Lecture 8: Memory, Attention and Self-Attention

“Neuralizing” Existing Algorithms

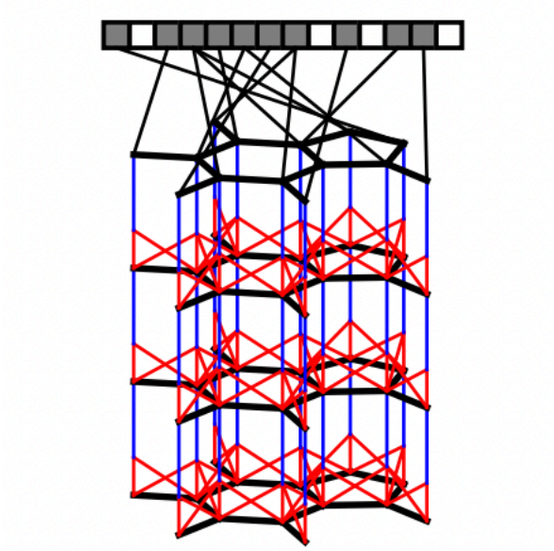
- A common approach in deep learning architecture design for various problems is to start with existing approaches or algorithms and add parameters + make them differentiable

- Non-differentiable components don’t allow us to provide learning signal to the model parameters

“Neuralizing” Existing Algorithms

Example 1

Original feature construction algorithm used for representing molecules



Algorithm 1 Circular fingerprints

Input: molecule, radius R, fingerprint length S
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ ▷ concatenate
    $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ ▷ hash function
    $i \leftarrow \text{mod}(r_a, S)$ ▷ convert to index
    $\mathbf{f}_i \leftarrow 1$ ▷ Write 1 at index
Return: binary vector $\mathbf{f}$

“Neuralized” Algorithm = Network “Architecture”/ Design inspired by hand crafted algorithm

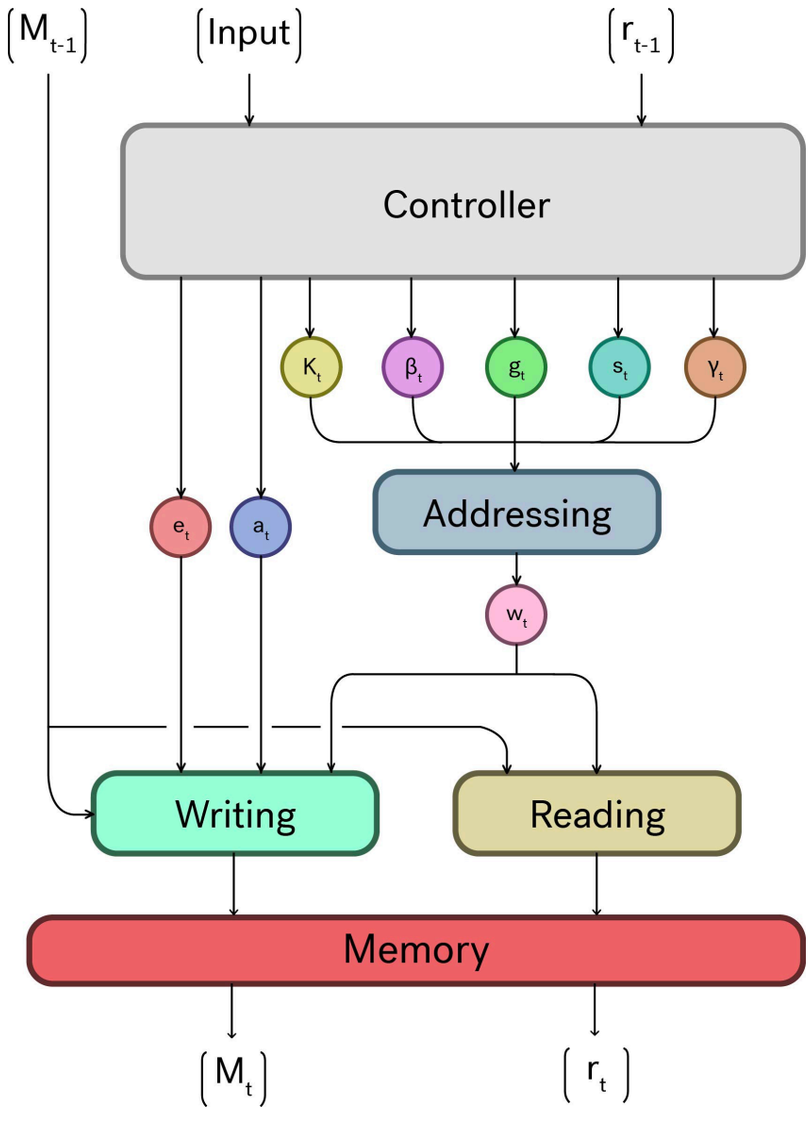
Algorithm 2 Neural graph fingerprints

Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

→ Neural Net

“Neuralizing” Existing Algorithms

Example 2

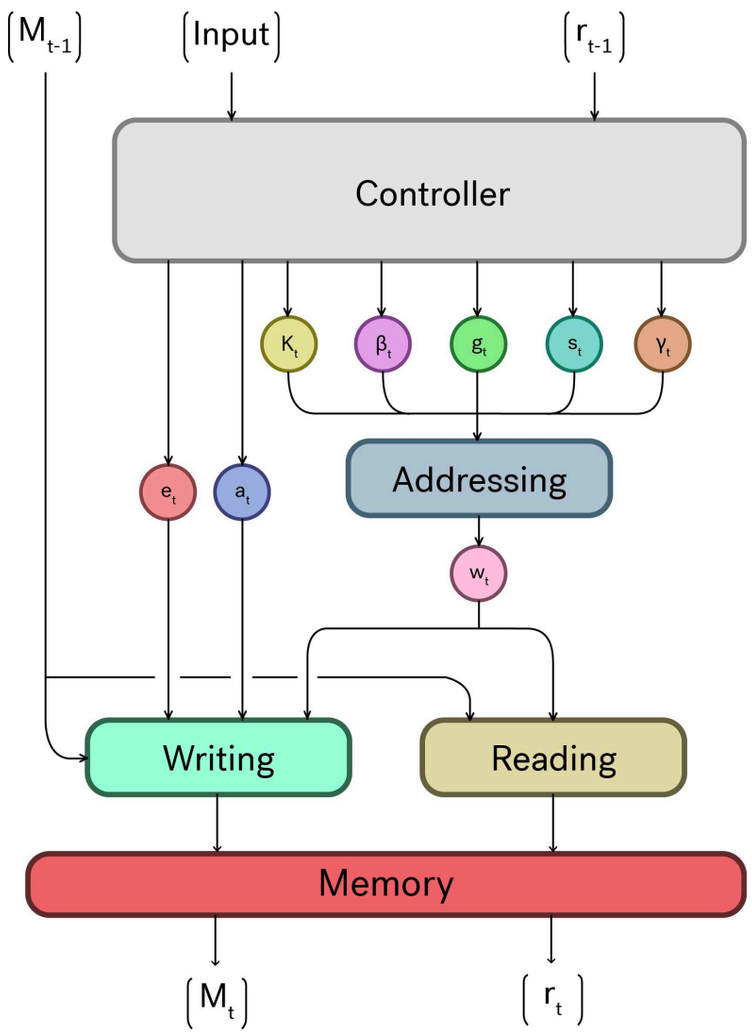


- To implement many useful algorithms we require working memory and sequential operations that read and write
- e.g. we want to try to learn a new kind of sorting algorithm

Read and Write Operations

Example 2

- We take a very basic design von Neumann architecture design from computer architecture
- Add parameters
- Make all components of this model differentiable



Normal Read:

i <- Controller

$$r_i = M_t[i]$$

Controller has no parameters + read is a discrete (hard) indexing

Differentiable + Parameterized Read:

Controller = NN(parameters)

a = NN(parameters)

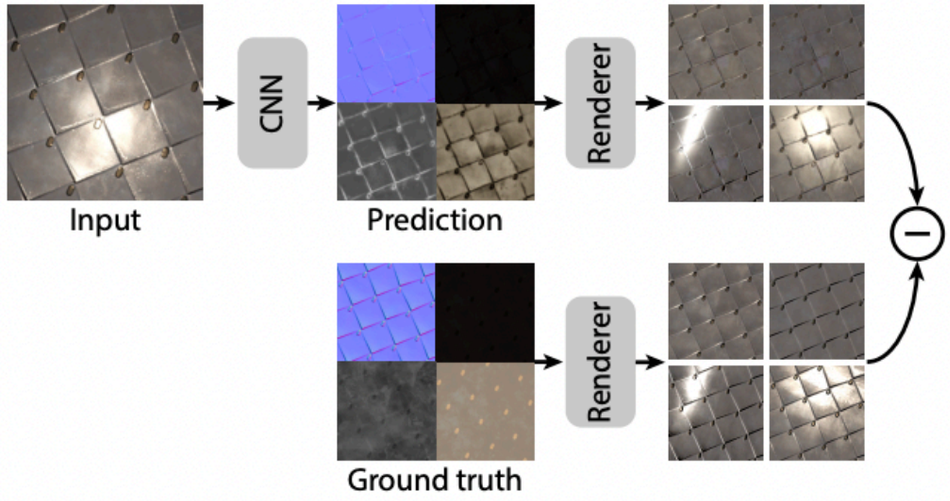
$w = softmax(a)$ → 0.02, 0.90, 0.05, 0.01, 0.02

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i),$$

Controller has parameters + read is a soft (differentiable) indexing

Making Things Differentiable

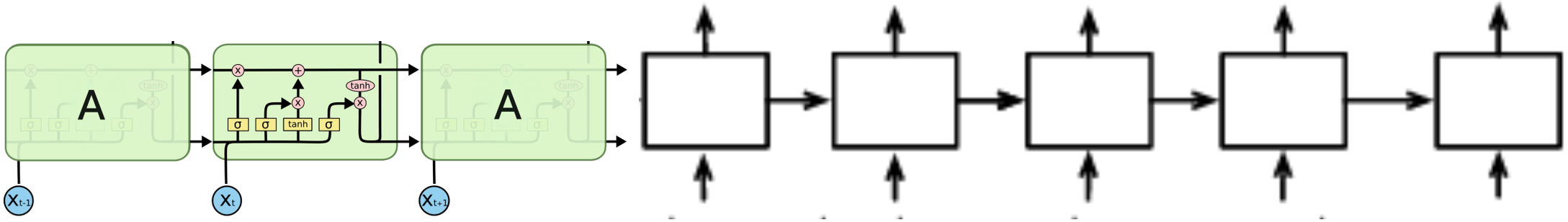
- In general making differentiable approximations of useful non-differentiable operations can be very useful in a number of contexts
  - Allows model to use operations suited to task
  - In some case can allow to directly connect to desired loss criteria
- Examples
  - Differentiable sorting
  - Differentiable QP solvers
  - Differentiable Renderers
  - GradSLAM — from your TA Soroush

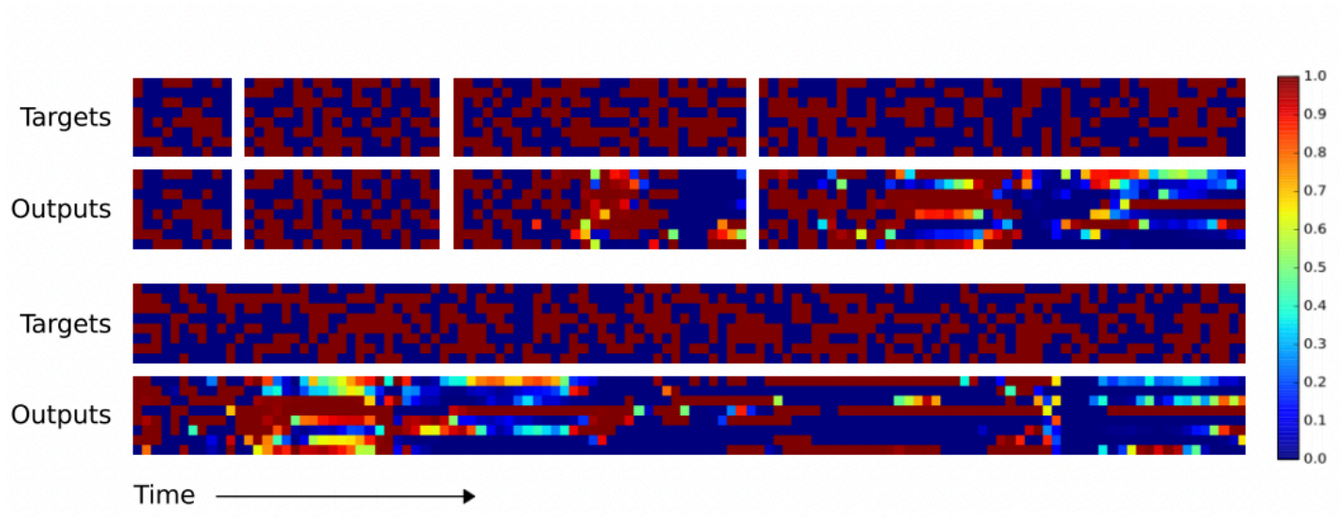


Differentiable renderer allows to learn models that provide good inputs to the renderer

LSTM Issue Review

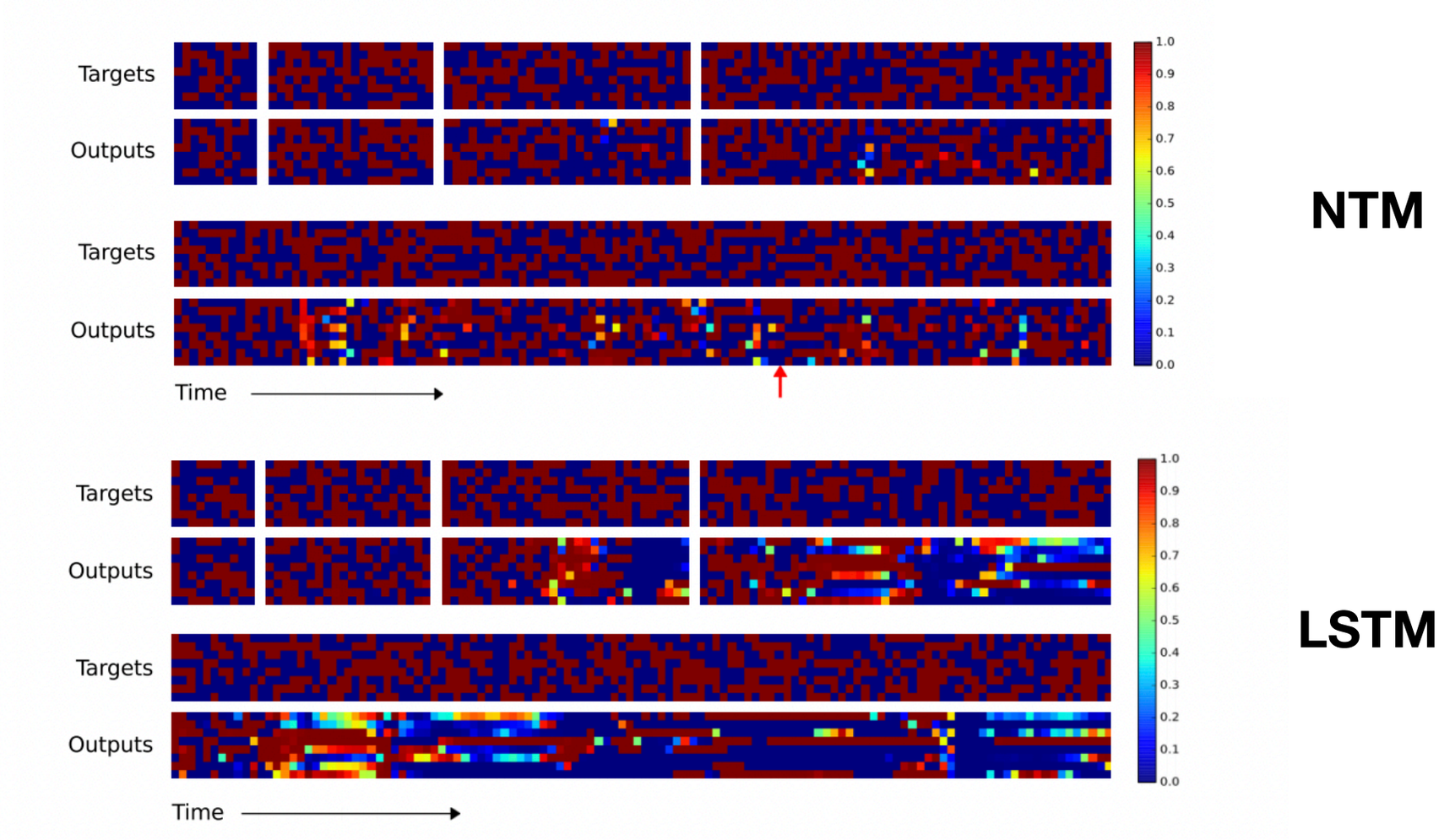
- Greatly improves vanishing gradient issues
- Still difficult to integrate long term information due to fixed and small size state vectors
- Gating mechanisms at time t are unaware what information might serve later steps best
- LSTM fails at simple copy tasks when input sequence is long





NTM on Copy Tasks

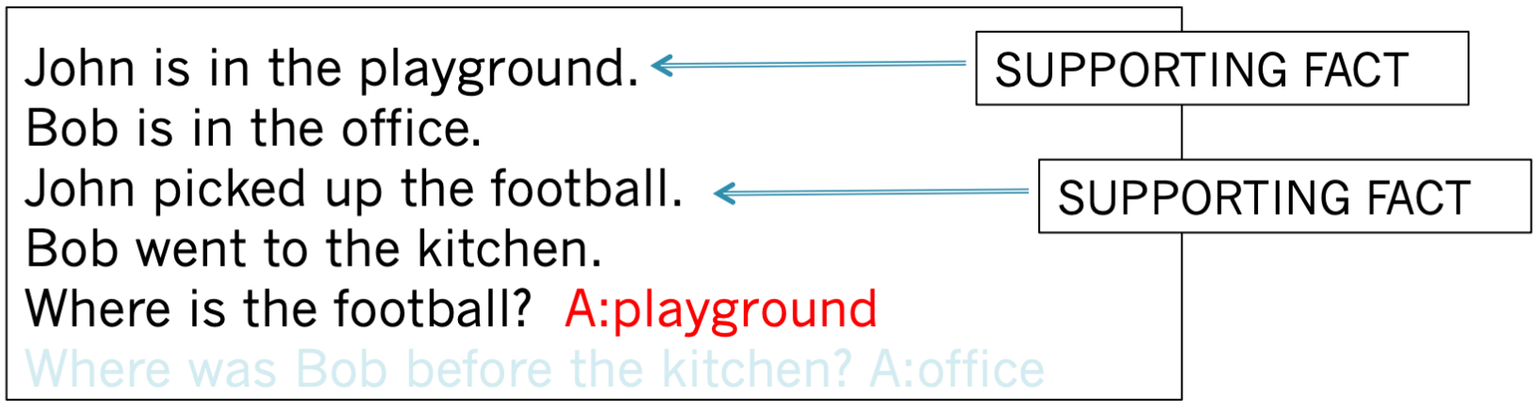
- Memory based networks such as NTM are able to succeed on copy task



Applications of Memory Networks

- Memory networks have been applied when we need models that have to go through multiple reasoning steps that require a long term time horizon

- Read a story and then answer questions about it

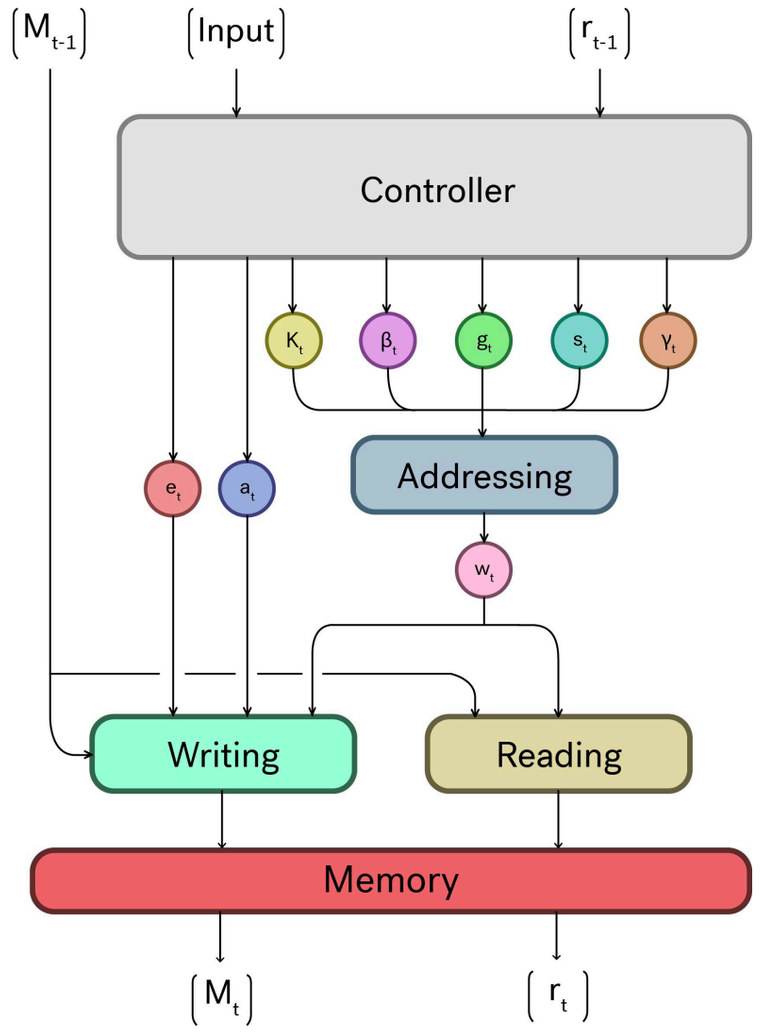


- Dialog, remember previous dialog and respond

Memory Network Example

John is in the playground. ← SUPPORTING FACT
Bob is in the office.
John picked up the football. ← SUPPORTING FACT
Bob went to the kitchen.
Where is the football? A:playground
Where was Bob before the kitchen? A:office

- Caricature of trained model behaviour at test time
  - Process the story storing each relevant fact in memory as a vector
  - When queried to answer select relevant facts from memory
- Read operation can be seen as a form of “attention”



Read op

a = NN(parameters)

$w = softmax(a)$ → [0.02, 0.90, 0.05, 0.01, 0.02]

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i),$$

Attention Models

- Attention at a high level refers to model components which focus on a part of the input or more generally on an abstract memory
- The idea is this can help reduce the space of processing needed
- This idea has a long history in deep learning/neural network research
    - Multiple works from Michael Mozer, Jeurgen Schmidhuber since 1990s
- A motivation and analog to human foveal glimpses

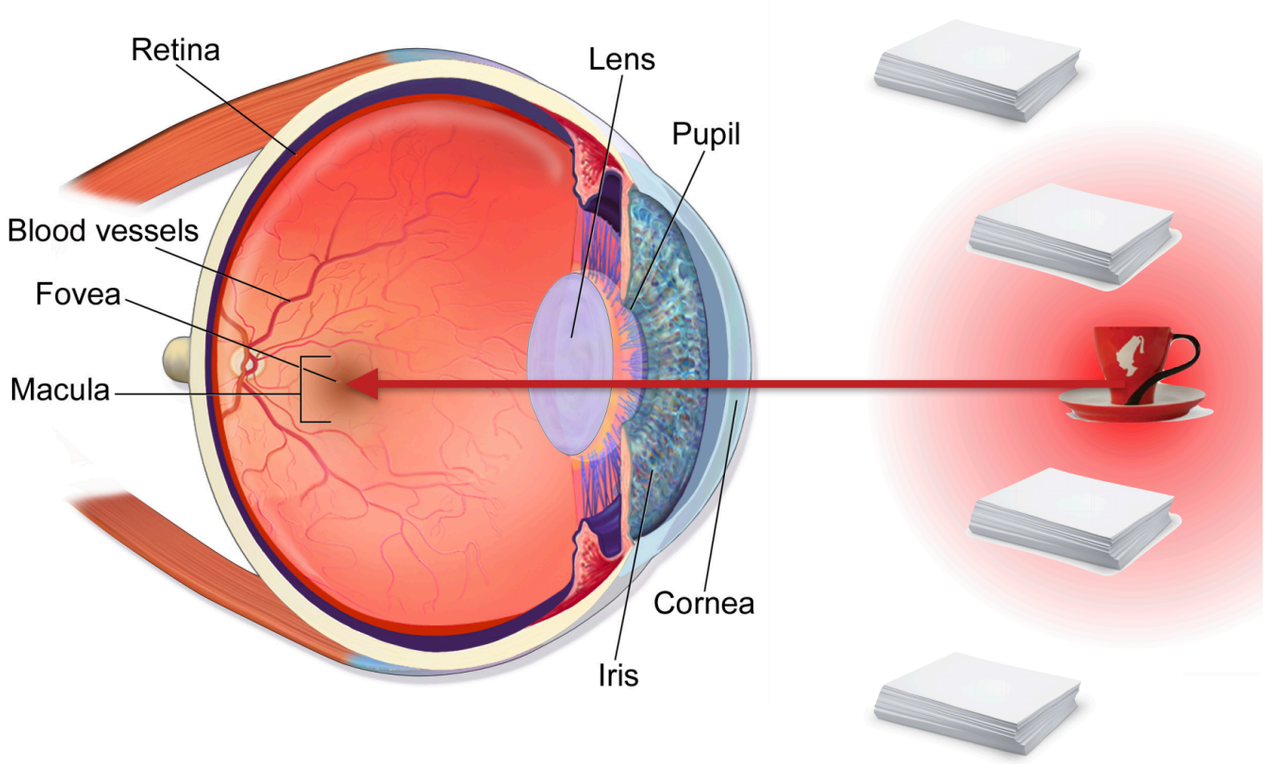
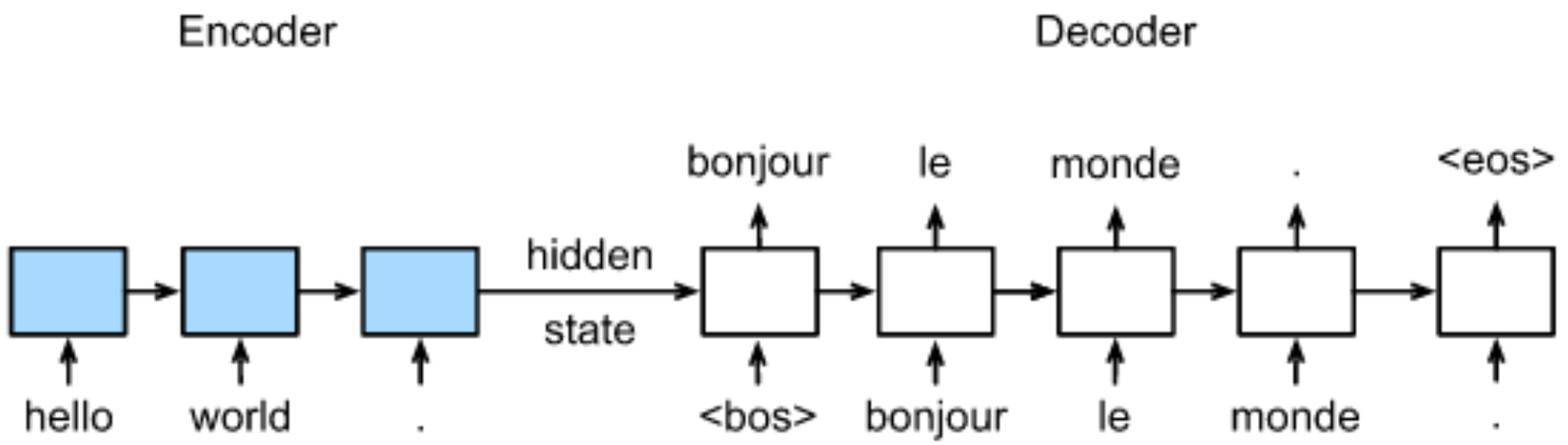


Image Credit: Alex Smola/AWS

Sequence to Sequence for NMT



- Cells are LSTM or GRU
- Can be deep
- Maximize log likelihood of output sequence
    - Softmax output over all possible words + Cross Entropy

NMT + Bahdanu Attention

- Adds Attention to Seq2Seq model

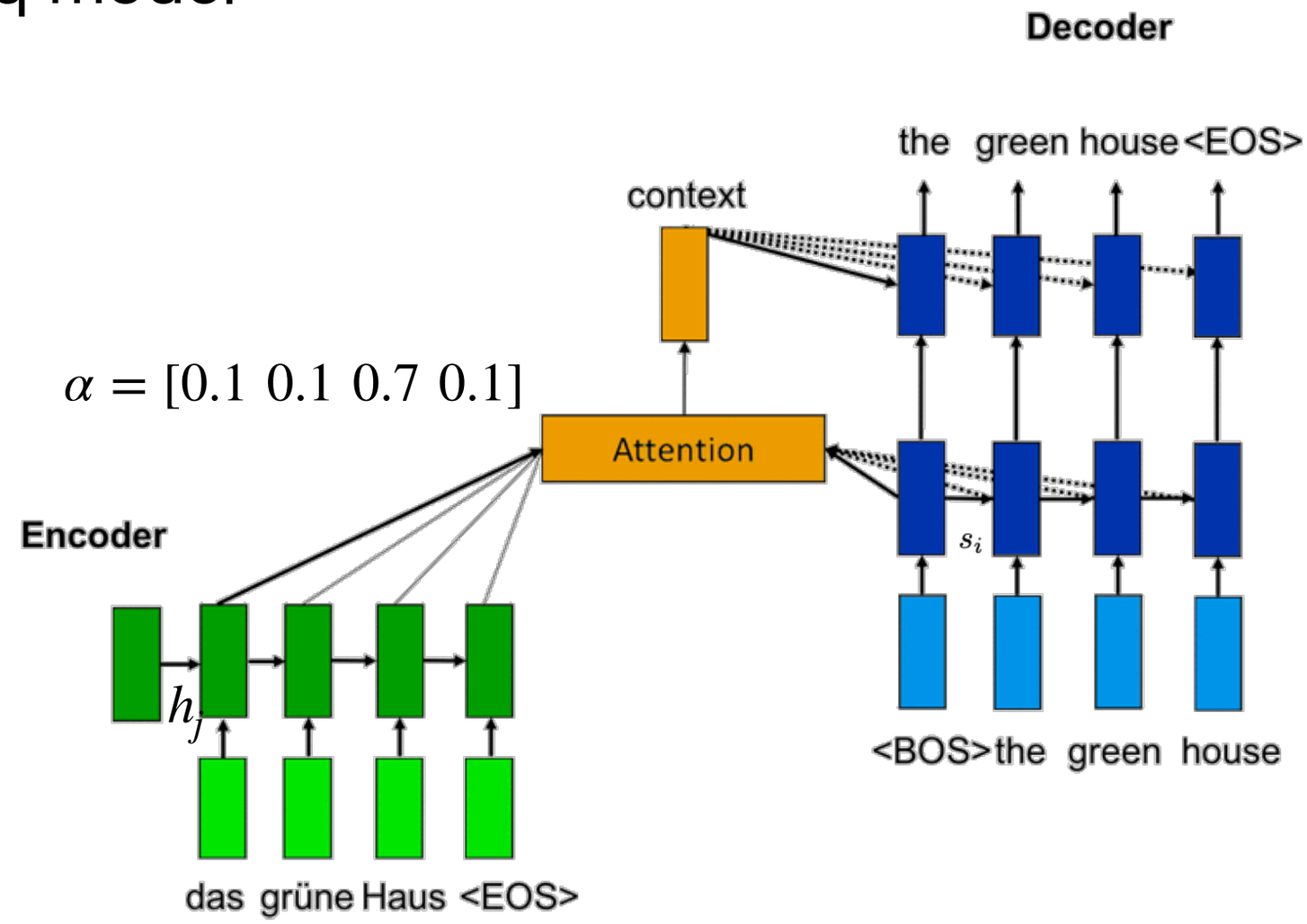
h_j **Encoder state**

s_i **Decoder state**

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

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

Attention



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

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

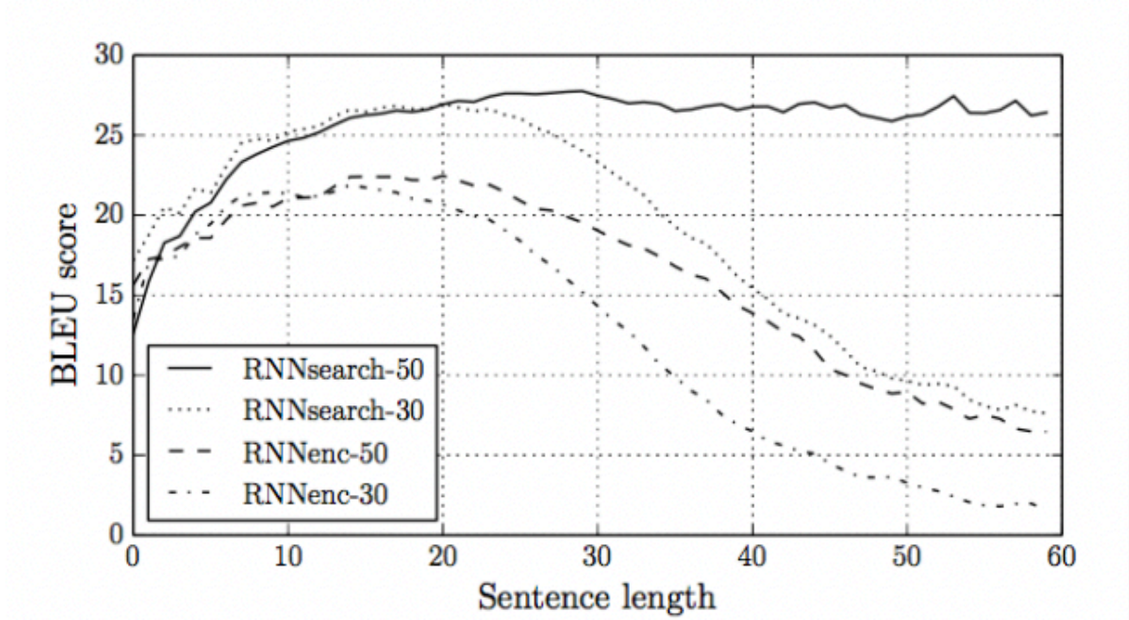
Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

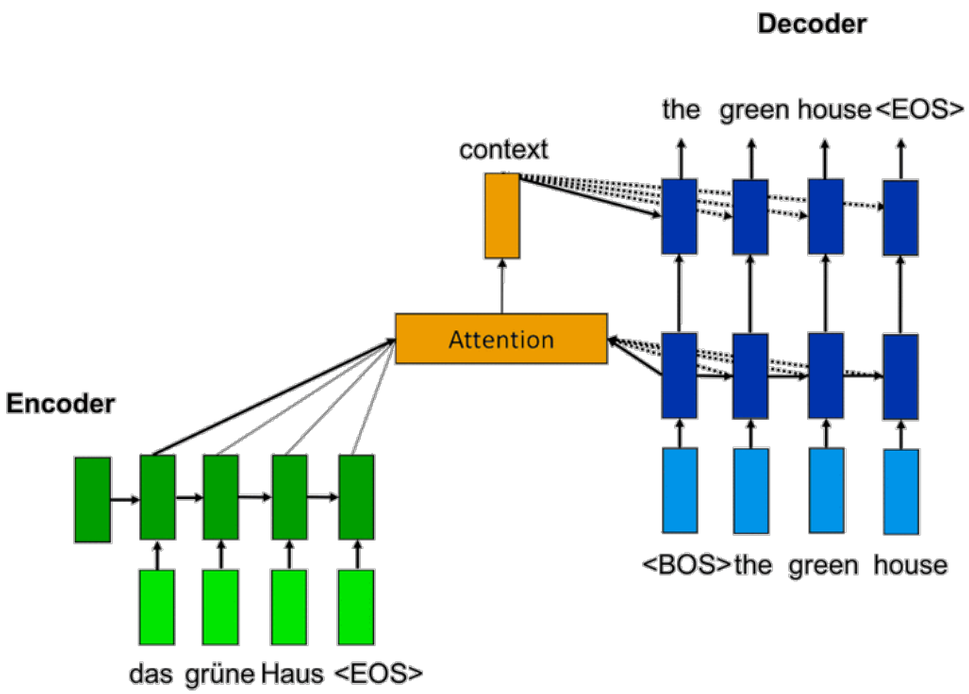
by D Bahdanau · 2014 · Cited by 16926 -

NMT + Bahdanu Attention

- Much better at longer sequences



s_i



NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

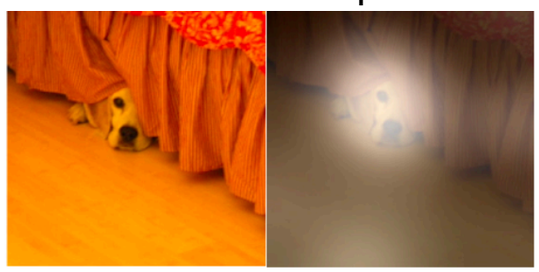
Attention in other Encoder-Decoder Models

- Consider caption generation

$$a = \{\mathbf{a}_1, \ldots, \mathbf{a}_L\}, \ \mathbf{a}_i \in \mathbb{R}^D$$

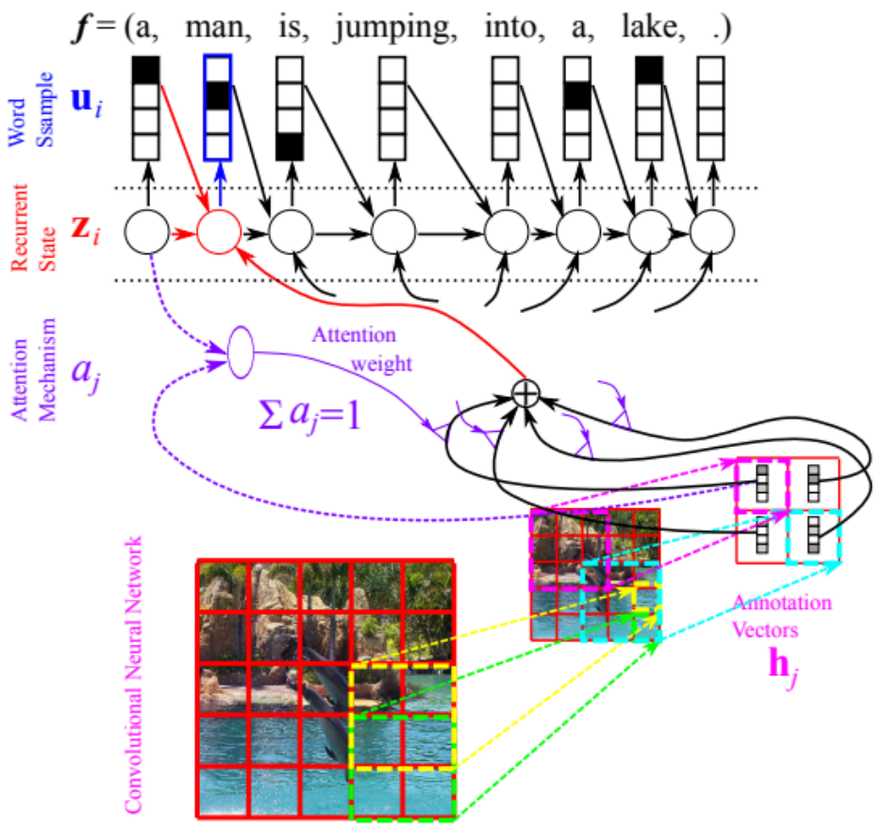
- Localized CNN Representations extracted
- Average of localized representations + NN fed as state vector to LSTM decoder
- LSTM decoder attends to regions at each time step

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.



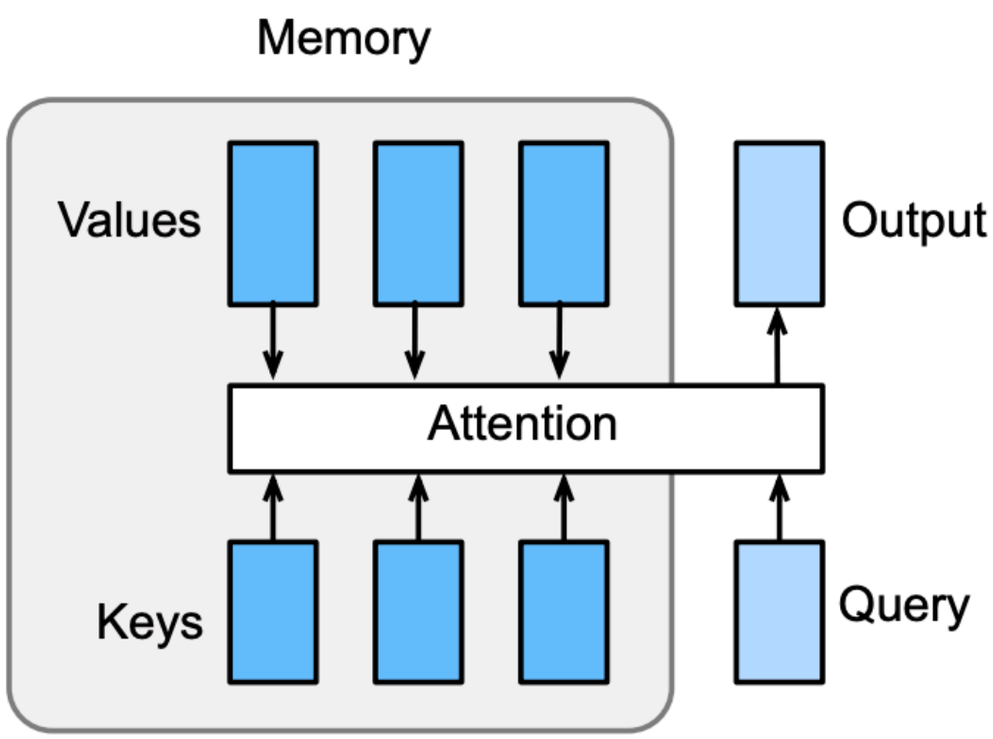
Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

Kelvin Xu KELVIN.XU@UMONTREAL.CA
Jimmy Lei Ba JIMMY@PSI.UTORONTO.CA
Ryan Kiros RKIROS@CS.TORONTO.EDU
Kyunghyun Cho KYUNGHYUN.CHO@UMONTREAL.CA
Aaron Courville AARON.COURVILLE@UMONTREAL.CA
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Richard S. Zemel ZEMEL@CS.TORONTO.EDU
Yoshua Bengio FIND-ME@THE.WEB

Attention Layer and Relationship to Memory

- We can generalize the attention idea by considering what it attends to as “memory”, similar to memory networks

- The attention operation now consists of query searched against keys giving an output values



$$\text{attention}(Q, K, V)$$

Attention Layer and Relationship to Memory

- We can generalize the attention idea by considering what it attends to as “memory”, similar to memory networks

$$\text{attention}(Q, K, V)$$

- The attention operation now consists of query searched against keys giving an output values

h_j **Encoder state - key, value**

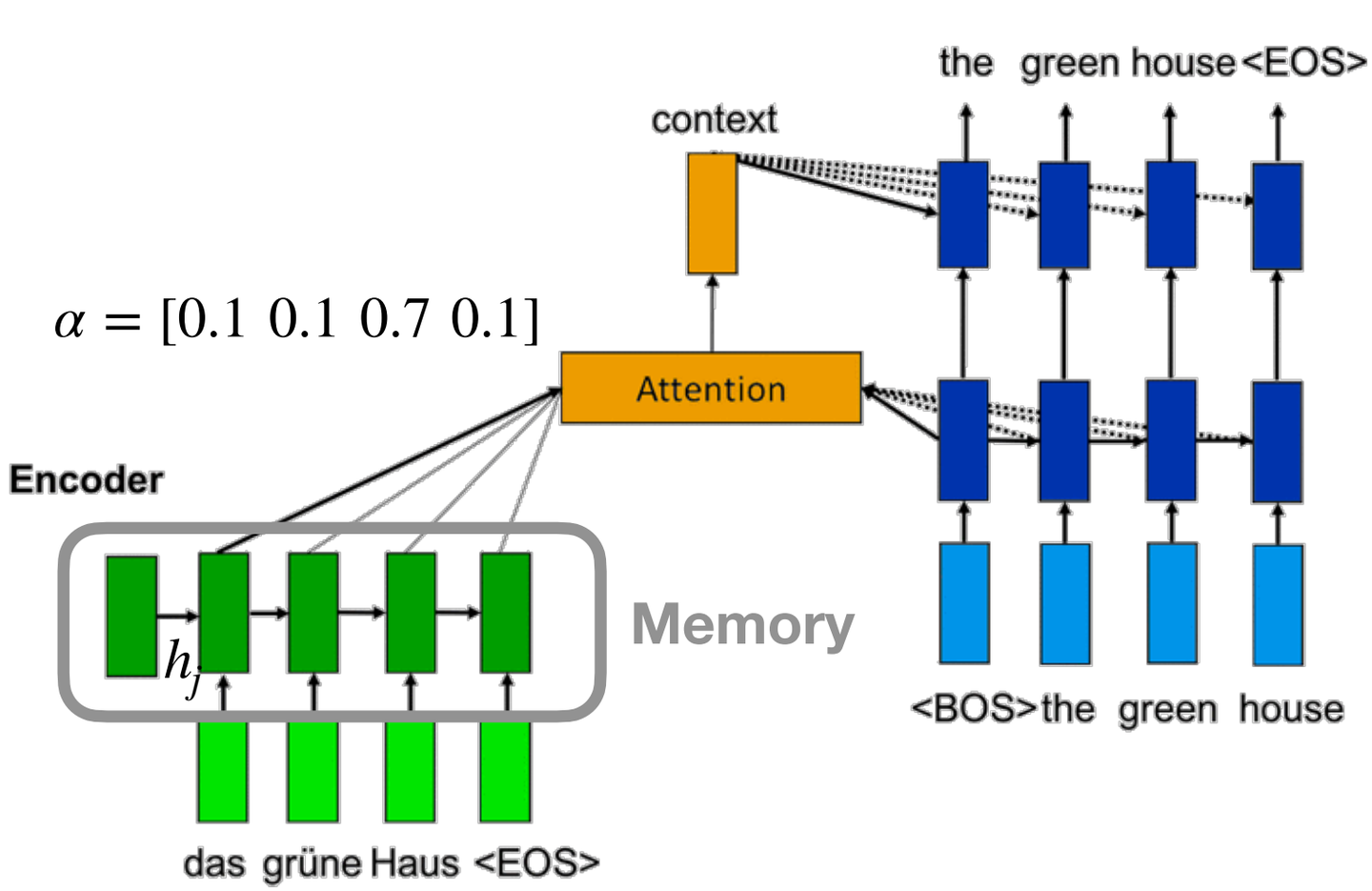
S_i **Decoder state - query**

AttentionLayer(S , H , H)

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

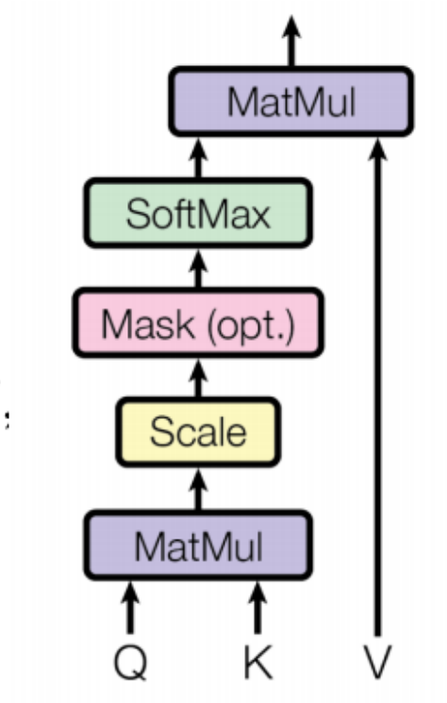
$$e_{ij} = a(s_{i-1}, h_j)$$

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



Scaled Dot Product Attention

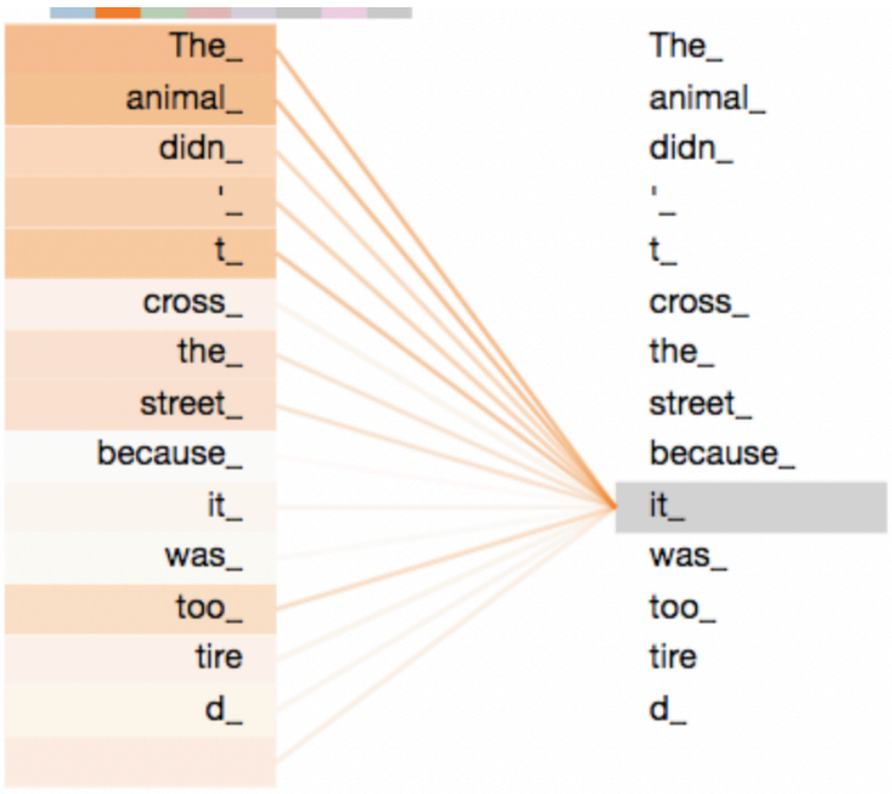
- Now most commonly used attention version



$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

Self-Attention

- We can use a similar idea to create a new feedforward layer where the model is forced to “attend” to parts of the input in producing the next output
- Can consider each entry in the input as maintaining and updating a state
- Similar ideas appear in early literature under names such as “non-local” neural networks and other names
- Related to non-local means algorithm in computer vision
- Now fundamental building block of NLP deep learning models and bleeding into other applications.



$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

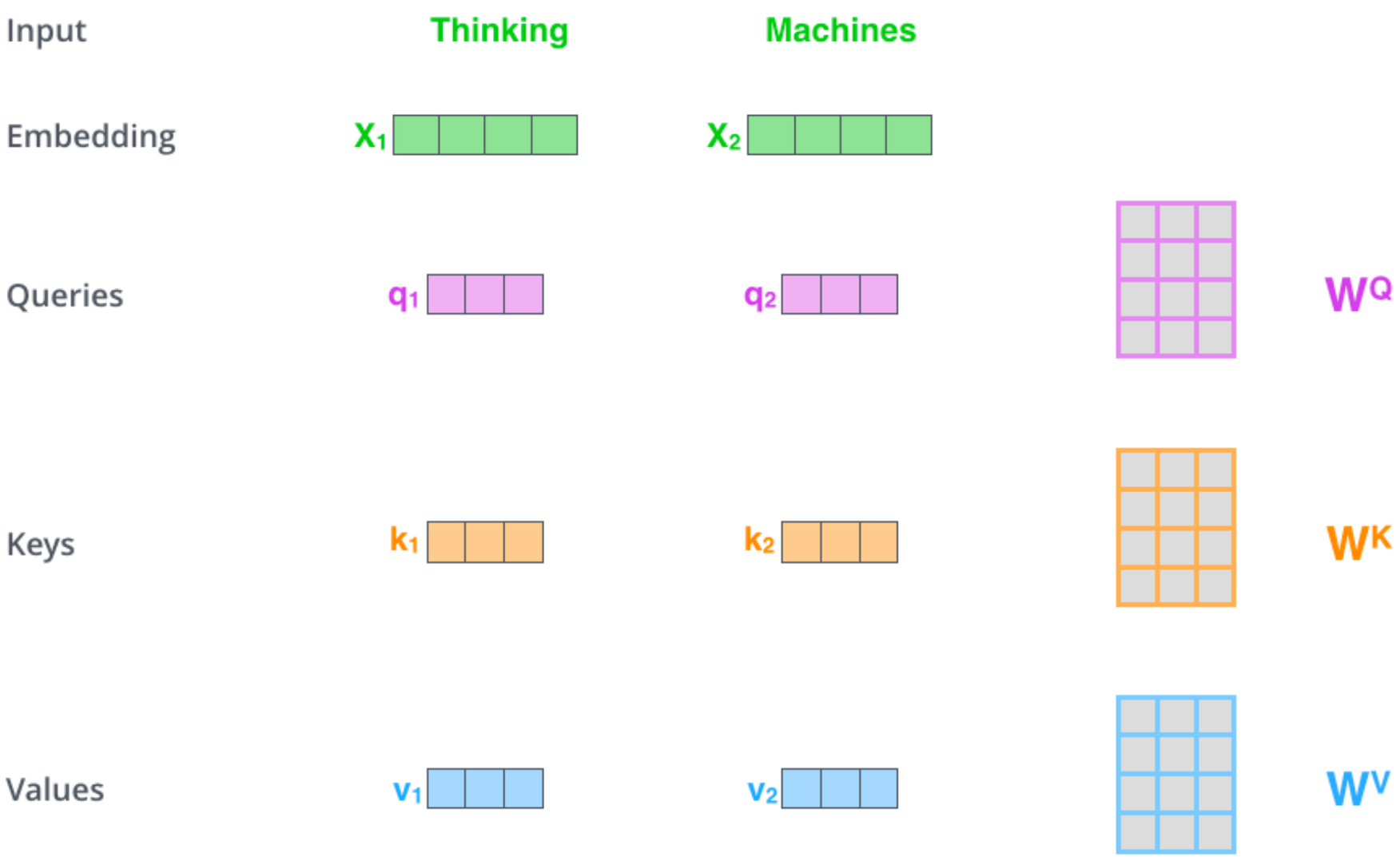
$$\boldsymbol{X} \in \mathbb{R}^{T \times D_{in}}$$

$$Q = W_{qry}\mathbf{X} \quad V = W_{val}\mathbf{X}$$
$$K = W_{key}\mathbf{X}$$

$$\mathbf{A} = \frac{\mathbf{X}\mathbf{W}_{\mathbf{qry}}\mathbf{W}_{\mathbf{key}}^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}}{\sqrt{\mathbf{d}_{\mathbf{k}}}}$$

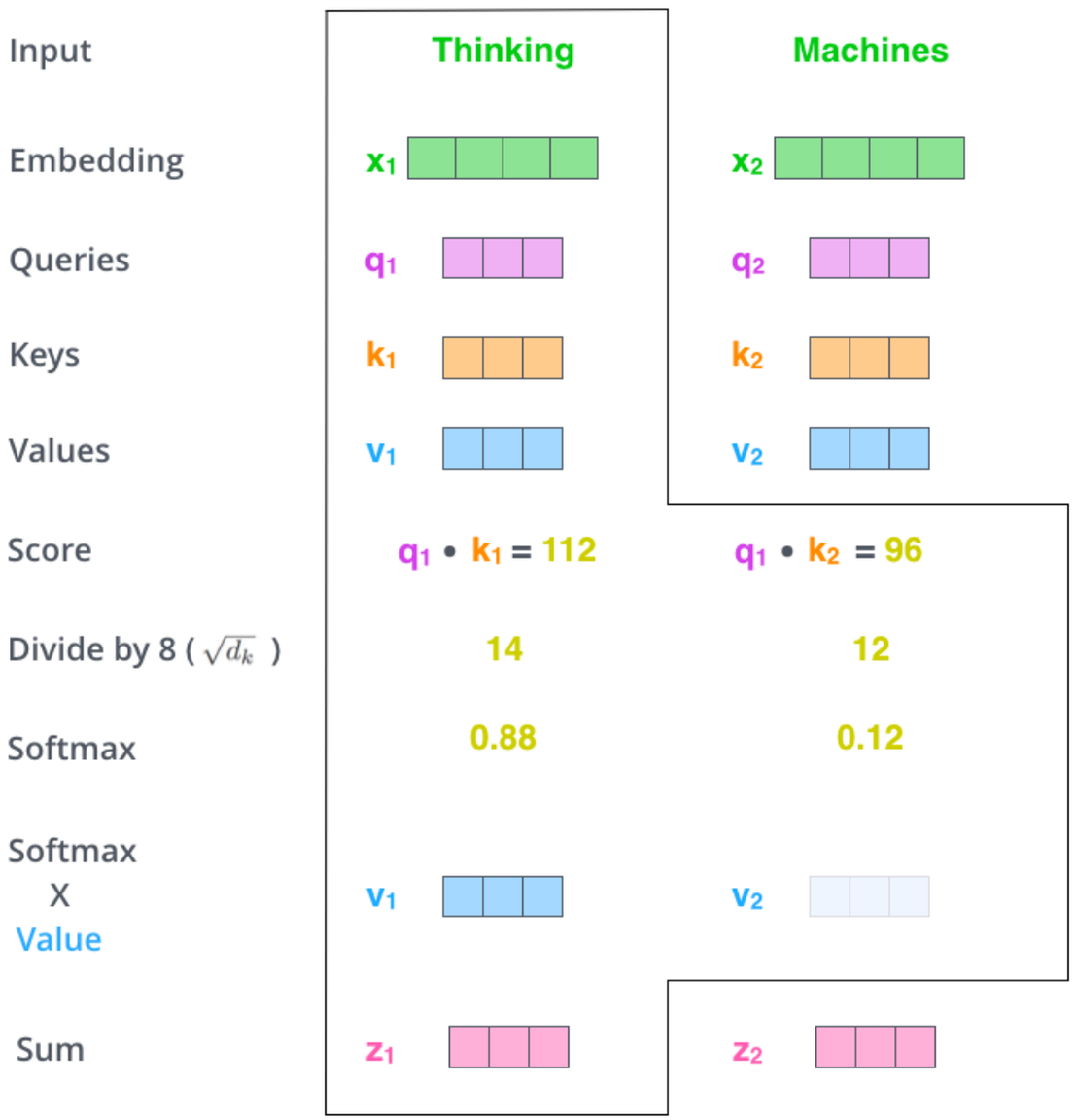
$$\text{Self-Attention}(\boldsymbol{X})_{t,:} := \text{softmax}\left(\boldsymbol{A}_{t,:}\right)\boldsymbol{X}\boldsymbol{W}_{val},$$

Self-Attention Visualized



Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention Visualized



$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

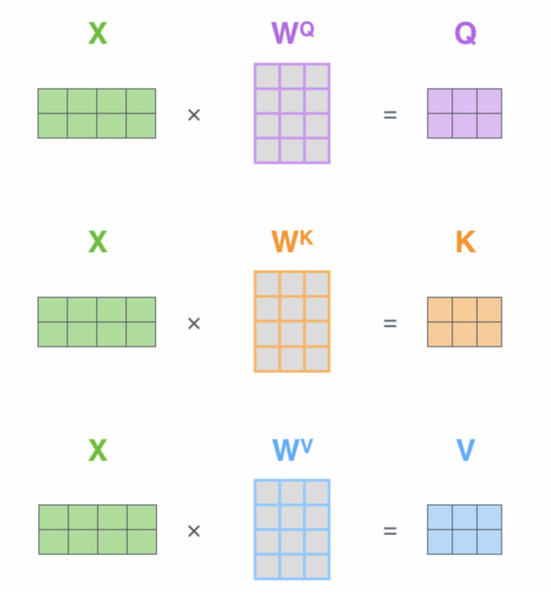
$$\boldsymbol{X} \in \mathbb{R}^{T \times D_{in}}$$

$$Q = W_{qry}\mathbf{X} \quad V = W_{val}\mathbf{X}$$

$$K = W_{key}\mathbf{X}$$

Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

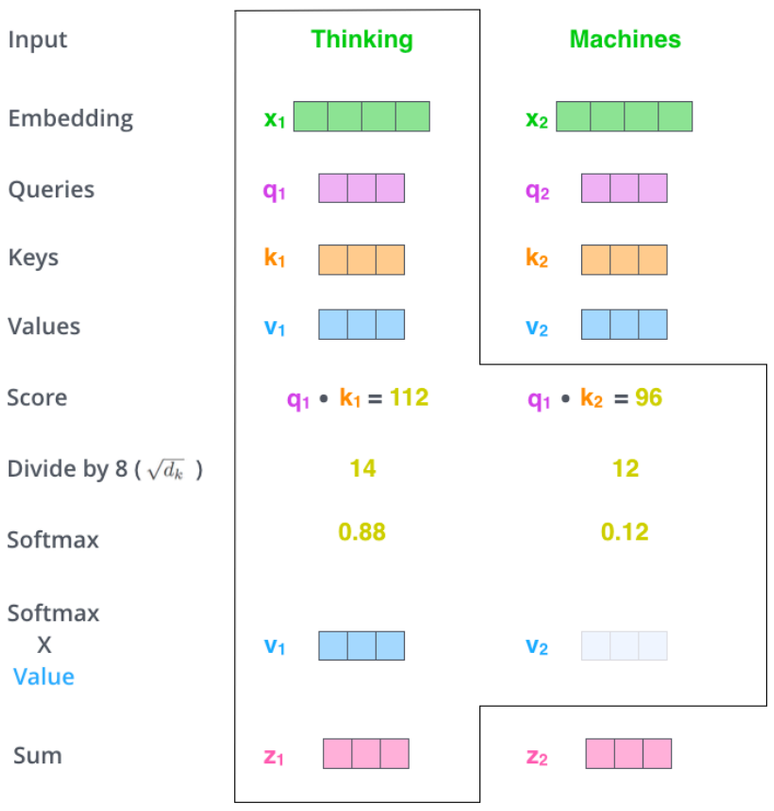
Self-Attention, Input Order?

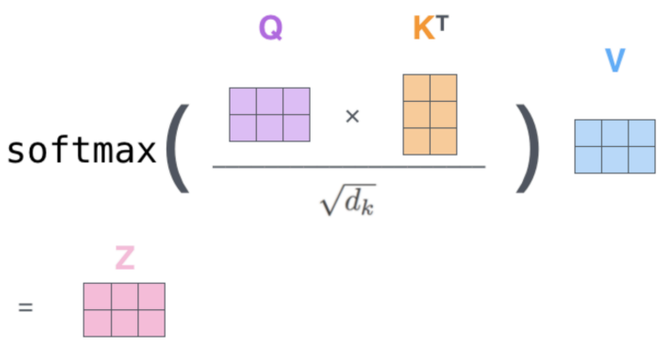


$$\boldsymbol{X} \in \mathbb{R}^{T \times D_{in}}$$

$$\mathbf{A} = \frac{\mathbf{X}\mathbf{W}_{\mathbf{qry}}\mathbf{W}_{\mathbf{key}}^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}}{\sqrt{\mathbf{d_k}}}$$

$$\text{Self-Attention}(\boldsymbol{X})_{t,:} := \text{softmax}\left(\boldsymbol{A}_{t,:}\right)\boldsymbol{X}\boldsymbol{W}_{val},$$





Matrix Form

Matrix Form Visualized

- What happens if we randomly permute the rows of X?

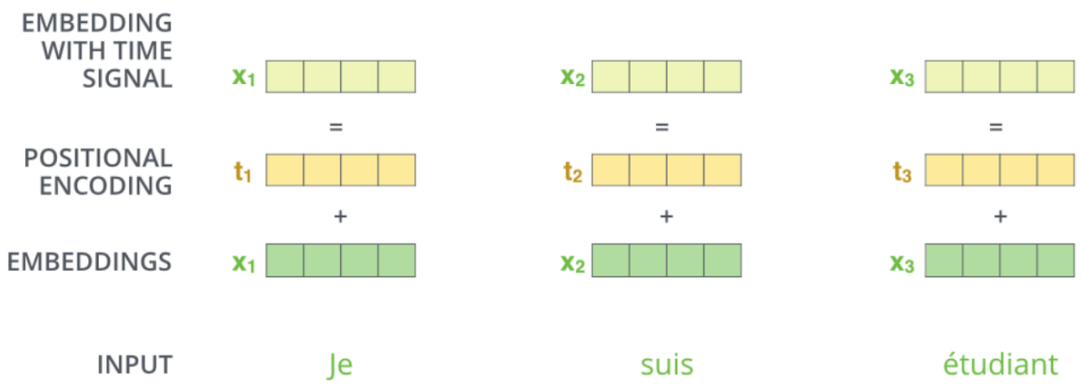
Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Positional Encodings

- To deal with this issue of position information we need to create encodings

- One way this is commonly done is by adding or concatenating a fixed or learnable embedding, distances of these embeddings can be designed to be meaningful

- Learned embeddings can often infer the relationship between input variables



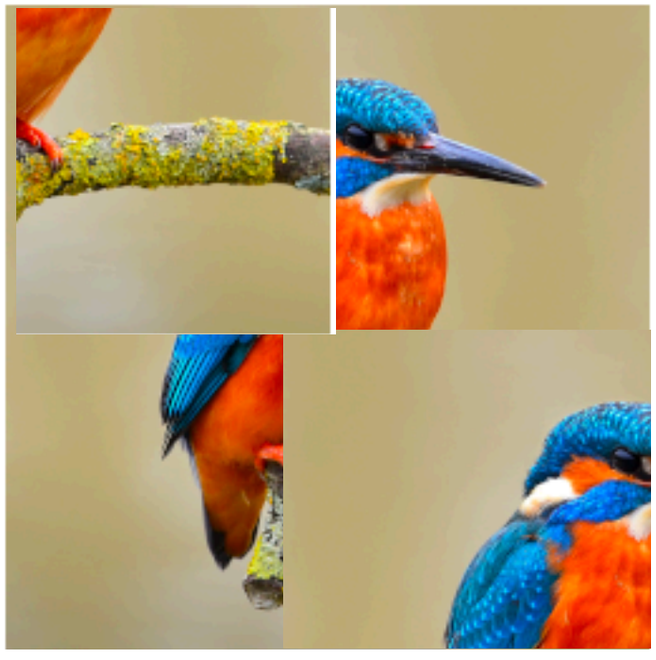
$$\boldsymbol{A} := (\boldsymbol{X} + \boldsymbol{P})\boldsymbol{W}_{qry}\boldsymbol{W}_{key}^{\top}(\boldsymbol{X} + \boldsymbol{P})^{\top},$$

$$\mathrm{PE}_{pos,2i} = sin(pos/10000^{2i/d_{emb}}),$$
$$\mathrm{PE}_{pos,2i+1} = cos(pos/10000^{2i/d_{emb}}),$$

Self-Attention Can Learn Locality from Data

- Consider a single convolutional layer. The local area of interest is predefined and rigid. What if the pre-defined locality constraints don’t apply

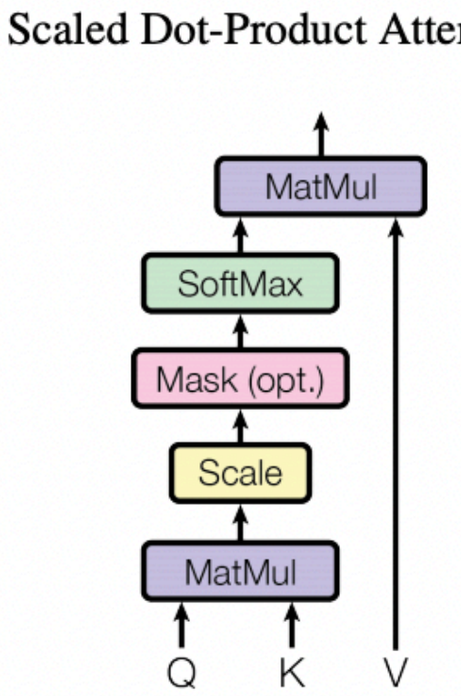
- Self-attention with learned positional encoding can infer the variable structure from data

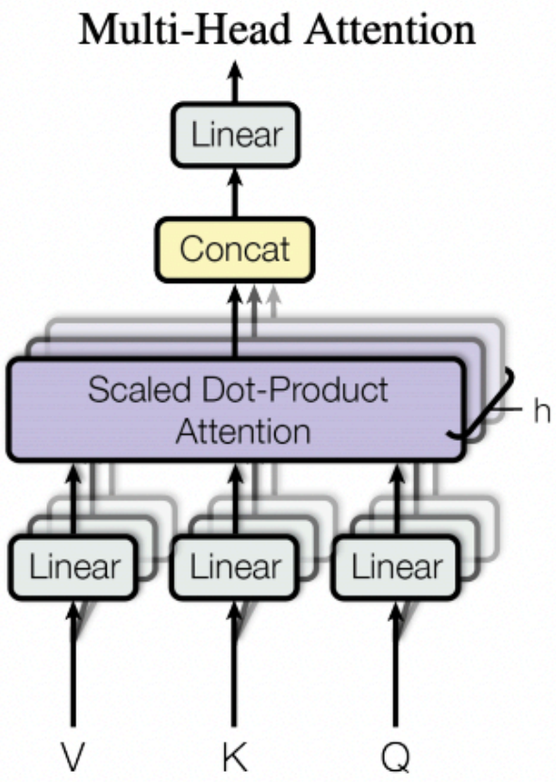


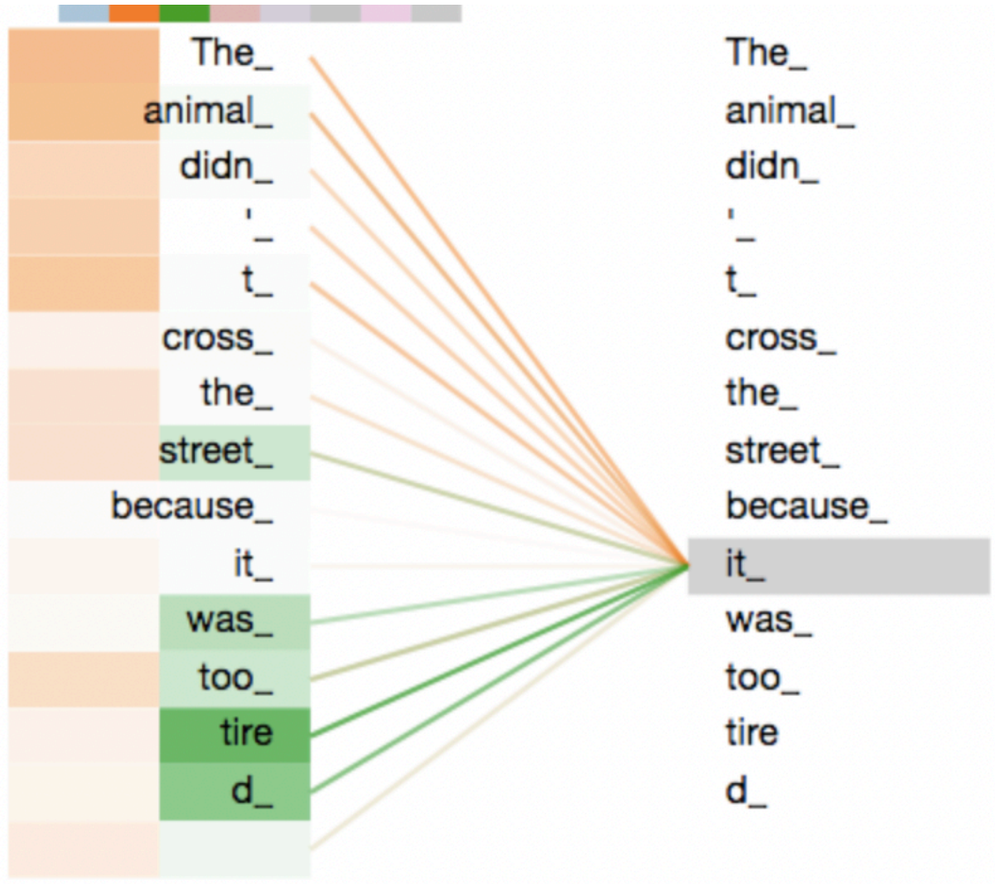
ON THE RELATIONSHIP BETWEEN SELF-ATTENTION AND CONVOLUTIONAL LAYERS

Jean-Baptiste Cordonnier, Andreas Loukas & Martin Jaggi
École Polytechnique Fédérale de Lausanne (EPFL)
{first.last}@epfl.ch

Multi-Head Attention







$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \dots, head_h)W^O$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\,.$$

Multi-Head Attention in Pytorch

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

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
             / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

From recommended resource:

https://nlp.seas.harvard.edu/2018/04/03/attention.html

Multi-Head Attention in Pytorch

```
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        "Implements Figure 2"
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)

        # 1) Do all the linear projections in batch from d_model => h x d_k
        query, key, value = \
            [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
             for l, x in zip(self.linears, (query, key, value))]

        # 2) Apply attention on all the projected vectors in batch.
        x, self.attn = attention(query, key, value, mask=mask,
                                 dropout=self.dropout)

        # 3) "Concat" using a view and apply a final linear.
        x = x.transpose(1, 2).contiguous() \
             .view(nbatches, -1, self.h * self.d_k)
        return self.linears[-1](x)
```

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

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
             / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

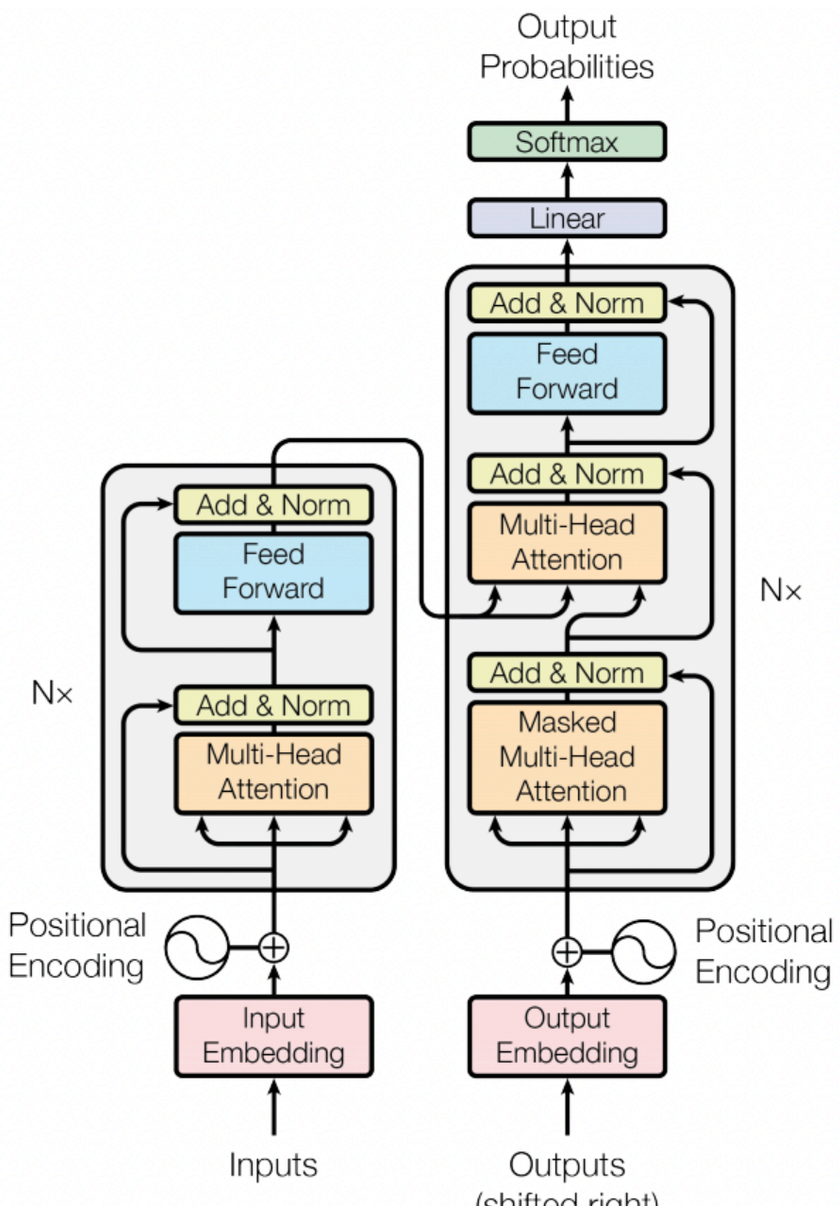
From recommended resource:

https://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer Models

- Transformer = Self-Attention idea + Encoder-Decoder

- Transformer now more often just used to refer to models built with self-attention + feedforward layers + positional encoding.



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

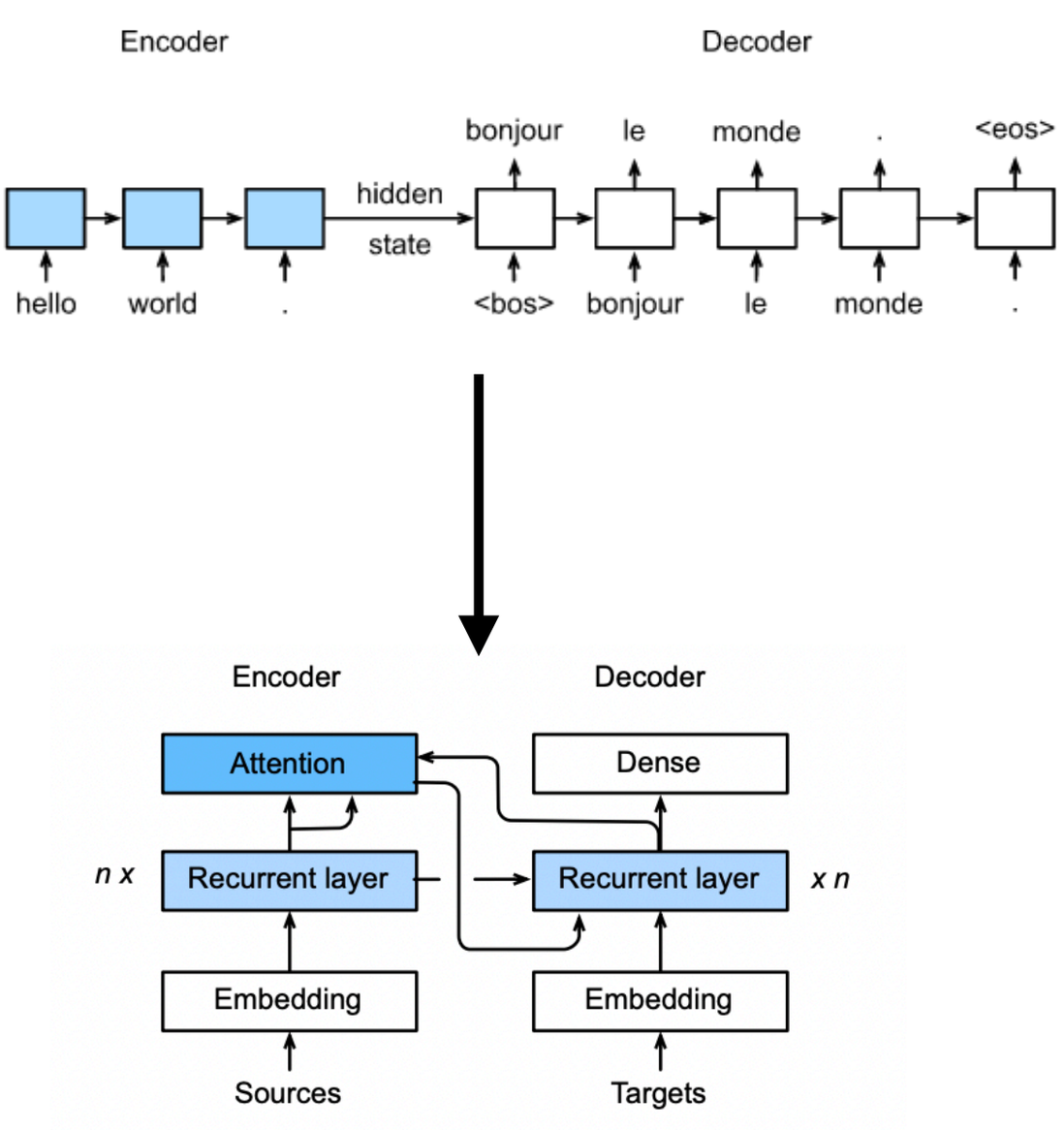
Llion Jones* Google Research llion@google.com

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

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer



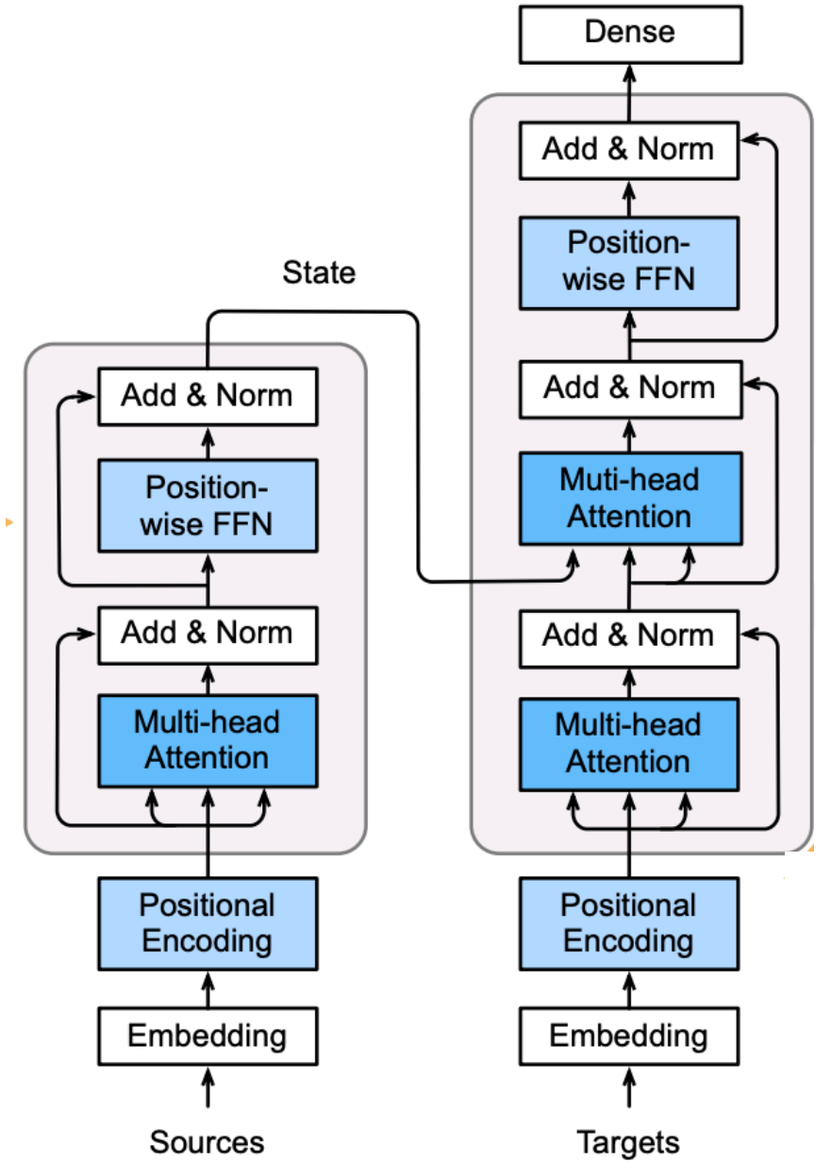
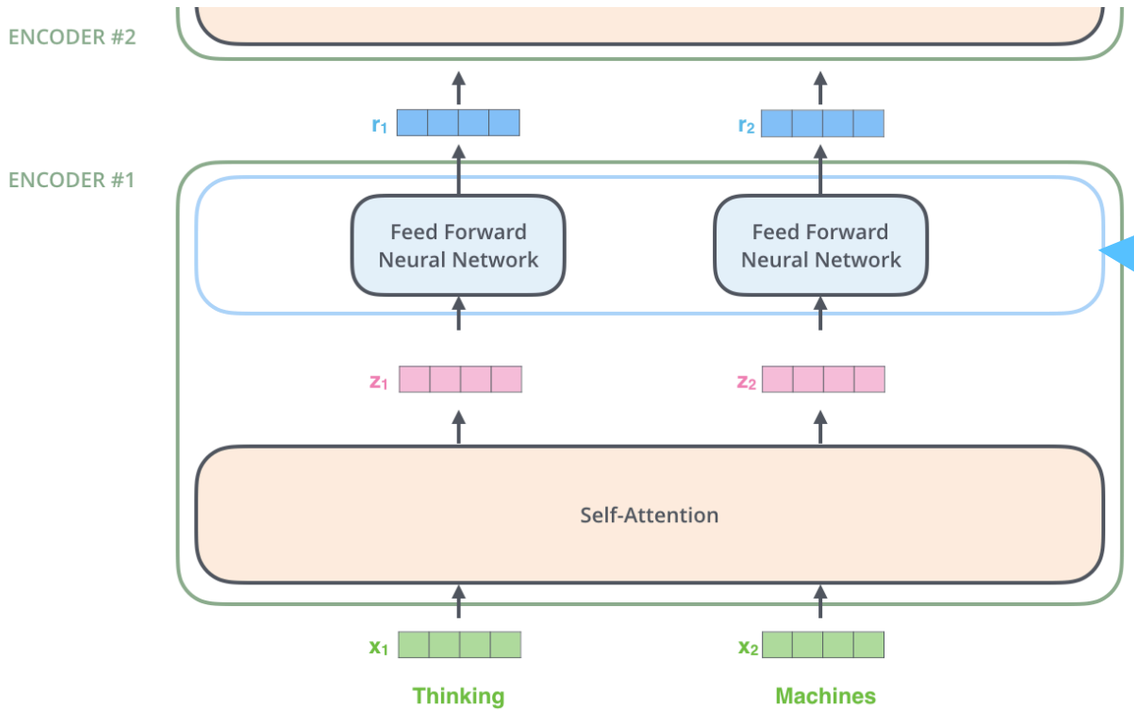
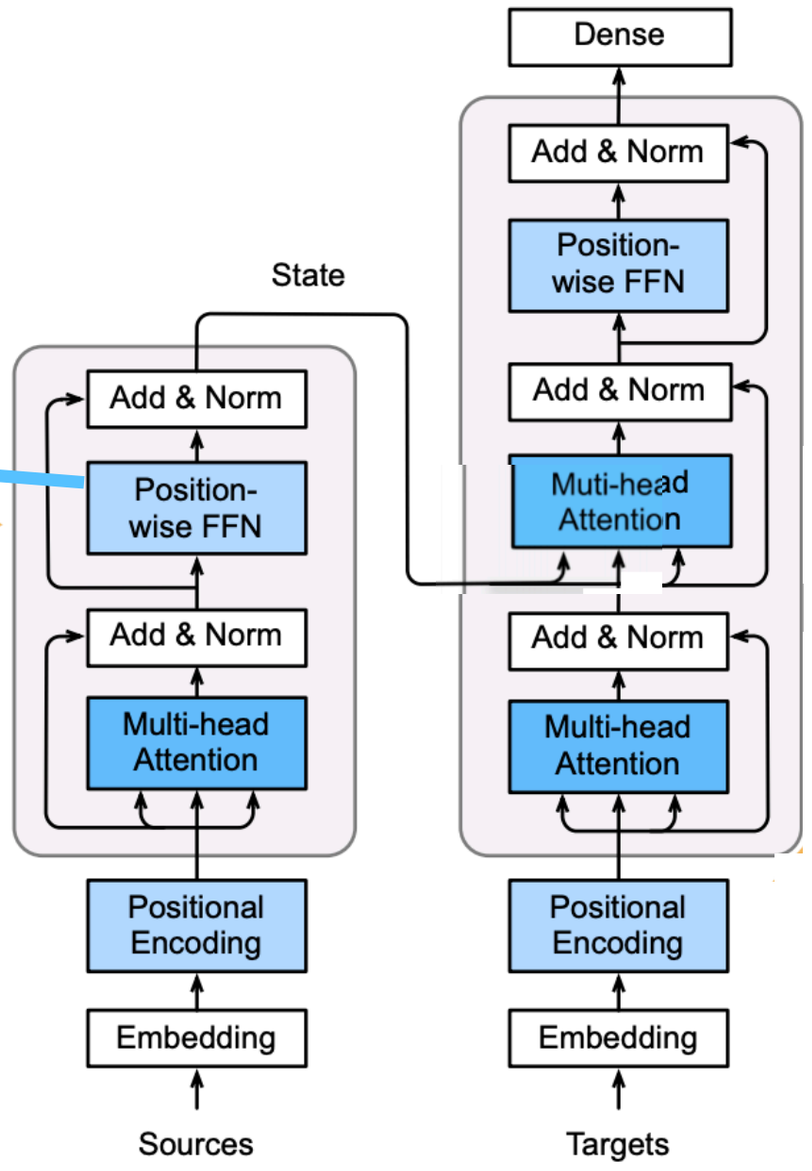


Image Credit: Alex Smola/AWS

Transformer Positionwise FFN



Position-wise FFN can also be thought of as a 1x1 conv layer



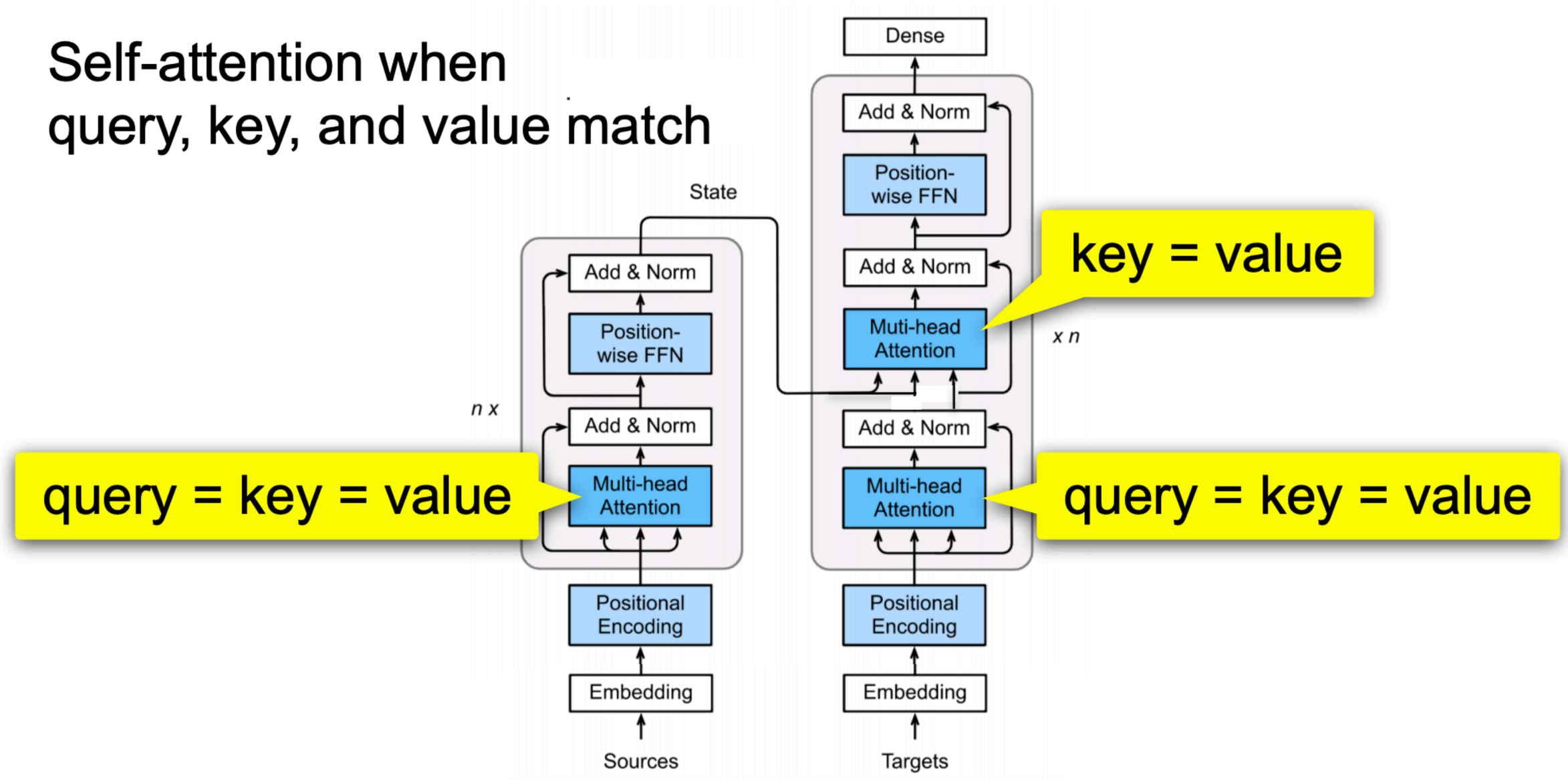
Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Image Credit: Alex Smola/AWS

Transformer

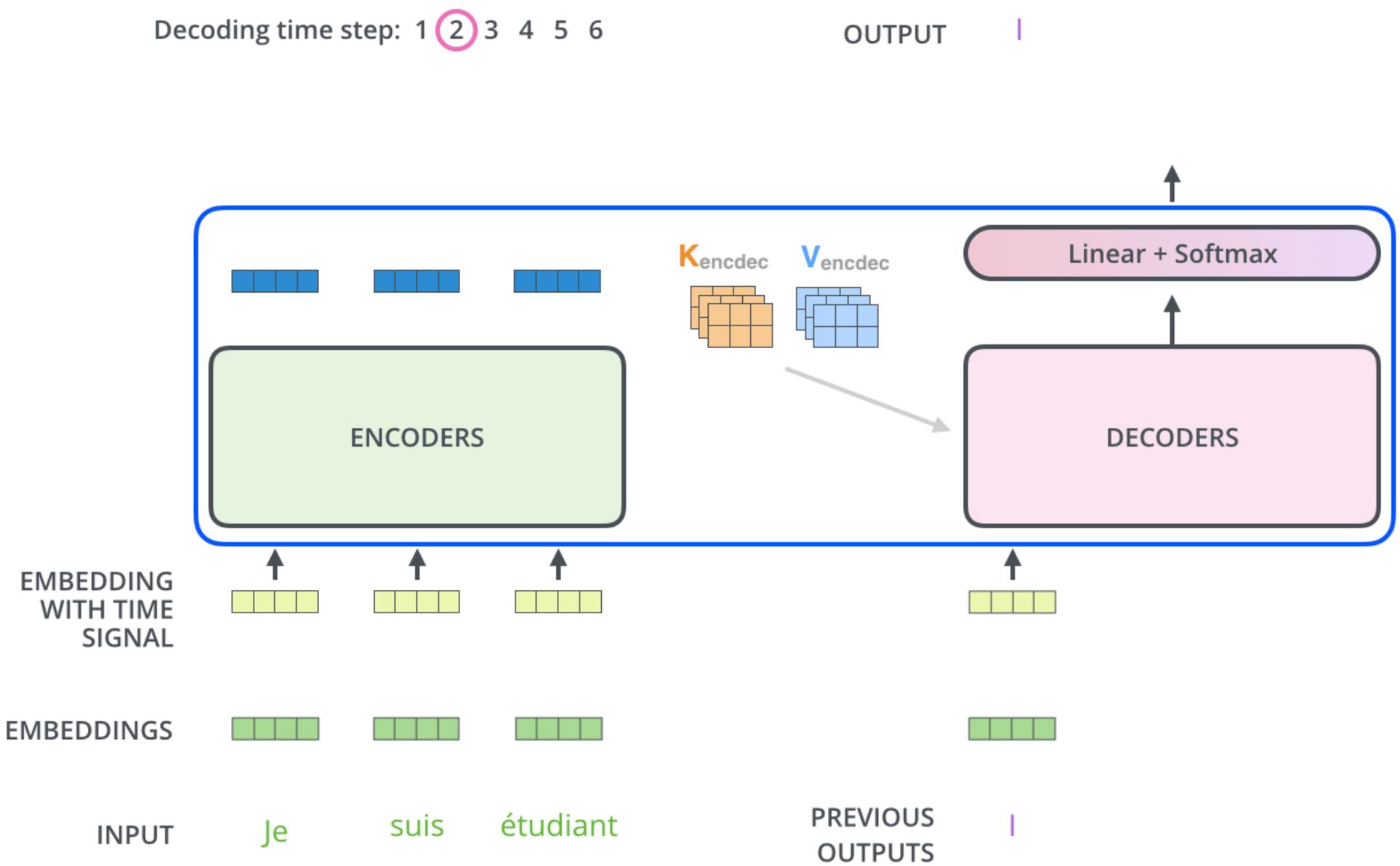
Transformer with multi-head attention (Vaswani et al., '17)

Self-attention when
query, key, and value match

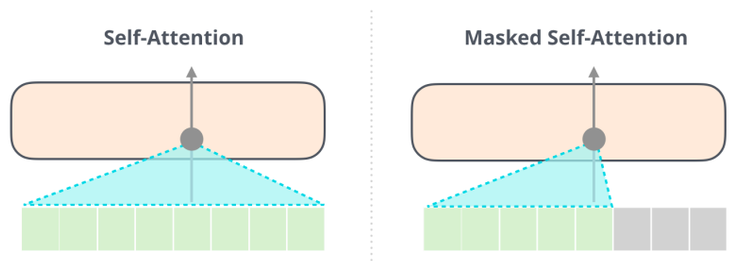


Slide Credit: Alex Smola/AWS

Transformer At Inference Time

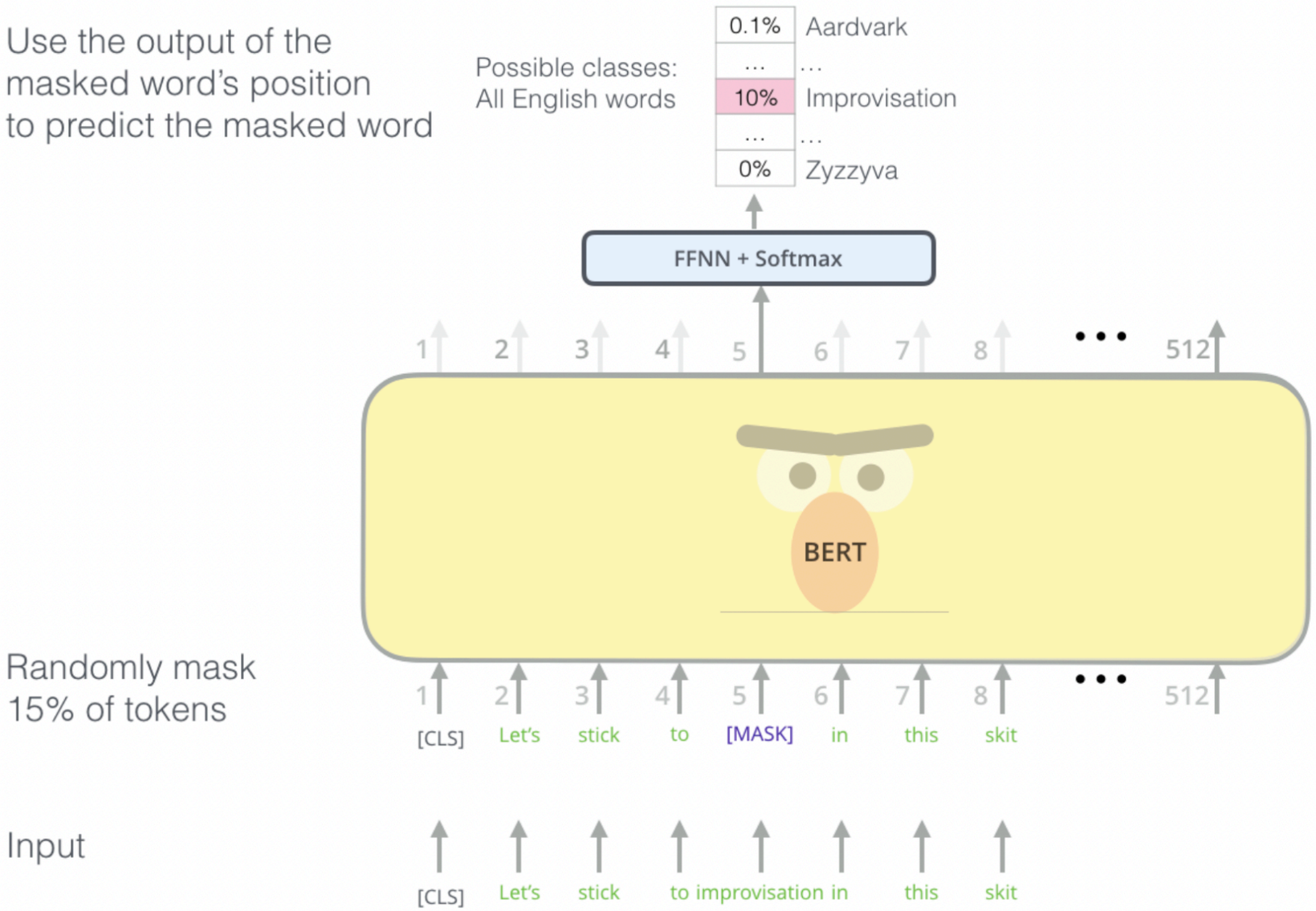


In training we rely on masking



Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Language Modeling With Transformers



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Language Modeling With Transformers

- BERT was an AlexNet like breakthrough in NLP, particularly reusable features

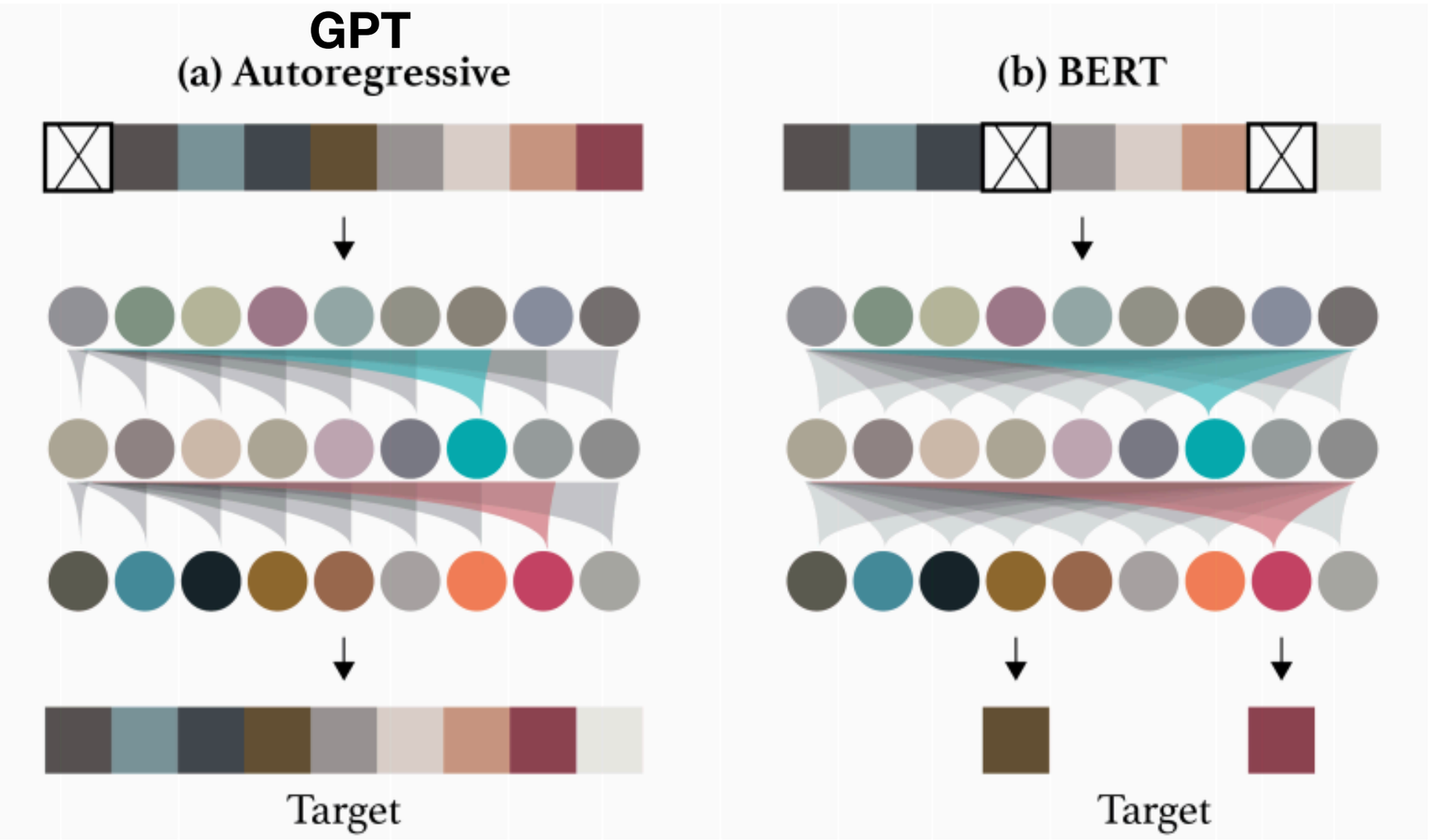
- Prior: For many tasks LSTM's were barely outperforming bag of words models that ignore the structure of the input

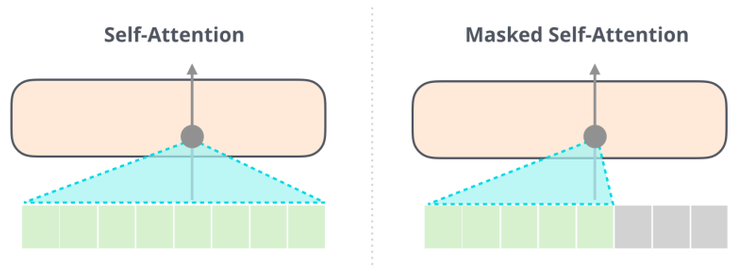
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

GPT





Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

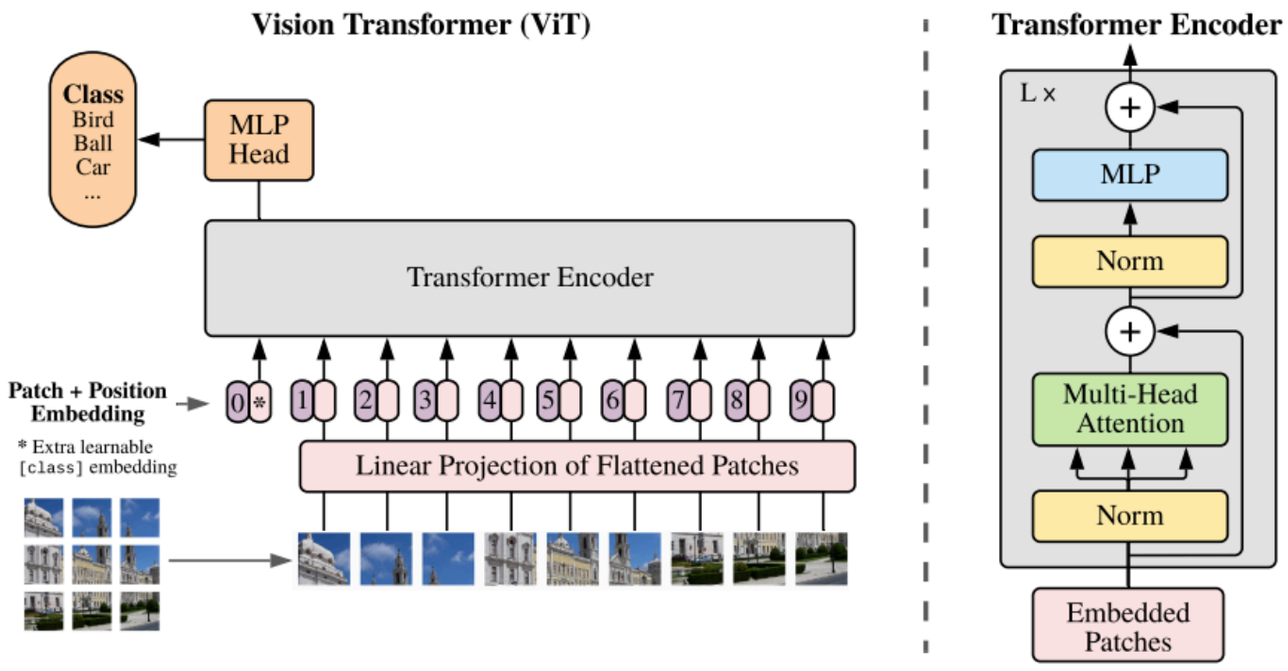
Mask+ softmax

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

Visual Transformers

- Transformers/Self-Attention are becoming used in other areas
- More general architectural bias
- Potentially allow better multi-modal learning
- Speed remains an issue compared to e.g. CNN



Advantages over RNN

- RNN
  - Parallelization is poor
  - Repeated operations lead to vanishing/exploding gradients -> issues in long term structure
  - The above two points apply to models like memory networks / neural turing machines as well
  - RNN variants like LSTM and GRU though much better than vanilla RNN still suffer issues in long horizon and are still

Scalability of Transformers

- Scalability is still an issue
  - Self-Attention layers are more expensive than convolution or regular fully connected layer
  - Transformers don’t have the bottleneck of RNN sequence parallelization but each layer can be very expensive
  - Big bottleneck for now in computer vision use
- Training of transformers is less robust than CNNs
  - Requires complicated learning rate schedules and optimizers

$$\boldsymbol{X} \in \mathbb{R}^{T \times D_{in}}$$

$$\mathbf{A} = \frac{\mathbf{X}\mathbf{W}_{\mathbf{qry}}\mathbf{W}_{\mathbf{key}}^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}}{\sqrt{\mathbf{d}_{\mathbf{k}}}}$$

$$\text{Self-Attention}(\boldsymbol{X})_{t,:} := \text{softmax}\left(\boldsymbol{A}_{t,:}\right)\boldsymbol{X}\boldsymbol{W}_{val},$$