Illustration: DeepMind

CMP784

DEEP LEARNING

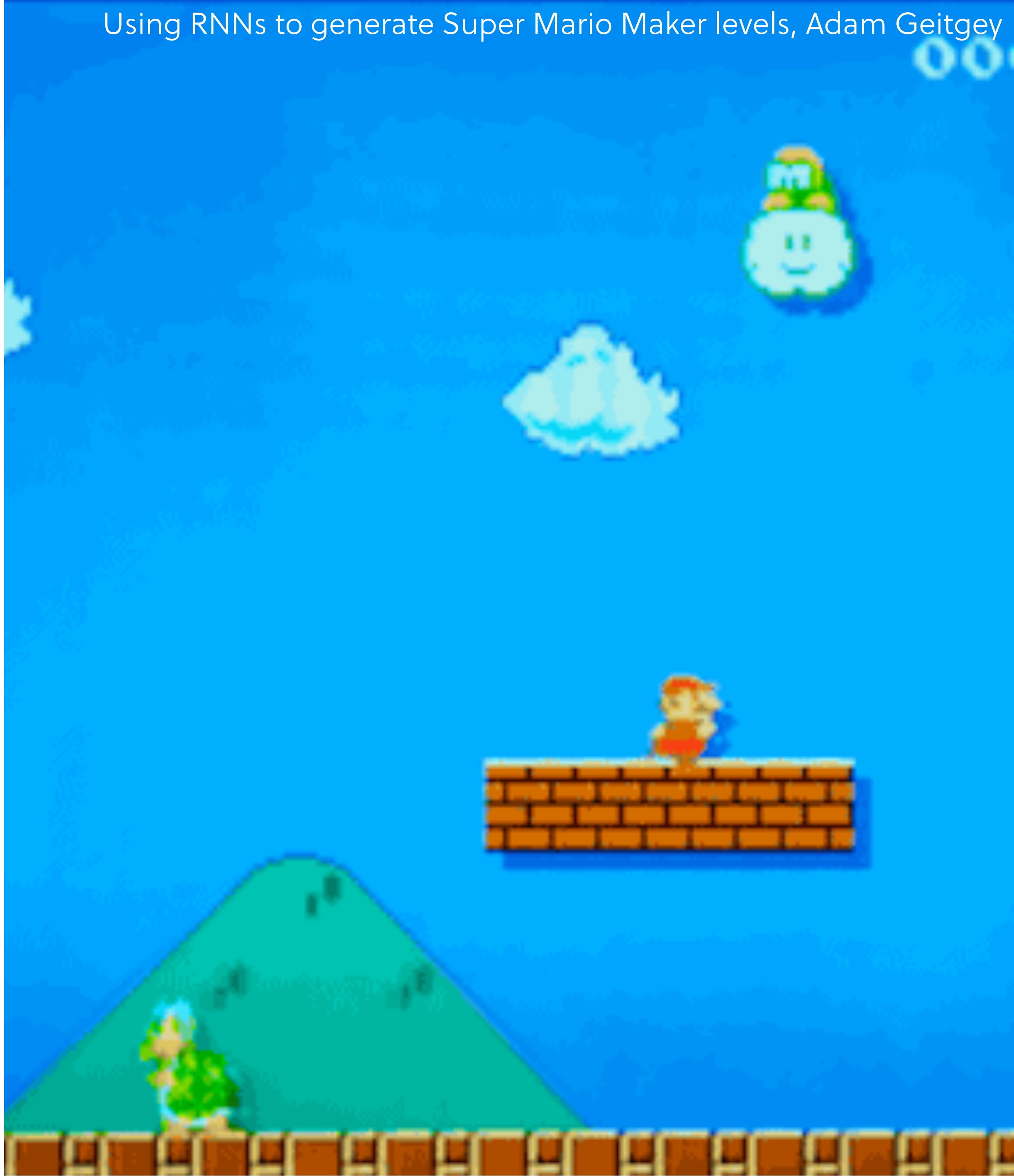
Lecture #08 – Attention and Memory

Erkut Erdem // Hacettepe University // Fall 2023

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Previously on CMP784

- Sequence modeling
- Recurrent Neural Networks (RNNs)
- The Vanilla RNN unit
- How to train RNNs
- The Long Short-Term Memory (LSTM) unit and its variants
- Gated Recurrent Unit (GRU)



Lecture overview

- Content-based attention
- Location-based attention
- Soft vs. hard attention
- Show, Attend and Tell
- Self-attention and Transformer networks
- Vision Transformers
- Pretraining during transformers

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

— Dzmitry Bahdanau's IFT 6266 slides

— Graham Neubig's CMU CS11-747 Neural Networks for NLP class

— Mateusz Malinowski's lecture on Attention-based Networks

— Yoshua Bengio's talk on From Attention to Memory and towards Longer-Term Dependencies

— Kyunghyun Cho's slides on neural sequence modeling

— Arian Hosseini's IFT 6135 slides

— Hongsheng Li's ELEG5491 class

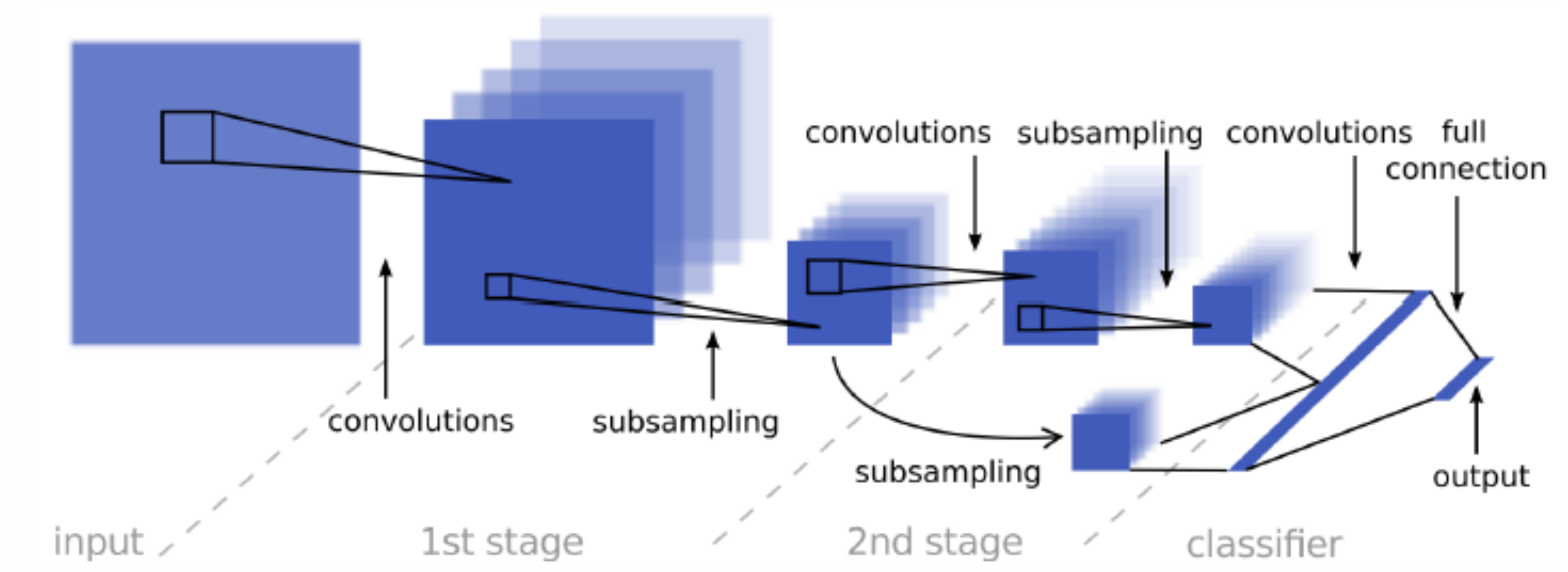
— Justin Johnson's EECS 498/598 class

— Jacob Devlin's slides on transformers

— Lucas Beyer's slides on transformers

— Philip Isola and Stefanie Jegelka's MIT 6.S898 Deep Learning class

Deep Learning for Vision



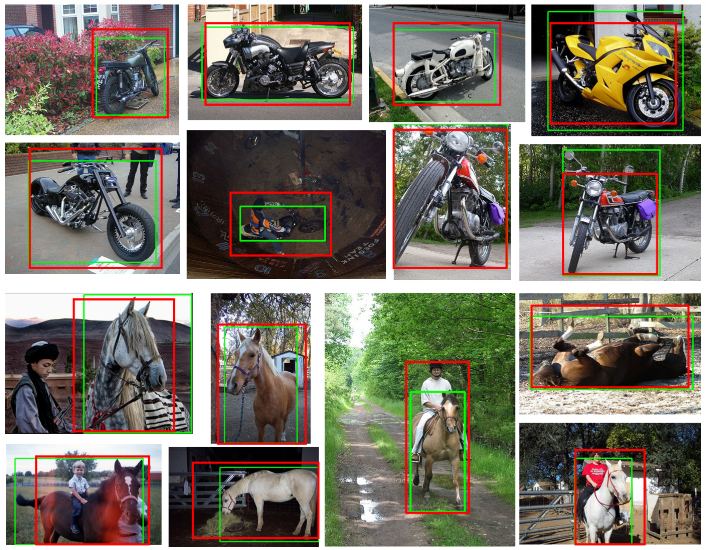
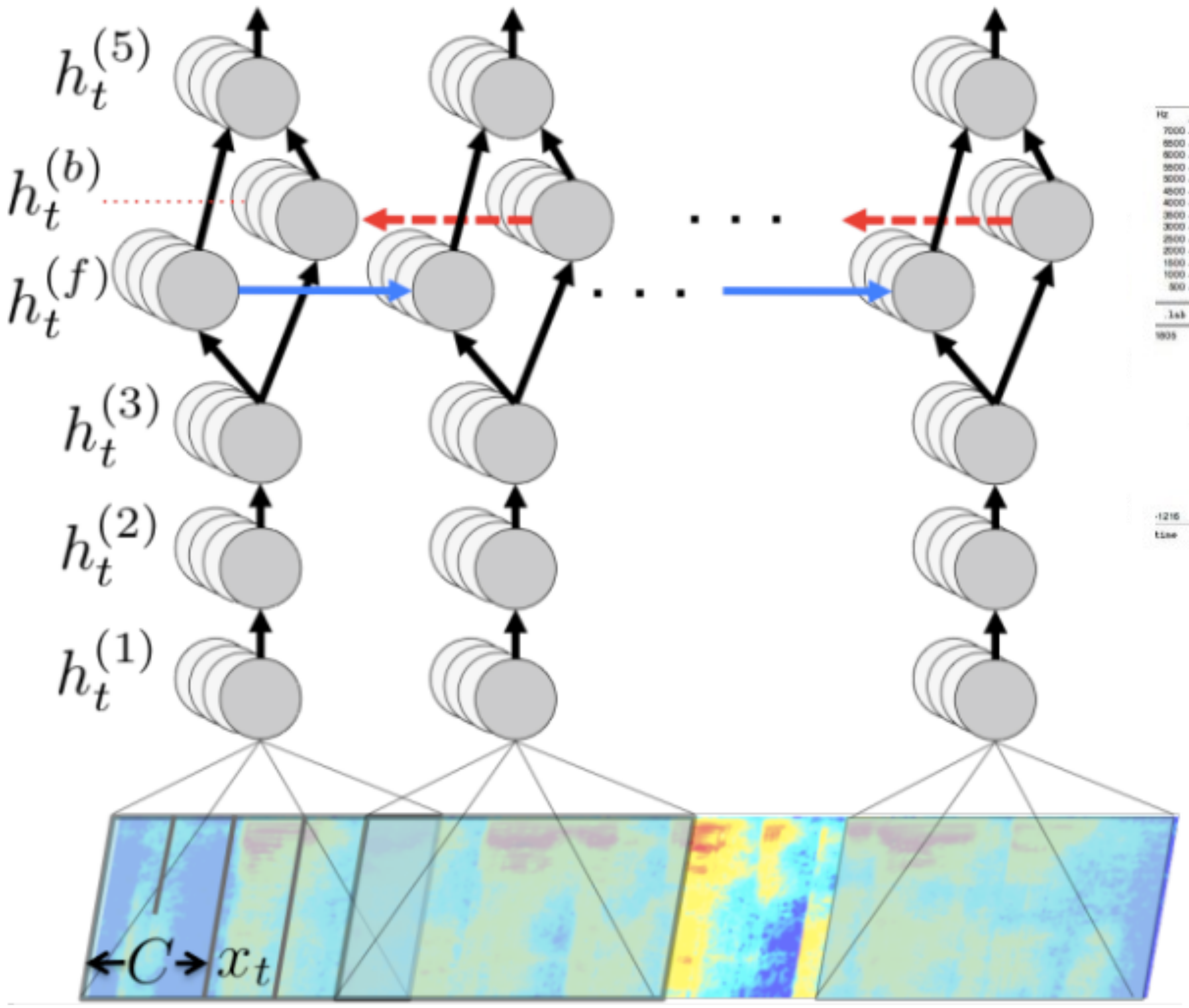
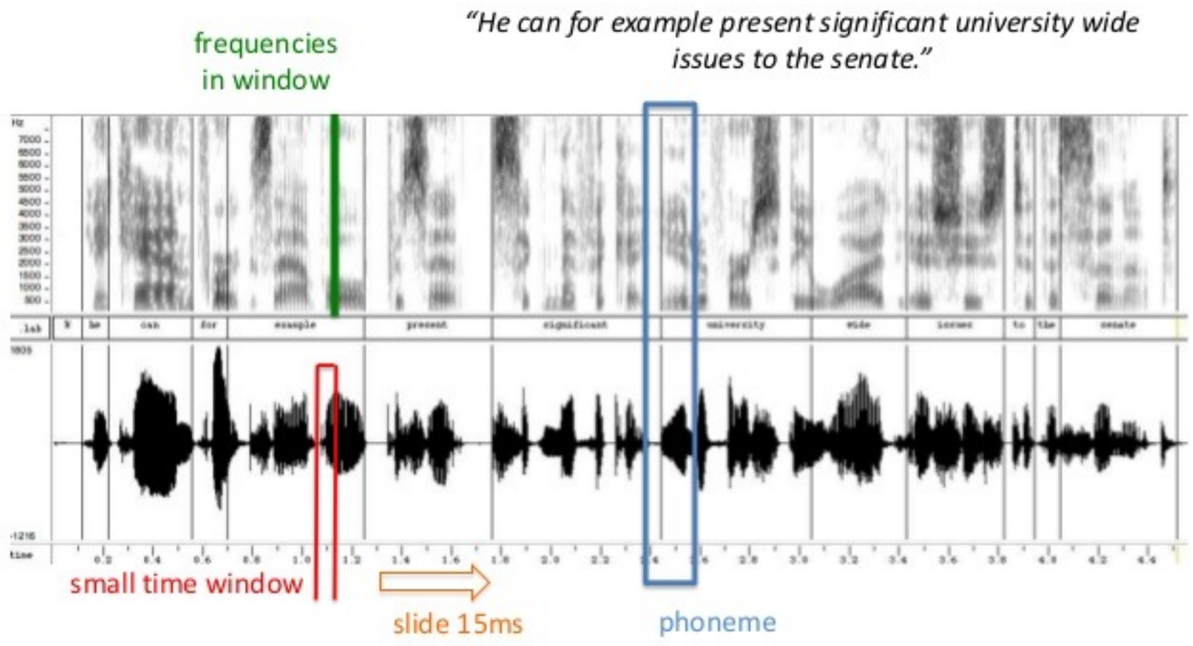


Figure credit: Xiaogang Wang

Deep Learning for Speech



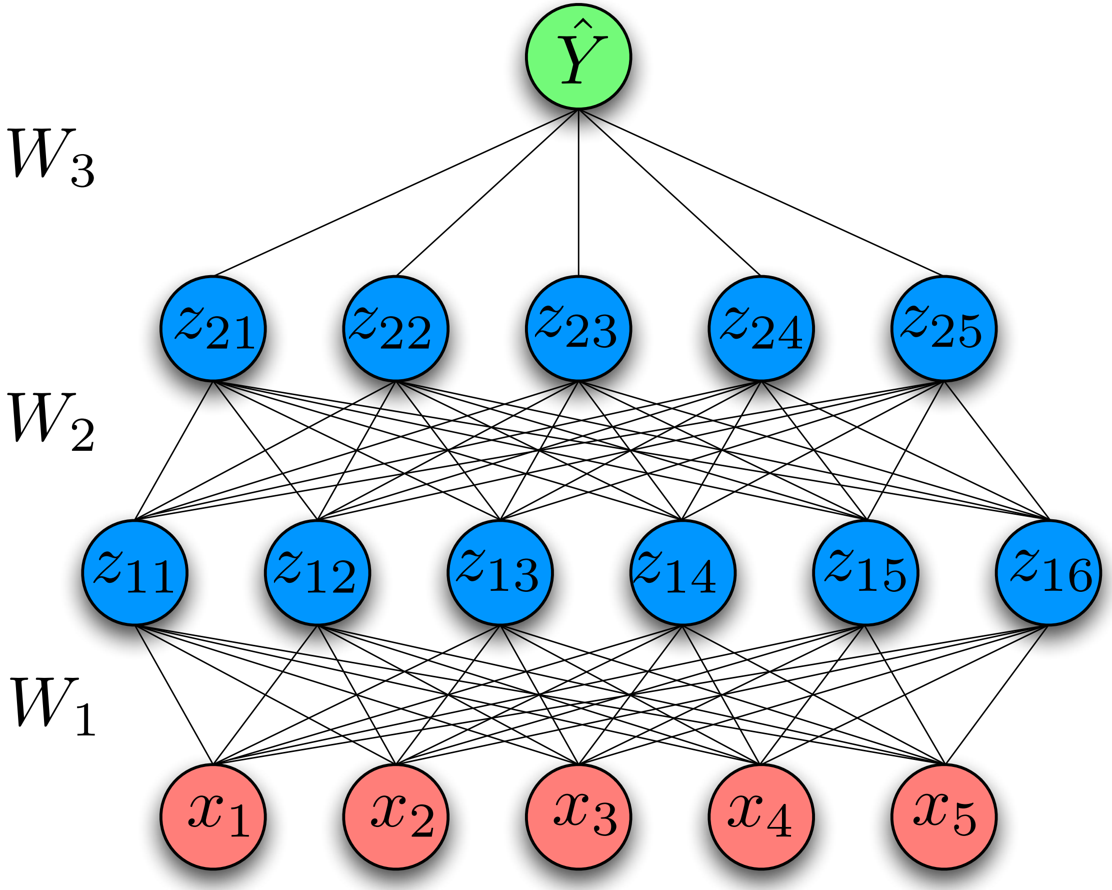


Spectrogram: window in time -> vector of frequences; slide; repeat

Figure credit: NVidia

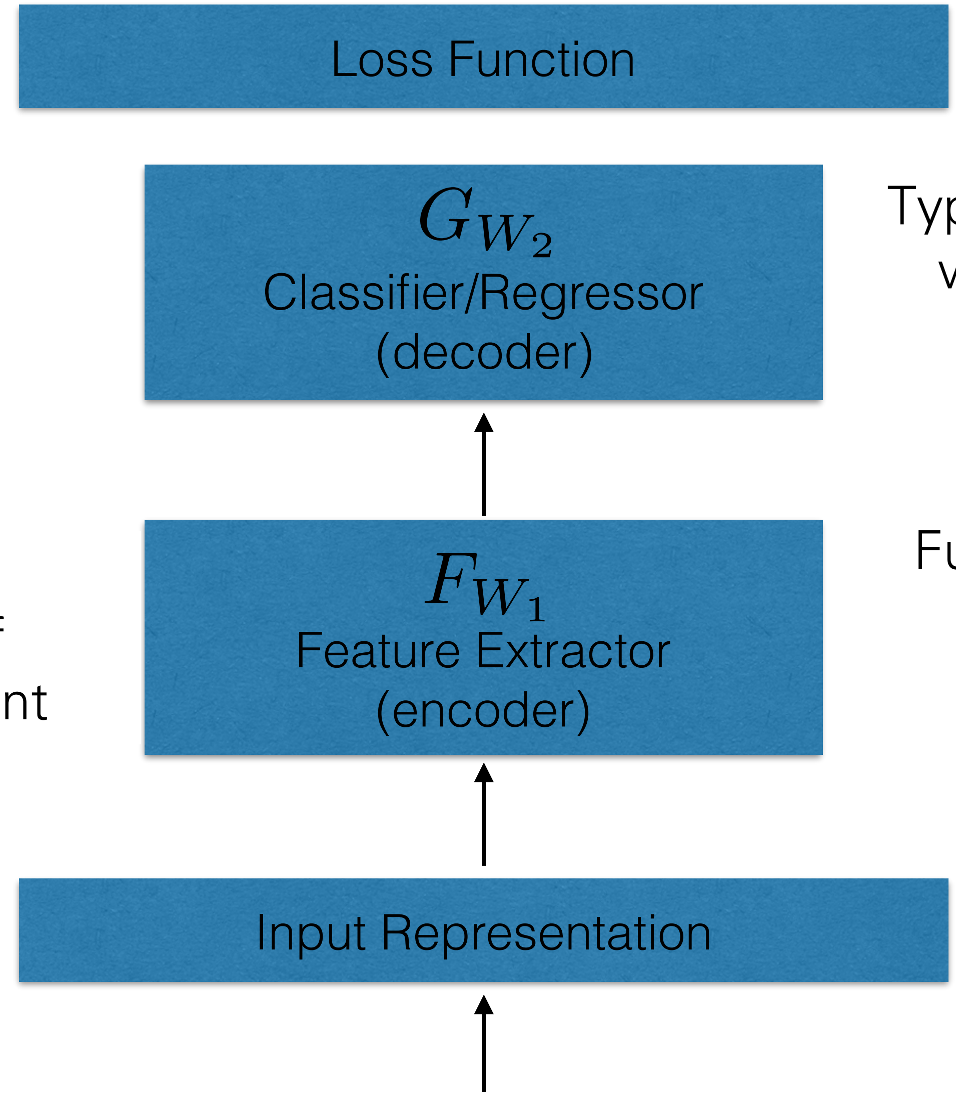
Deep Learning for Text

positive



"The movie was not bad at all. I had fun."

Deep Models



Typically a Linear Projection
with some non-linearity
(log-soft-max)

can be seen as
a prior on the type of
transformation you want

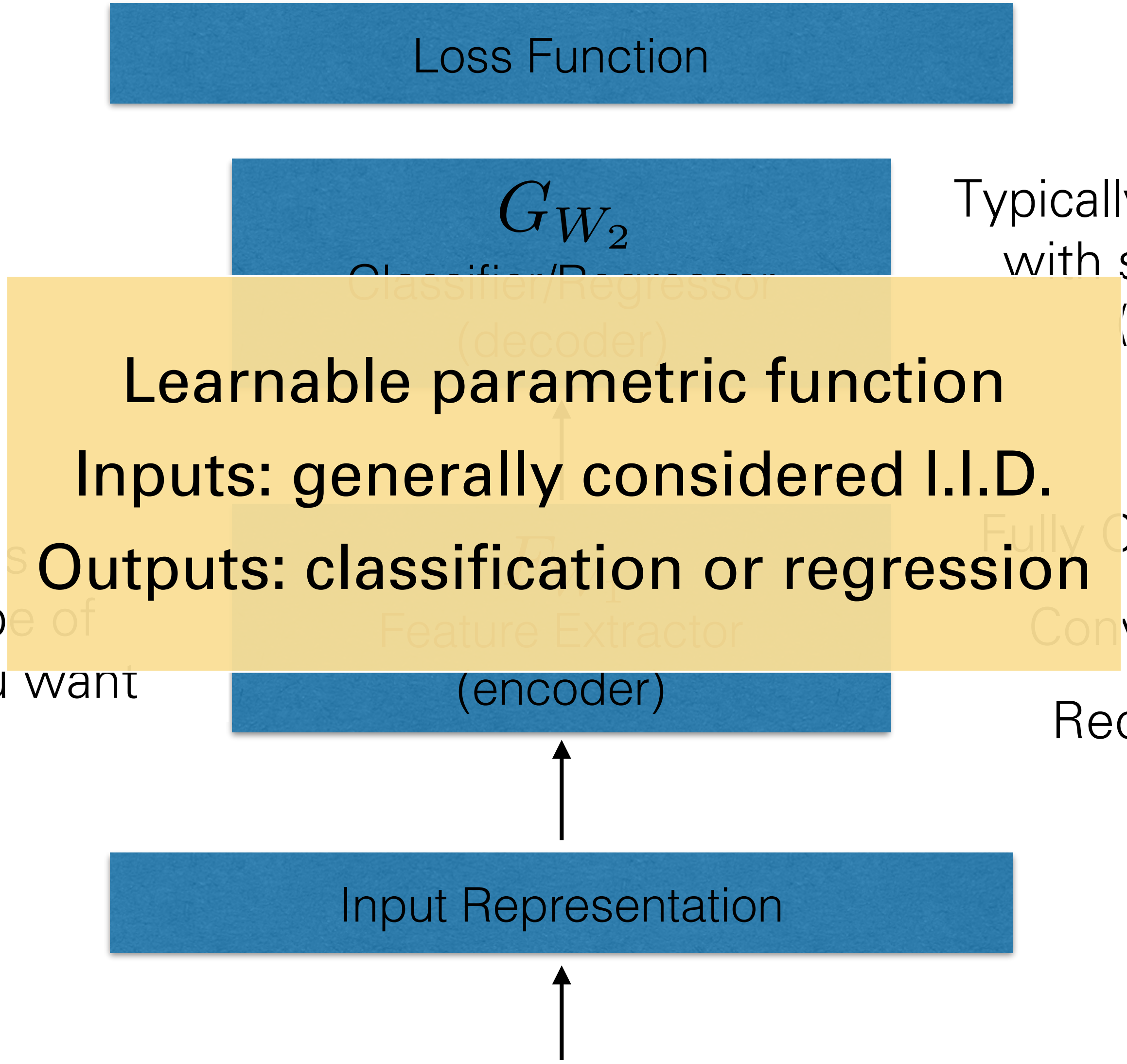
Fully Connected Network

Convolution Network

Recurrent Network

“The movie was not bad at all. I had fun.”

Deep Models



Typically a Linear Projection with some non-linearity (log-soft-max)

can be seen as a prior on the type of transformation you want

Fully Connected Network

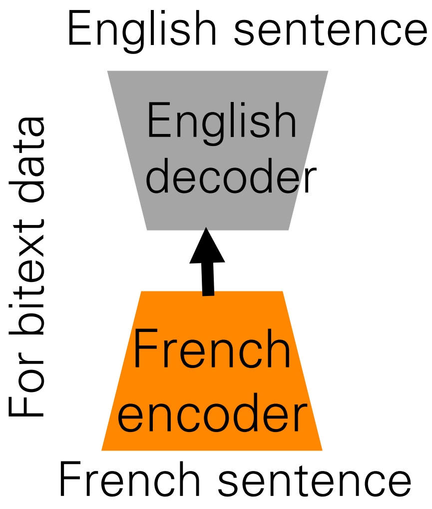
Convolution Network

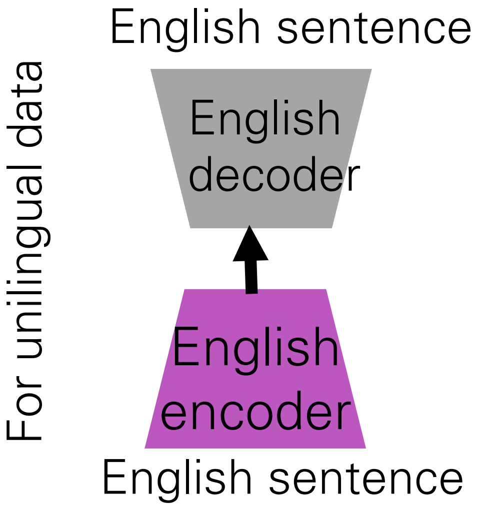
Recurrent Network

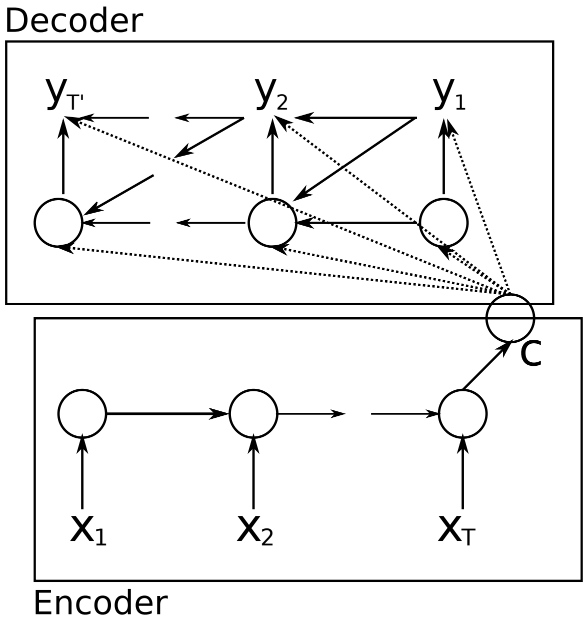
“The movie was not bad at all. I had fun.”

Encoder-Decoder Framework

- Intermediate representation of meaning
  = ‘universal representation’
- Encoder: from word sequence to sentence representation
- Decoder: from representation to word sequence distribution





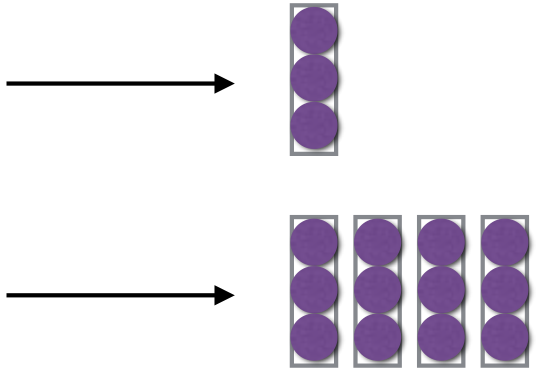


Sequence Representations

- But what if we could use multiple vectors, based on the length of the sequence

this is an example

this is an example



Attention Models in Deep Learning

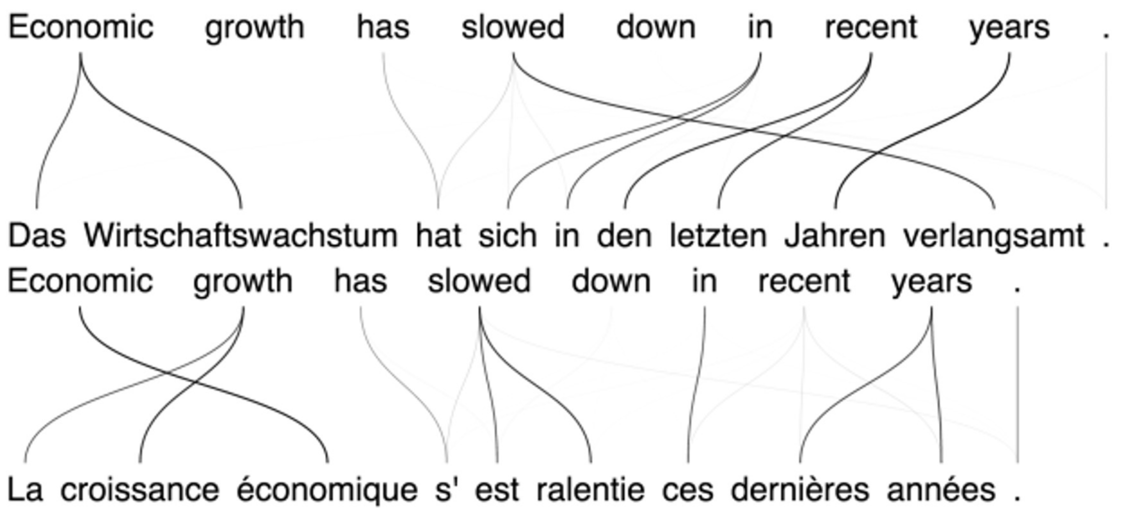
A lot of things are called “attention” these days…

1. Attention (alignment) models used in applications of deep supervised learning with **variable-length** inputs and outputs (typical sequential).
2. Models of visual attention that process a region of an image at high resolution or the whole image at low resolution.
3. Internal self-attention mechanisms can be used to replace recurrent and convolutional networks for sequential data.
4. Addressing schemes of memory-augmented neural networks

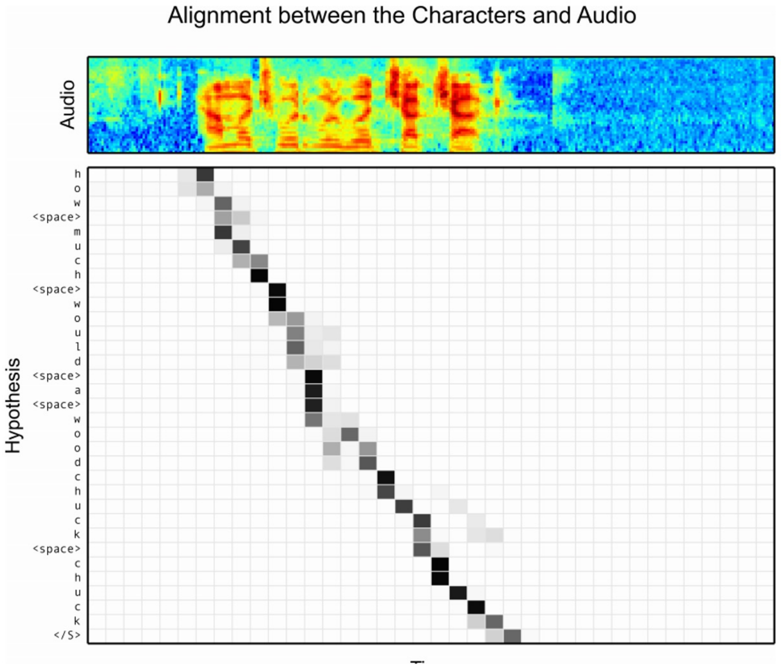
The shared idea: **focus on the relevant parts of the input (output).**

Attention in Deep Learning Applications [to Language Processing]

machine translation



speech recognition



speech synthesis, summarization, … any sequence-to-sequence (seq2seq) task

Traditional deep learning approach

input → d-dimensional feature vector → layer_1 → .... → layer_k → output

Good for: image classification, phoneme recognition, decision-making in reflex agents (ATARI)

Less good for: text classification

Not really good for: … everything else?!

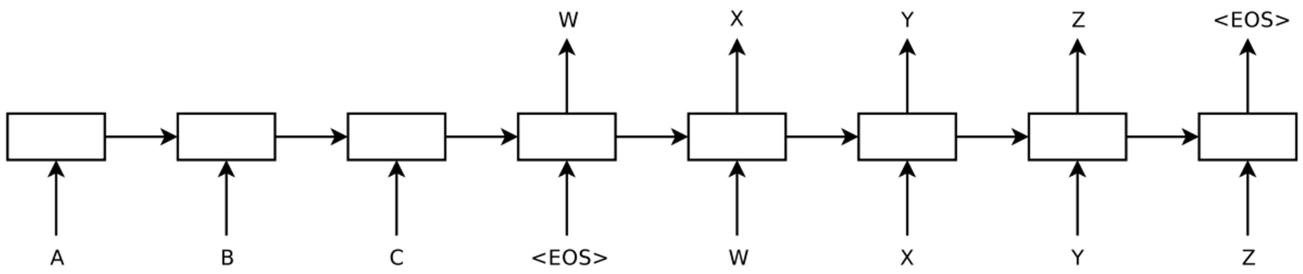
Example: Machine Translation

["An", "RNN", "example", "."] → ["Un", "example", "de", "RNN", "."]

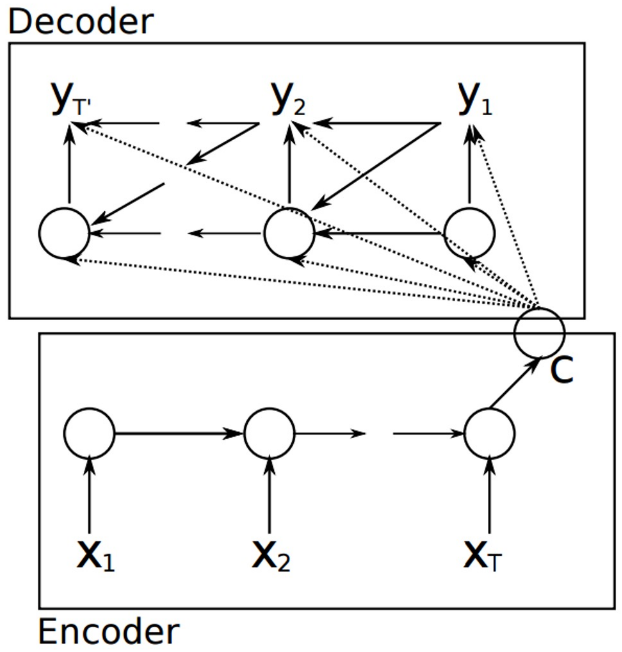
Machine translation presented a challenge to vanilla deep learning

- input and output are sequences
- the lengths vary
- input and output may have different lengths
- no obvious correspondence between positions in the input and in the output

Vanilla seq2seq learning for machine translation



input sequence output sequence

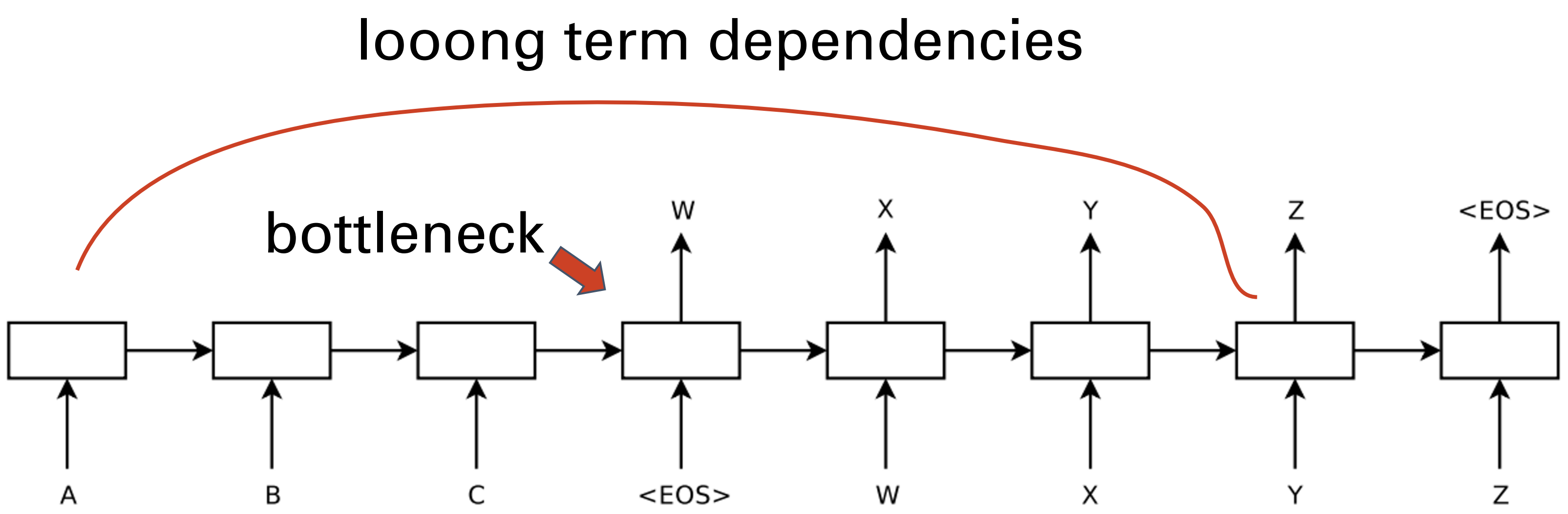


$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

fixed size representation

Recurrent Continuous Translation Models, Kalchbrenner et al, EMNLP 2013
Sequence to Sequence Learning with Recurrent Neural Networks, Sutskever et al., NIPS 2014
Learning Phrase Representations using RNN Encoder–Decoder for
Statistical Machine Translation, Cho et al., EMNLP 2014

Problems with vanilla seq2seq



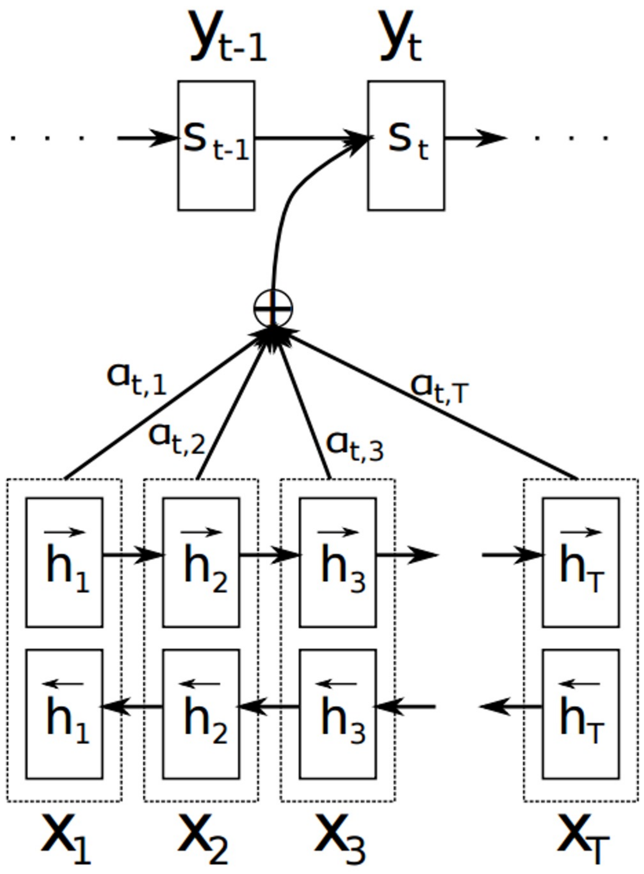
- training the network to encode 50 words in a vector is hard ⇒ very big models are needed
- gradients has to flow for 50 steps back without vanishing ⇒ training can be slow and require lots of data

Soft 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



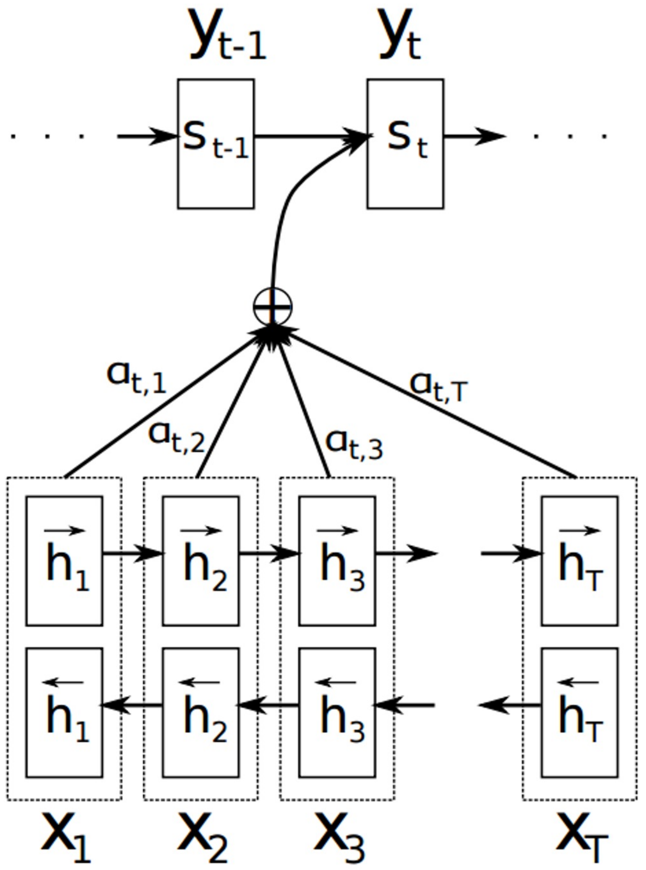
Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al, ICLR 2015

Soft attention - math 1

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

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

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



Soft attention - math 2

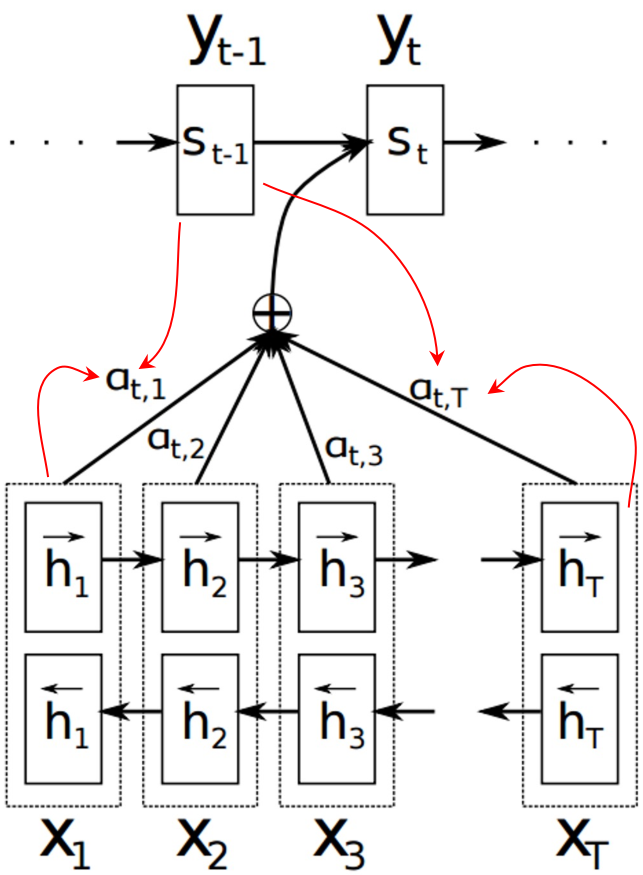
But where do the weights come from?
They are computed by another network!

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

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

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

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



Soft attention - computational aspects

The computational complexity of using soft attention is quadratic. But it's not slow:

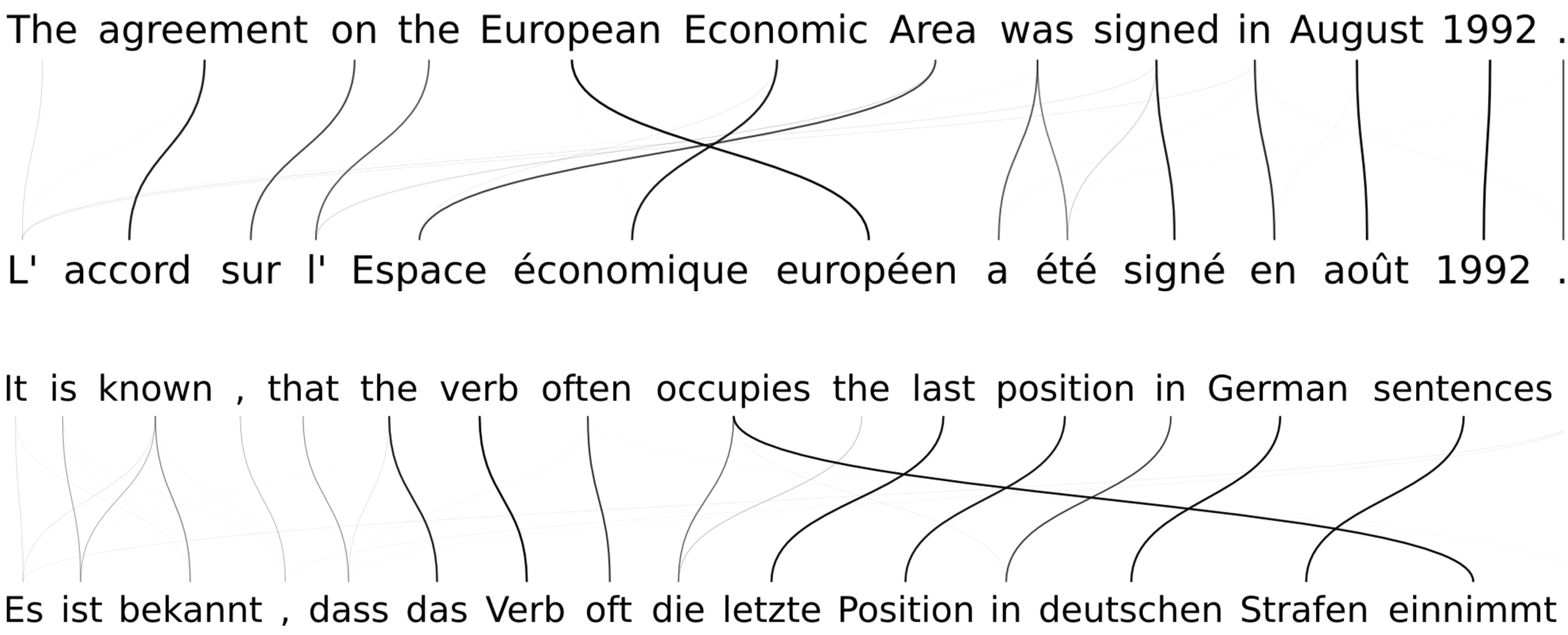
- for each pair of i and j
  - sum two vectors
  - apply tanh
  - compute dot product
- can be done in parallel for all j, i.e.
  - add a vector to a matrix
  - apply tanh
  - compute vector-matrix product
- softmax is cheap
- weighted combination is another vector-matrix product
- in summary: **just vector-matrix products = fast!**

$$e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

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

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

Soft attention - visualization



Great visualizations at https://distill.pub/2016/augmented-rnns/#attentional-interfaces

[penalty???]

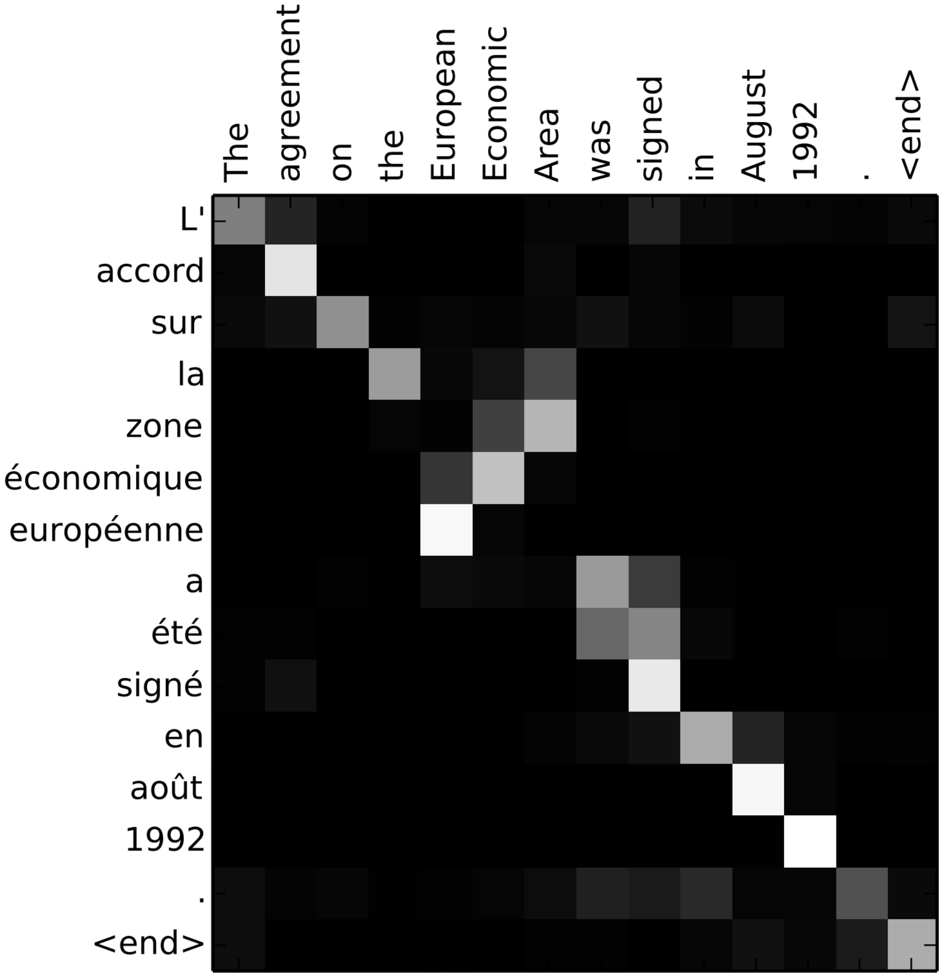
Soft attention - visualization

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 $a_{t,i}$



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

Soft attention - visualization

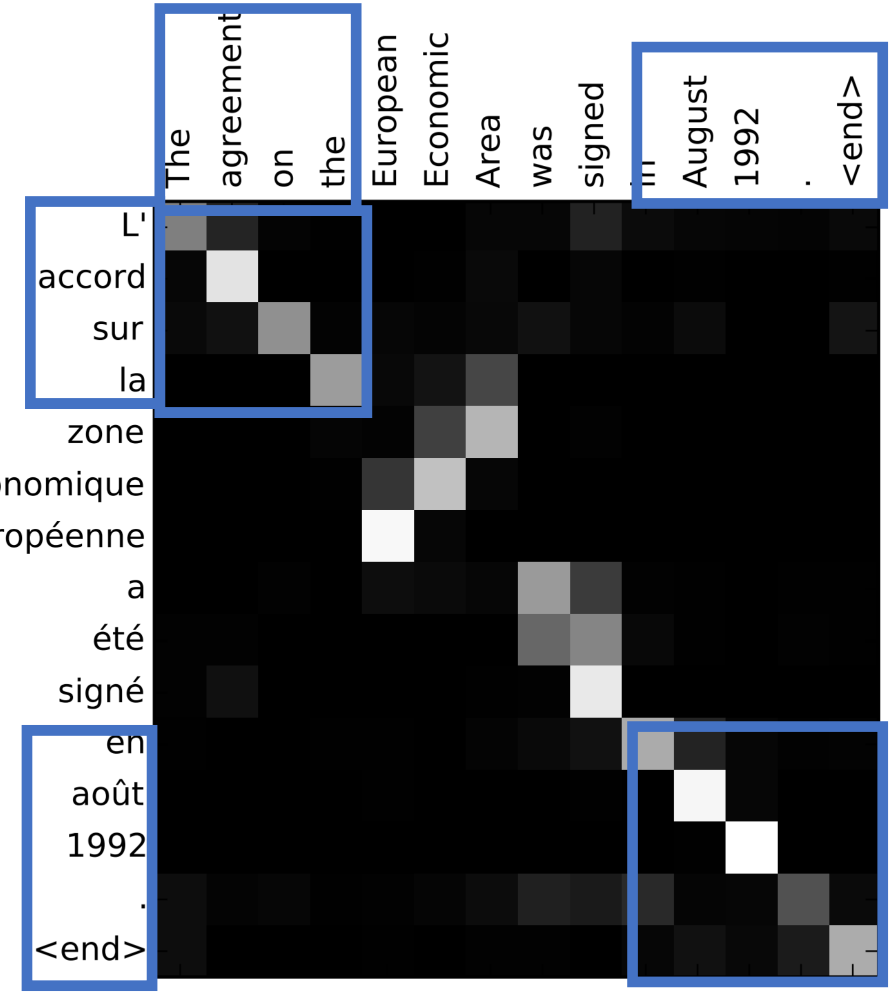
Example: English to French translation

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

Soft attention - visualization

Example: English to French translation

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

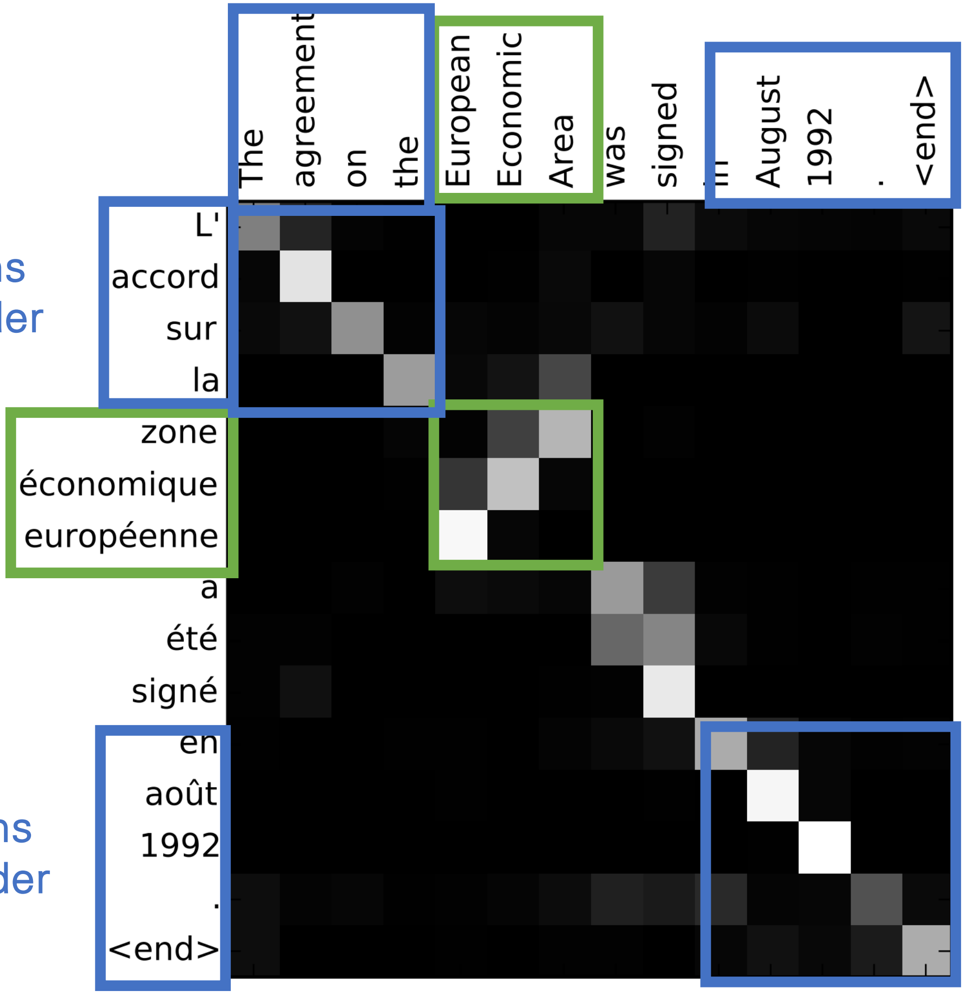
Output: "L'accord sur la zone économique européenne a été signé en août 1992."

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order



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

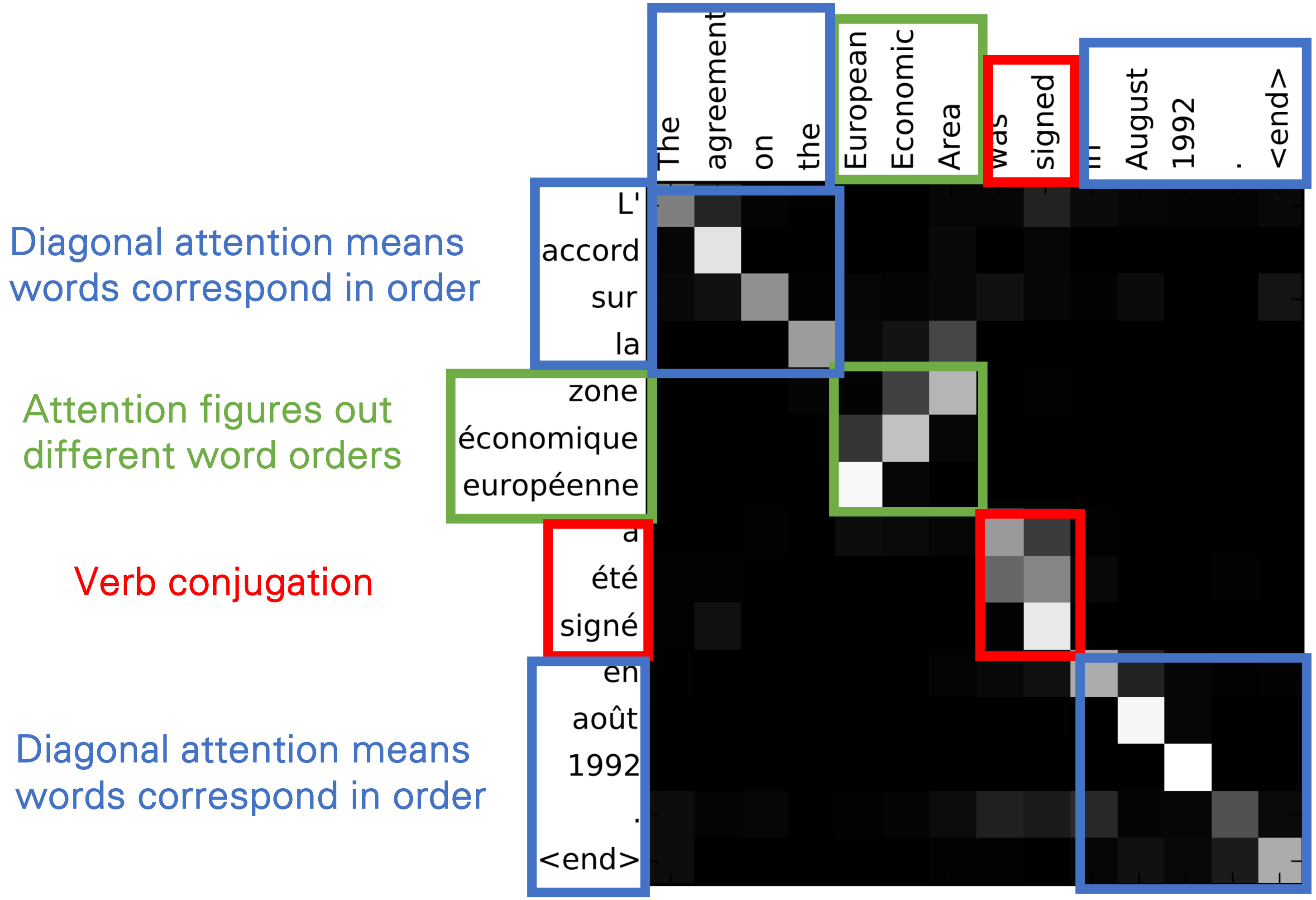
Soft attention - visualization

Example: English to French translation

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

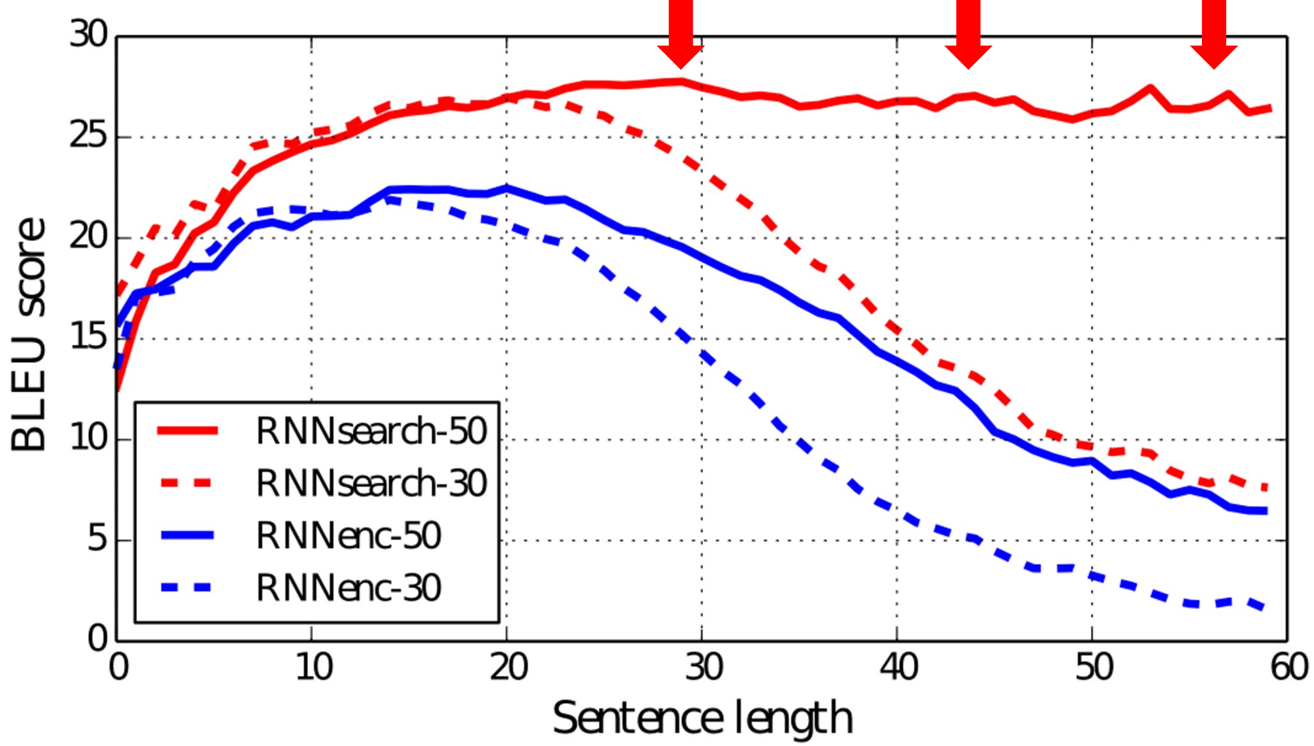
Output: "L'accord sur la zone économique européenne a été signé en août 1992."

Visualize attention weights $a_{t,i}$



Soft attention - improvements

no performance drop on long sentences



much better than RNN Encoder-Decoder

Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

without unknown words comparable with the SMT system

End-to-End Machine Translation with Recurrent Nets and Attention Mechanism

(Bahdanau et al 2014, Jean et al 2014, Gulcehre et al 2015, Jean et al 2015)

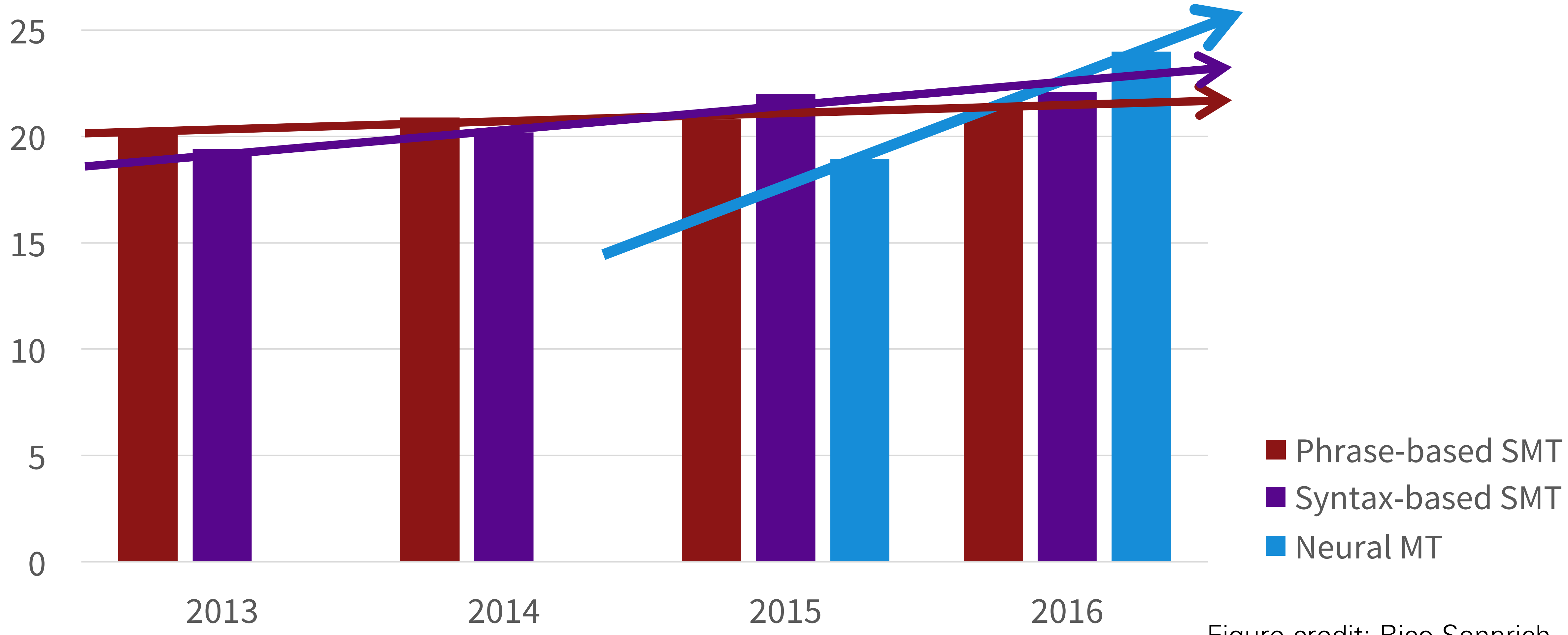


Figure credit: Rico Sennrich

Soft content-based attention pros and cons

Pros

- faster training, better performance
- good inductive bias for many tasks => lowers sample complexity

Cons

- not good enough inductive bias for tasks with monotonic alignment (handwriting recognition, speech recognition)
- chokes on sequences of length >1000

Location-based attention

- in **content-based** attention the attention weights depend on the content at different positions of the input (hence BiRNN)
- in **location-based** attention the current attention weights are computed relative to the previous attention weights

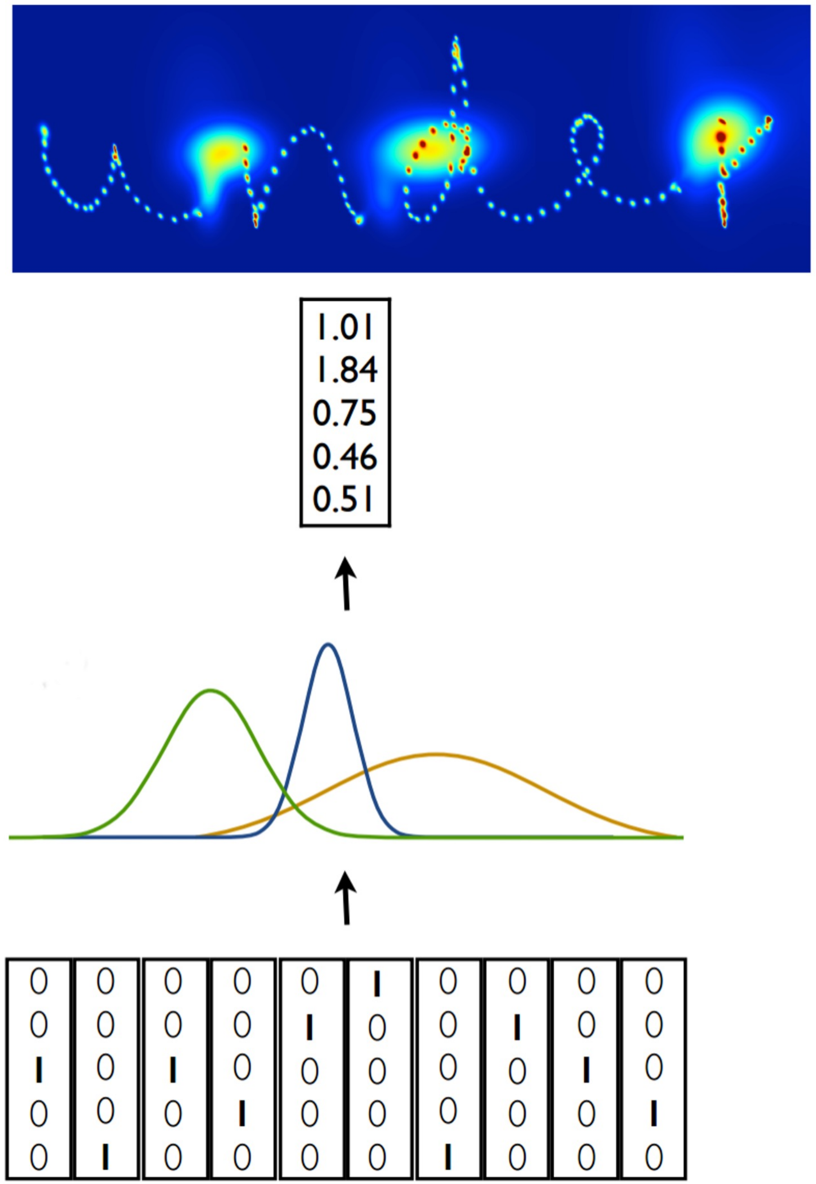
Gaussian mixture location-based attention

Originally proposed for handwriting synthesis.

The (unnormalized) weight of the input position u at the time step t is parametrized as a mixture of K Gaussians

$$\phi(t, u) = \sum_{k=1}^{K} \alpha_t^k \exp\left(-\beta_t^k \left(\kappa_t^k - u\right)^2\right)$$

$$w_t = \sum_{u=1}^{U} \phi(t, u) c_u$$



Section 5, Generating Sequence with Recurrent Neural Networks, A. Graves 2014

Gaussian mixture location-based attention

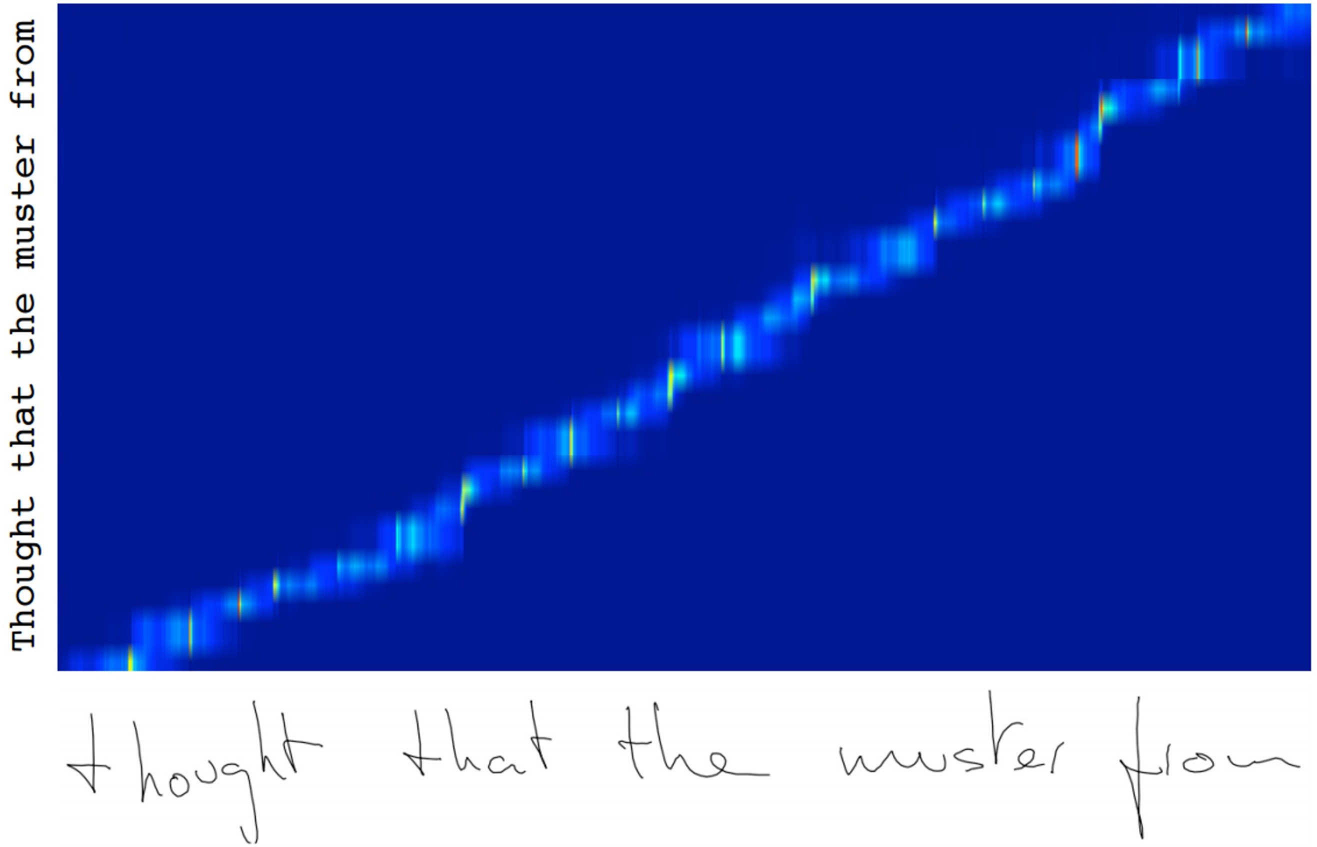
The new locations of Gaussians are computed as a sum of the previous ones and the predicted offsets

$$(\hat{\alpha}_t, \hat{\beta}_t, \hat{\kappa}_t) = W_{h^1 p} h_t^1 + b_p$$

$$\alpha_t = \exp\left(\hat{\alpha}_t\right)$$

$$\beta_t = \exp\left(\hat{\beta}_t\right)$$

$$\kappa_t = \kappa_{t-1} + \exp\left(\hat{\kappa}_t\right)$$



Gaussian mixture location-based attention

The first soft attention mechanism ever!

Pros:

- good for problems with monotonic alignment

Cons:

- predicting the offset can be challenging
- only monotonic alignment (although exp in theory could be removed)

Various Soft-Attentions

- use dot-product or non-linearity of choice instead of tanh in content-based attention
- use unidirectional RNN insteaf of Bi- (but not pure word embeddings!)
- explicitly remember past alignments with an RNN
- use a separate embedding for each of the positions of the input (heavily used in Memory Networks)
- mix content-based and location-based attentions

See “Attention-Based Models for Speech Recognition” by Chorowski et al (2015) for a scalability analysis of various attention mechanisms on speech recognition.

Various Attention Score Functions

- $\boldsymbol{q}$ is the query and $\boldsymbol{k}$ is the key
- Multi-layer Perceptron (Bahdanau et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^{\intercal} \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  - Flexible, often very good with large data
- Bilinear (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^{\intercal} W \boldsymbol{k}$$

- Dot Product (Luong et al. 2015)

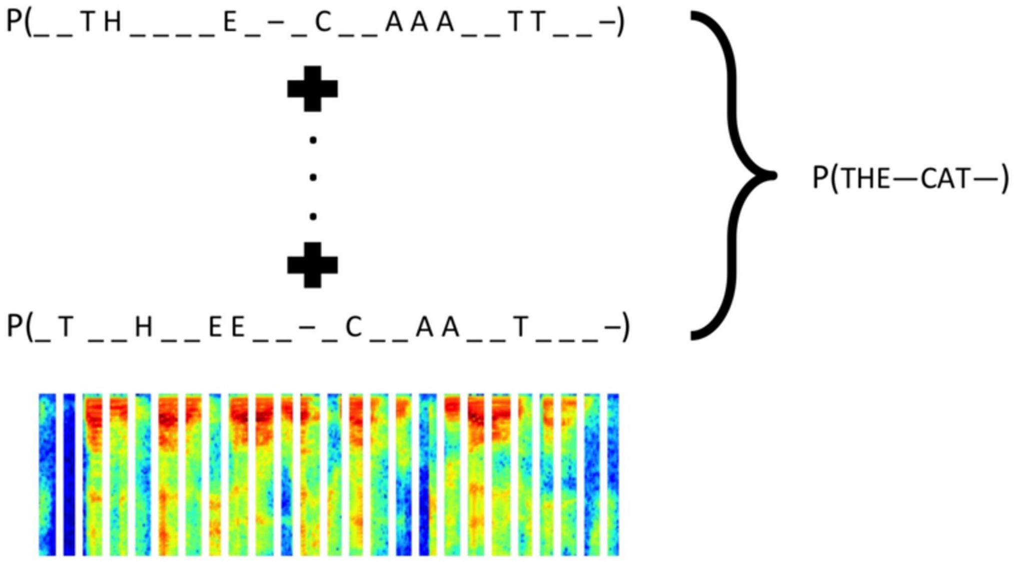
$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^{\intercal} \boldsymbol{k}$$

  - No parameters! But requires sizes to be the same.
- Scaled Dot Product (Vaswani et al. 2017)
  - Problem: scale of dot product increases as dimensions get • larger
  - Fix: scale by size of the vector

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^{\intercal} \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

Going back in time: Connection Temporal Classification (CTC)

- CTC is a predecessor of soft attention that is still widely used
- has very successful inductive bias for monotonous seq2seq transduction
- core idea: sum over all possible ways of inserting blank tokens in the output so that it aligns with the input



CTC

labeling

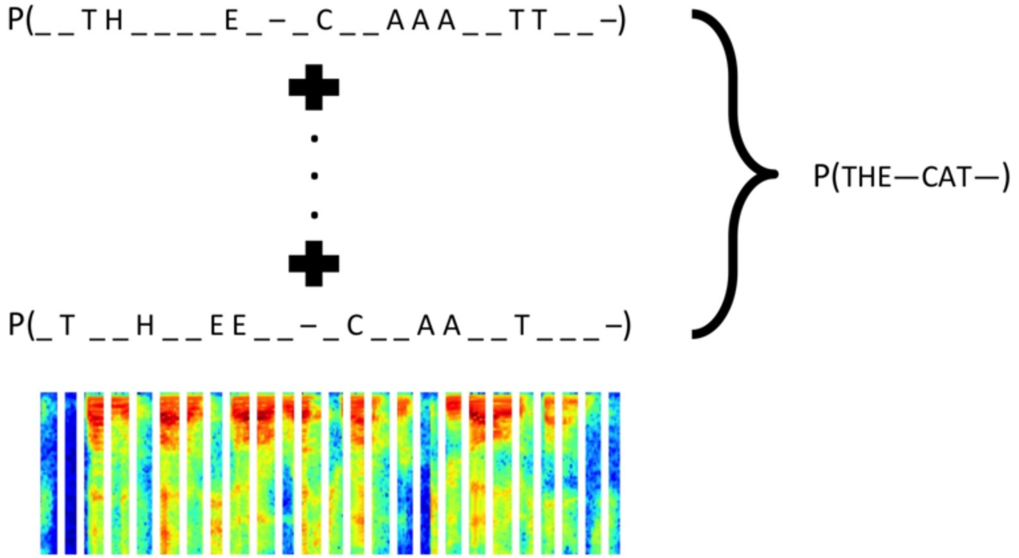
input

conditional probability of a labeling with blanks

probability of outputting \pi_t at the step t

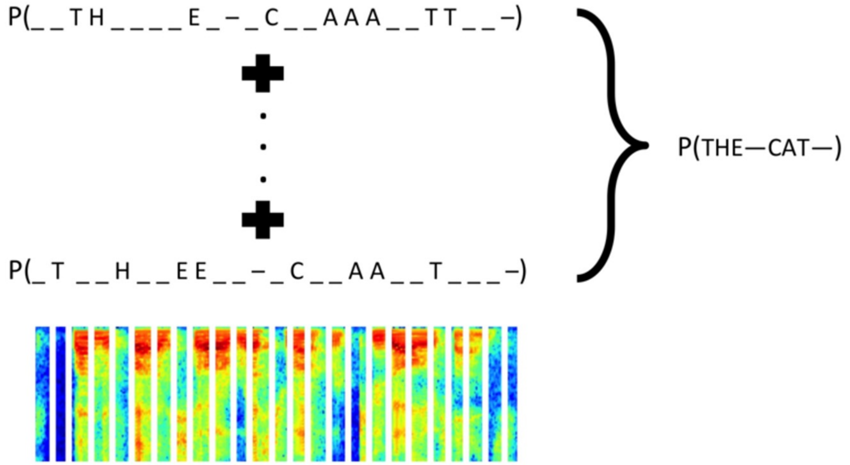
$$p(l|x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} \prod_t y^t_{\pi_t}$$

sum over all labelling with blanks



CTC

- can be viewed as modelling p(y|x) as sum of all p(y|a,x), where a is a monotonic alignment
- thanks to the monotonicity assumption the marginalization of a can be carried out with forward-backward algorithm (a.k.a. dynamic programming)
- **hard stochastic monotonic attention**
- popular in speech and handwriting recognition
- y_i are conditionally independent given a and x but this can be fixed



Soft Attention and CTC for seq2seq: summary

- the most flexible and general is content-based soft attention and it is very widely used, especially in natural language processing
- location-based soft attention is appropriate for when the input and the output can be monotonously aligned; location-based and content-based approaches can be mixed
- CTC is less generic but can be hard to beat on tasks with monotonous alignments

Visual and Hard Attention

A dog is standing on a hardwood floor.

Models of Visual Attention

- Convnets are great! But they process the whole image at a high resolution.
- *"Instead humans focus attention selectively on parts of the visual space to acquire information when and where it is needed, and combine information from different fixations over time to build up an internal representation of the scene"* (Mnih et al, 2014)
- hence the idea: build a recurrent network that focus on a patch of an input image at each step and combines information from multiple steps

Recurrent Models of Visual Attention, V. Mnih et al, NIPS 2014

Soft and Hard Attention

RAM attention mechanism is hard - it outputs a precise location where to look.

Content-based attention from neural MT is soft - it assigns weights to all input locations.

CTC can be interpreted as a hard attention mechanism with tractable gradient.

Soft and Hard Attention

Soft

- deterministic
- exact gradient
- O(input size)
- typically easy to train

Hard

- stochastic*
- gradient approximation**
- O(1)
- harder to train

* deterministic hard attention would not have gradients

** exact gradient can be computed for models with tractable marginalization (e.g. CTC)

Soft and Hard Attention

Can soft content-based attention be used for vision? Yes.

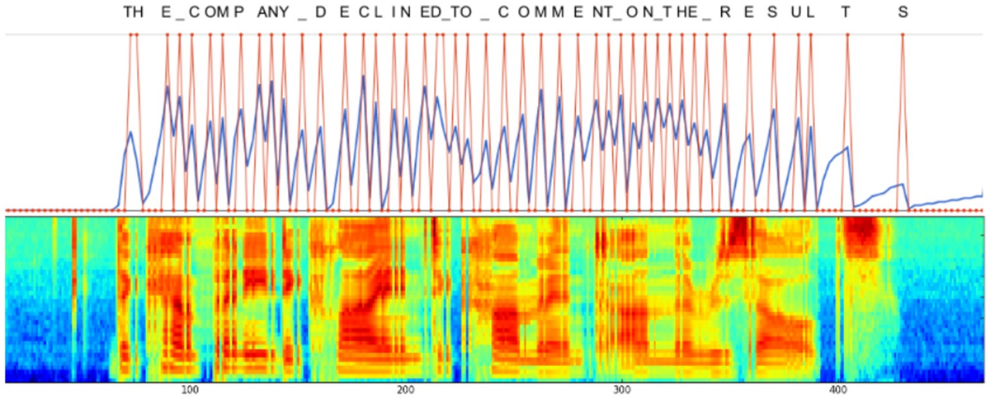
Show Attend and Tell, Xu et al, ICML 2015

A dog is standing on a hardwood floor.

Can hard attention be used for seq2seq? Yes.

Learning Online Alignments with
Continuous Rewards Policy Gradient,
Luo et al, NIPS 2016

(but the learning curves are a nightmare…)

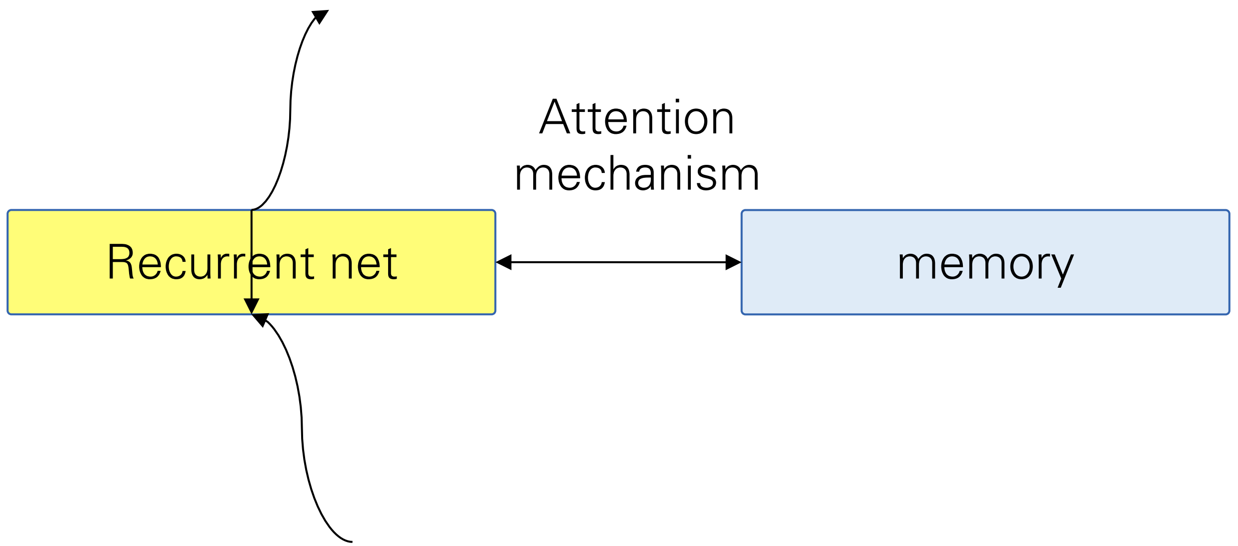


Why attention?

- Long term memories - attending to memories
  - Dealing with gradient vanishing problem
- Exceeding limitations of a global representation
  - Attending/focusing to smaller parts of data
    - patches in images
    - words or phrases in sentences
- Decoupling representation from a problem
  - Different problems required different sizes of representations
    - LSTM with longer sentences requires larger vectors
- Overcoming computational limits for visual data
  - Focusing only on the parts of images
  - Scalability independent of the size of images
- Adds some interpretability to the models (error inspection)

Attention on Memory Elements

- **Recurrent networks cannot remember things for very long**
  - The cortex only remember things for 20 seconds
- **We need a "hippocampus" (a separate memory module)**
  - LSTM [Hochreiter 1997], registers
  - **Memory networks** [Weston et 2014] (FAIR), associative memory
  - NTM [Graves et al. 2014], "tape".



Recall: Long-Term Dependencies

- The RNN gradient is a product of Jacobian matrices, each associated with a step in the forward computation. To store information robustly in a finite-dimensional state, the dynamics must be contractive [Bengio et al 1994].

$$L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))$$

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \cdots \frac{\partial s_{t+1}}{\partial s_t}$$

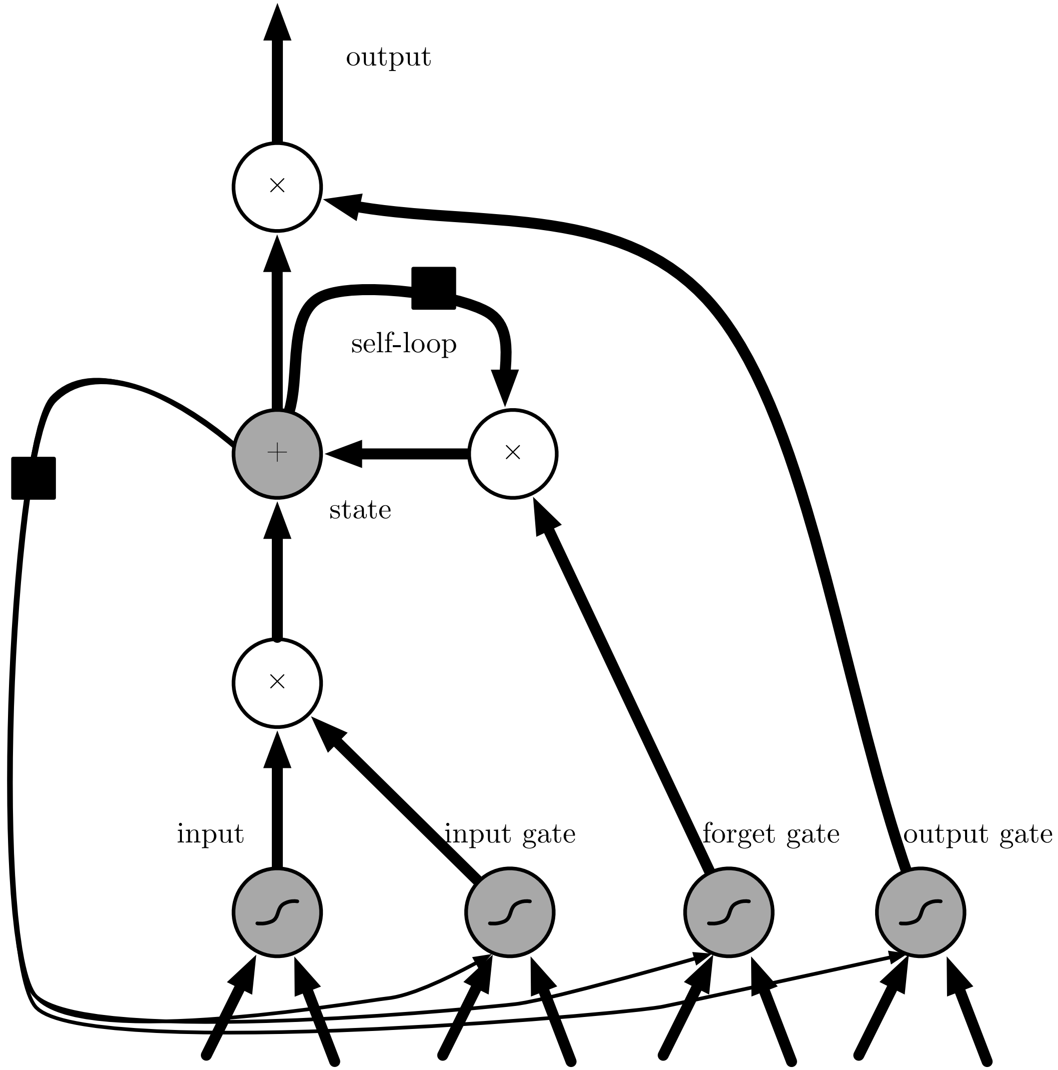
Storing bits robustly requires sing. values<1

- Problems:
  - sing. values of Jacobians > 1 → gradients explode
  - or sing. values < 1 → gradients shrink & vanish (Hochreiter 1991)
  - or random → variance grows exponentially

Gradient clipping

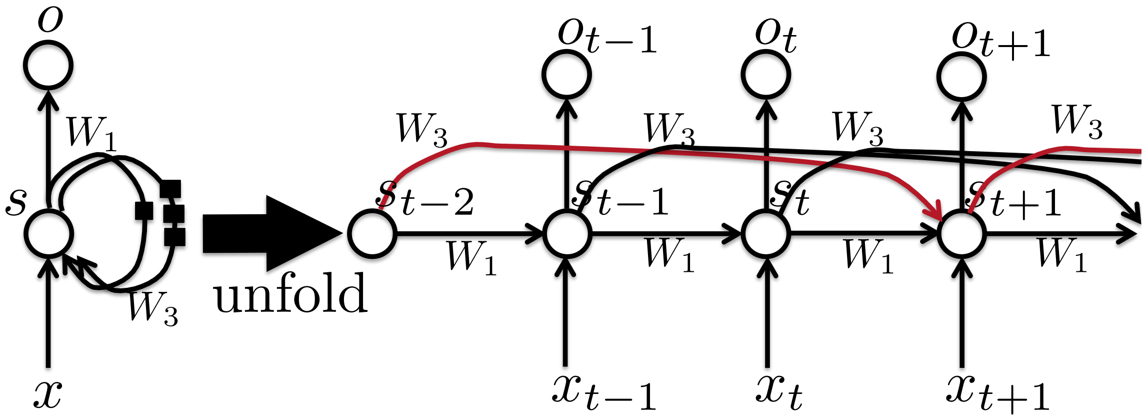
Gated Recurrent Units & LSTM

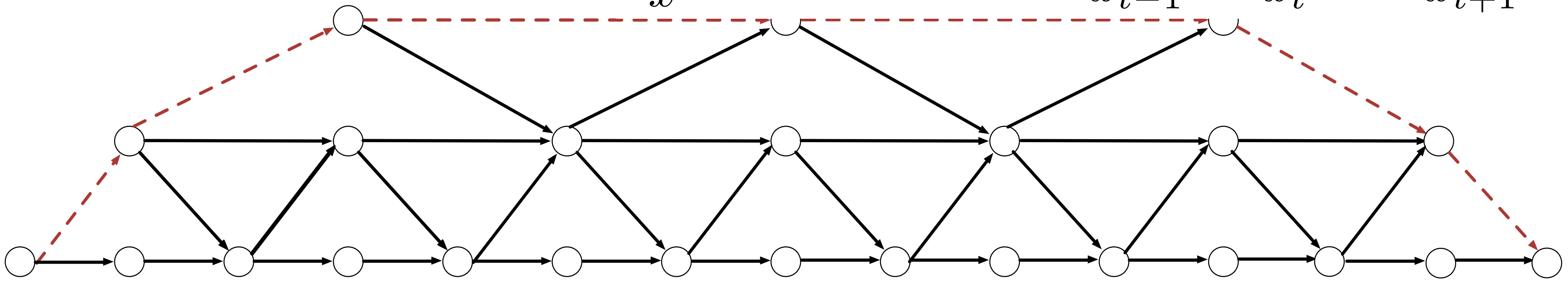
- **Create a path where gradients can flow for longer with self-loop**
- Corresponds to an eigenvalue of Jacobian slightly less than 1
- LSTM is **heavily used** (Hochreiter & Schmidhuber 1997)
- GRU light-weight version (Cho et al 2014)



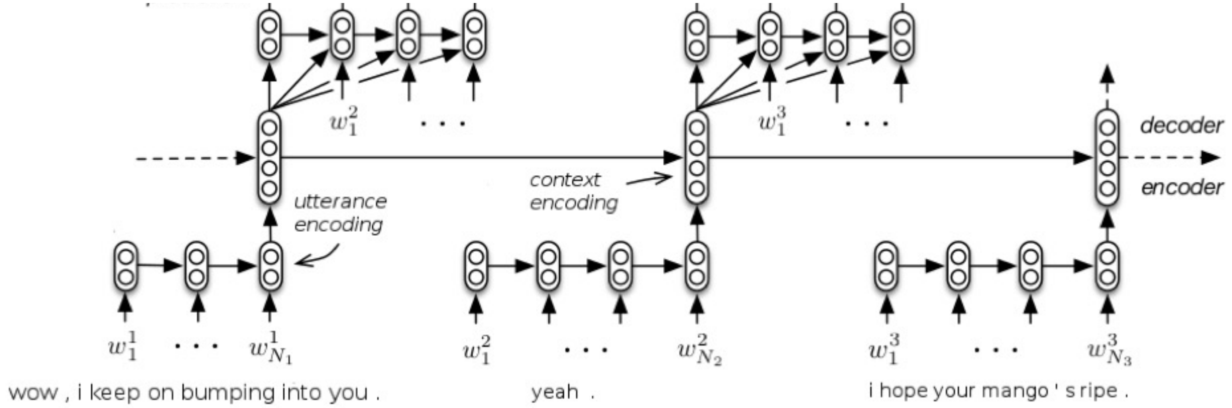
Delays & Hierarchies to Reach Farther

- Delays and multiple time scales, Elhihi & Bengio NIPS 1995, Koutnik et al ICML 2014



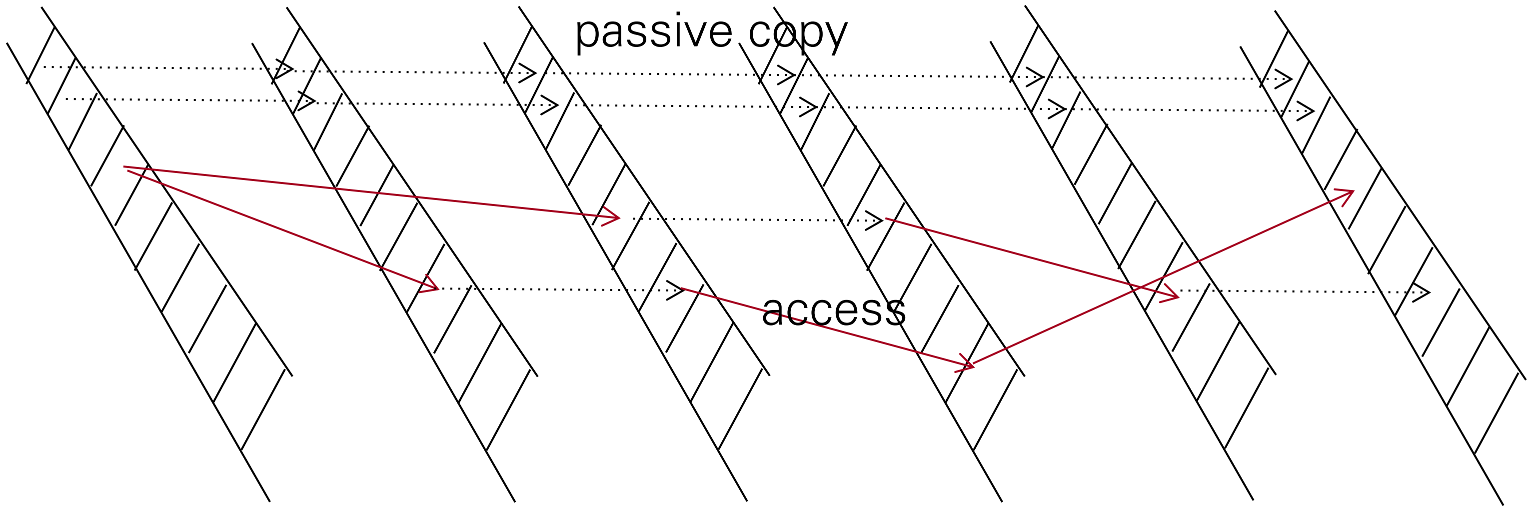


Hierarchical RNNs (words / sentences): Sordoni et al CIKM 2015, Serban et al AAAI 2016



Large Memory Networks: Sparse Access Memory for Long-Term Dependencies

- A mental state stored in an external memory can stay for arbitrarily long durations, until evoked for read or write
- Forgetting = vanishing gradient.
- Memory = larger state, avoiding the need for forgetting/vanishing



Memory Networks

- Class of models that combine large memory with learning component that can read and write to it.
- Incorporates **reasoning** with **attention** over **memory** (RAM).
- Most ML has limited memory which is more-or-less all that's needed for "low level" tasks e.g. object detection.

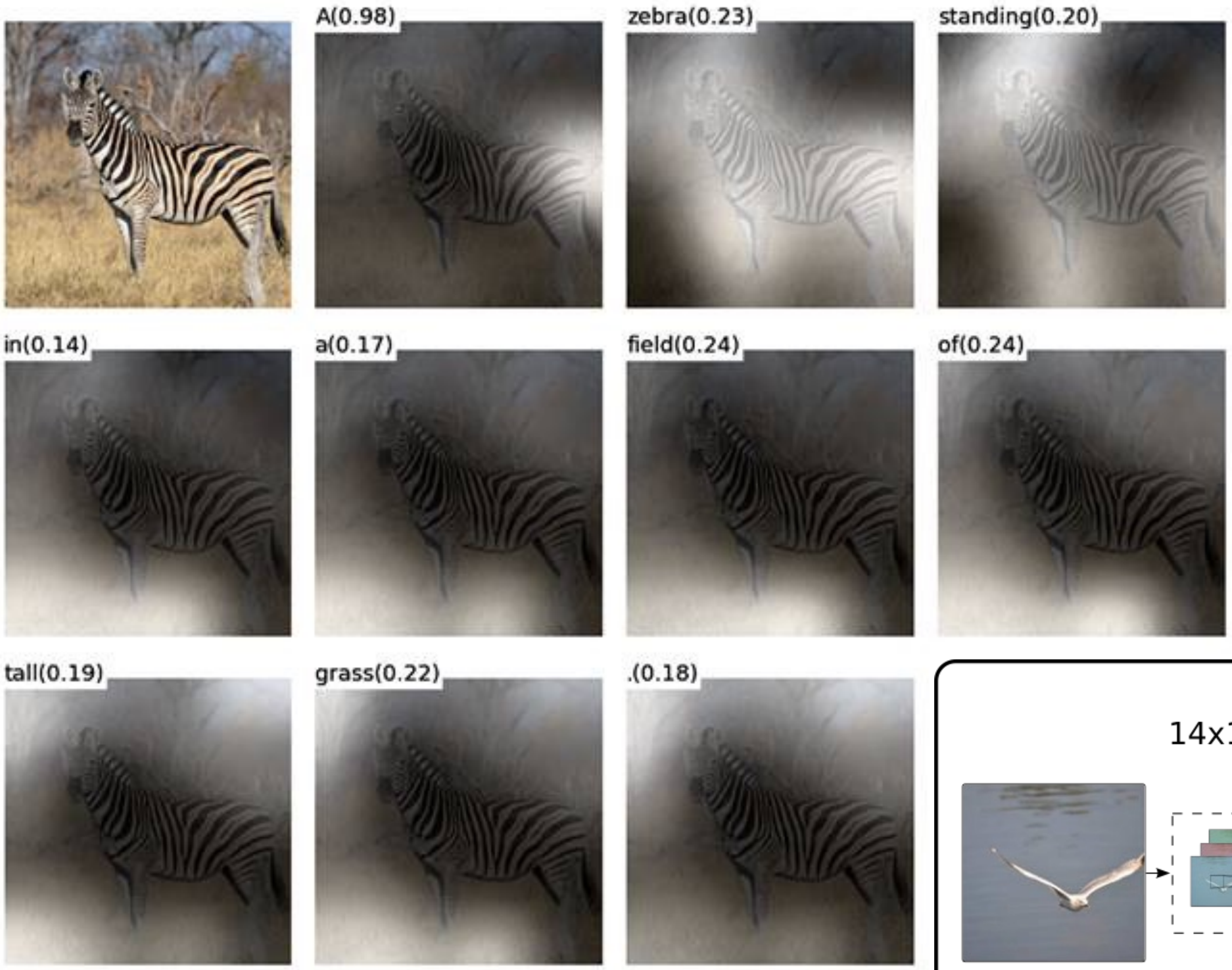
Jason Weston, Sumit Chopra, Antoine Bordes. **Memory Networks.** ICLR 2016

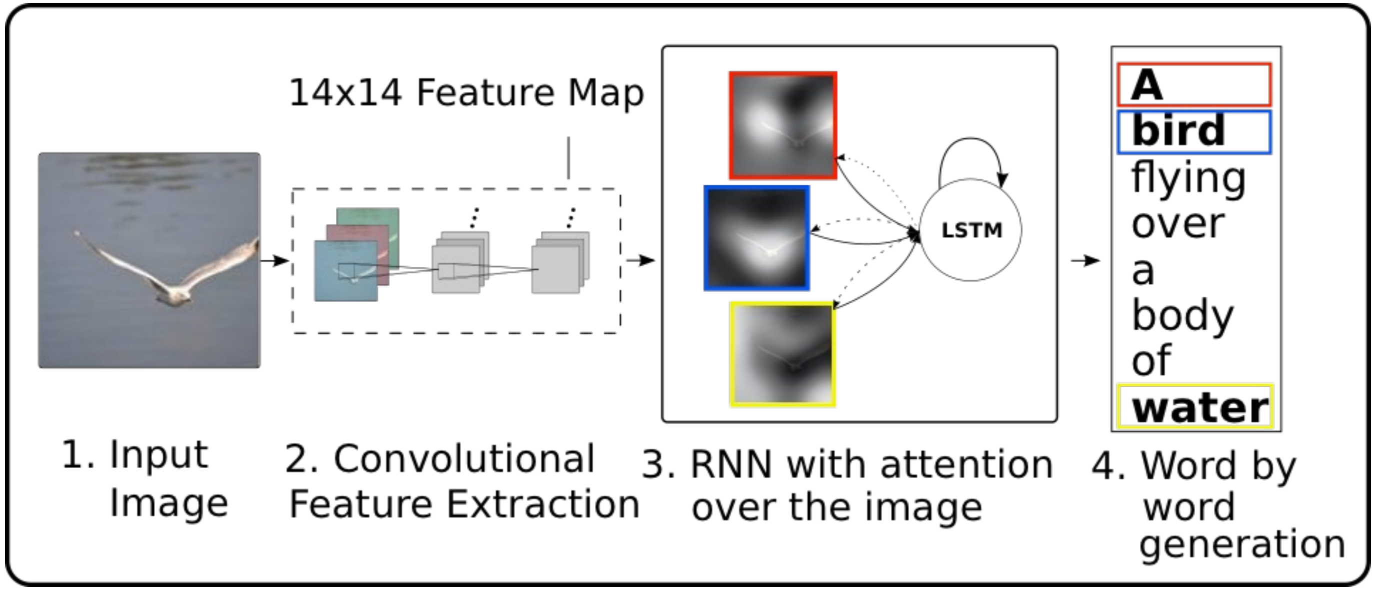
S. Sukhbaatar, A. Szlam, J. Weston, R. Fergus. **End-to-end Memory Networks.** NIPS 2015

Ankit Kumar et al. **Ask Me Anything: Dynamic Memory Networks for Natural Language Processing.** ICML 2016

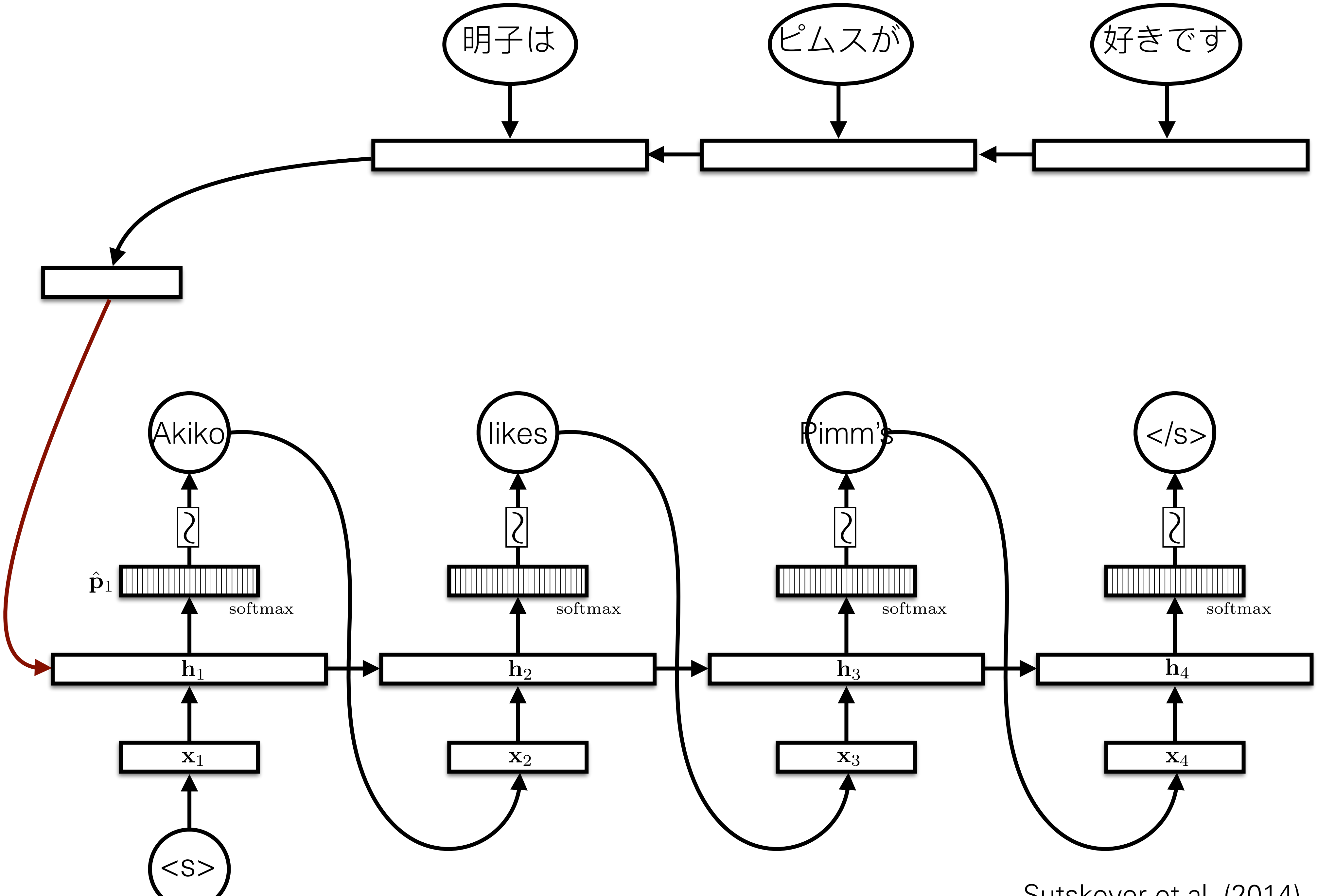
Alex Graves et al. **Hybrid computing using a neural network with dynamic external memory.** Nature, 538(7626): 471–476, 2016.

Paying Attention to Selected Parts of the Image While Uttering Words

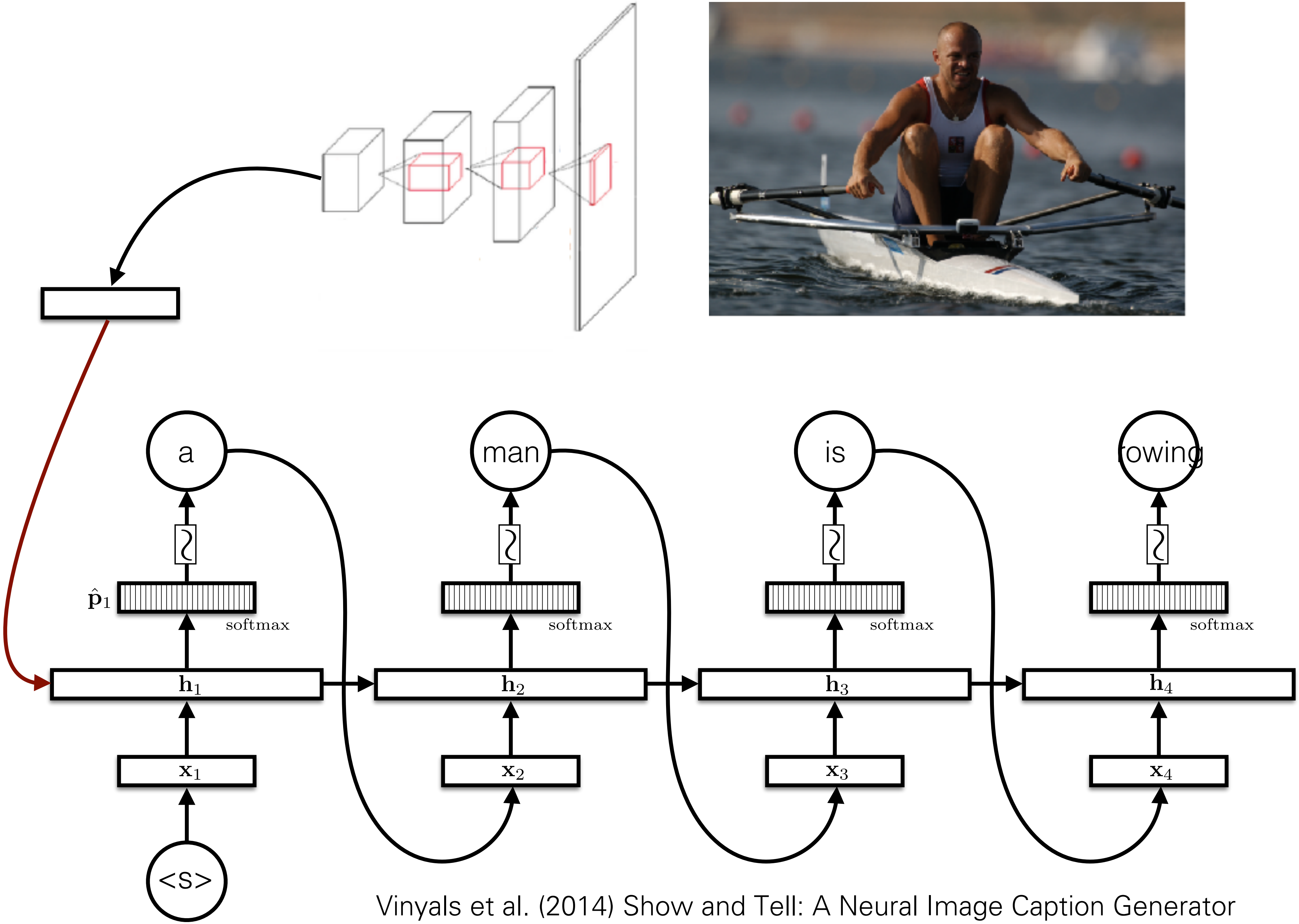




Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio. ICML 2015

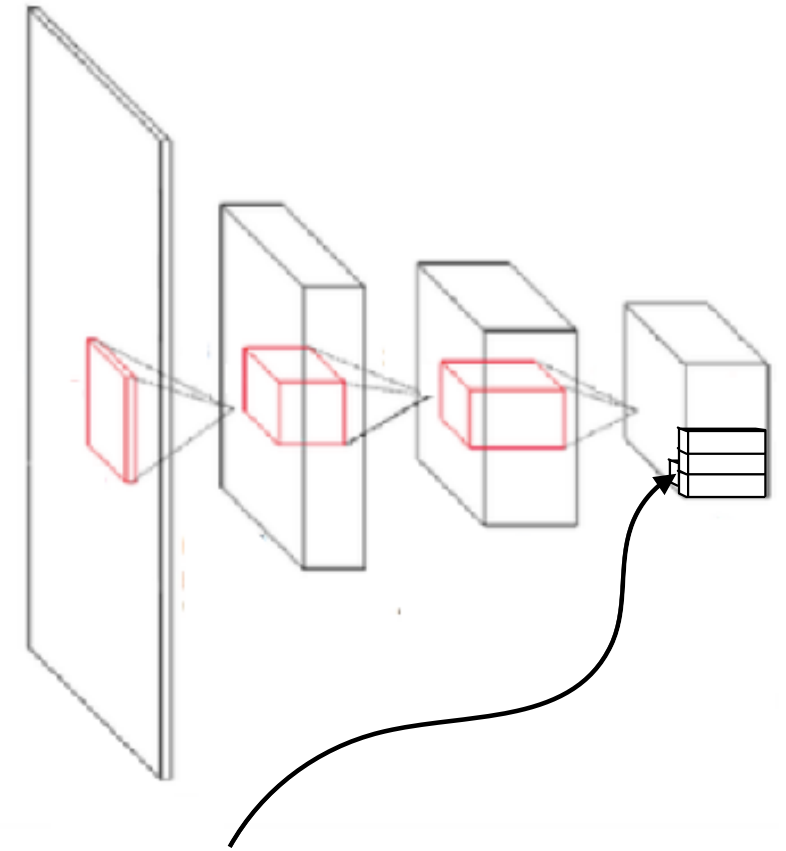


Sutskever et al. (2014)



Vinyals et al. (2014) Show and Tell: A Neural Image Caption Generator

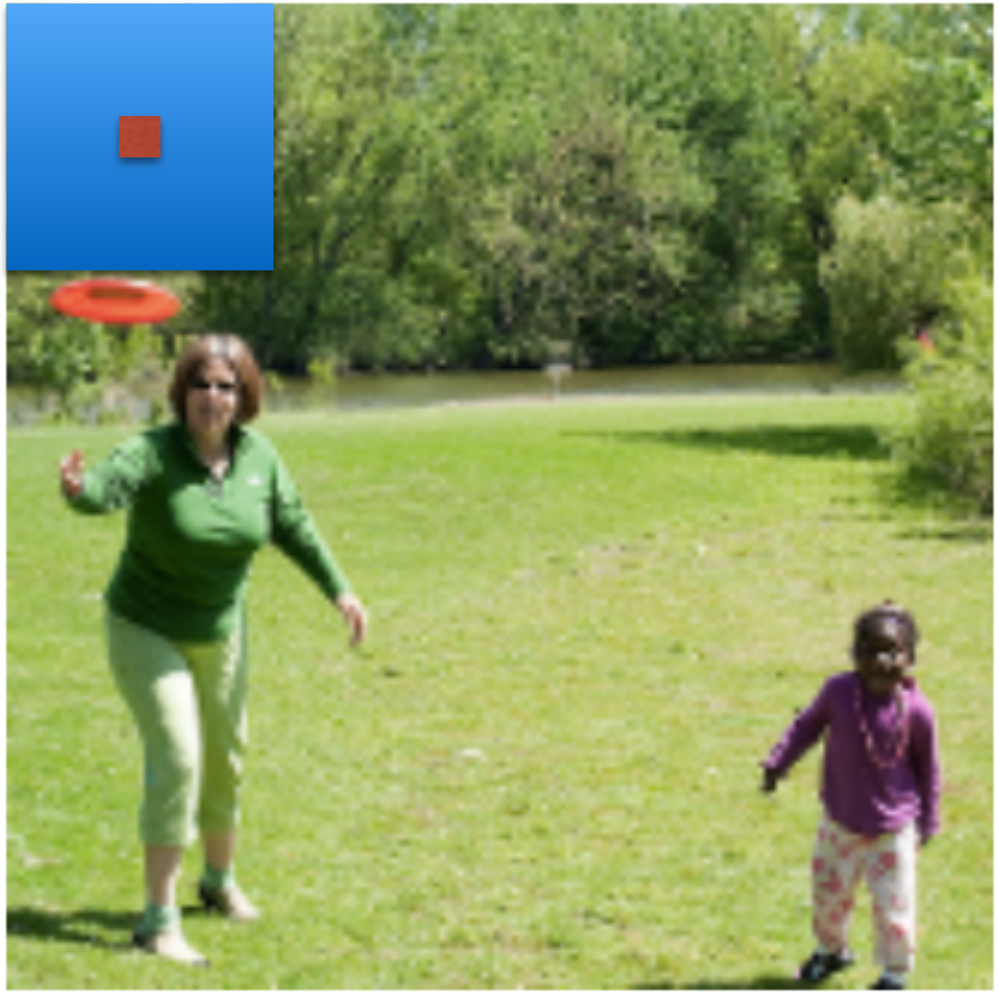
Regions in ConvNets



- Each point in a “higher” level of a convnet defines spatially localized feature vectors(/matrices).
- Xu et al. calls these “annotation vectors”, $\mathbf{a}_i,\ i \in \{1, \dots, L\}$

Regions in ConvNets

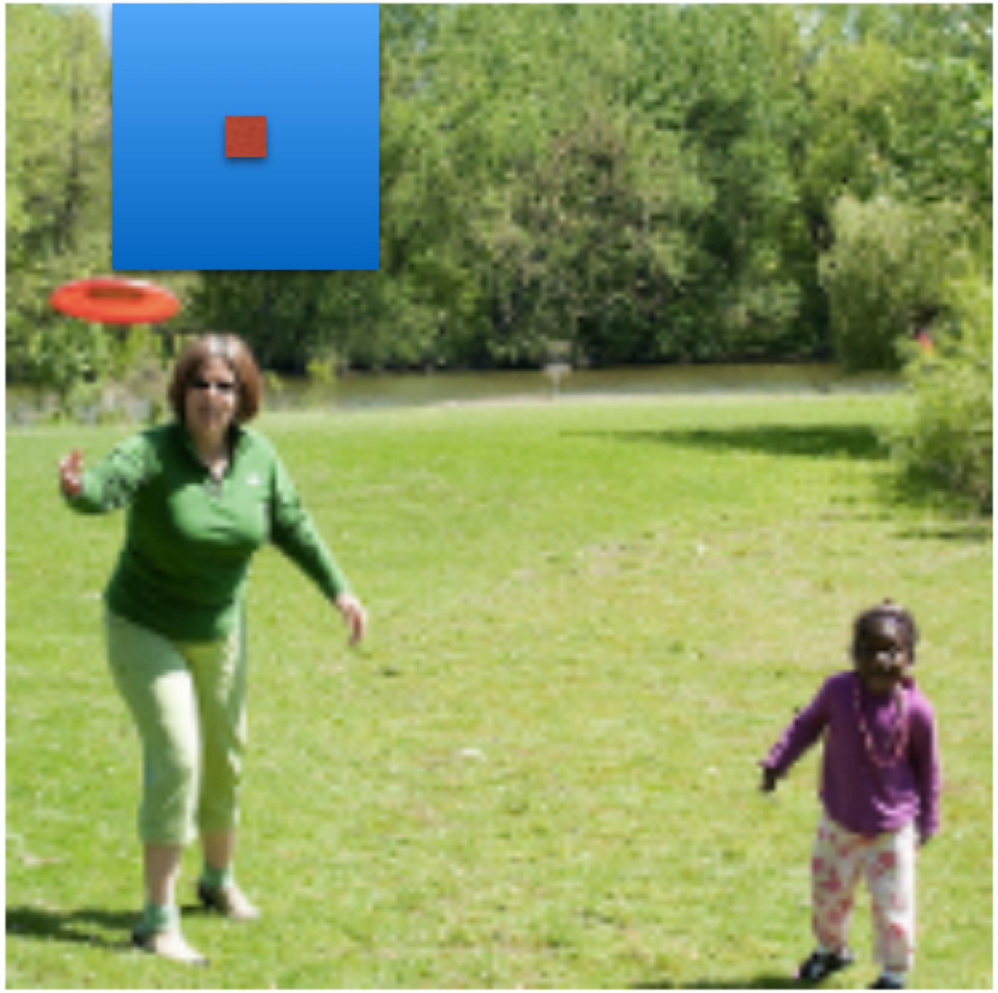
$\mathbf{a}_1$



$$\mathbf{F} = \begin{bmatrix} | & & \\ \mathbf{a}_1 & & \\ | & & \end{bmatrix}$$

Regions in ConvNets

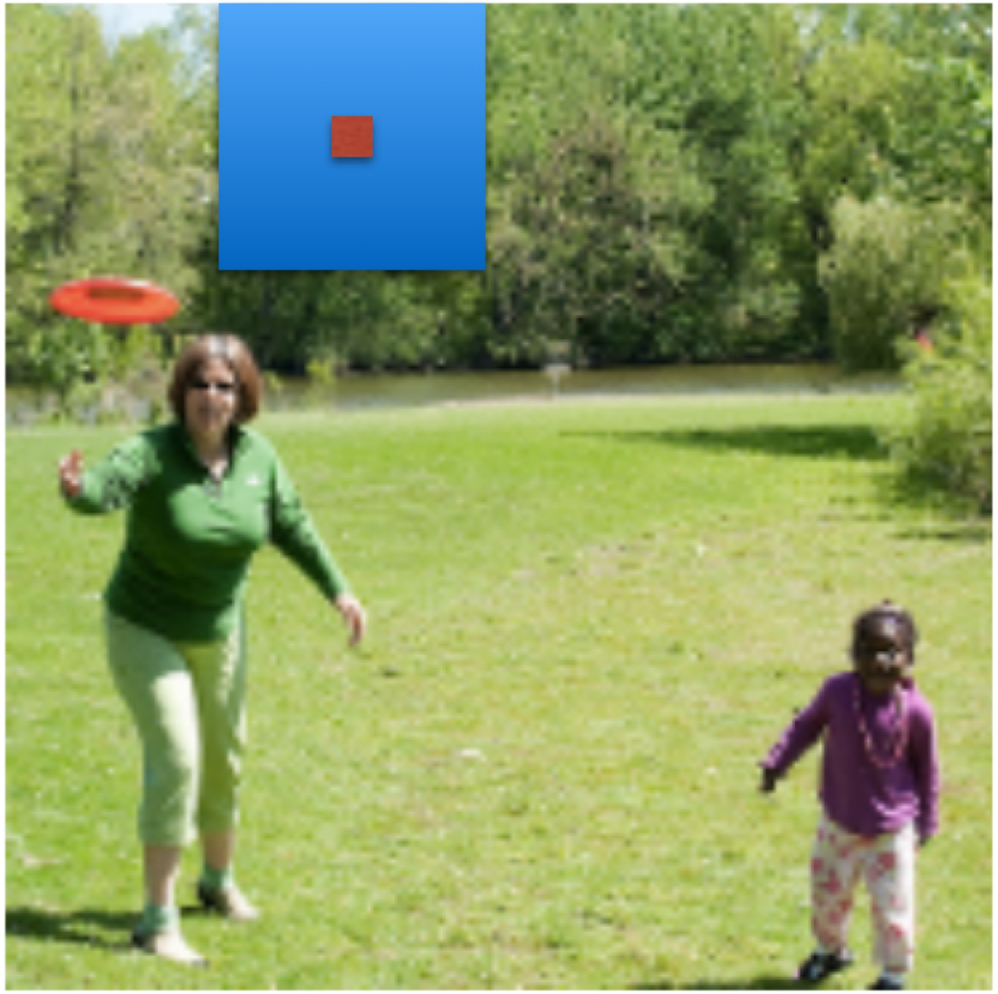
$\mathbf{a}_2$



$$\mathbf{F} = \begin{bmatrix} | & | & & & \\ \mathbf{a}_1 & \mathbf{a}_2 & & & \\ | & | & & & \end{bmatrix}$$

Regions in ConvNets

$\mathbf{a}_3$



$$\mathbf{F} = \begin{bmatrix} | & | & | & \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \cdots \\ | & | & | & \end{bmatrix}$$

Extension of LSTM via the context vector

- Extract L D-dimensional annotations
  - Lower convolutional layer to have the correspondence between the feature vectors and portions of the 2-D image

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{D+m+n,n} \begin{pmatrix} \mathbf{E}\mathbf{y}_{t-1} \\ \mathbf{h}_{t-1} \\ \boxed{\hat{\mathbf{z}}_t} \end{pmatrix} \quad (1)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \quad (2)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (3)$$

$\mathbf{E}$: embedding matrix
$\mathbf{y}$: captions
$\mathbf{h}$: previous hidden state
$\mathbf{z}$: context vector, a dynamic representation of the relevant part of the image input at time t

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}.$$

A MLP conditioned on the previous hidden state

$$\hat{\mathbf{z}}_t = \phi\left(\{\mathbf{a}_i\}, \{\alpha_i\}\right)$$

ϕ is the 'attention' ('focus') function – 'soft' / 'hard'

$$\boxed{p(\mathbf{y}_t | \mathbf{a}, \mathbf{y}_1^{t-1}) \propto \exp(\mathbf{L}_o(\mathbf{E}\mathbf{y}_{t-1} + \mathbf{L}_h \mathbf{h}_t + \mathbf{L}_z \hat{\mathbf{z}}_t))}$$

Hard attention

We have two sequences
'I' that runs over localizations
't' that runs over words

Stochastic decisions are discrete here, so derivatives are zero

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}$$

$$\hat{\mathbf{z}}_t = \phi\left(\{\mathbf{a}_i\}, \{\alpha_i\}\right)$$

Loss is a variational lower bound on the marginal log-likelihood

$$\begin{aligned} L_s &= \sum_s p(s \mid \mathbf{a}) \log p(\mathbf{y} \mid s, \mathbf{a}) \\ &\leq \log \sum_s p(s \mid \mathbf{a}) p(\mathbf{y} \mid s, \mathbf{a}) \\ &= \log p(\mathbf{y} \mid \mathbf{a}) \end{aligned}$$

Due to Jensen's inequality $E[\log(X)] \leq \log(E[X])$

$$p(s_{t,i} = 1 \mid s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i} \mathbf{a}_i.$$

$$\frac{\partial L_s}{\partial W} = \sum_s p(s \mid \mathbf{a}) \left[\frac{\partial \log p(\mathbf{y} \mid s, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid s, \mathbf{a}) \frac{\partial \log p(s \mid \mathbf{a})}{\partial W} \right]$$

$$\tilde{s}_t \sim \text{Multinoulli}_L(\{\alpha_i\})$$

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a}) \frac{\partial \log p(\tilde{s}^n \mid \mathbf{a})}{\partial W} \right]$$

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a})}{\partial W} + \lambda_r (\log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a}) - b) \frac{\partial \log p(\tilde{s}^n \mid \mathbf{a})}{\partial W} + \lambda_e \frac{\partial H[\tilde{s}^n]}{\partial W} \right]$$

To reduce the estimator variance, entropy term H[s] and bias are added [1,2]

[1] J. Ba et al. "Multiple object recognition with visual attention"
[2] A. Mnih et al. "Neural variational inference and learning in belief networks"

Hard attention

We have two sequences
'I' that runs over localizations
't' that runs over words

Stochastic decisions are discrete here, so derivatives are zero

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}$$

$$\hat{\mathbf{z}}_t = \phi\left(\{\mathbf{a}_i\}, \{\alpha_i\}\right)$$

Loss is a variational lower bound on the marginal log-likelihood

$$p(s_{t,i} = 1 \mid s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i} \mathbf{a}_i.$$

$$\begin{aligned} L_s &= \sum_s p(s \mid \mathbf{a}) \log p(\mathbf{y} \mid s, \mathbf{a}) \\ &\leq \log \sum_s p(s \mid \mathbf{a}) p(\mathbf{y} \mid s, \mathbf{a}) \\ &= \log p(\mathbf{y} \mid \mathbf{a}) \end{aligned}$$

$$\frac{\partial L_s}{\partial W} = \sum_s p(s \mid \mathbf{a}) \left[\frac{\partial \log p(\mathbf{y} \mid s, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid s, \mathbf{a}) \frac{\partial \log p(s \mid \mathbf{a})}{\partial W} \right]$$

Due to Jensen's inequality $E[\log(X)] \leq \log(E[X])$

- Instead of a soft interpolation, make a **zero-one decision** about where to attend
- Harder to train, requires methods such as reinforcement learning

$$\tilde{s}_t \sim \text{Multinoulli}_L(\{\alpha_i\})$$

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a}) \frac{\partial \log p(\tilde{s}^n \mid \mathbf{a})}{\partial W} \right]$$

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a})}{\partial W} + \lambda_r (\log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a}) - b) \frac{\partial \log p(\tilde{s}^n \mid \mathbf{a})}{\partial W} + \lambda_e \frac{\partial H[\tilde{s}^n]}{\partial W} \right]$$

To reduce the estimator variance, entropy term H[s] and bias are added [1,2]

[1] J. Ba et al. "Multiple object recognition with visual attention"
[2] A. Mnih et al. "Neural variational inference and learning in belief networks"

Soft attention

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i}\mathbf{a}_i$$

Instead of making hard decisions,
we take the expected context vector

$$\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t] = \sum_{i=1}^{L} \alpha_{t,i}\mathbf{a}_i$$

The whole model is smooth and differentiable under the deterministic attention; learning via a standard backprop

$$\phi\left(\{\mathbf{a}_i\},\{\alpha_i\}\right) = \sum_i^L \alpha_i \mathbf{a}_i$$

Theoretical arguments

- $\mathbb{E}_{p(s_t|a)}[\mathbf{h}_t]$ equals to computing $\mathbf{h}_t$ using a single forward prop with the expected context vector $\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t]$
- Normalized Weighted Geometric Mean approximation [1] $NWGM[p(y_t = k \mid \mathbf{a})] \approx \mathbb{E}[p(y_t = k \mid \mathbf{a})]$
- Finally

$$NWGM[p(y_t = k \mid \mathbf{a})] = \frac{\prod_i \exp(n_{t,k,i})^{p(s_{t,i}=1|a)}}{\sum_j \prod_i \exp(n_{t,j,i})^{p(s_{t,i}=1|a)}} = \frac{\exp(\mathbb{E}_{p(s_t|a)}[n_{t,k}])}{\sum_j \exp(\mathbb{E}_{p(s_t|a)}[n_{t,j}])}$$

$$\mathbb{E}[\mathbf{n}_t] = \mathbf{L}_o(\mathbf{E}\mathbf{y}_{t-1} + \mathbf{L}_h\mathbb{E}[\mathbf{h}_t] + \mathbf{L}_z\mathbb{E}[\hat{\mathbf{z}}_t])$$

[1] P. Baldi et al. "The dropout learning algorithm"

Soft attention

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i}\mathbf{a}_i$$

Instead of making hard decisions,
we take the expected context vector

$$\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t] = \sum_{i=1}^{L} \alpha_{t,i}\mathbf{a}_i$$

The whole model is smooth and differentiable under the deterministic attention; learning via a standard backprop

$$\phi\left(\{\mathbf{a}_i\}, \{\alpha_i\}\right) = \sum_i^L \alpha_i \mathbf{a}_i$$

Theoretical arguments

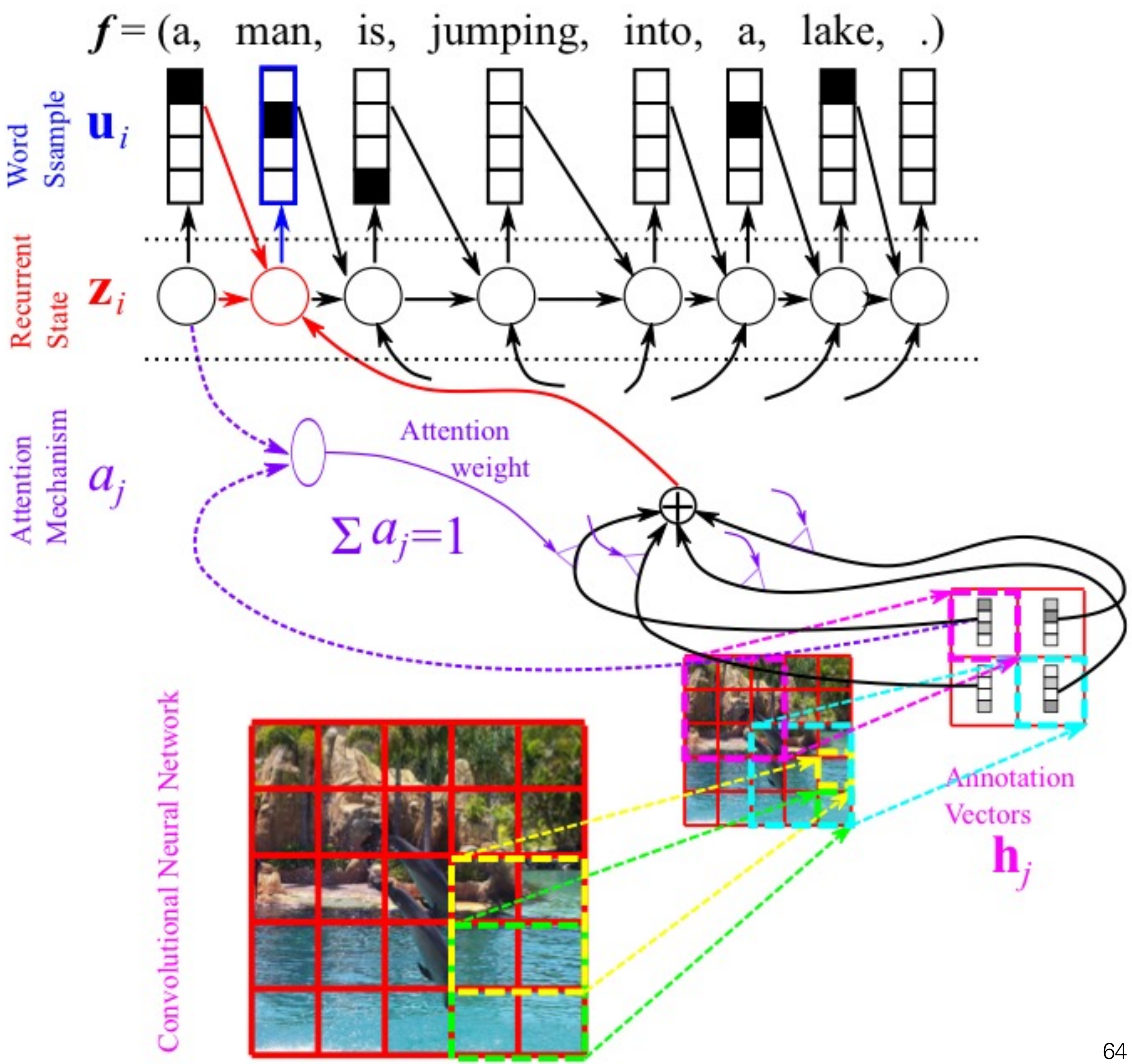
- $\mathbb{E}_{p(s_t|a)}[\mathbf{h}_t]$ equals to computing $\mathbf{h}_t$ using a single forward prop with the expected context vector $\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t]$
- Normalized Weighted Geometric Mean approximation [1] $NWGM[p(y_t = k \mid \mathbf{a})] \approx \mathbb{E}[p(y_t = k \mid \mathbf{a})]$
- Finally

$$NWGM[p(y_t = k \mid \mathbf{a})] = \frac{\prod_i \exp(n_{t,k,i})^{p(s_{t,i}=1|a)}}{\sum_j \prod_i \exp(n_{t,j,i})^{p(s_{t,i}=1|a)}} = \frac{\exp(\mathbb{E}_{p(s_t|a)}[n_{t,k}])}{\sum_j \exp(\mathbb{E}_{p(s_t|a)}[n_{t,j}])}$$

$$\mathbb{E}[\mathbf{n}_t] = \mathbf{L}_o(\mathbf{E}\mathbf{y}_{t-1} + \mathbf{L}_h\mathbb{E}[\mathbf{h}_t] + \mathbf{L}_z\mathbb{E}[\hat{\mathbf{z}}_t])$$

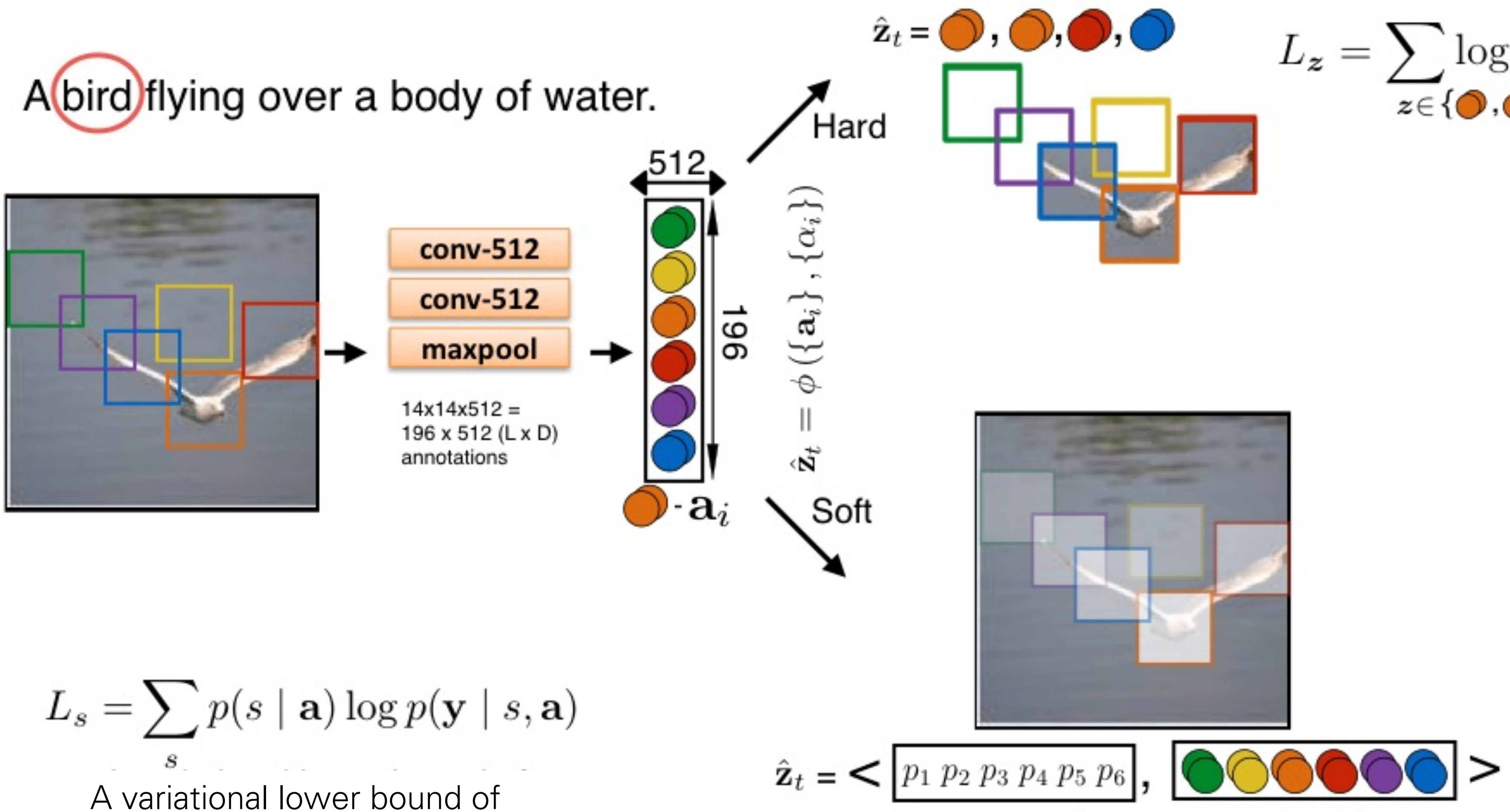
[1] P. Baldi et al. "The dropout learning algorithm"

How soft/hard attention works



How soft/hard attention works

Sample regions of attention

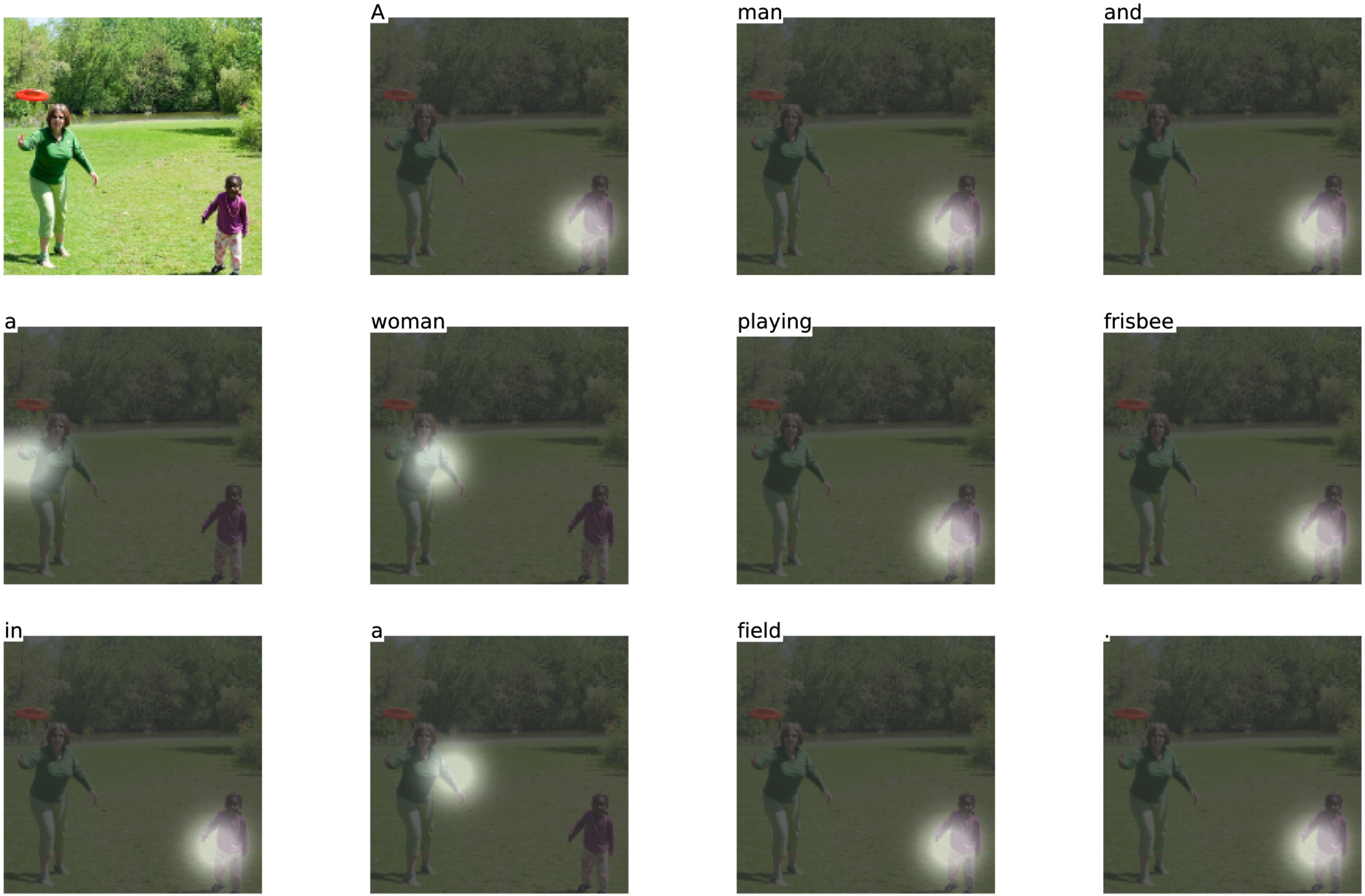


$$L_z = \sum_{z \in \{\ldots\}} \log p(\boldsymbol{y} \mid \boldsymbol{z})$$

$$L_s = \sum_s p(s \mid \mathbf{a}) \log p(\mathbf{y} \mid s, \mathbf{a})$$

A variational lower bound of
maximum likelihood

Computes the expexted attention



Hard Attention

Soft Attention

The Good

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

And the Bad

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and
a hat on a skateboard.

A person is standing on a beach
with a surfboard.

A woman is sitting at a table
with a large pizza.

A man is talking on his cell phone
while another man watches.

Quantitative results

Model	Human M1	Human M2	Automatic BLEU	Automatic CIDEr
Human	0.638	0.675	0.471	0.91
Google*	0.273	0.317	0.587	0.946
MSR$^\bullet$	0.268	0.322	0.567	0.925
Attention-based*	0.262	0.272	0.523	0.878
Captivator$^\circ$	0.250	0.301	0.601	0.937
Berkeley LRCN$^\diamond$	0.246	0.268	0.534	0.891

M1: human preferred (or equal) the method over human annotation
M2: turing test

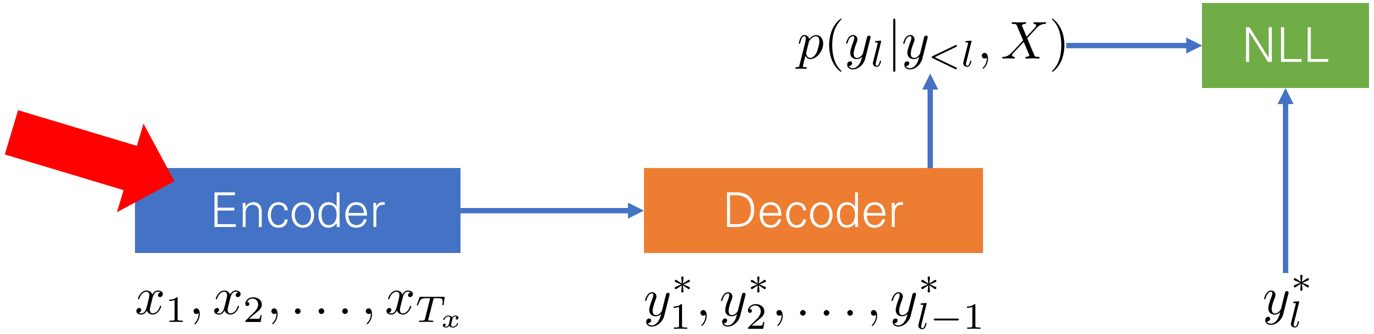
- Add soft attention to image captioning: **+2 BLEU**
- Add hard attention to image captioning: **+4 BLEU**

Parametrization – Recurrent Neural Nets

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

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

- 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

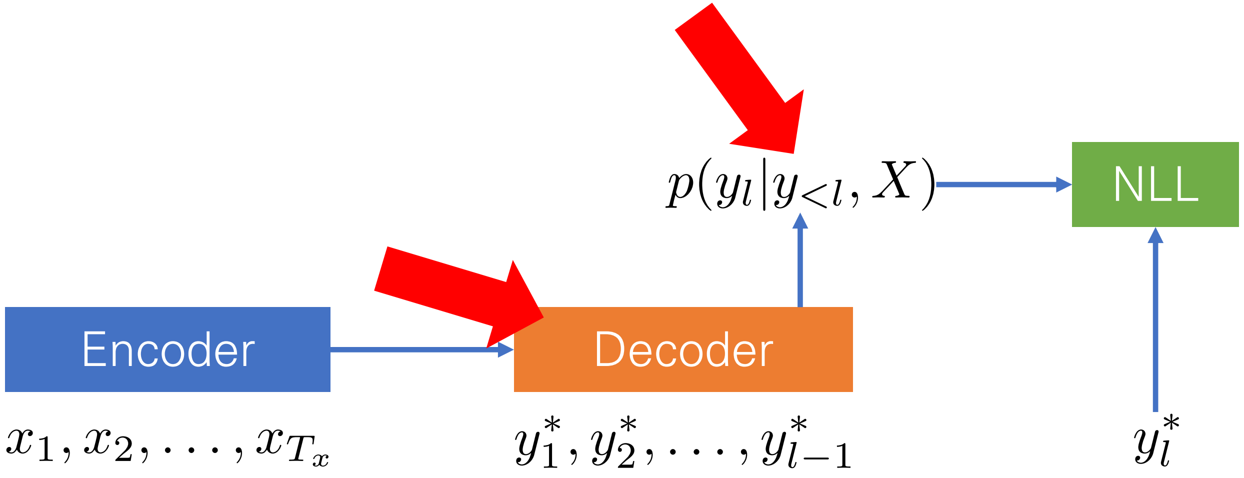
$$\alpha_{t'} \propto \exp(\mathrm{ATT}(h_{t'}, z_{t-1}, y_{t-1}))$$

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

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

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

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

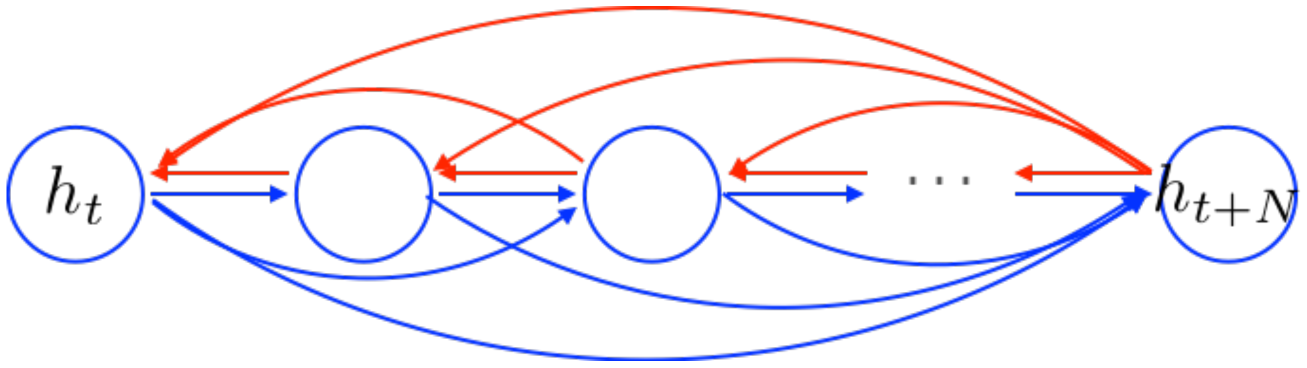


Side-note: gated recurrent units to attention

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

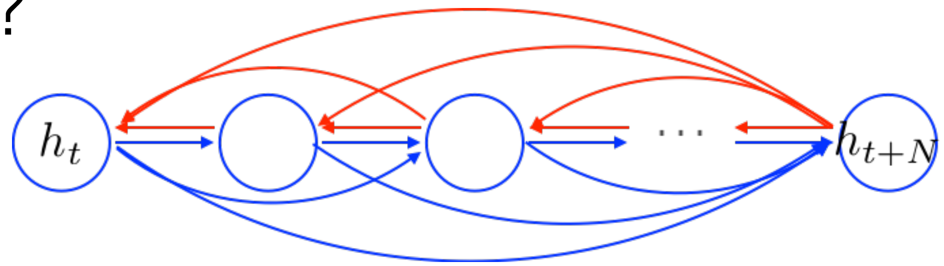
$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tilde{h}_t, \text{ where } \tilde{h}_t = f(x_t, h_{t-1})$$

- 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?

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

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

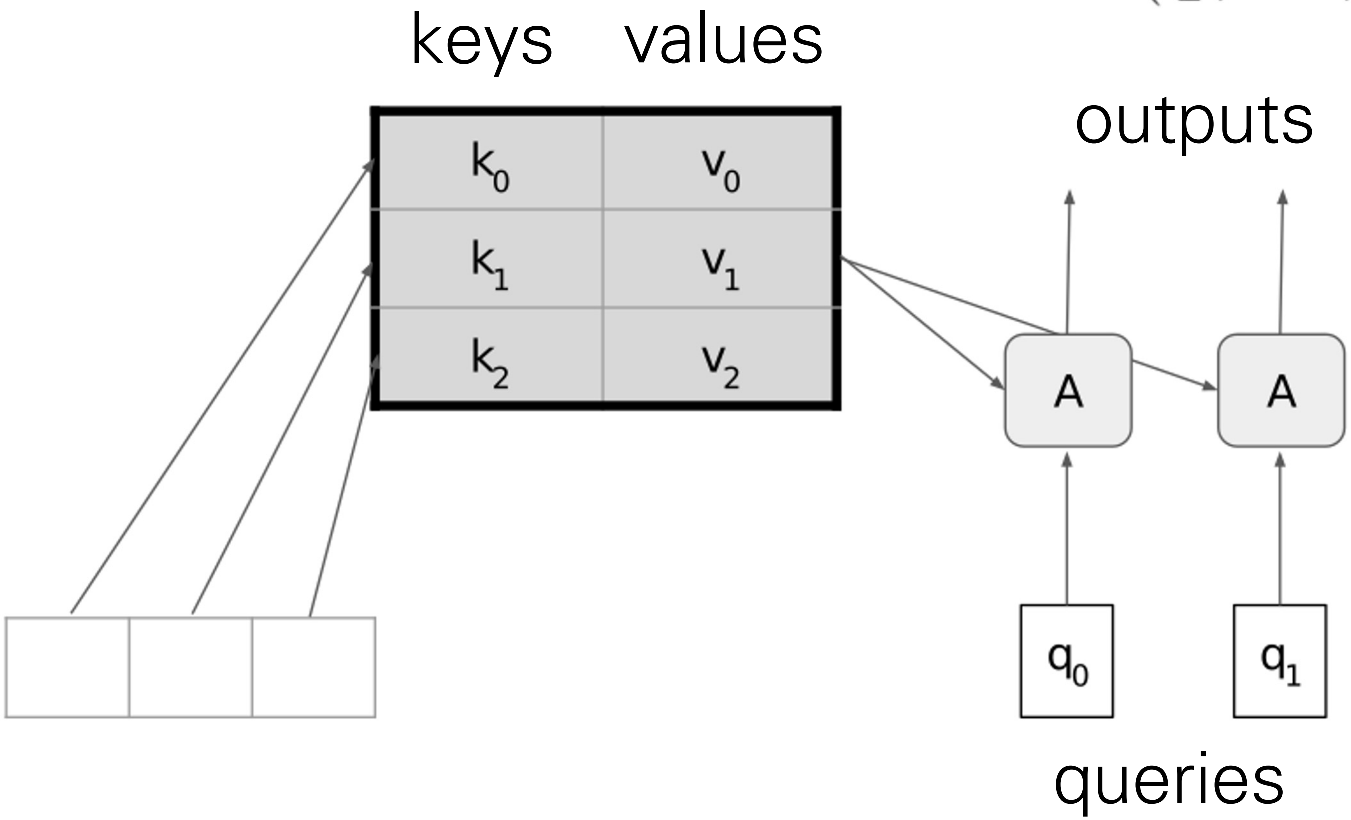
$$h_t = \sum_{i=1}^{t} \alpha_i f(x_i), \text{ where } \alpha_i \propto \exp(\mathrm{ATT}(f(x_i), x_t)) \quad \mathbf{2}$$

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

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

Generalized dot-product attention - vector form

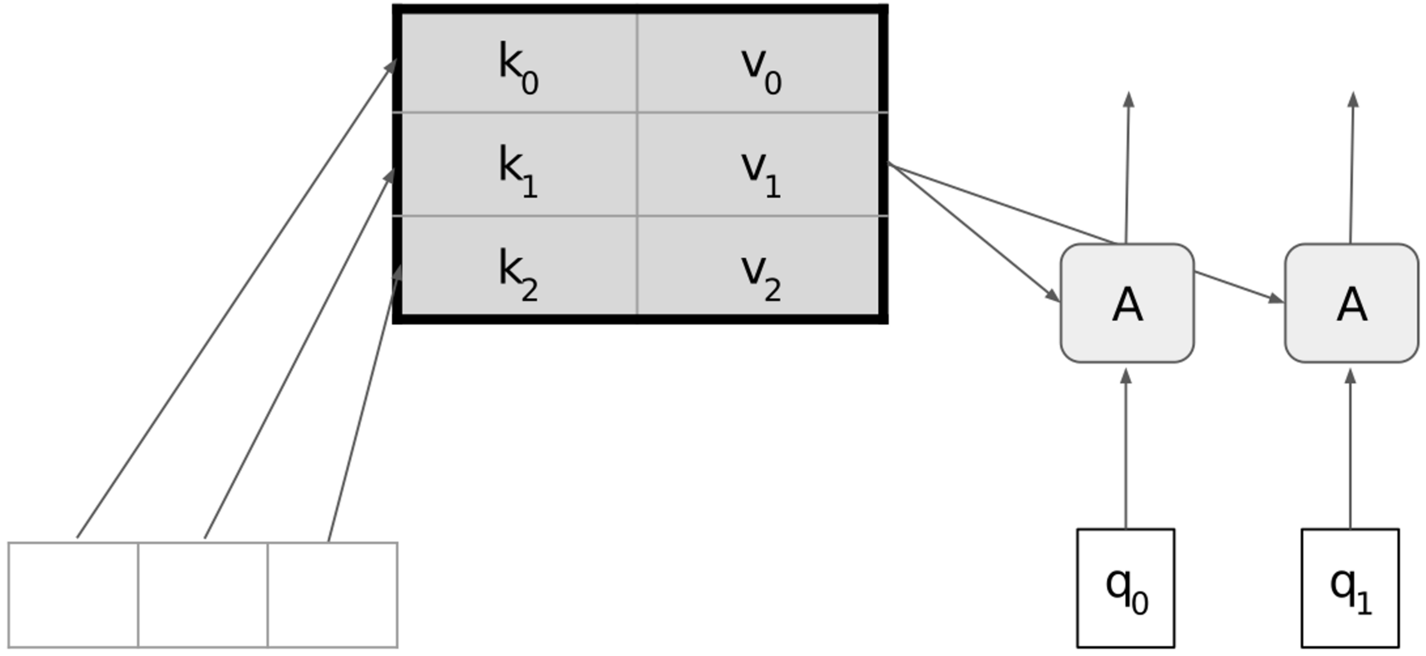
$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$



Generalized dot-product attention - matrix form

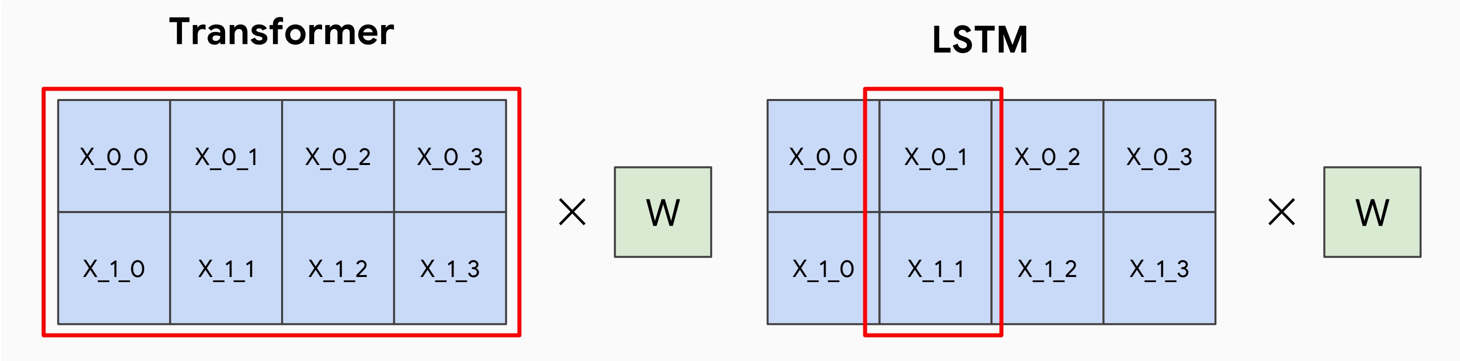
$$A(Q, K, V) = softmax(QK^T)V$$

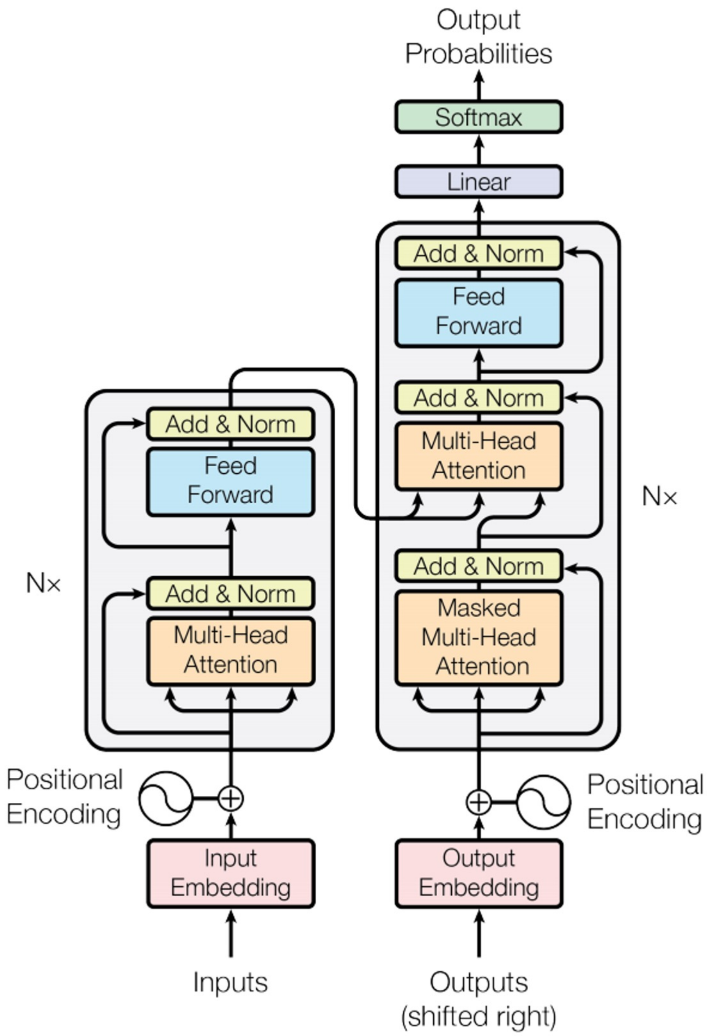
- rows of Q, K, V are keys, queries, values
- softmax acts row-wise



Transformer Architecture

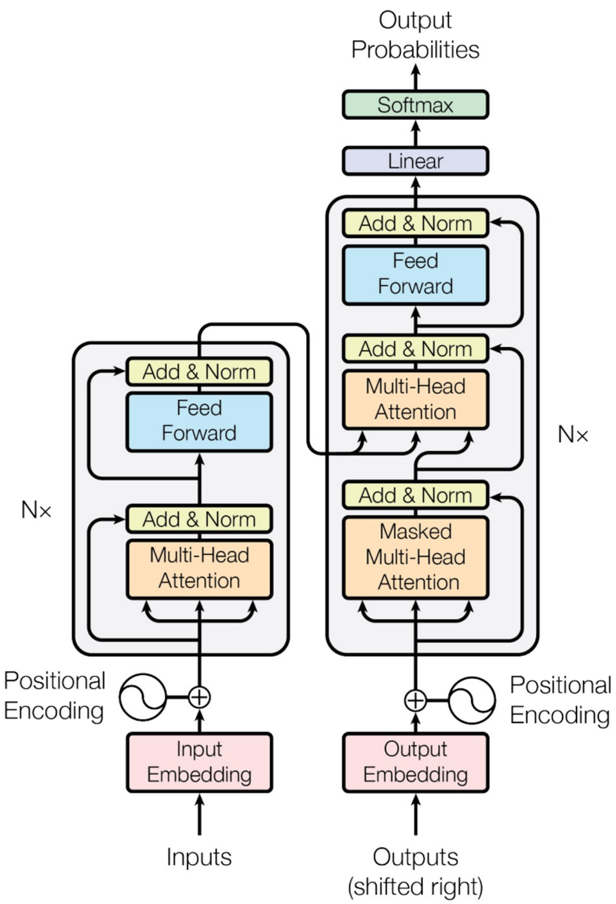
- introduces the self attention mechanism
  - No locality bias, i.e. long-distance context has "equal opportunity" as compared to LSTMs
- more efficient than RNNs/LSTMs
  - it breaks down the recurrent structure
  - Single multiplication per layer



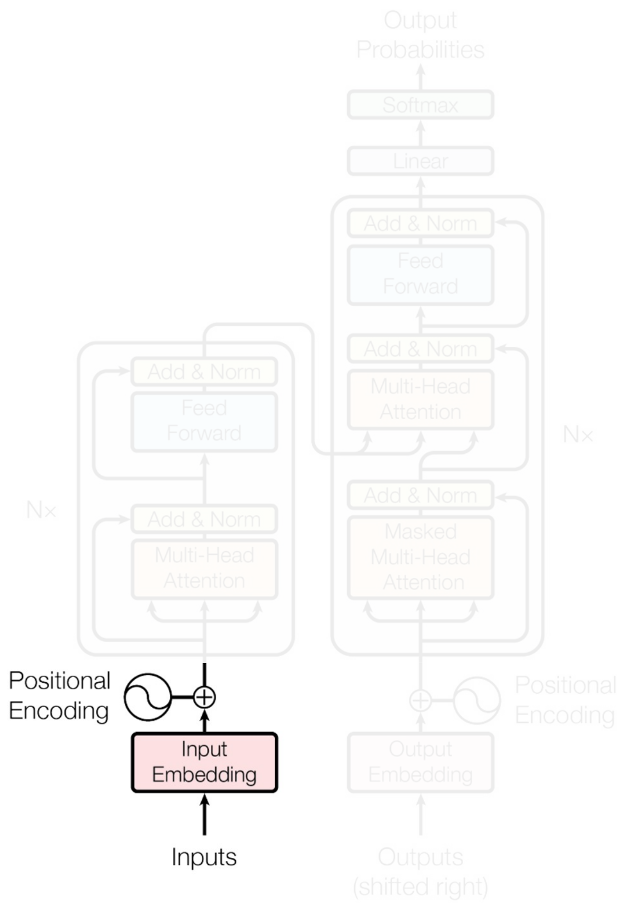


A. Vaswani et al. Attention Is All You Need. In NeurIPS 2017

Transformer Architecture



Transformer Architecture



Input (Tokenization and) Embedding

Input text is first split into pieces. Can be characters, word, "tokens":

```
"The detective investigated" -> [The_] [detective_] [invest] [igat] [ed_]
```

Tokens are indices into the "vocabulary":

```
[The_] [detective_] [invest] [igat] [ed_] -> [3 721 68 1337 42]
```

Each vocab entry corresponds to a learned d_{model}-dimensional vector.

```
[3 721 68 1337 42] -> [ [0.123, -5.234, ...], [...], [...], [...], [...] ]
```

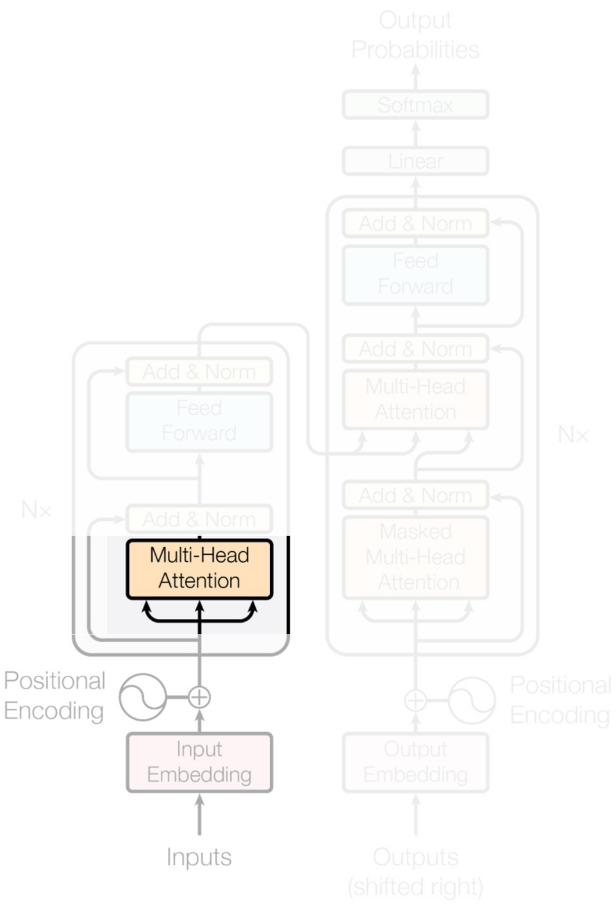
Positional Encoding

Remember attention is permutation invariant, but language is not!

Need to encode position of each word; just add something.

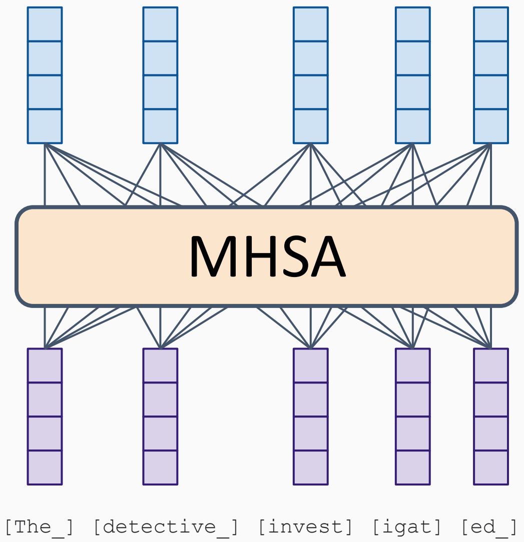
Think `[The_] + 10` `[detective_] + 20` `[invest] + 30` ... but smarter.

Transformer Architecture

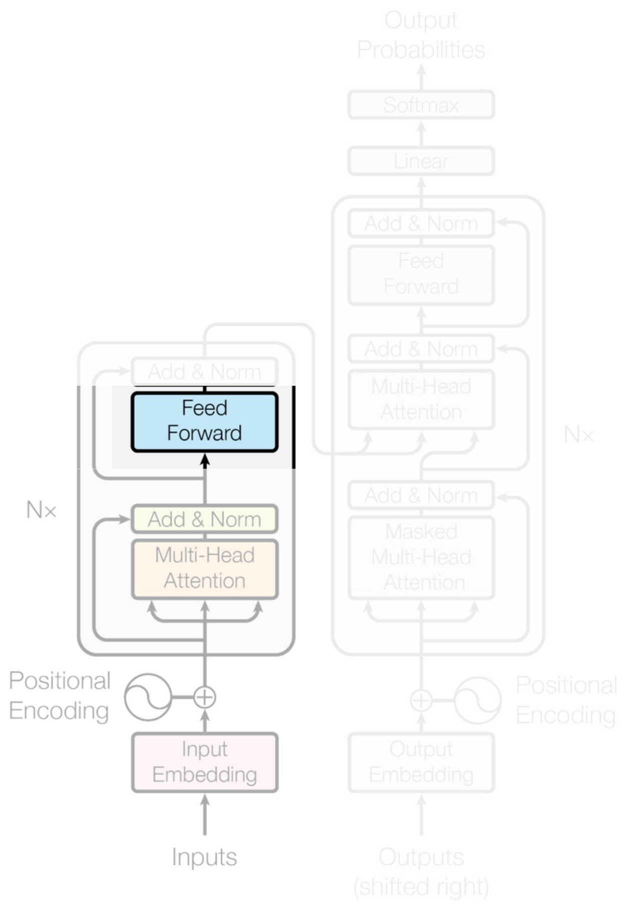


Multi-headed Self-Attention

Meaning the input sequence is used to create queries, keys, and values!
Each token can "look around" the whole input, and decide how to update its representation based on what it sees.



Transformer Architecture



Point-wise MLP

A simple MLP applied to each token individually:

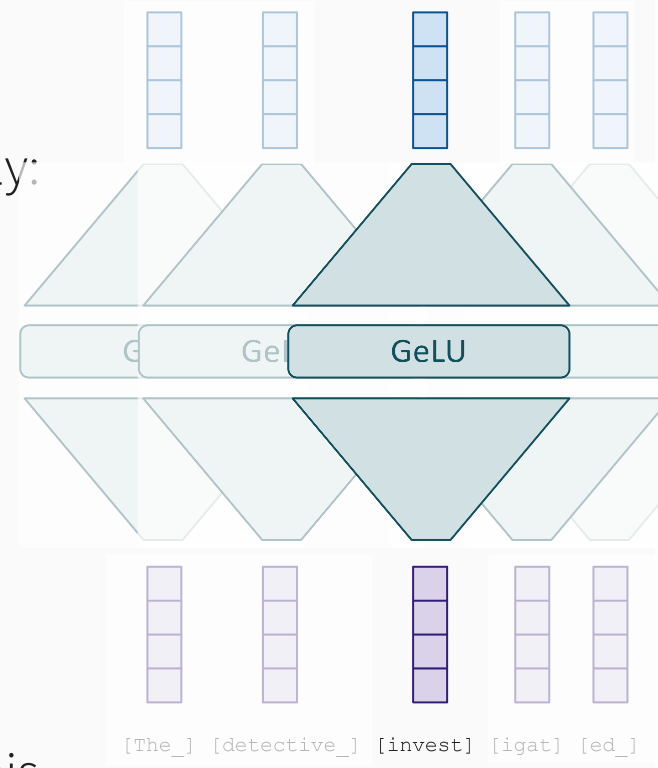
$$z_i = W_2\,\mathrm{GeLU}(W_1 x + b_1) + b_2$$

Think of it as each token pondering for itself about what it has observed previously.

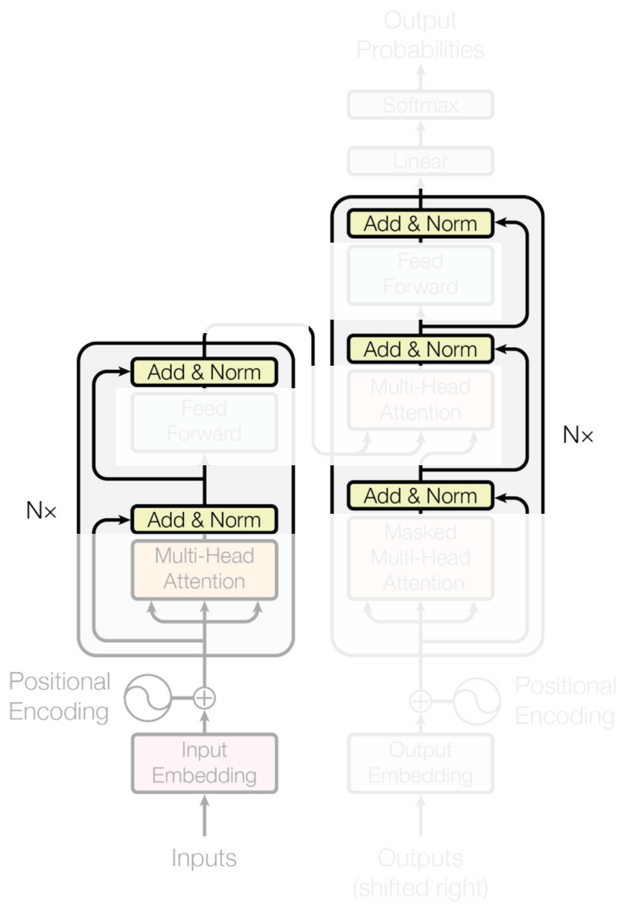
There's some weak evidence this is where "world knowledge" is stored, too.

It contains the bulk of the parameters. When people make giant models and sparse/moe, this is what becomes giant.

Some people like to call it 1x1 convolution.



Transformer Architecture



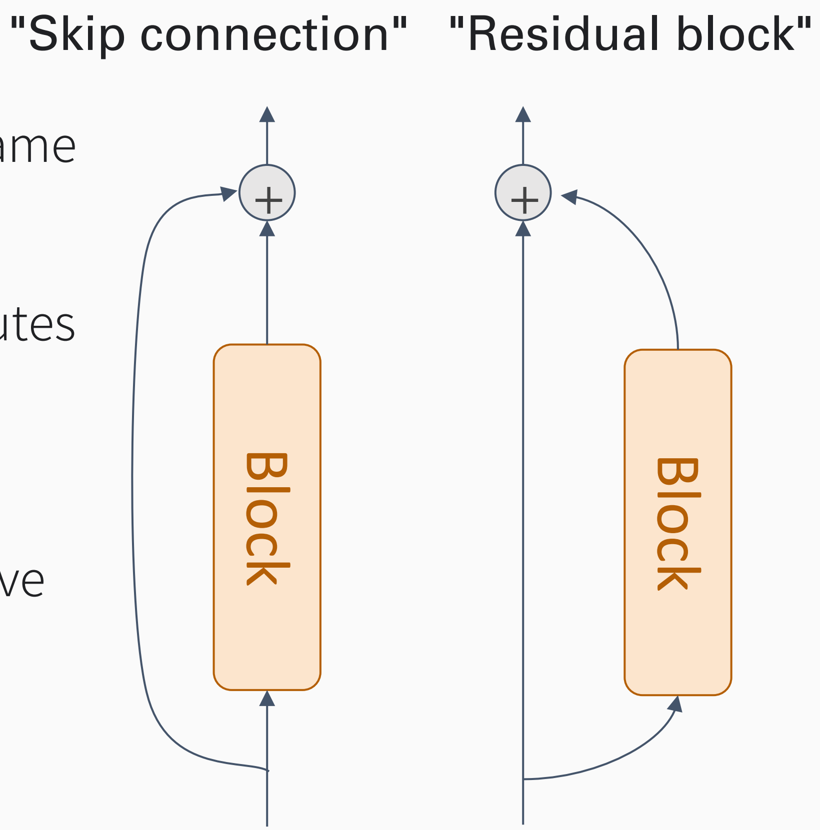
Residual connections

Each module's output has the exact same shape as its input.

Following ResNets, the module computes a "residual" instead of a new value:

$$z_i = \mathrm{Module}(x_i) + x_i$$

This was shown to dramatically improve trainability.



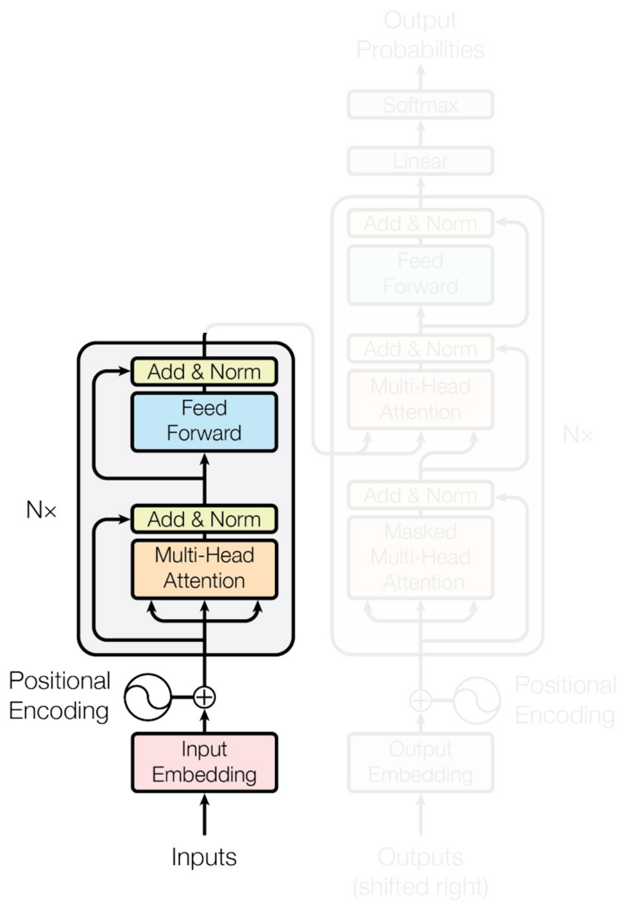
LayerNorm

Normalization also dramatically improves trainability.

There's **post-norm** (original) and **pre-norm** (modern)

$$z_i = \mathrm{LN}(\mathrm{Module}(x_i) + x_i) \qquad z_i = \mathrm{Module}(\mathrm{LN}(x_i)) + x_i$$

Transformer Architecture



Encoding / Encoder

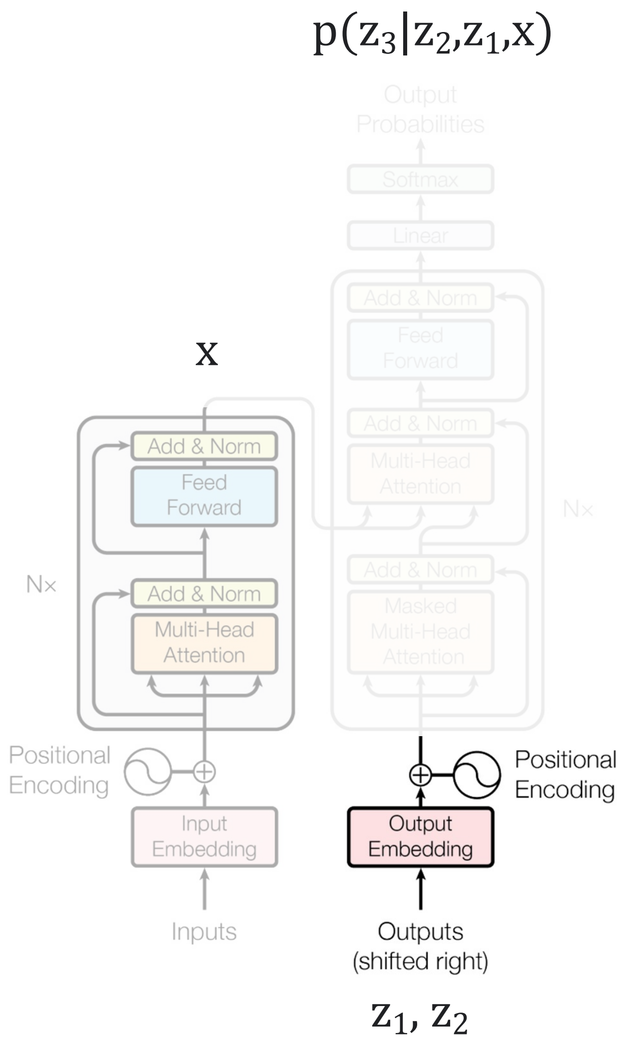
Since input and output shapes are identical, we can stack N such blocks.

Typically, N=6 ("base"), N=12 ("large") or more.

Encoder output is a "heavily processed" (think: "high level, contextualized") version of the input tokens, i.e. a sequence.

This has nothing to do with the requested output yet (think: translation). That comes with the decoder.

Transformer Architecture



Decoding / the Decoder (alternatively Generating / the Generator)

What we want to model: $p(z|x)$

e.g., in translation: $p(z \mid \text{"the detective investigated"})\ \forall z$

Seems impossible at first, but we can exactly decompose into tokens:

$$p(z|x) = p(z_1|x)\ p(z_2|z_1,x)\ p(z_3|z_2,z_1,x)\ldots$$

Meaning, we can generate the answer one token at a time.
Each p is a full pass through the model.

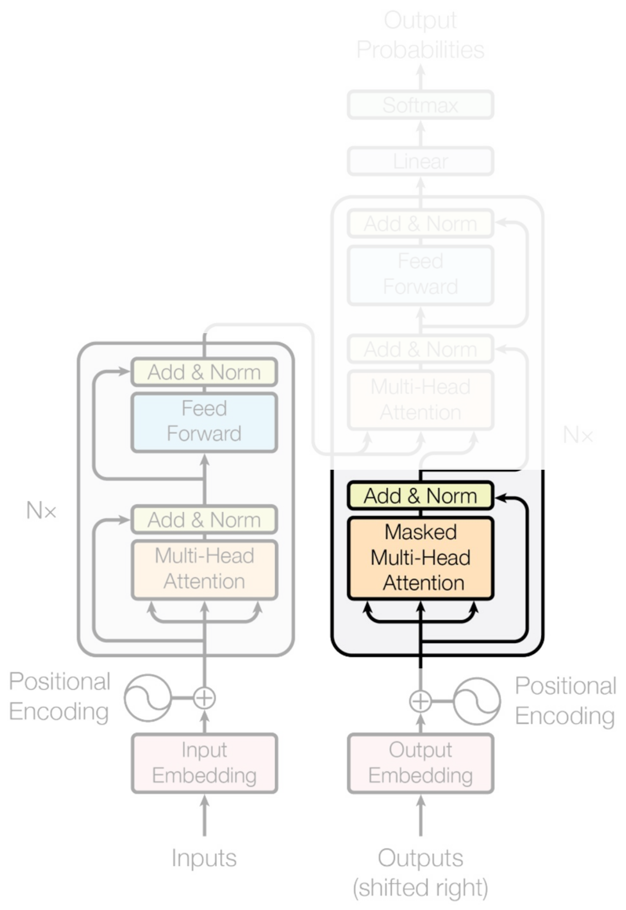
For generating $p(z_3|z_2,z_1,x)$:

x comes from the encoder,

z_1, z_2 is what we have predicted so far, goes into the decoder.

Once we have $p(z|x)$ we still need to actually sample a sentence such as **"le détective a enquêté"**. Many strategies: greedy, beam-search, ...

Transformer Architecture



Masked self-attention

This is regular self-attention as in the encoder, to process what's been decoded so far, but with a trick...

If we had to train on one single $p(z_3|z_2,z_1,x)$ at a time: SLOW!

Instead, train on all $p(z_i|z_{1:i},x)$ simultaneously.

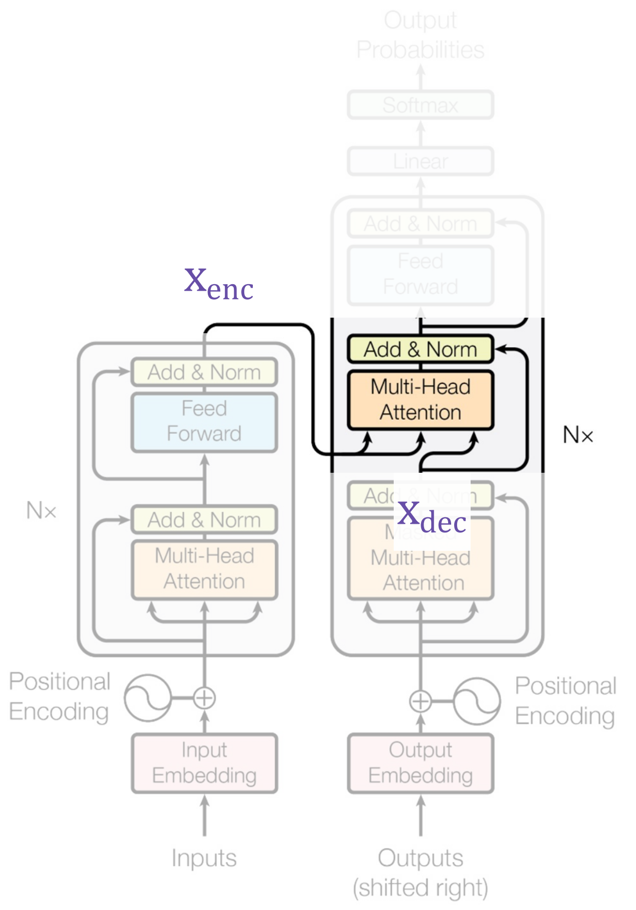
How? In the attention weights for z_i, set all entries $i{:}N$ to 0.

This way, each token only sees the already generated ones.

At generation time

There is no such trick. We need to generate one z_i at a time. This is why autoregressive decoding is extremely slow.

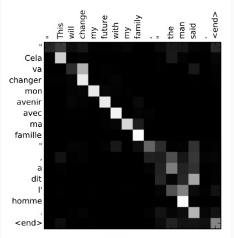
Transformer Architecture



"Cross" attention

Each decoded token can "look at" the encoder's output:

$$\mathrm{Attn}(q=W_q x_{dec},\ k=W_k x_{enc},\ v=W_v x_{enc})$$



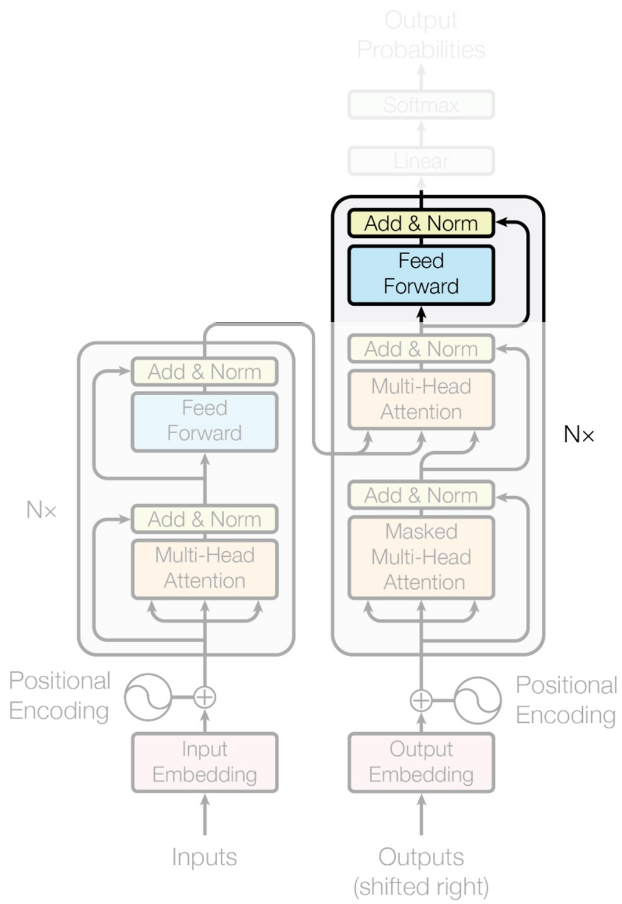
(d)

This is where $|x$ in $p(z_3|z_2,z_1,x)$ comes from.

Because self-attention is so widely used, people have started just calling it "attention".

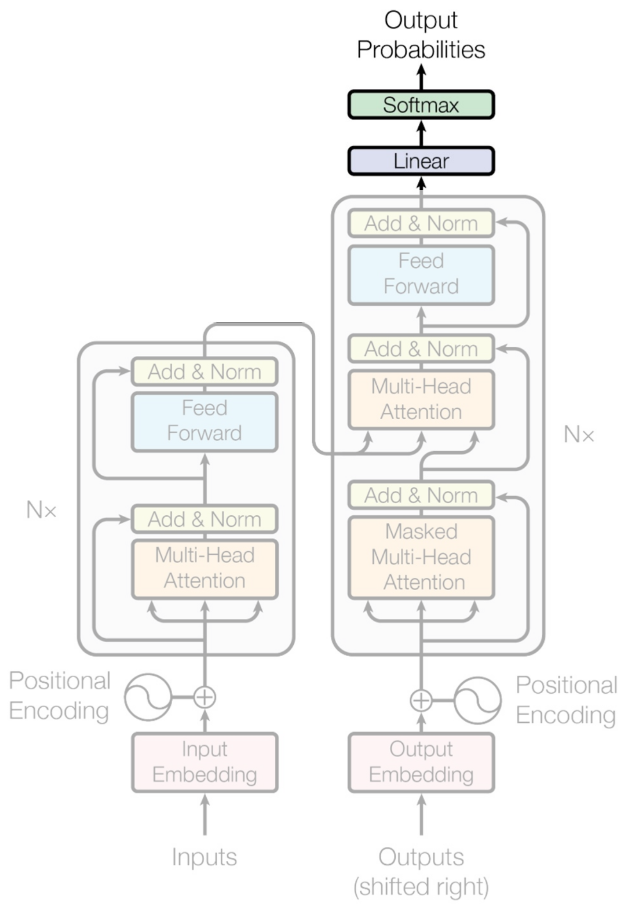
Hence, we now often need to explicitly call this "cross attention".

Transformer Architecture



Feedforward and stack layers.

Transformer Architecture



Output layer

Assume we have already generated K tokens, generate the next one.

The decoder was used to gather all information necessary to predict a probability distribution for the next token (K), over the whole vocab.

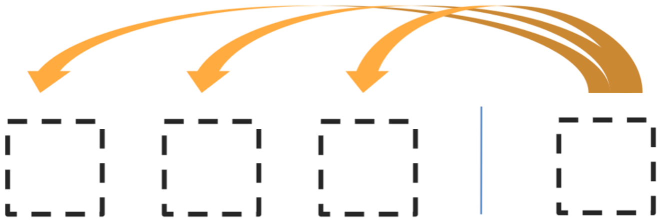
Simple:

linear projection of token K

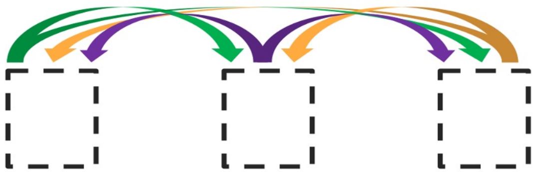
SoftMax normalization

Three types of attention in Transformer

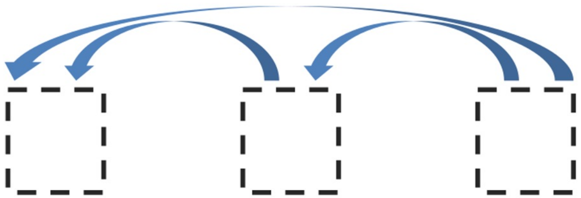
- usual attention between encoder and decoder:
  Q=[current state] K=V=[BiRNN states]

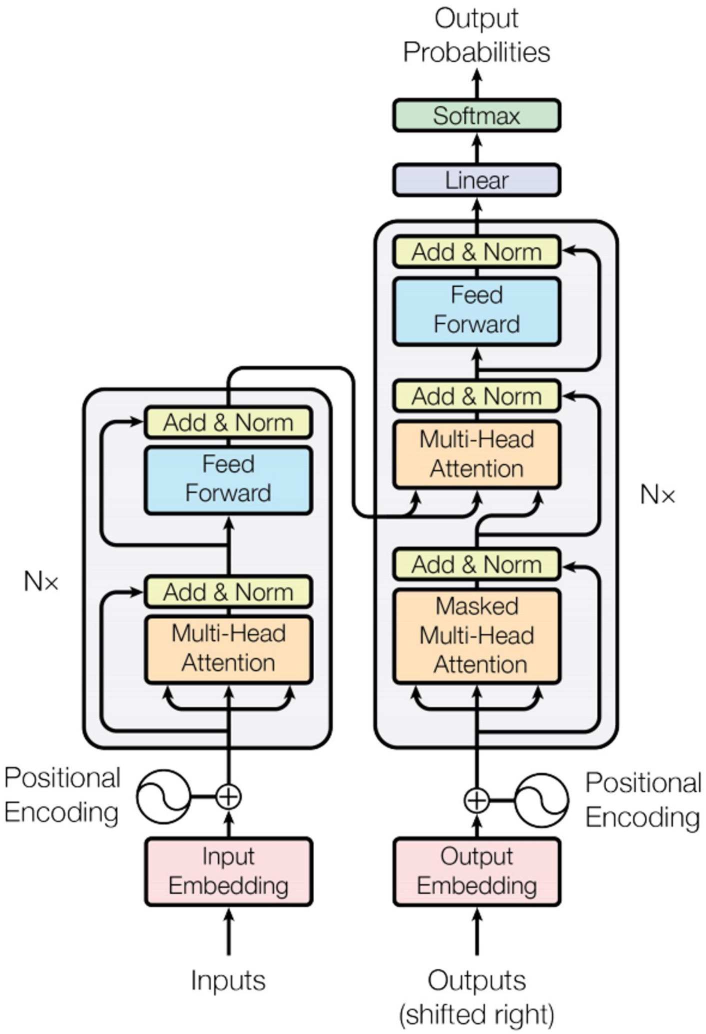


- self-attention in the encoder (encoder attends to itself!)
  Q=K=V=[encoder states]



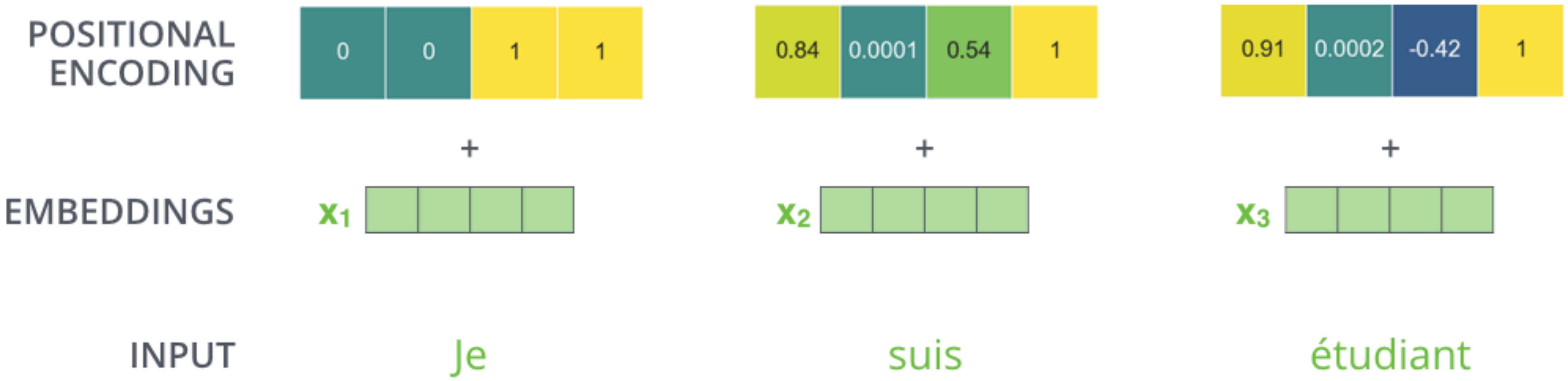
- masked self-attention in the decoder (attends to itself, but a states can only attend previous states)
  Q=K=V=[decoder states]





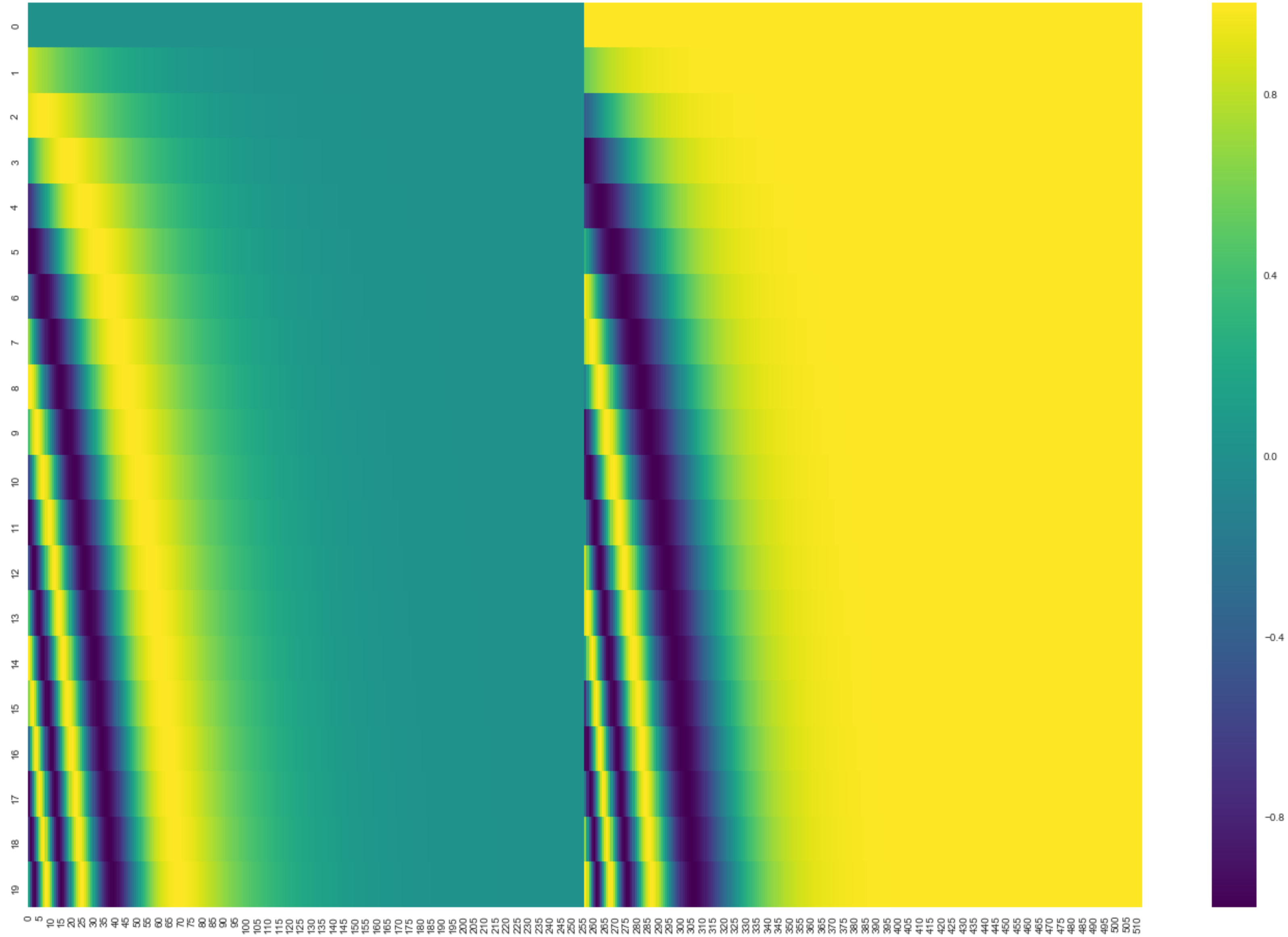
Positional Embeddings

- To give the model a sense of order
- Learned or predefined



Positional Embeddings

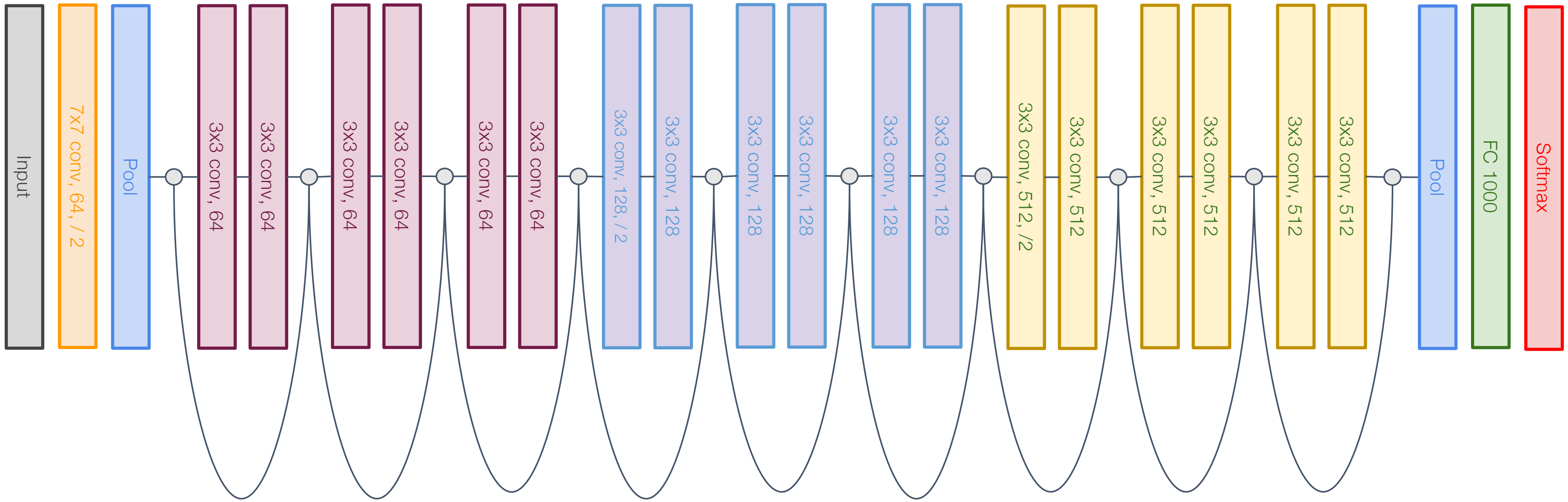
- What does it look like?



How to use Attention / Transformers for Vision?

Idea #1: Add attention to existing CNNs

Start from standard CNN architecture (e.g. ResNet)

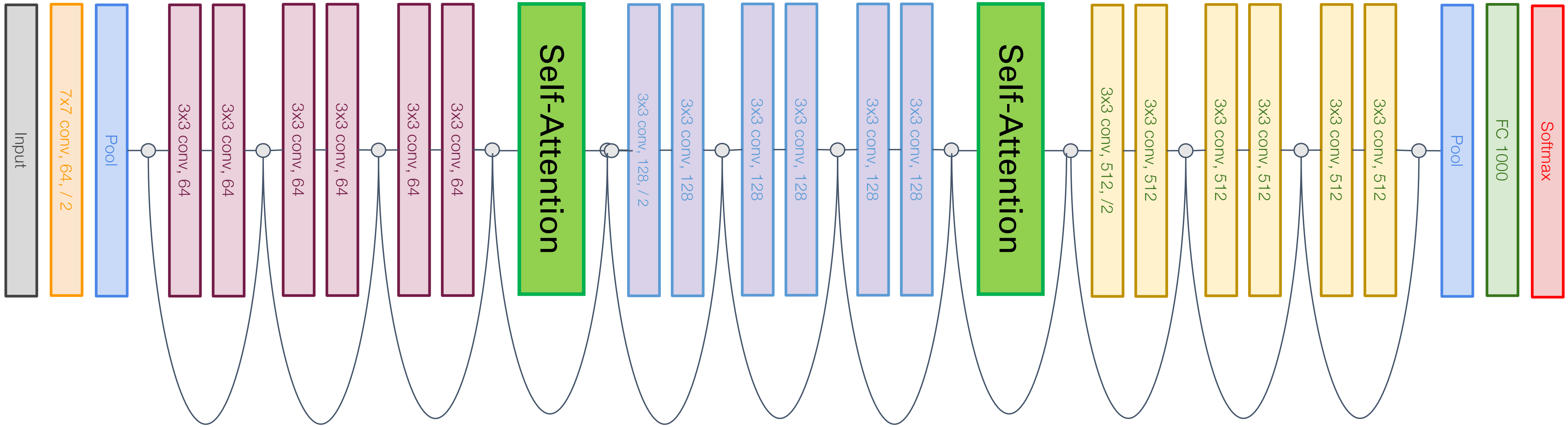


Zhang et al., "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al., "Non-local Neural Networks", CVPR 2018

Idea #1: Add attention to existing CNNs

Start from standard CNN architecture (e.g. ResNet)

Add Self-Attention blocks between existing ResNet blocks



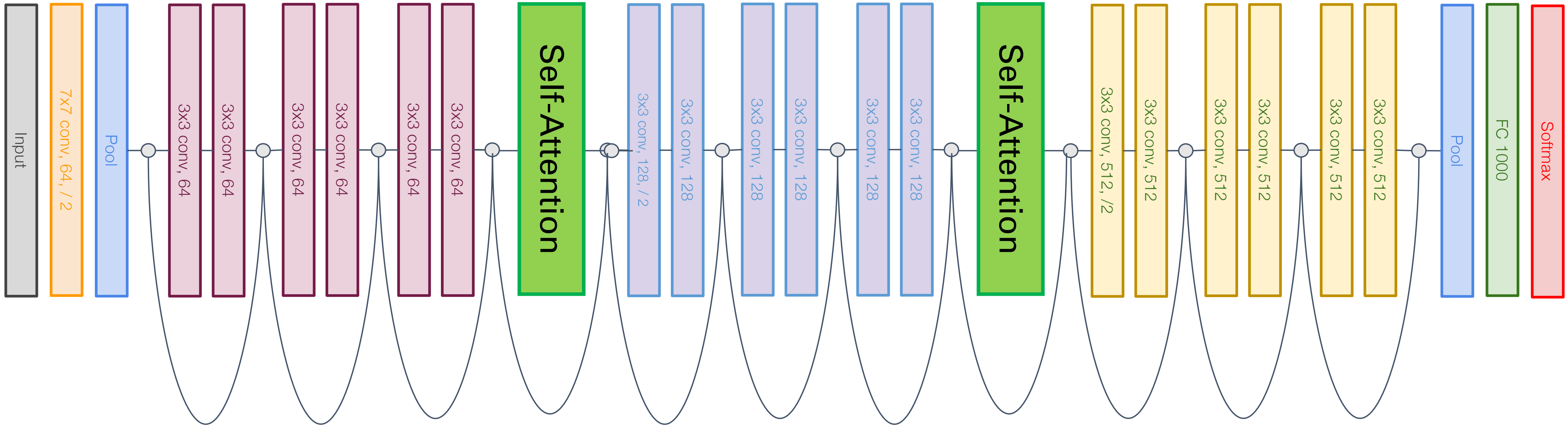
Zhang et al., "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al., "Non-local Neural Networks", CVPR 2018

Idea #1: Add attention to existing CNNs

Model is still a CNN!
Can we replace convolution entirely?

Start from standard CNN architecture (e.g. ResNet)

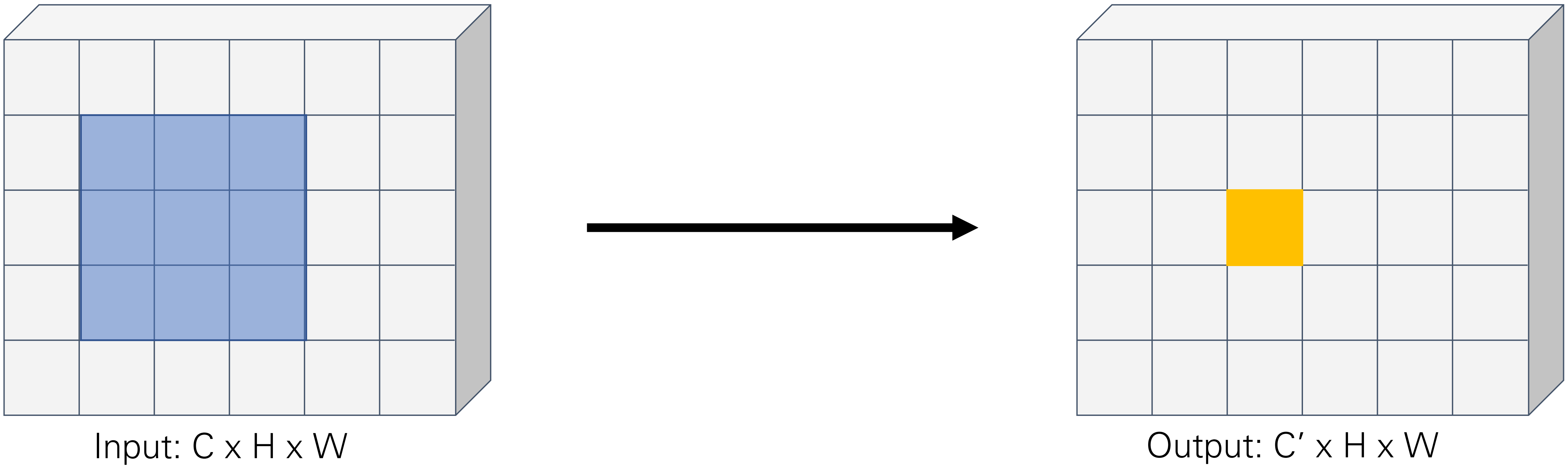
Add Self-Attention blocks between existing ResNet blocks



Zhang et al., "Self-Attention Generative Adversarial Networks", ICML 2018
Wang et al., "Non-local Neural Networks", CVPR 2018

Idea #2: Replace Convolution with "Local Attention"

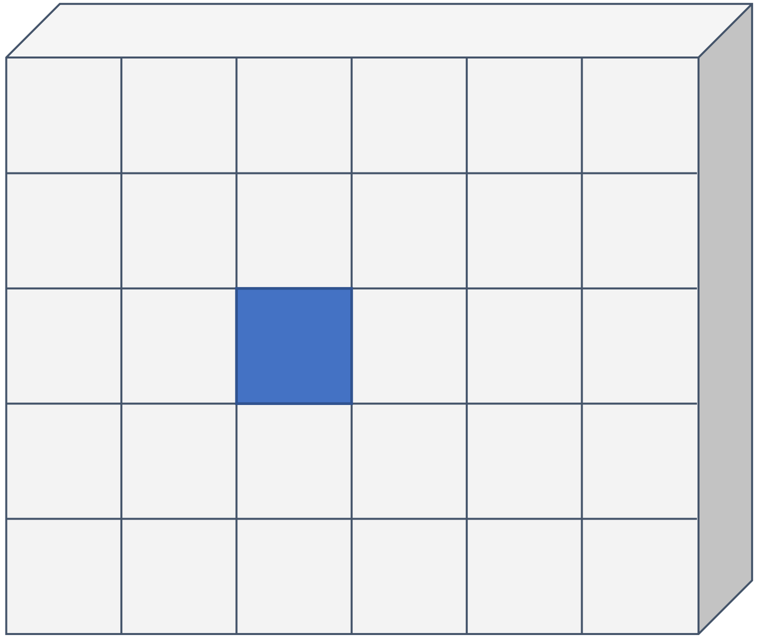
Convolution: Output at each position is inner product of conv kernel with receptive field in input



Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

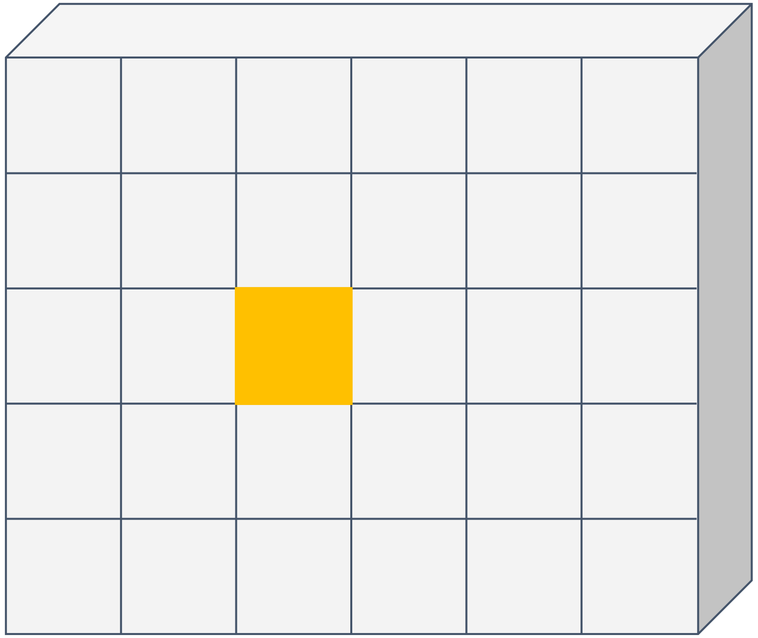
Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to query



Query: D_Q

Input: C x H x W

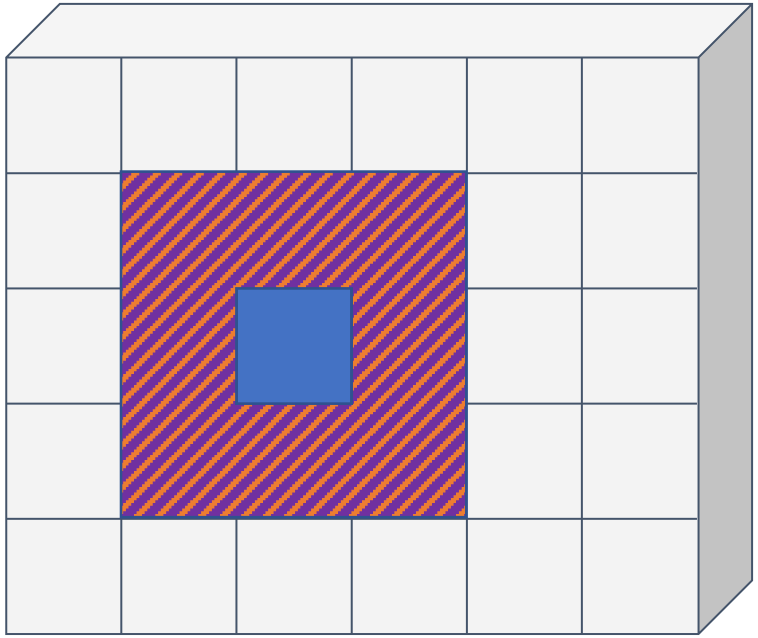


Output: C' x H x W

Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

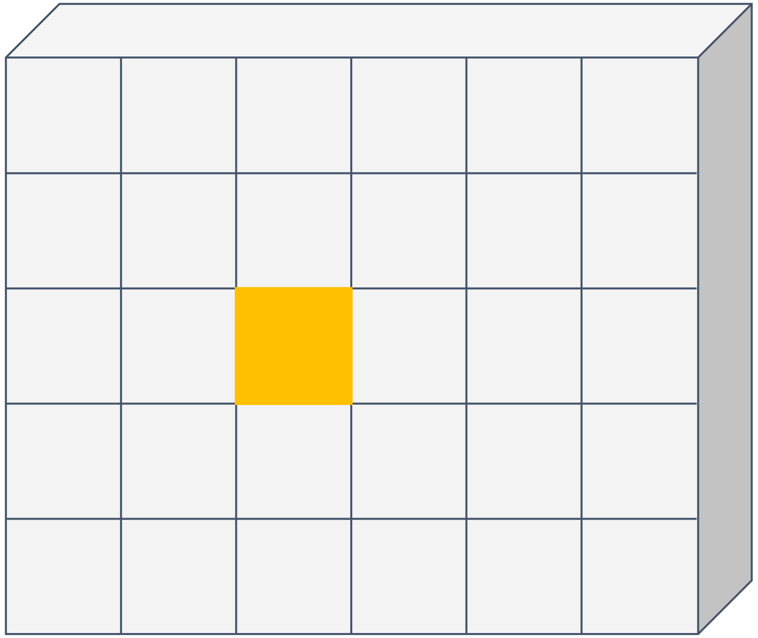
Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to query
Map each element in receptive field to key and value



Input: C x H x W

Query: D_Q
Keys: R x R x D_Q
Values: R x R x C'

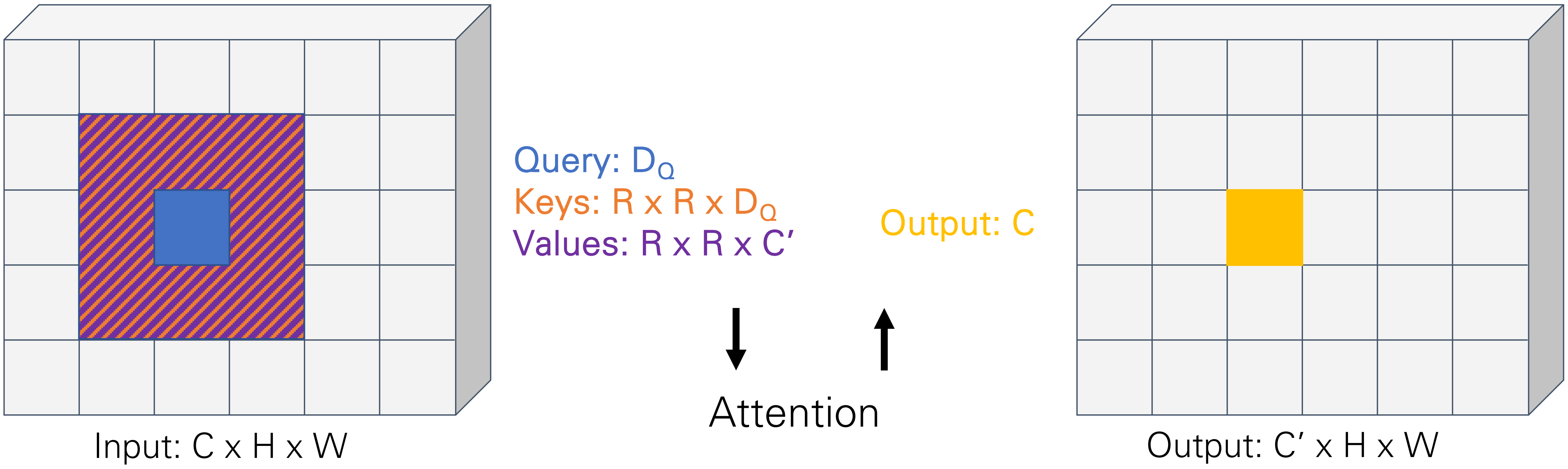


Output: C' x H x W

Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to query
Map each element in receptive field to key and value
Compute output using attention



Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to query
Map each element in receptive field to key and value
Compute output using attention
Replace all conv in ResNet with local attention

LR = "Local Relation"

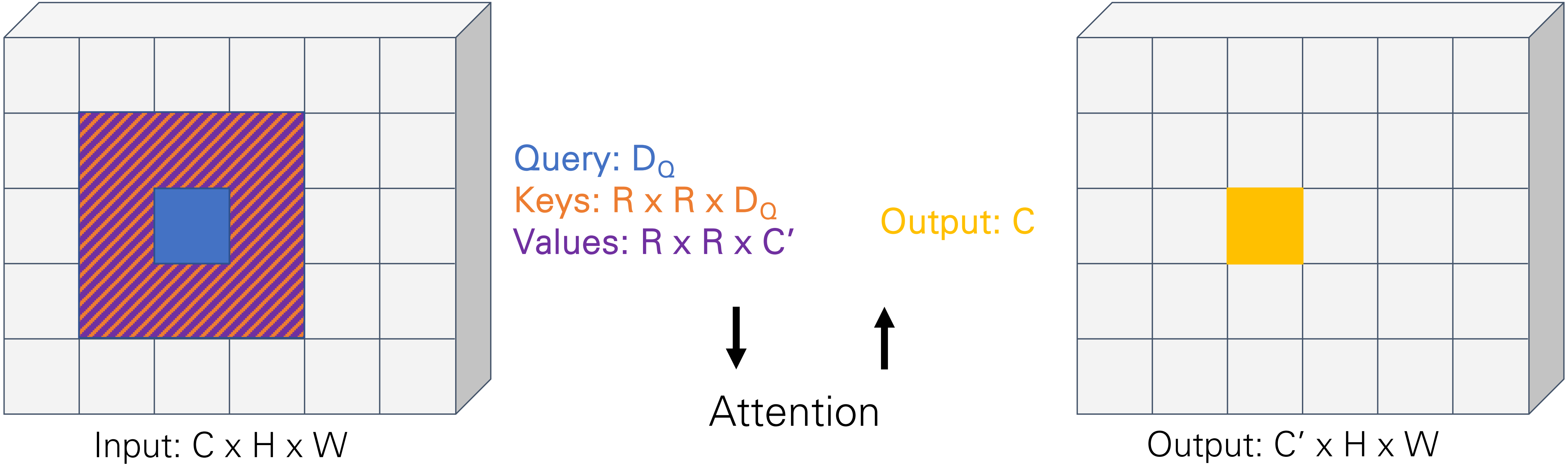
stage	output	ResNet-50	LR-Net-50 (7×7, m=8)
res1	112×112	7×7 conv, 64, stride 2	**1×1, 64** **7×7 LR, 64, stride 2**
res2	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3 \text{ conv}, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 100 \\ \mathbf{7\times7 \text{ LR}, 100} \\ 1\times1, 256 \end{bmatrix}\times3$
res3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3 \text{ conv}, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 200 \\ \mathbf{7\times7 \text{ LR}, 200} \\ 1\times1, 512 \end{bmatrix}\times4$
res4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3 \text{ conv}, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 400 \\ \mathbf{7\times7 \text{ LR}, 400} \\ 1\times1, 1024 \end{bmatrix}\times6$
res5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3 \text{ conv}, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 800 \\ \mathbf{7\times7 \text{ LR}, 800} \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		**25.5**$\times10^6$	**23.3**$\times10^6$
FLOPs		**4.3**$\times10^9$	**4.3**$\times10^9$

Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #2: Replace Convolution with "Local Attention"

Map center of receptive field to query
Map each element in receptive field to key and value
Compute output using attention
Replace all conv in ResNet with local attention

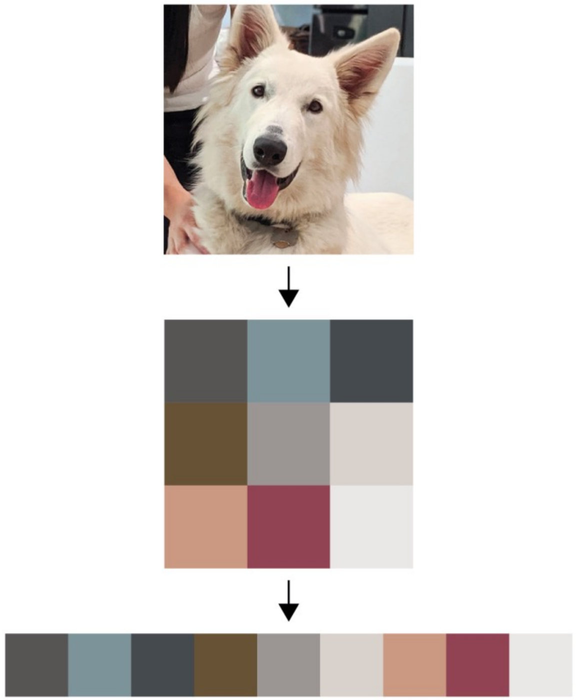
Lots of tricky details, hard to implement, only marginally better than ResNets

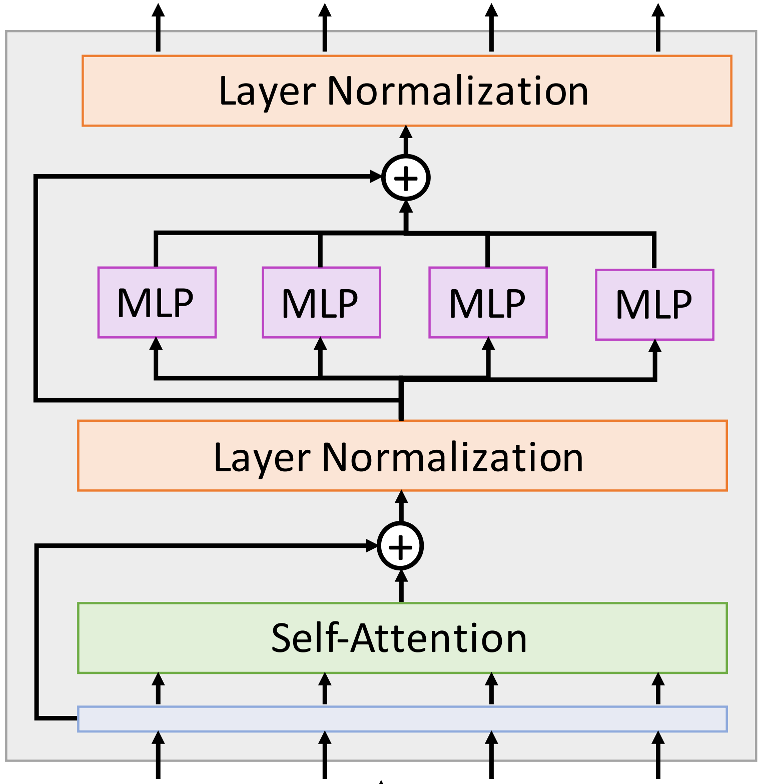


Hu et al., "Local Relation Networks for Image Recognition", ICCV 2019;
Ramachandran et al., "Stand-Alone Self-Attention in Vision Models", NeurIPS 2019

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values



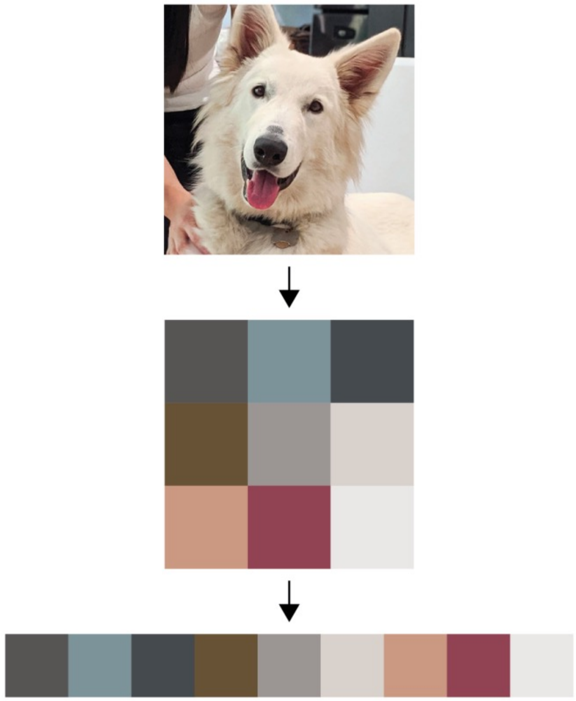


Feed as input to standard Transformer

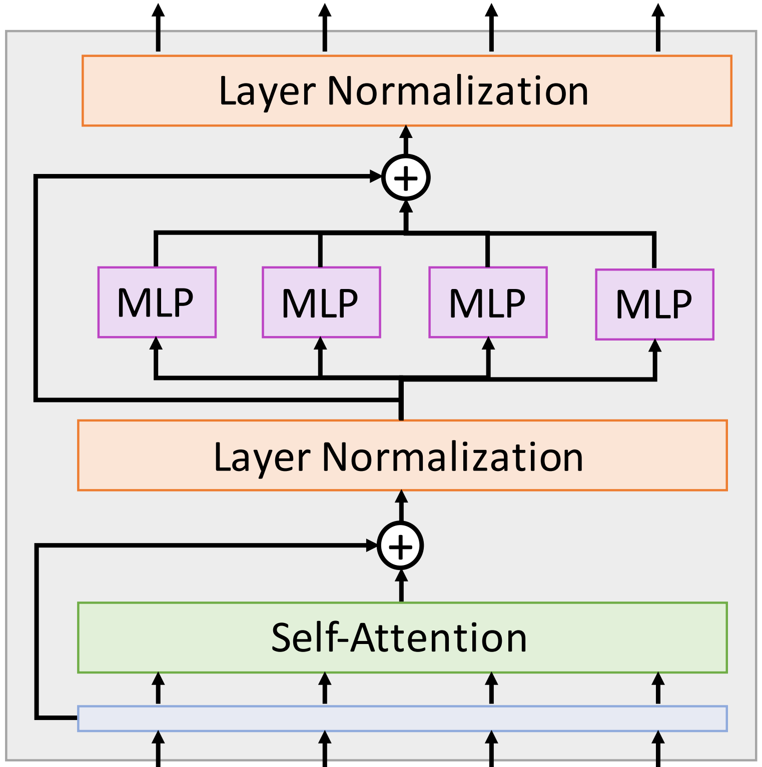
Chen et al., "Generative Pretraining from Pixels", ICML 2020

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values



Feed as input to standard Transformer

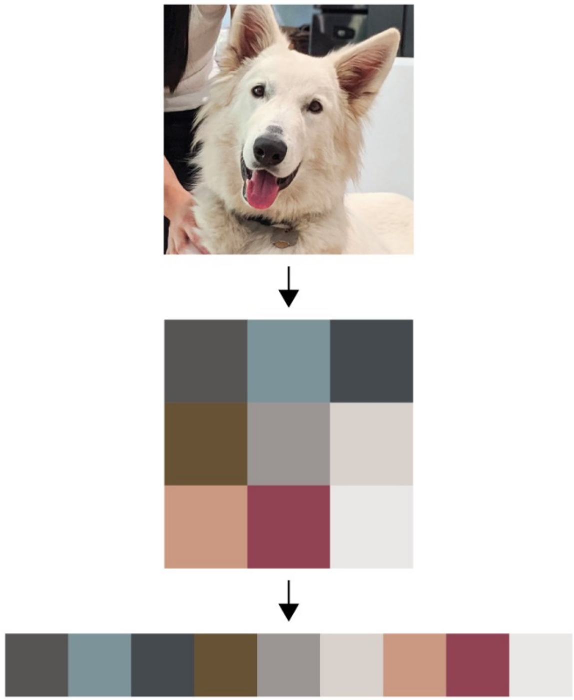


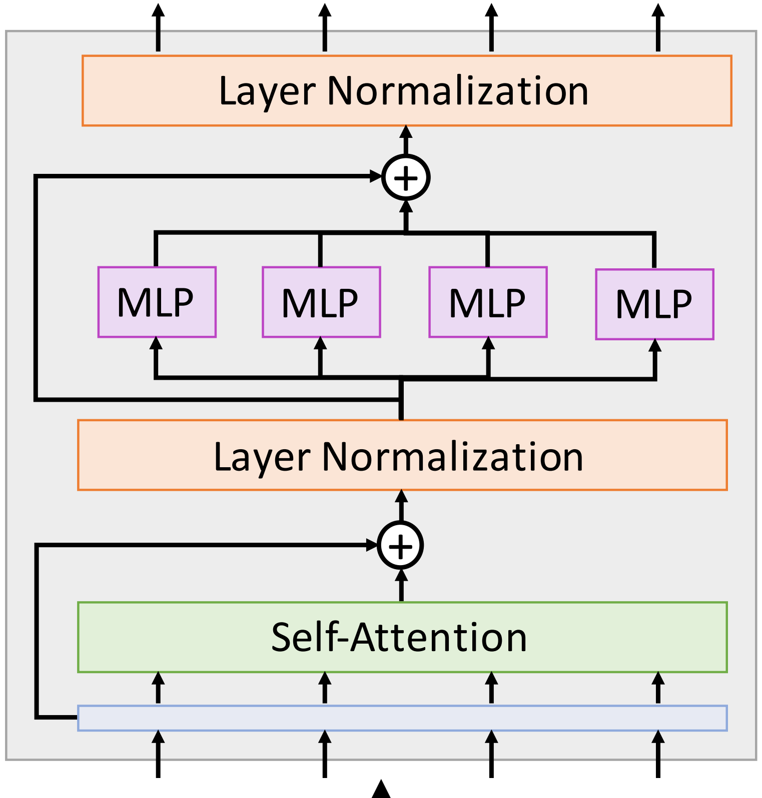
Problem: Memory use!

R x R image needs R^4 elements per attention matrix

Idea #3: Standard Transformer on Pixels

Treat an image as a set of pixel values





Feed as input to standard Transformer

Problem: Memory use!

R x R image needs R^4 elements per attention matrix

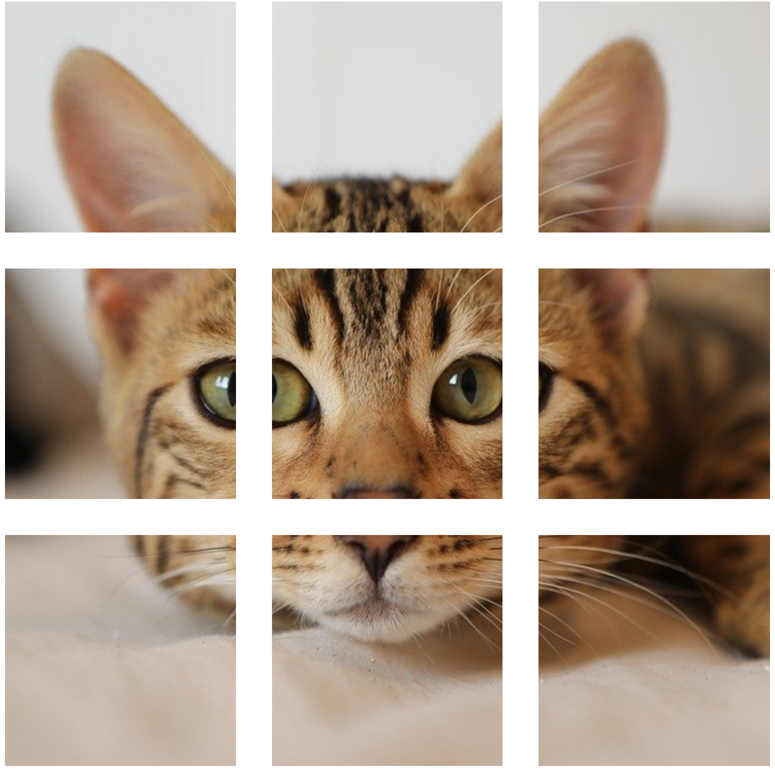
R=128, 48 layers, 16 heads per layer takes 768GB of memory for attention matrices for a single example…

Chen et al., "Generative Pretraining from Pixels", ICML 2020

Idea #4: Standard Transformer on Patches

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches



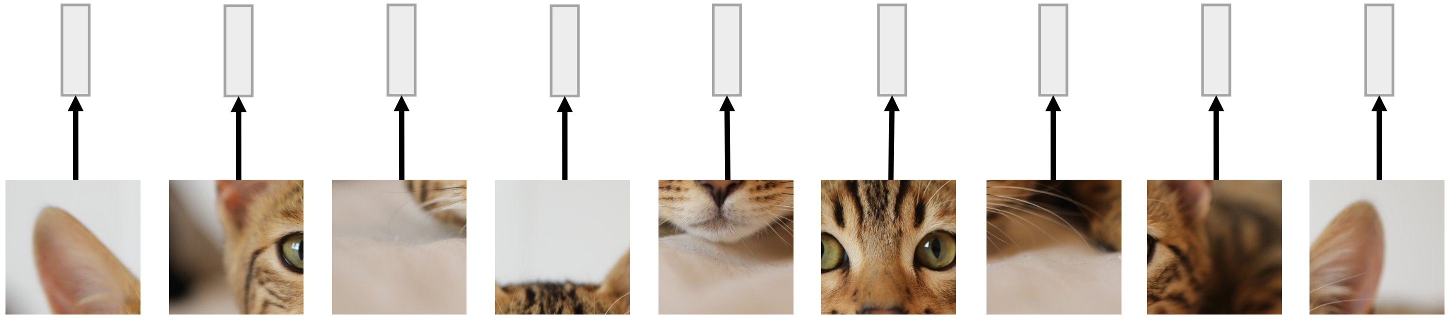
Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches

N input patches, each
of shape 3x16x16

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches



Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16

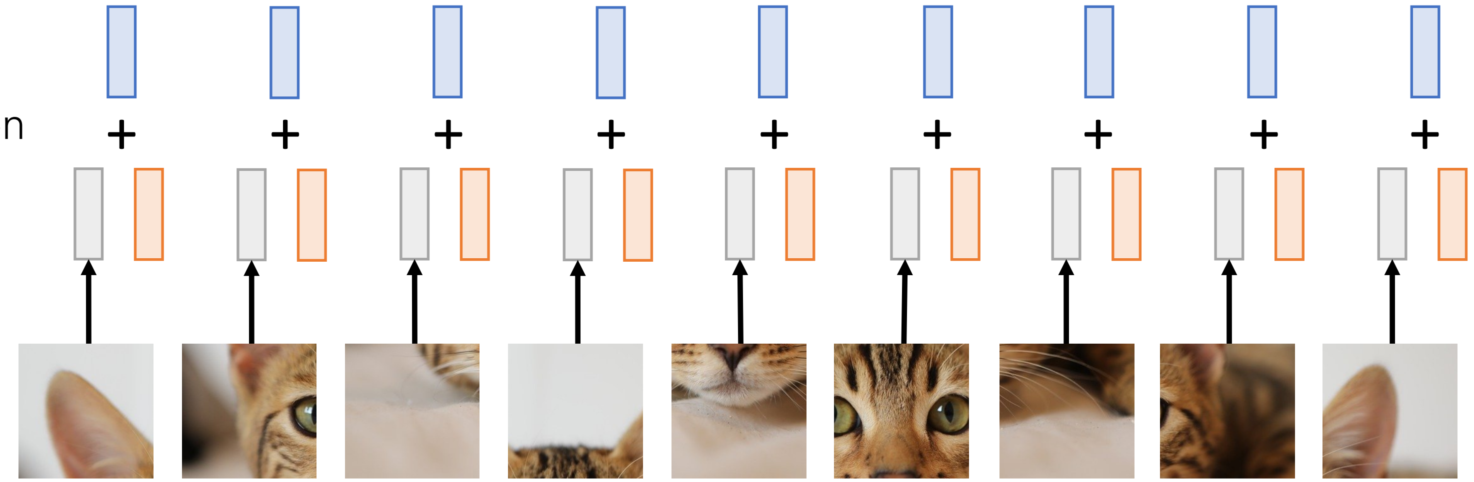
Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches

Add positional embedding: learned D-dim vector per position

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16



Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches

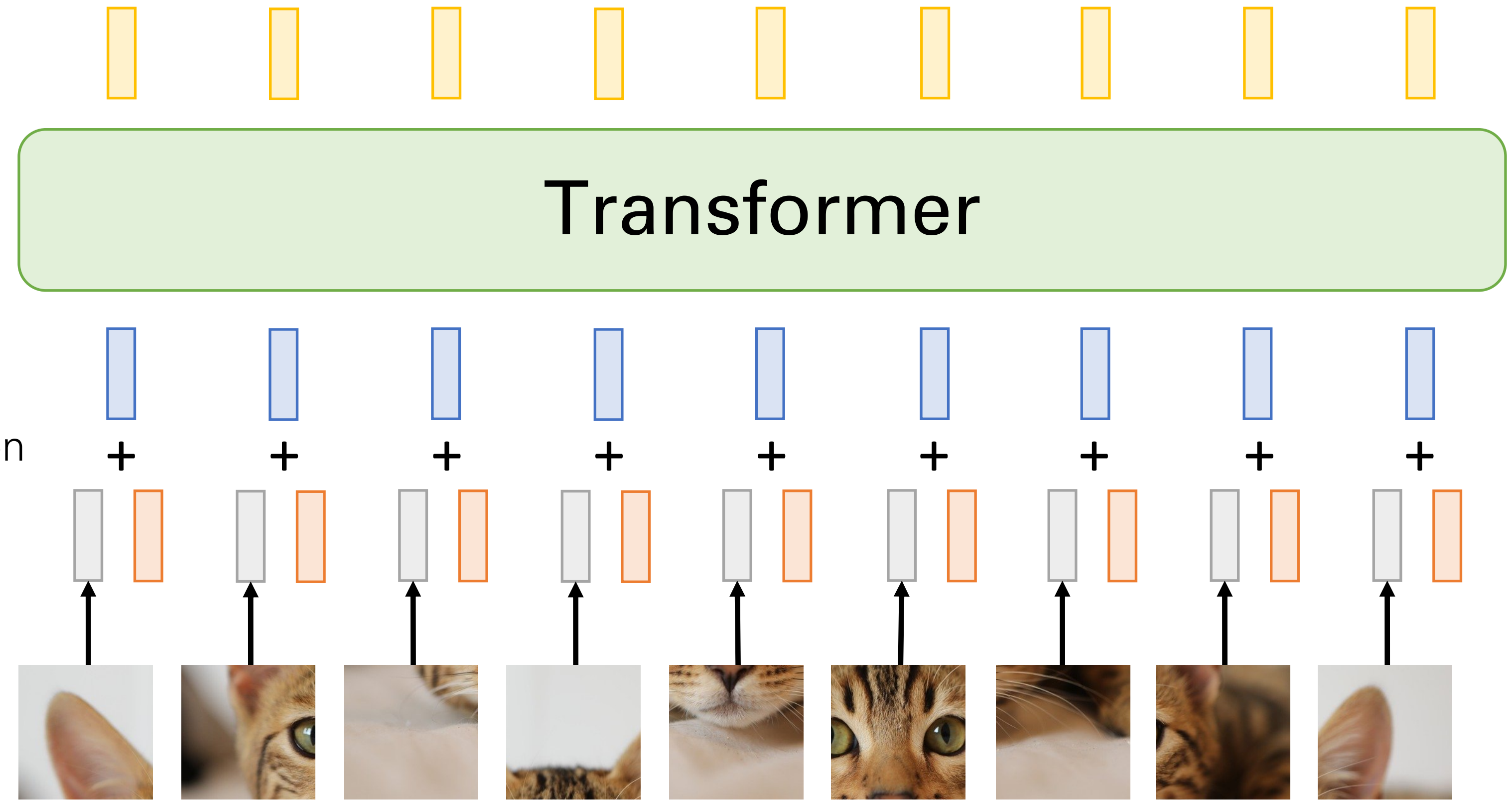
Output vectors

Exact same as
NLP Transformer!

Add positional
embedding: learned
D-dim vector per position

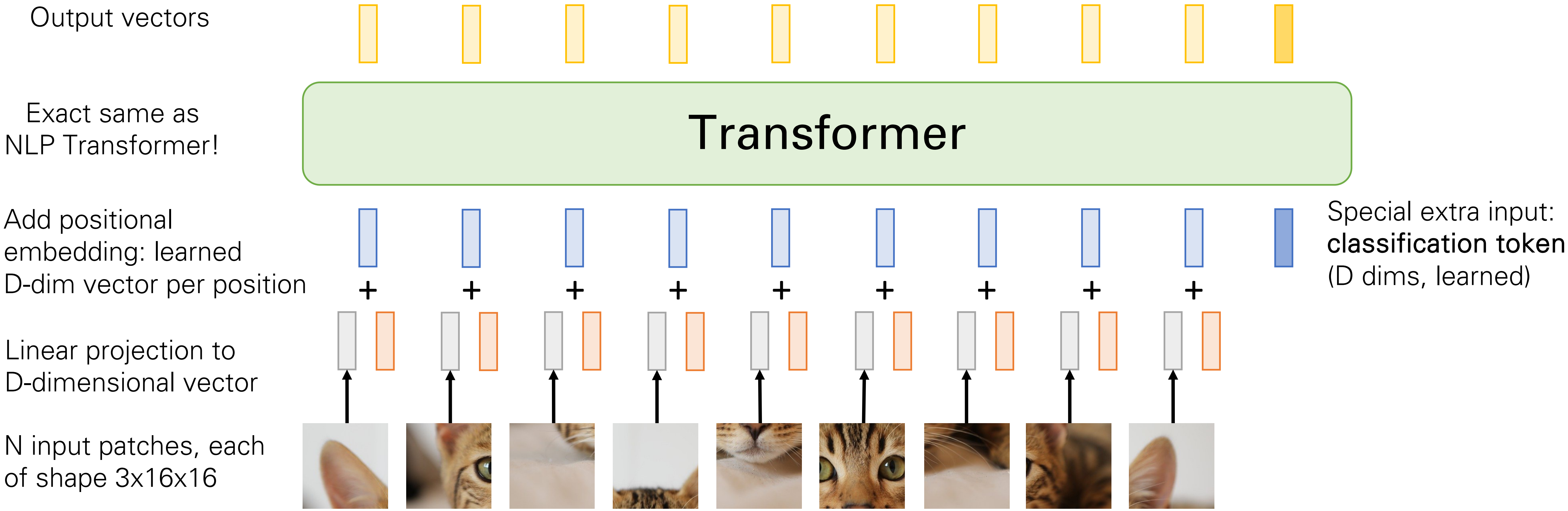
Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16



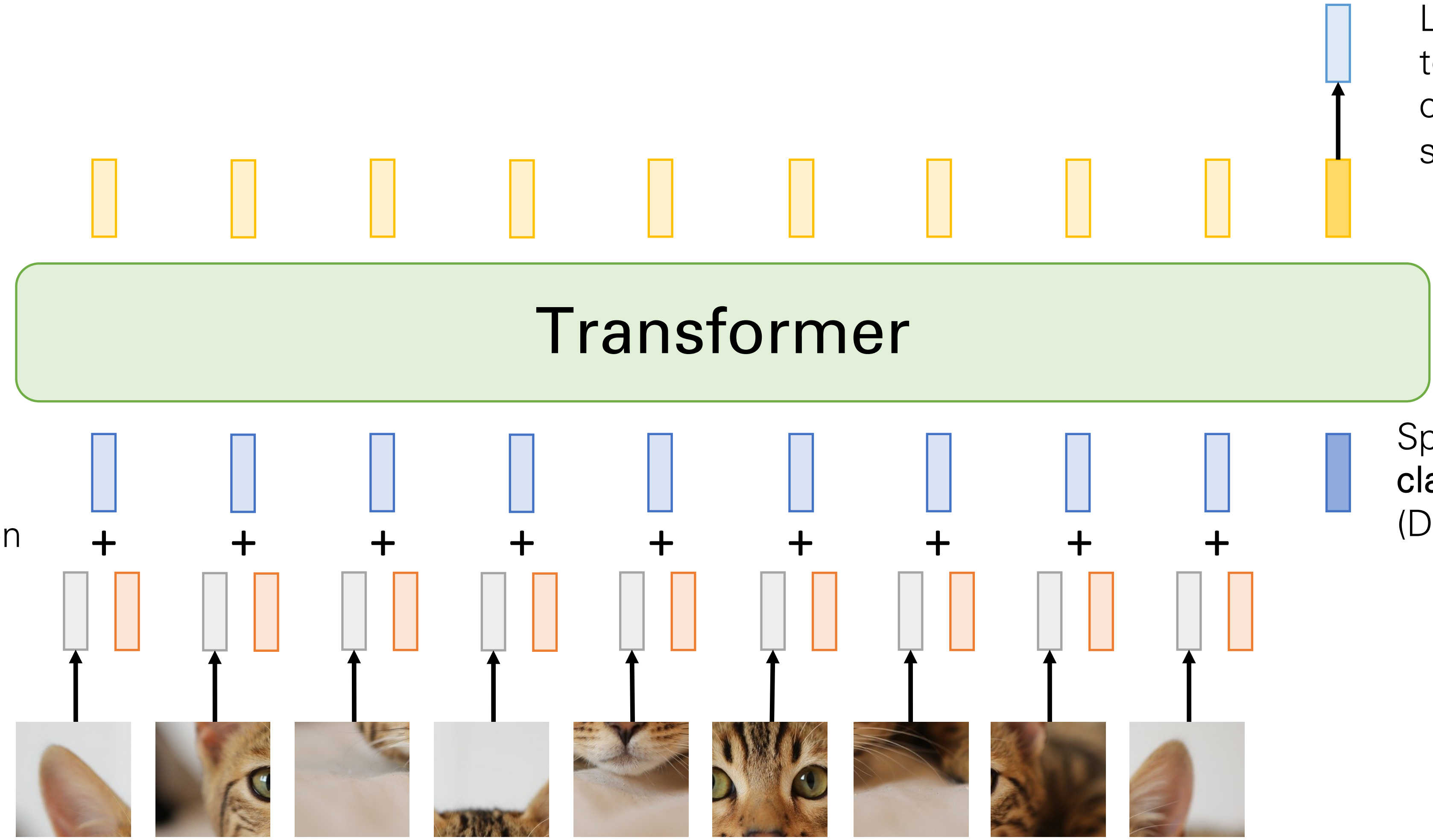
Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches



Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Idea #4: Standard Transformer on Patches



Output vectors

Exact same as
NLP Transformer!

Add positional
embedding: learned
D-dim vector per position

Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16

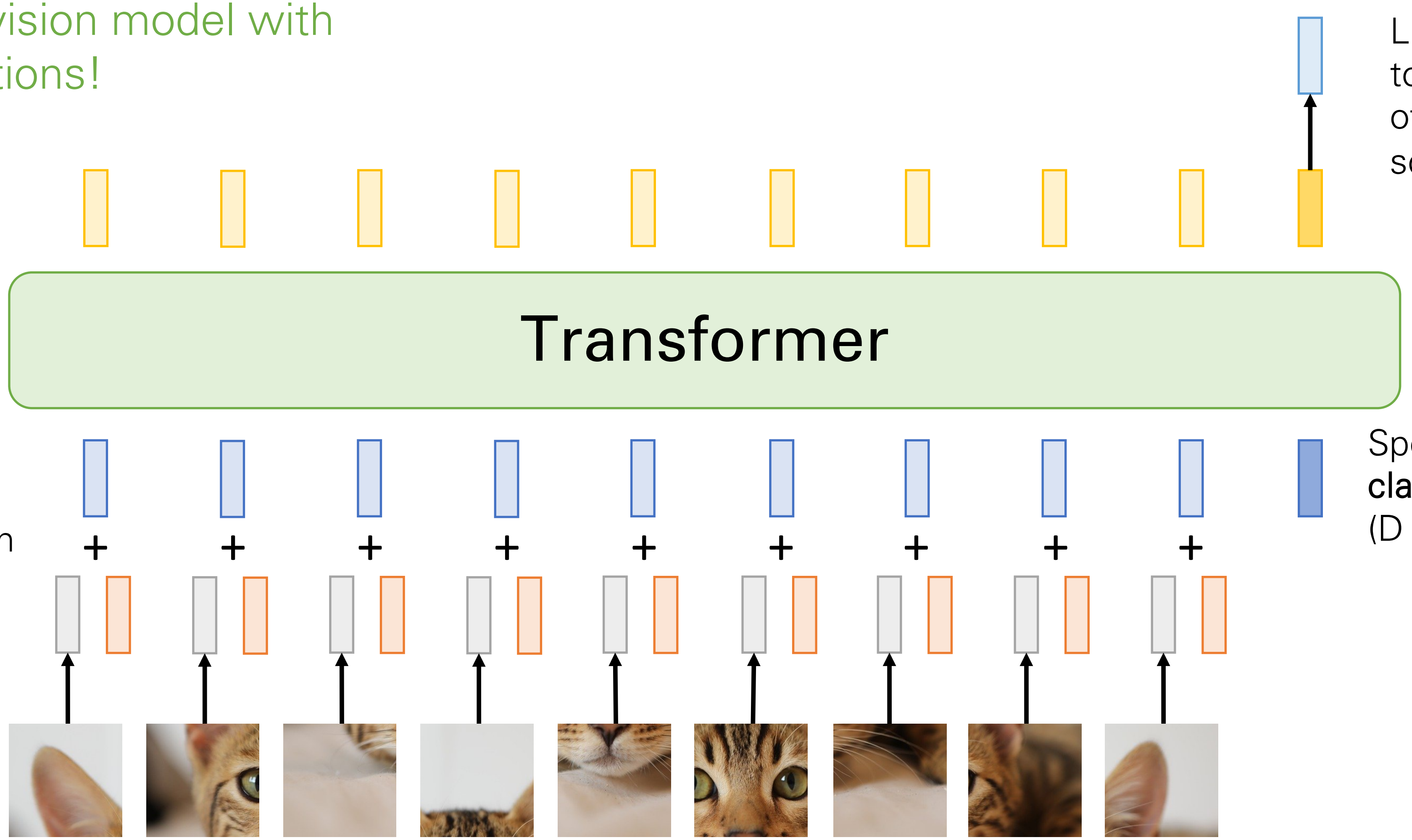
Linear projection
to C-dim vector
of predicted class
scores

Special extra input:
classification token
(D dims, learned)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT)

Computer vision model with
no convolutions!



Linear projection
to C-dim vector
of predicted class
scores

Output vectors

Exact same as
NLP Transformer!

Add positional
embedding: learned
D-dim vector per position

Special extra input:
classification token
(D dims, learned)

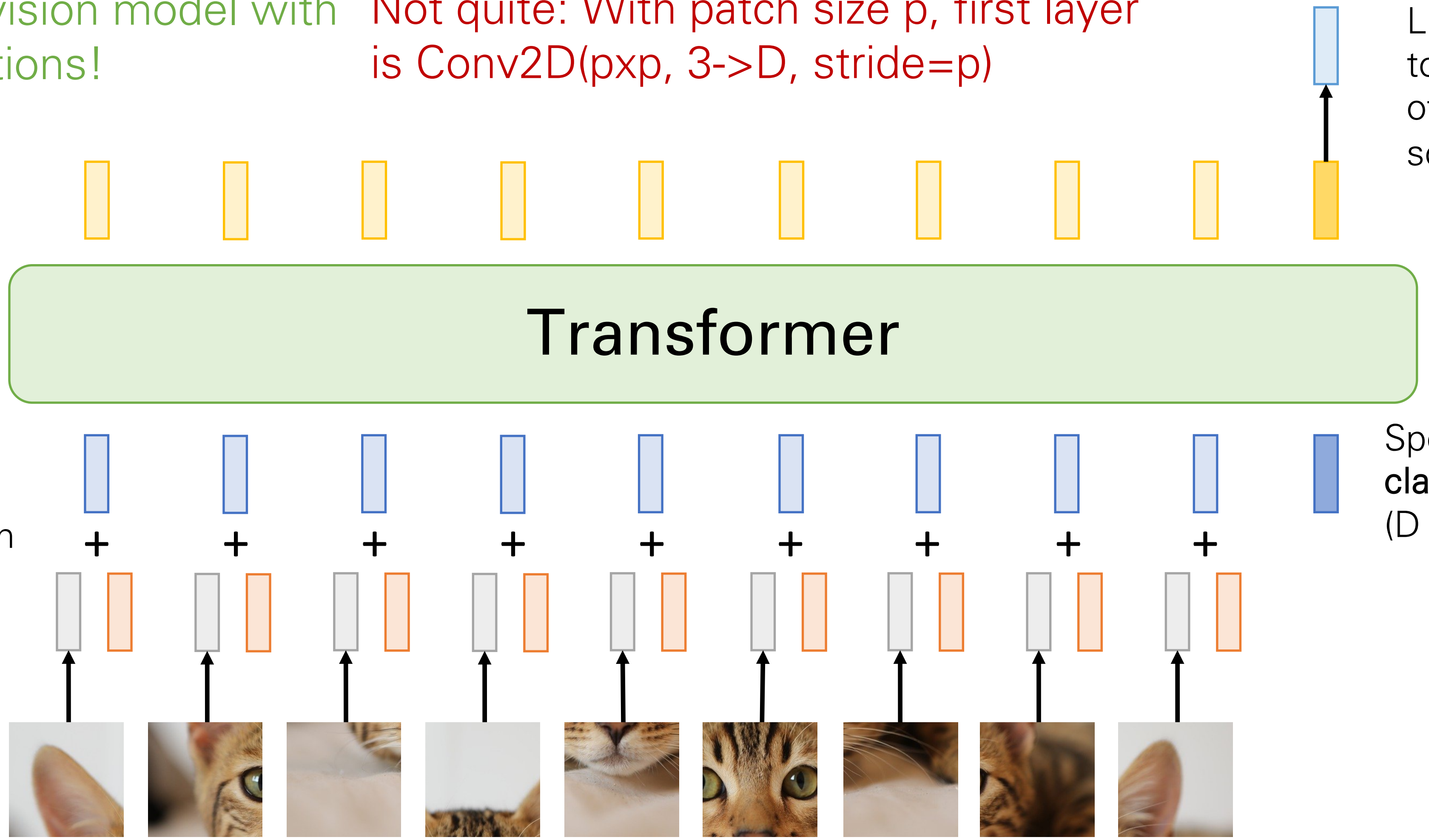
Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16

Vision Transformer (ViT)

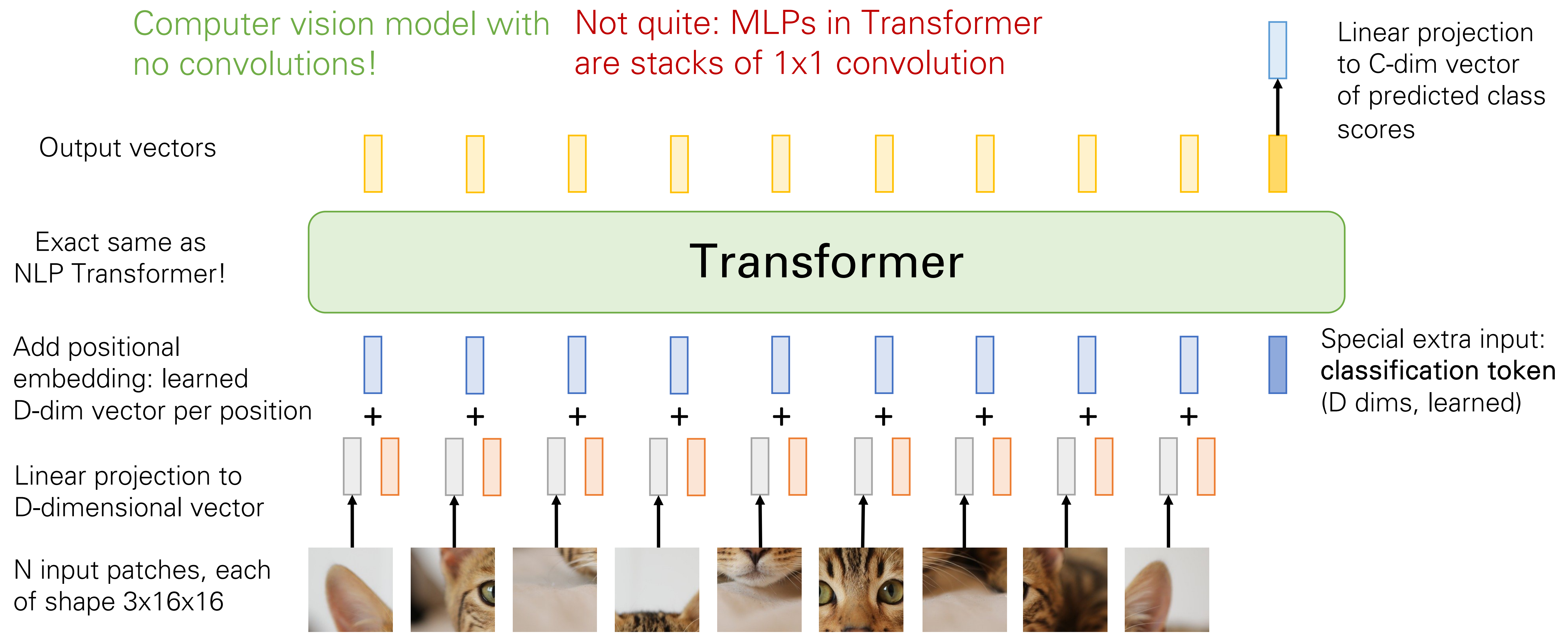
Computer vision model with no convolutions!

Not quite: With patch size p, first layer is Conv2D(pxp, 3->D, stride=p)



Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT)

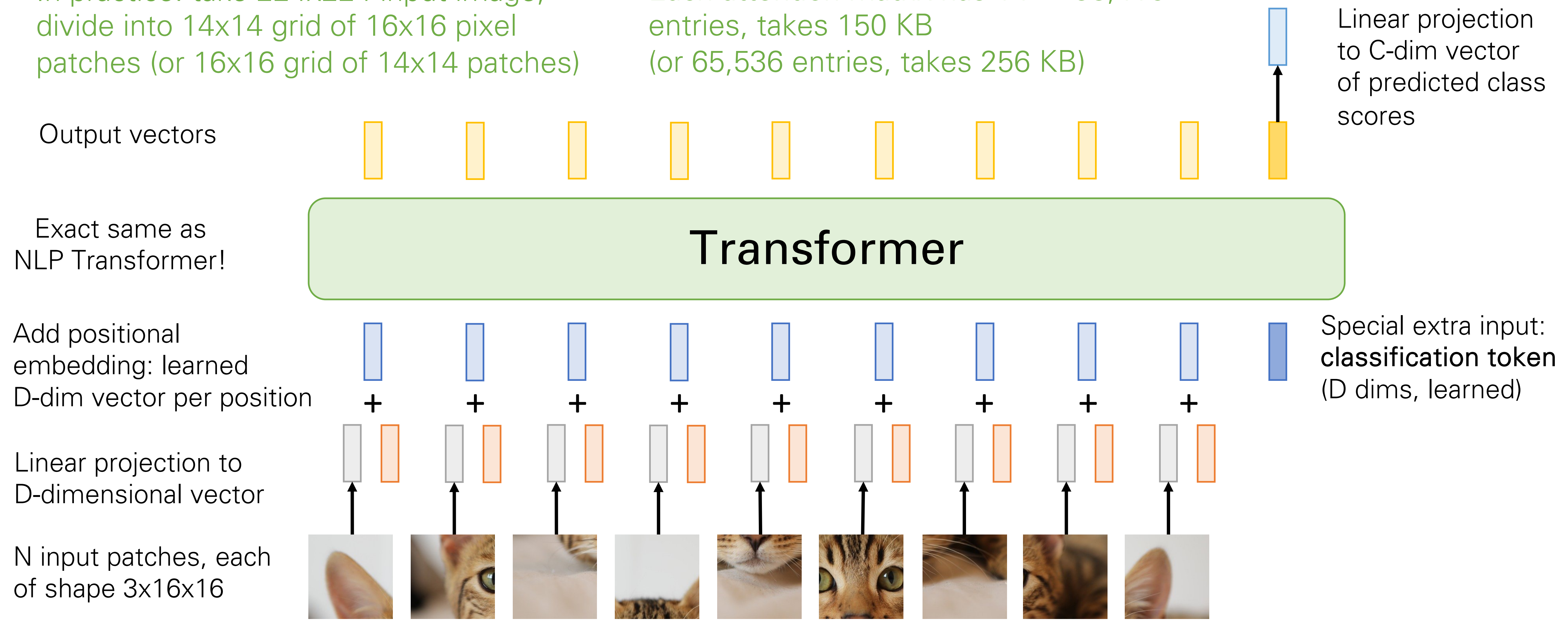


Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT)

In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

Each attention matrix has $14^4 = 38,416$ entries, takes 150 KB (or 65,536 entries, takes 256 KB)



Vision Transformer (ViT)

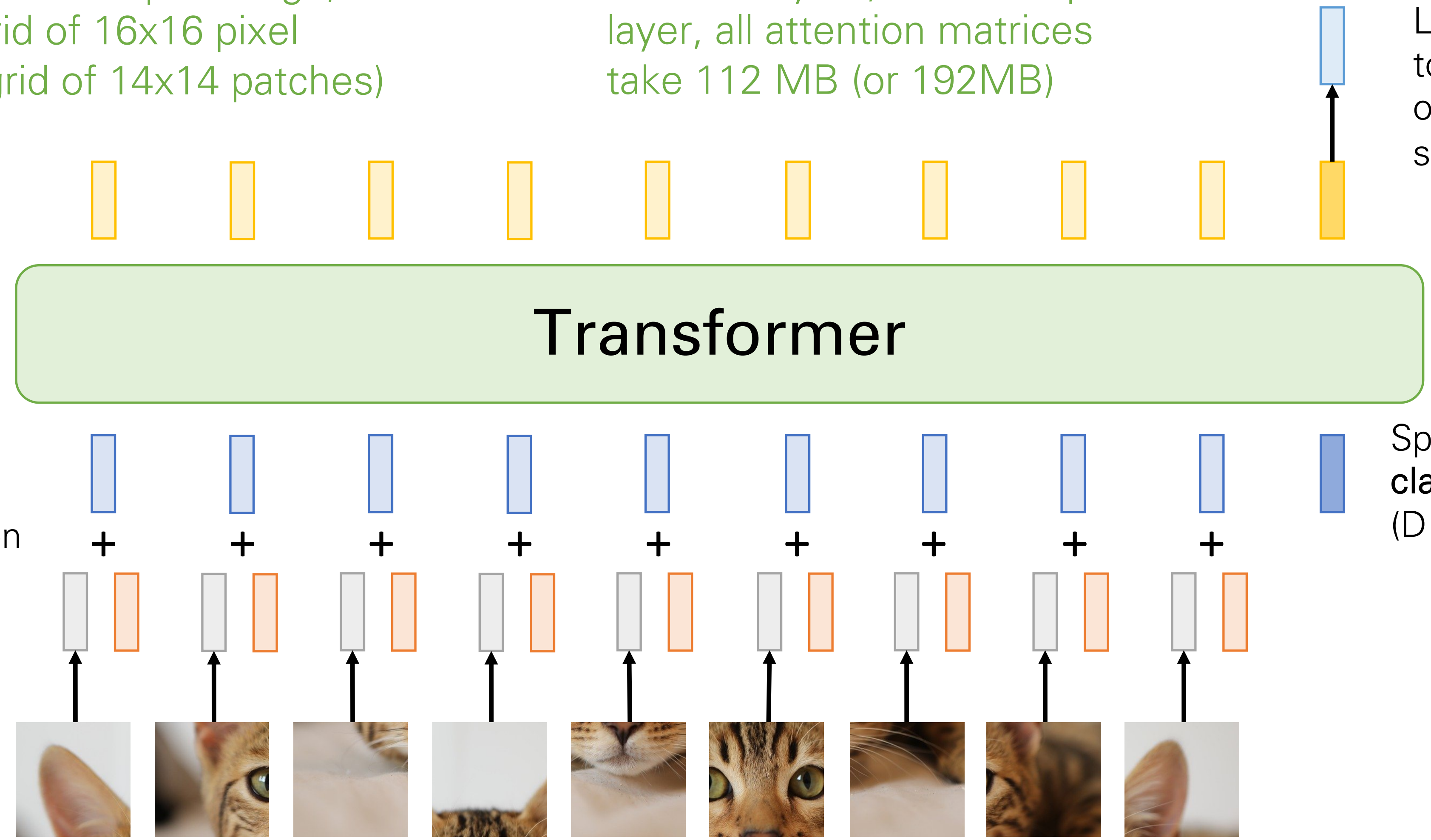
In practice: take 224x224 input image, divide into 14x14 grid of 16x16 pixel patches (or 16x16 grid of 14x14 patches)

With 48 layers, 16 heads per layer, all attention matrices take 112 MB (or 192MB)

Linear projection to C-dim vector of predicted class scores

Output vectors

Exact same as NLP Transformer!



Add positional embedding: learned D-dim vector per position

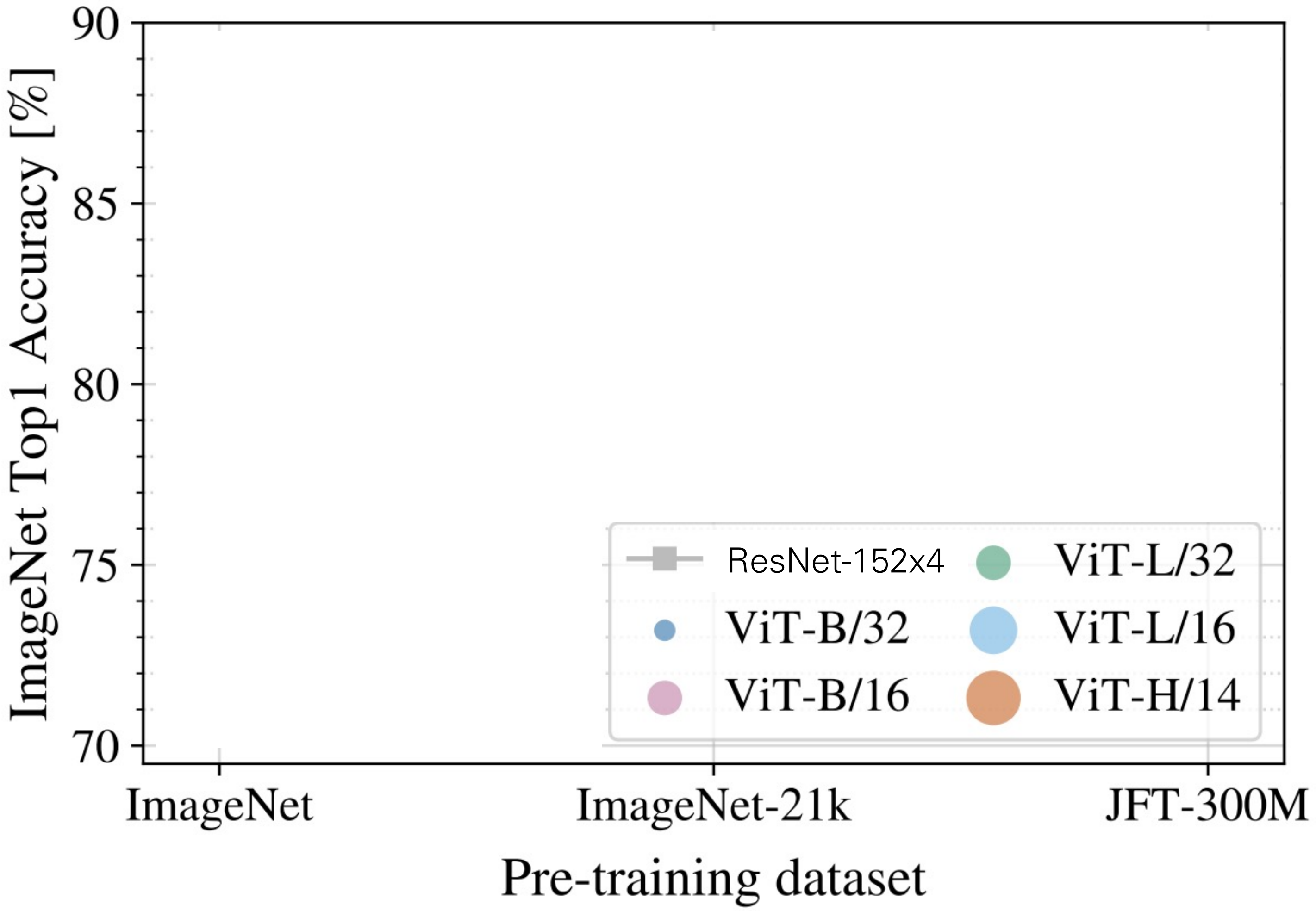
Special extra input: **classification token** (D dims, learned)

Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets



B = Base
L = Large
H = Huge

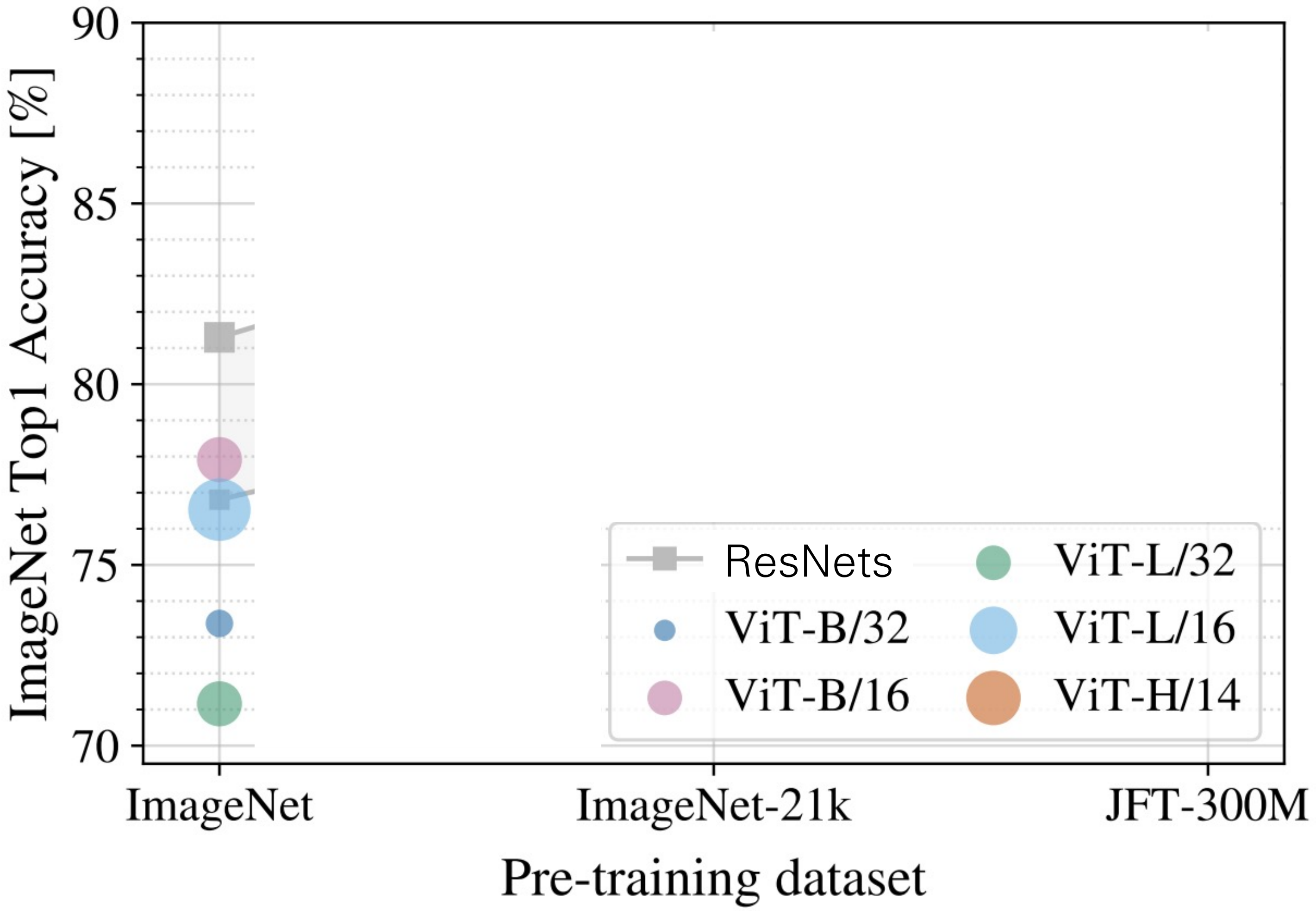
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

Recall: ImageNet dataset has 1k categories, 1.2M images

When trained on ImageNet, ViT models perform worse than ResNets



B = Base
L = Large
H = Huge

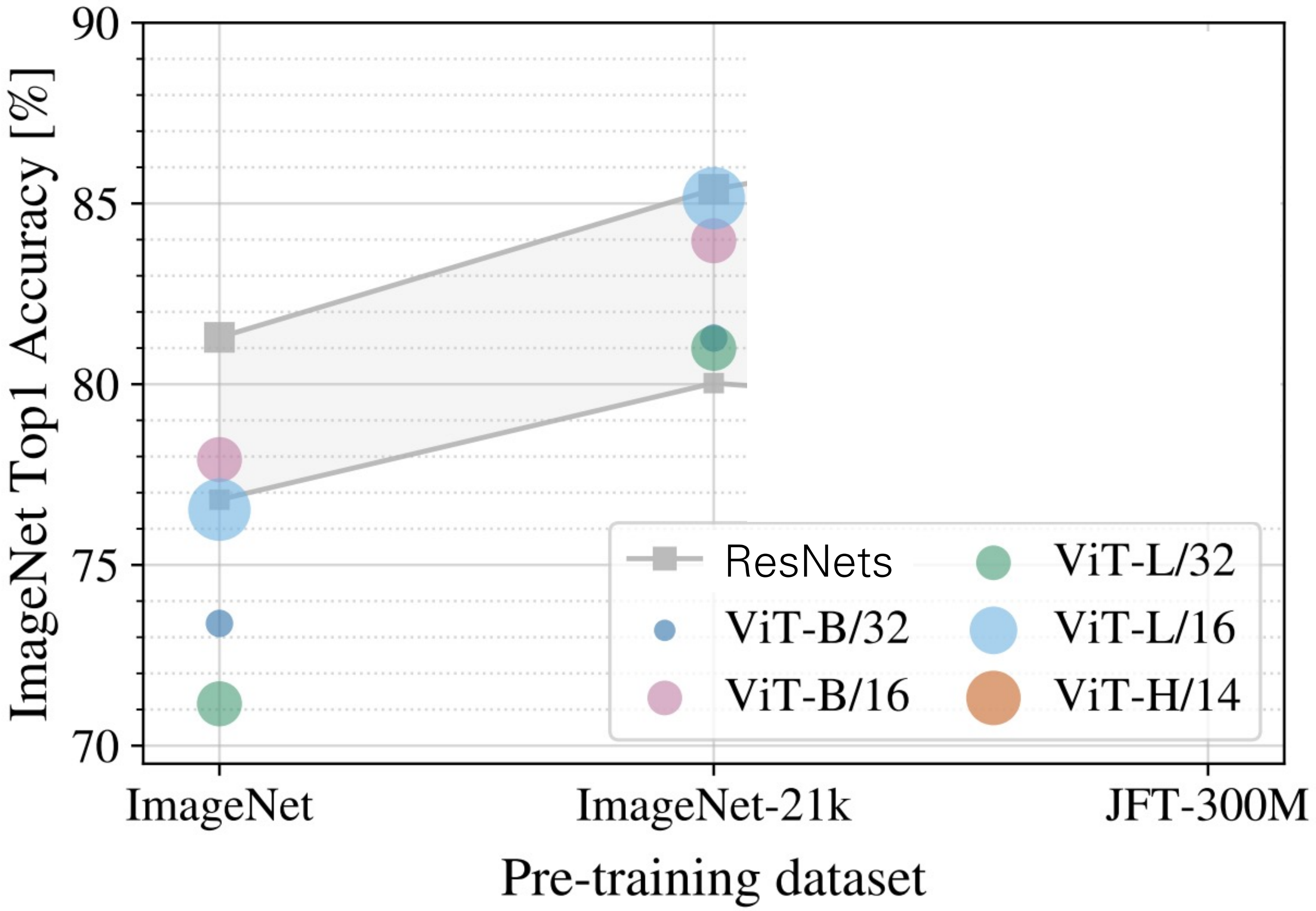
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

ImageNet-21k has 14M images with 21k categories

If you pretrain on ImageNet-21k and fine-tune on ImageNet, ViT does better: big ViTs match big ResNets



B = Base
L = Large
H = Huge

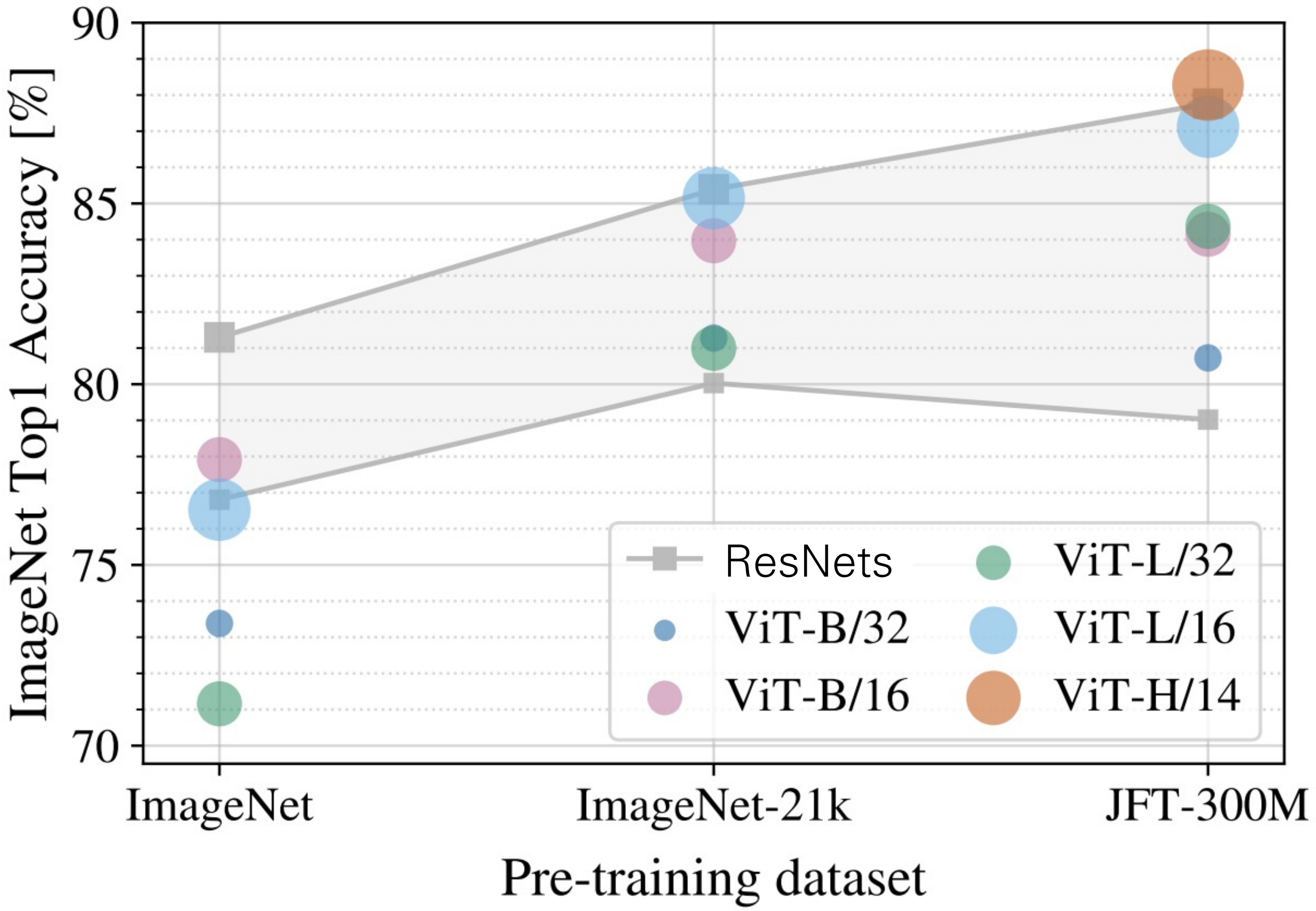
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



B = Base
L = Large
H = Huge

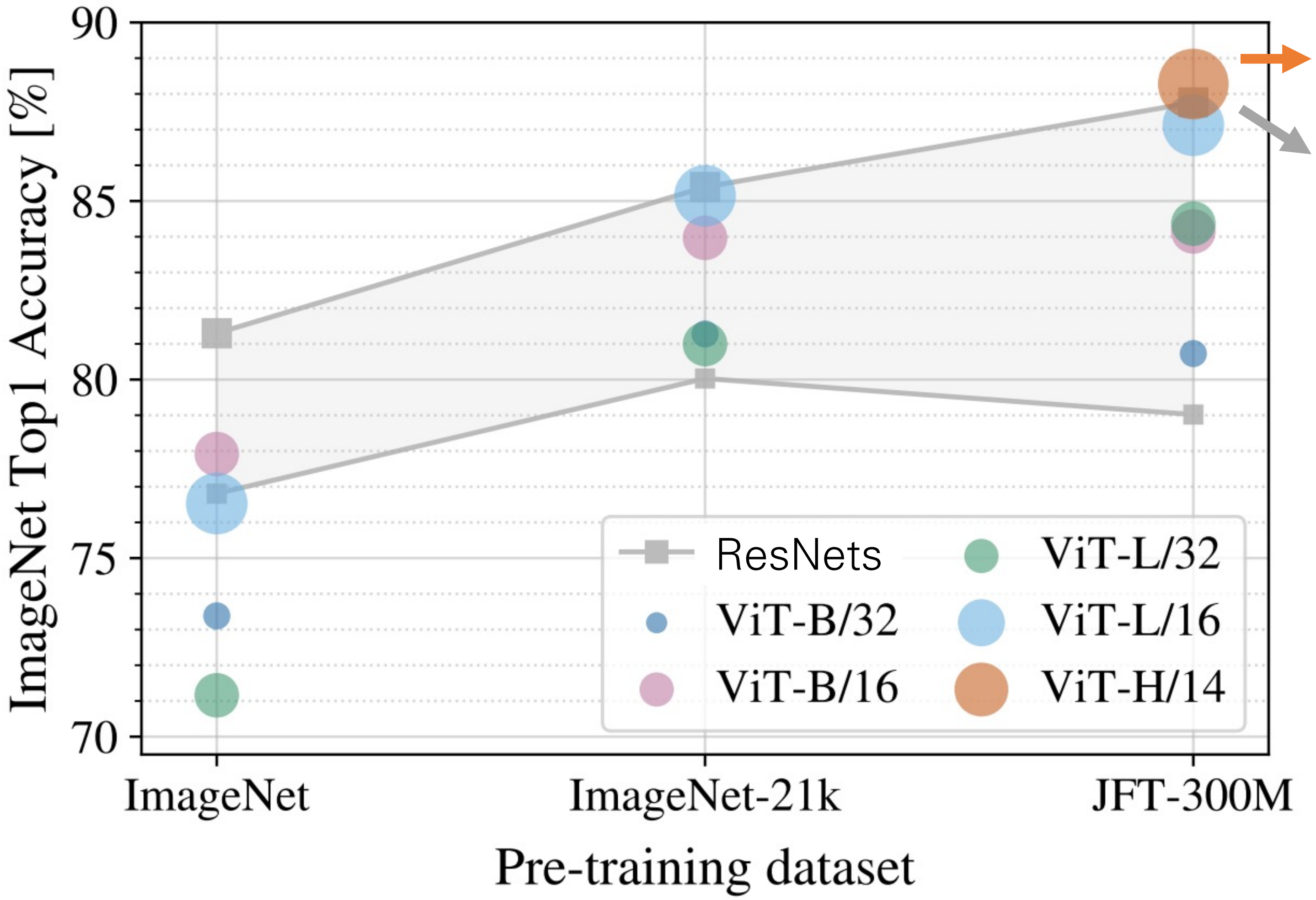
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



ViT: 2.5k TPU-v3 core days of training

ResNet: 9.9k TPU-v3 core days of training

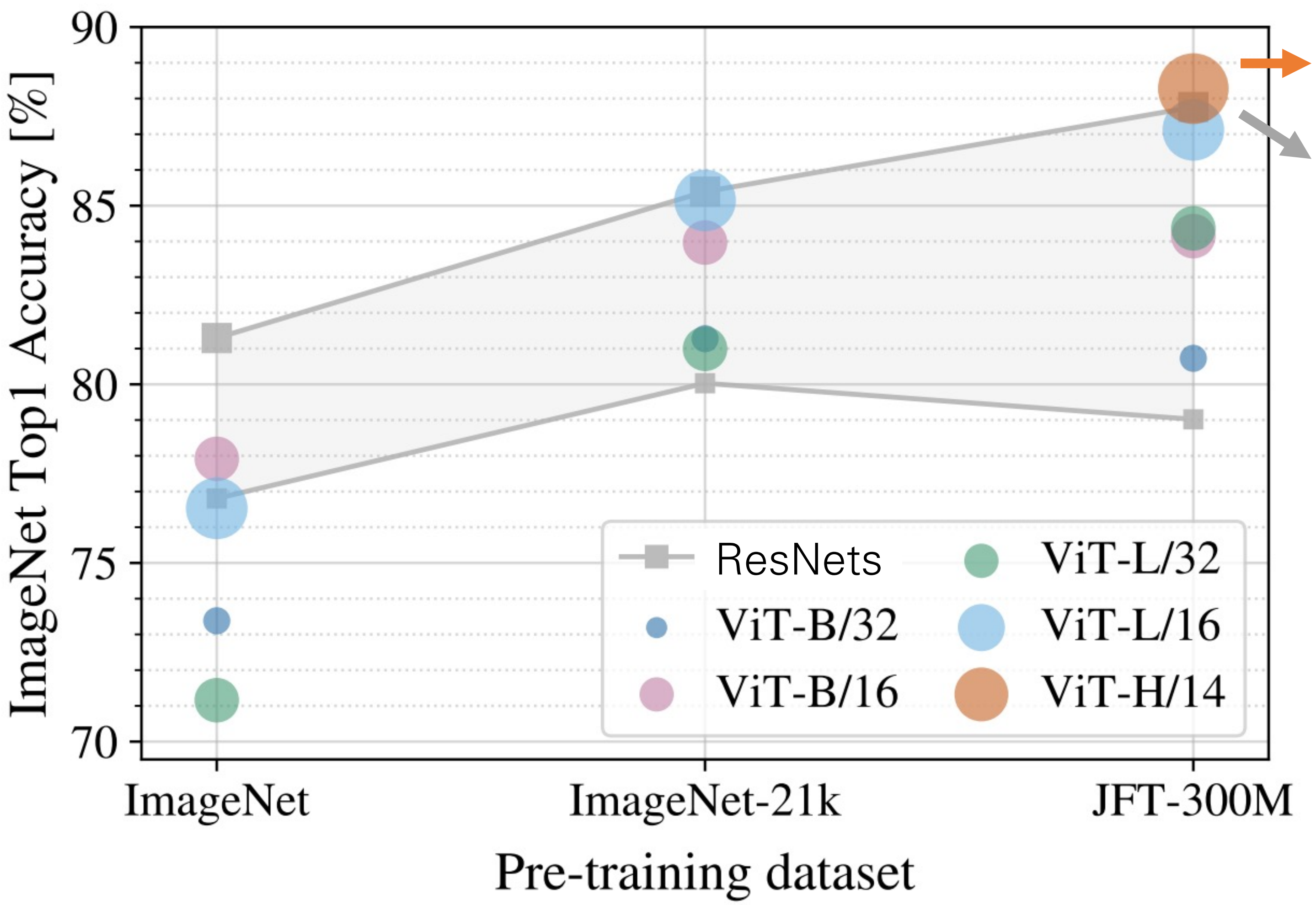
ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

Claim: ViT models have "less inductive bias" than ResNets, so need more pretraining data to learn good features

(Not sure I buy this explanation: "inductive bias" is not a well-defined concept we can measure!)



ViT: 2.5k TPU-v3 core days of training

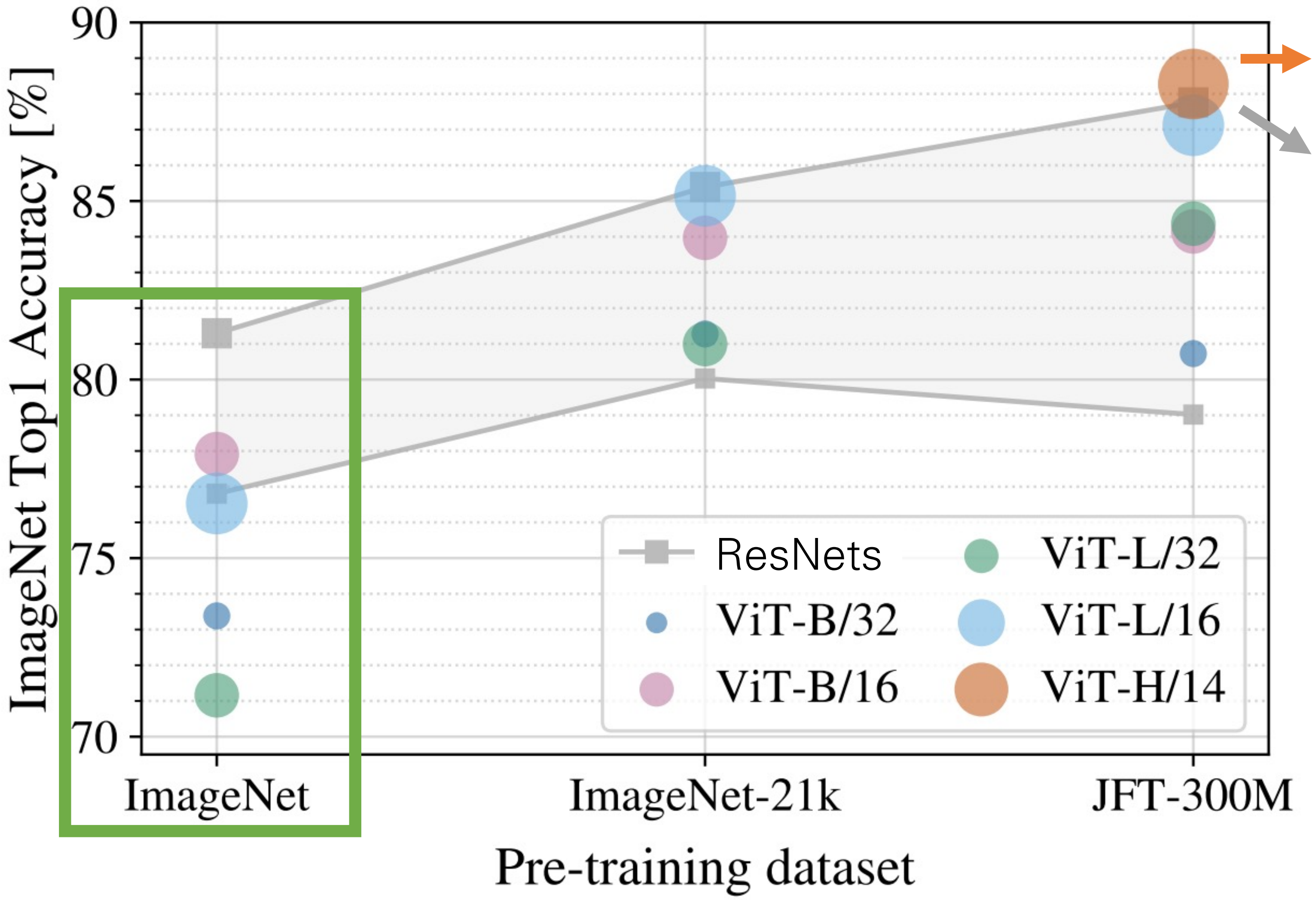
ResNet: 9.9k TPU-v3 core days of training

ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

How can we improve the performance of ViT models on ImageNet?



ViT: 2.5k TPU-v3 core days of training

ResNet: 9.9k TPU-v3 core days of training

ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

ViT vs CNN

In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

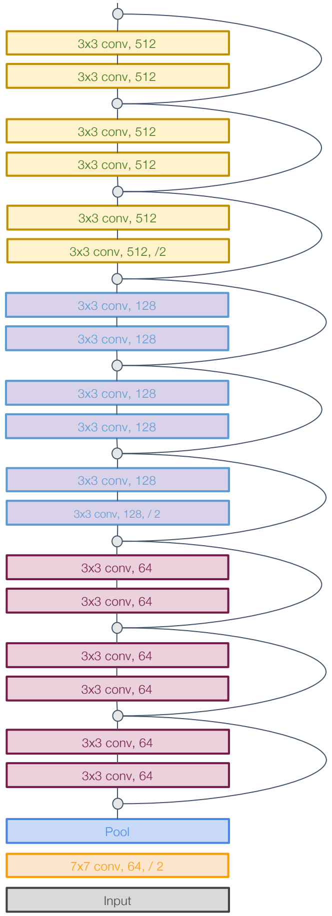
Useful since objects in images can occur at various scales

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224



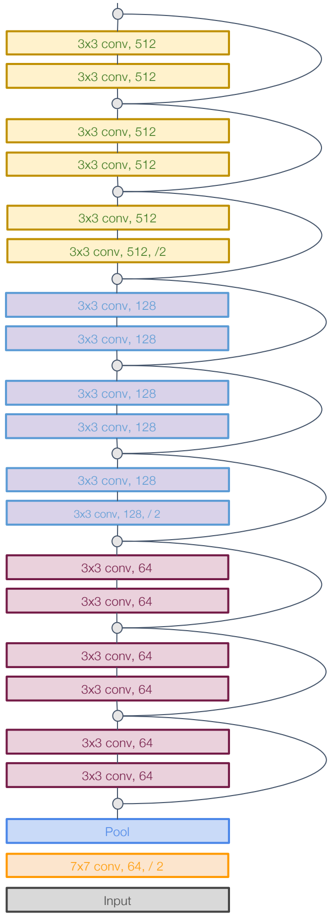
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

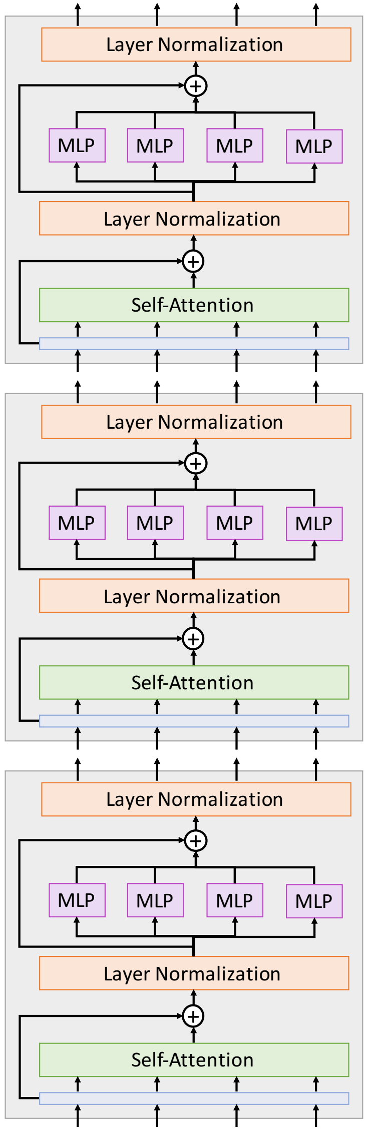
Input:
3 x 224 x 224



In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)



3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

1st block:
768 x 14 x 14

Input:
3 x 224 x 224

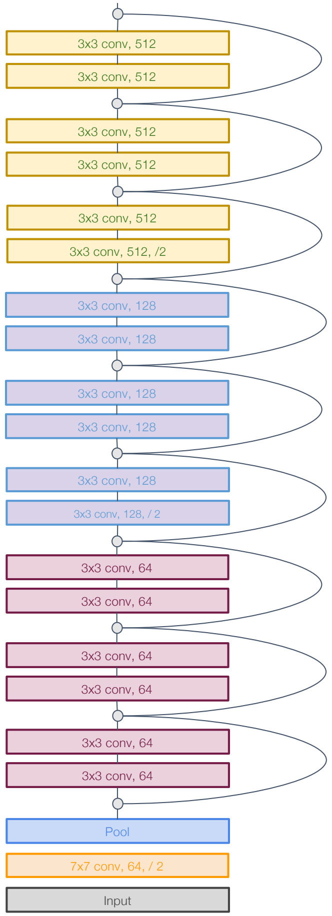
ViT vs CNN

Stage 3:
256 x 14 x 14

Stage 2:
128 x 28 x 28

Stage 1:
64 x 56 x 56

Input:
3 x 224 x 224

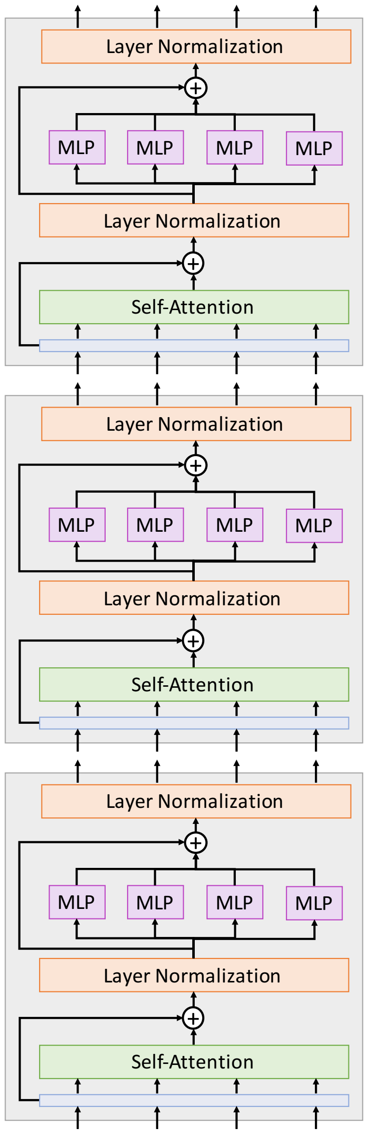


In most CNNs (including ResNets), **decrease** resolution and **increase** channels as you go deeper in the network (Hierarchical architecture)

Useful since objects in images can occur at various scales

In a ViT, all blocks have same resolution and number of channels (Isotropic architecture)

Can we build a **hierarchical** ViT model?



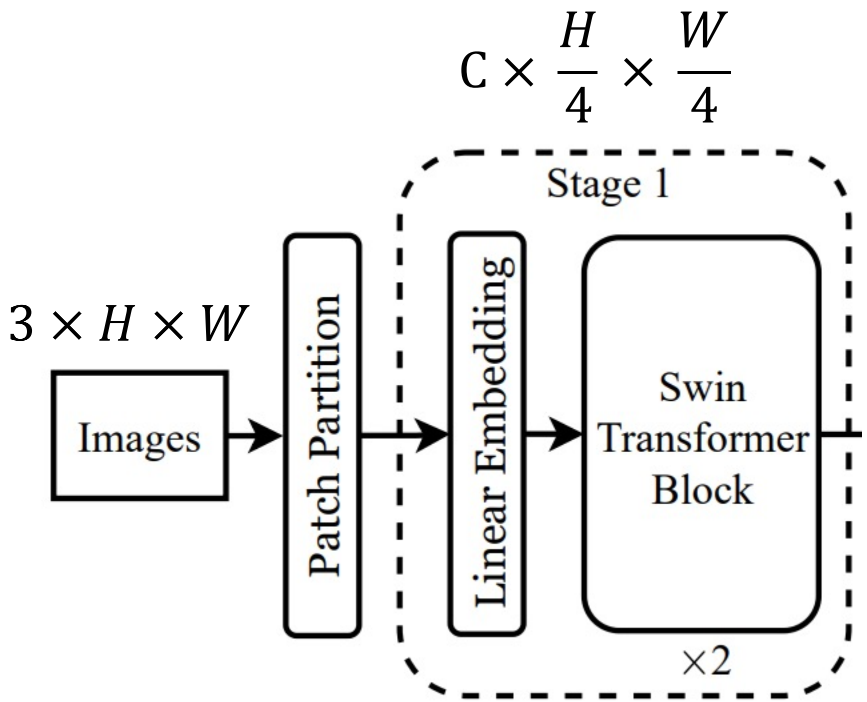
3rd block:
768 x 14 x 14

2nd block:
768 x 14 x 14

1st block:
768 x 14 x 14

Input:
3 x 224 x 224

Hierarchical ViT: Swin Transformer



Divide image into 4x4
patches and project to
C dimensions

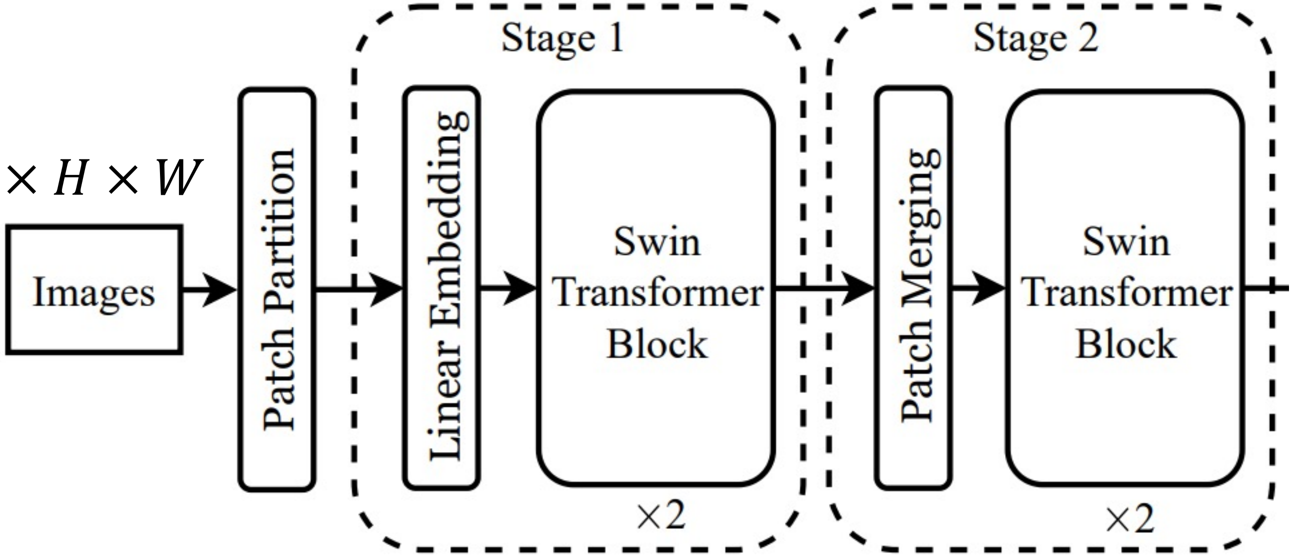
Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

$C \times \frac{H}{4} \times \frac{W}{4}$

$2C \times \frac{H}{8} \times \frac{W}{8}$

$3 \times H \times W$

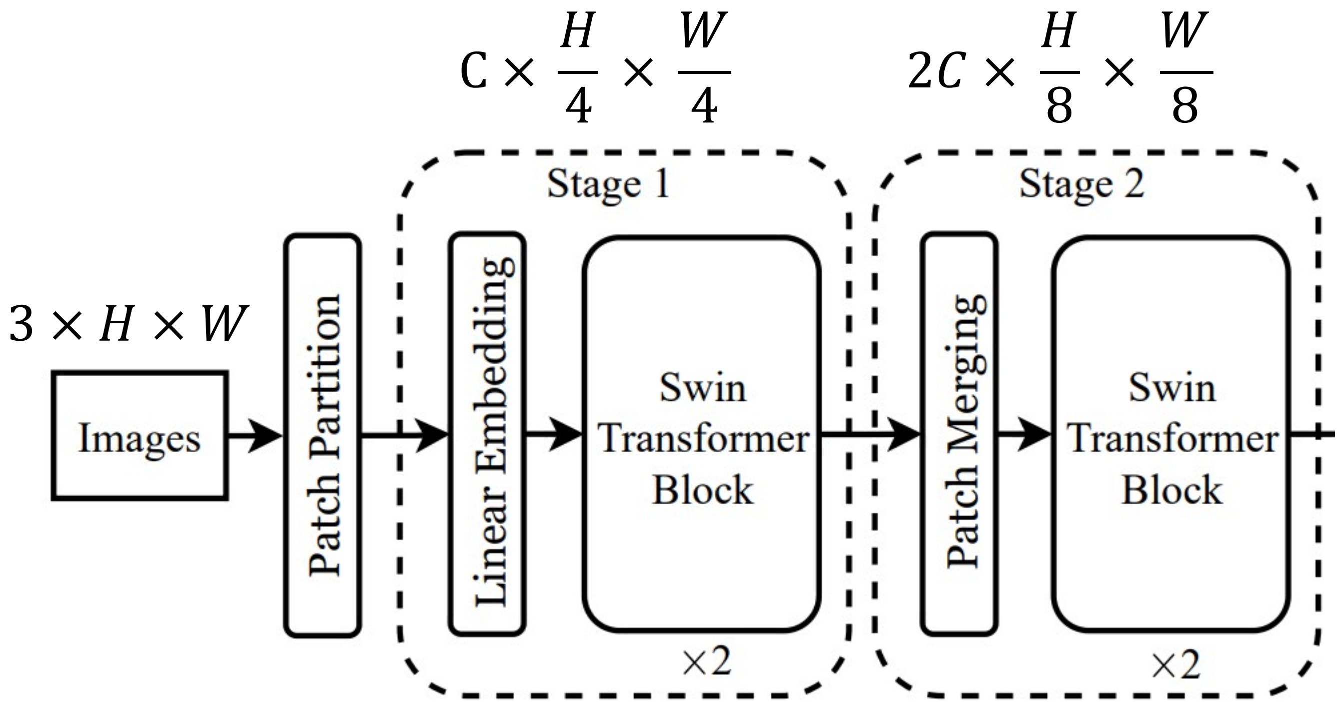


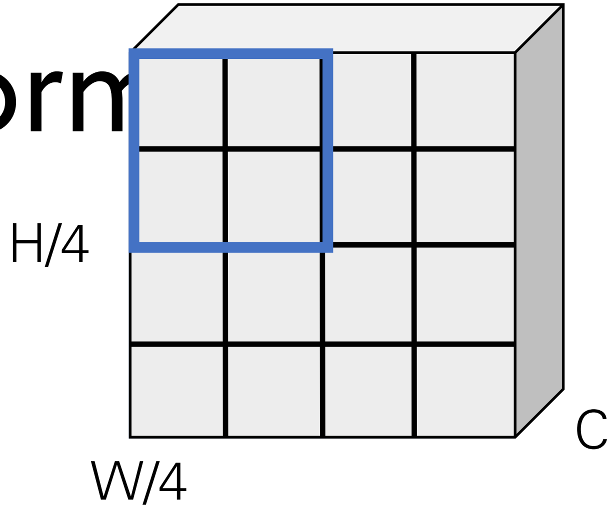
Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



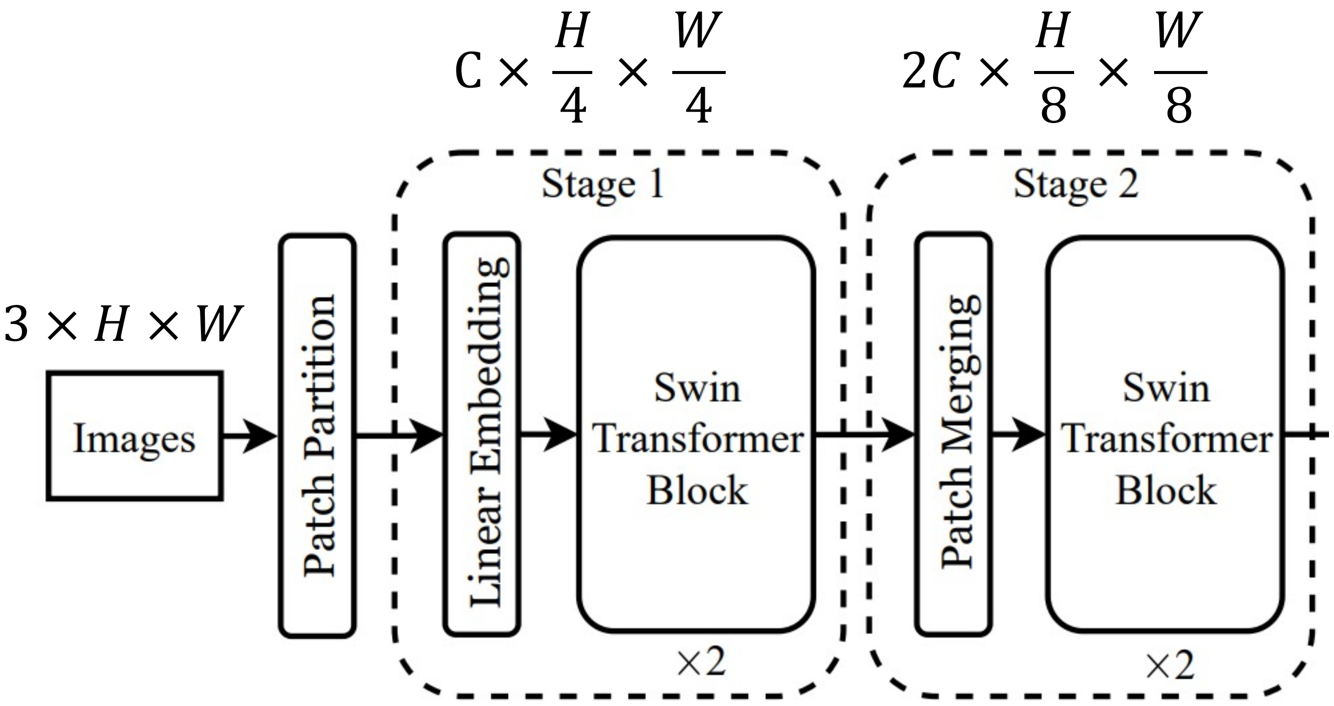


Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

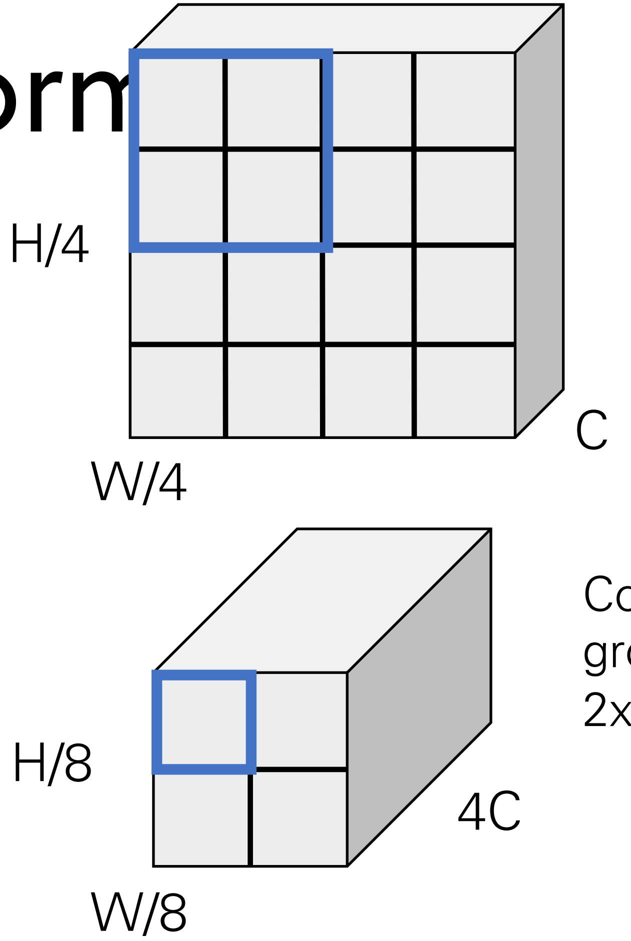
Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

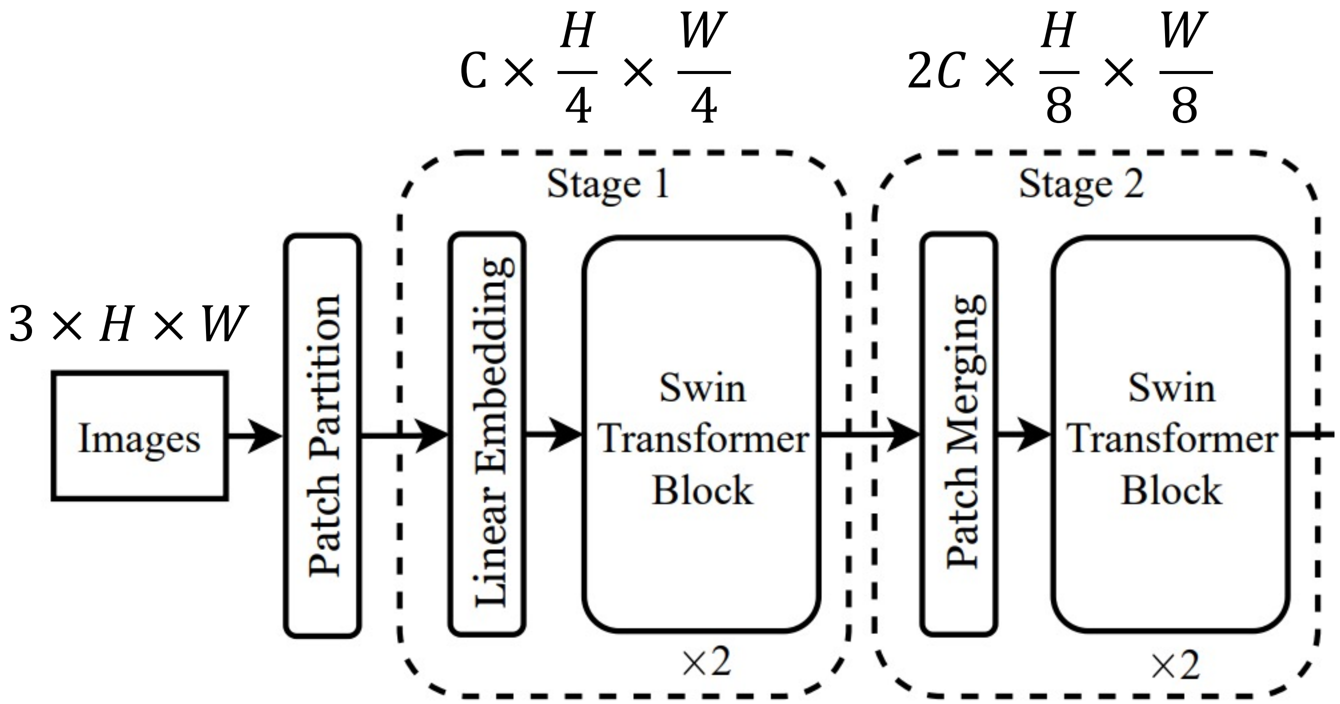
Merge 2x2 neighborhoods; now patches are (effectively) 8x8



Concatenate groups of 2x2 features

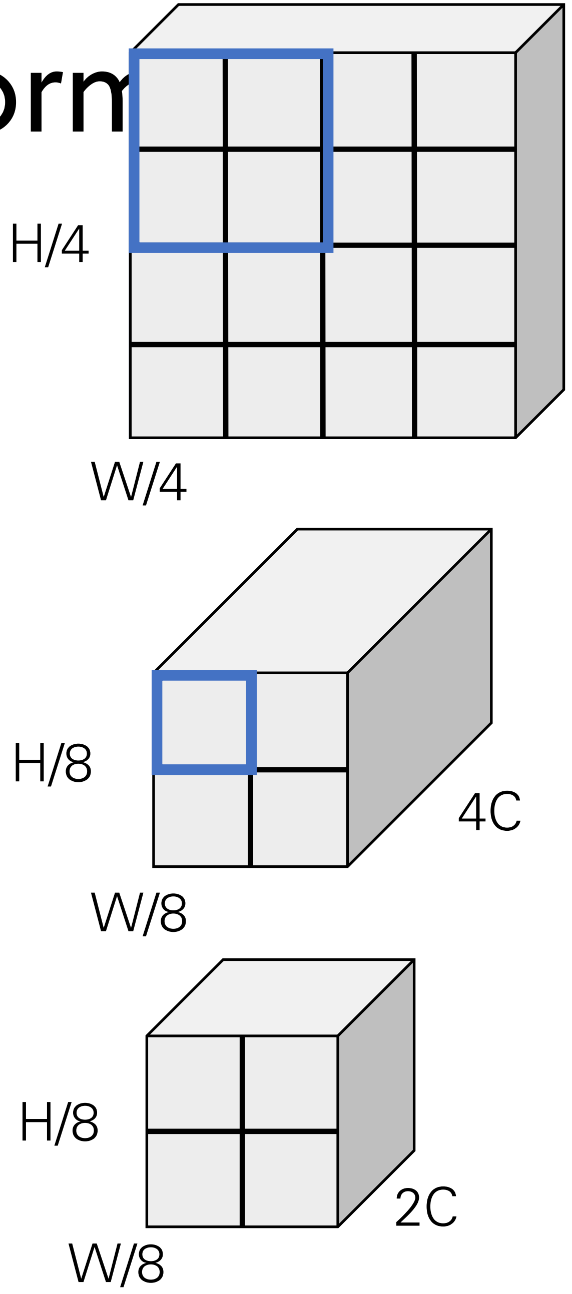
Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transform



Divide image into 4x4 patches and project to C dimensions

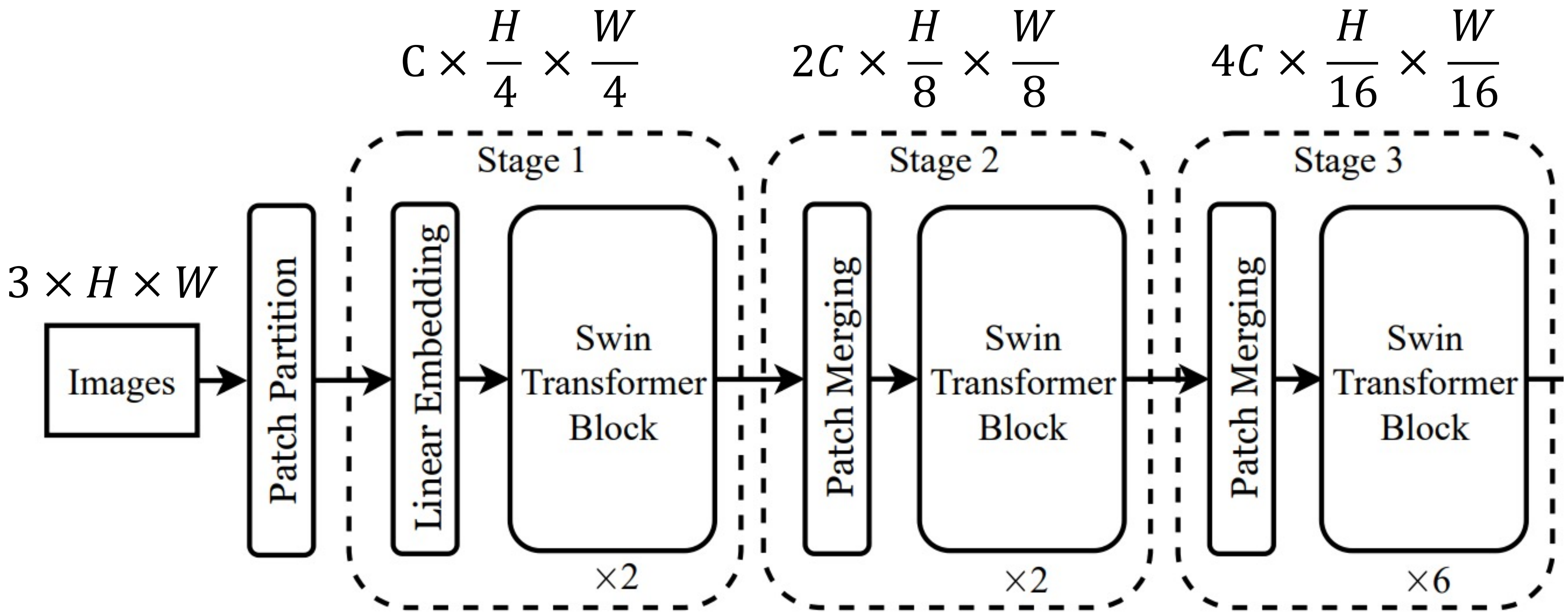
Merge 2x2 neighborhoods; now patches are (effectively) 8x8



Concatenate groups of 2x2 features

Linear projection from 4C to 2C channels (1x1 conv)

Hierarchical ViT: Swin Transformer

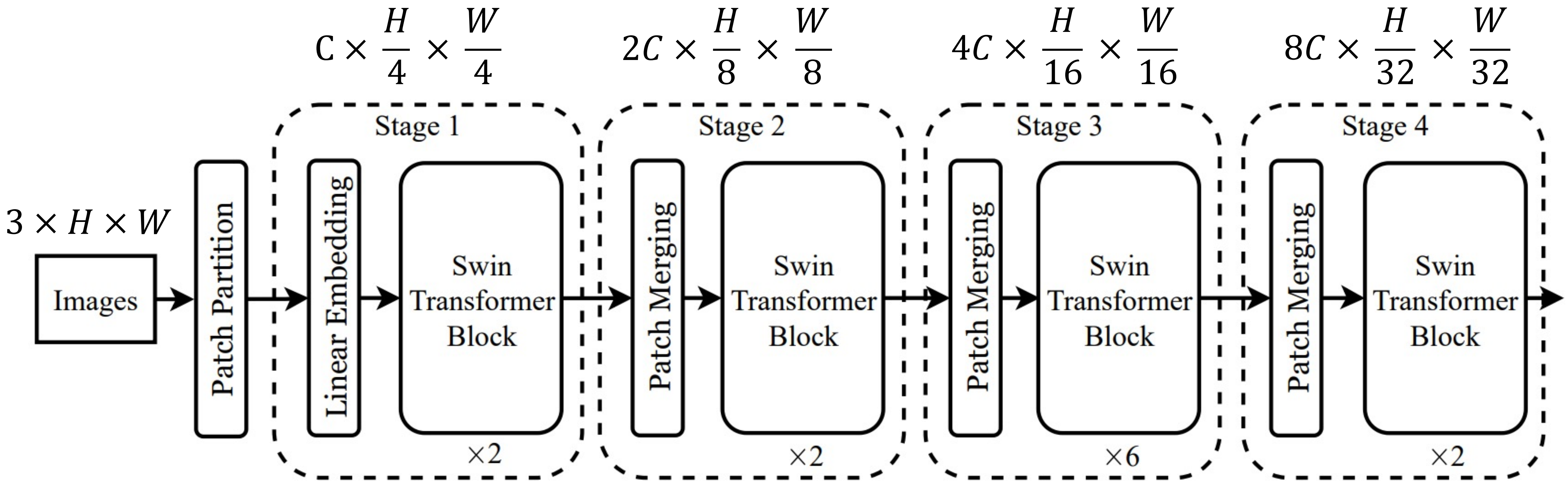


Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Hierarchical ViT: Swin Transformer



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

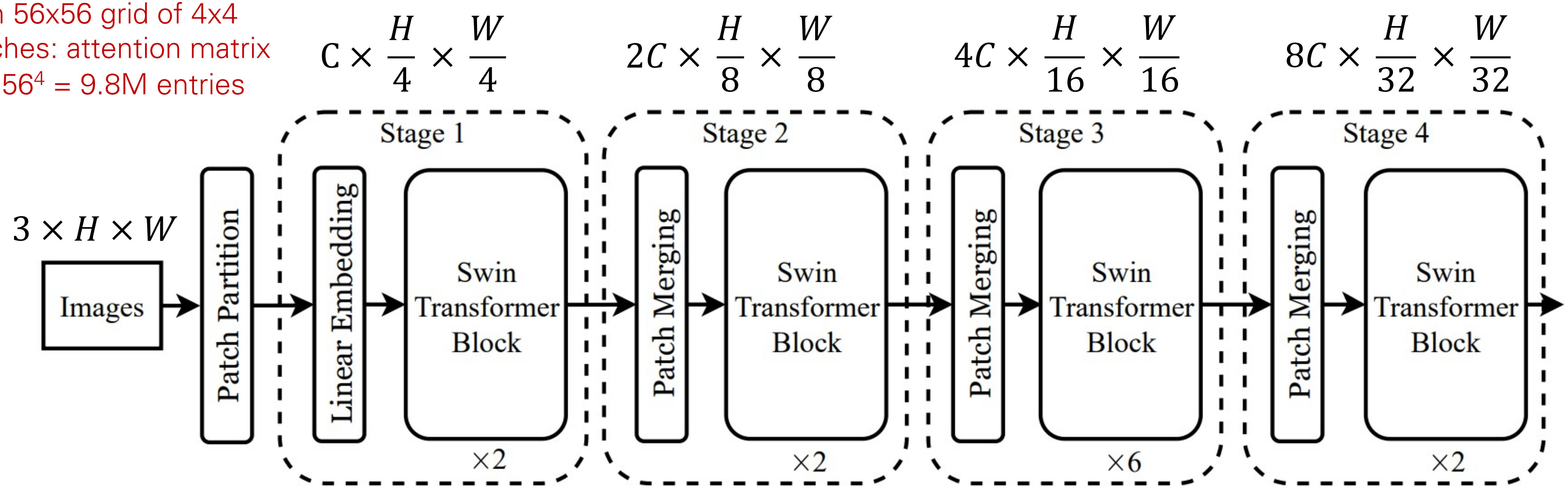
Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches: attention matrix has 56^4 = 9.8M entries



Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

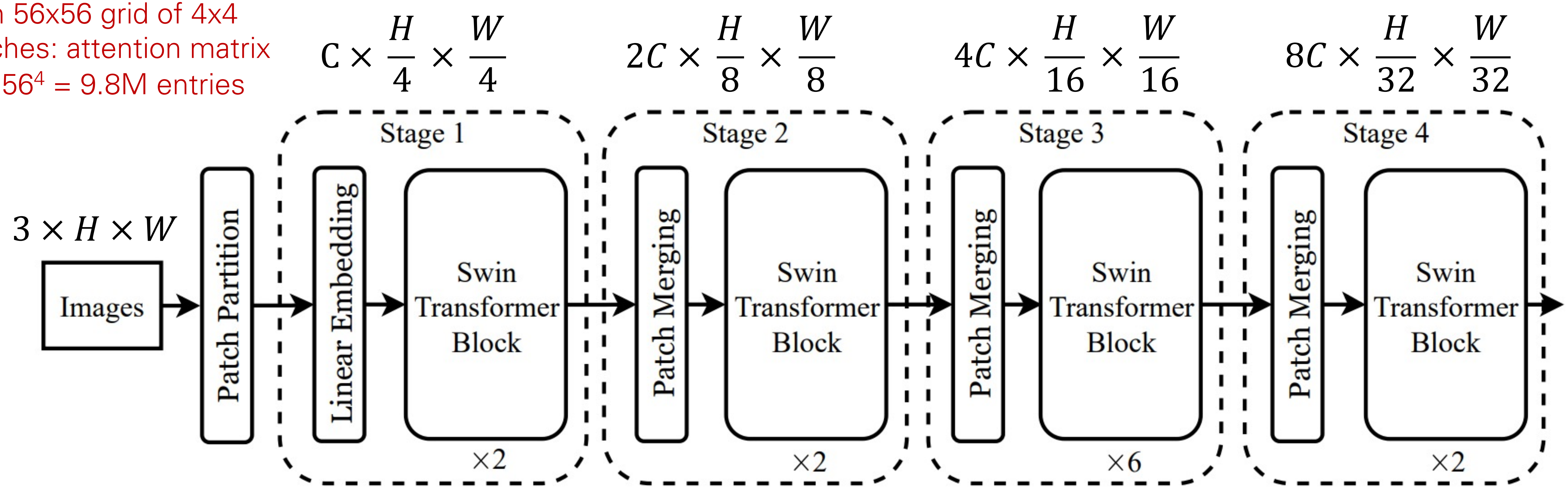
Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Hierarchical ViT: Swin Transformer

Problem: 224x224 image with 56x56 grid of 4x4 patches: attention matrix has 56^4 = 9.8M entries



Solution: don't use full attention, instead use attention over patches

Divide image into 4x4 patches and project to C dimensions

Merge 2x2 neighborhoods; now patches are (effectively) 8x8

Merge 2x2 neighborhoods; now patches are (effectively) 16x16

Merge 2x2 neighborhoods; now patches are (effectively) 32x32

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of tokens, each attention matrix is H^2W^2 – **quadratic** in image size

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of tokens, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each token to attend to all other tokens, instead divide into windows of M x M tokens (here M=4); only compute attention within each window

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

With H x W grid of tokens, each attention matrix is H^2W^2 – **quadratic** in image size

Rather than allowing each token to attend to all other tokens, instead divide into windows of M x M tokens (here M=4); only compute attention within each window

Total size of all attention matrices is now:
$M^4(H/M)(W/M) = M^2HW$

Linear in image size for fixed M!
Swin uses M=7 throughout the network

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Window Attention

Problem: tokens only interact with other tokens within the same window; no communication across windows

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and <u>shifted windows</u> in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Ugly detail: Non-square windows at edges and corners

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and <u>shifted windows</u> in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes absolute position of each token in the image

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and <u>shifted windows</u> in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes absolute position of each token in the image

Swin does not use positional embeddings, instead encodes relative position between patches when computing attention:

Standard Attention:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}}\right)V$$

$Q, K, V : M^2 \times D$ (Query, Key, Value)

Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Shifted Window Attention

Solution: Alternate between normal windows and <u>shifted windows</u> in successive Transformer blocks

Block L: Normal windows

Block L+1: Shifted Windows

Detail: Relative Positional Bias

ViT adds positional embedding to input tokens, encodes absolute position of each token in the image

Swin does not use positional embeddings, instead encodes relative position between patches when computing attention:

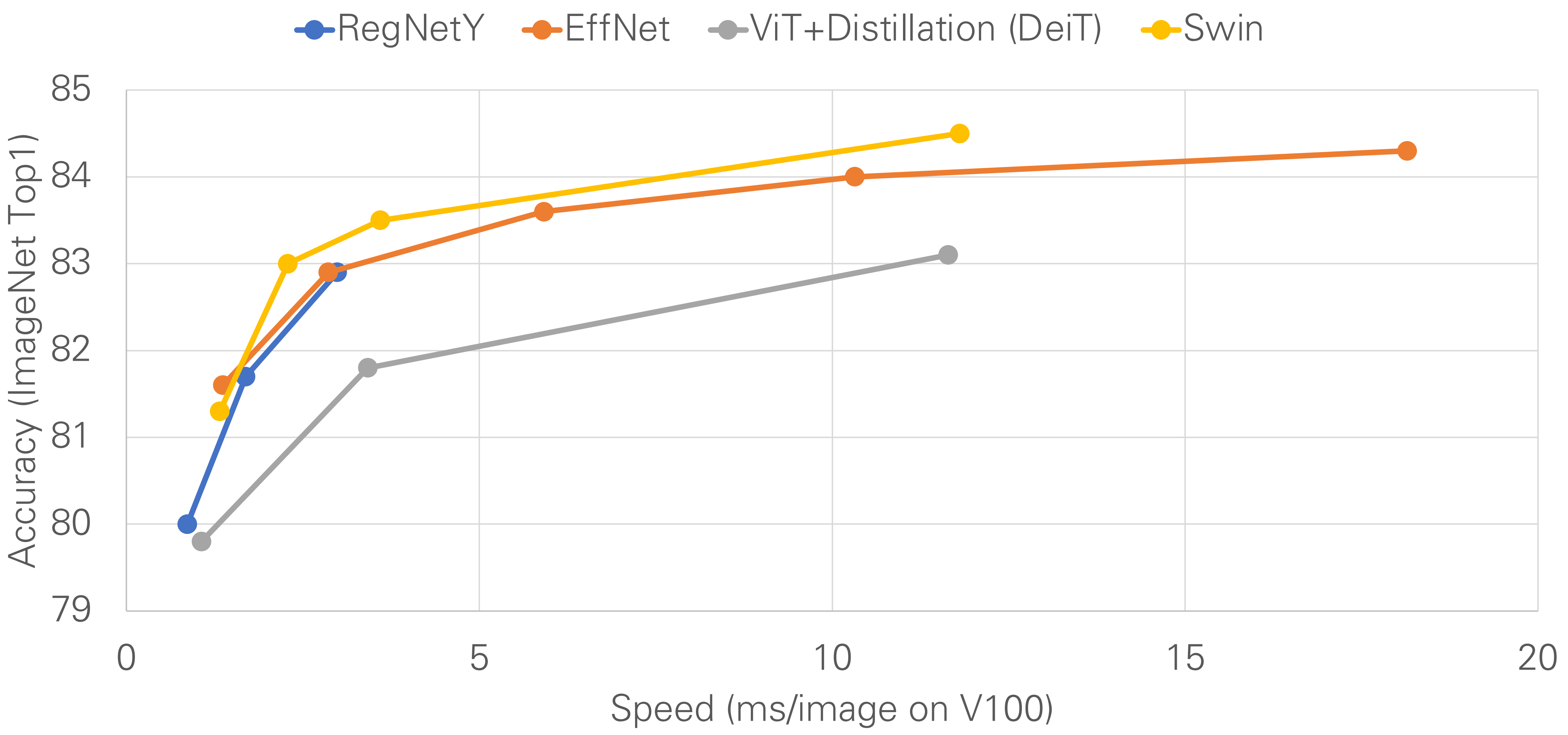
Attention with relative bias:

$$A = Softmax\left(\frac{QK^T}{\sqrt{D}} + B\right)V$$

$Q, K, V: M^2 \times D$ (Query, Key, Value)
$B: M^2 \times M^2$ (learned biases)

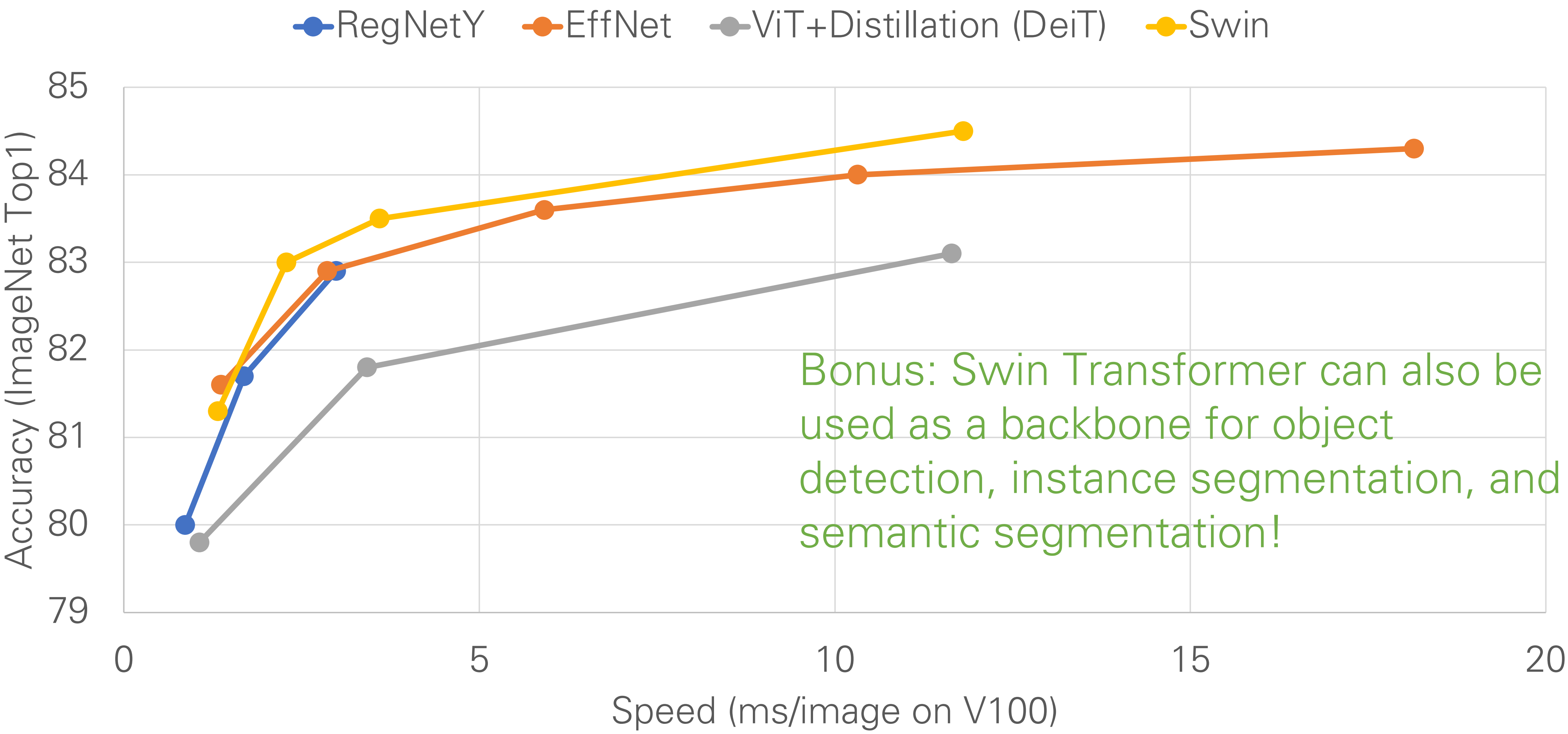
Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Swin Transformer: Speed vs Accuracy



Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

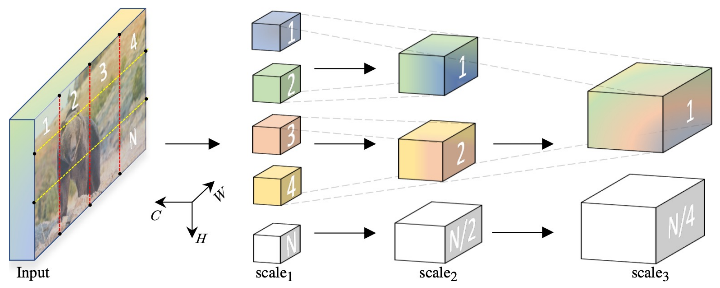
Swin Transformer: Speed vs Accuracy



Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

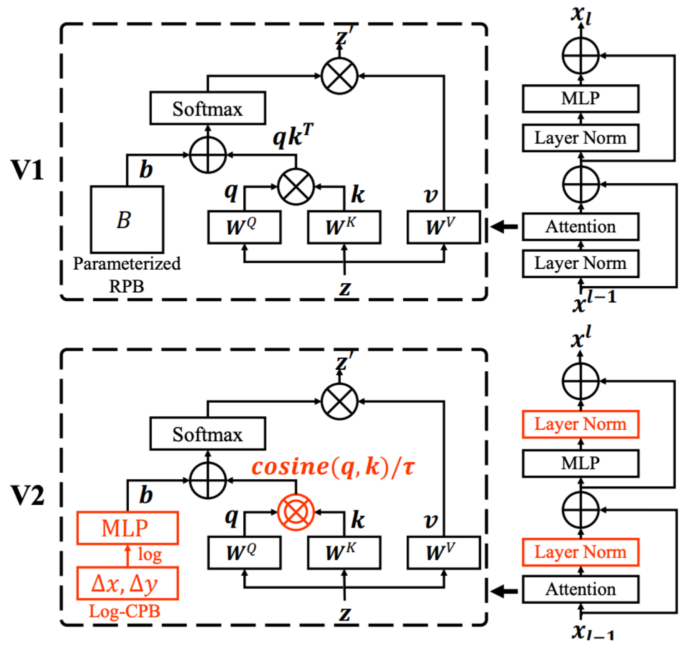
Other Hierarchical Vision Transformers

MViT



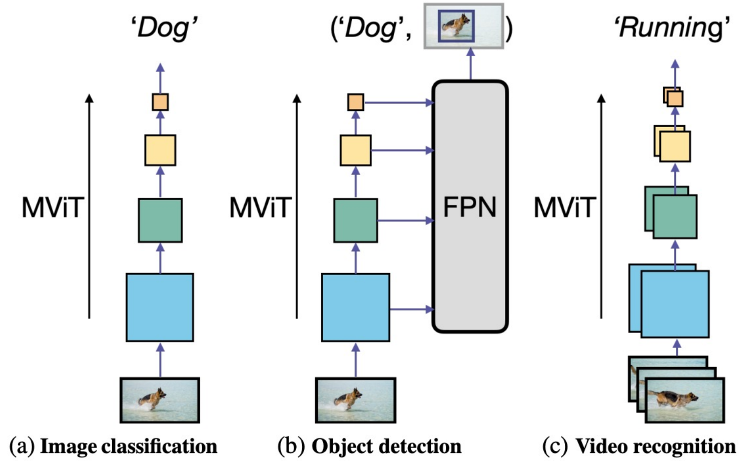
Fan et al., "Multiscale Vision Transformers", ICCV 2021

Swin-V2



Liu et al, "Swin Transformer V2: Scaling up Capacity and Resolution", CVPR 2022

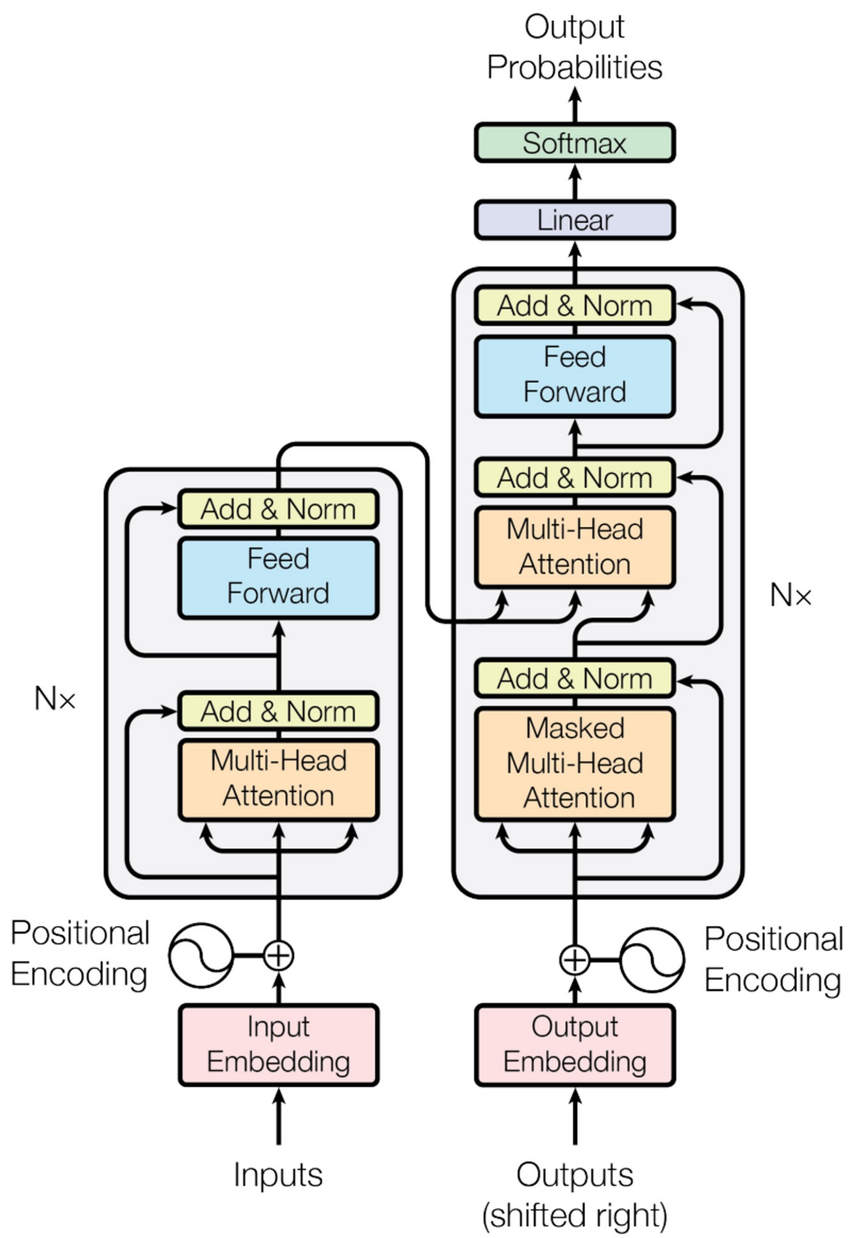
Improved MViT



Li et al, "Improved Multiscale Vision Transformers for Classification and Detection", arXiv 2021

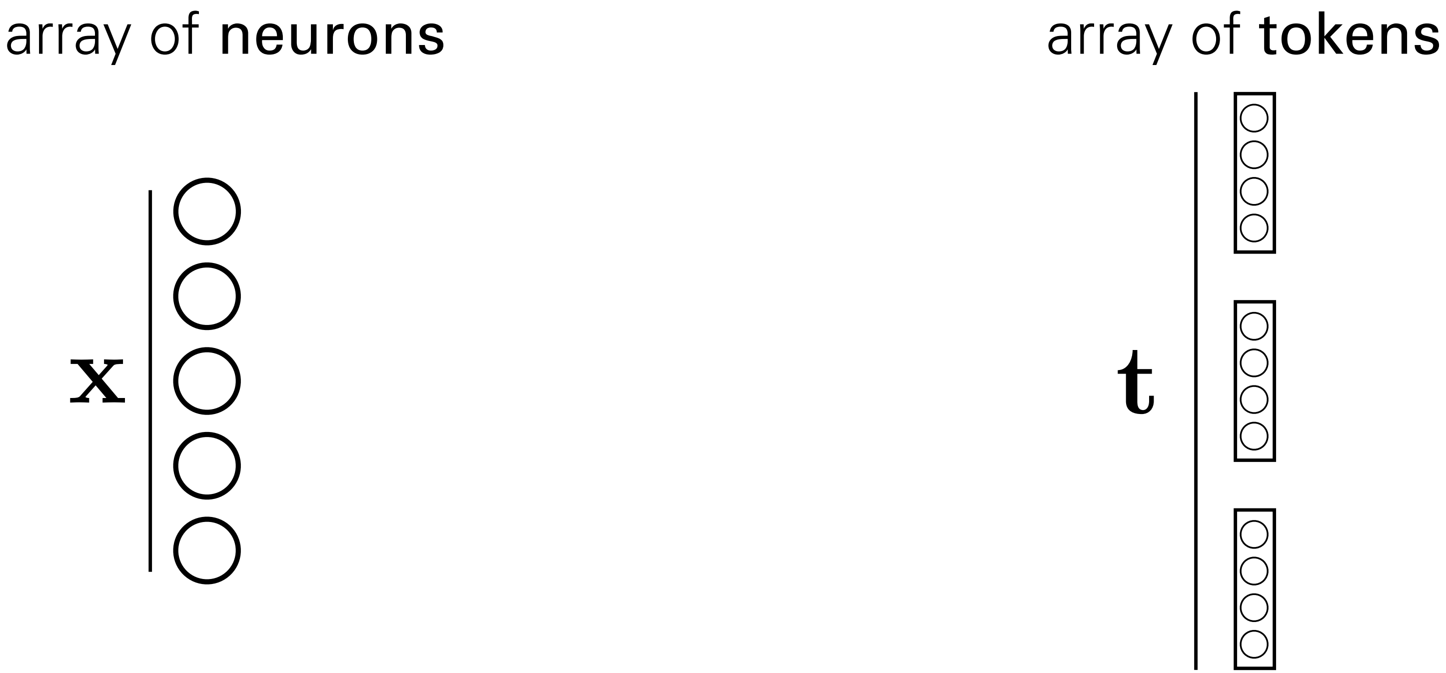
Recap of Transformers

- Three key ideas
  - Tokens
  - Attention
  - Positional encoding



Tokens: A new data structure

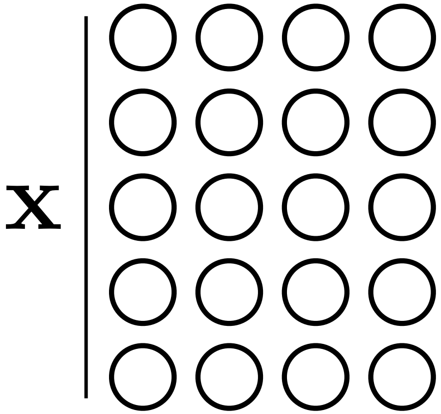
- A **token** is just transformer lingo for a vector of neurons (note: GNNs also operate over tokens, but over there we called them "node attributes" or node "feature descriptors")
- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens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



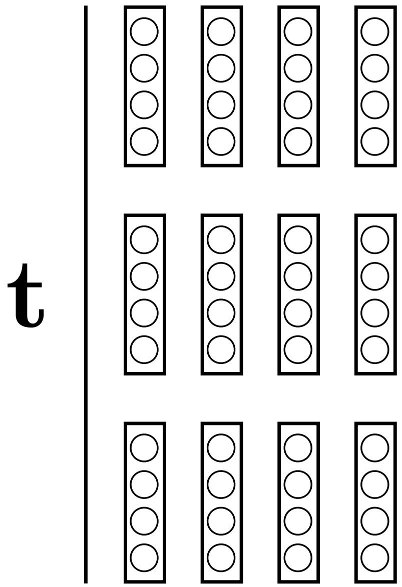
Tokens: A new data structure

- 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

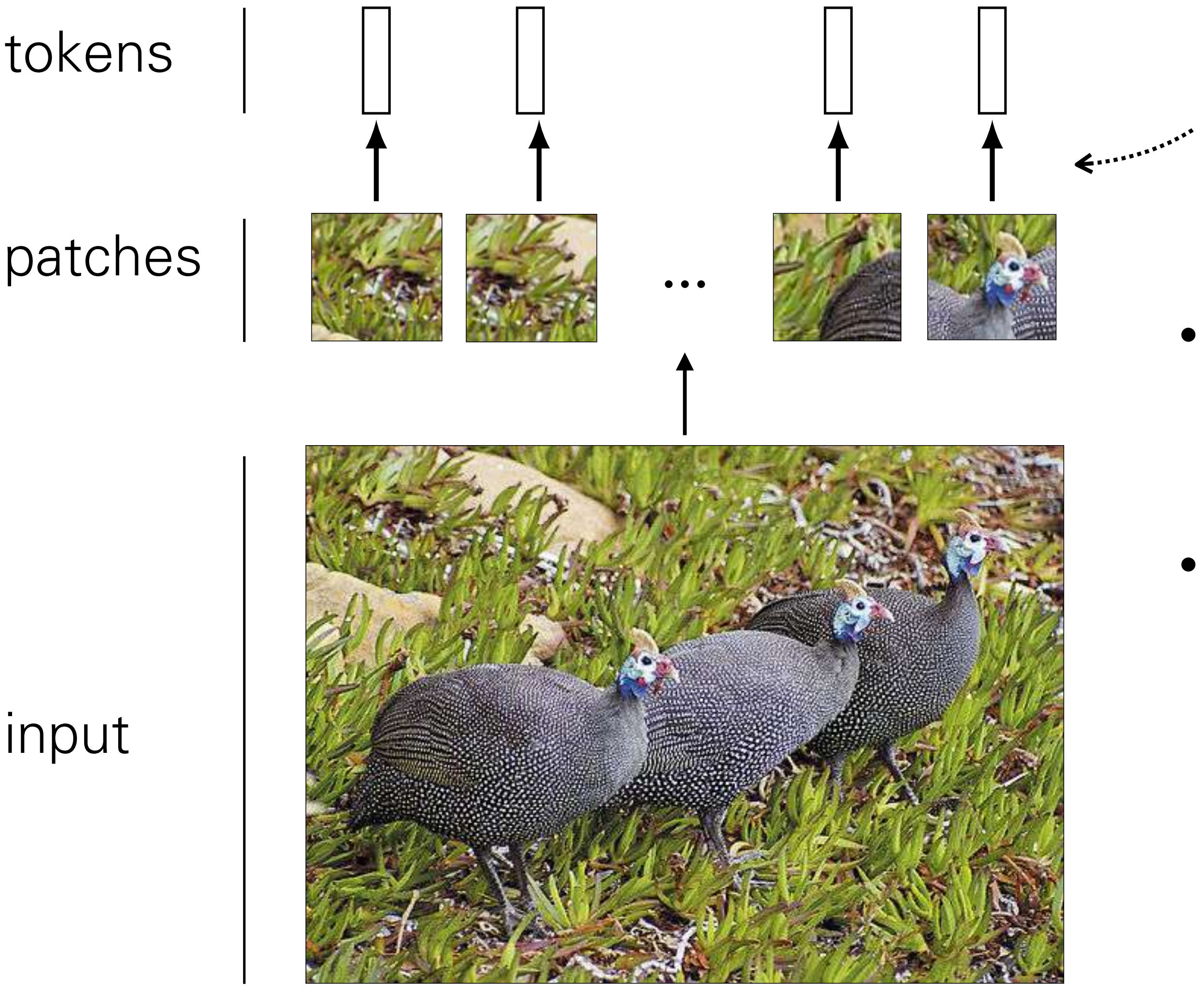
set of **neurons**



set of **tokens**



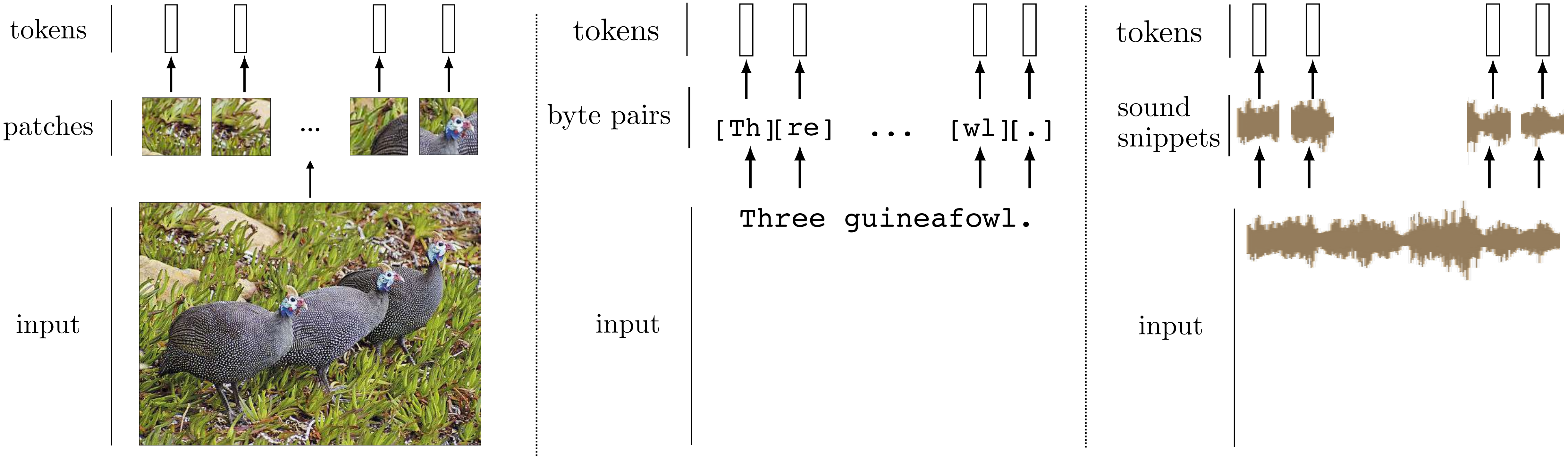
Tokenizing the input data



- 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

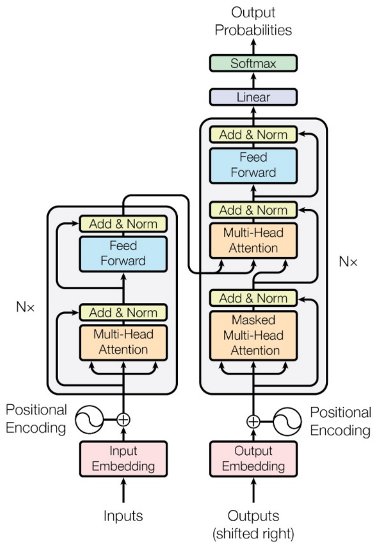
Tokenizing the input data

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

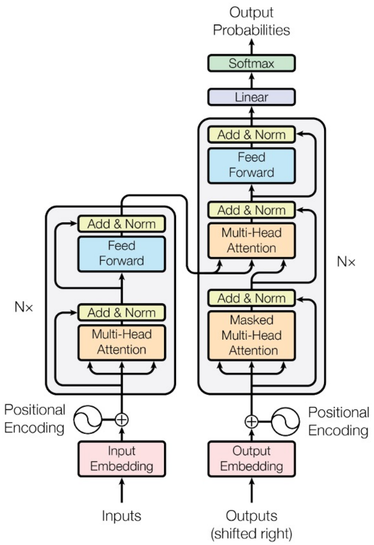


Transformers

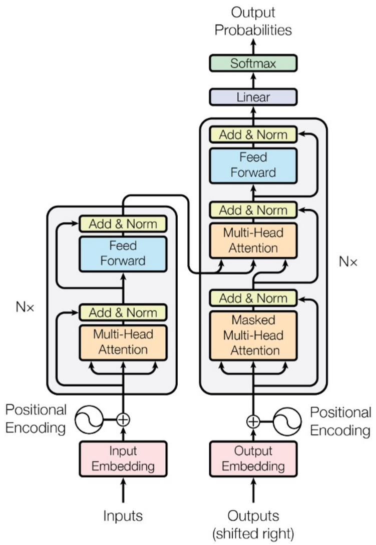
- Transformers takeover the communities since their introduction.



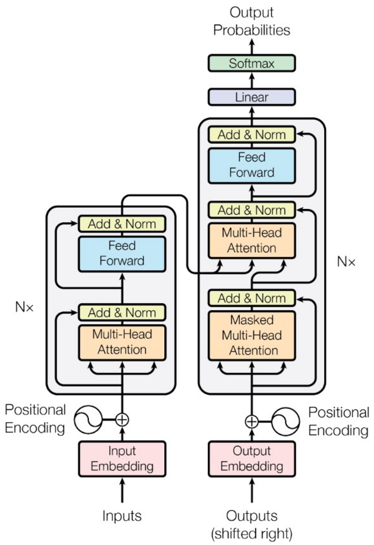
Computer Vision



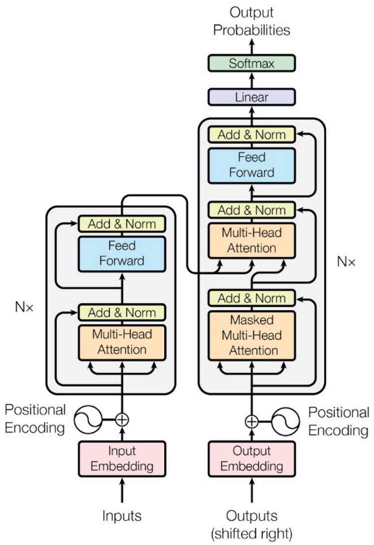
Natural Lang. Proc.



Speech

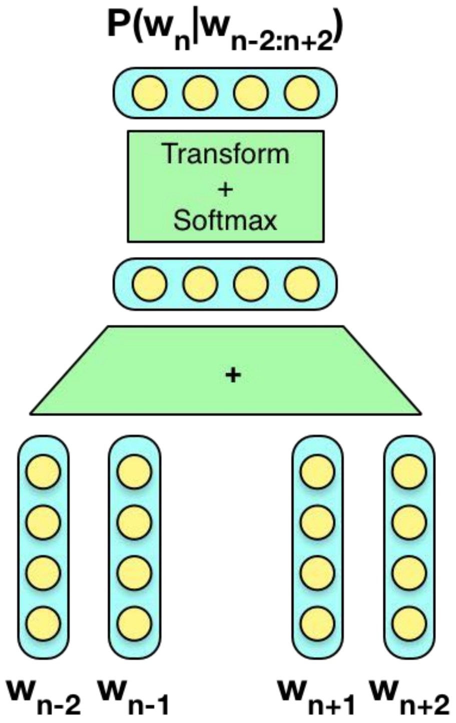


Reinf. Learning

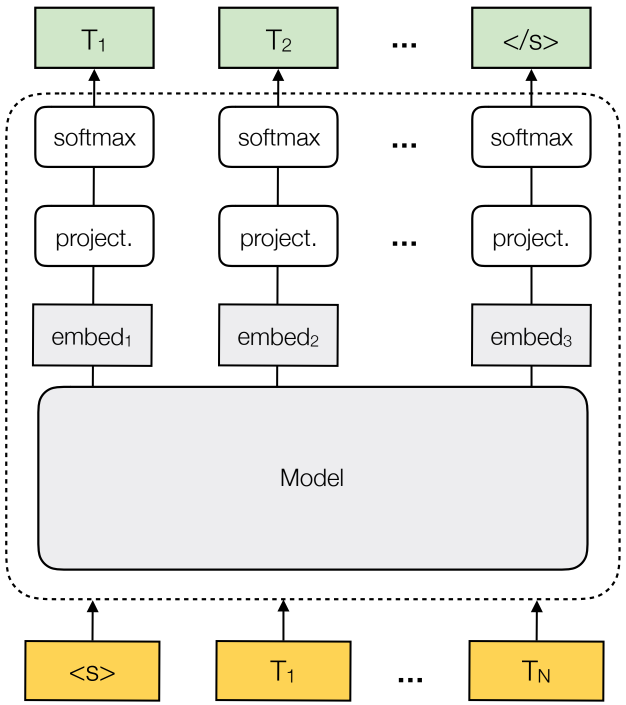


Graphs / Science

Pre-training in NLP (before Transformers)



word embeddings
word2vec
[Mikolov et al., 2013]

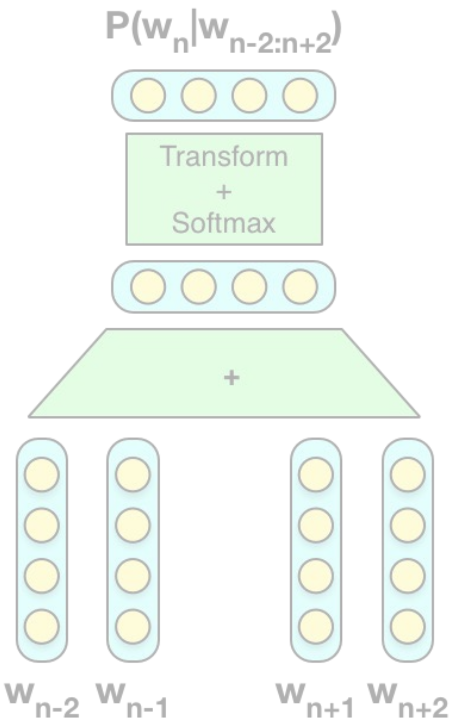


contextualized
word embeddings via LM
ELMo
[Peters et al., 2018]

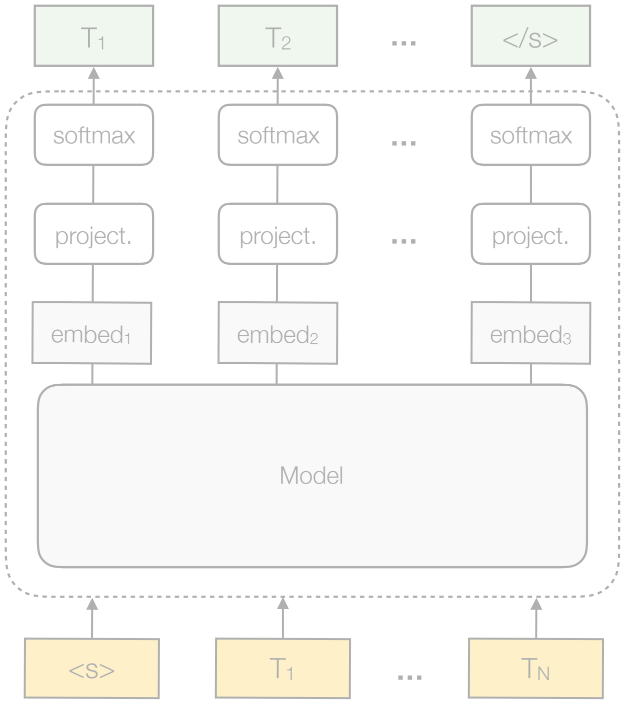
- Word embeddings ⇒ Contextualized word embeddings

Image credit: Noe Casas

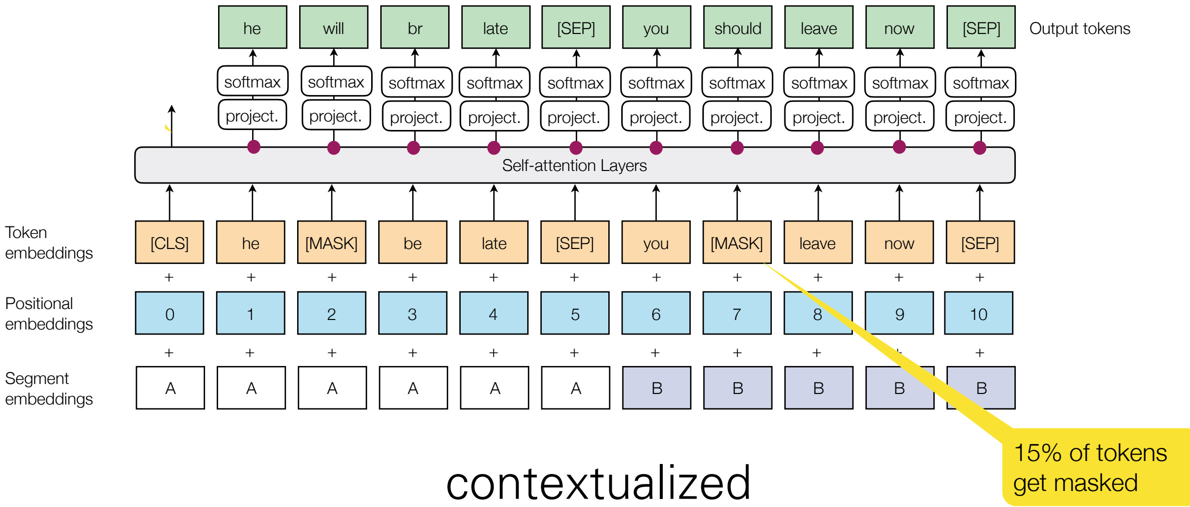
Pre-training in NLP (during Transformers)



word embeddings
word2vec
[Mikolov et al., 2013]



contextualized
word embeddings via LM
ELMo
[Peters et al., 2018]



contextualized
word embeddings via
masked LM +
next sentence prediction
BERT
[Devlin et al., 2019]

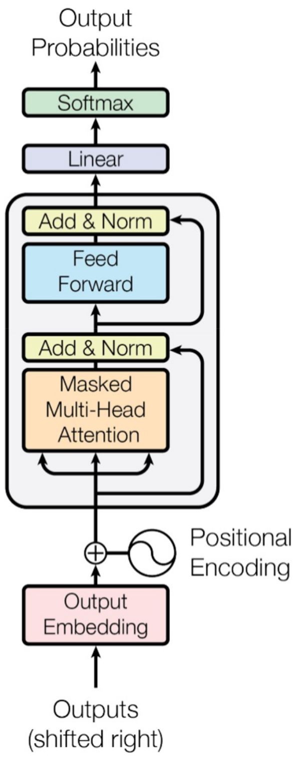
- Word embeddings ⇒ Contextualized word embeddings ⇒ Transformers
- Transformer-based models take over the language modelling / NLP domain

Image credit: Noe Casas

Pre-training in NLP (during Transformers)

Decoder-only
GPT

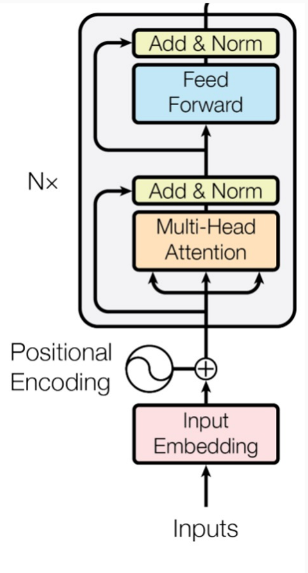
[sat_]



[START] [The_] [cat_]

Encoder-only
BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



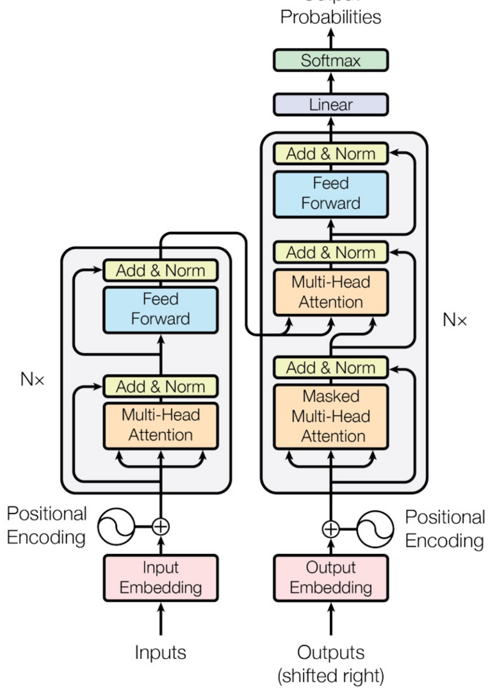
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Enc-Dec
T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

Pre-training in Vision (during Transformers)

Many prior works attempted to introduce self-attention at the pixel level.

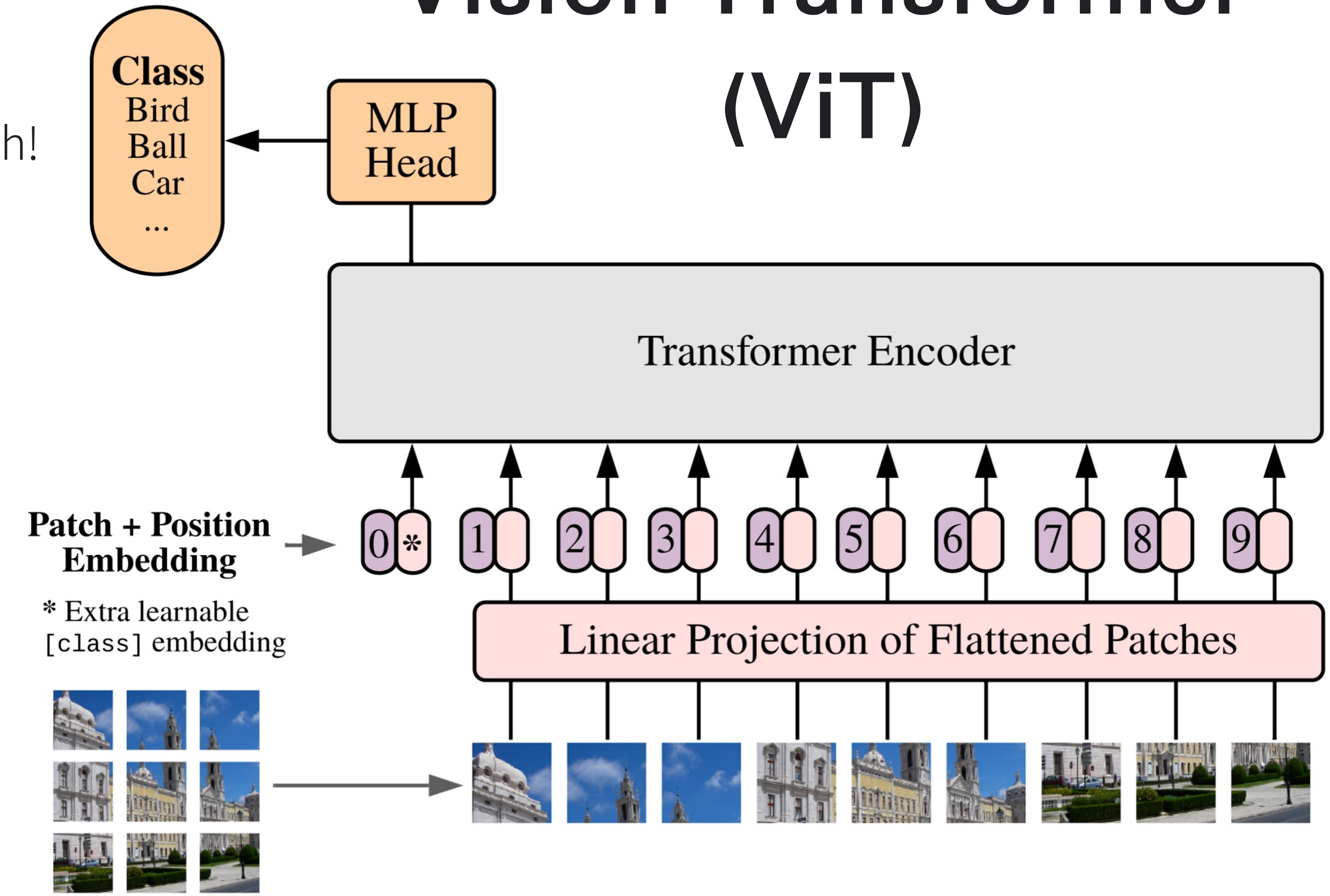
For $224px^2$, that's 50k sequence length, too much!

Thus, most works restrict attention to local pixel neighborhoods, or as high-level mechanism on top of detections.

The **key breakthrough** in using the full Transformer architecture, standalone, was to **"tokenize" the image** by **cutting it into patches** of $16px^2$, and treating each patch as a token, e.g. embedding it into input space.

Transformer-based models take over the vision domain!

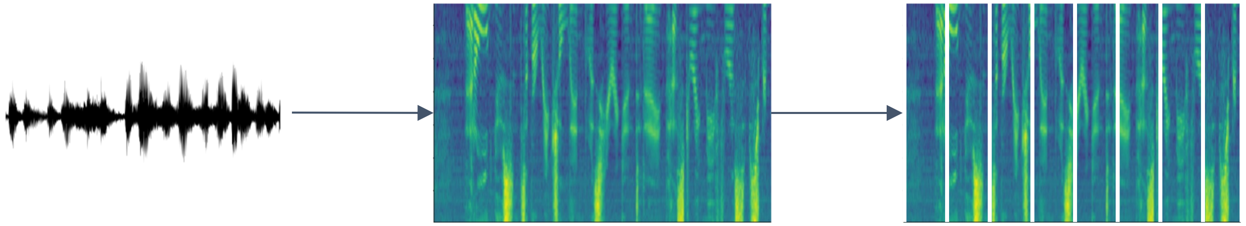
Vision Transformer (ViT)



Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Pre-training in Speech (during Transformers)

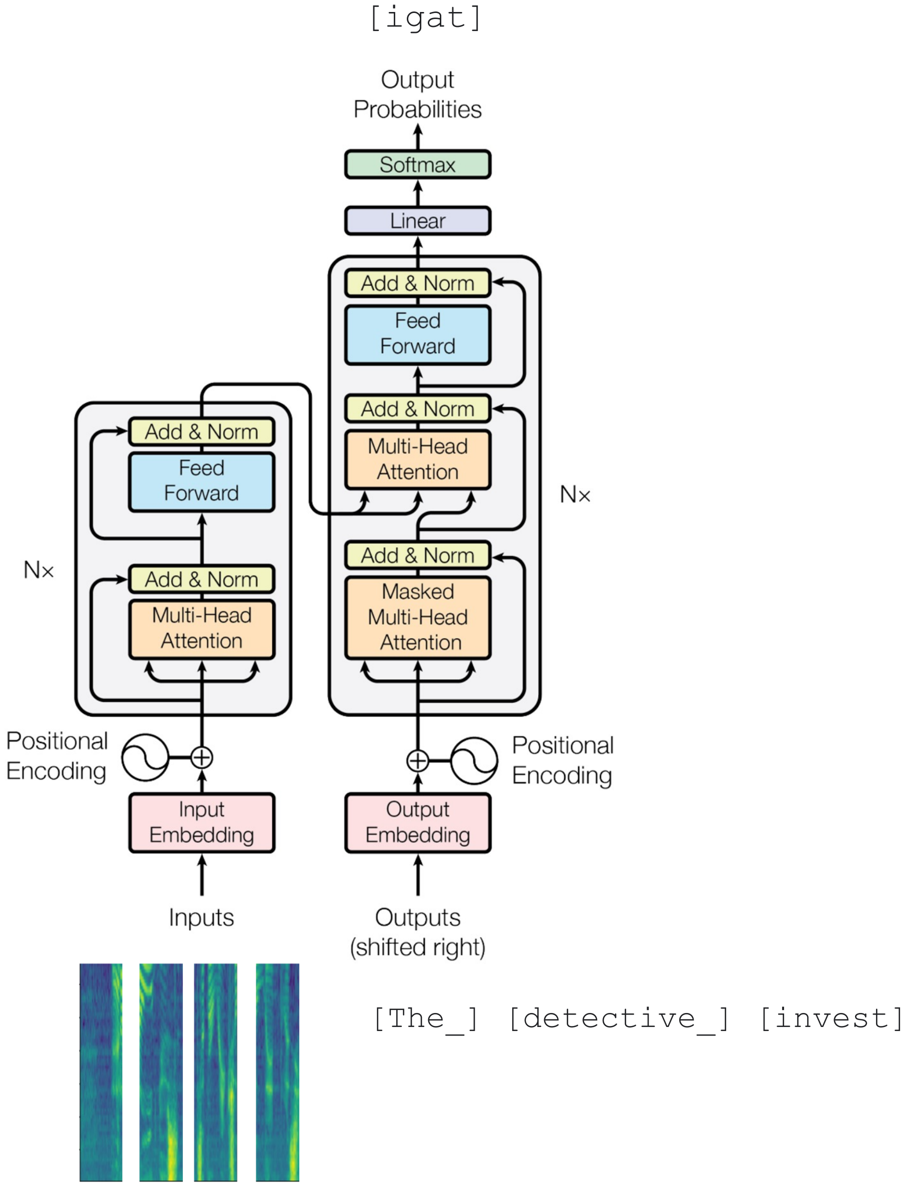
Largely the same story as in computer vision.
But with spectrograms instead of images.



Add a third type of block using convolutions, and slightly reorder blocks, but overall very transformer-like.

Exists as encoder-decoder variant, or as encoder-only variant with CTC loss.

Transformer-based models take over the speech domain!



Gulati et al. Conformer: Convolution-augmented Transformer for Speech Recognition. In INTERSPEECH 2020

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]
- ViTs are an evolution, not a revolution. We can still fundamentally solve the same problems as with CNNs.
- Matrix multiply is more hardware-friendly than convolution, so ViTs with same FLOPs as CNNs can train and run much faster

Next lecture:

Deep Generative Models