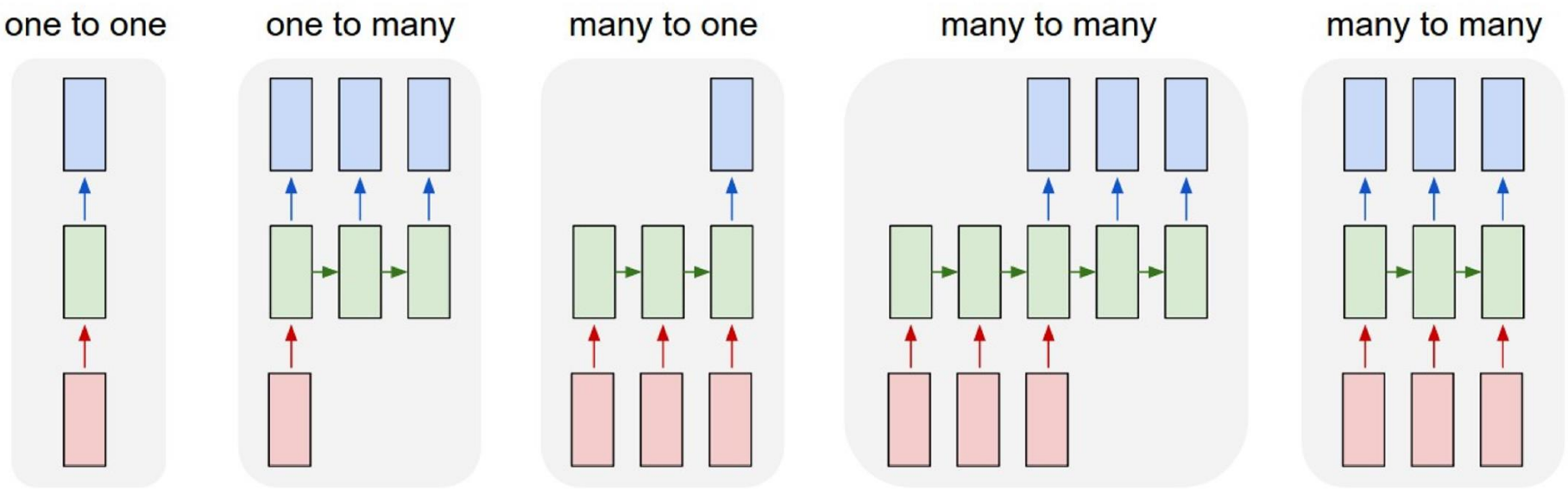
# Attention

- Attention on RNN
- Self-attention
- Transformer

source from University of Michigan EECS498

# Last Time: Recurrent Neural Networks

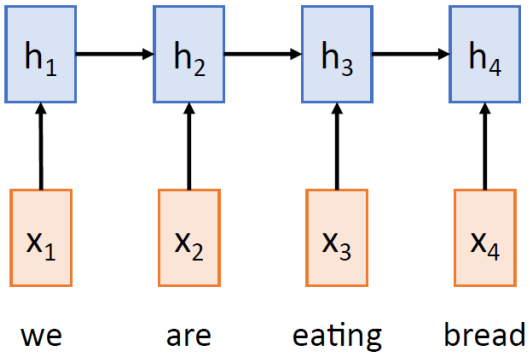


# Sequence-to-Sequence with RNNs

**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$




Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

**Input**: Sequence $x_1, \ldots x_T$

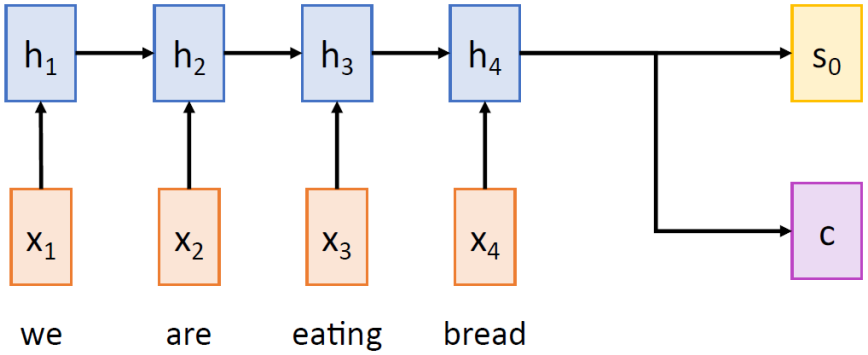
**Output**: Sequence $y_1, \ldots, y_{T'}$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

**Initial decoder state** $s_0$

**Context vector** c (often $c=h_T$)




Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

**Input**: Sequence $x_1, \ldots x_T$

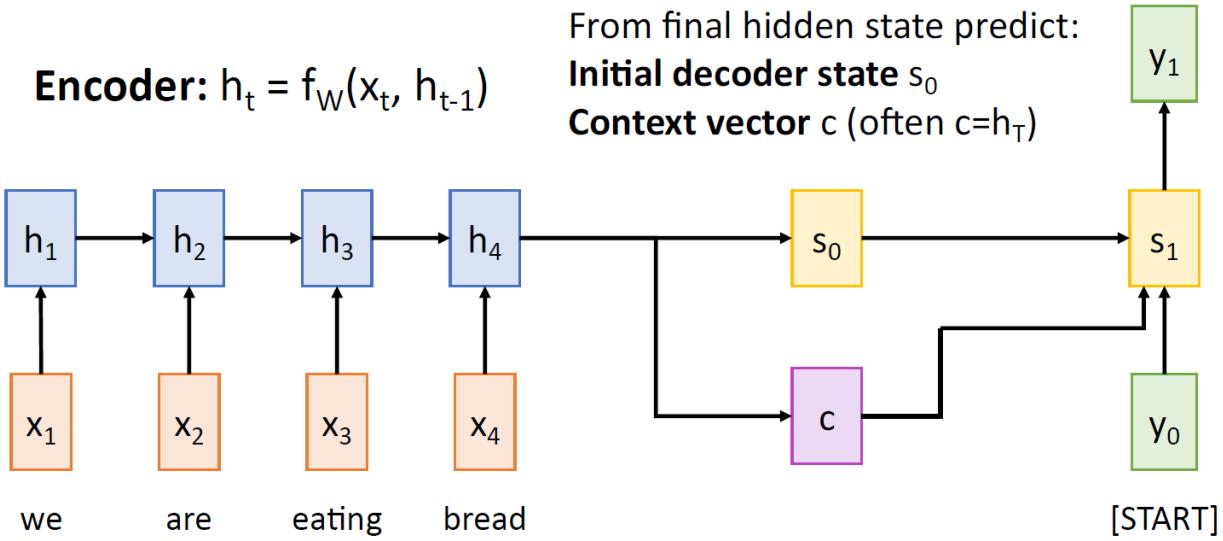
**Output**: Sequence $y_1, \ldots, y_{T'}$

**Decoder:** $s_t = g_U(y_{t-1}, s_{t-1}, c)$

我們

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** c (often $c=h_T$)




Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

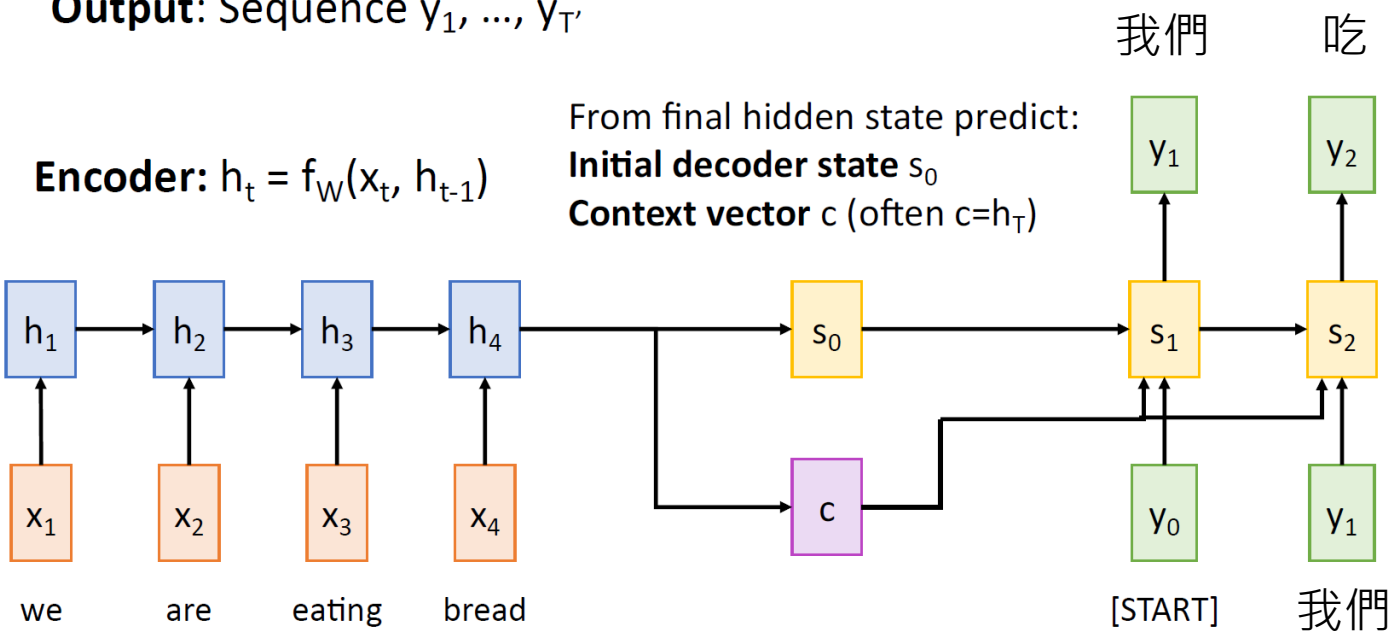
**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Decoder:** $s_t = g_U(y_{t-1}, s_{t-1}, c)$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** c (often $c=h_T$)




Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

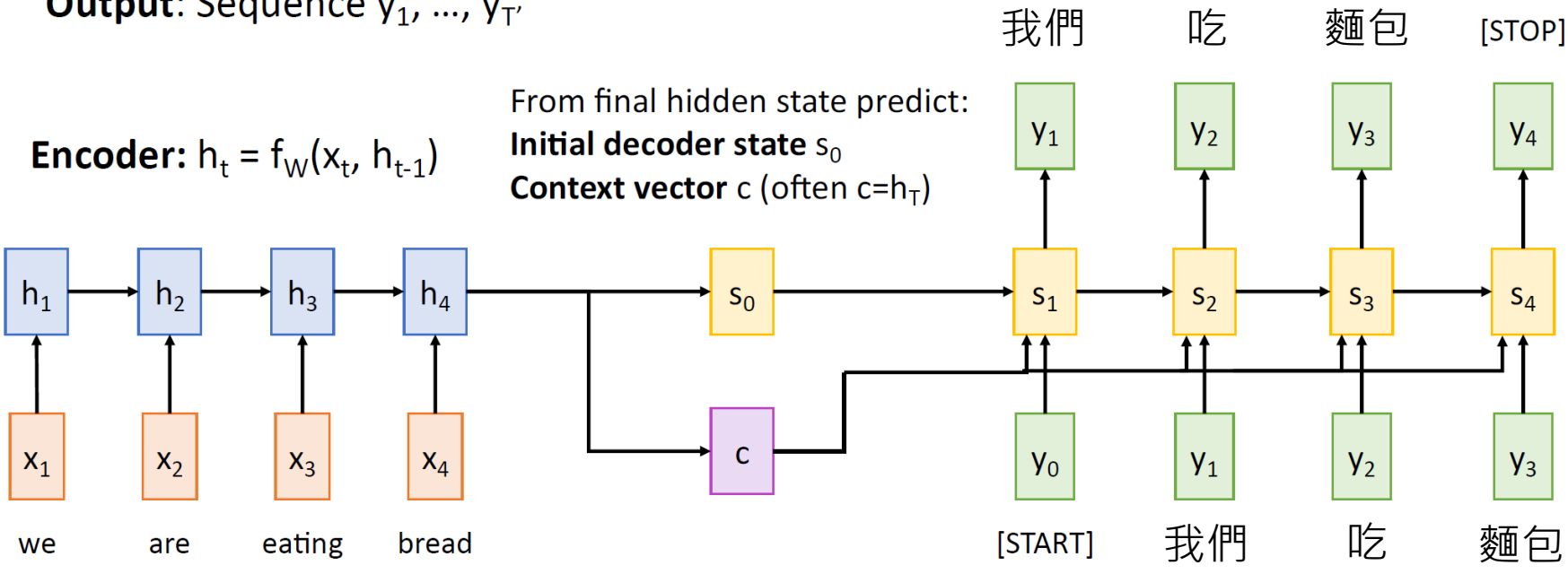
**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Decoder:** $s_t = g_U(y_{t-1}, s_{t-1}, c)$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** c (often $c=h_T$)




Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

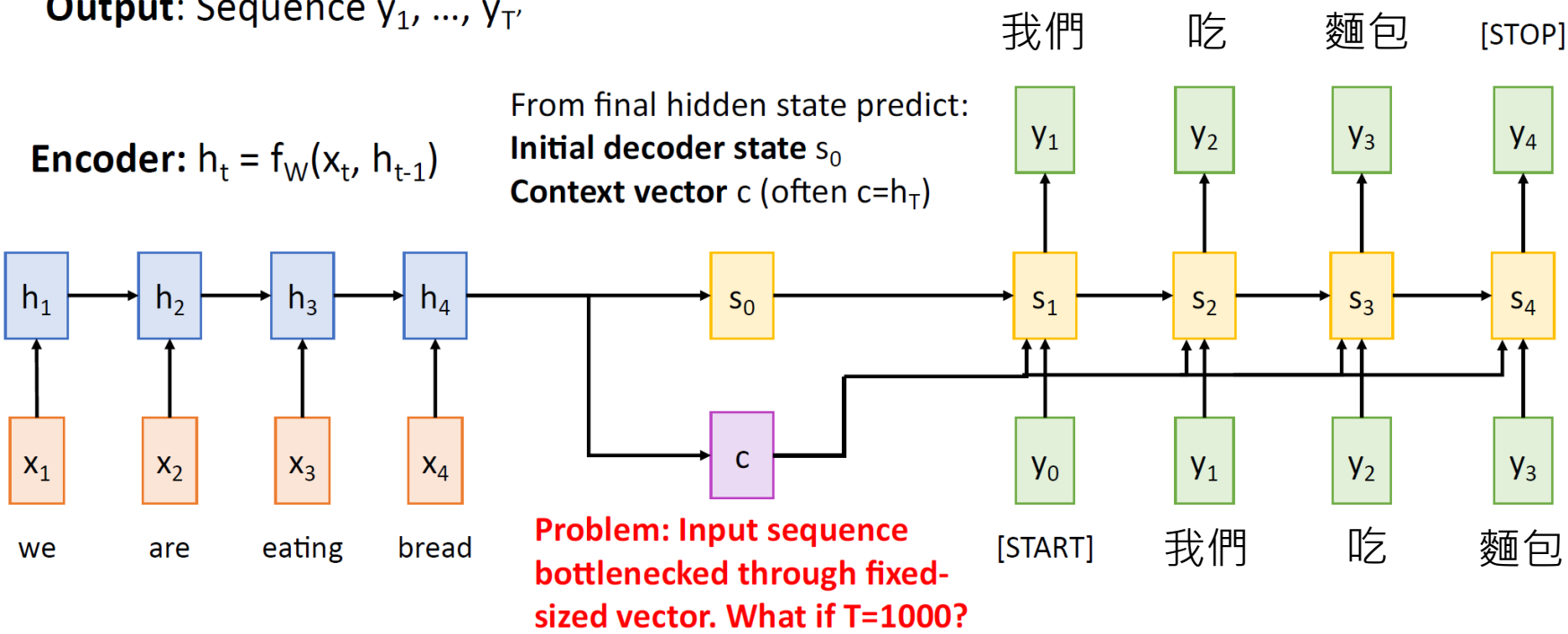
**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Decoder:** $s_t = g_U(y_{t-1}, s_{t-1}, c)$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** c (often $c=h_T$)




**Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?**

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

# Sequence-to-Sequence with RNNs

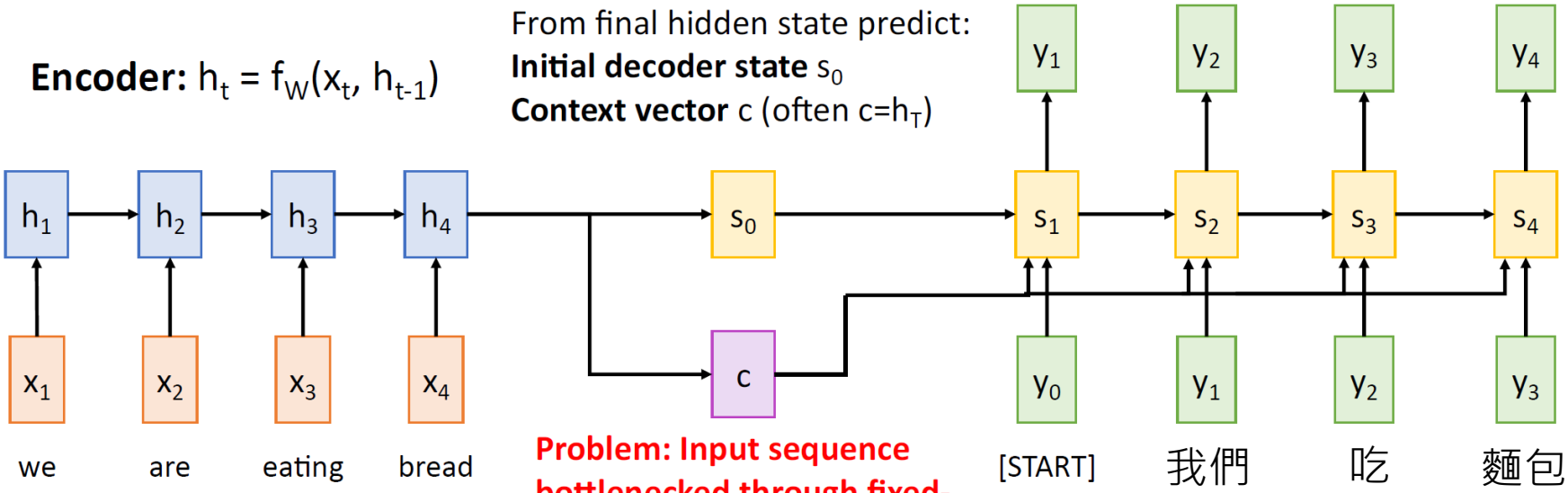
**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

**Decoder:** $s_t = g_U(y_{t-1}, s_{t-1}, c)$

From final hidden state predict:
**Initial decoder state** $s_0$
**Context vector** c (often $c=h_T$)




**Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?**

**Idea: use new context vector at each step of decoder!**

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

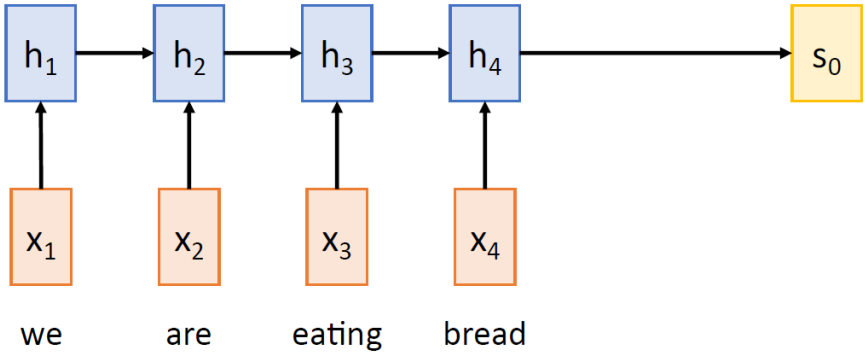
# Sequence-to-Sequence with RNNs **and Attention**

**Input**: Sequence $x_1, \ldots x_T$

**Output**: Sequence $y_1, \ldots, y_{T'}$

**Encoder:** $h_t = f_W(x_t, h_{t-1})$

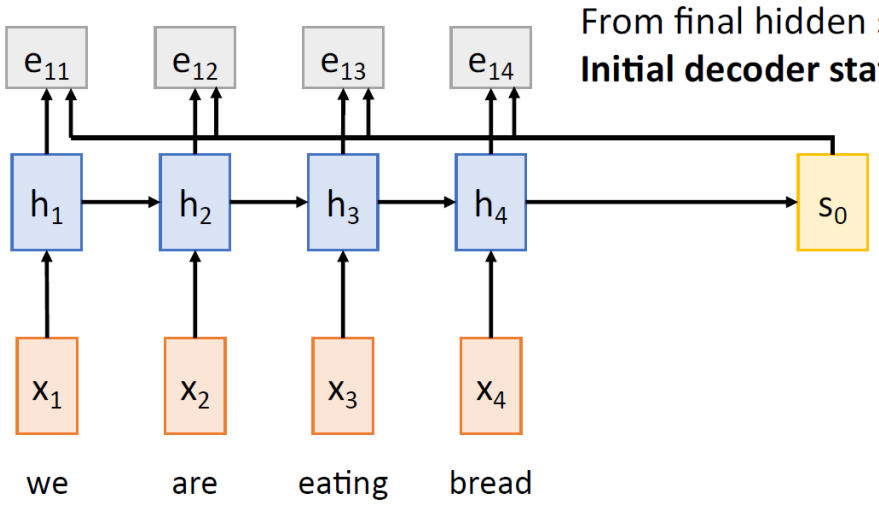
From final hidden state:
**Initial decoder state** $s_0$




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

# Sequence-to-Sequence with RNNs **and Attention**

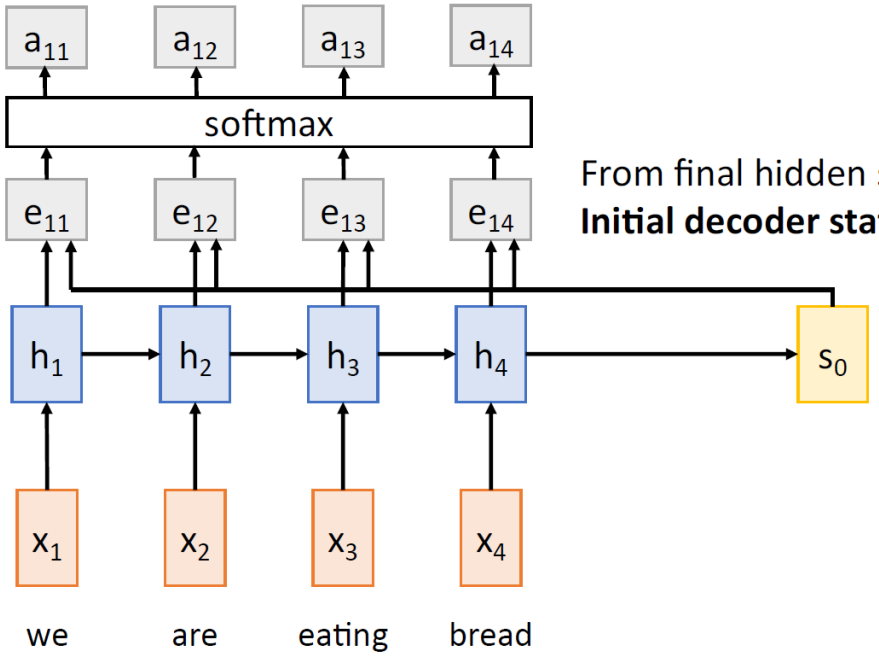
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ ($f_{att}$ is an MLP)




From final hidden state:
**Initial decoder state** $s_0$

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

# Sequence-to-Sequence with RNNs and Attention



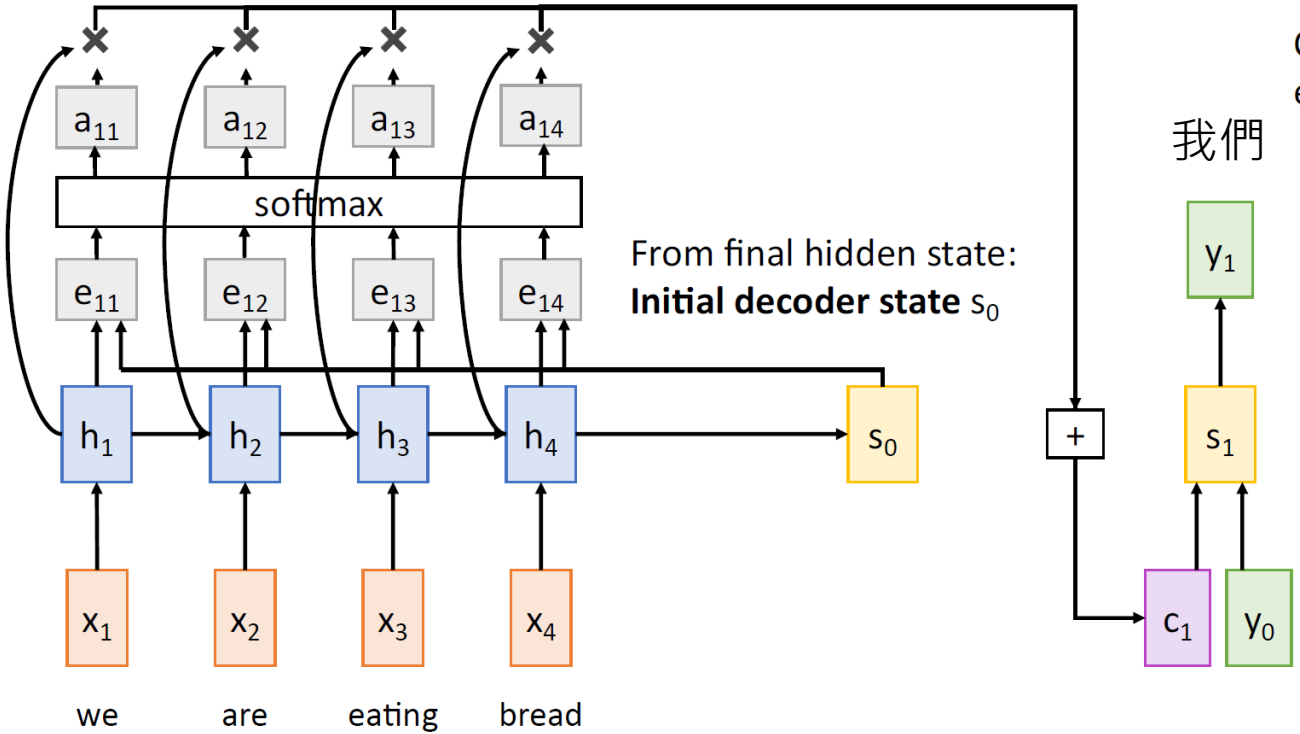

From final hidden state:
**Initial decoder state** $s_0$

Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ ($f_{att}$ is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

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

# Sequence-to-Sequence with RNNs and Attention




Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ ($f_{att}$ is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

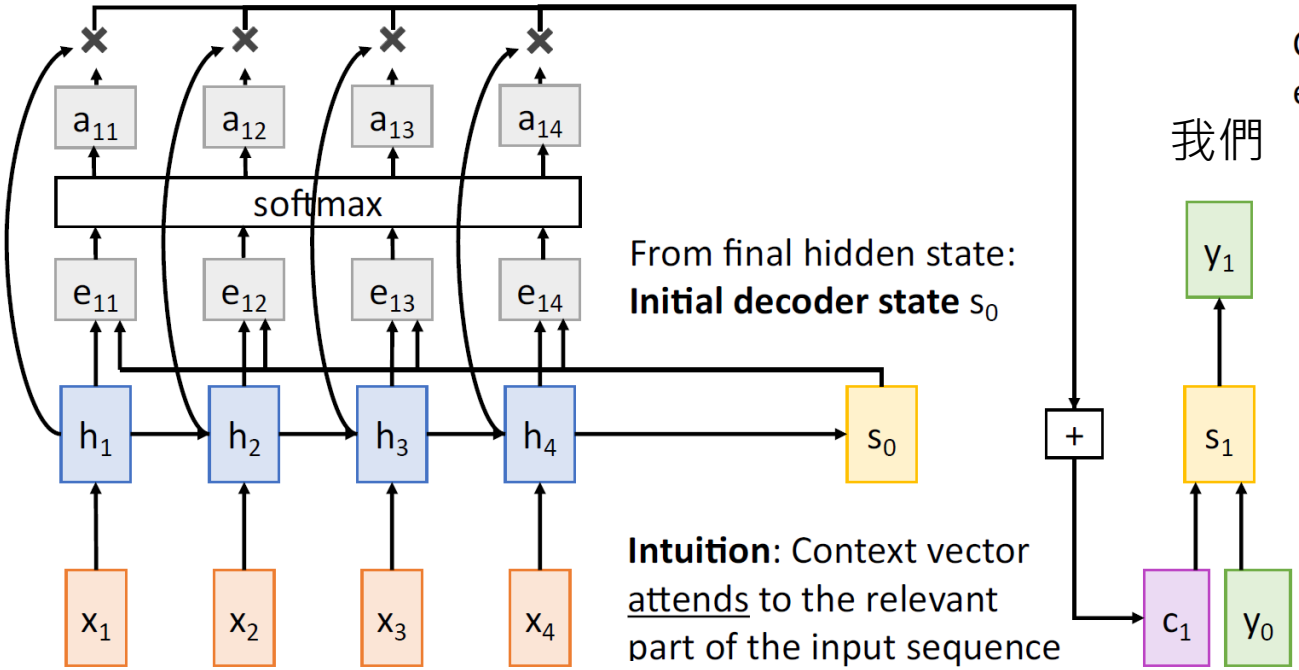
$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

**This is all differentiable! Do not supervise attention weights – backprop through everything**

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

# Sequence-to-Sequence with RNNs and Attention




From final hidden state:
**Initial decoder state** $s_0$

**Intuition**: Context vector attends to the relevant part of the input sequence
"我們" = *"we are"*
so maybe $a_{11}$=$a_{12}$=0.45, $a_{13}$=$a_{14}$=0.05

Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ ($f_{att}$ is an MLP)

Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

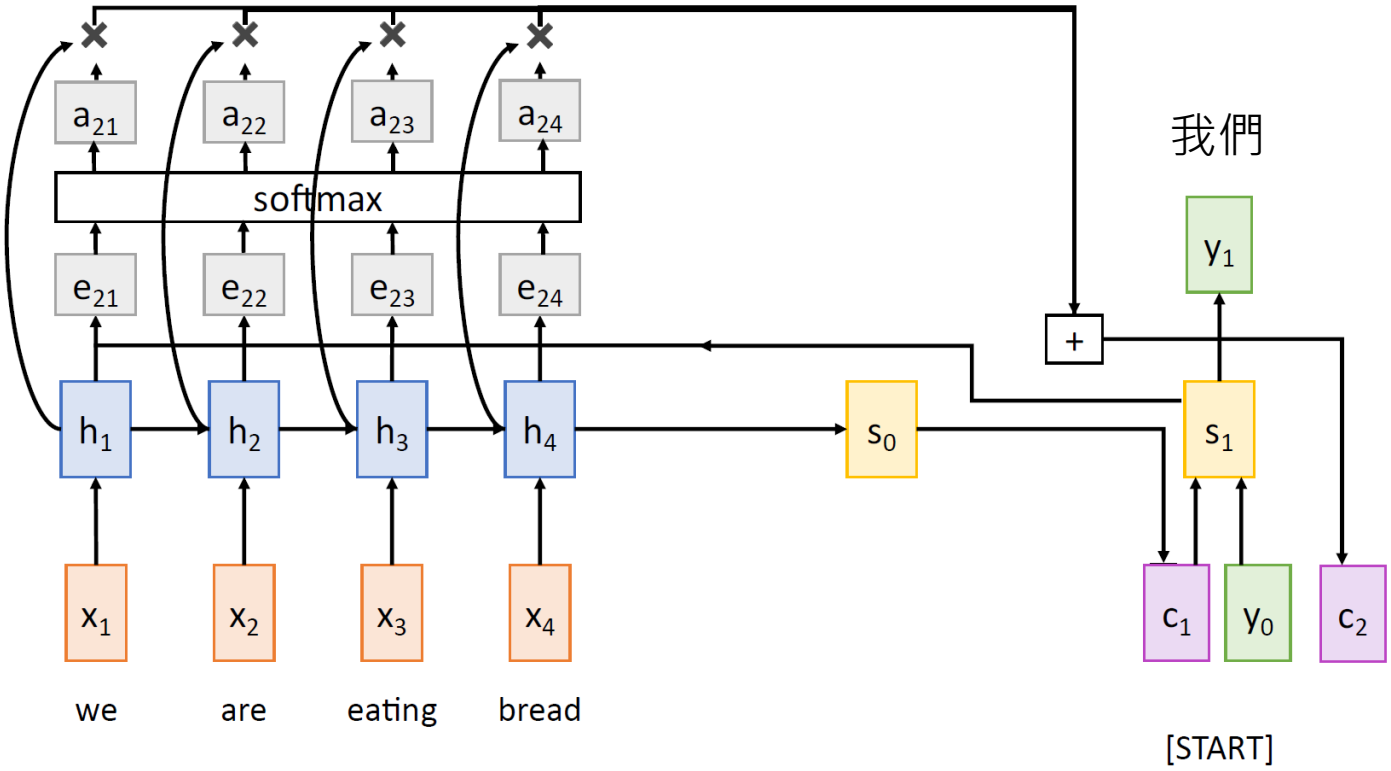
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

**This is all differentiable! Do not supervise attention weights – backprop through everything**

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

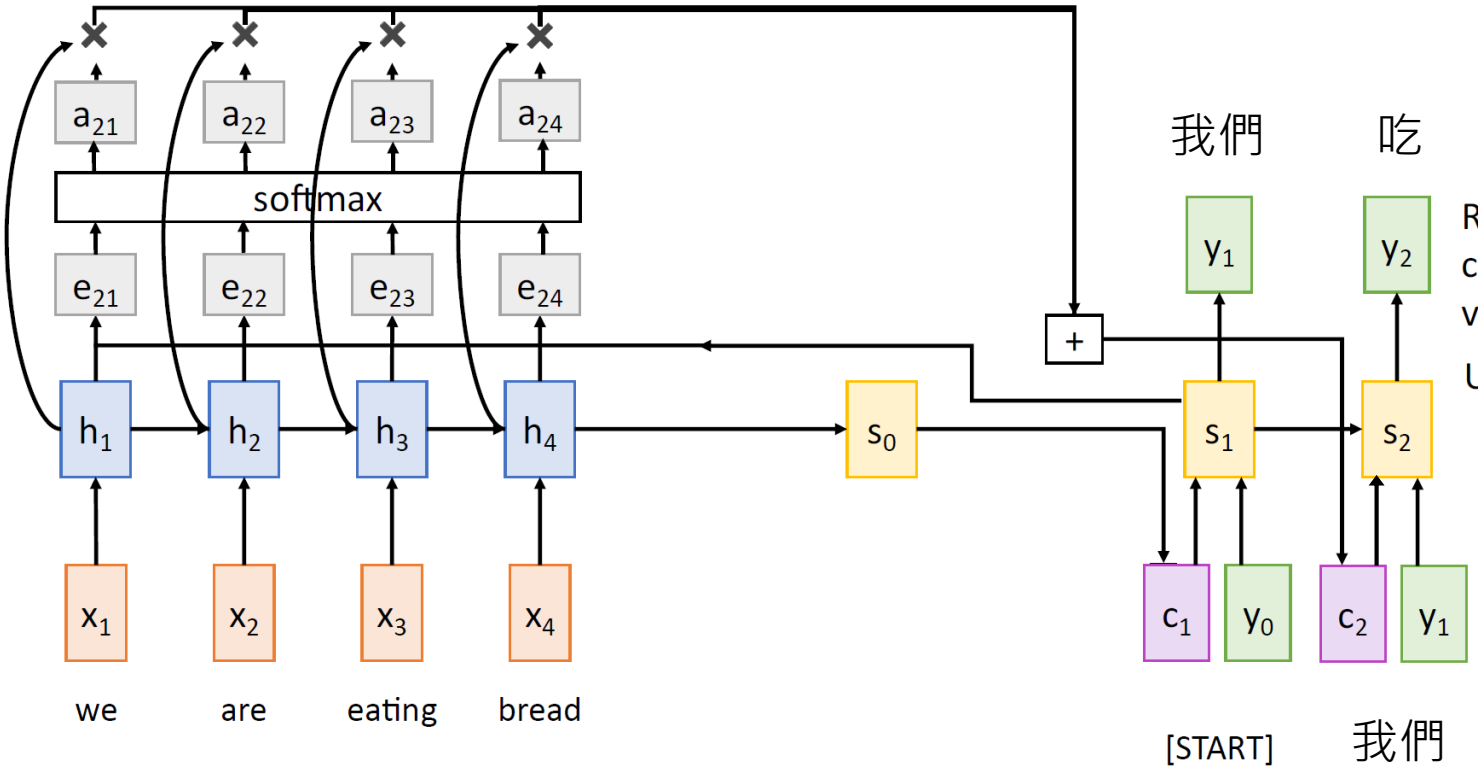
# Sequence-to-Sequence with RNNs

Repeat: Use $s_1$ to compute new context vector $c_2$



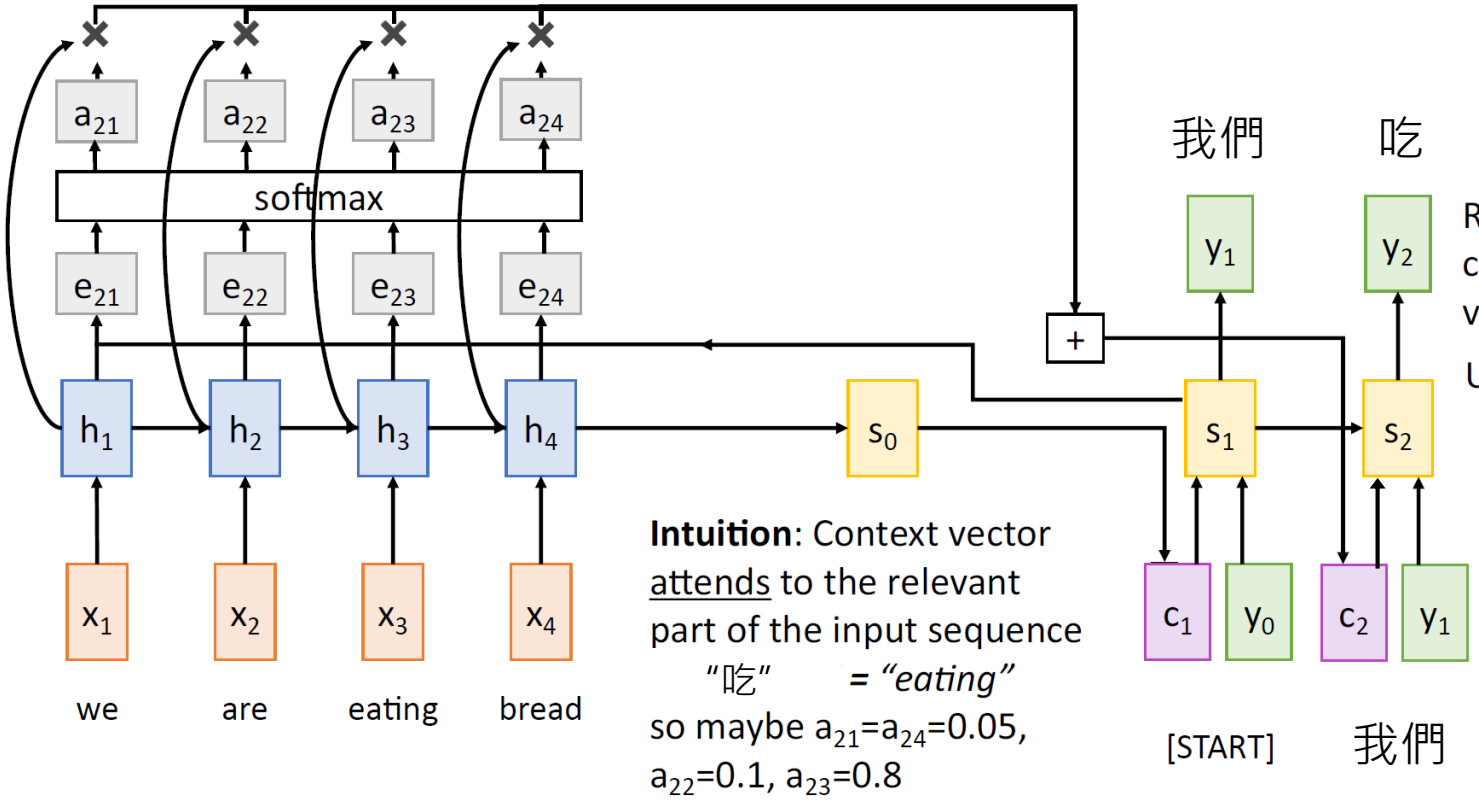

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

# Sequence-to-Sequence with RNNs and Attention




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

# Sequence-to-Sequence with RNNs and Attention




Repeat: Use $s_1$ to compute new context vector $c_2$

Use $c_2$ to compute $s_2$, $y_2$

**Intuition**: Context vector attends to the relevant part of the input sequence

"吃" = *"eating"*

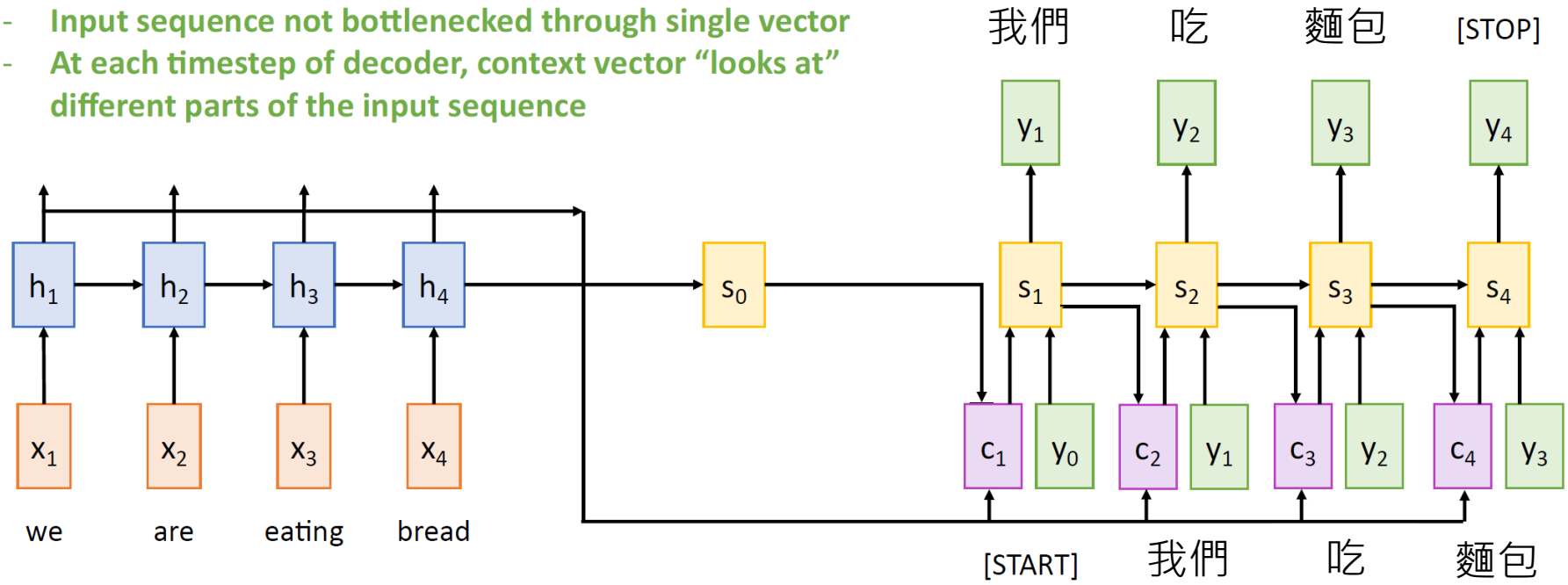
so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

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

# Sequence-to-Sequence with RNNs **and Attention**

**Use a different context vector in each timestep of decoder**

- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**




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

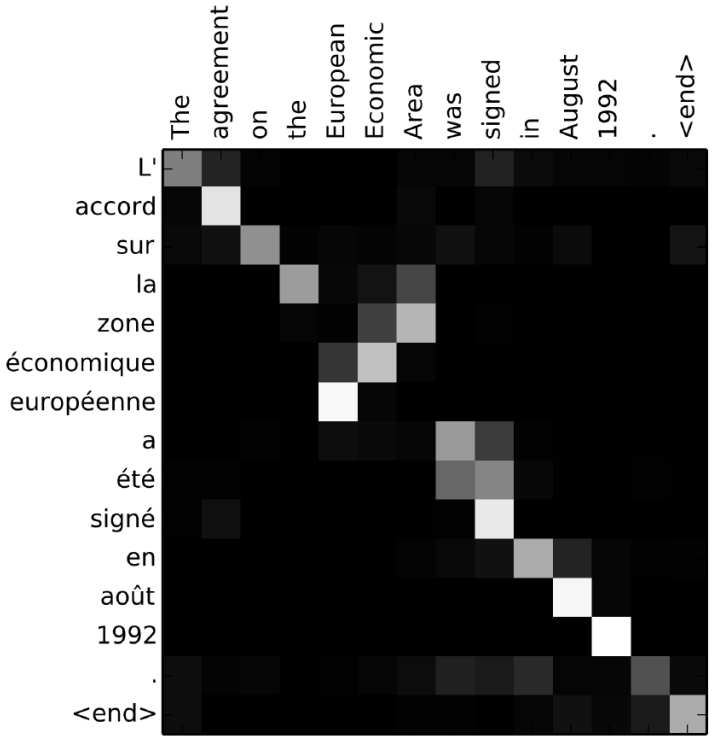
# Sequence-to-Sequence with RNNs and Attention

**Example**: English to French translation

**Input**: “The agreement on the European Economic Area was signed in August 1992.”

**Output**: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$




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

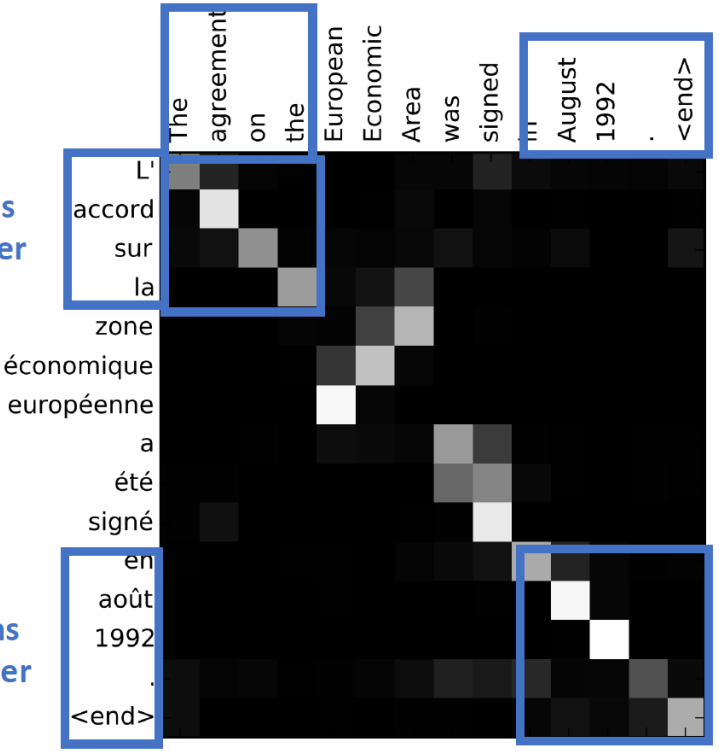
# Sequence-to-Sequence with RNNs and Attention

**Example**: English to French translation

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

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

Visualize attention weights $a_{t,i}$




Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

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

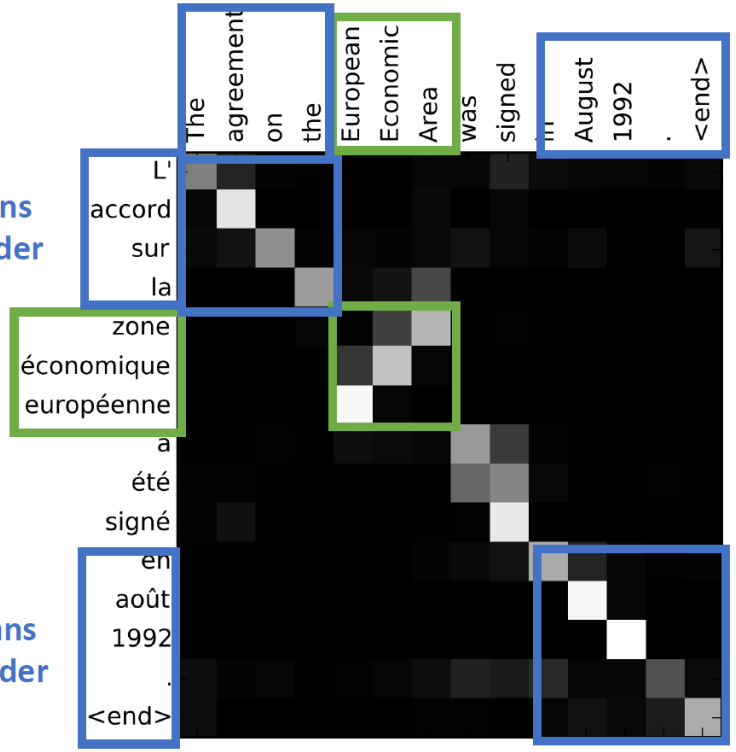
# Sequence-to-Sequence with RNNs and Attention

**Example**: English to French translation

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

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

Visualize attention weights $a_{t,i}$




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

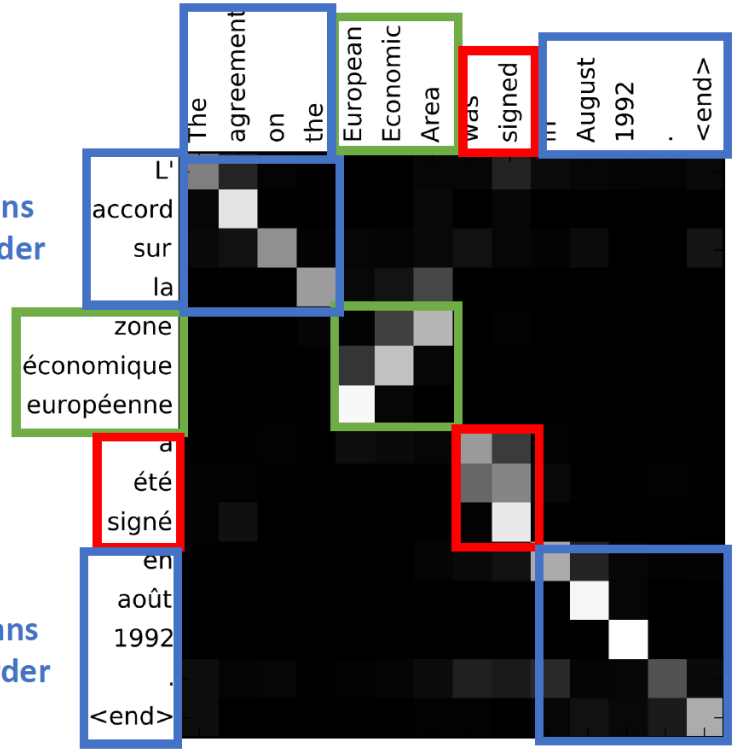
# Sequence-to-Sequence with RNNs and Attention

**Example**: English to French translation

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

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

Visualize attention weights $a_{t,i}$

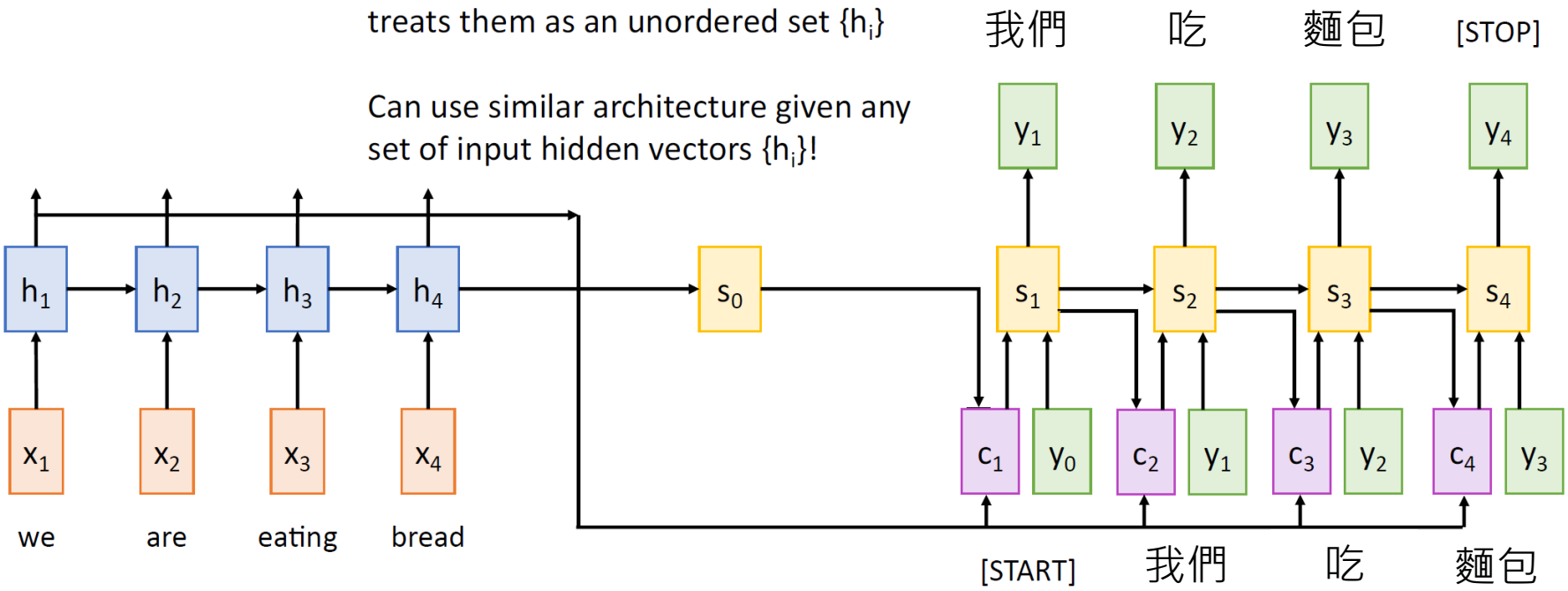



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

# Sequence-to-Sequence with RNNs and Attention

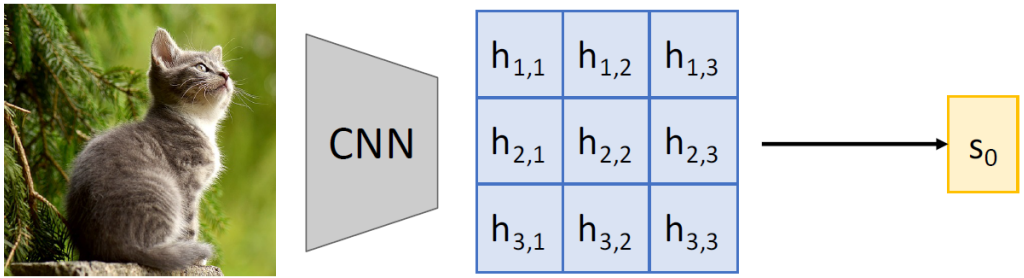
The decoder doesn't use the fact that $h_i$ form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!




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

# Image Captioning with RNNs and Attention




Use a CNN to compute a grid of features for an image

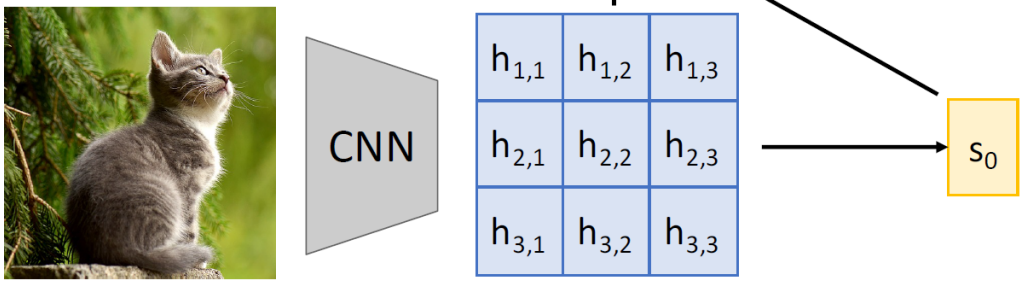


Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

Alignment scores



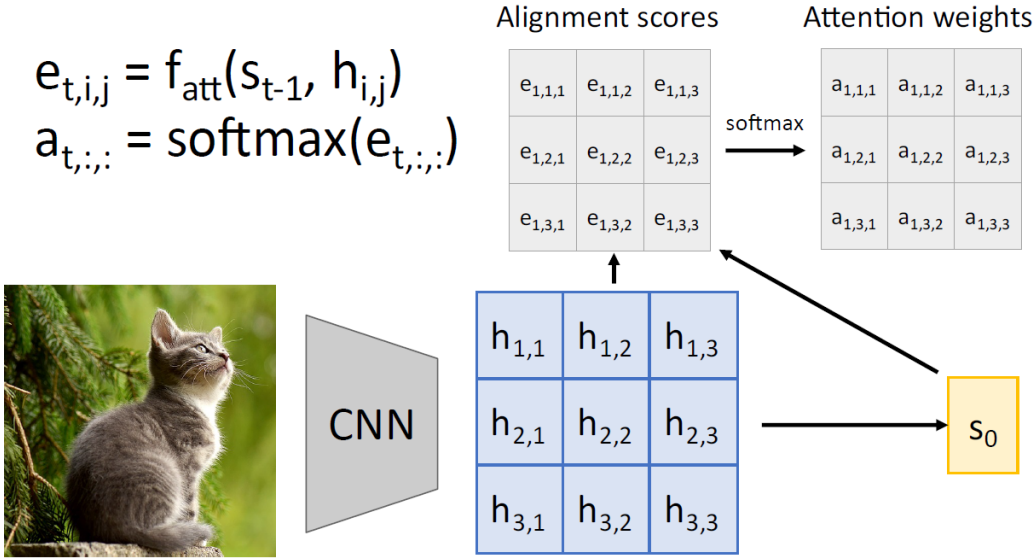

Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$




Use a CNN to compute a grid of features for an image

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

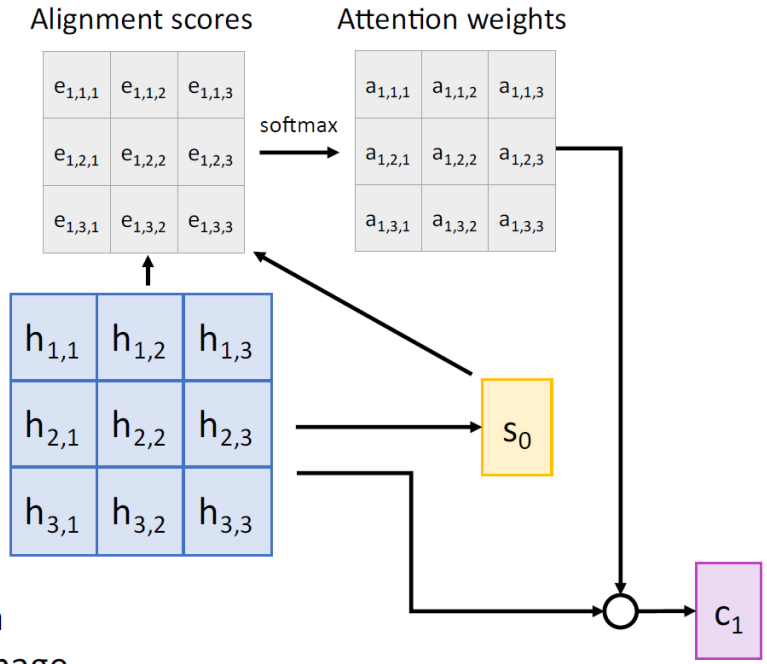
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

Use a CNN to compute a grid of features for an image




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

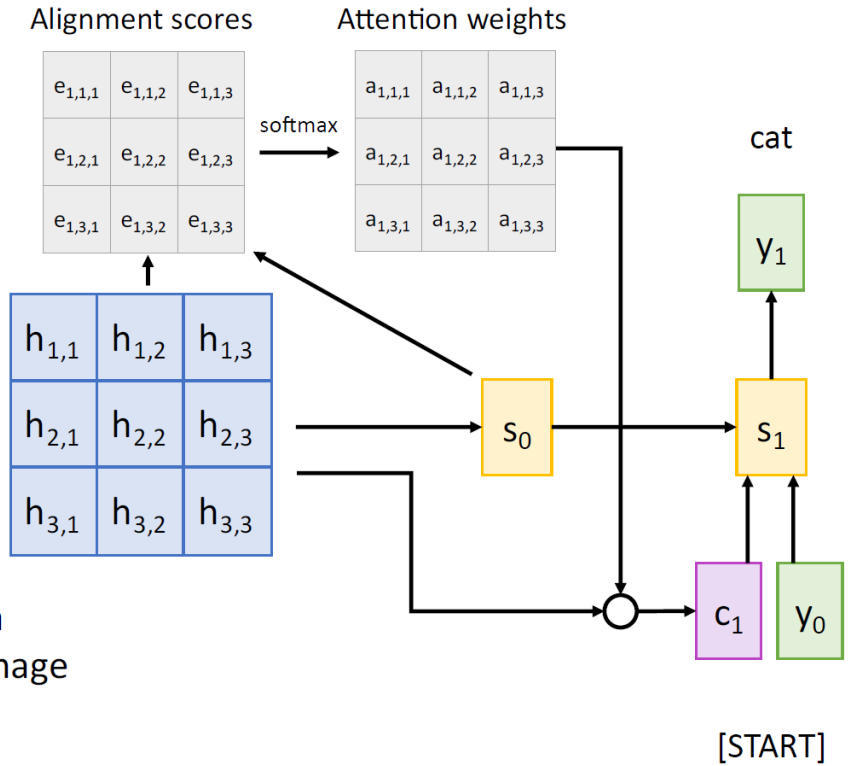
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

Use a CNN to compute a grid of features for an image




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

$s_0$ $s_1$

$c_1$ $y_0$ $y_1$

cat

[START]

Use a CNN to compute a grid of features for an image

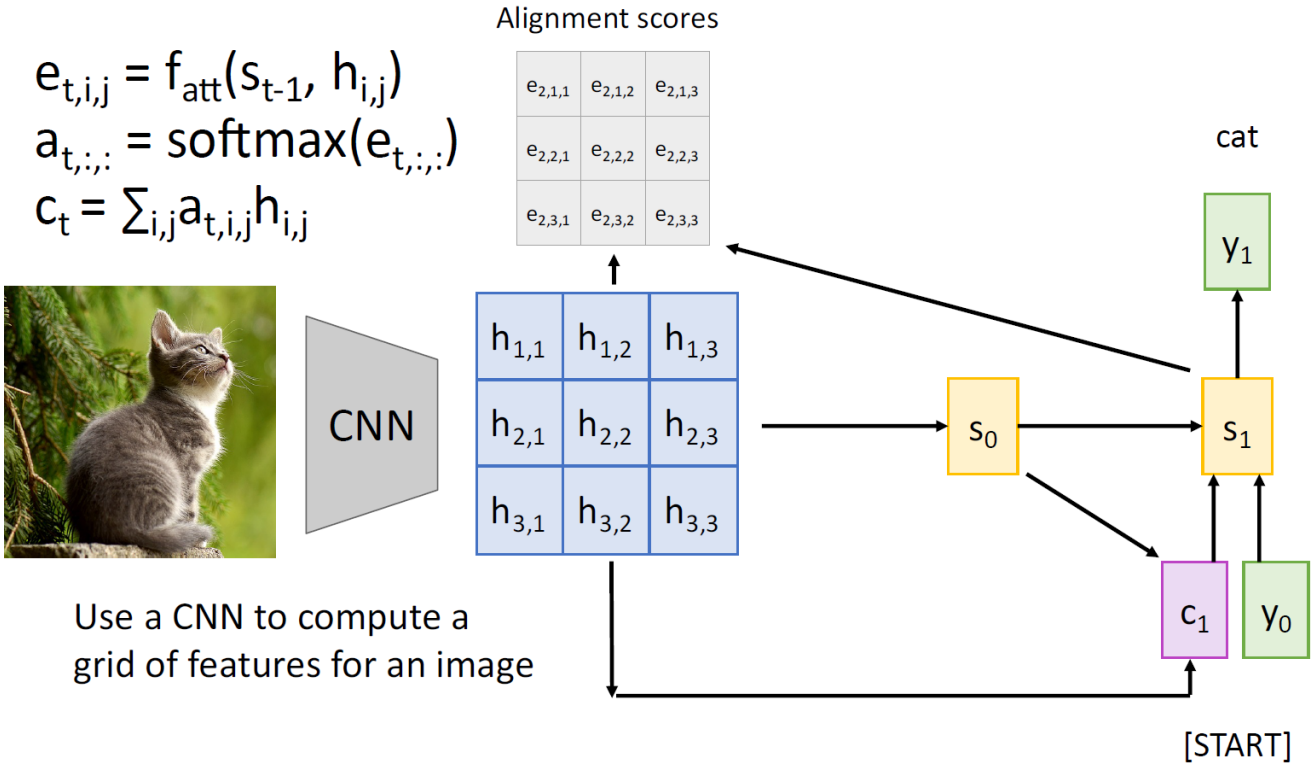
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$




Use a CNN to compute a grid of features for an image

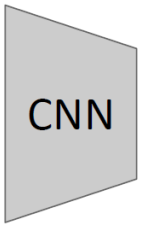
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

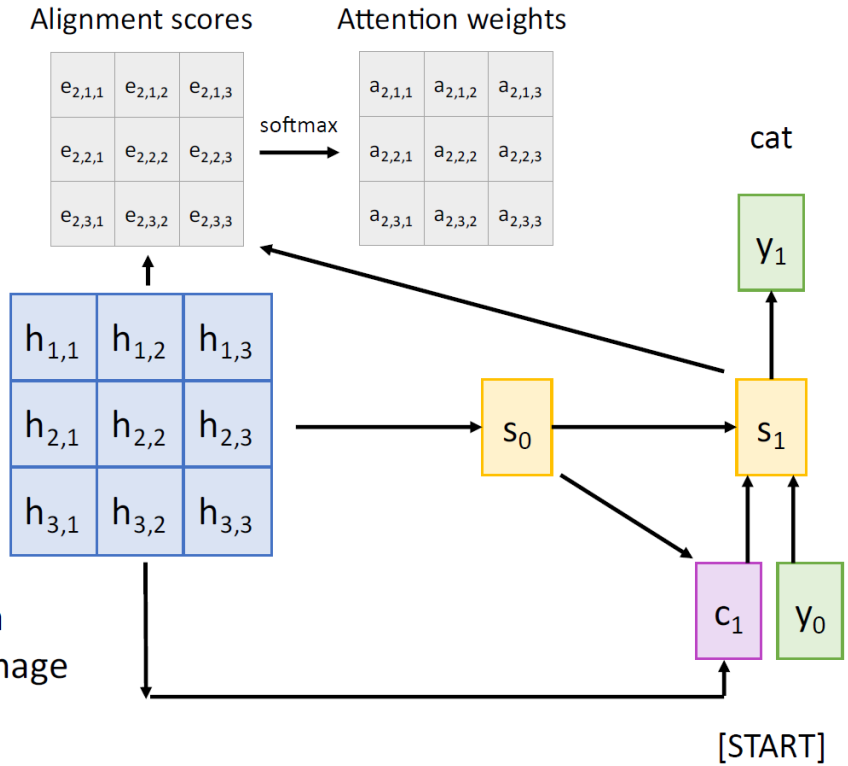
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$




Use a CNN to compute a grid of features for an image




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

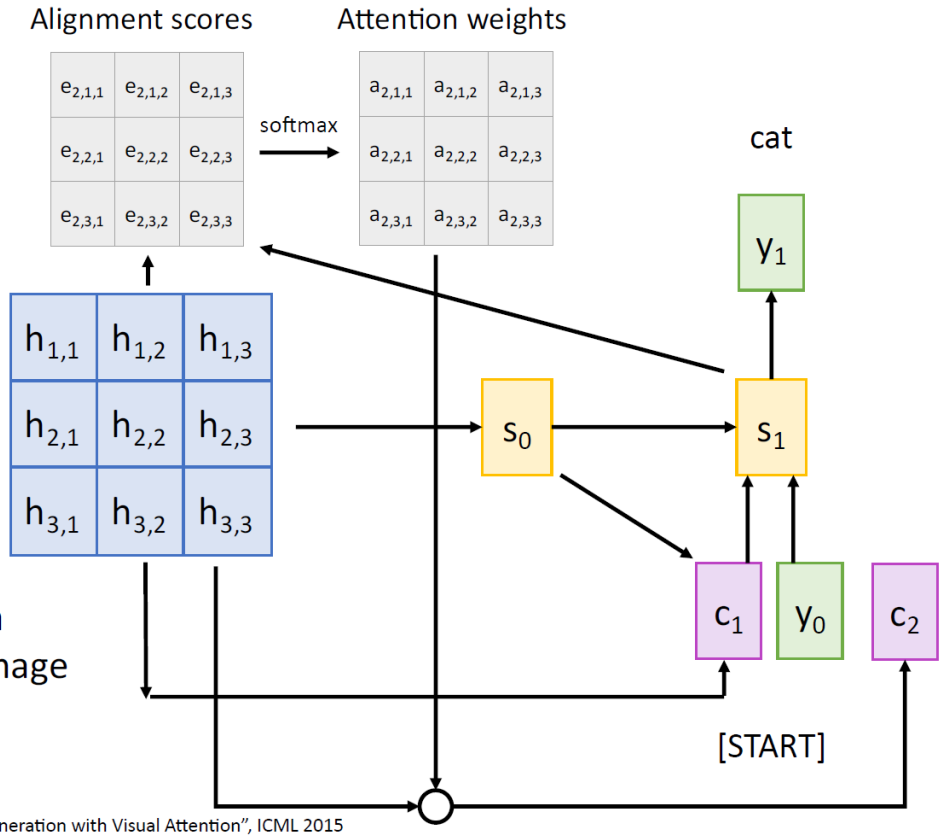
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

Use a CNN to compute a grid of features for an image




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

$a_{t,:,:} = \text{softmax}(e_{t,:,:})$

$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

CNN

Use a CNN to compute a grid of features for an image

Alignment scores

$e_{2,1,1}$ $e_{2,1,2}$ $e_{2,1,3}$
$e_{2,2,1}$ $e_{2,2,2}$ $e_{2,2,3}$
$e_{2,3,1}$ $e_{2,3,2}$ $e_{2,3,3}$

softmax

Attention weights

$a_{2,1,1}$ $a_{2,1,2}$ $a_{2,1,3}$
$a_{2,2,1}$ $a_{2,2,2}$ $a_{2,2,3}$
$a_{2,3,1}$ $a_{2,3,2}$ $a_{2,3,3}$

$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

$s_0$ $s_1$ $s_2$

$y_1$ $y_2$

cat sitting

$c_1$ $y_0$ $c_2$ $y_1$

[START] cat

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

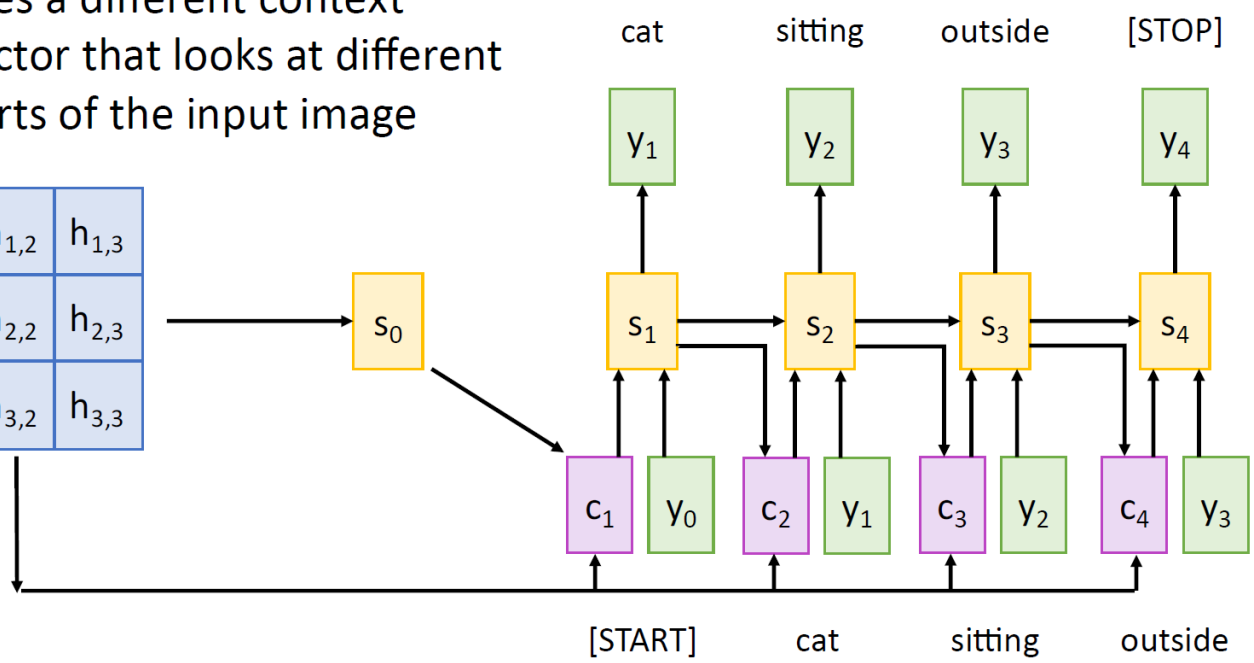
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

Each timestep of decoder uses a different context vector that looks at different parts of the input image

CNN

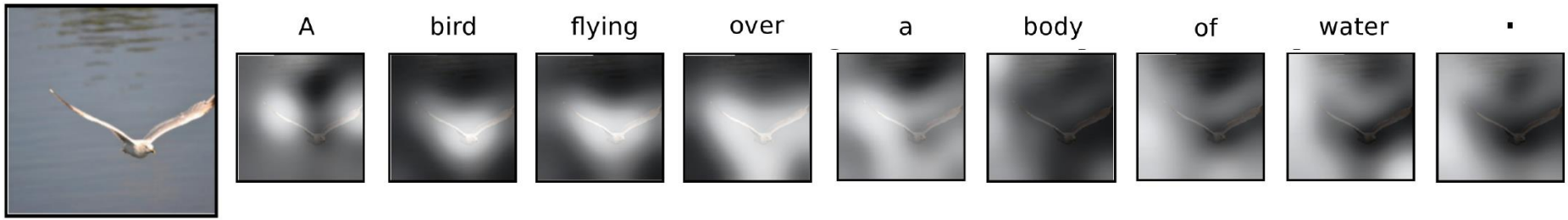
$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

Use a CNN to compute a grid of features for an image




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs and Attention

A dog is standing on a hardwood floor.

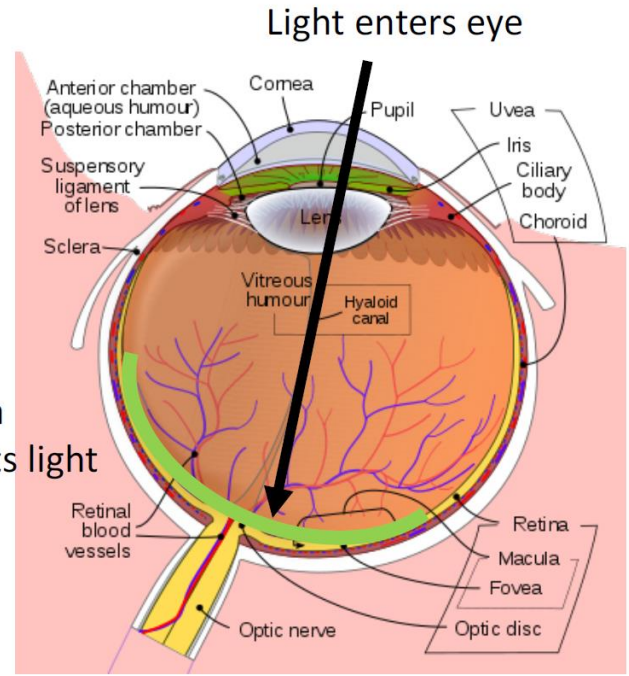

A stop sign is on a road with a mountain in the background.

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

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

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

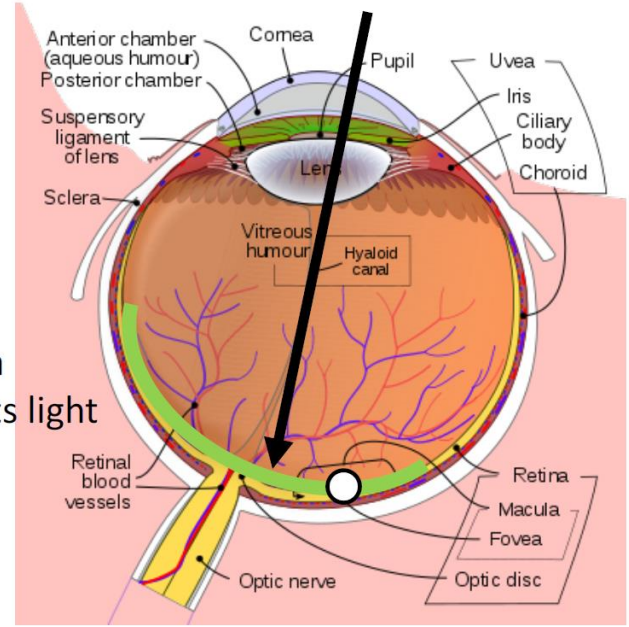
# Human Vision: Fovea




**Retina**
detects light

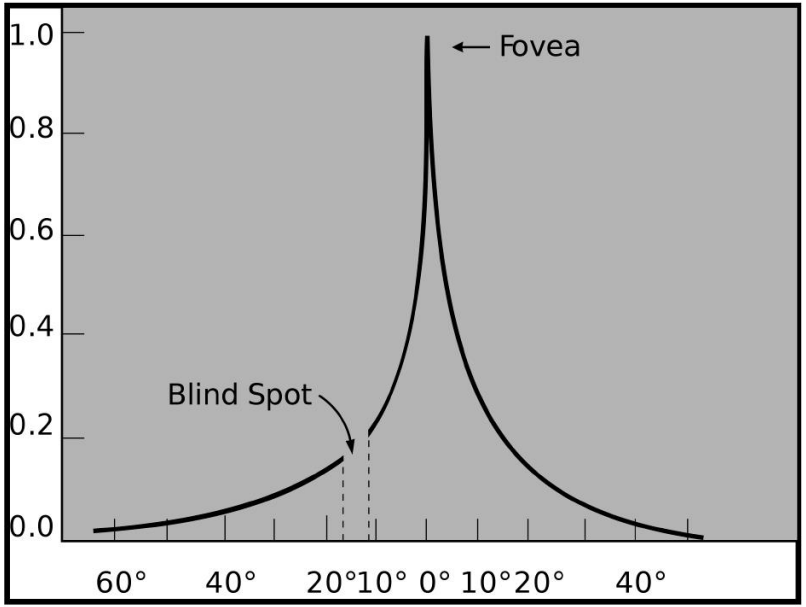
# Human Vision: Fovea

Light enters eye



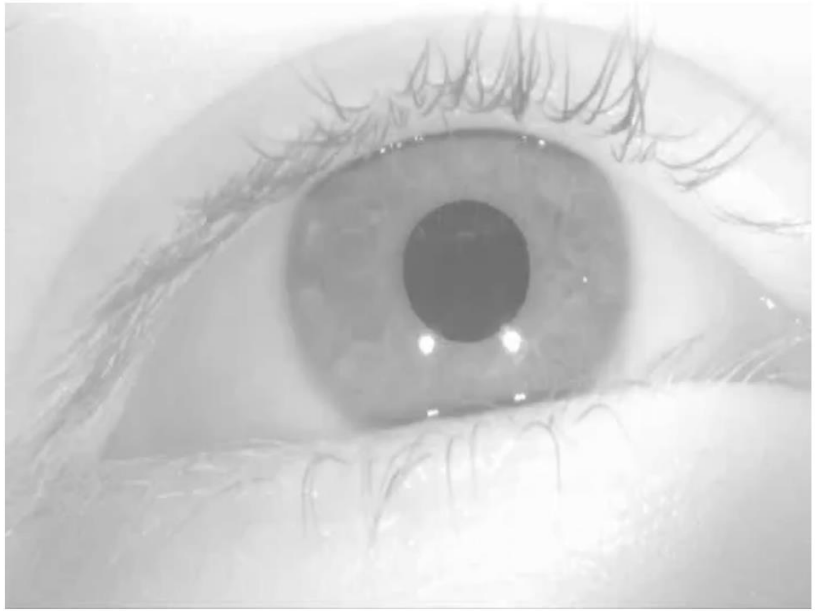

**Retina** detects light

The **fovea** is a tiny region of the retina that can see with high acuity

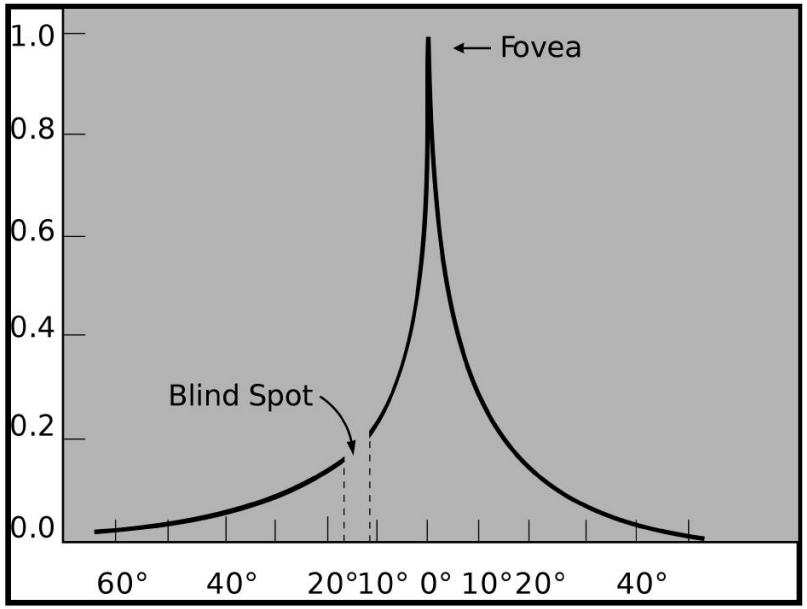


# Human Vision: Saccades

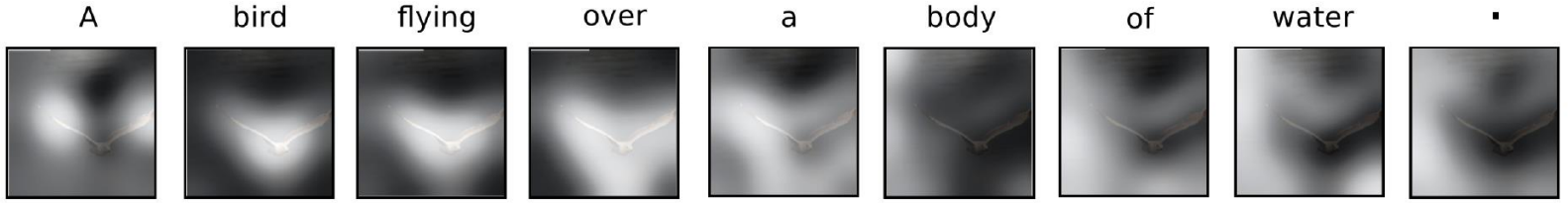
Human eyes are constantly moving so we don't notice



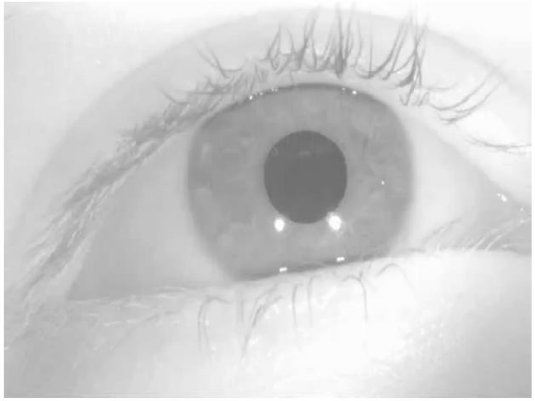
The **fovea** is a tiny region of the retina that can see with high acuity



# Image Captioning with RNNs and Attention




Attention weights at each timestep kind of like saccades of human eye



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# X, Attend, and Y

"**Show, attend, and tell**" *(Xu et al, ICML 2015)*
Look at image, attend to image regions, produce question

"**Ask, attend, and answer**" *(Xu and Saenko, ECCV 2016)*
"**Show, ask, attend, and answer"** *(Kazemi and Elqursh, 2017)*
Read text of question, attend to image regions, produce answer

"**Listen, attend, and spell**" *(Chan et al, ICASSP 2016)*
Process raw audio, attend to audio regions while producing text

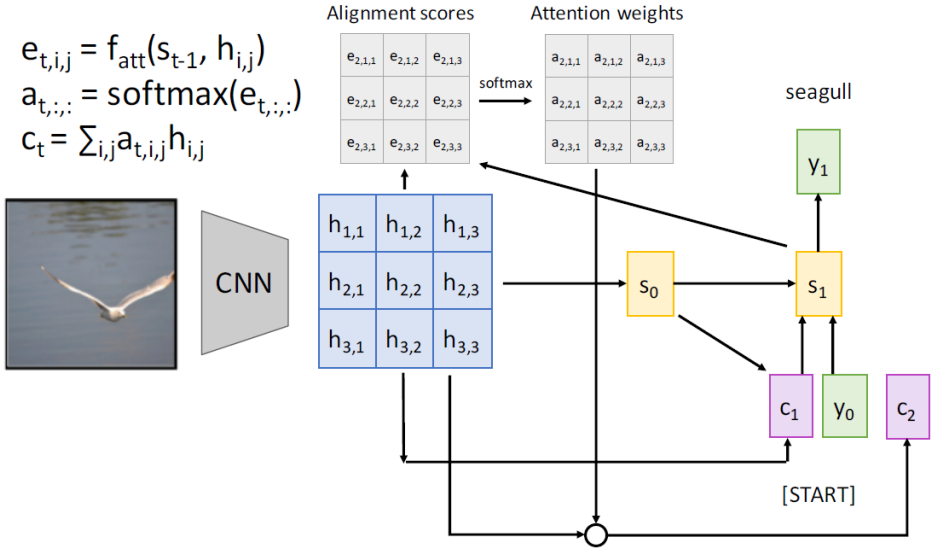
"**Listen, attend, and walk**" *(Mei et al, AAAI 2016)*
Process text, attend to text regions, output navigation commands

"**Show, attend, and interact**" *(Qureshi et al, ICRA 2017)*
Process image, attend to image regions, output robot control commands

"**Show, attend, and read**" *(Li et al, AAAI 2019)*
Process image, attend to image regions, output text

# Attention Layer

**Inputs**:
**Query vector**: **q** (Shape: $D_Q$)
**Input vectors**: **X** (Shape: $N_X$ x $D_X$)
**Similarity function**: $f_{att}$



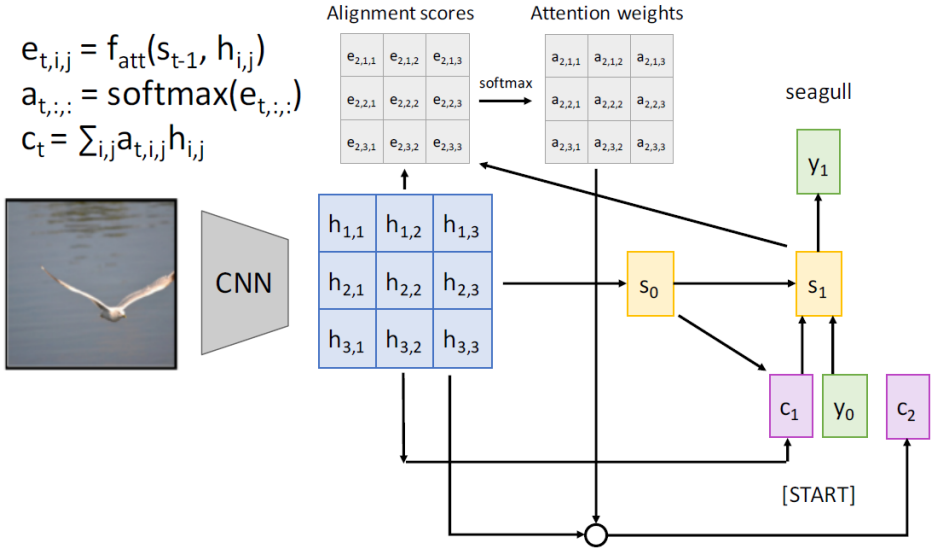

**Computation**:
**Similarities**: e (Shape: $N_X$) $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
**Attention weights**: a = softmax(e) (Shape: $N_X$)
**Output vector**: $y = \sum_i a_i \mathbf{X}_i$ (Shape: $D_X$)

# Attention Layer

**Inputs**:
**Query vector**: q (Shape: $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_Q$)
**Similarity function**: dot product

$$q \cdot x = |q||x|\cos(\theta)$$

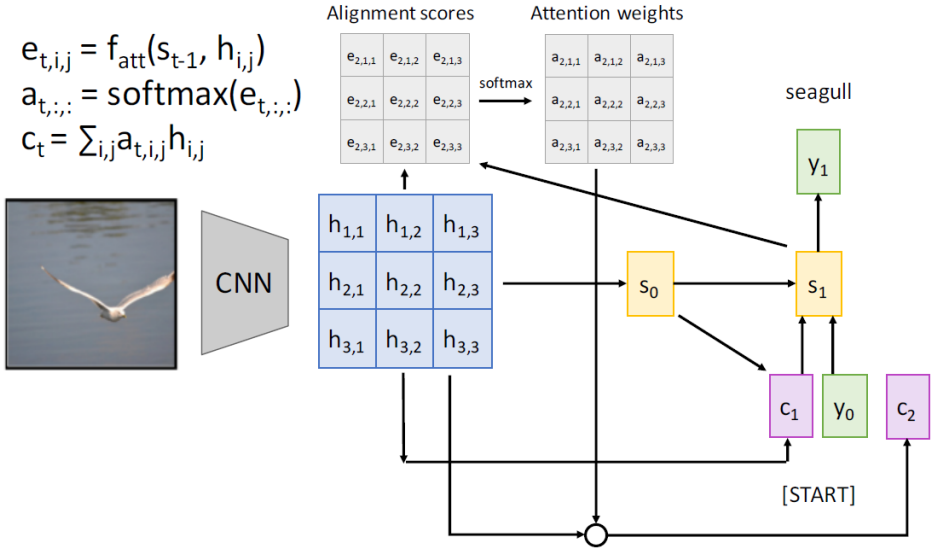

**Computation**:
**Similarities**: e (Shape: $N_X$) $e_i = q \cdot X_i$
**Attention weights**: a = softmax(e) (Shape: $N_X$)
**Output vector**: $y = \sum_i a_i X_i$ (Shape: $D_X$)




Changes:
- Use dot product for similarity

# Attention Layer

**Inputs**:
**Query vector**: **q** (Shape: $D_Q$)
**Input vectors**: **X** (Shape: $N_X$ x $D_Q$)
**Similarity function**: scaled dot product




**Computation**:
**Similarities**: e (Shape: $N_X$) $e_i = \mathbf{q} \cdot \mathbf{X}_i / \text{sqrt}(D_Q)$
**Attention weights**: a = softmax(e) (Shape: $N_X$)
**Output vector**: $y = \sum_i a_i \mathbf{X}_i$ (Shape: $D_X$)

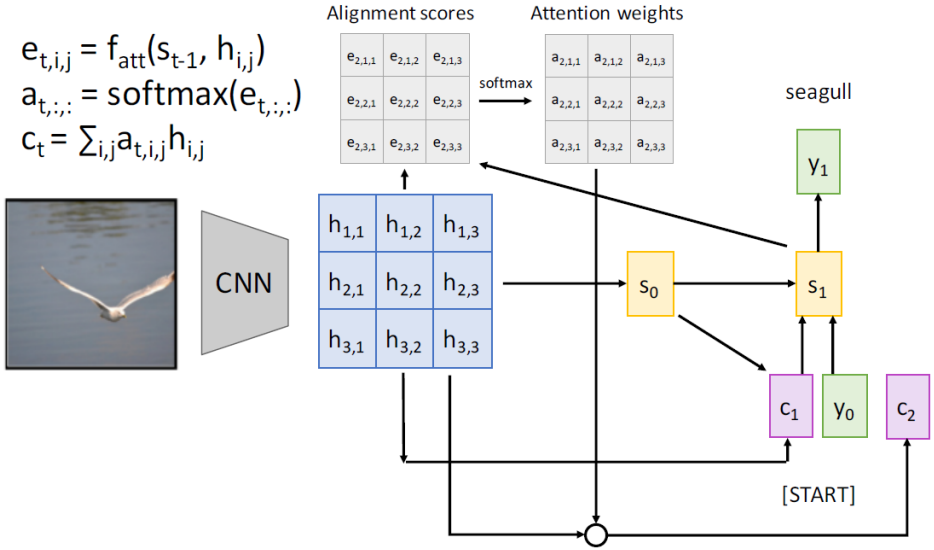
Changes:
- Use **scaled** dot product for similarity

# Attention Layer

**Inputs**:
**Query vector**: q (Shape: $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_Q$)
**Similarity function**: scaled dot product

Large similarities will cause softmax to saturate and give vanishing gradients
Recall $a \cdot b = |a||b| \cos(\text{angle})$
Suppose that a and b are constant vectors of dimension D
Then $|a| = (\sum_i a^2)^{1/2} = a\sqrt{D}$

**Computation**:
**Similarities**: e (Shape: $N_X$) $e_i = q \cdot X_i / \sqrt{D_Q}$
**Attention weights**: a = softmax(e) (Shape: $N_X$)
**Output vector**: $y = \sum_i a_i X_i$ (Shape: $D_X$)

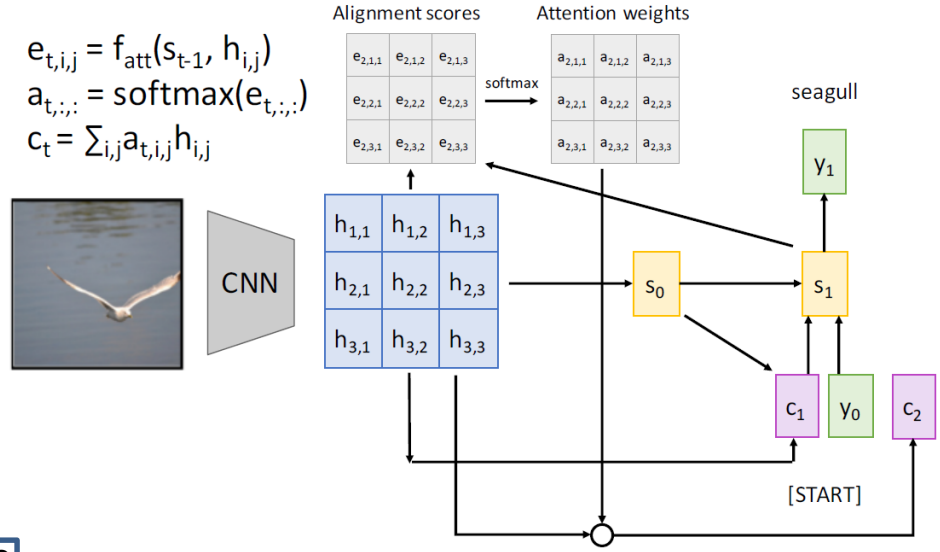


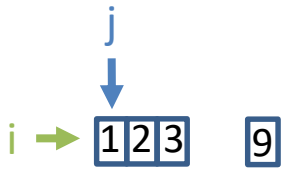

Changes:
- Use **scaled** dot product for similarity

# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_Q$)

$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$




**Computation**:
**Similarities**: $E = QX^T/\sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot X_j)/\sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: Y = AX (Shape: $N_Q$ x $D_X$) $Y_i = \sum_j A_{i,j} X_j$

Changes:
- Use dot product for similarity
- Multiple **query** vectors

# Attention Layer

**Inputs**:

**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)

**Input vectors**: X (Shape: $N_X$ x $D_X$)

**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)

**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)

**Computation**:

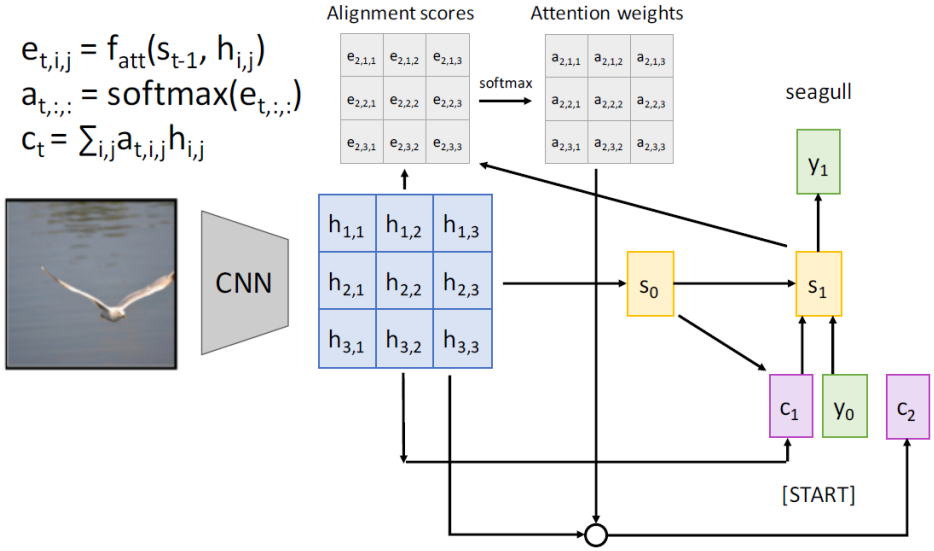
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)

**Output vectors**: $Y = AV$ (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



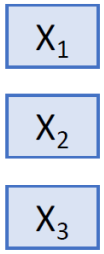

Changes:

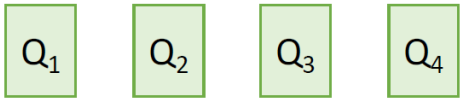
- Use dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**

# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

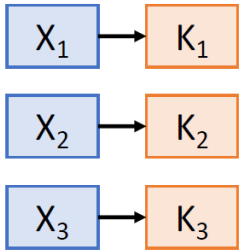


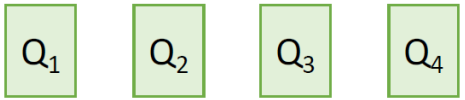



# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

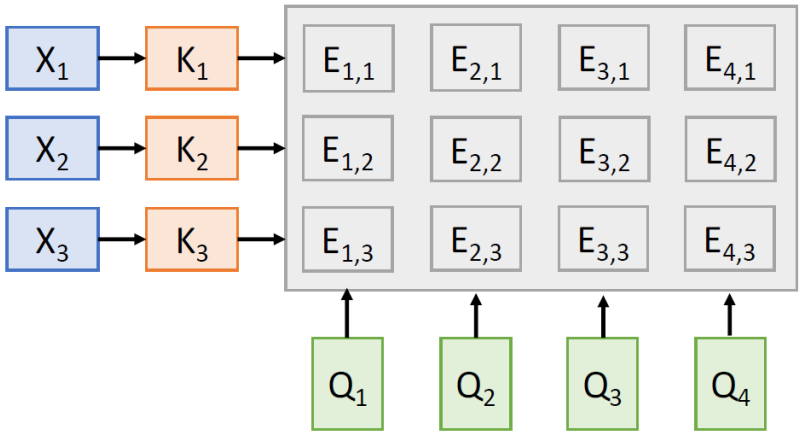


# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

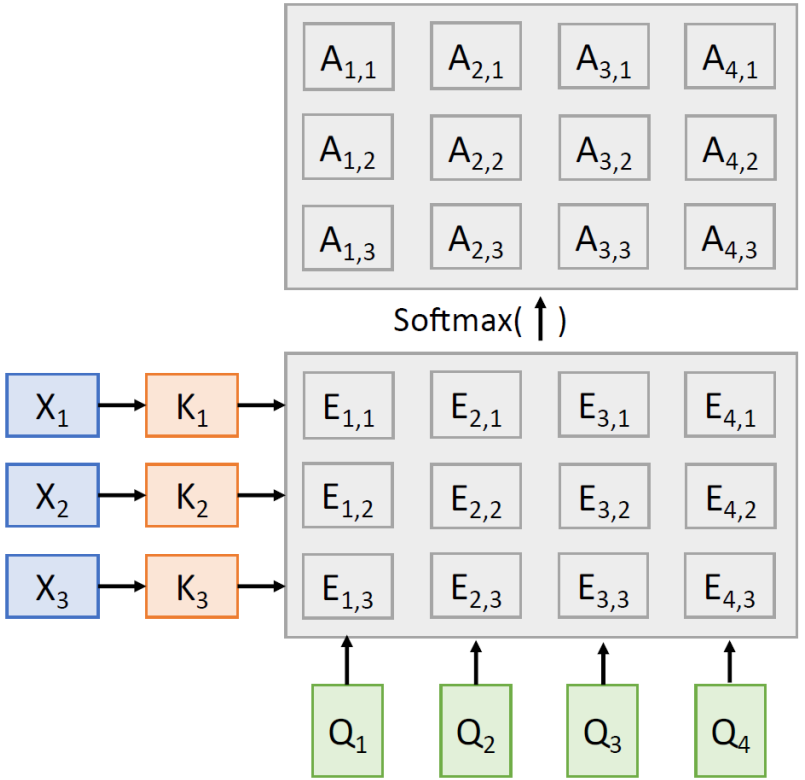
**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

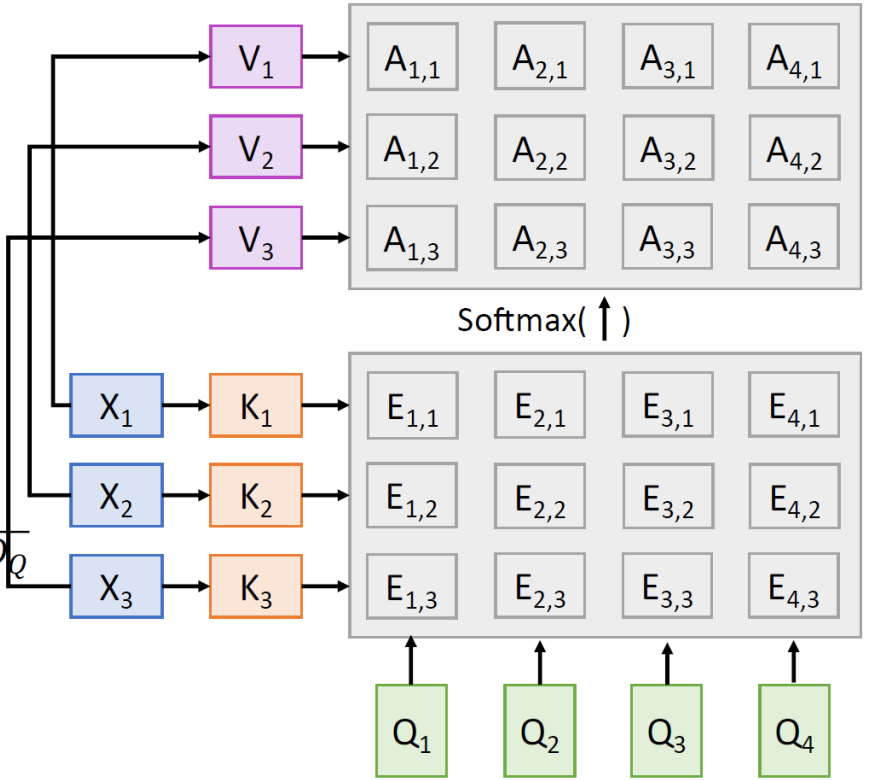
**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

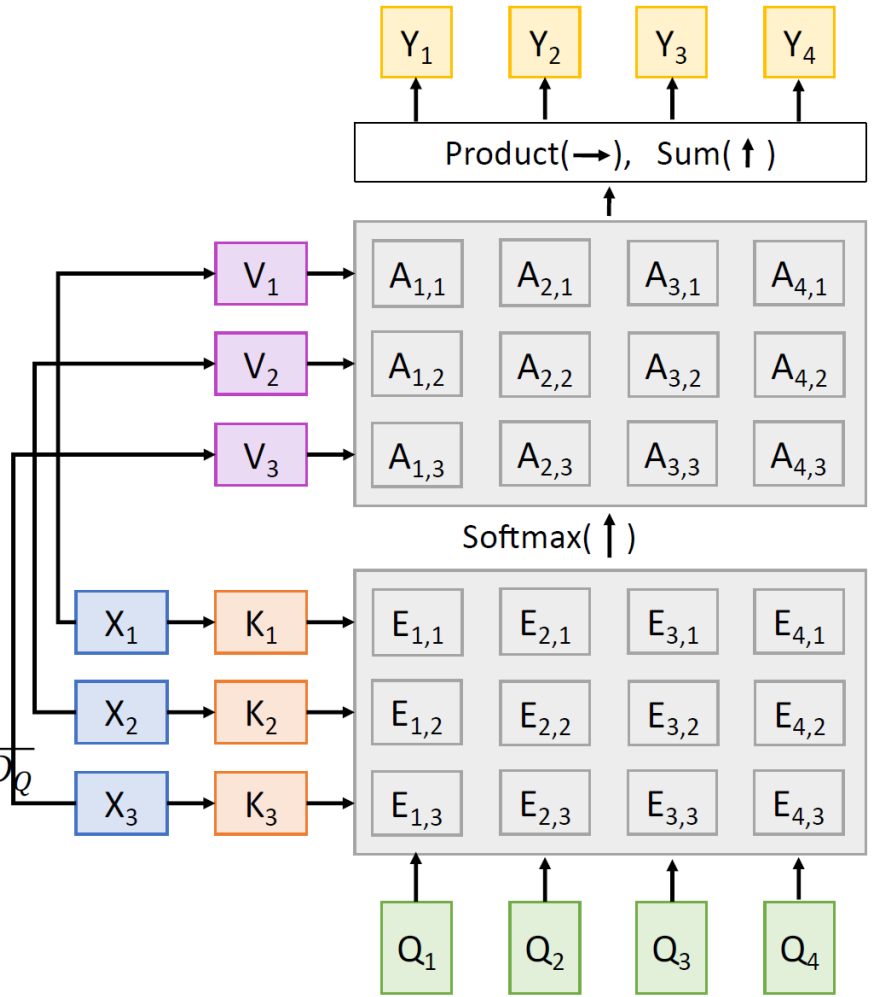
**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Attention Layer

**Inputs**:
**Query vectors**: Q (Shape: $N_Q$ x $D_Q$)
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

**Computation**:
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Self-Attention Layer

One **query** per **input vector**

**Inputs**:

**Query vectors**: **Q** (Shape: $N_Q$ x $D_Q$)

**Input vectors**: **X** (Shape: $N_X$ x $D_X$)

**Key matrix**: $\mathbf{W_K}$ (Shape: $D_X$ x $D_Q$)

**Value matrix:** $\mathbf{W_V}$ (Shape: $D_X$ x $D_V$)

**Computation**:

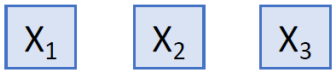
**Key vectors**: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: $N_X$ x $D_Q$)

**Value Vectors**: $\mathbf{V} = \mathbf{X}\mathbf{W_V}$ (Shape: $N_X$ x $D_V$)

**Similarities**: $E = \mathbf{Q}\mathbf{K}^T / \sqrt{D_Q}$ (Shape: $N_Q$ x $N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

**Attention weights**: A = softmax(E, dim=1) (Shape: $N_Q$ x $N_X$)

**Output vectors**: $Y = A\mathbf{V}$ (Shape: $N_Q$ x $D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

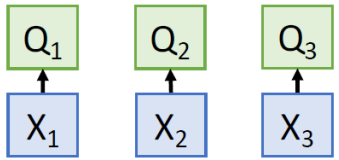


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

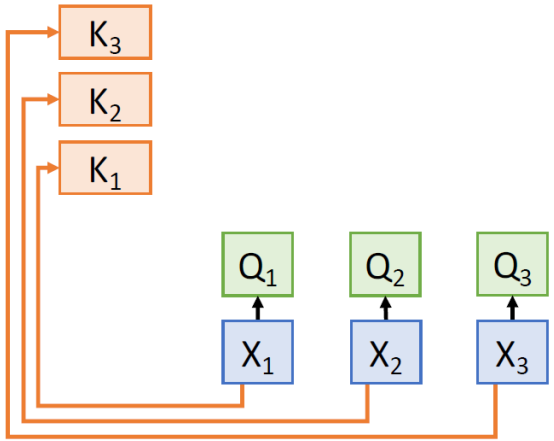


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

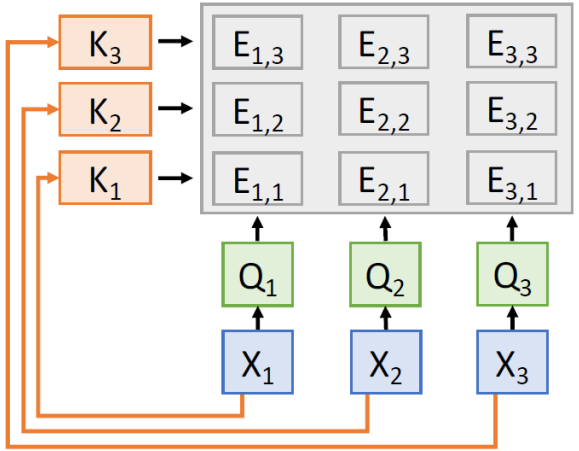


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

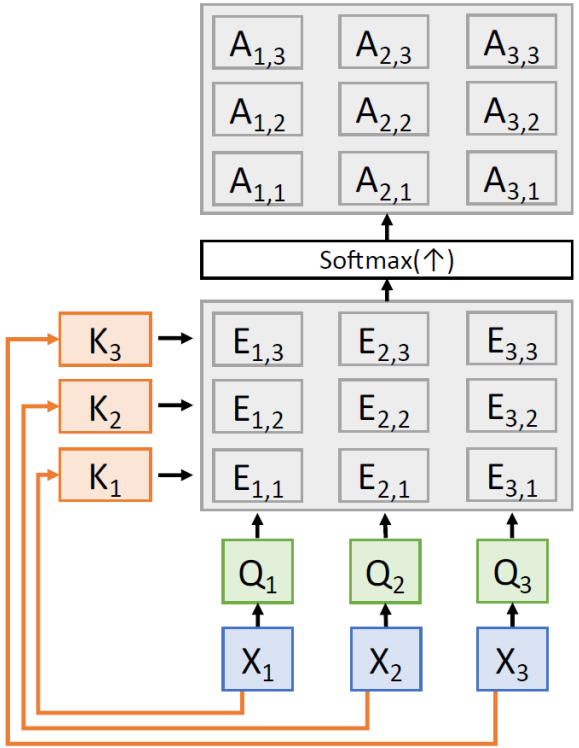


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

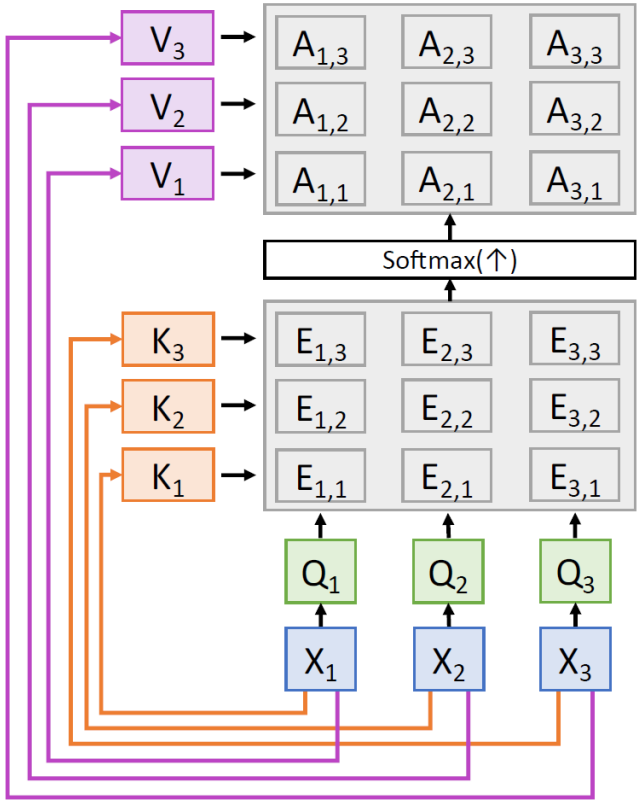


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

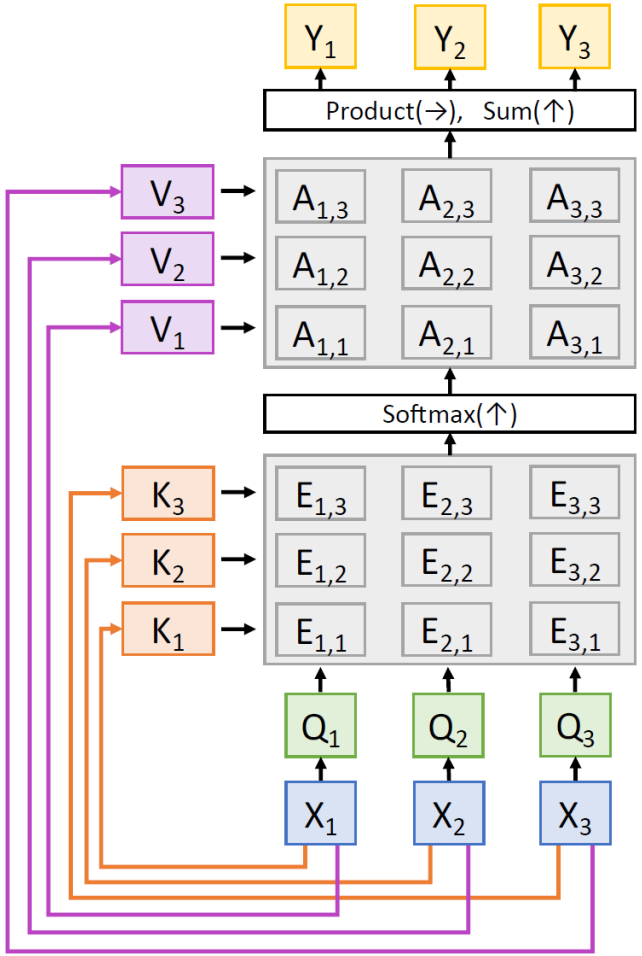


# Self-Attention Layer

One **query** per **input vector**

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

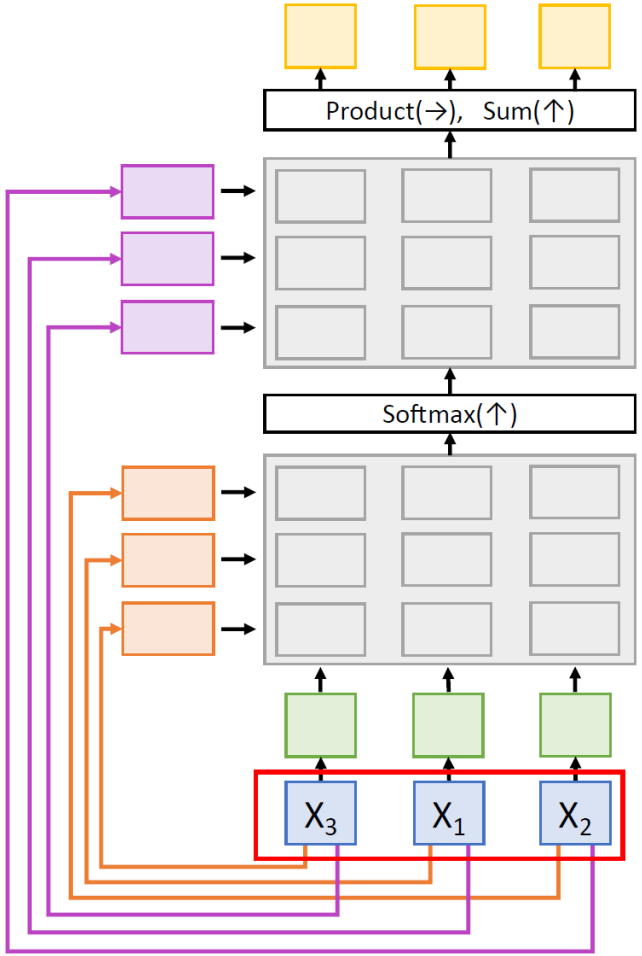


# Self-Attention Layer

Consider **permuting** the input vectors:

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



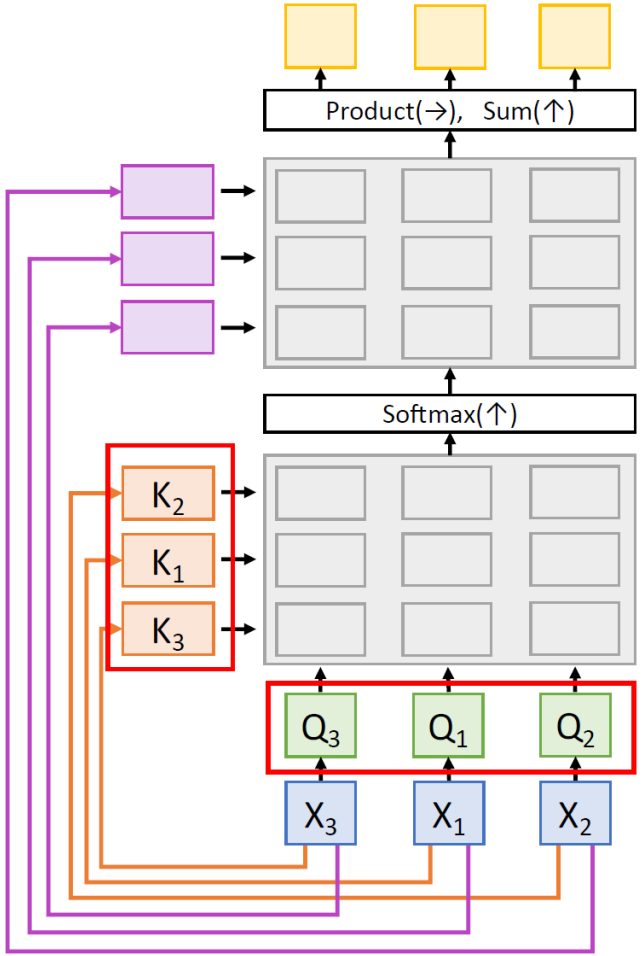
# Self-Attention Layer

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



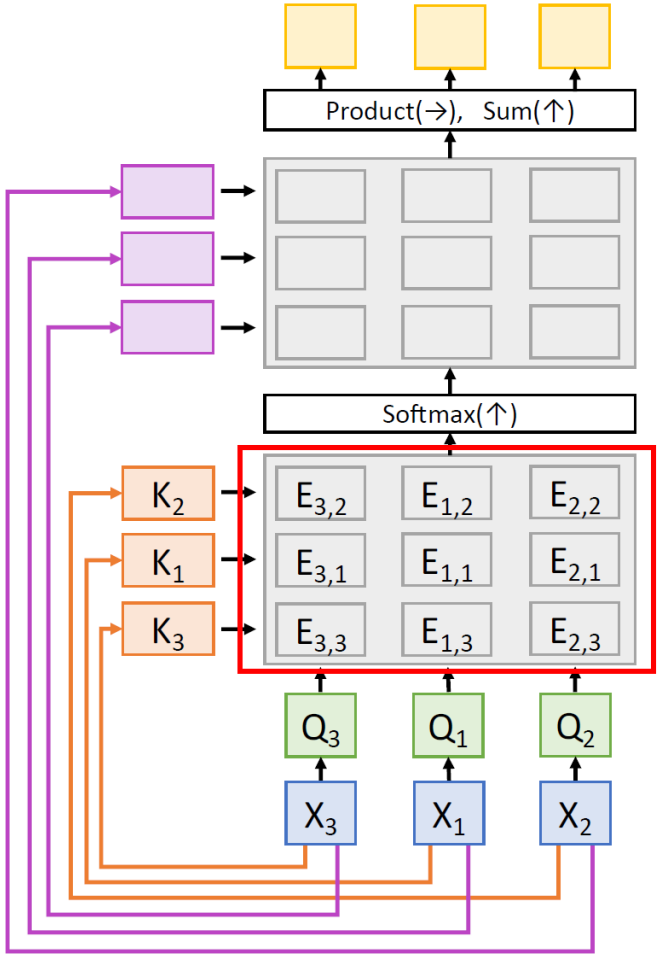
# Self-Attention Layer

Consider **permuting** the input vectors:

Similarities will be the same, but permuted

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



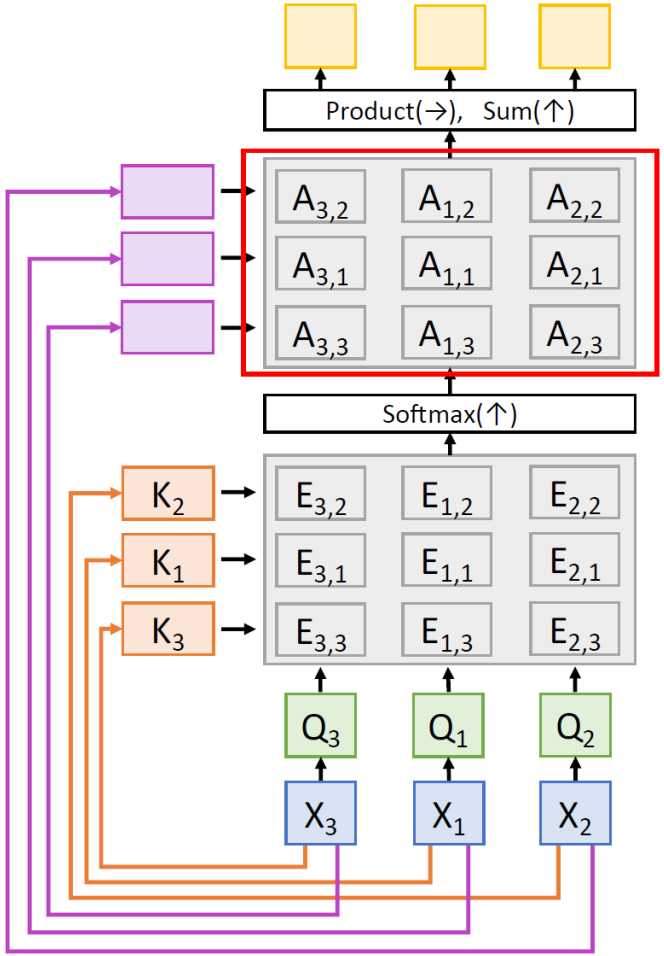
# Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



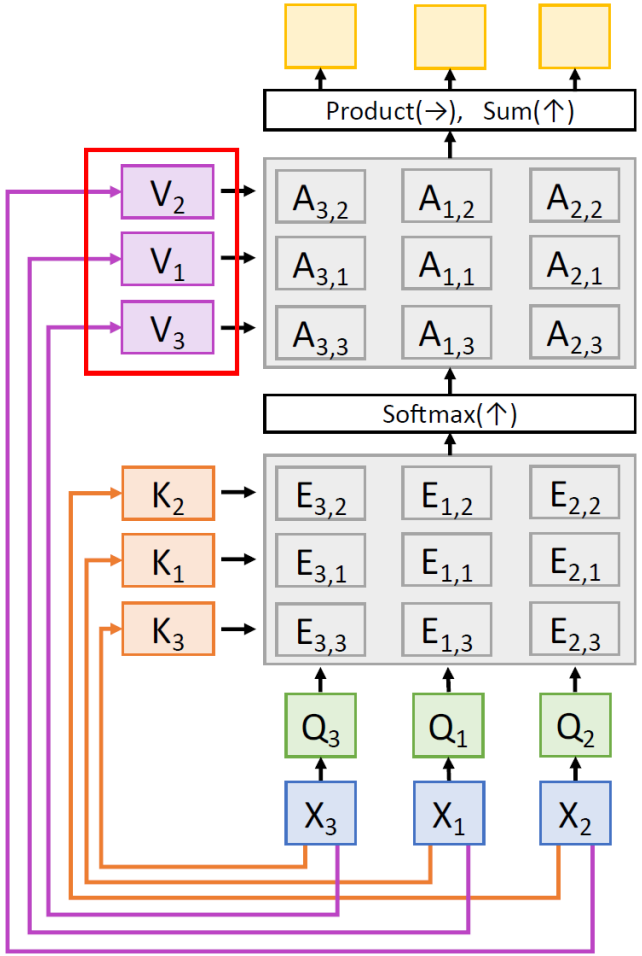
# Self-Attention Layer

Consider **permuting** the input vectors:

Values will be the same, but permuted

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



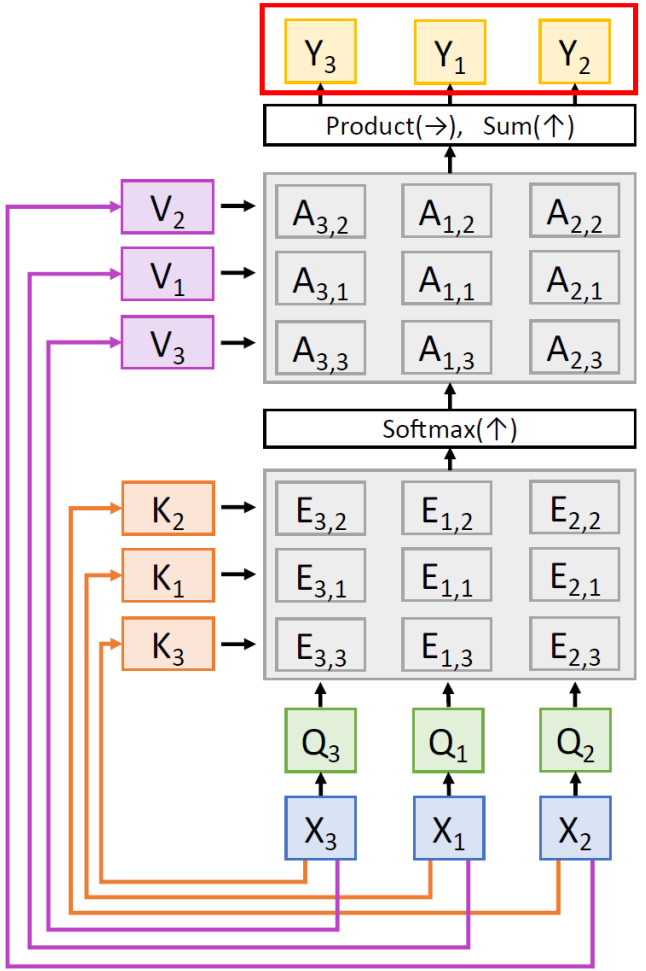
# Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Self-Attention Layer

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

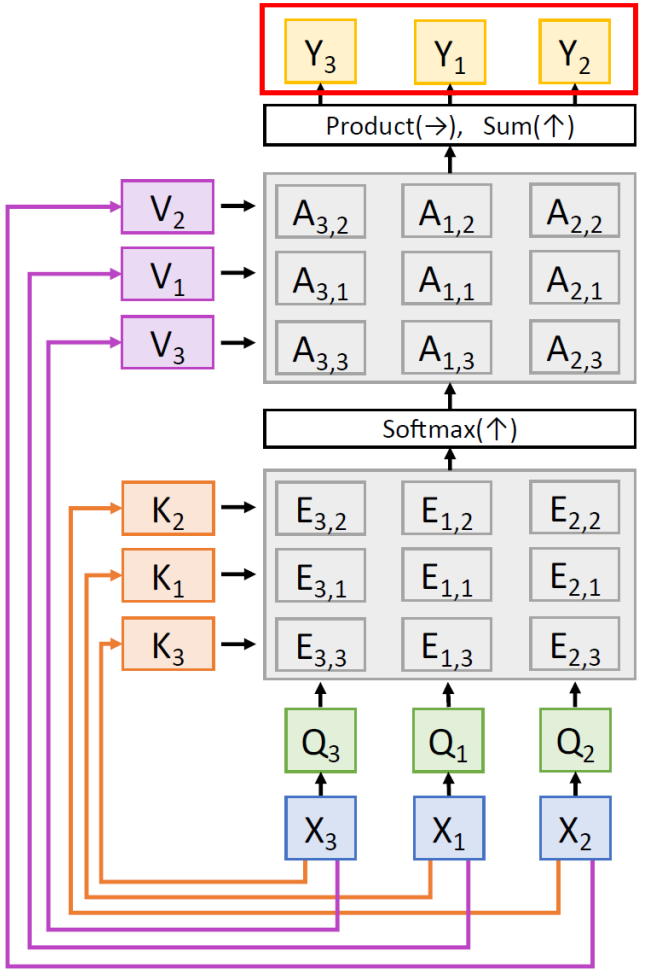
**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant**
f(s(x)) = s(f(x))

Self-Attention layer works on **sets** of vectors

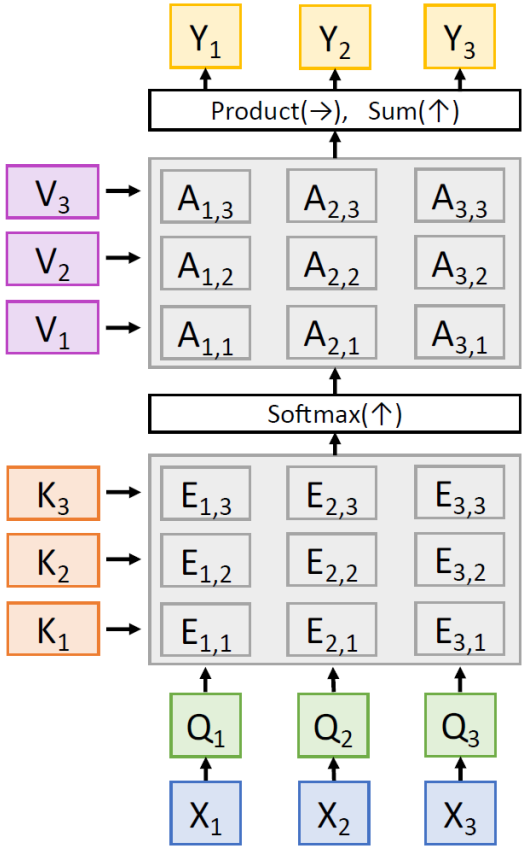


# Self-Attention Layer

Self attention doesn't "know" the order of the vectors it is processing!

**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Self-Attention Layer

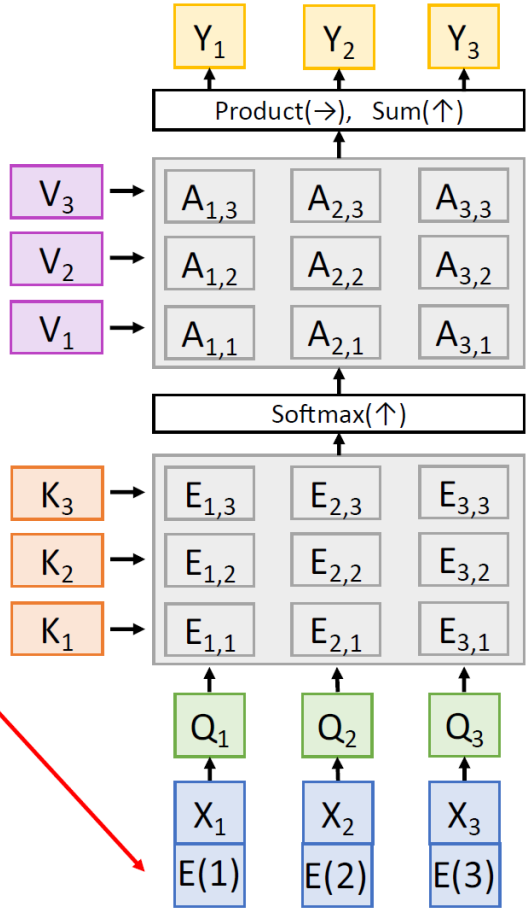
**Inputs**:
**Input vectors**: X (Shape: $N_X$ x $D_X$)
**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)
**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

Self attention doesn't "know" the order of the vectors it is processing!

In order to make processing position-aware, concatenate input with **positional encoding**

E can be learned lookup table, or fixed function



# Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence

**Inputs**:

**Input vectors**: X (Shape: $N_X$ x $D_X$)

**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)

**Value matrix**: $W_V$ (Shape: $D_X$ x $D_V$)

**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:

**Query vectors**: $Q = XW_Q$

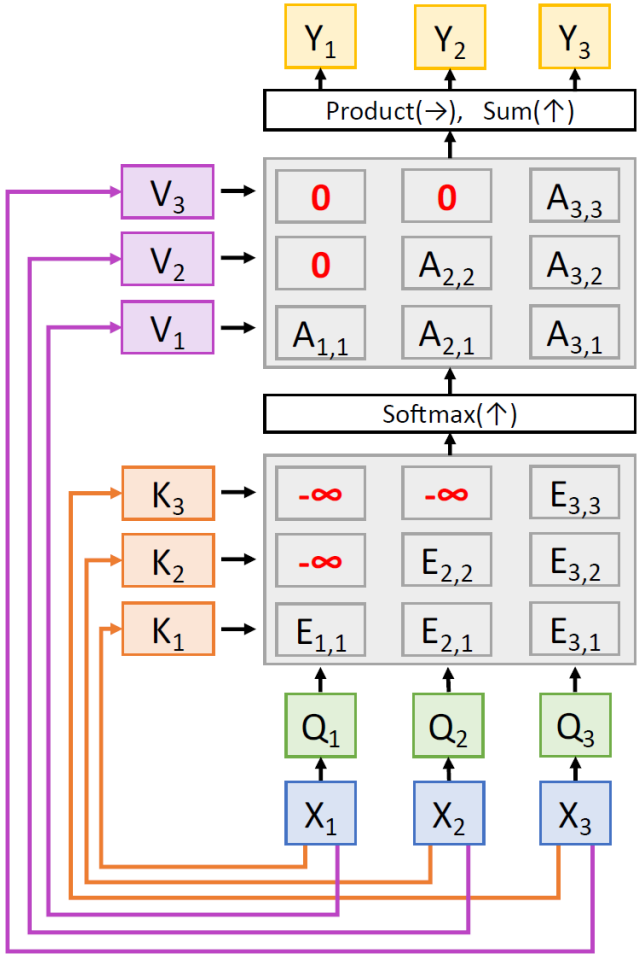
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)

**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence

Used for language modeling (predict next word)

**Inputs**:

**Input vectors**: X (Shape: $N_X$ x $D_X$)

**Key matrix**: $W_K$ (Shape: $D_X$ x $D_Q$)

**Value matrix:** $W_V$ (Shape: $D_X$ x $D_V$)

**Query matrix**: $W_Q$ (Shape: $D_X$ x $D_Q$)

**Computation**:

**Query vectors**: $Q = XW_Q$

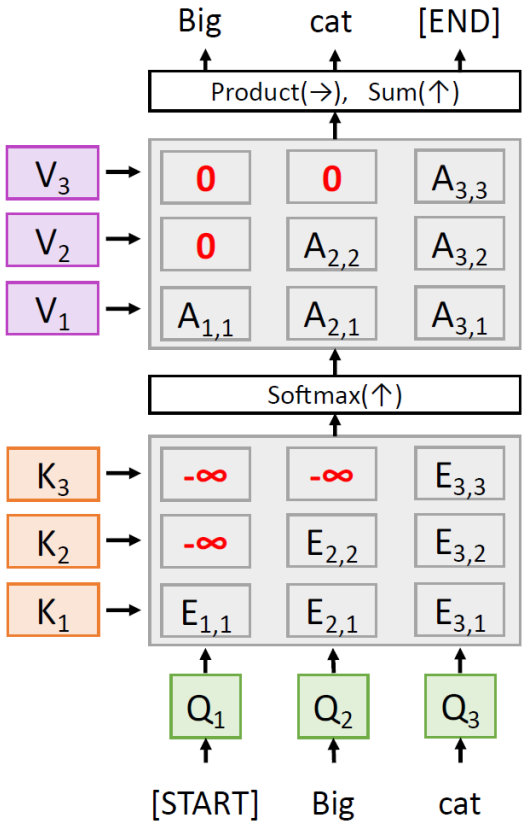
**Key vectors**: $K = XW_K$ (Shape: $N_X$ x $D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X$ x $D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X$ x $N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: A = softmax(E, dim=1) (Shape: $N_X$ x $N_X$)

**Output vectors**: Y = AV (Shape: $N_X$ x $D_V$) $Y_i = \sum_j A_{i,j} V_j$

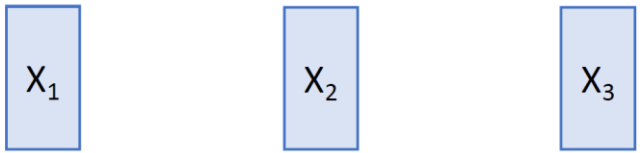


# Multihead Self-Attention

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Multihead Self-Attention

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Split

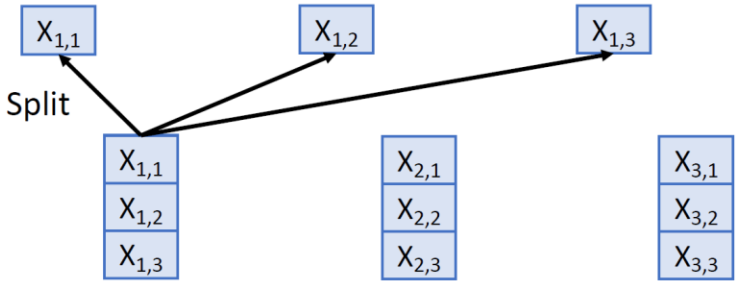
| | | |
|---|---|---|
| $X_{1,1}$ | $X_{2,1}$ | $X_{3,1}$ |
| $X_{1,2}$ | $X_{2,2}$ | $X_{3,2}$ |
| $X_{1,3}$ | $X_{2,3}$ | $X_{3,3}$ |

# Multihead Self-Attention

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

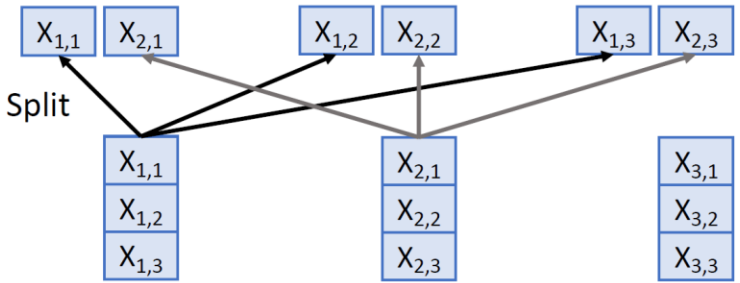


# Multihead Self-Attention

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

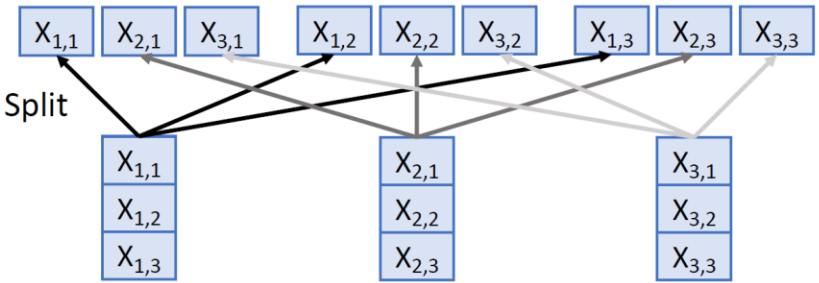


# Multihead Self-Attention

**Inputs**:
**Input vectors**: $X$ (Shape: $N_X \times D_X$)
**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:
**Query vectors**: $Q = XW_Q$
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)
**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)
**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Multihead Self-Attention

Run self-attention in parallel on each set of input vectors (different weights per head)

**Inputs**:

**Input vectors**: $X$ (Shape: $N_X \times D_X$)

**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)

**Value matrix**: $W_V$ (Shape: $D_X \times D_V$)

**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:

**Query vectors**: $Q = XW_Q$

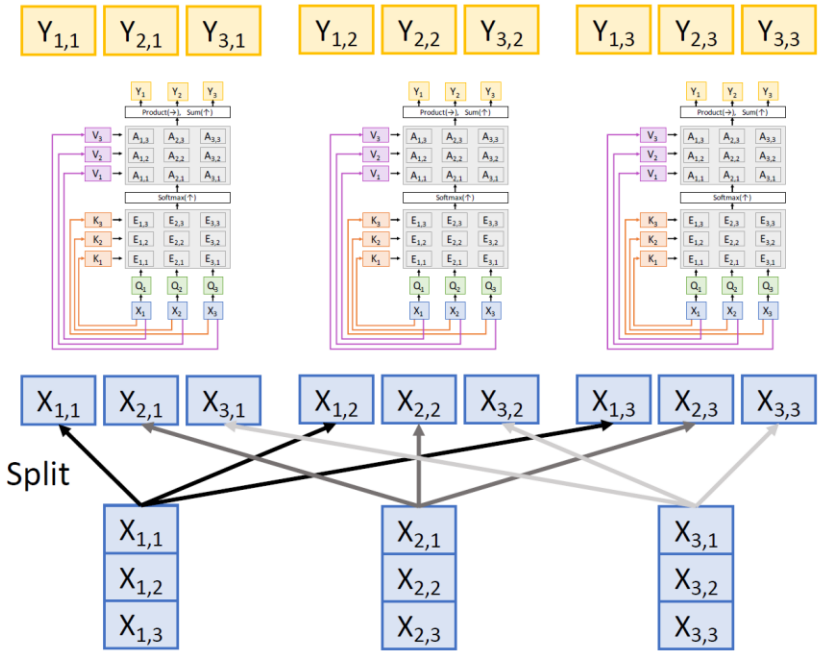
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)

**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Multihead Self-Attention

**Inputs**:

**Input vectors**: $X$ (Shape: $N_X \times D_X$)

**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)

**Value matrix**: $W_V$ (Shape: $D_X \times D_V$)

**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:

**Query vectors**: $Q = XW_Q$

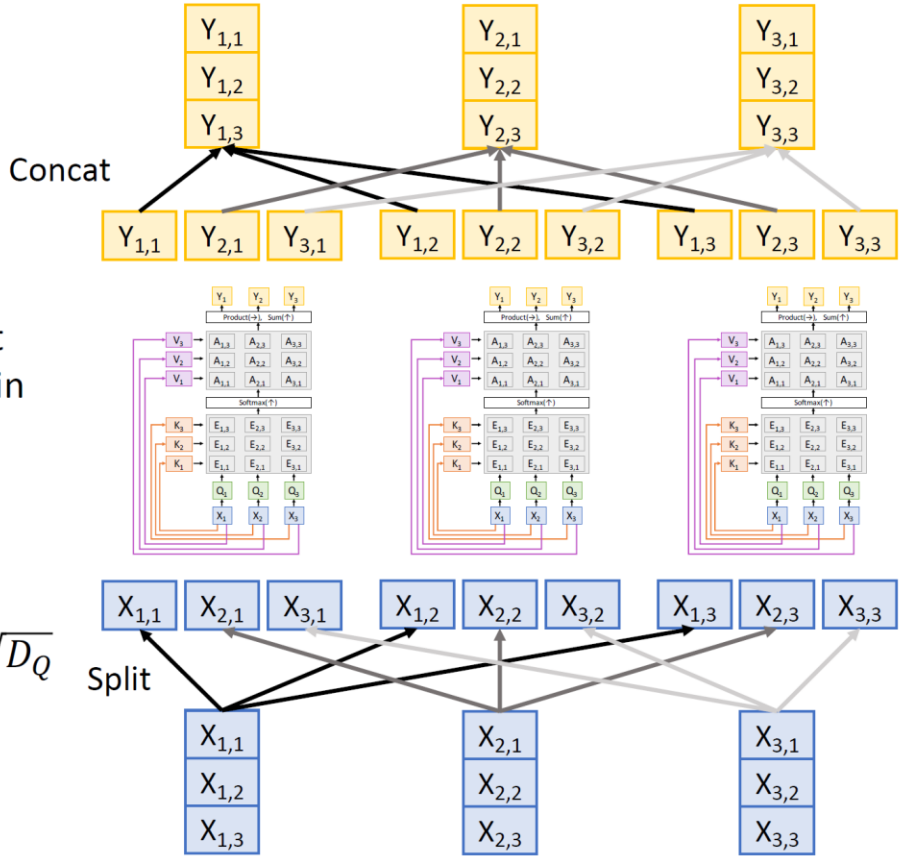
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)

**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Multihead Self-Attention

**Inputs**:

**Input vectors**: $X$ (Shape: $N_X \times D_X$)

**Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)

**Value matrix**: $W_V$ (Shape: $D_X \times D_V$)

**Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

Use H independent "Attention Heads" in parallel

**Computation**:

**Query vectors**: $Q = XW_Q$

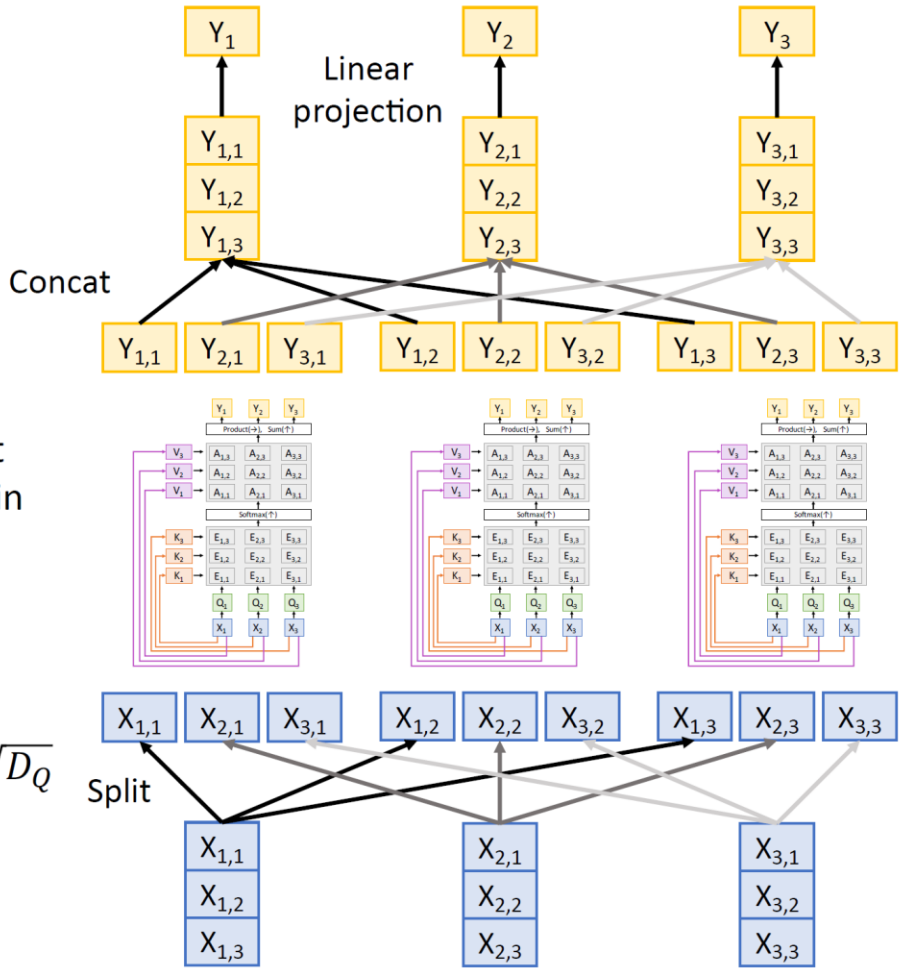
**Key vectors**: $K = XW_K$ (Shape: $N_X \times D_Q$)

**Value Vectors**: $V = XW_V$ (Shape: $N_X \times D_V$)

**Similarities**: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

**Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)

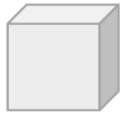
**Output vectors**: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



# Example: CNN with Self-Attention

Input Image

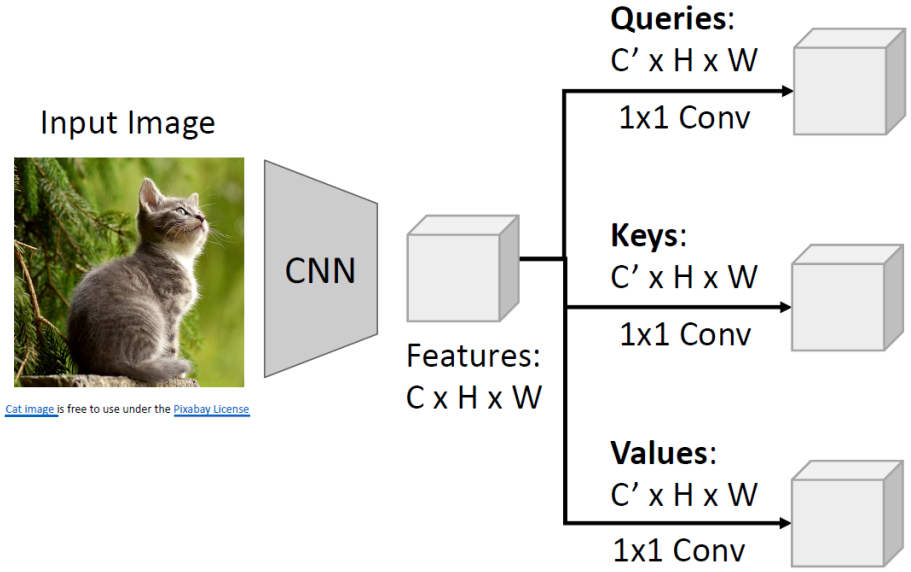


Features:
C x H x W

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

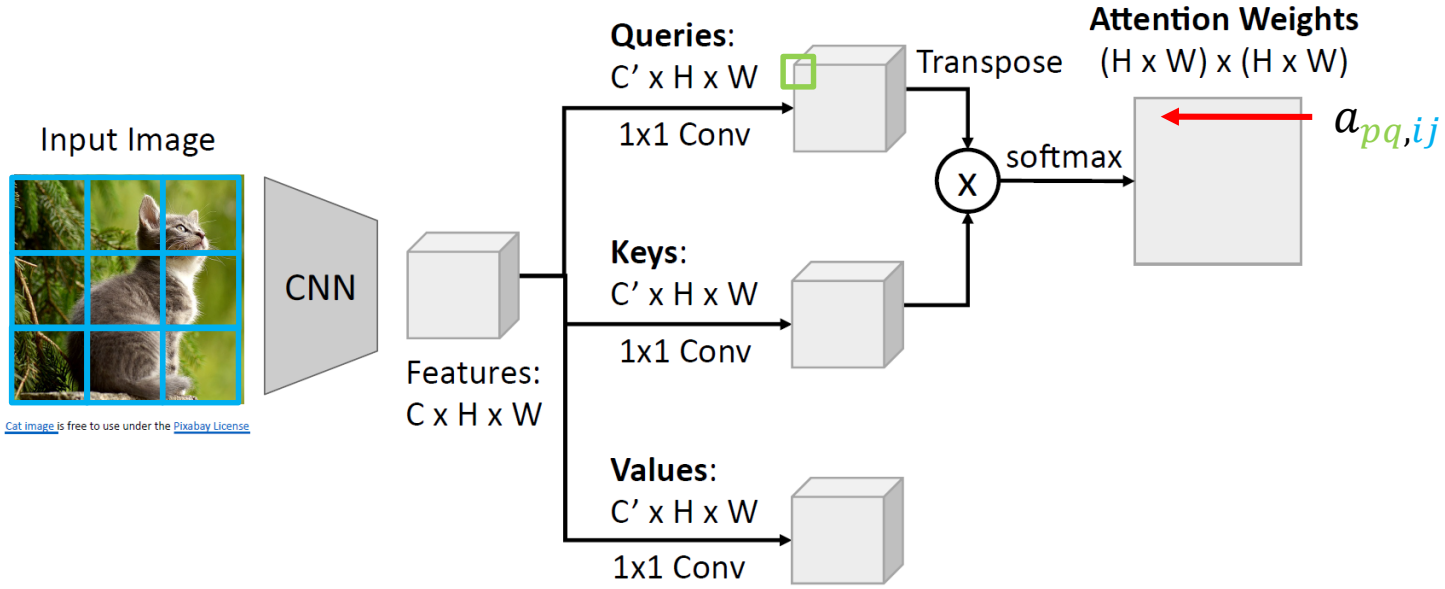
# Example: CNN with Self-Attention





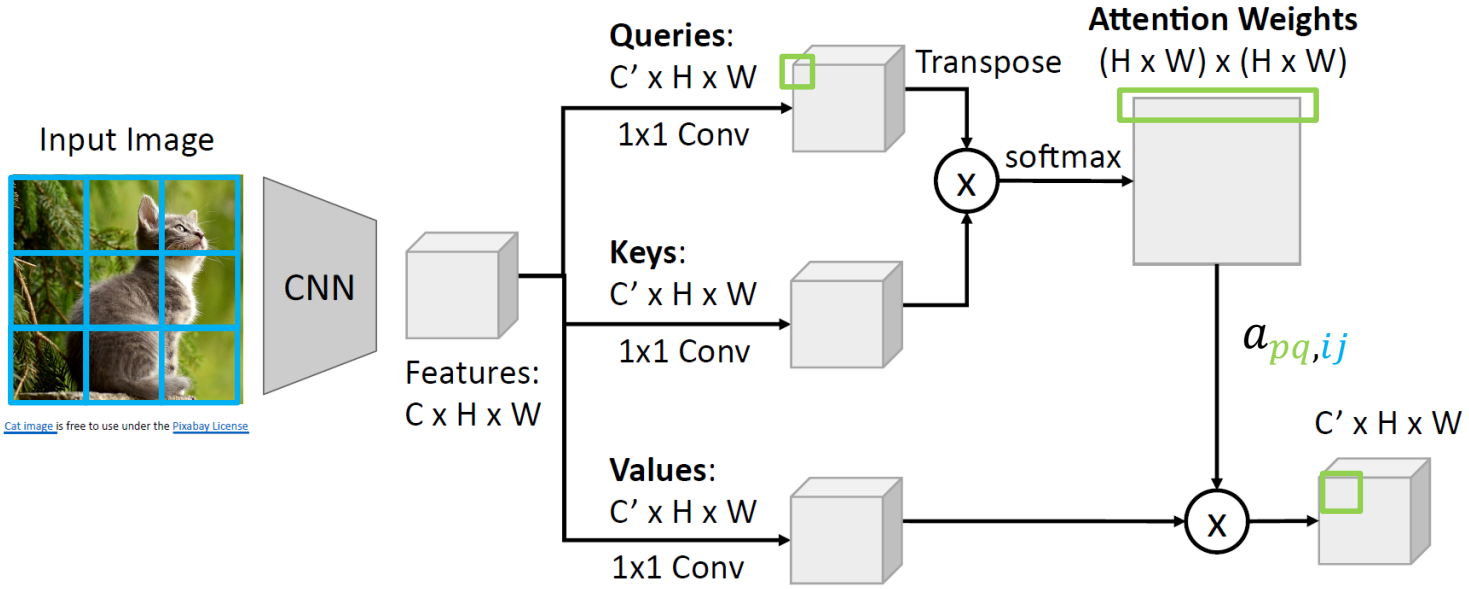

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

# Example: CNN with Self-Attention



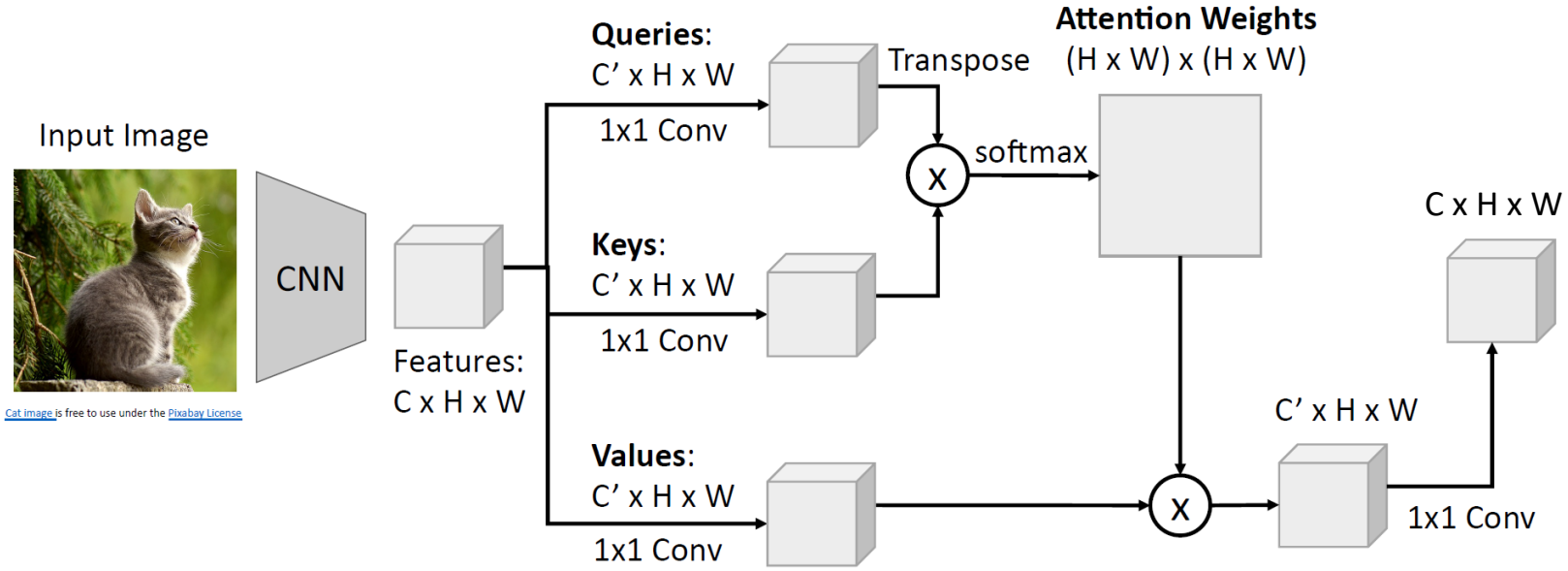

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

# Example: CNN with Self-Attention




Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

# Example: CNN with Self-Attention





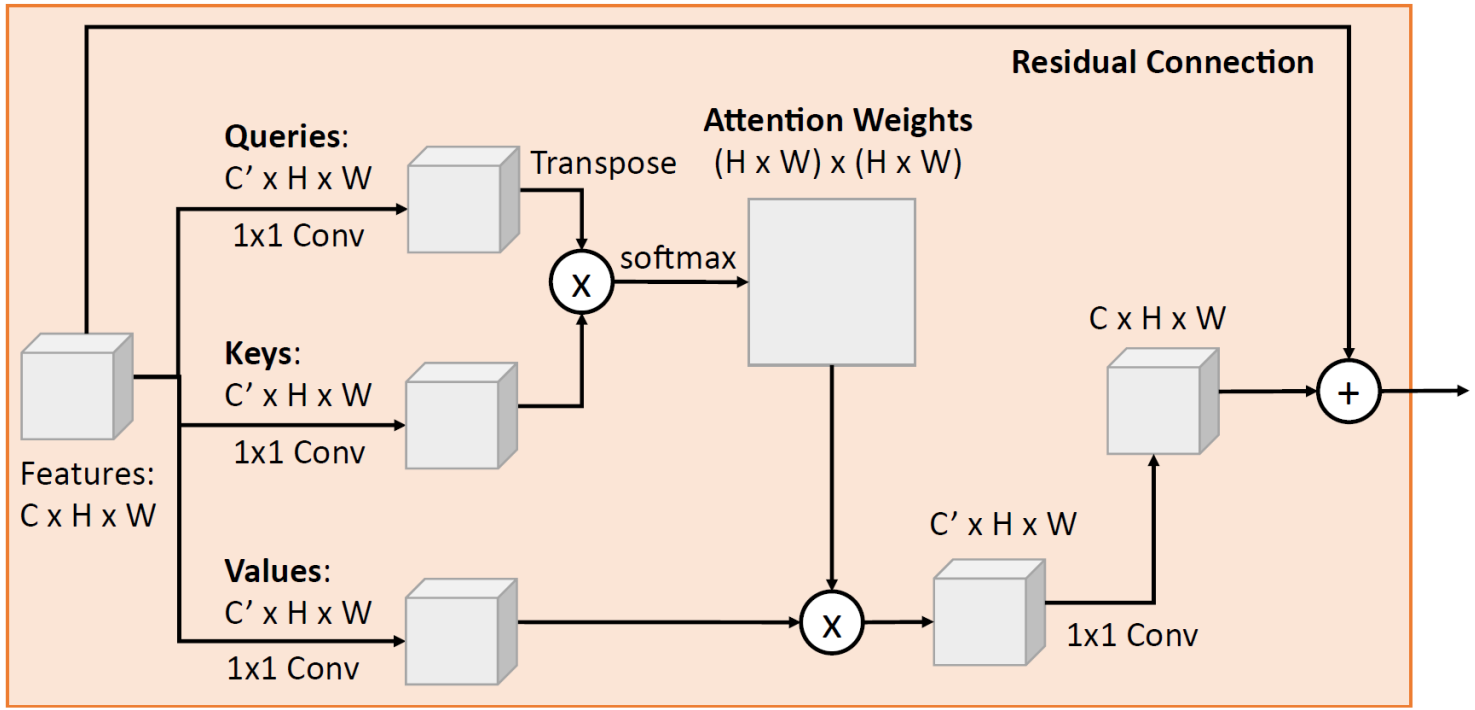

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

# Example: CNN with Self-Attention

Input Image



CNN

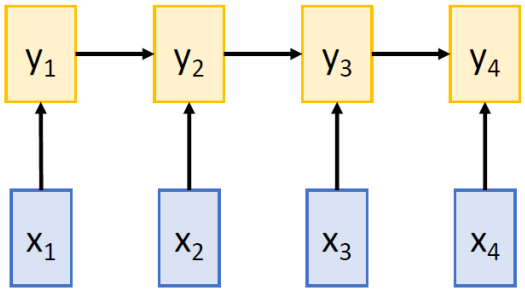



Self-Attention Module

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

# Three Ways of Processing Sequences

Recurrent Neural Network



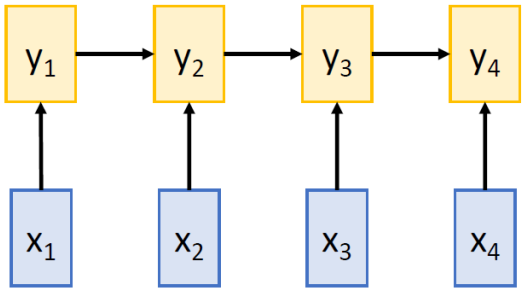

Works on **Ordered Sequences**

**(+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence**

**(-) Not parallelizable: need to compute hidden states sequentially**

# Three Ways of Processing Sequences

## Recurrent Neural Network

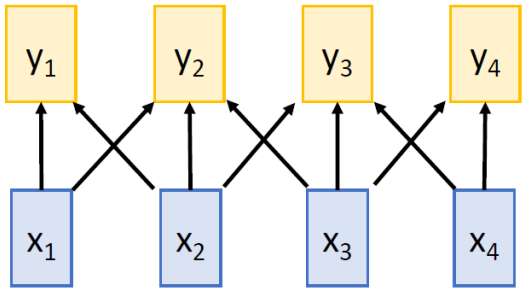



Works on **Ordered Sequences**

**(+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence**

**(-) Not parallelizable: need to compute hidden states sequentially**

## 1D Convolution



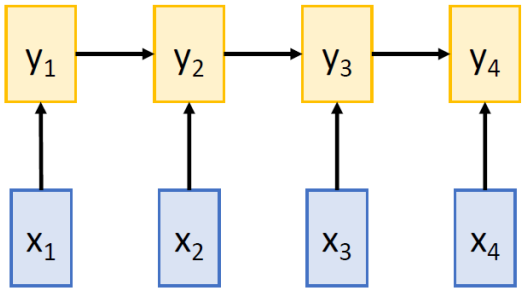

Works on **Multidimensional Grids**

**(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence**

**(+) Highly parallel: Each output can be computed in parallel**

# Three Ways of Processing Sequences

## Recurrent Neural Network

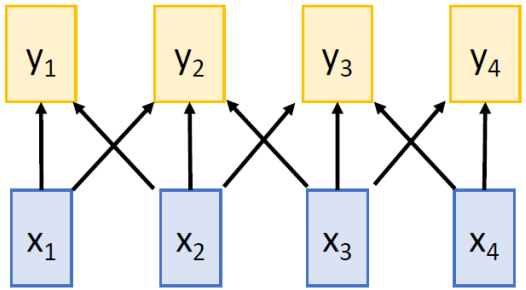



Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

## 1D Convolution

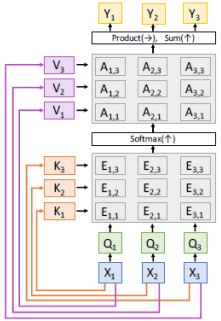



Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

## Self-Attention



Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

# Three Ways of Processing Sequences

Recurrent Neural Network | 1D Convolution | Self-Attention

## Attention is all you need

Vaswani et al, NeurIPS 2017

Works on **Ordered Sequences**

**(+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence**

**(-) Not parallelizable: need to compute hidden states sequentially**

Works on **Multidimensional Grids**

**(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence**

**(+) Highly parallel: Each output can be computed in parallel**

Works on **Sets of Vectors**

**(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!**

**(+) Highly parallel: Each output can be computed in parallel**

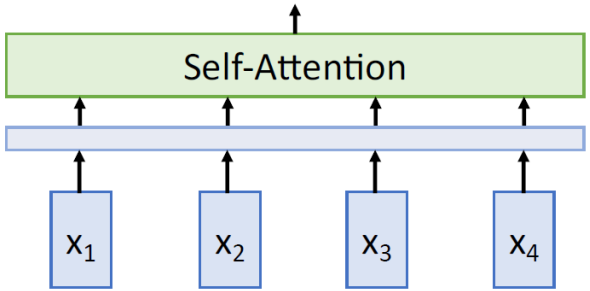
**(-) Very memory intensive**

# The Transformer


Vaswani et al, “Attention is all you need”, NeurIPS 2017

# The Transformer

All vectors interact with each other

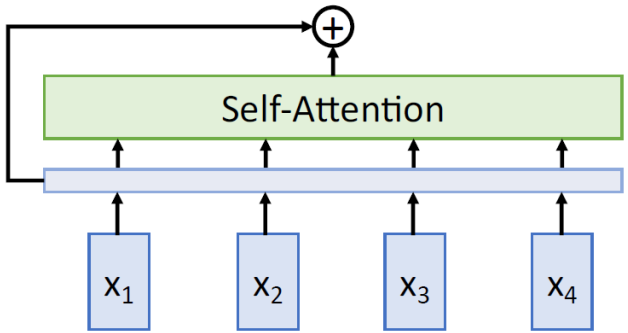



Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

Residual connection

All vectors interact with each other




Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: $\gamma$ (Shape: D)

shift: $\beta$ (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

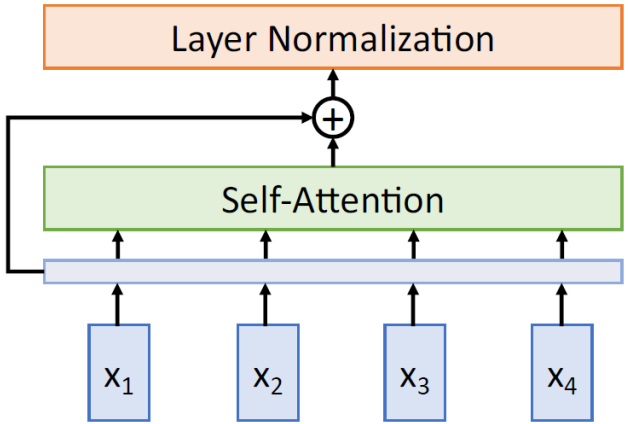
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016

Residual connection

All vectors interact with each other




Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: $\gamma$ (Shape: D)

shift: $\beta$ (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

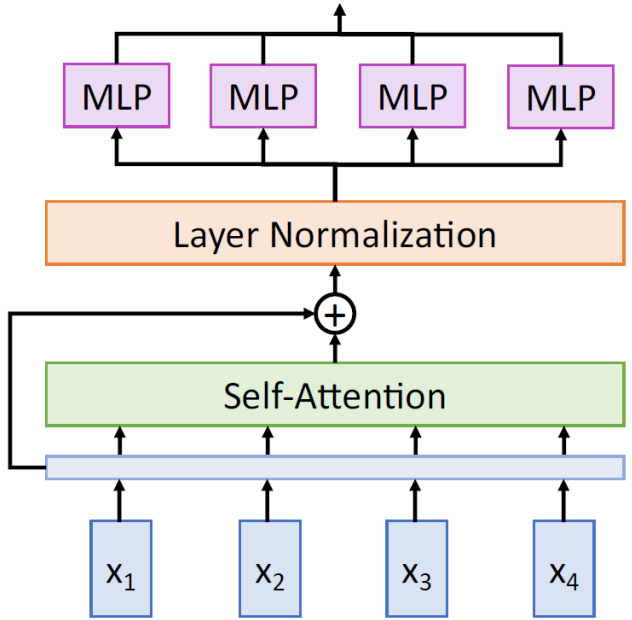
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently on each vector

Residual connection

All vectors interact with each other




Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: $\gamma$ (Shape: D)

shift: $\beta$ (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

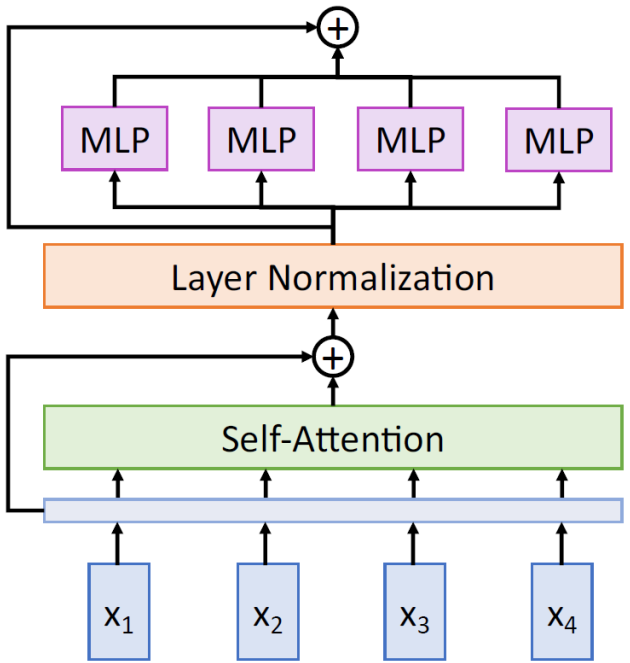
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other




Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: $\gamma$ (Shape: D)

shift: $\beta$ (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

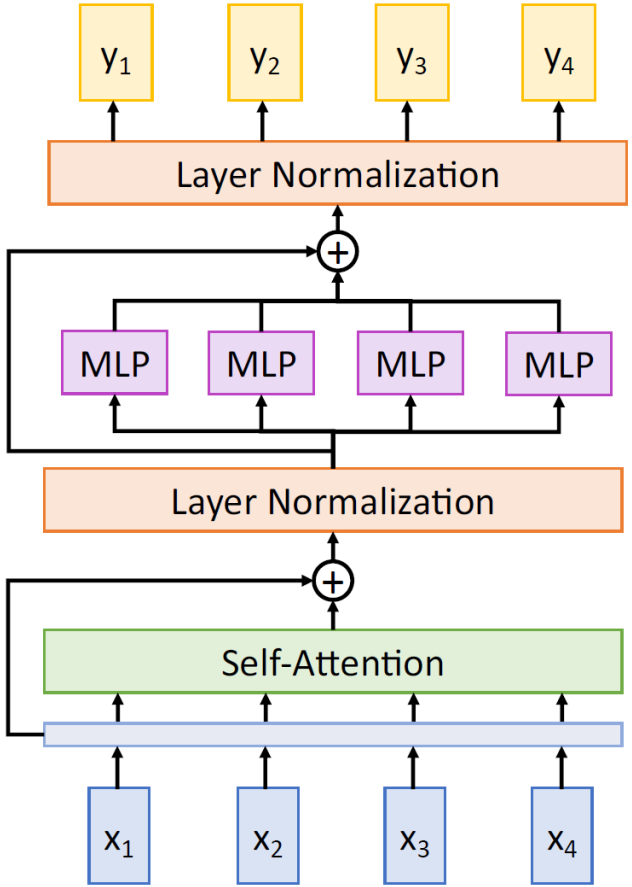
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other




Vaswani et al, "Attention is all you need", NeurIPS 2017

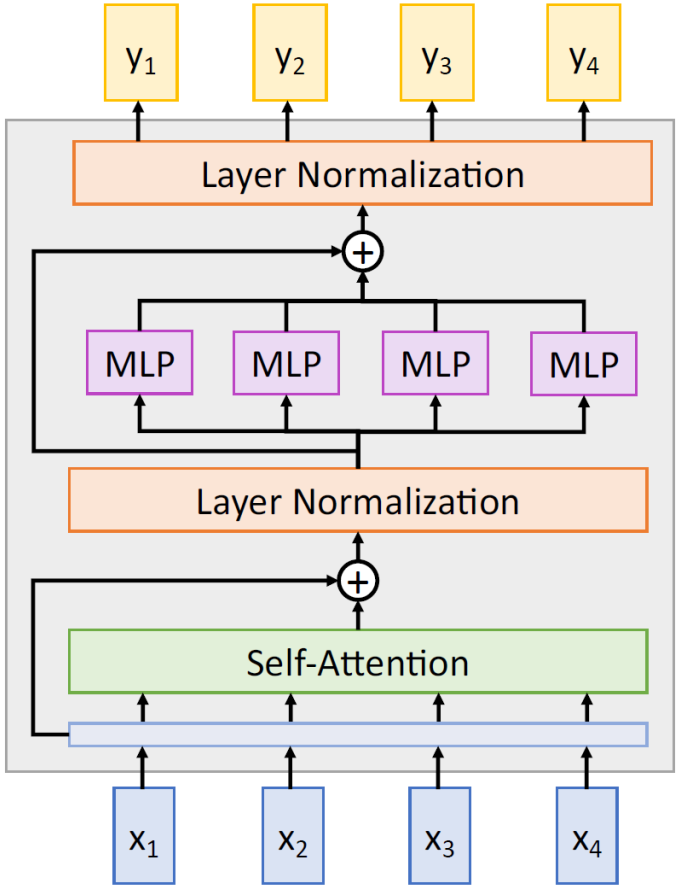
# The Transformer

**Transformer Block:**
**Input**: Set of vectors x
**Output**: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable




Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer

**Transformer Block:**
**Input**: Set of vectors x
**Output**: Set of vectors y

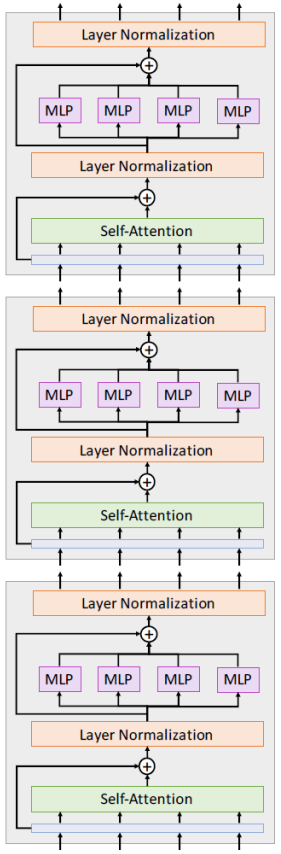
Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

A **Transformer** is a sequence of transformer blocks

Vaswani et al:
12 blocks, $D_Q$=512, 6 heads




Vaswani et al, "Attention is all you need", NeurIPS 2017

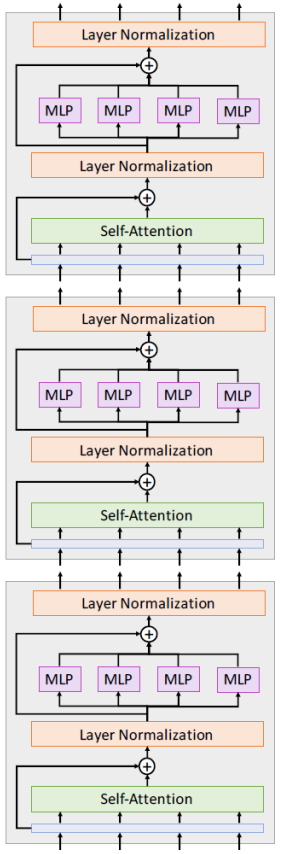
# The Transformer: Transfer Learning

"ImageNet Moment for Natural Language Processing"

**Pretraining**:
Download a lot of text from the internet

Train a giant Transformer model for language modeling

**Finetuning:**
Fine-tune the Transformer on your own NLP task




Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |

Vaswani et al, "Attention is all you need", NeurIPS 2017

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |

Yang et al, XLNet: Generalized Autoregressive Pretraining for Language Understanding", 2019
Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |

Radford et al, "Language models are unsupervised multitask learners", 2019

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |

Shoeybi et al, "Megatron-LM: Training Multi-Billion Parameter Languge Models using Model Parallelism", 2019

# Scaling up Transformers

~$430,000 on Amazon AWS!

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |

Shoeybi et al, "Megatron-LM: Training Multi-Billion Parameter Languge Models using Model Parallelism", 2019

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |
| Turing-NLG | 78 | 4256 | 28 | 17B | ? | 256x V100 GPU |

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |
| Turing-NLG | 78 | 4256 | 28 | 17B | ? | 256x V100 GPU |
| GPT-3 | 96 | 12288 | 96 | 175B | 694GB | ? |

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

# Scaling up Transformers

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |
| Turing-NLG | 78 | 4256 | 28 | 17B | ? | 256x V100 GPU |
| GPT-3 | 96 | 12,288 | 96 | 175B | 694GB | ? |
| Gopher | 80 | 16,384 | 128 | 280B | 10.55 TB | 4096x TPUv3 (38 days) |

Rae et al, “Scaling Language Models: Methods, Analysis, & Insights from Training Gopher”, arXiv 2021

# Scaling up Transformers

$3,768,320 on Google Cloud (eval price)

| Model | Layers | Width | Heads | Params | Data | Training |
|---|---|---|---|---|---|---|
| Transformer-Base | 12 | 512 | 8 | 65M | | 8x P100 (12 hours) |
| Transformer-Large | 12 | 1024 | 16 | 213M | | 8x P100 (3.5 days) |
| BERT-Base | 12 | 768 | 12 | 110M | 13 GB | |
| BERT-Large | 24 | 1024 | 16 | 340M | 13 GB | |
| XLNet-Large | 24 | 1024 | 16 | ~340M | 126 GB | 512x TPU-v3 (2.5 days) |
| RoBERTa | 24 | 1024 | 16 | 355M | 160 GB | 1024x V100 GPU (1 day) |
| GPT-2 | 48 | 1600 | ? | 1.5B | 40 GB | |
| Megatron-LM | 72 | 3072 | 32 | 8.3B | 174 GB | 512x V100 GPU (9 days) |
| Turing-NLG | 78 | 4256 | 28 | 17B | ? | 256x V100 GPU |
| GPT-3 | 96 | 12,288 | 96 | 175B | 694GB | ? |
| Gopher | 80 | 16,384 | 128 | 280B | 10.55 TB | 4096x TPUv3 (38 days) |

Rae et al, “Scaling Language Models: Methods, Analysis, & Insights from Training Gopher”, arXiv 2021

# Generated Text from GPT-3

**Bold text: Input prompt written by humans**
*Italics: Completion by GPT-3*

**Title: United Methodists Agree to Historic Split**
**Subtitle: Those who oppose gay marriage will form their own denomination**
**Article:**

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

# Generated Text from GPT-3

**Bold text: Input prompt written by humans**
*Italics: Completion by GPT-3*

**Title: United Methodists Agree to Historic Split**
**Subtitle: Those who oppose gay marriage will form their own denomination**
**Article:** *After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination. The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.*

# GPT-3: Programming by prompt

Input / Output examples
Test example
*Italics: Completion by GPT-3*

**Poor English input: I eated the purple berries.**
**Good English output: I ate the purple berries.**
**Poor English input: Thank you for picking me as your designer. I'd appreciate it.**
**Good English output: Thank you for choosing me as your designer. I appreciate it.**
**Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.**
**Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.**
**Poor English input: I'd be more than happy to work with you in another project.**
**Good English output:**

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

# GPT-3: Programming by prompt

**Input / Output examples**
**Test example**
*Italics: Completion by GPT-3*

**Poor English input: I eated the purple berries.**
**Good English output: I ate the purple berries.**
**Poor English input: Thank you for picking me as your designer. I'd appreciate it.**
**Good English output: Thank you for choosing me as your designer. I appreciate it.**
**Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.**
**Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.**
**Poor English input: I'd be more than happy to work with you in another project.**
**Good English output:** *I'd be more than happy to work with you on another project.*

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

https://www.163.com/dy/article/G7UTUG1D0511DSSR.html

# GPT-3

前文：泛起笑意，

GPT-3：

https://www.163.com/dy/article/G7UTUG1D0511DSSR.html

# GPT-3

**前文：泛起笑意，**

**GPT-3：他笑意中的那份溫柔，只想給她好好珍藏，即便這一切不會發生，至少在他的世界裡，她是無憂無慮的快樂的。就連他對她的關心，她都不在意了，還有什麼不放心的呢？“好，我答應你。不過，如果我們能有個孩子，我一定做個好爸爸。”**

https://edge.aif.tw/chinese-text-generation-using-gpt-2-and-an-overview-of-gpt-3/

# GPT-3

**前文：他正要離開**

**GPT-3：**

https://edge.aif.tw/chinese-text-generation-using-gpt-2-and-an-overview-of-gpt-3/

# GPT-3

**前文：他正要離開**

**GPT-3：，卻見他站到了自己身前，一下子拉住了她的手，用一種很奇怪的語調說道：“我的心情很壞，很生氣，“他看著她，一字一句的說道：“你能不能說的快一點，我不想傷害你。“看著她一臉的冷漠，他的心中微微一酸，但是很快又笑了，溫柔的笑了笑：“放心.....**

https://edge.aif.tw/chinese-text-generation-using-gpt-2-and-an-overview-of-gpt-3/

# GPT-3

**前文：阿sir，你看看他**

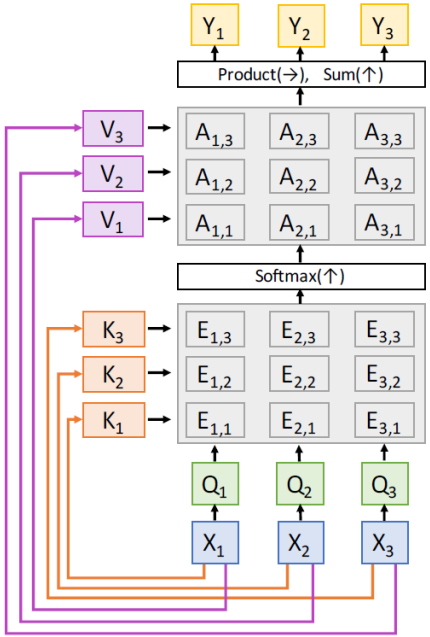
**GPT-3：（阿sir，你看看他）們幾時對阿爸、阿媽、阿婆笑過；你再看看他們幾時對你笑過。阿爸、阿媽、阿婆，你們笑笑就好了。其實，他們有笑，也有淚.....（此處省略200+字）阿sir，你是否真的愛你的父母？**

# Summary

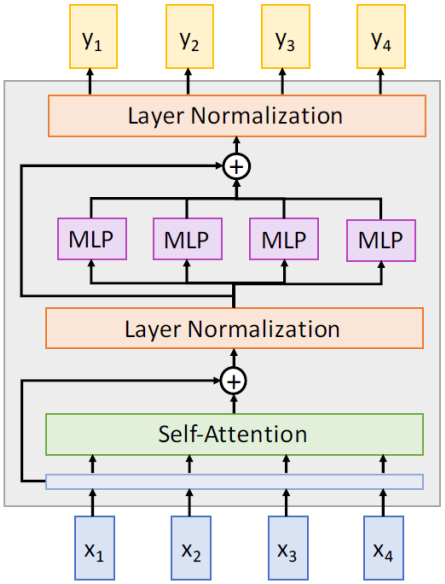
Adding **Attention** to RNN models lets them look at different parts of the input at each timestep

A dog is standing on a hardwood floor.

Generalized **Self-Attention** is new, powerful neural network primitive




**Transformers** are a new neural network model that only uses attention




Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015