Natural Language Processing

Transformers.

My own latex definitions

Today's lecture

- Limitations of RNN
- Self and Cross-Attention
- The Transformers Architecture

This lecture material is taken from

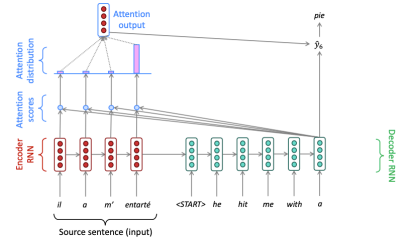
Illustrated Transformer

Limitations of RNN

Limitations of RNN: Linear interaction distance

Limitations of RNN: Lack of Parallelizability

Our last lecture: NLP + RNN

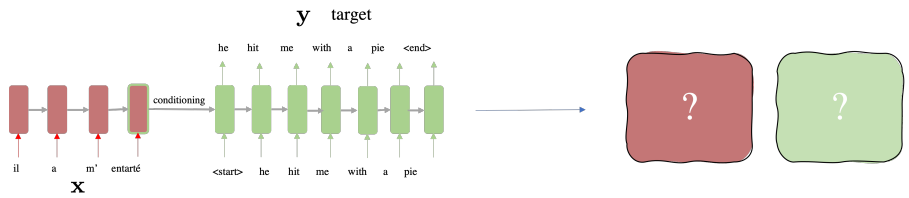


Limitations of RNN

Limitations of RNN

2014-2017

2023

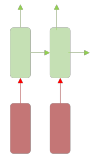


Limitations of RNN

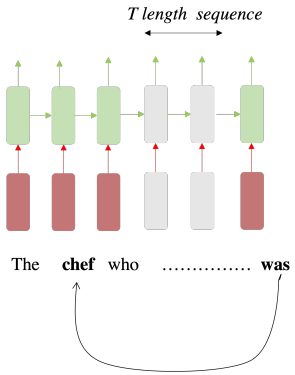
Linear interaction distance

Lack of Parallelizability

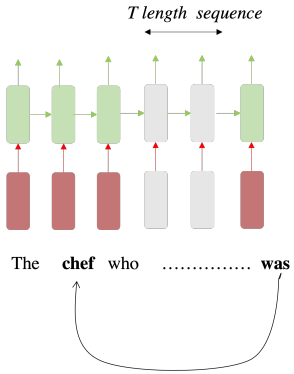
Limitations of RNN: Linear interaction distance



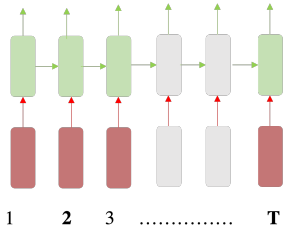
Limitations of RNN: Linear interaction distance



Limitations of RNN: Linear interaction distance

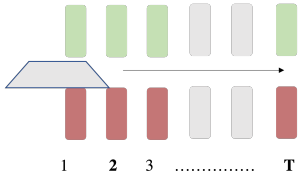


Limitations of RNN: Lack of Parallelizability

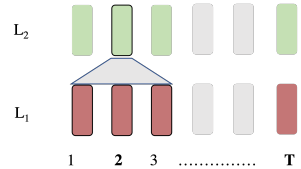


If not recurrence, then what?

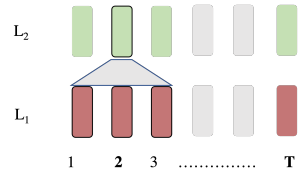
What about window-based classifier?



What about window-based classifier?

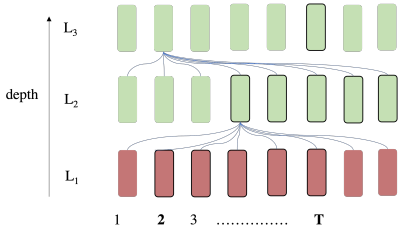


OK, good but how we make long-distance dependencies?

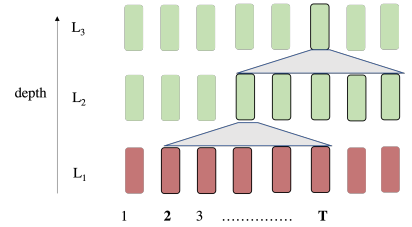


Ideas?

What about Attention? 🤯



Long-distance dependencies with local context [Receptive Field]

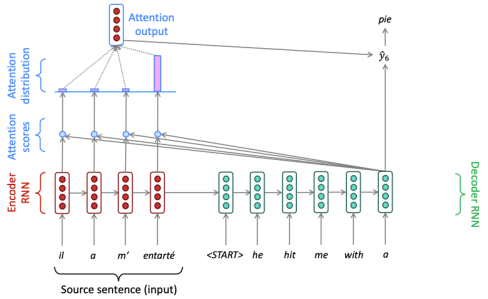


Overall take message so far:

- parallelizability in time is a must (RNN do not have that)

- we can "pass on" parallelizability over depth

What about Attention? 🤯



A few words on query, key, values

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Query, key, values terminology is from databases

In their simplest form they are collections of keys (k) and values (v). For instance, our database $\mathcal{D}$ might consist of tuples.

`{("key", "value")}`

`{("Zhang", "Aston"), ("Lipton", "Zachary"), ("Li", "Mu"), ("Smola", "Alex"), ("Hu", "Rachel"), ("Werness", "Brent")}`

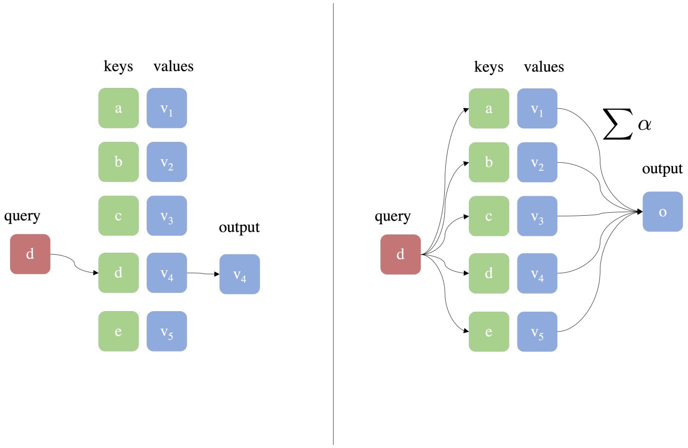
- query (q) for "Li" $\longrightarrow$ "Mu". Note query matches the key and returns the value.
- In case `("Li", "Mu")` was not a record in $\mathcal{D}$, there would be no valid answer!

- If we also allowed for **approximate matches**, we would retrieve `("Lipton", "Zachary")` instead.

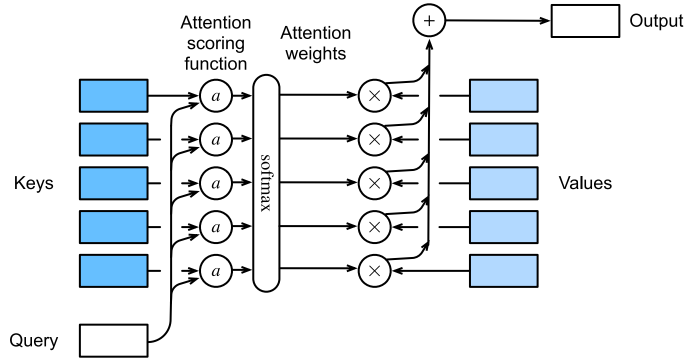
Query, key, values terminology is from databases

- We can design queries q that operate on (k,v) pairs in such a manner as to be valid regardless of the database size.
- The same query can receive different answers, according to the contents of the database.
- The "code" being executed to operate on a large state space (the database) can be quite simple (e.g., exact match, approximate match, top-k).
- There is no need to compress or simplify the database to make the operations effective.

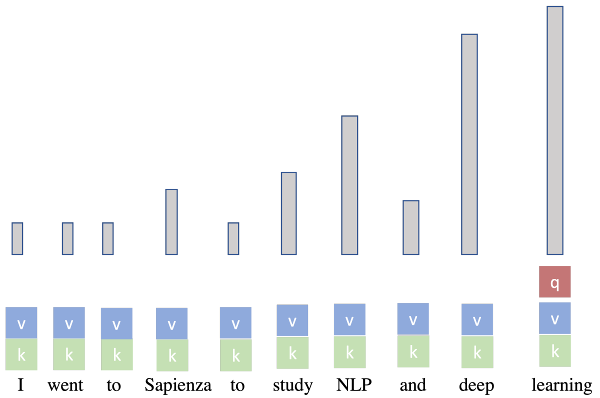
Lookup table vs Soft-average Lookup table



Self-Attention: soft, averaging lookup table



Self-Attention



Self-Attention: soft, averaging lookup table

Denote by $\mathcal{D} \stackrel{\text{def}}{=} \{(\mathbf{k}_1, \mathbf{v}_1), \ldots (\mathbf{k}_m, \mathbf{v}_m)\}$ a database of m tuples of *keys* and *values*. Moreover, denote by $\mathbf{q}$ a *query*. Then we can define the *attention* over $\mathcal{D}$ as

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i,$$

where $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ ($i = 1, \ldots, m$) are **scalar attention weights**. The operation itself is typically referred to as *attention pooling*.

The name *attention* derives from the fact that the operation pays particular attention to the terms for which the weight α is significant (i.e., large). As such, the attention over $\mathcal{D}$ generates a linear combination of values contained in the database. In fact, this contains the above example as a special case where all but one weight is zero.

Self-Attention Properties

- The weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ are nonnegative. In this case the output of the attention mechanism is contained in the convex cone spanned by the values $\mathbf{v}_i$.
- The weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ form a convex combination, i.e., $\sum_i \alpha(\mathbf{q}, \mathbf{k}_i) = 1$ and $\alpha(\mathbf{q}, \mathbf{k}_i) \geq 0$ for all i. This is the most common setting in deep learning.
- Exactly one of the weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ is 1, while all others are 0. This is akin to a traditional database query.
- All weights are equal, i.e., $\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{m}$ for all i. This amounts to averaging across the entire database, also called average pooling in deep learning.

Softmax normalization

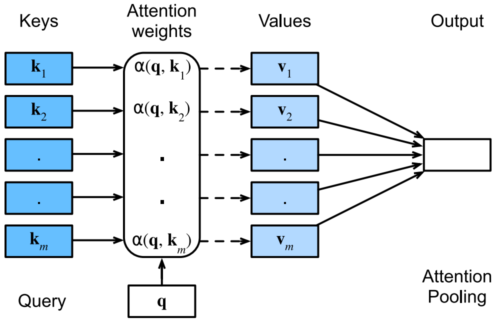
A common strategy to ensure that the weights sum up to 1 is to normalize them via

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\alpha(\mathbf{q}, \mathbf{k}_i)}{\sum_j \alpha(\mathbf{q}, \mathbf{k}_j)}.$$

In particular, to ensure that the weights are also nonnegative, one can resort to exponentiation. This means that we can now pick *any* function $a(\mathbf{q}, \mathbf{k})$ and then apply the softmax operation used for multinomial models to it via

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_j \exp(a(\mathbf{q}, \mathbf{k}_j))}.$$

Self-Attention

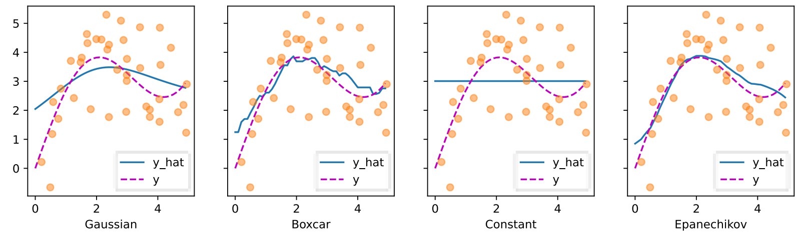


So far, nothing is learnable, just linear combination

Before going into the learning part, let us make a connection with classic machine learning

Attention Pooling via Nadaraya-Watson Regression [1964]

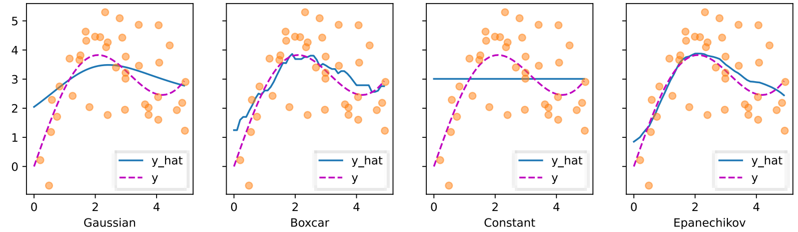
$$f(\mathbf{x}) = \sum_i \mathbf{y}_i \underbrace{\frac{\alpha(\mathbf{x}, \mathbf{x}_i)}{\sum_j \alpha(\mathbf{x}, \mathbf{x}_j)}}_{\text{attention}}.$$



[Taken from d2l.ai]](https://d2l.ai/chapter_attention-mechanisms-and-transformers/attention-pooling.html)

Attention Pooling via Nadaraya-Watson Regression

$$f(\mathbf{q}) = \sum_i \mathbf{v}_i \underbrace{\frac{\alpha(\mathbf{q}, \mathbf{k}_i)}{\sum_j \alpha(\mathbf{q}, \mathbf{k}_j)}}_{\text{attention}}$$



[Taken from d2l.ai](https://d2l.ai/chapter_attention-mechanisms-and-transformers/attention-pooling.html)

Attention Pooling via Nadaraya-Watson Regression

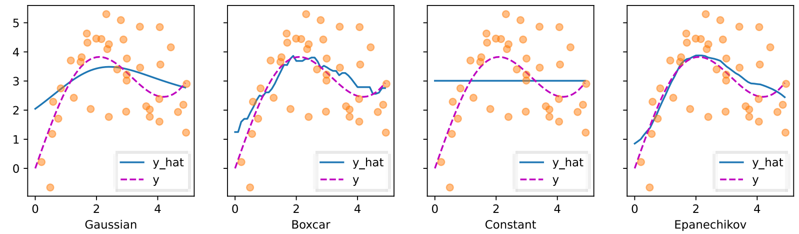
The way you compare $\alpha(\mathbf{q}, \mathbf{k}_i)$ gives rise at different shape in the regression (control the range and smoothness).

$$\begin{aligned}
\alpha(\mathbf{q}, \mathbf{k}) &= \exp\left(-\frac{1}{2}\|\mathbf{q}-\mathbf{k}\|^2\right) && \text{Gaussian} \\
\alpha(\mathbf{q}, \mathbf{k}) &= 1 \text{ if } \|\mathbf{q}-\mathbf{k}\| \leq 1 && \text{Boxcar} \\
\alpha(\mathbf{q}, \mathbf{k}) &= 1 && \text{constant} \\
\alpha(\mathbf{q}, \mathbf{k}) &= \max(0, 1-\|\mathbf{q}-\mathbf{k}\|) && \text{Epanechikov}
\end{aligned}$$

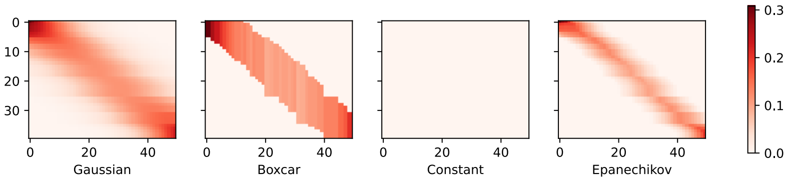
Attention Pooling via Nadaraya-Watson Regression

```python
def nadaraya_watson(x_train, y_train, x_val, kernel):
    dists = x_train.reshape((-1, 1)) - x_val.reshape((1, -1))
    # Each column/row corresponds to each query/key
    k = kernel(dists).type(torch.float32)
    # Normalization over keys for each query
    attention_w = k / k.sum(0)
    y_hat = y_train@attention_w
    return y_hat, attention_w
```

Attention Weights



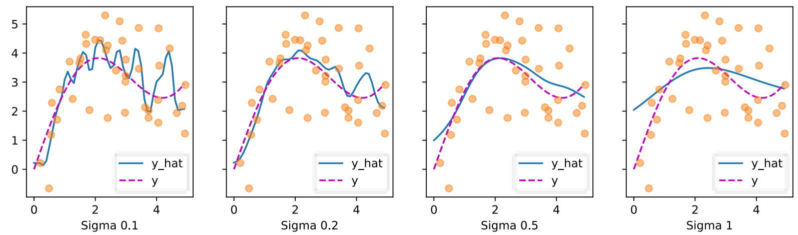
Note that, besides the constant kernel, they are all similar.

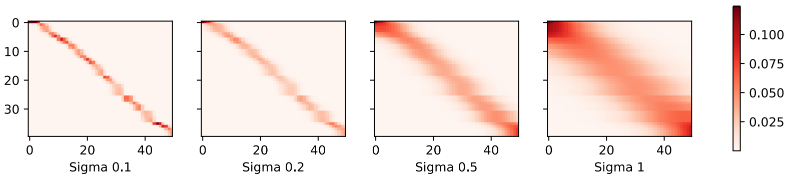


Why not stick to Gaussian kernel and tune its bandwidth?

Let's tune the bandwidth parameter in the [Gaussian] kernel

$$\alpha(\mathbf{q}, \mathbf{k}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{q}-\mathbf{k}\|^2\right)$$





Instead of tuning the bandwidth why not learning?

Learnable self-attention

Let $\{w_1, \dots, w_n\}$ be a sequence of words in vocabulary V, like

`Iacopo made his daughter food`.

For each word token w_i, let $\mathbf{x}_i = \mathbf{E}w_i$, where $\mathbf{E} \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

Learnable self-attention

1. We transform each word embedding $\mathbf{x}_i$ with **learnable weight matrices** $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times d}$

$$\mathbf{q}_i = \mathbf{Q}\mathbf{x}_i \quad \mathbf{k}_i = \mathbf{K}\mathbf{x}_i \quad \mathbf{v}_i = \mathbf{V}\mathbf{x}_i$$

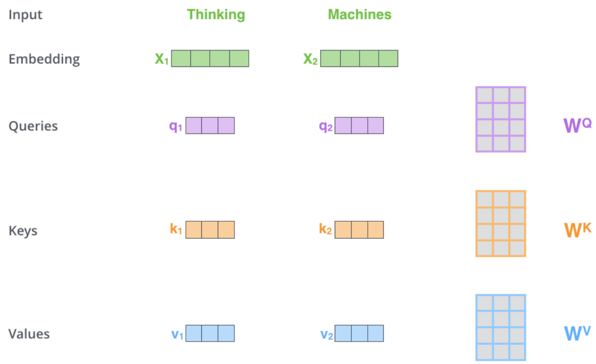
2. Compute pairwise similarities between keys and queries; normalize with softmax (across keys):

$$e_{ij} = \mathbf{q}_i^\top \mathbf{k}_j \quad \boldsymbol{\alpha}_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})}$$

1. Compute output for each word as weighted sum of values:

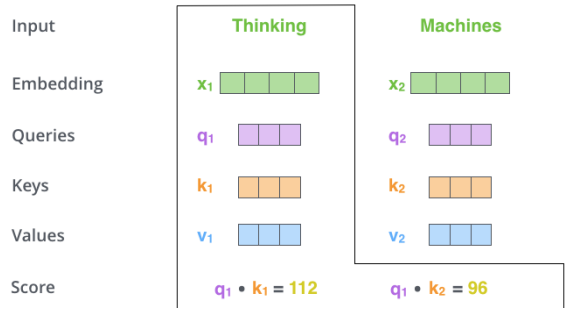
$$\mathbf{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \mathbf{v}_i$$

Computation with two word tokens

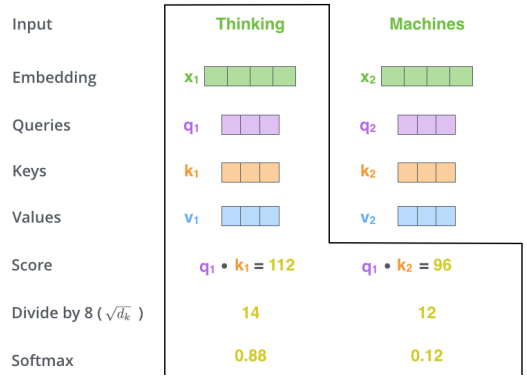


[Taken from illustrated-transformer](http://jalammar.github.io/illustrated-transformer/)

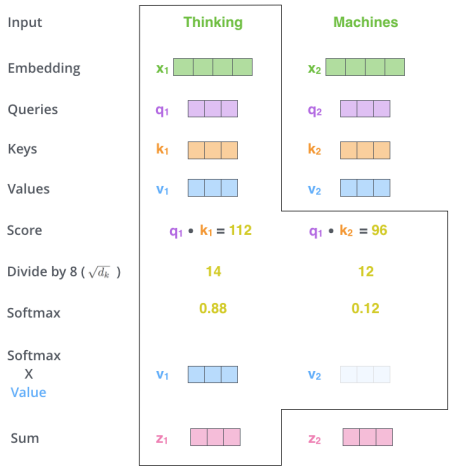
Computation with two word tokens



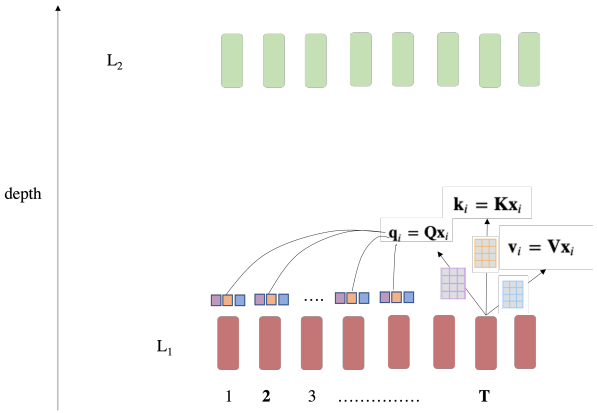
Computation with two word tokens

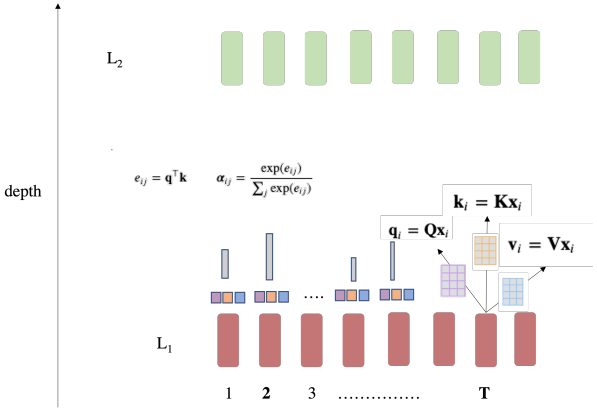


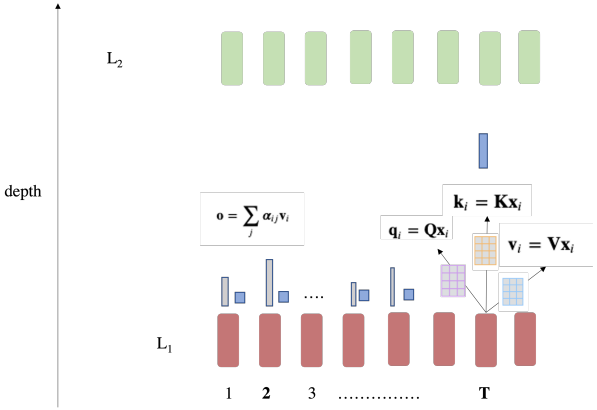
Computation with two word tokens



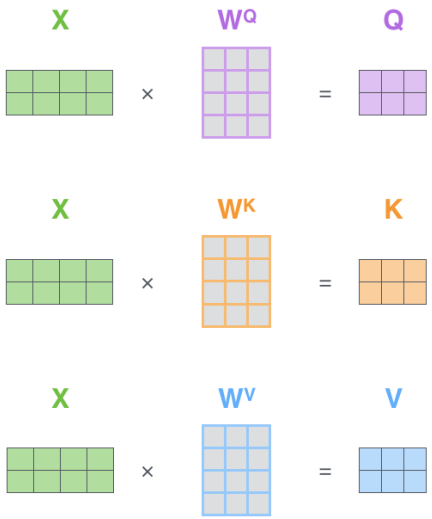
Computation with more than 2 word tokens

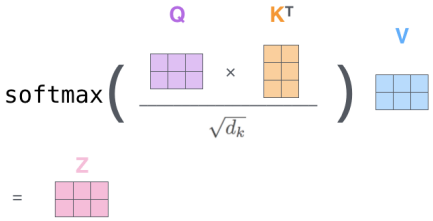




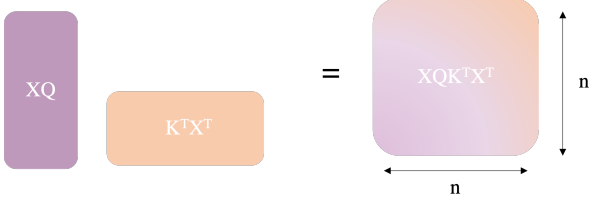


Attention can be easily made into tensor

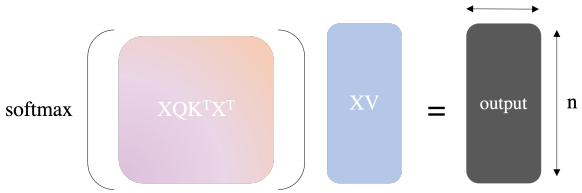




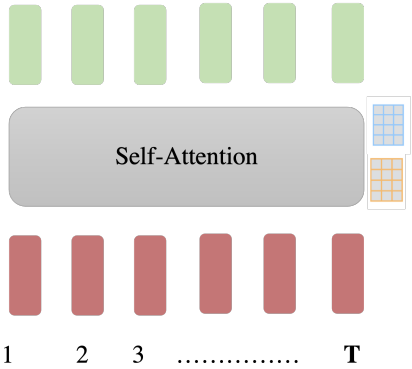
Attention with tensor



Self-Attention with tensor



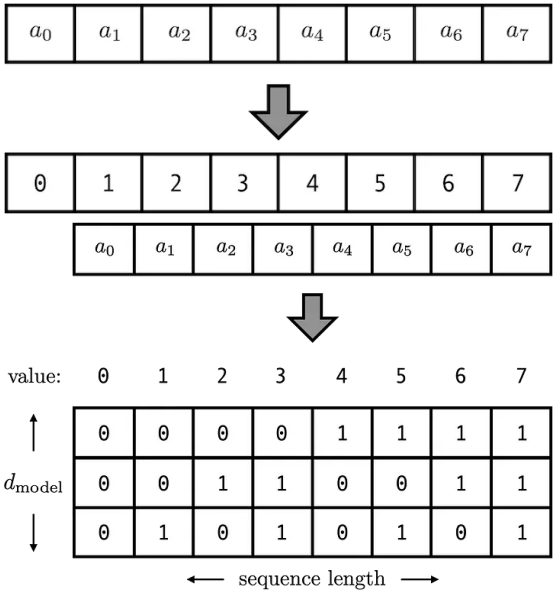
Are we done with Self-Attention?



Weight to learn:
Q, V, K

No! Still many things to fix 🤔

Self-attention as a NLP block: Fix 1) No notion of order!

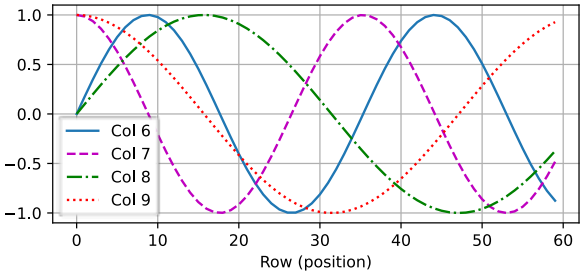


Positional Encoding

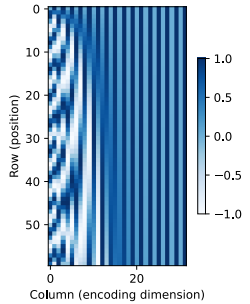
Absolute Positional Information

Positional Encoding

Positional Encoding

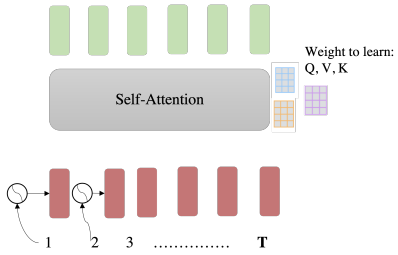


Positional Encoding – Absolute Positional Info



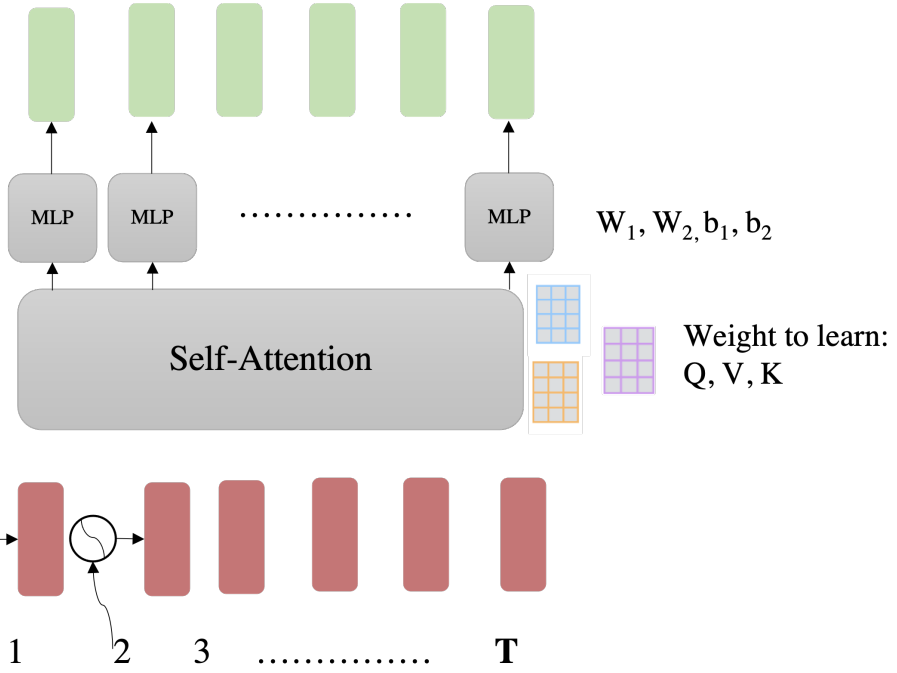
Positional Encoding – Relative Positional Info

Self-Attention so far

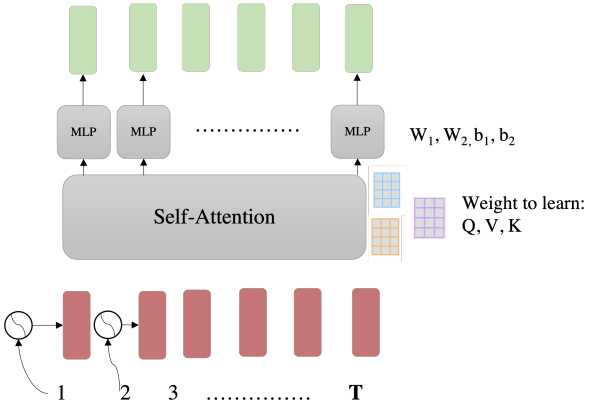


Self-attention as a NLP block: Fix 2) No nonlinearities! Just weighted average.

Fix 2) Adding non-linearity



Positionwise Feed-Forward Networks (MLP)



Self-attention as a NLP block: Fix 3) We can look in the future!

Only used in the Decoder

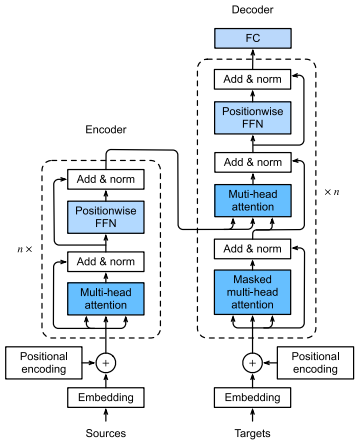
Masking Self-attention

We attend to those non-gray

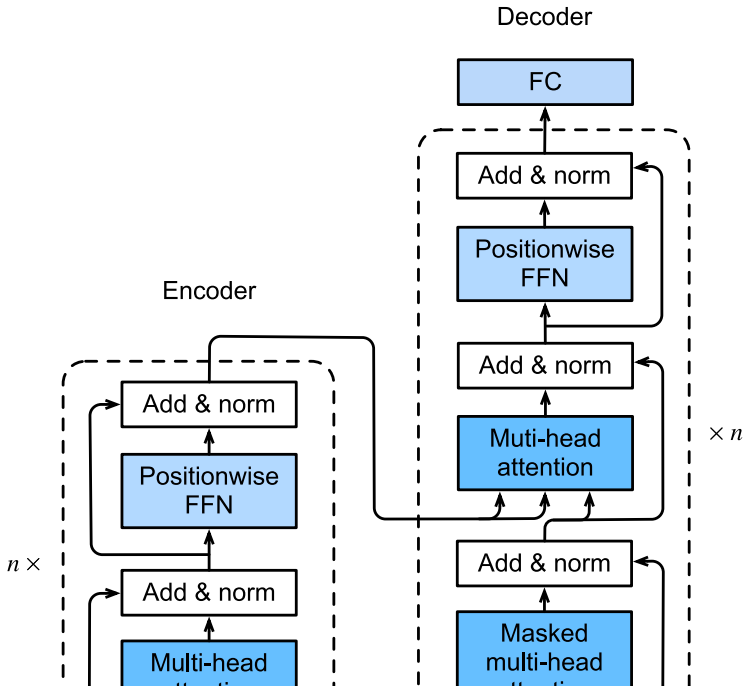
For encoding this word

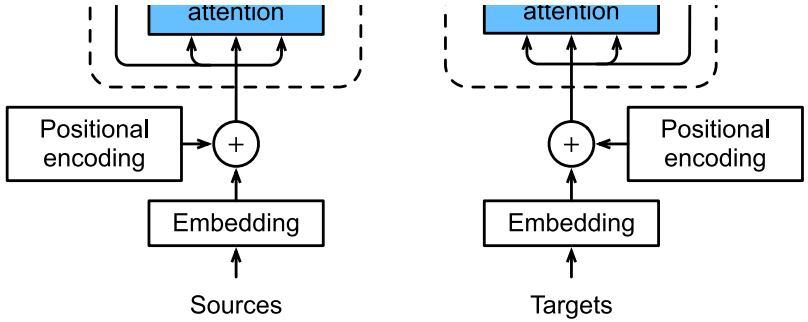
	<start>	Iacopo	made	his
<start>		-∞	-∞	-∞
Iacopo			-∞	-∞
made				-∞
his				

The Transformer



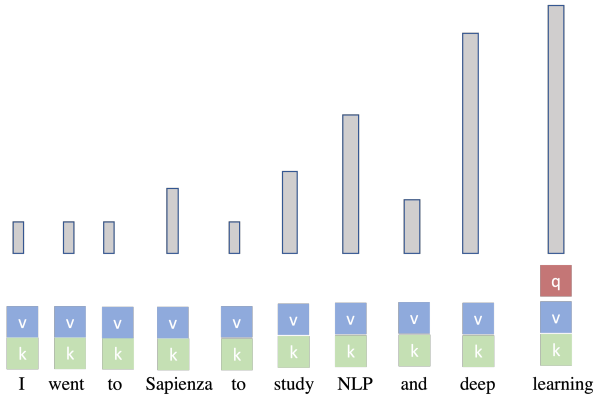
Missing pieces
1) Multi-Head Attention
2a) Residual Connection
2b) Layer Normalization
2c) Scaled Dot-Product





The world is "multi-modal"

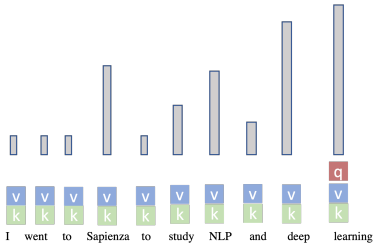
Self-Attention



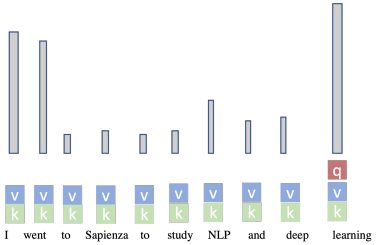
Multi-head self-attention

Multi-head self-attention

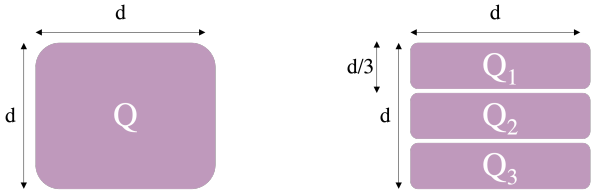
Attention mechanism #1 attends to **"entities"**



Attention mechanism #2 attends to **"syntactically relevant work"**

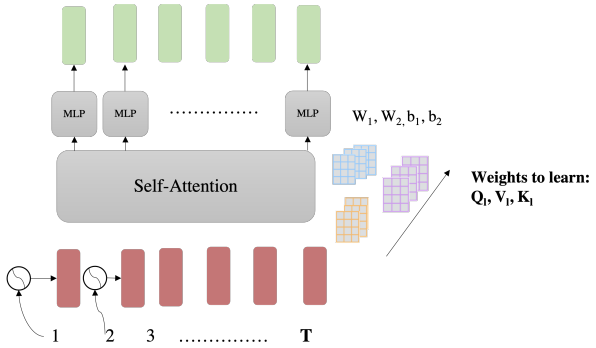


Multi Head Self-Attention with tensor

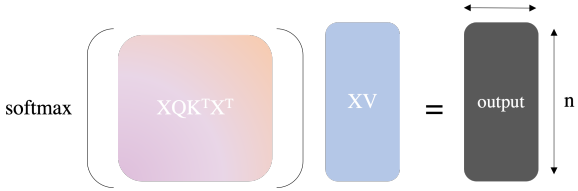


Multi Head Self-Attention with tensor

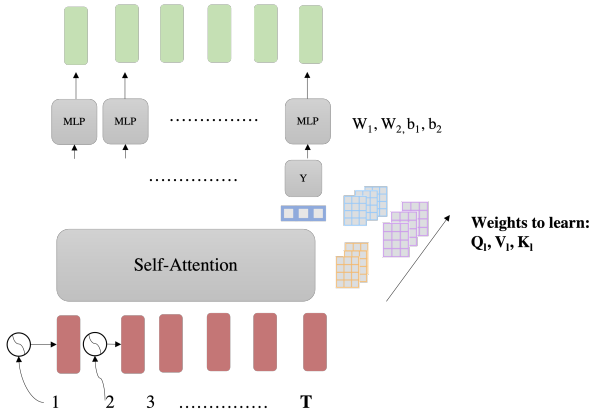
Multi-head self-attention



Self-Attention with tensor



1) Multi Head Self-Attention



2a) Residual Connection [He et al., 2016]

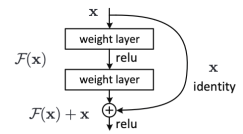


Figure 2. Residual learning: a building block.

2a) Residual Connection

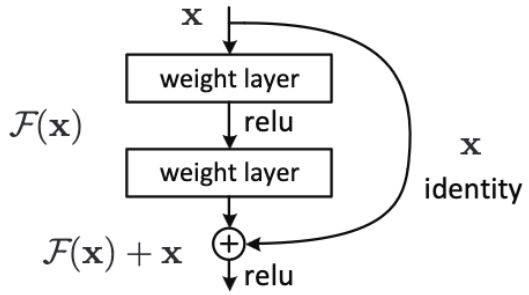
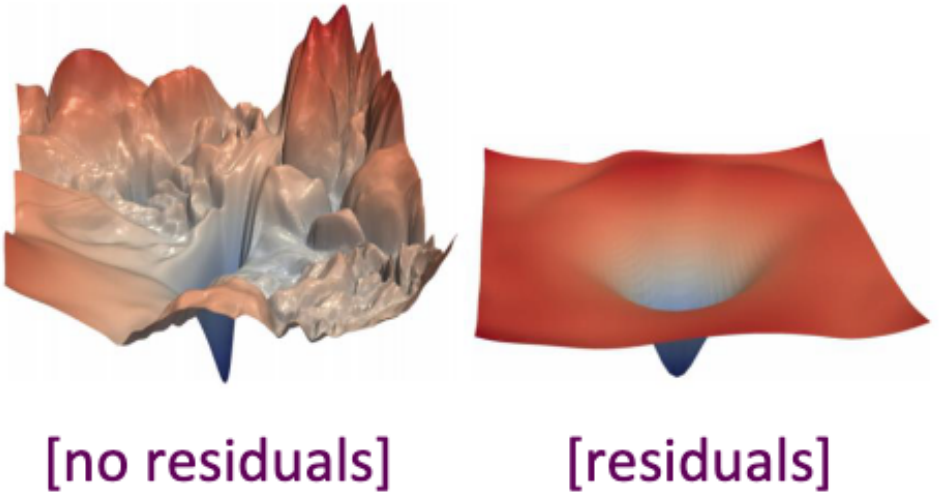


Figure 2. Residual learning: a building block.



2b) Layer Normalization [Ba et al., 2016]

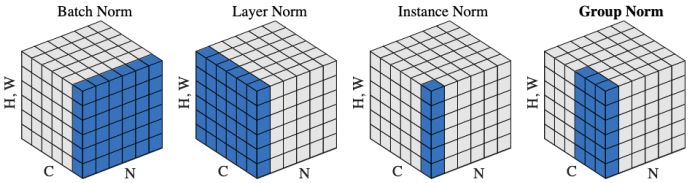
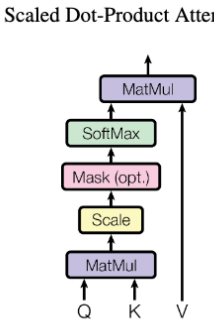
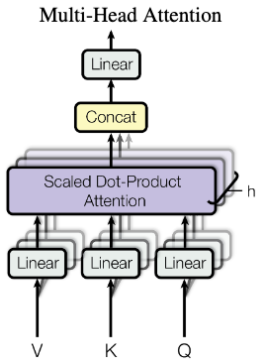


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

2c) Scaled Dot Product

The Transformer





The Transformer

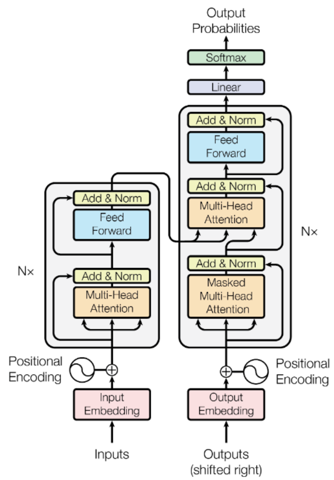


Figure 1: The Transformer - model architecture.

Cross-Attention

- Self-attention, query, key and values come from the same **source**
- In the decoder, we have attention that looks more like what we saw in NMT.
- $\{\mathbf{h}_1, \ldots, \mathbf{h}_T\}$ last output from the encoder
- $\{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ is the input of the decoder [this can be input representation of input to next layer]

Keys and Values are drawn from the encoder (like a memory)

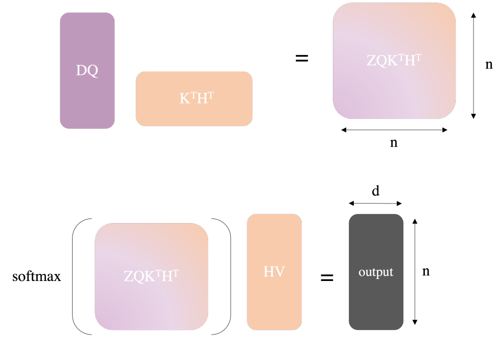
- $\underbrace{\mathbf{k}_i = \mathbf{K}\mathbf{h}_i \quad \mathbf{v}_i = \mathbf{V}\mathbf{h}_i}_{encoder} \quad \underbrace{\mathbf{q}_i = \mathbf{Q}\mathbf{z}_i}_{decoder}$

Cross-Attention with tensor

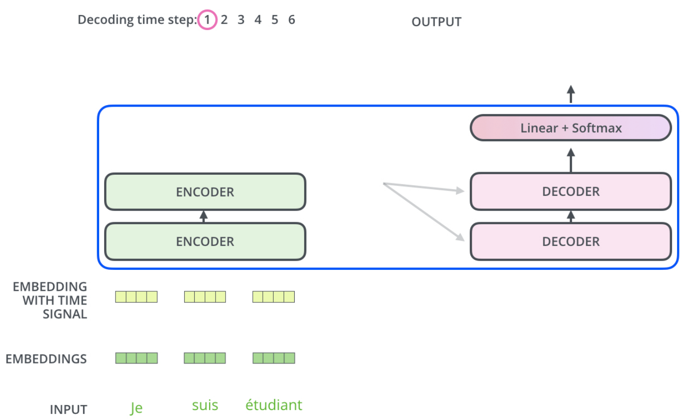
$\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_n] \in \mathbb{R}^{n \times d}$ is a matrix of concatenated last encodings.

$\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_n] \in \mathbb{R}^{n \times d}$ is a matrix of concatenated input in the decoder.

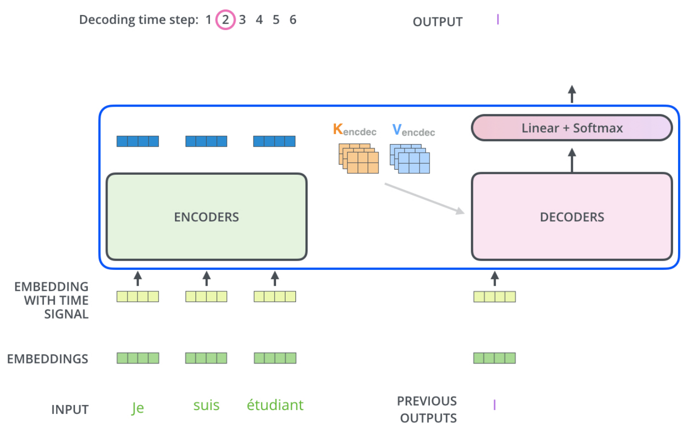
The output is defined as $\mathrm{softmax}(\mathbf{DQ}(\mathbf{HK})^\top) \cdot \mathbf{HV}$



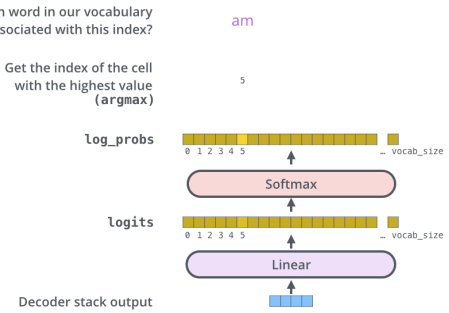
Transformers Decoding



Transformers Decoding



Final Linear Layer

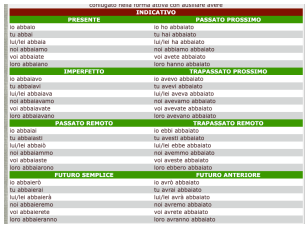


Machine Translation Performance

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Word structure and subword models: Words Morphology



Subword modeling: we give up the assumption of single word token

The Byte Pair Encoding [2015]

The Byte Pair Encoding

The Byte Pair Encoding

The Byte Pair Encoding

The Byte Pair Encoding

The Byte Pair Encoding

Merging process by most frequent adjacent subwords

English syntax and rules emerge

Out of Vocabulary

Text Summarization

Table 4: Performance of best models of each model architecture using the combined corpus and tf-idf extractor.

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

Natural Language Processing

Today's lecture

This lecture material is taken from

Subword Modeling

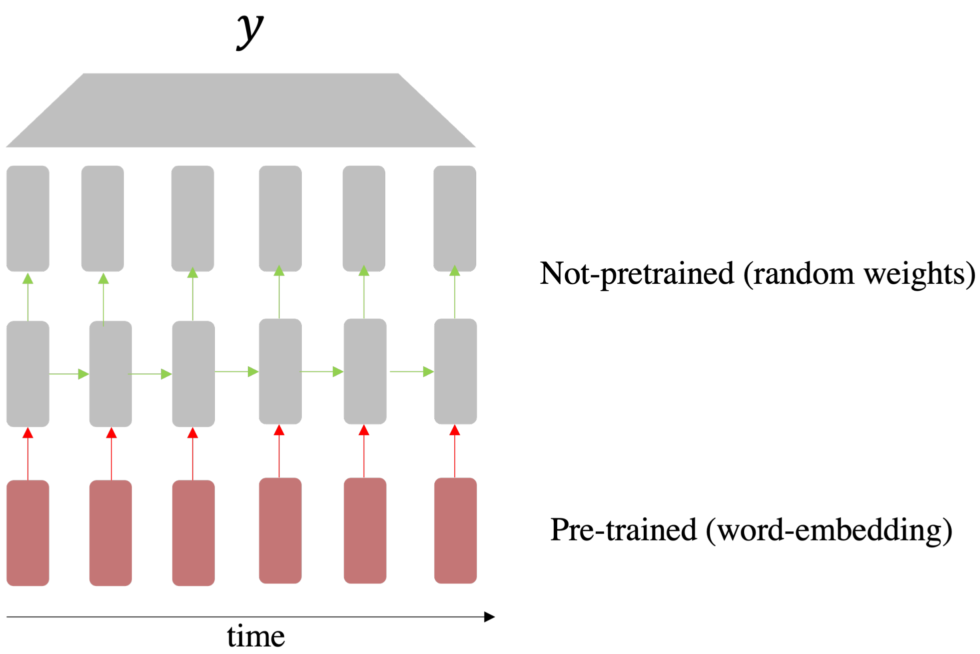
Word structure and subword models

Contextual Embedding

Motivating word meaning and context

Where we were: pretrained word embeddings

Pretrained word embeddings



Where we are: pretrained whole model

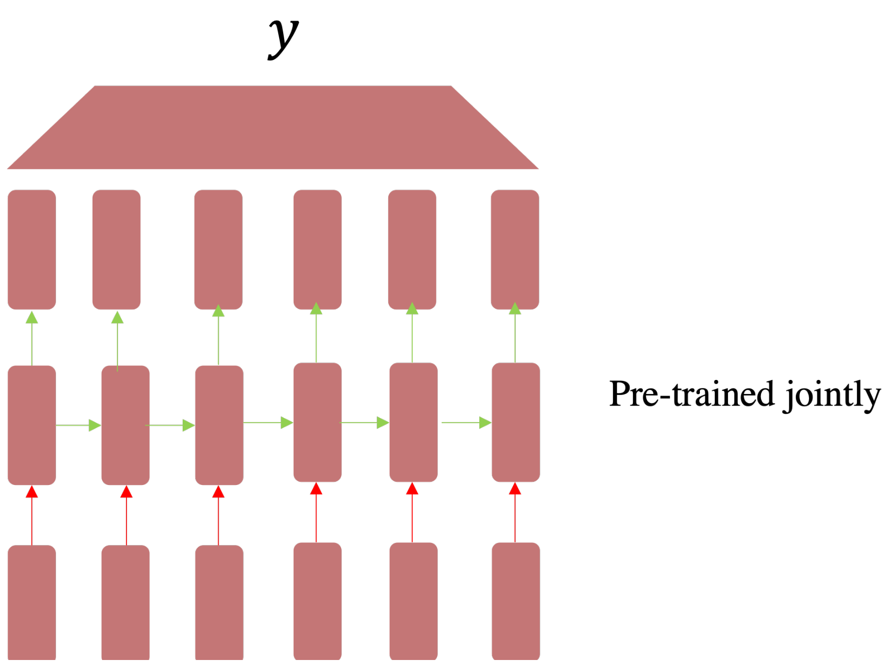
In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining.**
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts

This has been exceptionally effective at building strong:

- **representations of language**
- **parameter initializations** for strong NLP models.
- **Probability distributions** over language that we can sample from

Pretrained model



Pretraining Self-Supervised Learning

What can we learn from reconstructing the input?

Sapienza University is located in __________, Italy

What can we learn from reconstructing the input?

I put ___ fork down on the table.

What can we learn from reconstructing the input?

The woman walked across the street, checking for traffic over ___ shoulder.

What can we learn from reconstructing the input?

I went to the ocean to see the fish, turtles, seals, and _____.

What can we learn from reconstructing the input?

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink.
The movie was ______.

What can we learn from reconstructing the input?

Iroh went into the kitchen to make some tea.
Standing next to Iroh, Zuko pondered his destiny.
Zuko left the ______.

The Transformer (Encoder and Decoder)

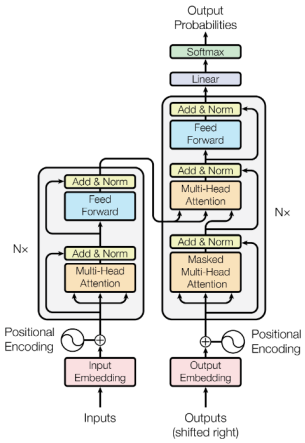


Figure 1: The Transformer - model architecture.

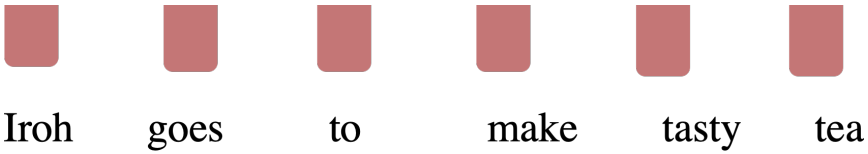
Pretraining through language modeling (LM)

Recall the language modeling task:

- Model $p_\theta(w_t | w_1, \ldots, w_{t-1})$, the probability distribution over words given their past contexts.
- There is lots, of data for free (on the internet)

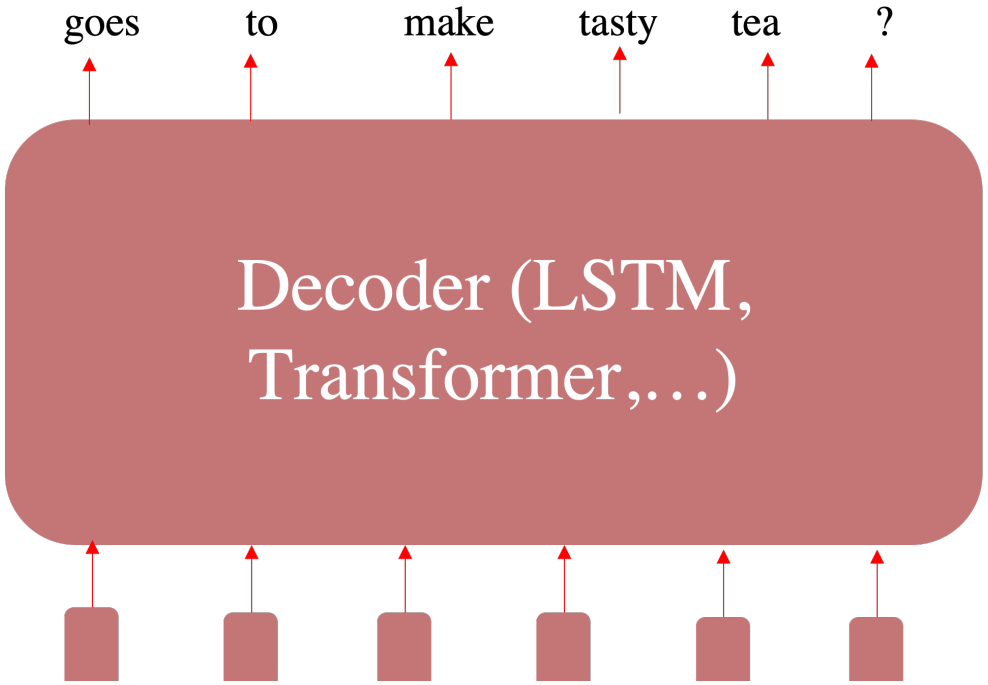
Pretraining through language modeling:

- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.



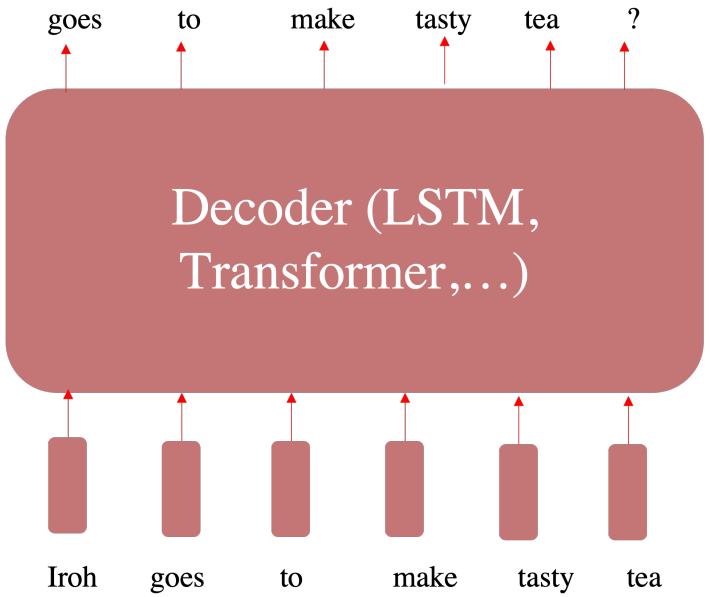
The Pretraining / Finetuning Paradigm

Now Pretraining can improve NLP!

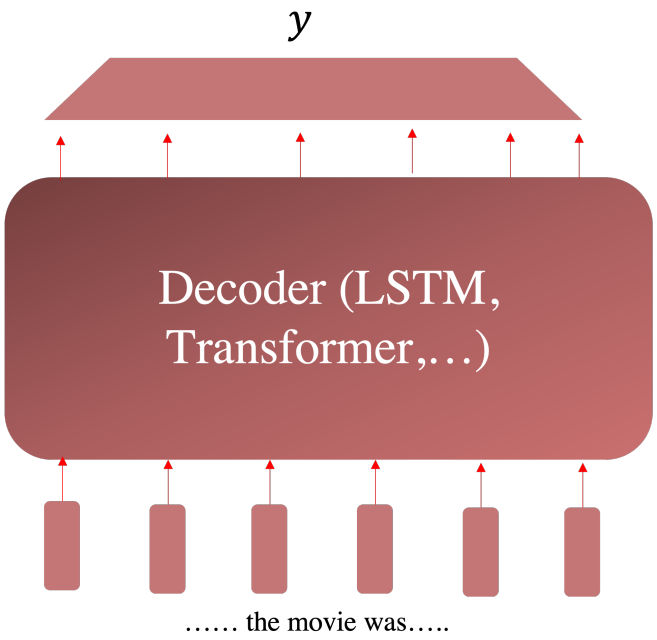


The Pretraining

Step 1: Pretrain (w/ language modeling): Lots of text; learn general things!



Finetuning Paradigm



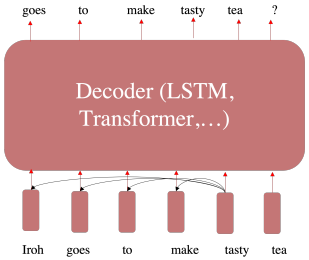
Pretraining/finetuning: optimization perspective

Pretraining/finetuning: optimization perspective

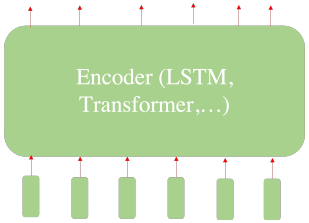
Three ways to pre-train a model

Pretraining for three types of architectures

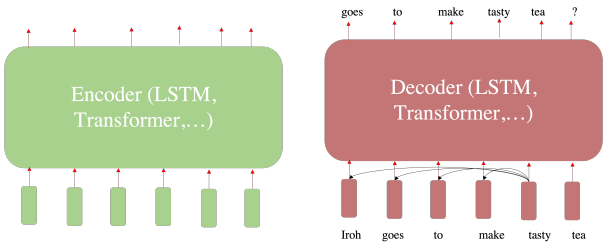
1) Decoder Only



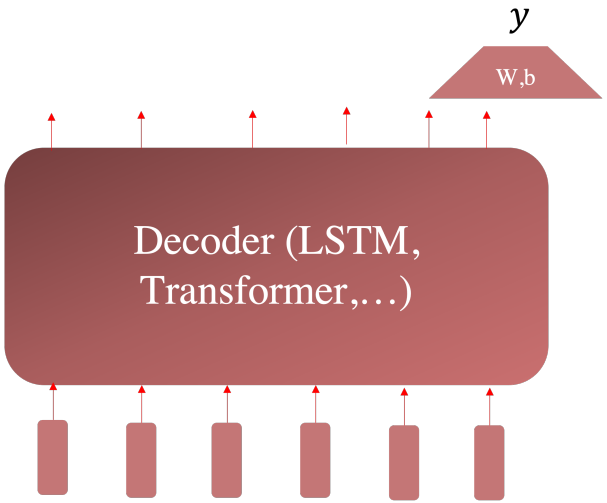
2) Encoder Only



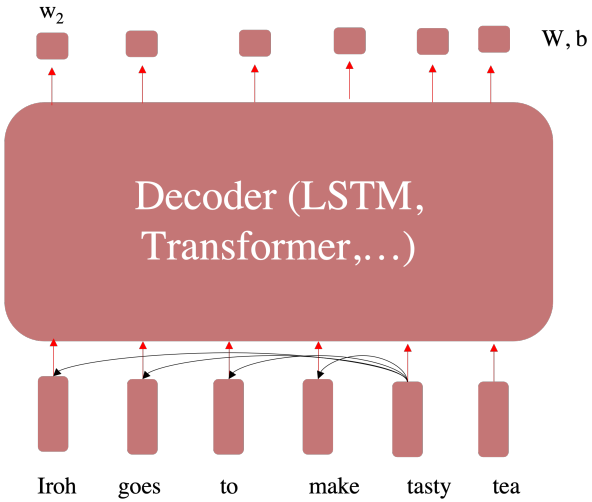
3) Encoder-Decoder

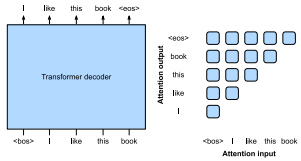


1) Decoder only - Option 1



1) Decoder only - Option 2





Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

1 Introduction

The ability to learn effectively from raw text is crucial to alleviating the dependence on supervised learning in natural language processing (NLP). Most deep learning methods require substantial amounts of manually labeled data, which restricts their applicability in many domains that suffer from a dearth of annotated resources [61]. In these situations, models that can leverage linguistic information from unlabeled data provide a valuable alternative to gathering more annotation, which

Table 2: Experimental results on natural language inference tasks, comparing our model with current state-of-the-art methods. 5x indicates an ensemble of 5 models. All datasets use accuracy as the evaluation metric.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

GPT2: increasingly convincing generations [Radford et al., 2018]

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

GPT3

Language Models are Few-Shot Learners

Tom B. Brown* **Benjamin Mann*** **Nick Ryder*** **Melanie Subbiah***

Jared Kaplan† **Prafulla Dhariwal** **Arvind Neelakantan** **Pranav Shyam** **Girish Sastry**

Amanda Askell **Sandhini Agarwal** **Ariel Herbert-Voss** **Gretchen Krueger** **Tom Henighan**

Rewon Child **Aditya Ramesh** **Daniel M. Ziegler** **Jeffrey Wu** **Clemens Winter**

Christopher Hesse **Mark Chen** **Eric Sigler** **Mateusz Litwin** **Scott Gray**

Benjamin Chess **Jack Clark** **Christopher Berner**

Sam McCandlish **Alec Radford** **Ilya Sutskever** **Dario Amodei**

OpenAI

GPT3

Abstract

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

GPT3

GPT3 has **175 Billion parameters**, trained on **45000 Gbytes of text** (context window is **2048 tokens** in her paper).

We train GPT-3 with 175 billion parameters, 10x more than any previous non-sparse language model, and evaluate in the few-shot setting. For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

GPT3 - Huge Model, Huge Training set

GPT3 has **175 Billion parameters**, trained on **45000 Gbytes of text** (context window is **2048 tokens** in her paper).

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

GPT3 - In-context learning

We test "internal" ability of models in two ways:

- Sample from the distribution they define (maybe providing a prompt)
- Ask how likely it is that we can solve some task from their predictions

Many large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

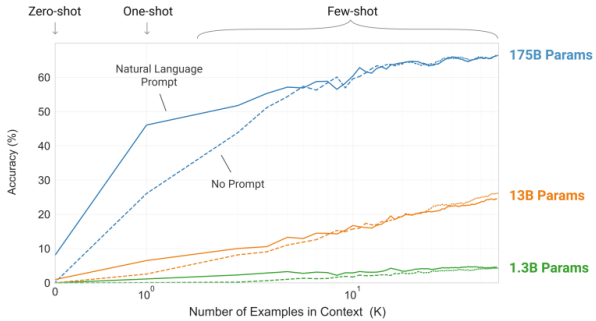


Figure 1.2: Larger models make increasingly efficient use of in-context information. We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper "in-context learning curves" for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.

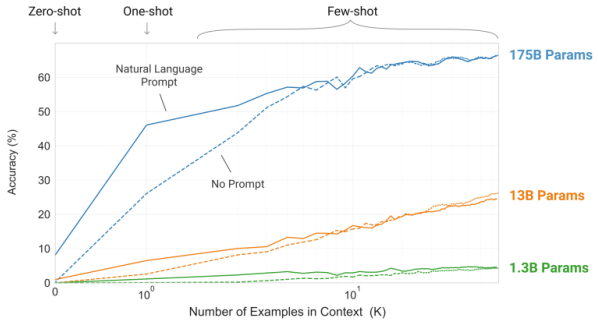


Figure 1.2: Larger models make increasingly efficient use of in-context information. We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper "in-context learning curves" for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

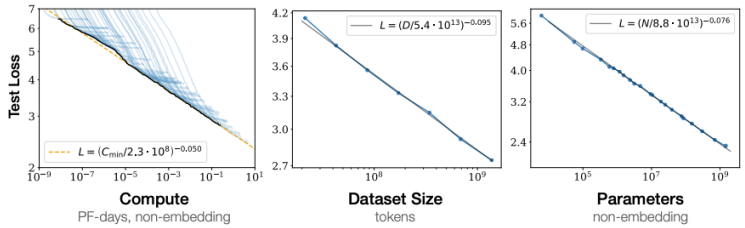


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width. (Section 3)

Smooth power laws: Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude (see Figure 1). We observe no signs of deviation from these trends on the upper end, though performance must flatten out eventually before reaching zero loss. (Section 3)

Universality of overfitting: Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. The performance penalty depends predictably on the ratio $N^{0.74}/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty. (Section 4)

Universality of training: Training curves follow predictable power-laws whose parameters are roughly independent of the model size. By extrapolating the early part of a training curve, we can roughly predict the loss that would be achieved if we trained for much longer. (Section 5)

Transfer improves with test performance: When we evaluate models on text with a different distribution than they were trained on, the results are strongly correlated to those on the training validation set with a roughly constant offset in the loss – in other words, transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set. (Section 3.2.2)

Sample efficiency: Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps (Figure 2) and using fewer data points (Figure 4).

Convergence is inefficient: When working within a fixed compute budget C but without any other restrictions on the model size N or available data D, we attain optimal performance by training *very large models* and stopping *significantly short of convergence* (see Figure 3). Maximally compute-efficient training would therefore be far more sample efficient than one might expect based on training small models to convergence, with data requirements growing very slowly as $D \sim C^{0.27}$ with training compute. (Section 6)

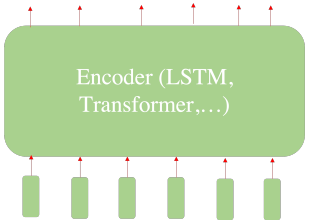
Optimal batch size: The ideal batch size for training these models is roughly a power of the loss only, and continues to be determinable by measuring the gradient noise scale [MKAT18]; it is roughly 1-2 million tokens at convergence for the largest models we can train. (Section 5.1)

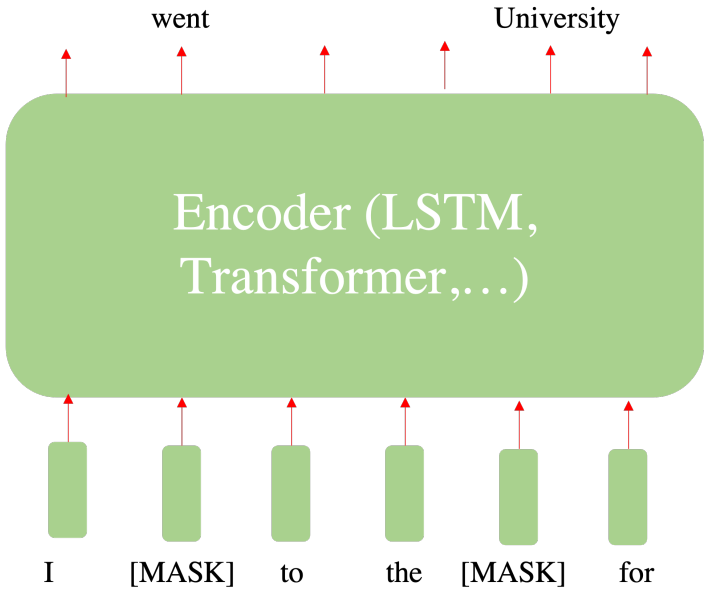
Taken together, these results show that language modeling performance improves smoothly and predictably as we appropriately scale up model size, data, and compute. We expect that larger language models will perform better and be more sample efficient than current models.

2) Encoder Only

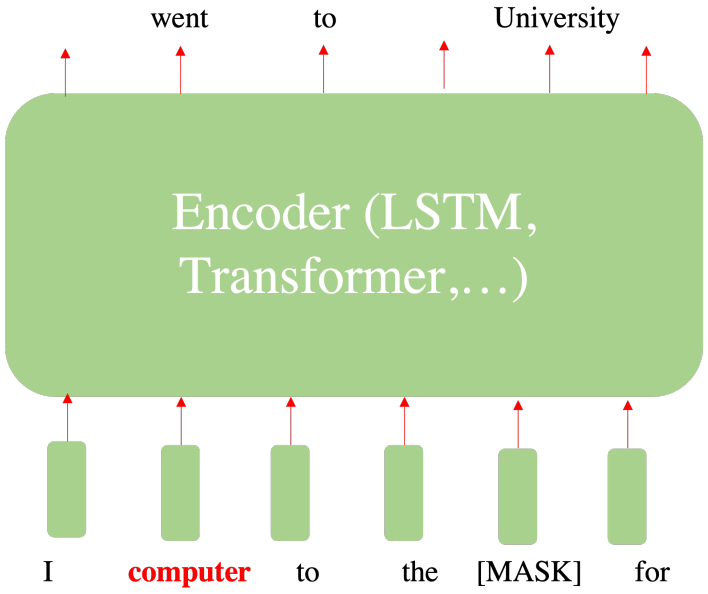
Masked Language Modeling + BERT (Bidirectional Encoder Representations from Transformers)

Masked Language Modeling

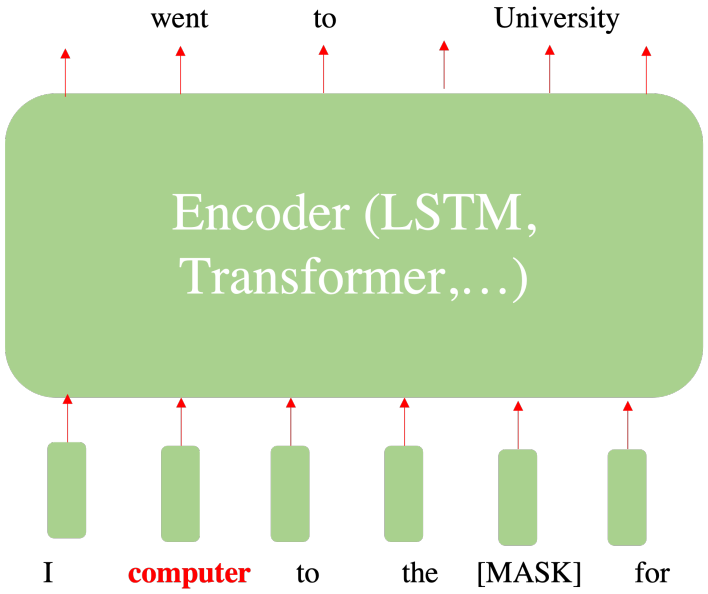




token



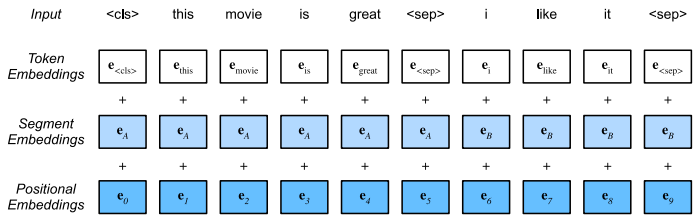
Replace w/ random



Not replaced
Yet predicted

BERT: Masked LM

BERT: Next Sentence Prediction

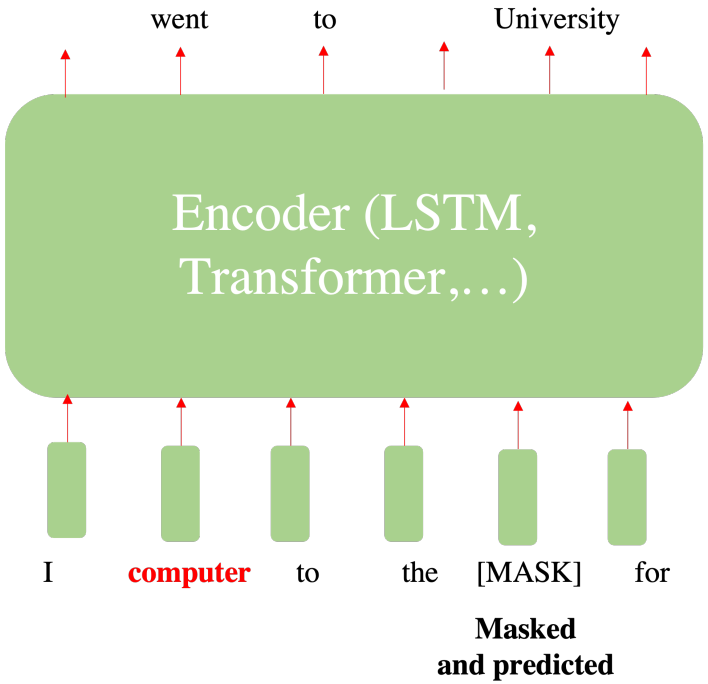


BERT Details

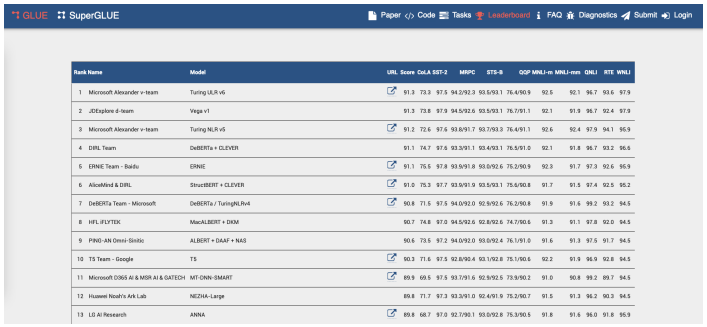
BERT: One model many tasks

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). 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.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.



GLUE Benchmarks



Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI
1	Microsoft Alexander v-team	Turing ULR v6		91.3	73.3	97.5	94.2/92.3	93.5/93.1	76.4/90.9	92.6		92.1	96.7	93.6	97.9
2	JDExplore d-team	Vega v1		91.3	73.8	97.9	94.5/92.6	93.5/93.1	76.7/91.1	92.1		91.9	96.7	92.4	97.9
3	Microsoft Alexander v-team	Turing NLR v5		91.2	72.6	97.6	93.8/91.7	93.7/93.3	76.4/91.1	92.6		92.4	97.9	94.1	95.9
4	DIRL Team	DeBERTa + CLEVER		91.1	74.7	97.6	93.3/91.1	93.4/93.1	76.5/91.0	92.1		91.8	96.7	93.2	96.6
5	ERNIE Team - Baidu	ERNIE		91.1	75.5	97.8	93.9/91.8	93.0/92.6	75.2/90.9	92.3		91.7	97.3	92.6	95.9
6	AliceMind & DIRL	StructBERT + CLEVER		91.0	75.3	97.7	93.9/91.9	93.5/93.1	75.6/90.8	91.7		91.5	97.4	92.5	95.2
7	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8	71.5	97.5	94.0/92.0	92.9/92.6	76.2/90.8	91.9		91.6	99.2	93.2	94.5
8	HFL-IFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3		91.1	97.8	92.0	94.5
9	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6		91.3	97.5	91.7	94.5
10	T5 Team - Google	T5		90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2		91.9	96.9	92.8	94.5
11	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART		89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0		90.8	99.2	89.7	94.5
12	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3	93.3/91.0	92.4/91.9	75.2/90.7	91.5		91.3	96.2	90.3	94.5
13	LG AI Research	ANNA		89.8	68.7	97.0	92.7/90.1	93.0/92.8	75.3/90.5	91.8		91.6	96.0	91.8	95.9

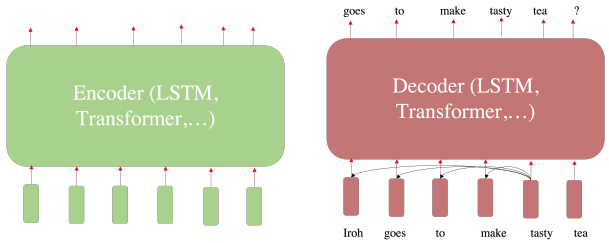
BERT EXtensions

RoBERTa: A Robustly Optimized BERT Pretraining Approach

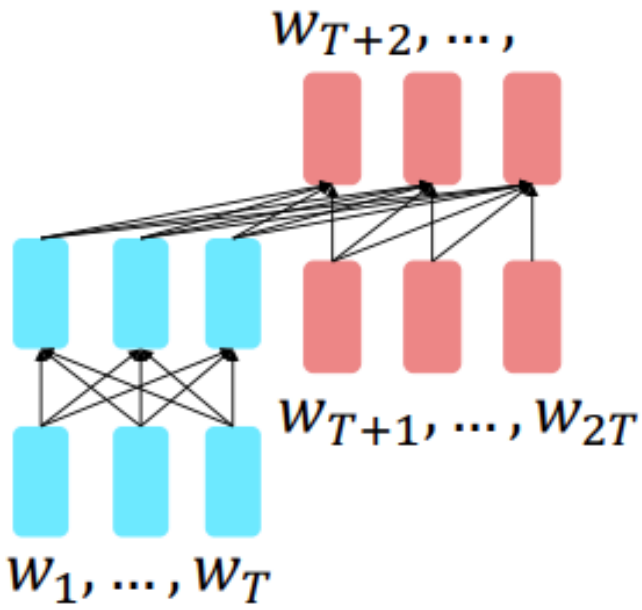
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

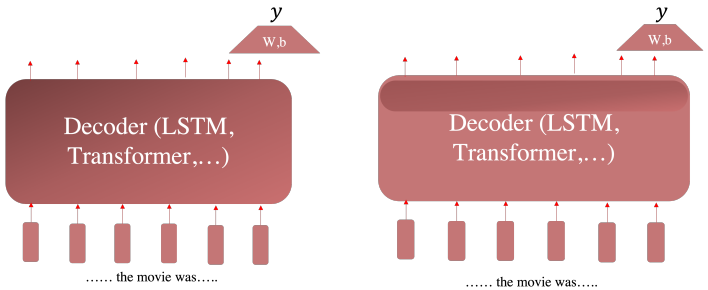
3) Encoder-Decoder



3) Encoder-Decoder Pretraining [T5 Model]



Full Fine-tuning vs. Parameter-Efficient Fine-tuning



Parameter-Efficient Fine-tuning: Prefix-Tuning, Prompt tuning

Fine-tuning approaches

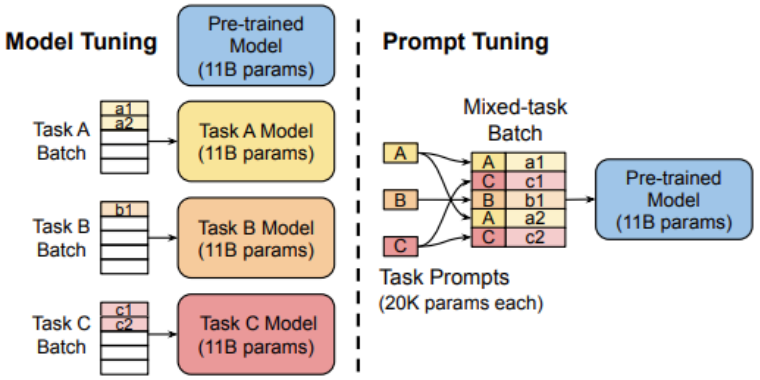


Figure 2: **Model tuning** requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches. **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pre-trained model. With a T5 "XXL" model, each copy

trained model. With a T5 XXL model, each copy of the tuned model requires 11 billion parameters. By contrast, our tuned prompts would only require 20,480 parameters per task—a reduction of *over five orders of magnitude*—assuming a prompt length of 5 tokens.

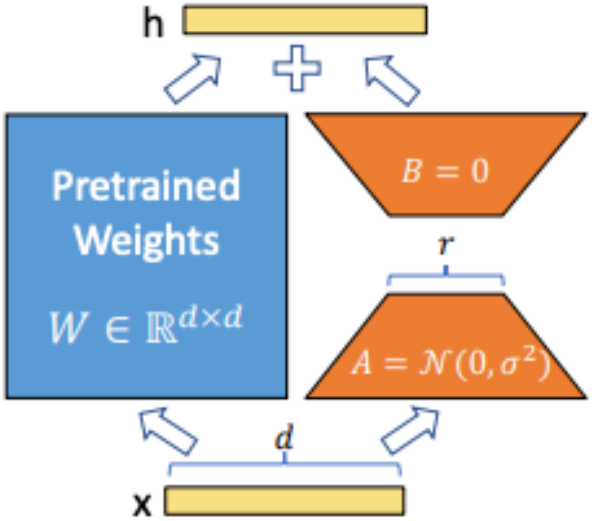


Figure 1: Our reparametrization. We only train A and B.