Trustworthy AI Systems

-- Pretrained Foundation Model

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

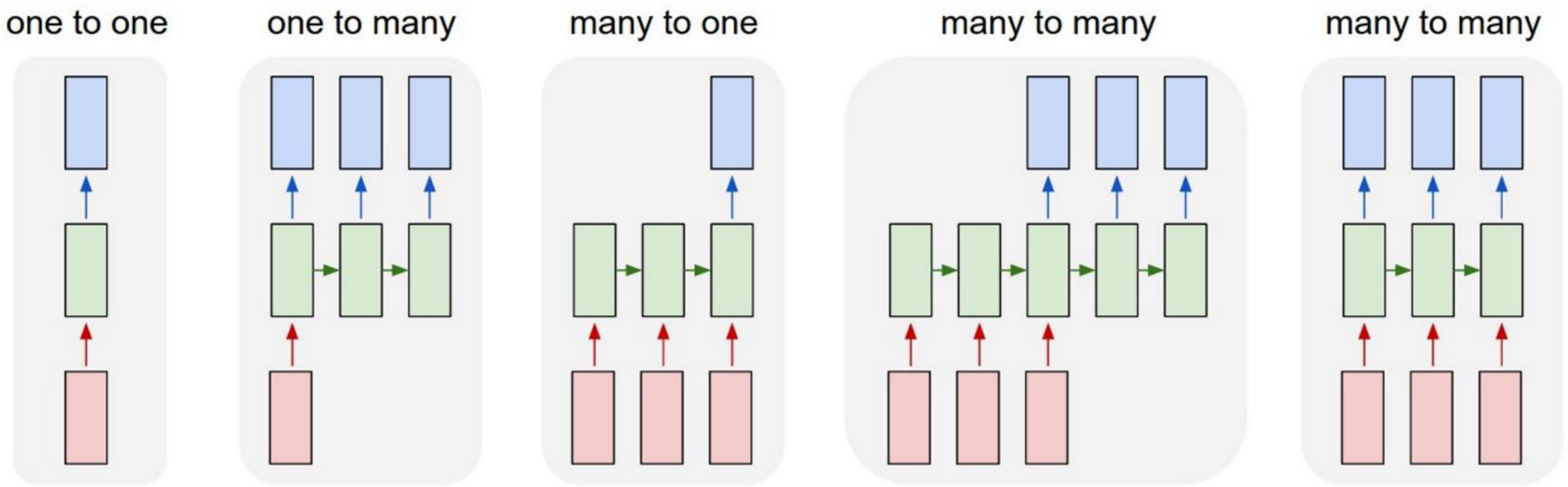
Voice Conversion

- Non-disentangle-based methods
  - Statistics-based methods
  - Generative-based methods
- Disentangle-based methods
  - Instance normalization
  - Quantization

This Lecture

- Recurrent Neural Network
- Attention
- Transformer
- Pretrained Foundation Model

Recurrent Neural Network



Vanilla Neural Networks

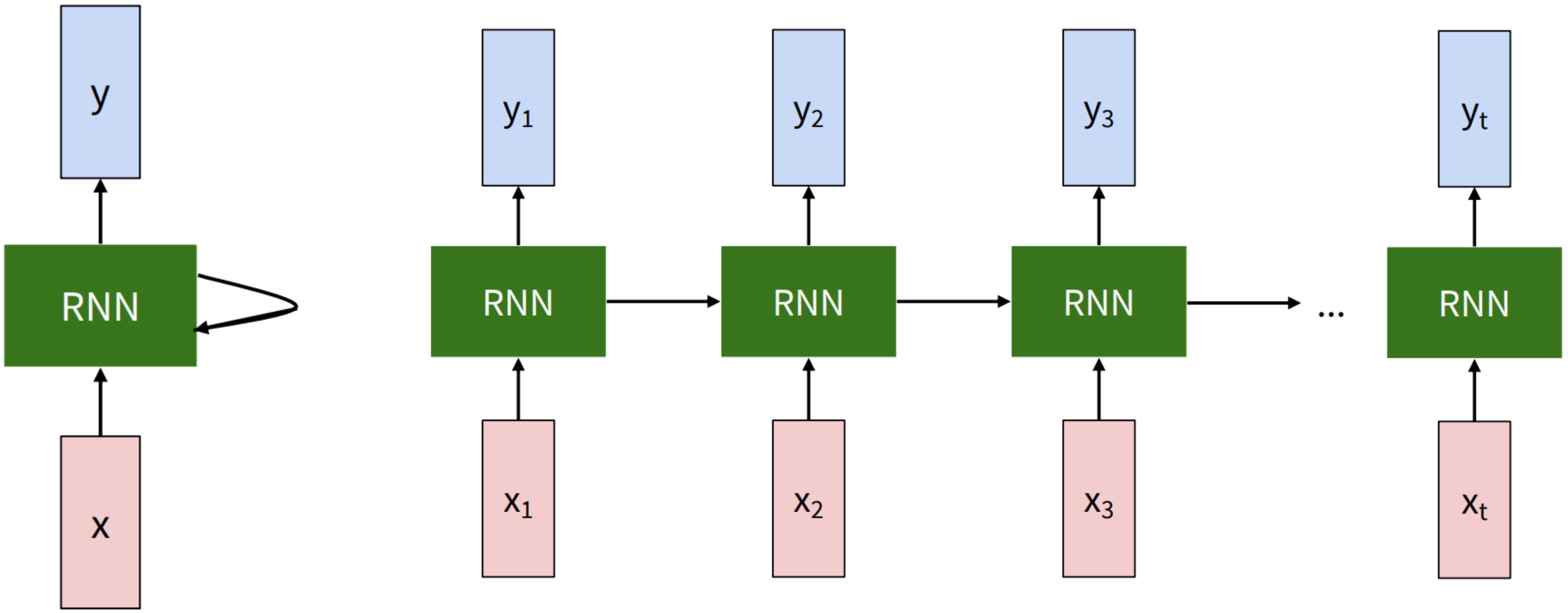
Image Captioning

Action Prediction

Video Captioning

Video classification on frame level

Recurrent Neural Network



RNN Hidden State Update

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

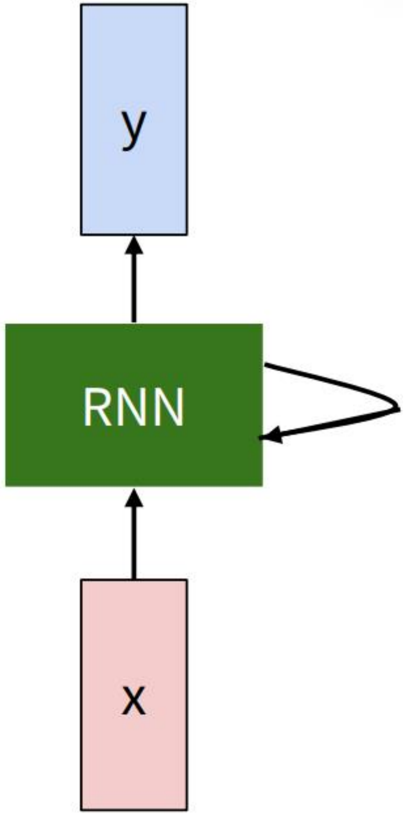
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step



RNN Output Generation

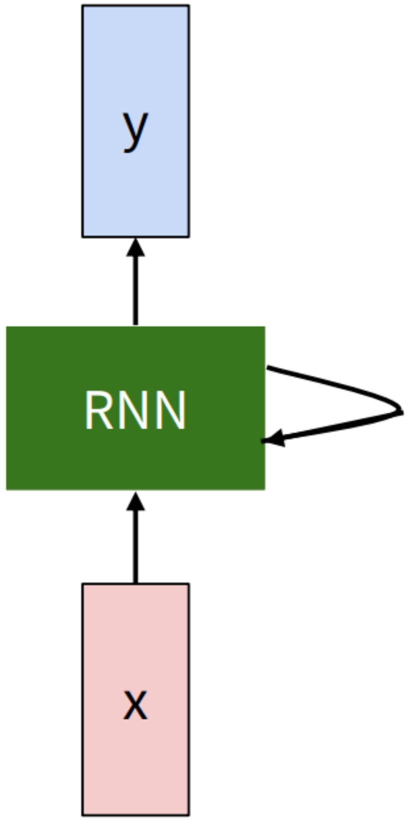
We can process a sequence of vectors x by applying a recurrence formula at every time step:

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

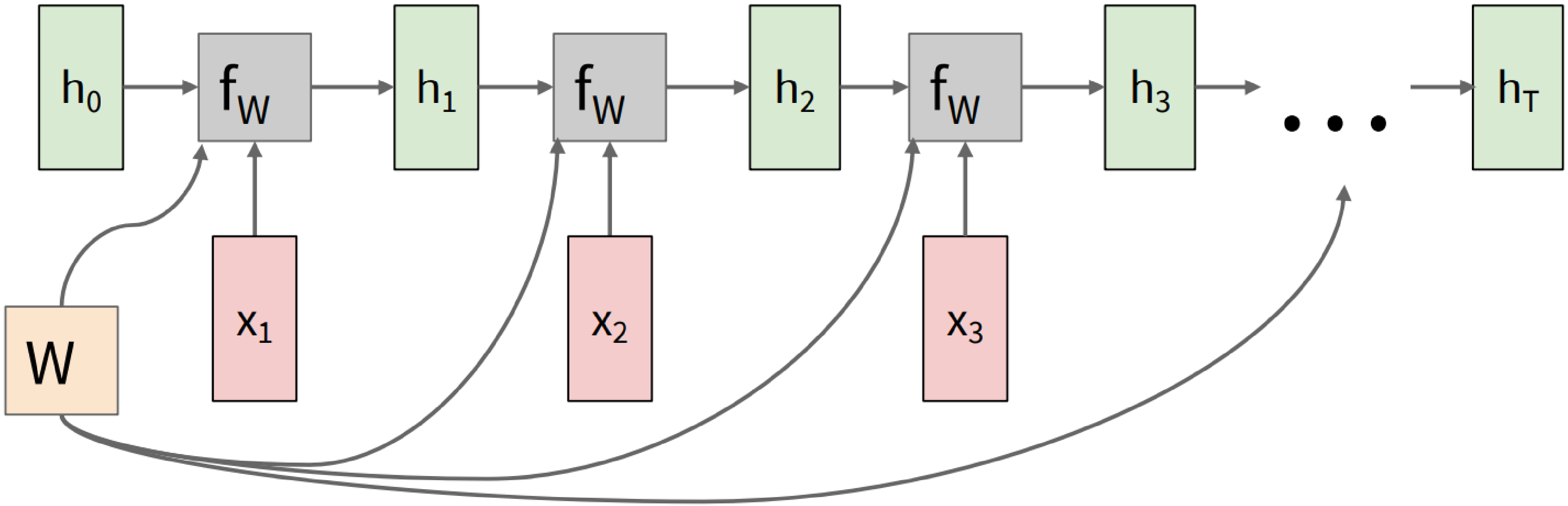
output

new state

another function
with parameters W_{hy}

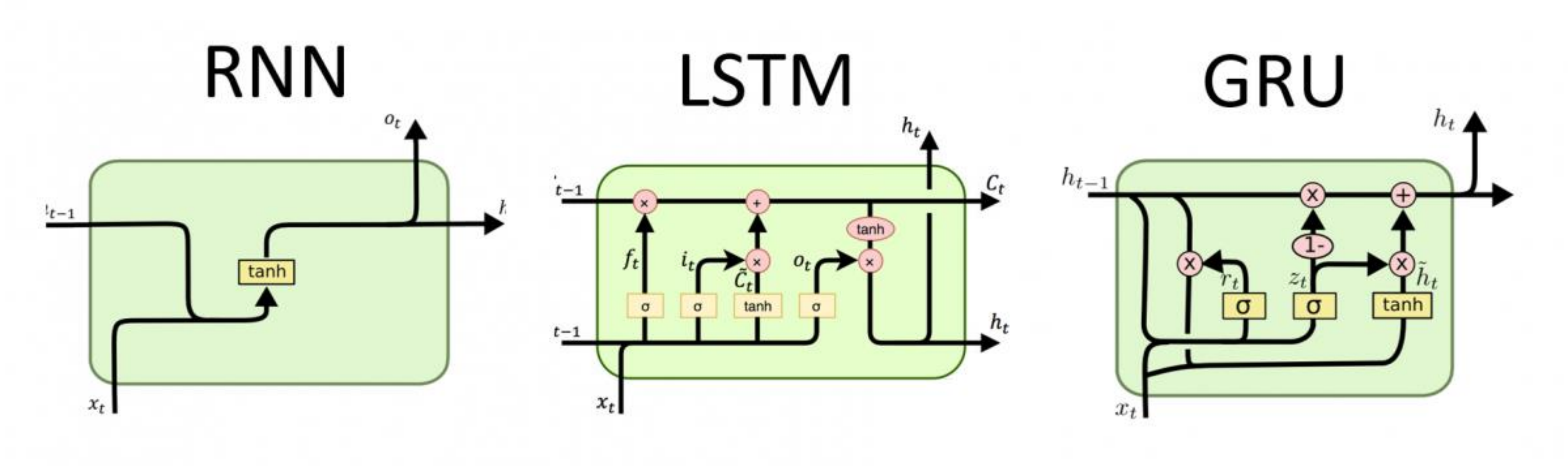


RNN: Computational Graph



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

RNN Variants



http://dprogrammer.org/rnn-lstm-gru

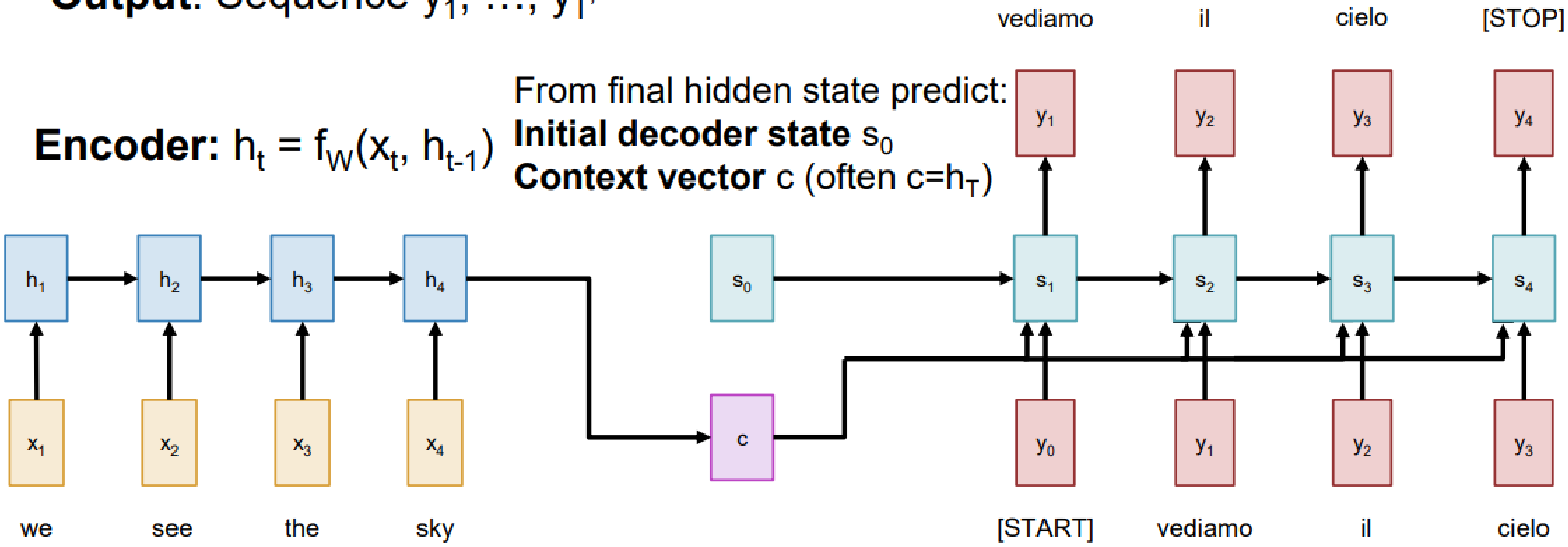
Sequence to Sequence with RNNs

Input: Sequence $x_1, \dots x_T$
Output: Sequence $y_1, \dots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



During training, we use the “correct” token even if the model is wrong.

RNN Tradeoffs

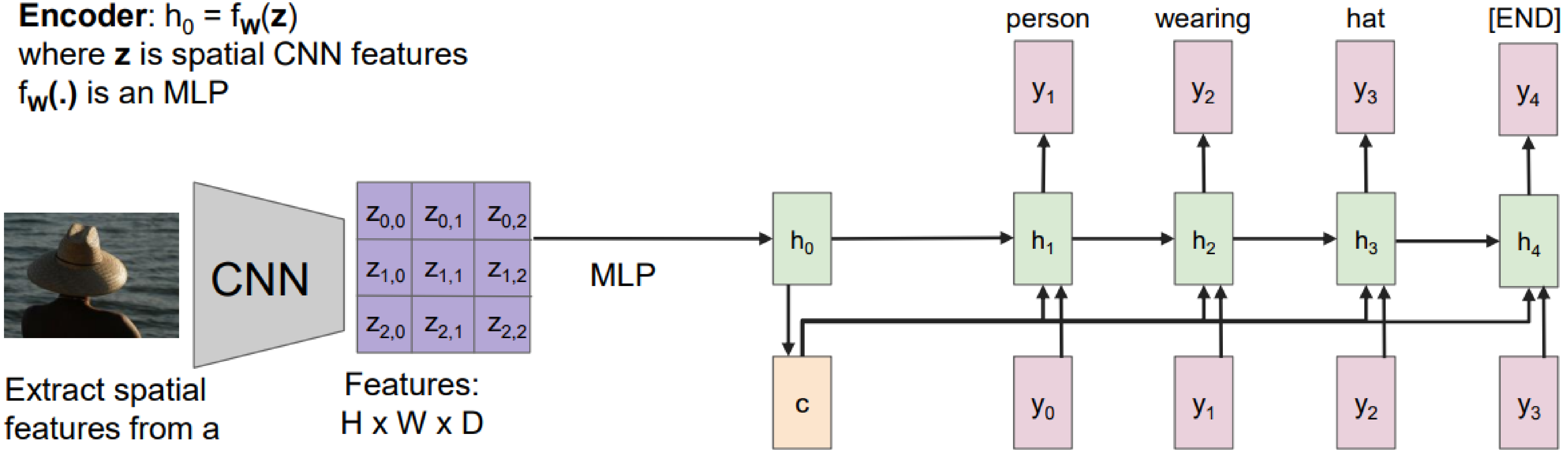
- RNN Advantages:
  - Can process any length of the input
  - Computation for step t can (in theory) use information from many steps back
  - Model size does not increase for longer input
  - The same weights are applied on every timestep, so there is symmetry in how inputs are processed.
- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

Image Captioning using Spatial Features

Input: Image **I**
Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

Decoder: $h_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$
and output $y_t = T(h_t)$

Encoder: $h_0 = f_W(\mathbf{z})$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



Extract spatial features from a pretrained CNN

Features:
H x W x D

This Lecture

- Recurrent Neural Network
- Attention: the relative importance of each component in a sequence
- Transformers
- Pretrained Foundation Model

Image Captioning with RNNs and Attention

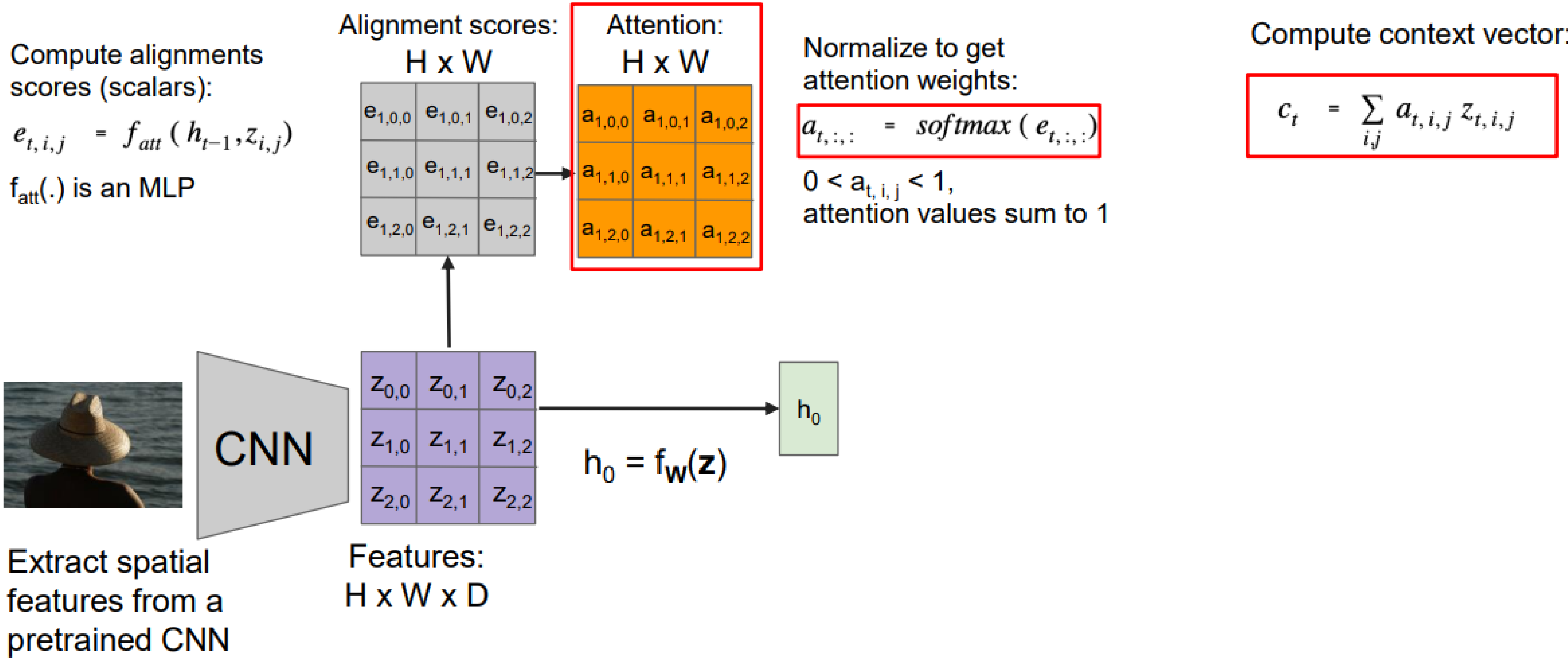


Image Captioning with RNNs and Attention

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

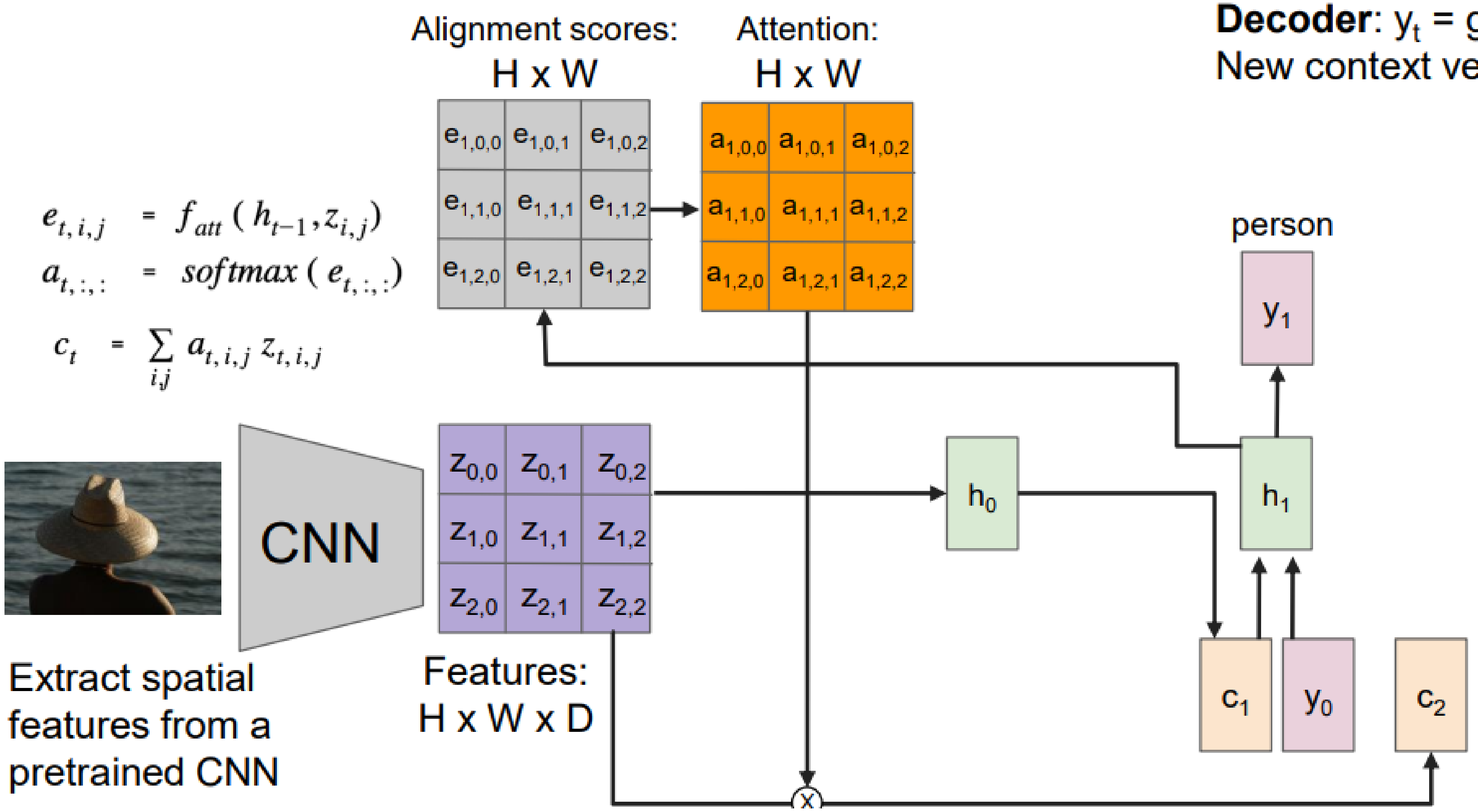
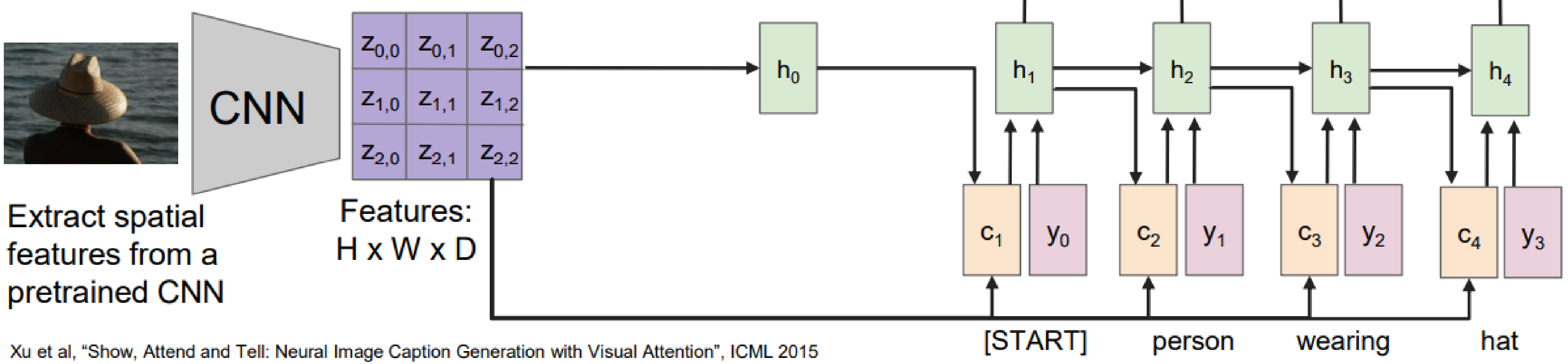


Image Captioning with RNNs and Attention

Each timestep of decoder uses a different context vector that looks at different parts of the input image

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

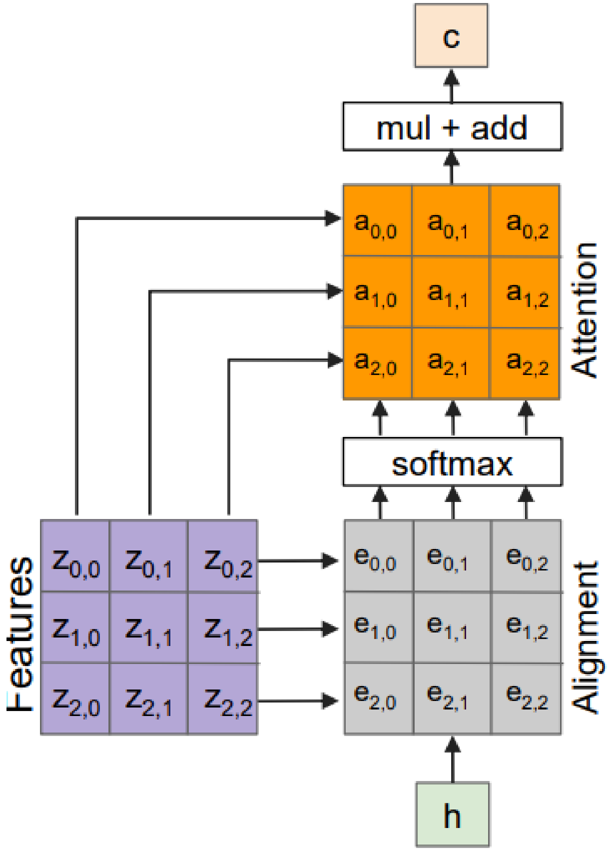


Extract spatial features from a pretrained CNN

Features: H x W x D

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in Image Captioning



Outputs:
context vector: **c** (shape: D)

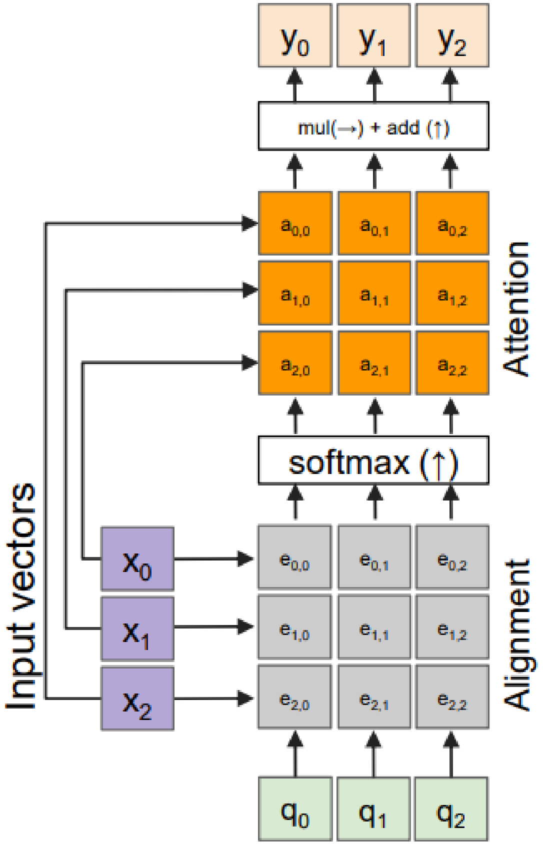
Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_{i,j} a_{i,j} z_{i,j}$

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

“query” refers to a vector used to calculate a corresponding context vector.

General Attention Layer (1)

Each query creates a new, corresponding output context vector



Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

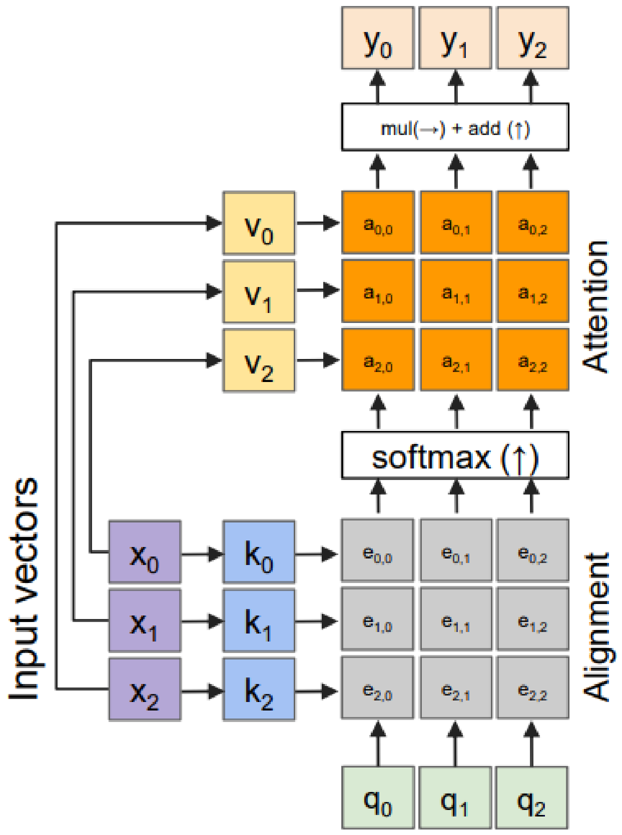
Inputs:
Input vectors: **x** (shape: N x D) Attention operation is permutation invariant, so reshape.
Queries: **q** (shape: M x D)

Multiple query vectors

Change $f_{att}(.)$ to a scaled simple dot product

- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by √D to reduce effect of large magnitude vectors
- Similar to Xavier and Kaiming Initialization!

General Attention Layer (2)



Outputs:
context vectors: **y** (shape: $\boxed{D_v}$)

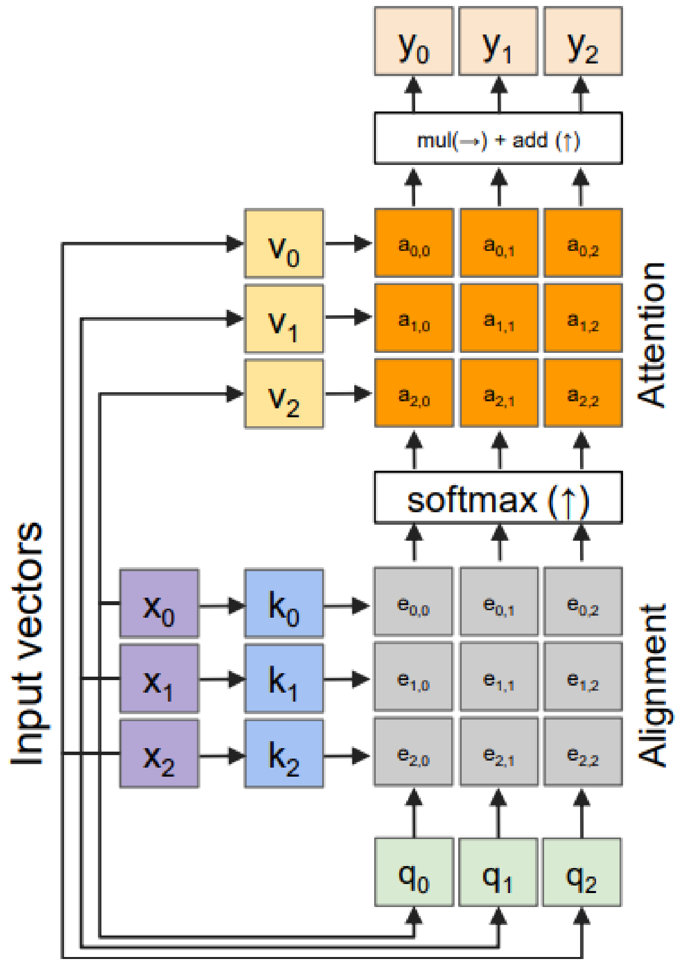
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x $\boxed{D_k}$)

We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Self-attention Layer

Permutation equivariant: Self-attention layer doesn't care about the orders of the inputs!



Outputs:
context vectors: $\mathbf{y}$ (shape: D_v)

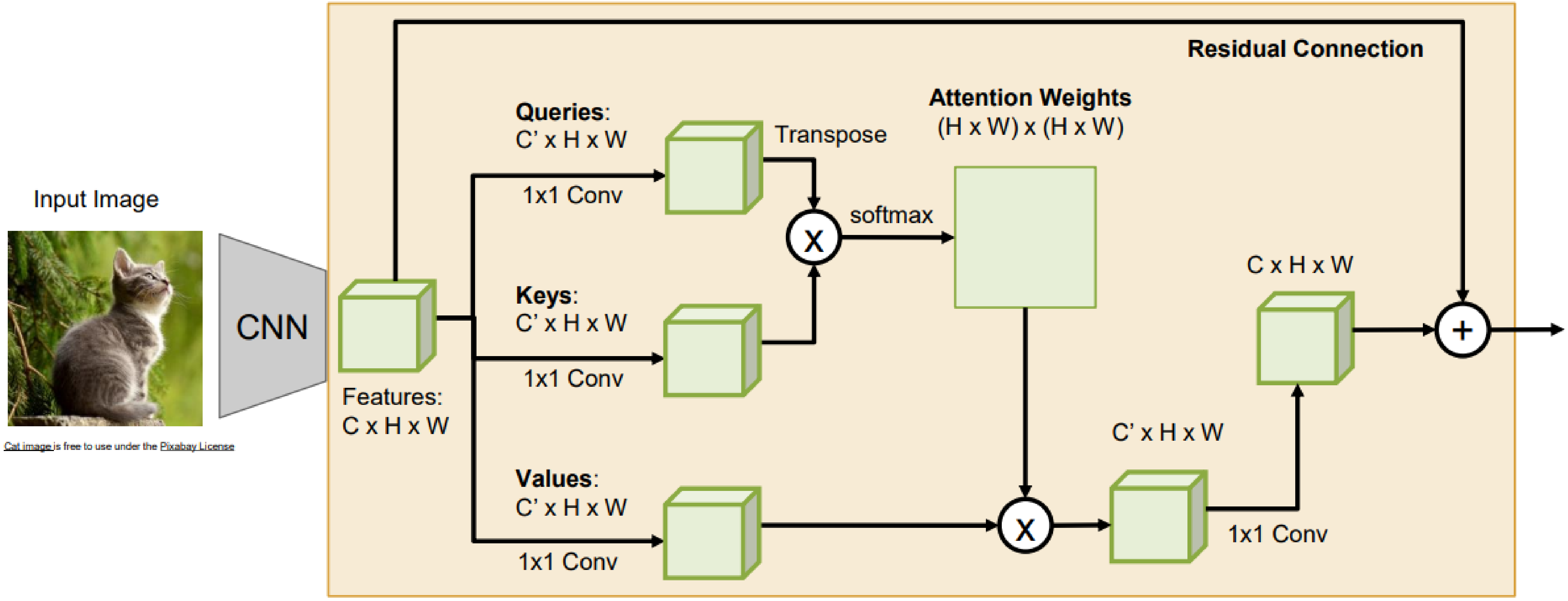
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: $\mathbf{x}$ (shape: N x D)

We can **calculate the query vectors from the input vectors**, therefore, defining a "self-attention" layer.

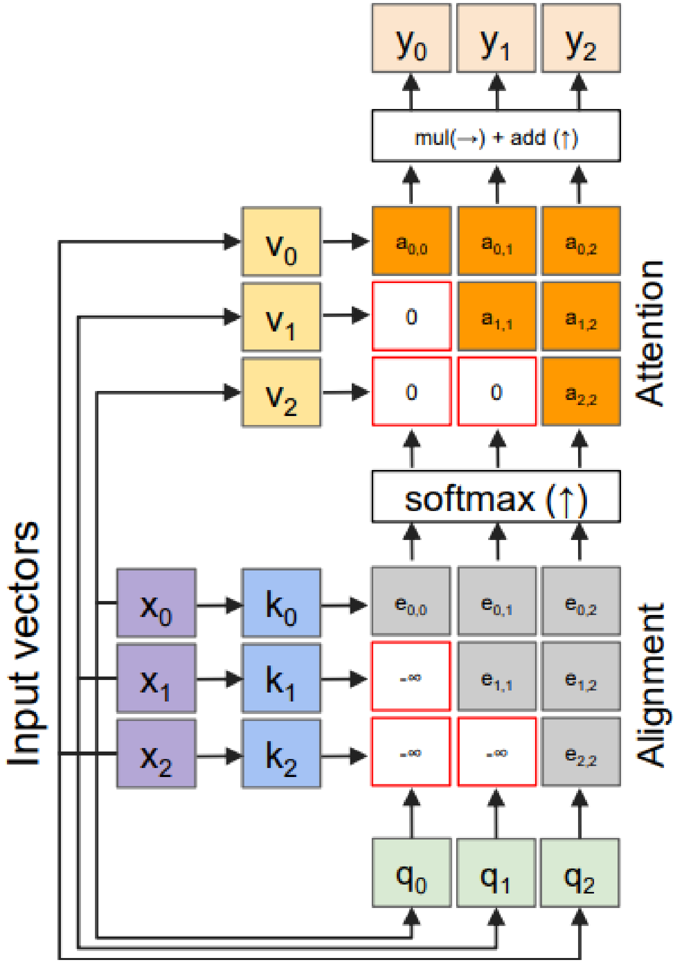
No input query vectors anymore

CNN with Self-Attention



Self-Attention Module

Masked self-attention layer



Outputs:
context vectors: **y** (shape: D_v)

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

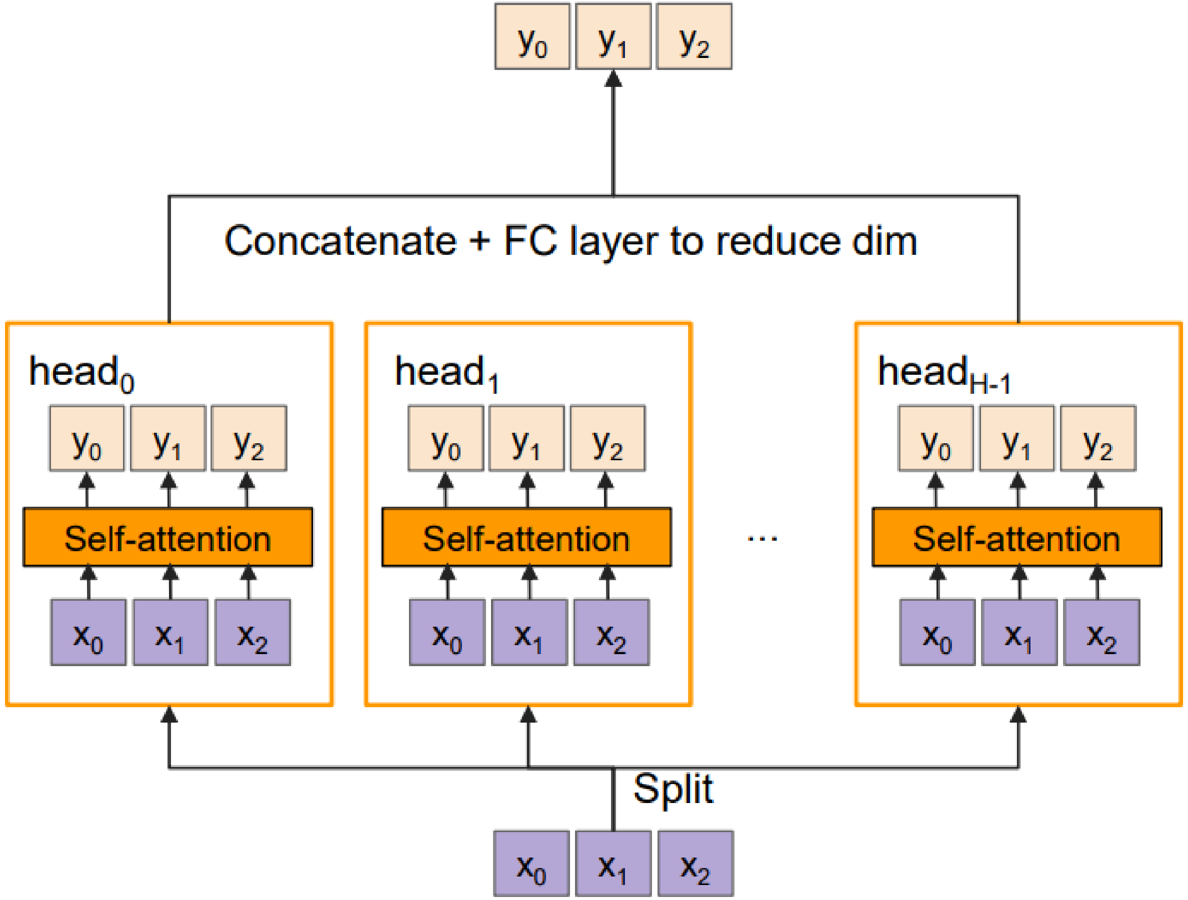
Inputs:
Input vectors: **x** (shape: N x D)

- Allows us to parallelize attention across time
- Don't need to calculate the context vectors from the previous timestep first!

- Prevent vectors from looking at future vectors.
- Manually set alignment scores to –infinity (-nan)

Multi-head self-attention layer

- Multiple self-attention "heads" in parallel

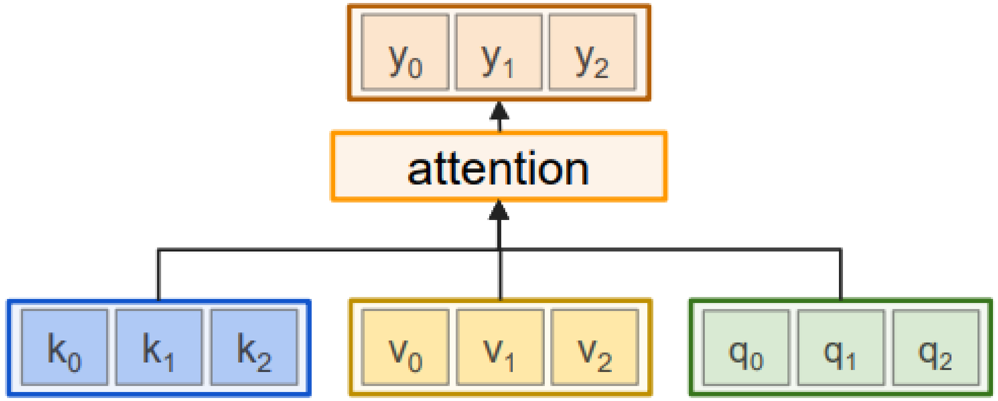


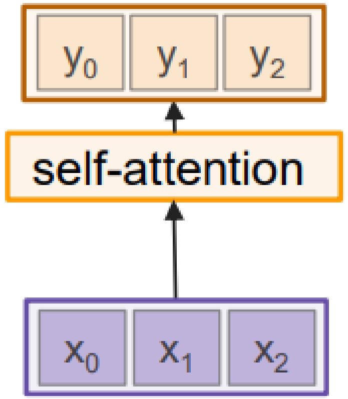
Why multi-head?

A: We may want to have multiple sets of queries/keys/values calculated in the layer. This is a similar idea to having multiple conv filters learned in a layer

General attention versus self-attention

Transformer models rely on many, stacked self-attention layers

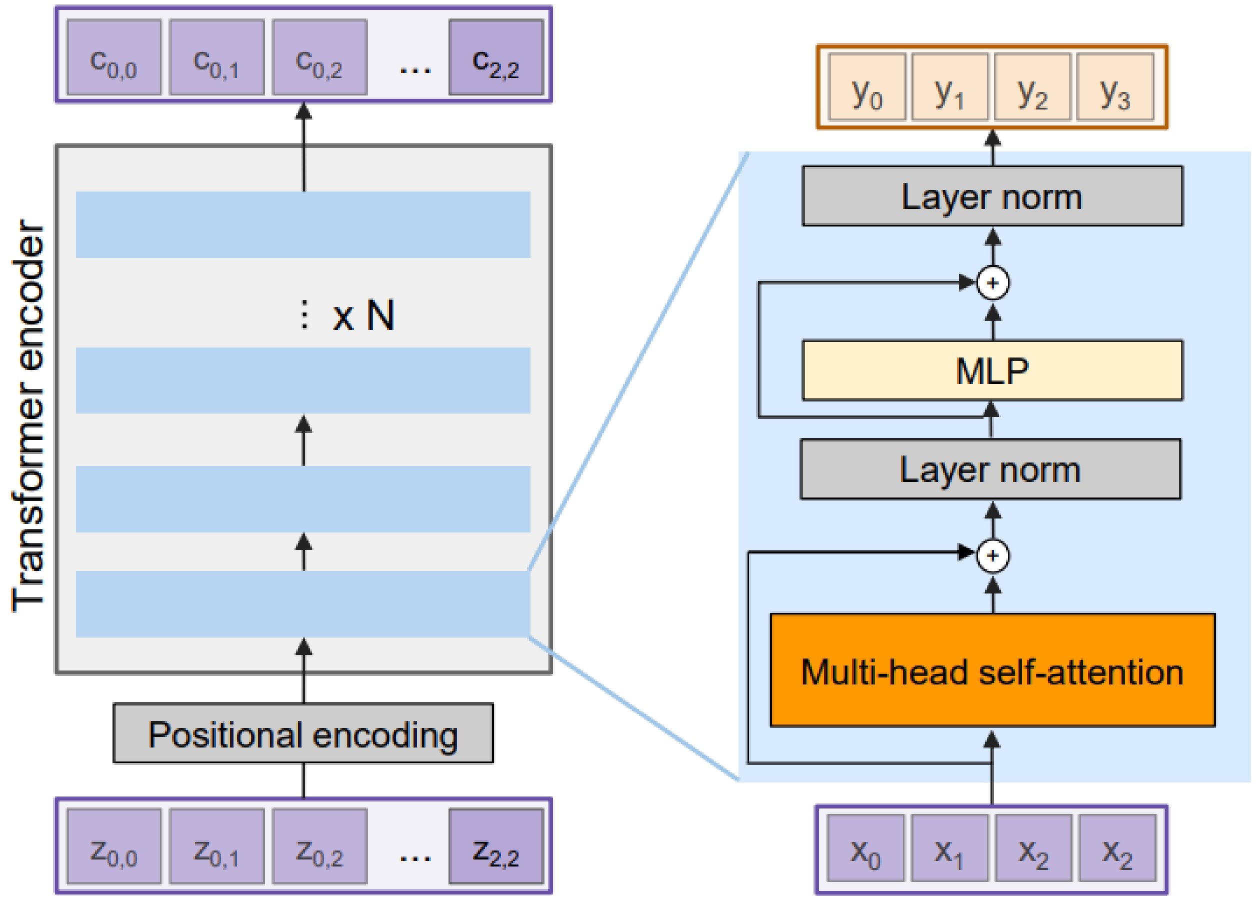




This Lecture

- Recurrent Neural Network
- Attention
- Transformer
- Pretrained Foundation Model

The Transformer encoder block



Transformer Encoder Block:

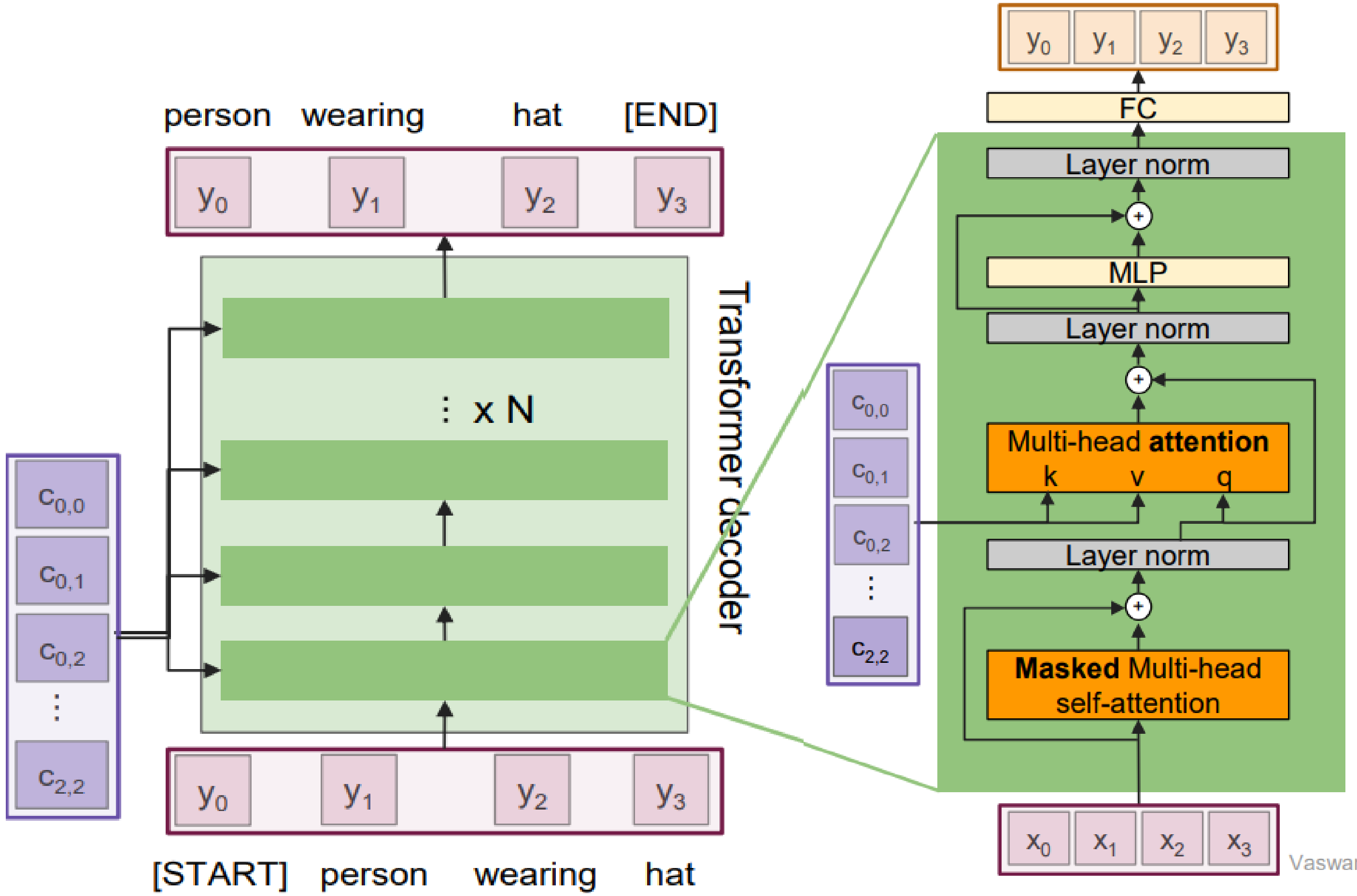
Inputs: Set of vectors **x**
Outputs: Set of vectors **y**

Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

The Transformer decoder block



Vaswani et al, "Attention is all you need", NeurIPS 2017

Transformer Decoder Block:

Inputs: Set of vectors **x** and Set of context vectors **c.**
Outputs: Set of vectors **y.**

Masked Self-attention only interacts with past inputs.

Multi-head attention block is NOT self-attention. It attends over encoder outputs.

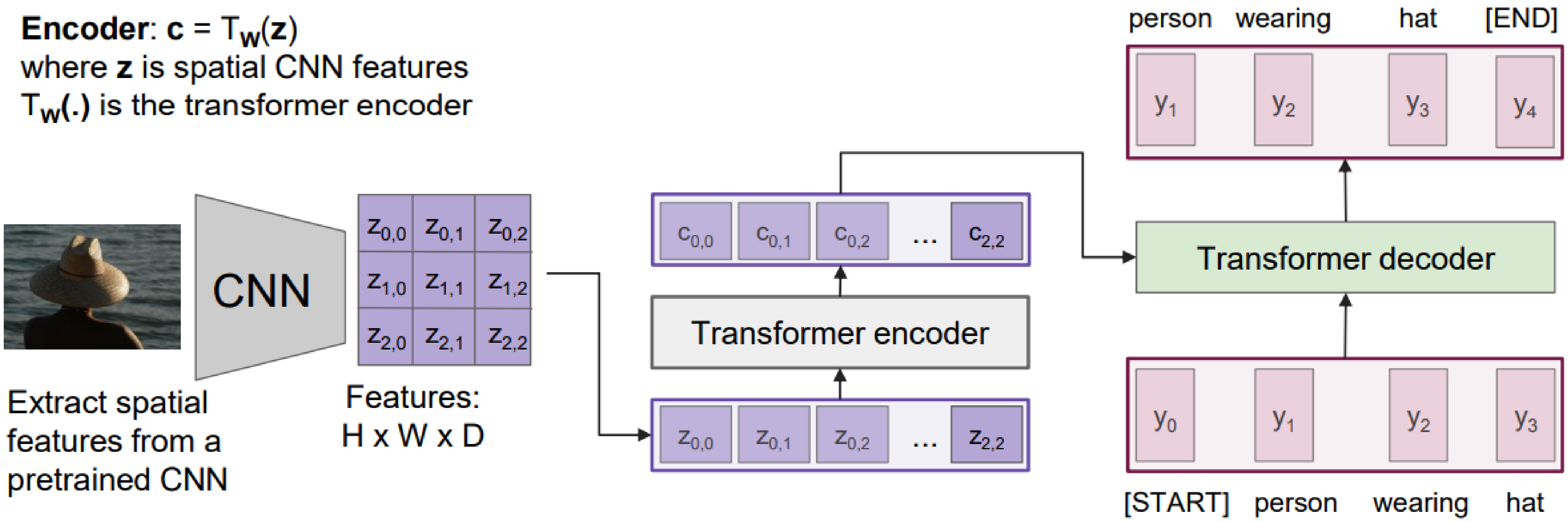
Highly scalable, highly parallelizable, but high memory usage.

Image Captioning using Transformers

Input: Image **I**
Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

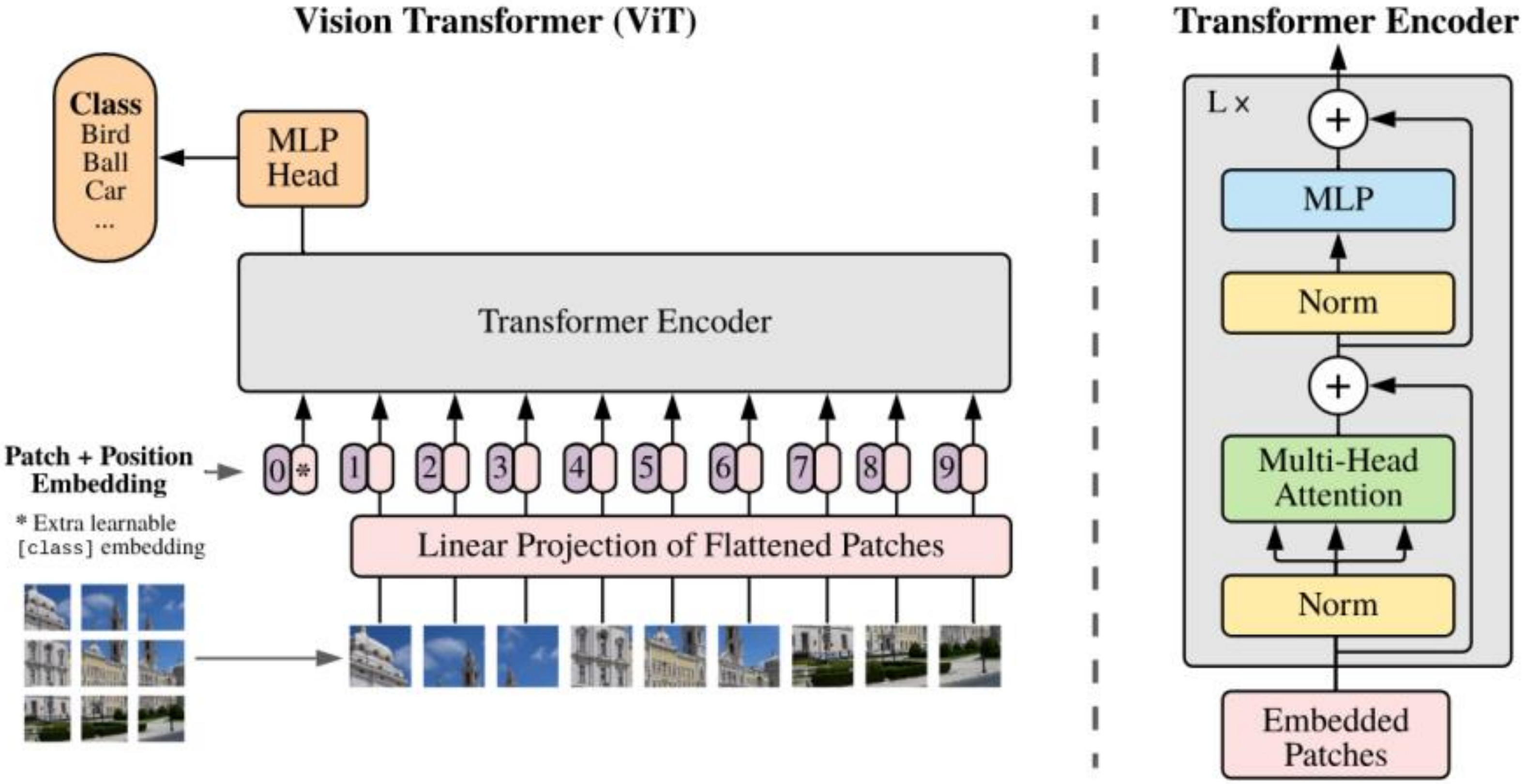
Decoder: $y_t = T_D(\mathbf{y}_{0:t-1}, \mathbf{c})$
where $T_D(.)$ is the transformer decoder

Encoder: $\mathbf{c} = T_W(\mathbf{z})$
where $\mathbf{z}$ is spatial CNN features
$T_W(.)$ is the transformer encoder



ViTs – Vision Transformers

- Transformers from pixels to language



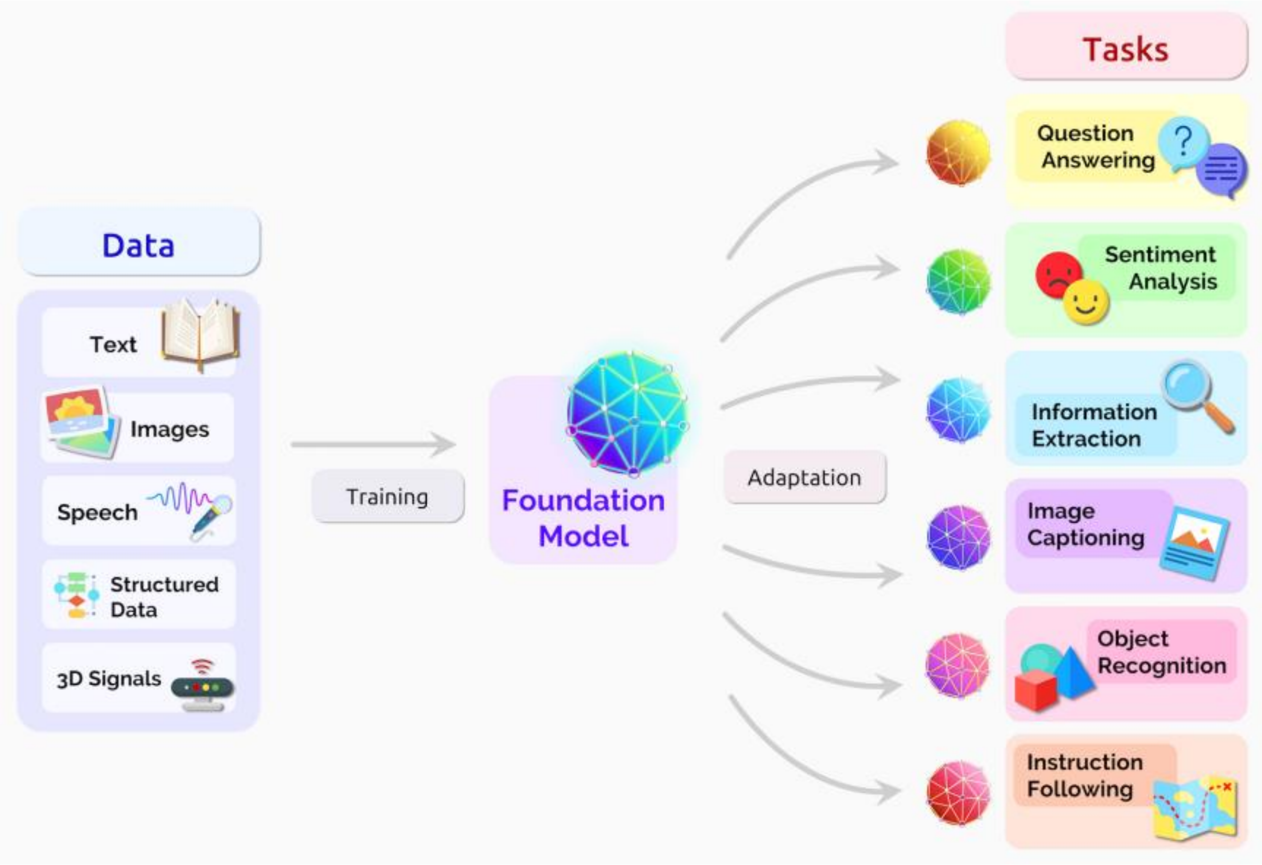
DeepSeek

https://www.youtube.com/watch?v=KTonvXhsxpc

This Lecture

- Recurrent Neural Network
- Attention
- Transformers
- Pretrained Foundation Model

Foundation Models in Different Modalities

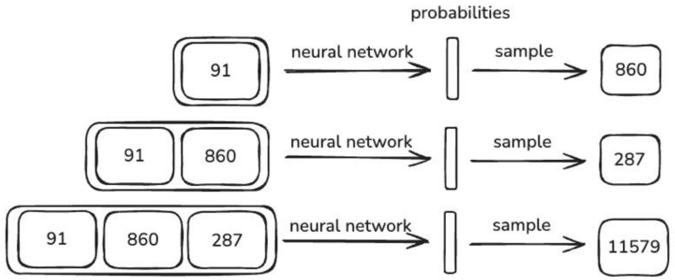


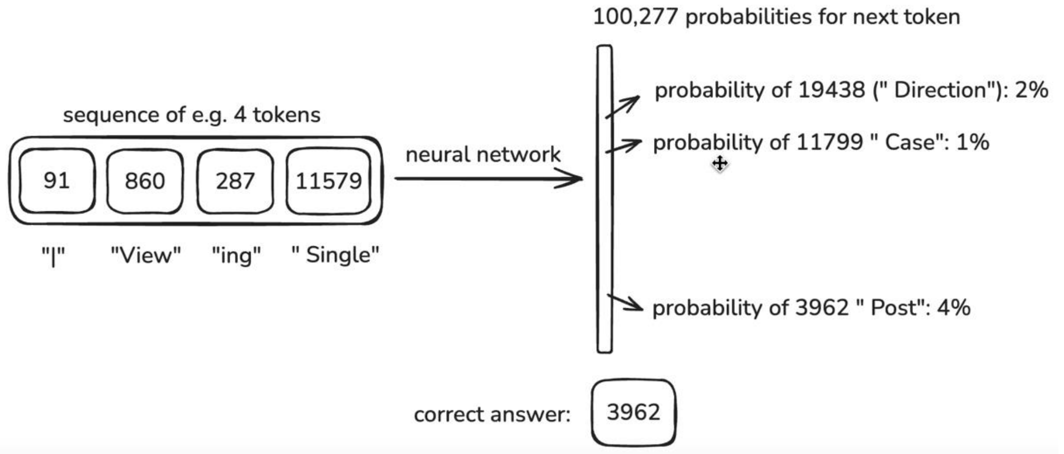
https://arxiv.org/pdf/2108.07258

- Foundation model is trained on large amounts of unlabeled/self-supervised data.
- A foundation model can centralize the information from all the data from various modalities.
- This one model can then be adapted to a wide range of downstream tasks.

Examples for Pre-training

- Download and preprocess the Internet
  - https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1
  - https://huggingface.co/datasets/HuggingFaceFW/fineweb
- Tokenization (GPT-4: 100,277 possible tokens)
  - https://tiktokenizer.vercel.app/
- Neural Network Training
  - https://bbycroft.net/llm
- Inference

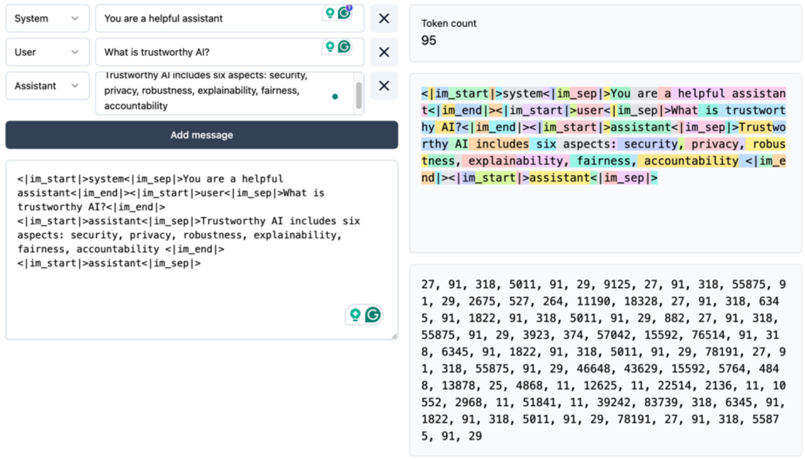


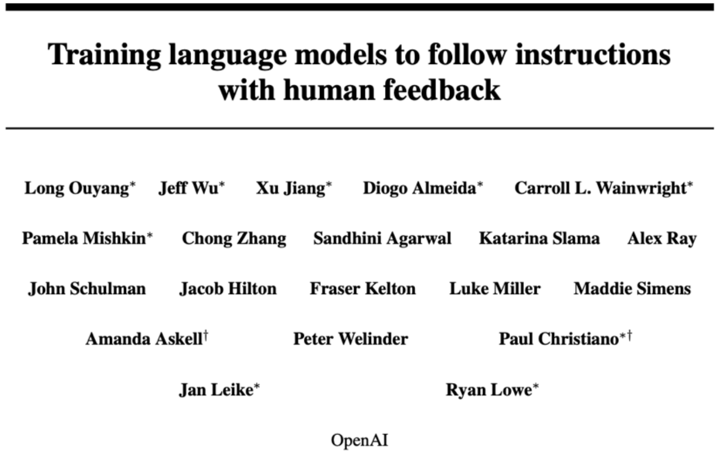


Source: https://www.youtube.com/watch?v=7xTGNNLPyMI

Examples for Post-training

- Base model: internet document simulator, stochastic, probabilistic
- ➔ Human conversation training
  - https://huggingface.co/datasets/OpenAssistant/oasst1
  - https://github.com/thunlp/UltraChat/tree/main





Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*
Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray
John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens
Amanda Askell† Peter Welinder Paul Christiano*†
Jan Leike* Ryan Lowe*
OpenAI

https://arxiv.org/pdf/2203.02155

https://huggingface.co/spaces/huggingface/inference-playground

Other Foundation Model Designs in NLP

2020	ACL	CamemBERT [87]	Transformer Encoder	Contextual	MLM(WWM)	https://camembert-model.fr
2020	ACL	XLM-R [88]	Transformer Encoder	Contextual	MLM	https://github.com/.../XLM
2020	ICLR	Reformer [89]	Reformer	Permutation	-	https://github.com/.../reformer
2020	ICLR	ELECTRA [46]	Transformer Encoder	Contextual	MLM	https://github.com/.../electra
2020	AAAI	Q-BERT [90]	Transformer Encoder	Contextual	MLM	-
2020	AAAI	XNLG [91]	Transformer	Contextual	MLM+DAE	https://github.com/.../xnlg
2020	AAAI	K-BERT [92]	Transformer Encoder	Contextual	MLM	https://github.com/.../K-BERT
2020	AAAI	ERNIE 2.0 [62]	Transformer Encoder	Contextual	MLM	https://github.com/.../ERNIE
2020	NeurIPS	GPT-3 [20]	Transformer Decoder	Autoregressive	LM	https://github.com/.../gpt-3
2020	NeurIPS	MPNet [57]	Transformer Encoder	Permutation	MLM+PLM	https://github.com/.../MPNet
2020	NeurIPS	ConvBERT [93]	Mixed Attention	Contextual	-	https://github.com/.../ConvBert
2020	NeurIPS	MiniLM [94]	Transformer Encoder	Contextual	MLM	https://github.com/.../minilm
2020	TACL	mBART [95]	Transformer	Contextual	DAE	https://github.com/.../mbart
2020	COLING	CoLAKE [96]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../CoLAKE
2020	LREC	FlauBERT [97]	Transformer Encoder	Contextual	MLM	https://github.com/.../Flaubert
2020	EMNLP	GLM [98]	Transformer Encoder	Contextual	MLM+KG	https://github.com/.../GLM
2020	EMNLP (Findings)	TinyBERT [99]	Transformer	Contextual	MLM	https://github.com/.../TinyBERT
2020	EMNLP (Findings)	RobBERT [100]	Transformer Encoder	Contextual	MLM	https://github.com/.../RobBERT
2020	EMNLP (Findings)	ZEN [64]	Transformer Encoder	Contextual	MLM	https://github.com/.../ZEN
2020	EMNLP (Findings)	BERT-MK [101]	KG-Transformer Encoder	Contextual	MLM	-
2020	RepL4NLP@ACL	CompressingBERT [35]	Transformer Encoder	Contextual	MLM(Pruning)	https://github.com/.../bert-prune
2020	JMLR	T5 [102]	Transformer	Contextual	MLM(Seq2Seq)	https://github.com/...transformer
2021	T-ASL	BERT-wwm-Chinese [63]	Transformer Encoder	Contextual	MLM	https://github.com/...BERT-wwm
2021	EACL	PET [103]	Transformer Encoder	Contextual	MLM	https://github.com/.../pet
2021	TACL	KEPLER [104]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../KEPLER
2021	EMNLP	SimCSE [105]	Transformer Encoder	Contextual	MLM+KE	https://github.com/.../SimCSE
2021	ICML	GLaM [106]	Transformer	Autoregressive	LM	-
2021	arXiv	XLM-E [107]	Transformer	Contextual	MLM	
2021	arXiv	T0 [108]	Transformer	Contextual	MLM	https://github.com/.../T0
2021	arXiv	Gopher [109]	Transformer	Autoregressive	LM	-
2022	arXiv	MT-NLG [110]	Transformer	Contextual	MLM	-
2022	arXiv	LaMDA [67]	Transformer Decoder	Autoregressive	LM	https://github.com/.../LaMDA
2022	arXiv	Chinchilla [111]	Transformer	Autoregressive	LM	-
2022	arXiv	PaLM [43]	Transformer	Autoregressive	LM	https://github.com/.../PaLM
2022	arXiv	OPT [112]	Transformer Decoder	Autoregressive	LM	https://github.com/.../MetaSeq

https://arxiv.org/pdf/2302.09419

Other Foundation Model Designs in NLP

- Encoder-only: BERT
  - Bidirectional attention, low rank attention matrix
  - masked language modeling
  - understanding

- Encoder-Decoder: T5, BART
  - Large amount of parameters, hard to train

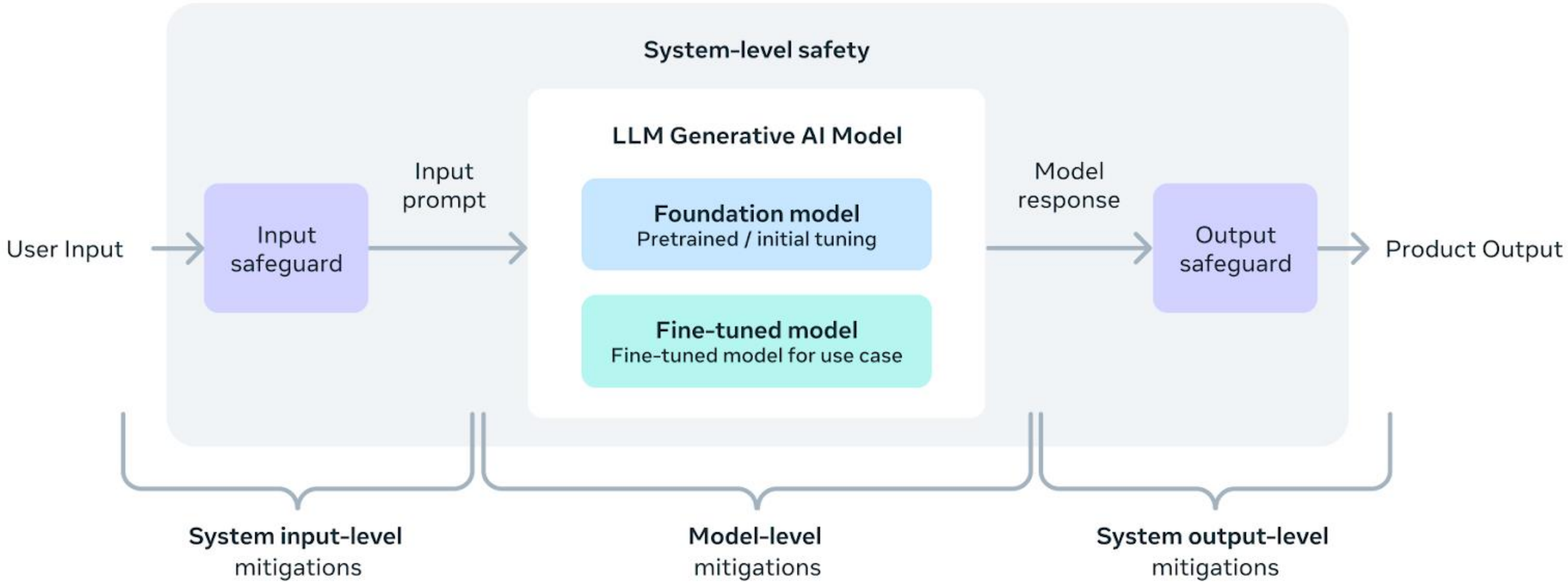
- Decoder-only: GPT
  - Next token prediction
  - Full rank attention matrix (e.g., unique solution, invertibility, represent richer feature space, capture diverse range of relationship within th data)
  - Understanding and generation
  - High zero-shot/few-shot generalization

Llama 3: Openly Available LLM to Date

- Llama 3 uses a tokenizer with a vocabulary of 128K tokens that encodes language much more efficiently, which leads to substantially improved model performance.
- Llama 3 is pretrained on over 15T tokens that were all collected from publicly available sources.
- The training runs on two custom-built 24K GPU clusters.
- Instruction fine-tuning: post-training is a combination of supervised fine-tuning (SFT), rejection sampling, proximal policy optimization (PPO), and direct preference optimization (DPO).

https://github.com/meta-llama/llama3

The Safety Measures of LLM (Llama)



Instruction-fine-tuned models have been red-teamed (tested) for safety through internal and external efforts.
The red teaming approach leverages human experts and automation methods to generate adversarial prompts that try to elicit problematic responses.

References

- https://cs231n.stanford.edu/slides/2024/lecture_7.pdf
- https://cs231n.stanford.edu/slides/2024/lecture_8.pdf
- On the Opportunities and Risks of Foundation Models
- A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT