution, low-rank projections with learnable weights

Examples:

Compressed Attention (Liu et al., 2018)

Linformer (Wang et al., 2020)

Synthesizers (Tay et al., 2020)



Kernel

Kernels

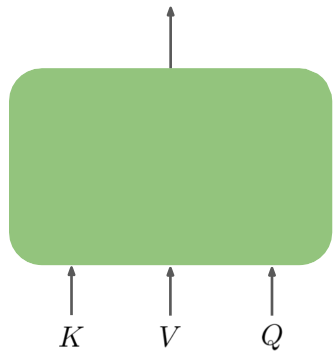
Recap attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

However, we can simplify things with a decomposable **kernel:**

$$\phi(Q_i, K_j) = \phi(Q_i)^\top \phi(K_j)$$

$$O_i = \sum_{j=0}^{l} a_{ij} V_j$$



$$a_{ij} = \frac{\phi(Q_i, K_j)}{\sum_{p=0}^{l} \phi(Q_i, K_p)}$$

Attention head

Kernel

Kernels

Recap attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

However, we can simplify things with a decomposable **kernel:**

$$\phi(Q_i, K_j) = \phi(Q_i)^\top \phi(K_j)$$

$$O_i = \sum_{j=0}^{l} a_{ij} V_j \qquad a_{ij} = \frac{\phi(Q_i, K_j)}{\sum_{p=0}^{l} \phi(Q_i, K_p)}$$

$$O_i = \frac{\sum_{j=0}^{l} \phi(Q_i)^\top \phi(K_j) V_j}{\sum_{j=0}^{l} \phi(Q_i)^\top \phi(K_j)}$$

Independent of query!

$$O_i = \frac{\phi(Q_i)^\top \boxed{\sum_{j=0}^{l} \phi(K_j) V_j}}{\phi(Q_i)^\top \boxed{\sum_{j=0}^{l} \phi(K_j)}}$$

Kernel

Kernels

Recap attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

However, we can simplify things with a decomposable **kernel:**

$$\phi(Q_i, K_j) = \phi(Q_i)^\top \phi(K_j)$$

In vectorized form:

$$O = \phi(Q)\boxed{\phi(K)^\top V}$$

Compute this d' x d matrix first

This allows us to compute attention in **linear** time with respect to sequence length!

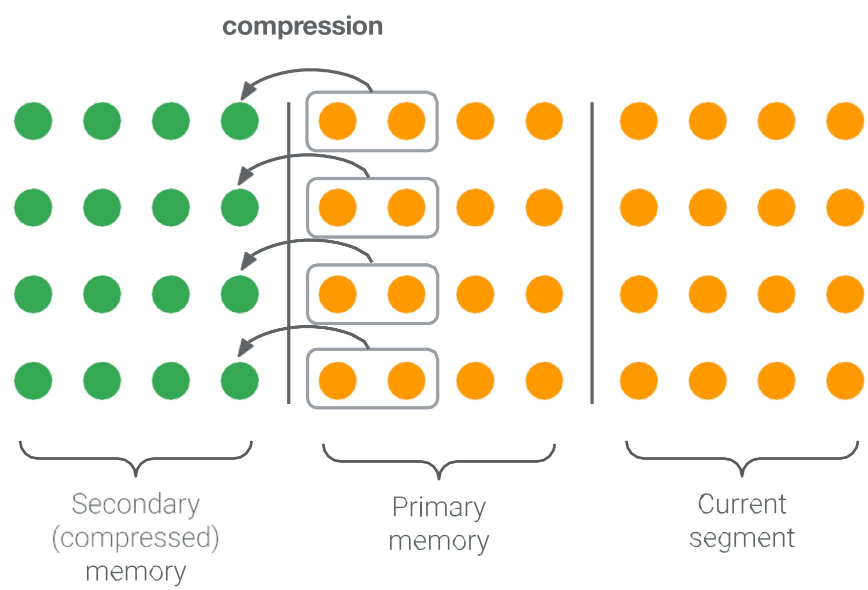
In [Katharopoulos et al., 2020]:

Recurrence

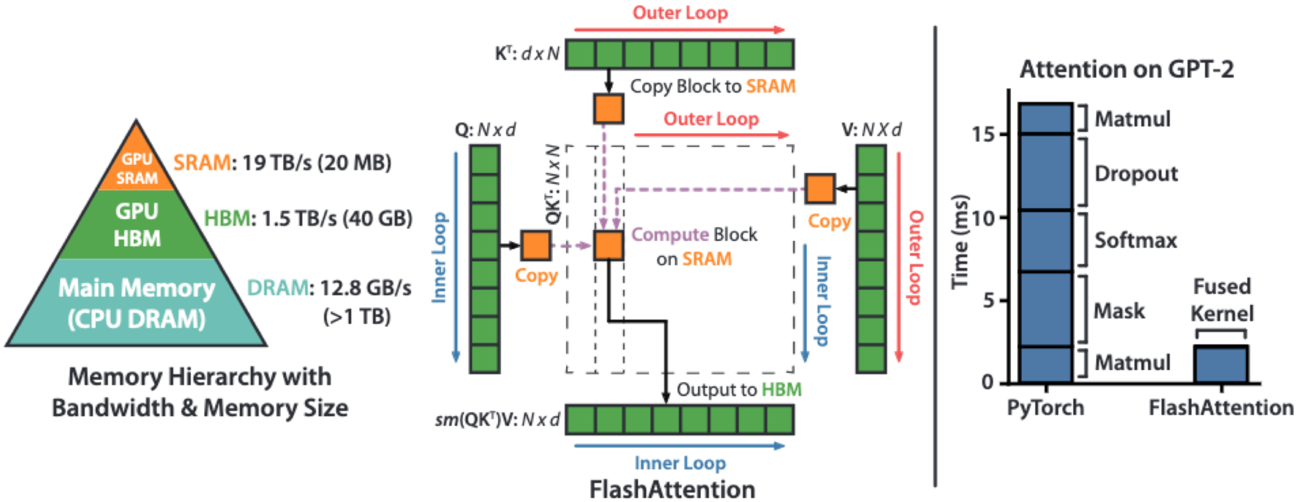
Compressive Transformers
([Rae et al., 2019](Rae et al., 2019))

Dual memory system:

- Primary mem. contains activations from previous segment
- Secondary mem. is compresses activations from all previous segments

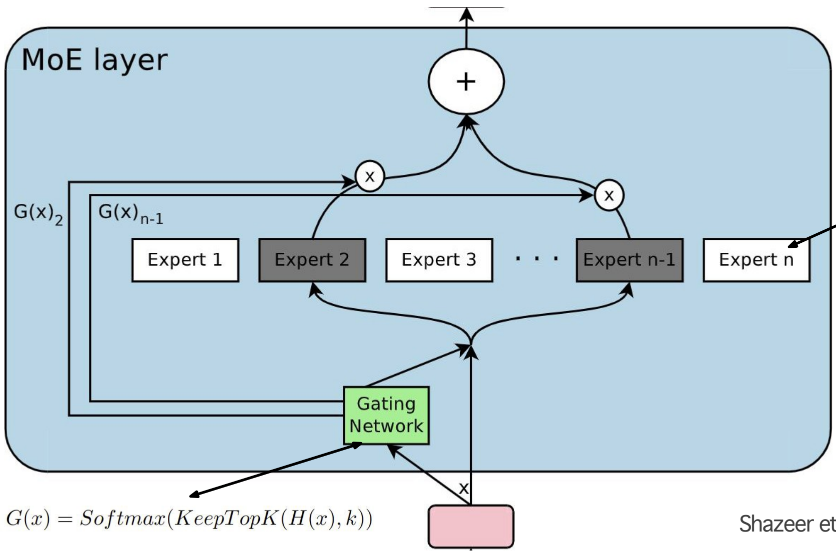


I/O Awareness



FlashAttention (Dao et al., 2019

Mixture-of-Experts



$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

$$G(x) = Softmax(KeepTopK(H(x), k))$$

Shazeer et al., 2017: arXiv

Lepikhin et al., 2020: arXiv

Mixture-of-Experts

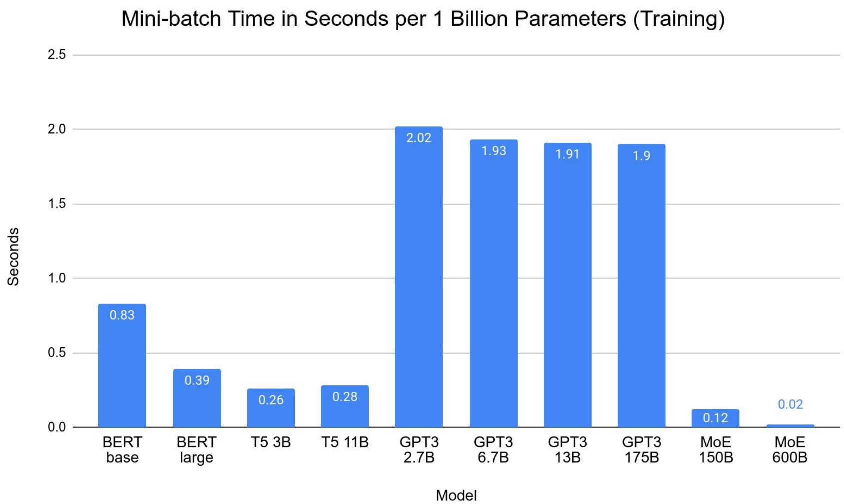
Version 2 (Lepikhin et al., 2020):

$$\mathcal{L} = \ell_{nll} + k * \ell_{aux}$$

Where k is a constant loss weight (a good value is 0.1; usually between 0.01 and 1.0)

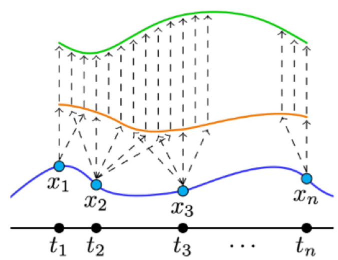
- Random dispatch: Use 2nd expert proportionally to the softmax gate probability.
- Have a frequency cutoff — a token budget — for each expert. If this budget is exceeded the expert degenerated to a zero matrix. This effectively reduces the output of the MoE layer to zero and thus only the residual connection output around the MoE layer is fed to the next layer.

Mixture-of-Experts



- Works well on diverse data like multilingual machine translation
- Can be difficult to train due to balancing/specialization issues
- Only faster than transformers if you can run it with a large enough batch size to saturate distributed experts
- If you scale the model across a cluster, you will need excellent interconnect performance (TPU v4 Pod, NVIDIA SuperPod)

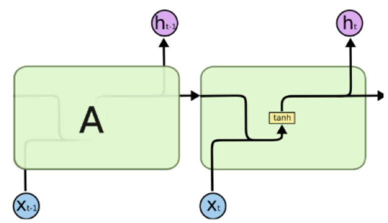
Structure State Spaces



✓ Continuous data
Irregular sampling

✗ Complex, very inefficient
Vanishing gradients

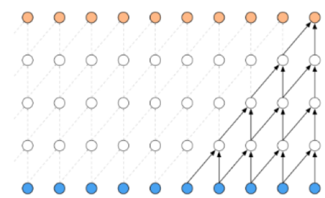
Continuous-time (CTM)



✓ Unbounded context
Stateful inference

✗ Inefficient training
Vanishing gradients

Recurrent (RNN)



✓ Easy optimization
Parallelizable training

✗ Inefficient inference
Bounded context

Convolutional (CNN)

Existing model families have clear tradeoffs
All struggle with long-range dependencies (LRD)

S4 (Gu et al., 2022)

Structure State Spaces