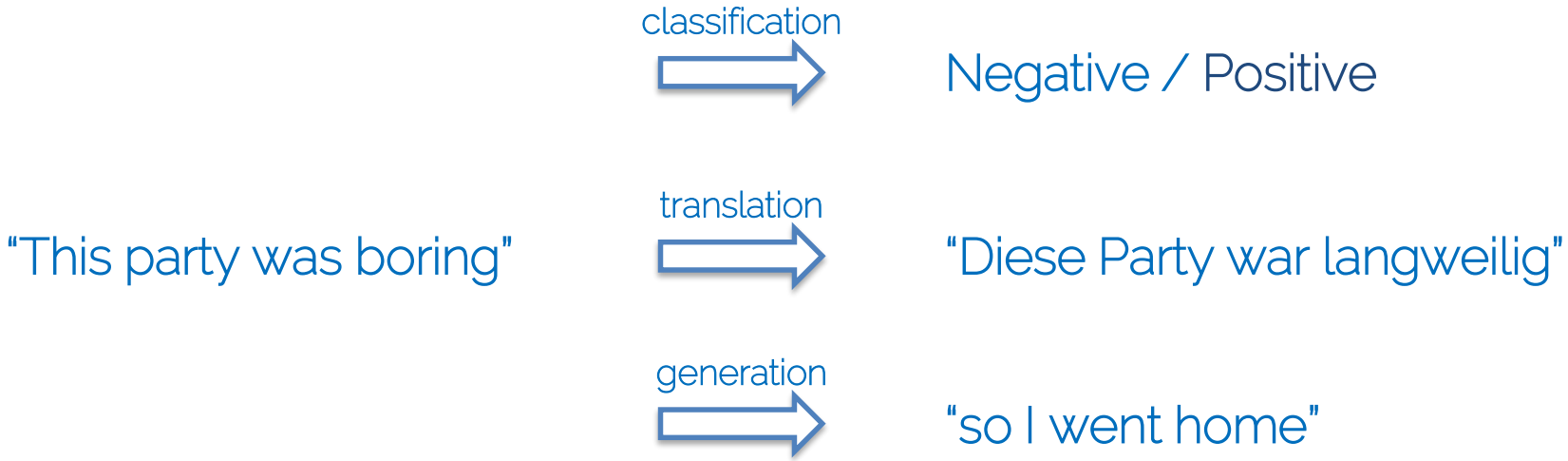
Sequence Models

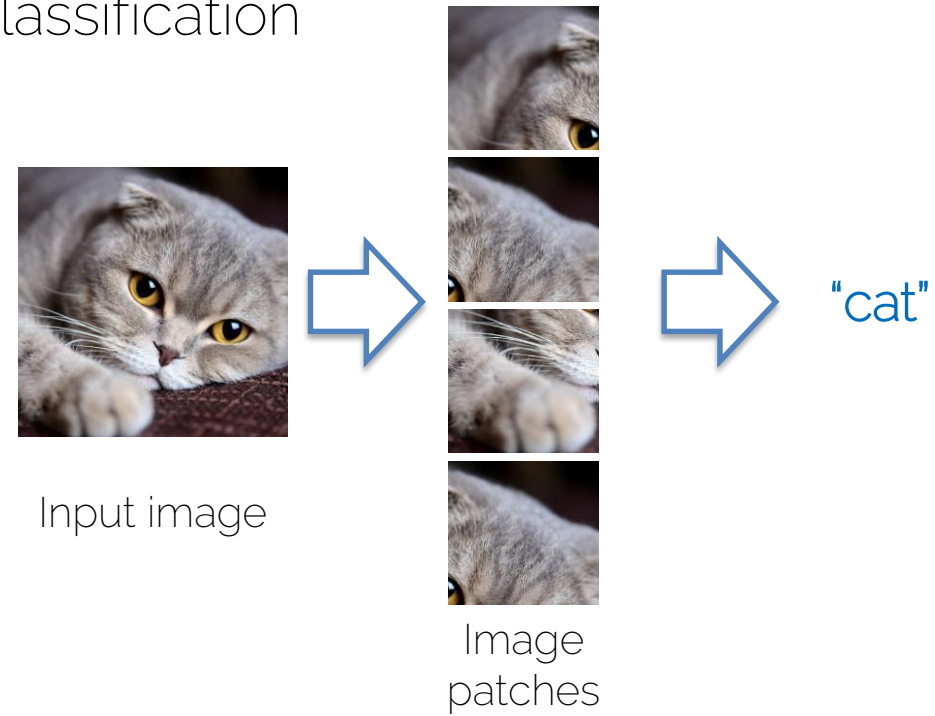
Sequence Modelling

- Texts as sequences
  - Sequences are natural representations for text data



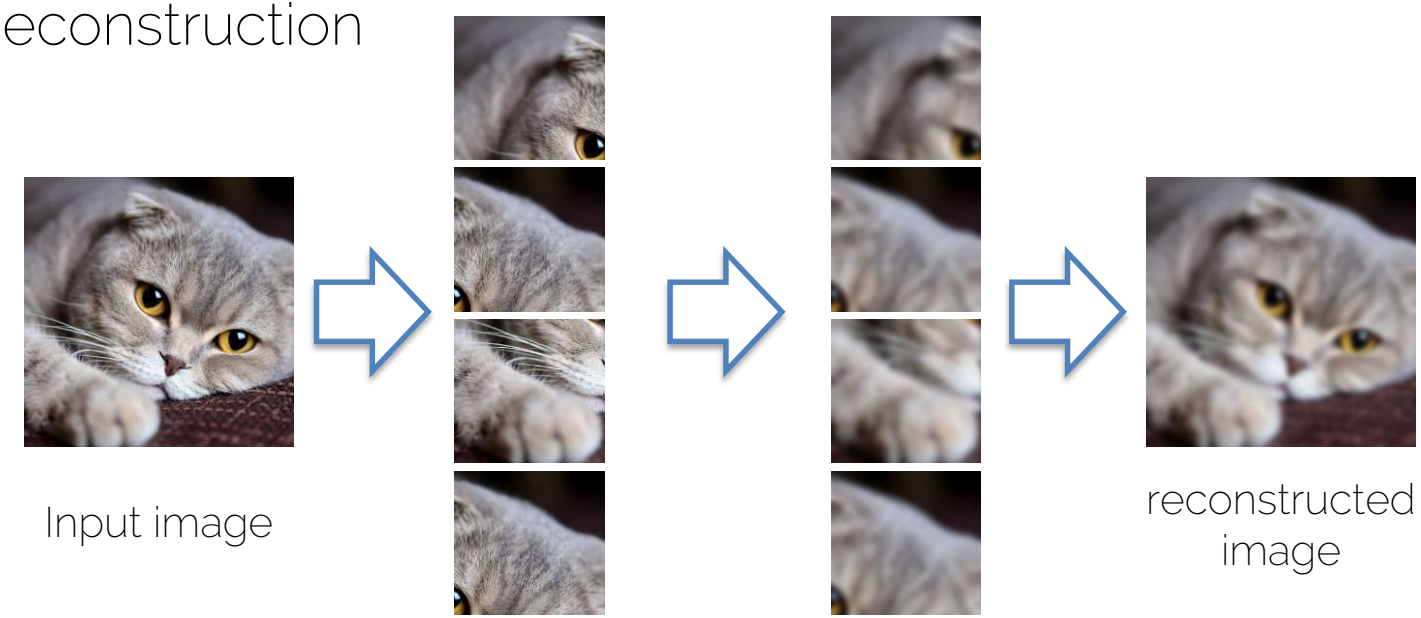
Sequence Modelling

- Images can also be represented as sequences!
  - Classification



Sequence Modelling

- Images can also be represented as sequences!
  - Reconstruction



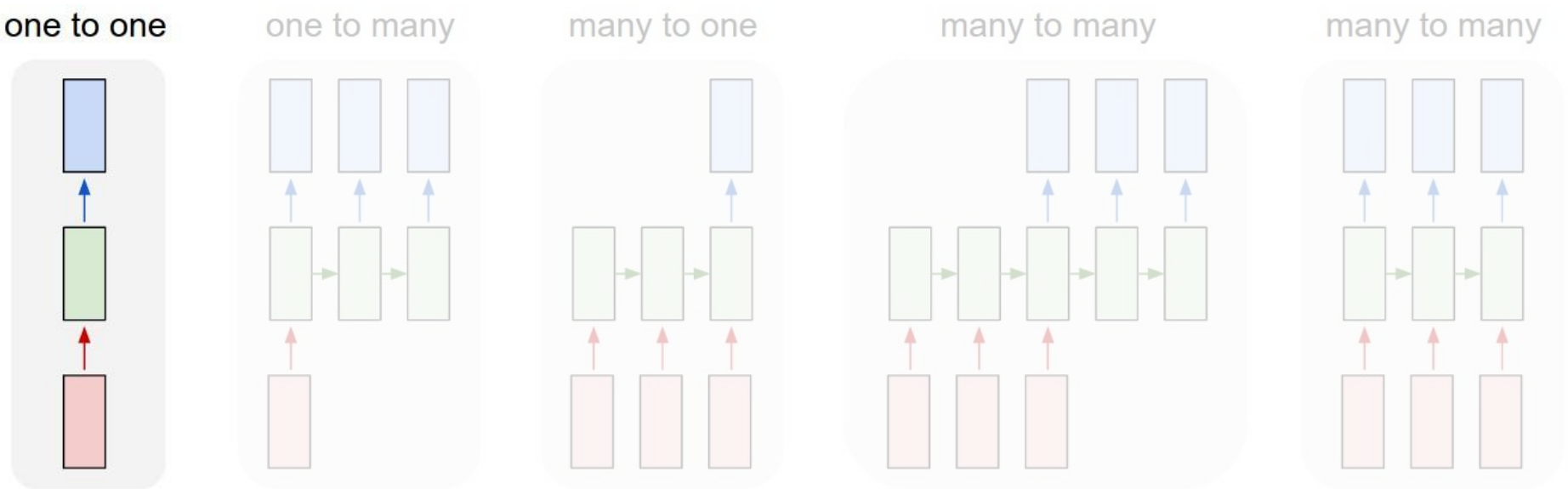
Processing Sequences

- Recurrent neural networks process sequence data

- Input/output can be sequences

Recurrent Neural Networks

RNNs are Flexible



Classical neural networks for image classification

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

RNNs are Flexible

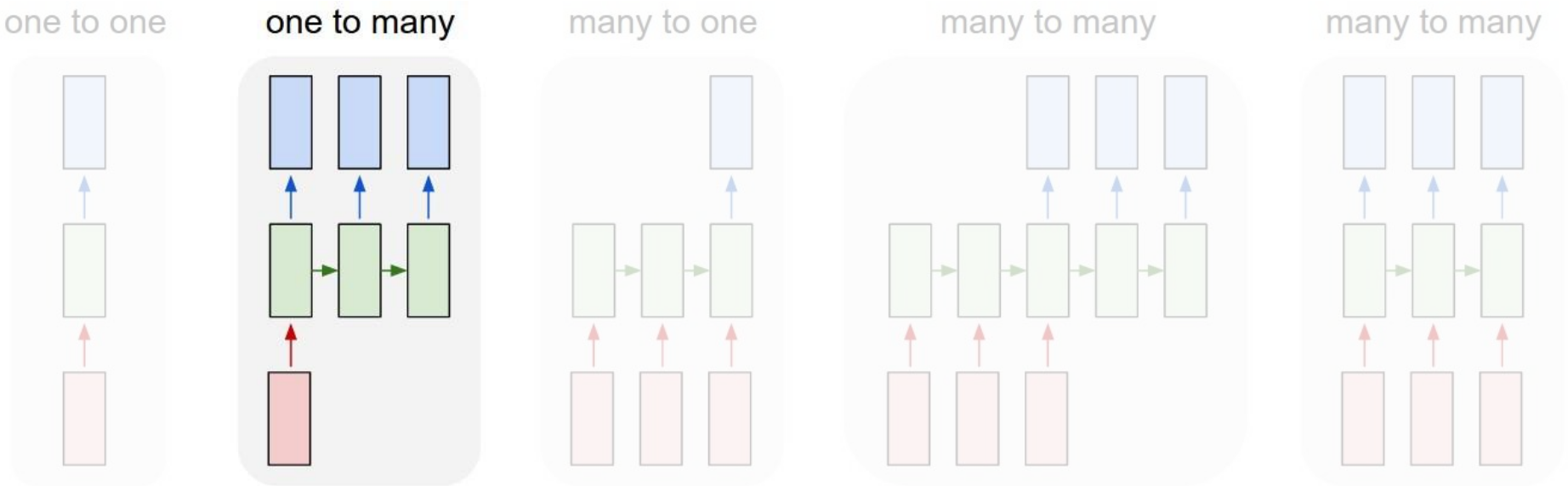
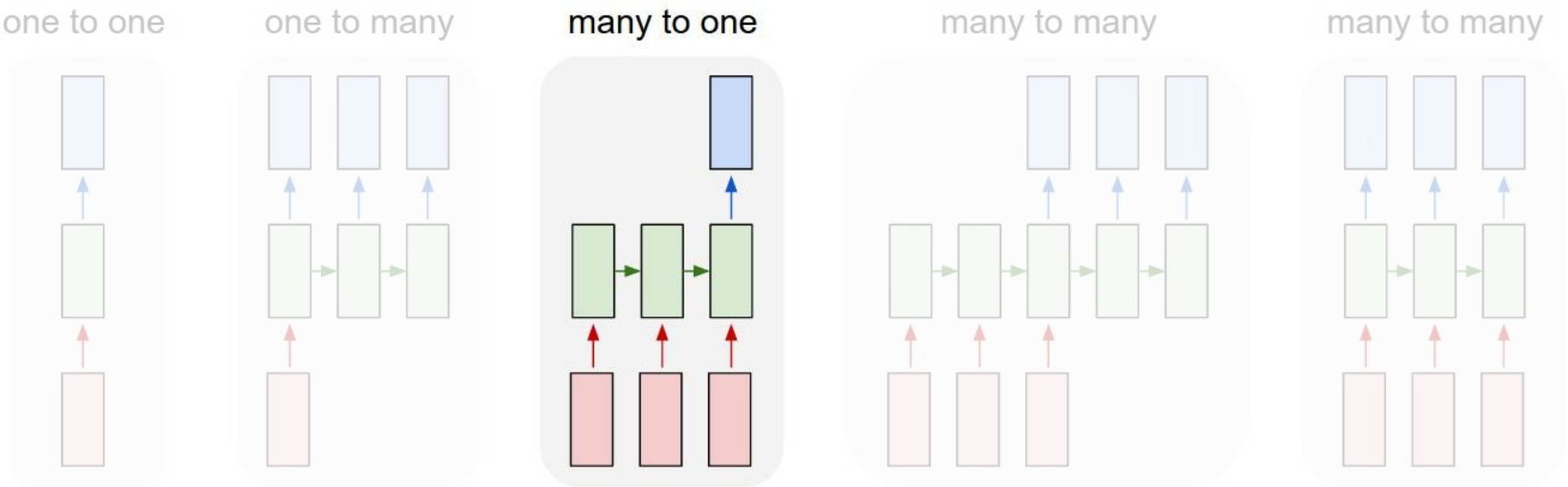


Image captioning

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

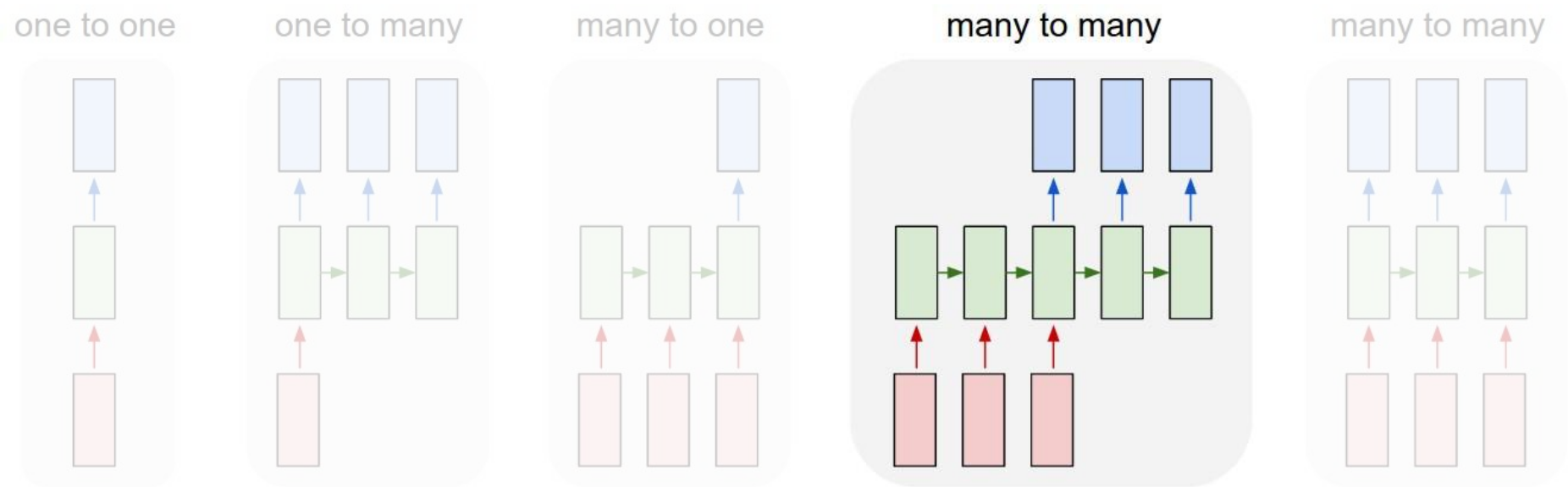
RNNs are Flexible



Language recognition

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

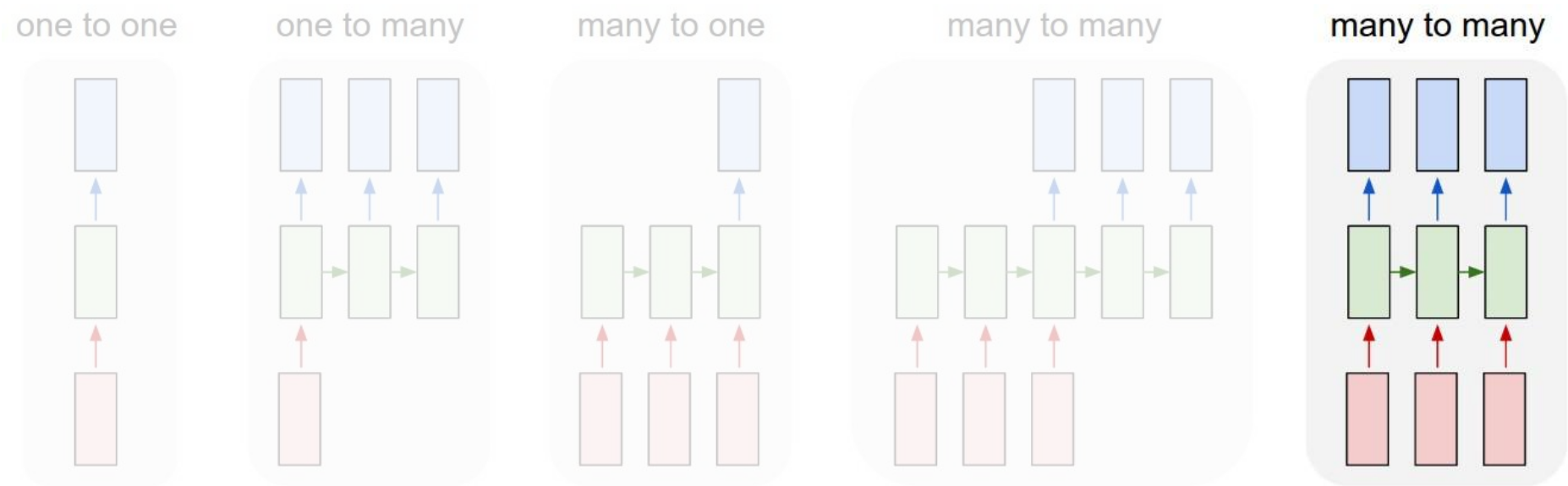
RNNs are Flexible



Machine translation

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

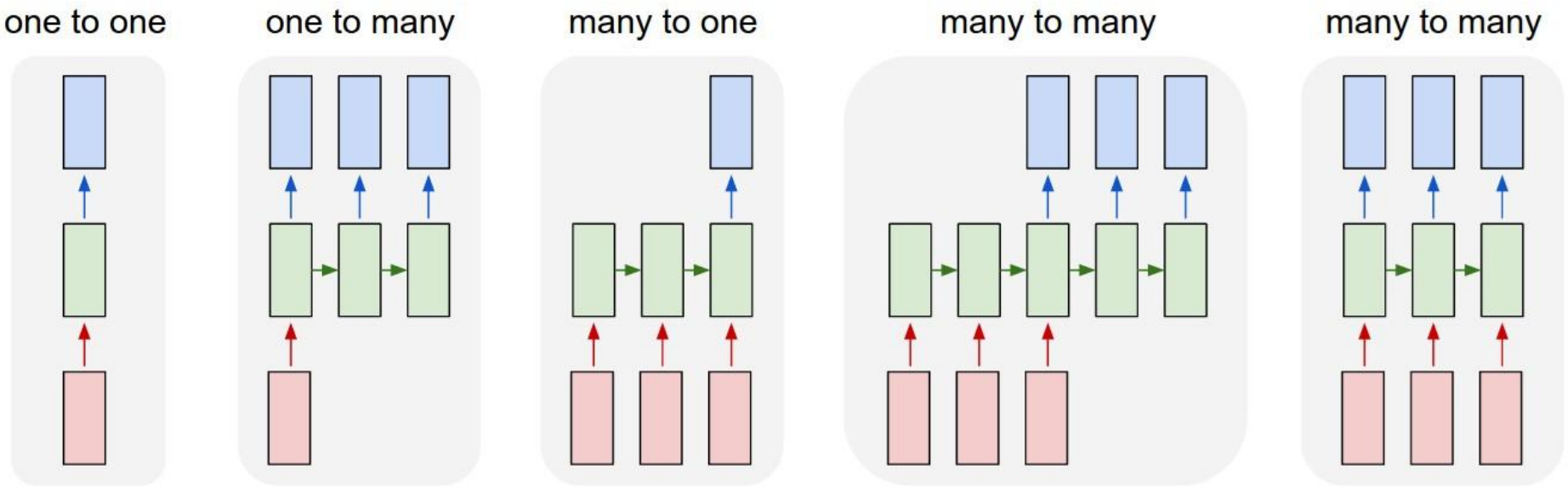
RNNs are Flexible



Event classification

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

RNNs are Flexible

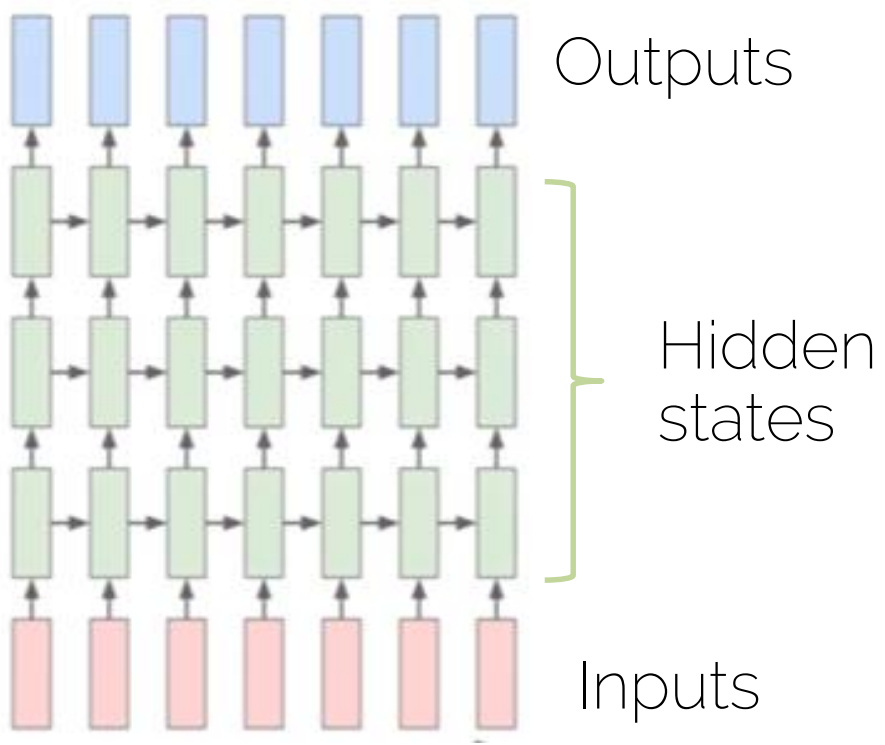


Event classification

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

Basic Structure of an RNN

- Multi-layer RNN

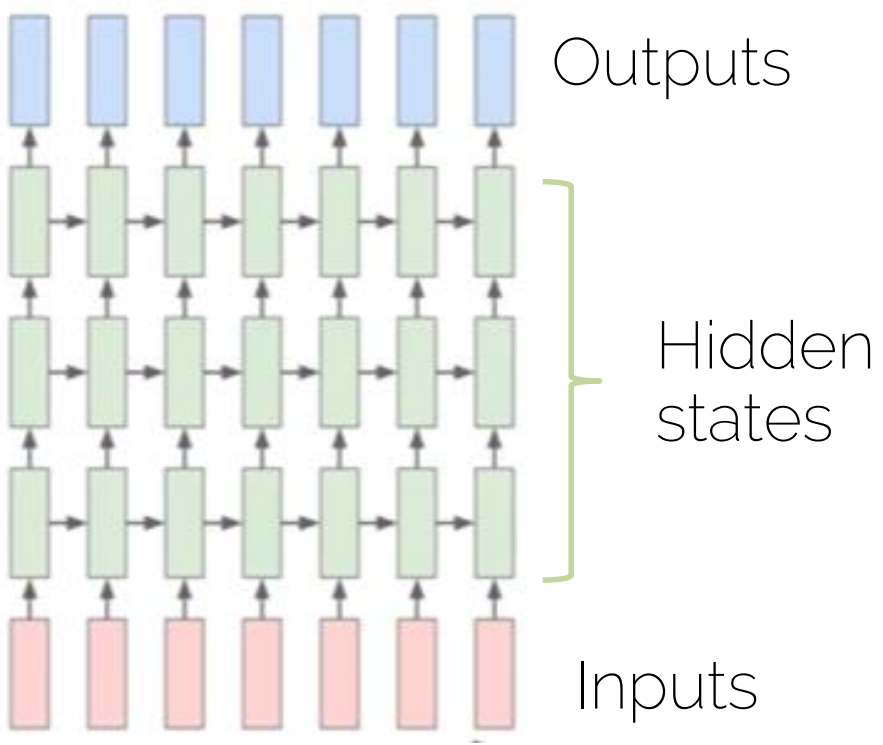


Basic Structure of an RNN

- Multi-layer RNN

The hidden state will have its own internal dynamics

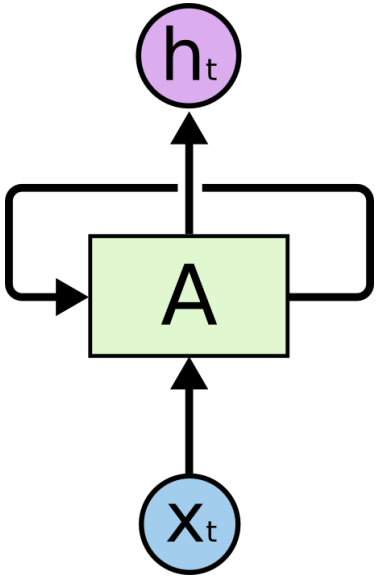
More expressive model!



Basic Structure of an RNN

- We want to have notion of "time" or "sequence"

Hidden state



$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

Previous hidden state

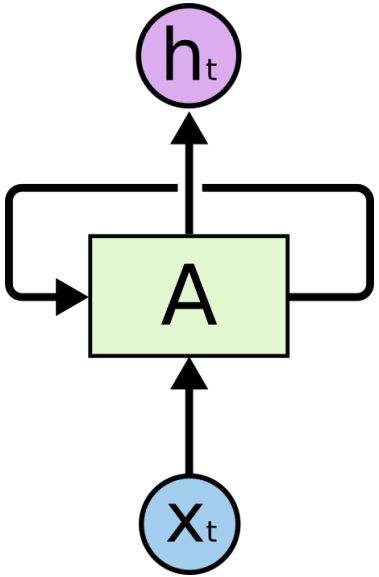
input

[Olah, https://colah.github.io '15] Understanding LSTMs

Basic Structure of an RNN

- We want to have notion of "time" or "sequence"

Hidden state



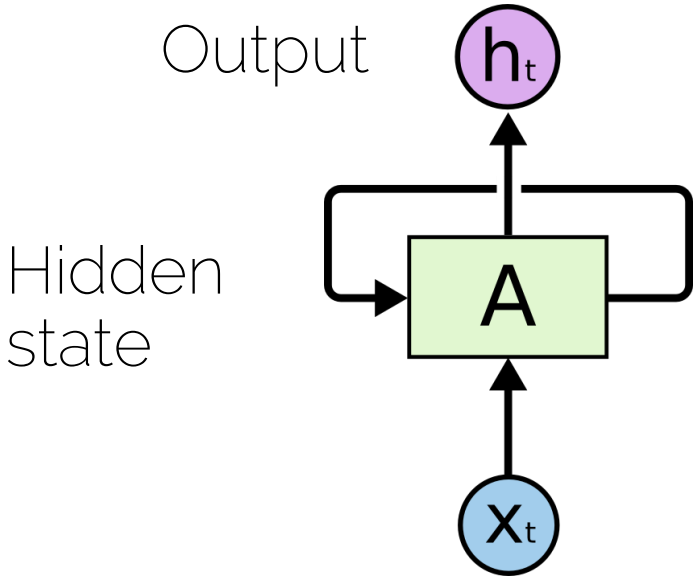
$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

Parameters to be learned

[Olah, https://colah.github.io '15] Understanding LSTMs

Basic Structure of an RNN

- We want to have notion of "time" or "sequence"



$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

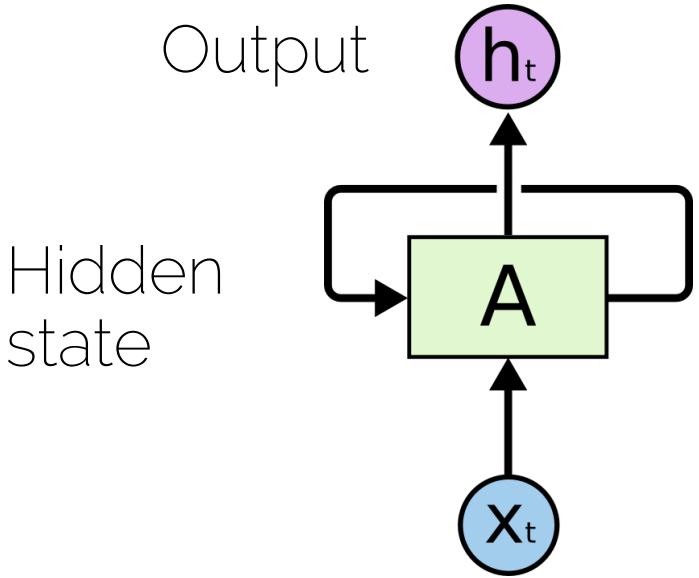
$$\boldsymbol{h_t = \theta_h A_t}$$

Note: non-linearities ignored for now

[Olah, https://colah.github.io '15] Understanding LSTMs

Basic Structure of an RNN

- We want to have notion of "time" or "sequence"



$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

$$\boldsymbol{h_t = \theta_h A_t}$$

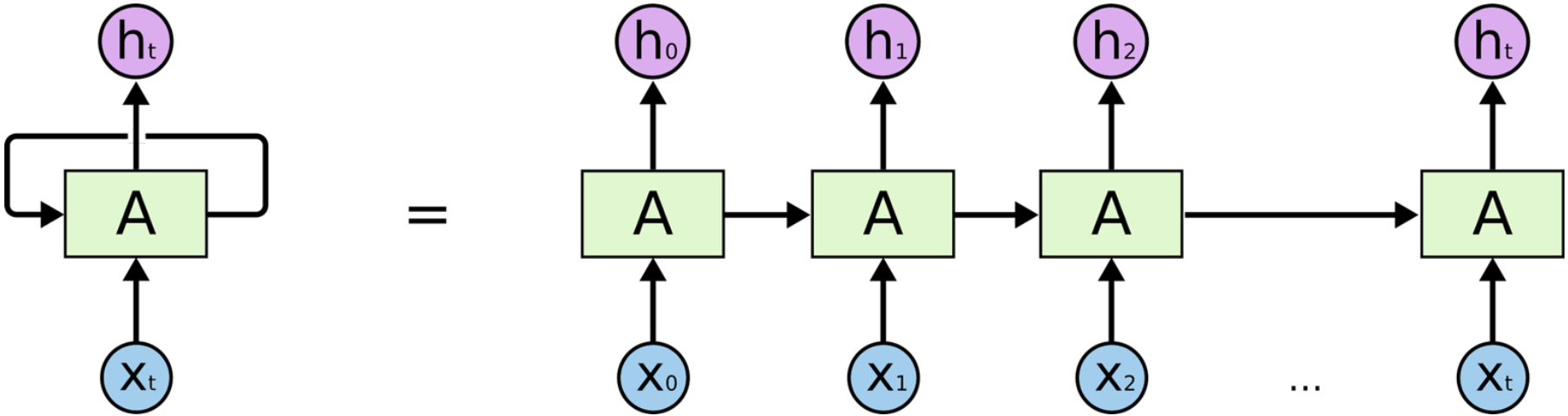
Same parameters for each time step = generalization!

[Olah, https://colah.github.io '15] Understanding LSTMs

Basic Structure of an RNN

- Unrolling RNNs

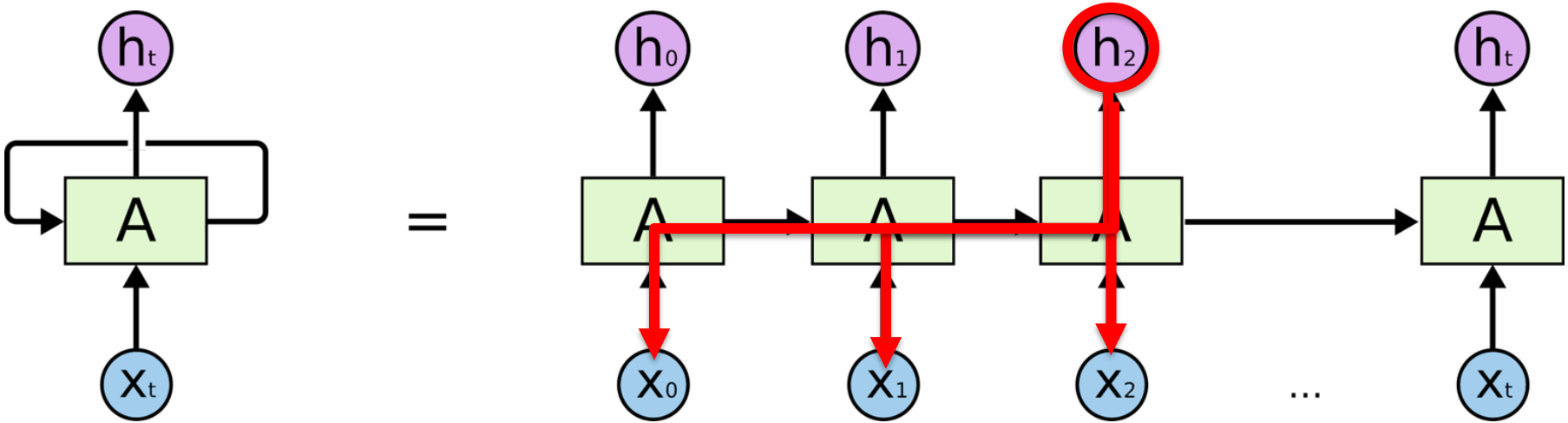
Same function for the hidden layers



[Olah, https://colah.github.io '15] Understanding LSTMs

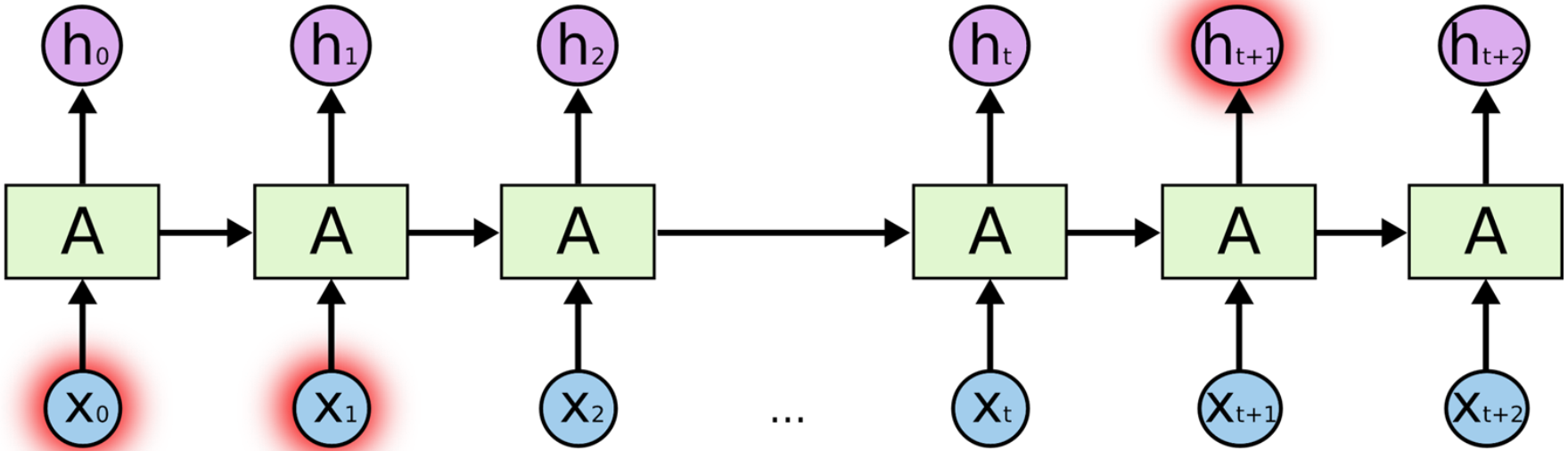
Basic Structure of an RNN

- Unrolling RNNs



[Olah, https://colah.github.io '15] Understanding LSTMs

Long-term Dependencies



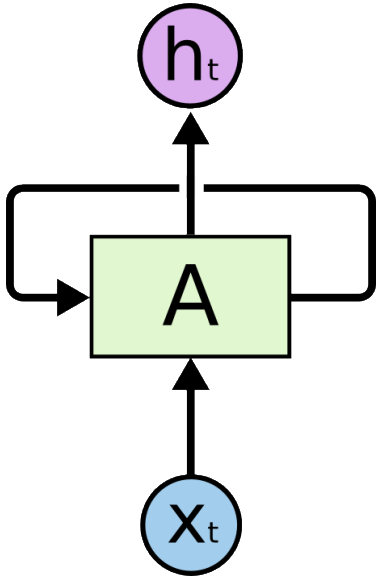
I moved to Germany … so I speak German fluently.

[Olah, https://colah.github.io '15] Understanding LSTMs

Long-term Dependencies

- Simple recurrence $\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$

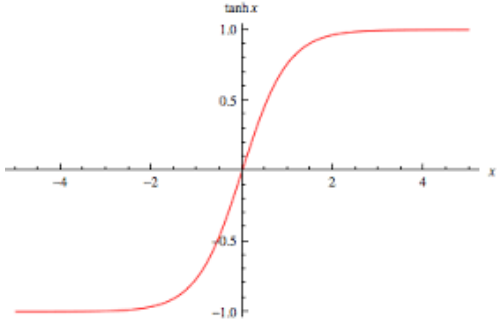
- Let us forget the input $\boldsymbol{A_t = \theta_c{}^t A_0}$



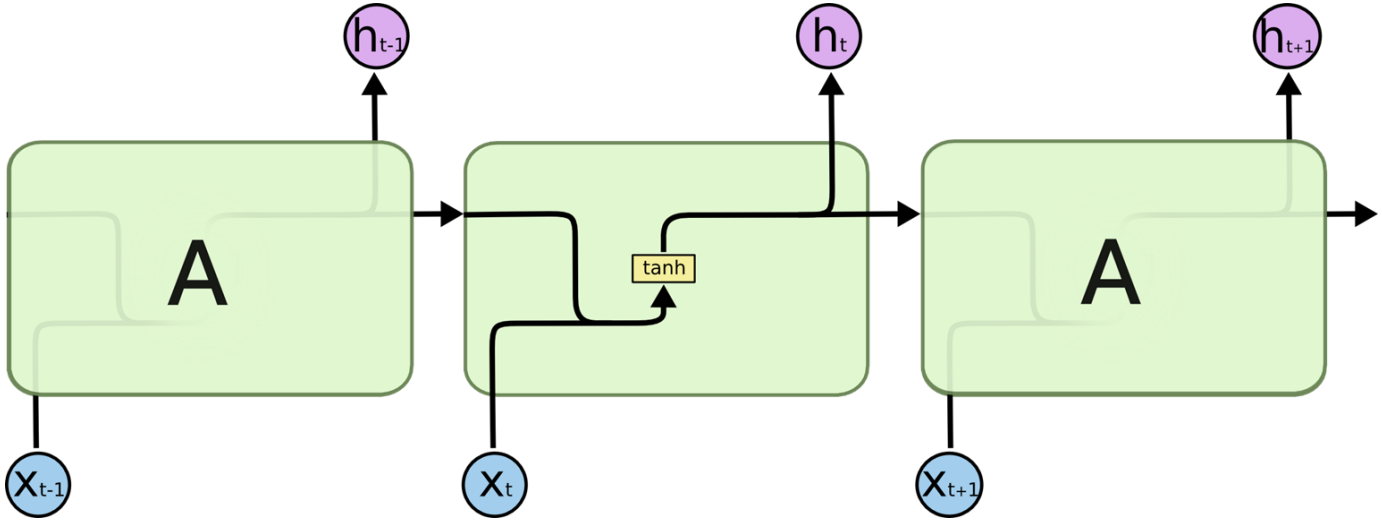
Same weights are multiplied over and over again

Vanishing Gradient

- 1. From the weights $\boldsymbol{A_t = \theta_c{}^t A_0}$



- 2. From the activation functions ($\boldsymbol{tanh}$)



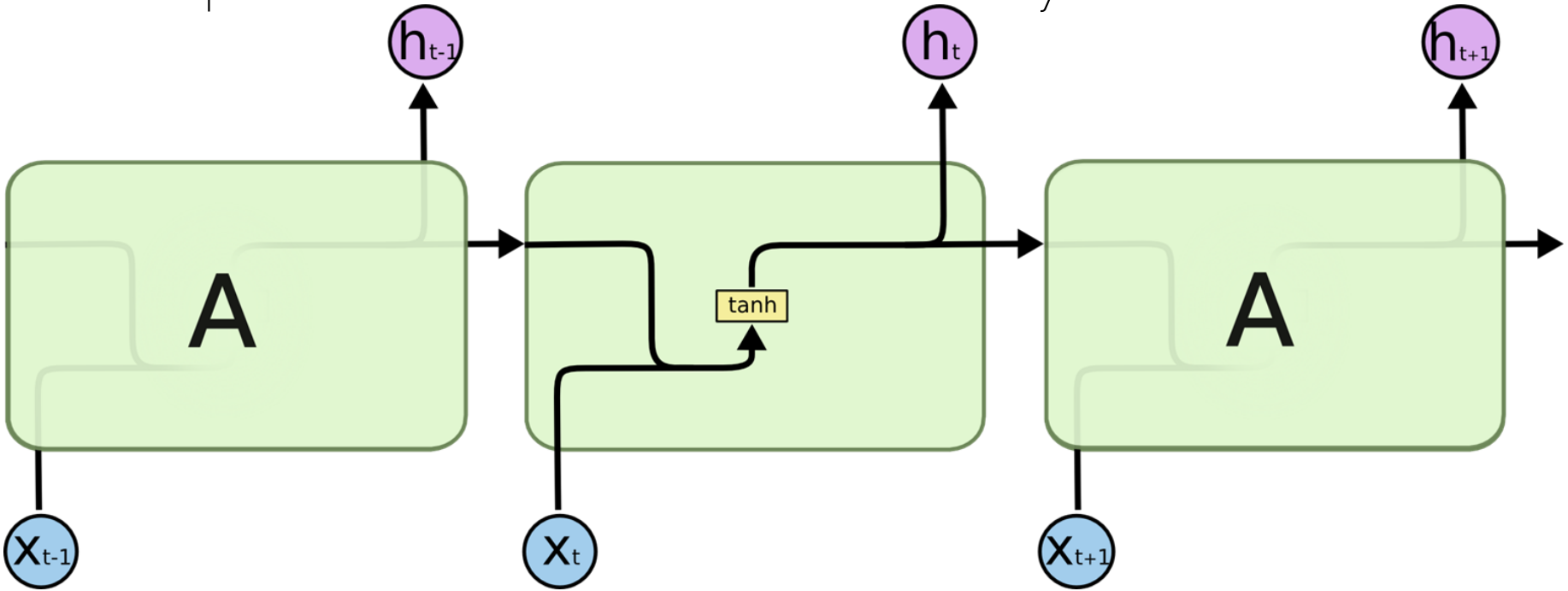
[Olah, https://colah.github.io '15] Understanding LSTMs

Long Short Term Memory

[Hochreiter et al., Neural Computation'97] Long Short-Term Memory

Long-Short Term Memory Units

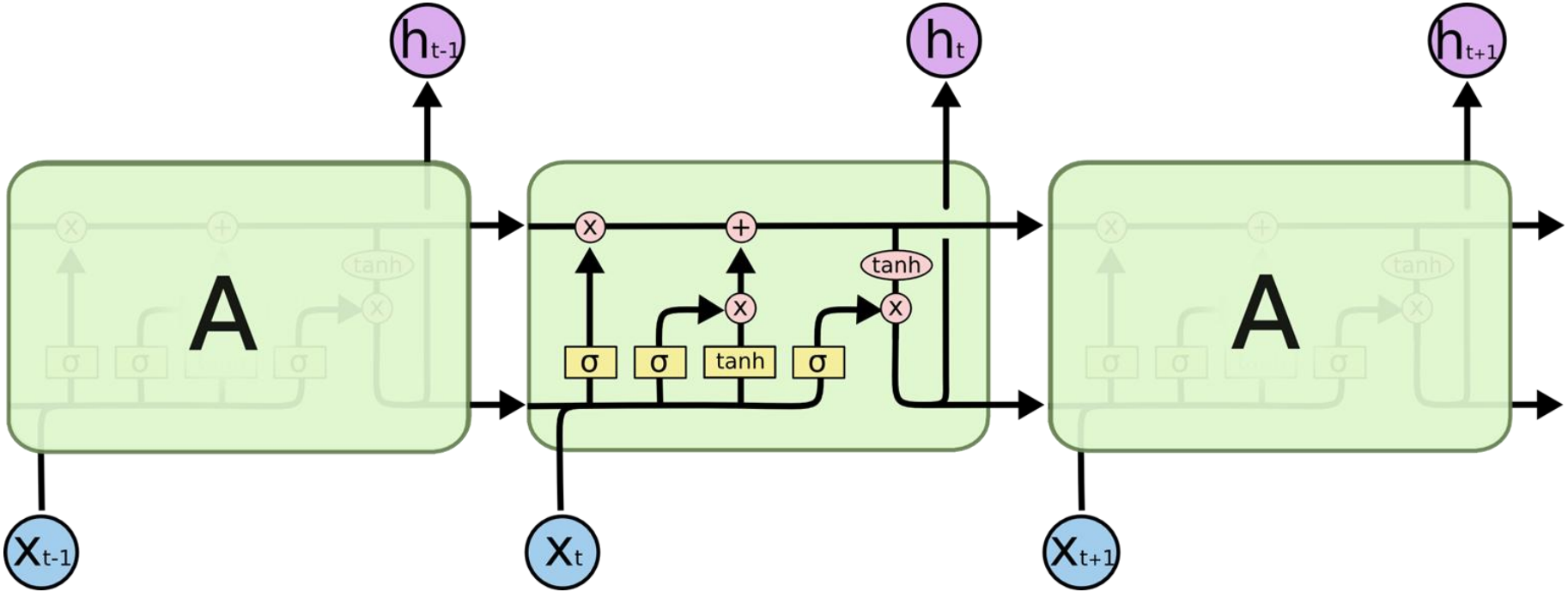
- Simple RNN has **tanh** as non-linearity



[Olah, https://colah.github.io '15] Understanding LSTMs

Long-Short Term Memory Units

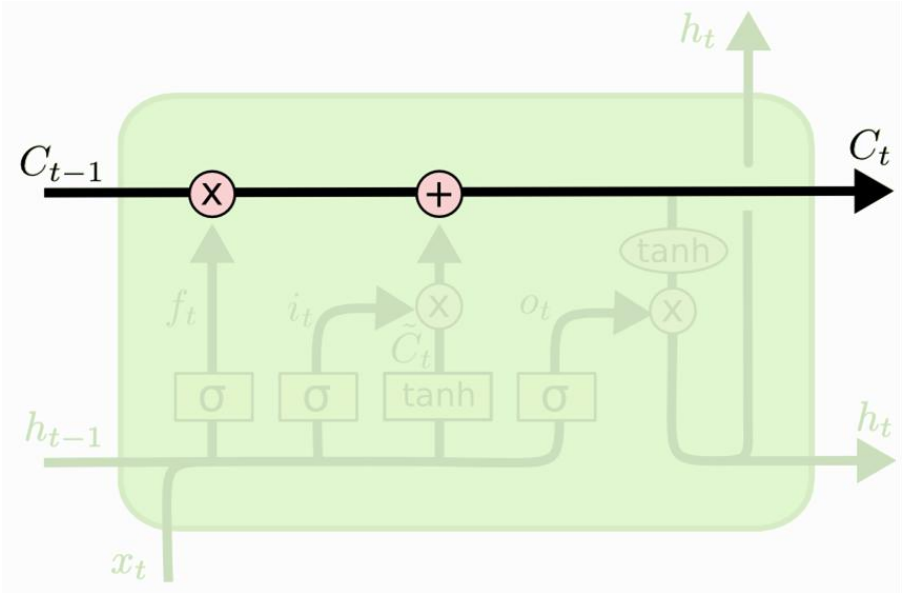
LSTM



[Olah, https://colah.github.io '15] Understanding LSTMs

Long-Short Term Memory Units

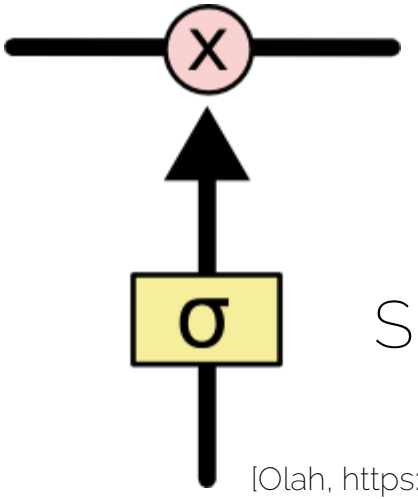
- Key ingredients
- Cell = transports the information through the unit



[Olah, https://colah.github.io '15] Understanding LSTMs

Long-Short Term Memory Units

- Key ingredients
- Cell = transports the information through the unit
- Gate = remove or add information to the cell state

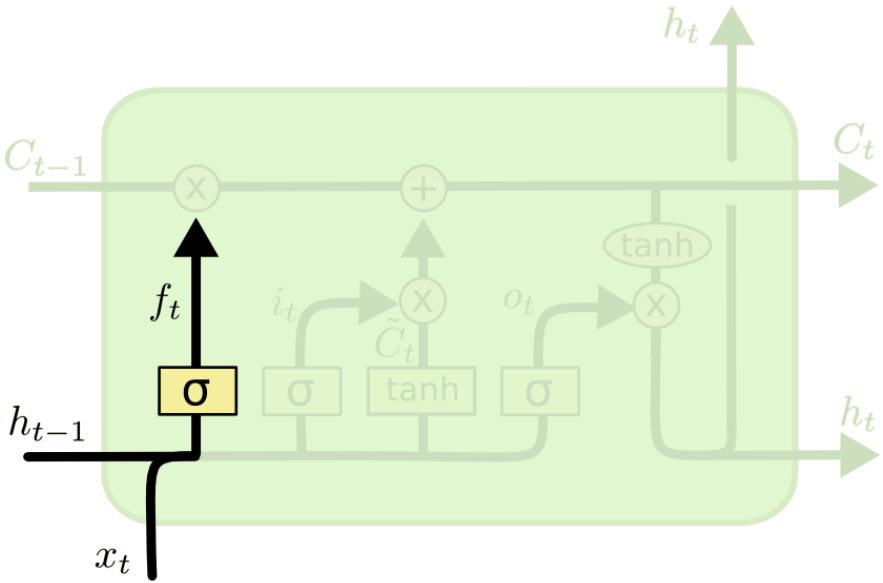


Sigmoid

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Step by Step

- Forget gate $\boldsymbol{f_t = sigm(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)}$



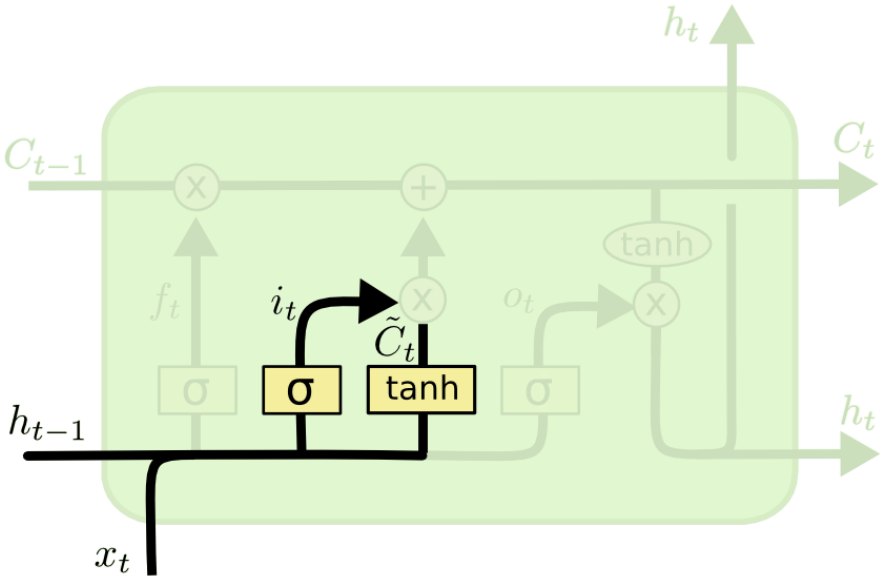
Decides when to erase the cell state

Sigmoid = output between **0** (forget) and **1** (keep)

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Step by Step

- Input gate $\boldsymbol{i_t = sigm(\theta_{xi} x_t + \theta_{hi} h_{t-1} + b_i)}$



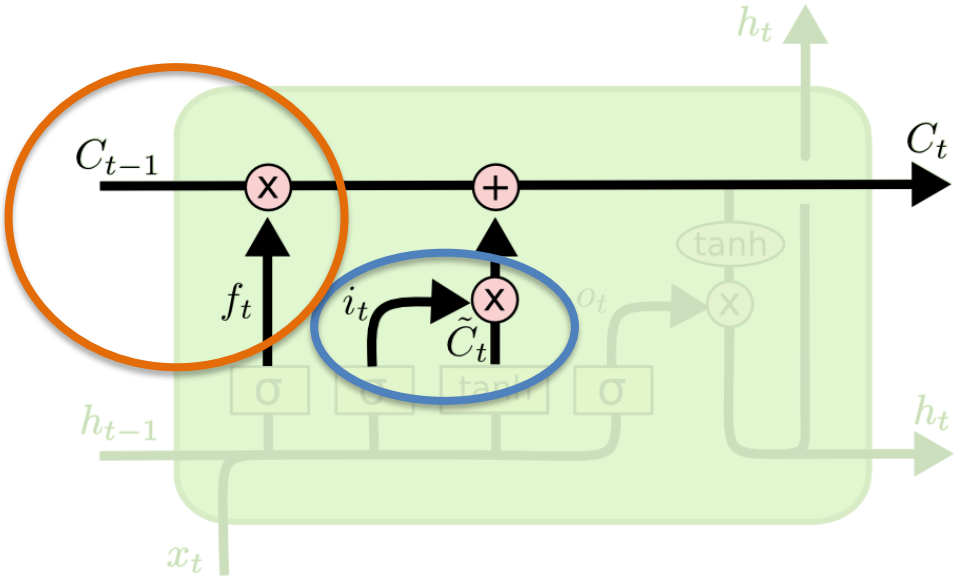
Decides which values will be updated

New cell state, output from a **tanh (−1,1)**

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Step by Step

- Element-wise operations



$$\boldsymbol{C_t} = \boldsymbol{f_t} \odot \boldsymbol{C_{t-1}} + \boldsymbol{i_t} \odot \boldsymbol{g_t}$$

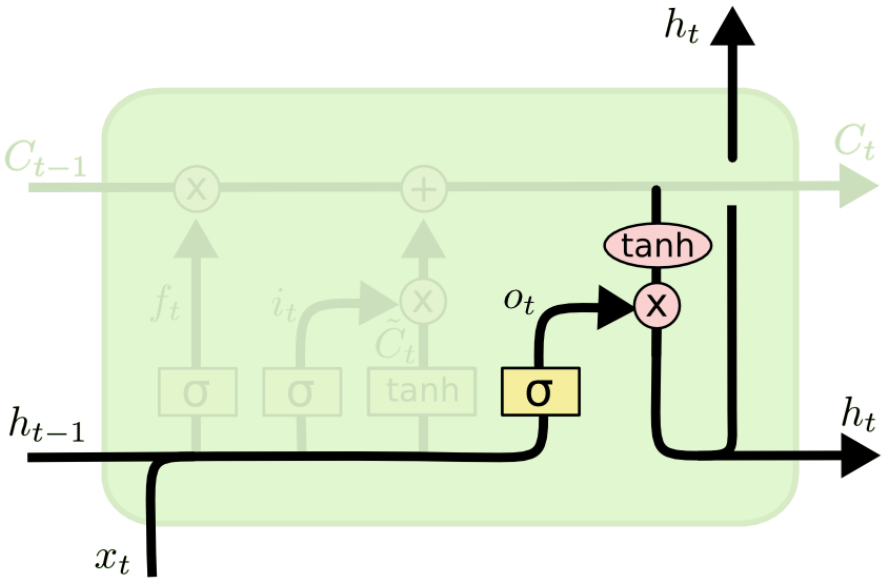
Previous states

Current state

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Step by Step

- Output gate $\boldsymbol{h_t = o_t \odot \tanh(C_t)}$



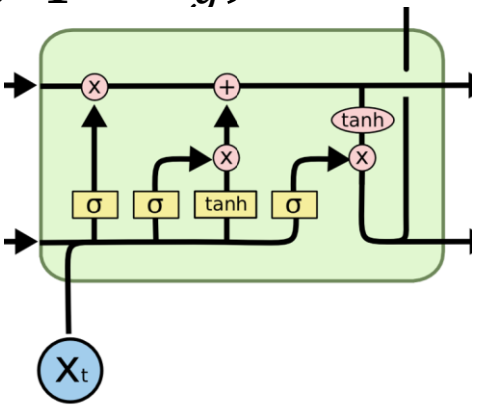
Decides which values will be outputted

Output from a $\boldsymbol{\tanh(-1, 1)}$

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Step by Step

- Forget gate $\boldsymbol{f}_t = sigm(\boldsymbol{\theta}_{xf}\boldsymbol{x}_t + \boldsymbol{\theta}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$
- Input gate $\boldsymbol{i}_t = sigm(\boldsymbol{\theta}_{xi}\boldsymbol{x}_t + \boldsymbol{\theta}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$
- Output gate $\boldsymbol{o}_t = sigm(\boldsymbol{\theta}_{xo}\boldsymbol{x}_t + \boldsymbol{\theta}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$
- Cell update $\boldsymbol{g}_t = tanh(\boldsymbol{\theta}_{xg}\boldsymbol{x}_t + \boldsymbol{\theta}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g)$
- Cell $\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t$
- Output $\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$



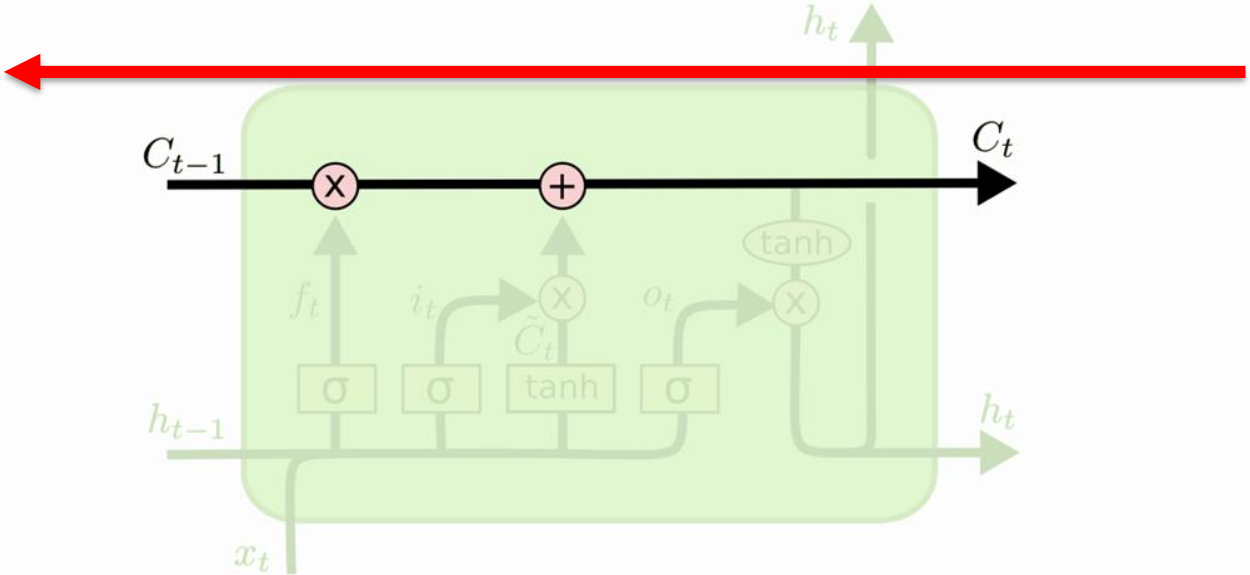
LSTM: Step by Step

- Forget gate $\boldsymbol{f}_t = sigm(\boldsymbol{\theta}_{xf}\boldsymbol{x}_t + \boldsymbol{\theta}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$
- Input gate $\boldsymbol{i}_t = sigm(\boldsymbol{\theta}_{xi}\boldsymbol{x}_t + \boldsymbol{\theta}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$
- Output gate $\boldsymbol{o}_t = sigm(\boldsymbol{\theta}_{xo}\boldsymbol{x}_t + \boldsymbol{\theta}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$
- Cell update $\boldsymbol{g}_t = tanh(\boldsymbol{\theta}_{xg}\boldsymbol{x}_t + \boldsymbol{\theta}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g)$
- Cell $\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t$
- Output $\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$

Learned through backpropagation

LSTM

- Highway for the gradient to flow

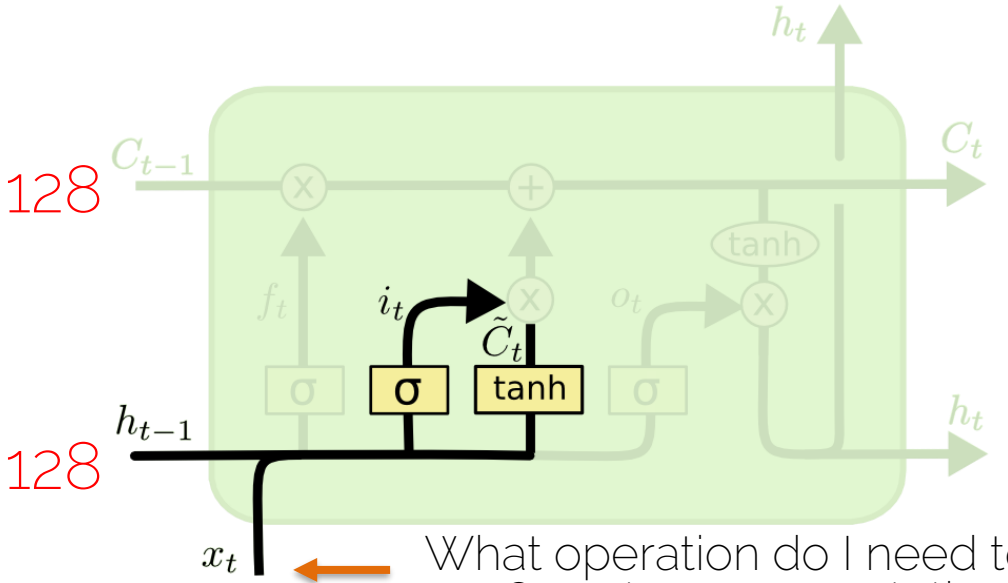


[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM: Dimensions

- Cell update $\boldsymbol{g}_t = tanh(\boldsymbol{\theta}_{xg}\boldsymbol{x}_t + \boldsymbol{\theta}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g)$

128 over $\boldsymbol{g}_t$, 128 over $\boldsymbol{\theta}_{xg}\boldsymbol{x}_t$, 128 over $\boldsymbol{\theta}_{hg}\boldsymbol{h}_{t-1}$



When coding an LSTM, we have to define the size of the hidden state

Dimensions need to match

What operation do I need to do to my input to get a 128 vector representation?

[Olah, https://colah.github.io '15] Understanding LSTMs

LSTM in code

```python
def lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b):
  """
  Forward pass for a single timestep of an LSTM.

  The input data has dimension D, the hidden state has dimension H, and we use
  a minibatch size of N.

  Inputs:
  - x: Input data, of shape (N, D)
  - prev_h: Previous hidden state, of shape (N, H)
  - prev_c: previous cell state, of shape (N, H)
  - Wx: Input-to-hidden weights, of shape (D, 4H)
  - Wh: Hidden-to-hidden weights, of shape (H, 4H)
  - b: Biases, of shape (4H,)

  Returns a tuple of:
  - next_h: Next hidden state, of shape (N, H)
  - next_c: Next cell state, of shape (N, H)
  - cache: Tuple of values needed for backward pass.
  """
  next_h, next_c, cache = None, None, None

  N, H = prev_h.shape
  # 1
  a = np.dot(x, Wx) + np.dot(prev_h, Wh) + b

  # 2
  ai = a[:, :H]
  af = a[:, H:2*H]
  ao = a[:, 2*H:3*H]
  ag = a[:, 3*H:]

  # 3
  i = sigmoid(ai)
  f = sigmoid(af)
  o = sigmoid(ao)
  g = np.tanh(ag)

  # 4
  next_c = f * prev_c + i * g

  # 5
  next_h = o * np.tanh(next_c)

  cache = i, f, o, g, a, ai, af, ao, ag, Wx, Wh, b, prev_h, prev_c, x, next_c, next_h

  return next_h, next_c, cache
```

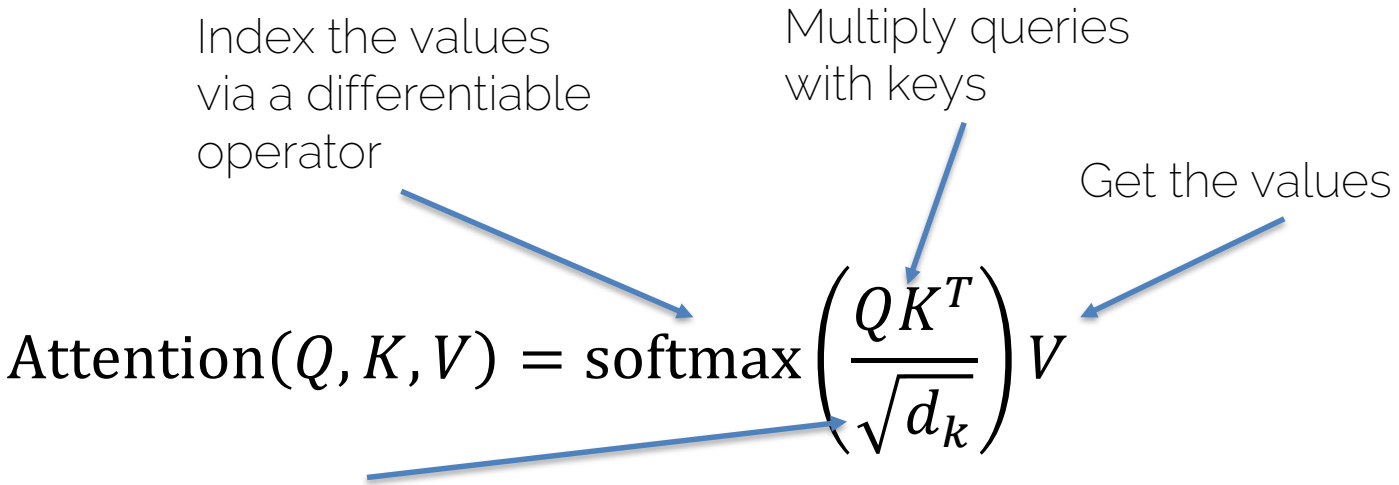
```python
def lstm_step_backward(dnext_h, dnext_c, cache):
  """
  Backward pass for a single timestep of an LSTM.

  Inputs:
  - dnext_h: Gradients of next hidden state, of shape (N, H)
  - dnext_c: Gradients of next cell state, of shape (N, H)
  - cache: Values from the forward pass

  Returns a tuple of:
  - dx: Gradient of input data, of shape (N, D)
  - dprev_h: Gradient of previous hidden state, of shape (N, H)
  - dprev_c: Gradient of previous cell state, of shape (N, H)
  - dWx: Gradient of input-to-hidden weights, of shape (D, 4H)
  - dWh: Gradient of hidden-to-hidden weights, of shape (H, 4H)
  - db: Gradient of biases, of shape (4H,)
  """
  dx, dh, dc, dWx, dWh, db = None, None, None, None, None, None

  i, f, o, g, a, ai, af, ao, ag, Wx, Wh, b, prev_h, prev_c, x, next_c, next_h = cache

  # backprop into step 5
  do = np.tanh(next_c) * dnext_h
  dnext_c += o * (1 - np.tanh(next_c) ** 2) * dnext_h

  # backprop into 4
  df = prev_c * dnext_c
  dprev_c = f * dnext_c
  di = g * dnext_c
  dg = i * dnext_c

  # backprop into 3
  dai = sigmoid(ai) * (1 - sigmoid(ai)) * di
  daf = sigmoid(af) * (1 - sigmoid(af)) * df
  dao = sigmoid(ao) * (1 - sigmoid(ao)) * do
  dag = (1 - np.tanh(ag) ** 2) * dg

  # backprop into 2
  da = np.hstack((dai, daf, dao, dag))

  # backprop into 1
  db = np.sum(da, axis = 0)
  dprev_h = np.dot(Wh, da.T).T
  dWh = np.dot(prev_h.T, da)
  dx = np.dot(da, Wx.T)
  dWx = np.dot(x.T, da)

  return dx, dprev_h, dprev_c, dWx, dWh, db
```

Attention

Attention

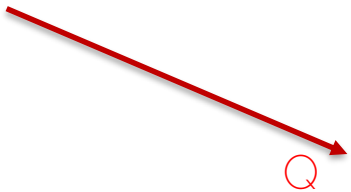
- Modality for attention
  - Self-attention
    - Attend to the input signal **itself**

Attention

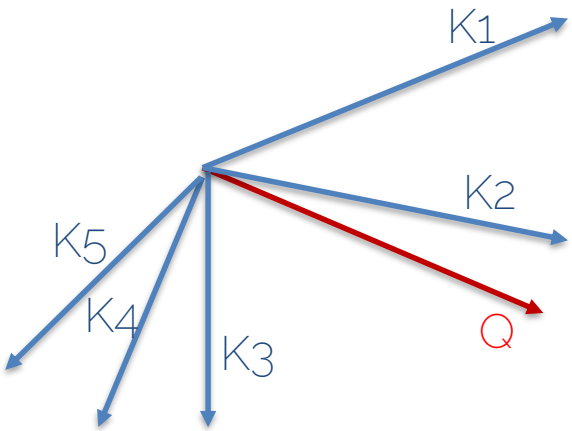


To train them well, divide by $\sqrt{d_k}$, "probably" because for large values of the key's dimension, the dot product grows large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

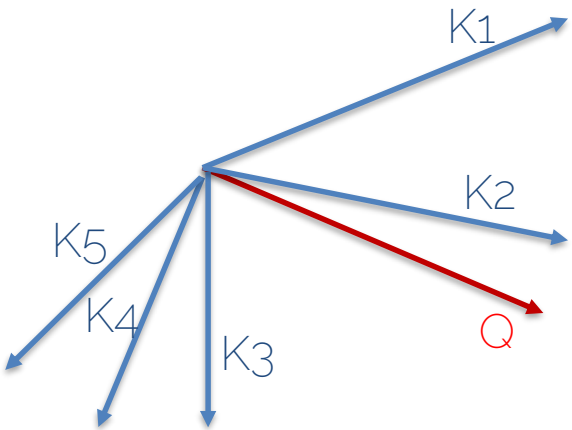
Attention



Attention

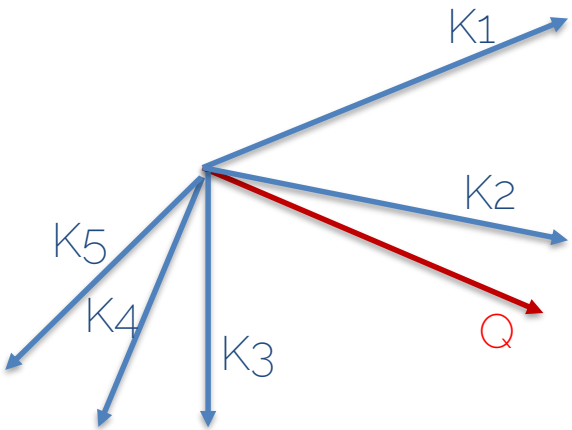


Attention



Values
V1
V2
V3
V4
V5

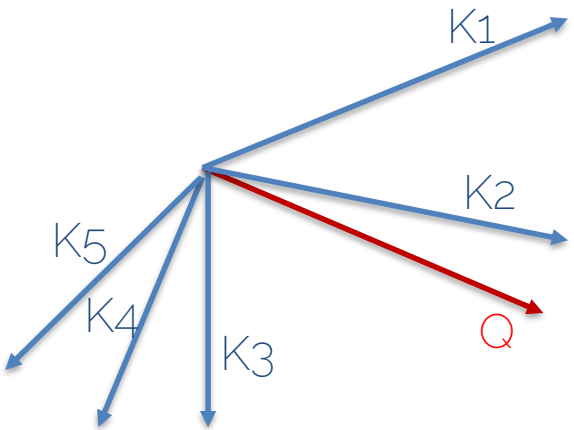
Attention



Values
V1
V2
V3
V4
V5

QK^T Dot product between (<Q,K1>), (<Q,K2>), (<Q,K3>), (<Q,K4>), (<Q,K5>).

Attention

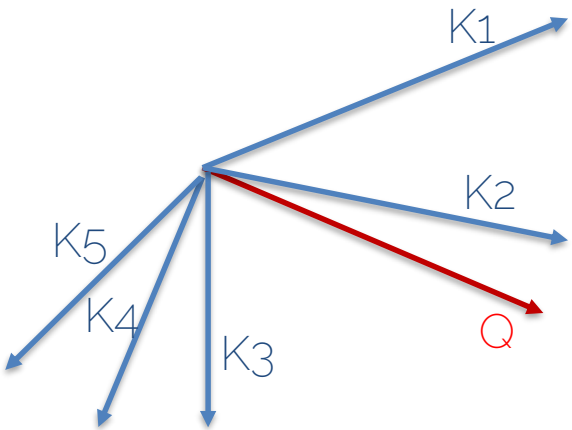


Values
V1
V2
V3
V4
V5

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Is simply inducing a distribution over the values.
The larger a value is, the higher is its softmax value.
Can be interpreted as a differentiable soft indexing.

Attention

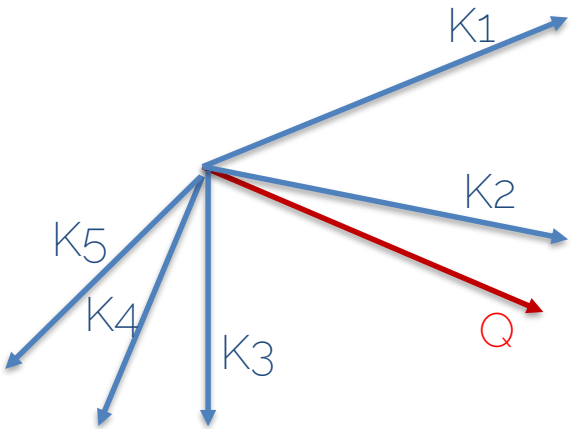


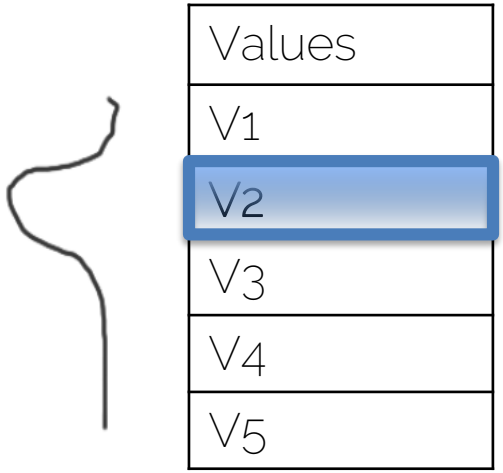
Values
V1
V2
V3
V4
V5

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Is simply inducing a distribution over the values.
The larger a value is, the higher is its softmax value.
Can be interpreted as a differentiable soft indexing.

Attention



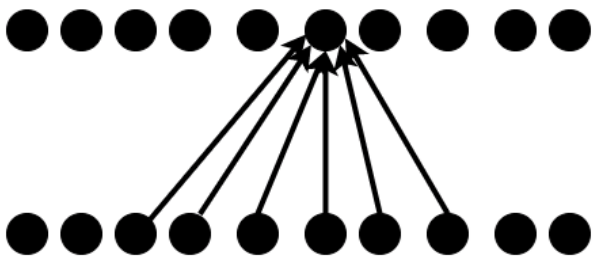


$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

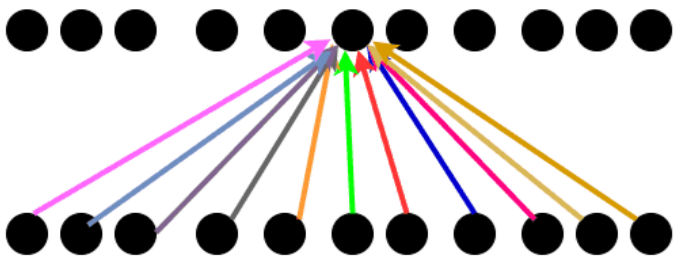
Selecting the value V where the network needs to attend..

Attention vs Convolution

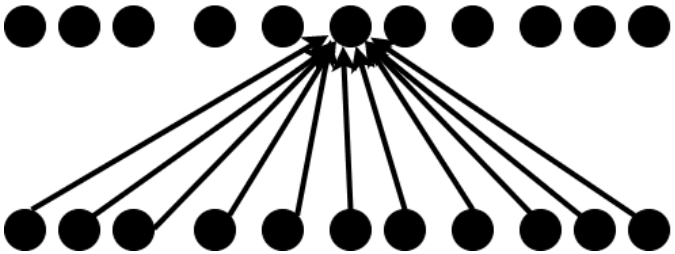
Convolution



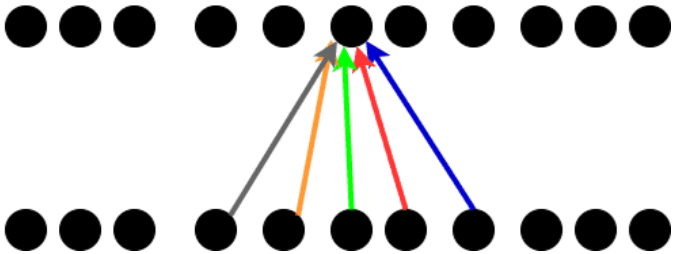
Global attention



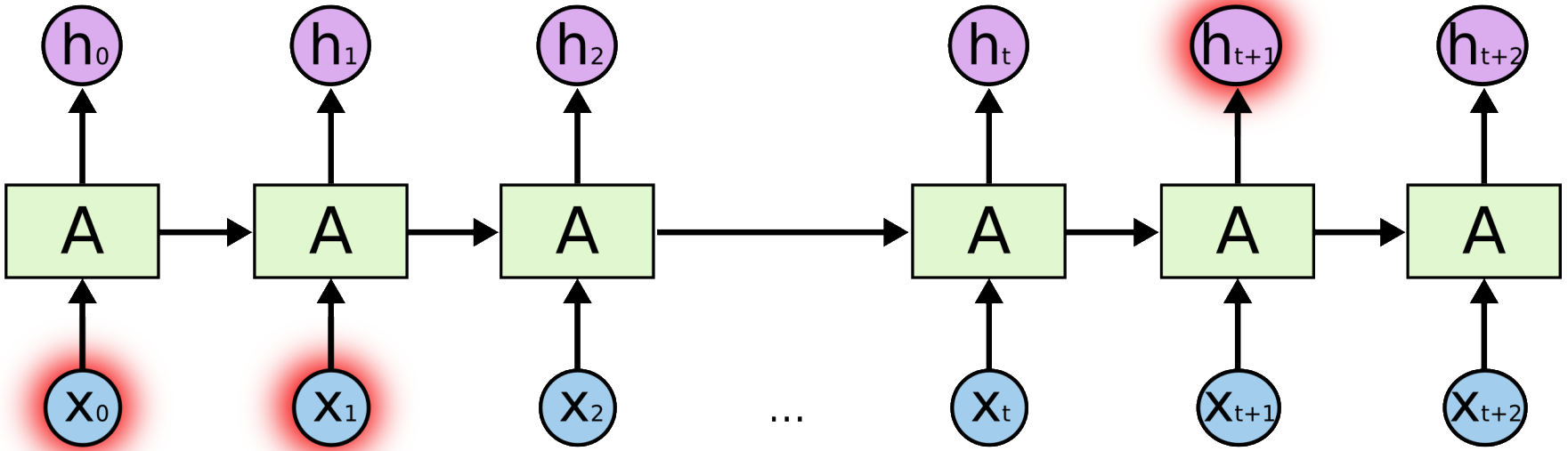
Fully Connected layer



Local attention



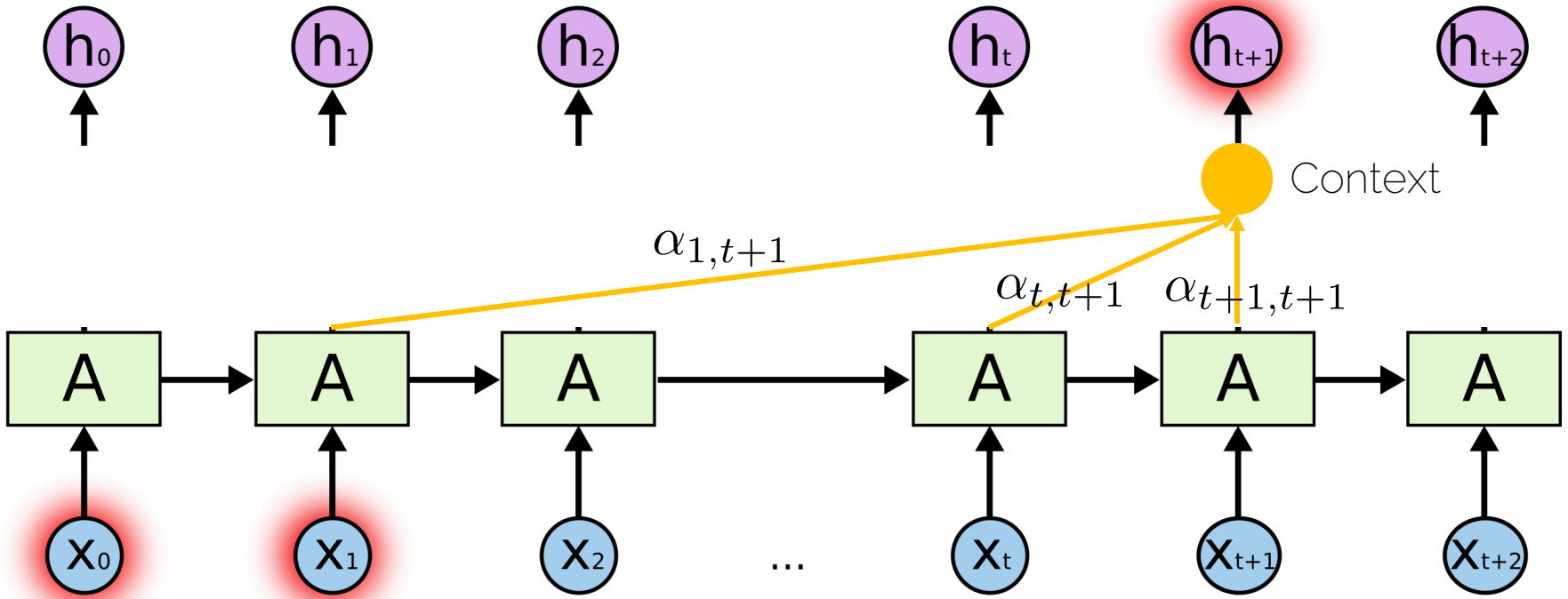
Long-Term Dependencies



I moved to Germany … so I speak German fluently.

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

Attention: Intuition

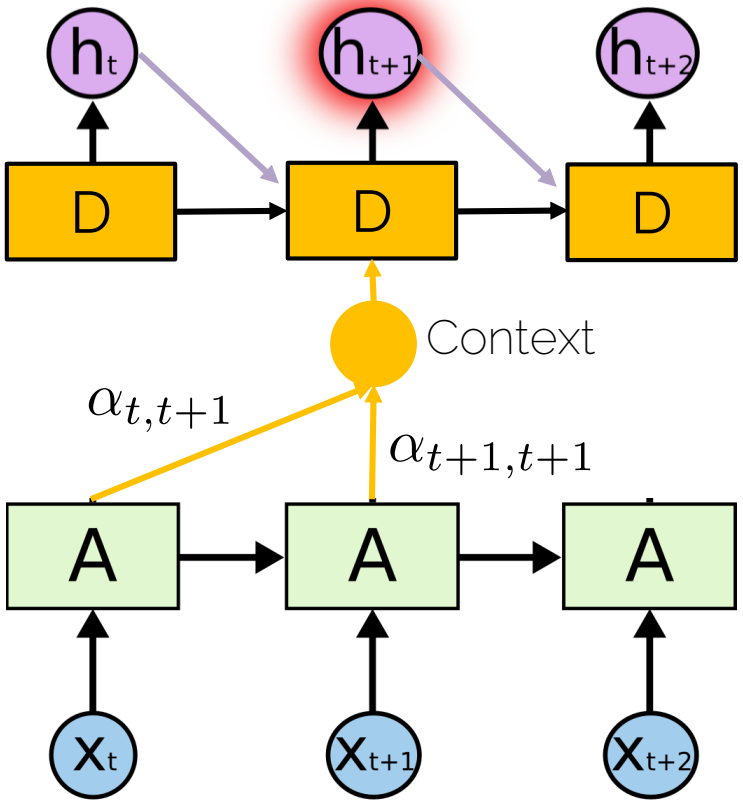


I moved to Germany ... so I speak German fluently

Attention: Architecture

- A decoder processes the information

- Decoders take as input:
  - Previous decoder hidden state
  - Previous output
  - Attention



Transformers

Attention is all you need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

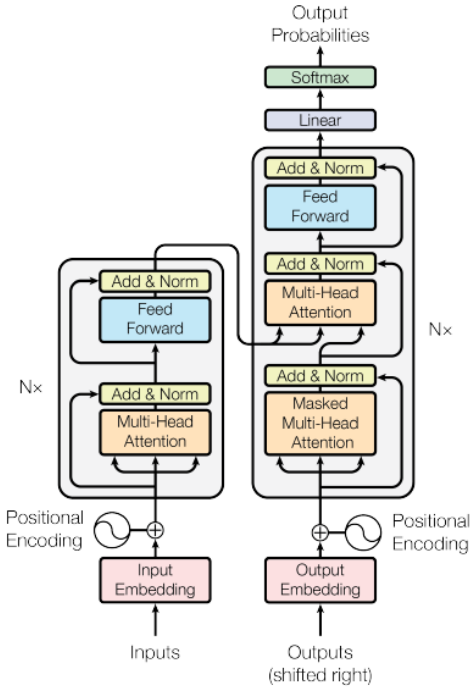
Llion Jones* Google Research llion@google.com

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

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers

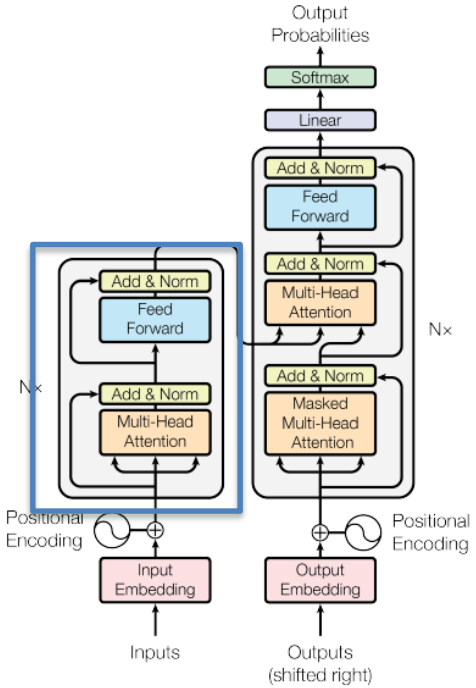


Not a single recurrent layer!

Vaswani et al. "Attention is All You Need".

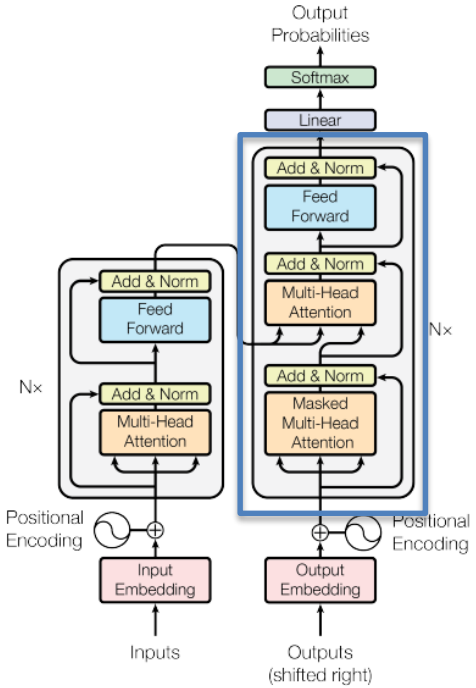
Transformers

Encoder



Vaswani et al. “Attention is All You Need”.

Transformers

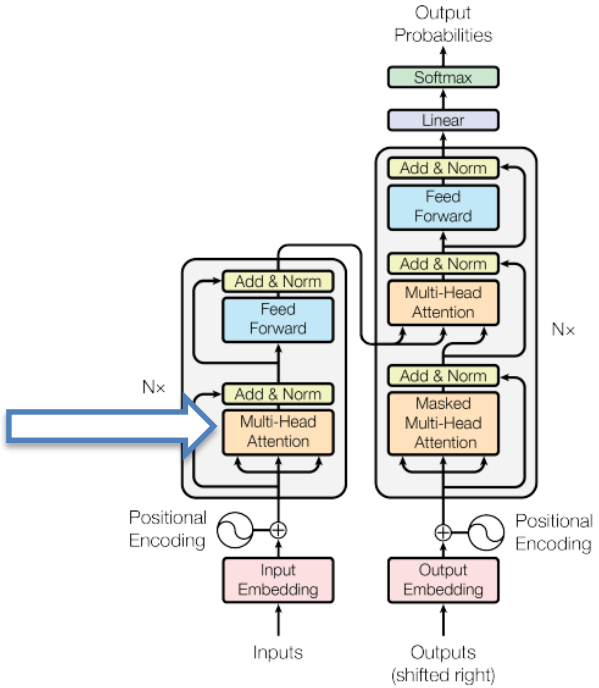


Decoder

Vaswani et al. "Attention is All You Need".

Transformers

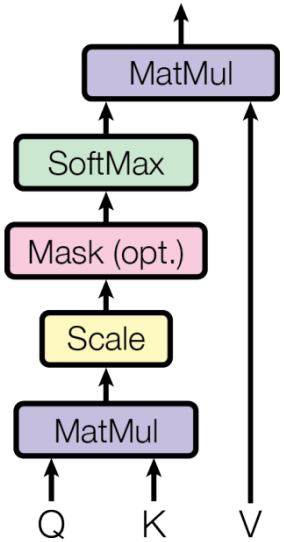
Core unit: scaled dot-product attention



Vaswani et al. "Attention is All You Need".

Transformers

- Scaled dot-product attention

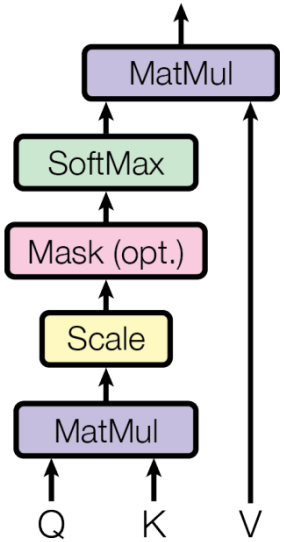


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Vaswani et al. "Attention is All You Need".

Transformers

- Scaled dot-product attention



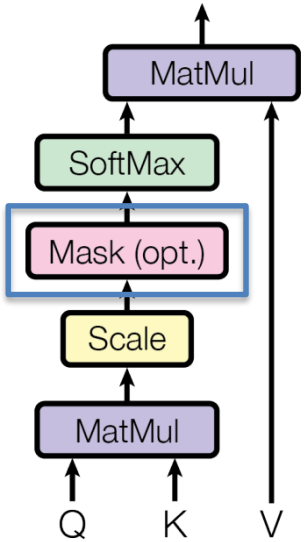
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Self-attention masks

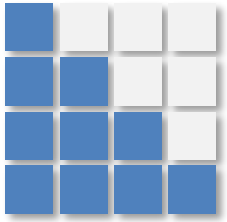
Vaswani et al. "Attention is All You Need".

Transformers

- Scaled dot-product attention



Triangular masking for unidirectional (left-to-right) modelling



attend

do not attend

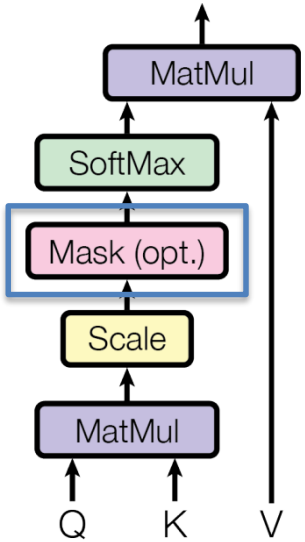
"German people speak German"

Only attend to the words **before it and itself**

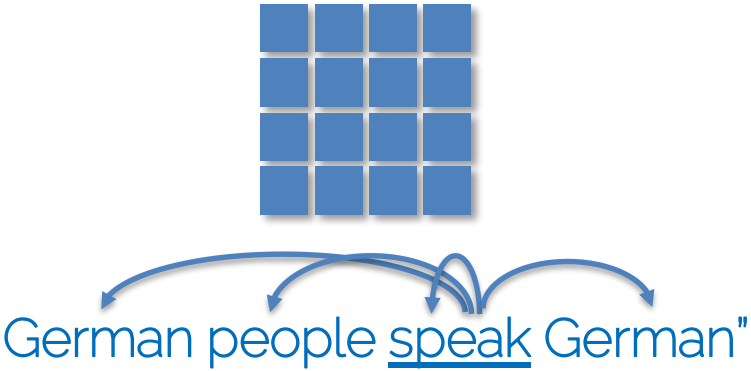
Vaswani et al. "Attention is All You Need".

Transformers

- Scaled dot-product attention



Full masking for bidirectional modelling



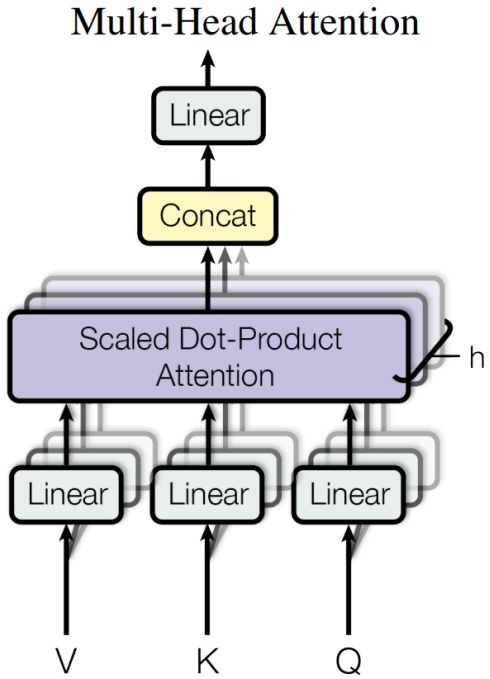
Attend to the words **before and after it, and itself**

Vaswani et al. "Attention is All You Need".

Transformers

- Multi-head Attention

Stack up the scaled dot-product attention module as attention heads



Different heads attend to different parts of the input signals

Vaswani et al. "Attention is All You Need".

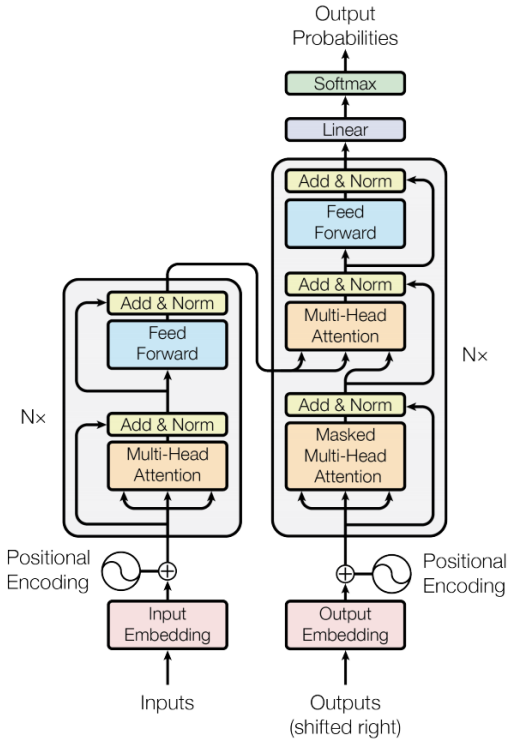
Positional Encoding

Uses fixed positional encoding based on trigonometric series, in order for the model to make use of the order of the sequence

dimension

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

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



Self-attention: complexity

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

where n is the sequence length, d is the representation dimension,
k is the convolutional kernel size, and r is the size of the neighborhood.

Self-attention: complexity

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

where n is the sequence length, d is the representation dimension,
k is the convolutional kernel size, and r is the size of the neighborhood.

Considering that most sentences have a smaller dimension than the representation dimension (in the paper, it is 512), self-attention is very efficient.

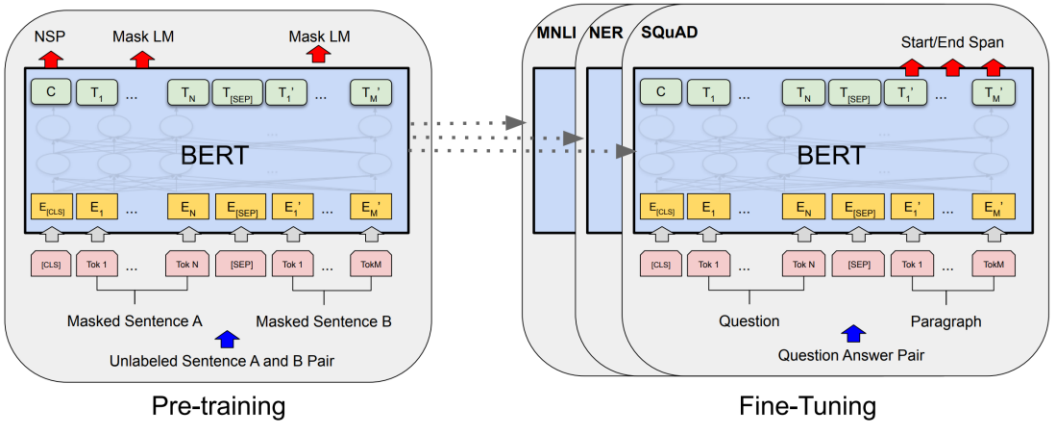
Transformers

- Significantly improved SOTA in Machine Translation
- Launched a transformer revolution in the NLP field
- Foundation of large NLP models like BERT (Google) and GPT-3, ChatGPT (OpenAI), Claude, Gemini!
- Transformers made its way to computer vision.

Language Transformer Architectures

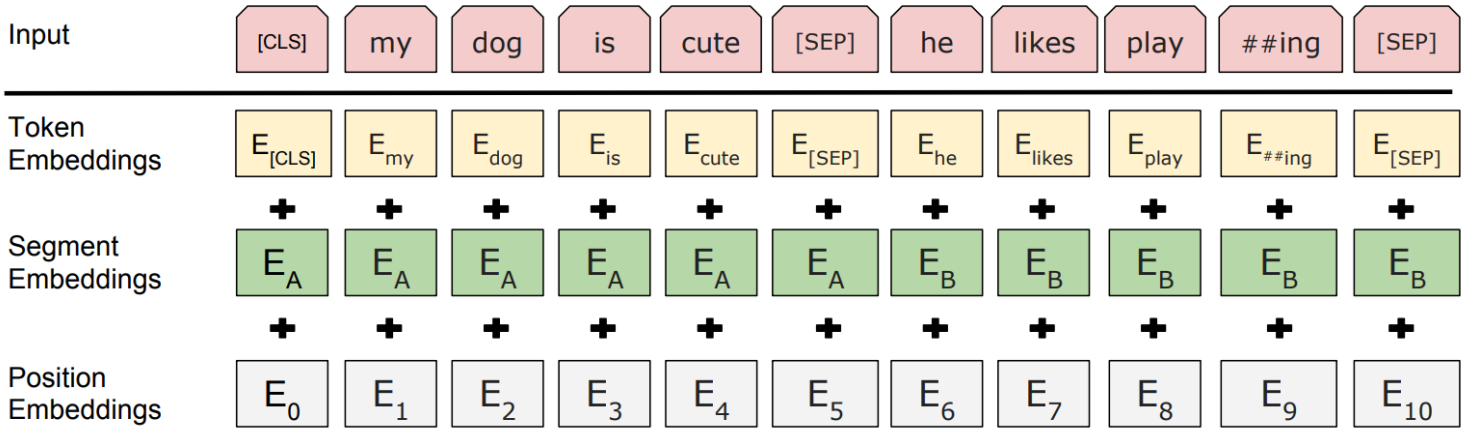
BERT

- Bidirectional Encoder Representations from Transformers
  - A big transformer as a text encoder



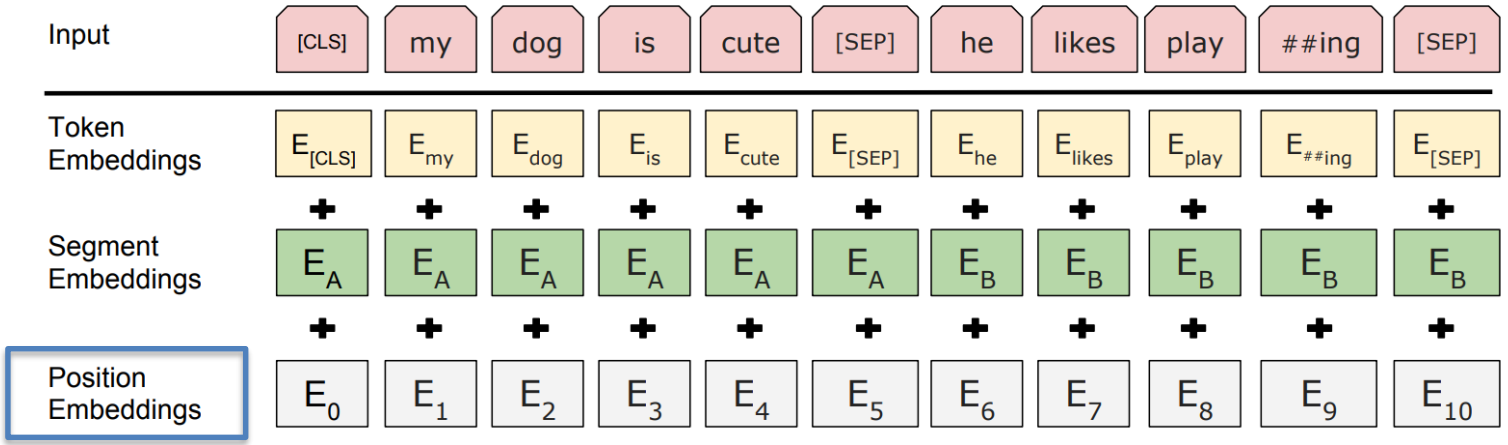
BERT

- BERT Input Representations – Three embeddings



BERT

- BERT Input Representations

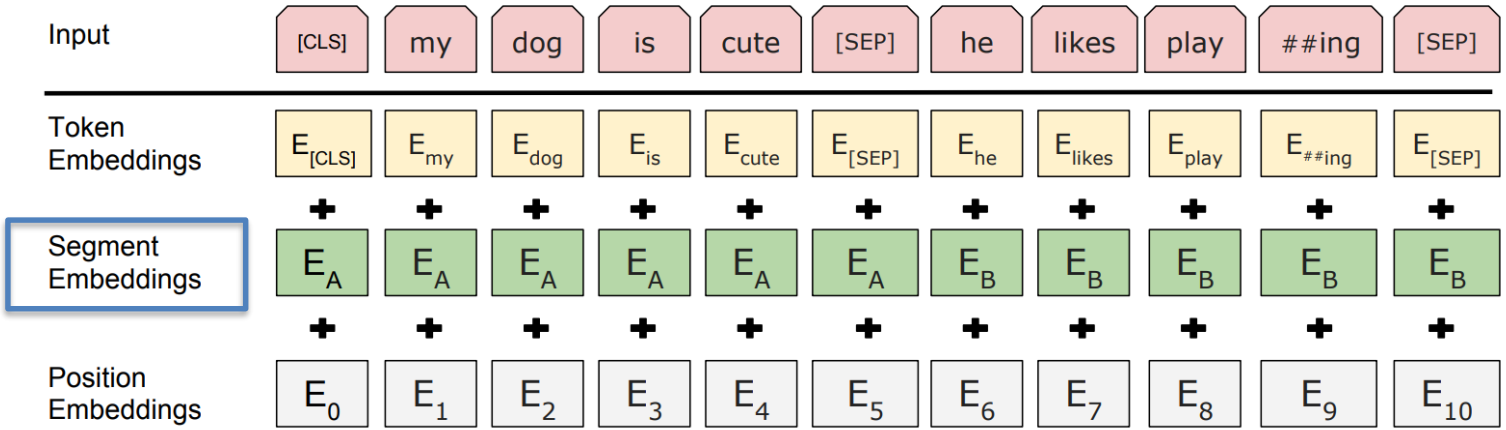


To indicate the positions in the sequence

 Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding".

BERT

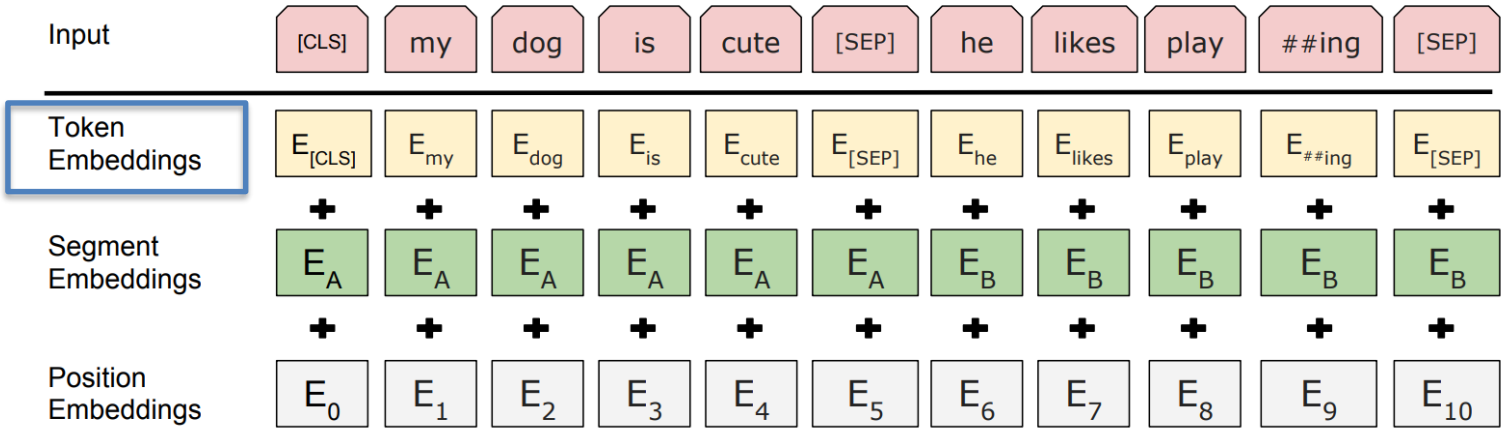
- BERT Input Representations



To indicate the sentence A or B

BERT

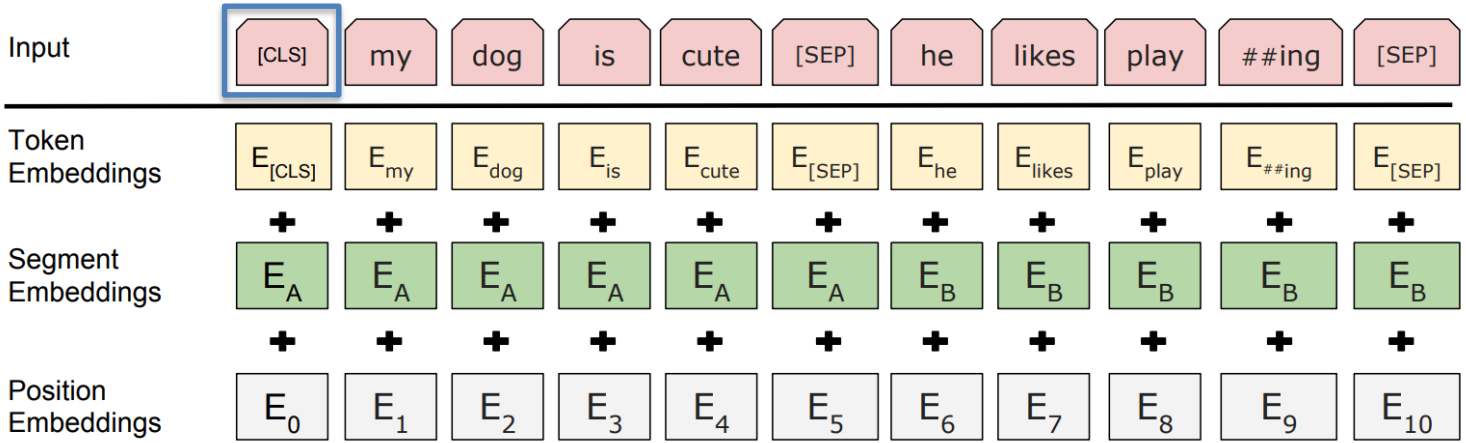
- BERT Input Representations



Word embeddings

BERT

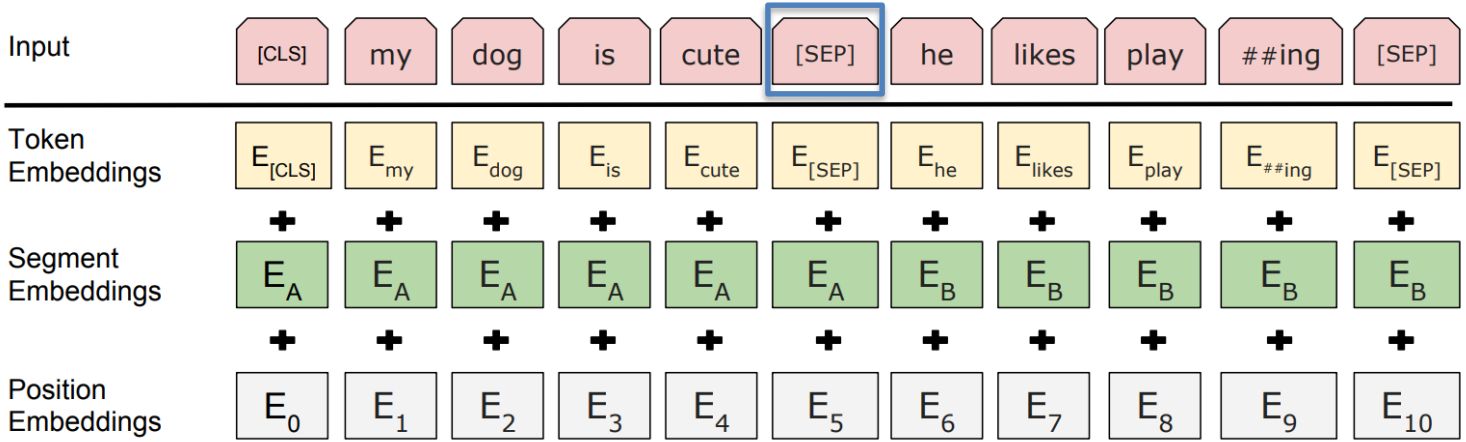
- BERT Input Representations



Learnable token for Next Sentence Prediction

BERT

- BERT Input Representations



Indicate the end of the sentence(s)

 Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding".

BERT

- Pre-training Objectives
  - Two unsupervised tasks
    - Masked Language Modelling (MLM)
    - Next Sentence Prediction (NSP)
  - No human annotation needed!

BERT

- Masked Language Modelling (MLM)
  - Key idea:
    - Randomly mask out some words from the input
    - Predict the masked words with the context from the input itself
    - Enforce the network to learn the word-level context

BERT

- Masked Language Modelling (MLM)

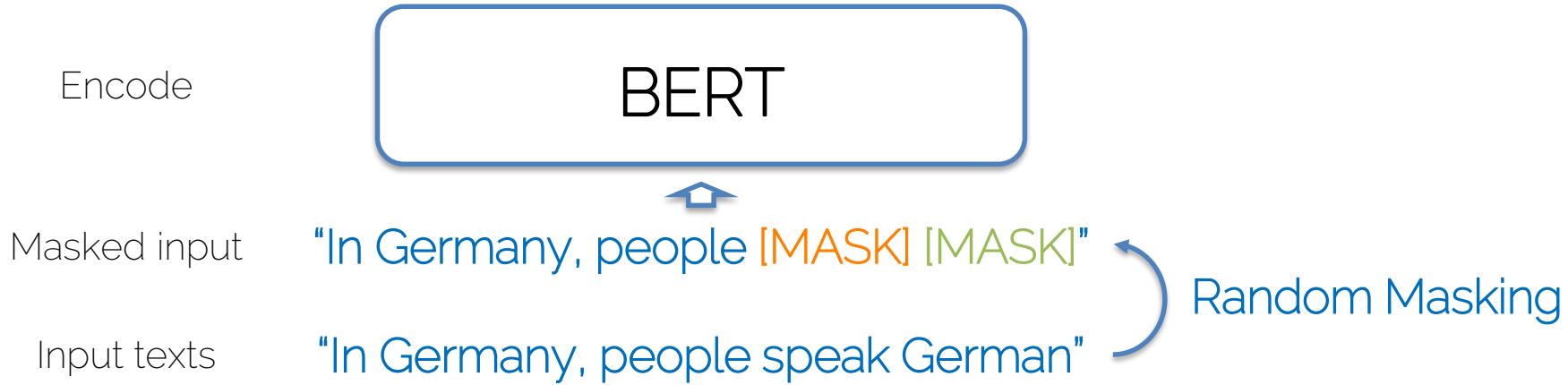
Input texts "In Germany, people speak German"

BERT

- Masked Language Modelling (MLM)

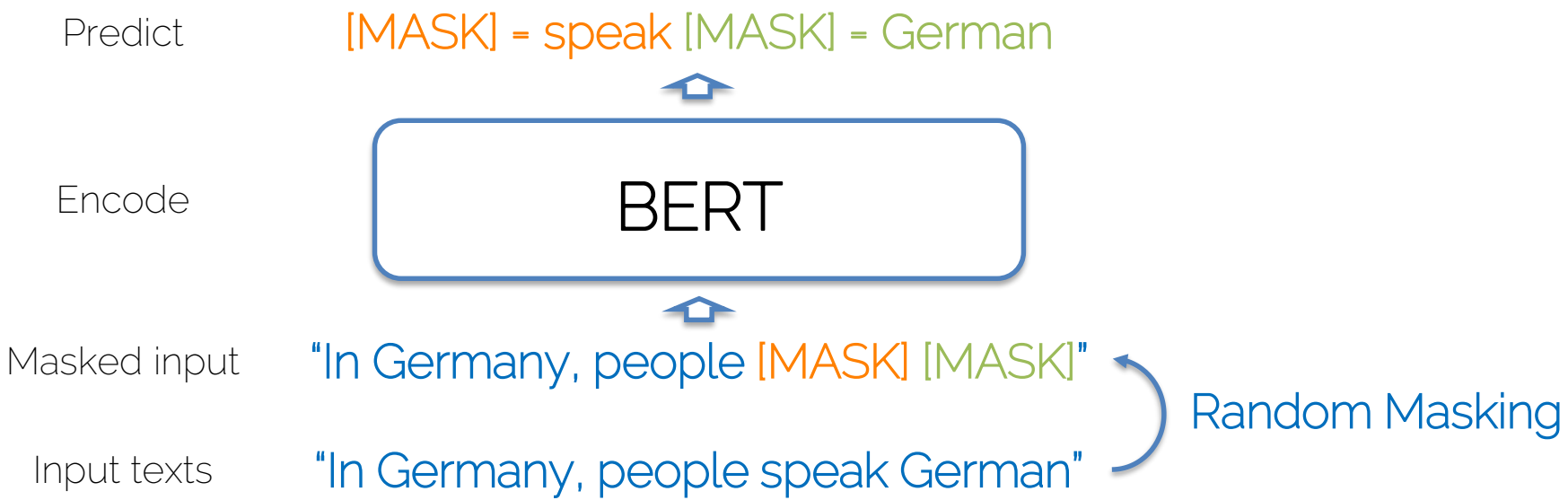
BERT

- Masked Language Modelling (MLM)



BERT

- Masked Language Modelling (MLM)



BERT

- Next Sentence Prediction (NSP)
  - Key idea:
    - Take two sentence A and B from the dataset
    - 50% of the time B is the actual next sentence of B, another 50% of the time B is randomly sampled from the dataset
    - Predict if B is the next sentence of A
    - Enforce the network to learn the sentence-level context

BERT

- Next Sentence Prediction (NSP)

A sample from the dataset

"In Germany, people speak German.
So German is the native language of the German people."

Break up

Sentence A "In Germany, people speak German. "

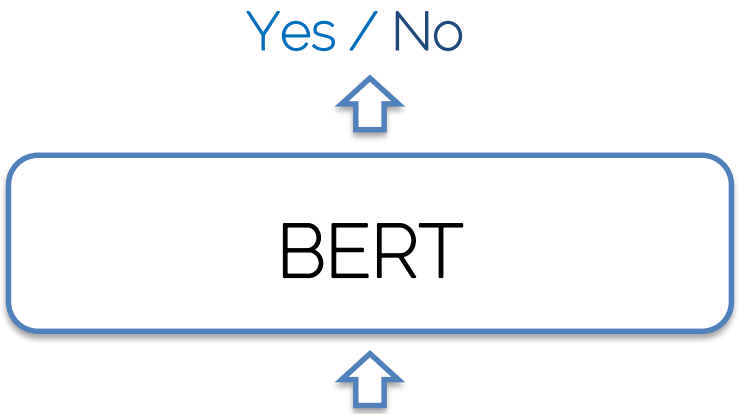
Sentence B "So German is the native language of the German people."

"Today is Wednesday."

Sentence B'
(another sentence from the dataset)

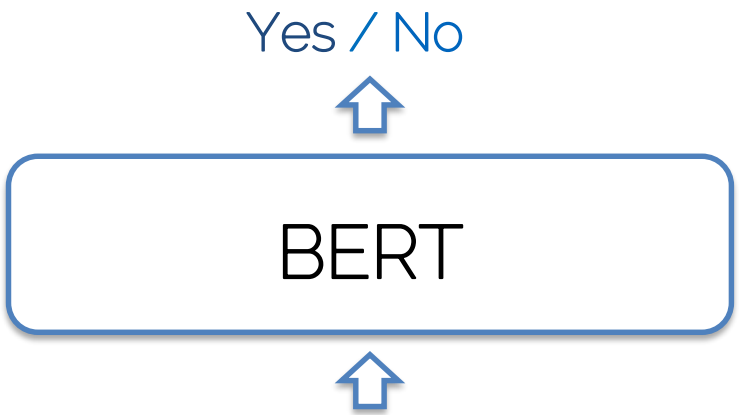
BERT

- Next Sentence Prediction (NSP)



BERT

- Next Sentence Prediction (NSP)

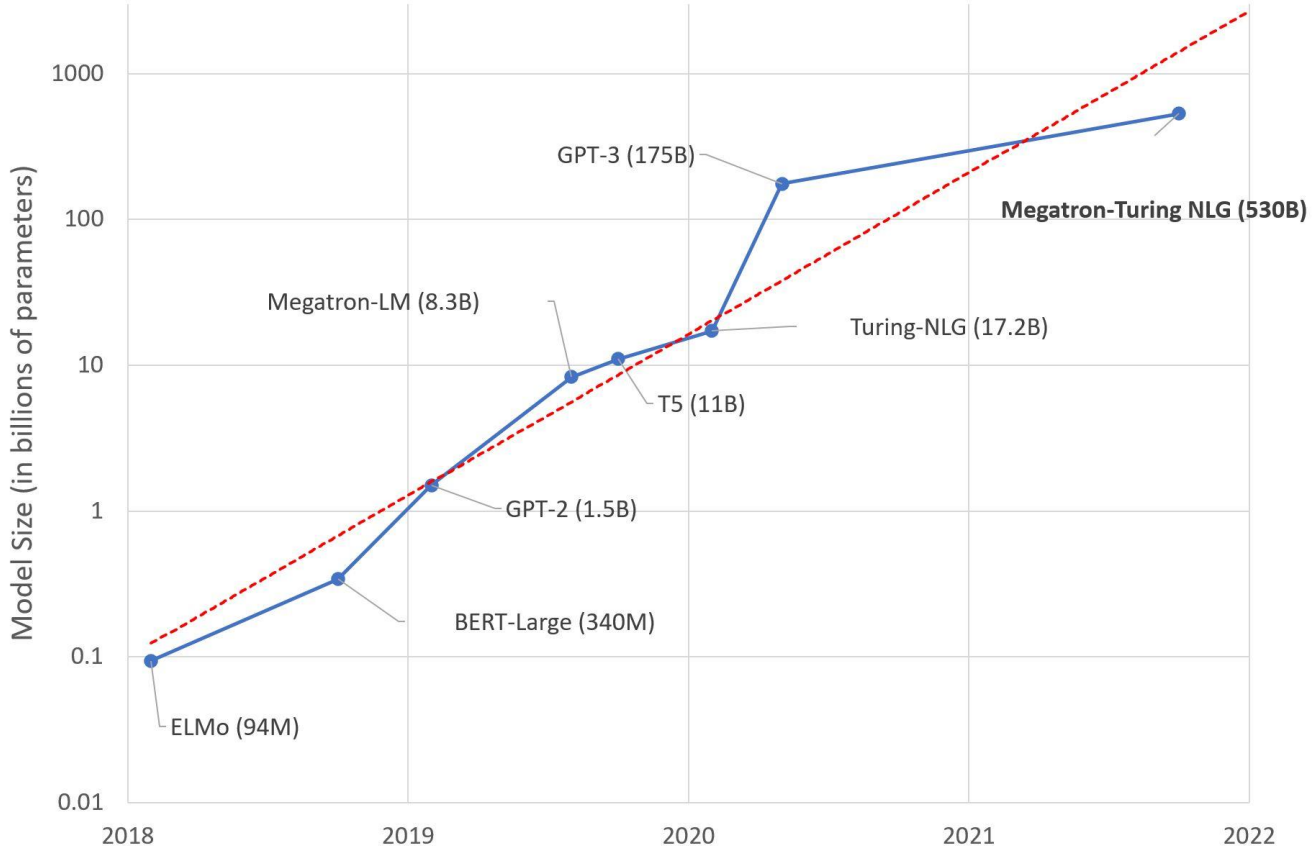


BERT

- Pre-training with two self-supervised objectives, then fine-tuned on downstream tasks
- No human annotations needed for pre-training! -> can be pre-trained on large-scale datasets, even those ones that have not been annotated, e.g., web documents.
- VERY BIG (back then in 2018)! – $\text{BERT}_{\text{large}}$ has 340M parameters
- Masked Language Modelling has a huge impact for representation learning, even for Computer Vision!

Language Model Sizes

GPT-4 1.7 trillion



How to build your GPT Model?

GPT: Generative Pre-trained Transformer

- Tokenization / Dictionary
- Sequence model (transformer)
- Training scheme (context etc.)

Dictionary

- 1 char == 1 token
- 1 word == 1 token
- Optimize for dictionary
- Learn embedding / token (e.g., WordNet)

Dictionary: Byte Pair Encoding

```
aaabdaaabac
```

The byte pair "aa" occurs most often, so it will be replaced by a byte that is not used in the data, such as "Z". Now there is the following data and replacement table:

```
ZabdZabac
Z=aa
```

Then the process is repeated with byte pair "ab", replacing it with "Y":

```
ZYdZYac
Y=ab
Z=aa
```

The only literal byte pair left occurs only once, and the encoding might stop here. Alternatively, the process could continue with recursive byte pair encoding, replacing "ZY" with "X":

```
XdXac
X=ZY
Y=ab
Z=aa
```

Dictionary size: of GPT-3.5 and GPT-4
https://platform.openai.com/tokenizer

https://en.m.wikipedia.org/wiki/Byte_pair_encoding

Context

- **Llama: 2K**
- **Llama 2: 4K**
- **GPT-3.5-turbo: 4K.** However, GPT-3.5-16k has a context length of 16K.
- **GPT-4: 8K**. Similarly, GPT-4-32k has a context window of up to 32K.
- **Mistral 7B: 8K**
- **Palm-2: 8k**. However, Google has reported their new Gemini multi-modal model as having a 32K
- **Claude: 9K**
- **Claude 2: 100,000K** (in beta stage at the time of writing).

Common trick to improve context: fine-tune with LoRA variant

Coding models have up to 1mio+ but there are hacks…

https://symbl.ai/developers/blog/guide-to-context-in-llms/

Training Scheme

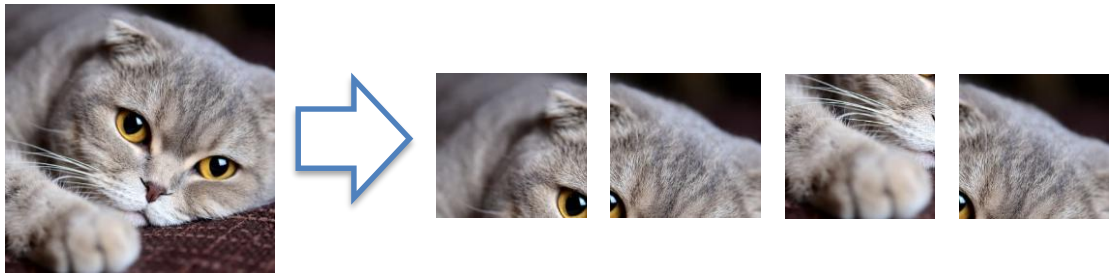
- Loss is usually a variant of cross entropy
- Multi-modal models
- Scale, scale, scale…

Transformers in Computer Vision

Vision Transformers (ViTs)

- CNNs can be computationally demanding and require a great amount of design tricks and efforts

- Images can be modelled as sequences of patches
  - Vision Transformers (ViTs)



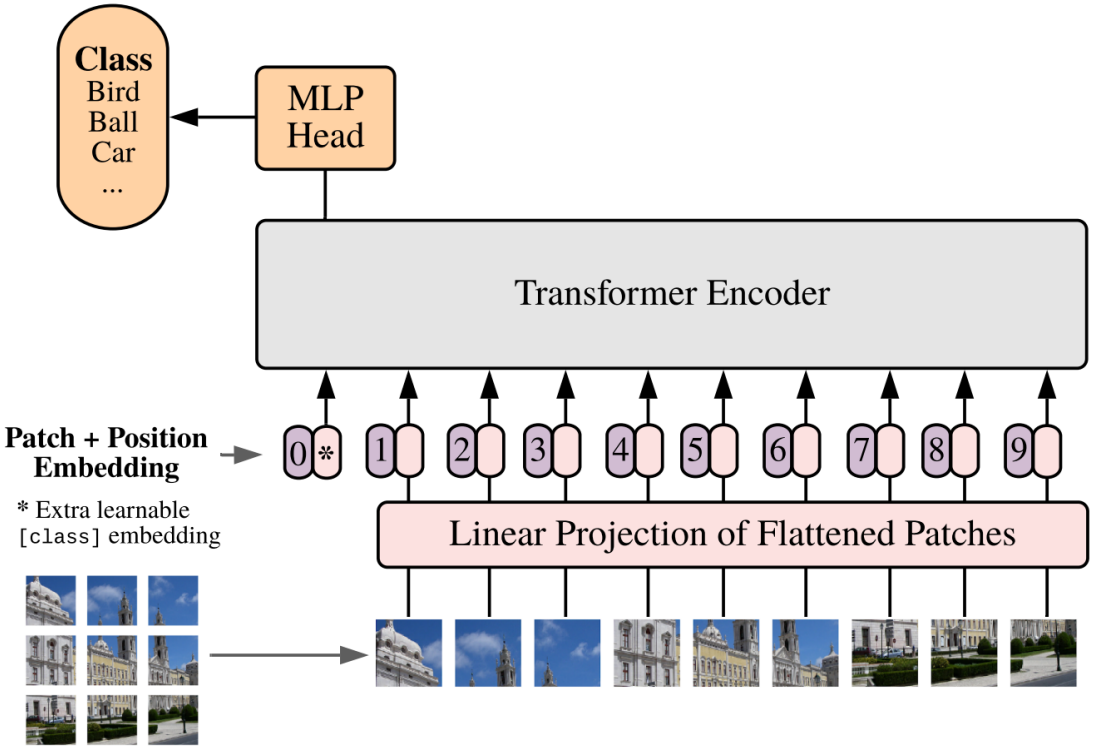
Input image

Image patches

Vision Transformers (ViTs)

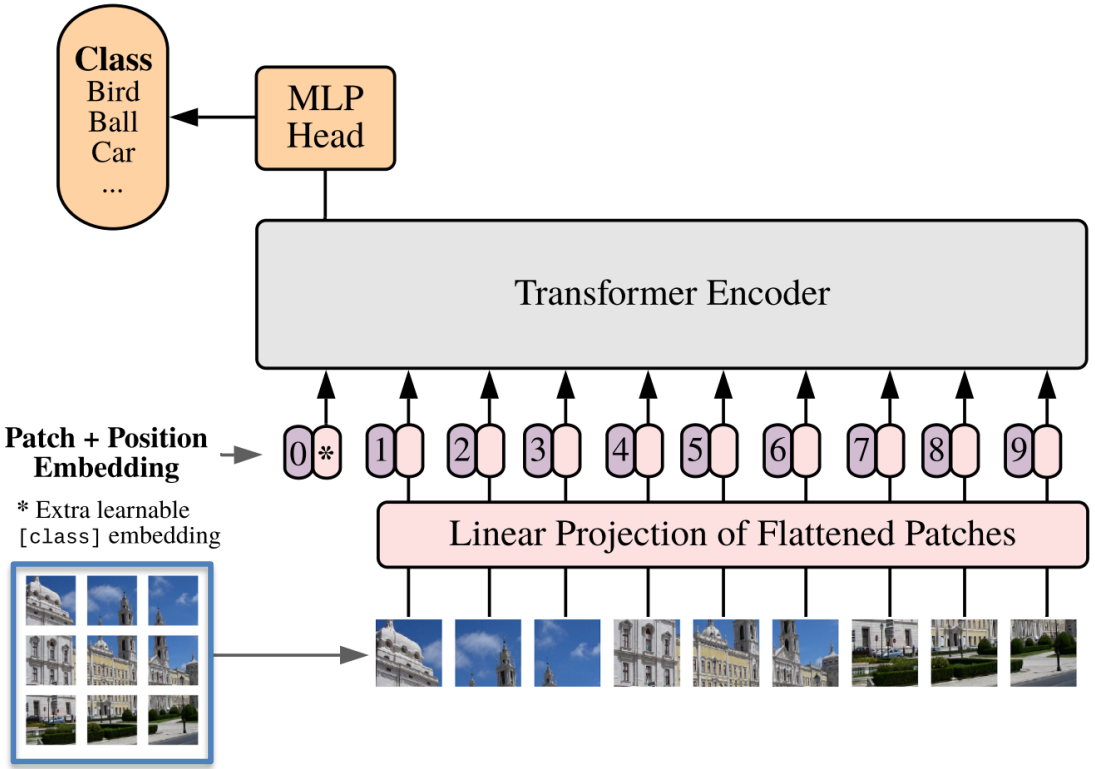
https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html

Vision Transformers (ViTs)



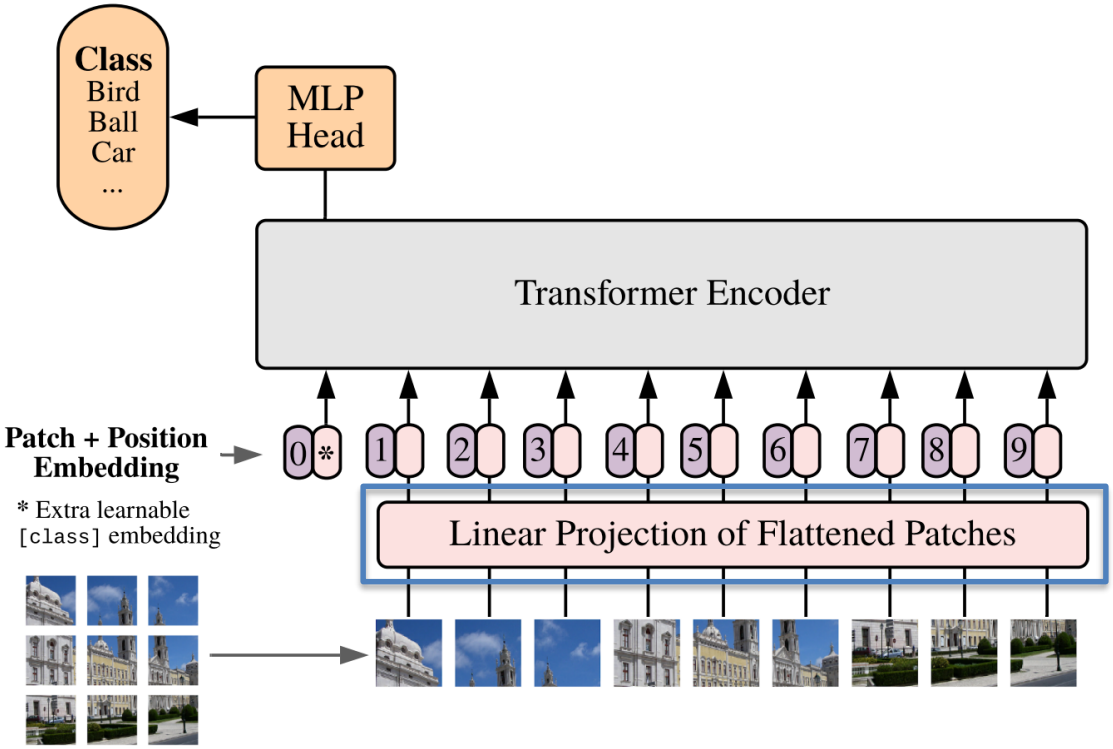
Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

Vision Transformers (ViTs)



Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

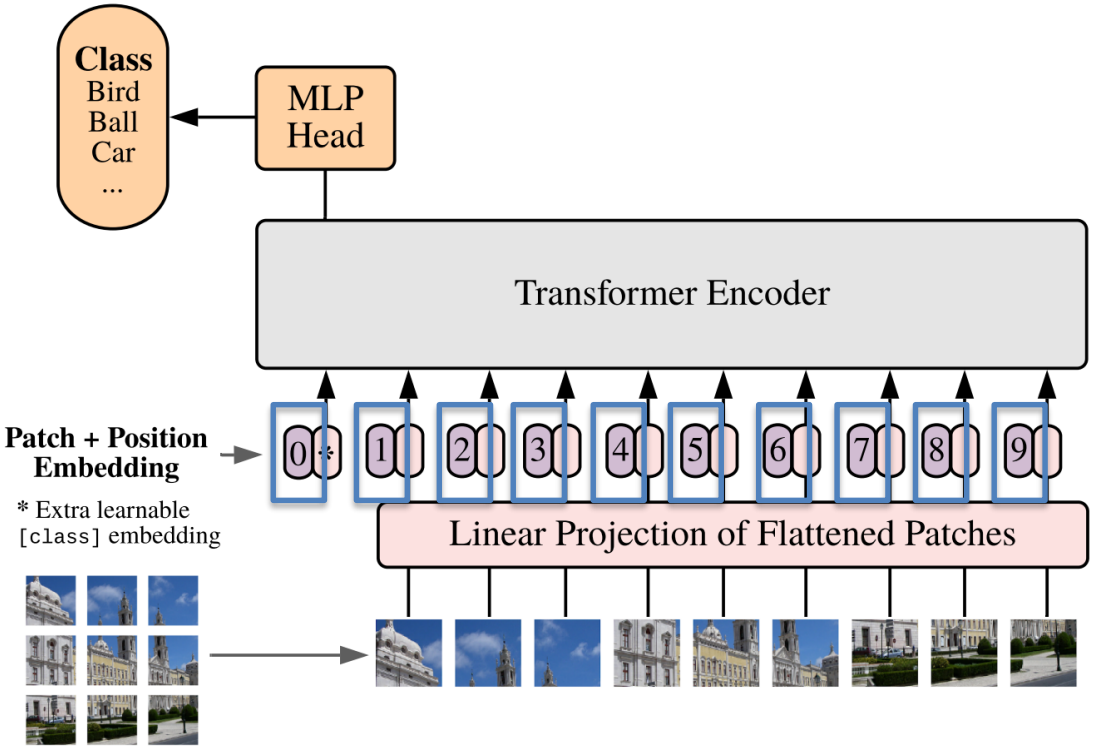
Vision Transformers (ViTs)



MLP (can be replaced by a tiny CNN)

Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

Vision Transformers (ViTs)

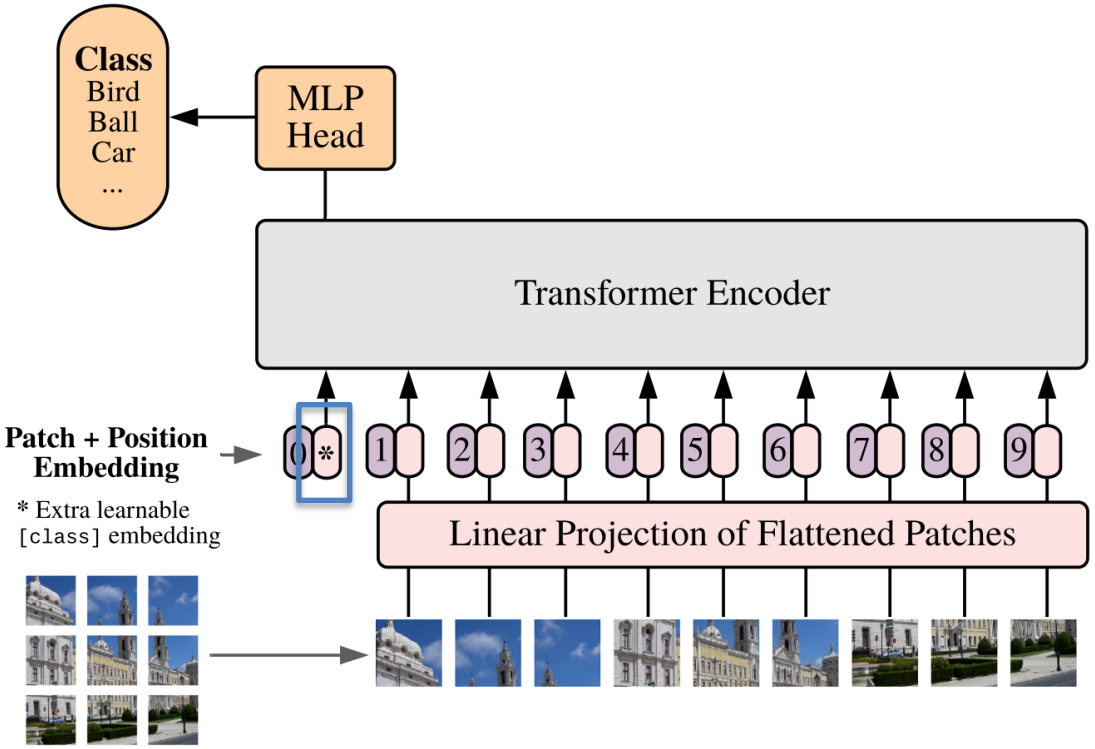


Indicate the positions of the patches

Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

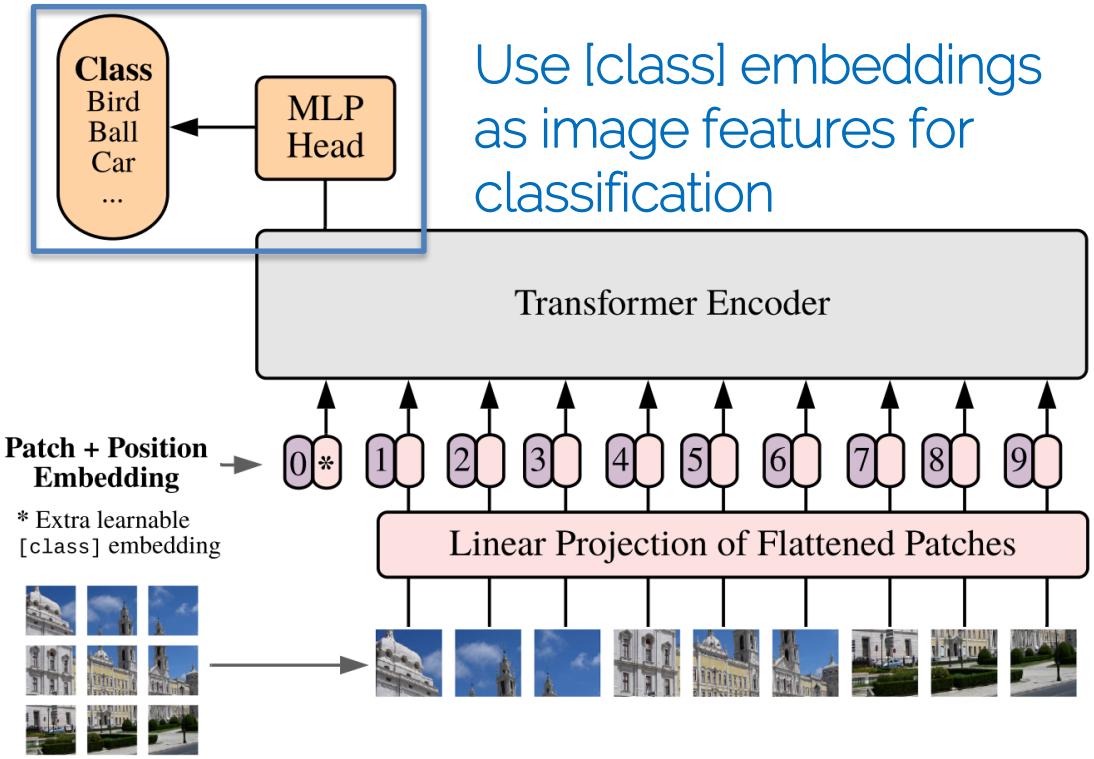
Vision Transformers (ViTs)

Learnable special token (similar to [CLS] in BERT)



Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

Vision Transformers (ViTs)



Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

Vision Transformers (ViTs)

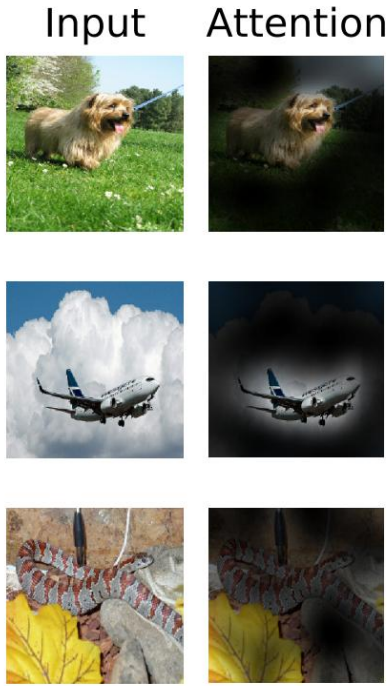
- Pre-trained on several big datasets
- Perform transfer learning on target datasets/benchmarks (freeze the pre-trained transformer backbone, fine-tune the classifier only)
- Outperform the ResNet baseline with substantially less computational resources for pre-training

Vision Transformers (ViTs)

- A closer look at ViTs
  - Transformers in language can give us the attention maps on the input words
  - Can ViTs provide the attention maps on the input image patches? -> YES!

Vision Transformers (ViTs)

- Attention maps in ViTs



Attention maps on the input image while computing the attended [class] embedding for classification

i.e., to classify an image, ViTs give us a hint for which part is most relevant to the predicted label.

Dosovitskiy et al. "AN IMAGE IS WORTH 16X16 WORDS".

Vision Transformers (ViTs)

- ViTs set a new form for image recognition
- ViTs are extremely powerful at representing image features
- ViTs are also applied in many other domains, such as representation learning (e.g., DINO, MoCo, CLIP, etc.), object detection, and multimodal learning.

See you next time!