CS 2770: Sequences (Language and Vision), and Transformers

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu

University of Pittsburgh

Plan for this lecture

- Sequences: Language and vision
    - Recurrent neural networks
    - Applications
        - Image captioning
        - Neural Machine Translation
    - Attention and Self-attention
- Transformers
    - Positional Encoding
    - Multiheaded Self-attention
    - Grouped Query Attention (GQA) and Sliding Window Attention
    - GPT models and Prompt-Engineering
- Vision Transformers
- Tweaking Transformers

Motivation: Descriptive Text for Images

"It was an arresting face, pointed of chin, square of jaw. Her eyes were pale green without a touch of hazel, starred with bristly black lashes and slightly tilted at the ends. Above them, her thick black brows slanted upward, cutting a startling oblique line in her magnolia-white skin–that skin so prized by Southern women and so carefully guarded with bonnets, veils and mittens against hot Georgia suns"

Scarlett O'Hara described in Gone with the Wind

Results with Recurrent Neural Networks

"man in black shirt is playing guitar."

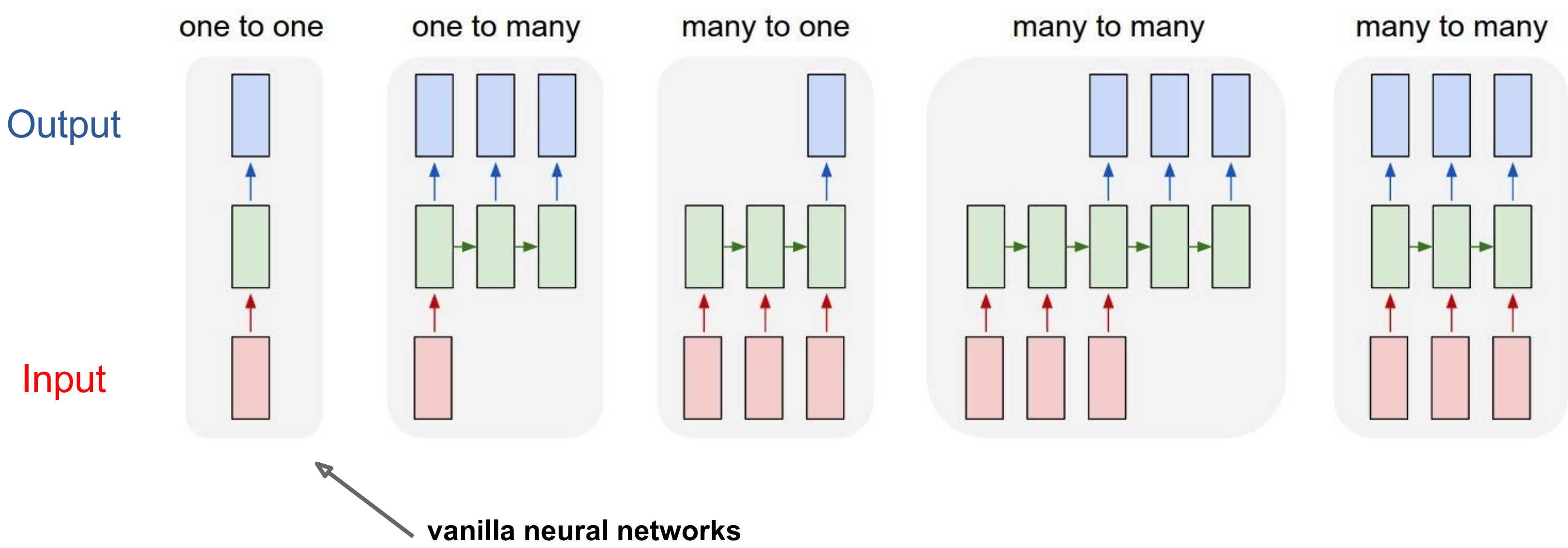
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

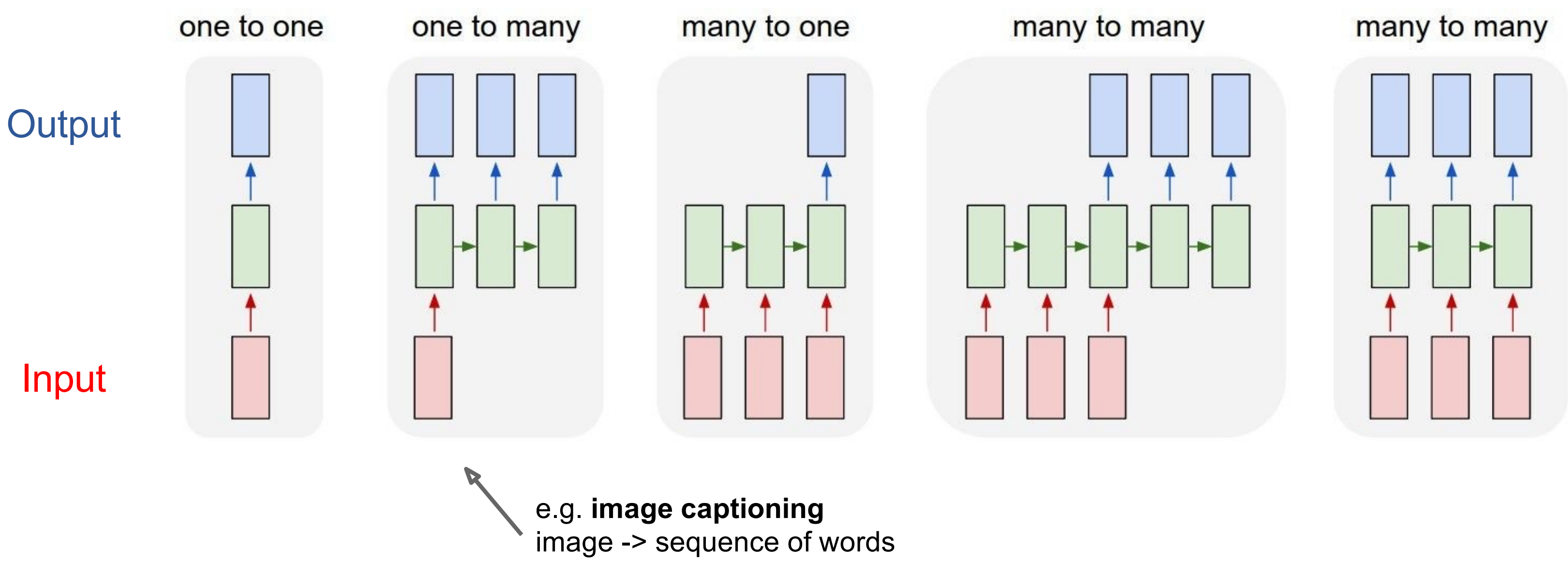
Karpathy and Fei-Fei, CVPR 2015

Recurrent Networks offer a lot of flexibility:



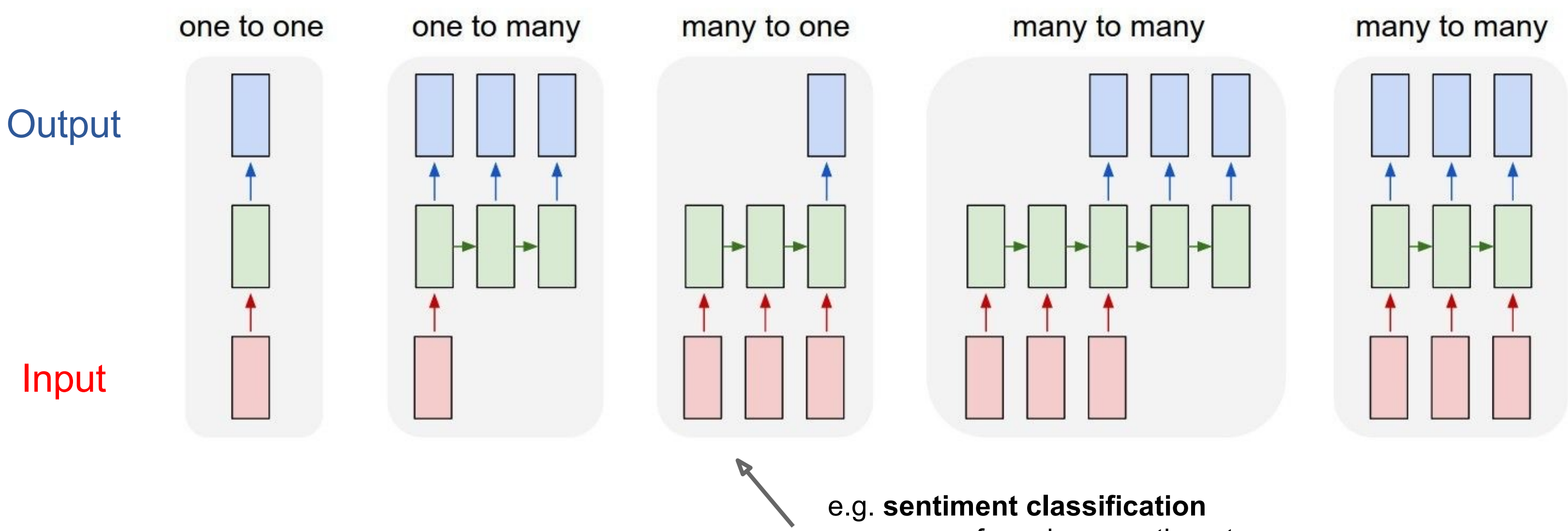
Andrej Karpathy

Recurrent Networks offer a lot of flexibility:



Andrej Karpathy

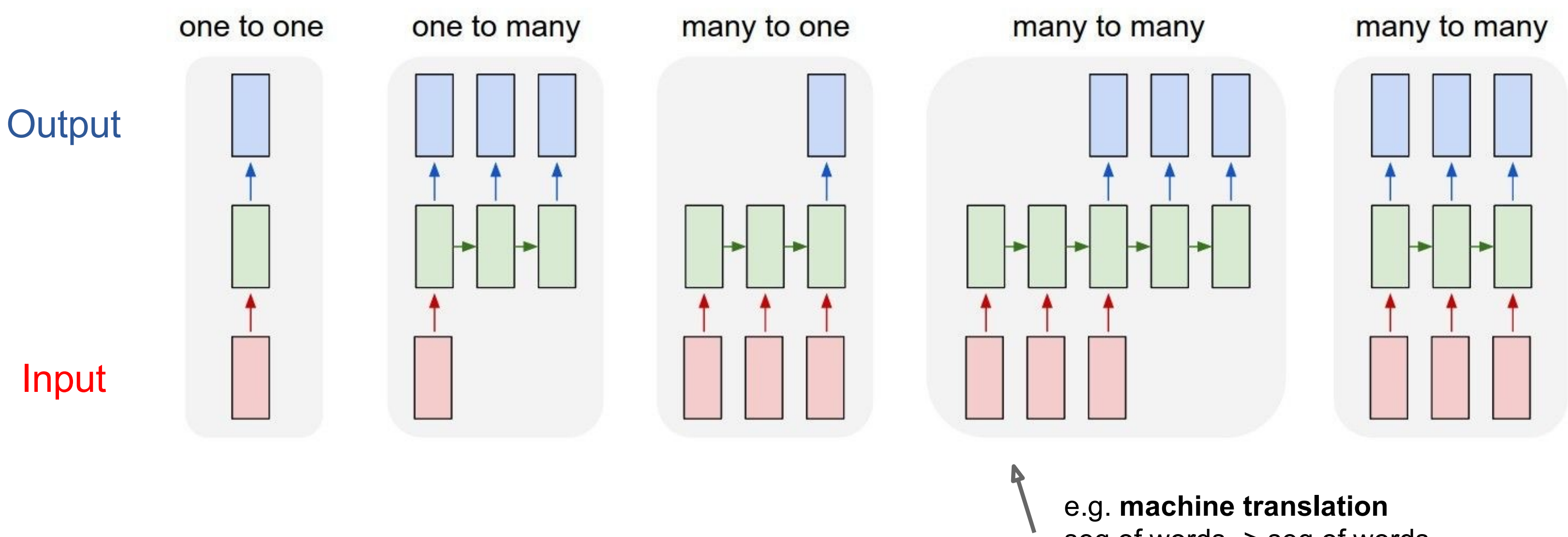
Recurrent Networks offer a lot of flexibility:



e.g. **sentiment classification**
sequence of words -> sentiment

Andrej Karpathy

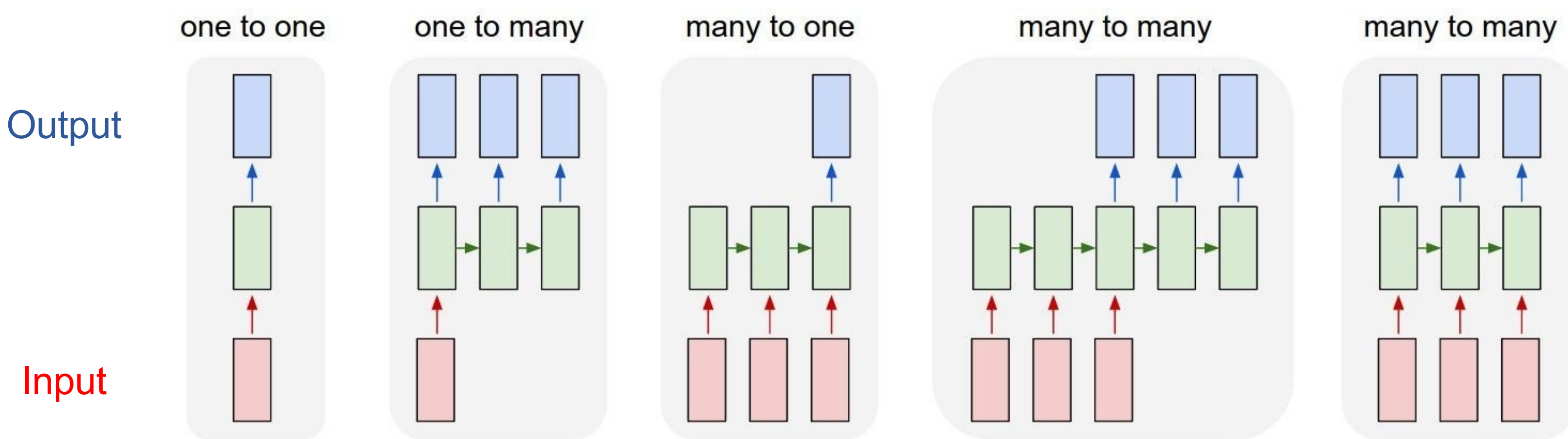
Recurrent Networks offer a lot of flexibility:



e.g. **machine translation**
seq of words -> seq of words

Andrej Karpathy

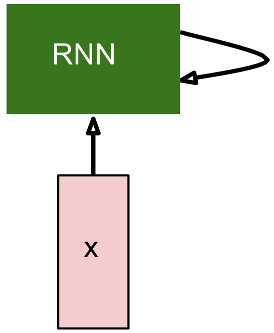
Recurrent Networks offer a lot of flexibility:



e.g. **video classification on frame level**

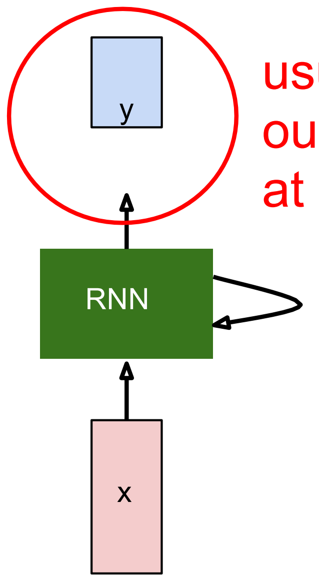
Andrej Karpathy

Recurrent Neural Network



Andrej Karpathy

Recurrent Neural Network



usually want to
output a prediction
at some time steps

Adapted from Andrej Karpathy

Recurrent Neural Network

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

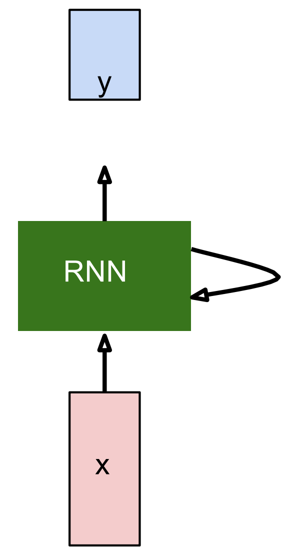
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



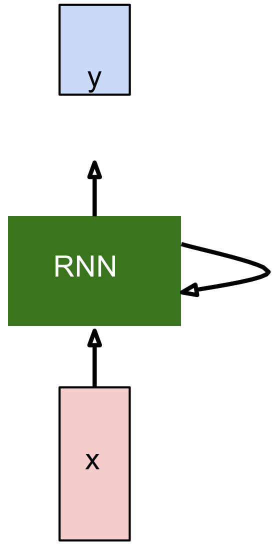
Andrej Karpathy

Recurrent Neural Network

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

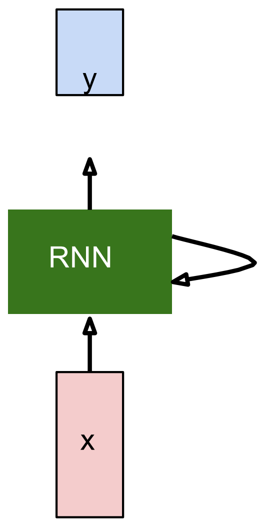
$$h_t = f_W(h_{t-1}, x_t)$$

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



Andrej Karpathy

(Vanilla) Recurrent Neural Network



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

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

$$y_t = \boxed{W_{hy}} h_t$$

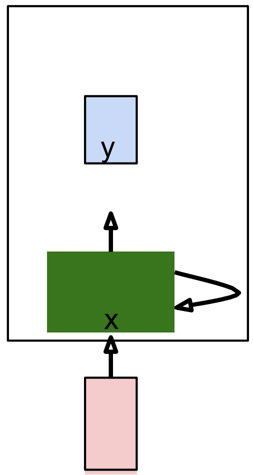
Andrej Karpathy

Example

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



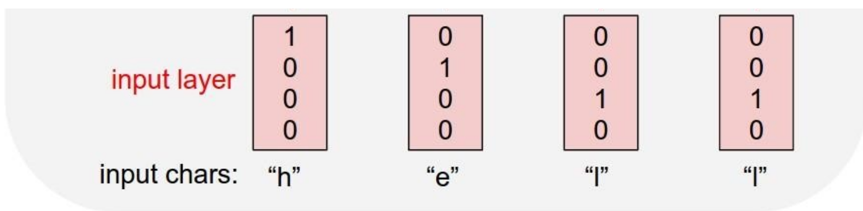
Andrej Karpathy

Example

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**



Andrej Karpathy

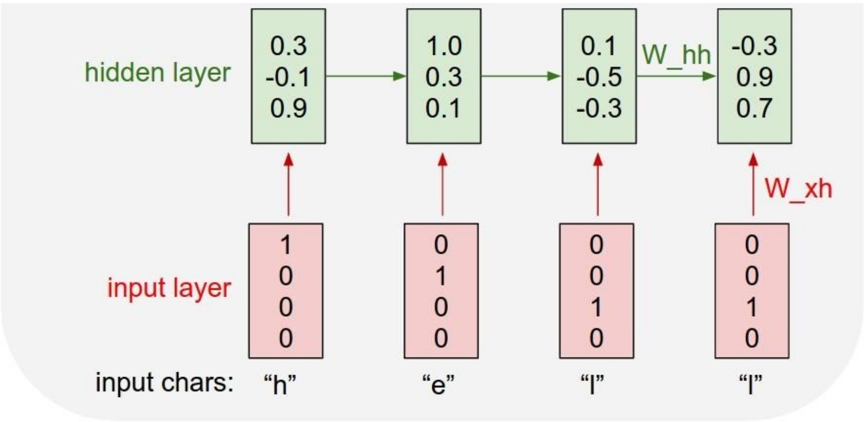
Example

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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



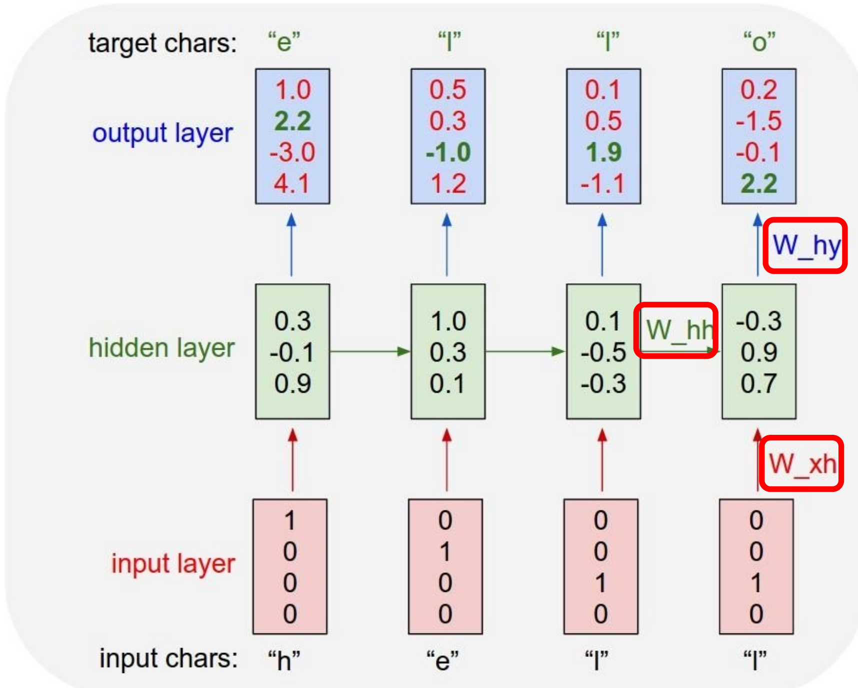
Andrej Karpathy

Example

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Andrej Karpathy

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.

Note: The animations below are videos. Touch or hover on them (if you're using a mouse) to get play controls so you can pause if needed.

Sequence-to-sequence models are deep learning models that have achieved a lot of success in tasks like machine translation, text summarization, and image captioning. Google Translate started using such a model in production in late 2016. These models are explained in the two pioneering papers (Sutskever et al., 2014, Cho et al., 2014).

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

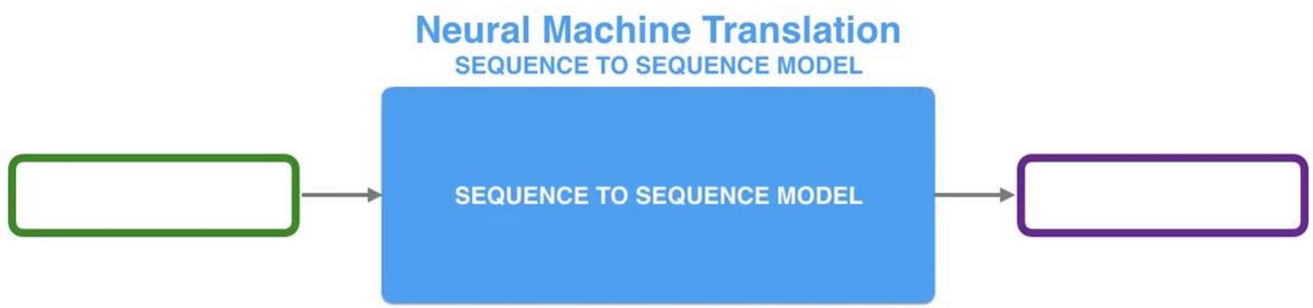
Slide Credit: Prof. Sandra Avila - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

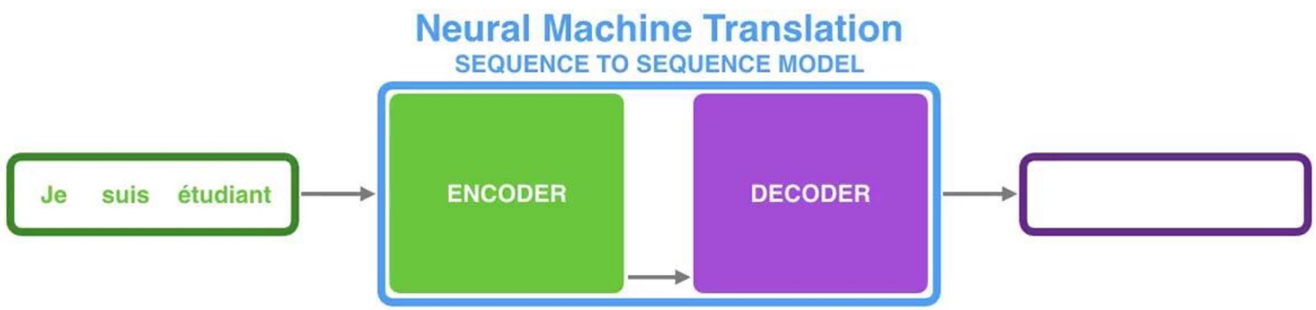
Slide Credit: Prof. Sandra Avila - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

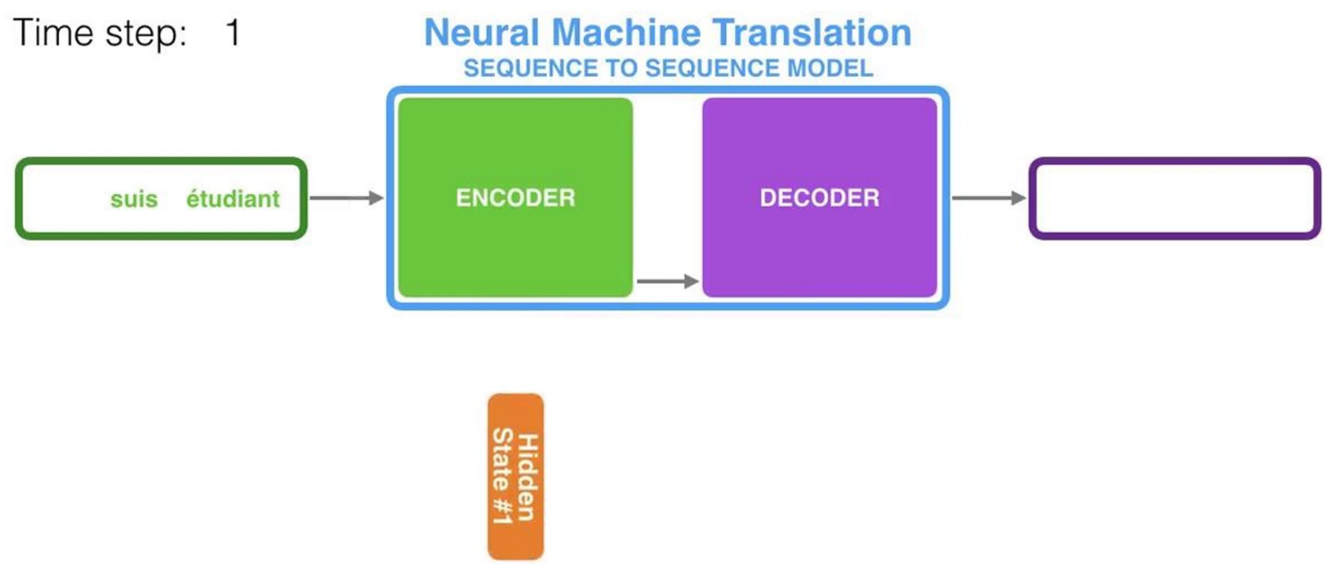
Slide Credit: Prof. Sandra Avila - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

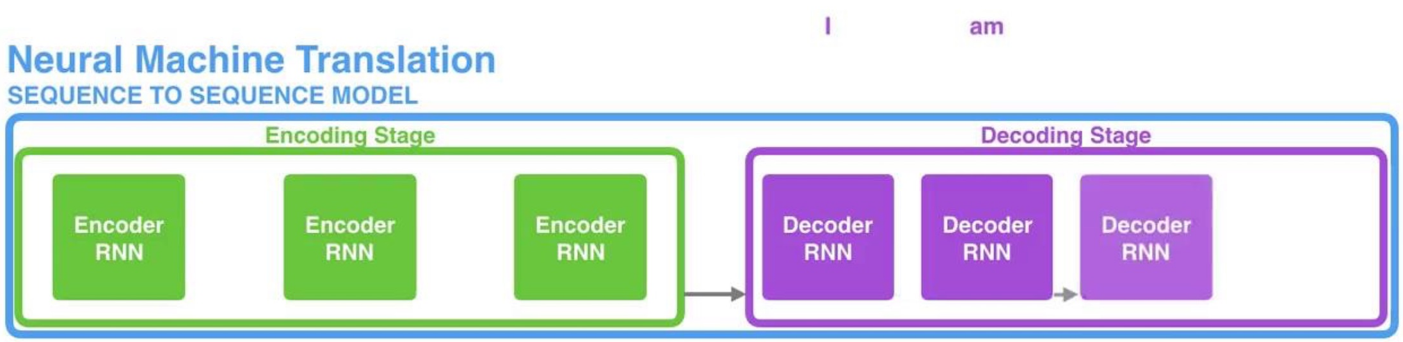
Slide Credit: Prof. Sandra Avila - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

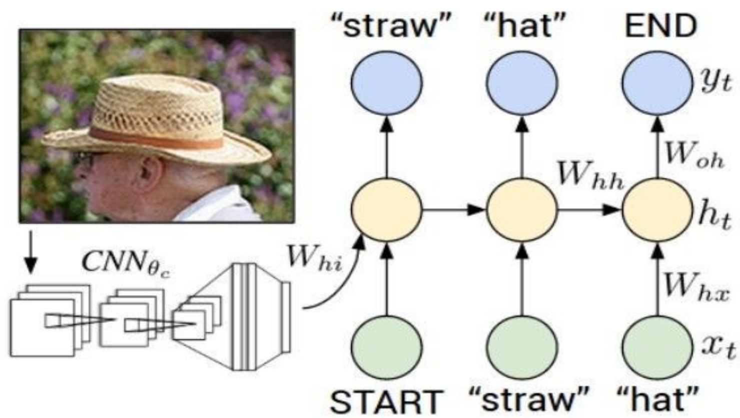
May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Slide Credit: Prof. Sandra Avila - UNICAMP

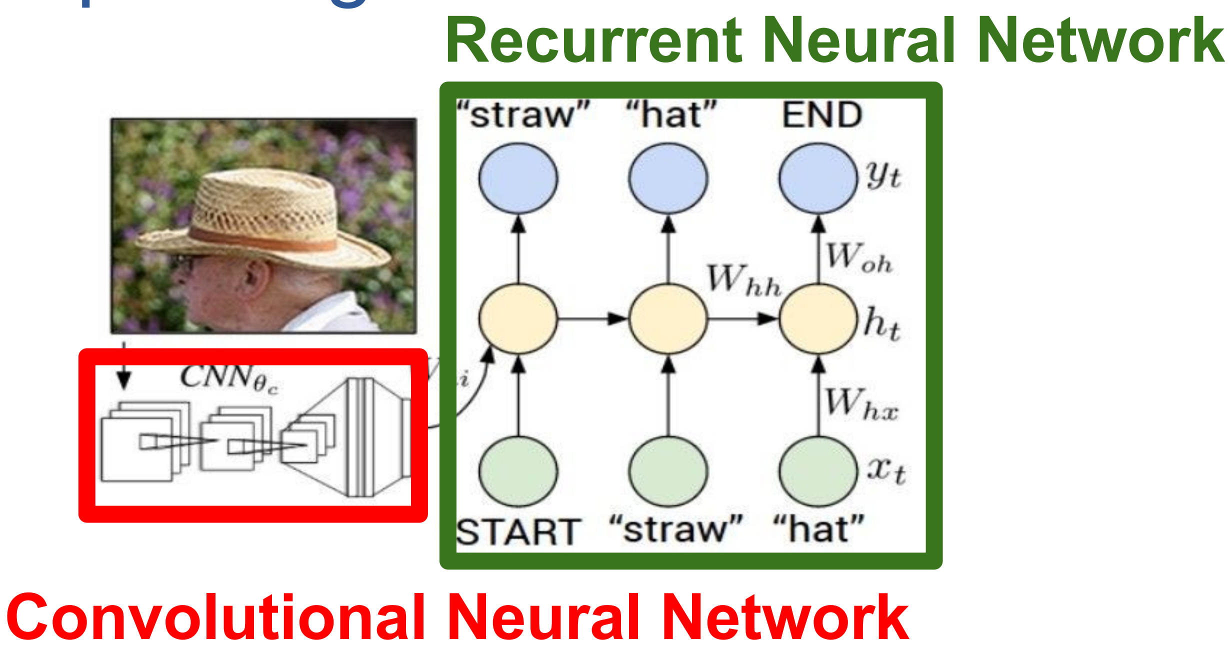
Image Captioning



CVPR 2015:
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

Adapted from Andrej Karpathy

Image Captioning

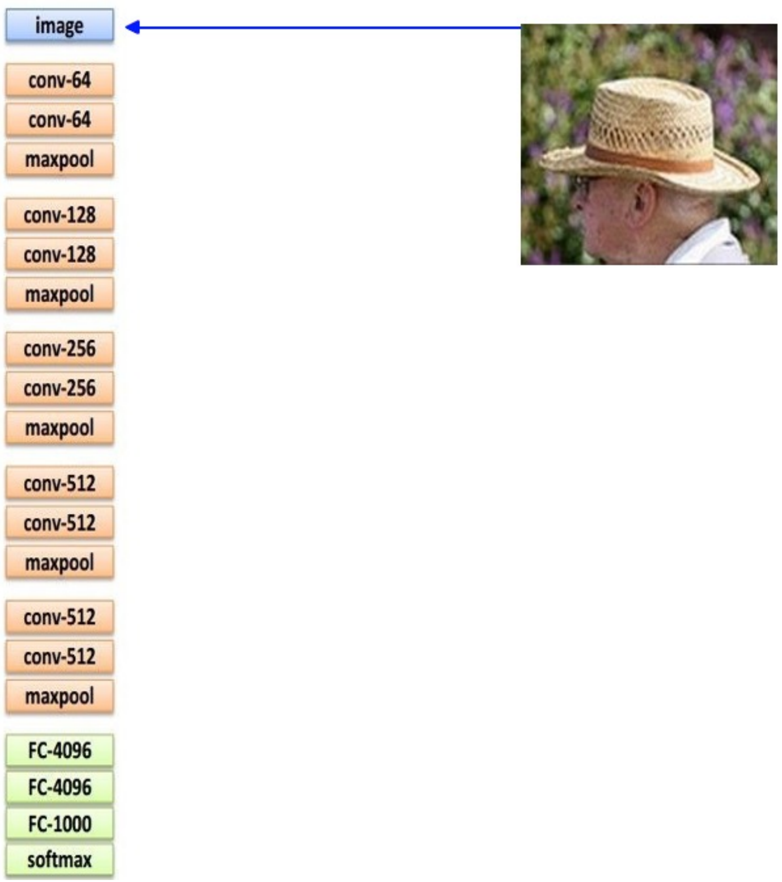


Andrej Karpathy

Image Captioning

test image

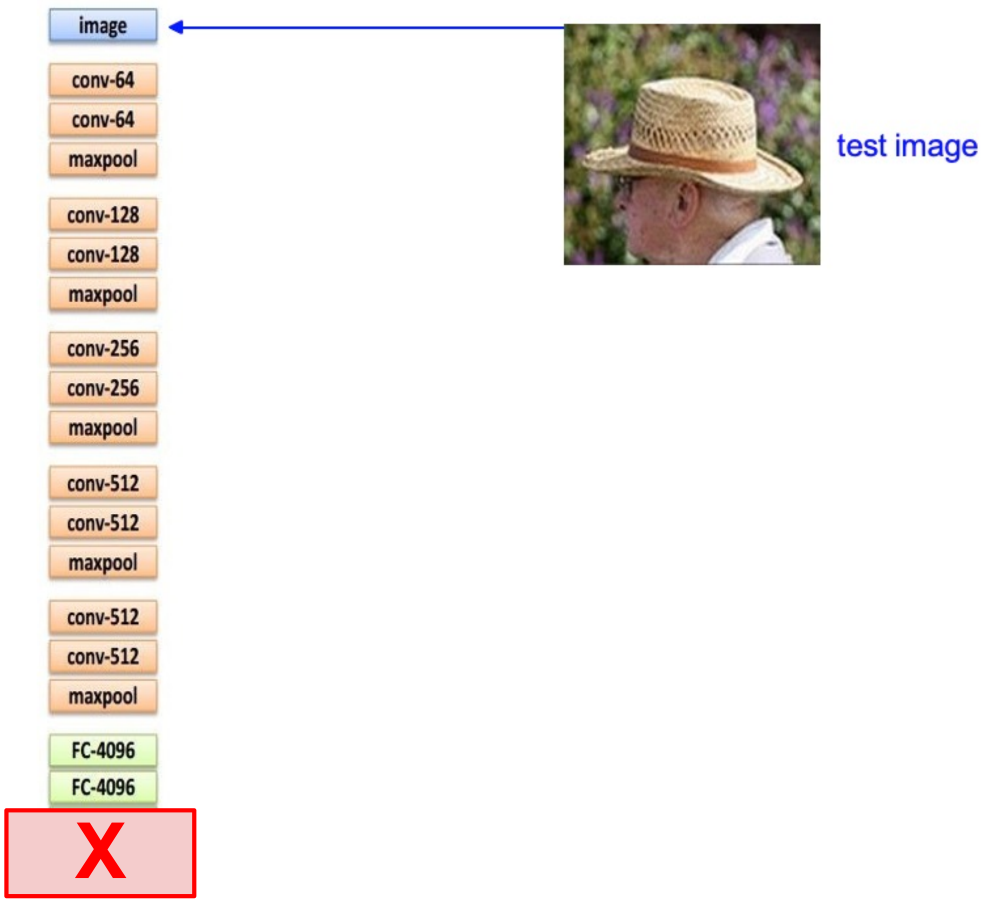
Image Captioning



test image

Andrej Karpathy

Image Captioning



Andrej Karpathy

Image Captioning

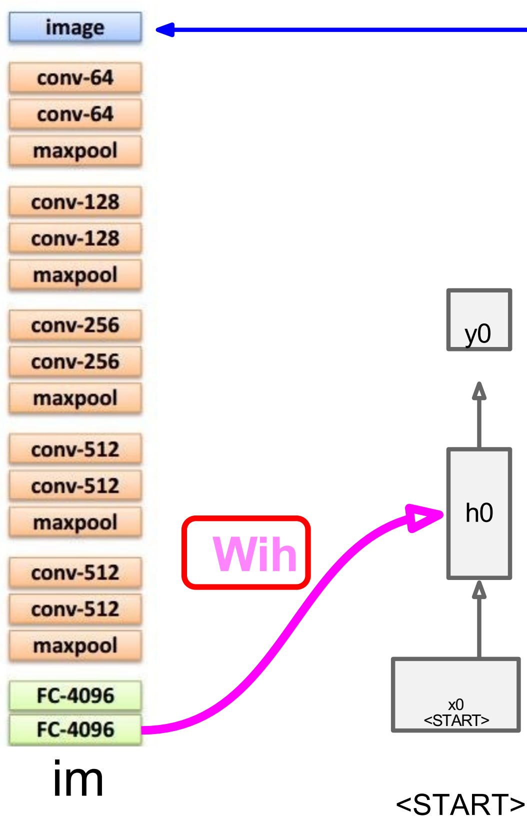
test image

x0
<START>

<START>

Andrej Karpathy

Image Captioning



test image

before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + \boxed{W_{ih} * im})$$

Andrej Karpathy

Image Captioning

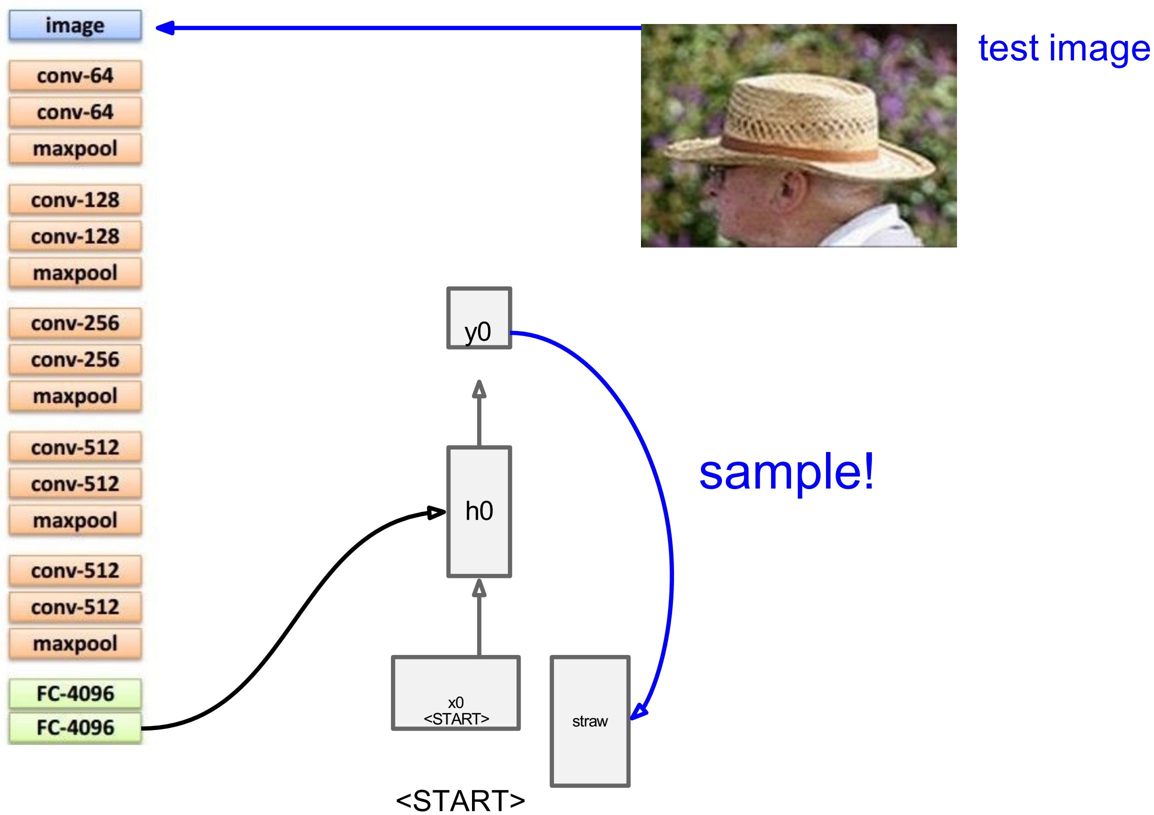
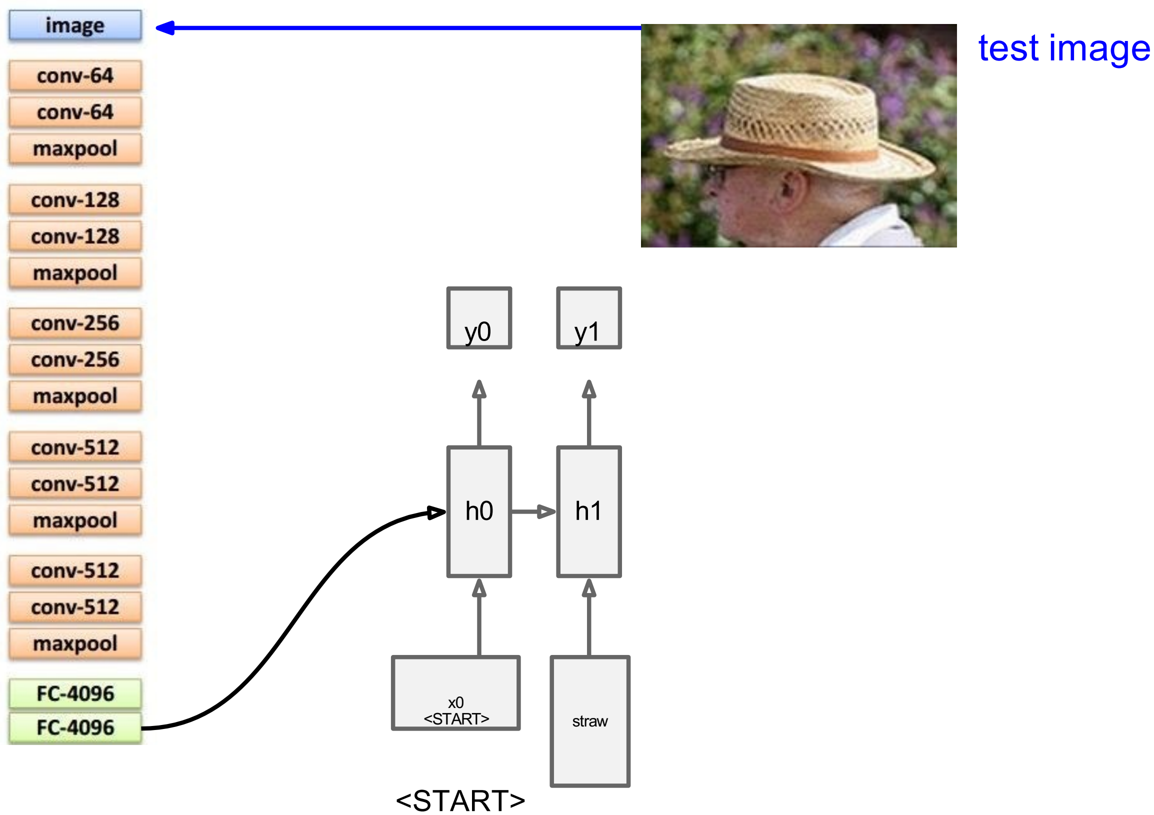
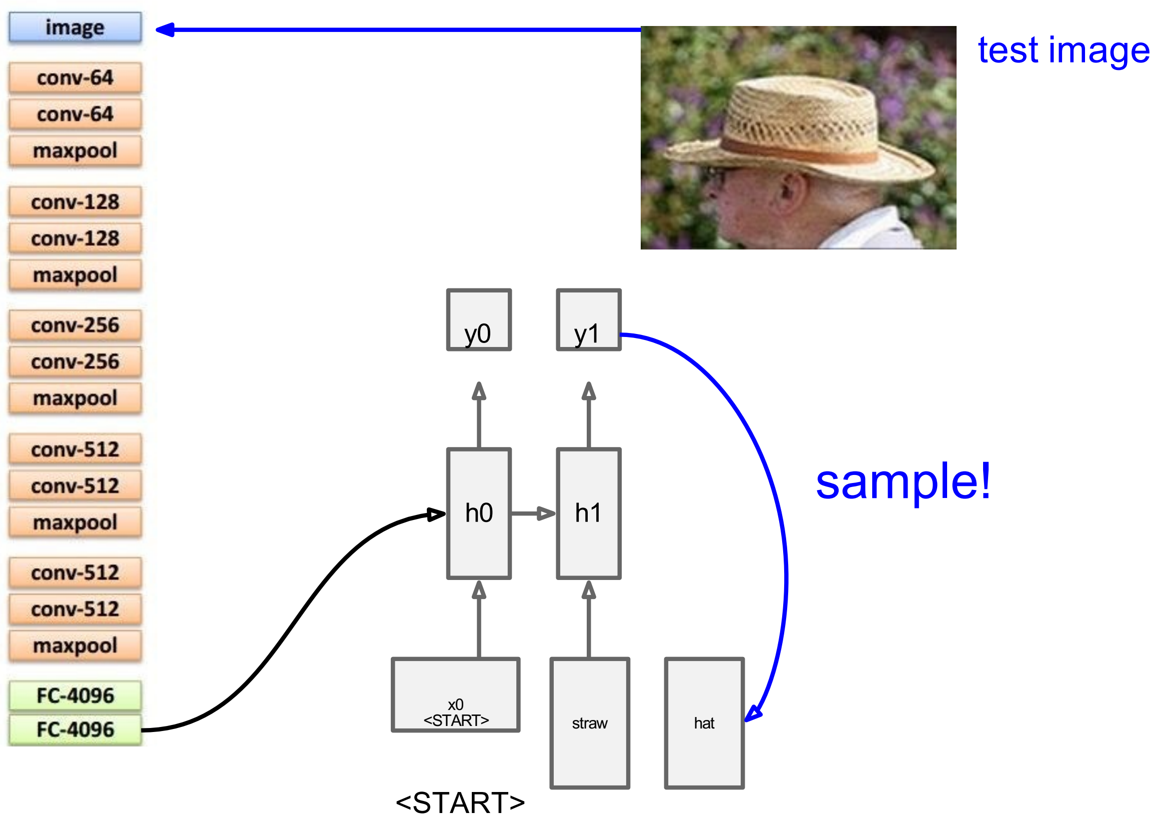


Image Captioning



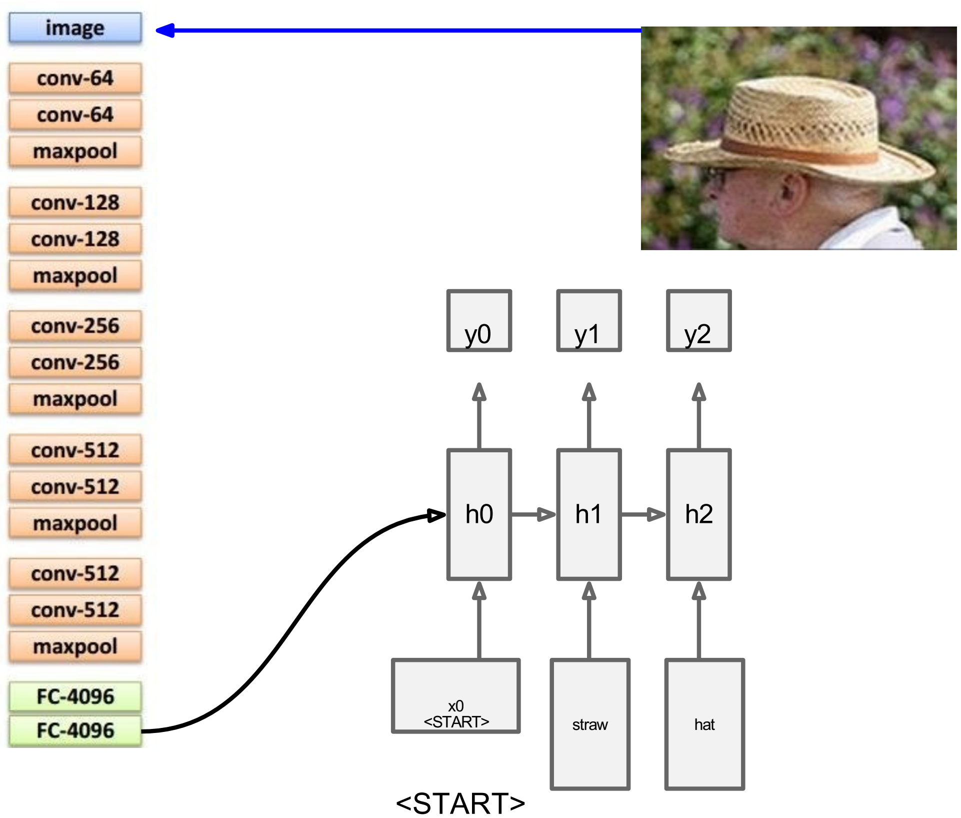
Andrej Karpathy

Image Captioning



Andrej Karpathy

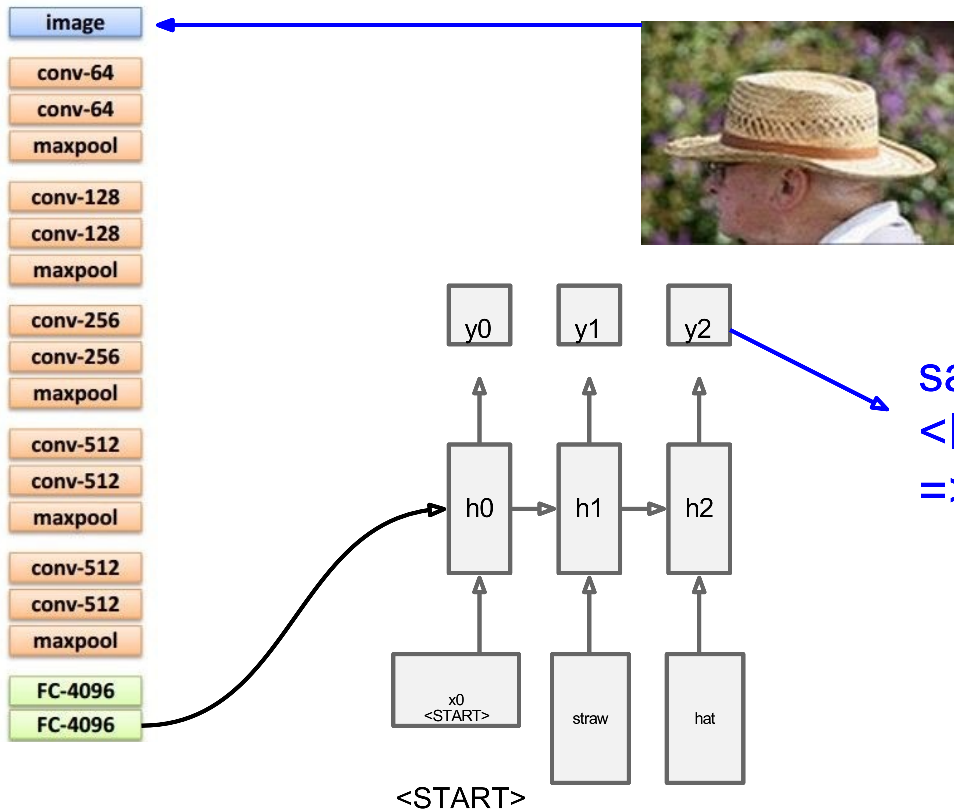
Image Captioning



test image

Andrej Karpathy

Image Captioning



test image

Caption generated:
"straw hat"

sample
<END> token
=> finish.

Adapted from Andrej Karpathy

Image Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

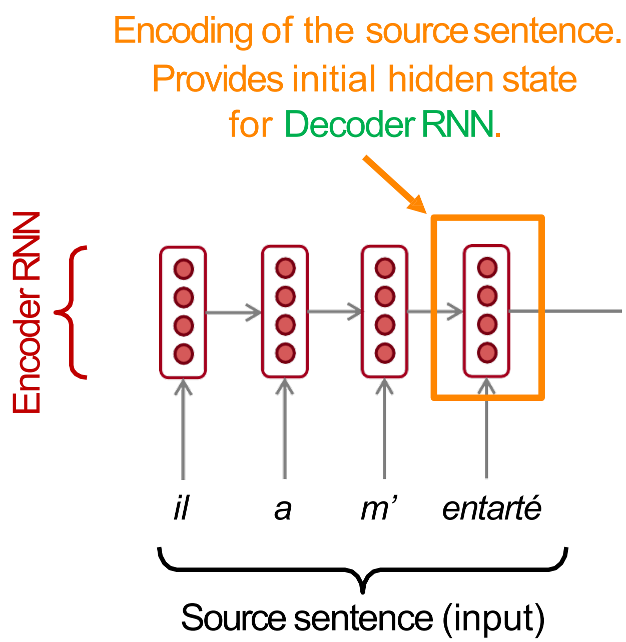
"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

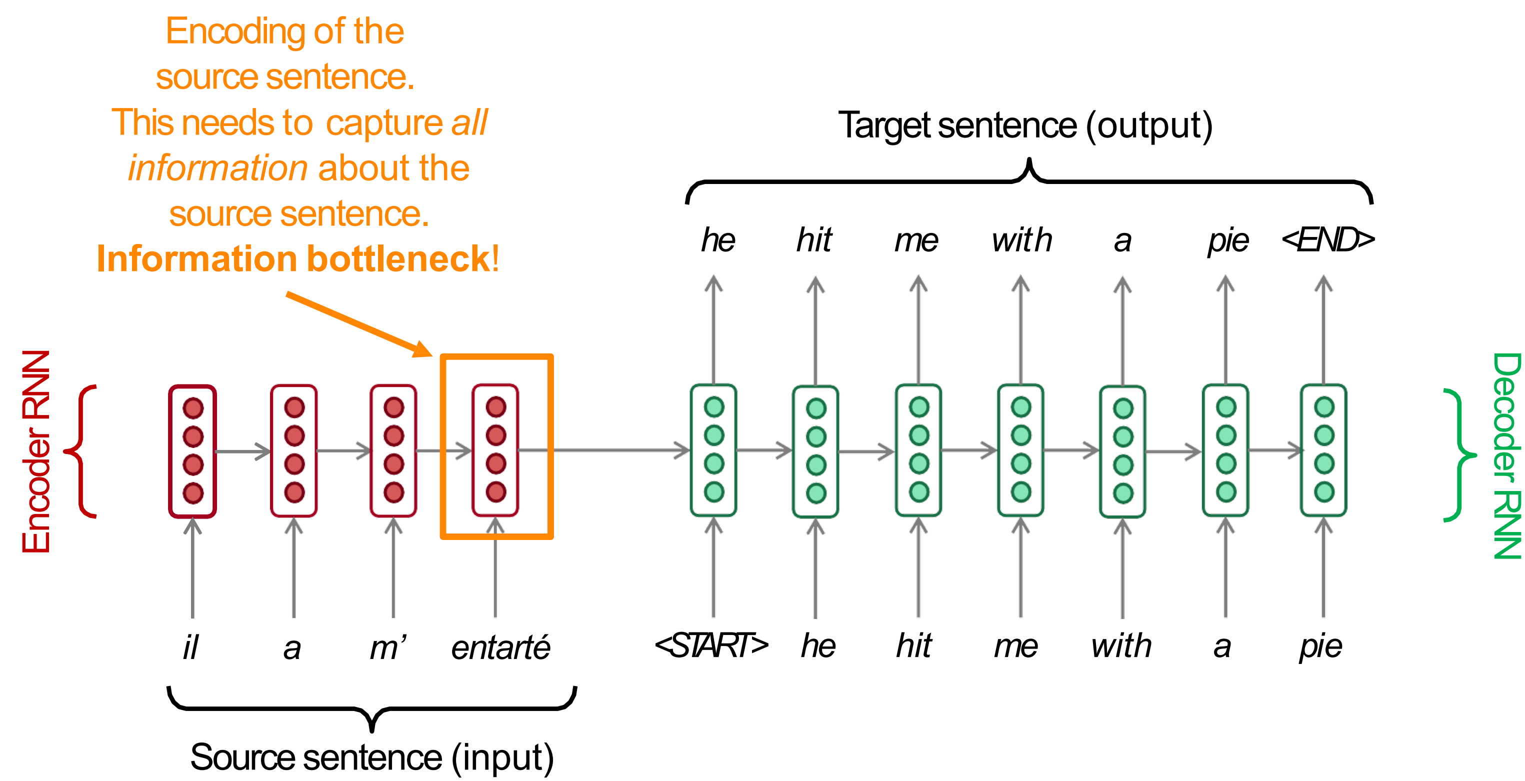
Andrej Karpathy

Neural Machine Translation (NMT)

The sequence-to-sequence model



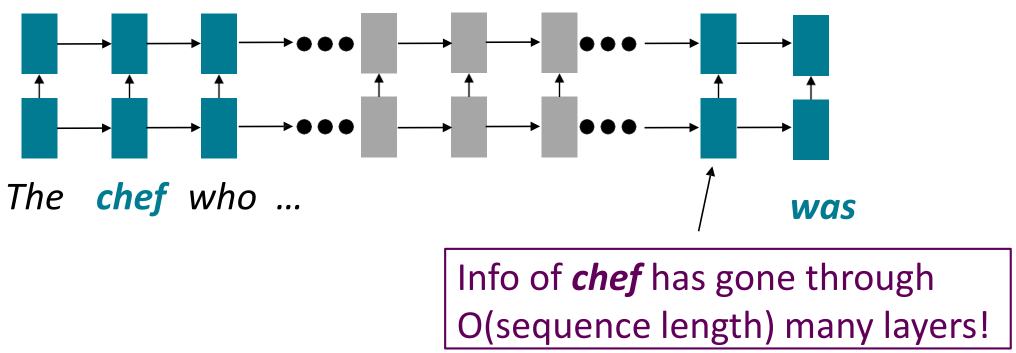
Sequence-to-sequence: the bottleneck problem



Abigail See

Issues with recurrent models: Linear interaction distance

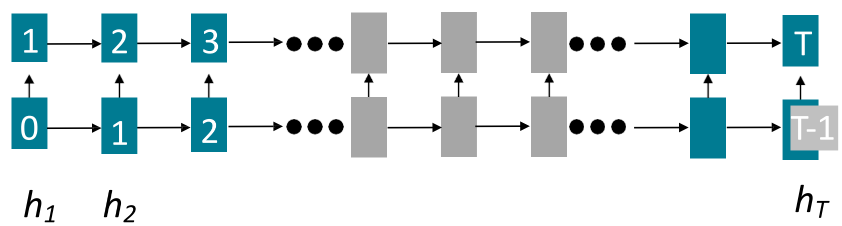
- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is “baked in”; not necessarily the right way to think about sentences…



Adapted from John Hewitt

Issues with recurrent models: Lack of parallelization

- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of 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!



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

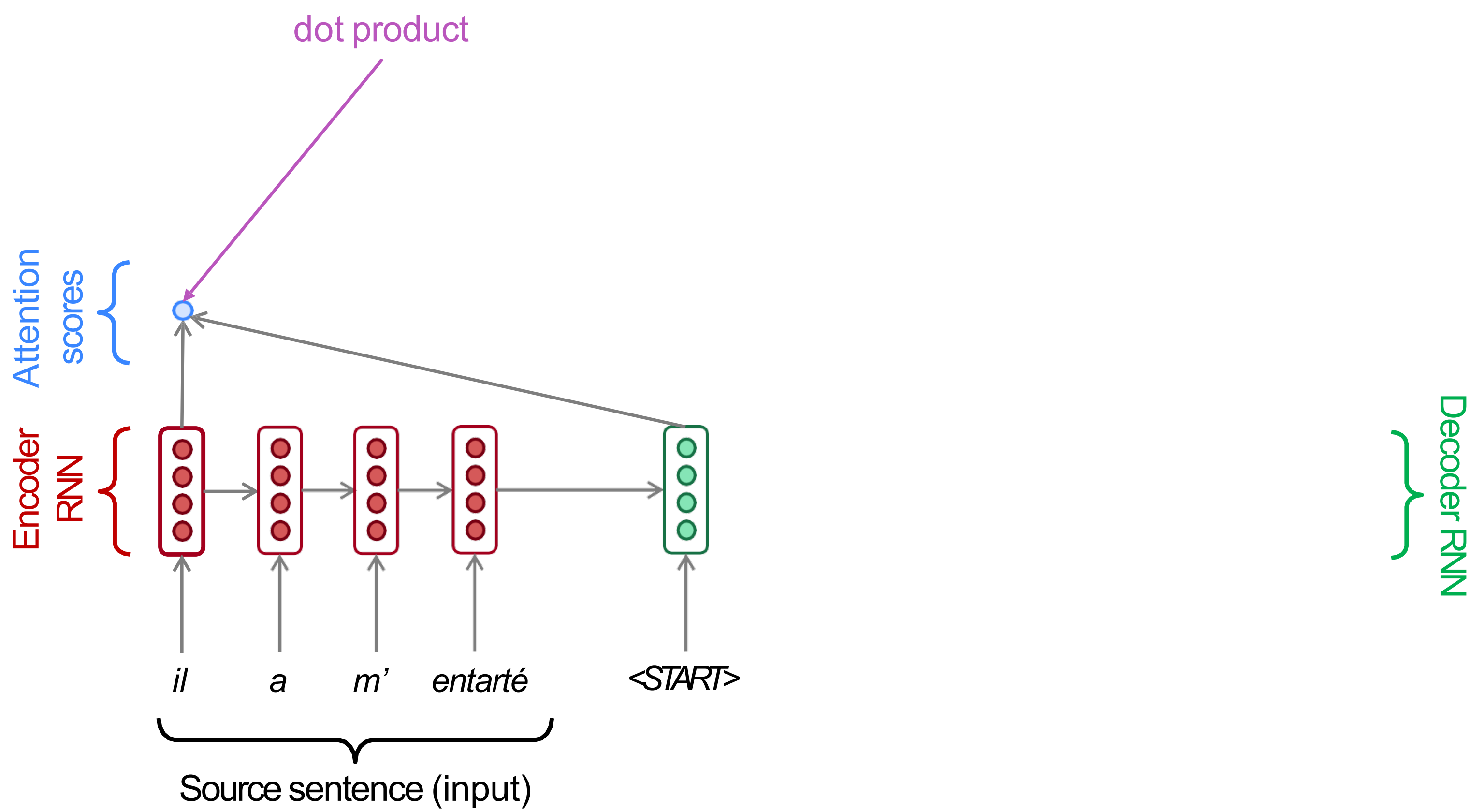
Attention

- 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

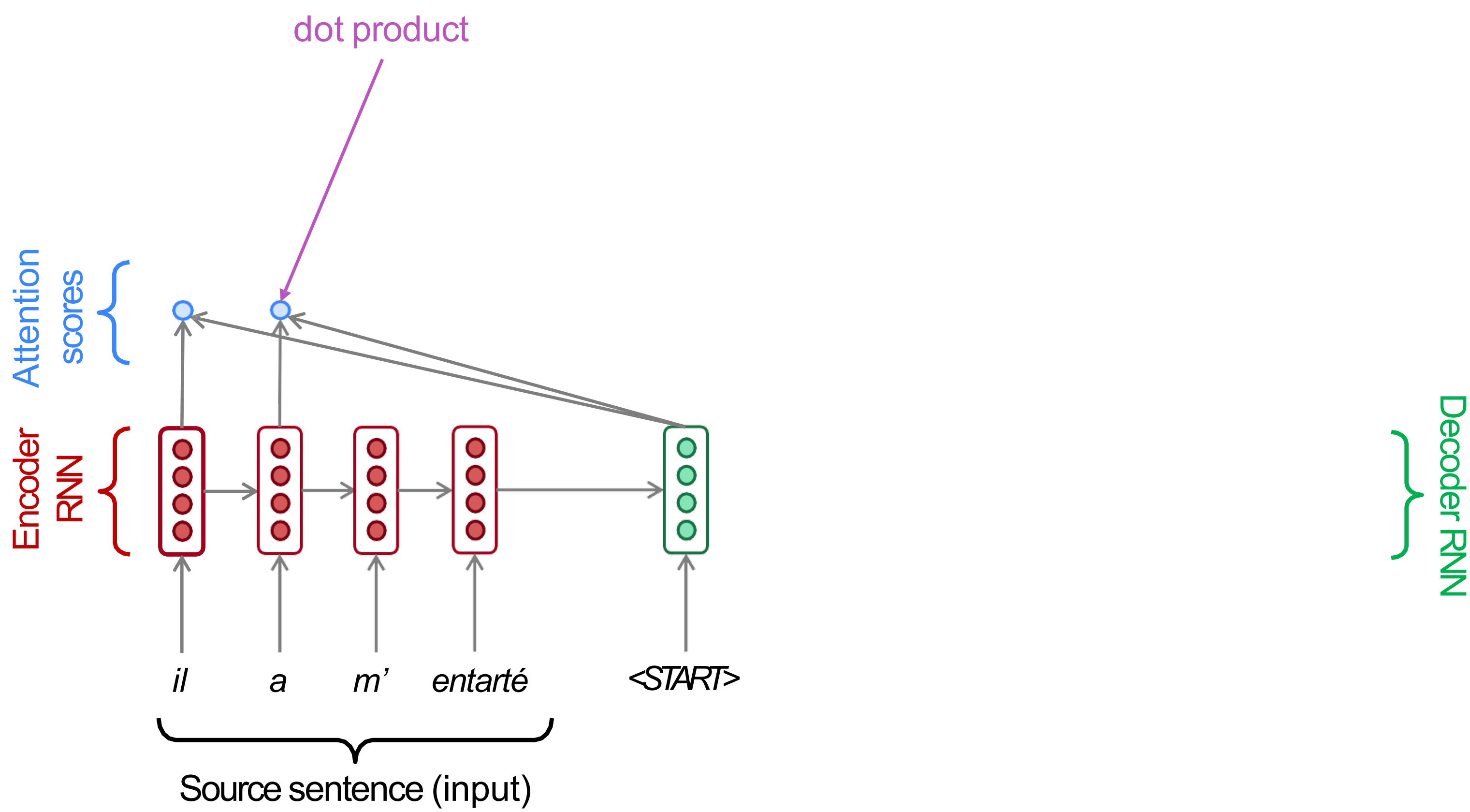
- First, we will show via diagram (no equations), then we will show with equations

Sequence-to-sequence with attention



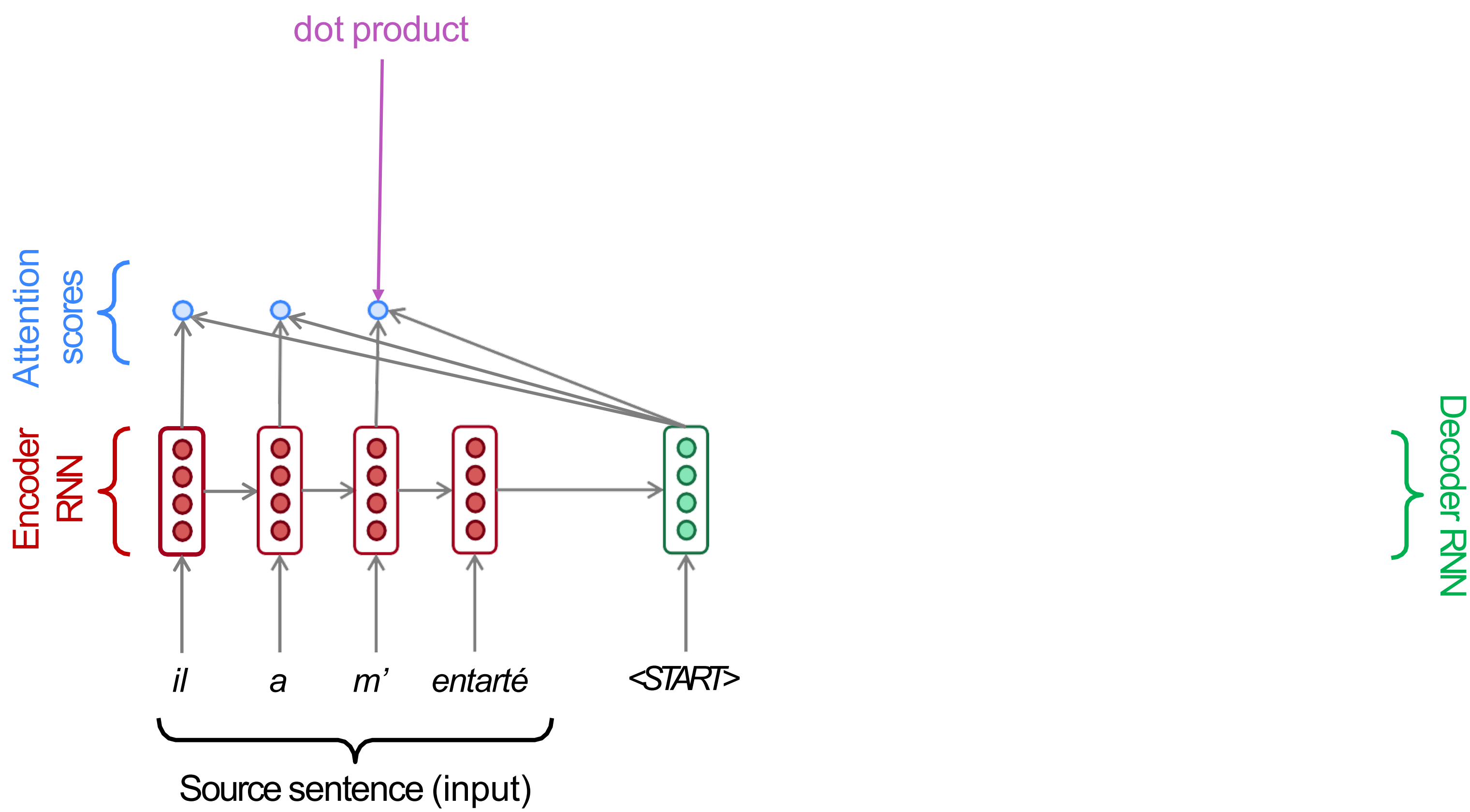
Abigail See

Sequence-to-sequence with attention



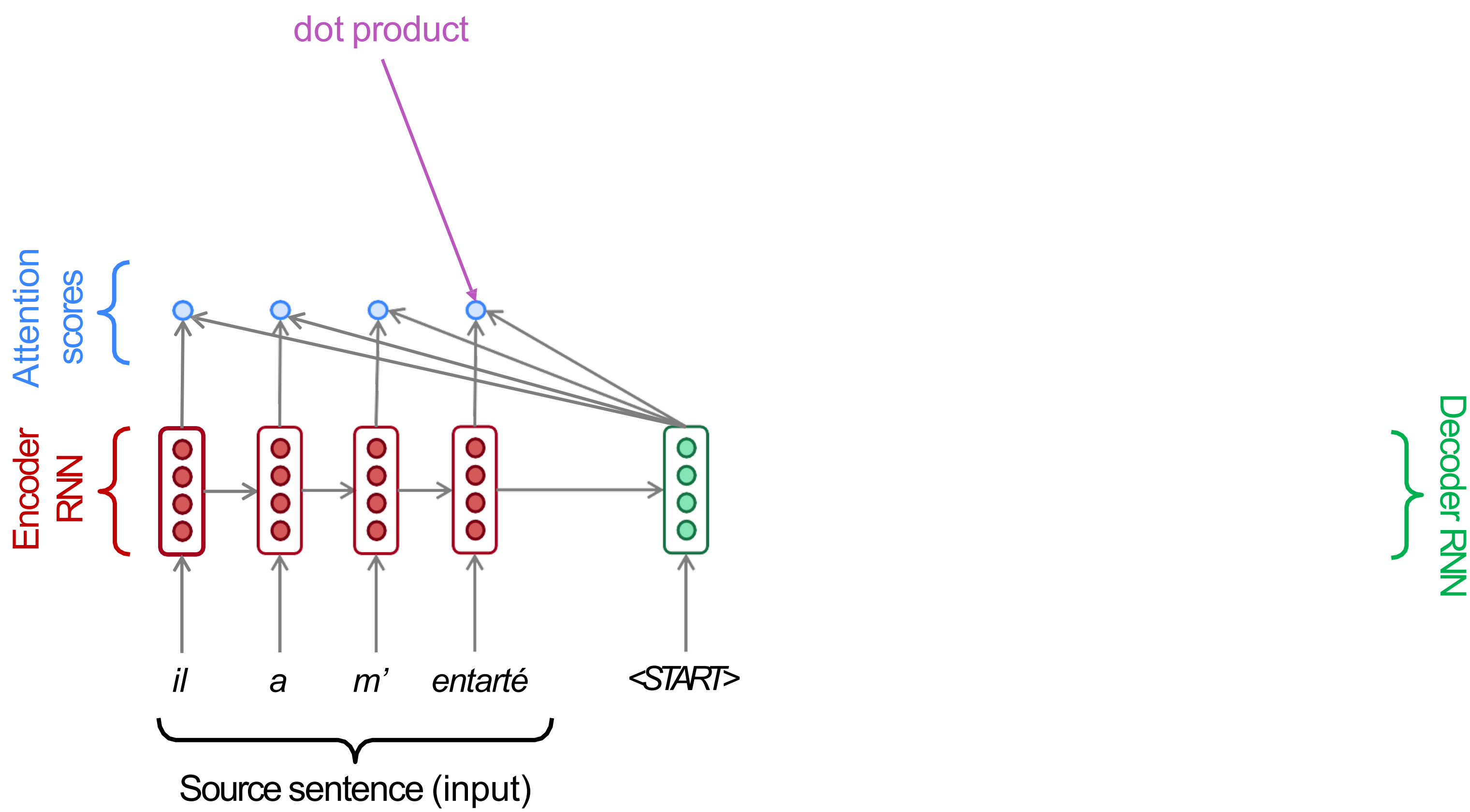
Abigail See

Sequence-to-sequence with attention



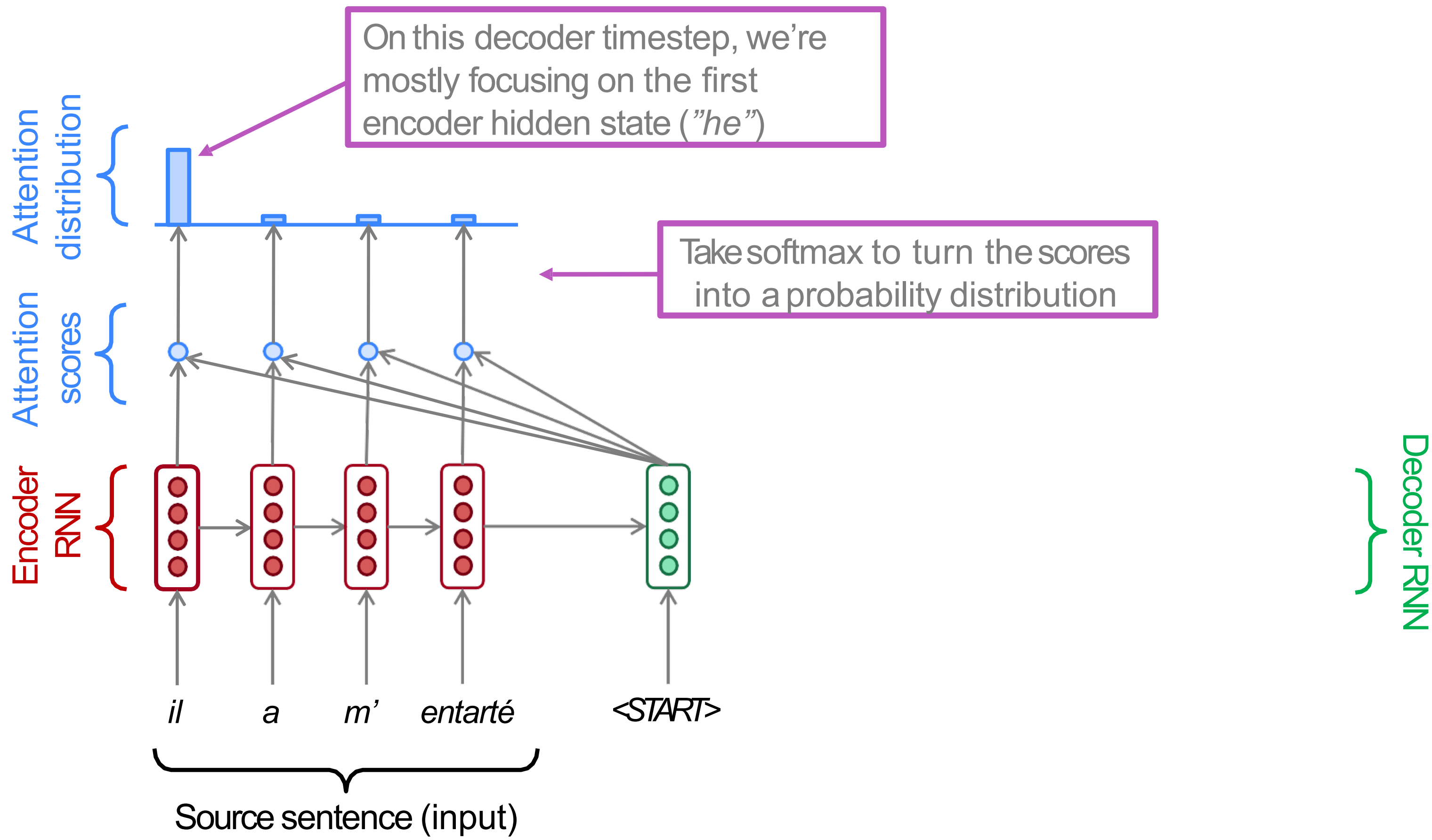
Abigail See

Sequence-to-sequence with attention



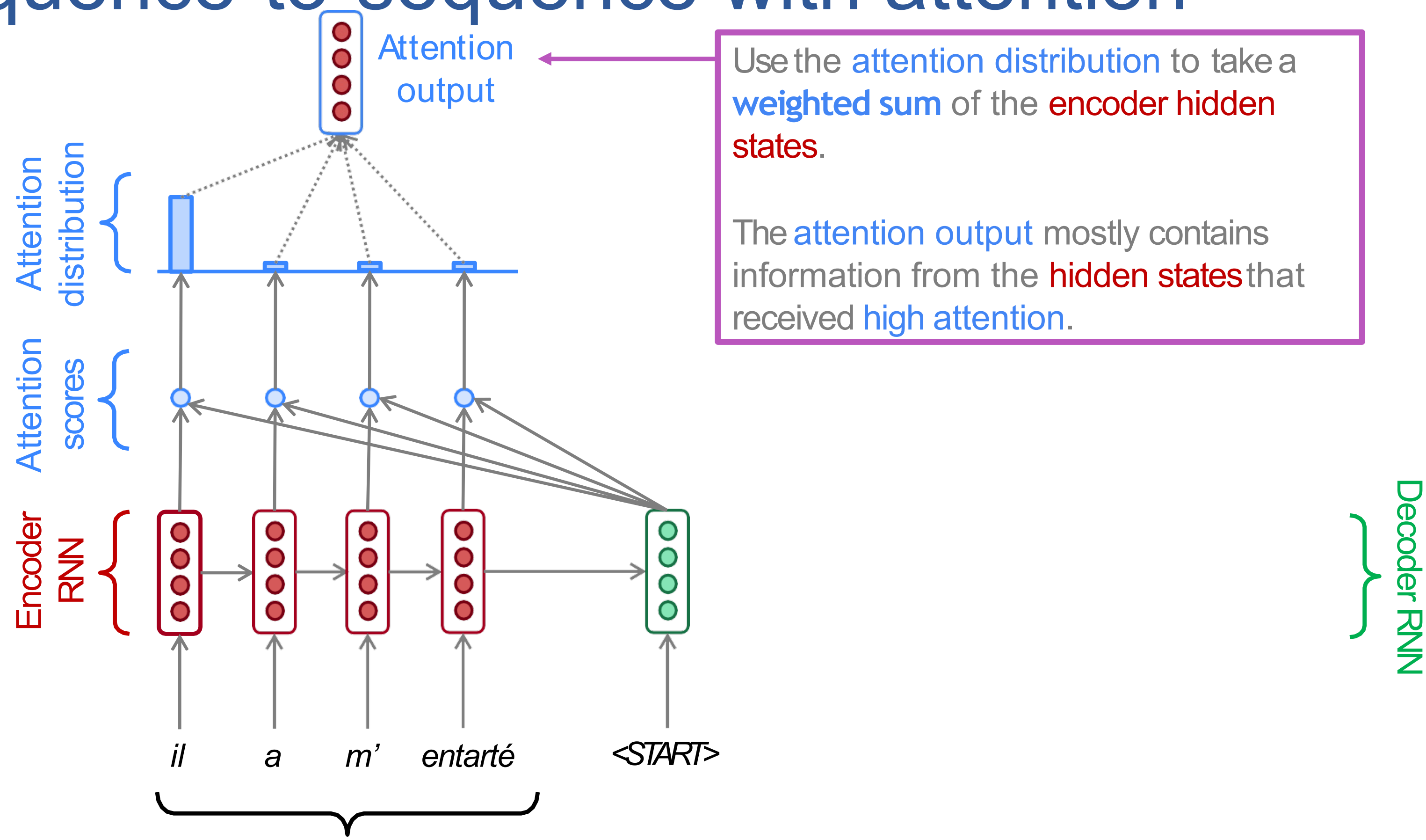
Abigail See

Sequence-to-sequence with attention



Abigail See

Sequence-to-sequence with attention

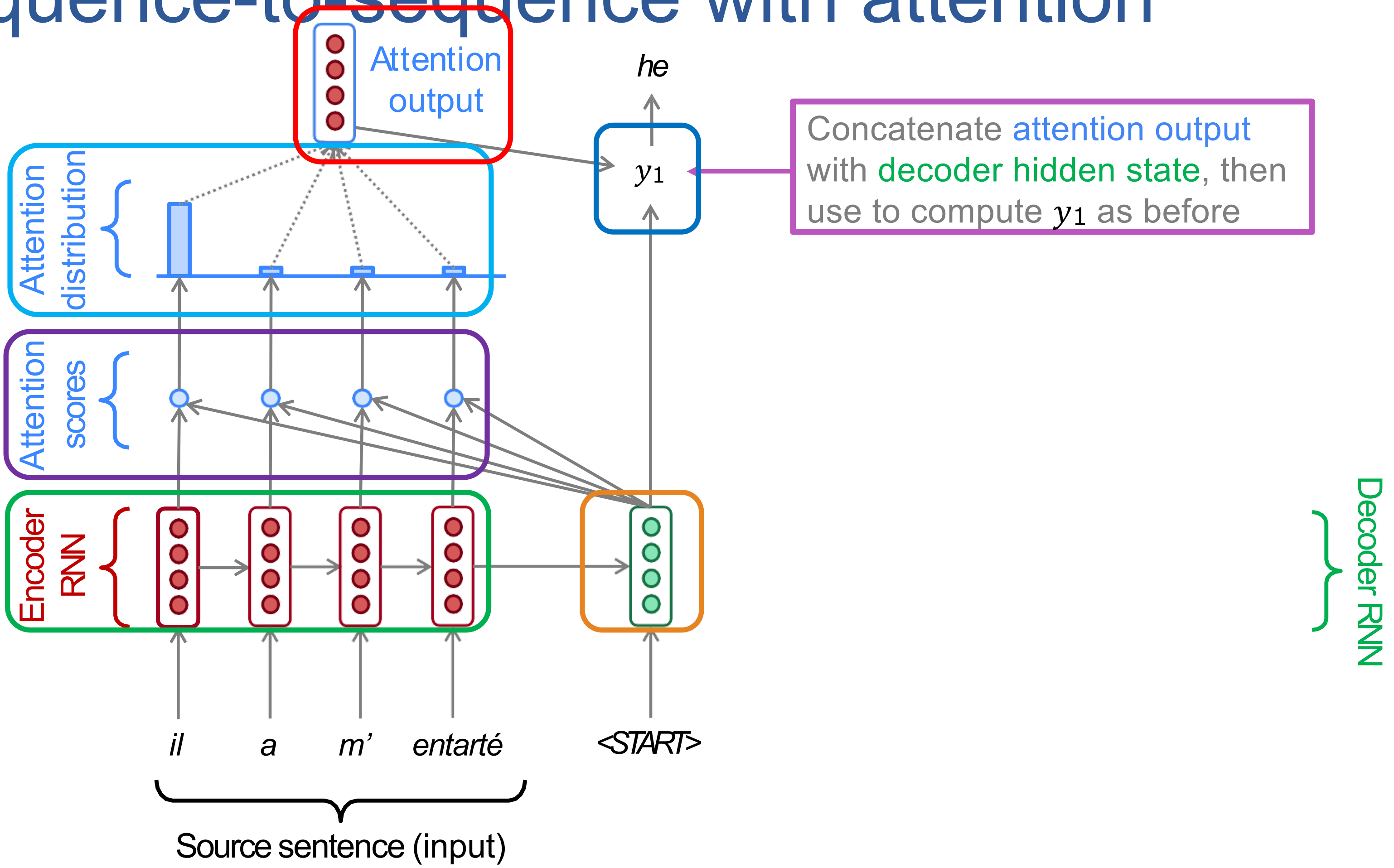


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

Abigail See

Sequence-to-sequence with attention



Concatenate attention output with decoder hidden state, then use to compute y_1 as before

Attention in equations

- 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 $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

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

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

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

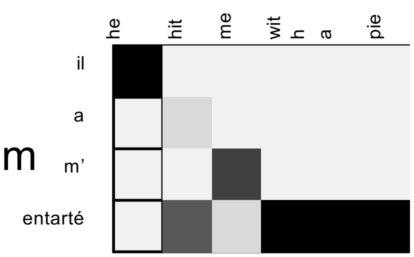
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally, we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is great!

- Attention significantly improves Neural Machine Trans. (NMT) performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can 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



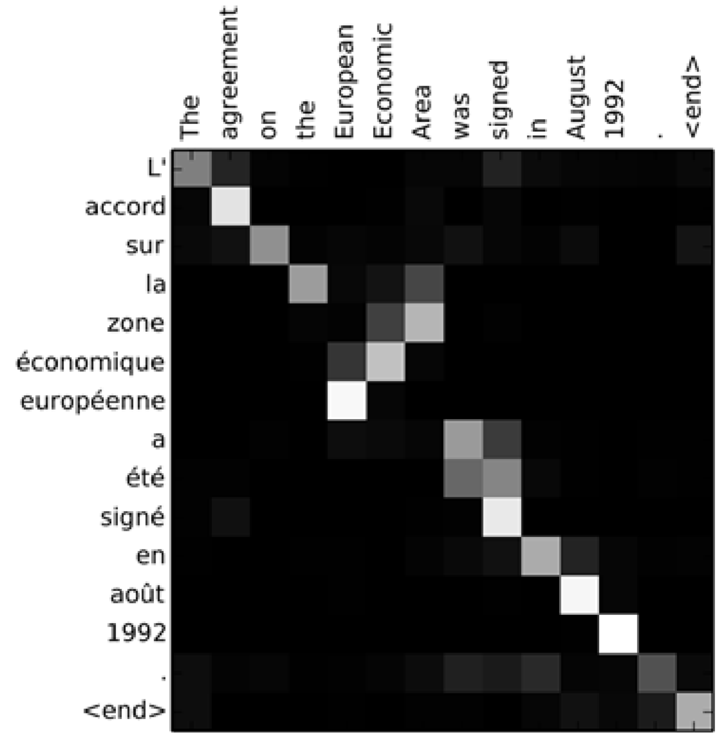
Attention is great: Example

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



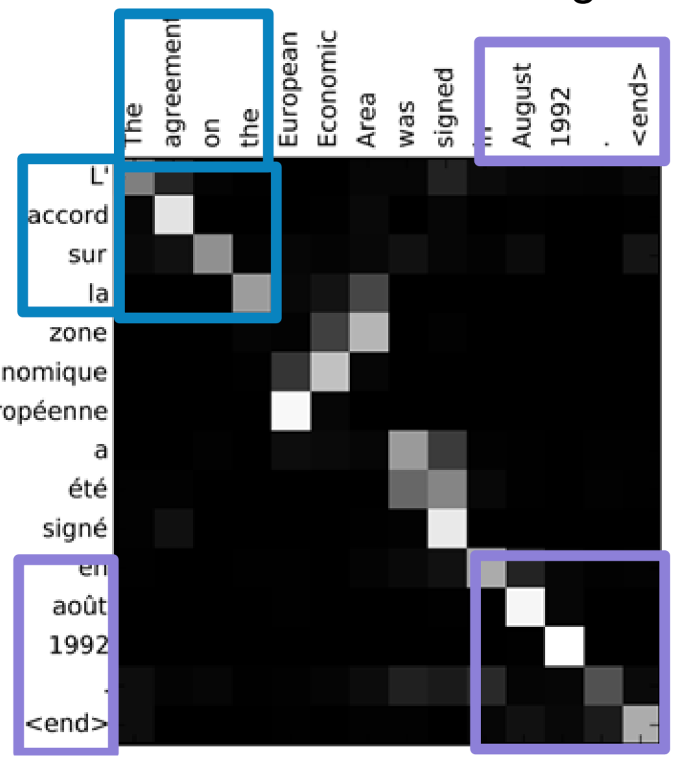
Attention is great: Example

Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed in **August 1992.**"

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992.**"

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

https://cs231n.stanford.edu/

Attention is great: Example

Example: English to French translation

Input: "**The agreement on the European Economic Area** was signed in **August 1992.**"

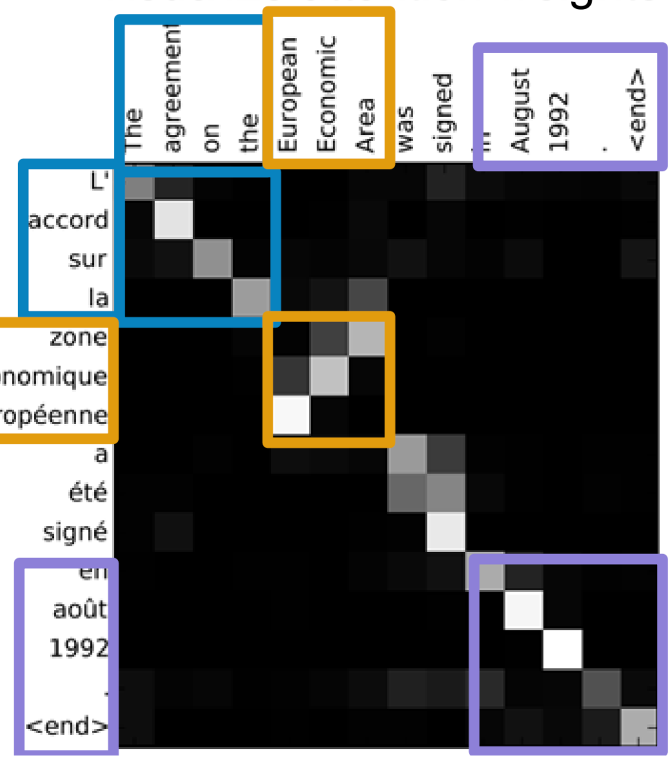
Output: "**L'accord sur la zone économique européenne** a été signé **en août 1992.**"

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

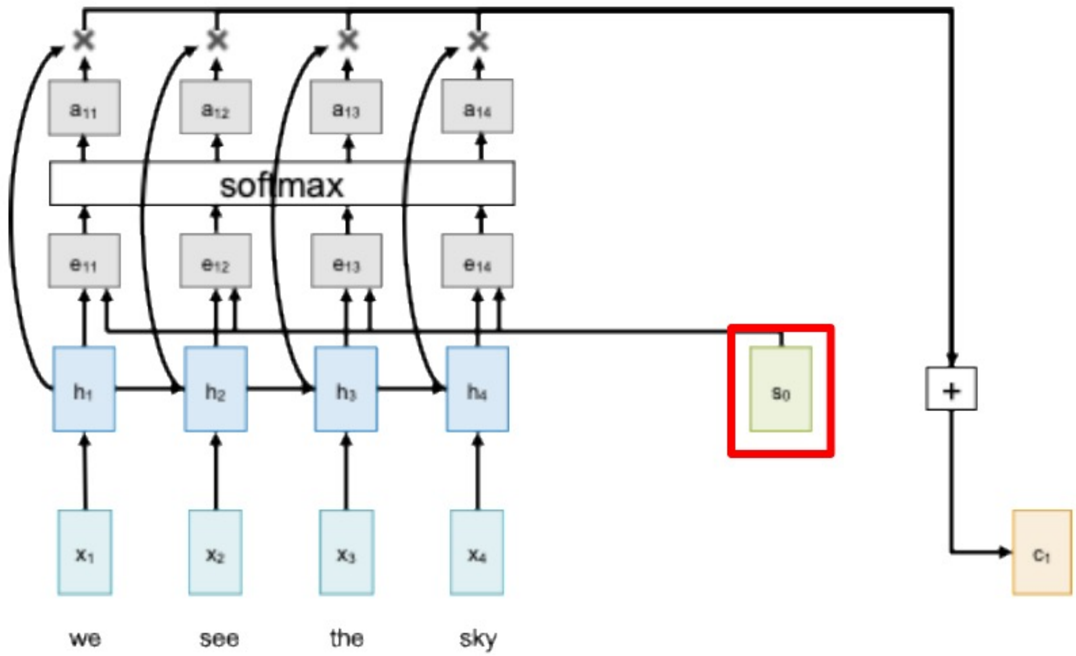
Attention figures out other word orders

Diagonal attention means words correspond in order



Attention Example

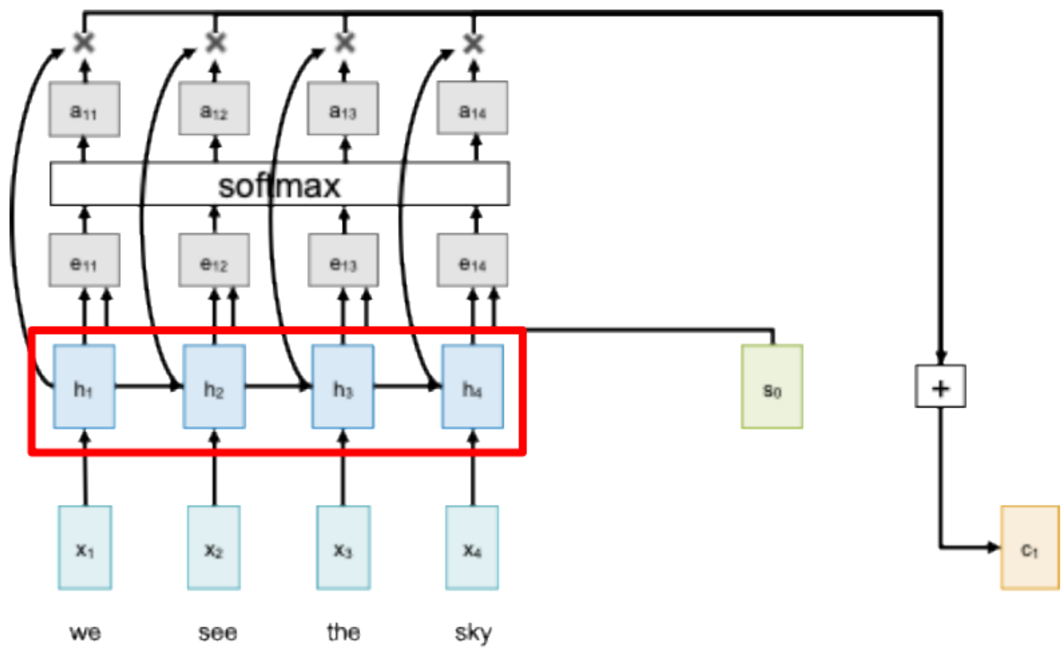
Inputs:
Query vector: **q** [D_Q]



https://cs231n.stanford.edu/

Attention Example

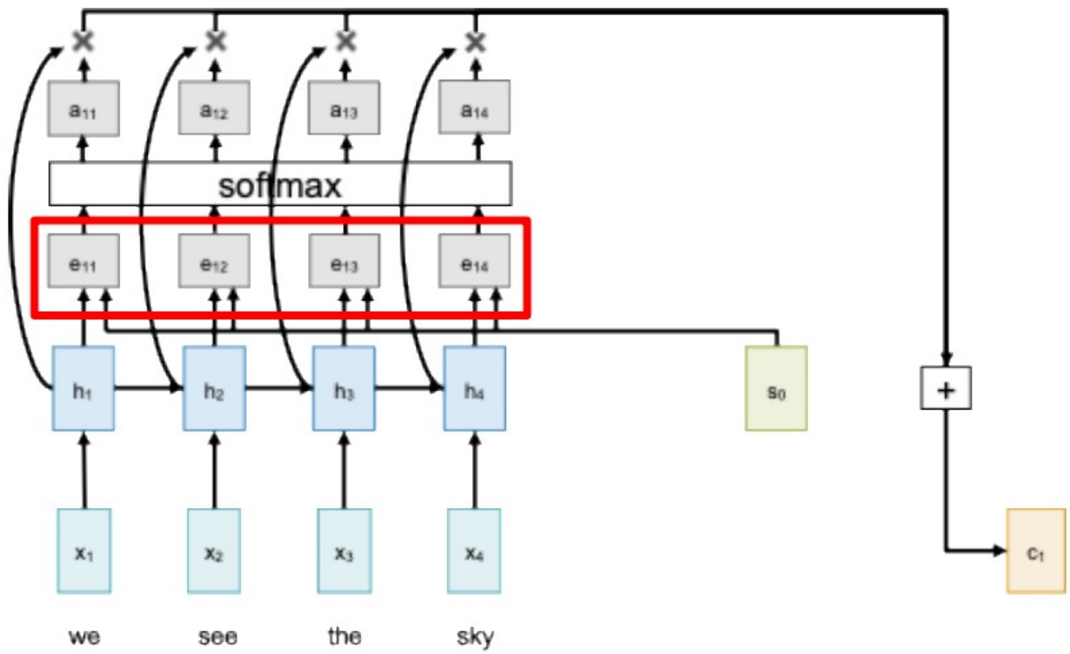
Inputs:
Query vector: q [D_Q]
Data vectors: X [N_X x D_X]



https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: $\mathbf{q}$ [D_Q]
Data vectors: $\mathbf{X}$ [N_X x D_X]

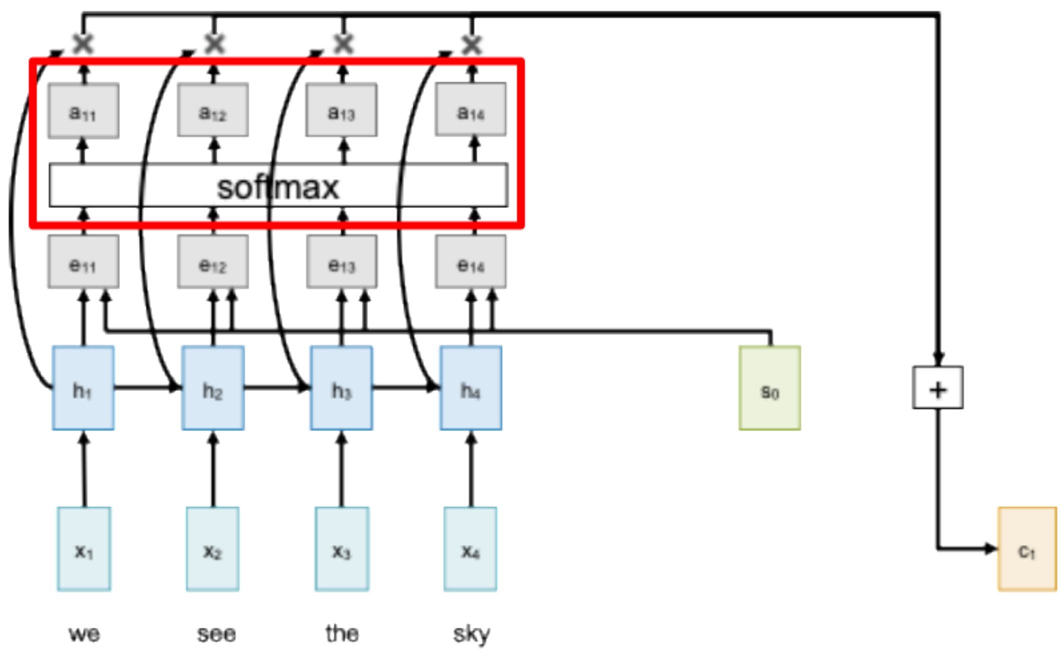


Computation:
Similarities: e [N_X] $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: $\mathbf{q}$ $[D_Q]$
Data vectors: $\mathbf{X}$ $[N_X \times D_X]$

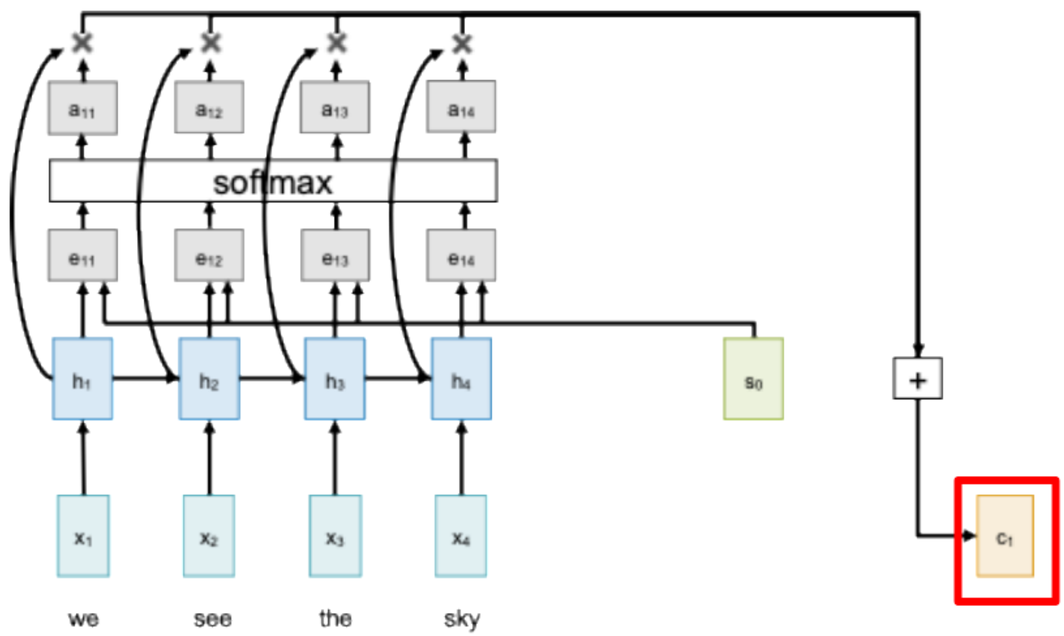


Computation:
Similarities: e $[N_X]$ $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
Attention weights: a = softmax(e) $[N_X]$

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: $\mathbf{q}$ $[D_Q]$
Data vectors: $\mathbf{X}$ $[N_X \times D_X]$



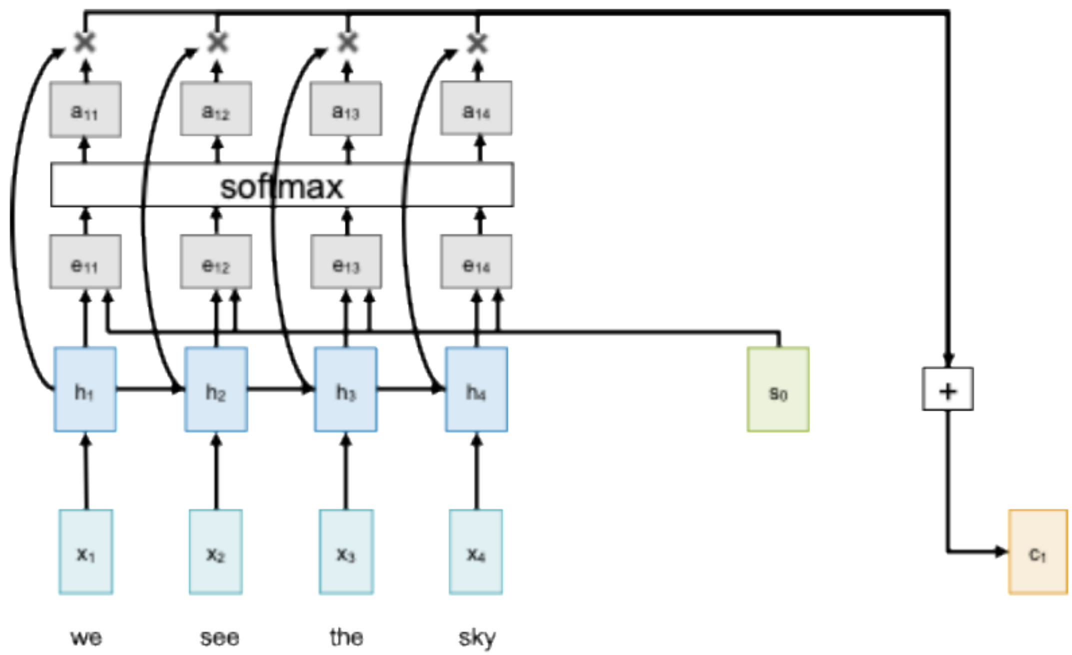
Computation:
Similarities: e $[N_X]$ $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
Attention weights: $a = \text{softmax}(e)$ $[N_X]$
Output vector: $\mathbf{y} = \sum_i a_i \mathbf{X}_i$ $[D_X]$

Let's generalize this!

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: **q** $[D_X]$
Data vectors: **X** $[N_X \times D_X]$



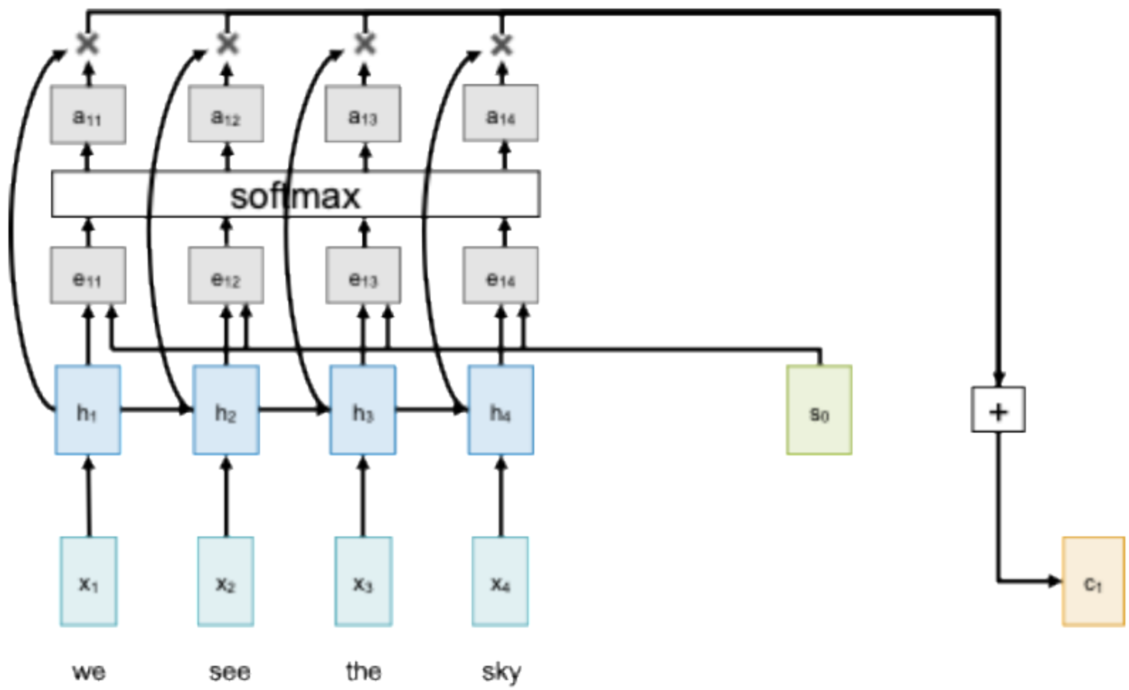
Computation:
Similarities: e $[N_X]$ $\boxed{e_i = q \cdot X_i}$
Attention weights: a = softmax(e) $[N_X]$
Output vector: $y = \sum_i a_i X_i$ $[D_X]$

Changes
- Use dot product for similarity

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: q [D_X]
Data vectors: X [N_X x D_X]



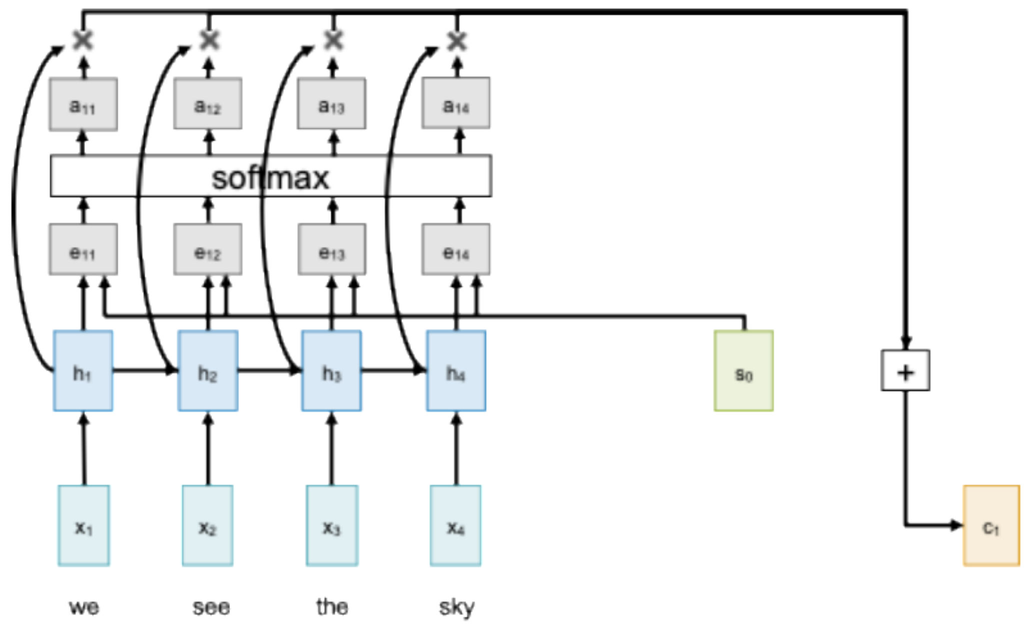
Computation:
Similarities: e [N_X] $\boxed{e_i = q \cdot X_i / \sqrt{D_X}}$
Attention weights: a = softmax(e) [N_X]
Output vector: $y = \sum_i a_i X_i$ [D_X]

Changes
- Use **scaled** dot product for similarity

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: $\boxed{\mathbf{Q}\ [N_Q \times D_X]}$
Data vectors: $\mathbf{X}$ $[N_X \times D_X]$

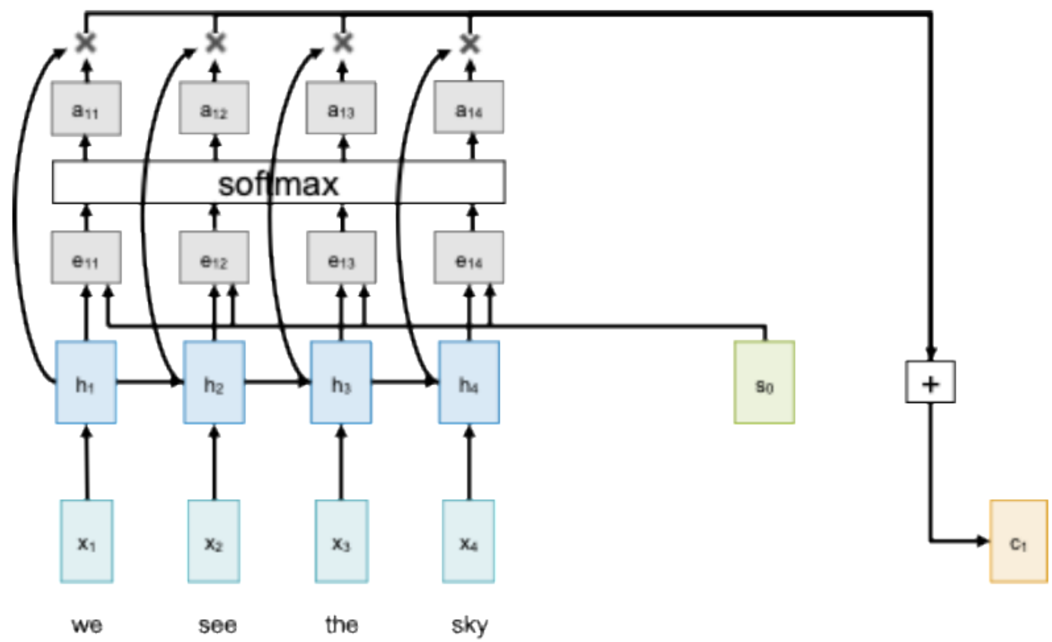


Computation:
Similarities: $E = \mathbf{QX}^T / \sqrt{D_X}$ $[N_Q \times N_X]$
$E_{ij} = \mathbf{Q_i} \cdot \mathbf{X_j} / \sqrt{D_X}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $\mathbf{Y} = A\mathbf{X}$ $[N_Q \times D_X]$
$\mathbf{Y_i} = \sum_j A_{ij} \mathbf{X_j}$

Changes
- Use scaled dot product for similarity
- Multiple **query** vectors

https://cs231n.stanford.edu/

Attention Example



Inputs:
Query vector: $\mathbf{Q}$ $[N_Q \times D_Q]$
Data vectors: $\mathbf{X}$ $[N_X \times D_X]$
Key matrix: $\mathbf{W_K}$ $[D_X \times D_Q]$
Value matrix: $\mathbf{W_V}$ $[D_X \times D_V]$

Computation:
Keys: $\mathbf{K} = \mathbf{XW_K}$ $[N_X \times D_Q]$
Values: $\mathbf{V} = \mathbf{XW_V}$ $[N_X \times D_V]$
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = \mathbf{Q_i} \cdot \mathbf{K_j} / \sqrt{D_Q}$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ $[N_Q \times N_X]$
Output vector: $\mathbf{Y} = A\mathbf{V}$ $[N_Q \times D_V]$
$\mathbf{Y_i} = \sum_j A_{ij} \mathbf{V_j}$

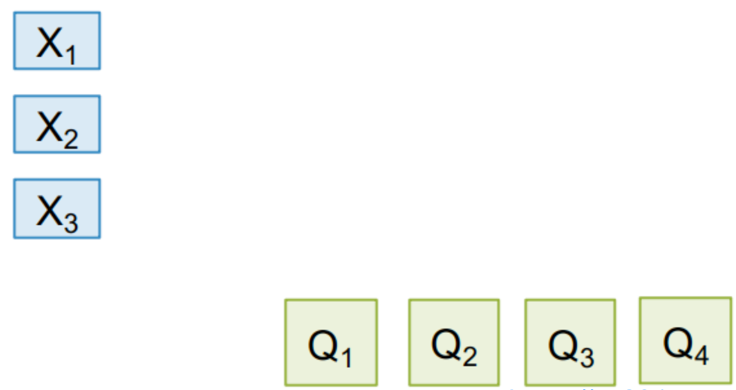
Changes
- Use scaled dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**

https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: Q $[N_Q \times D_Q]$
Data vectors: X $[N_X \times D_X]$
Key matrix: W_K $[D_X \times D_Q]$
Value matrix: W_V $[D_X \times D_V]$

Computation:
Keys: $K = XW_K$ $[N_X \times D_Q]$
Values: $V = XW_V$ $[N_X \times D_V]$
Similarities: $E = QK^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$

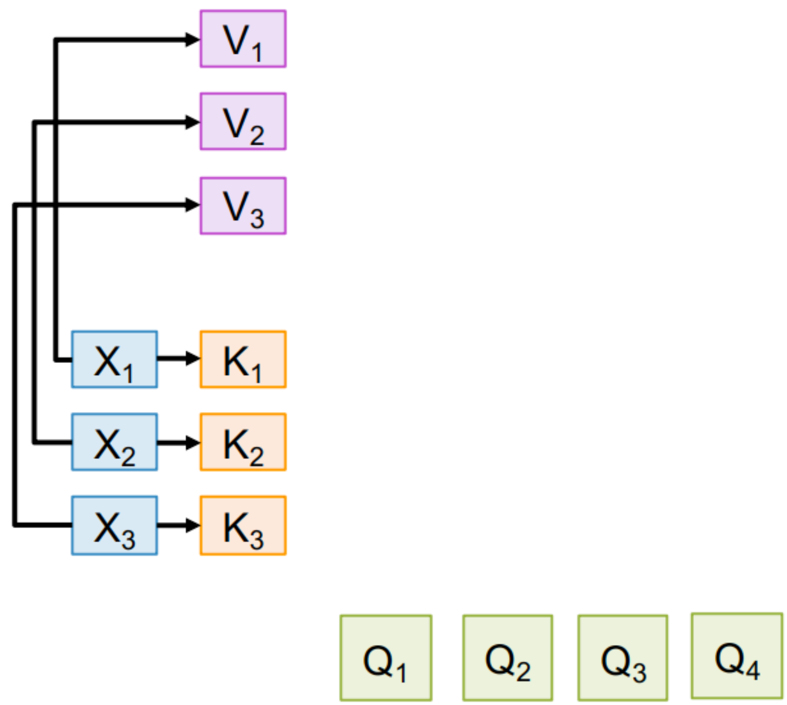


https://cs231n.stanford.edu/

Attention Example

<u>Inputs</u>:
Query vector: **Q** $[N_Q \times D_Q]$
Data vectors: **X** $[N_X \times D_X]$
Key matrix: $\mathbf{W_K}$ $[D_X \times D_Q]$
Value matrix: $\mathbf{W_V}$ $[D_X \times D_V]$

<u>Computation</u>:
Keys: $K = XW_K$ $[N_X \times D_Q]$
Values: $V = XW_V$ $[N_X \times D_V]$
Similarities: $E = QK^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$

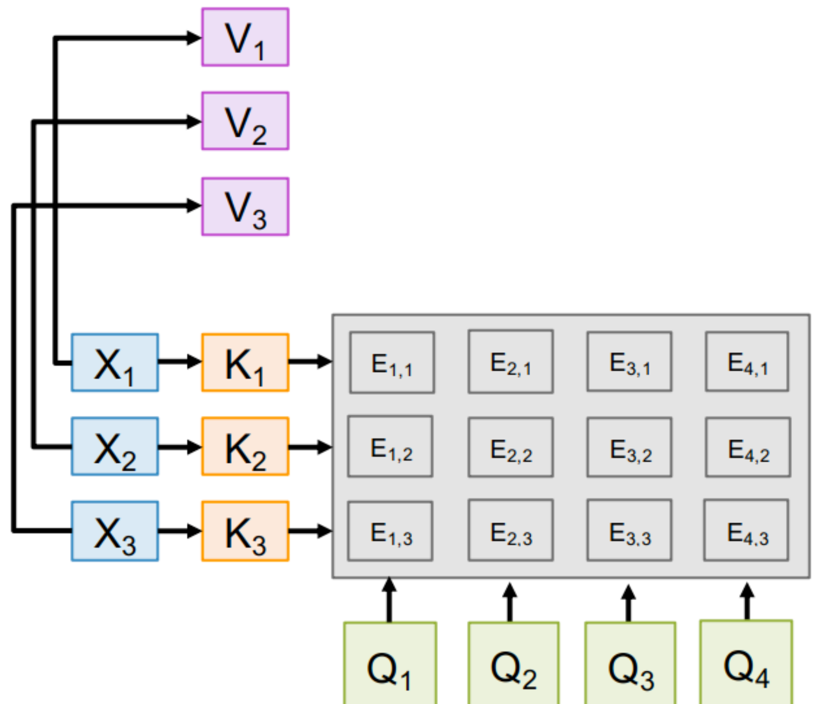


https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: **Q** $[N_Q \times D_Q]$
Data vectors: **X** $[N_X \times D_X]$
Key matrix: $\mathbf{W_K}$ $[D_X \times D_Q]$
Value matrix: $\mathbf{W_V}$ $[D_X \times D_V]$

Computation:
Keys: **K** = $\mathbf{XW_K}$ $[N_X \times D_Q]$
Values: **V** = $\mathbf{XW_V}$ $[N_X \times D_V]$
Similarities: E = $\mathbf{QK^T} / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = \mathbf{Q_i} \cdot \mathbf{K_j} / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) $[N_Q \times N_X]$
Output vector: **Y** = A**V** $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$



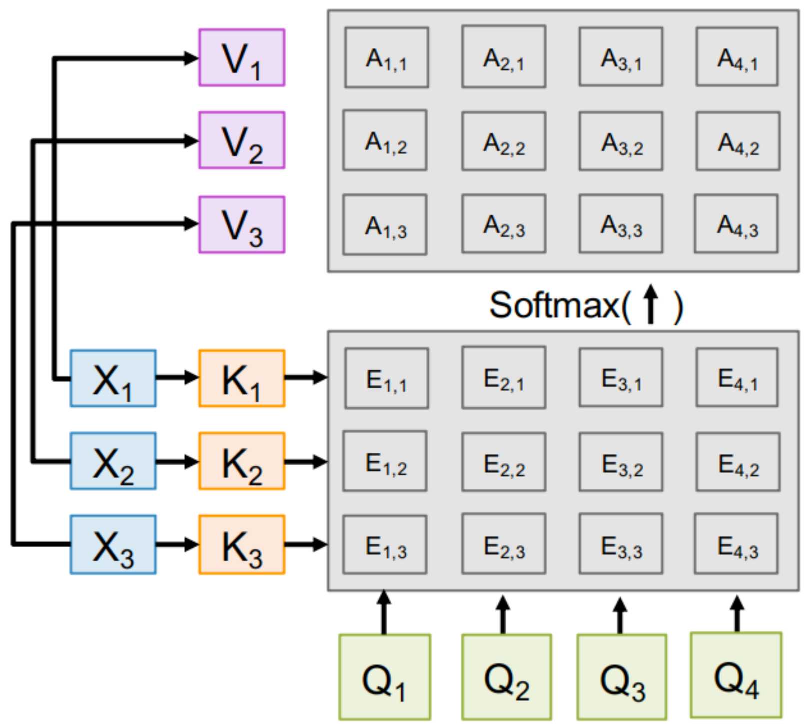
https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: **Q** $[N_Q \times D_Q]$
Data vectors: **X** $[N_X \times D_X]$
Key matrix: $\mathbf{W_K}$ $[D_X \times D_Q]$
Value matrix: $\mathbf{W_V}$ $[D_X \times D_V]$

Computation:
Keys: $\mathbf{K} = \mathbf{XW_K}$ $[N_X \times D_Q]$
Values: $\mathbf{V} = \mathbf{XW_V}$ $[N_X \times D_V]$
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = \mathbf{Q_i} \cdot \mathbf{K_j} / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$

Softmax normalizes each column: each **query** predicts a distribution over the **keys**



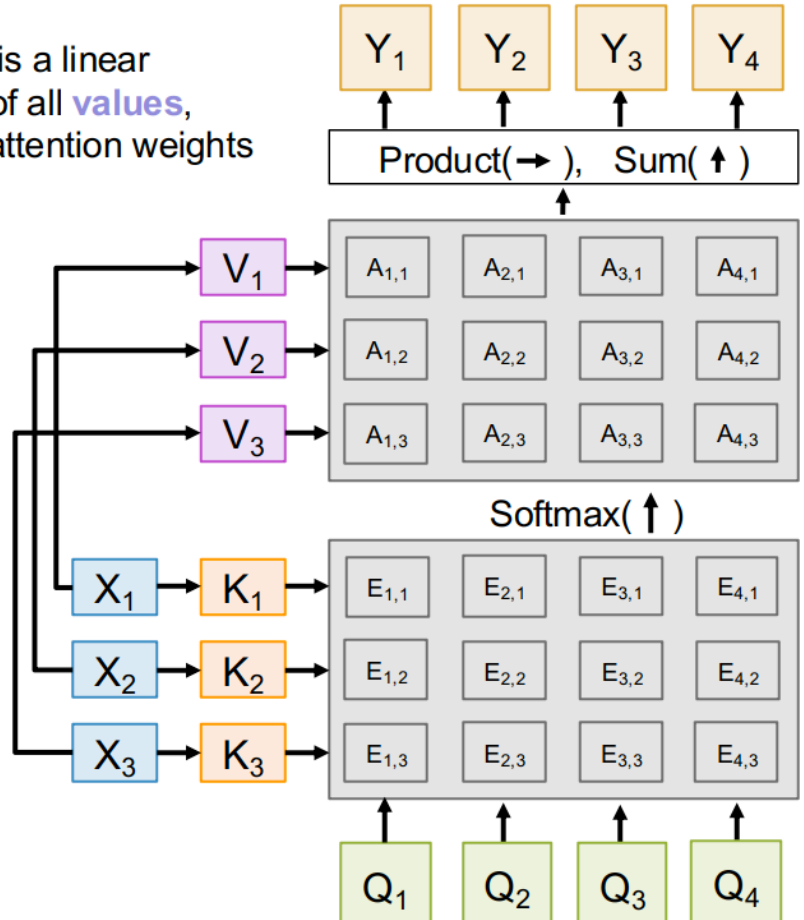
https://cs231n.stanford.edu/

Attention Example

Inputs:
Query vector: Q $[N_Q \times D_Q]$
Data vectors: X $[N_X \times D_X]$
Key matrix: W_K $[D_X \times D_Q]$
Value matrix: W_V $[D_X \times D_V]$

Computation:
Keys: $K = XW_K$ $[N_X \times D_Q]$
Values: $V = XW_V$ $[N_X \times D_V]$
Similarities: $E = QK^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$

Each **output** is a linear combination of all **values**, weighted by attention weights



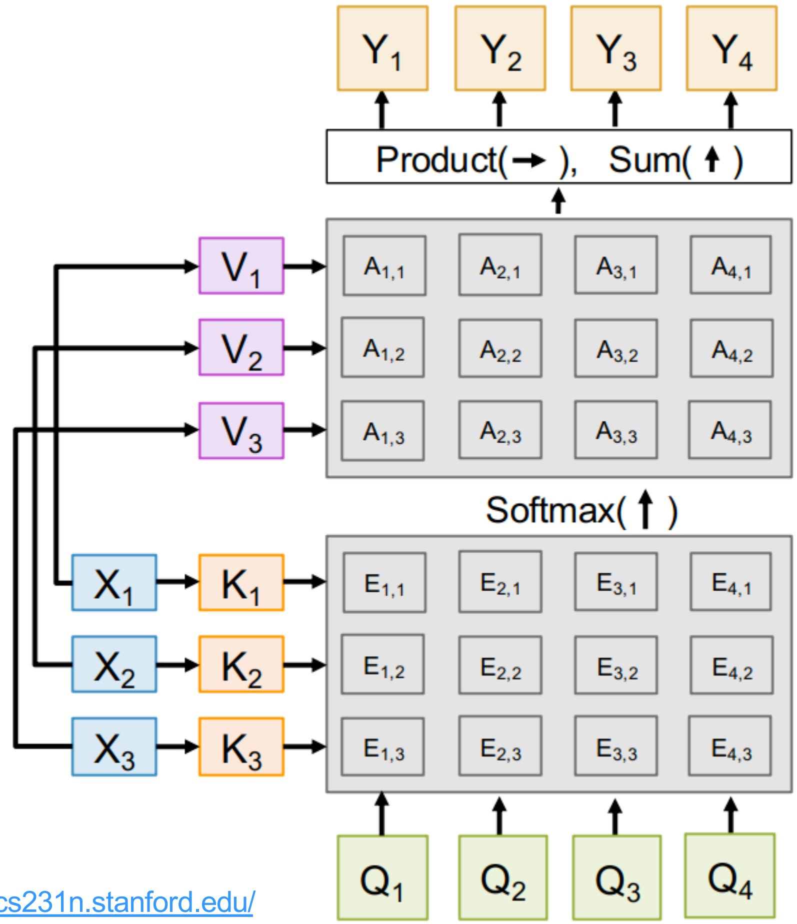
https://cs231n.stanford.edu/

Cross-Attention Example

Inputs:
Query vector: Q $[N_Q \times D_Q]$
Data vectors: X $[N_X \times D_X]$
Key matrix: W_K $[D_X \times D_Q]$
Value matrix: W_V $[D_X \times D_V]$

Each **query** produces one **output**, which is a mix of information in the **data** vectors

Computation:
Keys: $K = XW_K$ $[N_X \times D_Q]$
Values: $V = XW_V$ $[N_X \times D_V]$
Similarities: $E = QK^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$



https://cs231n.stanford.edu/

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 seq2seq + attention model, each decoder hidden state (query) *attends to* all encoder hidden states (values).

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.

Note: The animations below are videos. Touch or hover on them (if you're using a mouse) to get play controls so you can pause if needed.

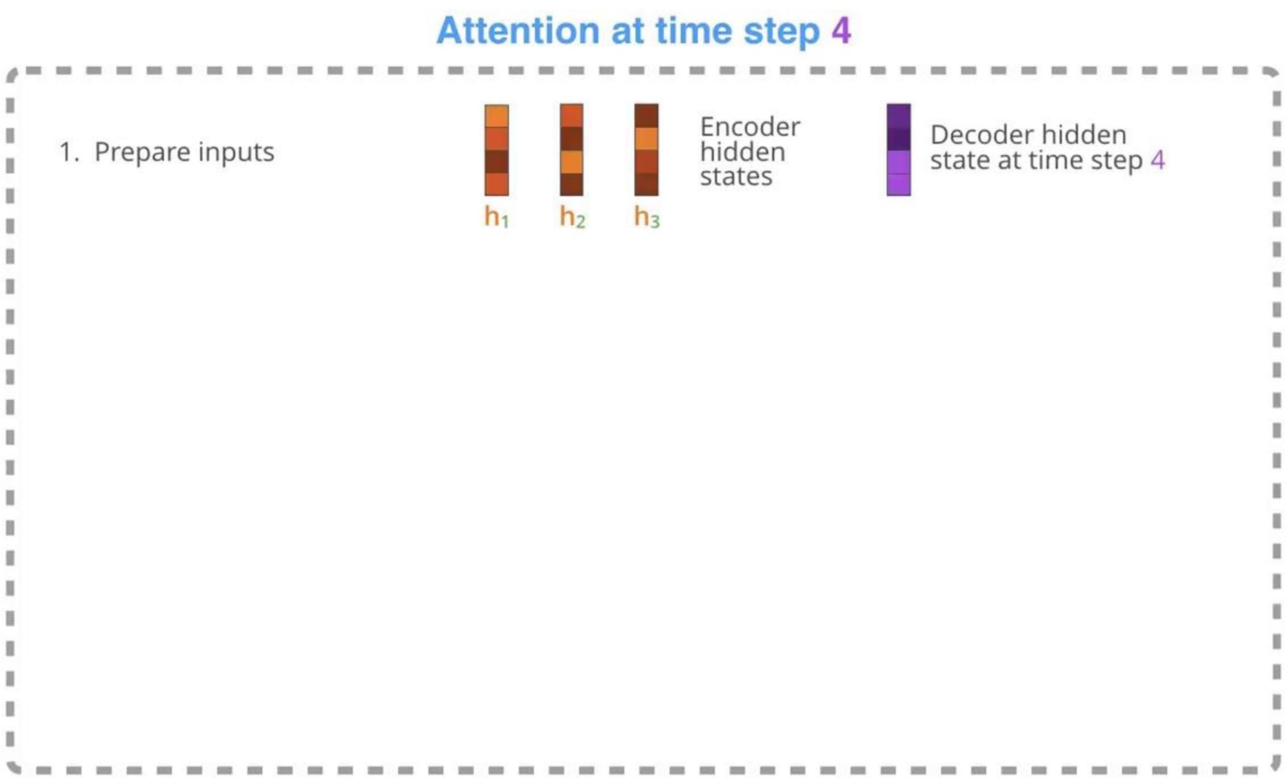
Sequence-to-sequence models are deep learning models that have achieved a lot of success in tasks like machine translation, text summarization, and image captioning. Google Translate started using such a model in production in late 2016. These models are explained in the two pioneering papers (Sutskever et al., 2014, Cho et al., 2014).

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Slide Credit: Prof. Sandra Avila - UNICAMP

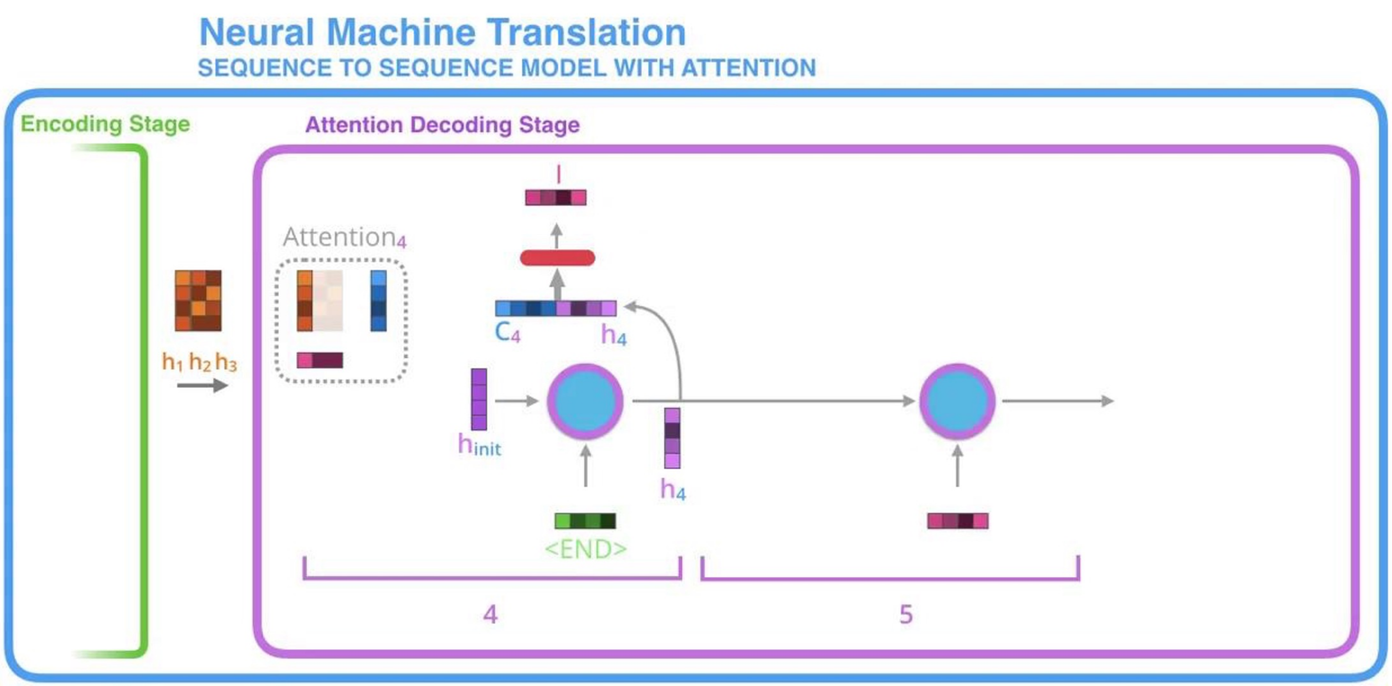
Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Slide Credit: Prof. Sandra Avila - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

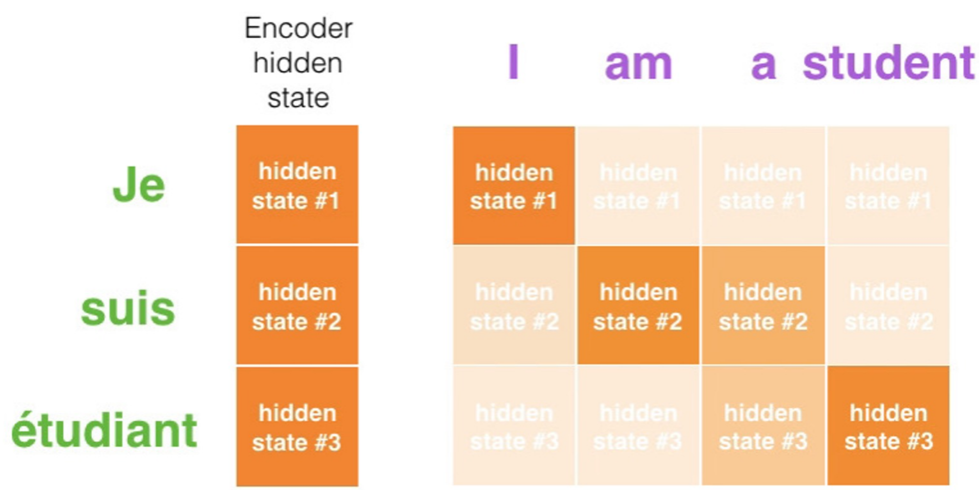
Slide Credit: [Prof. Sandra Avila](#) - UNICAMP

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Translations: Chinese (Simplified), Japanese, Korean, Russian

Watch: MIT's Deep Learning State of the Art lecture referencing this post

May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example.

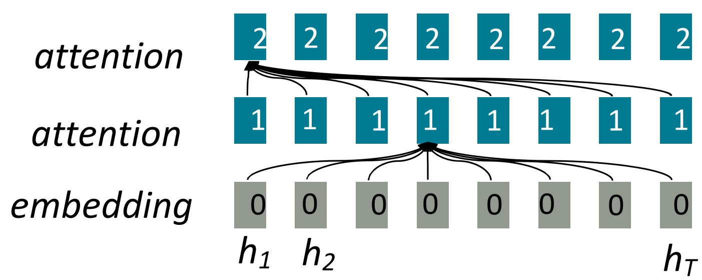


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Slide Credit: Prof. Sandra Avila - UNICAMP

If not recurrence, then what? How about attention?

- **Attention** treats each word's representation as a query to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; next we'll think about attention **within a single sentence**.
    - If **attention** gives us access to any state… maybe we can just use attention and don't need the RNN?
- Number of unparallelizable operations not tied to sequence length.
- All words interact at every layer!



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

Adapted from John Hewitt

Self-Attention

- Attention operates on **queries**, **keys**, and **values.**
  - We have some **queries** $q_1, q_2, \dots, q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, \dots, k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, \dots, v_T$. Each value is $v_i \in \mathbb{R}^d$
- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, \dots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:

The number of queries can differ from the number of keys and values in practice.

$$e_{ij} = q_i^\top k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad \text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute **key-query** affinities	Compute **attention weights** from affinities (softmax)	Compute **outputs** as weighted sum of **values**

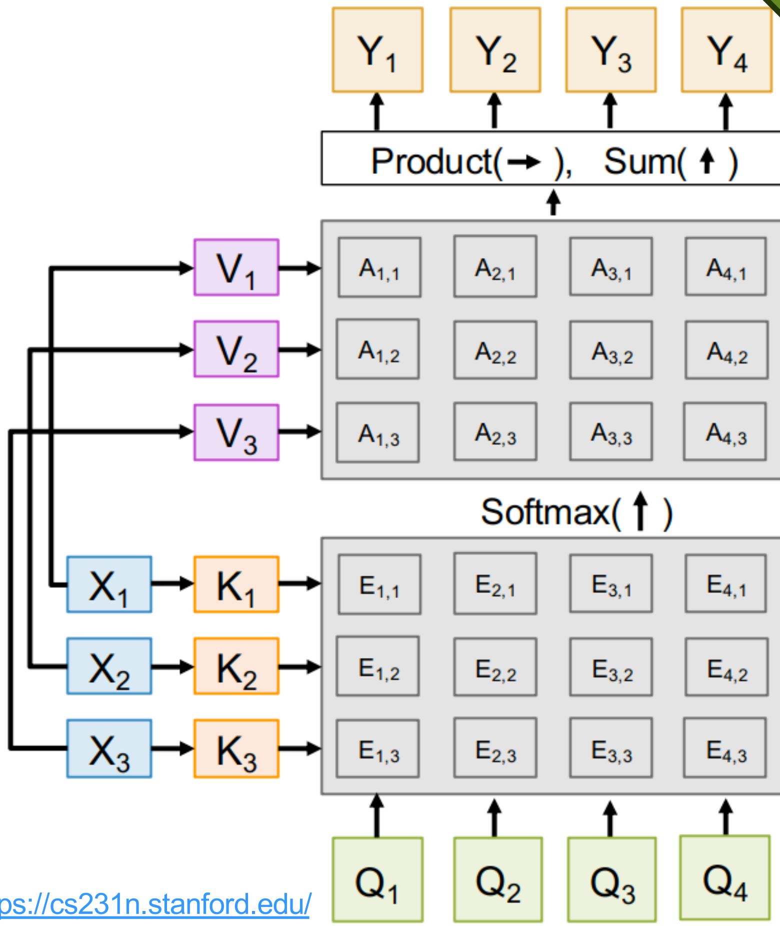
Cross-Attention Example

Recall

Inputs:
Query vector: Q $[N_Q \times D_Q]$
Data vectors: X $[N_X \times D_X]$
Key matrix: W_K $[D_X \times D_Q]$
Value matrix: W_V $[D_X \times D_V]$

Each **query** produces one **output**, which is a mix of information in the **data** vectors

Computation:
Keys: $K = XW_K$ $[N_X \times D_Q]$
Values: $V = XW_V$ $[N_X \times D_V]$
Similarities: $E = QK^T / \sqrt{D_Q}$ $[N_Q \times N_X]$
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$
Output vector: $Y = AV$ $[N_Q \times D_V]$
$Y_i = \sum_j A_{ij} V_j$



https://cs231n.stanford.edu/

Self-Attention Example

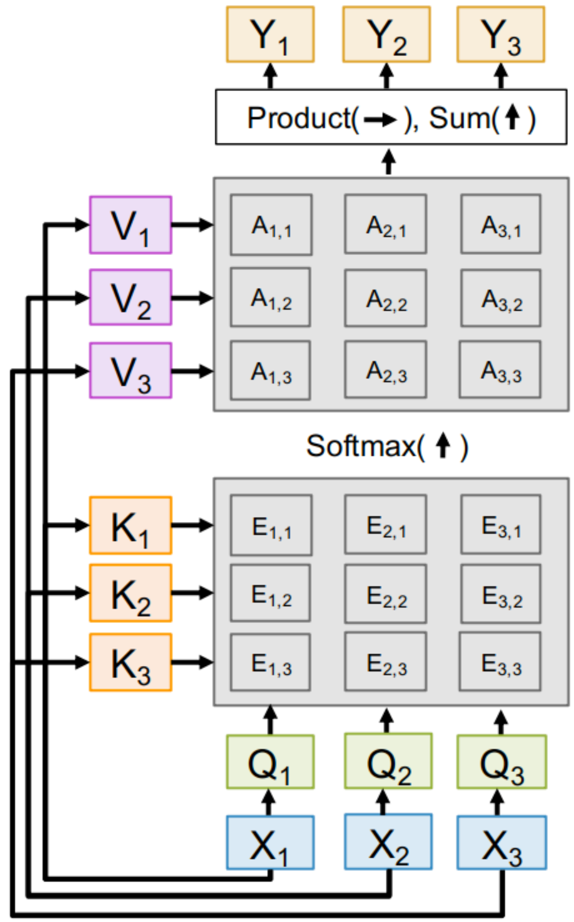
<u>**Inputs**</u>:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

<u>**Computation**</u>:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) [N x N]
Output vector: $Y = AV$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Each **input** produces one **output**, which is a mix of information from all **inputs**

Shapes get a little simpler:
- N input vectors, each D_{in}
- Almost always $D_Q = D_V = D_{out}$

https://cs231n.stanford.edu/

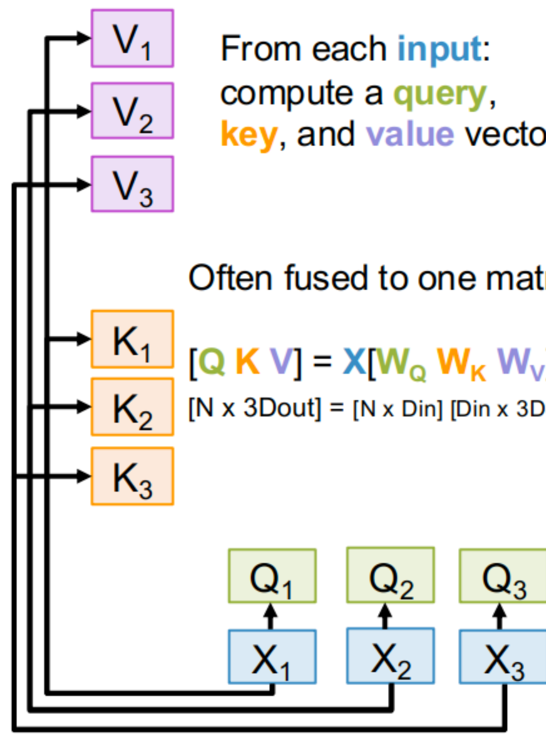


Self-Attention Example

Inputs:
Input vectors: X [$N \times D_{in}$]
Key matrix: W_K [$D_{in} \times D_{out}$]
Value matrix: W_V [$D_{in} \times D_{out}$]
Query matrix: W_Q [$D_{in} \times D_{out}$]

Each **input** produces one **output**, which is a mix of information from all **inputs**

Computation:
Queries: $Q = XW_Q$ [$N \times D_{out}$]
Keys: $K = XW_K$ [$N \times D_{out}$]
Values: $V = XW_V$ [$N \times D_{out}$]
Similarities: $E = QK^T / \sqrt{D_Q}$ [$N \times N$]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [$N \times N$]
Output vector: $Y = AV$ [$N \times D_{out}$]
$Y_i = \sum_j A_{ij} V_j$



From each **input**: compute a **query**, **key**, and **value** vector

Often fused to one matmul:

$[Q\ K\ V] = X[W_Q\ W_K\ W_V]$

[N x 3Dout] = [N x Din] [Din x 3Dout]

https://cs231n.stanford.edu/

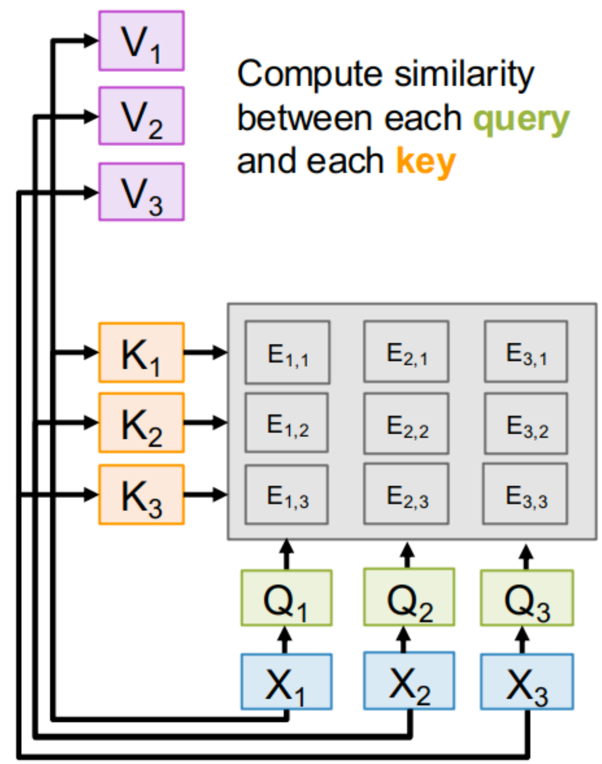
Self-Attention Example

Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Each **input** produces one **output**, which is a mix of information from all **inputs**

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) [N x N]
Output vector: $Y = AV$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

https://cs231n.stanford.edu/



Compute similarity between each **query** and each **key**

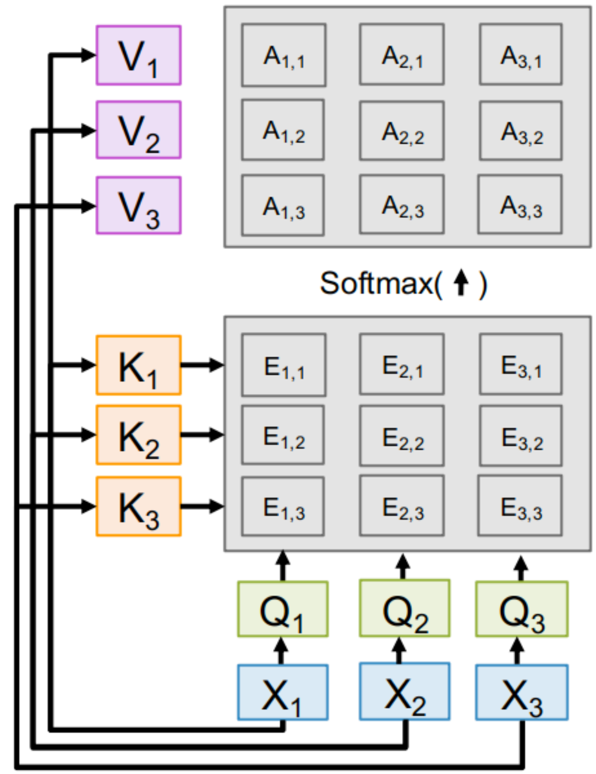
Self-Attention Example

Inputs:
Input vectors: **X** [N x D_{in}]
Key matrix: **W_K** [D_{in} x D_{out}]
Value matrix: **W_V** [D_{in} x D_{out}]
Query matrix: **W_Q** [D_{in} x D_{out}]

Each **input** produces one **output**, which is a mix of information from all **inputs**

Computation:
Queries: **Q** = **XW_Q** [N x D_{out}]
Keys: **K** = **XW_K** [N x D_{out}]
Values: **V** = **XW_V** [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) [N x N]
Output vector: **Y** = **AV** [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Normalize over each column: each **query** computes a distribution over **keys**



https://cs231n.stanford.edu/

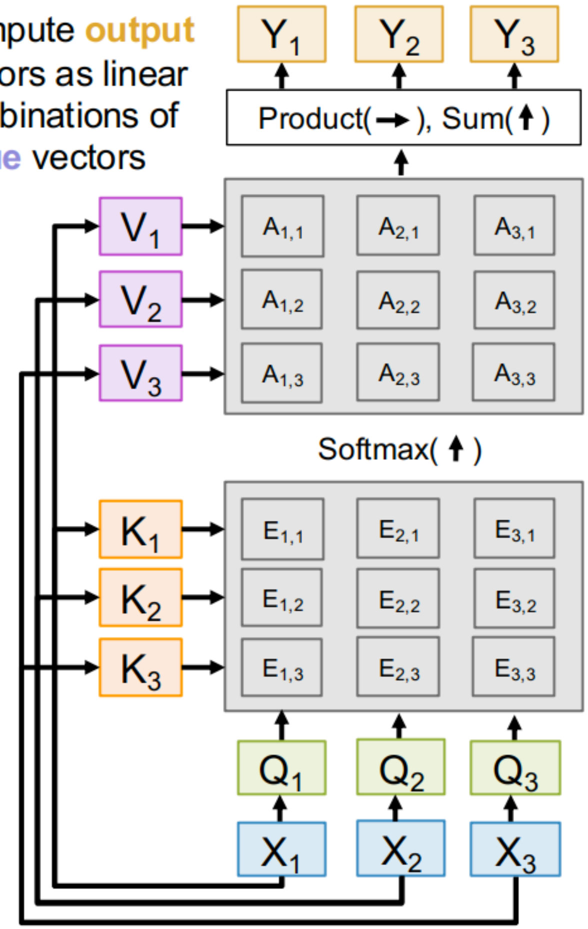
Self-Attention Example

Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Each input produces one output, which is a mix of information from all inputs

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]
Output vector: $Y = AV$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Compute output vectors as linear combinations of value vectors



https://cs231n.stanford.edu/

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!

Solutions

Fixing the first self-attention problem: Sequence order

- The transformer model is purely attentional.
- If embeddings were used, there would be no way to distinguish between identical words.

A big dog and a big cat

- Positional encodings add an embedding based on the word position.

$W_{big} + W_{pos2}$ $W_{big} + W_{pos6}$

https://phontron.com/class/anlp-fall2024/

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 v_i', k_i', q_i' be our old values, keys, and queries.

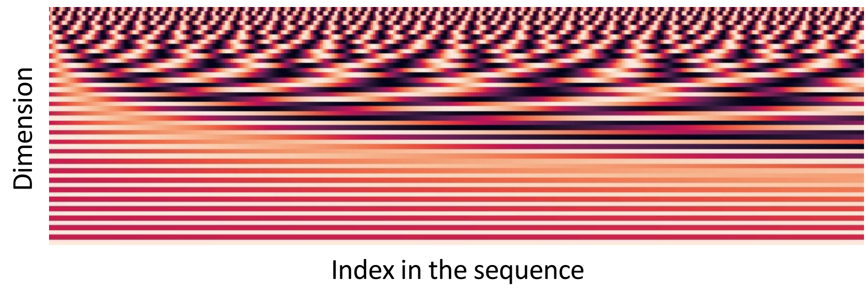
$$v_i = v_i' + p_i$$
$$q_i = q_i' + p_i$$
$$k_i = k_i' + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position representation vectors through sinusoids

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

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



- Periodicity indicates that maybe "absolute position" isn't as important

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Barriers and solutions for Self-Attention as a building block

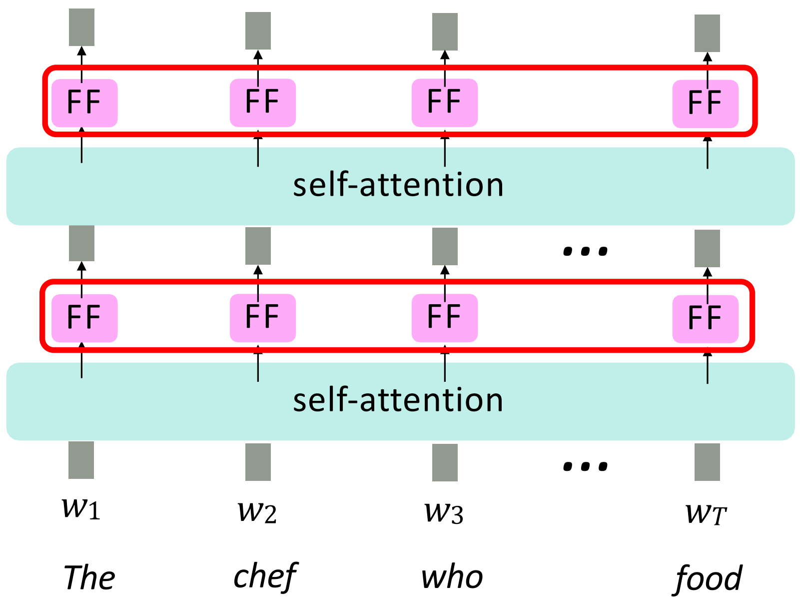
Barriers		Solutions
• Doesn’t have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning! It’s all just weighted averages	→	

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

- Easy fix: add a **feed-forward network** to post-process each output vector.

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



Intuition: the FF network processes the result of attention

Barriers and solutions for Self-Attention as a building block

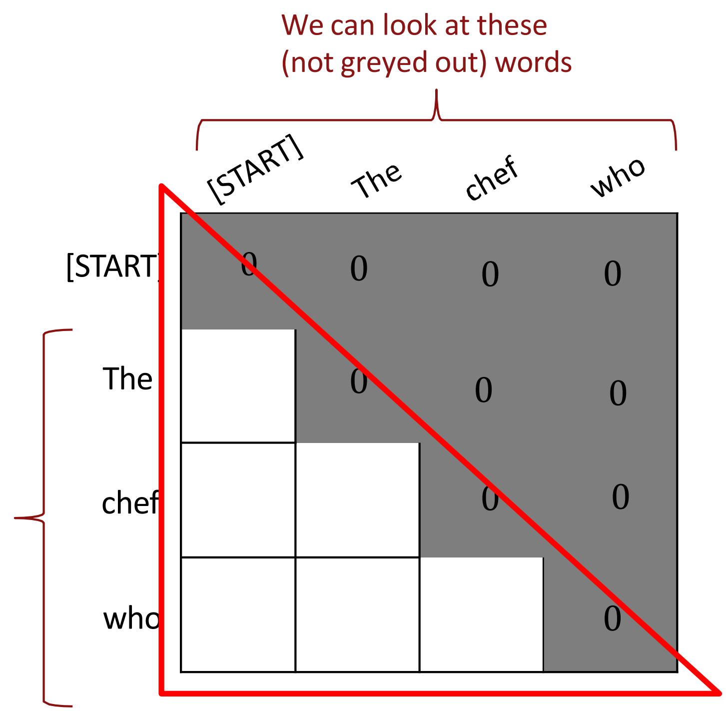
Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	→	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling
	→	

Masking the future in 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 0.

For encoding these words



$$e_{ij} = \begin{cases} q_i^\top k_j, & j < i \\ 0, & j \geq i \end{cases}$$

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning magic! It's all just weighted averages
- Need to ensure we don't "look at the future" when predicting a sequence
  - Like in machine translation
  - Or language modeling

Solutions

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self- attention output.
- Mask out the future by artificially setting attention weights to 0!

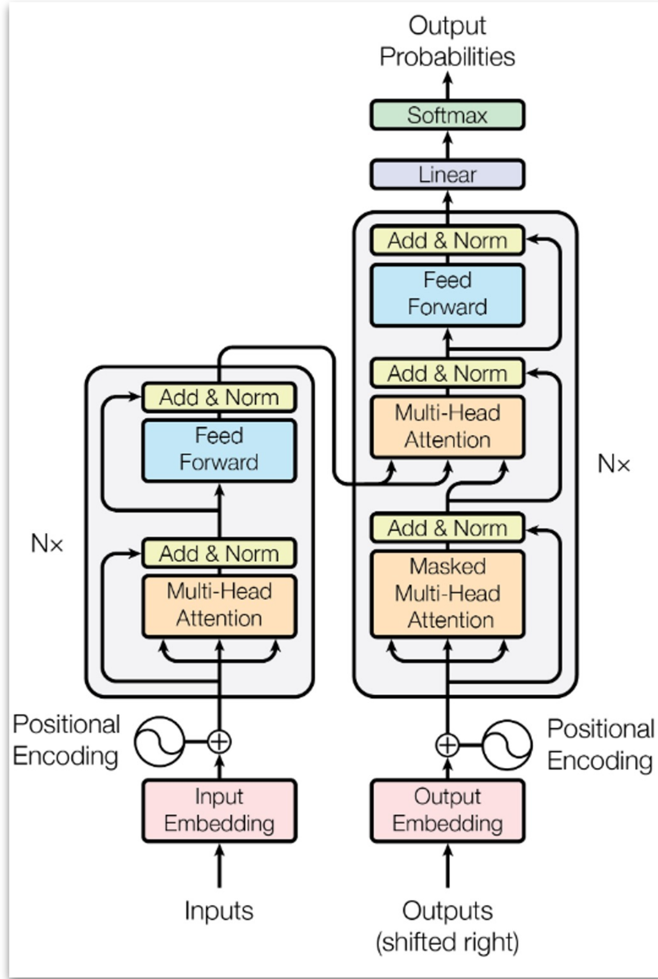
Necessities for a self-attention building block:

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.

- That's it! But this is not the **Transformer** model we've been hearing about.

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward



https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

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

The Illustrated Transformer

Discussions: Hacker News (65 points, 4 comments), Reddit r/MachineLearning (29 points, 3 comments)

Translations: Chinese (Simplified), French, Japanese, Korean, Russian, Spanish

Watch: MIT's Deep Learning State of the Art lecture referencing this post

In the previous post, we looked at Attention – a ubiquitous method in modern deep learning models. Attention is a concept that helped improve the performance of neural machine translation applications. In this post, we will look at **The Transformer** – a model that uses attention to boost the speed with which these models can be trained. The Transformers outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit, however, comes from how The Transformer lends itself to parallelization. It is in fact Google Cloud's recommendation to use The Transformer as a reference model to use their Cloud TPU offering. So let's try to break the model apart and look at how it functions.

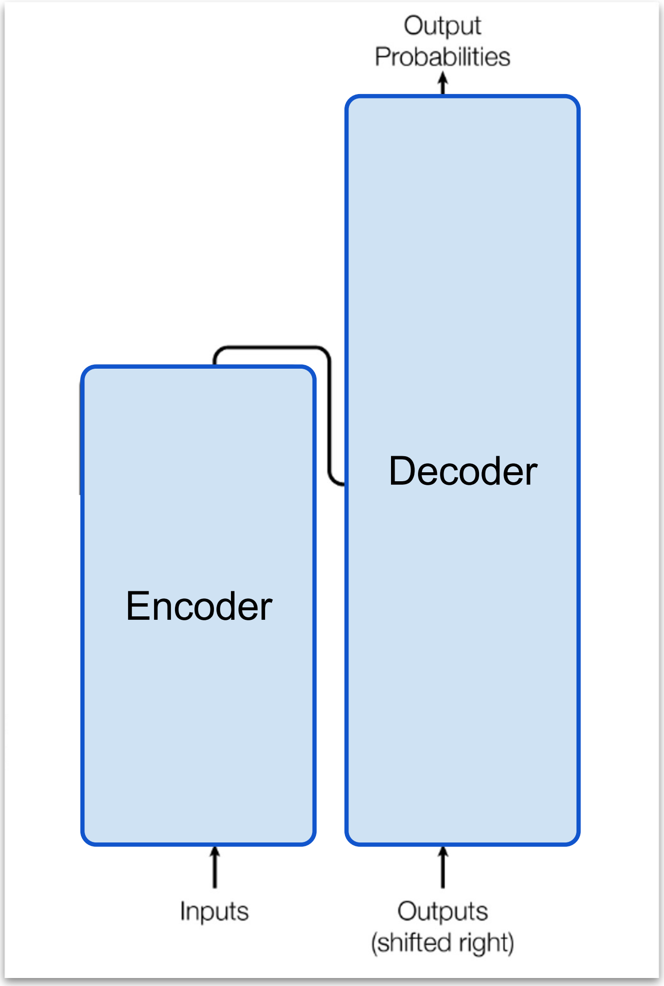
The Transformer was proposed in the paper Attention is All You Need. A TensorFlow implementation of it is available as a part of the Tensor2Tensor package. Harvard's NLP group created a guide annotating the paper with PyTorch implementation. In this post, we will attempt to oversimplify things a bit and introduce the concepts one by one to hopefully make it easier to understand to people without in-depth knowledge of the subject matter.

2020 Update: I've created a "Narrated Transformer" video which is a gentler approach to the topic:

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward



https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

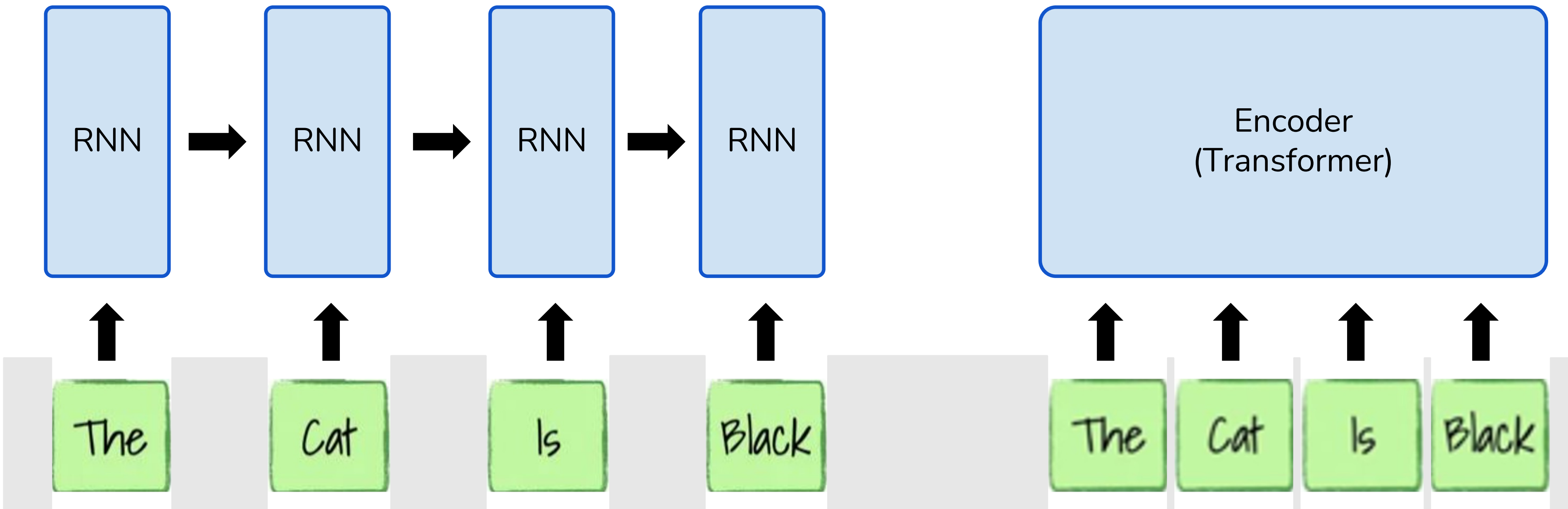
Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer: Encoder & Decoder

- The model is primarily composed of two blocks:
  - The **encoder** receives an input and builds a representation of it (its features). This means that the model is optimized to acquire understanding from the input.
  - The **decoder** uses the encoder's representation (features) along with other inputs to generate a target sequence. This means that the model is optimized for generating outputs.

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformers vs. RNNs



Slide Credit: Prof. Sandra Avila - UNICAMP

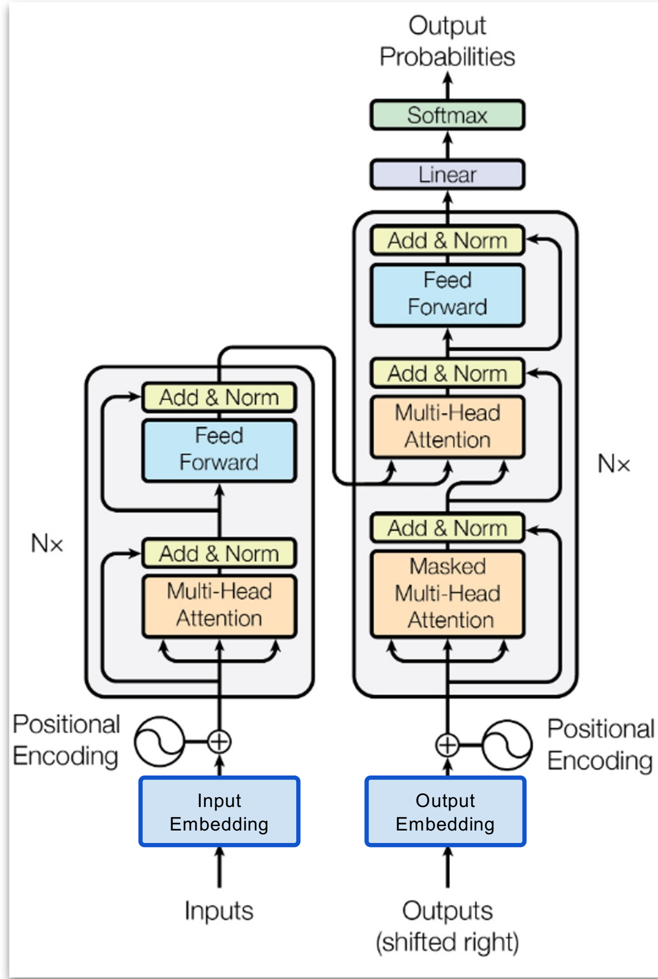
Transformer: Encoder & Decoder

- **Encoder-only models**: ALBERT, BERT, DistilBERT, ELECTRA, RoBERTa
- **Decoder-only models**: CTRL, GPT, GPT-2, GPT-3, GPT-4, Transformer XL.
- **Encoder-decoder models or sequence-to-sequence models**: BART, mBART, Marian, T5.

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer

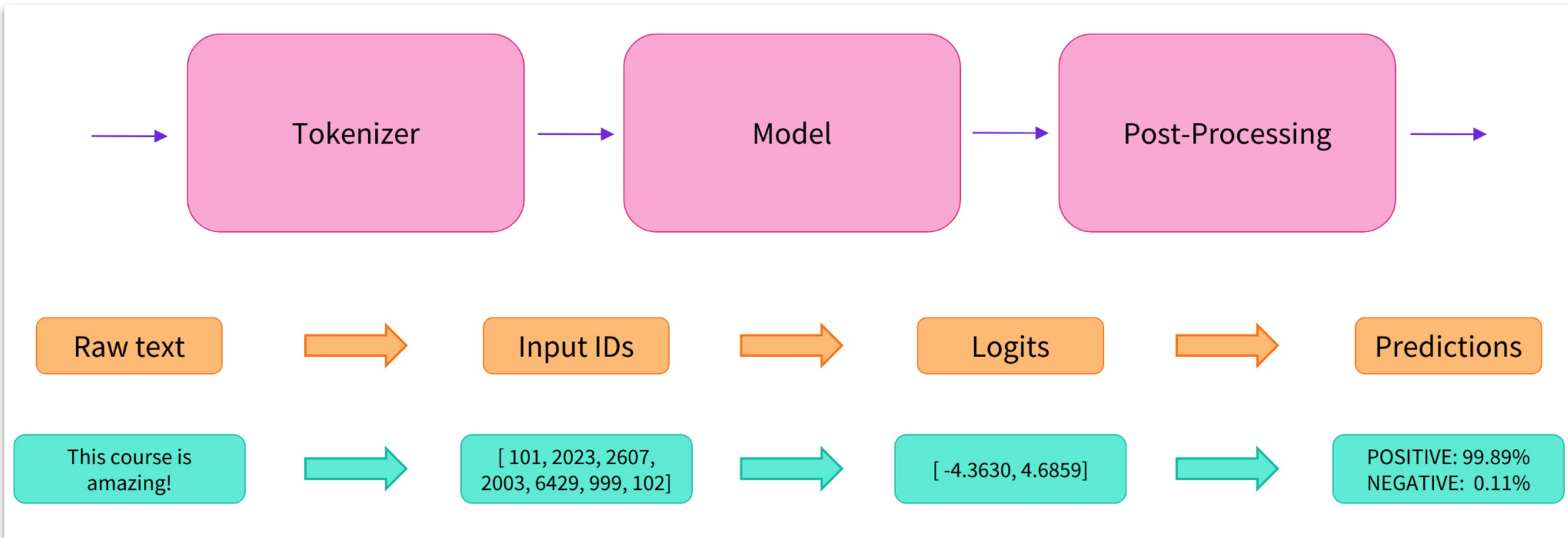
- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward



https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

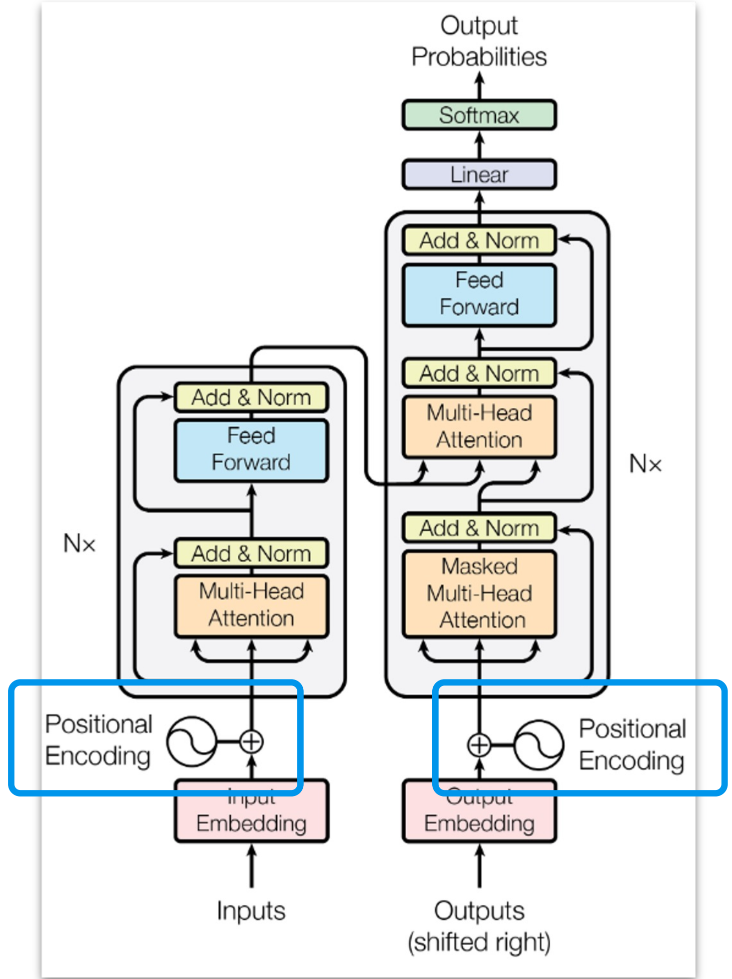
Transformer: Input & output embedding



Slide Credit: [Prof. Sandra Avila]- UNICAMP

Transformer

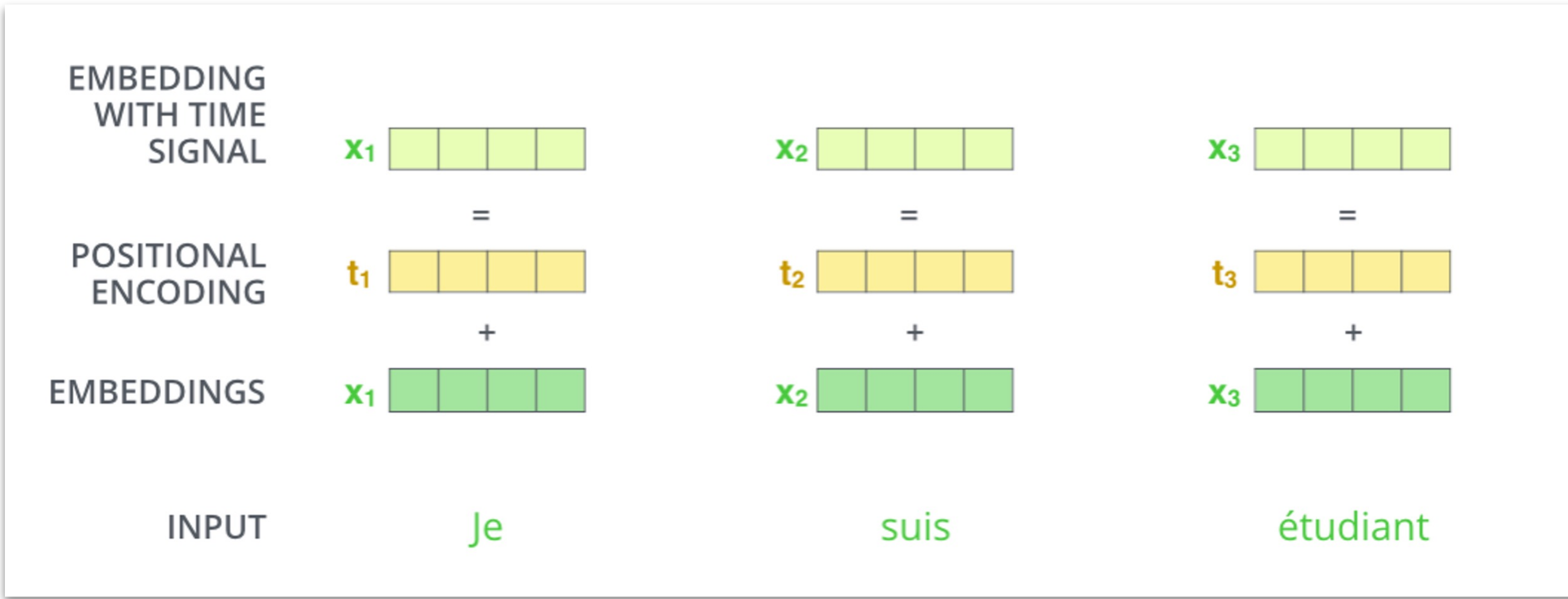
- Transformer Architecture
    - Encoder & Decoder
    - Input & output embedding
    - Positional encoding
    - Self-attention
    - Multi-head attention
    - Masked multi-head attention
    - Residual connections
    - Layer Normalization
    - Feedforward



https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer: Positional Encoding

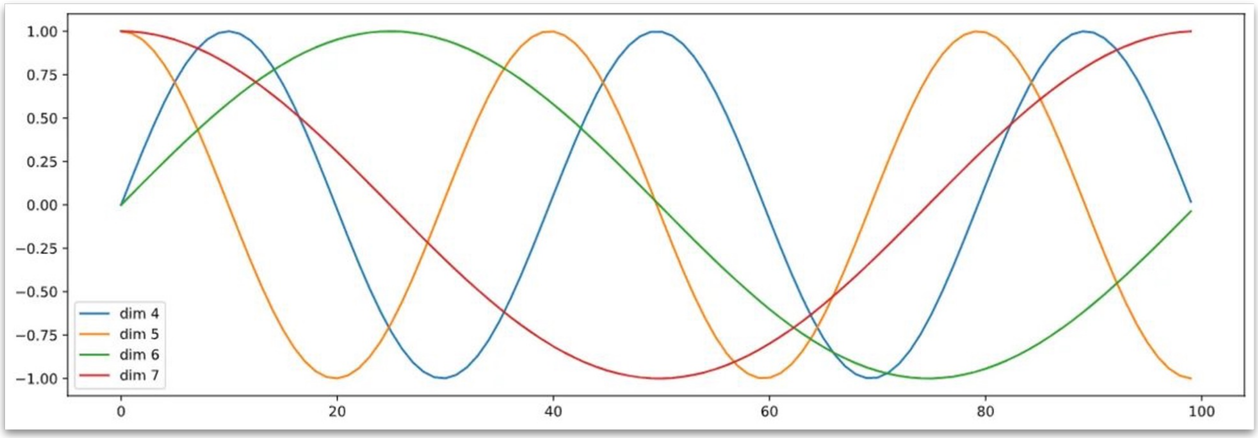


https://huggingface.co/transformers/tokenizer_summary.html

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer: Positional Encoding

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



https://huggingface.co/transformers/tokenizer_summary.html

Slide Credit: Prof. Sandra Avila - UNICAMP

How can we improve Positional Encoding?

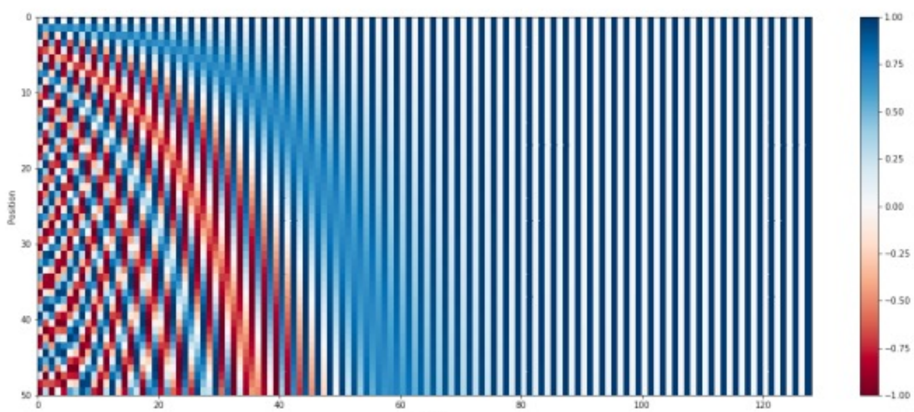
Transformer: Sinusoidal Encoding

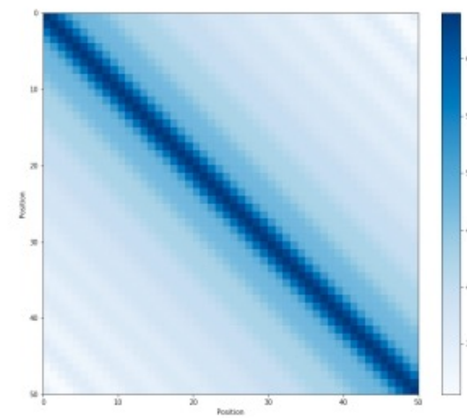
(Vaswani+ 2017, Kazemnejad 2019)

- Calculate each dimension with a sinusoidal function

Notable Models:
Orig. Transformer

$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$





- Why? So the dot product between two embeddings becomes higher relatively.

https://phontron.com/class/anlp-fall2024/

Transformer: Absolute vs Relative Encoding

(Shaw+ 2018)

Notable Models:
GPT 1, 2, 3 and OPT

- Absolute positional encodings add an encoding to the input in hope that relative position will be captured

Disadvantages
We can have fixed positional embeddings for each index training position (e.g., 1, 2, 3, … 1000) → What happens if we get a sequence with 5000 words at test time?

We want something that can generalize to arbitrary sequence lengths.

Notable Models:
T5, Gopher, Chinchilla

- Relative positional encodings explicitly encode relative position
  Each position is computed on its distance from the other positions it is attending to.

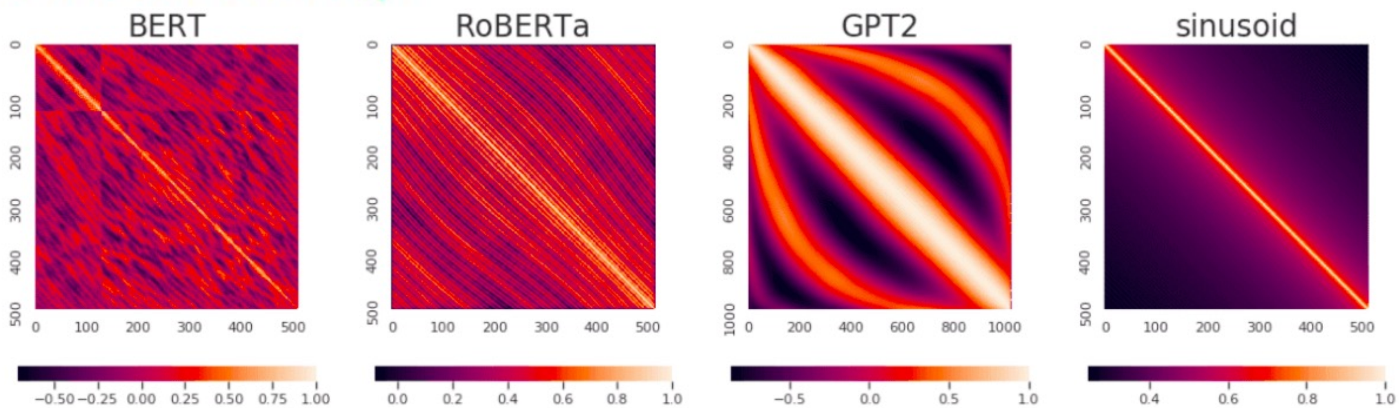
https://phontron.com/class/anlp-fall2024/
https://self-supervised.cs.jhu.edu/fa2024/

Transformer: Relative Positional Encoding

- You can rewrite the statement from the previous slide in the following form:

$$QK_{ij} = (W_q[\boldsymbol{x}_i + \boldsymbol{p}_i])^T (W_k[\boldsymbol{x}_j + \boldsymbol{p}_j]) = \boldsymbol{x}_i^T W_q{}^T W_k \boldsymbol{x}_j + P_{ij}$$

- Note, the values of P_{ij} encode the relative of $\boldsymbol{i}$ and $\boldsymbol{j}$.
- How should we construct P_{ij}?



https://self-supervised.cs.jhu.edu/fa2024/

Transformer: Relative Positional Encoding

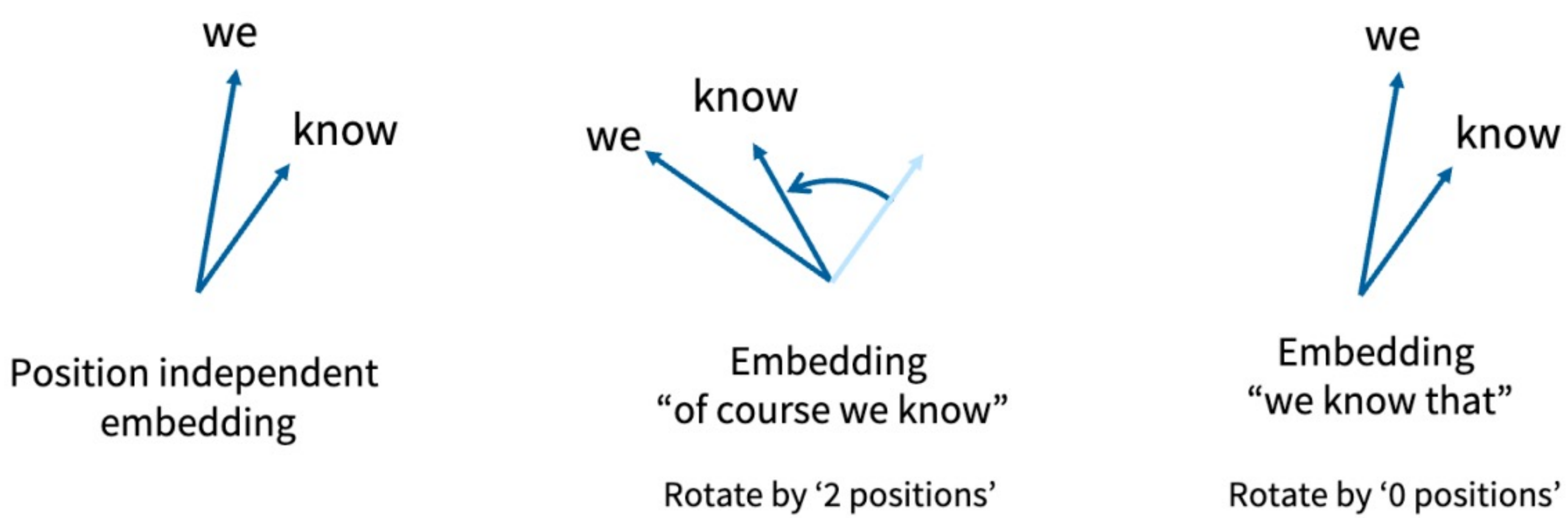
- There have been various choices:
  - T5 models simplify this into learnable relative embeddings $\boldsymbol{P_{ij}}$ such that:
    $$QK_{ij} = \boldsymbol{x_i^T} W_q{}^T W_k \boldsymbol{x_j} + \boldsymbol{P_{ij}}$$
  - DeBERTa learns relative positional embeddings $\widetilde{\boldsymbol{p}}_{i-j}$ such that:
    $$QK_{ij} = \boldsymbol{x_i^T} W_q{}^T W_k \boldsymbol{x_j} + \boldsymbol{x_i^T} W_q{}^T W_k \, \widetilde{\boldsymbol{p}}_{i-j} + \widetilde{\boldsymbol{p}}_{i-j}^T W_q{}^T W_k \boldsymbol{x_j}$$
  - Tranformer-XL learns relative positional embeddings $\widetilde{\boldsymbol{p}}_{i-j}$ and trainable vectors $\boldsymbol{u}$, $\boldsymbol{v}$ s.t.:
    $$QK_{ij} = \boldsymbol{x_i^T} W_q{}^T W_k \boldsymbol{x_j} + \boldsymbol{x_i^T} W_q{}^T W_k \, \widetilde{\boldsymbol{p}}_{i-j} + \boldsymbol{u}^T W_q{}^T W_k \boldsymbol{x_j} + \boldsymbol{v}^T W_q{}^T W_k \, \widetilde{\boldsymbol{p}}_{i-j}$$
  - ALiBi learns learns a scalar m such that:
    $$QK_{ij} = \boldsymbol{x_i^T} W_q{}^T W_k \boldsymbol{x_j} - m \, |i - j|$$

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, 2020

Transformer: Rotary Positional Encoding (RoPE)

Notable Models:
GPTJ, PaLM, LLaMA

- We want our embeddings to be invariant to absolute position.
- We know that inner products are invariant to arbitrary rotation.

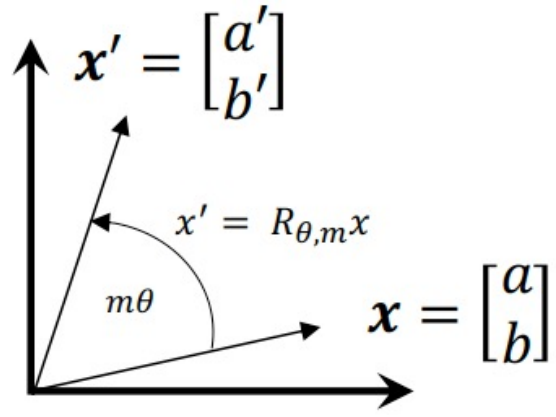


Transformer: Rotary Positional Encoding (RoPE)

- In 2D, a rotation matrix can be defined in the following form:

$$R_{\theta,m} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$

- The rotation increases with increasing θ and m.



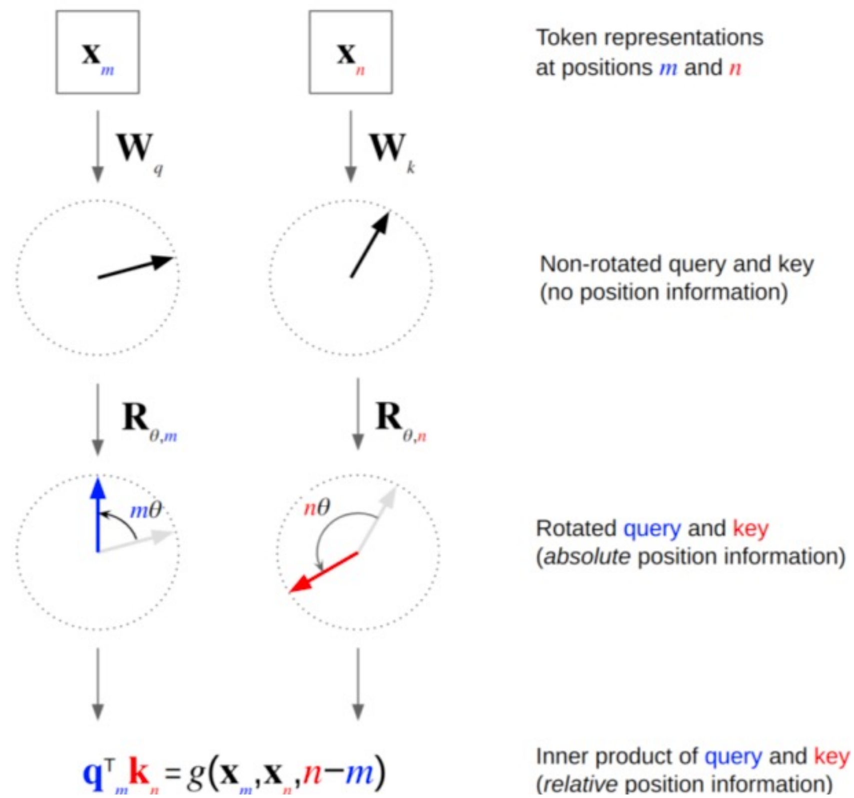
https://self-supervised.cs.jhu.edu/fa2024/

Transformer: Rotary Positional Encoding (RoPE)

- Drop the additive positional encoding and make it multiplicative.

$$qk_{mn} = \left(R_{\theta,m} W_q \boldsymbol{x}_m\right)^T \left(R_{\theta,n} W_k \boldsymbol{x}_n\right)$$
$$= \boldsymbol{x}_m^T W_q^T R_{\theta,m}^T R_{\theta,n} W_k \boldsymbol{x}_j$$

  - θ: the size of rotation
  - $R_{\theta,m}$: rotation matrix, rotates a vector it gets multiplied to proportional to θ and the position index m.
- Intuition: **nearby** words have **smaller relative rotation**.



RoFormer: Enhanced Transformer with Rotary Position Embedding (2022)

https://self-supervised.cs.jhu.edu/fa2024/

Transformer: Rotary Positional Encoding (RoPE)

- In practice, we are rotating d dimensional embedding matrices.
- Idea: rotate different dimensions with different angles:
  - $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \dots, \theta_{d/2}\}$

$$\mathbf{R}^d_{\Theta,t} = \begin{pmatrix} \cos t\theta_1 & -\sin t\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin t\theta_1 & \cos t\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos t\theta_2 & -\sin t\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin t\theta_2 & \cos t\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos t\theta_{d/2} & -\sin t\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin t\theta_{d/2} & \cos t\theta_{d/2} \end{pmatrix}$$

RoFormer: Enhanced Transformer with Rotary Position Embedding (2022)

https://self-supervised.cs.jhu.edu/fa2024/

Transformer: Rotary Positional Encoding (RoPE)

[Key idea]

- Break each d-dimensional input vector into d/2 vectors of length 2
- Rotate each of the d/2 vectors by an amount scaled by m
- m is the absolute position of the query or the key

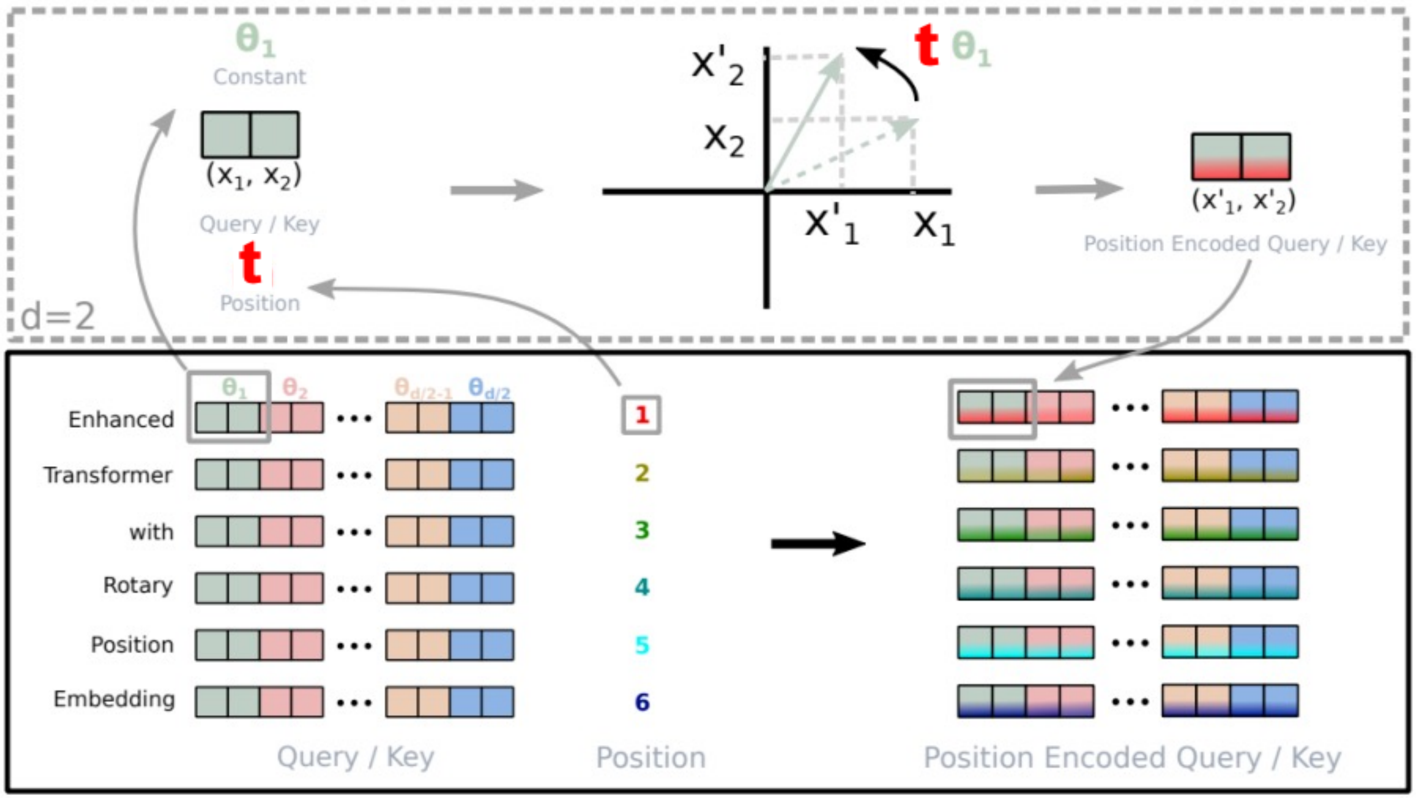


Figure from http://arxiv.org/abs/2104.09864

https://www.cs.cmu.edu/~mgormley/courses/10423/

Transformer: Rotary Positional Encoding (RoPE)

$$qk_{mn} = \left(R^d_{\Theta,m} W_q \boldsymbol{x}_m\right)^T \left(R^d_{\Theta,m} W_k \boldsymbol{x}_n\right),$$

- where $R^d_{\Theta,m}$ is a d-dimensional rotation matrix.
- Since $R^d_{\Theta,m}$ is a sparce matrix, its multiplication is implemented via dense operations:

$$\mathbf{R}^d_{\Theta,t}\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{d-1} \\ u_d \end{pmatrix} \otimes \begin{pmatrix} \cos t\theta_1 \\ \cos t\theta_1 \\ \cos t\theta_2 \\ \cos t\theta_2 \\ \vdots \\ \cos t\theta_{d/2} \\ \cos t\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -u_2 \\ u_1 \\ -u_4 \\ u_3 \\ \vdots \\ -u_d \\ u_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin t\theta_1 \\ \sin t\theta_1 \\ \sin t\theta_2 \\ \sin t\theta_2 \\ \vdots \\ \sin t\theta_{d/2} \\ \sin t\theta_{d/2} \end{pmatrix}$$

https://self-supervised.cs.jhu.edu/fa2024/

Transformer: RoPE - resources

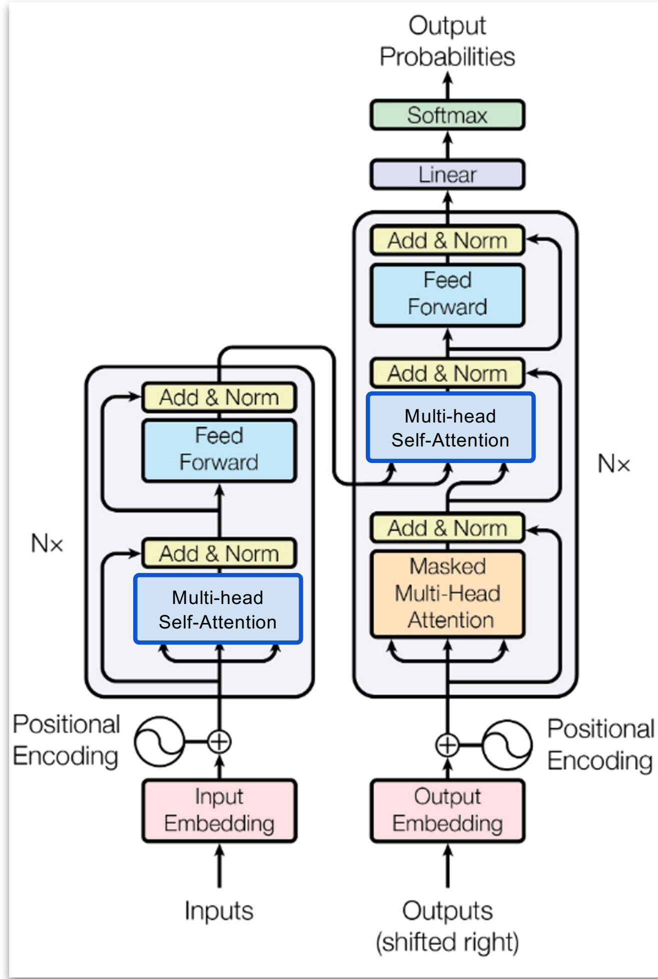
Implementation
https://github.com/lucidrains/rotary-embedding-torch

Blog Post
https://mbrenndoerfer.com/writing/rotary-position-embedding-rope-transformers

Visualization
https://www.abhik.ai/concepts/attention/rotary-position-embeddings

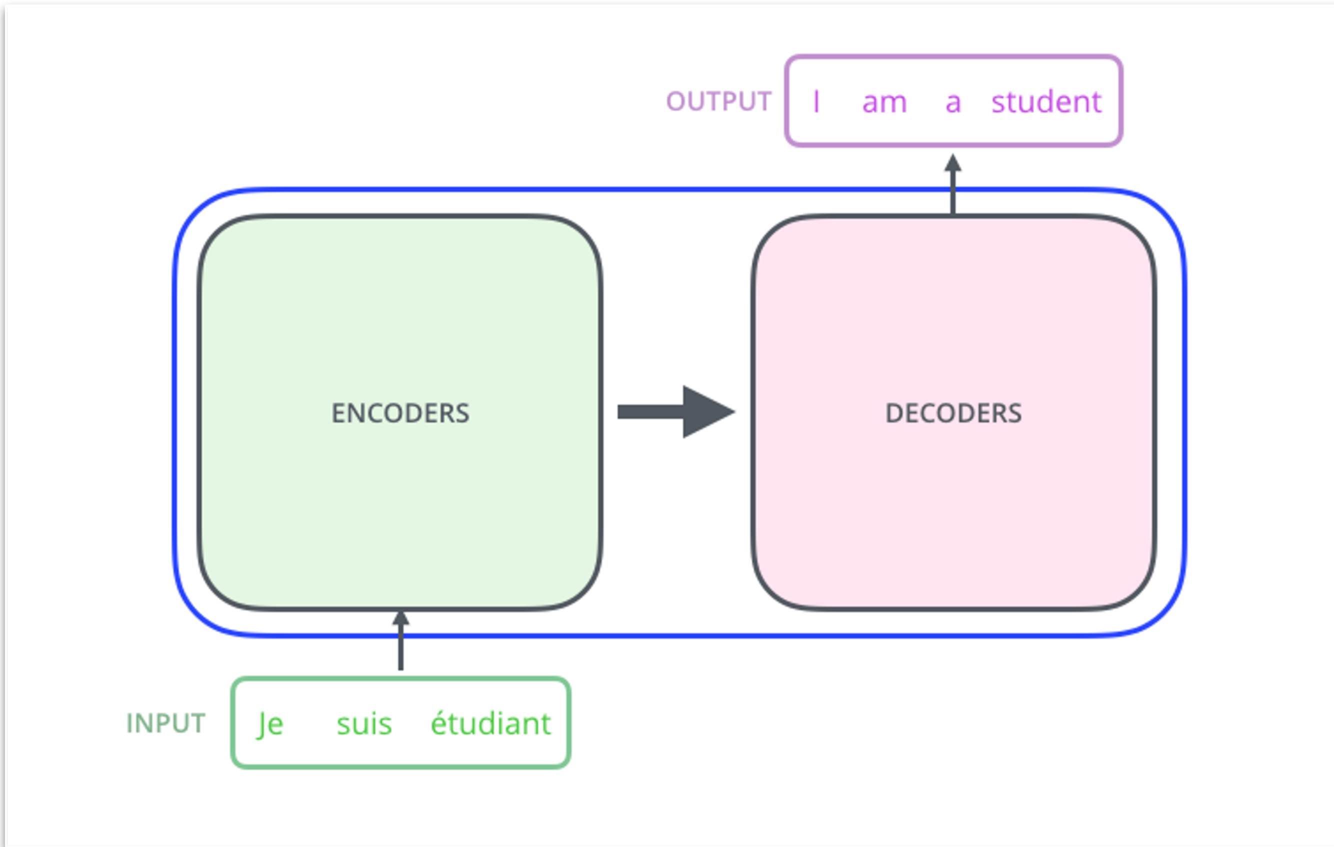
Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward



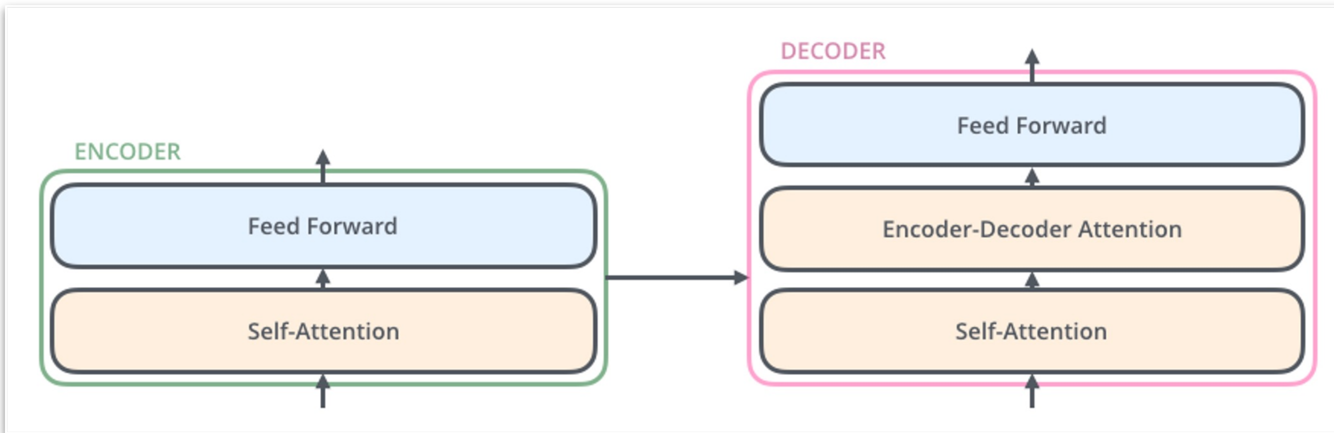
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP



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

Slide Credit: [Prof. Sandra Avila](#) - UNICAMP



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

Slide Credit: Prof. Sandra Avila - UNICAMP

The Transformer Encoder: Dot-Product Attention

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality d_k , value have d_v

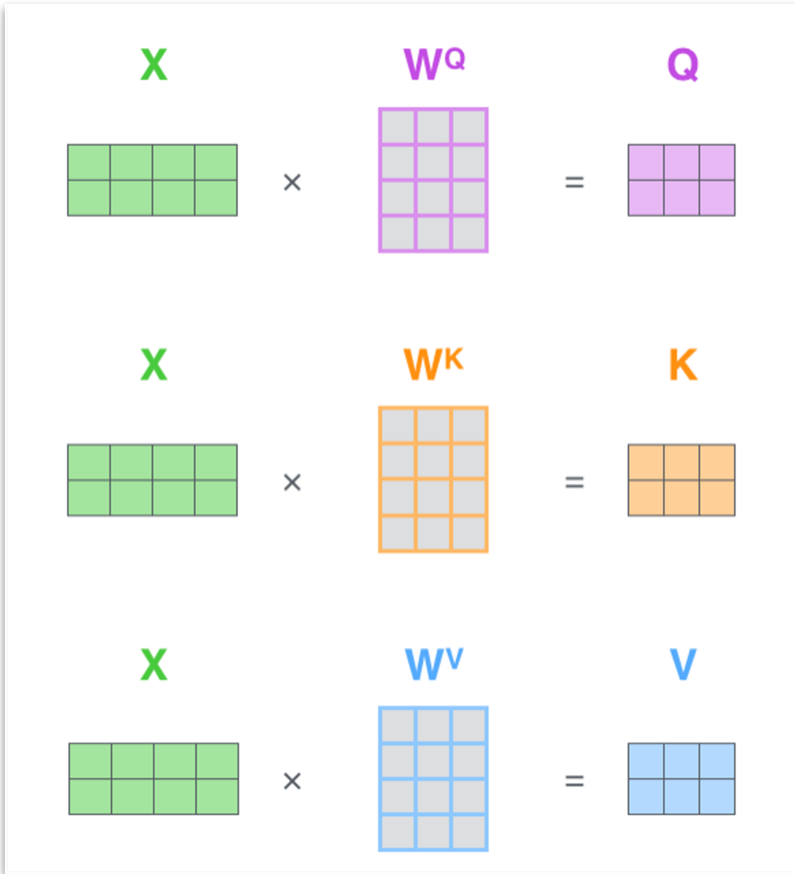
$$A(q, K, V) = \sum_i \boxed{\frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}}} v_i$$

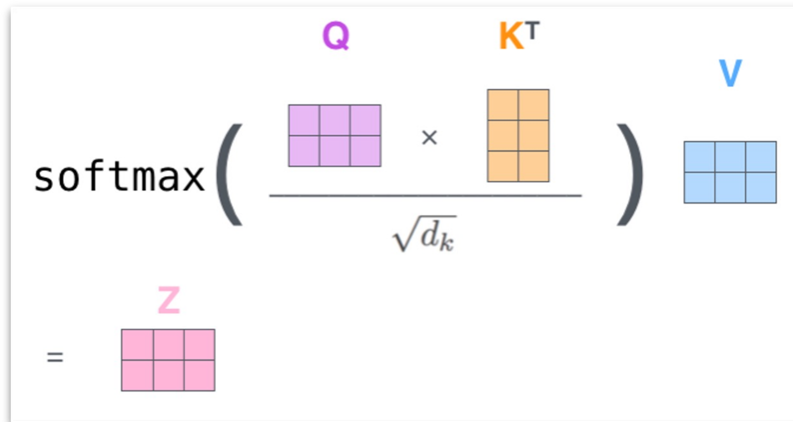
The Transformer Encoder: Key-Query-Value Attention

- We saw that self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular way:
  - Let $x_1, \dots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^{d1}$

- Then keys, queries, values are:
  - $k_i = x_i^{\mathrm{T}}\ \mathrm{W}^{\mathrm{K}}$, where $\mathrm{W}^{\mathrm{K}} \in \mathbb{R}^{d1 \times d2}$ is the key matrix.
  - $q_i = x_i^{\mathrm{T}}\ \mathrm{W}^{\mathrm{Q}}$, where $\mathrm{W}^{\mathrm{Q}} \in \mathbb{R}^{d1 \times d2}$ is the query matrix.
  - $v_i = x_i^{\mathrm{T}}\ \mathrm{W}^{\mathrm{V}}$, where $\mathrm{W}^{\mathrm{V}} \in \mathbb{R}^{d1 \times d2}$ is the value matrix.

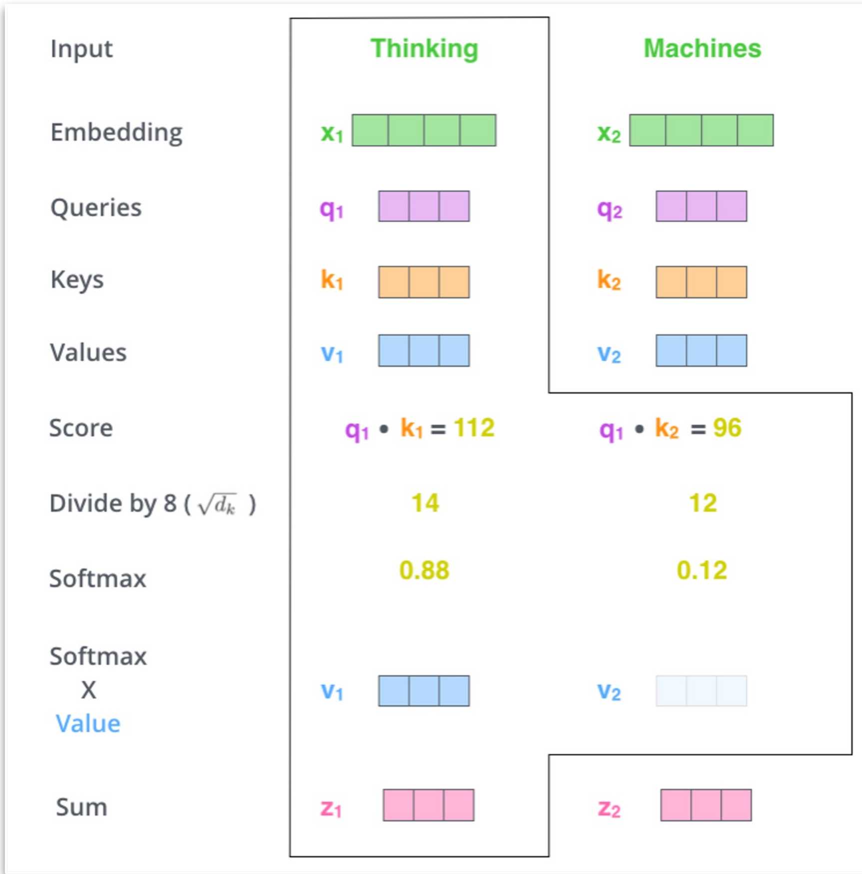
- These matrices allow *different aspects* of the x vectors to be used/emphasized in each of the three roles.

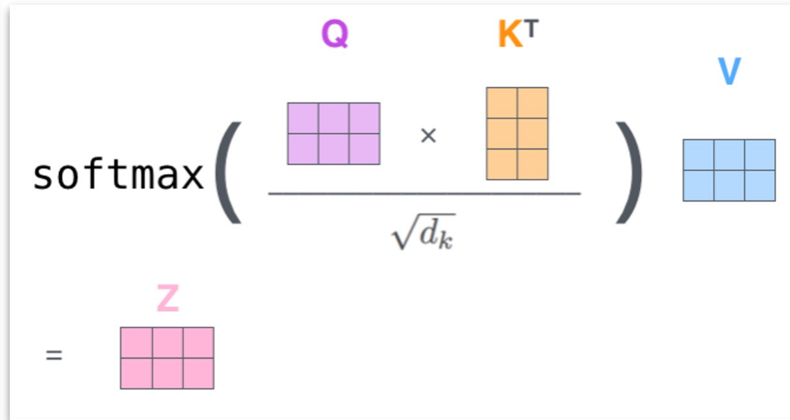




https://jalammar.github.io/illustrated-transformer

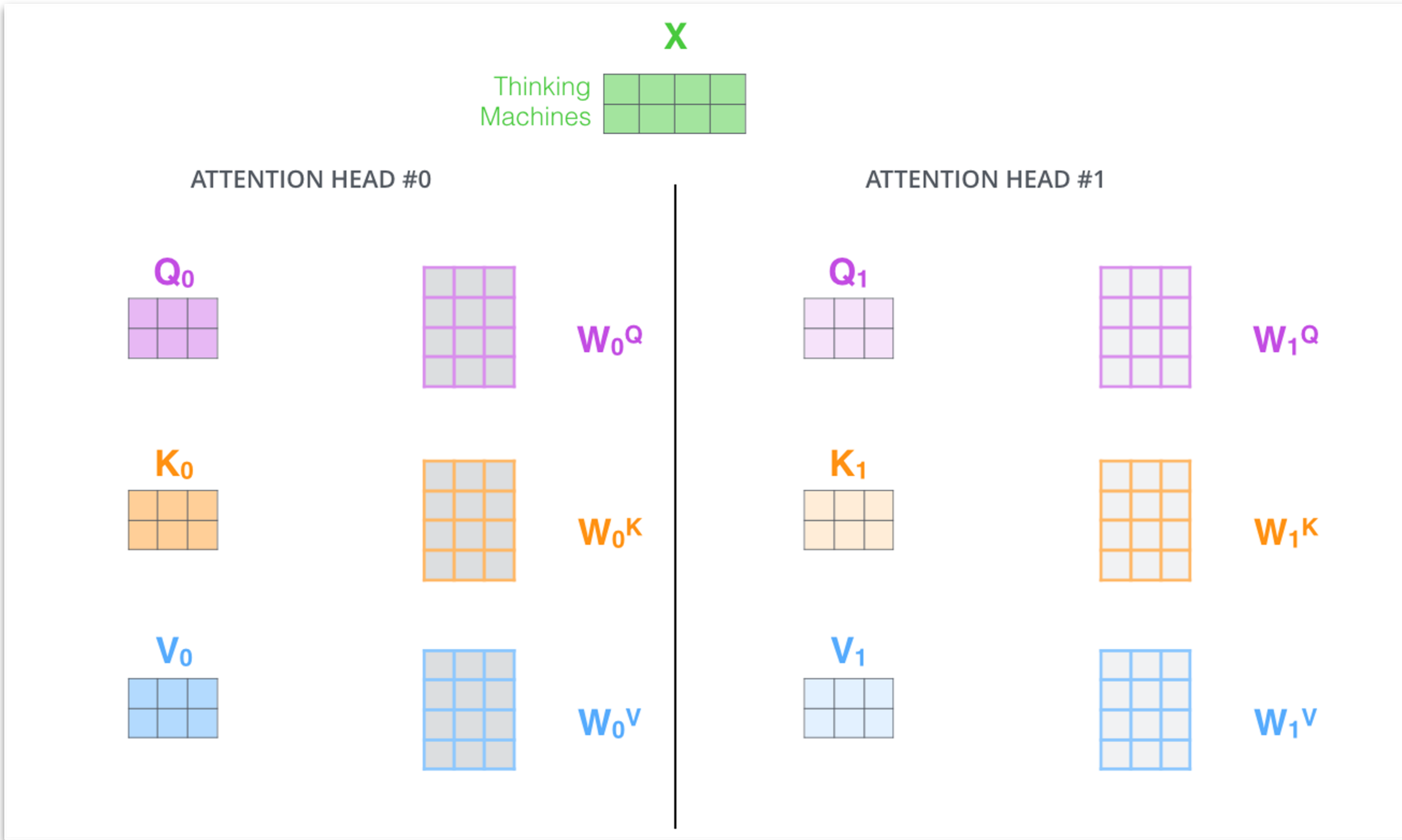
Slide Credit: Prof. Sandra Avila - UNICAMP





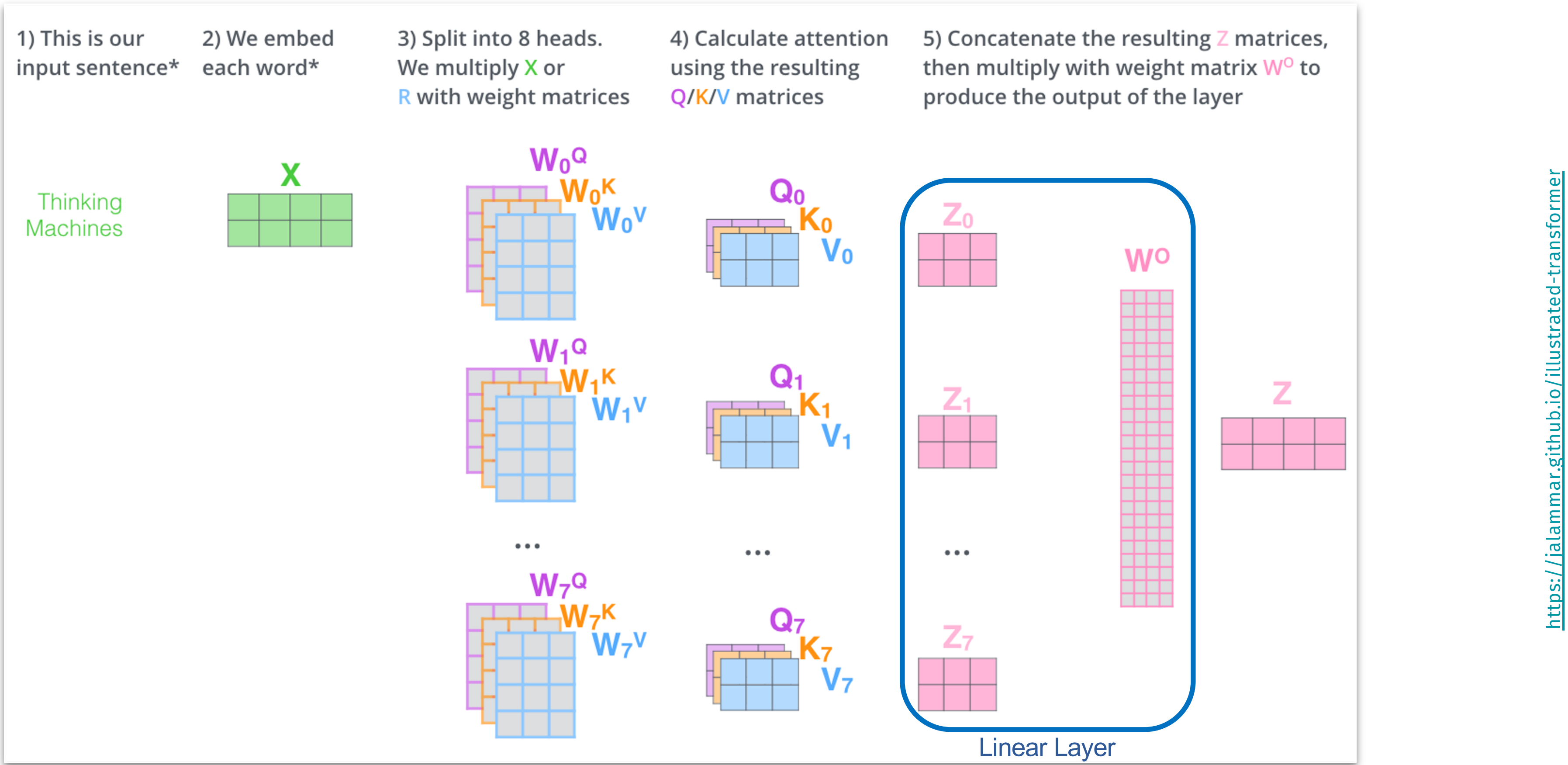
https://jalammar.github.io/illustrated-transformer

Slide Credit: [Prof. Sandra Avila](#) - UNICAMP



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

Slide Credit: [Prof. Sandra Avila](#) - UNICAMP



Slide Credit: [Prof. Sandra Avila](#) - UNICAMP

Recall

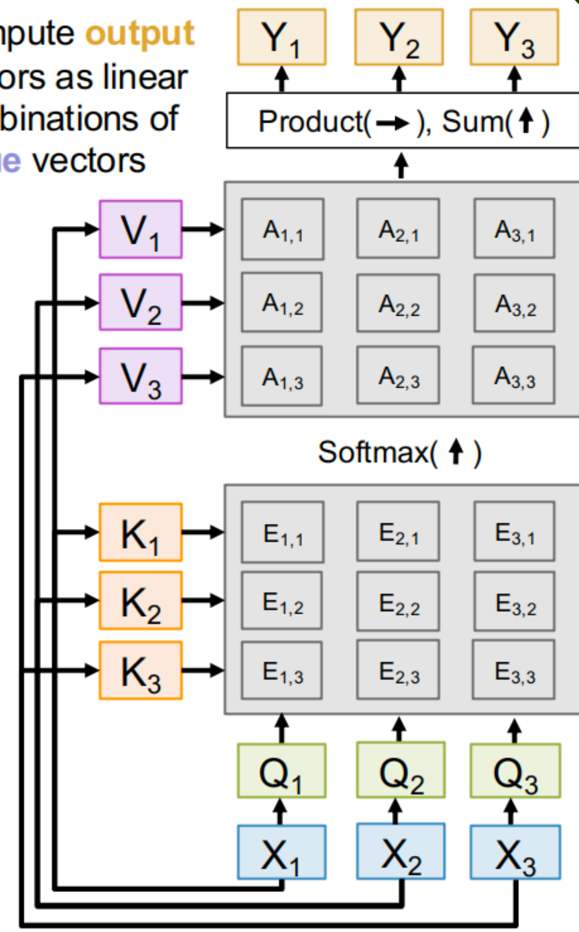
Self-Attention Example

Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Each input produces one output, which is a mix of information from all inputs

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]
Output vector: $Y = AV$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Compute output vectors as linear combinations of value vectors



https://cs231n.stanford.edu/

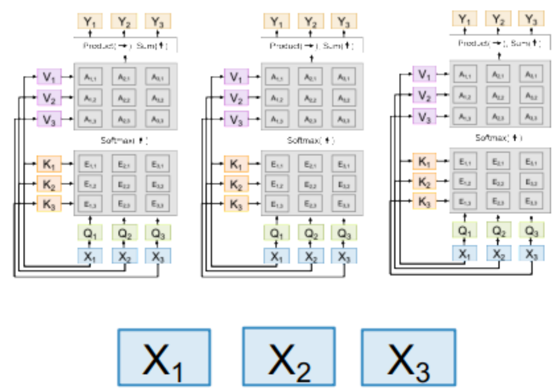
Multiheaded Self-Attention Example

Run H copies of Self-Attention in parallel

Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]
Output vector: $Y = AX$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

H = 3 independent self-attention layers (called heads), each with their own weights



https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

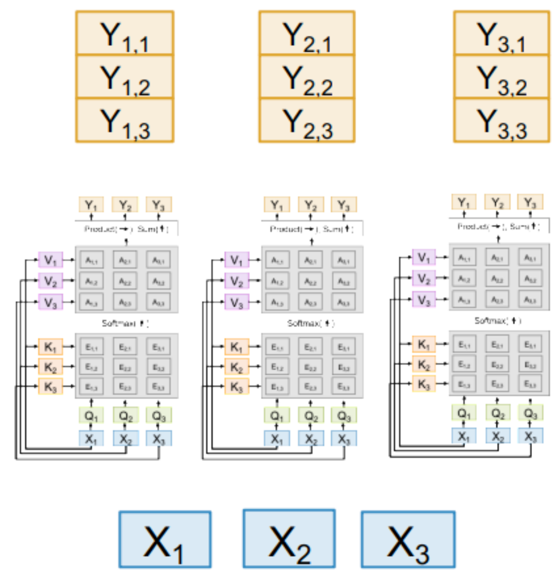
Run H copies of Self-Attention in parallel

Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]
Output vector: $Y = AX$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Stack up the H independent outputs for each input X

H = 3 independent self-attention layers (called heads), each with their own weights



https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Run H copies of Self-Attention in parallel

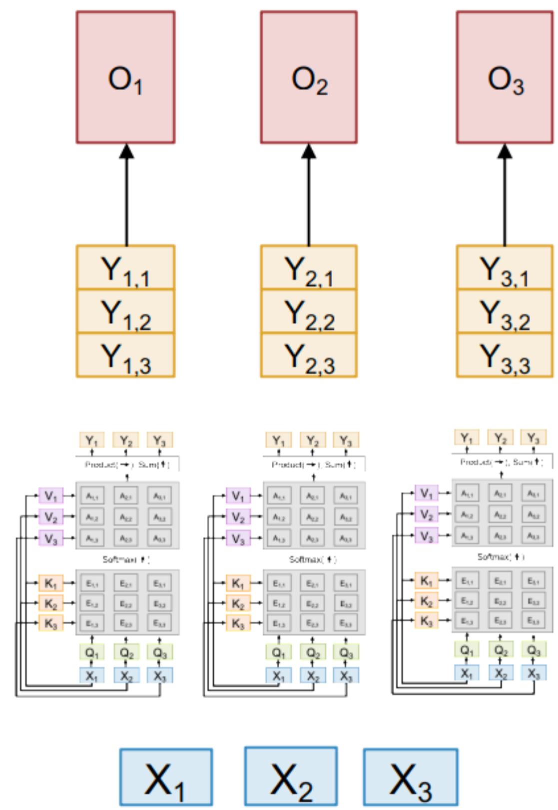
Inputs:
Input vectors: X [N x D_{in}]
Key matrix: W_K [D_{in} x D_{out}]
Value matrix: W_V [D_{in} x D_{out}]
Query matrix: W_Q [D_{in} x D_{out}]

Computation:
Queries: $Q = XW_Q$ [N x D_{out}]
Keys: $K = XW_K$ [N x D_{out}]
Values: $V = XW_V$ [N x D_{out}]
Similarities: $E = QK^T / \sqrt{D_Q}$ [N x N]
$E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]
Output vector: $Y = AX$ [N x D_{out}]
$Y_i = \sum_j A_{ij} V_j$

Output projection fuses data from each head

Stack up the H independent outputs for each input X

H = 3 independent self-attention layers (called heads), each with their own weights



https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Run H copies of Self-Attention in parallel

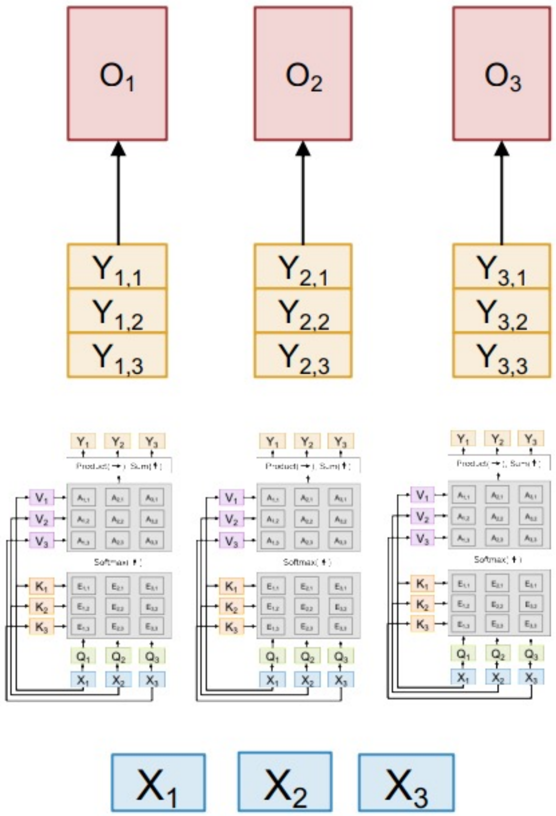
Inputs:
Input vectors: **X** [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

Each of the H parallel layers use a qkv dim of D_H = "head dim"

Usually $D_H = D / H$, so inputs and outputs have the same dimension

https://cs231n.stanford.edu/



Multiheaded Self-Attention Example

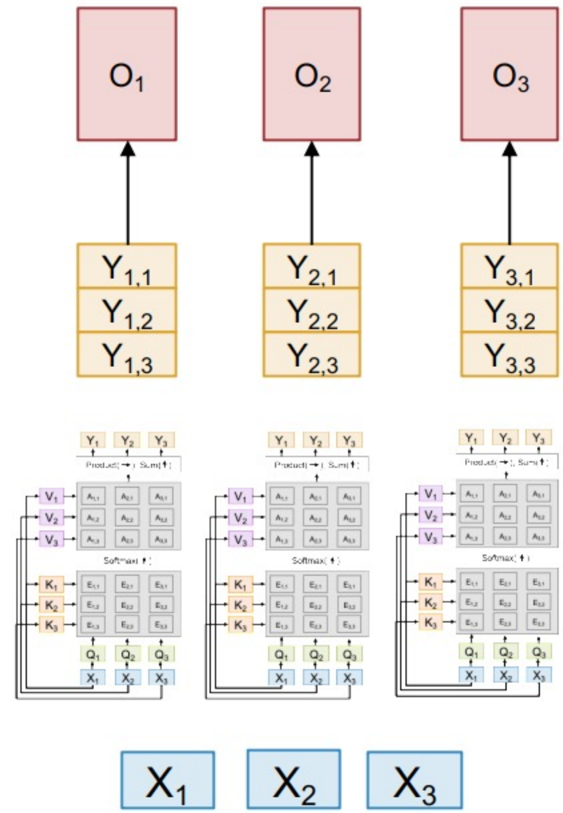
Run H copies of Self-Attention in parallel

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

In practice, compute all H heads in parallel using batched matrix multiply operations.

Used everywhere in practice.

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]



https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: $A = \text{softmax}(E, \text{dim}=2)$ [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
$\boxed{\textbf{Head outputs}: Y = AV \text{ [H x N x } D_H] \Rightarrow \text{[N x } HD_H]}$
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]
4. Output Projection
   [N x HD_H] [HD_H x D] => [N x D]

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]
4. Output Projection
   [N x HD_H] [HD_H x D] => [N x D]

Q: How much compute does this take as the number of vectors N increases?

A: $O(N^2)$

https://cs231n.stanford.edu/

Multiheaded Self-Attention Example

Inputs:
Input vectors: X [N x D]
Key matrix: W_K [D x HD_H]
Value matrix: W_V [D x HD_H]
Query matrix: W_Q [D x HD_H]
Output matrix: W_O [HD_H x D]

Computation:
Queries: $Q = XW_Q$ [H x N x D_H]
Keys: $K = XW_K$ [H x N x D_H]
Values: $V = XW_V$ [H x N x D_H]
Similarities: $E = QK^T / \sqrt{D_Q}$ [H x N x N]
Attention weights: A = softmax(E, dim=2) [H x N x N]
Head outputs: $Y = AV$ [H x N x D_H] => [N x HD_H]
Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]
4. Output Projection
   [N x HD_H] [HD_H x D] => [N x D]

Q: How much memory does this take as the number of vectors N increases?

~~A: $O(N^2)$~~ A: $O(N)$

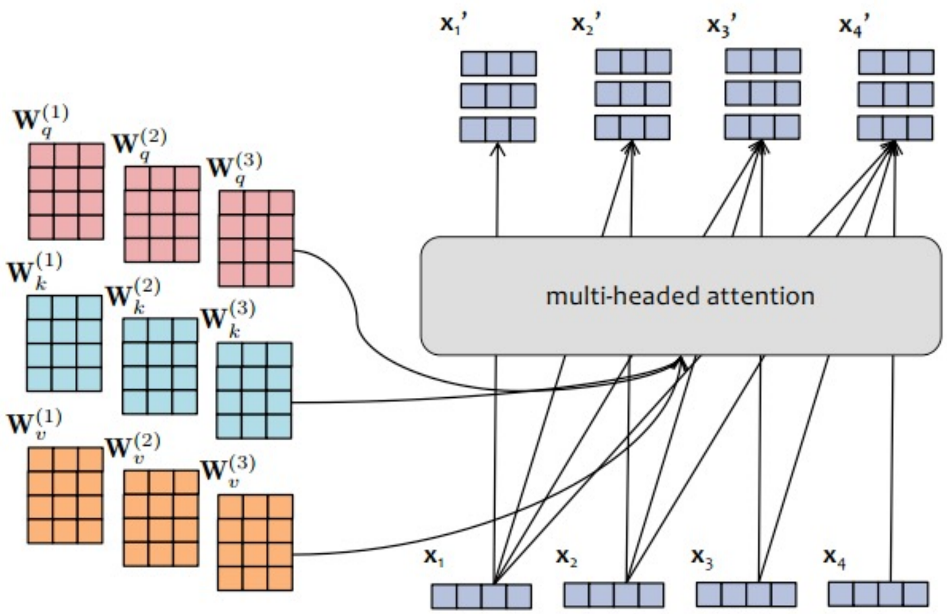
https://cs231n.stanford.edu/

Dao et al, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", 2022

How can we improve Self-Attention?

Matrix Version of Multi-Headed Attention

Recall



$$\mathbf{X}' = \text{concat}(\mathbf{X}'^{(1)}, \mathbf{X}'^{(2)}, \mathbf{X}'^{(3)})$$

$$\mathbf{X}'^{(i)} = \text{softmax}\left(\frac{\mathbf{Q}^{(i)}(\mathbf{K}^{(i)})^T}{\sqrt{d_k}}\right)\mathbf{V}^{(i)}$$

$$\mathbf{Q}^{(i)} = \mathbf{X}\mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_4]^T$$

https://www.cs.cmu.edu/~mgormley/courses/10423/

Reduce Parameters: Grouped Query Attention (GQA)

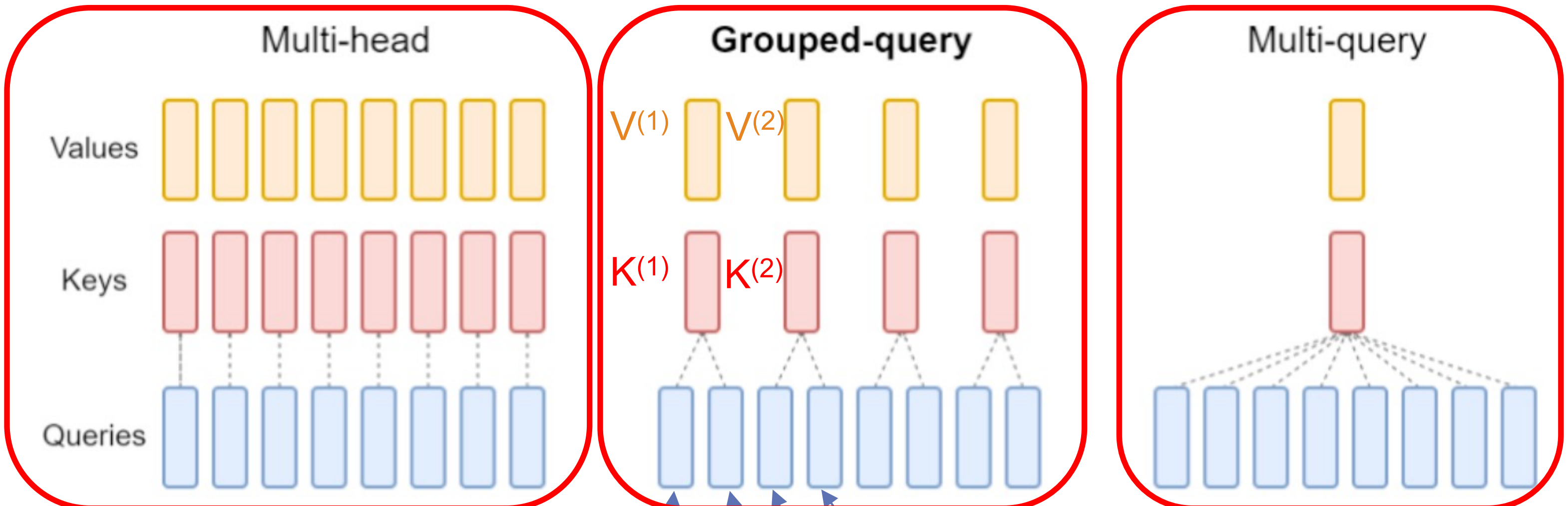
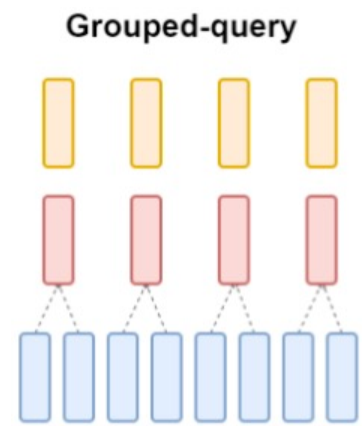


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Figure from http://arxiv.org/abs/2305.13245

Grouped Query Attention (GQA)



- Key idea: reuse the same key-value heads for multiple different query heads.
- Parameters: The parameter matrices are all the same size, but now we have fewer key/value parameter matrices (heads) than query parameter matrices (heads).

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^T$$

$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$

$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$

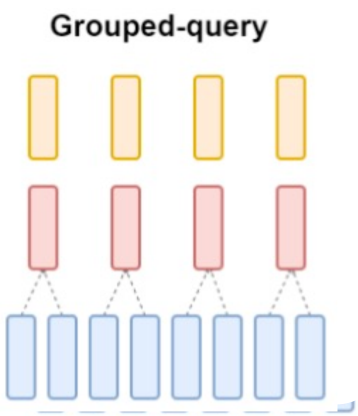
$$\mathbf{Q}^{(i,j)} = \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

- h_q = the number of query heads
- h_{kv} = the number of key/value heads
- Assume h_q is divisible by h_{kv}
- $g = h_q / h_{kv}$ is the size of each group (i.e. the number of query vectors per key/value vector).

$h_q = 8$
$h_{kv} = 4$
$g = ??$

Figure from http://arxiv.org/abs/2305.13245

Grouped Query Attention (GQA)



- Key idea: reuse the same key-value heads for multiple different query heads.

- Parameters: The parameter matrices are all the same size, but now we have fewer key/value parameter matrices (heads) than query parameter matrices (heads).

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^T$$

$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$

$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$

$$\mathbf{Q}^{(i,j)} = \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{S}^{(i,j)} = \mathbf{Q}^{(i,j)}(\mathbf{K}^{(i)})^T / \sqrt{d_k}, \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{A}^{(i,j)} = \text{softmax}(\mathbf{S}^{(i,j)}), \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{X}'^{(i,j)} = \mathbf{A}^{(i,j)}\mathbf{V}^{(i)}, \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{X}' = \text{concat}(\mathbf{X}'^{(i,j)}), \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{X} = \mathbf{X}'\mathbf{W}_o \qquad \text{(where } \mathbf{W}_o \in \mathbb{R}^{d_{model} \times d_{model}}\text{)}$$

Figure from http://arxiv.org/abs/2305.13245

Reduce Parameters: Multi-Head Latent Attention

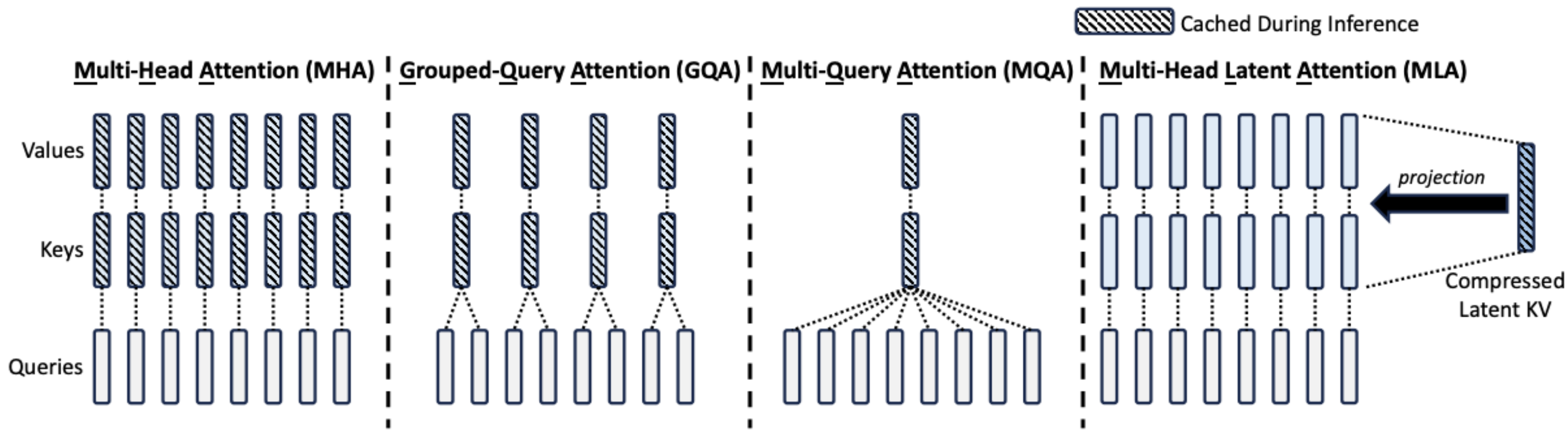
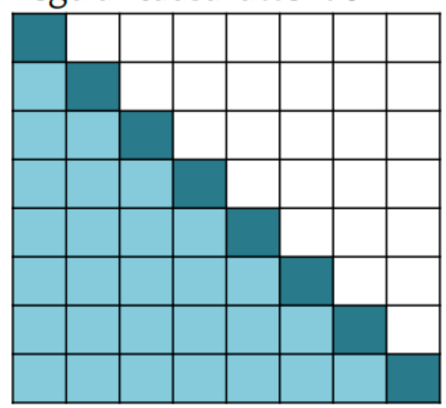


Figure from DeepSeek - https://arxiv.org/pdf/2405.04434

https://learning.oreilly.com/library/view/hands-on-llm-serving/9798341621480/

Reduce Space: Sliding Window Attention

- Also called “local attention” and introduced for the Longformer model (2020).
- Problem: regular attention is computationally expensive and requires a lot of memory.
- Solution: apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)



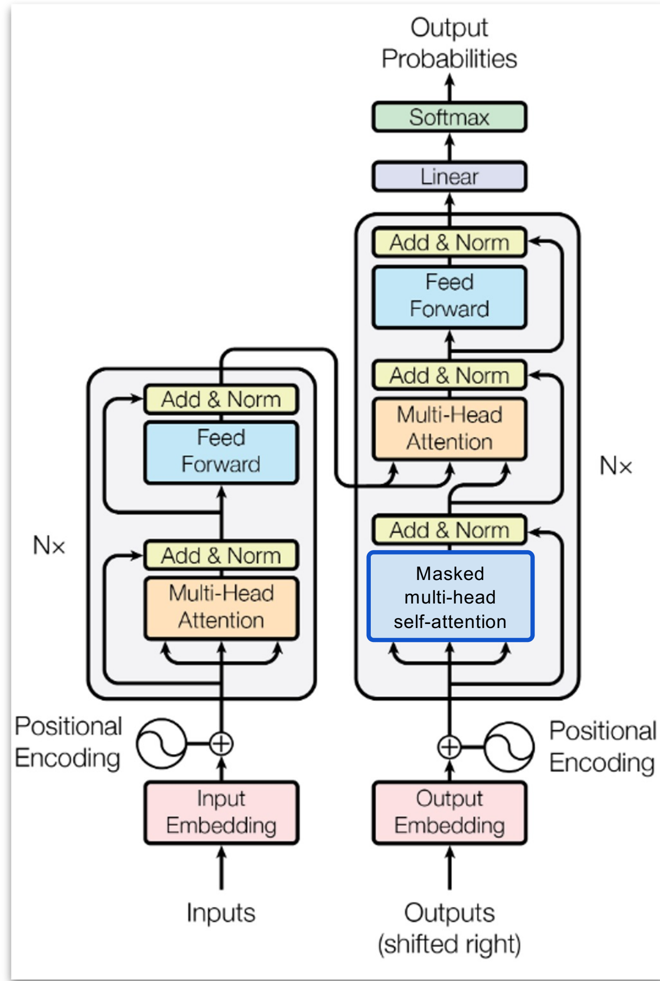
$$\mathbf{X}' = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure from http://arxiv.org/abs/2305.13245

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward

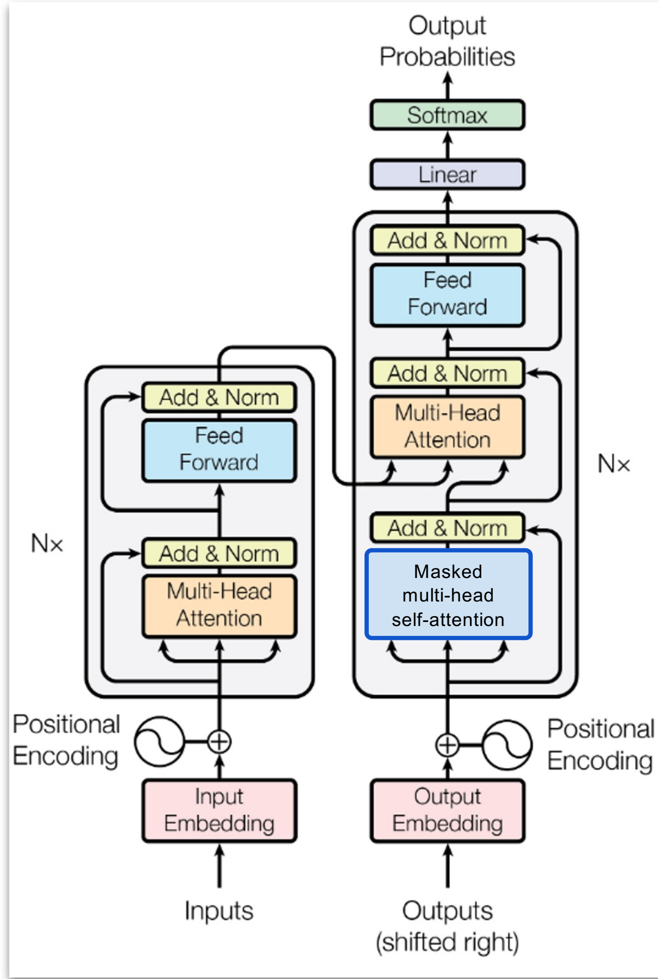


https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - **Masked multi-head attention**
  - Residual connections
  - Layer Normalization
  - Feedforward

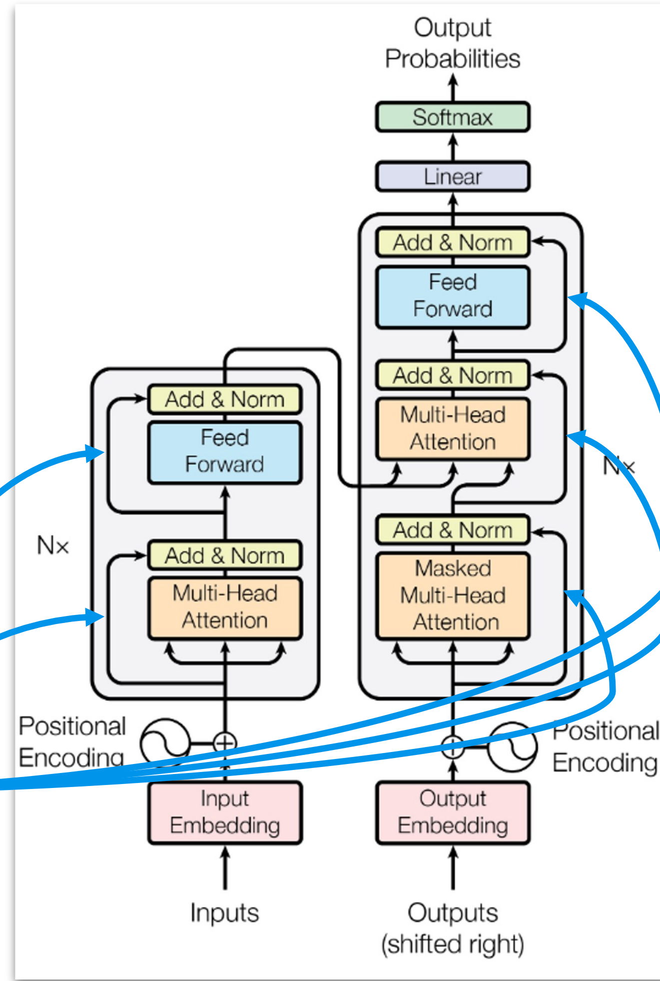


https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - **Residual connections**
  - Layer Normalization
  - Feedforward



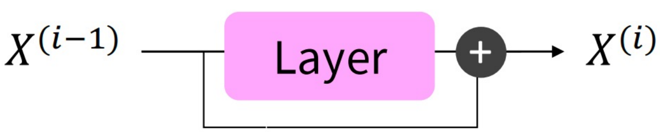
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

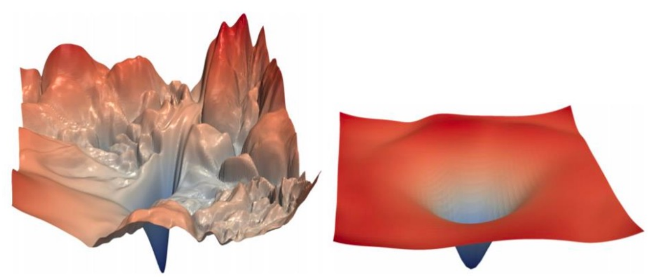
Transformer

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



[no residuals] [residuals]

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

https://web.stanford.edu/class/cs224n/

Transformer

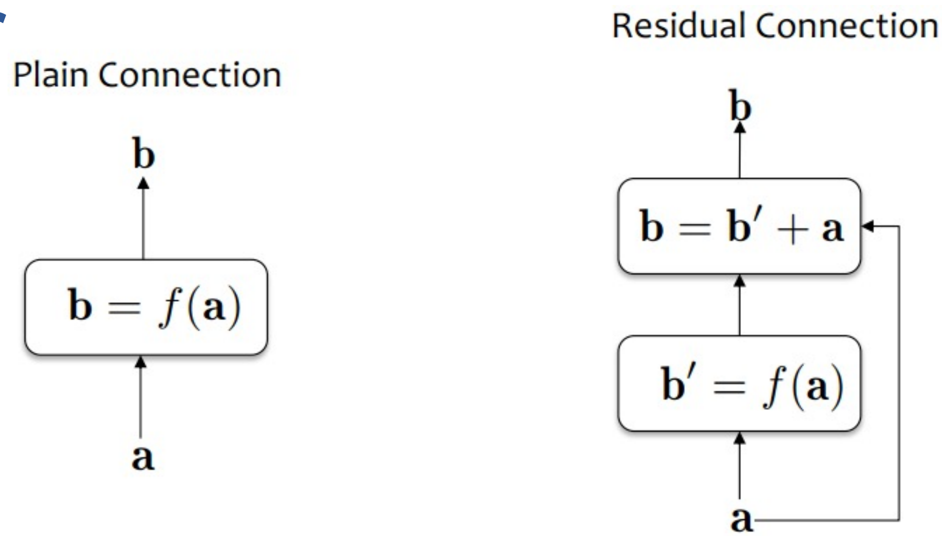


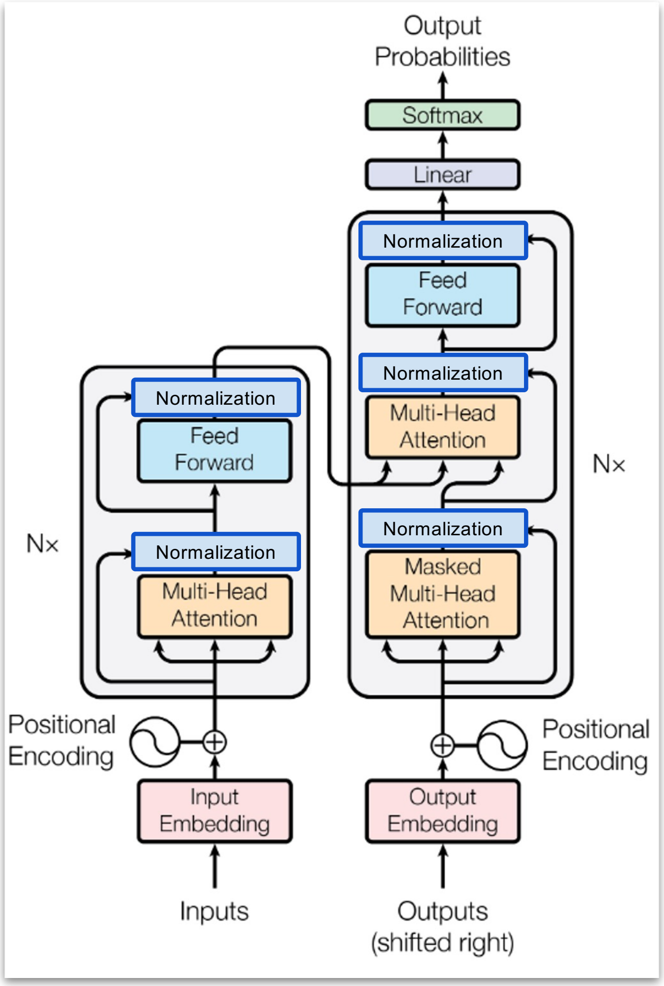
Figure from https://arxiv.org/pdf/1512.03385.pdf

Why are residual connections helpful?

Instead of f(a) having to learn a full transformation of a, f(a) only needs to learn an additive modification of a (i.e. the residual).

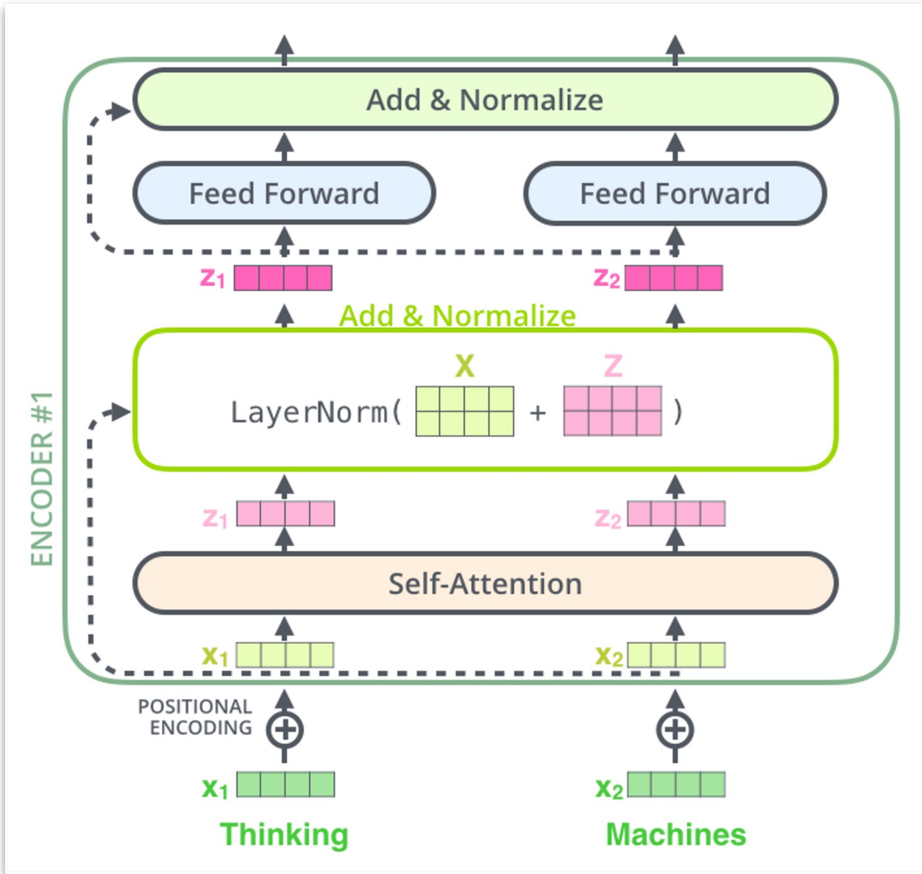
Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward



Slide Credit: Prof. Sandra Avila - UNICAMP

https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

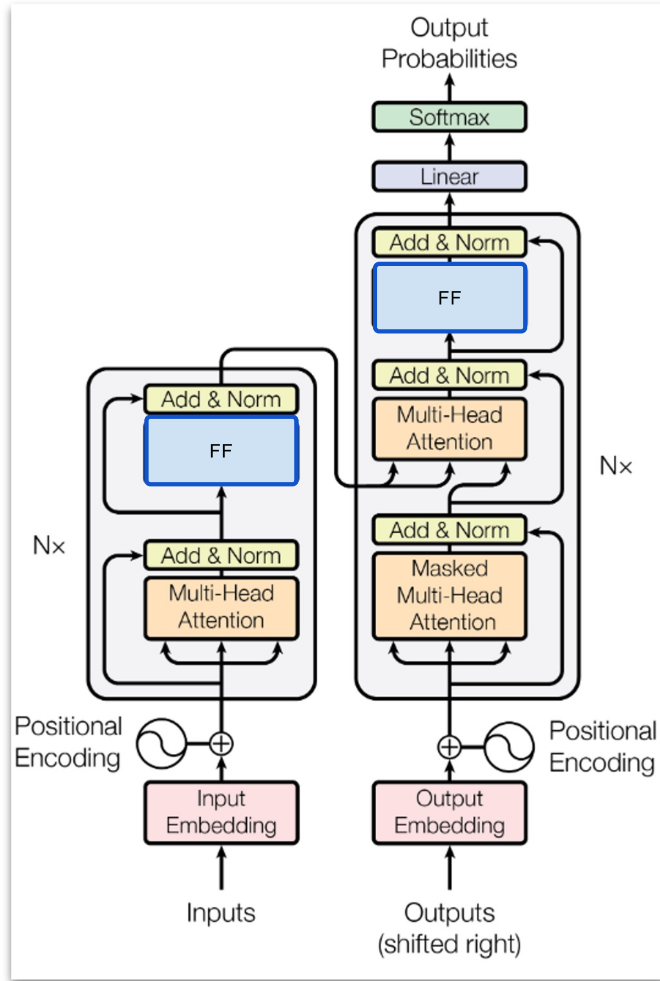


https://jalammar.github.io/illustrated-transformer

Slide Credit: [Prof. Sandra Avila](#) - UNICAMP

Transformer

- Transformer Architecture
  - Encoder & Decoder
  - Input & output embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention
  - Masked multi-head attention
  - Residual connections
  - Layer Normalization
  - Feedforward

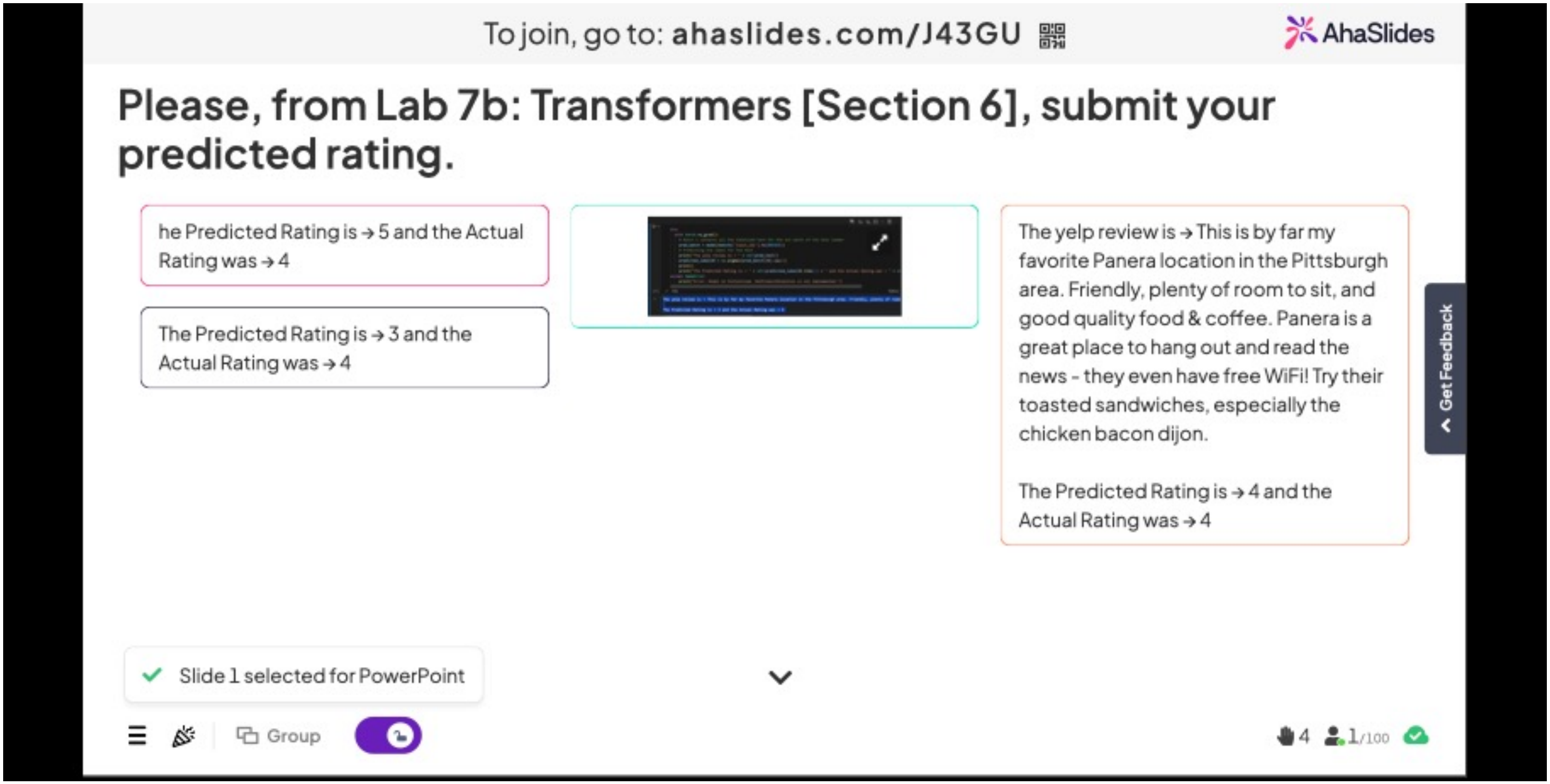


https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

Lab 7b: Transformers

Duration: 20 min

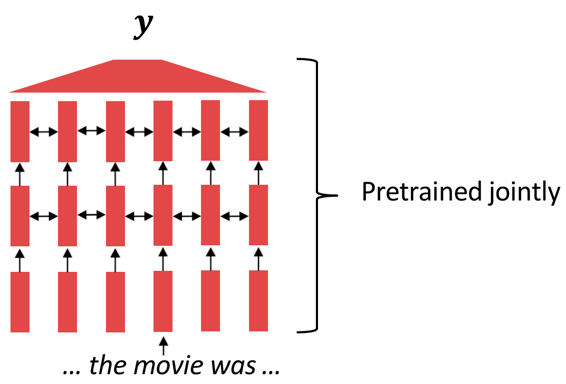

To join, go to: ahaslides.com/J43GU
AhaSlides
Please, from Lab 7b: Transformers [Section 6], submit your predicted rating.
he Predicted Rating is → 5 and the Actual Rating was → 4
The Predicted Rating is → 3 and the Actual Rating was → 4
The yelp review is → This is by far my favorite Panera location in the Pittsburgh area. Friendly, plenty of room to sit, and good quality food & coffee. Panera is a great place to hang out and read the news - they even have free WiFi! Try their toasted sandwiches, especially the chicken bacon dijon.
The Predicted Rating is → 4 and the Actual Rating was → 4
Get Feedback
Slide 1 selected for PowerPoint
Group
4
1/100

Pretraining models

In modern NLP:

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

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.



[This model has learned how to represent entire sentences through pretraining]

Adapted from John Hewitt

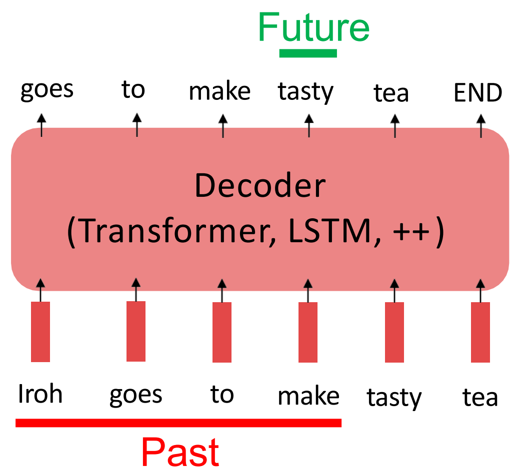
Pretraining through language modeling [Dai and Le, 2015]

Recall the **language modeling** task:

- Model $p_\theta(w_t \mid w_{1:t-1})$, 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.



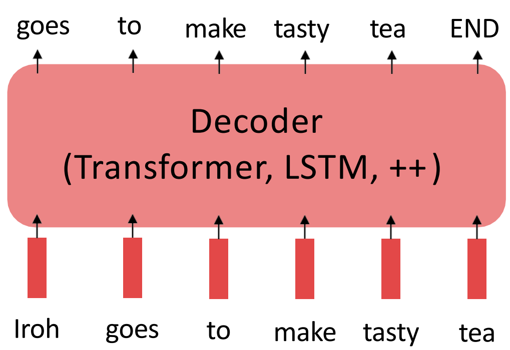
Lab 7a

The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

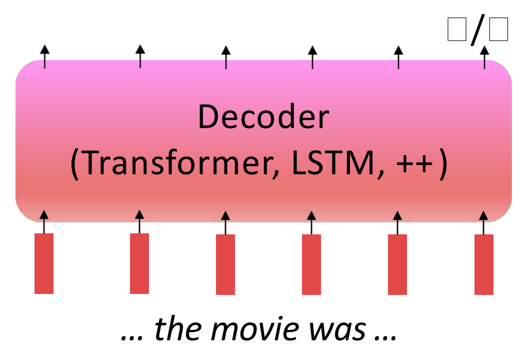
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Generative Pretrained Transformer (GPT) [Radford et al., 2018]

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

- Transformer decoder with 12 layers.
- 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"

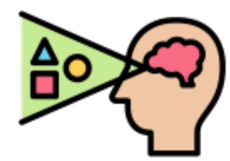
[Devlin et al., 2018]

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

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



GPT Tokenizer

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**.

GPT-3, in-context learning, 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.

GPT-3 has 175 billion parameters.

Adapted from John Hewitt

GPT-3, in-context learning, 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.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context):

" thanks -> merci
hello -> bonjour
mint -> menthe
otter -> "

Output (conditional generations):

loutre…"

GPT-3: Prompt Engineering

Translate English to French

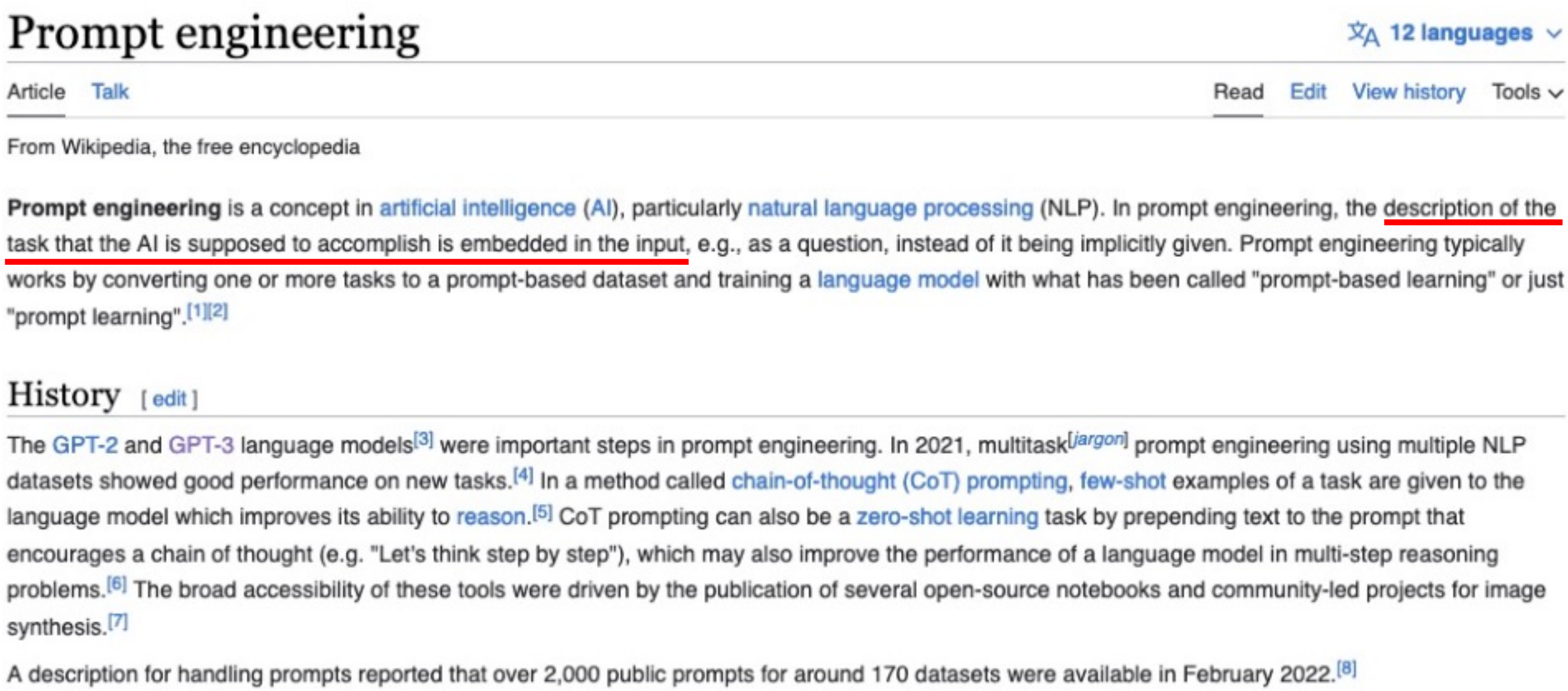
```
sea otter => loutre de mer

peppermint => menthe poivrée

plush girafe => girafe peluche

cheese => ............................
```

Language Models are Few-Shot Learners

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei

https://arxiv.org/abs/2005.14165

Prompt Engineering


Prompt engineering

文A 12 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Prompt engineering is a concept in artificial intelligence (AI), particularly natural language processing (NLP). In prompt engineering, the description of the task that the AI is supposed to accomplish is embedded in the input, e.g., as a question, instead of it being implicitly given. Prompt engineering typically works by converting one or more tasks to a prompt-based dataset and training a language model with what has been called "prompt-based learning" or just "prompt learning".[1][2]

History [edit]

The GPT-2 and GPT-3 language models[3] were important steps in prompt engineering. In 2021, multitask[jargon] prompt engineering using multiple NLP datasets showed good performance on new tasks.[4] In a method called chain-of-thought (CoT) prompting, few-shot examples of a task are given to the language model which improves its ability to reason.[5] CoT prompting can also be a zero-shot learning task by prepending text to the prompt that encourages a chain of thought (e.g. "Let's think step by step"), which may also improve the performance of a language model in multi-step reasoning problems.[6] The broad accessibility of these tools were driven by the publication of several open-source notebooks and community-led projects for image synthesis.[7]

A description for handling prompts reported that over 2,000 public prompts for around 170 datasets were available in February 2022.[8]

How would you come with a solution for this problem?

The kid is throwing rocks at the window

The <subject>kid</subject> is throwing <object>rocks</object> at the <destination>window</destination>

Prompt Engineering

Input: The cat is throwing the ball into the ground
Output: The <subject>cat</subject> is throwing the <object>ball</object> into the <destination>ground</ground>

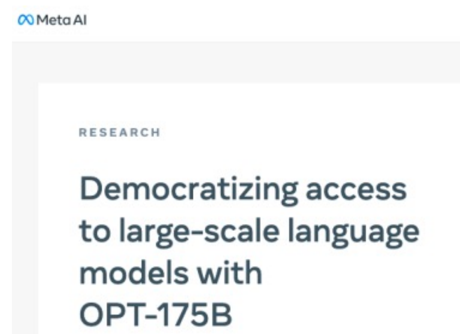
Input: The snake is being attacked by the wolf
Output: The <object>snake</object> is being attacked by the <actor>wolf</actor>

Input: The kid is throwing rocks at the window
Output:

Prompt Engineering

- Any Large Language Model (LLM) such as GPT-3 can be turned into a general-purpose problem solver in this way.
- Obviously, it is not going to work well for every use case.
- Other Large Language Models trained at the scale of GPT-3 that are actually publicly available.
- BLOOM-176B and OPT-175B:

https://huggingface.co/bigscience/bloom



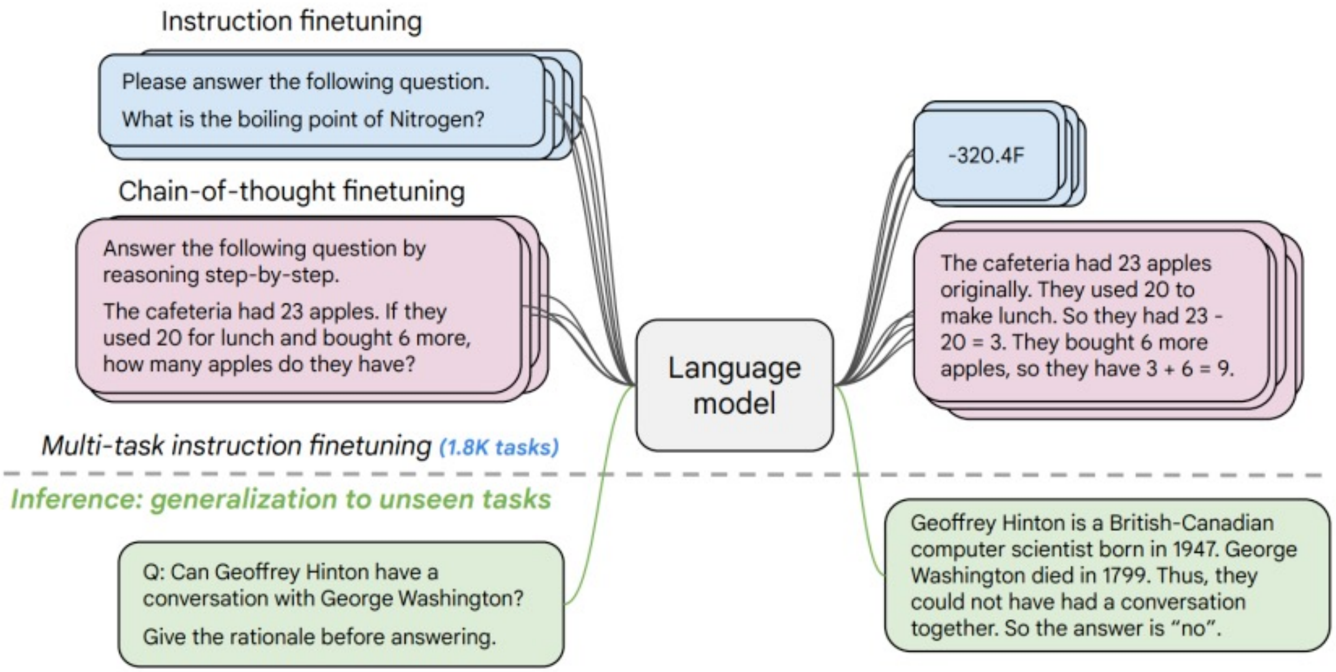
https://huggingface.co/docs/transformers/model_doc/opt

However, these are still limited

- Predicting the next word can lead to intelligent behavior such as the one exemplified earlier however this still limited

- What makes some of the new LLMs special? ChatGPT (GPT-3.5, 3.5 Turbo, 4, 4-turbo), FLAN-T5, OPT-IML

Instruction Tuning (e.g. FLAN-T5 by Google)



Lab 7a

https://arxiv.org/pdf/2210.11416.pdf

FLAN-T5

Model input (Disambiguation QA)

Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.

Sentence: The reporter and the chef will discuss their favorite dishes.

Options:
(A) They will discuss the reporter's favorite dishes
(B) They will discuss the chef's favorite dishes
(C) Ambiguous

A: Let's think step by step.

Before instruction finetuning

The reporter and the chef will discuss their favorite dishes.
The reporter and the chef will discuss the reporter's favorite dishes.
The reporter and the chef will discuss the chef's favorite dishes.
The reporter and the chef will discuss the reporter's and the chef's favorite dishes.

✖ (doesn't answer question)

FLAN-T5

Model input (Disambiguation QA)

Q: In the following sentences, explain the antecedent of the pronoun (which thing the pronoun refers to), or state that it is ambiguous.

Sentence: The reporter and the chef will discuss their favorite dishes.

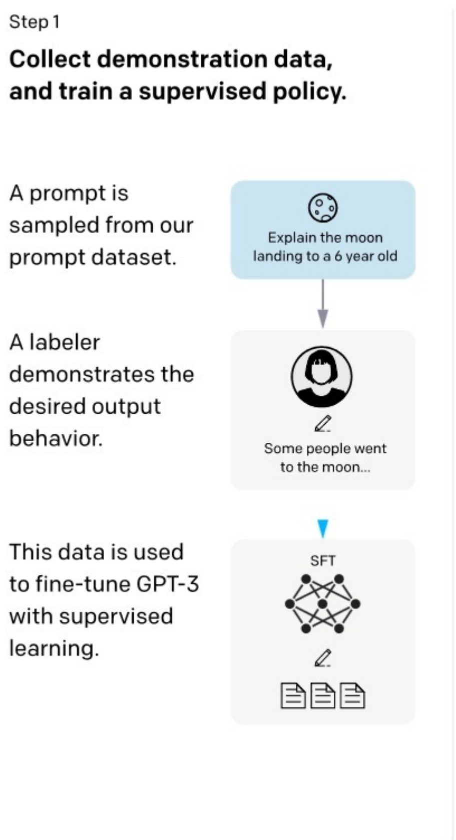
Options:
(A) They will discuss the reporter's favorite dishes
(B) They will discuss the chef's favorite dishes
(C) Ambiguous

A: Let's think step by step.

After instruction finetuning

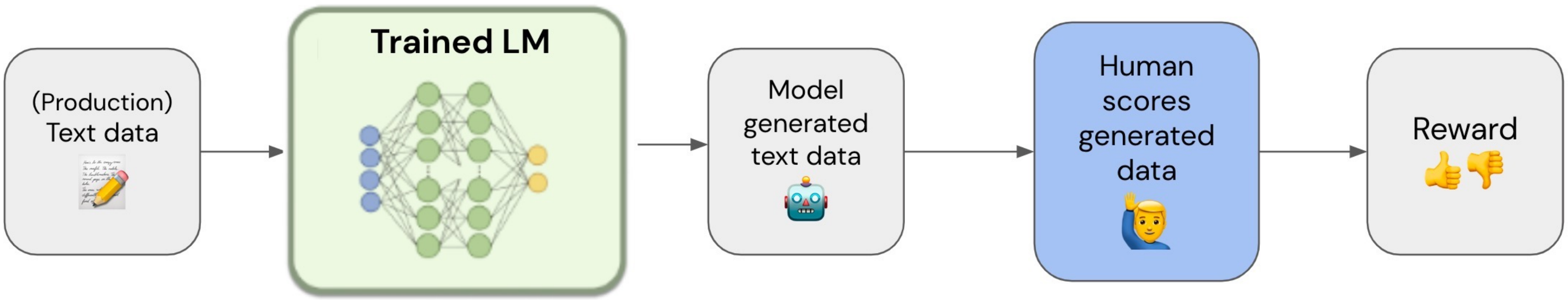
The reporter and the chef will discuss their favorite dishes does not indicate whose favorite dishes they will discuss. So, the answer is (C).

InstructGPT (ChatGPT)



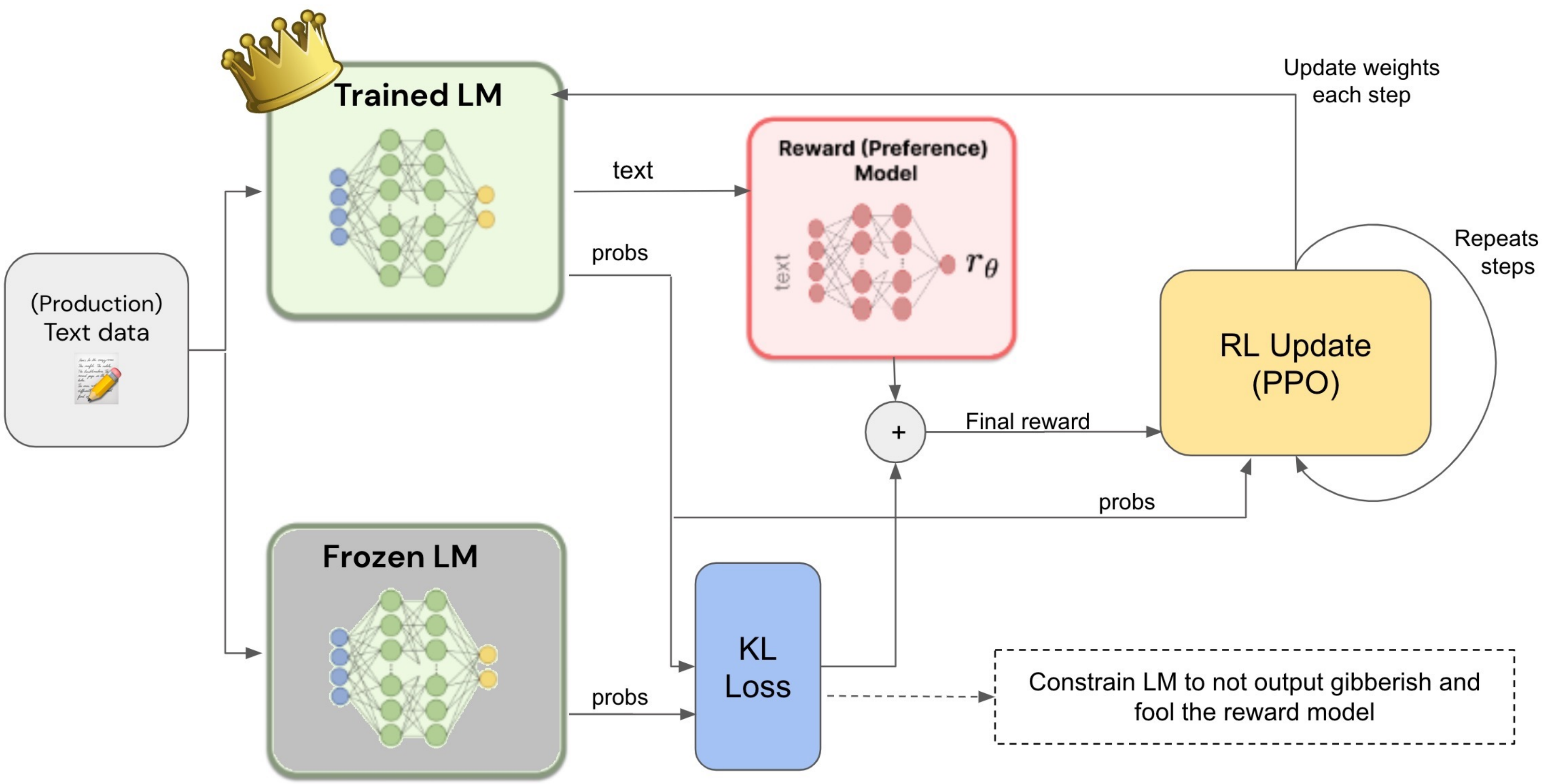
https://arxiv.org/abs/2203.02155

Step by step: Train a reward model that learns from Human Ratings (e.g. from 1 to 5)



https://gist.github.com/JoaoLages/c6f2dfd13d2484aa8bb0b2d567fbf093

Step by step: Train the LM to generate text that get high reward but still produces stuff that makes sense



https://gist.github.com/JoaoLages/c6f2dfd13d2484aa8bb0b2d567fbf093

Pretraining encoders: What pretraining objective to use?

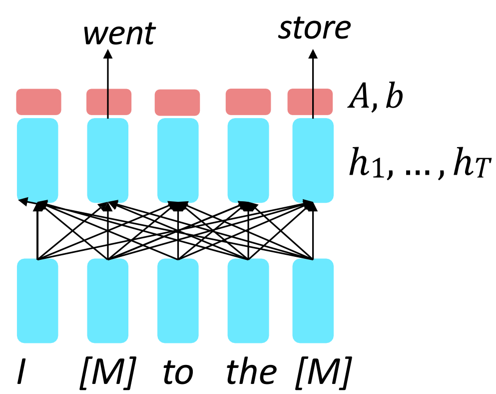
So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$

$$y_i \sim Aw_i + b$$

Only add loss terms from words that are "masked out." If x' is the masked version of x, we're learning $p_\theta(x|x')$. Called **Masked LM**.



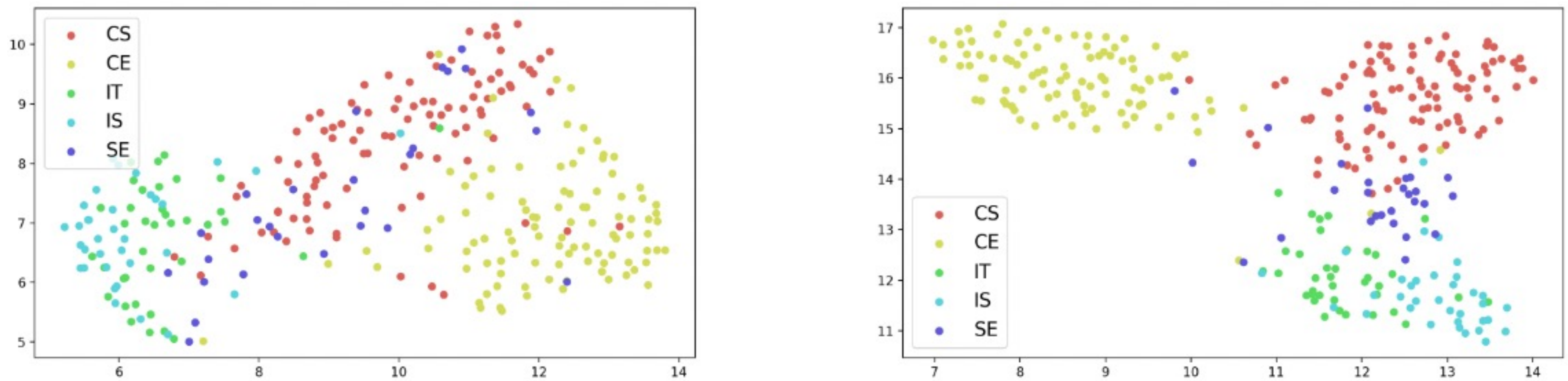
[Devlin et al., 2018]

Example: BERT: Bidirectional Encoder Representations from Transformers

Adapted from John Hewitt

Case Study: Improving Embeddings Representations for Comparing Higher Education Curricula

- Umap (MacInnes et al, JOSS 2018) visualizations for Bert and our approach.
- Our approach separates computing programs more clearly.



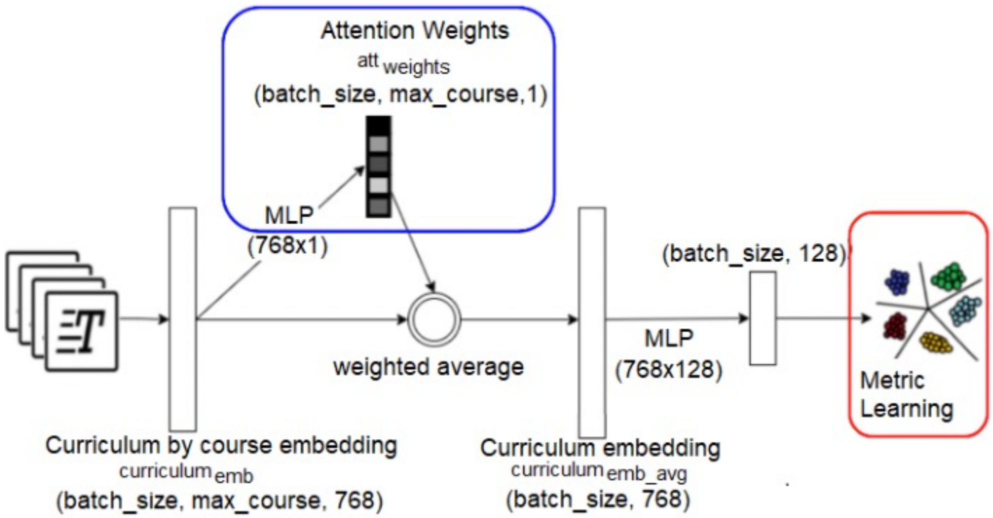
(a) $Bert$ (b) $Bert_{met+att}$

Murrugarra-Llerena et al, EMNLP 2022

Case Study: Improving Embeddings Representations for Comparing Higher Education Curricula

- Course-Based attention: Identifies the most and the least important courses following the intuition of core and elective courses.

- Metric Learning: Learns boundaries to form well-defined groups.



Murrugarra-Llerena et al, EMNLP 2022

Capturing meaning via context: What kinds of things does pretraining learn?

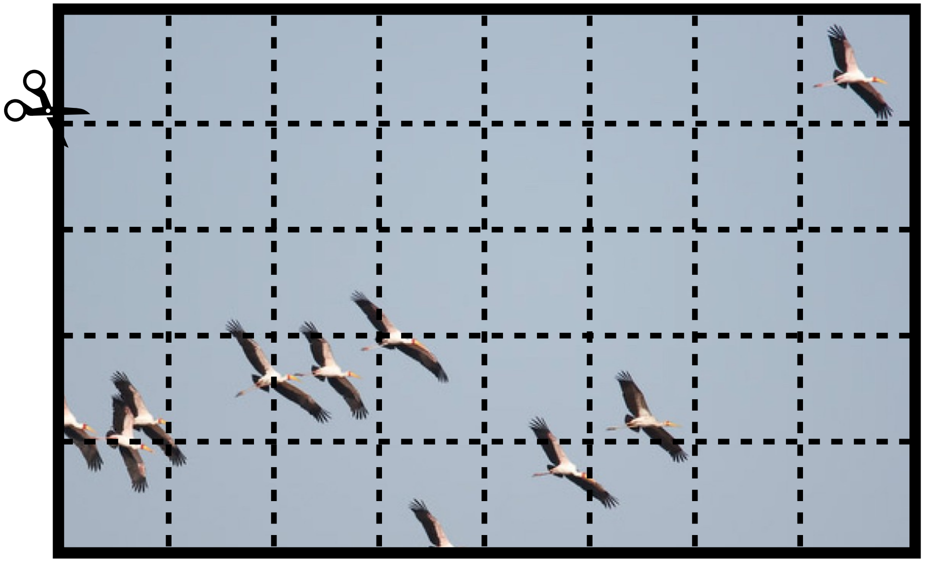
There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language:

- *Stanford University is located in______, California.* [Trivia]
- *I put___fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over______shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and______.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was___.* [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the_______________. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21,__ [some basic arithmetic; they don't learn the Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.

Adapted from John Hewitt

Vision Transformers

A limitation of CNNs

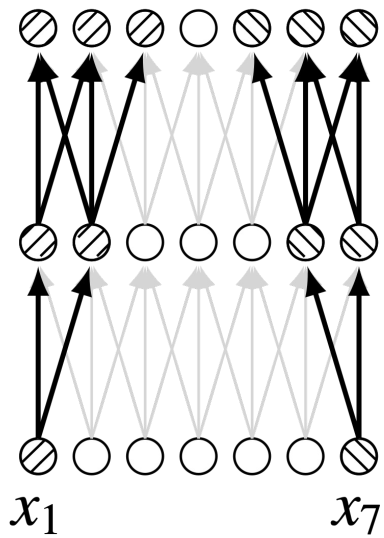


How many birds are in this image?

Is the top right bird the same species as the bottom left bird?

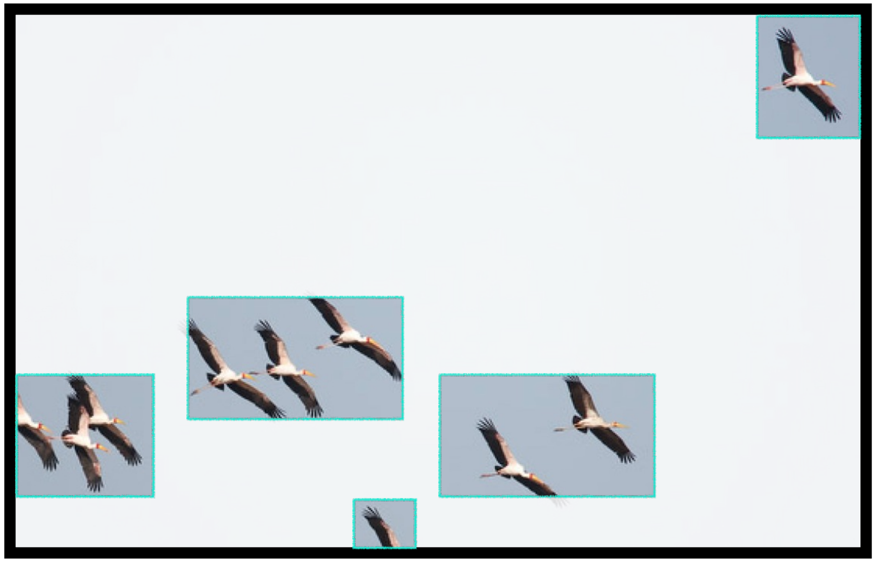
CNNs are built around the idea of locality, and are not well-suited to modeling long distance relationships

A limitation of CNNs



Far apart image patches do not interact

How Attention helps Computer Vision?



How many birds are in this image?

How Attention helps Computer Vision?

Is the top right bird the same species as the bottom left bird?

How Attention helps Computer Vision?

What's the color of the sky?

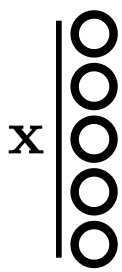
New Idea #1: Tokens

A New Data Type: Tokens

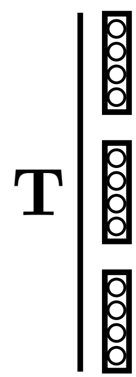
- A **token** is just a vector of neurons.

- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.

array of **neurons**



array of **tokens**

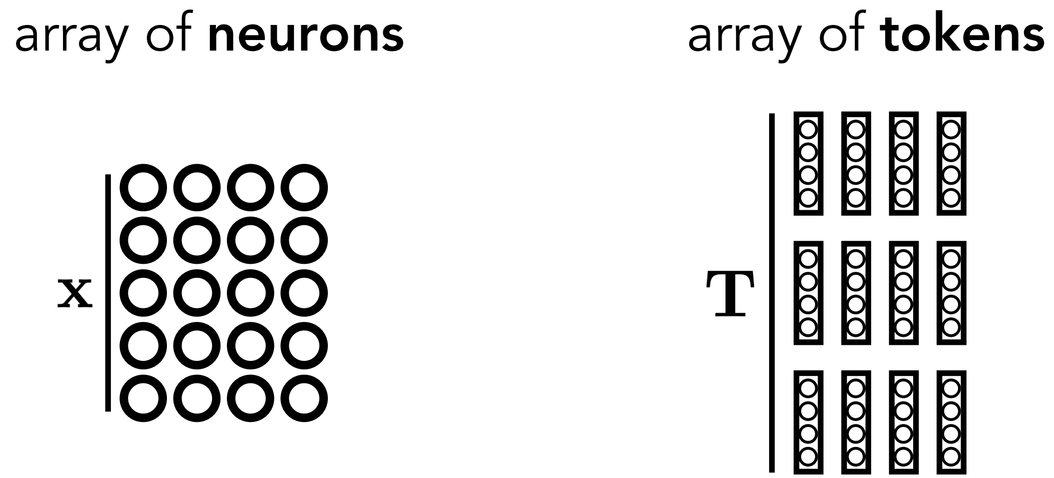


Note: sometimes the word "token" is instead used to refer to the atomic units of the data sequence we will model. In this usage tokens are the representation of the data only at the input and output layers. We use a more general definition where tokens are the representation of the data at *any* layer.

A new data structure: Tokens

- A **token** is just a vector of neurons.

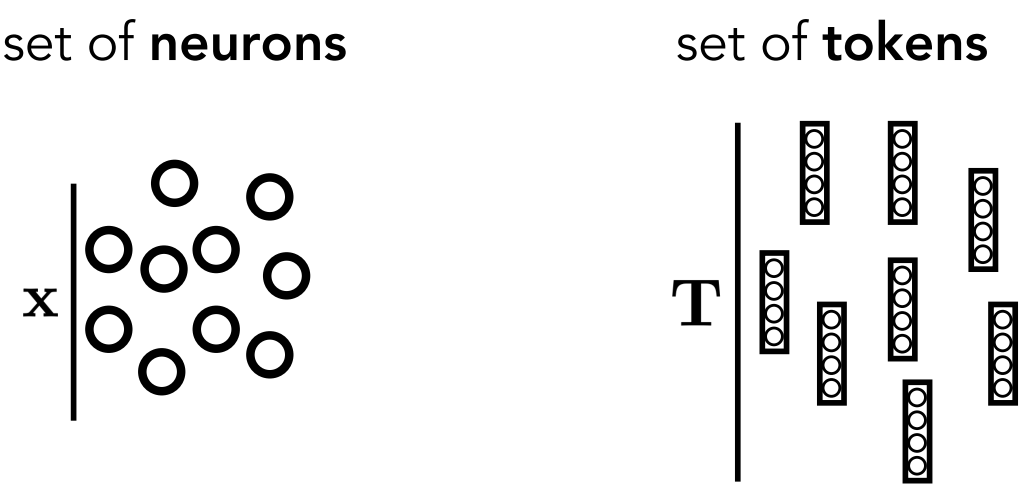
- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.



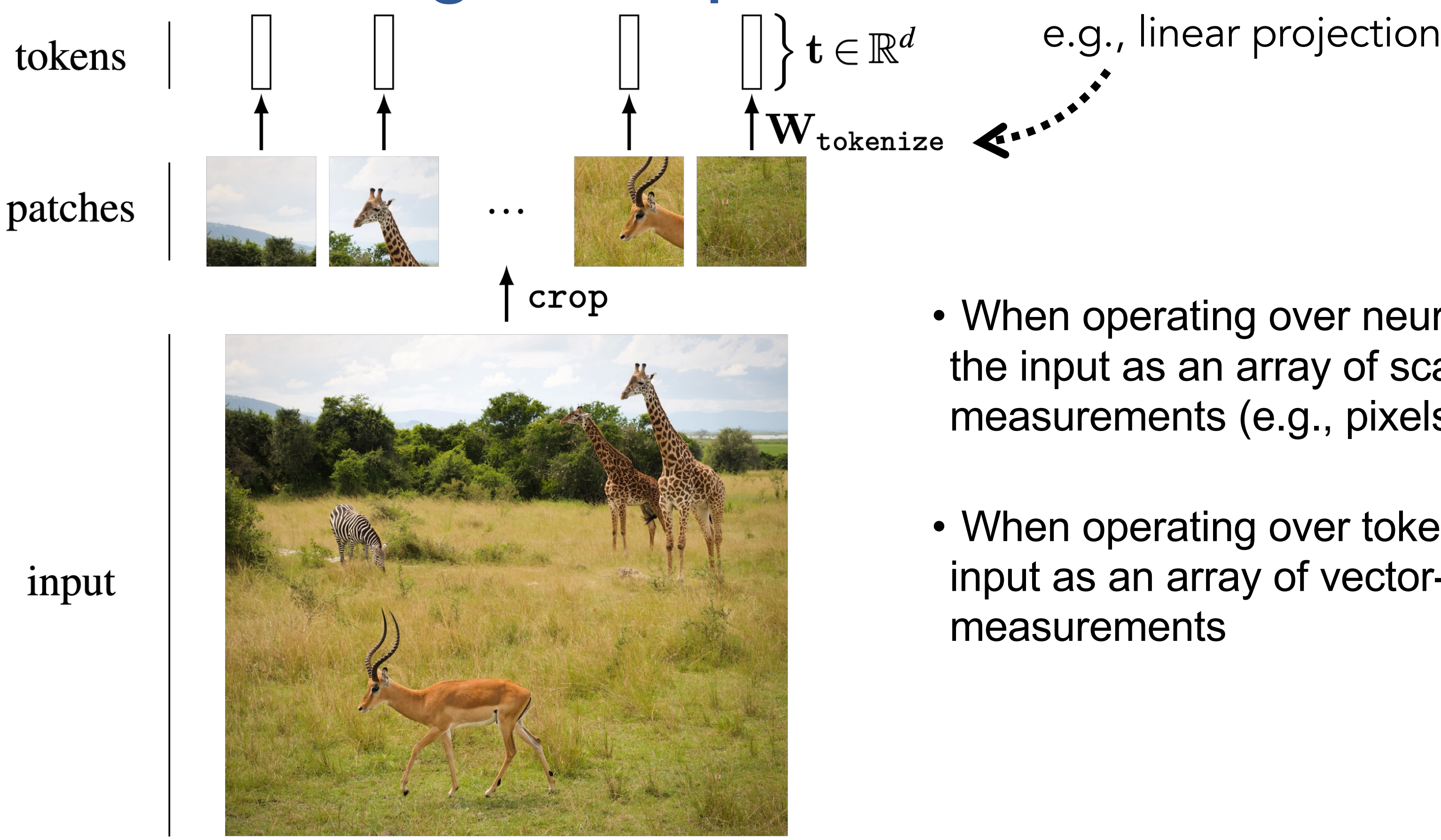
A new data structure: Tokens

- A **token** is just a vector of neurons.

- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.



Tokenizing the input data

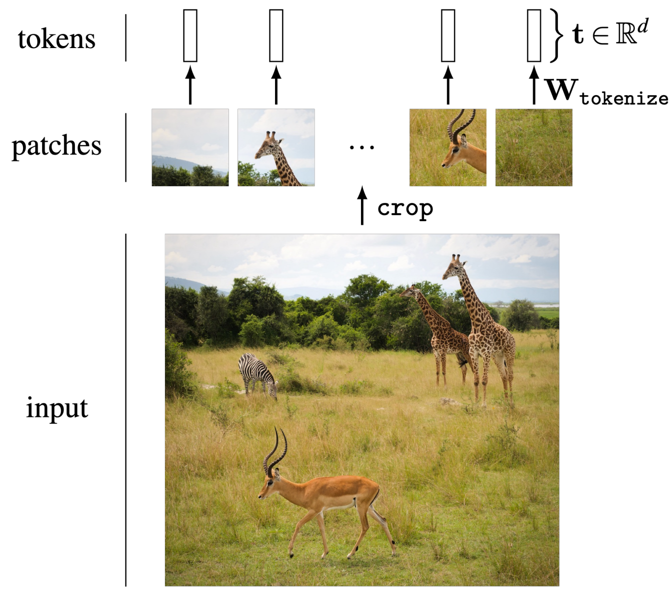


- When operating over neurons, we represent the input as an array of scalar-valued measurements (e.g., pixels)

- When operating over tokens, we represent the input as an array of vector-valued measurements

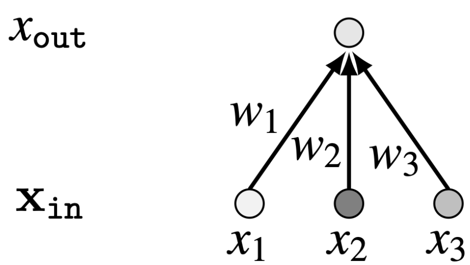
Tokenizing the input data

You can tokenize anything.
General strategy: chop the input up into chunks, project each chunk to a vector.



Linear combination of tokens

Linear combination of neurons

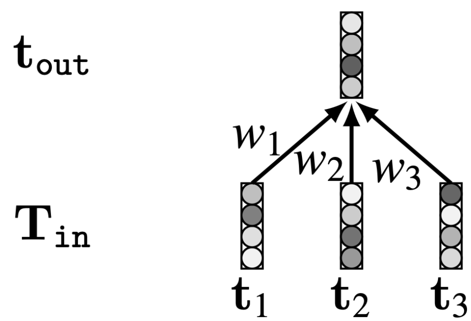


$$x_{\texttt{out}} = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$x_{\texttt{out}}[i] = \sum_{j=1}^{N} w_{ij} x_{\texttt{in}}[j]$$

$$\mathbf{x}_{\texttt{out}} = \mathbf{W}\mathbf{x}_{\texttt{in}}$$

Linear combination of tokens



$$\mathbf{t}_{\texttt{out}} = w_1 \mathbf{t}_1 + w_2 \mathbf{t}_2 + w_3 \mathbf{t}_3$$

$$\mathbf{T}_{\texttt{out}}[i,:] = \sum_{j=1}^{N} w_{ij} \mathbf{T}_{\texttt{in}}[j,:]$$

$$\mathbf{T}_{\texttt{out}} = \mathbf{W}\mathbf{T}_{\texttt{in}}$$

Token-wise nonlinearity

$$\mathbf{x}_{\texttt{out}} = \begin{bmatrix} \texttt{relu}(x_{\texttt{in}}[0]) \\ \vdots \\ \texttt{relu}(x_{\texttt{in}}[N-1]) \end{bmatrix}$$

$$\mathbf{T}_{\texttt{out}} = \begin{bmatrix} F_{\theta}(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_{\theta}(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix}$$

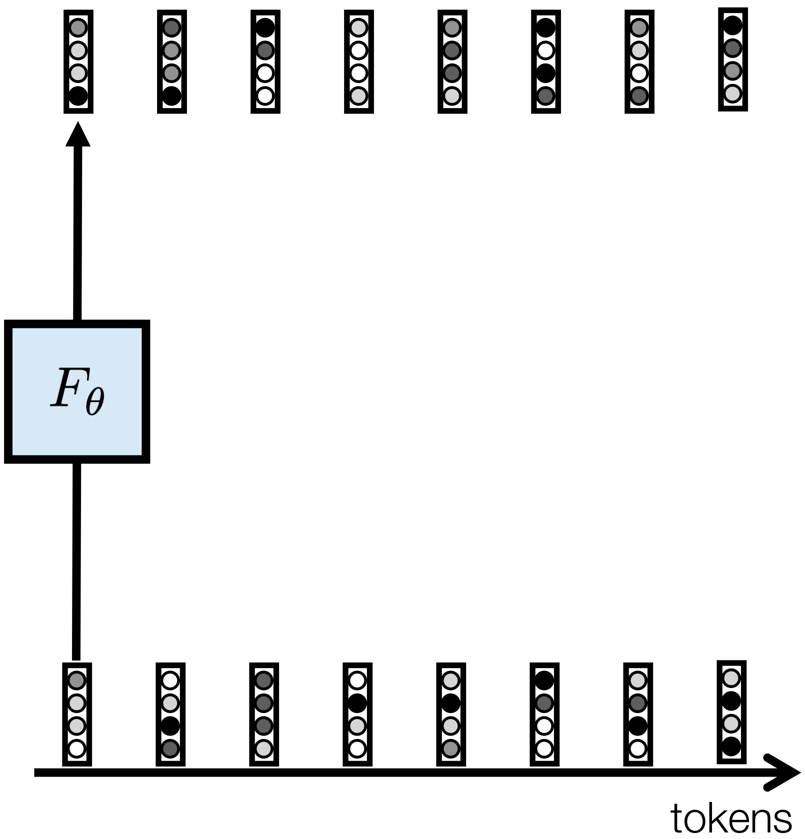
F is typically an MLP

Equivalent to a CNN with 1x1 kernels run over token sequence

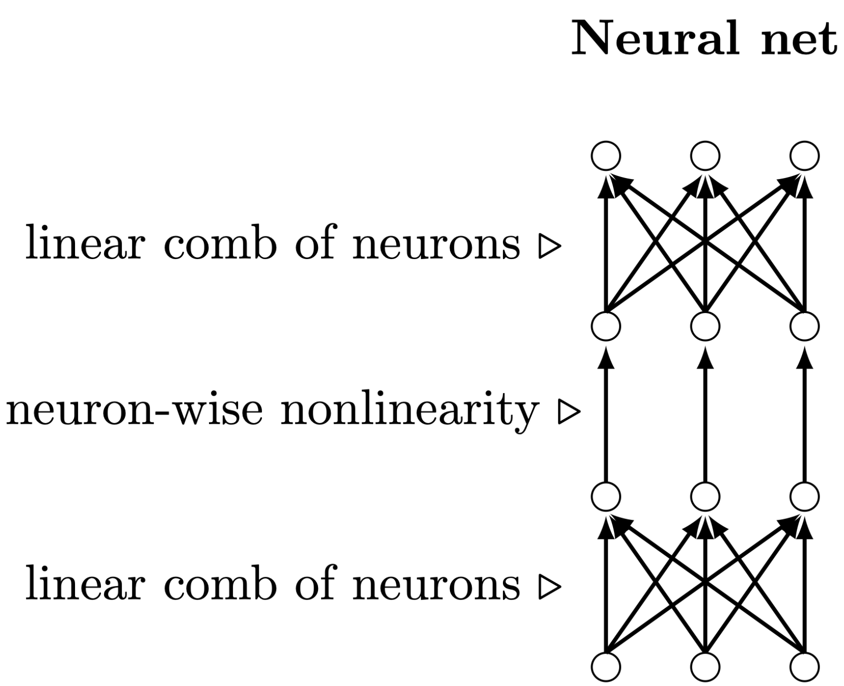
Token-wise nonlinearity

$$\mathbf{x}_{\texttt{out}} = \begin{bmatrix} \texttt{relu}(x_{\texttt{in}}[0]) \\ \vdots \\ \texttt{relu}(x_{\texttt{in}}[N-1]) \end{bmatrix}$$

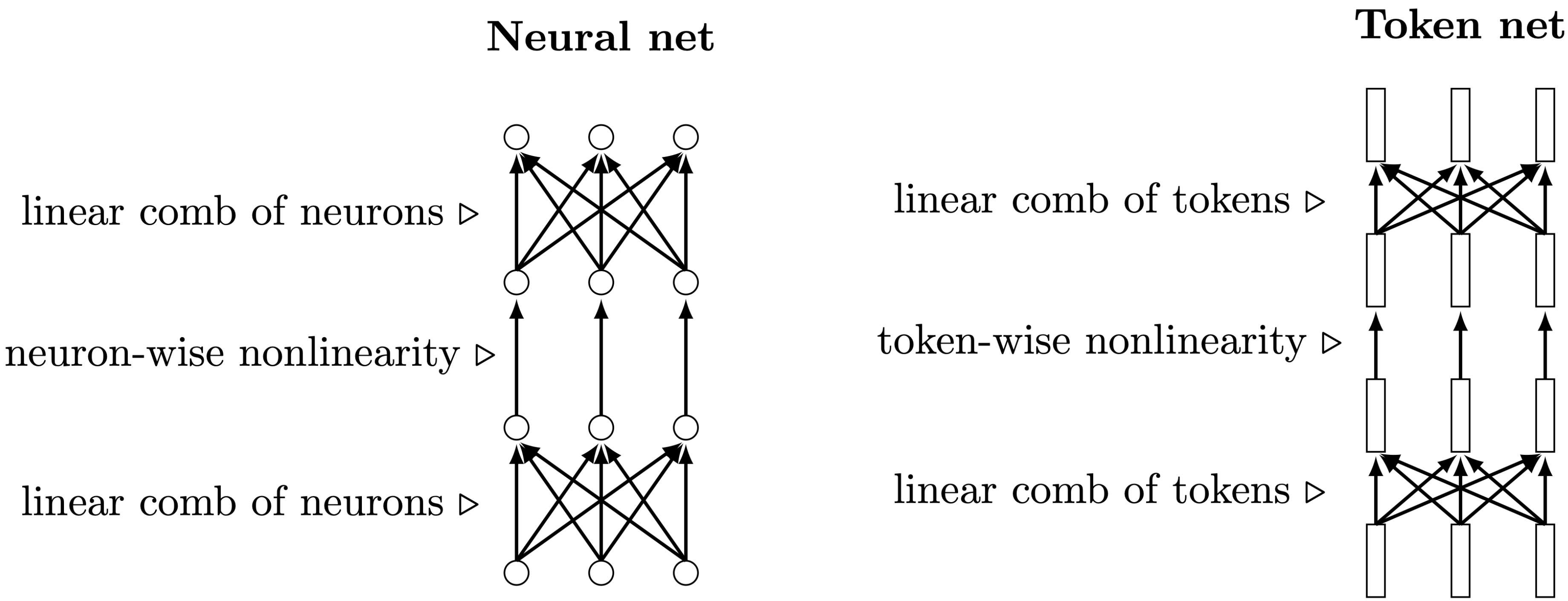
$$\mathbf{T}_{\texttt{out}} = \begin{bmatrix} F_{\theta}(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_{\theta}(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix}$$



Token nets


Neural net
linear comb of neurons ▷
neuron-wise nonlinearity ▷
linear comb of neurons ▷

Token nets

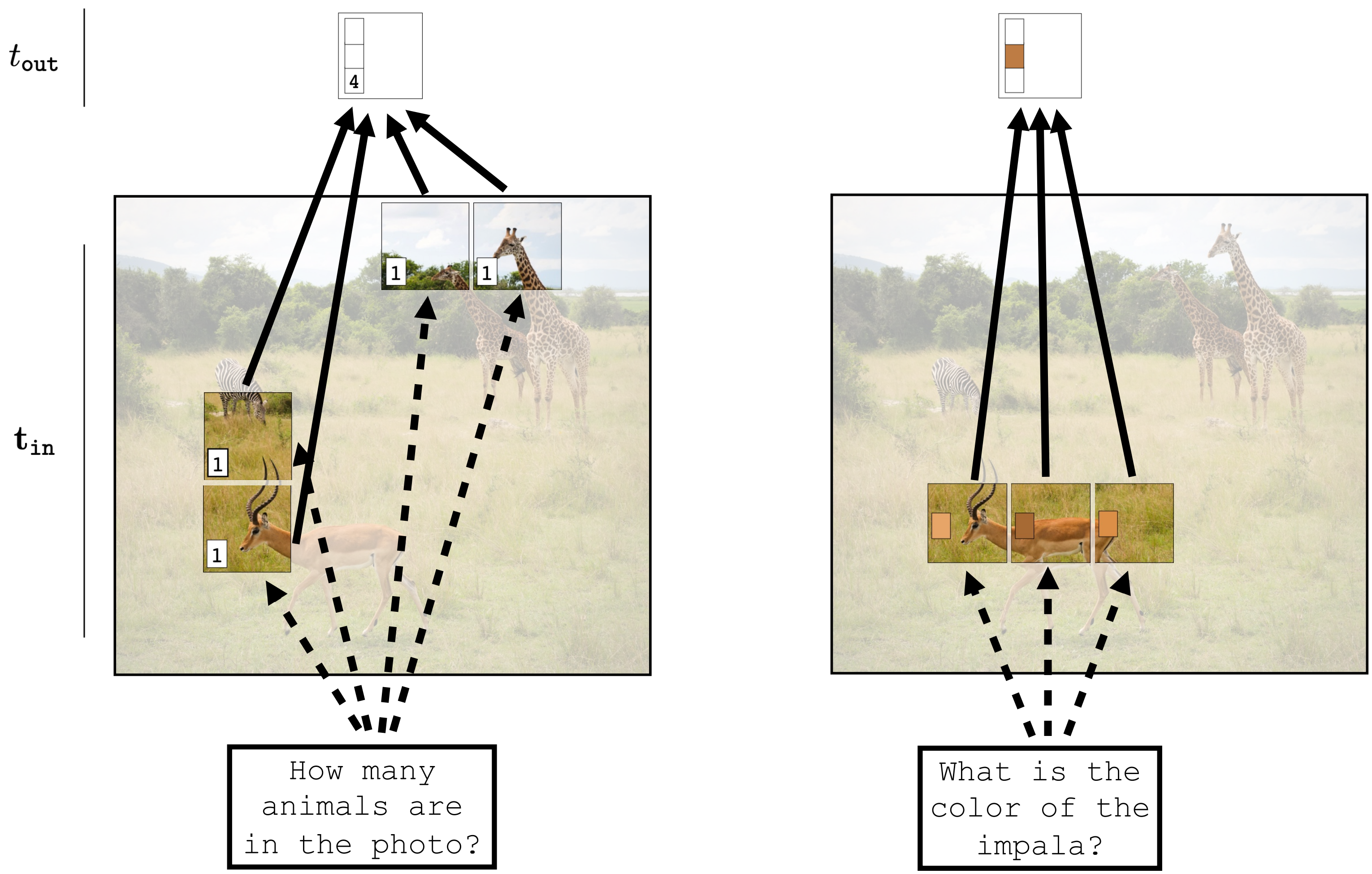

Neural net
linear comb of neurons ▷
neuron-wise nonlinearity ▷
linear comb of neurons ▷
Token net
linear comb of tokens ▷
token-wise nonlinearity ▷
linear comb of tokens ▷

New Idea #2: Attention

$t_{\texttt{out}}$

$\mathbf{t}_{\texttt{in}}$

How many animals are in the photo?

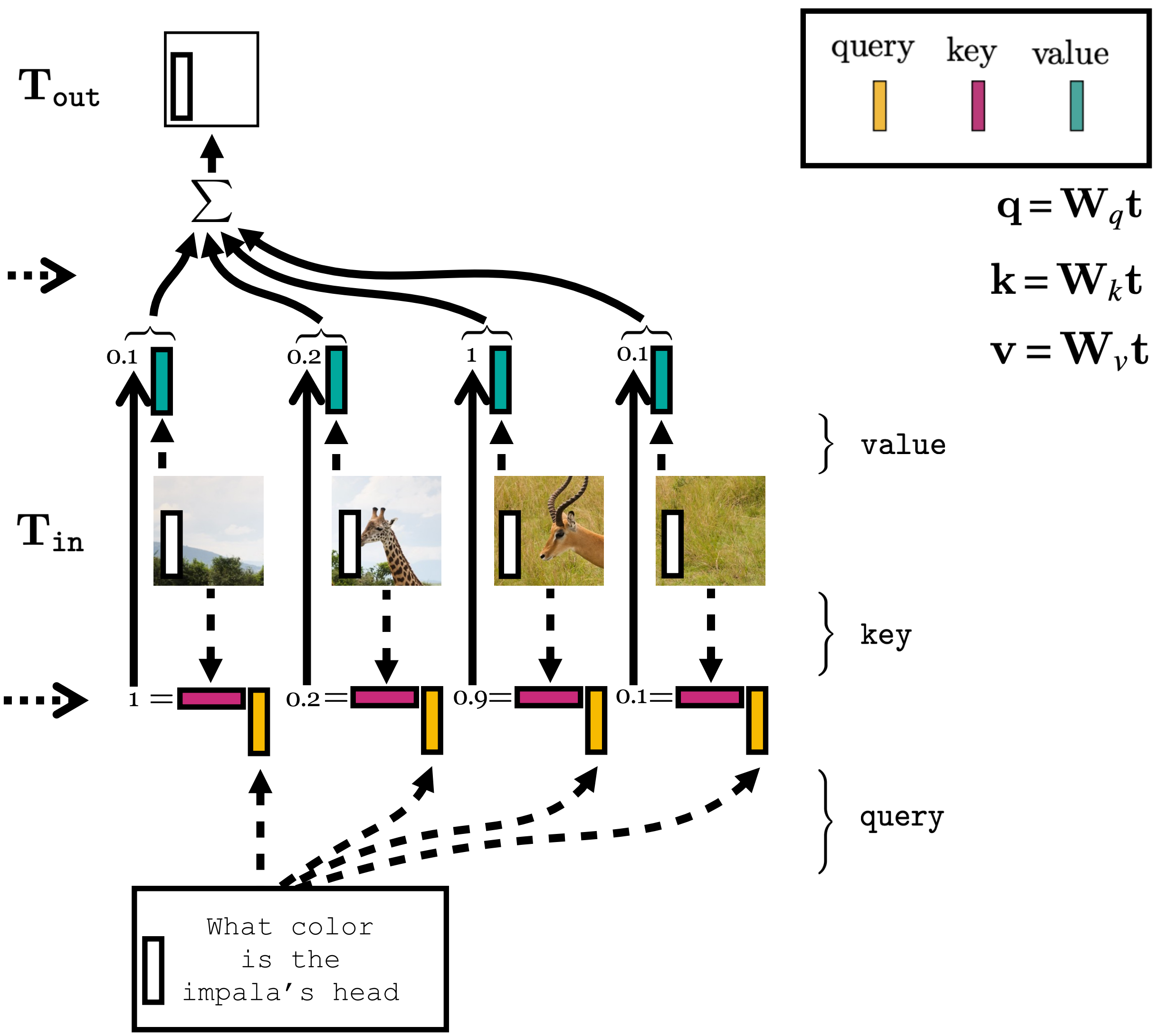

t_{out}
$\mathbf{t}_{\mathrm{in}}$
4
1
1
1
1
How many animals are in the photo?
What is the color of the impala?

query-key-value attention

$\mathbf{A} = \mathtt{softmax}(\mathbf{s})$

$$\mathbf{T}_{\mathtt{out}} = \begin{bmatrix} a_1 \mathbf{v}_1^\mathsf{T} \\ \vdots \\ a_N \mathbf{v}_N^\mathsf{T} \end{bmatrix}$$

$$\mathbf{s} = [\mathbf{q}_{\mathtt{question}}^T \mathbf{k}_1, \ldots, \mathbf{q}_{\mathtt{question}}^T \mathbf{k}_N]$$

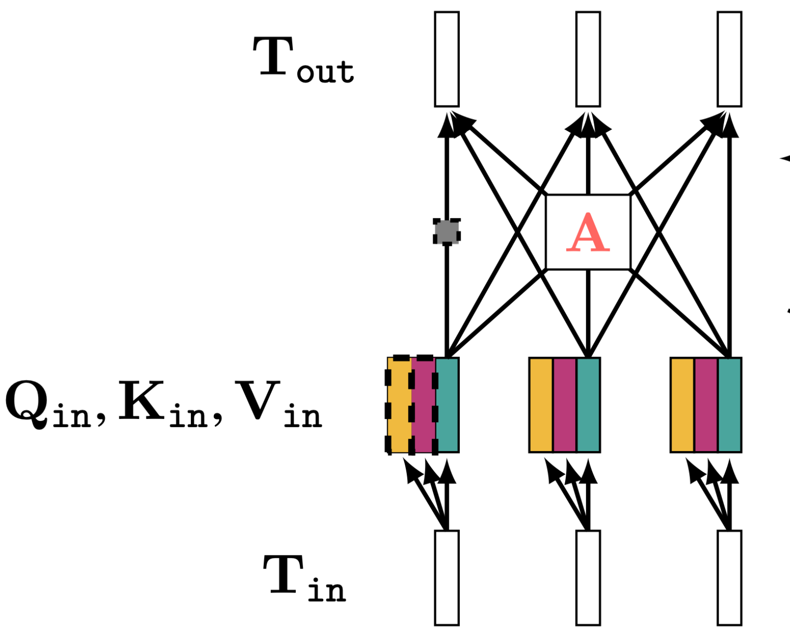


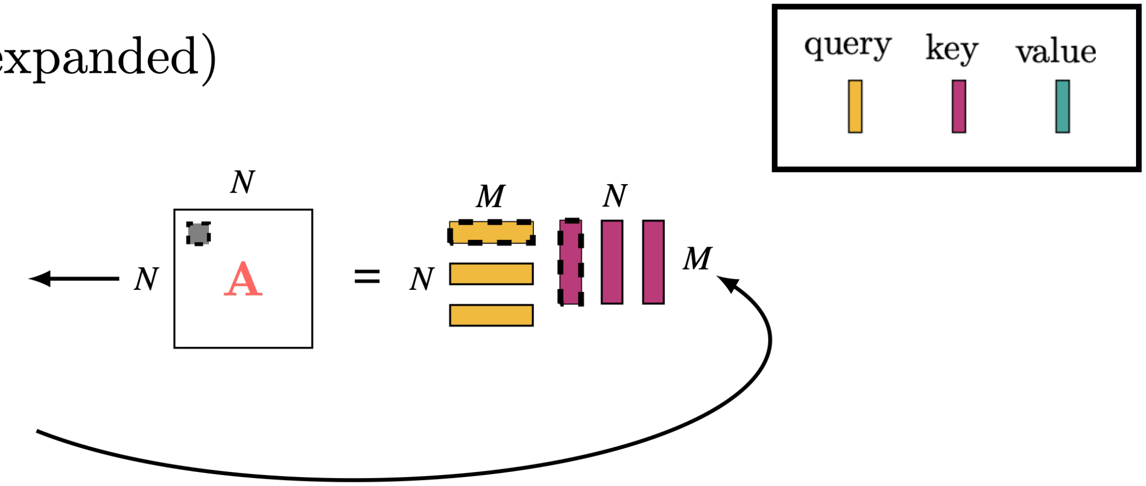
$\mathbf{q} = \mathbf{W}_q \mathbf{t}$

$\mathbf{k} = \mathbf{W}_k \mathbf{t}$

$\mathbf{v} = \mathbf{W}_v \mathbf{t}$

$\texttt{self attn}$ layer (expanded)





$$\mathbf{Q}_{\texttt{in}} = \begin{bmatrix} \mathbf{q}_1^\mathsf{T} \\ \vdots \\ \mathbf{q}_N^\mathsf{T} \end{bmatrix} = \begin{bmatrix} (\mathbf{W}_q \mathbf{t}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{W}_q \mathbf{t}_N)^\mathsf{T} \end{bmatrix} = \mathbf{T}_{\texttt{in}} \mathbf{W}_q^\mathsf{T} \qquad \triangleleft \quad \text{query matrix}$$

$$\mathbf{K}_{\texttt{in}} = \begin{bmatrix} \mathbf{k}_1^\mathsf{T} \\ \vdots \\ \mathbf{k}_N^\mathsf{T} \end{bmatrix} = \begin{bmatrix} (\mathbf{W}_k \mathbf{t}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{W}_k \mathbf{t}_N)^\mathsf{T} \end{bmatrix} = \mathbf{T}_{\texttt{in}} \mathbf{W}_k^\mathsf{T} \qquad \triangleleft \quad \text{key matrix}$$

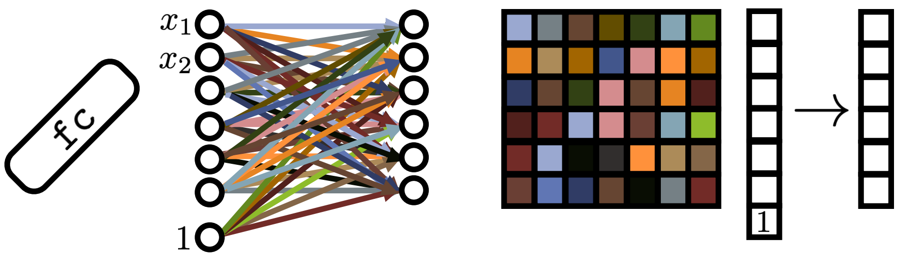
$$\mathbf{V}_{\texttt{in}} = \begin{bmatrix} \mathbf{v}_1^\mathsf{T} \\ \vdots \\ \mathbf{v}_N^\mathsf{T} \end{bmatrix} = \begin{bmatrix} (\mathbf{W}_v \mathbf{t}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{W}_v \mathbf{t}_N)^\mathsf{T} \end{bmatrix} = \mathbf{T}_{\texttt{in}} \mathbf{W}_v^\mathsf{T} \qquad \triangleleft \quad \text{value matrix}$$

$$\mathbf{A} = f(\mathbf{T}_{\texttt{in}}) = \texttt{softmax}\left(\frac{\mathbf{Q}_{\texttt{in}} \mathbf{K}_{\texttt{in}}^\mathsf{T}}{\sqrt{m}}\right) \qquad \triangleleft \quad \text{attention matrix}$$

$$\mathbf{T}_{\texttt{out}} = \mathbf{A}\mathbf{V}_{\texttt{in}}$$

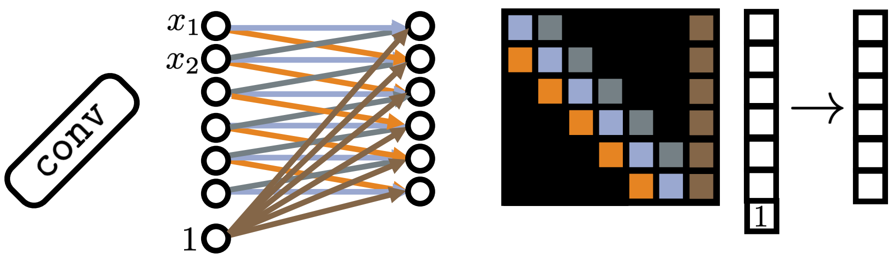
A family of linear layers

Wiring graph | Matrix | Properties



Fixed input dimensionality

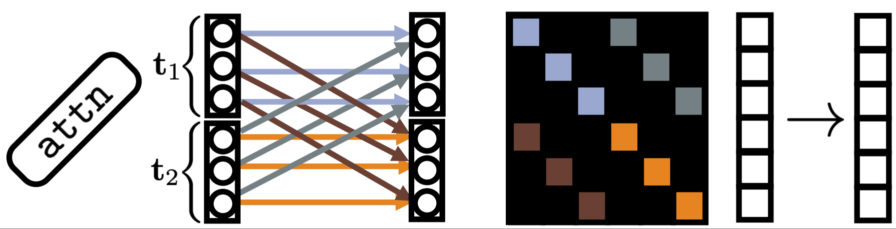
N^2 learnable parameters



Variable input dimensionality

k +1 learnable parameters (k= kernel size)

$\texttt{conv}(\texttt{translate}(\mathbf{x})) = \texttt{translate}(\texttt{conv}(\mathbf{x}))$

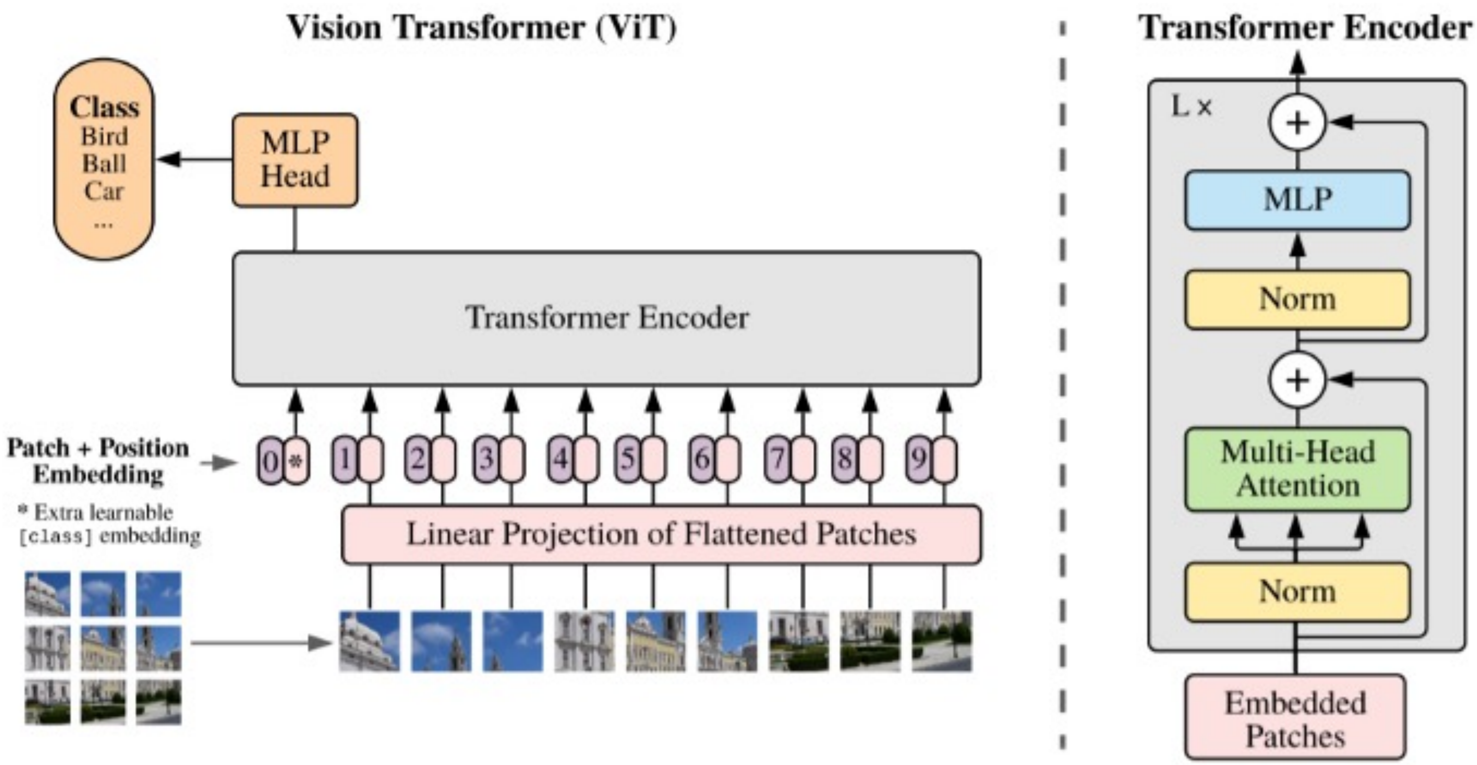


Variable input dimensionality

$|\mathbf{W}_q| + |\mathbf{W}_k| + |\mathbf{W}_v|$ learnable parameters

$\texttt{attn}(\texttt{permute}(\mathbf{T})) = \texttt{permute}(\texttt{attn}(\mathbf{T}))$

Transformers in vision



Dosovitskiy, ICLR 2021, https://github.com/google-research/vision_transformer https://www.youtube.com/watch?v=TrdevFK_am4

Cross-modal transformers

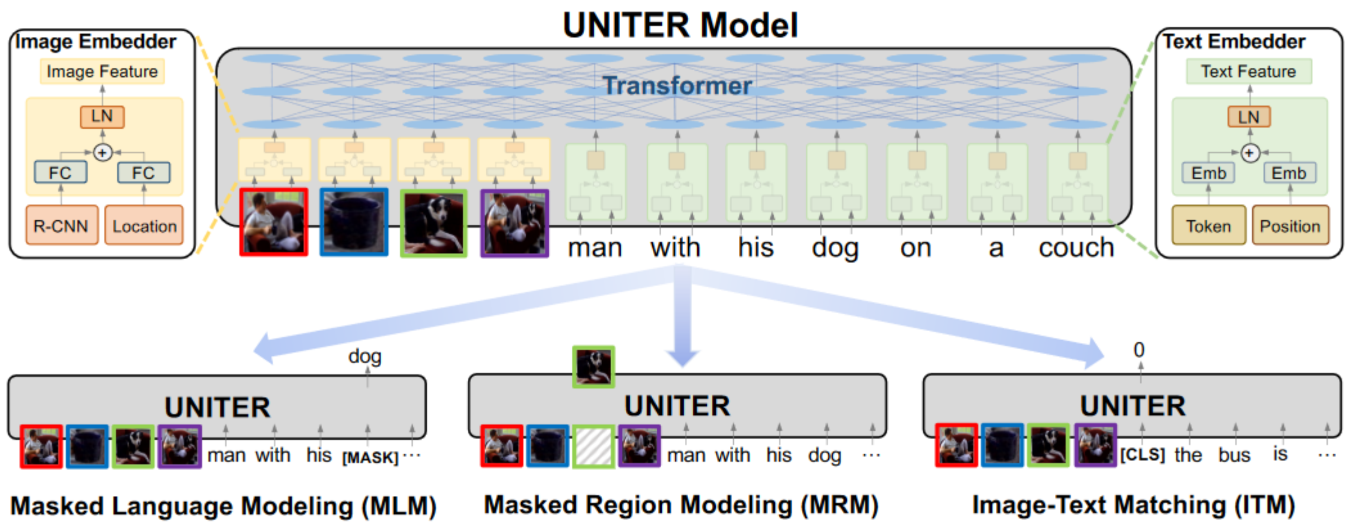
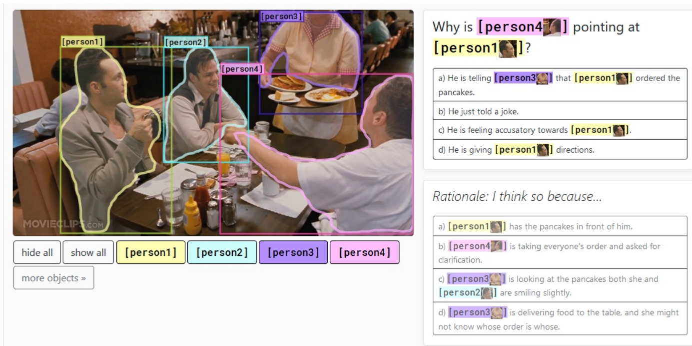


Figure 1: Overview of the proposed UNITER model (best viewed in color), consisting of an Image Embedder, a Text Embedder and a multi-layer self-attention Transformer, learned through three pre-training tasks.

Chen et al., "UNITER: Learning UNiversal Image-TExt Representations", arxiv 2019

Visual Commonsense Reasoning leaderboard



Rank	Model	Q->A	QA->R	Q->AR
	Human Performance *University of Washington* (Zellers et al. '18)	91.0	93.0	85.0
September 30, 2019	UNITER-large (ensemble) *MS D365 AI* https://arxiv.org/abs/1909.11740	**79.8**	**83.4**	**66.8**
2 September 23, 2019	UNITER-large (single model) *MS D365 AI* https://arxiv.org/abs/1909.11740	77.3	80.8	62.8
3 August 9,2019	ViLBERT (ensemble of 10 models) *Georgia Tech & Facebook AI Research* https://arxiv.org/abs/1908.02265	76.4	78.0	59.8
4 September 23,2019	VL-BERT (single model) *MSRA & USTC* https://arxiv.org/abs/1908.08530	75.8	78.4	59.7
5 August 9,2019	ViLBERT (ensemble of 5 models) *Georgia Tech & Facebook AI Research* https://arxiv.org/abs/1908.02265	75.7	77.5	58.8

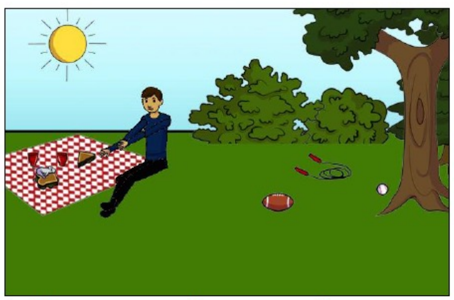
https://visualcommonsense.com/leaderboard/

Visual Question Answering (VQA)

Task: Given an image and a natural language open-ended question, generate a natural language answer.

What color are her eyes?
What is the mustache made of?

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

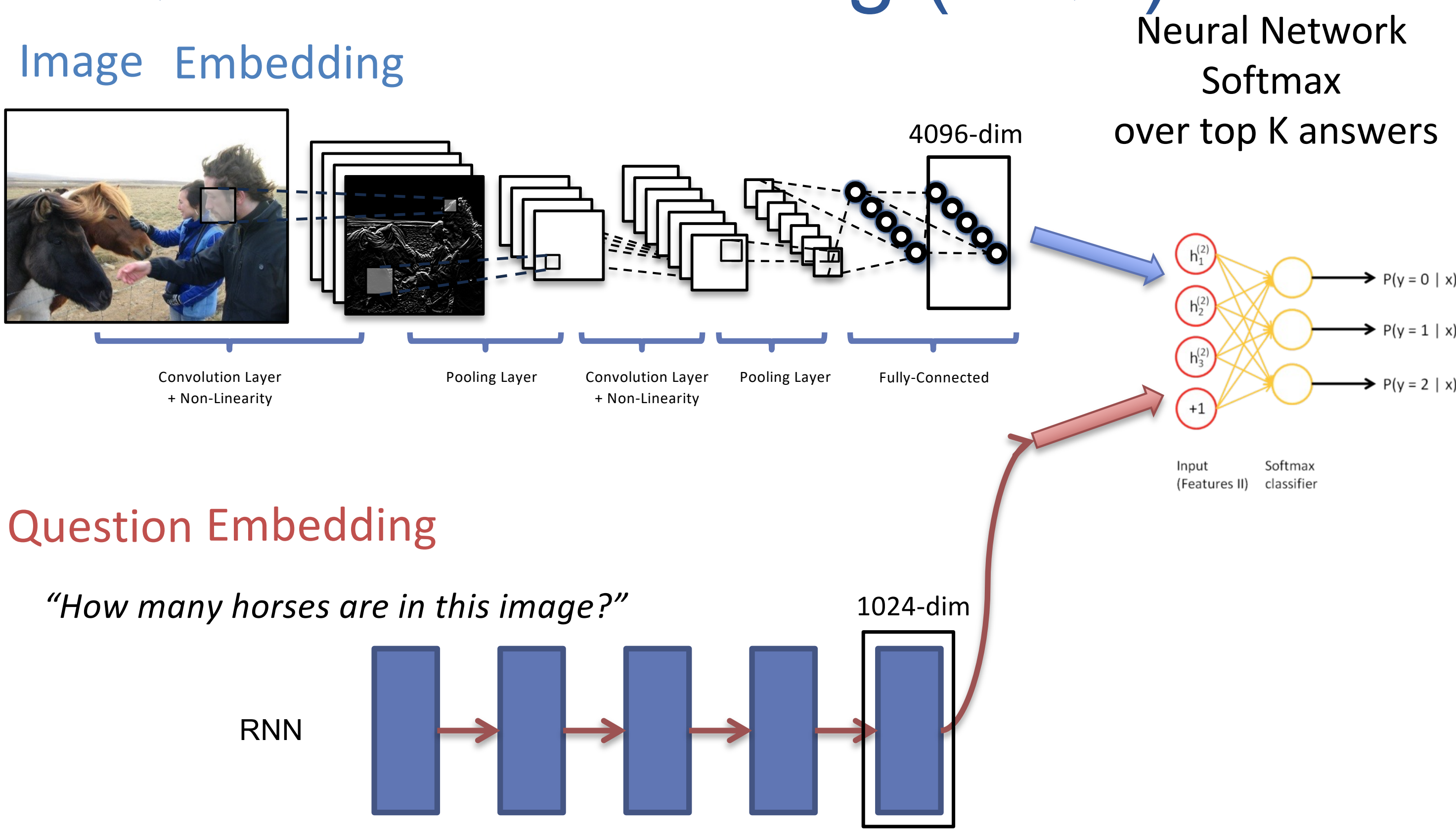


Is this person expecting company?
What is just under the tree?

Does it appear to be rainy?
Does this person have 20/20 vision?

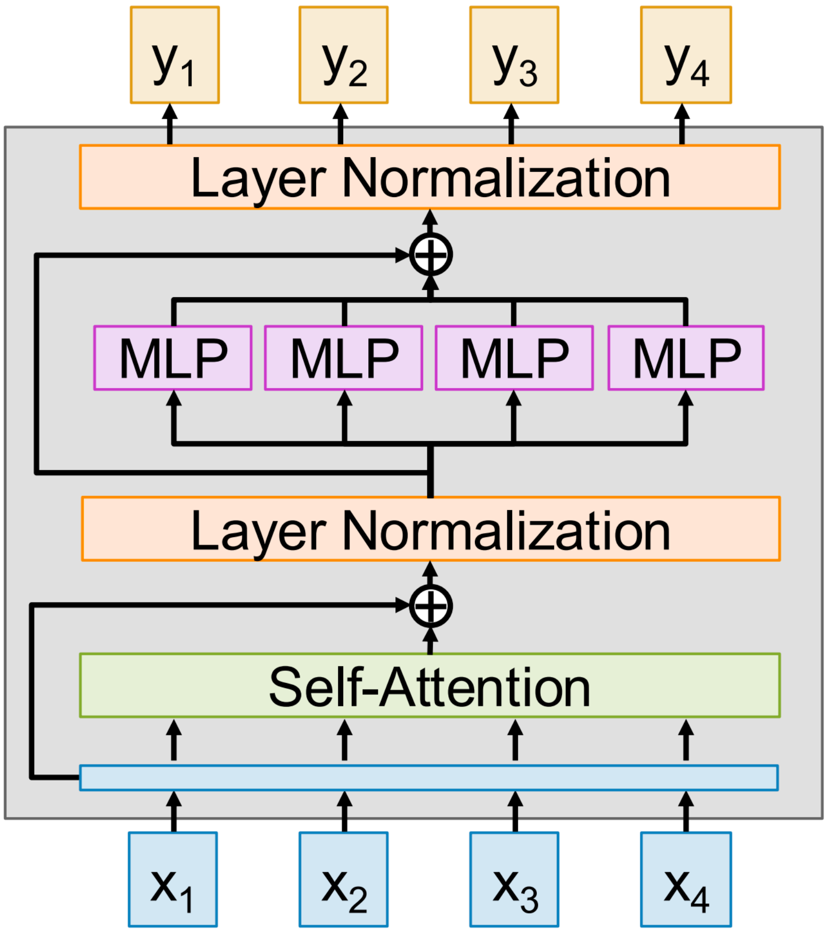
Agrawal et al., "VQA: Visual Question Answering", ICCV 2015

Visual Question Answering (VQA)



Agrawal et al., "VQA: Visual Question Answering", ICCV 2015

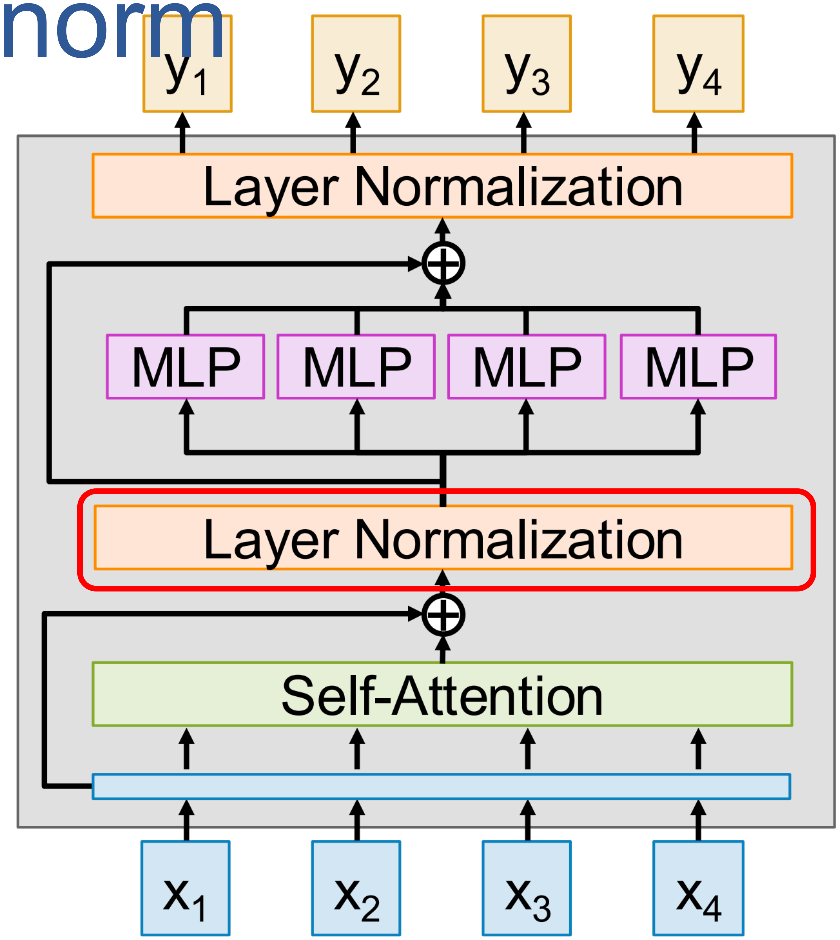
Tweaking Transformers



The Transformer architecture has not changed much since 2017.

But a few changes have become common:

Tweaking Transf. - Pre-norm

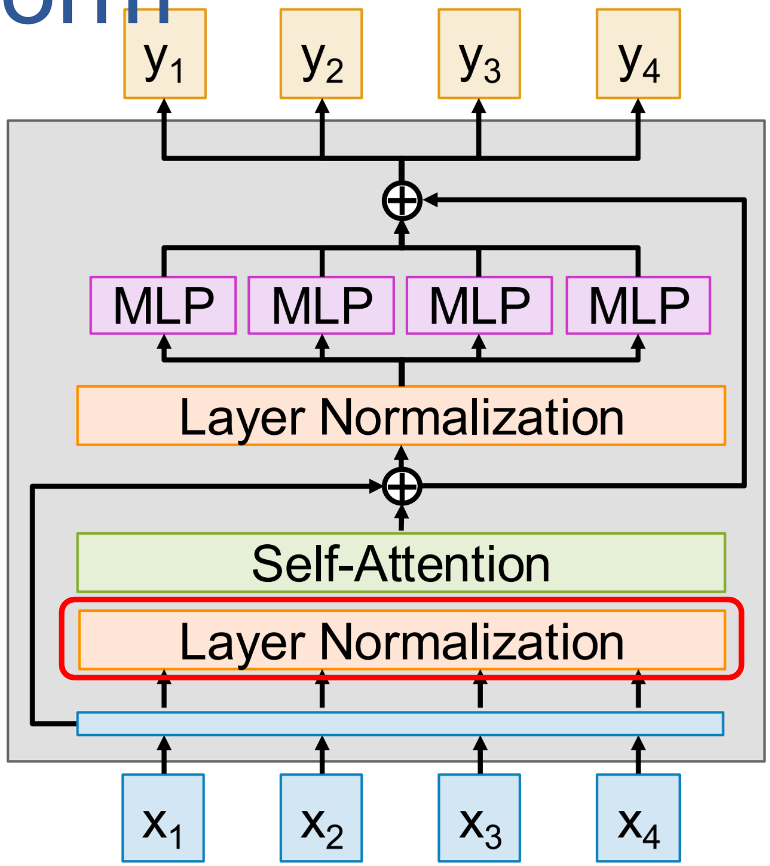


Layer normalization is outside the residual connections

Kind of weird, the model can't actually learn the identify function

Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

Tweaking Transf. - Pre-norm



Layer normalization is outside the residual connections

Kind of weird, the model can't actually learn the identify function

Solution: Move layer normalization before the Self-Attention and MLP, inside the residual connections. Training is more stable.

Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

Tweaking Transf. - RMSNorm

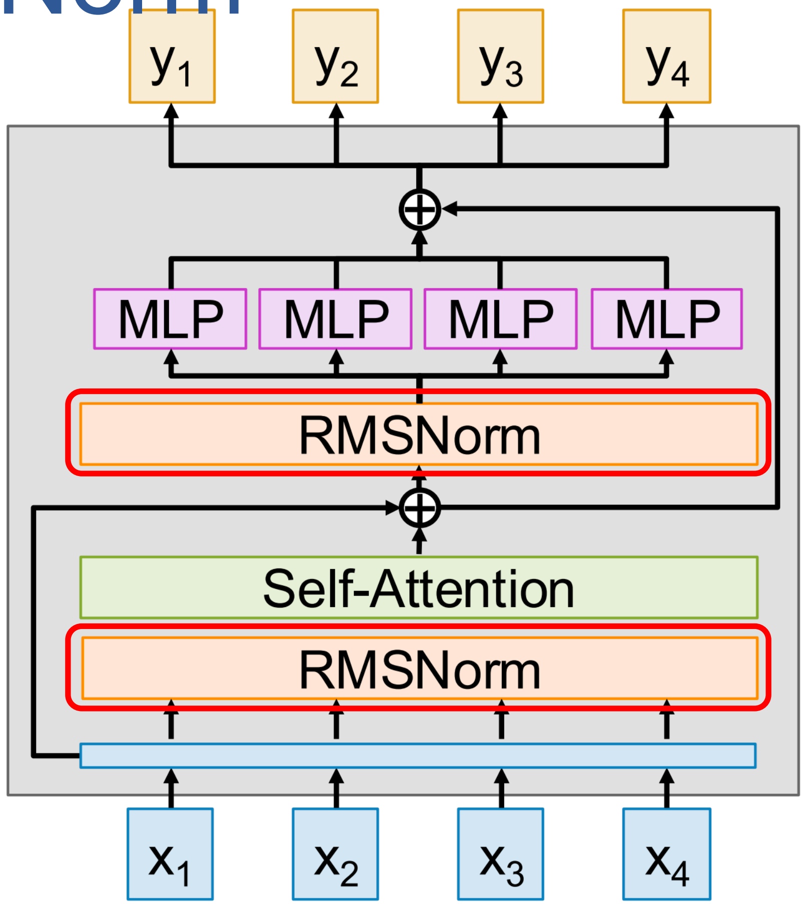
Replace Layer Normalization with Root-Mean-Square Normalization (RMSNorm)

Input: x [shape D]
Output: y [shape D]
Weight: γ [shape D]

$$y_i = \frac{x_i}{RMS(x)} * \gamma_i$$

$$RMS(x) = \sqrt{\varepsilon + \frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

Training is a bit more stable



Zhang and Sennrich, "Root Mean Square Layer Normalization", NeurIPS 2019

https://cs231n.stanford.edu/

Tweaking Transf. - SwiGLU MLP

Classic MLP:

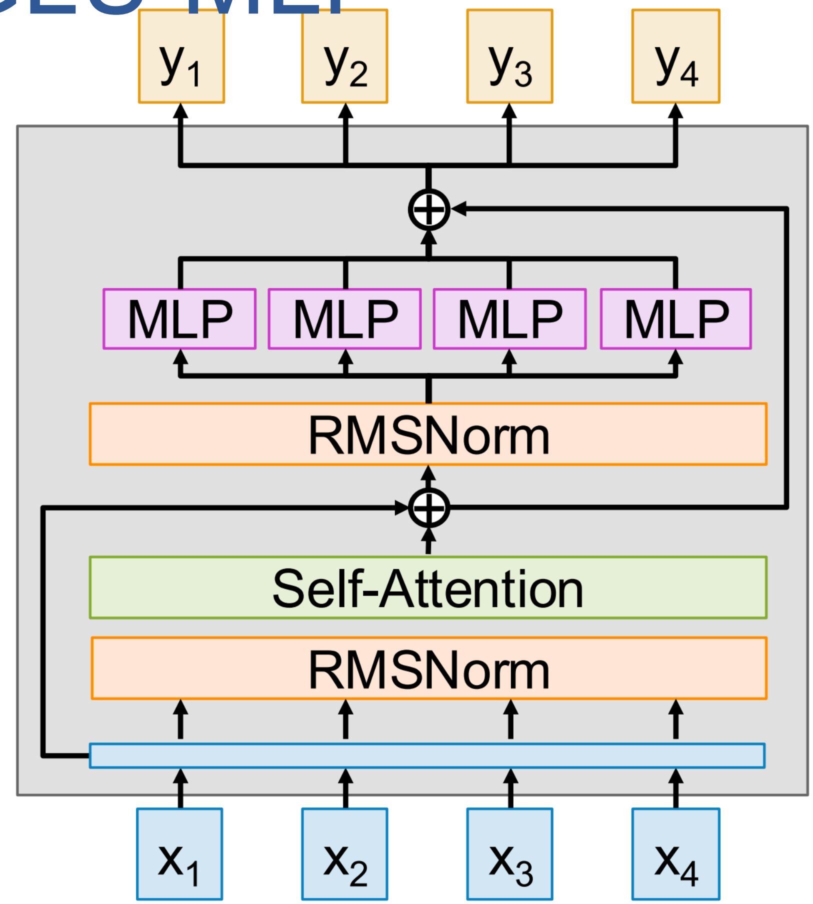
Input: X [N x D]
Weights: W_1 [D x 4D]
W_2 [4D x D]
Output: $Y = \sigma(XW_1)W_2$ [N x D]

SwiGLU MLP:

Input: X [N x D]
Weights: W_1 , W_2 [D x H]
W_3 [H x D]
Output:

$$Y = (\sigma(XW_1) \odot XW_2)W_3$$

Setting H = 8D/3 keeps same total params



Shazeer, "GLU Variants Improve Transformers", 2020

https://cs231n.stanford.edu/

Tweaking Transf. - Mixture of Experts (MoE)

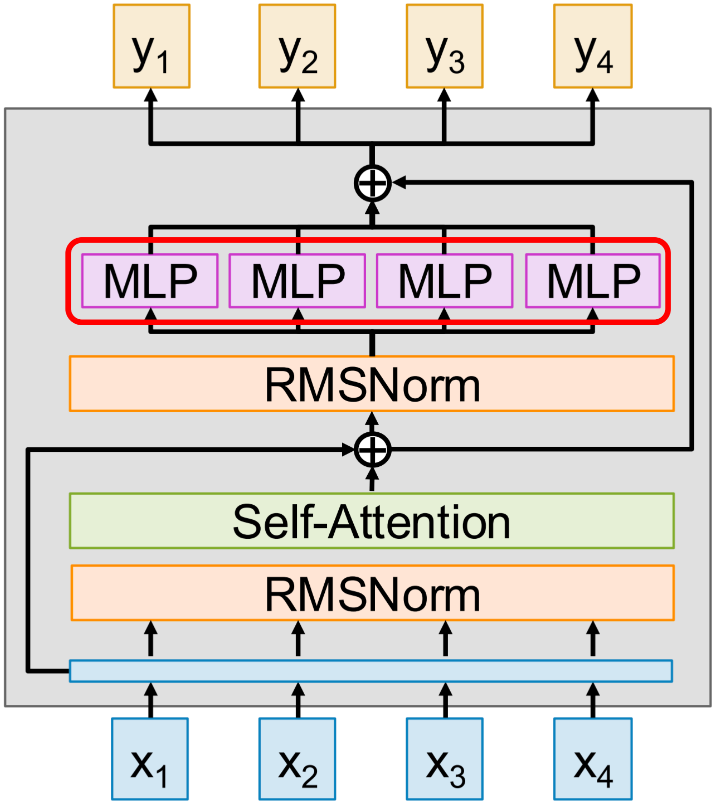
Learn E separate sets of MLP weights in each block; each MLP is an expert

W1: [D x 4D] => [E x D x 4D]
W2: [4D x D] => [E x 4D x D]

Each token gets routed to A < E of the experts. These are the active experts.

Increases params by E, But only increases compute by A

All of the biggest LLMs today (e.g. GPT4o, GPT4.5, Claude 3.7, Gemini 2.5 Pro, etc) almost certainly use MoE and have > 1T params; but they don't publish details anymore.



Shazeer et al, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", 2017

OMET Midterm Surveys

CS 2770 - COMPUTER VISION - 1000 - Lecture

Students

https://go.blueja.io/93Ika0Dq-USIBoW7Y8xYKQ

To access the evaluation, scan this QR code with your mobile phone.

CS2770
- https://go.blueja.io/93Ika0Dq-USIBoW7Y8xYKQ

ISSP 2180
- https://go.blueja.io/IWqC-UnAuE-R-yZqUdcxoA

ISSP 2180 - COMPUTER VISION - 1000 - Lecture

Students

https://go.blueja.io/IWqC-UnAuE-R-yZqUdcxoA

To access the evaluation, scan this QR code with your mobile phone.

Extra

Some pre-RNN Good Results

This is a picture of one sky, one road and one sheep. The gray sky is over the gray road. The gray sheep is by the gray road.

This is a picture of two dogs. The first dog is near the second furry dog.

Here we see one road, one sky and one bicycle. The road is near the blue sky, and near the colorful bicycle. The colorful bicycle is within the blue sky.

Kulkarni et al., CVPR 2011

Some pre-RNN Good Results

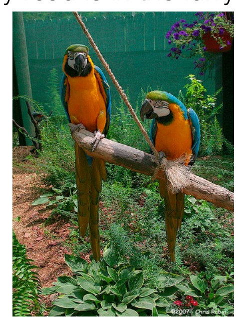
Missed detections:

Here we see one potted plant.

This is a picture of one dog.

False detections:

There are one road and one cat. The furry road is in the furry cat.



This is a picture of one tree, one road and one person. The rusty tree is under the red road. The colorful person is near the rusty tree, and under the red road.

Incorrect attributes:

This is a photograph of two sheeps and one grass. The first black sheep is by the green grass, and by the second black sheep. The second black sheep is by the green

This is a photograph of two horses and one grass. The first feathered horse is within the green grass, and by the second feathered horse. The second feathered horse is within the green grass.

Kulkarni et al., CVPR 2011

Extensions

- Vanishing gradient problem makes it hard to model long sequences
  - Multiplying together many values between 0 and 1 (range of gradient of sigmoid, tanh)

- One solution: Use RELU

- Another solution: Use RNNs with gates
  - Adaptively decide how much of memory to keep
  - Gated Recurrent Units (GRUs), Long Short Term Memories (LSTMs)

Generating poetry with RNNS

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
        Admit impediments. Love is not love
Which alters when it alteration finds,
        Or bends with the remover to remove:
O no! it is an ever-fixed mark
        That looks on tempests and is never shaken;
It is the star to every wandering bark,
        Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
        Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
        But bears it out even to the edge of doom.
If this be error and upon me proved,
        I never writ, nor no man ever loved.

Generating poetry with RNNS

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

More info: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Andrej Karpathy

Generating poetry with RNNS

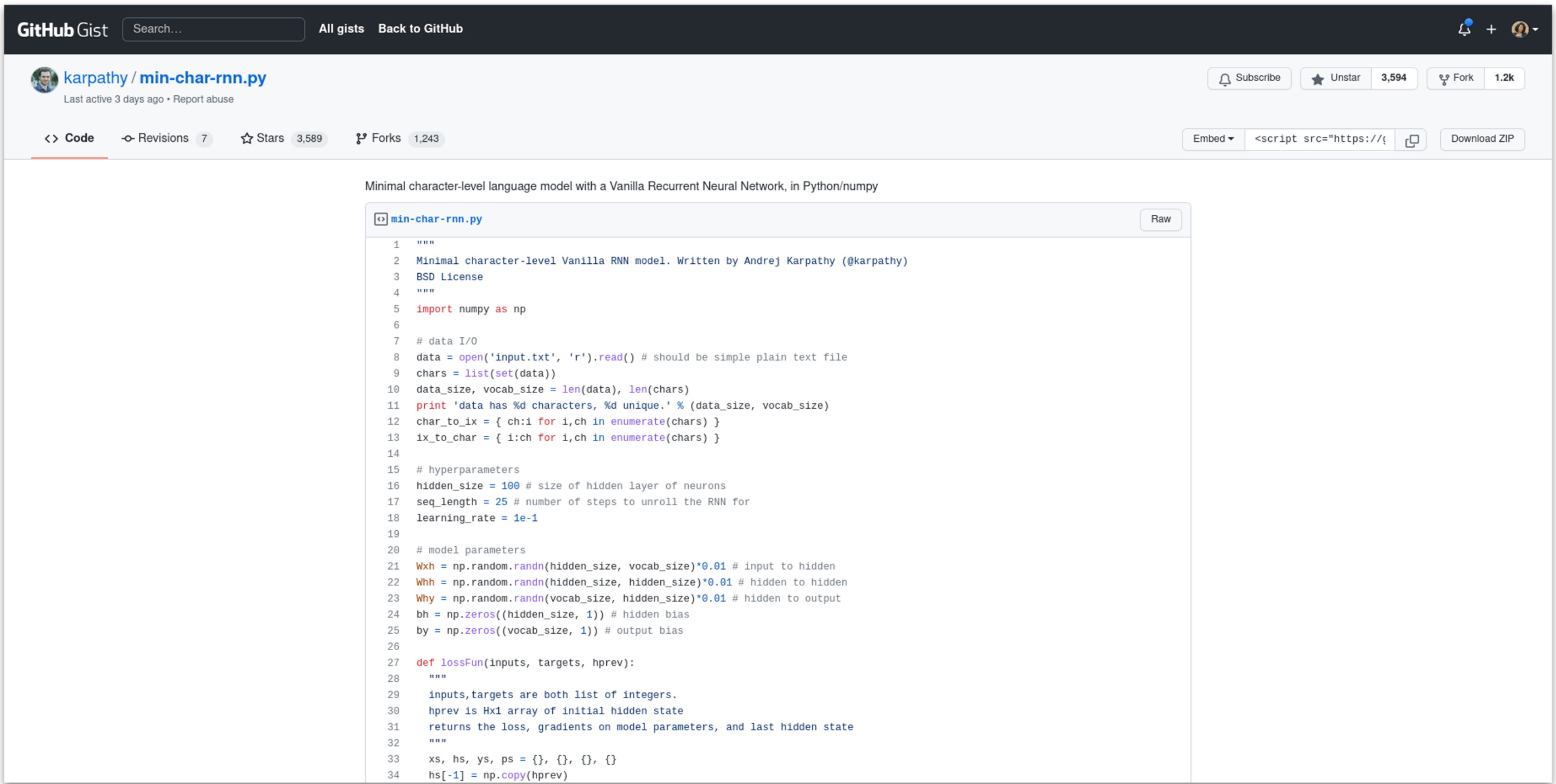
```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Andrej Karpathy

https://gist.github.com/karpathy/d4dee566867f8291f086 (Andrej Karpathy)

RNN Vanilla: 112 lines of Python



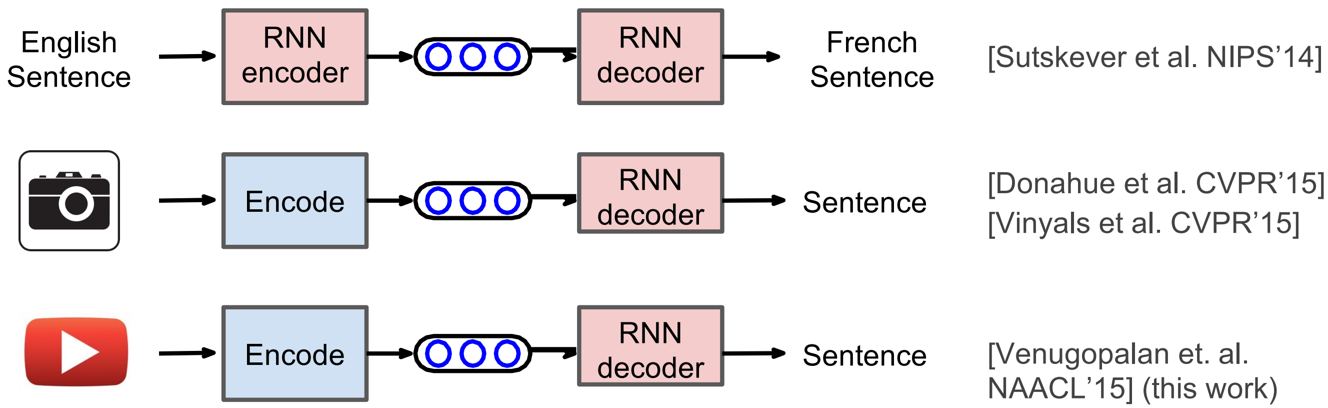
Video Captioning

Generate descriptions for events depicted in video clips

A monkey pulls a dog's tail and is chased by the dog.

Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

Video Captioning

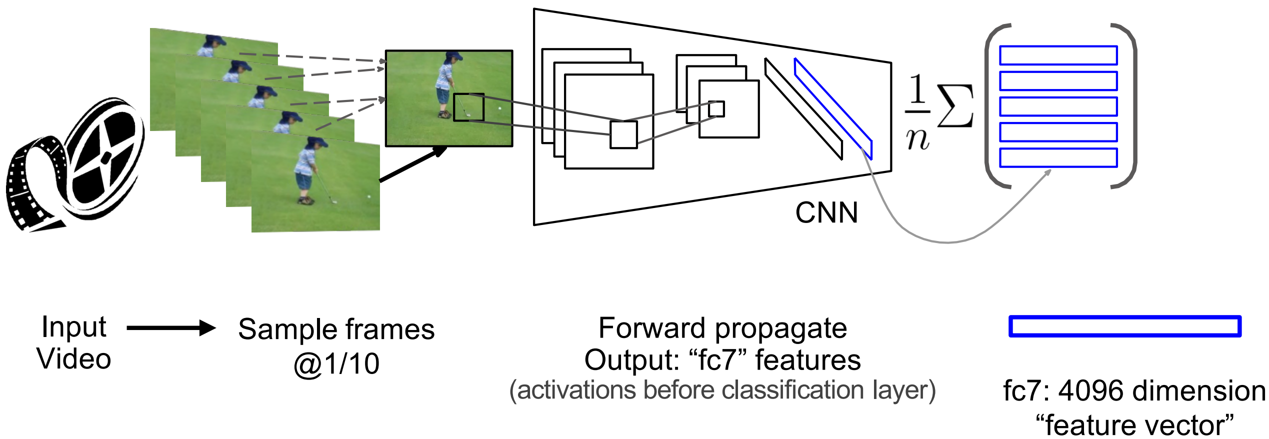


Key Insight:

Generate feature representation of the video and "decode" it to a sentence

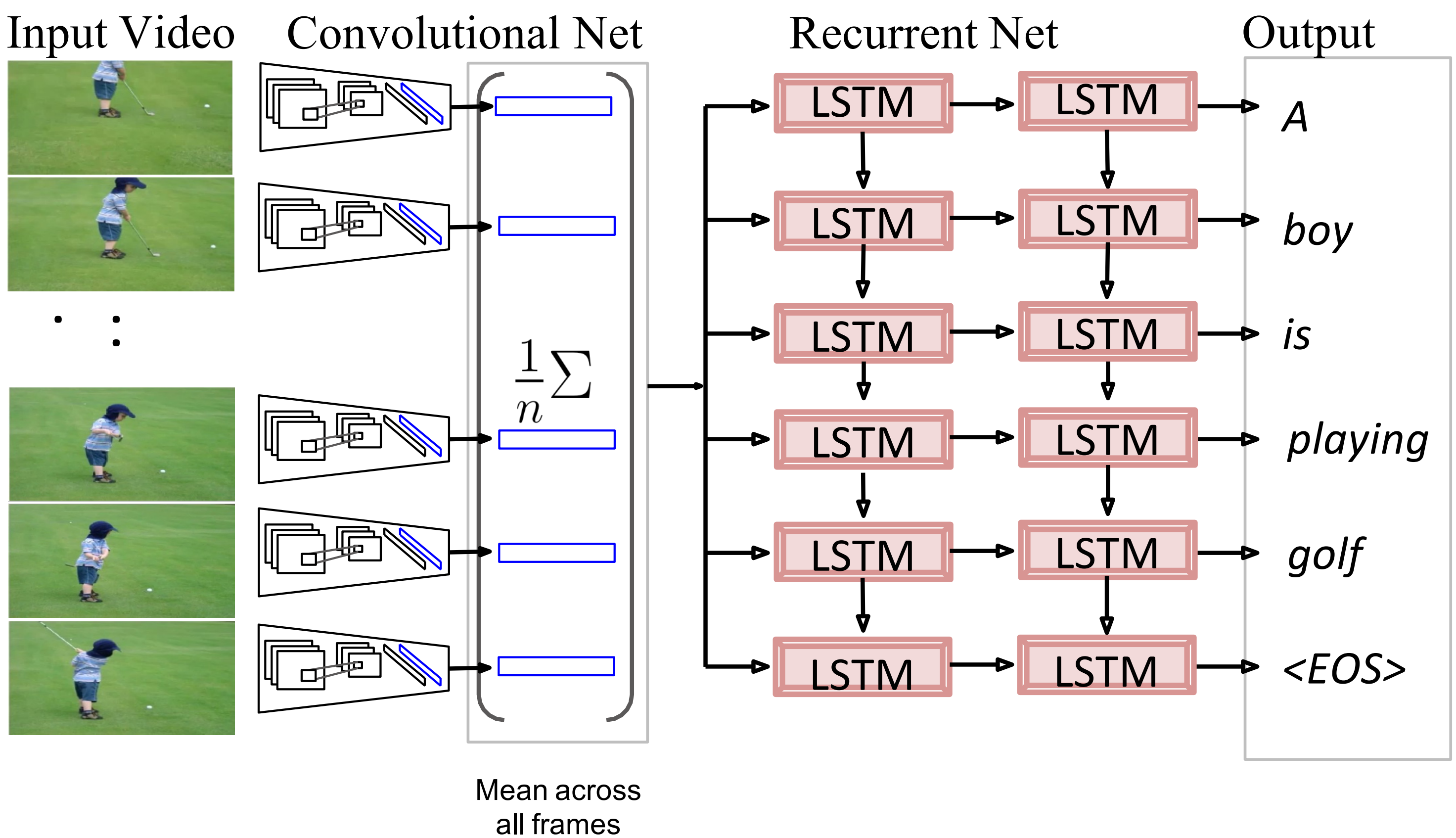
Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

Video Captioning



Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

Video Captioning

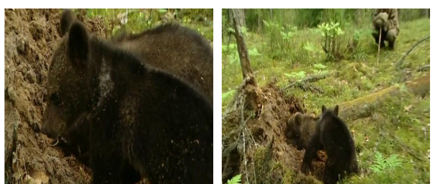


Venugopalan et al., “Translating Videos to Natural Language using Deep Recurrent Neural Networks”, NAACL-HTL 2015

Video Captioning

FGM: A person is dancing with the person on the stage.
YT: A group of men are riding the forest.
I+V: **A group of people are dancing.**
GT: Many men and women are dancing in the street.

FGM: A person is cutting a potato in the kitchen.
YT: A man is slicing a tomato.
I+V: **A man is slicing a carrot.**
GT: A man is slicing carrots.

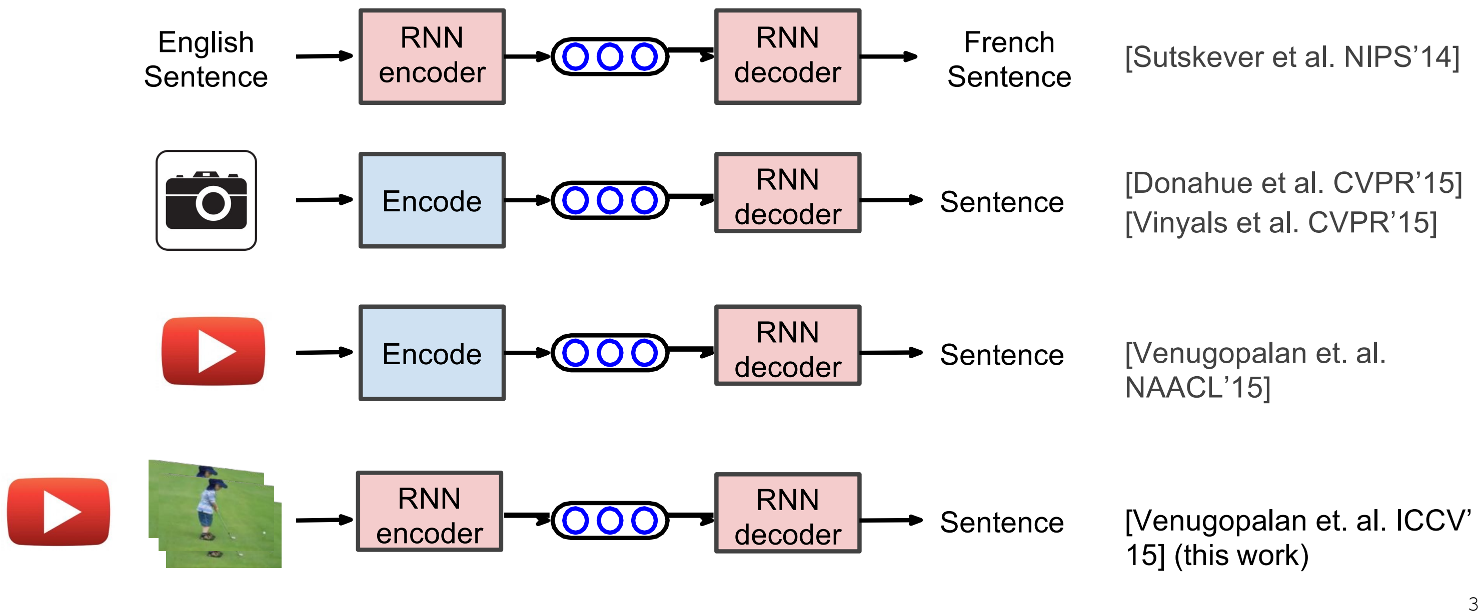


FGM: A person is walking with a person in the forest.
YT: A monkey is walking.
I+V: **A bear is eating a tree.**
GT: Two bear cubs are digging into dirt and plant matter at the base of a tree.

FGM: A person is riding a horse on the stage.
YT: A group of playing are playing in the ball.
I+V: **A basketball player is playing**.
GT: Dwayne wade does a fancy layup in an allstar game.

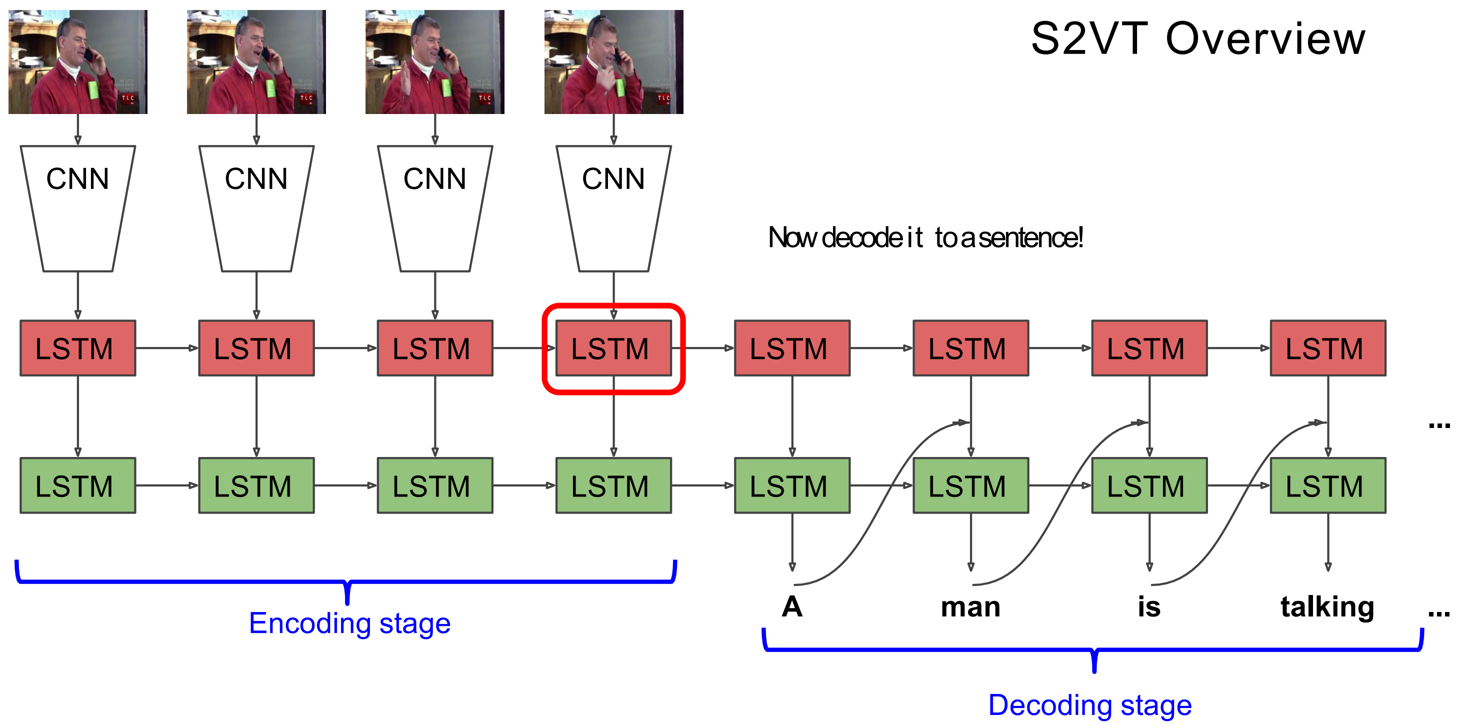
Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

Video Captioning



Venugopalan et al., "Sequence to Sequence - Video to Text", ICCV 2015

Video Captioning

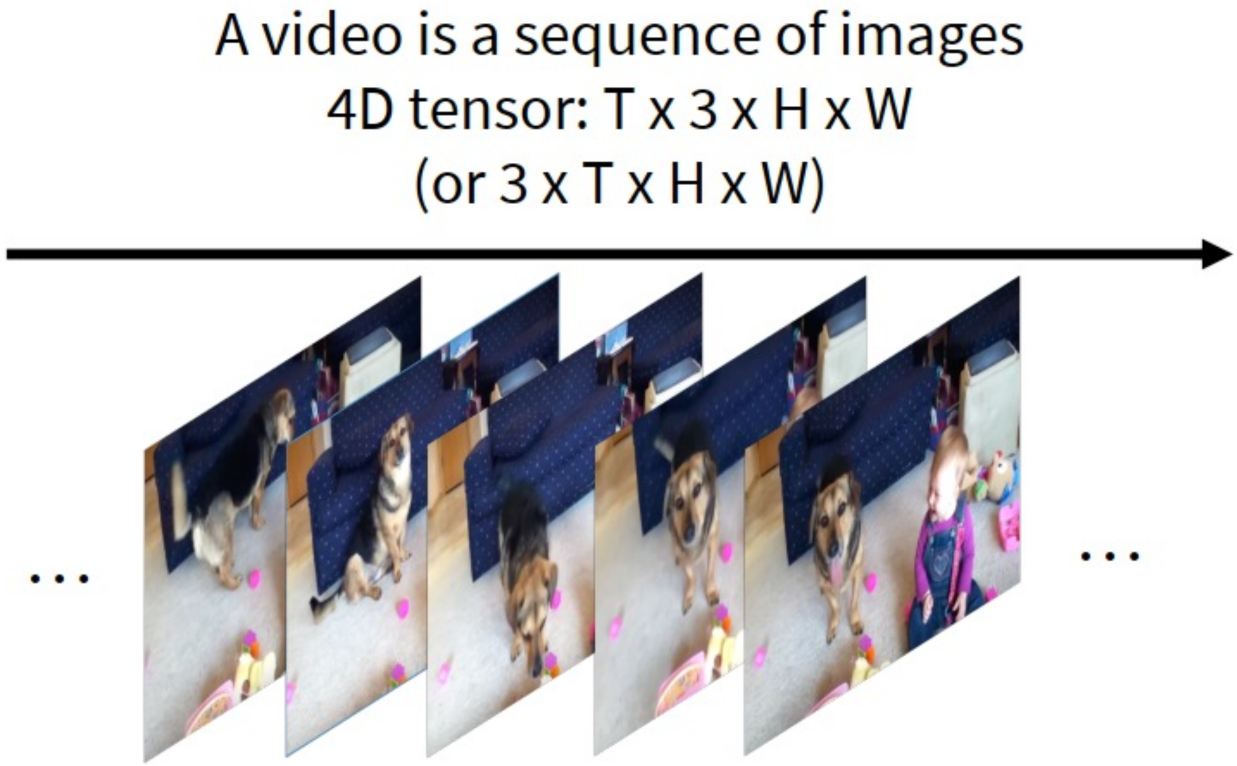


Venugopalan et al., “Sequence to Sequence - Video to Text”, ICCV 2015

Plan for this lecture

- Language and vision
  - Application: Image and video captioning
  - Tool: Recurrent neural networks
  - Tool: Transformers
  - Application: Visual question answering
- Motion and video
  - Video classification
  - Measuring motion
  - Tracking objects

Video Classification



Video Classification: Example

Input video:
T x 3 x H x W

Swimming
Running
Jumping
Eating
Standing

Running video is in the public domain

Slide credit: Justin Johnson

Video Classification: Example

Images: Recognize objects

Dog
Cat
Fish
Truck

Videos: Recognize actions

Swimming
Running
Jumping
Eating
Standing

Slide credit: Justin Johnson

Problem: Videos are big!

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video (3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): **~10 GB per minute**

Solution: Train on short **clips:** low fps and low spatial resolution
e.g. T = 16, H=W=112
(3.2 seconds at 5 fps, 588 KB)

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS

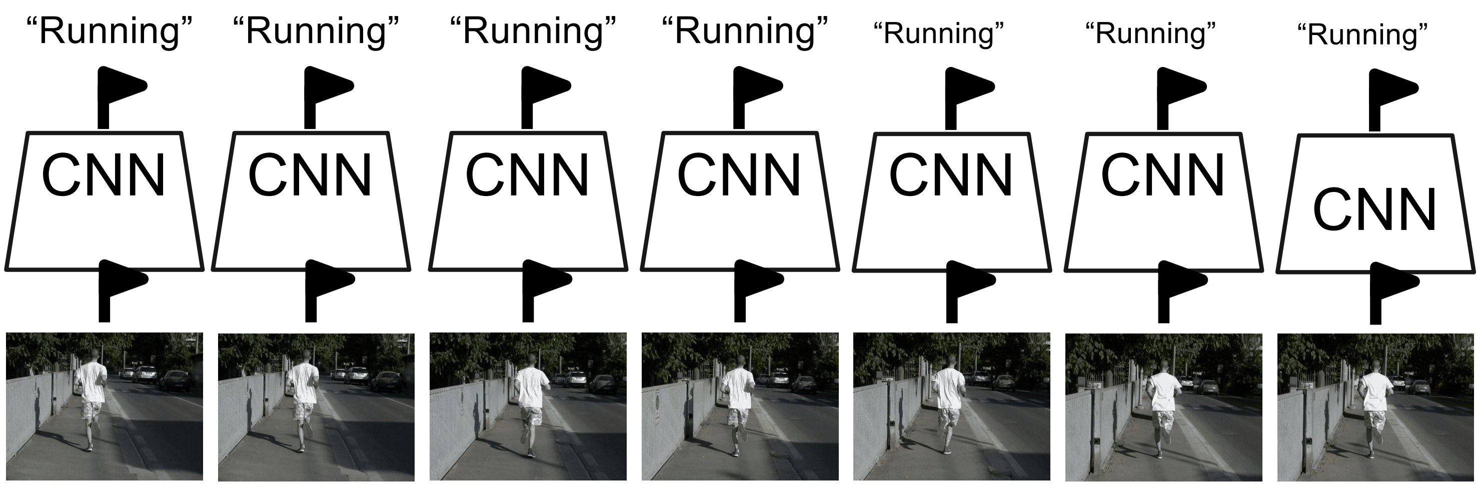
Training: Train model to classify short **clips** with low FPS

Testing: Run model on different clips, average predictions

Slide credit: Justin Johnson

Video Classification: Single-Frame CNN

Simple idea: train normal 2D CNN to classify video frames independently! (Average predicted probs at test-time)
Often a **very** strong baseline for video classification

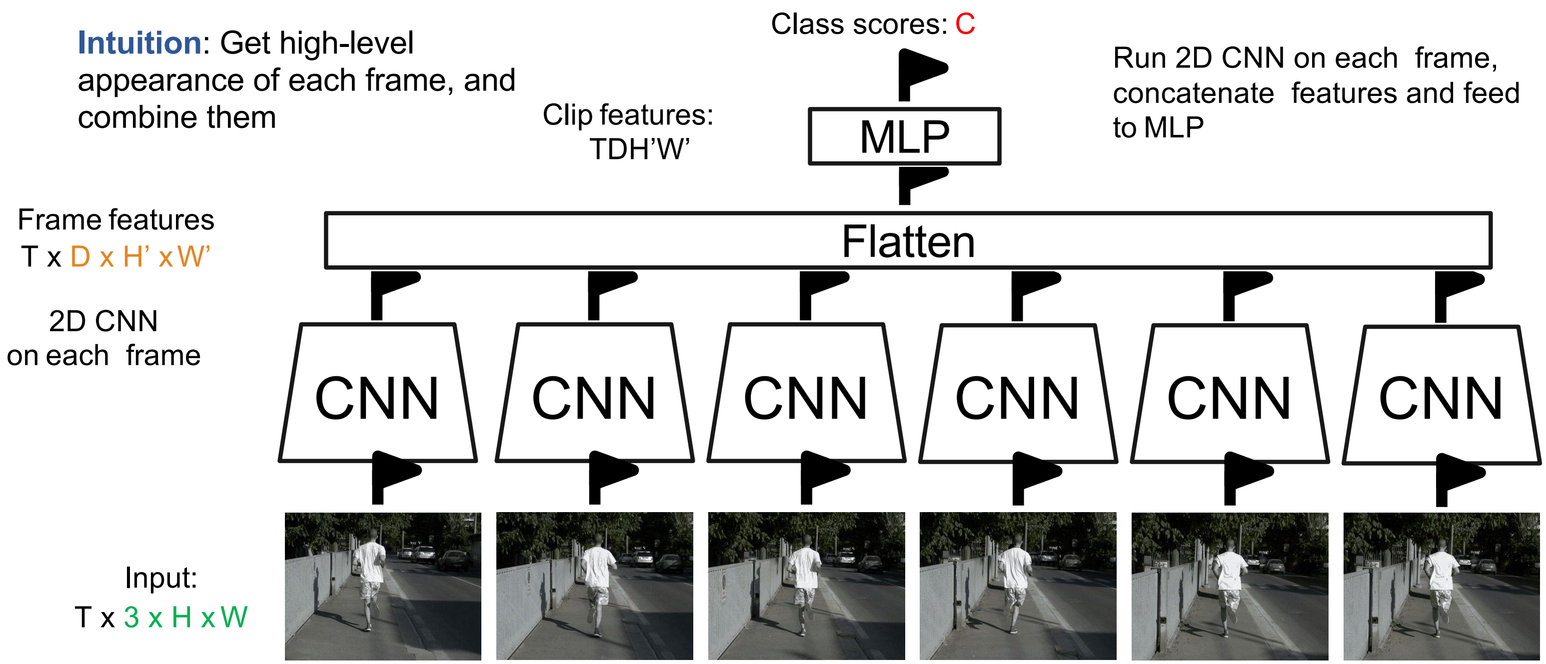


Slide credit: Justin Johnson

Video Classification: Late Fusion (with FC layers)

Intuition: Get high-level appearance of each frame, and combine them

Run 2D CNN on each frame, concatenate features and feed to MLP



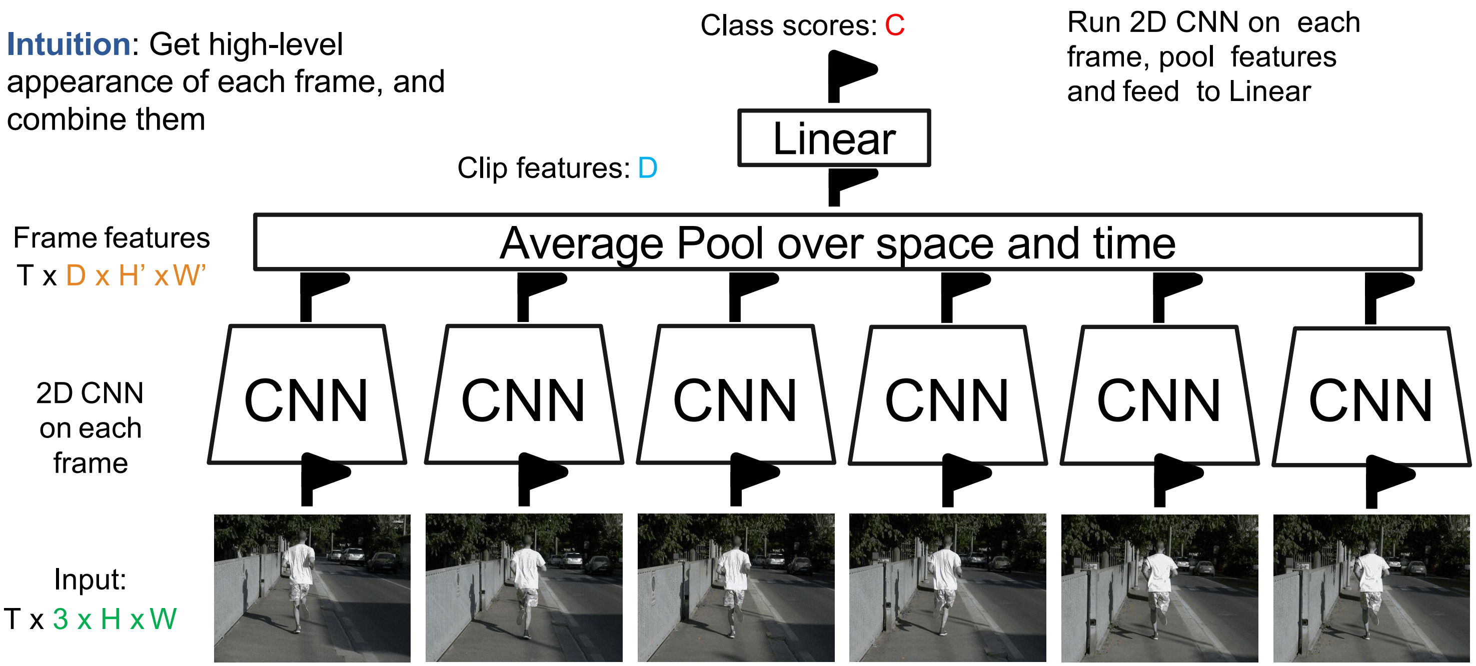
Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Video Classification: Late Fusion (with pooling)

Intuition: Get high-level appearance of each frame, and combine them

Run 2D CNN on each frame, pool features and feed to Linear



Slide credit: Justin Johnson

Video Classification: Early Fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

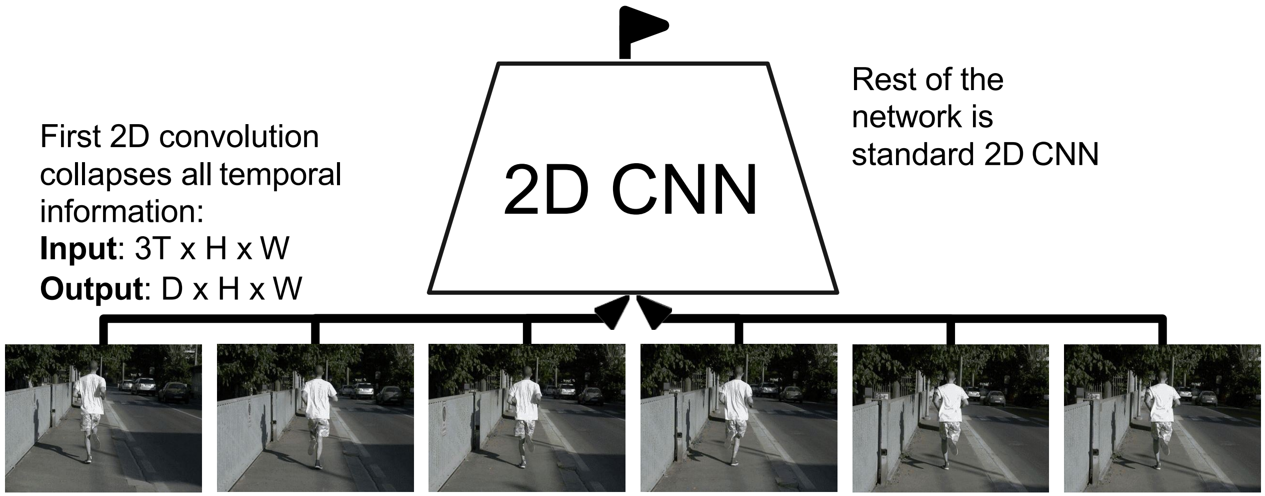
Class scores: C

Rest of the network is standard 2D CNN

First 2D convolution collapses all temporal information:
Input: 3T x H x W
Output: D x H x W

Reshape:
3T x H x W

Input:
T x 3 x H x W



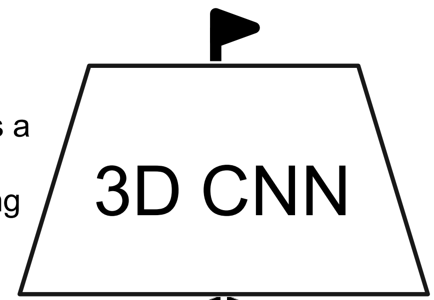
Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Video Classification: 3D CNN

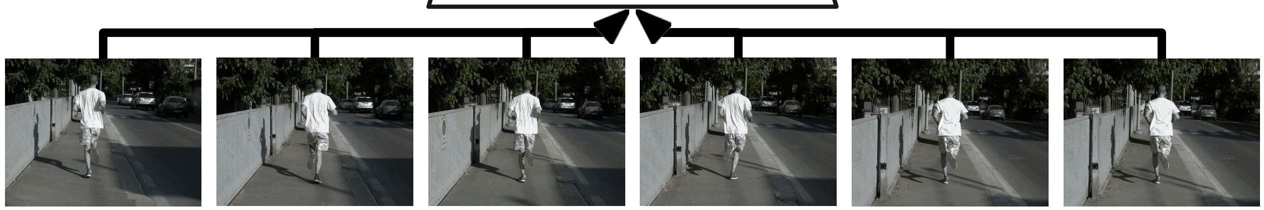
Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

Class scores: C



Each layer in the network is a 4D tensor: D x T x H x W
Use 3D conv and 3D pooling operations

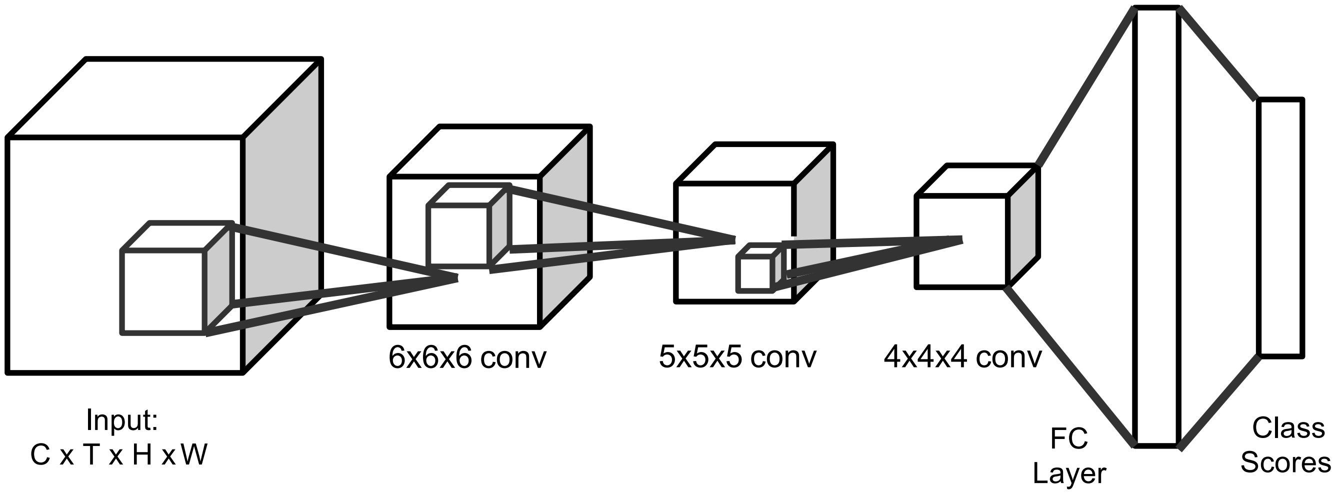
Input:
3 x T x H x W



Ji et al, "3D Convolutional Neural Networks for Human Action Recognition", TPAMI 2010 ; Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

3D Convolution



Slide credit: Fei-Fei Li, Yunzhu Li, Ruohan Gao

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on Sports-1M:
Many people used this as a video feature extractor

See on our

Layer	Size
Input	3 x 16 x 112 x 112
Conv1 (3x3x3)	64 x 16 x 112 x 112
Pool1 (1x2x2)	64 x 16 x 56 x 56
Conv2 (3x3x3)	128 x 16 x 56 x 56
Pool2 (2x2x2)	128 x 8 x 28 x 28
Conv3a (3x3x3)	256 x 8 x 28 x 28
Conv3b (3x3x3)	256 x 8 x 28 x 28
Pool3 (2x2x2)	256 x 4 x 14 x 14
Conv4a (3x3x3)	512 x 4 x 14 x 14
Conv4b (3x3x3)	512 x 4 x 14 x 14
Pool4 (2x2x2)	512 x 2 x 7 x 7
Conv5a (3x3x3)	512 x 2 x 7 x 7
Conv5b (3x3x3)	512 x 2 x 7 x 7
Pool5	512 x 1 x 3 x 3
FC6	4096
FC7	4096
FC8	C

Slide credit: Fei-Fei Li, Yunzhu Li, Ruohan Gao

Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015

Example Video Dataset: Sports-1M

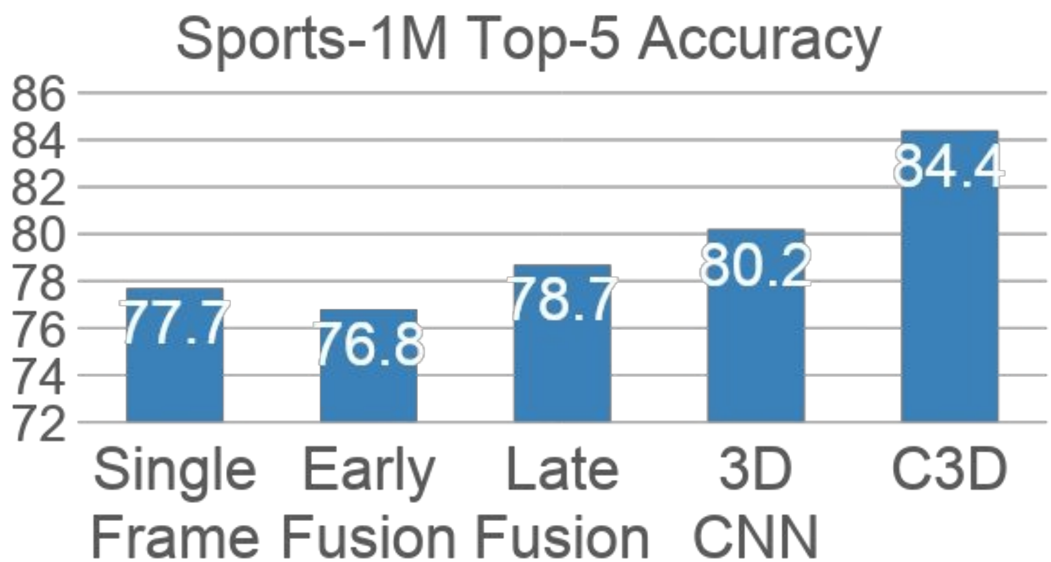
1 million YouTube videos annotated with labels for 487 different types of sports

Ground Truth
Correct prediction
Incorrect prediction

Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014
Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015

Slide credit: Justin Johnson

Motion: Why is it useful?

Derek Hoiem

Motion: Why is it useful?

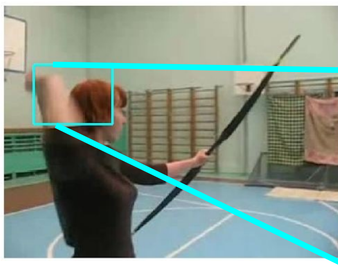
- Even “impoverished” motion data can evoke a strong percept

G. Johansson, “Visual Perception of Biological Motion and a Model For Its Analysis", *Perception and Psychophysics 14, 201-211, 1973.*

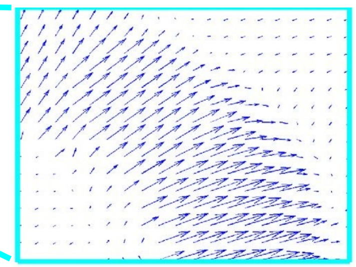
Derek Hoiem

Measuring Motion: Optical Flow

Image at frame t



Optical flow gives a displacement field F between images I_t and I_{t+1}



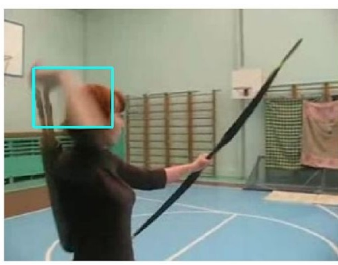


Image at frame t+1

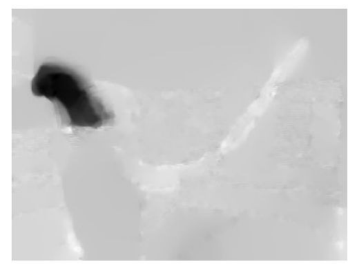
Tells where each pixel will move in the next frame:

$F(x, y) = (dx, dy)$

$I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Optical Flow highlights **local motion**

Horizontal flow dx



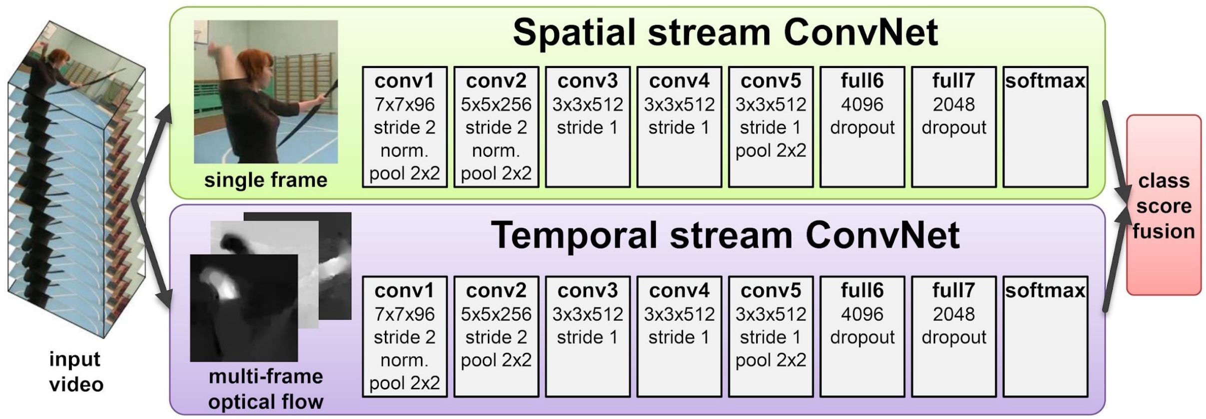
Vertical Flow dy

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Slide credit: Justin Johnson

Separating Motion and Appearance: Two-Stream Networks

Input: Single Image
3 x H x W



Input: Stack of optical flow:
[2*(T-1)] x H x W

Early fusion: First 2D conv processes all flow images

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

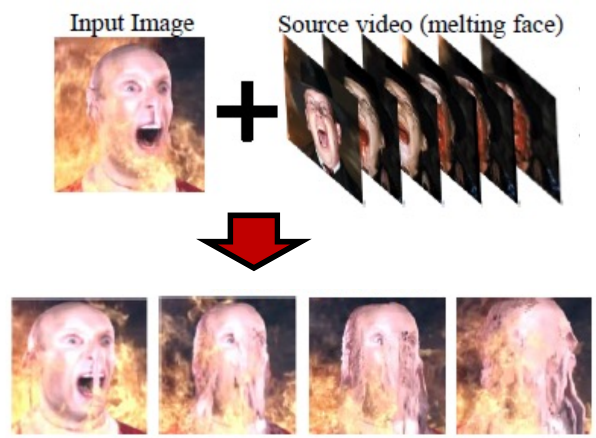
Slide credit: Justin Johnson

Modeling Motion: Optical Flow

(a) Input Image (b) Prediction

Walker et al., "Dense Optical Flow Prediction from a Static Scene", ICCV 2015

Transferring Motion

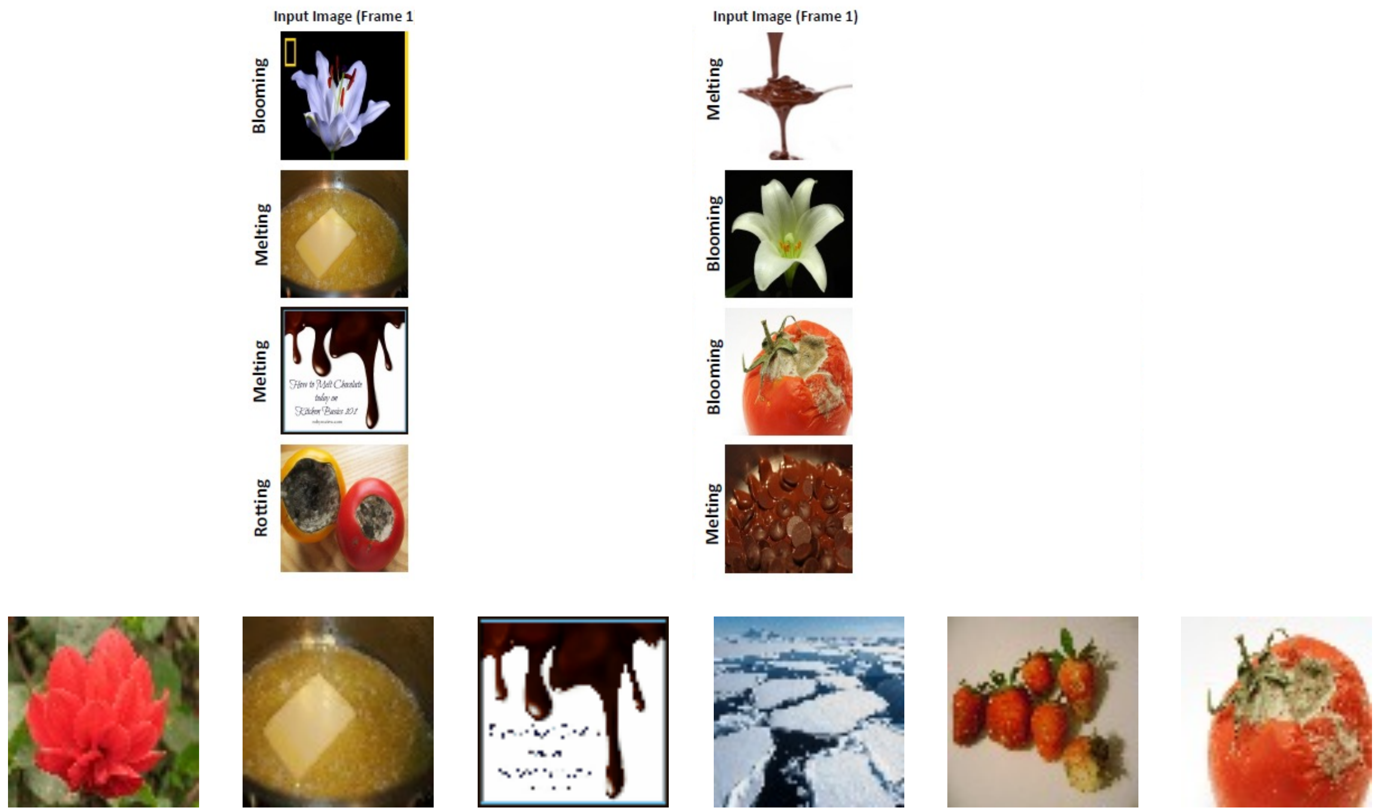


$$\mathcal{L}_{\text{flow}}(\mathbf{y}_{i-1}, \mathbf{y}_i; \mathbf{s}_{i-1}, \mathbf{s}_i) = \sum_l \frac{1}{C_l H_l W_l} \| \underbrace{\Xi(\mathbf{y}_{i-1}, \mathbf{y}_i)_l}_{\text{Optical flow in generated video}} - \underbrace{\Xi(\mathbf{s}_{i-1}, \mathbf{s}_i)_l}_{\text{Optical flow in source video}} \|_2^2$$

Key idea: Generate videos with **similar flow patterns** as source videos (+ many details).

Thomas, Song and Kovashka

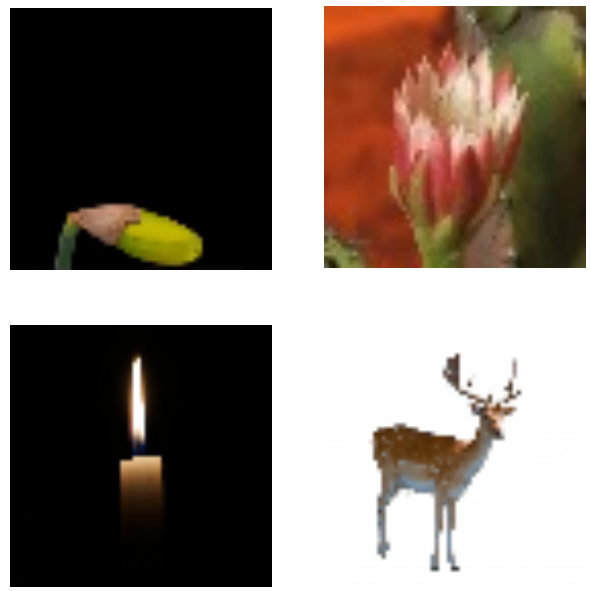
Transferring Motion



Thomas, Song and Kovashka

Transferring Motion

Baking

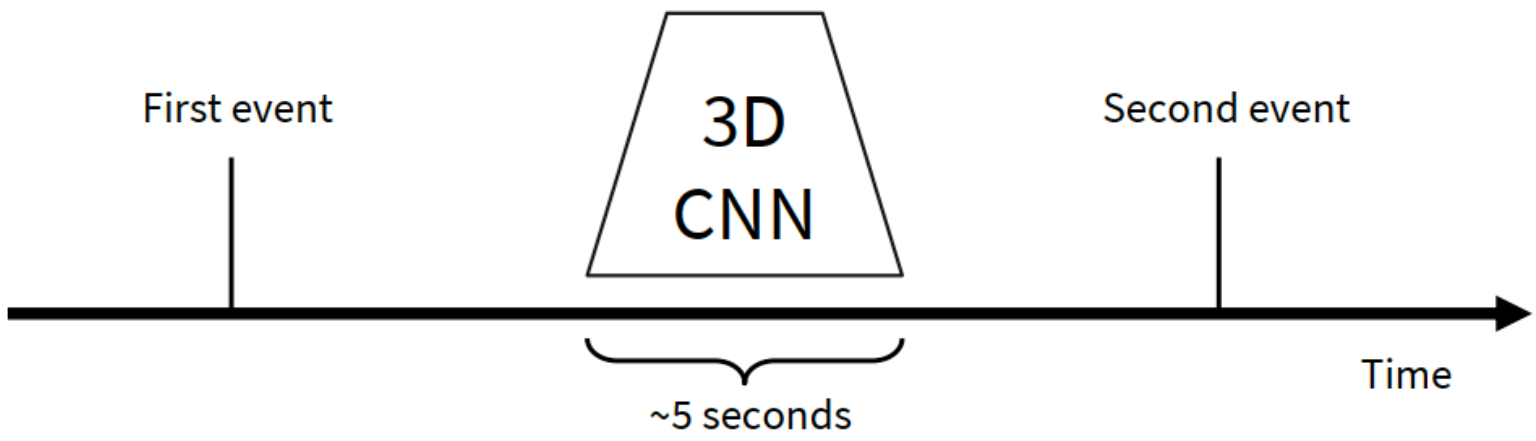


Blooming

Modeling Long-term Temporal Structure

So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

We know how to handle sequences! How about recurrent networks?

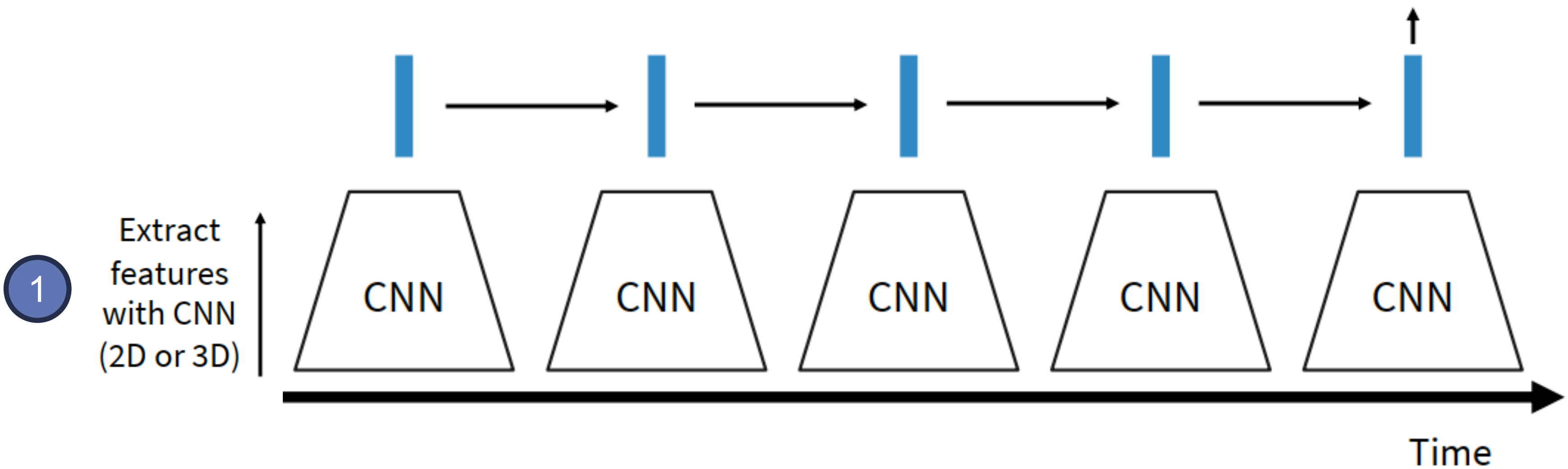


Slide credit: Justin Johnson

Modeling Long-term Temporal Structure

2 Process local features using recurrent network (e.g. LSTM)

3 Many to one: One output at end of video

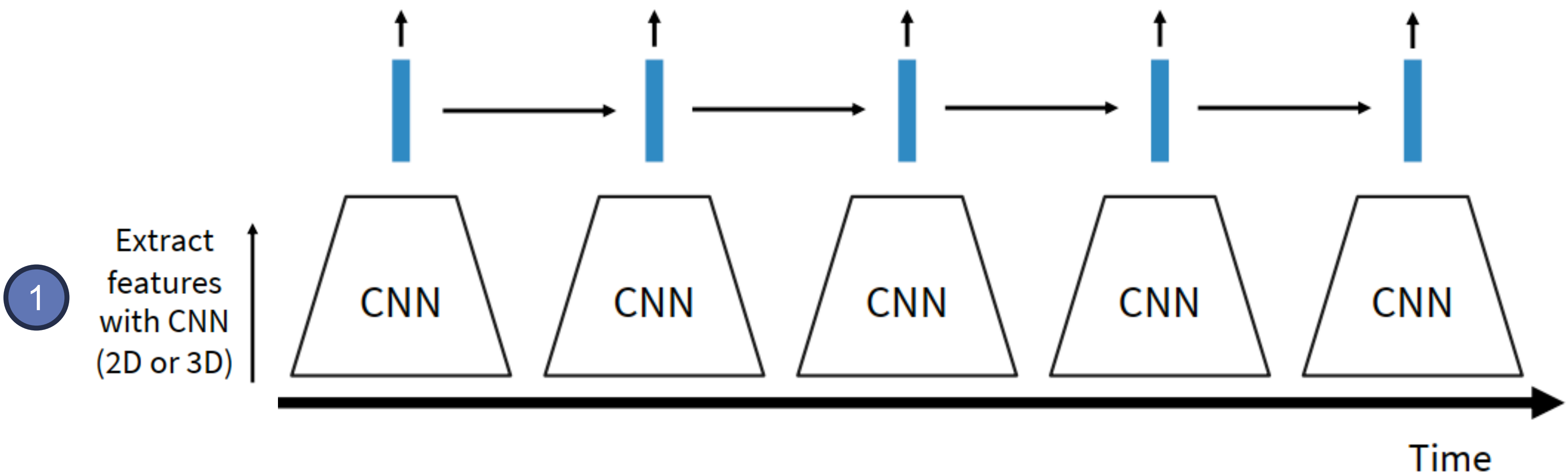


Slide credit: Justin Johnson

Modeling Long-term Temporal Structure

2 Process local features using recurrent network (e.g. LSTM)

3 Many to many: one output per video frame



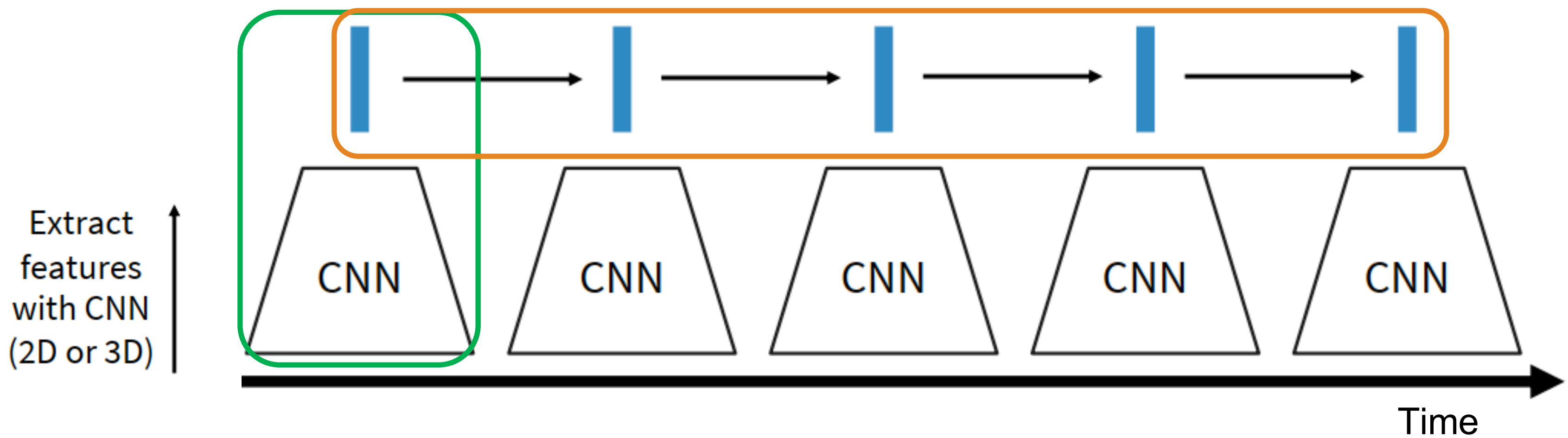
Slide credit: Justin Johnson

Modeling Long-term Temporal Structure

Inside CNN: Each value is a function of a fixed temporal window (local temporal structure)

Inside RNN: Each vector is a function of all previous vectors (global temporal structure)

Can we merge both approaches?

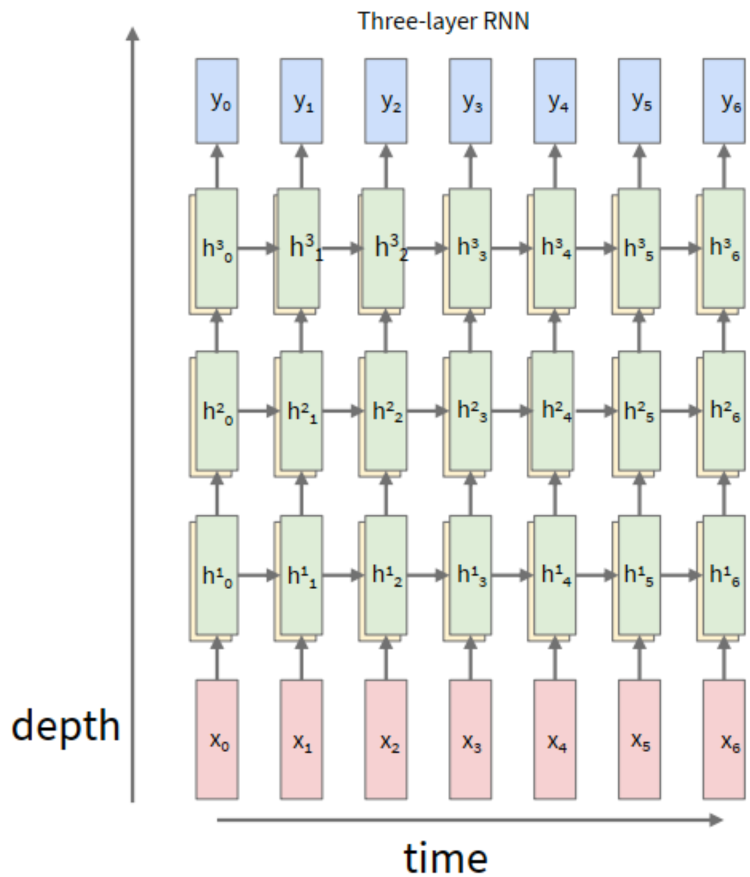


Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Slide credit: Justin Johnson

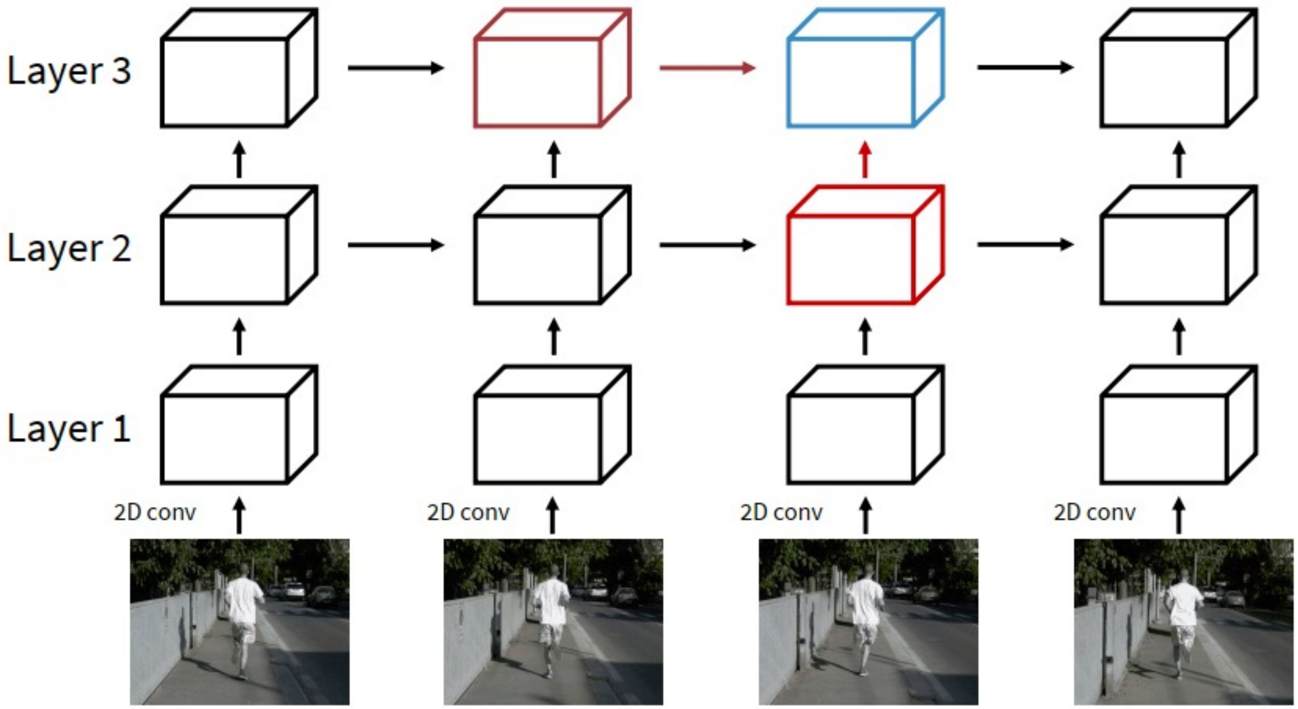
Intuition: Multi-layer RNN

We can use a similar structure to process videos!



Slide credit: Justin Johnson

Recurrent Convolutional Network



Entire network uses 2D feature maps: C x H x W

Each depends on two inputs:
1. Same layer, previous timestep
2. Prev layer, same timestep

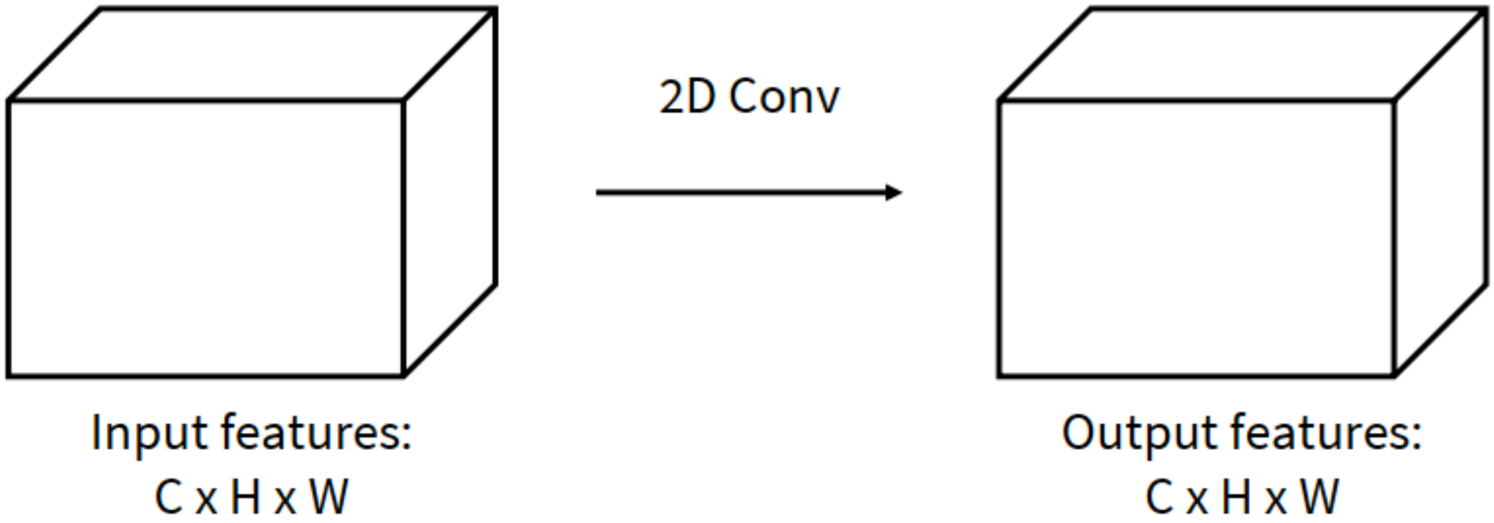
Use different weights at each layer, share weights across time

Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

Recurrent Convolutional Network

Normal 2D CNN:



Slide credit: Justin Johnson

Recurrent Convolutional Network

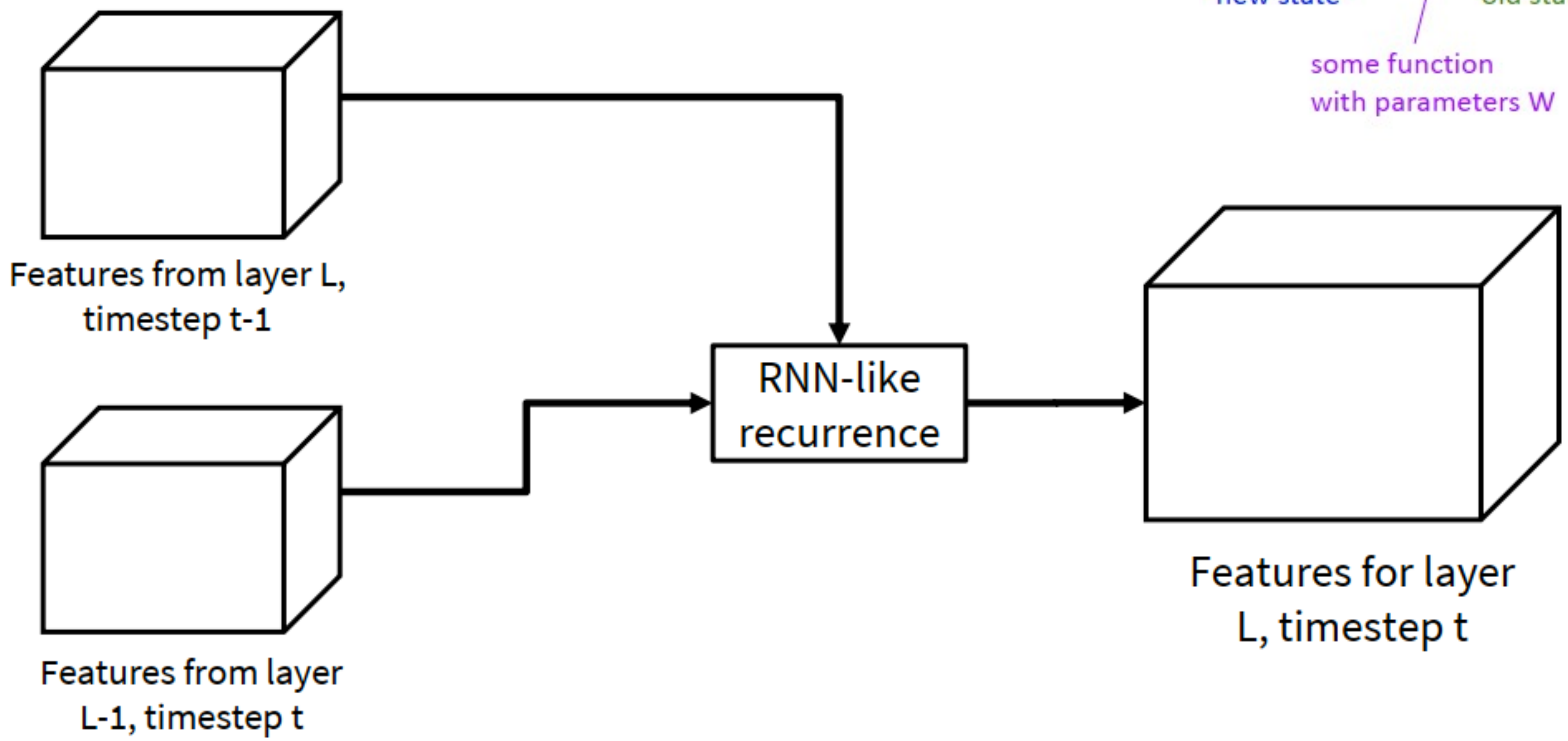
Recall: Recurrent Network

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

new state

old state

some function
with parameters W



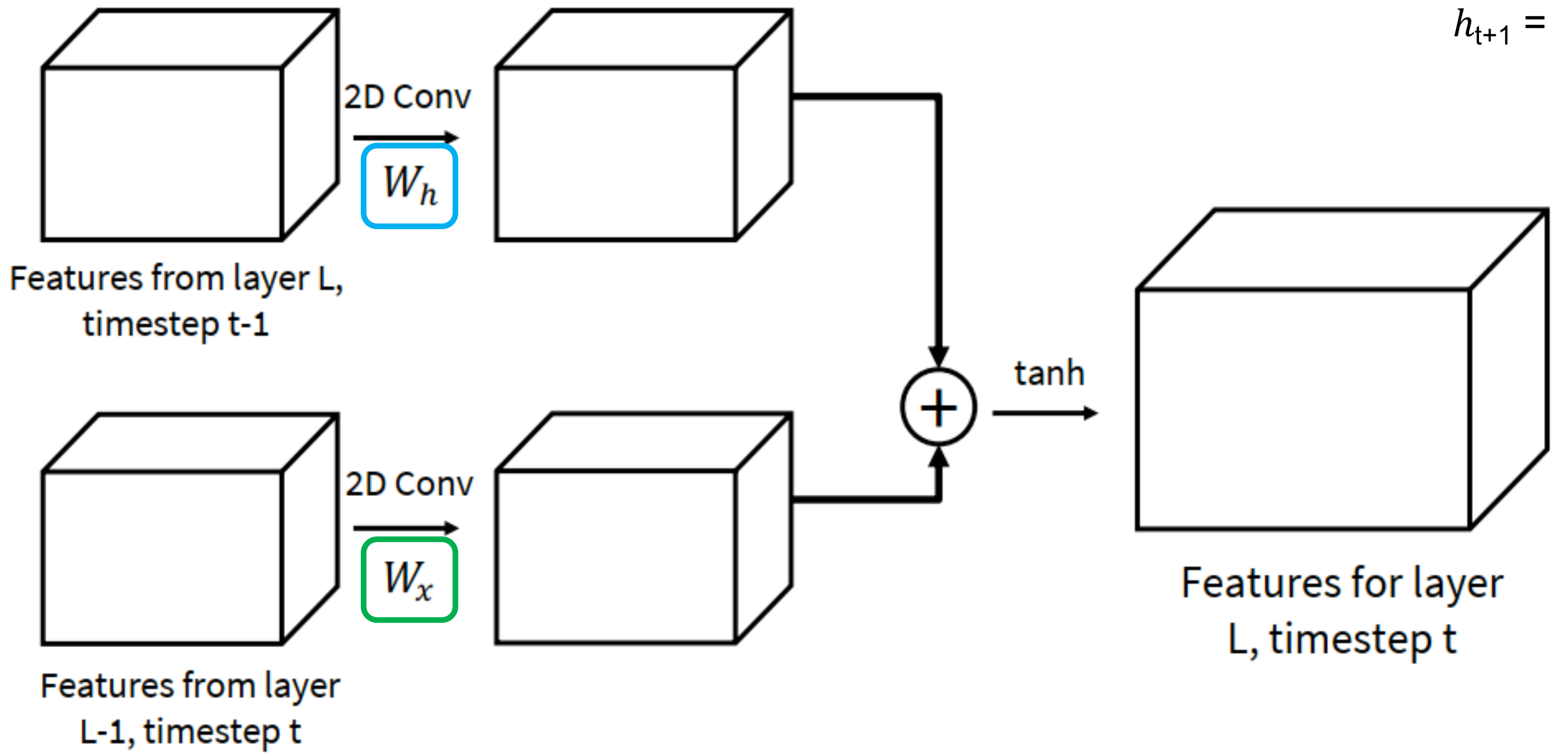
Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

Recurrent Convolutional Network

Recall: Vanilla RNN

$h_{t+1} = \tanh(W_h h_t + W_x x)$

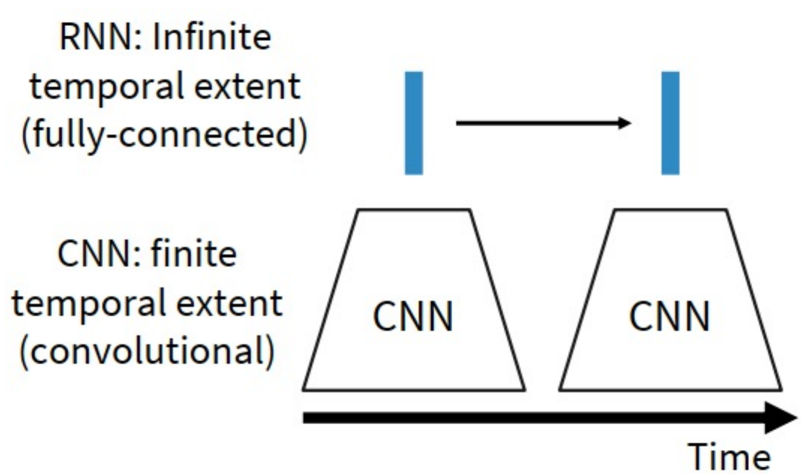


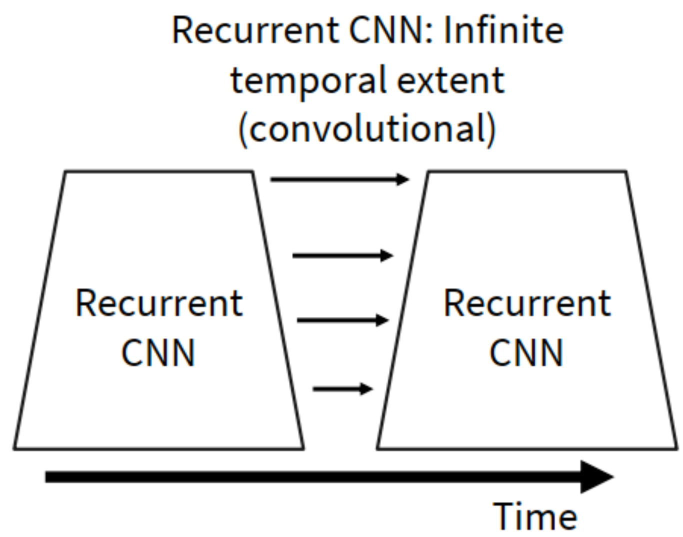
Ballas et al, “Delving Deeper into Convolutional Networks for Learning Video Representations”, ICLR 2016

Slide credit: Justin Johnson

Modeling Long-term Temporal Structure

Problem: RNNs are slow for long sequences (can't be parallelized)





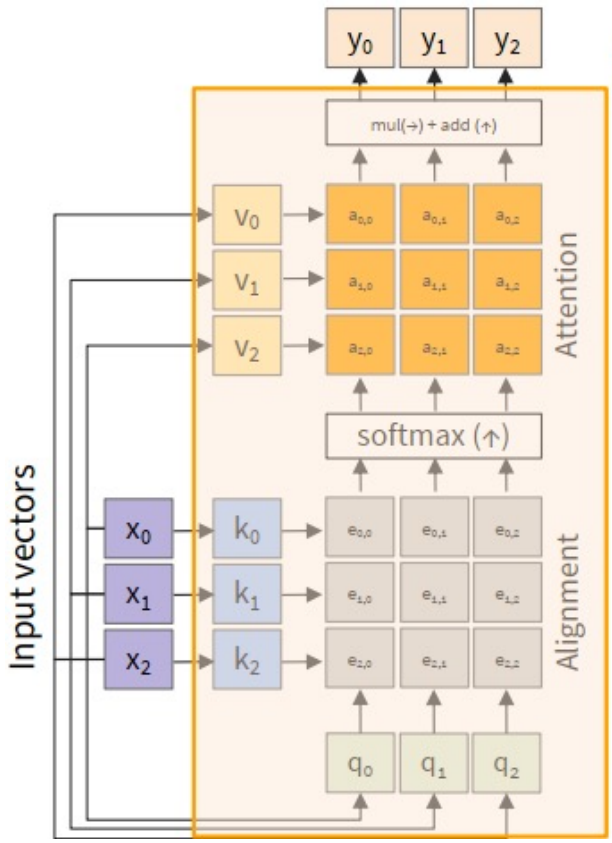
Time

Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

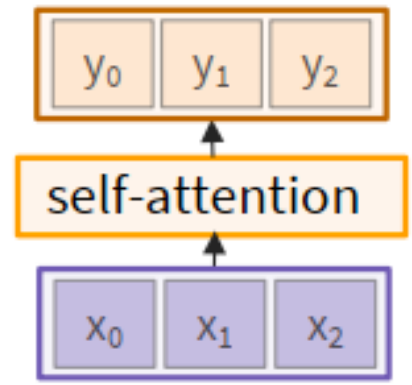
Recall: Self-Attention



Outputs:
context vectors: y (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: x (shape: N x D)



Slide credit: Fei-Fei Li

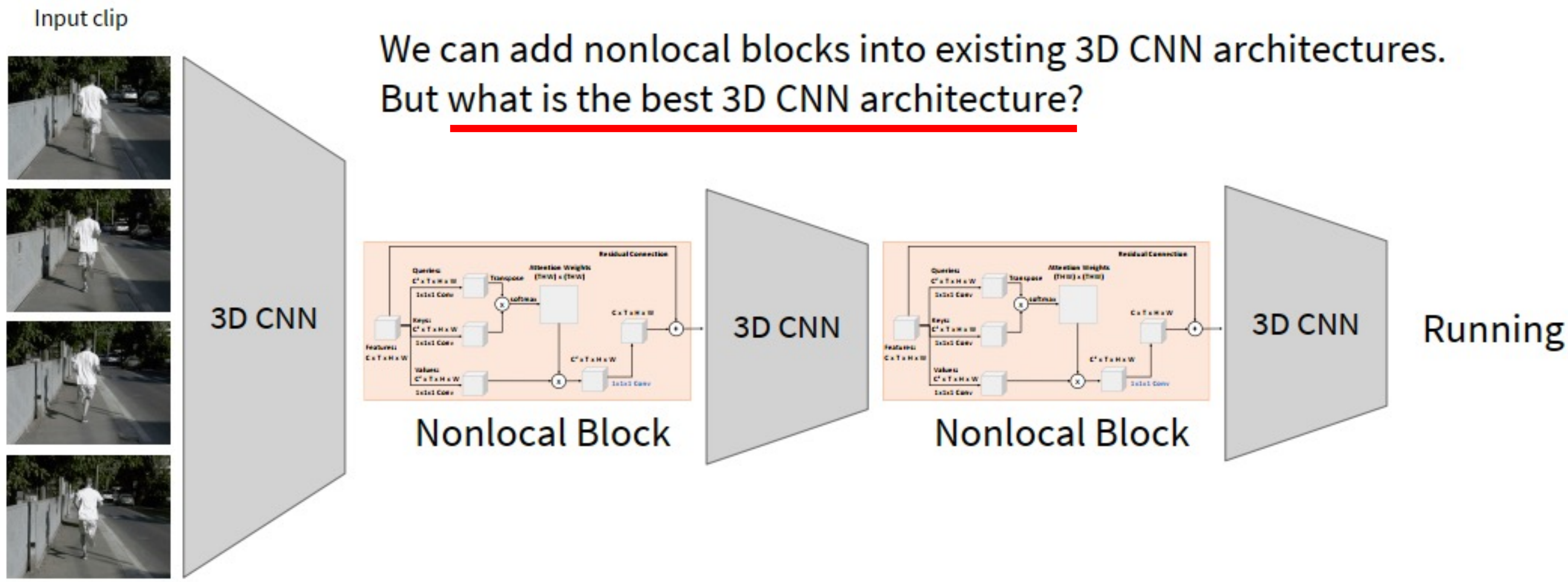
Spatio-Temporal Self-Attention (Nonlocal Block)

Wang et al, "Non-local neural networks", CVPR 2018

Slide credit: Justin Johnson

Spatio-Temporal Self-Attention (Nonlocal Block)

We can add nonlocal blocks into existing 3D CNN architectures.
But what is the best 3D CNN architecture?



Wang et al, "Non-local neural networks", CVPR 2018

Slide credit: Justin Johnson

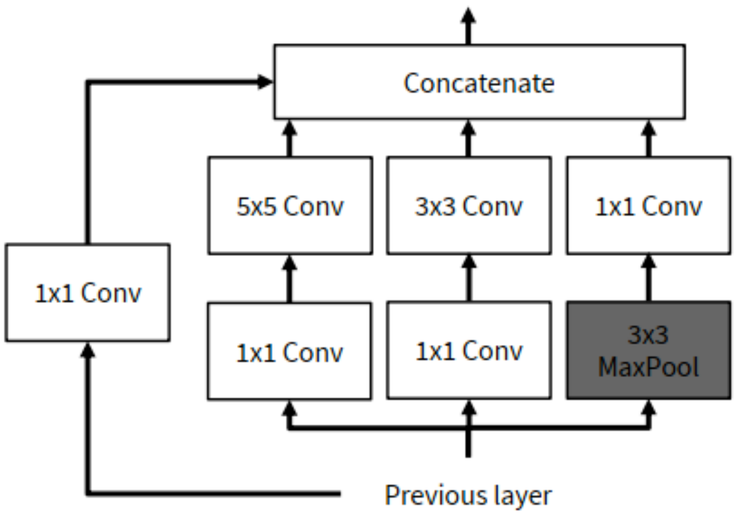
Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.
Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each 2D K_h x K_w conv/pool layer with a 3D K_t x K_h x K_w version

Inception Block: Original



Carreira and Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", CVPR 2017

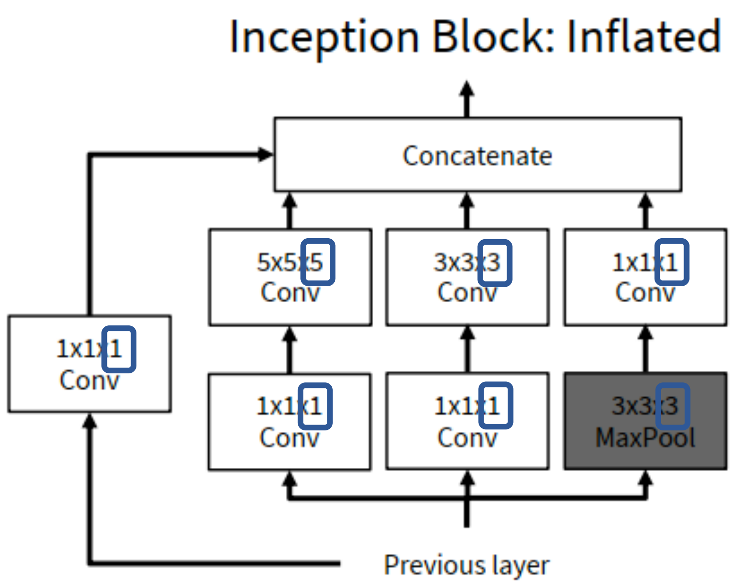
Slide credit: Justin Johnson

Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.
Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each 2D K_h x K_w conv/pool layer with a 3D K_t x K_h x K_w version



Carreira and Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", CVPR 2017

Slide credit: Justin Johnson

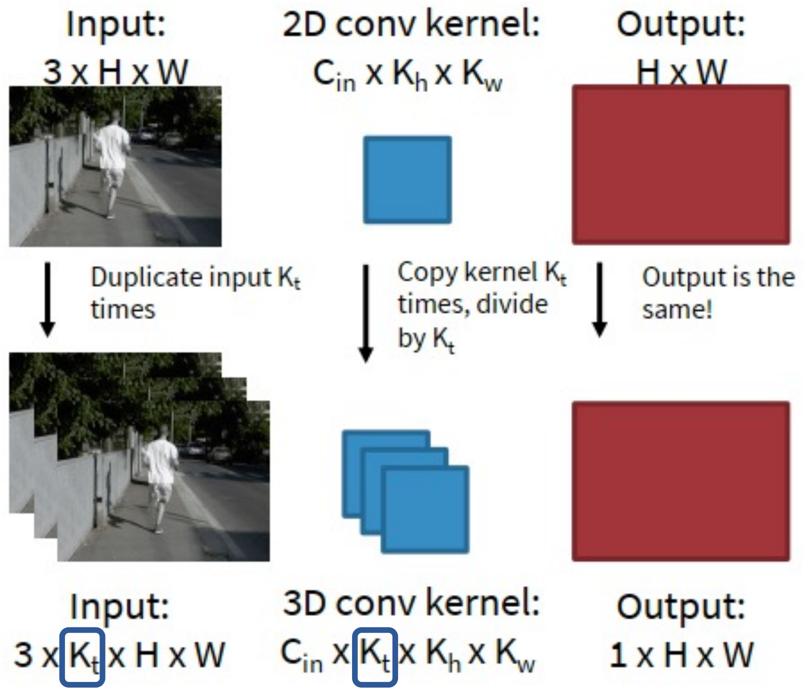
Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.
Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each 2D K_h x K_w conv/pool layer with a 3D K_t x K_h x K_w version

Can use weights of 2D conv to initialize 3D conv: copy K_t times in space and divide by K_t
This gives the same result as 2D conv given “constant” video input



Carreira and Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”, CVPR 2017

Slide credit: Justin Johnson

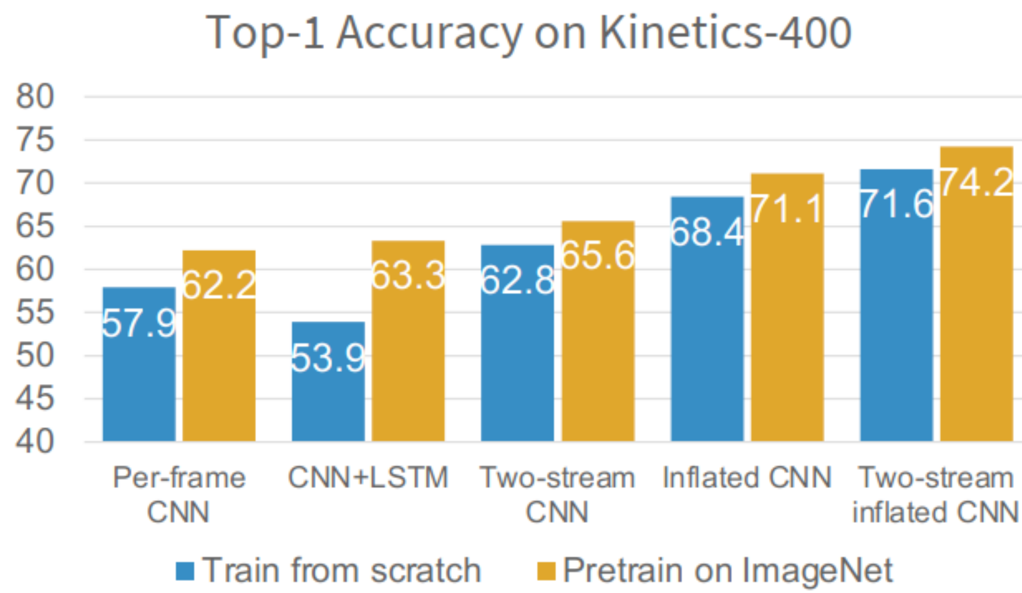
Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.
Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each 2D K_h x K_w conv/pool layer with a 3D K_t x K_h x K_w version

Can use weights of 2D conv to initialize 3D conv: copy K_t times in space and divide by K_t
This gives the same result as 2D conv given "constant" video input

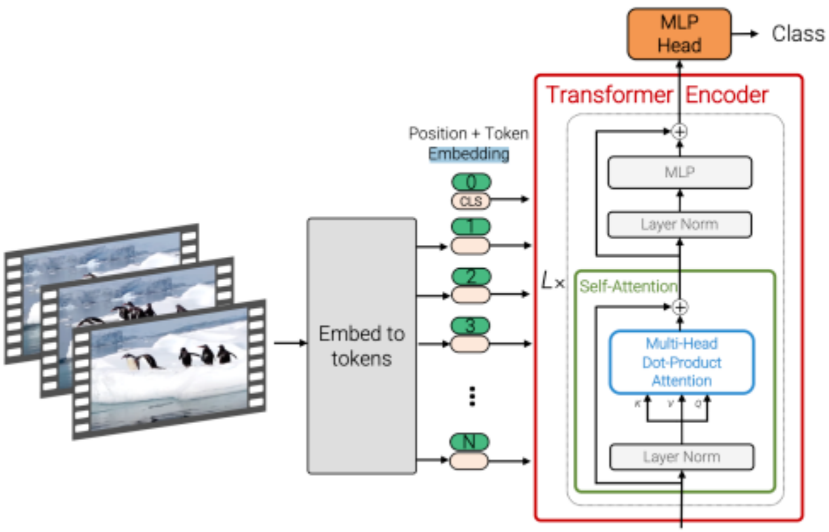


Carreira and Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", CVPR 2017

Slide credit: Fei-Fei Li

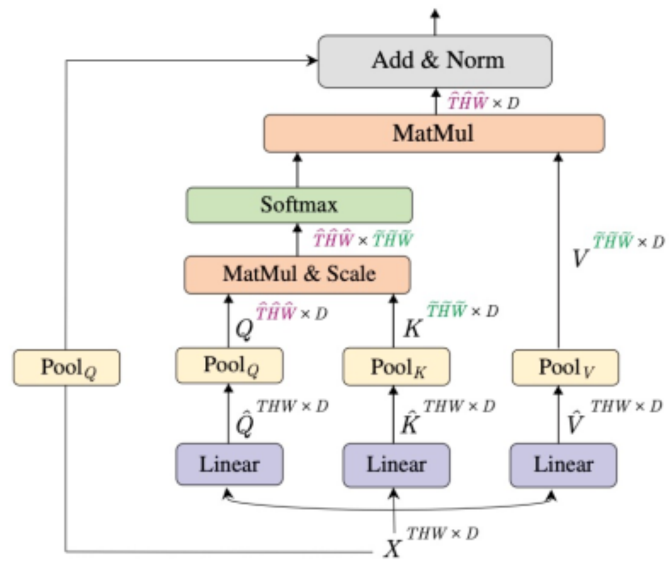
Vision Transformers for Video

Factorized attention: Attend over space / time



Bertasius et al, "Is Space-Time Attention All You Need for Video Understanding?", ICML 2021
Arnab et al, "ViViT: A Video Vision Transformer", ICCV 2021
Neimark et al, "Video Transformer Network", ICCV 2021

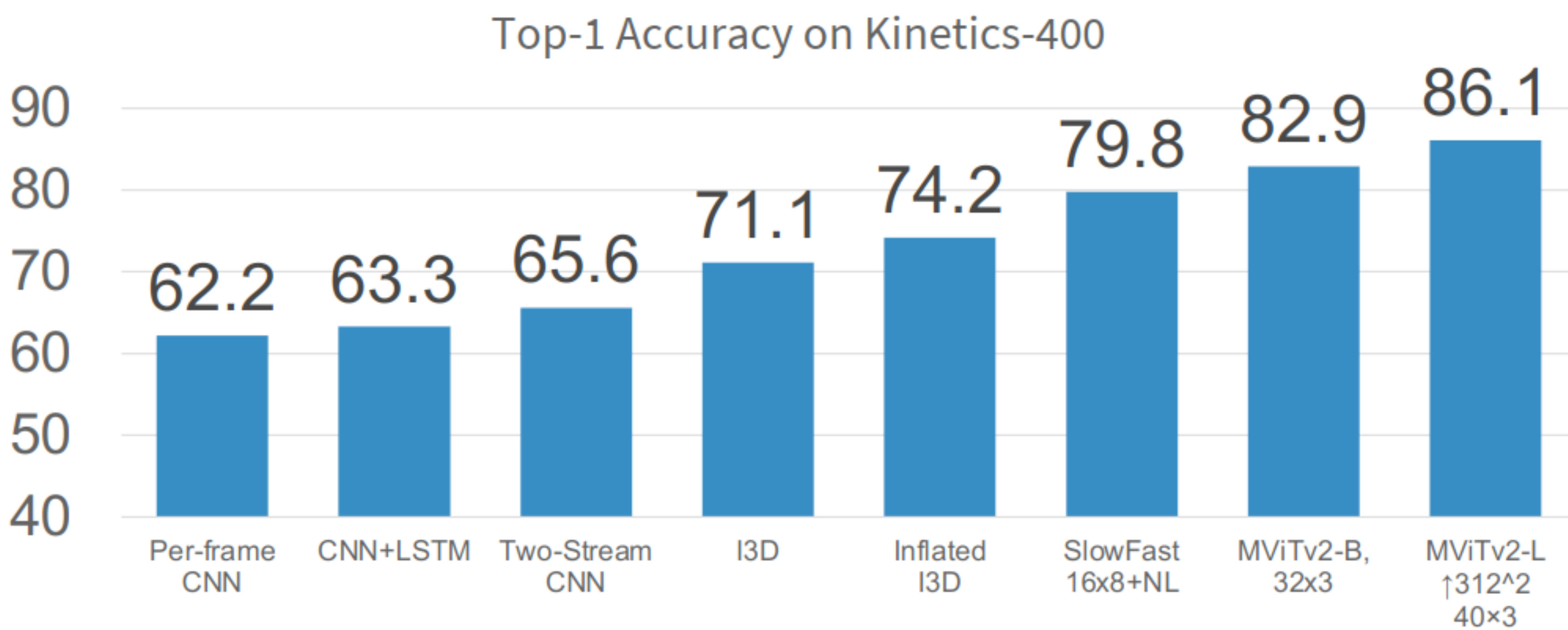
Pooling module: Reduce number of tokens



Fan et al, "Multiscale Vision Transformers", ICCV 2021
Li et al, "MViTv2: Improved Multiscale Vision Transformers for Classification and Detection", CVPR 2022

Slide credit: Fei-Fei Li

Vision Transformers for Video



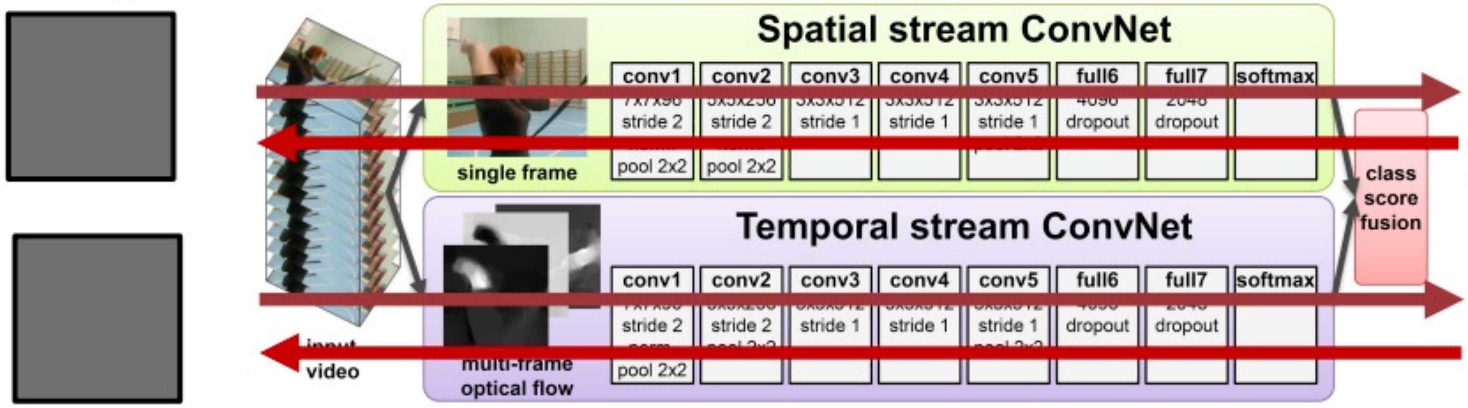
Li et al, "MViTv2: Improved Multiscale Vision Transformers for Classification and Detection", CVPR 2022

Slide credit: Justin Johnson

Visualizing Video Models

Image

Forward: Compute class score



”weightlifting ” score

Flow

Backward: Compute gradient

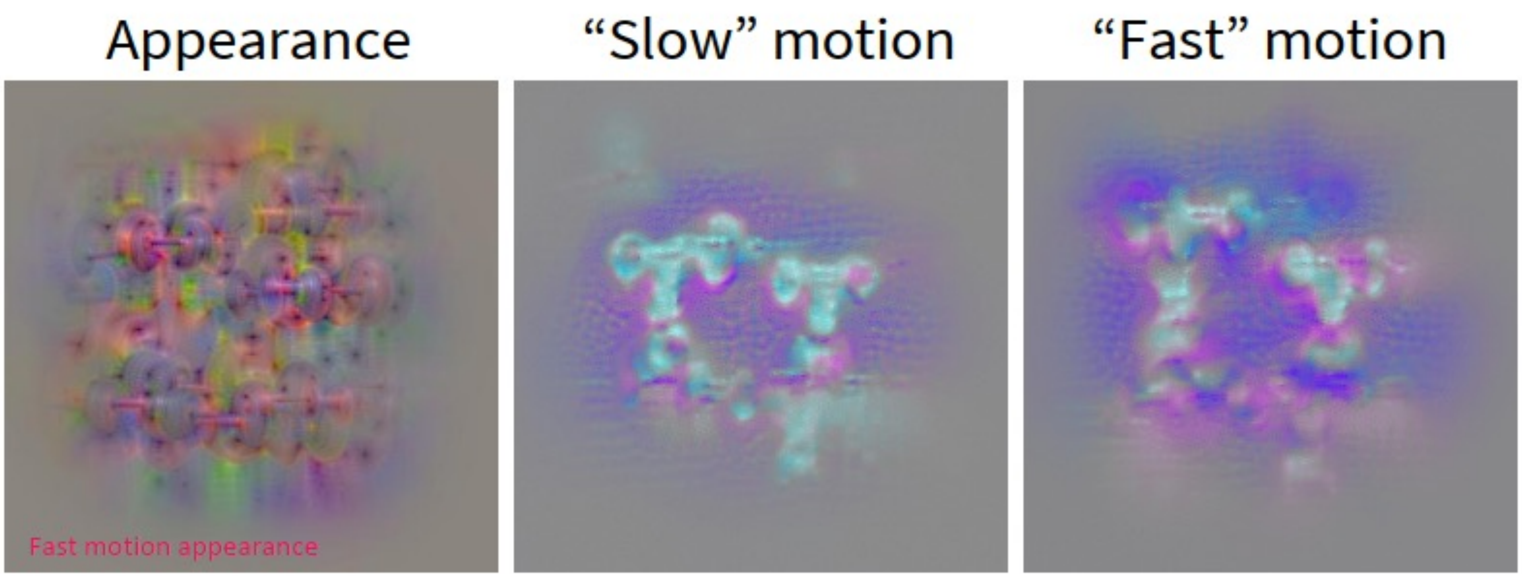
Add a term to encourage spatially smooth flow; tune penalty to pick out “slow” vs “fast” motion

Figure credit: Simonyan and Zisserman, “Two-stream convolutional networks for action recognition in videos”, NeurIPS 2014
Feichtenhofer et al, “What have we learned from deep representations for action recognition?”, CVPR 2018
Feichtenhofer et al, “Deep insights into convolutional networks for video recognition?”, IJCV 2019.

Slide credit: Justin Johnson

Visualization: Can you guess the action?

Feichtenhofer et al, "What have we learned from deep representations for action recognition?", CVPR 2018
Feichtenhofer et al, "Deep insights into convolutional networks for video recognition?", IJCV 2019.
Slide credit: Christoph Feichtenhofers

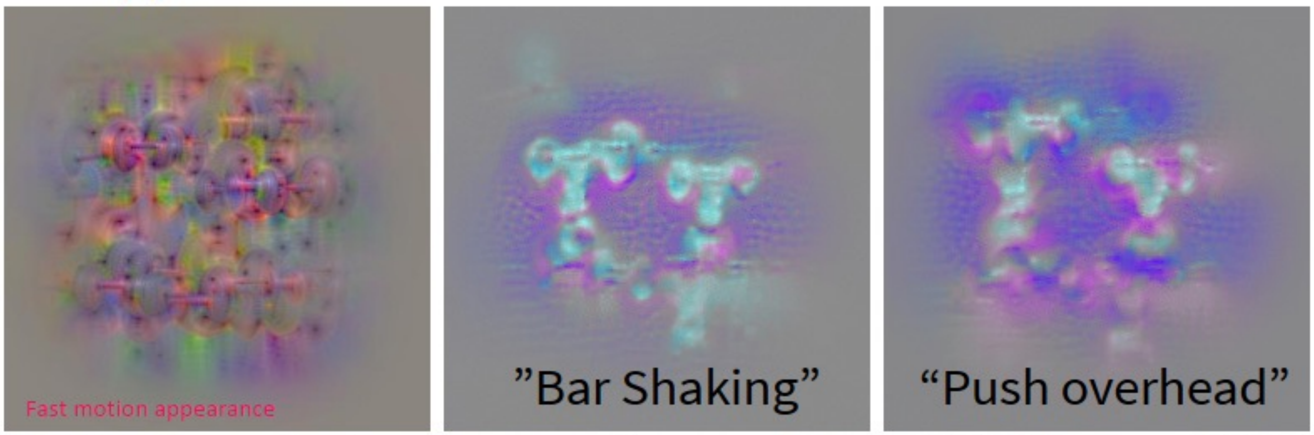


Slide credit: Justin Johnson

Visualization: Can you guess the action?

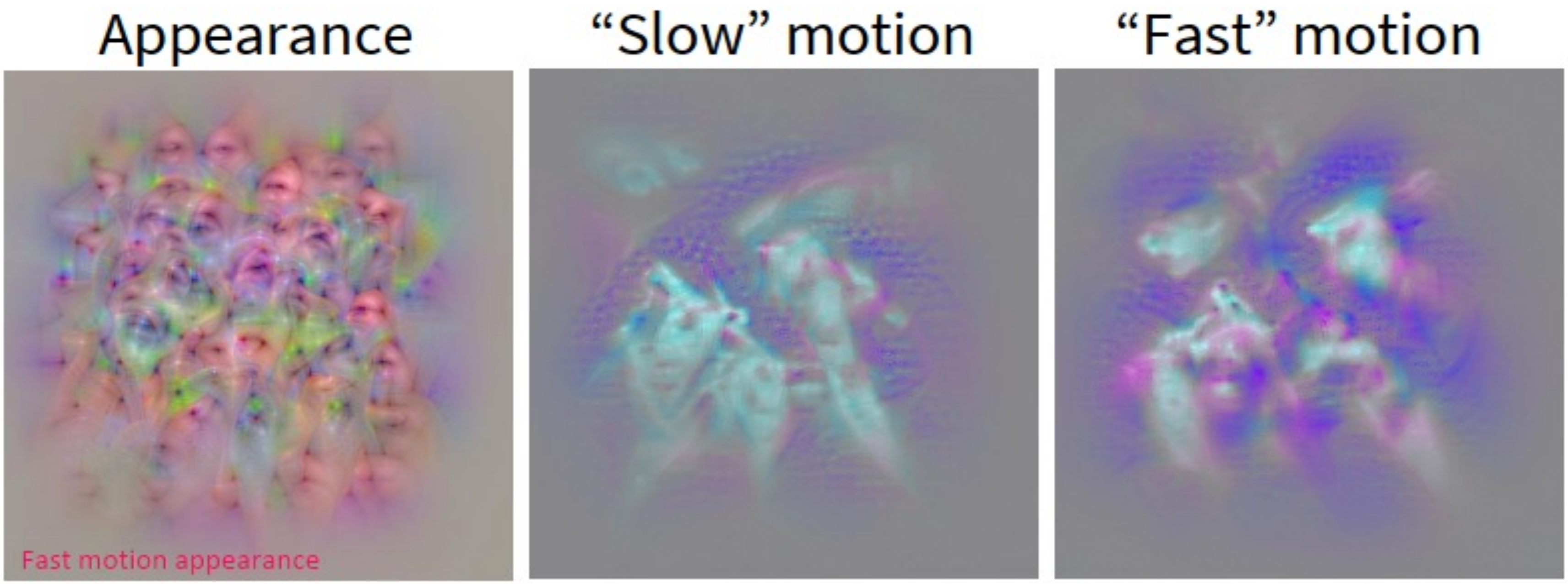
Weightlifting

Appearance | “Slow” motion | “Fast” motion



Slide credit: Justin Johnson

Visualization: Can you guess the action?

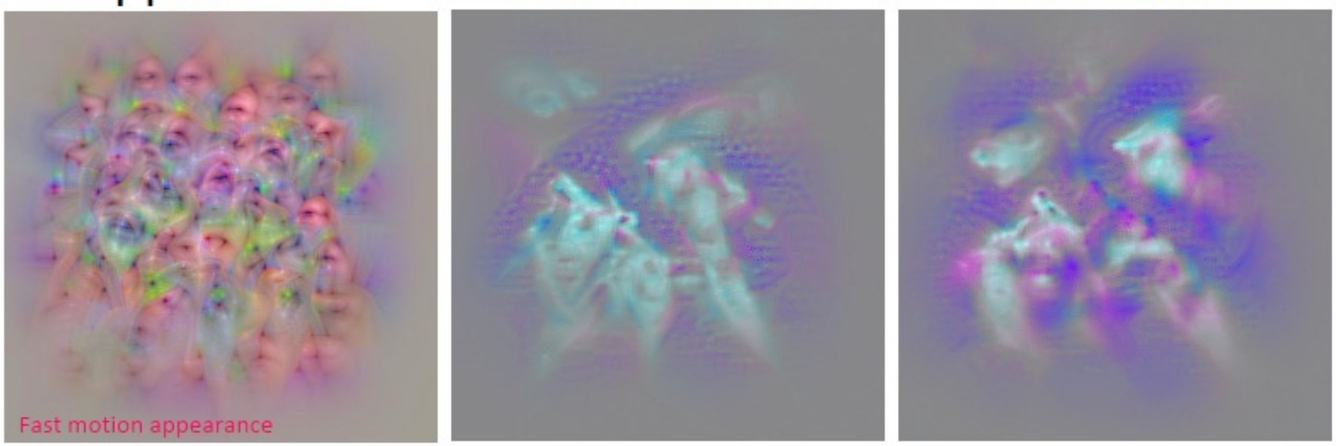


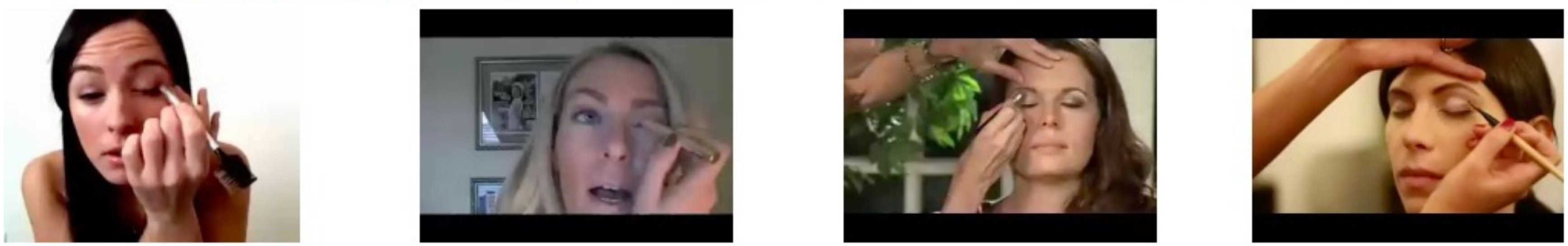
Slide credit: Justin Johnson

Visualization: Can you guess the action?

Apply Eye Makeup

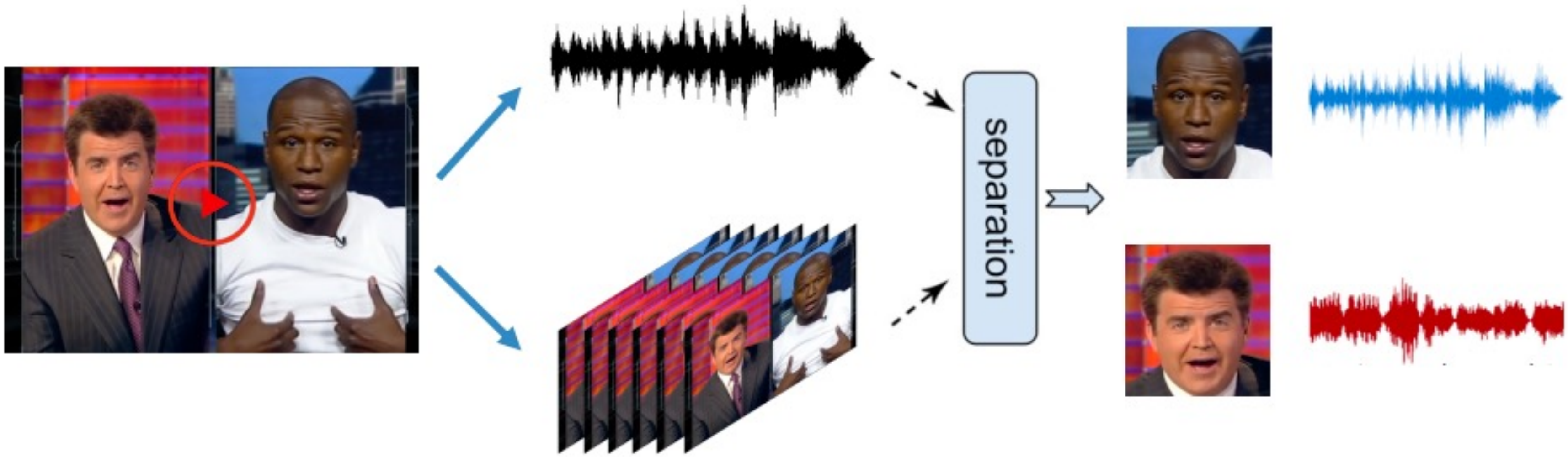
Appearance "Slow" motion "Fast" motion





Slide credit: Justin Johnson

Frontiers: Visually-guided audio source separation



[Gao et al. ECCV 2018, Afouras et al. Interspeech'18, Gabby et al. Interspeech'18, Owens & Efros ECCV'18, Ephrat et al. SIGGRAPH'18, Zhao et al. ECCV 2018, Gao & Grauman ICCV 2019, Zhao et al. ICCV 2019, Xu et al. ICCV 2019, Gan et al. CVPR 2020, Gao et al. CVPR 2021]

Slide credit: Fei-Fei Li

Frontiers: Musical instruments source separation

Train on 100,000 unlabeled multi-source video clips, then separate audio for novel video.

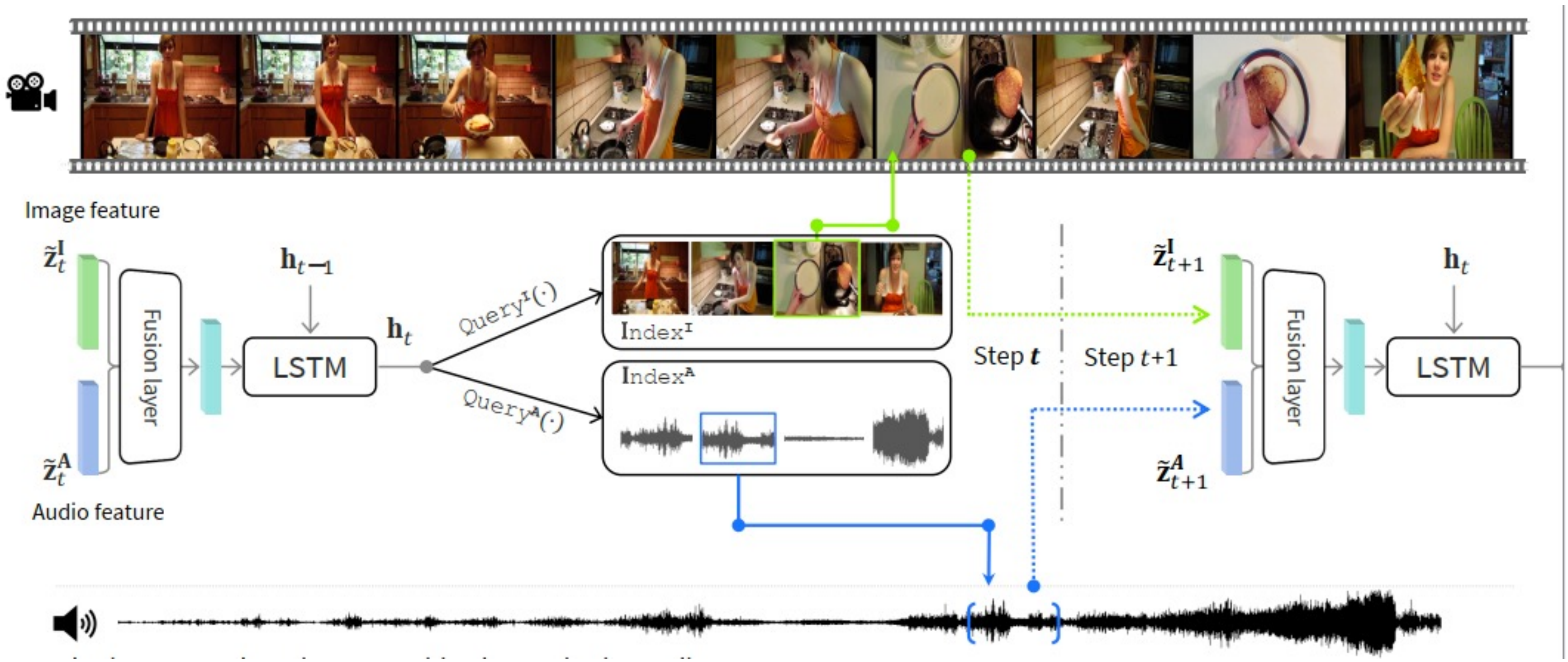
Gao & Grauman, Co-Separating Sounds of Visual Objects, ICCV 2019

original video
(before separation)

object detections:
violin & flute

Slide credit: Fei-Fei Li

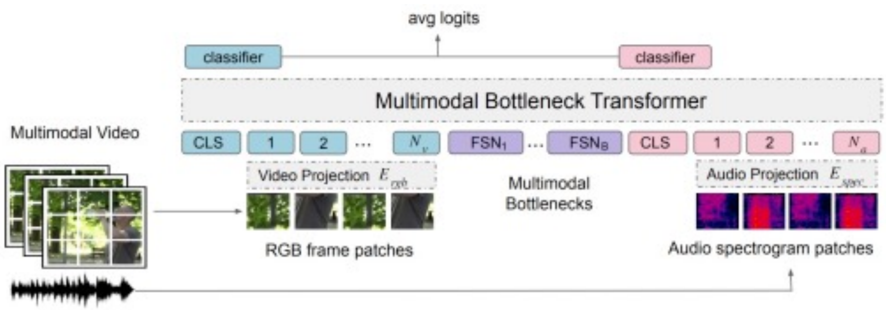
Frontiers: Audio as a preview mechanism for efficient action recognition in untrimmed videos



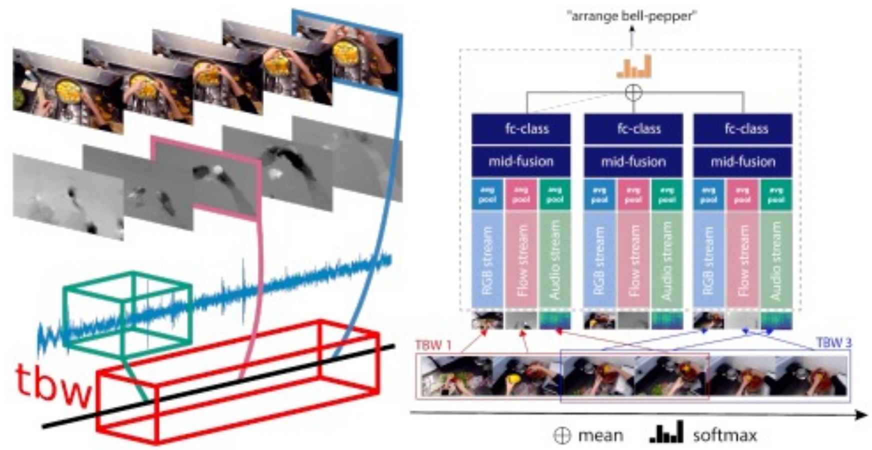
Gao et al., Listen to Look: Action Recognition by Previewing Audio, CVPR 2020

Slide credit: Fei-Fei Li

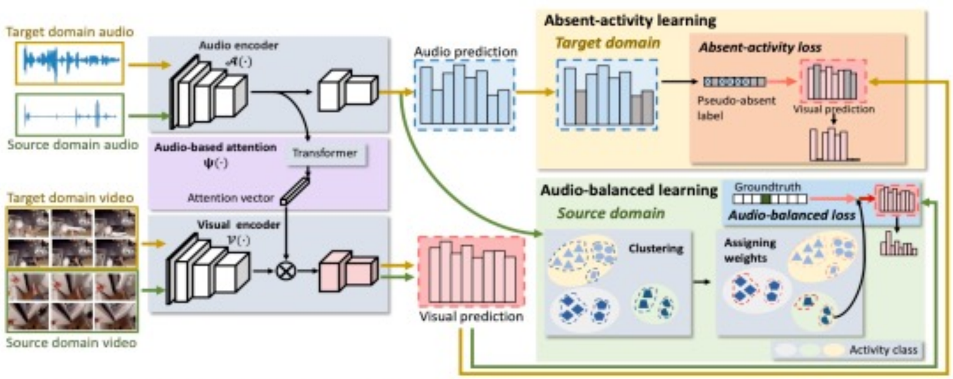
Frontiers: Multimodal Video Understanding



Attention Bottlenecks for Multimodal Fusion, Nagrani et al. NeurIPS 2021



EPIC-Fusion: Audio-Visual Temporal Binding for Egocentric Action Recognition, Kazakos et al., ICCV 2019



Audio-Adaptive Activity Recognition Across Video Domains, Yunhua et al. CVPR 2022

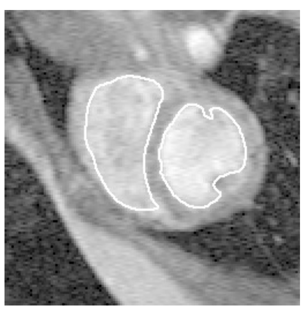
Slide credit: Fei-Fei Li

Tracking: some applications

Body pose tracking, activity recognition

Censusing a bat population

Video-based interfaces



Medical apps

Surveillance

Kristen Grauman