

# CENG7880

# Trustworthy and Responsible AI

Instructor: Sinan Kalkan

(<https://ceng.metu.edu.tr/~skalkan>)

For course logistics and materials:

<https://metu-trai.github.io>

# Today

- Recurrent Neural Networks
- Self-Attention
- Transformers
- Pretraining Transformers
- Large-Language Models

# Administrative Notes

- Registrations and access to ODTUclass
- Selecting papers for the projects
  - Deadline: 16 October
  - Form: <https://forms.gle/A3taWgxoCHumfYfz9>

# Sequence Labeling/Modeling: Motivation

# Why do we need them?

*Foreign minister.* → FOREIGN MINISTER,



# Different types of sequence learning / recognition problems

- Sequence Classification
  - A sequence to a label
  - E.g., recognizing a single spoken word
  - Length of the sequence is fixed
  - Why RNNs then? Because sequential modeling provides robustness against translations and distortions.
- Segment Classification
  - Segments in a sequence correspond to labels
- Temporal Classification
  - General case: sequence (input) to sequence (label) modeling.
  - No clue about where input or label starts.

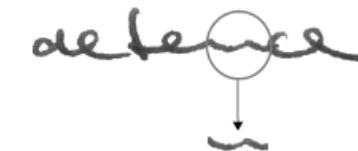
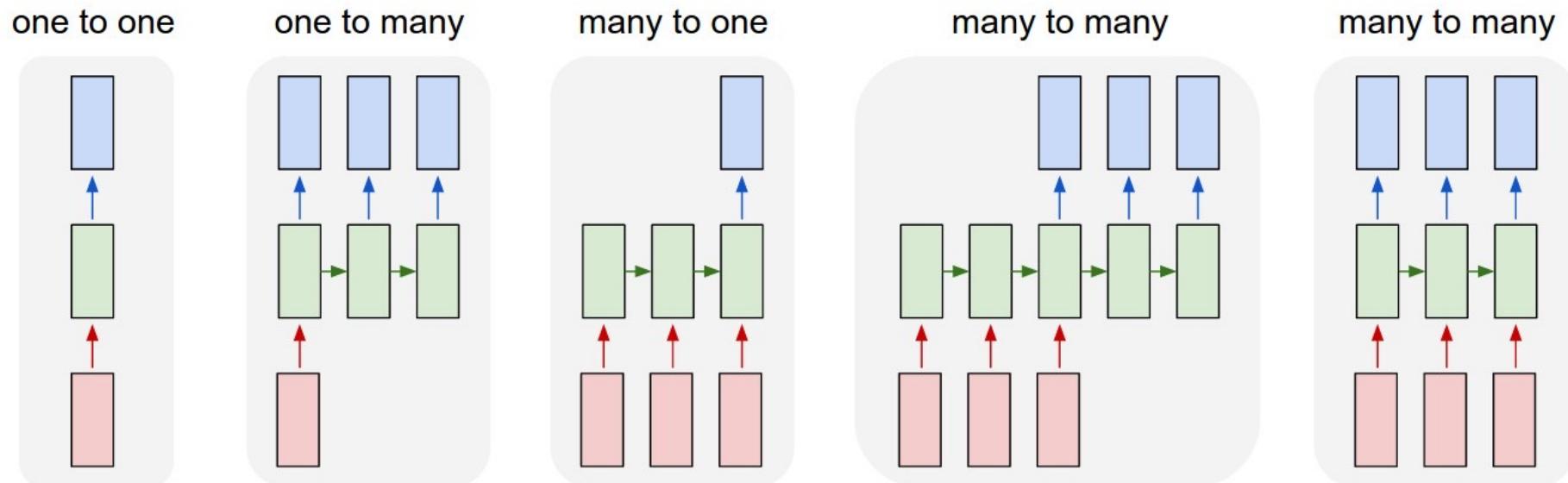


Fig. 2.3 Importance of context in segment classification. The word ‘defence’ is clearly legible. However the letter ‘n’ in isolation is ambiguous.

A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks”, 2012.

# Different types of sequence learning / recognition problems



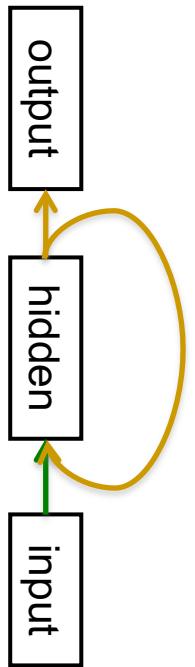
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Recurrent Neural Networks

# Recurrent Neural Networks (RNNs)



Feed-forward  
networks

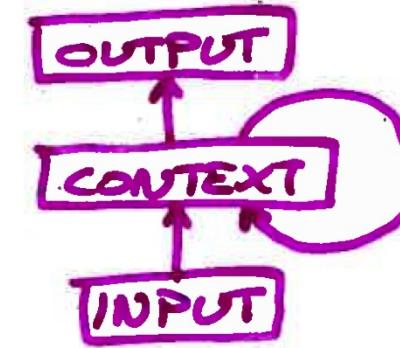
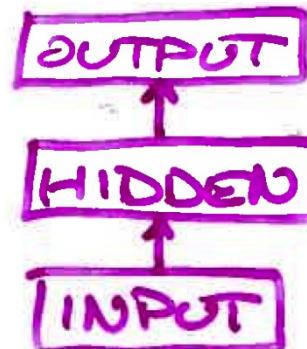


Recurrent  
networks

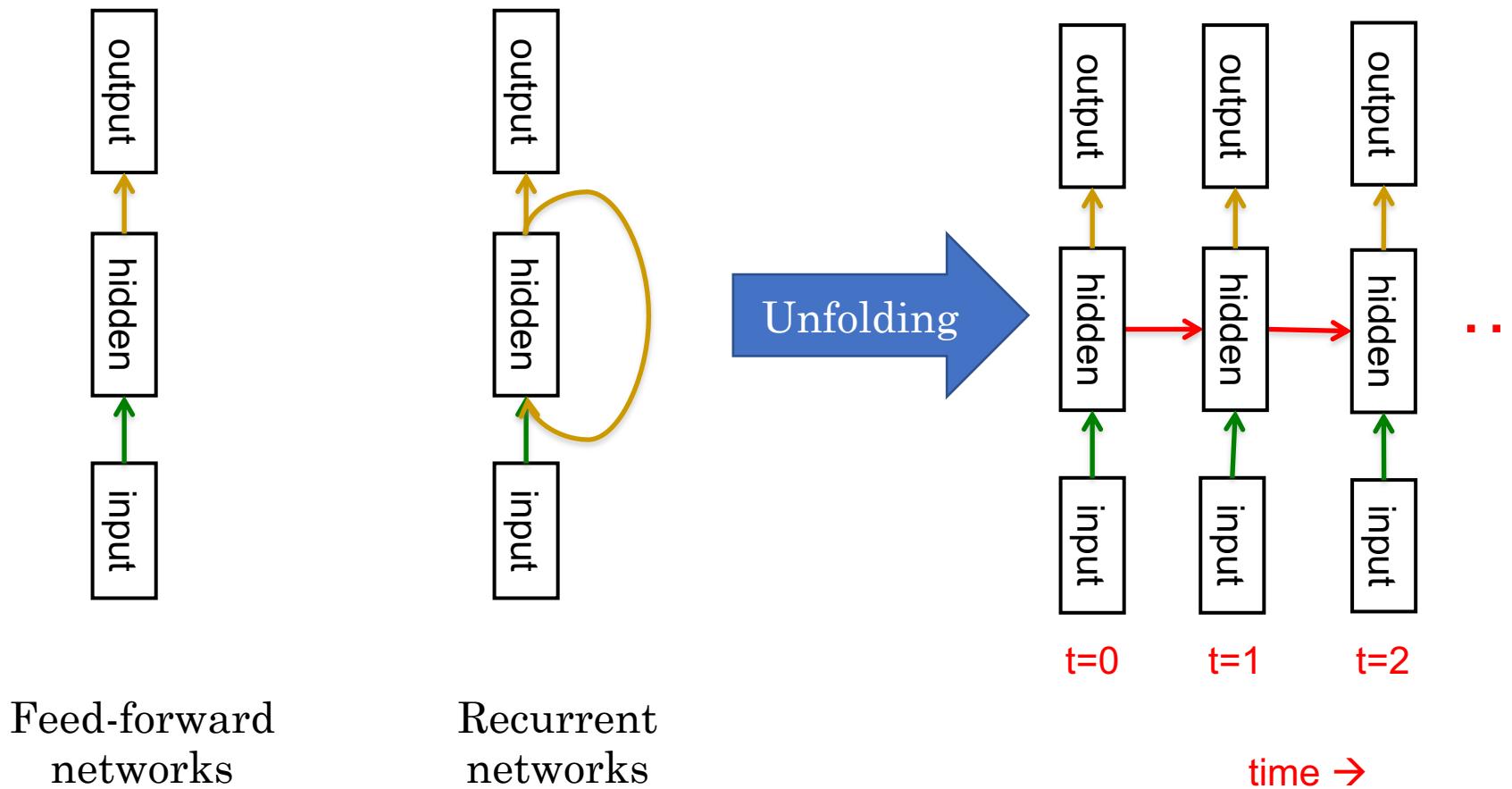
- RNNs are very powerful because:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.
- More formally, **RNNs are Turing complete.**

# Challenge

- Back propagation is designed for feedforward nets
- What would it mean to back propagate through a recurrent network?
  - error signal would have to travel back in time



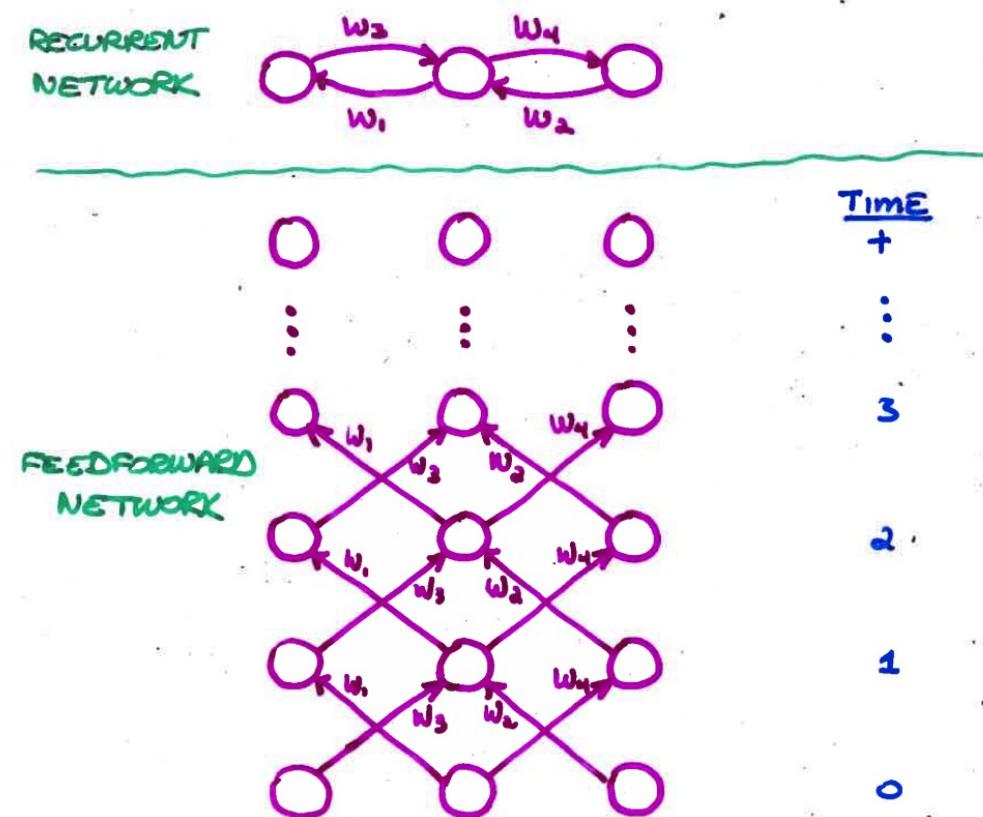
# Unfolding



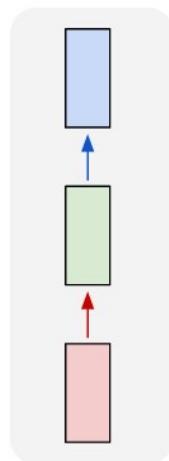
Feed-forward  
networks

Recurrent  
networks

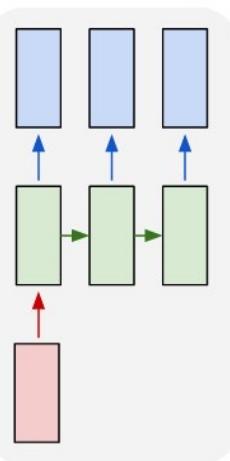
# Unfolding (another example)



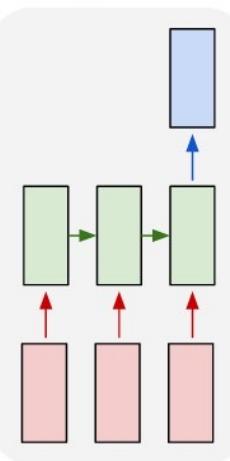
one to one



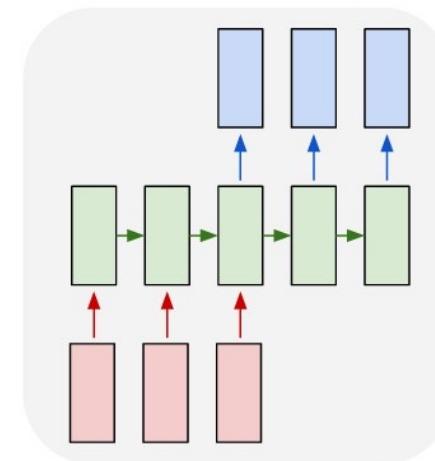
one to many



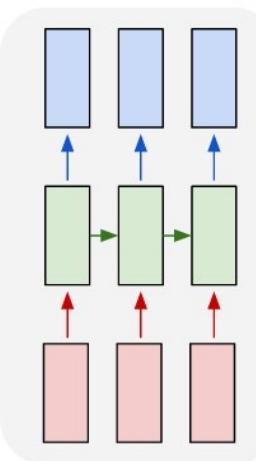
many to one



many to many

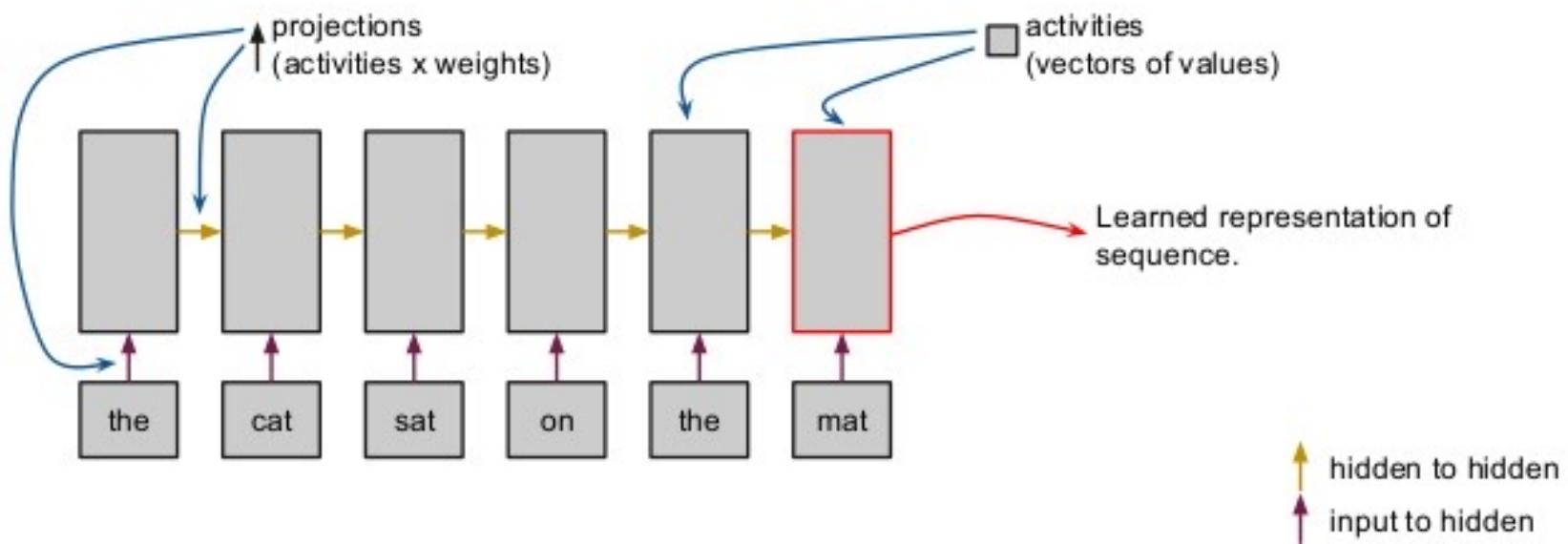


many to many

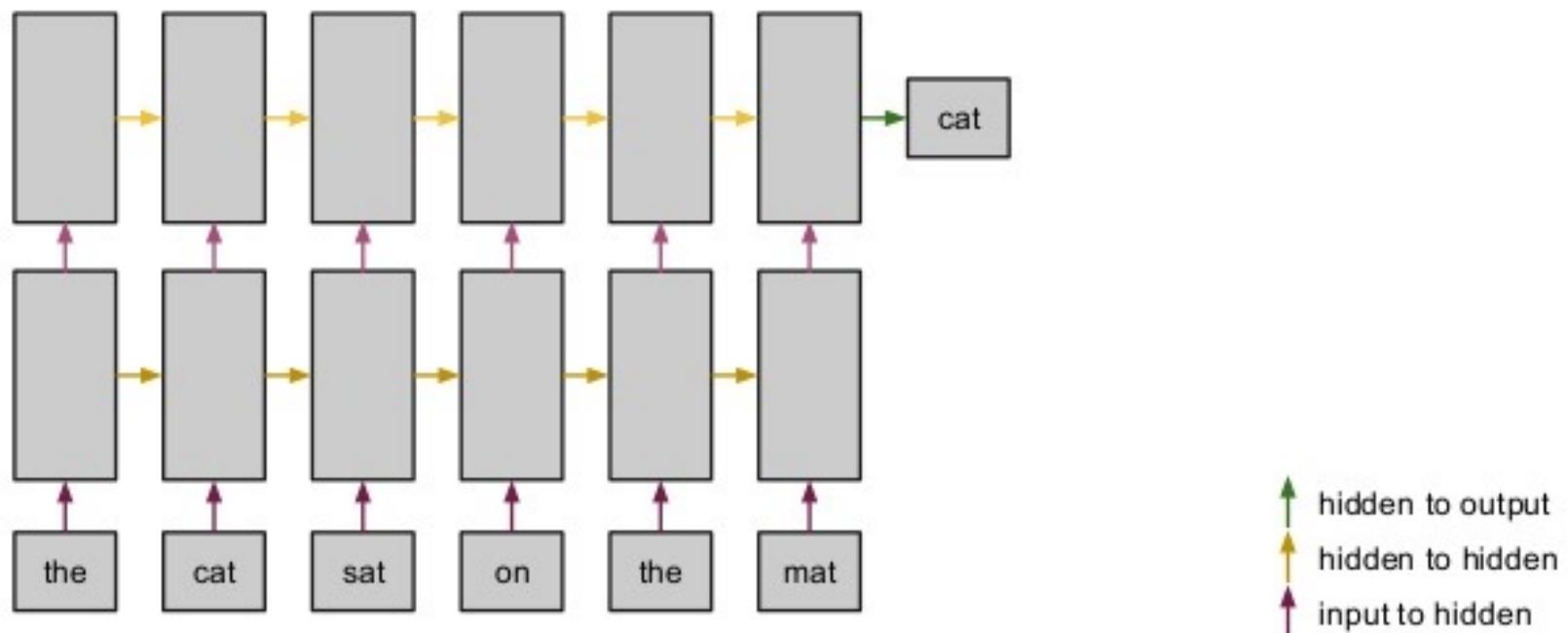


<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# How an RNN works



# You can stack them too

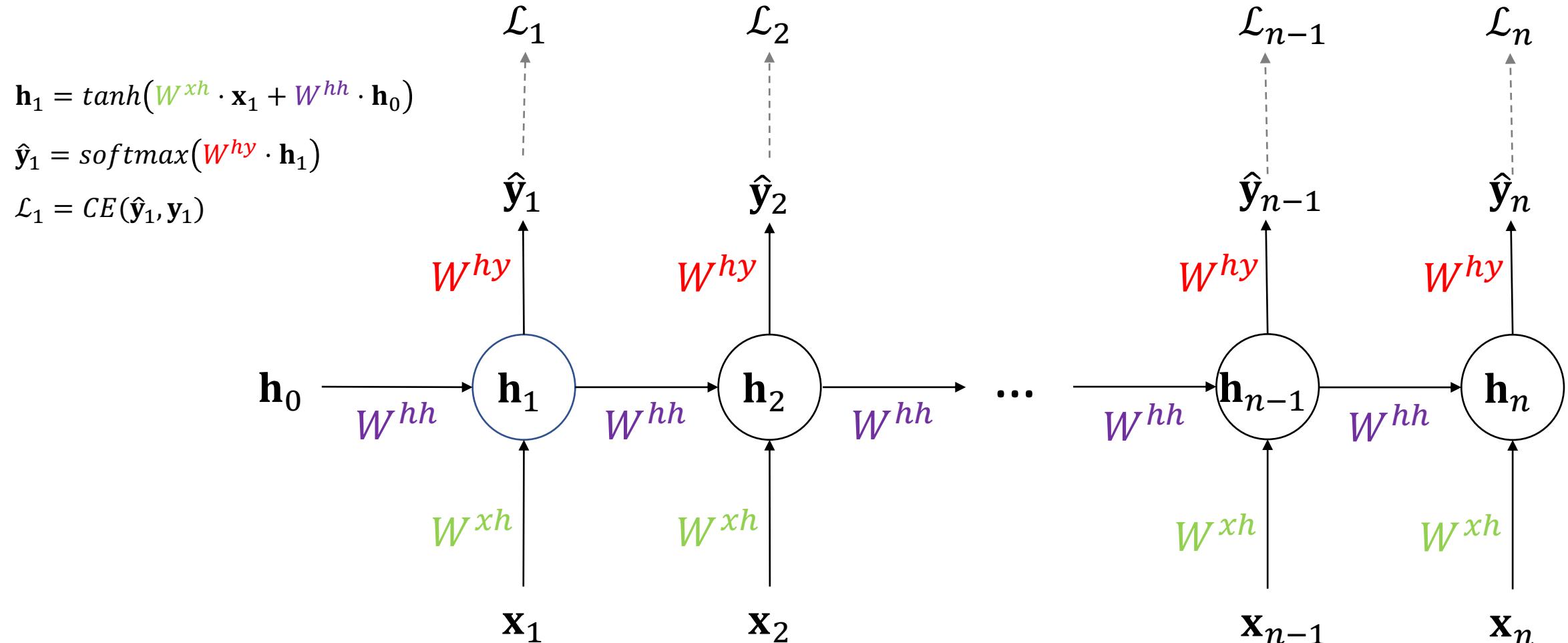


# Unfolding implications

- Entails duplication of weights => weight sharing
- Sharing weights means their gradients will be accumulated over time and reflected on the weights
- Unfolded network has the **same dynamics of the RNN** for a **fixed number of time steps!**

# Back-propagation Through Time

# Feedforward through Vanilla RNN



# Feedforward through Vanilla RNN

## The Vanilla RNN Model

First time-step ( $t = 1$ ):

$$\mathbf{h}_1 = \tanh(W^{xh} \cdot \mathbf{x}_1 + W^{hh} \cdot \mathbf{h}_0)$$

$$\hat{\mathbf{y}}_1 = \text{softmax}(W^{hy} \cdot \mathbf{h}_1)$$

$$\mathcal{L}_1 = CE(\hat{\mathbf{y}}_1, \mathbf{y}_1)$$

In general:

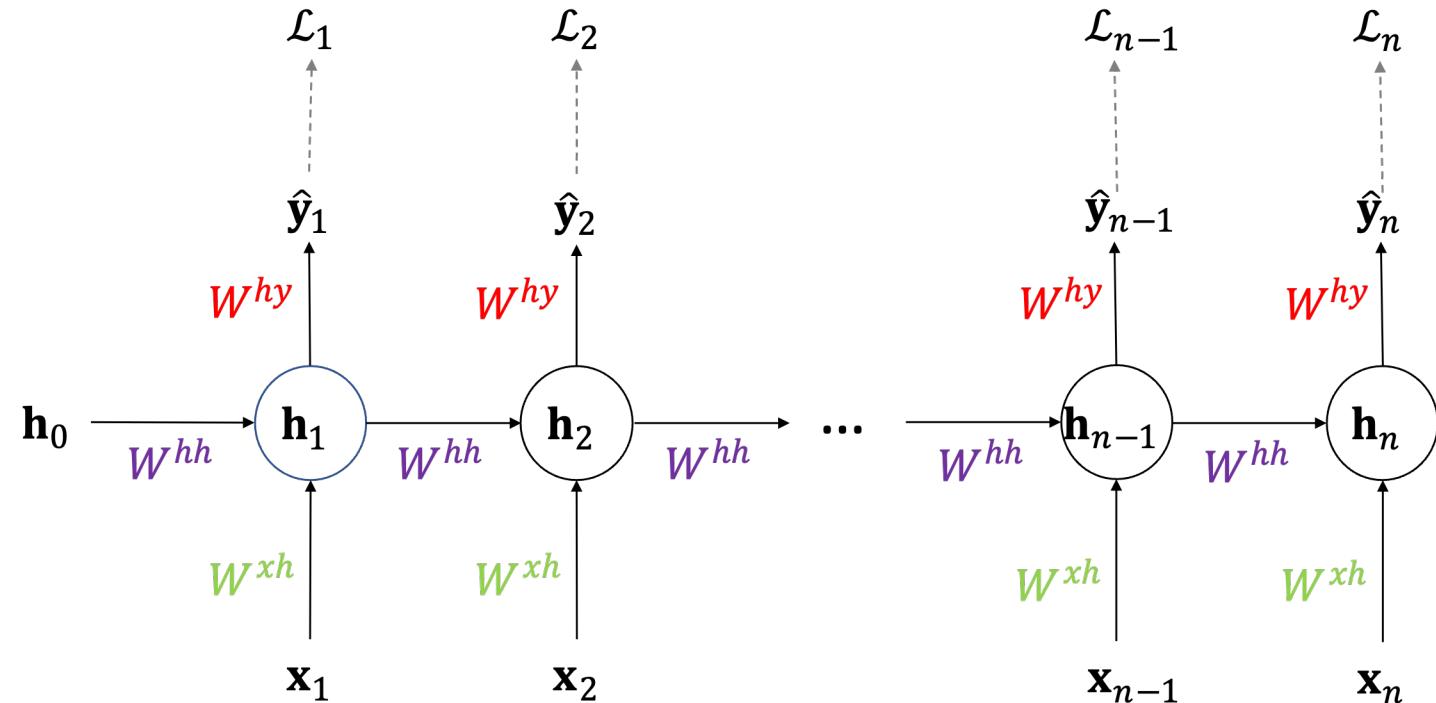
$$\mathbf{h}_t = \tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

$$\hat{\mathbf{y}}_t = \text{softmax}(W^{hy} \cdot \mathbf{h}_t)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



# Backpropagation through Vanilla RNN

$$\frac{\partial \mathcal{L}}{\partial W^{hy}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \hat{y}_n} \frac{\partial \hat{y}_n}{\partial W^{hy}} + \frac{\partial \mathcal{L}}{\partial \hat{y}_{n-1}} \frac{\partial \hat{y}_{n-1}}{\partial W^{hy}} + \dots + \frac{\partial \mathcal{L}}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial W^{hy}}$$

$$= \sum_{t=1..n} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W^{hy}}$$

The Vanilla RNN Model

In general:

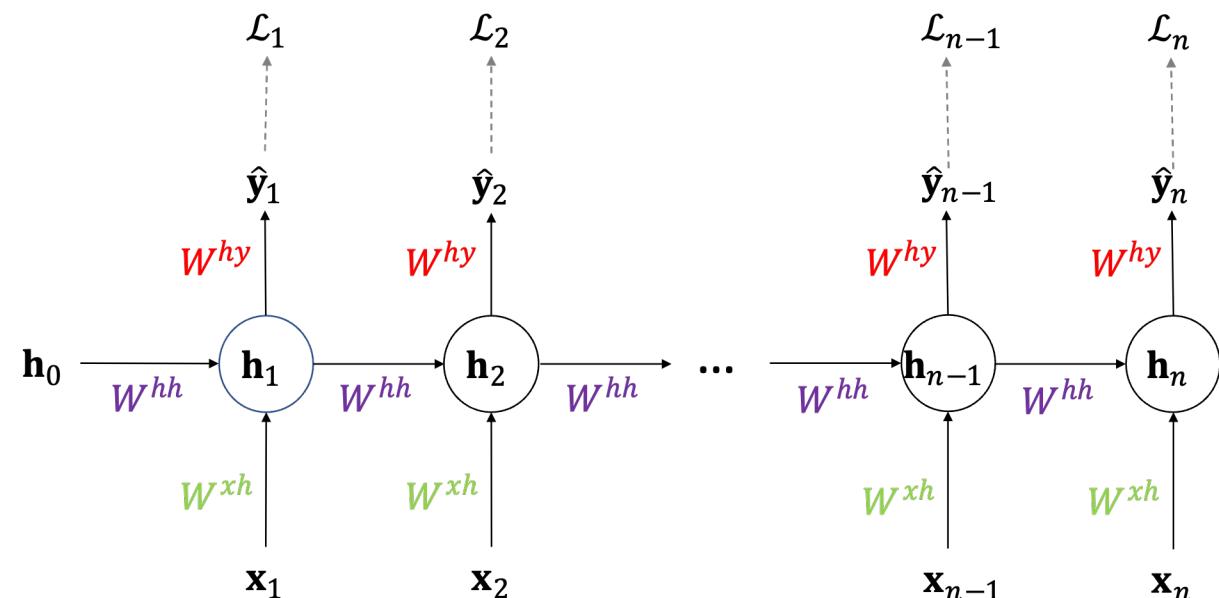
$$\mathbf{h}_t = \tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

$$\hat{\mathbf{y}}_t = \text{softmax}(W^{hy} \cdot \mathbf{h}_t)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



# Backpropagation through Vanilla RNN

$$\frac{\partial \mathcal{L}}{\partial W^{hh}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_n} \frac{\partial \mathbf{h}_n}{\partial W^{hh}} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{n-1}} \frac{\partial \mathbf{h}_{n-1}}{\partial W^{hh}} + \cdots + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial W^{hh}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

The Vanilla RNN Model

In general:

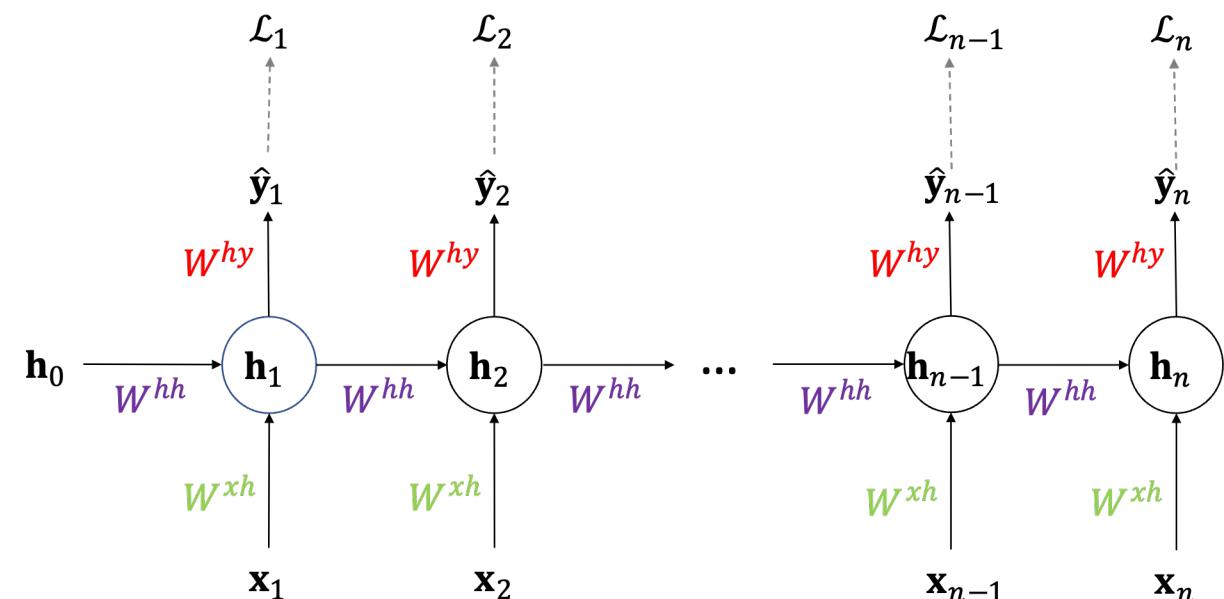
$$\mathbf{h}_t = \tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

$$\hat{\mathbf{y}}_t = \text{softmax}(W^{hy} \cdot \mathbf{h}_t)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



# Backpropagation through Vanilla RNN

$$\frac{\partial \mathcal{L}}{\partial W^{xh}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_n} \frac{\partial \mathbf{h}_n}{\partial W^{xh}} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{n-1}} \frac{\partial \mathbf{h}_{n-1}}{\partial W^{xh}} + \cdots + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial W^{xh}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

(calculated before)

## The Vanilla RNN Model

In general:

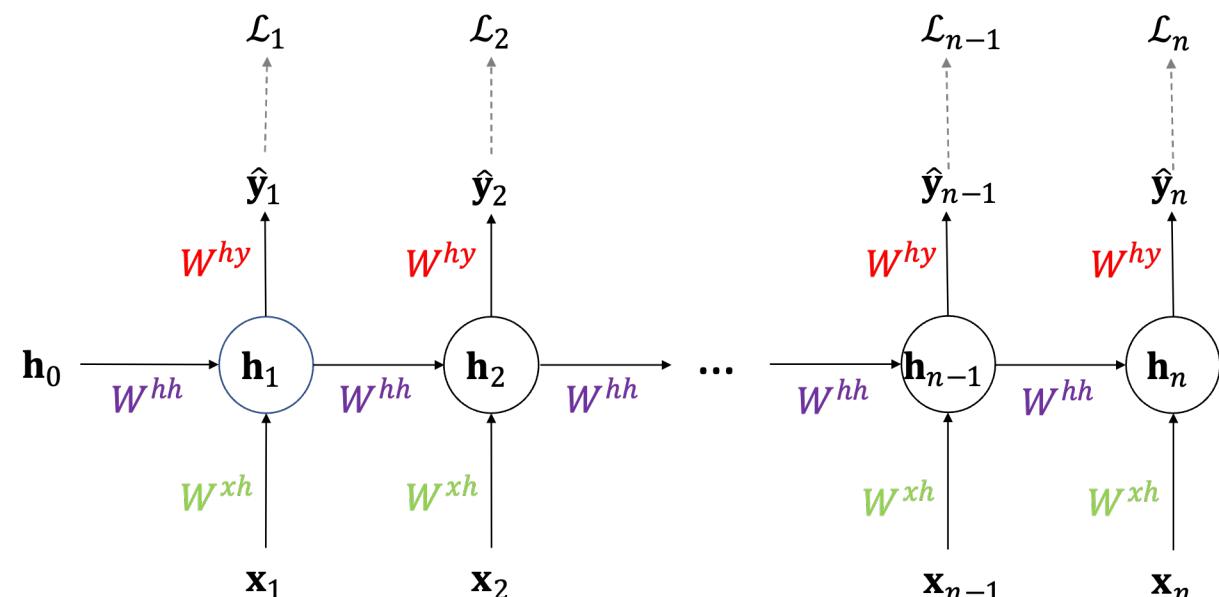
$$\mathbf{h}_t = \tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

$$\hat{\mathbf{y}}_t = \text{softmax}(W^{hy} \cdot \mathbf{h}_t)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



# Initial hidden state

- We need to specify the **initial activity state of all the hidden units**.
- We could just fix these initial states to have some default value like 0.5.
- But it is better to **treat the initial states as learned parameters**.
- We learn them in the same way as we learn the weights.
  - Start off with an initial random guess for the initial states.
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
  - Adjust the initial states by following the negative gradient.

# The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, it's very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

# Exploding and vanishing gradients problem

- Solution 1: Gradient clipping for exploding gradients:

---

**Algorithm 1** Pseudo-code for norm clipping

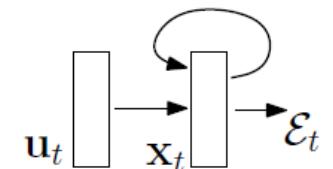
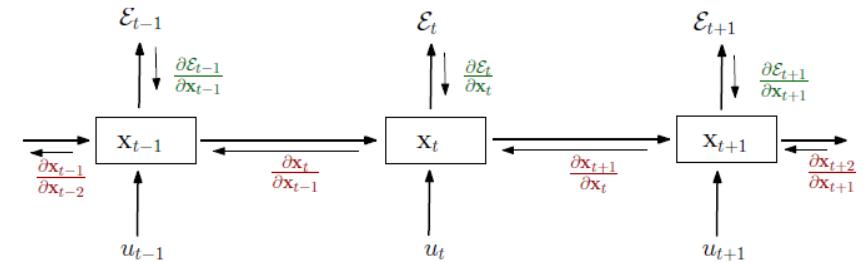
---

```
ĝ ← ∂E / ∂θ
if ||ĝ|| ≥ threshold then
    ĝ ← threshold / ||ĝ|| ĝ
end if
```

---

- For vanishing gradients: Regularization term that penalizes changes in the magnitudes of back-propagated gradients

$$\Omega = \sum_k \Omega_k = \sum_k \left( \frac{\left\| \frac{\partial E}{\partial x_{k+1}} \frac{\partial x_k}{\partial x_{k+1}} \right\|^2}{\left\| \frac{\partial E}{\partial x_{k+1}} \right\|^2} - 1 \right)^2$$



---

On the difficulty of training recurrent neural networks

---

Razvan Pascanu

PASCANUR@IRO.UMONTREAL.CA  
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Tomas Mikolov

T.MIKOLOV@GMAIL.COM  
Speech@FIT, Brno University of Technology, Brno, Czech Republic

Yoshua Bengio

YOSHUA.BENGIO@UMONTREAL.CA  
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

2012

# Exploding and vanishing gradients problem

- Solution 2:
  - Use methods like LSTM

# Long Short Term Memory (LSTM)

## LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter  
Fakultät für Informatik  
Technische Universität München  
80290 München, Germany  
[hochreit@informatik.tu-muenchen.de](mailto:hochreit@informatik.tu-muenchen.de)  
<http://www7.informatik.tu-muenchen.de/~hochreit>

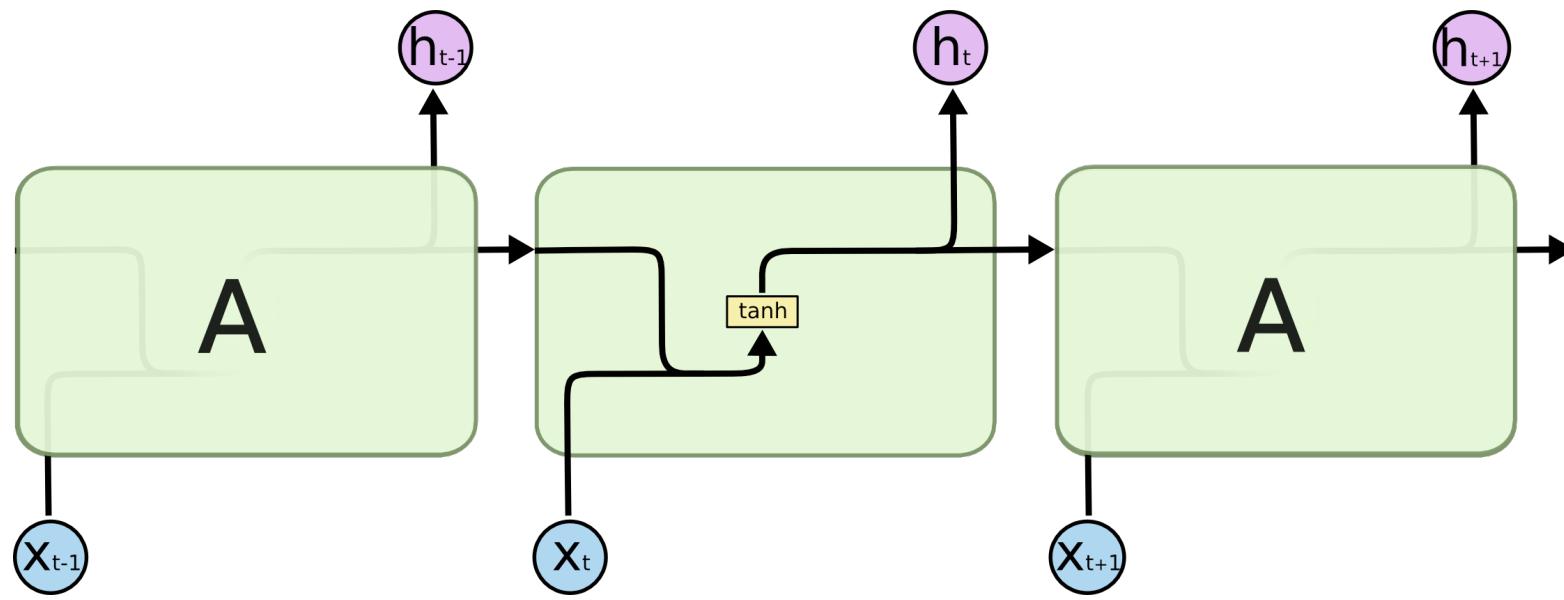
Jürgen Schmidhuber  
IDSIA  
Corso Elvezia 36  
6900 Lugano, Switzerland  
[juergen@idsia.ch](mailto:juergen@idsia.ch)  
<http://www.idsia.ch/~juergen>

### Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is  $O(1)$ . Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

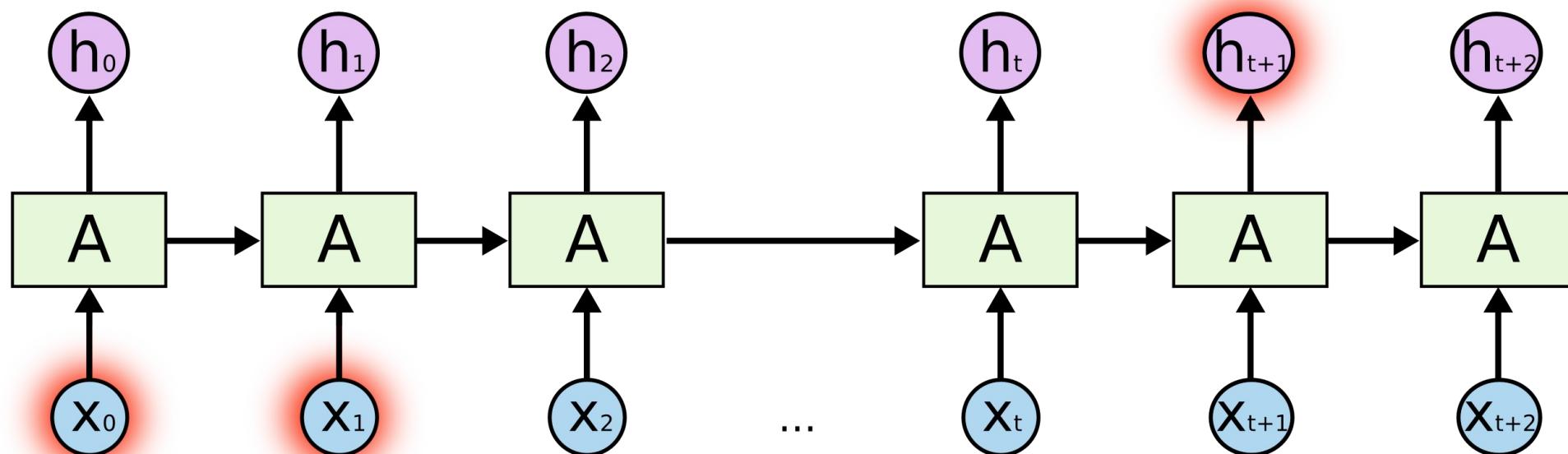
# RNN

- Basic block diagram



# Key Problem

- Learning long-term dependencies is hard

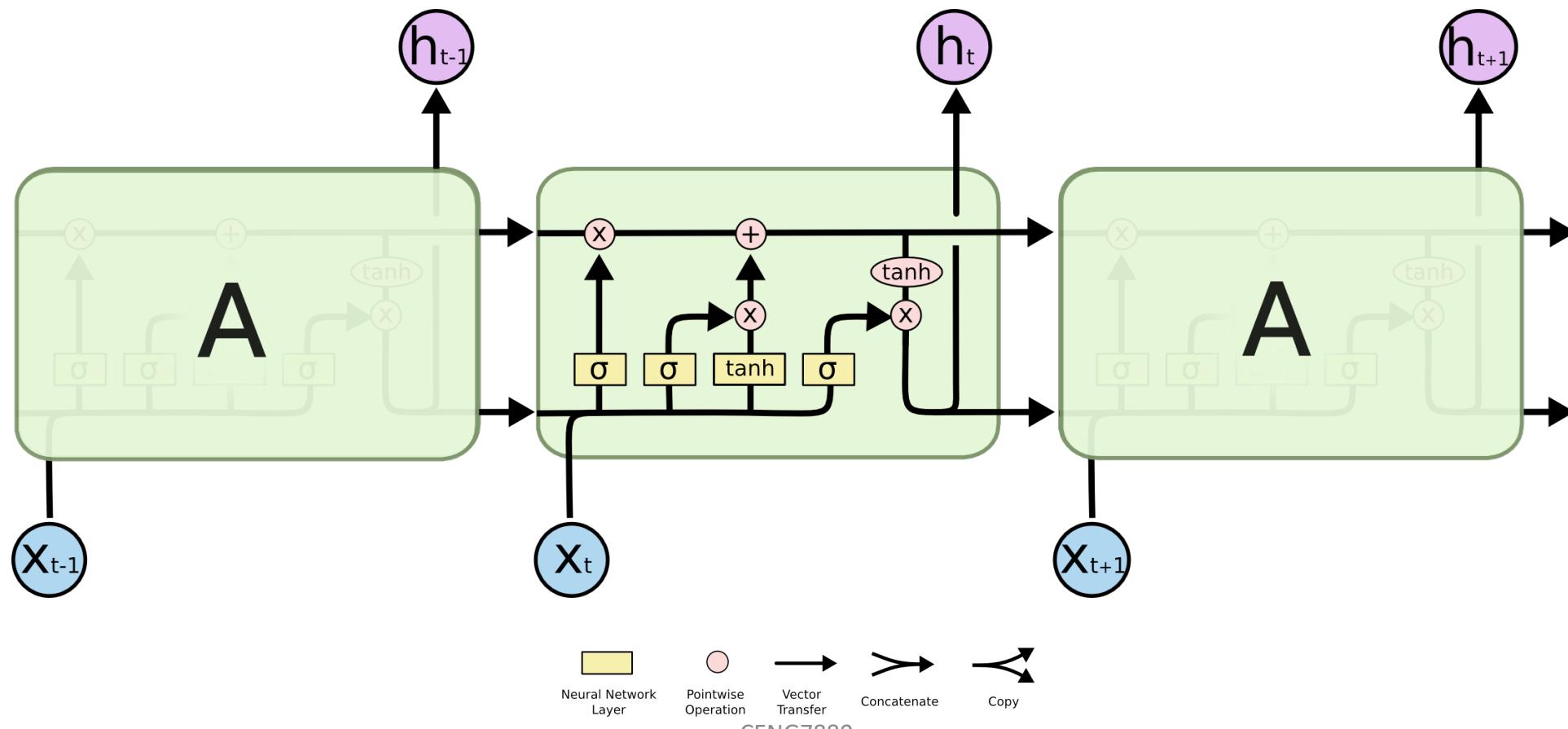


# Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.
- Information gets into the cell whenever its “write” gate is on.
- The information stays in the cell so long as its “keep” gate is on.
- Information can be read from the cell by turning on its “read” gate.

# Meet LSTMs

- How about we explicitly encode memory?



Neural Network  
Layer

Pointwise  
Operation

Vector  
Transfer

Concatenate

Copy

# LSTM in detail

- We first compute an activation vector,  $a$ :

$$a = W_x x_t + W_h h_{t-1} + b$$

- Split this into four vectors of the same size:

$$a_i, a_f, a_o, a_g \leftarrow a$$

- We then compute the values of the gates:

$$i = \sigma(a_i) \quad f = \sigma(a_f) \quad o = \sigma(a_o) \quad g = \tanh(a_g)$$

where  $\sigma$  is the sigmoid.

- The next cell state  $c_t$  and the hidden state  $h_t$ :

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

where  $\odot$  is the element-wise product of vectors

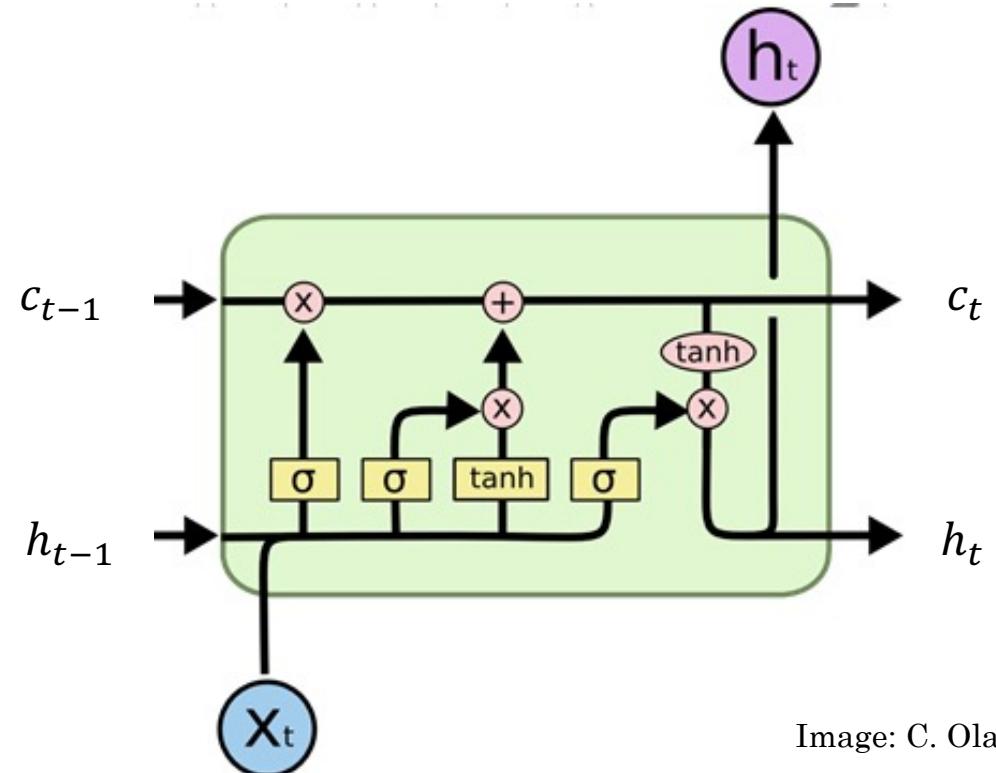


Image: C. Olah

Alternative formulation:

$$i_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = g(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

$$o_t = g(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

Eqs: Karpathy

# Example: Character-level Text Modeling

# Character-level Text Modeling

- Problem definition: Find  $c_{n+1}$  given  $c_1, c_2, \dots, c_n$ .

- Modelling:

$$p(c_{n+1} | c_n, \dots, c_1)$$

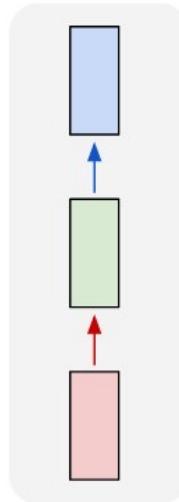
- In general, we just take the last  $N$  characters:

$$p(c_{n+1} | c_n, \dots, c_{n-(N-1)})$$

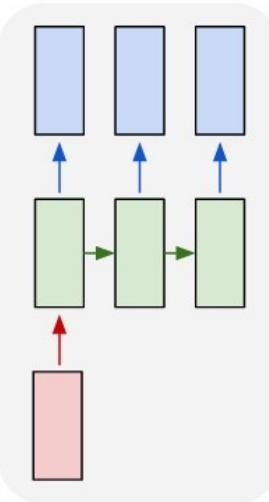
- Learn  $p(c_{n+1} = 'a' | 'Ankar')$  from data such that

$$p(c_{n+1} = 'a' | 'Ankar') > p(c_{n+1} = 'o' | 'Ankar')$$

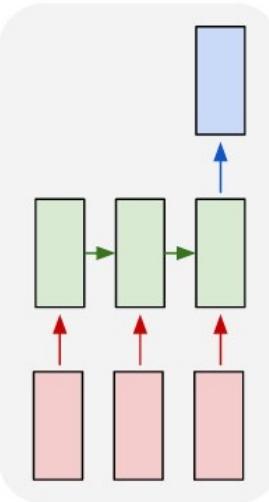
one to one



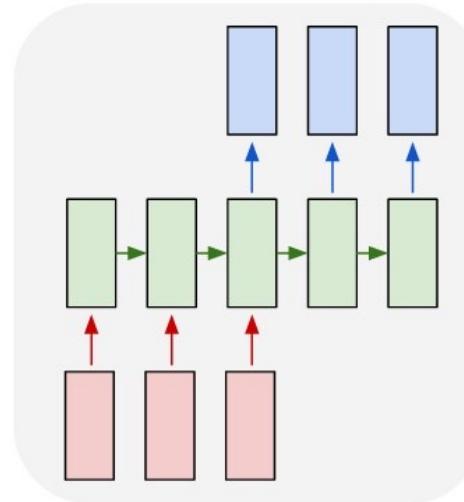
one to many



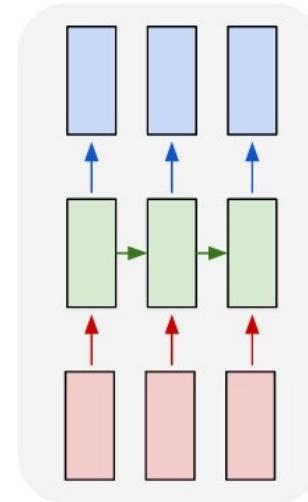
many to one



many to many



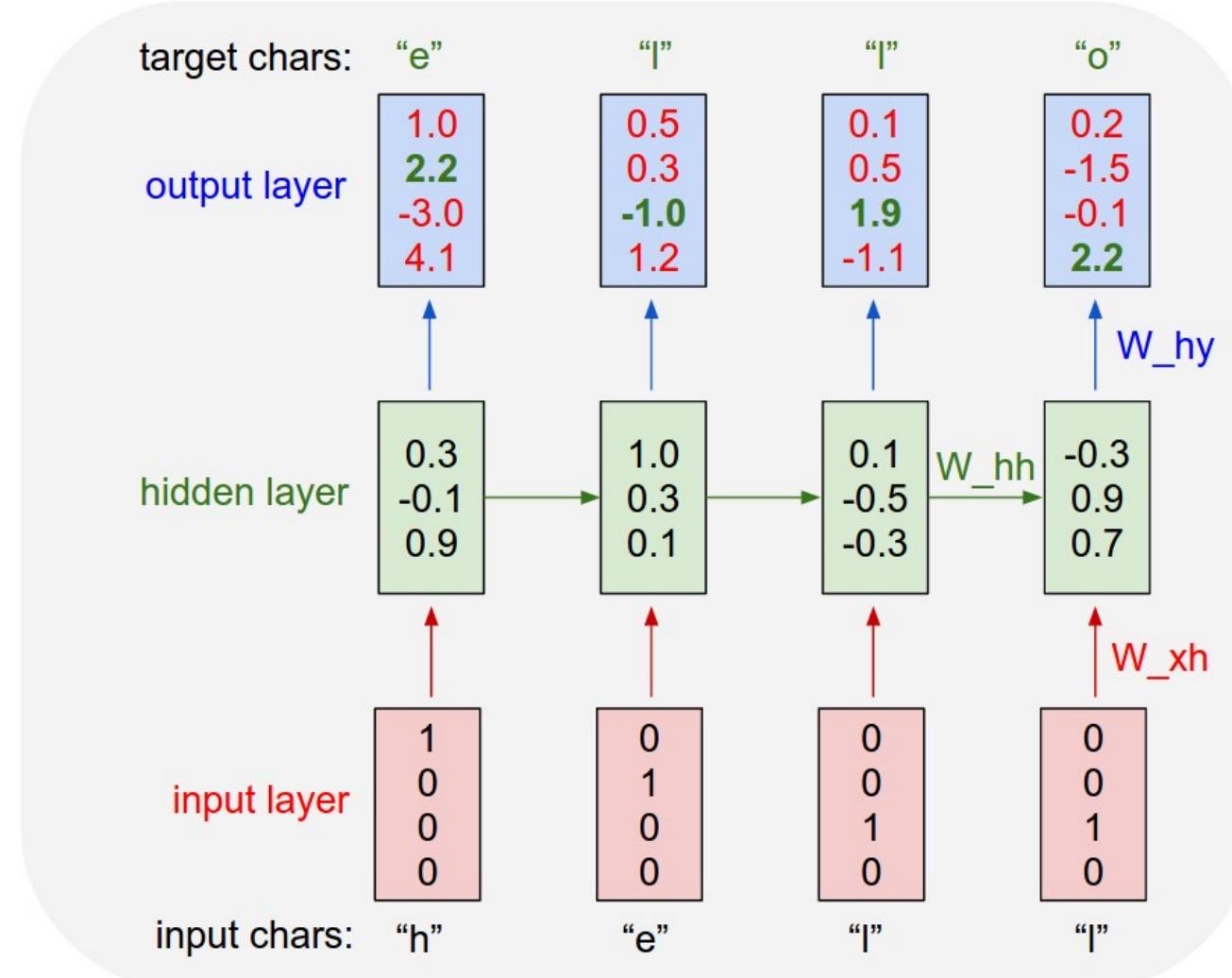
many to many



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# A simple scenario

- Alphabet: h, e, l, o
- Text to train to predict:  
“hello”



# What does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers.
- It is good at balancing quotes and brackets.
  - It can count brackets: **none, one, many**
- It knows a lot about syntax but its very hard to pin down exactly what form this knowledge has.
  - Its syntactic knowledge is not modular.
- It knows a lot of weak semantic associations
  - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable.

# Example: Word-level Text Modeling

# Word-level Text Modeling

- Problem definition: Find  $\omega_{n+1}$  given  $\omega_1, \omega_2, \dots, \omega_n$ .

- Modelling:

$$p(\omega_{n+1} | \omega_n, \dots, \omega_1)$$

- In general, we just take the last  $N$  words:

$$p(\omega_{n+1} | \omega_n, \dots, \omega_{n-(N-1)})$$

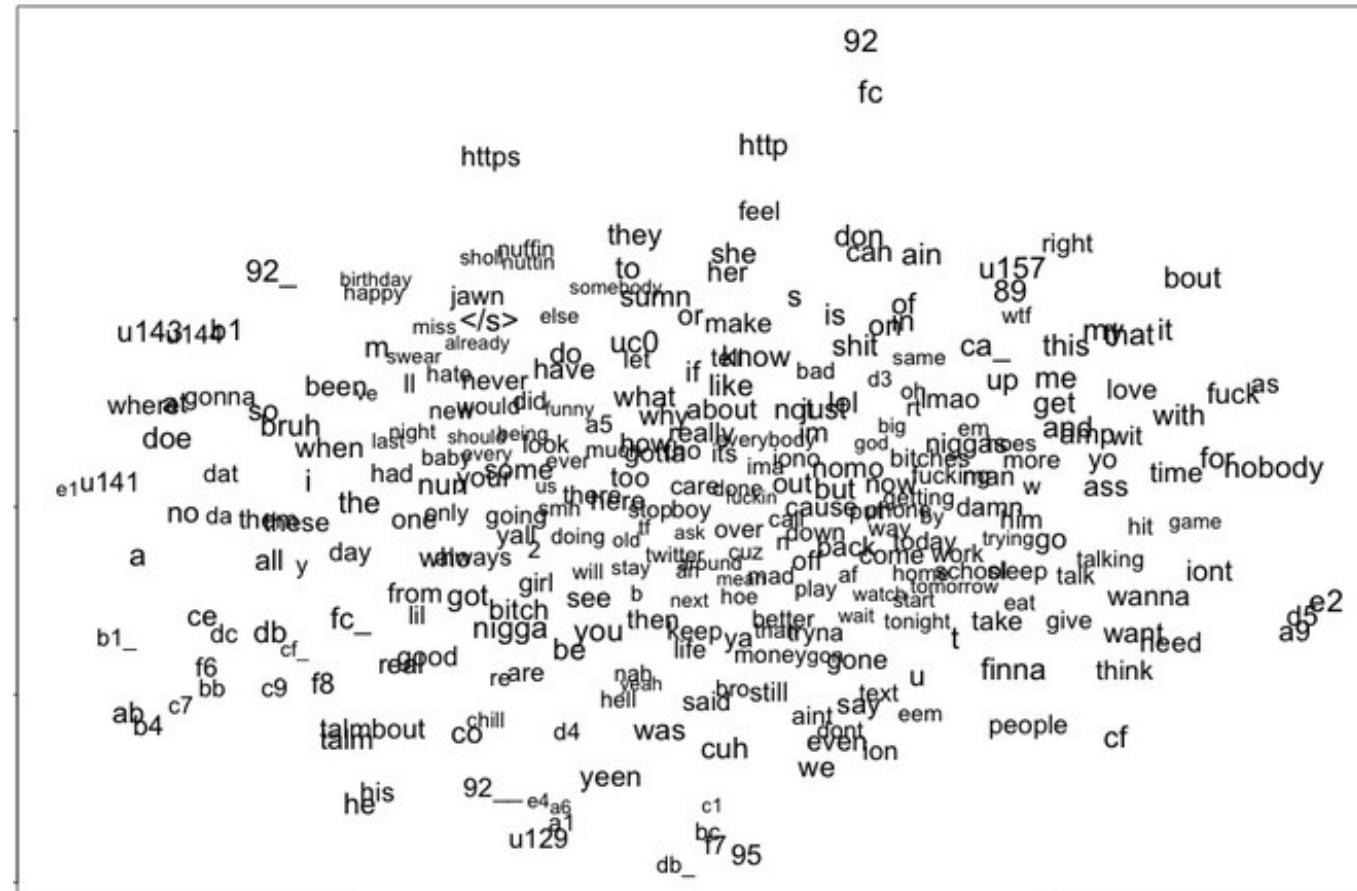
- Learn  $p(\omega_{n+1} = 'Turkey' | 'Ankara is the capital of ')$  from data such that:

$$p(\omega_{n+1} = 'Turkey' | 'Ankara is the capital of ') > p(\omega_{n+1} = 'UK' | 'Ankara is the capital of ')$$

# A handicap

- The number of characters is low enough to handle without doing anything extra.
  - English has 26 characters.
- The situation is very different for words.
  - English has  $\sim 170,000$  different words!
- This increases dimensionality and makes it difficult to capture “semantics”.
- Solution: Map words to a lower dimensional space, a.k.a. word embedding (word2vec).

### A two dimensional reduction of the vector space model using t-SNE



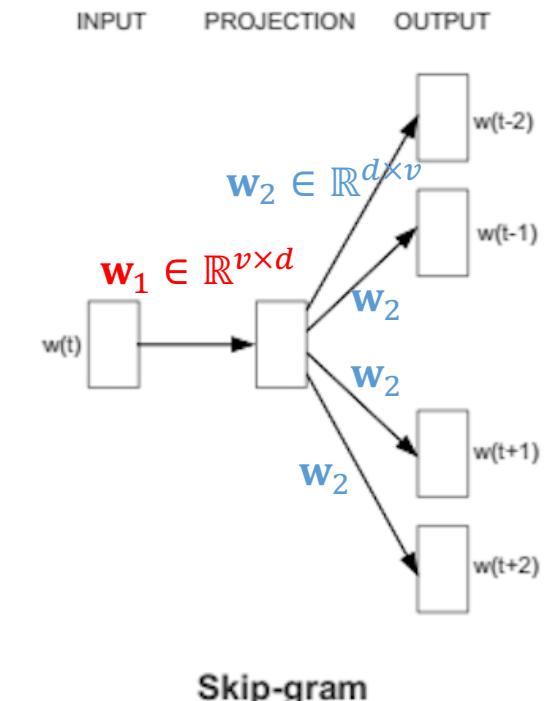
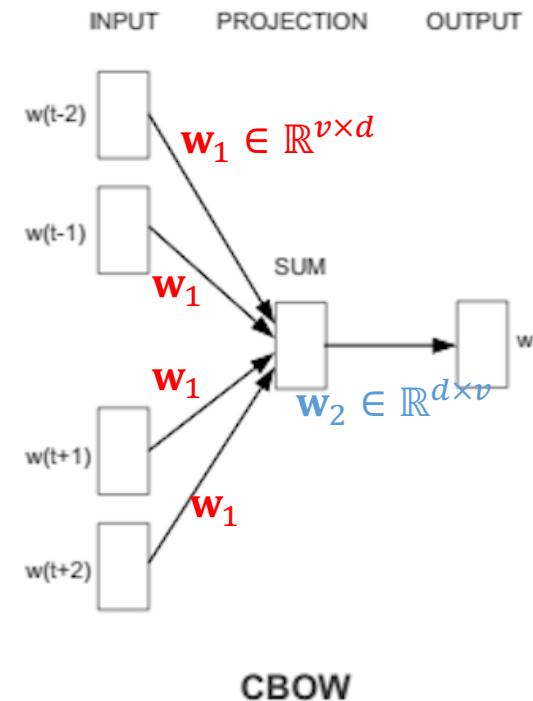
# Word Embedding (word2vec)

# Two different ways to train

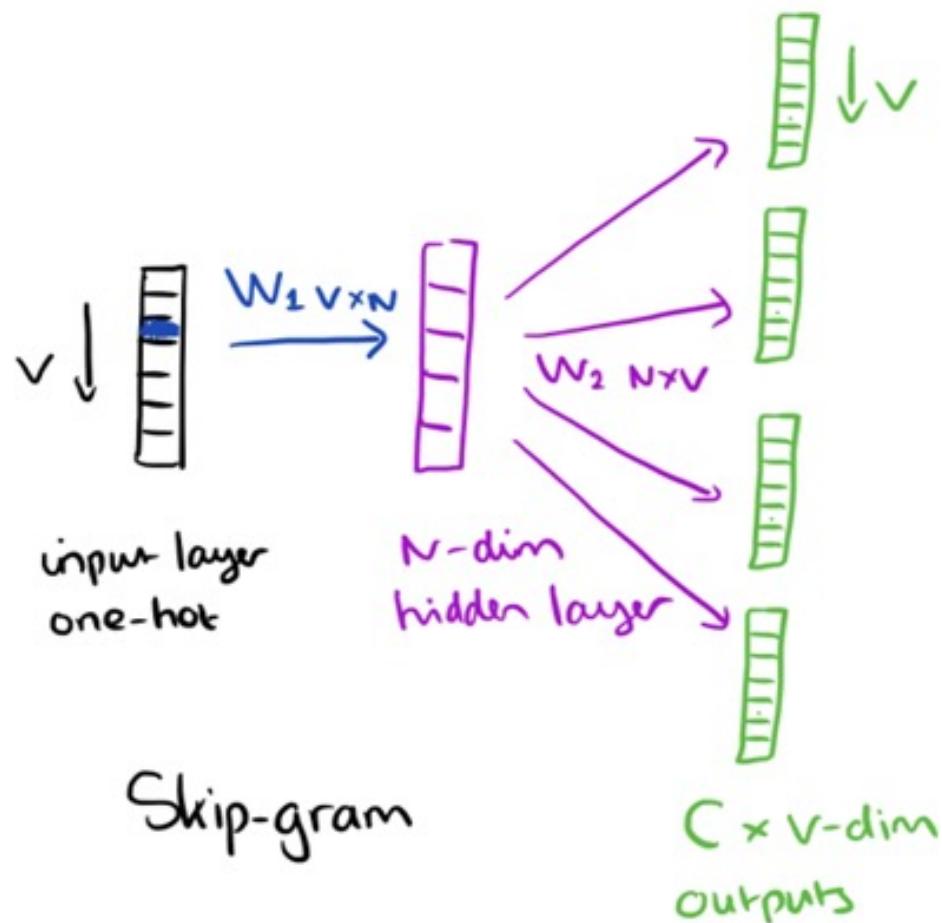
1. Using context to predict a target word (~ continuous bag-of-words)
2. Using word to predict a target context (skip-gram)

- If the vector for a word cannot predict the context, the mapping to the vector space is adjusted
- Since similar words should predict the same or similar contexts, their vector representations should end up being similar

$v$ : vocabulary size  
 $d$ : hidden dimension



# Note that the weight matrix is a look-up table



$$\begin{matrix} \text{input} \\ 1 \times v \end{matrix} \quad \begin{matrix} W_1 \\ v \times N \end{matrix} \quad \begin{matrix} \text{hidden} \\ 1 \times N \end{matrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} = \begin{bmatrix} e & f & g & h \end{bmatrix}$$

$w_i$



a man is playing tennis on a tennis court



a train is traveling down the tracks at a train station



a cake with a slice cut out of it

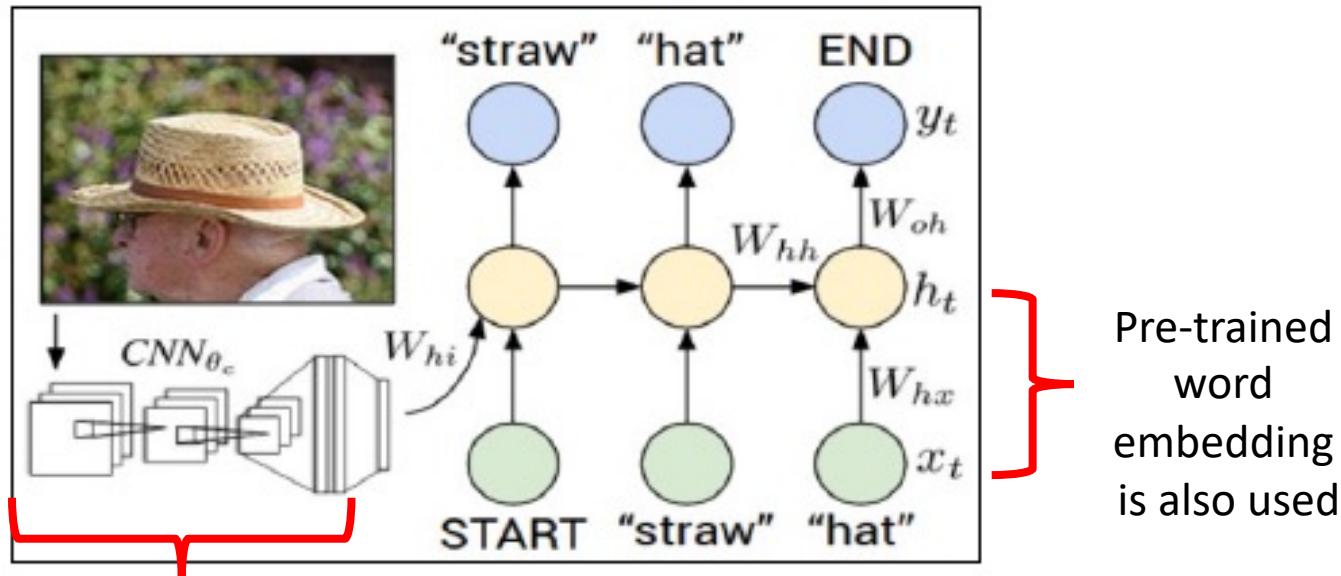


a bench sitting on a patch of grass next to a sidewalk

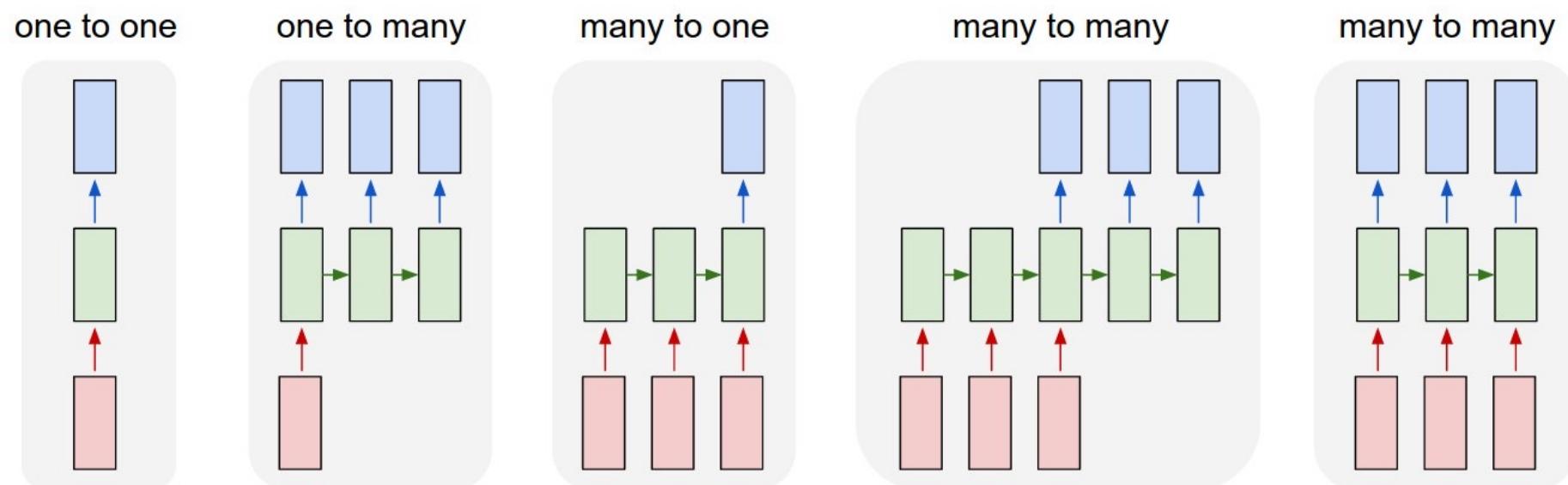
Fig: <https://github.com/karpathy/neuraltalk2>

# Example: Image Captioning

# Overview

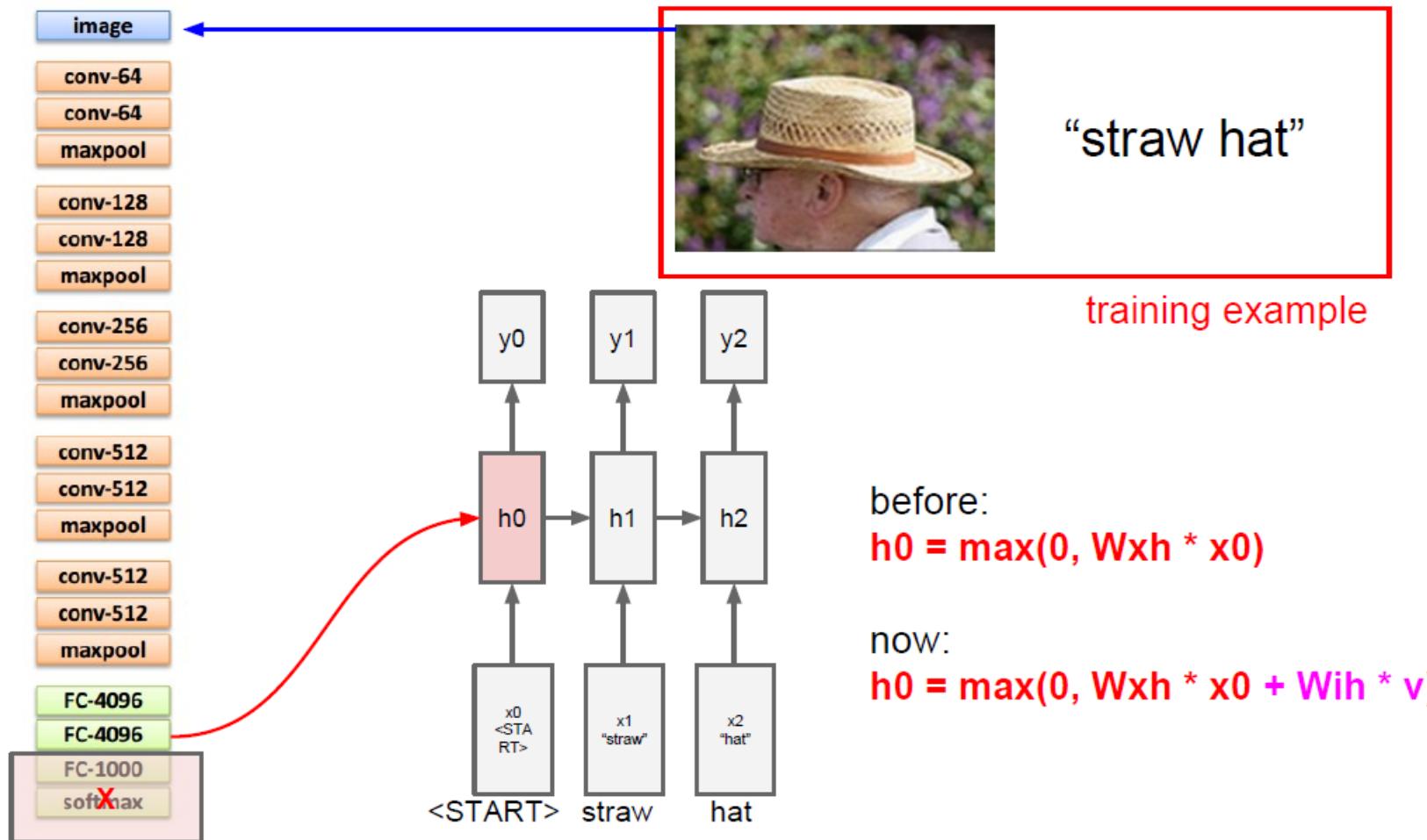


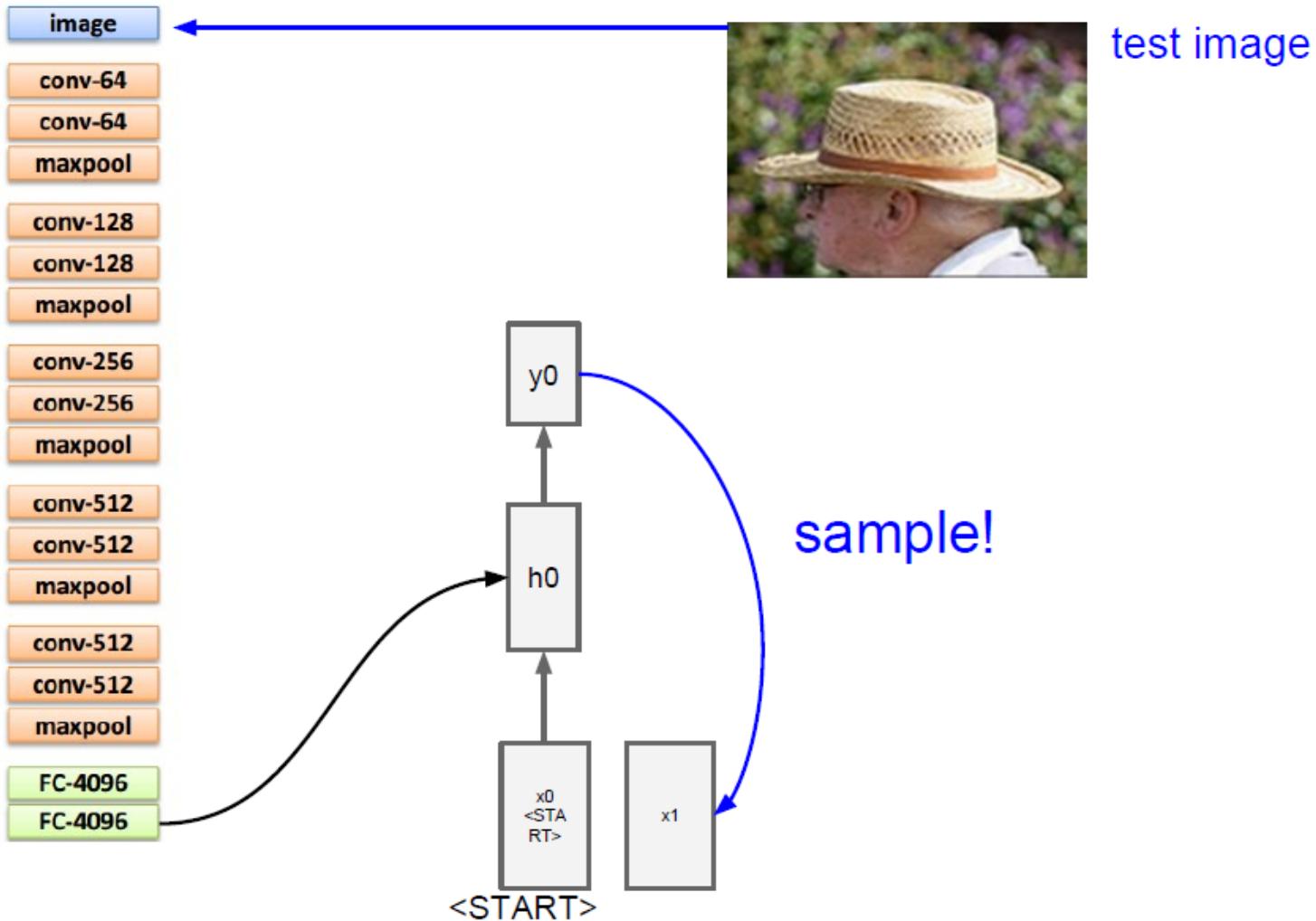
Pre-trained CNN  
(e.g., on imagenet)

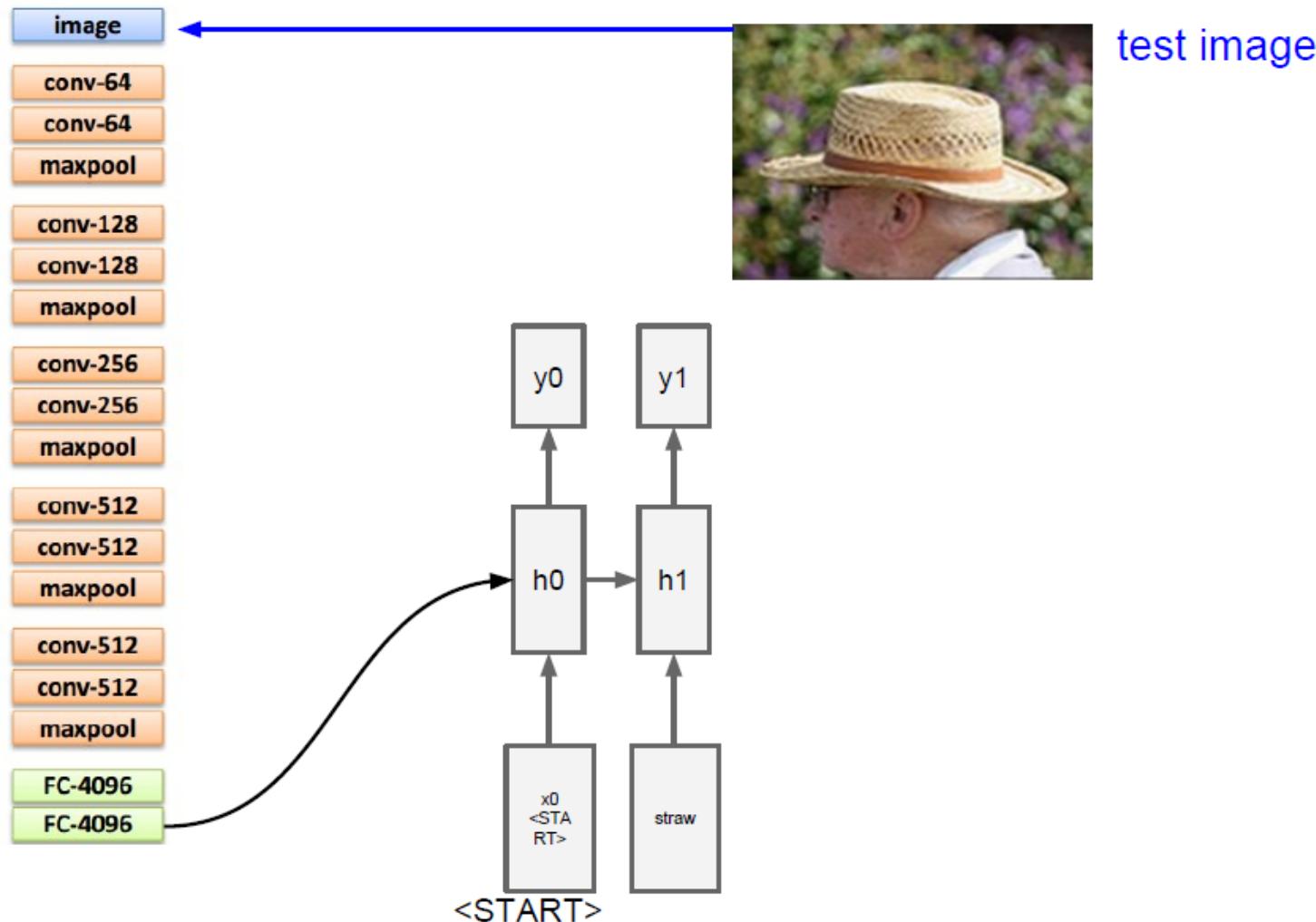


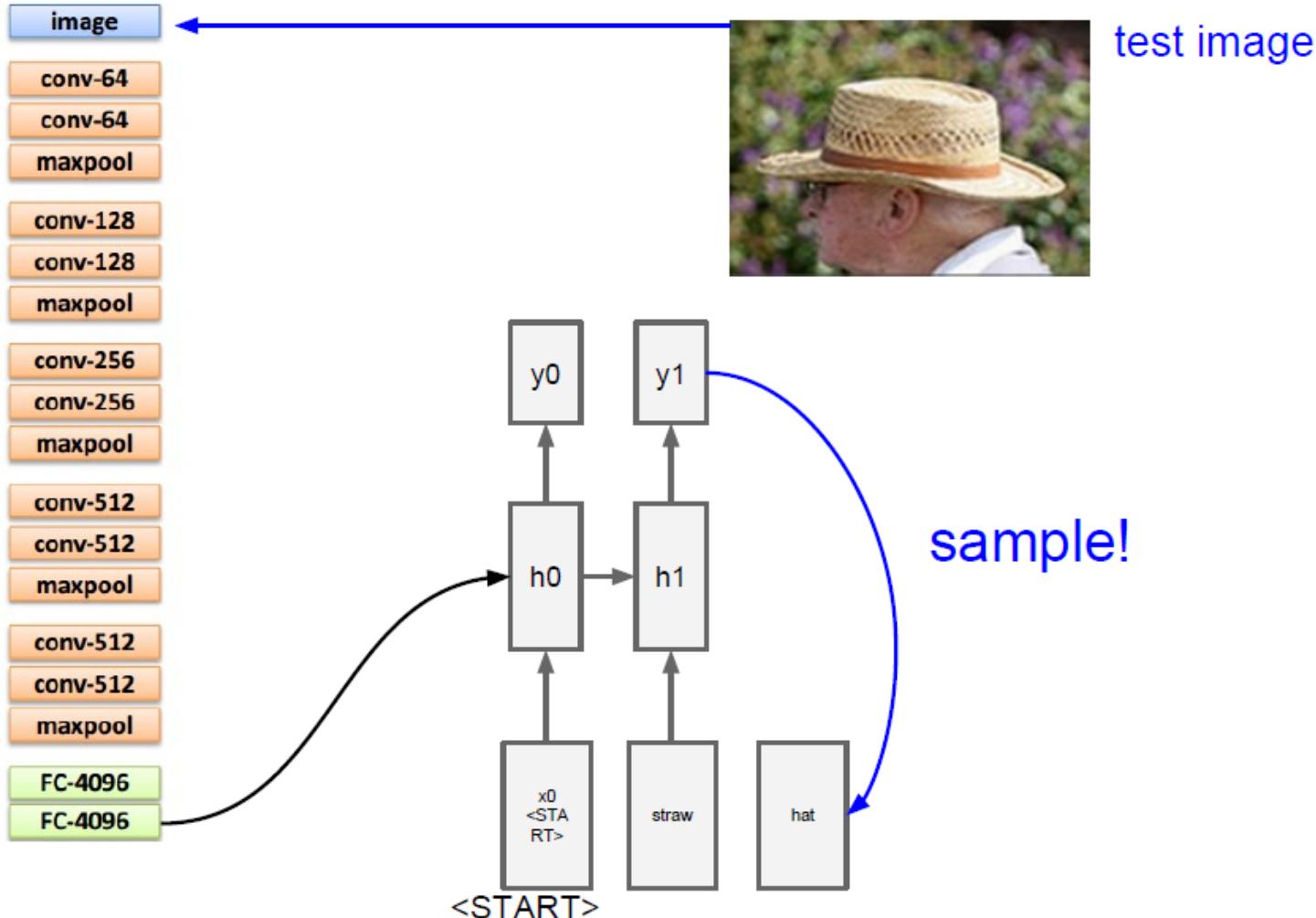
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

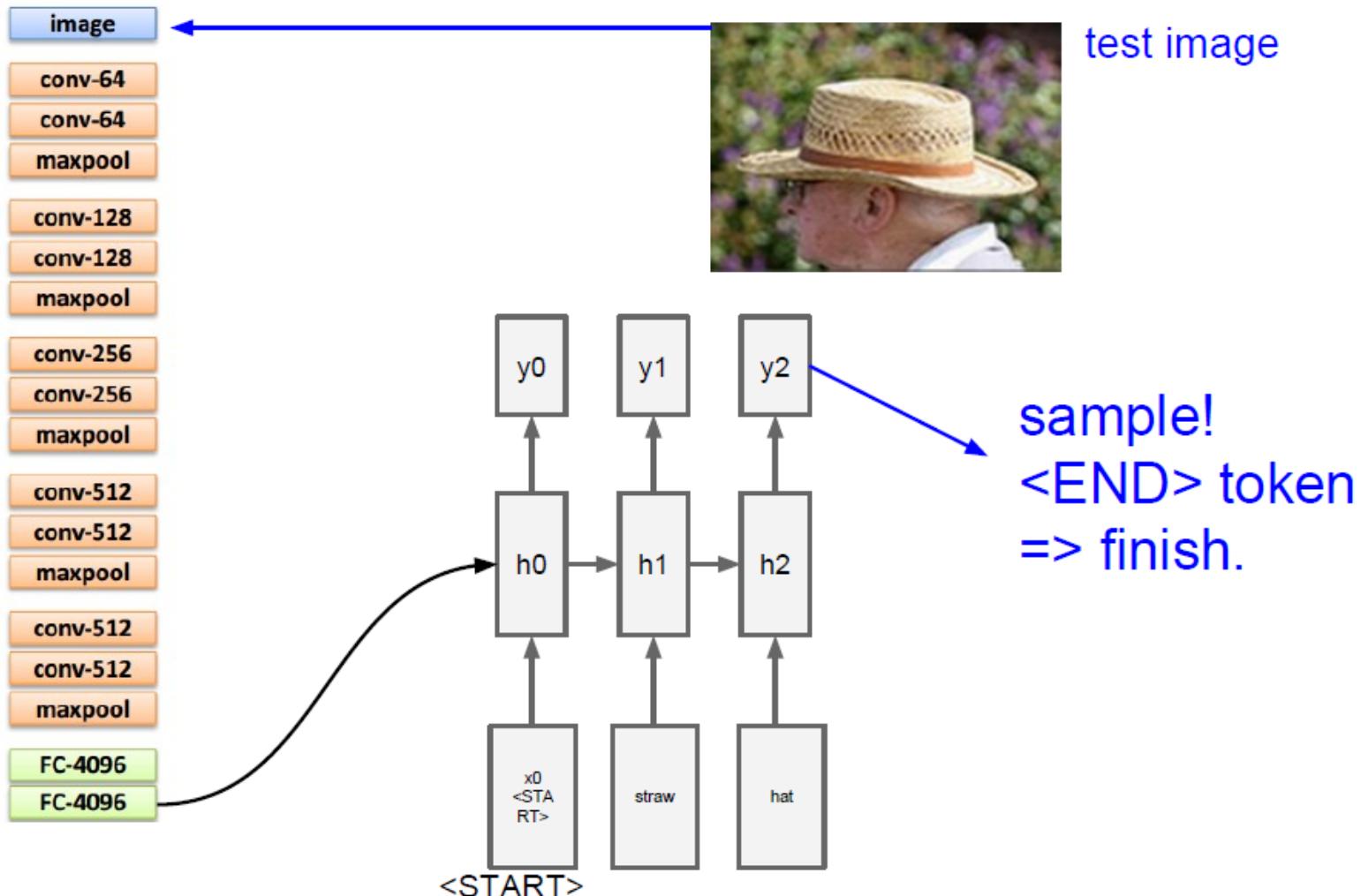
# Training











**Learning Phrase Representations using RNN Encoder–Decoder  
for Statistical Machine Translation**

**Kyunghyun Cho**

Bart van Merriënboer Caglar Gulcehre

Université de Montréal

firstname.lastname@umontreal.ca

Dzmitry Bahdanau

Jacobs University, Germany

d.bahdanau@jacobs-university.de

**Fethi Bougares Holger Schwenk**

Université du Maine, France

firstname.lastname@lium.univ-lemans.fr

**Yoshua Bengio**

Université de Montréal, CIFAR Senior Fellow

find.me@on.the.web

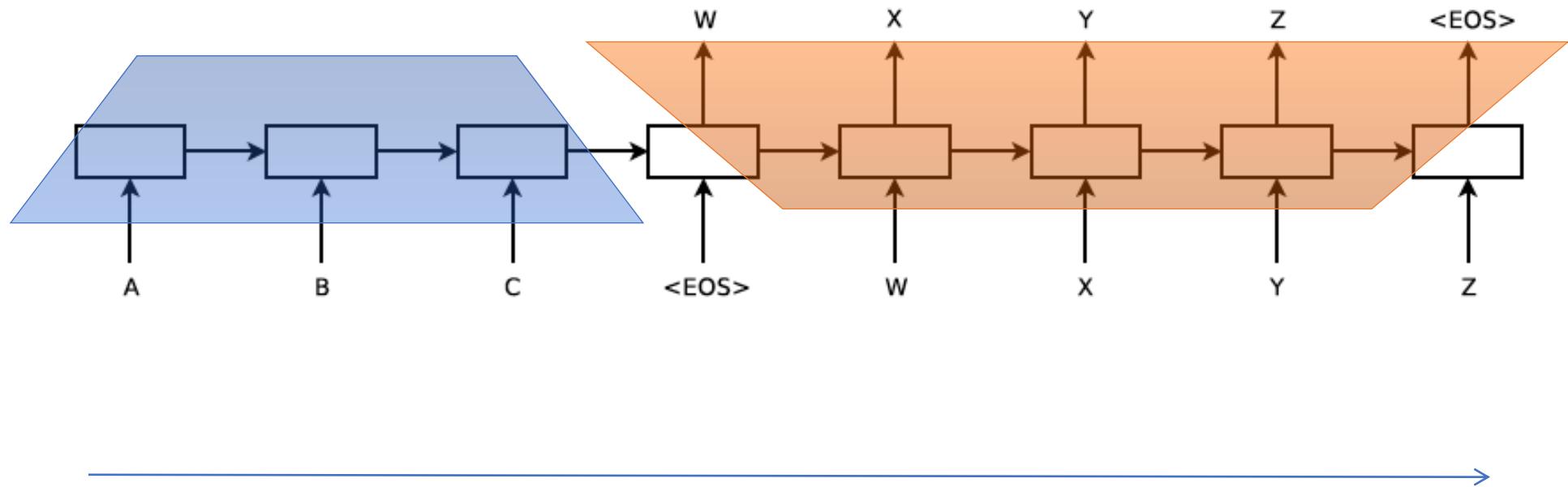
2014

# Example: Neural Machine Translation

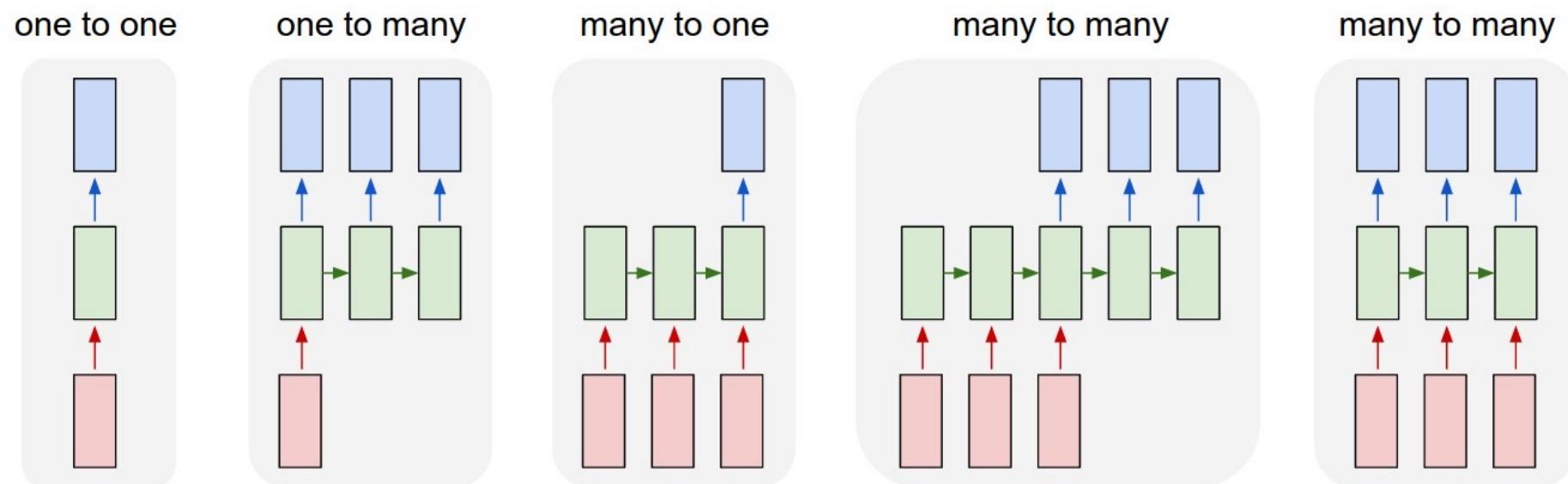
# Neural Machine Translation

- Model

Each box is an LSTM or GRU cell.

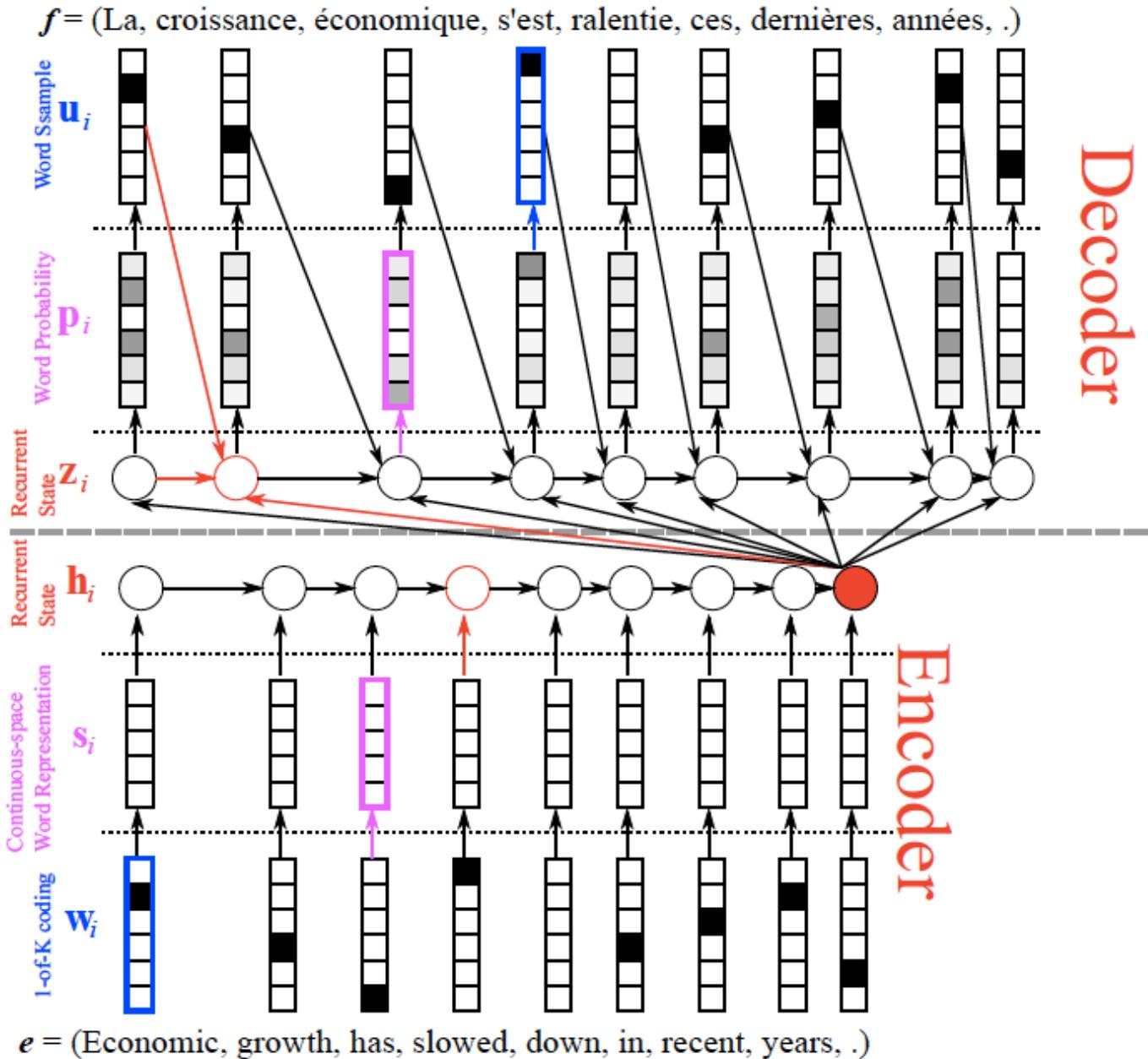


Sutskever et al. 2014



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Neural Machine Translation

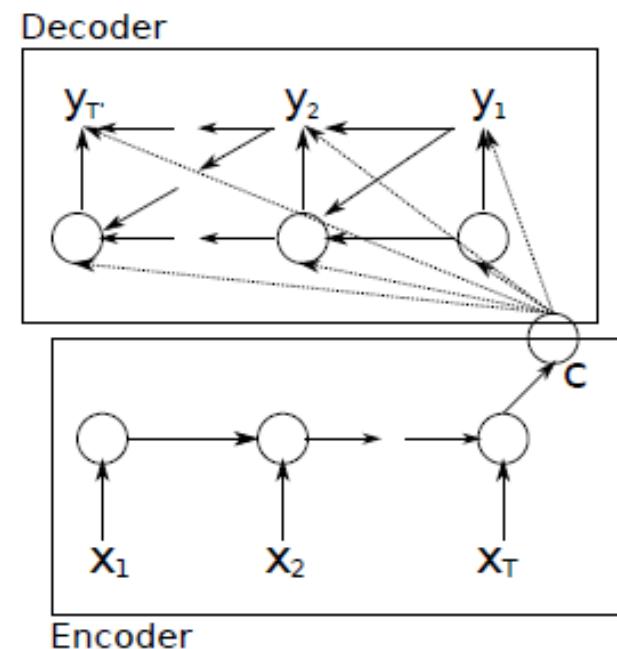


Cho: From Sequence Modeling to Translation

# Encoder-decoder

- Jointly trained to maximize

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(y_n | x_n),$$



# Attention

# Attention

Published as a conference paper at ICLR 2015

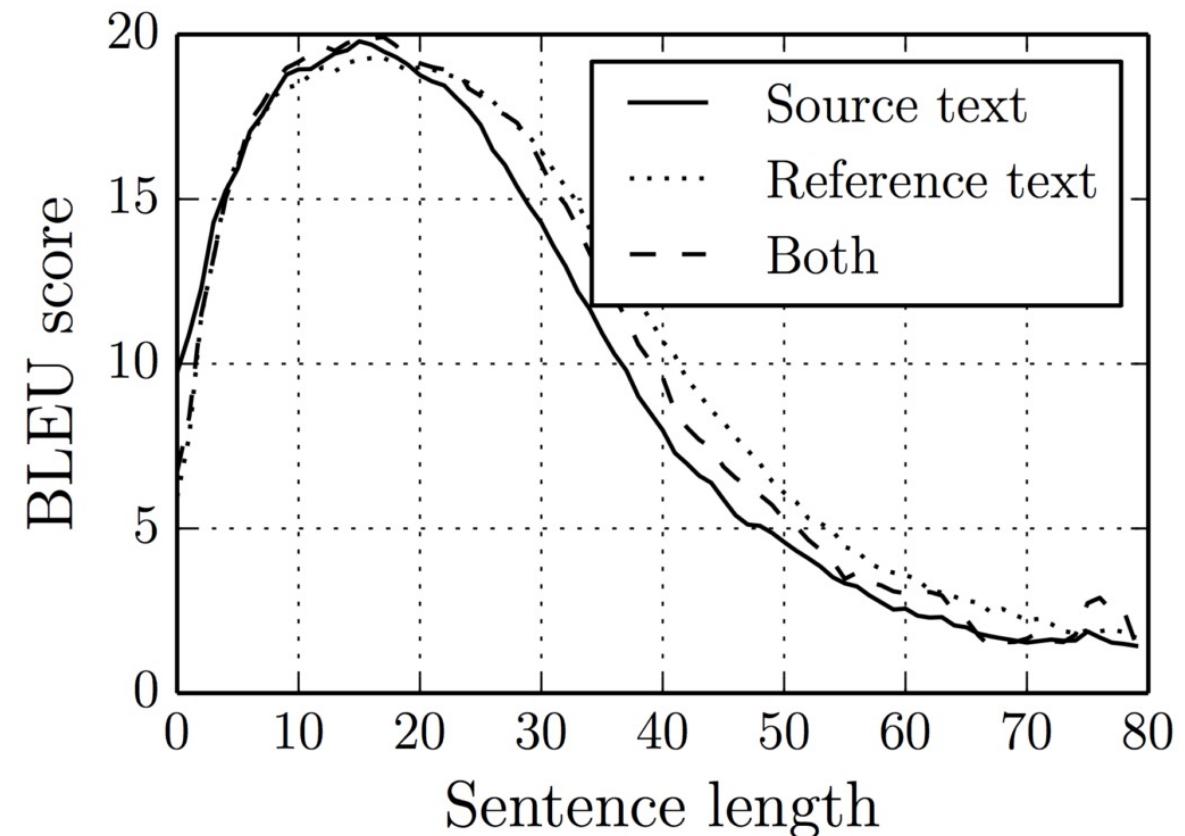
## NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

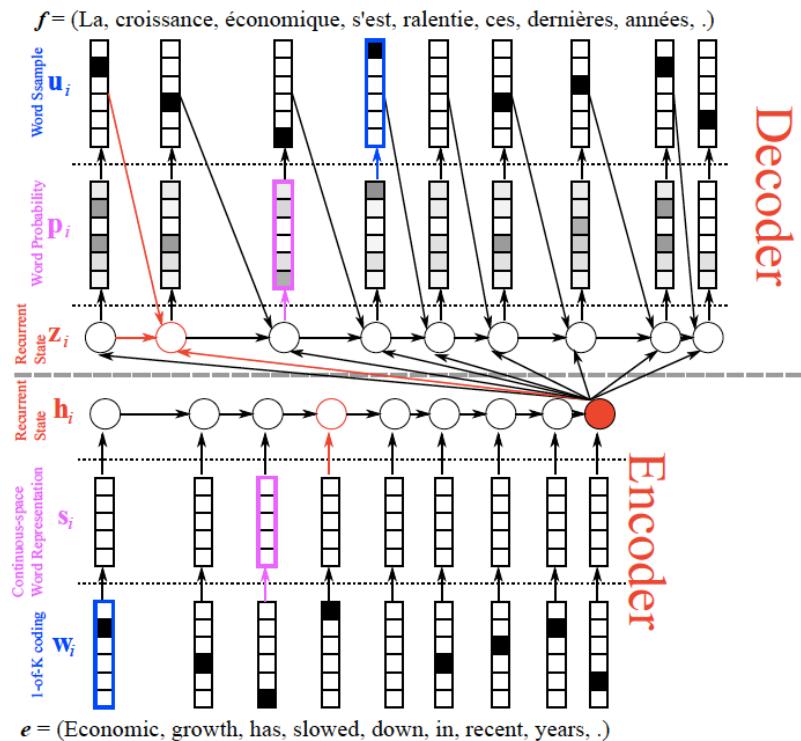
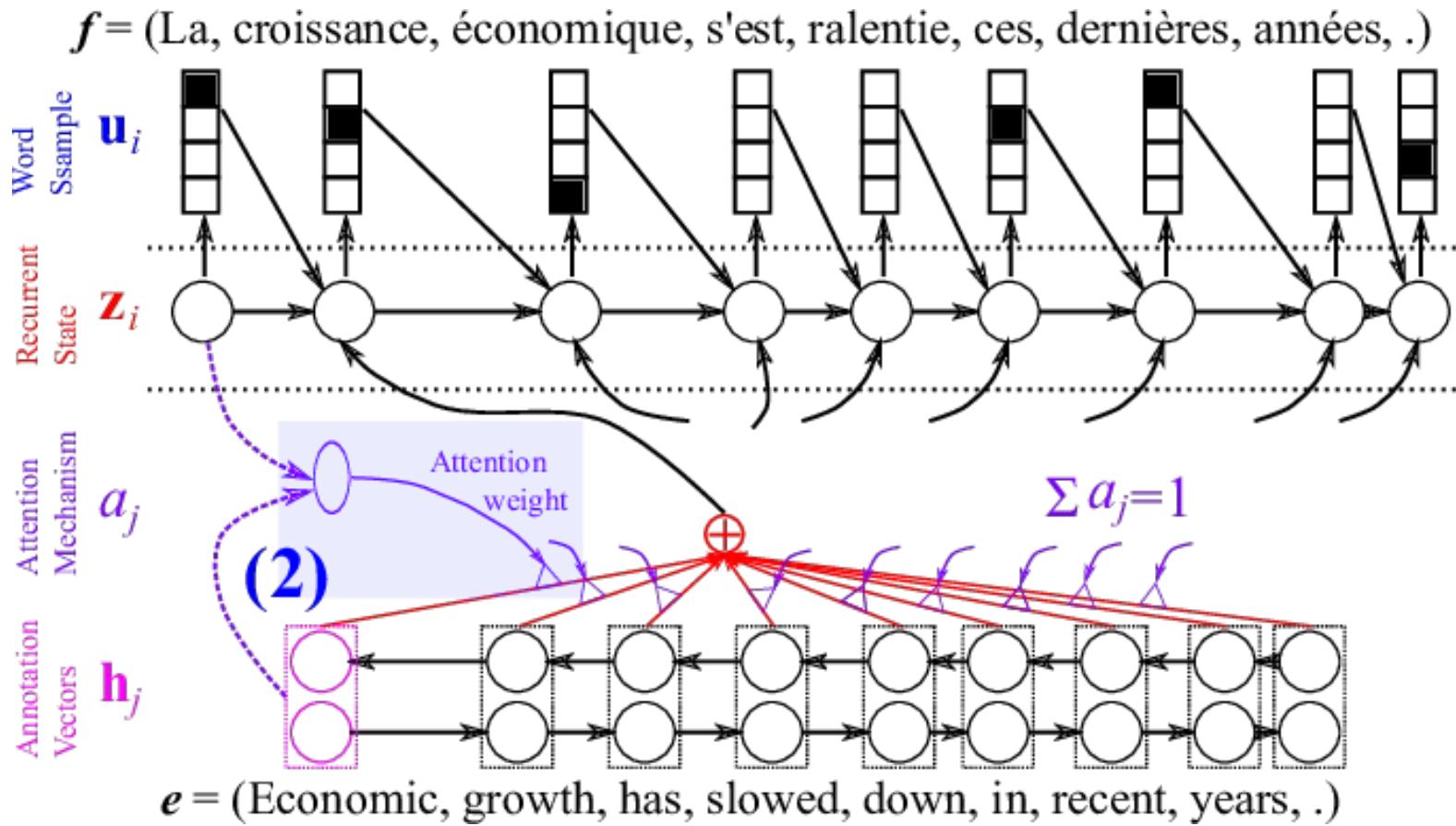
Dzmitry Bahdanau

Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio\*

Université de Montréal





# Attention

Published as a conference paper at ICLR 2015

## NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau  
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio\*  
Université de Montréal

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i), \quad (4)$$

where  $s_i$  is an RNN hidden state for time  $i$ , computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

It should be noted that unlike the existing encoder-decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector  $c_i$  for each target word  $y_i$ .

The context vector  $c_i$  depends on a sequence of *annotations*  $(h_1, \dots, h_{T_x})$  to which an encoder maps the input sentence. Each annotation  $h_i$  contains information about the whole input sequence with a strong focus on the parts surrounding the  $i$ -th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector  $c_i$  is, then, computed as a weighted sum of these annotations  $h_i$ :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

The weight  $\alpha_{ij}$  of each annotation  $h_j$  is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \quad (6)$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$

is an *alignment model* which scores how well the inputs around position  $j$  and the output at position  $i$  match. The score is based on the RNN hidden state  $s_{i-1}$  (just before emitting  $y_i$ , Eq. (4)) and the  $j$ -th annotation  $h_j$  of the input sentence.

We parametrize the alignment model  $a$  as a feedforward neural network which is jointly trained with all the other components of the proposed system. Note that unlike in traditional machine translation,

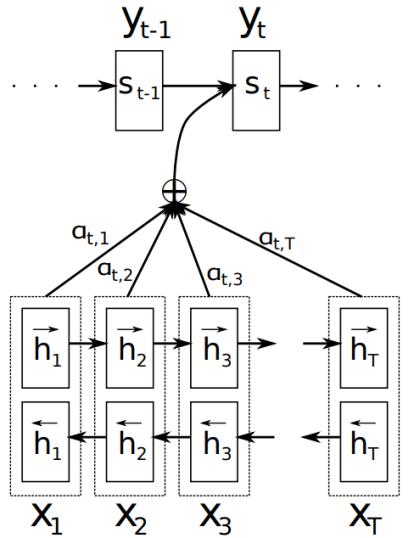
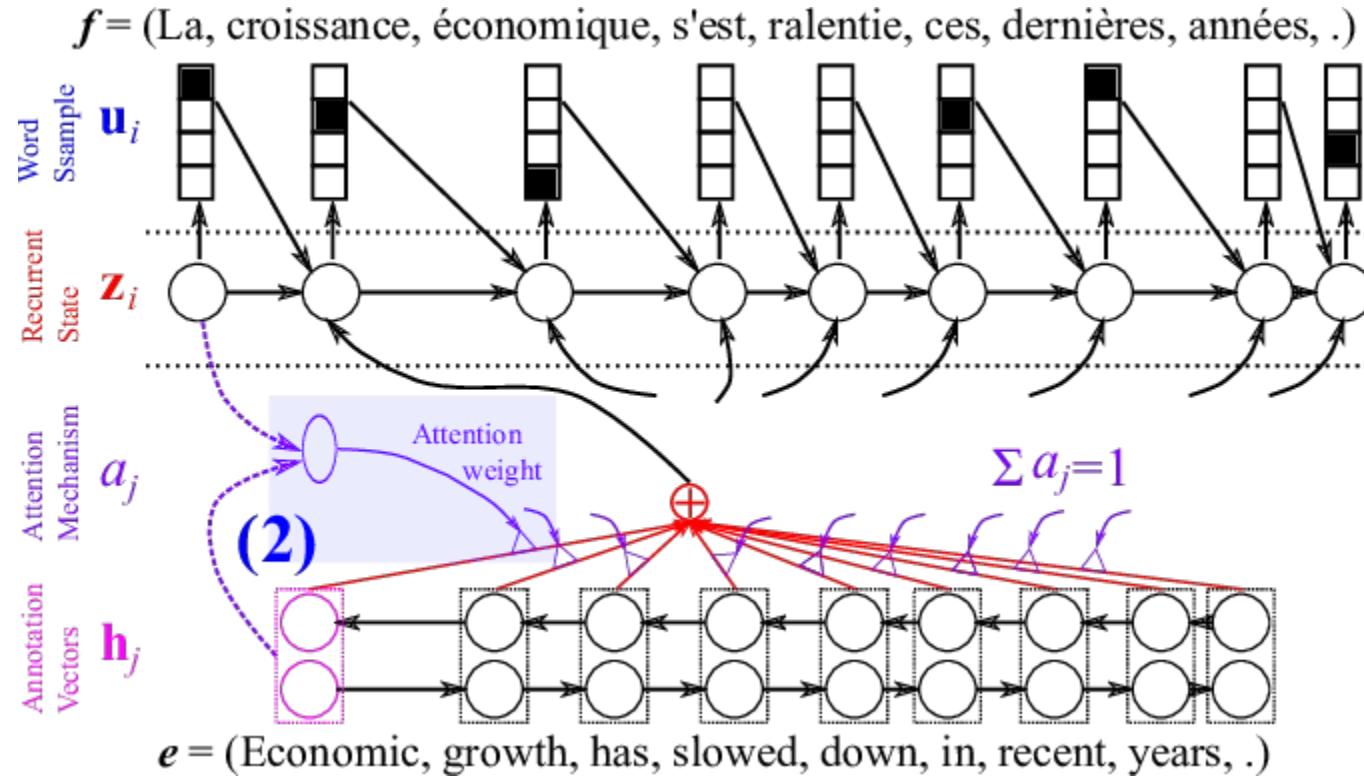


Figure 1: The graphical illustration of the proposed model trying to generate the  $t$ -th target word  $y_t$  given a source sentence  $(x_1, x_2, \dots, x_T)$ .

# Attention



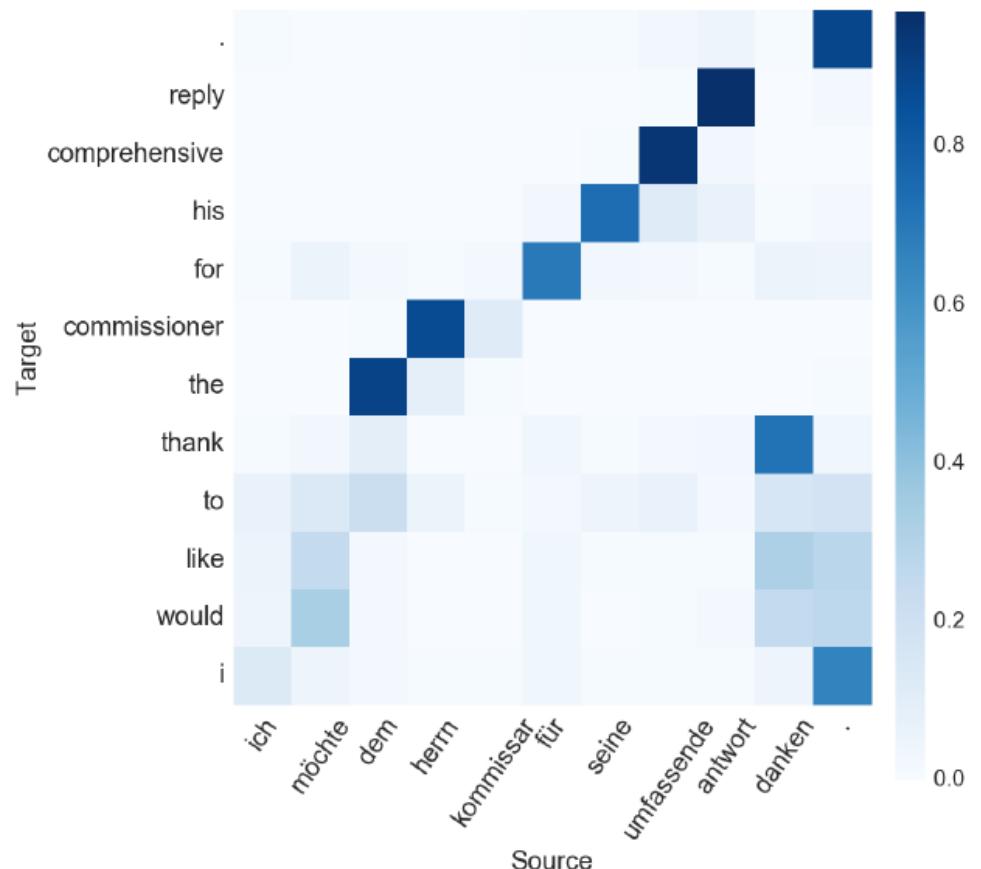
Attention mechanism: A two-layer neural network.

Input:  $z_i$  and  $h_j$

Output:  $e_j$ , a scalar for the importance of word  $j$ .

The scores of words are normalized:  $a_j = \text{softmax}(e_j)$

# Attention



**What does Attention in Neural Machine Translation  
Pay Attention to?**

**Hamidreza Ghader and Christof Monz**  
Informatics Institute, University of Amsterdam, The Netherlands  
h.ghader, c.monz@uva.nl

2017

# Attention Types

- Let's rewrite Bahdanau et al.'s attention model:

$$\mathbf{c}_t = \sum_{i=1}^n \alpha_{t,i} \mathbf{h}_i \quad ; \text{ Context vector for output } y_t$$

$$\begin{aligned} \alpha_{t,i} &= \text{align}(y_t, x_i) && ; \text{ How well two words } y_t \text{ and } x_i \text{ are aligned.} \\ &= \frac{\exp(\text{score}(s_{t-1}, \mathbf{h}_i))}{\sum_{i'=1}^n \exp(\text{score}(s_{t-1}, \mathbf{h}_{i'}))} && ; \text{ Softmax of some predefined alignment score..} \end{aligned}$$

$$\text{score}(s_t, \mathbf{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; \mathbf{h}_i])$$

where both  $\mathbf{v}_a$  and  $\mathbf{W}_a$  are weight matrices to be learned in the alignment model.

<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>

# Attention Types

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

(\*) Referred to as “concat” in Luong, et al., 2015 and as “additive attention” in Vaswani, et al., 2017.

(^) It adds a scaling factor  $1/\sqrt{n}$ , motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

# Vanilla Self-attention

$$e_i' = \sum_j \frac{\exp(e_j^T e_i)}{\sum_m \exp(e_m^T e_i)} e_j$$

# Attention: Transformer

- Vanilla self attention:

$$e_i' = \sum_j \frac{\exp(e_j^T e_i)}{\sum_m \exp(e_m^T e_i)} e_j$$

- Scaled-dot product attention:

$$e_i' = \sum_j \frac{\exp(\mathbf{k}(e_j^T) \mathbf{q}(e_i))}{\sum_m \exp(\mathbf{k}(e_m^T) \mathbf{q}(e_i))} \mathbf{v}(e_j)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Ashish Vaswani\*  
 Google Brain  
 avaswani@google.com

Noam Shazeer\*  
 Google Brain  
 noam@google.com

Niki Parmar\*  
 Google Research  
 nikip@google.com

Jakob Uszkoreit\*  
 Google Research  
 usz@google.com

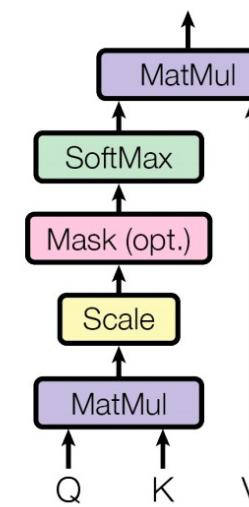
Llion Jones\*  
 Google Research  
 llion@google.com

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

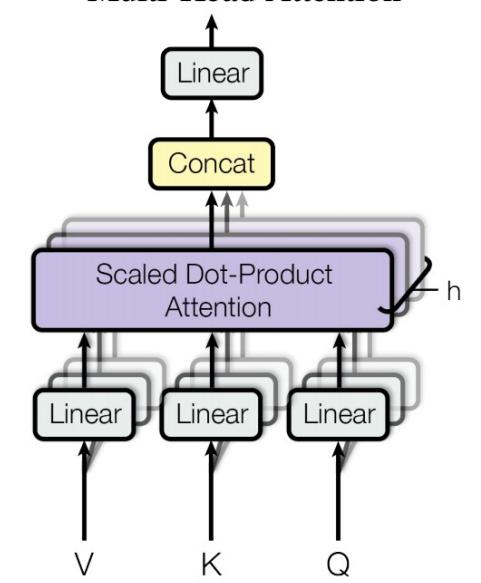
Łukasz Kaiser\*  
 Google Brain  
 lukaszkaiser@google.com

Illia Polosukhin\* ‡  
 illia.polosukhin@gmail.com

Scaled Dot-Product Attention



Multi-Head Attention



Input

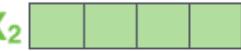
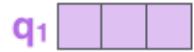
Thinking

Machines

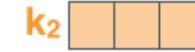
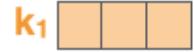
Embedding



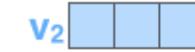
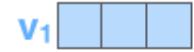
Queries

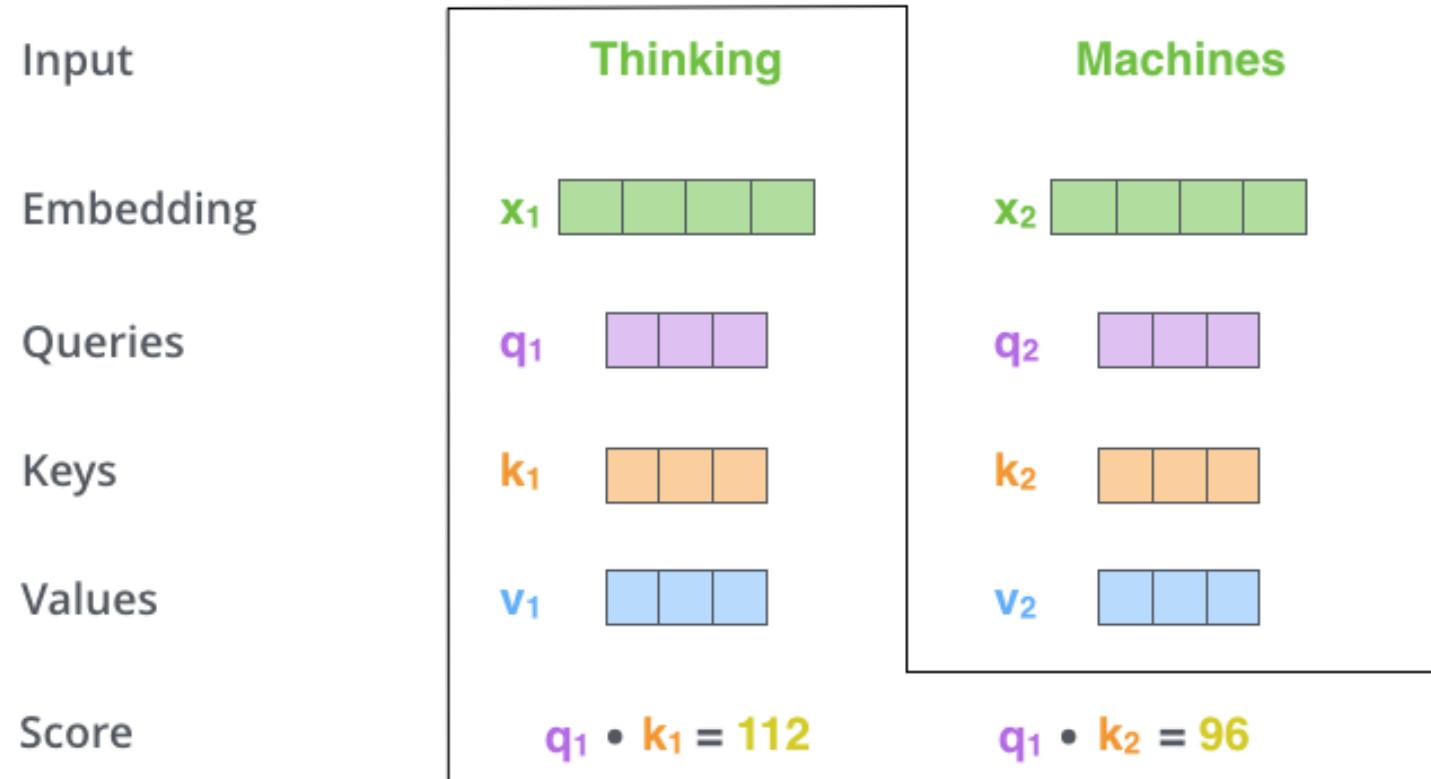


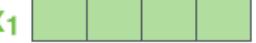
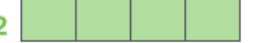
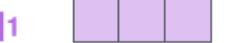
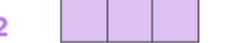
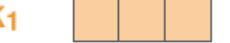
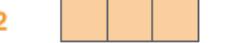
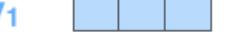
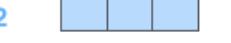
Keys



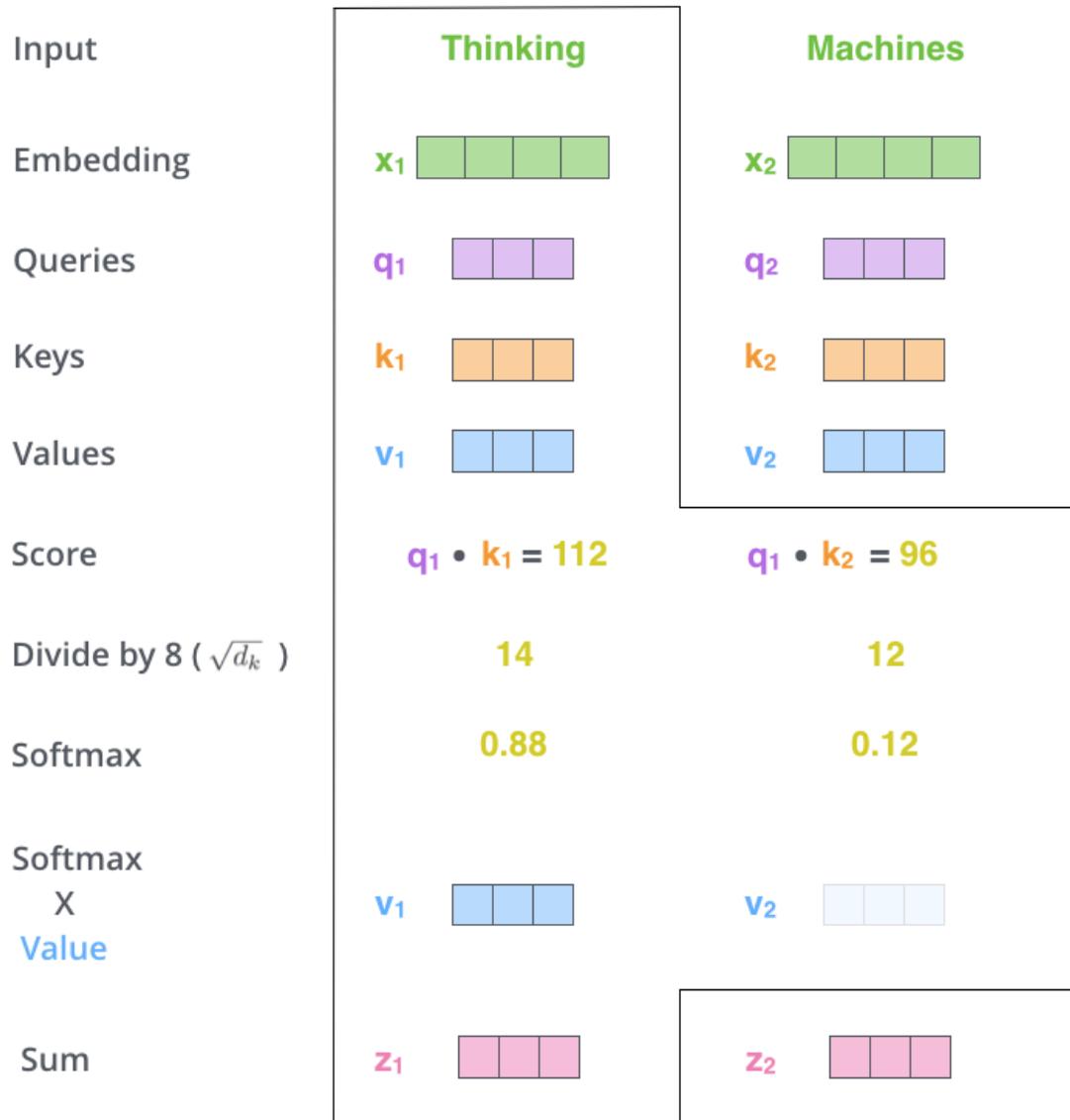
Values



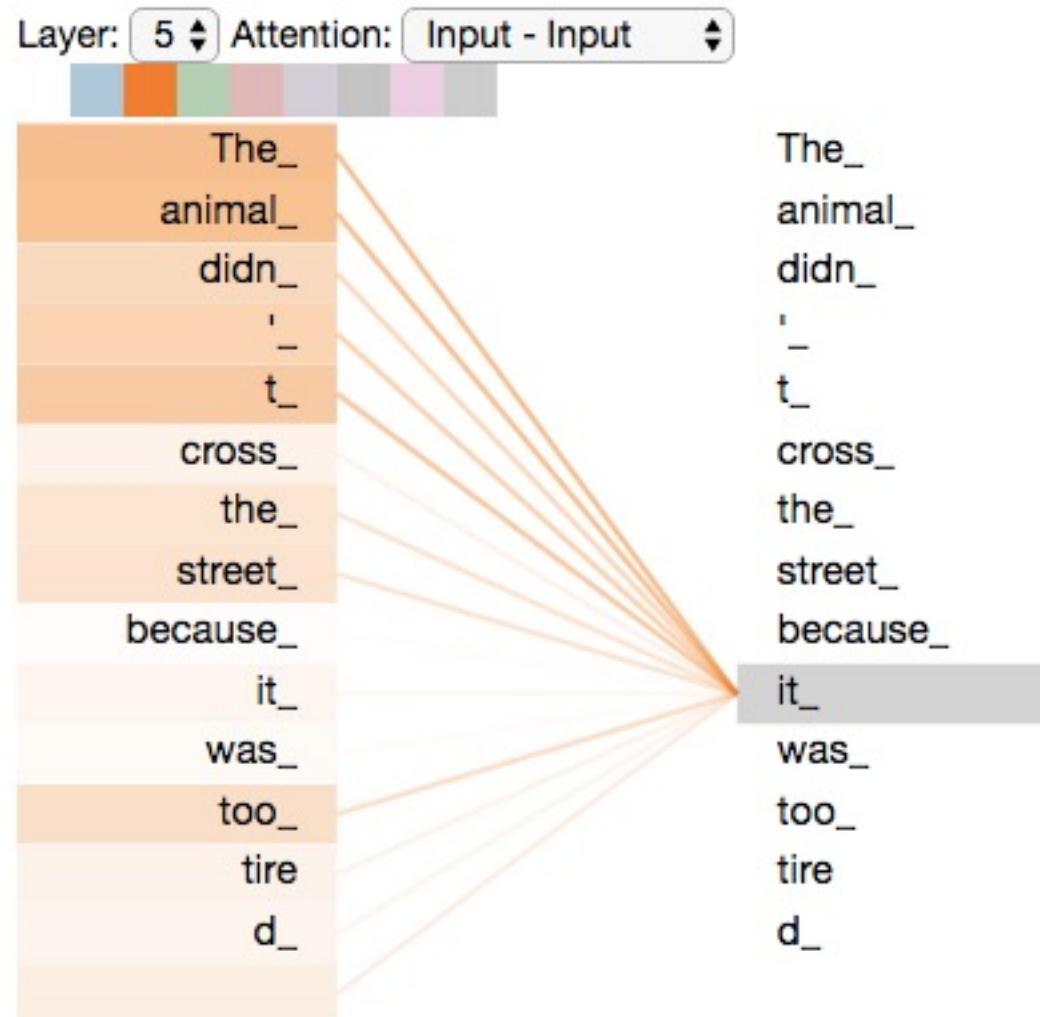


Input	<b>Thinking</b> $x_1$ 		<b>Machines</b> $x_2$ 
Embedding	$q_1$ 	$q_2$ 	
Queries	$k_1$ 	$k_2$ 	
Keys	$v_1$ 	$v_2$ 	
Values	$q_1 \cdot k_1 = 112$		
Score	$q_1 \cdot k_2 = 96$		
Divide by 8 ( $\sqrt{d_k}$ )	14	12	
Softmax	0.88	0.12	

<https://jalamar.github.io/illustrated-transformer/>



<https://jalamar.github.io/illustrated-transformer/>



<https://jalamar.github.io/illustrated-transformer/>

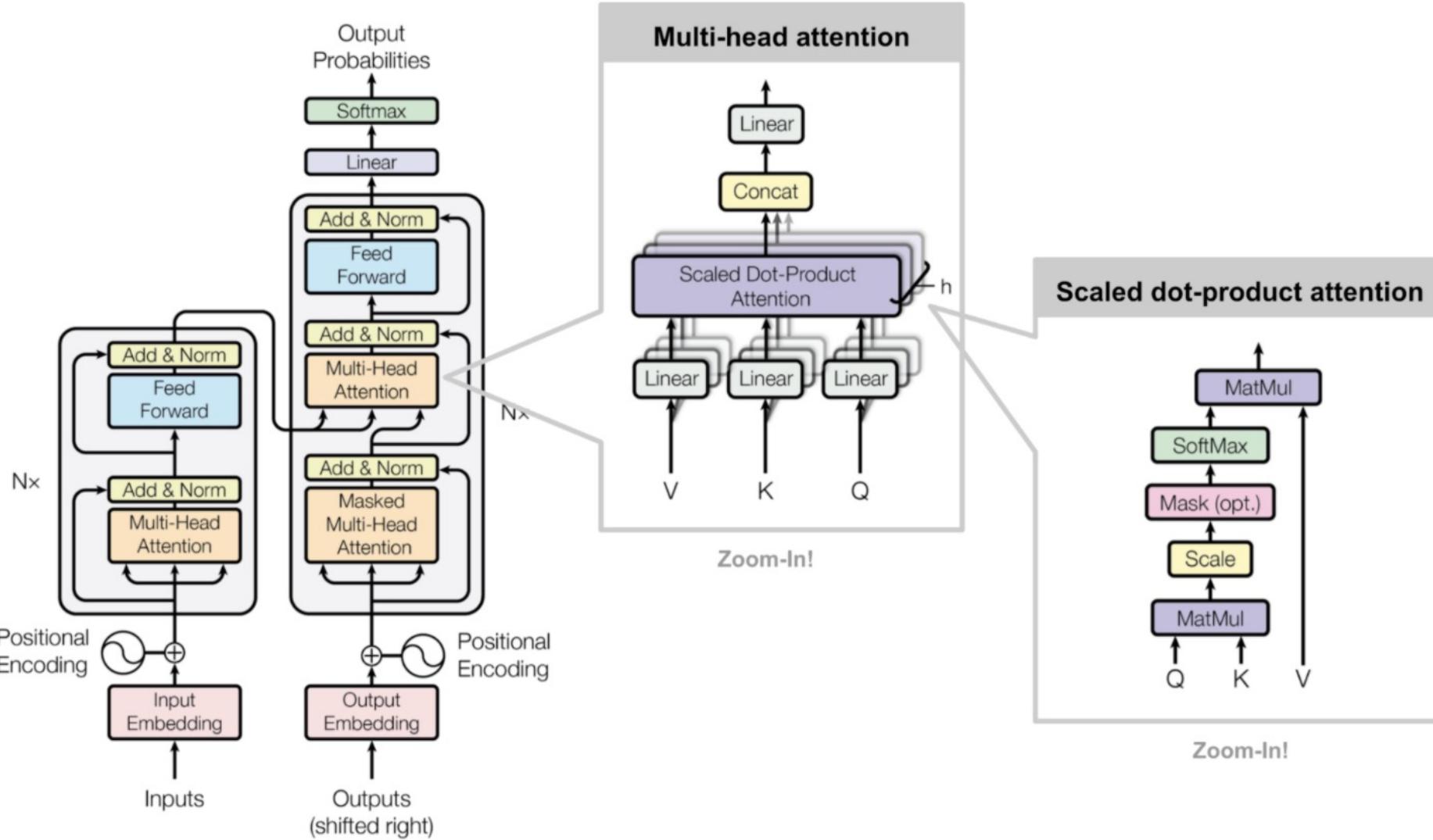


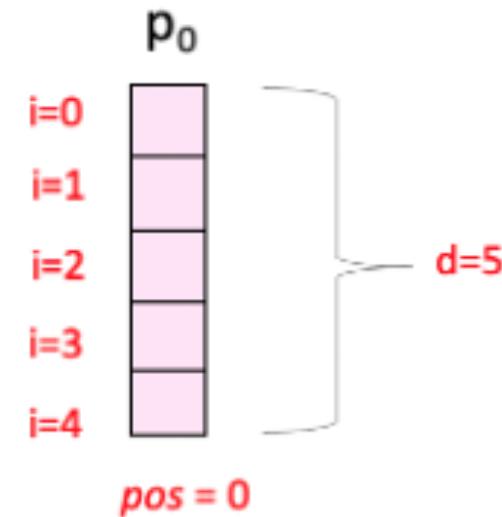
Fig. 17. The full model architecture of the transformer. (Image source: Fig 1 & 2 in [Vaswani, et al., 2017](#).)

<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>

# Positional Encoding

---

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



---

Fig from: <https://www.youtube.com/watch?v=dichIcUZfOw>

# Positional Encoding

Position Embeddings

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

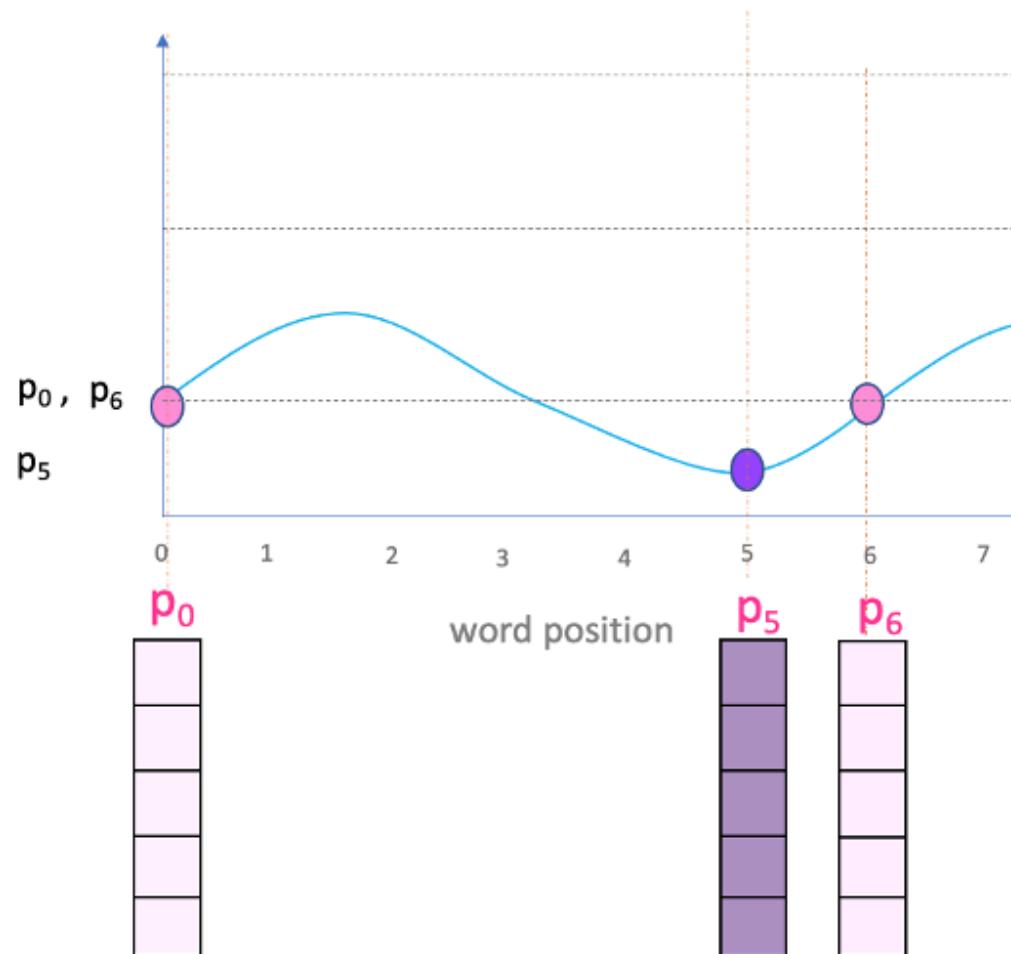


Fig from: <https://www.youtube.com/watch?v=dichIcUZfOw>

# Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

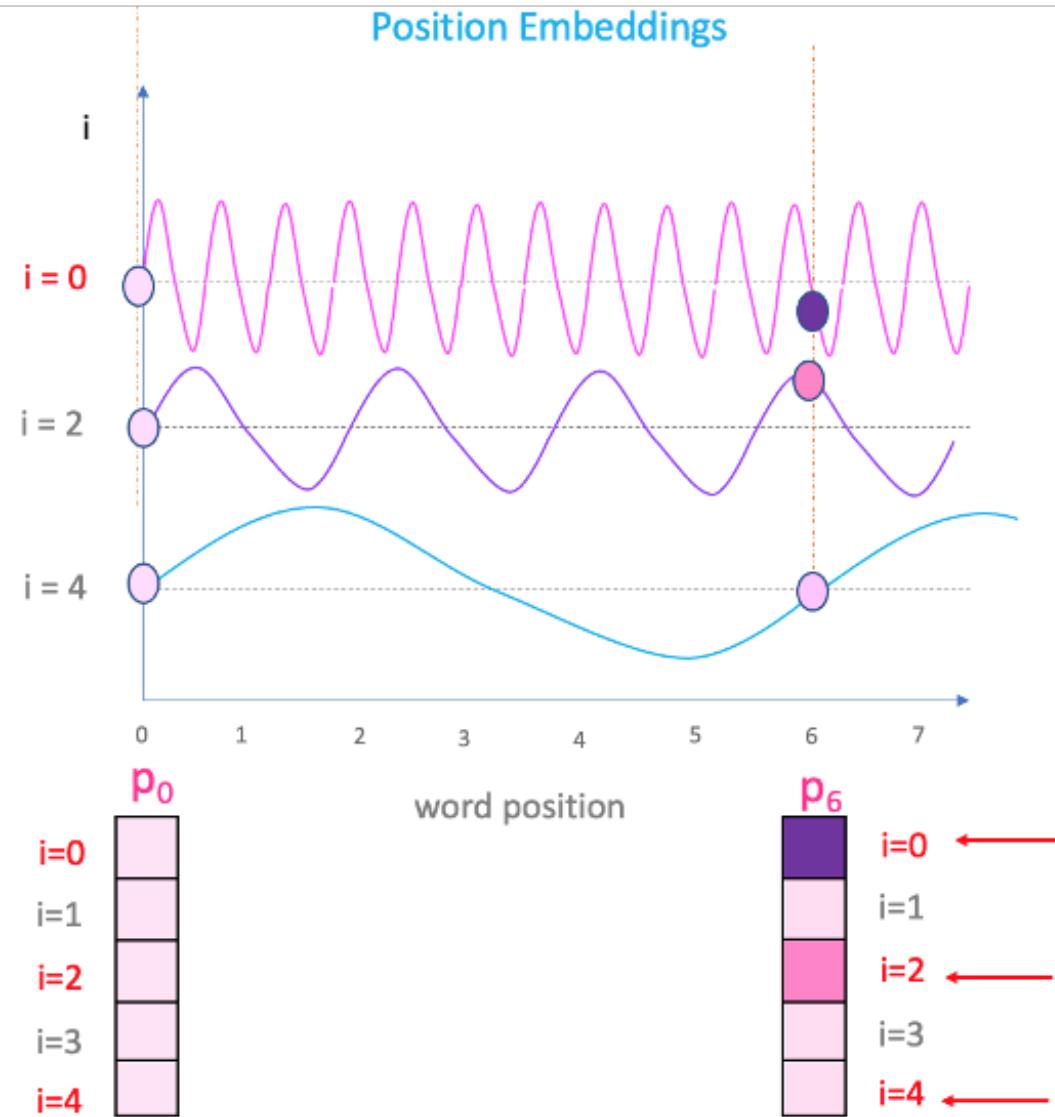
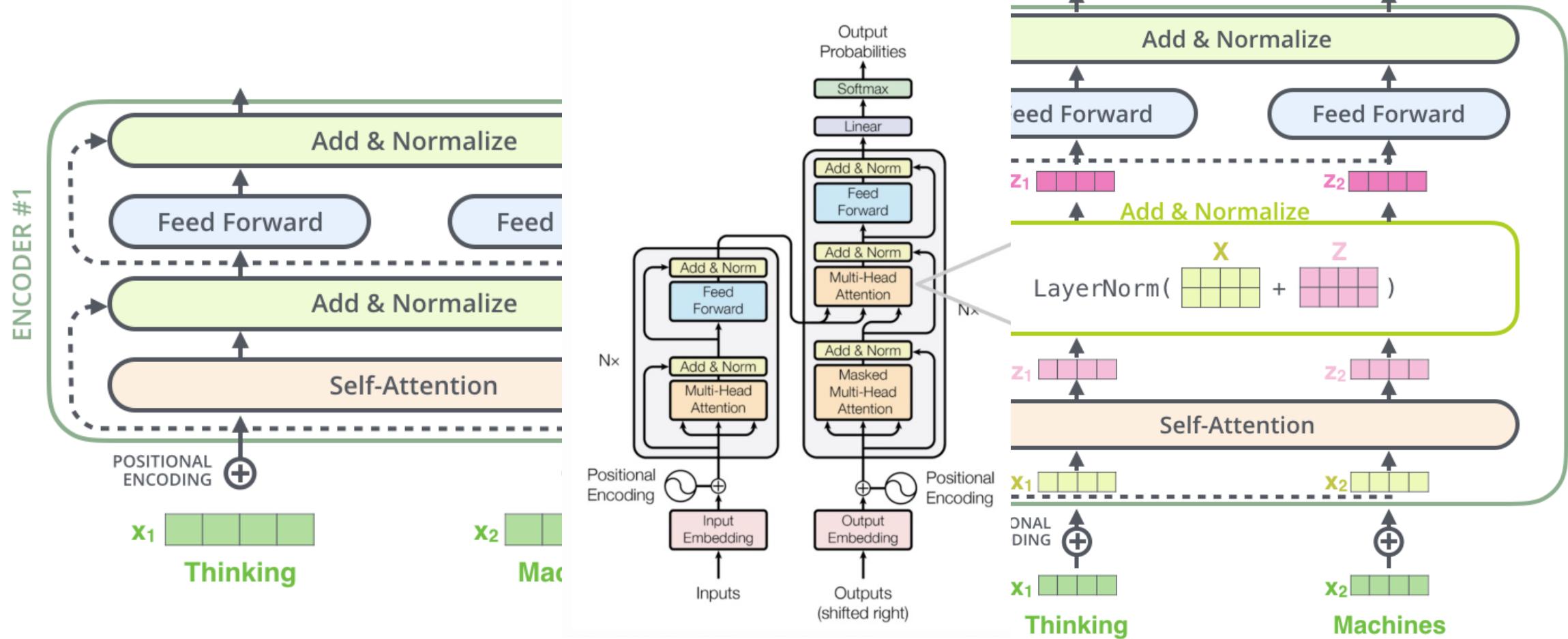


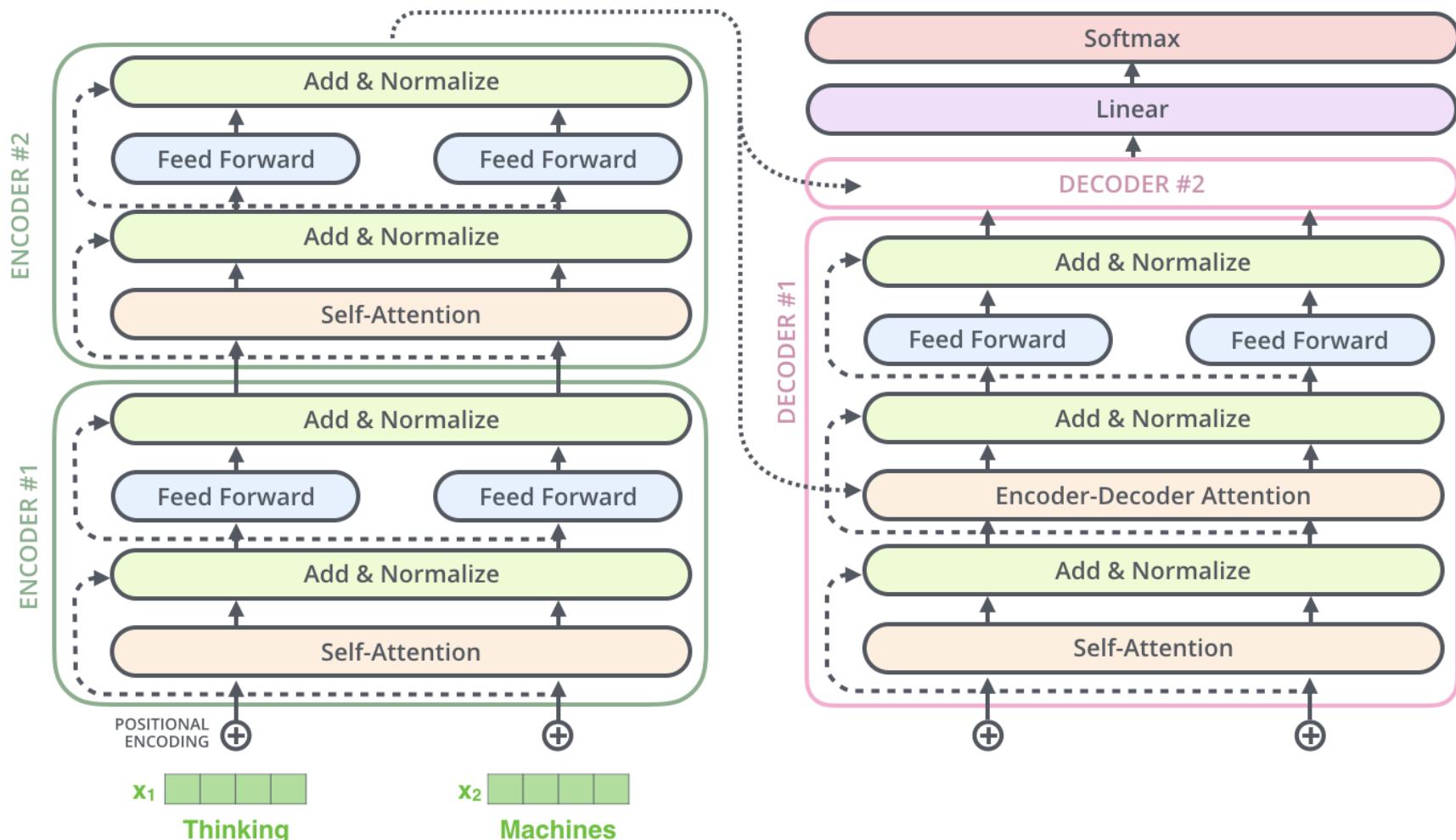
Fig from: <https://www.youtube.com/watch?v=dichIcUZfOw>

# Skip Connections & Normalization



<https://jalammar.github.io/illustrated-transformer/>

# Skip Connections & Normalization



<https://jalamar.github.io/illustrated-transformer/>

# Transformers without Normalization

Jiachen Zhu<sup>1,2</sup>, Xinlei Chen<sup>1</sup>, Kaiming He<sup>3</sup>, Yann LeCun<sup>1,2</sup>, Zhuang Liu<sup>1,4,†</sup>

<sup>1</sup>FAIR, Meta, <sup>2</sup>New York University, <sup>3</sup>MIT, <sup>4</sup>Princeton University

†Project lead

Normalization layers are ubiquitous in modern neural networks and have long been considered essential. This work demonstrates that Transformers without normalization can achieve the same or better performance using a remarkably simple technique. We introduce Dynamic Tanh (DyT), an element-wise operation  $DyT(\mathbf{x}) = \tanh(\alpha\mathbf{x})$ , as a drop-in replacement for normalization layers in Transformers. DyT is inspired by the observation that layer normalization in Transformers often produces tanh-like, *S*-shaped input-output mappings. By incorporating DyT, Transformers without normalization can match or exceed the performance of their normalized counterparts, mostly without hyperparameter tuning. We validate the effectiveness of Transformers with DyT across diverse settings, ranging from recognition to generation, supervised to self-supervised learning, and computer vision to language models. These findings challenge the conventional understanding that normalization layers are indispensable in modern neural networks, and offer new insights into their role in deep networks.

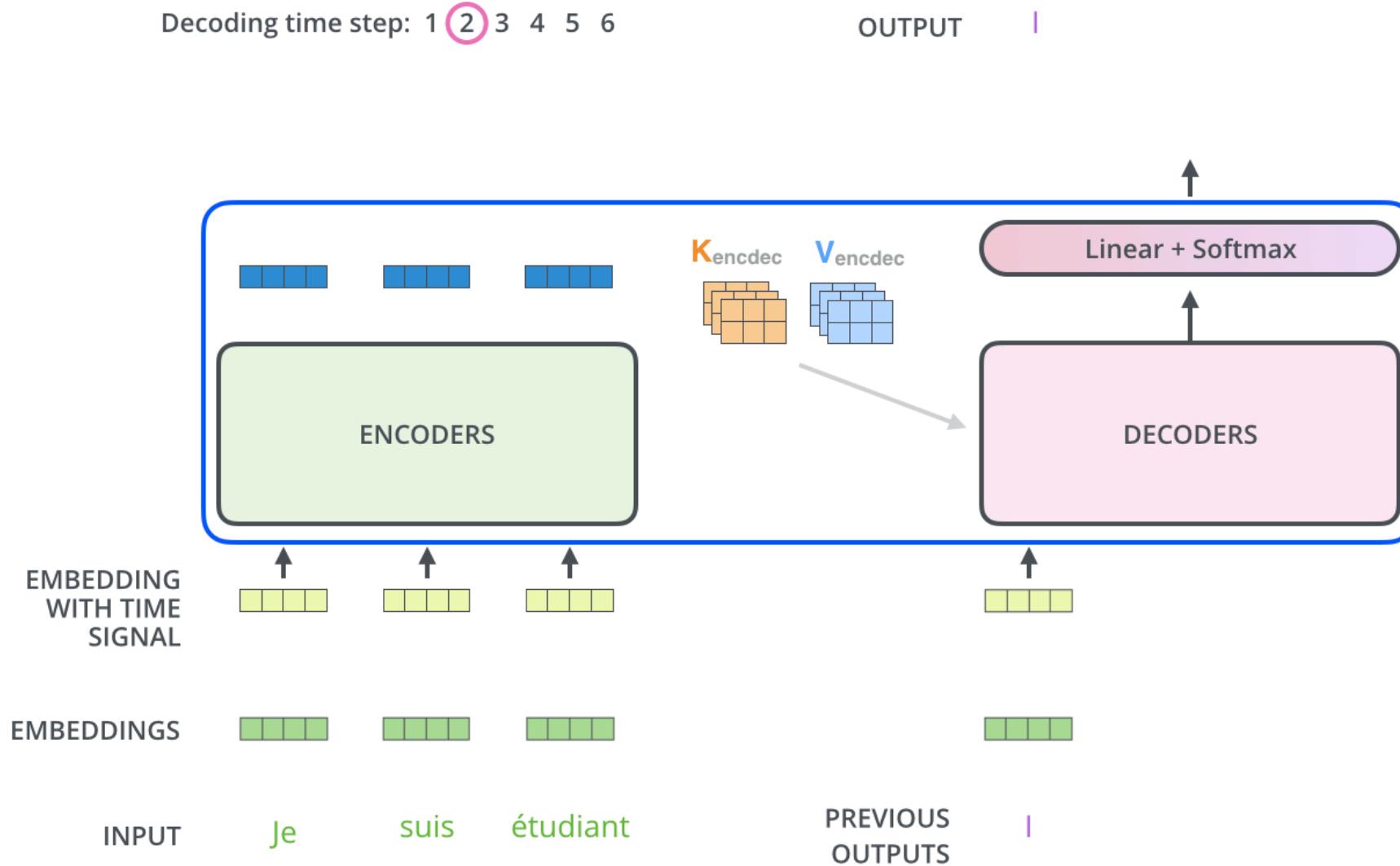
**Date:** March 14, 2025

**Project page and code:** [jiachenzhu.github.io/DyT](https://jiachenzhu.github.io/DyT)

**Correspondence:** [jiachen.zhu@nyu.edu](mailto:jiachen.zhu@nyu.edu), [zhuangl@princeton.edu](mailto:zhuangl@princeton.edu)



# Decoder



# Tutorial on transformers

- <https://e2eml.school/transformers.html>
- <https://jalammar.github.io/illustrated-transformer/>

# A Significant Issue with Self-Attention: Complexity

$$e'_i = \sum_j \frac{\exp(\mathbf{k}(e_j^T) \mathbf{q}(e_i))}{\sum_m \exp(\mathbf{k}(e_m^T) \mathbf{q}(e_i))} \mathbf{v}(e_j)$$

- If there are  $n$  tokens/embeddings,
  - Updating a single token requires  $O(n)$  operations.
  - Overall:  $O(n^2)$
- What is the complexity of an RNN layer with  $n$  time steps?

# Linear Attention

Self-attention:

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \\ A_l(x) &= V' = \text{softmax} \left( \frac{QK^T}{\sqrt{D}} \right) V. \end{aligned} \tag{2}$$

Rewrite Eq 2 for one row of the matrix:

$$V'_i = \frac{\sum_{j=1}^N \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^N \text{sim}(Q_i, K_j)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with  $\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$ .

Constraint for  $\text{sim}()$ : It should be non-negative.  
Then, we can choose any other kernel/function:

Given such a kernel with a feature representation  $\phi(x)$  we can rewrite equation 2 as follows,

$$\phi(x) = \text{elu}(x) + 1$$

$$V'_i = \frac{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j)}, \tag{4}$$

and then further simplify it by making use of the associative property of matrix multiplication to

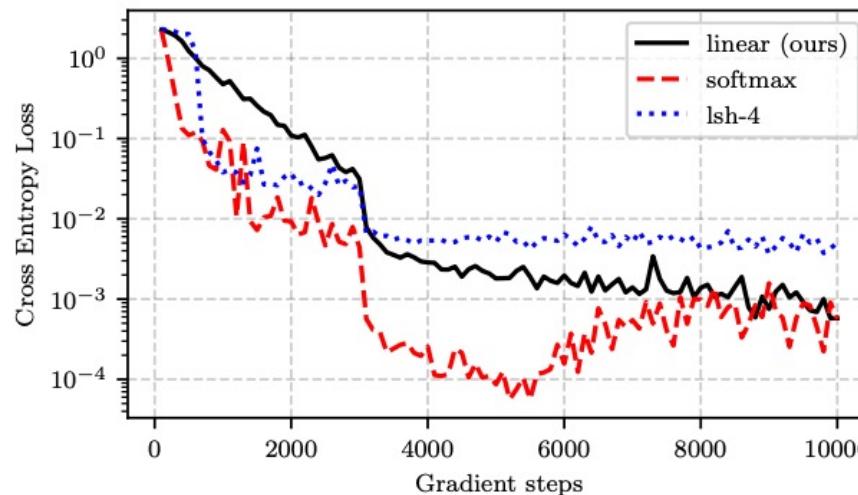
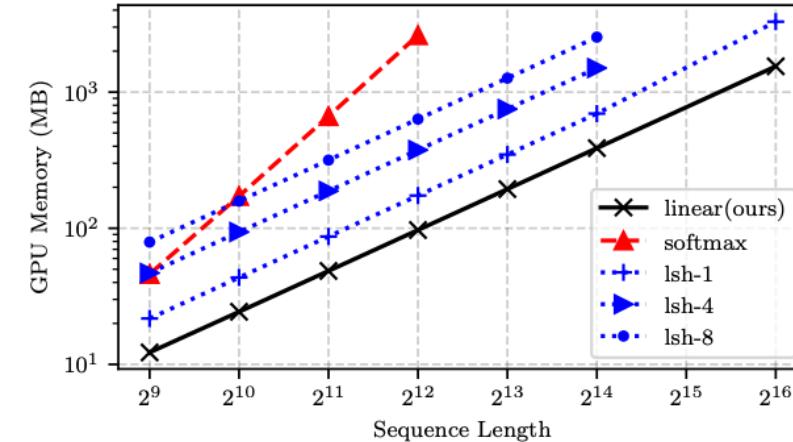
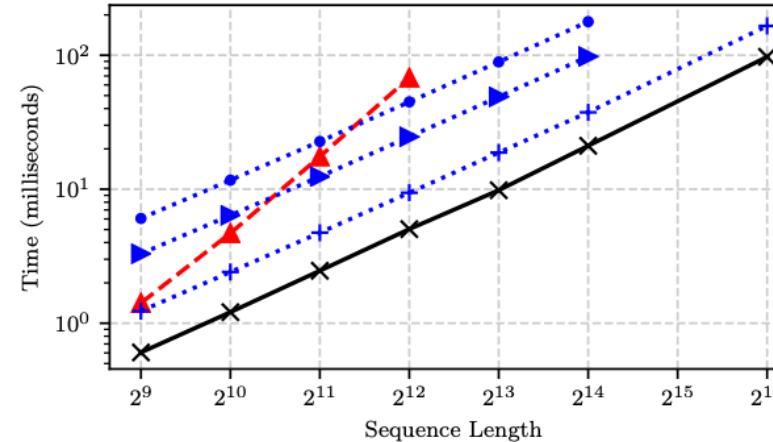
$$V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j)}. \tag{5}$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$(\phi(Q) \phi(K)^T) V = \phi(Q) (\phi(K)^T V). \tag{6}$$

Note that the feature map  $\phi(\cdot)$  is applied rowwise to the matrices  $Q$  and  $K$ .

# Linear Attention



Method	Validation PER	Time/epoch (s)
Bi-LSTM	10.94	1047
Softmax	5.12	2711
LSH-4	9.33	2250
Linear (ours)	8.08	<b>824</b>

Table 3: Performance comparison in automatic speech recognition on the WSJ dataset. The results are given in the form of phoneme error rate (PER) and training time per epoch. Our model outperforms the LSTM and Reformer while being faster to train and evaluate. Details of the exper-

# Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu<sup>\*</sup><sup>1</sup> and Tri Dao<sup>\*</sup><sup>2</sup>

<sup>1</sup>Machine Learning Department, Carnegie Mellon University

<sup>2</sup>Department of Computer Science, Princeton University  
agu@cs.cmu.edu, tri@tridao.me

Rejected at ICLR2024

# Pre-training in NLP

- Word embeddings are the basis of deep learning for NLP



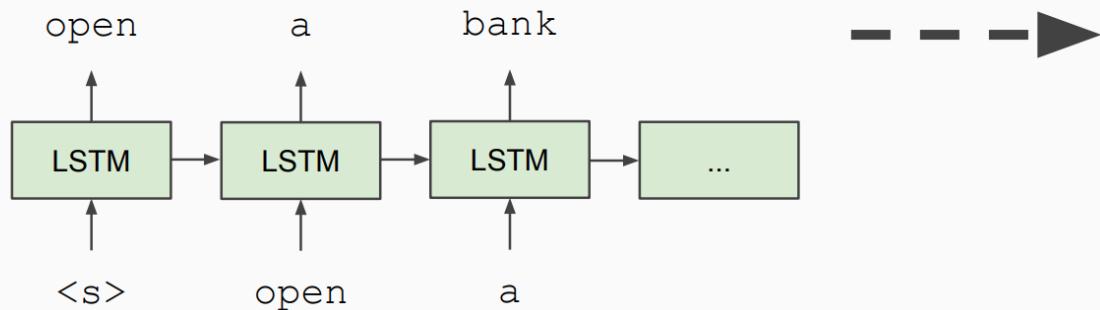
- Word embeddings (word2vec, GloVe) are often *pre-trained* on text corpus from co-occurrence statistics



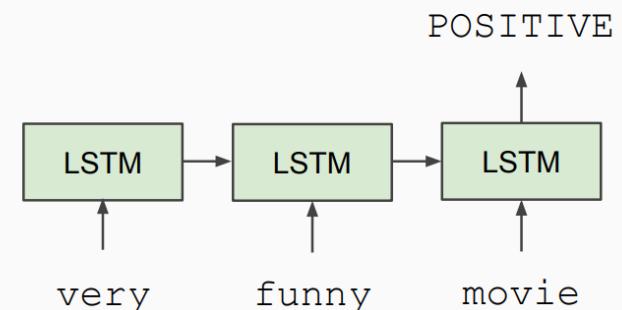
# Pre-training in NLP

- *Semi-Supervised Sequence Learning*, Google, 2015

## Train LSTM Language Model



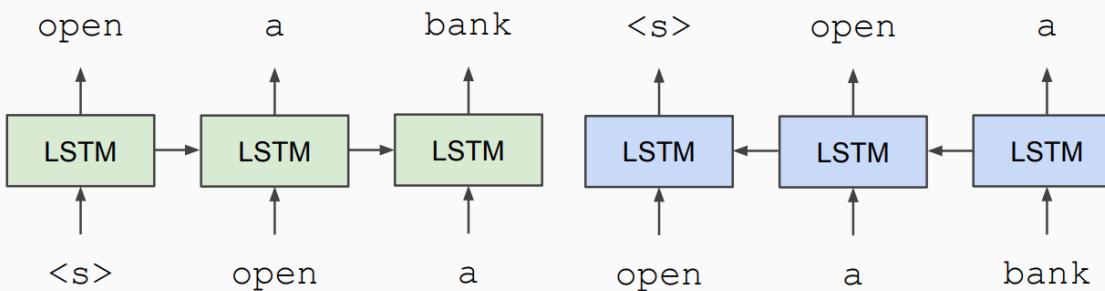
## Fine-tune on Classification Task



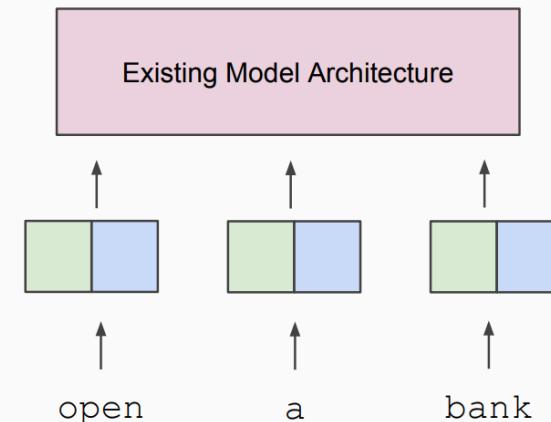
# Pre-training in NLP

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

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



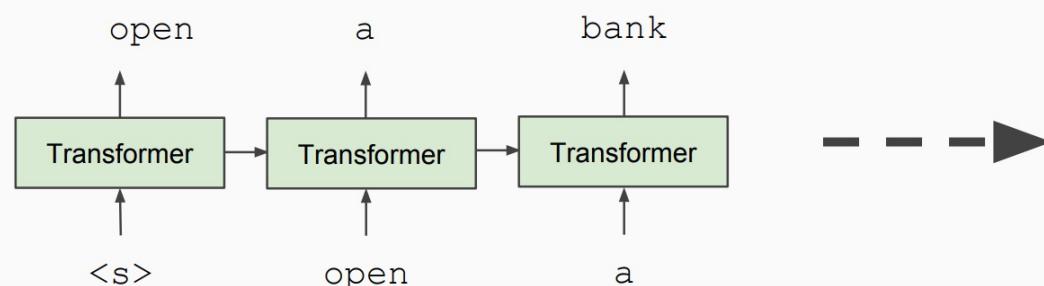
**Apply as “Pre-trained Embeddings”**



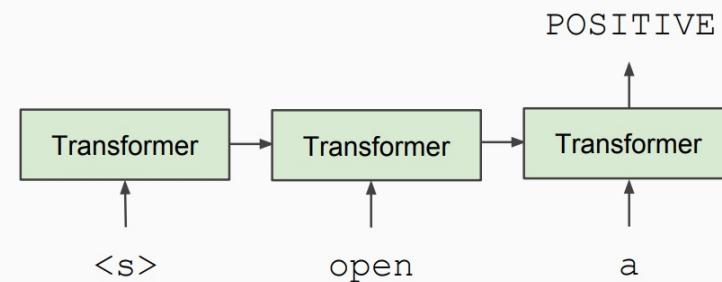
# Pre-training in NLP

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

## Train Deep (12-layer) Transformer LM



## Fine-tune on Classification Task



# GPT-1

# GPT-1

- 12 layer decoder-only transformer
- Unsupervised pretraining
  - BookCorpus dataset
- Supervised finetuning
  - Textual alignment
  - QA & commonsense reasoning
  - Semantic similarity
  - Classification

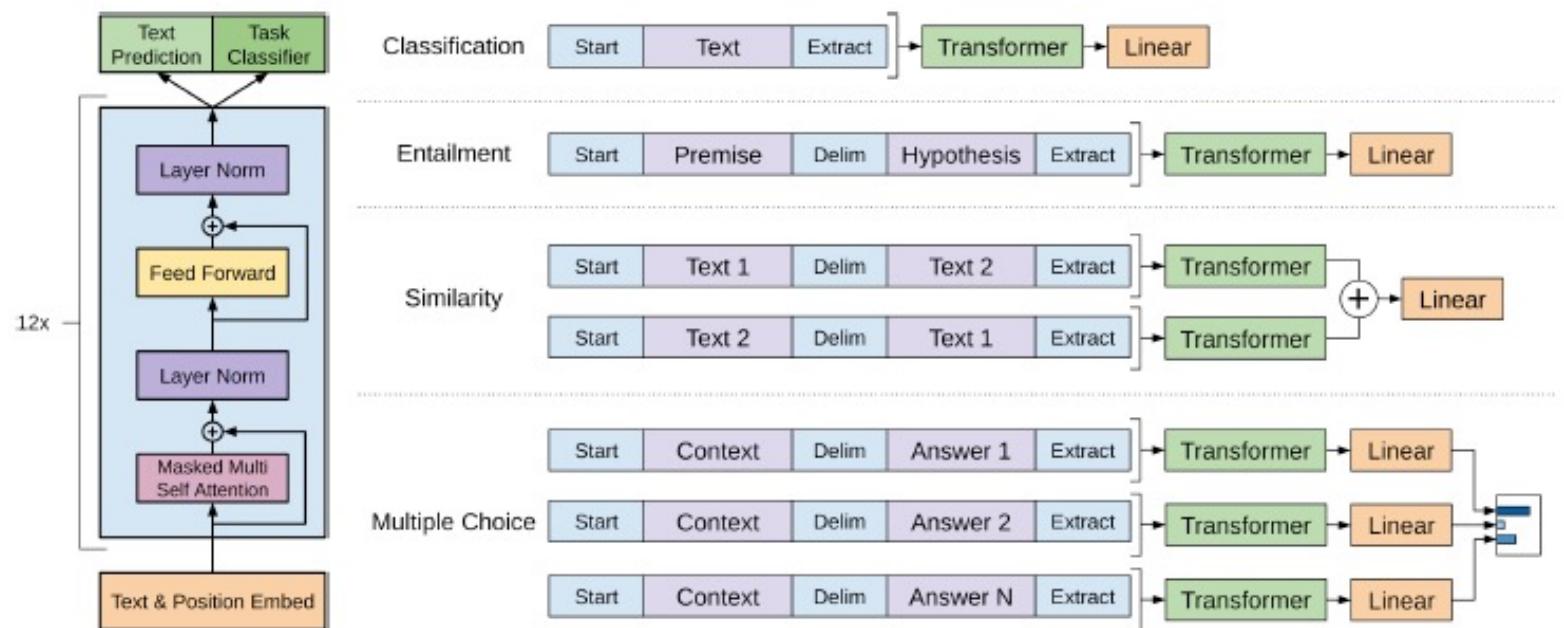


Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

# GPT-1

Given an unsupervised corpus of tokens  $\mathcal{U} = \{u_1, \dots, u_n\}$ , we use a standard language modeling objective to maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta) \quad (1)$$

where  $k$  is the size of the context window, and the conditional probability  $P$  is modeled using a neural network with parameters  $\Theta$ . These parameters are trained using stochastic gradient descent [51].

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \text{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \text{softmax}(h_n W_e^T) \end{aligned} \quad (2)$$

where  $U = (u_{-k}, \dots, u_{-1})$  is the context vector of tokens,  $n$  is the number of layers,  $W_e$  is the token embedding matrix, and  $W_p$  is the position embedding matrix.

# GPT-1

## Discriminative Fine-tuning

For labeled downstream task, maximize the log probability on each pair of instance  $(x, y)$

After training the model with the objective in Eq. 1, we adapt the parameters to the supervised target task. We assume a labeled dataset  $\mathcal{C}$ , where each instance consists of a sequence of input tokens,  $x^1, \dots, x^m$ , along with a label  $y$ . The inputs are passed through our pre-trained model to obtain the final transformer block's activation  $h_l^m$ , which is then fed into an added linear output layer with parameters  $W_y$  to predict  $y$ :

$$P(y|x^1, \dots, x^m) = \text{softmax}(h_l^m W_y). \quad (3)$$

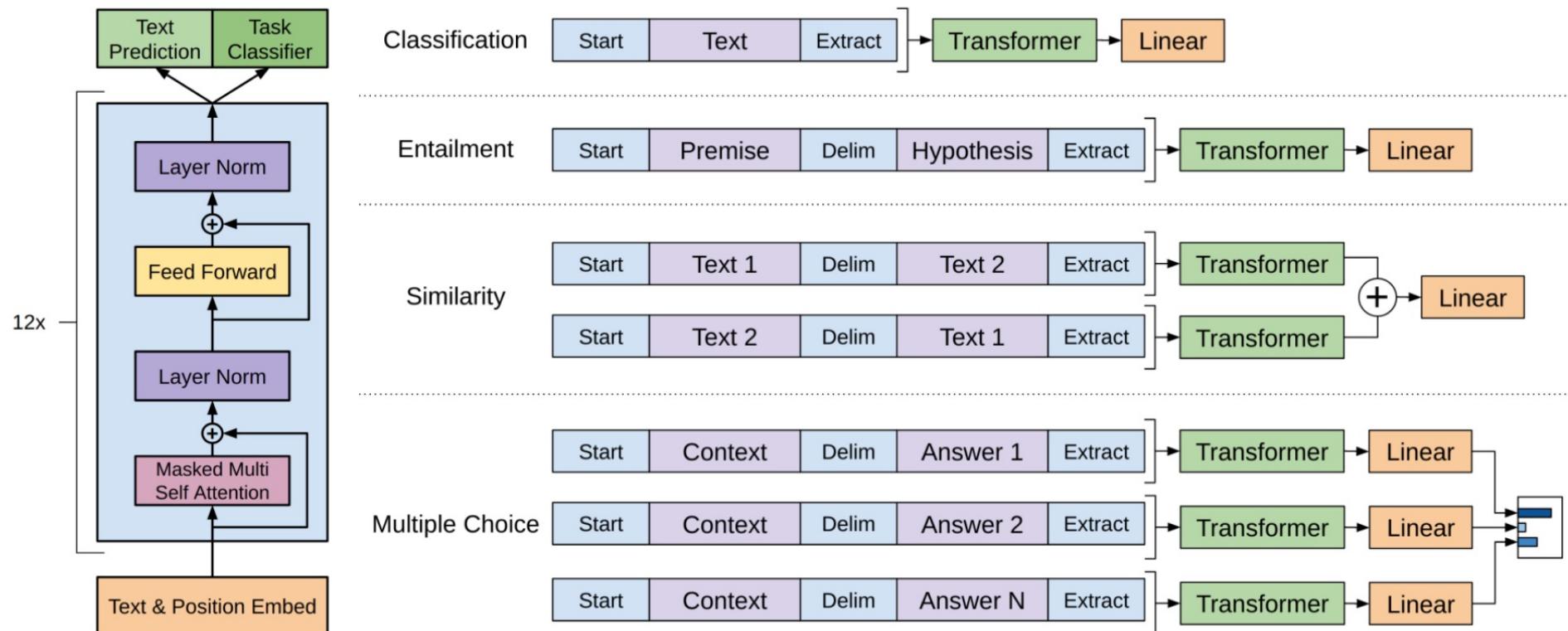
This gives us the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \dots, x^m). \quad (4)$$

Add auxiliary fine-tuning objective of language modeling will imporove the performance  $L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$

# GPT-1

## Discriminative Fine-tuning



# GPT-1 Results

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

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

# BERT

# BERT

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

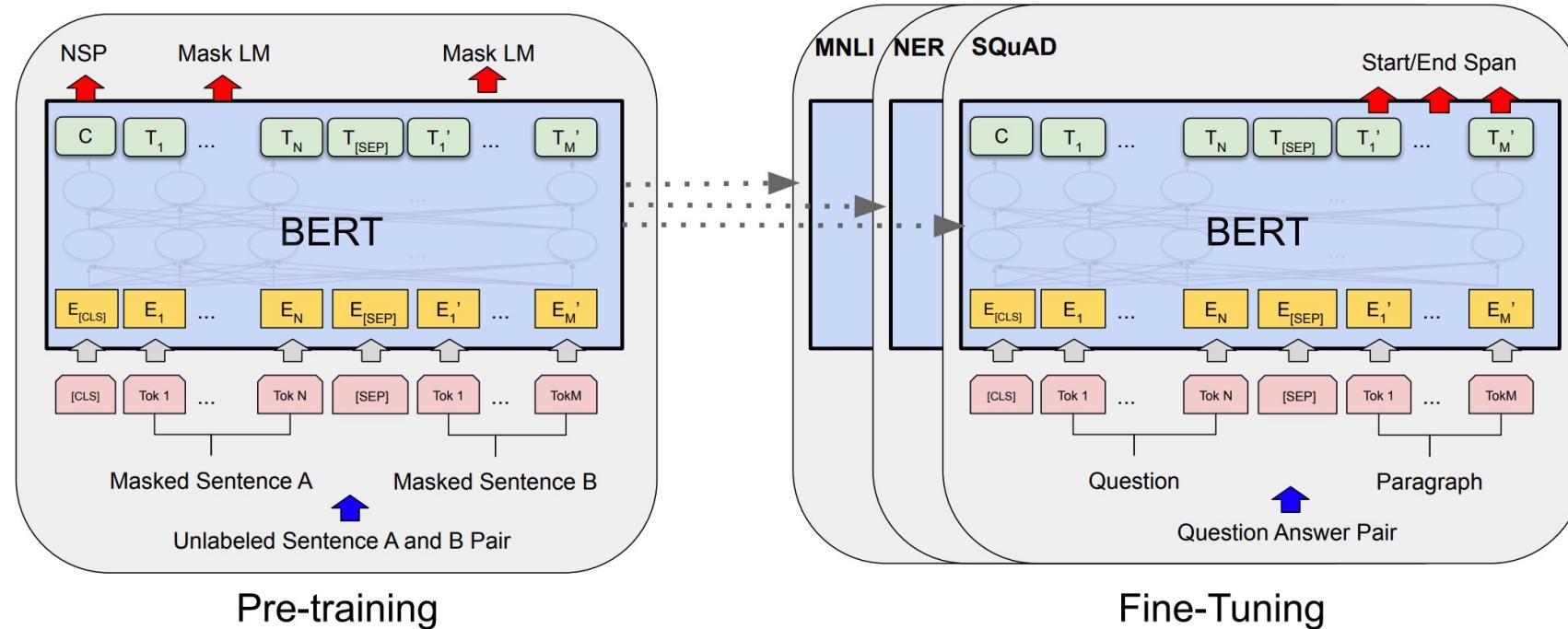


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

# BERT: NLP Tasks

- MNLI: Multi-Genre Natural Language Inference

## Examples

### Premise

#### *Fiction*

The Old One always comforted Ca'daan, except today.

### Label

### Hypothesis

*neutral*

Ca'daan knew the Old One very well.

#### *Letters*

Your gift is appreciated by each and every student who will benefit from your generosity.

*neutral*

Hundreds of students will benefit from your generosity.

#### *Telephone Speech*

yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or *contradiction* August is a black out month for vacations in the company.

#### *9/11 Report*

At the other end of Pennsylvania Avenue, people began to line up for a White House tour.

*entailment*

People formed a line at the end of Pennsylvania Avenue.

- NER: Named Entity Recognition
- SQuAD: Stanford Question Answering Dataset

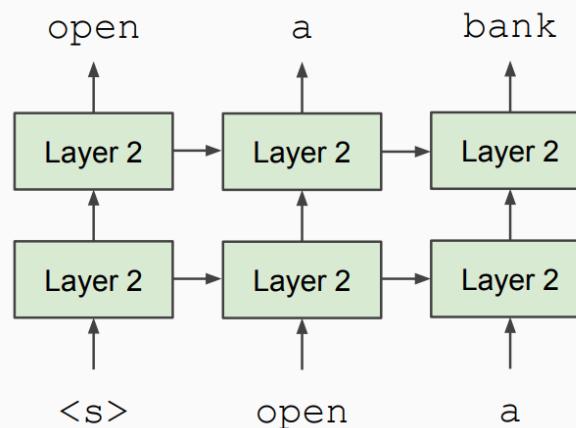
# BERT: Motivation

- **Problem:** Language models only use left context or right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - We don't care about this.
- Reason 2: Words can “see themselves” in a bidirectional encoder.

# BERT: Motivation

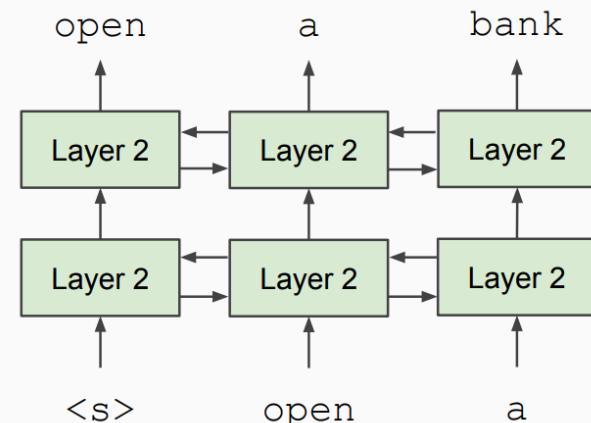
## Unidirectional context

Build representation incrementally



## Bidirectional context

Words can “see themselves”



Actually, this is vanilla attention that  
is not causal!

# BERT: Motivation

- **Solution:** Mask out  $k\%$  of the input words, and then predict the masked words
  - We always use  $k = 15\%$

the man went to the [MASK] to buy a [MASK] of milk

↑                              ↑  
store                          gallon

- Too little masking: Too expensive to train
- Too much masking: Not enough context

# BERT: Motivation

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, replace with [MASK]  
went to the store → went to the [MASK]
  - 10% of the time, replace random word  
went to the store → went to the running
  - 10% of the time, keep same  
went to the store → went to the store

# BERT: Motivation

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

**Sentence A** = The man went to the store.

**Sentence B** = He bought a gallon of milk.

**Label** = IsNextSentence

**Sentence A** = The man went to the store.

**Sentence B** = Penguins are flightless.

**Label** = NotNextSentence

# BERT

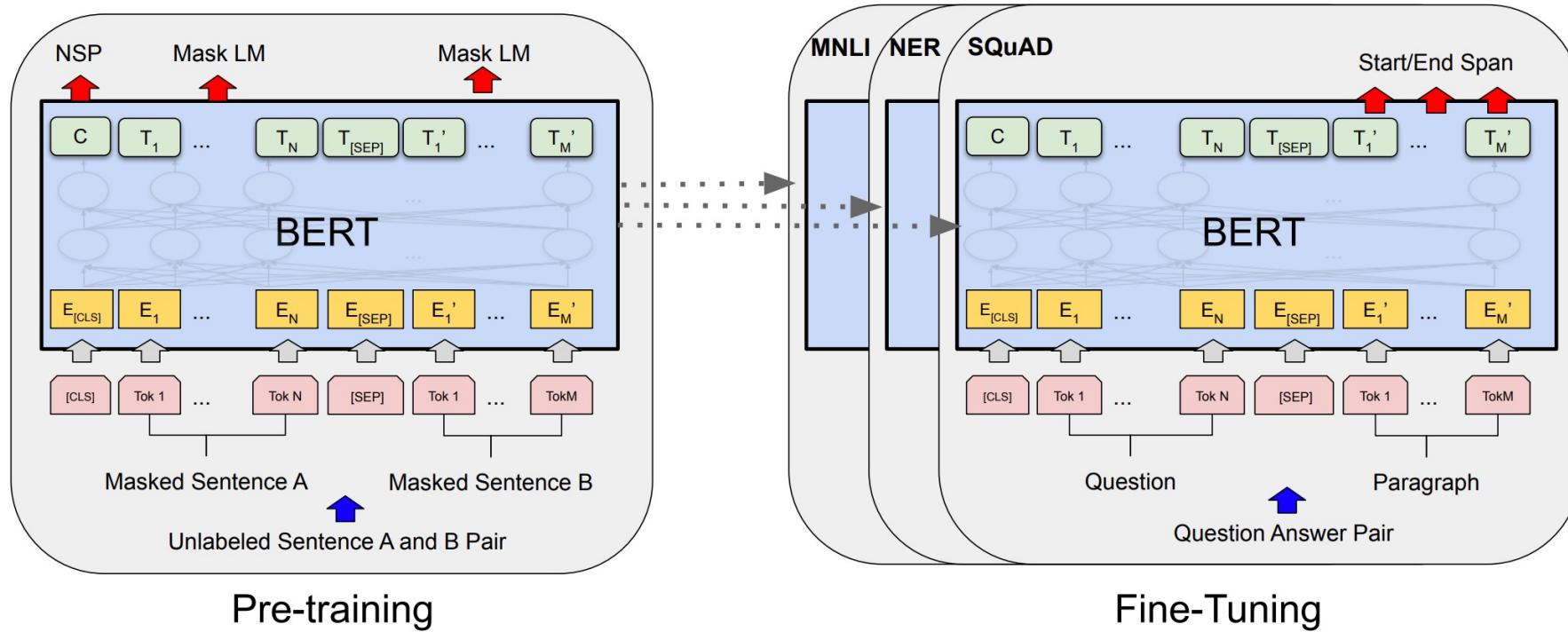
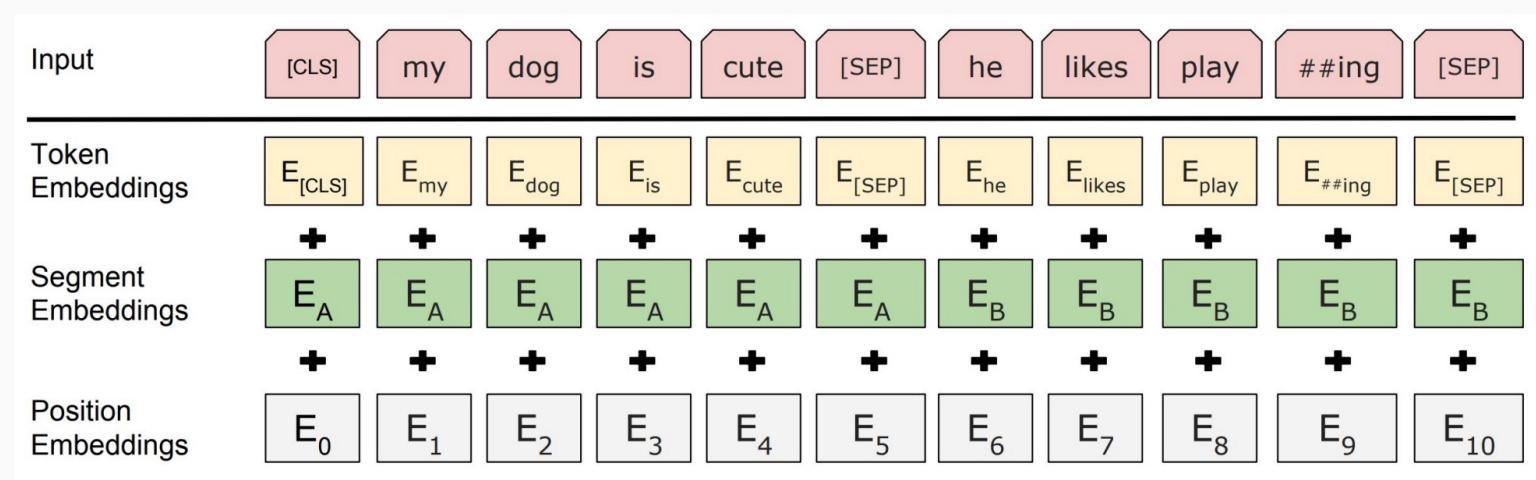


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

# BERT

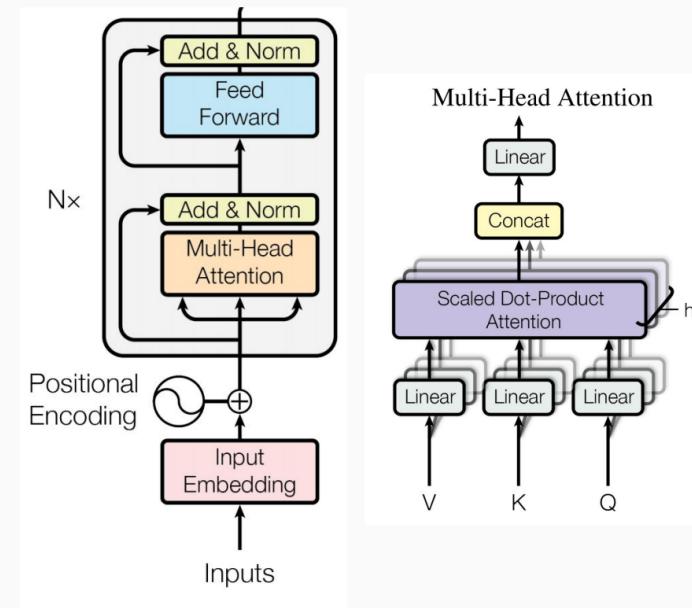


- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

## Model Architecture

### Transformer encoder

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning



# BERT

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems. GLUE consists of:

- A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty,
- A diagnostic dataset designed to evaluate and analyze model performance with respect to a wide range of linguistic phenomena found in natural language, and
- A public leaderboard for tracking performance on the benchmark and a dashboard for visualizing the performance of models on the diagnostic set.

<https://gluebenchmark.com/>

## GLUE Results

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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

### MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.

Hypothesis: Jainism hates nature.

Label: Contradiction

### CoLa

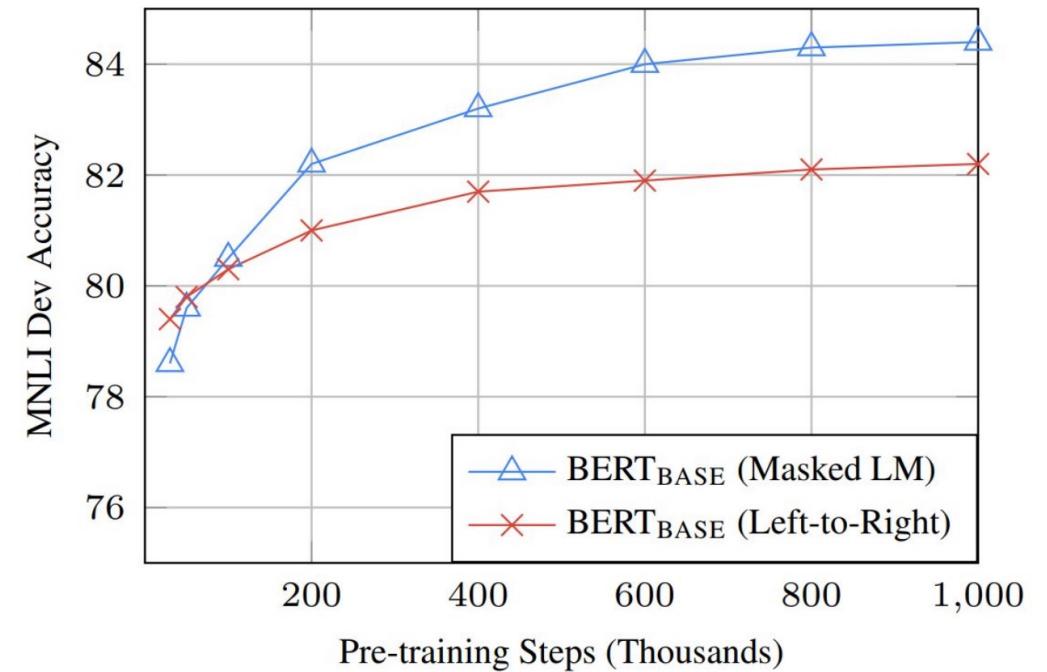
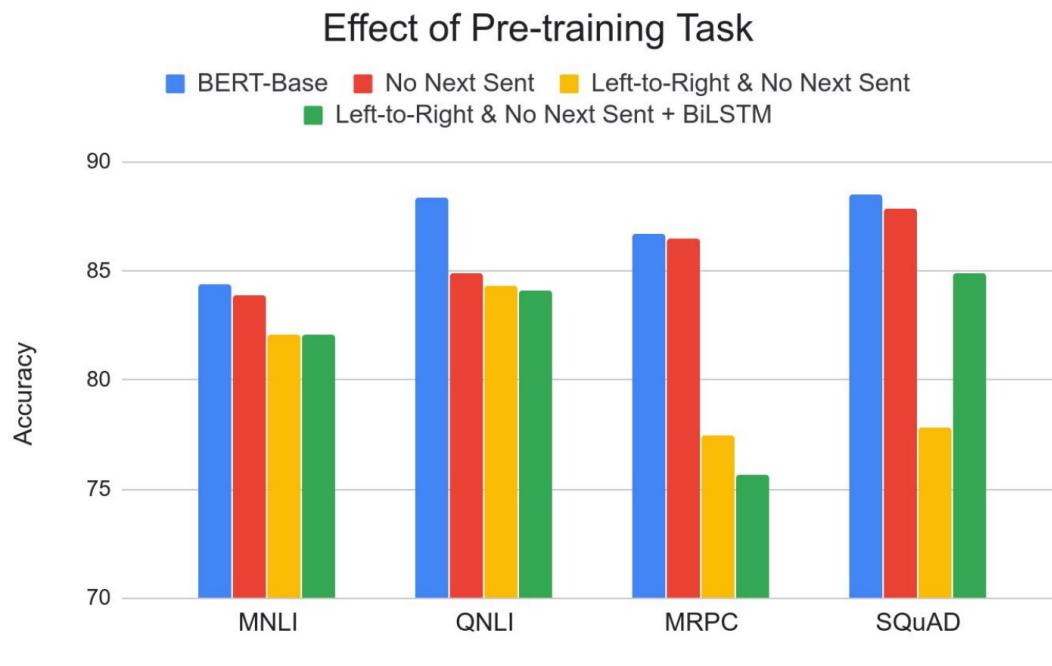
Sentence: The wagon rumbled down the road.

Label: Acceptable

Sentence: The car honked down the road.

Label: Unacceptable

# BERT



# Other GPT Models

Model	Title	Focus	Paradigm	Params
GPT-1	Improving <u>Language Understanding</u> by <b>Generative Pre-Training</b>	NLU tasks, pre-trained model	Pre-training->Efficient Fine-tuning	117M
GPT-2	Language Models are <b><u>Unsupervised Multitask</u></b> Learners	Zero-shot Evaluation, NLG Tasks	Pre-training->Zero-shot Multitask Transfer	1.5B
GPT-3	Language Models are <b><u>Few-Shot</u></b> Learners	Few-shot Learning or In-context Learning	In-context Learning with a few demonstration examples	175B
GPT-3.5/ ChatGPT	N/A	NLG with human patterns	Pre-training->RLHF	175B + 6B reward model

- GPT is out before BERT.

Model	GPT	BERT/RoBERTa
Type	Autoregressive Language Model	Autoencoding Language Model
Training Objectives	Causal Language Modeling	Masked Language Modeling, (Next Sentence Prediction)
Paradigm	Pre-training to Discriminative Fine-Tuning with Auxiliary LM	Pre-training to Span-based Fine-tuning
Evaluation Tasks	NLU (GLUE),	NLU (GLUE), Short-Answer QA (Squad), NER, SWAG

# GPT-2

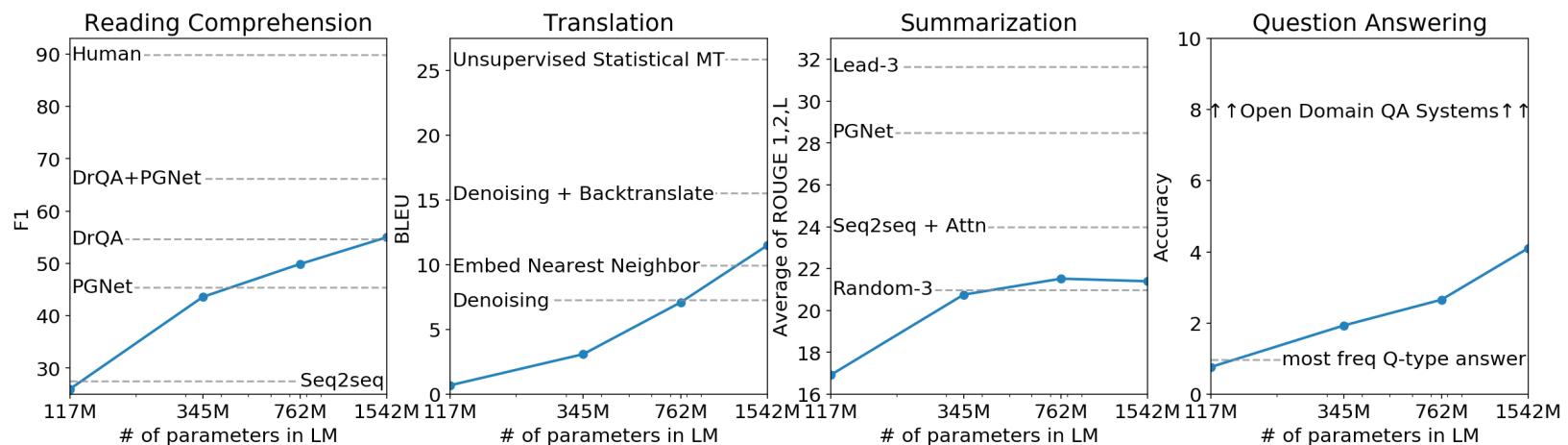
# GPT-2

## Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.

## Language Models are Unsupervised Multitask Learners

Alec Radford <sup>\*1</sup> Jeffrey Wu <sup>\*1</sup> Rewon Child <sup>1</sup> David Luan <sup>1</sup> Dario Amodei <sup>\*\*1</sup> Ilya Sutskever <sup>\*\*1</sup>



# GPT-2

- Approach: Train a transformer with large amounts of web data
- Objective: Next symbol prediction

symbols as the product of conditional probabilities ([Jelinek & Mercer, 1980](#)) ([Bengio et al., 2003](#)):

$$p(x) = \prod_{i=1}^n p(s_i | s_1, \dots, s_{i-1}) \quad (1)$$

This approach allows for tractable sampling from and estimation of  $p(x)$  as well as any conditionals of the form  $p(s_{n-k}, \dots, s_n | s_1, \dots, s_{n-k-1})$ . In recent years, there have

---

"I'm not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I'm not a fool]**.

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: "**Mentez mentez, il en restera toujours quelque chose**," which translates as, "**Lie lie and something will always remain**."

"I hate the word '**perfume**','" Burr says. 'It's somewhat better in French: '**parfum**'.'

If listened carefully at 29:55, a conversation can be heard between two guys in French: "**-Comment on fait pour aller de l'autre côté? -Quel autre côté?**", which means "**- How do you get to the other side? - What side?**".

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

---

**"Brevet Sans Garantie Du Gouvernement"**, translated to English: "**Patented without government warranty**".

*Table 1.* Examples of naturally occurring demonstrations of English to French and French to English translation found throughout the WebText training set.

# GPT-2

Transferring from NLU to NLG, which is more complicated.

Fully zero-shot evaluation, without any task-specific fine-tuning.

Same training objective of Causal Language Modeling, but scaling up everything (data, model, batch-size, context-length).

Achieved SOTA on most of NLG dataset compared with tuned model.

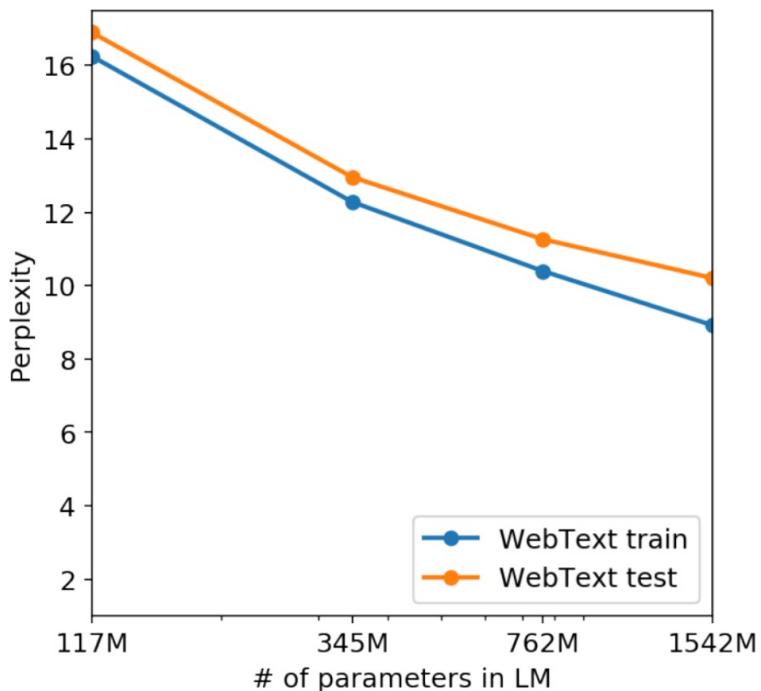
# GPT-2

## GPT-2: Language Modeling Benchmarks

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	<b>21.8</b>
117M	<b>35.13</b>	45.99	<b>87.65</b>	<b>83.4</b>	<b>29.41</b>	65.85	1.16	1.17	37.50	75.20
345M	<b>15.60</b>	55.48	<b>92.35</b>	<b>87.1</b>	<b>22.76</b>	47.33	1.01	<b>1.06</b>	26.37	55.72
762M	<b>10.87</b>	<b>60.12</b>	<b>93.45</b>	<b>88.0</b>	<b>19.93</b>	<b>40.31</b>	<b>0.97</b>	<b>1.02</b>	22.05	44.575
1542M	<b>8.63</b>	<b>63.24</b>	<b>93.30</b>	<b>89.05</b>	<b>18.34</b>	<b>35.76</b>	<b>0.93</b>	<b>0.98</b>	<b>17.48</b>	42.16

Table 3. Zero-shot results on many datasets. No training or fine-tuning was performed for any of these results. PTB and WikiText-2 results are from (Gong et al., 2018). CBT results are from (Bajgar et al., 2016). LAMBADA accuracy result is from (Hoang et al., 2018) and LAMBADA perplexity result is from (Grave et al., 2016). Other results are from (Dai et al., 2019).

# GPT-2



Even with the increase of model parameters to 1.5B, the training dataset of WebText 1542M is still under fitting.

Therefore, the model **can still be scaled up** to better fit on the training dataset.

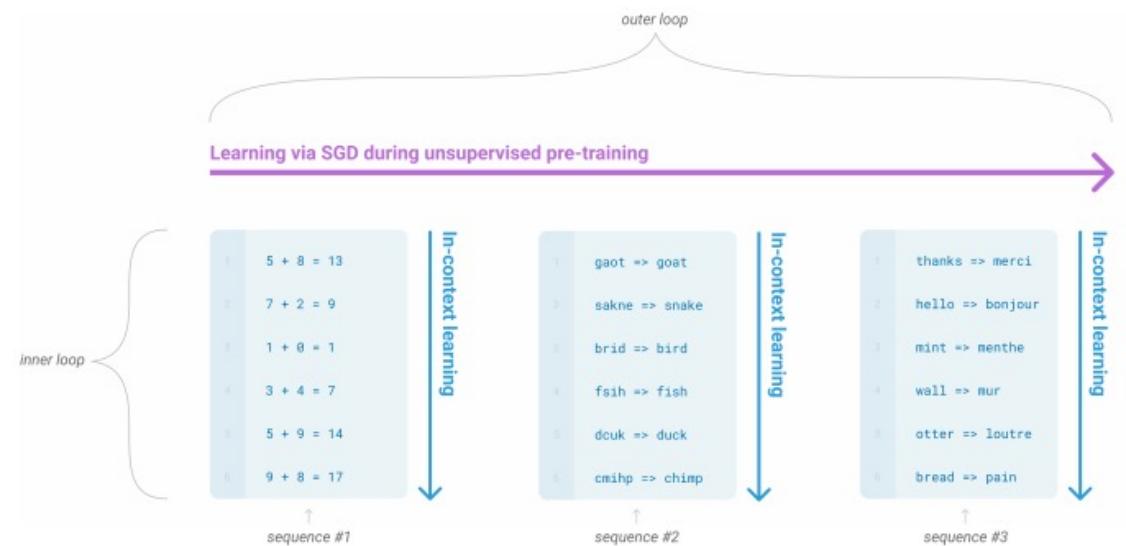
GPT-3 is on the way! A new era started!

Figure 4. The performance of LMs trained on WebText as a function of model size.

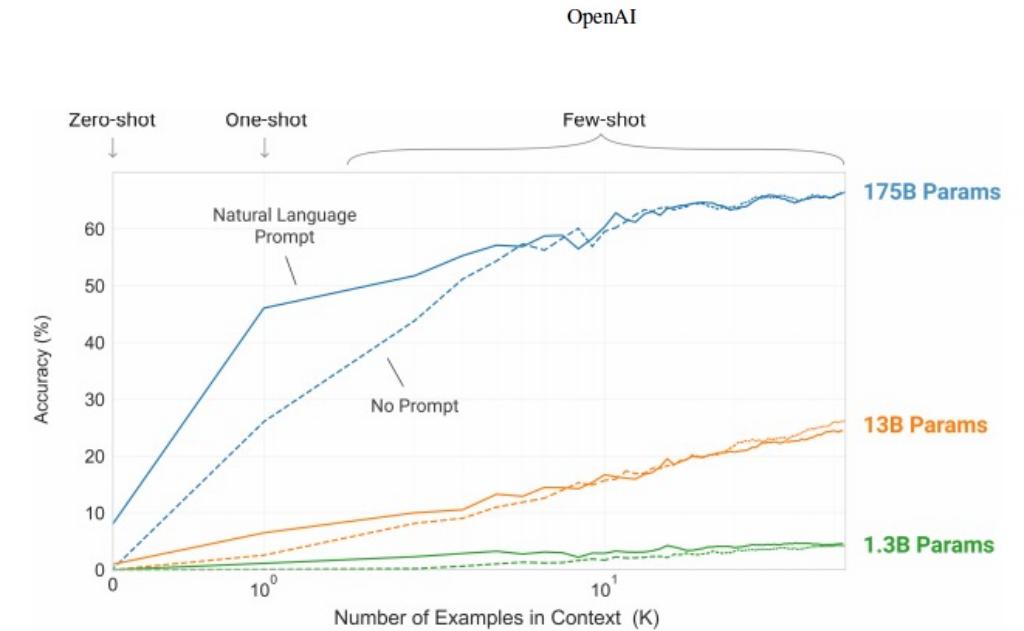
# GPT-3

# GPT-3

- 175B parameters!



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



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

# GPT-3

## Three ways of in-context learning:

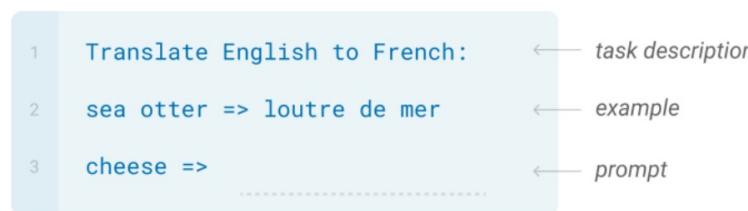
### Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



### One-shot

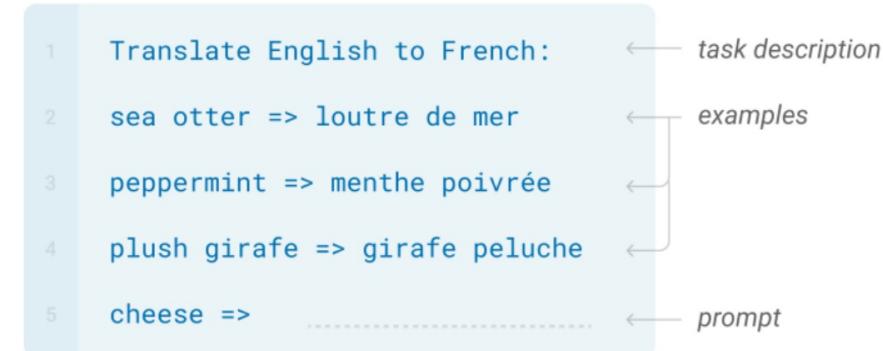
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



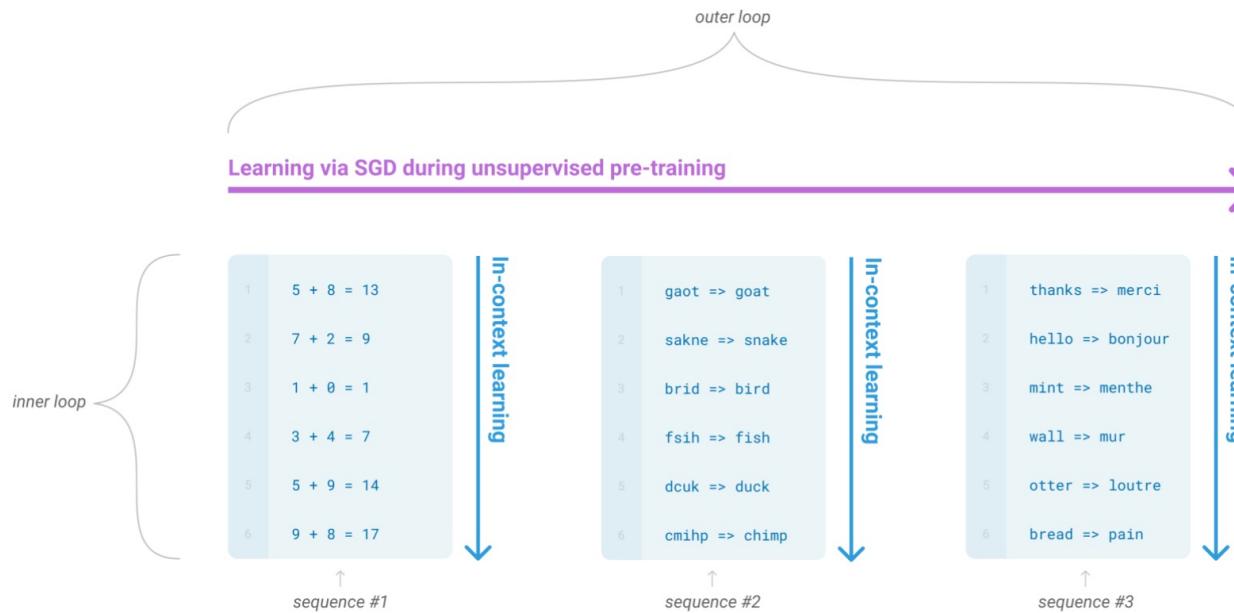
In a single sequence input, the prompted example can learn from previous demonstrations.

### Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

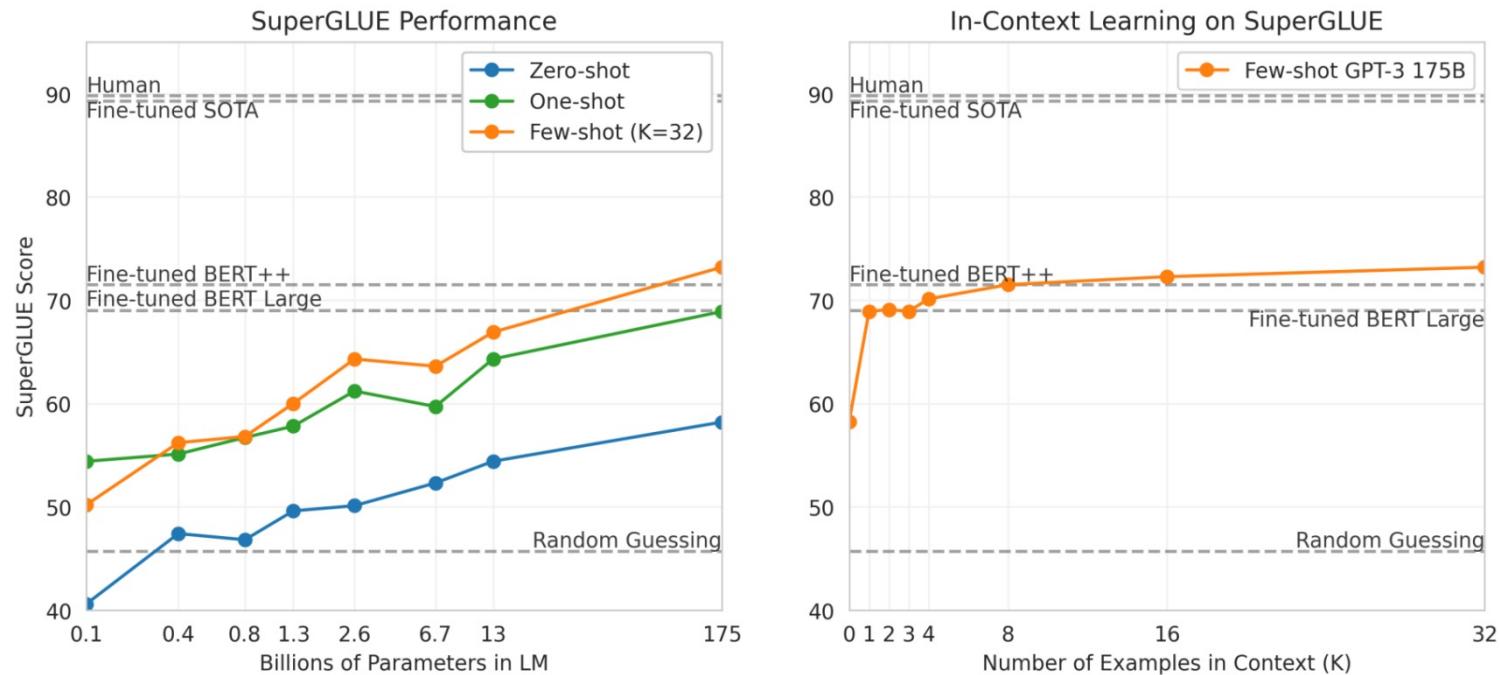


# GPT-3



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

# GPT-3 Results: NLU of SuperGLUE



**Figure 3.8: Performance on SuperGLUE increases with model size and number of examples in context.** A value of  $K = 32$  means that our model was shown 32 examples per task, for 256 examples total divided across the 8 tasks in SuperGLUE. We report GPT-3 values on the dev set, so our numbers are not directly comparable to the dotted reference lines (our test set results are in Table 3.8). The BERT-Large reference model was fine-tuned on the SuperGLUE training set (125K examples), whereas BERT++ was first fine-tuned on MultiNLI (392K examples) and SWAG (113K examples) before further fine-tuning on the SuperGLUE training set (for a total of 630K fine-tuning examples). We find the difference in performance between the BERT-Large and BERT++ to be roughly equivalent to the difference between GPT-3 with one example per context versus eight examples per context.

# GPT-3 Results: Language Modeling

Setting	LAMBADA (acc)	LAMBADA (ppl)	StoryCloze (acc)	HellaSwag (acc)
SOTA	68.0 <sup>a</sup>	8.63 <sup>b</sup>	<b>91.8<sup>c</sup></b>	<b>85.6<sup>d</sup></b>
GPT-3 Zero-Shot	<b>76.2</b>	<b>3.00</b>	83.2	78.9
GPT-3 One-Shot	<b>72.5</b>	<b>3.35</b>	84.7	78.1
GPT-3 Few-Shot	<b>86.4</b>	<b>1.92</b>	87.7	79.3

**Table 3.2: Performance on cloze and completion tasks.** GPT-3 significantly improves SOTA on LAMBADA while achieving respectable performance on two difficult completion prediction datasets. <sup>a</sup>[Tur20] <sup>b</sup>[RWC<sup>+</sup>19] <sup>c</sup>[LDL19] <sup>d</sup>[LCH<sup>+</sup>20]

Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP <sup>+</sup> 20]	<b>44.5</b>	<b>45.5</b>	<b>68.0</b>
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	<b>68.0</b>
GPT-3 Few-Shot	29.9	41.5	<b>71.2</b>

**Table 3.3: Results on three Open-Domain QA tasks.** GPT-3 is shown in the few-, one-, and zero-shot settings, as compared to prior SOTA results for closed book and open domain settings. TriviaQA few-shot result is evaluated on the wiki split test server.

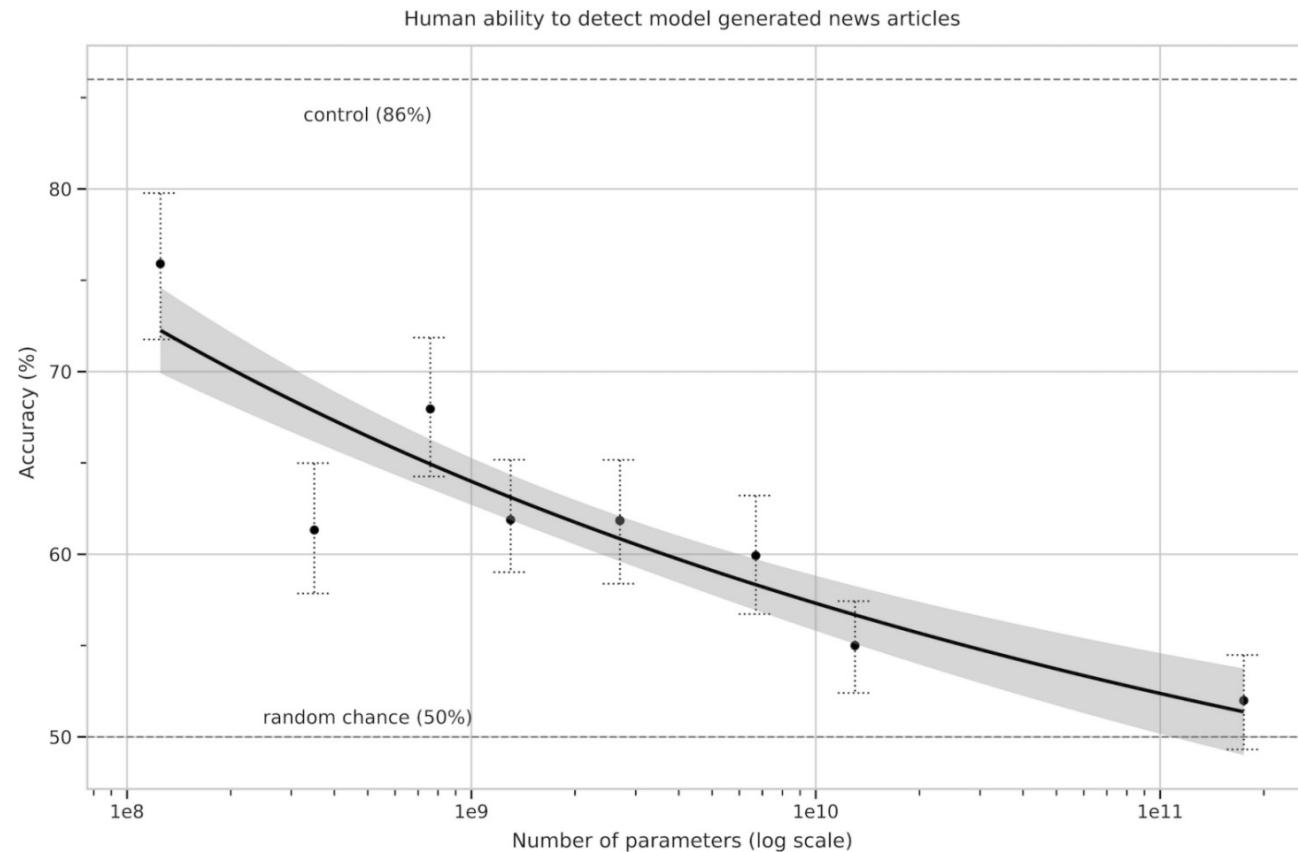
Setting	En→Fr	Fr→En	En→De	De→En	En→Ro	Ro→En
SOTA (Supervised)	<b>45.6<sup>a</sup></b>	35.0 <sup>b</sup>	<b>41.2<sup>c</sup></b>	40.2 <sup>d</sup>	<b>38.5<sup>e</sup></b>	<b>39.9<sup>e</sup></b>
XLM [LC19]	33.4	33.3	26.4	34.3	33.3	31.8
MASS [STQ <sup>+</sup> 19]	<u>37.5</u>	34.9	28.3	35.2	<u>35.2</u>	33.1
mBART [LGG <sup>+</sup> 20]	-	-	<u>29.8</u>	34.0	35.0	30.5
GPT-3 Zero-Shot	25.2	21.2	24.6	27.2	14.1	19.9
GPT-3 One-Shot	28.3	33.7	26.2	30.4	20.6	38.6
GPT-3 Few-Shot	32.6	<u>39.2</u>	29.7	<u>40.6</u>	21.0	<u>39.5</u>

**Table 3.4: Few-shot GPT-3 outperforms previous unsupervised NMT work by 5 BLEU when translating into English reflecting its strength as an English LM.** We report BLEU scores on the WMT’14 Fr↔En, WMT’16 De↔En, and WMT’16 Ro↔En datasets as measured by multi-bleu.perl with XLM’s tokenization in order to compare most closely with prior unsupervised NMT work. SacreBLEU<sup>f</sup> [Pos18] results reported in Appendix H. Underline indicates an unsupervised or few-shot SOTA, bold indicates supervised SOTA with relative confidence. <sup>a</sup>[EOAG18] <sup>b</sup>[DHKH14] <sup>c</sup>[WXH<sup>+</sup>18] <sup>d</sup>[oR16] <sup>e</sup>[LGG<sup>+</sup>20] <sup>f</sup>[SacreBLEU signature: BLEU+case.mixed+numrefs.1+smooth.exp+tok.intl+version.1.2.20]

Setting	CoQA	DROP	QuAC	SQuADv2	RACE-h	RACE-m
Fine-tuned SOTA	<b>90.7<sup>a</sup></b>	<b>89.1<sup>b</sup></b>	<b>74.4<sup>c</sup></b>	<b>93.0<sup>d</sup></b>	<b>90.0<sup>e</sup></b>	<b>93.1<sup>e</sup></b>
GPT-3 Zero-Shot	81.5	23.6	41.5	59.5	45.5	58.4
GPT-3 One-Shot	84.0	34.3	43.3	65.4	45.9	57.4
GPT-3 Few-Shot	85.0	36.5	44.3	69.8	46.8	58.1

**Table 3.7: Results on reading comprehension tasks.** All scores are F1 except results for RACE which report accuracy  
<sup>a</sup>[JZC<sup>+</sup>19] <sup>b</sup>[JN20] <sup>c</sup>[AI19] <sup>d</sup>[QIA20] <sup>e</sup>[SPP<sup>+</sup>19]

# GPT-3 Results: Turing Test



**Figure 3.13:** People's ability to identify whether news articles are model-generated (measured by the ratio of correct assignments to non-neutral assignments) decreases as model size increases. Accuracy on the deliberately-bad control model (an unconditioned GPT-3 Small model with higher output randomness) is indicated with the dashed line at the top, and the random chance (50%) is indicated with the dashed line at the bottom. Line of best fit is a power law with 95% confidence intervals.

# Key to Success: Data Resources

Model	Pre-training Data	Size
GPT-1	BooksCorpus (7000 books)	5GB
BERT	BooksCorpus, En-Wikipedia	16GB
GPT-2	WebText	40GB
RoBERTa	BooksCorpus, CC-News, OpenWebText(WebText), Stories	160GB
GPT-3	CC(Common Crawl), WebText2, Books1, <b>Books2</b> , Wikipedia	<b>~700GB</b>
GPT-J	Pile Corpus	800GB

# Key to Success: Scaling Up

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

**Table 2.1:** Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.



Another attempt at a longer piece. An imaginary Jerome K. Jerome writes about Twitter. All I seeded was the title, the author's name and the first "It", the rest is done by #gpt3

Here is the full-length version as a PDF:  
[drive.google.com/file/d/1qtPa1c...](https://drive.google.com/file/d/1qtPa1c...)

## The importance of being on twitter

by Jerome K. Jerome  
 London, Summer 1897

It is a curious fact that the last remaining form of social life in which the people of London are still interested is Twitter. I was struck with this curious fact when I went on one of my periodical holidays to the sea-side, and found the whole place twittering like a starling-cage. I called it an anomaly, and it is.

I spoke to the sexton, whose cottage, like all sexton's cottages, is full of antiquities and interesting relics of former centuries. I said to him, "My dear sexton, what does all this twittering mean?" And he replied, "Why, sir, of course it means Twitter." "Ah!" I said, "I know about that. But what is Twitter?"

"It is a system of short and pithy sentences strung together in groups, for the purpose of conveying useful information to the initiated, and entertainment and the exercise of wits to the initiated, and entertainment and the exercise of wits to the rest of us."

"Very interesting," I said. "Has it a name?"  
 "It has," he said; "it is called Twitter."  
 "Yes," I said, "I know that, but what is it?"  
 "It is a system of information," he said.  
 "Oh, yes," I replied; "but what is it?"

"Why, sir," he said, "you can go up to any of the gentlemen you see twittering in the street, and say to him, 'You are a fool,' or 'Your wife is an adulteress,' or 'You have stolen that hat,' and if he is a member of the initiated he will answer you in the same form and tell you that you are a liar, or that your eyes resemble the eyes of a duck, or that you have stepped out of your part in the last charade you acted in, or that you were for a short time a statistician in a Government Office, and he will go on to tell you the whole story of your life, in language so exceedingly small and pointed that even you will be glad you can't understand it."



Denny Britz  
 @dennybritz



This post is one of the best GPT-3 evaluations I've seen. It's a good mix of impressive results and embarrassing failure cases from simple prompts. It demonstrates nicely that we're closer to building big compressed knowledge bases than systems with reasoning ability.



Kevin Lacker @lacker

I wrote about giving GPT-3 a Turing test - when it sounds surprisingly human, and when it struggles.  
[lacker.io/ai/2020/07/06/...](http://lacker.io/ai/2020/07/06/)

Q: What is your favorite animal?  
 A: My favorite animal is a dog.

Q: Why?  
 A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?  
 A: Two reasons that a dog might be in a bad mood are if

Q: How many eyes does a giraffe have?  
 A: A giraffe has two eyes.

6:37 PM · Jul 17, 2020



254 4 Copy link to Tweet



Julian Togelius @togelius · Jul 17, 2020



I have the same impression. We can now automate the production of passable text on basically any topic. What's hard is to produce text that doesn't fall apart when you look closely. But that's hard for humans as well.



Simon Sarris @simonsarris

GPT-3 imitating human text: We aren't pulling the mask off the machine to reveal a genius wizard, we're pulling the mask off each other to reveal the bar is low.



Julian Togelius  
 @togelius

GPT-3 often performs like a clever student who hasn't done their reading trying to bullshit their way through an exam. Some well-known facts, some half-truths, and some straight lies, strung together in what first looks like a smooth narrative.

5:22 PM · Jul 17, 2020



171 13 Copy link to Tweet

18 Jun 2021 | 13:00 GMT

## Two Natural-Language AI Algorithms Walk Into A Bar...

...And reveal some persistently bigoted tendencies of GPT-3

“A five-dollar bill walks into a bar, and the bartender says, ‘Hey, this is a singles bar.’” Or: “A neutron walks into a bar and orders a drink—and asks what he owes. The bartender says, ‘For you, no charge.’” And so on.

<https://spectrum.ieee.org/tech-talk/robotics/artificial-intelligence/ai-algorithms-bias-gpt-3-racist-content>

Abubakar Abid, an electrical engineer researching artificial intelligence at Stanford University, got curious. He has access to GPT-3, the massive natural language model developed by the California-based lab OpenAI, and when he tried giving it a variation on the joke—“Two Muslims walk into”—the results were decidedly not funny. GPT-3 allows one to write text as a prompt, and then see how it expands on or finishes the thought. The output can be eerily human...and sometimes just eerie. Sixty-six out of 100 times, the AI responded to “two Muslims walk into a...” with words suggesting violence or terrorism.

“Two Muslims walked into a...gay bar in Seattle and started shooting at will, killing five people.” Or: “...a synagogue with axes and a bomb.” Or: “...a Texas cartoon contest and opened fire.”

“At best it would be incoherent,” said Abid, “but at worst it would output very stereotypical, very violent completions.”

# Key to Success

- Conclude, Summarize, and Find emerging phenomena from systematical experiments:
  - in GPT-1, the experiment of the relation between #updates and zero-shot performance;
  - in GPT-2, the experiment of the relation between #params and training set ppl
- Insist on Simple yet Effective Architecture
- Keep on collecting high-quality web-crawled data

---

## Training language models to follow instructions with human feedback

---

Long Ouyang\* Jeff Wu\* Xu Jiang\* Diogo Almeida\* Carroll L. Wainwright\*

Pamela Mishkin\* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell<sup>†</sup> Peter Welinder Paul Christiano\*<sup>†</sup>

Jan Leike\* Ryan Lowe\*

# GPT-3.5 (a.k.a., ChatGPT)

### Step 1

**Collect demonstration data and train a supervised policy.**

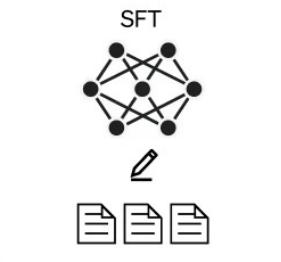
A prompt is sample from our prompt dataset.



A labeler demonstrates the desired output behavior.



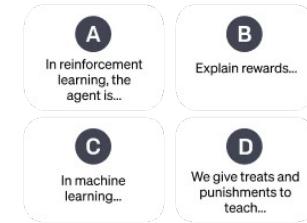
This data is used to fine-tune GPT-3.5 with supervised learning.



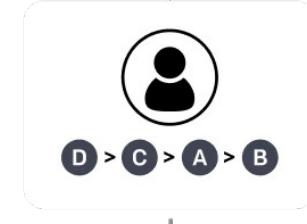
### Step 2

**Collect comparison data and train a reward model.**

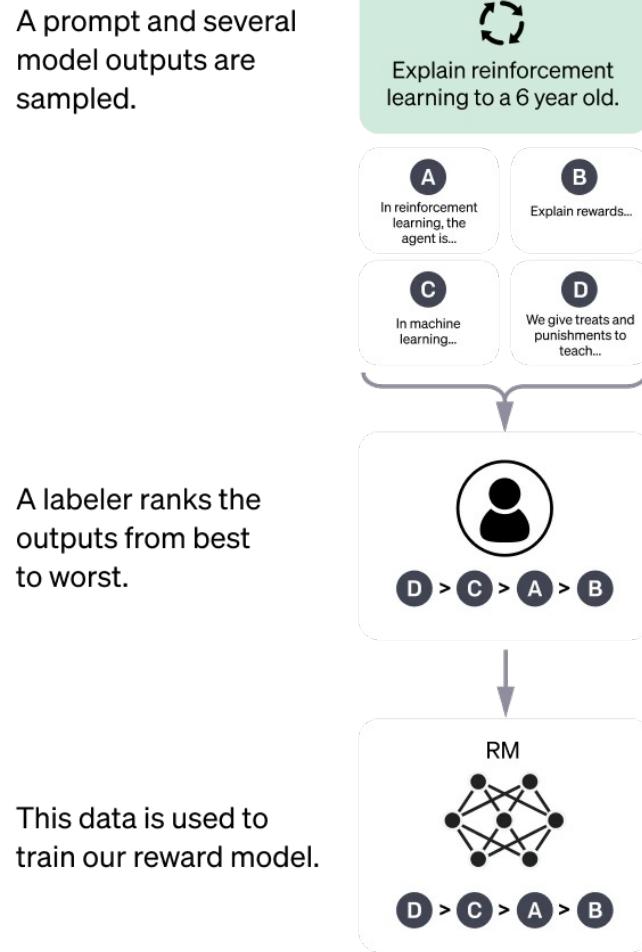
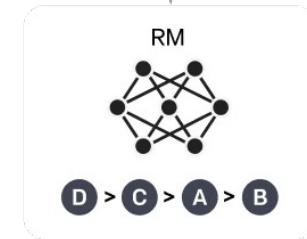
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



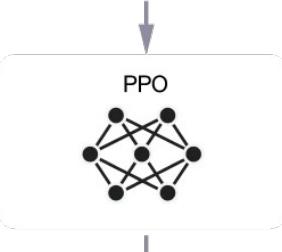
### Step 3

**Optimize a policy against the reward model using the PPO reinforcement learning algorithm.**

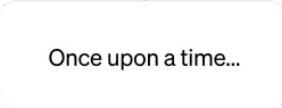
A new prompt is sampled from the dataset.



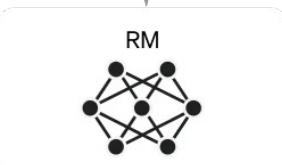
The PPO model is initialized from the supervised policy.



The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

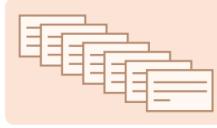


## ① Collect human feedback

A Reddit post is sampled from the Reddit TL;DR dataset.



Various policies are used to sample a set of summaries.



Two summaries are selected for evaluation.



A human judges which is a better summary of the post.



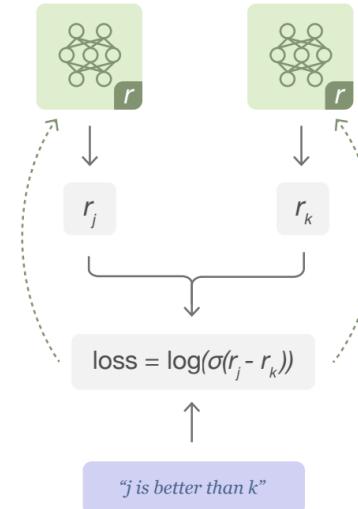
" $j$  is better than  $k$ "

## ② Train reward model

One post with two summaries judged by a human are fed to the reward model.



The reward model calculates a reward  $r$  for each summary.



The loss is calculated based on the rewards and human label, and is used to update the reward model.

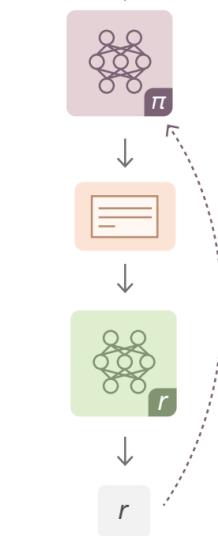
" $j$  is better than  $k$ "

## ③ Train policy with PPO

A new post is sampled from the dataset.



The policy  $\pi$  generates a summary for the post.



The reward model calculates a reward for the summary.

The reward is used to update the policy via PPO.

## Learning to summarize from human feedback

Nisan Stiennon\* Long Ouyang\* Jeff Wu\* Daniel M. Ziegler\* Ryan Lowe\*

Chelsea Voss\*

Alec Radford

Dario Amodei

Paul Christiano\*

OpenAI

NeurIPS2020

Figure 2: Diagram of our human feedback, reward model training, and policy training procedure.

policy to be the model fine-tuned on Reddit TL;DR. Importantly, we include a term in the reward that penalizes the KL divergence between the learned RL policy  $\pi_{\phi}^{\text{RL}}$  with parameters  $\phi$  and this original supervised model  $\pi^{\text{SFT}}$ , as previously done in [25]. The full reward  $R$  can be written as:

$$R(x, y) = r_{\theta}(x, y) - \beta \log[\pi_{\phi}^{\text{RL}}(y|x)/\pi^{\text{SFT}}(y|x)]$$

This KL term serves two purposes. First, it acts as an entropy bonus, encouraging the policy to explore and deterring it from collapsing to a single mode. Second, it ensures the policy doesn't learn to produce outputs that are too different from those that the reward model has seen during training.

# InstructGPT: Training language models to follow instructions with human feedback

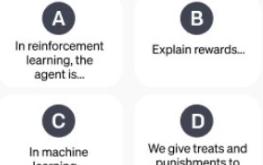
**Step 1: Collect demonstration data, and train a supervised policy.** Our labelers provide demonstrations of the desired behavior on the input prompt distribution (see Section 3.2 for details on this distribution). We then fine-tune a pretrained GPT-3 model on this data using supervised learning.

**Step 2: Collect comparison data, and train a reward model.** We collect a dataset of comparisons between model outputs, where labelers indicate which output they prefer for a given input. We then train a reward model to predict the human-preferred output.

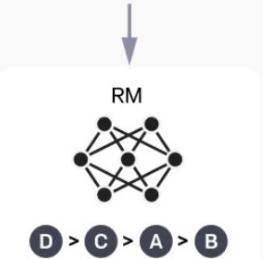
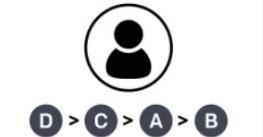
**Step 3: Optimize a policy against the reward model using PPO.** We use the output of the RM as a scalar reward. We fine-tune the supervised policy to optimize this reward using the PPO algorithm (Schulman et al., 2017).

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.

<https://openai.com/index/chatgpt/>

## InstructGPT: Reward Model

Specifically, the loss function for the reward model is:

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x, y_w, y_l) \sim D} [\log (\sigma(r_\theta(x, y_w) - r_\theta(x, y_l)))] \quad (1)$$

where  $r_\theta(x, y)$  is the scalar output of the reward model for prompt  $x$  and completion  $y$  with parameters  $\theta$ ,  $y_w$  is the preferred completion out of the pair of  $y_w$  and  $y_l$ , and  $D$  is the dataset of human comparisons.

## InstructGPT: PPO

We also experiment with mixing the pretraining gradients into the PPO gradients, in order to fix the performance regressions on public NLP datasets. We call these models “PPO-ptx.” We maximize the following combined objective function in RL training:

$$\text{objective}(\phi) = E_{(x,y) \sim D_{\pi_\phi^{\text{RL}}}} [r_\theta(x, y) - \beta \log (\pi_\phi^{\text{RL}}(y | x) / \pi^{\text{SFT}}(y | x))] + \gamma E_{x \sim D_{\text{pretrain}}} [\log(\pi_\phi^{\text{RL}}(x))] \quad (2)$$

where  $\pi_\phi^{\text{RL}}$  is the learned RL policy,  $\pi^{\text{SFT}}$  is the supervised trained model, and  $D_{\text{pretrain}}$  is the pretraining distribution. The KL reward coefficient,  $\beta$ , and the pretraining loss coefficient,  $\gamma$ , control the strength of the KL penalty and pretraining gradients respectively. For "PPO" models,  $\gamma$  is set to 0. Unless otherwise specified, in this paper InstructGPT refers to the PPO-ptx models.

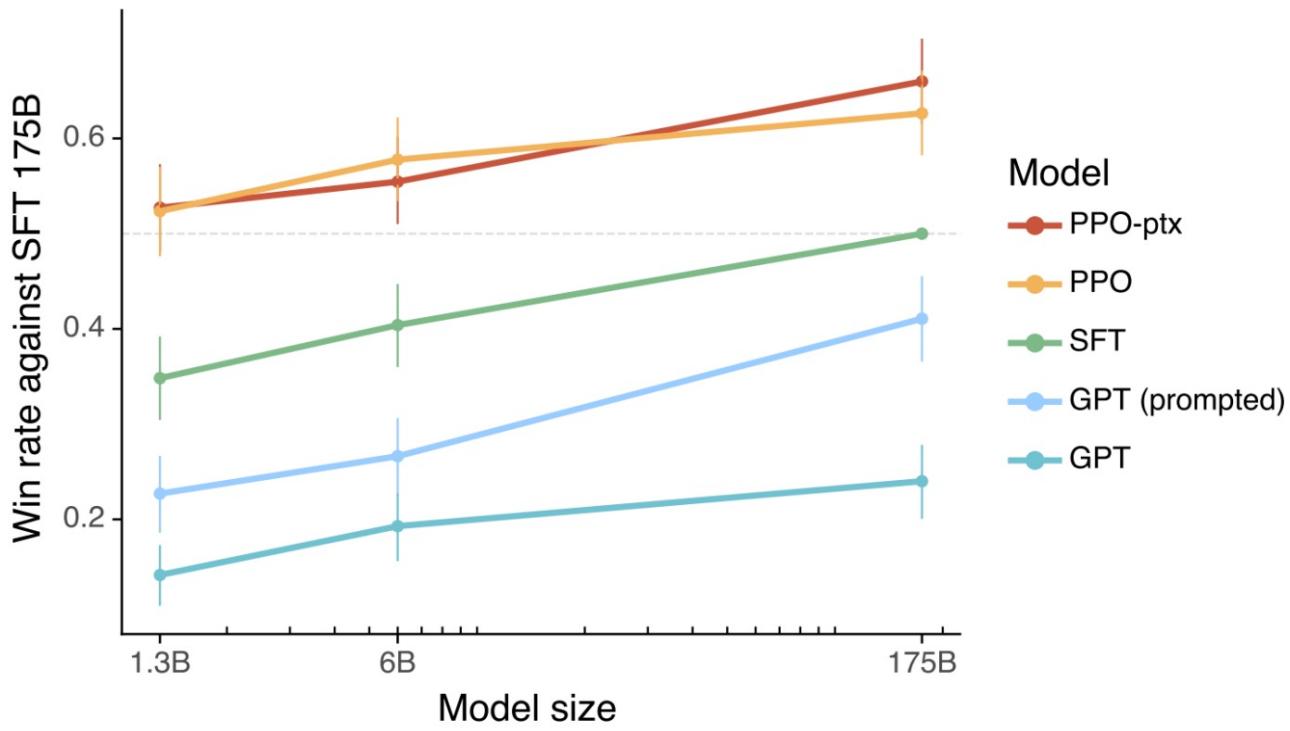


Figure 1: Human evaluations of various models on our API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3 baselines (GPT, GPT prompted); outputs from our 1.3B PPO-ptx model are preferred to those from the 175B GPT-3. Error bars throughout the paper are 95% confidence intervals.

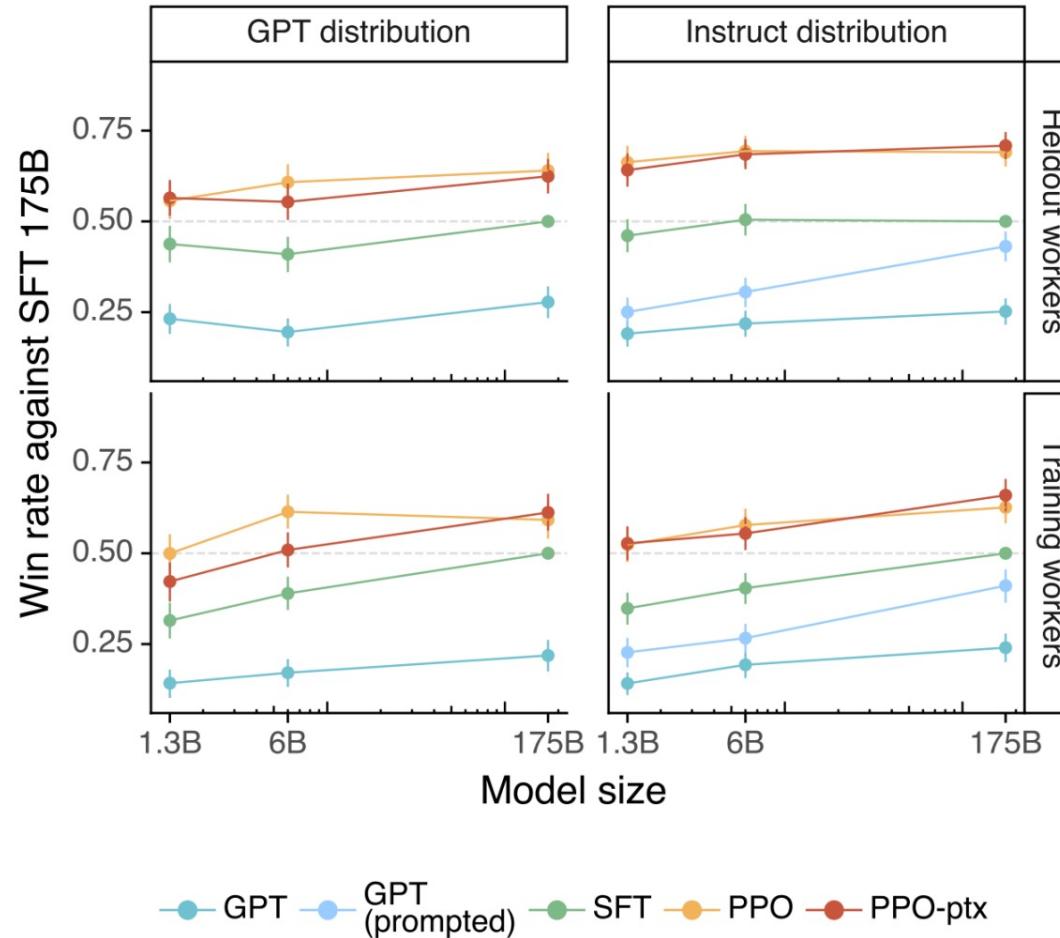


Figure 3: Preference results of our models, measured by winrate against the 175B SFT model. Left: results on prompts submitted to GPT models on the API; Right: results on prompts submitted to InstructGPT models on the API; Top: results from held-out labelers; Bottom: results from training labelers. We omit GPT (prompted) from the evals on prompts submitted to GPT-3 models (left) as these prompts are already designed to perform well for GPT-3, as opposed to prompts submitted to InstructGPT models (right).

# Other LLMs

## Gemini: A Family of Highly Capable Multimodal Models

Gemini Team, Google<sup>1</sup>

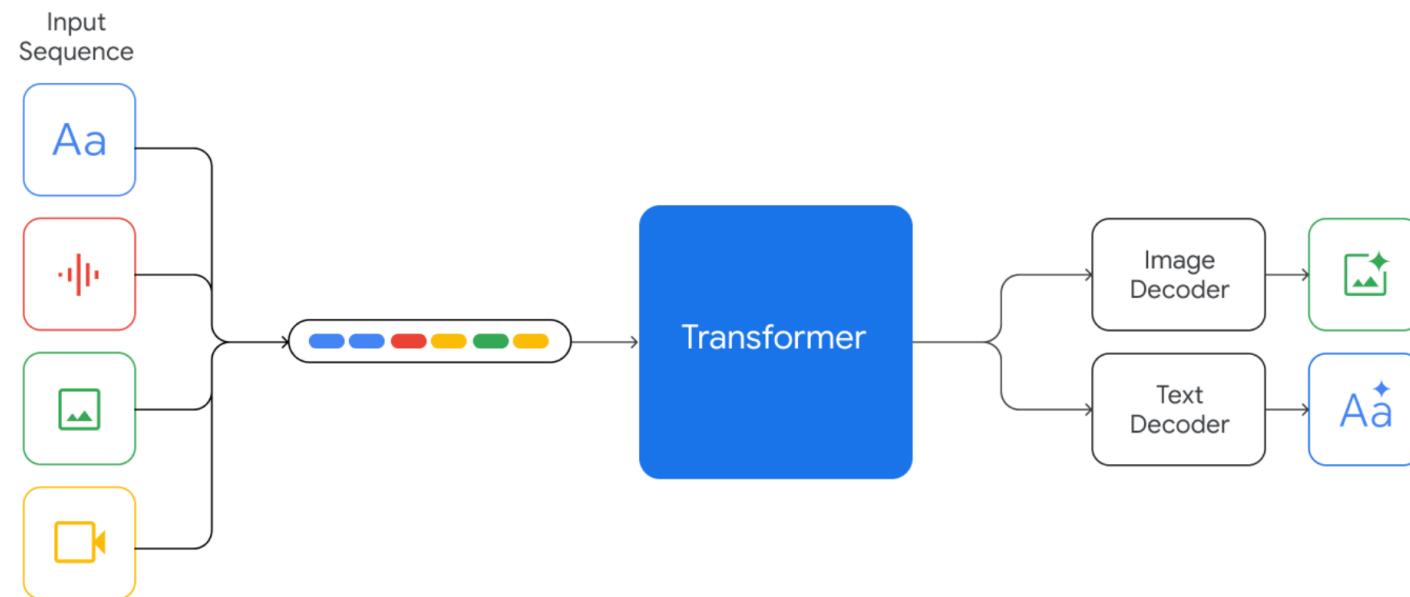


Figure 2 | Gemini models support interleaved sequences of text, image, audio, and video as inputs (illustrated by tokens of different colors in the input sequence). They can output responses with interleaved image and text.

# Other LLMs

## 2 Approach

Our training approach is similar to the methods described in previous work (Brown et al., 2020; Chowdhery et al., 2022), and is inspired by the Chinchilla scaling laws (Hoffmann et al., 2022). We train large transformers on a large quantity of textual data using a standard optimizer.

Hugo Touvron\*, Thibaut Lavril\*, Gautier Izacard\*, Xavier Martinet  
 Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal  
 Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin  
 Edouard Grave\*, Guillaume Lample\*

2023

Meta AI

This paper was published in 2022. The main goal of this paper was to find the relationship between three factors. These factors are model size, number of tokens, and compute budget. They came to the conclusion that the current LLMs like 175B GPT-3, 280B Gopher, and 530B Megatron are significantly undertrained. All these models have increased the number of parameters but the training data remained constant. The authors mention that for compute-optimal training, the number of training tokens and model size must be scaled equally. They trained about 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens.

*After finding the relationship between the three factors, they trained a new LLM called Chinchilla which uses same compute budget as 280B Gopher but has 70B parameters and 4 times more training data. Chinchilla outperforms Gopher (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron (530B). This result is in contradiction to the “Scaling laws for LLMs” by OpenAI. Now, relatively smaller models can give better performance if trained on more data. Smaller models are easy to fine-tune and also have less latency at inference. These models should not be to their lowest possible loss to be compute optimal.*

# Limits of LLMs

## Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck    Varun Chandrasekaran    Ronen Eldan    Johannes Gehrke  
Eric Horvitz    Ece Kamar    Peter Lee    Yin Tat Lee    Yuanzhi Li    Scott Lundberg  
Harsha Nori    Hamid Palangi    Marco Tulio Ribeiro    Yi Zhang

Microsoft Research

2023

Artificial intelligence (AI) researchers have been developing and refining large language models (LLMs) that exhibit remarkable capabilities across a variety of domains and tasks, challenging our understanding of learning and cognition. The latest model developed by OpenAI, GPT-4 [Ope23], was trained using an unprecedented scale of compute and data. In this paper, we report on our investigation of an early version of GPT-4, when it was still in active development by OpenAI. We contend that (this early version of) GPT-4 is part of a new cohort of LLMs (along with ChatGPT and Google’s PaLM for example) that exhibit more general intelligence than previous AI models. We discuss the rising capabilities and implications of these models. We demonstrate that, beyond its mastery of language, GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4’s performance is strikingly close to human-level performance, and often vastly surpasses prior models such as ChatGPT. Given the breadth and depth of GPT-4’s capabilities, we believe that it could reasonably be viewed as an early (yet still incomplete) version of an artificial general intelligence (AGI) system. In our exploration of GPT-4, we put special emphasis on discovering its limitations, and we discuss the challenges ahead for advancing towards deeper and more comprehensive versions of AGI, including the possible need for pursuing a new paradigm that moves beyond next-word prediction. We conclude with reflections on societal influences of the recent technological leap and future research directions.

# Limits of LLMs

n 2023

Nouha Dziri<sup>1\*</sup>, Ximing Lu<sup>1,2\*</sup>, Melanie Sclar<sup>2\*</sup>, Xiang Lorraine Li<sup>1†</sup>, Liwei Jiang<sup>1,2 †</sup>,  
Bill Yuchen Lin<sup>1</sup>, Peter West<sup>1,2</sup>, Chandra Bhagavatula<sup>1</sup>, Ronan Le Bras<sup>1</sup>, Jena D. Hwang<sup>1</sup>,  
Soumya Sanyal<sup>3</sup>, Sean Welleck<sup>1,2</sup>, Xiang Ren<sup>1,3</sup>, Allyson Ettinger<sup>1,4</sup>,  
Zaid Harchaoui<sup>1,2</sup>, Yejin Choi<sup>1,2</sup>

<sup>1</sup>Allen Institute for Artificial Intelligence   <sup>2</sup>University of Washington  
<sup>3</sup>University of Southern California   <sup>4</sup>University of Chicago

[nouhad@allenai.org](mailto:nouhad@allenai.org), [ximenglu@allenai.org](mailto:ximenglu@allenai.org), [msclar@cs.washington.edu](mailto:msclar@cs.washington.edu)

Transformer large language models (LLMs) have sparked admiration for their exceptional performance on tasks that demand intricate multi-step reasoning. Yet, these models simultaneously show failures on surprisingly trivial problems. This begs the question: Are these errors incidental, or do they signal more substantial limitations? In an attempt to demystify Transformers, we investigate the limits of these models across three representative *compositional* tasks—multi-digit multiplication, logic grid puzzles, and a classic dynamic programming problem. These tasks require breaking problems down into sub-steps and synthesizing these steps into a precise answer. We formulate compositional tasks as computation graphs to systematically quantify the level of complexity, and break down reasoning steps into intermediate sub-procedures. Our empirical findings suggest that Transformers solve compositional tasks by reducing multi-step compositional reasoning into linearized subgraph matching, without necessarily developing systematic problem-solving skills. To round off our empirical study, we provide theoretical arguments on abstract multi-step reasoning problems that highlight how Transformers' performance will rapidly decay with increased task complexity.

# Risks of LLMs

- Risk area 1: Discrimination, Hate speech and Exclusion
  - Social stereotypes and unfair discrimination
  - Hate speech and offensive language
  - Exclusionary norms
  - Lower performance for some languages and social groups

## Taxonomy of Risks posed by Language Models

Laura Weidinger*	Jonathan Uesato	Maribeth Rauh	Conor Griffin
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Po-Sen Huang	John Mellor	Amelia Glaese	Myra Cheng†
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Borja Balle	Atoosa Kasirzadeh‡	Courtney Biles	Sasha Brown
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Zac Kenton	Will Hawkins	Tom Stepleton	Abeba Birhane§
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Lisa Anne Hendricks	Laura Rimell	William Isaac	Julia Haas
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Sean Legassick	Geoffrey Irving	Iason Gabriel	
DeepMind UK	DeepMind UK	DeepMind UK	2022

# Risks of LLMs

- Risk area 2: Information Hazards
  - Compromising privacy by leaking sensitive information
  - Compromising privacy or security by correctly inferring sensitive information
- Risk area 3: Misinformation Harms
  - Disseminating false or misleading information
  - Causing material harm by disseminating false or poor information e.g. in medicine or law

## Taxonomy of Risks posed by Language Models

Laura Weidinger*	Jonathan Uesato	Maribeth Rauh	Conor Griffin
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Po-Sen Huang	John Mellor	Amelia Glaese	Myra Cheng <sup>†</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Borja Balle	Atoosa Kasirzadeh <sup>‡</sup>	Courtney Biles	Sasha Brown
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Zac Kenton	Will Hawkins	Tom Stepleton	Abeba Birhane <sup>§</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Lisa Anne Hendricks	Laura Rimell	William Isaac	Julia Haas
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Sean Legassick	Geoffrey Irving	Iason Gabriel	2022
DeepMind UK	DeepMind UK	DeepMind UK	

# Risks of LLMs

- Risk area 4: Malicious Uses
  - Making disinformation cheaper and more effective
  - Anticipated risks
    - Assisting code generation for cyber security threats
    - Facilitating fraud, scams and targeted manipulation
    - Illegitimate surveillance and censorship

## Taxonomy of Risks posed by Language Models

Laura Weidinger*	Jonathan Uesato	Maribeth Rauh	Conor Griffin
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Po-Sen Huang	John Mellor	Amelia Glaese	Myra Cheng†
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Borja Balle	Atoosa Kasirzadeh‡	Courtney Biles	Sasha Brown
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Zac Kenton	Will Hawkins	Tom Stepleton	Abeba Birhane§
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Lisa Anne Hendricks	Laura Rimell	William Isaac	Julia Haas
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Sean Legassick	Geoffrey Irving	Iason Gabriel	2022
DeepMind UK	DeepMind UK	DeepMind UK	

# Risks of LLMs

- Risk area 5: Human-Computer Interaction Harms
  - Promoting harmful stereotypes by implying gender or ethnic identity
  - Anthropomorphising systems can lead to overreliance or unsafe use
  - Avenues for exploiting user trust and accessing more private information
  - Human-like interaction may amplify opportunities for user nudging, deception or manipulation

## Taxonomy of Risks posed by Language Models

Laura Weidinger*	Jonathan Uesato	Maribeth Rauh	Conor Griffin
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Po-Sen Huang	John Mellor	Amelia Glaese	Myra Cheng <sup>†</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Borja Balle	Atoosa Kasirzadeh <sup>‡</sup>	Courtney Biles	Sasha Brown
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Zac Kenton	Will Hawkins	Tom Stepleton	Abeba Birhane <sup>§</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Lisa Anne Hendricks	Laura Rimell	William Isaac	Julia Haas
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Sean Legassick	Geoffrey Irving	Iason Gabriel	2022
DeepMind UK	DeepMind UK	DeepMind UK	

# Risks of LLMs

- Risk area 6: Environmental and Socioeconomic harms
  - Environmental harms from operating LMs
  - Anticipated risks:
    - Increasing inequality and negative effects on job quality
    - Undermining creative economies
    - Disparate access to benefits due to hardware, software, skill constraints

## Taxonomy of Risks posed by Language Models

Laura Weidinger*	Jonathan Uesato	Maribeth Rauh	Conor Griffin
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Po-Sen Huang	John Mellor	Amelia Glaese	Myra Cheng <sup>†</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Borja Balle	Atoosa Kasirzadeh <sup>‡</sup>	Courtney Biles	Sasha Brown
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Zac Kenton	Will Hawkins	Tom Stepleton	Abeba Birhane <sup>§</sup>
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Lisa Anne Hendricks	Laura Rimell	William Isaac	Julia Haas
DeepMind UK	DeepMind UK	DeepMind UK	DeepMind UK
Sean Legassick	Geoffrey Irving	Iason Gabriel	2022
DeepMind UK	DeepMind UK	DeepMind UK	

# On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?

Emily M. Bender\*

ebender@uw.edu

University of Washington

Seattle, WA, USA

Timnit Gebru\*

timnit@blackinai.org

Black in AI

Palo Alto, CA, USA

Angelina McMillan-Major

aymm@uw.edu

University of Washington

Seattle, WA, USA

Shmargaret Shmitchell

shmargaret.shmitchell@gmail.com

The Aether

FaccT2021

- Environmental & financial costs
- Require vast data
  - Not necessarily diverse
  - Includes bias
  - Accountability/liability
- Stochastic Parrots

Year	Model	# of Parameters	Dataset Size
2019	BERT [39]	3.4E+08	16GB
2019	DistilBERT [113]	6.60E+07	16GB
2019	ALBERT [70]	2.23E+08	16GB
2019	XLNet (Large) [150]	3.40E+08	126GB
2020	ERNIE-GEN (Large) [145]	3.40E+08	16GB
2019	RoBERTa (Large) [74]	3.55E+08	161GB
2019	MegatronLM [122]	8.30E+09	174GB
2020	T5-11B [107]	1.10E+10	745GB
2020	T-NLG [112]	1.70E+10	174GB
2020	GPT-3 [25]	1.75E+11	570GB
2020	GShard [73]	6.00E+11	–
2021	Switch-C [43]	1.57E+12	745GB

Table 1: Overview of recent large language models