COMP547

DEEP UNSUPERVISED LEARNING

Lecture #4 – Attention and Transformers

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2024

Good news, everyone!

- Paper list for the paper presentations is out!

Previously on COMP547

- sequence modeling
- recurrent neural networks (RNNs)
- how to train RNNs
- long short-term memory (LSTM)
- gated recurrent unit (GRU)
- sequence to sequence modeling

Lecture overview

- memory
- tokens
- attention
- positional encoding
- transformers vs. attention in RNNs

Disclaimer: Much of the material and slides for this lecture were borrowed from

—Phillip Isola and Sara Beery's MIT 6.S898 slides

Recap: Memory in RNNs

Why not remember everything?

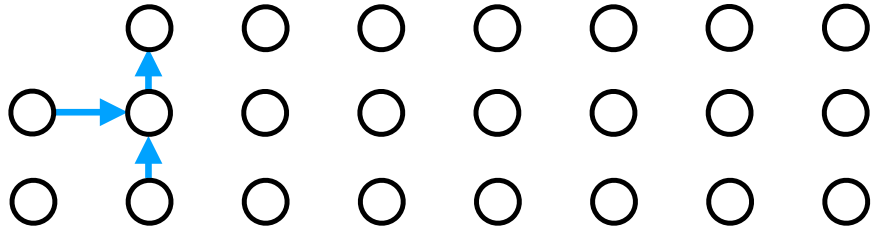
- Memory size grows with t
- This kind of memory is **nonparametric**: there is no finite set of parameters we can use to model it
- RNNs make a Markov assumption — the future hidden state only depends on the immediately preceding hidden state
- By putting the right info into the hidden state, RNNs can model dependencies that are arbitrarily far apart

The problem of long-range dependences

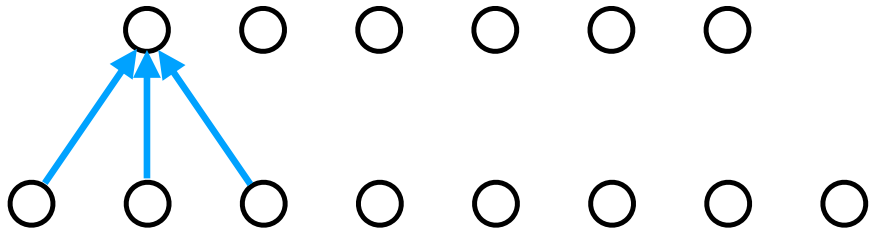
- Other methods exist that do directly link old "memories" (observations or hidden states) to future predictions:
- Temporal convolutions
- Attention / Transformers (see https://arxiv.org/abs/1706.03762)
- Memory networks (see https://arxiv.org/abs/1410.3916)

Modeling arbitrarily long sequences

- **Recurrence** — recurrent weights are shared across time



- **Convolution** — conv weights are shared across time



- **Attention** — weights are dynamically determined as a function of the data
  (conv kernel with attention weights is shown on the right)

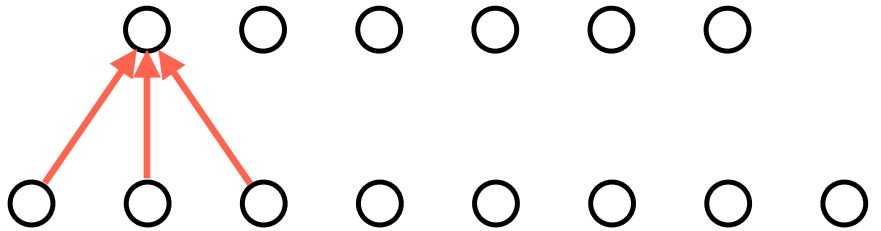


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

[Attention is All you Need, https://arxiv.org/abs/1706.03762]

Transformers

- Three key ideas
  - Tokens
  - Attention
  - Positional encoding
- Examples of architectures and applications

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	GFLOPs	Extra Training Data	Paper	Code	Result	Year	Tags
1	**CoCa** (finetuned)	91.0%		2100M		✓	CoCa: Contrastive Captioners are Image-Text Foundation Models			2022	Transformer, JFT-3B, ALIGN
2	**Model soups** (BASIC-L)	90.98%		2440M		✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	Conv+Transformer, JFT-3B, ALIGN
3	**Model soups** (ViT-G/14)	90.94%		1843M		✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing			2022	Transformer, JFT-3B

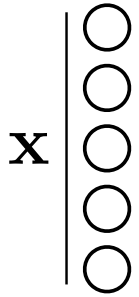
https://paperswithcode.com/sota/image-classification-on-imagenet

Idea #1: tokens

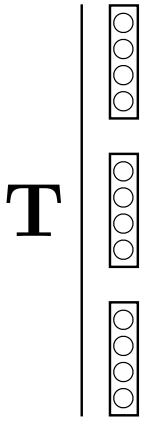
A new data structure: Tokens

- A **token** is just transformer lingo for a vector of neurons (note: GNNs also operate over tokens, but over there we called them "node attributes" or node "feature descriptors")
- 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**



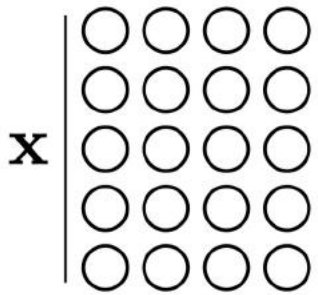
array of **tokens**



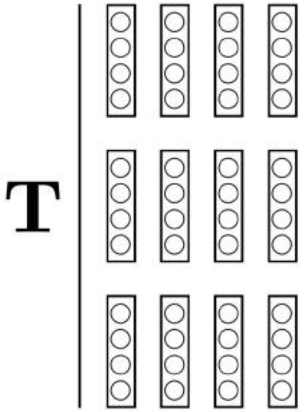
A new data structure: Tokens

- A **token** is just transformer lingo for a vector of neurons (note: GNNs also operate over tokens, but over there we called them "node attributes" or node "feature descriptors")
- 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**



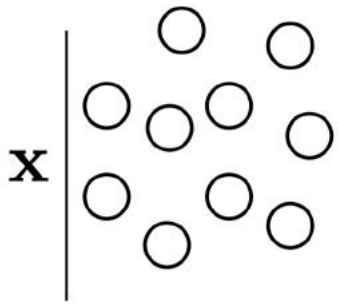
array of **tokens**



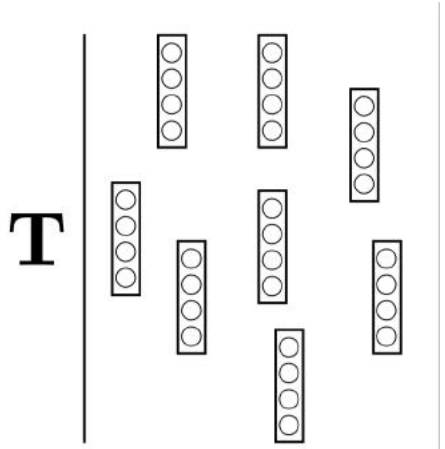
A new data structure: Tokens

- A **token** is just transformer lingo for a vector of neurons (note: GNNs also operate over tokens, but over there we called them "node attributes" or node "feature descriptors")
- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons

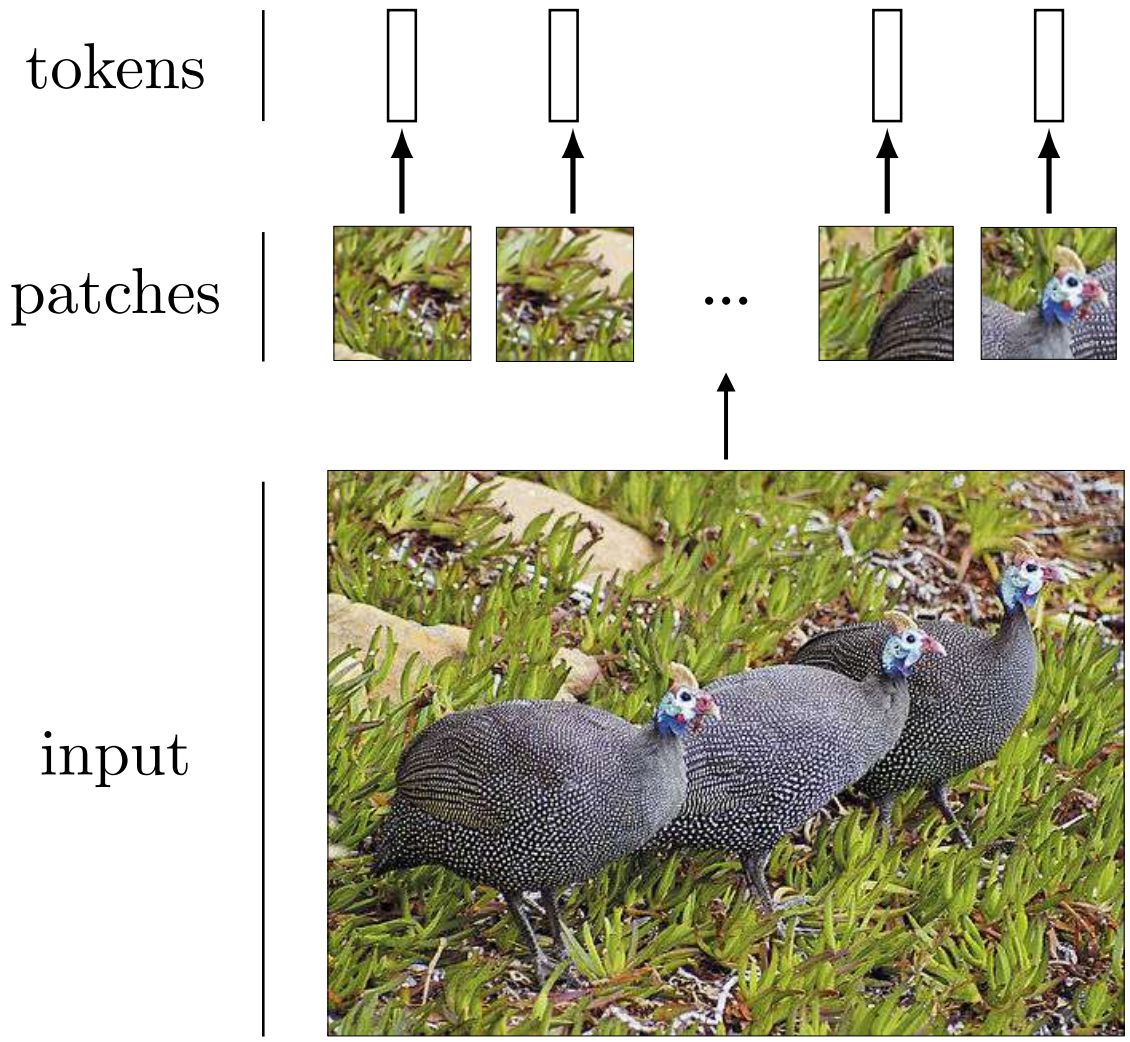
set of **neurons**



set of **tokens**



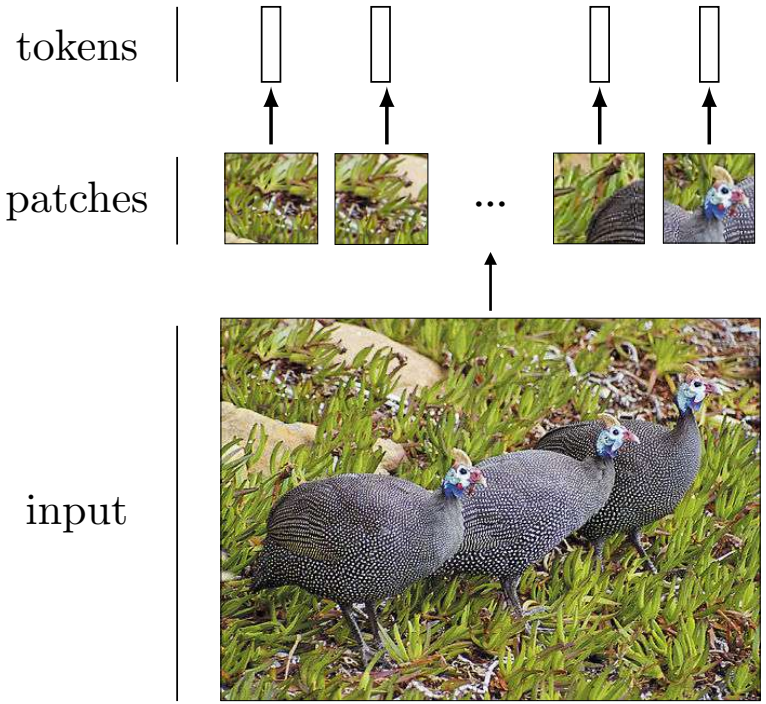
Tokenizing the input data

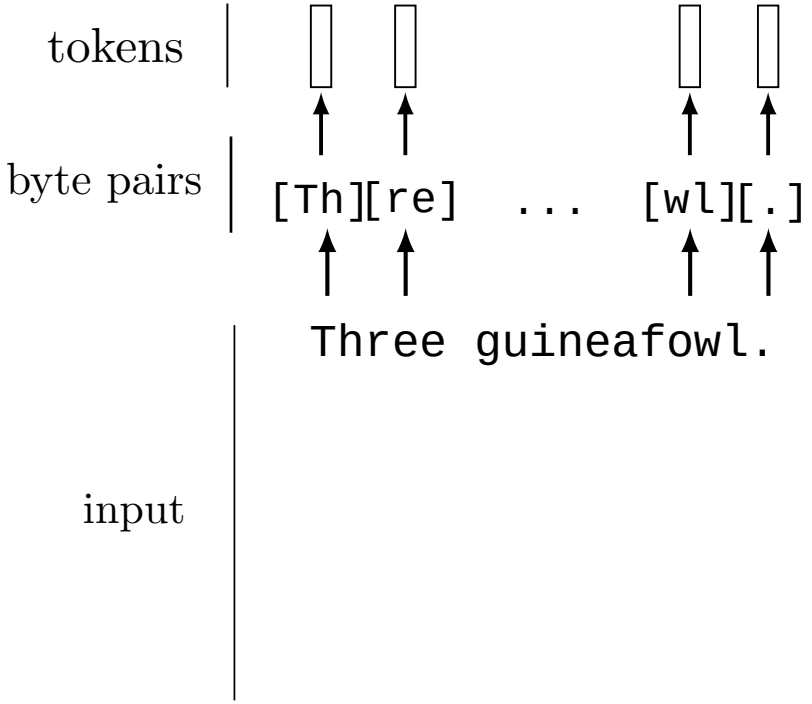


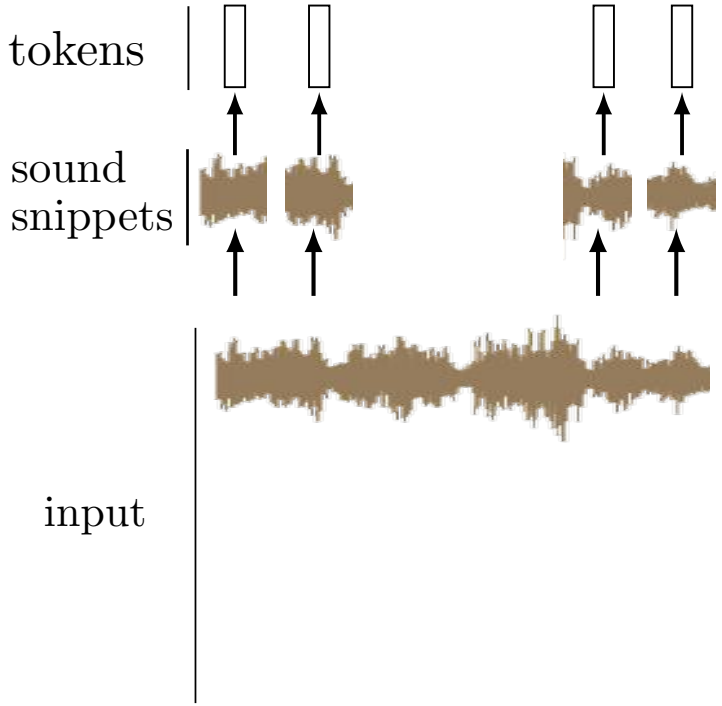
- When operating over neurons, we represent the input as an array of scalar-valued measurements (e.g., pixels)
- When operating over tokens, we represent the input as an array of vector-valued measurements

Tokenizing the input data

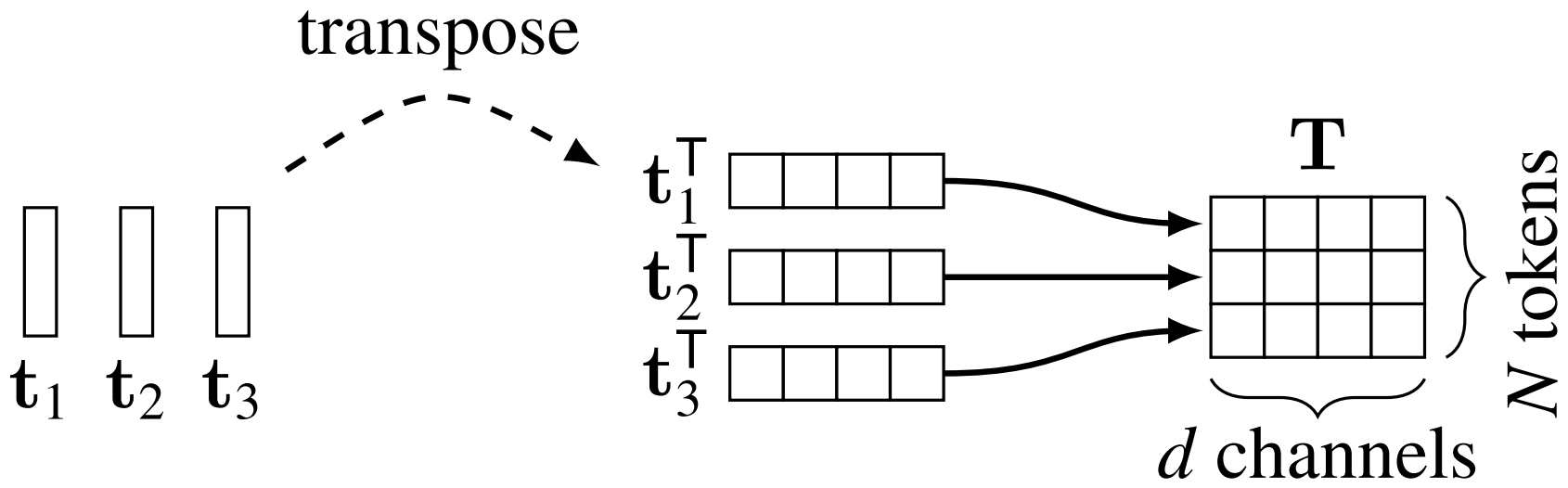
- You tokenize anything.
- General strategy: chop the input up into chunks, project each chunk to a vector





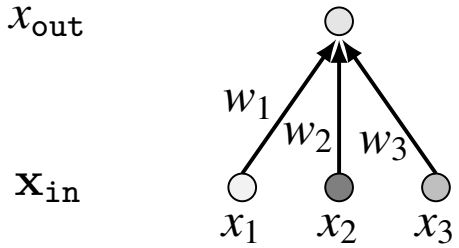


Notation


transpose
$\mathbf{t}_1$ $\mathbf{t}_2$ $\mathbf{t}_3$
$\mathbf{t}_1^\top$
$\mathbf{t}_2^\top$
$\mathbf{t}_3^\top$
$\mathbf{T}$
N tokens
d channels

Linear combination of tokens

Linear combination of neurons

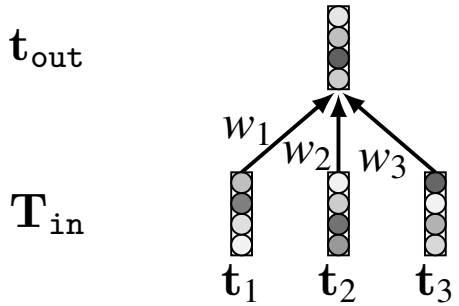


$$x_{\text{out}} = w_1 x_1 + w_2 x_2 + w_3 x_3$$

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

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

Linear combination of tokens



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

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

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

Token-wise nonlinearity

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

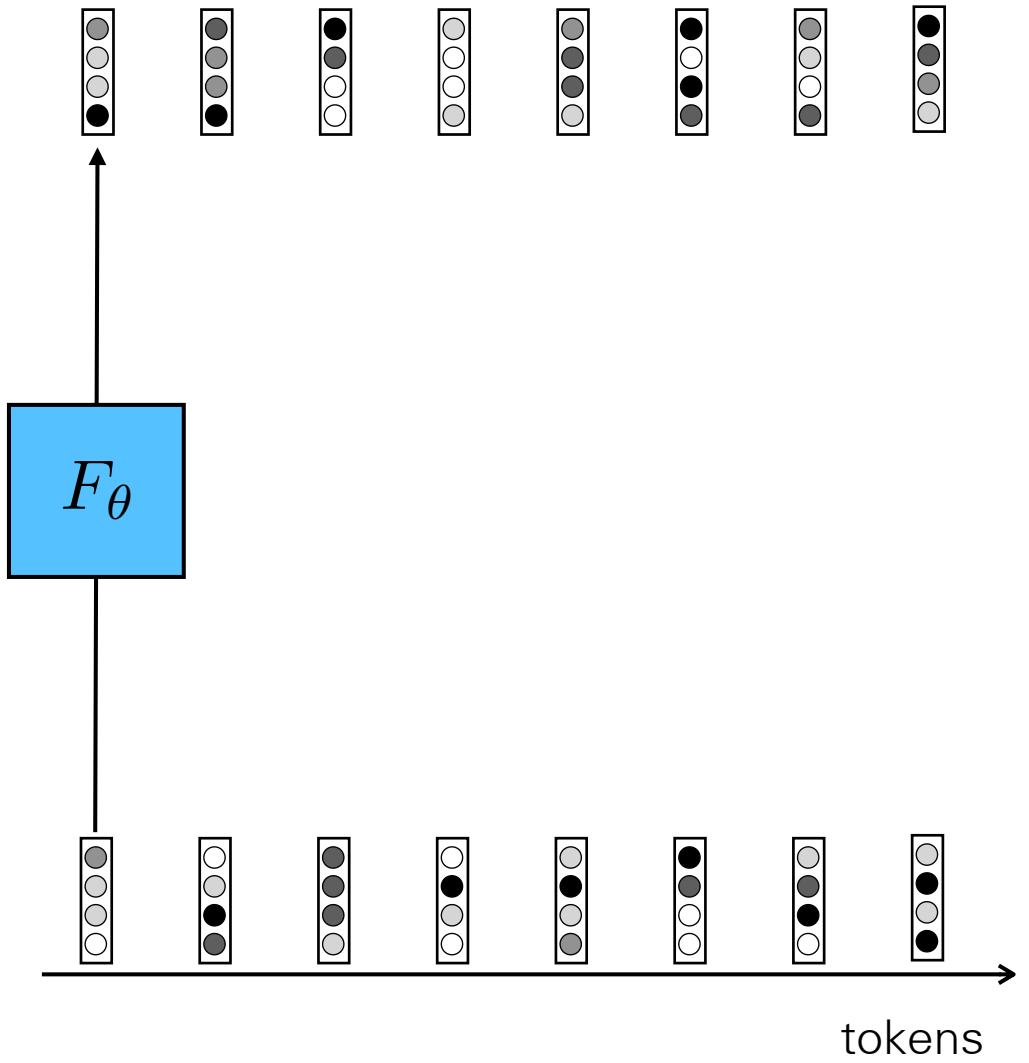
$$\mathbf{T}_{\texttt{out}} = \begin{bmatrix} F_\theta(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_\theta(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix}$$

- F is typically an MLP
- Equivalent to a CNN with 1x1 kernels run over token sequence

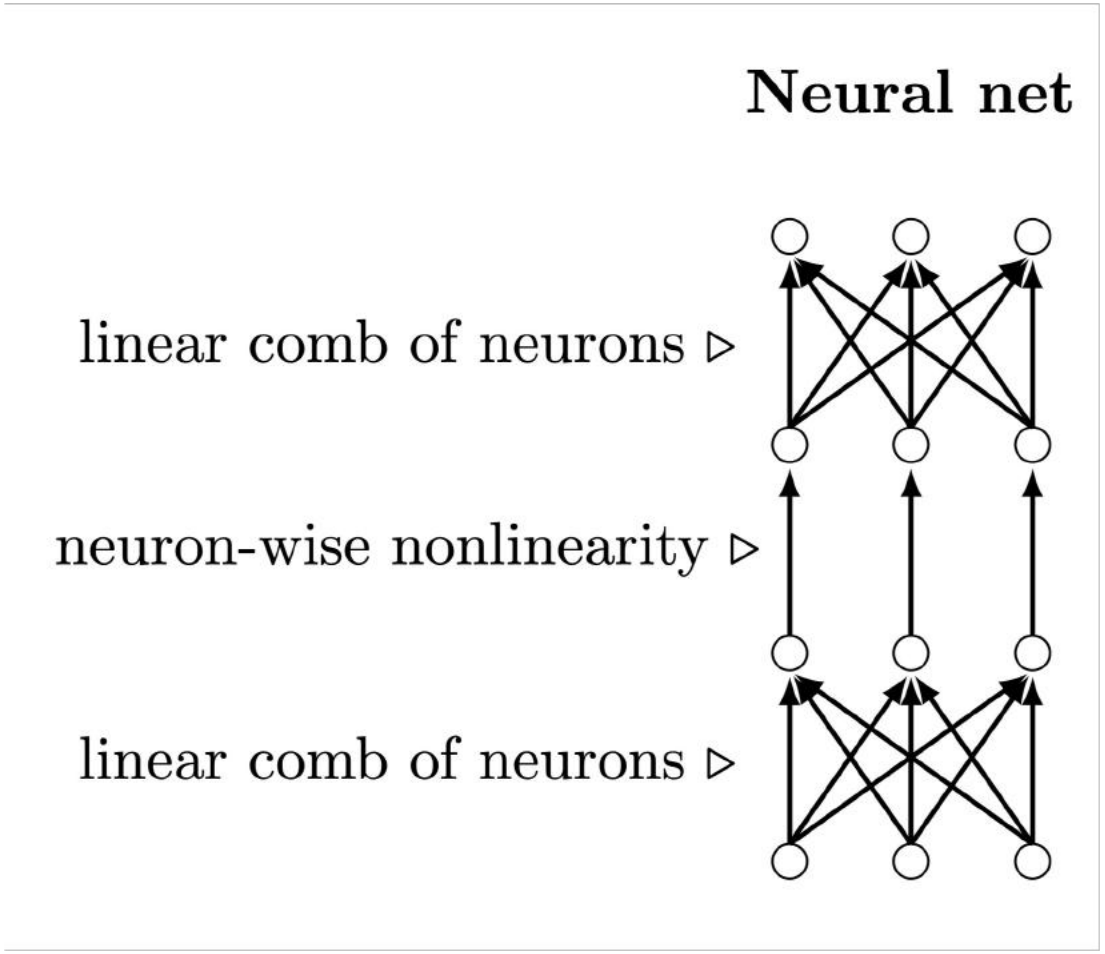
Token-wise nonlinearity

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

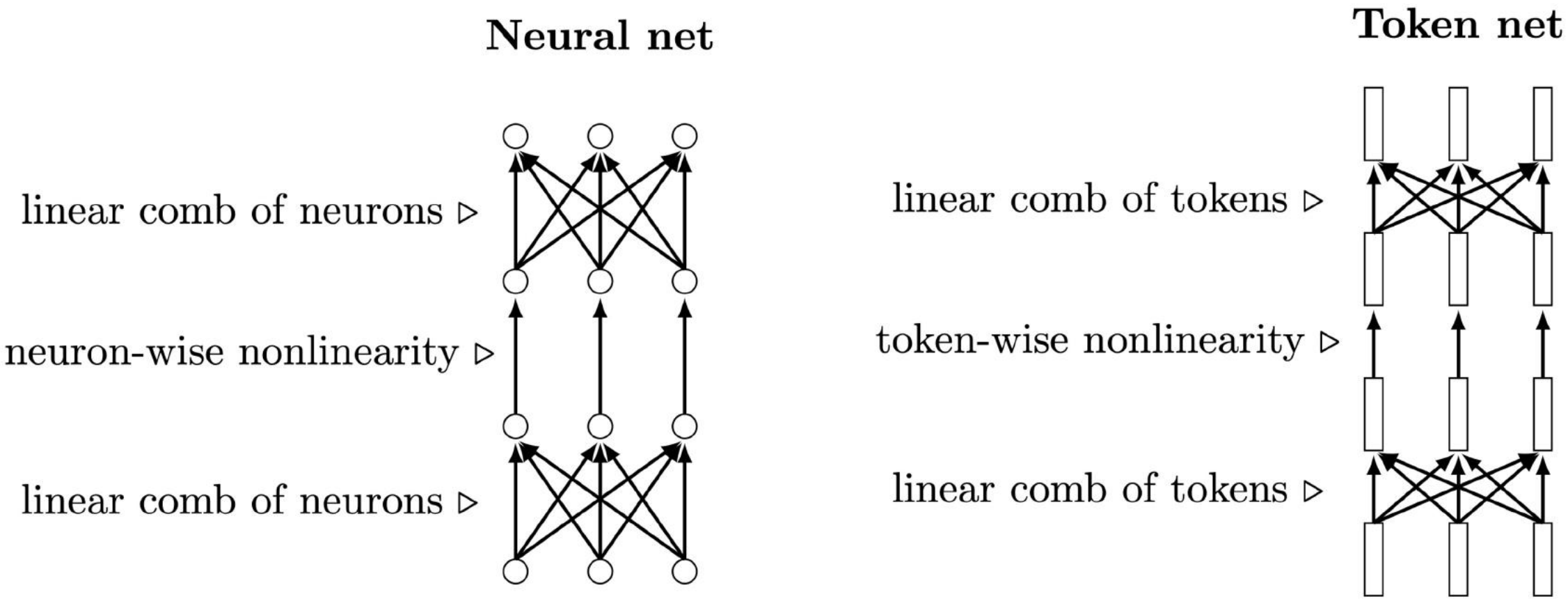
$$\mathbf{T}_{\texttt{out}} = \begin{bmatrix} F_\theta(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_\theta(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix}$$

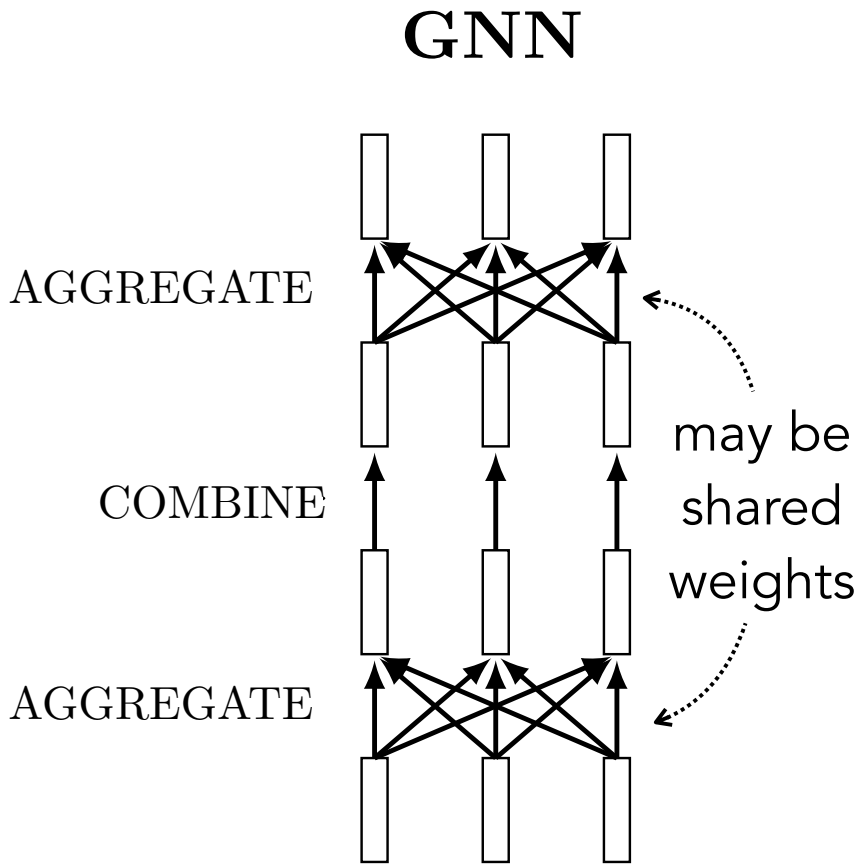


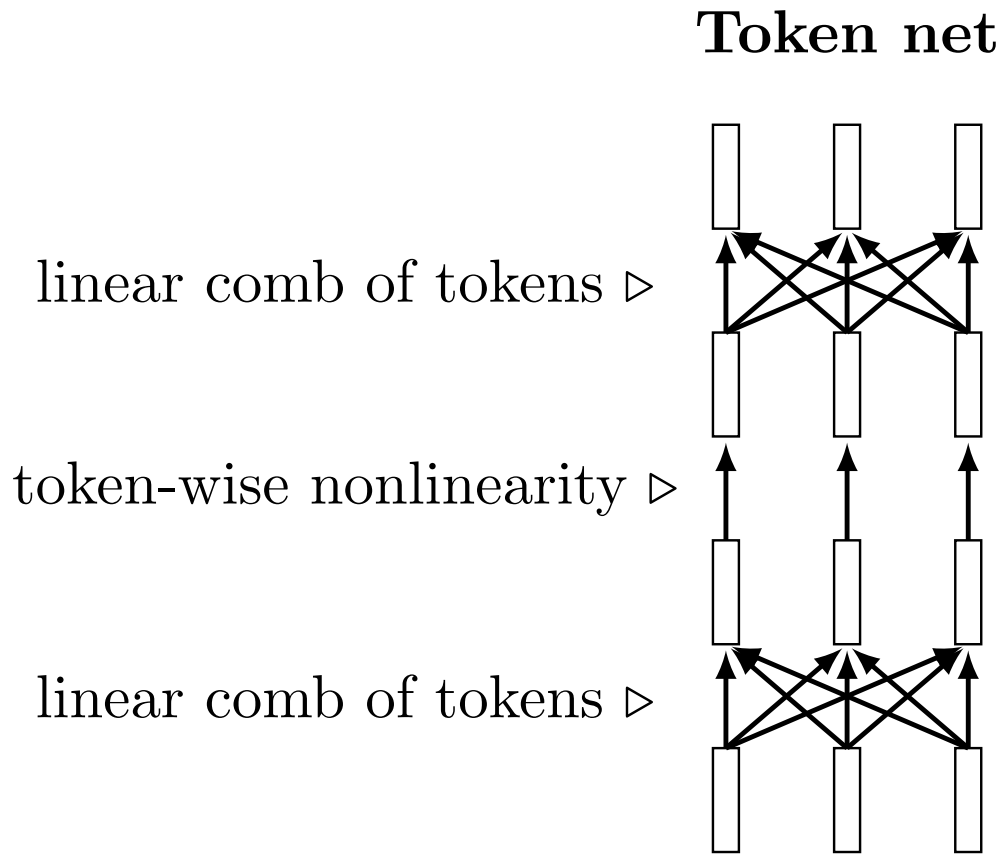
Token nets


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

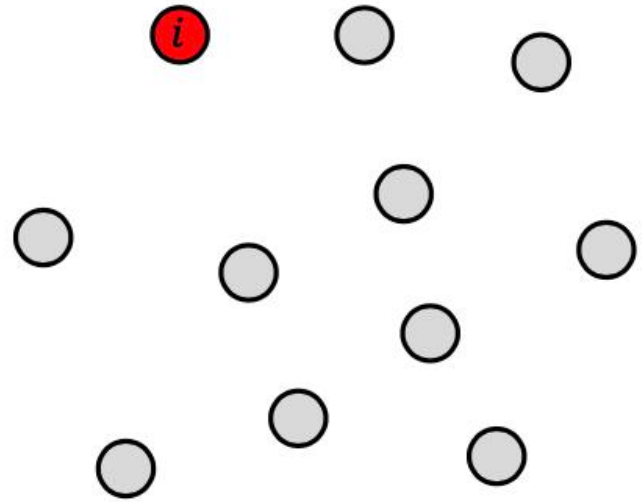
Token nets


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 ▷

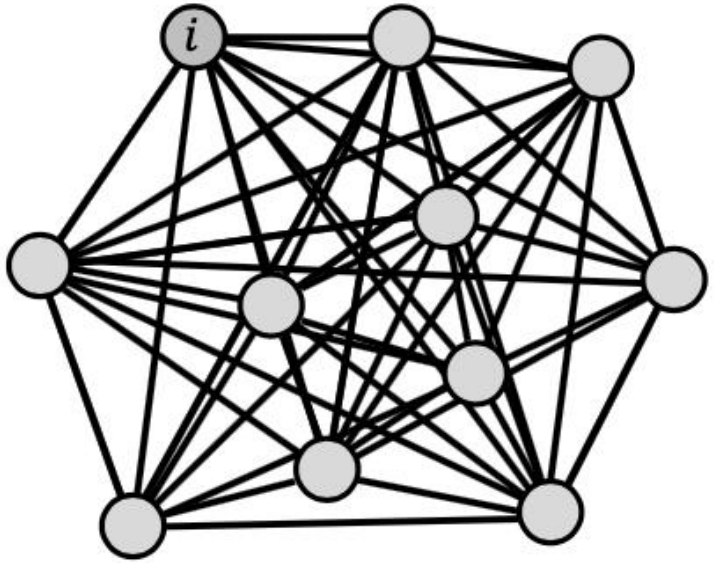

GNN
AGGREGATE
COMBINE
AGGREGATE
may be
shared
weights


Token net
linear comb of tokens ▷
token-wise nonlinearity ▷
linear comb of tokens ▷

A view from the graph perspective



Sets are like graphs
without edges

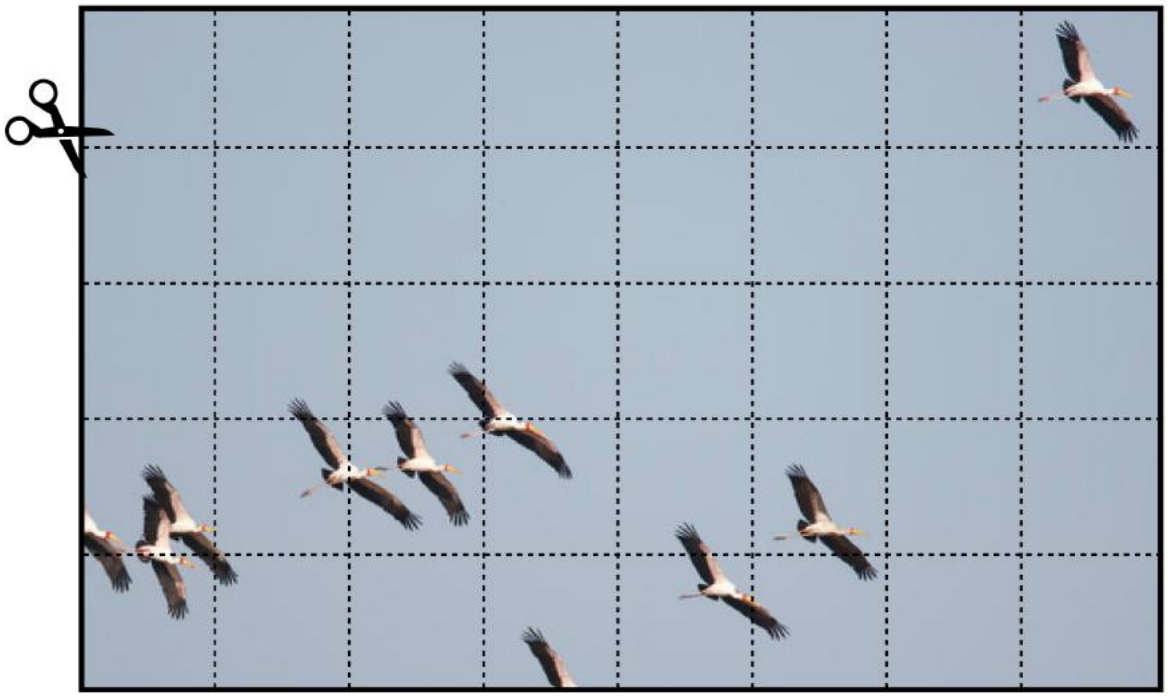


Sets as complete graphs:
focus on pairwise interactions
(e.g., **transformer**: aggregation
with attention)

DeepSets and Transformer architectures may be viewed as Graph Neural Networks

Idea #2: Attention

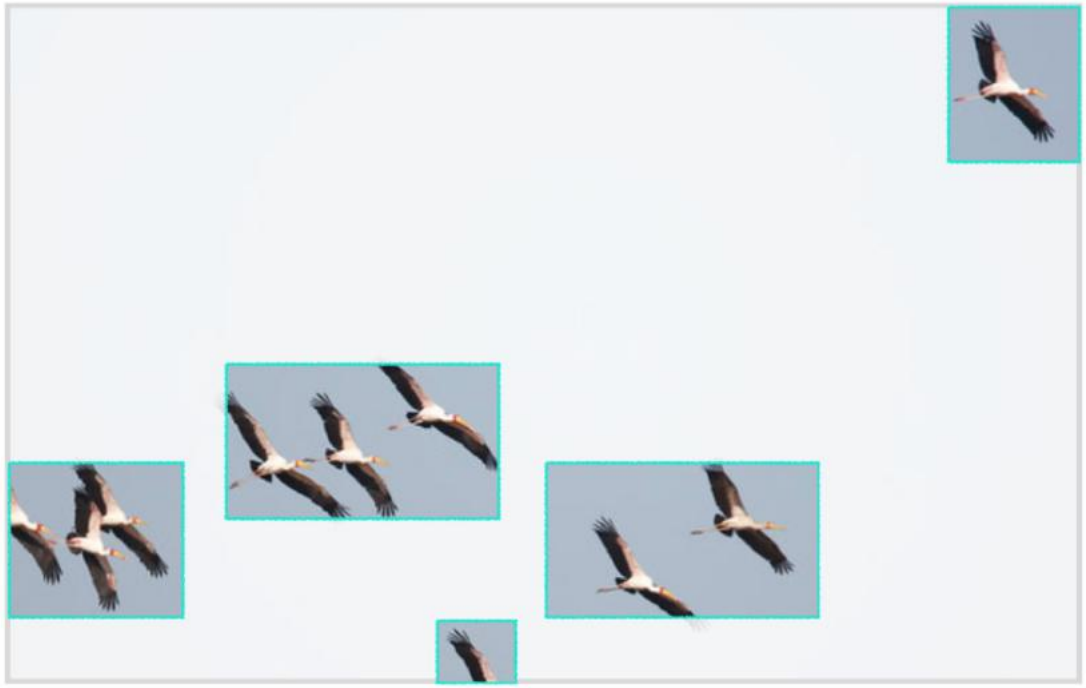
A limitation of CNNs



How many birds are in this image?

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

What is attention?

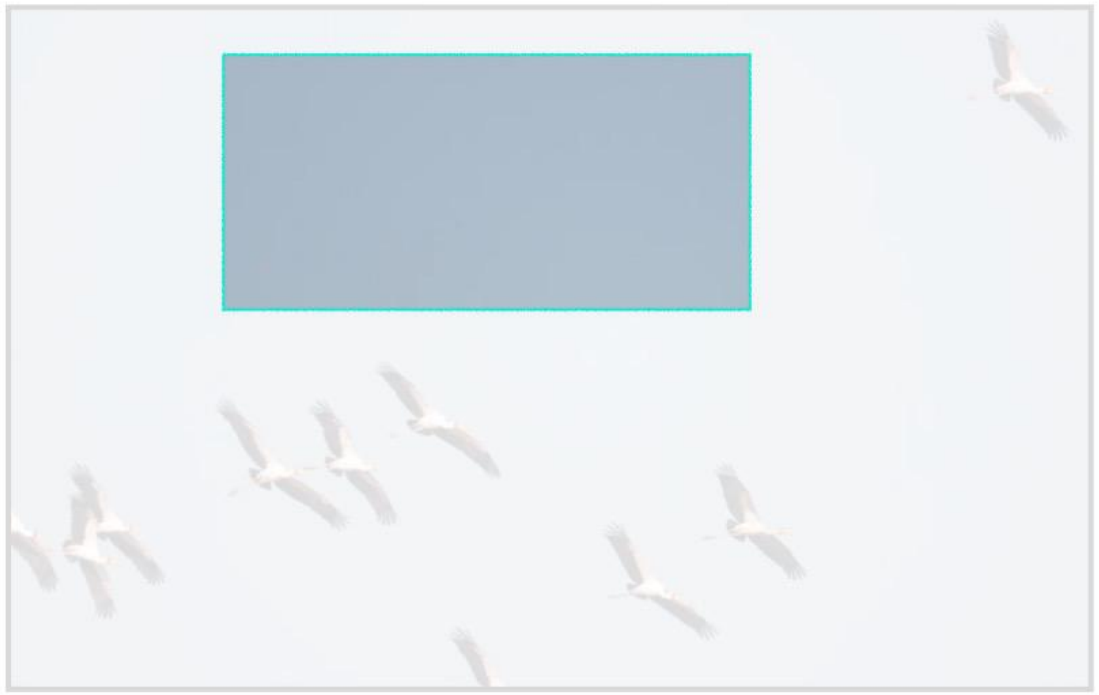


How many birds are in this image?

What is attention?

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

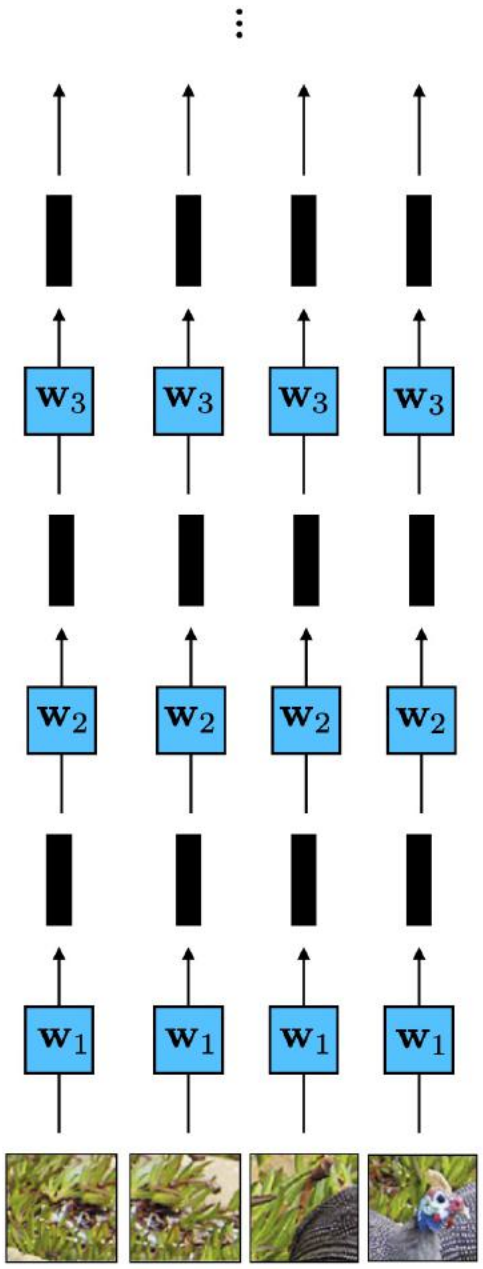
What is attention?



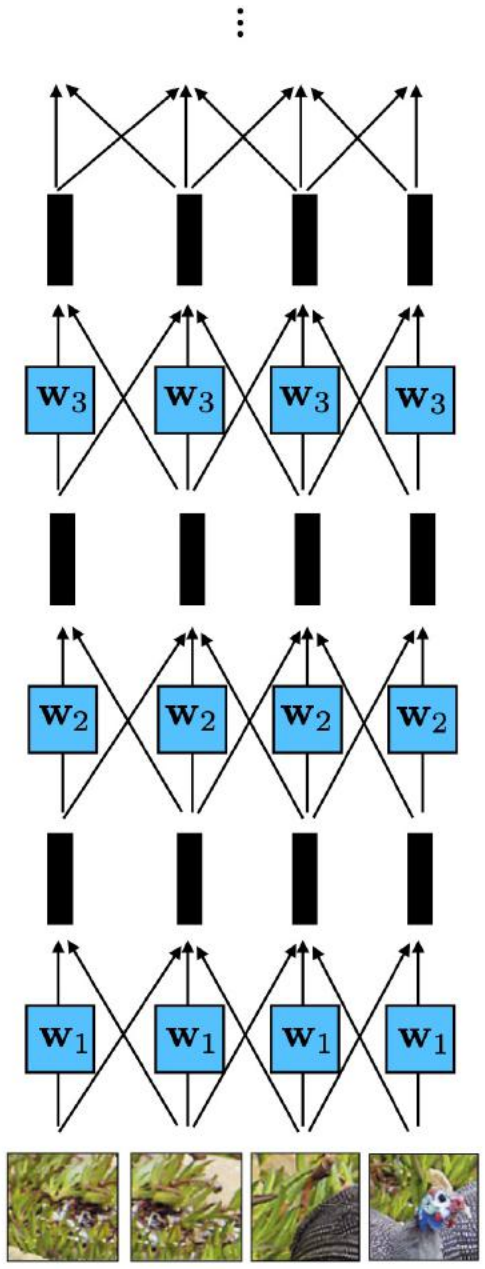
What’s the color of the sky?

Different ways of aggregating information over space

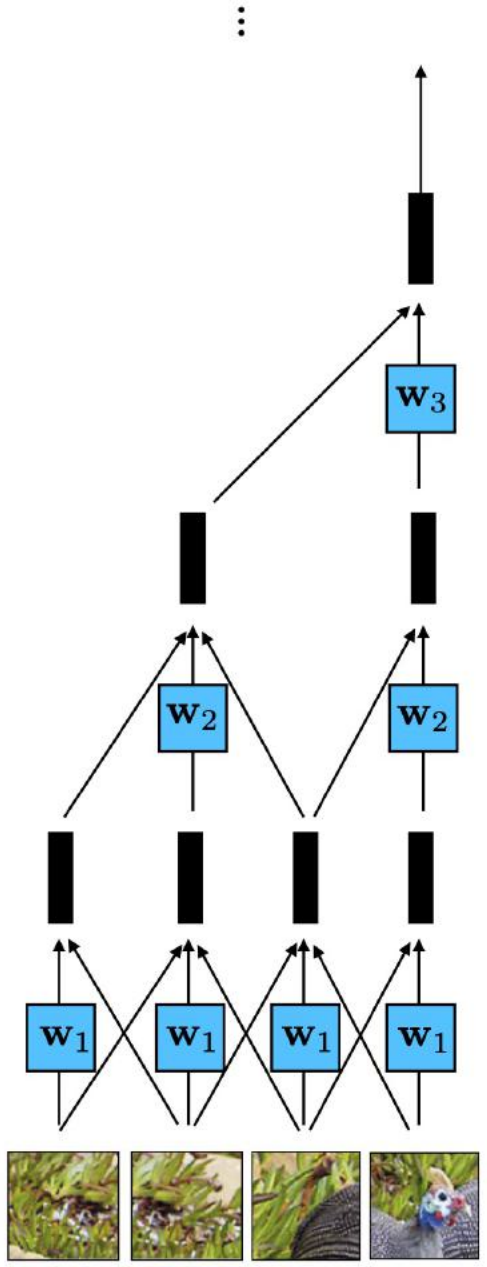
conv w/o overlap



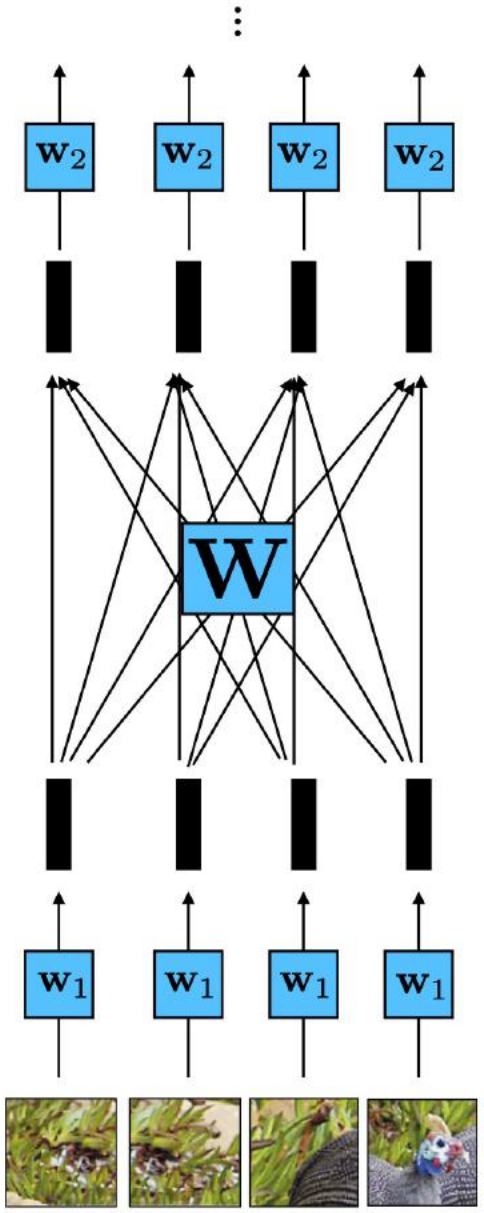
conv w overlap



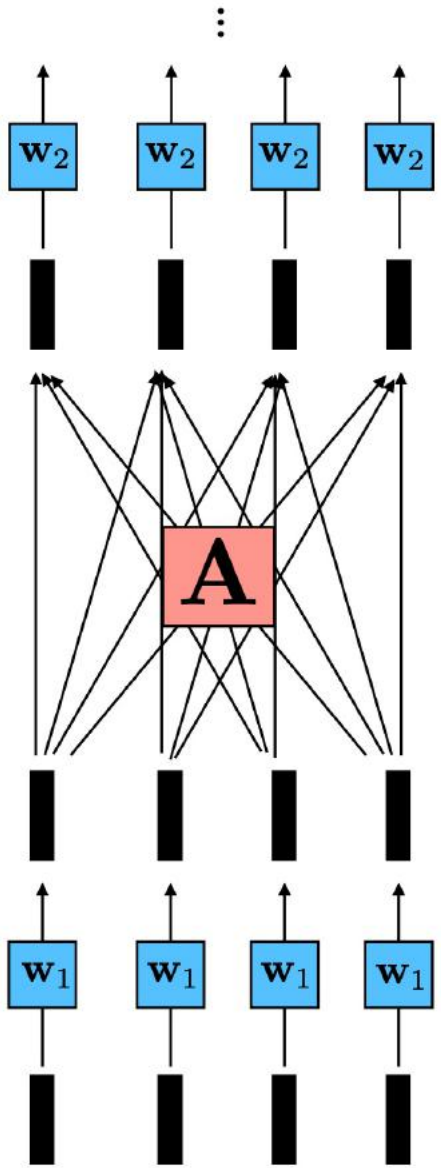
conv pyramid

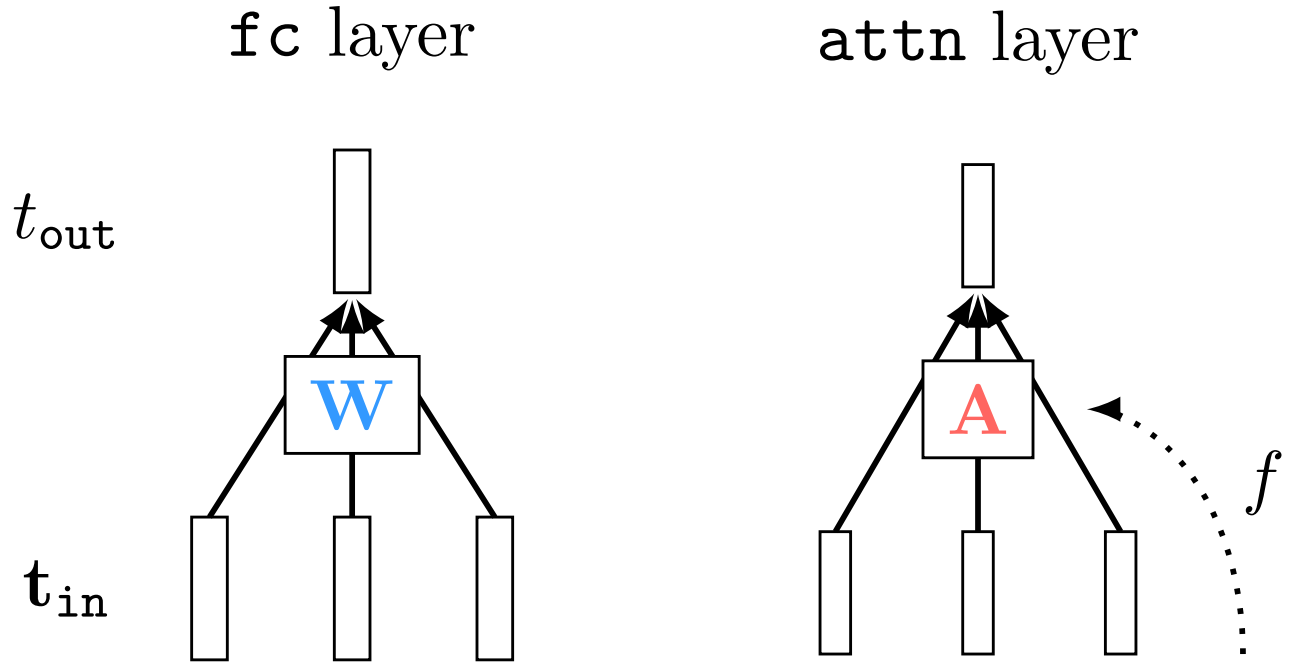


fc layer



Attention Layer



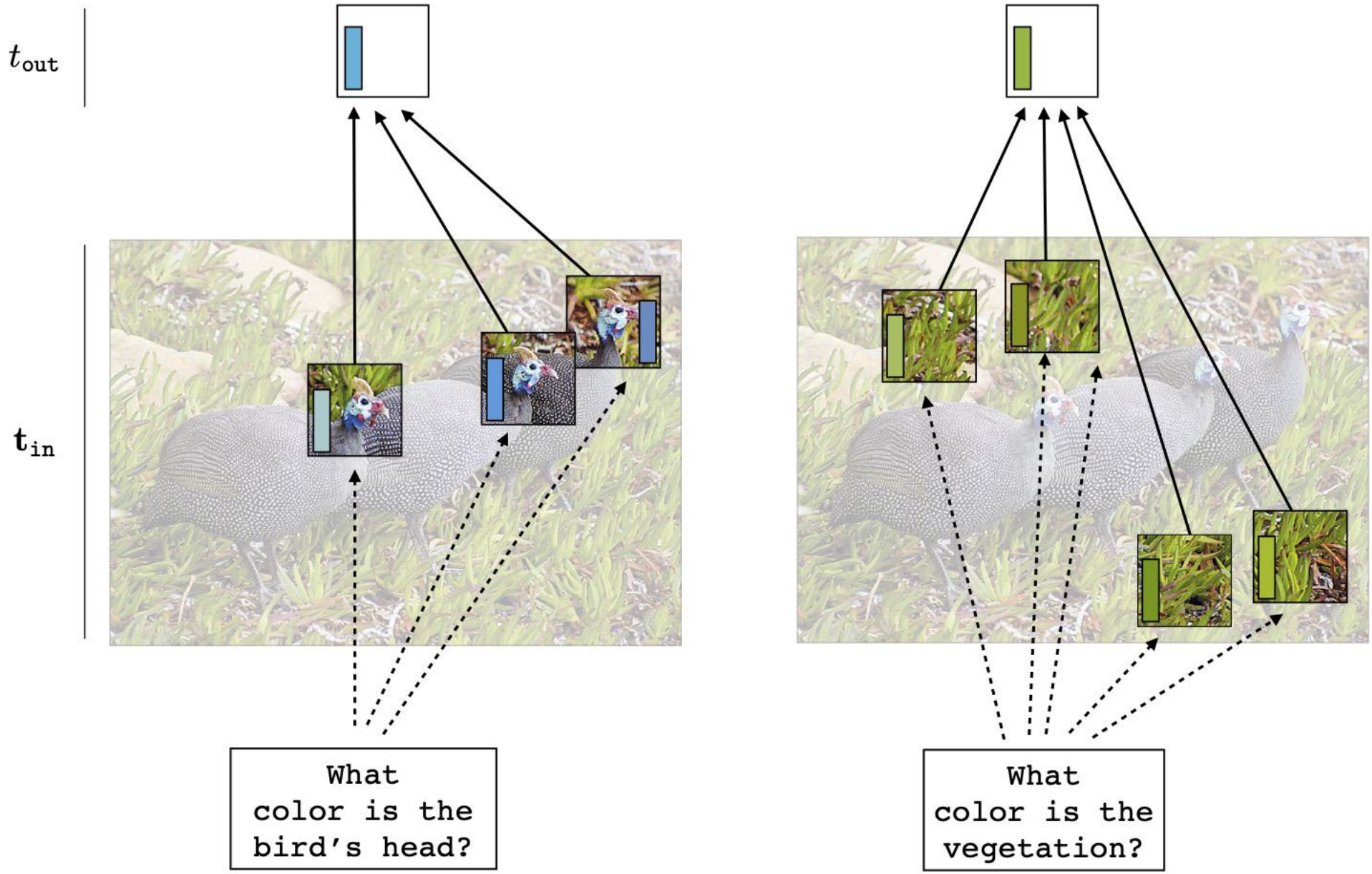


- W is free parameters
- A is a function of some input data. The data tells us which tokens to attend to (assign high weight in weighted sum)

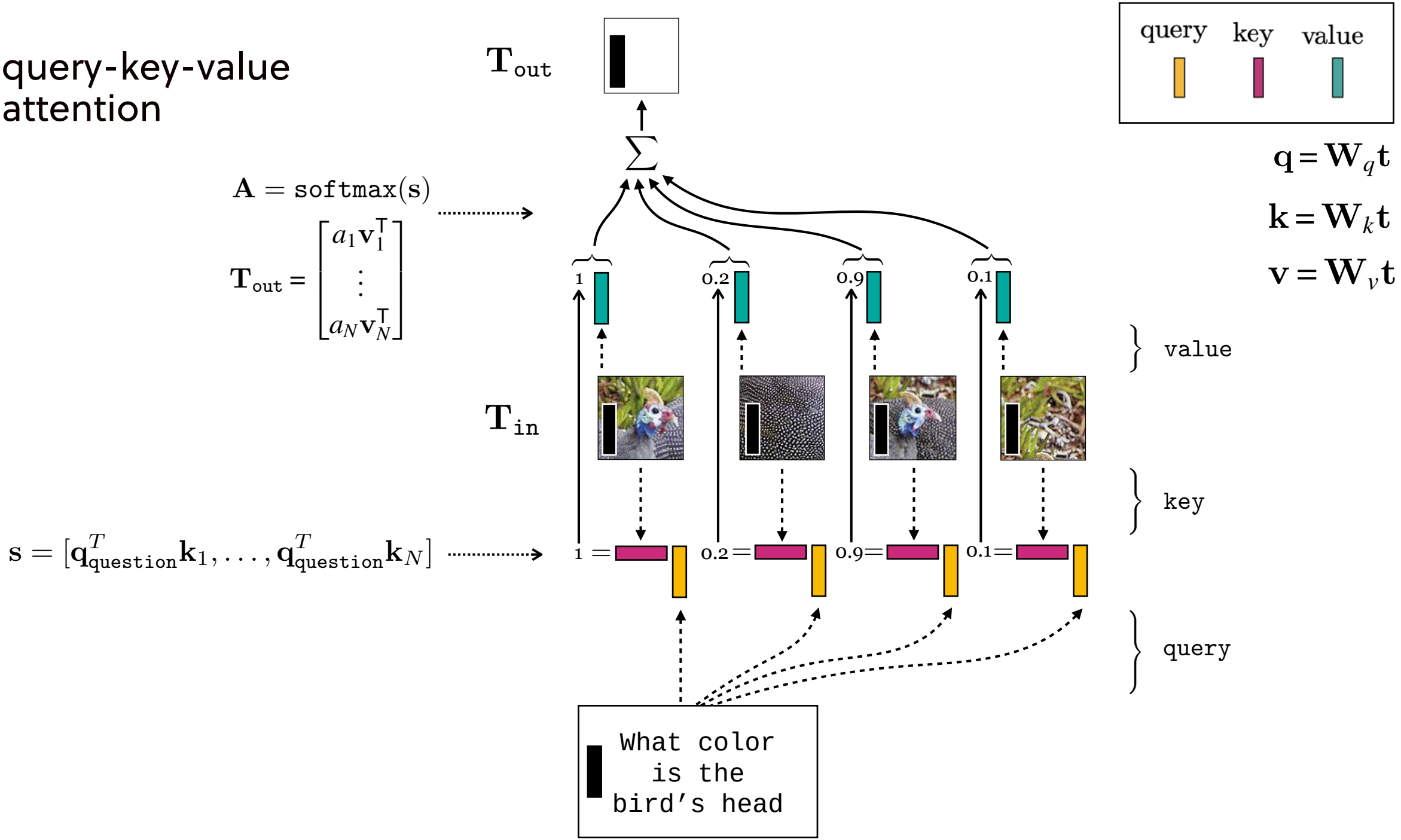
t_{out}

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

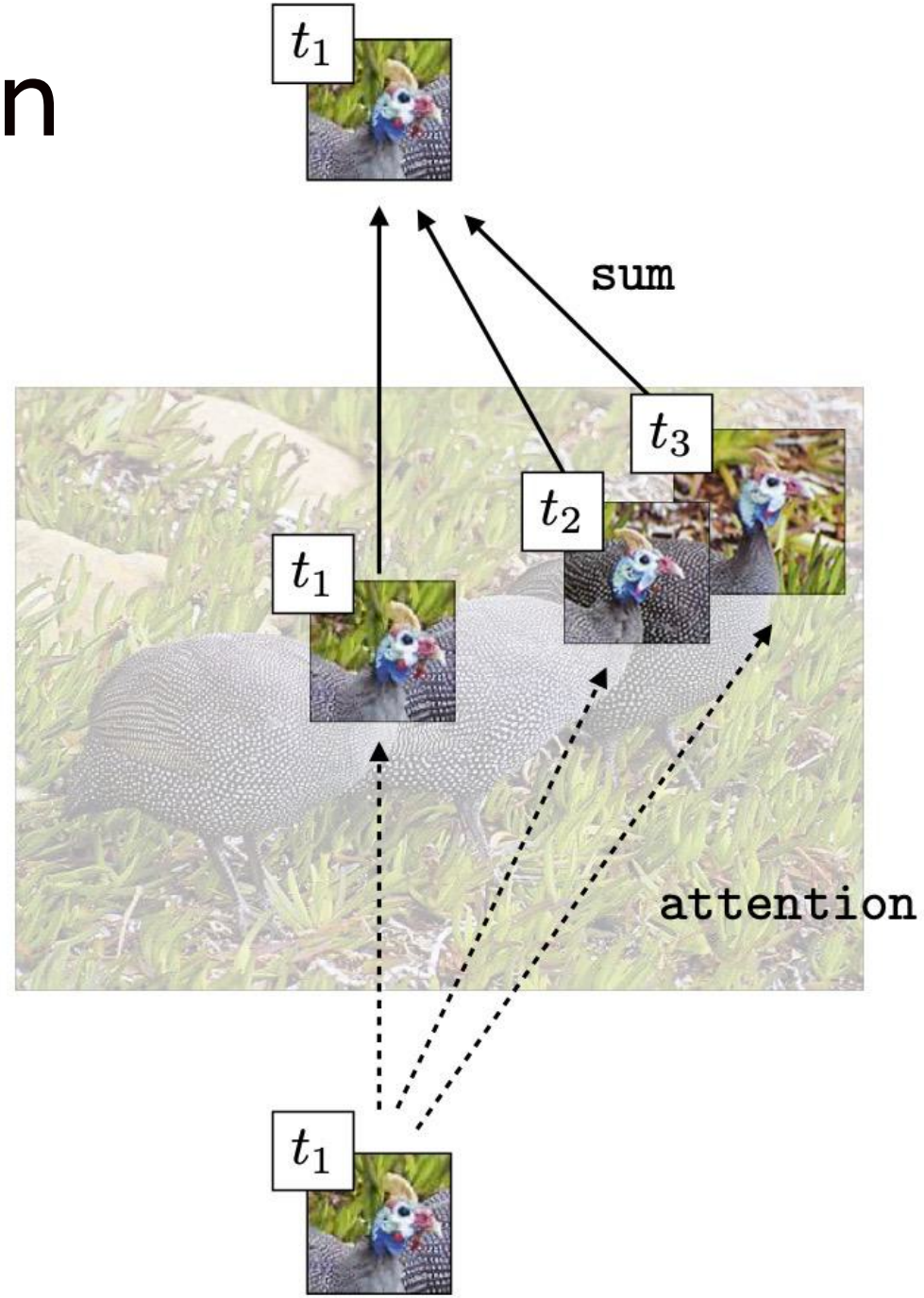
What color is the bird’s head?


t_{out}
$\mathbf{t}_{\text{in}}$
What color is the bird's head?
What color is the vegetation?

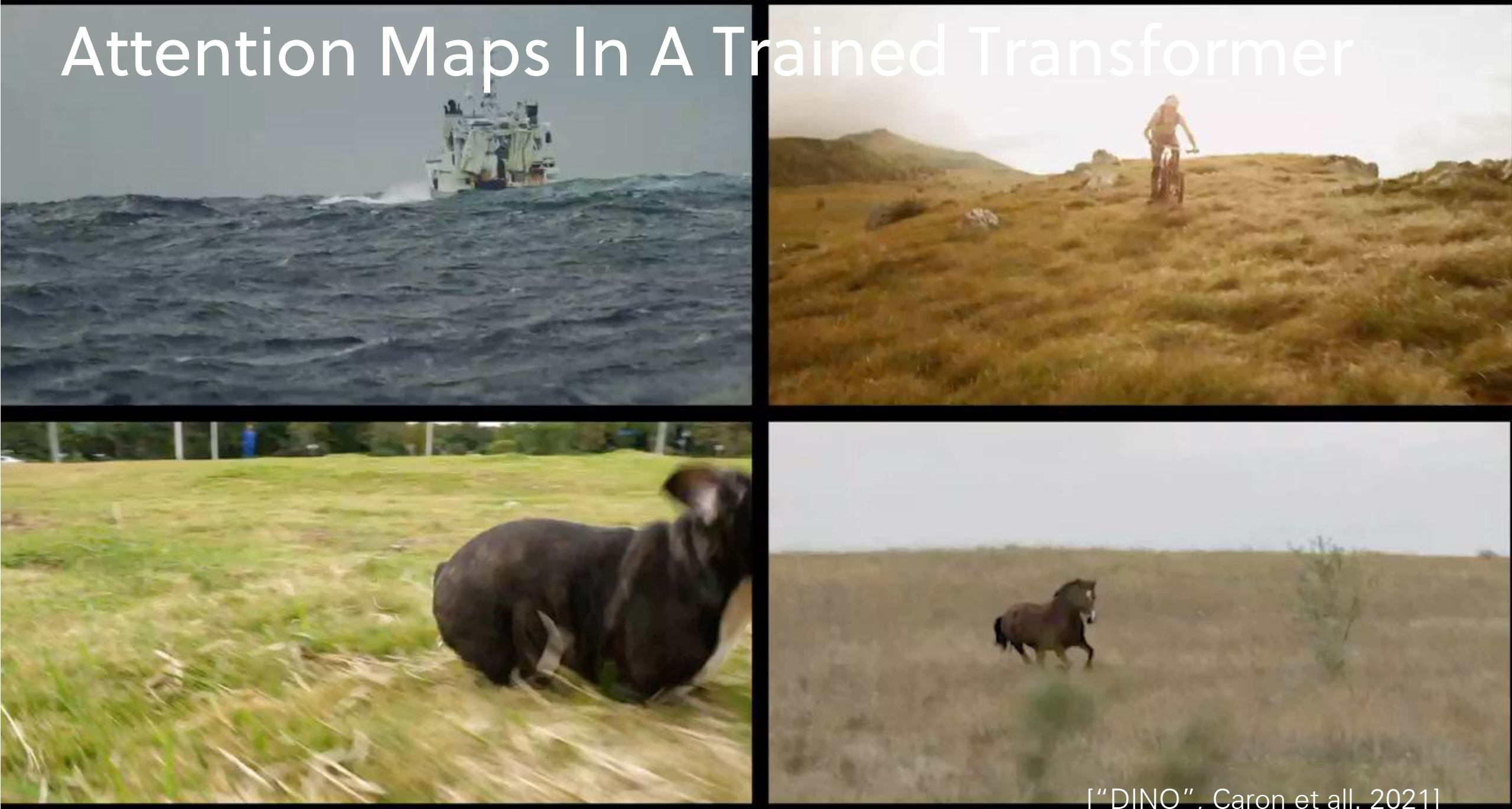
query-key-value attention



Self-attention

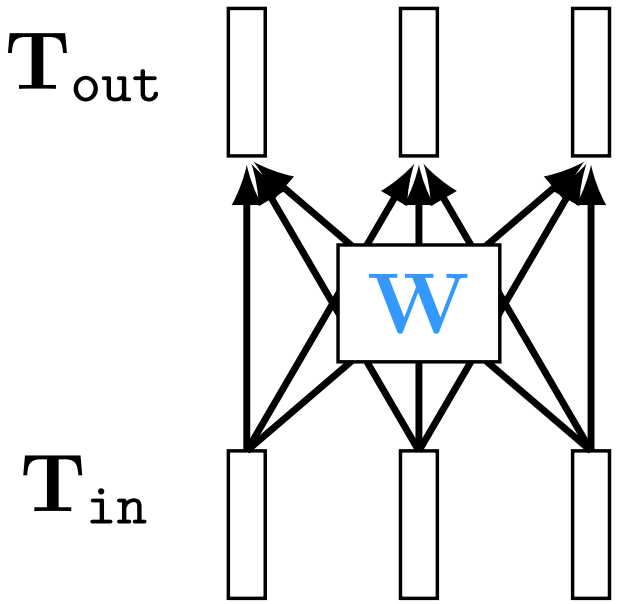


Attention Maps In A Trained Transformer

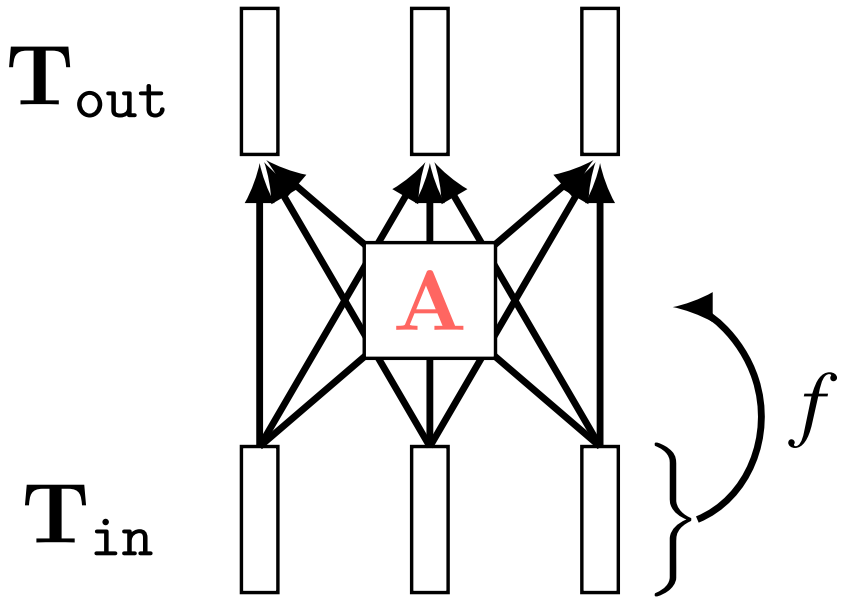


["DINO", Caron et all. 2021]

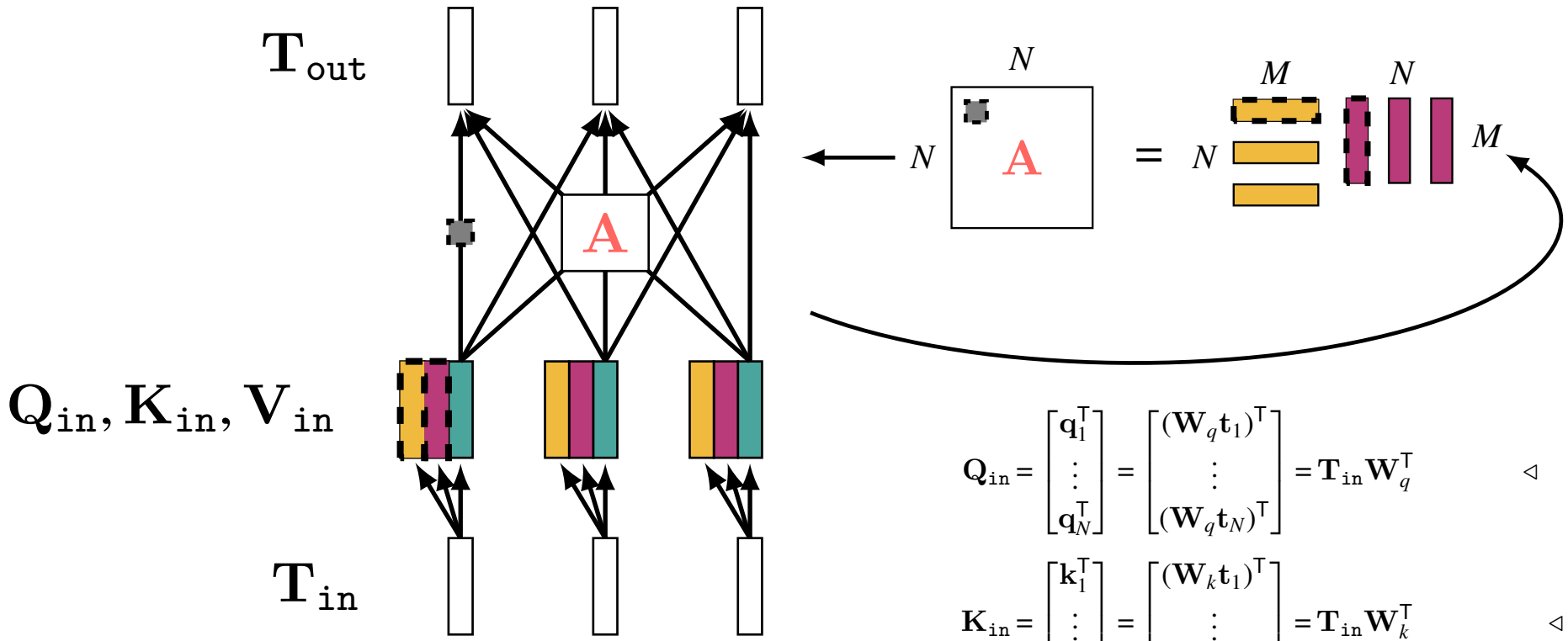
fc layer



self attn layer



`self attn` layer (expanded)



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

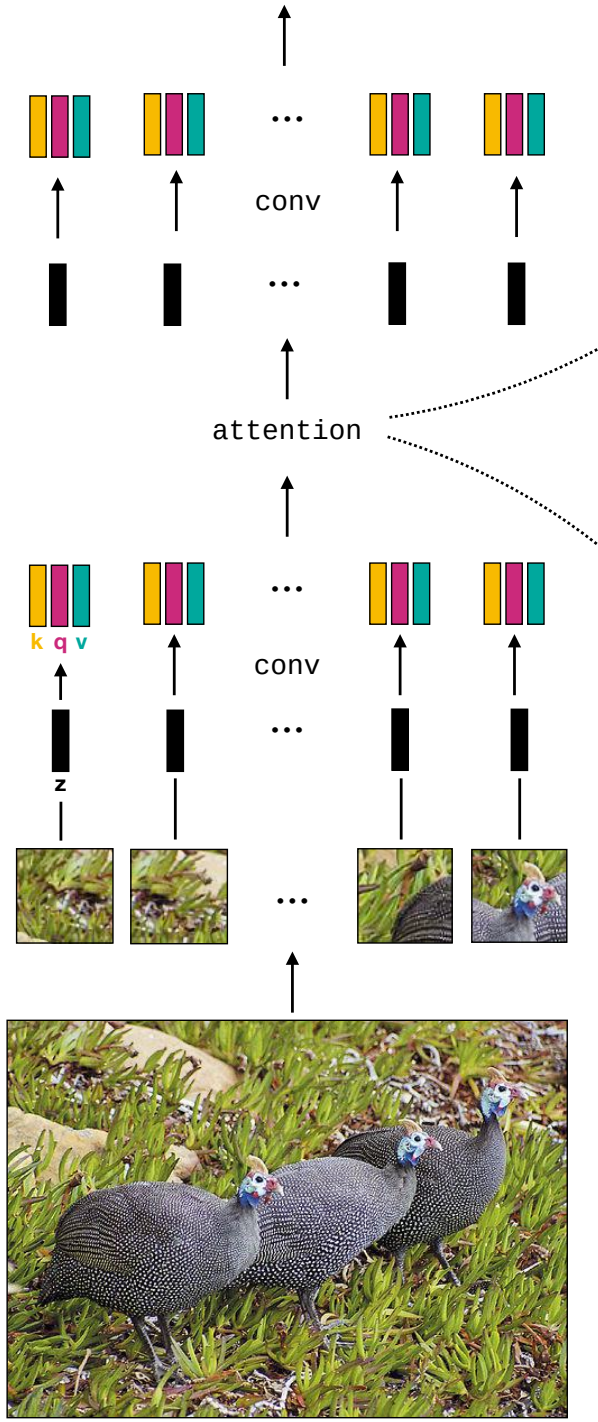
$$\mathbf{K}_{\texttt{in}} = \begin{bmatrix} \mathbf{k}_1^\mathsf{T} \\ \vdots \\ \mathbf{k}_N^\mathsf{T} \end{bmatrix} = \begin{bmatrix} (\mathbf{W}_k \mathbf{t}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{W}_k \mathbf{t}_N)^\mathsf{T} \end{bmatrix} = \mathbf{T}_{\texttt{in}} \mathbf{W}_k^\mathsf{T} \quad \triangleleft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

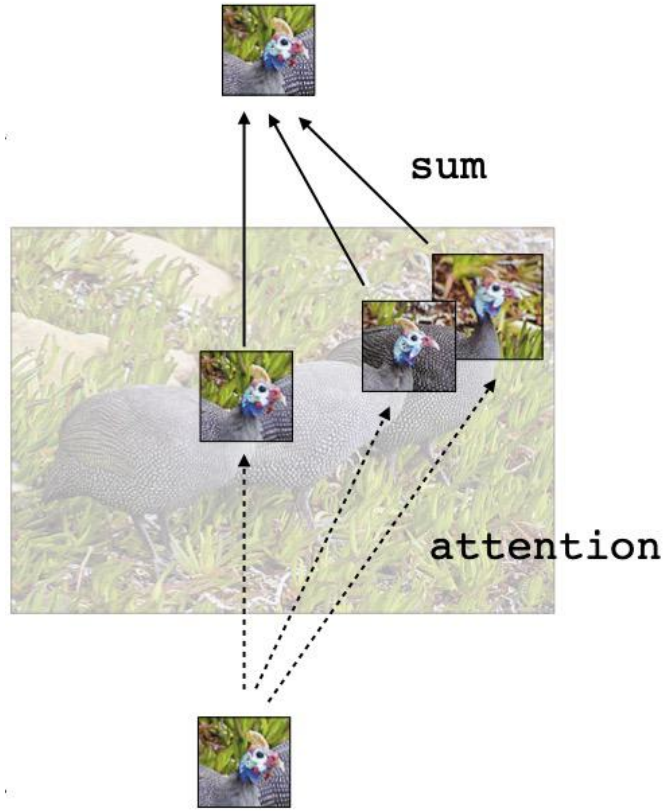
$$\mathbf{V}_{\texttt{in}} = \begin{bmatrix} \mathbf{v}_1^\mathsf{T} \\ \vdots \\ \mathbf{v}_N^\mathsf{T} \end{bmatrix} = \begin{bmatrix} (\mathbf{W}_v \mathbf{t}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{W}_v \mathbf{t}_N)^\mathsf{T} \end{bmatrix} = \mathbf{T}_{\texttt{in}} \mathbf{W}_v^\mathsf{T} \quad \triangleleft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

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

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

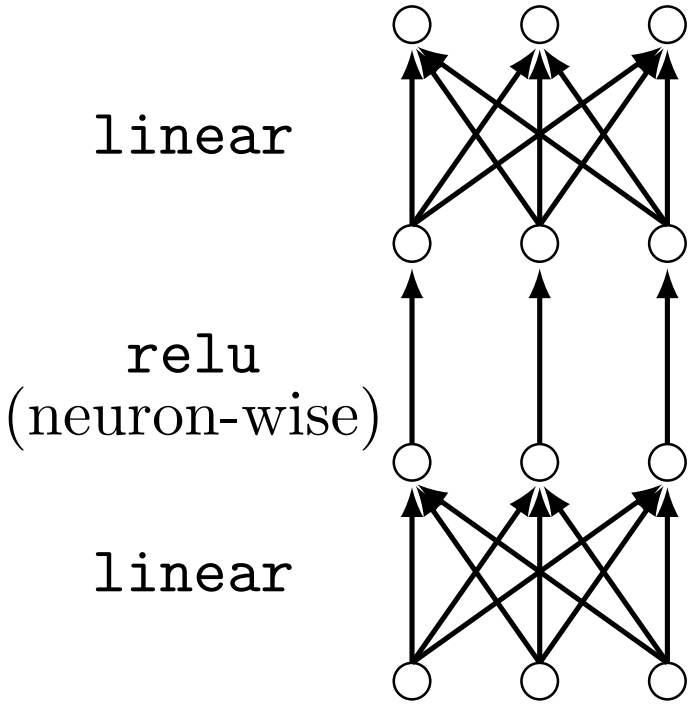
Transformer (simplified)



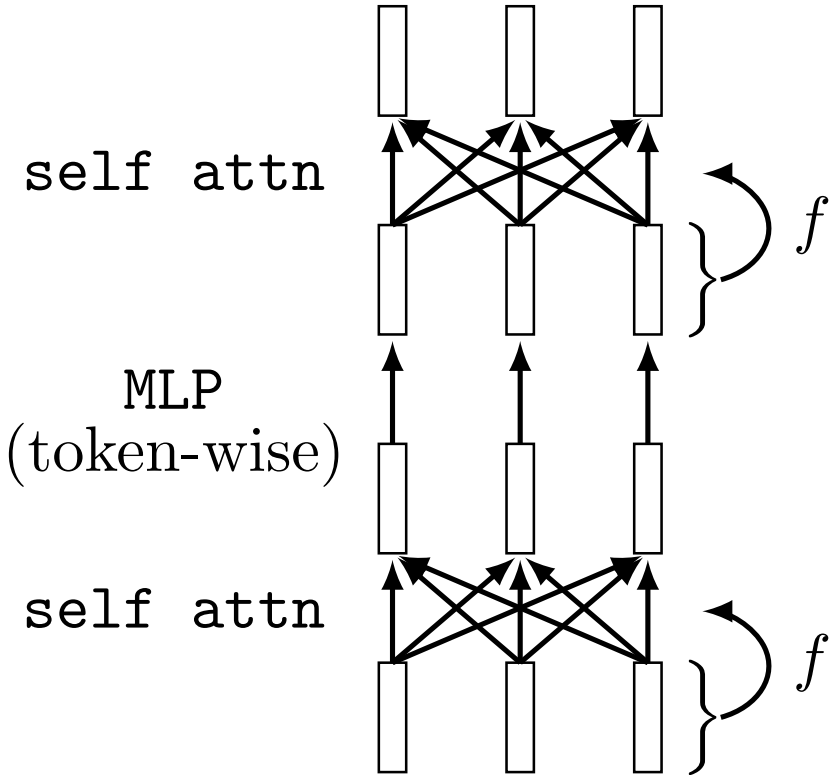


["Attention is All You Need", Vaswani et al. 2017]
["Vision Transformer", Dosovitskiy et al. 2020]

MLP



Transformer (vanilla)



Multihead self-attention (MSH)

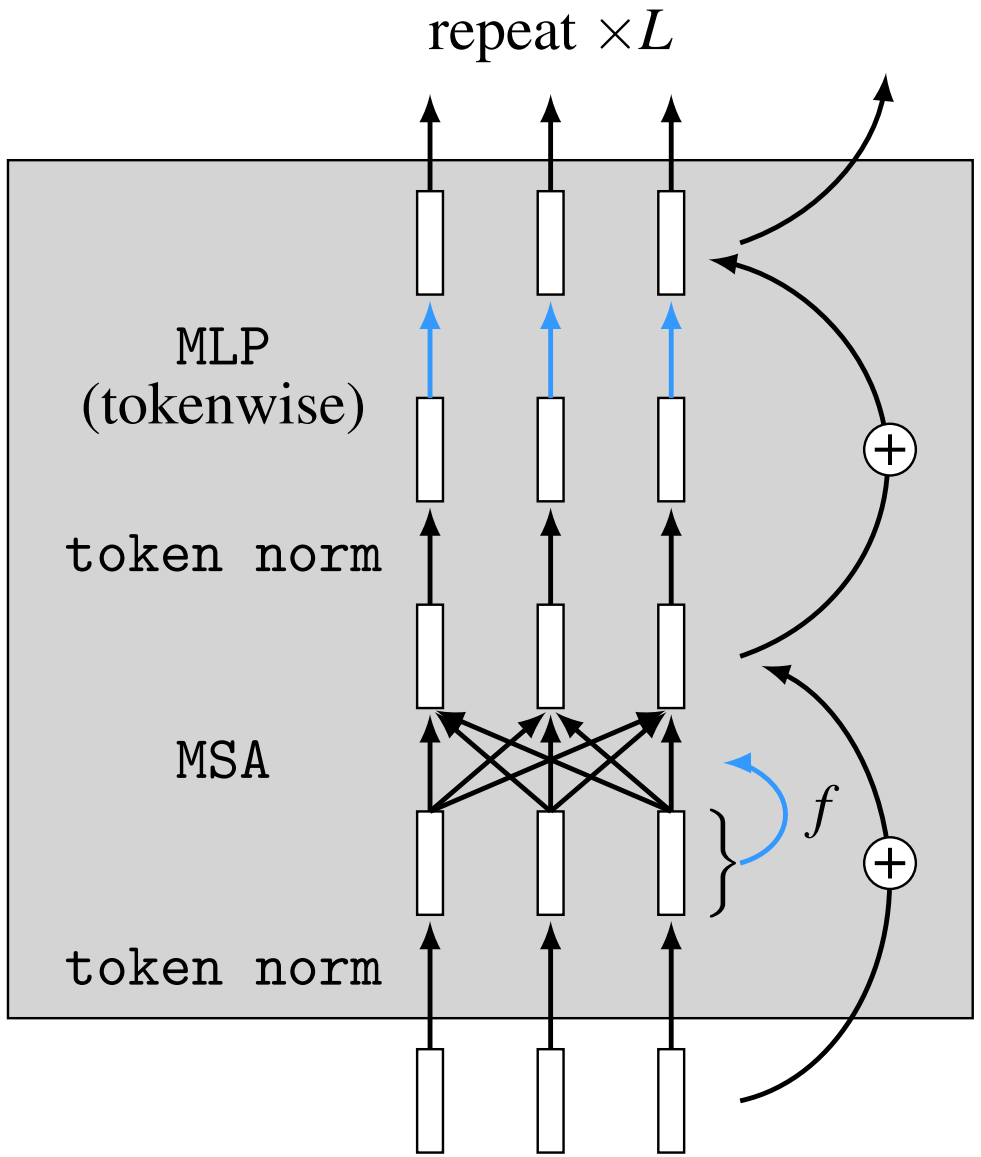
- Rather than having just one way of attending, why not have k?
- Each gets its own parameterized query(), key(), value() functions.
- Run them all in parallel, then (weighted) sum the output token code vectors

$$\mathbf{T}^i_{\texttt{out}} = \texttt{attn}^i(\mathbf{T}_{\texttt{in}}) \quad \forall i$$

$$\bar{\mathbf{T}}_{\texttt{out}} = \begin{bmatrix} \mathbf{T}^1_{\texttt{out}}[0,:] & \dots & \mathbf{T}^k_{\texttt{out}}[0,:] \\ \vdots & \vdots & \vdots \\ \mathbf{T}^1_{\texttt{out}}[N-1,:] & \dots & \mathbf{T}^k_{\texttt{out}}[N-1,:] \end{bmatrix} \quad \triangleleft \quad \bar{\mathbf{T}}_{\texttt{out}} \in \mathbb{R}^{N \times kv}$$

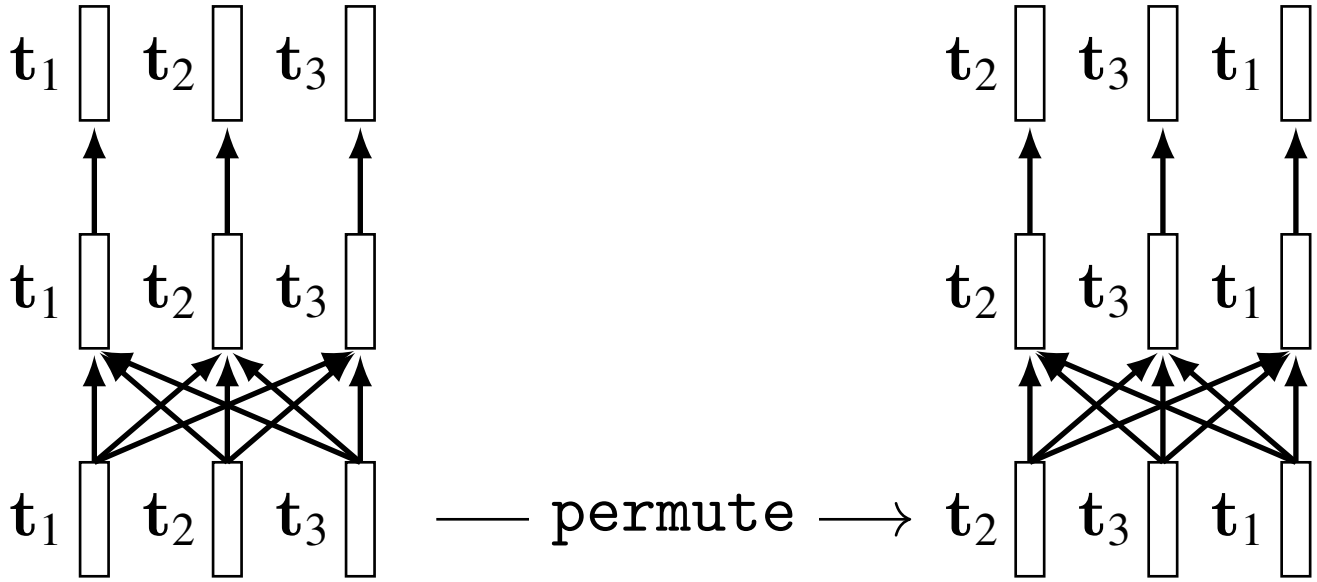
$$\mathbf{T}_{\texttt{out}} = \bar{\mathbf{T}}_{\texttt{out}} \mathbf{W} \quad \triangleleft \quad \mathbf{W} \in \mathbb{R}^{kv \times d}$$

Transformer (ViT)



Idea #3: Positional Encoding

Permutation equivariance



$$F_\theta(\texttt{permute}(\mathbf{T}_{\texttt{in}})) = \texttt{permute}(F_\theta(\mathbf{T}_{\texttt{in}}))$$

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

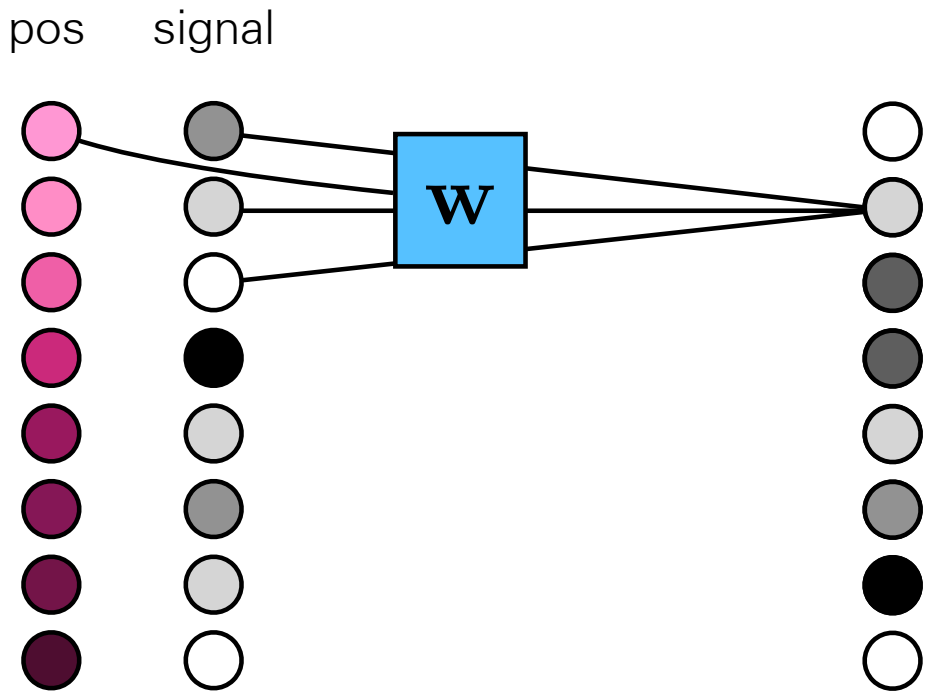
$$\downarrow$$

$$\texttt{transformer}(\texttt{permute}(\mathbf{T}_{\texttt{in}})) = \texttt{permute}(\texttt{transformer}(\mathbf{T}_{\texttt{in}}))$$

Set2Set

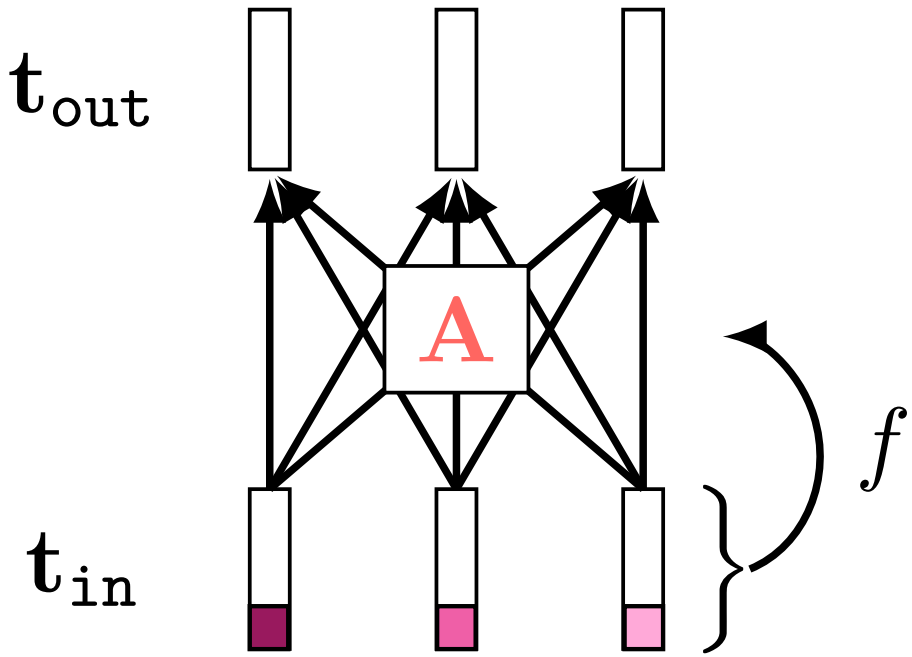
What if you don't want to be shift invariant?

1. Use an architecture that is not permutation invariant (e.g., MLP)
2. Add location information to the token code vectors — this is called **positional encoding**



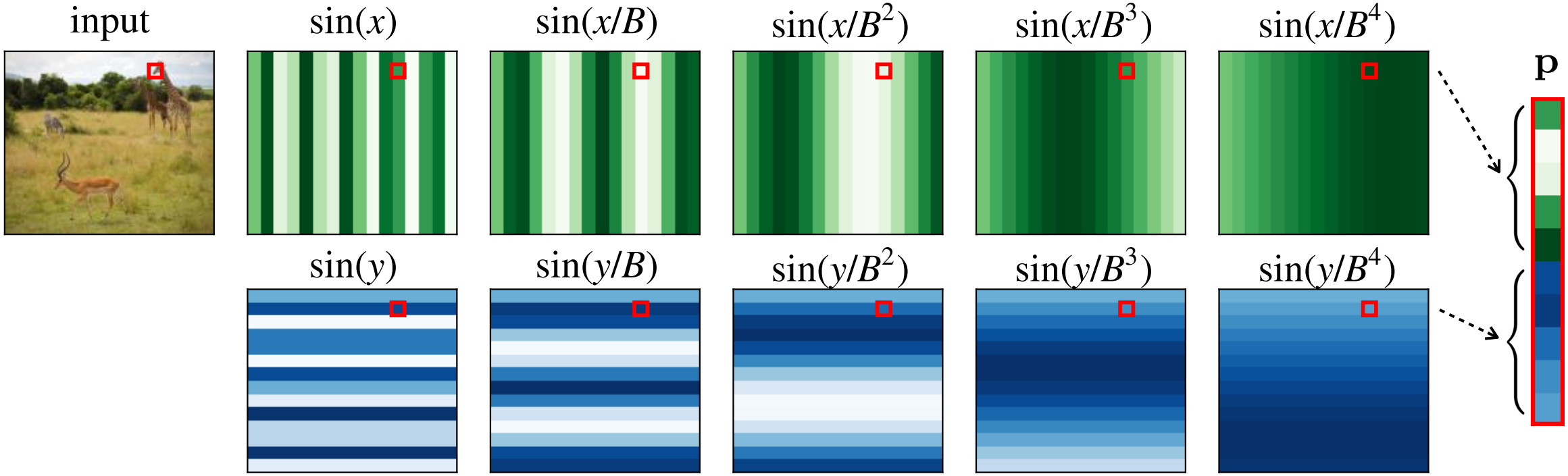
What if you don't want to be shift invariant?

1. Use an architecture that is not permutation invariant (e.g., MLP)
2. Add location information to the token code vectors — this is called **positional encoding**



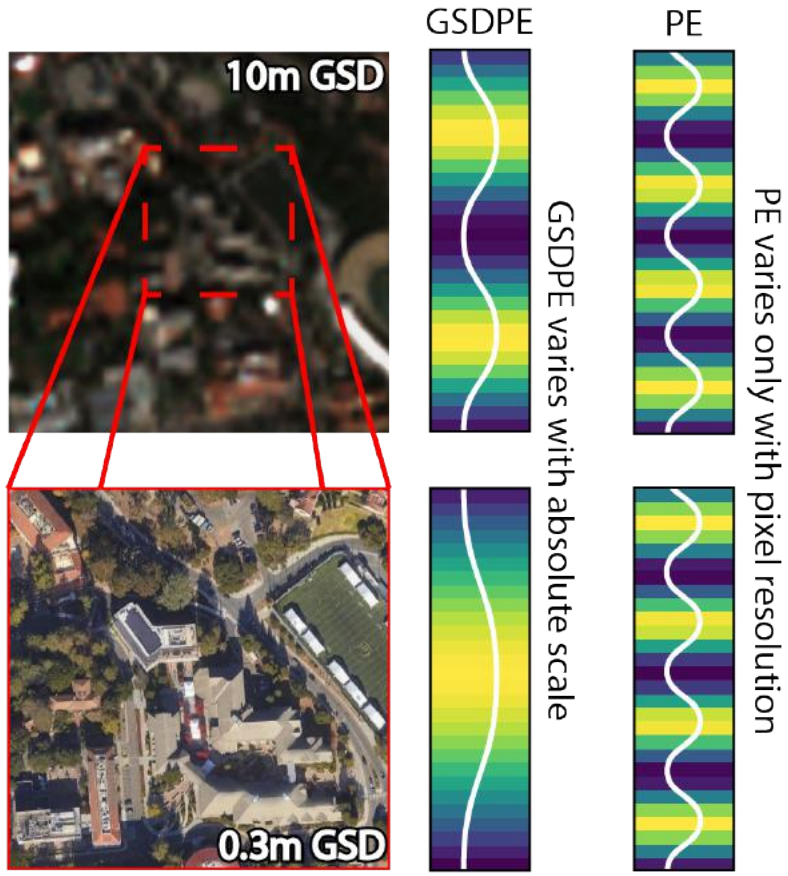
Fourier positional codes

- Represent coordinates on Fourier basis



Other positional encodings

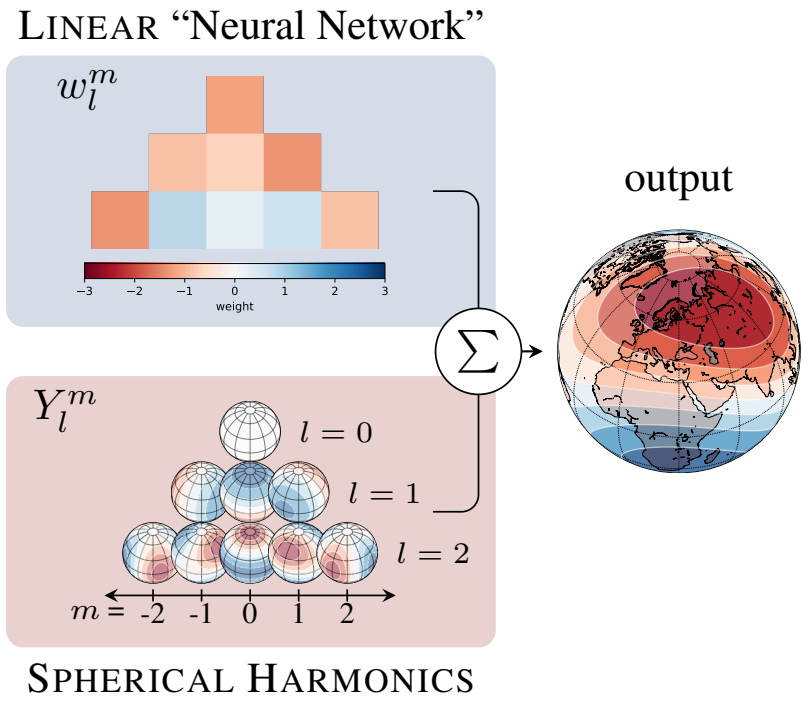
- ScaleMAE uses ground sample distance positional encoding to train a MAE across spatial scales of remote sensing data

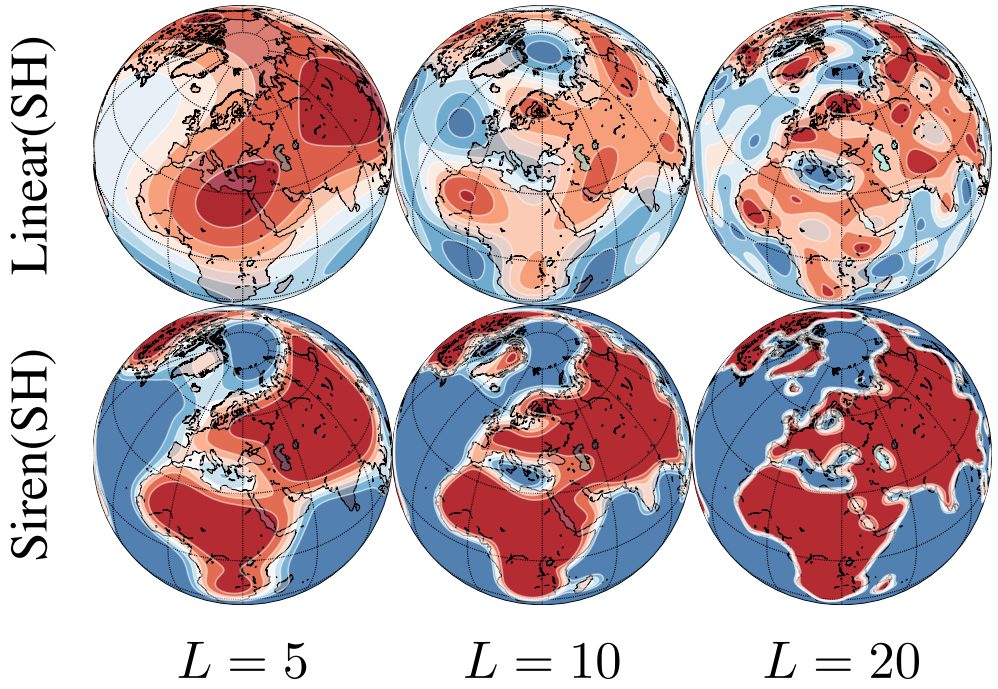


https://arxiv.org/abs/2212.14532

Other positional encodings

- Geographic location encoding with spherical harmonics and sinusoidal representation networks

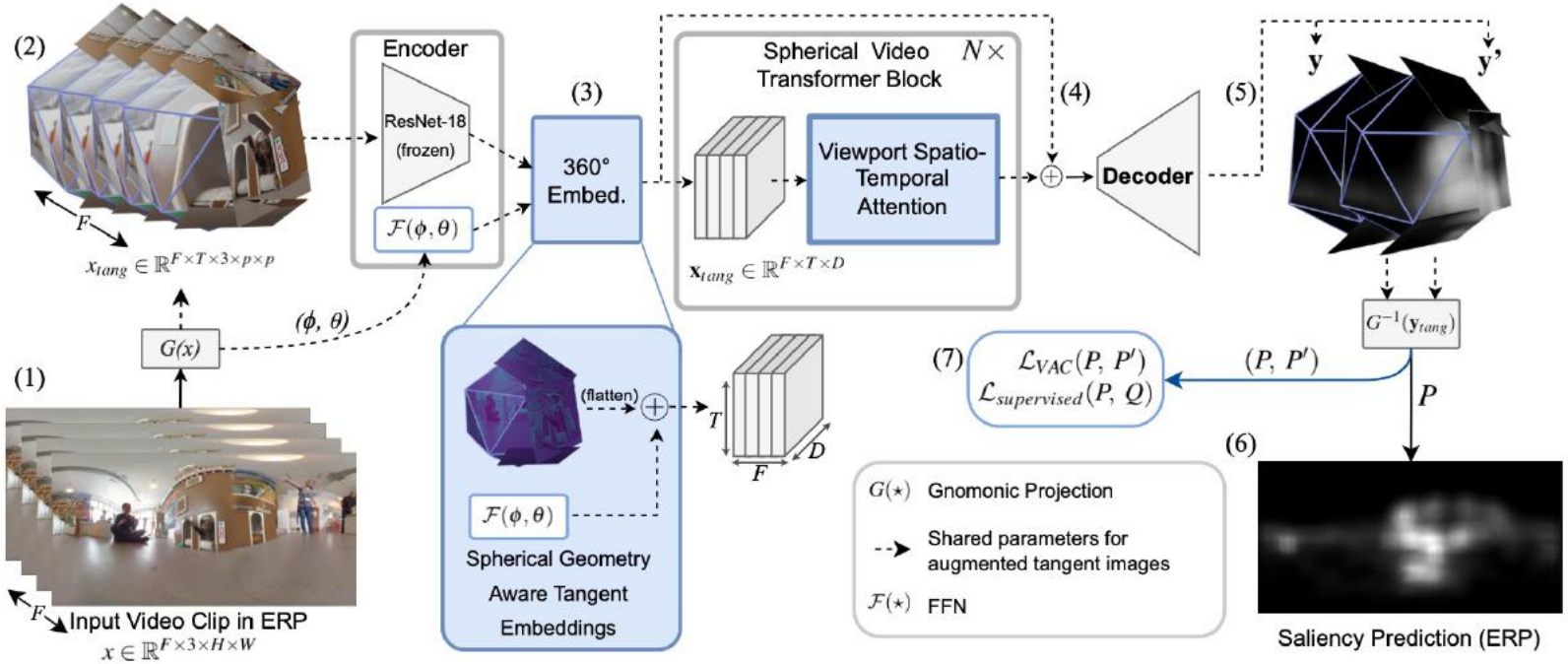




https://arxiv.org/abs/2310.06743

Other positional encodings

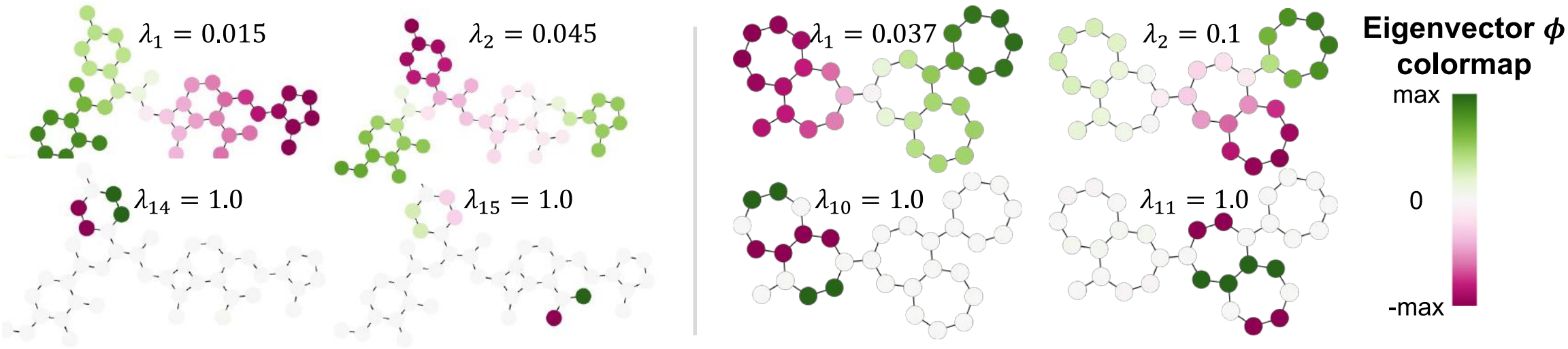
- Each tangent image is encoded and fused with spherical-geometry-aware position embeddings



https://arxiv.org/abs/2308.13004

Other positional encodings

- Laplacian positional encodings to encode node positions in a graph



https://arxiv.org/abs/2106.03893

Some fancy architectures and applications

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

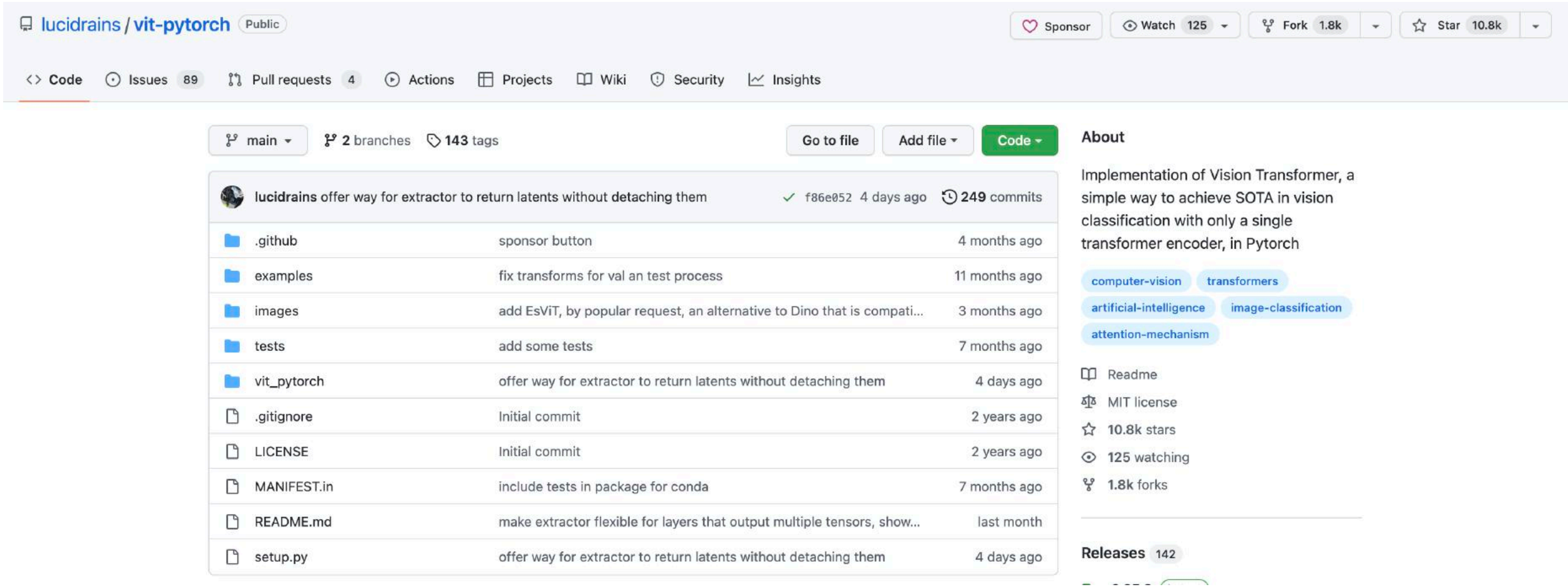
ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

1 INTRODUCTION

Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP). The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019). Thanks to Transformers' computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters (Brown et al., 2020; Lepikhin et al., 2020). With the models and datasets growing, there is still no sign of saturating performance.

https://arxiv.org/abs/2010.11929



https://github.com/lucidrains/vit-pytorch

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

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

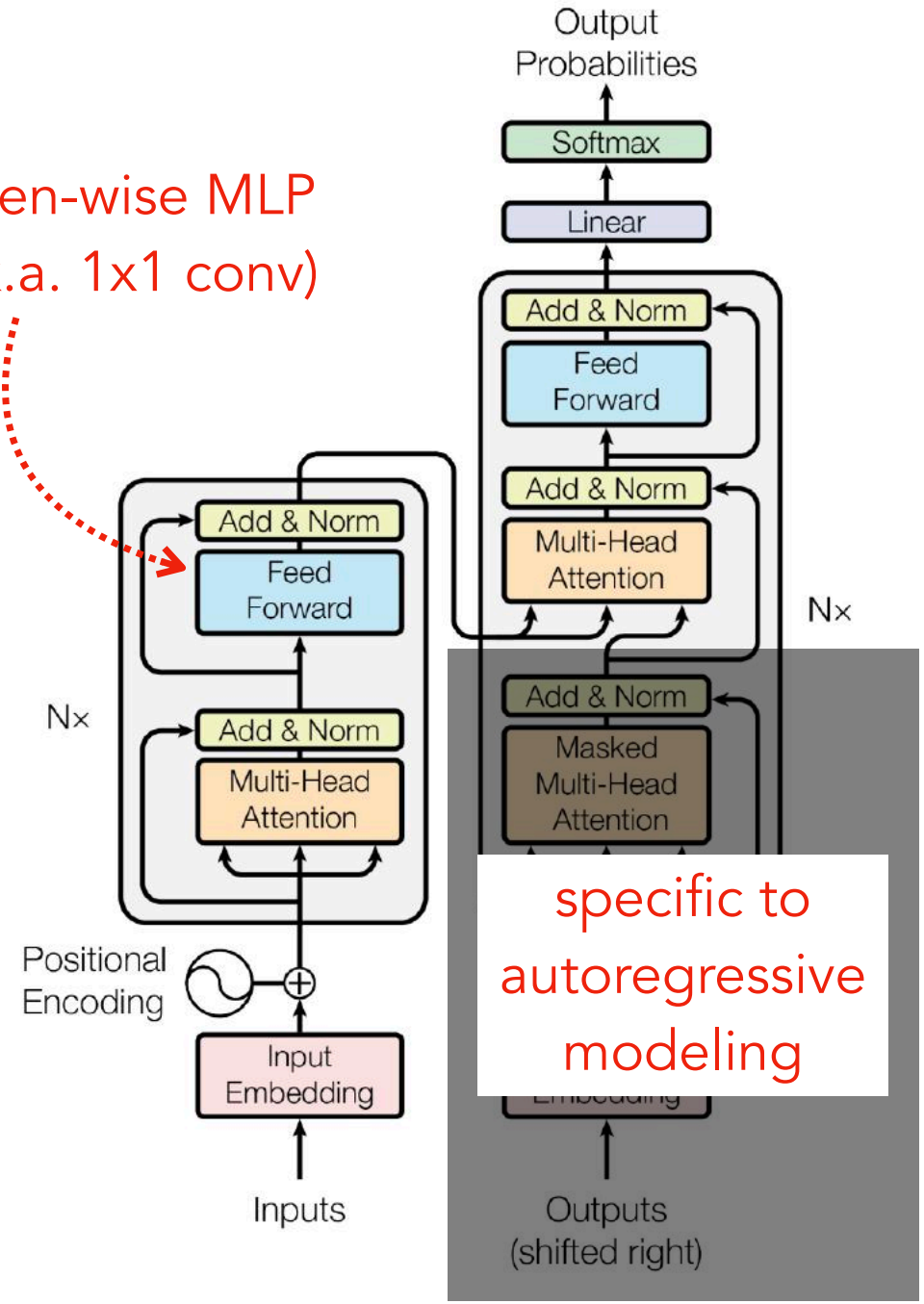
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

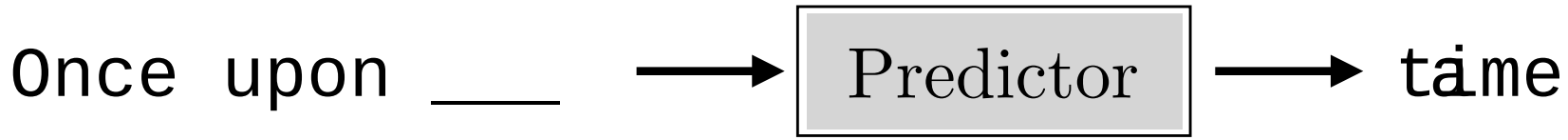
Abstract

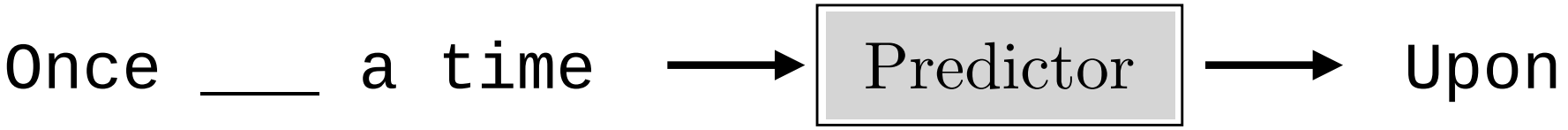
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

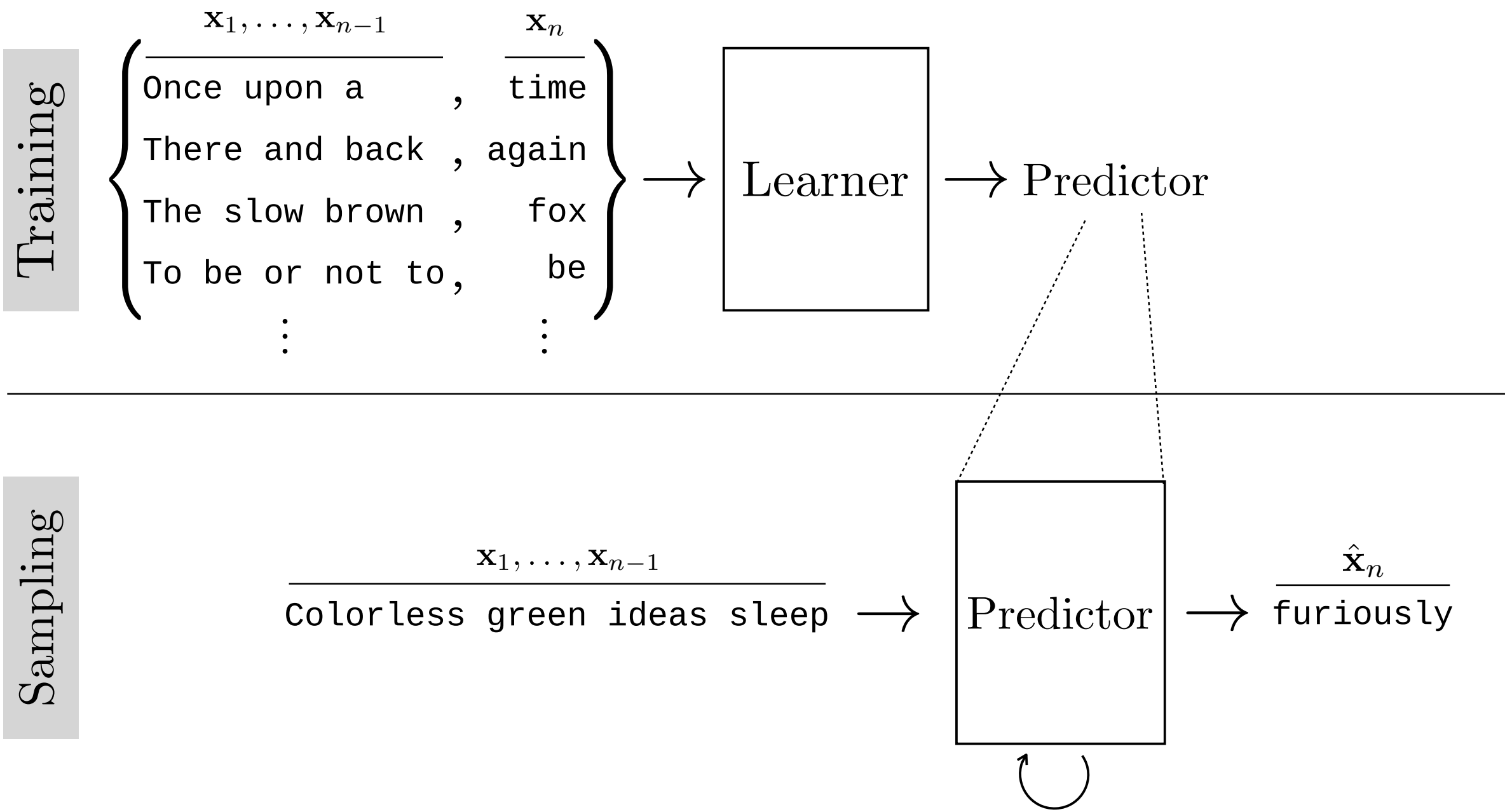
token-wise MLP
(a.k.a. 1x1 conv)



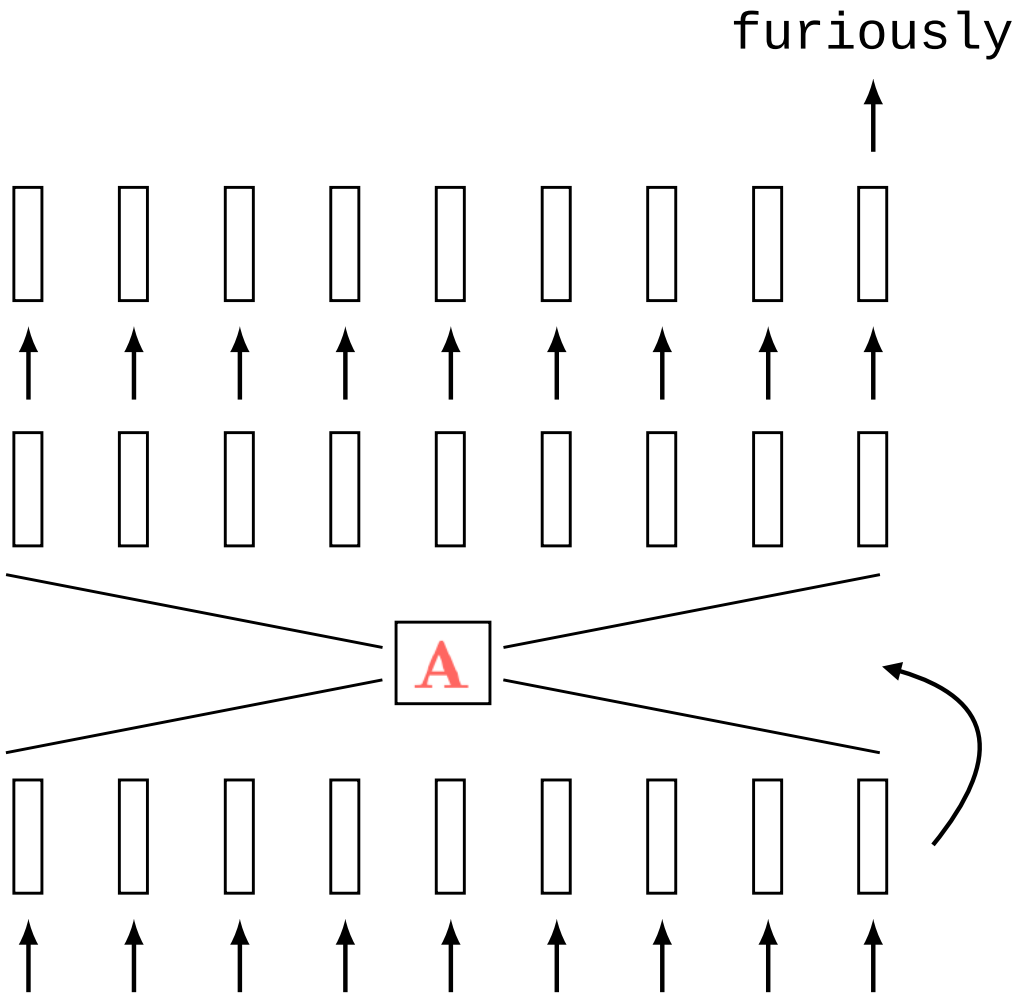
Autoregressive models



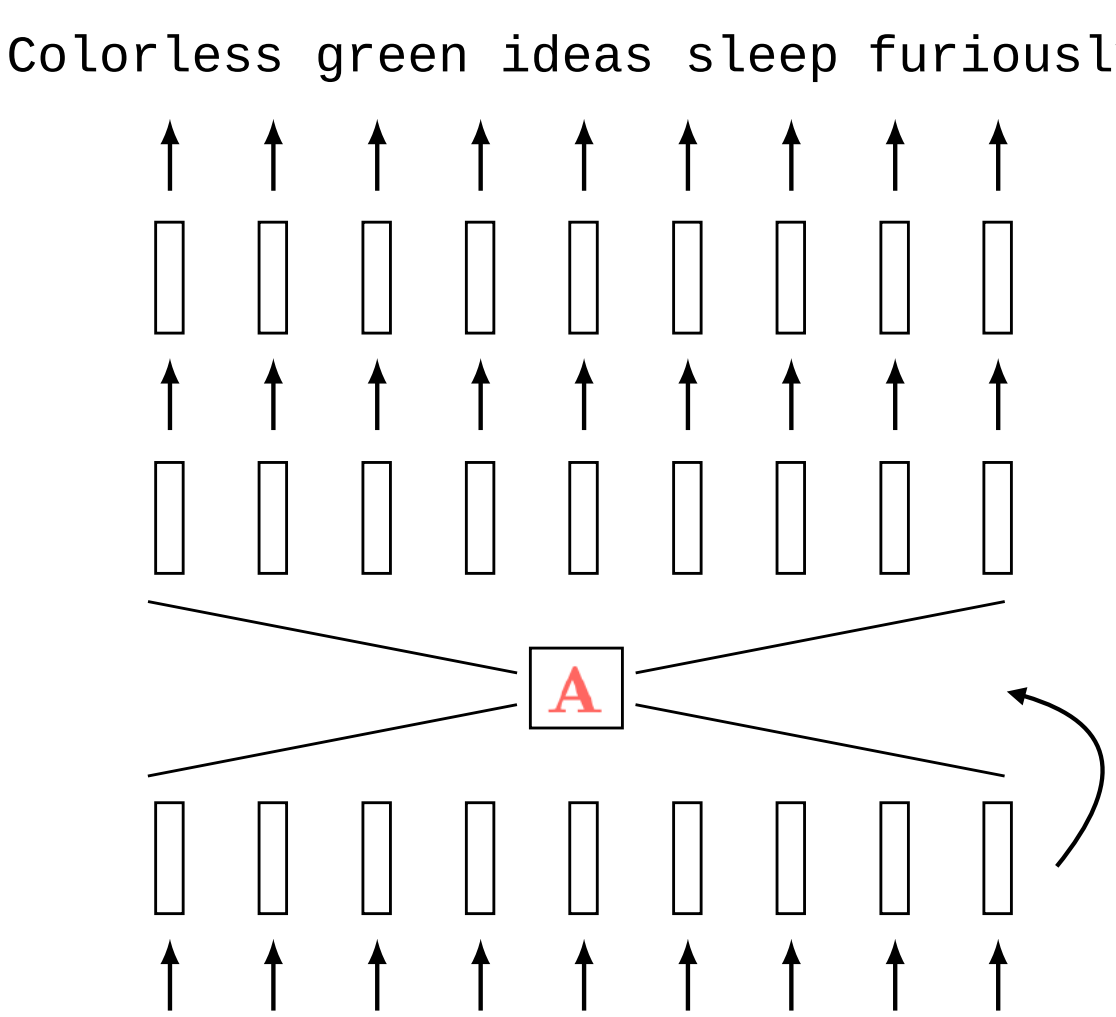


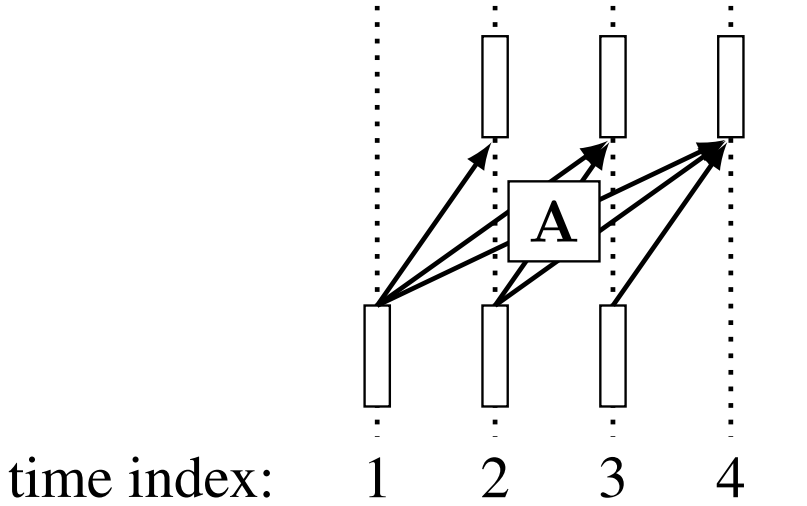

Training
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
$\mathbf{x}_n$
Once upon a , time
There and back , again
The slow brown , fox
To be or not to , be
Learner
Predictor
Sampling
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
Colorless green ideas sleep
Predictor
$\hat{\mathbf{x}}_n$
furiously

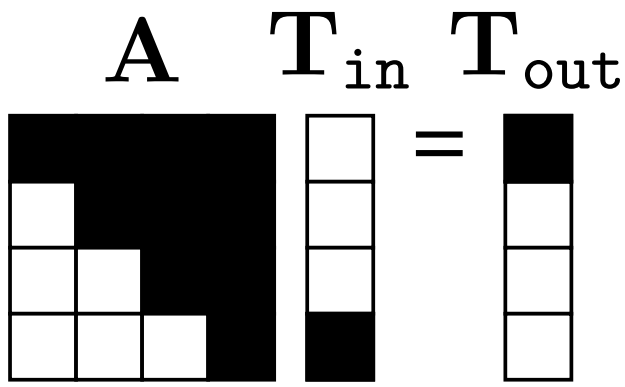
GPT (and many other related models)



GPT training (and many other related models)







karpathy / minGPT Public

Code | Issues 22 | Pull requests 11 | Actions | Projects | Security | Insights

master | 3 branches | 0 tags | Go to file | Add file | Code

karpathy Merge pull request #84 from ericjang/master ... 7218bcf on Aug 4 | 93 commits

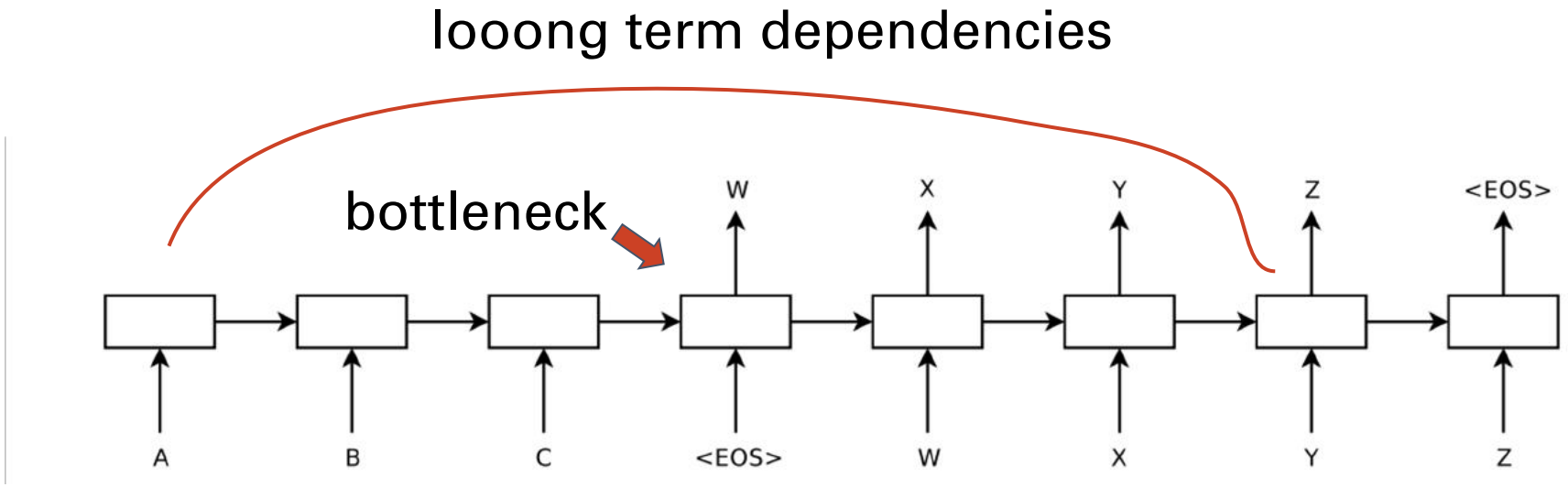
mingpt	Use XOR operator ^ for checking assertion `type_given XOR param...	2 months ago
projects	refactor sequence generation into the model and match the huggingf...	3 months ago
tests	add a refactored BPE encoder from openai. Basically I dont super tru...	3 months ago
.gitignore	tiny tweaks to printing and some function apis	4 months ago
LICENSE	mit license file	2 years ago
README.md	Add setup.py to allow mingpt to be used as a third-party library	2 months ago
demo.ipynb	refactor sequence generation into the model and match the huggingf...	3 months ago
generate.ipynb	add a refactored BPE encoder from openai. Basically I dont super tru...	3 months ago
mingpt.jpg	first commit, able to multigpu train fp32 GPTs on math and character...	2 years ago
setup.py	Add setup.py to allow mingpt to be used as a third-party library	2 months ago

README.md

minGPT

Transformers vs. Attention in RNNs

Problems with vanilla seq2seq



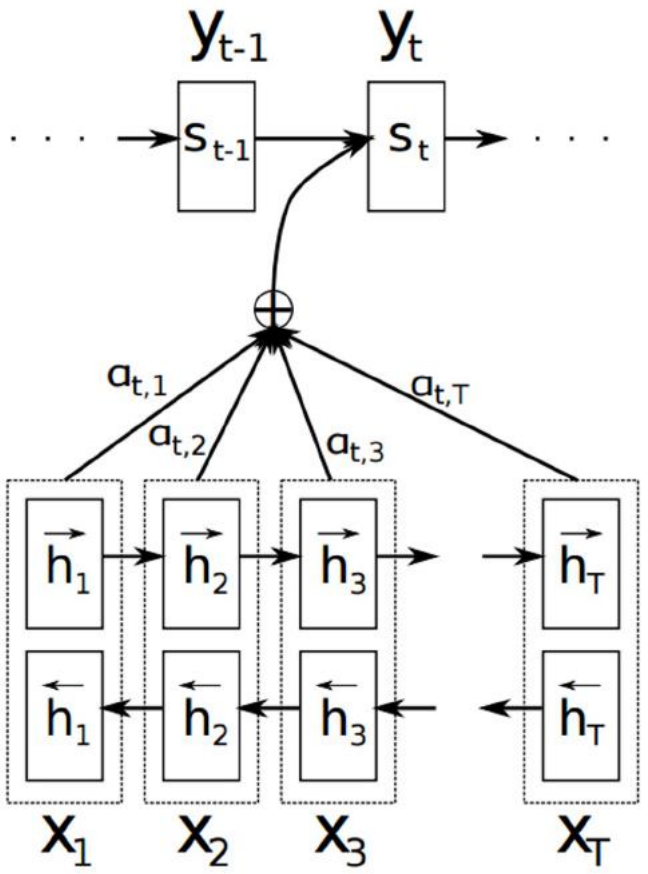
- training the network to encode 50 words in a vector is hard ⇒ very big models are needed
- gradients has to flow for 50 steps back without vanishing ⇒ training can be slow and require lots of data

Attention

lets decoder focus on the relevant hidden states of the encoder, avoids squeezing everything into the last hidden state ⇒ **no bottleneck**!

dynamically creates shortcuts in the computation graph that allow the gradient to flow freely ⇒ **shorter dependencies**!

best with a bidirectional encoder



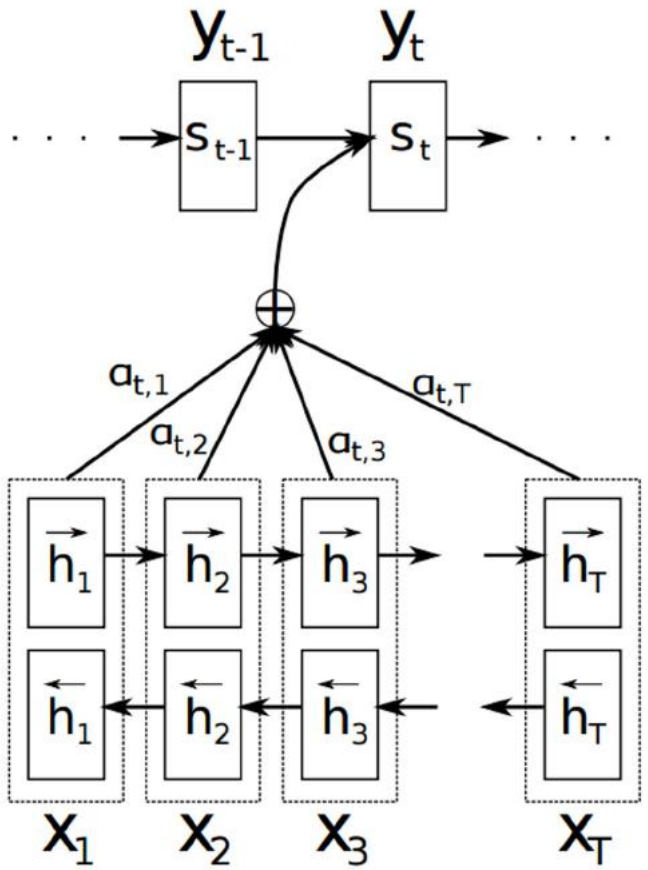
Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al, ICLR 2015

Attention - math 1

At each step the decoder consumes a different weighted combination of the encoder states, called **context vector** or **glimpse**.

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

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$



Attention - math 2

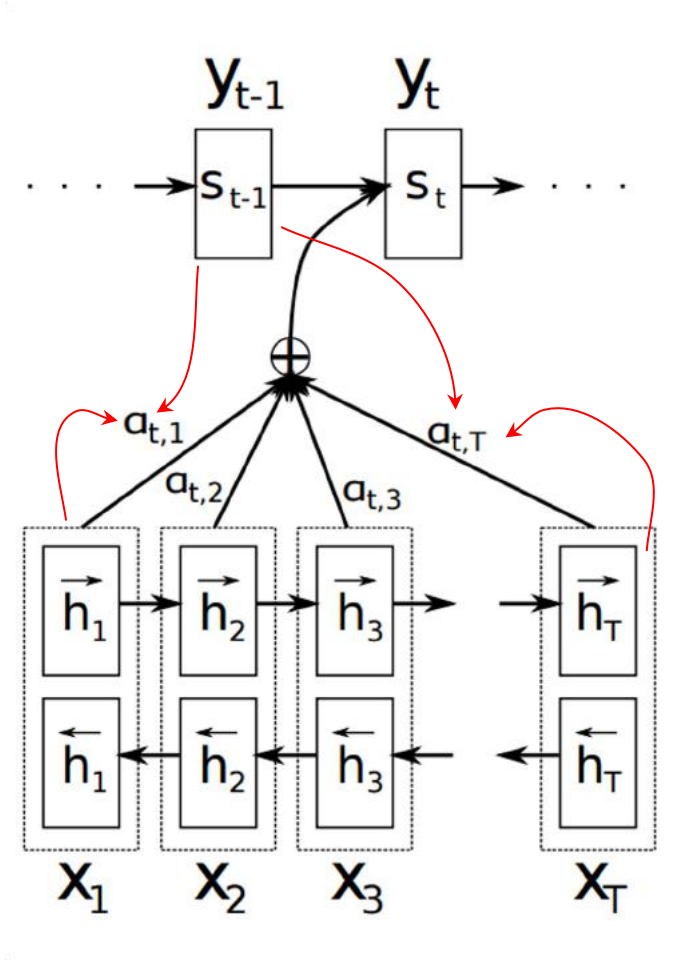
But where do the weights come from?
They are computed by another network!

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

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

The choice from the original paper is 1-layer MLP:

$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$

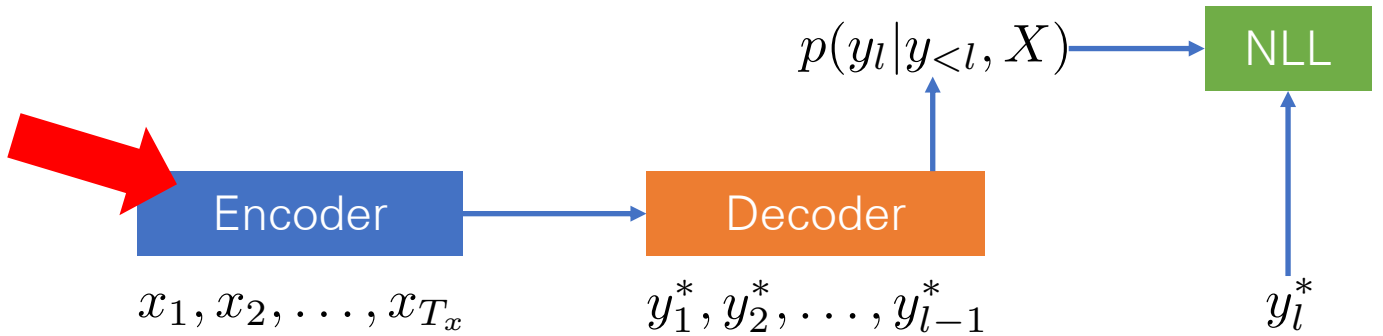


Parametrization – Recurrent Neural Nets

- Following Bahdanau et al. [2015]
- The encoder turns a sequence of tokens into a sequence of contextualized vectors.

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t], \text{ where } \overrightarrow{h}_t = \text{RNN}(x_t, \overrightarrow{h}_{t-1}), \overleftarrow{h}_t = \text{RNN}(x_t, \overleftarrow{h}_{t+1})$$

- The underlying principle behind recently successful contextualized embeddings
  - ELMo [Peters et al., 2018], BERT [Devlin et al., 2019] and all the other muppets



Parametrization – Recurrent Neural Nets

- Following Bahdanau et al. [2015]
- The decoder consists of three stages
  1. Attention: attend to a small subset of source vectors
  2. Update: update its internal state
  3. Predict: predict the next token

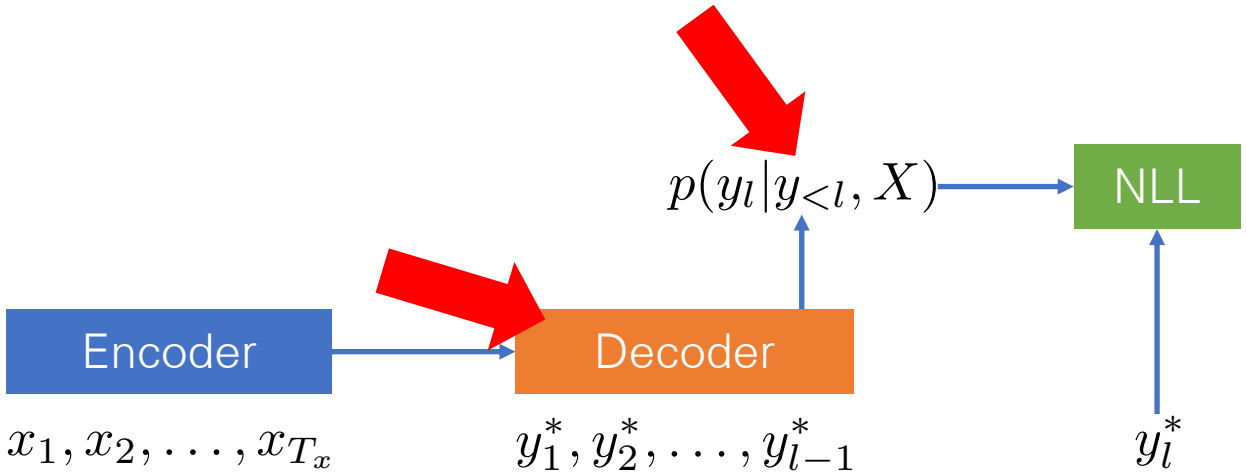
$$\alpha_{t'} \propto \exp(\mathrm{ATT}(h_{t'}, z_{t-1}, y_{t-1}))$$

$$c_t = \sum_{t'=1}^{T_x} \alpha_{t'} h_{t'}$$

$$z_t = \mathrm{RNN}([y_{t-1}; c_t], z_{t-1})$$

$$p(y_t = v | y_{<t}, X) \propto \exp(\mathrm{OUT}(z_t, v))$$

- Attention has become the core component in many recent advances
  - Transformers [Vaswani et al., 2017], …

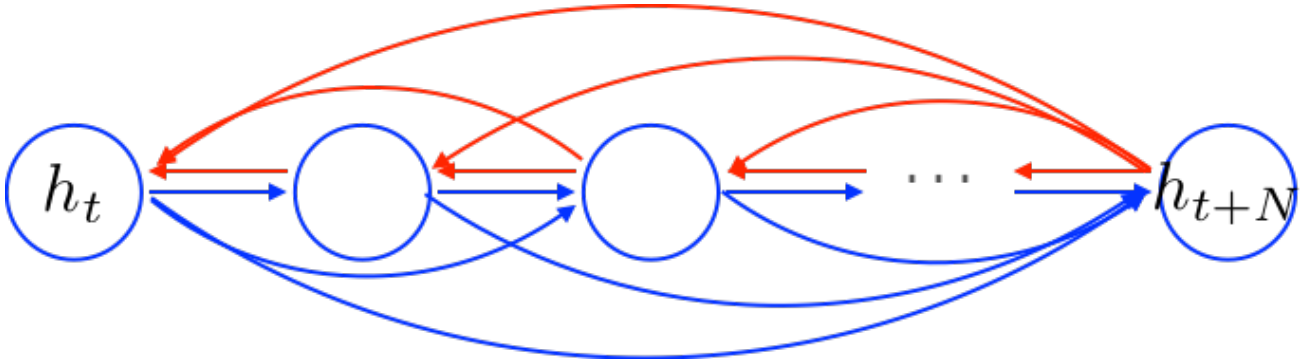


Side-note: gated recurrent units to attention

- A key idea behind LSTM and GRU is the additive update

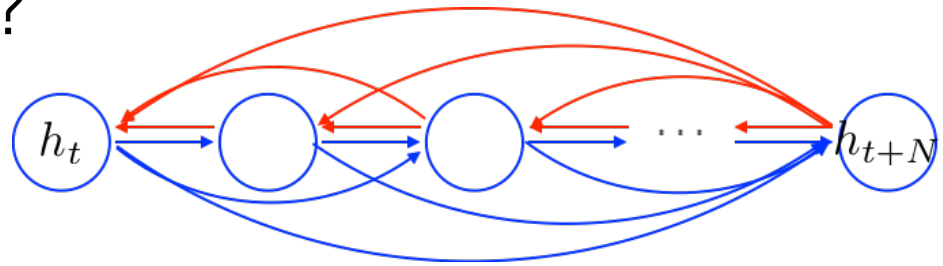
$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tilde{h}_t, \text{ where } \tilde{h}_t = f(x_t, h_{t-1})$$

- This additive update creates linear short-cut connections



Side-note: gated recurrent units to attention

- What are these shortcuts?



- If we unroll it, we see it's a weighted combination of all previous hidden vectors:

$$
\begin{aligned}
h_t =& u_t \odot h_{t-1} + (1-u_t) \odot \tilde{h}_t, \\
=& u_t \odot (u_{t-1} \odot h_{t-2} + (1-u_{t-1}) \odot \tilde{h}_{t-1}) + (1-u_t) \odot \tilde{h}_t, \\
=& u_t \odot (u_{t-1} \odot (u_{t-2} \odot h_{t-3} + (1-u_{t-2}) \odot \tilde{h}_{t-2}) + (1-u_{t-1}) \odot \tilde{h}_{t-1}) + (1-u_t) \odot \tilde{h}_t, \\
& \vdots \\
=& \sum_{i=1}^{t} \left(\prod_{j=i}^{t-i+1} u_j \right) \left(\prod_{k=1}^{i-1} (1-u_k) \right) \tilde{h}_i
\end{aligned}
$$

Side-note: gated recurrent units to attention

1. Can we "free" these dependent weights?
2. Can we "free" candidate vectors?
3. Can we separate keys and values?
4. Can we have multiple attention heads?

$$h_t = \sum_{i=1}^{t} \left(\prod_{j=i}^{t-i+1} u_j \right) \left(\prod_{k=1}^{i-1} (1 - u_k) \right) \tilde{h}_i \quad \textbf{0}$$

$$h_t = \sum_{i=1}^{t} \alpha_i \tilde{h}_i, \text{ where } \alpha_i \propto \exp(\text{ATT}(\tilde{h}_i, x_t)) \quad \textbf{1}$$

$$h_t = \sum_{i=1}^{t} \alpha_i f(x_i), \text{ where } \alpha_i \propto \exp(\text{ATT}(f(x_i), x_t)) \quad \textbf{2}$$

$$h_t = \sum_{i=1}^{t} \alpha_i V(f(x_i)), \text{ where } \alpha_i \propto \exp(\text{ATT}(K(f(x_i)), Q(x_t))) \quad \textbf{3}$$

$$h_t = [h_t^1; \cdots; h_t^K], \text{ where } h_t^k = \sum_{i=1}^{t} \alpha_i^k V^k(f(x_i)), \text{ where } \alpha_i^k \propto \exp(\text{ATT}(K^k(f(x_i)), Q^k(x_t))) \quad \textbf{4}$$

→ Transformers

Summary

- attention is used to focus on parts of inputs/outputs
- it can be content/location based and hard/soft
- it’s three main distinct uses are
  - connecting encoder and decoder in sequence-to-sequence task
  - achieving scale-invariance and focus in image processing
  - self-attention can be a basic building block for neural nets, often replacing RNNs and CNNs [recent research, take it with a grain of salt]

Next lecture:
Autoregressive Models