Deep Learning for Applications - II

심스리얼리티 임직원을 위한 딥러닝 교육 과정

Dr. Kyong-Ha Lee (kyongha@kisti.re.kr)

KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

Contents

Class2 : 뉴럴 네트워크에 대한 기초를 토대로 좀더 복잡한 모델 (RNN, Attention, Transformer, LLM)들을 이해합니다.

* Disclaimer: Most contents of this presentation are borrowed from lectures cs224n, cs231n from Stanford University and MJ Ma's presentation at Korea Univ. for educational purpose

A Few Quates about Machine Learning

- **Motivation :**
  - **Why do we need Sequential Modeling?**

Examples of Sequence data	Input Data	Output
Speech Recognition		*This is RNN*
Machine Translation	Hello, I am Pankaj.	Hallo, ich bin Pankaj. हैलो, मैं पंकज हूं।
Language Modeling	Recurrent neural ? based ? model	network language
Named Entity Recognition	Pankaj lives in Munich	Pankaj lives in Munich *person* *location*
Sentiment Classification	There is nothing to like in this movie.	★☆☆☆☆
Video Activity Analysis		Punching

Motivation: Need for Sequential Modeling

- **Inputs, outputs can be in different lengths in different examples**

Example:

Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE

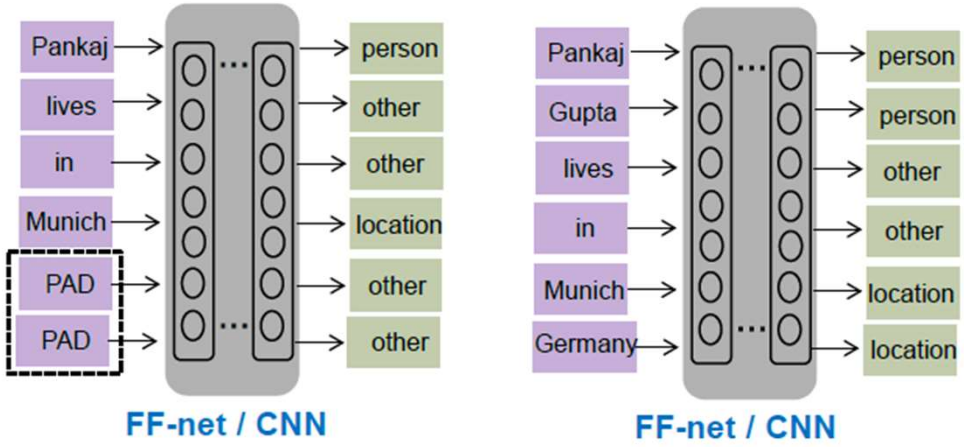
Motivation: Need for Sequential Modeling

- **Inputs, outputs can be in different lengths in different examples**

Example:

Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE



Additional word 'PAD' i.e., padding

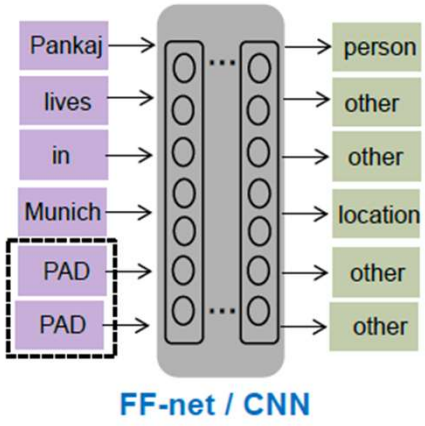
Motivation: Need for Sequential Modeling

- **Inputs, outputs can be in different lengths in different examples**

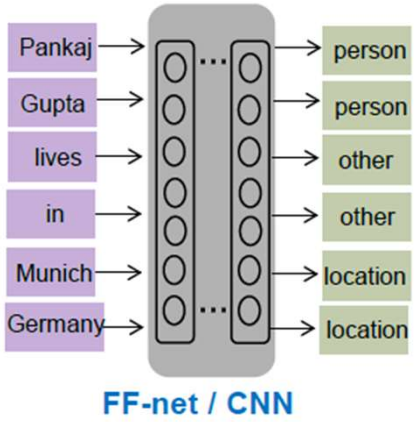
Example:

Sentence1: Pankaj lives in Munich

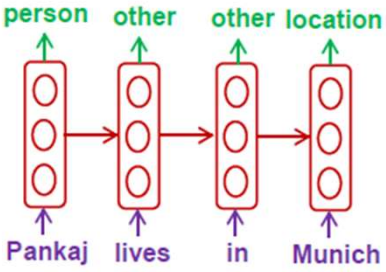
Sentence2: Pankaj Gupta lives in Munich DE



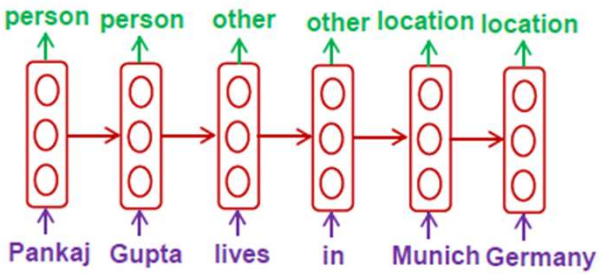
FF-net / CNN



FF-net / CNN



Models variable length sequences



Sequential model: RNN

Motivation: Need for Sequential Modeling

- **Shared features learned across different positions or time steps**

Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*

Same uni-gram statistics

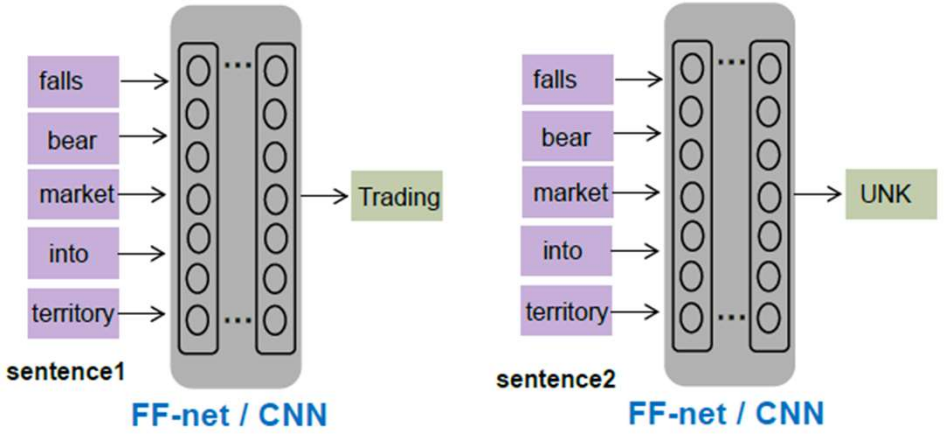
Motivation: Need for Sequential Modeling

- **Shared features learned across different positions or time steps**

Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*



No sequential or temporal modeling, i.e., order-less

Treats the two sentences the same

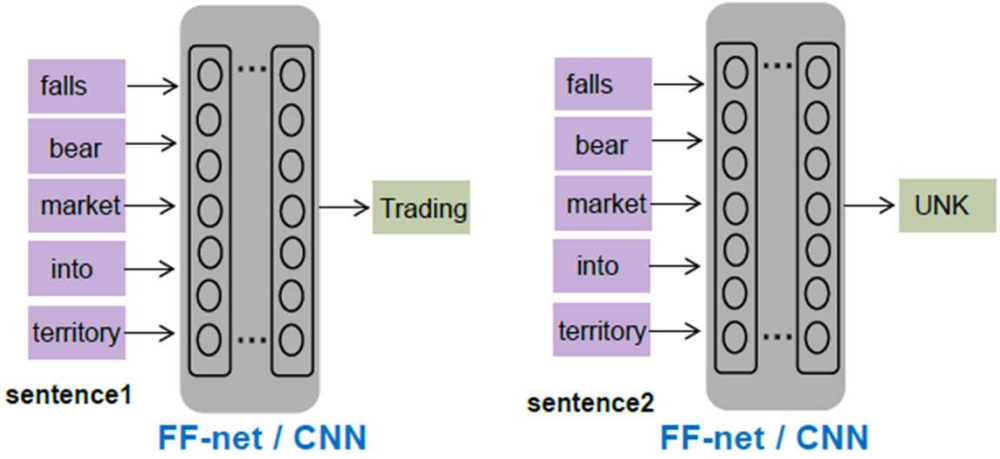
Motivation: Need for Sequential Modeling

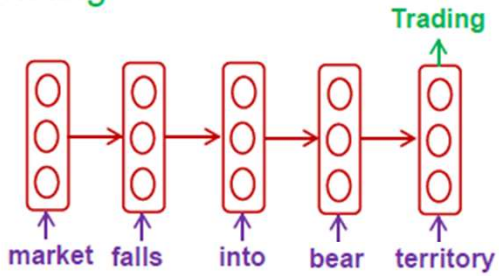
- **Shared features learned across different positions or time steps**

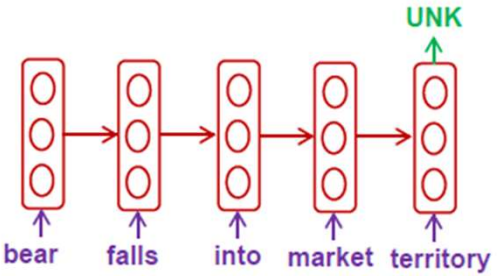
Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*







Sequential model: RNN

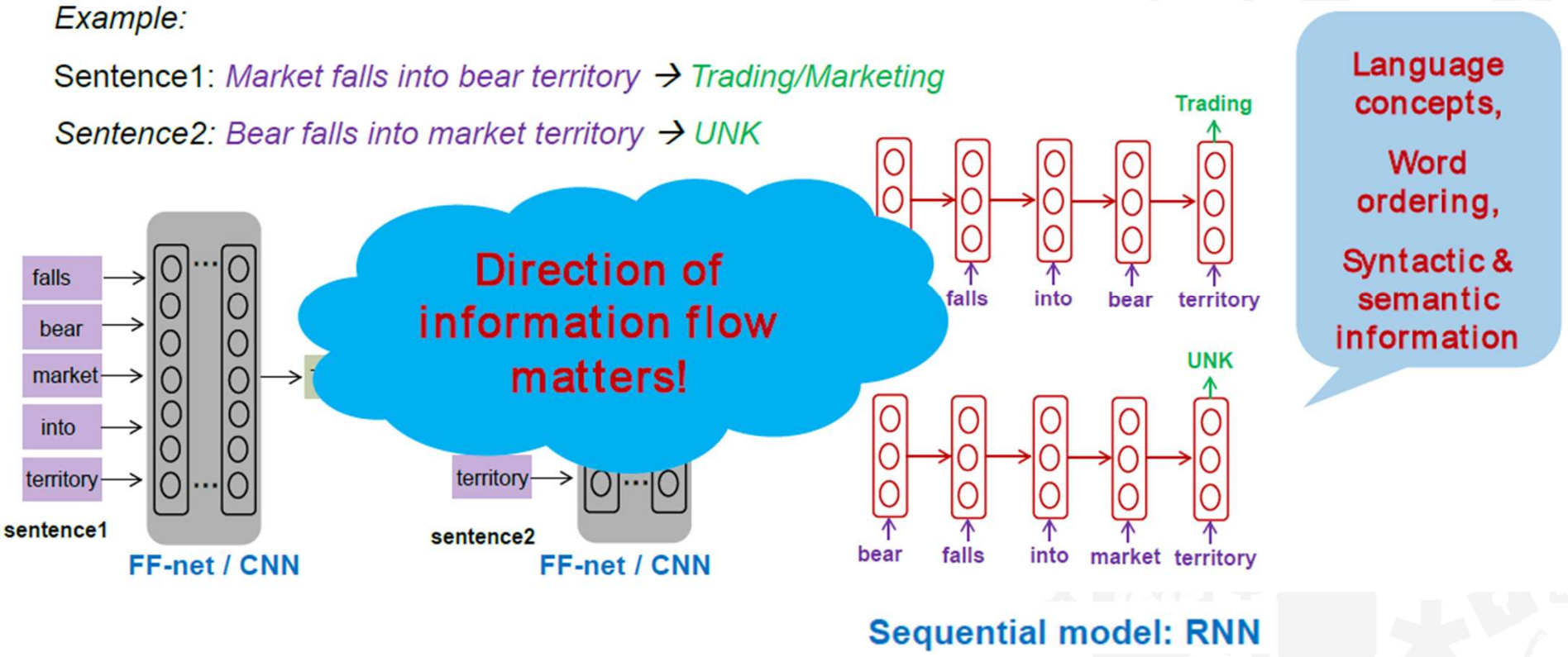
Language concepts,

Word ordering,

Syntactic & semantic information

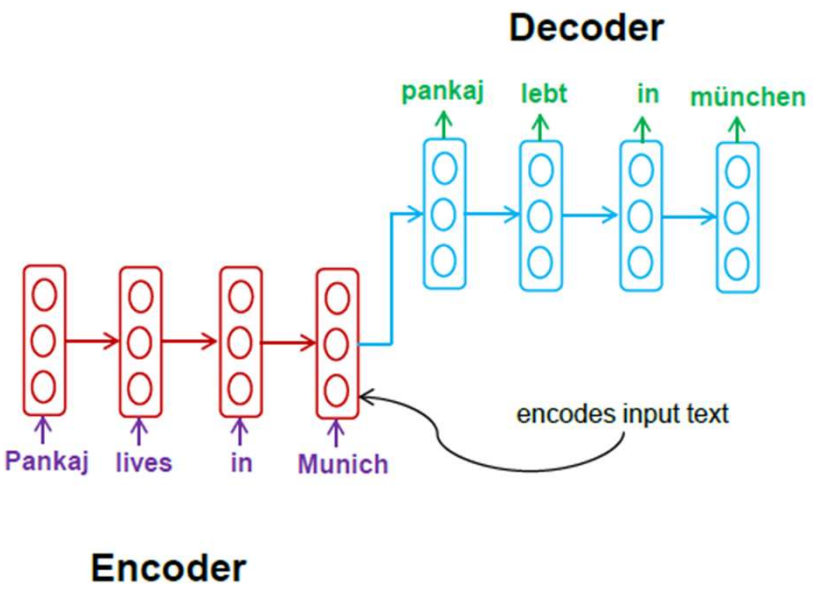
Motivation: Need for Sequential Modeling

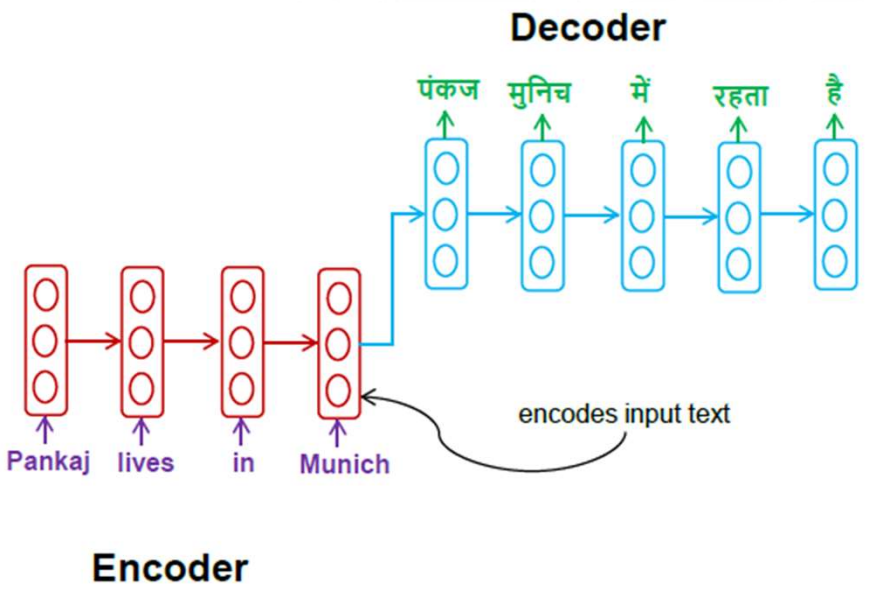
- **Shared features learned across different positions or time steps**



Motivation: Need for Sequential Modeling

- **Machine Translation:** different input & output sizes, incurring sequential patterns





Motivation: Need for Sequential Modeling

- **Convolutional vs. Recurrent Neural Networks**

RNN

- perform well when the input data is interdependent in a sequential pattern
- correlation between previous input to the next input
- introduce bias based on your previous output

CNN/FF-Nets

- all the outputs are self dependent
- *Feed-forward nets don't remember historic input data at test time unlike recurrent networks.*

Motivation: Need for Sequential Modeling

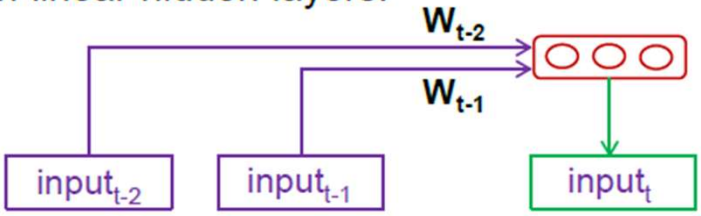
Memory-less Models

Autoregressive models:

Predict the next input in a sequence from a fixed number of previous inputs using “delay taps”.

Feed-forward neural networks:

Generalize autoregressive models by using non-linear hidden layers.



Memory Networks

-possess a dynamic hidden state that can store long term information, e.g., RNNs.

Recurrent Neural Networks:

RNNs are very powerful, because they combine the following properties-

Distributed hidden state: can efficiently store a lot of information about the past.

Non-linear dynamics: can update their hidden state in complicated ways

Temporal and accumulative: can build semantics, e.g., word-by-word in sequence over time

Term dependencies

Short Term Dependencies

→ need recent information to perform the present task.

For example in a language model, predict the next word based on the previous ones.

*"the clouds are in the **?**"* → *'sky'* *"the clouds are in the **sky**"*

→ Easier to predict 'sky' given the context, i.e., *short term dependency*

Long Term Dependencies

→ Consider longer word sequence "I grew up in France…..…… I speak fluent ***French***."

→ Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

Foundation of Recurrent Neural Networks

Goal

- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

punching

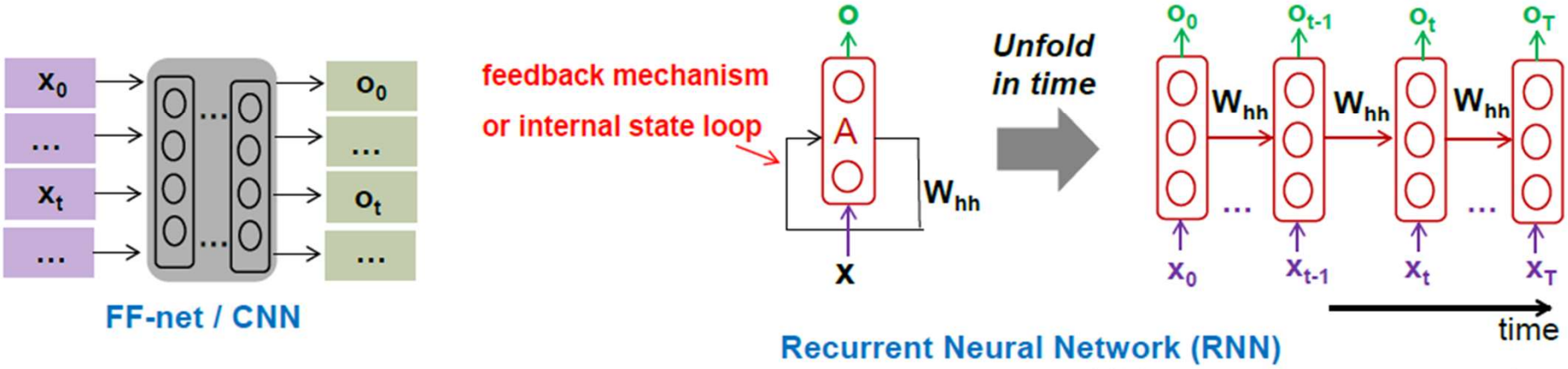
Foundation of Recurrent Neural Networks

Goal

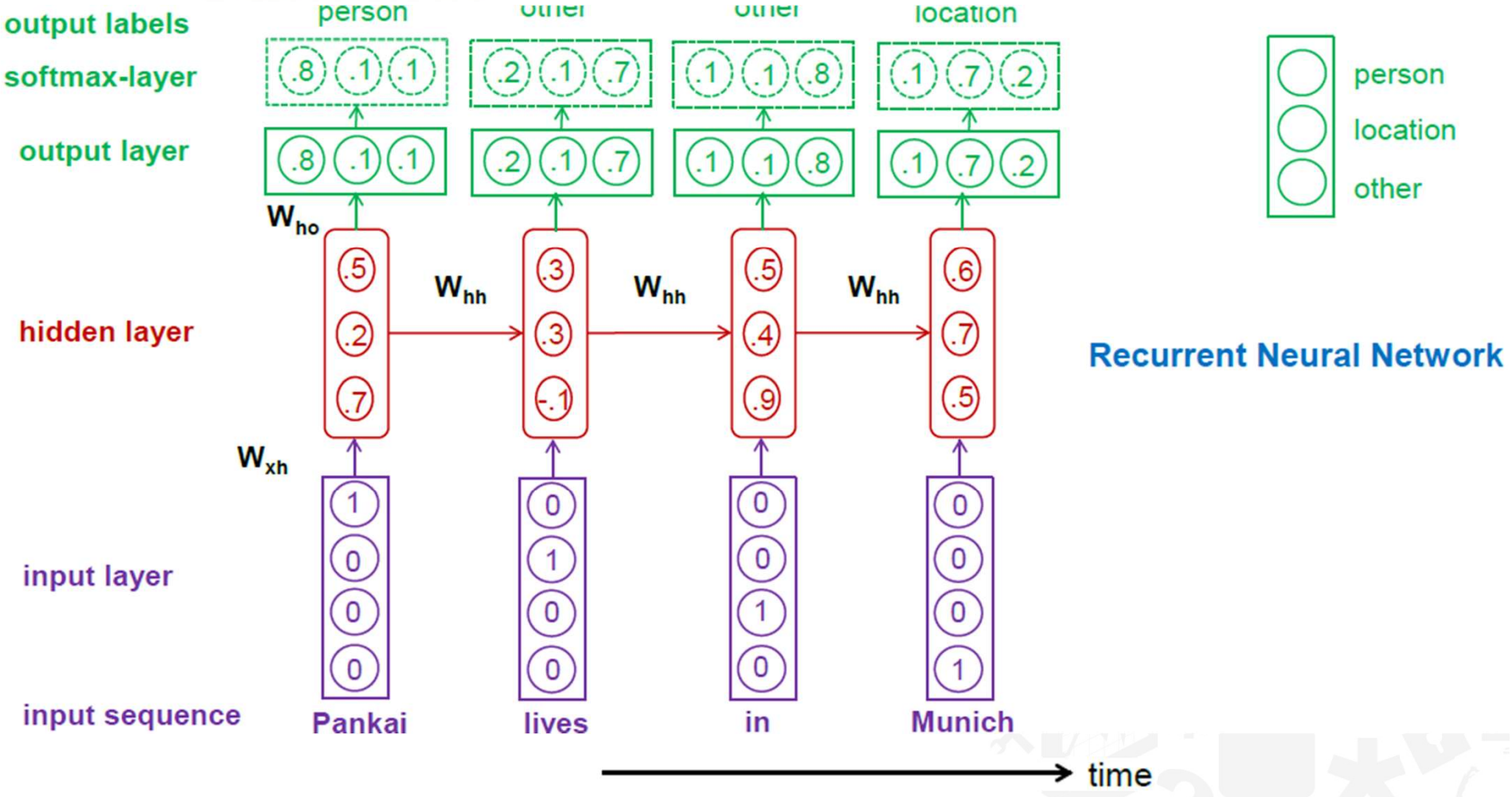
- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

Feed Forward NNets can **not** take **time dependencies** into account.

Sequential data needs a **Feedback Mechanism**.



Foundation of Recurrent Neural Networks



(Vanilla) Recurrent Neural Network

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

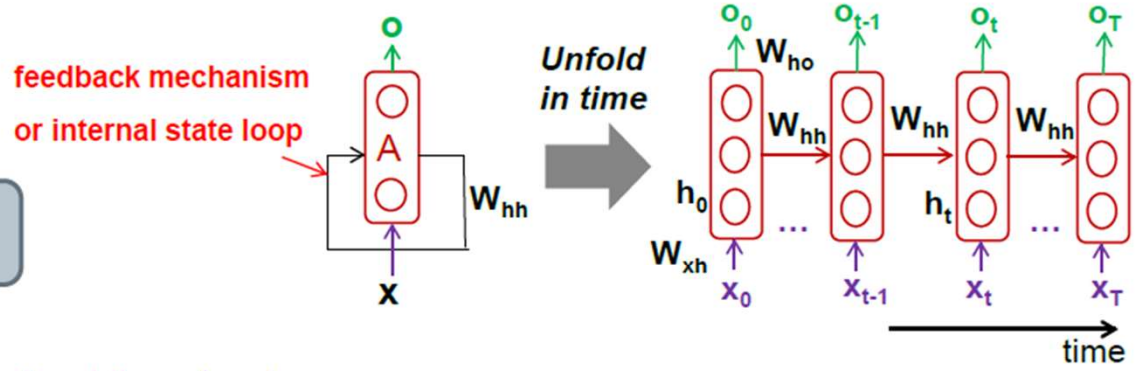
$$h_t = g_\theta(h_{t-1}, x_t)$$

new hidden state at time step, t

some function with parameters W_{hh} W_{xh}

old hidden state at time step, t-1

Input vector at time step, t



Vanilla Recurrent Neural Network (RNN)

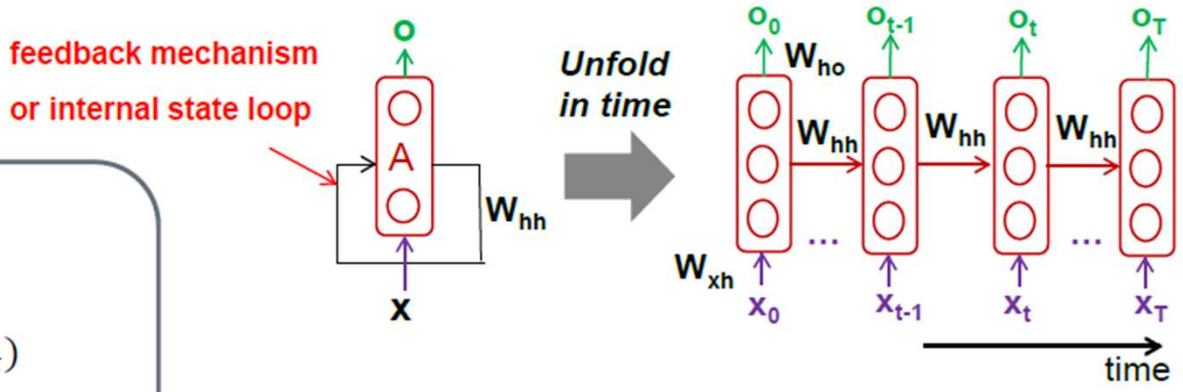
Remark: The same function *g* and same set of parameters **W** are used at every time step

(Vanilla) Recurrent Neural Network

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

$$h_t = g_\theta(h_{t-1}, x_t)$$
$$\downarrow$$
$$h_t = \tanh(W_{hh}\, h_{t-1} + W_{xh} x_t)$$

$$o_t = softmax(W_{ho} h_t)$$



Vanilla Recurrent Neural Network (RNN)

Remark: RNN's can be seen as **selective summarization** of input sequence in a fixed-size state/hidden vector via a recursive update.

RNN : Probablistic Interpretation

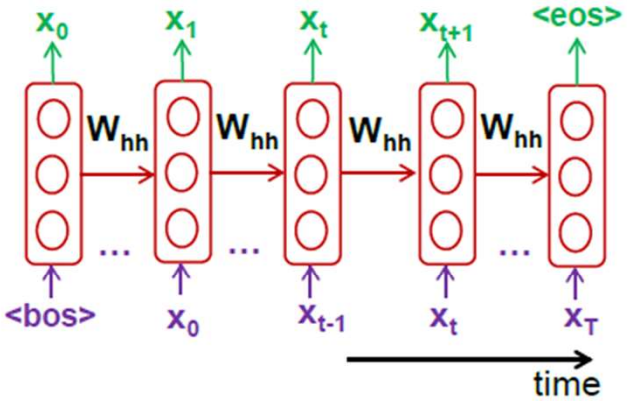
RNN as a generative model

- induces a set of procedures to model

the conditional distribution of $\mathbf{x}_{t+1}$ given $\mathbf{x}$<=t

for all t = 1, …,T

$$P(x) = P(x_1, \dots x_T) = \sum_{t=1}^{T} P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$



Generative Recurrent Neural Network (RNN)

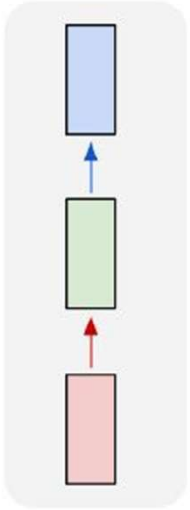
- Think of the output as the probability distribution of the $\mathbf{x}_t$ given the previous ones in the sequence
- Training: Computing probability of the sequence and Maximum likelihood training

$$L_t = -\log P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$

Details: https://www.cs.cmu.edu/~epxing/Class/10708-17/project-reports/project10.pdf

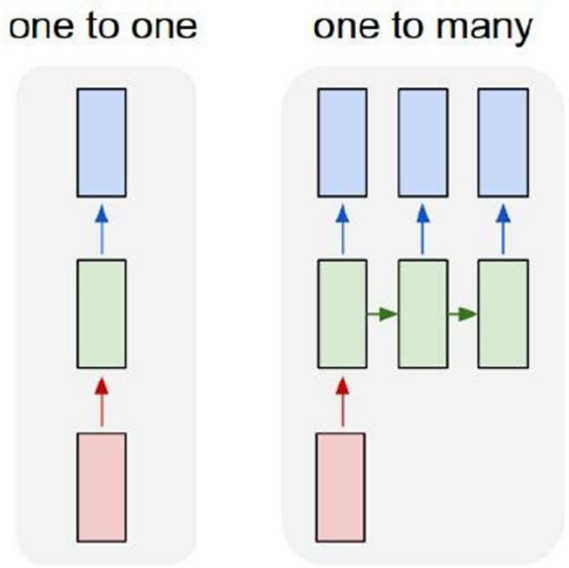
RNN : Computational Graphs

one to one



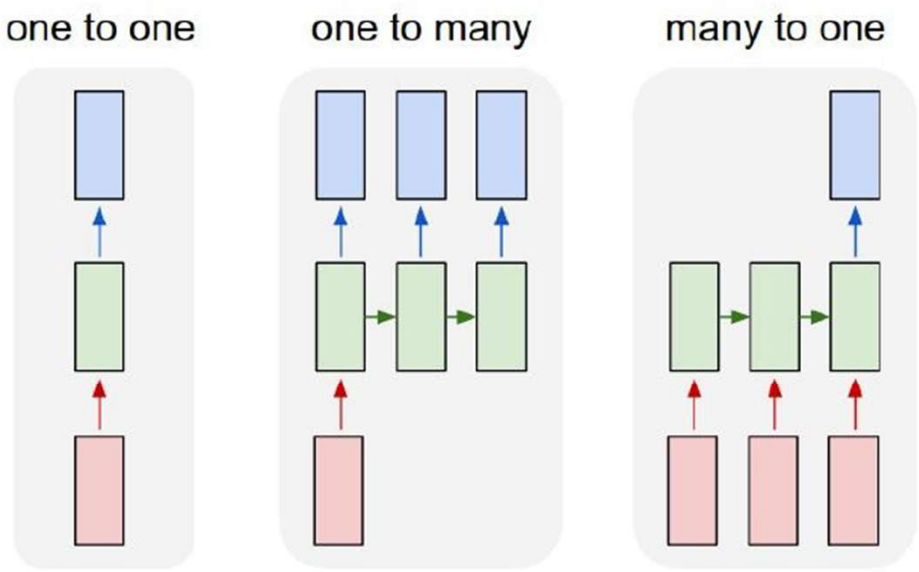
Vanilla Neural Networks

RNN : Computational Graphs



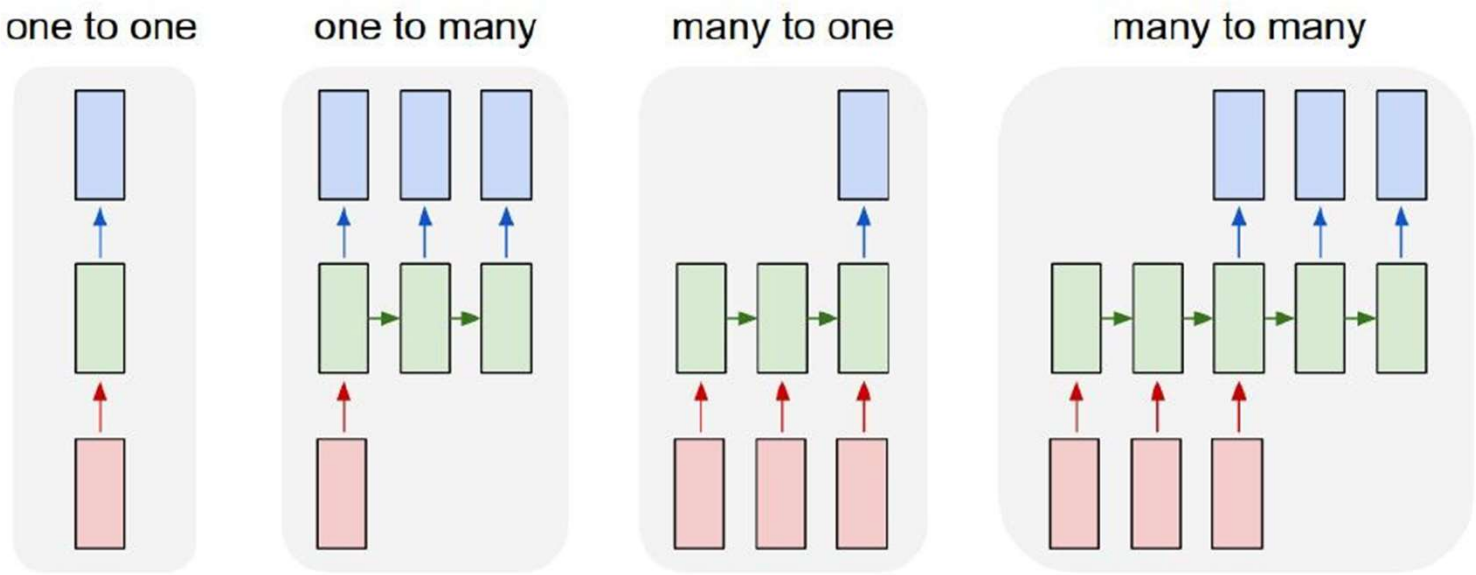
e.g. **Image Captioning**
image -> sequence of words

RNN : Computational Graphs



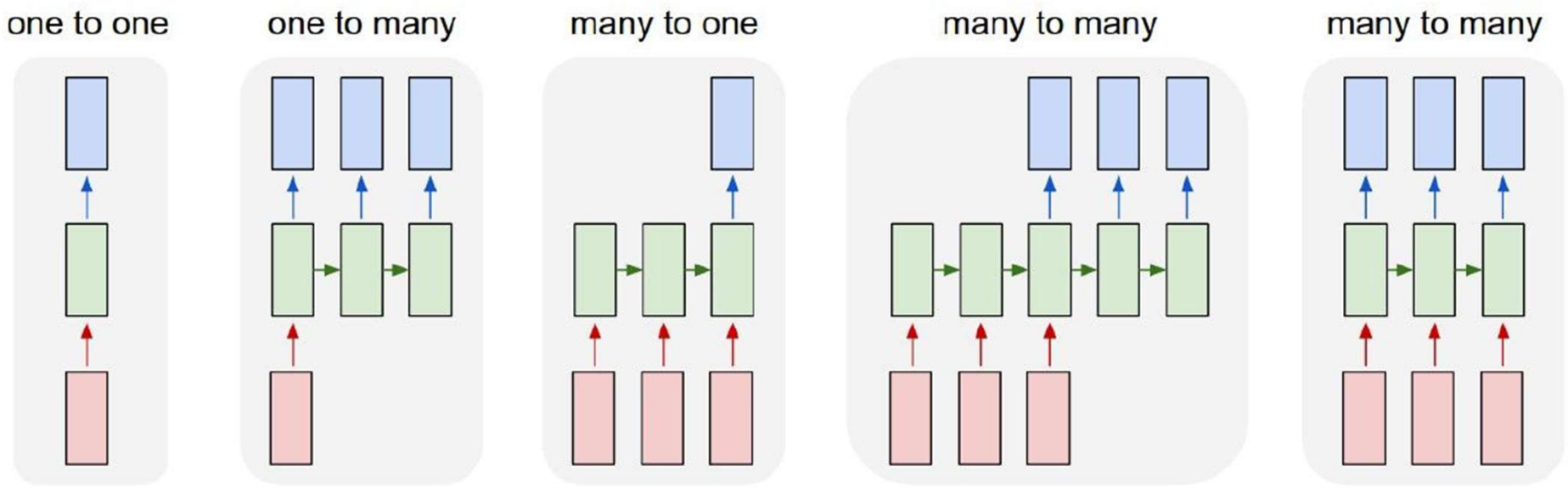
e.g. **action prediction**
sequence of video frames -> action class

RNN : Computational Graphs

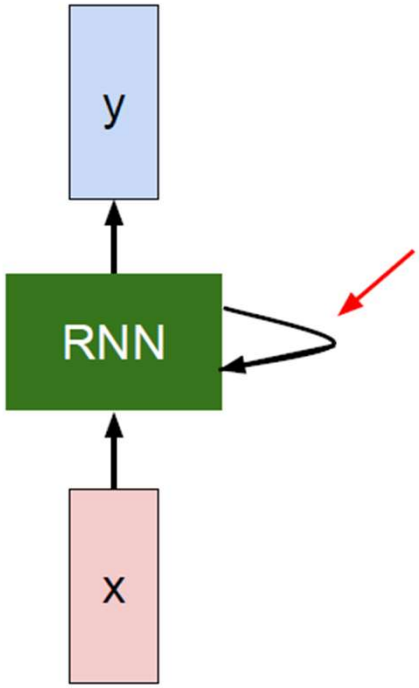


E.g. **Video Captioning**
Sequence of video frames -> caption

RNN : Computational Graphs

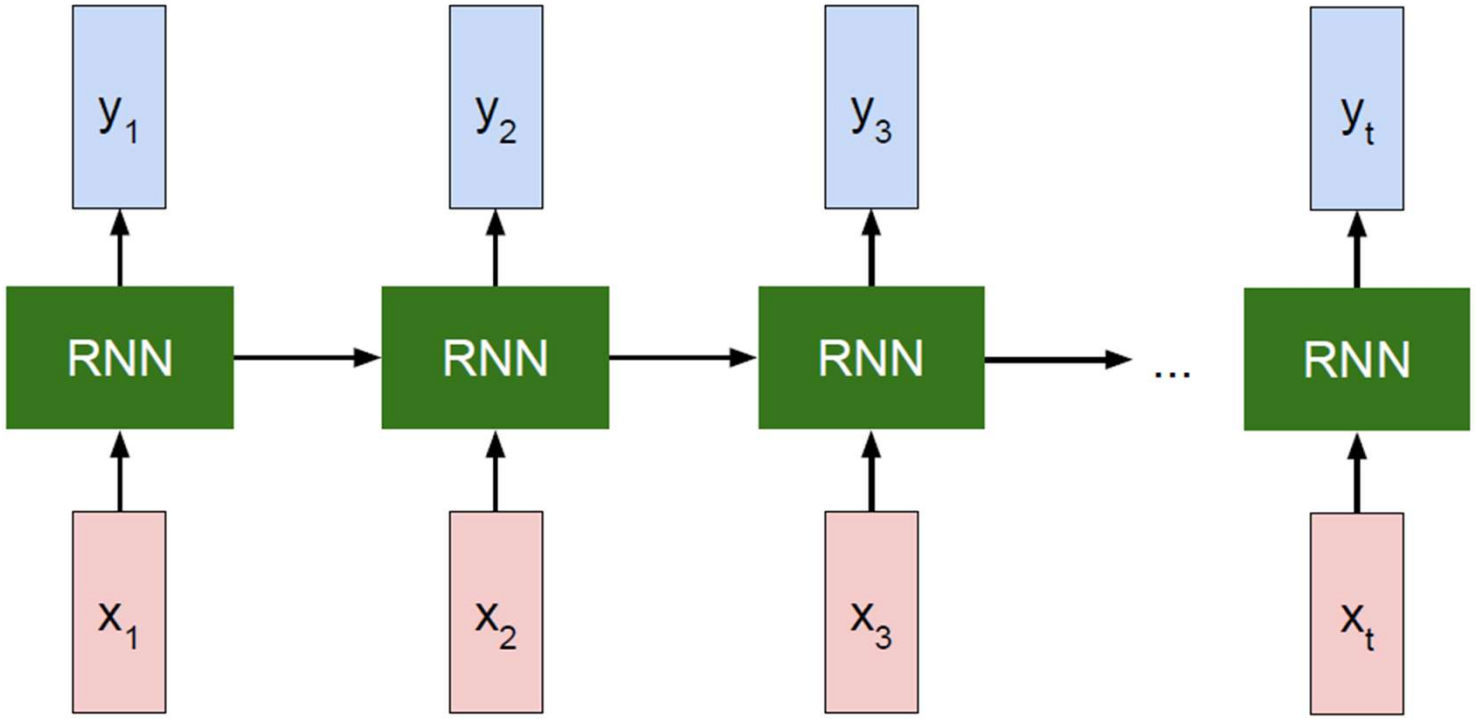


Recurrent Neural Neetwork



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

RNN Unrolled



RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

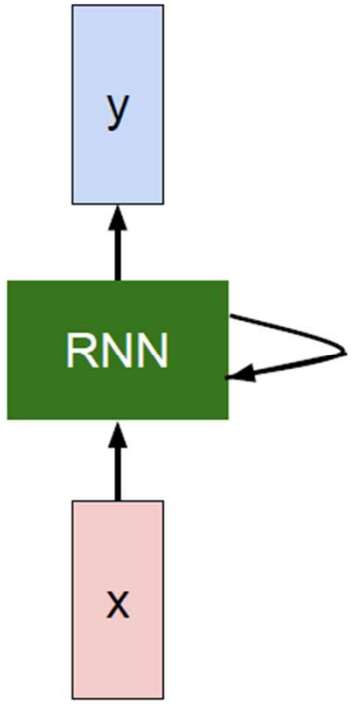
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN output generation

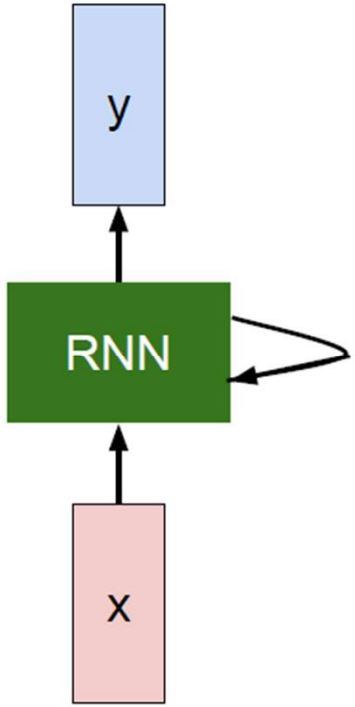
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

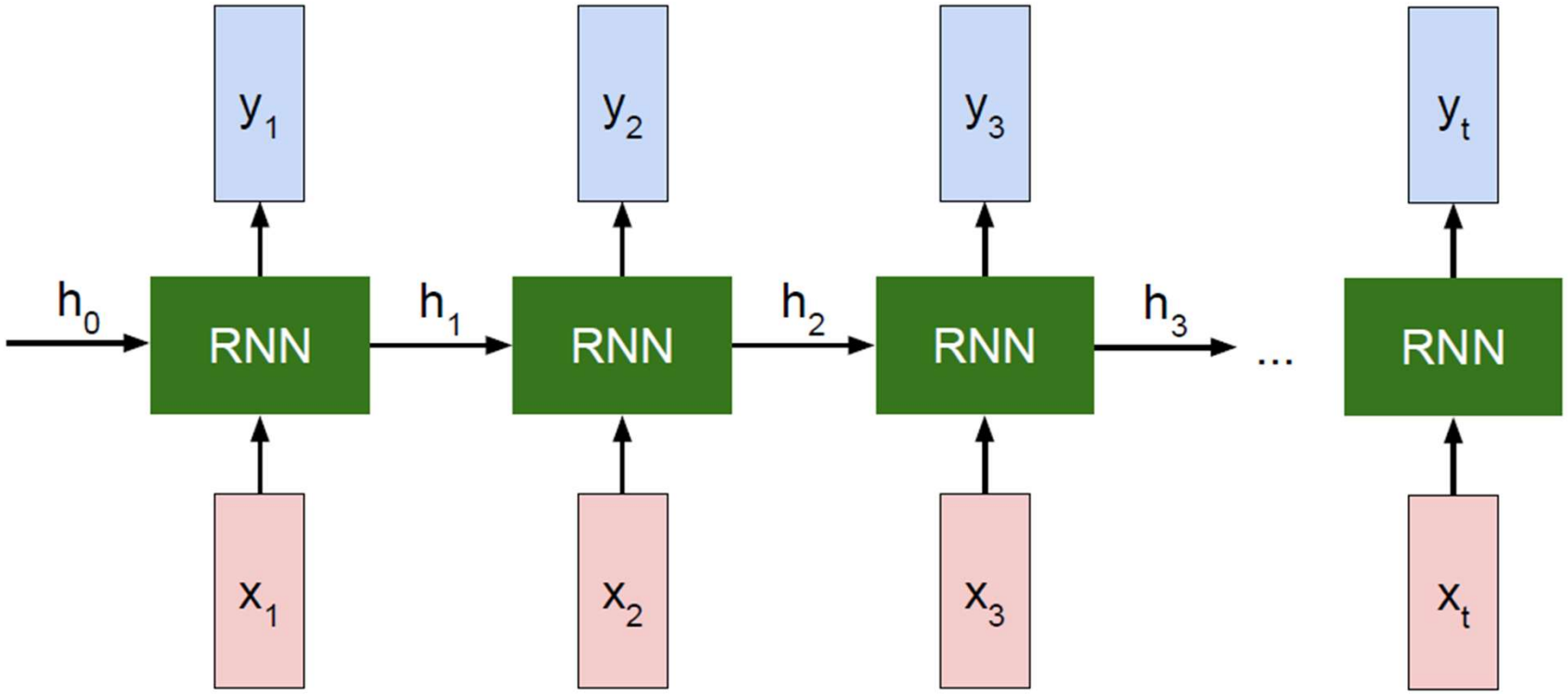
output

new state

another function with parameters W_o



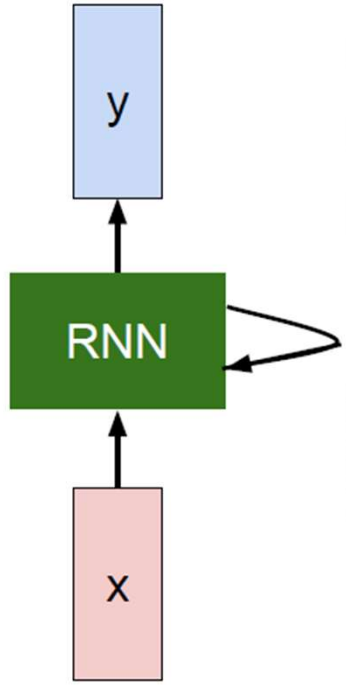
Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

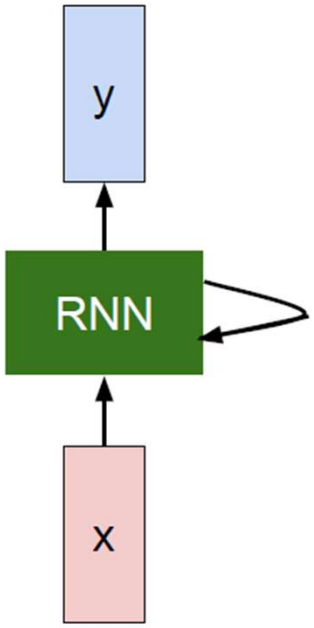
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

Vanilla RNN

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

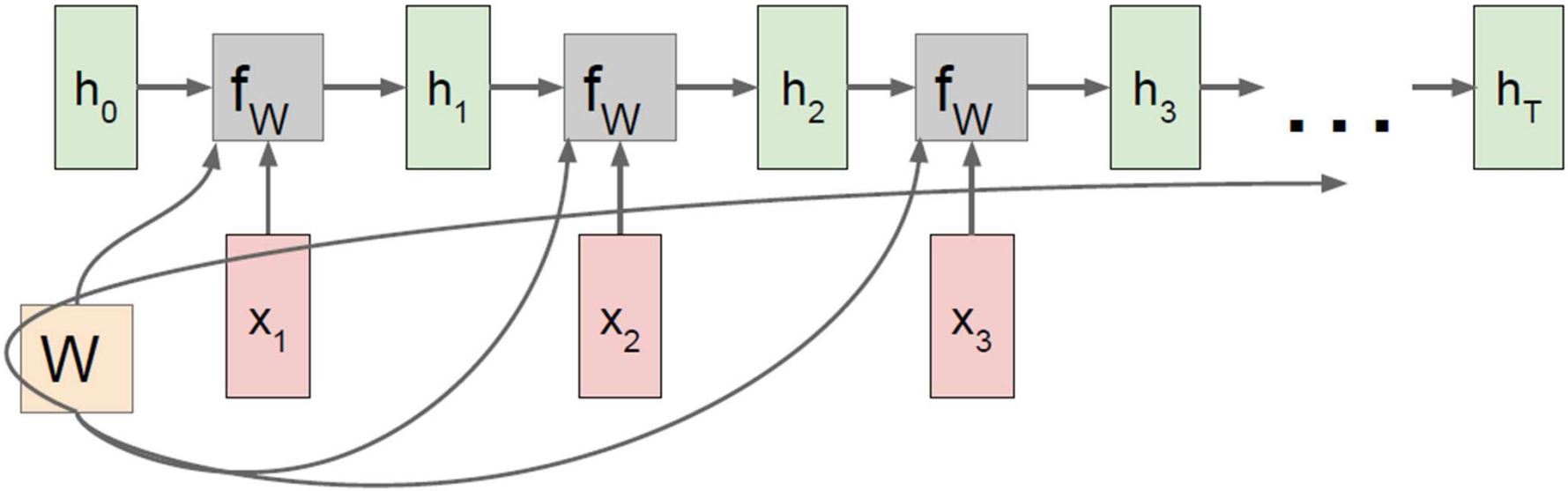
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

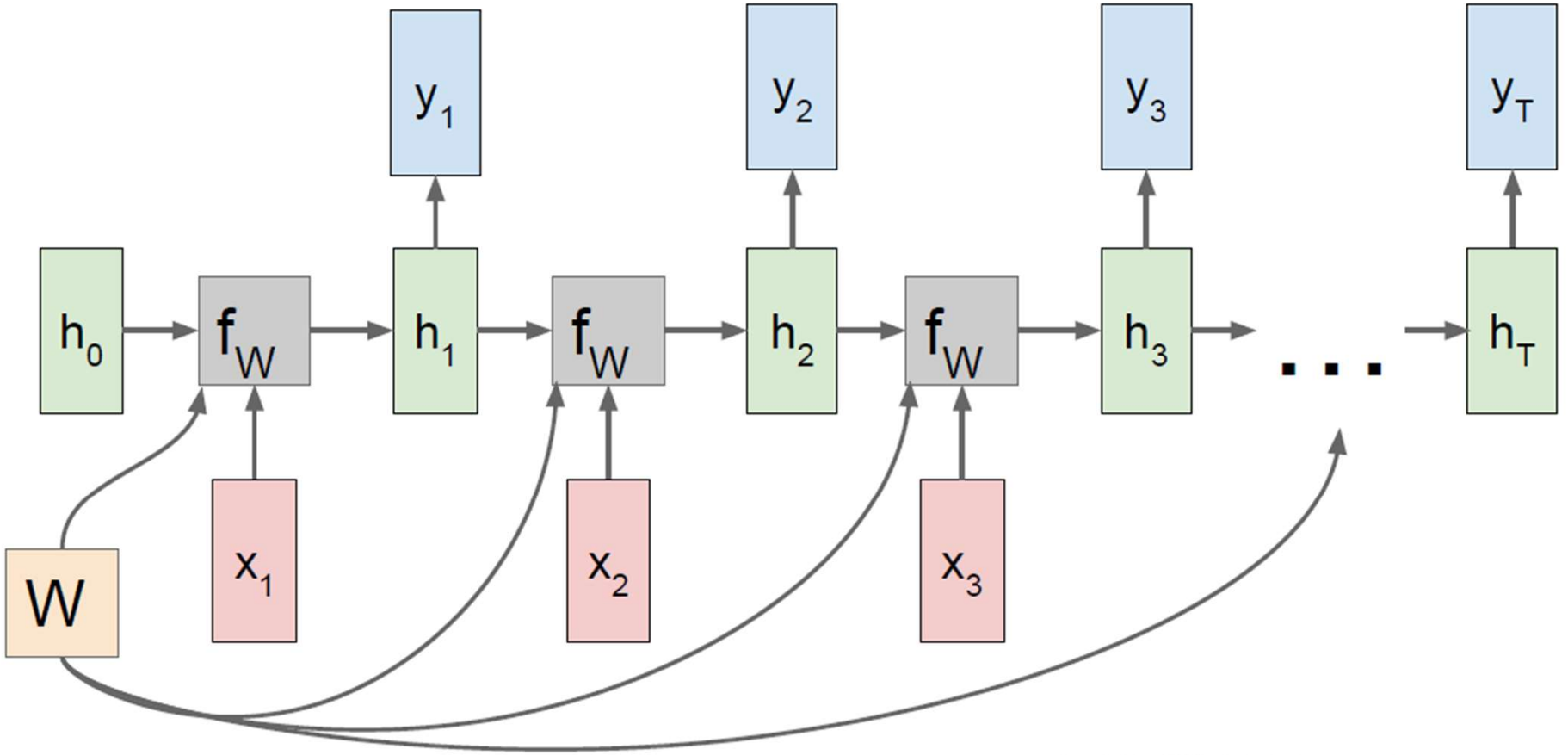
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN: Computational Graphs

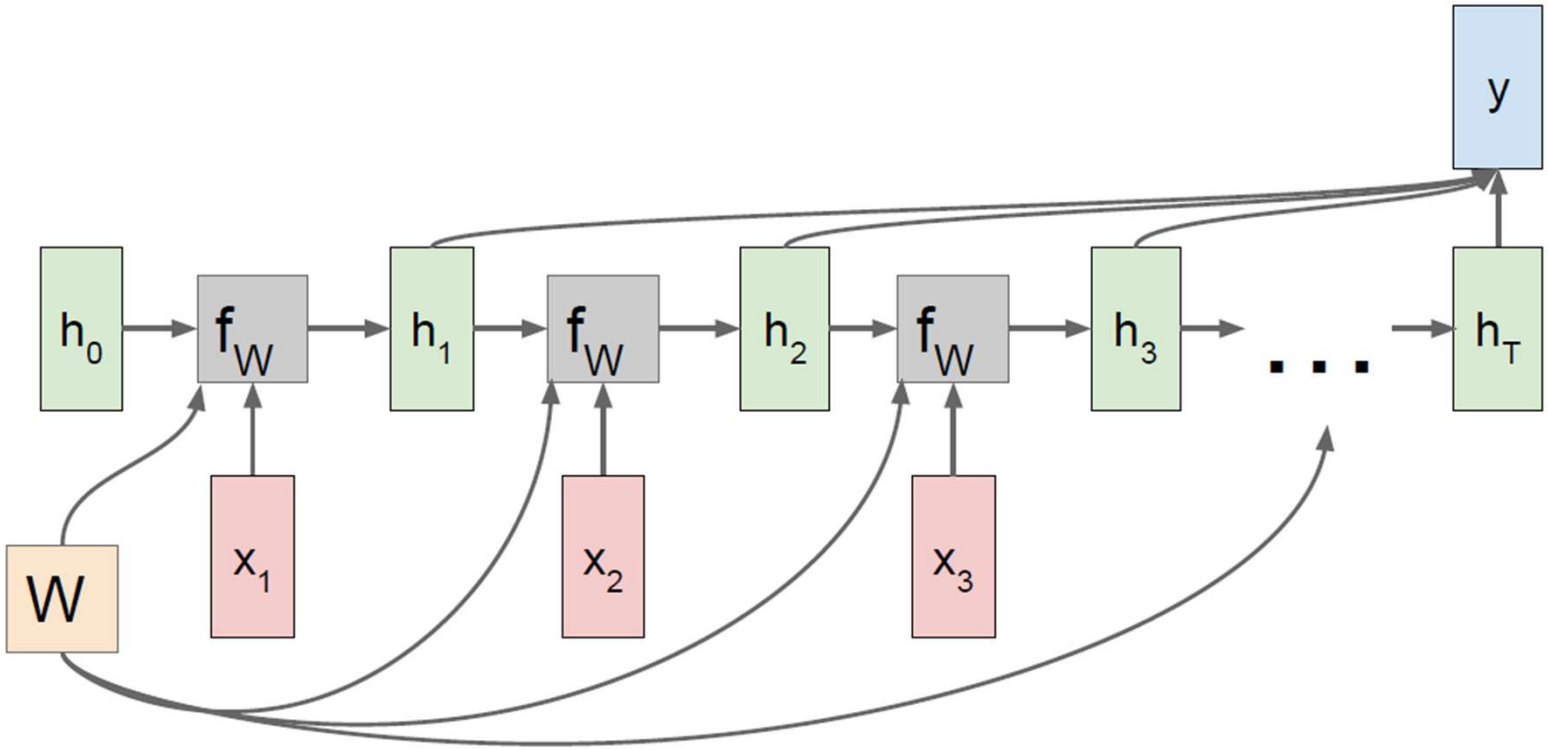
Re-use the same weight matrix at every time-step



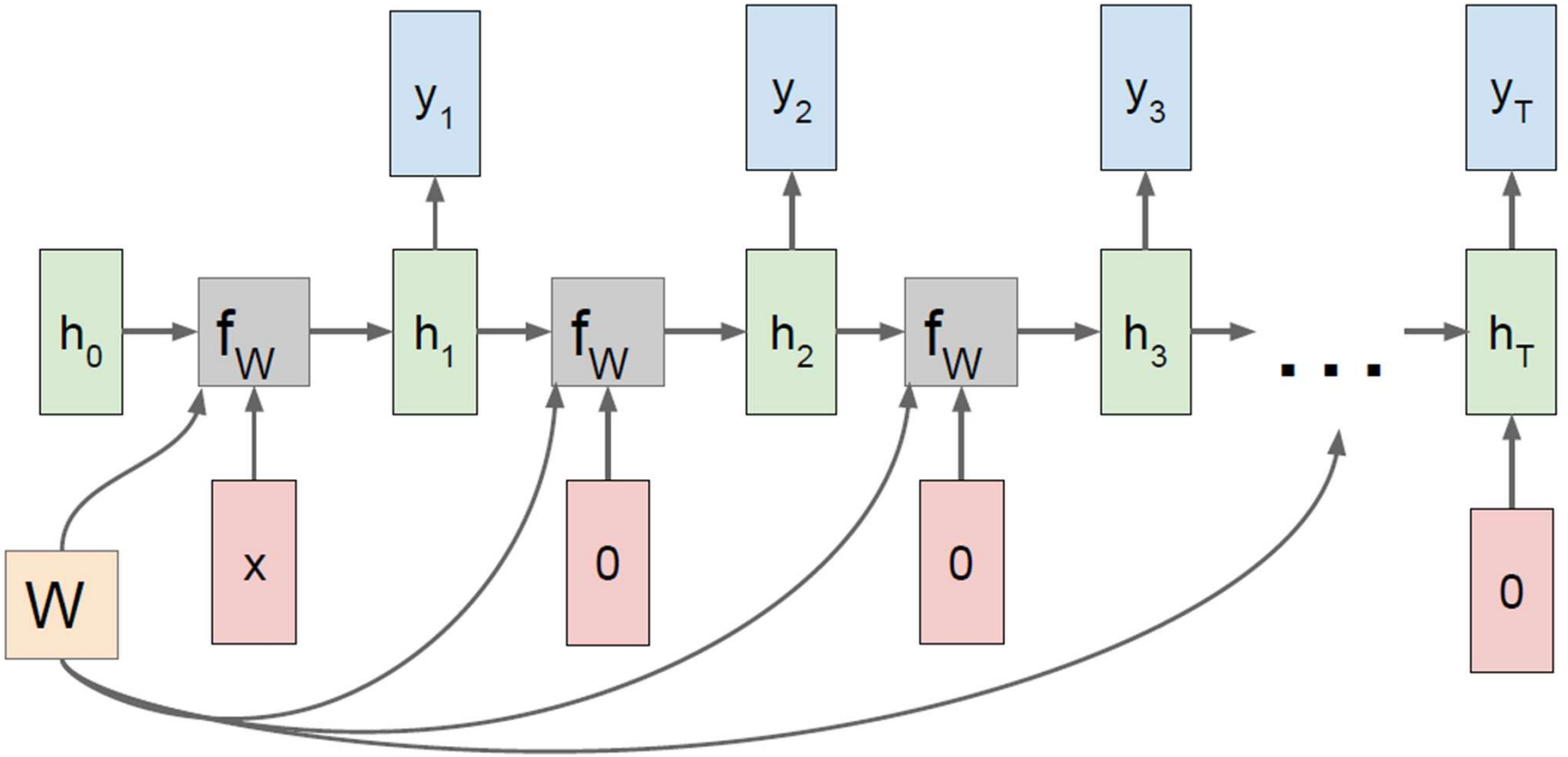
RNN: Computational Graphs: Many to Many



RNN: Computational Graphs: Many to One



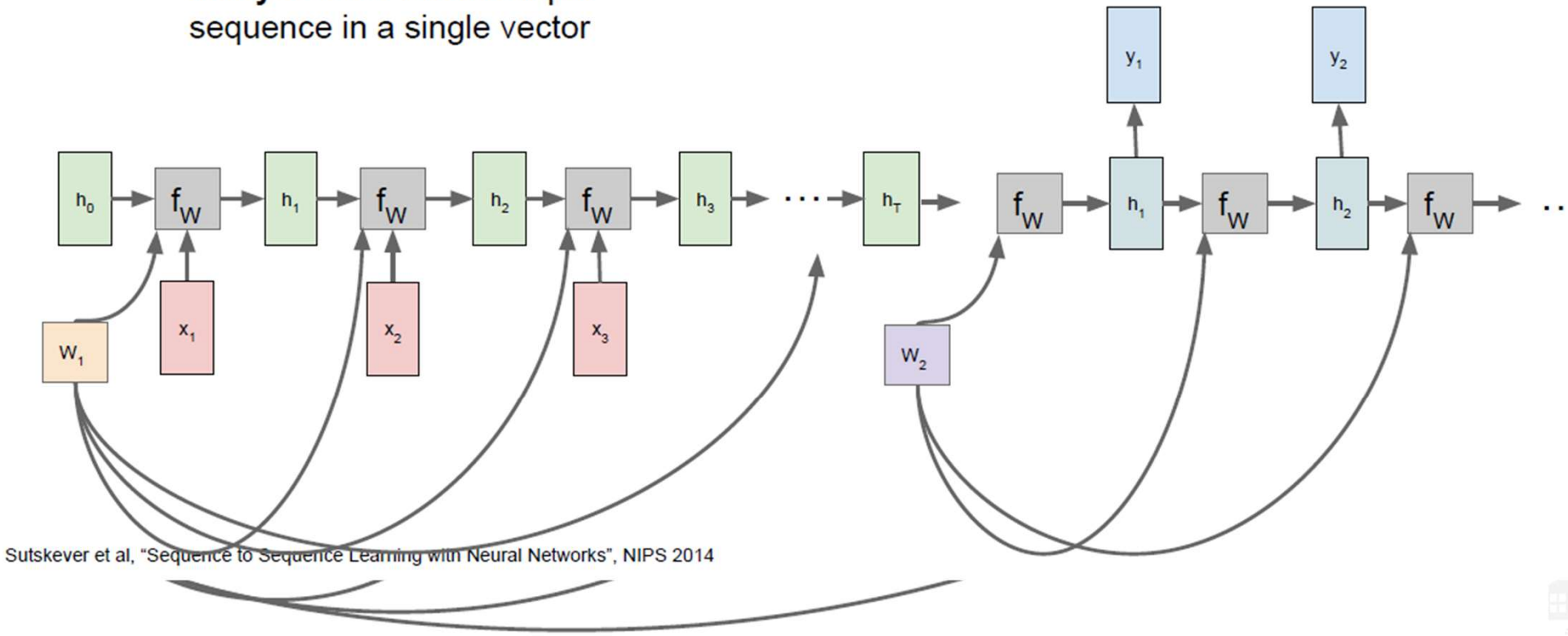
RNN: Computational Graphs: One to Many



RNN: Computational Graphs: One to Many

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector

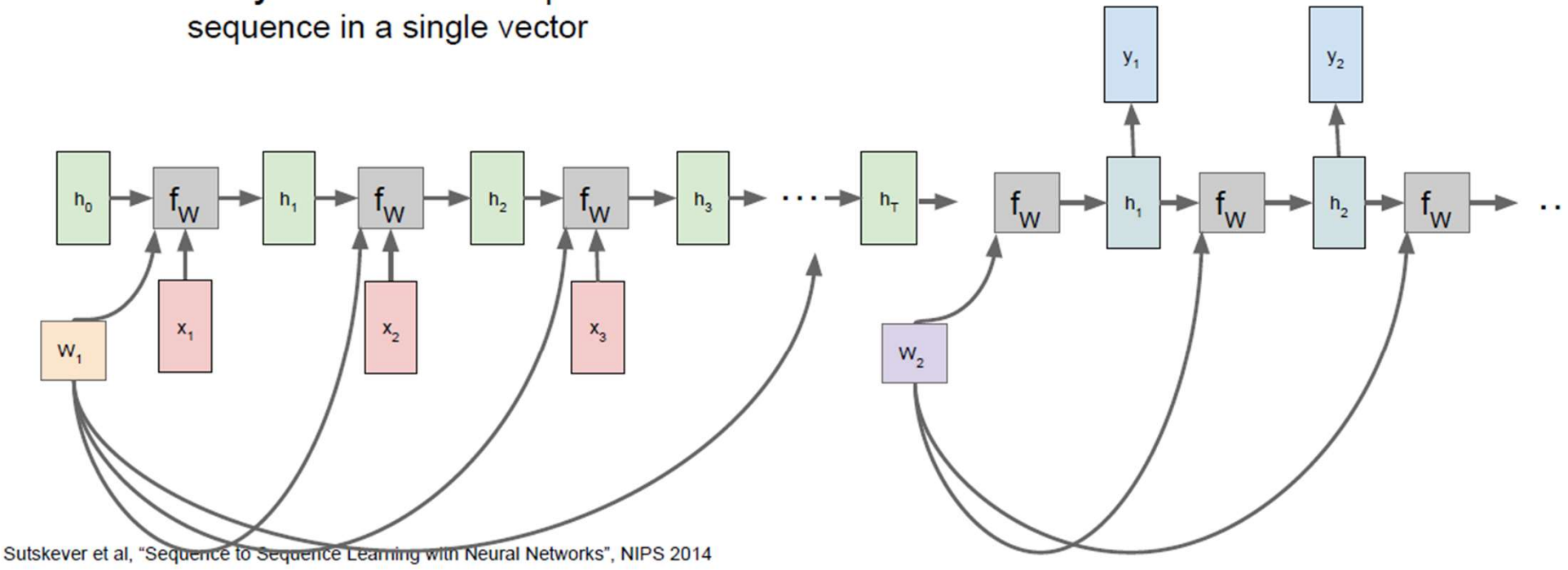


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Seq2Seq. : Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



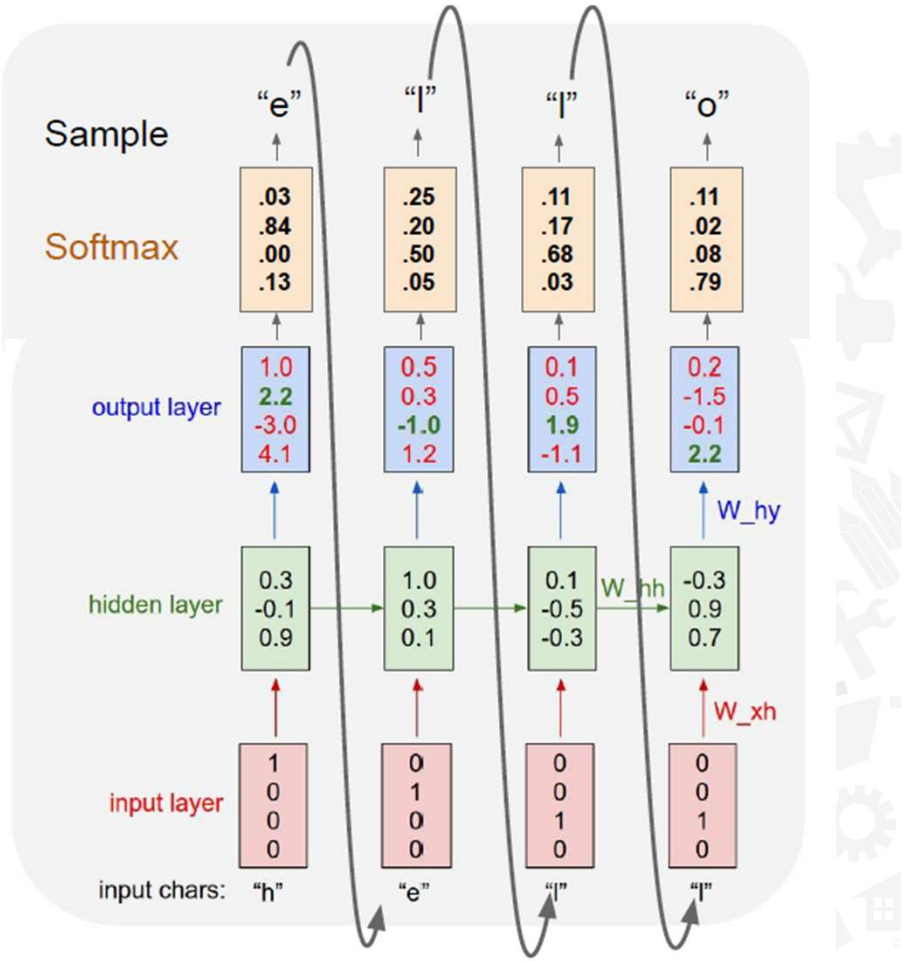
Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

RNN Example

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

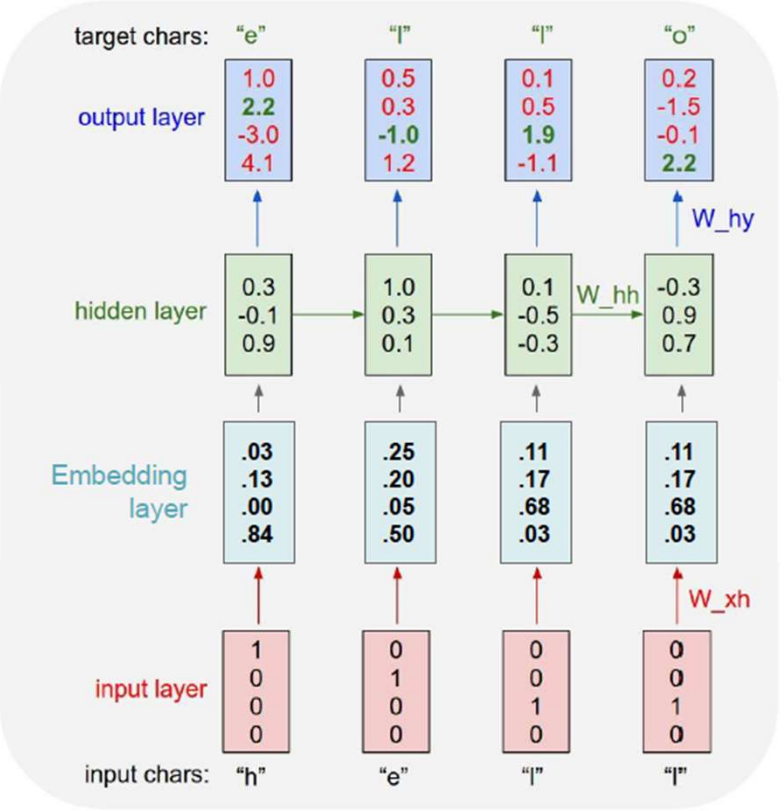


RNN Example

Example: Character-level Language Model Sampling

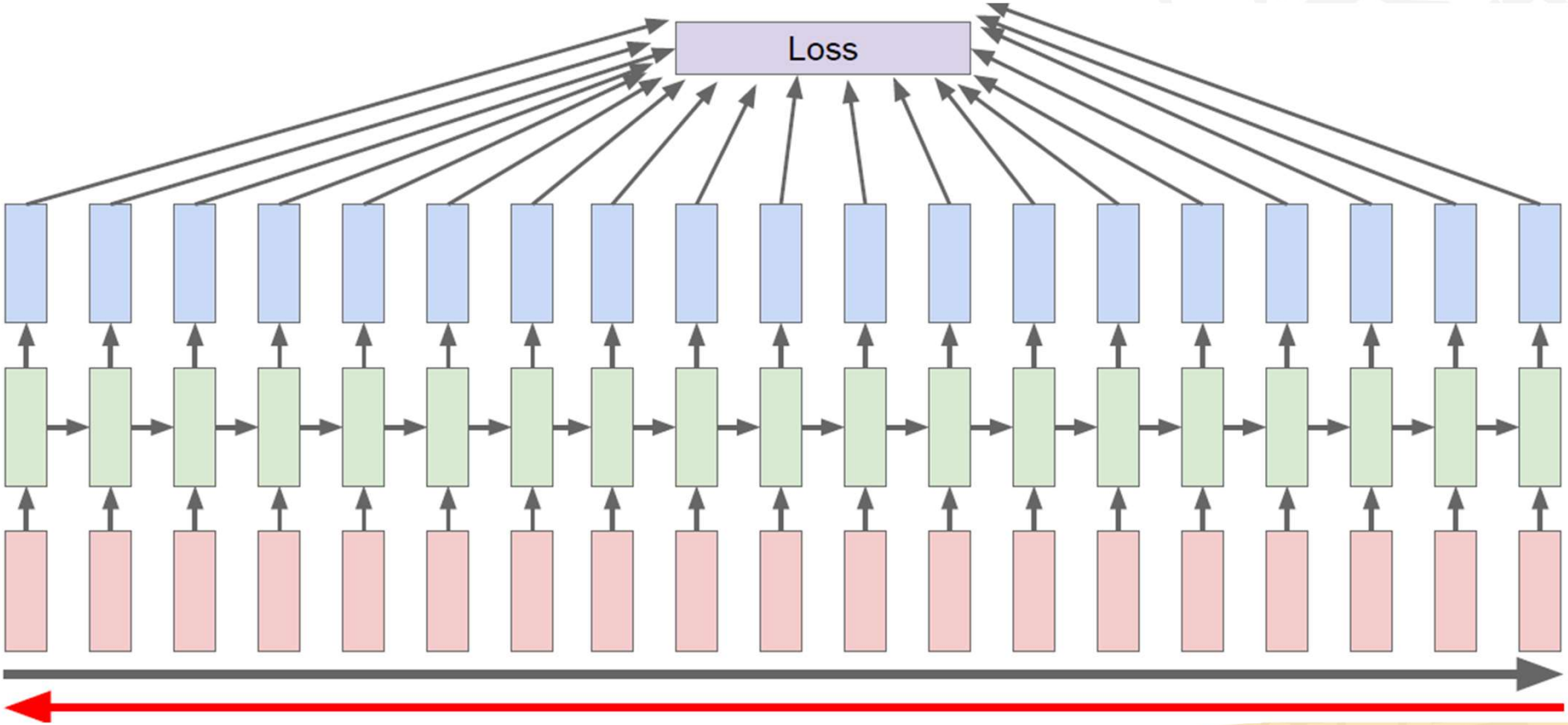
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. We often put a separate **embedding** layer between input and hidden layers.

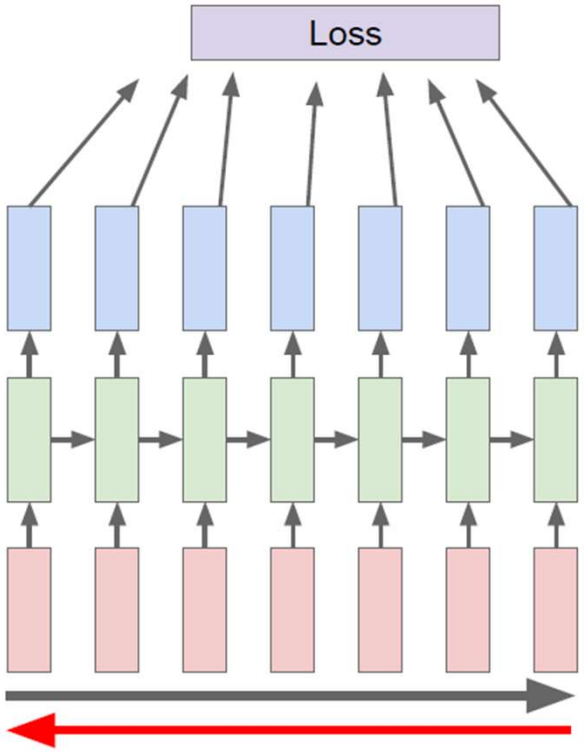


Backpropagation through time

Forward through entire sequence to compute loss,
then backward through entire sequence to compute gradient

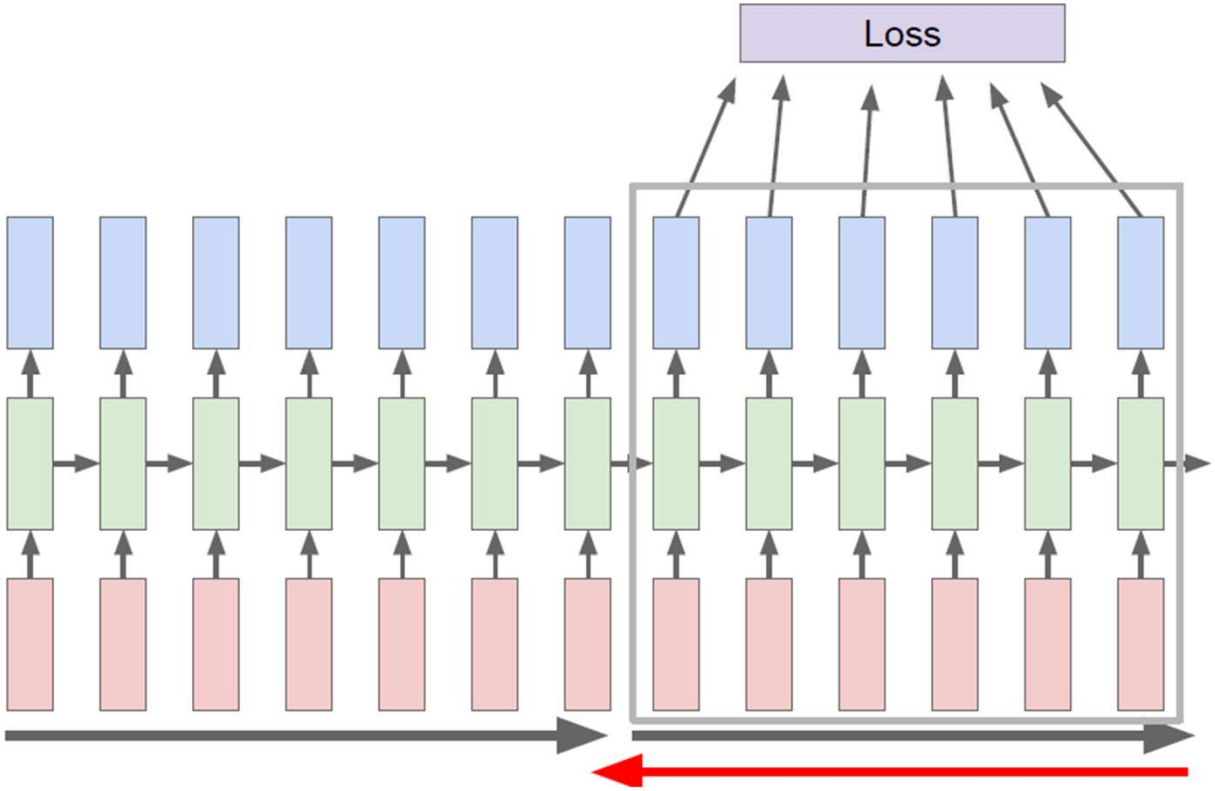


Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Trade-offs of RNN

RNN Advantages:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Example: Image Captioning

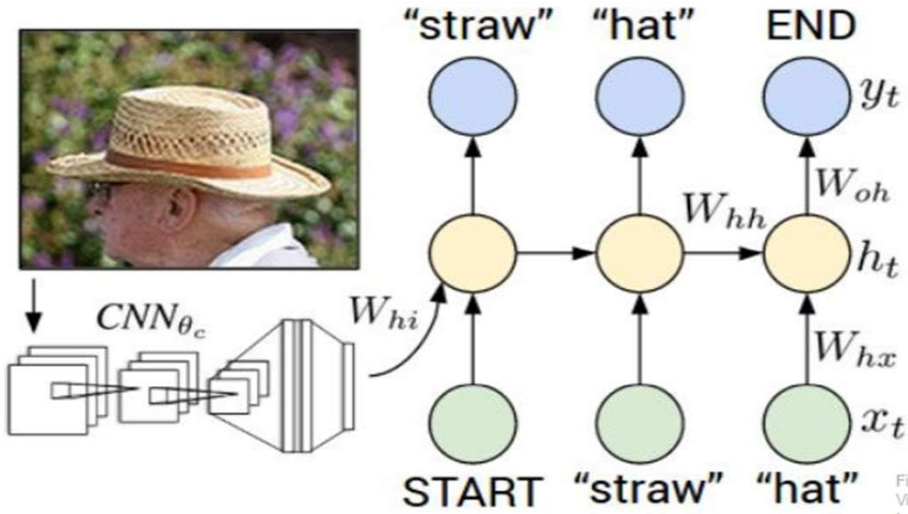
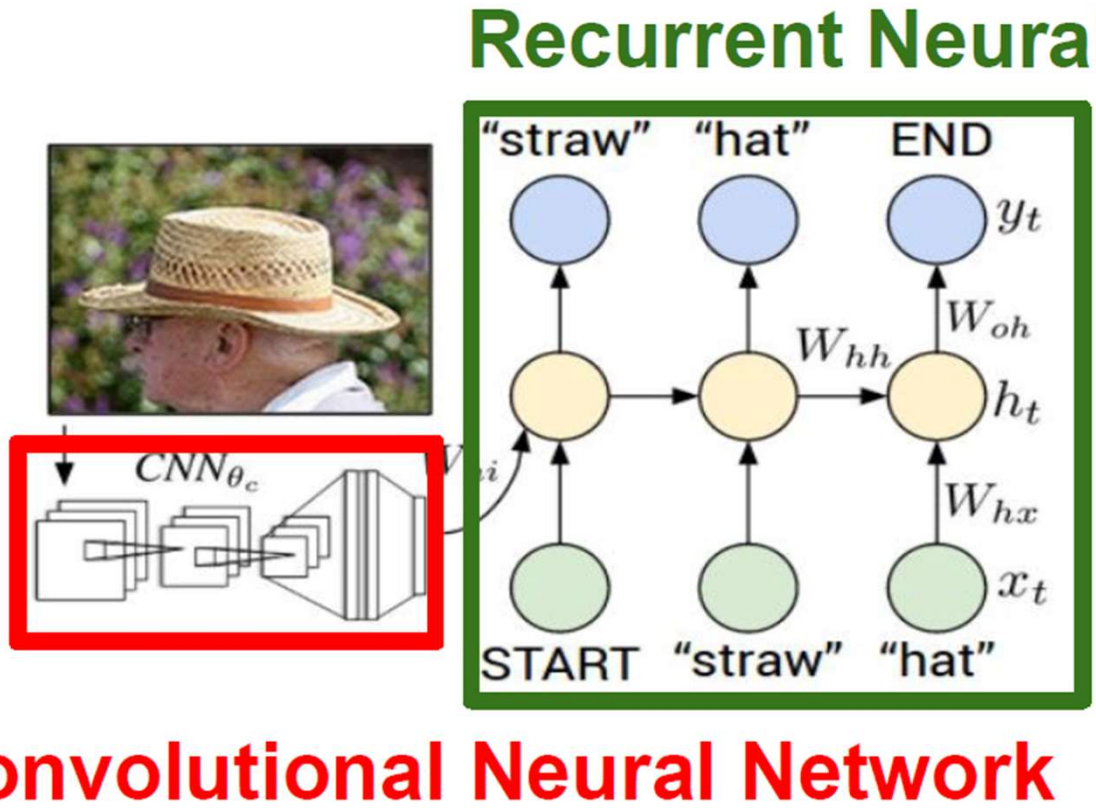


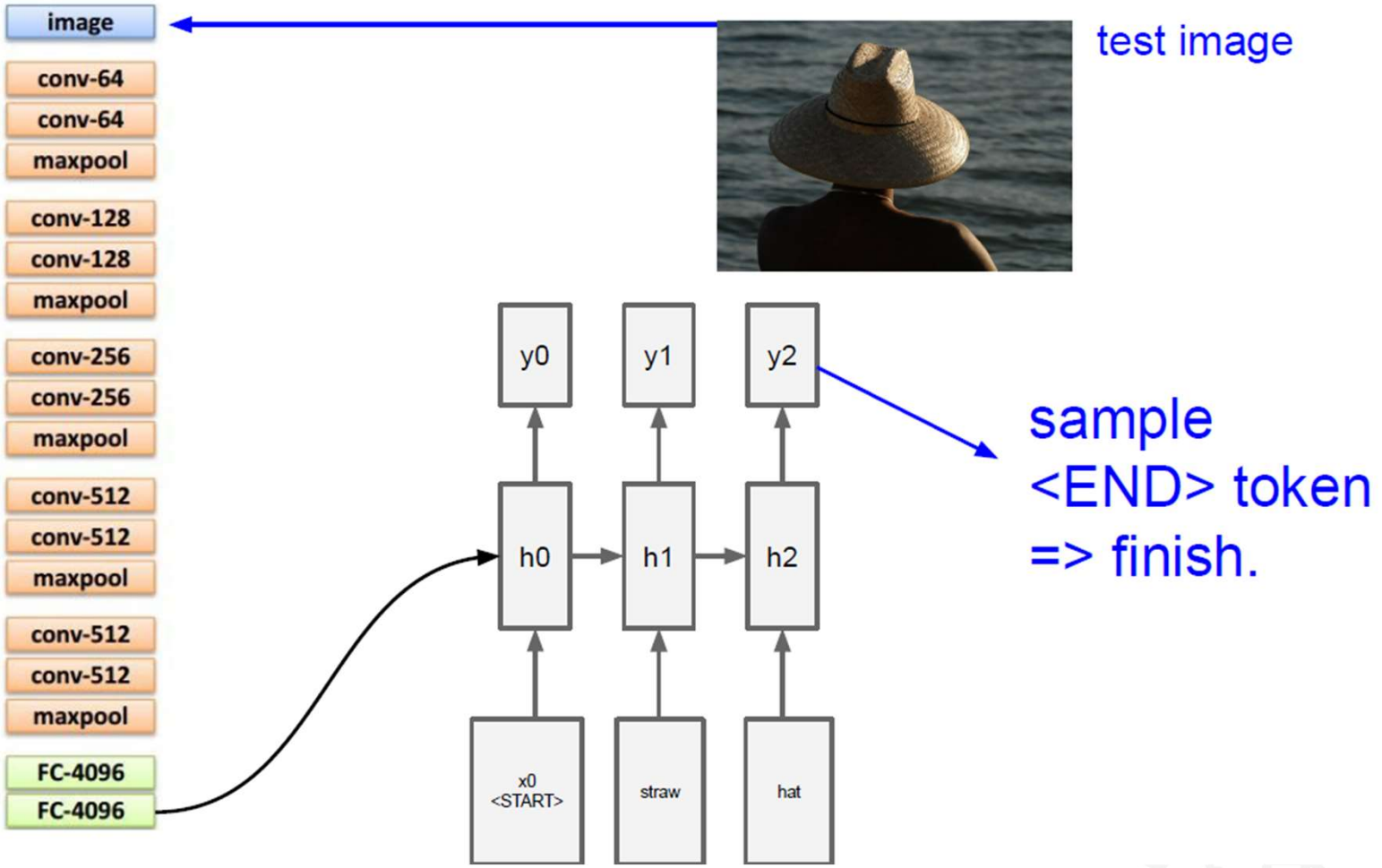
Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Example: Image Captioning



Example: Image Captioning



Example: Image Captioning

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Example: Visual Question Answering(VQA)

Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

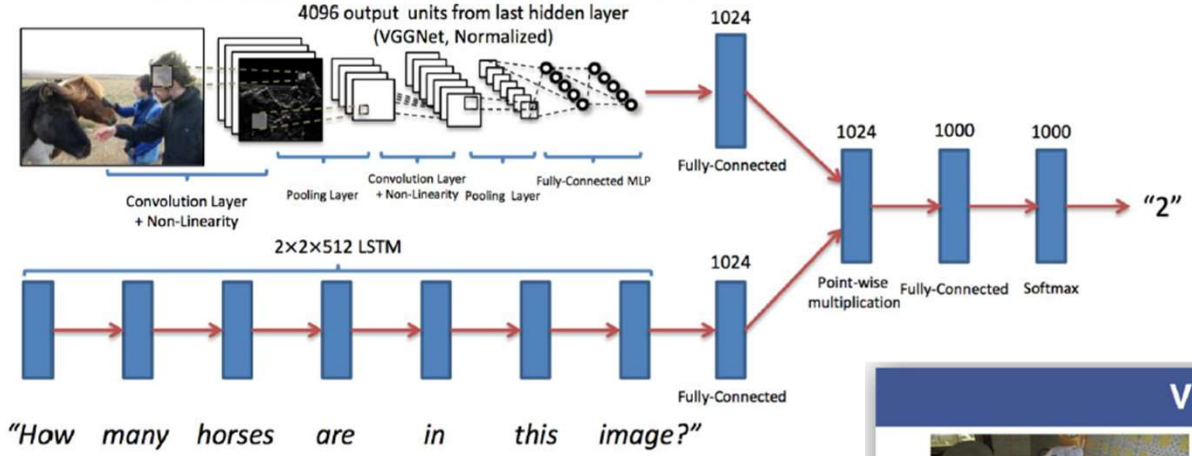
A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

Q: Who is under the umbrella?

A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Example: Visual Question Answering(VQA)



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.

Multi-layer RNN

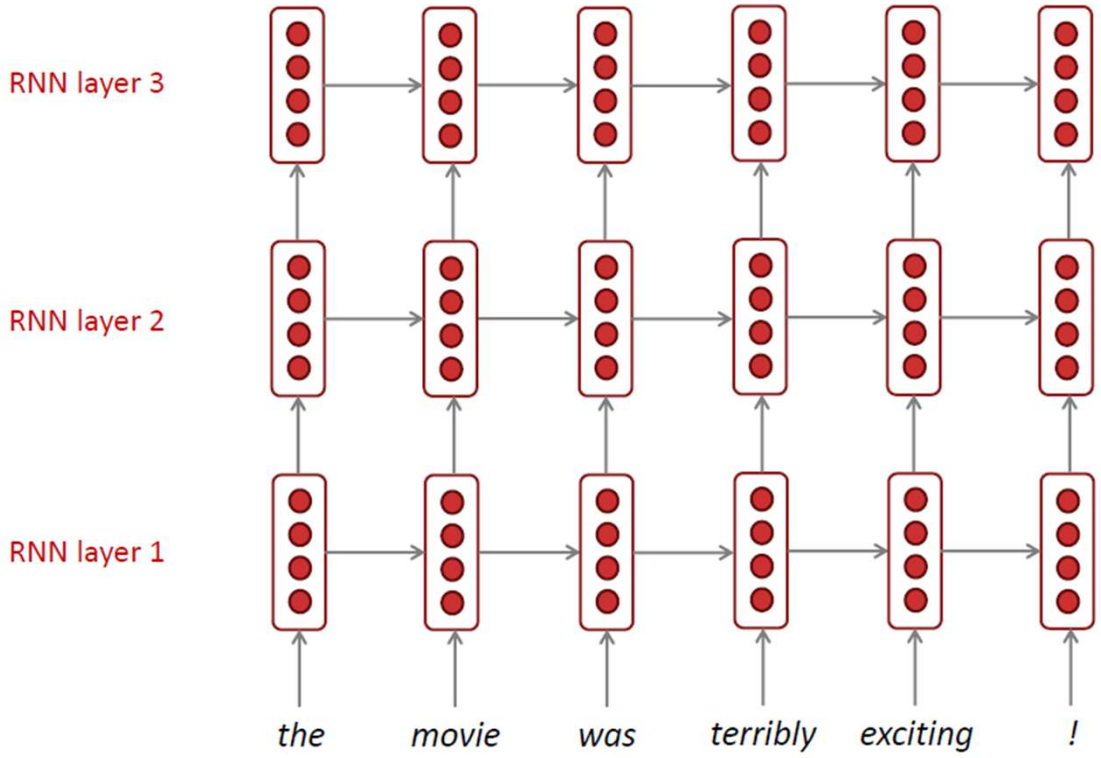
This contextual representation of "terribly" has both left and right context!

- RNNs are already "deep" on one dimension (they unroll over many timesteps)
- We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer RNN

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Bi-directional RNNs

Bidirectional Recurrent Neural Networks (BRNN)

- connects two hidden layers of opposite directions to the same output
- output layer can get information from past (backwards) and future (forward) states simultaneously
- learn representations from future time steps to better understand the context and eliminate ambiguity

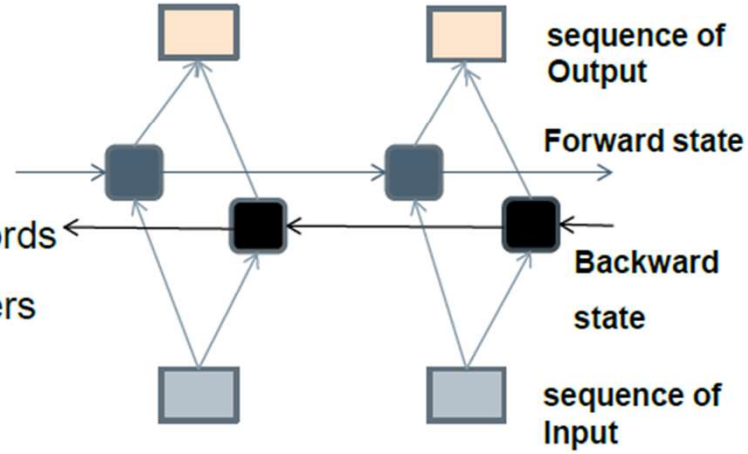
Example sentences:

Sentence1: "He said, ***Teddy*** bears are on sale"

Sentnce2: "He said, ***Teddy*** Roosevelt was a great President".

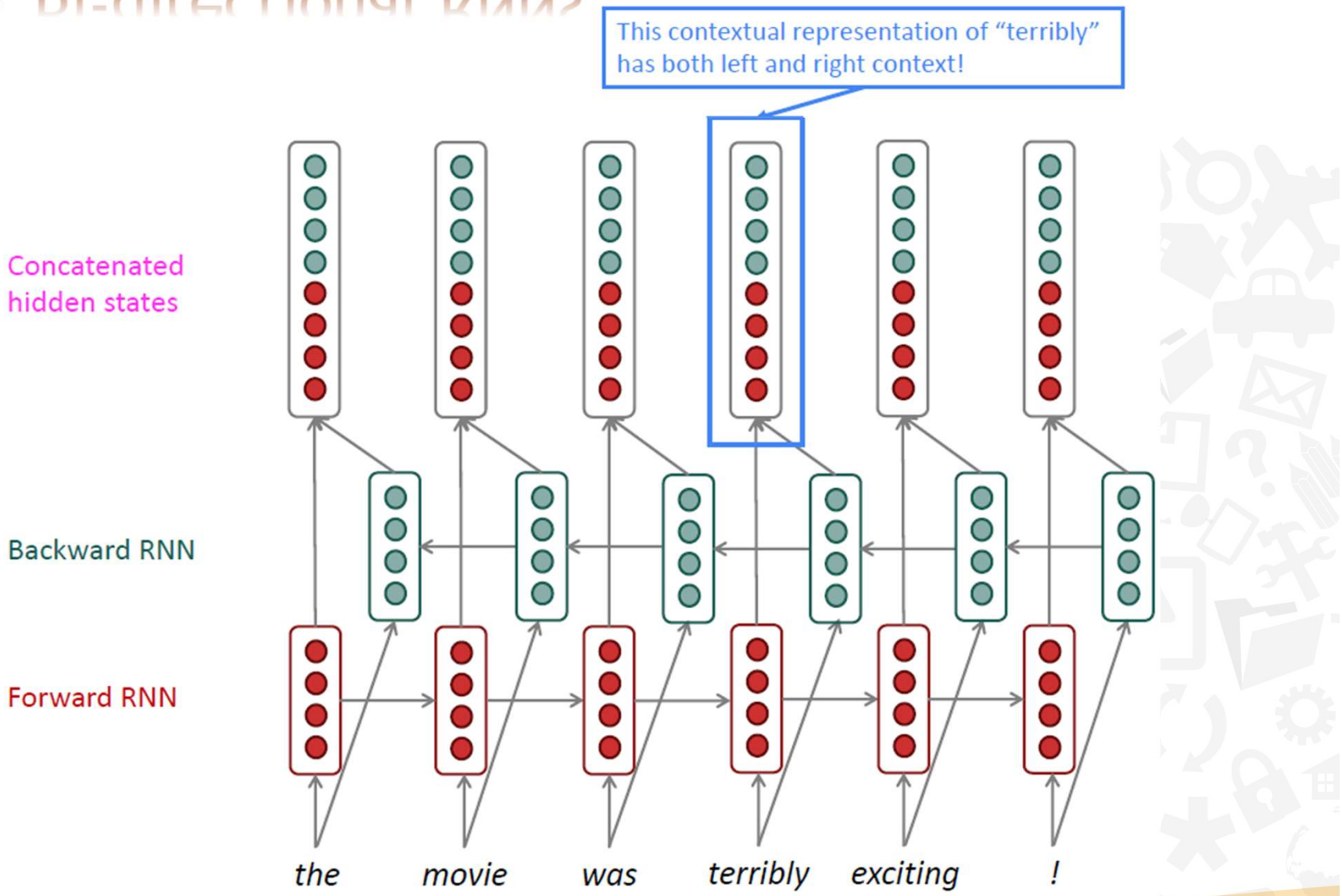
when we are looking at the word "Teddy" and the previous two words "He said", we might not be able to understand if the sentence refers to the President or Teddy bears.

Therefore, to resolve this ambiguity, we need to look ahead.



https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

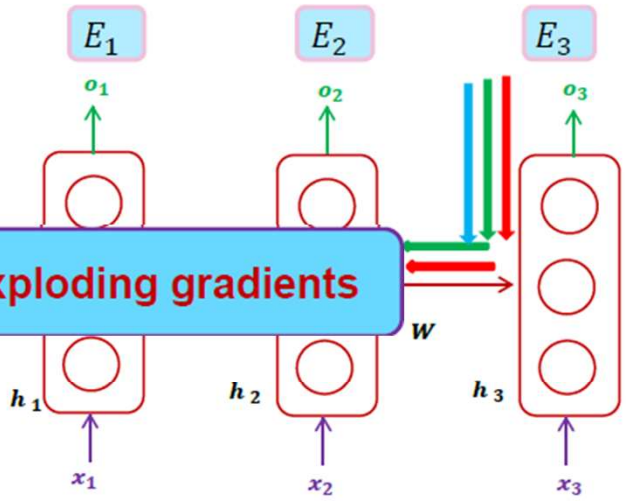
Bi-directional RNNs



Vanishing vs Exploding Gradient

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \ll\ll 1 \quad + \quad \ll 1 \quad + \quad < 1$$



Repeated matrix multiplications leads to vanishing and exploding gradients

Total Gradient = **Gradient due to long term dependencies** + **Gradient due to short term dependencies**

Remark: *The gradients due to short term dependencies (just previous dependencies) dominates the gradients due to long-term dependencies.*

This means network will tend to focus on short term dependencies which is often not desired

Problem of Vanishing Gradient

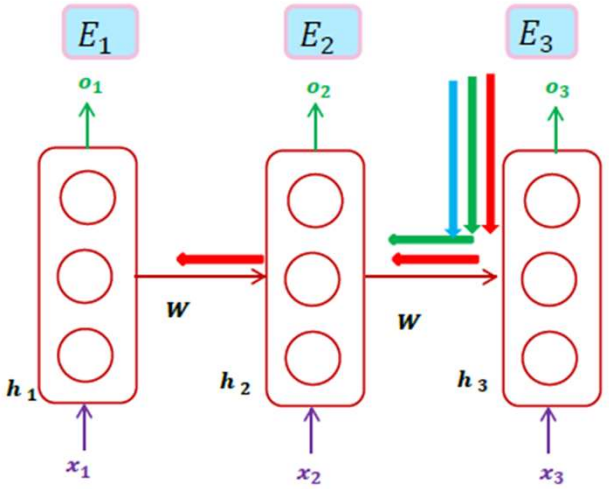
Vanishing vs Exploding Gradient

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \gg\gg 1 \quad + \quad \ggg 1 \quad + \quad \ggg 1$$

= Very large number, i.e., NaN

Problem of Exploding Gradient



Vanishing vs Exploding Gradient

$$\left\|\frac{\partial h_3}{\partial h_k}\right\| \leq (\gamma_w \gamma_g)^{t-k}$$

For tanh or linear activation

$\gamma_w \gamma_g$ **greater than** 1

Gradient *Expodes* !!!

$\gamma_w \gamma_g$ **less than** 1

Gradient *Vanishes* !!!

Remark: This problem of exploding/vanishing gradient occurs because the same number is multiplied in the gradient repeatedly.

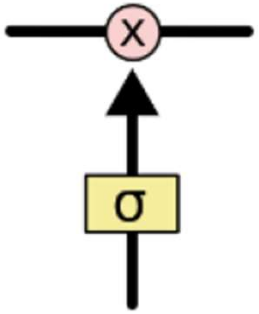
Long Short Term Memory (LSTM) : Gating

Gates :

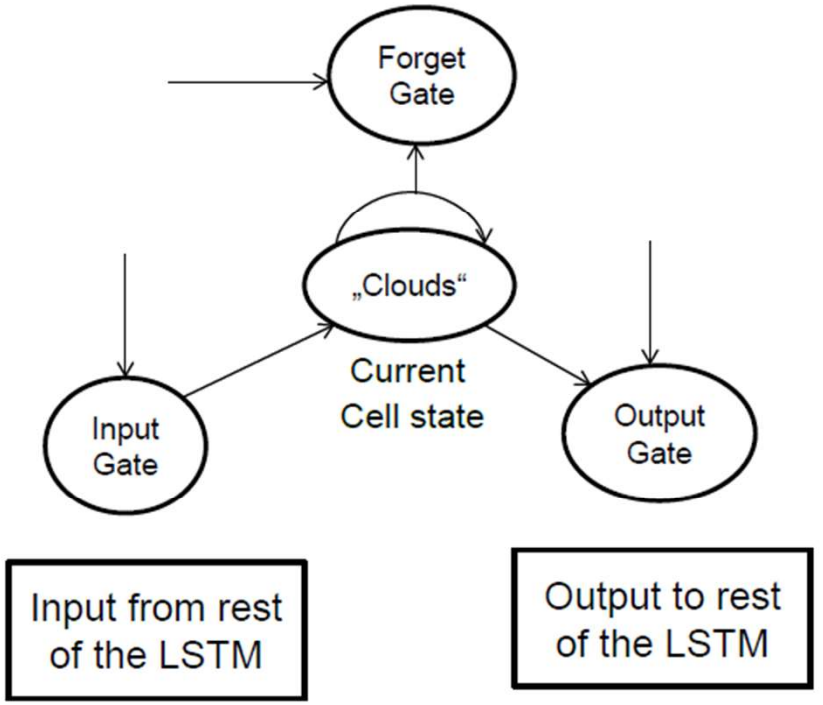
→ way to optionally let information through.

→ composed out of a sigmoid neural net layer and a pointwise multiplication operation.

→ remove or add information to the cell state



→ 3 gates in LSTM

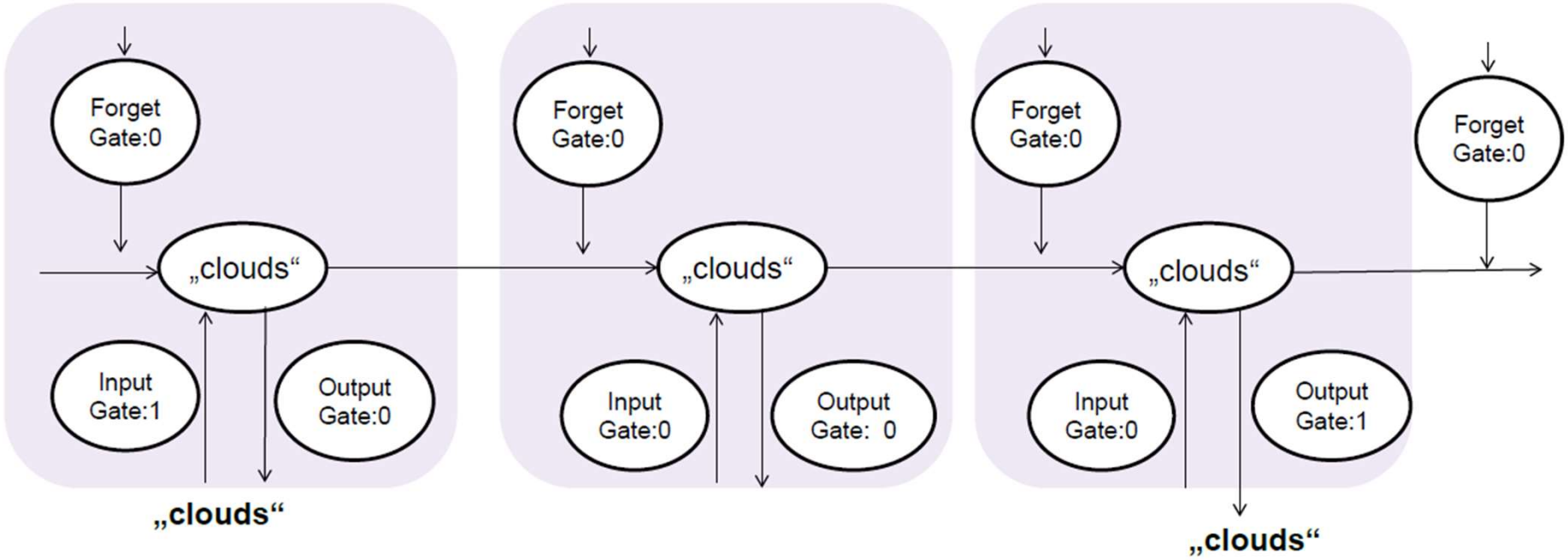


→ gates to protect and control the cell state.

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

Long Short Term Memory (LSTM) : Gating

Remember the word „ **clouds**“ over time….

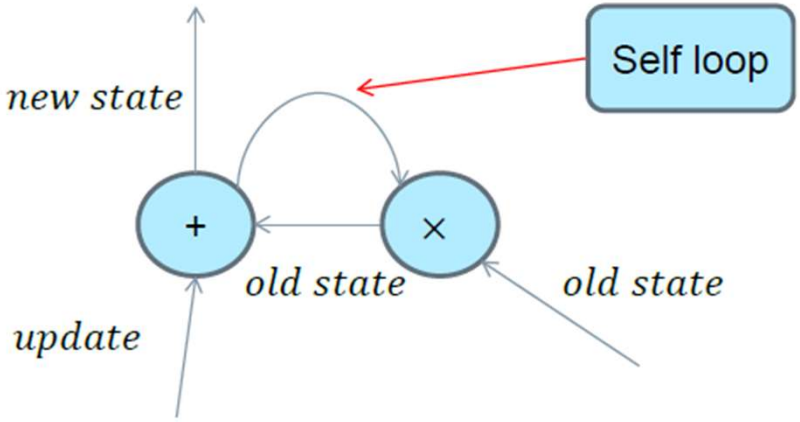


Lecture from the course Neural Networks for Machine Learning by Greff Hinton

Long Short Term Memory (LSTM) : Gating

Motivation:

- **Create a self loop path from where gradient can flow**
- **self loop corresponds to an eigenvalue of Jacobian to be slightly less than 1**



$$new\ state = old\ state + update$$

$$\frac{\partial new\ state}{\partial old\ state} \cong Identity$$

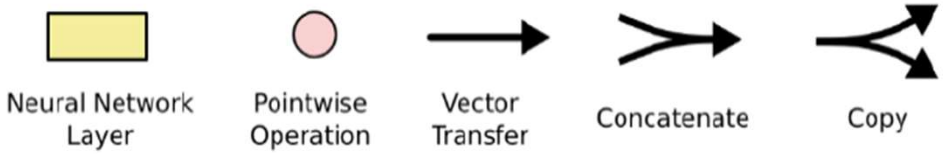
LONG SHORT-TERM MEMORY, Sepp Hochreiter and Jürgen Schmidhuber

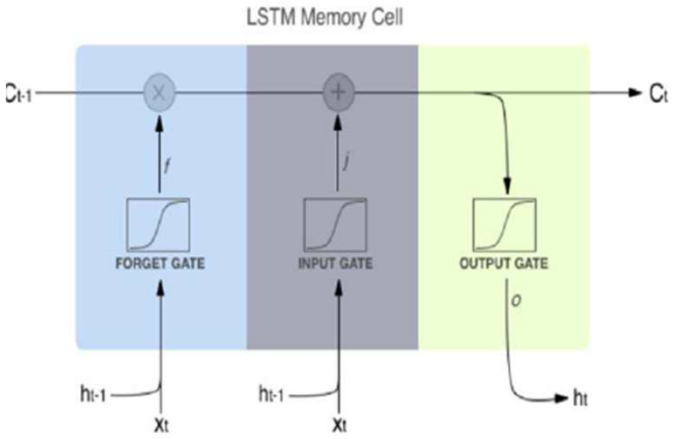
Long Short Term Memory (LSTM) : Step by Step

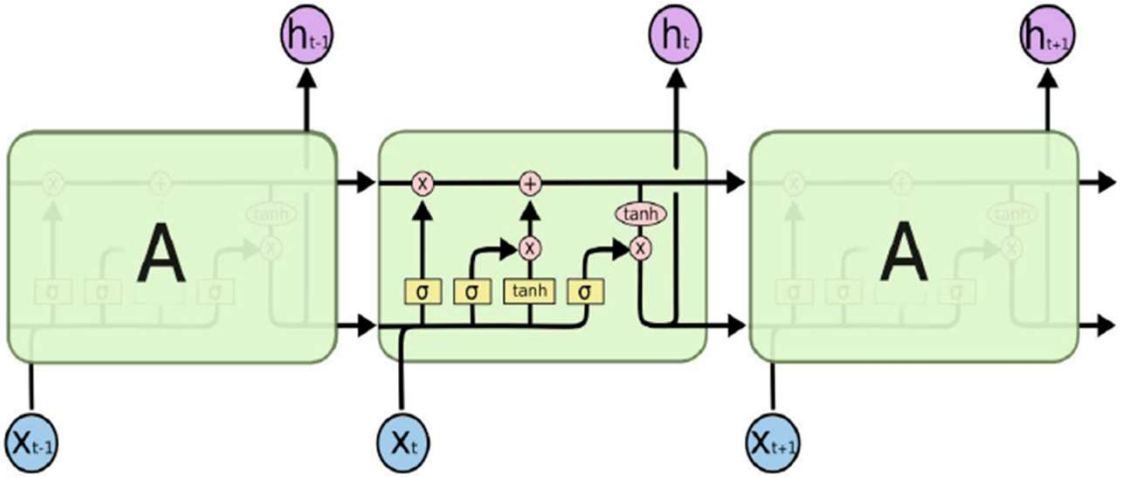
Key Ingredients

Cell state - transport the information through the units

Gates – optionally allow information passage







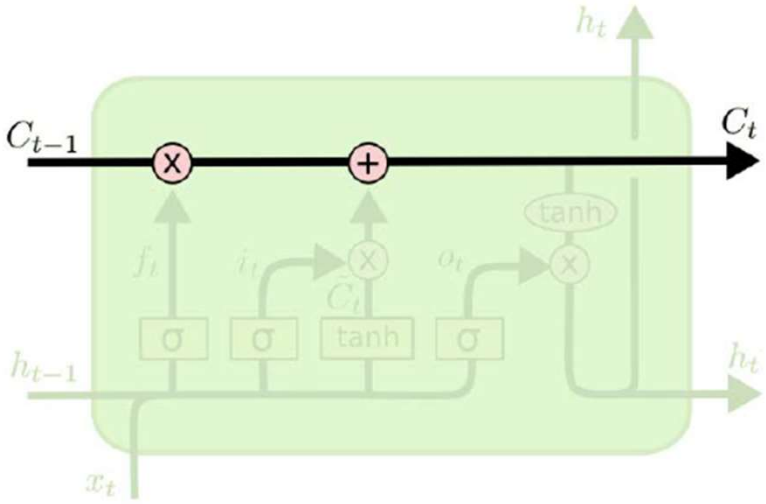
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

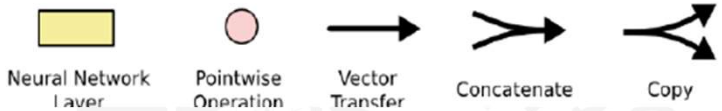
Long Short Term Memory (LSTM) : Step by Step

Cell: Transports information through the units (key idea)

→ the horizontal line running through the top

LSTM removes or adds information to the cell state using gates.





Long Short Term Memory (LSTM) : Step by Step

Forget Gate:

→ decides what information to throw away or remember from the previous cell state

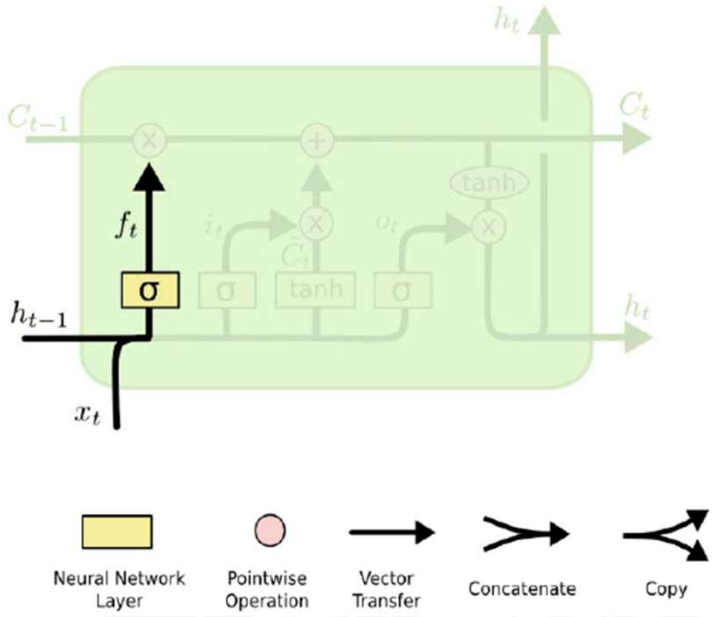
→ decision maker: sigmoid layer (*forget gate layer*)

The output of the sigmoid lies between 0 to 1,

→ 0 being forget, 1 being keep.

$$f_t = sigmoid(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$$

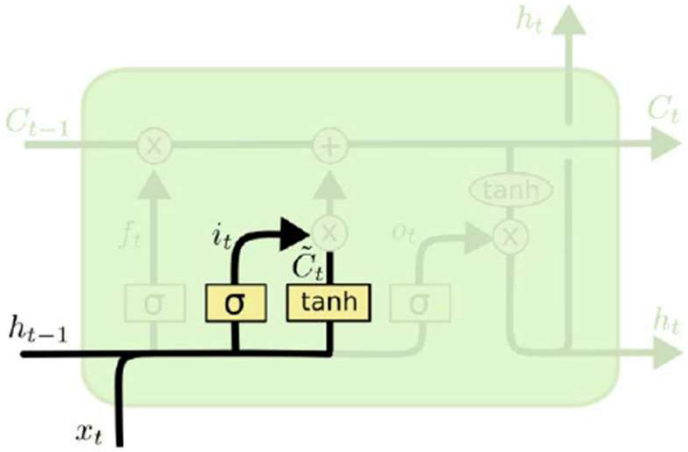
→ looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}



Long Short Term Memory (LSTM) : Step by Step

Input Gate: Selectively updates the cell state based on the new input.

A multiplicative input gate unit to protect the memory contents stored in j from perturbation by irrelevant inputs



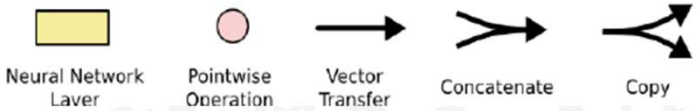
$$i_t = sigmoid(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = Tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts:

1. A sigmoid layer called the "input gate layer" decides **which values we'll update**.

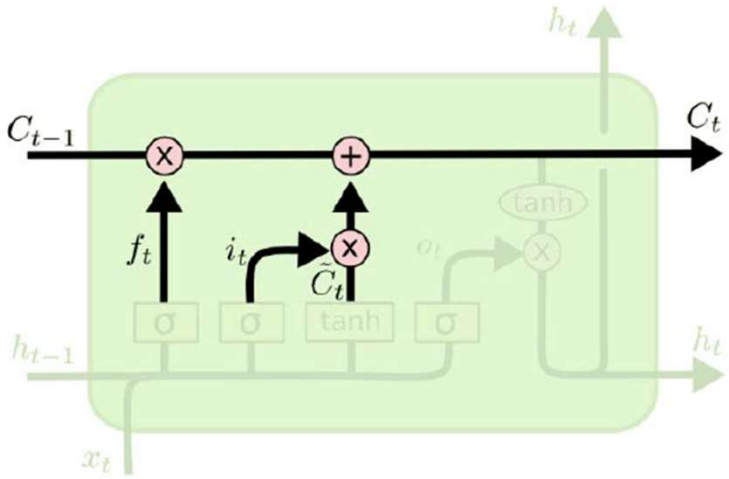
2. A tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that **could be added to the state**.

In the next step, we'll combine these two to **create an update** to the state.



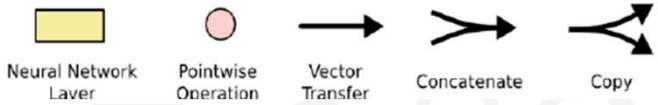
Long Short Term Memory (LSTM) : Step by Step

Cell Update



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

- update the old cell state, C_{t-1}, into the new cell state C_t
- multiply the old state by f_t, forgetting the things we decided to forget earlier
- add $i_t * \tilde{C}_t$ to get the new candidate values, scaled by how much we decided to update each state value.



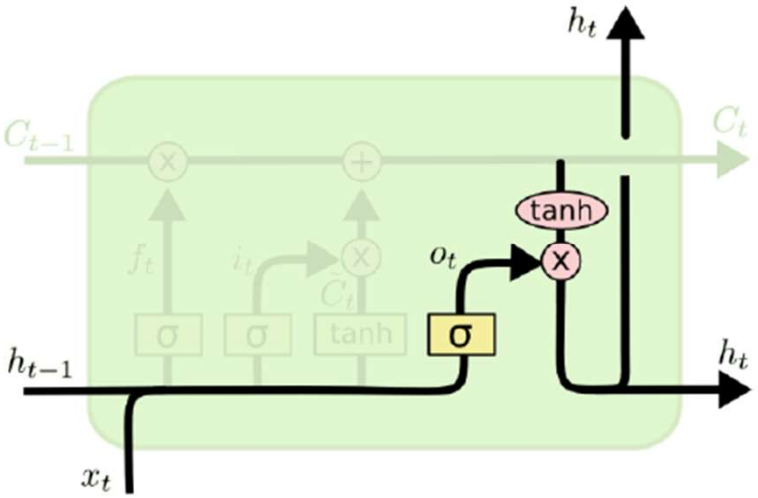
Long Short Term Memory (LSTM) : Step by Step

Output Gate: Output is the filtered version of the cell state

- Decides the part of the cell we want as our output in the form of new hidden state

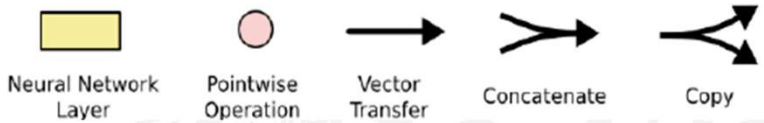
- multiplicative output gate to protect other units from perturbation by currently irrelevant memory contents

- a sigmoid layer decides what parts of the cell state goes to output. Apply tanh to the cell state and multiply it by the output of the sigmoid gate → only output the parts decided



$$o_t = sigmoid(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$$

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



Long Short Term Memory (LSTM) : Step by Step

As seen, the gradient vanishes due to the recurrent part of the RNN equations

$$h_t = W_{hh} \mathrm{h}_{t-1} + \text{some other terms}$$

How LSTM tackled vanishing gradient?

Answer: **forget gate**

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

- The forget gate parameters takes care of the vanishing gradient problem
- Activation function becomes identity and therefore, the problem of vanishing gradient is addressed.
- The derivative of the identity function is, conveniently, always one. **So if f = 1**, information from the previous cell state can pass through this step unchanged

LSTMs: real-world sucess

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

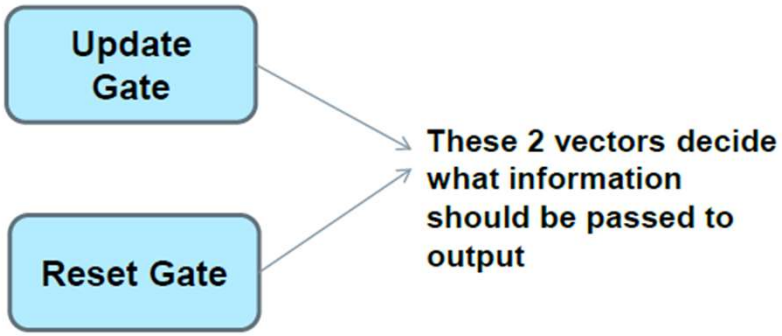
- Now (2019–2024), Transformers have become dominant for all tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
    - In WMT 2014, there were 0 neural machine translation systems (!)
    - In WMT 2016, the summary report contains "RNN" 44 times (and these systems won)
    - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

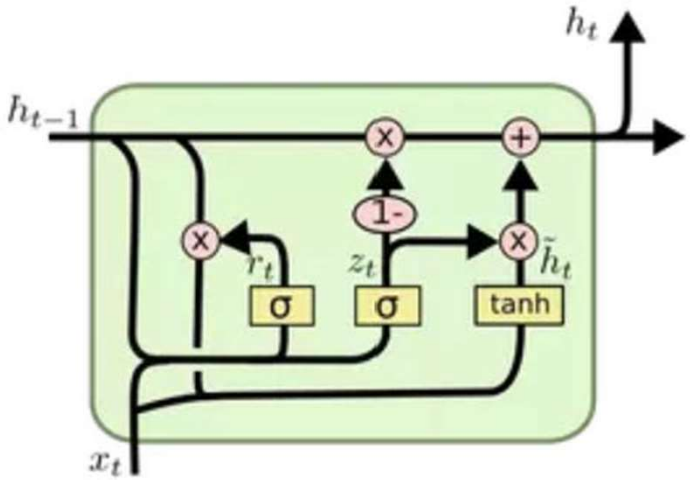
Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

Gated Recurrent Unit(GRU)

- GRU like LSTMs, attempts to solve the Vanishing gradient problem in RNN

Gates:





- Units with short-term dependencies will have active reset gates r
- Units with long term dependencies have active update gates z

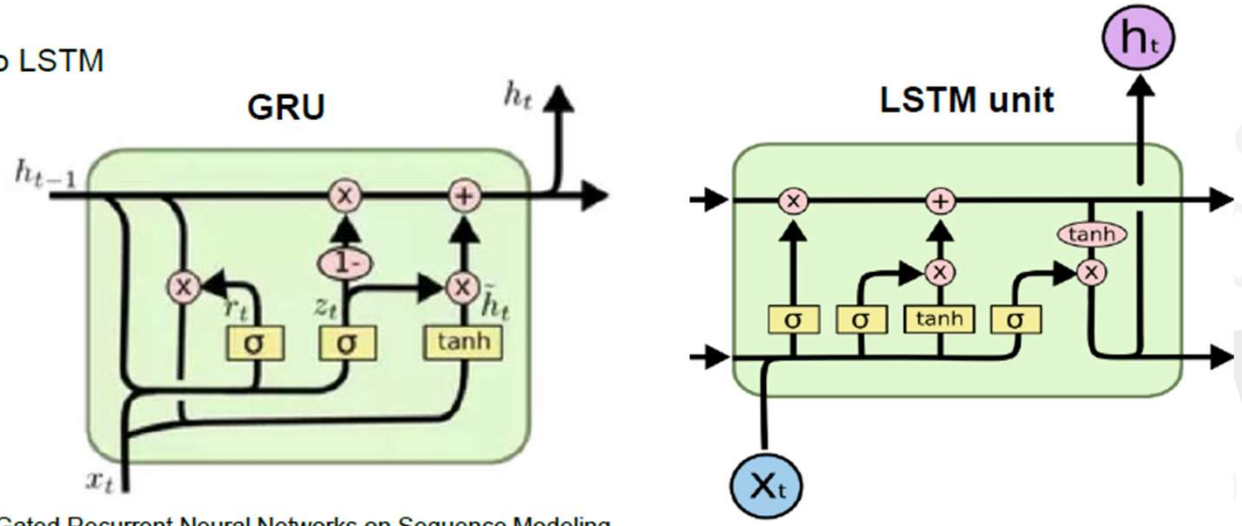
Gated Recurrent Unit(GRU)

LSTM over GRU

One feature of the LSTM has: **controlled exposure of the memory content**, not in GRU.

In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate. On the other hand **the GRU exposes its full content without any control**.

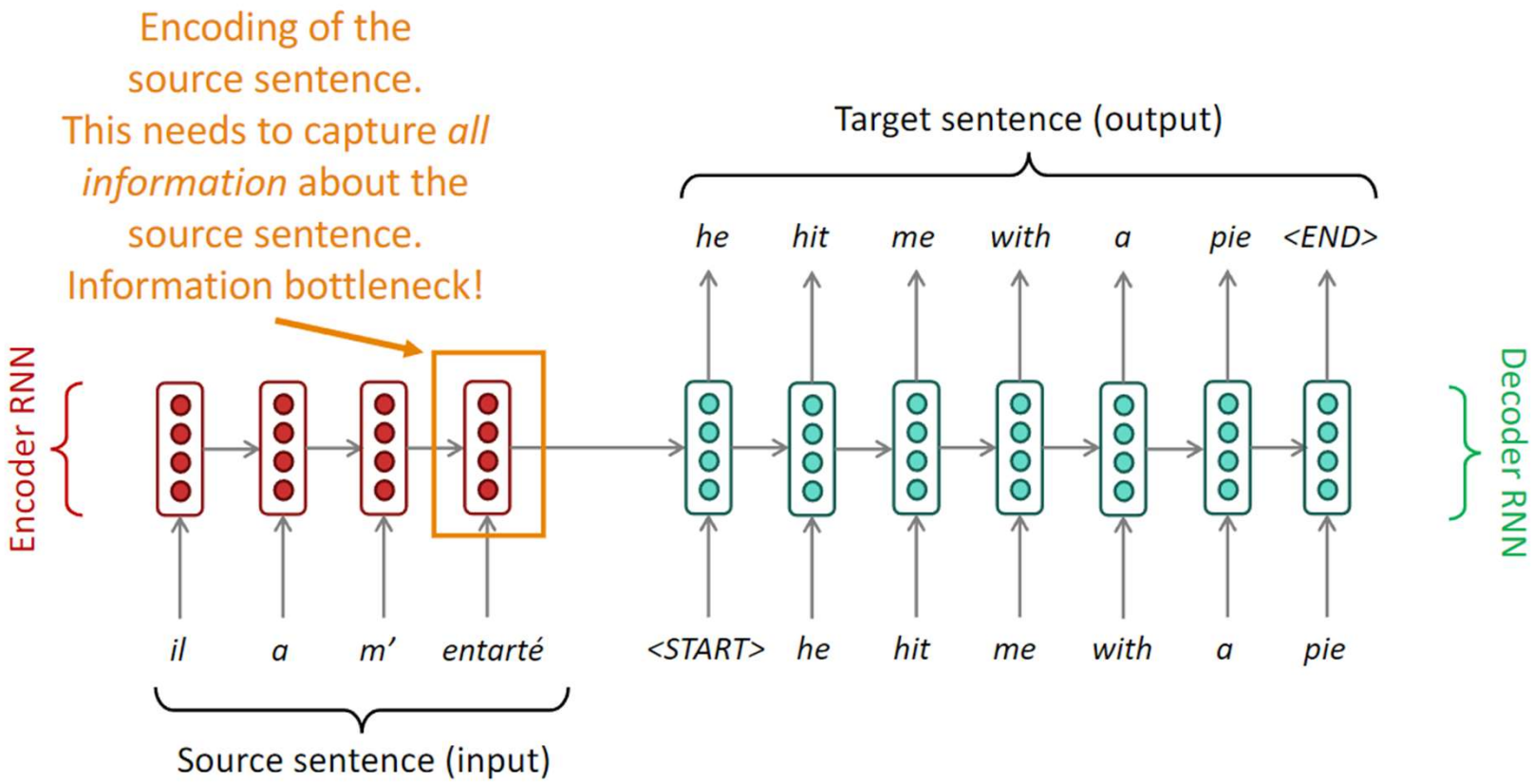
- GRU performs comparably to LSTM



Chung et al, 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Attention & Transformers

Bottleneck problem in Seq2Seq model

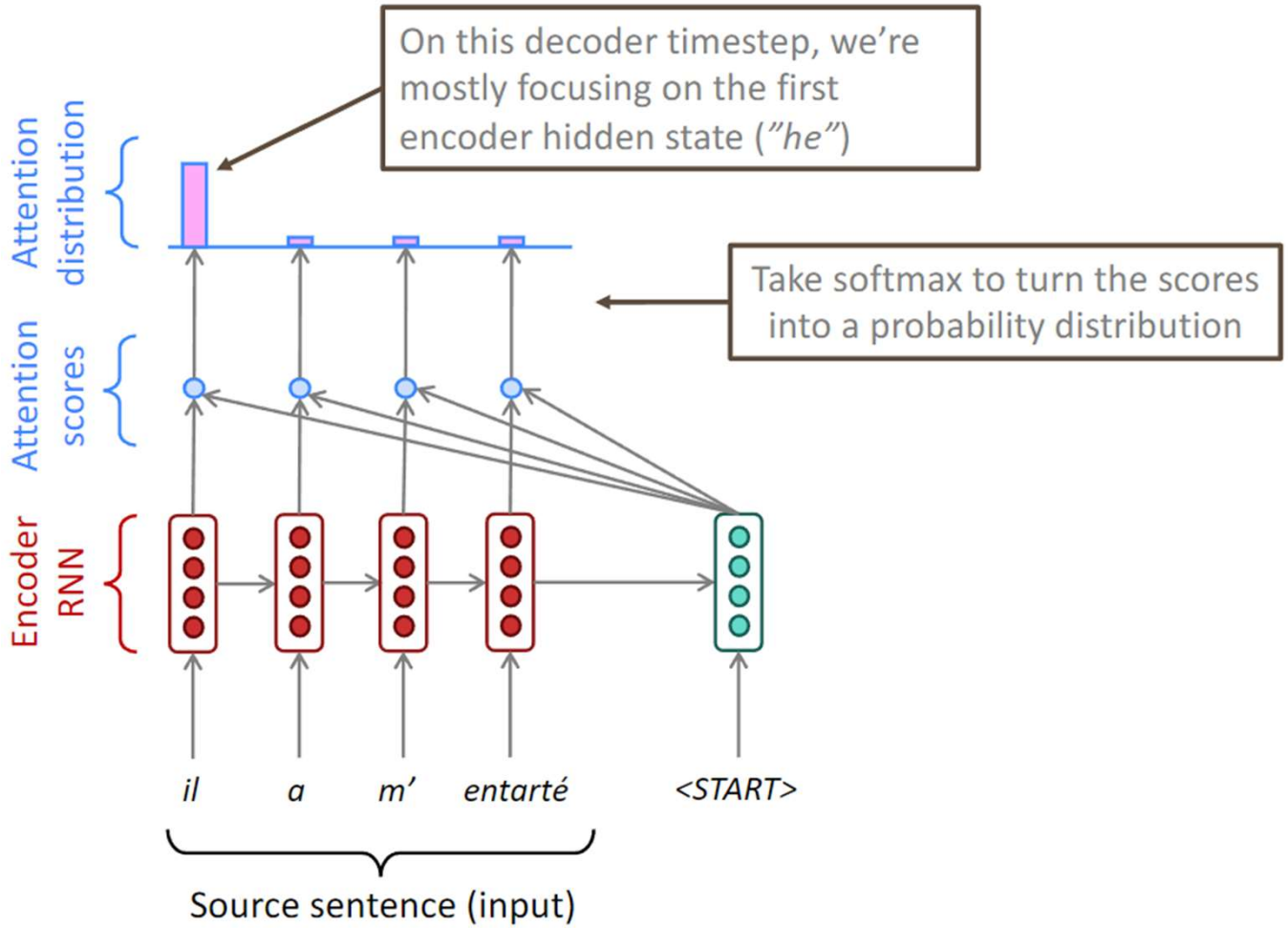


Bottleneck problem in Seq2Seq model

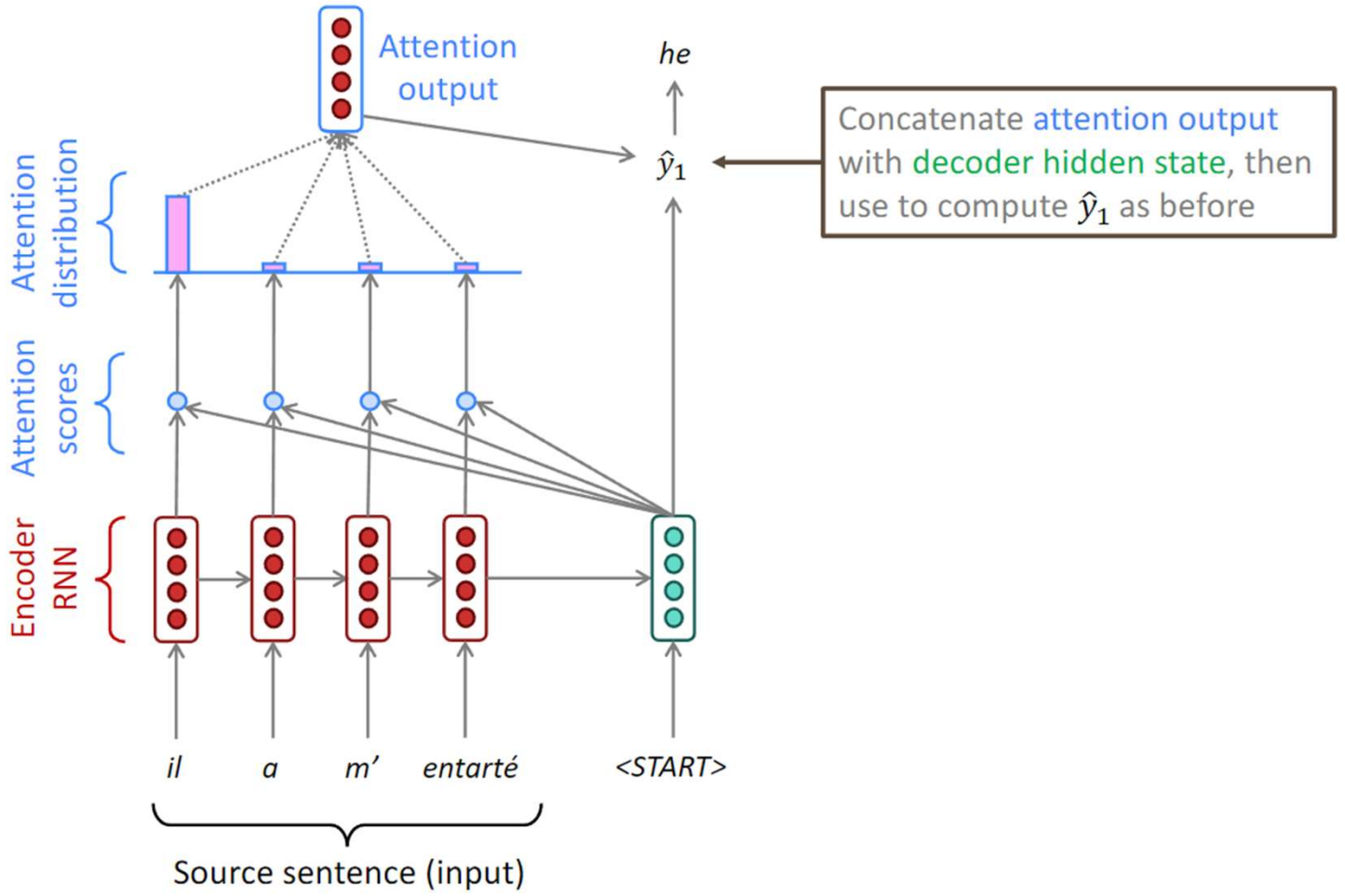
- **Attention** provides a solution to the bottleneck problem.
- **Core idea**: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First, we will show via diagram (no equations), then we will show with equations

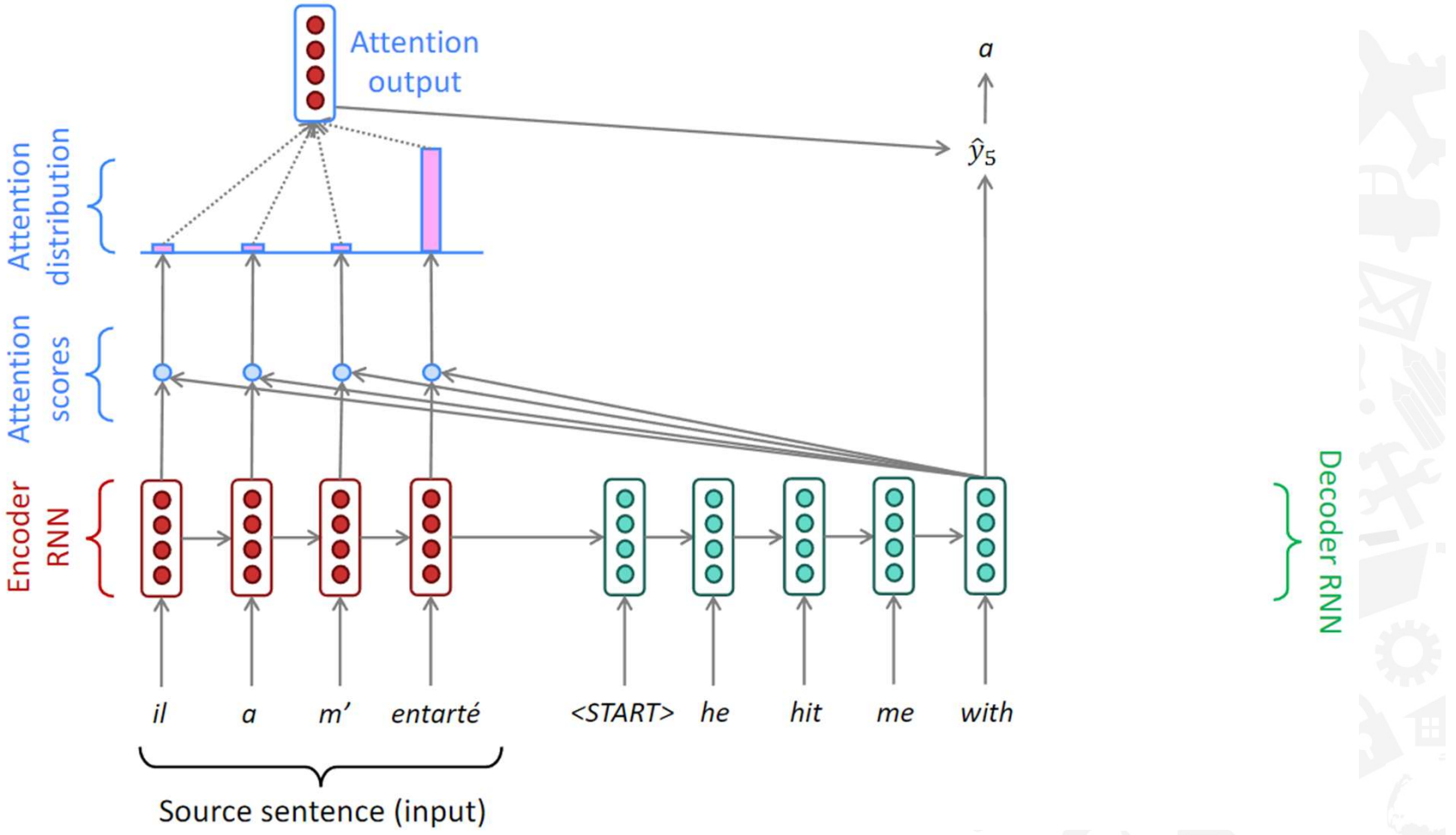
Seq2Seq with attention



Seq2Seq with attention



Seq2Seq with attention



Seq2Seq with attention

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

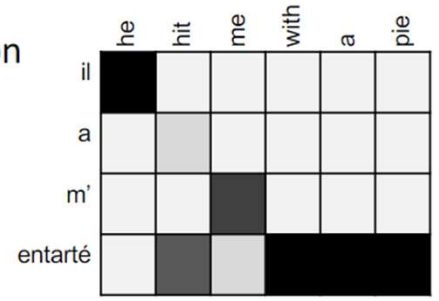
$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Attention is great!

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides a more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

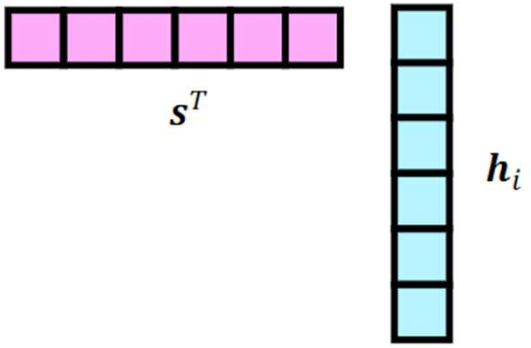


23

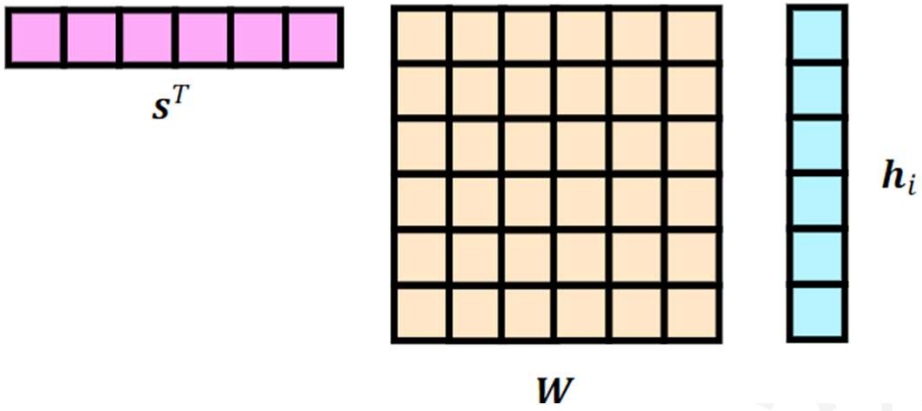
Attention Variants

There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - This assumes $d_1 = d_2$. This is the version we saw earlier.



- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$ [Luong, Pham, and Manning 2015]
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix. Perhaps better called "bilinear attention"



Attention is a general Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

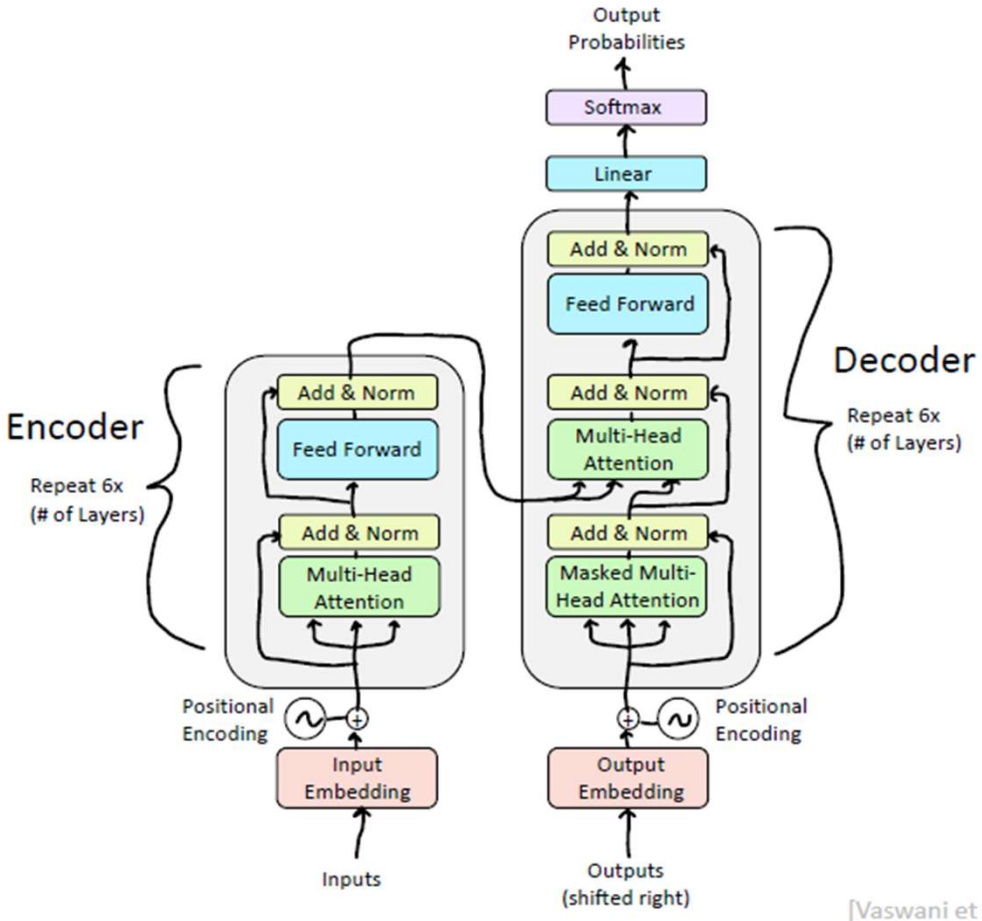
Attention is a general Deep Learning technique

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Transformers now dominate AI fields



[Vaswani et al., 2017]

Great results with Transformers: SuperGLUE

SuperGLUE is a suite of challenging NLP tasks, including question-answering, word sense disambiguation, coreference resolution, and natural language inference.

	Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
	1	JDExplore d-team	Vega v2		91.3	90.5	98.6/99.2	99.4	88.2/62.4	94.4/93.9	96.0	77.4	98.6	-0.4	100.0/50.0
+	2	Liam Fedus	ST-MoE-32B		91.2	92.4	96.9/98.0	99.2	89.6/65.8	95.1/94.4	93.5	77.7	96.6	72.3	96.1/94.1
	3	Microsoft Alexander v-team	Turing NLR v5		90.9	92.0	95.9/97.6	98.2	88.4/63.0	96.4/95.9	94.1	77.1	97.3	67.8	93.3/95.5
	4	ERNIE Team - Baidu	ERNIE 3.0		90.6	91.0	98.6/99.2	97.4	88.6/63.2	94.7/94.2	92.6	77.4	97.3	68.6	92.7/94.7
	5	Yi Tay	PaLM 540B		90.4	91.9	94.4/96.0	99.0	88.7/63.6	94.2/93.3	94.1	77.4	95.9	72.9	95.5/90.4
+	6	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4	91.4	95.8/97.6	98.0	88.3/63.0	94.2/93.5	93.0	77.9	96.6	69.1	92.7/91.9
+	7	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
	8	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	9	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
	10	SPoT Team - Google	Frozen T5 1.1 + SPoT		89.2	91.1	95.8/97.6	95.6	87.9/61.9	93.3/92.4	92.9	75.8	93.8	66.9	83.1/82.6

[Test sets: SuperGLUE Leaderboard Version: 2.0]

[Wang et al., 2019]

Great results with Transformers: Chatbot

Today, Transformer-based models dominate LMSYS Chatbot Arena Leaderboard!

Rank	Model	Arena Elo	95% CI	Votes	Organization	License	Knowledge Cutoff
1	GPT-4-Turbo-2024-04-09	1258	+4/-4	26444	OpenAI	Proprietary	2023/12
1	GPT-4-1106-preview	1253	+3/-3	68353	OpenAI	Proprietary	2023/4
1	Claude 3 Opus	1251	+3/-3	71500	Anthropic	Proprietary	2023/8
2	Gemini 1.5 Pro API-0409-Preview	1249	+4/-5	22211	Google	Proprietary	2023/11
3	GPT-4-0125-preview	1248	+2/-3	58959	OpenAI	Proprietary	2023/12
6	Meta Llama 3 70b Instruct	1213	+4/-6	15809	Meta	Llama 3 Community	2023/12
6	Bard (Gemini Pro)	1208	+7/-6	12435	Google	Proprietary	Online
7	Claude 3 Sonnet	1201	+4/-2	73414	Anthropic	Proprietary	2023/8

Gemini/Bard (Google)

ChatGPT/GPT-4 (OpenAI)

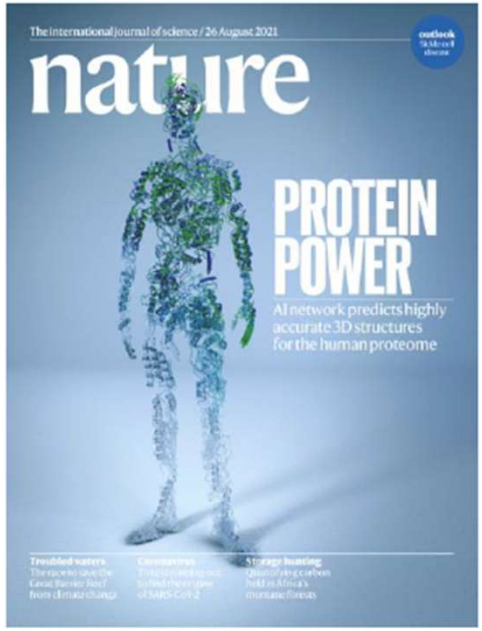
Claude3 (Anthropic AI)

Llama 3 (Meta)

Qwen 2.5 (Alibaba Cloud)

Transformers are even promising outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

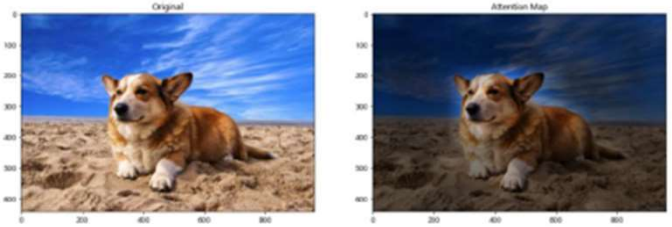
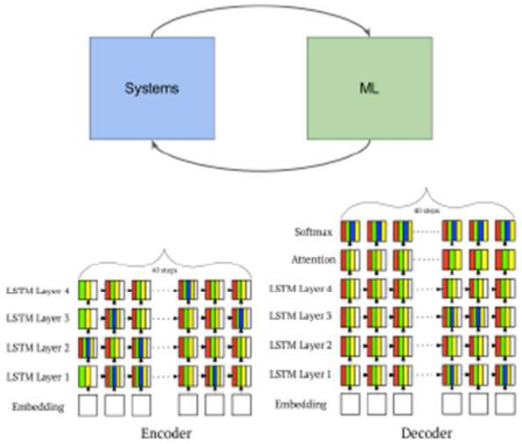


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k



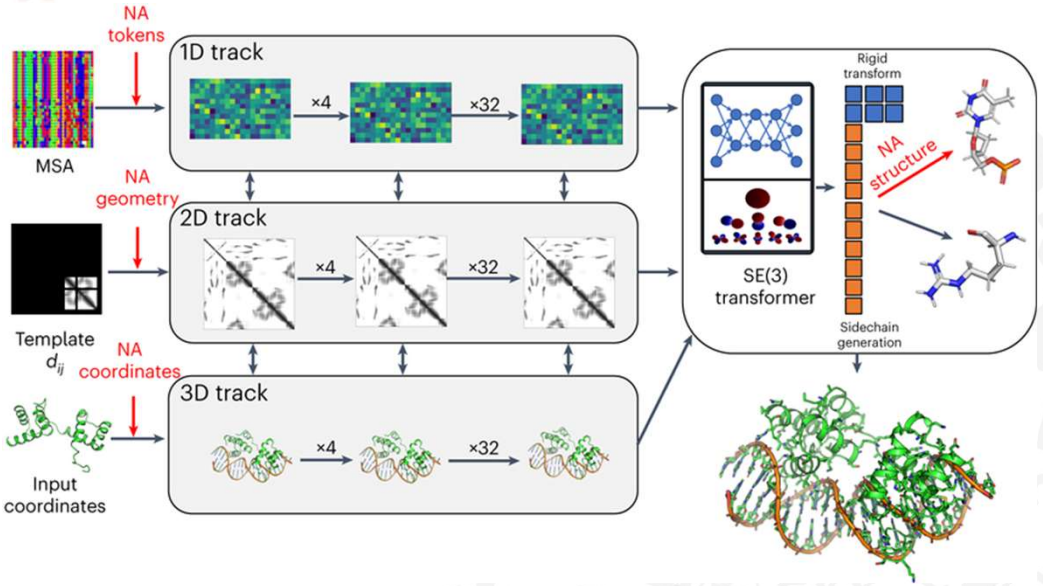
ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

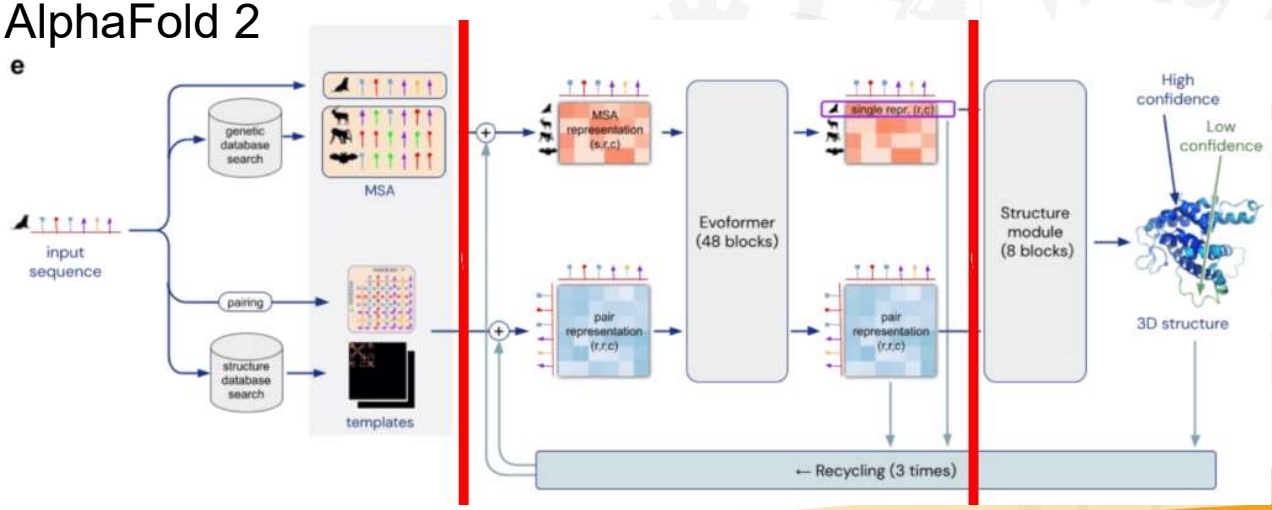
Model (8devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Transformers are even promising outside of NLP

RosettaFold

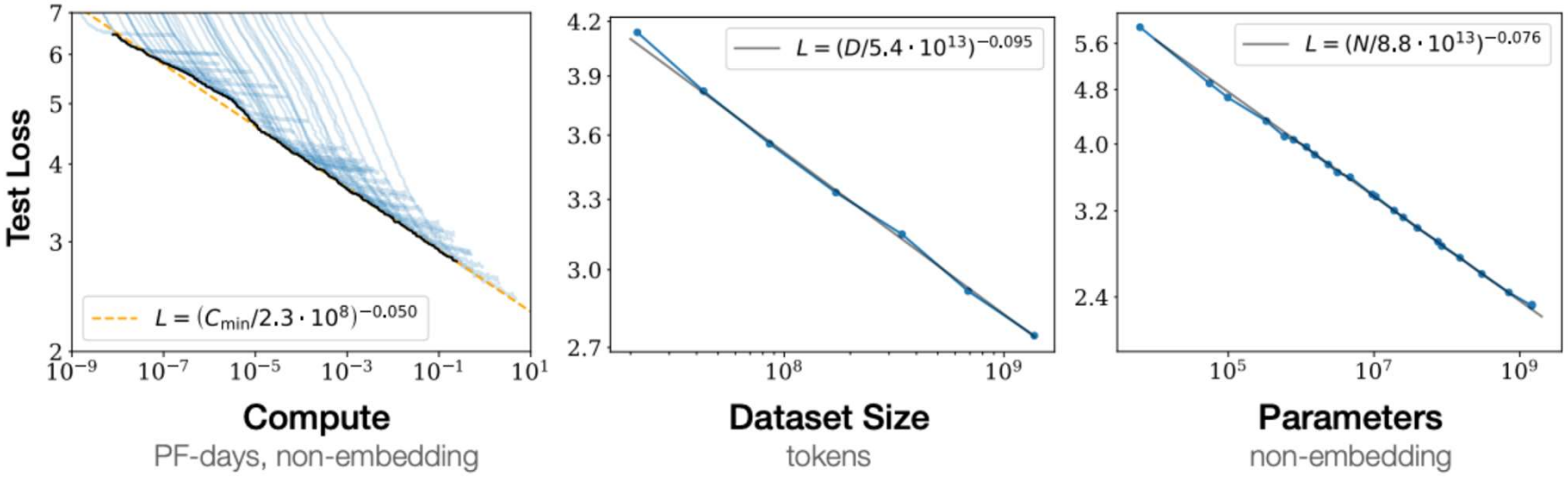


AlphaFold 2



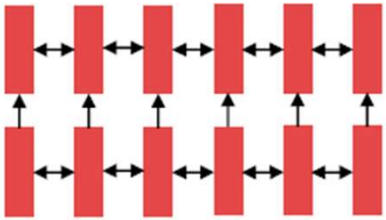
Scaling laws: Are Transformers All we need?

- With Transformers, language modeling performance improves smoothly as we increase model size, training data, and compute resources in tandem.
- This power-law relationship has been observed over multiple orders of magnitude with no sign of slowing!
- If we keep scaling up these models (with no change to the architecture), could they eventually match or exceed human-level performance?

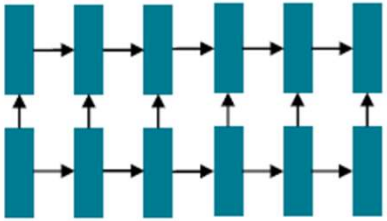


RNN for (most NLP)

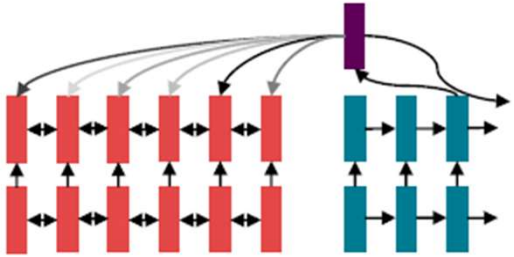
- Circa 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM: (for example, the source sentence in a translation)



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.



- Use attention to allow flexible access to memory



Why Move Beyond Recurrence? Motivation for Transformer Architecture

The Transformers authors had 3 desirata when designing this architecture:

1. Minimize (or at least not increase) computational complexity per layer.
2. Minimize path length between any pair of words to facilitate learning of long-range dependencies.
3. Maximize the amount of computation that can be parallelized.

1. Transformer Motivation: Computational Complexity Per Layer

When sequence length (n) << representation dimension (d), complexity per layer is lower for a Transformer compared to the recurrent models we've learned about so far.

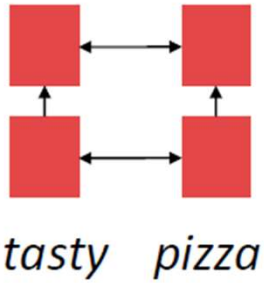
Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

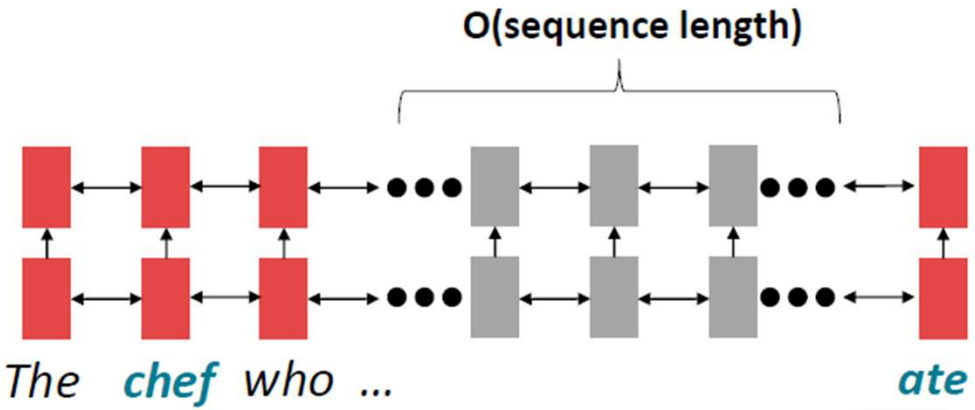
Table 1 of the Transformer paper.

2. Transformer Motivation: Minimize Linear Interaction Distance

- RNNs are unrolled "left-to-right".
- It encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings

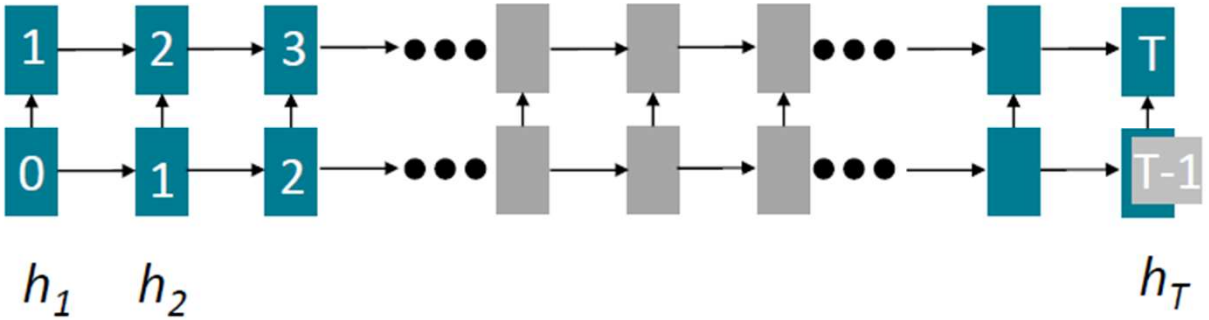


- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



3. Transformer Motivation: Maximize Parallelizability

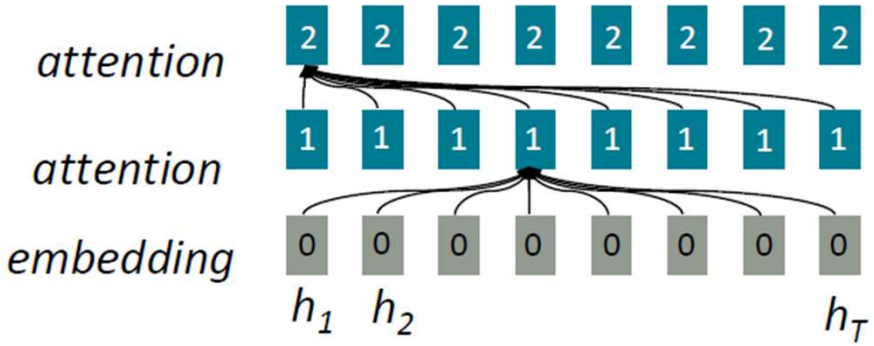
- Forward and backward passes have **O(seq length)** unparallelizable operations
  - GPUs (and TPUs) can perform many independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!
  - Particularly problematic as sequence length increases, as we can no longer batch many examples together due to memory limitations



Numbers indicate min # of steps before a state can be computed

High-level Architecture: transformer is all about (self) attention

- To recap, **attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - Last lecture, we saw attention from the **decoder** to the **encoder** in a recurrent sequence-to-sequence model
  - **Self-attention** is **encoder-encoder** (or **decoder-decoder**) attention where each word attends to each other word **within the input (or output)**.

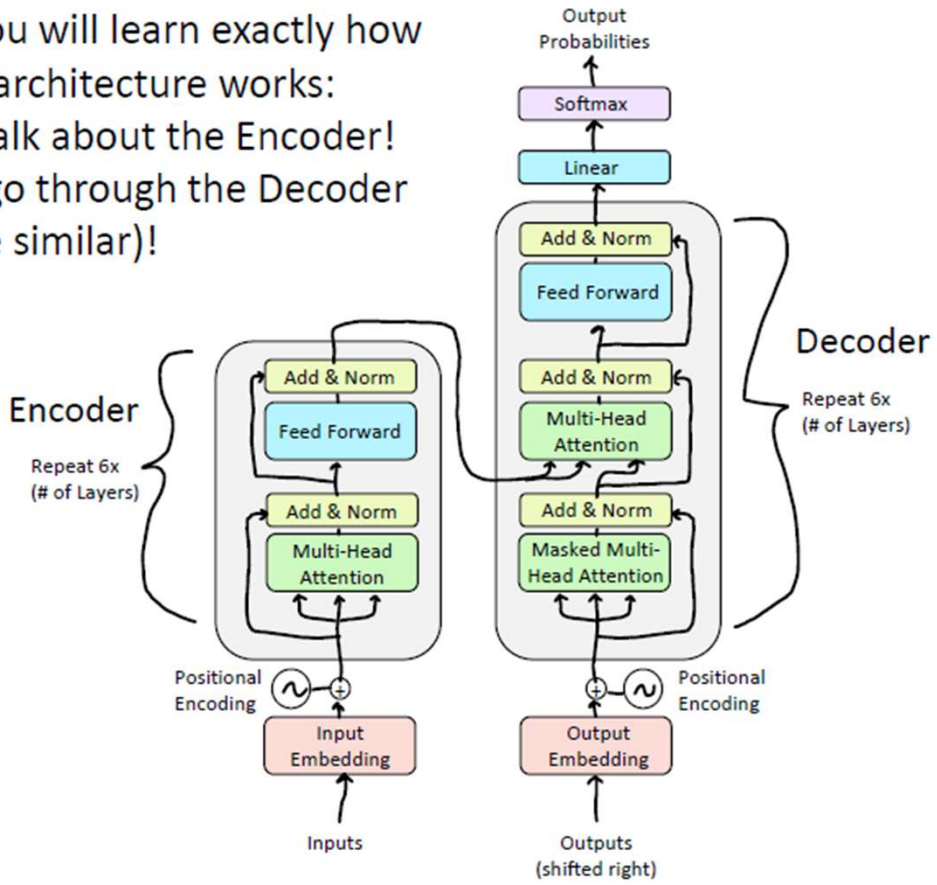


All words attend to all words in previous layer; most arrows here are omitted

The Transformer Encoder-Decoder[VASWANI et al., 2017]

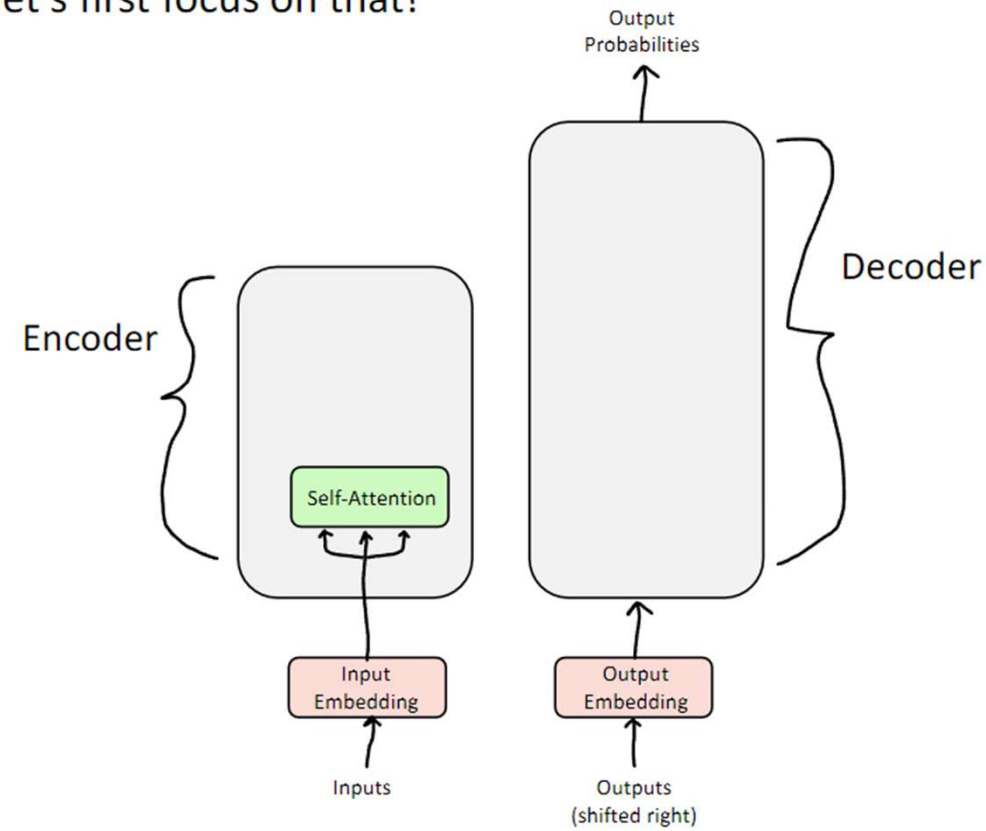
In this section, you will learn exactly how the Transformer architecture works:

- First, we will talk about the Encoder!
- Next, we will go through the Decoder (which is quite similar)!



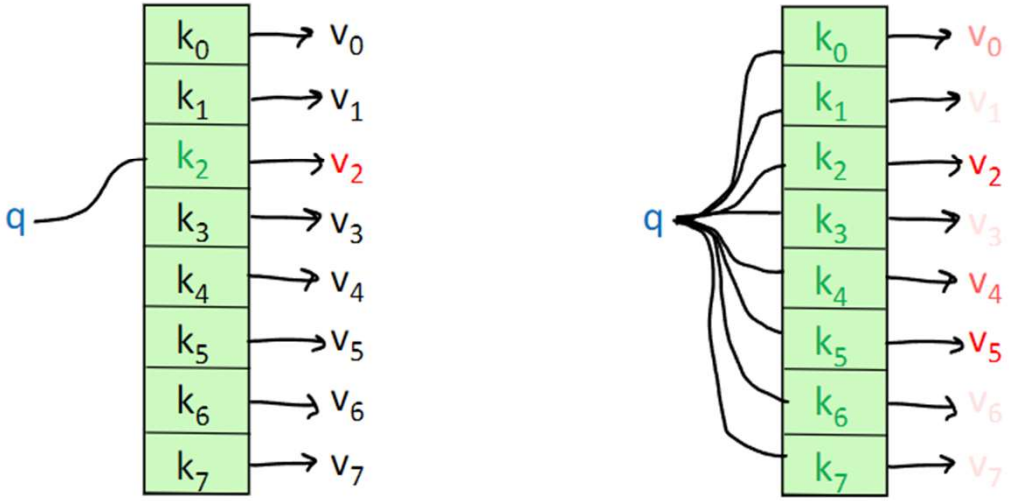
Encoder: Self-Attention

Self-Attention is the core building block of Transformer, so let's first focus on that!



Encoder: Self-Attention

- Let's think of attention as a "fuzzy" or approximate hashtable:
  - To look up a **value**, we compare a **query** against **keys** in a table.
  - In a hashtable (shown on the bottom left):
    - Each **query** (hash) maps to exactly one **key-value** pair.
  - In (self-)attention (shown on the bottom right):
    - Each **query** matches each **key** to varying degrees.
    - We return a sum of **values** weighted by the **query-key** match.



Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**.

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query** and **keys**.

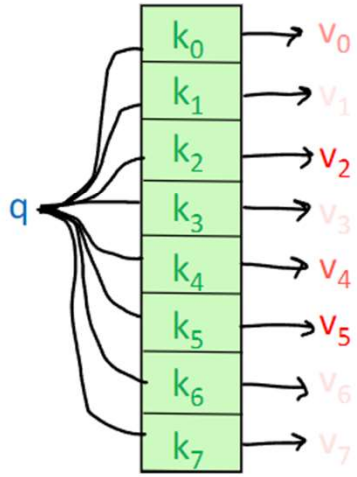
$$e_{ij} = q_i \cdot k_j$$

- Step 3: Take the softmax to normalize attention scores.

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

$$Output_i = \sum_j \alpha_{ij} v_j$$



Recipe for Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in X, calculate **queries**, **keys**, and **values**.

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

- Step 2: Calculate attention scores between **query** and **keys**.

$$E = QK^T$$

- Step 3: Take the softmax to normalize attention scores.

$$A = softmax(E)$$

- Step 4: Take a weighted sum of **values**.

$$Output = AV$$

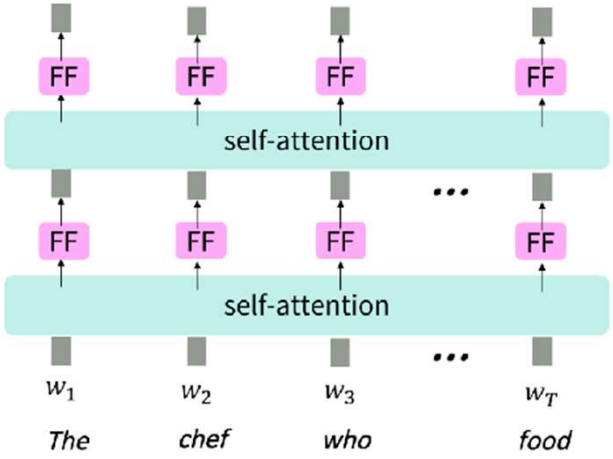
$$Output = softmax\left(QK^T\right)V$$

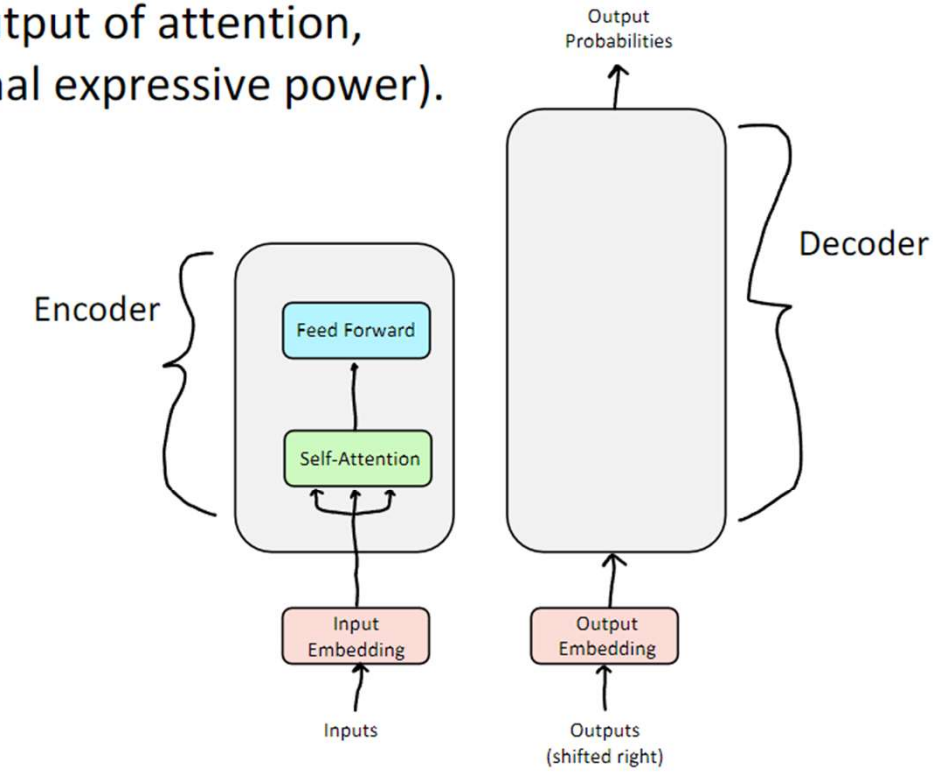
But attention isn't quite all you need!

- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

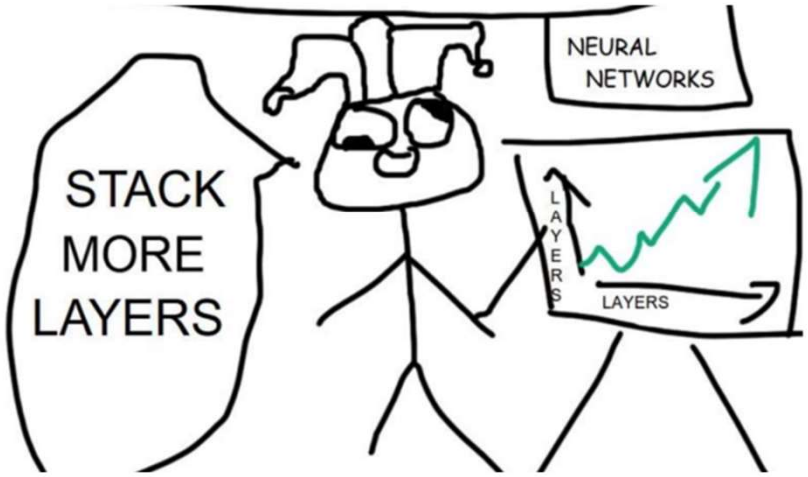
Equation for Feed Forward Layer

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$

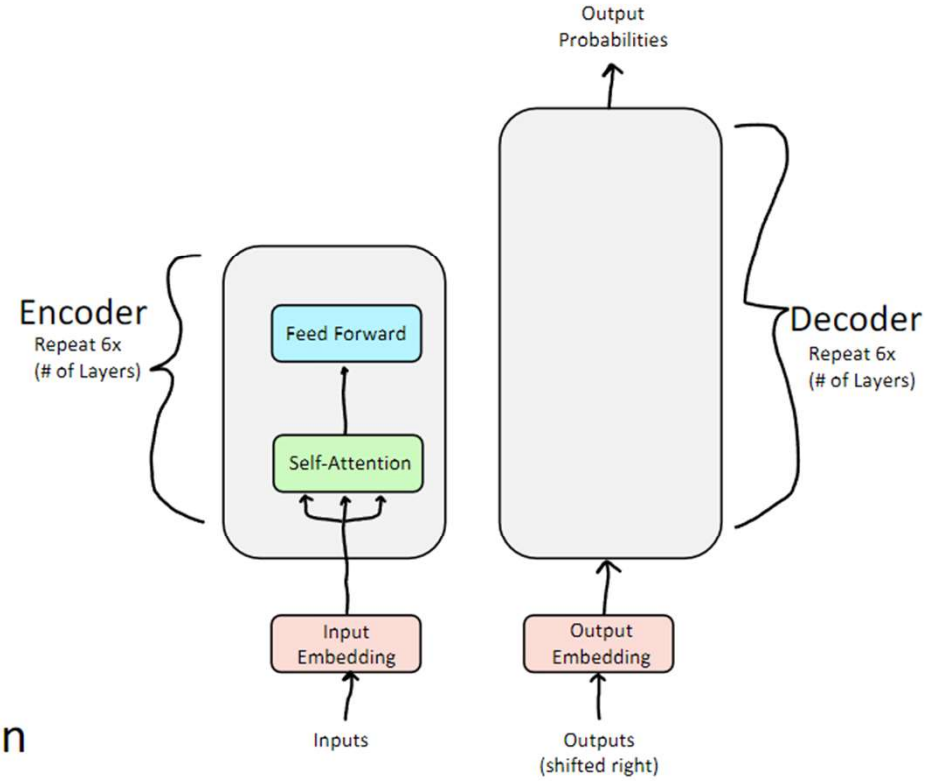




But how do we make this work for deep networks?



Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention

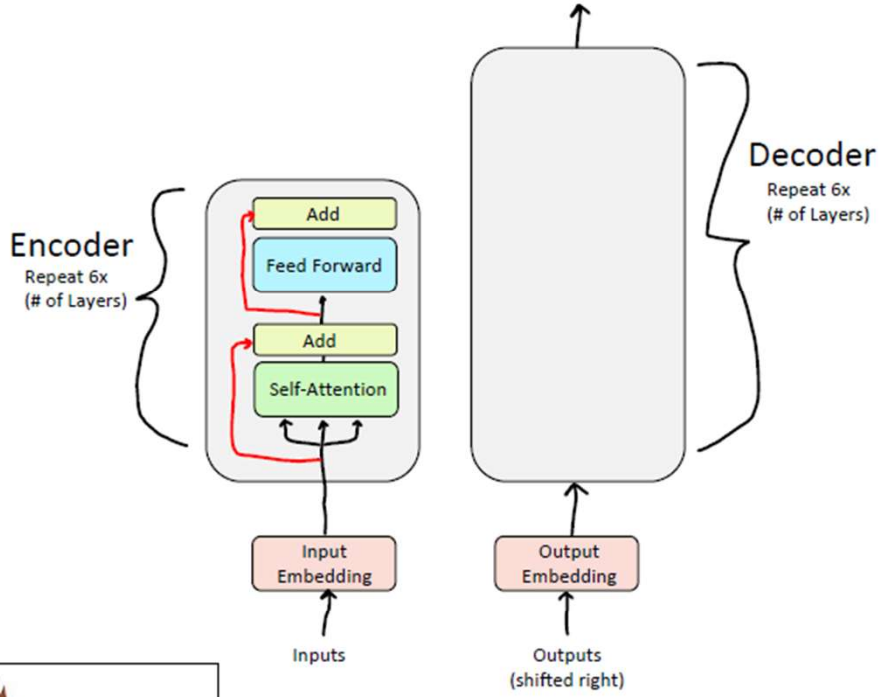


Training Trick #1: Residual Connections [He et al., 2016]

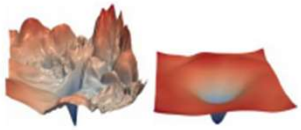
- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



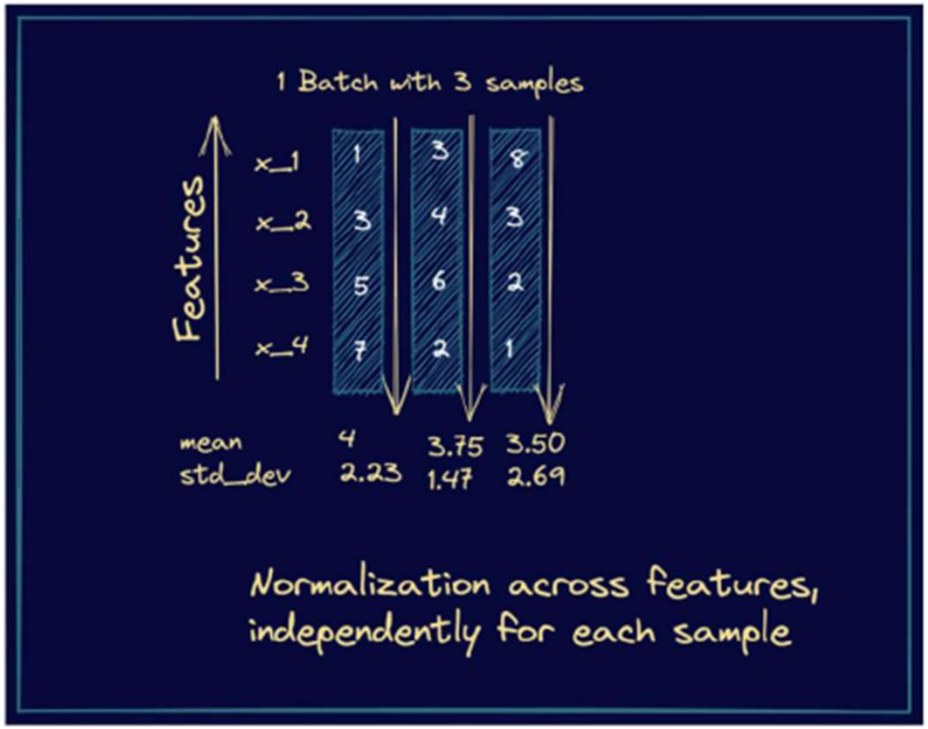
Residual connections are also thought to smooth the loss landscape and make training easier!



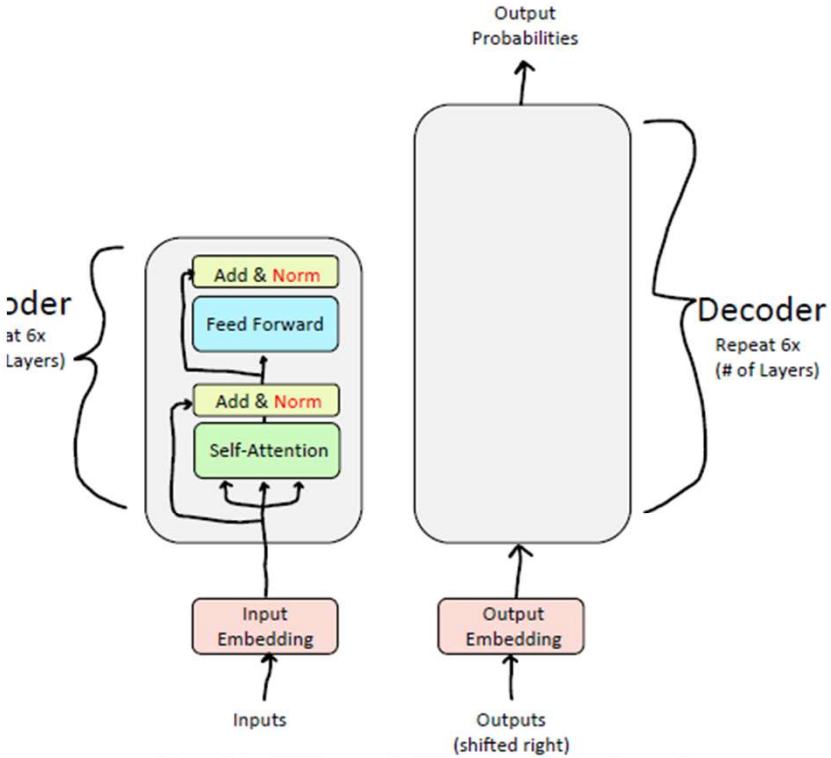
[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Training Trick #2: Layer Normalization
[Ba et al., 2016]



An Example of How LayerNorm Works (Image by Bala Priya C, Pinecone)

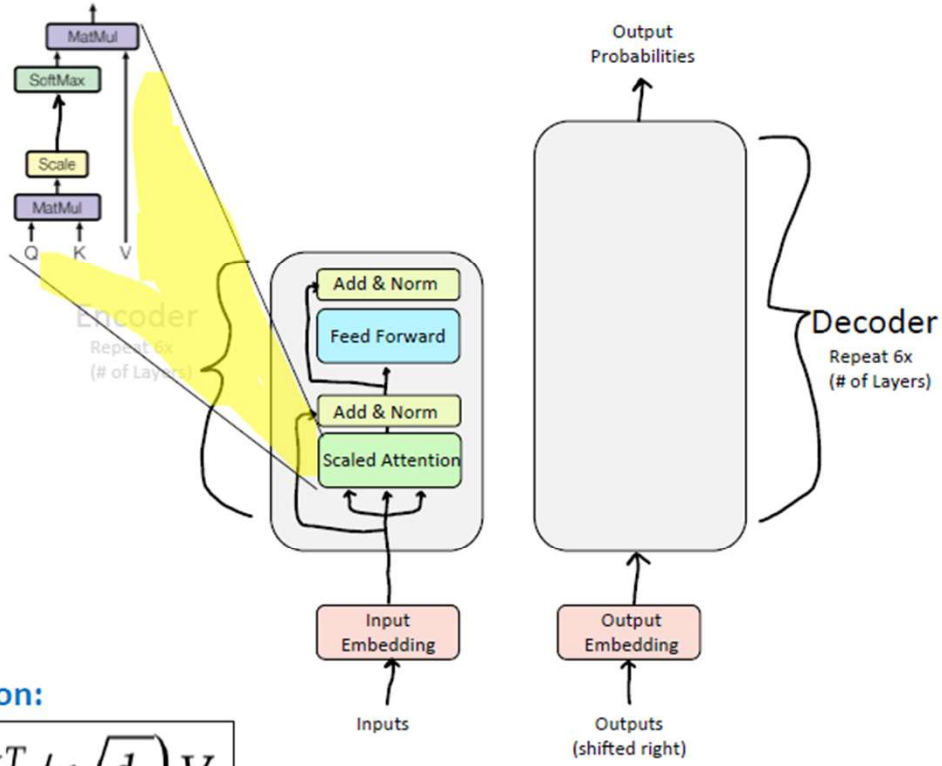


Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!



Updated Self-Attention Equation:

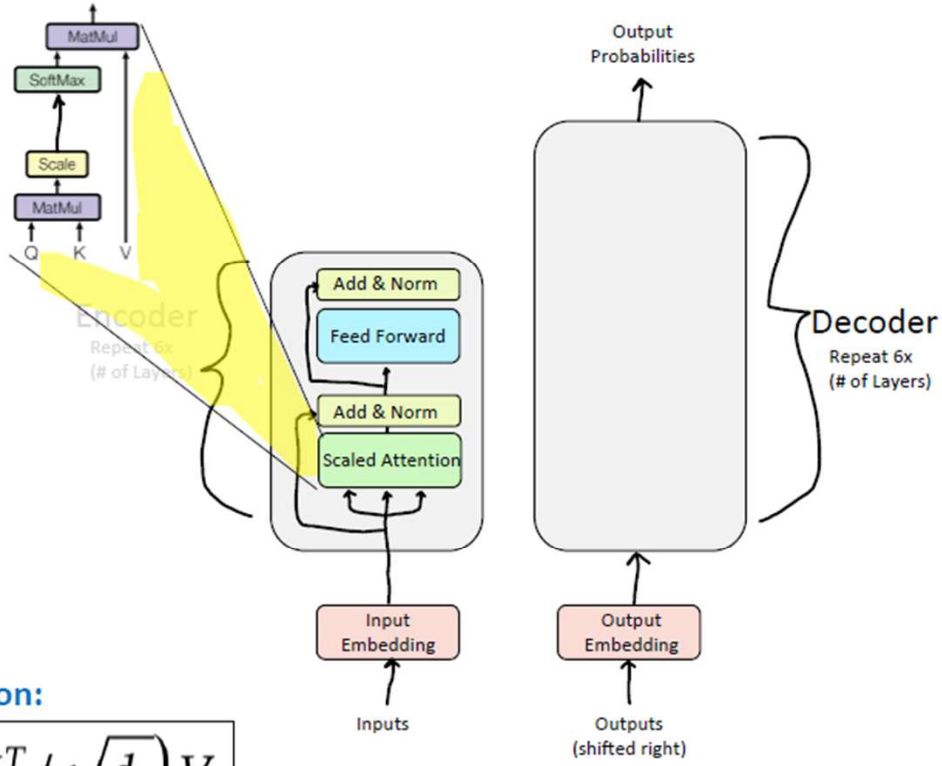
$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!



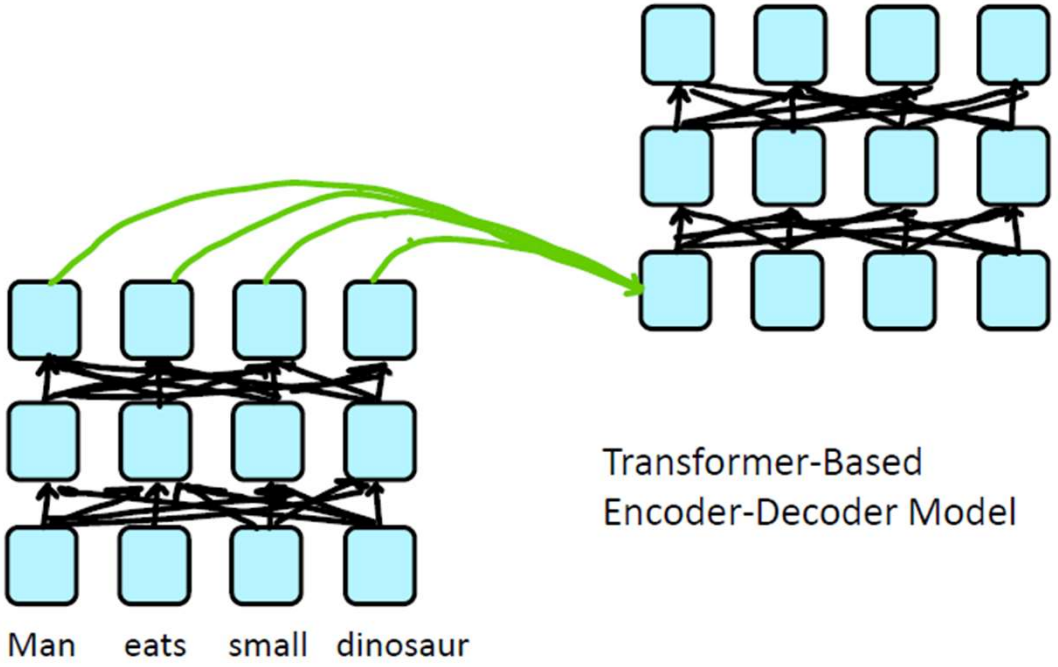
Updated Self-Attention Equation:

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

Major Issue!

- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
  - "Man eats small dinosaur."
- Wait a minute, order doesn't impact the network at all!
- This seems wrong given that word order does have meaning in many languages, including English!

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$



Fixing the first self-attention problem: sequence order

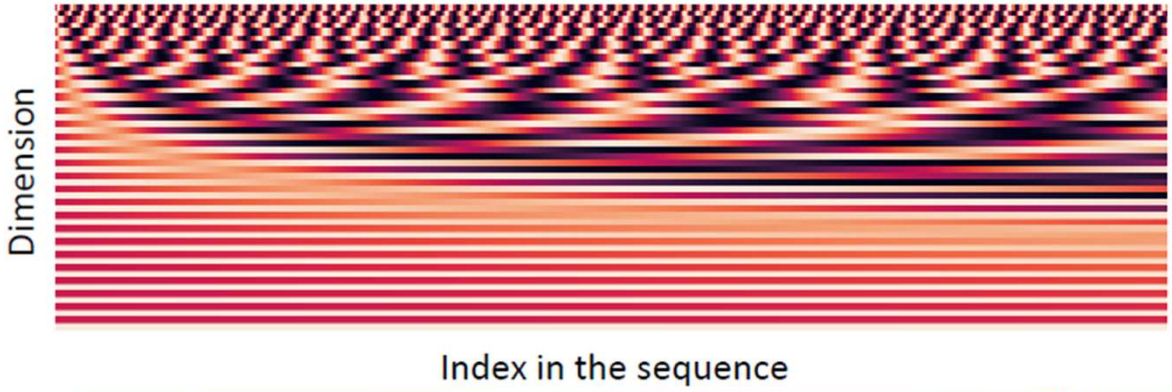
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i$ $\tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

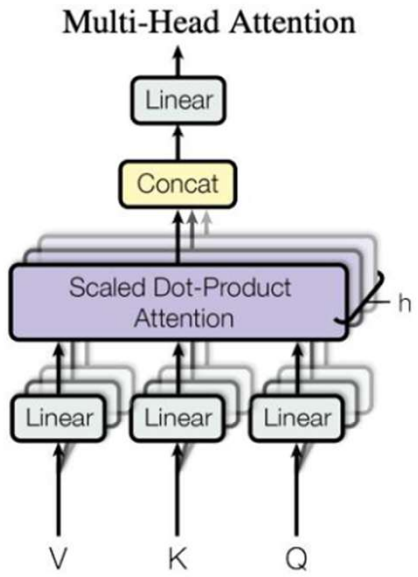
- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Multi-head self-attention: k heads are better than 1!

- **High-Level Idea:** Let's perform self-attention multiple times in parallel and combine the results.

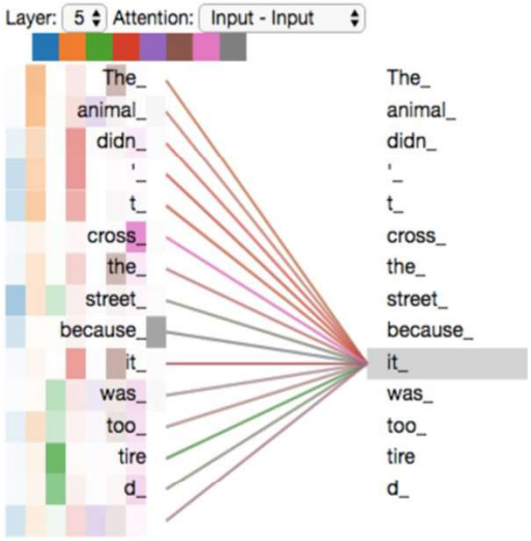


[Vaswani et al. 2017]

Wizards of the Coast, Artist: Todd Lockwood

Multi-head self-attention: k heads are better than 1!

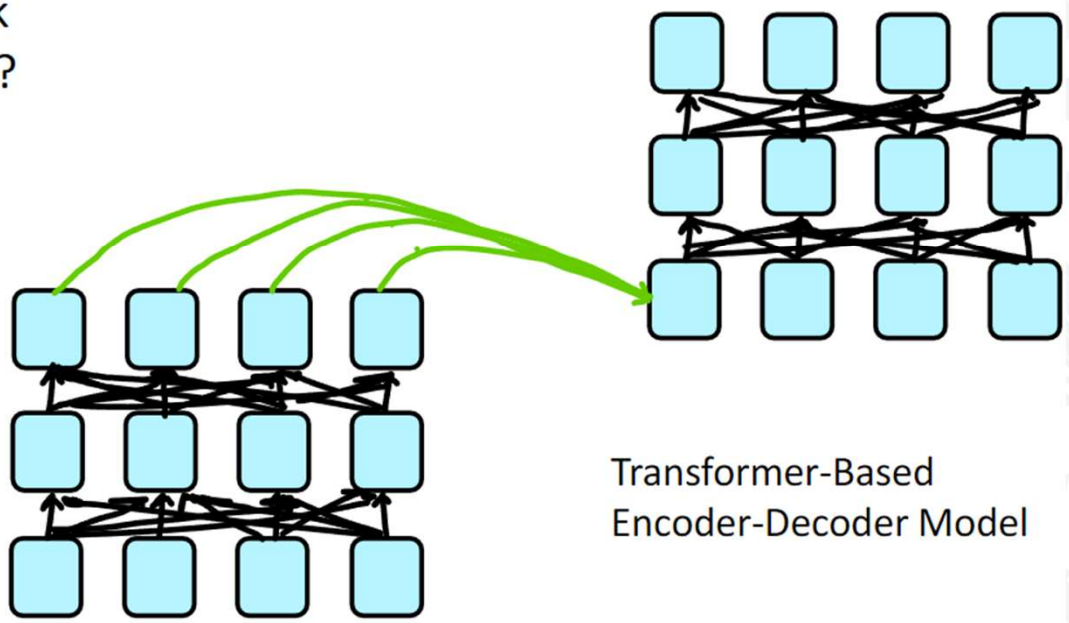
- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.



Credit to https://jalammar.github.io/illustrated-transformer/

Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from “cheating”? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.

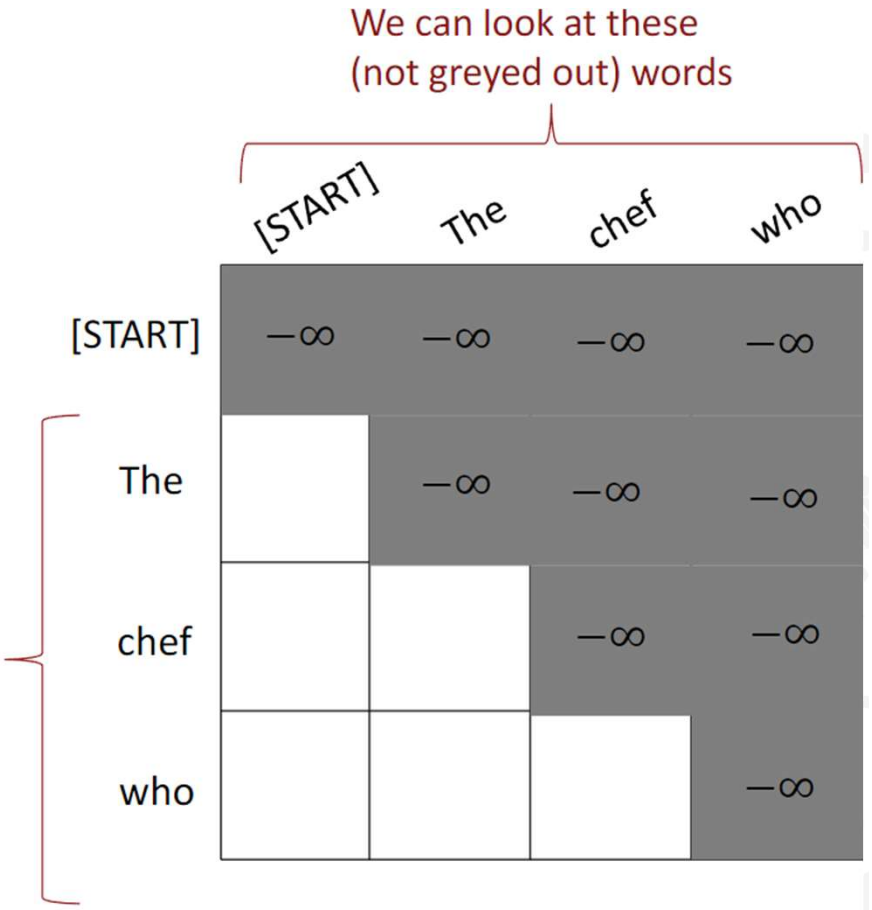


Transformer-Based Encoder-Decoder Model

Decoder: Masked Multi-Head Self-Attention

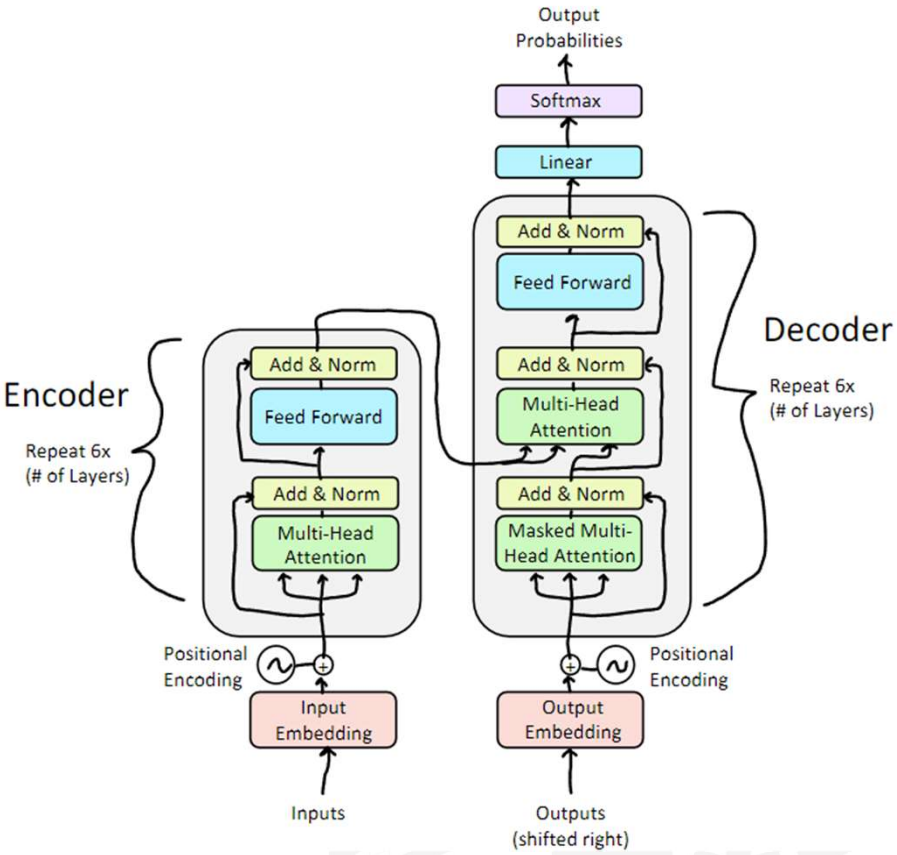
- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



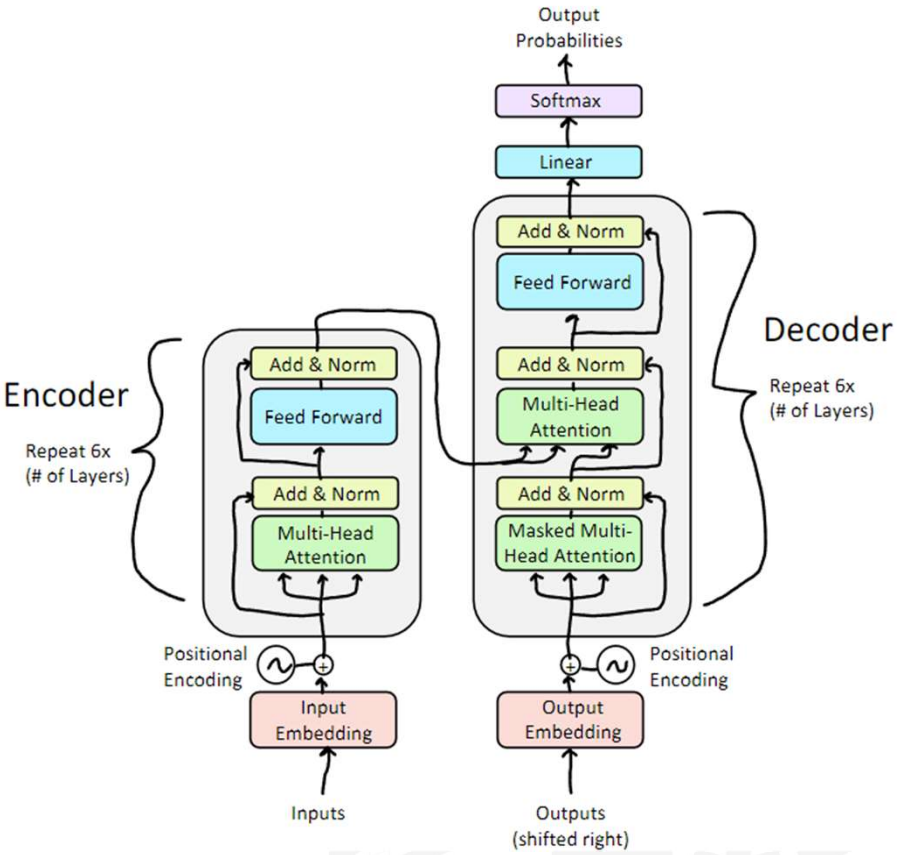
Decoder

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!



Decoder

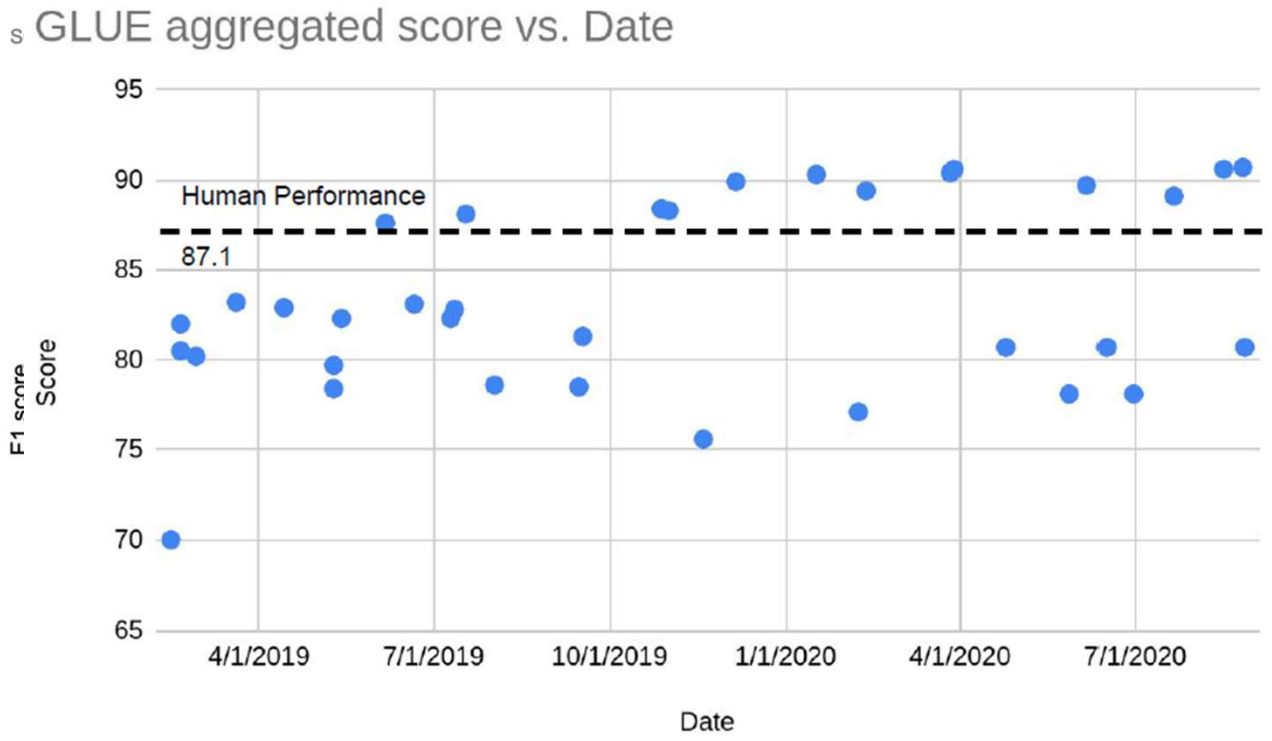
- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!



Transformer-based Language Models

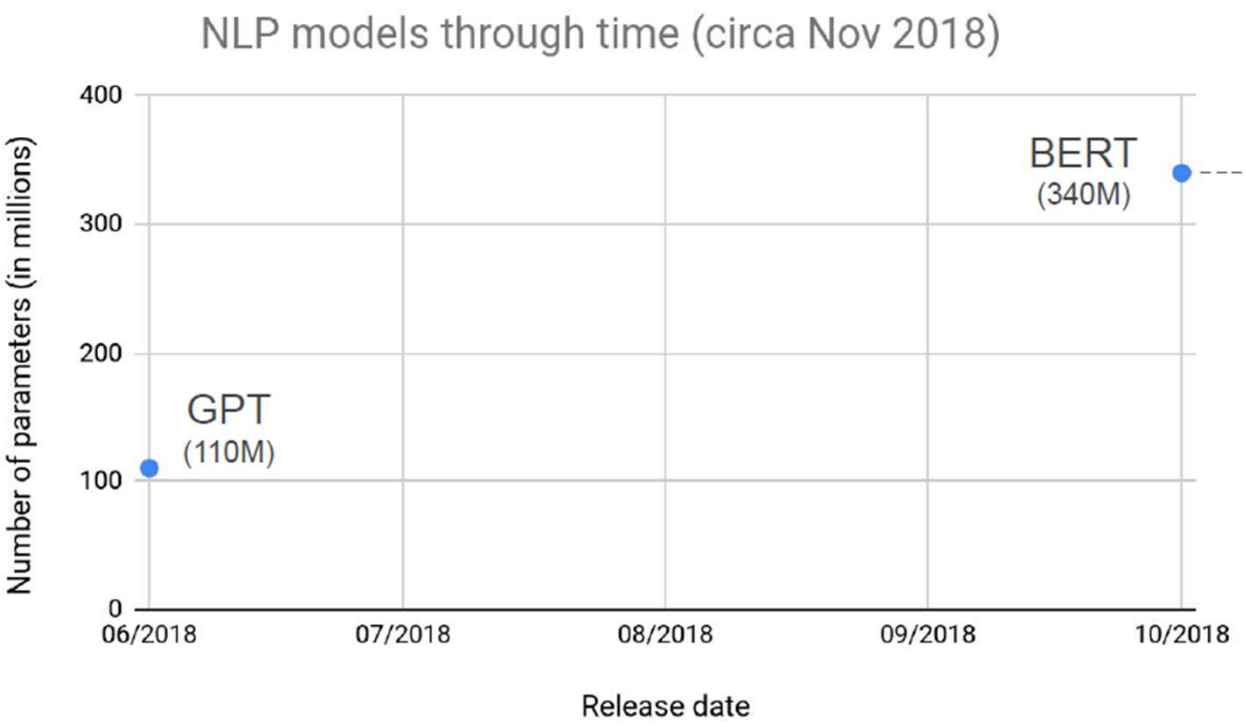
Recent years in NLP

- Benchmarks through the years



The GLUE Benchmark (Wang et al., 2018)

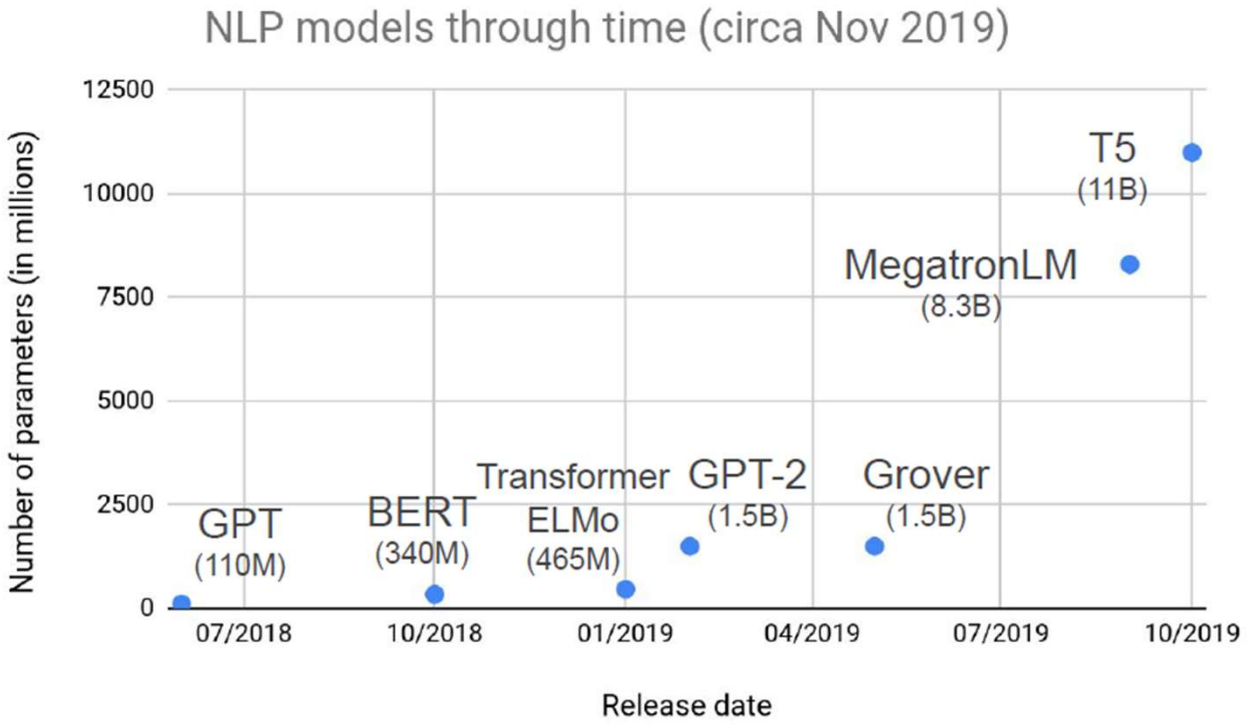
A brief recent history of scale in NLP



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

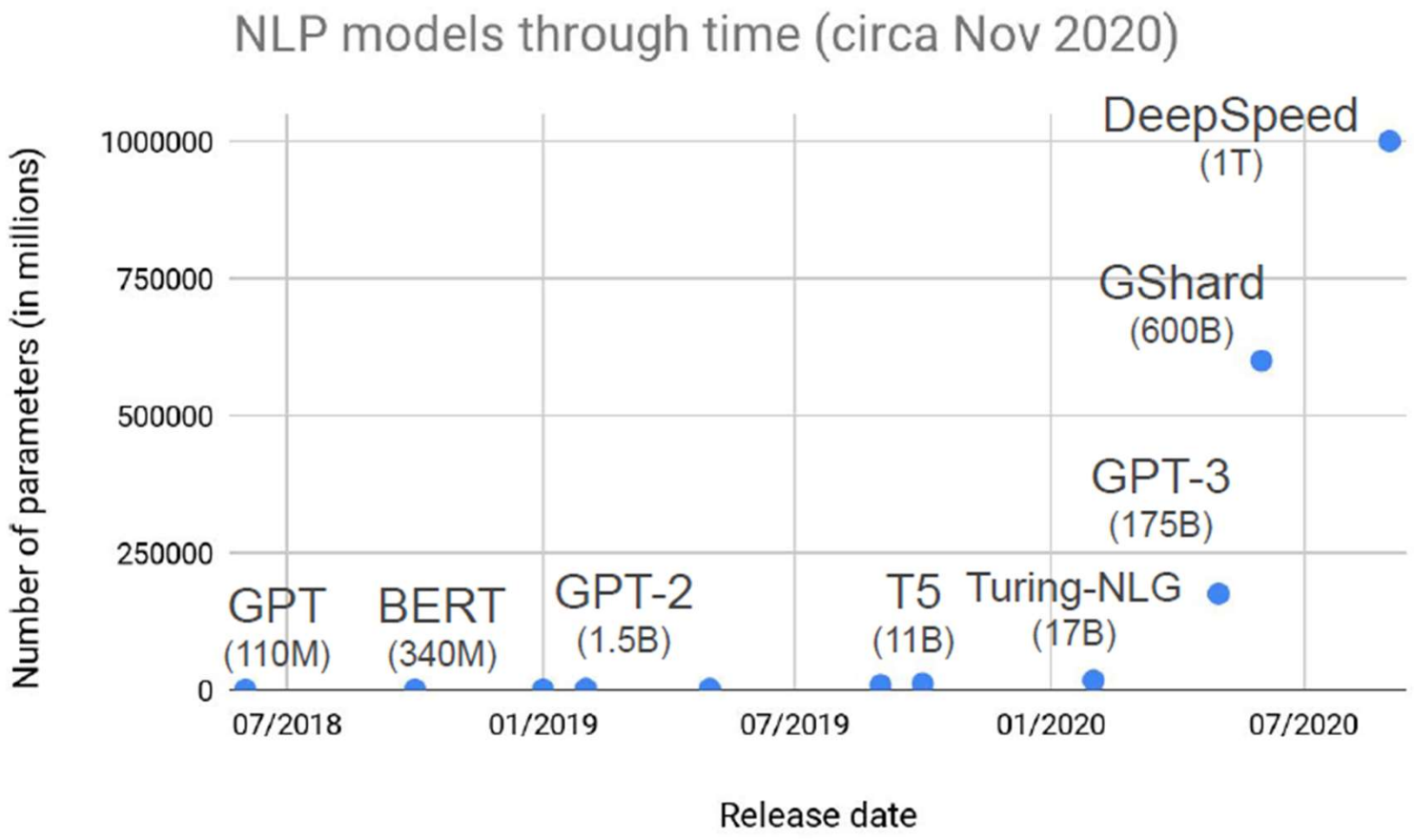
(Devlin et al., 2018)

A brief recent history of scale in NLP

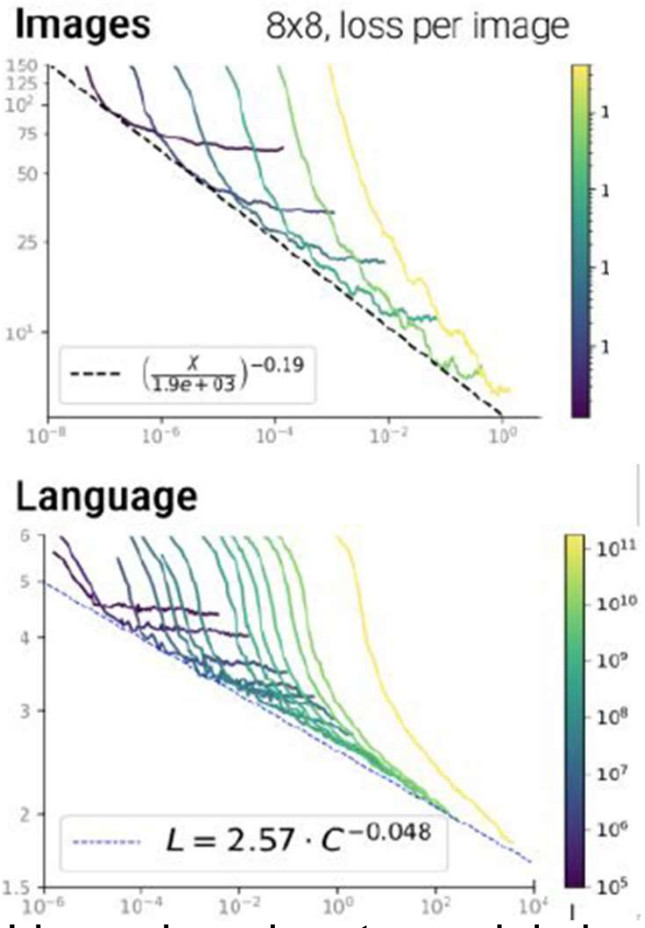


"[...] scaling the model size to 11 billion parameters was the most important ingredient for achieving our best performance."
(Raffel et al, 2019)

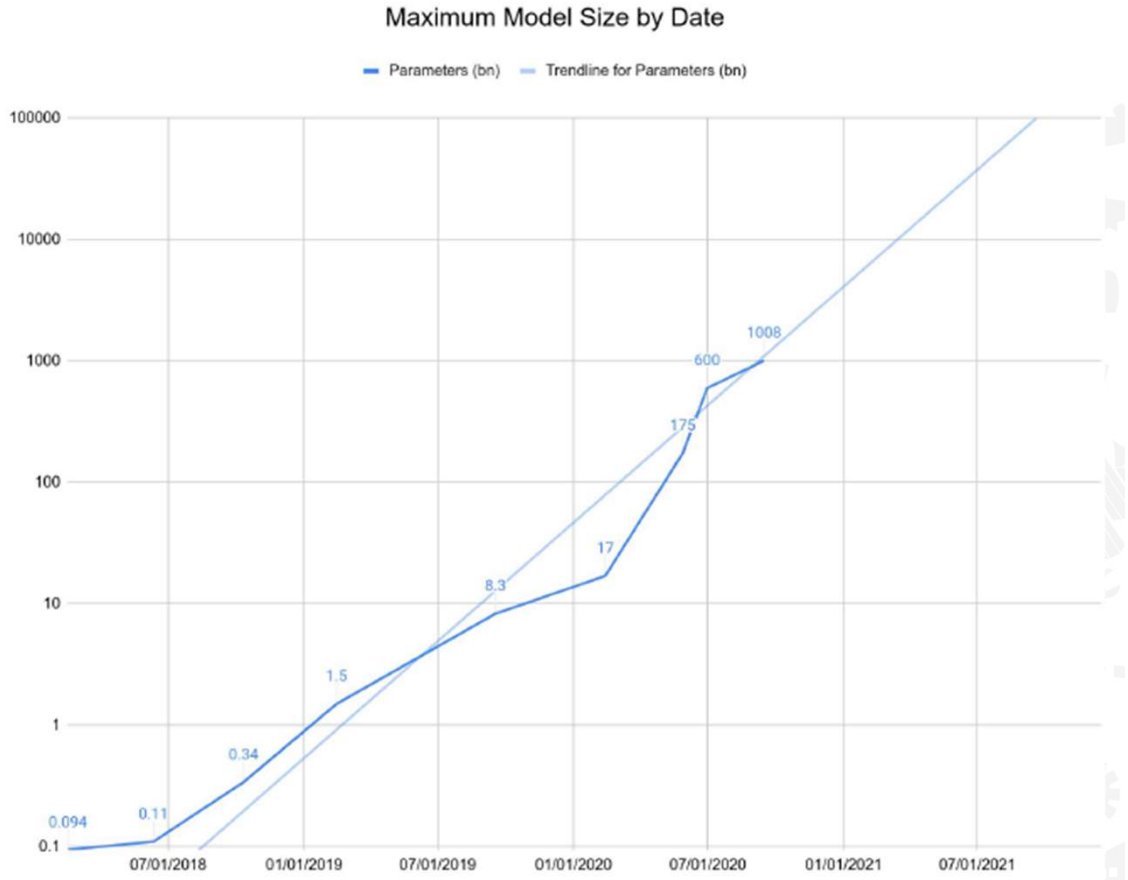
A brief recent history of scale in NLP



Scaling laws



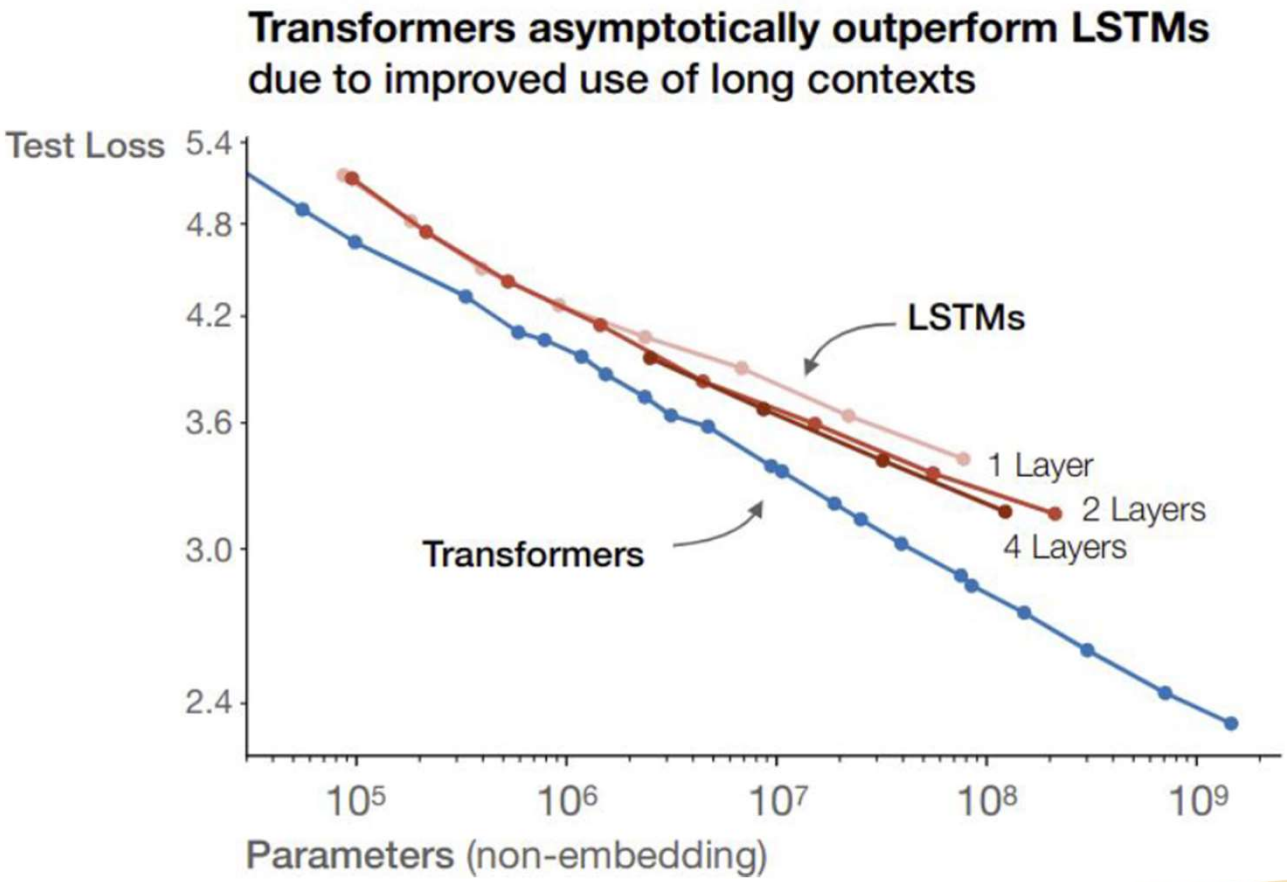
Line colors denote model sizes



* Scaling laws for autoregressive generative modeling, arXiv:2010.14701v2 cs.LG 6 Nov 2020

Why do we need scale?

- Transformers are now ubiquitous



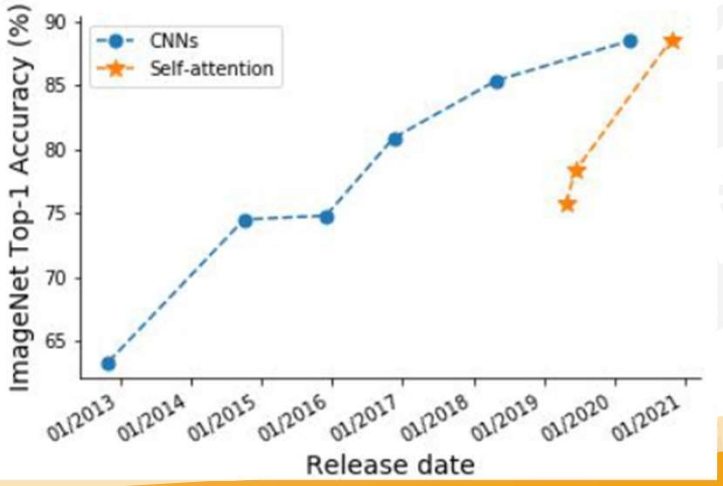
Kaplan et al., 2020: arXiv

Transformers in recent years

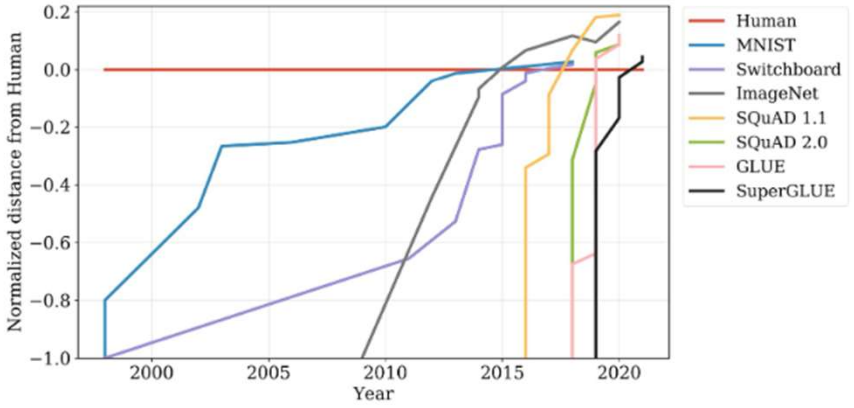
- Transformers have become successful in a wide range of domains and applications including:
  - Mathematics and theorem proving (e.g., Lample et al., 2019, Clark et al., 2020)
  - Music generation (e.g., Anna Huang et la., 2019)
  - Biology (Madani et al., 2020)
  - Vision Language (e.g., Tan et al., 2019, Chen et al., 2020)
  - Computer vision(e.g., Ramachandran et al., 2019, Dosovitskiy et al., 2020)

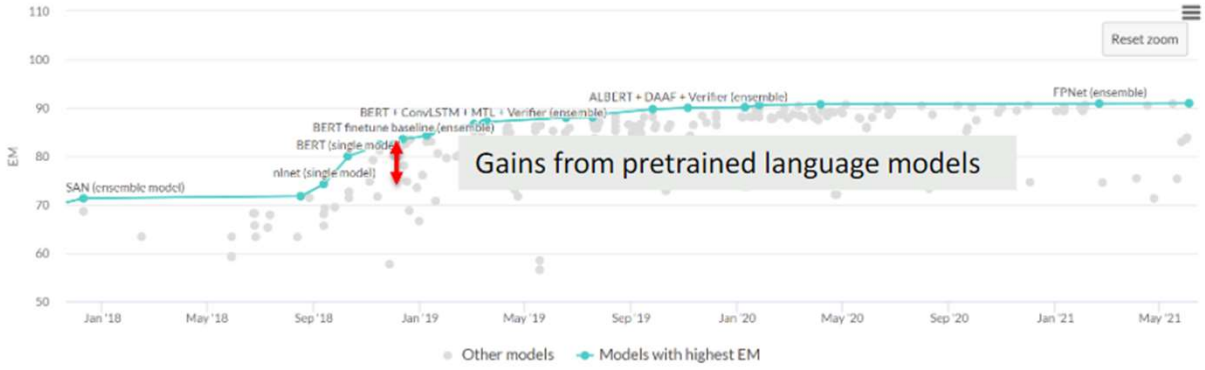
How many slices of pizza are there?
Is this a vegetarian pizza?

Visual Question Answering
(Agrawal et al., 2015)



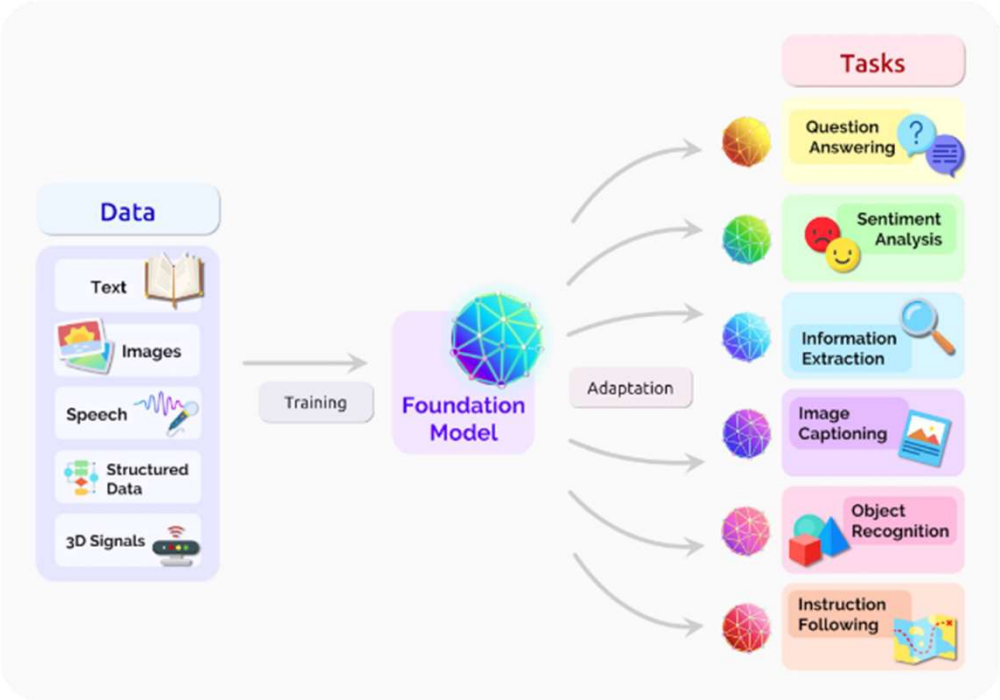
The pretraining revolution





Pretraining has had a major, tangible impact on how well NLP systems work

Pretraining: Scaling unsupervised learning on the Internet



Key ideas in pretraining

- Make sure your model can process large-scale, diverse datasets
- Don't use labeled data (otherwise you can't scale!)
- Compute-aware scaling

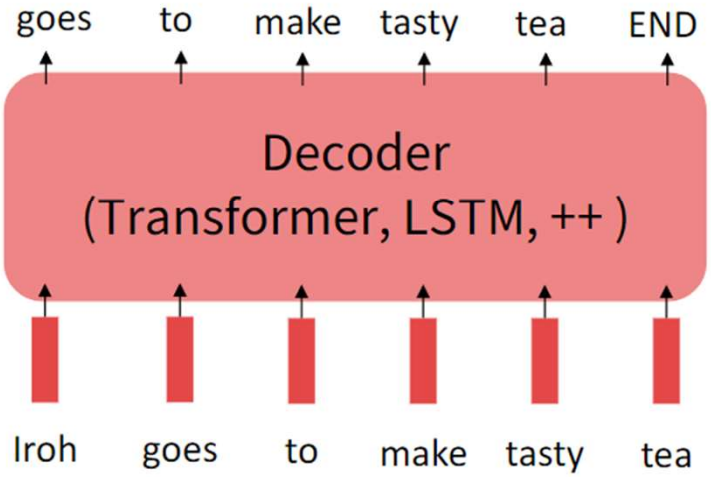
Pretraining through language modeling

Recall the **language modeling** task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

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

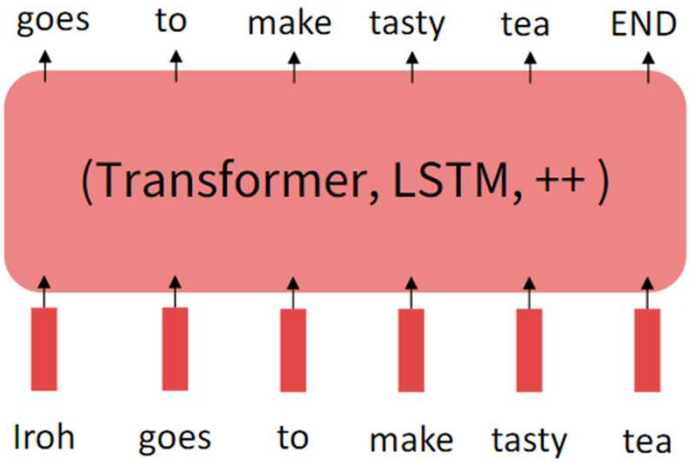


The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

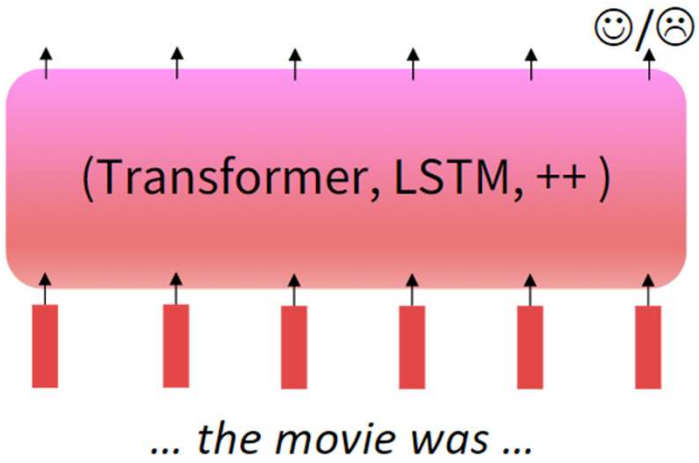
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Stochastic gradient descent and Pretrain/finetune

Why should pretraining and finetuning help, from a "training neural nets" perspective?

- Pretraining provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{\text{pretrain}}(\theta)$.
  - (The pretraining loss.)
- Then, finetuning approximates $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$.
  - (The finetuning loss)
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to $\hat{\theta}$ during finetuning.
  - So, maybe the finetuning local minima near $\hat{\theta}$ tend to generalize well!
  - And/or, maybe the gradients of finetuning loss near $\hat{\theta}$ propagate nicely!

Aside : Vision Trasformer

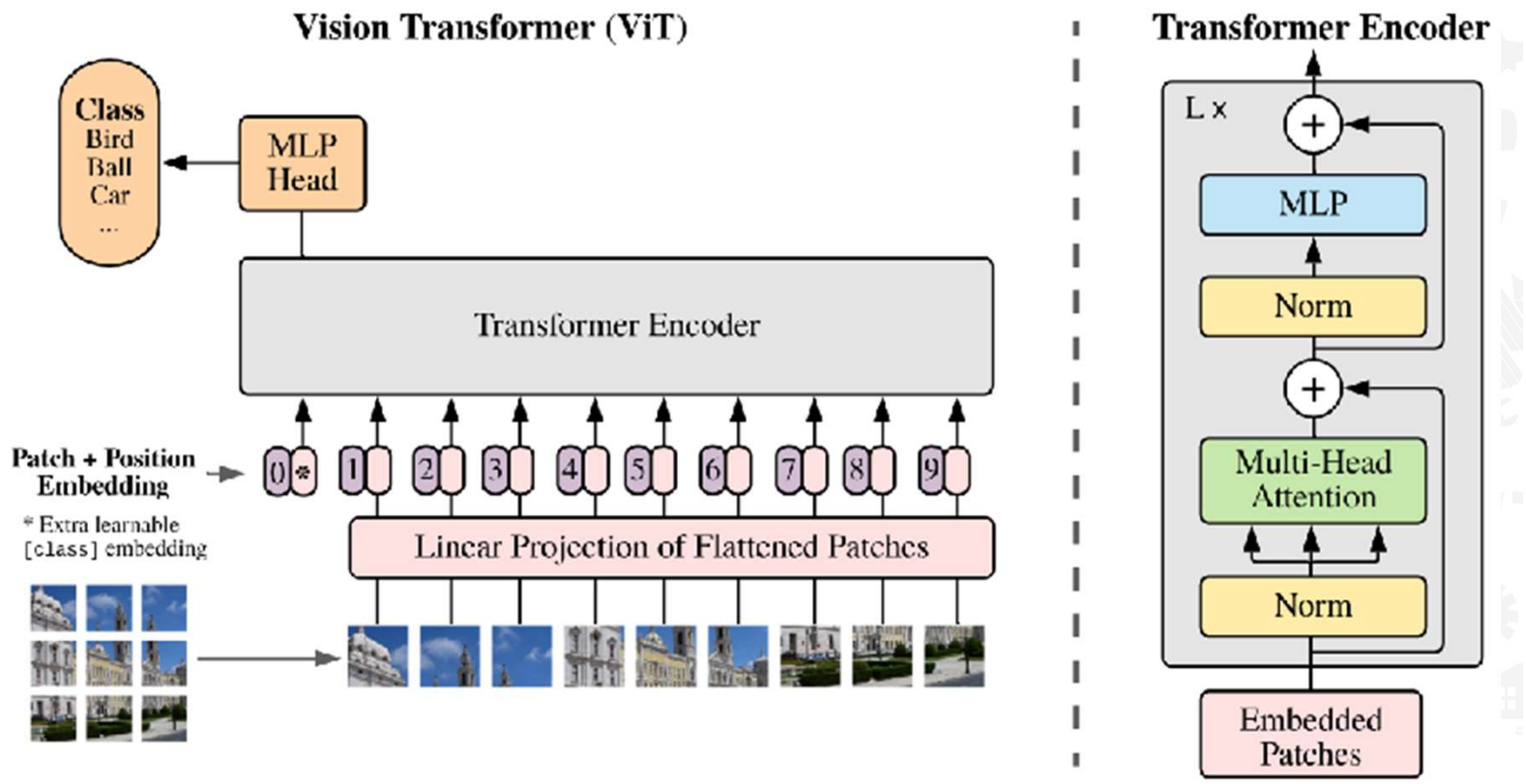
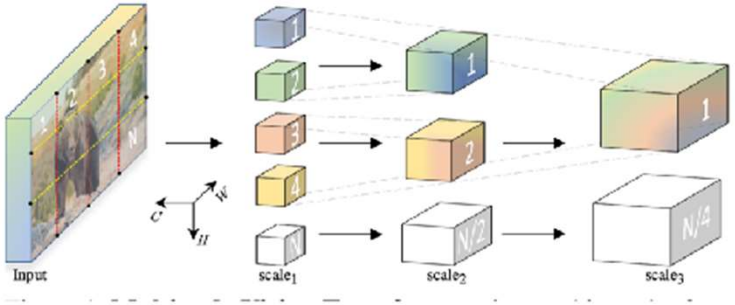
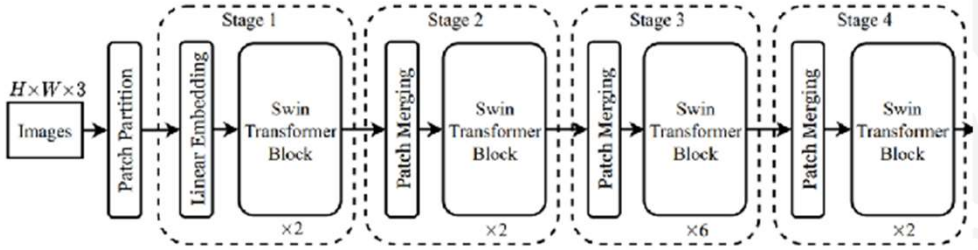


Figure from:
Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020

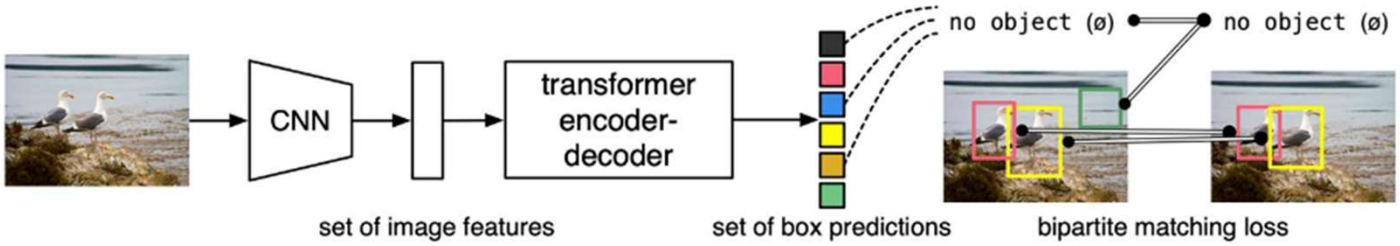
Aside : Vision Trasformer



Fan et al, “Multiscale Vision Transformers”, ICCV 2021

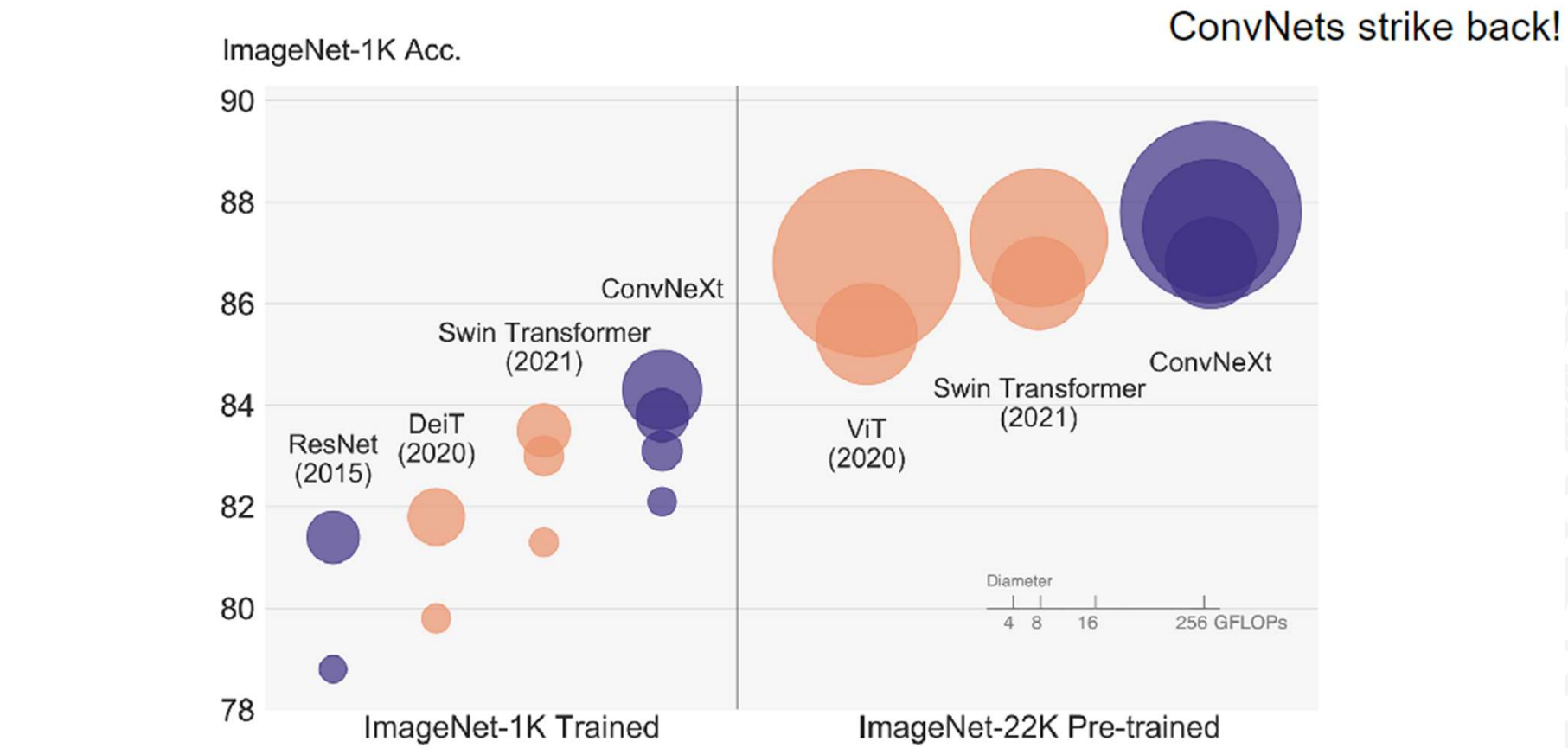


Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021



Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

Aside : Vision Trasformer

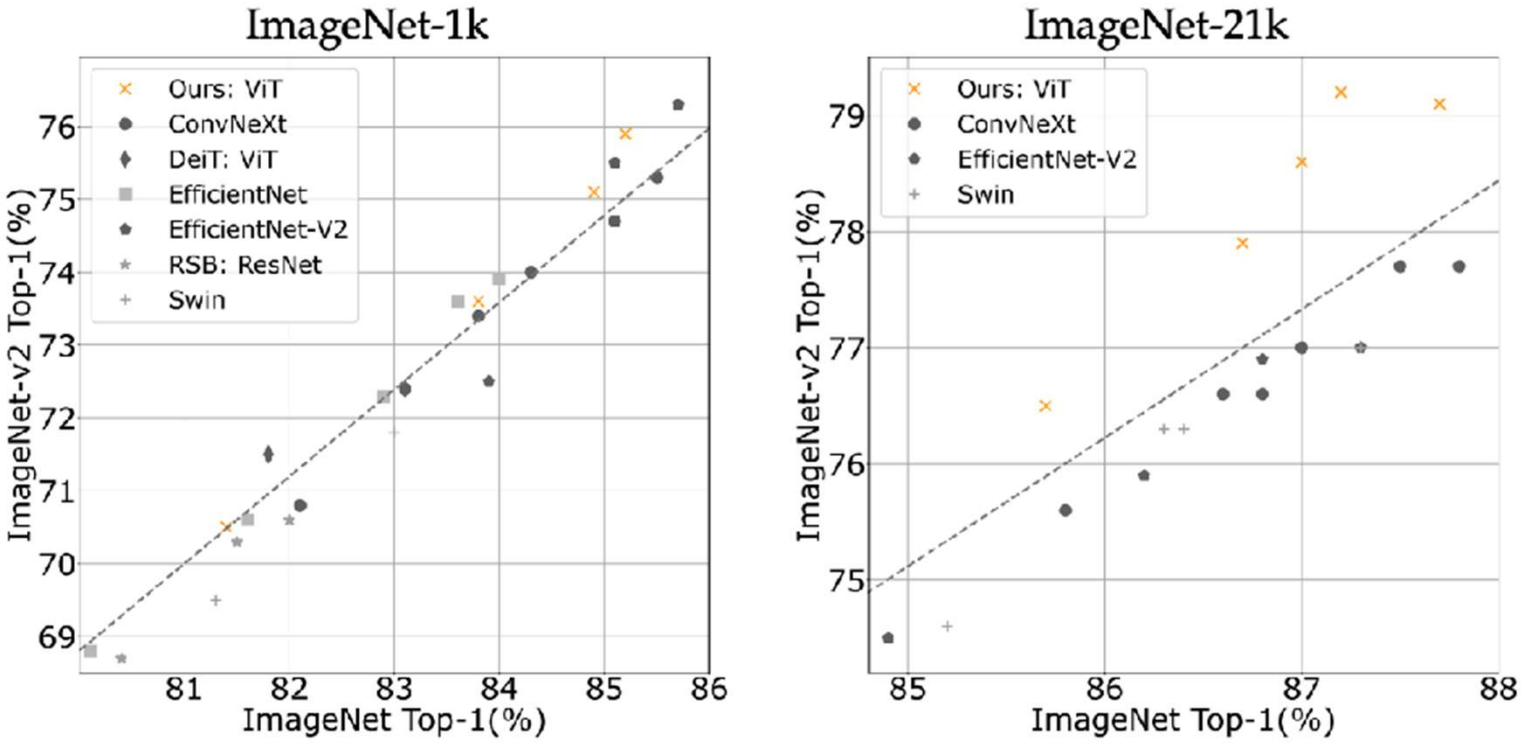


A ConvNet for the 2020s. Liu et al. CVPR 2022

Aside : Vision Trasformer

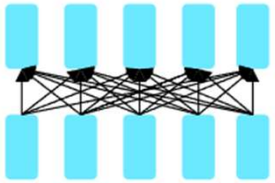
DeiT III: Revenge of the ViT

Hugo Touvron*,† Matthieu Cord† Hervé Jégou*



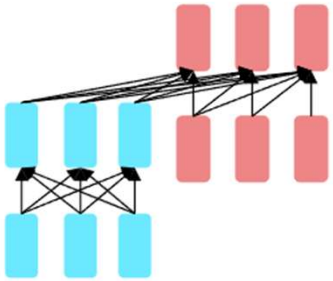
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



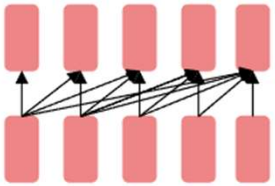
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

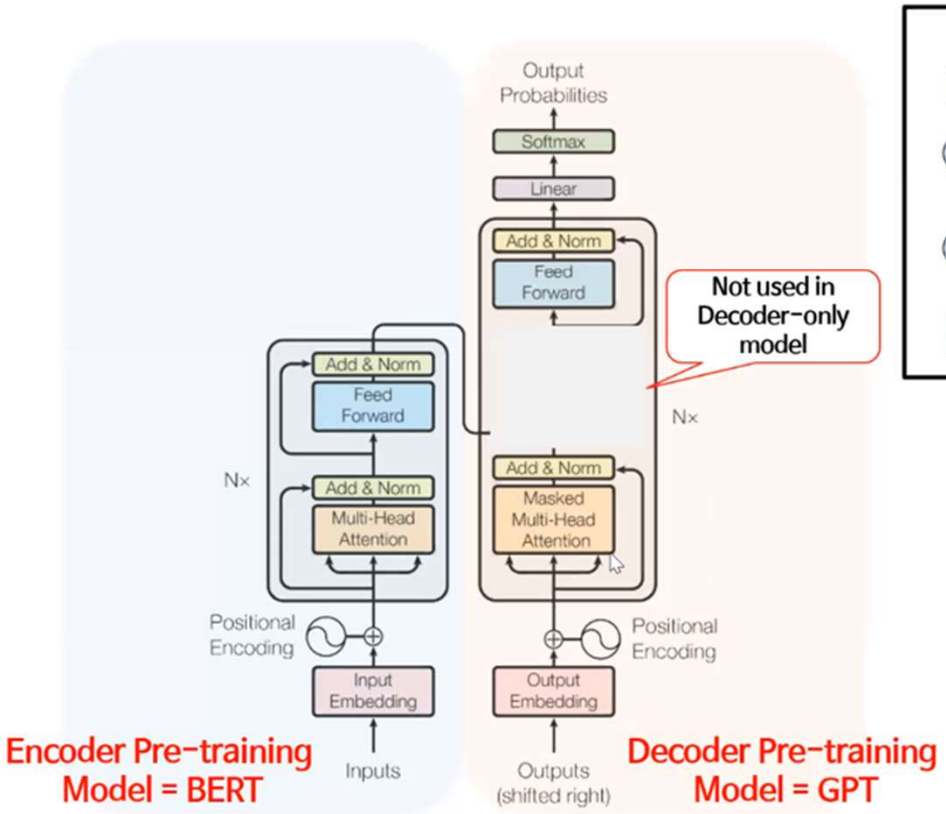
- Good parts of decoders and encoders?
- What's the best way to pretrain them?

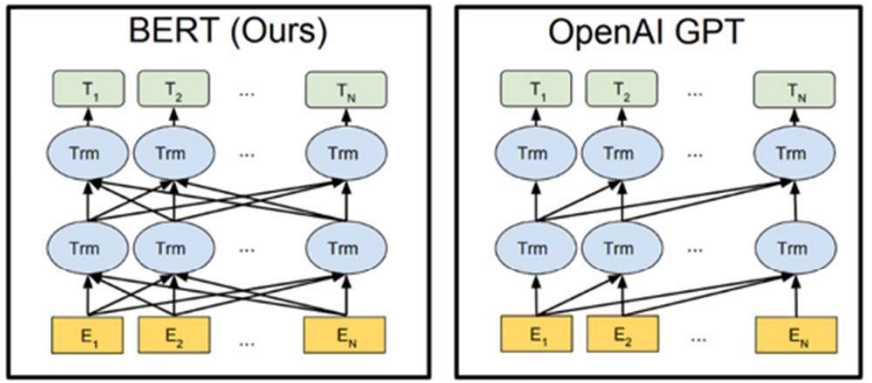


Decoders

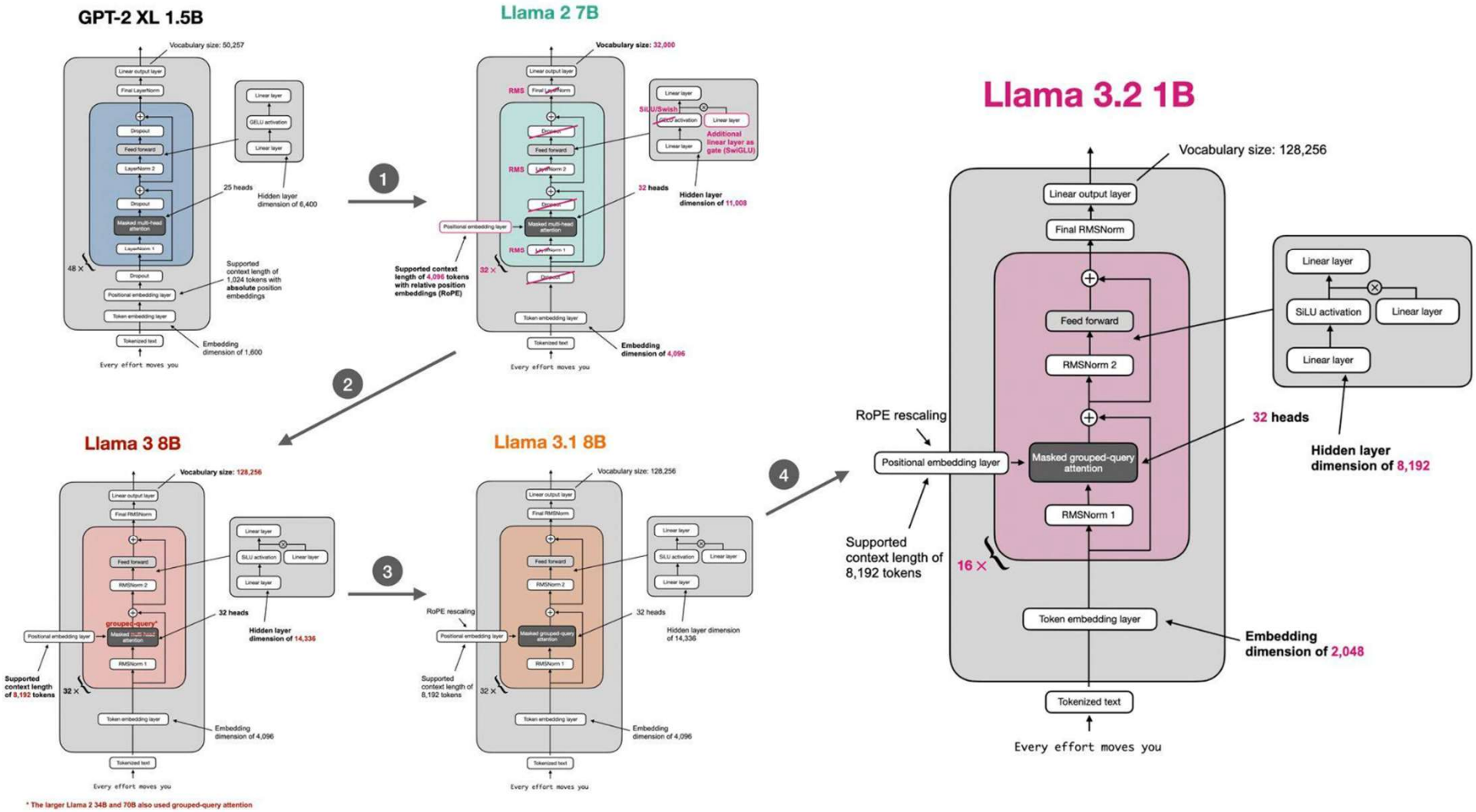
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

BERT & GPT


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Not used in Decoder-only model
Encoder Pre-training Model = BERT
Decoder Pre-training Model = GPT


BERT (Ours)
OpenAI GPT
Trm
E1
E2
EN
T1
T2
TN

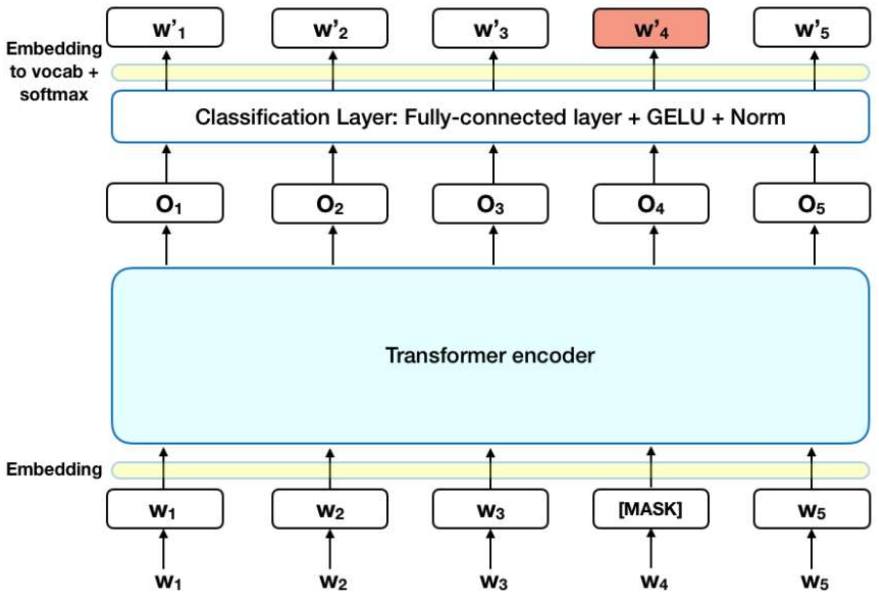
GPT and beyond



*source: [LLMs-from-scratch/ch05/07_gpt_to_llama/converting-llama2-to-llama3.ipynb at main · rasbt/LLMs-from-scratch (github.com)]

BERT

- **Masked language modeling** instead of predicting every next token
  - 15% tokens are chosen at random
    - 80% are actually replaced with the token [MASK]
    - 10% are replaced with a random token
    - 10% are left unchanged
- **NSP**(Next Sentence Prediction)



Loss functions in BERT

- Predicting masked token

```
                          store          gallon
                            ↑              ↑
the man went to the [MASK] to buy a [MASK] of milk
```

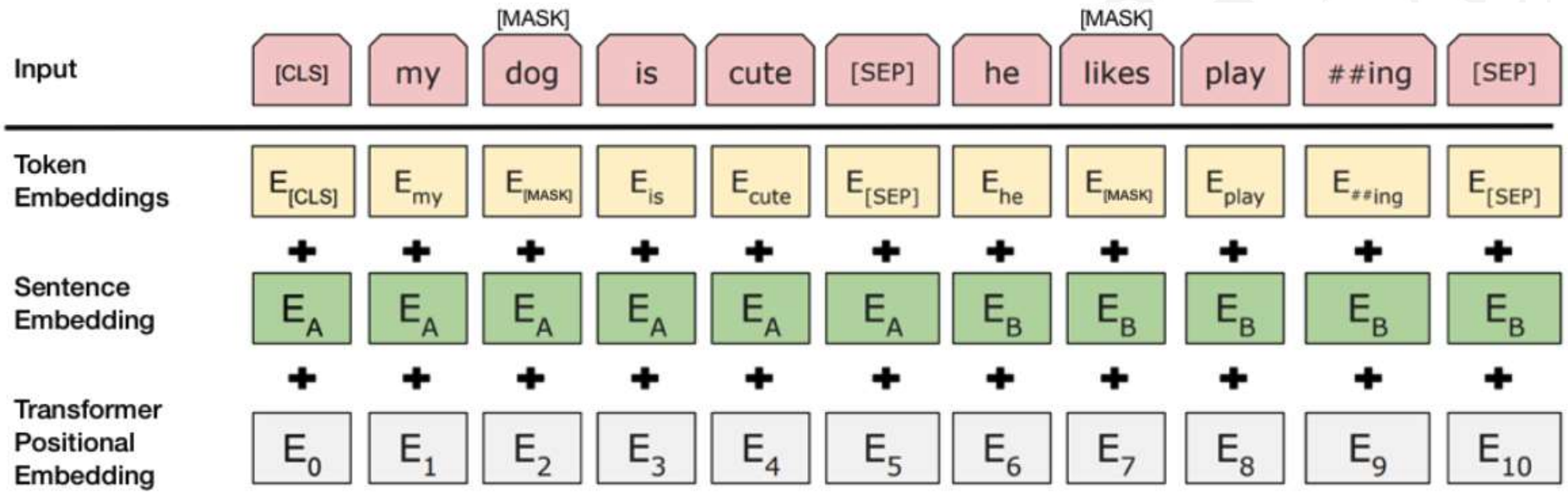
- Next Sentence Prediction
  - Binary classification task if the 2nd sentence is the actual next sentence of the first one

```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
        penguin [MASK] are flight ##less birds [SEP]
Label = NotNext
```

- These two tasks are self-supervised

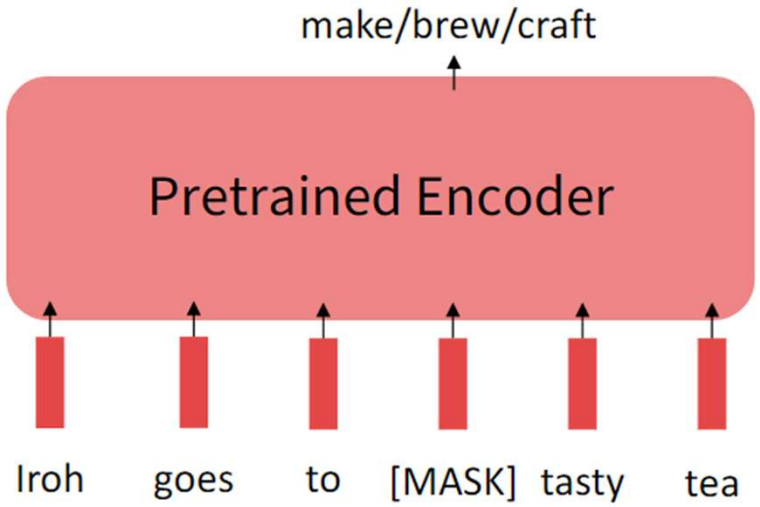
Input Embeddings in BERT

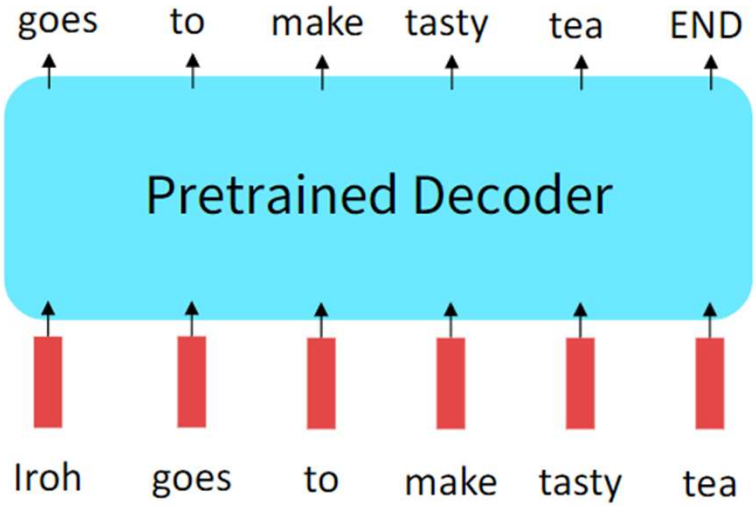


Limitations of pretrained encoders

Those results looked great! Why not use pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder; BERT and oth pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





Subword-based encoding: Byte Pair Encoding

- Originally a compression algorithm
  - Most frequent byte pair → a new byte

Replace bytes with character ngrams

(though, actually, some people have done interesting things with bytes)

Rico Sennrich, Barry Haddow, and Alexandra Birch. ***Neural Machine Translation of Rare Words with Subword Units****. ACL 2016.*

https://arxiv.org/abs/1508.07909

https://github.com/rsennrich/subword-nmt

https://github.com/EdinburghNLP/nematus

Byte Pair Encoding

- A word segmentation algorithm:
  - Though done as bottom-up clustering
  - Start with a unigram vocabulary of all (Unicode) **characters** in data
  - Most frequent **ngram pairs** → a new **ngram**

Dictionary

5 l o w
2 l o w e r
6 n e w e s t
3 w i d e s t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

Start with all characters in vocab

Byte Pair Encoding

- A word segmentation algorithm:
  - Though done as bottom-up clustering
  - Start with a unigram vocabulary of all (Unicode) **characters** in data
  - Most frequent **ngram pairs** → a new **ngram**

Dictionary

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair (e, s) with freq 9

Byte Pair Encoding

- A word segmentation algorithm:
  - Though done as bottom-up clustering
  - Start with a unigram vocabulary of all (Unicode) **characters** in data
  - Most frequent **ngram pairs** → a new **ngram**

Dictionary

5 l o w
2 l o w e r
6 n e w **est**
3 w i d **est**

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Add a pair (es, t) with freq 9

Byte Pair Encoding

- A word segmentation algorithm:
  - Though done as bottom-up clustering
  - Start with a unigram vocabulary of all (Unicode) **characters** in data
  - Most frequent **ngram pairs** → a new **ngram**

Dictionary

5 **lo** w
2 **lo** w e r
6 n e w est
3 w i d est

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Add a pair (l, o) with freq 7

Byte Pair Encoding

Dictionary

5 l o w
2 l o w e r
6 n e w e s t
3 w i d e s t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

l, o, w, e, r, n, w, s, t, i, d, **es**

5 l o w
2 l o w e r
6 n e w **est**
3 w i d **est**

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Byte Pair Encoding

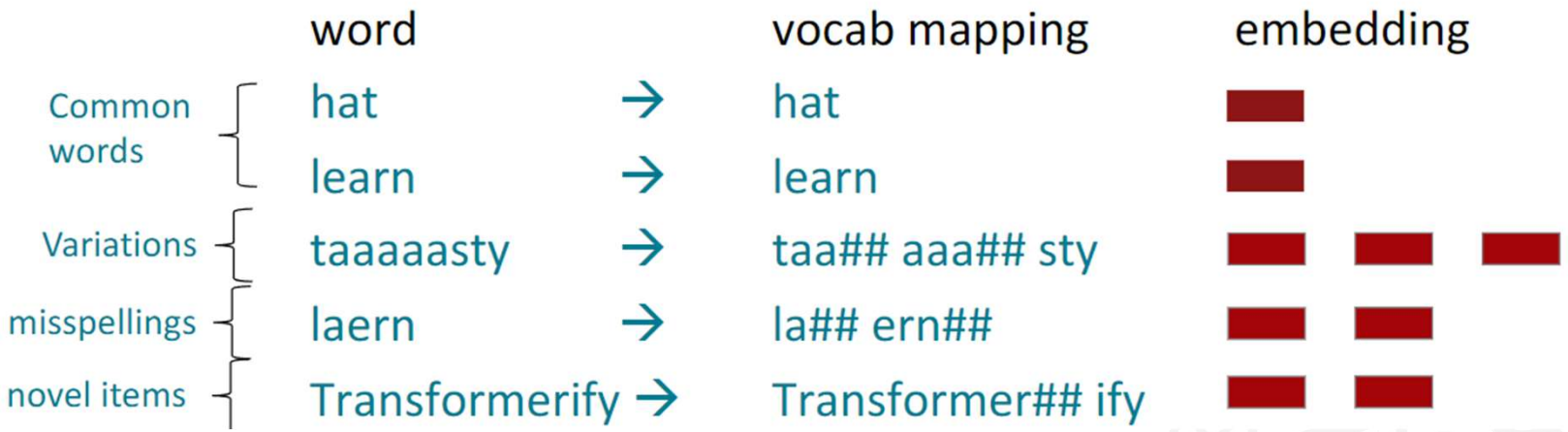
- Have a target vocabulary size and stop when you reach it
- Do deterministic longest piece segmentation of words
- Segmentation is only within words identified by some prior tokenizer

- Automatically decides vocabulary for systems
  - No longer strongly "word" based in conventional way

Word structure and Subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.



Words in writing Systems

Writing systems vary in how they represent words – or don't

- No word segmentation: 安理会认可利比亚问题柏林峰会成果
- Words (mainly) segmented: *This is a sentence with words.*
  - Clitics/pronouns/agreement?
    - Separated — **Je vous ai apporté** des bonbons
    - Joined — ففلناها = ها +نا +قال +ف = so+said+we+it
  - Compounds?
    - Separated — life insurance company employee
    - Joined — Lebensversicherungsgesellschaftsangestellter

WordPiece/SentencePiece Model

- Google NMT (GNMT) uses a variant of BPE
  - v1: WordPiece model
  - V2: SentencePiece model
- Rather than char n-gram count, uses **a greedy approximation to maximizing log likelihood to choose the piece**
  - Add *n*-gram that maximally reduces perplexity
- **WordPiece** model tokenizes inside words
- **SentencePiece** model works from raw text
  - Whitespace is retained as special token (_) and grouped normally
  - You can reverse things at end by joining pieces and recoding them to space

WordPiece/SentencePiece Model

- BERT uses a variant of the WordPiece model
  - (relatively) common words are in the vocab:
    - At firafax, 1910s
  - Other words are built from WordPieces:
    - Hypatia = h ##yp ##ati ##a

GLUE Benchmark

- General Language Understanding Evaluation
  - A collection of sentence- or sentence-pair lang. understanding tasks
  - Built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty

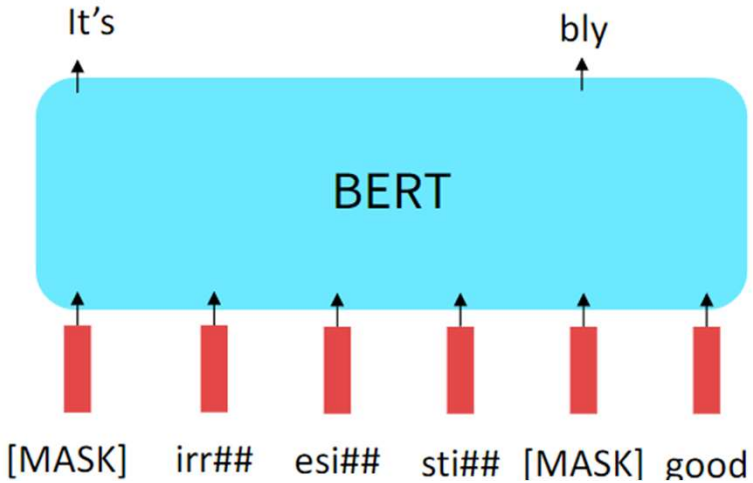
Dataset	Description	Data example	Metric
CoLA	Is the sentence grammatical or ungrammatical?	"This building is than that one." = **Ungrammatical**	Matthews
SST-2	Is the movie review positive, negative, or neutral?	"The movie is funny , smart , visually inventive , and most of all , alive ." = **.93056 (Very Positive)**	Accuracy
MRPC	Is the sentence B a paraphrase of sentence A?	A) "Yesterday , Taiwan reported 35 new infections , bringing the total number of cases to 418 ." B) "The island reported another 35 probable cases yesterday , taking its total to 418 ." = **A Paraphrase**	Accuracy / F1
STS-B	How similar are sentences A and B?	A) "Elephants are walking down a trail." B) "A herd of elephants are walking along a trail." = **4.6 (Very Similar)**	Pearson / Spearman
QQP	Are the two questions similar?	A) "How can I increase the speed of my internet connection while using a VPN?" B) "How can Internet speed be increased by hacking through DNS?" = **Not Similar**	Accuracy / F1
MNLI-mm	Does sentence A entail or contradict sentence B?	A) "Tourist Information offices can be very helpful." B) "Tourist Information offices are never of any help." = **Contradiction**	Accuracy
QNLI	Does sentence B contain the answer to the question in sentence A?	A) "What is essential for the mating of the elements that create radio waves?" B) "Antennas are required by any radio receiver or transmitter to couple its electrical connection to the electromagnetic field." = **Answerable**	Accuracy
RTE	Does sentence A entail sentence B?	A) "In 2003, Yunus brought the microcredit revolution to the streets of Bangladesh to support more than 50,000 beggars, whom the Grameen Bank respectfully calls Struggling Members." B) "Yunus supported more than 50,000 Struggling Members." = **Entailed**	Accuracy
WNLI	Sentence B replaces sentence A's ambiguous pronoun with one of the nouns - is this the correct noun?	A) "Lily spoke to Donna, breaking her concentration." B) "Lily spoke to Donna, breaking Lily's concentration." = **Incorrect Referent**	Accuracy

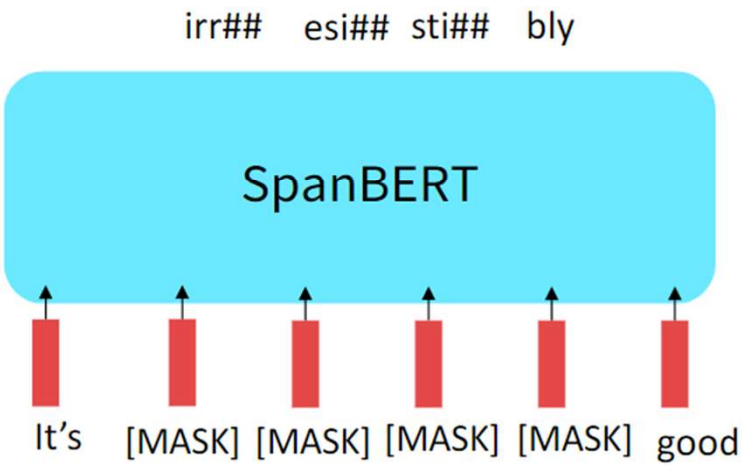
Extensions of BERT

You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++

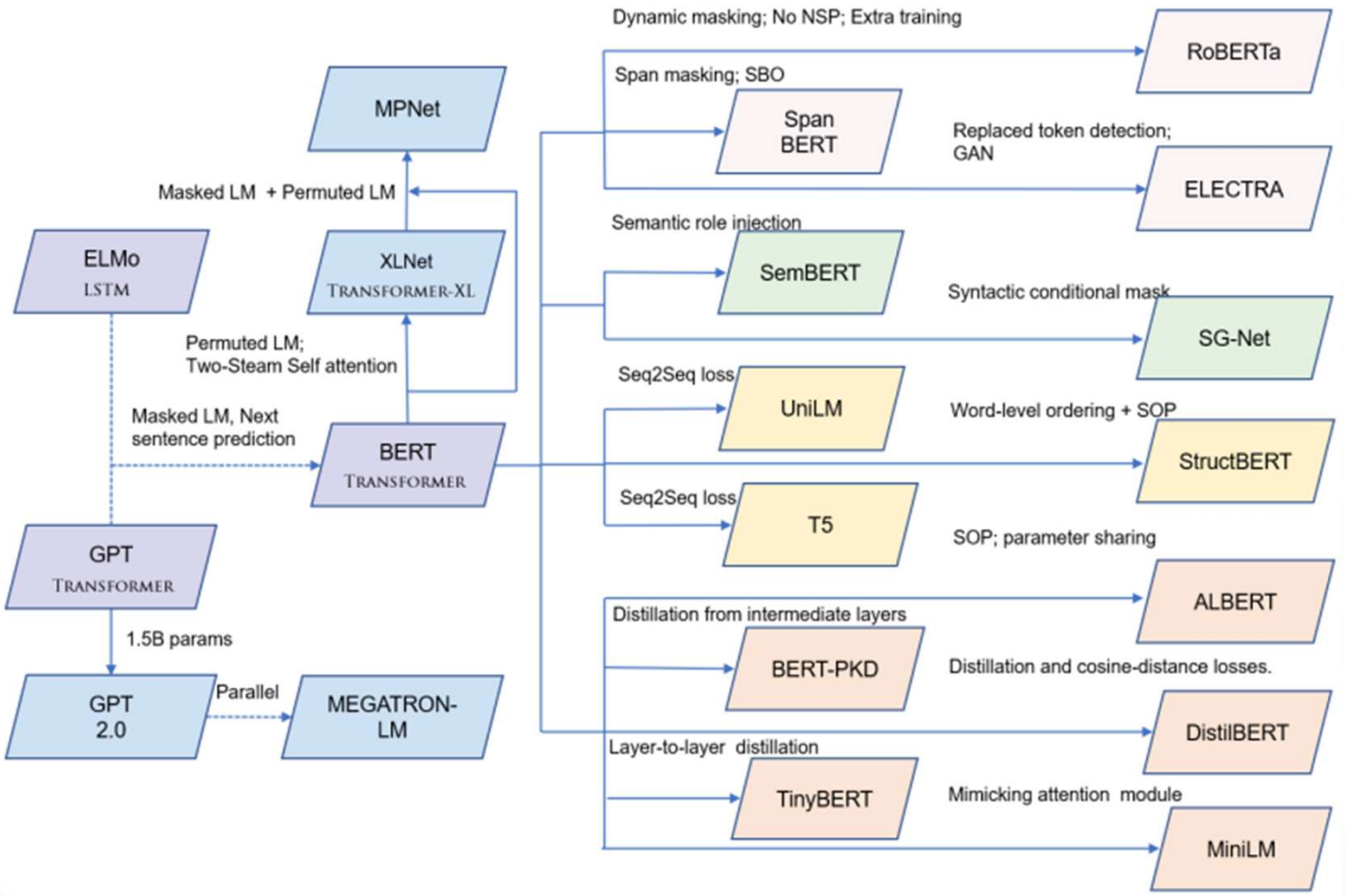
Some generally accepted improvements to the BERT pretraining formula:

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task





Encoder-based LM (BERT family)



Pretraining encoder-decoders: what pretraining objective to use?

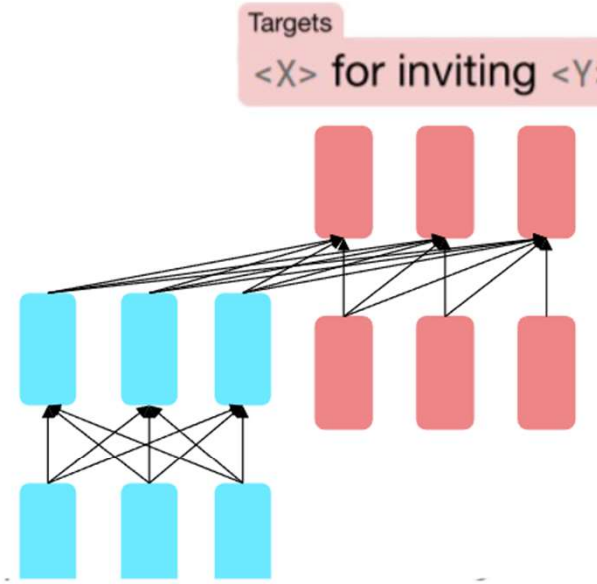
What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators

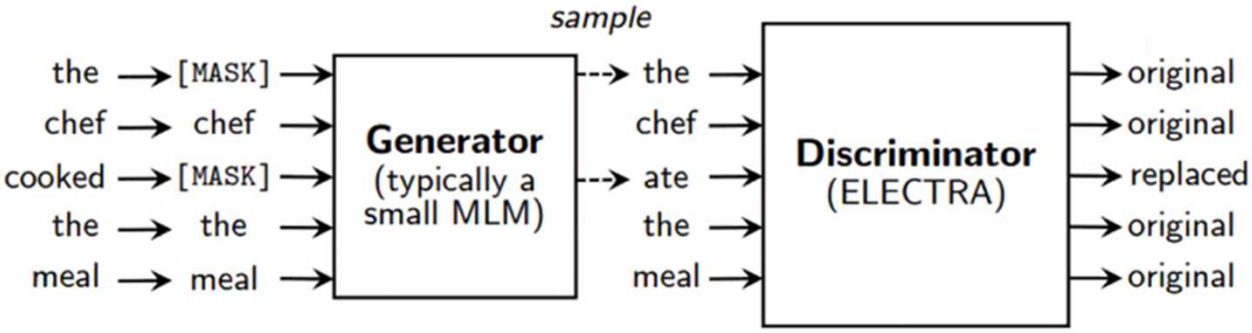


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA

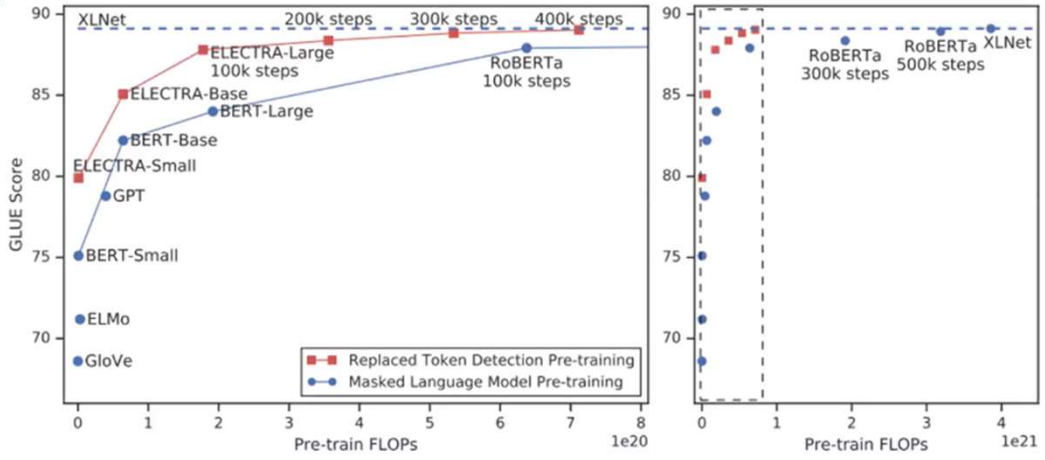


Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

Pretraing decoders

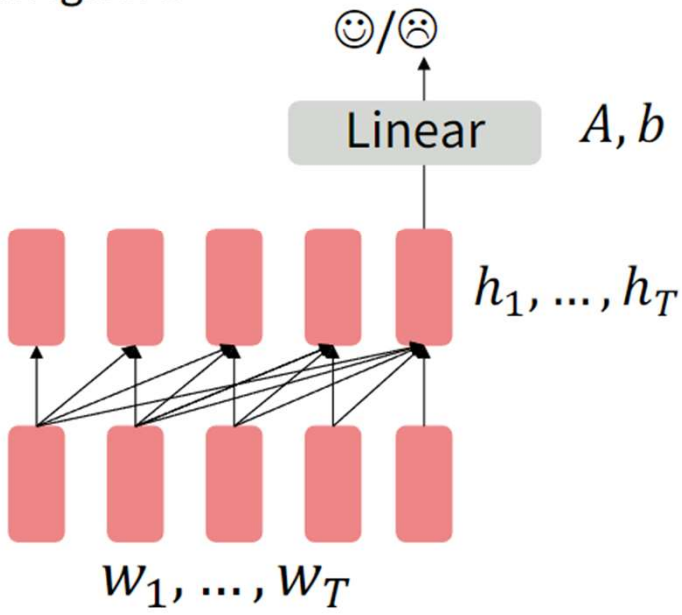
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

We can finetune them by training a softmax classifier on the last word's hidden state.

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$y \sim Ah_T + b$$

Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraing decoders

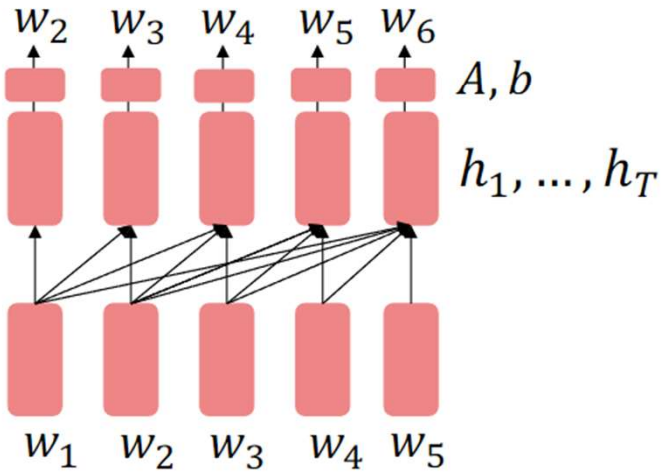
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

Pretraing decoders

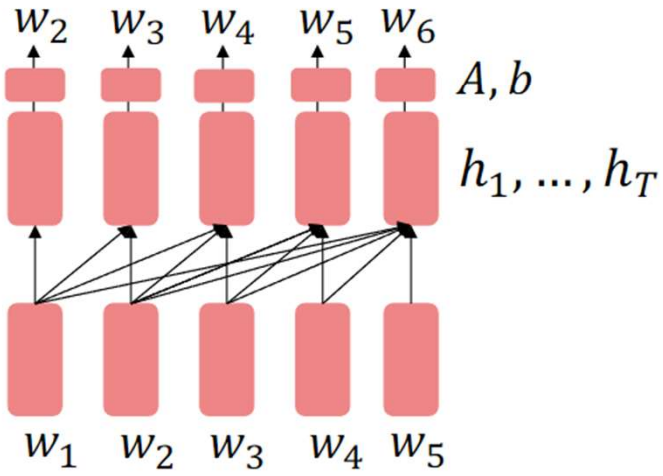
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer(GPT)

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

Generative Pretrained Transformer(GPT)

How do we format inputs to our decoder for **finetuning tasks?**

Natural Language Inference: Label pairs of sentences as *entailing/contradictory/neutral*

Premise: *The man is in the doorway*
Hypothesis: *The person is near the door*
} **entailment**

Radford et al., 2018 evaluate on natural language inference.
Here's roughly how the input was formatted, as a sequence of tokens for the decoder.

[START] *The man is in the doorway* [DELIM] *The person is near the door* [EXTRACT]

The linear classifier is applied to the representation of the [EXTRACT] token.

Increasingly convincing generations (GPT2)

We mentioned how pretrained decoders can be used **in their capacities as language models.**

GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT-3, In-context learning, and very large models

So far, we've interacted with pretrained models in two ways:

- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about and take their predictions.

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

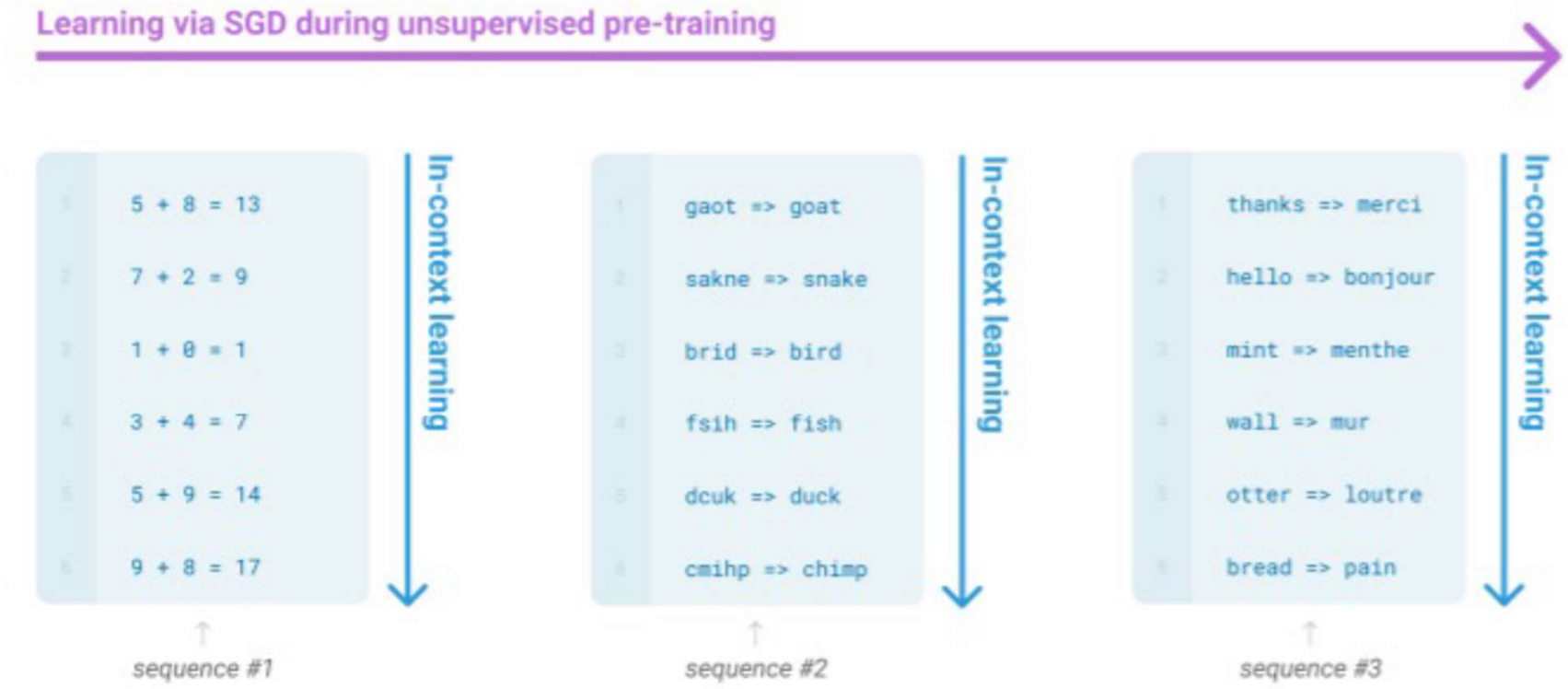
GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.

GPT-3 has 175 billion parameters.

ChatGPT/GPT-4/GPT-3.5 Turbo introduced a further instruction-tuning idea that we cover next lecture

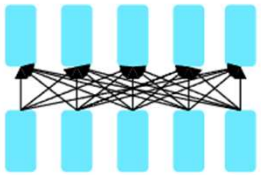
GPT-3, In-context learning, and very large models

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



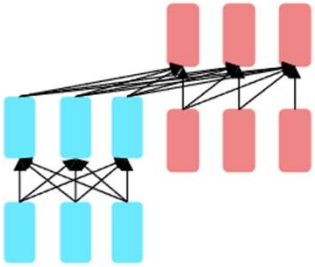
Three types of architectures for pretraining

The neural architecture influences the type of pretraining, and natural use cases.



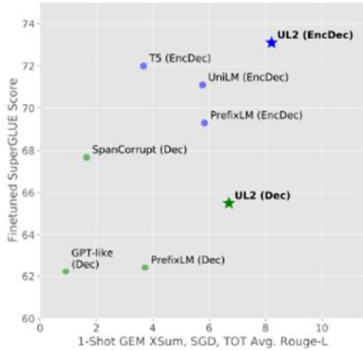
Encoders

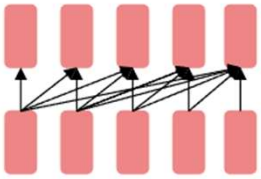
- Gets bidirectional context – can condition on future!
- Good if only doing analysis of text (better than decoders)



Encoder-Decoders

- Good parts of decoders and encoders?
- Some evidence they are better for NLU
  - [Tay et al. 2022. UL2]





Decoders

- Language models! What we've seen so far. Scale well.
- Best to generate from; have won as to what people build