Attention! Attention

или

Внимание! Внимание)

by metya

Disclaimer

- Almost all content from this presentation has been impudently stolen from well-known sources.
- All links to the desecrated sources will be at the end.
- But all needed original sources will be in footnotes.
- There is a lot of formulae so
- Pay ATTENTION.
- Lol.

What to know paperswithcode about attention?

Attention

Add Method | Edit Category

METHOD	YEAR	PAPERS
Multi-Head Attention	2017	2253
Scaled Dot-Product Attention	2017	2225
Additive Attention	2014	68
Dot-Product Attention	2015	44
SAGAN Self-Attention Module	2018	31
Location-based Attention	2015	23
Content-based Attention	2014	22
Spatial Attention Module	2018	17
Channel Attention Module	2018	10
DV3 Attention Block	2017	8
Location Sensitive Attention	2015	7
Spatial Attention-Guided Mask	2019	6
LAMA	2019	5
Channel-wise Soft Attention	2017	4
Global and Sliding Window Attention	2020	2
Strided Attention	2019	2
Point-wise Spatial Attention	2018	2
Sliding Window Attention	2020	2
Single-Headed Attention	2019	2
Dilated Sliding Window Attention	2020	2
LSH Attention	2020	2
Fixed Factorized Attention	2019	2
SortCut Sinkhorn Attention	2020	1
Dense Synthesized Attention	2020	1
Graph Self-Attention	2019	1
Multi-Head Linear Attention	2020	1
Adaptive Masking	2019	1
Sparse Sinkhorn Attention	2020	1
Factorized Random Synthesized Attention	2020	1
Global Context Block	2019	1
Factorized Dense Synthesized Attention	2020	1
CBAM	2018	1
Routing Attention	2020	1
Random Synthesized Attention	2020	1
Attention-augmented Convolution	2019	1
Multiplicative Attention	2015	1

Where the legs grow from

- Machine Translation!
- But you know it already, right?
- So it's about *seq2seq* for translation task, but a little augmented.

Tags, Sections, Whatever

- Attention
- Self-Attention
- Other types of attention
- Some applications on some other fields, not only text
- Modern Attention and esoteric maybe.

Attention

Global/Local

Hard/Soft

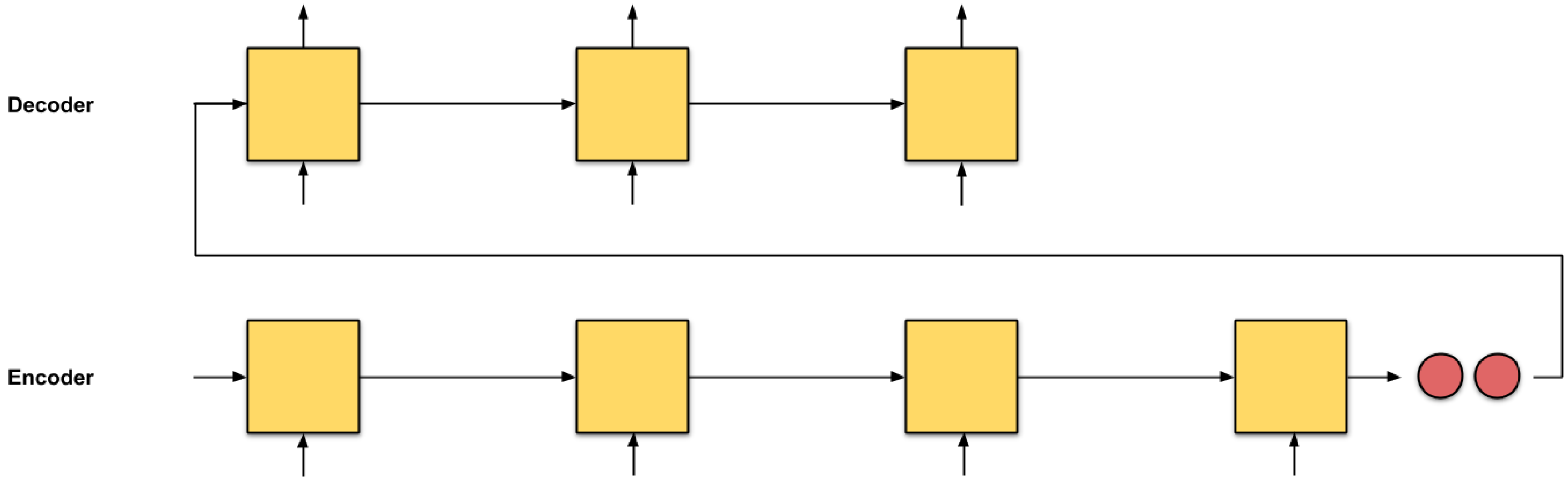
Other types like dot product))))0))0

So attention, ha?

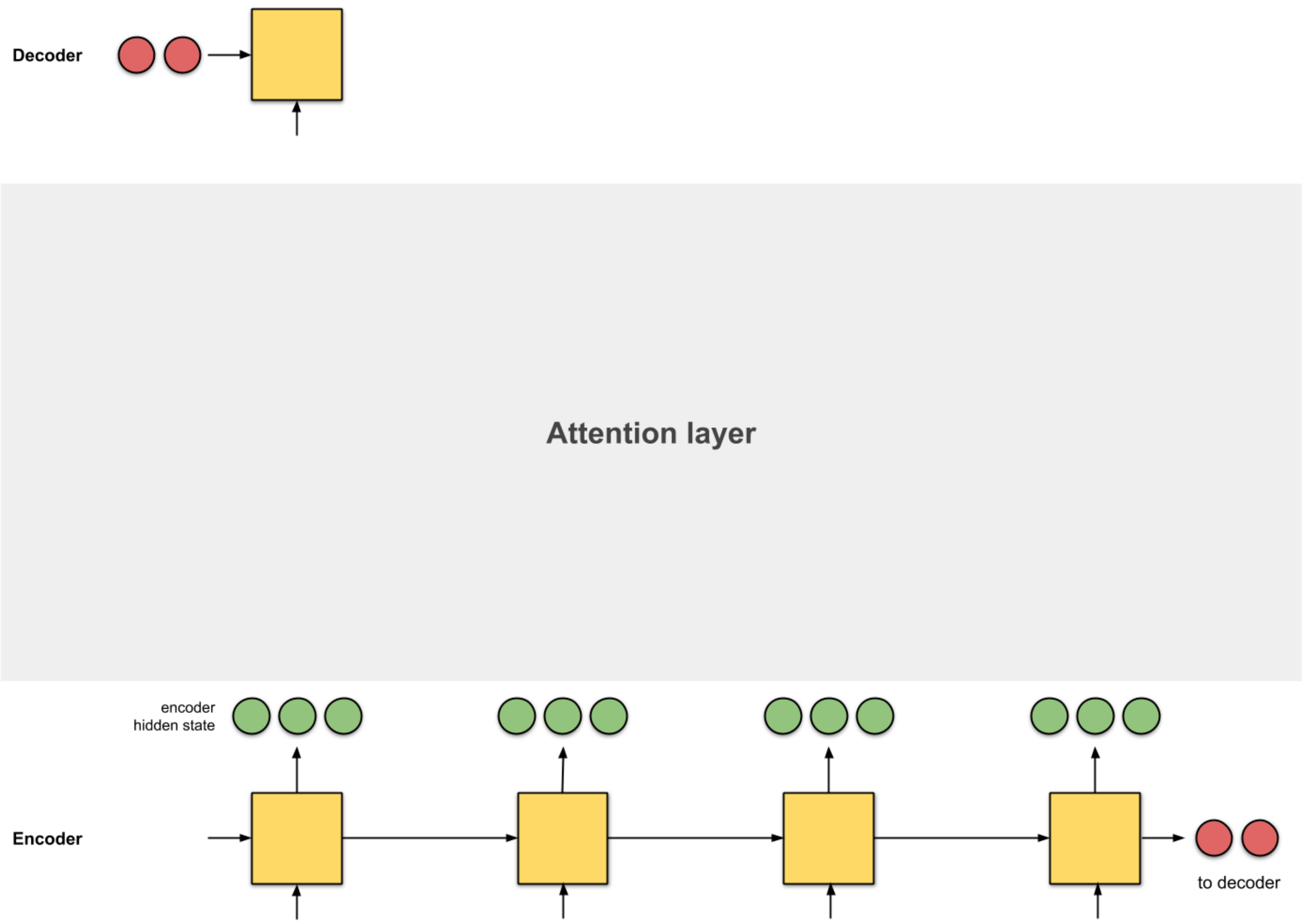
Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

1. Neural Machine Translation By Jointly Learning To Align And Translate. Dzmitry Bahdanau. et al 14
2. Effective Approaches to Attention-based Neural Machine Translation. Minh-Thang Luong et al 15
3. Attention Is All You Need. Ashish Vaswani et al. 17

Seq2seq



Seq2seq + attention

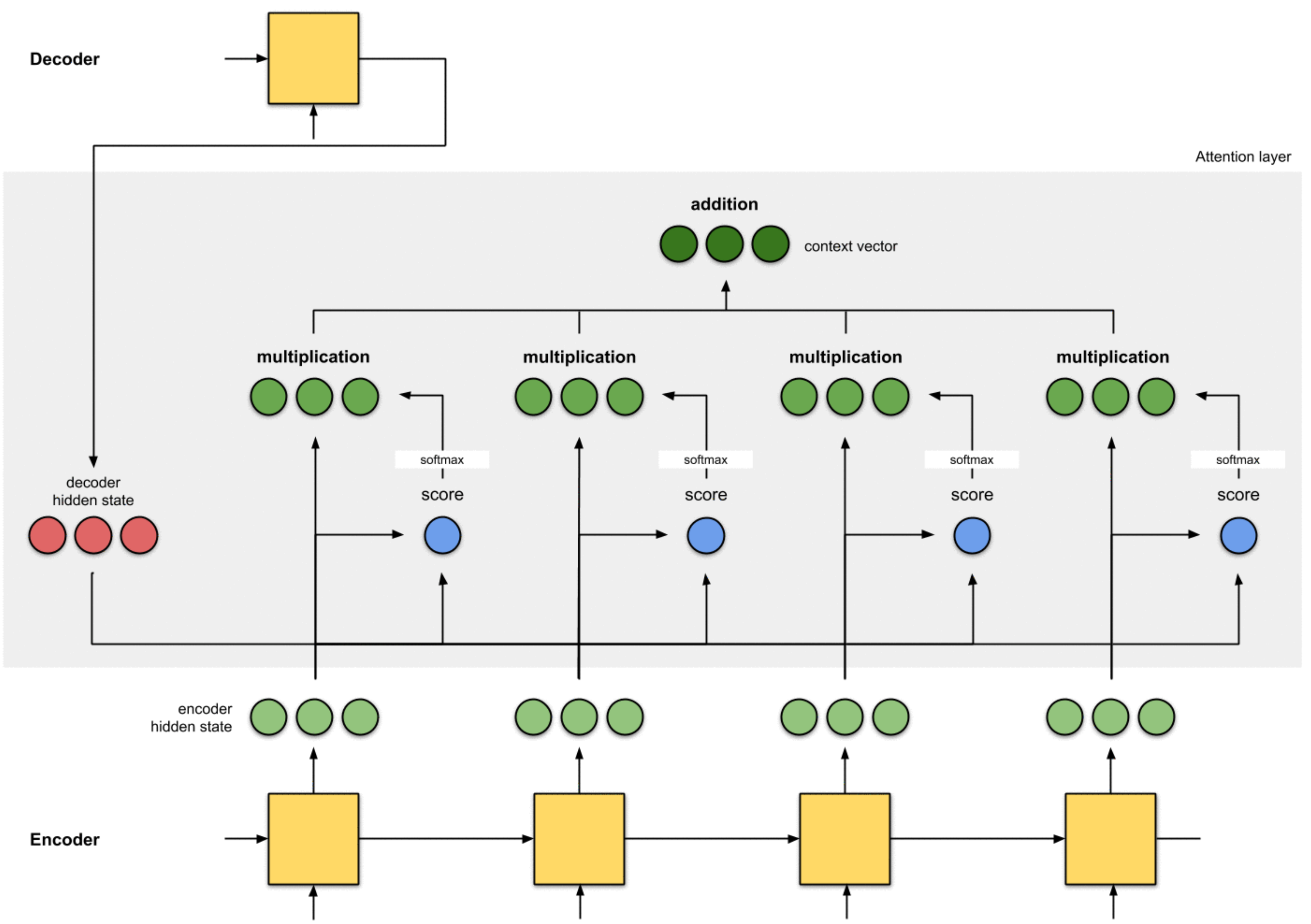


Aligner

- In fact in paper it calls alignment model

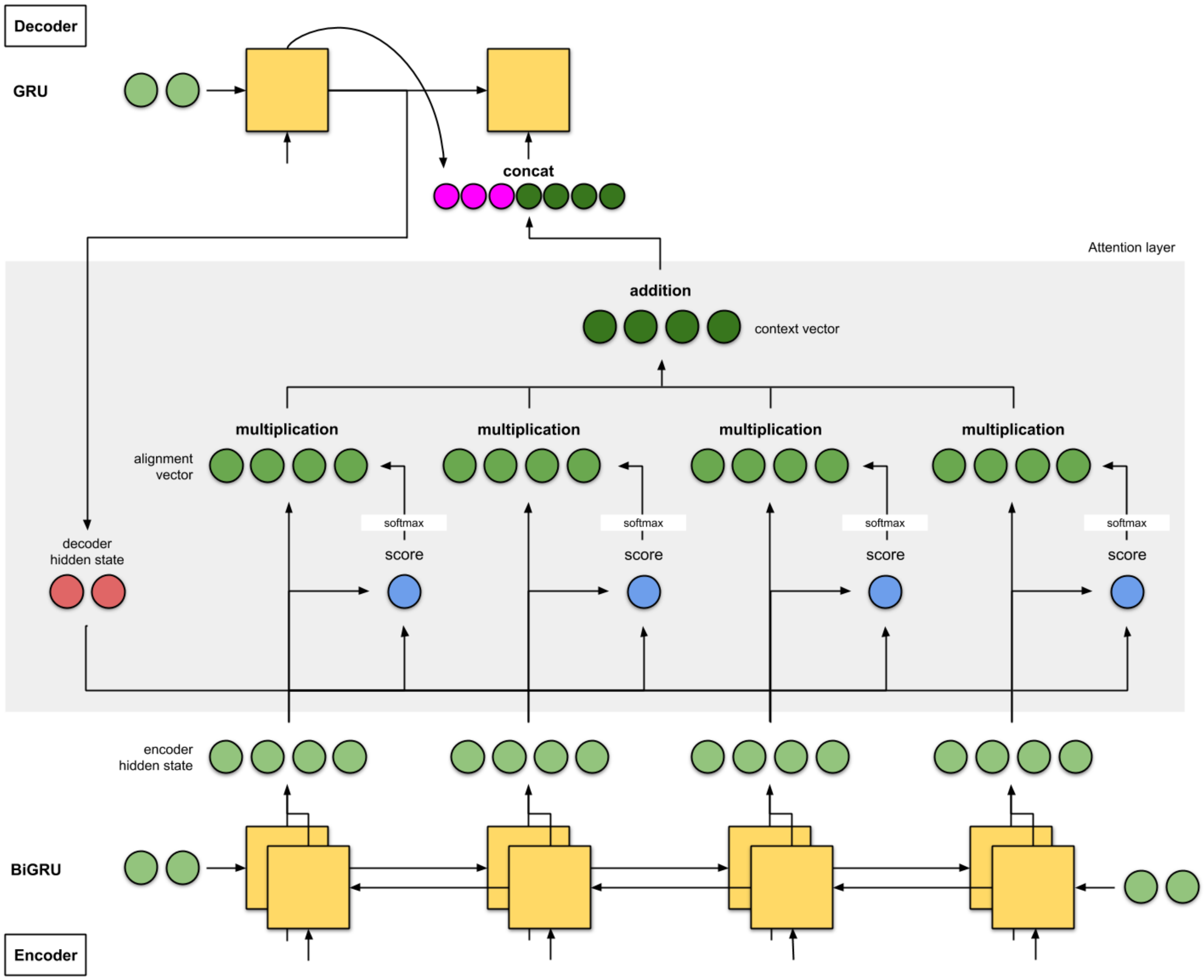
alignment
means matching segments of original text with their corresponding segments of the translation.

Encoder


Decoder
Attention layer
addition
context vector
multiplication
multiplication
multiplication
multiplication
softmax
softmax
softmax
softmax
score
score
score
score
decoder
hidden state
encoder
hidden state
Encoder

Additive

- **Bahdanau et. al 2014**



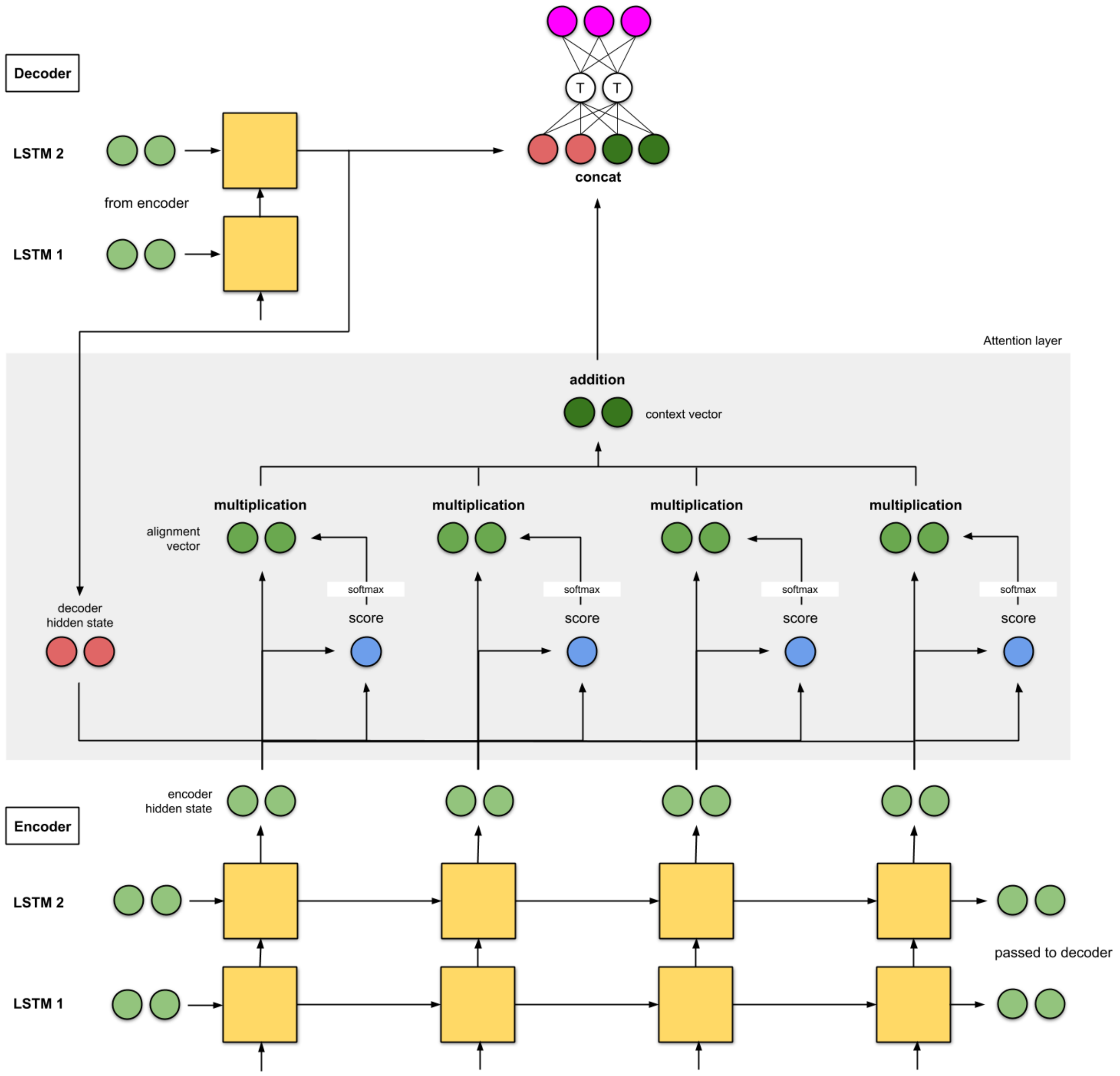
Local

- Luong et. al 2015

- There was a lot types of attention in paper study
- General
- Additive
- Dot Product
- Local

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

$$\boldsymbol{a}_t = \mathrm{softmax}(\boldsymbol{W_a} \boldsymbol{h}_t) \qquad \textit{location}$$



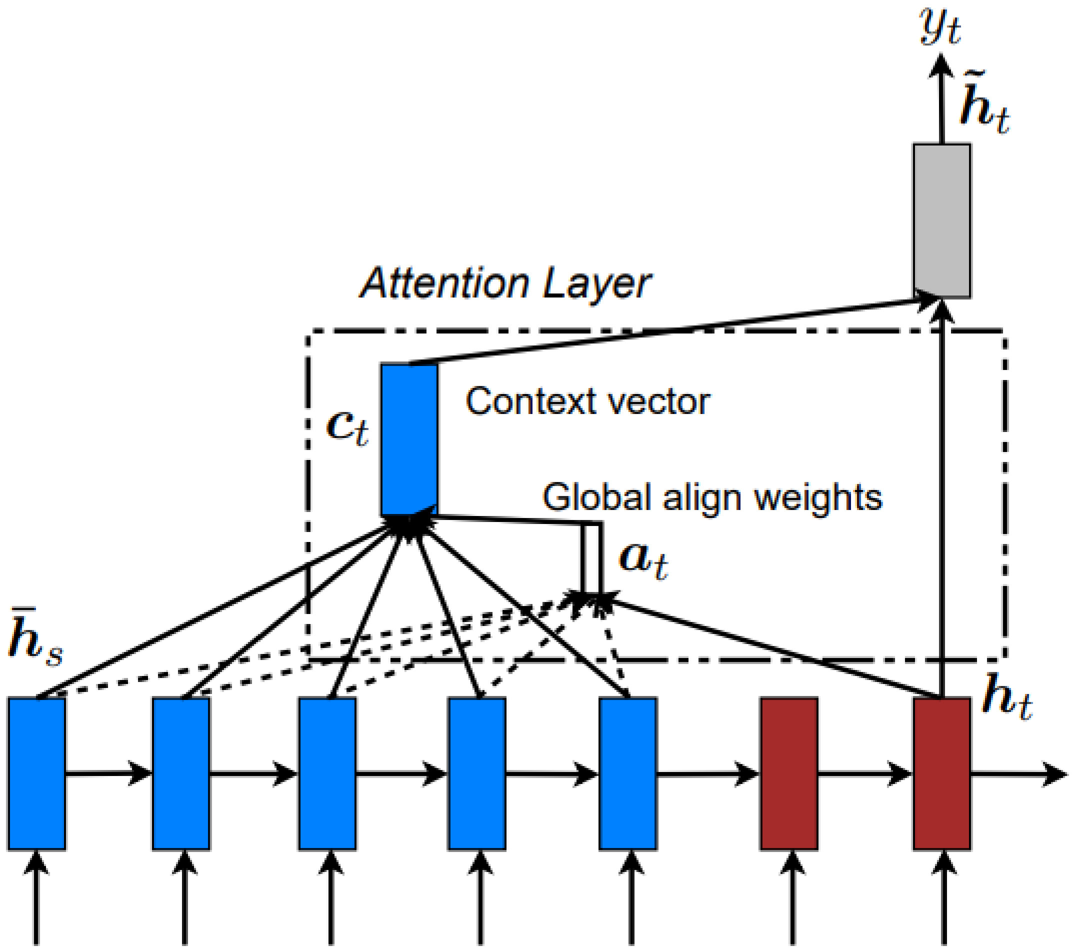


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed as the weighted average, according to $\boldsymbol{a}_t$, over all the source states.

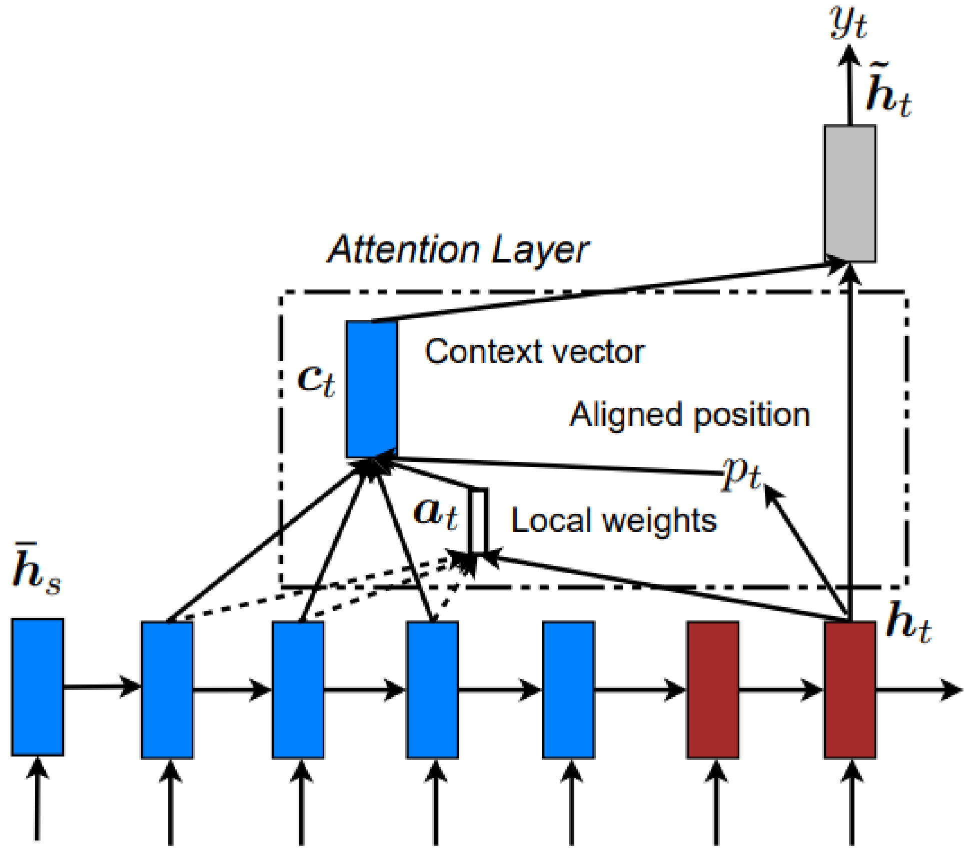


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a}_t$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

Formalization (one more time)

Alignment model

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

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

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

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

$$\boldsymbol{h}_j = f(\boldsymbol{h}_{j-1}, \boldsymbol{s}),$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

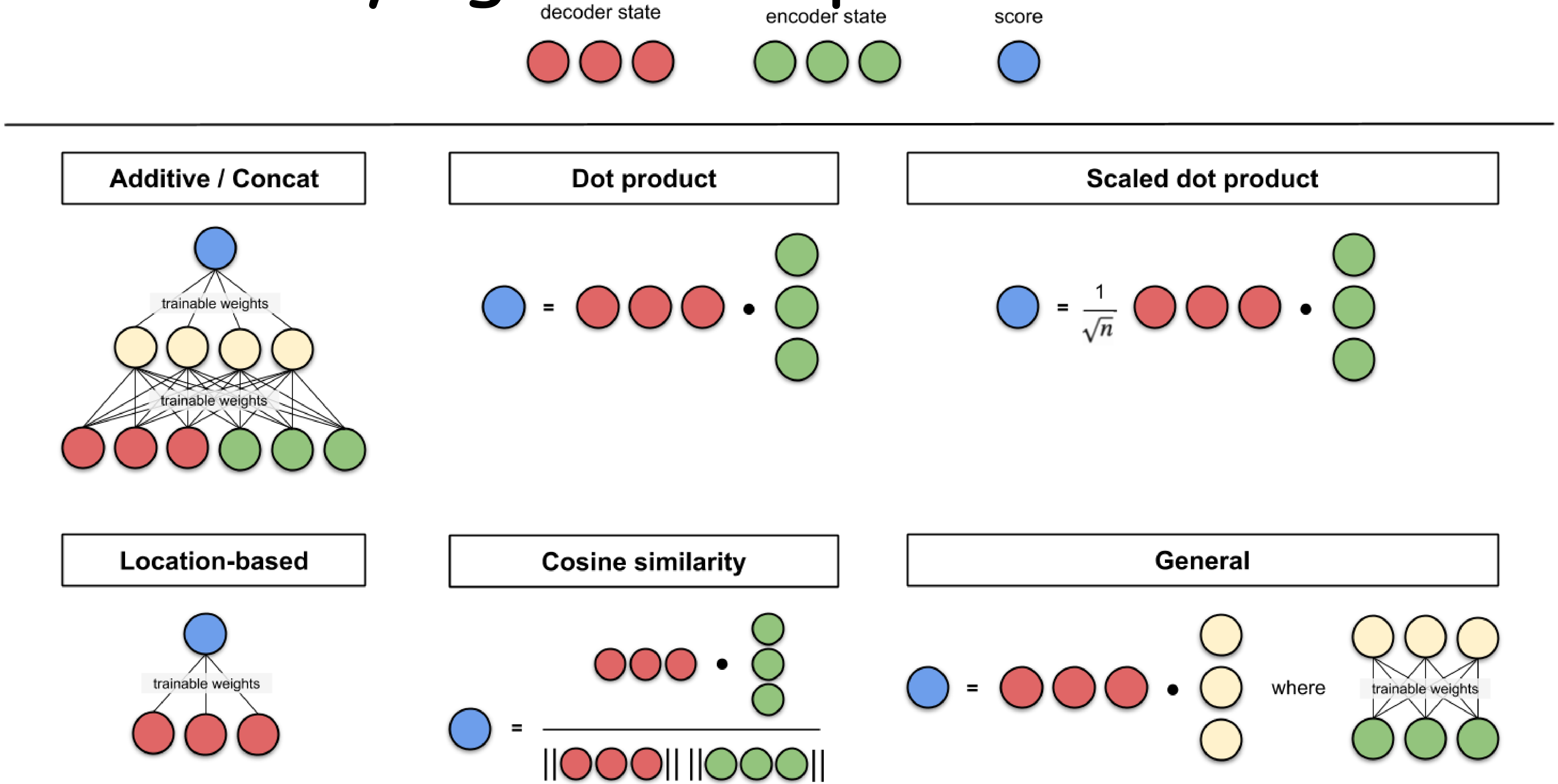
$$\begin{aligned} \boldsymbol{a}_t(s) &= \mathrm{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \end{aligned}$$

Global

Local

$$\boldsymbol{a}_t = \mathrm{softmax}(\boldsymbol{W_a} \boldsymbol{h}_t)$$

Summary again that pictures



Summary again that pictures

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

NAKONEC TO KARTINKI EPAT (images finally)

- In paper

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

by **Kelvin Xu et al 15**

authors use img2seq model to caption images and they augmented it with … attention!

So how it was?

Soft/Hard (pic first!)

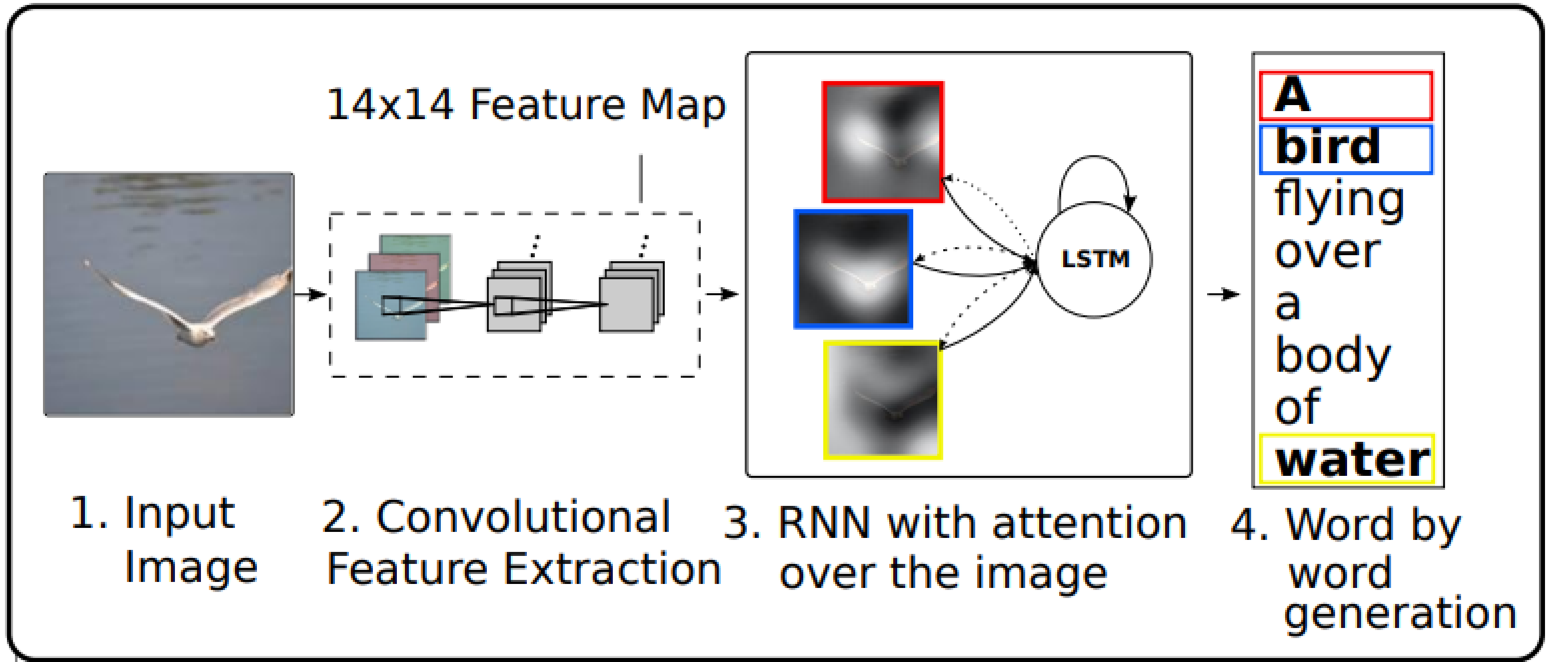
"A woman is throwing a frisbee in a park." Xu et al. 2015

Soft/Hard

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

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

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



Soft Attention

Attention score is used as weights in the weighted average context vector calculation. This is a differentiable function.

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

Hard Attention

Attention score is used as the probability of the i-th location getting selected. We could use a simple argmax to make the selection, but it is not differentiable and so complex techniques are employed.

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

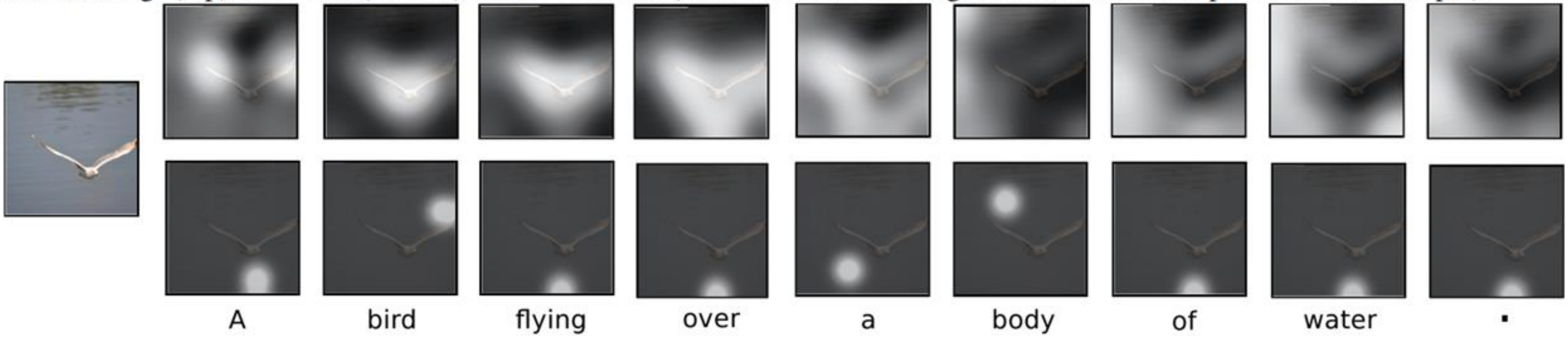
$$p(s_{t,i} = 1 \mid s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

$$\tilde{s_t^n} \sim \text{Multinoulli}_L(\{\alpha_i^n\})$$

"a" represents encoder/input hidden states, "α" represents the attention scores, "$s_{t,i}$" is a one-hot variable with "1" if "i-th" location is to be selected.

Soft/Hard

Figure 3. Visualization of the attention for each generated word. The rough visualizations obtained by upsampling the attention weights and smoothing. (top)"soft" and (bottom) "hard" attention (note that both models generated the same captions in this example).



And there was many others that I decided not to tell you about…

but next there was a game changer?

Self-Attention

Single, Multihead, Sparse, Huyars, Linear

Transformers, GANs, CNNs, Graphs

And other shit

Intra-Attention

- Initially was introduced for machine reading and language modeling in paper

Long Short-Term Memory-Networks for Machine Reading

by Jianpeng Cheng et al 16

Intra-Attention

$$a_i^t = v^{\mathrm{T}} \tanh(W_h h_i + W_x x_t + W_{\tilde{h}} \tilde{h}_{t-1})$$

$$s_i^t = \mathrm{softmax}(a_i^t)$$

$$\begin{bmatrix} \tilde{h}_t \\ \tilde{c}_t \end{bmatrix} = \sum_{i=1}^{t-1} s_i^t \cdot \begin{bmatrix} h_i \\ c_i \end{bmatrix}$$

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \cdot [\tilde{h}_t, x_t]$$

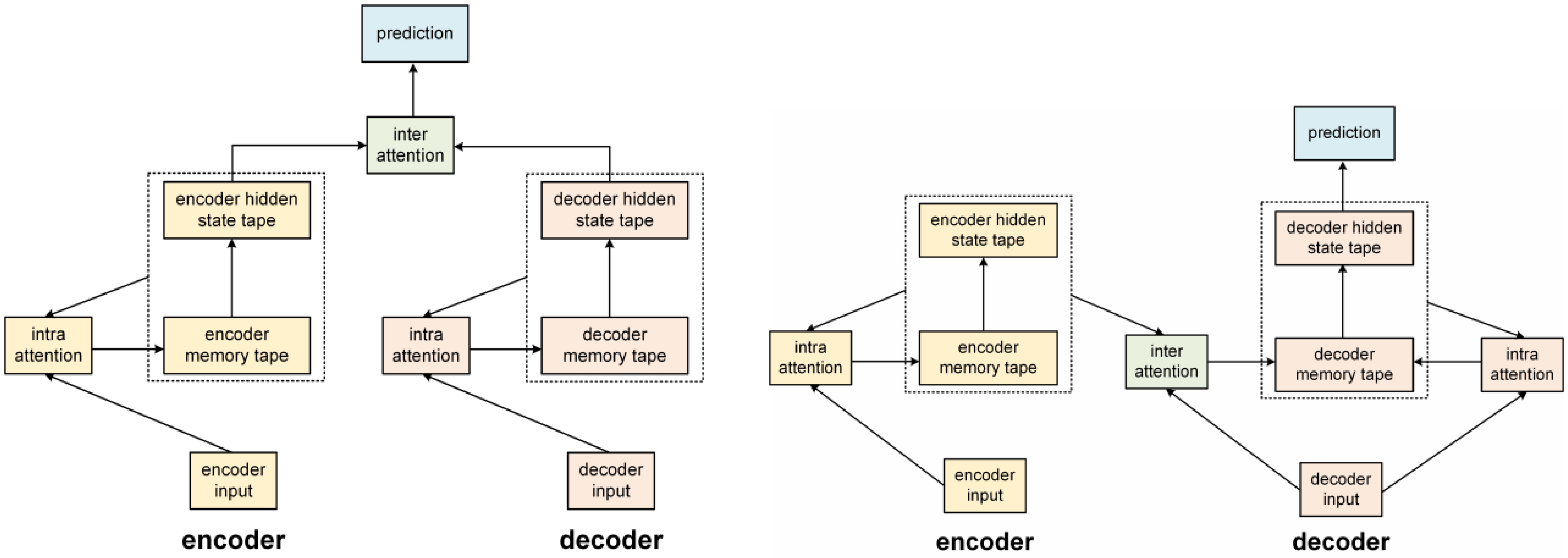
$$c_t = f_t \odot \tilde{c}_t + i_t \odot \hat{c}_t$$

$$h_t = o_t \odot \tanh(c_t)$$

Something familiar in many ways?

$$a_{i,k+1}^t = v^{\mathrm{T}} \tanh(W_h h_i^{k+1} + W_l h_t^k + W_{\tilde{h}} \tilde{h}_{t-1}^{k+1})$$

Intra-Attention



(a) Decoder with shallow attention fusion.

(b) Decoder with deep attention fusion.

Figure 3: LSTMNs for sequence-to-sequence modeling. The encoder uses intra-attention, while the decoder incorporates both intra- and inter-attention. The two figures present two ways to combine the intra- and inter-attention in the decoder.

Intra-Attention

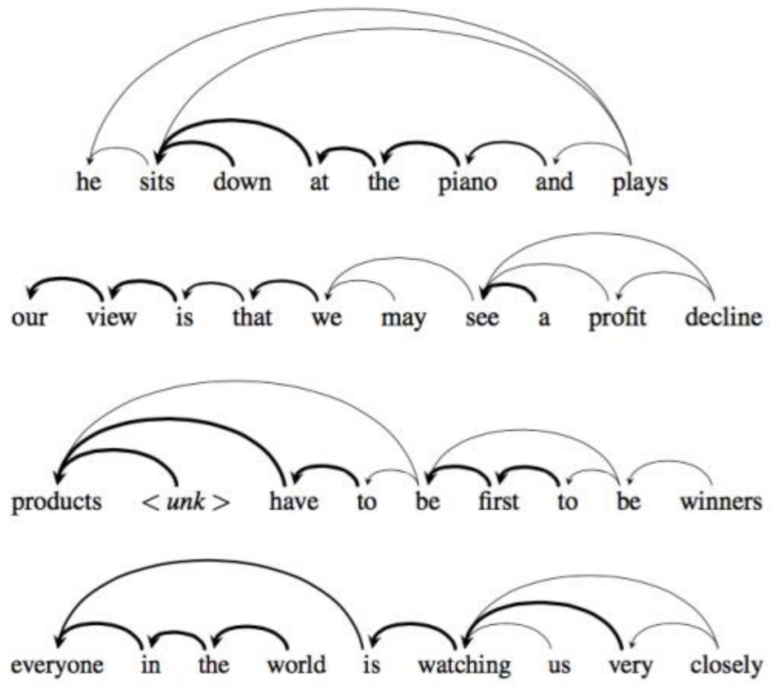


Figure 4: Examples of intra-attention (language modeling). Bold lines indicate higher attention scores. Arrows denote which word is being focused when attention is computed, but not the direction of the relation.

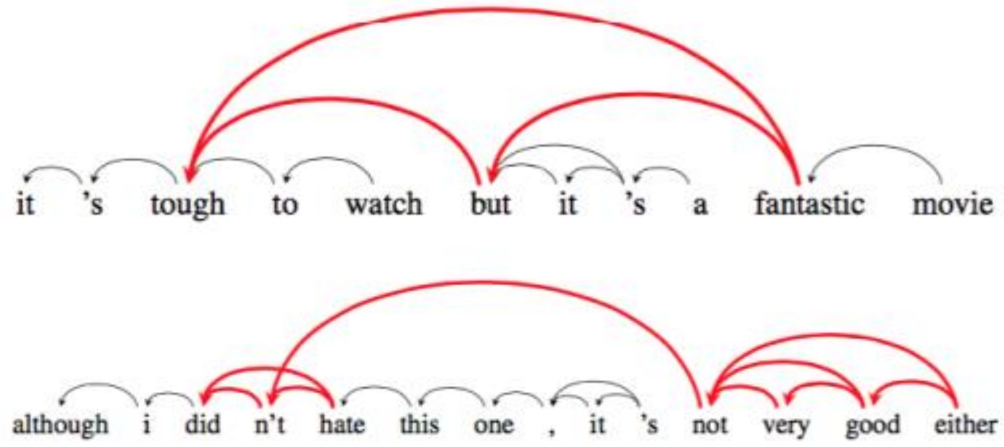


Figure 5: Examples of intra-attention (sentiment analysis). Bold lines (red) indicate attention between sentiment important words.

Intra-Attention

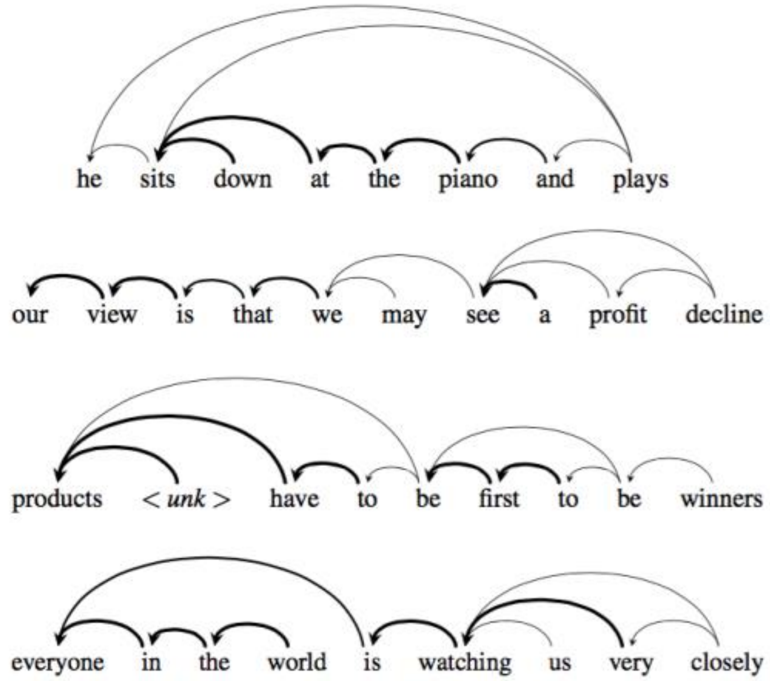


Figure 4: Examples of intra-attention (language modeling). Bold lines indicate higher attention scores. Arrows denote which word is being focused when attention is computed, but not the direction of the relation.

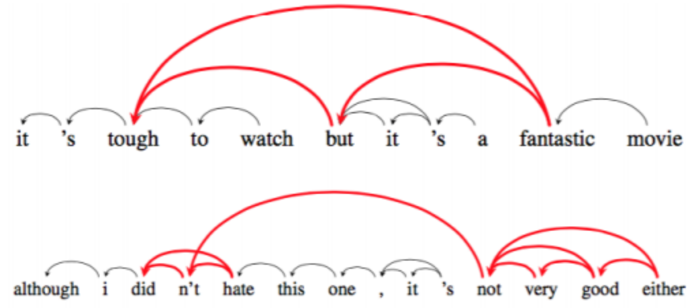


Figure 5: Examples of intra-attention (sentiment analysis). Bold lines (red) indicate attention between sentiment important words.

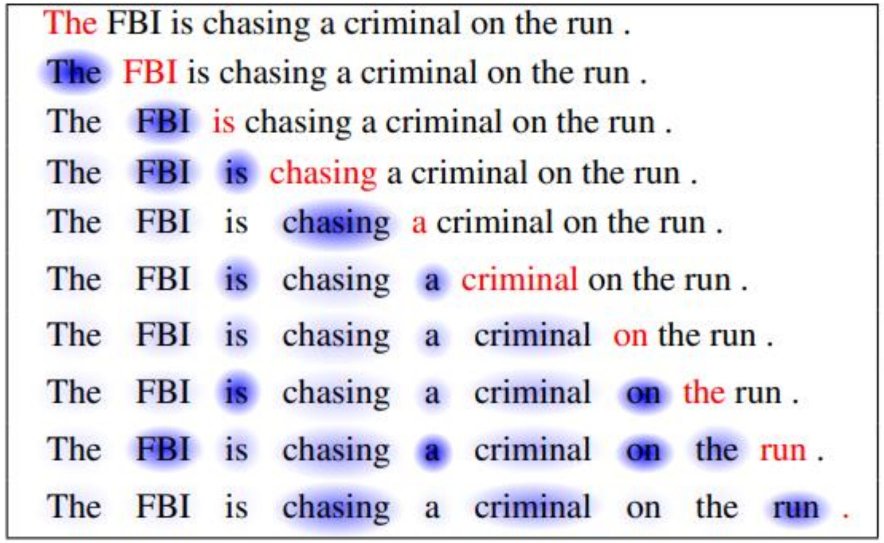


Figure 1: Illustration of our model while reading the sentence *The FBI is chasing a criminal on the run.* Color *red* represents the current word being fixated, *blue* represents memories. Shading indicates the degree of memory activation.

Self-Attention

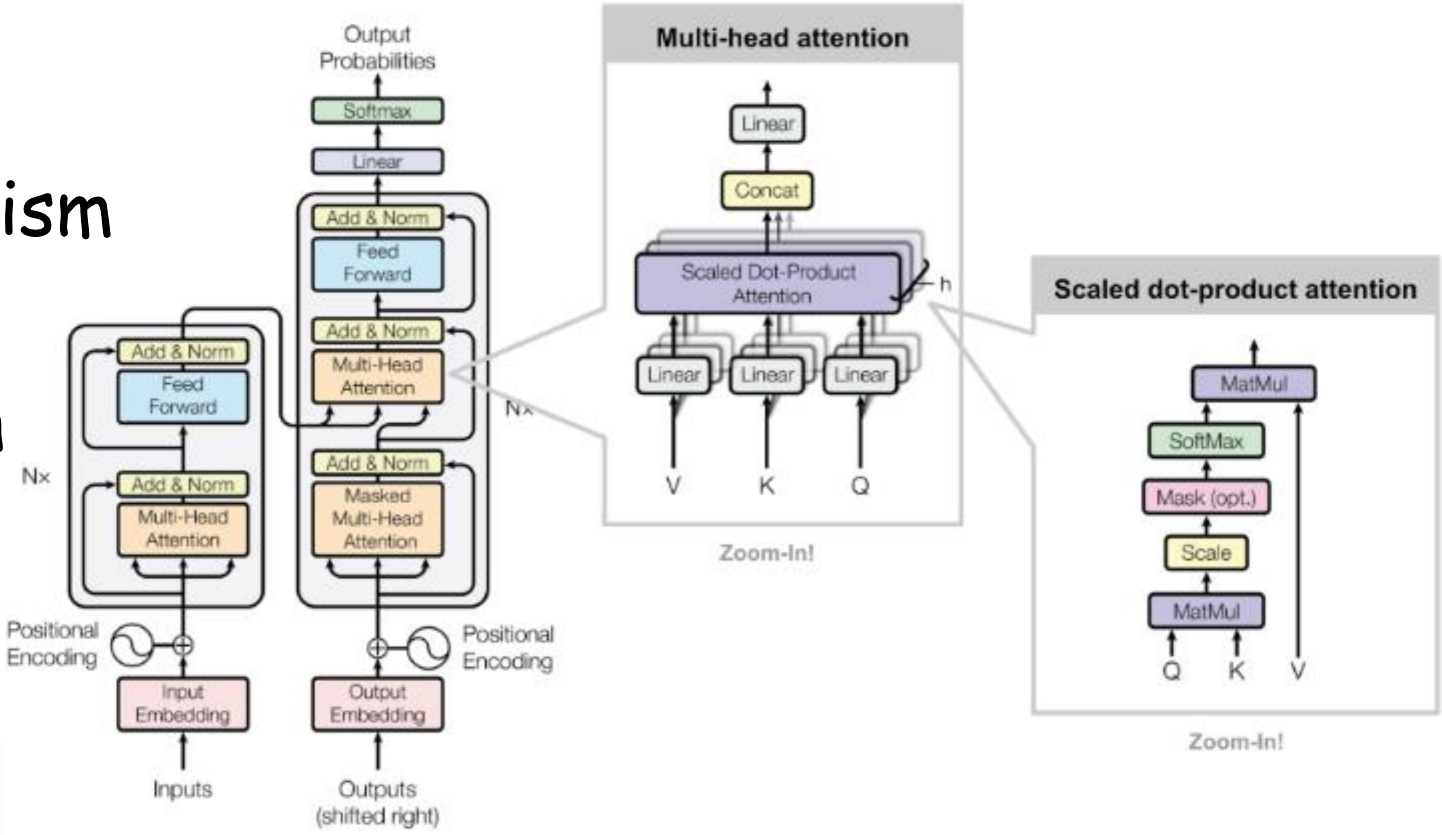
ATTENTION IS ALL YOU NEED!!!111one

(You've been waiting that, right?)

<u>Vaswani, et al., 2017</u>

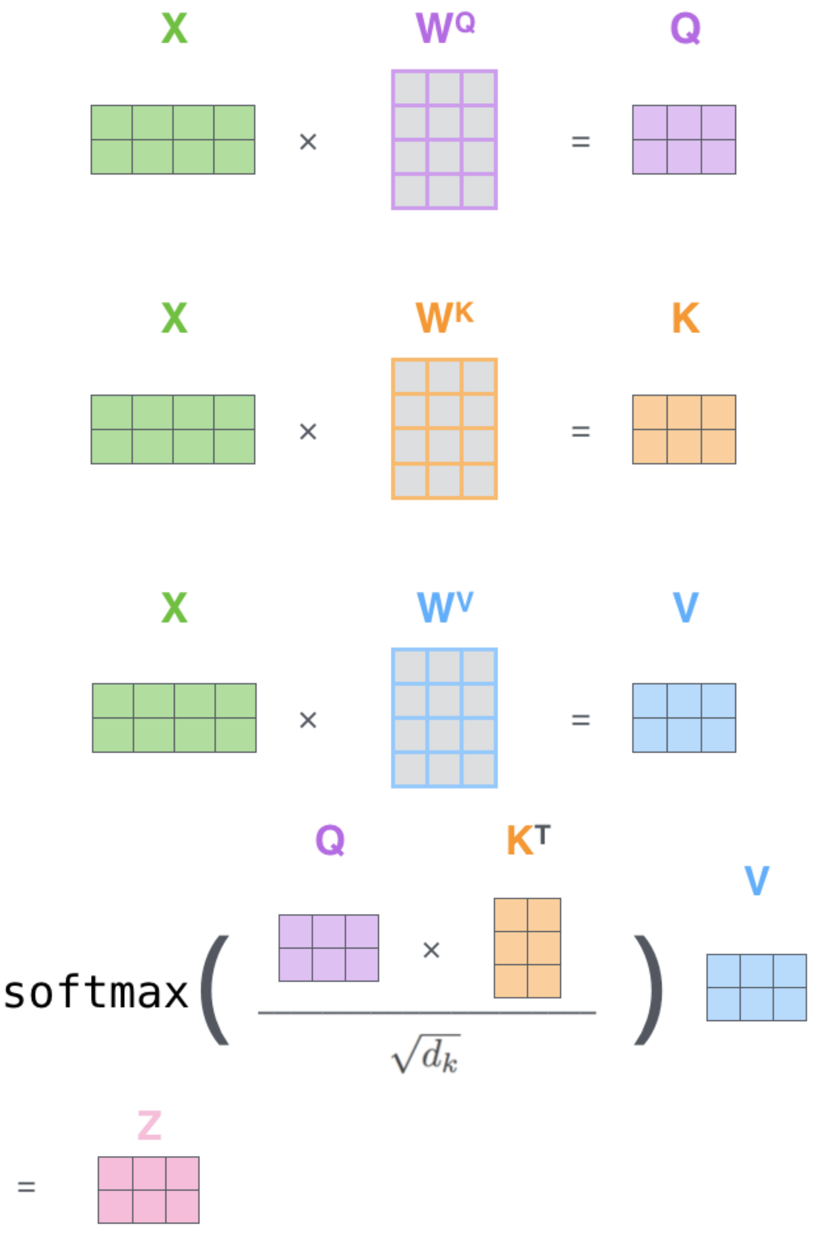
Self-Attention (Transformer)

- Very much basenka
- Key, Value, Query formalism
- So Multihead
- Easiest type of attention

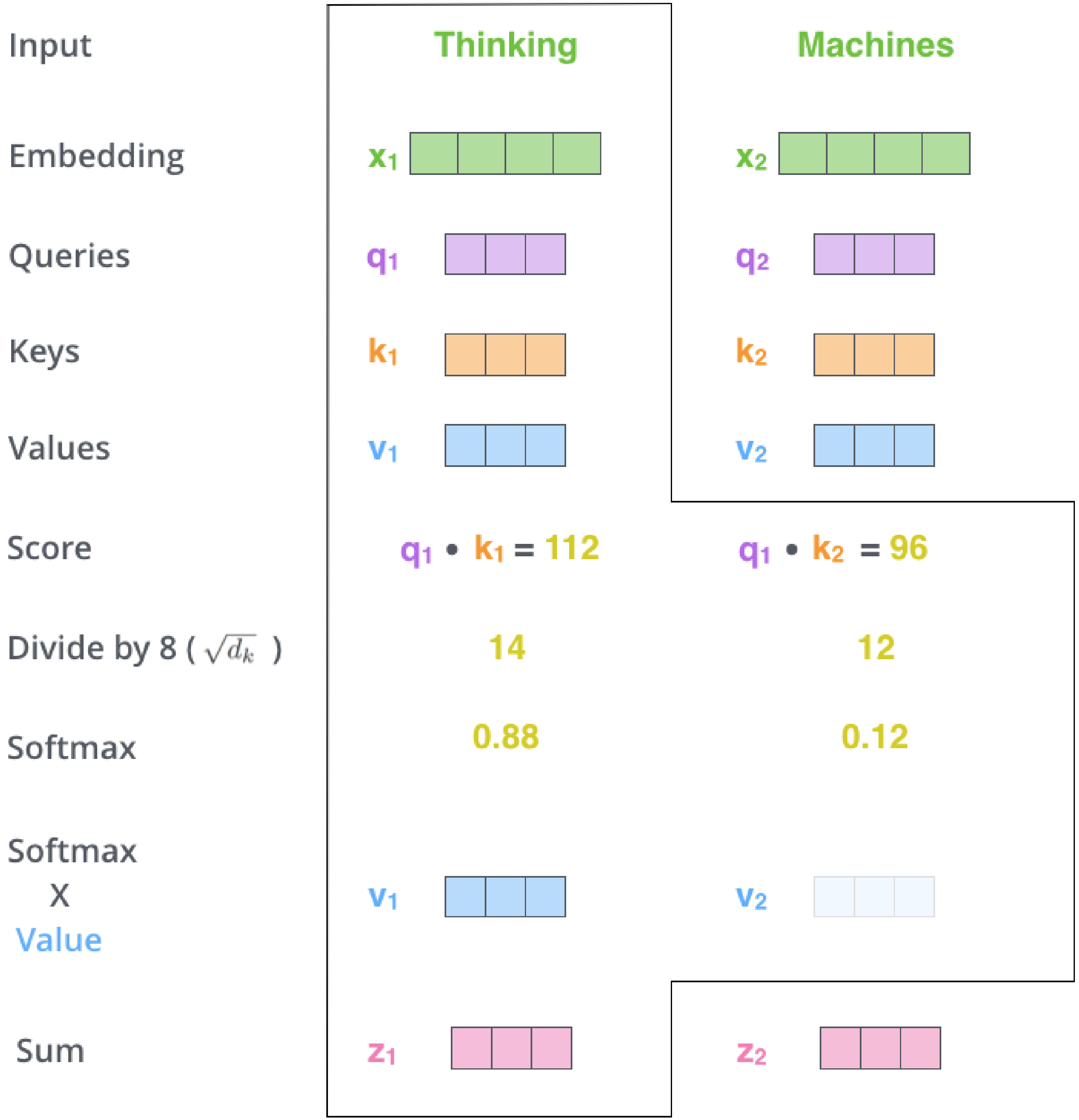


$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{n}}\right)\mathbf{V}$$

Les't deep dive



The self-attention calculation in matrix form



Les't deep dive

Self-attention

input #1

1	0	1	0

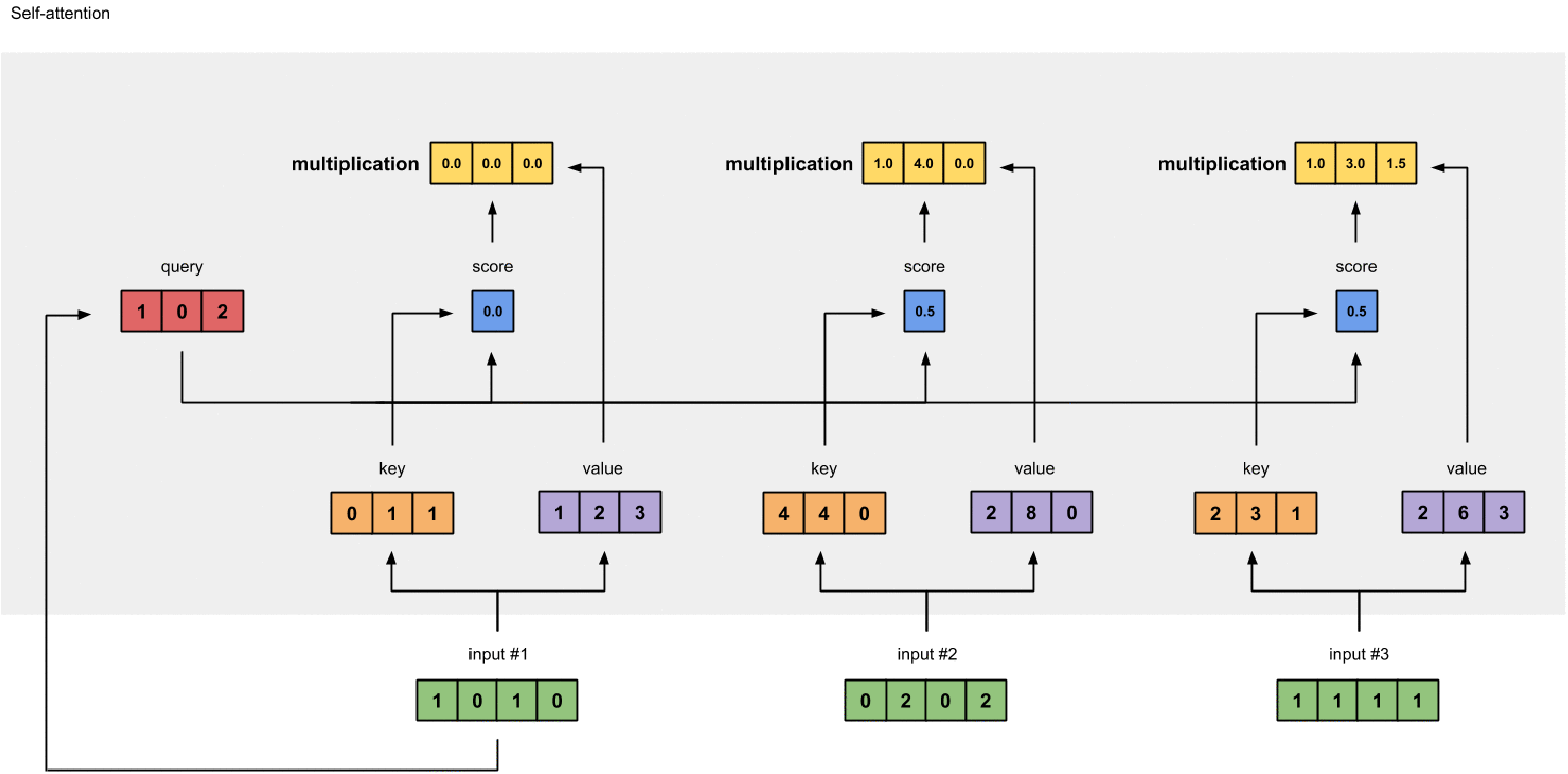
input #2

0	2	0	2

input #3

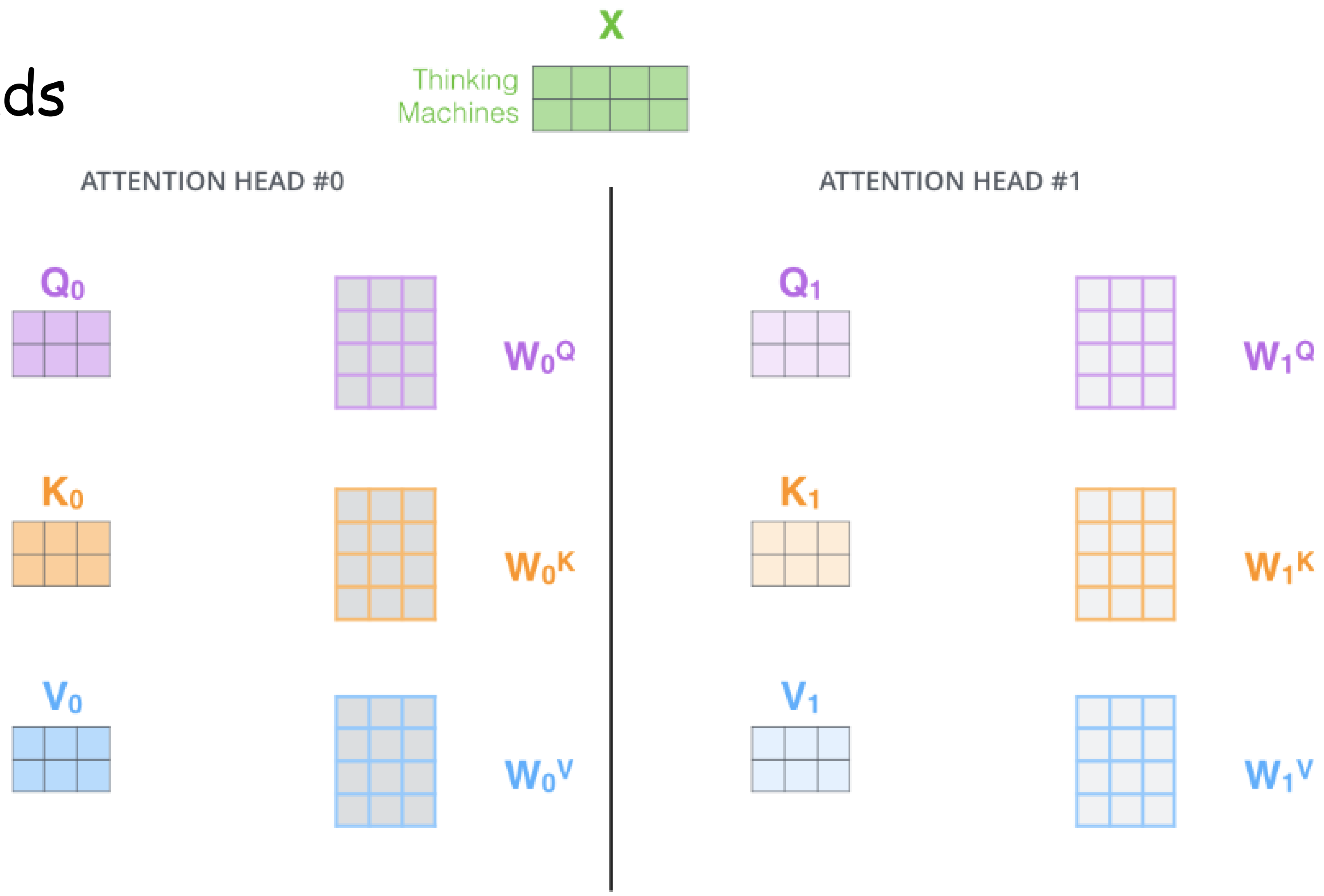
1	1	1	1

Les't deep dive



Les't deep dive

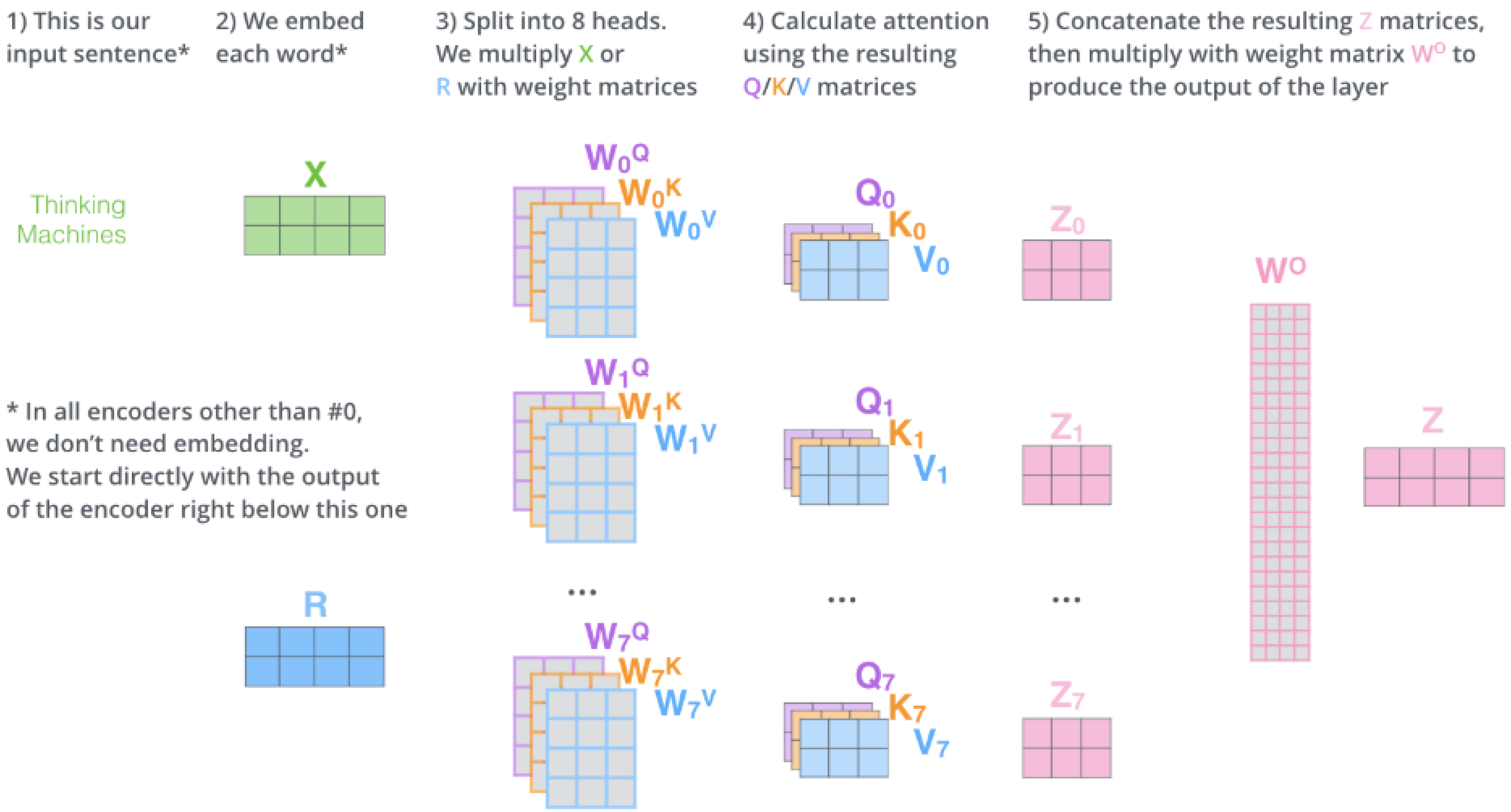
Add some heads



With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

Les't deep dive

Add some heads



Self-Attention

Some code

```python
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
```

```python
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)
```

Self-Attention with relation aware

In paper

Self-Attention with Relative Position Representations

by Peter Shaw at el 18 propose some kind of positional encoding in attention layer. The edge between input elements xi and xj is represented by vectors $a^V_{ij}, a^K_{ij} \in \mathbb{R}^{d_a}$.

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + a^V_{ij}) \qquad e_{ij} = \frac{x_i W^Q (x_j W^K + a^K_{ij})^T}{\sqrt{d_z}}$$

$$a^K_{ij} = w^K_{\mathrm{clip}(j-i,k)}$$
$$a^V_{ij} = w^V_{\mathrm{clip}(j-i,k)}$$
$$\mathrm{clip}(x,k) = \max(-k, \min(k,x))$$

Local Self-Attention (Images!)

In paper **Image Transformer** by Niki Parmar et al 18

They use transformer with local self-attention to produce images with super resolution task.

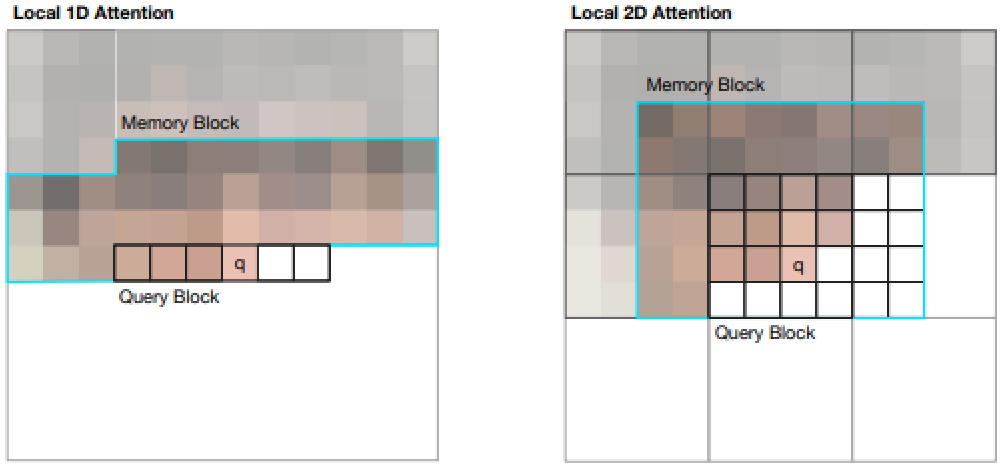


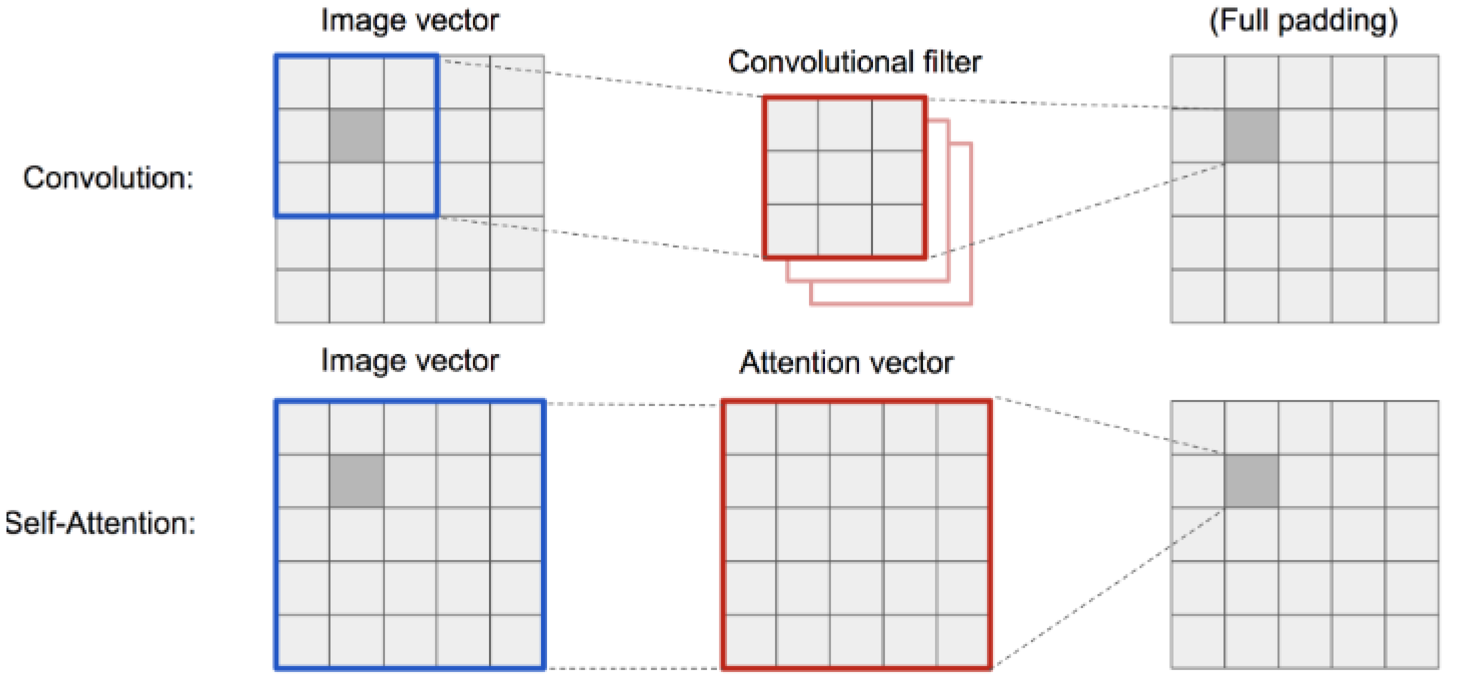
Figure 2. The two different conditional factorizations used in our experiments, with 1D and 2D local attention on the left and right, respectively. In both, the image is partitioned into non-overlapping query blocks, each associated with a memory block covering a superset of the query block pixels. In every self-attention layer, each position in a query block attends to all positions in the memory block. The pixel marked as q is the last that was generated. All channels of pixels in the memory and query blocks shown in white have masked attention weights and do not contribute to the next representations of positions in the query block. While the effective receptive field size in this figure is the same for both schemes, in 2D attention the memory block contains a more evenly balanced number of pixels next to and above the query block, respectively.

Self-Attention GAN (Images!)

Finally! The TRUE COMPUTER VISION.

SAGAN

Zhang et al., 2018



Self-Attention GAN (Images!)

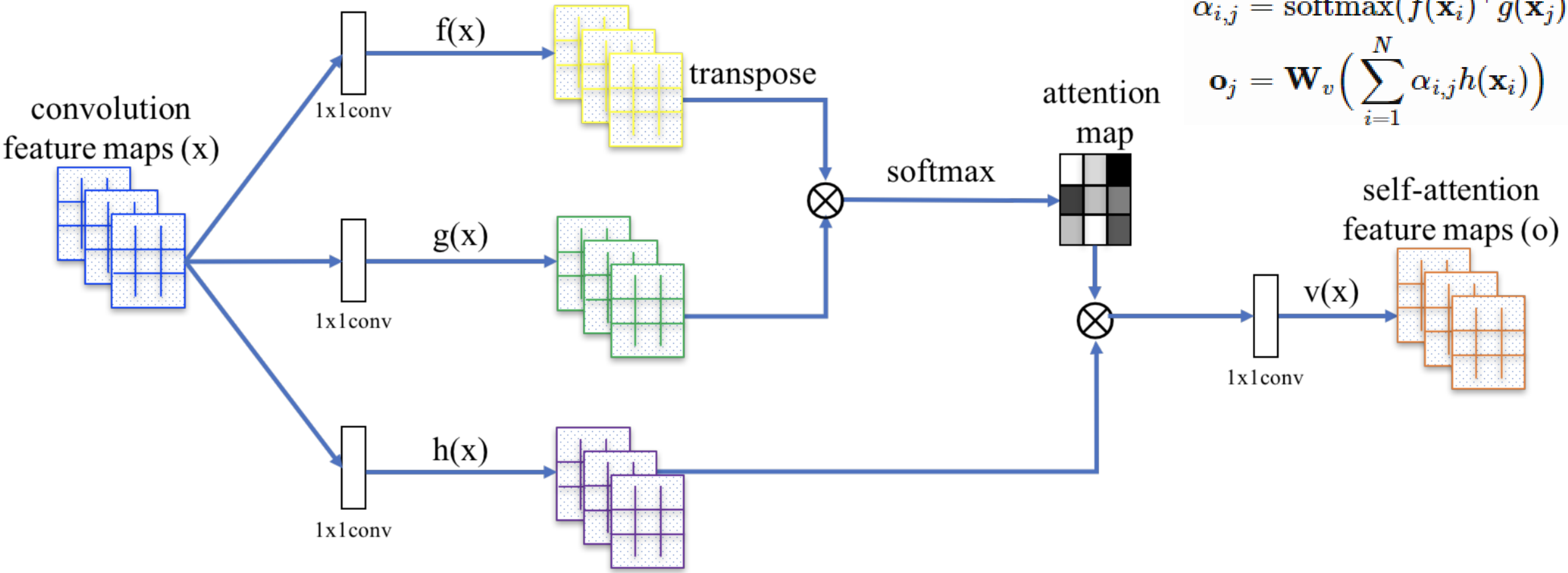
Key: $f(\mathbf{x}) = \mathbf{W}_f \mathbf{x}$

Query: $g(\mathbf{x}) = \mathbf{W}_g \mathbf{x}$

Value: $h(\mathbf{x}) = \mathbf{W}_h \mathbf{x}$

$$\alpha_{i,j} = \text{softmax}(f(\mathbf{x}_i)^\top g(\mathbf{x}_j))$$

$$\mathbf{o}_j = \mathbf{W}_v \left(\sum_{i=1}^{N} \alpha_{i,j} h(\mathbf{x}_i) \right)$$



Self-Attention GAN (Images!)

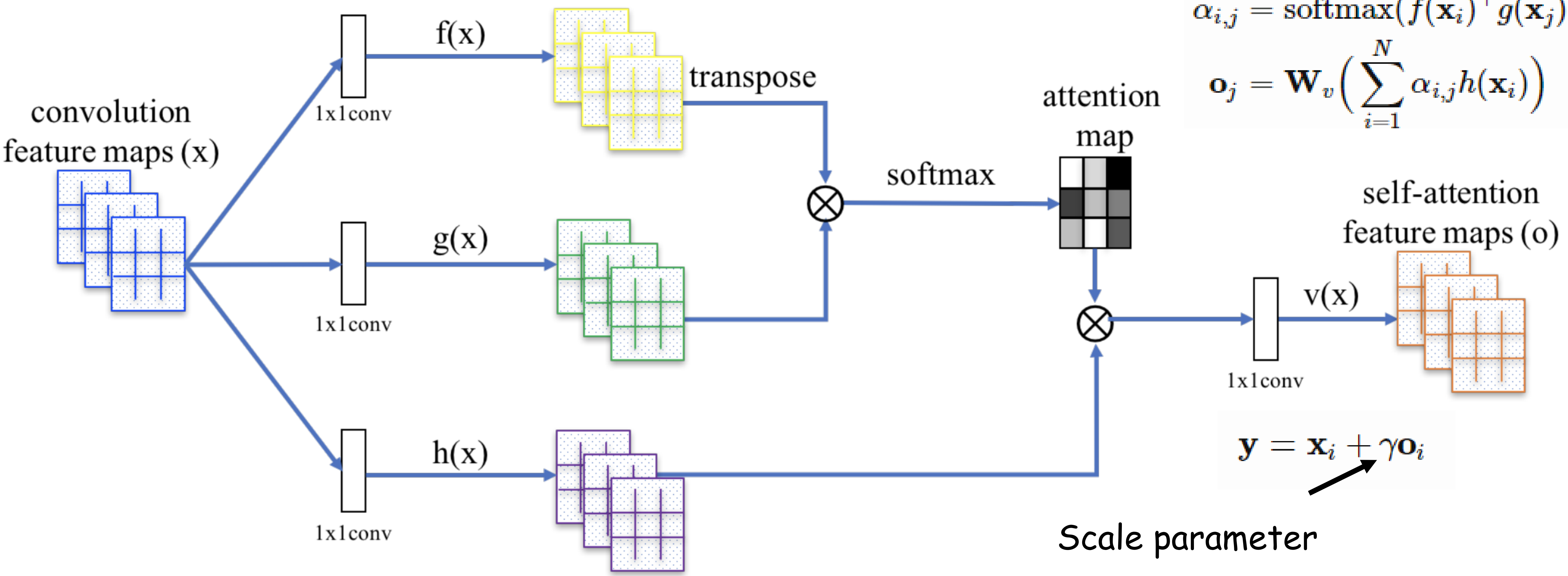
Key: $f(\mathbf{x}) = \mathbf{W}_f\mathbf{x}$

Query: $g(\mathbf{x}) = \mathbf{W}_g\mathbf{x}$

Value: $h(\mathbf{x}) = \mathbf{W}_h\mathbf{x}$

$$\alpha_{i,j} = \text{softmax}(f(\mathbf{x}_i)^\top g(\mathbf{x}_j))$$

$$\mathbf{o}_j = \mathbf{W}_v\left(\sum_{i=1}^{N}\alpha_{i,j}h(\mathbf{x}_i)\right)$$



$$\mathbf{y} = \mathbf{x}_i + \gamma\mathbf{o}_i$$

Scale parameter

Self-Attention GAN (Images!)

Figure 1. The proposed SAGAN generates images by leveraging complementary features in distant portions of the image rather than local regions of fixed shape to generate consistent objects/scenarios. In each row, the first image shows five representative query locations with color coded dots. The other five images are attention maps for those query locations, with corresponding color coded arrows summarizing the most-attended regions.

Simple Self-Attention (Images!)

SAGAN

```
class SelfAttention(nn.Module):

 "Self attention layer for nd."

 def __init__(self, n_channels:int):
     super().__init__()
     self.query = conv1d(n_channels, n_channels//8)
     self.key   = conv1d(n_channels, n_channels//8)
     self.value = conv1d(n_channels, n_channels)
     self.gamma = nn.Parameter(tensor([0.]))

 def forward(self, x):
     #Notation from https://arxiv.org/pdf/1805.08318.pdf
     size = x.size()
     x = x.view(*size[:2],-1)
     f,g,h = self.query(x),self.key(x),self.value(x)
     beta = F.softmax(torch.bmm(f.permute(0,2,1).contiguous(), g), dim=1)
     o = self.gamma * torch.bmm(h, beta) + x
     return o.view(*size).contiguous()
```

Simple Self Attention)))

```
class SimpleSelfAttention(nn.Module):

def __init__(self, n_in:int, ks=1):#, n_out:int):
    super().__init__()
    self.conv = conv1d(n_in, n_in, ks, padding=ks//2, bias=False)
    self.gamma = nn.Parameter(tensor([0.]))

def forward(self,x):
    size = x.size()
    x = x.view(*size[:2],-1)
    o = torch.bmm(x.permute(0,2,1).contiguous(),self.conv(x))
    o = self.gamma * torch.bmm(x,o) + x

    return o.view(*size).contiguous()
```

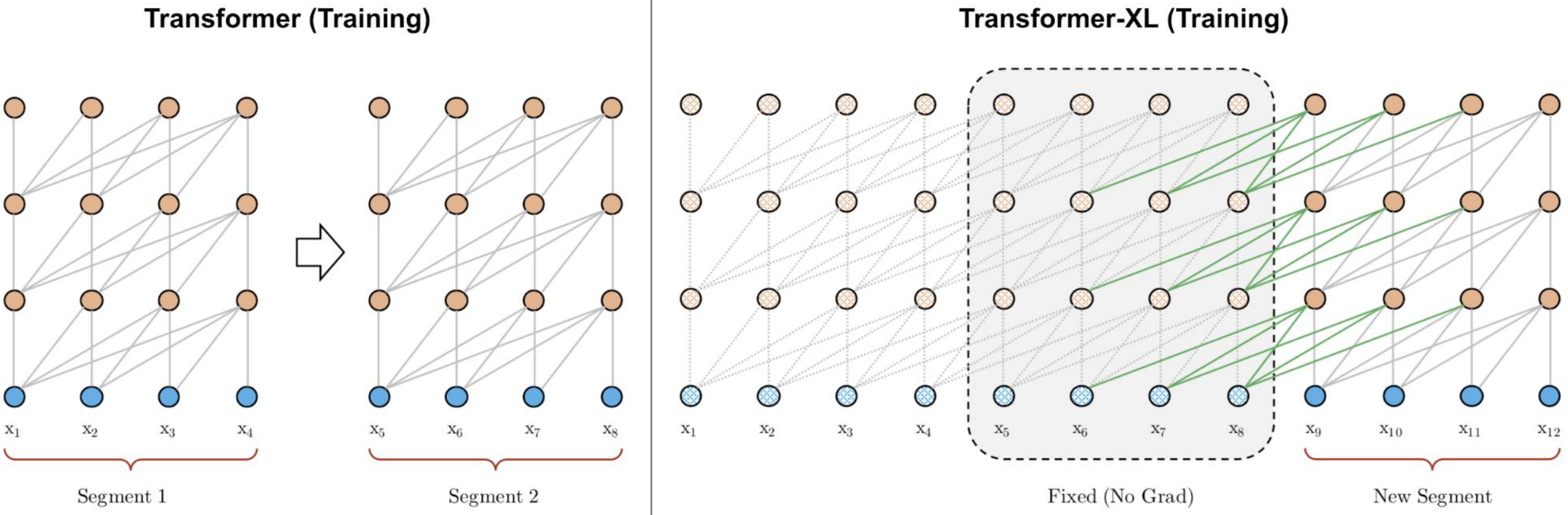
https://github.com/sdoria/SimpleSelfAttention

Modern Attention

It’s all about transformers of course!

Longer Attention Span (Transformer-XL)

- Transformer-XL (Dai et al., 2019; “XL” means “extra long”)



A comparison between the training phrase of vanilla Transformer & Transformer-XL with a segment length 4.

Longer Attention Span (Transformer-XL)

The long attention span

$$
\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} &= [\text{stop-gradient}(\mathbf{h}_{\tau}^{(n-1)}) \circ \mathbf{h}_{\tau+1}^{(n-1)}] \\
\mathbf{Q}_{\tau+1}^{(n)} &= \mathbf{h}_{\tau+1}^{(n-1)} \mathbf{W}^q \\
\mathbf{K}_{\tau+1}^{(n)} &= \widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} \mathbf{W}^k \\
\mathbf{V}_{\tau+1}^{(n)} &= \widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} \mathbf{W}^v \\
\mathbf{h}_{\tau+1}^{(n)} &= \text{transformer-layer}(\mathbf{Q}_{\tau+1}^{(n)}, \mathbf{K}_{\tau+1}^{(n)}, \mathbf{V}_{\tau+1}^{(n)})
\end{aligned}
$$

And his positional encoding

$$
\begin{aligned}
a_{ij} &= \mathbf{q}_i \mathbf{k}_j^\top = (\mathbf{x}_i + \mathbf{p}_i)\mathbf{W}^q((\mathbf{x}_j + \mathbf{p}_j)\mathbf{W}^k)^\top \\
&= \mathbf{x}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{x}_j^\top + \mathbf{x}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{p}_j^\top + \mathbf{p}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{x}_j^\top + \mathbf{p}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{p}_j^\top
\end{aligned}
$$

$$
a_{ij}^{\text{rel}} = \underbrace{\mathbf{x}_i \mathbf{W}^q {\mathbf{W}_E^k}^\top \mathbf{x}_j^\top}_{\text{content-based addressing}} + \underbrace{\mathbf{x}_i \mathbf{W}^q {\mathbf{W}_R^k}^\top \mathbf{r}_{i-j}^\top}_{\text{content-dependent positional bias}} + \underbrace{\mathbf{u} {\mathbf{W}_E^k}^\top \mathbf{x}_j^\top}_{\text{global content bias}} + \underbrace{\mathbf{v} {\mathbf{W}_R^k}^\top \mathbf{r}_{i-j}^\top}_{\text{global positional bias}}
$$

- Replace $\mathbf{p}_j$ with relative positional encoding $\mathbf{r}_{i-j} \in \mathbf{R}^d$;
- Replace $\mathbf{p}_i \mathbf{W}^q$ with two trainable parameters $\mathbf{u}$ (for content) and $\mathbf{v}$ (for location) in two different terms;
- Split $\mathbf{W}^k$ into two matrices, $\mathbf{W}_E^k$ for content information and $\mathbf{W}_R^k$ for location information.

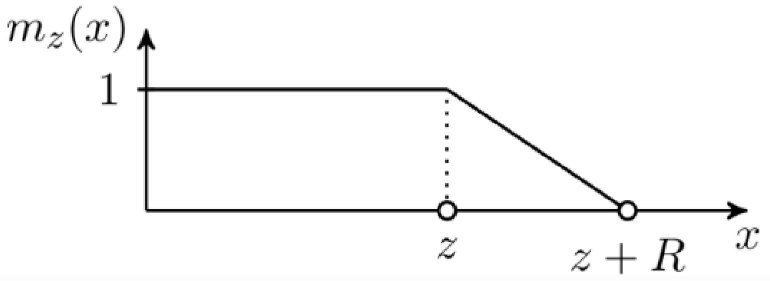
Adaptive Attention Span [Sukhbaatar, et al., (2019)]

$$e_{ij} = \mathbf{q}_i \mathbf{k}_j^{\top}$$

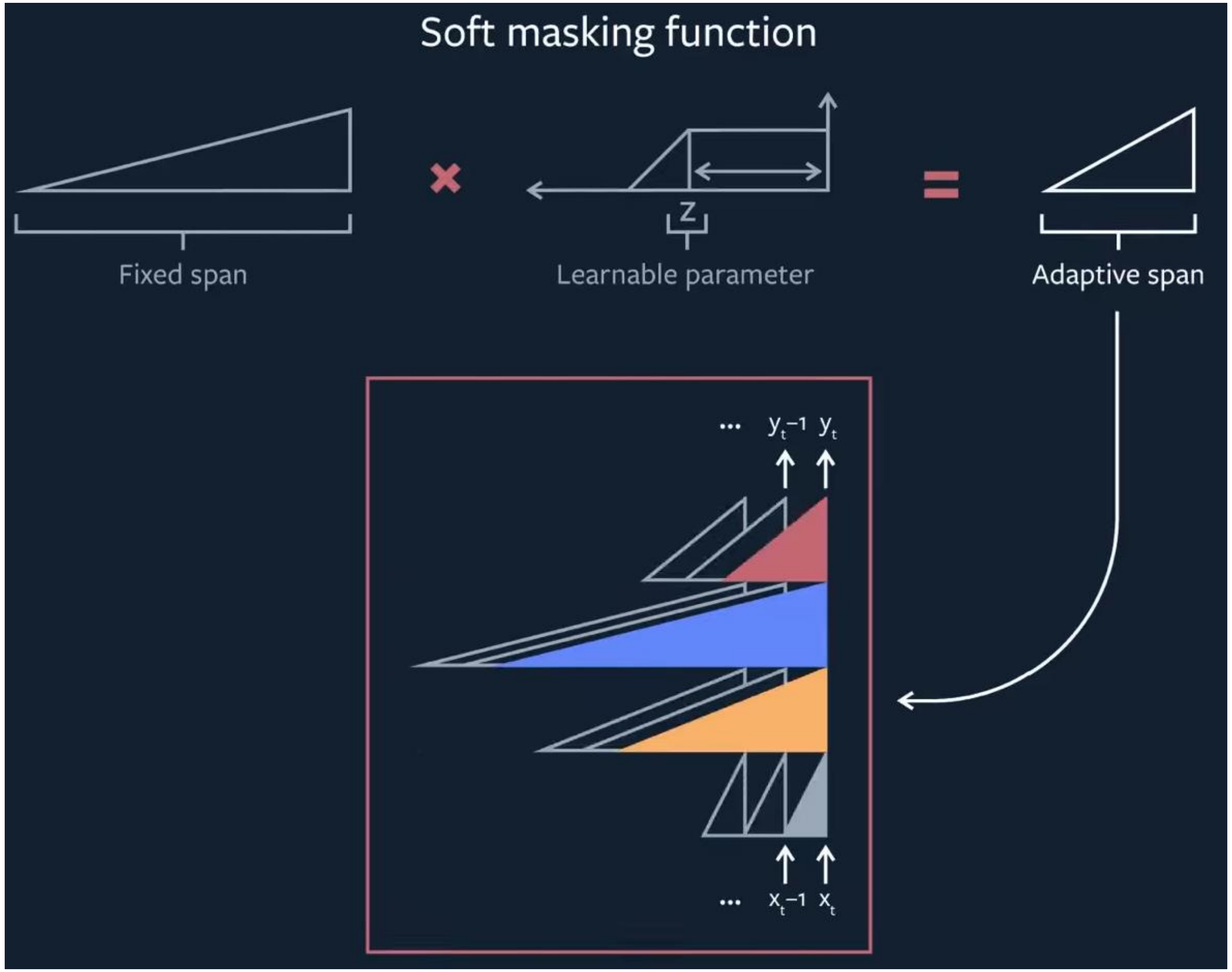
$$a_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{r=i-s}^{i-1} \exp(e_{ir})}$$

$$\mathbf{y}_i = \sum_{r=i-s}^{i-1} a_{ir}\mathbf{v}_r = \sum_{r=i-s}^{i-1} a_{ir}\mathbf{x}_r \mathbf{W}^v$$

$$m_z(x) = \text{clamp}\left(\frac{1}{R}(R + z - x), 0, 1\right)$$



$$a_{ij} = \frac{m_z(i-j)\exp(s_{ij})}{\sum_{r=i-s}^{i-1} m_z(i-r)\exp(s_{ir})}$$



https://ai.facebook.com/blog/making-transformer-networks-simpler-and-more-efficient/

All-attention layer [Sukhbaatar, et al., (2019)]

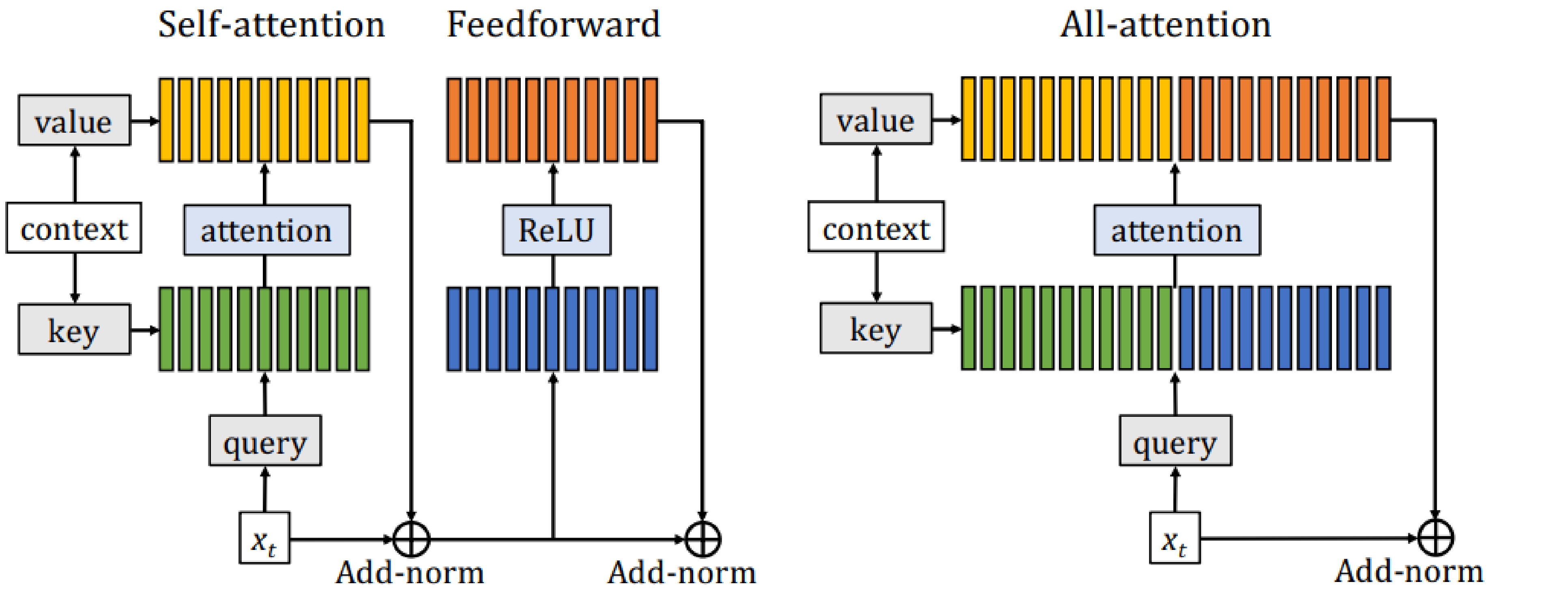


Figure 1: On the left panel, the standard transformer layer is composed of a self-attention sublayer followed by a feedforward sublayer. On the right panel, our all-attention layer merges the weights of the feedforward sublayer with the self-attention sublayer. We represent both models in the case of a single head, but in the general case, both the self-attention sublayer and our all-attention layers have multiple heads.

https://ai.facebook.com/blog/making-transformer-networks-simpler-and-more-efficient/

Sparse Attention Matrix Factorization (Sparse Transformers) Child et al., 2019

aka factorized self-attention with O(sqrt(T)) complexity

$$\mathrm{Attend}(X, S) = \Big(a(\mathbf{x}_i, S_i) \Big)_{i \in \{1,\ldots,n\}}$$

$$a(\mathbf{x}_i, S_i) = \mathrm{softmax}\left(\frac{(W_q \mathbf{x}_i) K_{S_i}^T}{\sqrt{d}} \right) V_{S_i}$$

$$K_{S_i} = \Big(W_k \mathbf{x}_j \Big)_{j \in S_i} \quad V_{S_i} = \Big(W_v \mathbf{x}_j \Big)_{j \in S_i}$$

https://openai.com/blog/sparse-transformer/

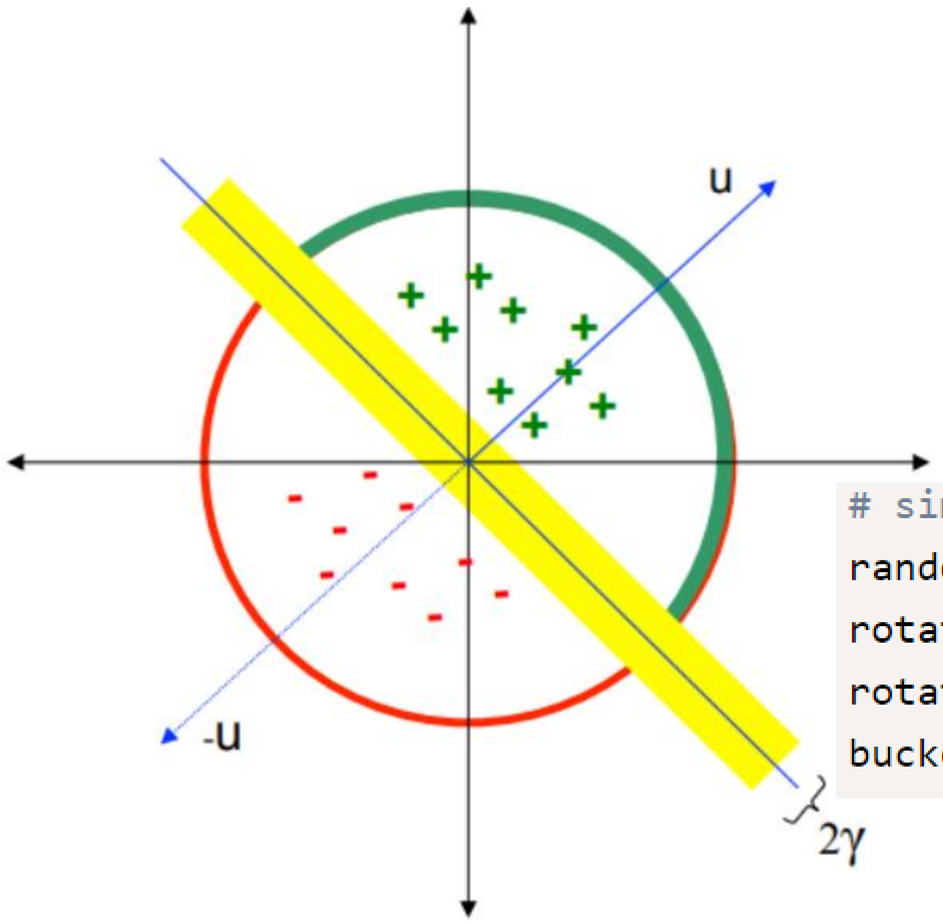
Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020

Вообще другая нахуй другая хуйня!

- Replace the dot-product attention with locality-sensitive hashing (LSH) attention, reducing the complexity from $O(L2^2)$ to $O(L\log L)$.

- Replace the standard residual blocks with reversible residual layers, which allows storing activations only once during training instead of N times (i.e. proportional to the number of layers).

Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020



Local Sensitive Hashing
Alrogythm

```
# simplified to only compute a singular hash
random_rotations = np.random.randn(hidden_dim, n_buckets // 2)
rotated_vectors = np.dot(x, random_rotations)
rotated_vectors = np.hstack([rotated_vectors, -rotated_vectors]
buckets = np.argmax(rotated_vectors, axis=-1)
```

```
lsh_proj = np.random.randn(hidden_size, hash_size)
hash_value = np.sign(np.dot(x, lsh_proj.T))
```

Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020

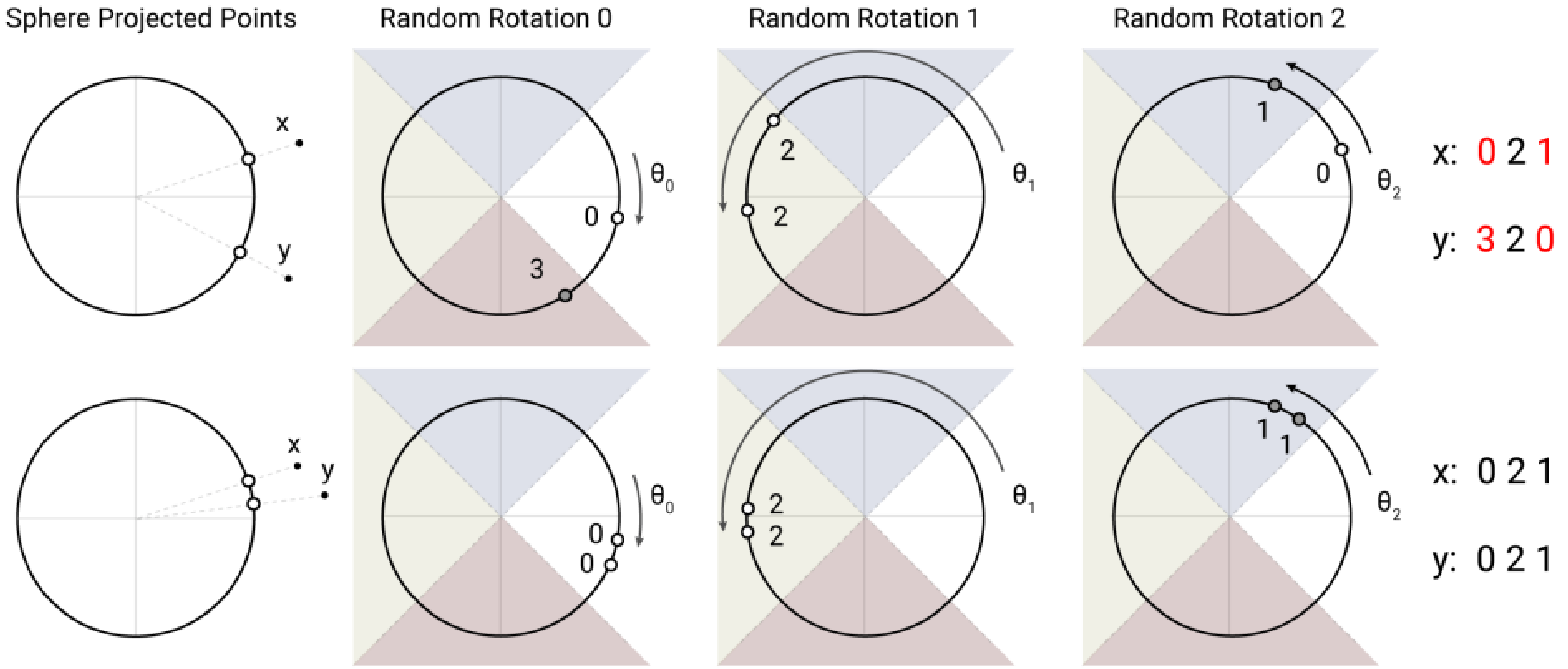


Figure 1: An angular locality sensitive hash uses random rotations of spherically projected points to establish buckets by an argmax over signed axes projections. In this highly simplified 2D depiction, two points x and y are unlikely to share the same hash buckets (above) for the three different angular hashes unless their spherical projections are close to one another (below).

Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020

$x \mapsto h(x)$ locality-sensitive hashing

$\mathbf{R} \in \mathbb{R}^{d \times b/2}$

$$h(x) = \arg\max([xR; -xR]).$$

$$S_i = \{j : h(\mathbf{q}_i) = h(\mathbf{k}_j)\}.$$

$S_i = P_i$

$$\mathcal{P}_i = \bigcup_{r=1}^{n_{rounds}} \mathcal{P}_i^{(r)} \quad \text{where } \mathcal{P}_i^{(r)} = \left\{j : h^{(r)}(q_i) = h^{(r)}(q_j)\right\}$$

Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020

- (a) The attention matrix for full attention is often sparse.

- (b) Using LSH, we can sort the keys and queries to be aligned according to their hash buckets.

- (c) Set **Q=KQ=K** (precisely **kj=qj/‖qj‖kj=qj/‖qj‖**), so that there are equal numbers of keys and queries in one bucket, easier for batching. Interestingly, this “shared-QK” config does not affect the performance of the Transformer.

- (d) Apply batching where chunks of mm consecutive queries are grouped together.

Locality-Sensitive Hashing Attention (Reformer)

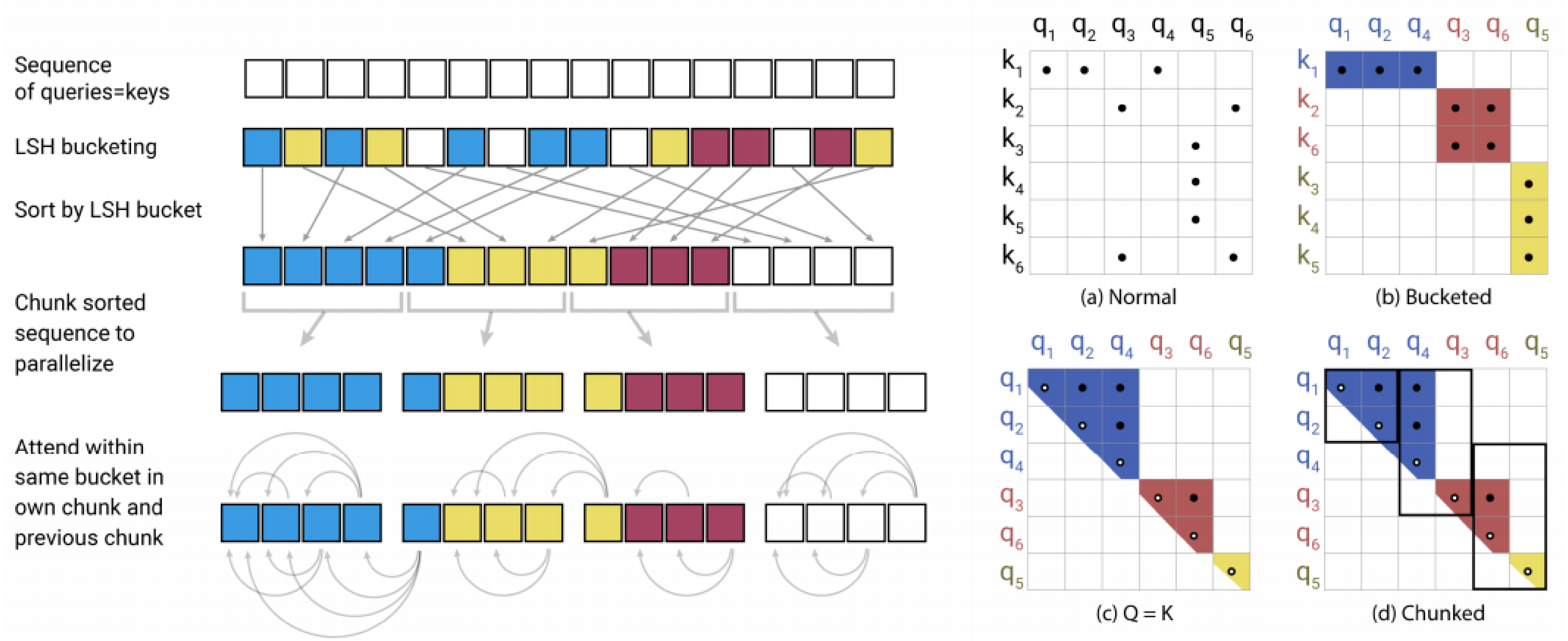


Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.

Locality-Sensitive Hashing Attention (Reformer)
Kitaev, et al. 2020

- Reversible Residual Network

$$y_1 = x_1 + F(x_2), \; y_2 = x_2 + G(y_1)$$

$$x_2 = y_2 - G(y_1), \; x_1 = y_1 - F(x_2)$$

$$Y_1 = X_1 + \mathrm{Attention}(X_2), \; Y_2 = X_2 + \mathrm{FeedForward}(Y_1)$$

$$Y_2 = [Y_2^{(1)}; \ldots; Y_2^{(c)}] = [X_2^{(1)} + \mathrm{FeedForward}(Y_1^{(1)}); \ldots; X_2^{(c)} + \mathrm{FeedForward}(Y_1^{(c)})]$$

To be continued…

Cause there is an attention in detection

Cause there is attention in graphs

Cause there is attention in 3d

Cause there is some theoretical evidence of relation between Attention and CNN and Graphs and Oh boy..

Desecrated links

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html

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

http://jalammar.github.io/illustrated-transformer/

https://towardsdatascience.com/attention-in-neural-networks-e66920838742

https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#ba24