Transformer

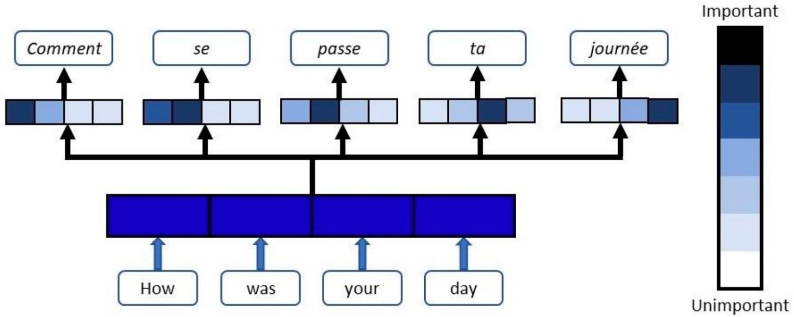
- With self attention as feature extractor

Outline

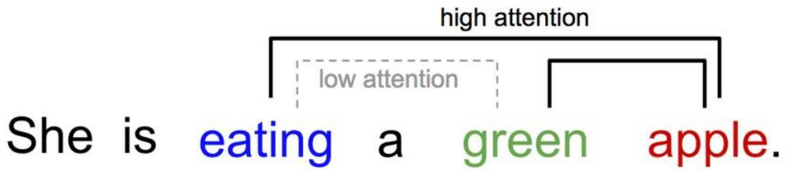
- Attention & Self Attention
- Transformers
- Classification using BERT(Demo Code)
- Self Attention(Demo Code)

Attention and Self Attention

- Attention: Between input and output elements



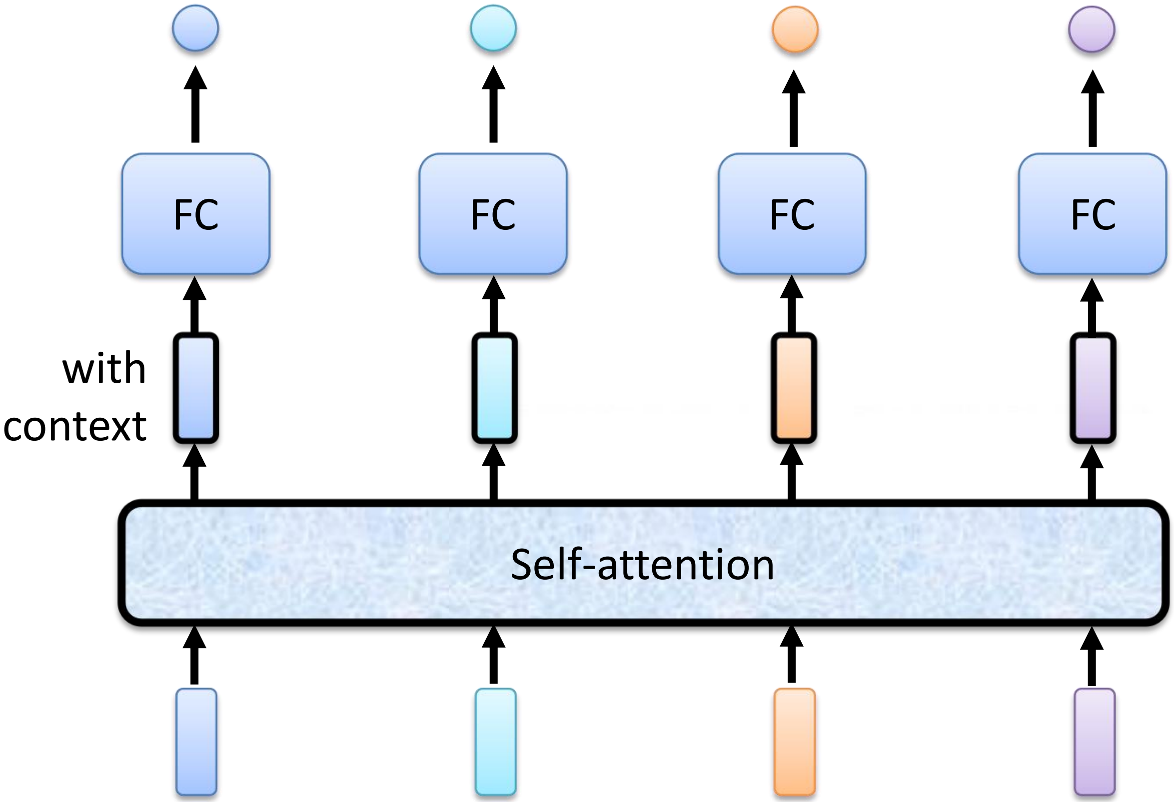
- Self Attention: Within input elements

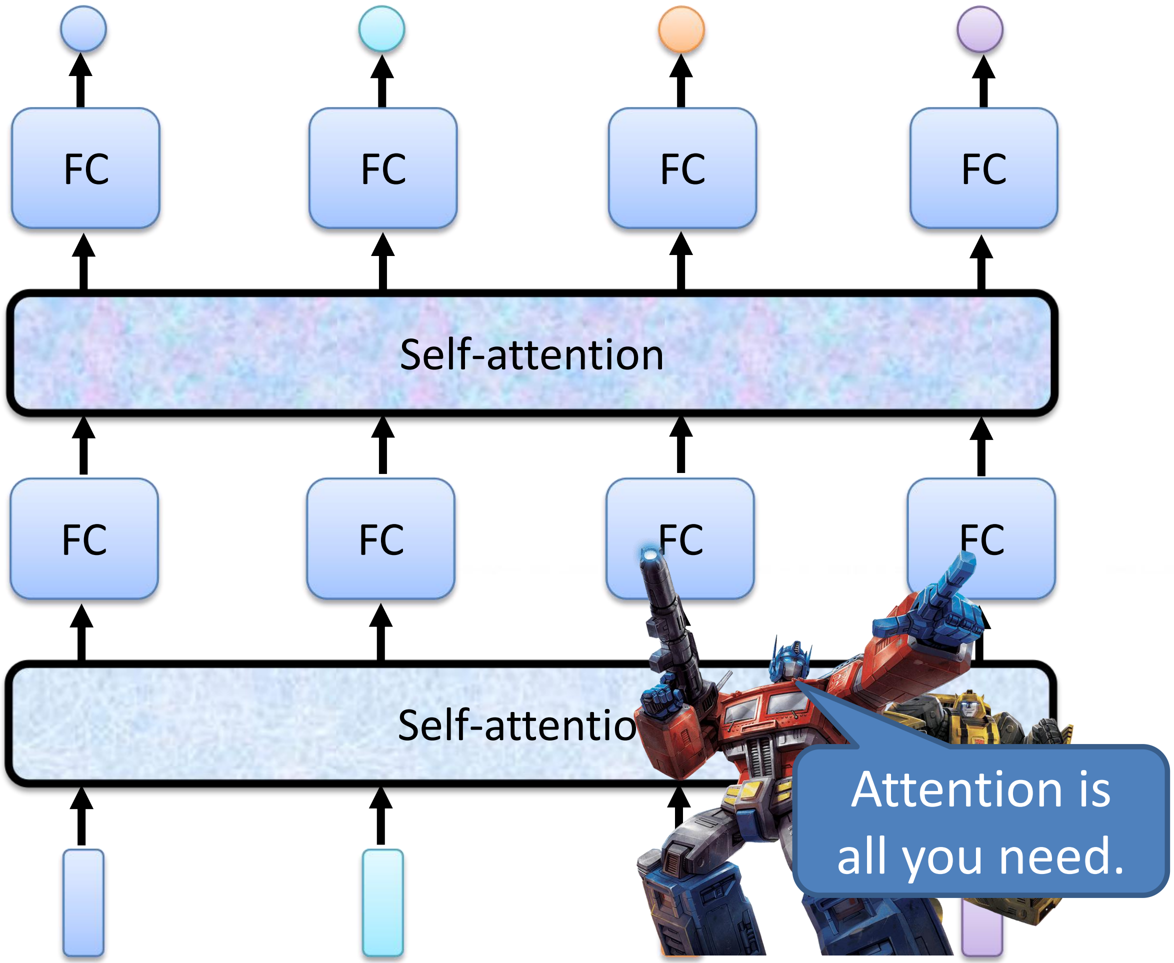


Self Attention

- Self attention is the relative degree of attendance each token should ensure to the fellow tokens of the sentence. It can be thought of as a table that enlists each token both on row and column and *(i, j)* th cell accounts for the relative degree of attendance *ith* row should ensure to the *jth* column.
- Self-attention is a new spin on the attention technique. Instead of looking at prior hidden vectors when considering a word embedding, self-attention is a weighted combination of all other word embeddings including those that appear later in the sentence.

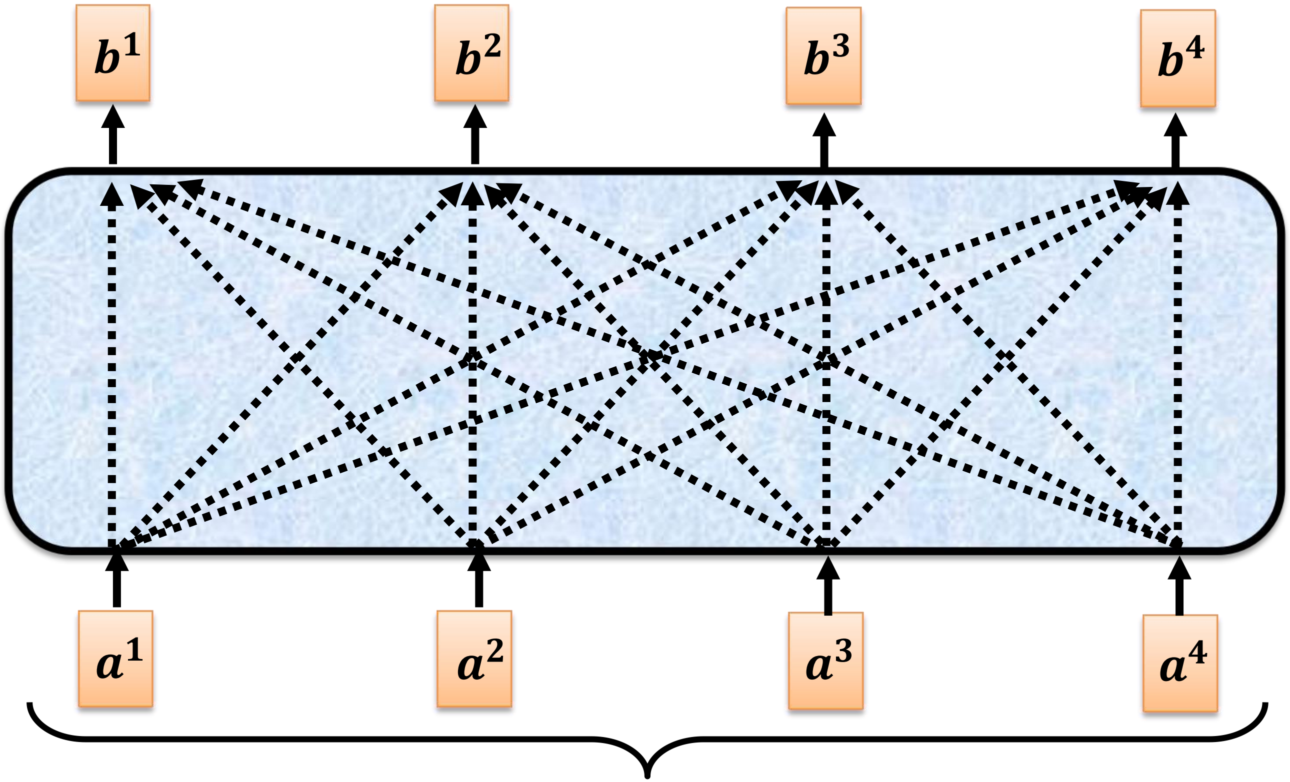
Self-attention





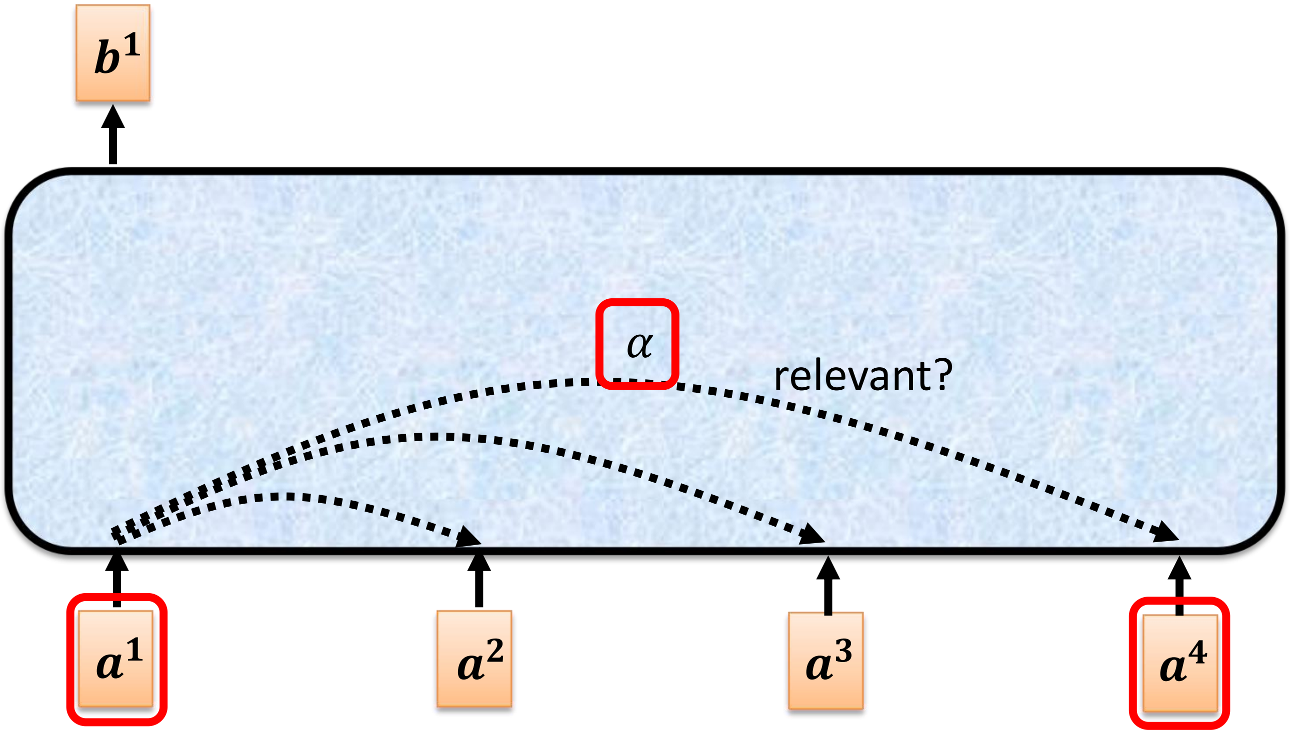
https://arxiv.org/abs/1706.037
62

Self-attention



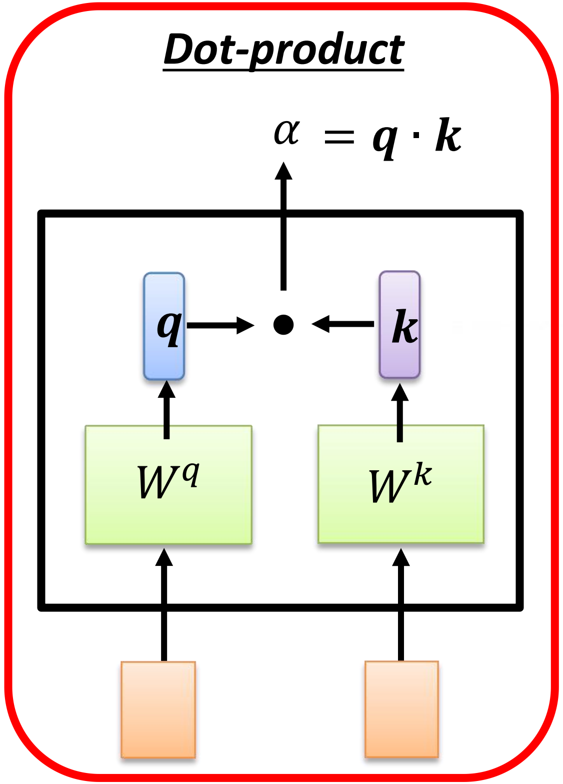
Can be either **input** or **a hidden layer**

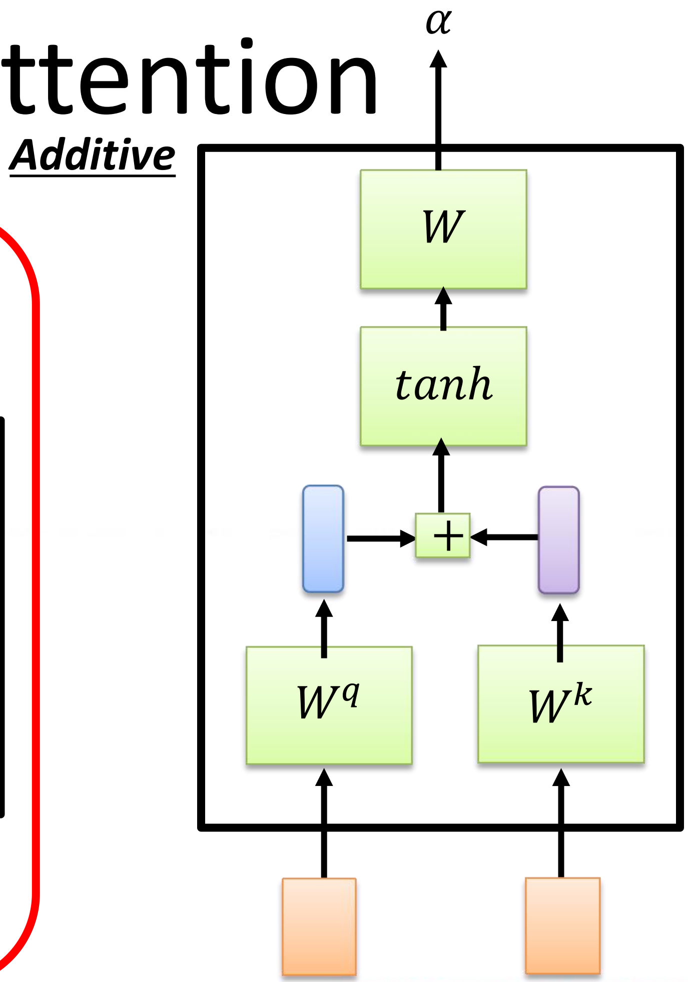
Self-attention



Find the relevant vectors in a sequence

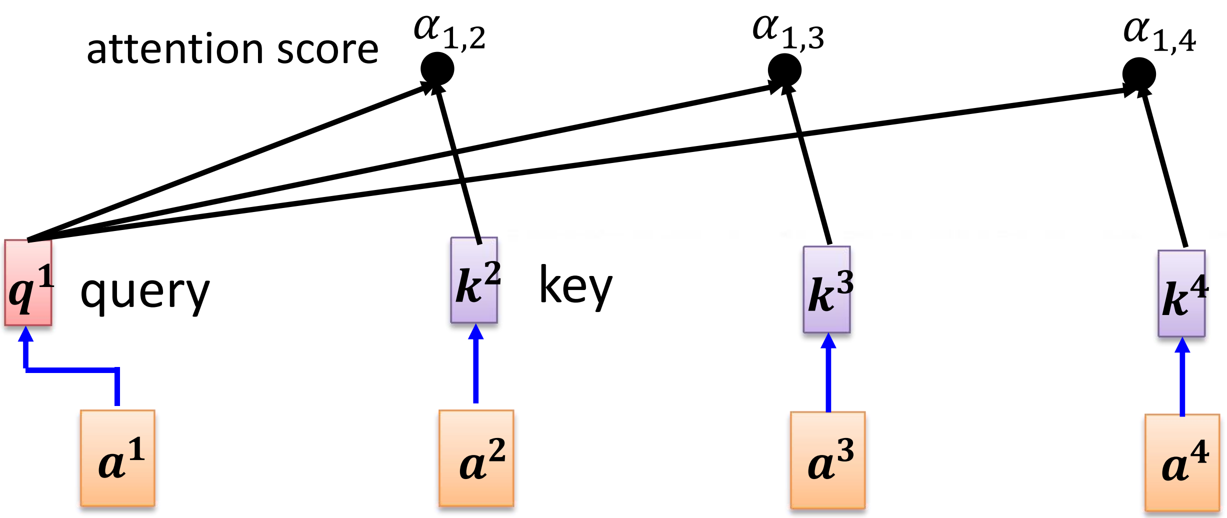
Self-attention





Self-attention

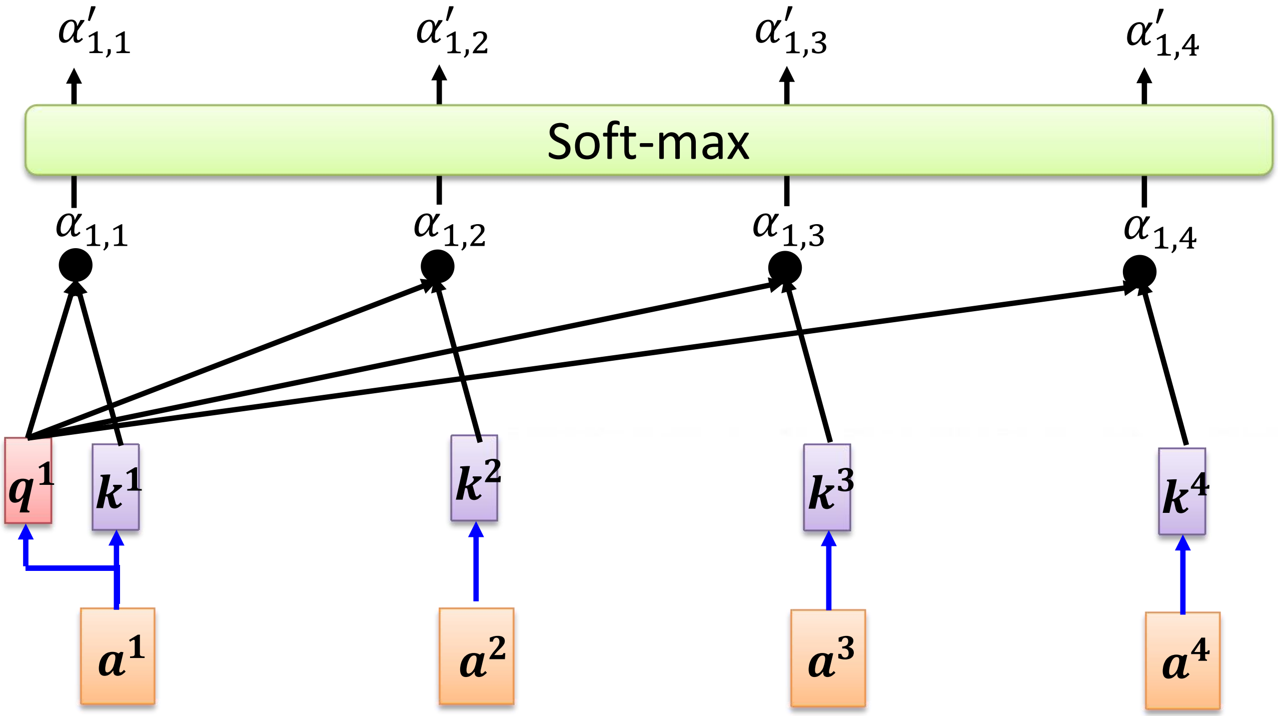
$\alpha_{1,2} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^2 \qquad \alpha_{1,3} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^3 \qquad \alpha_{1,4} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^4$



$\boldsymbol{q}^1 = W^q \boldsymbol{a}^1 \qquad \boldsymbol{k}^2 = W^k \boldsymbol{a}^2 \qquad \boldsymbol{k}^3 = W^k \boldsymbol{a}^3 \qquad \boldsymbol{k}^4 = W^k \boldsymbol{a}^4$

Self-attention

$$\alpha'_{1,i} = exp(\alpha_{1,i})/\sum_j exp(\alpha_{1,j})$$



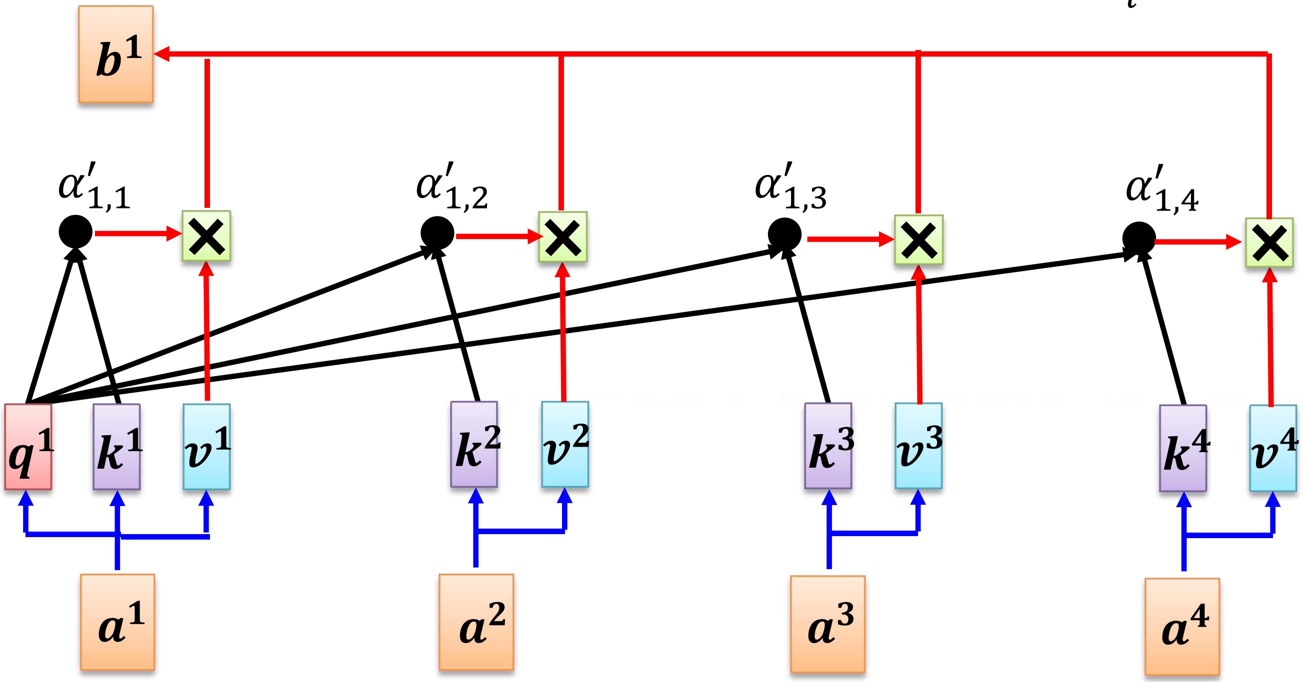
$q^1 = W^q a^1$ $\quad k^2 = W^k a^2$ $\quad k^3 = W^k a^3$ $\quad k^4 = W^k a^4$

$k^1 = W^k a^1$

Self-attention

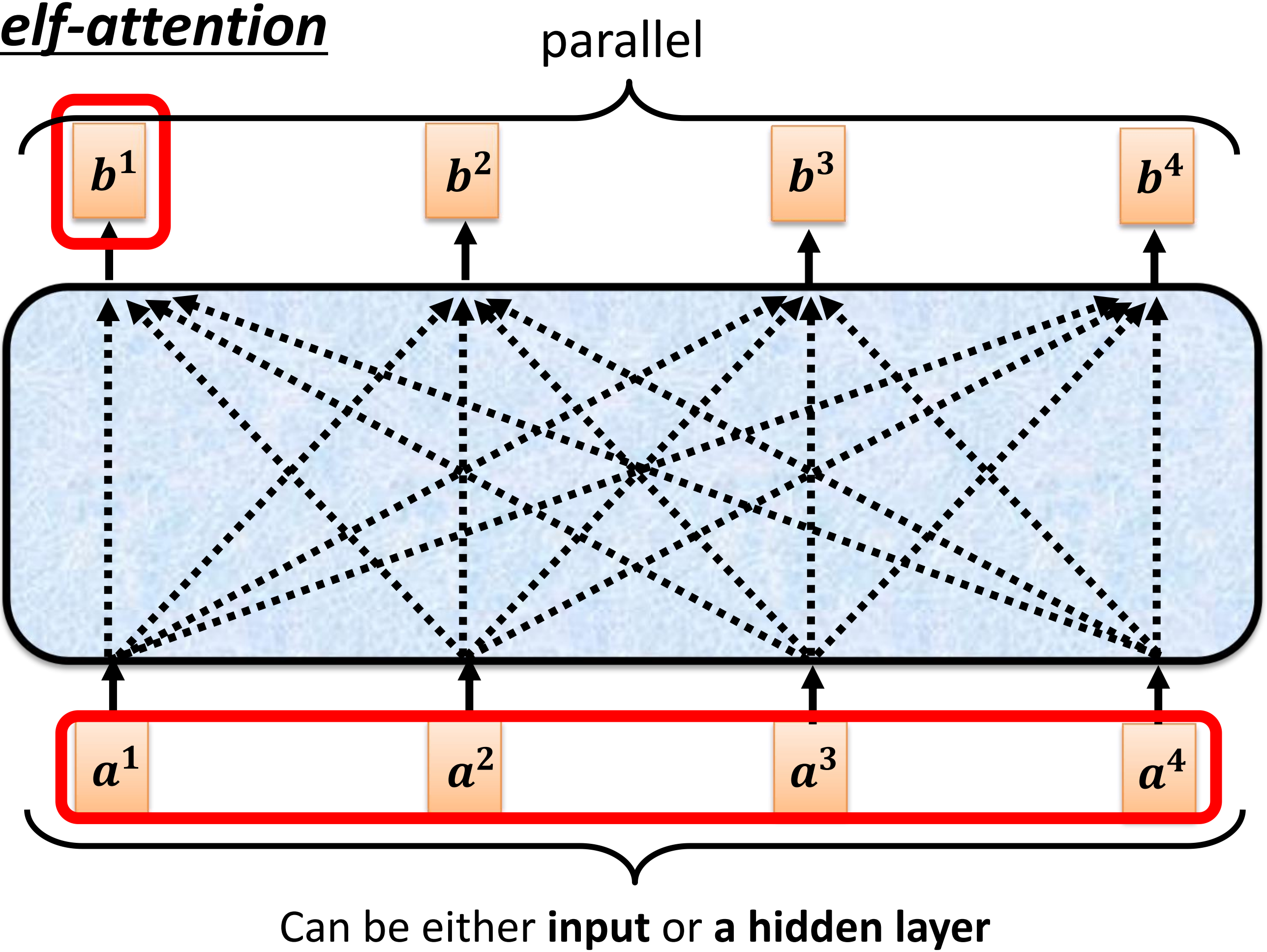
Extract information based on attention scores

$$\boldsymbol{b}^1 = \sum_i \alpha'_{1,i} \boldsymbol{v}^i$$



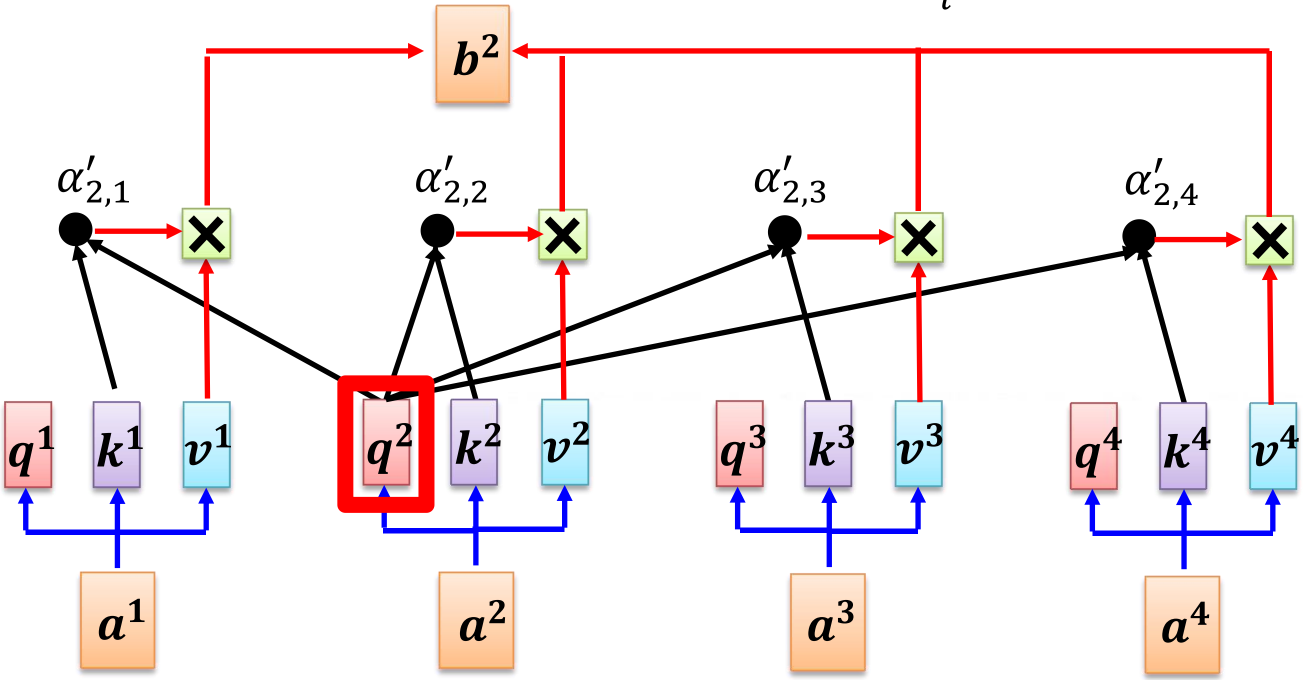
$\boldsymbol{v}^1 = W^v \boldsymbol{a}^1$ $\boldsymbol{v}^2 = W^v \boldsymbol{a}^2$ $\boldsymbol{v}^3 = W^v \boldsymbol{a}^3$ $\boldsymbol{v}^4 = W^v \boldsymbol{a}^4$

Self-attention



Self-attention

$$b^2 = \sum_i \alpha'_{2,i} v^i$$

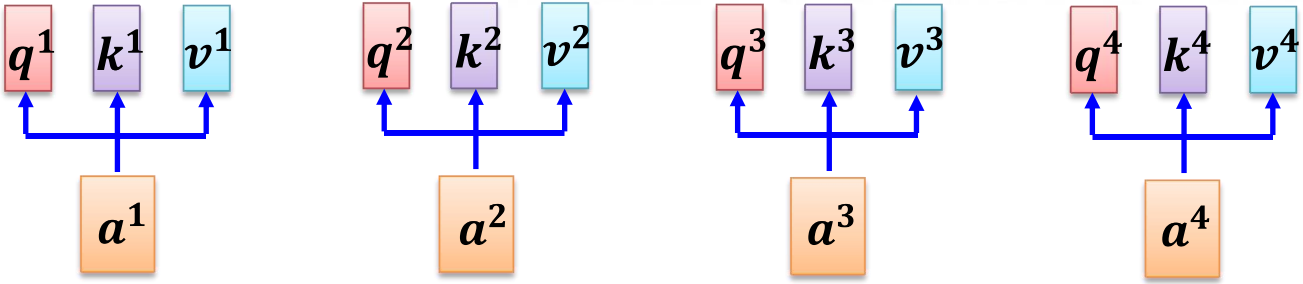


Self-attention

$$q^i = W^q a^i \qquad \underbrace{q^1\ q^2\ q^3\ q^4}_{Q} = W^q\ \underbrace{a^1\ a^2\ a^3\ a^4}_{\mathrm{I}}$$

$$k^i = W^k a^i \qquad \underbrace{k^1\ k^2\ k^3\ k^4}_{K} = W^k\ \underbrace{a^1\ a^2\ a^3\ a^4}_{\mathrm{I}}$$

$$v^i = W^v a^i \qquad \underbrace{v^1\ v^2\ v^3\ v^4}_{V} = W^v\ \underbrace{a^1\ a^2\ a^3\ a^4}_{\mathrm{I}}$$

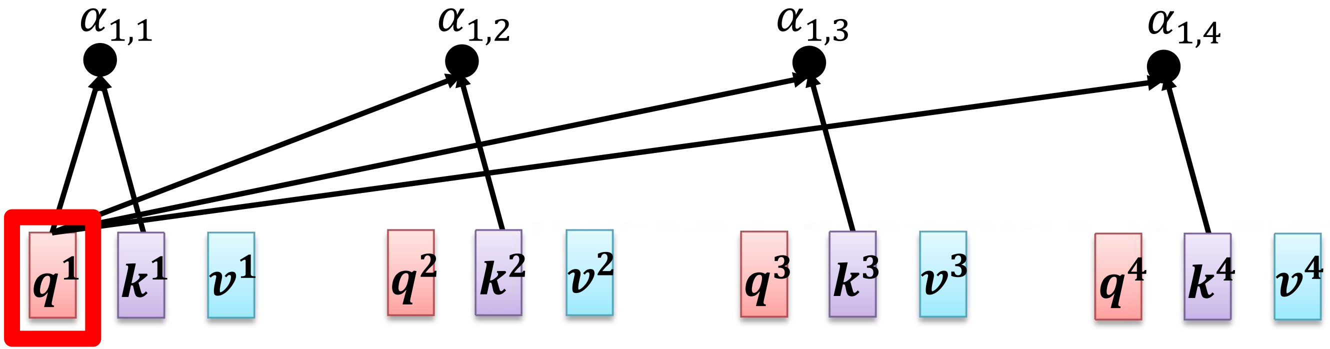


Self-attention

$$\alpha_{1,1} = k^1 \, q^1 \qquad \alpha_{1,2} = k^2 \, q^1$$

$$\alpha_{1,3} = k^3 \, q^1 \qquad \alpha_{1,4} = k^4 \, q^1$$

$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} q^1$$

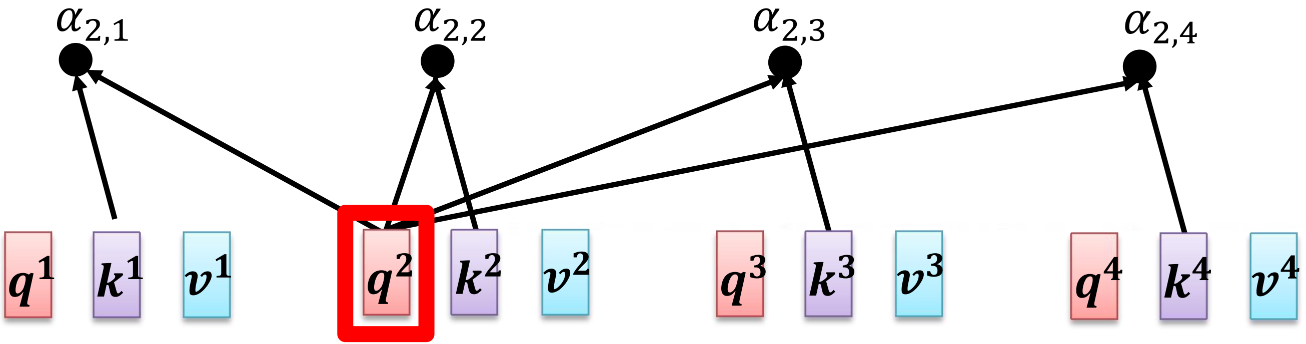


Self-attention

$\alpha_{1,1} = k^1 \; q^1 \qquad \alpha_{1,2} = k^2 \; q^1$

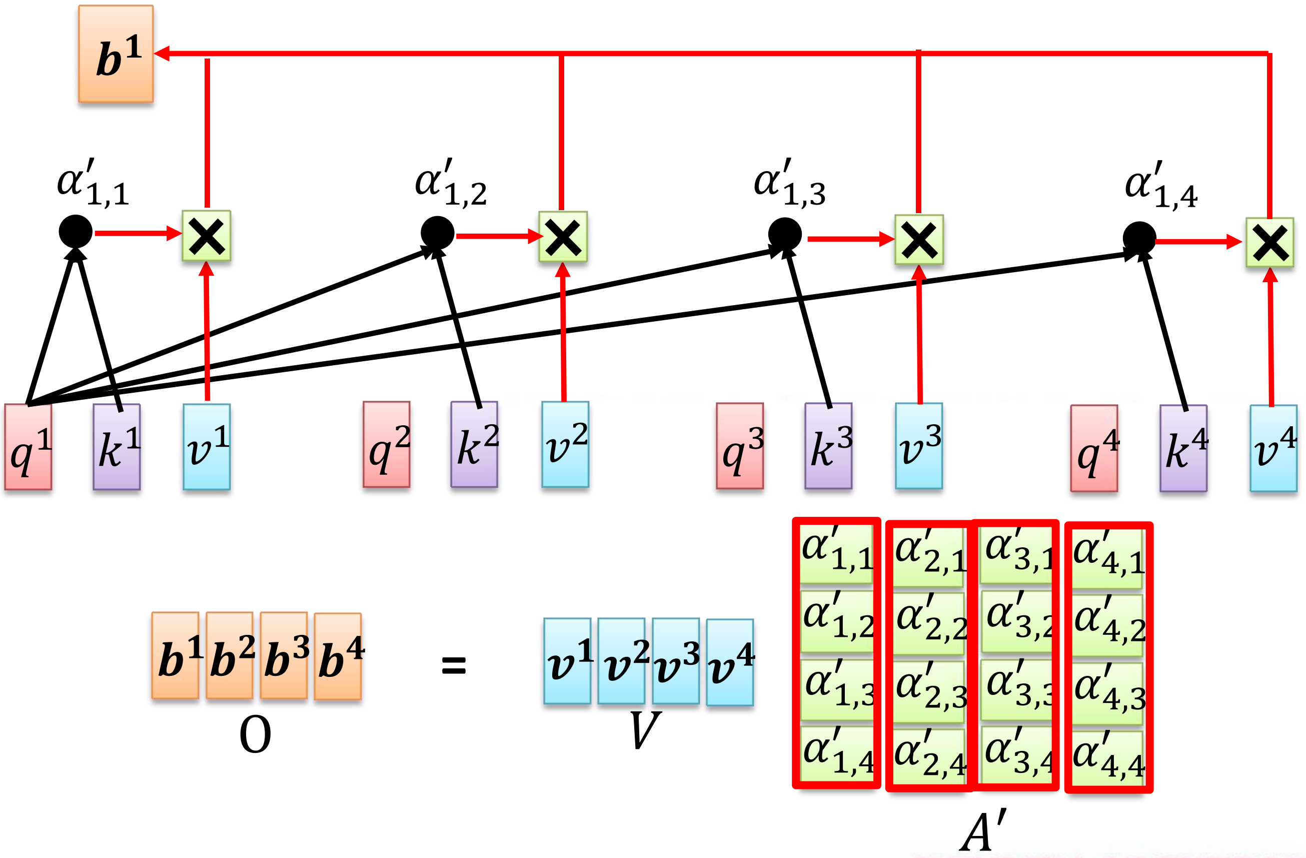
$\alpha_{1,3} = k^3 \; q^1 \qquad \alpha_{1,4} = k^4 \; q^1$

$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} q^1$$



$$\underbrace{\begin{bmatrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} \end{bmatrix}}_{A'} \Leftarrow \underbrace{\begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix}}_{K^T} \underbrace{\begin{bmatrix} q^1 & q^2 & q^3 & q^4 \end{bmatrix}}_{Q}$$

Self-attention



Self-attention

$$Q = W^q I$$

$$K = W^k I$$

$$V = W^v I$$

Parameters to be learned

$$A' \Leftarrow A = K^T Q$$

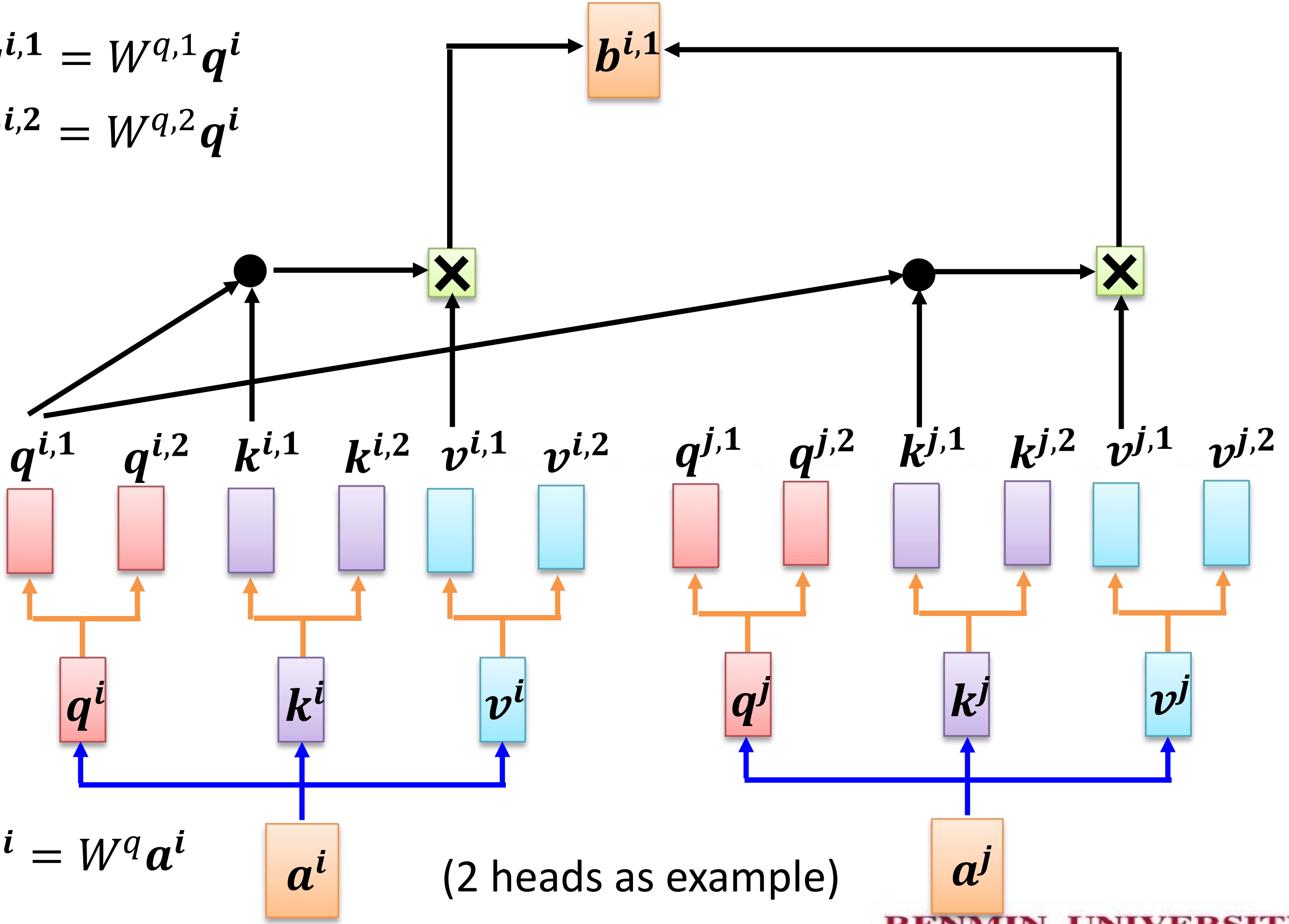
Attention Matrix

$$O = V A'$$

Multi-head Self-attention Different types of relevance

$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$



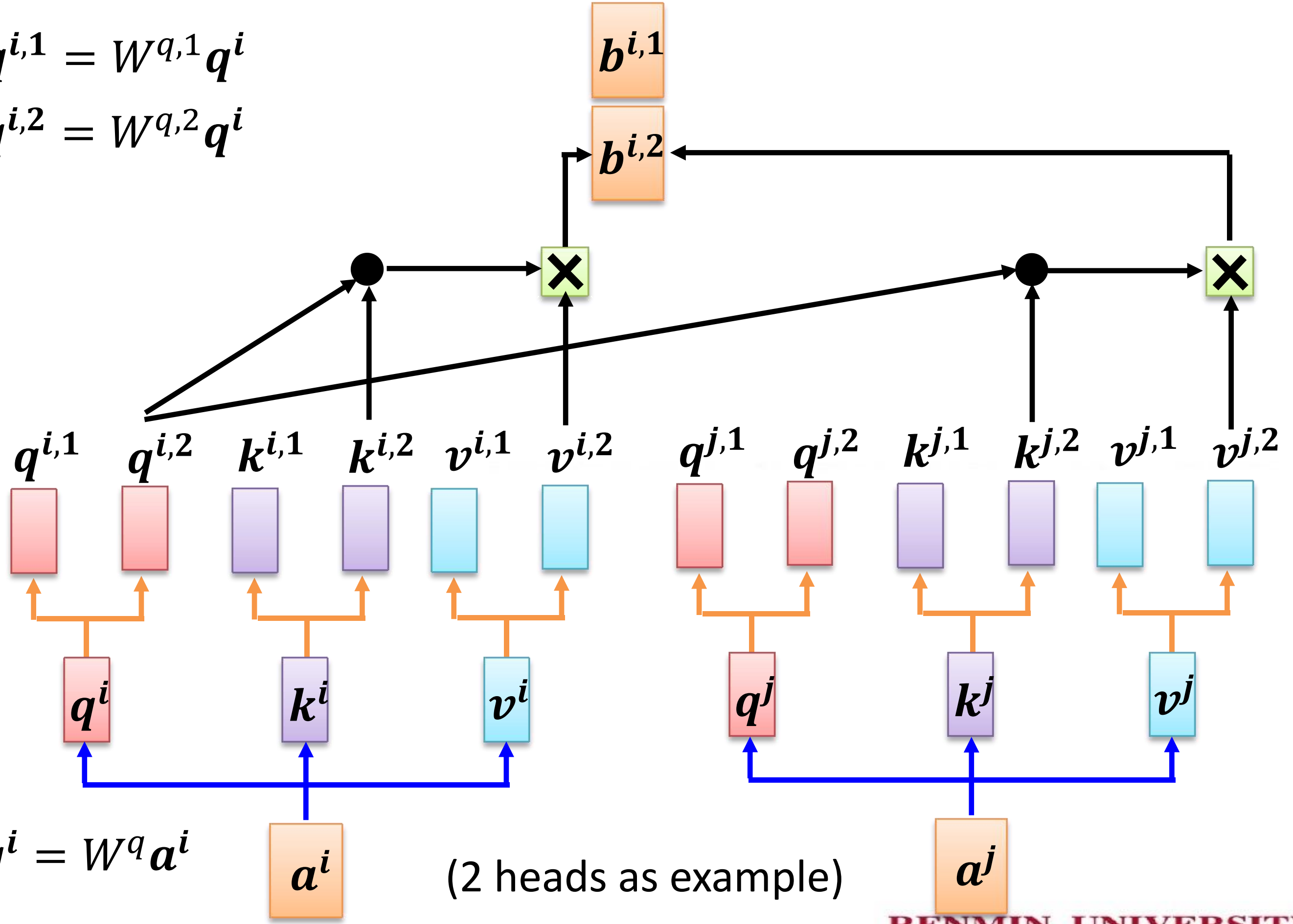
$$q^i = W^q a^i$$

(2 heads as example)

Multi-head Self-attention Different types of relevance

$q^{i,1} = W^{q,1} q^i$

$q^{i,2} = W^{q,2} q^i$



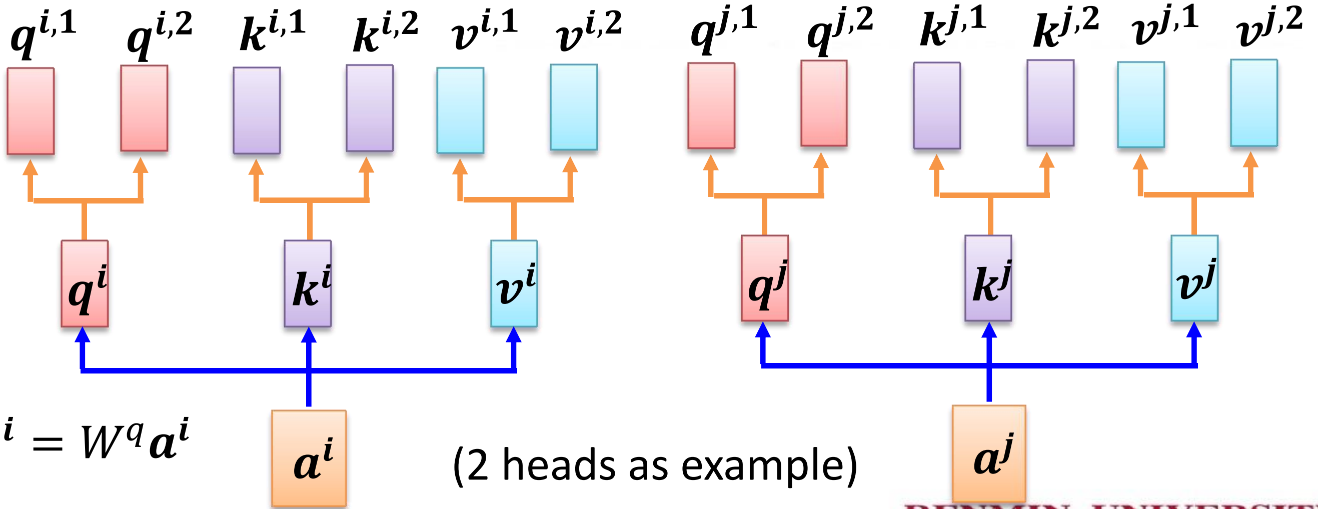
$q^i = W^q a^i$

(2 heads as example)

Multi-head Self-attention

Different types of relevance

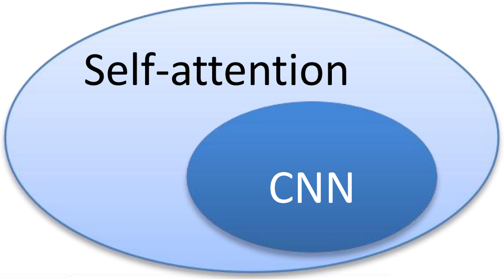
$$b^i = W^O \begin{bmatrix} b^{i,1} \\ b^{i,2} \end{bmatrix}$$

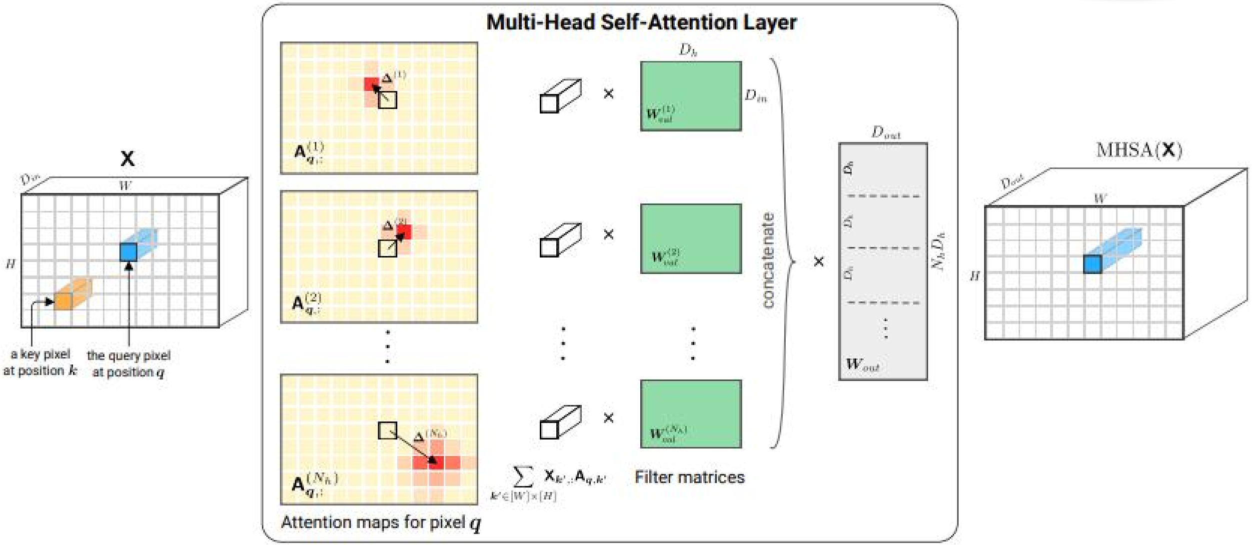


$q^i = W^q a^i$

(2 heads as example)

Self-attention v.s. CNN





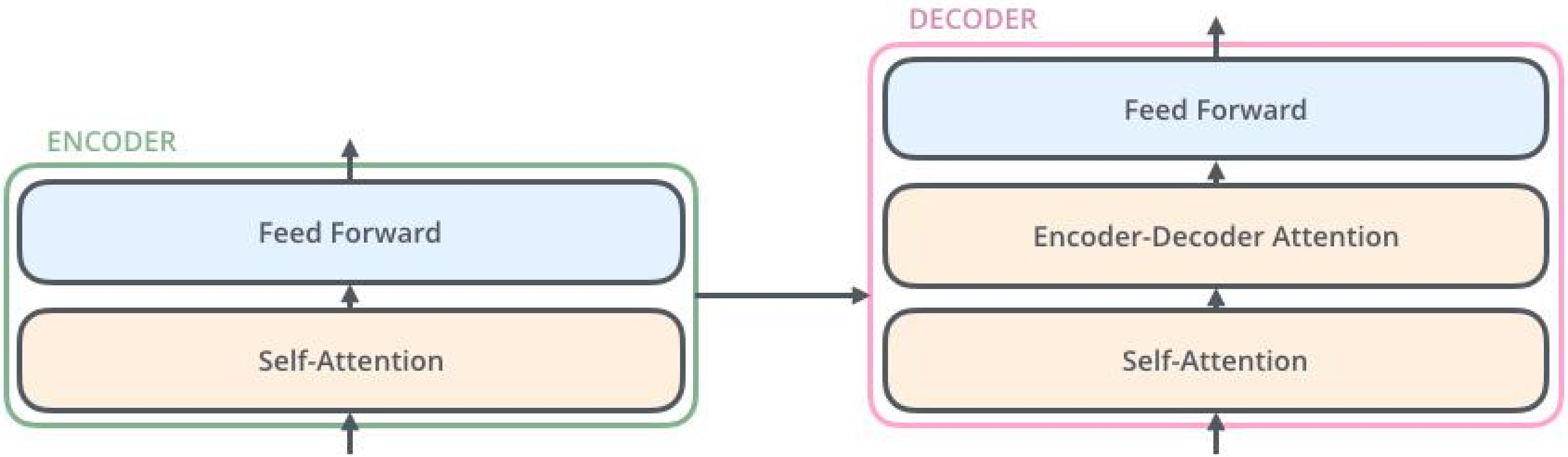
On the Relationship between Self-Attention and Convolutional Layers

https://arxiv.org/abs/1911.03584

Transformer

- Transformer is a model architecture for the machine translation task which uses attention to outperform the previous SOTA, sequence-to-sequence models with attention, on many tasks.
- Transformers are a sequence-to-sequence model architecture. The difference is in their use of an improved form of attention known as self-attention, which in addition to its expressive power has the computational advantage that it's expressible as a matrix operation.
- The core architecture consists of a stack of encoders fully connected to a stack of decoders. Each encoder consists of two blocks: a self-attention component, and a feed forward network. Each decoder consists of three blocks: a self-attention component, an encoder-decoder attention component, and a feed forward component.

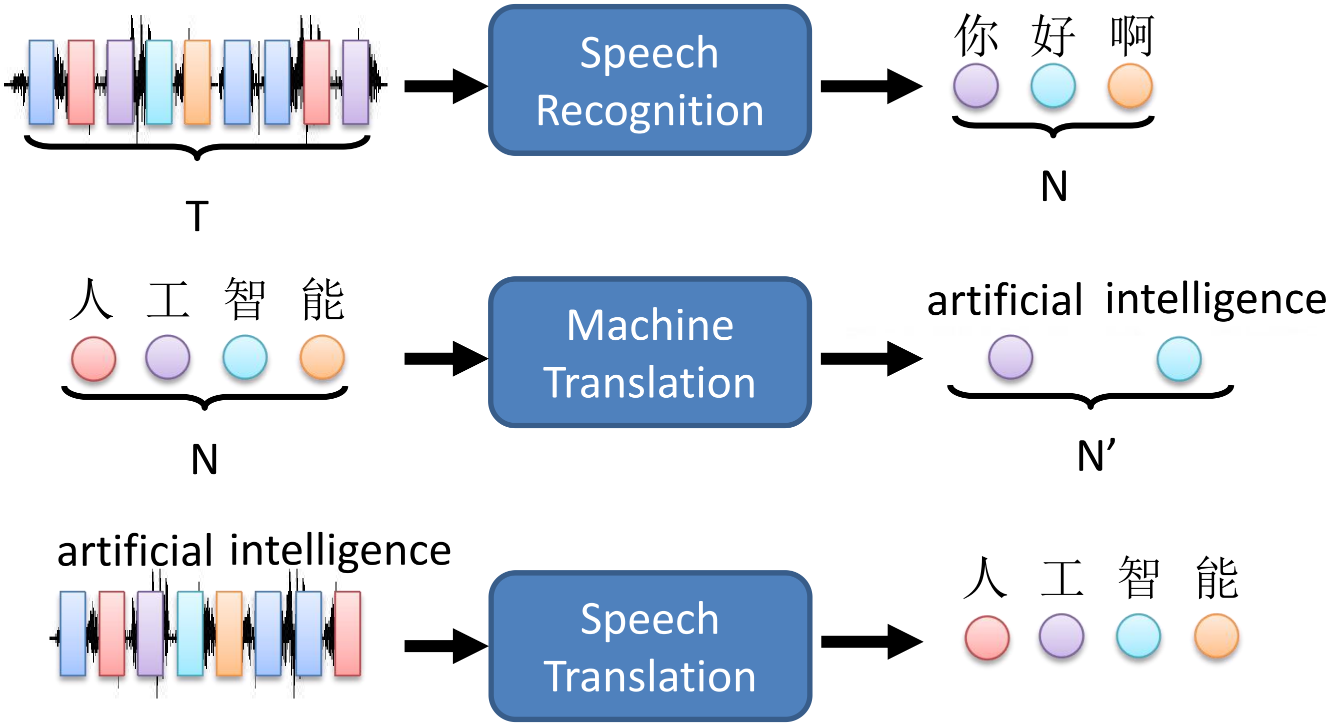
Transformer



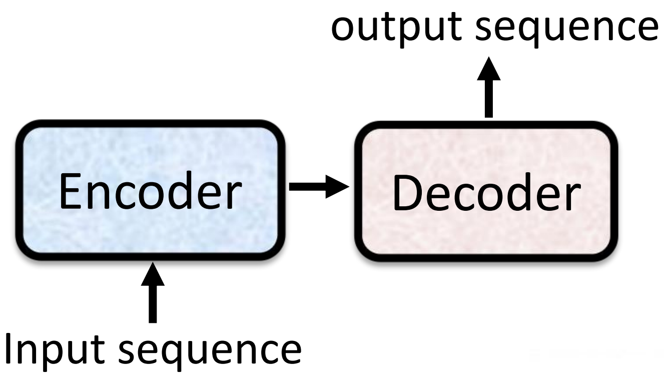
Sequence-to-sequence (Seq2seq)

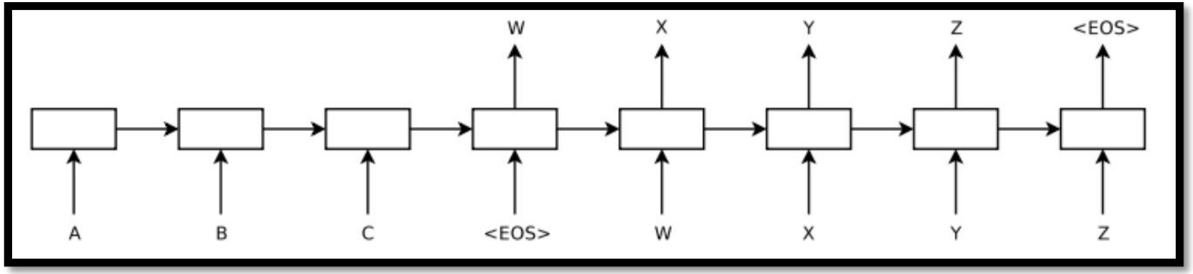
Input a sequence, output a sequence

The output length is determined by model.



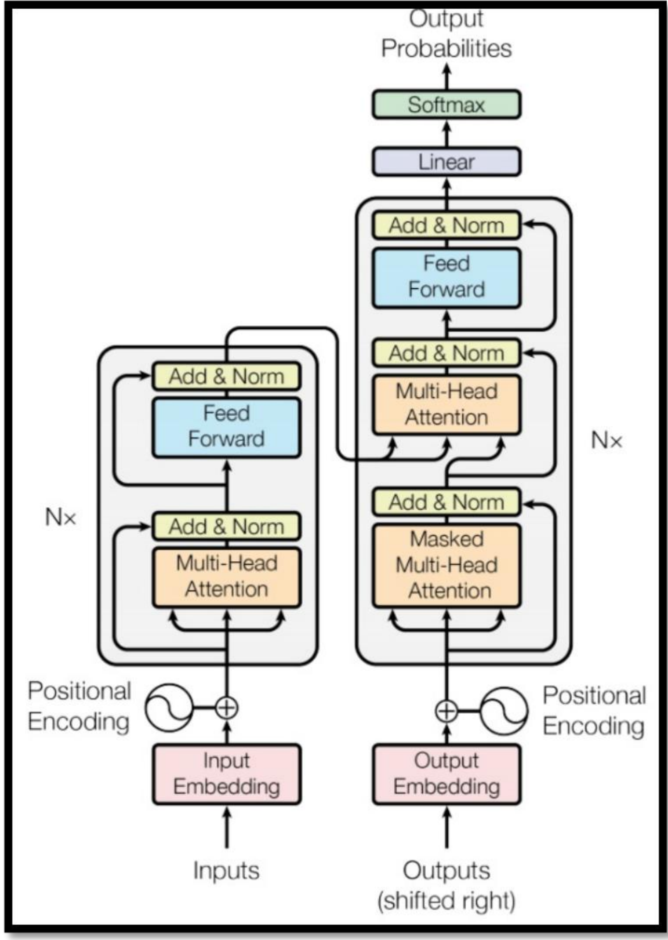
Seq2seq





Sequence to Sequence Learning with Neural Networks

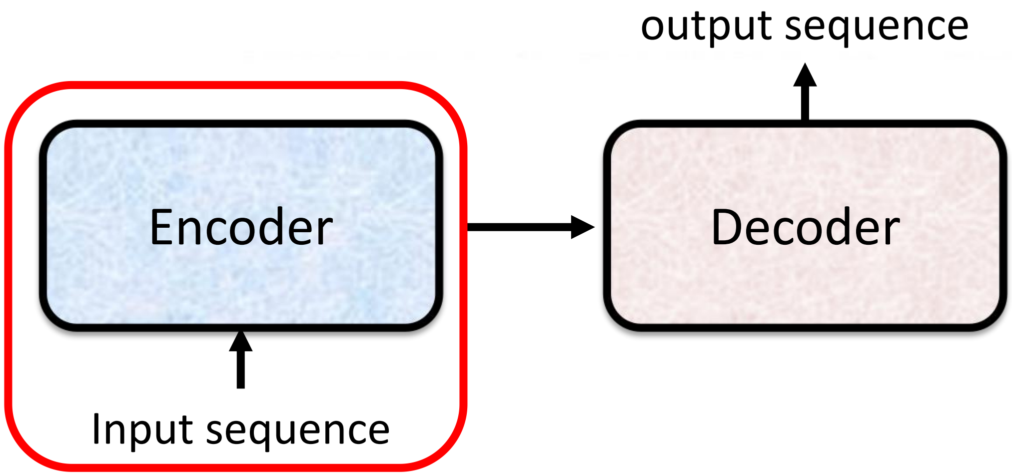
https://arxiv.org/abs/1409.3215



Transformer

https://arxiv.org/abs/1706.03762

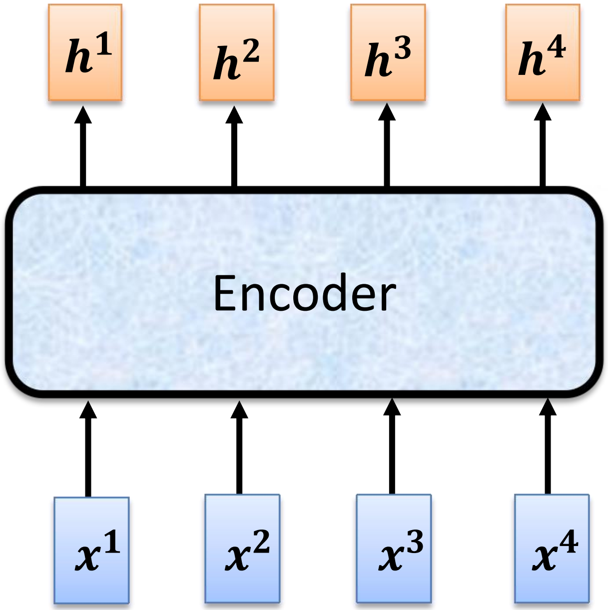
Encoder

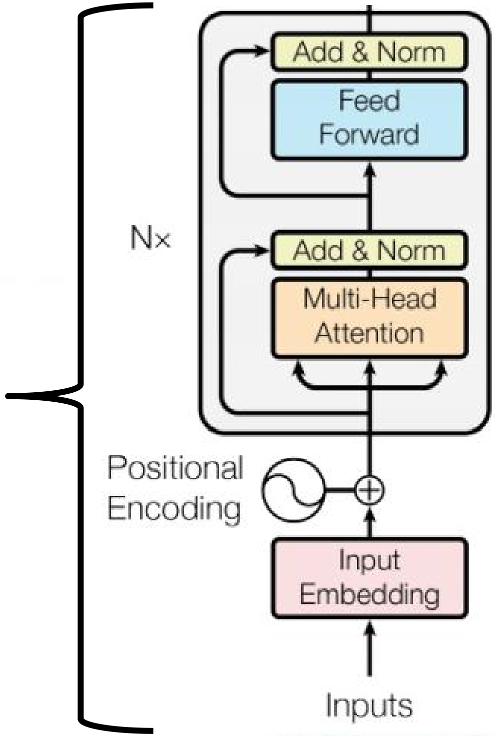


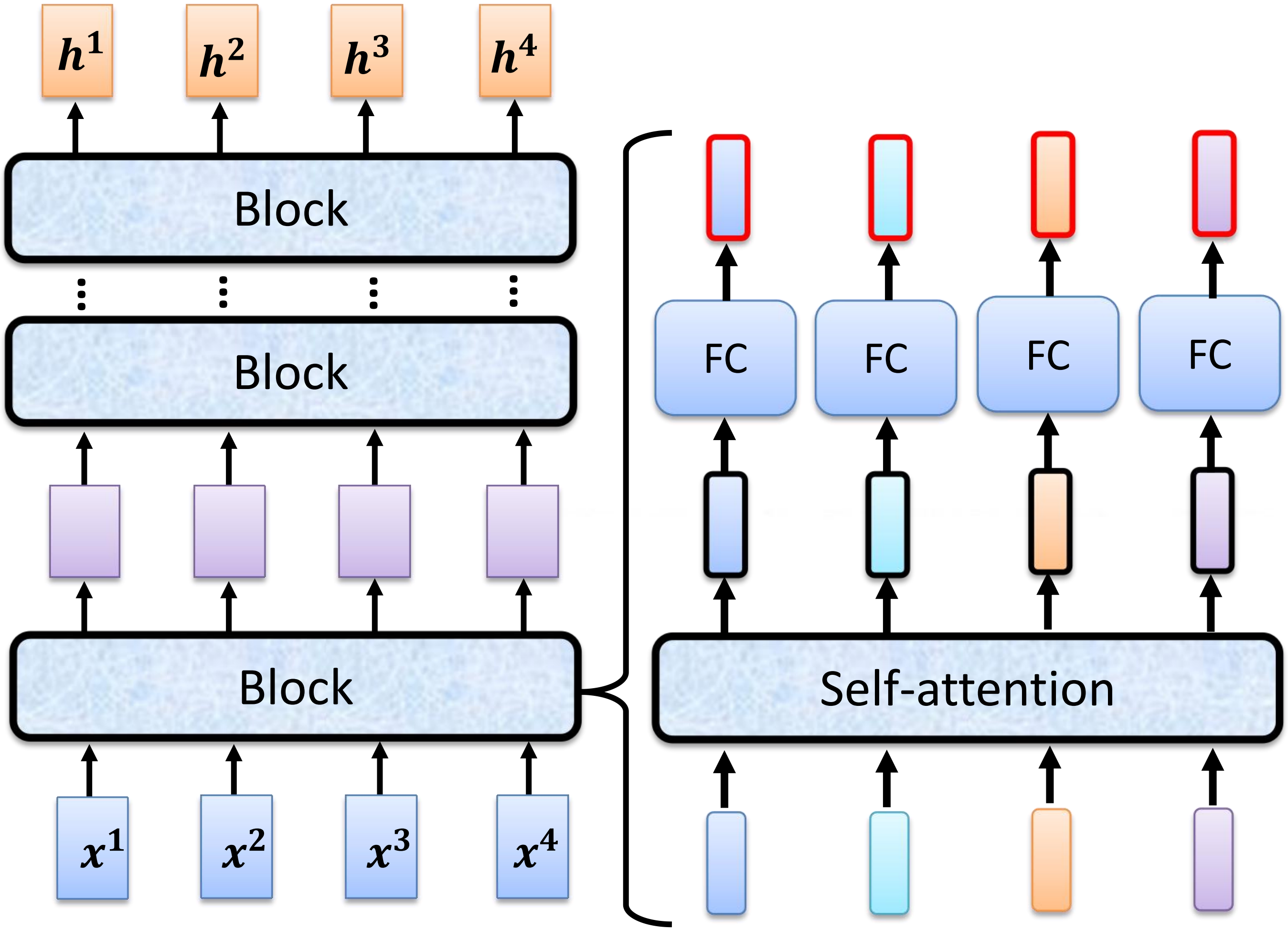
Encoder

Transformer's Encoder

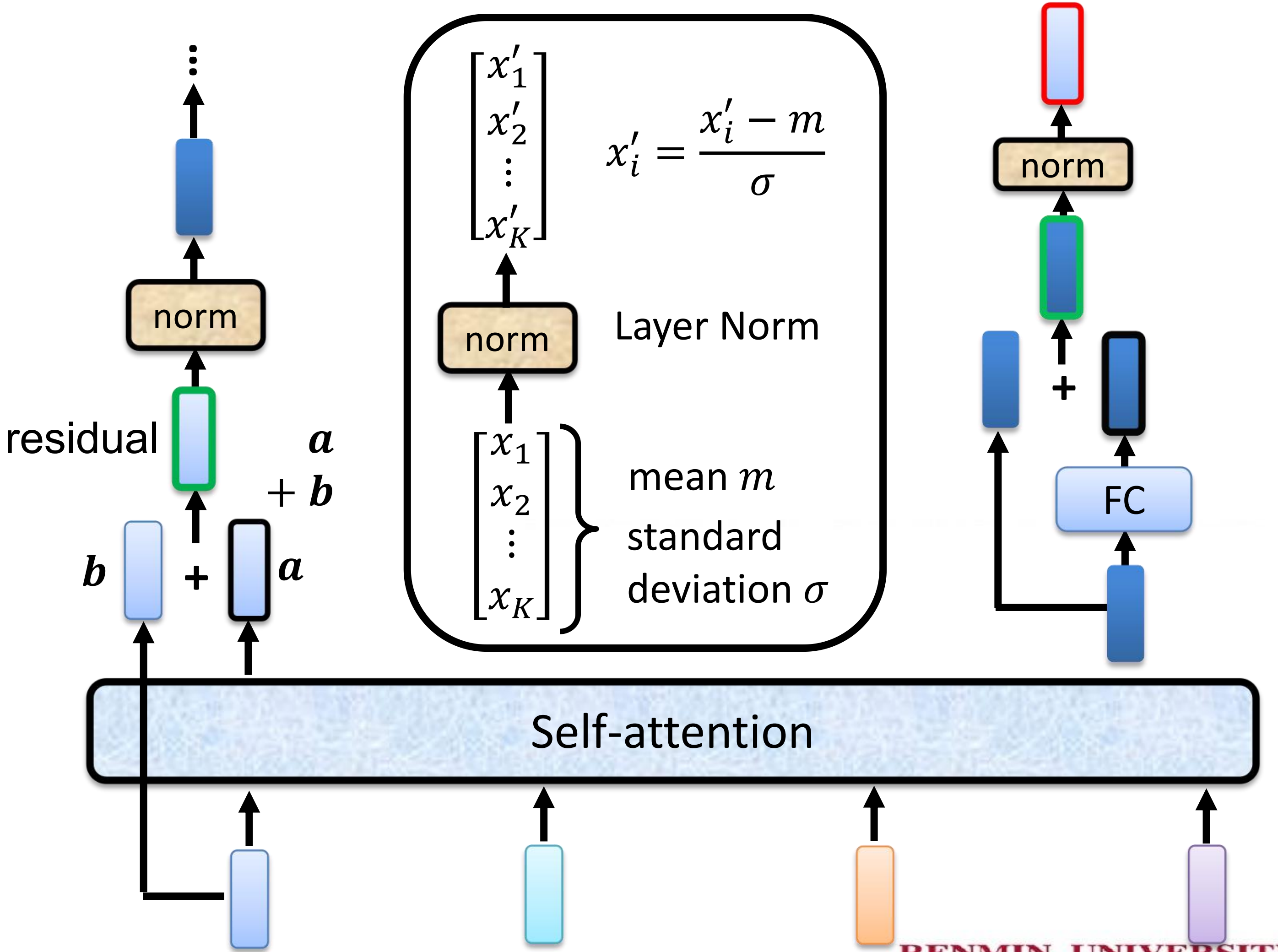
You can use **RNN** or **CNN**.





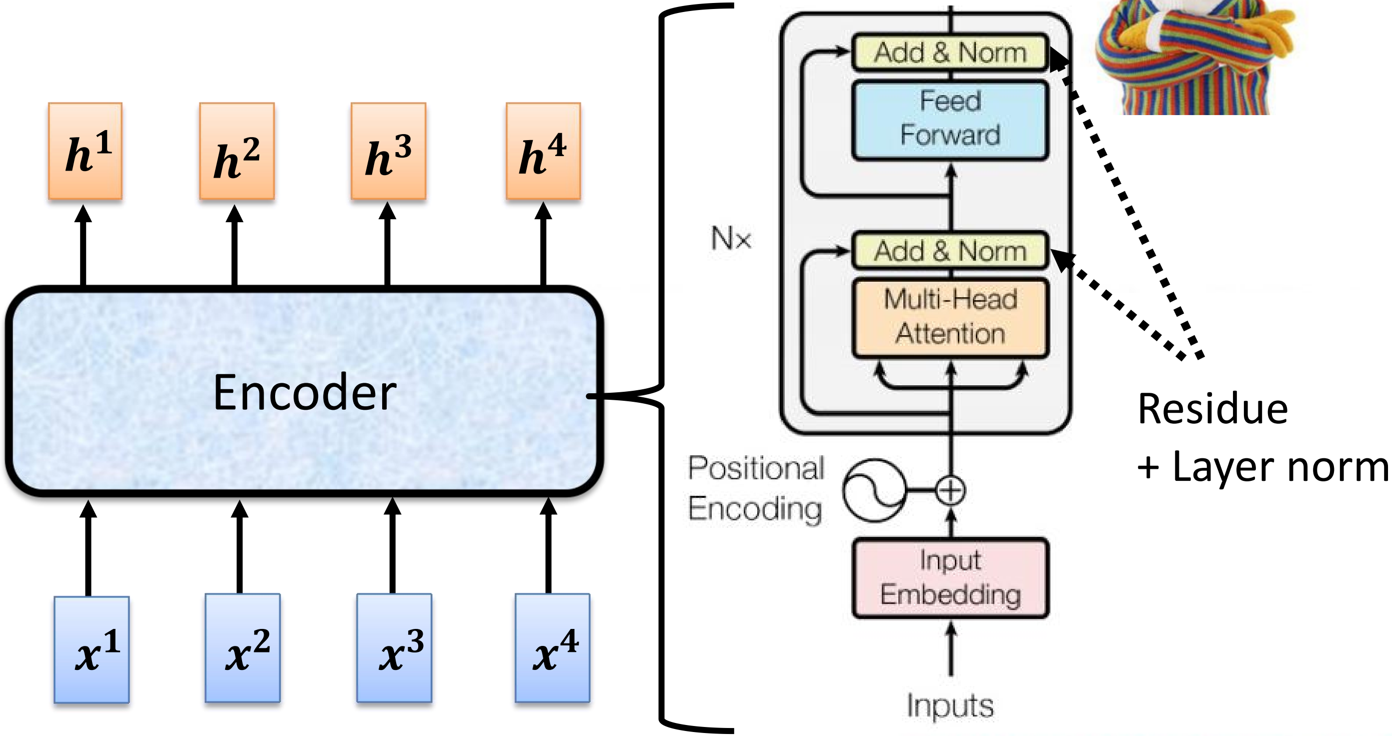


h^1
h^2
h^3
h^4
Block
Block
Block
x^1
x^2
x^3
x^4
FC
FC
FC
FC
Self-attention

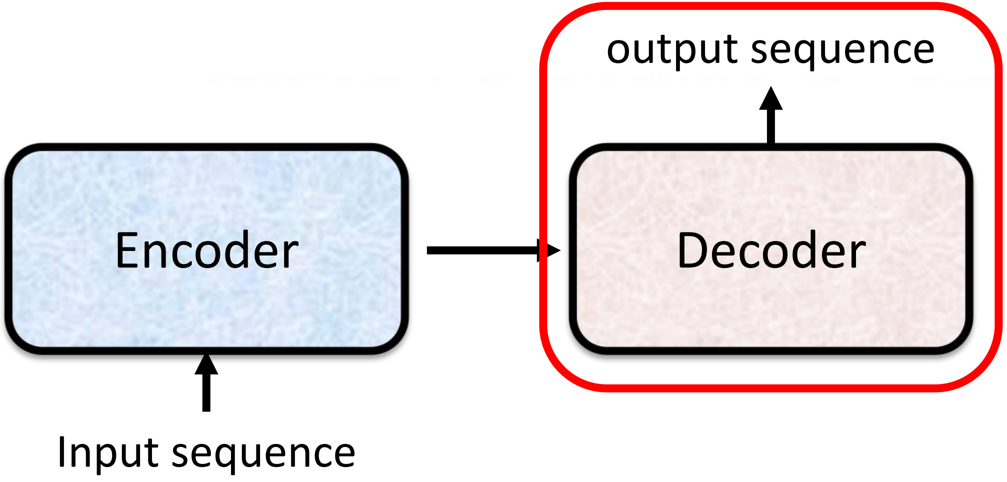

norm
residual
$a + b$
b
$+$
a
$x'_1, x'_2, \vdots, x'_K$
$x'_i = \frac{x'_i - m}{\sigma}$
norm
Layer Norm
$x_1, x_2, \vdots, x_K$
mean m
standard deviation σ
norm
$+$
FC
Self-attention

I use the **same** network architecture as **transformer encoder**.

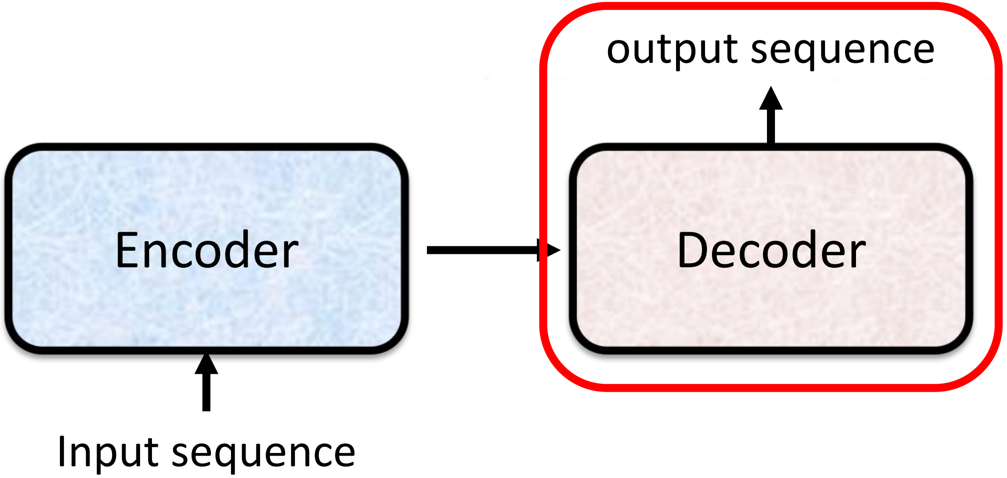
BERT



Decoder

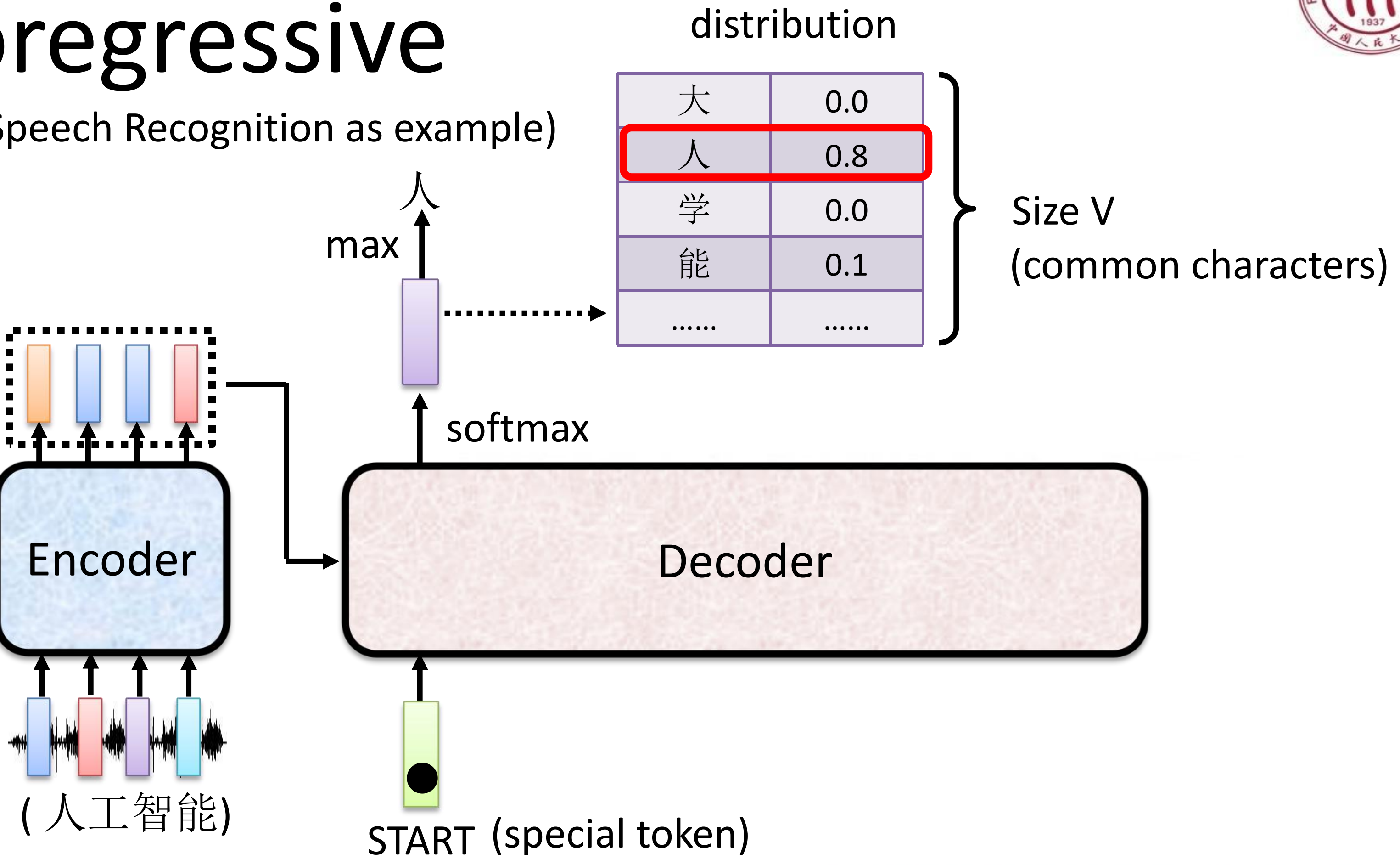


Decoder
– Autoregressive (AT)

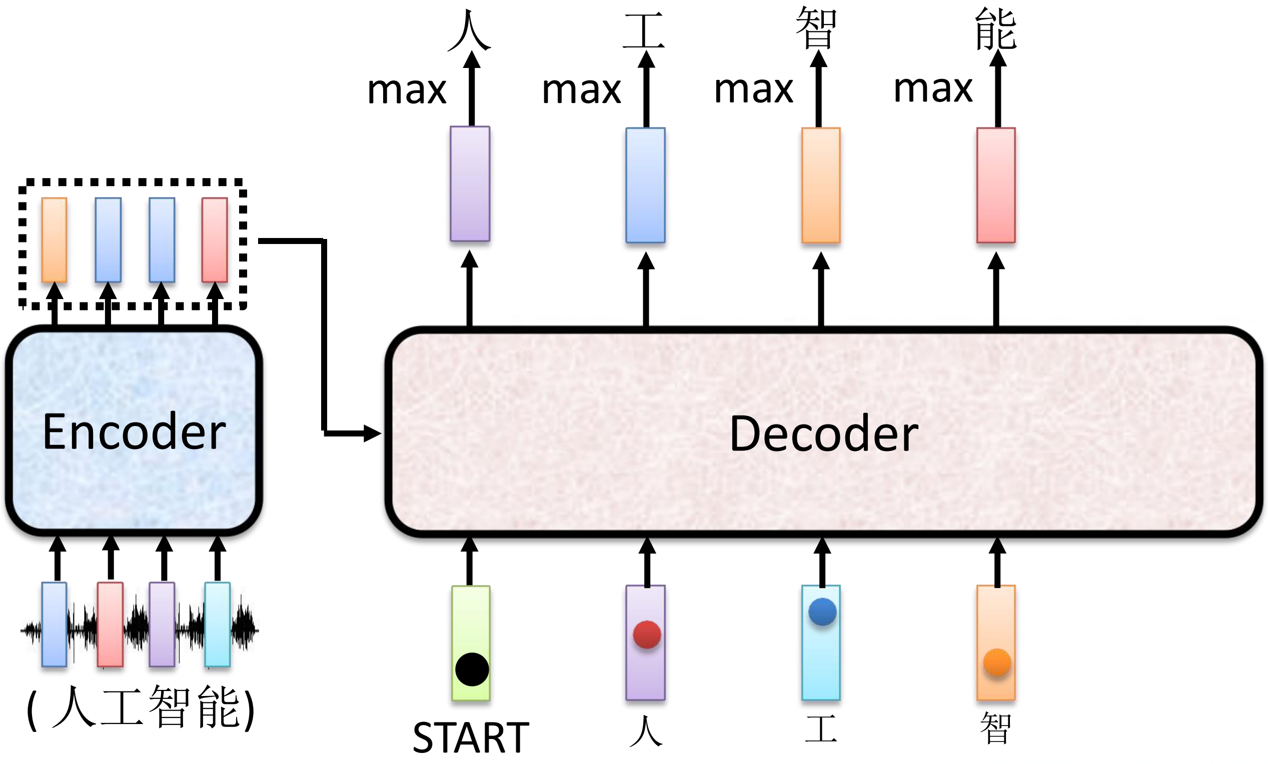


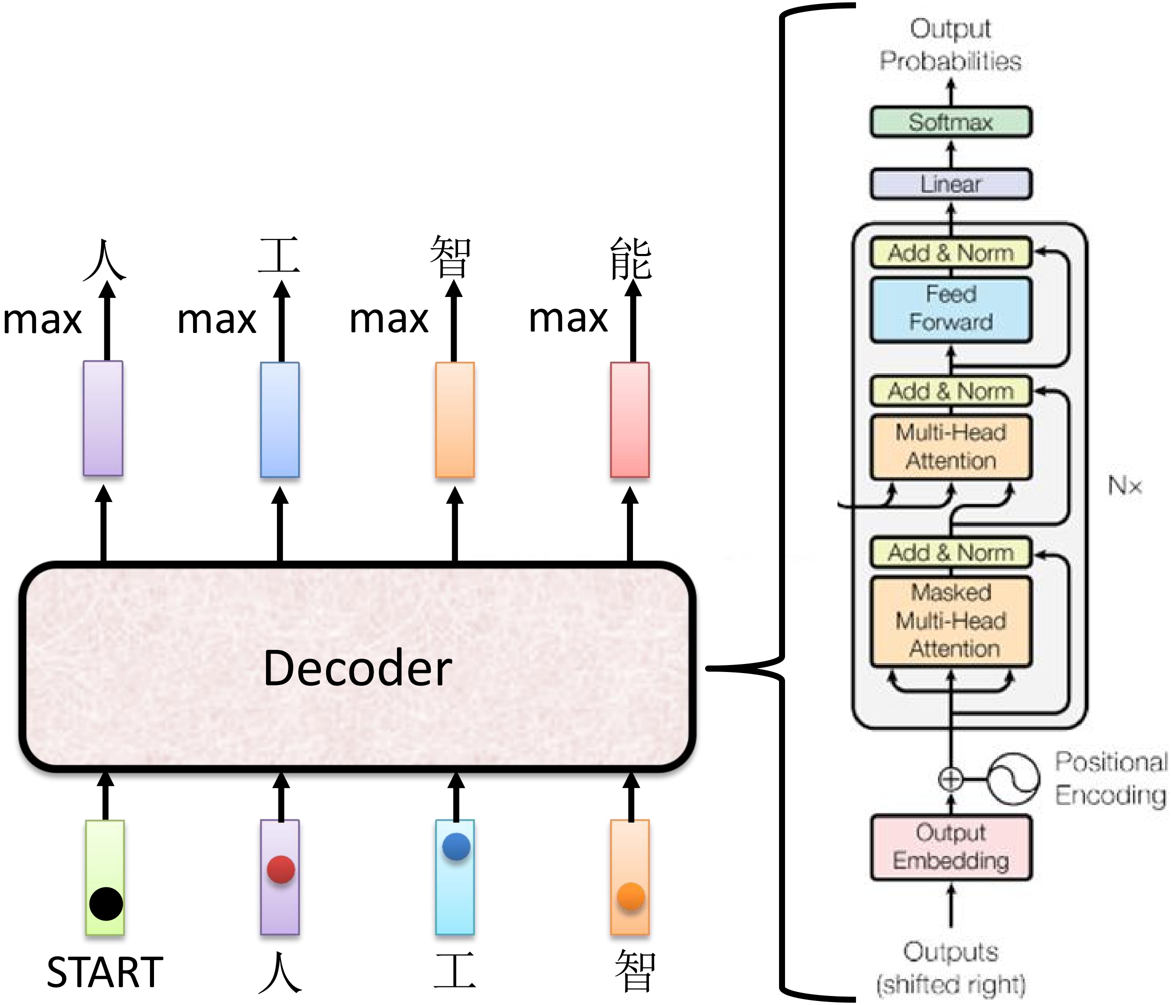
Autoregressive

(Speech Recognition as example)

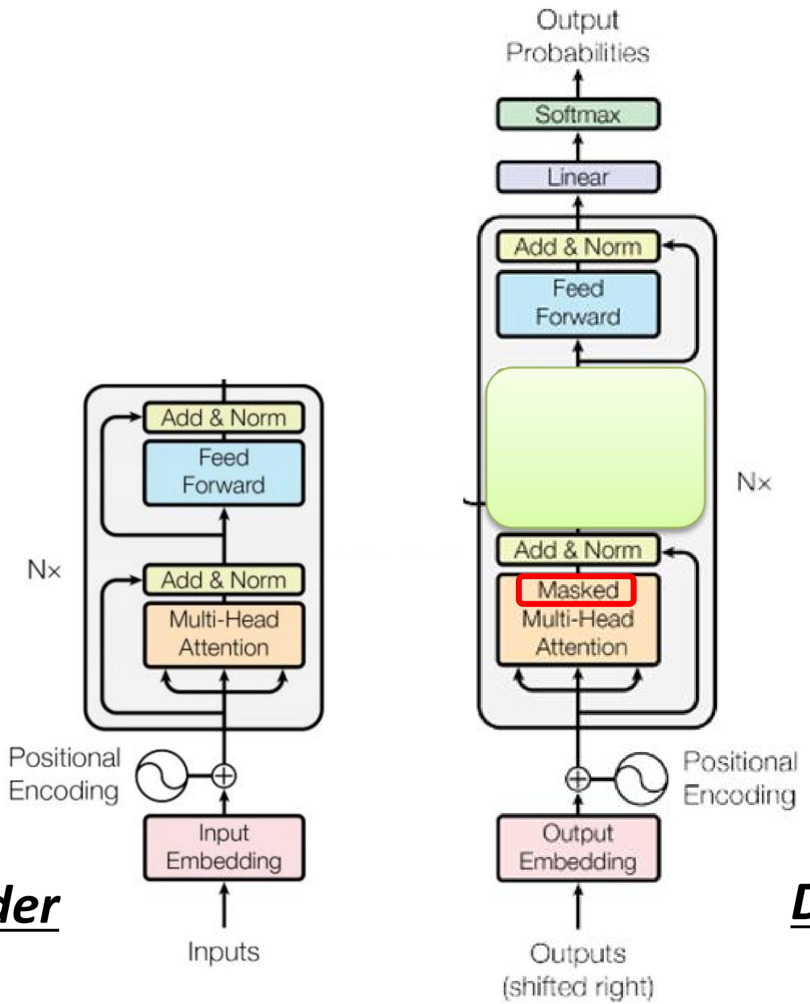


Autoregressive





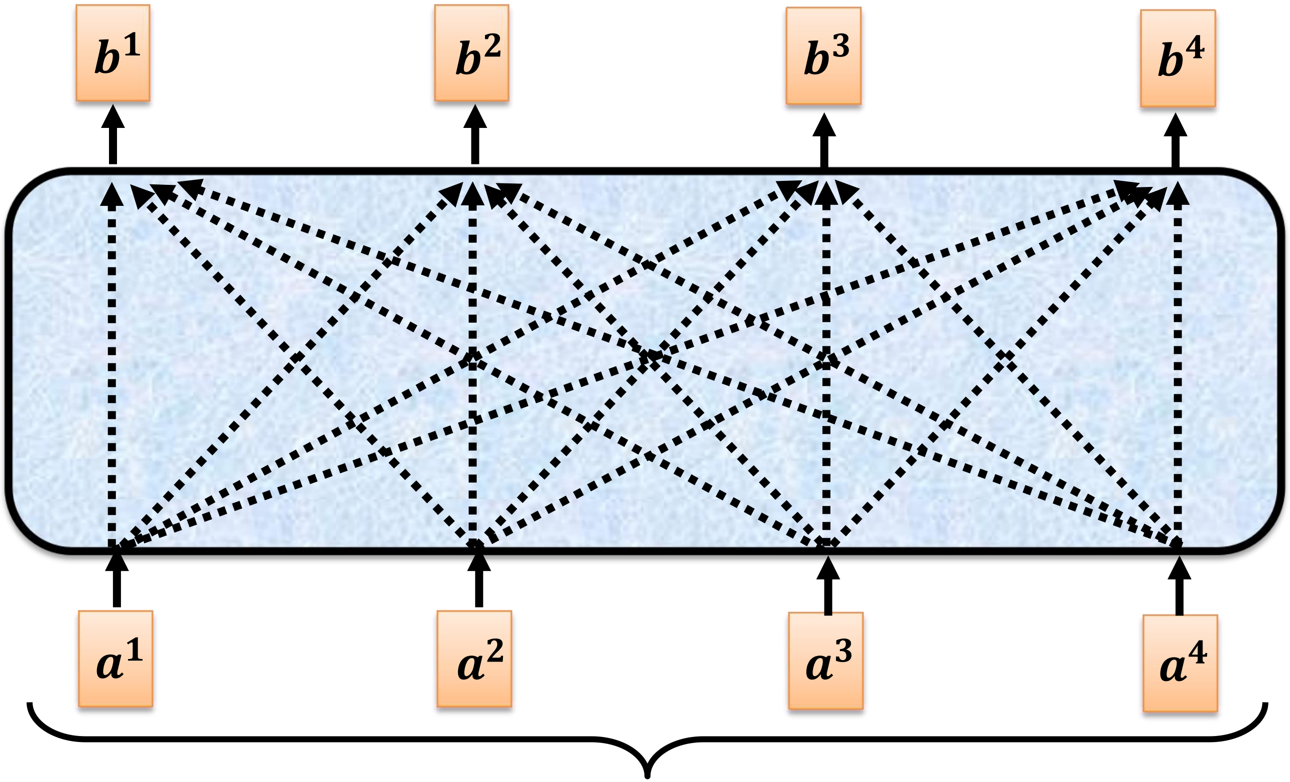
人
工
智
能
max
max
max
max
Decoder
START
人
工
智
Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)



Encoder

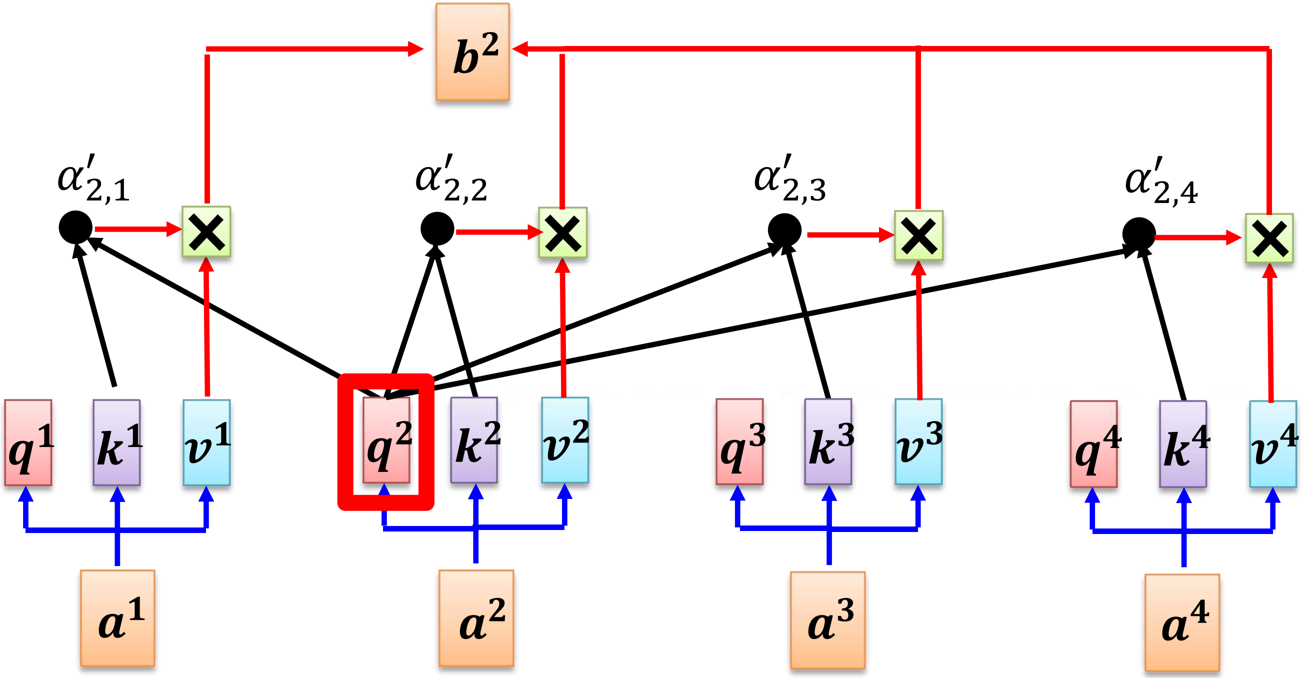
Decoder

Self-attention ➡ Masked Self-attention



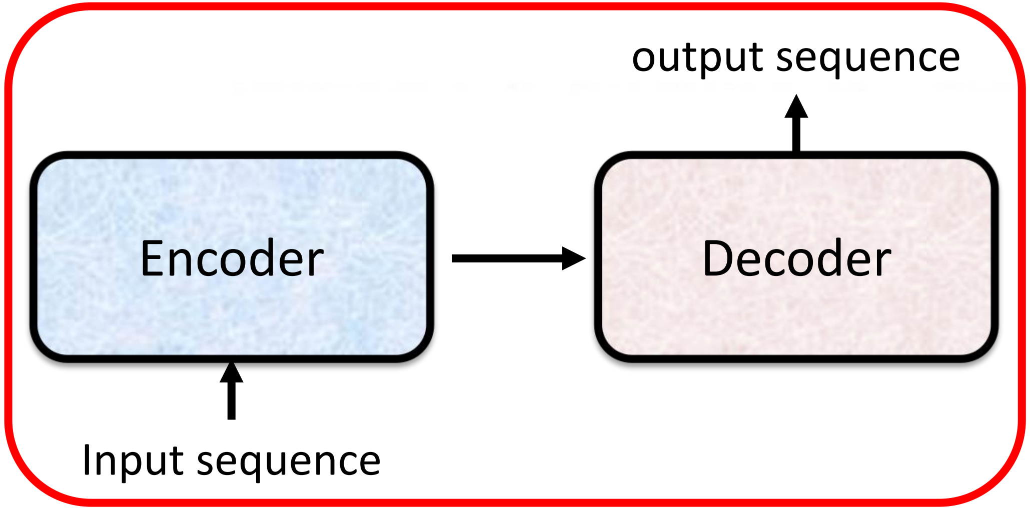
Can be either **input** or **a middle hidden layer**

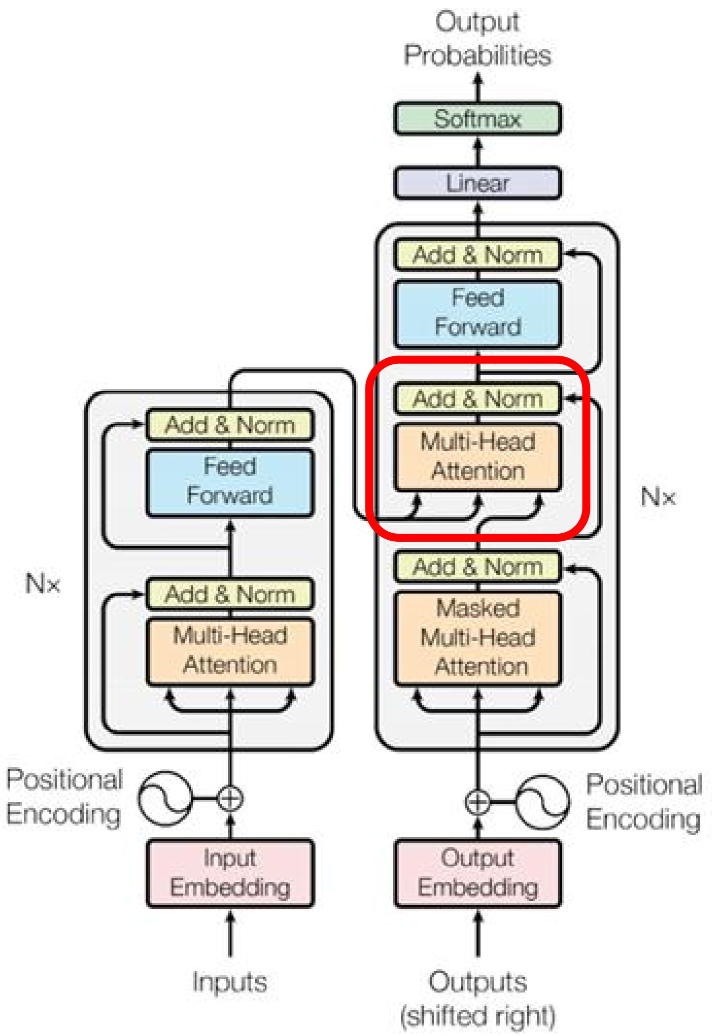
Self-attention → Masked Self-attention



Why masked? Consider how does decoder work

Encoder-Decoder




Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Classify Text Data Using BERT

- This example shows how to classify text data using a pretrained BERT model as a feature extractor.
- The simplest use of a pretrained BERT model is to use it as a feature extractor. In particular, you can use the BERT model to convert documents to feature vectors which you can then use as input to train a deep learning classification network.
- This example shows how to use a pretrained BERT model to classify failure events given a data set of factory reports.

Load Pretrained BERT Model

- Load a pretrained BERT model using the |bert| function. The model consists of a tokenizer that encodes text as sequences of integers, and a structure of parameters.

```
mdl = bert
```

- View the BERT model tokenizer. The tokenizer encodes text as sequences of integers and holds the details of padding, start, separator and mask tokens.

```
tokenizer = mdl.Tokenizer
```

Load Data

- Load the example data. The file |factoryReports.csv| contains factory reports, including a text description and categorical labels for each event.

```
filename = "factoryReports.csv";
data = readtable(filename,"TextType","string");
head(data)
```

8×5 table

Description	Category	Urgency	Resolution	Cost
"Items are occasionally getting stuck in the scanner spools."	"Mechanical Failure"	"Medium"	"Readjust Machine"	45
"Loud rattling and banging sounds are coming from assembler pistons."	"Mechanical Failure"	"Medium"	"Readjust Machine"	35
"There are cuts to the power when starting the plant."	"Electronic Failure"	"High"	"Full Replacement"	16200
"Fried capacitors in the assembler."	"Electronic Failure"	"High"	"Replace Components"	352
"Mixer tripped the fuses."	"Electronic Failure"	"Low"	"Add to Watch List"	55
"Burst pipe in the constructing agent is spraying coolant."	"Leak"	"High"	"Replace Components"	371
"A fuse is blown in the mixer."	"Electronic Failure"	"Low"	"Replace Components"	441
"Things continue to tumble off of the belt."	"Mechanical Failure"	"Low"	"Readjust Machine"	38

Load Data

- The goal of this example is to classify events by the label in the |Category| column. To divide the data into classes, convert these labels to categorical.

```
data.Category = categorical(data.Category);
```

- View the number of classes.

```
classes = categories(data.Category);
numClasses = numel(classes)
% 4
```

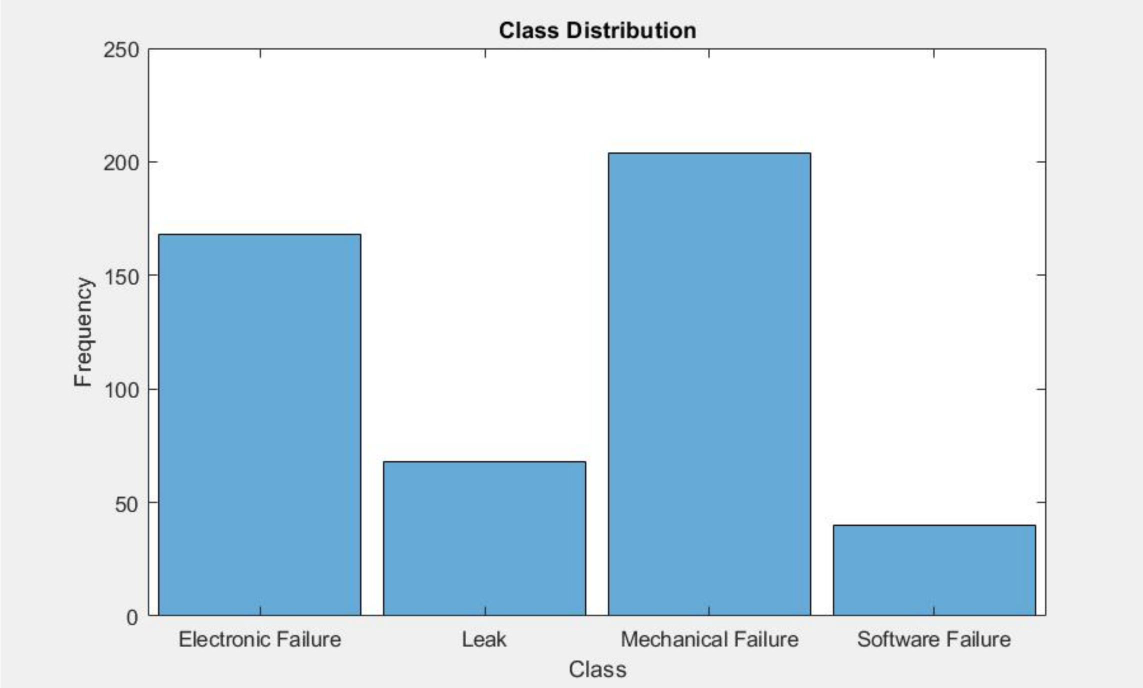
4x1 cell

	1
1	Electronic Failure
2	Leak
3	Mechanical Failure
4	Software Failure

Load Data

- View the distribution of the classes in the data using a histogram.

```
figure
histogram(data.Category);
xlabel("Class")
ylabel("Frequency")
title("Class Distribution")
```



Load Data

- Encode the text data using the BERT model tokenizer using the |encode| function and add the tokens to the training data table.

```
data.Tokens = encode(tokenizer, data.Description);
```

480x6 table

	1 Description	2 Category	3 Urgency	4 Resolution	5 Cost	6 Tokens
1	"Items are occasionally getting st...	Mechanica...	"Medium"	"Readjust M...	45	1x14 double
2	"Loud rattling and banging soun...	Mechanica...	"Medium"	"Readjust M...	35	1x14 double
3	"There are cuts to the power whe...	Electronic ...	"High"	"Full Replac...	16200	1x13 double
4	"Fried capacitors in the assemble...	Electronic ...	"High"	"Replace Co...	352	1x11 double
5	"Mixer tripped the fuses."	Electronic ...	"Low"	"Add to Wat...	55	[102,23229,21130,1997,19977,2016,1013,103]
6	"Burst pipe in the constructing ag...	Leak	"High"	"Replace Co...	371	1x13 double
7	"A fuse is blown in the mixer."	Electronic ...	"Low"	"Replace Co...	441	[102,1038,19977,2004,10677,2000,1997,23229,1013,103]
8	"Things continue to tumble off of...	Mechanica...	"Low"	"Readjust M...	38	1x11 double
9	"Falling items from the conveyor ...	Mechanica...	"Low"	"Readjust M...	41	[102,4635,5168,2014,1997,16637,2954,5584,1013,103]
10	"The scanner reel is split, it will so...	Mechanica...	"Medium"	"Replace Co...	407	1x15 double
11	"Fuses are blown in the scanner."	Electronic ...	"High"	"Replace Co...	445	[102,19977,2016,2025,10677,2000,1997,26222,1013,103]
12	"Shrill cry from the scanner comp...	Electronic ...	"Low"	"Add to Wat...	77	[102,28350,5391,2014,1997,26222,3275,1013,103]
13	"Sorter controller neglects to int...	Software F...	"Low"	"Update Fir...	119	[102,4067,2122,11487,19047,2016,2001,8279,1013,103]
14	"Clunky sounds made by the sorti...	Mechanica...	"Low"	"Add to Wat...	63	1x12 double

OF CHINA

Load Data

- Partition the data into a training partition and a held-out partition for validation and testing. Specify the holdout percentage to be 20%.

```
cvp = cvpartition(data.Category,"Holdout",0.2);
dataTrain = data(training(cvp),:);     % 384 × 6
dataValidation = data(test(cvp),:);   % 96 × 6
```

- View the number of training and validation observations.

```
numObservationsTrain = size(dataTrain,1)   % 384
numObservationsValidation = size(dataValidation,1)   % 96
```

Load Data

- Extract the text data, labels, and encoded BERT tokens from the partitioned tables.

```
textDataTrain = dataTrain.Description;
textDataValidation = dataValidation.Description;

TTrain = dataTrain.Category;
TValidation = dataValidation.Category;

tokensTrain = dataTrain.Tokens;
tokensValidation = dataValidation.Tokens;
```

Load Data

- To check that you have imported the data correctly, visualize the training text data using a word cloud.

```
figure
wordcloud(textDataTrain);
title("Training Data")
```

Load Data

- To check that you have imported the data correctly, visualize the training text data using a word cloud.

```
figure
wordcloud(textDataTrain);
title("Training Data")
```

Prepare Data for Training

- Mini-batch queues require a single datastore that outputs both the predictors and responses. Create array datastores containing the training BERT tokens and labels and combine them using the |combine| function.

```
dsXTrain = arrayDatastore(tokensTrain,"OutputType","same");
dsTTrain = arrayDatastore(TTrain);
cdsTrain = combine(dsXTrain,dsTTrain);
```

- Create a combined datastore for the validation data using the same steps.

```
dsXValidation = arrayDatastore(tokensValidation,"OutputType","same");
dsTValidation = arrayDatastore(TValidation);
cdsValidation = combine(dsXValidation,dsTValidation);
```

Prepare Data for Training

- Mini-batch queues require a single datastore that outputs both the predictors and responses. Create array datastores containing the training BERT tokens and labels and combine them using the |combine| function.

```
dsXTrain = arrayDatastore(tokensTrain,"OutputType","same");
dsTTrain = arrayDatastore(TTrain);
cdsTrain = combine(dsXTrain,dsTTrain);
```

- Create a combined datastore for the validation data using the same steps.

```
dsXValidation = arrayDatastore(tokensValidation,"OutputType","same");
dsTValidation = arrayDatastore(TValidation);
cdsValidation = combine(dsXValidation,dsTValidation);
```

Prepare Data for Training

- Create a mini-batch queue for the training data. Specify a mini-batch size of 32 and preprocess the mini-batches using|preprocessPredictors|.

```
miniBatchSize = 32;
paddingValue = mdl.Tokenizer.PaddingCode;   % 1
maxSequenceLength = mdl.Parameters.Hyperparameters.NumContext;  % 512
mbqTrain = minibatchqueue(cdsTrain,1,...     % output -> 1
    "MiniBatchSize",miniBatchSize, ...
    "MiniBatchFcn",@(X) preprocessPredictors(X,paddingValue,maxSequenceLength));
mbqValidation = minibatchqueue(cdsValidation,1,...
    "MiniBatchSize",miniBatchSize, ...   % X -> 1×SequenceLength×BatchSize (1×15×32)
    "MiniBatchFcn",@(X) preprocessPredictors(X,paddingValue,maxSequenceLength));
```

Prepare Data for Training

- Convert the training sequences of BERT model tokens to a |N|-by-|embeddingDimension| array of feature vectors, where |N| is the number of training observations and |embeddingDimension| is the dimension of the BERT embedding.

```
featuresTrain = [];
reset(mbqTrain);
while hasdata(mbqTrain)
    X = next(mbqTrain);   % 1×15×32 dlarray
    features = bertEmbed(X,mdl.Parameters);  % 768×32 dlarray
    featuresTrain = [featuresTrain gather(extractdata(features))];
end
featuresTrain = featuresTrain.';  % 384×768 dlarray
```

Prepare Data for Training

- Convert the validation data to feature vectors using the same steps.

```
featuresValidation = [];
reset(mbqValidation);
while hasdata(mbqValidation)
    X = next(mbqValidation);   % 1×14×32 dlarray
    features = bertEmbed(X,mdl.Parameters);    % 768×32 dlarray
    featuresValidation = cat(2,featuresValidation,gather(extractdata(features)));
end
featuresValidation = featuresValidation.';   % 96×768 dlarray
```

Define Deep Learning Network

```
numFeatures = mdl.Parameters.Hyperparameters.HiddenSize;   % 768
layers = [
    featureInputLayer(numFeatures)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Specify Training Options

- Specify the training options using the |trainingOptions| function. Train with a mini-batch size of 64. Shuffle the data every epoch. Validate the network using the validation data. Display the training progress in a plot and suppress the verbose output.

```
opts = trainingOptions('adam',...
    "MiniBatchSize",64,...
    "ValidationData",{featuresValidation,dataValidation.Category},...
    "Shuffle","every-epoch", ...
    "Plots","training-progress", ...
    "Verbose",0);
```

Train & Test Network

- Train the network using the |trainNetwork| function.

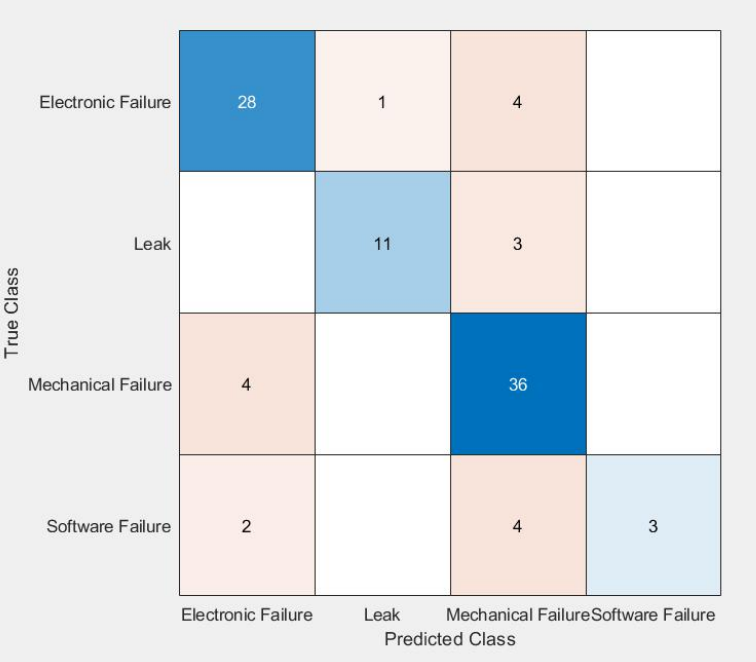
net = trainNetwork(featuresTrain,dataTrain.Category,layers,opts);

- Make predictions using the validation data and display the results in a confusion matrix.

YPredValidation = classify(net,featuresValidation);

figure

confusionchart(TValidation,YPredValidation)



Train & Test Network

- Calculate the validation accuracy.

```
accuracy = mean(dataValidation.Category == YPredValidation)   % 0.8125
```

Predict Using New Data

- Classify the event type of three new reports. Create a string array containing the new reports.

```
reportsNew = [ ...
    "Coolant is pooling underneath sorter."
    "Sorter blows fuses at start up."
    "There are some very loud rattling sounds coming from the assembler."];
```

- Tokenize the text data using the same steps as the training documents.

```
tokensNew = encode(tokenizer,reportsNew);
```

tokensNew

3x1 cell

	1
1	*1x11 double*
2	*1x11 double*
3	*1x15 double*

Predict Using New Data

- Pad the sequences of tokens to the same length using the |padsequences| function and pad using the tokenizer padding code.

```
XNew = padsequences(tokensNew,2,"PaddingValue",tokenizer.PaddingCode);
% 1×15×3 double
```

- Classify the new sequences using the trained model.

```
featuresNew = bertEmbed(XNew,mdl.Parameters)';  % 3×768 dlarray
featuresNew = gather(extractdata(featuresNew));   % 3×768 single
labelsNew = classify(net,featuresNew)
%     Leak
%     Electronic Failure
%     Mechanical Failure
```

Self Attention Function

```
function [A, present] = attention(X, past, weights, hyperParameters, nvp)
% attention   Full Multi-head Attention
%   [A, present] = attention(X, past, weights, hyperParameters) computes a
%   multi-head attention block on X
% Inputs:
%      X              - A (numFeatures*numHeads)-by-numInputSubwords-by-numObs
%                       input array.
%   Outputs:
%      A              - A (numFeatures*numHeads)-by-numInputSubwords-by-numObs
%                       output array.
```

```
% Use a fully connected layer to generate queries, keys and values from the input.
C = transformer.layer.convolution1d( X, ...  % 768×15×32 dlarray
    weights.attn_c_attn_w_0, ...   % 2304×768 dlarray
    weights.attn_c_attn_b_0 );     % 2304×1 dlarray
% 2304×15×32 dlarray
% Split the results into Q (Query), K (Keys) and V (Values).
splitSize = size(C,1)/3;  % 768
Q = C(1:splitSize,:,:);     % 768×15×32 dlarray
K = C((splitSize+1):(2*splitSize),:,:);      % 768×15×32 dlarray
V = C((2*splitSize+1):(3*splitSize),:,:);   % 768×15×32 dlarray

% Split heads
Q = iSplitHeads(Q, splitSize, hyperParameters.NumHeads);  % 64×15×12×32 dlarray
K = iSplitHeads(K, splitSize, hyperParameters.NumHeads);    % 64×15×12×32 dlarray
V = iSplitHeads(V, splitSize, hyperParameters.NumHeads);    % 64×15×12×32 dlarray
```

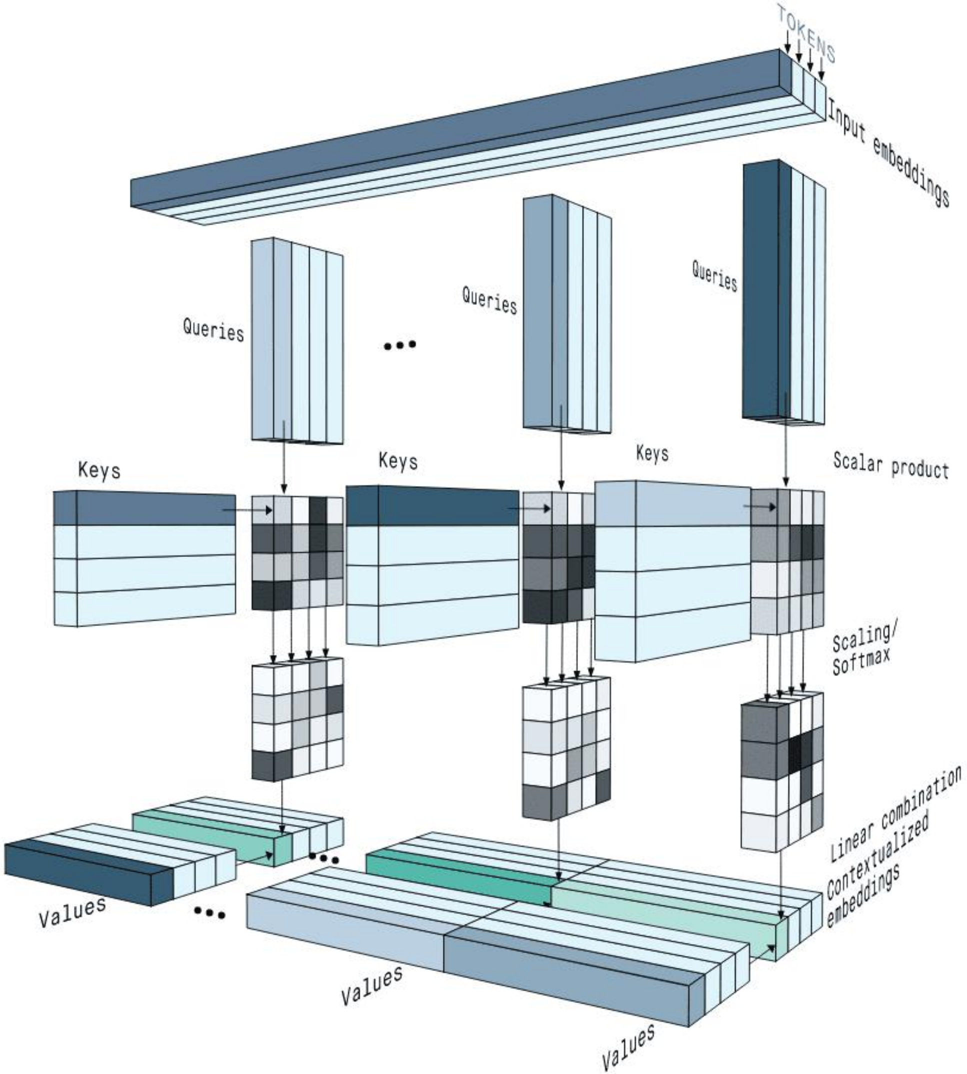
```
A = transformer.layer.multiheadAttention(Q,K,V,'CausalMask',nvp.CausalMask,
                                         'Dropout',nvp.Dropout,
                                         'InputMask',nvp.InputMask);
% 64×15×12×32 dlarray

A = iMergeHeads(A);
% 768×15×32 dlarray

A = transformer.layer.convolution1d( A, ...   % 768×15×32 dlarray
    weights.attn_c_proj_w_0, ...               % 768×768 dlarray
    weights.attn_c_proj_b_0 );                 % 768×1 dlarray
end
% 768×15×32 dlarray
```

Multi-Head Self Attention Function

```
function A = multiheadAttention(Q, K, V,nvp)
% multiheadAttention   Multi-head Attention
%
%   A = multiheadAttention(Q, K, V) computes scaled dot product attention
%   for multiple attention heads. The function computes the attention for multiple
%   attention heads at once for efficiency. Q is a collection of query
%   matrices, K is a collection of key matrices and V is a collection of
%   value matrices. The output A is a collection of attention matrices.
```

```
% We compute attention weights by taking the product between Q and K
% matrices. W is numAllSubWords-by-numInputSubWords-by-numHeads. Each
% element of W is the dot product of a query vector from Q and a key vector
% from K.

W = dlmtimes(permute(K, [2 1 3 4]), Q);   % 64×15×12×32 dlarray
% 15×15×12×32 dlarray

% Divide by square root of d
W = W./sqrt(size(Q,1));

% Apply masking
W = transformer.layer.maskAttentionWeights(W,'CausalMask',nvp.CausalMask,'InputMask',nvp.InputMask);
% 15×15×12×32 dlarray
```

```
% Apply softmax
W = softmax(W, 'DataFormat', 'CTUB');
% 15×15×12×32 dlarray

% Apply dropout
W = transformer.layer.dropout(W,nvp.Dropout);

% We compute the attention by taking products between the attention weights
% W and V. A is numFeatures-by-numInputSubWords-by-numHeads. One
% interpretation of A is that it is the expected value of V according to
% the probability distribution given by W.
A = dlmtimes(V, W); % V: 64×15×12×32 dlarray   W: 15×15×12×32 dlarray
% 64×15×12×32 dlarray

end
```


TOKENS
Input embeddings
Queries
Queries
Queries
Keys
Keys
Keys
Scalar product
Scaling/
Softmax
Linear combination
Contextualized
embeddings
Values
Values
Values