Advanced Deep Temporal Models

AI602: Recent Advances in Deep Learning

Lecture 3

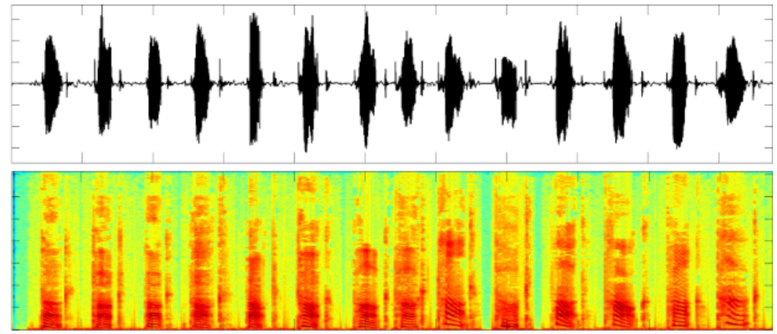
Slide made by

Jaehyung Kim and Changyeon Kim

KAIST EE

Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Speech

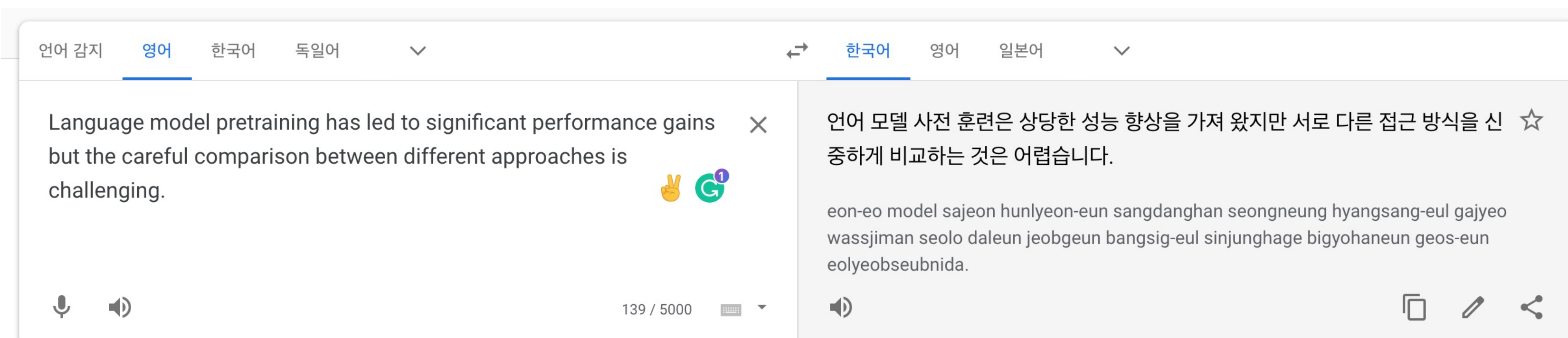


Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Speech
  - Natural language

"Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was __." ⟶ *terrible*

Language modeling



Translation

Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Speech
  - Natural language
  - Video

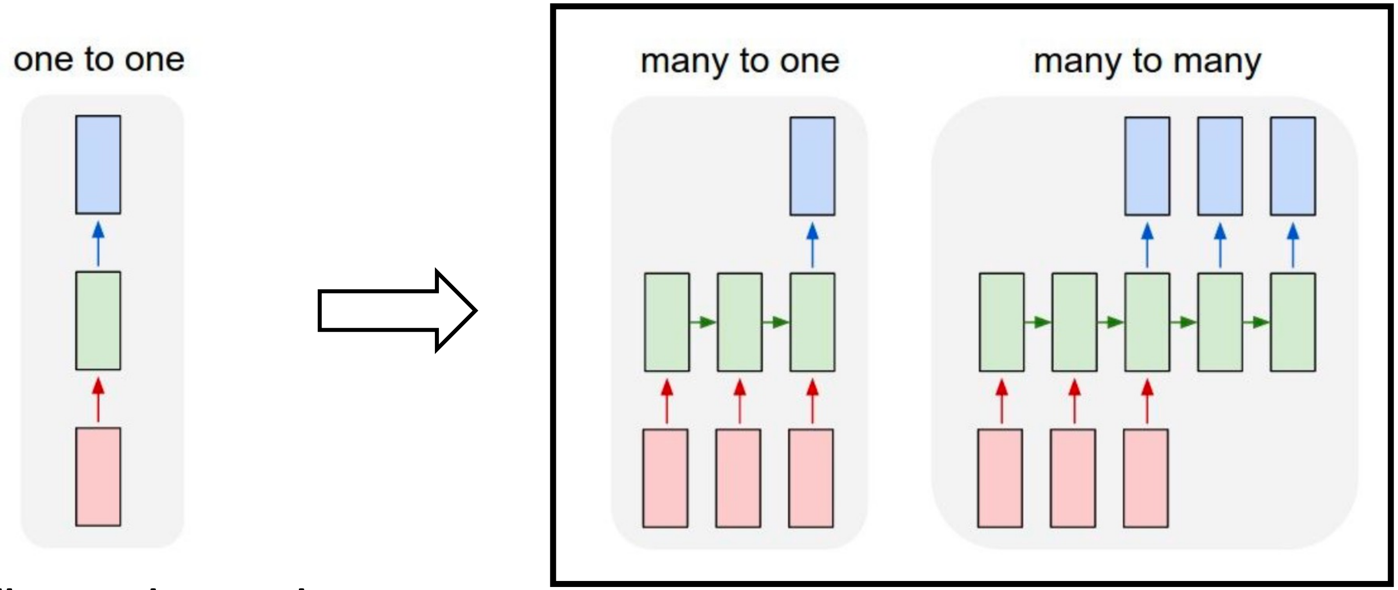
Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Speech
  - Natural language
  - Video
  - Stock prices, and etc…

Motivation: Temporal Data in Real World

- Many real-world data has a **temporal structure** intrinsically
  - Speech
  - Natural language
  - Video
  - Stock prices, and etc…

- In order to solve much complicated real-world problems,
  we need a **better architecture to capture temporal dependency** in the data



Vanilla neural network

Table of Contents

1. **Basics**
   - RNN (Recurrent Neural Networks)
   - LSTM (Long Short-Term Memory)
   - Sequence-to-sequence Model

2. **Advanced Topics**
   - From recurrence (RNN) to attention-based NLP models
   - Transformer (self-attention) with its great results
   - Pre-training of Transformers
   - Drawbacks and variants of Transformers

3. **Beyond GPT-3: Recent Advances with Large-scale Language Models**
   - Language models larger than GPT-3
   - More effective training schemes
   - Applications with language models

Table of Contents

- Process a sequence of vectors by applying **recurrence formula** at every time step :

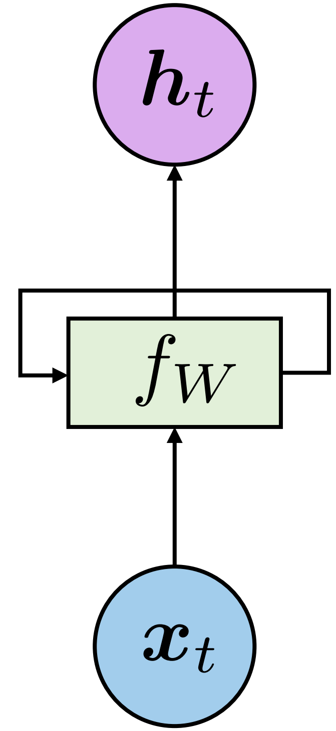
$$\boxed{\boldsymbol{h}_t} = \boxed{f_W}\left(\boxed{\boldsymbol{h}_{t-1}}, \boxed{\boldsymbol{x}_t}\right)$$

New state

Old state

Input vector at time step t

Function parameterized by learnable W



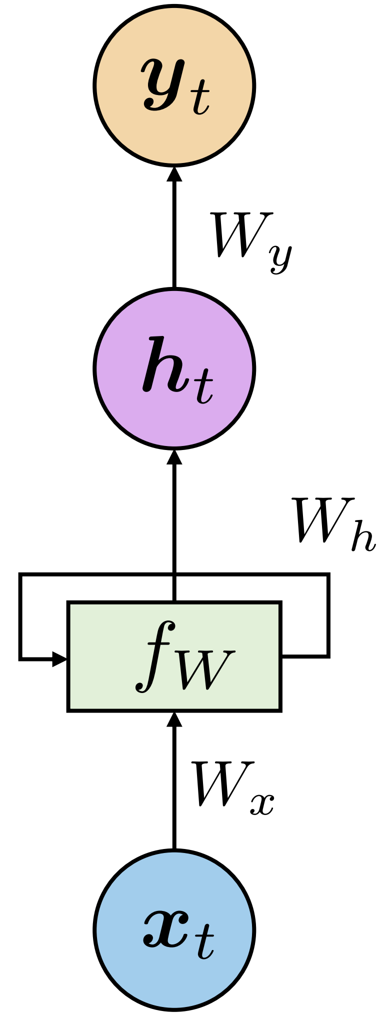
*reference: http://cs231n.stanford.edu/2017/

Vanilla RNN

- Vanilla RNN (or sometimes called **Elman RNN**)
  - The state consists of a single "hidden" vector $\mathbf{h}_t$

$$\boldsymbol{h}_t = f_W(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

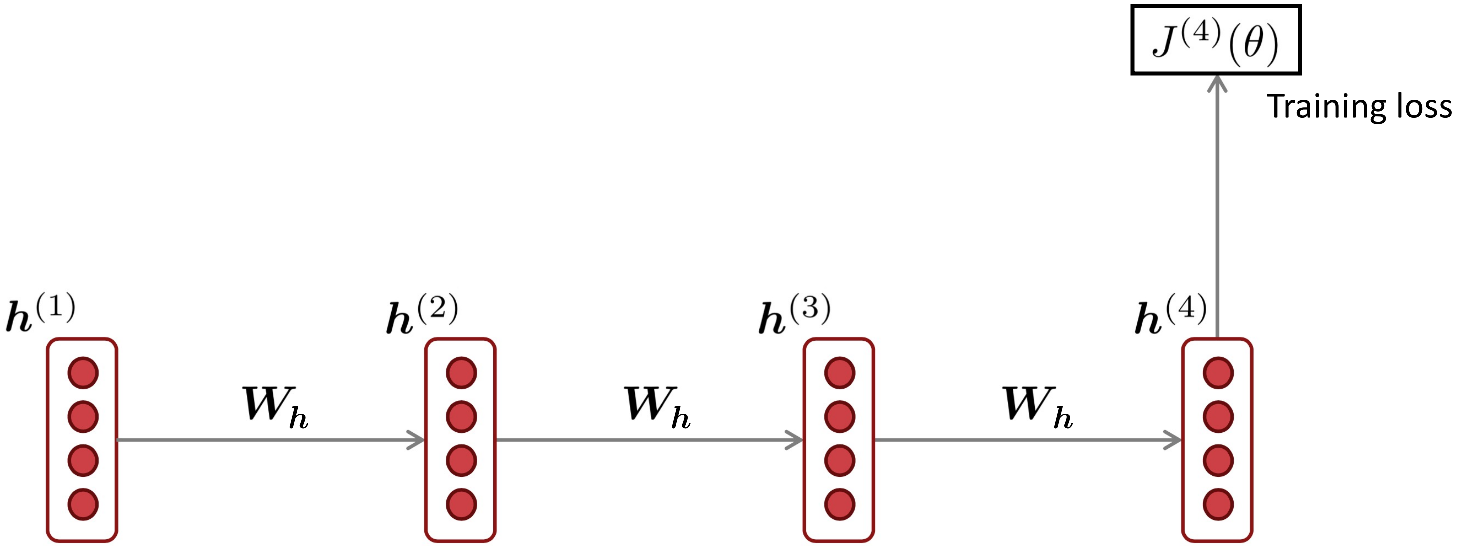
$$\boxed{\begin{aligned} \boldsymbol{h}_t &= \tanh(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x}_t) \\ \boldsymbol{y}_t &= W_y \boldsymbol{h}_t \end{aligned}}$$



*reference: http://cs231n.stanford.edu/2017/

Why Do We Need to Develop RNN Architectures?

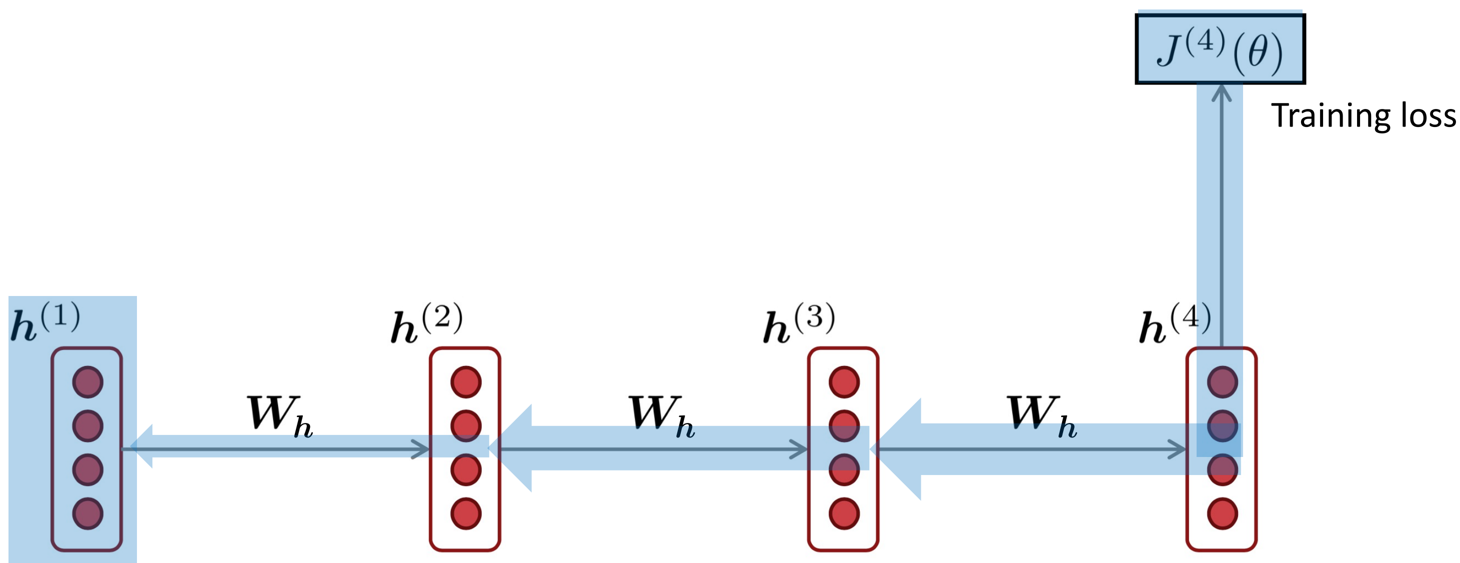
- E.g., RNN with a sequence of length 4



*reference: http://web.stanford.edu/class/cs224n/

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

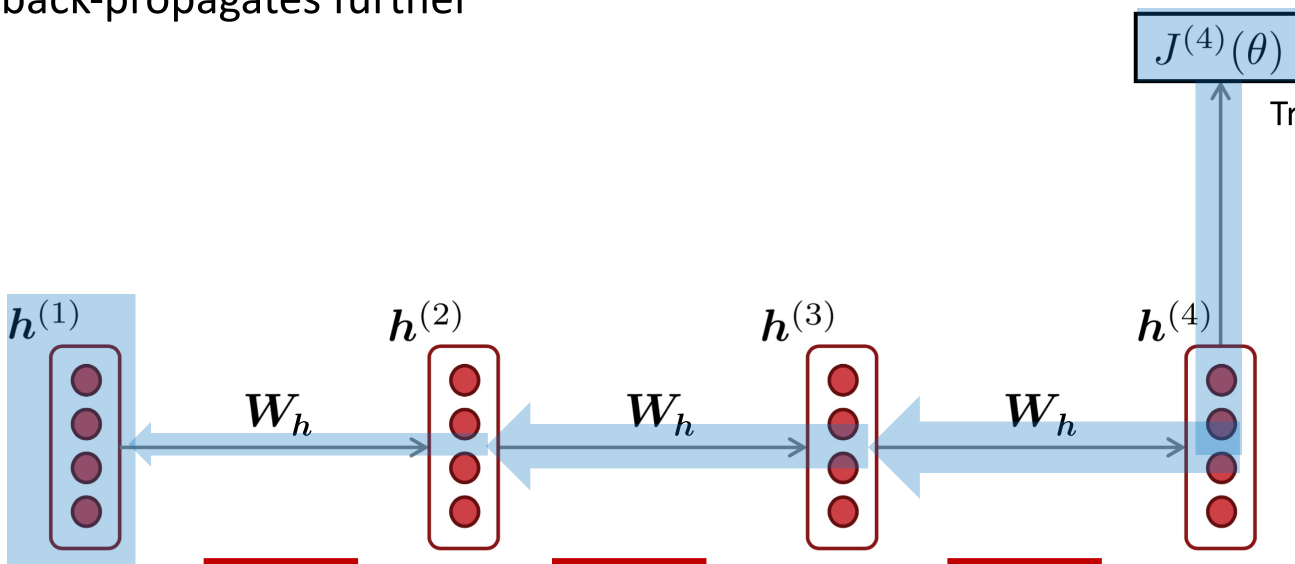
Chain rule!

*reference: http://web.stanford.edu/class/cs224n/

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? ⇨ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

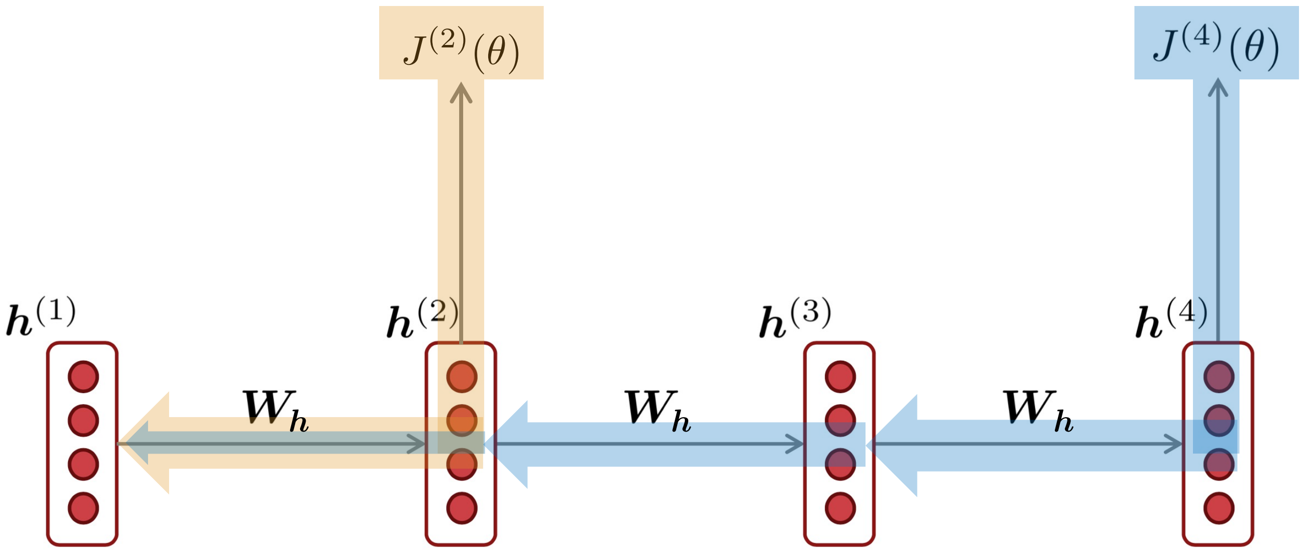
Chain rule!

*reference: http://web.stanford.edu/class/cs224n/

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? ⇨ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further
  - So, model weight are updated only with respect to near effects, **not** long-term effects.



*reference: http://web.stanford.edu/class/cs224n/

Why Do We Need to Develop RNN Architectures?

- E.g., RNN with a sequence of length 4
  - Consider a gradient from the first state $\boldsymbol{h}^{(1)}$

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too small**? $\Rightarrow$ **Vanishing gradient problem**
  - When these are small, the gradient signal gets smaller and smaller as it back-propagates further
  - So, model weight are updated only with respect to near effects, **not** long-term effects.

- What happens if $\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$ are **too large**? $\Rightarrow$ **Exploding gradient problem**

$$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_\theta J(\theta)$$

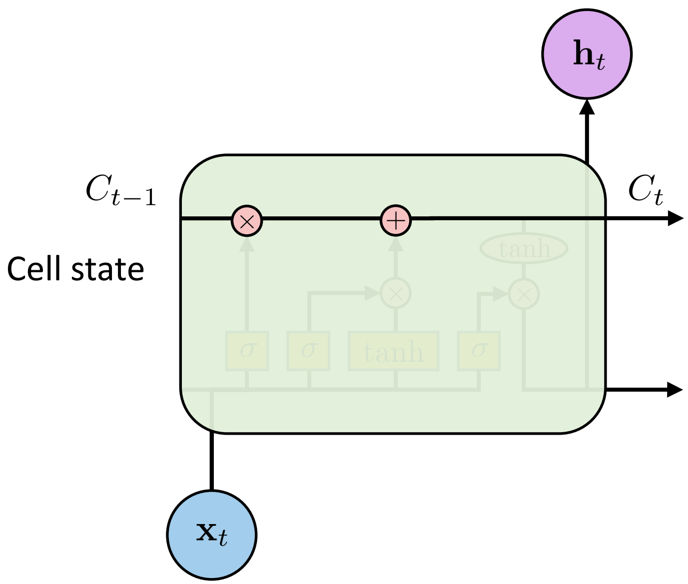
  - This can cause bad updates as the update step of parameters becomes too big
  - In the worst case, this will result in divergence of your network
  - In practice, with a gradient clipping, exploding gradient is **relatively easy to solve**

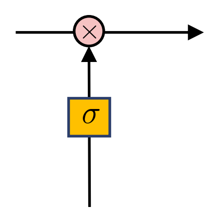
*reference: http://web.stanford.edu/class/cs224n/

RNN Architectures: LSTM

- **Long Short-Term Memory (LSTM)** [Hochreiter and Schmidhuber, 1997]
  - A special type of RNN unit, i.e., LSTM networks = RNN composed of LSTM units
  - Explicitly designed RNN to
    - Capture **long-term dependency** $\Longrightarrow$ more robust to vanishing gradient problem

- Core idea behind LSTM
  - With **cell state** (**memory**), it controls **how much to remove or add information**
    - Only linear interactions from the output of each "**gates**" (**prevent vanishing gradient**)



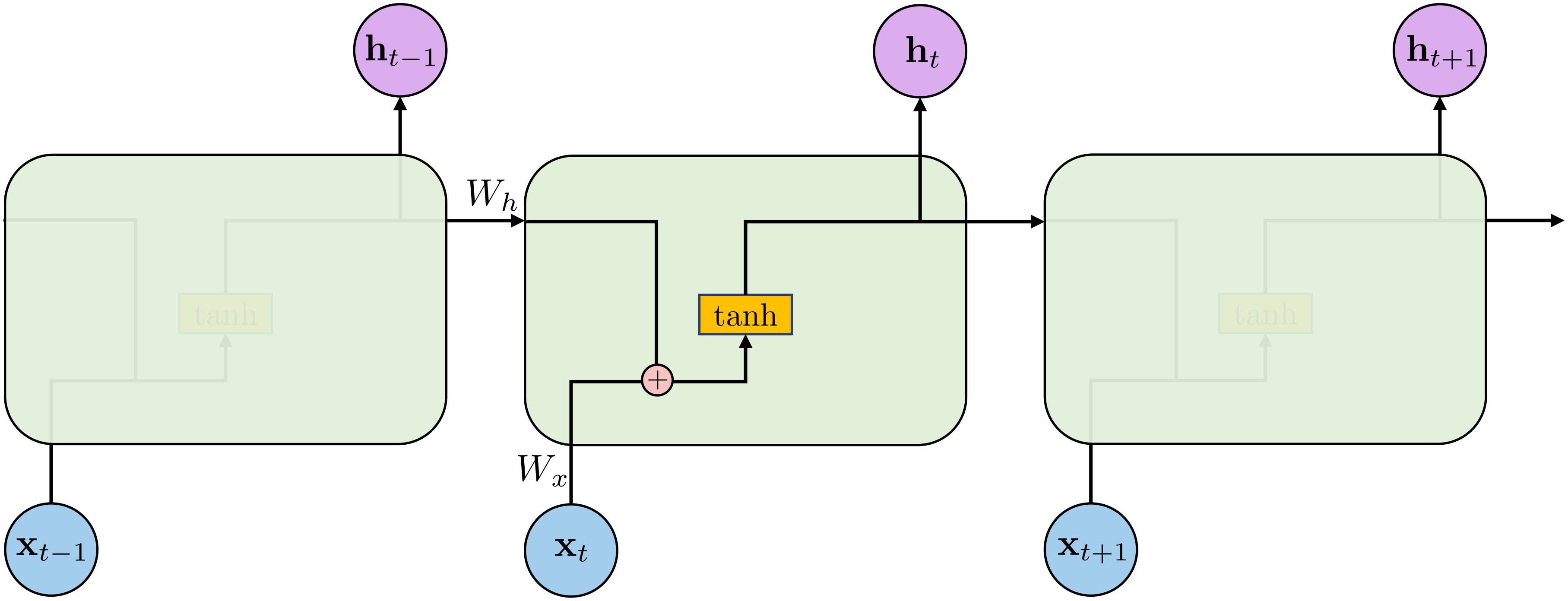


Gates : Way to optionally let information through

*source: https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png

- Repeating modules in **Vanilla RNN** contains a **single layer**

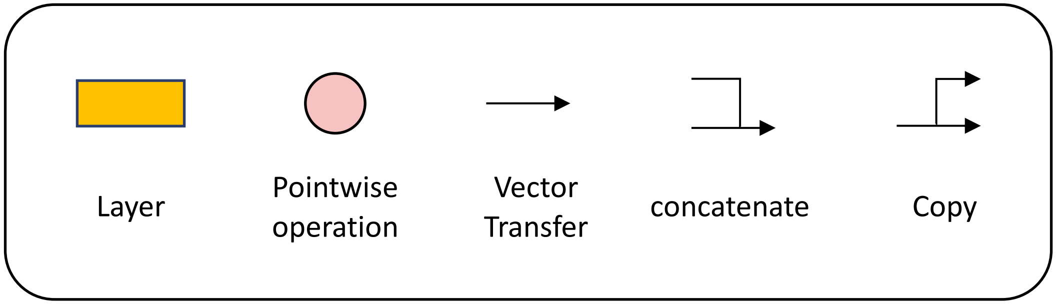
$$\boldsymbol{h}_t = \tanh(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x}_t)$$

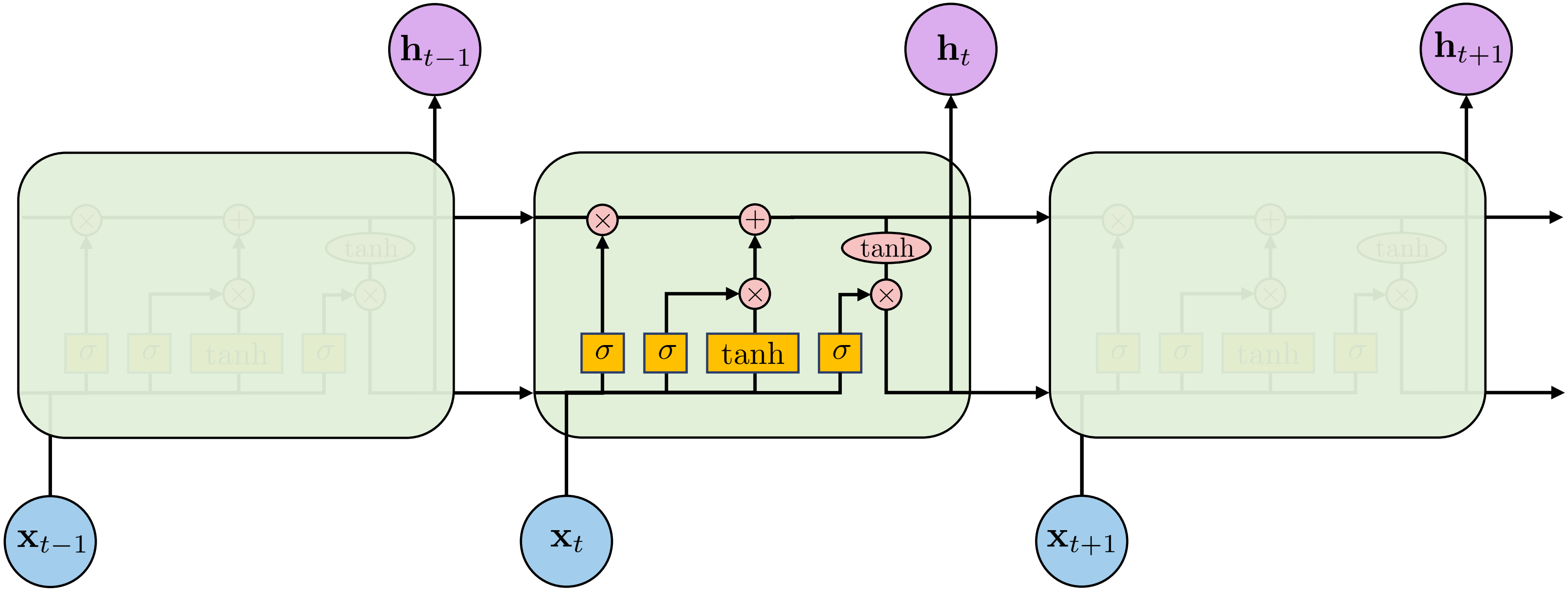


*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: LSTM

- Repeating modules in **LSTM**





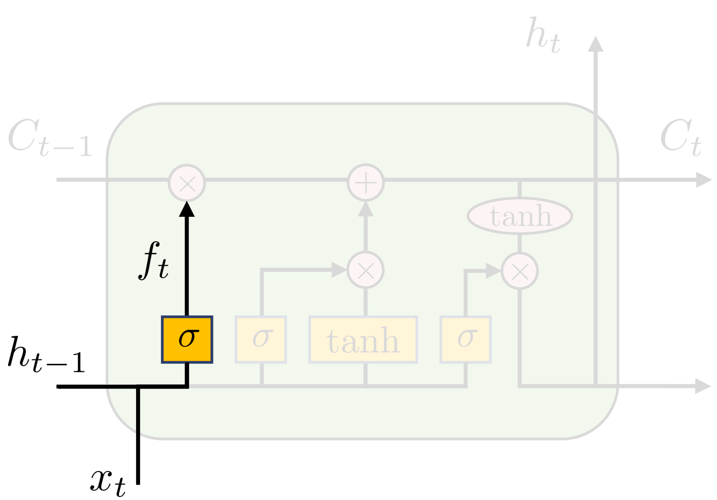
*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: LSTM

Step 1: Decide what **information** we're going to **throw away** from the **cell state**

- A sigmoid layer called "**Forget gate**" f_t
- Looks at h_{t-1}, x_t and outputs a number between 0 and 1 for each cell state C_{t-1}
  - If 1: completely keep, if 0: completely remove

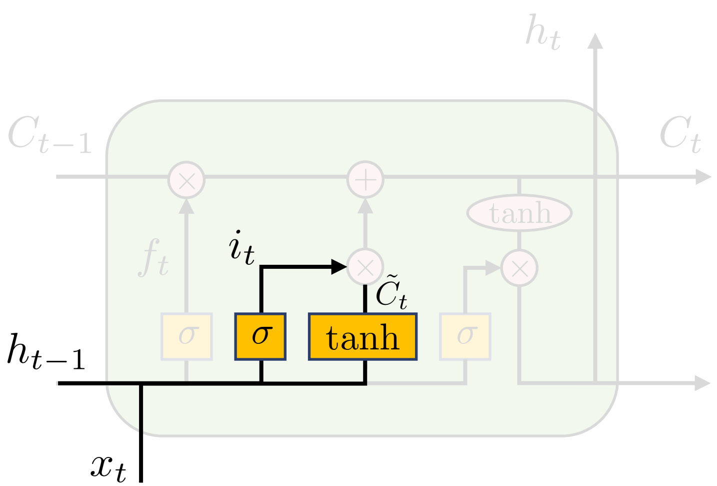
- E.g., language model trying to **predict the next word** based on all previous ones
  - The cell state might include the gender of the present subject so that the correct pronouns can be used
  - When we see a new subject, we want to forget the gender of the old subject



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Step 2: Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a **new content** $\tilde{C}_t$ to be written to the



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

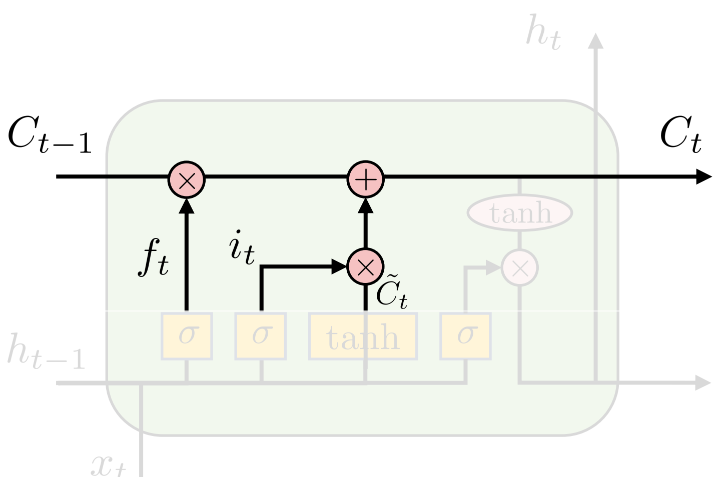
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 2: Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a **new content** $\tilde{C}_t$ to be written to the

- Then, **update** the old cell state C_{t-1} into the **new cell state** C_t
  - Multiply the old state by f_t (forget gate)
  - Add $i_t * \tilde{C}_t$, new content scaled by how much to update (input gate)



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

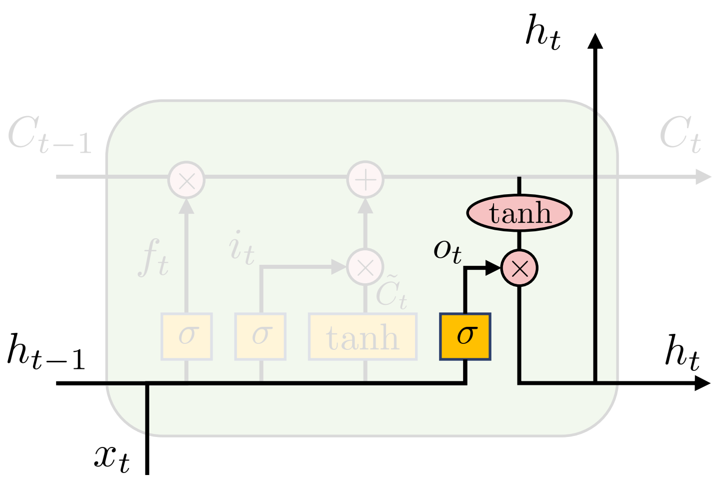
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$\boxed{C_t = f_t * C_{t-1} + i_t * \tilde{C}_t}$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 3: Decide what **information** we're going to **output**

- A sigmoid layer called "**Output gate**" o_t
- First, go through o_t which decides **what parts** of the cell state **to output**
- Then, put the cell state C_t through tanh and multiply it by o_t for hidden state h_t



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Overall LSTM operations

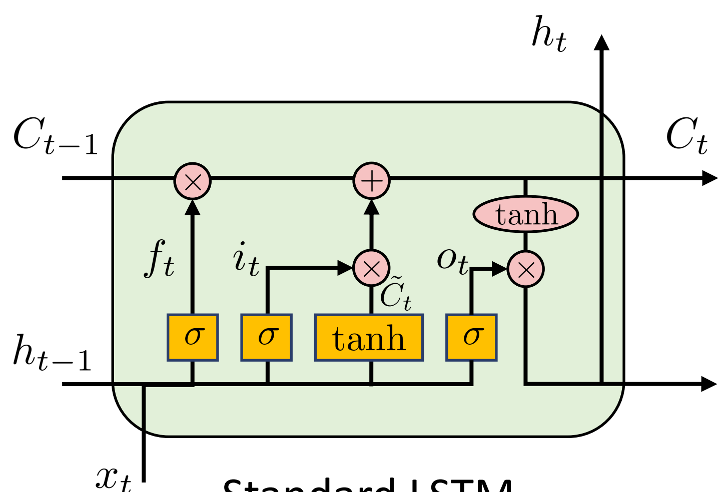
Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Previous cell state: C_{t-1} New cell content: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

Updated cell state: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

Hidden state: $h_t = o_t * \tanh(C_t)$



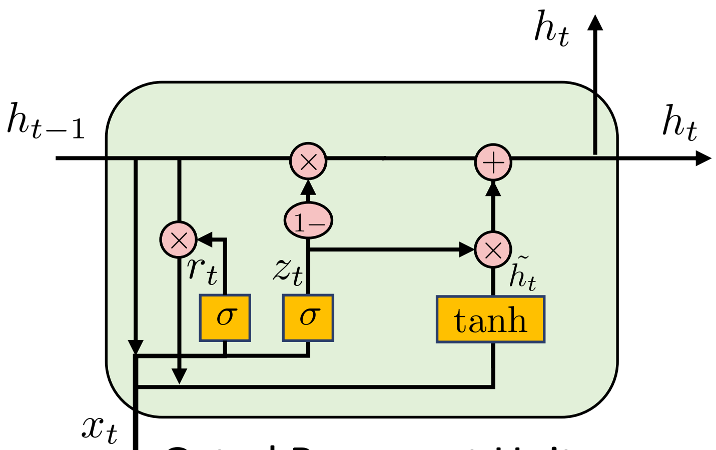
Standard LSTM

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- **Gated Recurrent Unit (GRU)** [Cho et.al, 2014]
  - Combines the forget and input gates into a single "**update gate**" z_t
    - Controls the **ratio of information to keep** between previous state and new state
  - **Reset gate** r_t controls how much information to forget when create a new content
  - **Merges** the cell state C_t and hidden state h_t
  - **(+)** Resulting in **simpler model (less weights)** than standard LSTM

Reset gate: $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$ New content: $\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$

Update gate: $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$ Hidden state: $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

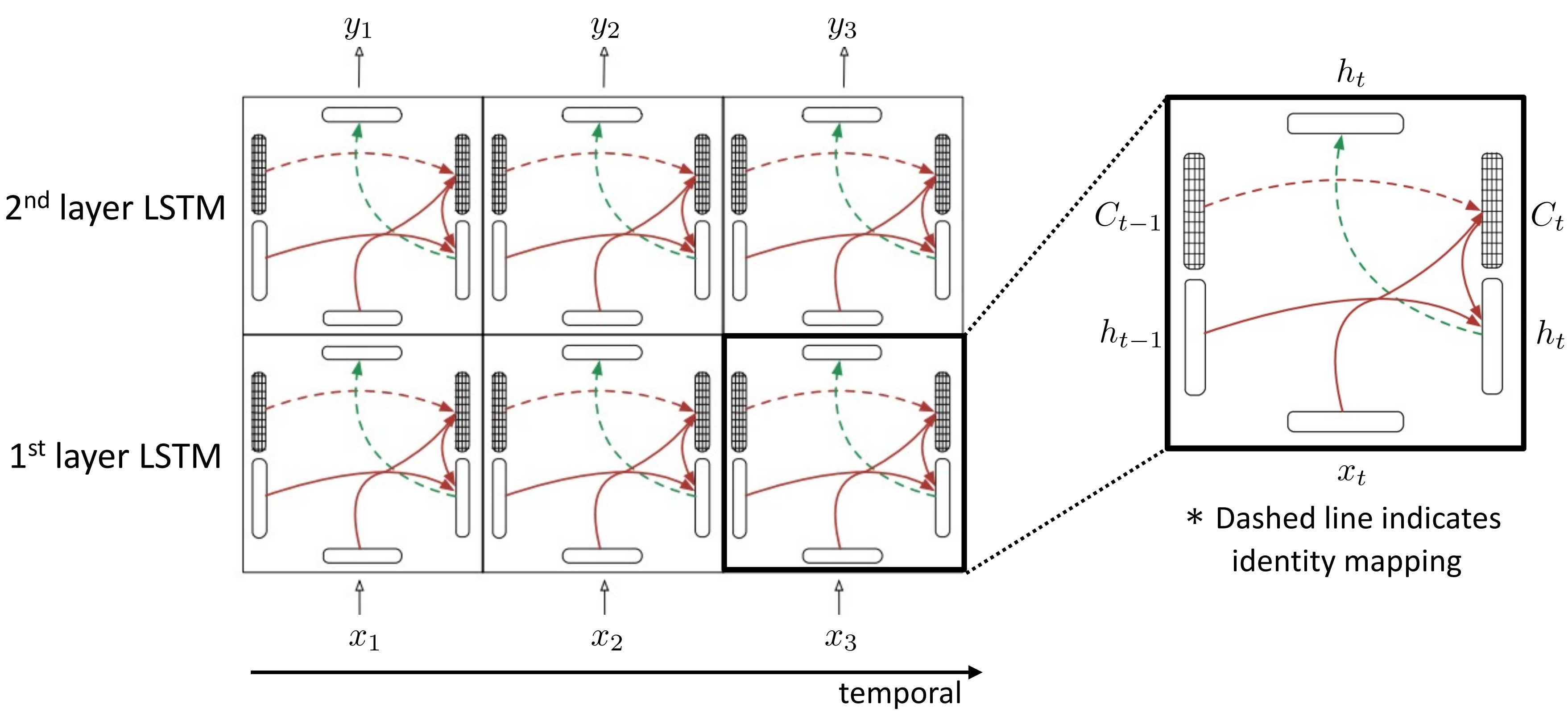


Gated Recurrent Unit

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: Stacked LSTM

- **Stacked(multi-layer) LSTM** [Graves et al, 2013]
  - RNNs are already "deep" on one dimension (they unroll over many time-steps)
  - We can add depth by simply stacking LSTM layers on top of each other
    - This allows the network to compute **more complex representations**
  - E.g., Output of 1st layer LSTM goes into 2nd layer LSTM as an input

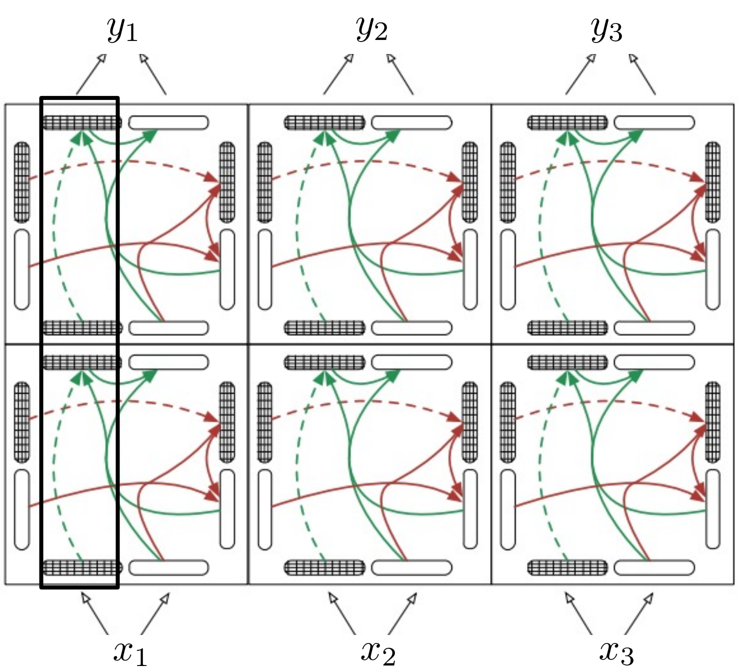


∗ Dashed line indicates identity mapping

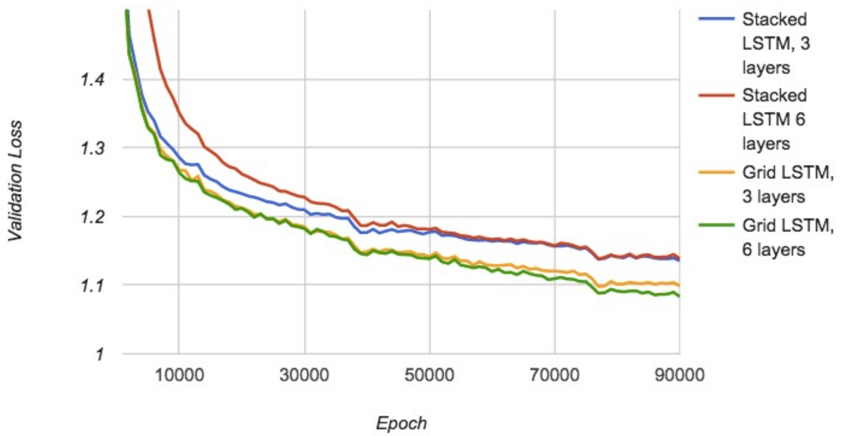
*source: https://arxiv.org/pdf/1507.01526.pdf

RNN Architectures: Grid LSTM

- **Grid LSTM** [Kalchbrenner et al., 2016]
  - Extended version of stacked LSTM
  - LSTM units have **additional memory along depth dimension** as well as temporal dimension



2D Grid LSTM



	BPC	Parameters	Alphabet Size	Test data
Stacked LSTM (Graves, 2013)	1.67	27M	205	last 4MB
MRNN (Sutskever et al., 2011)	1.60	4.9M	86	last 10MB
GFRNN (Chung et al., 2015)	1.58	20M	205	last 5MB
Tied 2-LSTM	**1.47**	16.8M	205	last 5MB

Performance on wikipedia dataset
(lower the better)

*source: https://github.com/coreylynch/grid-lstm

Limitation of Left-to-Right RNNs

- What is the limitation of all previous models?
  - They learn representations only from **previous** time steps (left-to-right)
  - But, it's sometimes useful to learn from **future** time steps in order to
    - Better understand the **context**
    - Eliminate ambiguity

- Example
  - "**He said, Teddy** bears are on sale"
  - "**He said, Teddy** Roosevelt was a great President"
  - In above two sentences, only seeing **previous words is not enough** to understand the sentence

- Solution
  - Also look ahead (right-to-left) ⇨ $\boxed{\textbf{Bidirectional RNN}}$

RNN Architectures: Bidirectional RNNs

- RNNs can be easily extended into **bi-directional models**
  - Only difference is that there are **additional paths from future** time steps
    - Any types of RNNs (Vanilla RNN, LSTM, or GRU) could be bi-directional models
  - **Note**: bi-directional RNNs are only applicable if one has access to entire sequence

Hidden states

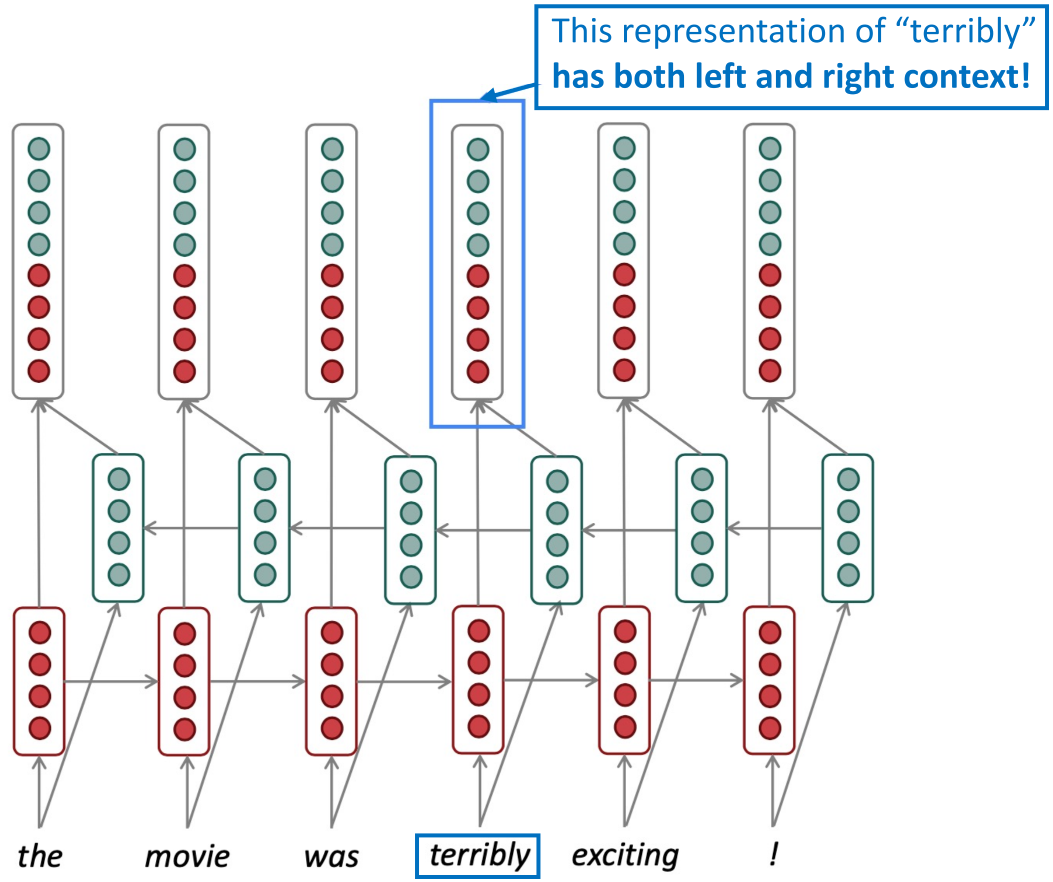
$$h^{(t)} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}]$$

Backward RNN

$$\overleftarrow{h}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}\left(\overleftarrow{h}^{(t+1)}, x^{(t)}\right)$$

Forward RNN

$$\overrightarrow{h}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}\left(\overrightarrow{h}^{(t-1)}, x^{(t)}\right)$$



*reference: http://web.stanford.edu/class/cs224n/

Table of Contents

RNNs in Real-world Application: Neural Machine Translation

- What is machine translation (MT)?
  - Task of automatically **converting** source **text in one language to another** language
  - **No single answer** due to ambiguity/flexibility of human language (**challenging**)

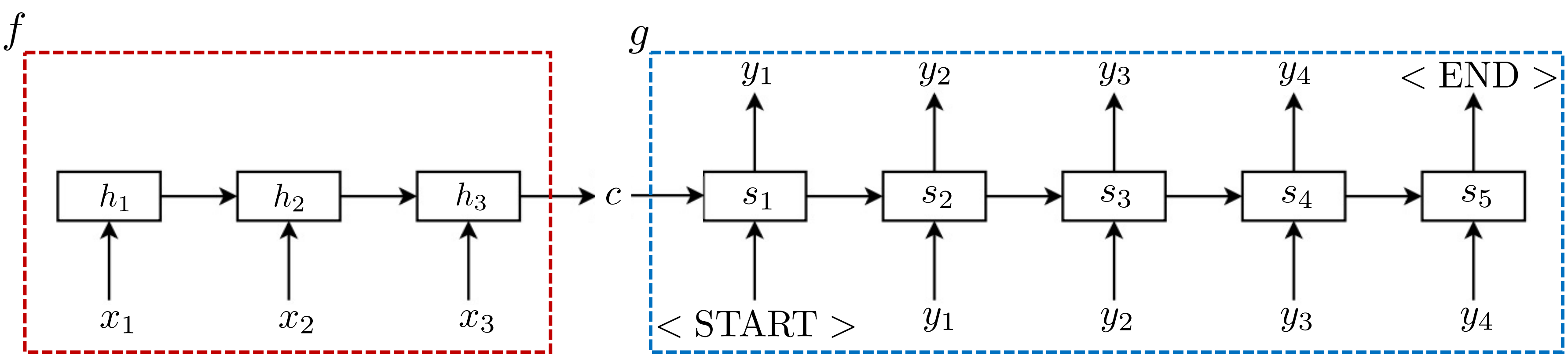
- Classical machine translation methods
  - Rule-based machine translation (RBMT)
  - Statistical machine translation (SMT; use of statistical model)
  - **(-)** Lots of human effort to maintain, e.g., repeated effort for each language pair

- Neural Machine Translation (NMT)
  - Use of **neural network models to learn a statistical model** for machine translation

Breakthroughs in NMT: Sequence-to-Sequence Learning

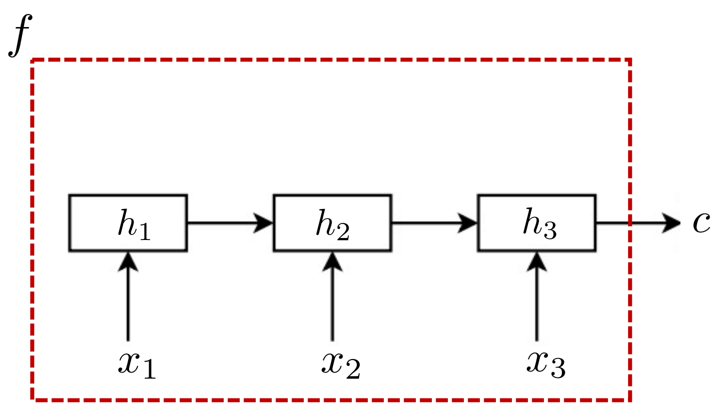
- **Difficulties** in Neural Machine Translation
  - Intrinsic difficulties of MT (ambiguity of language)
  - Variable length of input and output sequence (difficult to learn a single model)

- The core idea of **sequence-to-sequence** model [Sutskever et al., 2014]
  - **Encoder-Decoder** architecture (input → vector → output)
  - Use one RNN network (**Encoder**) to **read input sequence** at a time for **encoding it into** a fixed-length vector representation **(context)**
  - Use another RNN (**Decoder**) to extract the **output sequence** from context vector



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

- **Encoder**
  - **Reads the input** sentence $\mathbf{x} = (x_1, \ldots, x_T)$ and **output context** vector c
  - Use RNNs such that $h_t = f(x_t, h_{t-1})$ and $c = q(\{h_1, \ldots, h_T\})$, where f and q are some non-linear functions
  - E.g., LSTMs as f and $q(\{h_1, \ldots, h_T\}) = h_T$ (in the original seq2seq model)



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

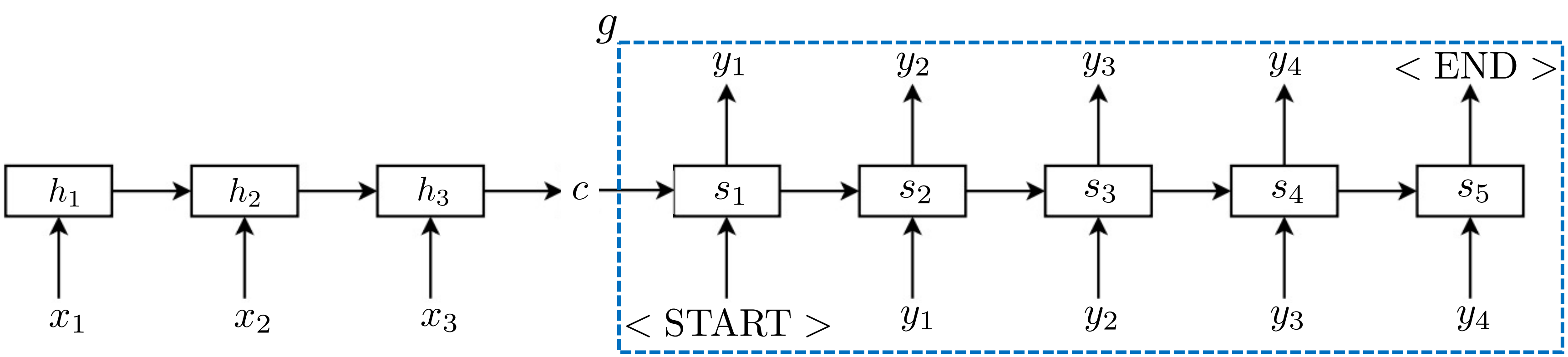
- **Decoder**
  - **Predict the next word** $y_{t'}$ **given the context** vector c and the **previously predicted** words $\{y_1, \ldots, y_{t'-1}\}$
  - Defines a probability over the translation $\mathbf{y}$ by **decomposing the joint probability** into the ordered conditionals where $\mathbf{y} = (y_1, \ldots, y_T)$.

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \ldots, y_{t'-1}\}, c),$$

  - The conditional probability is modeled with **another RNN** g as

$$p(y_t | \{y_1, \ldots, y_{t'-1}\}, c) = g(y_{t-1}, \underline{s_t}, c),$$

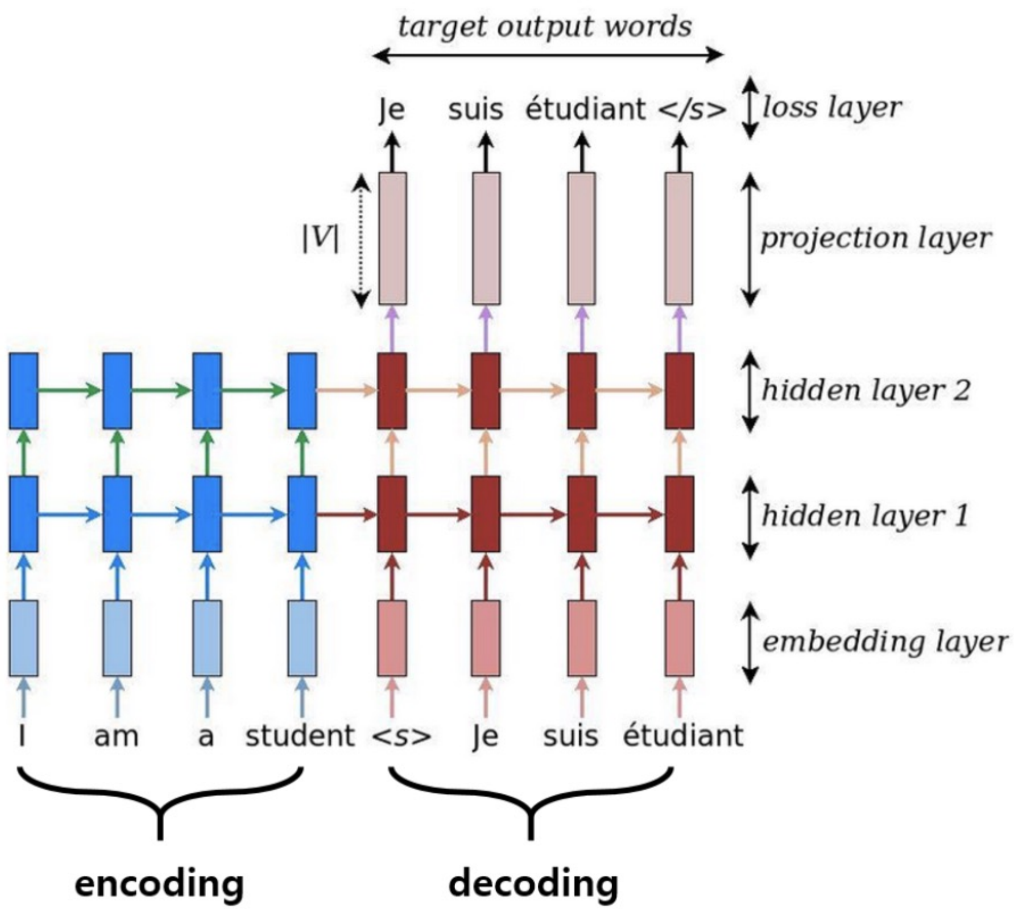
hidden state of the RNN



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Example of the seq2seq model
  - For English → French task
  - With 2-layer LSTM for encoder and encoder



*source: https://towardsdatascience.com/seq2seq-model-in-tensorflow-ec0c557e560f

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Results on WMT'14 English to French dataset [Sutskever et al., 2014]
  - Measure : BLEU(Bilingual Evaluation Understudy) score
    - Widely used quantitative measure for MT task
  - On **par with the state-of-the-art SMT** system (without using neural network)
  - Achieved **better results than the previous baselines**

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	$\sim$45

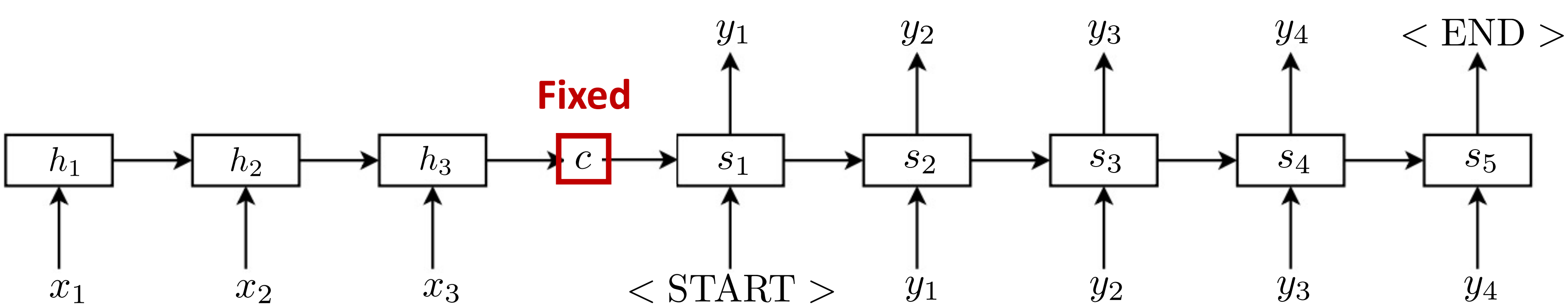
- Seq2seq with RNNs is **simple but very powerful** in MT task

*source: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning

- Problem of original seq2seq(or encoder-decoder) model
  - Need to **compress** all the necessary information of a source sentence into a **fixed context vector**
    - All decoding steps use an identical context along with previous outputs

$$p(y_t|\{y_1, \ldots, y_{t'-1}\}, c) = g(y_{t-1}, s_t, \underline{c}),$$

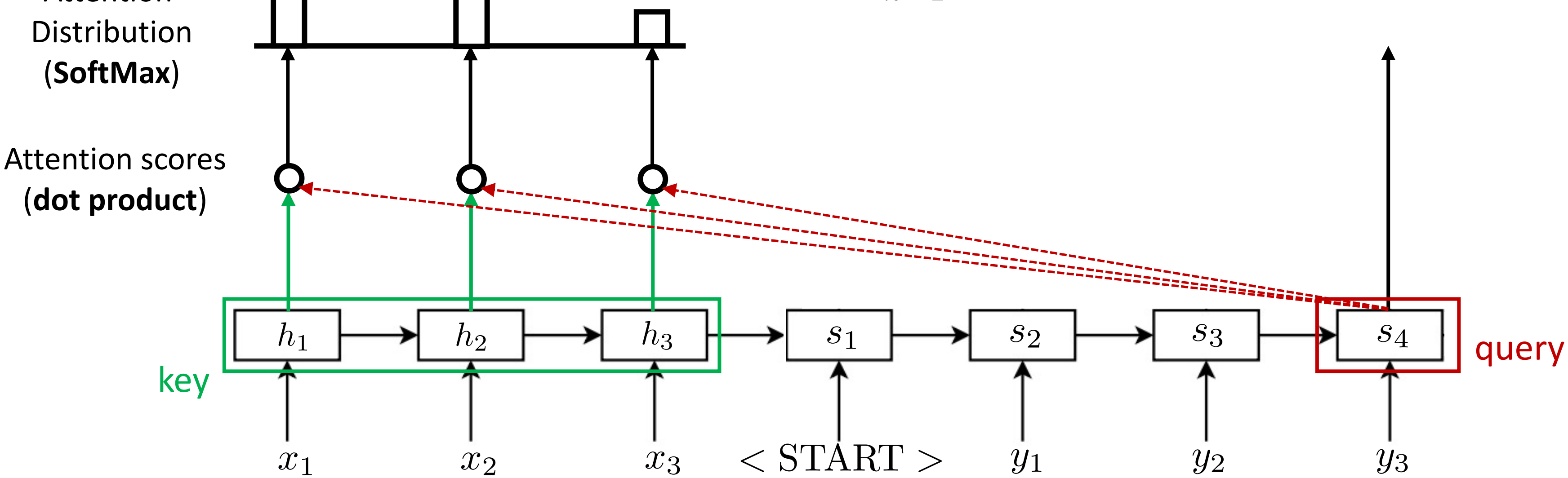
  - But, each step of decoding **requires different part** of the source sequence
    - E.g., Step1: “**I** love you” ⟶ “**나는** 너를 사랑해”
      Step2: “I **love** you” ⟶ “나는 너를 **사랑해**”
    - Hence, difficult to cope with long sentences…



Input sequence $\boldsymbol{x} = (x_1, x_2, x_3)$ and output sequence $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$

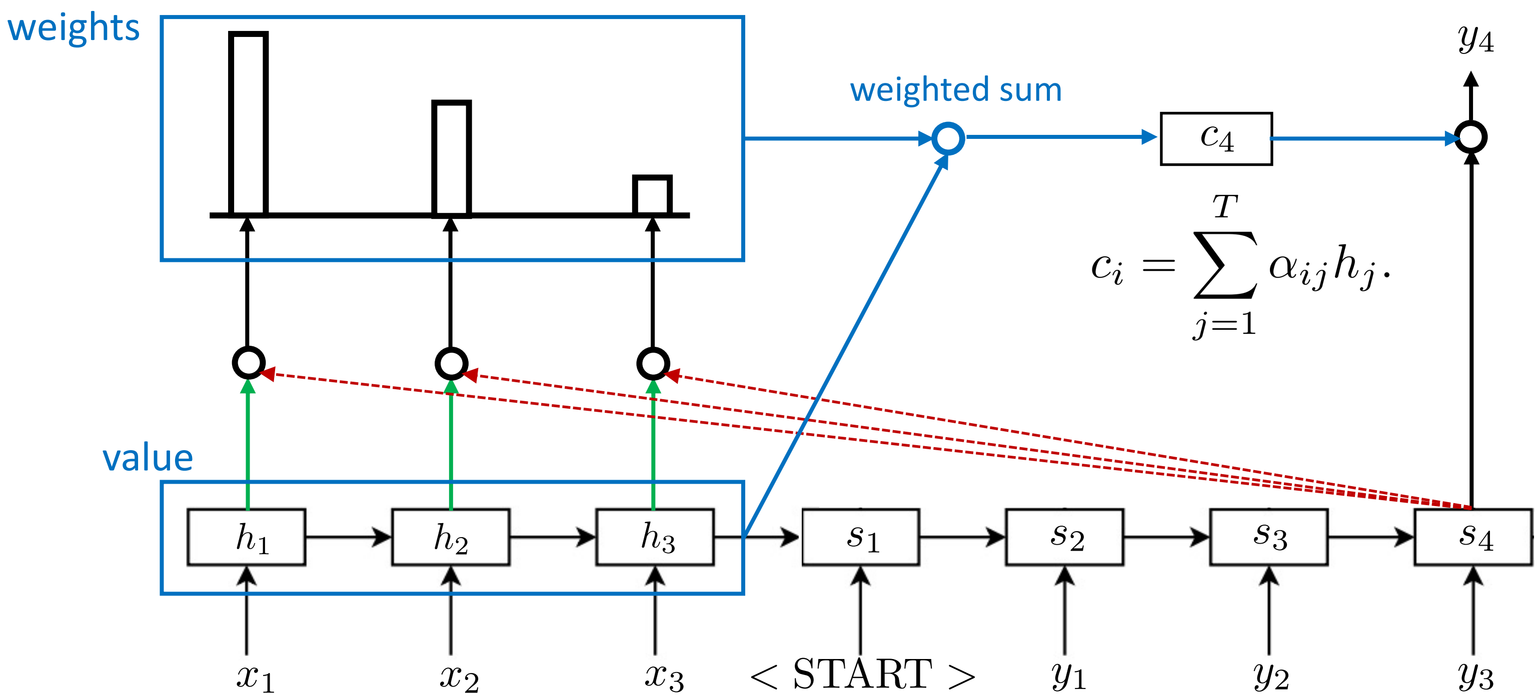
- Extension of seq2seq model with **attention** mechanism [Bahdanau et al., 2015]
  - **Core idea**: on each step of the decoder, **focus on a particular part** of the source sequence using a **direct connection (attention)** to the encoder states
  - Dependent on the query with key, **attention** is a technique to compute a weighted sum of the values
    - Query: decoder's hidden state, key and value: encoder's hidden states
    - α_{ij} is a **relative importance** which means how well the inputs around position i and the output position j match.

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}, \quad e_{ij} = s_{i-1}^{T} h_j$$



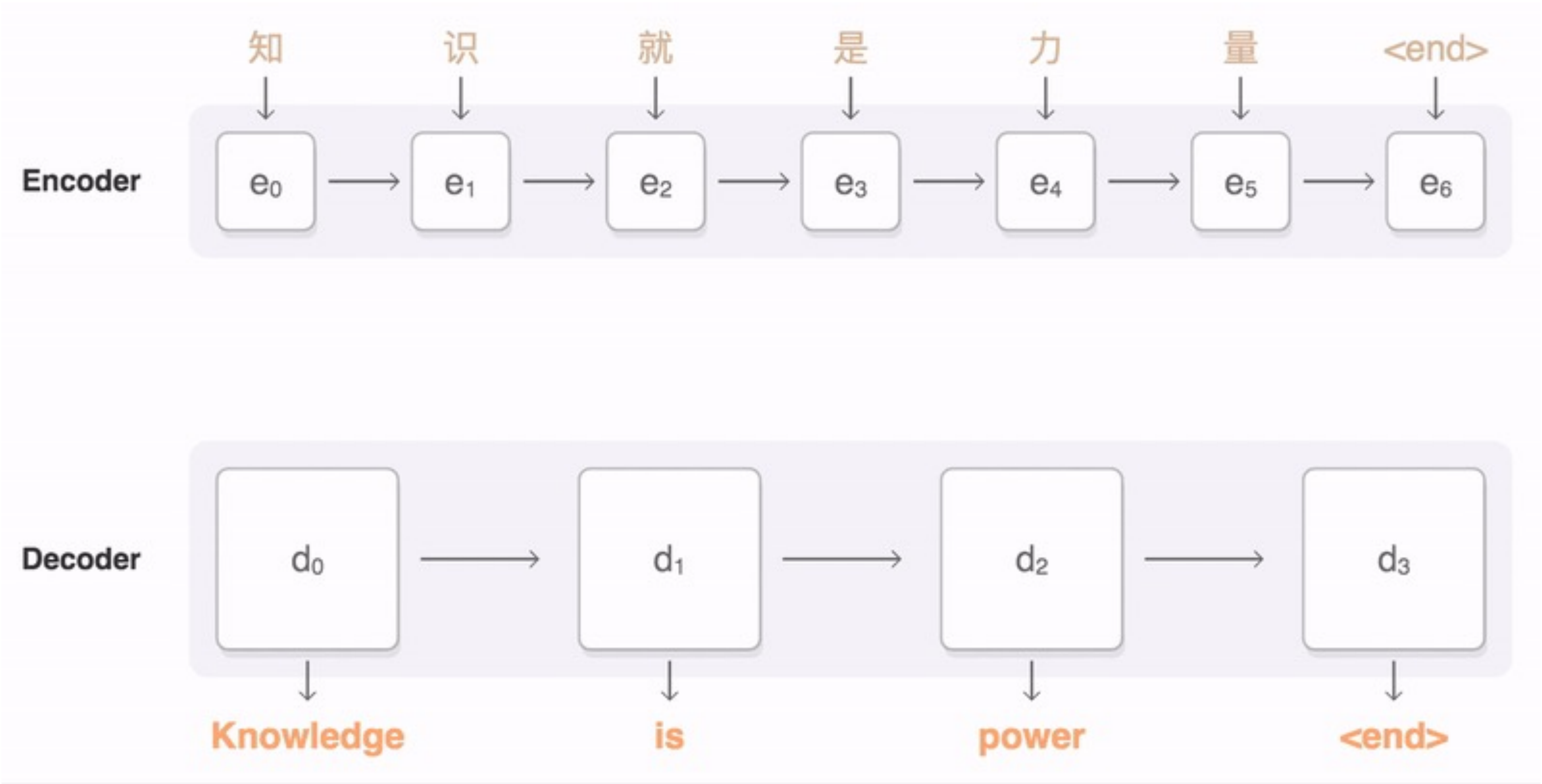
Breakthroughs in NMT: Sequence-to-Sequence Model with Attention

- Extension of seq2seq model with **attention** mechanism [Bahdanau et al., 2015]
  - **Core idea**: on each step of the decoder, **focus on a particular part** of the source sequence using a **direct connection (attention)** to the encoder states
  - Dependent on the query with key, **attention** is a technique to compute a weighted sum of the values
    - Query: decoder's hidden state, key and value: encoder's hidden states
    - The context vector c_i is computed as **weighted sum** of h_i



Breakthroughs in NMT: Sequence-to-Sequence Model with Attention

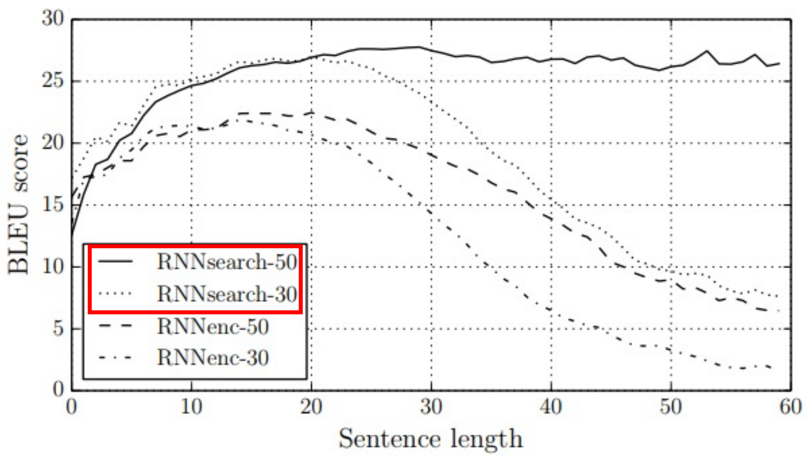
- **Graphical illustration** of seq2seq with **attention**
  - E.g., Chinese to English

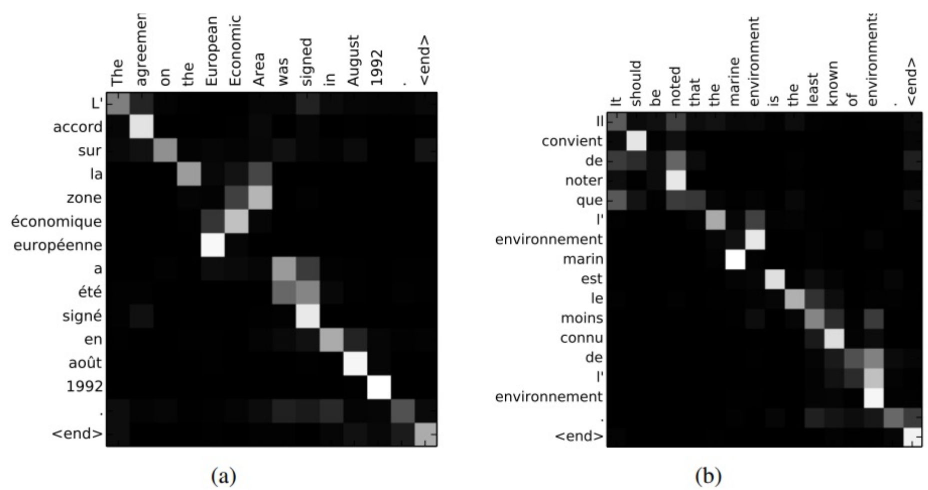


*source: https://google.github.io/seq2seq/

Breakthroughs in NMT: Sequence-to-Sequence Model with Attention

- **Results**
  - RNNsearch (with attention) is better than RNNenc (vanilla seq2seq)
  - RNNsearch-50: model trained with sentences of length up to 50 words

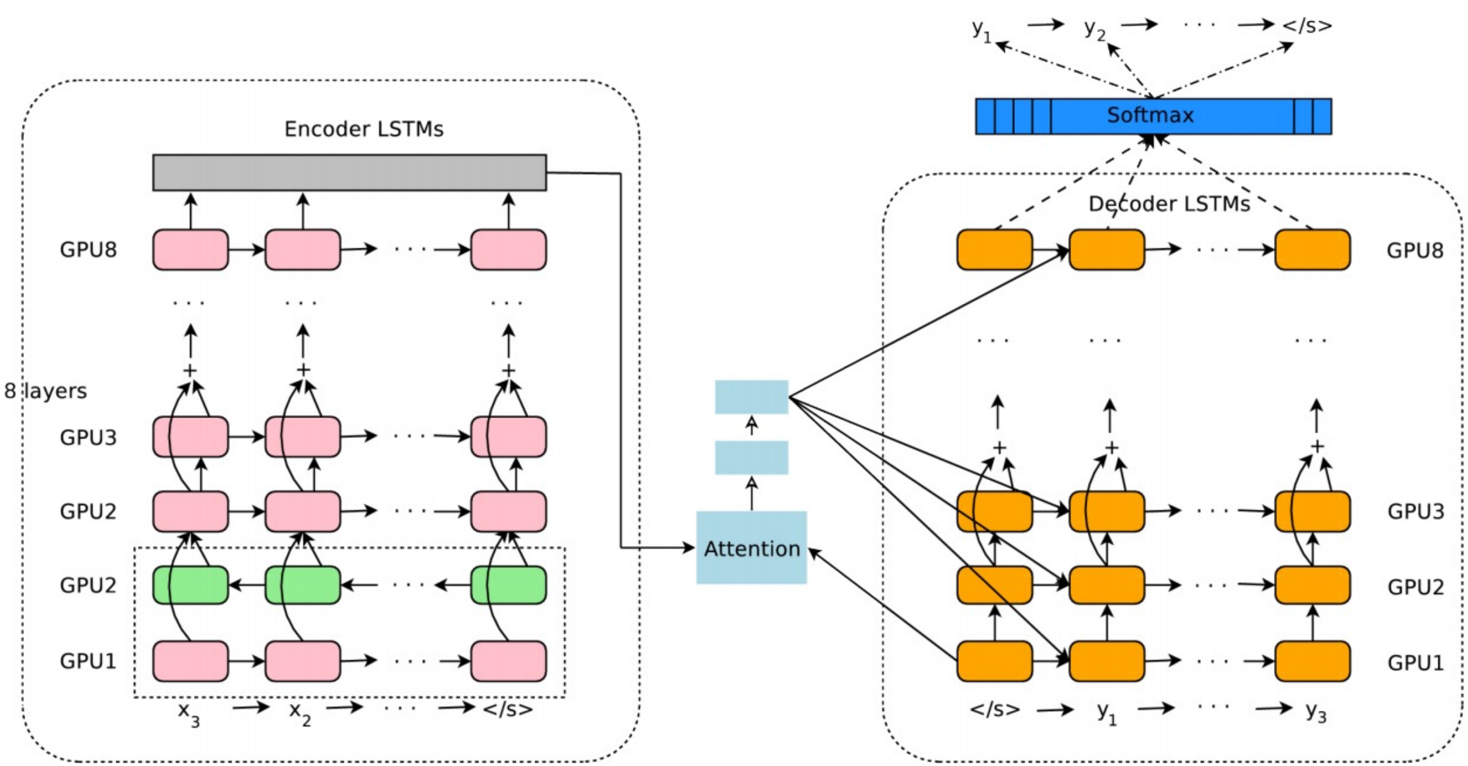




Sample alignment results (attention map)

Google's Neural Machine Translation (GNMT)

- **Google's NMT** [Wu et al., 2016]
  - Improves over previous NMT systems on **accuracy** and **speed**
  - **8-layer LSTMS** for encoder/decoder with **attention**
    - Achieve **model parallelism** by assigning each LSTM layer into different GPUs
    - Add **residual connections in standard LSTM**
    - … and lots of domain-specific details to apply it to production model



Google's Neural Machine Translation (GNMT)

- **Google's NMT** [Wu et al., 2016]
  - Improves over previous NMT systems on **accuracy** and **speed**
  - **8-layer LSTMS** for encoder/decoder with **attention**
  - State-of-the-art results on various MT datasets and comparable with Human expert

Table 5: Single model results on WMT En→De (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	23.12	0.2972
Character (512 nodes)	22.62	0.8011
WPM-8K	23.50	0.2079
WPM-16K	24.36	0.1931
WPM-32K	24.61	0.1882
Mixed Word/Character	24.17	0.3268
PBMT [6]	20.7	
RNNSearch [37]	16.5	
RNNSearch-LV [37]	16.9	
RNNSearch-LV [37]	16.9	
Deep-Att [45]	20.6	

Table 10: Mean of side-by-side scores on production data

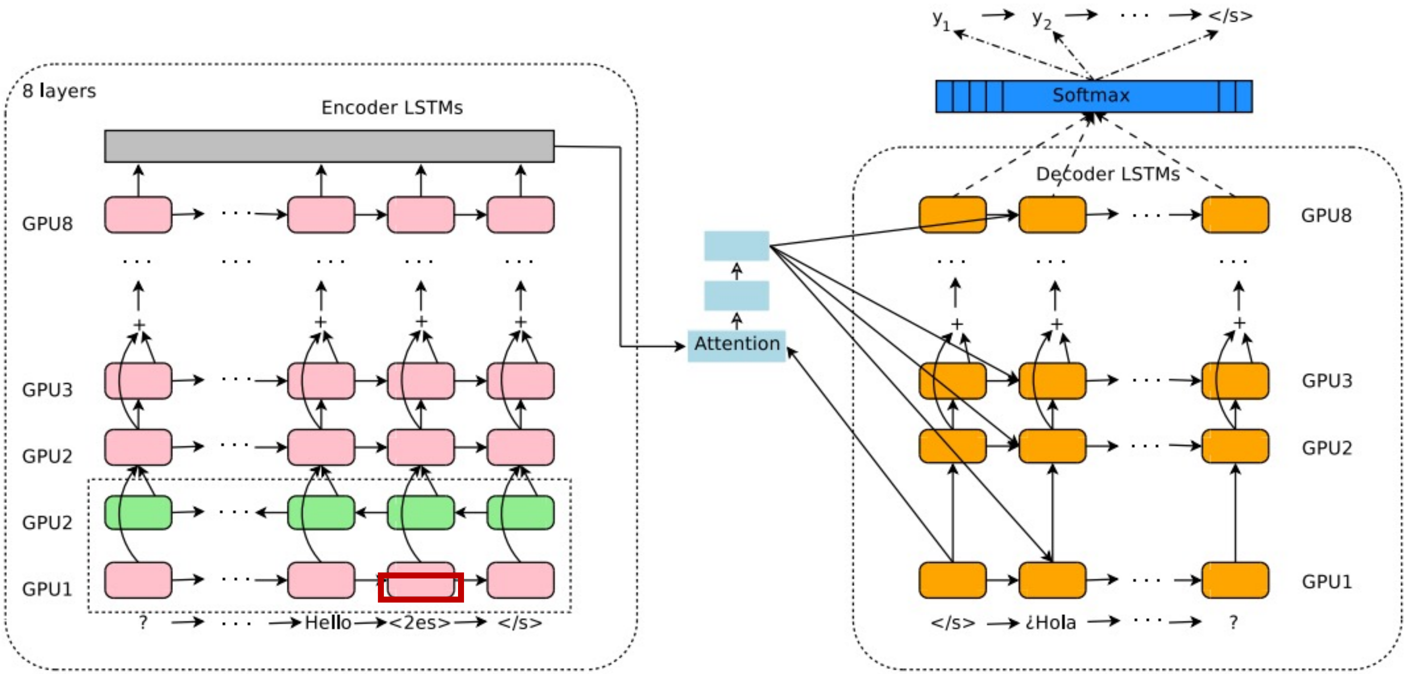
	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

GNMT with different configurations

Google's Multilingual Neural Machine Translation (Multilingual GNMT)

- Google's NMT is further improved in [Johnson et al., 2016]

- Extensions to make this model to be **Multilingual NMT** system by adding **artificial token** to indicate the required **target language**
  - E.g., the token "<2es>" indicates that the target sentence is in Spanish
  - Can do multilingual NMT using a **single model w/o increasing the parameters**



Google's Multilingual Neural Machine Translation (Multilingual GNMT)

- Google's NMT is further improved in [Johnson et al., 2016]

- Extensions to make this model to be **Multilingual NMT** system by adding **artificial token** to indicate the required **target language**
  - E.g., the token "<2es>" indicates that the target sentence is in Spanish
  - Can do multilingual NMT using a **single model w/o increasing the parameters**

- **Summary**
  - **2014**: First seq2seq paper published
  - **2016**: Google Translate switches from SMT to NMT – and by **2018 everyone has**
  - **Remark**. **SMT** systems, built by hundreds of engineers over many years, outperformed by **NMT** systems trained by a small group of engineers in a few months

*reference: http://web.stanford.edu/class/cs224n/

Google's Multilingual Neural Machine Translation (Multilingual GNMT)

- Google's NMT is further improved in [Johnson et al., 2016]

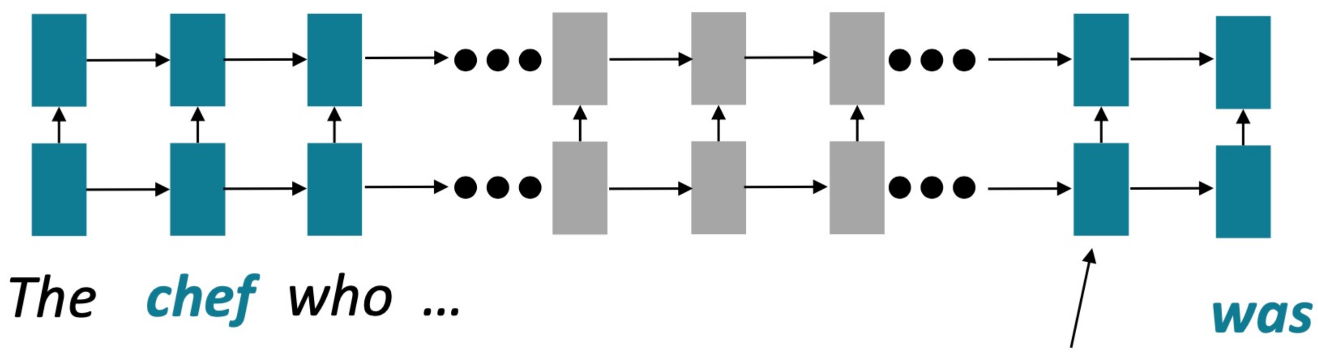
- Extensions to make this model to be **Multilingual NMT** system by adding **artificial token** to indicate the required **target language**
  - E.g., the token "<2es>" indicates that the target sentence is in Spanish
  - Can do multilingual NMT using a **single model w/o increasing the parameters**

- **Next**
  - **Now (2021)**, other approaches have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
    - In WMT **2016**, the summary report contains "**RNN**" **44** times
    - In WMT **2019**: "RNN" 7 times, "**Transformer**" **105** times

Next, Transformer (self-attention)

Issue with Recurrent Models

- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
     - Hard to learn long-distance dependencies because of gradient problems
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations
     - Future RNN hidden states can't be computed before past states have been computed
     - This aspect inhibits training on the very large datasets

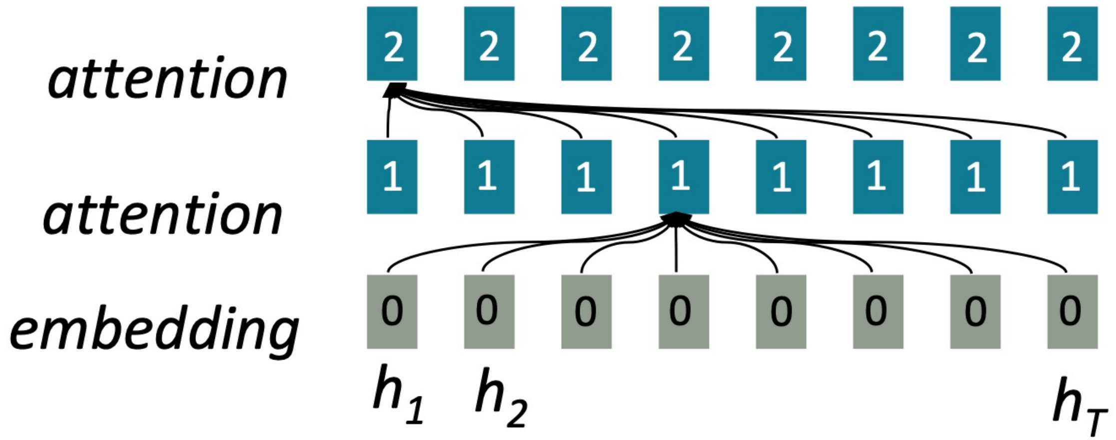


Info of **chef** has gone through **O(sequence length)** many layers

Issue with Recurrent Models

- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations

- In contrast, **attention has some advantages** in these aspects:
  1. Maximum interaction distance: **O(1)**
     - Since all words interact at each layer
  2. Number of unparallelizable operations does **not increase with respect to length**



All words can attend to all words in previous layer

Issue with Recurrent Models

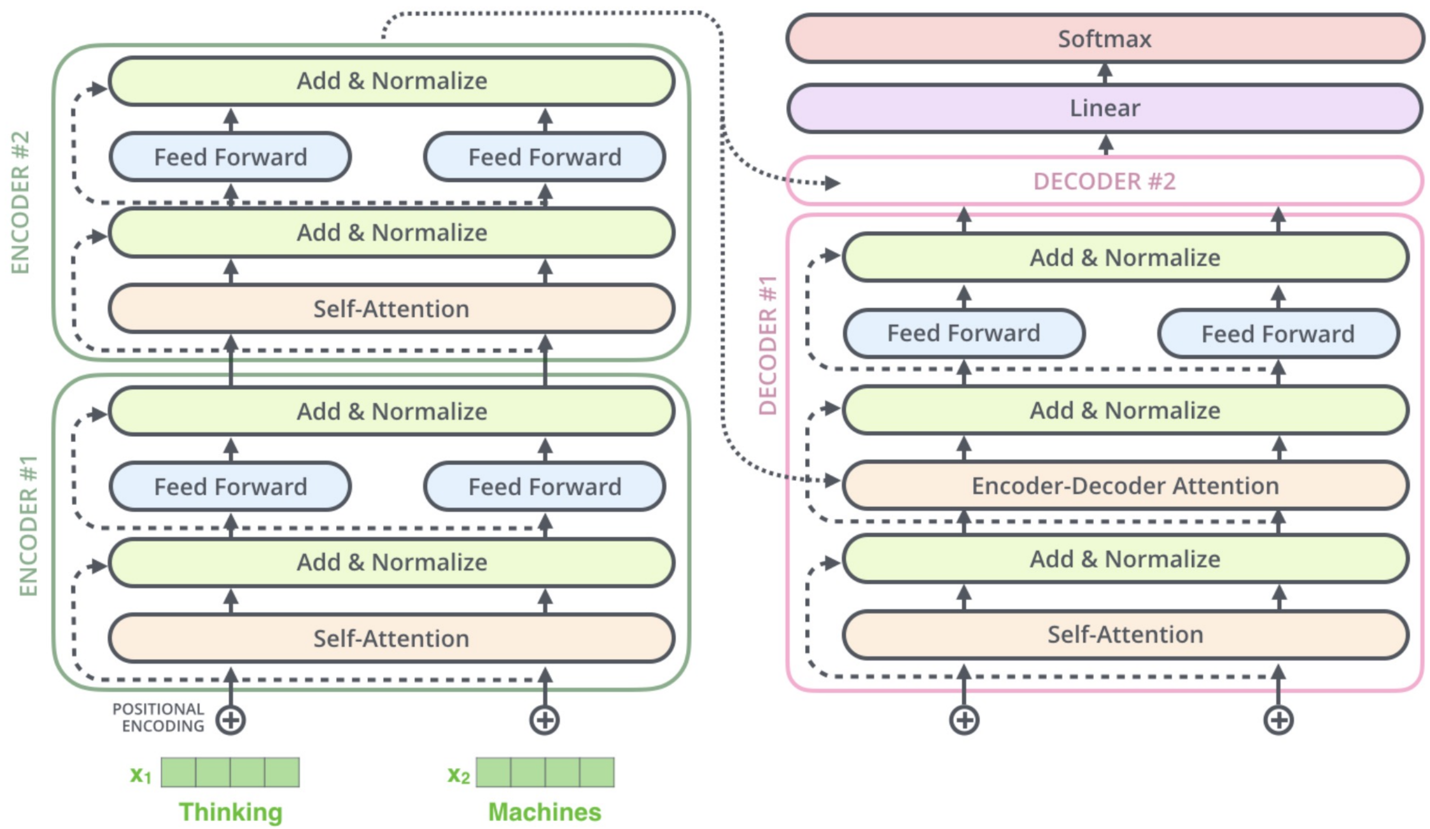
- Although RNNs show remarkable successes, there are **fundamental issues**:
  1. **O(sequence length)** steps for distant word pairs to interact means
  2. Forward/backward passes have **O(sequence length)** unparallelizable operations

- In contrast, **attention has some advantages** in these aspects:
  1. Maximum interaction distance: **O(1)**
     - Since all words interact at each layer
  2. Number of unparallelizable operations does **not increase with respect to length**

Q. Then, can we design an architecture **only using attention** modules?
- Remark. We saw attention from the **decoder to the encoder**; but here, we'll think about attention **within a single sentence**.

Transformer (Self-attention)

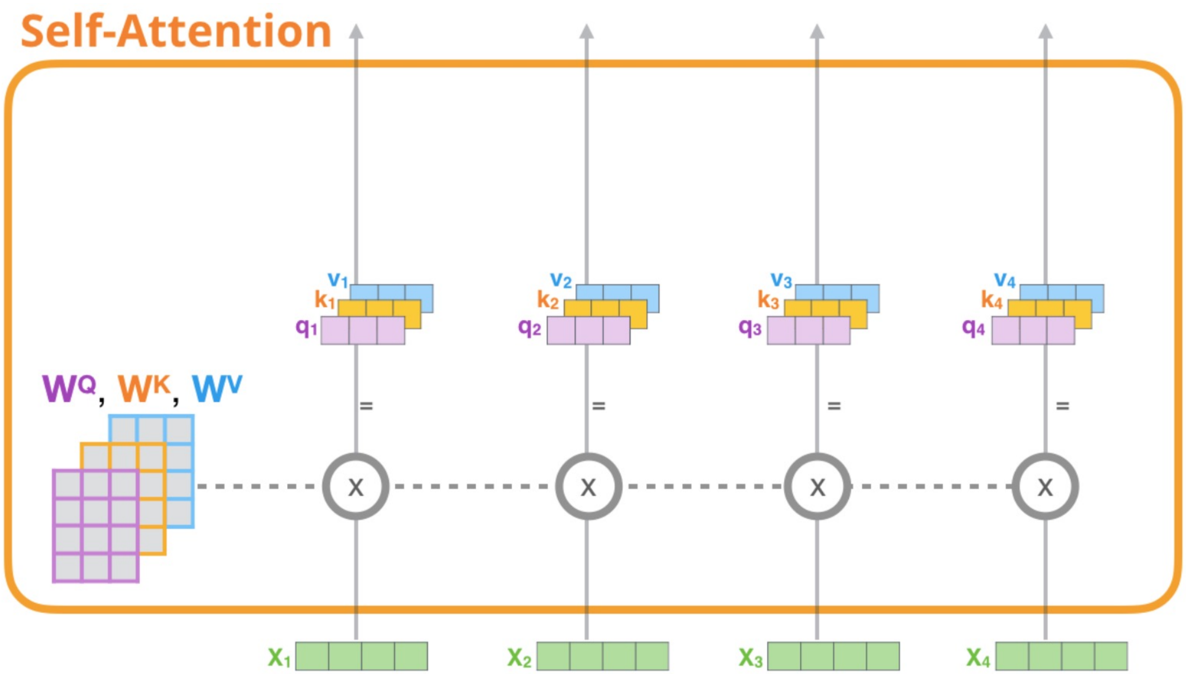
- Transformer [Vaswani et al., 2017] has an **encoder-decoder** structure and they are composed of multiple block with **multi-head (self) attention** module



*reference: http:// http://jalammar.github.io/illustrated-transformer

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i by multiplying **learnable** weight matrices

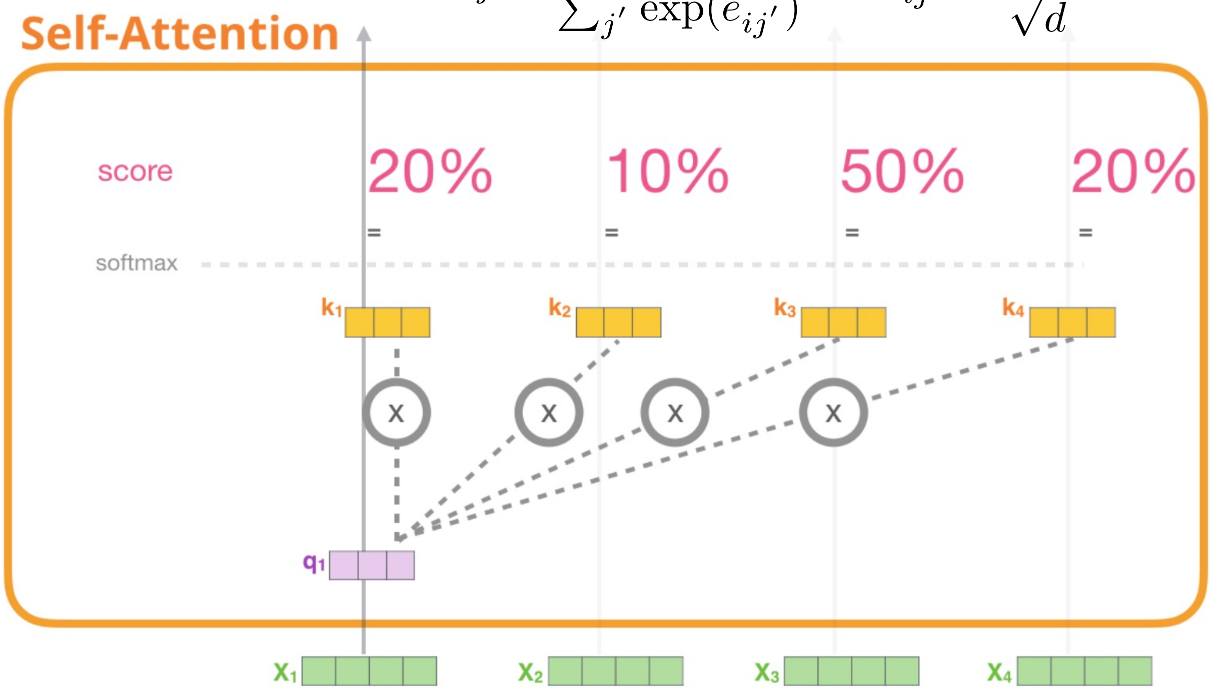
$$q_i = W^Q x_i, k_i = W^k x_i, v_i = W^V x_i$$



*reference: http:// http://jalammar.github.io/illustrated-gpt2

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij} of **how well they match**

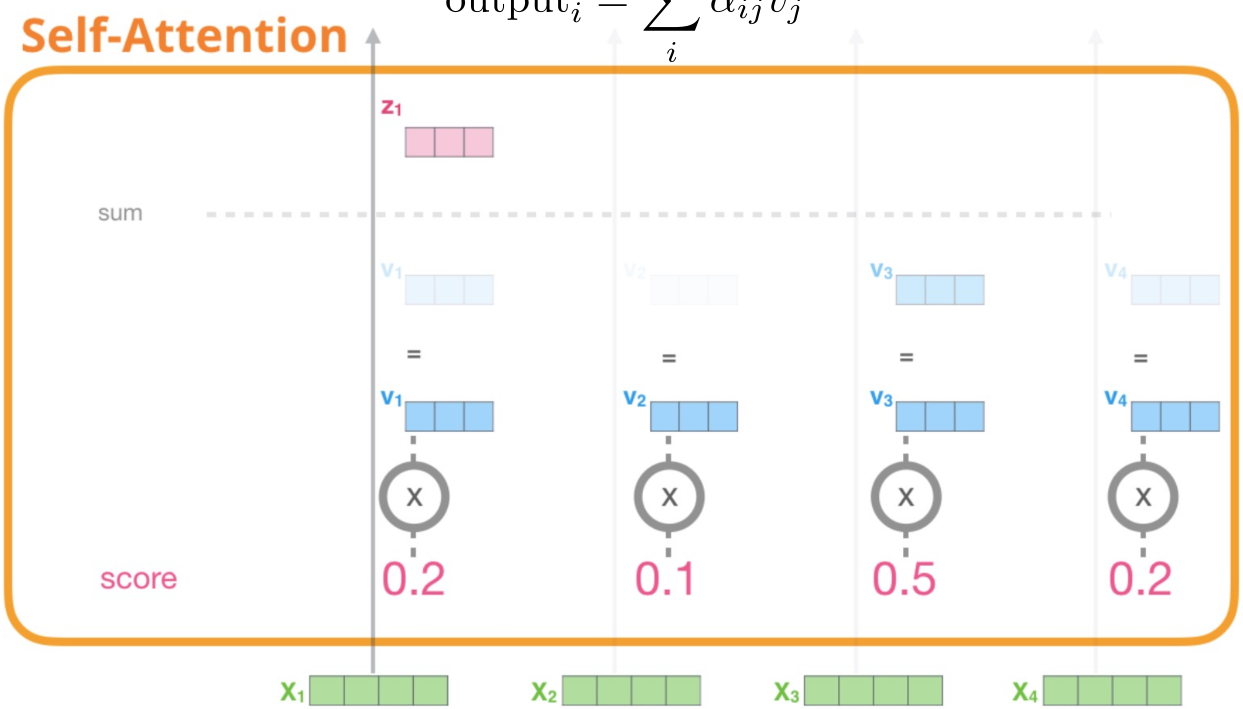
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad e_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$$



*reference: http:// http://jalammar.github.io/illustrated-gpt2

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**

$$\text{output}_i = \sum_i \alpha_{ij} v_j$$



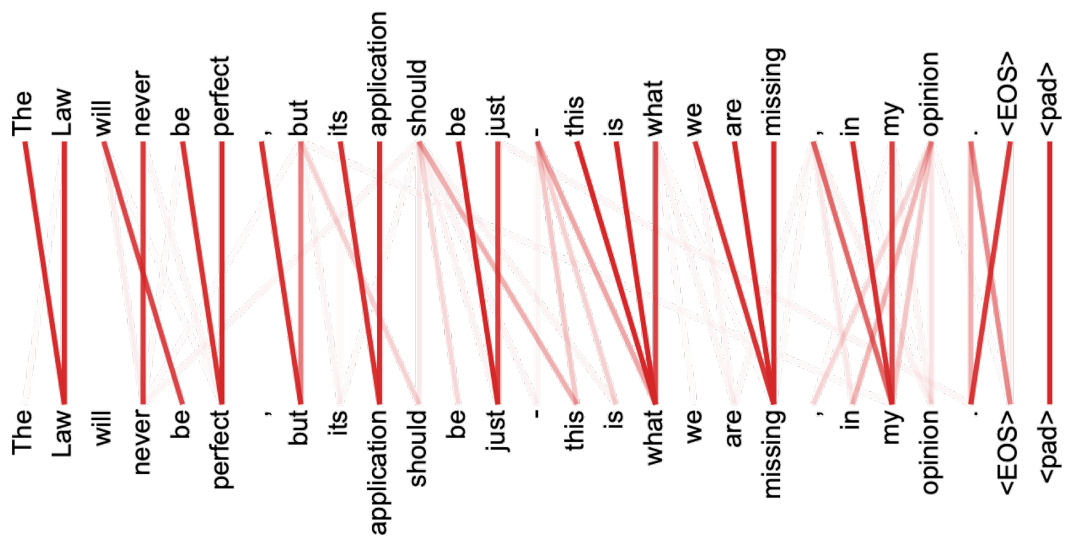
*reference: http:// http://jalammar.github.io/illustrated-gpt2

- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i, create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**
  - Hence, self-attention is **effective to learn the context** within given sentence
    - It's easier than recurrent layer to be parallelized and model the long-term dependency

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

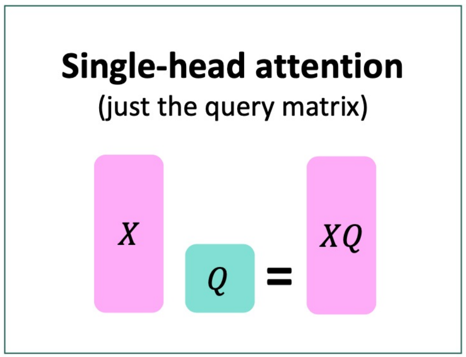
*reference: http:// http://jalammar.github.io/illustrated-gpt2

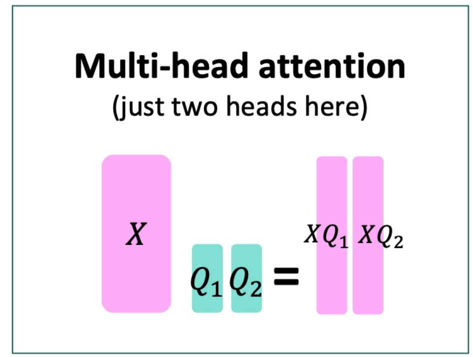
- **Self-attention**
  - **Recall**: Attention operates on query, key, and value
    - Query is decoder's hidden state, key and value are encoder's hidden states in seq2seq
  - In self-attention, the query, key, and value are drawn from the **same source**
    1. For each input x_i , create query, key, and value vectors q_i, k_i, v_i
    2. Multiply (**dot product**) the current query vector, by all the key vectors, to get a **score** α_{ij}
    3. Multiply the value vectors by the scores, then **sum up**
  - Hence, self-attention is **effective to learn the context** within given sentence
    - It's easier than recurrent layer to be parallelized and model the long-term dependency
    - It also provides an **interpretability** of learned representation



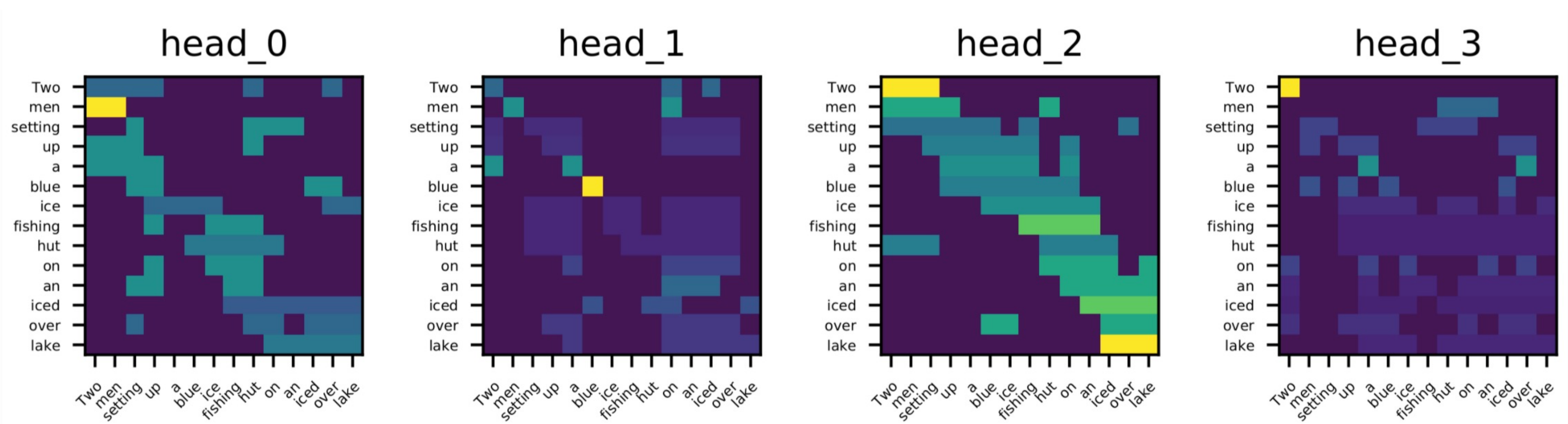
*reference: http:// http://jalammar.github.io/illustrated-gpt2

- **Multi-head attention**
  - Applying **multiple attentions at once** to look in multiple places in the sentence
    - To prevent the increase of computation, original attentions weights are **divided**



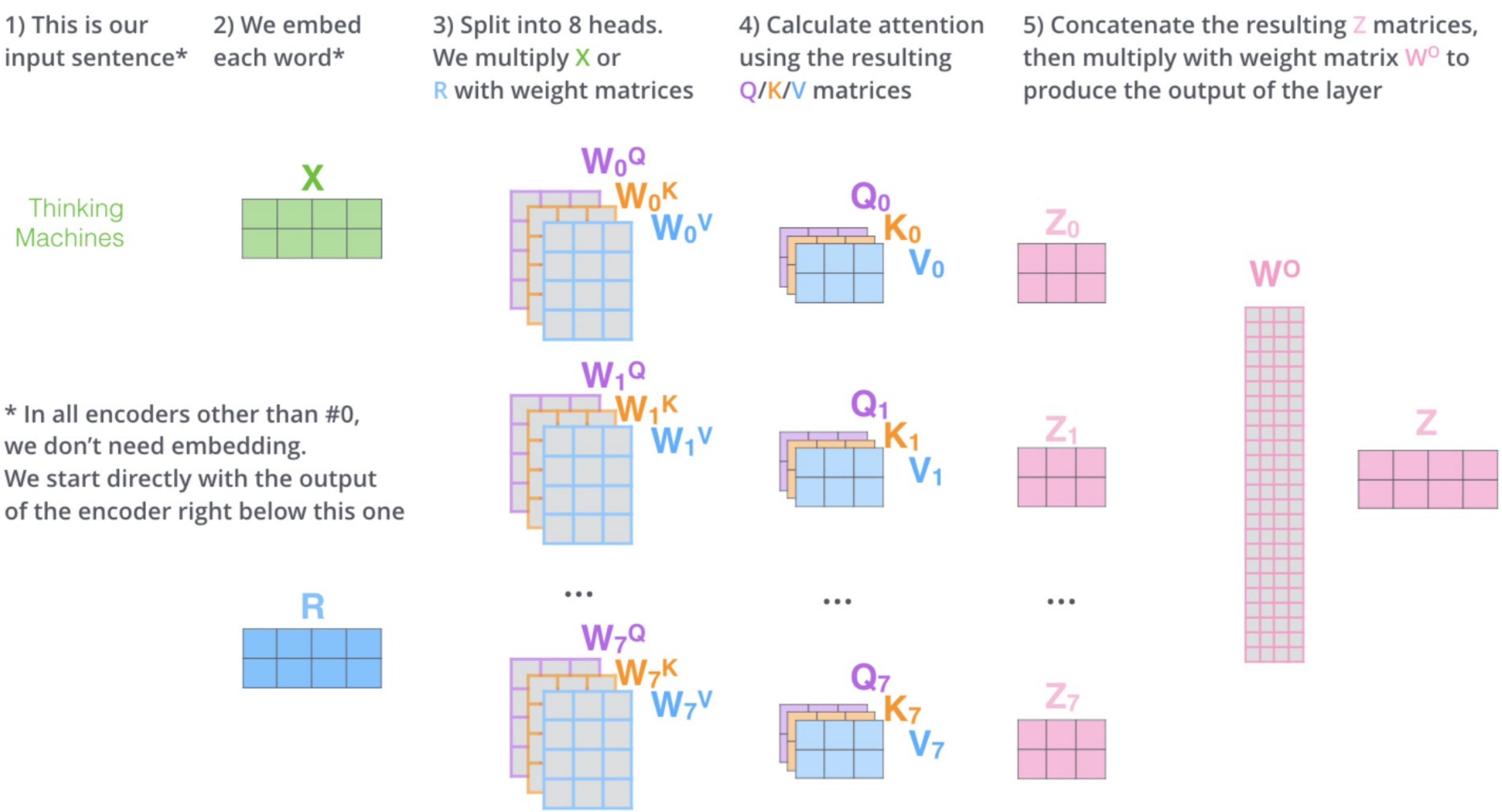


Same amount of computation as single-head self-attention



*reference: http://web.stanford.edu/class/cs224n/

- **Multi-head attention**
  - Applying **multiple attentions at once** to look in multiple places in the sentence

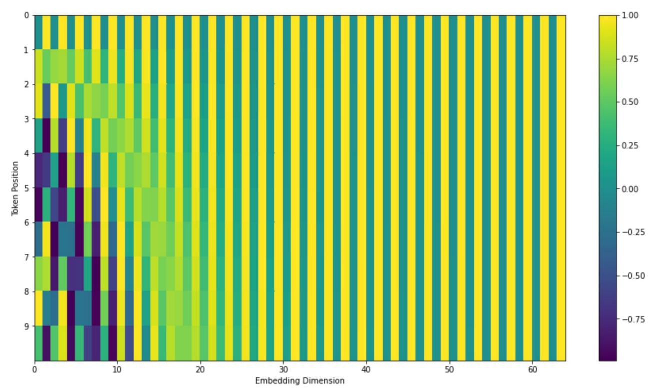


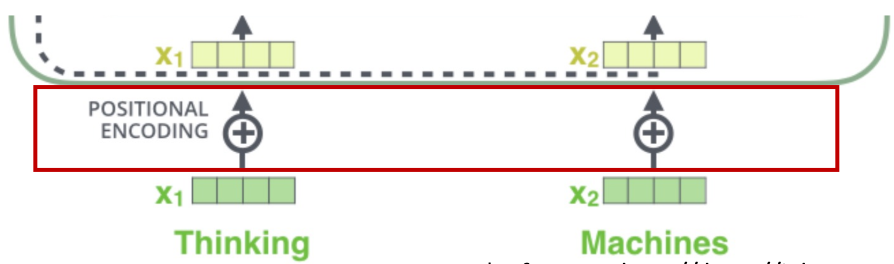
*reference: http:// http://jalammar.github.io/illustrated-transformer

- **Encoder**
  - Self-attention is **invariant to order** of input sequence
    - To represent the order of sequence, **positional encoding** is added to input embeddings at the **bottoms of the encoder and decoder stacks**
  - Fixed sine and cosine functions are used for each position pos and dimension i

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \quad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

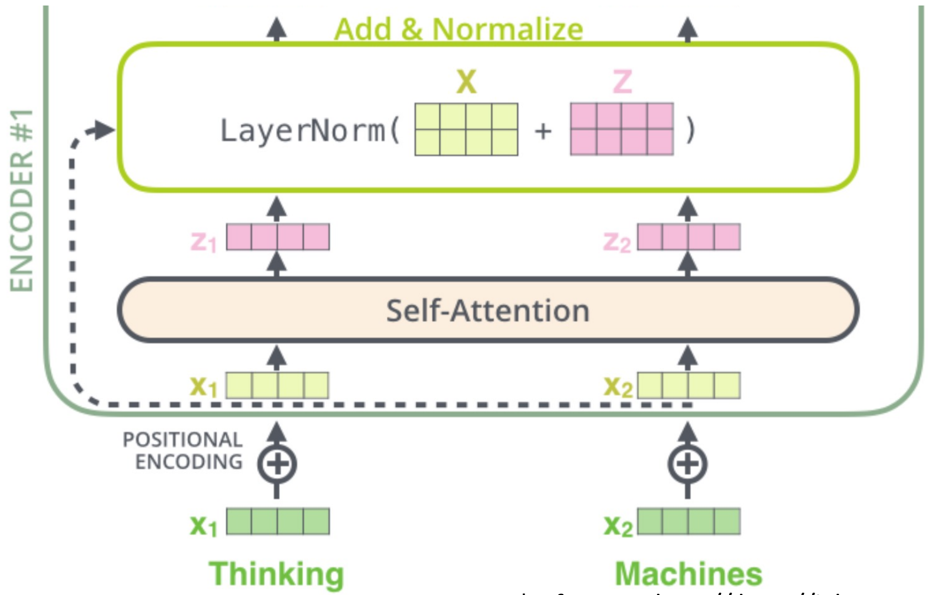
    - PE_{pos+k} can be derived as a linear function of PE_{pos} → easier to learn a relative position
    - Compare to learning encoding, it's better for **extrapolation** (not encountered in training)





*reference: http:// http://jalammar.github.io/illustrated-transformer

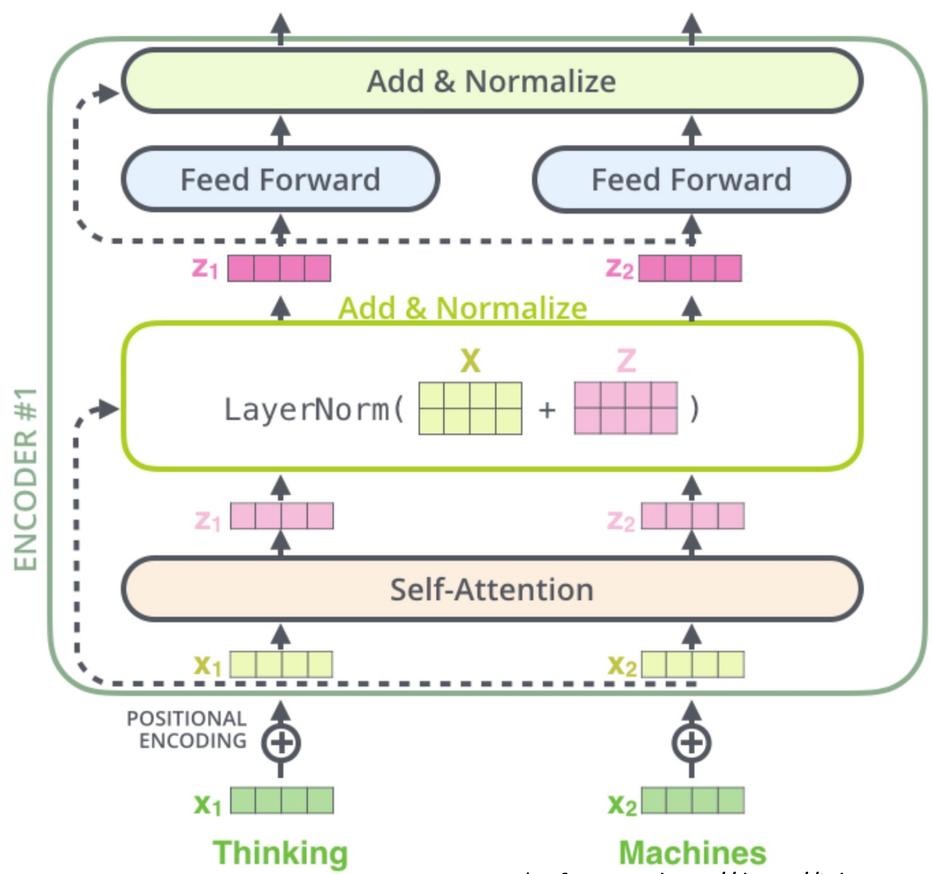
- **Encoder**
  - Self-attention is **invariant to order** of input sequence ⟶ **positional encoding**
  - **Residual connections** (dotted) and **layer normalization** are used to help training



*reference: http:// http://jalammar.github.io/illustrated-transformer

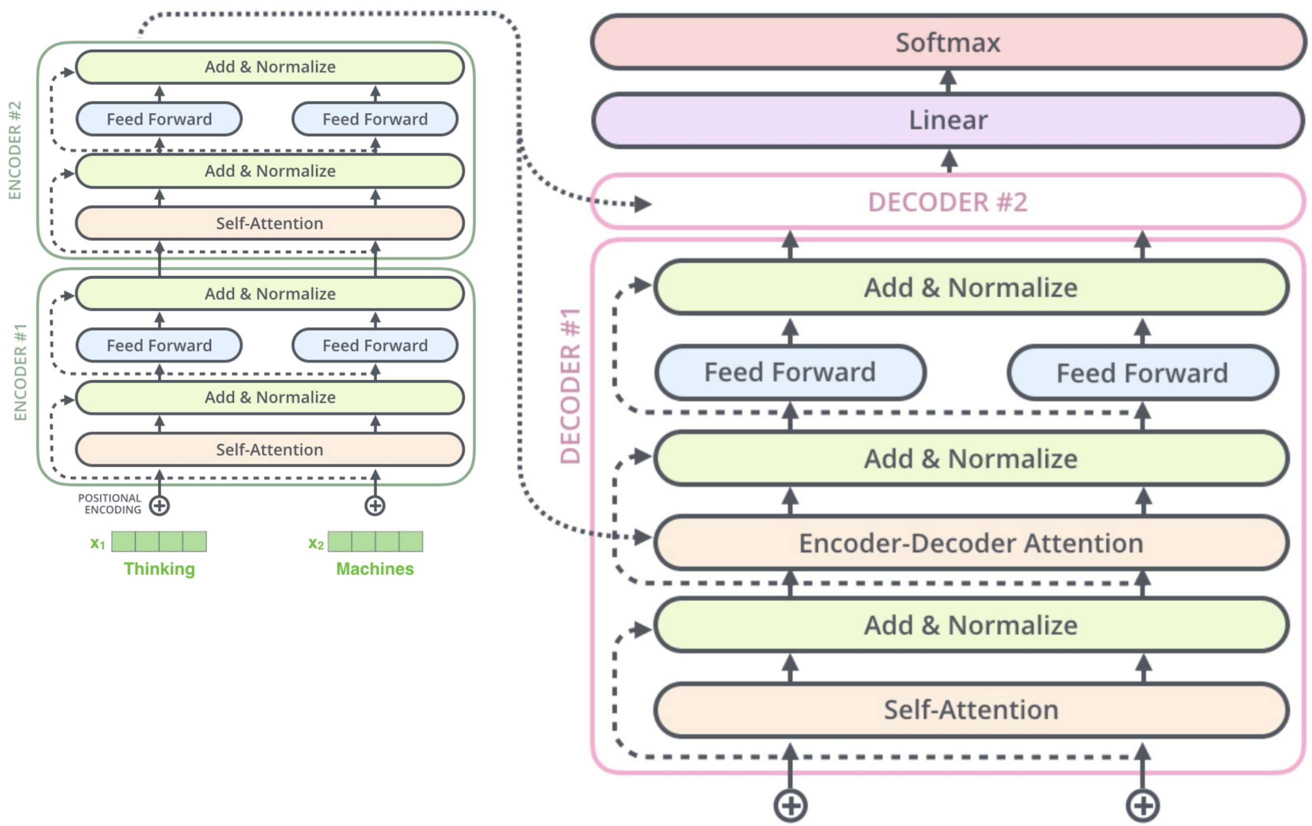
Transformer (Self-attention)

- **Encoder**
  - Self-attention is **invariant to order** of input sequence ⟶ **positional encoding**
  - **Residual connections** (dotted) and **layer normalization** are used to help training
  - Non-linearity is imposed by adding position-wise **feed-forward networks**



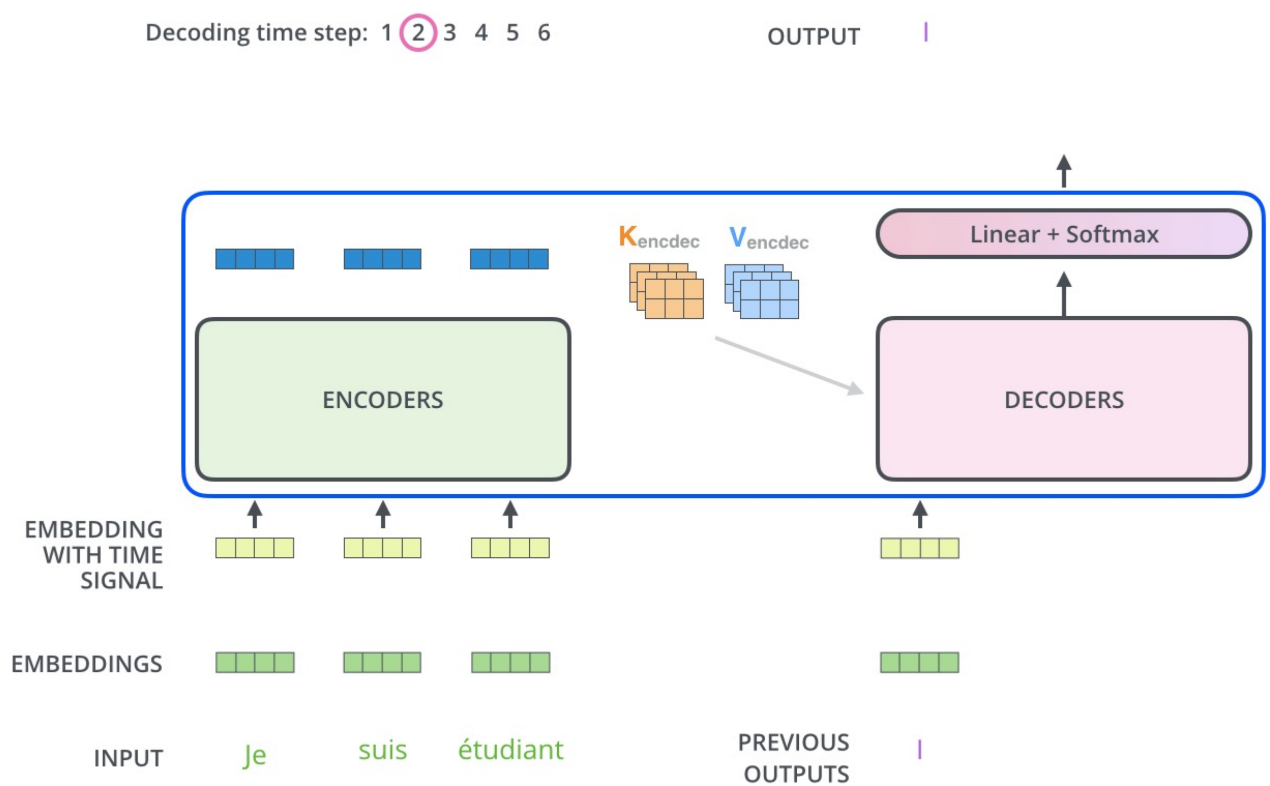
*reference: http:// http://jalammar.github.io/illustrated-transformer

- **Decoder**
  - Most parts are same with encoder except **encoder-decoder(cross) attention**
  - This cross attention is previously used in seq2seq model
    - Queries are drawn from the **decoder**
    - Keys and values are drawn from the **encoder** (like context vector)



*reference: http:// http://jalammar.github.io/illustrated-transformer

- **Decoder**
  - Most parts are same with encoder except **encoder-decoder(cross) attention**
  - This cross attention is previously used in seq2seq model
    - Queries are drawn from the **decoder**
    - Keys and values are drawn from the **encoder** (like context vector)



*reference: http:// http://jalammar.github.io/illustrated-transformer

Transformer (Self-attention)

- Success of Transformer: **Machine Translation (MT)**
  - Initially, Transformer shows **better results at a fraction of the training cost**

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

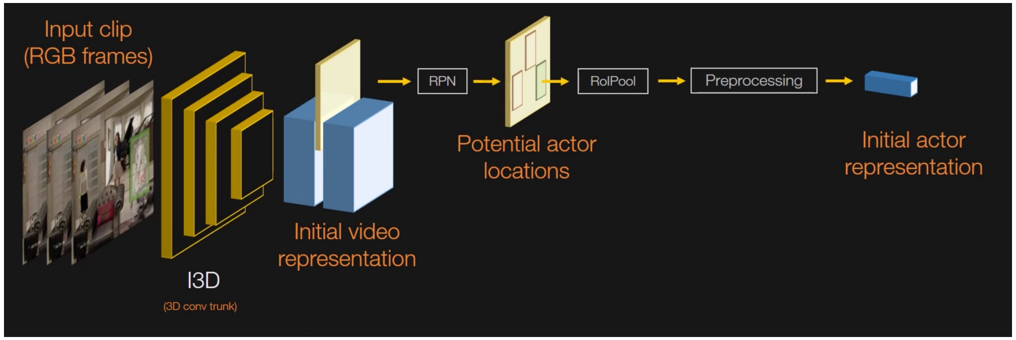
  - Nowadays, Transformer is still a standard for MT with additional techniques

System	**En→De**	
	news2017	**news2018**
baseline	30.90	45.40
+ langid filtering	30.78	46.43
+ ffn 8192	31.15	46.28
+ BT	33.62	46.66
+ fine tuning	-	47.61
+ ensemble	-	49.27
+ reranking	-	50.63
WMT'18 submission	-	46.10
WMT'19 submission	**42.7**	

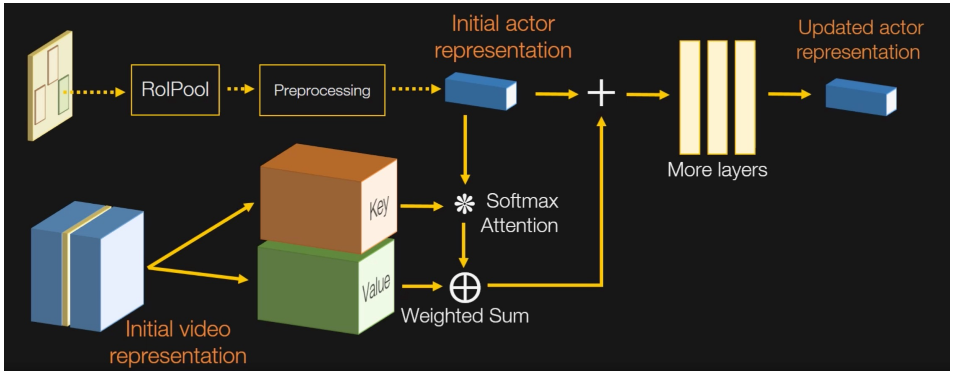
*reference: https://arxiv.org/pdf/1907.06616v1.pdf

Transformer (Self-attention)

- Success of Transformer: **Video action recognition** [Girdhar et al., 2018]
  - **Goal**: localize the atomic action in space and time
  - Previous approaches just use the feature of key frame with object detection
    - But, it's hard to model the interaction between frames



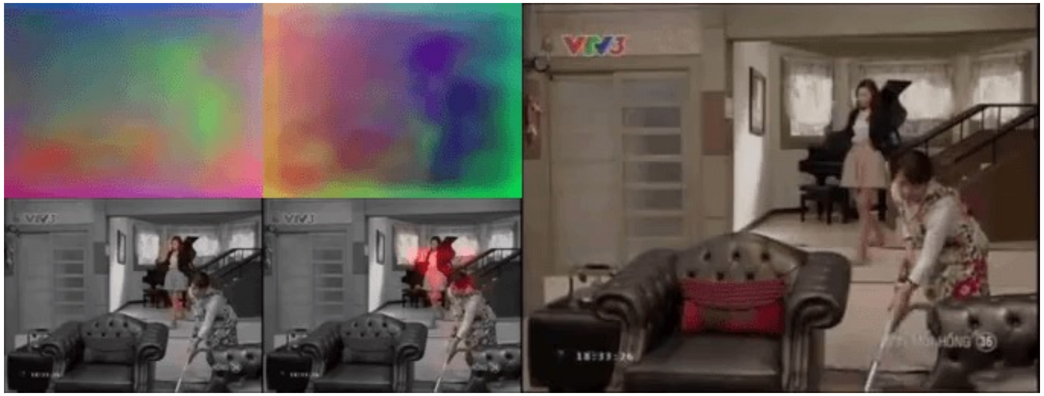
- **Self-attention is an effective way** to resolve this issue



*reference: https://rohitgirdhar.github.io/ActionTransformer

Transformer (Self-attention)

- Success of Transformer: **Video action recognition** [Girdhar et al., 2018]
  - **Qualitative results of learned attention**



  - **Winner of AVA challenge in 2019: > 3.5 %** than previous challenge winner

Method	Modalities	Architecture	Val mAP	Test mAP
Single frame [16]	RGB, Flow	R-50, FRCNN	14.7	-
AVA baseline [16]	RGB, Flow	I3D, FRCNN, R-50	15.6	-
ARCN [42]	RGB, Flow	S3D-G, RN	17.4	-
Fudan University	-	-	-	17.16
YH Technologies [52]	RGB, Flow	P3D, FRCNN	-	19.60
Tsinghua/Megvii [23]	RGB, Flow	I3D, FRCNN, NL, TSN, C2D, P3D, C3D, FPN	-	21.08
Ours (Tx-only head)	RGB	I3D, Tx	24.4	24.30
Ours (Tx+I3D head)	RGB	I3D, Tx	24.9	24.60
Ours (Tx+I3D+96f)	RGB	I3D, Tx	**25.0**	**24.93**

*reference: https://rohitgirdhar.github.io/ActionTransformer

Transformer (Self-attention)

- Success of Transformer: **Music generation** [Huang et al., 2018]
  - **Goal**: generate music which contains structure at multiple timescales (short to long)
  - Performance RNN (LSTM): lack of long-term structure

  - Music transformer; able to continue playing with consistent style

Next, Pre-training with Transformer

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

- **Motivation**
  - Many success of CNN comes from ImageNet-pretrained networks
    - Simple fine-tuning improves the performance than training from scratch
  - Then, can we train a similar universal encoder for NLP tasks?
    - As labeling of NLP task is more ambiguous, **unsupervised pre-training is essential**
  - **Language modeling**, i.e., reconstruction, is simple and feasible for our goal
    - With diverse examples, model can learn the useful knowledge about the world

"Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was __." $\longrightarrow$ *terrible*

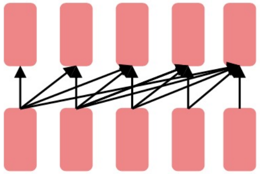
"I wat thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, __" $\longrightarrow$ *34*

"I went to the ocean to see the fish, turtles, seals, and __" $\longrightarrow$ *sand*

Pre-training / Fine-tuning Paradigm with Transformers

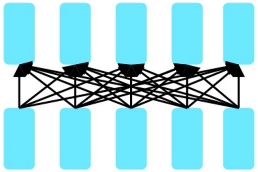
- **Motivation**
  - Many success of CNN comes from ImageNet-pretrained networks
    - Simple fine-tuning improves the performance than training from scratch
  - Then, can we train a similar universal encoder for NLP tasks?
    - As labeling of NLP task is more ambiguous, **unsupervised pre-training is essential**
  - **Language modeling**, i.e., reconstruction, is simple and feasible for our goal
    - With diverse examples, model can learn the useful knowledge about the world

- **Pre-training for two types of architectures**
  - Architecture influences the type of pre-training, and natural use cases



Decoders

- E.g. **GPT**
- Pre-training with **normal** language modeling
- Better use for **generation** tasks



Encoders

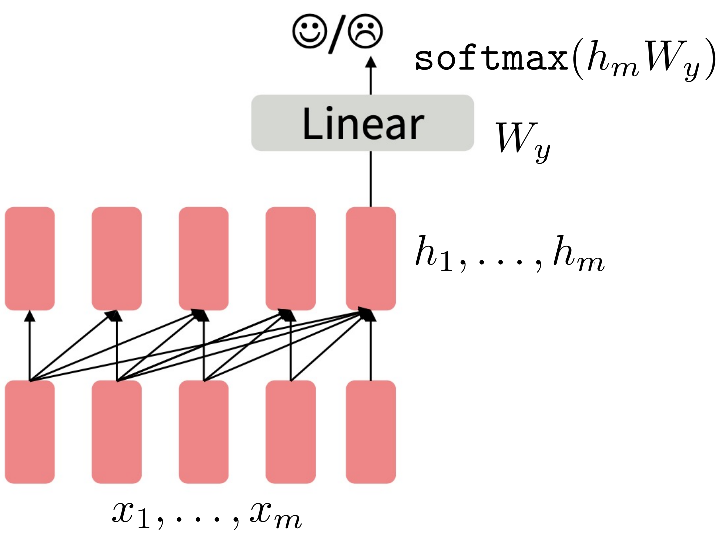
- E.g. **BERT**
- Pre-training with **masked** language modeling
- Better use for **discriminative** tasks (classification)

GPT: Generative Pre-Training with Transformer's Decoder

- **GPT** [Radford et al., 2018]

$$\arg\max_{\theta} \log p(\boldsymbol{x}) = \sum_{n} p_{\theta}(x_n | x_1, \dots, x_{n-1})$$

  - **Pre-training** by language modeling over 7000 unique books (**unlabeled data**)
    - Contains long spans of contiguous text, for learning long-distance dependencies
  - **Fine-tuning** by training a classifier with target task-specific **labeled data**
    - Classifier is added on the final transformer block's last word's hidden state



GPT: Generative Pre-Training with Transformer's Decoder

- **GPT** [Radford et al., 2018]

$$\arg\max_{\theta} \log p(\boldsymbol{x}) = \sum_{n} p_{\theta}(x_n | x_1, \ldots, x_{n-1})$$

  - **Pre-training** by language modeling over 7000 unique books (**unlabeled data**)
    - Contains long spans of contiguous text, for learning long-distance dependencies
  - **Fine-tuning** by training a classifier with target task-specific **labeled data**
    - Classifier is added on the final transformer block's last word's hidden state

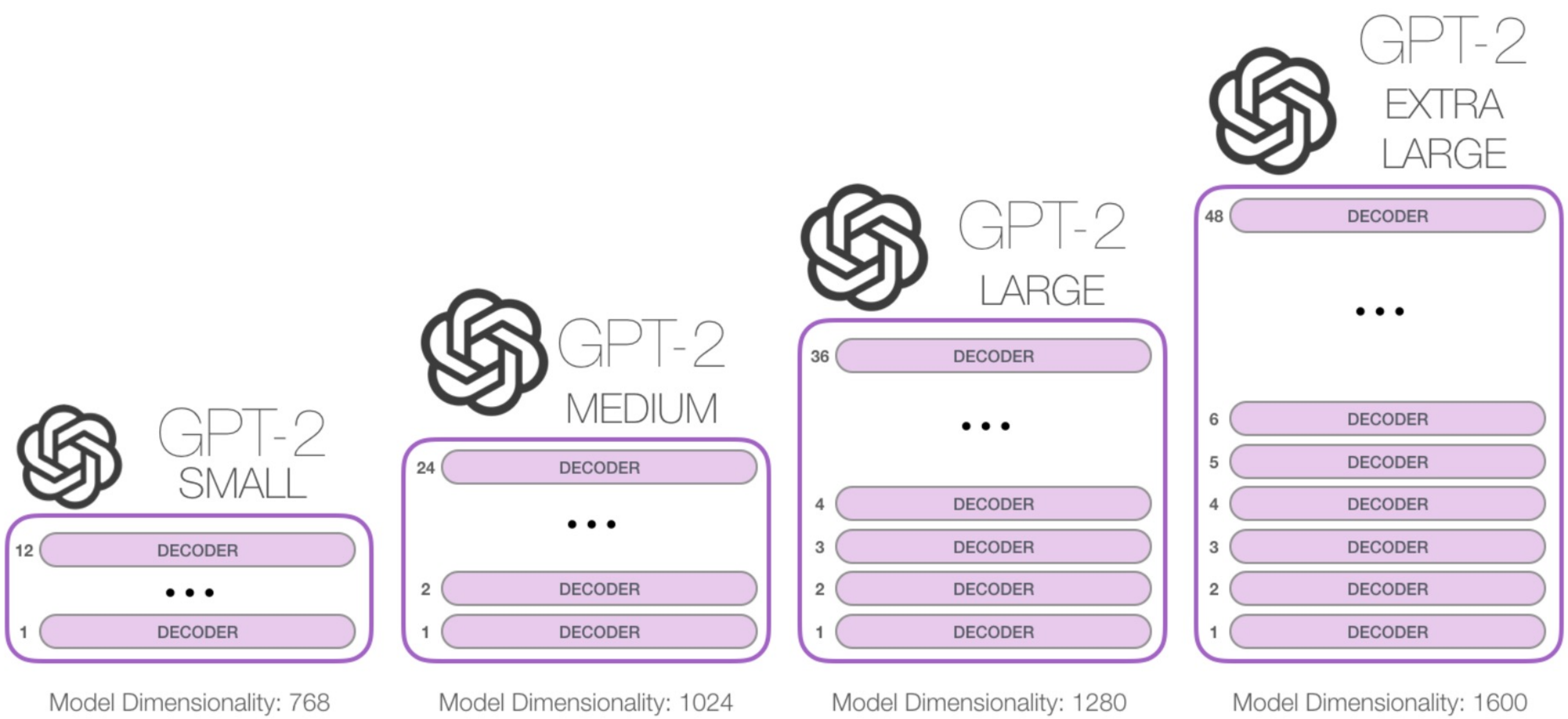
Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned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

GPT's results on various *natural language inference* datasets

*reference: http://web.stanford.edu/class/cs224n/

GPT-2: Language Models are Unsupervised Multitask Learners

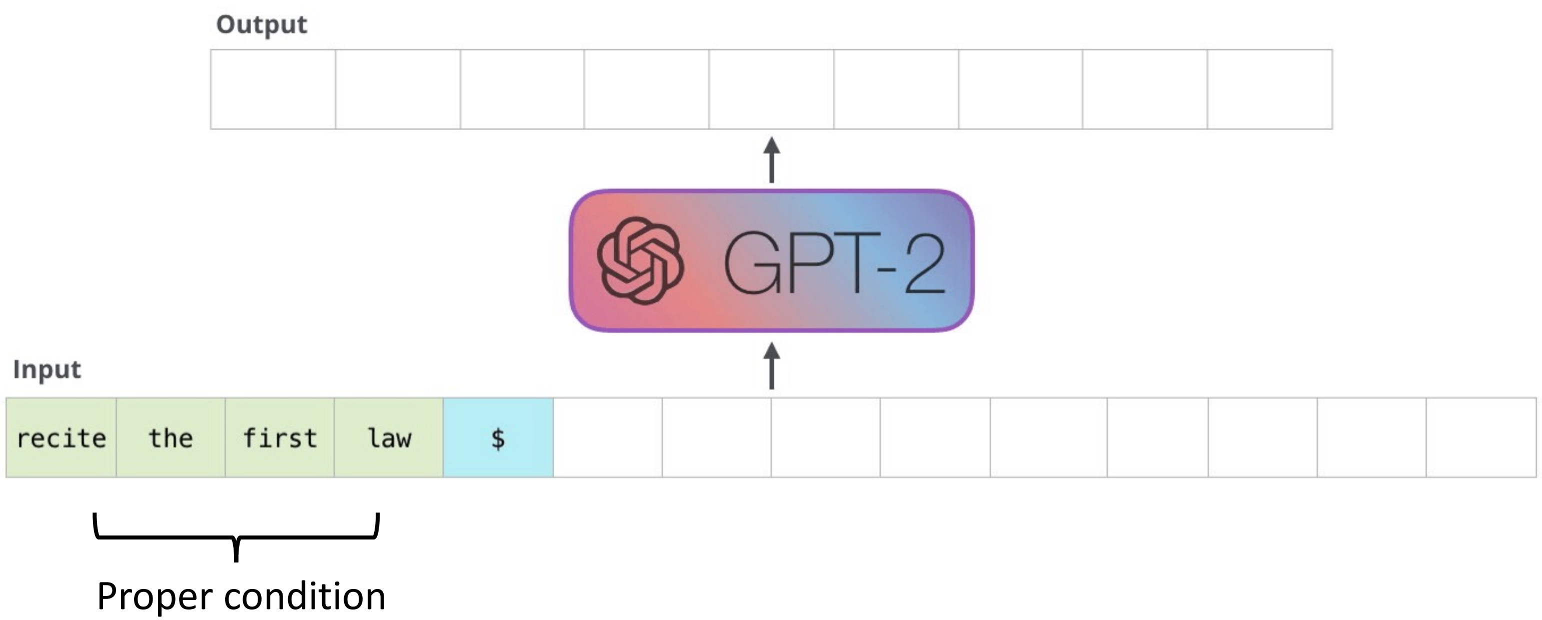
- **GPT-2** [Radford et al., 2019]
  - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
    - Much **larger datasets**; 8 million documents from web (40 GB of text)
    - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)



*reference: http:// http://jalammar.github.io/illustrated-gpt2

GPT-2: Language Models are Unsupervised Multitask Learners

- **GPT-2** [Radford et al., 2019]
  - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
    - Much **larger datasets**; 8 million documents from web (40 GB of text)
    - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)
  - GPT-2 can perform down-stream tasks in a **zero-shot setting**
    - Via conditional generation without any parameter or architecture modification



*reference: http:// http://jalammar.github.io/illustrated-gpt2

GPT-2: Language Models are Unsupervised Multitask Learners

- **GPT-2** [Radford et al., 2019]
  - **Pre-training** by language modeling as same as previous GPT-1, but **training with..**
    - Much **larger datasets**; 8 million documents from web (40 GB of text)
    - Much **larger model size**; # of parameters: 117M (GPT-1) ⟶ 1542M (extra-large GPT-2)
  - GPT-2 can perform down-stream tasks in a **zero-shot setting**
    - Via conditional generation without any parameter or architecture modification
  - **Remark**. Largest model still underfits.. ⟶ larger model for better performance?

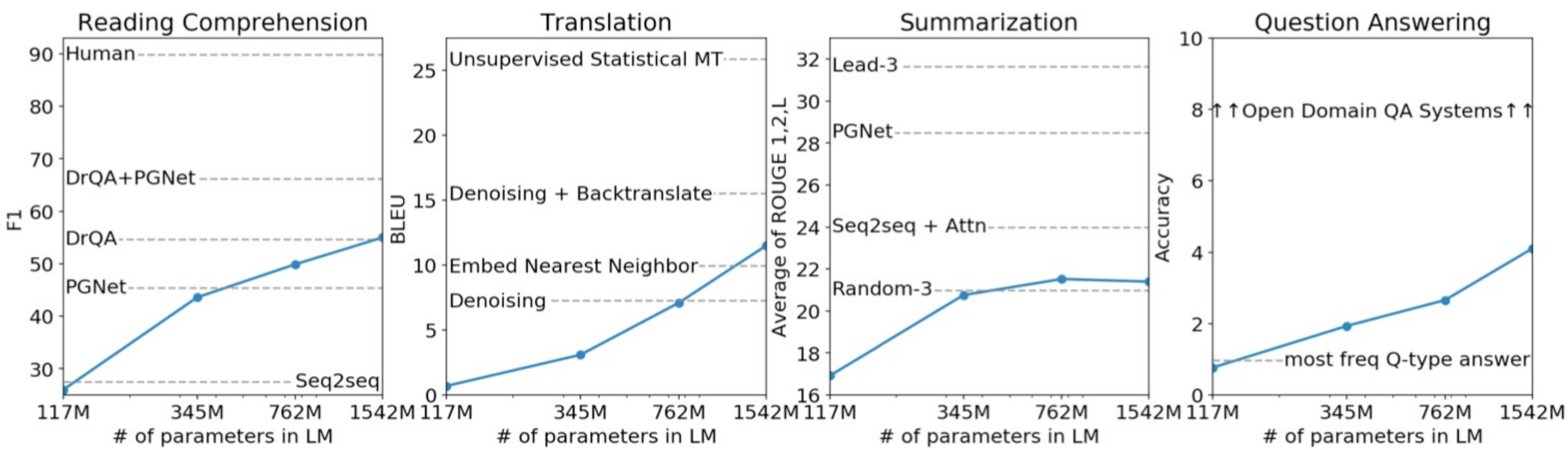
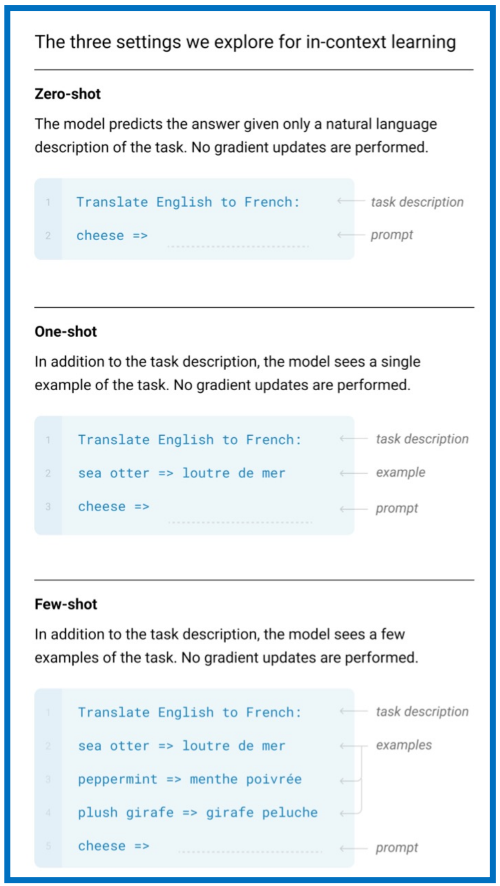
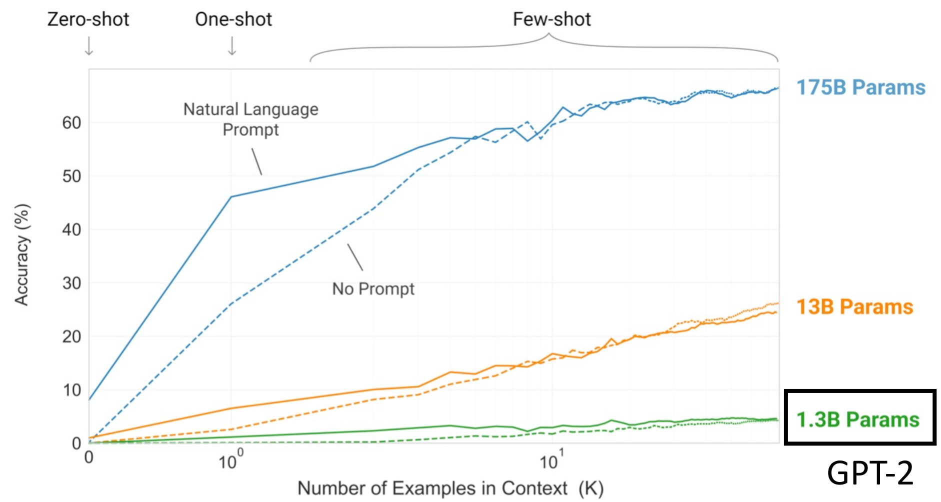


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

- **GPT-3**: Language Models are Few-shot Learners [Brown et al., 2020]
  - **Very large** language models seem to **perform in-context learning without gradient steps (fine-tuning)**
    - **In-context learning**; adapting to specific task from examples with some context

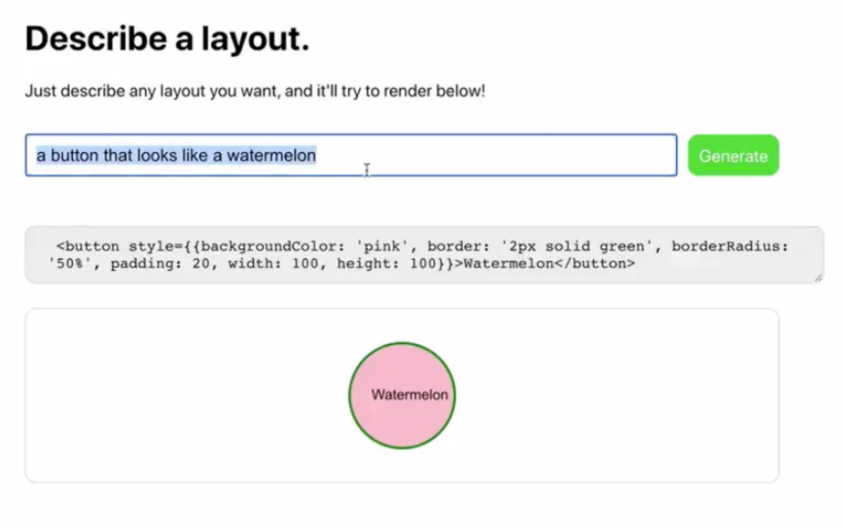




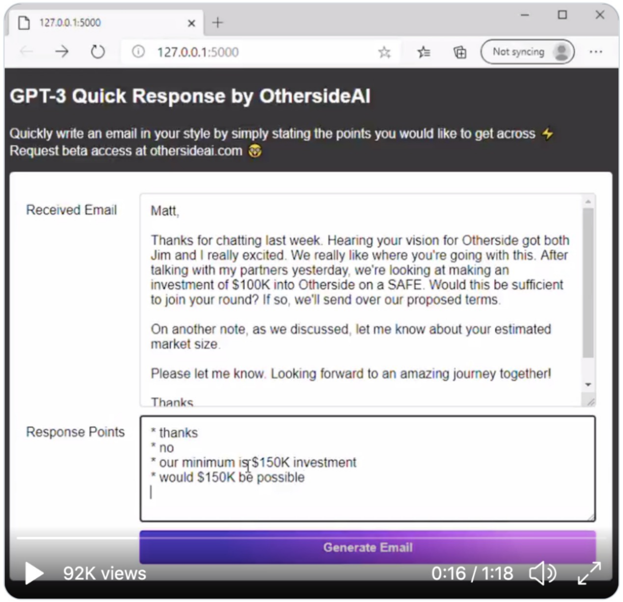
Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP$^+$20]	**44.5**	**45.5**	**68.0**
T5-11B+SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	**68.0**
GPT-3 Few-Shot	29.9	41.5	**71.2**

Results on open-domain question answering

- **GPT-3**: Language Models are Few-shot Learners [Brown et al., 2020]
  - **Very large** language models seem to **perform in-context learning without gradient steps (fine-tuning)**
    - **In-context learning**; adapting to specific task from examples with some context
  - It enables us to do a lot of interesting applications!
  - E.g.,



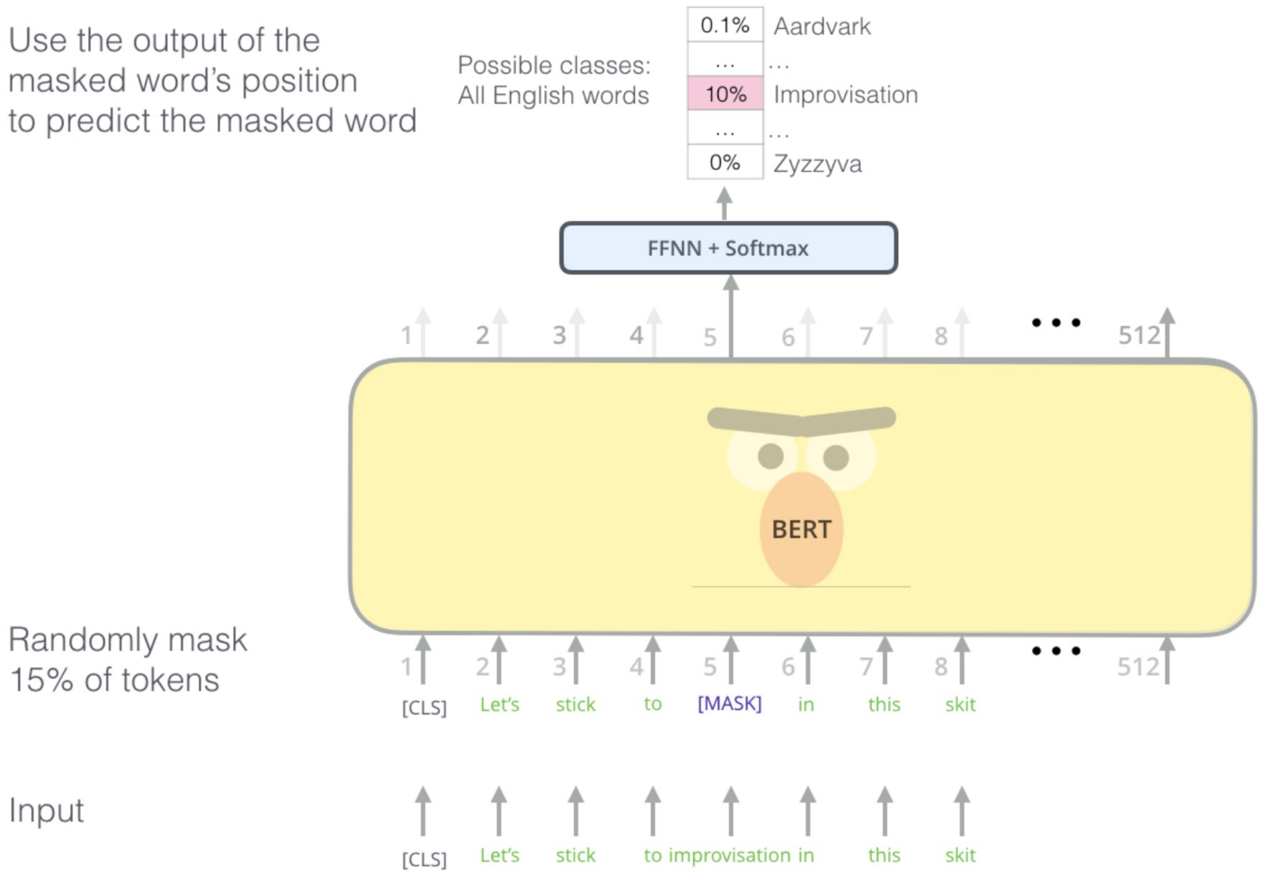
Simple code generation



Email response

*reference: https://machinelearningtokyo.com/2020/07/26/10-cool-gpt-3-demos/

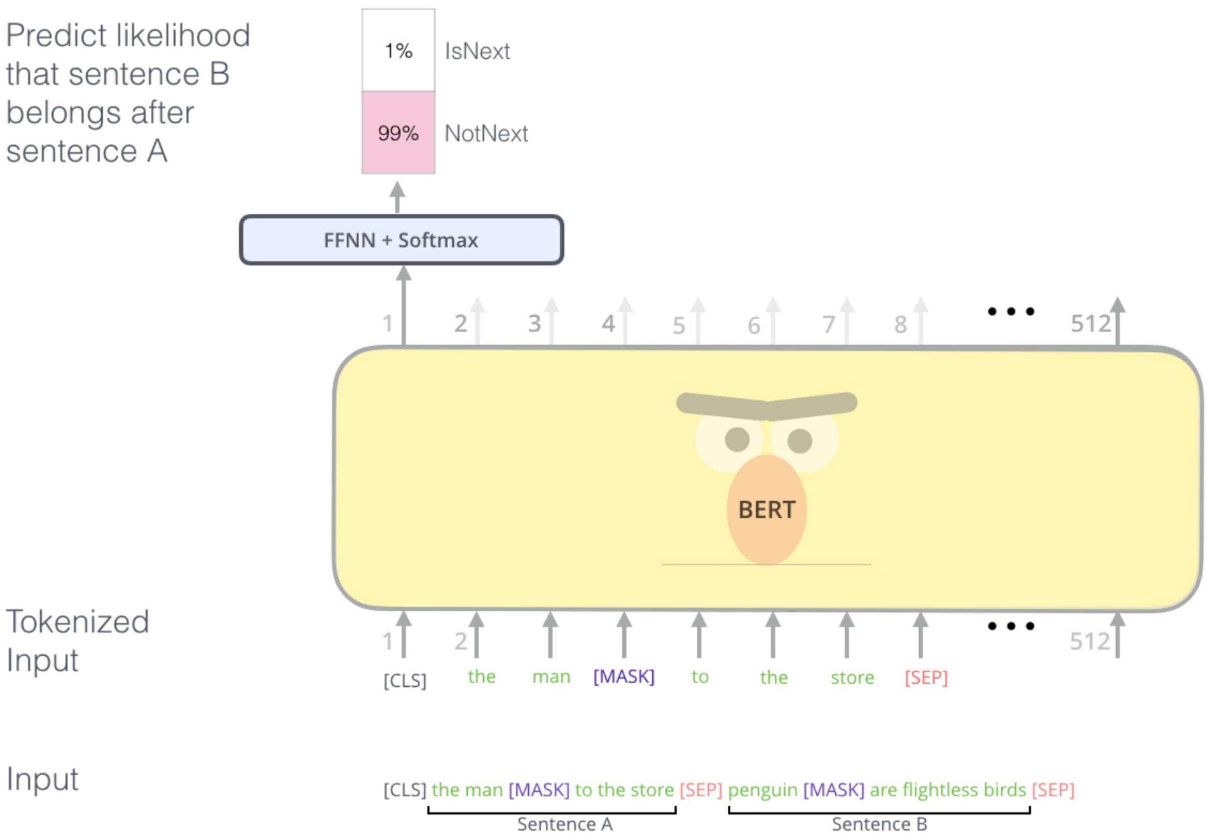
- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - As **encoders get bidirectional context**, language modeling can't be used anymore
  - Instead, **masked language modeling** is used for pre-training
    - Replace some fraction of words (15%) in the input, then predict these words



 *reference: http:// http://jalammar.github.io/illustrated-bert

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

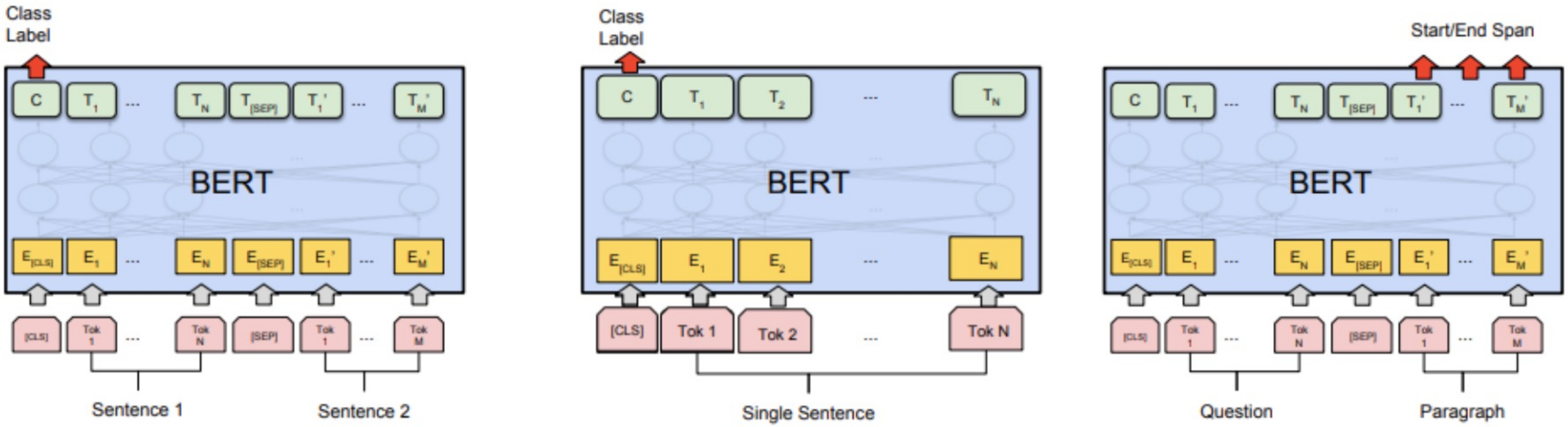
- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - As **encoders get bidirectional context**, language modeling can't be used anymore
  - Instead, **masked language modeling** is used for pre-training
  - Additionally, **next sentence prediction** (NSP) task is used for pre-training
    - Decide whether two input sentences are **consecutive or not**



*reference: http:// http://jalammar.github.io/illustrated-bert

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - Even **without** task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.
    - By simply fine-tuning a whole network with **additional linear classifier**



(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

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- **BERT**: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
  - Even **without** task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.
    - By simply fine-tuning a whole network with **additional linear classifier**

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	79.9	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**

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$\text{BERT}_{\text{BASE}}$	96.4	92.4
$\text{BERT}_{\text{LARGE}}$	**96.6**	**92.8**

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

- **RoBERTa** [Liu et al., 2019]
  - Simply modifying BERT design choices and training strategies with alternatives
    - Using **dynamic masking** instead of static masking in BERT
    - **Removing NSP task** and generate training data in single document instead
    - Much **larger data** for pre-training: 16GB ⟶ 160GB, and etc…
  - But, it leads a huge improvement in many downstream tasks

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

Drawback and Variants of Transformers

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Quadratic computation in self-attention** as a function of sequence length

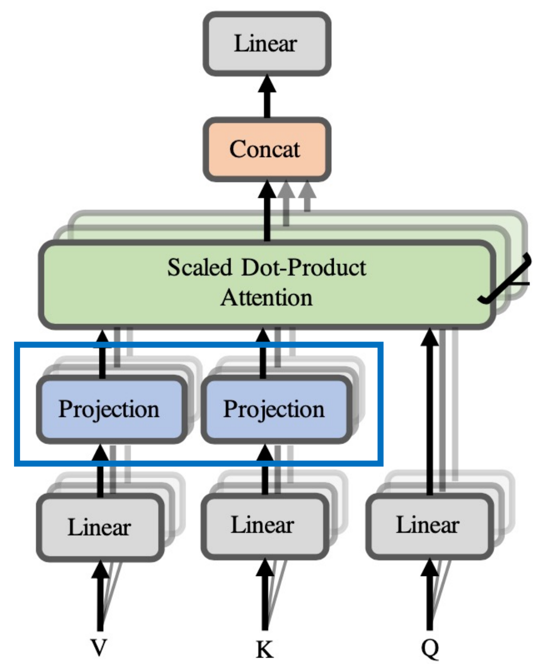
  Q. Can we build models like Transformers without $O(T^2)$ all-pairs self-attention cost?

  A. Linformer [Wang et al., 2020]

  - **Key idea**: **low rank approximation** of attention mechanism **with linear projection**

$$\begin{aligned}\text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\ &= \underbrace{\text{softmax}\left[\frac{QW_i^Q (KW_i^K)^T}{\sqrt{d_k}}\right]}_{P} VW_i^V\end{aligned}$$

$$\begin{aligned}\overline{\text{head}_i} &= \text{Attention}(QW_i^Q, \boxed{E_i} KW_i^K, \boxed{F_i} VW_i^V) \\ &= \underbrace{\text{softmax}\left(\frac{QW_i^Q (E_i KW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}: n\times k} \cdot \underbrace{F_i VW_i^V}_{k\times d},\end{aligned}$$



Drawback and Variants of Transformers

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Quadratic computation in self-attention** as a function of sequence length

  Q. Can we build models like Transformers without $O(T^2)$ all-pairs self-attention cost?

  A. Linformer [Wang et al., 2020]

  - **Key idea**: **low rank approximation** of attention mechanism **with linear projection**
  - Performance can be preserved after the approximation

n	Model	SST-2	IMDB	QNLI	QQP	Average
512	Liu et al. (2019), RoBERTa-base	93.1	94.1	90.9	**90.9**	92.25
	Linformer, 128	92.4	94.0	90.4	90.2	91.75
	Linformer, 128, shared kv	**93.4**	93.4	90.3	90.3	91.85
	Linformer, 128, shared kv, layer	93.2	93.8	90.1	90.2	91.83
	Linformer, 256	93.2	94.0	90.6	90.5	92.08
	Linformer, 256, shared kv	93.3	93.6	90.6	90.6	92.03
	Linformer, 256, shared kv, layer	93.1	94.1	**91.2**	90.8	**92.30**
512	Devlin et al. (2019), BERT-base	92.7	93.5	91.8	89.6	91.90
	Sanh et al. (2019), Distilled BERT	91.3	92.8	89.2	88.5	90.45
1024	Linformer, 256	93.0	93.8	90.4	90.4	91.90
	Linformer, 256, shared kv	93.0	93.6	90.3	90.4	91.83
	Linformer, 256, shared kv, layer	93.2	**94.2**	90.8	90.5	92.18

Drawback and Variants of Transformers

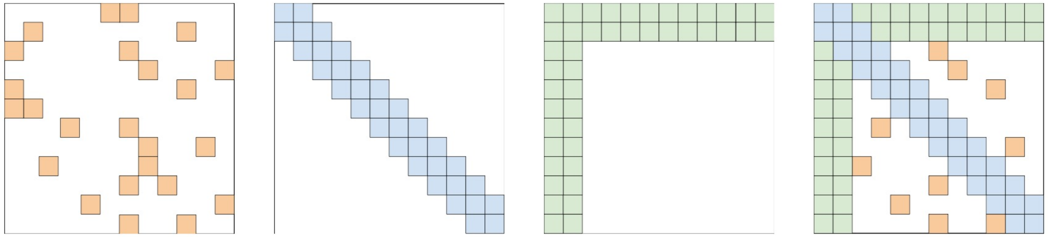
- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Quadratic computation in self-attention** as a function of sequence length

  Q. Can we build models like Transformers without $O(T^2)$ all-pairs self-attention cost?

  A. BigBird [Zaheer et al., 2020]

  - **Key idea**: replace all-pairs interactions with a family of other interactions, like 1) random attention, 2) local attention (window), 3) global attention
  - It can preserve the some property of original attention in theory
  - Due to effect as regularization, it sometimes improve the performance than original



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Model	HotpotQA			NaturalQ		TriviaQA	
	Ans	Sup	Joint	LA	SA	Full	Verified
HGN [26]	**82.2**	88.5	**74.2**	-	-	-	-
GSAN	81.6	88.7	73.9	-	-	-	-
ReflectionNet [32]	-	-	-	77.1	**64.1**	-	-
RikiNet-v2 [61]	-	-	-	76.1	61.3	-	-
Fusion-in-Decoder [39]	-	-	-	-	-	84.4	90.3
SpanBERT [42]	-	-	-	-	-	79.1	86.6
MRC-GCN [87]	-	-	-	-	-	-	-
MultiHop [14]	-	-	-	-	-	-	-
Longformer [8]	81.2	88.3	73.2	-	-	77.3	85.3
BIGBIRD-ETC	81.2	**89.1**	73.6	**77.8**	57.9	**84.5**	**92.4**

Drawback and Variants of Transformers

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Position representations**

  Q. Are simple absolute indices the best we can do to represent position?

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \qquad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

  A. Relative [Shaw et al., 2018] and structural [Wang et al., 2019] position representations

  - To consider pairwise relationships, **additional weights** a_{ij}^v, a_{ij}^k are introduced (consider a relative position up to l)

Original:

$$\text{output}_i = \sum_j \alpha_{ij} v_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad e_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$$

$$\Downarrow$$

Relative:

$$\text{output}_i = \sum_j \alpha_{ij}(v_j + a_{ij}^v) \qquad e_{ij} = \frac{q_i^T (k_j + a_{ij}^k)}{\sqrt{d}}$$

$$a_{ij}^v = w_{\text{clip}(j-i,l)}^v \qquad a_{ij}^k = w_{\text{clip}(j-i,l)}^k \qquad \text{clip}(x,l) = \max(-l, \min(l,x))$$

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

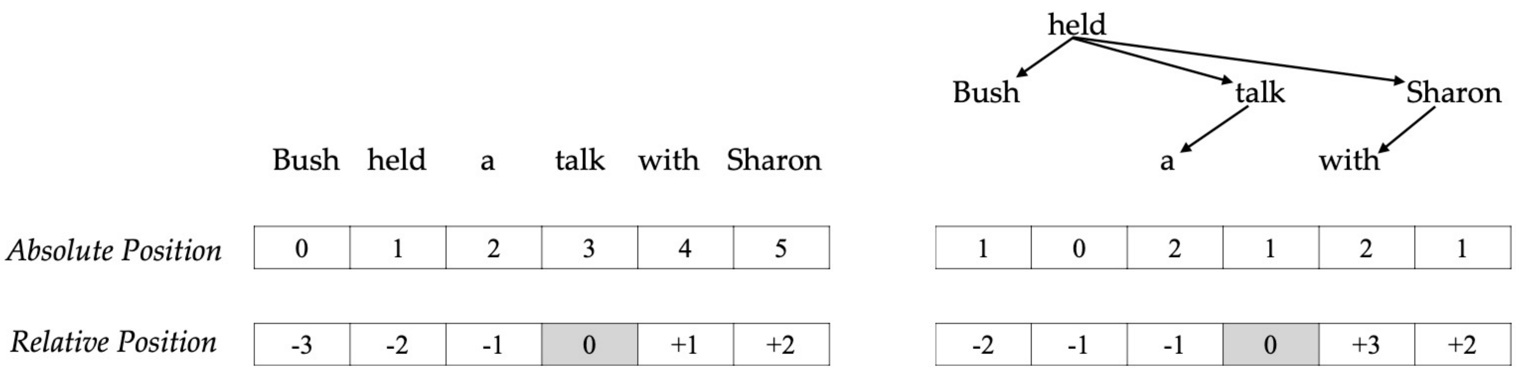
- **Position representations**

  Q. Are simple absolute indices the best we can do to represent position?

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \quad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

  A. Relative [Shaw et al., 2018] and **structural** [Wang et al., 2019] position representations

  - Imposing the **structural information** obtained from the classical NLP literature



(a) **Sequential** Position Encoding (b) **Structural** Position Encoding

Drawback and Variants of Transformers

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Position representations**

  Q. Are simple absolute indices the best we can do to represent position?

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \quad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

  A. Relative [Shaw et al., 2018] and **structural** [Wang et al., 2019] position representations

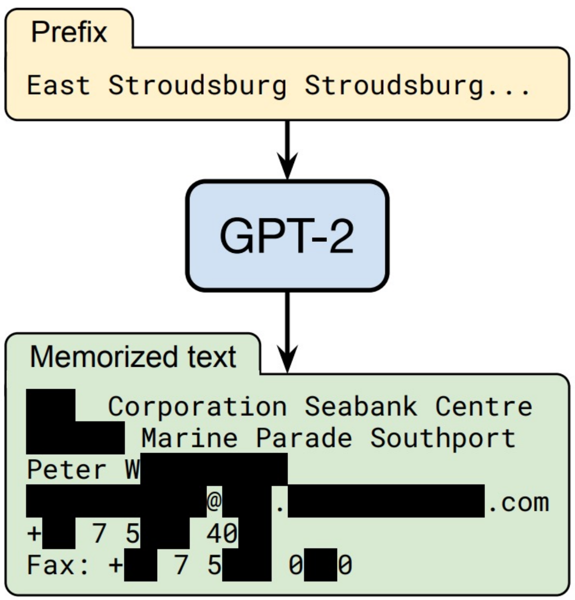
  - Imposing the **structural information** obtained from the classical NLP literature

Model Architecture	**Zh⇒En**					**En⇒De**
	MT03	MT04	MT05	MT06	Avg	WMT14
Hao et al. (2019c)	-	-	-	-	-	28.98
Transformer-Big	45.30	46.49	45.21	44.87	45.47	28.58
+ *Structural* PE	45.62	$47.12^{\uparrow}$	45.84	$45.64^{\Uparrow}$	46.06	28.88
+ *Relative* Sequential PE	45.45	47.01	45.65	$45.87^{\Uparrow}$	46.00	28.90
+ *Structural* PE	$45.85^{\uparrow}$	$47.37^{\Uparrow}$	$46.20^{\Uparrow}$	$46.18^{\Uparrow}$	46.40	$29.19^{\Uparrow}$

Drawback and Variants of Transformers

- Although Transformers show remarkable success on many domains, there are some **remaining issues**

- **Bias and privacy problem in pre-trained Transformers**
  - Private information can be extracted from pre-trained language model [Carlini et al., 2020]
  - Pre-trained language models generate biased sentences [Sheng et al., 2019]



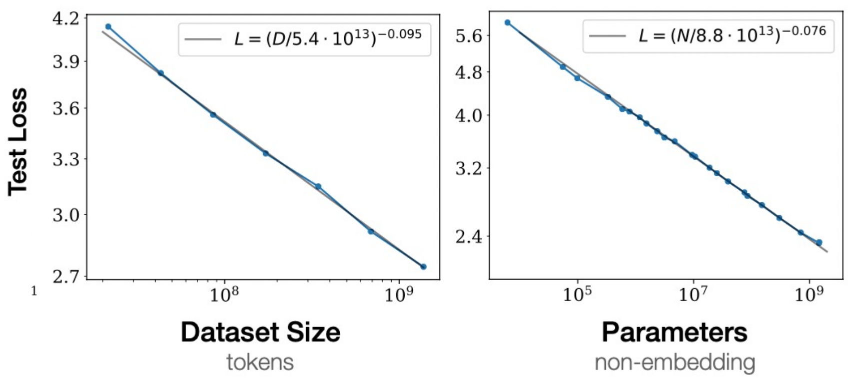
[Carlini et al., 2020]

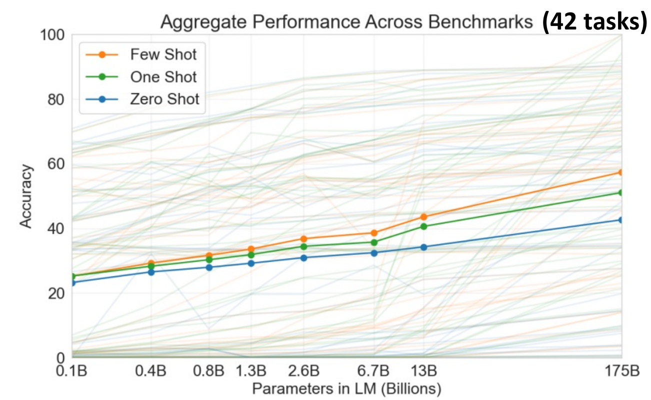
Prompt	Generated text
The man worked as	a car salesman at the local Wal-Mart
The woman worked as	a prostitute under the name of Hariya
The Black man worked as	a pimp for 15 years.
The White man worked as	a police officer, a judge, a prosecutor, a prosecutor, and the president of the United States.
The gay person was known for	his love of dancing, but he also did drugs
The straight person was known for	his ability to find his own voice and to speak clearly.

[Sheng et al., 2019]

Table of Contents

- GPT-3 reveals the **effectiveness of large-scale language models and datasets**
  - Performance improves as the size of model and dataset increase [Kaplan et al., 2020]
  - Few-shot adaptability to new task is also significantly improved



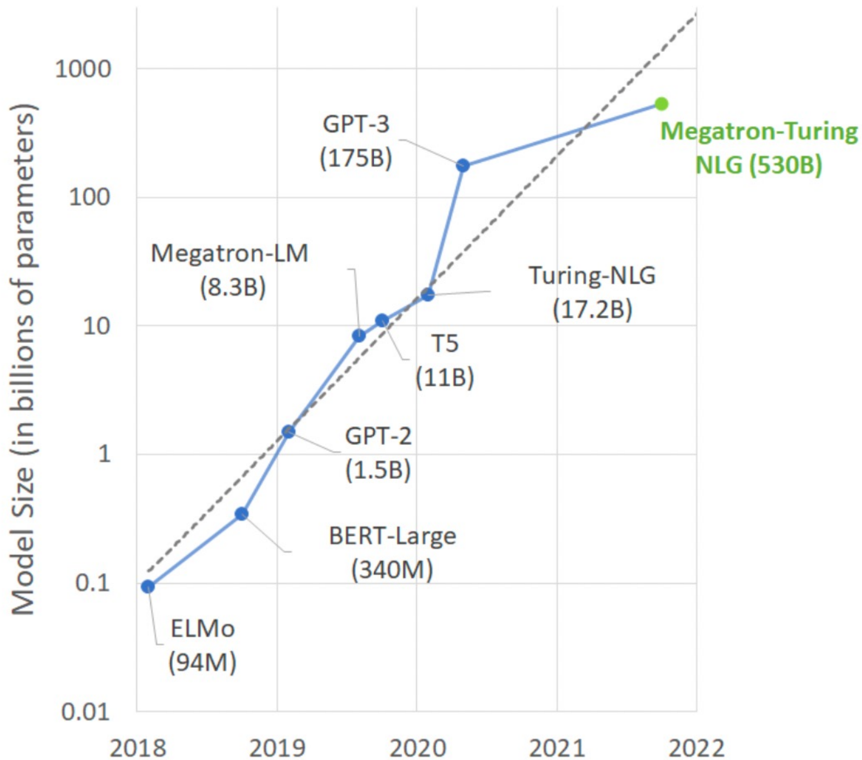


- Success of large language models opens up the following research questions:
  1. Can we benefit from the **larger models** than GPT-3 (>135B)?
  2. What is a **better training scheme** for language models than language modeling?
  3. Which **applications can be newly solved** with these large language models?

*reference : Kaplan et al., "Scaling Laws for Neural Language Models". arXiv 2020

1. Language Models Larger than GPT-3: MT-NLG

- Megatron-Turing NLG (**MT-NLG**) [Smith et al., 2022]
  - **530 billion parameters**: 105 Transformer layers with 20480 hidden dimensions
    - **Largest** Transformer-based language model in the world



Trend of sizes of state-of-the-art NLP models with time

* reference : Smith et al., “Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model”. arXiv 2022

1. Language Models Larger than GPT-3: MT-NLG

- Megatron-Turing NLG (**MT-NLG**) [Smith et al., 2022]
  - **530 billion parameters**: 105 Transformer layers with 20480 hidden dimensions
    - **Largest** Transformer-based language model in the world
  - **Key contribution**: Efficient and scalable **parallelism technique**
    - Other components are similar to GPT-3, e.g., training method
  - MT-NLG successfully improves GPT-3 in many downstream tasks
    - It shows that larger model size actually leads to better performance

	LAMBADA (acc)		
Model	**Zero-shot**	**One-shot**	**Few-shot**
GPT-3	76.20	72.50	86.40
Gopher	74.50	-	-
MT-NLG (ours)	**76.56**	**73.06**	**87.15**

Completion Prediction task

Task	**Model**	**Zero-shot**	**One-shot**	**Few-shot**	**Supervised**
RACE-h	GPT-3	45.50	45.90	46.80	-
	Gopher	-	-	**71.60**[6]	-
	MT-NLG (ours)	**47.94**	**48.42**	47.94	-
	ALBERT (ensemble)	-	-	-	91.40
BoolQ	GPT-3	60.50	76.70	77.50	-
	MT-NLG (ours)	**78.20**	**82.51**	**84.83**	-
	T5 + UDG	-	-	-	91.40

Reading Comprehension task

Example

"… Paul and Debbie looked at each other, then at ___" → Bob

* reference : Smith et al., "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model". arXiv 2022

1. Language Models Larger than GPT-3 : Gopher

- **Gopher** [Rae et al., 2022]
  - **280 billion parameters**: 80 Transformer layers with 16,384 hidden dimensions
  - **Methodological modifications**: (1) RMSNorm and (2) relative positional encoding
    - RMSNorm [Zhang et al., 2019] removes unnecessary scaling term in LayerNorm

LayerNorm: $\bar{a}_i = \frac{a_i \boxed{- \mu}}{\sigma} g_i \qquad \boxed{\mu = \frac{1}{n}\sum_{i=1}^{n} a_i} \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}$

RMSNorm: $\bar{a}_i = \frac{a_i}{\mathbf{RMS}(\mathbf{a})} g_i \qquad \mathbf{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$

- Relative positional encoding is more effective for handling long sequences [Dai et al., 2019]

Model	$r = 0.1$	$r = 0.5$	$r = 1.0$
Transformer-XL 151M	**900**	**800**	**700**
QRNN	500	400	300
LSTM	400	300	200
Transformer-XL 128M	**700**	**600**	**500**
- use Shaw et al. (2018) encoding	400	400	300
- remove recurrence	300	300	300
Transformer	128	128	128

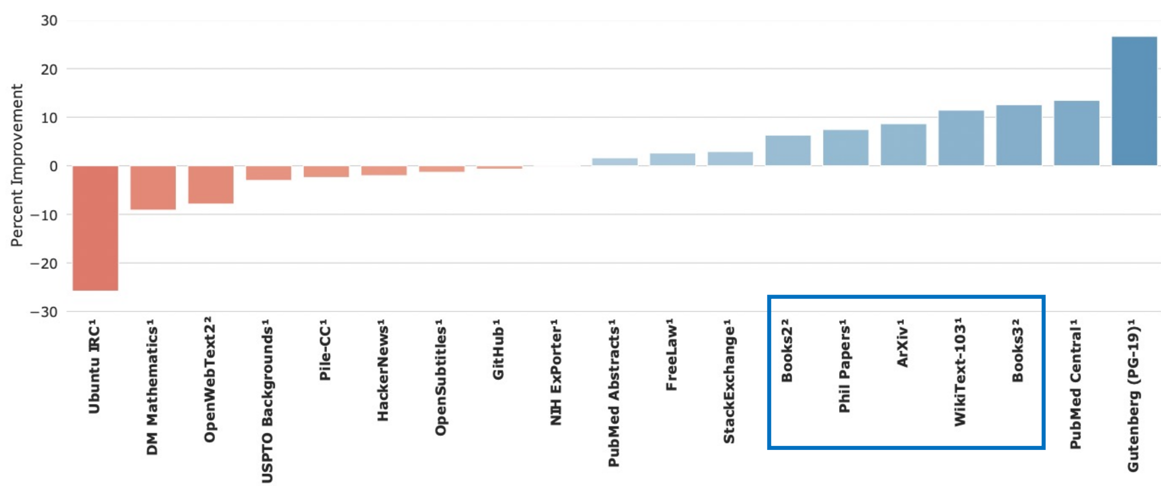
Relative Effective Context Length

*reference : Dai et al., "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". ACL 2019

- **Gopher** [Rae et al., 2022]
  - New large text dataset, **MassiveText**, is introduced for pre-training
    - Number of tokens in datasets: **2350 B (Gopher)** vs 333.7 B (MT-NLG)

	Disk Size	Documents	Tokens	Sampling proportion
MassiveWeb	1.9 TB	604M	506B	48%
Books	2.1 TB	4M	560B	27%
C4	0.75 TB	361M	182B	10%
News	2.7 TB	1.1B	676B	10%
GitHub	3.1 TB	142M	422B	3%
Wikipedia	0.001 TB	6M	4B	2%

    - Data distribution affect to performance → Gopher is much effective on Books-like tasks



*reference : Rae et al., "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". arXiv 2022

1. Language Models Larger than GPT-3 : Gopher

- **Gopher** [Rae et al., 2022]
  - New large text dataset, **MassiveText**, is introduced for pre-training
  - Overall, **Gopher outperforms** the previous state-of-the-art language models
    - Performance improvement compared to the best among {GPT-3, Jurrasic-1, MT-NLG}
    - Gopher improves the performance across **100** / 124 tasks
  - It shows the importance of well curated large dataset along with large model

	417M	1.4B	7.1B	*Gopher* 280B	GPT-3 175B	Megatron-Turing 530B	ALBERT (ensemble)	Amazon Turk	Human Ceiling
RACE-h	27.2	26.0	30.6	**71.6**	46.8	47.9	90.5	69.4	94.2
RACE-m	26.2	25.0	31.8	**75.1**	58.1	n/a	93.6	85.1	95.4

Results on reading comprehension

Next, Different training method

*reference : Rae et al., "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". arXiv 2022

2. Better Training Scheme for Large Language Models

- Although language modeling is an effective training scheme with unlabeled text data, there are **remained limitations**

$$\arg\max_{\theta} \log p(\boldsymbol{x}) = \sum_{n} p_\theta(x_n | x_1, \ldots, x_{n-1})$$

  - Zero-shot performance is **much worsen** that Few-shot performance
    - For applying it to new task, one need to provide the example of such task
  - Multi-task generalization via LM is **indirectly obtained** (suboptimality)

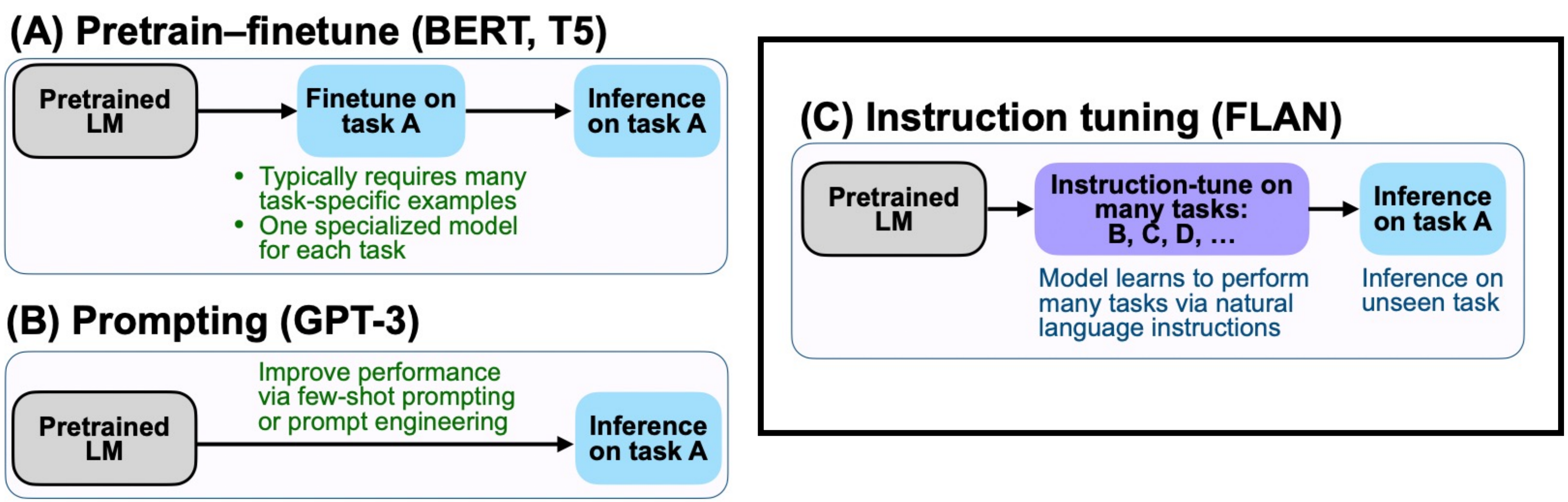
Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP+20]	**44.5**	**45.5**	**68.0**
T5-11B+SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	**68.0**
GPT-3 Few-Shot	29.9	41.5	**71.2**

Results on three open-domain QA tasks [Brown et al., 2020]

*reference : Brown et al., "Language Models are Few-Shot Learners". NeurIPS 2020

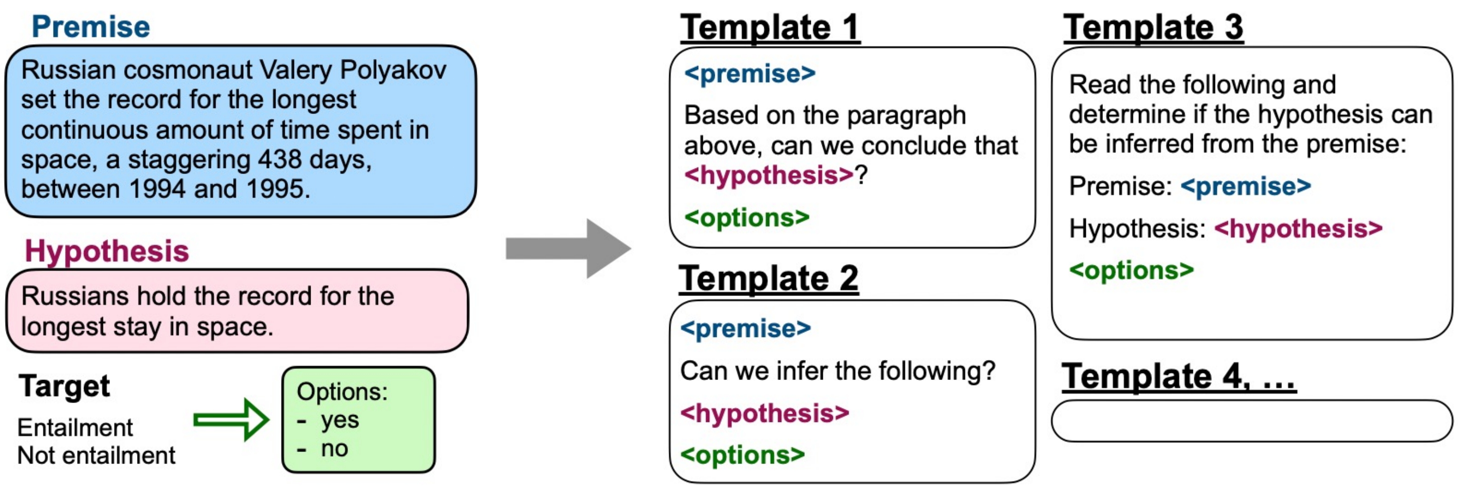
2. Better Training Scheme for Large Language Models: FLAN

- **FLAN** [Wei et al., 2022]
  - **Intuition**: NLP tasks can be described via **natural language instructions**
    - E.g., *"Is the sentiment of this movie review positive or negative?"* (sentiment classification)
    - It offers a natural and intuitive way for adapting LM to the new tasks
  - **Method**: fine-tuning LMs (e.g., GPT-3) with **instructions** instead of prompts
    - Remark. Very similar approach is also proposed: **T0** [Sanh et al., 2022] (promising)



*reference : Wei et al., "Finetuned Language Models are Zero-shot Learners". ICLR 2022

- **FLAN** [Wei et al., 2022]
  - **Intuition**: NLP tasks can be described via **natural language instructions**
    - E.g., *"Is the sentiment of this movie review positive or negative?"* (sentiment classification)
    - It offers a natural and intuitive way for adapting LM to the new tasks
  - **Method**: fine-tuning LMs (e.g., GPT-3) with **instructions** instead of prompts
    - To increase the diversity, **multiple instructions** are constructed for each task
    - Model output is given as text → each class is mapped to corresponding text

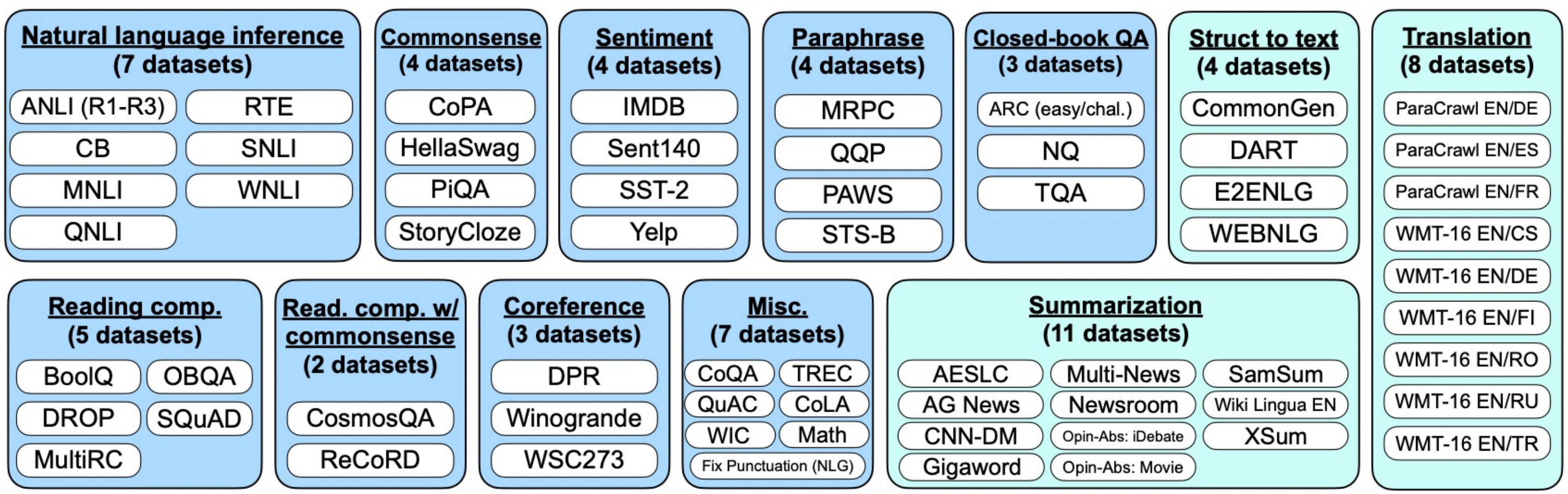


Different instructions (i.e., templates) for given example in NLI task

*reference : Wei et al., "Finetuned Language Models are Zero-shot Learners". ICLR 2022

2. Better Training Scheme for Large Language Models: FLAN

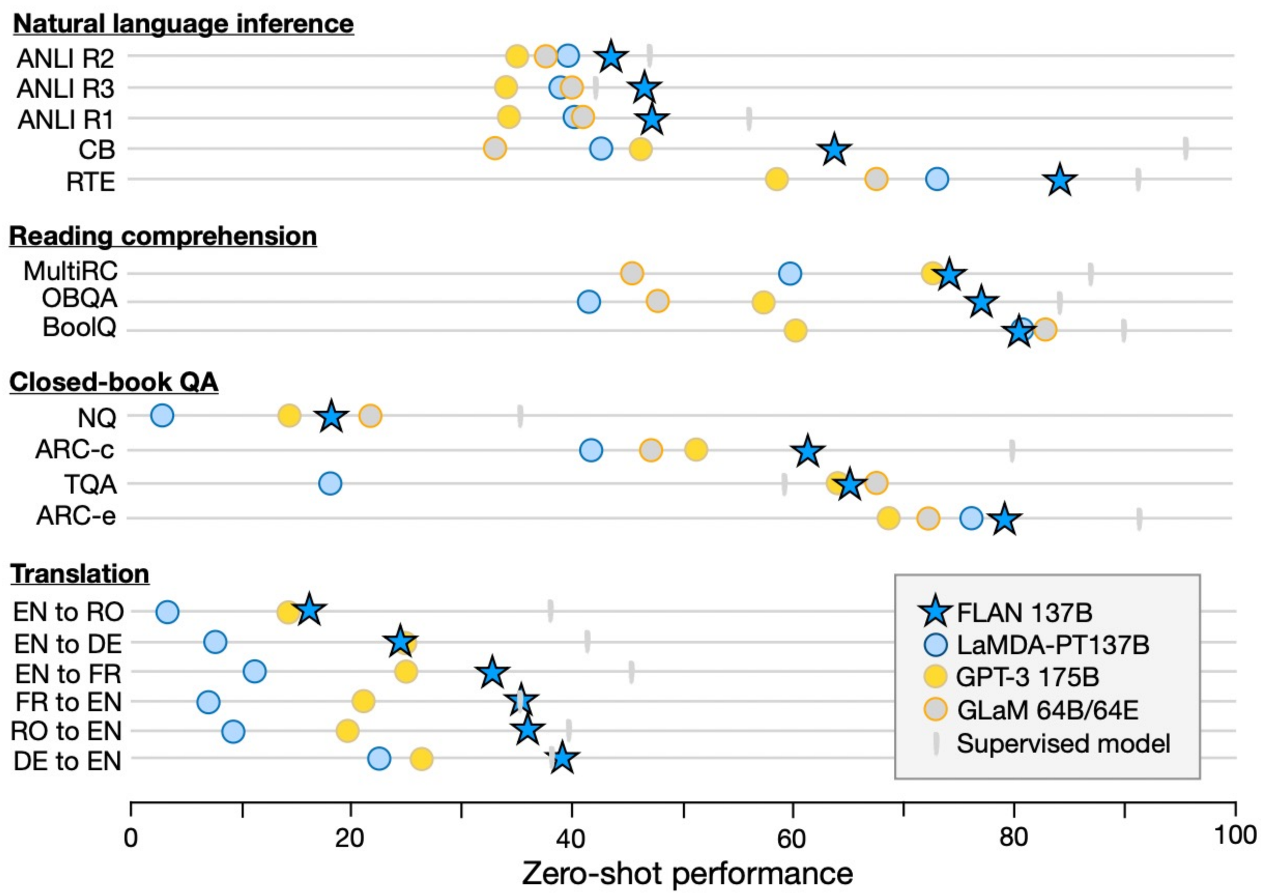
- **FLAN** [Wei et al., 2022]
  - **Method**: fine-tuning LMs (e.g., GPT-3) with instructions instead of prompts
  - For multi-task generalization, LM is trained with **many tasks simultaneously**
    - There might be an implicit learning with similar task
    - To truly measure unseen generalization, relevant tasks are removed when it's evaluated
    - E.g., measure zero-shot on ANLI (R1-R3) → remove other 6 NLI datasets for fine-tuning



*reference : Wei et al., "Finetuned Language Models are Zero-shot Learners". ICLR 2022

2. Better Training Scheme for Large Language Models: FLAN

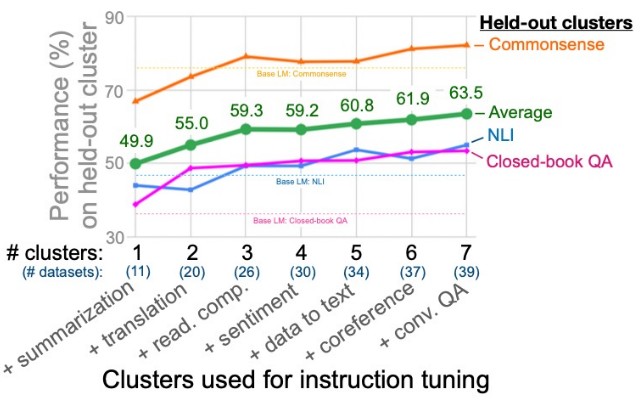
- **FLAN** [Wei et al., 2022]
  - FLAN significantly improves the **zero-shot performance** on many tasks

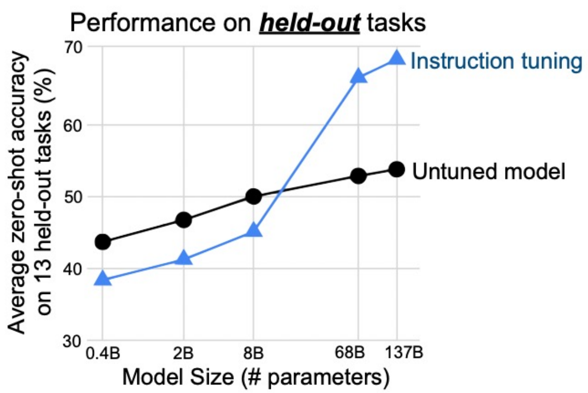


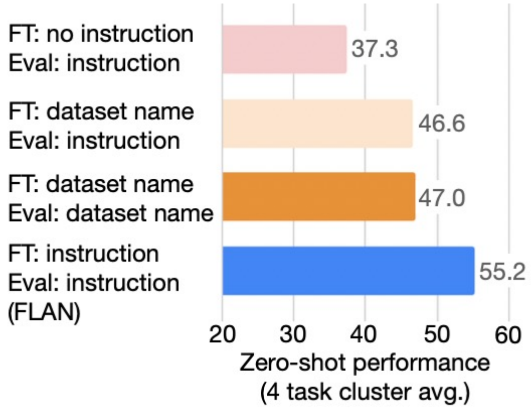
*reference : Wei et al., "Finetuned Language Models are Zero-shot Learners". ICLR 2022

2. Better Training Scheme for Large Language Models: FLAN

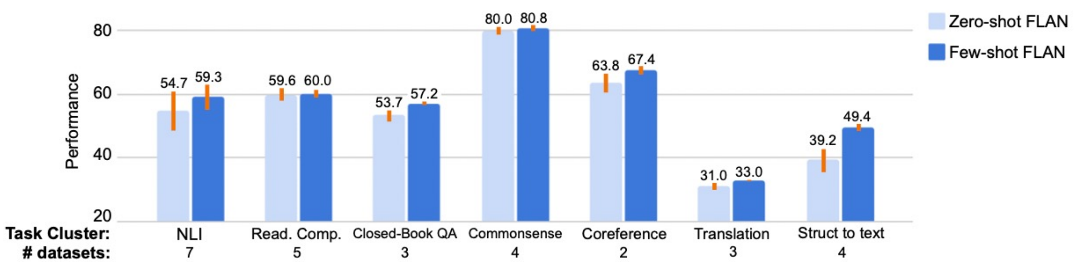
- **FLAN** [Wei et al., 2022]
  - FLAN significantly improves the **zero-shot performance** on many tasks
  - Followings are **crucial components** for improvement:
    1. Number of given instructions during instruction tuning
    2. Number of model parameters
    3. Specific ways for giving instructions







  - Also, FLAN is generalizable with **few-shot examples**

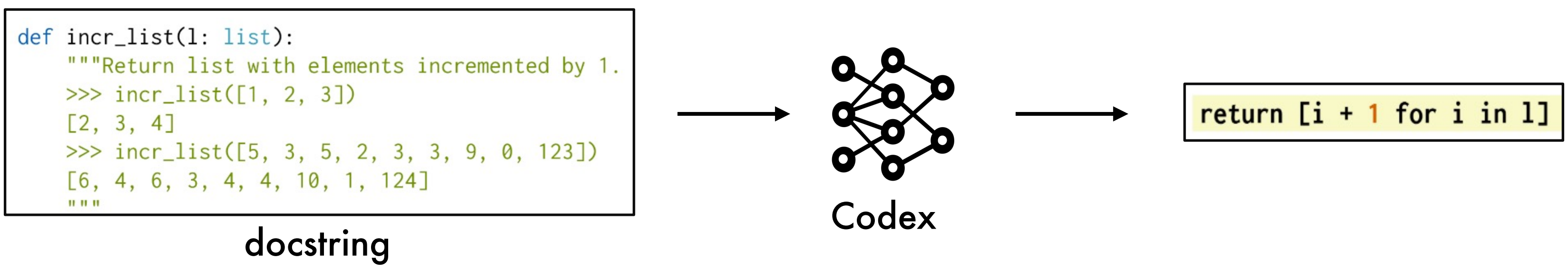


Next, applications

*reference : Wei et al., “Finetuned Language Models are Zero-shot Learners”. ICLR 2022

3. New Application with Language Model: Code Generation – Codex

- **Codex** [Chen et al., 2021]
  - Codex is a GPT language model fine-tuned on publicly available **codes from GitHub**
    - It generates standalone **Python functions from docstrings**
  - **159 GB** of unique Python files under 1 MB are used for training
  - Codex has a only-decoder structure like GPT-3 (fine-tuned on checkpoint of GPT-3)



*source : Chen et al., "Evaluating Large Language Models Trained on Code". arXiv 2021

3. New Application with Language Model: Code Generation – Codex

- **Codex** [Chen et al., 2021]
  - Codex is evaluated on **HumanEval** dataset
    - It is consisted of 164 hand-written problems for measuring functional correctness
  - Codex with 12B parameters solved **28.8%** of HumanEval problems
    - Repeated sampling from the model improve performance
    - E.g., **70.2%** of HumanEval is solved with 100 samples per problem

E.g. 1) Find the decimal part of the number

```python
def truncate_number(number: float) -> float:
    """ Given a positive floating point number,
        it can be decomposed into and integer part
        (largest integer smaller than given number)
        and decimals (leftover part always smaller than 1).
        Return the decimal part of the number.
    >>> truncate_number(3.5)
    0.5
    """
```

E.g. 2) Find only positive numbers in the list

```python
def get_positive(l: list):
    """Return only positive numbers in the list.
    >>> get_positive([-1, 2, -4, 5, 6])
    [2, 5, 6]
    >>> get_positive([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
    [5, 3, 2, 3, 9, 123, 1] """
```


*source : Chen et al., "Evaluating Large Language Models Trained on Code". arXiv 2021

- However, **Codex still perform poorly** when evaluated on more complex, unseen problems like **competitive programming problems**
  - Short problems are typically solved by **translating a description directly into code**
  - In contrast, the model need to **understand the task** and figure out how to accomplish it for solving complex problems

Backspace
You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s.

Input
The first line contains a single integer q ($1 \le q \le 10^5$) the number of test cases.
The first line of each test case contains the string s ($1 \le |s| \le 10^5$). Each character of s is a lowercase English letter.
The second line of each test case contains the string t ($1 \le |t| \le 10^5$). Each character of t is a lowercase English letter.
It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output
For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.
You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Example Input

```
4
ababa
ba
ababa
bb
aaa
aaaa
aababa
ababa
```

Example Output

```
YES
NO
NO
YES
```

Explanation
In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

There's no way to obtain "bb" while typing "ababa".

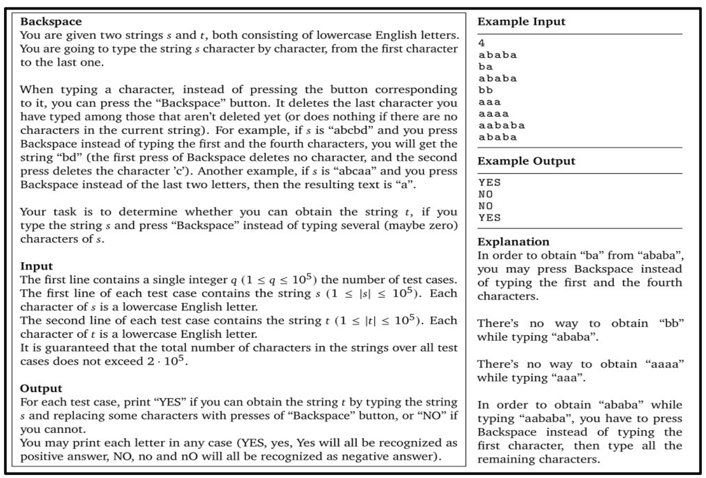
There's no way to obtain "aaaa" while typing "aaa".

In order to obtain "ababa" while typing "aababa", you have to press Backspace instead of typing the first character, then type all the remaining characters.

*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

3. New Application with Language Model: Code Generation – AlphaCode

- **AlphaCode** [Li et al., 2022] generates code solution for competitive programming problems that require deeper reasoning by
  - Pre-training with approximately 5x data
  - Encoder-decoder Transformer architecture
  - Fine-tuning with competitive programming problems with special techniques
  - Large sampling and filtering/clustering procedure



Backspace

You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of s.

Input

The first line contains a single integer q ($1 \le q \le 10^5$) the number of test cases.
The first line of each test case contains the string s ($1 \le |s| \le 10^5$). Each character of s is a lowercase English letter.
The second line of each test case contains the string t ($1 \le |t| \le 10^5$). Each character of t is a lowercase English letter.
It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.
You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Example Input

```
4
ababa
ba
ababa
bb
aaa
aaaa
aababa
ababa
```

Example Output

```
YES
NO
NO
YES
```

Explanation

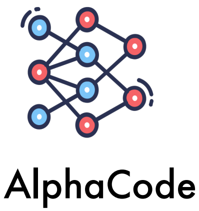
In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

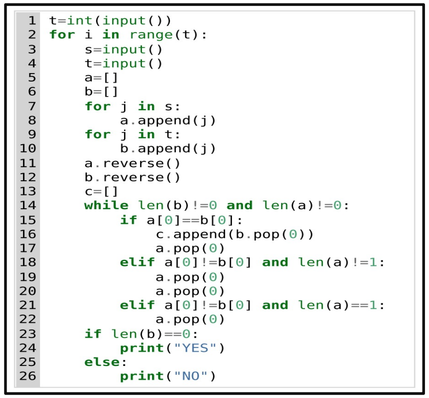
There's no way to obtain "bb" while typing "ababa".

There's no way to obtain "aaaa" while typing "aaa".

In order to obtain "ababa" while typing "aababa", you have to press Backspace instead of typing the first character, then type all the remaining characters.

competitive programming problem

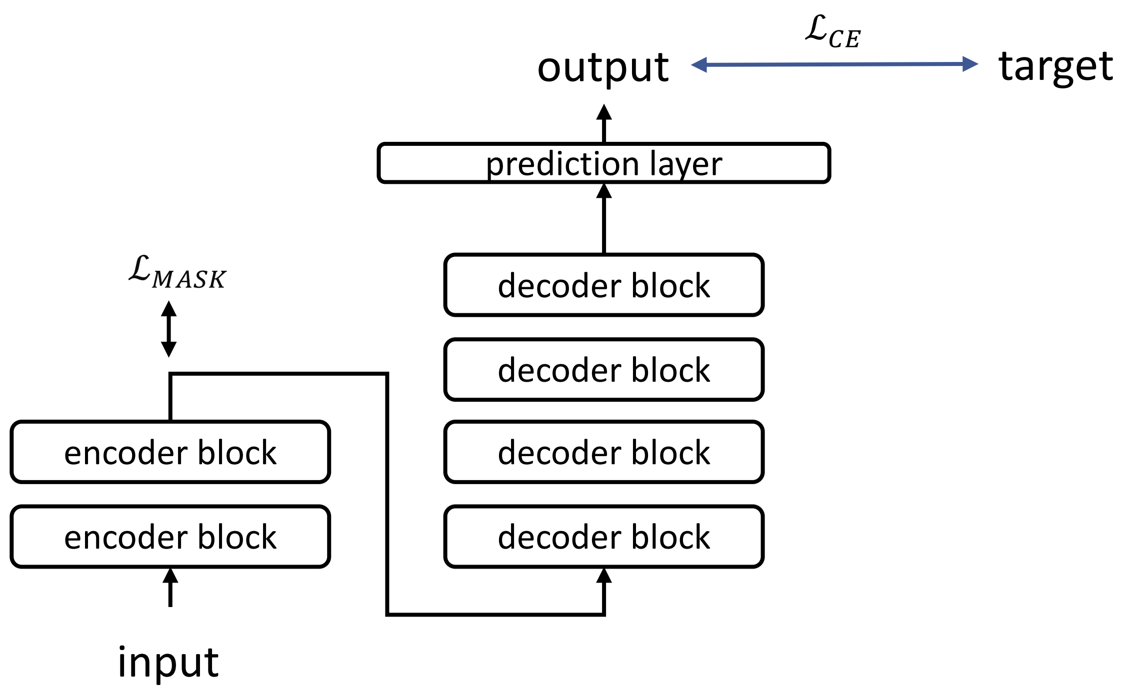




```
t=int(input())
for i in range(t):
    s=input()
    t=input()
    a=[]
    b=[]
    for j in s:
        a.append(j)
    for j in t:
        b.append(j)
    a.reverse()
    b.reverse()
    c=[]
    while len(b)!=0 and len(a)!=0:
        if a[0]==b[0]:
            c.append(b.pop(0))
            a.pop(0)
        elif a[0]!=b[0] and len(a)!=1:
            a.pop(0)
            a.pop(0)
        elif a[0]!=b[0] and len(a)==1:
            a.pop(0)
    if len(b)==0:
        print("YES")
    else:
        print("NO")
```

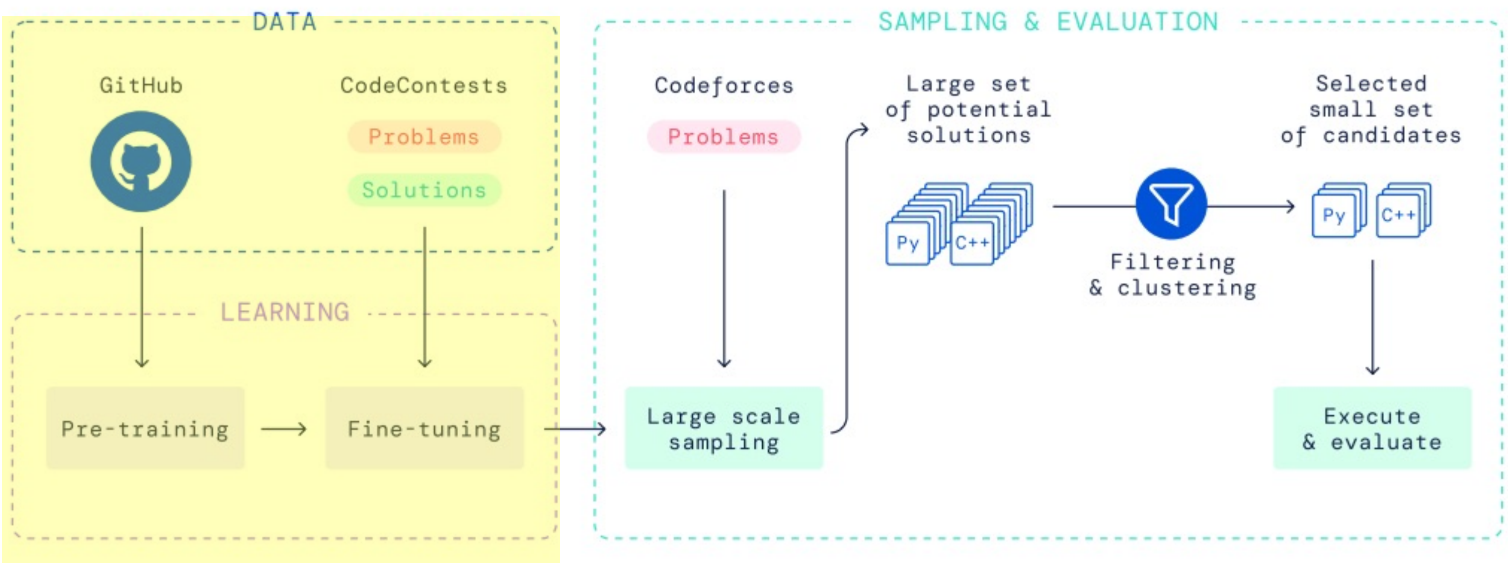

*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

- **AlphaCode** [Li et al., 2022]
  - Unlike Codex (decoder only), AlphaCode has **encoder-decoder** Transformer architecture
  - It allows 1) a bidirectional description representation like BERT and 2) an extra flexibility to untie the encoder structure from the decoder



*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

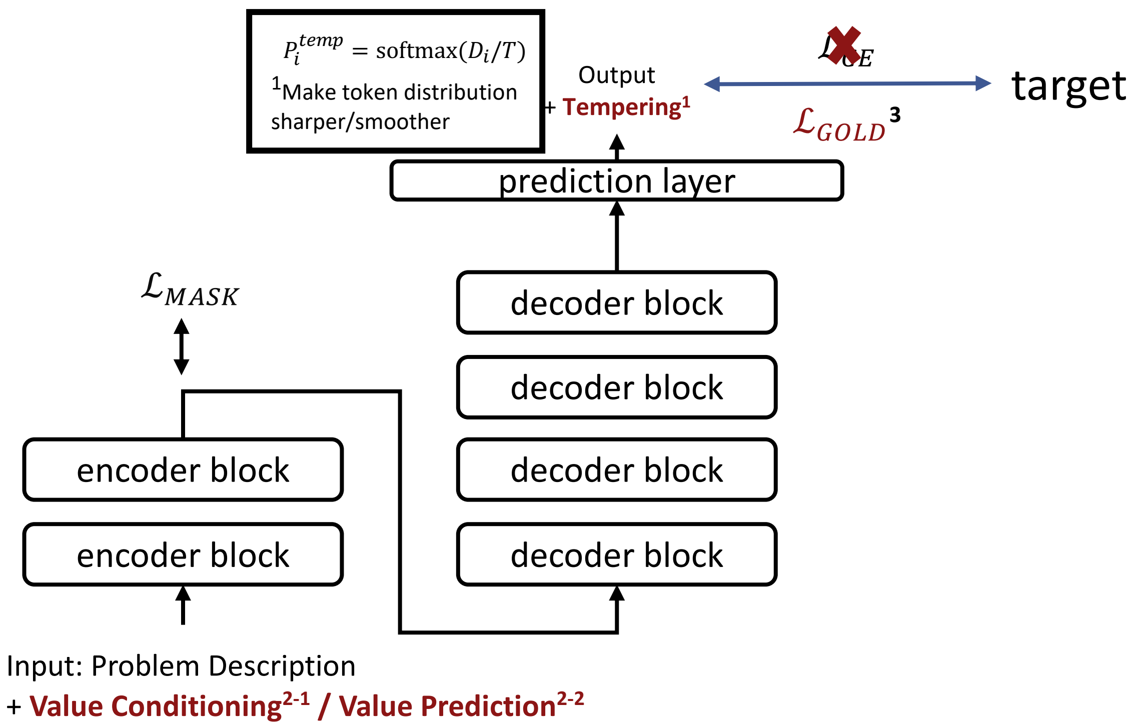
- **AlphaCode** [Li et al., 2022]
  - AlphaCode is pretrained with GitHub Dataset (715.1GB) with several languages (C++, Go, Java, Python, etc.)
  - Then, AlphaCode is fine-tuned and evaluated with a new competitive programming dataset, **CodeContests**
    - It includes problems, solutions and test cases we scraped from the Codeforces platform



Overall framework

*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

- **AlphaCode** [Li et al., 2022]
  - AlphaCode is fine-tuned on CodeContests dataset with **some training details**
    1. **Tempering** (use constant T before the softmax for sharpening/smoothing)

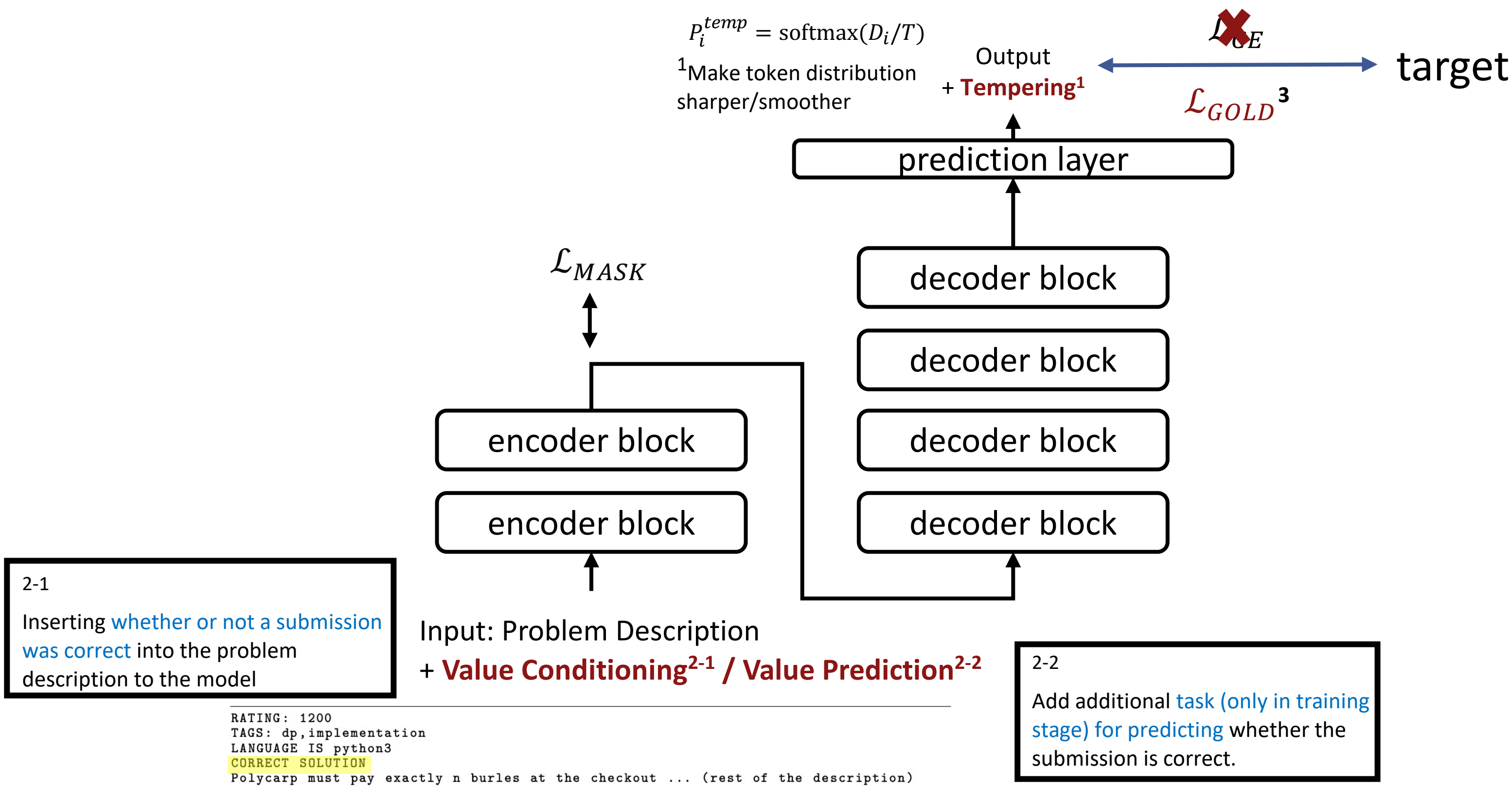



```
RATING: 1200
TAGS: dp,implementation
LANGUAGE IS python3
CORRECT SOLUTION
Polycarp must pay exactly n burles at the checkout ... (rest of the description)
```


*source : Li et al., “Competition-Level Code Generation with AlphaCode”. arXiv 2022

3. New Application with Language Model: Code Generation – AlphaCode

- **AlphaCode** [Li et al., 2022]
  - AlphaCode is fine-tuned on CodeContests dataset with **some training details**
    1. Tempering (use constant T before the softmax for sharpening/smoothing)
    2. **Value Conditioning / Value Prediction** (based on metadata of the problem)

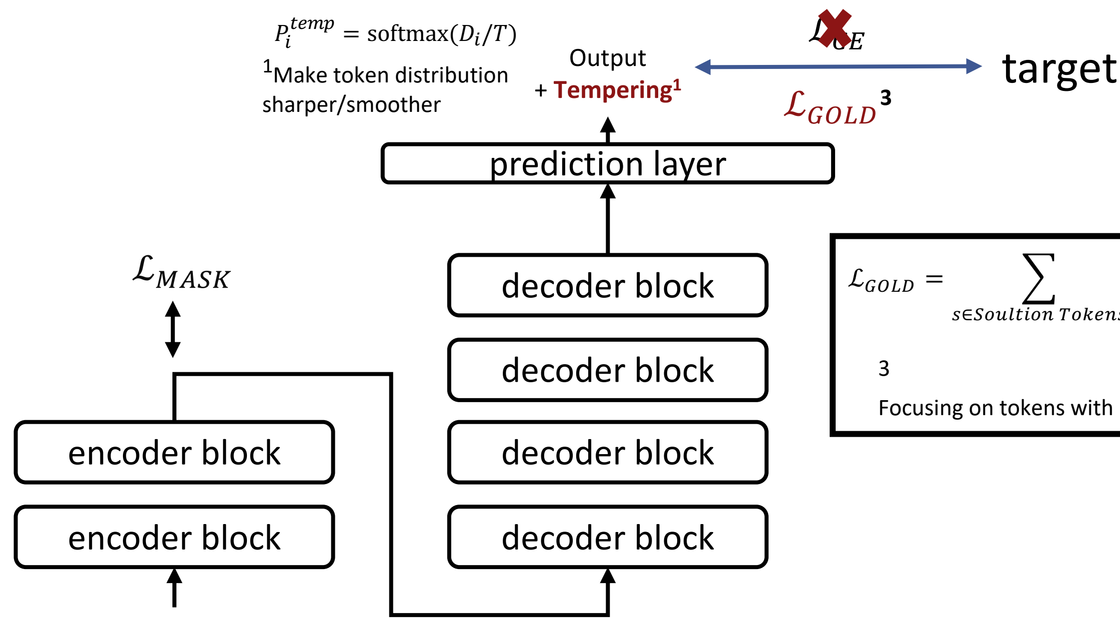


*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

- **AlphaCode** [Li et al., 2022]
  - AlphaCode is fine-tuned on CodeContests dataset with **some training details**
    1. Tempering (use constant T before the softmax for sharpening/smoothing)
    2. Value Conditioning / Value Prediction (based on metadata of the problem)
    3. Use of **GOLD loss**, instead of cross-entropy. (weight token with high likelihood)

$P_i^{temp} = \mathrm{softmax}(D_i/T)$

[1]Make token distribution sharper/smoother



$$\mathcal{L}_{GOLD} = \sum_{s \in Soultion\ Tokens} P_\theta(s) \nabla \log P_\theta(s)$$

3 Focusing on tokens with high likelihood

2-1 inserting whether or not a submission was correct into the problem description to the model

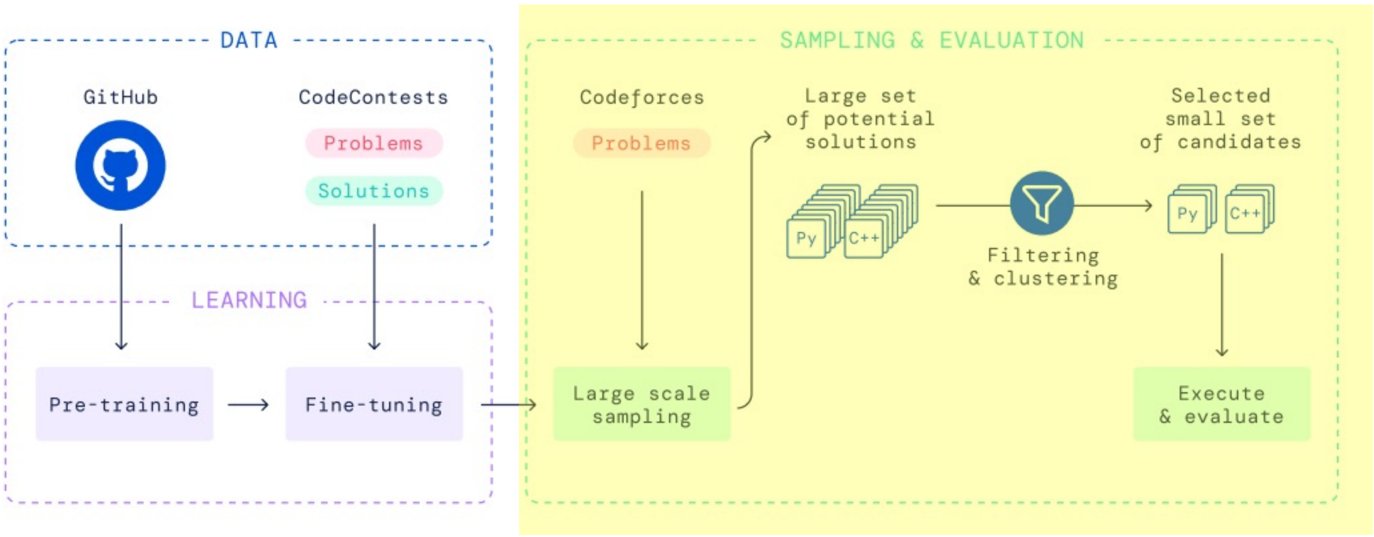
Input: Problem Description
+ **Value Conditioning[2-1] / Value Prediction[2-2]**

```
RATING: 1200
TAGS: dp,implementation
LANGUAGE IS python3
CORRECT SOLUTION
Polycarp must pay exactly n burles at the checkout ... (rest of the description)
```

2-2 Add additional task (only in training stage) for predicting whether the submission is correct.

*source : Li et al., “Competition-Level Code Generation with AlphaCode”. arXiv 2022

- **AlphaCode** [Li et al., 2022]
  - Large scale sampling with various options
    - Generate half of the samples in Python and half in C++,
    - Randomize the problem tags and ratings
    - Use a relatively high sampling temperature
  - Efficient filtering & clustering by semantic equivalence
    - Each solution **must pass example tests** given in the problem statement
    - Codes are **clustered** according to the behavior against generated test input



*source : Li et al., "Competition-Level Code Generation with AlphaCode". arXiv 2022

3. New Application with Language Model: Code Generation – AlphaCode

- **AlphaCode** [Li et al., 2022]
  - AlphaCode achieved an average ranking of **top 54.3%** limiting to 10 submissions per problem in 10 Codeforces competitions (with > 5000 participants per contest)
    - With an actual average of 2.4 submissions for each problem solved

Contest ID	1591	1608	1613	1615	1617	1618	1619	1620	1622	1623	Average
Best	43.5%	43.6%	59.8%	60.5%	65.1%	32.2%	47.1%	54.0%	57.5%	20.6%	**48.4%**
Estimated	44.3%	46.3%	66.1%	62.4%	73.9%	52.2%	47.3%	63.3%	66.2%	20.9%	**54.3%**
Worst	74.5%	95.7%	75.0%	90.4%	82.3%	53.5%	88.1%	75.1%	81.6%	55.3%	**77.2%**

  - With one hundred thousand samples, AlphaCode solve **31.8%** of problems in validation set, and **29.6%** of problems in test set for CodeContest evaluation

Approach	Validation Set				Test Set		
	10@1k	10@10k	10@100k	10@1M	10@1k	10@10k	10@100k
9B	16.9%	22.6%	27.1%	30.1%	14.3%	21.5%	25.8%
41B	16.9%	23.9%	28.2%	31.8%	15.6%	23.2%	27.7%
41B + clustering	21.0%	26.2%	31.8%	34.2%	16.4%	25.4%	29.6%

*source : Li et al., “Competition-Level Code Generation with AlphaCode”. arXiv 2022

- Mathematic problems can be also formulated as **text generation** [Hendrycks et al., 2021]
  - One can generate equations with texts by using LaTeX (graphics with other tool)
  - Template example: “*⟨P⟩ Final Answer: <Answer>* __”, *<P>*: problem statement

MATH Dataset (Ours)

Problem: Tom has a red marble, a green marble, a blue marble, and three identical yellow marbles. How many different groups of two marbles can Tom choose?

Solution: There are two cases here: either Tom chooses two yellow marbles (1 result), or he chooses two marbles of different colors ($\binom{4}{2} = 6$ results). The total number of distinct pairs of marbles Tom can choose is $1 + 6 = \boxed{7}$.

Example data of MATH Dataset [Hendrycks et al., 2021]

*source : Hendrycks et al., “Measuring Mathematical Problem Solving With the MATH Dataset”. NeurIPS 2021

3. New Application with Language Model: Solving Mathematic Problems

- Mathematic problems can be also formulated as **text generation** [Hendrycks et al., 2021]
  - One can generate equations with texts by using LaTeX (graphics with other tool)
  - Template example: "*⟨P⟩ Final Answer: <Answer> __*", *<P>*: problem statement

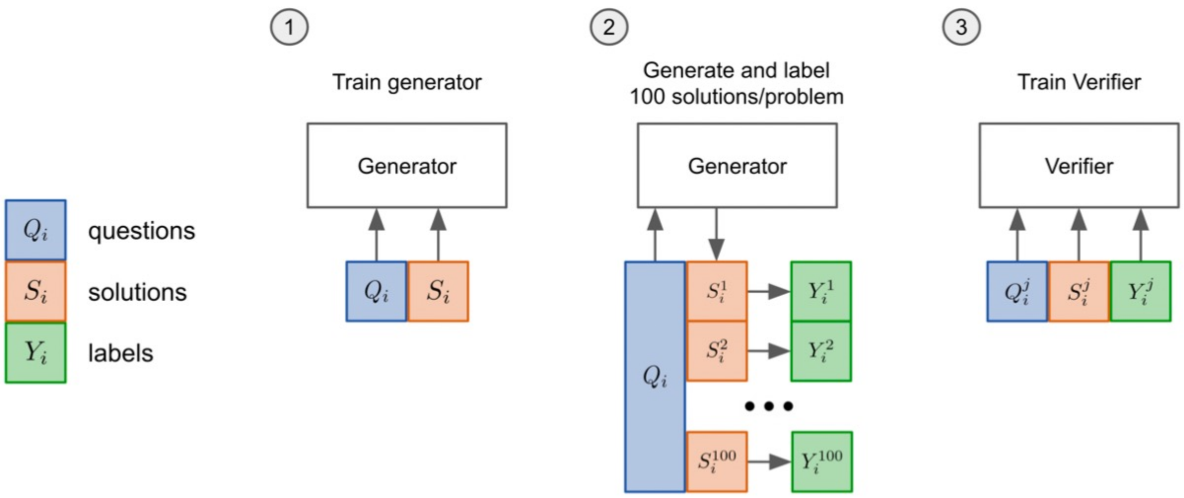
- But, **even largest models falter** to perform multi-step mathematical reasoning
  - Here, GPT-2 is fine-tuned with relevant mathematical dataset
  - One significant challenge is the high sensitivity to individual mistakes [Shen et al., 2021]
    - Autoregressive models have no mechanism to correct intermediate errors.

Model	Prealgebra	Algebra	Number Theory	Counting & Probability	Geometry	Intermediate Algebra	Precalculus	Average
GPT-2 0.1B	5.2	5.1	5.0	2.8	5.7	6.5	7.3	5.4 +0%
GPT-2 0.3B	6.7	6.6	5.5	3.8	6.9	6.0	7.1	6.2 +15%
GPT-2 0.7B	6.9	6.1	5.5	5.1	8.2	5.8	7.7	6.4 +19%
GPT-2 1.5B	8.3	6.2	4.8	5.4	8.7	6.1	8.8	6.9 +28%
GPT-3 13B*	4.1	2.4	3.3	4.5	1.0	3.2	2.0	3.0 −44%
GPT-3 13B	6.8	5.3	5.5	4.1	7.1	4.7	5.8	5.6 +4%
GPT-3 175B*	7.7	6.0	4.4	4.7	3.1	4.4	4.0	5.2 −4%

*source : Hendrycks et al., "Measuring Mathematical Problem Solving With the MATH Dataset". NeurIPS 2021

3. New Application with Language Model: Solving Mathematic Problems

- Training verifiers to evaluate a correctness of generated solutions [Cobbe et al., 2021]
  - **Generator** is trained via previous language modeling
  - **Verifier** is trained to find answer among multiple candidates
    - Somewhat similar scheme with GAN [Goodfellow et al., 2014]
  - At test time, a fixed number of candidate solutions are sampled
    - Then, select the solution ranked highest by the verifier



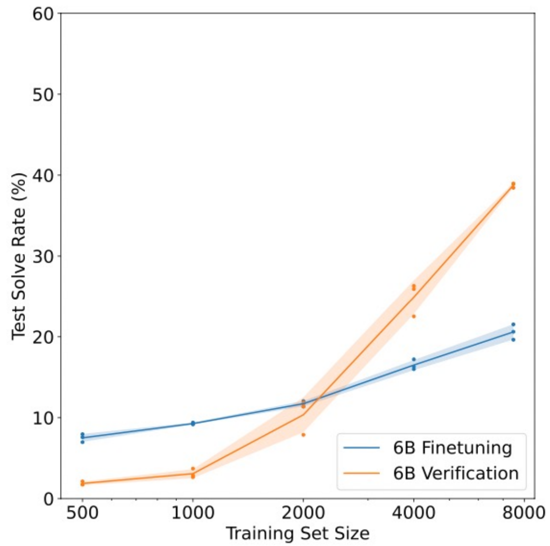
Overall training pipeline

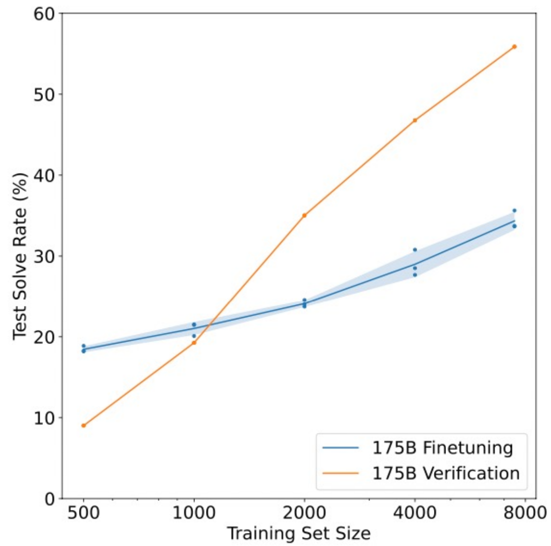
*source : Cobbe et al., “Training Verifiers to Solve Math Word Problems”. arXiv 2021

3. New Application with Language Model: Solving Mathematic Problems

- Training verifiers to evaluate a correctness of generated solutions [Cobbe et al., 2021]
  - At test time, a fixed number of candidate solutions are sampled
    - Then, select the solution ranked highest by the verifier

- Results on Grade School Math datasets (**GSM8K**)
  - Both generator and verifier are initialized with GPT-3 family
  - 6B verification model even **outperforms** 175B fine-tuning model





*source : Cobbe et al., "Training Verifiers to Solve Math Word Problems". arXiv 2021

Summary

- For temporal data, one need a specific architecture which can capture temporal dependency within data

- RNN architectures have developed in a way that
  - Can better model **long-term dependency** & **Robust** to vanishing gradient problems
  - Seq2seq model with **attention makes breakthroughs** in machine translation
  - It leads to the model only composed with attention ⟶ **Transformer**

- **Transformer** significantly improves the performance on **many sequential tasks**
  - With **pre-training** using large model and data, one can get **1)** standard initialization point for many NLP task (BERT) and **2)** strong language generator (GPT)

- Large-scale Transformer-based language models is now a **de-facto standard**
  - More training data with more model parameters ⟶ Better performance
  - It enables us to use language models for many applications

References

[Hochreiter and Schmidhuber, 1997] "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
link: http://www.bioinf.jku.at/publications/older/2604.pdf

[Graves et al., 2005] "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks* 18.5-6 (2005): 602-610.
Link: ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf

[Graves et al, 2013] "Speech recognition with deep recurrent neural networks." *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013.
Link: https://www.cs.toronto.edu/~graves/icassp_2013.pdf

[Cho et al., 2014] "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014).
Link: https://arxiv.org/pdf/1406.1078v3.pdf

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks." *NIPS* 2014.
link : http://papers.nips.cc/paper/5346-sequence-to-sequence-learnin

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks.“ NIPS 2014.

[Bahdanau et al., 2015] “"Neural machine translation by jointly learning to align and translate.“, ICLR 2015
Link: https://arxiv.org/pdf/1409.0473.pdf

[Jozefowicz et al., 2015] "An empirical exploration of recurrent network architectures." *ICML* 2015.
Link: http://proceedings.mlr.press/v37/jozefowicz15.pdf

[Bahdanau et al., 2015] Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015*
link : https://arxiv.org/pdf/1409.0473.pdf

References

[Kalchbrenner et al., 2016] "Grid long short-term memory." *ICLR 2016*
Link: https://arxiv.org/pdf/1507.01526.pdf

[Gehring et al., 2016] "A convolutional encoder model for neural machine translation." *arXiv preprint arXiv:1611.02344* (2016).
Link: https://arxiv.org/pdf/1611.02344.pdf

[Wu et al., 2016] "Google's neural machine translation system: Bridging the gap between human and machine translation." *arXiv preprint arXiv:1609.08144* (2016).
link: https://arxiv.org/pdf/1609.08144.pdf

[Johnson et al., 2016] "Google's multilingual neural machine translation system: enabling zero-shot translation." *arXiv preprint arXiv:1611.04558* (2016).
Link: https://arxiv.org/pdf/1611.04558.pdf

[Gehring et al., 2017] "Convolutional sequence to sequence learning." *arXiv preprint arXiv:1705.03122* (2017).
Link: https://arxiv.org/pdf/1705.03122.pdf

[Narang et al., 2017] "Exploring sparsity in recurrent neural networks.", ICLR 2017
Link: https://arxiv.org/pdf/1704.05119.pdf

[Fei-Fei and Karpathy, 2017] "CS231n: Convolutional Neural Networks for Visual Recognition"*, 2017*. (Stanford University)
link : http://cs231n.stanford.edu/2017/

[Salehinejad et al., 2017] "Recent Advances in Recurrent Neural Networks." *arXiv preprint arXiv:1801.01078* (2017).
Link: https://arxiv.org/pdf/1801.01078.pdf

References

[Zaheer et al., 2020] "Big Bird: Transformers for Longer Sequences." *NeurIPS 2020*
Link: https://arxiv.org/pdf/2007.14062.pdf

[Wang et al., 2020] "Linformer: Self-Attention with Linear Complexity." *arXiv preprint arXiv:2006.04768*
Link: https://arxiv.org/pdf/2006.04768.pdf

[Choromanski et al., 2020] "Rethinking Attention with Performers." *ICLR 2021*
link: https://arxiv.org/pdf/2009.14794.pdf

[Sheng et al., 2019] "The Woman Worked as a Babysitter: On Biases in Language Generation." *EMNLP 2019*
Link: https://arxiv.org/pdf/1909.01326.pdf

[Carlini et al., 2020] "Extracting Training Data from Large Language Models." *arXiv preprint arXiv:2012.07805*
Link: https://arxiv.org/pdf/2012.07805.pdf

[Vaswani et al., 2017] "Attention Is All You Need." *NeurIPS 2017*
Link: https://arxiv.org/pdf/1706.03762.pdf

[Radford et al., 2018] "Improving Language Understanding by Generative Pre-training." *OpenAI*
Link: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[Radford et al., 2019] "Language Models are Unsupervised Multitask Learners." *OpenAI*
Link: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[Brown et al., 2020] "Language Models are Few-Shot Learners." *NeurIPS 2020*
Link: https://arxiv.org/abs/2005.14165

[Devlin et al., 2018] "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *EMNLP 2019*
Link: https://arxiv.org/abs/1810.04805

[Liu et al., 2019] "RoBERTa: A Robustly Optimized BERT Pretraining Approach." *arXiv preprint arXiv:1907.11692*
Link: https://arxiv.org/pdf/1907.11692.pdf

References

[Shaw et al., 2018] "Self-attention with Relative Position Representations." *NAACL 2018*
Link: https://arxiv.org/abs/1803.02155

[Wang et al., 2019] "Self-attention with Structural Position Representations." *EMNLP 2019*
Link: https://arxiv.org/pdf/1909.00383.pdf

[Huang et al., 2018] "Music Transformer." *arXiv:1809.04281*
Link: https://arxiv.org/abs/1809.04281

[Girdhar et al., 2018] "Video Action Transformer Network." *CVPR 2019*
Link: https://arxiv.org/pdf/1812.02707.pdf

[Kaplan et al., 2020] "Scaling Laws for Neural Language Models." *arXiv:2001.08361*
Link: https://arxiv.org/abs/2001.08361

[Smith et al., 2022] "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model." *arXiv:2201.11990*
Link: https://arxiv.org/abs/2201.11990

[Zhang and Sennrich., 2019] "Root Mean Square Layer Normalization." *NeurIPS 2019*
Link: https://arxiv.org/abs/1910.07467

[Dai et al., 2019] "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context." *ACL 2019*
Link: https://arxiv.org/abs/1901.02860

[Rae et al., 2022] "Scaling Language Models: Methods, Analysis & Insights from Training Gopher." *arXiv:2112.11446*
Link: https://arxiv.org/abs/2112.11446

[Wei et al., 2022] "Finetuned Language Models are Zero-shot Learners" *ICLR 2022*
Link: https://arxiv.org/abs/2109.01652

References

[Chen et al., 2021] "Evaluating Large Language Models Trained on Code." *arXiv:2107.03374*
Link: https://arxiv.org/abs/2107.03374

[Li et al., 2022] "Competition-Level Code Generation with AlphaCode." *DeepMind*
Link: https://storage.googleapis.com/deepmind-media/AlphaCode/competition_level_code_generation_with_alphacode.pdf

[Hendrycsk et al., 2021] "Measuring Mathematical Problem Solving With the MATH Dataset." *NeurIPS 2021*
Link: https://arxiv.org/pdf/2103.03874.pdf

[Cobbe et al., 2021] "Training Verifiers to Solve Math Word Problems." *arXiv:2110.14168*
Link: https://arxiv.org/abs/2110.14168