High Performance Natural Language Processing

EMNLP 2020

Google Research

UNIVERSITY of WASHINGTON

UWNLP

Presenters

Gabriel Ilharco

University of Washington

Cesar Ilharco

Google

Iulia Turc

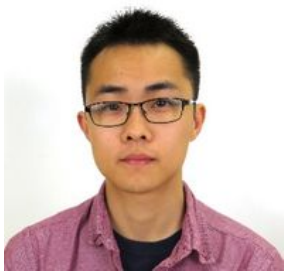
Google

Tim Dettmers

University of Washington

Felipe Ferreira

Google



Kenton Lee

Google

High Performance NLP

Slides available at: bit.ly/2SmhKY7

Google Research

W
UNIVERSITY of
WASHINGTON

UWNLP

Agenda

01 Introduction

02 Fundamentals

03 Core Techniques

04 Efficient Attention

05 Case Studies

06 Scaling in Practice

07 Closing Notes

01

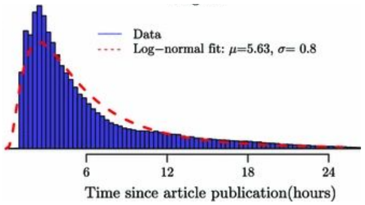
Introduction

Motivation & Applications

Why do we need it ?

SCALE

- NEWS
  - Realtime: Majority of content is consumed within a few hours after publication [1]
  - Thousands of news articles per second
  - 40-80 sentences per article
- SOCIAL NETWORKS: ~6 Thousand tweets per second [2]
- THE WEB: Orders of magnitude bigger

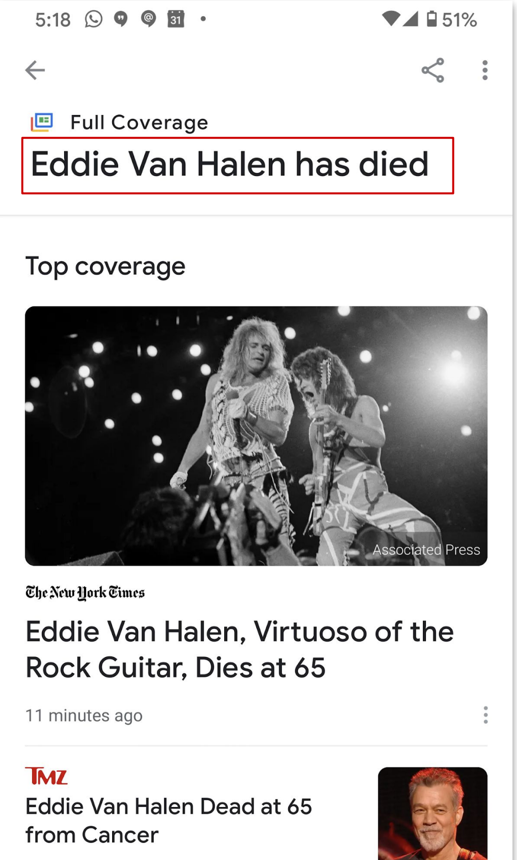


What could we do, if we had it ?

[1] Tatar, A., Antoniadis, P., Amorim, M.D.d. et al. From popularity prediction to ranking online news. Soc. Netw. Anal. Min. 4, 174 (2014). https://doi.org/10.1007/s13278-014-0174-8

[2] https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html

Summarization



5:18 51%

Full Coverage

Eddie Van Halen has died

Top coverage

Associated Press

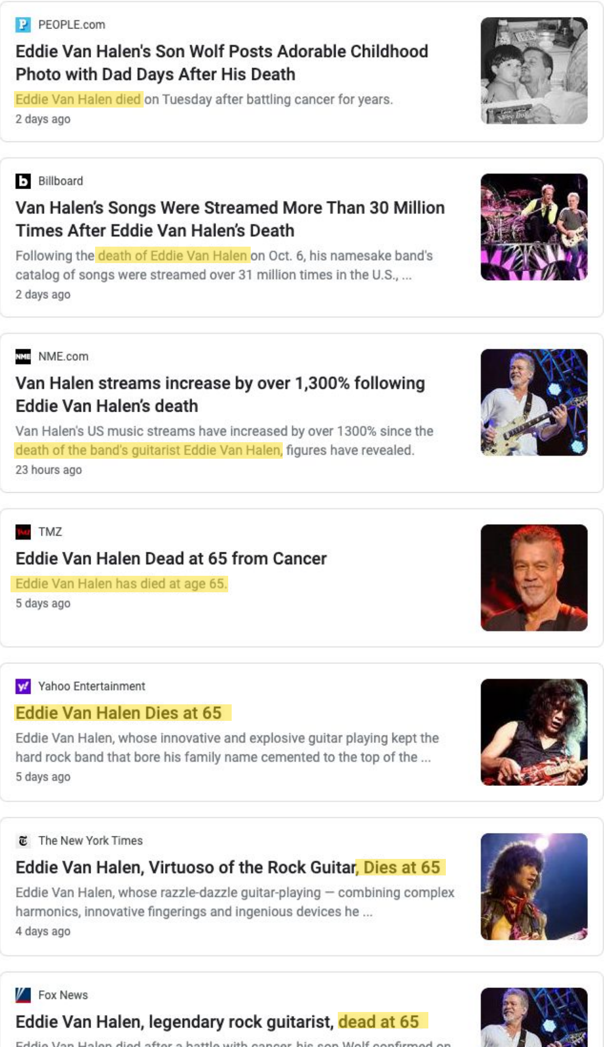
The New York Times

Eddie Van Halen, Virtuoso of the Rock Guitar, Dies at 65

11 minutes ago

TMZ

Eddie Van Halen Dead at 65 from Cancer



PEOPLE.com

Eddie Van Halen's Son Wolf Posts Adorable Childhood Photo with Dad Days After His Death

Eddie Van Halen died on Tuesday after battling cancer for years.

2 days ago

Billboard

Van Halen's Songs Were Streamed More Than 30 Million Times After Eddie Van Halen's Death

Following the death of Eddie Van Halen on Oct. 6, his namesake band's catalog of songs were streamed over 31 million times in the U.S., ...

2 days ago

NME.com

Van Halen streams increase by over 1,300% following Eddie Van Halen's death

Van Halen's US music streams have increased by over 1300% since the death of the band's guitarist Eddie Van Halen, figures have revealed.

23 hours ago

TMZ

Eddie Van Halen Dead at 65 from Cancer

Eddie Van Halen has died at age 65.

5 days ago

Yahoo Entertainment

Eddie Van Halen Dies at 65

Eddie Van Halen, whose innovative and explosive guitar playing kept the hard rock band that bore his family name cemented to the top of the ...

5 days ago

The New York Times

Eddie Van Halen, Virtuoso of the Rock Guitar, Dies at 65

Eddie Van Halen, whose razzle-dazzle guitar-playing — combining complex harmonics, innovative fingerings and ingenious devices he ...

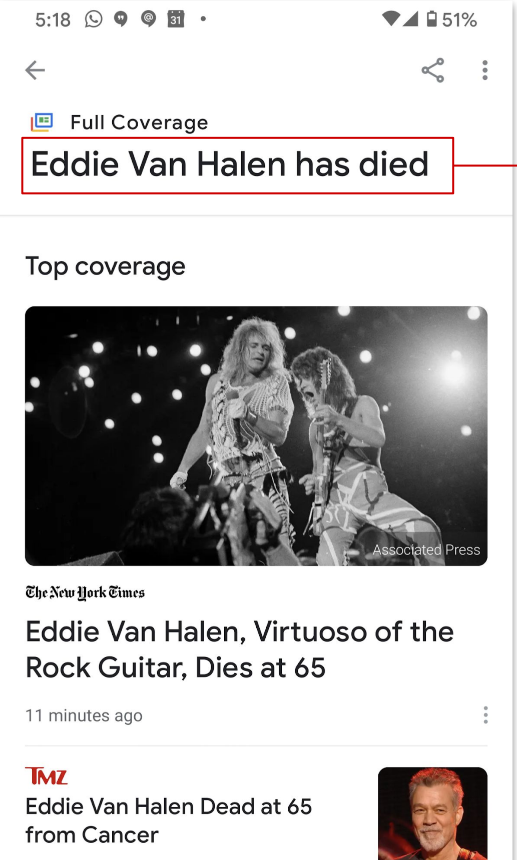
4 days ago

Fox News

Eddie Van Halen, legendary rock guitarist, dead at 65

Eddie Van Halen died after a battle with cancer, his son Wolf confirmed on

Summarization



5:18 51%

Full Coverage

Eddie Van Halen has died

Top coverage

Associated Press

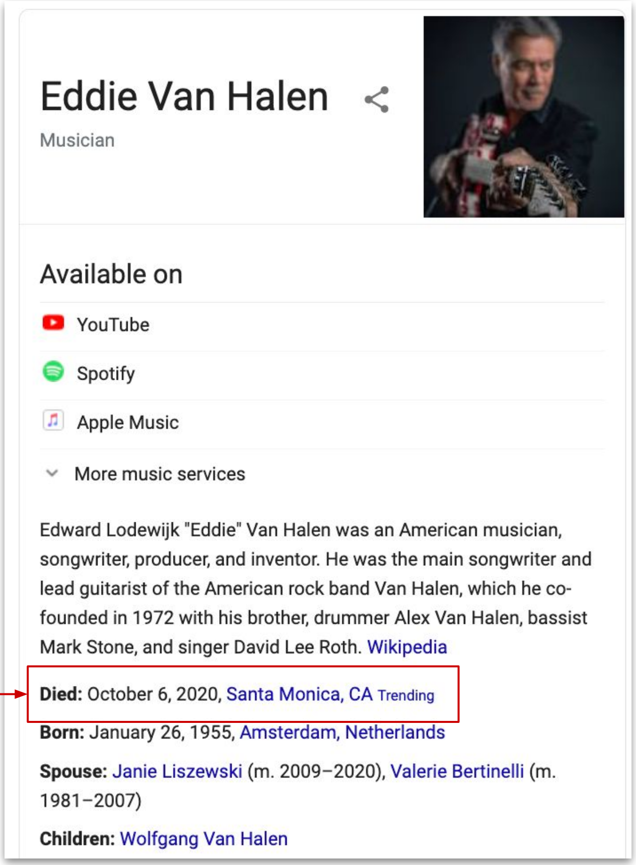
The New York Times

Eddie Van Halen, Virtuoso of the Rock Guitar, Dies at 65

11 minutes ago

TMZ

Eddie Van Halen Dead at 65 from Cancer



Eddie Van Halen

Musician

Available on

- YouTube
- Spotify
- Apple Music
- More music services

Edward Lodewijk "Eddie" Van Halen was an American musician, songwriter, producer, and inventor. He was the main songwriter and lead guitarist of the American rock band Van Halen, which he co-founded in 1972 with his brother, drummer Alex Van Halen, bassist Mark Stone, and singer David Lee Roth. Wikipedia

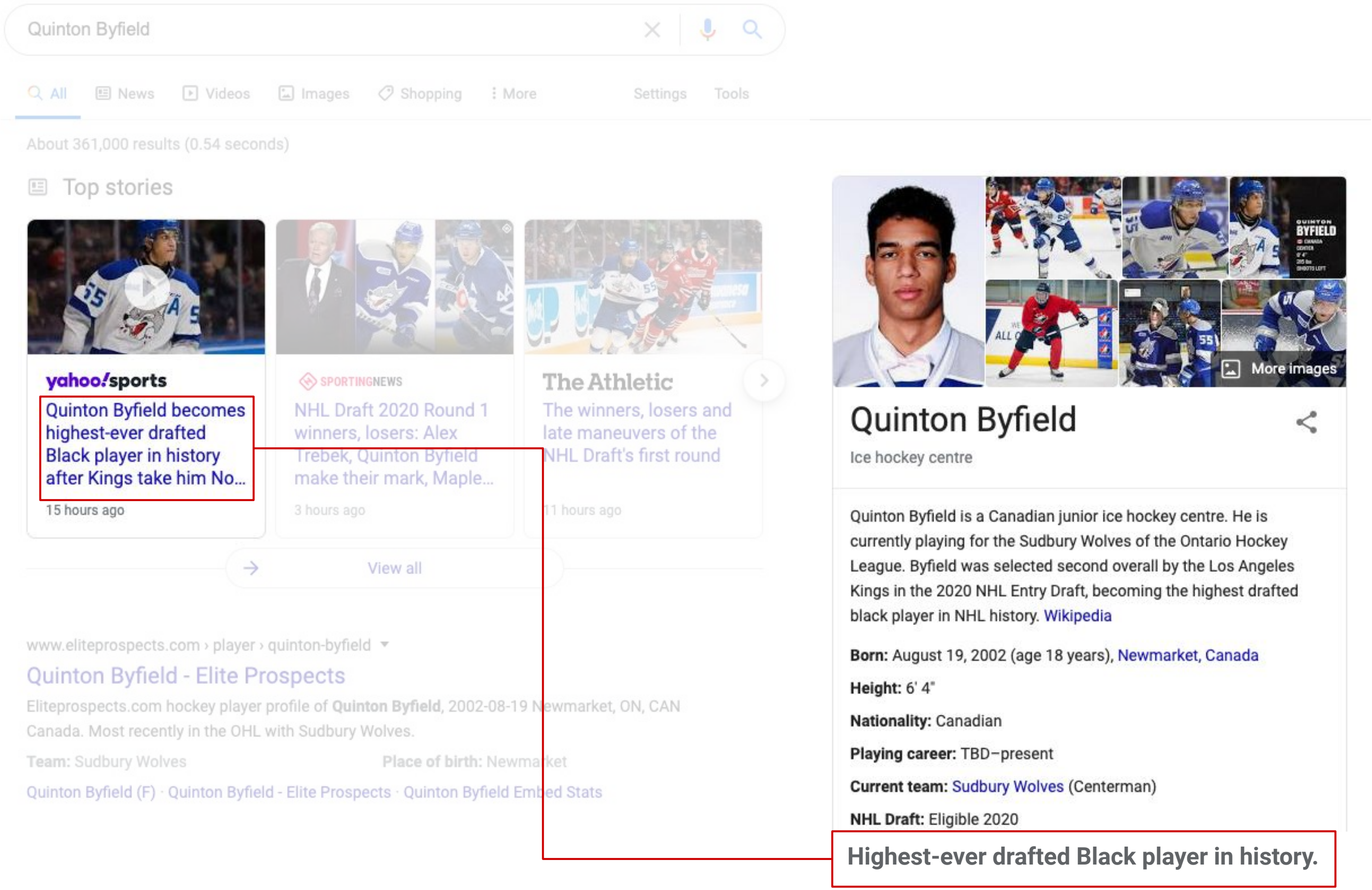
Died: October 6, 2020, Santa Monica, CA Trending

Born: January 26, 1955, Amsterdam, Netherlands

Spouse: Janie Liszewski (m. 2009–2020), Valerie Bertinelli (m. 1981–2007)

Children: Wolfgang Van Halen

Facts Extraction



About 361,000 results (0.54 seconds)

Top stories

yahoo!sports
Quinton Byfield becomes highest-ever drafted Black player in history after Kings take him No...
15 hours ago

SPORTINGNEWS
NHL Draft 2020 Round 1 winners, losers: Alex Trebek, Quinton Byfield make their mark, Maple...
3 hours ago

The Athletic
The winners, losers and late maneuvers of the NHL Draft's first round
11 hours ago

View all

www.eliteprospects.com › player › quinton-byfield
Quinton Byfield - Elite Prospects
Eliteprospects.com hockey player profile of **Quinton Byfield**, 2002-08-19 Newmarket, ON, CAN Canada. Most recently in the OHL with Sudbury Wolves.
Team: Sudbury Wolves **Place of birth:** Newmarket
Quinton Byfield (F) · Quinton Byfield - Elite Prospects · Quinton Byfield Embed Stats

Quinton Byfield
Ice hockey centre

Quinton Byfield is a Canadian junior ice hockey centre. He is currently playing for the Sudbury Wolves of the Ontario Hockey League. Byfield was selected second overall by the Los Angeles Kings in the 2020 NHL Entry Draft, becoming the highest drafted black player in NHL history. Wikipedia

Born: August 19, 2002 (age 18 years), Newmarket, Canada

Height: 6′ 4″

Nationality: Canadian

Playing career: TBD–present

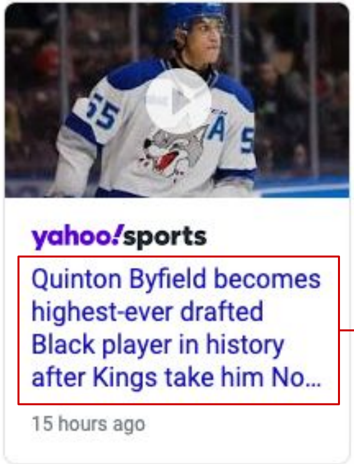
Current team: Sudbury Wolves (Centerman)

NHL Draft: Eligible 2020

Highest-ever drafted Black player in history.

Facts Extraction

- Noteworthy Facts
- Trendiness

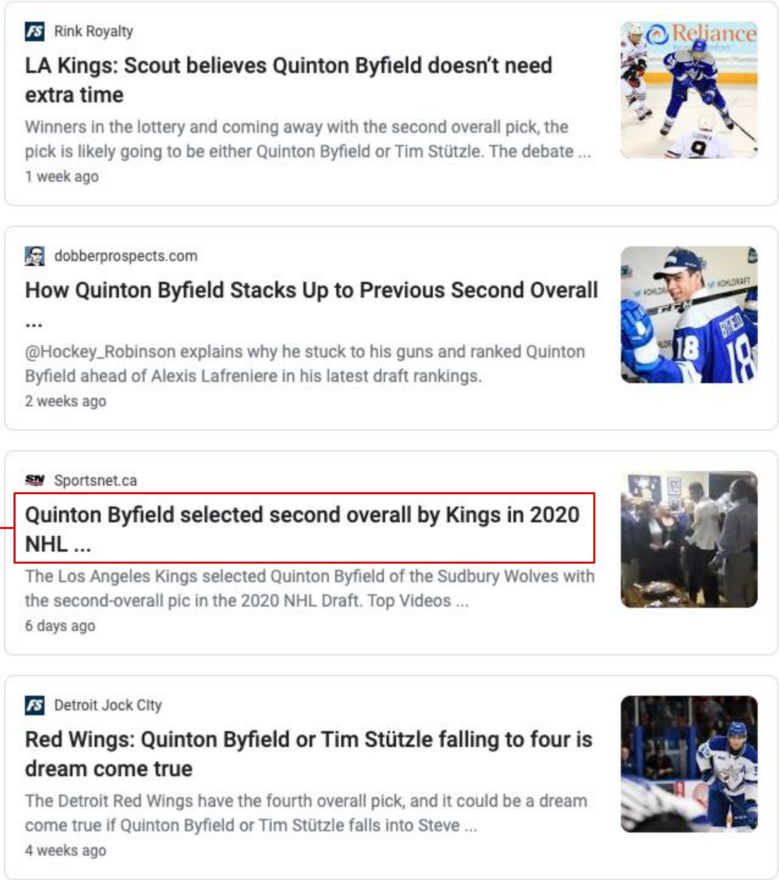


yahoo!sports

Quinton Byfield becomes highest-ever drafted Black player in history after Kings take him No...

15 hours ago

VS



Rink Royalty

LA Kings: Scout believes Quinton Byfield doesn't need extra time

Winners in the lottery and coming away with the second overall pick, the pick is likely going to be either Quinton Byfield or Tim Stützle. The debate ...

1 week ago

dobberprospects.com

How Quinton Byfield Stacks Up to Previous Second Overall ...

@Hockey_Robinson explains why he stuck to his guns and ranked Quinton Byfield ahead of Alexis Lafreniere in his latest draft rankings.

2 weeks ago

Sportsnet.ca

Quinton Byfield selected second overall by Kings in 2020 NHL ...

The Los Angeles Kings selected Quinton Byfield of the Sudbury Wolves with the second-overall pic in the 2020 NHL Draft. Top Videos ...

6 days ago

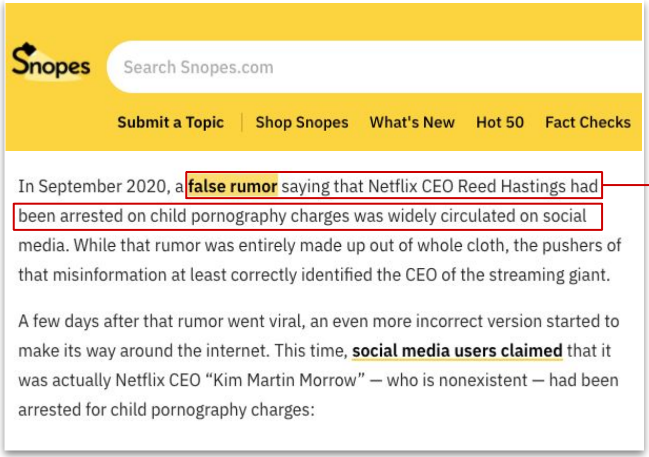
Detroit Jock City

Red Wings: Quinton Byfield or Tim Stützle falling to four is dream come true

The Detroit Red Wings have the fourth overall pick, and it could be a dream come true if Quinton Byfield or Tim Stützle falls into Steve ...

4 weeks ago

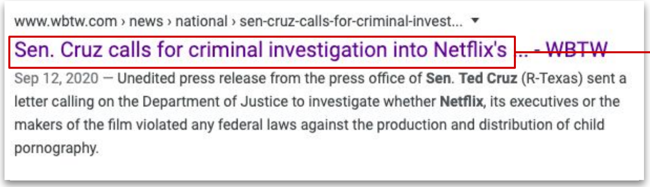
Sentence Entailment



In September 2020, a **false rumor** saying that Netflix CEO Reed Hastings had been arrested on child pornography charges was widely circulated on social media. While that rumor was entirely made up out of whole cloth, the pushers of that misinformation at least correctly identified the CEO of the streaming giant.

A few days after that rumor went viral, an even more incorrect version started to make its way around the internet. This time, **social media users claimed** that it was actually Netflix CEO "Kim Martin Morrow" — who is nonexistent — had been arrested for child pornography charges:

False
About this rating

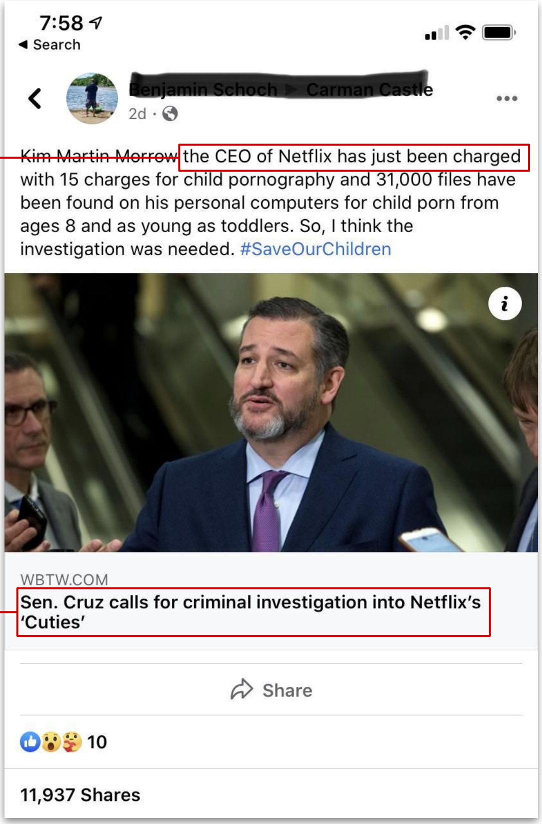


www.wbtw.com › news › national › sen-cruz-calls-for-criminal-invest... ▾

Sen. Cruz calls for criminal investigation into Netflix's ... - WBTW

Sep 12, 2020 — Unedited press release from the press office of **Sen. Ted Cruz** (R-Texas) sent a letter calling on the Department of Justice to investigate whether **Netflix**, its executives or the makers of the film violated any federal laws against the production and distribution of child pornography.

Correct Attribution
About this rating



[redacted]
2d ·

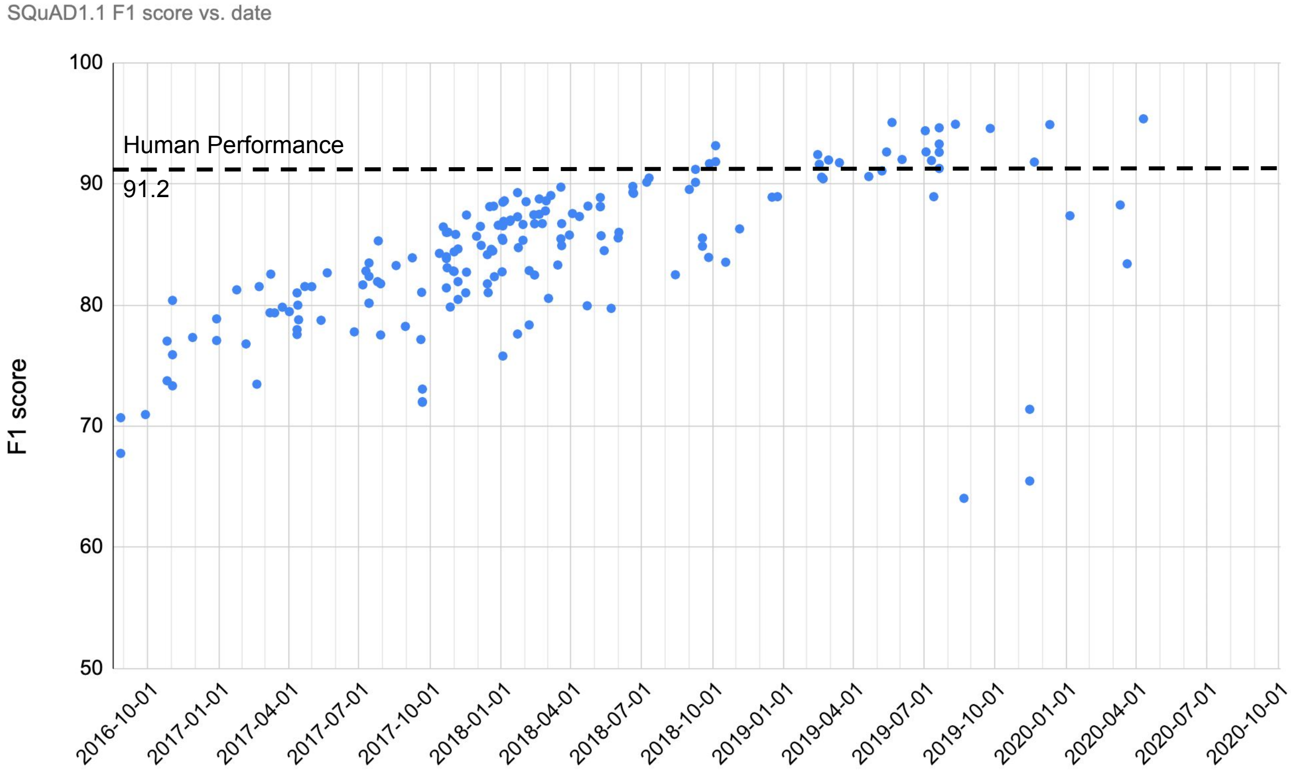
~~Kim Martin Morrow~~ the CEO of Netflix has just been charged with 15 charges for child pornography and 31,000 files have been found on his personal computers for child porn from ages 8 and as young as toddlers. So, I think the investigation was needed. #SaveOurChildren

WBTW.COM
Sen. Cruz calls for criminal investigation into Netflix's 'Cuties'

11,937 Shares

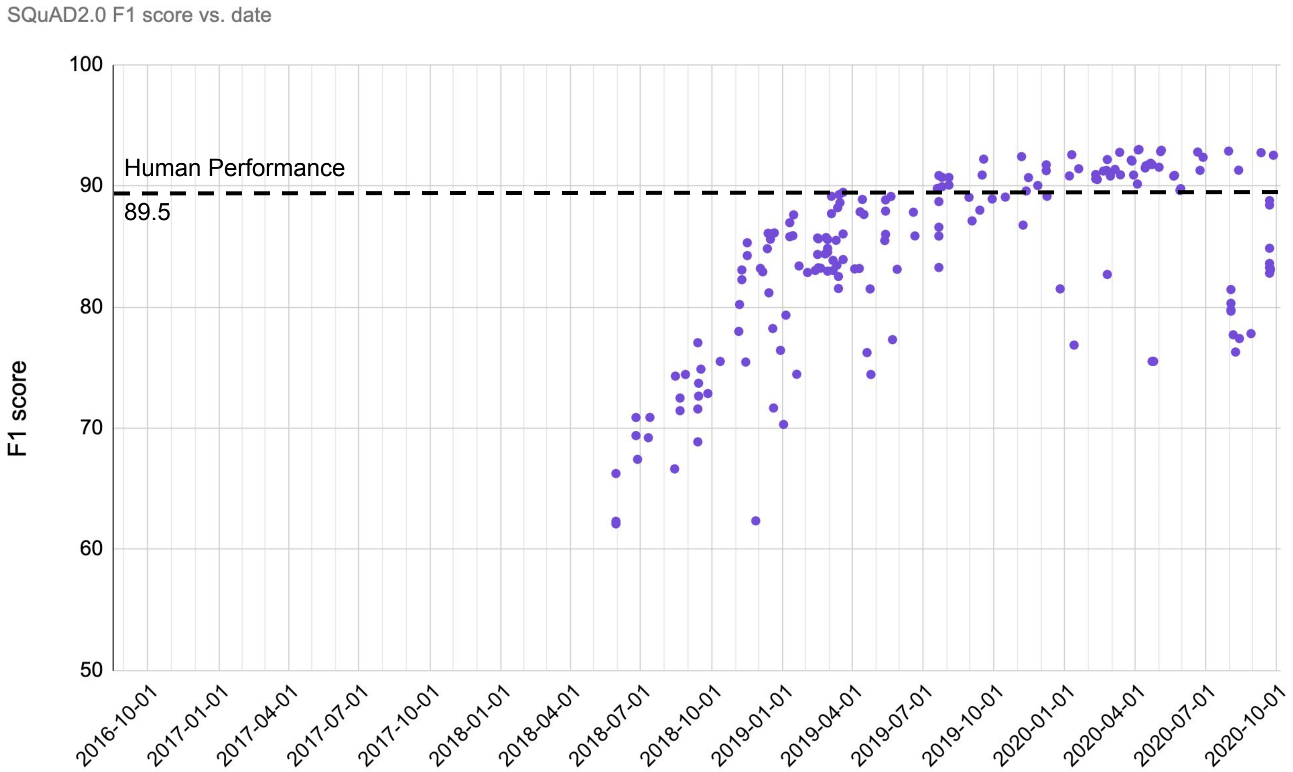
Recent years in Natural Language Processing

Benchmarks through the years - SQuAD 1.1



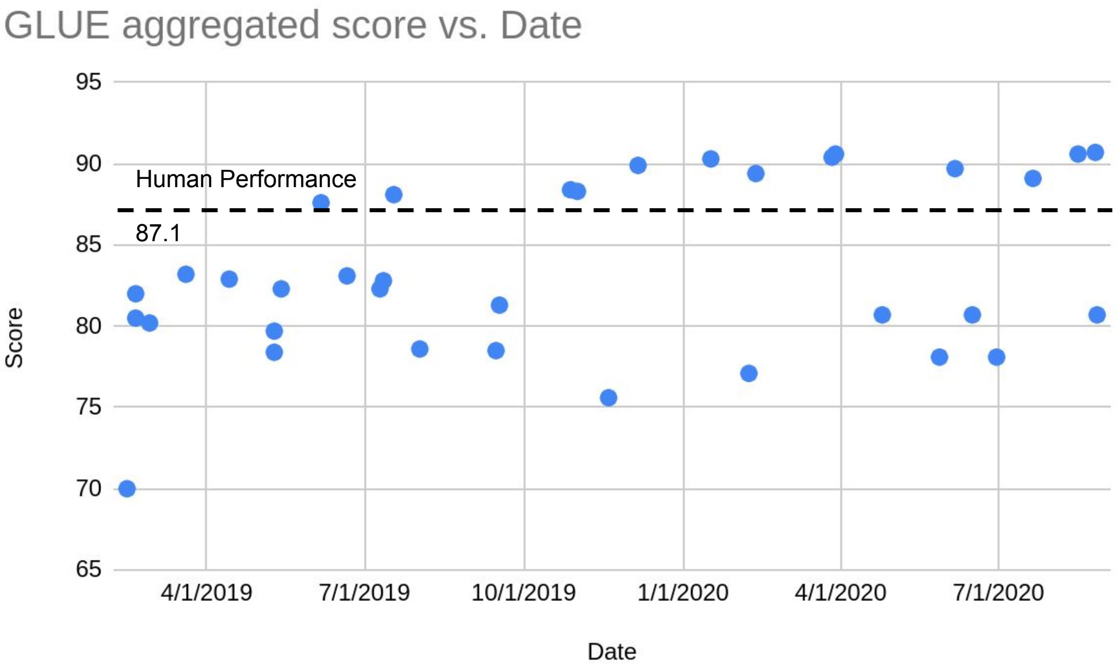
The Stanford Question Answering Dataset, https://rajpurkar.github.io/SQuAD-explorer/

Benchmarks through the years - SQuAD 2.0



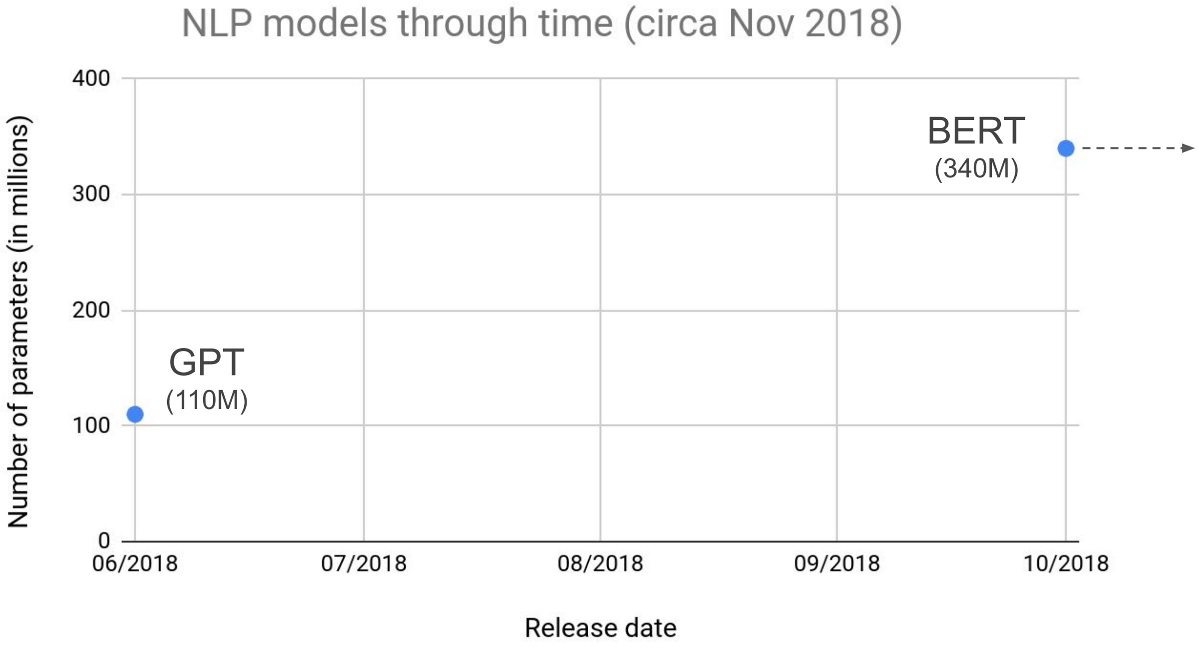
The Stanford Question Answering Dataset, https://rajpurkar.github.io/SQuAD-explorer/

Benchmarks through the years - GLUE



The GLUE Benchmark (Wang et al., 2018)

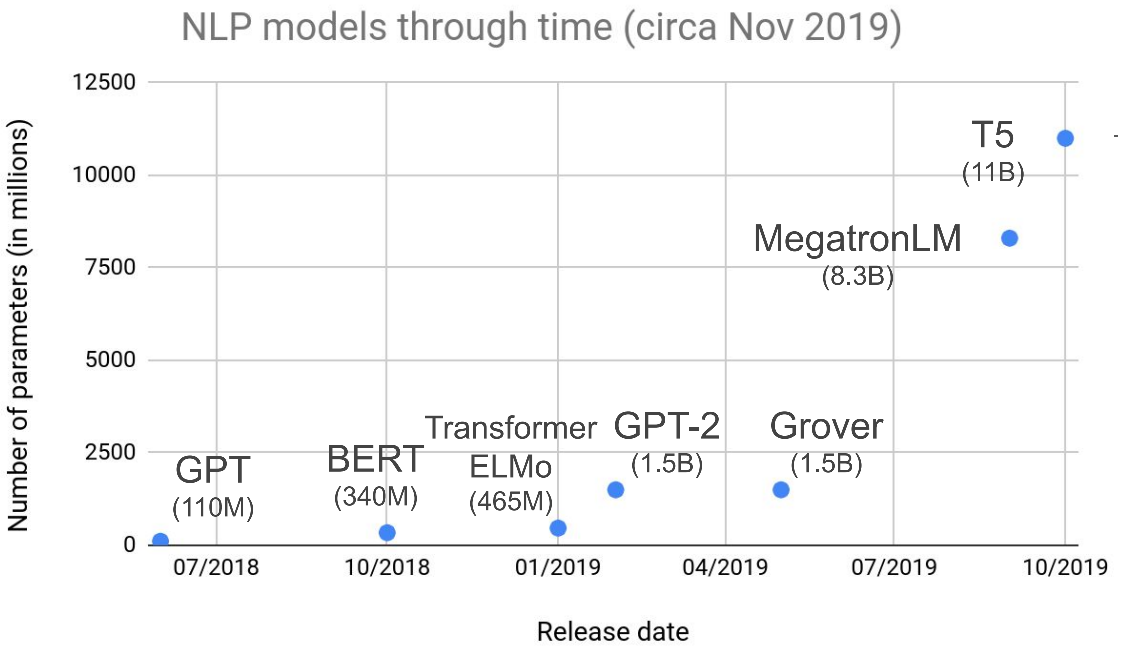
A brief recent history of scale in NLP



"[...] scaling to **extreme** model sizes also leads to large improvements [...]"

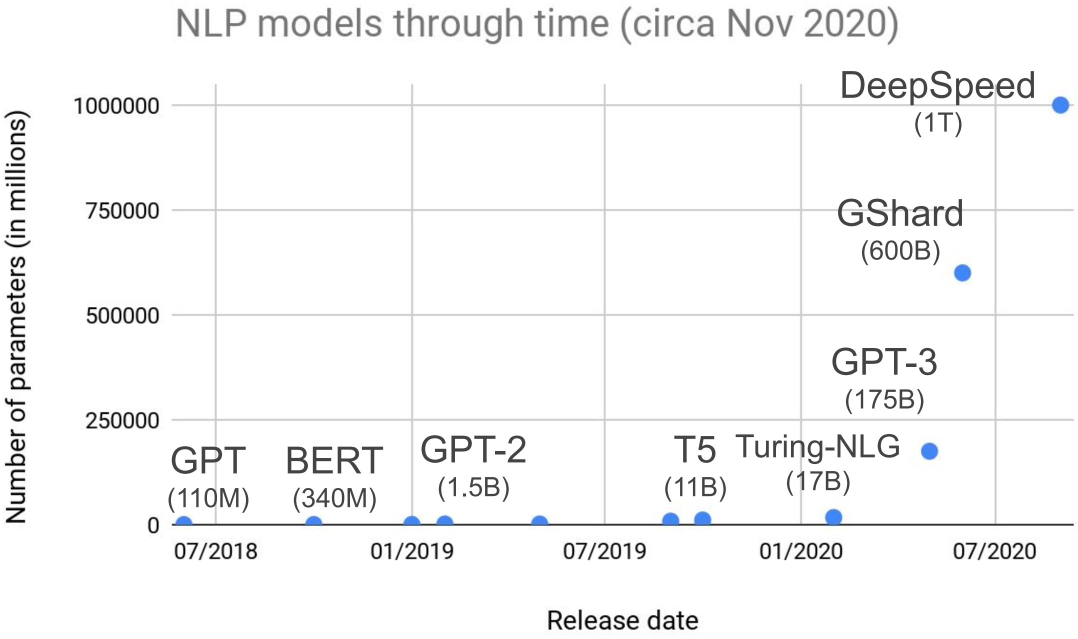
(Devlin et al., 2018)

A brief recent history of scale in NLP

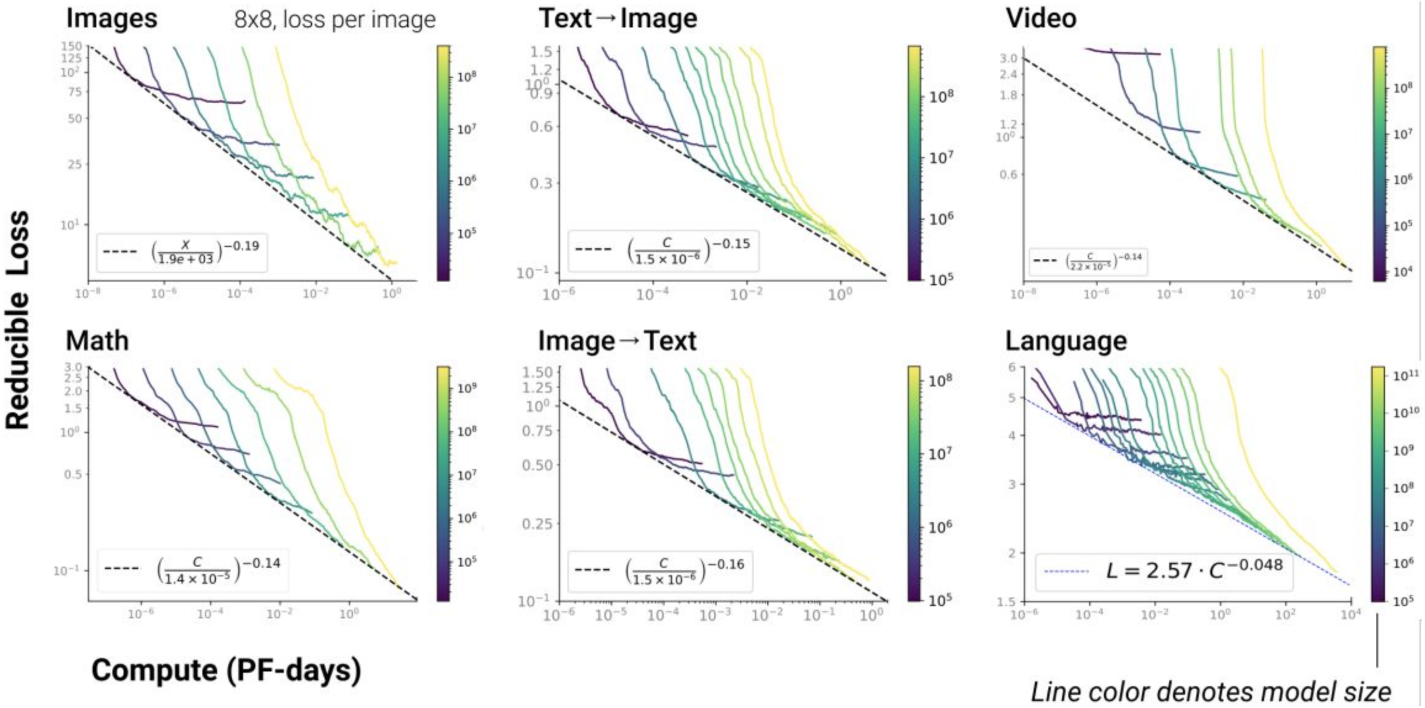


"[...] scaling the model size to 11 billion parameters was the most important ingredient for achieving our best performance." (Raffel et al, 2019)

A brief recent history of scale in NLP



Scaling Laws

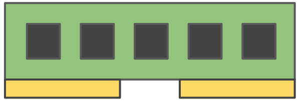


Henighan et al., 2020

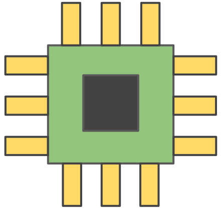
The drawbacks of naive scaling

1) **Disconnect with production systems**

   - Latency
   - Hardware constraints
   - Energy costs

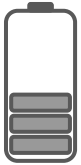


Memory



CPU/GPU/TPU

Storage



Battery

The drawbacks of naive scaling

1) Disconnect with production systems

2) Costs

- Hardware
  - 2048 TPU v3 accelerators (*GShard*, Lepikhin et al., 2020)
  - 285,000 CPU cores, 10,000 GPUs (*GPT-3*, Brown et al., 2020)
- Financial
  - GPT-3 training cost is estimated at 4.6 million dollars.

The drawbacks of naive scaling

1) Disconnect with production systems

2) Costs

3) Accessibility

- Ever-larger hardware and financial requirements impose great barriers to many researchers and institutions

- This can have a serious impact in our research community

    - For instance, 62% of PhD students have access to 4 or less GPUs, according to a recent poll.

The drawbacks of naive scaling

1) Disconnect with production systems
2) Costs
3) Accessibility

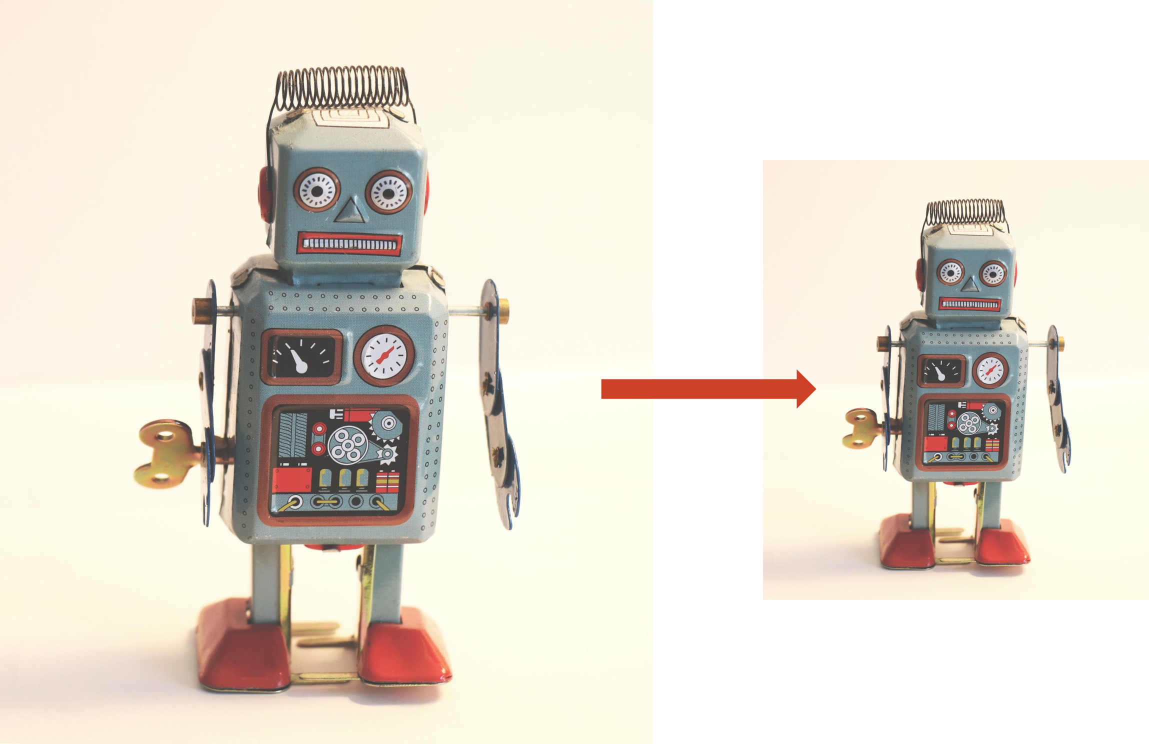
Altogether, this is especially relevant to a field that scaled by 3 orders of magnitude in 2 years.

We should **strive for efficiency**

Towards more efficient NLP

1) **Core techniques**

   - Knowledge Distillation



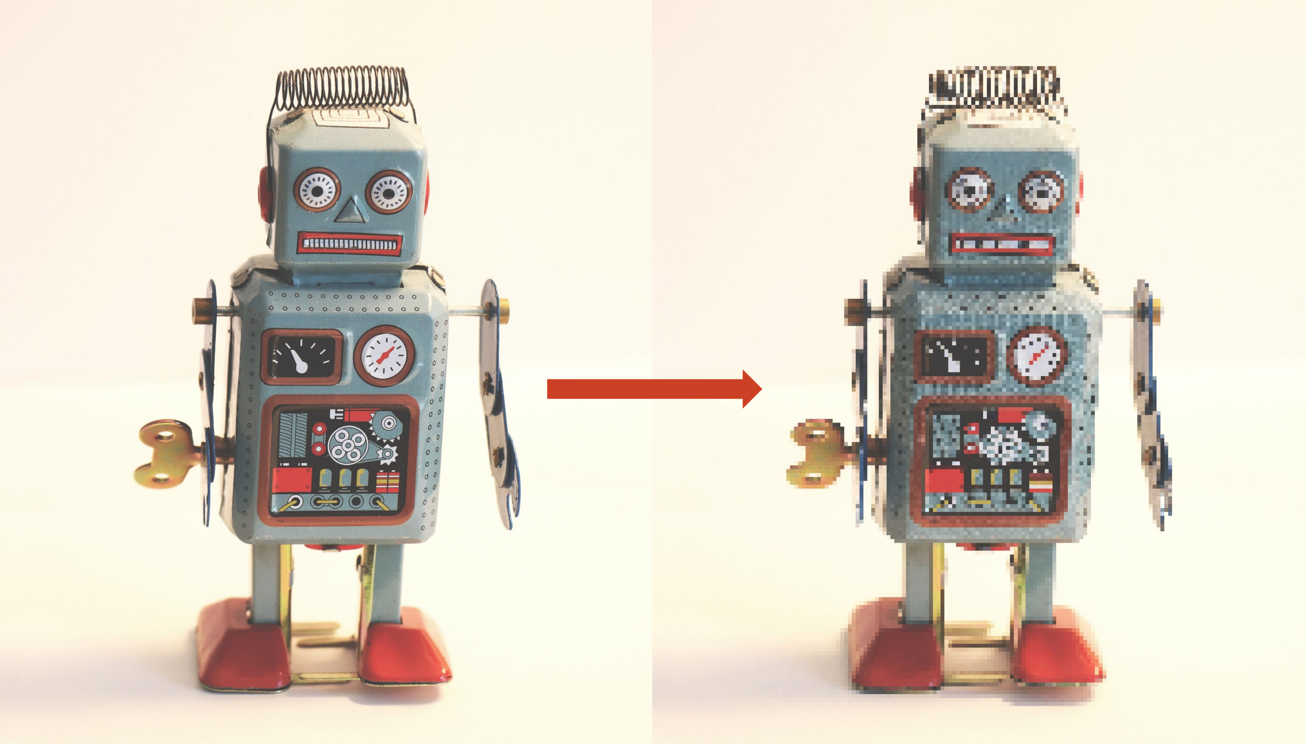
Source: unsplash.com

Towards more efficient NLP

1) **Core techniques**

   - Knowledge Distillation

   - Quantization



Source: unsplash.com

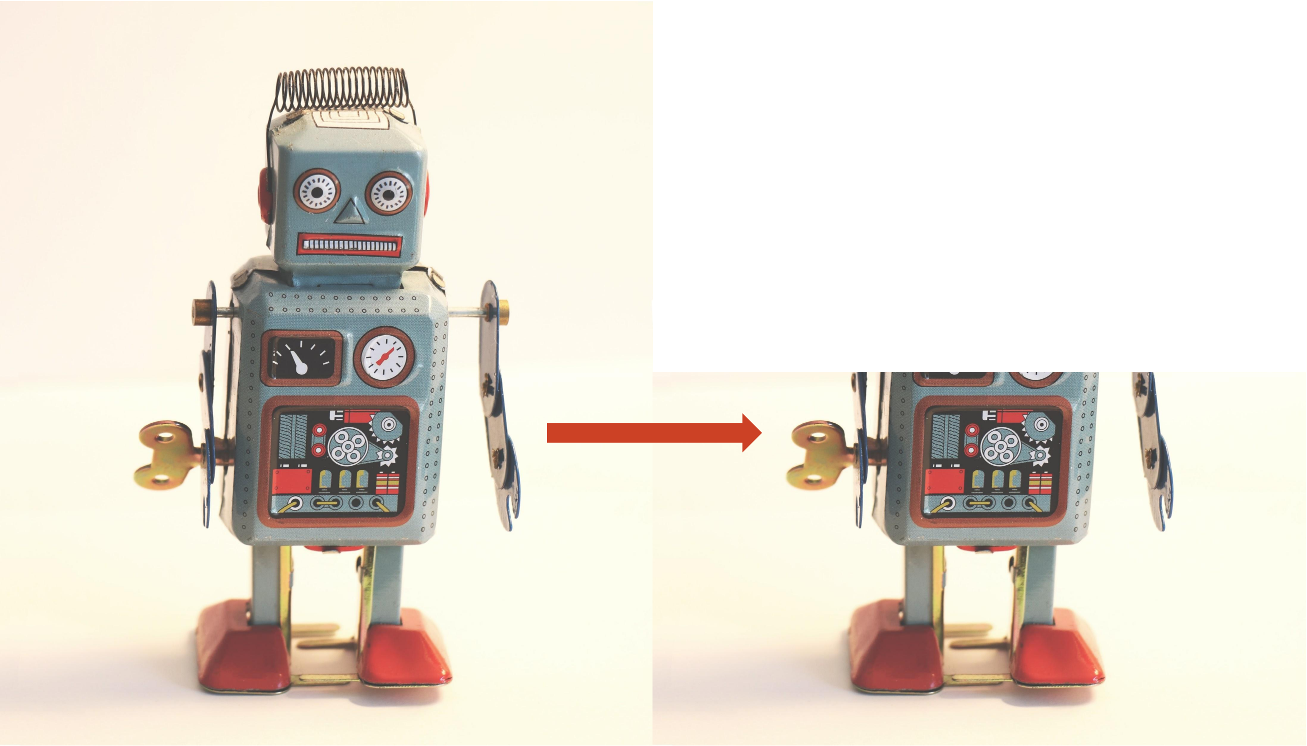
Towards more efficient NLP

1) **Core techniques**

    - Knowledge Distillation

    - Quantization

    - Pruning



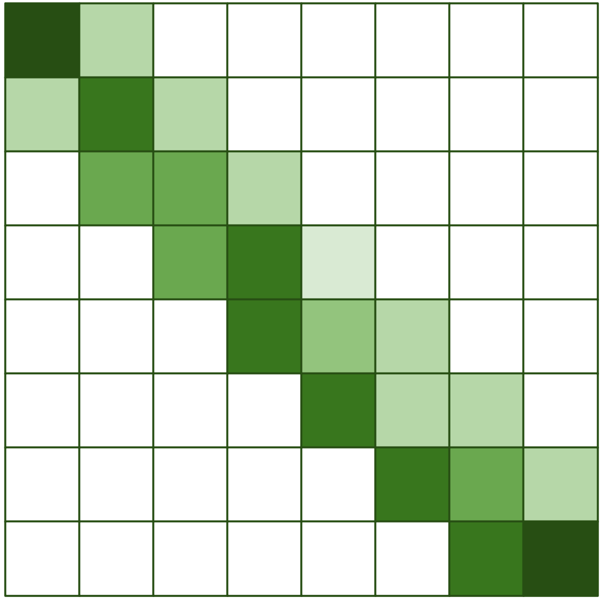
Source: unsplash.com

Towards more efficient NLP

1) Core techniques

2) **Efficient attention**

   - Data-Independent Patterns



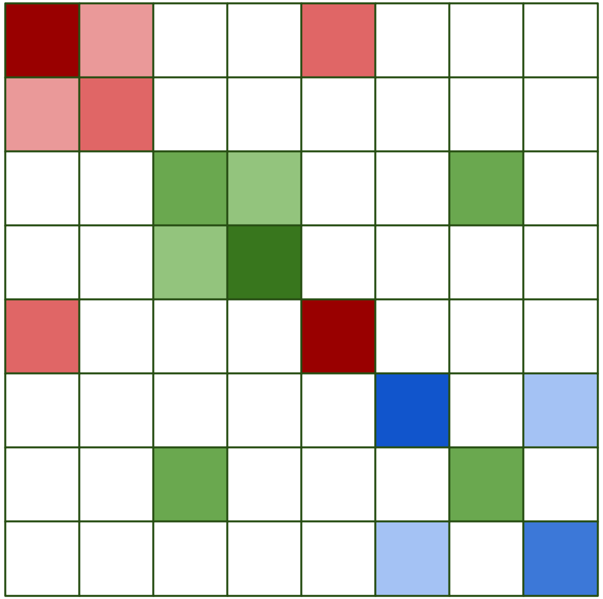
Towards more efficient NLP

1) Core techniques

2) **Efficient attention**

   - Data-Independent Patterns

   - Data-Dependent Patterns



Towards more efficient NLP

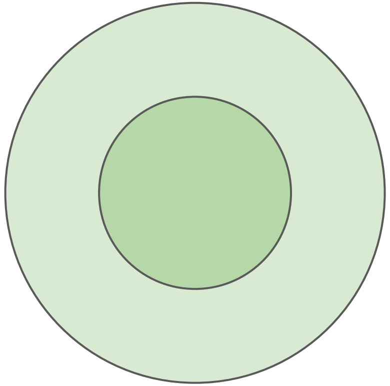
1) Core techniques

2) Efficient attention

- Data-Independent Patterns

- Data-Dependent Patterns

- Kernels and Alternative Attention Mechanisms

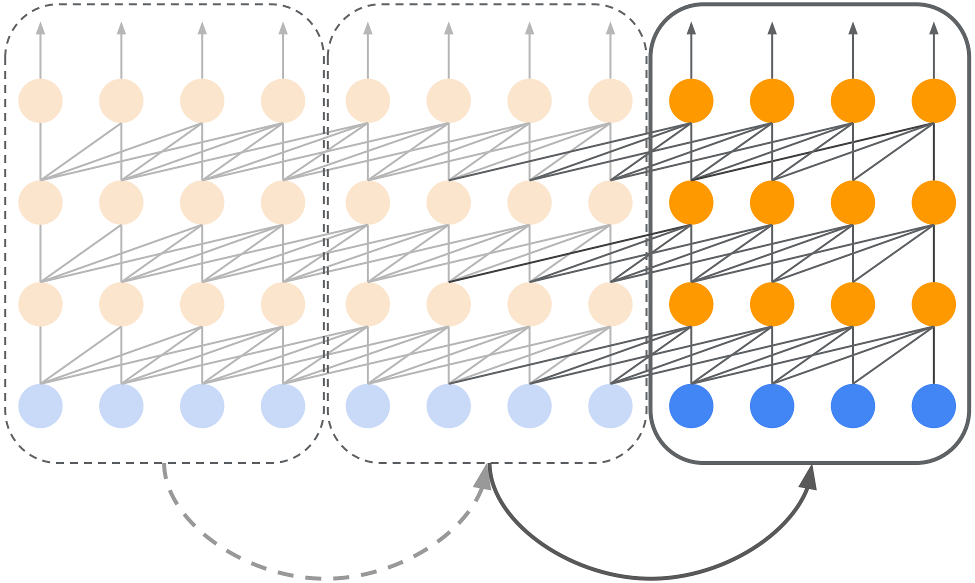


Towards more efficient NLP

1) Core techniques

2) Efficient attention

- Data-Independent Patterns
- Data-Dependent Patterns
- Alternative Attention Mechanisms
- Recurrence



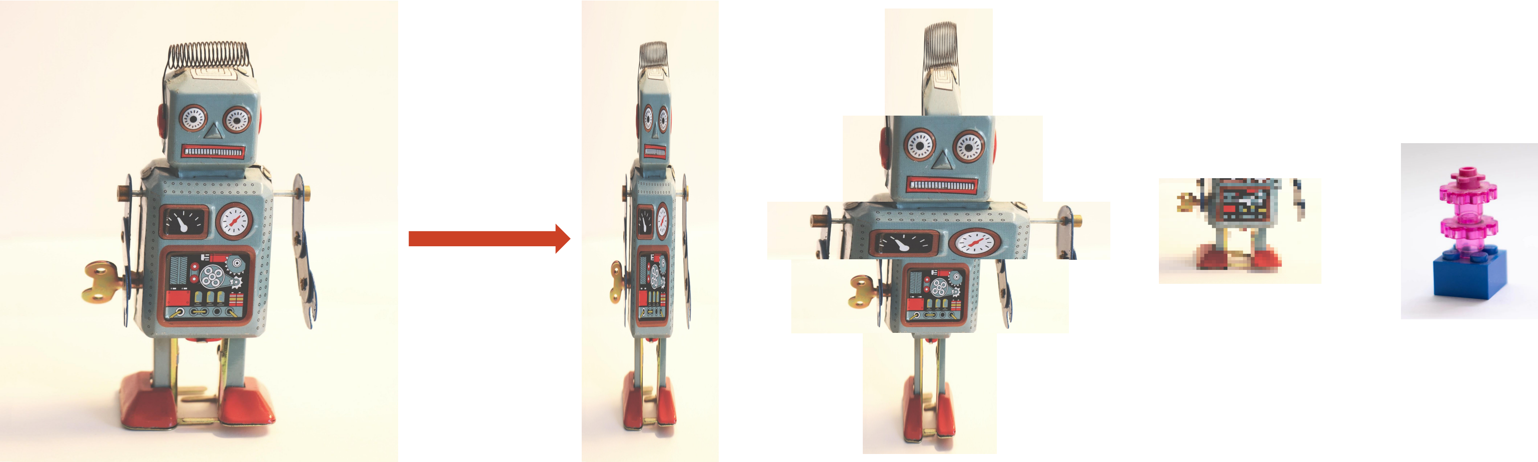
Towards more efficient NLP

1) Core techniques

2) Efficient attention

3) **Case studies**

   - Efficient Language Models



Source: unsplash.com

Towards more efficient NLP

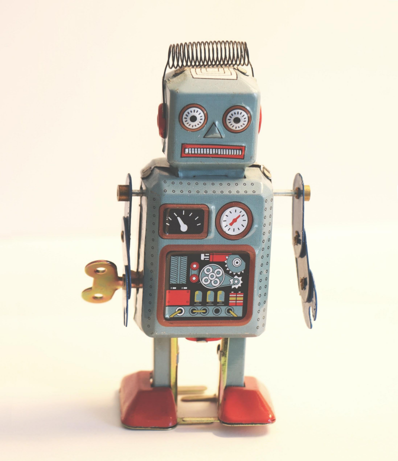
1) Core techniques

2) Efficient attention

3) **Case studies**

    - Efficient Language Models

    - Retrieval

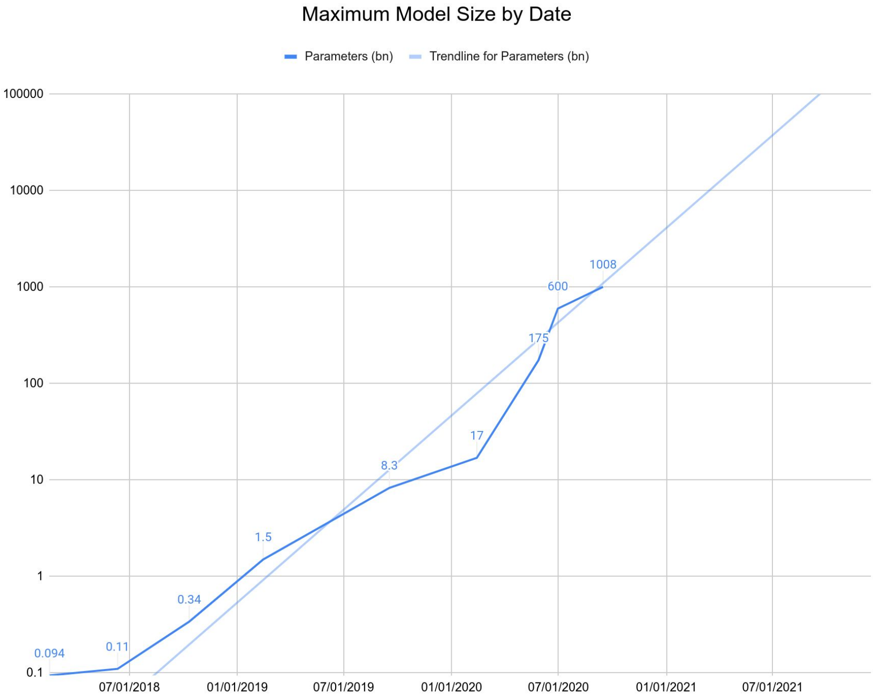


+

Source: unsplash.com

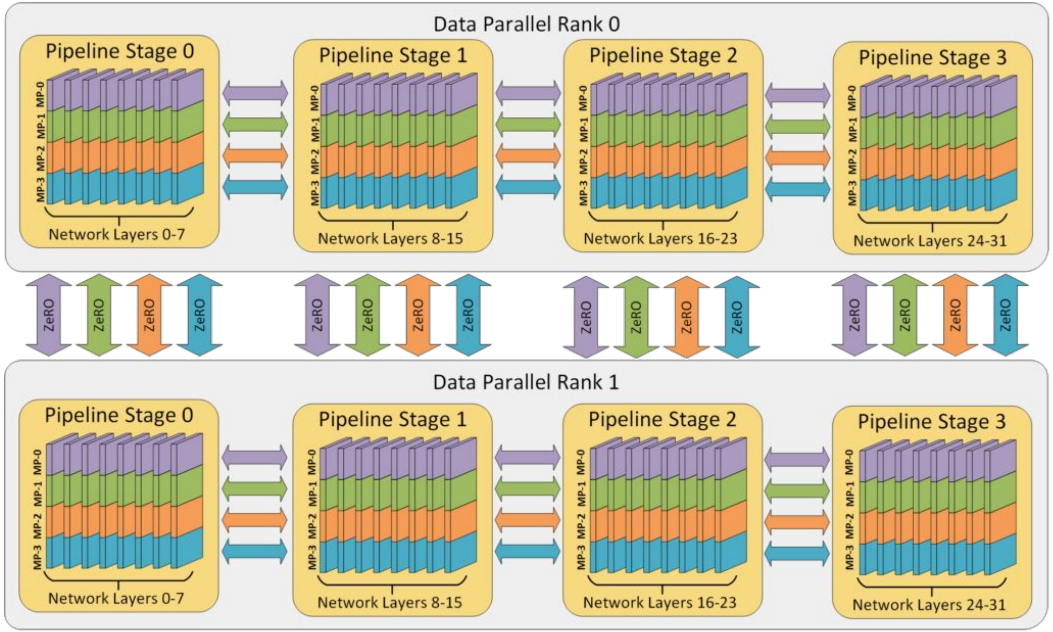
Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) Case studies
4) **Scaling in Practice**
    - Scaling Laws of Neural Language Models



Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) Case studies
4) **Scaling in Practice**
    - Scaling Laws of Neural Language Models
    - Parallelism Techniques

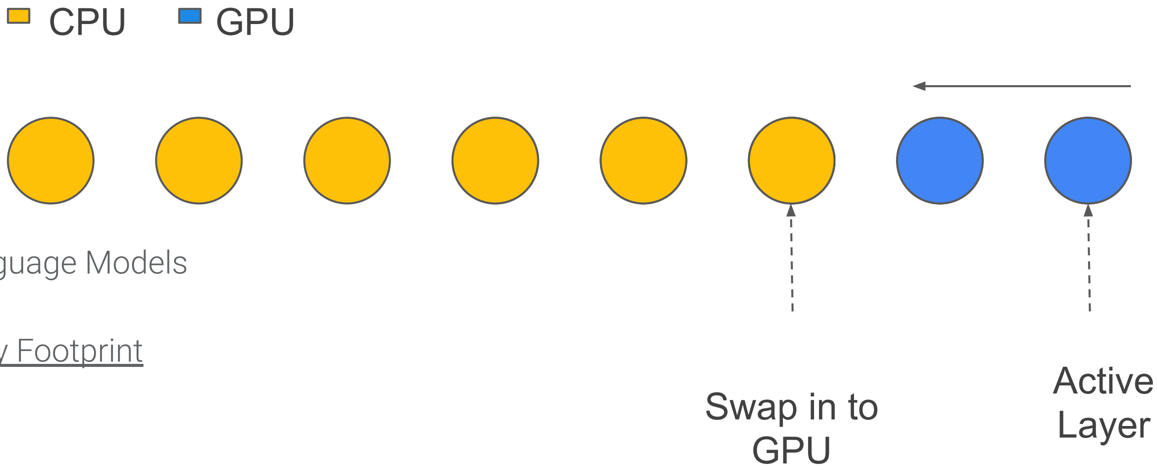


Source: Microsoft Blog Post

Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) Case studies
4) **Scaling in Practice**

- Scaling Laws of Neural Language Models
- Parallelism Techniques
- Methods to Reduce Memory Footprint



Towards more efficient NLP

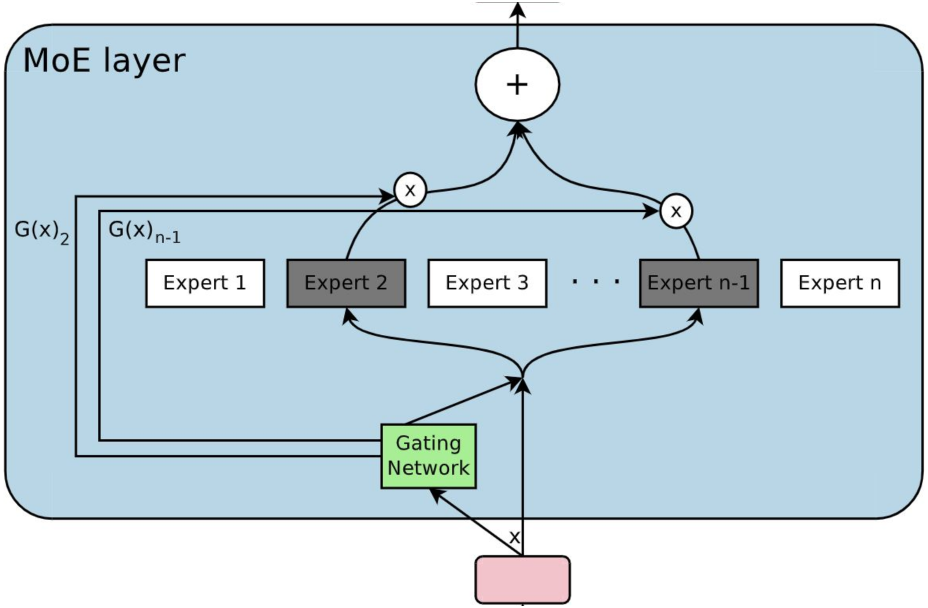
1) Core techniques

2) Efficient attention

3) Case studies

4) **Scaling in Practice**

- Scaling Laws of Neural Language Models
- Parallelism Techniques
- Methods to Reduce Memory Footprint
- Mixture of Experts



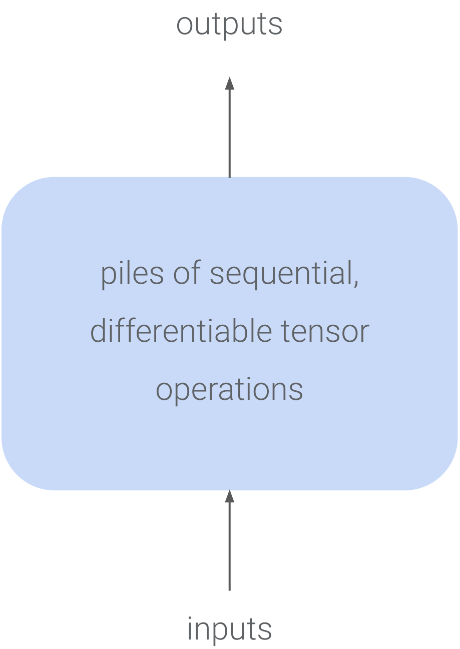
02

Fundamentals

Sequence-to-sequence models

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

https://xkcd.com/1838/

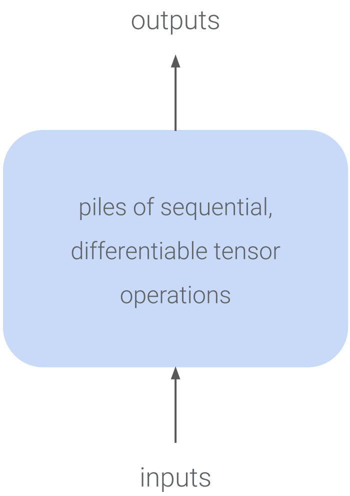


outputs

piles of sequential, differentiable tensor operations

inputs

Sequence-to-sequence models



outputs

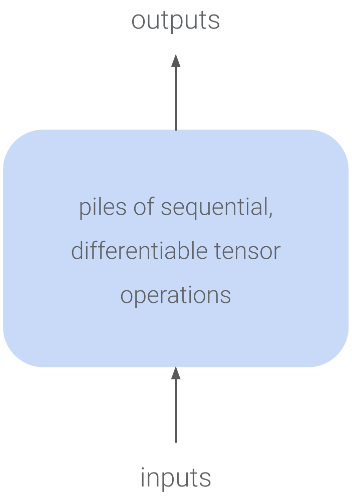
piles of sequential, differentiable tensor operations

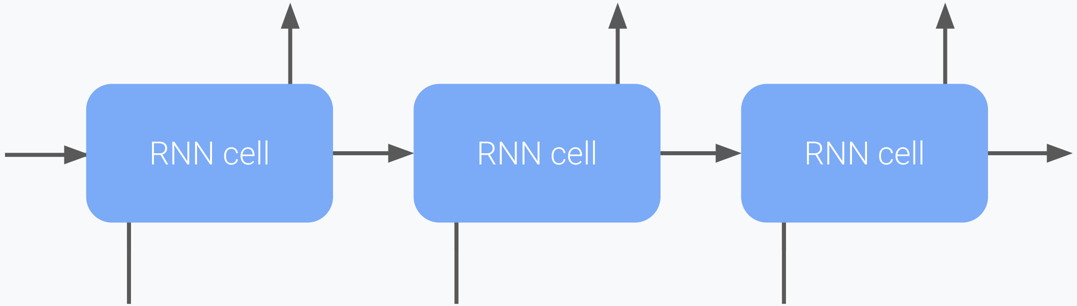
inputs

	inputs	outputs
Machine Translation	Hello, world	Olá, mundo
Sentiment Analysis	Amazing movie, 10/10!	★★★★★
Language modeling	-	The quick brown fox...
Speech recognition		Hello, world

RNNs

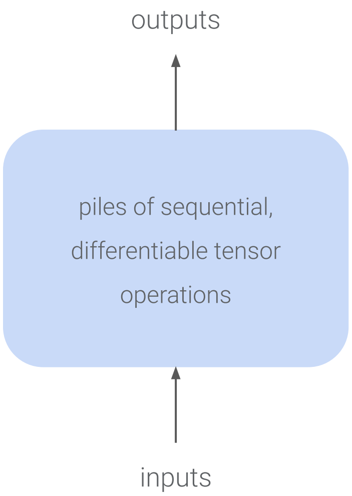
RNNs allow computations over sequences of arbitrary length

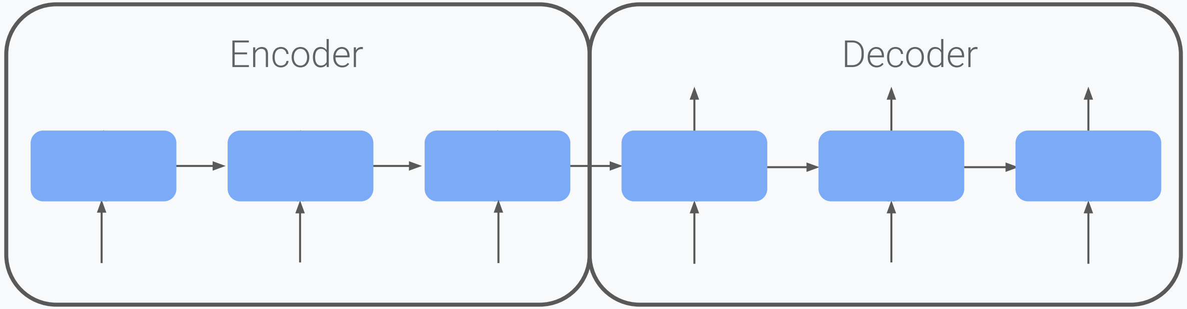




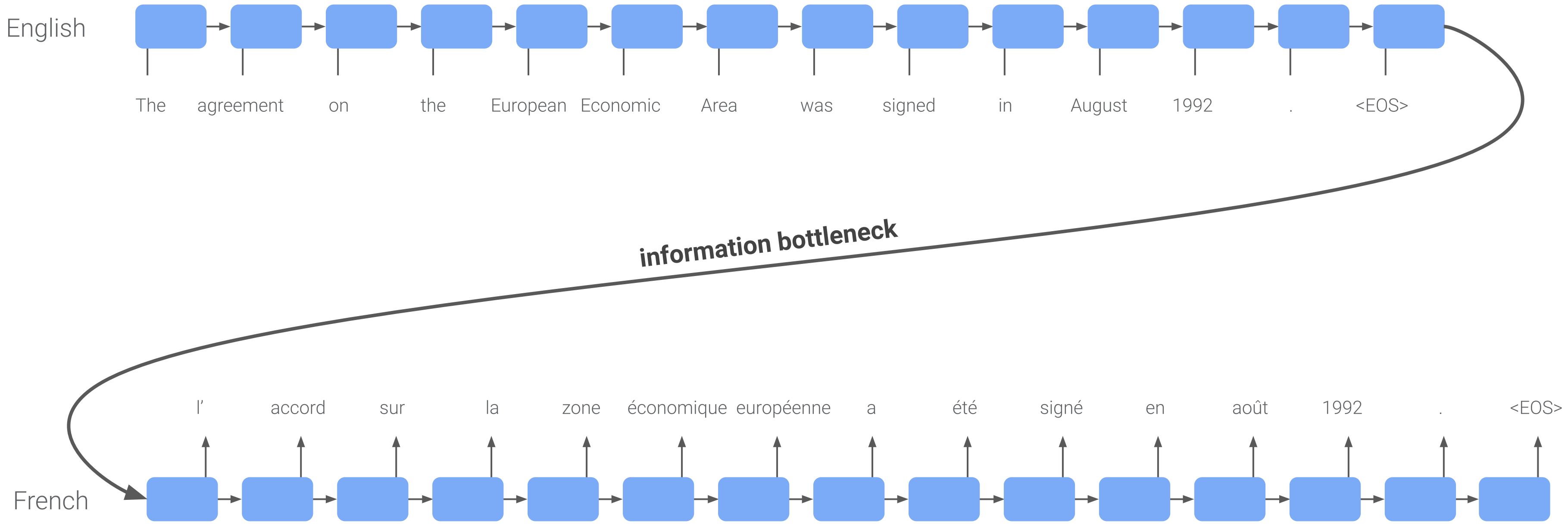
Encoders and Decoders

RNNs allow computations over sequences of arbitrary length



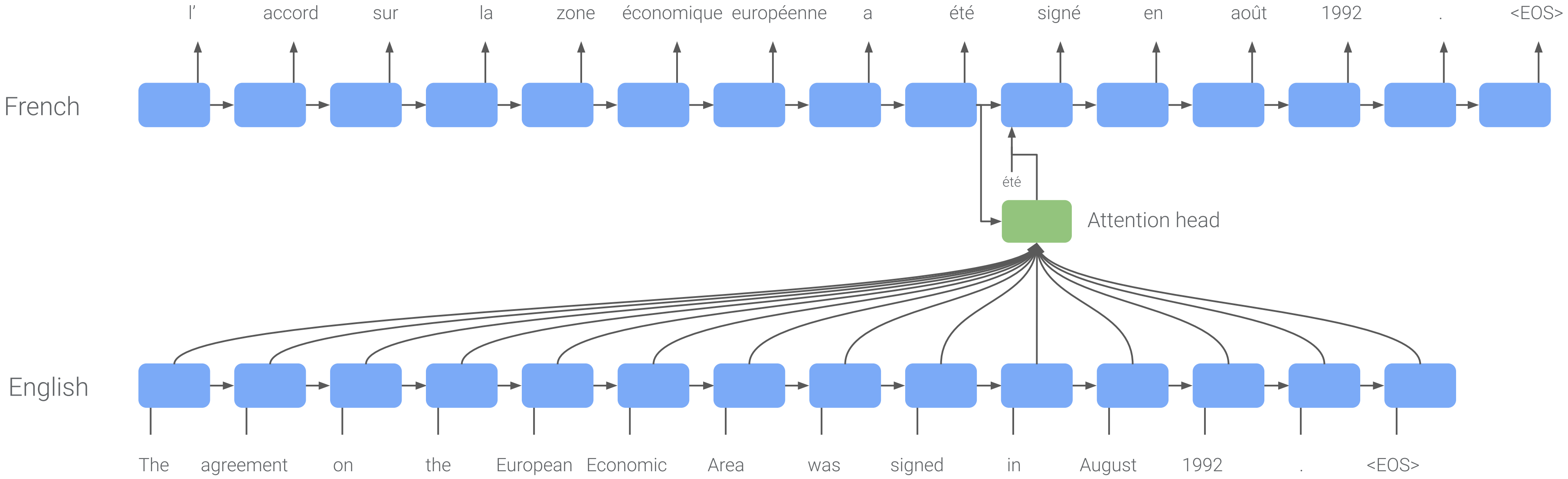


The encoder-decoder bottleneck



Example derived from Bahdanau, et al. 2014 (https://arxiv.org/pdf/1409.0473.pdf)

Attention



Example derived from Bahdanau, et al. 2014 (https://arxiv.org/pdf/1409.0473.pdf)

Attention

Bahdanau et al.

Neural machine translation by jointly learning to align and translate. 2014

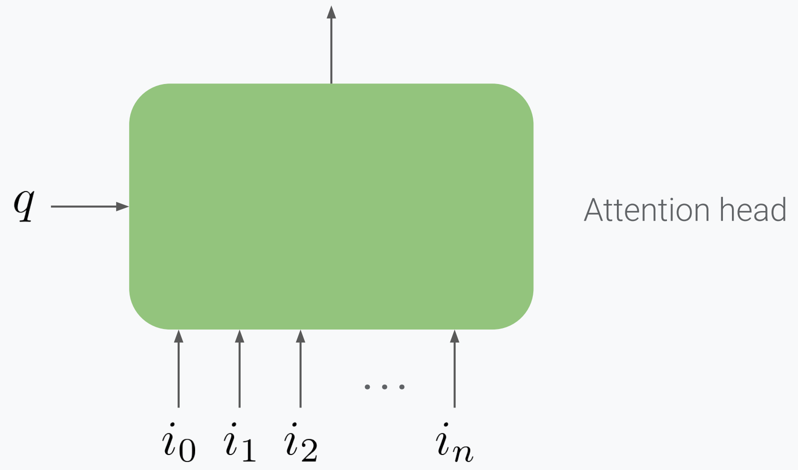
Thang Luong et al.

Effective approaches to attention-based neural machine translation. 2015

A *summary* of i,

based on how similar their

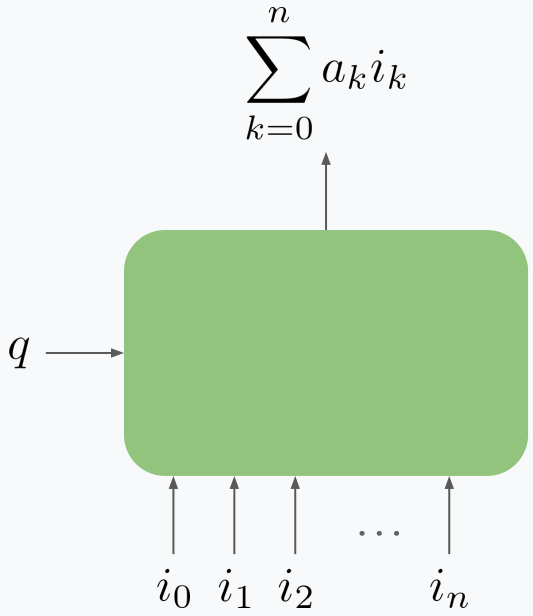
are with the query



Dot product attention

Thang Luong et al.

Effective approaches to attention-based neural machine translation. 2015



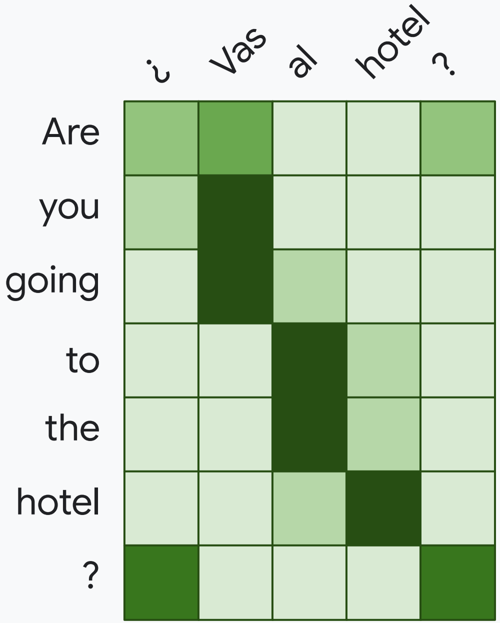
Attention head

$$\phi(q, i_k) = \exp(q^\top i_k)$$

$$a_k = \frac{\phi(q, i_k)}{\sum_{j=0}^{n} \phi(q, i_j)}$$

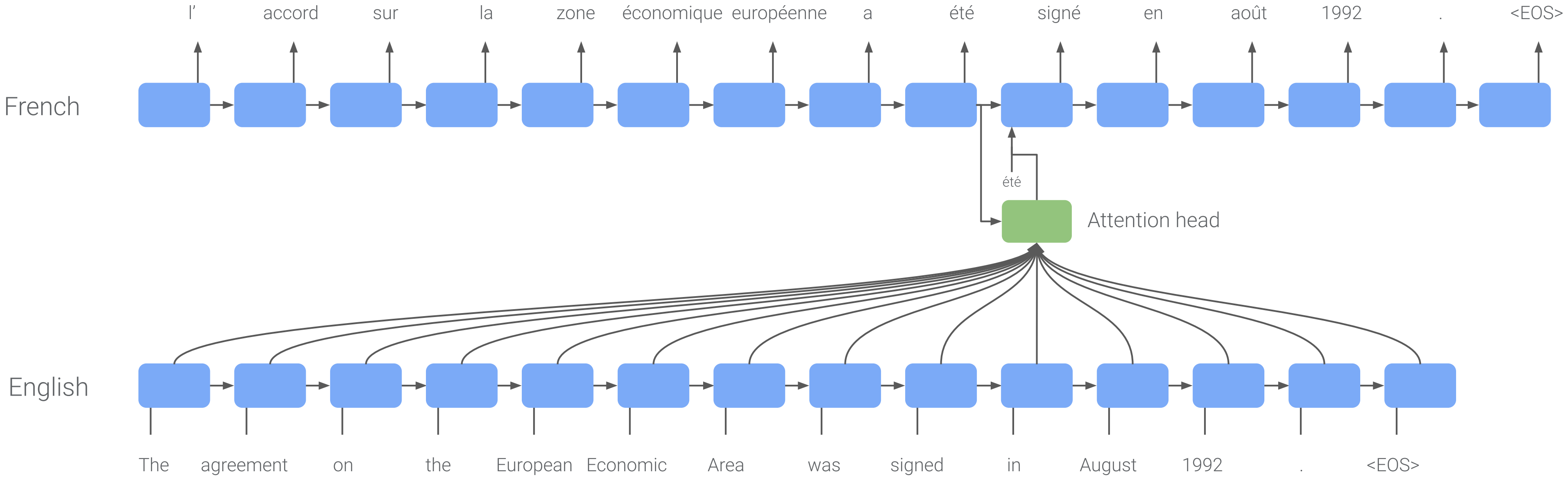
Attention mechanisms

The attention matrix



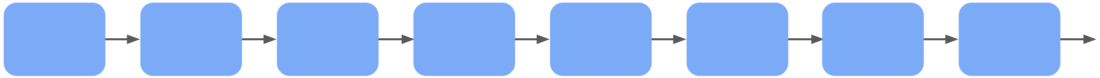
Transformers

Motivation

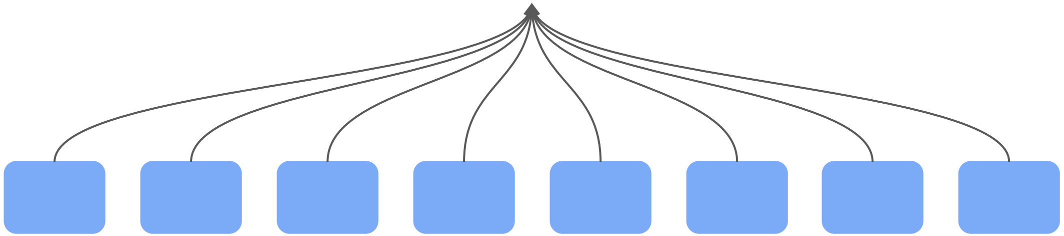


Example derived from Bahdanau, et al. 2014 (https://arxiv.org/pdf/1409.0473.pdf)

Motivation



sequential

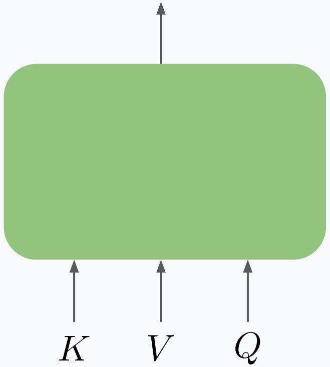


parallel

Scaled Dot-Product Attention

Queries, keys and values

A *summary* of values, based on how similar their corresponding keys are with the query

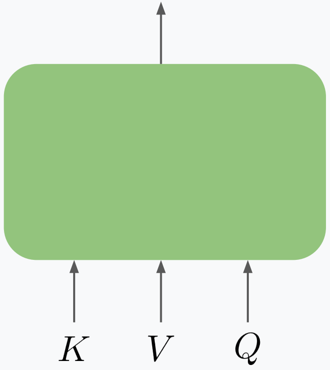


Scaled Dot-Product Attention

Queries, keys and values

For some similarity function ϕ

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



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

Scaled Dot-Product Attention

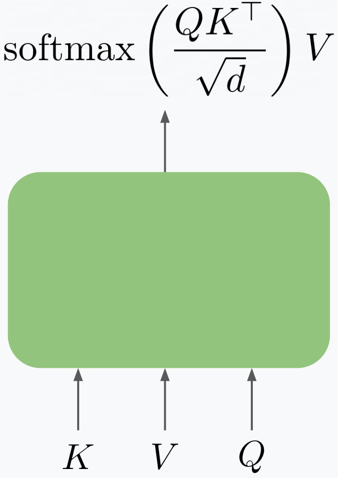
Using dot-product similarity, we can vectorize nicely

$$\phi(Q_i, K_j) = \exp\left(\frac{Q_i K_j^\top}{\sqrt{d}}\right)$$

Normalization factor for numerical stability

d = feature dim

$$\text{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}$$



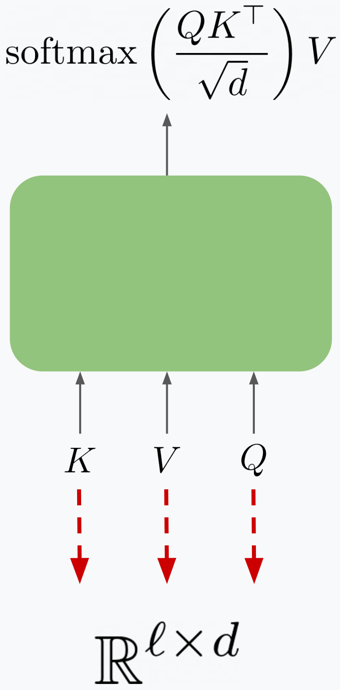
$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

Scaled Dot-Product Attention

Let’s dive into the dimensions (batch omitted for simplicity)

ℓ = sequence length

d = feature dim

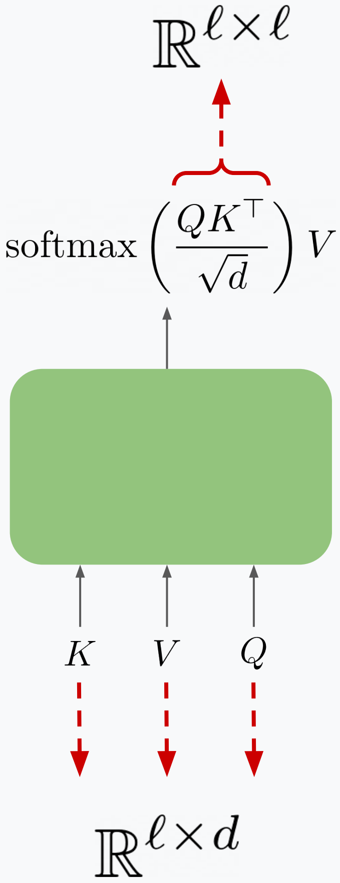

$\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$
K
V
Q
$\mathbb{R}^{\ell \times d}$

Scaled Dot-Product Attention

Let's dive into the dimensions (batch omitted for simplicity)

ℓ = sequence length

d = feature dim



$\mathbb{R}^{\ell\times\ell}$
$\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$
K
V
Q
$\mathbb{R}^{\ell\times d}$

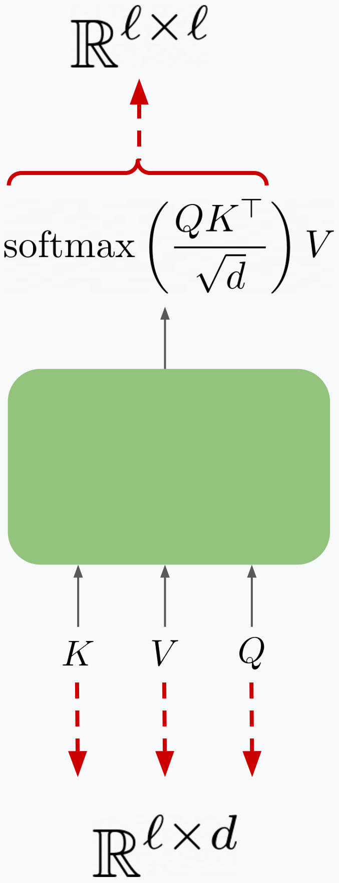
Scaled Dot-Product Attention

Let's dive into the dimensions

(batch omitted for simplicity)

ℓ = sequence length

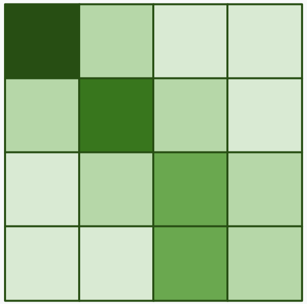
d = feature dim



$$\mathbb{R}^{\ell\times\ell}$$

$$\mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

$$\mathbb{R}^{\ell\times d}$$

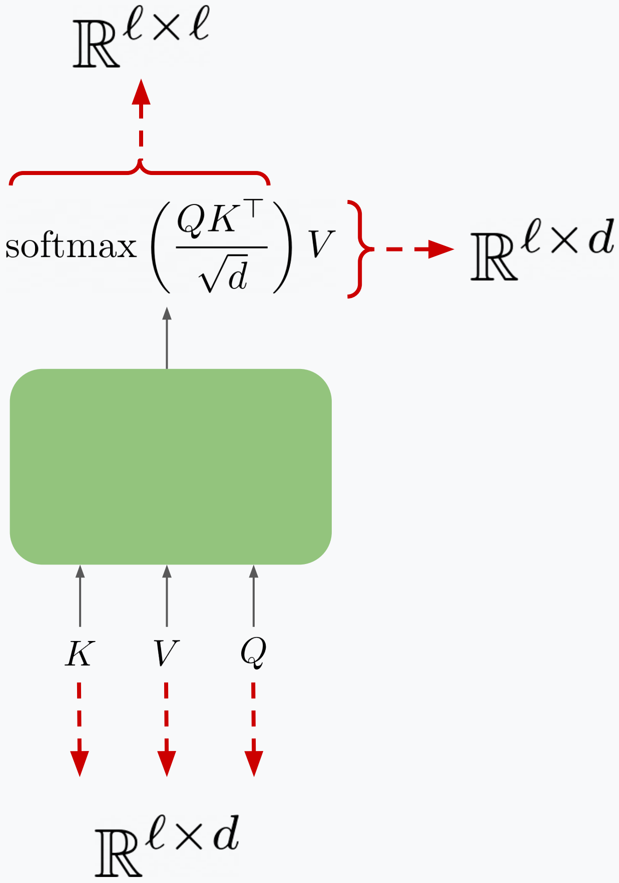


Scaled Dot-Product Attention

Let's dive into the dimensions (batch omitted for simplicity)

ℓ = sequence length

d = feature dim

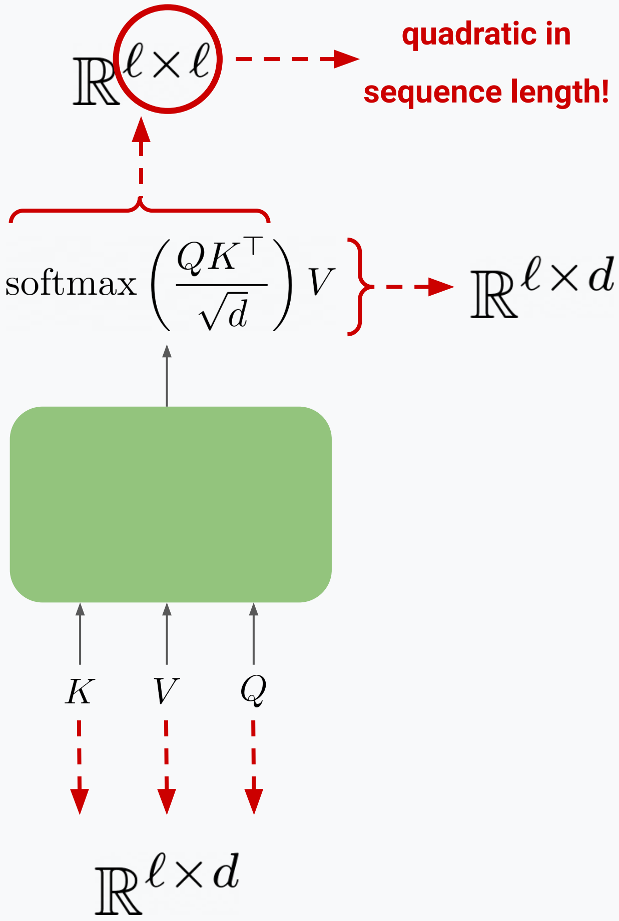

$\mathbb{R}^{\ell \times \ell}$
$\underbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)}$ V
$\mathbb{R}^{\ell \times d}$
K
V
Q
$\mathbb{R}^{\ell \times d}$

Scaled Dot-Product Attention

Let's dive into the dimensions (batch omitted for simplicity)

ℓ = sequence length

d = feature dim

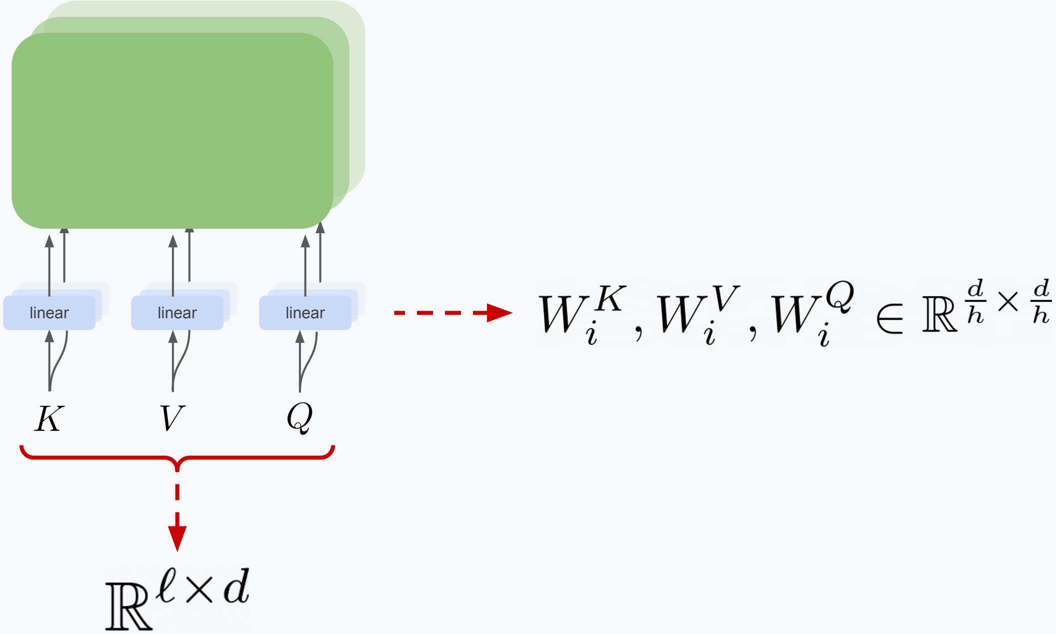

$\mathbb{R}^{\ell \times \ell}$
quadratic in sequence length!
$\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$
$\mathbb{R}^{\ell \times d}$
K
V
Q
$\mathbb{R}^{\ell \times d}$

Multi-head attention

ℓ = sequence length

d = feature dim

h = # of attention heads



$$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$$

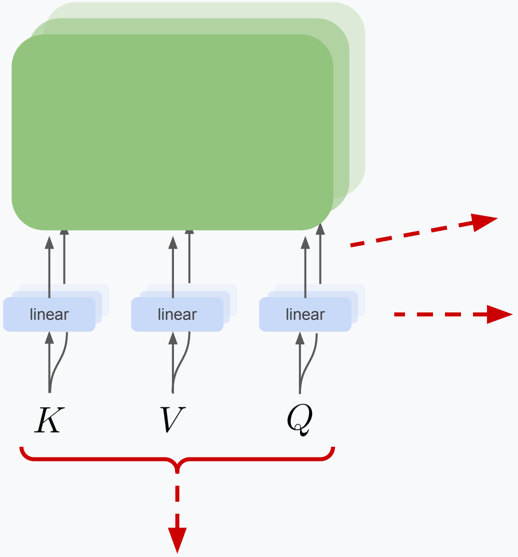
$$\mathbb{R}^{\ell \times d}$$

Multi-head attention

ℓ = sequence length

d = feature dim

h = # of attention heads



each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

$$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$$

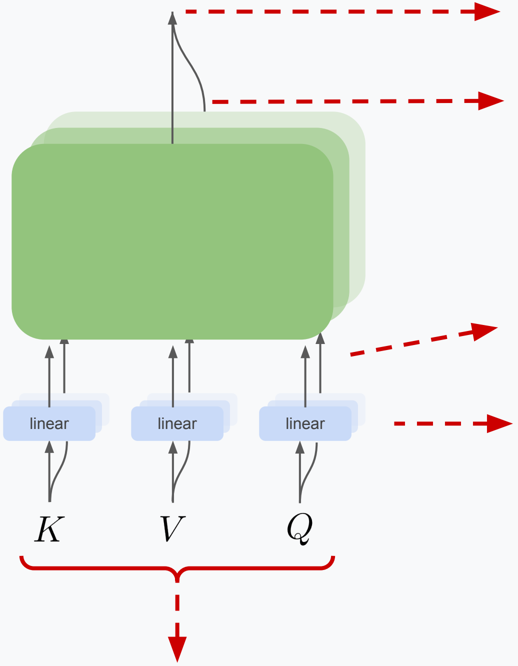
$$\mathbb{R}^{\ell \times d}$$

Multi-head attention

ℓ = sequence length

d = feature dim

h = # of attention heads



$\mathbb{R}^{\ell \times d}$

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$

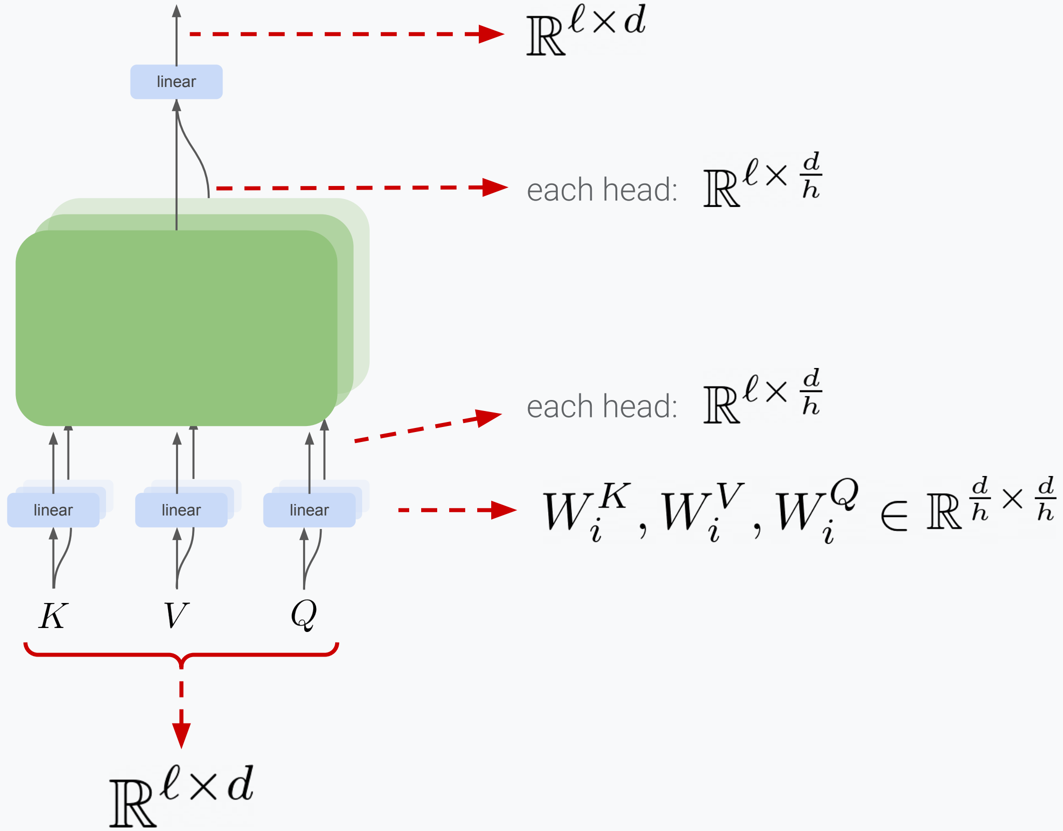
$\mathbb{R}^{\ell \times d}$

Multi-head attention

ℓ = sequence length

d = feature dim

h = # of attention heads



$\mathbb{R}^{\ell \times d}$

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$

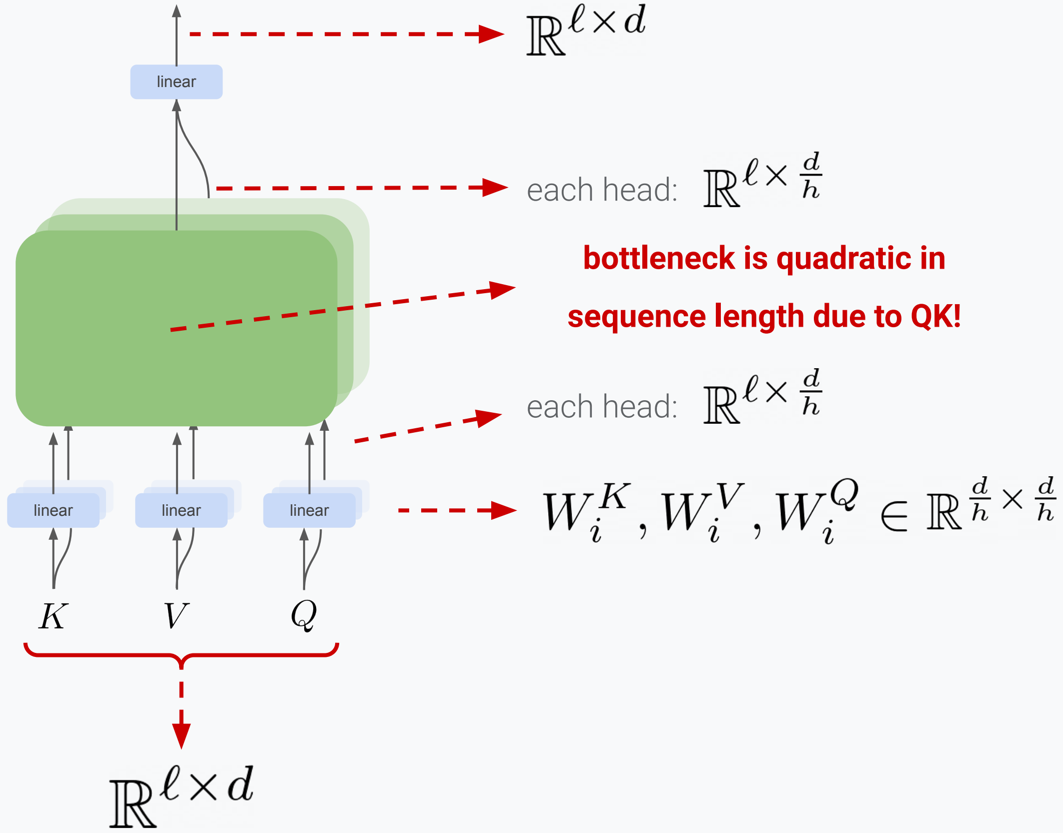
$\mathbb{R}^{\ell \times d}$

Multi-head attention

ℓ = sequence length

d = feature dim

h = # of attention heads



$\mathbb{R}^{\ell \times d}$

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

bottleneck is quadratic in sequence length due to QK!

each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

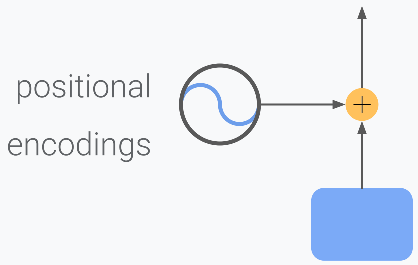
$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$

$\mathbb{R}^{\ell \times d}$

Positional encodings

So far, attention has been a set operation.

Let’s add positional information!

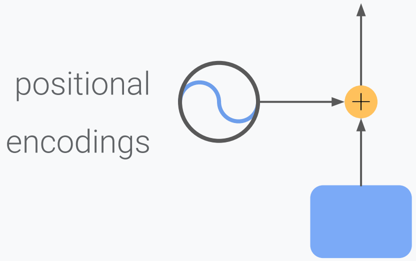


Positional encodings

So far, attention has been a set operation.

Let's add positional information!

These can be either **learned** or **fixed**.



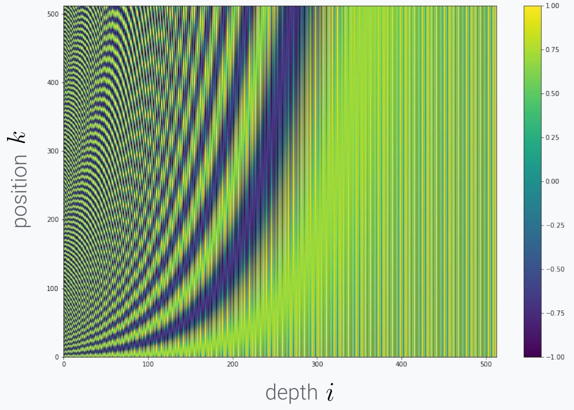
positional encodings

Fixed:

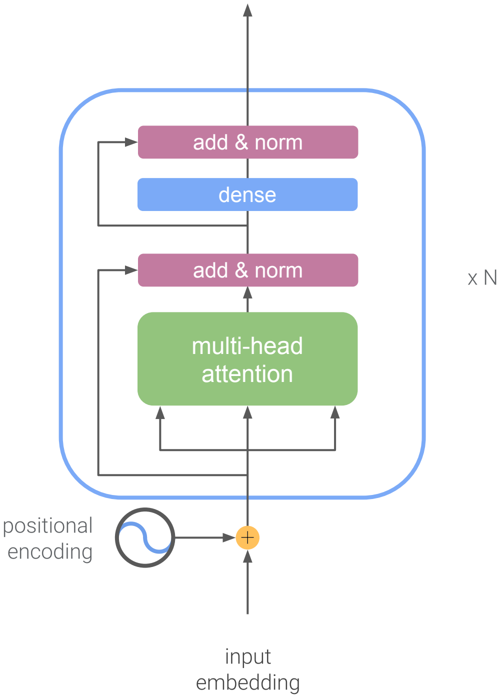
for a position k in the sequence

and i in the feature space

$$E_{ki} = \begin{cases} \sin\left(k/10000^{\frac{i}{N}}\right) & \text{if } i \text{ is even} \\ \cos\left(k/10000^{\frac{i-1}{N}}\right) & \text{if } i \text{ is odd} \end{cases}$$



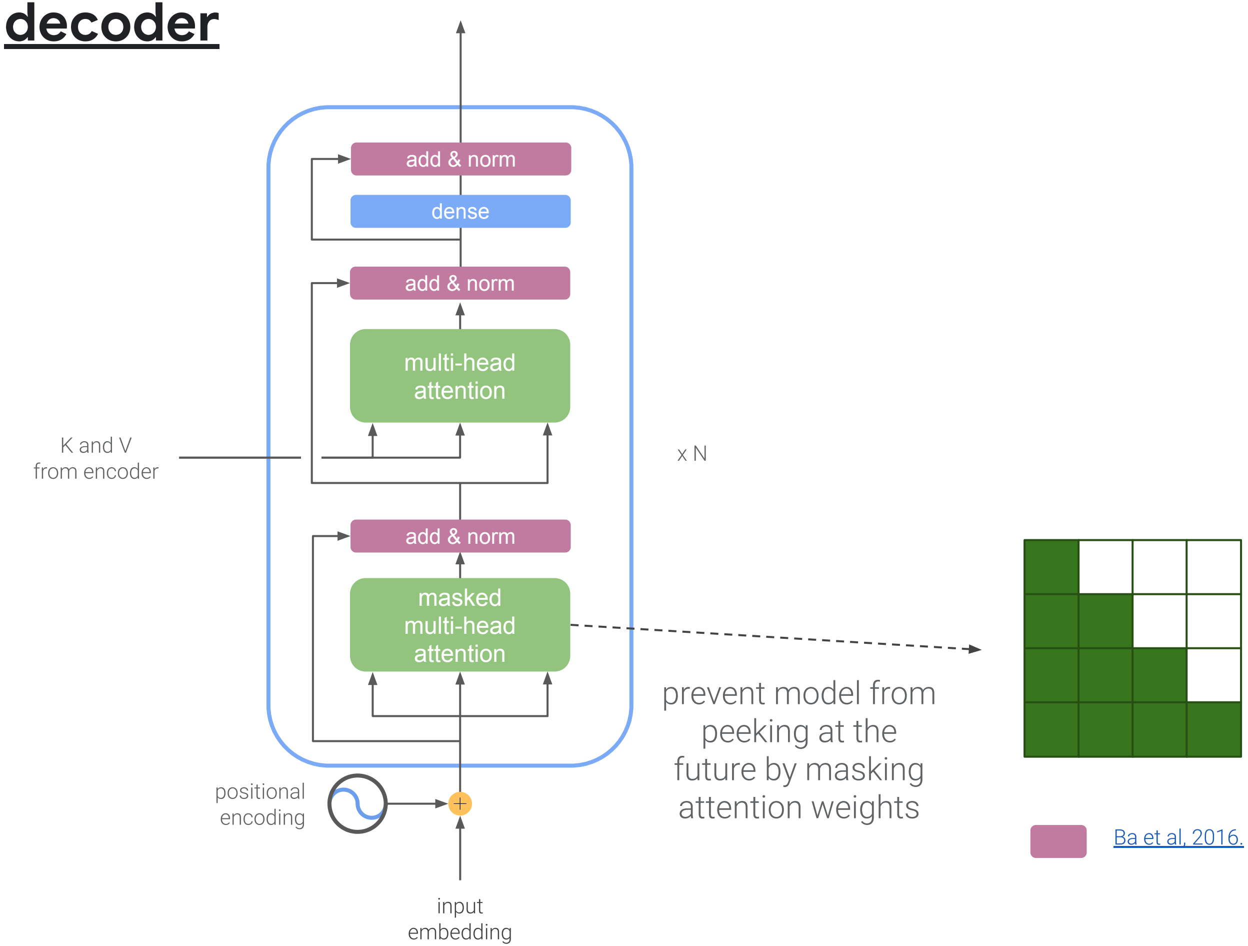
The transformer encoder



🟪 Ba et al, 2016.

Vaswani et al., 2017

The transformer decoder

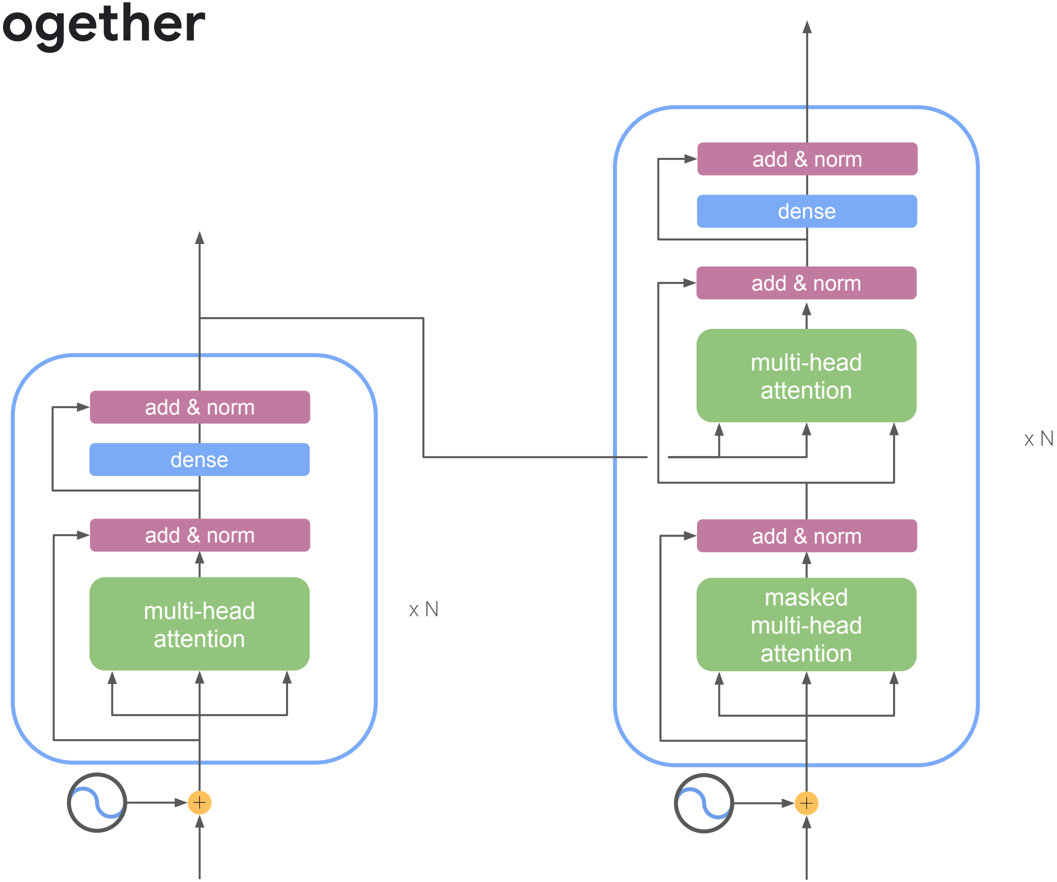


prevent model from peeking at the future by masking attention weights

🟪 Ba et al, 2016.

Vaswani et al., 2017

Putting it all together

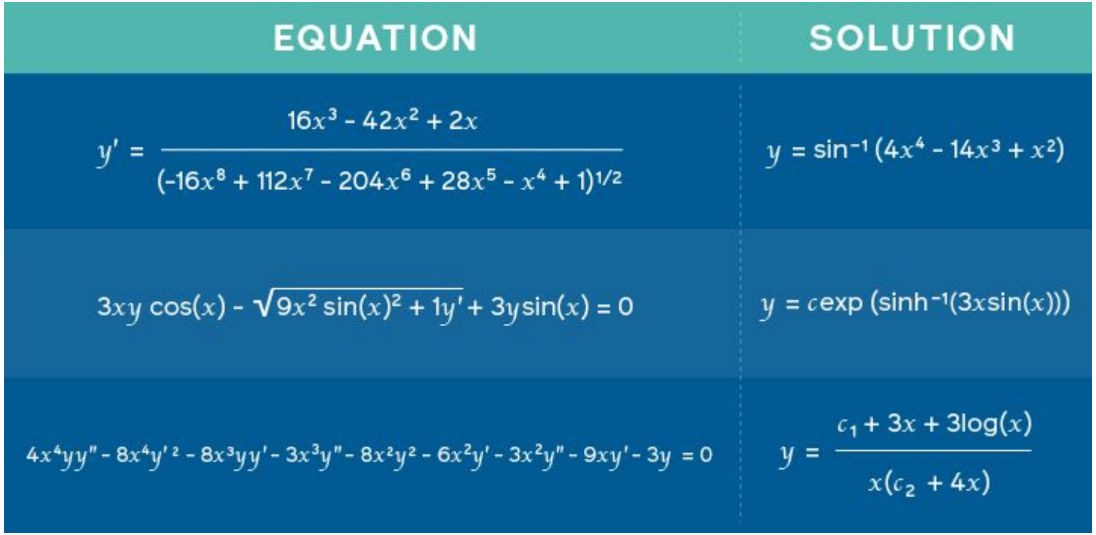


Vaswani et al., 2017

Transformers in recent literature

Transformers have become successful in a wide range of domains and applications, including:

- Mathematics and theorem proving (e.g. Lample et al., 2019, Clark et al., 2020)



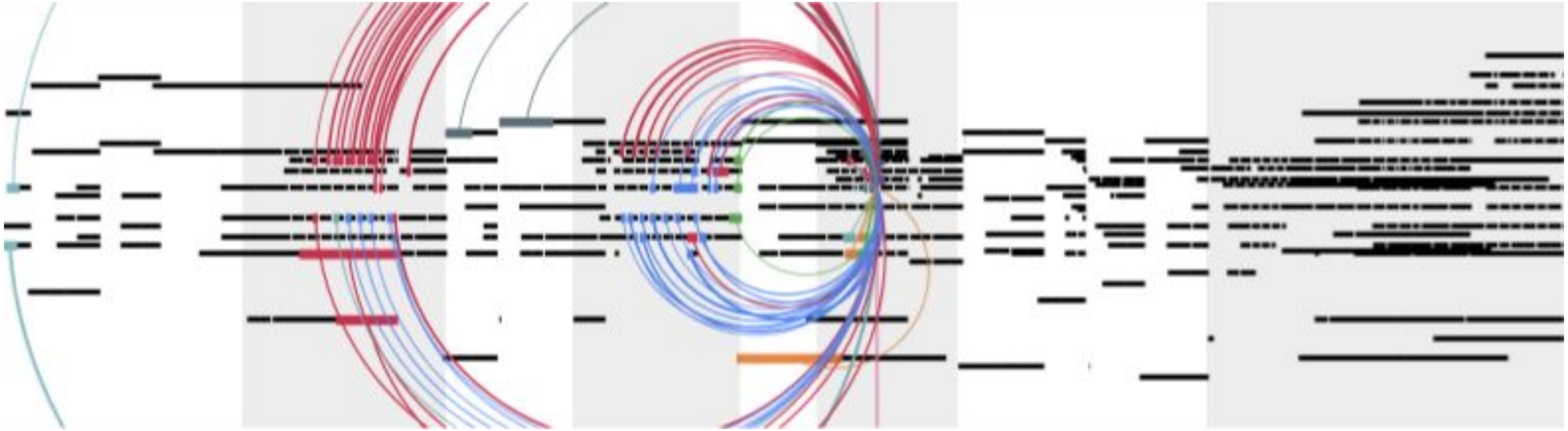
EQUATION	SOLUTION
$y' = \dfrac{16x^3 - 42x^2 + 2x}{(-16x^8 + 112x^7 - 204x^6 + 28x^5 - x^4 + 1)^{1/2}}$	$y = \sin^{-1}(4x^4 - 14x^3 + x^2)$
$3xy\cos(x) - \sqrt{9x^2\sin(x)^2 + 1}\,y' + 3y\sin(x) = 0$	$y = c\exp(\sinh^{-1}(3x\sin(x)))$
$4x^4yy'' - 8x^4y'^2 - 8x^3yy' - 3x^3y'' - 8x^2y^2 - 6x^2y' - 3x^2y'' - 9xy' - 3y = 0$	$y = \dfrac{c_1 + 3x + 3\log(x)}{x(c_2 + 4x)}$

https://ai.facebook.com/blog/using-neural-networks-to-solve-advanced-mathematics-equations/

Transformers in recent literature

Transformers have become successful in a wide range of domains and applications, including:

- Mathematics and theorem proving (e.g. Lample et al., 2019, Clark et al., 2020)
- Music generation (e.g. Anna Huang et al., 2019)

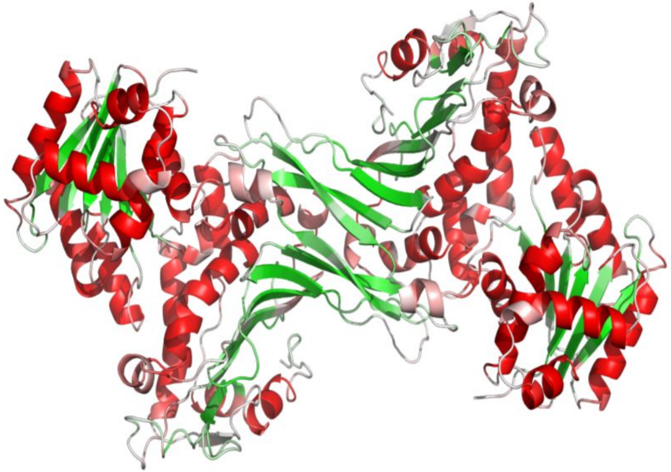


https://magenta.tensorflow.org/music-transformer

Transformers in recent literature

Transformers have become successful in a wide range of domains and applications, including:

- Mathematics and theorem proving (e.g. Lample et al., 2019, Clark et al., 2020)
- Music generation (e.g. Anna Huang et al., 2019)
- Biology (e.g. Rives et al., 2019, Madani et al., 2020)



Transformers in recent literature

Transformers have become successful in a wide range of domains and applications, including:

- Mathematics and theorem proving (e.g. Lample et al., 2019, Clark et al., 2020)
- Music generation (e.g. Anna Huang et al., 2019)
- Biology (e.g. Rives et al., 2019, Madani et al., 2020)
- Vision and Language (e.g. Tan et al., 2019, Lu et al., 2019, Chen et al., 2020)

What color are her eyes?
What is the mustache made of?

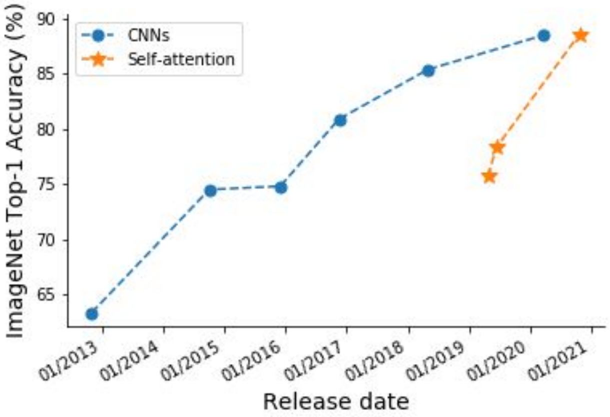
How many slices of pizza are there?
Is this a vegetarian pizza?

Visual Question Answering
(Agrawal et al., 2015)

Transformers in recent literature

Transformers have become successful in a wide range of domains and applications, including:

- Mathematics and theorem proving (e.g. Lample et al., 2019, Clark et al., 2020)
- Music generation (e.g. Anna Huang et al., 2019)
- Biology (e.g. Rives et al., 2019, Madani et al., 2020)
- Vision and Language (e.g. Tan et al., 2019, Lu et al., 2019, Chen et al., 2020)
- Computer Vision (e.g. Ramachandran et al., 2019, Dosovitskiy et al., 2020)



Transformers in NLP

Transformers are ubiquitous in NLP.

Large-scale pre-training has been enormously successful (e.g. BERT, ALBERT, T5, GPT-3).

Models are typically used in 3 scenarios:

Transformers in NLP

Transformers are ubiquitous in NLP.

Large-scale pre-training has been enormously successful (e.g. BERT, ALBERT, T5, GPT-3).

Models are typically used in 3 scenarios:

Pre-training

- Large corpus (e.g. web crawled data)
- Typically unsupervised (e.g. masked language modeling)
- Usually runs in GPUs or TPUs

Transformers in NLP

Transformers are ubiquitous in NLP.

Large-scale pre-training has been enormously successful (e.g. BERT, ALBERT, T5, GPT-3).

Models are typically used in 3 scenarios:

Pre-training

- Large corpus (e.g. web crawled data)
- Typically unsupervised (e.g. masked language modeling)
- Usually runs in GPUs or TPUs

Fine-tuning

- Smaller corpus
- Typically supervised (e.g. question answering, natural language inference)
- Usually runs in GPUs or TPUs

Transformers in NLP

Transformers are ubiquitous in NLP.

Large-scale pre-training has been enormously successful (e.g. BERT, ALBERT, T5, GPT-3).

Models are typically used in 3 scenarios:

Pre-training

- Large corpus (e.g. web crawled data)
- Typically unsupervised (e.g. masked language modeling)
- Usually runs in GPUs or TPUs

Fine-tuning

- Smaller corpus
- Typically supervised (e.g. question answering, natural language inference)
- Usually runs in GPUs or TPUs

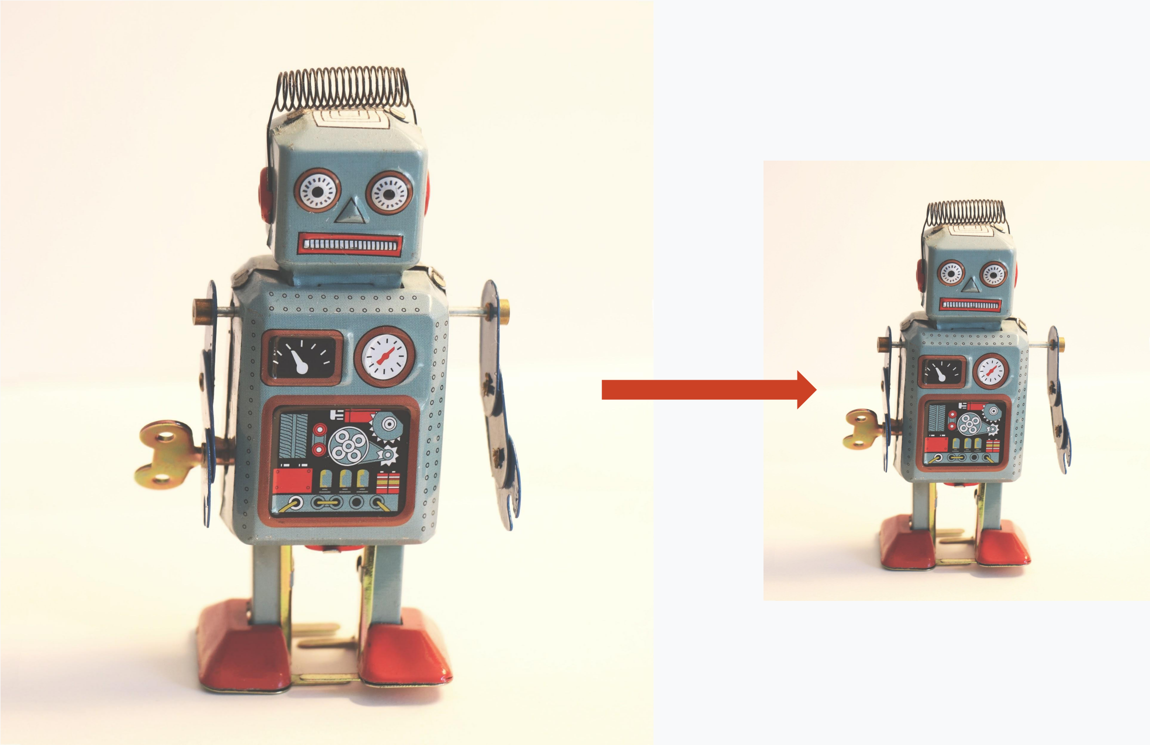
Production

- Inference
- Usually runs in CPUs, sometimes in mobile devices

03

Core Techniques

Knowledge Distillation

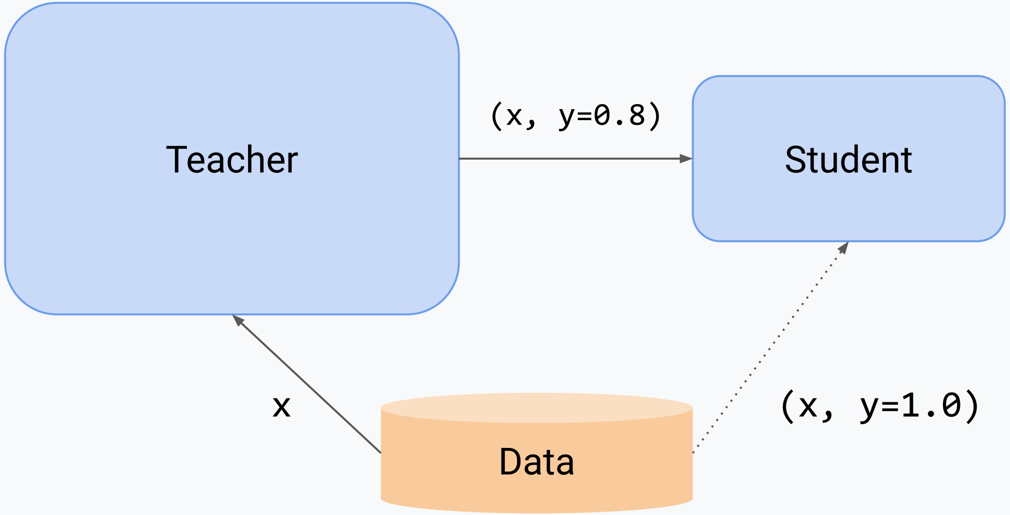


Source: unsplash.com

Knowledge Distillation

Hinton et al., 2015

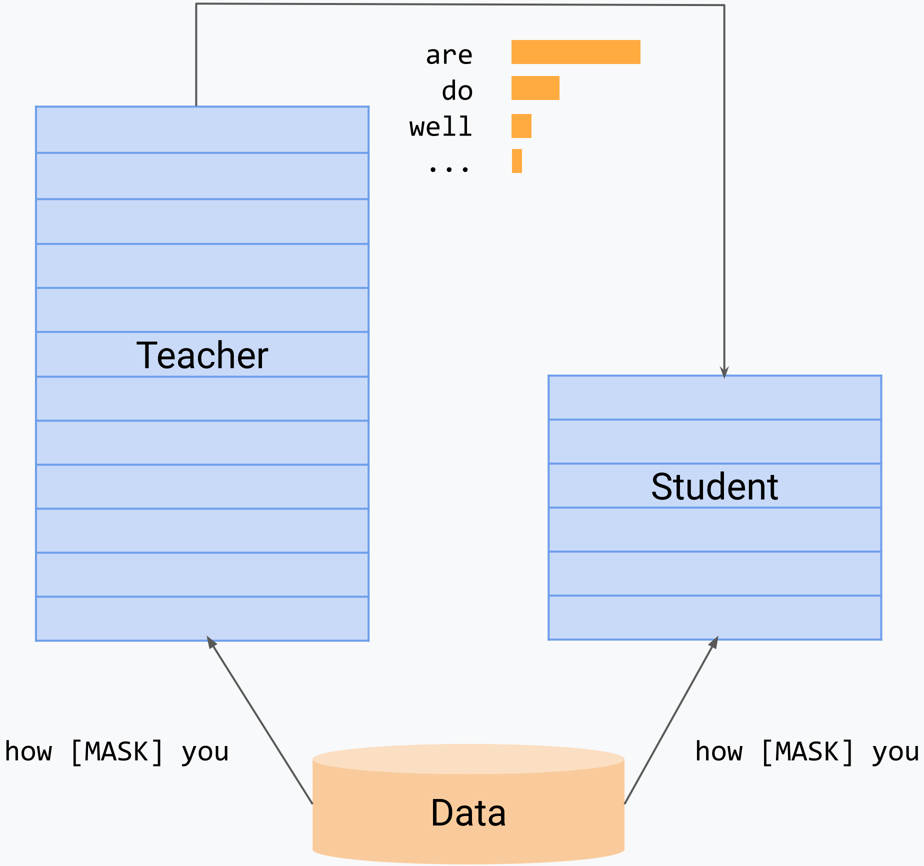
Distilling the Knowledge in a Neural Network



Knowledge Distillation
for Pre-training

Sanh et al., 2019

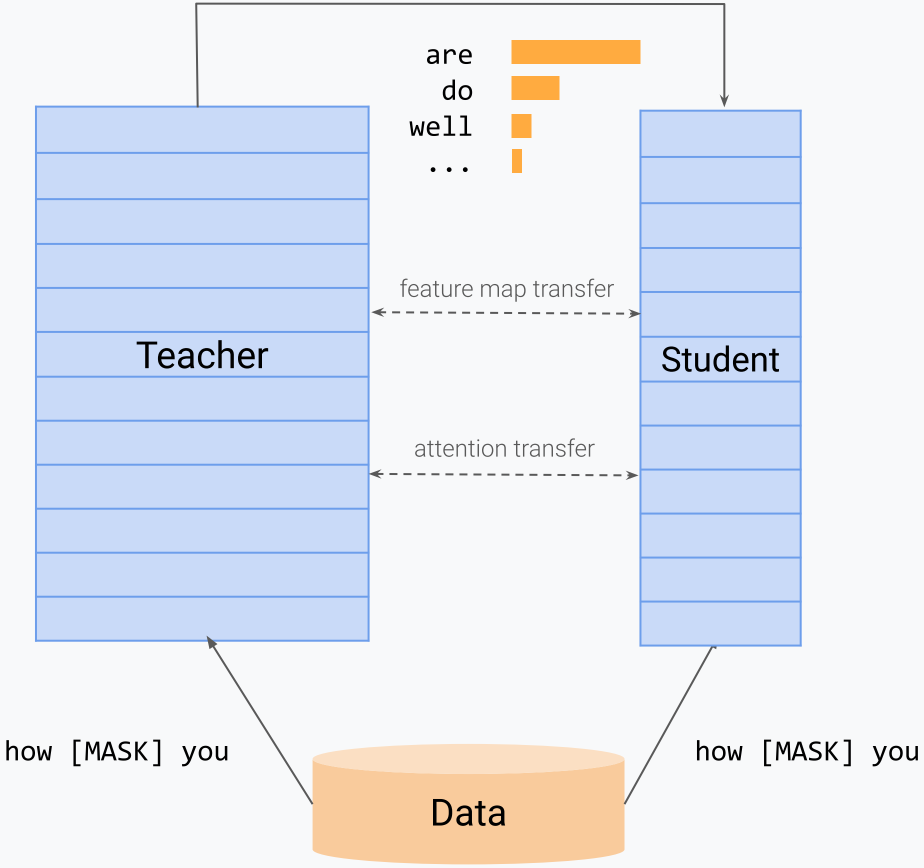
DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter



Knowledge Distillation
for Pre-training

Sun et al., 2019

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

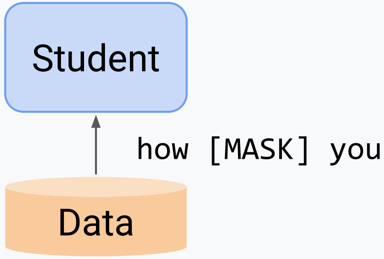


Knowledge Distillation for Fine-Tuning

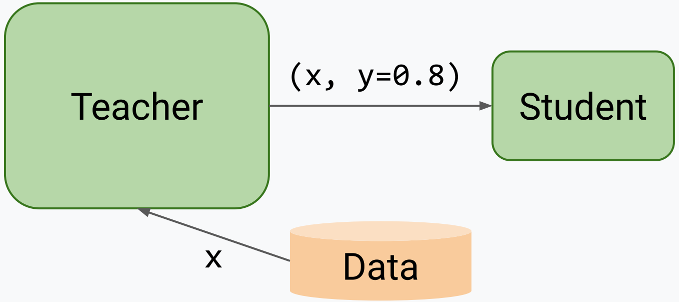
Turc et al., 2019

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

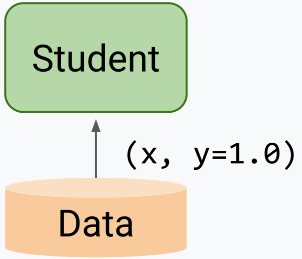
1 Regular Pre-training



2 Fine-tuning via distillation



3 (Optional) regular fine-tuning

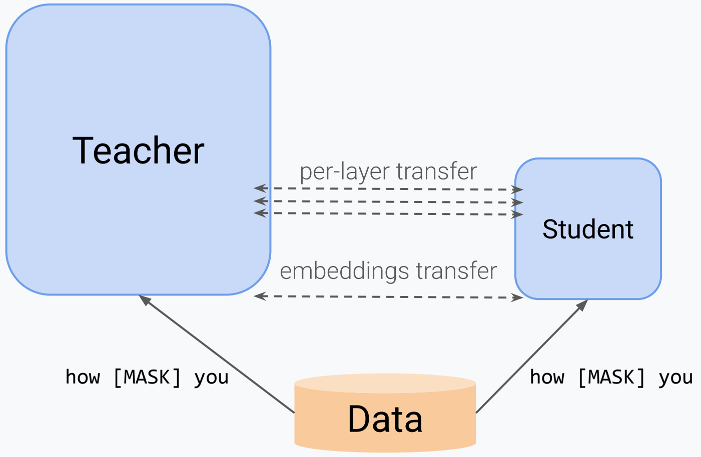


Knowledge Distillation for Pre-training and Fine-tuning

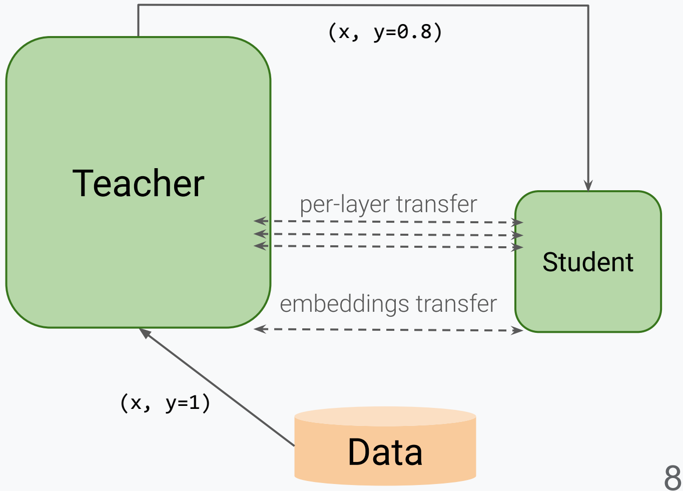
Jiao et al., 2019

TinyBERT: Distilling BERT for Natural Language Understanding

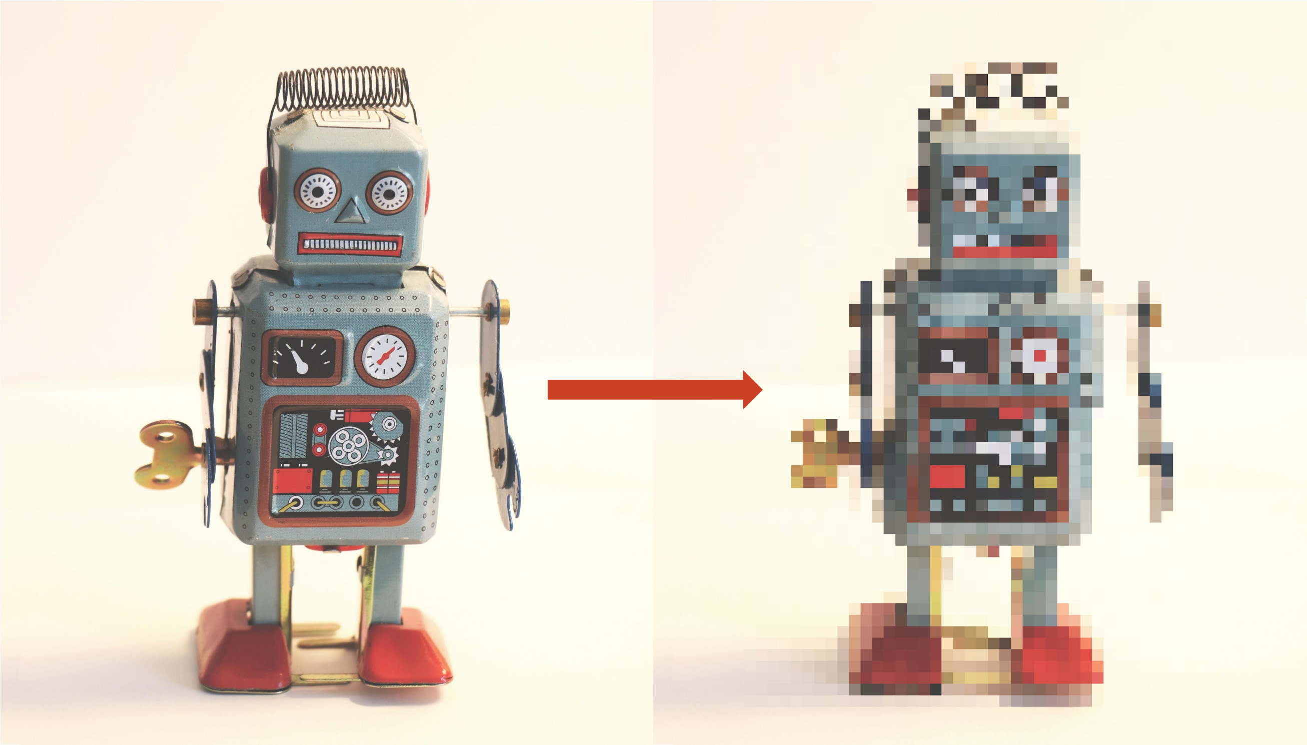
1 Pre-training via distillation



2 Fine-training via distillation



Quantization



Source: unsplash.com

Quantization

Definition

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

real-valued tensor (activation or weight)

quantization precision

quantization operator

Quantization

Definition

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

real-valued tensor (activation or weight)

quantization precision

quantization operator

Linear Quantization

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

zero point

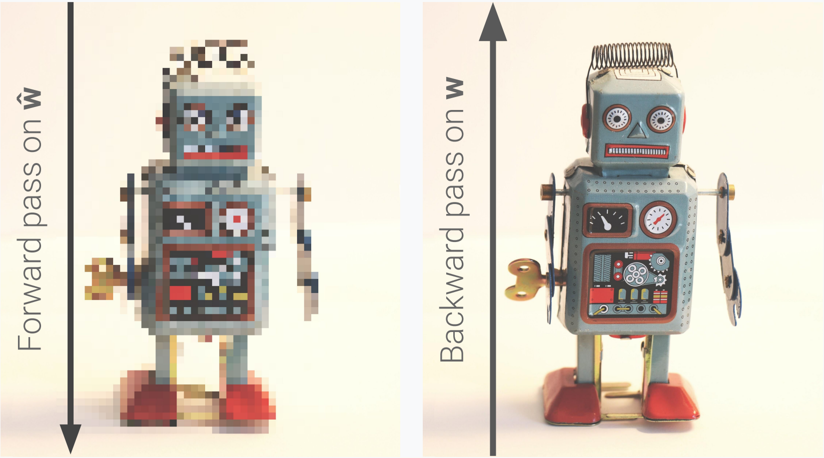
scaling factor

Quantization

Jacob et al., 2017

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

Quantization-Aware Training


Forward pass on $\hat{\mathbf{w}}$
Backward pass on $\mathbf{w}$

$$\mathbf{w}^{t+1} = \mathrm{UpdateParameter}(\mathbf{w}^{t}, \frac{\partial L}{\partial \hat{\mathbf{w}}^{t}}, \eta^{t})$$

Quantization

Zafrir et al., 2019

Q8BERT: Quantized 8Bit BERT

Shen et al., 2019

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

- **Q8BERT**: symmetric linear quantization:

  $Q(z) = \mathrm{clamp}(\lfloor z \times S^z \rceil, -127, +127)$, where S^z is a statistic computed during or post-training.

- **Q-BERT**: uniform quantization to $\{0, \ldots, 2^k-1\}$ 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)

Quantization
with Distillation

Zhang et al., 2020

TernaryBERT: Distillation-aware Ultra-low Bit BERT

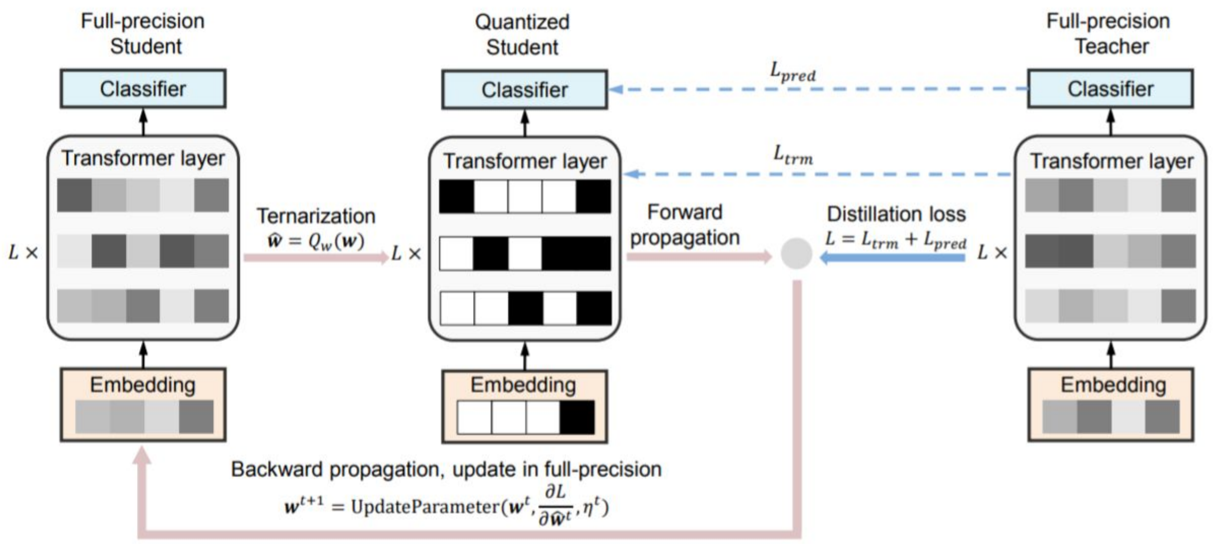
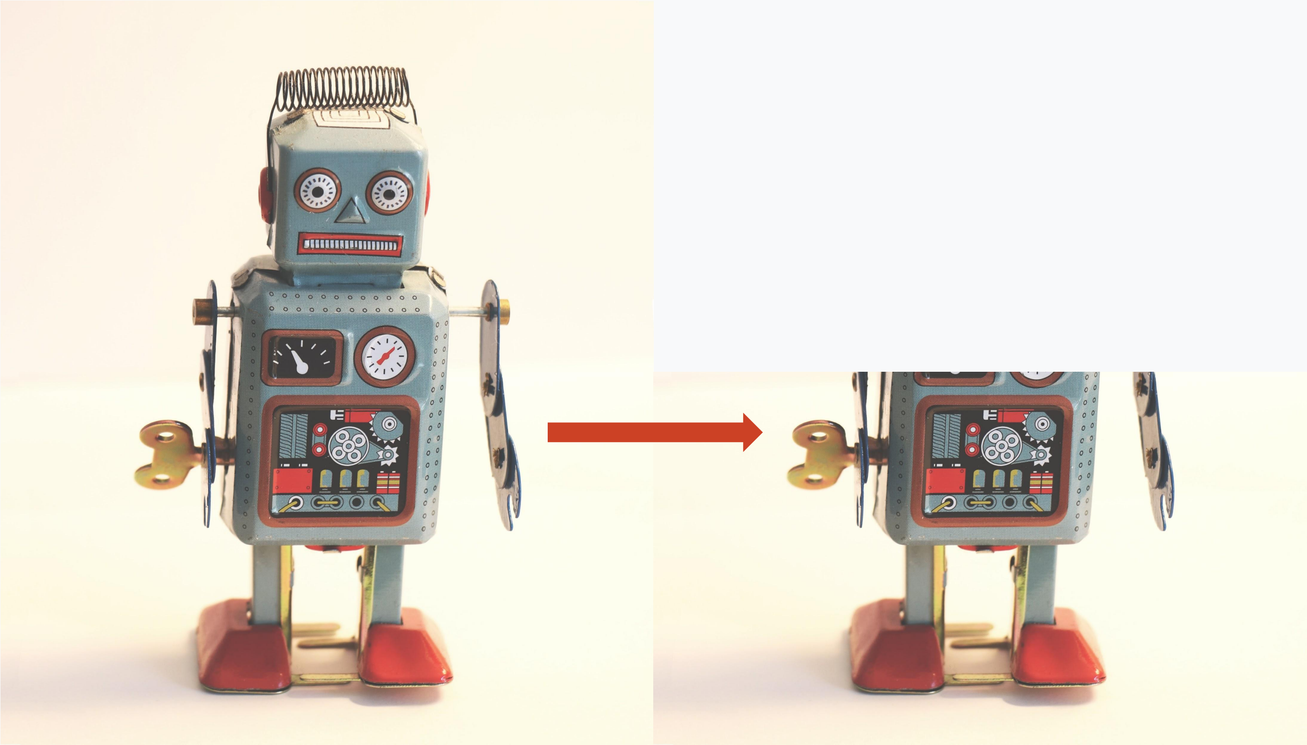


Figure 2: Depiction of the proposed distillation-aware ternarization of BERT model.

Pruning



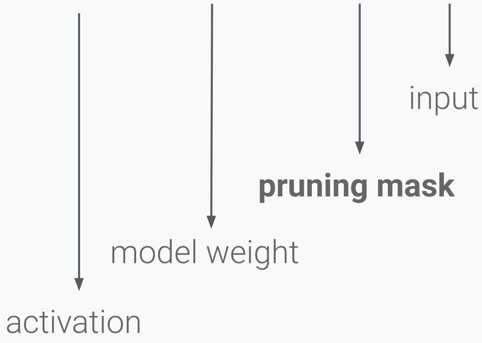
Source: unsplash.com

Pruning

Definition

Pruning removes “unimportant” weights from a network:

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



input

pruning mask

model weight

activation

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
Early Work

LeCun et al., 1990

OBD: Optimal Brain Damage

Hassibi and Stork, 1993

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

Pruning based on second-order derivatives

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

Pruning
Early Work

LeCun et al., 1990

OBD: Optimal Brain Damage

Hassibi and Stork, 1993

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

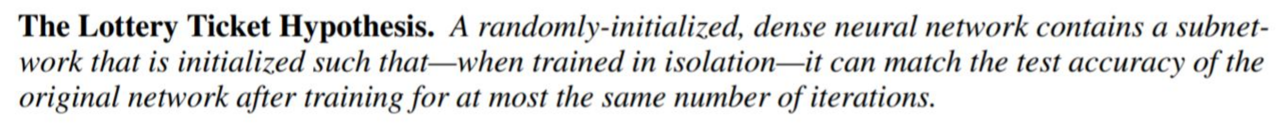
Pruning based on second-order derivatives

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

Why do we not train this smaller architecture instead?

Pruning
The LTH

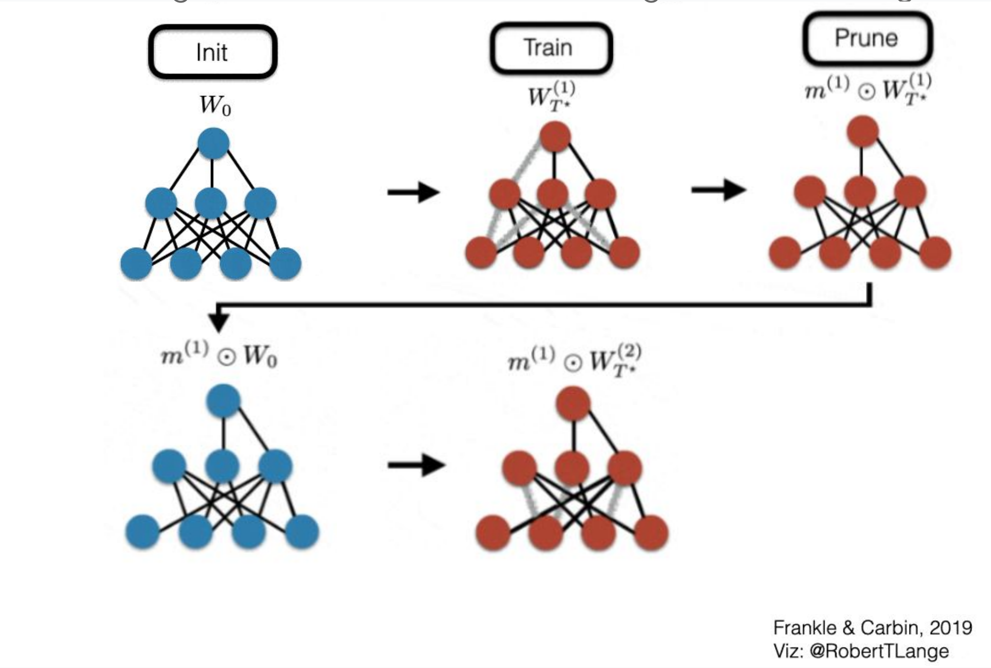


The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

Frankle and Carbin, 2018

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

Searching for Tickets: **One-Shot** Magnitude Pruning



Frankle & Carbin, 2019
Viz: @RobertTLange

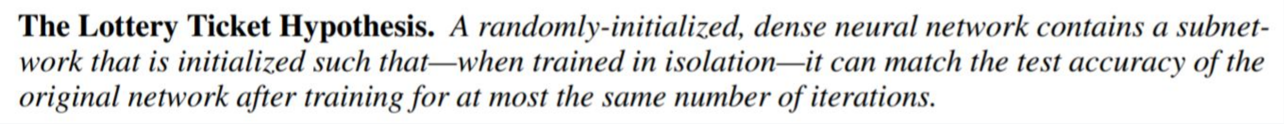
Source: https://roberttlange.github.io/posts/2020/06/lottery-ticket-hypothesis/

Pruning

The LTH

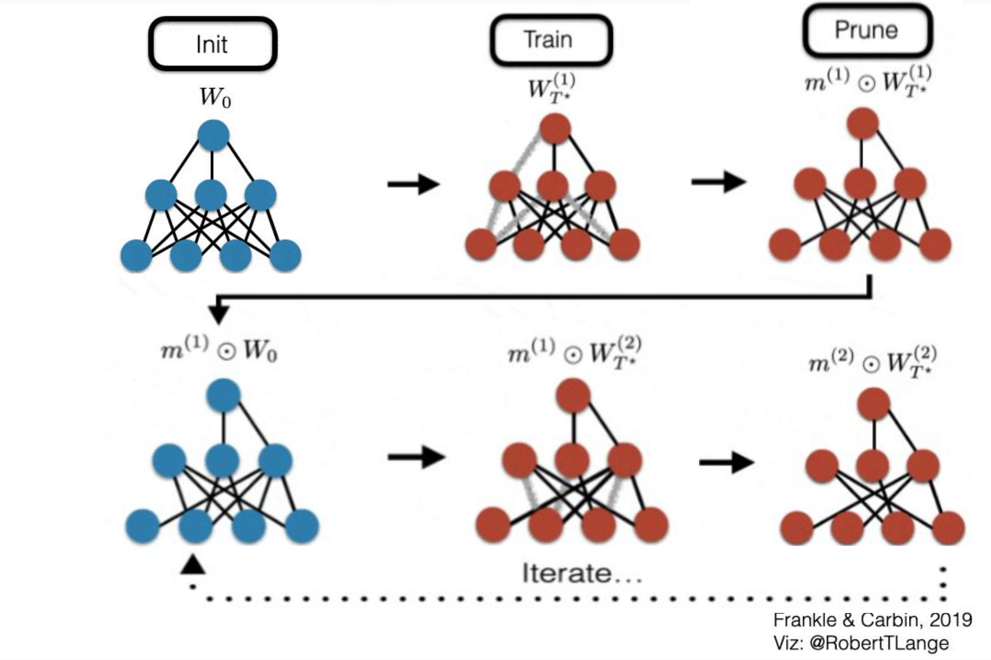
Frankle and Carbin, 2018

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: **Iterative** Magnitude Pruning


Init
W_0
Train
$W_{T^*}^{(1)}$
Prune
$m^{(1)} \odot W_{T^*}^{(1)}$
$m^{(1)} \odot W_0$
$m^{(1)} \odot W_{T^*}^{(2)}$
$m^{(2)} \odot W_{T^*}^{(2)}$
Iterate…

Frankle & Carbin, 2019
Viz: @RobertTLange

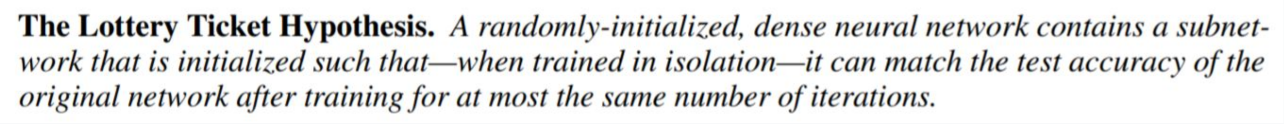
Source: https://roberttlange.github.io/posts/2020/06/lottery-ticket-hypothesis/

Pruning

The LTH

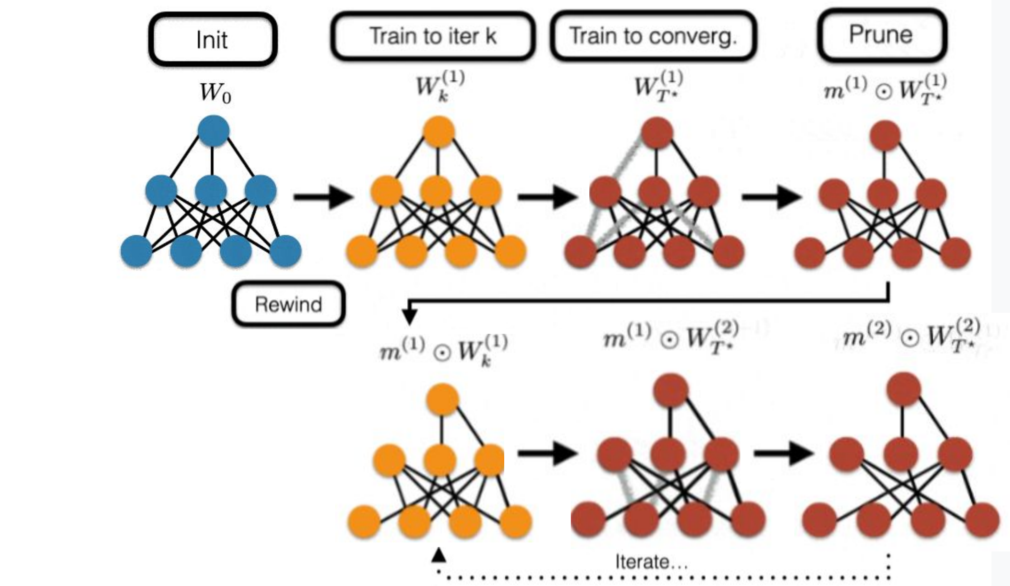
Frankle et al, 2019

Stabilizing the Lottery Ticket Hypothesis



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: **Iterative** Magnitude Pruning **with Rewinding**



Init
Train to iter k
Train to converg.
Prune
W_0
$W_k^{(1)}$
$W_{T^*}^{(1)}$
$m^{(1)} \odot W_{T^*}^{(1)}$
Rewind
$m^{(1)} \odot W_k^{(1)}$
$m^{(1)} \odot W_{T^*}^{(2)}$
$m^{(2)} \odot W_{T^*}^{(2)}$
Iterate...

Frankle et al., 2019
Viz: @RobertTLange

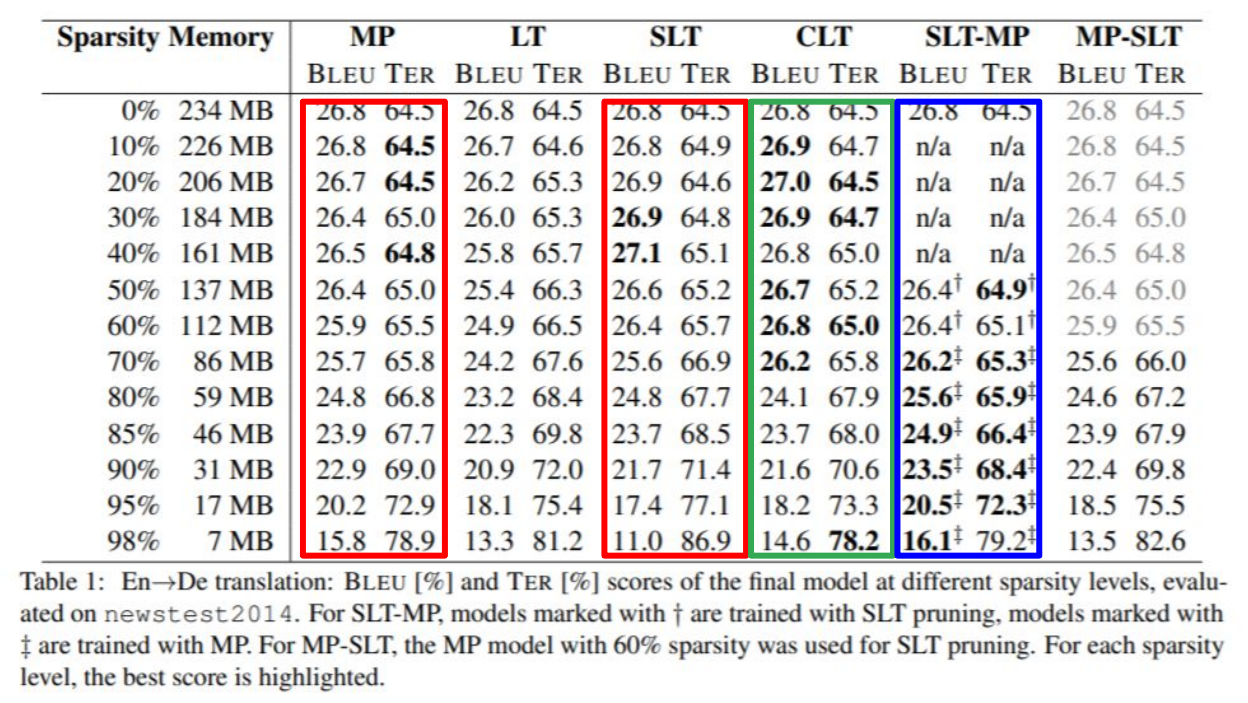
Source: https://roberttlange.github.io/posts/2020/06/lottery-ticket-hypothesis/

Pruning

The LTH, ctd

Brix et al., 2020

Successfully Applying the Stabilized Lottery Ticket Hypothesis to the Transformer Architecture



Sparsity	Memory	MP		LT		SLT		CLT		SLT-MP		MP-SLT	
		BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER
0%	234 MB	26.8	64.5	26.8	64.5	26.8	64.5	26.8	64.5	26.8	64.5	26.8	64.5
10%	226 MB	26.8	**64.5**	26.7	64.6	26.8	64.9	**26.9**	64.7	n/a	n/a	26.8	64.5
20%	206 MB	26.7	**64.5**	26.2	65.3	26.9	64.6	**27.0**	**64.5**	n/a	n/a	26.7	64.5
30%	184 MB	26.4	65.0	26.0	65.3	**26.9**	64.8	**26.9**	**64.7**	n/a	n/a	26.4	65.0
40%	161 MB	26.5	**64.8**	25.8	65.7	**27.1**	65.1	26.8	65.0	n/a	n/a	26.5	64.8
50%	137 MB	26.4	65.0	25.4	66.3	26.6	65.2	**26.7**	65.2	26.4†	**64.9**†	26.4	65.0
60%	112 MB	25.9	65.5	24.9	66.5	26.4	65.7	**26.8**	**65.0**	26.4†	65.1†	25.9	65.5
70%	86 MB	25.7	65.8	24.2	67.6	25.6	66.9	**26.2**	65.8	**26.2**‡	**65.3**‡	25.6	66.0
80%	59 MB	24.8	66.8	23.2	68.4	24.8	67.7	24.1	67.9	**25.6**‡	**65.9**‡	24.6	67.2
85%	46 MB	23.9	67.7	22.3	69.8	23.7	68.5	23.7	68.0	**24.9**‡	**66.4**‡	23.9	67.9
90%	31 MB	22.9	69.0	20.9	72.0	21.7	71.4	21.6	70.6	**23.5**‡	**68.4**‡	22.4	69.8
95%	17 MB	20.2	72.9	18.1	75.4	17.4	77.1	18.2	73.3	**20.5**‡	**72.3**‡	18.5	75.5
98%	7 MB	15.8	78.9	13.3	81.2	11.0	86.9	14.6	**78.2**	**16.1**‡	79.2‡	13.5	82.6

Table 1: En→De translation: BLEU [%] and TER [%] scores of the final model at different sparsity levels, evaluated on newstest2014. For SLT-MP, models marked with † are trained with SLT pruning, models marked with ‡ are trained with MP. For MP-SLT, the MP model with 60% sparsity was used for SLT pruning. For each sparsity level, the best score is highlighted.

MP = Magnitude Pruning
LT = Lottery Ticket
SLT = Stabilized Lottery Ticket
CLT = Constant Lottery Ticket

Pruning

Sanh et al., 2020

Movement Pruning: Adaptive Sparsity by Fine-Tuning

Movement Pruning

- **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.

Pruning
& Hardware

Hooker, 2020

The Hardware Lottery

On standard hardware:

	Unstructured Pruning	Structured Pruning
Storage	✓	✓
Inference	✗	✓
Flexibility	✓	✗

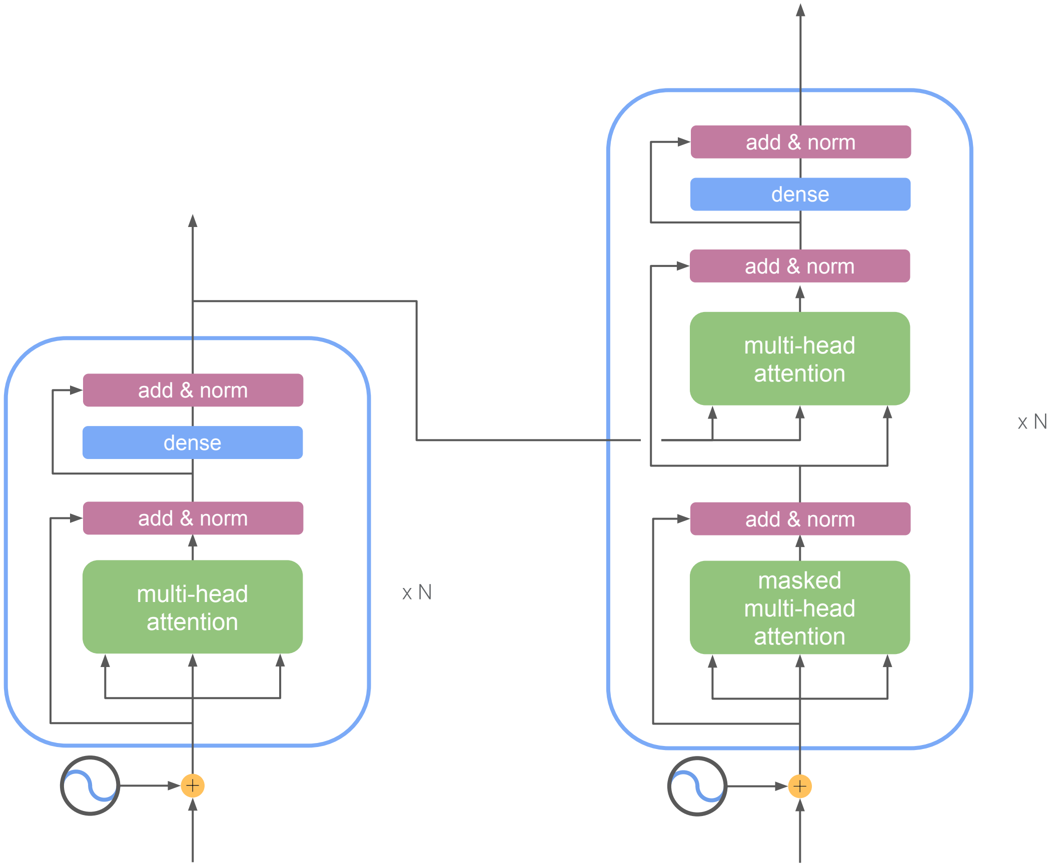
- “In many ways, hardware is catching up to the present state of ML research.”
- There is research for specialized software kernels to support unstructured sparsity (see paper for references).

04

Efficient Attention

Recap

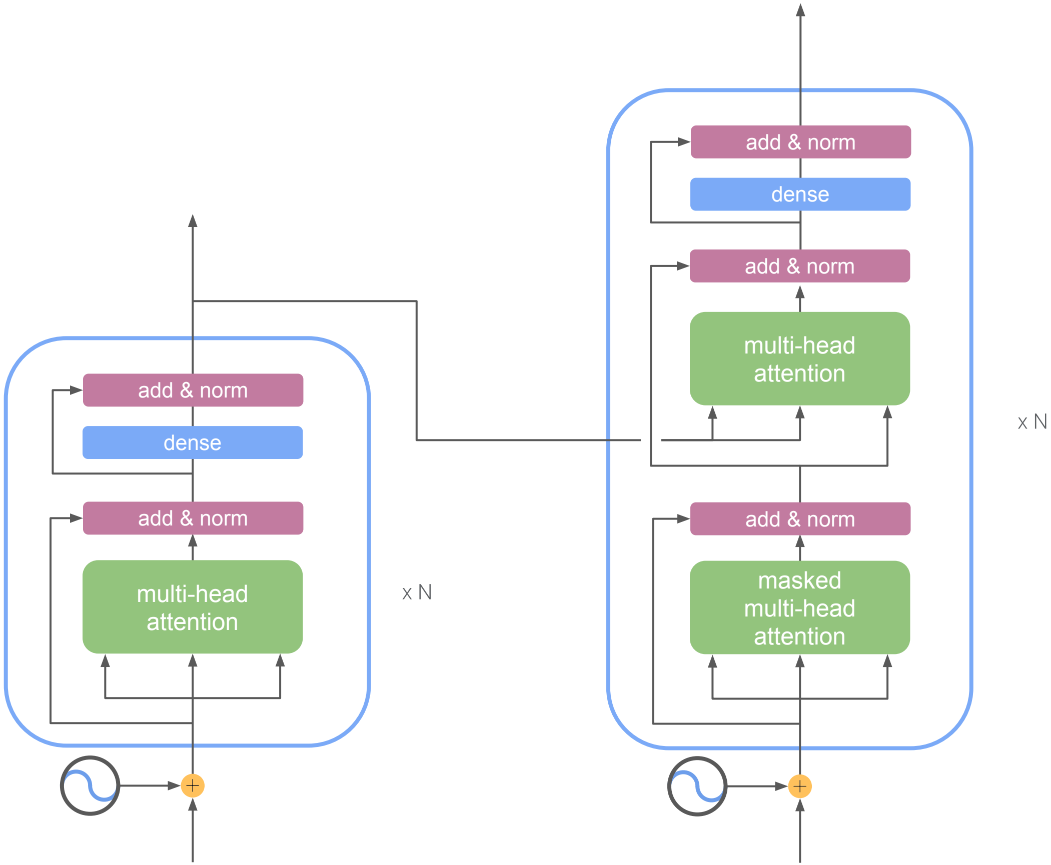
The Transformer architecture



Recap

The Transformer architecture

Quadratic bottleneck in sequence length due to multi-head attention



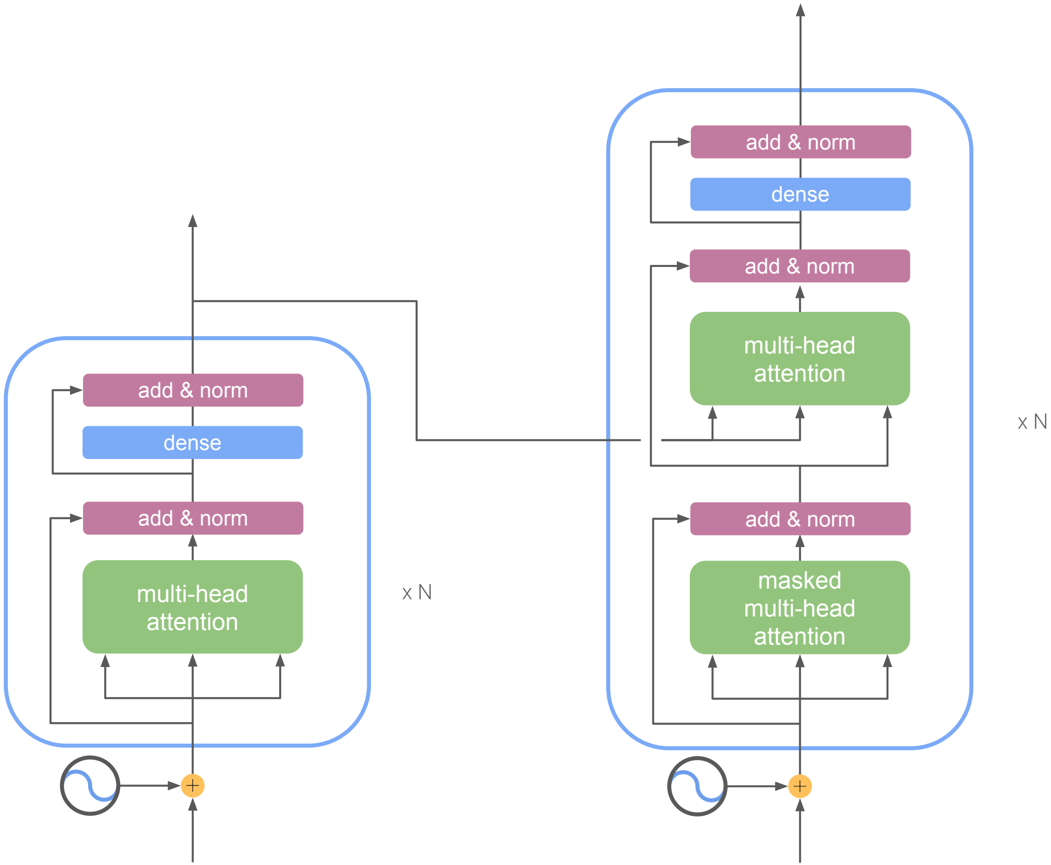
Recap

The Transformer architecture

Quadratic bottleneck in sequence length due to multi-head attention

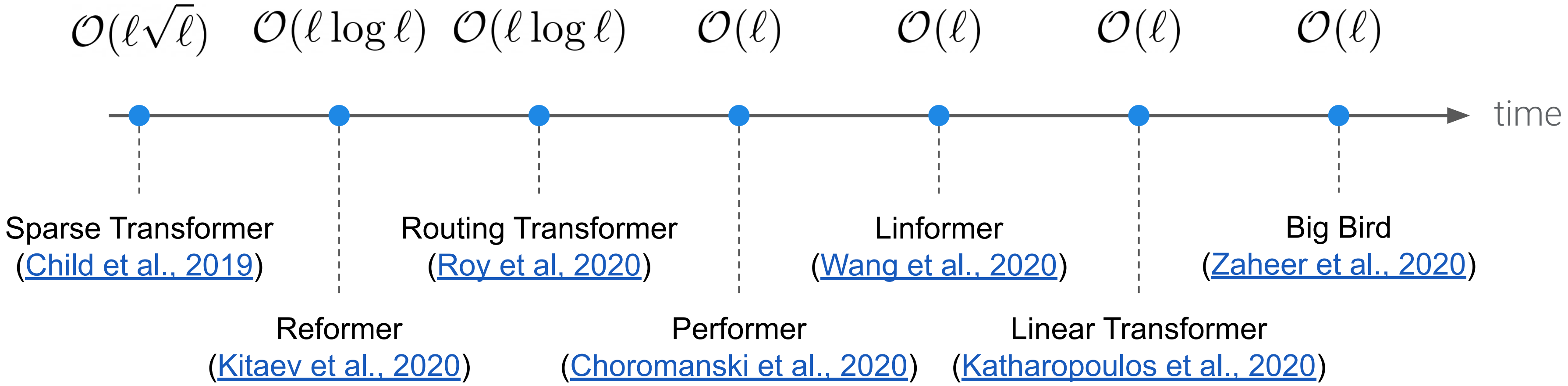
This poses a serious problem when large sequences are required, e.g.:

- Long-range dependencies
- Character-level models
- Speech processing
- High-resolution image processing



Efficient Attention

In the past months, there has been much progress in making self-attention more efficient



$\mathcal{O}(\ell\sqrt{\ell})$ $\mathcal{O}(\ell \log \ell)$ $\mathcal{O}(\ell \log \ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$

time

Sparse Transformer (Child et al., 2019)

Reformer (Kitaev et al., 2020)

Routing Transformer (Roy et al, 2020)

Performer (Choromanski et al., 2020)

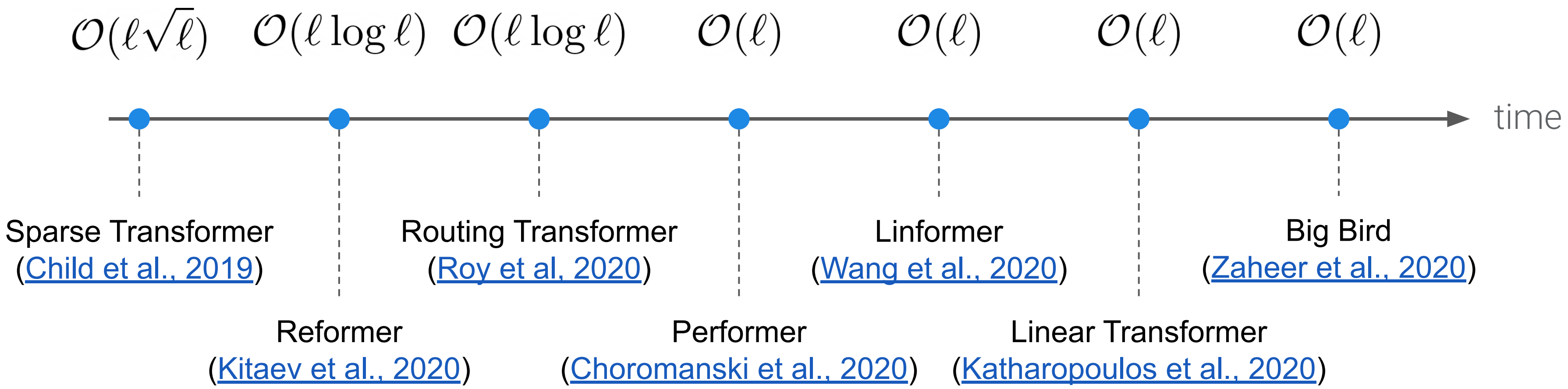
Linformer (Wang et al., 2020)

Linear Transformer (Katharopoulos et al., 2020)

Big Bird (Zaheer et al., 2020)

Efficient Attention

In the past months, there has been much progress in making self-attention more efficient



$\mathcal{O}(\ell\sqrt{\ell})$ $\mathcal{O}(\ell \log \ell)$ $\mathcal{O}(\ell \log \ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$ $\mathcal{O}(\ell)$

time

Sparse Transformer (Child et al., 2019)

Reformer (Kitaev et al., 2020)

Routing Transformer (Roy et al, 2020)

Performer (Choromanski et al., 2020)

Linformer (Wang et al., 2020)

Linear Transformer (Katharopoulos et al., 2020)

Big Bird (Zaheer et al., 2020)

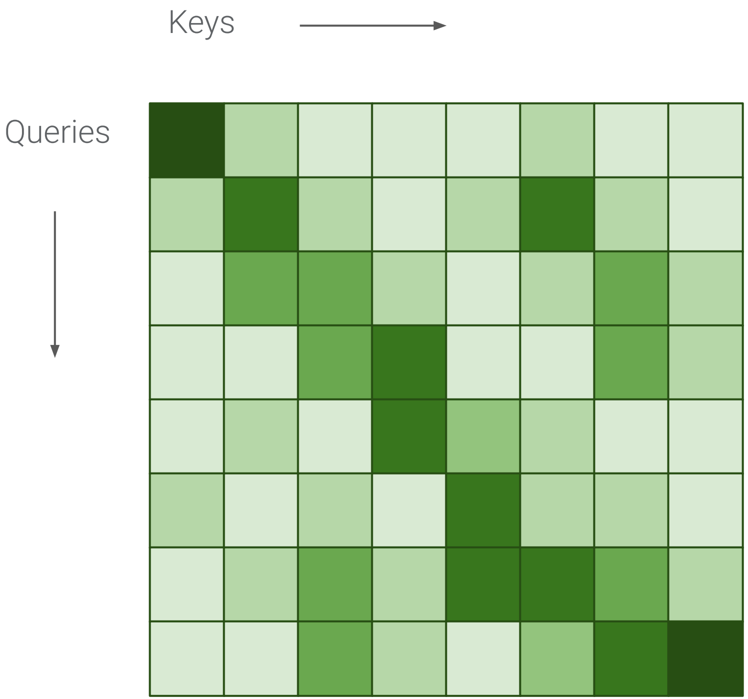
We are going to cover some ideas that make this possible

Beyond a Dense Attention Matrix

Goal:

Approximate the computation of attention

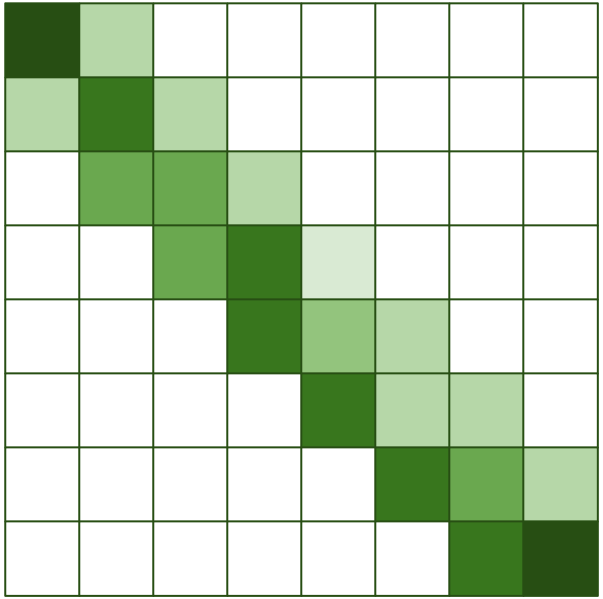
via more efficient operations



Efficient Attention

A wide range of recent techniques!

- **Data-Independent Patterns**
  - Blockwise Transformer (Qiu et al., 2019)
  - Sparse Transformer (Child et al., 2019)
  - Longformer (Beltagy et al., 2020)
  - Big Bird (Zaheer et al., 2020)

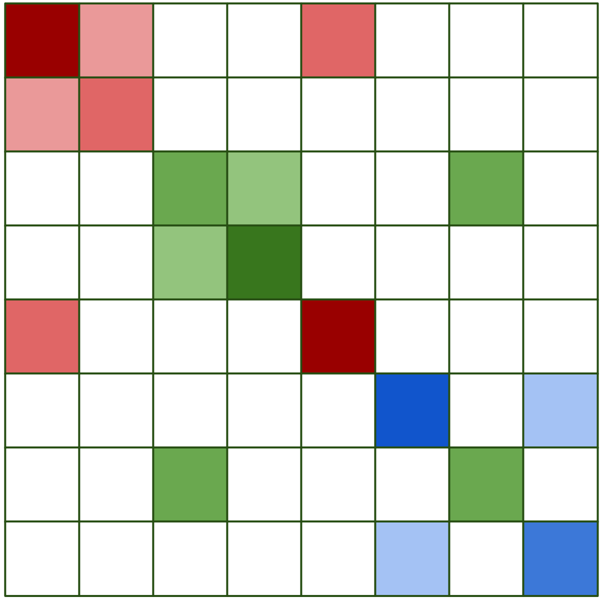


Taxonomy inspired by Tay et al., 2020

Efficient Attention

A wide range of recent techniques!

- Data-Independent Patterns
- **Data-Dependent Patterns**
  - Linformer (Wang et al., 2020)
  - Reformer (Kitaev et al., 2020)
  - Routing Transformer (Roy et al., 2020)
  - Clustered Attention (Vyas et al., 2020)
  - Sinkhorn Transformer (Tay et al., 2020)

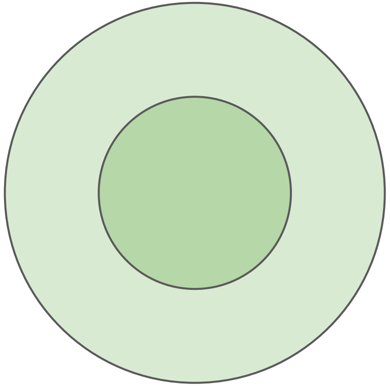


Taxonomy inspired by Tay et al., 2020

Efficient Attention

A wide range of recent techniques!

- Data-Independent Patterns
- Data-Dependent Patterns
- **Kernels and Alternative Attention Mechanisms**
    - Linear Transformer (Katharopoulos et al., 2020)
    - Random Feature Attention (Anonymous, 2020)
    - Performer (Choromanski et al., 2020)
    - Synthesizer (Tay et al., 2020)

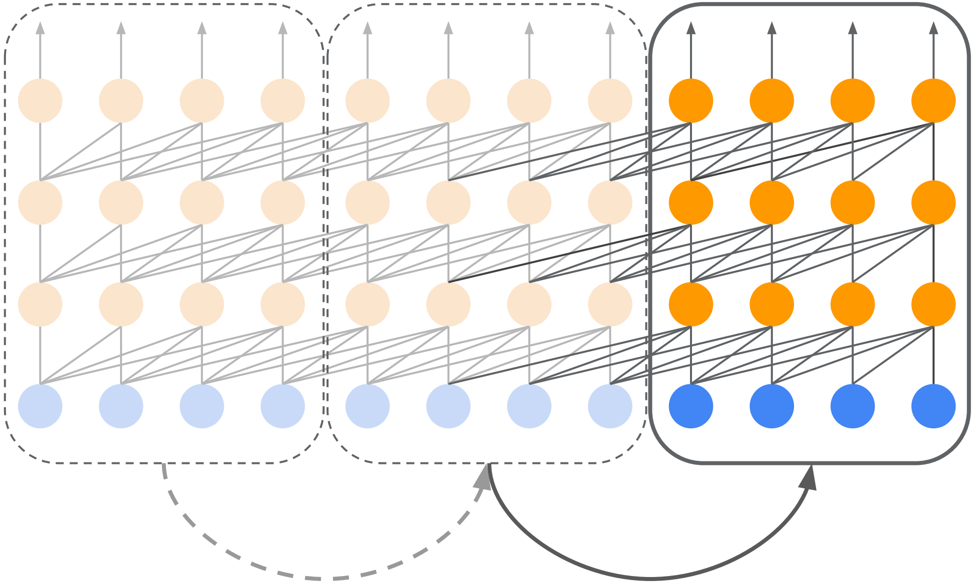


Taxonomy inspired by Tay et al., 2020

Efficient Attention

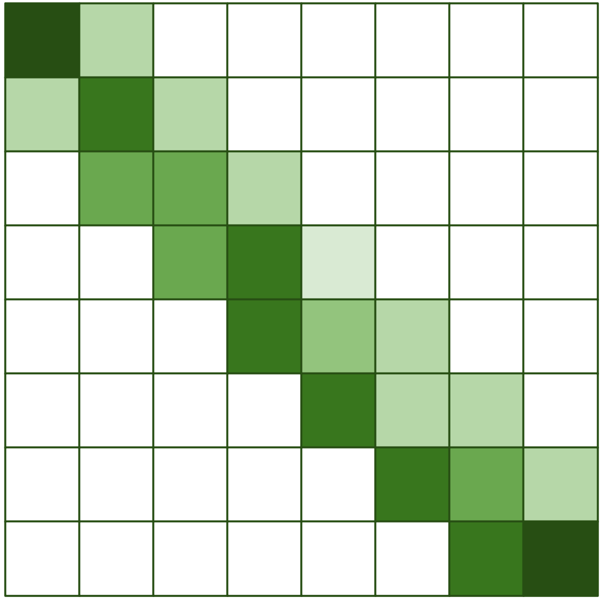
A wide range of recent techniques!

- Data-Independent Patterns
- Data-Dependent Patterns
- Alternative Attention Mechanisms
- **Recurrence**
  - Transformer XL (Dai et al., 2019)
  - Compressive Transformers (Rae et al., 2019)



Taxonomy inspired by Tay et al., 2020

Data-Independent Patterns



Data-Independent Patterns

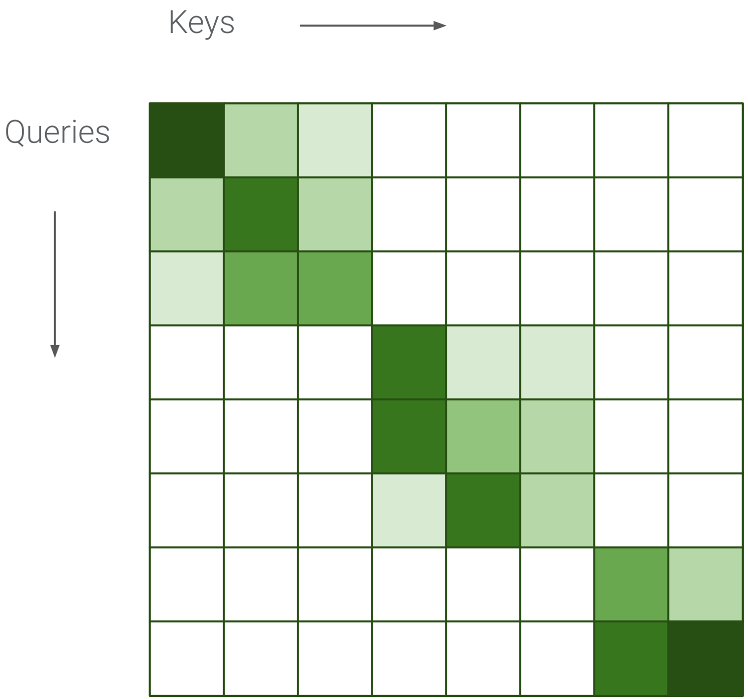
Blockwise Patterns

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

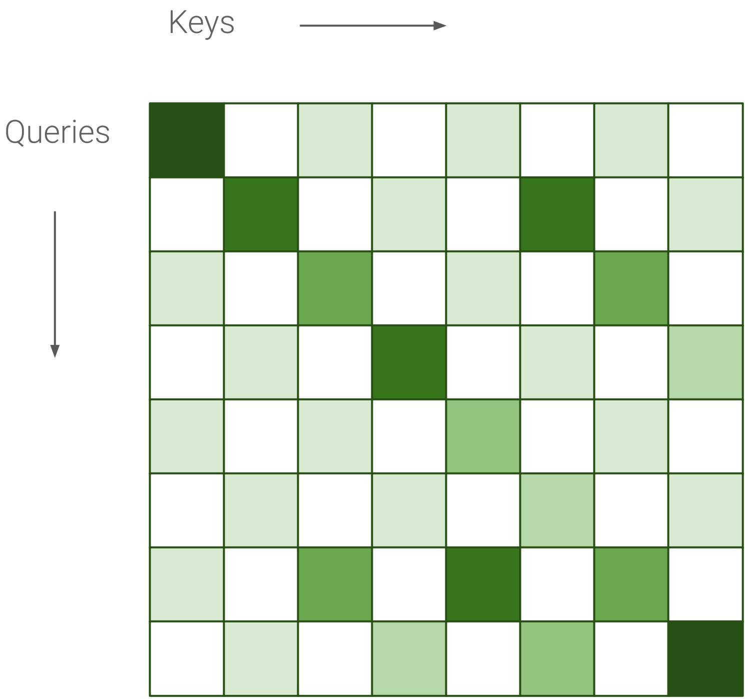
Skip some query/key pairs.

Quadratic in sequence length / stride

Examples:

Sparse Transformer (Child et al., 2019)

Longformer (Beltagy et al, 2020)



Data-Independent Patterns

Diagonal Patterns

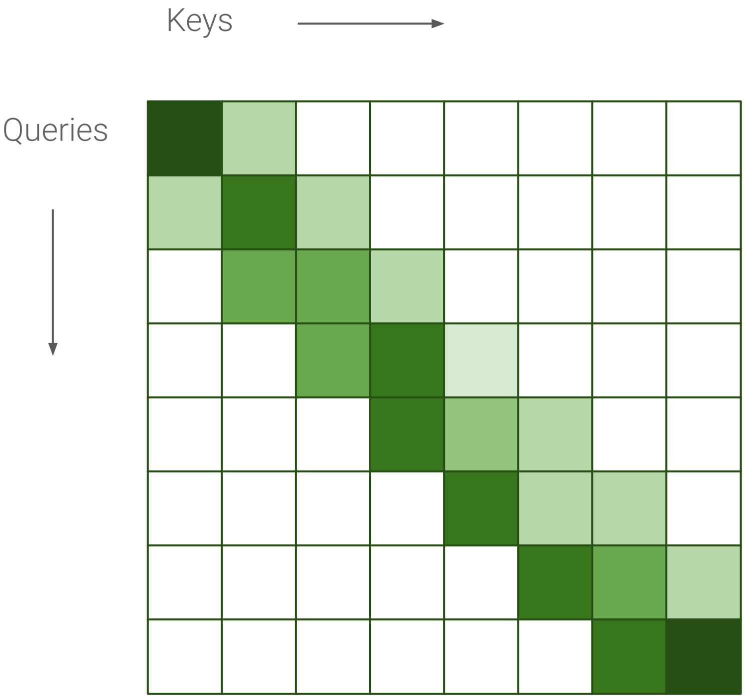
Compute attention over the diagonal.

Linear in sequence length and window size.

Examples:

Longformer (Beltagy et al, 2020)

Big Bird (Zaheer et al., 2020)



Data-Independent Patterns

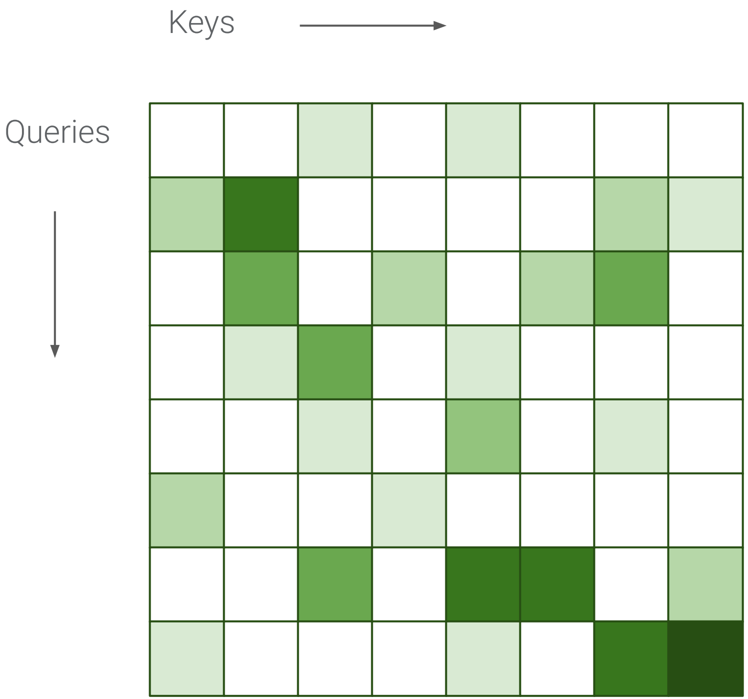
Random Patterns

Compute attention over random query/key pairs.

Linear in number of points.

Examples:

Big Bird (Zaheer et al., 2020)



Data-Independent Patterns

Global Attention

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

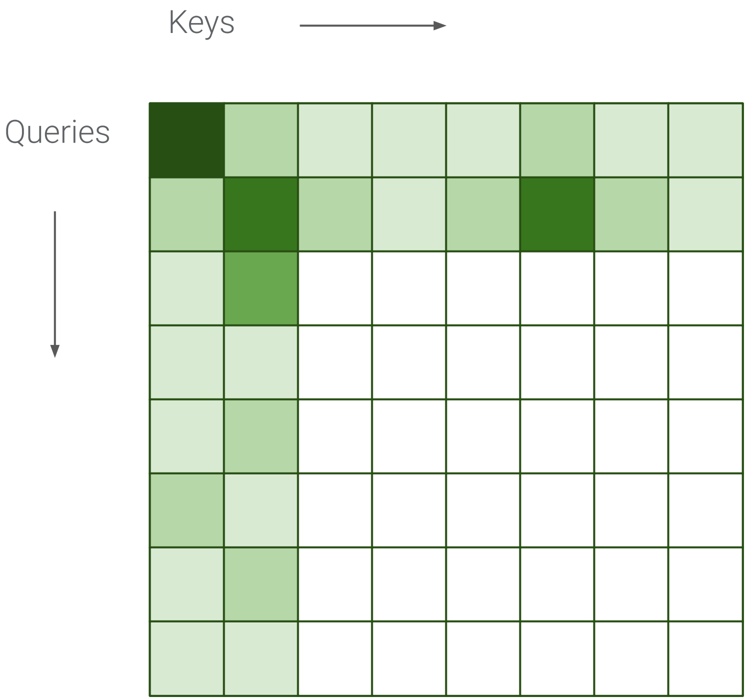
Usually combined with other patterns

Examples:

Big Bird (Zaheer et al., 2020)

Longformer (Beltagy et al., 2020)

ETC (Ainslie et al., 2020)



Data-Independent Patterns

Combination of Patterns

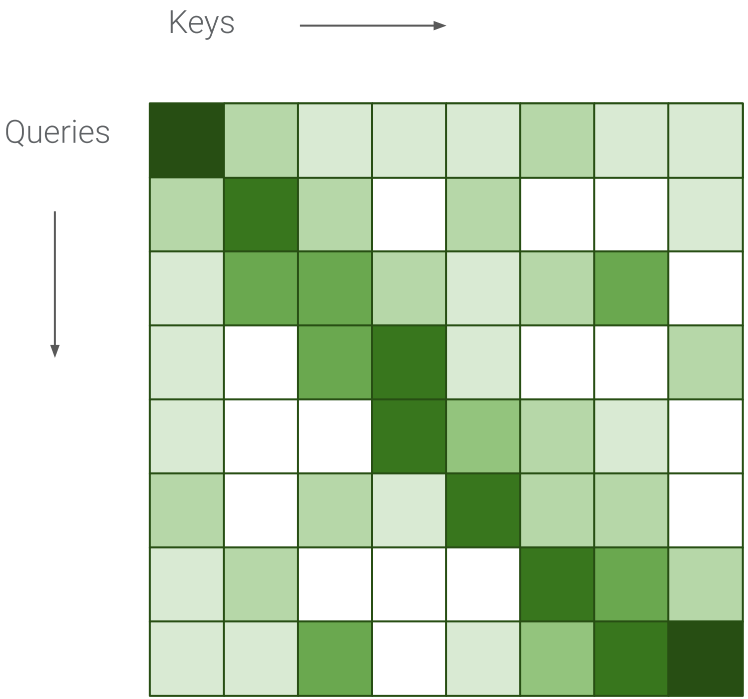
Combine multiple patterns

(e.g. Global + Diagonal + Random)

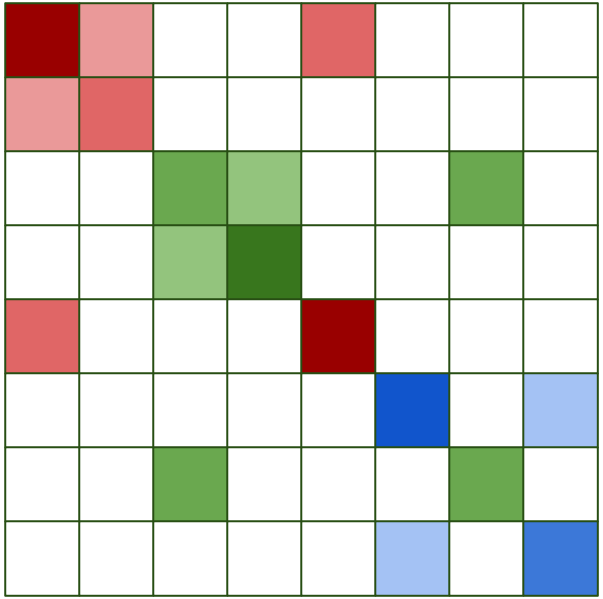
Examples:

Big Bird (Zaheer et al., 2020)

Longformer (Beltagy et al., 2020)



Data-Dependent Patterns



Data-Dependent Patterns

Buckets

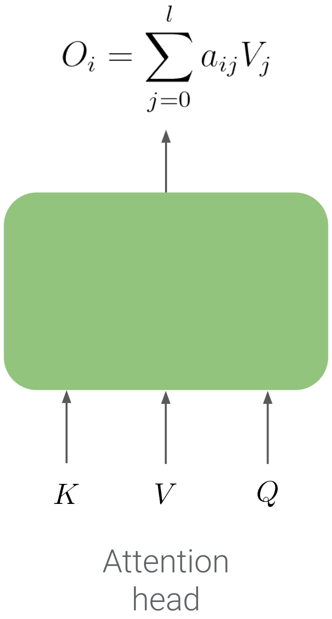
Create buckets/clusters and compute attention within.

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

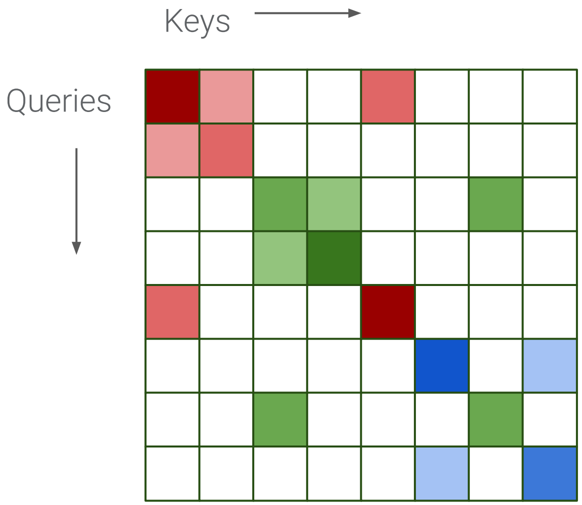
Examples:

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

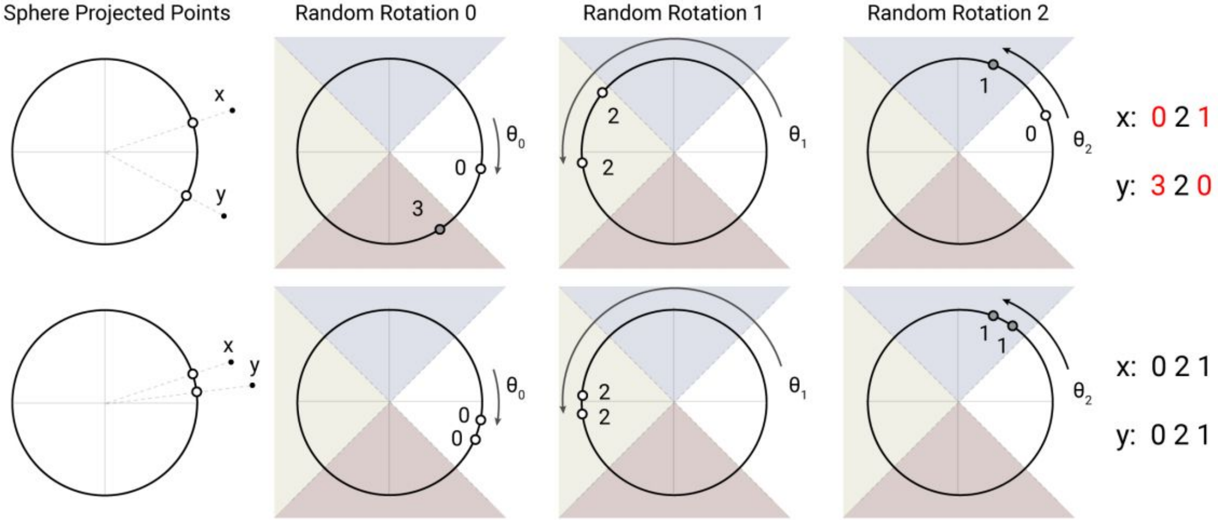
Locality-Sensitive Hashing (LSH)

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

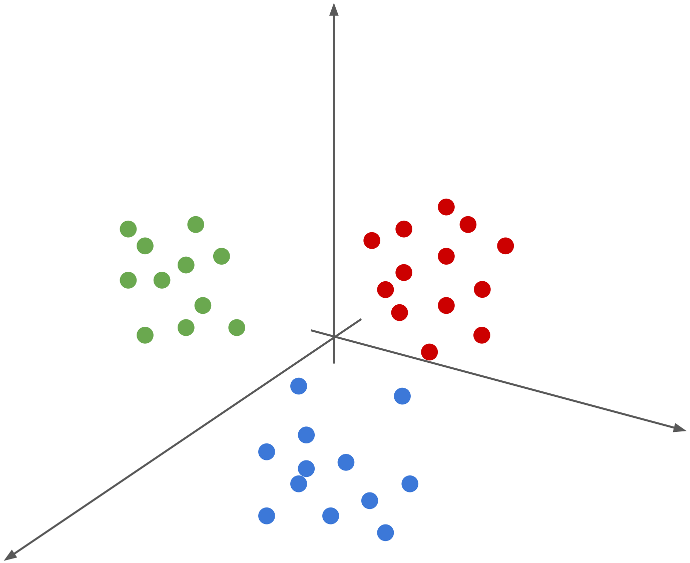
Buckets: Clustering

E.g. online k-means

Examples:

Routing Transformer ([Roy et al., 2020](Roy et al., 2020))

Clustered Attention (Vyas et al., 2020)



Data-Dependent Patterns

Sorting and blocking

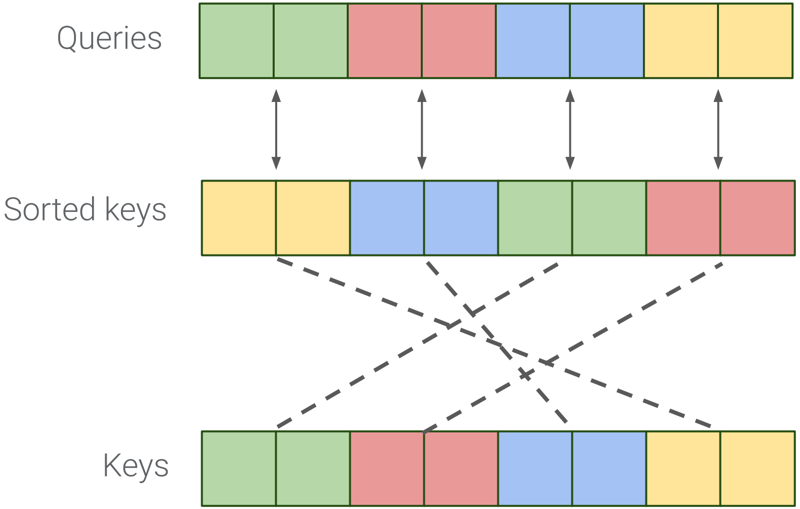
E.g. Sparse Sinkhorn Attention

Key ideas:

- A differentiable sorting network that learns to rearrange blocked inputs, using the Sinkhorn balancing mechanism to create a permutation matrix
- Attention is computed only on local neighborhoods (before and after sorting)

Examples:

Sinkhorn Transformer (Tay et al., 2020)



Data-Dependent Patterns

Compression

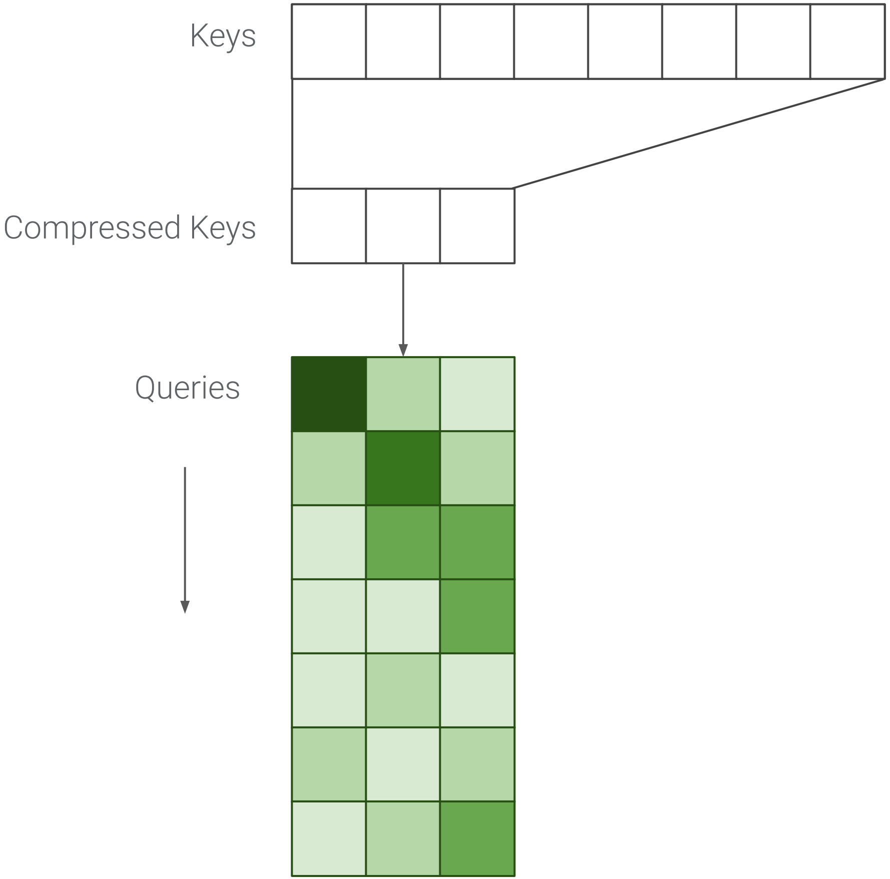
E.g. pooling, strided convolution, low-rank projections with learnable weights

Examples:

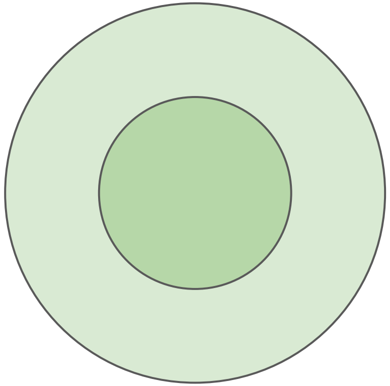
Compressed Attention (Liu et al., 2018)

Linformer (Wang et al., 2020)

Synthesizers (Tay et al., 2020)



Kernels and Alternative Attention Mechanisms



Kernels and Alternative Attention Mechanisms

Kernels

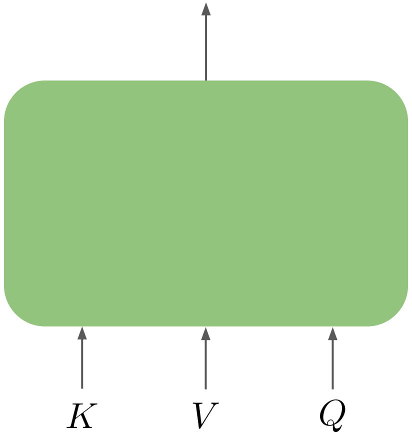
Recap: attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

Standard transformers use dot product attention:

$$\phi(Q_i, K_j) = \exp\left(\frac{Q_i K_j^\top}{\sqrt{d}}\right)$$

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



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

Attention head

Kernels and Alternative Attention Mechanisms

Kernels

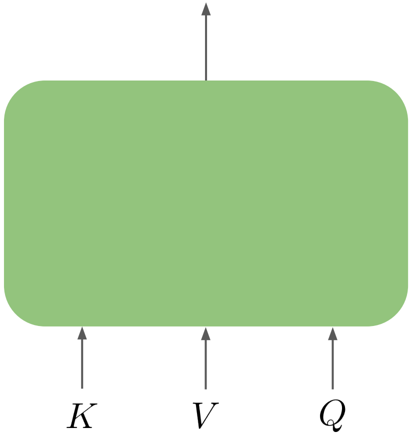
Recap: attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

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

Kernels and Alternative Attention Mechanisms

Kernels

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

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

Recap: attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

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

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

Kernels and Alternative Attention Mechanisms

Kernels

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

Recap: attention in its general form uses a similarity function

$$\phi(Q_i, K_j)$$

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

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

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

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

Kernels and Alternative Attention Mechanisms

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!

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

Kernels and Alternative Attention Mechanisms

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

Kernels and Alternative Attention Mechanisms

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:

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

Then this l x d matrix

Kernels and Alternative Attention Mechanisms

Kernels

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

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

In Katharopoulos et al., 2020:

$$\phi(x) = \mathrm{elu}(x) + 1 \;= \max(\alpha(e^x - 1), 0) + 1$$

Kernels and Alternative Attention Mechanisms

Kernels

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

Random Feature Attention (Anonymous, 2020)

Performer (Choromanski et al., 2020)

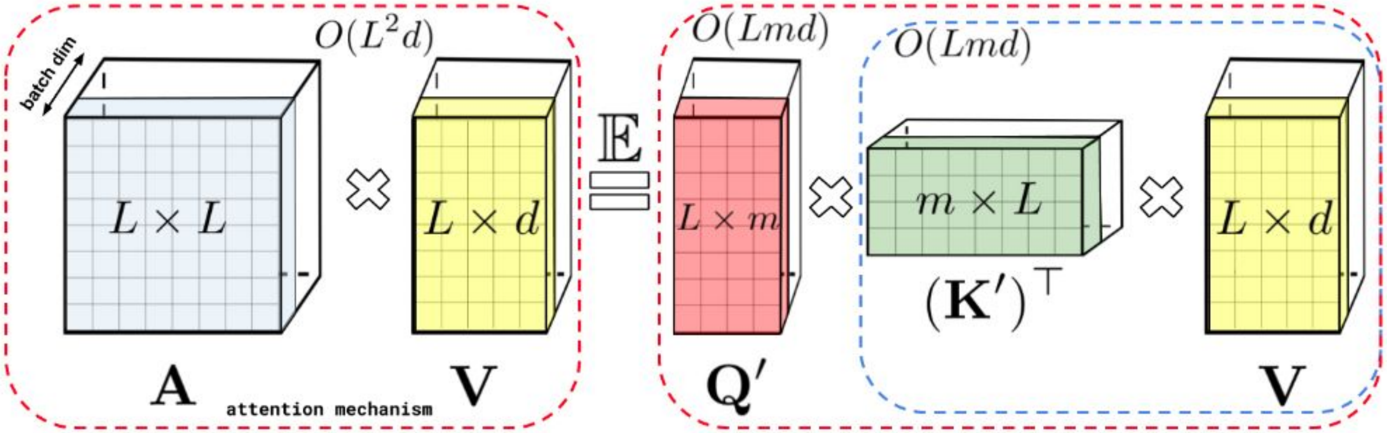
Random features can be used to generate an unbiased estimation of the standard softmax function!

Kernels and Alternative Attention Mechanisms

Performer: Generalized Attention and FAVOR (Choromanski et al., 2020)

Rethink attention as $a_{ij} = g(Q_i^\top)K(Q_i^\top, K_j^\top)h(K_j^\top)$, parametrized by a kernel K and functions g and h

This work presents an unbiased, low-variance approximation of attention via random feature map decompositions, with linear time and space complexity.



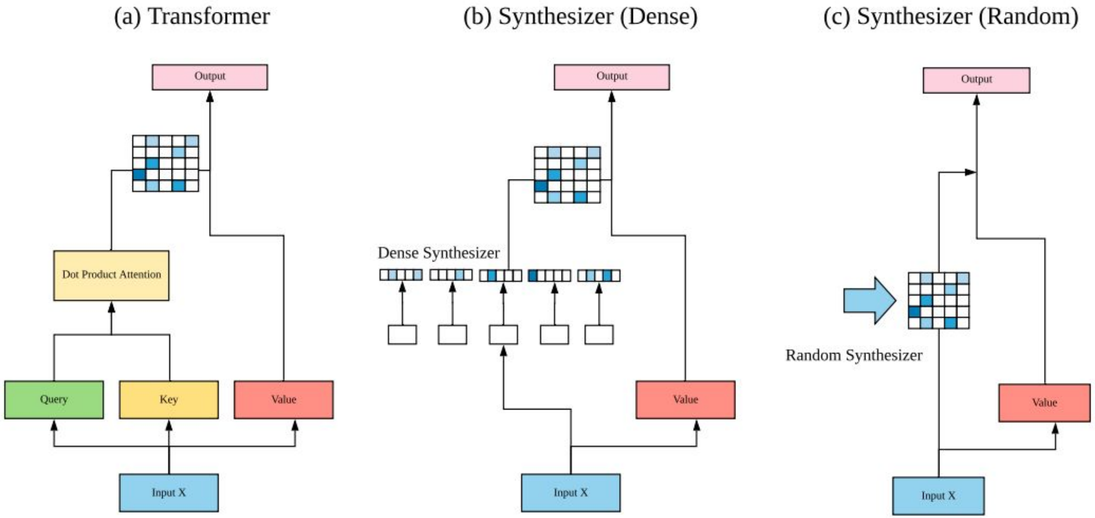
Kernels and Alternative Attention Mechanisms

Synthesizers (Tay et al., 2020)

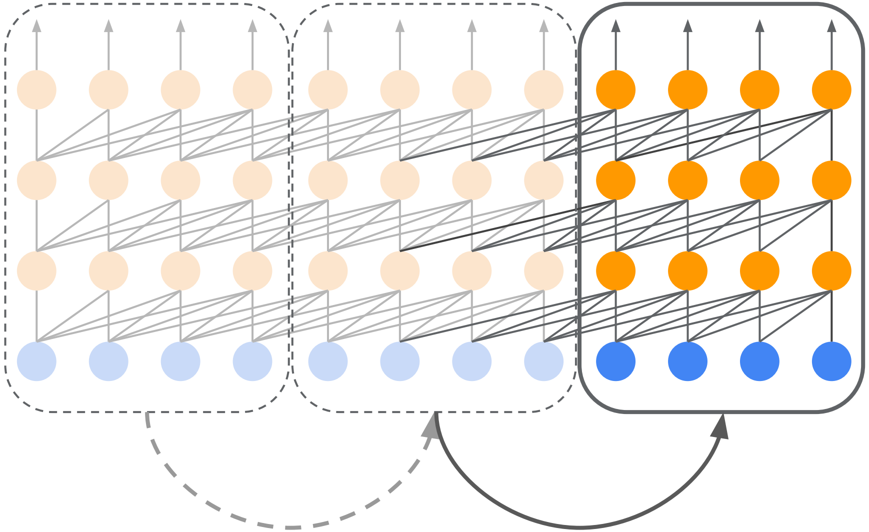
Are token-to-token interactions really necessary?

Random attention matrices are surprisingly competitively!

Low-rank alternatives can be used



Recurrence

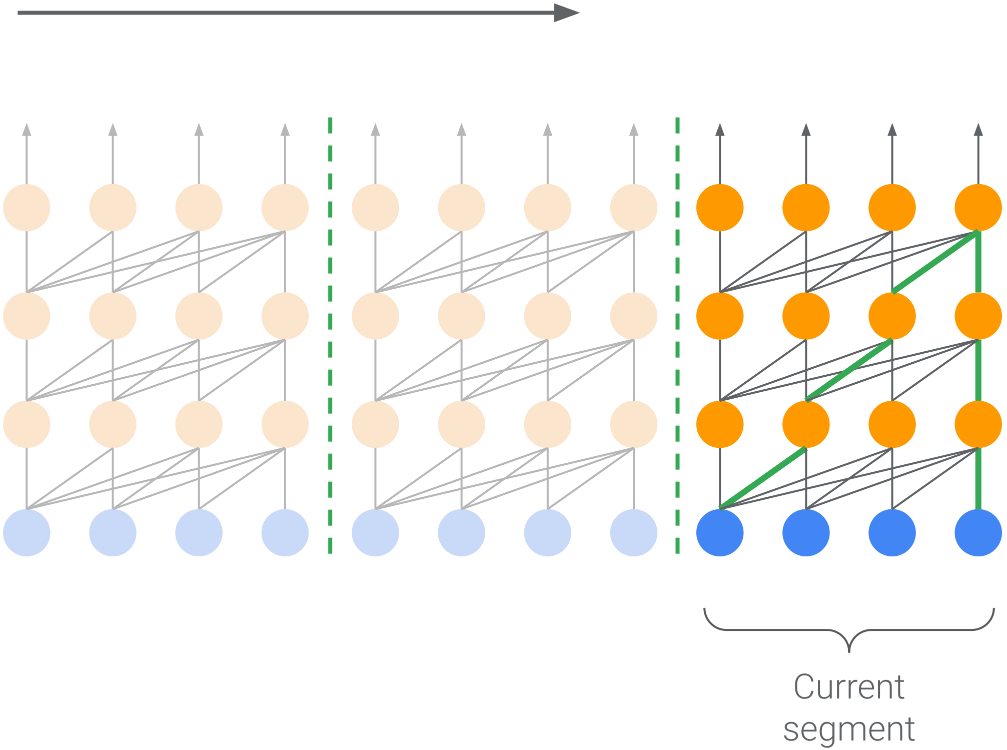


Recurrence

Transformer-XL (Dai et al., 2019)

How can models process long sequences under limited hardware constraints?

A naive approach is to split the sequence into multiple smaller ones and process them separately



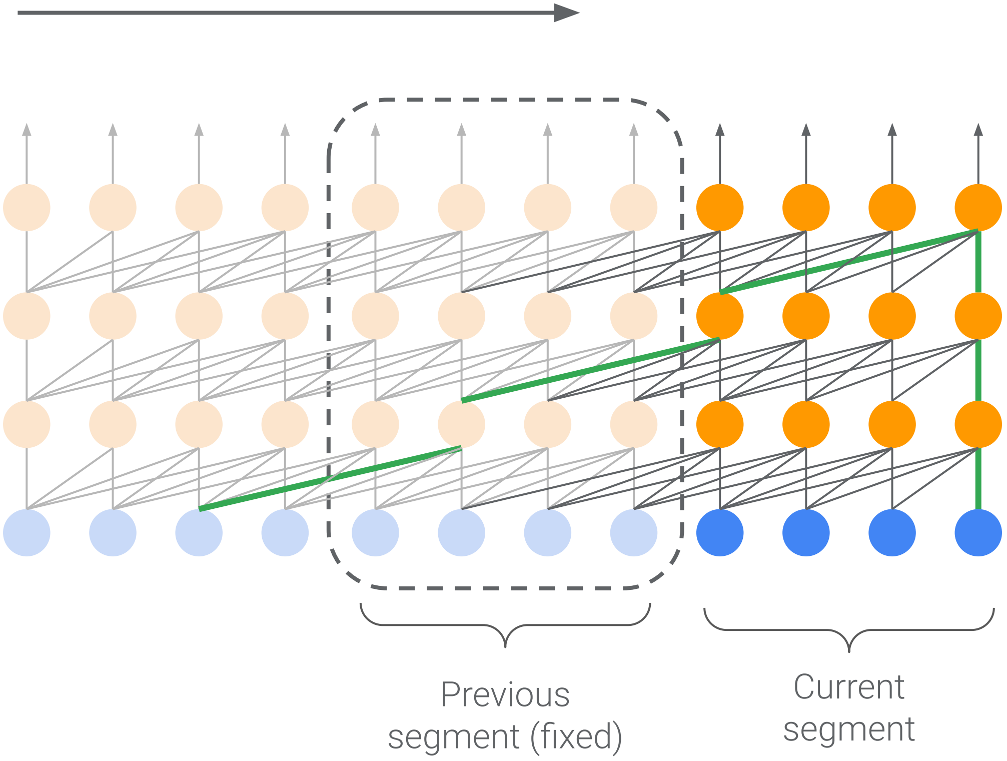
Recurrence

Transformer-XL ([Dai et al., 2019](#))

A better way is to add a segment-level recurrence mechanism

Representations from the previous segment are cached and re-used (no gradients flowing at training)

This increases **receptive field** proportionally to the depth of the transformer



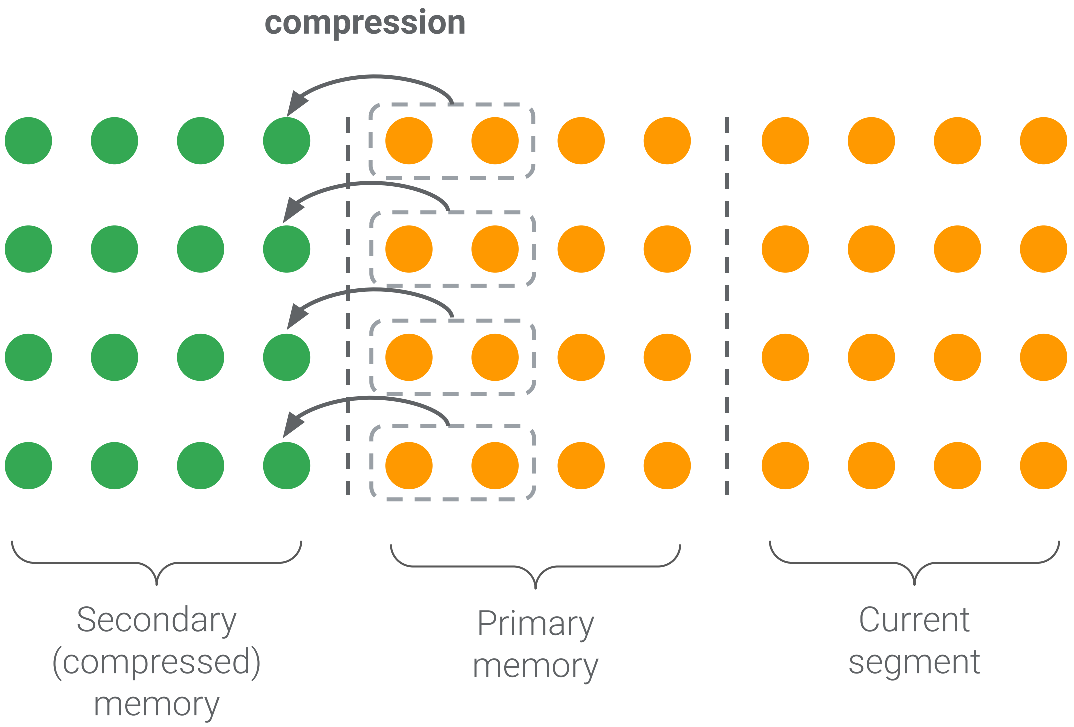
Recurrence

Compressive Transformers

(Rae et al., 2019)

Dual memory system:

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



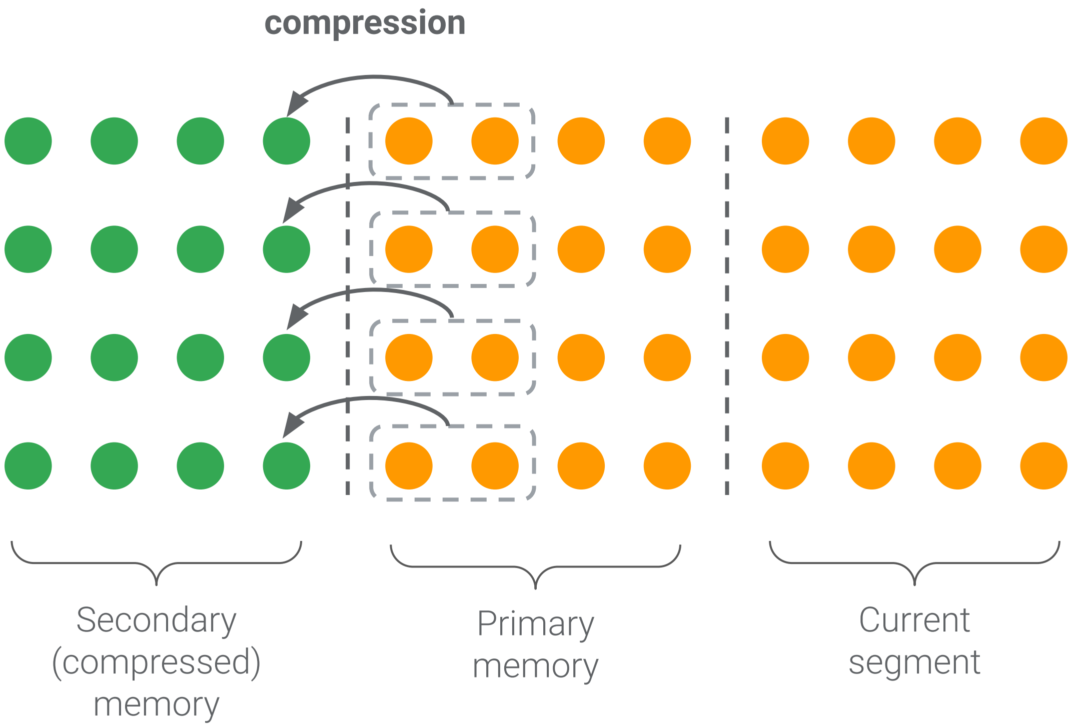
Recurrence

Compressive Transformers

(Rae et al., 2019)

When a new segment comes:

- Primary memory is updated with activations from previous segment
- Secondary memory is updated with the activations from the primary memory, where a **compression function** is applied (e.g. pooling, convolutions, most used)



Overview

Benchmarking

How do these models compare in practice?

The Long-Range Arena: a benchmark for efficient transformers

Benchmarking

How do these models compare in practice?

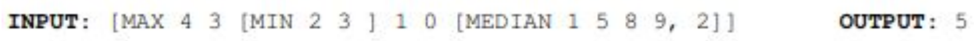
The Long-Range Arena: a benchmark for efficient transformers

Longer sequences: 1K-16K

5 tasks:

- List operations (e.g. max, min, median)
- Byte-level text classification
- Byte-level document retrieval
- Image classification
- Long-range spatial dependency

List operations example:



```
INPUT: [MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]        OUTPUT: 5
```

Benchmarking

How do these models compare in practice?

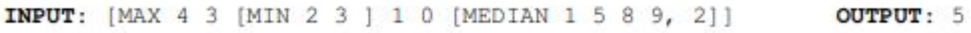
The Long-Range Arena: a benchmark for efficient transformers

Longer sequences: 1K-16K

5 tasks:

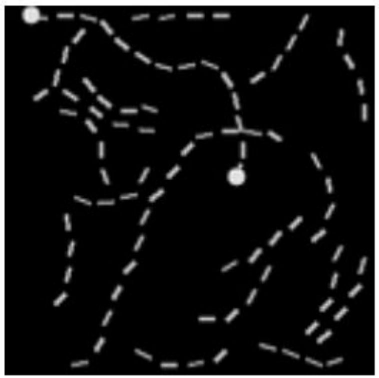
- List operations (e.g. max, min, median)
- Byte-level text classification
- Byte-level document retrieval
- Image classification
- Long-range spatial dependency

List operations example:

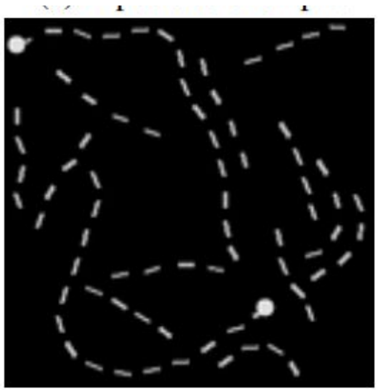


```
INPUT: [MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]        OUTPUT: 5
```

Long-range spatial dependency example:



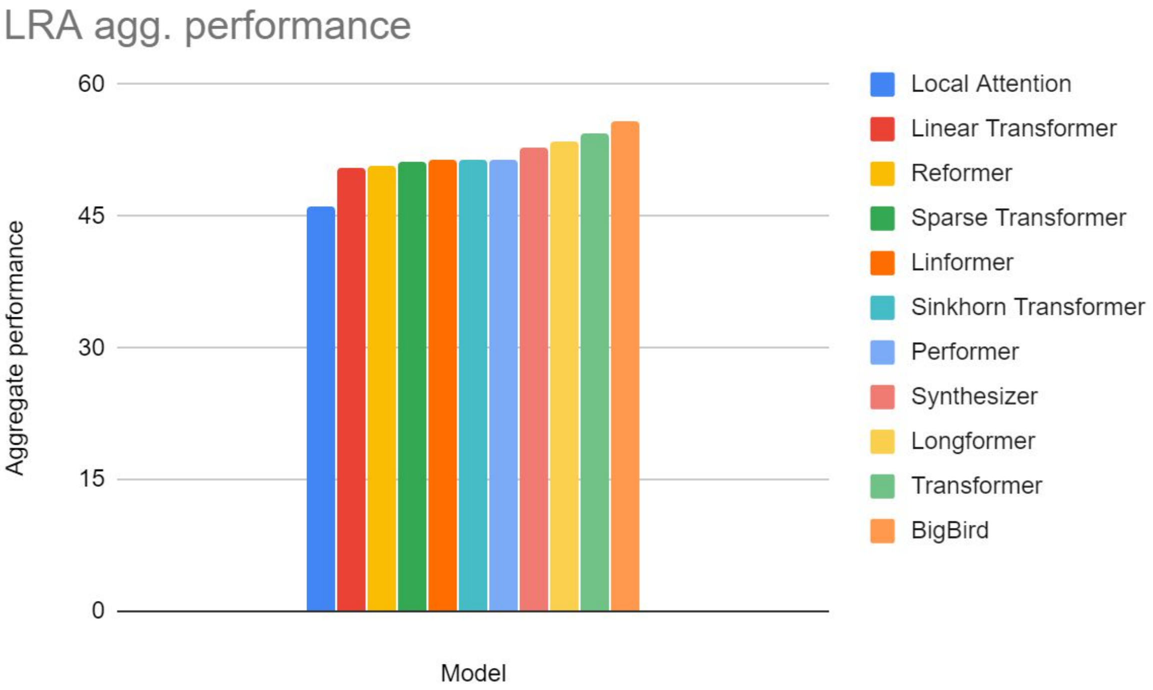
(positive)



(negative)

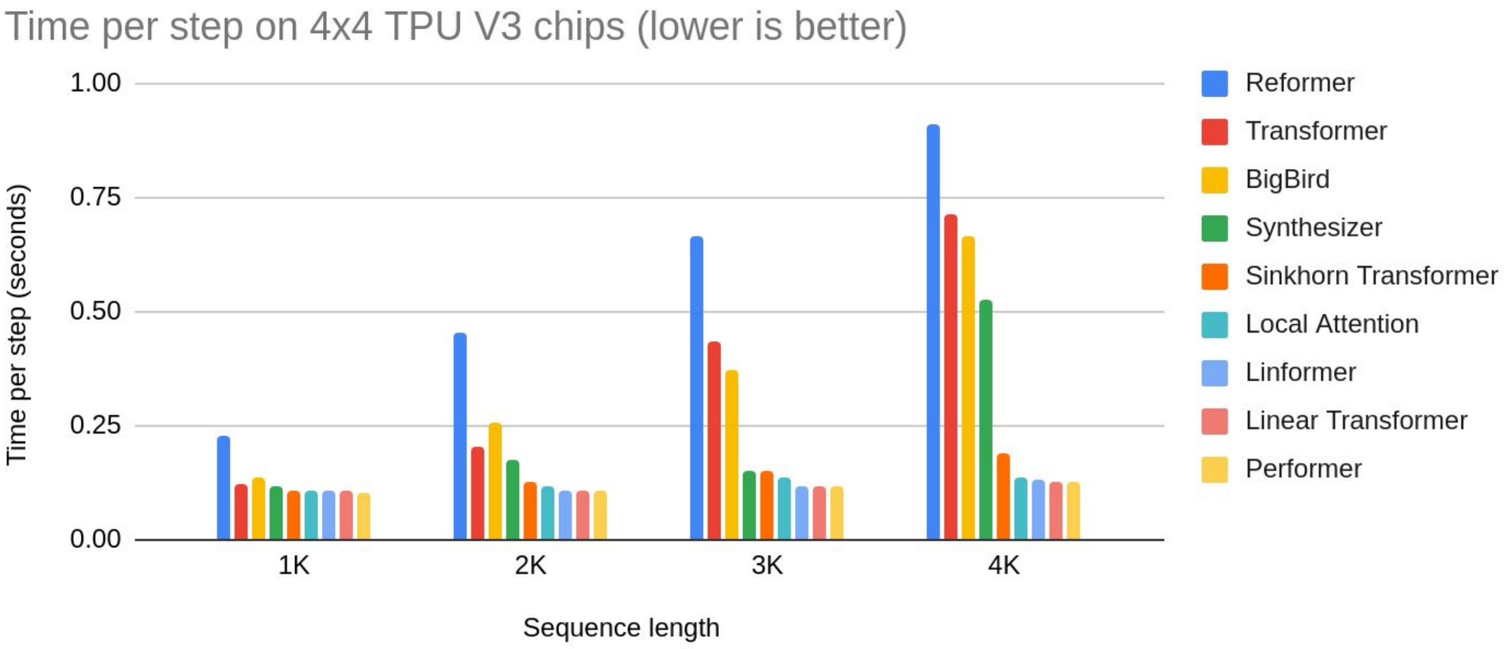
Benchmarking

The Long-Range Arena: a benchmark for efficient transformers



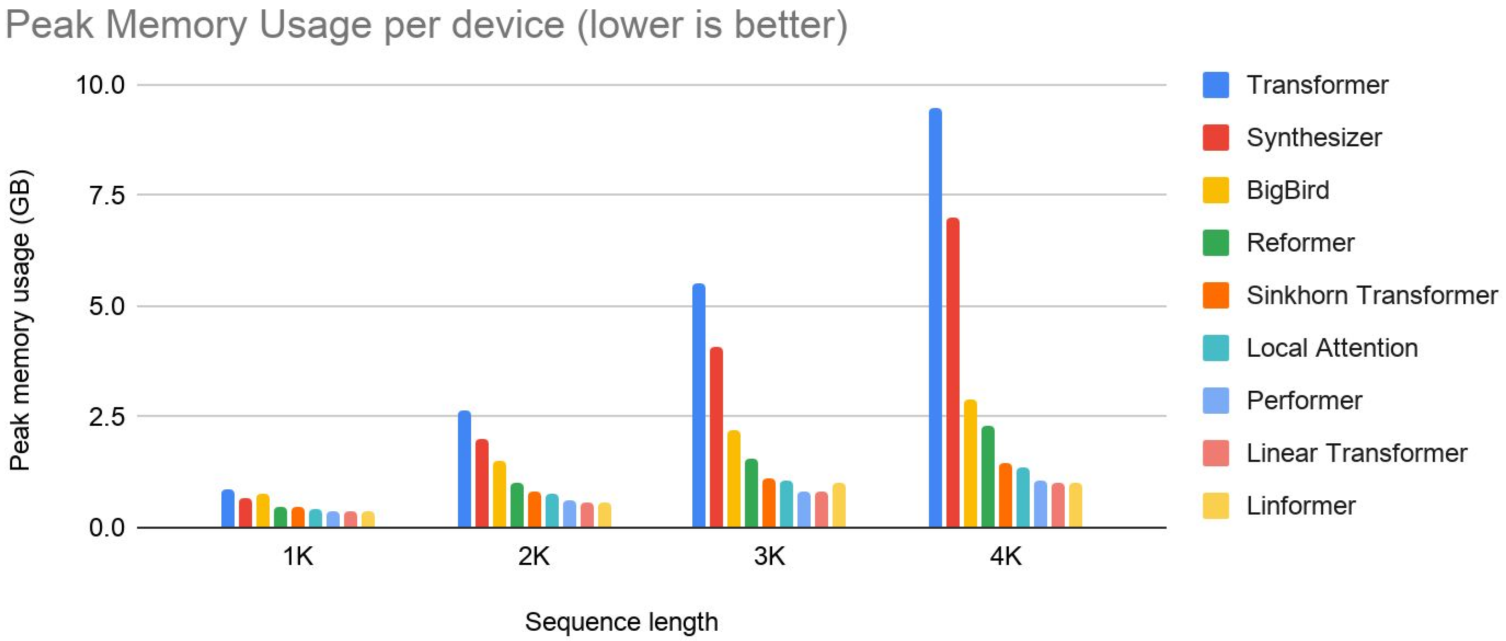
Benchmarking

The Long-Range Arena: a benchmark for efficient transformers



Benchmarking

The Long-Range Arena: a benchmark for efficient transformers

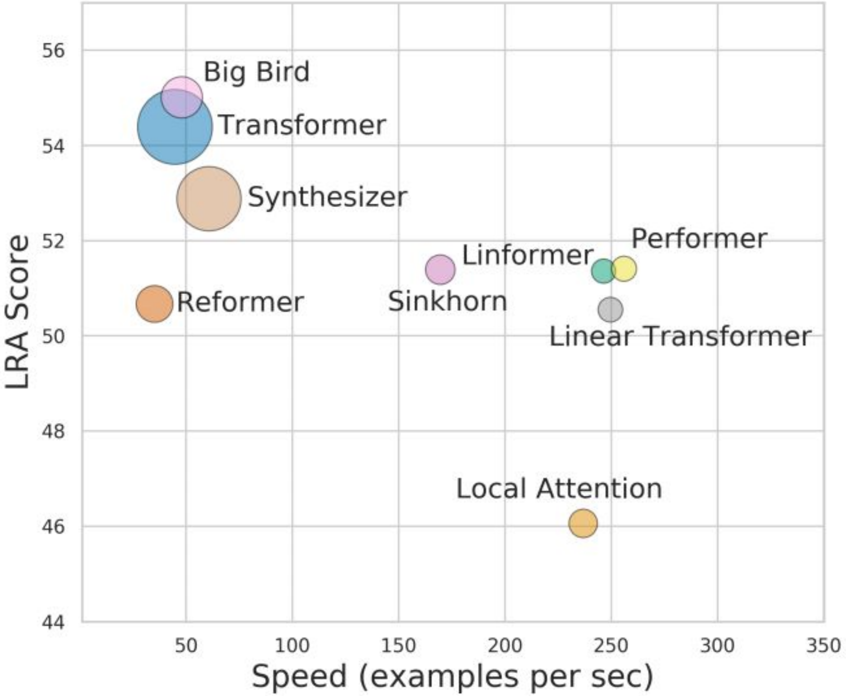


Benchmarking

The Long-Range Arena: a benchmark for efficient transformers

Putting it all together (size of circles corresponds to memory footprint)

Note: these results might be sensitive to implementation details, hardware and hyper-parameters.



Key Takeaways

There has been a surge in ideas for improving the efficiency of attention and transformers, especially for improving their capacity to handle long sequences.

Key Takeaways

There has been a surge in ideas for improving the efficiency of attention and transformers, especially for improving their capacity to handle long sequences.

There has been good progress in recent months: we are now able to compute attention in linear time with respect to sequence length, leading to large speed improvements without much performance drops for large sequences.

Key Takeaways

There has been a surge in ideas for improving the efficiency of attention and transformers, especially for improving their capacity to handle long sequences.

There has been good progress in recent months: we are now able to compute attention in linear time with respect to sequence length, leading to large speed improvements without much performance drops for large sequences.

Future improvements in hardware, e.g. on the efficiency of sparse computations, may make these ideas even more appealing in the long run (Hooker, 2020)

Key Takeaways

There has been a surge in ideas for improving the efficiency of attention and transformers, especially for improving their capacity to handle long sequences.

There has been good progress in recent months: we are now able to compute attention in linear time with respect to sequence length, leading to large speed improvements without much performance drops for large sequences.

Future improvements in hardware, e.g. on the efficiency of sparse computations, may make these ideas even more appealing in the long run (Hooker, 2020)

The ideas presented in this section are often orthogonal to each other and to other efforts presented in this tutorial, and can be combined for more efficient models.

05

Case Studies

Efficient Language models

Towards more efficient NLP

1) Core techniques

2) Efficient attention

3) **Case studies**

   a) Efficient Language Models

   - Natural Language Processing with Small Feed-Forward Networks

Source: unsplash.com

Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) **Case studies**

    a) Efficient Language Models

    - Natural Language Processing with Small Feed-Forward Networks
    - The Evolved Transformer

NAS

(Neural Architecture Search)

Towards more efficient NLP

1) Core techniques

2) Efficient attention

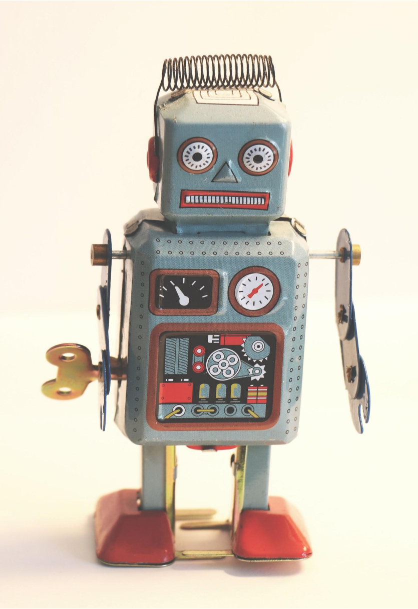
3) **Case studies**

   a) Efficient Language Models

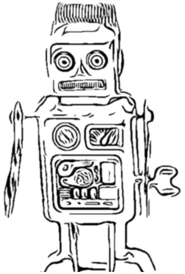
   - Natural Language Processing with Small Feed-Forward Networks

   - The Evolved Transformer

   - PRADO + pQRNN



Source: unsplash.com



Towards more efficient NLP

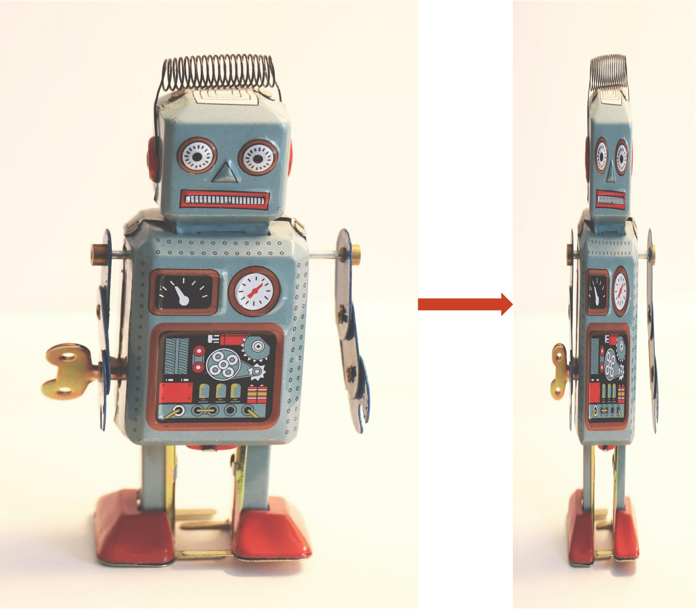
1) Core techniques

2) Efficient attention

3) **Case studies**

   a) Efficient Language Models

   - Natural Language Processing with Small Feed-Forward Networks
   - The Evolved Transformer
   - PRADO + pQRNN
   - MobileBERT



Source: unsplash.com

Towards more efficient NLP

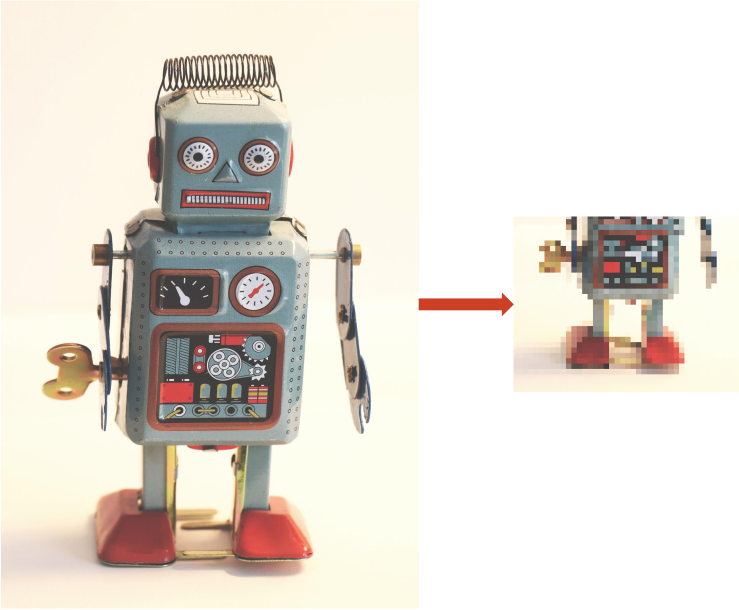
1) Core techniques

2) Efficient attention

3) **Case studies**

   a) Efficient Language Models

   - Natural Language Processing with Small Feed-Forward Networks
   - The Evolved Transformer
   - PRADO + pQRNN
   - MobileBERT
   - Lite Transformer with Long-Short Range Attention



Source: unsplash.com

Towards more efficient NLP

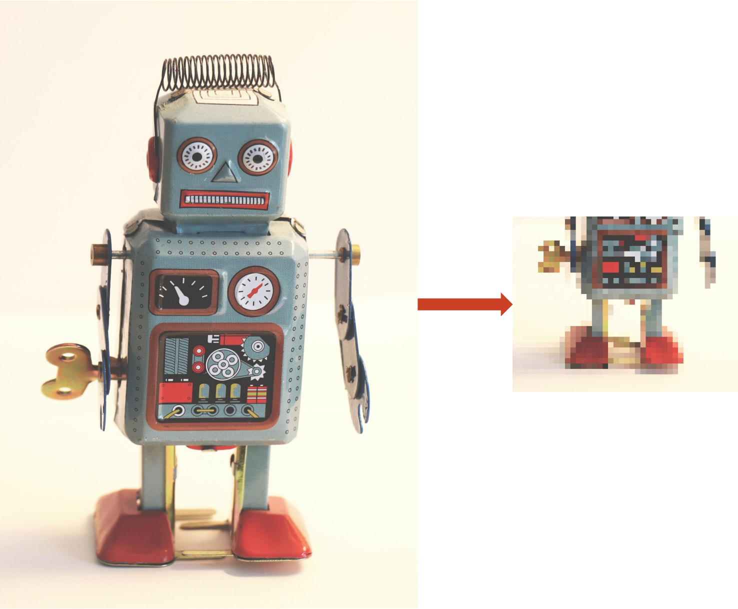
1) Core techniques

2) Efficient attention

3) **Case studies**

   a) Efficient Language Models

   - Natural Language Processing with Small Feed-Forward Networks
   - The Evolved Transformer
   - PRADO + pQRNN
   - MobileBERT
   - Lite Transformer with Long-Short Range Attention
   - MicroNet for Efficient Language Modeling



Source: unsplash.com

Towards more efficient NLP

1) Core techniques

2) Efficient attention

3) Case studies

a) Efficient Language Models

- Natural Language Processing with Small Feed-Forward Networks
- The Evolved Transformer
- PRADO + pQRNN
- MobileBERT
- Lite Transformer with Long-Short Range Attention
- MicroNet for Efficient Language Modeling
- Hardware-Aware Transformers

NAS

(Neural Architecture Search)

Towards more efficient NLP

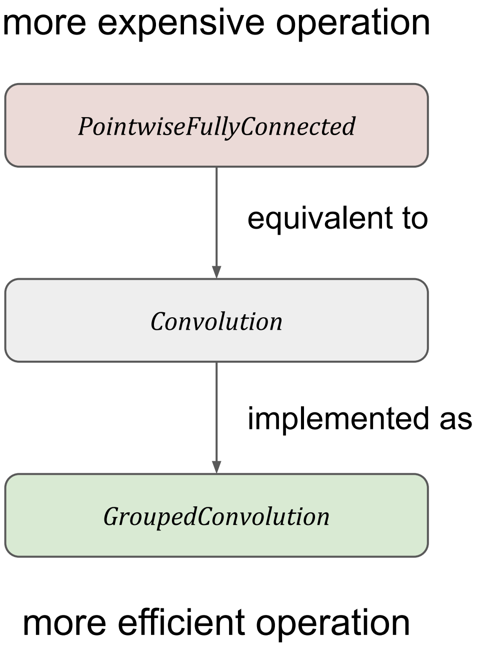
1) Core techniques

2) Efficient attention

3) Case studies

a) Efficient Language Models

- Natural Language Processing with Small Feed-Forward Networks
- The Evolved Transformer
- PRADO + pQRNN
- MobileBERT
- Lite Transformer with Long-Short Range Attention
- MicroNet for Efficient Language Modeling
- Hardware-Aware Transformers
- SqueezeBERT



Towards more efficient NLP

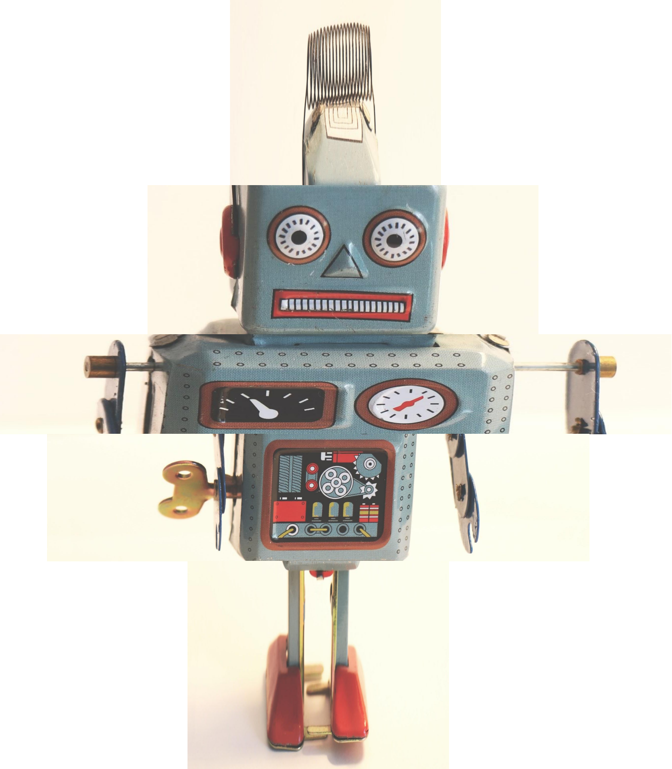
1) Core techniques

2) Efficient attention

3) **Case studies**

a) Efficient Language Models

- Natural Language Processing with Small Feed-Forward Networks
- The Evolved Transformer
- PRADO + pQRNN
- MobileBERT
- Lite Transformer with Long-Short Range Attention
- MicroNet for Efficient Language Modeling
- Hardware-Aware Transformers
- SqueezeBERT
- DeLighT: Very Deep and Light-weight Transformer



Natural Language Processing with Small Feed-Forward Networks

Botha et al., 2017

arxiv.org/abs/1708.00214

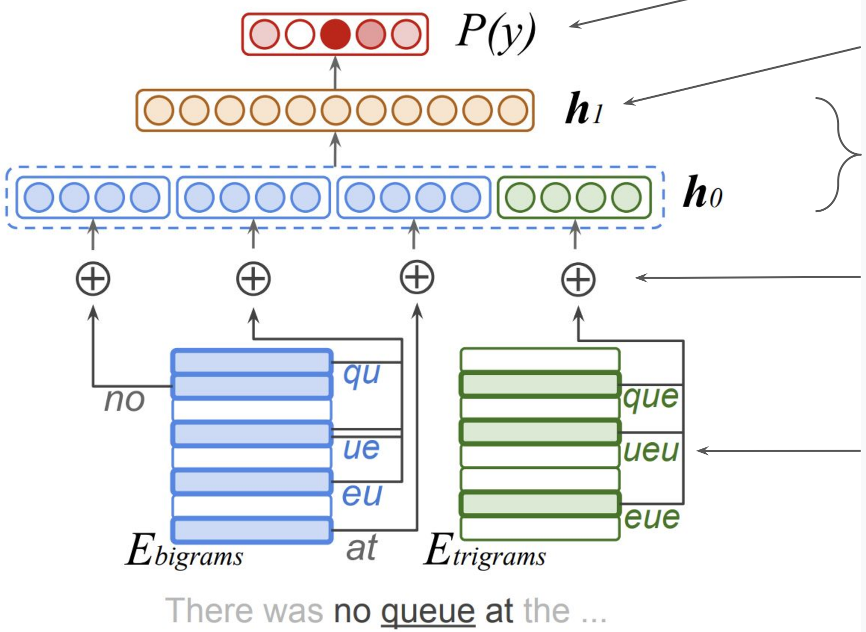
- Useful accuracies on a variety of tasks
- Great runtime and memory value in resource constrained environments

- Features defined over character n-grams, embeddings learned from scratch
- Random feature mixing hashing for small feature vocabularies
- Quantization for embedding weight compression

Natural Language Processing with Small Feed-Forward Networks

Botha et al., 2017

arxiv.org/abs/1708.00214



softmax

ReLU activated

fully connected

reshaping

Discrete feature embedding matrix

Figure 1: An example network structure for a model using bigrams of the previous, current and next word, and trigrams of the current word. Does not illustrate hashing.

Natural Language Processing with Small Feed-Forward Networks

Botha et al., 2017

arxiv.org/abs/1708.00214

Example result:

POS Tagging, compared to BTS (Gillick et al., 2016)

- +0.3% accuracy (**95.4%**, near state-of-the-art)
- 6x fewer parameters
- 36x fewer FLOPs

The Evolved Transformer

So et al., 2019

arxiv.org/abs/1901.11117

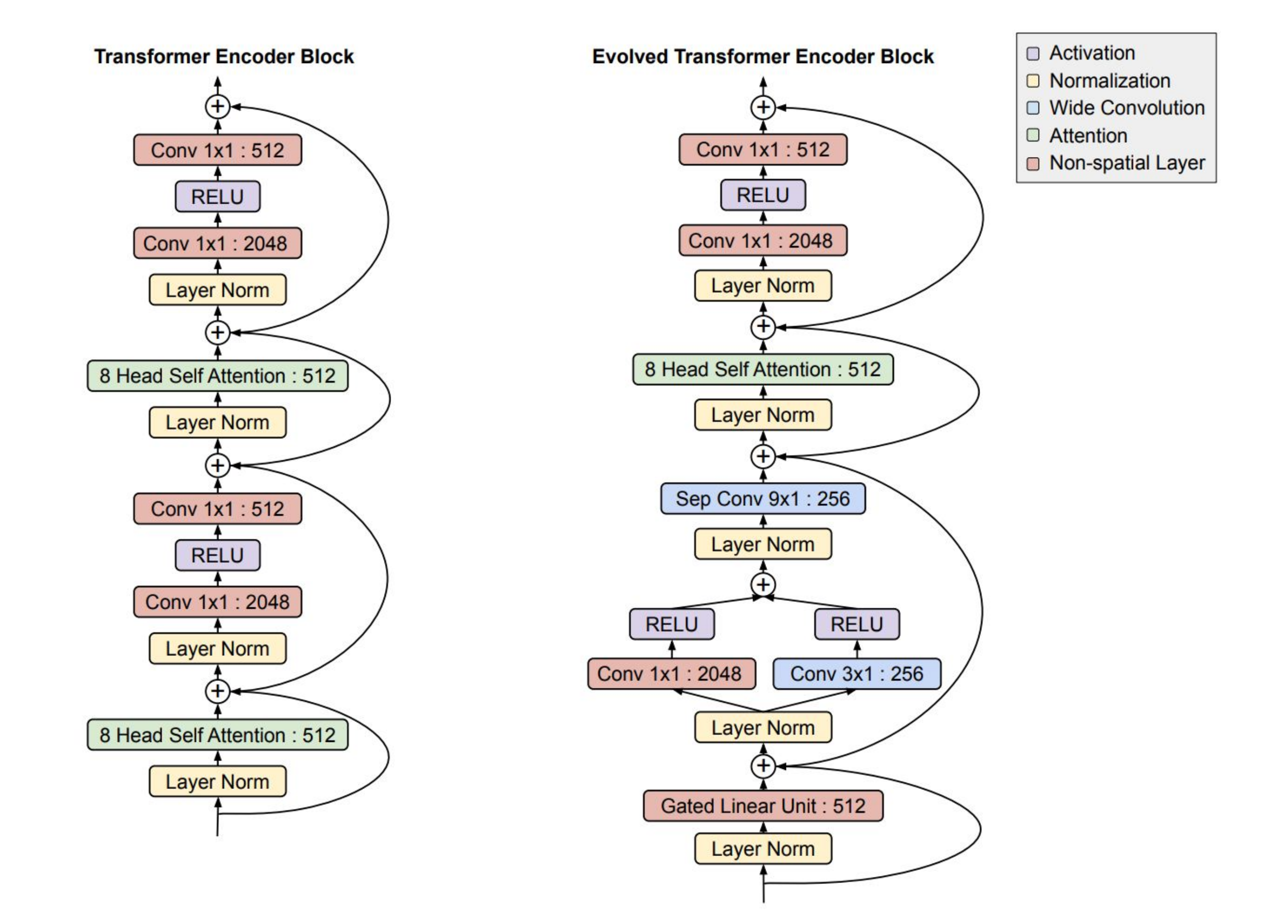
- Consistent improvement over Transformer on well established WMT and LM1B.

- NAS to search Transformer alternatives
- Large search space from feed-forward sequence models
- Evolutionary architecture search

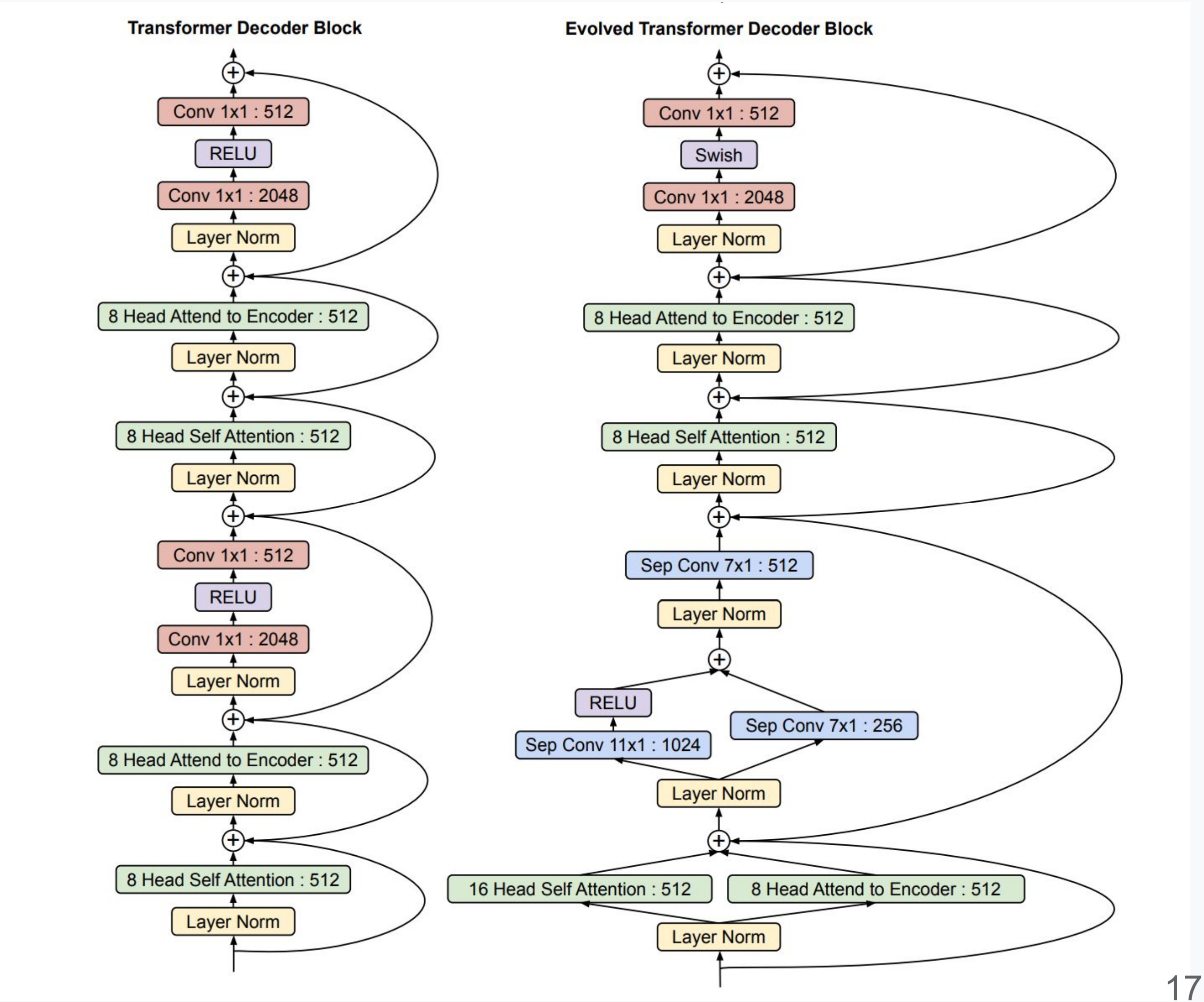
The Evolved Transformer

So et al., 2019

arxiv.org/abs/1901.11117



The Evolved Transformer



Transformer Decoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Evolved Transformer Decoder Block
Conv 1x1 : 512
Swish
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Sep Conv 7x1 : 512
Layer Norm
RELU
Sep Conv 11x1 : 1024
Sep Conv 7x1 : 256
Layer Norm
16 Head Self Attention : 512
8 Head Attend to Encoder : 512
Layer Norm

So et al., 2019

arxiv.org/abs/1901.11117

The Evolved Transformer

Same quality as original “big” Transformer with 37.6% fewer parameters and outperforms Transformer by 0.7 BLEU at a mobile-friendly model size of ~7M params.

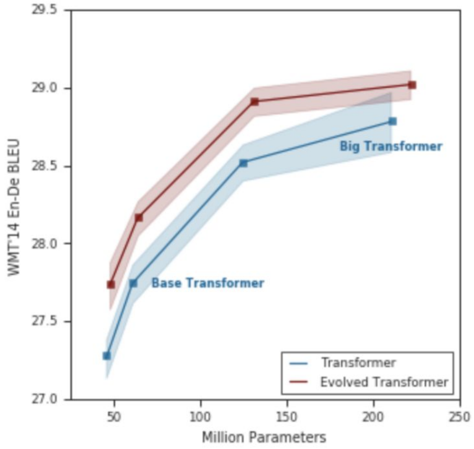


Figure 4. **Performance comparison of the Evolved Transformer against the Transformer across number of parameters.**

So et al., 2019

arxiv.org/abs/1901.11117

PRADO + pQRNN

PRADO: Projection Attention Networks for Document Classification On-Device

- Combines trainable projections with attention and convolutions
- With only 200 Kilobytes in size, outperformed prior CNN and LSTM models and achieved near state of the art performance on multiple long document classification tasks.

Kaliamoorthi et al., 2019-2020

www.aclweb.org/anthology/D19-1506/

https://ai.googleblog.com/2020/09/advancing-nlp-with-efficient-projection.html

PRADO + pQRNN

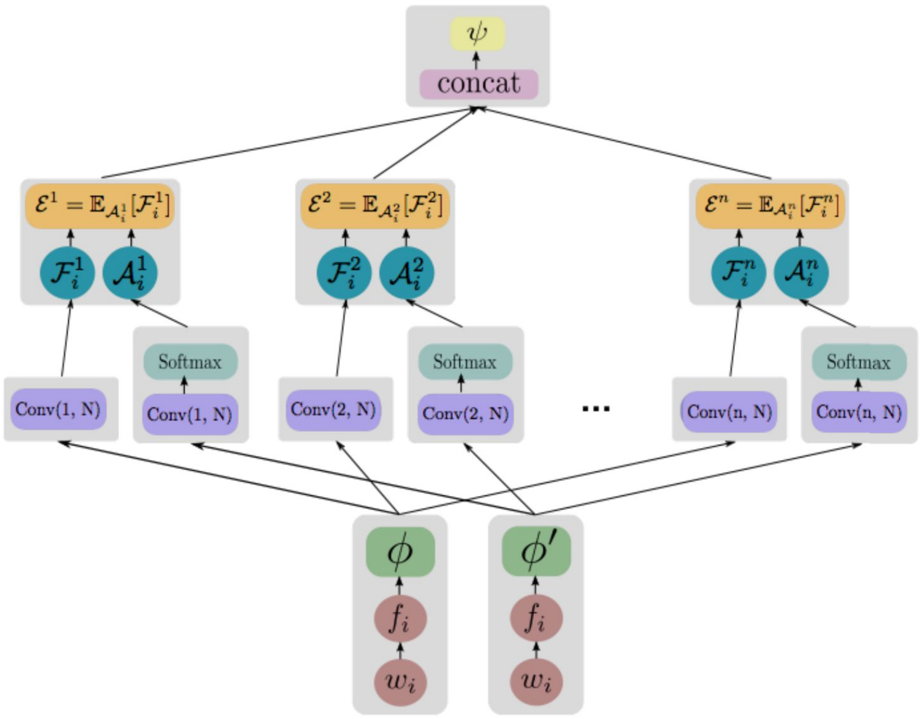


Figure 1: PRADO Model Architecture

Kaliamoorthi et al., 2019-2020

www.aclweb.org/anthology/D19-1506/

https://ai.googleblog.com/2020/09/advancing-nlp-with-efficient-projection.html

PRADO + pQRNN

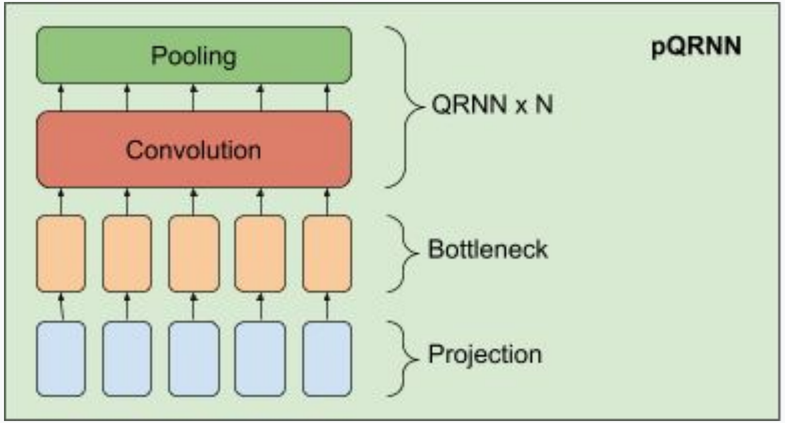
Kaliamoorthi et al., 2019-2020

www.aclweb.org/anthology/D19-1506/

https://ai.googleblog.com/2020/09/advancing-nlp-with-efficient-projection.html

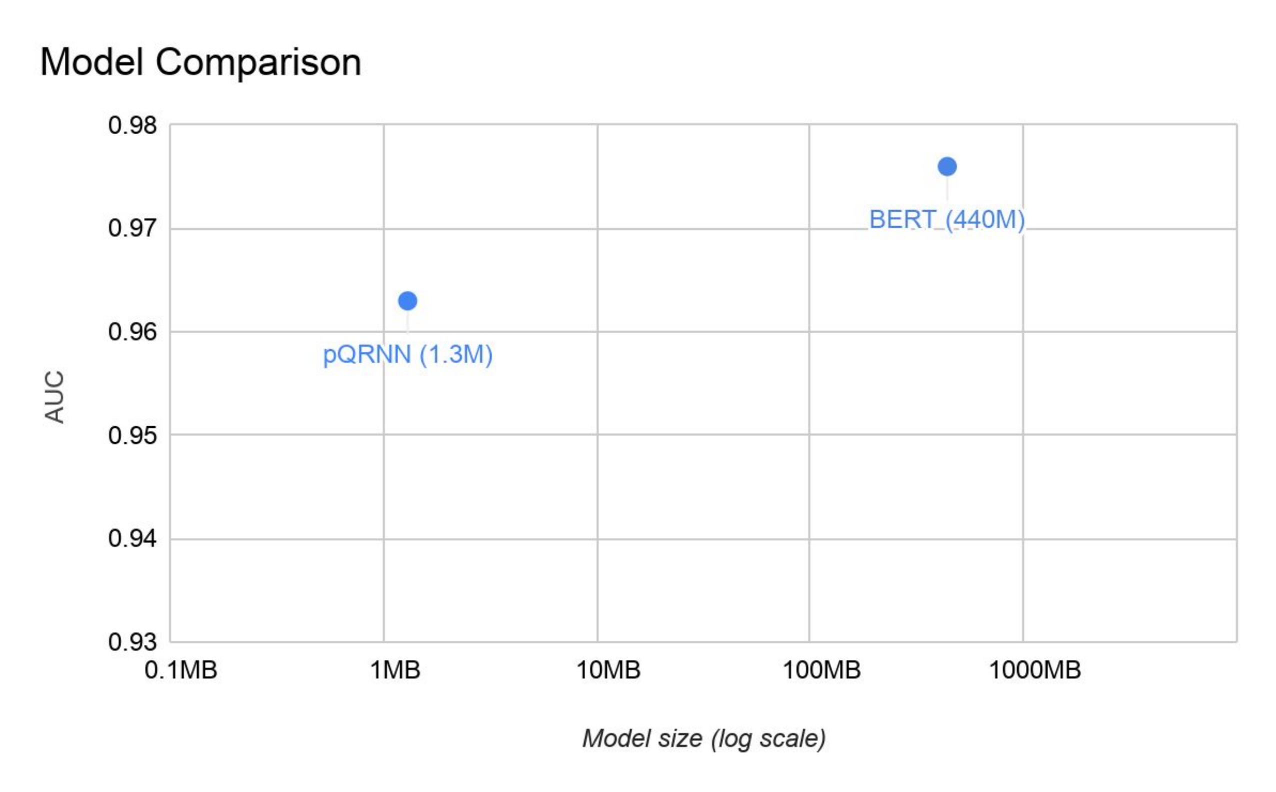
pQRNN

- A projection layer with a quasi-RNN encoder
- Same projection layer used in PRADO



- pQRNN is also quantized

PRADO + pQRNN



Kaliamoorthi et al., 2019-2020

www.aclweb.org/anthology/D19-1506/

https://ai.googleblog.com/2020/09/advancing-nlp-with-efficient-projection.html

MobileBERT

- Designed for running on mobile phones with acceptable latency
- Inverted-Bottleneck BERT$_{\text{LARGE}}$ teacher
- Distilled into a compact MobileBERT student
- As deep as BERT$_{\text{LARGE}}$, but narrower
- Task-agnostic compression (task specific fine-tuning performed directly on the compact model)

Sun et al., 2020

arxiv.org/abs/2004.02984

MobileBERT

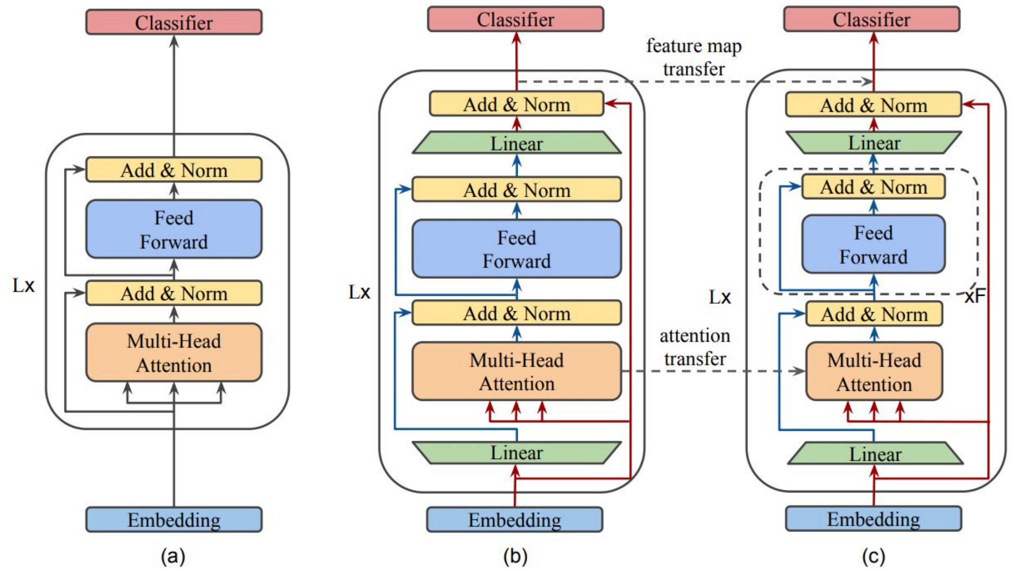
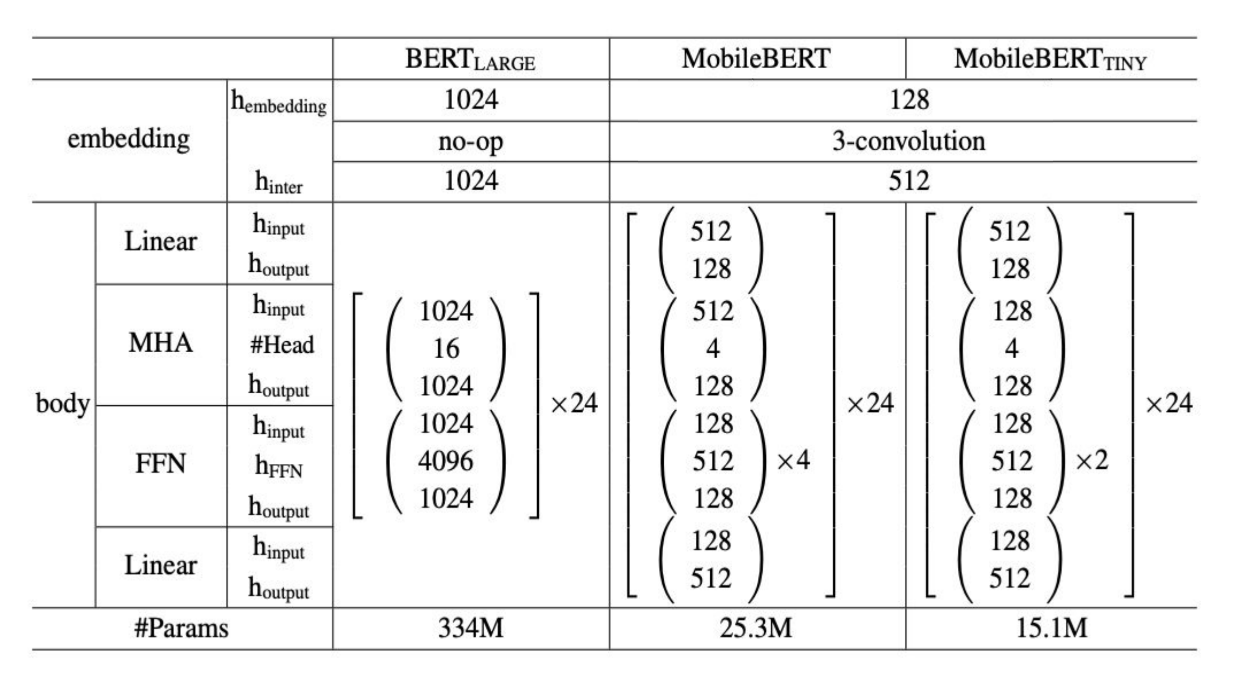


Figure 1: Illustration of three models: (a) BERT; (b) Inverted-Bottleneck BERT (IB-BERT); and (c) MobileBERT. In (b) and (c), red lines denote inter-block flows while blue lines intra-block flows. MobileBERT is trained by layer-to-layer imitating IB-BERT.

Sun et al., 2020

arxiv.org/abs/2004.02984

MobileBERT



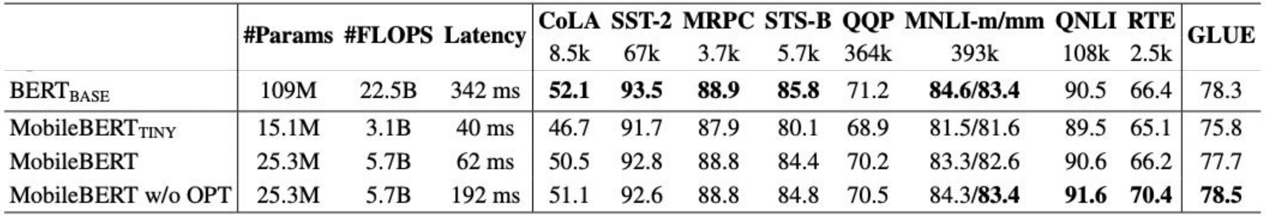
			BERT$_{\text{LARGE}}$		MobileBERT			MobileBERT$_{\text{TINY}}$		
embedding		$h_{\text{embedding}}$	1024		128					
			no-op		3-convolution					
		h_{inter}	1024		512					
body	Linear	h_{input}			512			512		
		h_{output}			128			128		
	MHA	h_{input}	1024		512			128		
		#Head	16		4			4		
		h_{output}	1024	×24	128		×24	128		×24
	FFN	h_{input}	1024		128			128		
		h_{FFN}	4096		512	×4		512	×2	
		h_{output}	1024		128			128		
	Linear	h_{input}			128			128		
		h_{output}			512			512		
#Params			334M		25.3M			15.1M		

Sun et al., 2020

arxiv.org/abs/2004.02984

MobileBERT

- 4.3x smaller, 5.5x faster than $\text{BERT}_{\text{BASE}}$
- 77.7 GLUE score ~ $\text{BERT}_{\text{BASE}}$
- 90.0/79.2 SQuAD v1.1/v2.0 F1 ~ $\text{BERT}_{\text{BASE}}$
- 62 ms latency on a Pixel 4 phone



	#Params	**#FLOPS**	**Latency**	**CoLA**	**SST-2**	**MRPC**	**STS-B**	**QQP**	**MNLI-m/mm**	**QNLI**	**RTE**	**GLUE**
				8.5k	67k	3.7k	5.7k	364k	393k	108k	2.5k	
$\text{BERT}_{\text{BASE}}$	109M	22.5B	342 ms	**52.1**	**93.5**	**88.9**	**85.8**	71.2	**84.6/83.4**	90.5	66.4	78.3
$\text{MobileBERT}_{\text{TINY}}$	15.1M	3.1B	40 ms	46.7	91.7	87.9	80.1	68.9	81.5/81.6	89.5	65.1	75.8
MobileBERT	25.3M	5.7B	62 ms	50.5	92.8	88.8	84.4	70.2	83.3/82.6	90.6	66.2	77.7
MobileBERT w/o OPT	25.3M	5.7B	192 ms	51.1	92.6	88.8	84.8	70.5	84.3/**83.4**	**91.6**	**70.4**	**78.5**

Sun et al., 2020

arxiv.org/abs/2004.02984

Lite Transformer with Long-Short Range Attention

Wu et al., 2020

arxiv.org/abs/2004.11886

- Long-Short Range Attention (LSRA)
  - Local context modeling by convolution
  - Long distance modeling by attention

- 2.5× reduced computation vs Transformer base
- 18.2× smaller with pruning and quantization

- 0.5 higher BLUE compared to Evolved Transformer, without the 250 GPU-year NAS cost.

MicroNet for Efficient Language Modeling

Yan et al., 2020

arxiv.org/abs/2005.07877

NeurIPS 2019 MicroNet Challenge[1]

Language modeling track: train efficient word-level language models on the Wikitext-103 Dataset[2] (word-level perplexity < 35)

Score = Normalized Parameter Storage +

Normalized Math Operations

(Normalized by LSTM Rae et at, 2018 arxiv.org/abs/1803.10049)

[1] Gale et al 2019, micronet-challenge.github.io

[2] Merity et al 2016, arxiv.org/abs/1609.07843

MicroNet for Efficient Language Modeling

Yan et al., 2020

arxiv.org/abs/2005.07877

- Core Language Model
    - Transformer-XL
    - Short Context Group Joint Optimization
    - Adaptive Embedding and Softmax
    - Hebbian Updates
- Compression Techniques
    - Knowledge Distillation
    - Pruning
    - Quantization

MicroNet for Efficient Language Modeling

Yan et al., 2020

arxiv.org/abs/2005.07877

- 90-fold reduction in parameter size and a 36-fold reduction in math operations compared to the MicroNet baseline

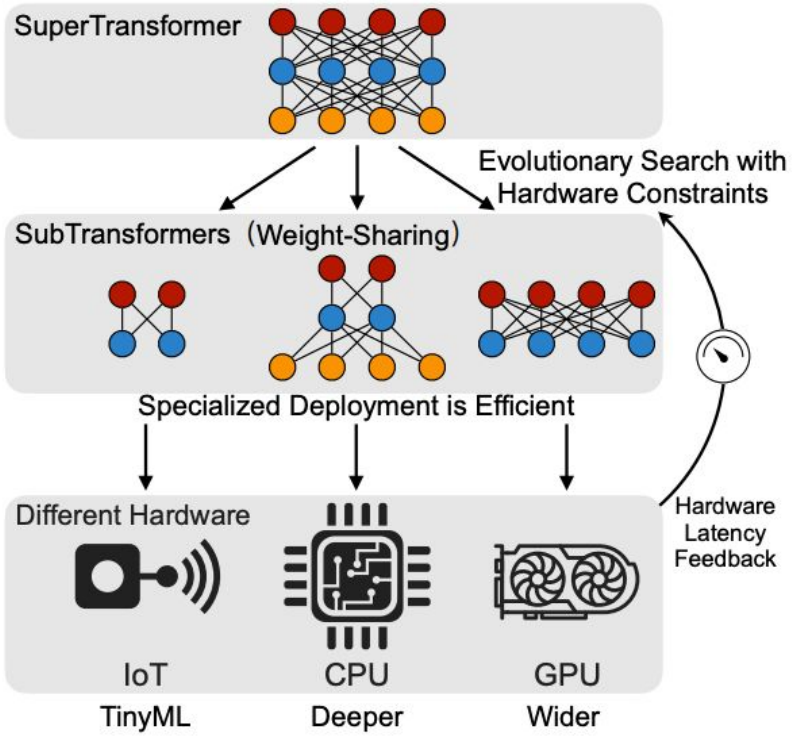
Hardware-Aware Transformers

- Neural Architecture Search
  - Train a *SuperTransformer* to cover a large space
  - Evolutionary search with hardware latency constraint to find a specialized *SubTransformer*
- Speed up and smaller size over baseline Transformer, and low search cost

Wang et al., 2020

arxiv.org/abs/2005.14187

Hardware-Aware Transformers



SuperTransformer
Evolutionary Search with Hardware Constraints
SubTransformers (Weight-Sharing)
Specialized Deployment is Efficient
Hardware Latency Feedback
Different Hardware
IoT
CPU
GPU
TinyML
Deeper
Wider

Figure 1: Framework for searching Hardware-Aware Transformers. We first train a SuperTransformer that contains numerous sub-networks, then conduct an evolutionary search with hardware latency feedback to find one **specialized** SubTransformer for each hardware.

Wang et al., 2020

arxiv.org/abs/2005.14187

Hardware-Aware Transformers

SubTransformer search

- Evolutionary search
- Find a satisfactory SubTransformer given a latency requirement
- Latency predictor trained for offline latency estimation (fast and accurate)

Wang et al., 2020

arxiv.org/abs/2005.14187

Hardware-Aware Transformers

WMT’14 results on Raspberry Pi-4:

- 3× speedup, 3.7× smaller size over baseline Transformer
- 2.7× speedup, 3.6× smaller size over Evolved Transformer with 12,041× less search cost

Wang et al., 2020

arxiv.org/abs/2005.14187

SqueezeBERT

- Replace several operations in self-attention layers with grouped convolutions

- Much faster inference on mobile devices

Iandola et al., 2020

arxiv.org/abs/2006.11316

SqueezeBERT

- Previous takeaways from CV into NLP (already adopted in MobileBERT)
  - Bottleneck layers
  - High-information flow residual connections
- New contributions from CV incorporated into SqueezeBERT's self-attention
  - Convolutions
  - Grouped convolutions

Iandola et al., 2020

arxiv.org/abs/2006.11316

SqueezeBERT

- Results
  - 4.3x faster than BERT-base (while MobileBERT is reported as 3.0x faster than BERT-base) on a Pixel 3 phone.
  - GLUE score 76.9 (vs 79.0 for BERT-base)

Iandola et al., 2020

arxiv.org/abs/2006.11316

DeLighT: Very Deep and Light-weight Transformer

Mehta et al., 2020

arxiv.org/abs/2008.00623

- More efficient parameter allocation within and across Transformer blocks

- Similar performance with substantially fewer parameters compared to baseline transformers.

DeLighT: Very Deep and Light-weight Transformer

Mehta et al., 2020

arxiv.org/abs/2008.00623

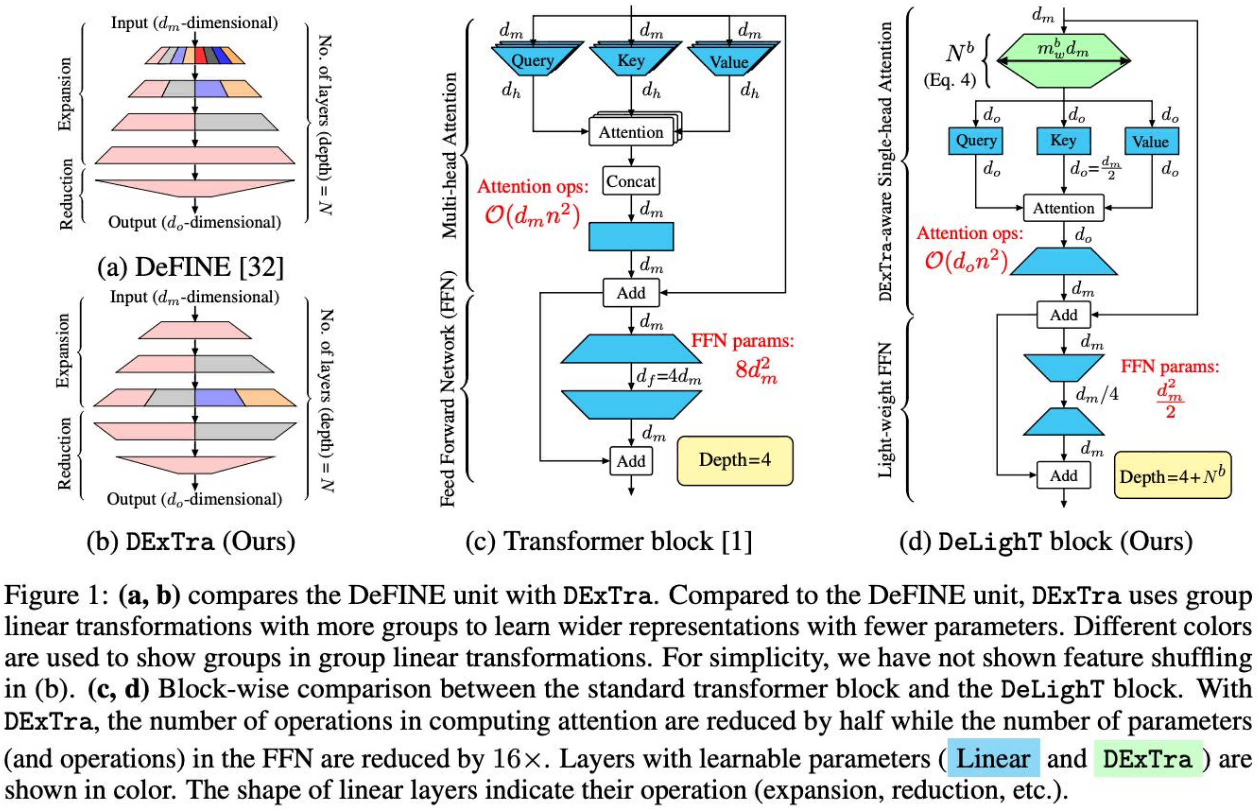
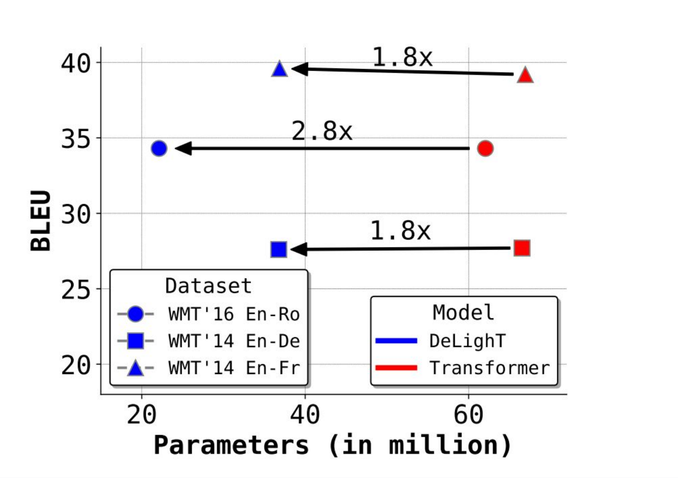
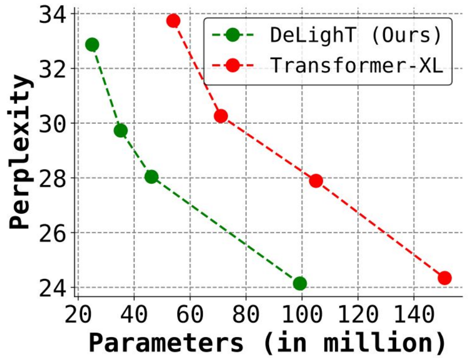

Input (d_m-dimensional)
Expansion
Reduction
No. of layers (depth) = N
Output (d_o-dimensional)
(a) DeFINE [32]
Input (d_m-dimensional)
Expansion
Reduction
No. of layers (depth) = N
Output (d_o-dimensional)
(b) DExTra (Ours)
Multi-head Attention
Feed Forward Network (FFN)
d_m
Query
Key
Value
d_h
Attention
Concat
Attention ops: $\mathcal{O}(d_m n^2)$
Add
FFN params: $8d_m^2$
d_f=$4d_m$
Depth=4
(c) Transformer block [1]
DExTra-aware Single-head Attention
Light-weight FFN
N^b (Eq. 4)
$m_w^b d_m$
d_o
d_o=$\frac{d_m}{2}$
Attention ops: $\mathcal{O}(d_o n^2)$
$d_m/4$
FFN params: $\frac{d_m^2}{2}$
Depth=4+N^b
(d) DeLighT block (Ours)

Figure 1: **(a, b)** compares the DeFINE unit with DExTra. Compared to the DeFINE unit, DExTra uses group linear transformations with more groups to learn wider representations with fewer parameters. Different colors are used to show groups in group linear transformations. For simplicity, we have not shown feature shuffling in (b). **(c, d)** Block-wise comparison between the standard transformer block and the DeLighT block. With DExTra, the number of operations in computing attention are reduced by half while the number of parameters (and operations) in the FFN are reduced by 16×. Layers with learnable parameters (Linear and DExTra) are shown in color. The shape of linear layers indicate their operation (expansion, reduction, etc.).

DeLighT: Very Deep and Light-weight Transformer



(b) DeLighT vs. Transformer-XL



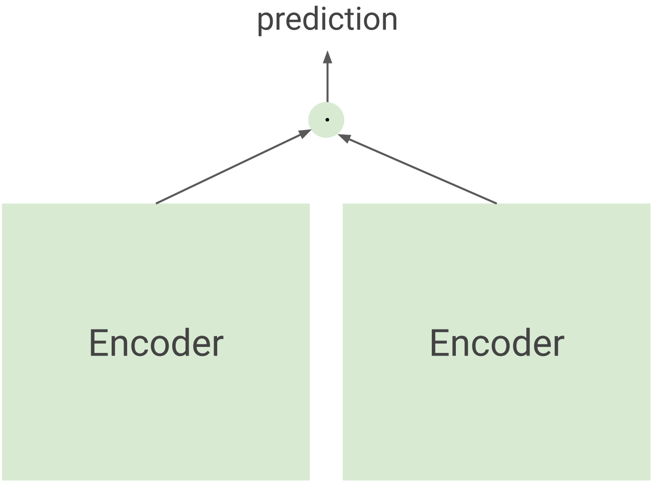
Mehta et al., 2020

arxiv.org/abs/2008.00623

Retrieval

Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) **Case studies**
    a) Efficient Language Models
    b) Retrieval
        - Sentence Embeddings using Siamese BERT-Networks



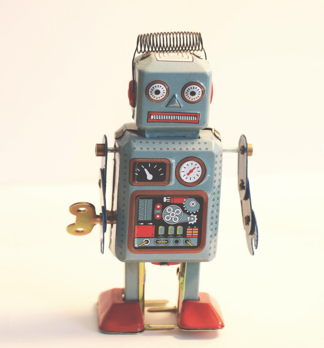
Towards more efficient NLP

1) Core techniques
2) Efficient attention
3) **Case studies**

   a) Efficient Language Models

   b) Retrieval

   - Sentence Embeddings using Siamese BERT-Networks
   - Generalization through Memorization: Nearest Neighbor Language Models



Source: unsplash.com

Towards more efficient NLP

1) Core techniques

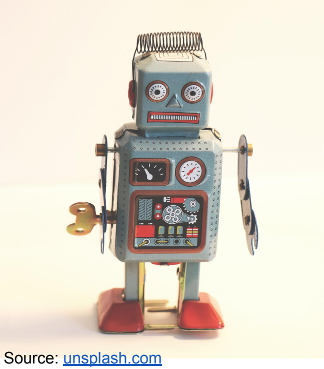
2) Efficient attention

3) **Case studies**

    a) Efficient Language Models

    b) Retrieval

    - Sentence Embeddings using Siamese BERT-Networks
    - Generalization through Memorization: Nearest Neighbor Language Models
    - REALM: Retrieval-Augmented Language Model Pre-Training



Source: unsplash.com

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

Reimers et al., 2019

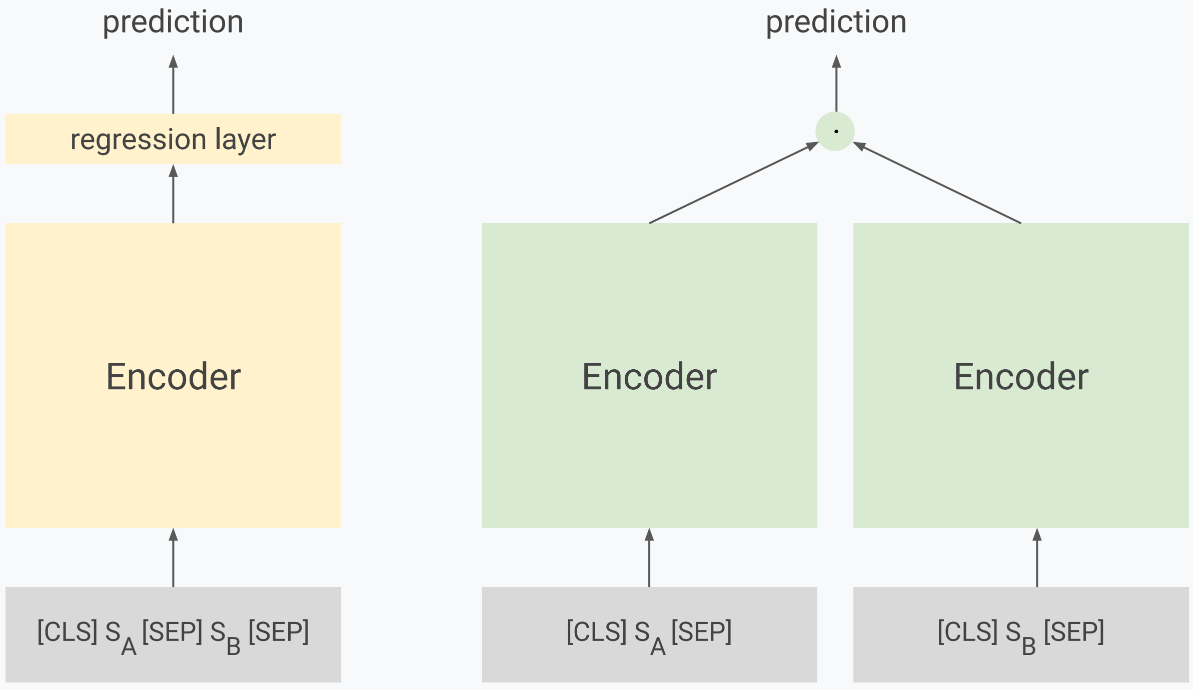
arxiv.org/abs/1908.10084

- Cross-attention, single tower models such as BERT have set state-of-the-art results on sentence-pair tasks such as STS.
- For sentence-retrieval tasks, cross-attention model requires expensive re-encoding the entire retrieval corpus.
- Sentence-BERT modifies the pretrained encoder to perform a single inference per input sentence, followed by cheap pairwise comparisons e.g. cosine similarity.

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

Reimers et al., 2019

arxiv.org/abs/1908.10084



Cross-attentional

(Single tower)

Dual-encoder

(Two tower)

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

Reimers et al., 2019

arxiv.org/abs/1908.10084

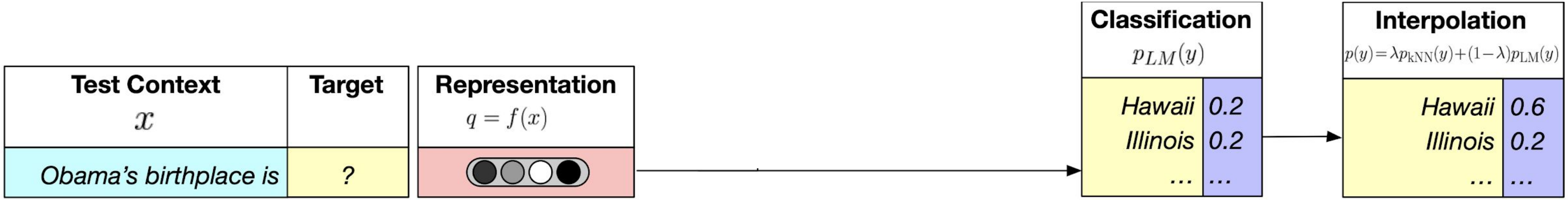
- Finding the most similar sentence in a collection of 10,000 sentences on a V100 GPU
  - BERT (cross-attention): 65 hours
  - SBERT (dual encoder): 5 seconds
- Can also be combined with Maximum Inner Product Search tools for sublinear scaling
  - https://github.com/google-research/google-research/tree/master/scann
  - https://github.com/facebookresearch/faiss
  - https://github.com/spotify/annoy

Generalization through Memorization: Nearest Neighbor Language Models

Khandelwal et al., 2019

arxiv.org/abs/1911.00172

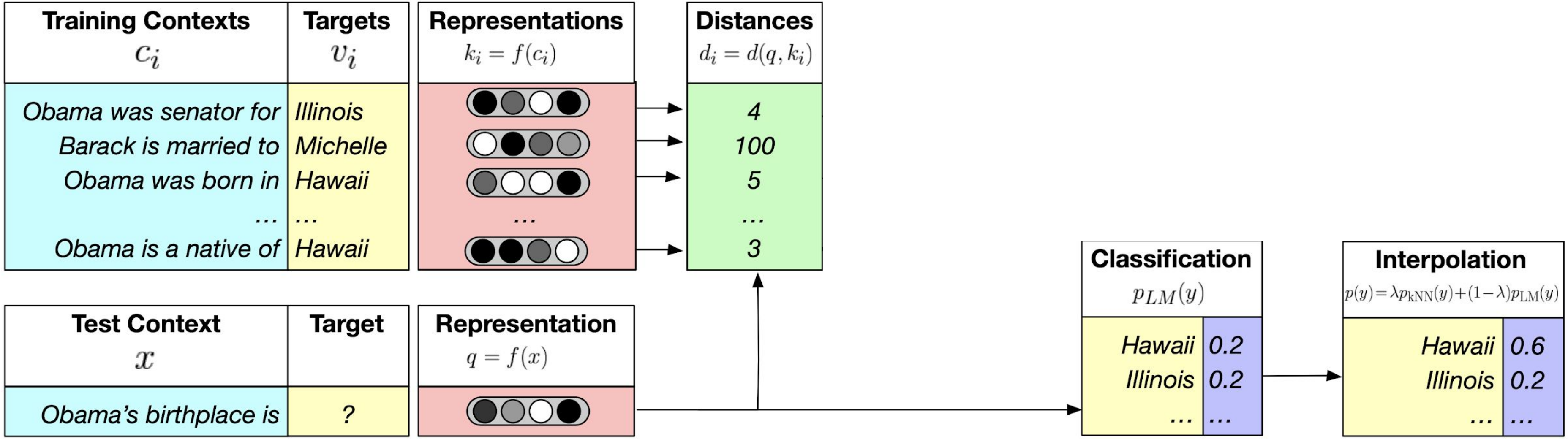
- Introduces kNN-LMs, which extends a pre-trained neural language model (LM) by linearly interpolating it with a k-nearest neighbors (kNN) model.
- Allows for efficiently scaling up to larger training sets and for effective domain adaptation



Test Context x	Target	Representation $q = f(x)$
Obama's birthplace is	*?*	

Classification $p_{LM}(y)$	
Hawaii	*0.2*
Illinois	*0.2*
…	…

Interpolation $p(y) = \lambda p_{\mathrm{kNN}}(y) + (1-\lambda) p_{\mathrm{LM}}(y)$	
Hawaii	*0.6*
Illinois	*0.2*
…	…

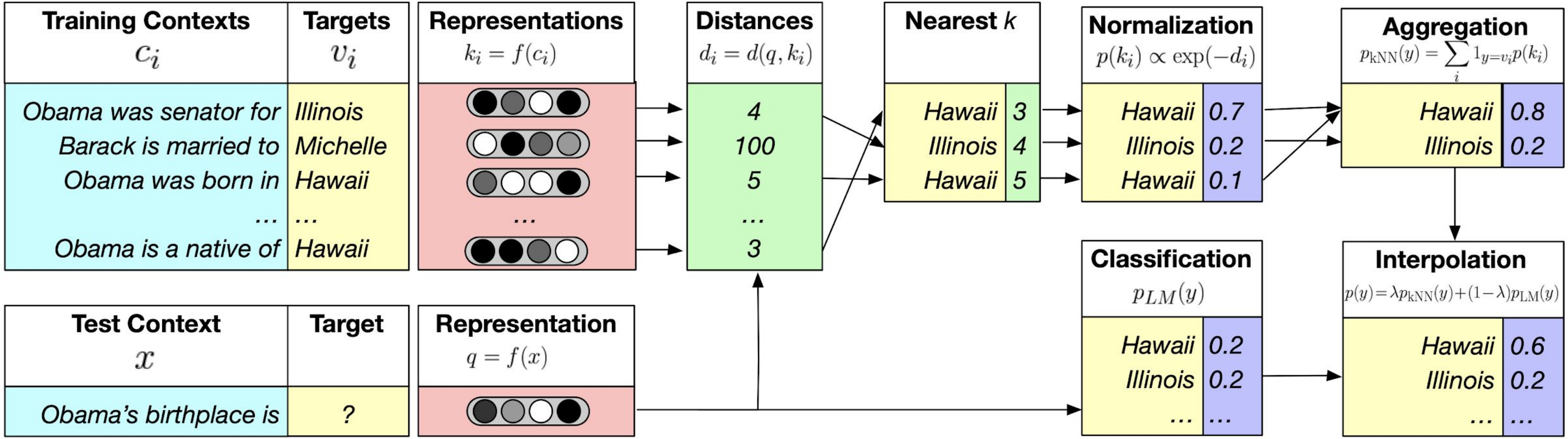


Training Contexts c_i	Targets v_i	Representations $k_i = f(c_i)$	Distances $d_i = d(q, k_i)$
Obama was senator for	*Illinois*	●◉○●	*4*
Barack is married to	*Michelle*	○●◉◉	*100*
Obama was born in	*Hawaii*	◉○○●	*5*
…	…	…	…
Obama is a native of	*Hawaii*	●●◉○	*3*

Test Context x	Target	Representation $q = f(x)$
Obama's birthplace is	*?*	●◉○●

Classification $p_{LM}(y)$	
Hawaii	*0.2*
Illinois	*0.2*
…	…

Interpolation $p(y) = \lambda p_{\mathrm{kNN}}(y) + (1 - \lambda) p_{\mathrm{LM}}(y)$	
Hawaii	*0.6*
Illinois	*0.2*
…	…



Training Contexts c_i	Targets v_i
Obama was senator for	Illinois
Barack is married to	Michelle
Obama was born in	Hawaii
…	…
Obama is a native of	Hawaii

Representations $k_i = f(c_i)$

…

Distances $d_i = d(q, k_i)$
4
100
5
…
3

Nearest k	
Hawaii	3
Illinois	4
Hawaii	5

Normalization $p(k_i) \propto \exp(-d_i)$	
Hawaii	0.7
Illinois	0.2
Hawaii	0.1

Aggregation $p_{\text{kNN}}(y) = \sum_i 1_{y=v_i} p(k_i)$	
Hawaii	0.8
Illinois	0.2

Test Context x	Target
Obama's birthplace is	?

Representation $q = f(x)$

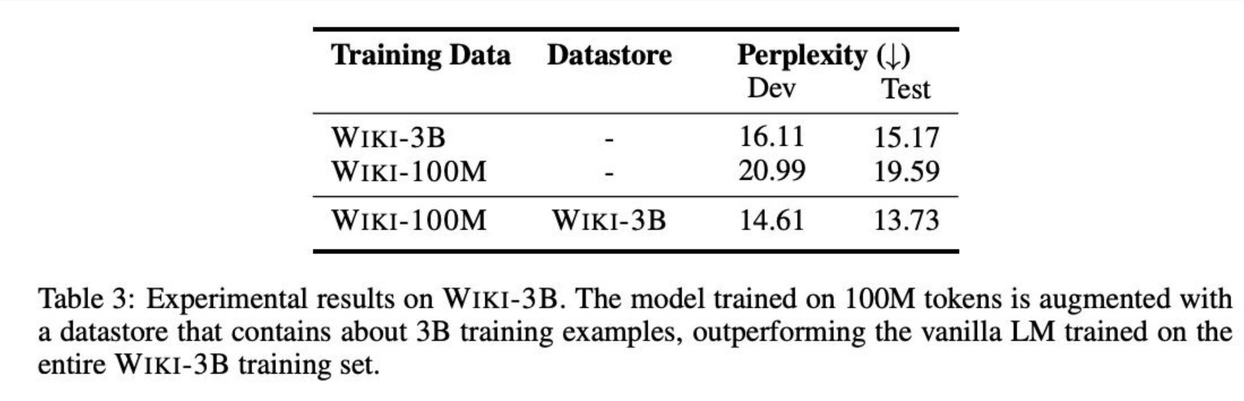
Classification $p_{LM}(y)$	
Hawaii	0.2
Illinois	0.2
…	…

Interpolation $p(y)=\lambda p_{\text{kNN}}(y)+(1-\lambda)p_{\text{LM}}(y)$	
Hawaii	0.6
Illinois	0.2
…	…

Generalization through Memorization: Nearest Neighbor Language Models

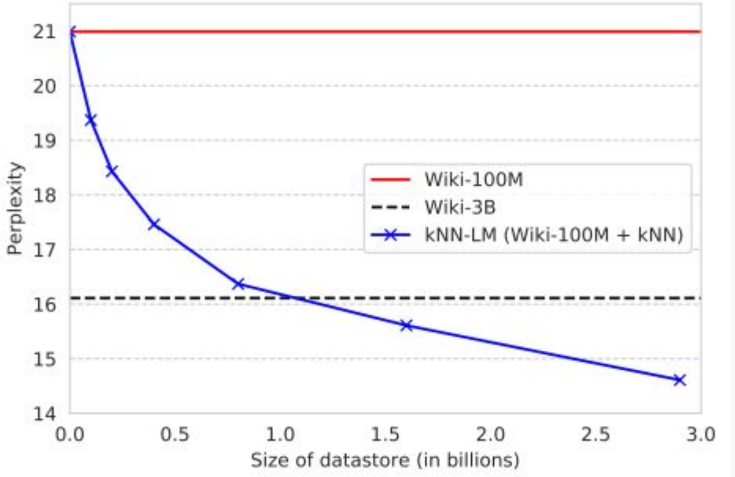
Khandelwal et al., 2019

arxiv.org/abs/1911.00172



Training Data	Datastore	Perplexity (↓)	
		Dev	Test
Wiki-3B	-	16.11	15.17
Wiki-100M	-	20.99	19.59
Wiki-100M	Wiki-3B	14.61	13.73

Table 3: Experimental results on Wiki-3B. The model trained on 100M tokens is augmented with a datastore that contains about 3B training examples, outperforming the vanilla LM trained on the entire Wiki-3B training set.



(a) Effect of datastore size on perplexities.

REALM: Retrieval-Augmented Language Model Pre-Training

Guu et al., 2020

arxiv.org/abs/2002.08909

- Language model pre-training can capture world knowledge by storing it implicitly in the network parameters, but storage space is limited by the network size (prompting for ever-larger networks).

- REALM introduces a latent knowledge retriever to augment the language model, and shows for the first time how to pretrain it in an unsupervised manner.

REALM: Retrieval-Augmented Language Model Pre-Training

Guu et al., 2020

arxiv.org/abs/2002.08909

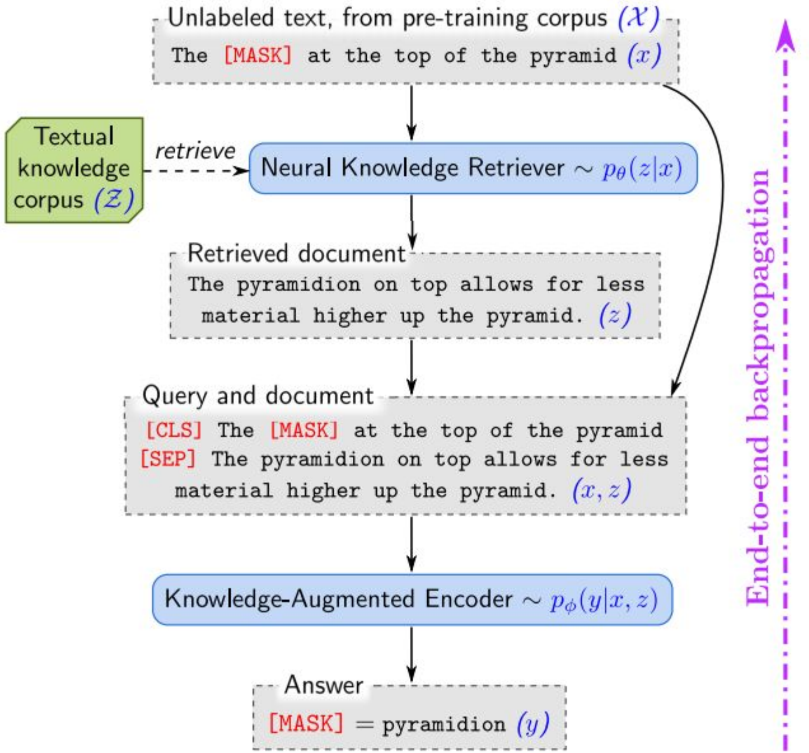

Unlabeled text, from pre-training corpus $(\mathcal{X})$
The [MASK] at the top of the pyramid (x)
Textual knowledge corpus $(\mathcal{Z})$
retrieve
Neural Knowledge Retriever $\sim p_\theta(z|x)$
Retrieved document
The pyramidion on top allows for less material higher up the pyramid. (z)
Query and document
[CLS] The [MASK] at the top of the pyramid [SEP] The pyramidion on top allows for less material higher up the pyramid. (x, z)
Knowledge-Augmented Encoder $\sim p_\phi(y|x, z)$
Answer
[MASK] $=$ pyramidion (y)
End-to-end backpropagation

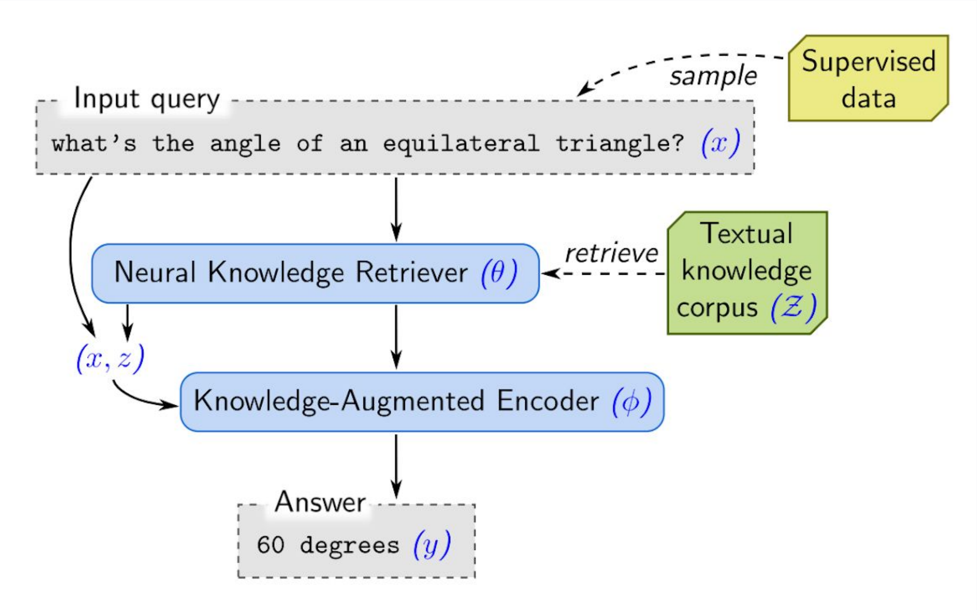
Figure 1. REALM augments language model pre-training with a **neural knowledge retriever** that retrieves knowledge from a **textual knowledge corpus**, $\mathcal{Z}$ (e.g., all of Wikipedia). Signal from the language modeling objective backpropagates all the way through the retriever, which must consider millions of documents in $\mathcal{Z}$—a significant computational challenge that we address.

REALM: Retrieval-Augmented Language Model Pre-Training

Guu et al., 2020

arxiv.org/abs/2002.08909

- Fine-tuning for open-domain question answering


Supervised data
sample
Input query
what's the angle of an equilateral triangle? (x)
Textual knowledge corpus $(\mathcal{Z})$
retrieve
Neural Knowledge Retriever (θ)
(x, z)
Knowledge-Augmented Encoder (ϕ)
Answer
60 degrees (y)

REALM: Retrieval-Augmented Language Model Pre-Training

- State-of-the-art Open-QA, with a relatively small model size (e.g. REALM outperforms T5-11b while being 30 times smaller)

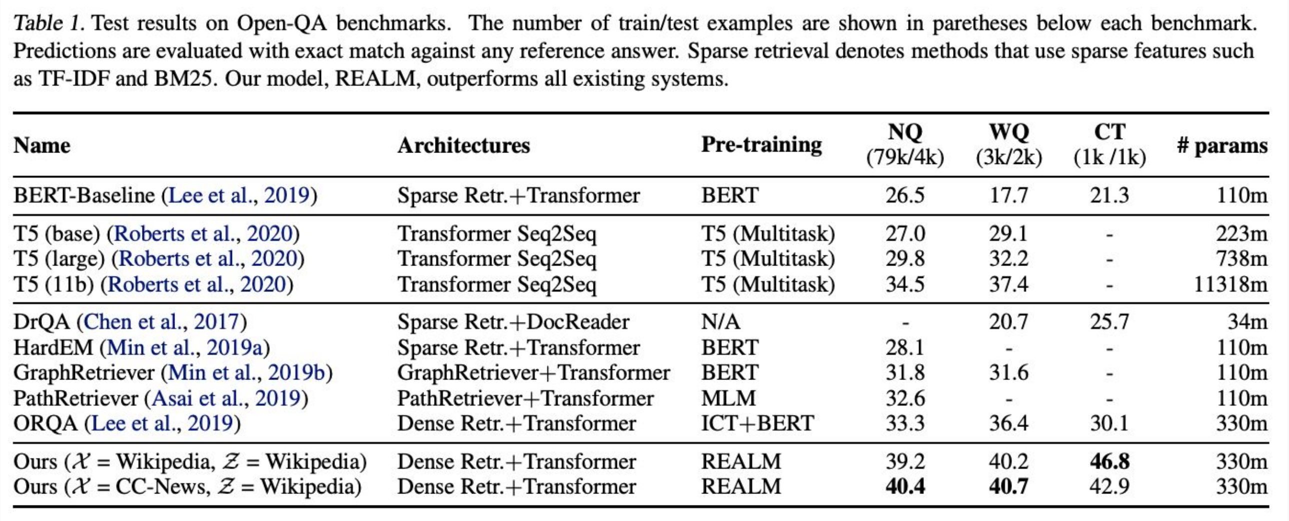


Table 1. Test results on Open-QA benchmarks. The number of train/test examples are shown in paretheses below each benchmark. Predictions are evaluated with exact match against any reference answer. Sparse retrieval denotes methods that use sparse features such as TF-IDF and BM25. Our model, REALM, outperforms all existing systems.

Name	**Architectures**	**Pre-training**	**NQ** (79k/4k)	**WQ** (3k/2k)	**CT** (1k /1k)	**# params**
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

Guu et al., 2020

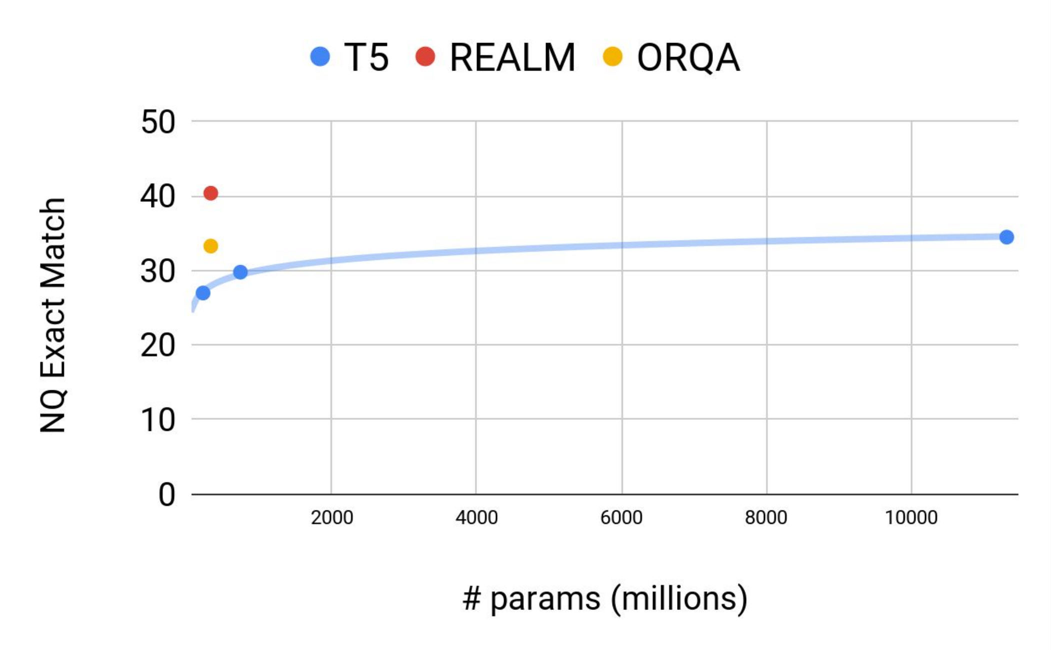
arxiv.org/abs/2002.08909

REALM: Retrieval-Augmented Language Model Pre-Training

Guu et al., 2020

arxiv.org/abs/2002.08909

- State-of-the-art Open-QA, with a relatively small model size (e.g. REALM outperforms T5-11b while being 30 times smaller)

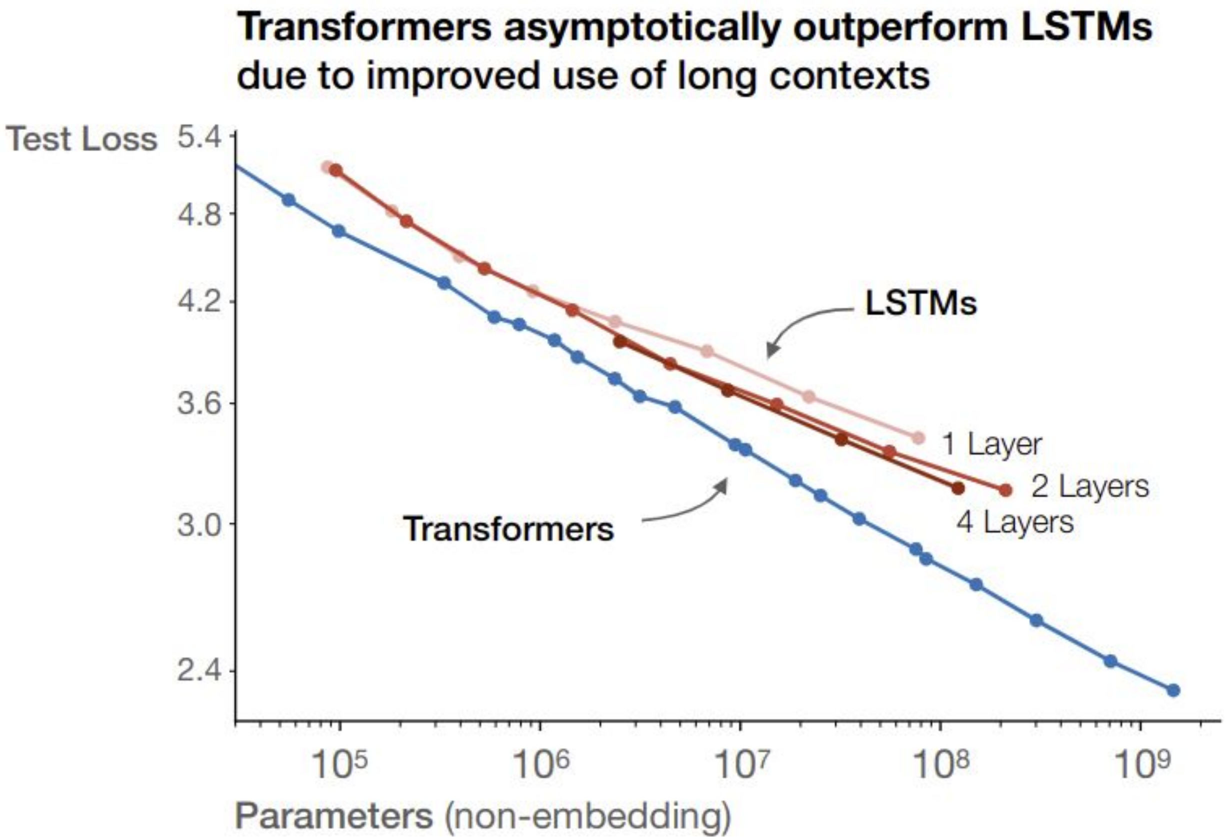


06

Scaling in Practice

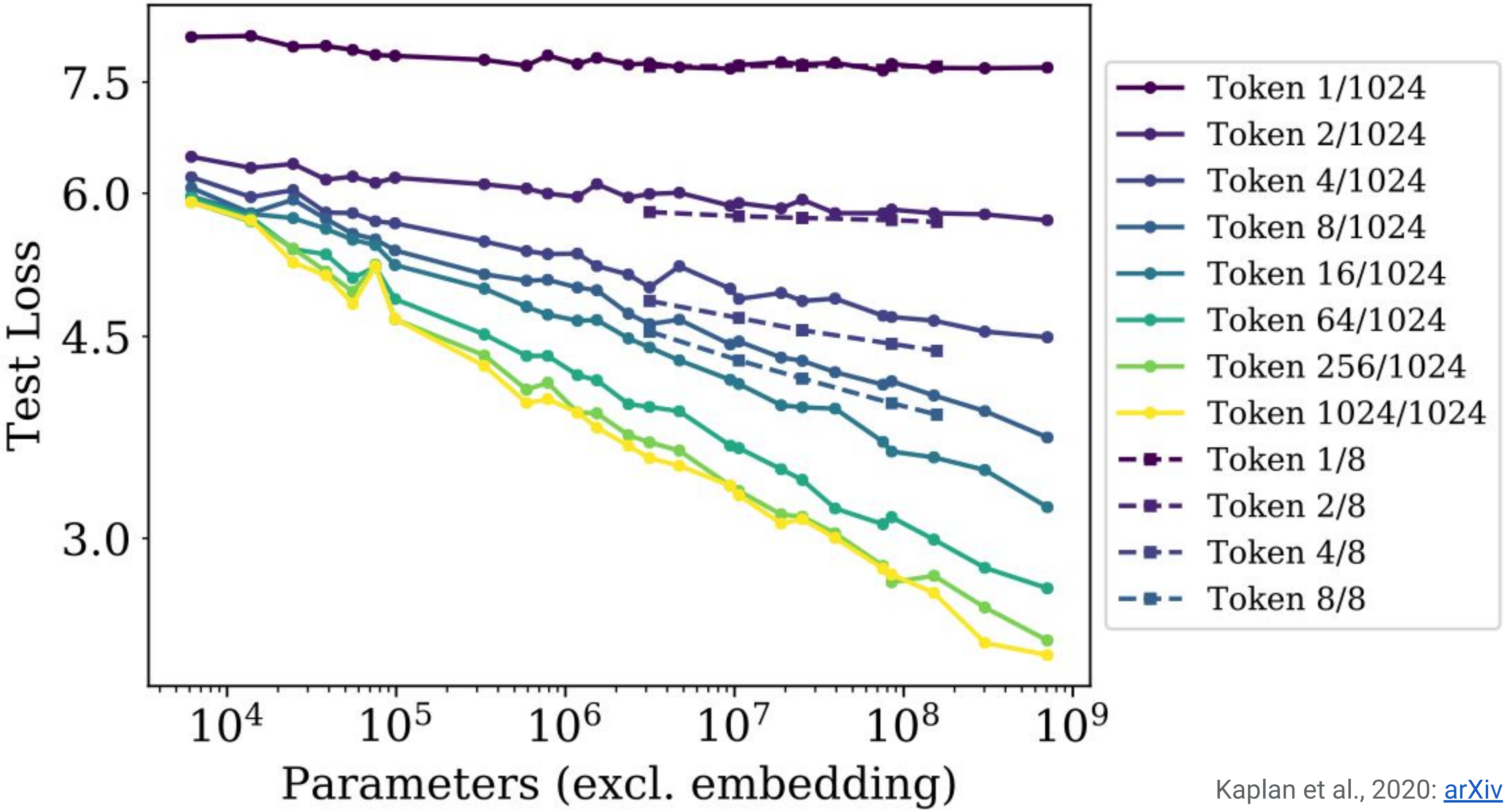
Why Do We Need Scale?

Scale More Important Than Architecture



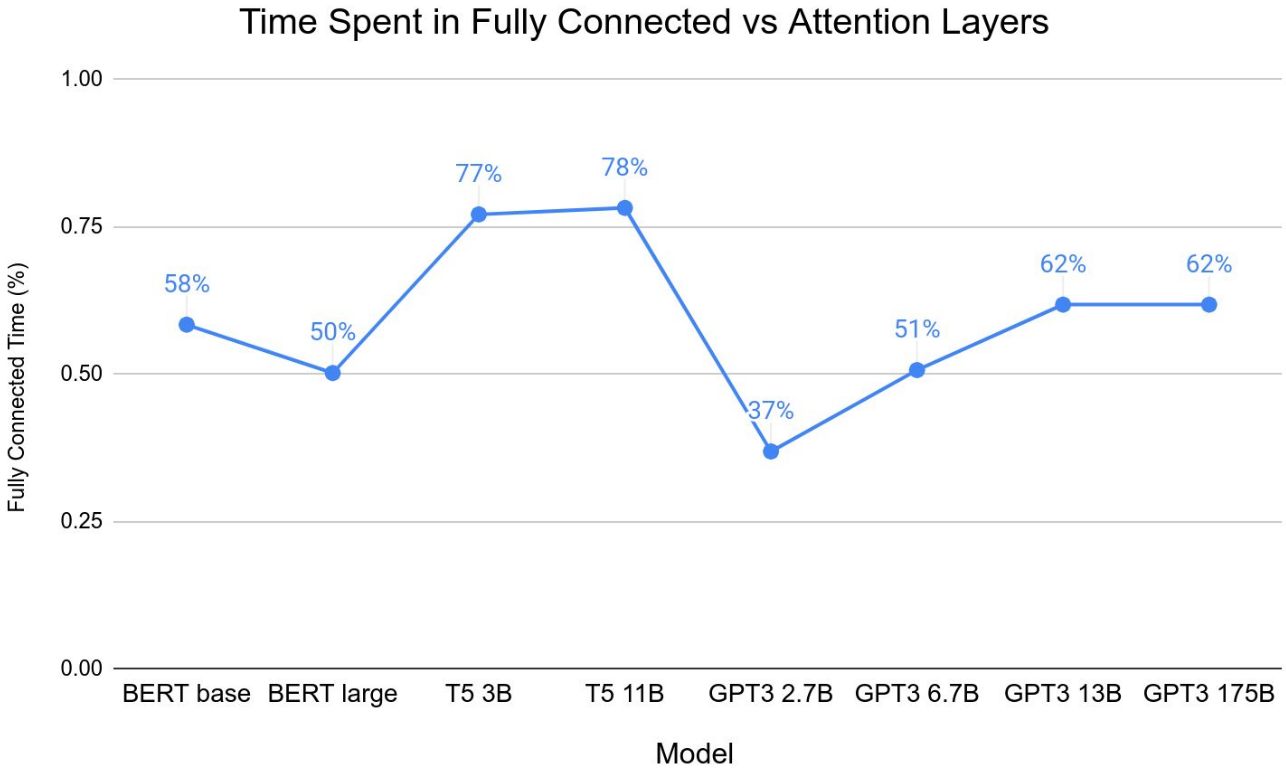
Kaplan et al., 2020: arXiv

Attention Size vs Model Size vs Test Loss



Kaplan et al., 2020: arXiv

Attention vs Fully Connected Time for Various Transformers



Conclusions From Measuring Scaling

- Performance increases further and further the more parameters a model has
- Attention is very important for efficiency: Transformers scale better than LSTMs
- Attention has diminishing returns (on general “internet data”)
- Size of data and model are more important than architecture

Practical Considerations

Experimental vs Theoretical Perspective

Theoretical:

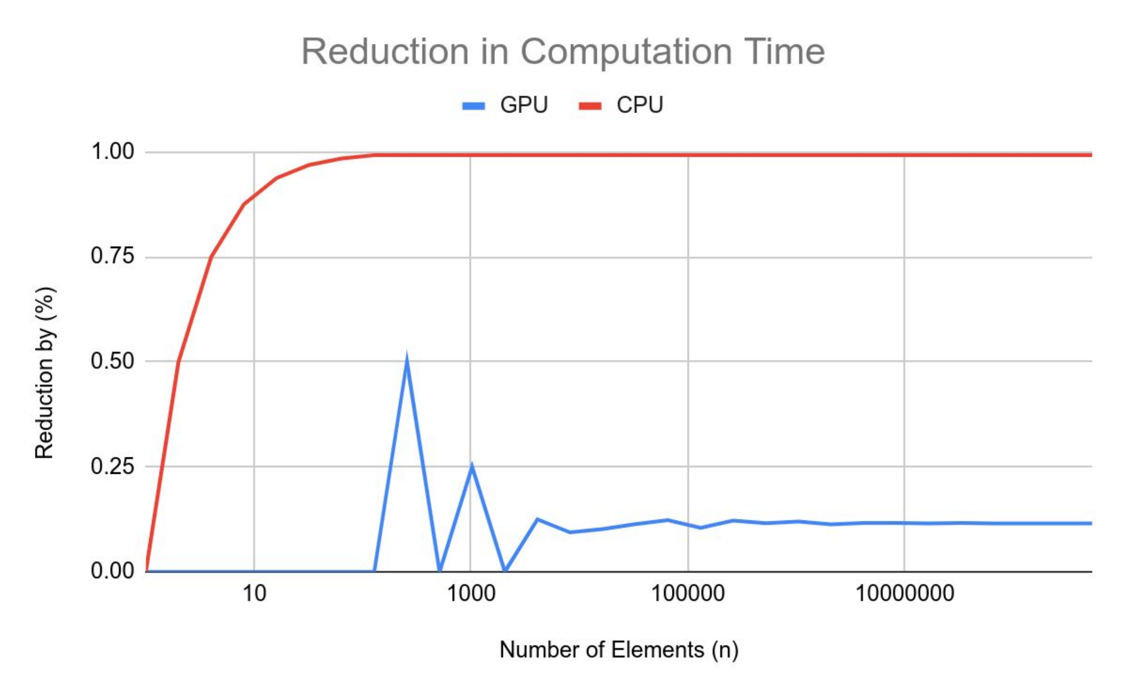
- FLOPS/Operation Complexity/Memory: O(n) better than O(n^2); 100 FLOPS better than 1000
- (Possibly) Analysis of occupancy, memory access patterns for certain hardware

Experimental:

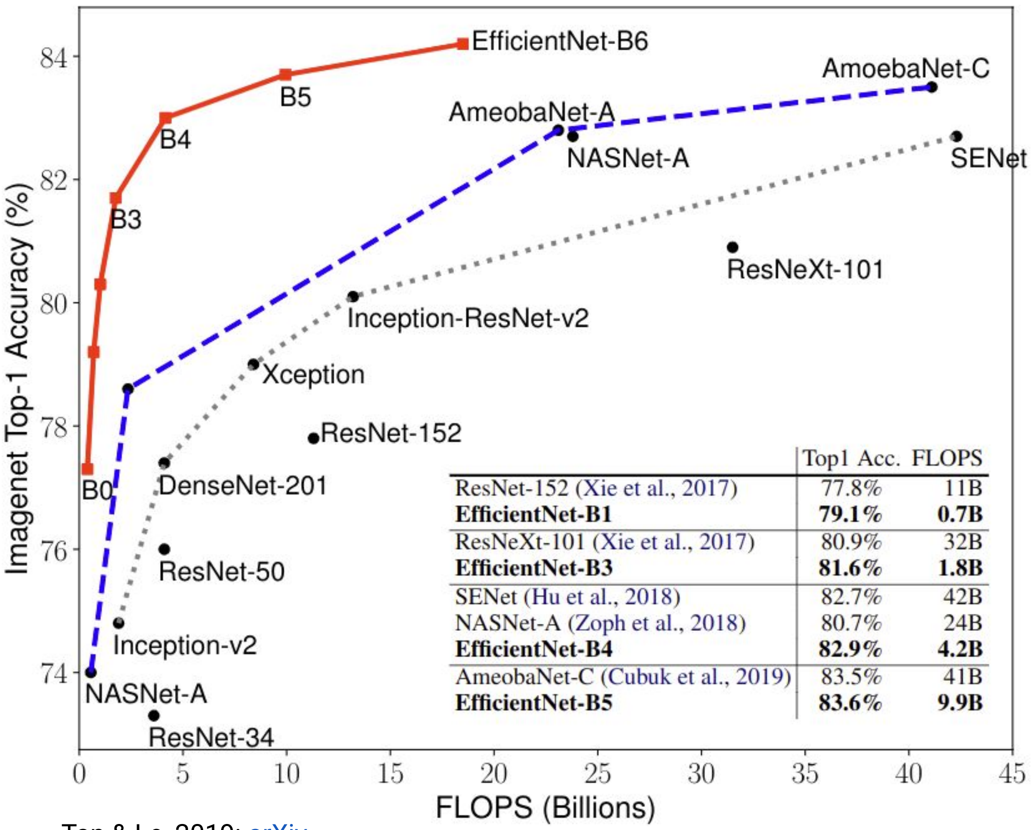
- Three criteria:
    a. Does it fit into my GPU/TPU/Accelerator?
    b. Is it faster than other methods?
    c. Can most people use it (+62% of PhD students)?
- Device oriented walltime/memory: CPU for inference, GPU/TPU for training

Theory vs Practice

- Algorithm: (1) Divide matrix B into chunks of 128; (2) take the maximum element, set others to zero; (3) perform matrix multiply A*B=C and skip all zero elements

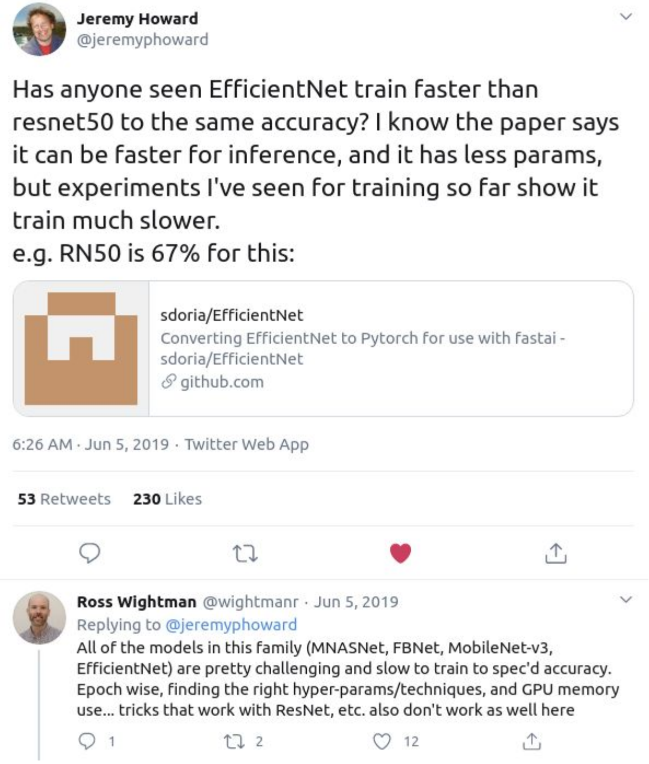


Theory vs Practice



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Tan & Le, 2019: arXiv



Jeremy Howard
@jeremyphoward

Has anyone seen EfficientNet train faster than resnet50 to the same accuracy? I know the paper says it can be faster for inference, and it has less params, but experiments I've seen for training so far show it train much slower.
e.g. RN50 is 67% for this:

sdoria/EfficientNet
Converting EfficientNet to Pytorch for use with fastai - sdoria/EfficientNet
github.com

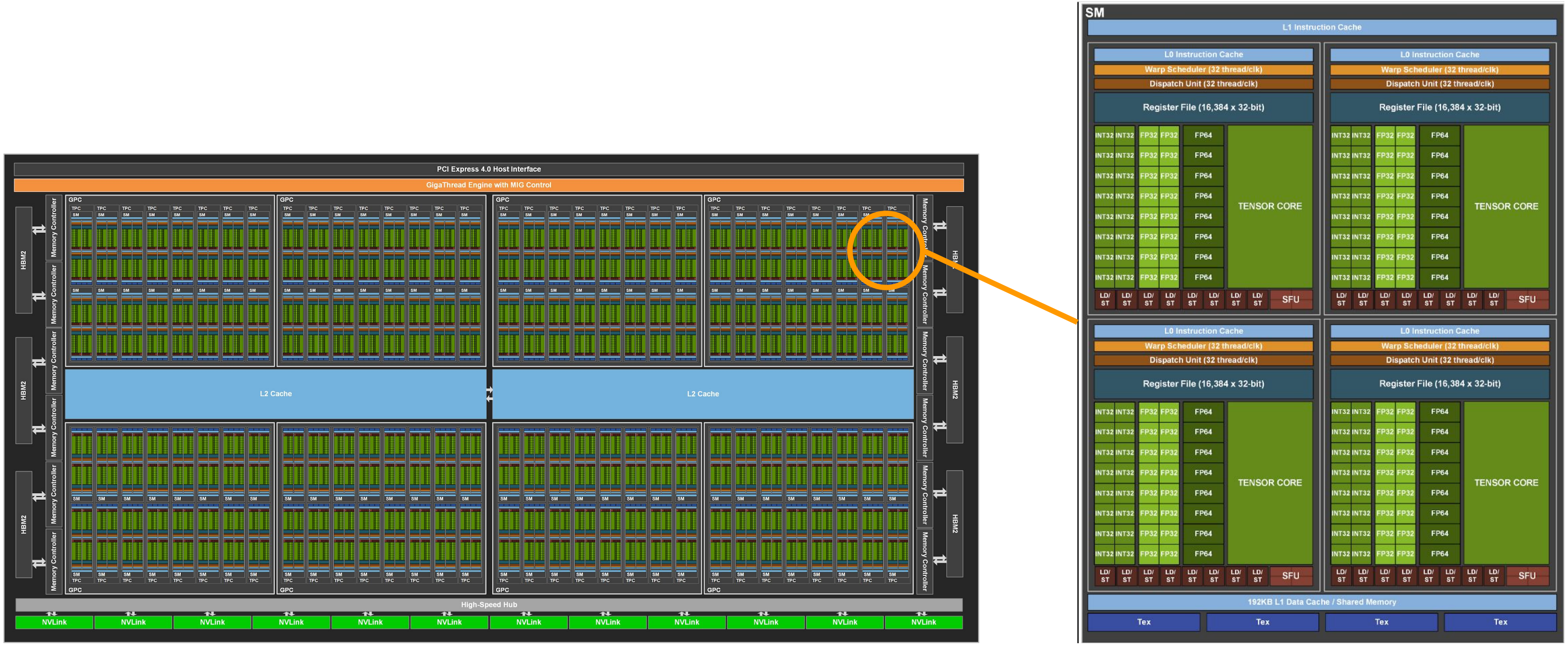
6:26 AM · Jun 5, 2019 · Twitter Web App

53 Retweets **230** Likes

Ross Wightman @wightmanr · Jun 5, 2019
Replying to @jeremyphoward
All of the models in this family (MNASNet, FBNet, MobileNet-v3, EfficientNet) are pretty challenging and slow to train to spec'd accuracy. Epoch wise, finding the right hyper-params/techniques, and GPU memory use... tricks that work with ResNet, etc. also don't work as well here

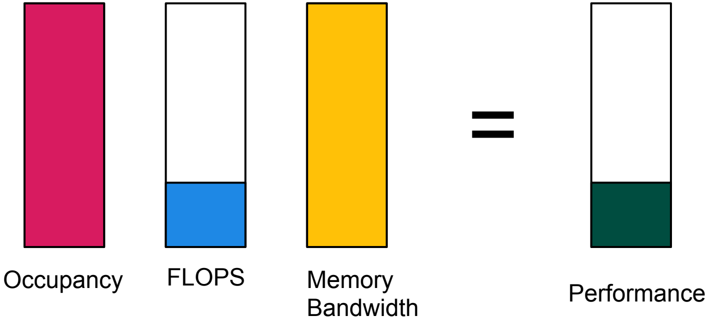
1 2 12

GPU Architecture

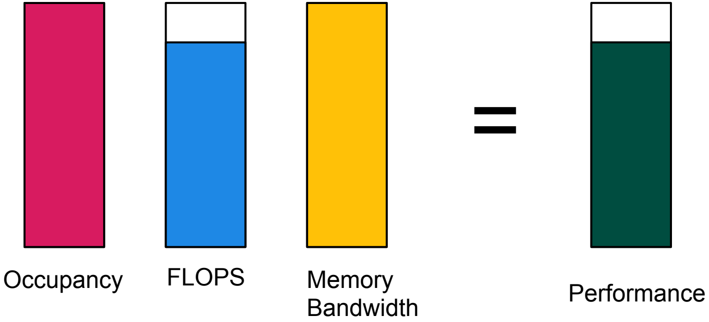

PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
HBM2
Memory Controller
L2 Cache
High-Speed Hub
NVLink
SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

Ampere Architecture (NVIDIA)

Occupancy vs Memory Bandwidth vs FLOPS

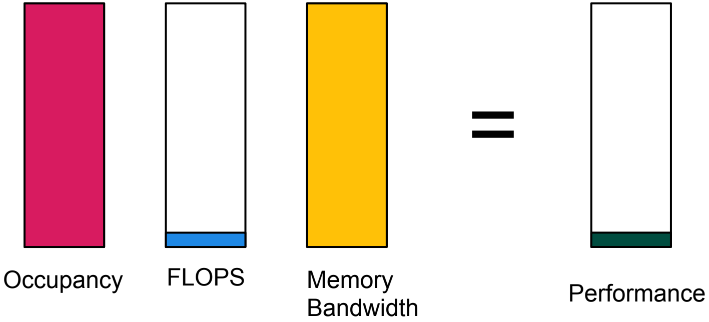


Matrix multiplication

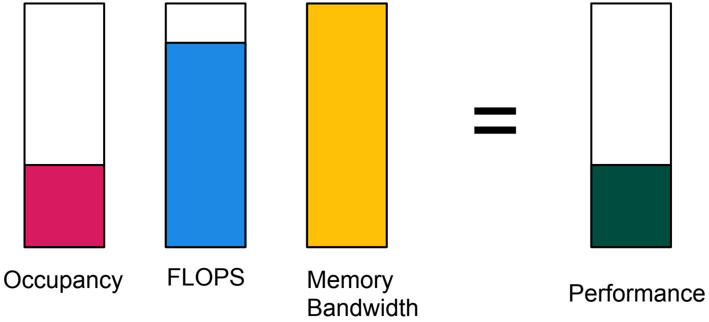


Convolution

Occupancy vs Memory Bandwidth vs FLOPS

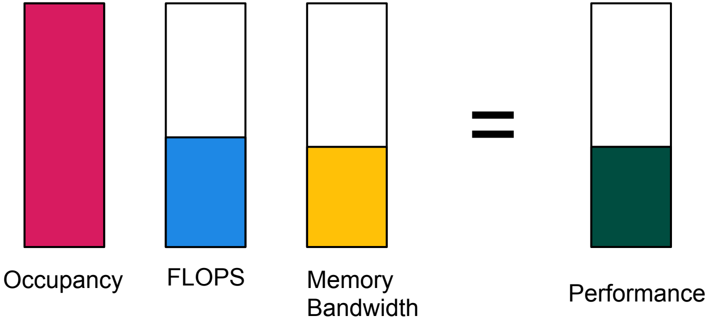


Sparse Matrix multiplication

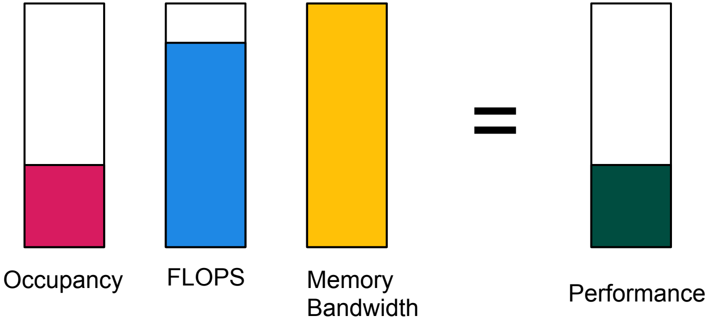


Depthwise Convolution

Occupancy vs Memory Bandwidth vs FLOPS

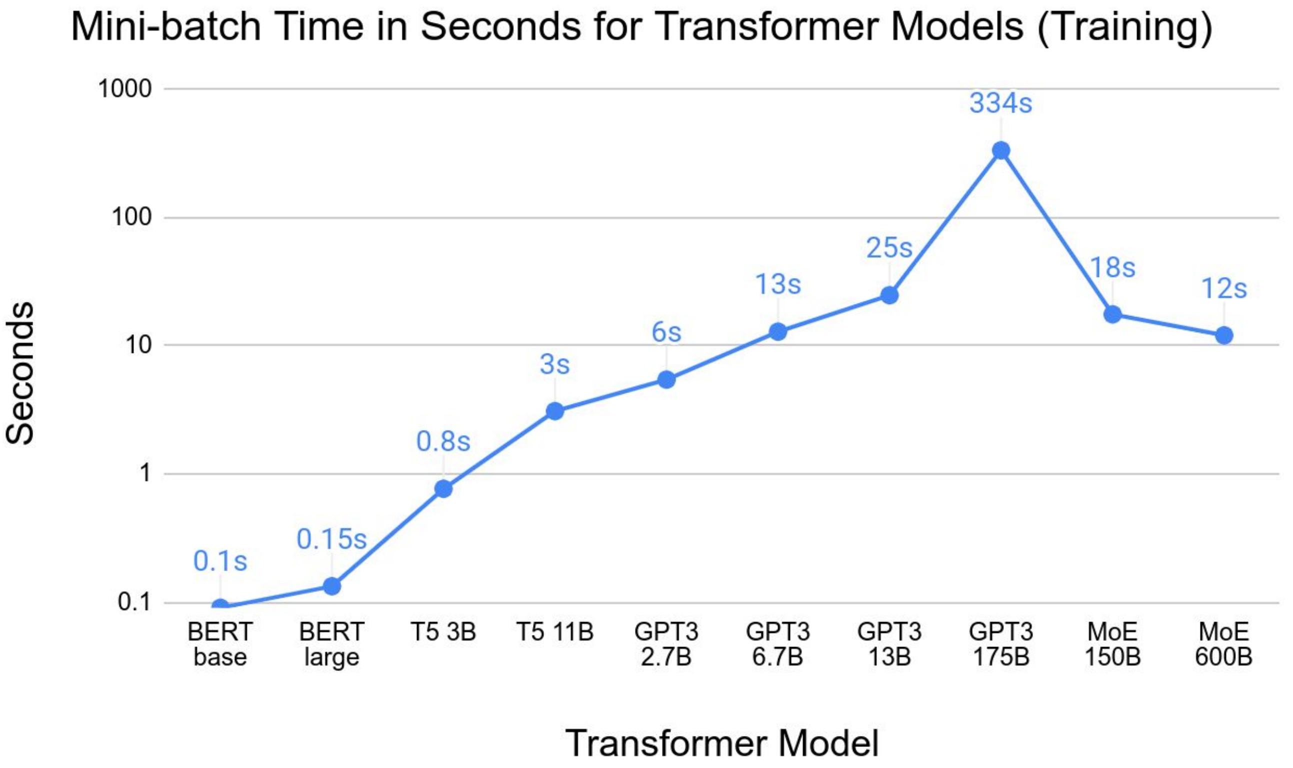


Depthwise Convolution
(custom implementation)



Depthwise Convolution

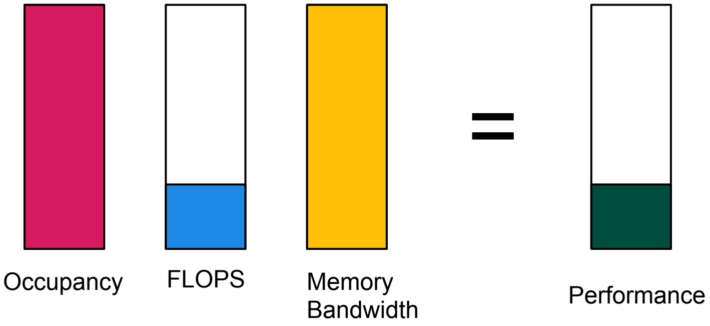
BERT Large vs BERT Base



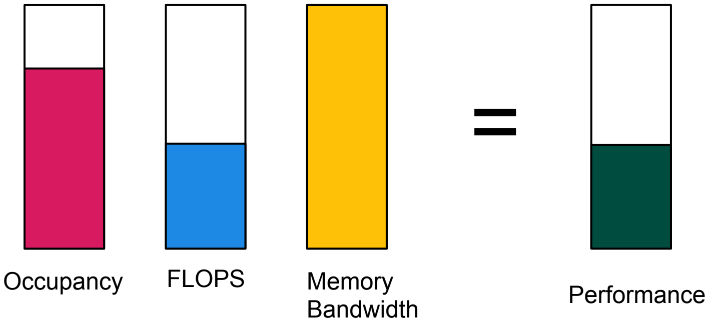
BERT Base is 3.1x smaller than BERT Large but only trains 1.5x faster.

BERT Base is too small to saturate modern GPUs.

Better Performance at Lower Occupancy



Matrix multiplication



Matrix Multiplication
(lower occupancy, higher instruction parallelism)

Volkov, 2010

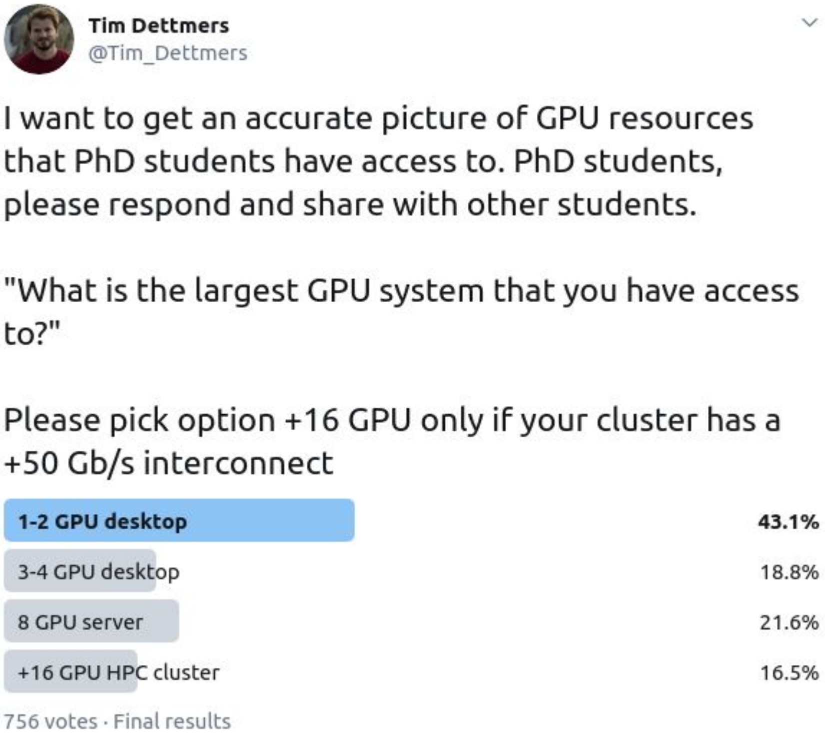
Conclusion

- Occupancy, and FLOPS/memory bandwidth utilization are important for runtime performance
- Understanding of hardware needed for performance analysis
- Even with deep understanding of hardware, it is difficult to analyze performance theoretically
- Runtime performance of different algorithms can often only be understood if they are run on the actual device

- Conclusion: To estimate deep neural network runtime performance, it is best to run the network and measure its performance directly.

Memory Optimizations

Resources: Academia vs Industry



Tim Dettmers
@Tim_Dettmers

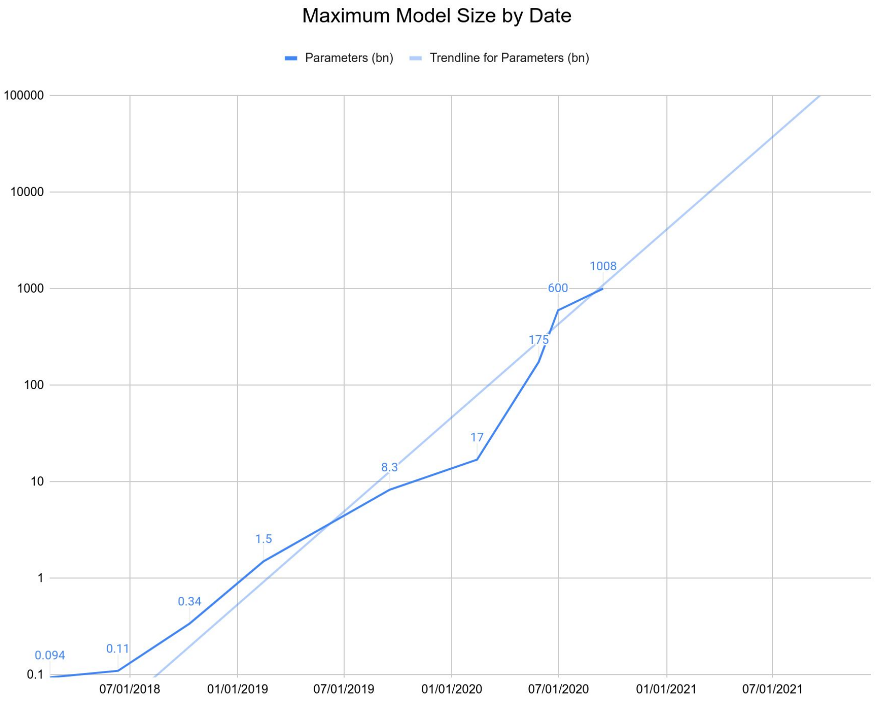
I want to get an accurate picture of GPU resources that PhD students have access to. PhD students, please respond and share with other students.

"What is the largest GPU system that you have access to?"

Please pick option +16 GPU only if your cluster has a +50 Gb/s interconnect

1-2 GPU desktop	**43.1%**
3-4 GPU desktop	18.8%
8 GPU server	21.6%
+16 GPU HPC cluster	16.5%

756 votes · Final results

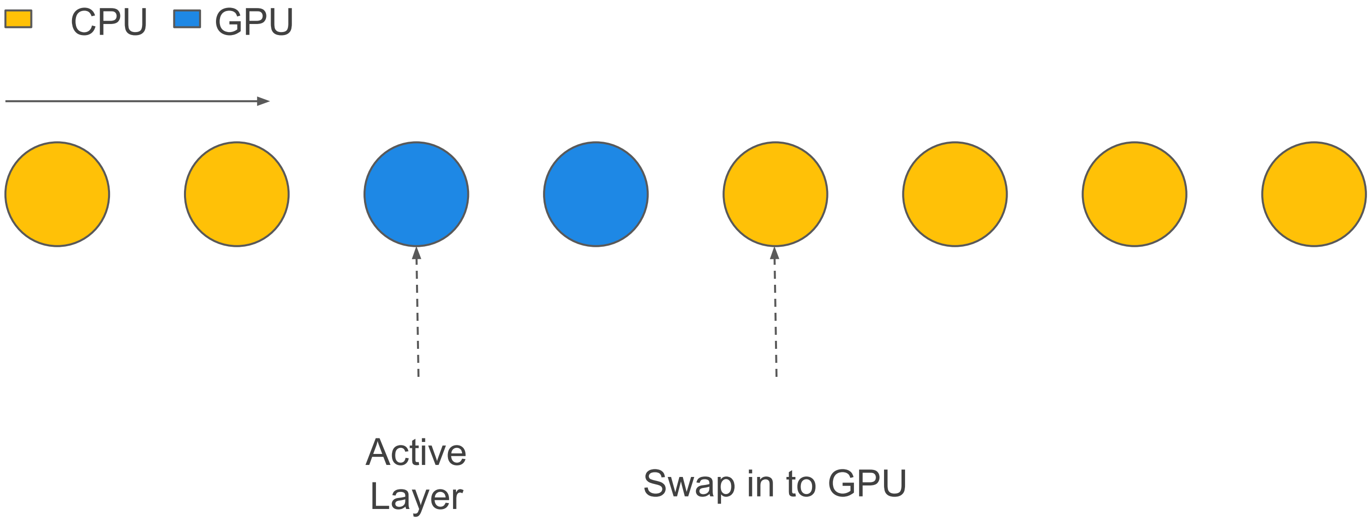


Memory Optimizations Overview

- Memory Swapping/Memory Paging
- FP16/BF16 training
- Gradient checkpointing
- Gradient accumulation
- Reversible residual connections

CPU<->GPU Memory Swapping / Paging

- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



🟨 CPU 🟦 GPU

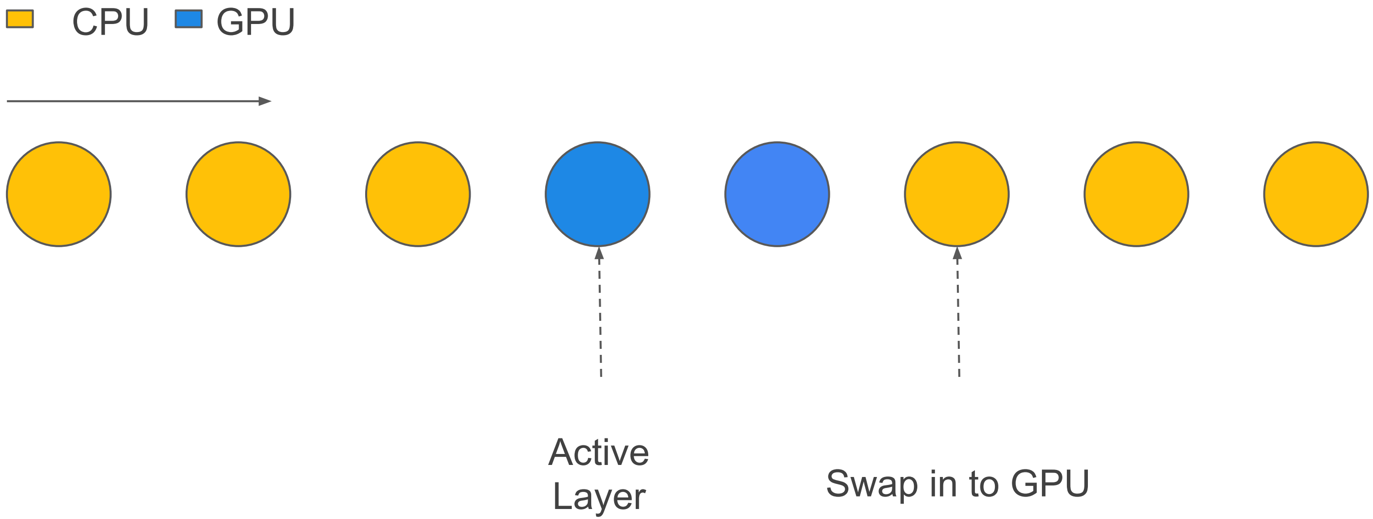
Active Layer

Swap in to GPU

Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

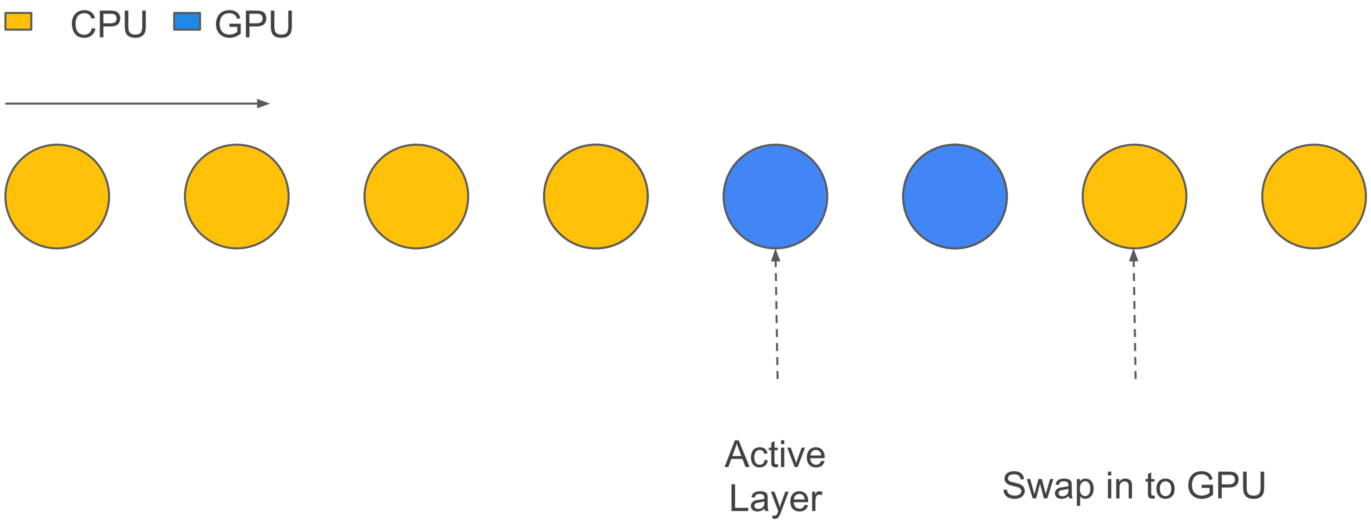
- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

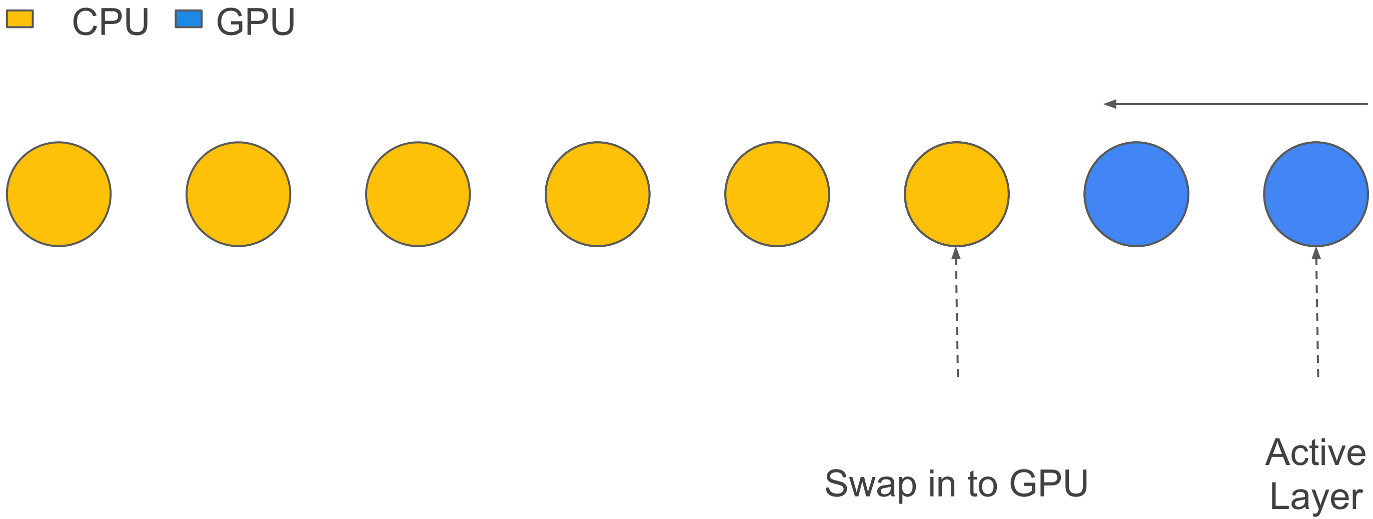
- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

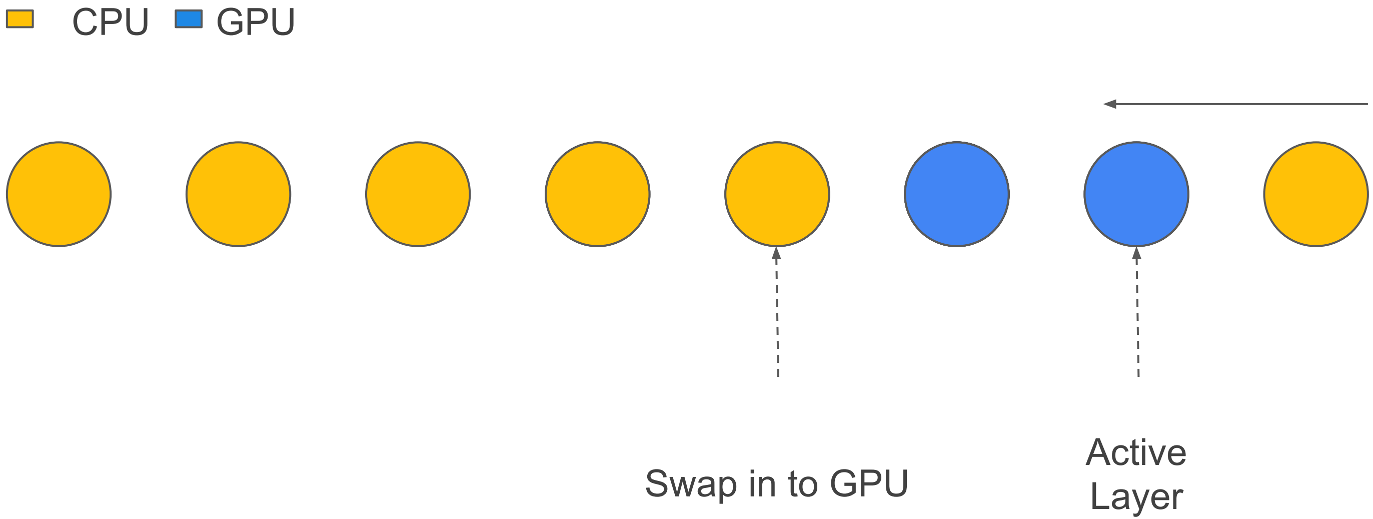
- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

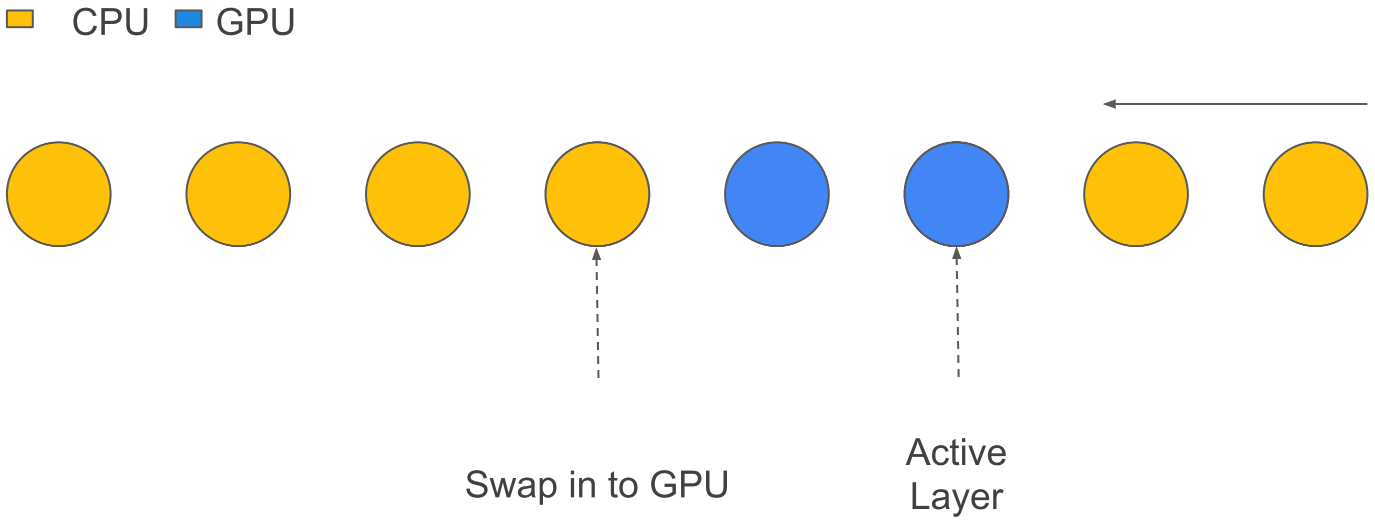
- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time



🟨 CPU 🟦 GPU

Swap in to GPU

Active Layer

Pupipeddi et al., 2020: arXiv

CPU<->GPU Memory Swapping / Paging

- Swap-out activations / weights to CPU once a layer is completed
- Swap-in activations /weights to GPU before a layer is started
- Exact timing of swap-in/swap-out depends on layers size and layer forward/backward time

- Benefits:
  - 60-80% memory reduction
  - Network usually not slower. If it is slower, swap-int layers earlier (less memory reduction)
  - Faster training due to larger batch size for very large models

Mixed Precision Training (FP16+FP32) / BF16 training

Mixed Precision Training:

- Keep 32-bit master weights
- Do forward pass with 16-bit
- Scale 16-bit loss to prevent under/overflow
- Compute gradients
- Update 32-bit weights; copy 32-bit weights to 16-bit buffers

BrainFloat-16 Training:

- Range: FP16 +-65504; BF16 & FP32 -+3e^38
- Cast everything to BF16
- Train normally (no under/overflow due to larger range)

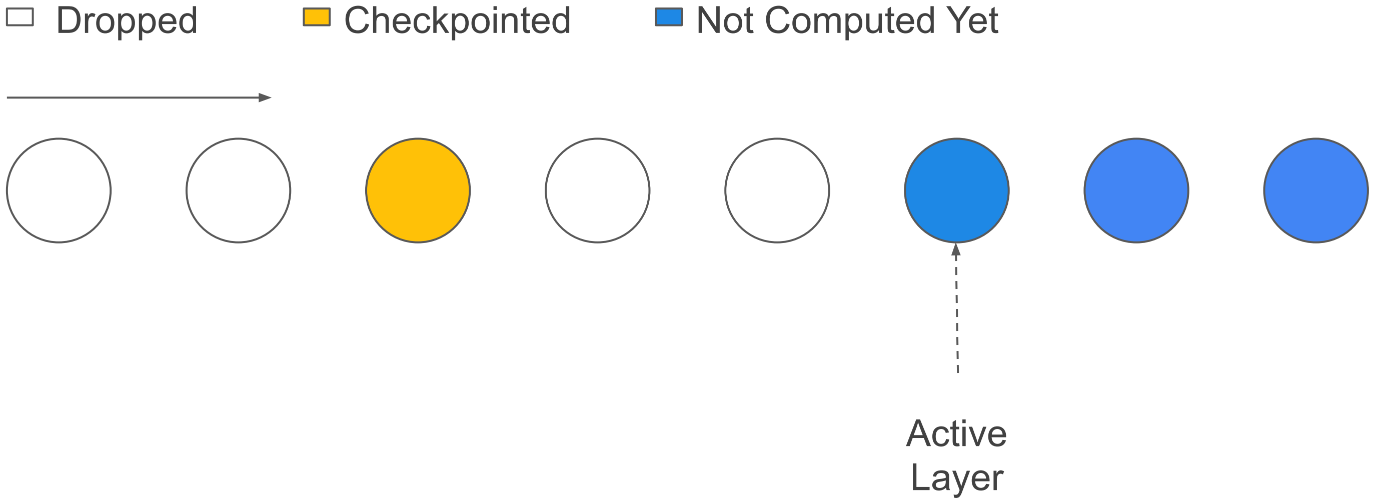
Benefits:

- Faster training, depending on network about 2x speedup
- Usually save some memory, especially if your activations are large

Micikevicius et al., 2018: arXiv

Gradient Checkpointing: Forward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

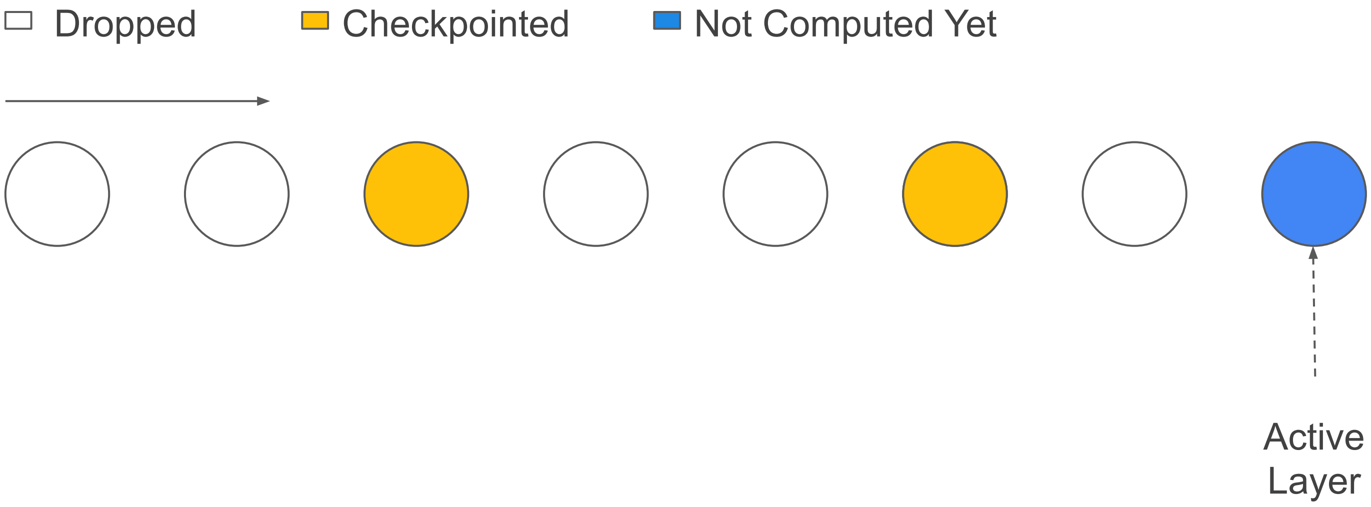


□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet

Chen et al., 2016: arXiv

Gradient Checkpointing: Forward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

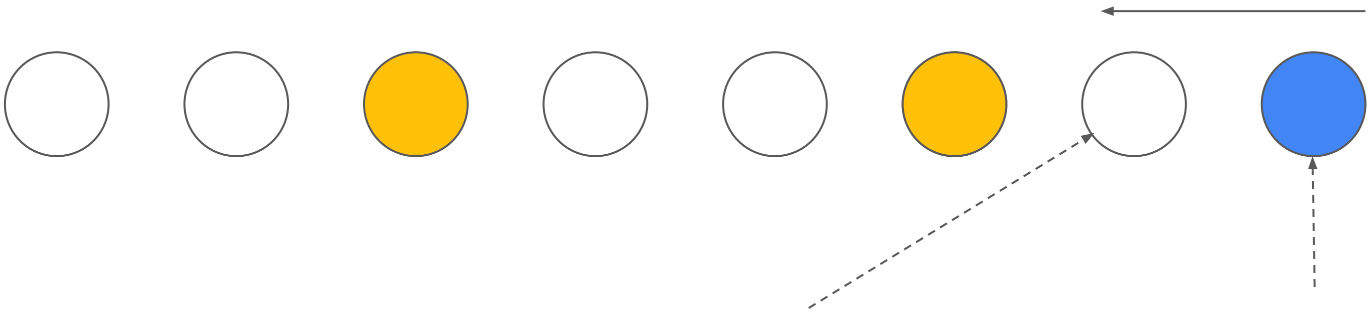


Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient



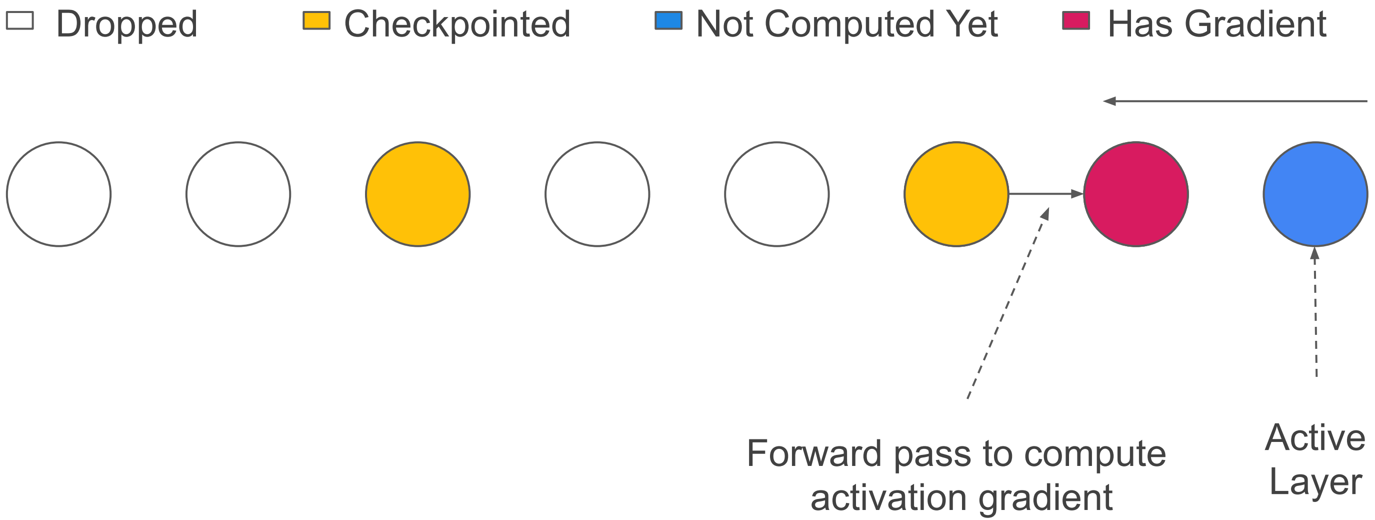
Missing gradients!
Recompute from last checkpoint
with forward pass

Active
Layer

Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node



□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient

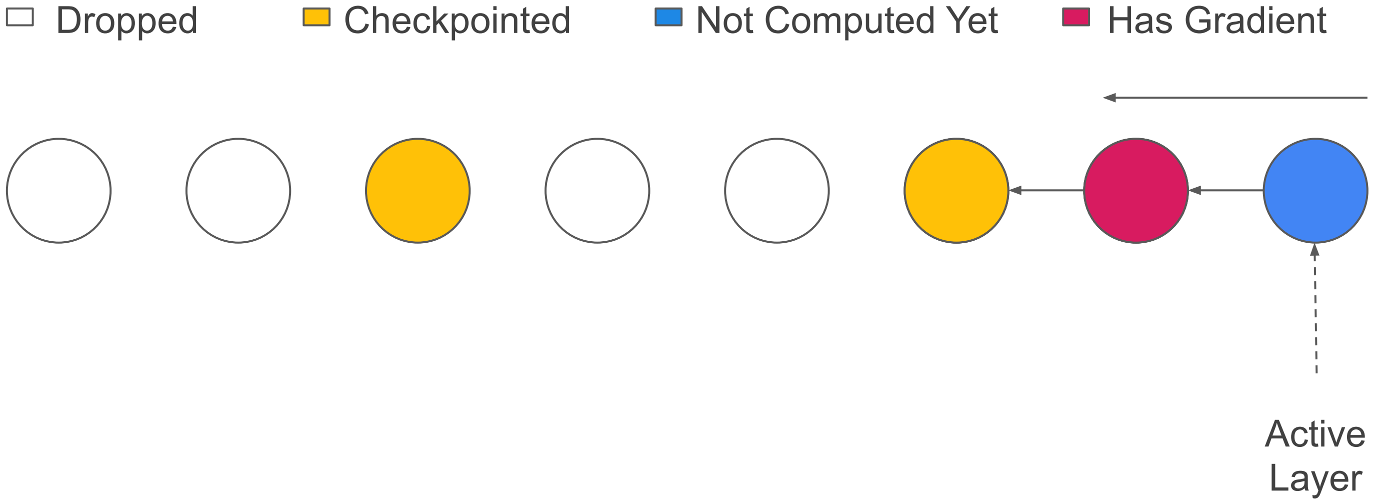
Forward pass to compute activation gradient

Active Layer

Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node



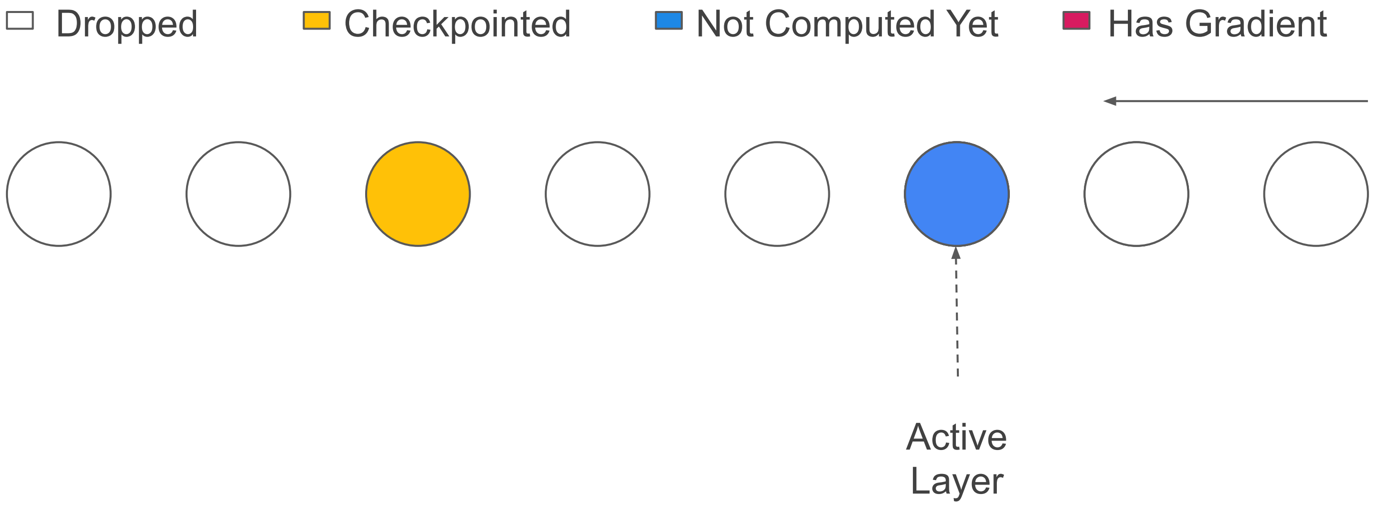
□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient

Active Layer

Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

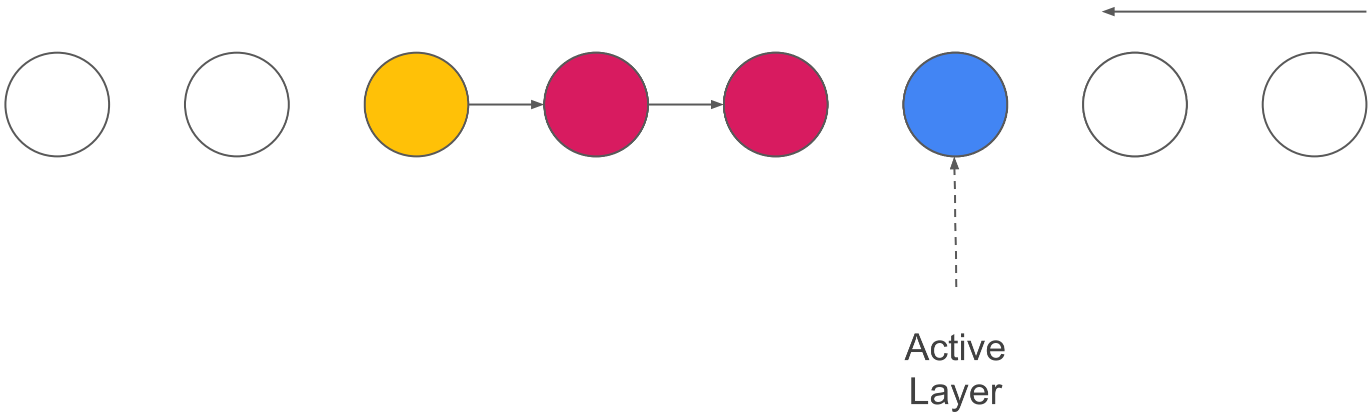


Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

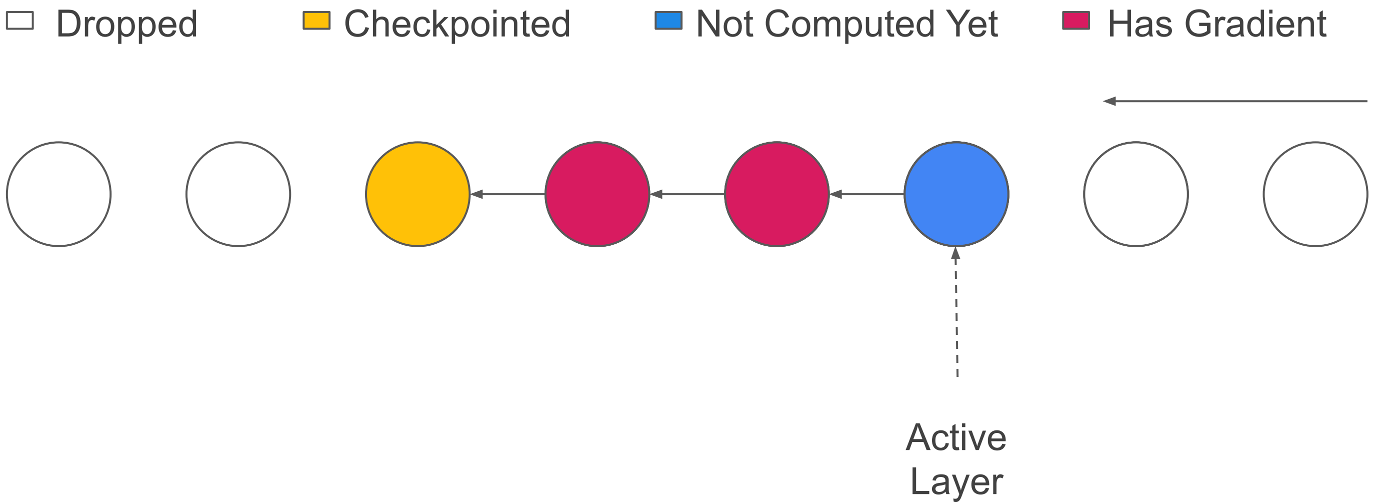
□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient



Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node



□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient

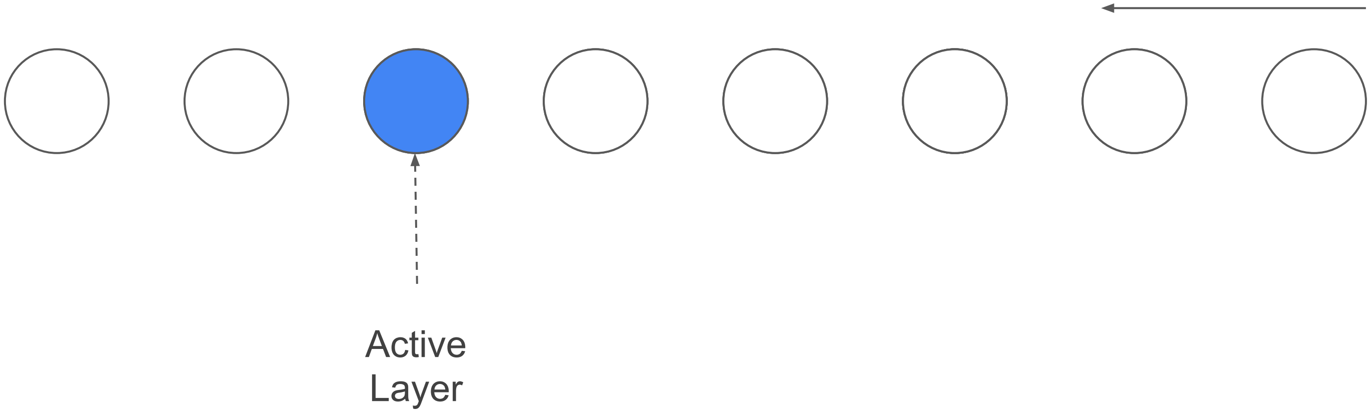
Active Layer

Chen et al., 2016: arXiv

Gradient Checkpointing: Backward

- Do not store activation gradients in the forward pass
- Recompute activation gradients in the backward pass by restarting a forward pass from a checkpoint node

□ Dropped 🟨 Checkpointed 🟦 Not Computed Yet 🟥 Has Gradient



Chen et al., 2016: arXiv

Gradient Checkpointing

Benefits:

- Trade computation to reduce memory footprint
- Best used for functions that are cheap to compute but have a large activation gradient (ReLU, layer norm, softmax)
- Very beneficial for nonlinear activation functions
- Easy to use in PyTorch (torch.utils.checkpoint) and TensorFlow 2.0 (recompute_grad (nightly))

Reversible Residual Connections

- Divide network output and residual connection into two halves. Compound them into a reversible structure:

Forward

$$y_1 = x_1 + \mathcal{F}(x_2)$$
$$y_2 = x_2 + \mathcal{G}(y_1)$$

Backward

$$x_2 = y_2 - \mathcal{G}(y_1)$$
$$x_1 = y_1 - \mathcal{F}(x_2)$$

Benefits:

- Saves some memory for free (if your framework supports it, e.g. JAX)
- Usually, gradient checkpointing should be prefered:
  - Can save more memory due to being more general.
  - Easy to implement. Supported by major frameworks.

Gomez et al., 2017: arXiv

Gradient Accumulation

- Split larger mini-batches into “micro-batches”
- Do standard forward/backward passes with micro-batches, but do not update the weights right away (and do not reset the gradient on the weights)
- Accumulate the gradient on the weights for all micro-batches
- Update the weights once enough micro-batches have been computed

Benefits / Tradeoffs:

- As long as your model runs with batch size 1 you can simulate any batch size
- Easy to implement and can reduce memory footprint significantly
- Slow if micro-batch size is very small
- Can improve data parallel performance significantly (speedups) especially for very large models

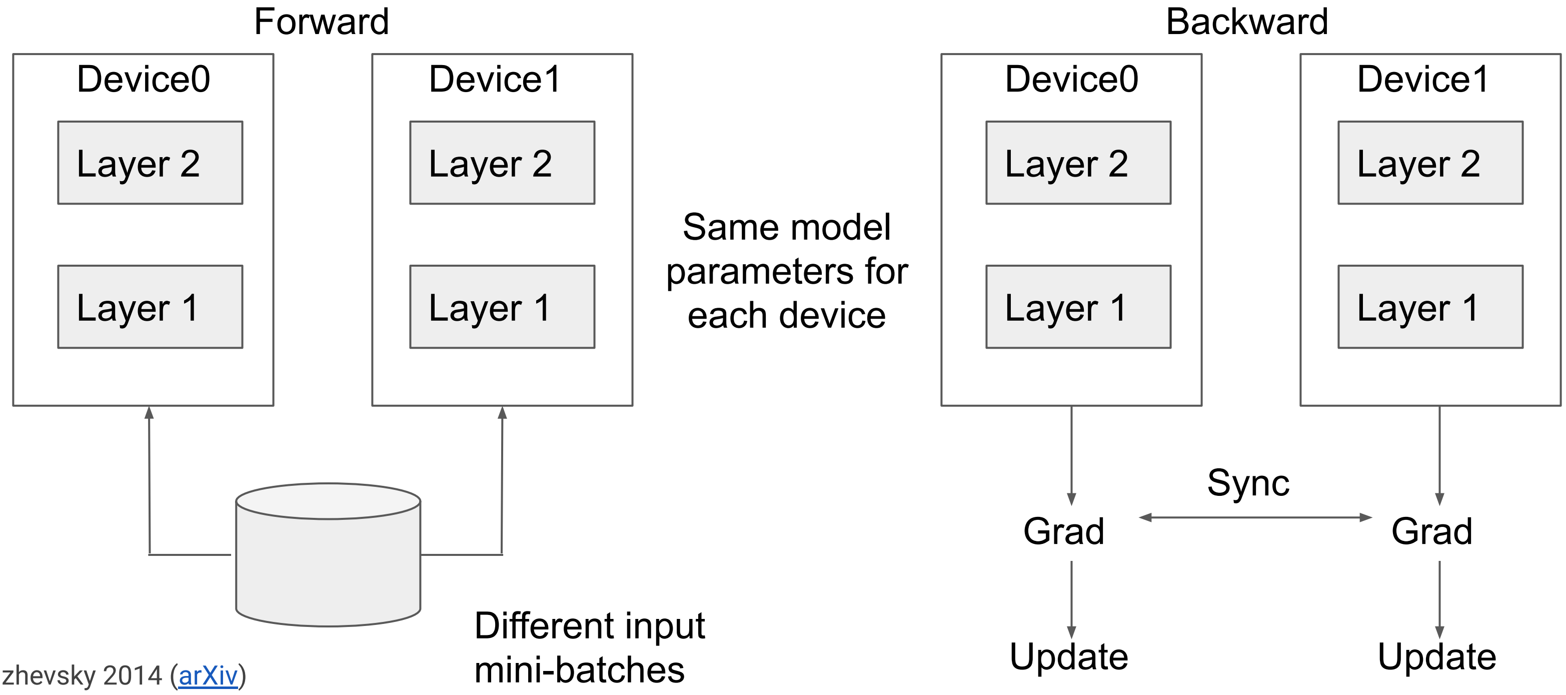
Parallelism

Parallelism Overview

- Data parallelism
- Model parallelism
- Pipeline parallelism
- ZeRo parallelism optimizations
- 3D parallelism

Data Parallelism

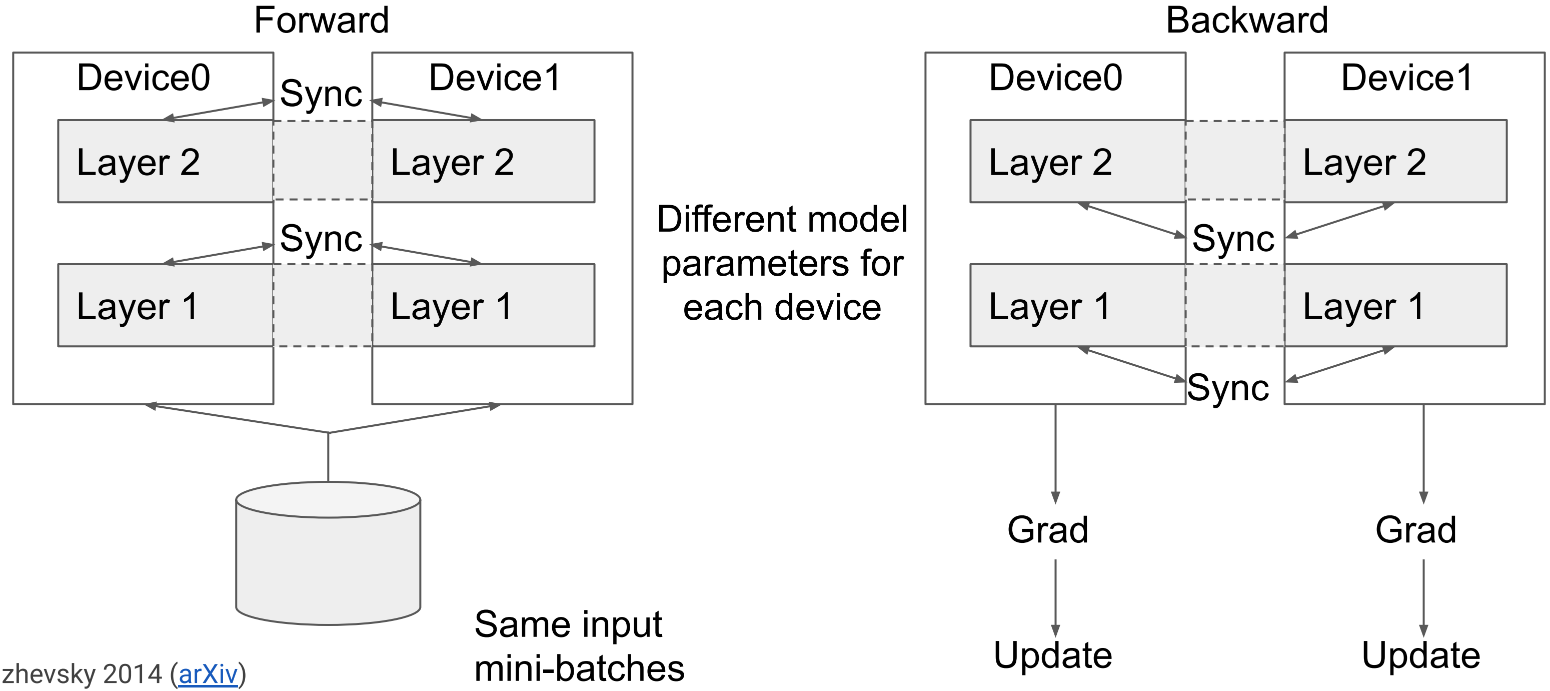
Idea: Keep the same model parameters across multiple accelerators. Feed them different mini-batches and average the gradient across accelerators.



Krizhevsky 2014 (arXiv)

Model Parallelism

Idea: Keep the same mini-batch across multiple accelerators; split the layers parameters across all devices and synchronize layer outputs after each layer.



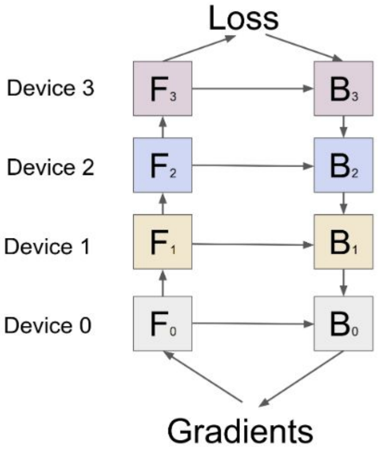
Different model parameters for each device

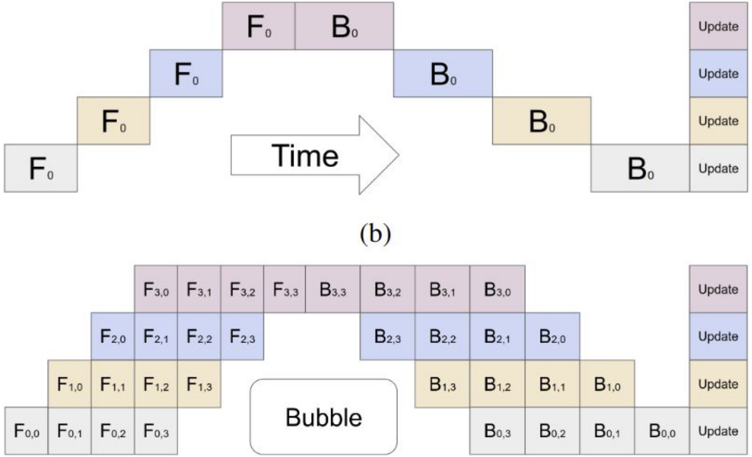
Same input mini-batches

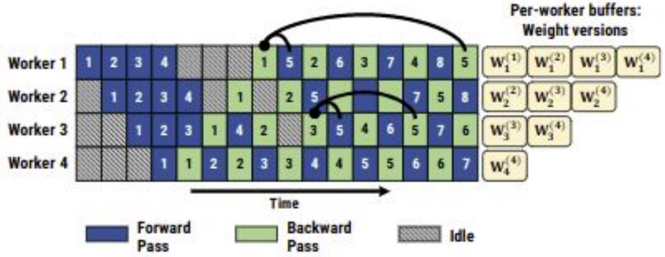
Krizhevsky 2014 (arXiv)

Pipeline Parallelism

Idea: Split network by depth into k pieces onto k accelerators. Each accelerator holds 1/kth of layers. Use micro-batches to overlap computation and communication.



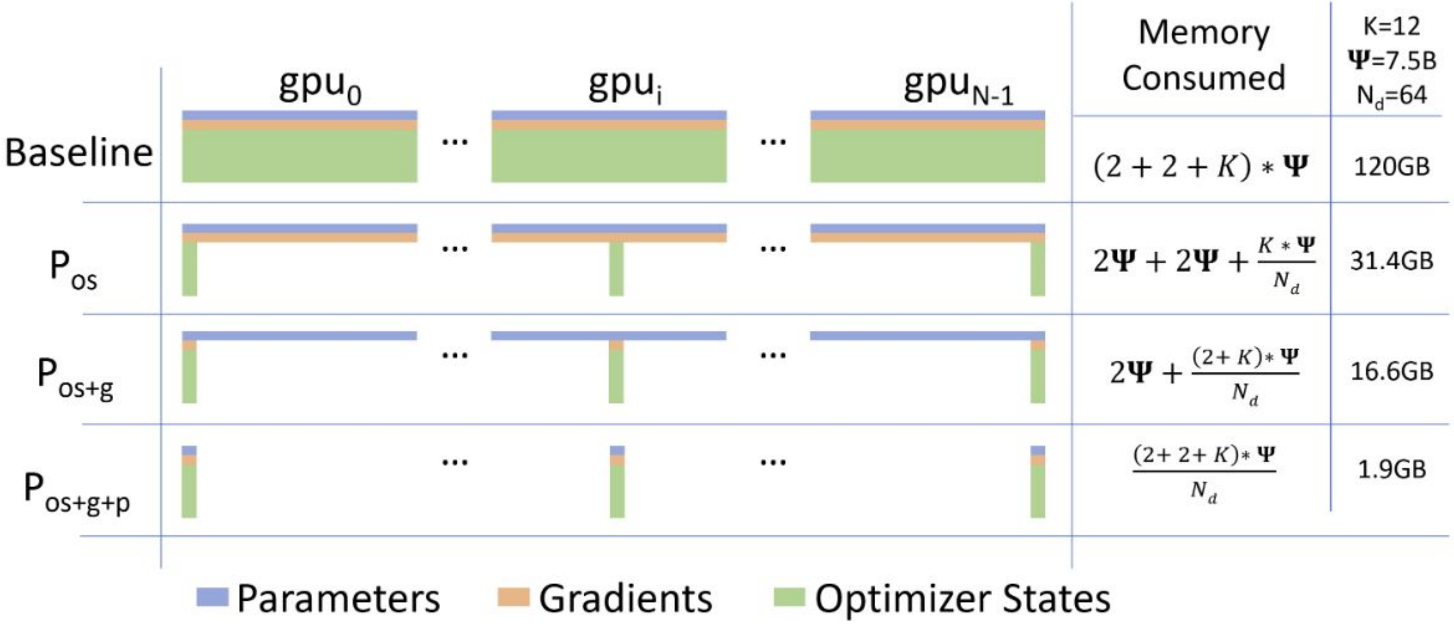




Krizhevsky 2014 (arXiv); Harlap et al., 2018 (arXiv); Huang et al., 2018 (arXiv)

ZeRO Parallelism Optimizations

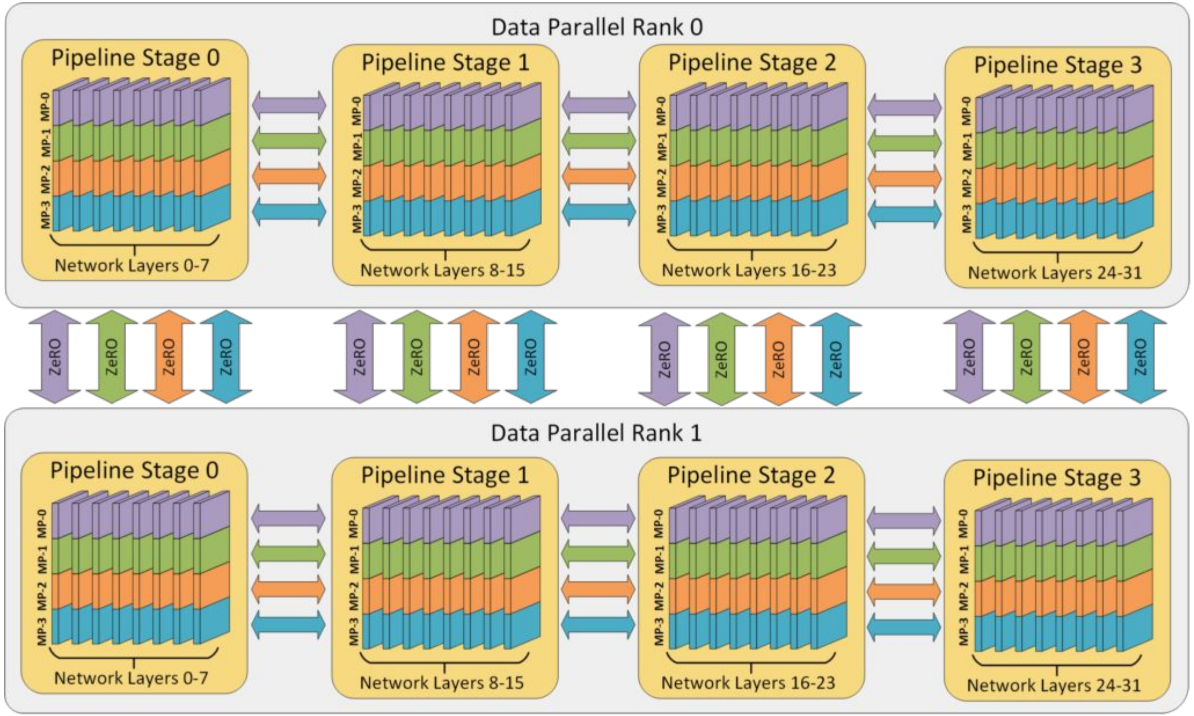
Idea: Gradients, parameters, and optimizer state only needed for active layer. We distribute the state across all GPUs and gather them together when we need them (when they become “active”).

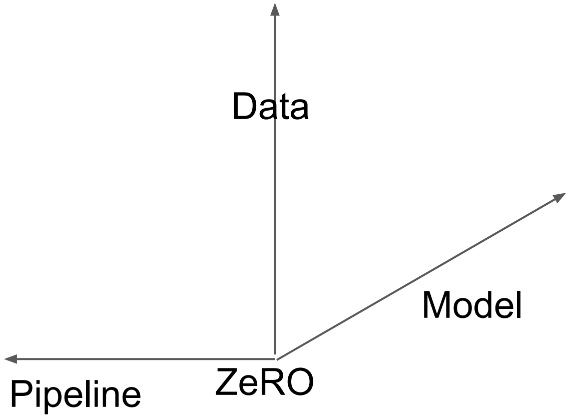


	Memory Consumed	K=12 Ψ=7.5B N_d=64
Baseline	$(2+2+K)*\Psi$	120GB
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB

Rajbhandari et al., 2020 (arXiv)

3D Parallelism





Microsoft Blog Post
(paper coming soon?)

Why 3D Parallelism?

- Model parallelism bad if batch size is too large. Communication cannot be overlapped with computation.
- Data parallelism bad if the batch size is too small.
- Pipeline parallelism decreases mini-batch size through micro-batches.
- Pipeline parallelism increase min-batch size through aggregation of micro-batches.
- Pipeline parallelism allows for simple overlap of communication and computation..

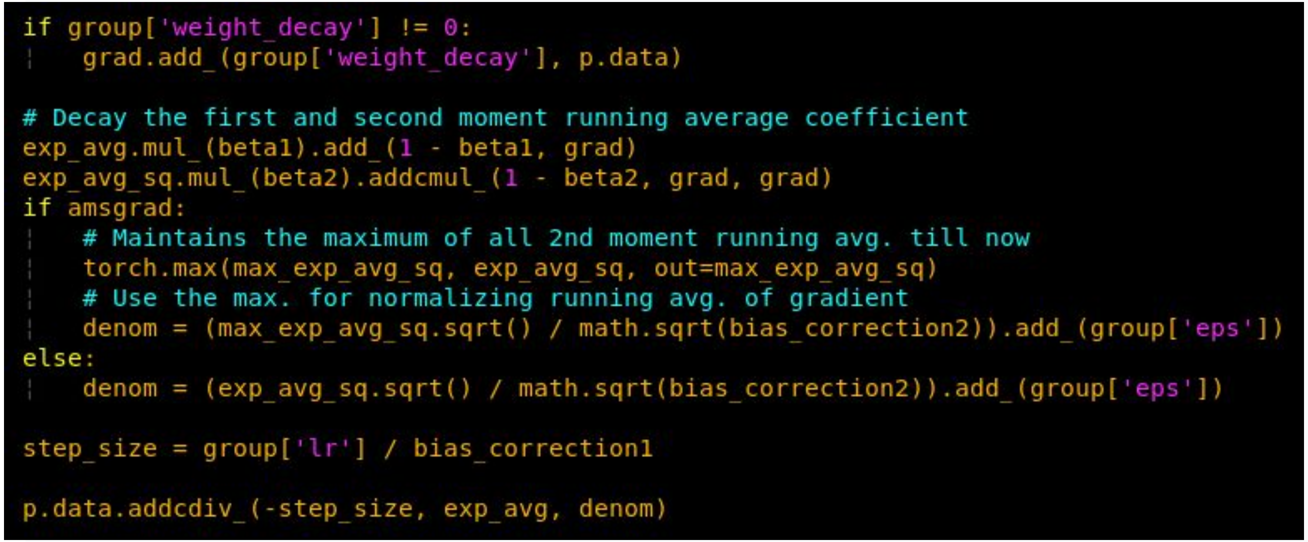
Microsoft Blog Post
(paper coming soon?)

Efficiency Optimizations

Larger Batch Size

- GPUs are more efficient if fully utilized. That usually only happens if batch size is large
- GPUs run better if the mini-batch dimension is 32 or larger
- Often you can achieve *faster* training by using a memory efficiency technique which *slows* down training but enables training with larger batch size
- Larger batch sizes enables larger learning rates. While computation is slower, training might be faster.

Fused Kernels

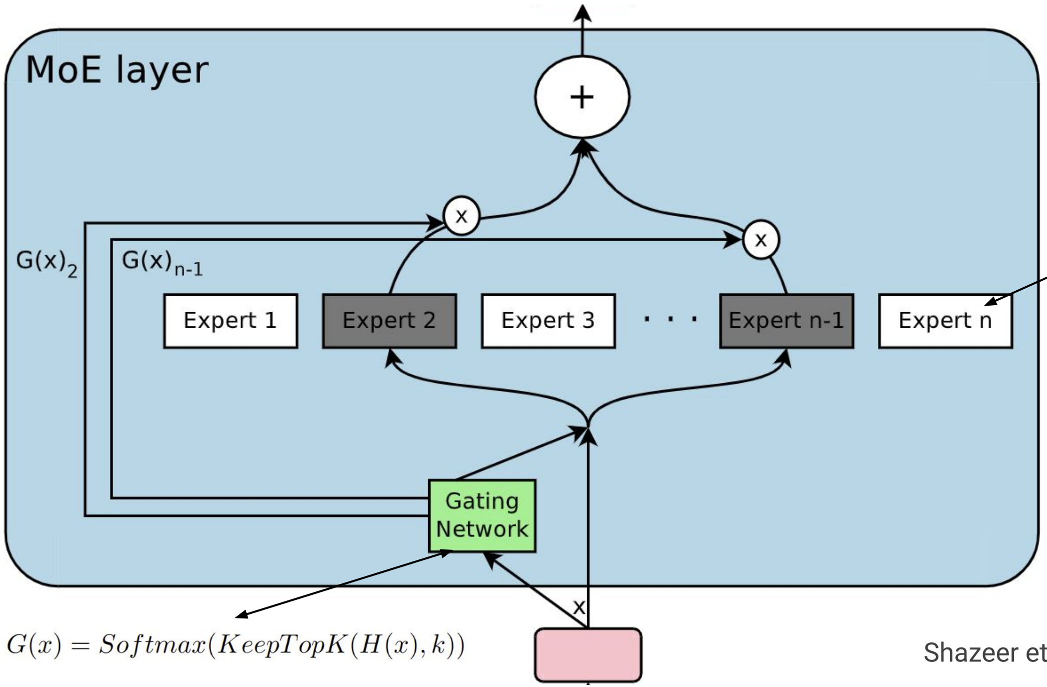


```
if group['weight_decay'] != 0:
    grad.add_(group['weight_decay'], p.data)

# Decay the first and second moment running average coefficient
exp_avg.mul_(beta1).add_(1 - beta1, grad)
exp_avg_sq.mul_(beta2).addcmul_(1 - beta2, grad, grad)
if amsgrad:
    # Maintains the maximum of all 2nd moment running avg. till now
    torch.max(max_exp_avg_sq, exp_avg_sq, out=max_exp_avg_sq)
    # Use the max. for normalizing running avg. of gradient
    denom = (max_exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(group['eps'])
else:
    denom = (exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(group['eps'])

step_size = group['lr'] / bias_correction1

p.data.addcdiv_(-step_size, exp_avg, denom)
```

- Adam with 10^9 parameters:
  - 14 read/writes
  - 32-bit 10^9 parameters = 4 GB
  - Normal Adam: GPU with 600 GB/s -> 14*4/600 = 100ms
  - Fused Adam: 6ms

Mixture of Experts

Mixture of Experts: Overview



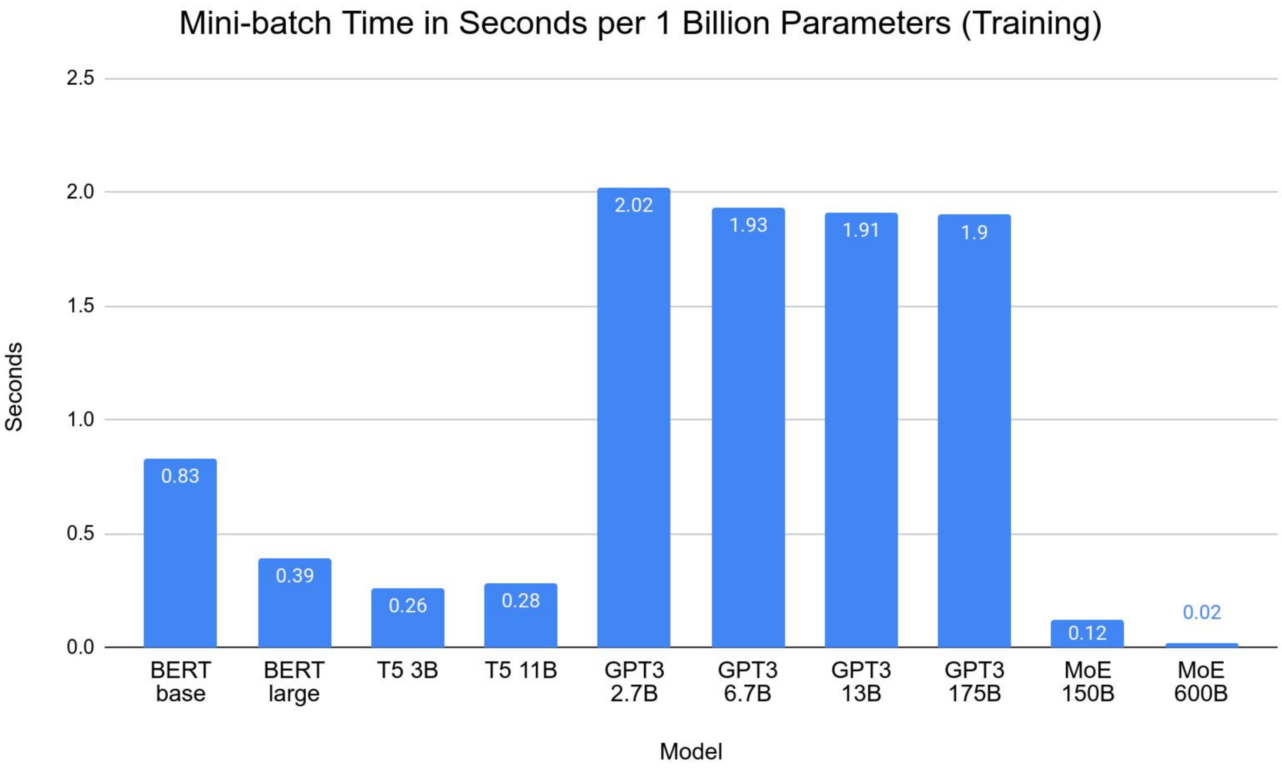
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

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

Shazeer et al., 2017: arXiv

Lepikhin et al., 2020: arXiv

Transformers Mini-batch Time



Mixture of Experts: Balancing and Specialization v1

Version 1 (Shazeer et al., 2017):

Initialize W_g and W_noise with zeros, so outputs are driven by standard normal noise. This guarantees balancing across experts at the start of training.

The noise also helps to decrease early advantage of previously picked experts.

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

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

Mixture of Experts: Balancing and Specialization v1

Version 1 (Shazeer et al., 2017):

An additional balancing loss assigns high loss to experts which have very high probability. This prevents failure cases where an expert is always picked with 100% probability.

$$Importance(X) = \sum_{x \in X} G(x)$$

$$L_{importance}(X) = w_{importance} \cdot CV(Importance(X))^2$$

Coefficient of variation: CV(X) = std(X)/mean(X)

Mixture of Experts: Balancing and Specialization v1

Version 1 (Shazeer et al., 2017)

Importance loss can be satisfied by picking a subset of experts. To prevent this degeneration we want to pick *all* experts with roughly the same probability over time.

If we view the softplus term as something analogous to a standard deviation and the mean softmax as the expected value, we can express an approximate probability for this with a CDF of the normal distribution.

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$P(x,i) = \Phi\Big(\frac{(x \cdot W_g)_i - kth\_excluding(H(x), k, i)}{Softplus((x \cdot W_{noise})_i)}\Big)$$

$$Load(X)_i = \sum_{x \in X} P(x,i)$$

$$L_{load}(X) = w_{load} \cdot CV(Load(X))^2$$

Mixture of Experts: Balancing and Specialization v2

Version 2 (Lepikhin et al., 2020):

No noise. Initialize layers normally. Keep track of c_e, how many times each expert was used for the sequence S. With the mean gate probability of m_e we can now define a balancing auxiliary loss:

$$\ell_{aux} = m_e(c_e/S)$$

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

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

Mixture of Experts: Balancing and Specialization v2

Version 2 (Lepikhin et al., 2020):

- 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: Balancing and Specialization

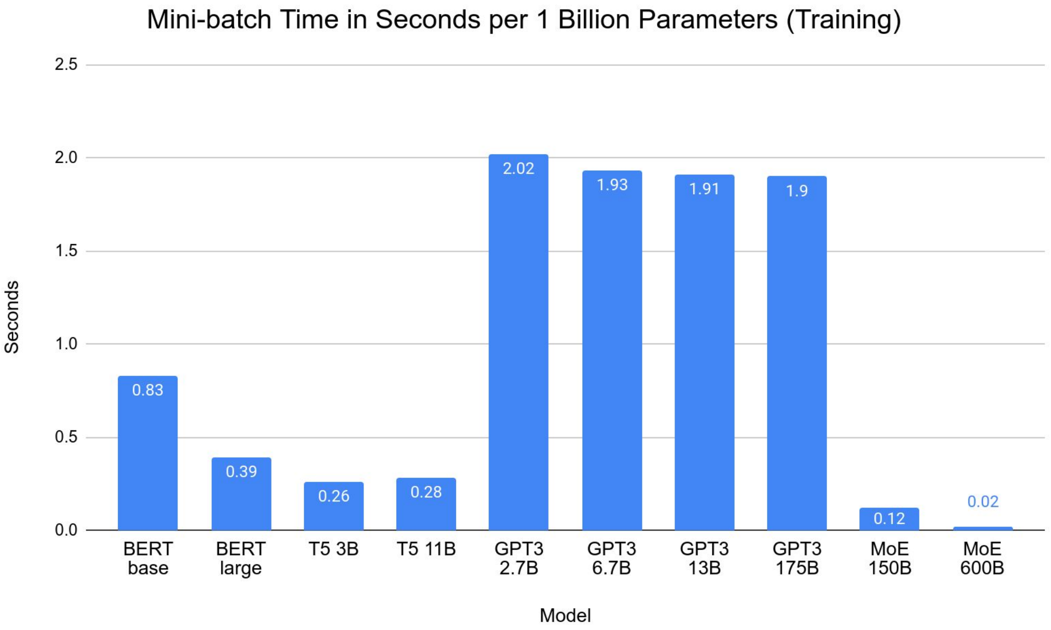
Many cases of expert degeneration:

1. **Overbalancing:** All experts are approximately equally used. However, gate probability approaches 1/#Experts. No expert is better than another expert.

2. **Underbalancing:** The same top-k experts are used for every token. This leads to two strong experts, but all other experts do not learn anything and are “wasted capacity”.

3. **Sequence-level degeneration:** Model balances experts by using each expert for a particular sequence index. For example, for indices 0, 1, 2, 3 always experts E3, E1, E2, E0. This leads to sequence experts, but not content experts.

Mixture of Experts: Benefits



- 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)

Shazeer et al., 2017: arXiv

Lepikhin et al., 2020: arXiv

07

Closing Notes

Why we should strive for efficiency

Our field has seen a dramatic increase in scale in the past 2 years.

Striving for efficiency means caring about:

1) Costs
2) Accessibility
3) Production needs
4) The sustainability of this growth

Closing Notes

In this tutorial, we covered a wide range of ideas, applications and practical considerations that helps us build more efficient systems, including:

1) Core efficiency techniques
2) Efficiency improvements to attention mechanisms
3) Case studies of efficient models
4) Practical considerations for scaling models

We hope you enjoyed it and learned something new!

Thank you!

Slides available at: bit.ly/2SmhKY7

Google Research

UNIVERSITY of WASHINGTON

UWNLP