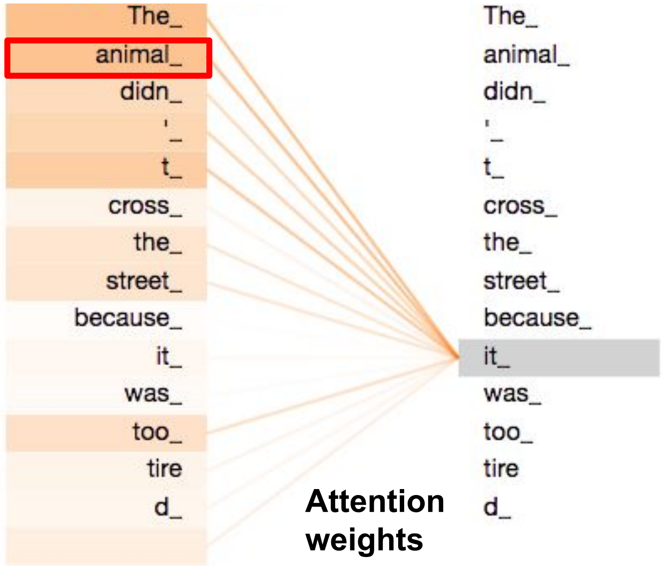
Going Beyond CNNs
Self Attention & Transformers

Self Attention

"The animal didn't cross the street because **it** was too tired"

- Transformers are Widely used in NLP: "Attention is All you Need [1]" .

- **Example:** Machine Translation Task
- Understanding **"it"** in the context of the sentence.



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

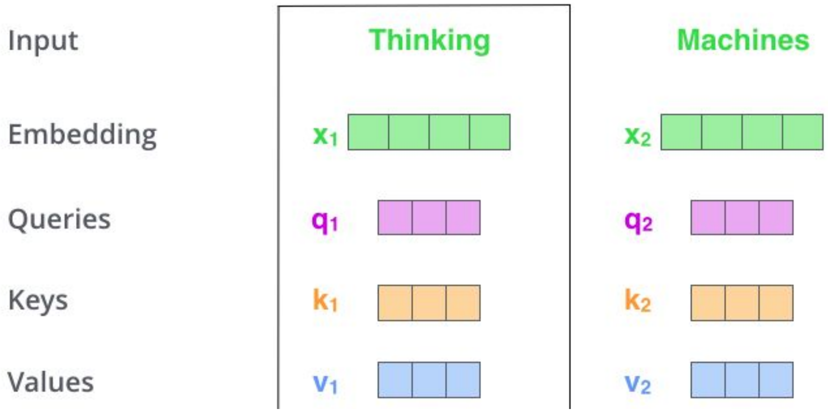
[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Self Attention

- Transformers are Widely used in NLP: “Attention is All you Need [1]” .

- **Dictionary Lookup**
- **Relate, Aggregate** operations.

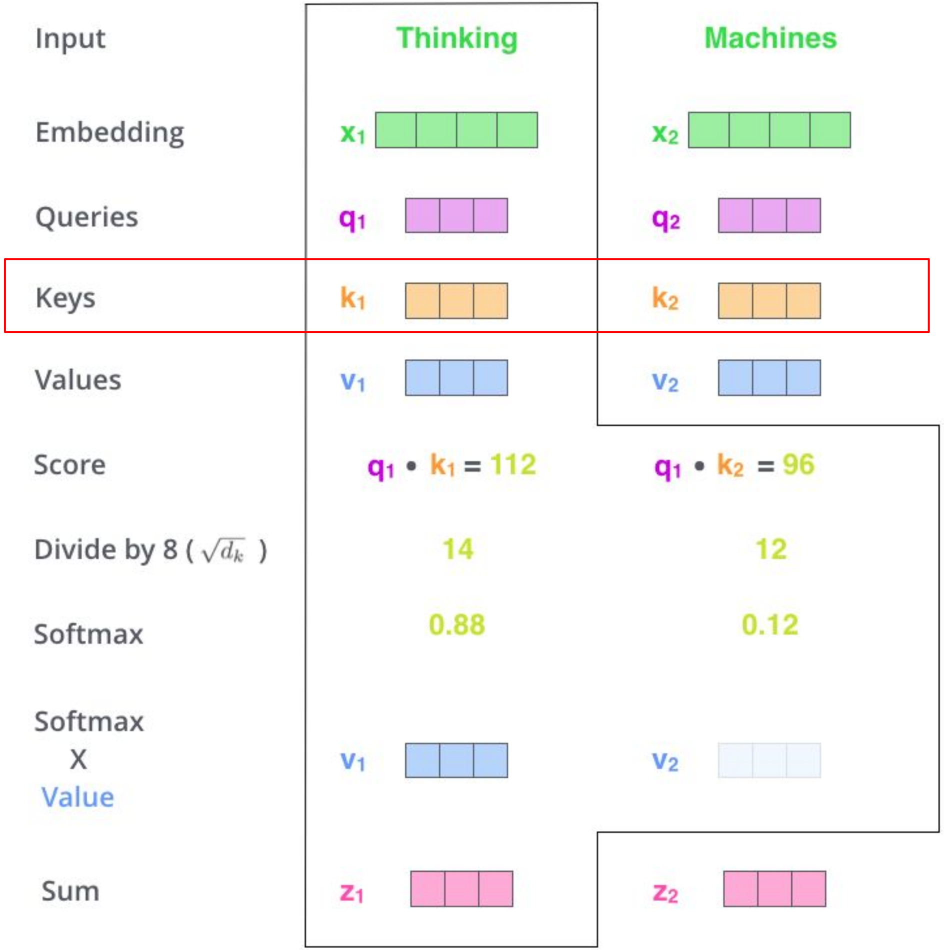
“Thinking Machines”



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

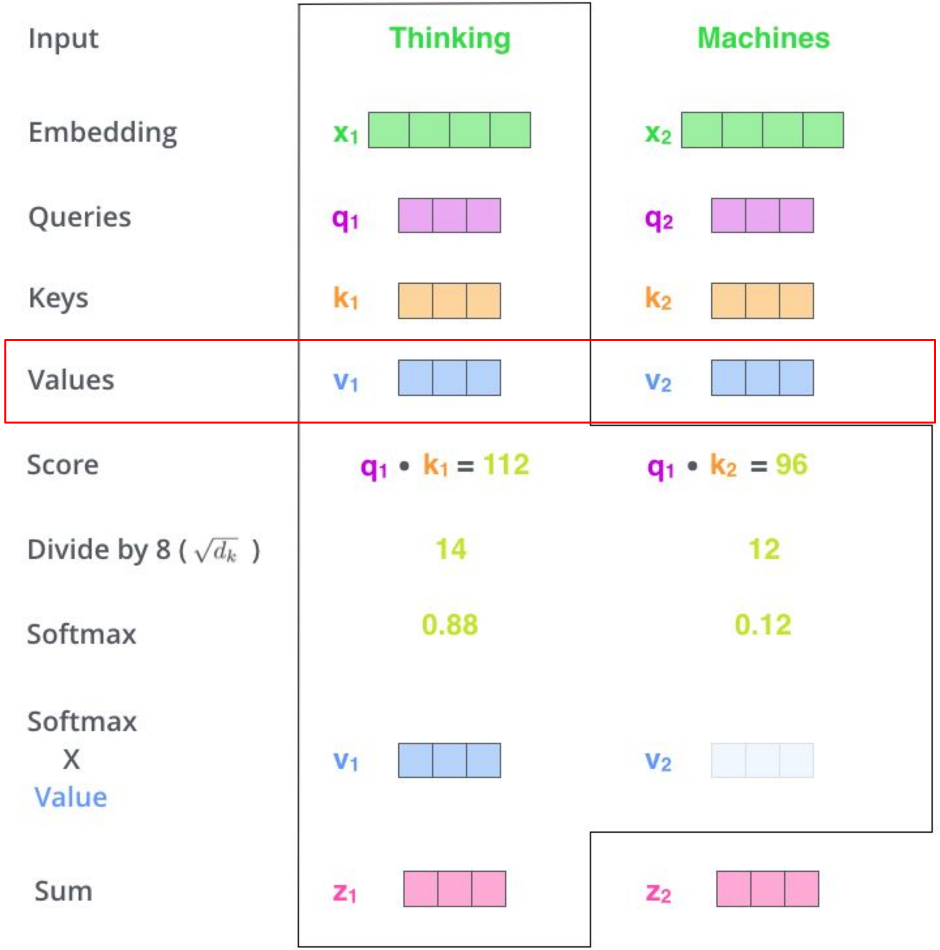
[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Think of this as your Dictionary keys used for addressing



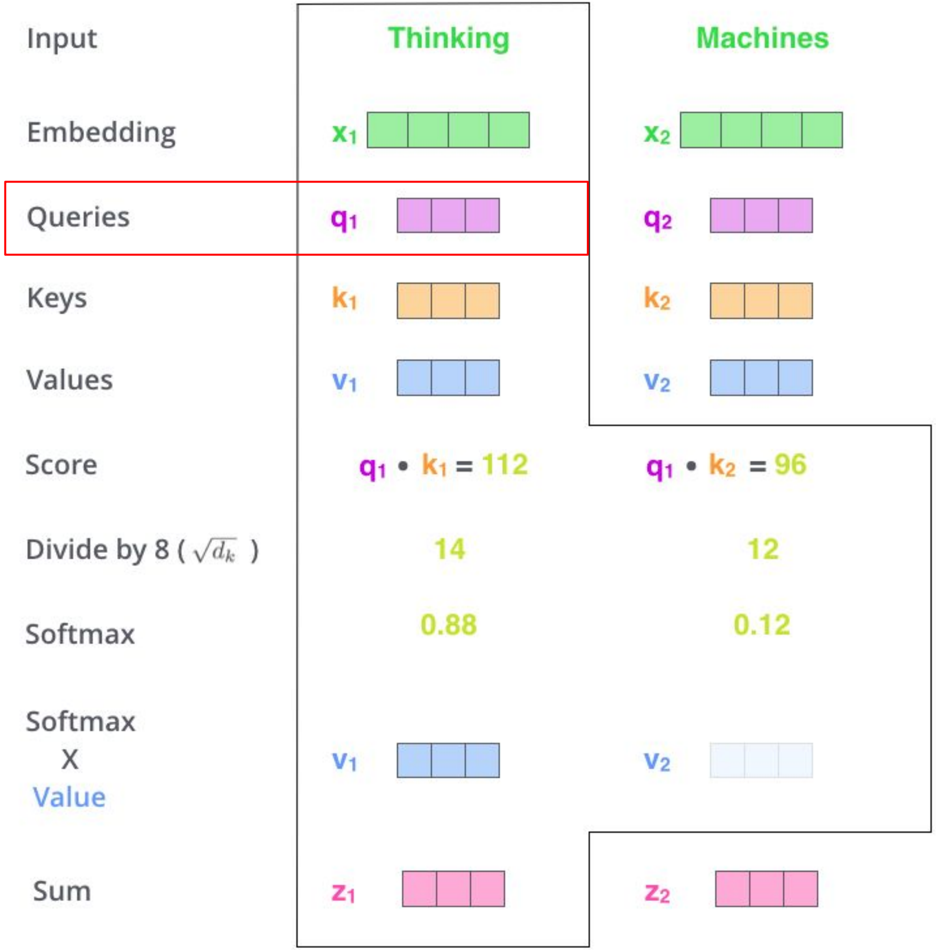
http://jalammar.github.io/illustrated-transformer/

Detailed information - what I want to read out from memory



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

My current word embedding

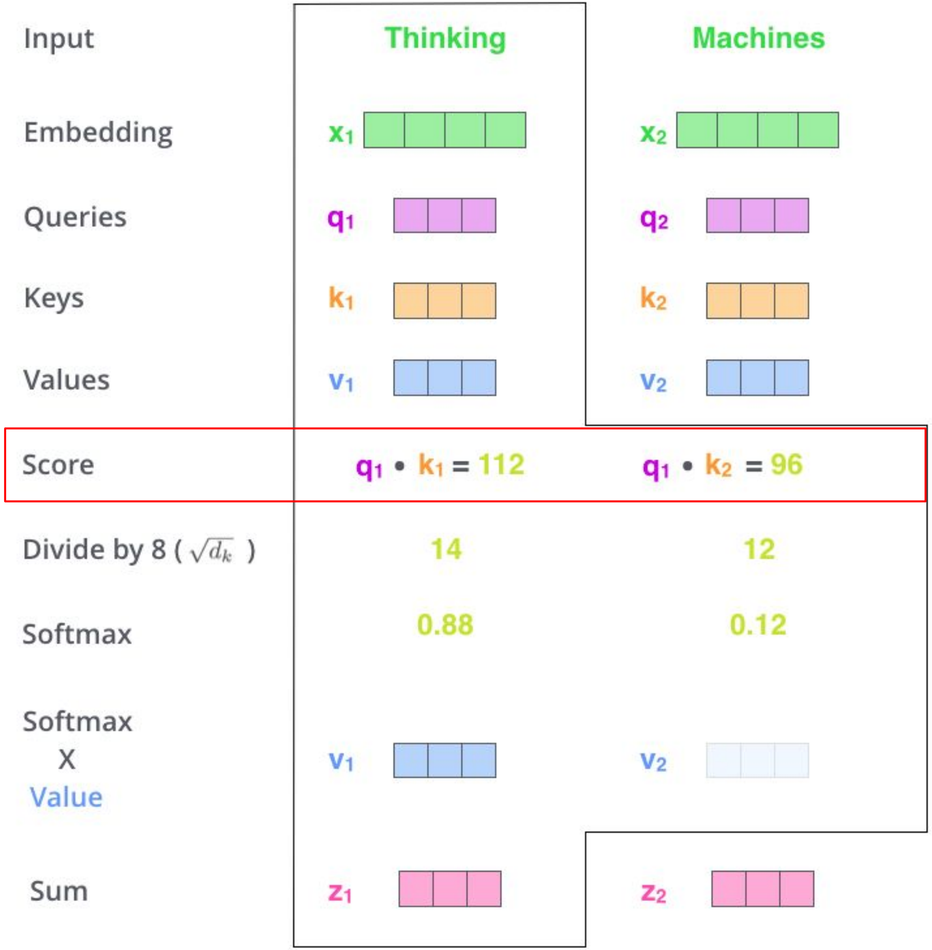


http://jalammar.github.io/illustrated-transformer/

1] Relate

Compatibility bet. each embedding in the dictionary to myself

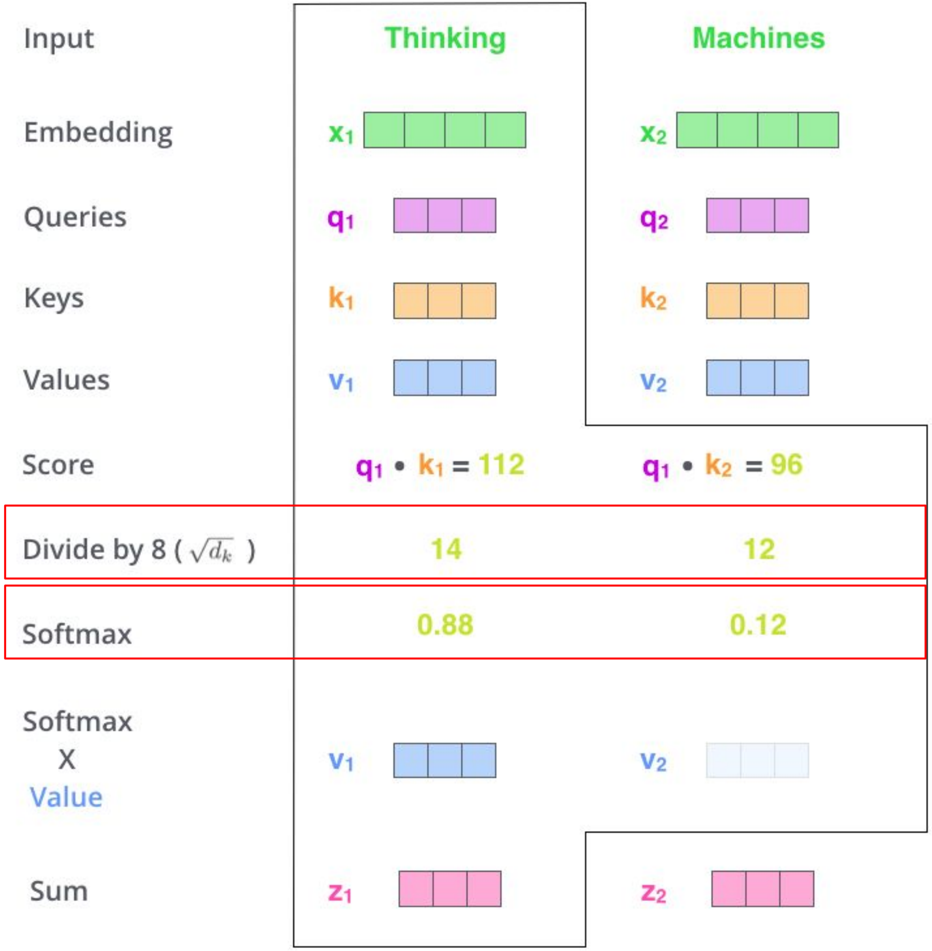
Scalar, Pairwise



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

1] Relate

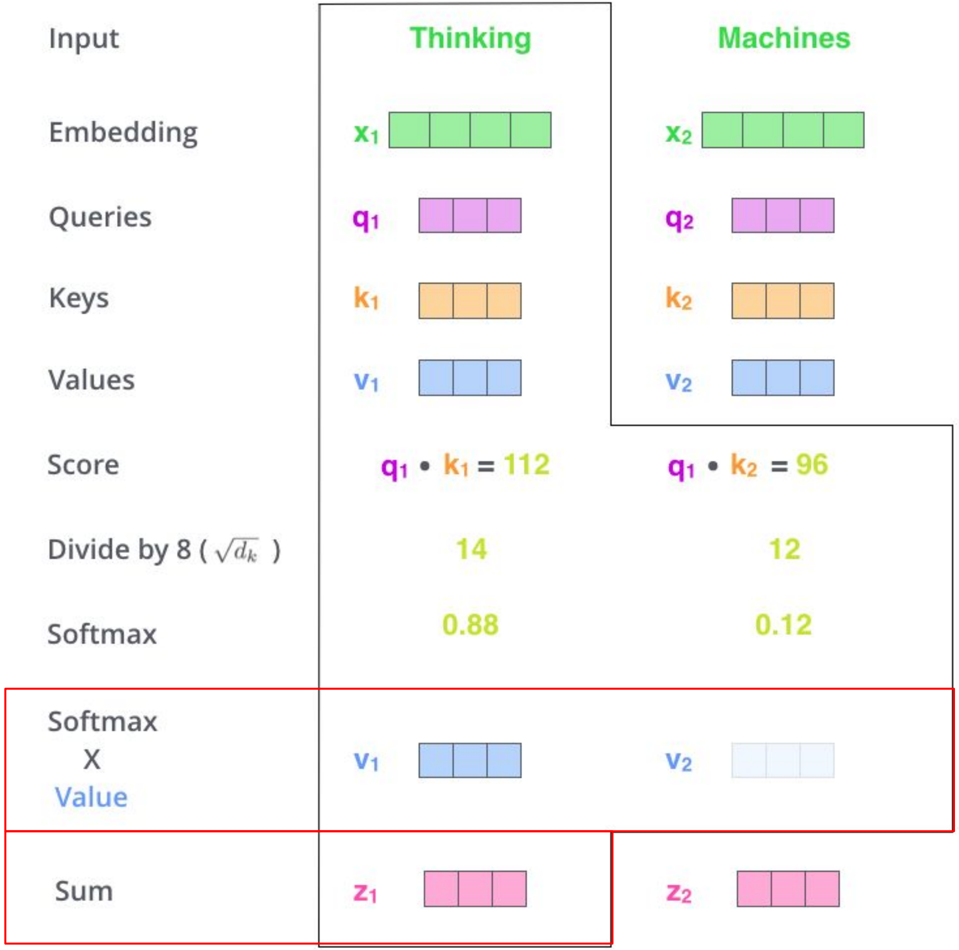
Scaled: because for large values of d_k
→large values of dot product
→ pushes the softmax to have small gradients.



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

2] Aggregate

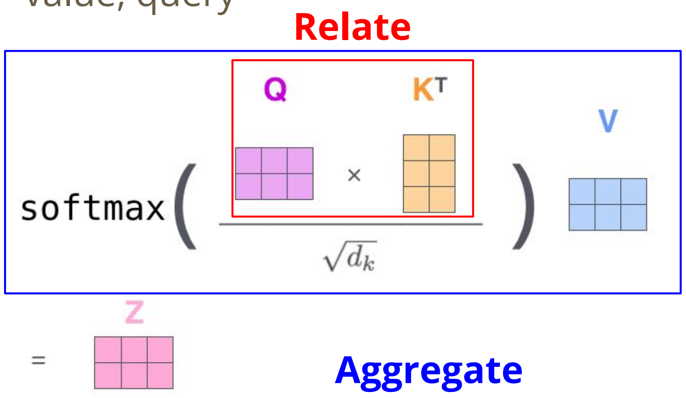
Aggregate information from all tokens

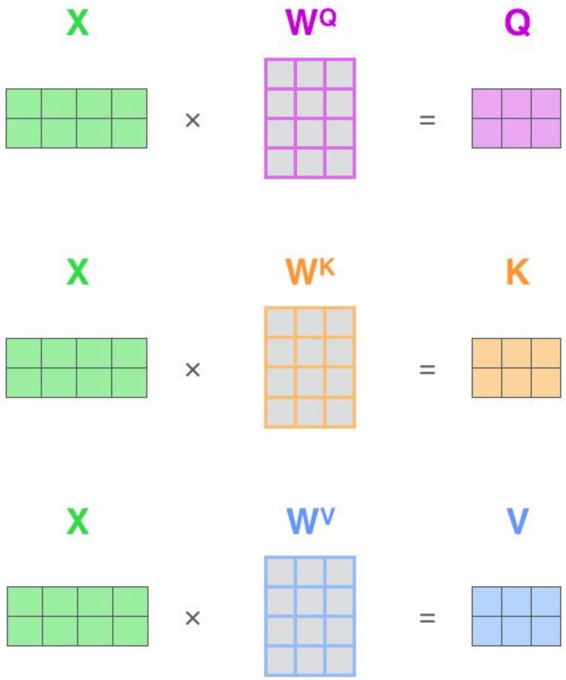


http://jalammar.github.io/illustrated-transformer/

Single-Headed Attention

- Single Headed Attention:
  - Learn weights to obtain key, value, query



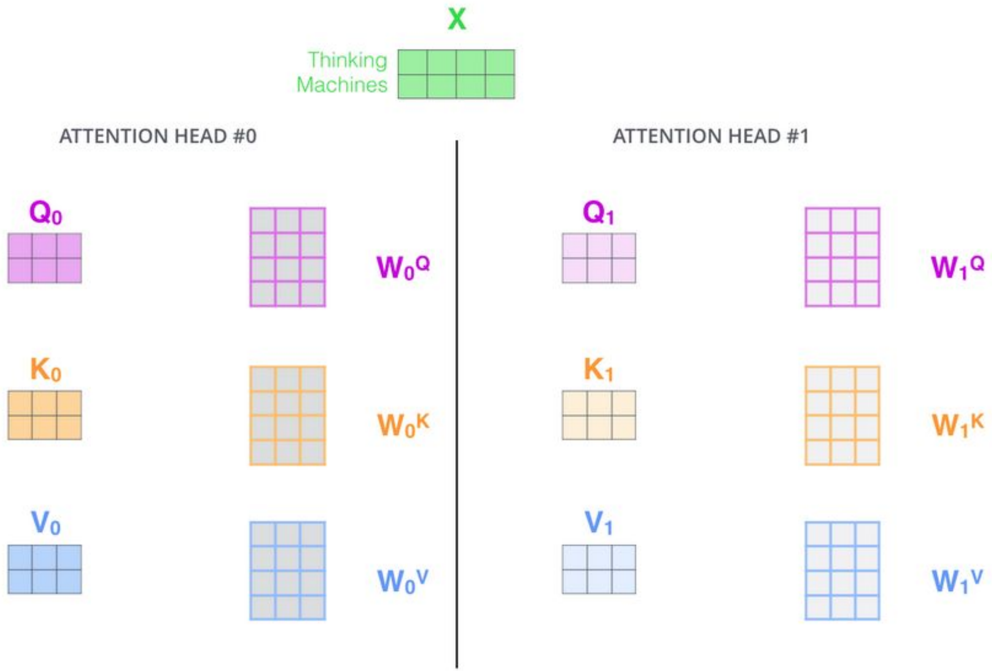


http://jalammar.github.io/illustrated-transformer/

[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Multi-Headed Attention

- Single Headed Attention:
  - Learn weights to obtain key, value, query
- Multi Headed Attention:
  - Multiple sets of weight matrices.
  - Multiple representation subspaces.

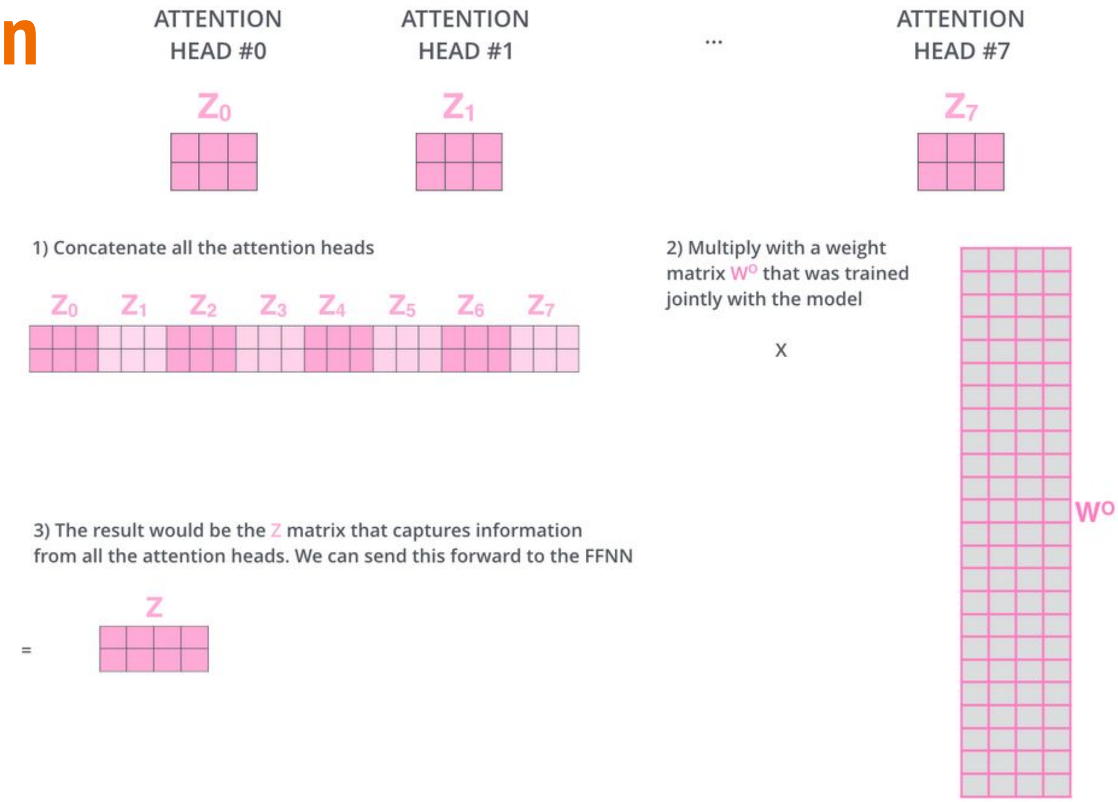


http://jalammar.github.io/illustrated-transformer/

[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Multi-Headed Attention

- Single Headed Attention:
  - Learn weights to obtain key, value, query
- Multi Headed Attention:
  - Multiple sets of weight matrices

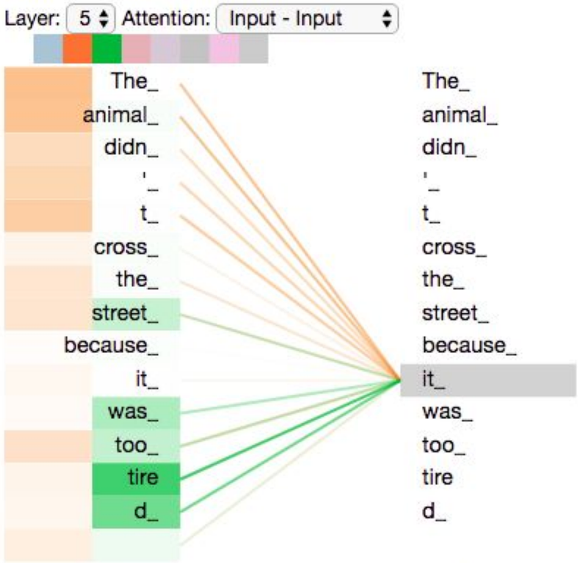


http://jalammar.github.io/illustrated-transformer/

[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

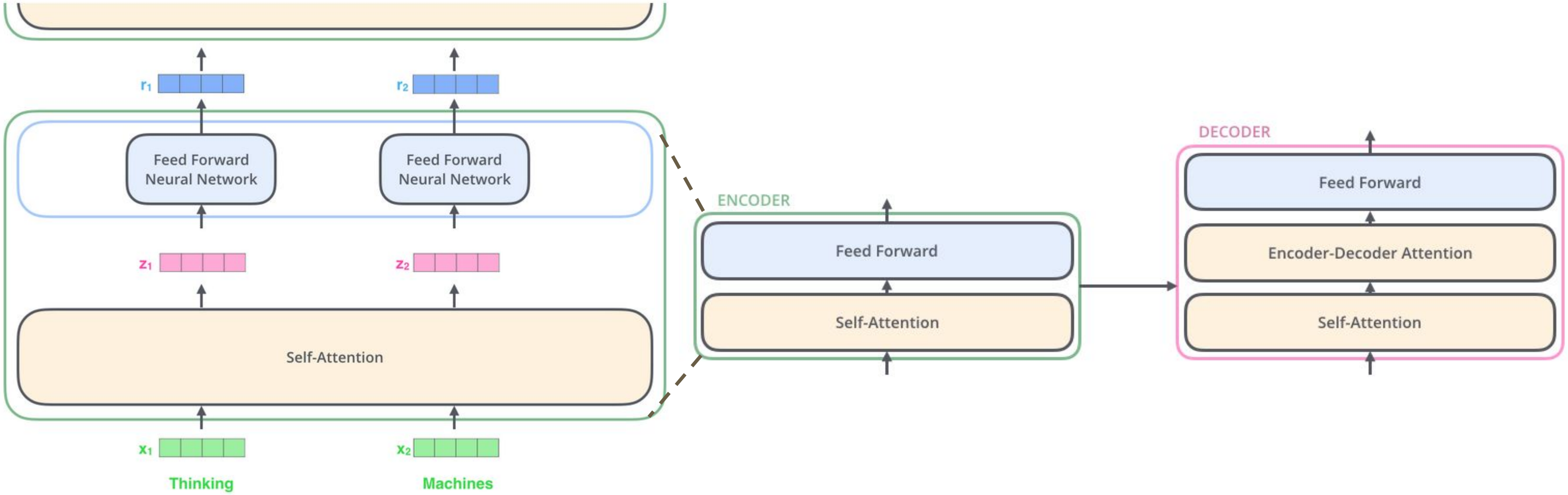
Multi-Headed Attention

- Multiple representation subspaces.



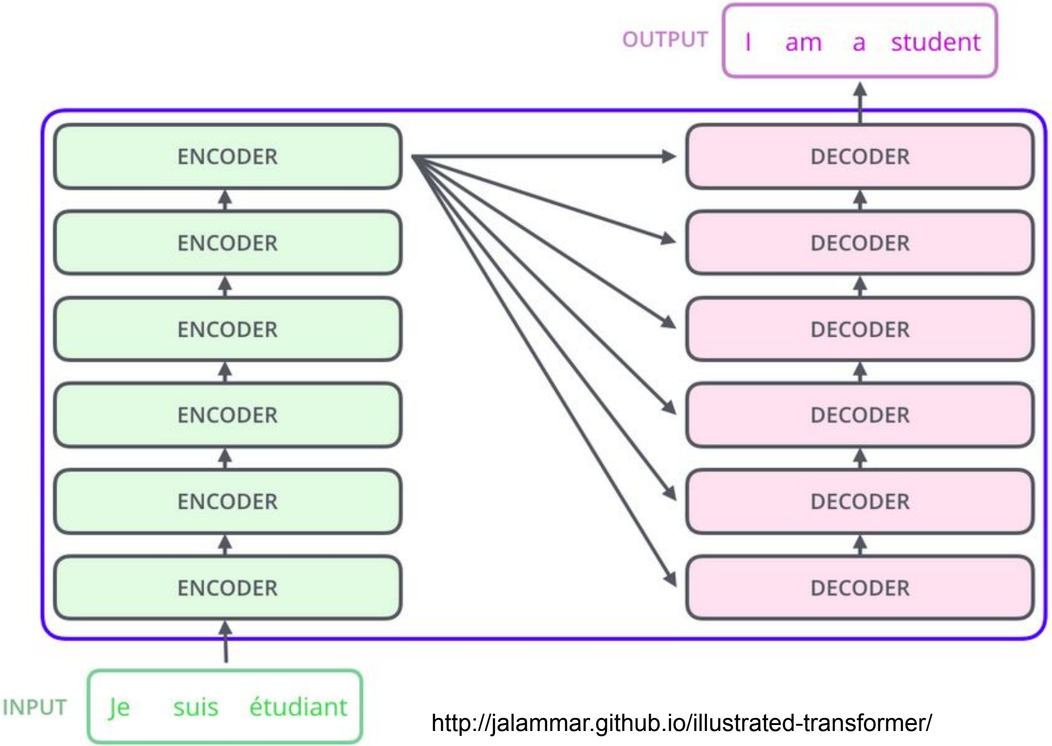
As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Transformers - Encoder



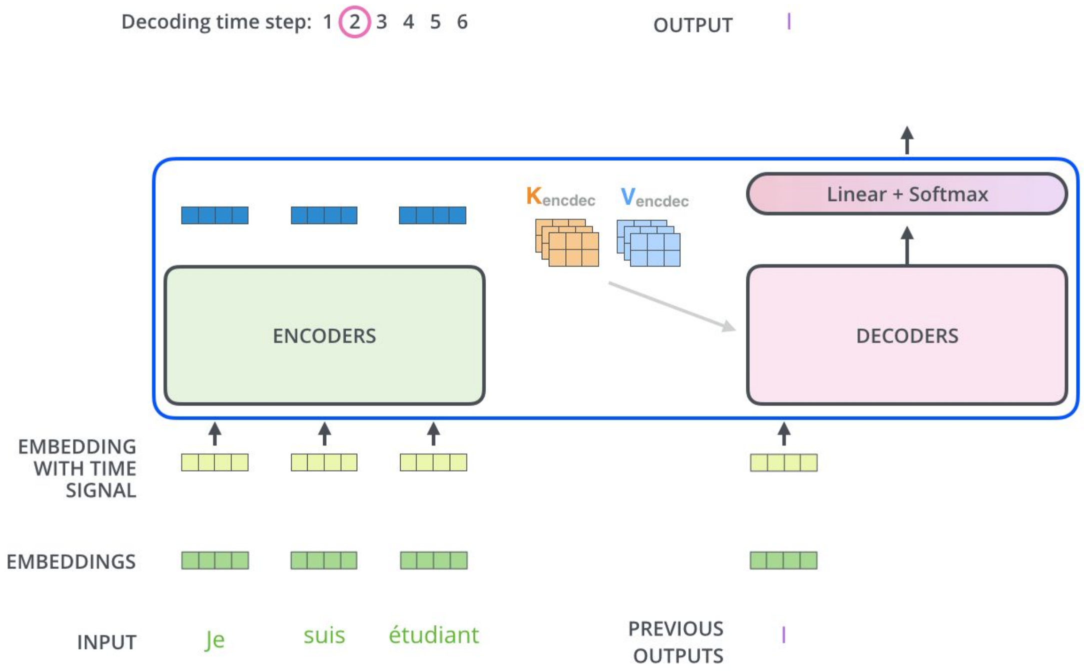
http://jalammar.github.io/illustrated-transformer/

Transformers - Full Model



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

Transformers - Full Model



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

Positional Encoding

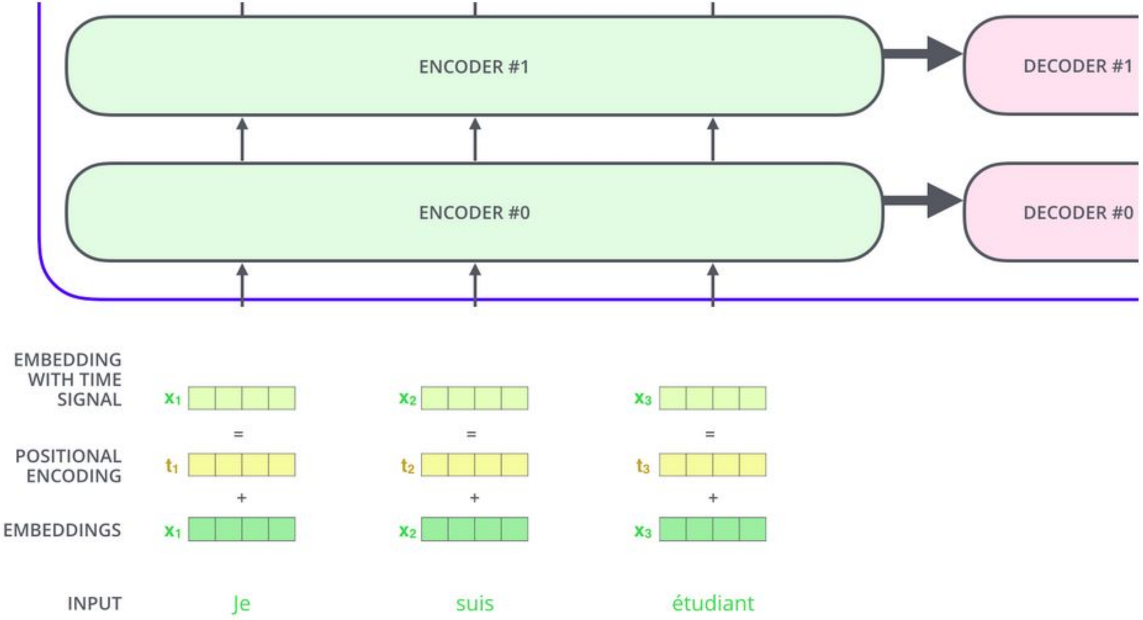
- Helps to determine the position of each word, to encode order of the sequence.

Index dim. In embedding

in/out dim.

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

Word position



Positional Encoding

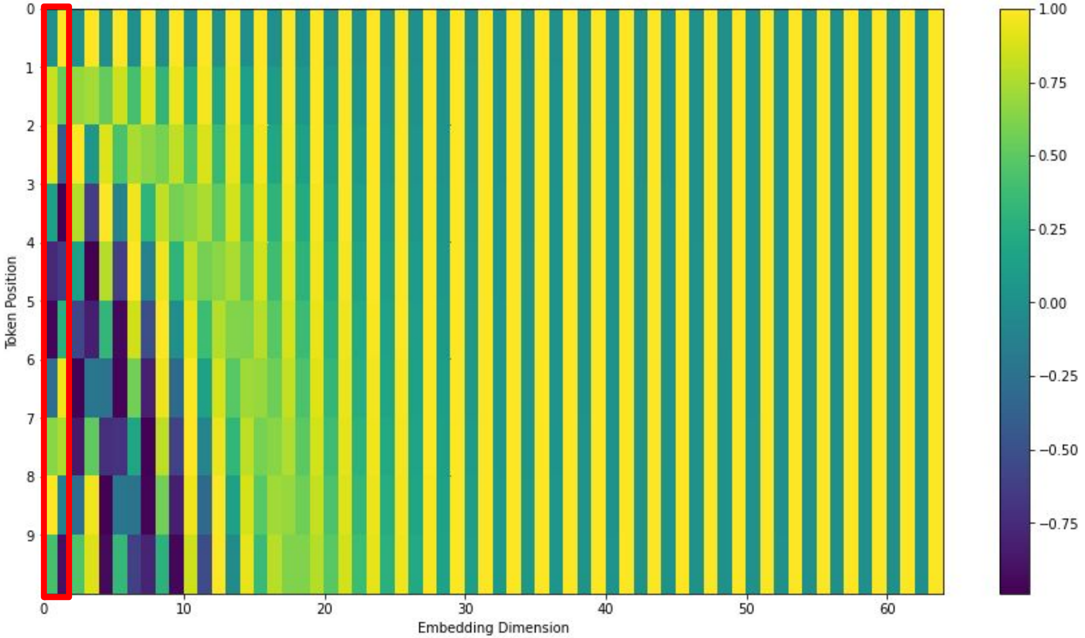
- Helps to determine the position of each word, to encode order of the sequence

Index dim. In embedding

in/out dim.

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

Word position

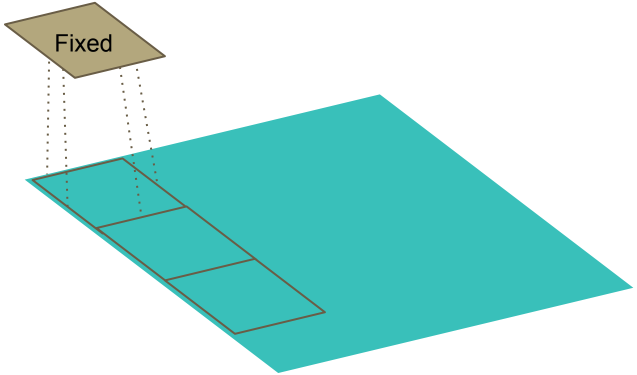


What about in Computer Vision Tasks?

- Widely used Convolutions: is there something better?
- Two main Operations
  - **Feature aggregation**: happens in the convolution
  - Feature transformation: successive linear and non-linear mappings

Fixed weights through all positions

- Can weights be **Adaptive to input**?

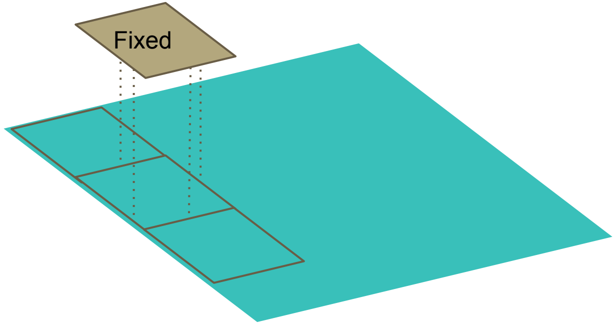


https://www.youtube.com/watch?v=GuAsn4E3fP8

What about in Computer Vision Tasks?

- Widely used Convolutions: is there something better?
- Two main Operations
  - **Feature aggregation**: happens in the convolution
  - Feature transformation: successive linear and non-linear mappings

Fixed weights through all positions



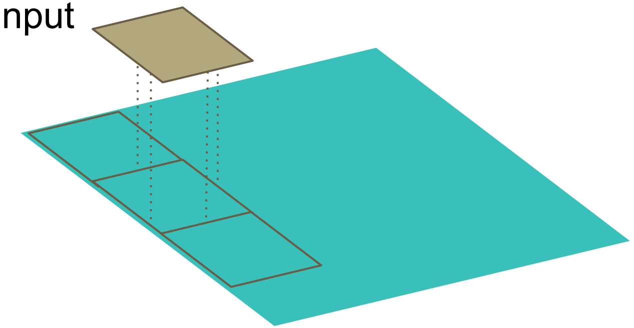
- Can weights be **Adaptive to input**?

https://www.youtube.com/watch?v=GuAsn4E3fP8

What about in Computer Vision Tasks?

- **Feature aggregation**: replace convolution with self attention.

- Can weights be **Adaptive to input**?
    - Pairwise Attention
    - Patchwise Attention



https://www.youtube.com/watch?v=GuAsn4E3fP8

Pairwise Attention

- R(i): local footprint of aggregation, set of indices.
- $\beta(\mathbf{x}_j)$, : Values, what to read from.
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weights.

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \odot \beta(\mathbf{x}_j),$$

2] Aggregate

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

1] Relate

Nonlinear transformations

Compatibility Function

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

1] Relate

Summation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) + \psi(\mathbf{x}_j)$

Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Concatenation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = [\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)]$

Hadamard product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \odot \psi(\mathbf{x}_j)$

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$

Compatibility Functions

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

1] Relate

Summation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) + \psi(\mathbf{x}_j)$

Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Concatenation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = [\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)]$

Hadamard product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \odot \psi(\mathbf{x}_j)$

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$

Scalar Attention similar to what Transformers used

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

1] Relate

Summation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) + \psi(\mathbf{x}_j)$

Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Concatenation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = [\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)]$

Hadamard product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \odot \psi(\mathbf{x}_j)$

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$

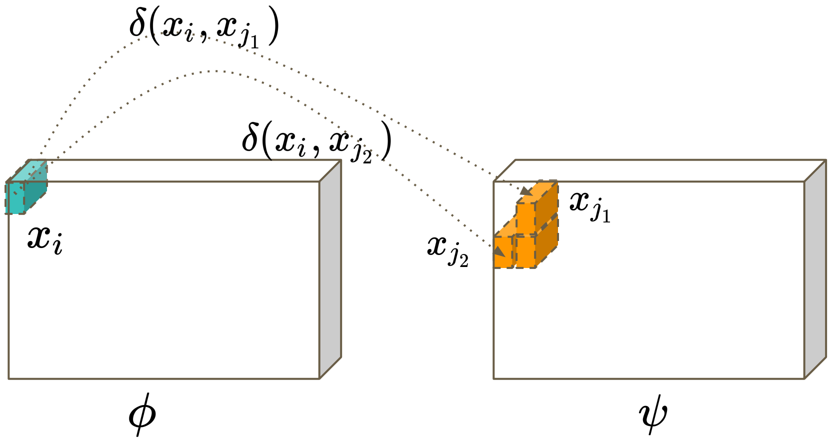
Vector Attention, not all feature channels will be treated the same

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weights

1] Relate

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$



Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

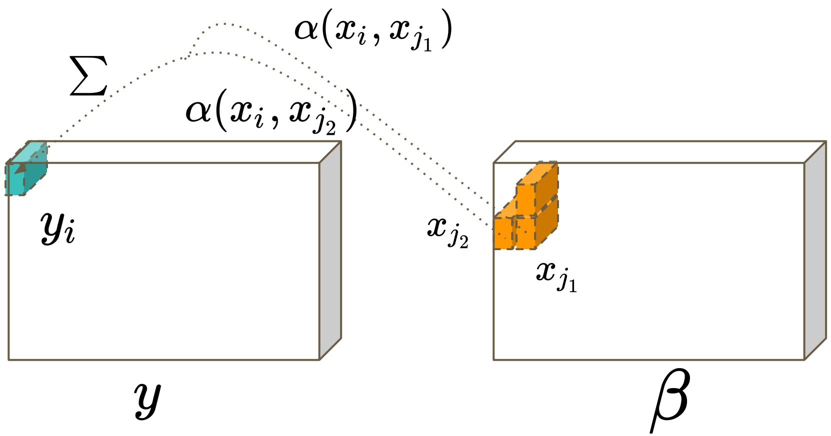
γ={Linear→ReLU→Linear}

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weights

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \odot \beta(\mathbf{x}_j),$$

2] Aggregate



Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \odot \boxed{\beta}(\mathbf{x}_j),$$

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$

β

Values-Fn

Summation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) + \psi(\mathbf{x}_j)$

Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Concatenation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = [\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)]$

Hadamard product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \odot \psi(\mathbf{x}_j)$

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \boxed{\varphi(\mathbf{x}_i)}^\top \boxed{\psi(\mathbf{x}_j)}$

ϕ ψ

Queries-Fn Keys-Fn

Pairwise Attention

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \odot \beta(\mathbf{x}_j),$$

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j))$$

Operation	Content adaptive	Channel adaptive
Convolution [19]	✗	✓
Scalar attention [33, 35, 27, 13]	✓	✗
Vector attention (ours)	✓	✓

Summation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) + \psi(\mathbf{x}_j)$

Subtraction: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) - \psi(\mathbf{x}_j)$

Concatenation: $\delta(\mathbf{x}_i, \mathbf{x}_j) = [\varphi(\mathbf{x}_i), \psi(\mathbf{x}_j)]$

Hadamard product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \odot \psi(\mathbf{x}_j)$

Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$

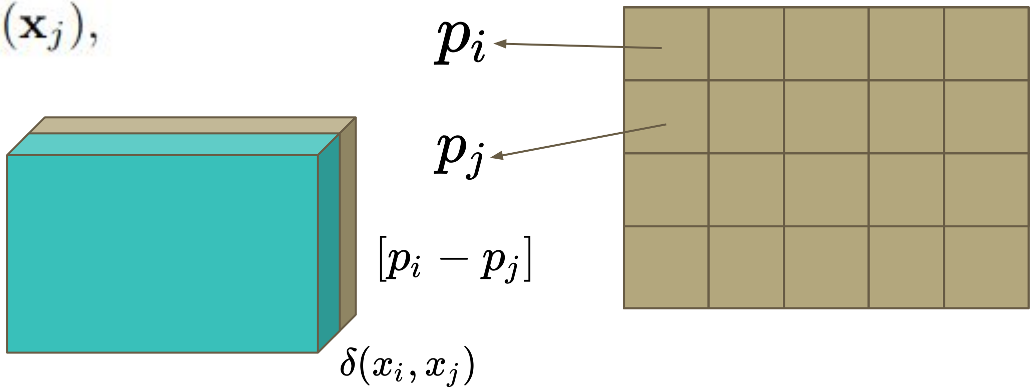
Positional Encoding

- R(i): local footprint of aggregation
- $\beta(\mathbf{x}_j)$, : Values
- $\alpha(\mathbf{x}_i, \mathbf{x}_j)$ adaptive weight vectors

Augment feature maps with positions normalized [-1, 1] that goes through a trainable linear layer

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_i, \mathbf{x}_j) \odot \beta(\mathbf{x}_j),$$

$$\alpha(\mathbf{x}_i, \mathbf{x}_j) = \gamma(\delta(\mathbf{x}_i, \mathbf{x}_j)).$$



Relative positions

Patchwise Attention

2] Aggregate

Adaptive Weights → similar to convolutional Kernel

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \boxed{\alpha(\mathbf{x}_{\mathcal{R}(i)})}_j \odot \beta(\mathbf{x}_j),$$

Patch for the local footprint $R(i)$

Patchwise Attention

2] Aggregate

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_{\mathcal{R}(i)})_j \odot \beta(\mathbf{x}_j),$$

1] Relate

$$\alpha(\mathbf{x}_{\mathcal{R}(i)}) = \gamma(\delta(\mathbf{x}_{\mathcal{R}(i)}))$$

Star-product: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)]_{\forall j \in \mathcal{R}(i)}$

Clique-product: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_j)^\top \psi(\mathbf{x}_k)]_{\forall j,k \in \mathcal{R}(i)}$

Concatenation: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_i), [\psi(\mathbf{x}_j)]_{\forall j \in \mathcal{R}(i)}]$

Compatibility Fns

Patchwise Attention

2] Aggregate

$$\mathbf{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathbf{x}_{\mathcal{R}(i)})_j \odot \beta(\mathbf{x}_j),$$

$$\alpha(\mathbf{x}_{\mathcal{R}(i)}) = \gamma(\delta(\mathbf{x}_{\mathcal{R}(i)}))$$

1] Relate

Star-product: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)]_{\forall j \in \mathcal{R}(i)}$

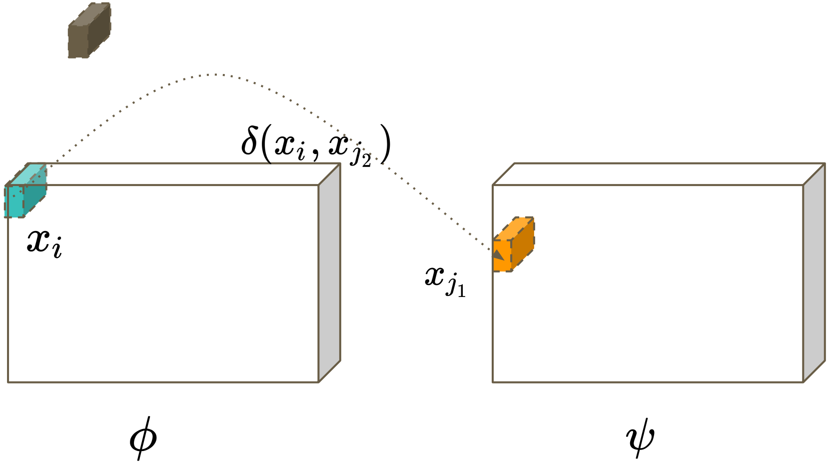
Clique-product: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_j)^\top \psi(\mathbf{x}_k)]_{\forall j,k \in \mathcal{R}(i)}$

Concatenation: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_i), [\psi(\mathbf{x}_j)]_{\forall j \in \mathcal{R}(i)}]$

β	ϕ	ψ
Values	Query	Key

Pairwise vs Patchwise Relate

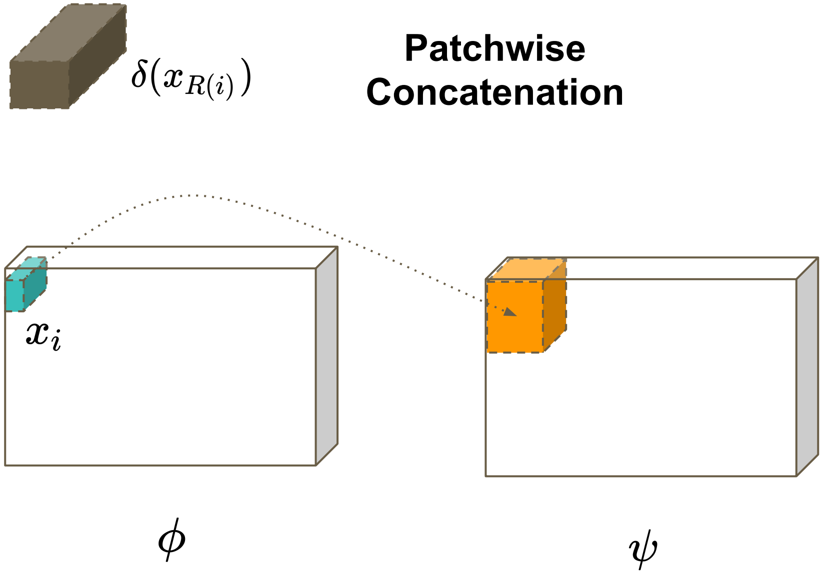
Pairwise



Dot product: $\delta(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$

Single scalar/vector (if not dot product)

Patchwise Concatenation



Concatenation: $\delta(\mathbf{x}_{\mathcal{R}(i)}) = [\varphi(\mathbf{x}_i), [\psi(\mathbf{x}_j)]_{\forall j \in \mathcal{R}(i)}]$

Output Tensor based on local footprint

Patchwise Attention

K: patch size

X $\quad$ X

$C \times W \times H$ $\quad \boxed{\phi}$ $\quad$ Fn generate **Query** $\quad$ Fn generate **Keys** $\quad \boxed{\psi}$ $\quad C \times W \times H$

$$\boxed{\delta(x_{R(i)}) = [\phi(x_i), [\psi(x_j)]_{\forall j \in R(i)}]}$$

$C \times 1 \times WH \qquad C.K^2 \times 1 \times WH$

Local footprint size similar to kernel size in convolution

Patchwise Attention

K: patch size

X $\qquad$ X

$C \times W \times H$ $\boxed{\phi}$ Fn generate **Query**

Fn generate **Keys** $\boxed{\psi}$ $C \times W \times H$

$$\boxed{\delta(x_{R(i)}) = [\phi(x_i), [\psi(x_j)]_{\forall j \in R(i)}]}$$

$C \times 1 \times WH$ $\qquad$ $C.K^2 \times 1 \times WH$

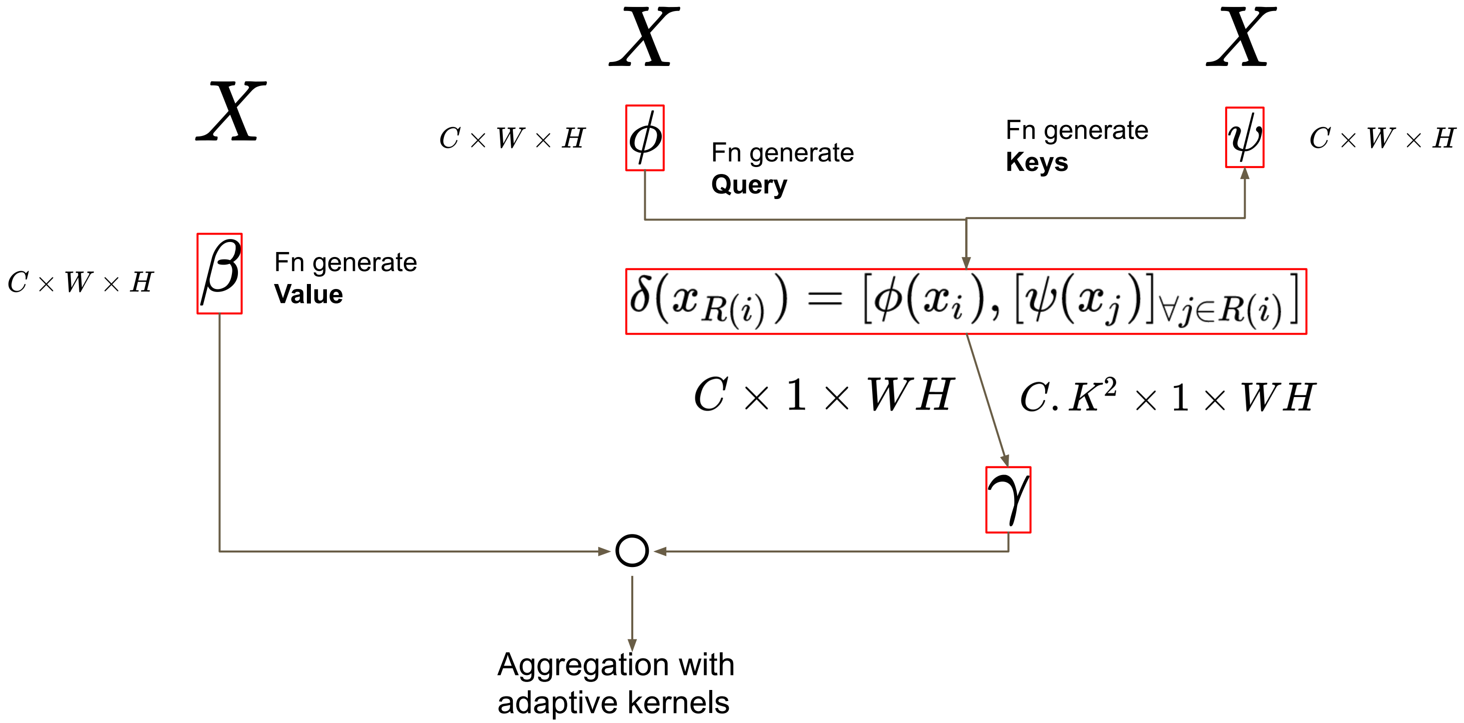
$\boxed{\gamma}$ $\qquad$ γ={Linear→ReLU→Linear}

α : $C \times K^2 \times WH$

For every pixel corresponding adaptive weights

α $\quad C \times K \times K$

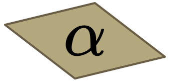
Patchwise Attention



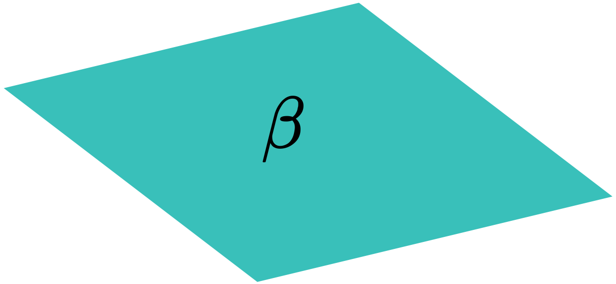
Patchwise Attention

K: patch size

For every pixel
corresponding adaptive
weights



$C \times K \times K$



X $\qquad$ X

$C \times W \times H$ $\boxed{\phi}$ Fn generate **Query**

Fn generate **Keys** $\boxed{\psi}$ $C \times W \times H$

$$\boxed{\delta(x_{R(i)}) = [\phi(x_i), [\psi(x_j)]_{\forall j \in R(i)}]}$$

$C \times 1 \times WH$ $\qquad$ $C.K^2 \times 1 \times WH$

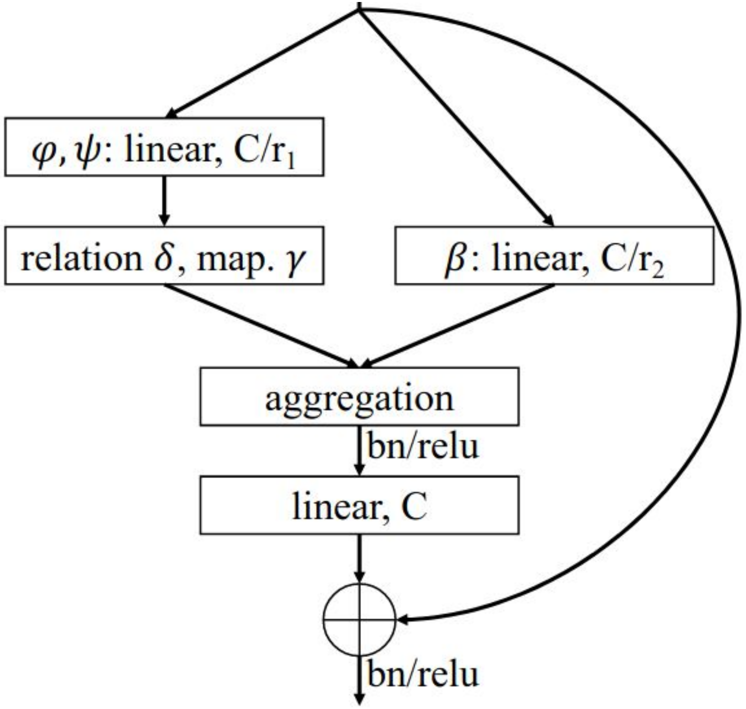
$\boxed{\gamma}$

$C \times K^2 \times WH$

SAN Module

- Performing feature aggregation and transformation in self attention blocks with residual connections.

- SANX: [10, 15, 19] , X→ Number of self attention block.



Experiments

They use **subtraction** for **pairwise**.
Concatenation for **patchwise**

Method	ResNet26 vs. SAN10				ResNet38 vs. SAN15				ResNet50 vs. SAN19			
	top-1	top-5	Params	Flops	top-1	top-5	Params	Flops	top-1	top-5	Params	Flops
Convolutional	73.6	91.7	13.7M	2.4G	76.0	93.0	19.6M	3.2G	76.9	93.5	25.6M	4.1G
SAN, pairwise	74.9	92.1	10.5M	2.2G	76.6	93.1	14.1M	3.0G	76.9	93.4	17.6M	3.8G
SAN, patchwise	77.1	93.5	11.8M	1.9G	78.0	93.9	16.2M	2.6G	78.2	93.9	20.5M	3.3G

Table 3. Comparison of self-attention networks and convolutional residual networks on ImageNet classification. Single-crop testing on the val-original set.

Adaptive kernels are better than static kernels

Relation Fn Ablation

Vector attention is better than scalar attention

Method		top-1	top-5	Params	Flops
Conv.-ResNet26		76.0	92.8	13.7M	2.4G
SAN10-pair.	summation	77.4	93.3	10.5M	2.2G
	subtraction	77.4	93.3	10.5M	2.2G
	concatenate	76.4	92.6	10.6M	2.5G
	Had. product	77.4	93.4	10.5M	2.2G
	dot product	77.0	93.0	10.5M	1.8G
SAN10-patch.	star-product	78.7	94.0	10.9M	1.7G
	clique-product	79.1	94.2	11.5M	1.9G
	concatenation	79.3	94.2	11.8M	1.9G

Transformation Fn ABlation

Different Transformation Fns better accuracy + you can control to reduce number of parameters

Method		top-1	top-5	Params	Flops
Conv.-ResNet26		76.0	92.8	13.7M	2.4G
SAN10-pair.	$\varphi = \psi = \beta$	76.5	92.8	9.5M	3.0G
	$\varphi = \psi \neq \beta$	76.3	92.6	10.0M	2.1G
	$\varphi \neq \psi \neq \beta$	77.4	93.3	10.5M	2.2G
SAN10-patch.	$\varphi = \psi = \beta$	78.9	94.1	13.4M	2.2G
	$\varphi = \psi \neq \beta$	79.0	94.0	11.3M	1.8G
	$\varphi \neq \psi \neq \beta$	79.3	94.2	11.8M	1.9G

FootPrint Ablation

Method		top-1	top-5	Params	Flops
Conv.-ResNet26	3×3	76.0	92.8	13.7M	2.4G
	5×5	77.4	93.6	22.7M	4.0G
	7×7	77.9	93.7	36.1M	6.5G
SAN10-pair.	3×3	75.3	92.0	10.5M	1.7G
	5×5	76.6	92.9	10.5M	1.9G
	7×7	77.4	93.3	10.5M	2.2G
	9×9	77.8	93.5	10.5M	2.5G
	11×11	77.6	93.3	10.5M	3.0G
SAN10-patch.	3×3	77.4	93.4	10.7M	1.6G
	5×5	78.7	94.0	11.2M	1.7G
	7×7	79.3	94.2	11.8M	1.9G
	9×9	79.3	94.1	12.7M	2.1G
	11×11	79.4	94.1	13.8M	2.3G

Larger footprints leads to significant increase of computations in Convolutional Nets unlike Self Attention Nets.

Position Encoding Ablation

Relative position encoding has great influence on pairwise attention accuracy

Method		top-1	top-5	Params	Flops
Conv.-ResNet26		76.0	92.8	13.7M	2.4G
SAN10-pair.	none	72.3	90.3	10.5M	2.1G
	absolute	74.7	91.7	10.5M	2.2G
	relative	77.4	93.3	10.5M	2.2G

Robustness

Pairwise attention is more robust than CNNs or patchwise att. Because it is a set operator

Method	no rotation		clockwise 90°		clockwise 180°		clockwise 270°		upside-down	
	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
ResNet26	73.6	91.7	49.1(24.5)	72.7(19.0)	50.6(23.0)	75.4(16.3)	49.2(24.4)	72.8(18.9)	50.5(23.1)	75.4(16.3)
SAN10-pair.	74.9	92.1	51.8(**23.1**)	74.6(**17.5**)	54.7(**20.2**)	78.5(**13.6**)	51.7(23.2)	74.5(**17.6**)	54.7(**20.2**)	78.5(**13.6**)
SAN10-patch.	77.1	93.5	53.1(24.0)	75.7(17.8)	54.6(22.5)	78.4(15.1)	53.3(23.8)	76.0(17.5)	54.7(22.4)	78.3(15.2)
ResNet38	76.0	93.0	51.2(24.8)	74.2(18.8)	52.2(23.8)	76.9(16.1)	51.6(24.4)	74.6(18.4)	52.2(23.8)	76.8(16.2)
SAN15-pair.	76.6	93.1	54.5(**22.1**)	77.1(**16.0**)	57.9(**18.7**)	80.8(**12.3**)	54.8(**21.8**)	77.0(**16.1**)	58.0(**18.6**)	80.8(**12.3**)
SAN15-patch.	78.0	93.9	53.7(24.5)	76.1(17.8)	56.0(22.2)	79.5(14.4)	53.9(24.3)	76.2(17.7)	56.0(22.2)	79.4(14.5)
ResNet50	76.9	93.5	52.6(24.3)	75.3(18.2)	52.9(24.0)	77.4(16.2)	52.6(24.3)	75.5(18.0)	53.0(23.9)	77.3(16.2)
SAN19-pair.	76.9	93.4	54.7(**22.2**)	77.1(**16.3**)	58.0(**18.9**)	80.4(**13.0**)	55.0(**21.9**)	77.1(**16.3**)	57.9(**19.0**)	80.4(**13.0**)
SAN19-patch.	78.2	93.9	54.2(24.0)	76.3(17.6)	56.2(22.0)	79.5(14.4)	54.1(24.1)	76.4(17.5)	56.3(21.9)	79.5(14.4)