# DSC250: Advanced Data Mining

## Language Models
## Text Embedding

**Zhiting Hu**

Lecture 10, October 30, 2023

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

# Last Lecture

- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention
  - Transformer
  - BERT

# Recap: RNNs in Various Forms

One to One

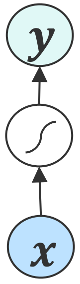



Image
classification

One to Many

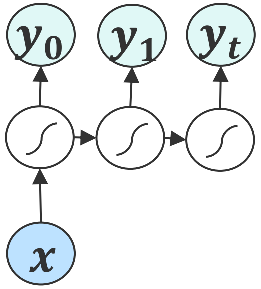
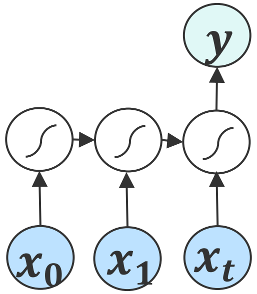


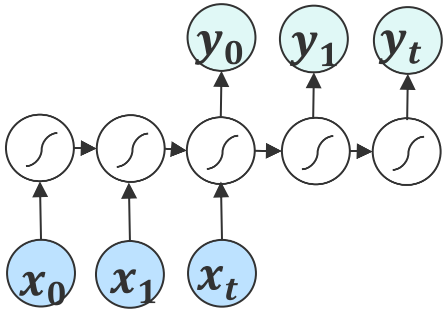

Image
captioning

Many to One




Sentence sentiment analysis /
Video recognition

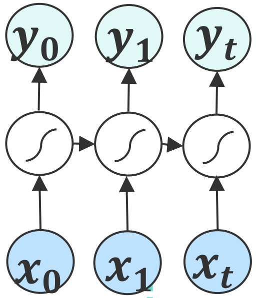
Many to Many




Machine Translation

*(Sequence-to-sequence)*

Many to Many

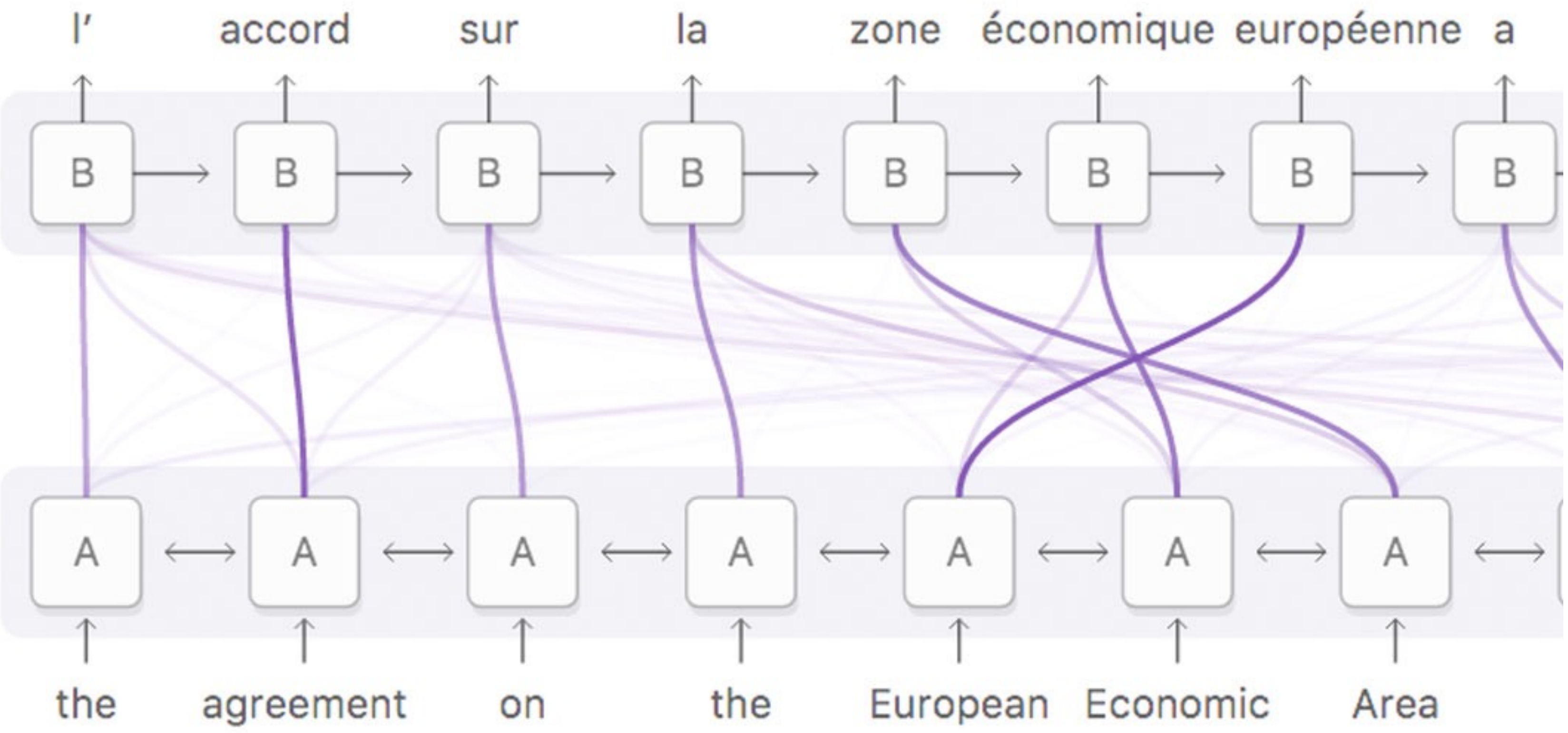



**Named Entity Recognition**

*(Sequence tagging)*

# Recap: Attention

- Chooses which features to pay attention to




Machine Translation

Figure courtesy: Olah & Carter, 2016

# Recap: Attention Variants

- Popular attention mechanisms with different alignment score functions

Alignment score = f(Query, Keys)

- Query: decoder state $s_t$
- Key: all encoder states $h_i$
- Value: all encoder states $h_i$

| Name | Alignment score function | Citation |
|---|---|---|
| Content-base attention | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$ | Graves2014 |
| Additive(*) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$ | Bahdanau2015 |
| Location-Base | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$<br>Note: This simplifies the softmax alignment to only depend on the target position. | Luong2015 |
| General | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$<br>where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer. | Luong2015 |
| Dot-Product | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{s}_t^\top \boldsymbol{h}_i$ | Luong2015 |
| Scaled Dot-Product(^) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\mathbf{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$<br>Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. | Vaswani2017 |

Courtesy: Lilian Weng

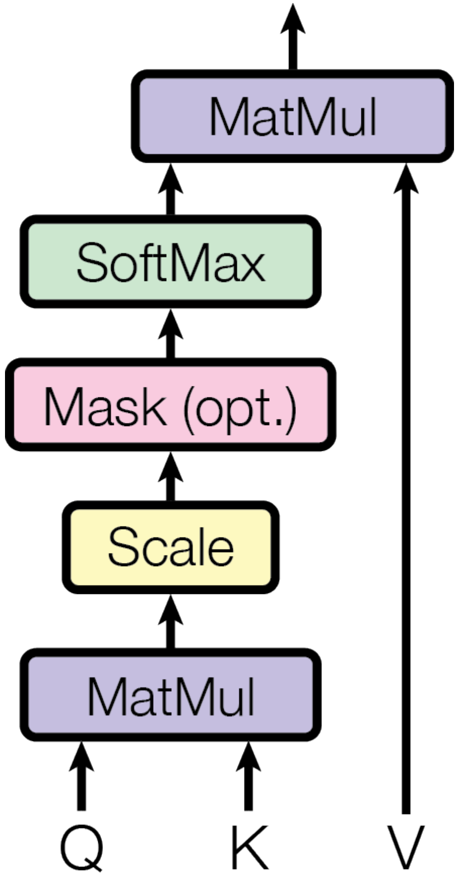
# Outline

- Recurrent Networks (RNNs)
  - Long-range dependency, vanishing gradients
  - LSTM
  - RNNs in different forms
- Attention Mechanisms
  - (Query, Key, Value)
  - Attention on Text and Images
- Transformers: Multi-head Attention

# Transformers – Multi-head (Self-)Attention

- State-of-the-art Results by Transformers
  - [Vaswani et al., 2017] Attention Is All You Need
    - Machine Translation
  - [Devlin et al., 2018] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
    - Pre-trained Text Representation
  - [Radford et al., 2019] Language Models are Unsupervised Multitask Learners
    - Language Models

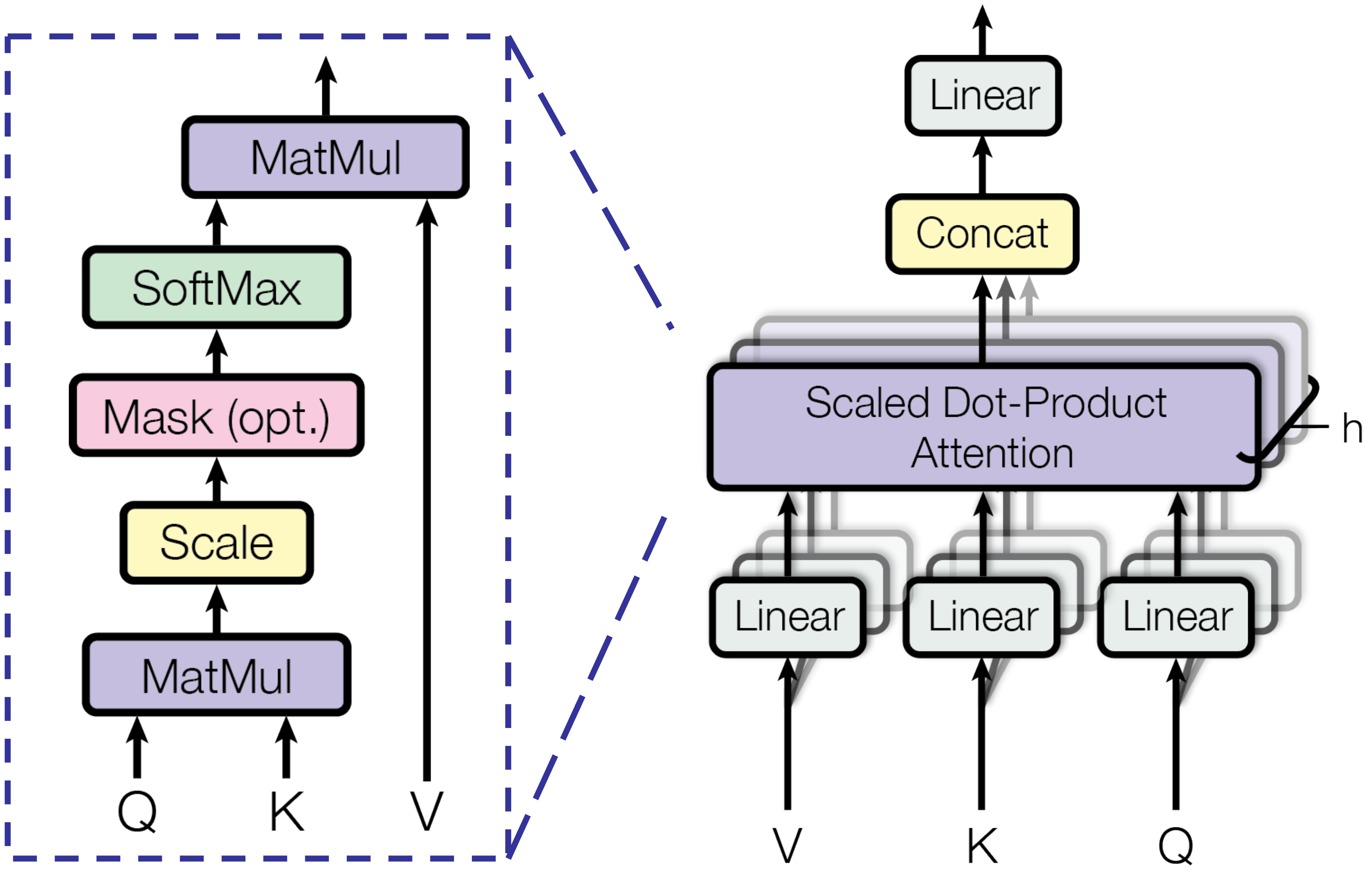
# Multi-head Attention




Scaled Dot-Product Attention

Image source: Vaswani, et al., 2017

# Multi-head Attention

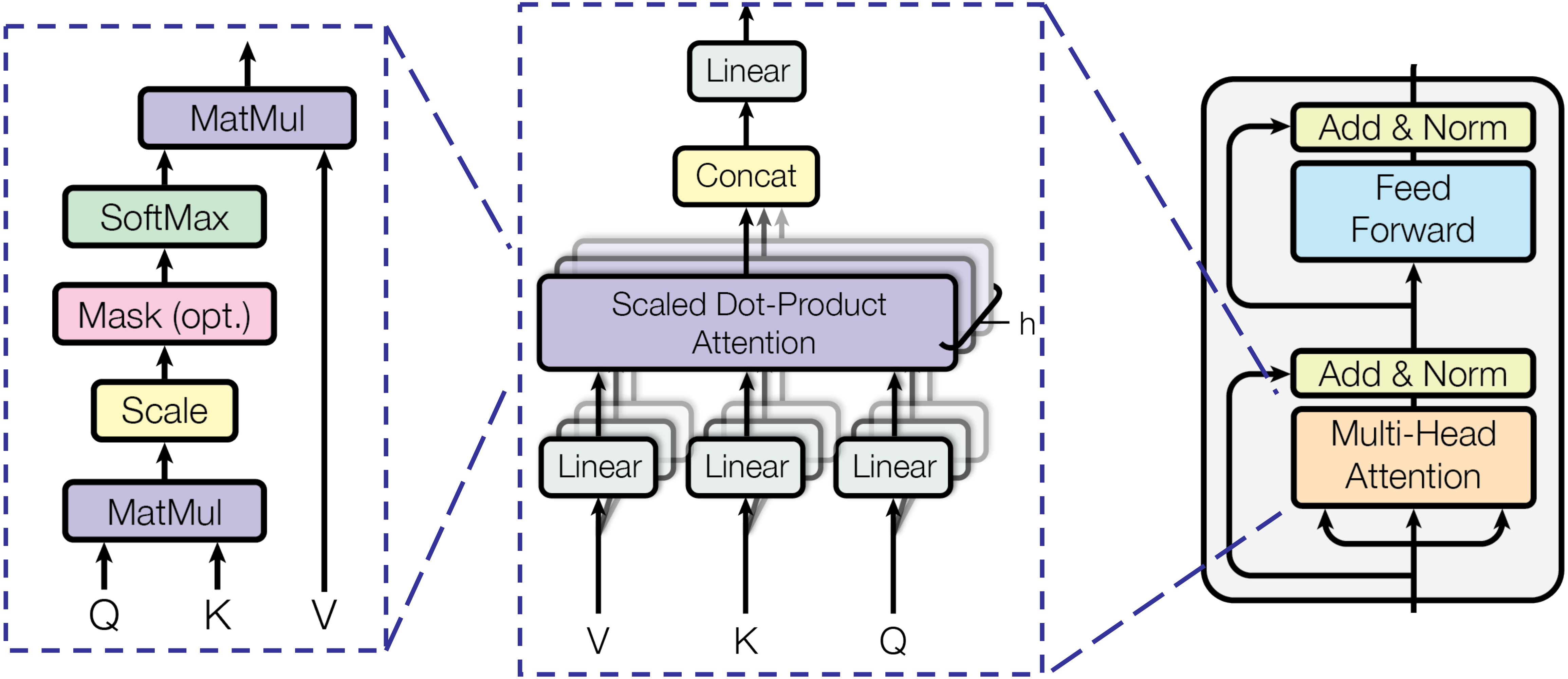



Scaled Dot-Product Attention

Multi-head Attention

Image source: [Vaswani, et al., 2017]

# Multi-head Attention



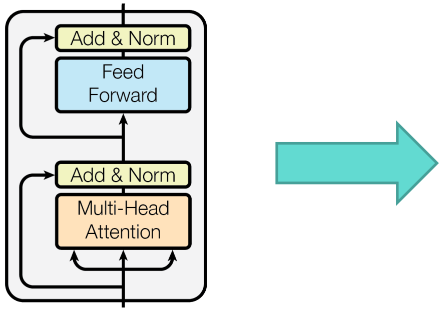

Scaled Dot-Product Attention

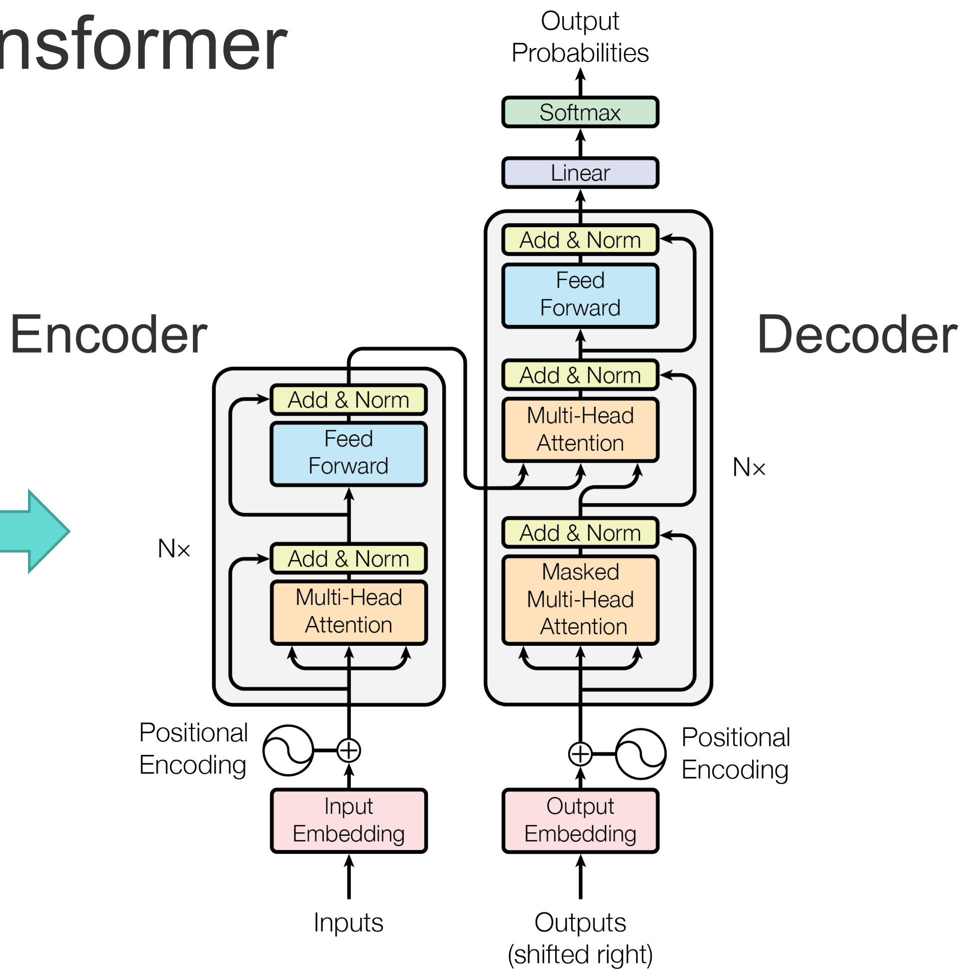
Multi-head Attention

Image source: Vaswani, et al., 2017

# Multi-head Attention in Encoders and Decoders

## Transformer



# Multi-head Attention in Encoders and Decoders

Transformer

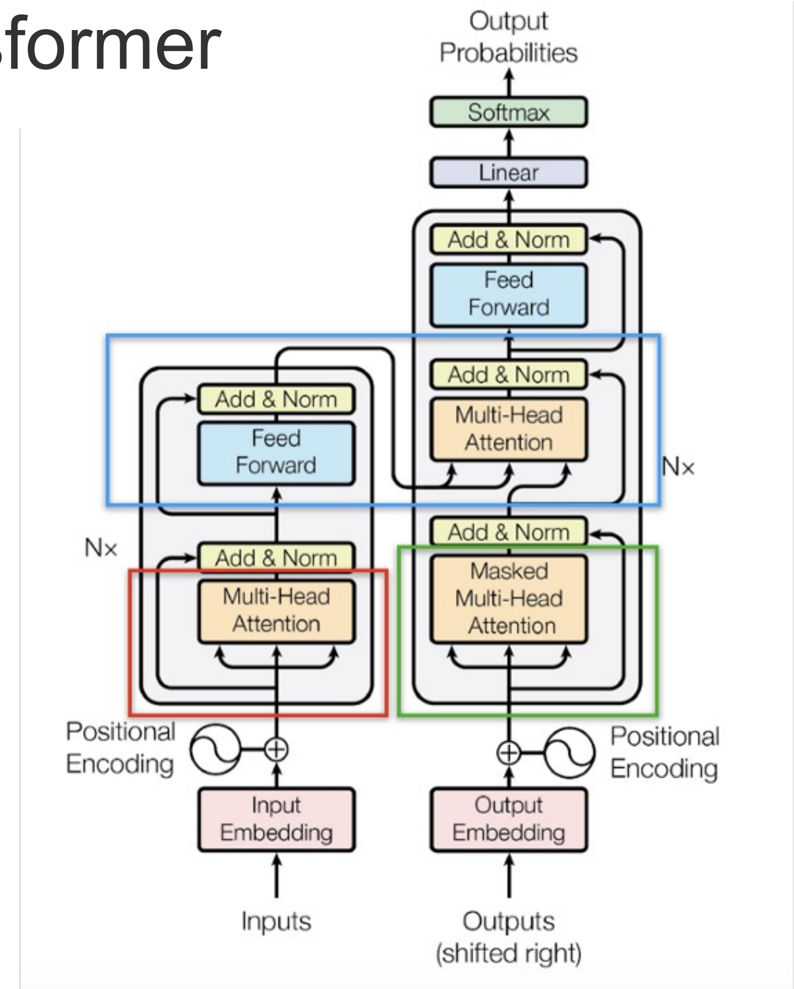



Figure 1: The Transformer - model architecture.

**encoder self attention**

1. Multi-head Attention
2. Query=Key=Value

**decoder self attention**

1. Masked Multi-head Attention
2. Query=Key=Value

**encoder-decoder attention**

1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query

Image source: Bgg

# Transformer-based LM

Transformer

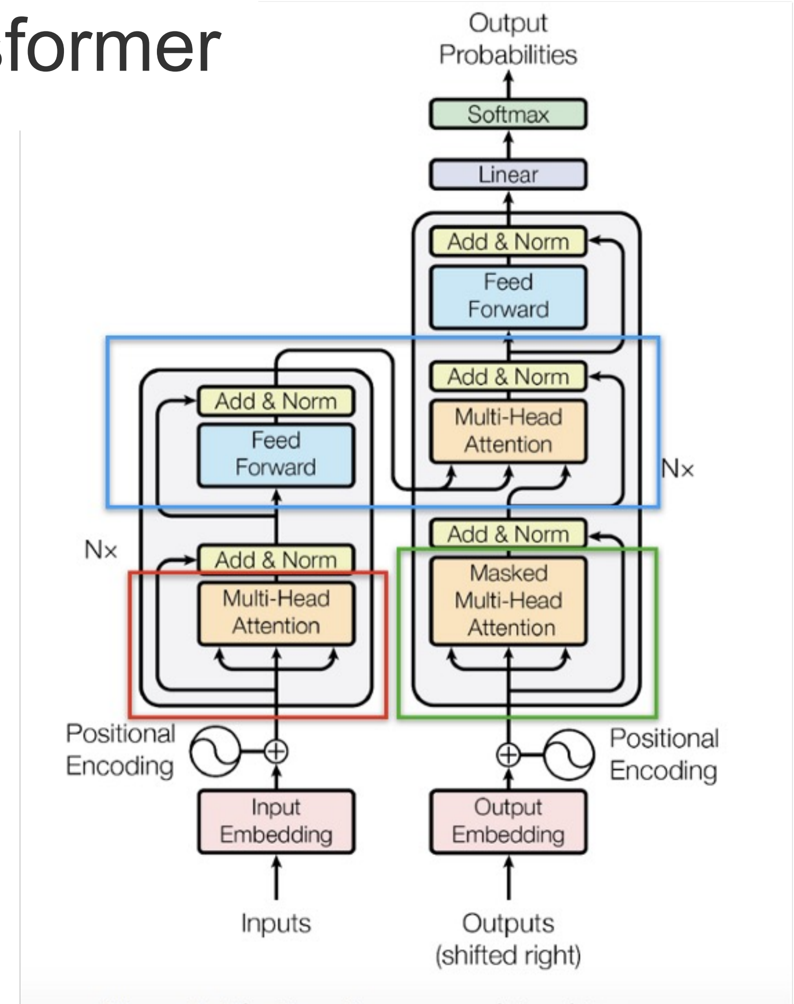



Figure 1: The Transformer - model architecture.

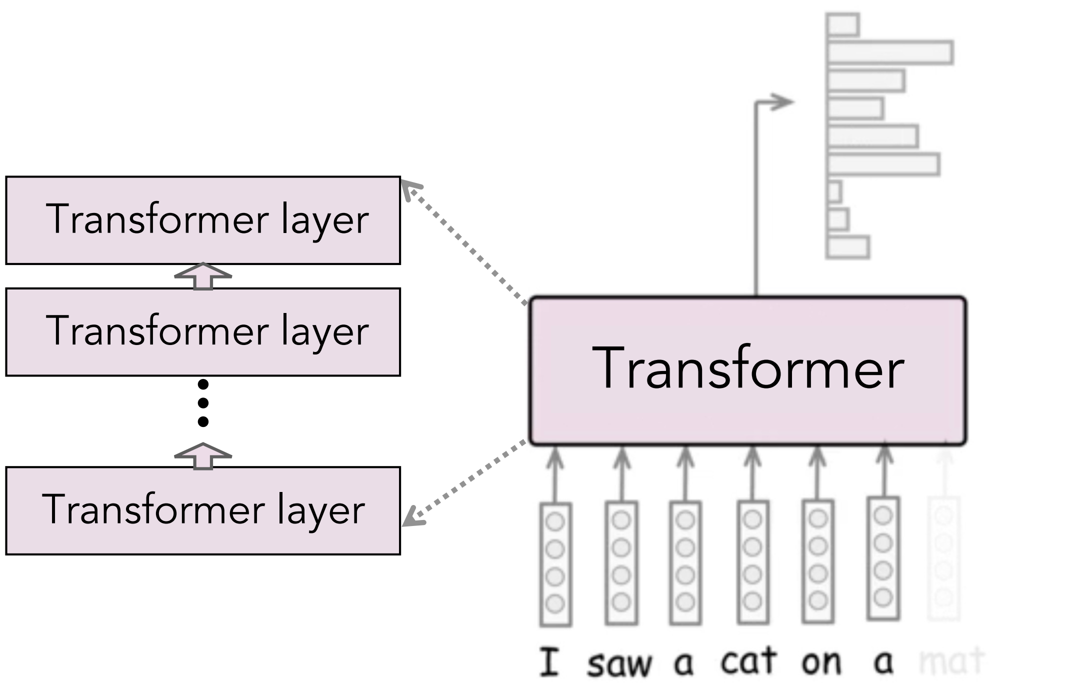



Image source: Bgg

# Neural LM Training

# Neural LMs: Next Word Prediction

$$p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

I ____

# Neural LMs: Training

- Given data example $\boldsymbol{y}^*$
- Minimizes negative log-likelihood of the data

$$\min_\theta \mathcal{L}_{\text{MLE}} = -\log p_\theta(\boldsymbol{y}^*) = -\prod_{t=1}^{T} p_\theta(y_t^* \mid \boldsymbol{y}_{1:t-1}^*)$$

# Neural LMs: GPT3

- A Transformer-based LM with 125M to 175B parameters
- Trained on massive text data

| Dataset | # Tokens (Billions) |
|---|---|
| Total | 499 |
| Common Crawl (filtered by quality) | 410 |
| WebText2 | 19 |
| Books1 | 12 |
| Books2 | 55 |
| Wikipedia | 3 |

Brown et al., 2020 "Language Models Are Few-Shot Learners"

[Table from https://lambdalabs.com/blog/demystifying-gpt-3/]

# Natural Language Processing (NLP): Before 2017

Automated understanding and generation of natural language

Core NLP tasks handled by respective machine learning models, e.g.,:

## Named Entity Recognition

PERSON CITY STATE_OR_PROVINCE DATE 1983-11-19

Adam Driver was born in San Diego , California , on November 19 , 1983 .

## Sentiment Analysis

POSITIVE

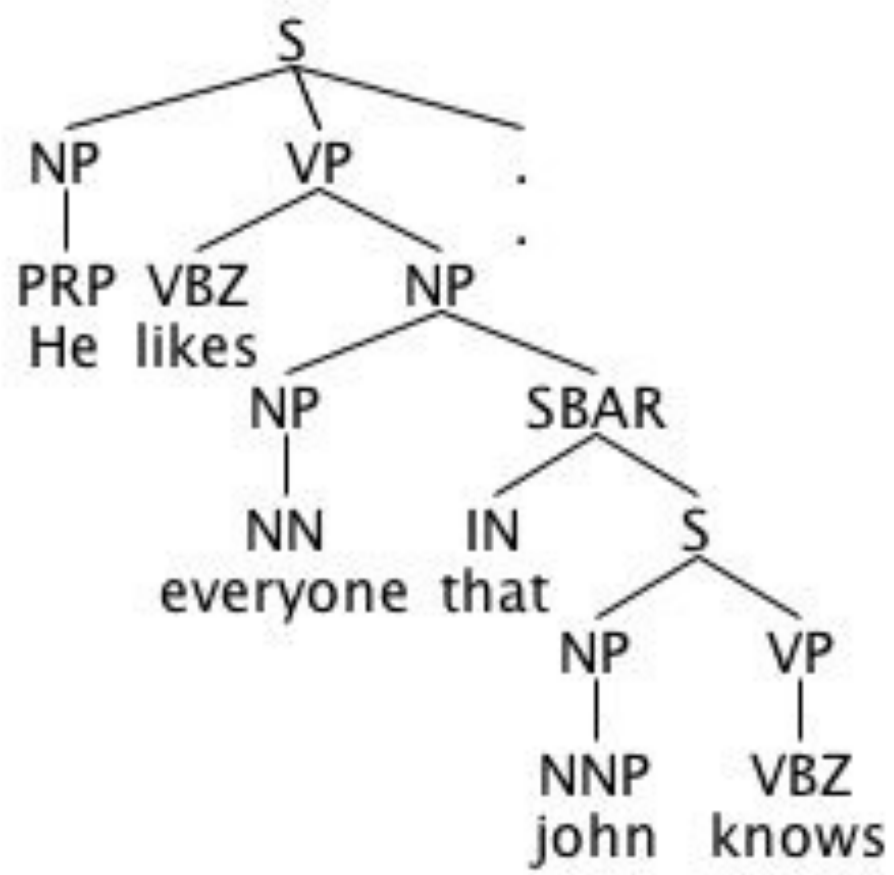
There are slow and repetitive parts , but the movie has just enough spice to keep it interesting .

(Slide courtesy: Qin, 2023)

# Natural Language Processing (NLP): Before 2017

Automated understanding and generation of natural language

Hand annotation of linguistic structures
(e.g., the Penn Treebank, 1990s)




(Slide courtesy: Qin, 2023)

# NLP breakthrough with large language models, since 2017

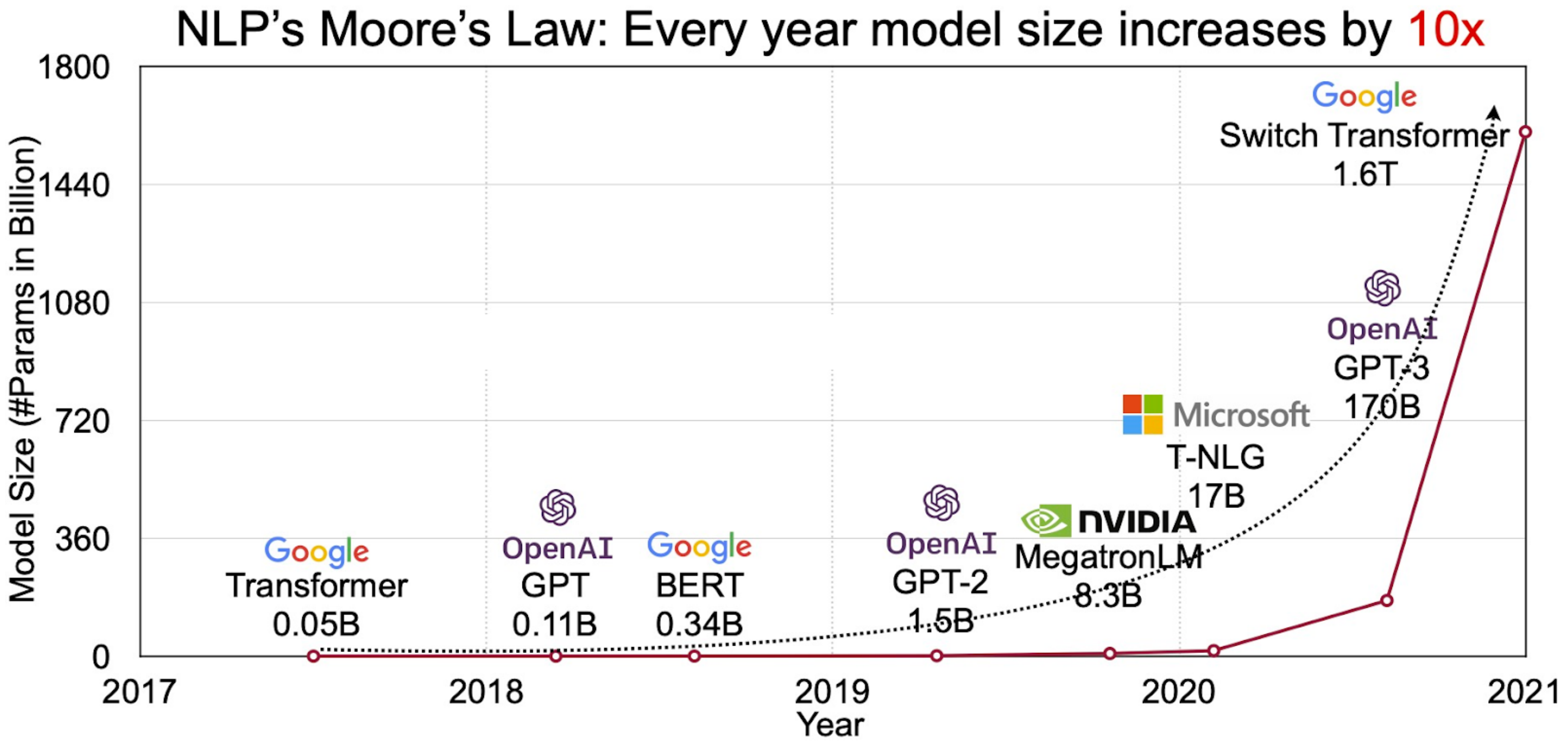
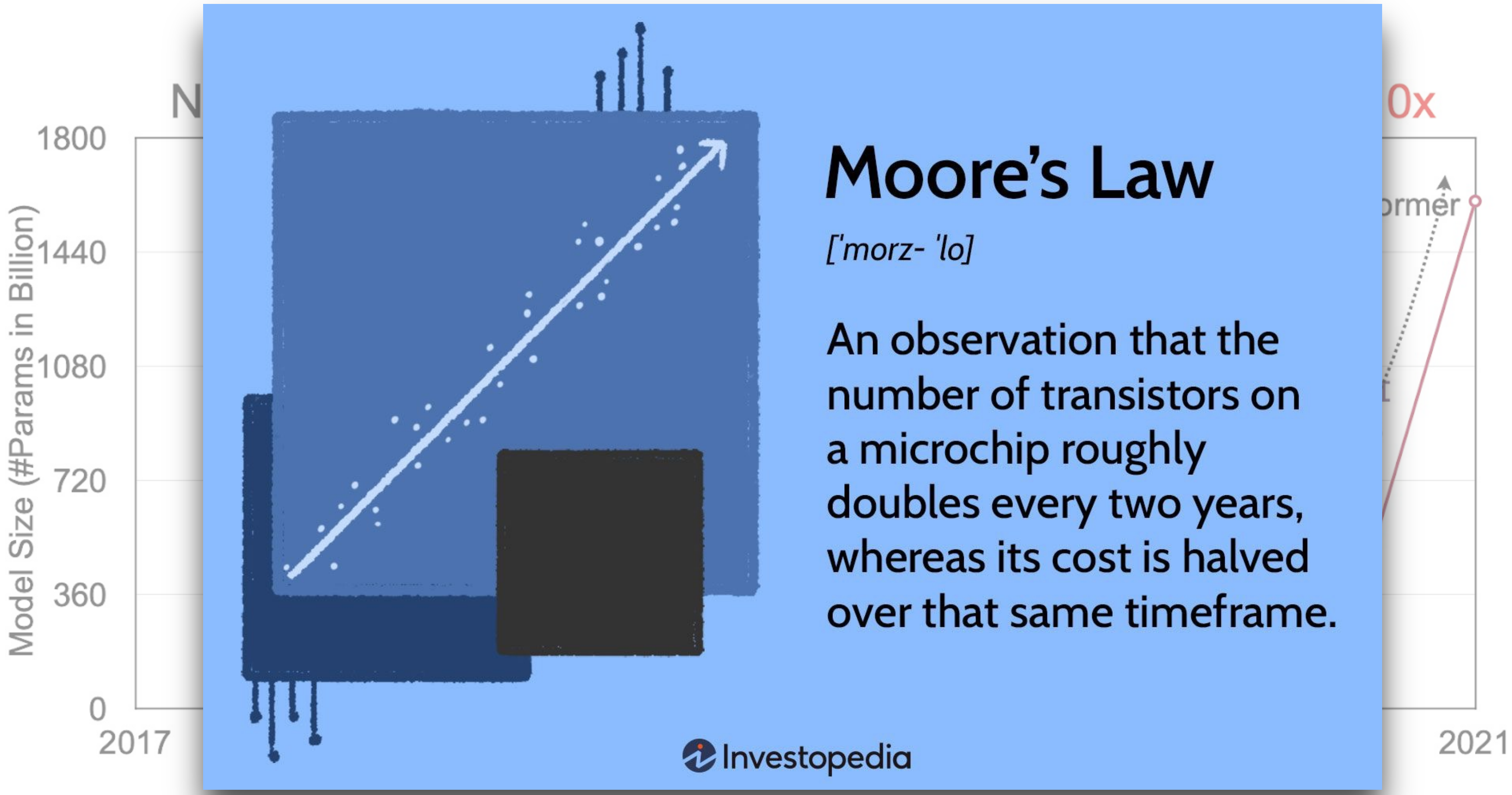



Figure credit: https://indiaai.gov.in/article/the-future-of-large-language-models-llms-strategy-opportunities-and-challenges

# NLP breakthrough with large language models, since 2017




microchip industry

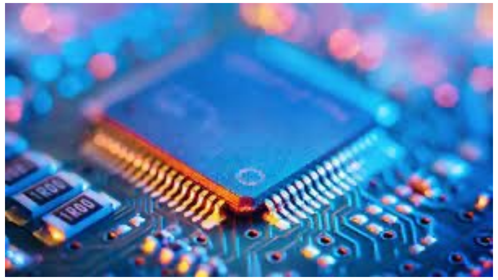


Figure credit: Investopedia

# NLP breakthrough with large language models, since 2017

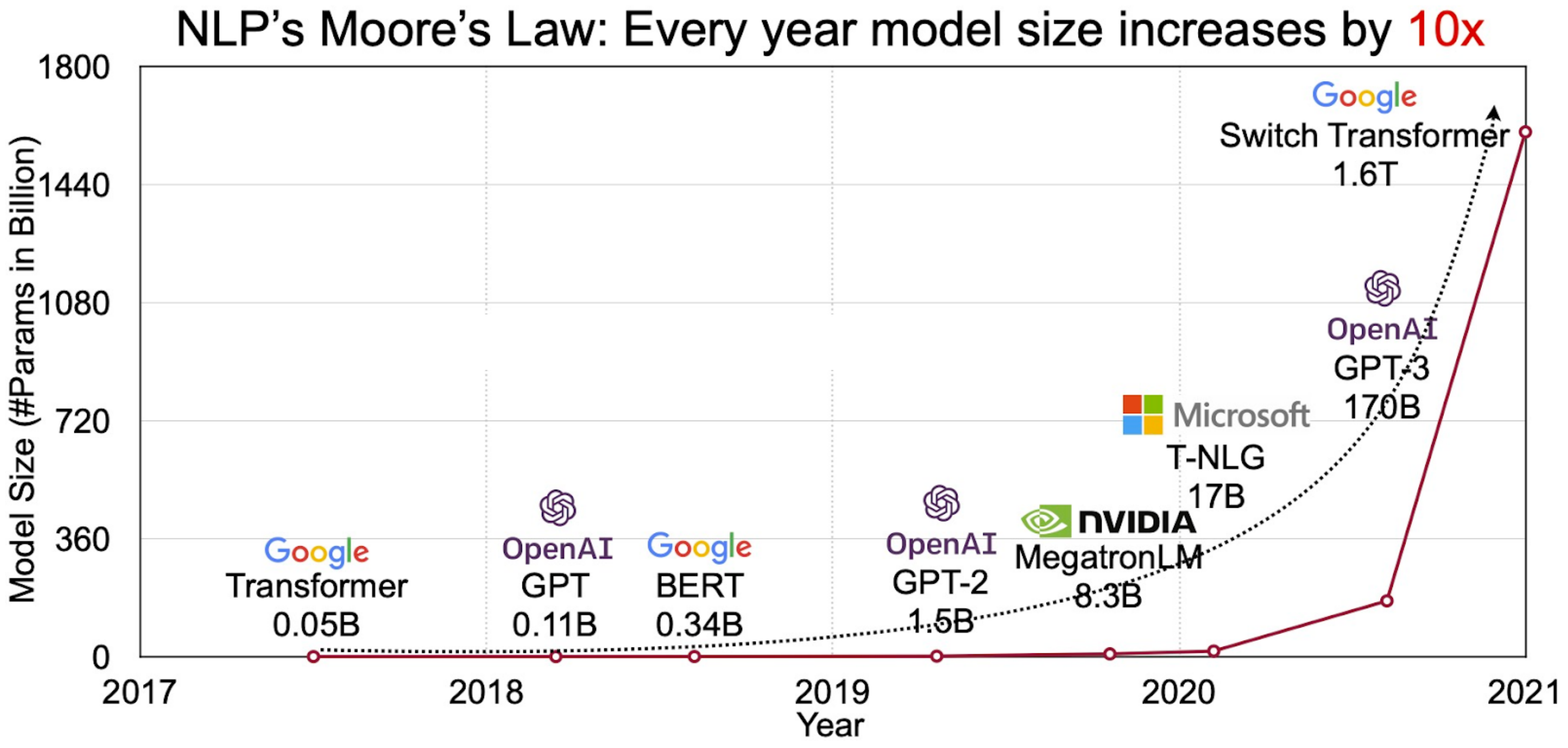
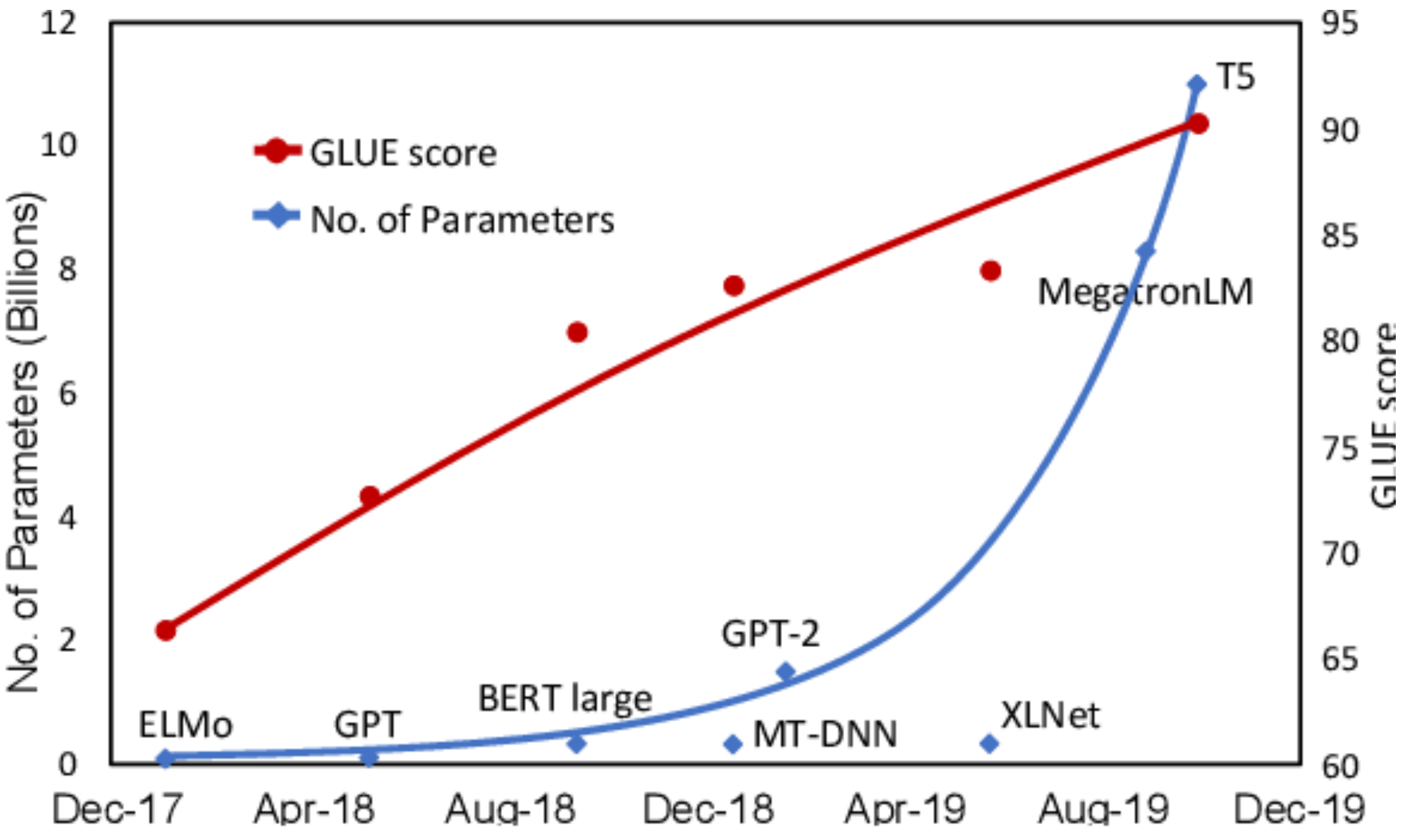



Figure credit: https://indiaai.gov.in/article/the-future-of-large-language-models-llms-strategy-opportunities-and-challenges

# NLP breakthrough with large language models, since 2017



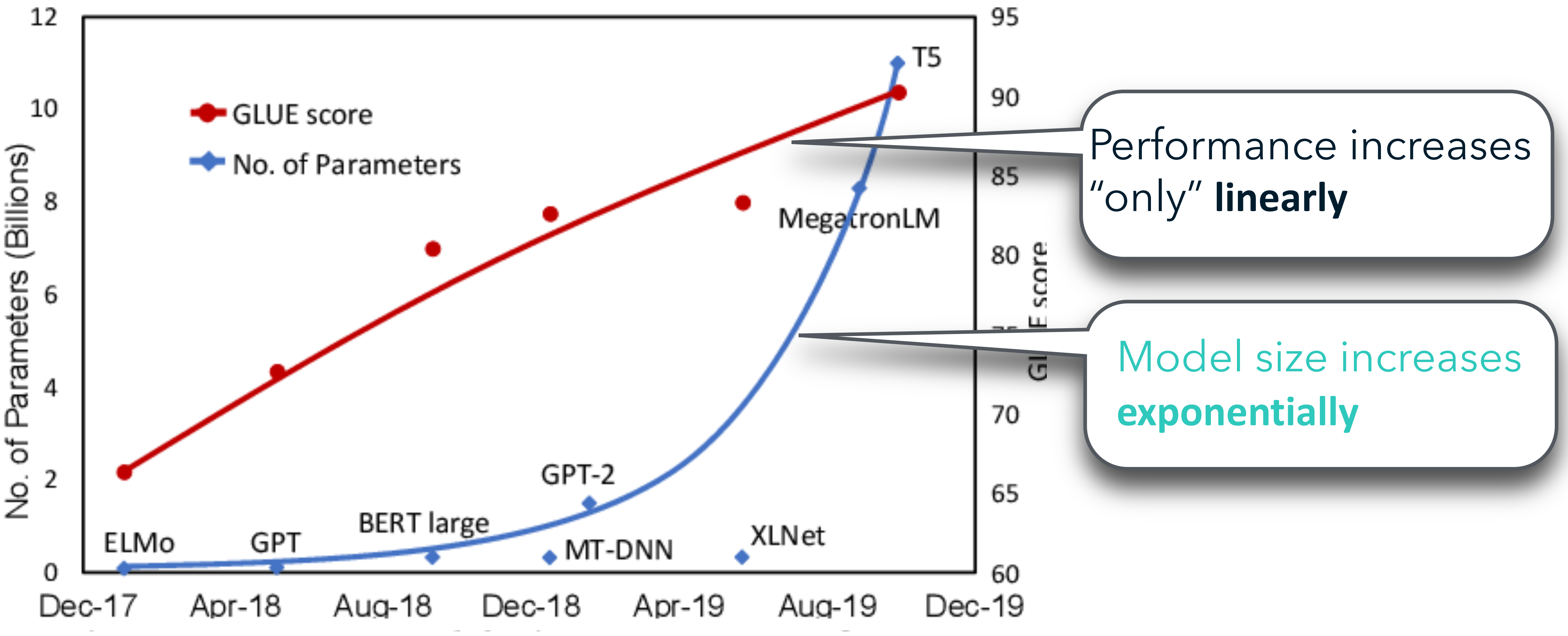

Language Model Size & GLUE Performance

(Slide courtesy: Qin, 2023)

**GLUE: General Language Understanding Evaluation**

- Sentiment analysis
- Text similarity
- Paraphrase detection
- Textual entailment
- Question answering
- Linguistic acceptability (grammaticality)
- ….

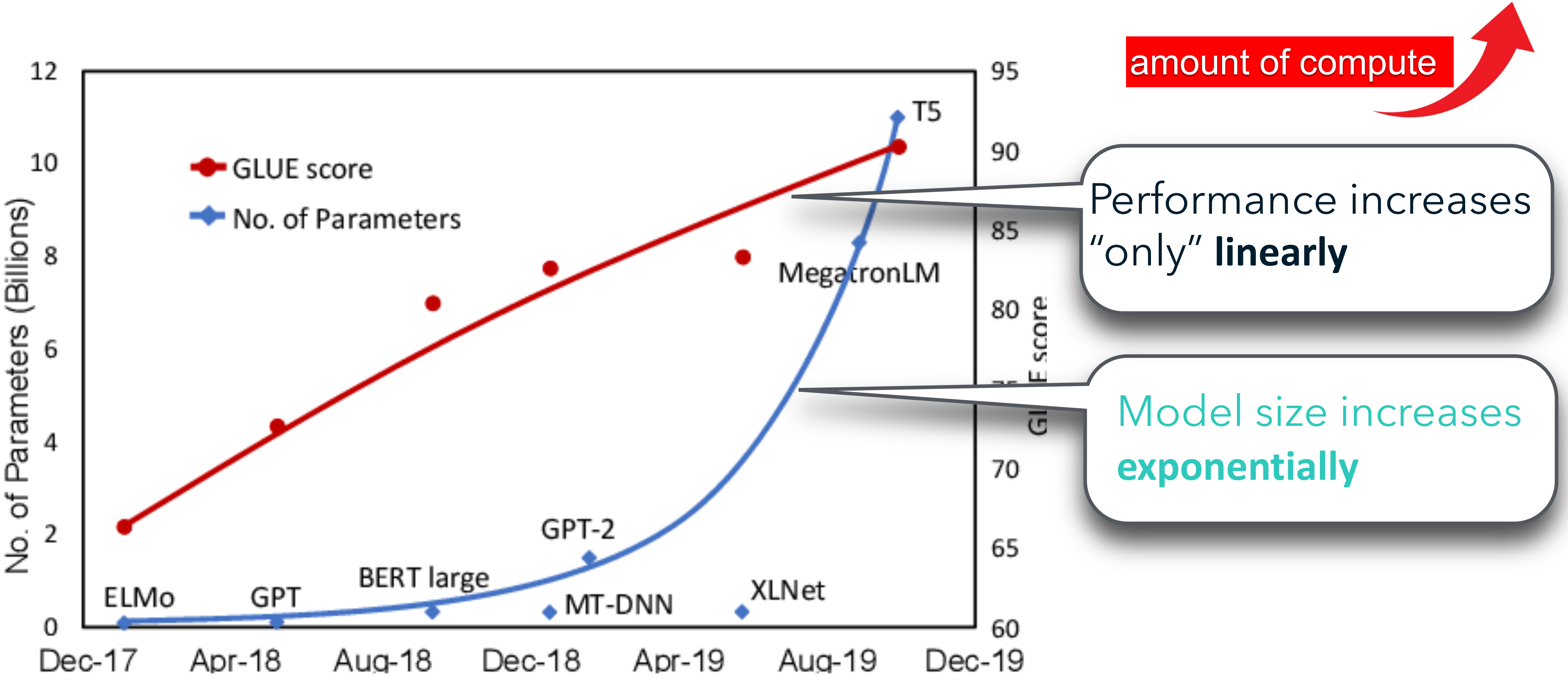
# NLP breakthrough with large language models, since 2017




Language Model Size & GLUE Performance

(Slide courtesy: Qin, 2023)

# NLP breakthrough with large language models, since 2017

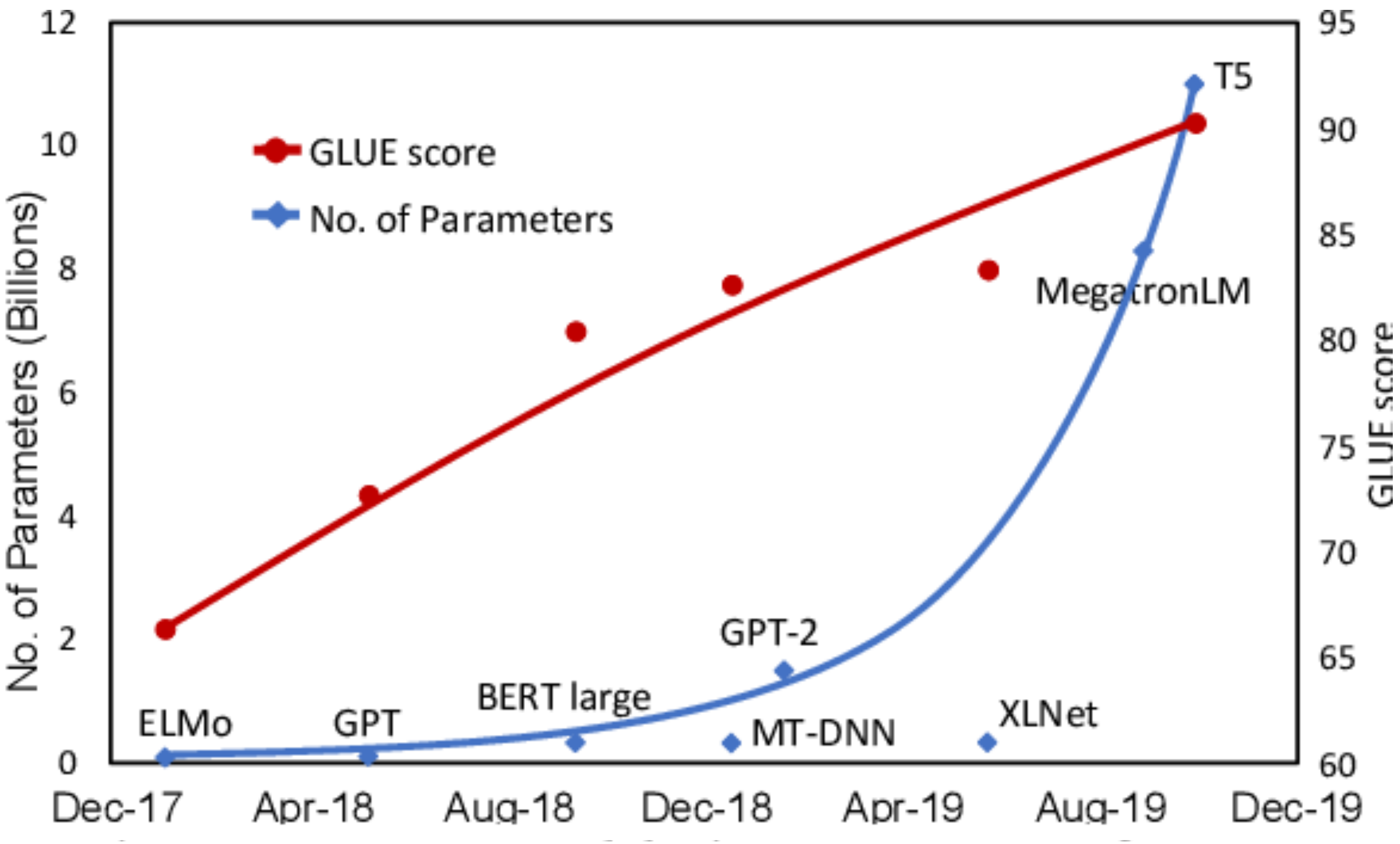



Language Model Size & GLUE Performance

(Slide courtesy: Qin, 2023)

# NLP breakthrough with large language models, since 2017

amount of compute




Language Model Size & GLUE Performance

A. Ng

AI is the new electricity

C. Manning

Electricity is the new AI?

(Slide courtesy: Qin, 2023)

# Text Embedding

# Word Embedding

- A pre-trained **matrix**, each row is an embedding vector of a word

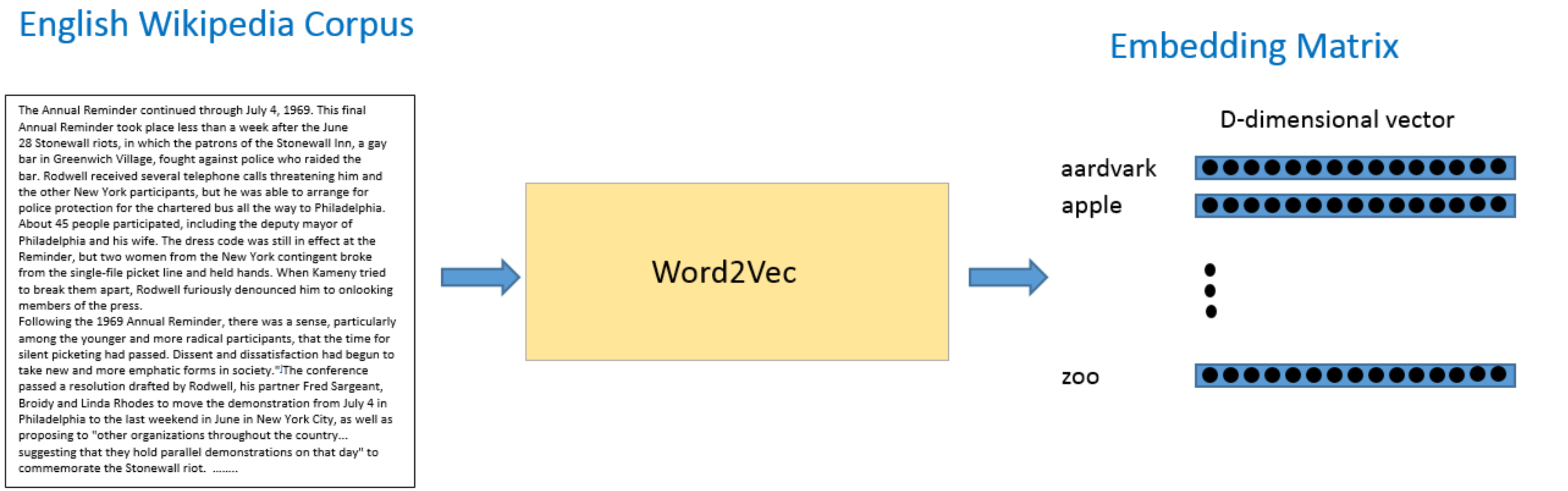
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **fox** | -0.348680 | -0.077720 | 0.177750 | -0.094953 | -0.452890 | 0.237790 | 0.209440 | 0.037886 | 0.035064 | 0.899010 | ... |
| **ham** | -0.773320 | -0.282540 | 0.580760 | 0.841480 | 0.258540 | 0.585210 | -0.021890 | -0.463680 | 0.139070 | 0.658720 | ... |
| **brown** | -0.374120 | -0.076264 | 0.109260 | 0.186620 | 0.029943 | 0.182700 | -0.631980 | 0.133060 | -0.128980 | 0.603430 | ... |
| **beautiful** | 0.171200 | 0.534390 | -0.348540 | -0.097234 | 0.101800 | -0.170860 | 0.295650 | -0.041816 | -0.516550 | 2.117200 | ... |
| **jumps** | -0.334840 | 0.215990 | -0.350440 | -0.260020 | 0.411070 | 0.154010 | -0.386110 | 0.206380 | 0.386700 | 1.460500 | ... |
| **eggs** | -0.417810 | -0.035192 | -0.126150 | -0.215930 | -0.669740 | 0.513250 | -0.797090 | -0.068611 | 0.634660 | 1.256300 | ... |
| **beans** | -0.423290 | -0.264500 | 0.200870 | 0.082187 | 0.066944 | 1.027600 | -0.989140 | -0.259950 | 0.145960 | 0.766450 | ... |
| **sky** | 0.312550 | -0.303080 | 0.019587 | -0.354940 | 0.100180 | -0.141530 | -0.514270 | 0.886110 | -0.530540 | 1.556600 | ... |
| **bacon** | -0.430730 | -0.016025 | 0.484620 | 0.101390 | -0.299200 | 0.761820 | -0.353130 | -0.325290 | 0.156730 | 0.873210 | ... |
| **breakfast** | 0.073378 | 0.227670 | 0.208420 | -0.456790 | -0.078219 | 0.601960 | -0.024494 | -0.467980 | 0.054627 | 2.283700 | ... |
| **toast** | 0.130740 | -0.193730 | 0.253270 | 0.090102 | -0.272580 | -0.030571 | 0.096945 | -0.115060 | 0.484000 | 0.848380 | ... |
| **today** | -0.156570 | 0.594890 | -0.031445 | -0.077586 | 0.278630 | -0.509210 | -0.066350 | -0.081890 | -0.047986 | 2.803600 | ... |
| **blue** | 0.129450 | 0.036518 | 0.032298 | -0.060034 | 0.399840 | -0.103020 | -0.507880 | 0.076630 | -0.422920 | 0.815730 | ... |
| **green** | -0.072368 | 0.233200 | 0.137260 | -0.156630 | 0.248440 | 0.349870 | -0.241700 | -0.091426 | -0.530150 | 1.341300 | ... |
| **kings** | 0.259230 | -0.854690 | 0.360010 | -0.642000 | 0.568530 | -0.321420 | 0.173250 | 0.133030 | -0.089720 | 1.528600 | ... |
| **dog** | -0.057120 | 0.052685 | 0.003026 | -0.048517 | 0.007043 | 0.041856 | -0.024704 | -0.039783 | 0.009614 | 0.308416 | ... |
| **sausages** | -0.174290 | -0.064869 | -0.046976 | 0.287420 | -0.128150 | 0.647630 | 0.056315 | -0.240440 | -0.025094 | 0.502220 | ... |
| **lazy** | -0.353320 | -0.299710 | -0.176230 | -0.321940 | -0.385640 | 0.586110 | 0.411160 | -0.418680 | 0.073093 | 1.486500 | ... |
| **love** | 0.139490 | 0.534530 | -0.252470 | -0.125650 | 0.048748 | 0.152440 | 0.199060 | -0.065970 | 0.128830 | 2.055900 | ... |
| **quick** | -0.445630 | 0.191510 | -0.249210 | 0.465900 | 0.161950 | 0.212780 | -0.046480 | 0.021170 | 0.417660 | 1.686900 | ... |

[Courtesy: Vaswani, et al., 2017]

# Word Embedding

- A pre-trained **matrix**, each row is an embedding vector of a word

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **fox** | -0.348680 | -0.077720 | 0.177750 | -0.094953 | -0.452890 | 0.237790 | 0.209440 | 0.037886 | 0.035064 | 0.899010 | ... |
| **ham** | -0.773320 | -0.282540 | 0.580760 | 0.841480 | 0.258540 | 0.585210 | -0.021890 | -0.463680 | 0.139070 | 0.658720 | ... |
| **brown** | -0.374120 | -0.076264 | 0.109260 | 0.186620 | 0.029943 | 0.182700 | -0.631980 | 0.133060 | -0.128980 | 0.603430 | ... |
| **beautiful** | 0.171200 | 0.534390 | -0.348540 | -0.097234 | 0.101800 | -0.170860 | 0.295650 | -0.041816 | -0.516550 | 2.117200 | ... |
| **jumps** | -0.334840 | 0.215990 | -0.350440 | -0.260020 | 0.411070 | 0.154010 | -0.386110 | 0.206380 | 0.386700 | 1.460500 | ... |
| **eggs** | -0.417810 | -0.035192 | -0.126150 | -0.215930 | -0.669740 | 0.513250 | -0.797090 | -0.068611 | 0.634660 | 1.256300 | ... |
| **beans** | -0.423290 | -0.264500 | 0.200870 | 0.082187 | 0.066944 | 1.027600 | -0.989140 | -0.259950 | 0.145960 | 0.766450 | ... |
| **sky** | 0.312550 | -0.303080 | 0.019587 | -0.354940 | 0.100180 | -0.141530 | -0.514270 | 0.886110 | -0.530540 | 1.556600 | ... |
| **bacon** | -0.430730 | -0.016025 | 0.484620 | 0.101390 | -0.299200 | 0.761820 | -0.353130 | -0.325290 | 0.156730 | 0.873210 | ... |
| **breakfast** | 0.073378 | 0.227670 | 0.208420 | -0.456790 | -0.078219 | 0.601960 | -0.024494 | -0.467980 | 0.054627 | 2.283700 | ... |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | -0.115060 | 0.484000 | 0.848380 | ... |
| | | | | | | | [illegible] | -0.081890 | -0.047986 | 2.803600 | ... |
| | | | | | | | [illegible] | 0.076630 | -0.422920 | 0.815730 | ... |
| | | | | | | | [illegible] | -0.091426 | -0.530150 | 1.341300 | ... |
| | | | | | | | [illegible] | 0.133030 | -0.089720 | 1.528600 | ... |
| | | | | | | | [illegible] | -0.039783 | 0.009614 | 0.308416 | ... |
| | | | | | | | [illegible] | -0.240440 | -0.025094 | 0.502220 | ... |
| | | | | | | | [illegible] | -0.418680 | 0.073093 | 1.486500 | ... |
| | | | | | | | [illegible] | -0.065970 | 0.128830 | 2.055900 | ... |
| | | | | | | | [illegible] | 0.021170 | 0.417660 | 1.686900 | ... |




[Image source: Va

# Word2vec: Skip-Gram Model

- (Mikolov et al., 2013a,b)

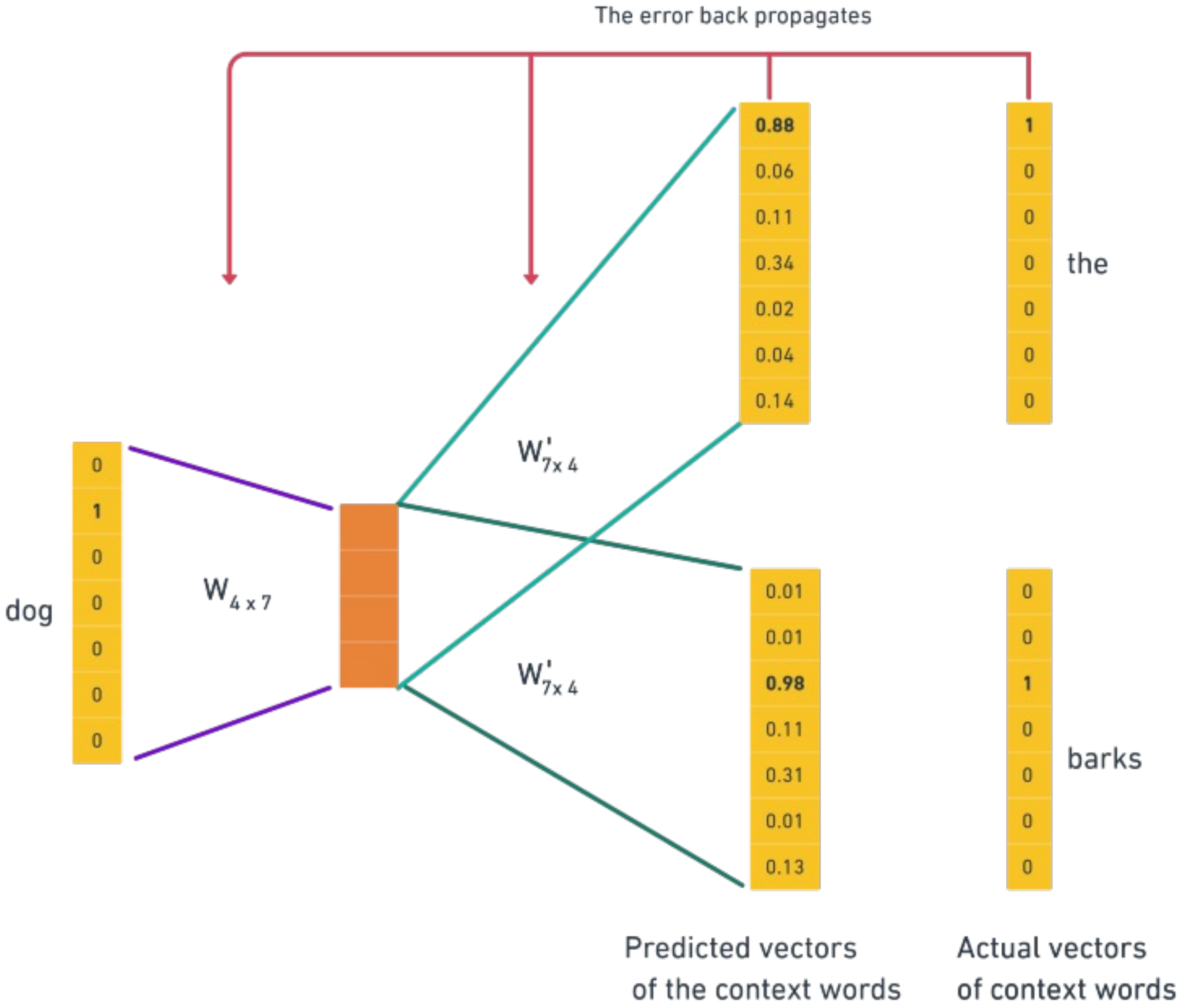
$$p(C = c \mid X = v) = \frac{1}{Z_v} \exp \mathbf{c}_c^\top \mathbf{v}_v$$

  - Two different vectors for each element of $\mathcal{V}$: one when it is "$v$" ($\mathbf{v}$) and one when it is "$c$" ($\mathbf{c}$).
  - This should remind you of a neural network; SGD on the likelihood function is the conventional approach to estimating the vectors.
  - Normalization term $Z_v$ is expensive, so approximations are required for efficiency.
  - Can expand this to be over the whole sentence or document, or otherwise choose which words "count" as context.

[Courtesy: UW CSE 447 by Noah Smith]

# Word2vec: Skip-Gram Model

“the dog barks”




[Figure courtesy: Maryam Fallah]

# Word Embedding Evaluation

Several popular methods for *intrinsic* evaluations:

- Do (cosine) similarities of pairs of words' vectors correlate with judgments of similarity by humans?
- TOEFL-like synonym tests, e.g., *rug* $\stackrel{?}{\rightarrow}$ {*sofa*, *ottoman*, *carpet*, *hallway*}
- Syntactic analogies, e.g., "*walking* is to *walked* as *eating* is to what?" Solved via:

$$\max_{v \in \mathcal{V}} \cos\left(\mathbf{v}_v, -\mathbf{v}_{\mathit{walking}} + \mathbf{v}_{\mathit{walked}} + \mathbf{v}_{\mathit{eating}}\right)$$

[Courtesy: UW CSE 447 by Noah Smith]

# Word Embedding Evaluation

Extrinsic evaluation:

1. Use large unannotated corpus to get your word vectors (sometimes called **pretraining**).
2. Use them in a text classifier (or some other NLP system). Two options:
   - Plug in word vectors as "frozen" features, and estimate the other parameters of your model.
   - Treat them as parameters of the text classifier; pretraining gives initial values, but they get updated, or "finetuned" during supervised learning.
3. Does that system's performance improve?

[Courtesy: UW CSE 447 by Noah Smith]

# Word Embedding

- **Problem**: word embeddings are applied in a context free manner

```
open a bank account        on the river bank

            [0.3, 0.2, -0.8, …]
```

Courtesy: Devlin 2019

# Word Embedding

- **Problem**: word embeddings are applied in a context free manner

```
open a bank account          on the river bank

              [0.3, 0.2, -0.8, …]
```

- **Solution**: Train contextual representations on text corpus

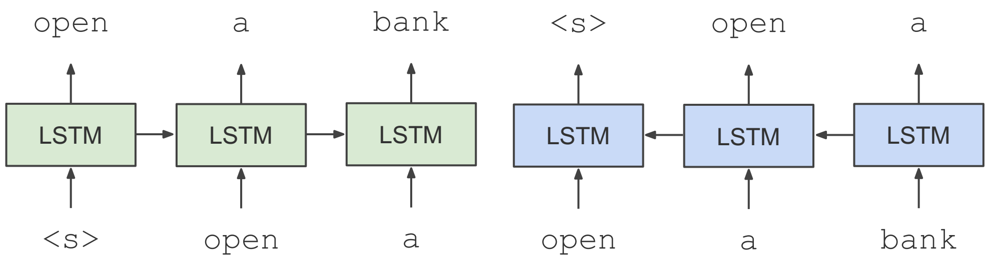
```
[0.9, -0.2, 1.6, …]          [-1.9, -0.4, 0.1, …]

open a bank account          on the river bank
```

Courtesy: Devlin 2019

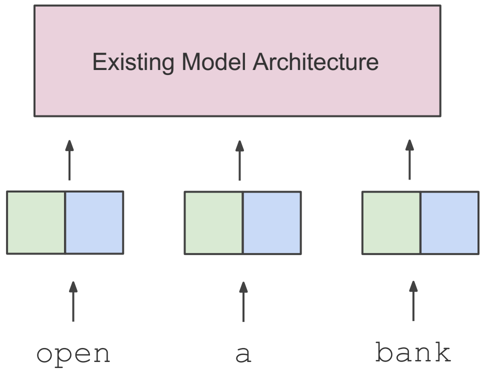
# Contextual Representations

- *ELMo: Deep Contextual Word Embeddings*, AI2 & University of Washington, 2017

**Train Separate Left-to-Right and Right-to-Left LMs**




**Apply as "Pre-trained Embeddings"**



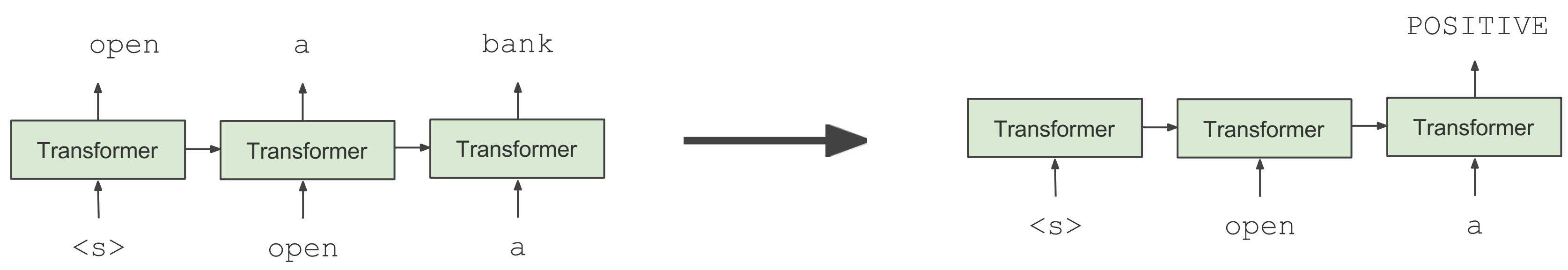

Courtesy: Devlin 2019

# Contextual Representations

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018

**Train Deep (12-layer) Transformer LM**

**Fine-tune on Classification Task**




Courtesy: Devlin 2019

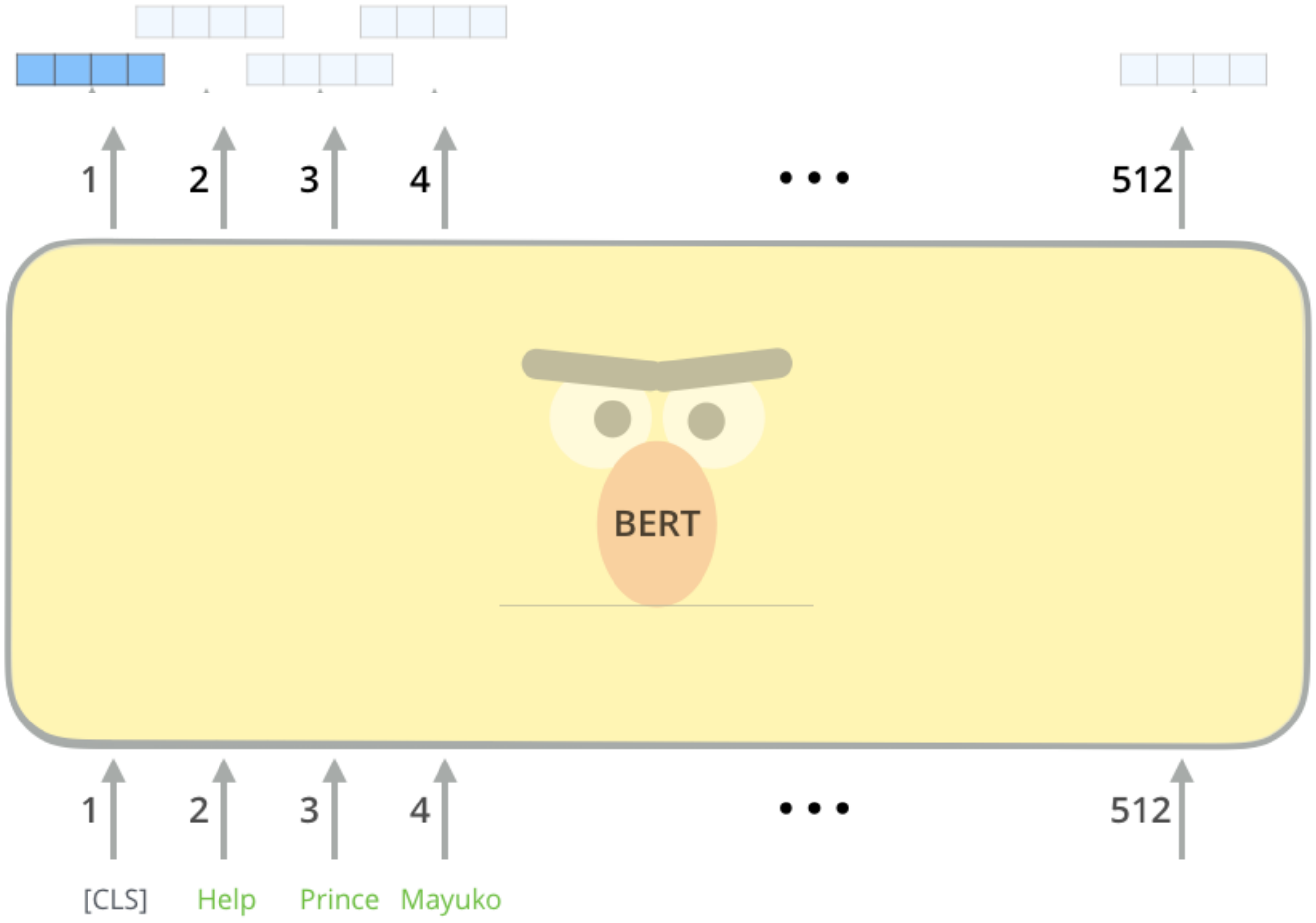
# Problem with Previous Methods

- **Problem**: Language models only use left context *or* right context, but language understanding is bidirectional.

courtesy: Devlin 2019

# BERT

- BERT: A bidirectional model to extract contextual word embedding



# BERT: Pre-training Procedure

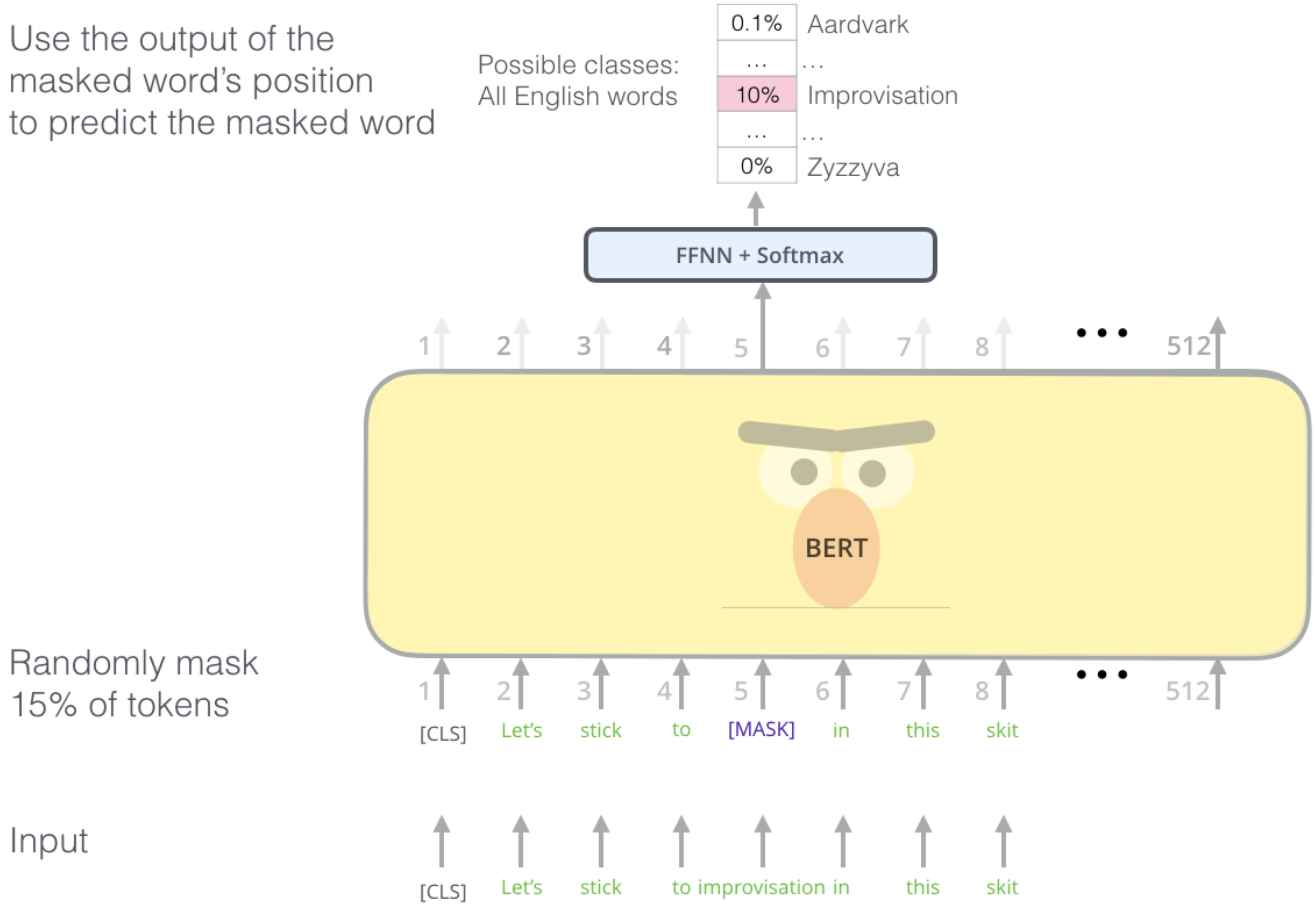
- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)

# BERT: Pre-training Procedure

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - **masked language model** (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context

# BERT: Pre-training Procedure

- Masked LM



# BERT: Pre-training Procedure

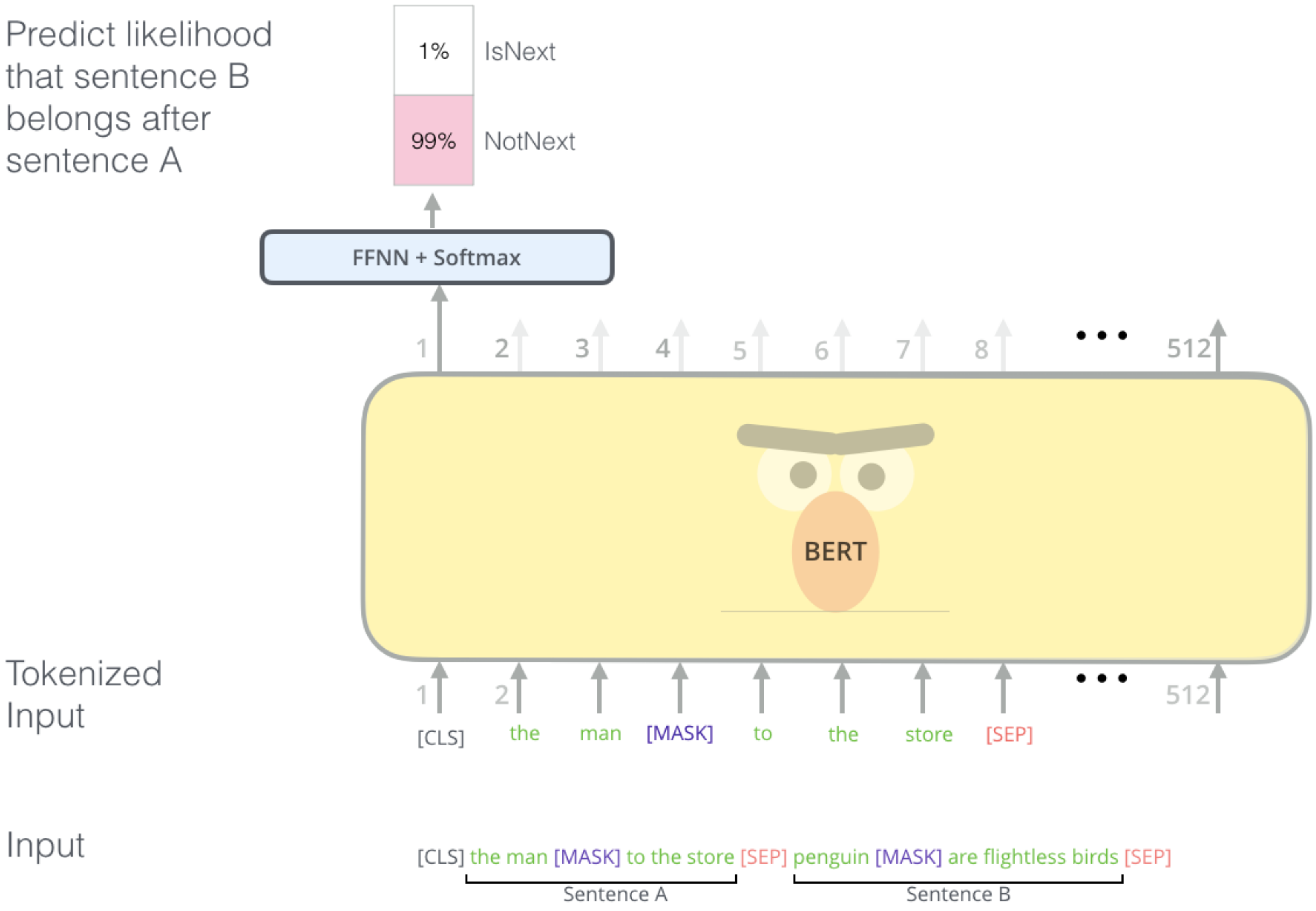
- Masked LM
- 15% masking:
  - Too little masking: Too expensive to train (few supervision signals per example)
  - Too much masking: Not enough context
- Problem: Mask token never seen at fine-tuning
- Solution: don't replace with `[MASK]` 100% of the time. Instead:
- 80% of the time, replace with `[MASK]`
  - `went to the store → went to the [MASK]`
- 10% of the time, replace random word
  - `went to the store → went to the running`
- 10% of the time, keep same
  - `went to the store → went to the store`

# BERT: Pre-training Procedure

- Dataset:
  - Wikipedia (2.5B words) + a collection of free ebooks (800M words)
- Training procedure
  - **masked language model** (masked LM)
    - Masks some percent of words from the input and has to reconstruct those words from context
  - **Two-sentence task**
    - To understand relationships between sentences
    - Concatenate two sentences A and B and predict whether B actually comes after A in the original text

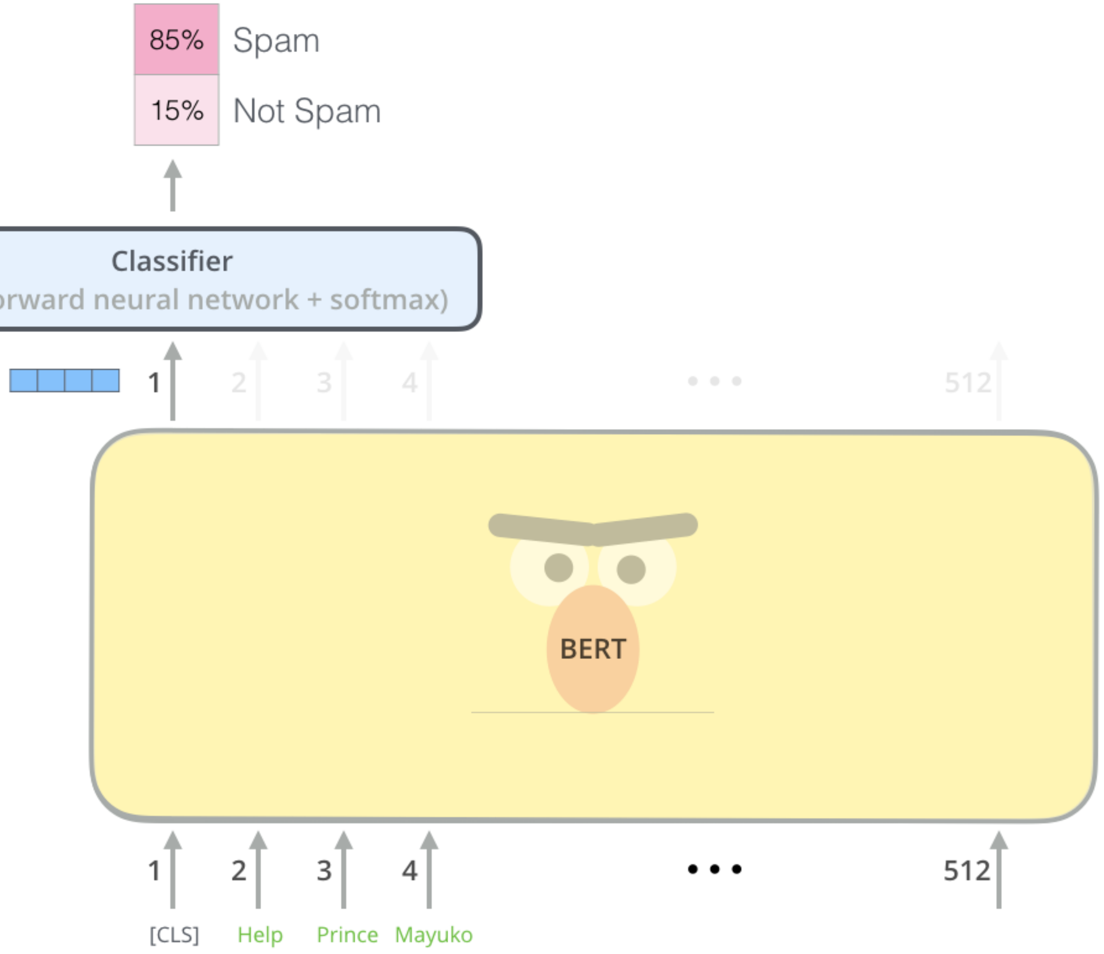
# BERT: Pre-training Procedure

- Two sentence task

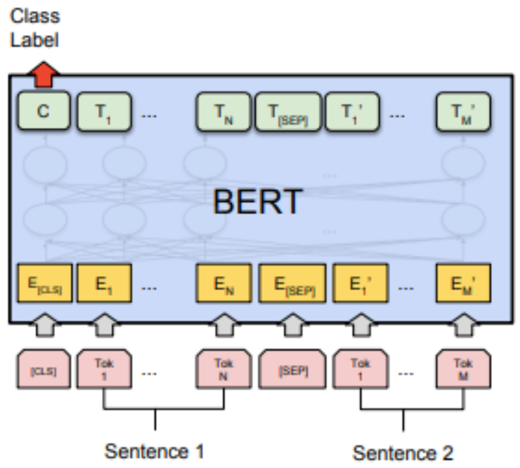


# BERT: Downstream Fine-tuning

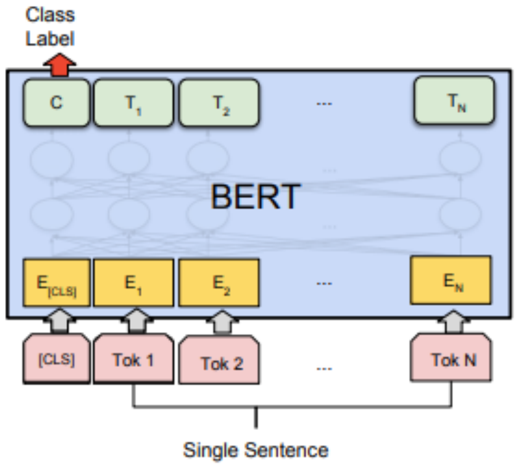
- Use BERT for sentence classification



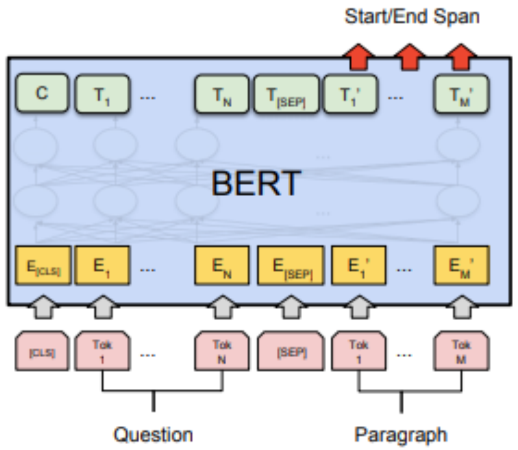
# BERT: Downstream Fine-tuning



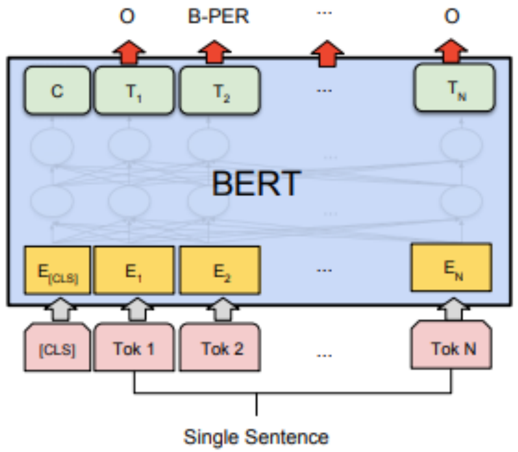

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG




(b) Single Sentence Classification Tasks: SST-2, CoLA




(c) Question Answering Tasks: SQuAD v1.1




(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

# BERT Results

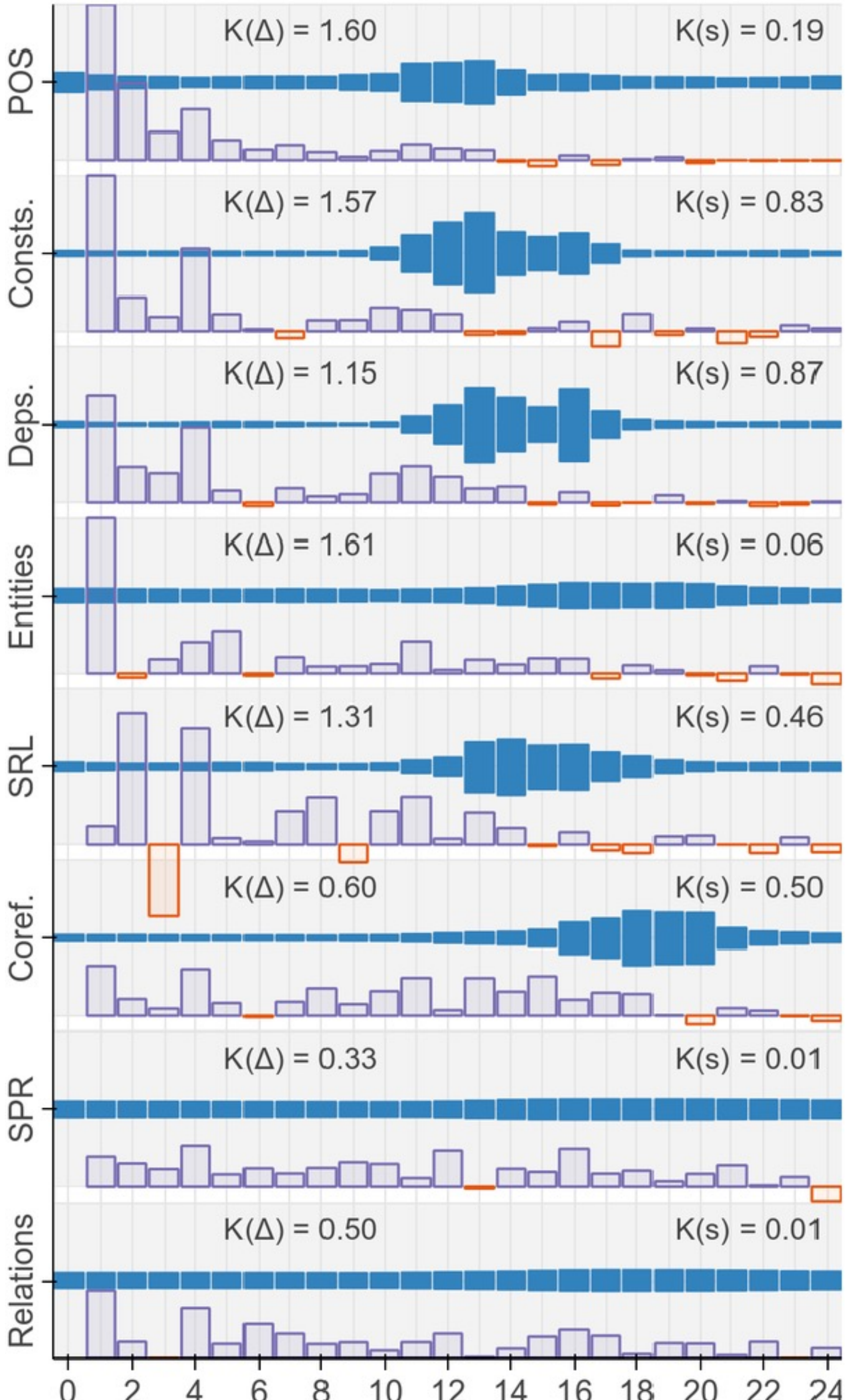
- Huge improvements over SOTA on 12 NLP task

| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.9 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 88.1 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.2 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.1 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **91.1** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **81.9** |

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

# Analysis

- *BERT Rediscovers the Classical NLP Pipeline.* Tenney et al., 2019



# Questions?