ENGO 697

Remote Sensing Systems and Advanced Analytics

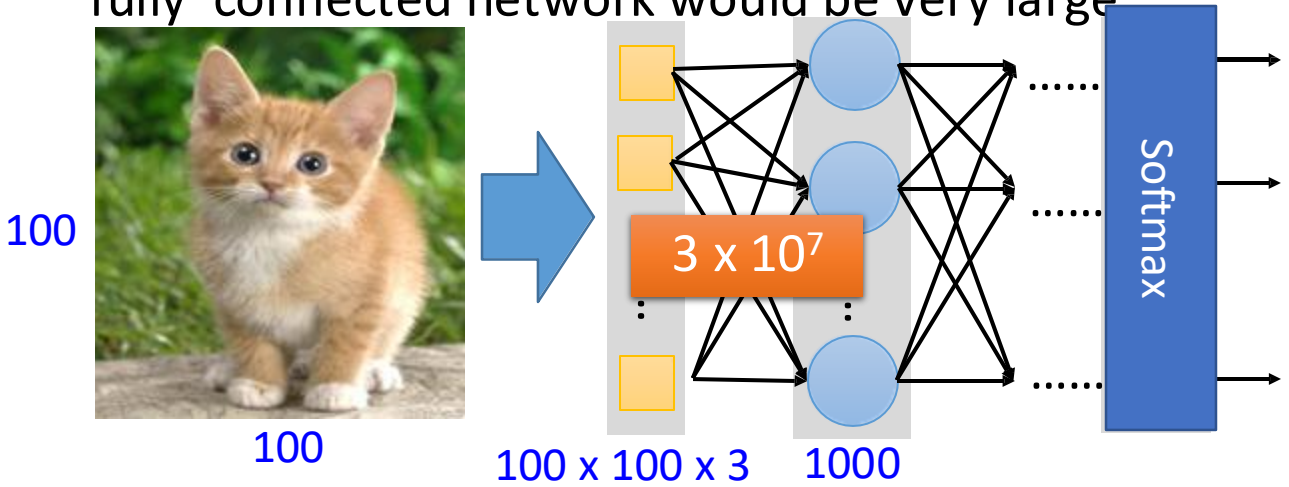
Session 10: CNN, Transformer, Graph neural networks, and
Neural Radiative Field (NeRF)

Dr. Linlin (Lincoln) Xu
Linlin.xu@ucalgary.ca
Office: ENE 221

	(1) Direct inversion	(2) LUT approach	(3) Numerical Approach	(4) Simulation & ML	(5) ML	**(6) DL**
f(.) is known	yes	yes	yes	yes	yes	**yes**
f(.) is partially known, i.e., form known, but with some unknown parameters U	no	no	Yes, estimate X and U together	no	no	**no**
f(.) unknown, (X,Y) known	no	no	no	no	yes	**yes**
f(.) unknown, (X,Y) unknown	no	no	no	no	no	**no**
If both f(.) and (X,Y) known, can accommodate both?	no	yes?	Yes? Use (X,Y) to estimate parameters in f(.)	yes?	Yes, use both simulated and observed data	**Yes, use both simulated and observed data**
Can use prior information? e.g., spatial prior and value prior	no	Yes? Use value prior for sampling	Yes? Use value prior of X in Bayesian estimation	Yes, Use value prior in sampling and spatial prior in Random fields	Yes, spatial prior in Random field approaches	**Yes, similar to ML**
Advantages	Knowledge -driven; Simple, easy	Knowledge-driven; Intuitive, easy, discrete fitting;	Knowledge-driven; estimate U; Efficient for simple f(.) in convex problems	Knowledge-driven; flexible; continuous fitting; good inter/extrapolation; faster than LUT	Data-driven; flexible; Classic;	**Strong modeling capability; automatic feature learning;**
Disadvantages	Unrealistic; rely on simple f(.)	Sensitive to accuracy of f(.), similarity metrics, sampling density and range; slow if LUT is large; bad for extrapolation;	Rely on efficiency of nonlinear solver; Slow; Local optimum;	Overfitting and underfitting risk to simulated data; difficult model selection; Sensitive to accuracy of f(.), similarity metrics, sampling density and range;	Weak modeling capability; Rely on "good" engineered features; Black-box; Overfitting, underfitting; Feature and model selection is difficult and slow	**Overfitting and underfitting; Black-box;**

Why CNN for Image?

- When processing image, the first layer of fully connected network would be very large



Can the fully connected network be simplified by considering the properties of image

Why CNN for Image

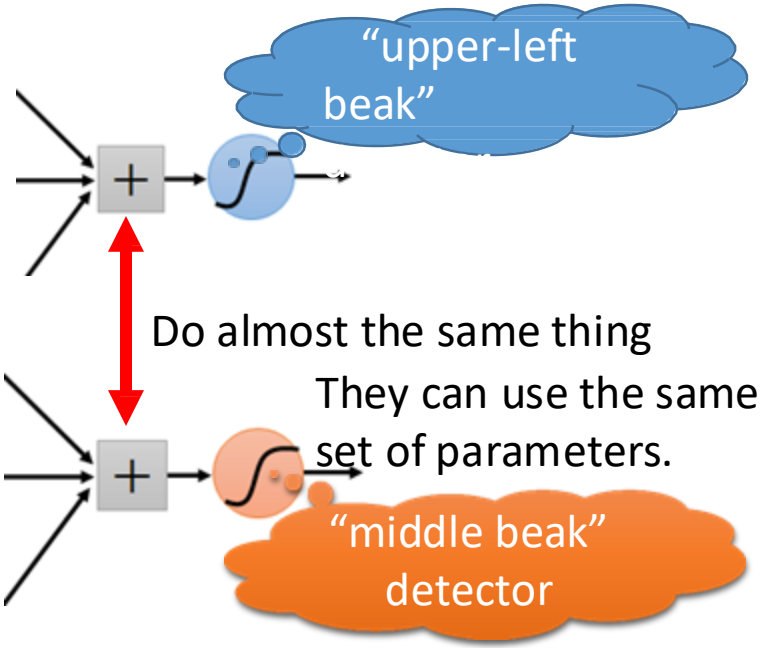
- Some patterns are much smaller than the whole image

A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less

Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- Subsampling the pixels will not change the object bird

subsampling

bird

We can subsample the pixels to make image smaller

Less parameters for the network to process the image

Three Steps for Deep Learning

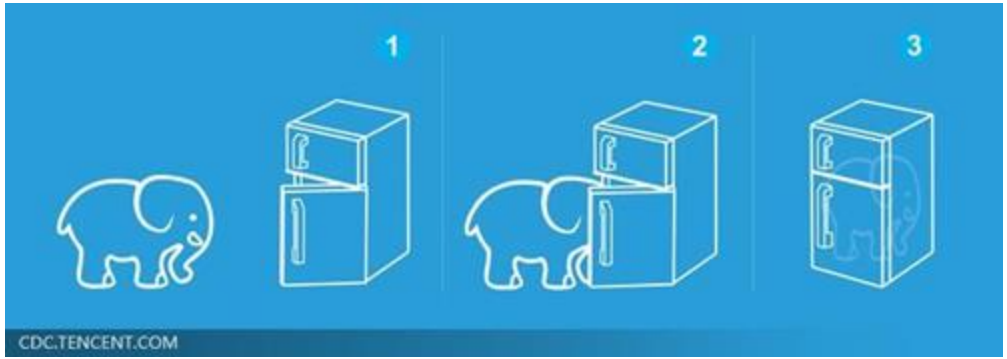
Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

Deep Learning is so simple

……



The whole CNN

cat dog ……

Fully Connected Feedforward

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

The whole CNN

Property 1

- Some patterns are much smaller than the whole

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

The whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

Fully Connected Feedforward

cat dog ……

Convolutional Layer

Consider channel = 1
(black and white image)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

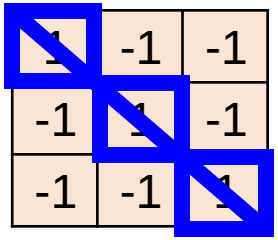
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

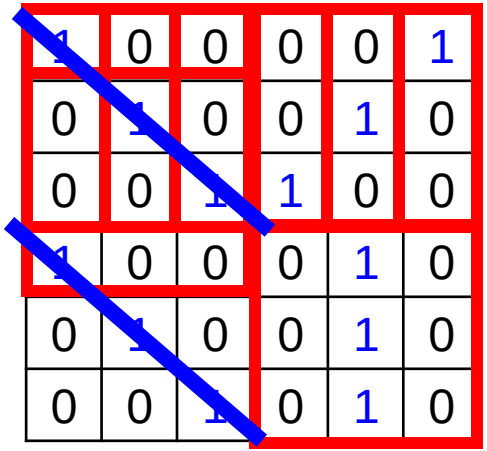
(The values in the filters are unknown parameters.)

Convolutional Layer

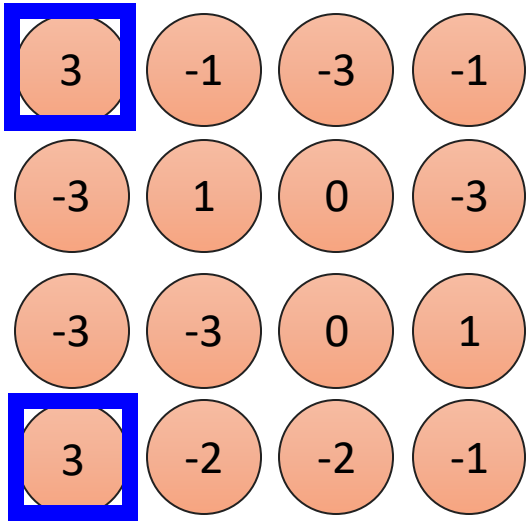


Filter 1

stride=1



6 x 6 image



Convolutional Layer

-1	1	-1
-1	1	-1
-1	1	-1

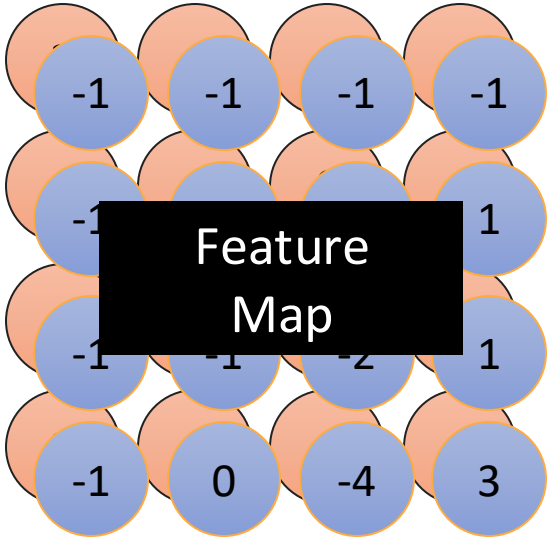
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Do the same process for every filter

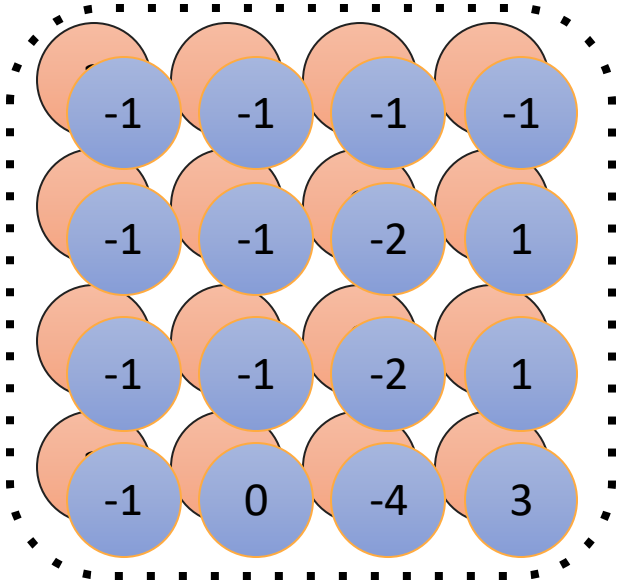


Convolutional Layer

64 filters

Convolution

Convolution



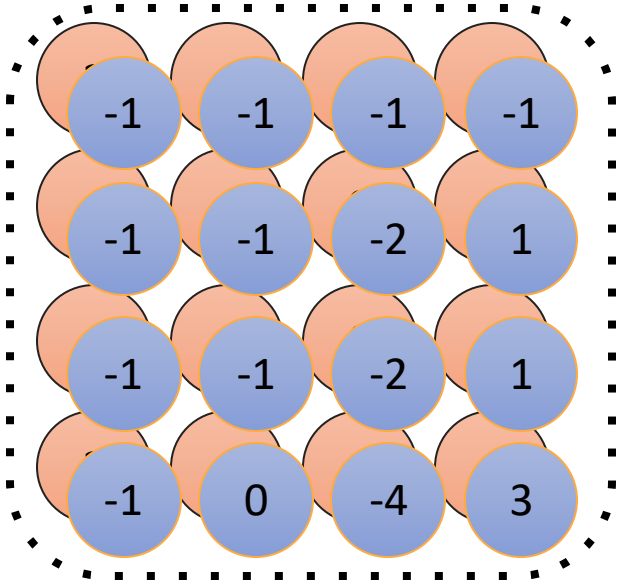
"Image" with 64 channels

Multiple Convolutional Layers

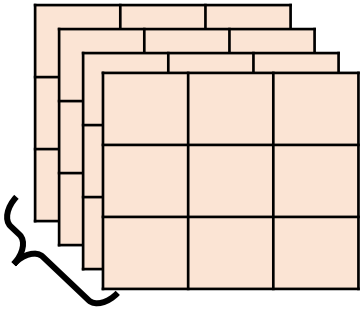
64 filters

Convolution

Convolution



"Image" with 64 channels



Filter:
3 x 3 x 64

64

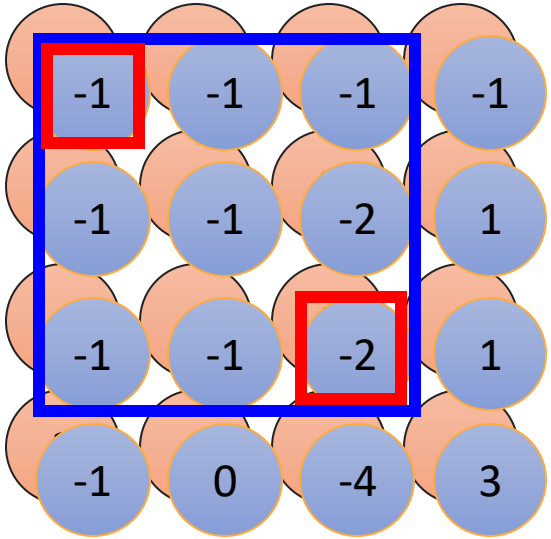
Multiple Convolutional Layers

64 filters

Convolution

Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0



The whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

Fully Connected Feedforward

cat dog ……

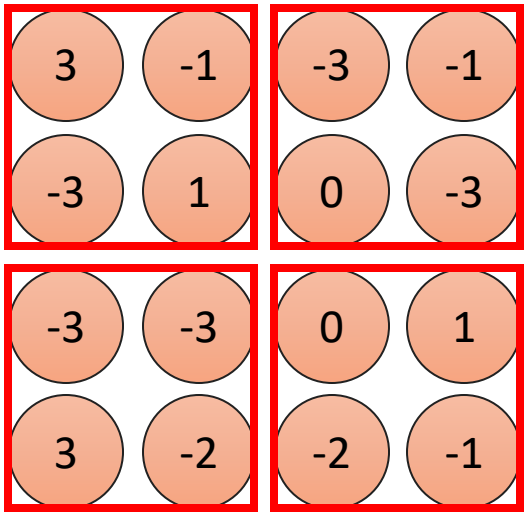
Pooling – Max Pooling

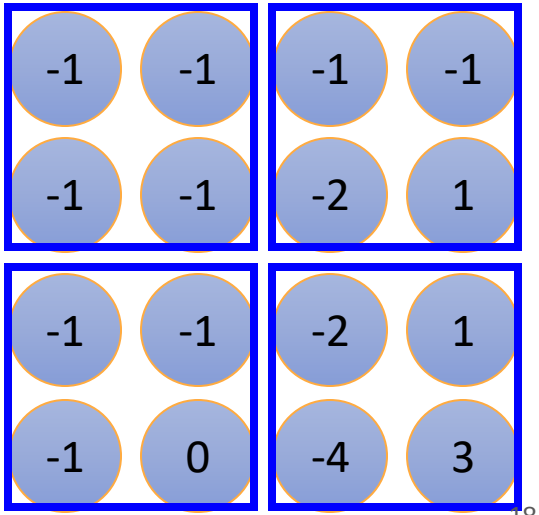
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Convolutional Layers + Pooling

Repeat

Convolution

Pooling

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

"Image" with 64 channels

-1	1
0	3

The whole CNN

Convolution

Pooling

Convolution

Pooling

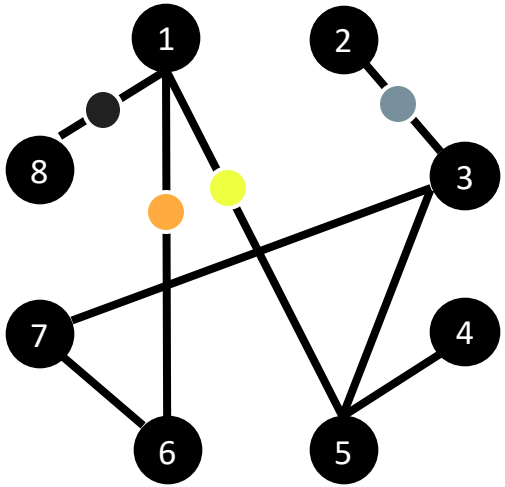
Flatten

Fully Connected Layers

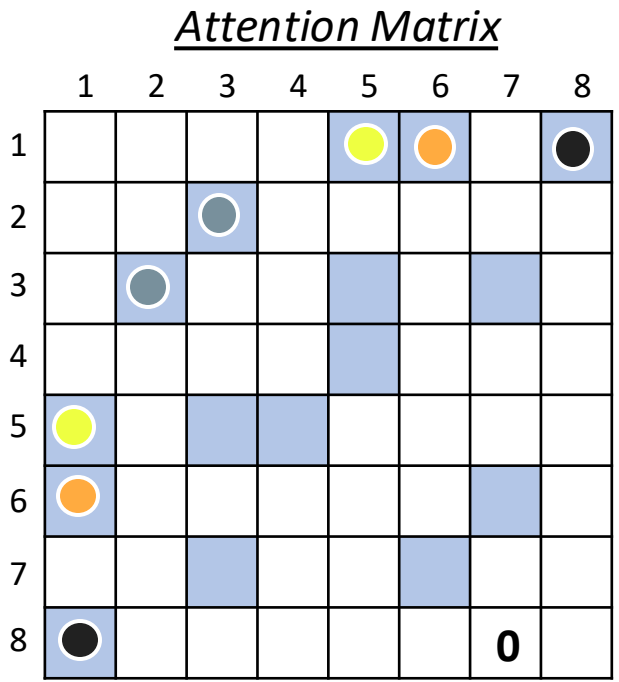
softmax

cat dog ……

Self-attention for Graph



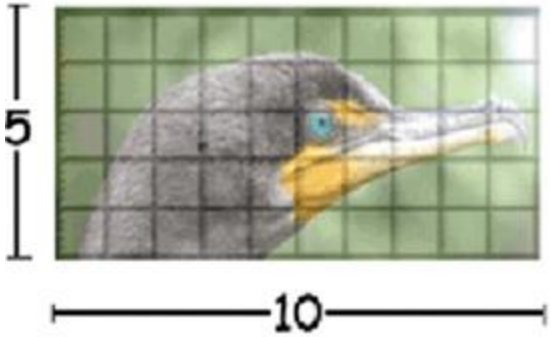
Consider **edge**: only attention to connected nodes



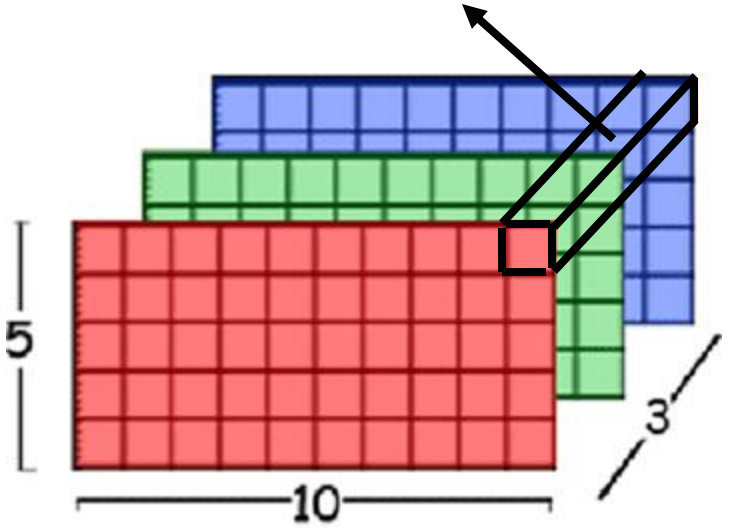
This is one type of **Graph Neural Network (GNN)**.

Self-attention for Image

An **image** can also be considered as a **vector set**.

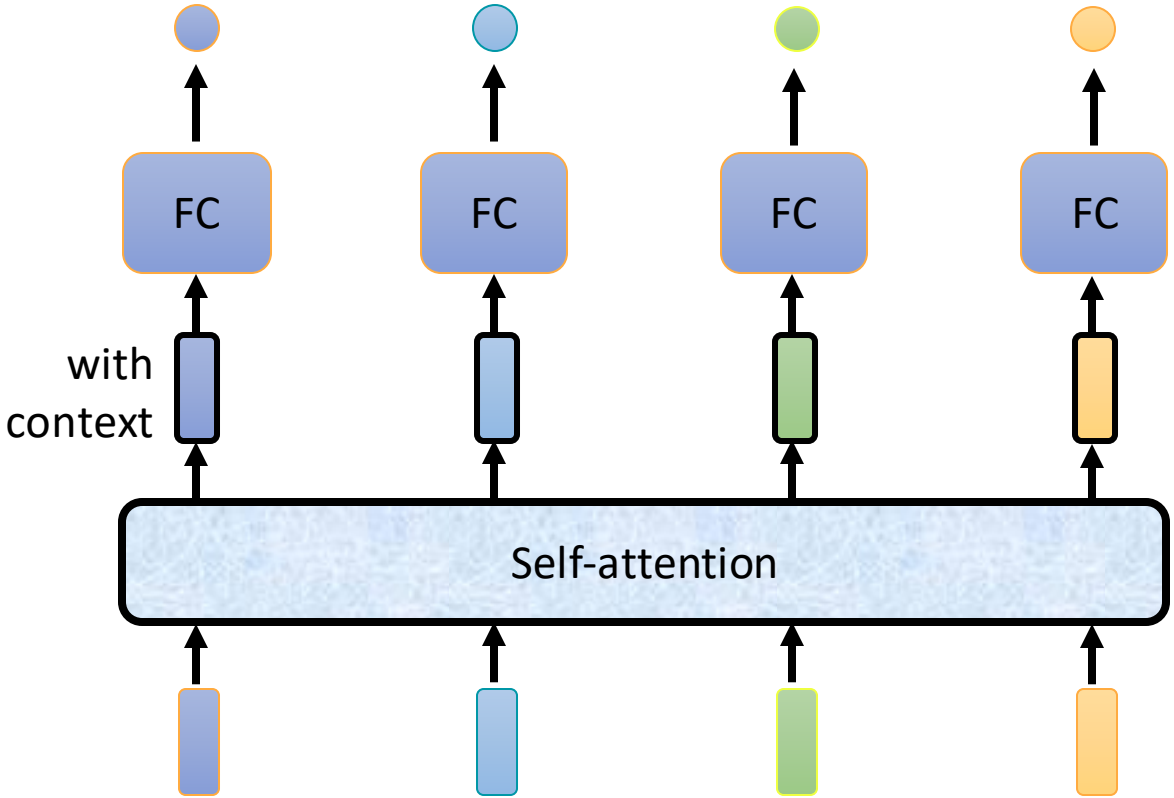


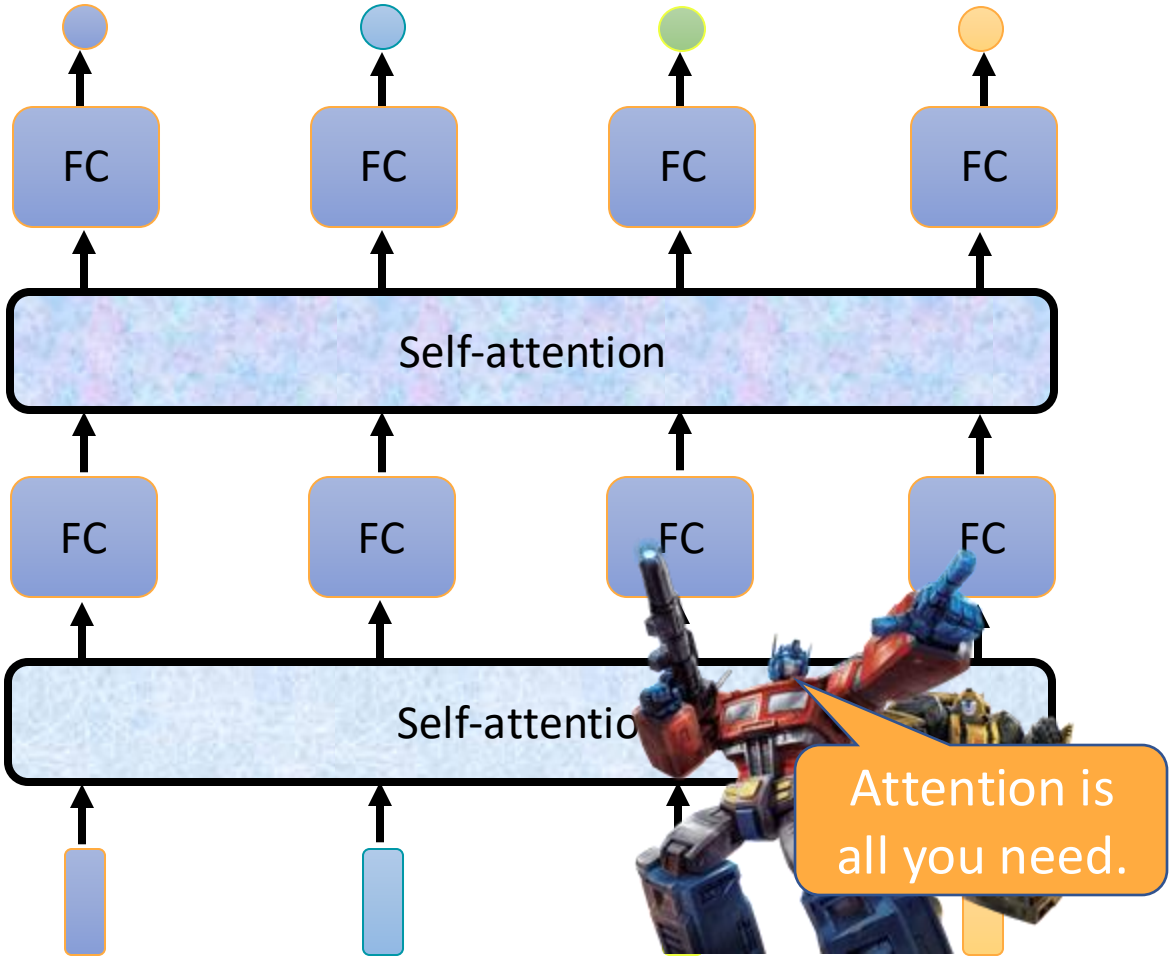
This is a vector.



Source of image: https://www.researchgate.net/figure/Color-image-representation-and-RGB-matrix_fig15_282798184

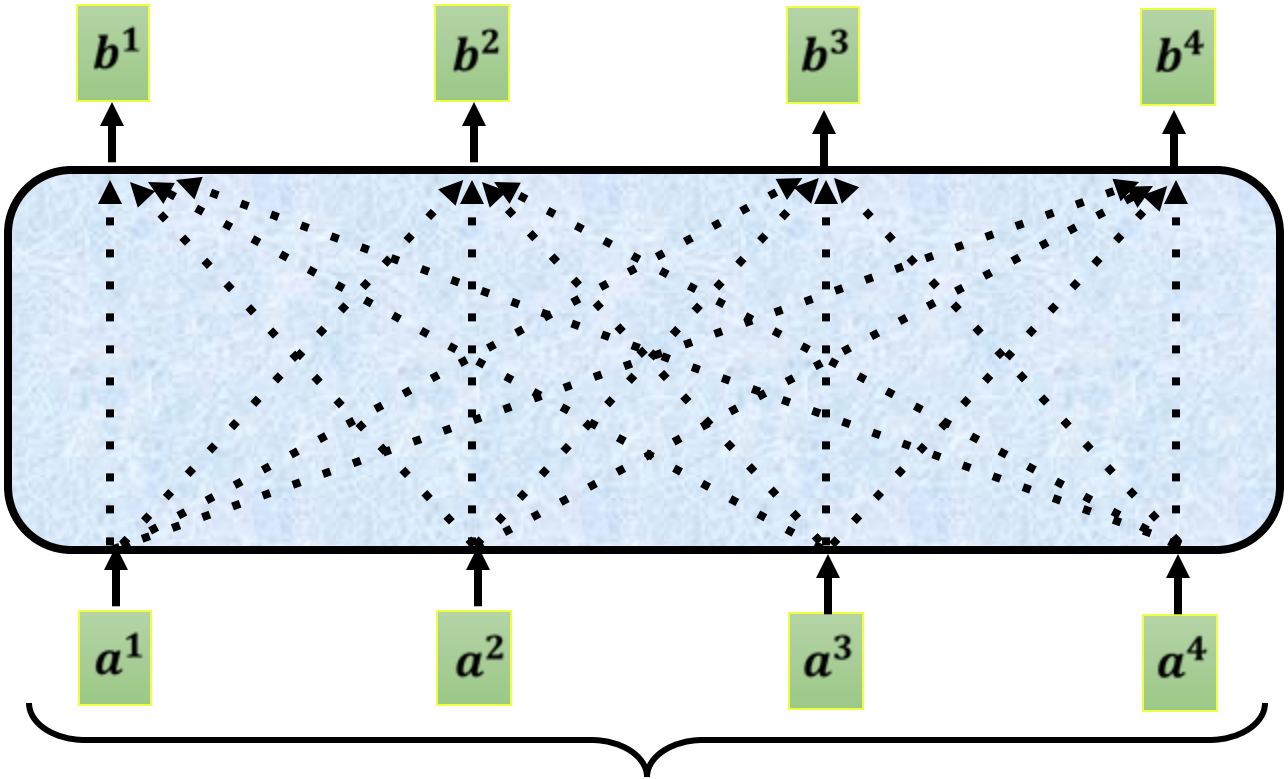
Self-attention





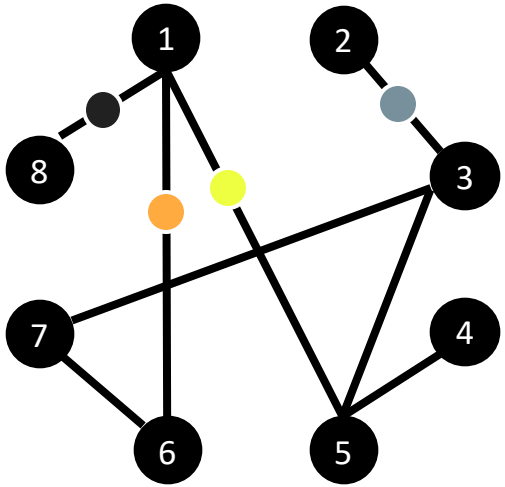
https://arxiv.org/abs/1706.03762

Self-attention

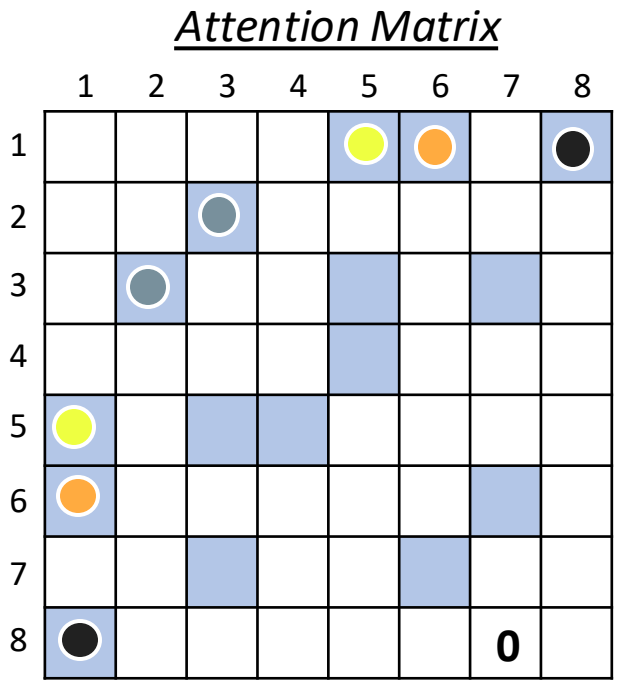


Can be either **input** or **a hidden layer**

Self-attention for Graph

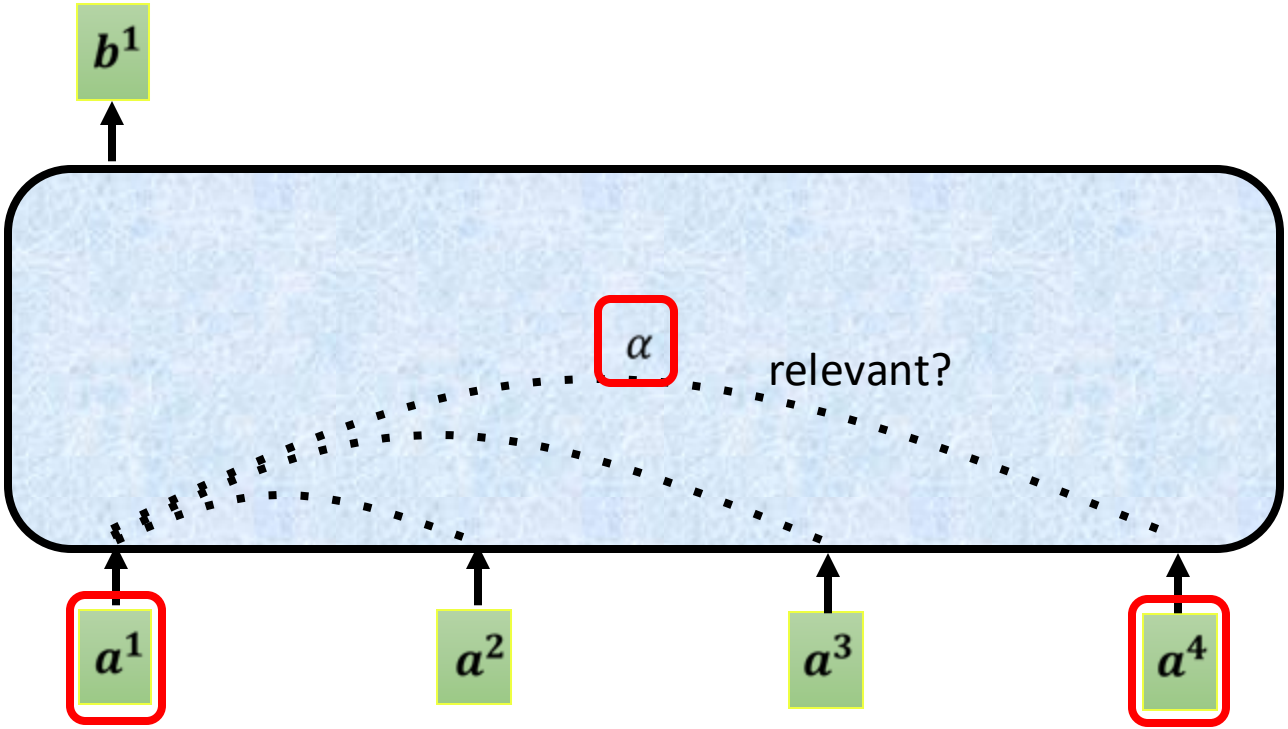


Consider **edge**: only attention to connected nodes



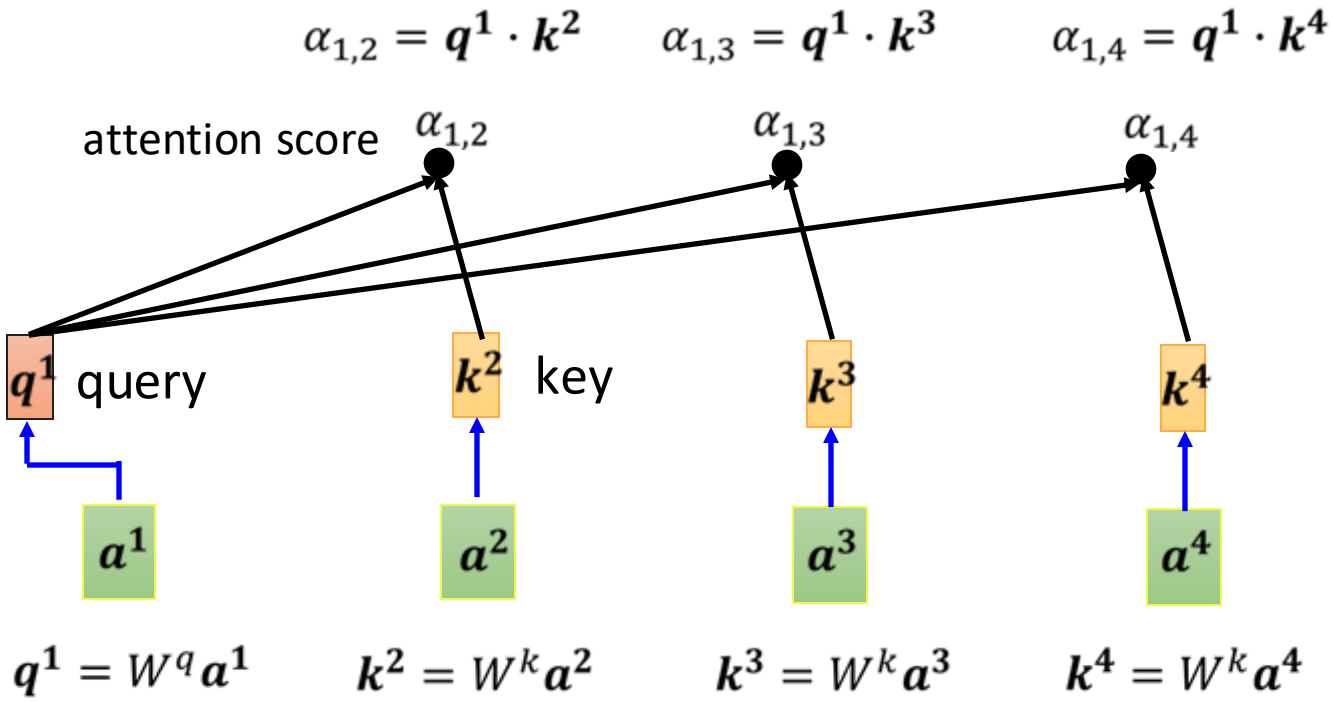
This is one type of **Graph Neural Network (GNN)**.

Self-attention



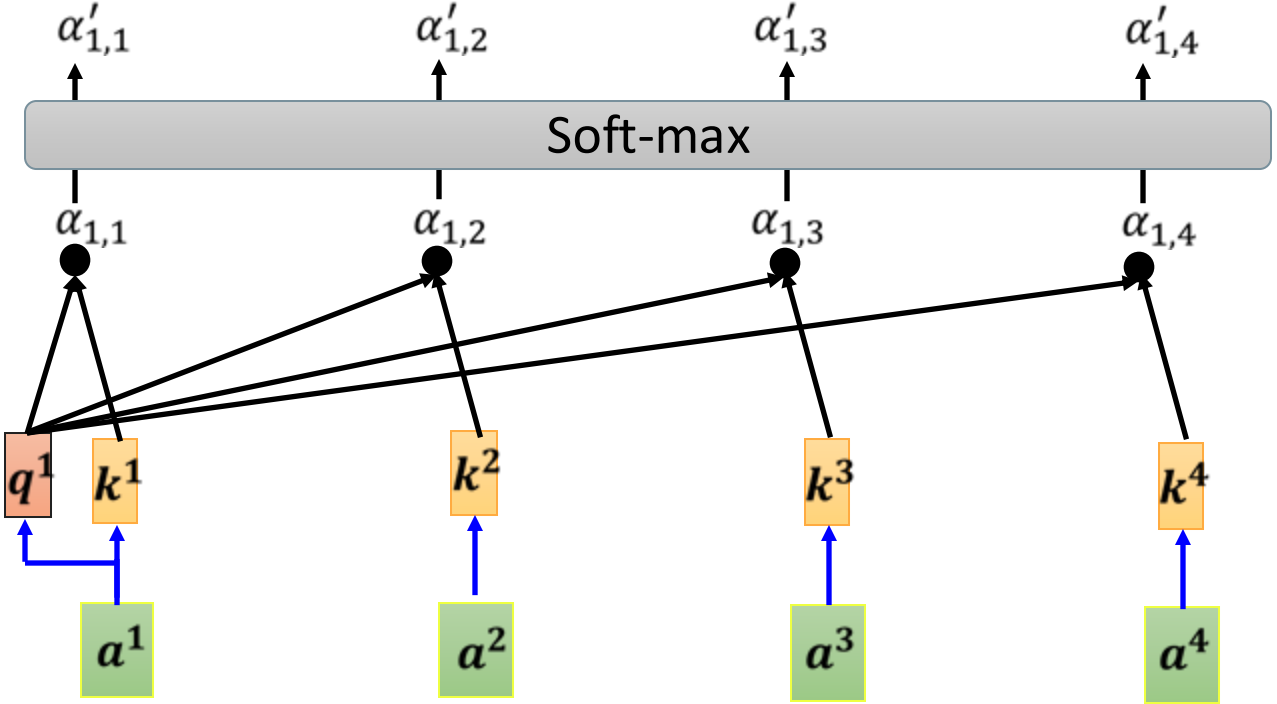
Find the relevant vectors in a sequence

Self-attention



Self-attention

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



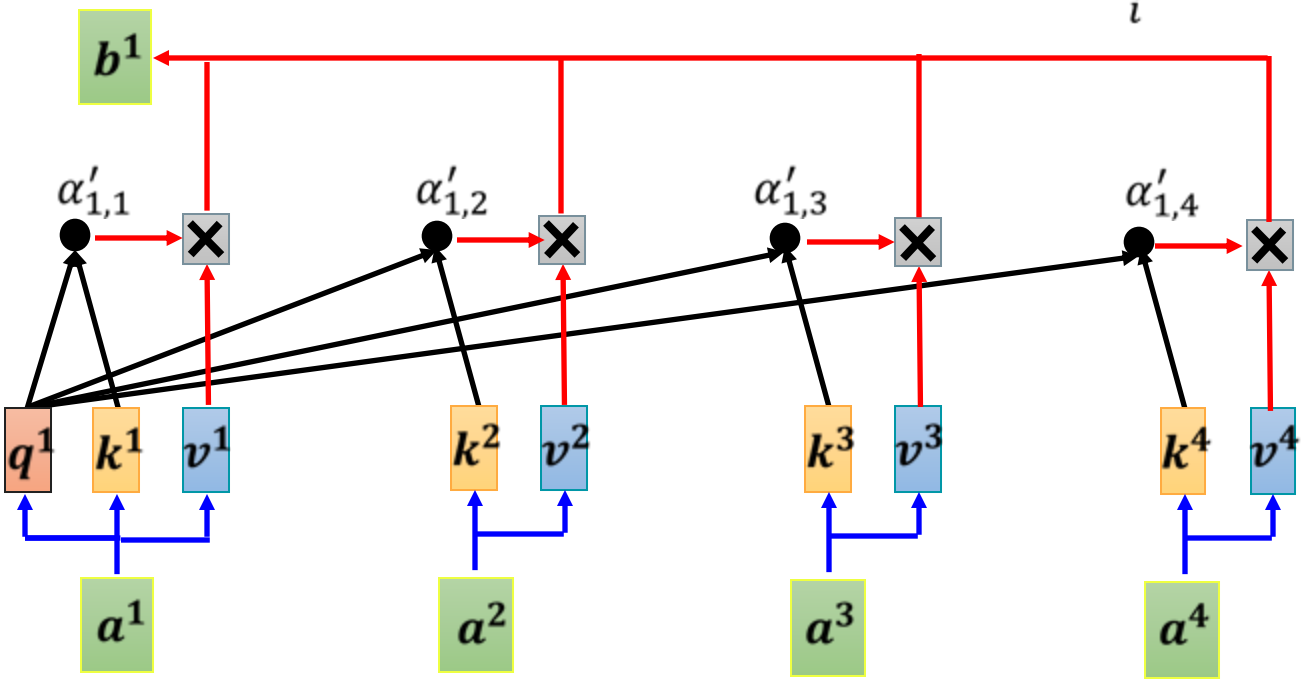
$q^1 = W^q a^1$ $k^2 = W^k a^2$ $k^3 = W^k a^3$ $k^4 = W^k a^4$

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

Self-attention

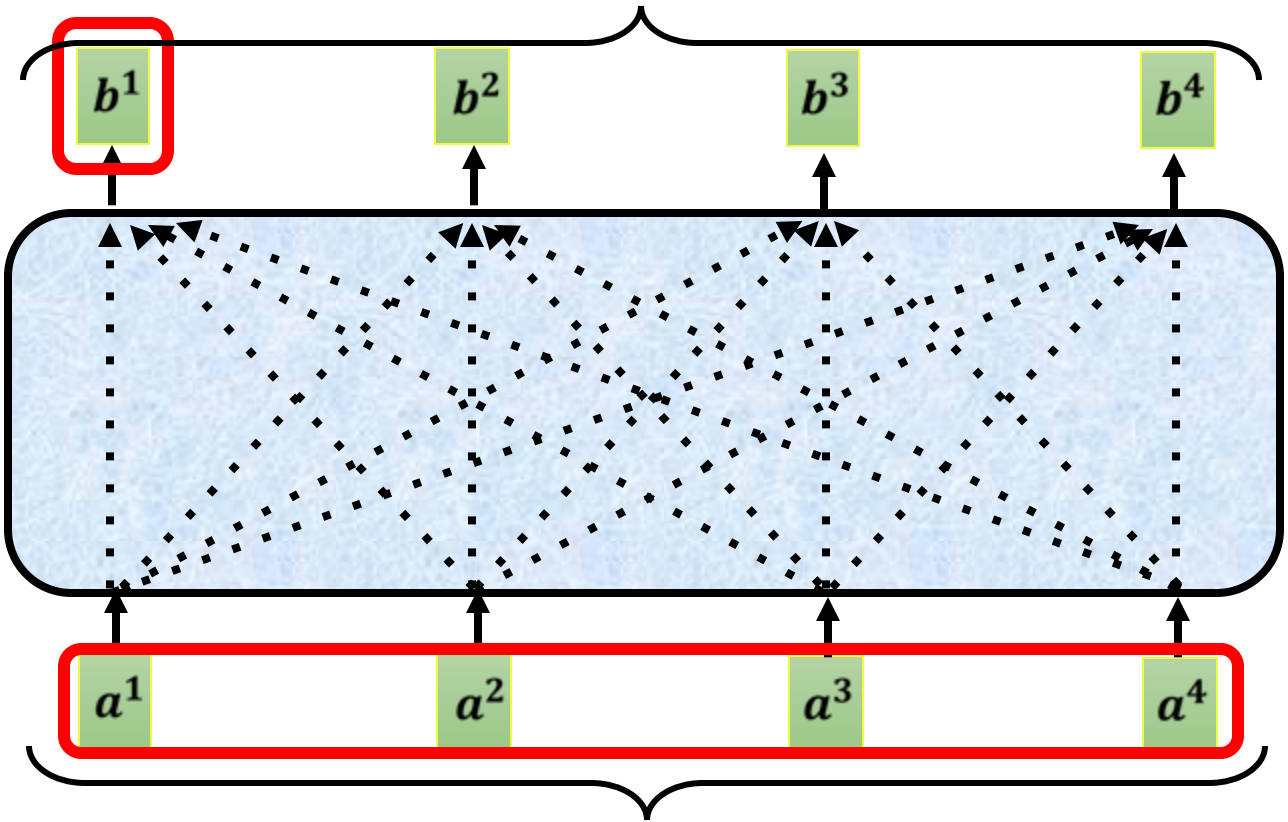
Extract information based on attention scores

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



$v^1 = W^v a^1$ $v^2 = W^v a^2$ $v^3 = W^v a^3$ $v^4 = W^v a^4$

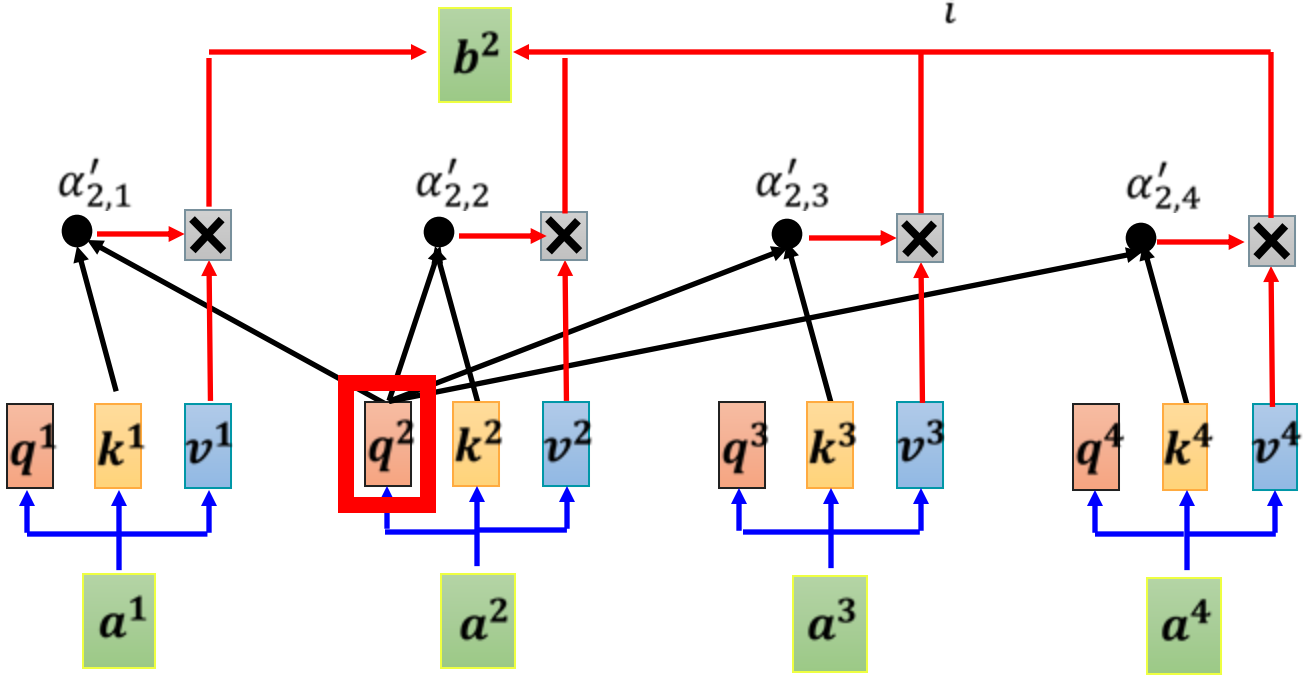
Self-attention



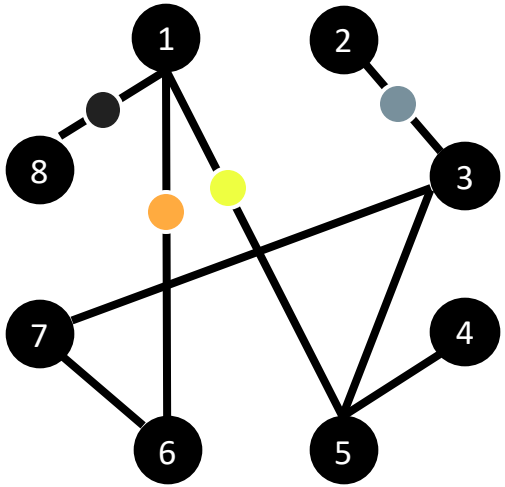
Can be either **input** or **a hidden layer**

Self-attention

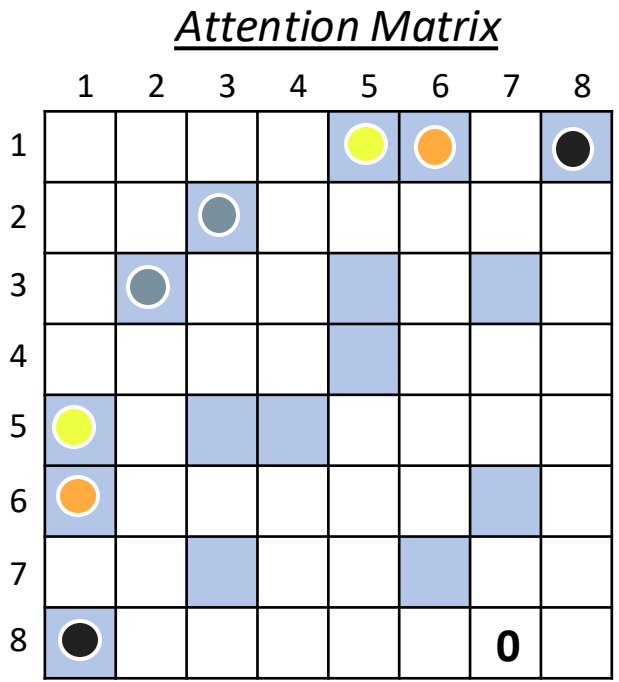
$$b^2 = \sum_i \alpha'_{2,i} v^i$$



Self-attention for Graph



Consider **edge**: only attention to connected nodes



This is one type of **Graph Neural Network (GNN)**.

Self-attention

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

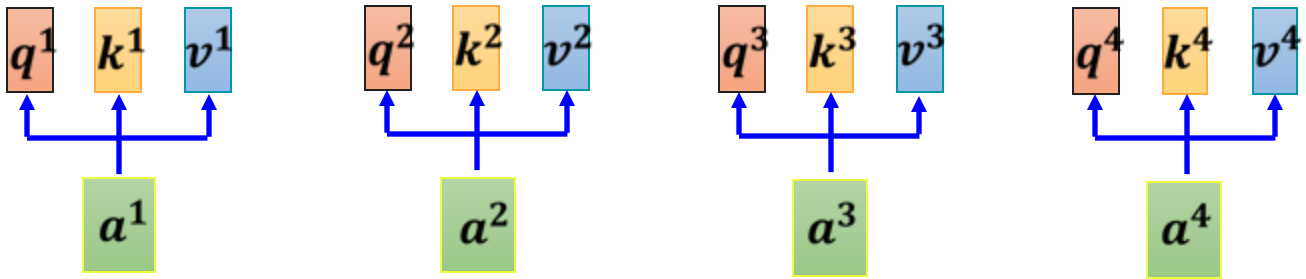
$$\begin{matrix} q^1 & q^2 & q^3 & q^4 \\ Q \end{matrix} = W^q \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$

$$k^i = W^k a^i$$

$$\begin{matrix} k^1 & k^2 & k^3 & k^4 \\ K \end{matrix} = W^k \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$

$$v^i = W^v a^i$$

$$\begin{matrix} v^1 & v^2 & v^3 & v^4 \\ V \end{matrix} = W^v \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$

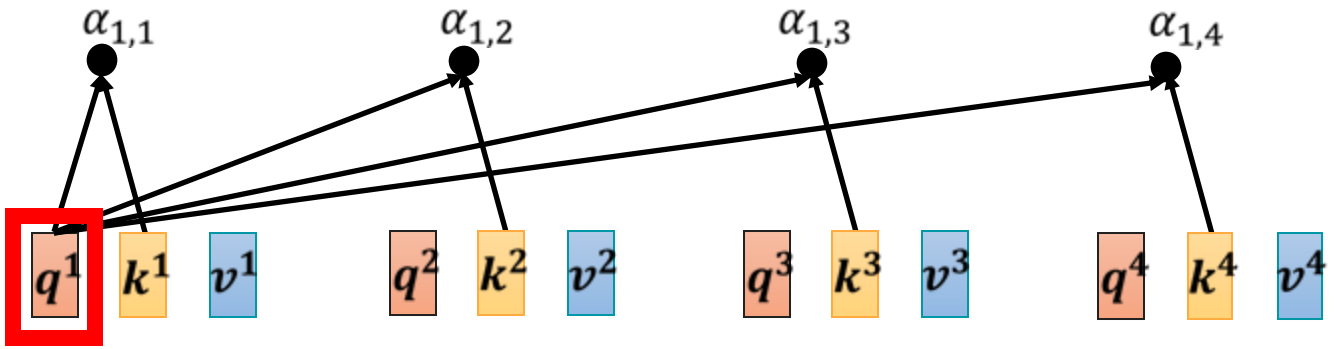


Self-attention

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

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

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

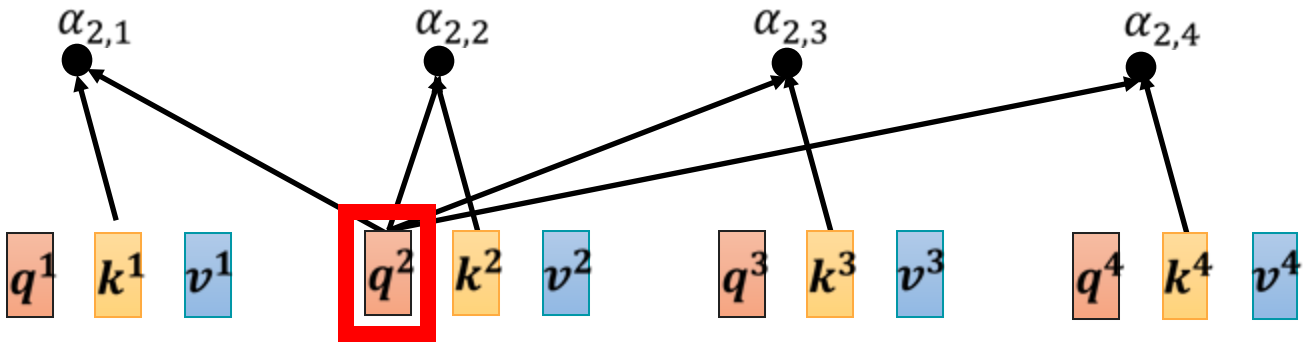


Self-attention

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

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

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



$\begin{bmatrix}\alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1}\\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2}\\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3}\\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4}\end{bmatrix} = A'$

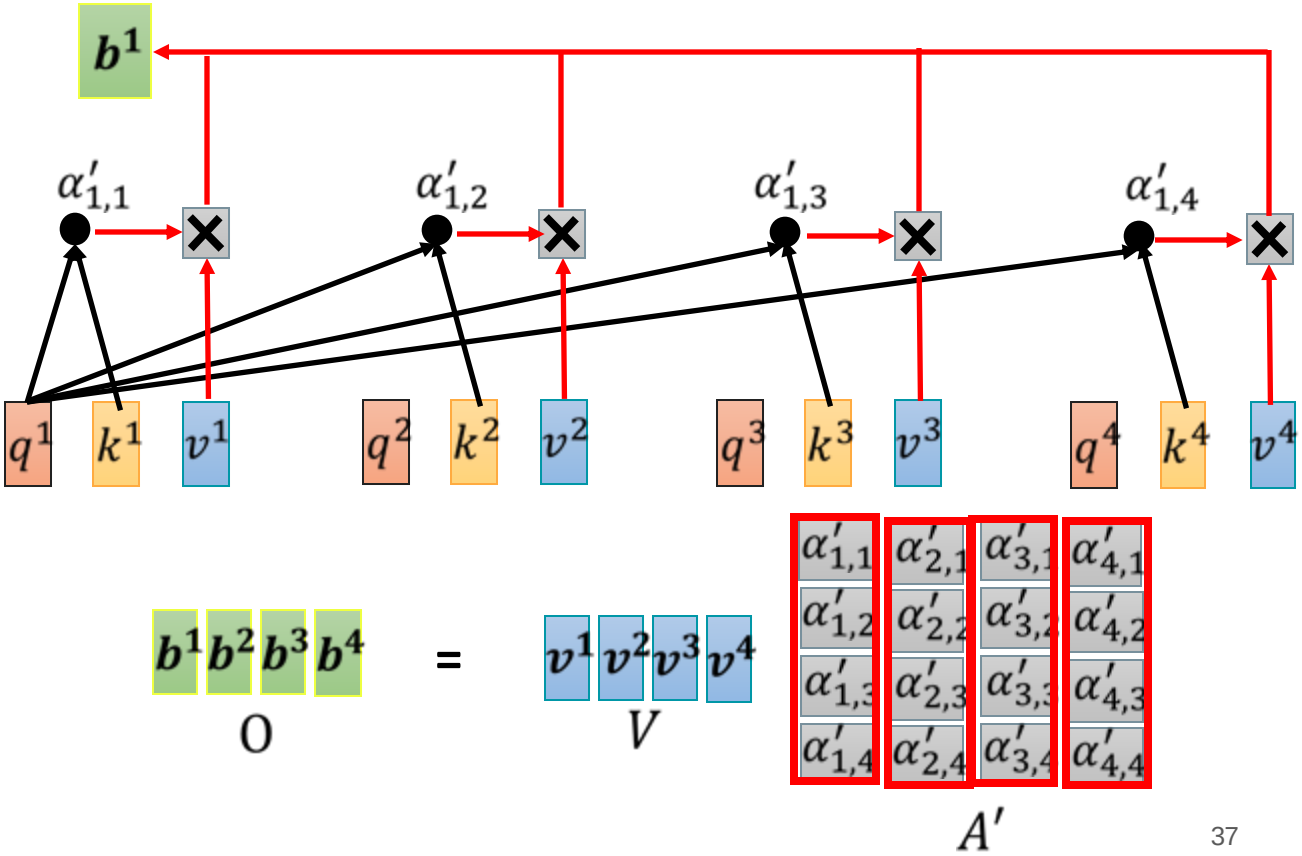
softmax

$\begin{bmatrix}\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1}\\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2}\\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3}\\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4}\end{bmatrix} = A$

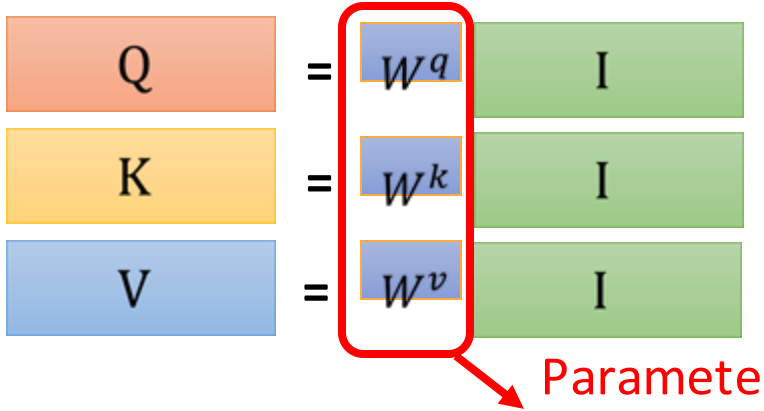
$\begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} = K^T$

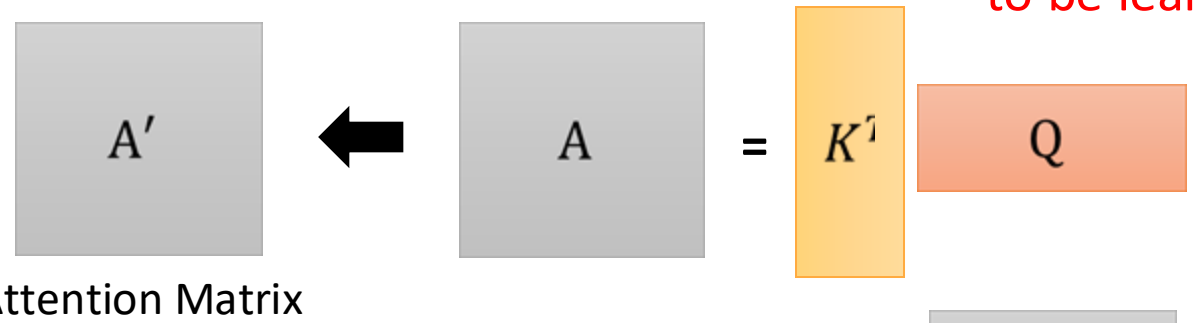
$\begin{bmatrix}q^1 & q^2 & q^3 & q^4\end{bmatrix} = Q$

Self-attention

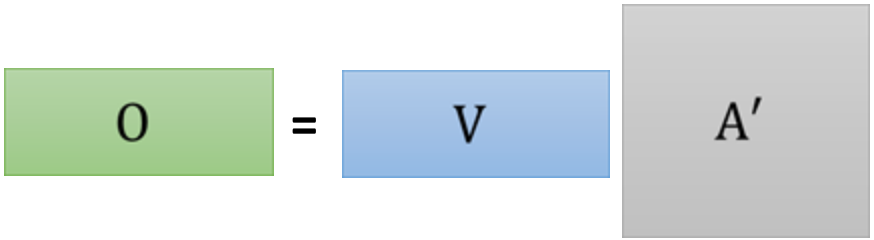


Self-attention

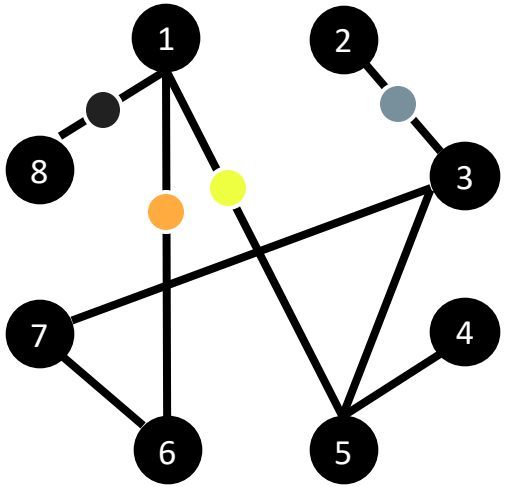




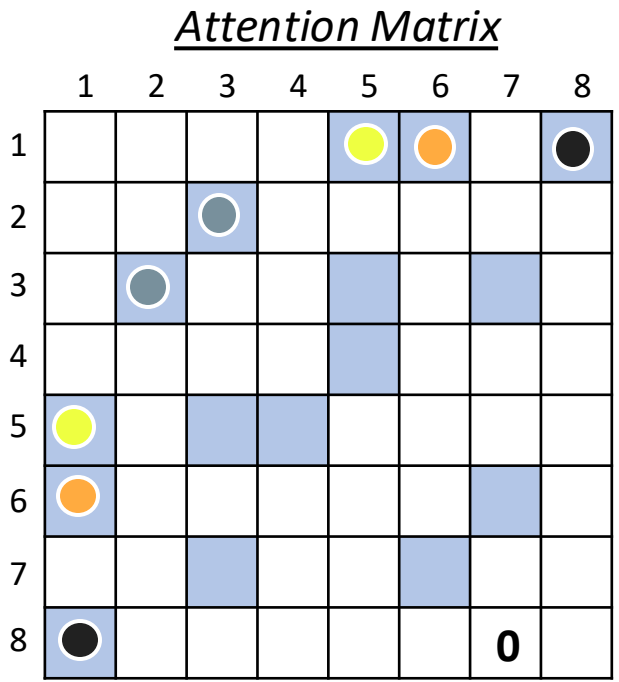
Attention Matrix



Self-attention for Graph



Consider **edge**: only attention to connected nodes



This is one type of **Graph Neural Network (GNN)**.

Many applications …

Transformer

https://arxiv.org/abs/1706.03762

BERT

https://arxiv.org/abs/1810.04805

Widely used in Natural Langue Processing (NLP)!

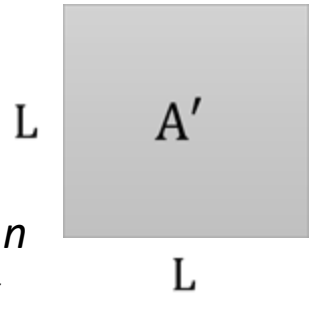
https://arxiv.org/abs/1910.12977

Self-attention for Speech

Speech is a very long vector sequence.

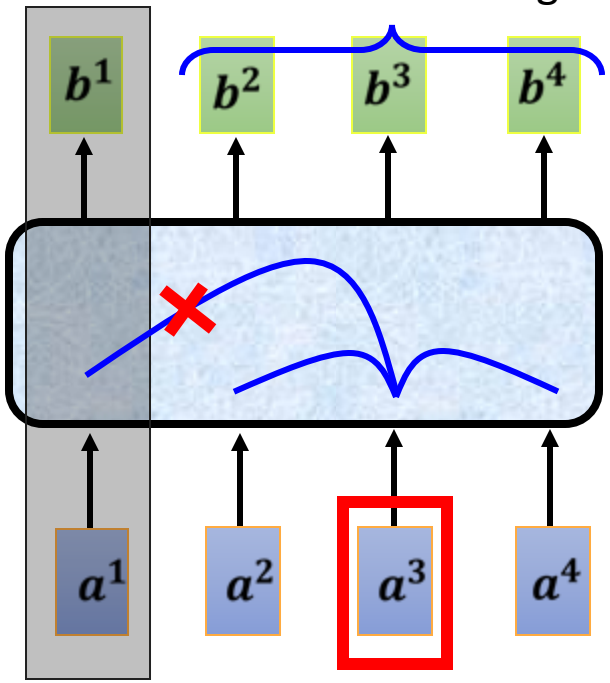
10ms

If input sequence is length L



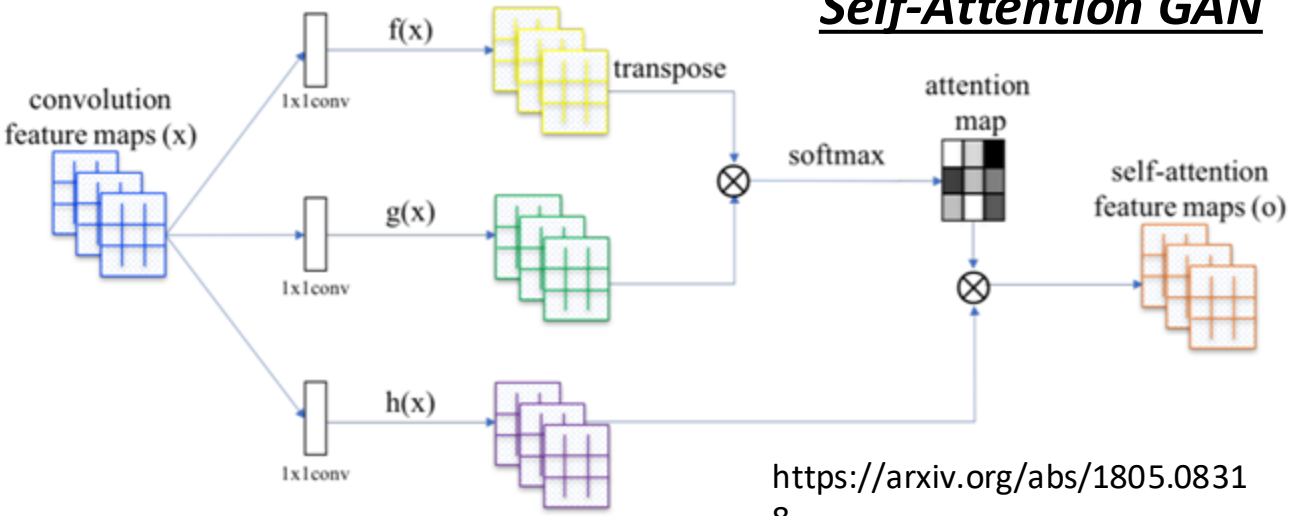
Attention Matrix

Attention in a range



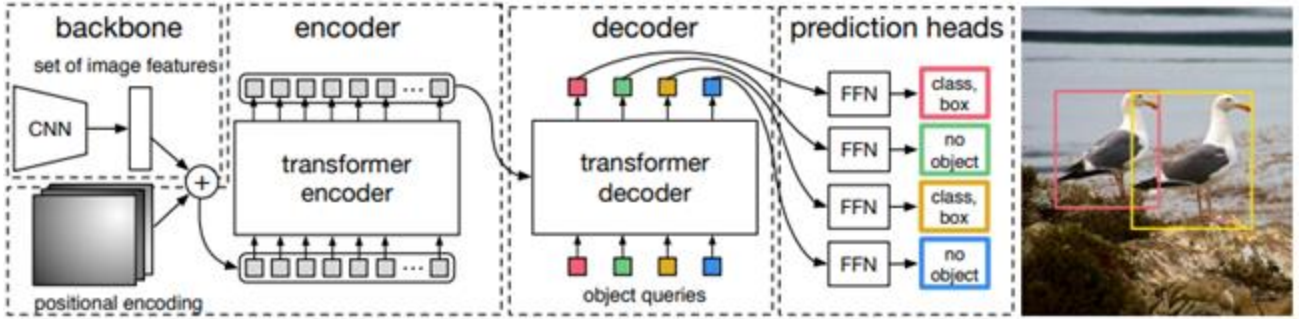
Truncated Self-attention

Self-Attention GAN



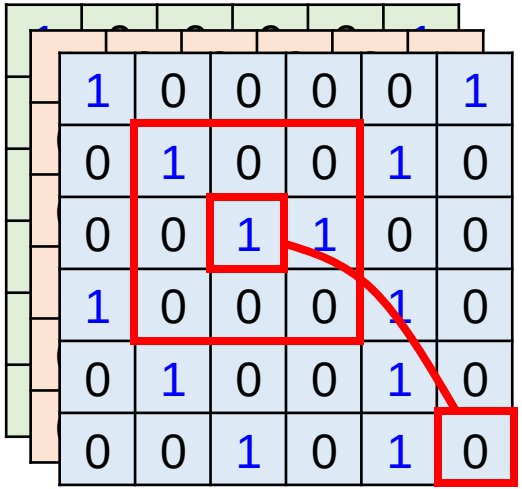
https://arxiv.org/abs/1805.08318

DEtection Transformer (DETR)



https://arxiv.org/abs/2005.12872

Self-attention v.s. CNN



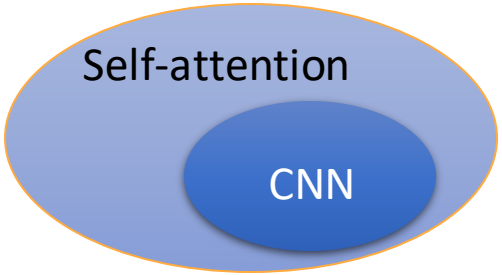
CNN: self-attention that can only attends in a receptive field

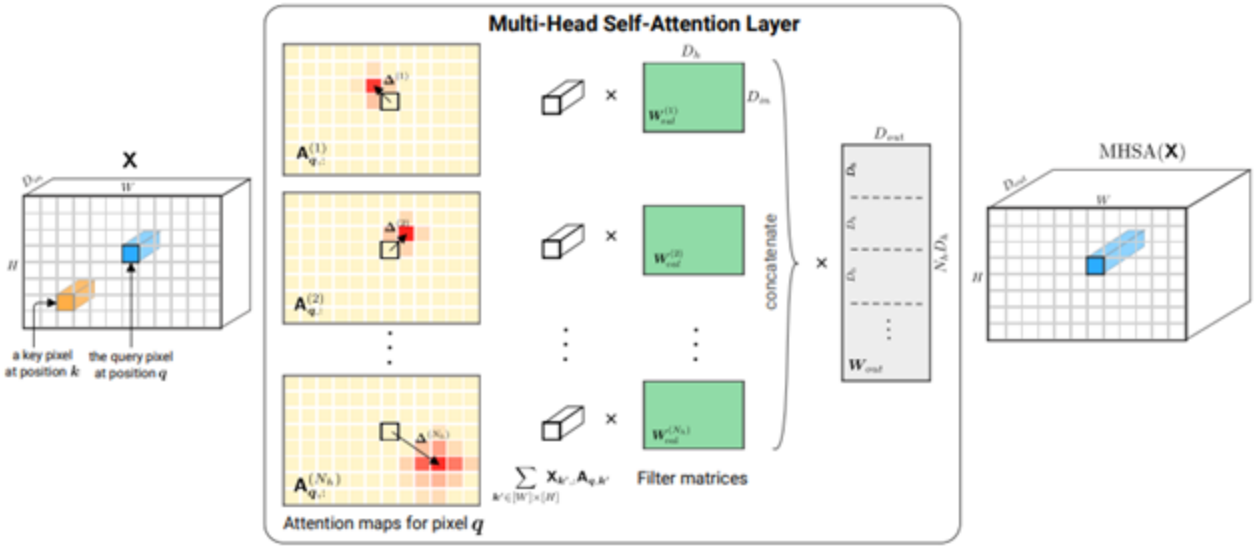
- CNN is simplified self-attention.

Self-attention: CNN with learnable receptive field

- Self-attention is the complex version of CNN.

Self-attention v.s. CNN





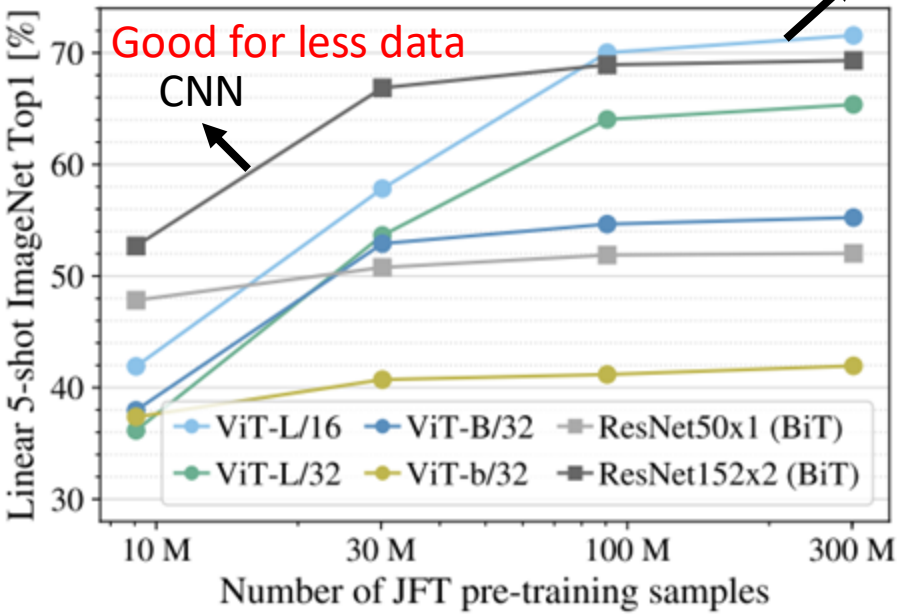
On the Relationship between Self-Attention and Convolutional Layers

https://arxiv.org/abs/1911.03584

Self-attention v.s. CNN

Good for more data

Self-attention

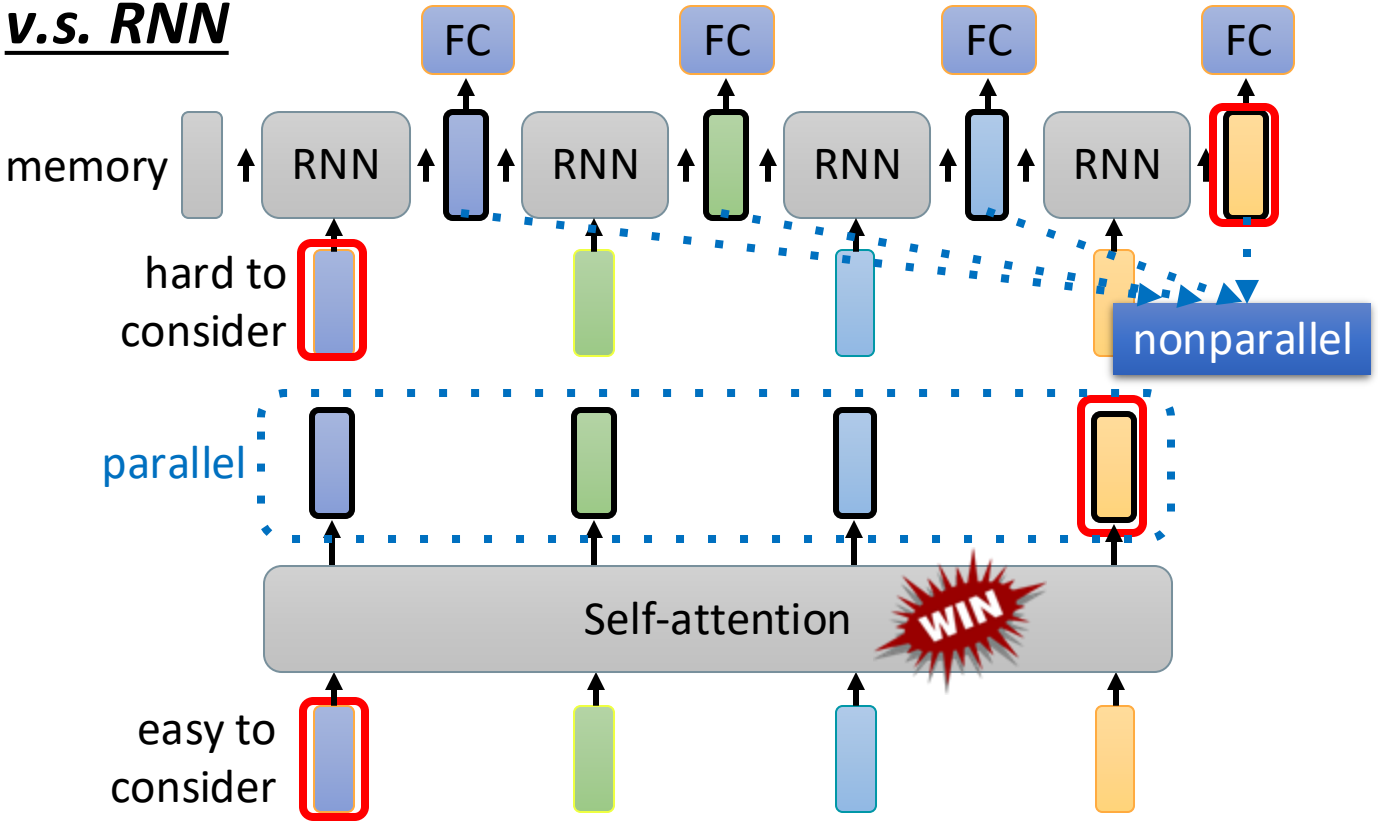


An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

https://arxiv.org/pdf/2010.11929.pdf

Self-attention v.s. RNN

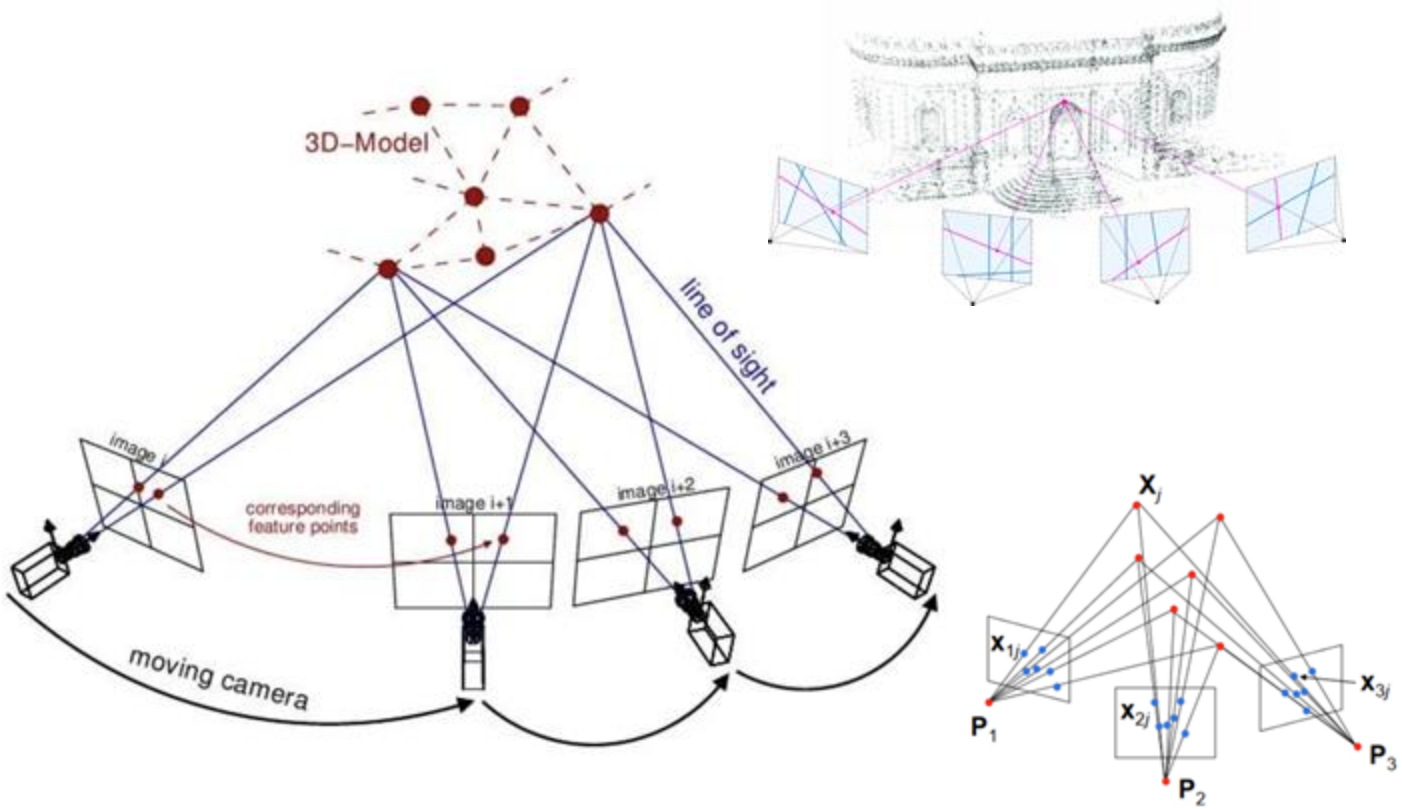
Recurrent Neural Network (RNN)



Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

https://arxiv.org/abs/2006.16236

Structure from Motion: 2D -> 3D



Rendering: 3D -> 2D

- Rendering or image synthesis is a process of generating an image from a 2D or 3D model using a computer program. The resulting image is called render
- Rendering is usually the last step in the graphics pipeline which gives models and animations their final appearance.
- Any rendering application gets an *input file* called a *scene file*. This scene file contains multiple information like for example:
  - Model (3D or 2D model itself)
  - Texture
  - Shading
  - Shadows
  - Reflection
  - Lighting
  - Viewpoint
- The information stated above is considered as a feature for rendering.
- It means we can say that each scene file contains multiple features that need to be understood and processed by the rendering algorithm or application to generate a processed image.

Rendering: 3D -> 2D

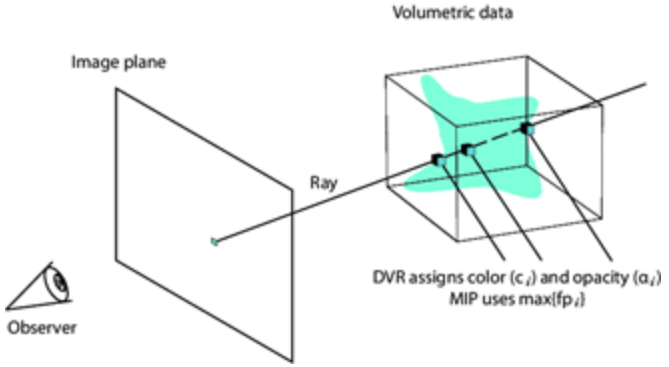


Image formation model

Given a set of 3D points, possibly, the most interesting part is to see how can it be used for rendering. You might be previously familiar with a point-wise α-blending used in NeRF. Turns out that **NeRFs and Gaussian splatting share the same image formation model.** To see this, let's take a little detour and re-visit the volumetric rendering formula given in NeRF[2] and many of its follow-up works (1). We will also rewrite it using simple transitions (2):

$$
\begin{aligned}
C(p) &= \\
&= \sum_{i=1}^{N} c_i (1 - \exp(-\sigma_i \delta_i)) T_i = \\
&= \sum_{i=1}^{N} c_i (1 - \exp(-\sigma_i \delta_i)) \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j) = \qquad (1) \\
&= \sum_{i=1}^{N} c_i \underbrace{(1 - \exp(-\sigma_i \delta_i))}_{\alpha_i} \prod_{j=1}^{i-1} \underbrace{\exp(-\sigma_j \delta_j)}_{1-\alpha_j} = \\
&= \sum_{i=1}^{N} c_i \alpha_i \underbrace{\prod_{j=1}^{i-1} (1 - \alpha_j)}_{transmittance} \qquad (2)
\end{aligned}
$$

NeRF: Neural Radiative Field

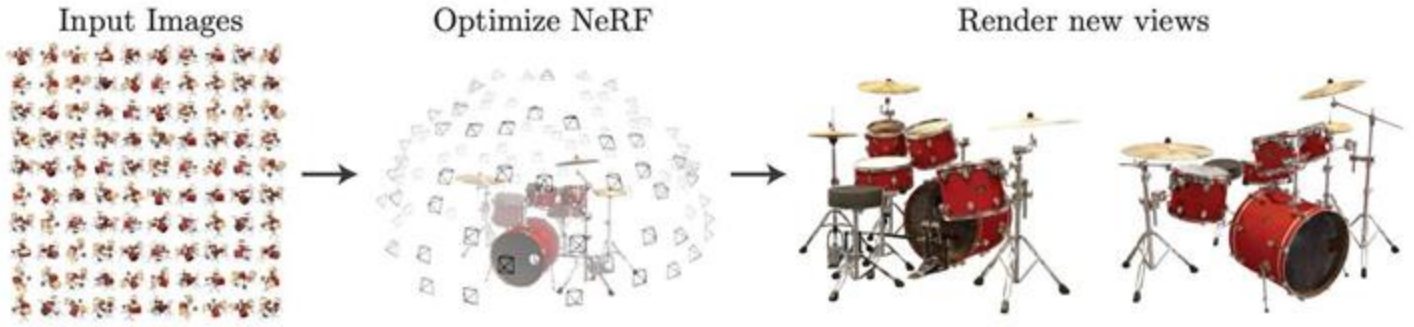


Fig. 1: We present a method that optimizes a continuous 5D neural radiance field representation (volume density and view-dependent color at any continuous location) of a scene from a set of input images. We use techniques from volume rendering to accumulate samples of this scene representation along rays to render the scene from any viewpoint. Here, we visualize the set of 100 input views of the synthetic *Drums* scene randomly captured on a surrounding hemisphere, and we show two novel views rendered from our optimized NeRF representation.

Neural radiative field

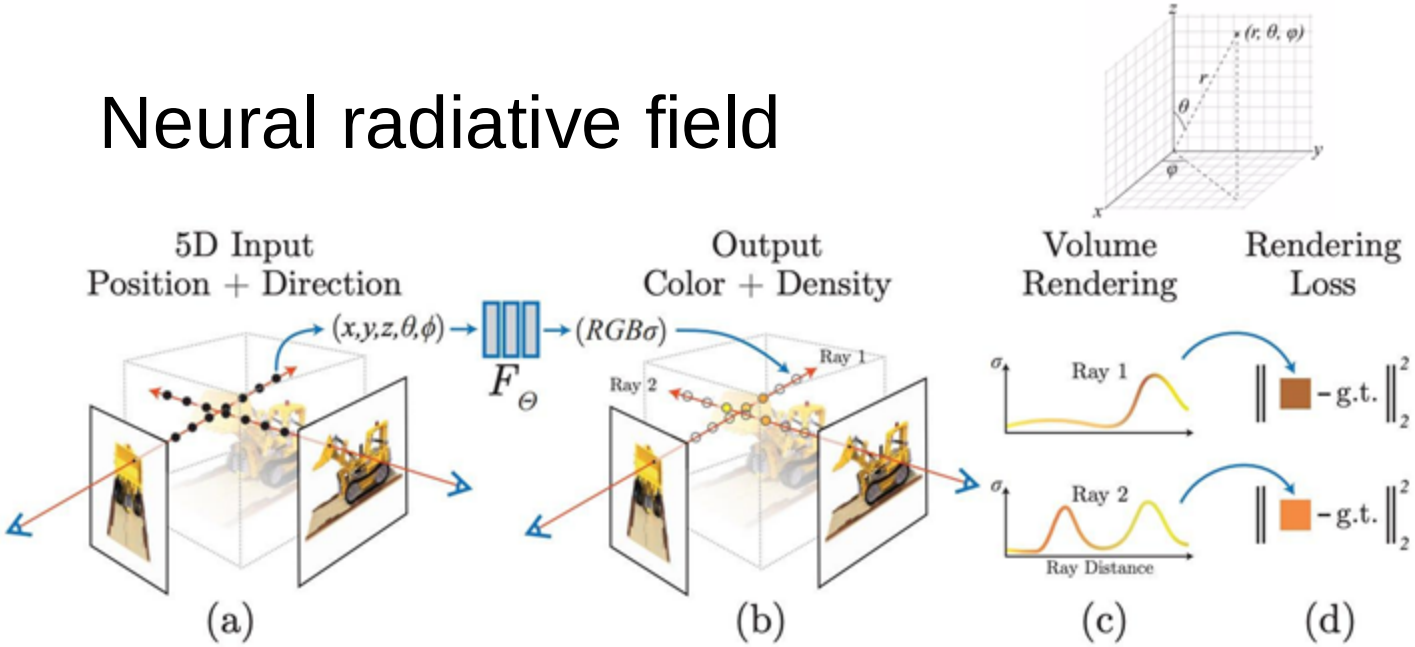
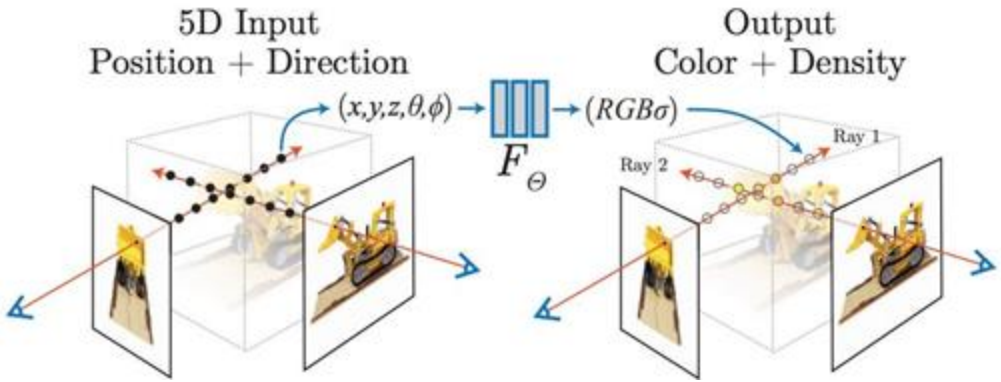
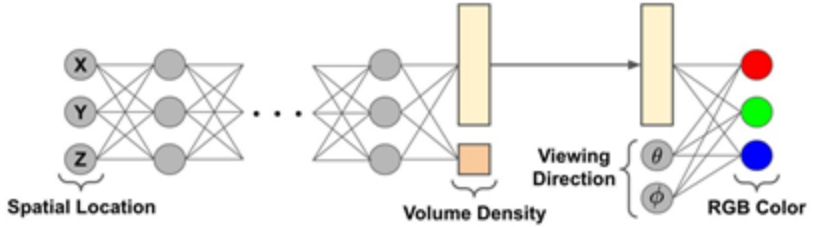


Fig. 2: An overview of our neural radiance field scene representation and differentiable rendering procedure. We synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays (a), feeding those locations into an MLP to produce a color and volume density (b), and using volume rendering techniques to composite these values into an image (c). This rendering function is differentiable, so we can optimize our scene representation by minimizing the residual between synthesized and ground truth observed images (d).



the neural network. In [1], we model radiance fields with a feed-forward neural network, which takes a 5D input and is trained to produce the corresponding color and volume density as output; see above. Recall, however, that color is view-dependent and volume density is not. To account for this, we first pass the input 3D coordinate through several feed-forward layers, which produce both the volume density and a feature vector as output. This feature vector is then concatenated with the viewing direction and passed through an extra feed-forward layer to predict the view-dependent, RGB color; see below.



View dependent RGB output

a few extra details. We now understand most of the components of a NeRF. However, the approach that we've described up to this point is actually shown in [1] to be inefficient and generally bad at representing scenes. To improve the model, we can:

1. Replace spatial coordinates (for both the spatial location and the viewing direction) with positional embeddings.
2. Adopt a hierarchical sampling approach for volume rendering.

By using positional embeddings, we map the feed-forward network's inputs (i.e., the spatial location and viewing direction coordinates) to a higher-dimension. Prior work showed that such an approach, as opposed to using spatial or directional coordinates as input directly, allows neural networks to better model high-frequency (i.e., changing a lot/quickly) features of a scene [5]. This makes the quality of the NeRF's output much better; see below.

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

(from [1])

Put simply, positional embeddings take a scalar number as input (e.g., a coordinate value or an index representing position in a sequence) and produce a higher-dimensional vector as output. We can either learn these embeddings during training or use a fixed function to generate them. For NeRFs, we use the function shown above, which takes a scalar p as input and produces a $2L$-dimensional position encoding as output.

The hierarchical sampling approach used by NeRF makes the rendering process more efficient by only sampling (and passing through the feed-forward neural network) locations and viewing directions that are likely to impact the final rendering result. This way, we only evaluate the neural network where needed and avoid wasting computation on empty or occluded areas.

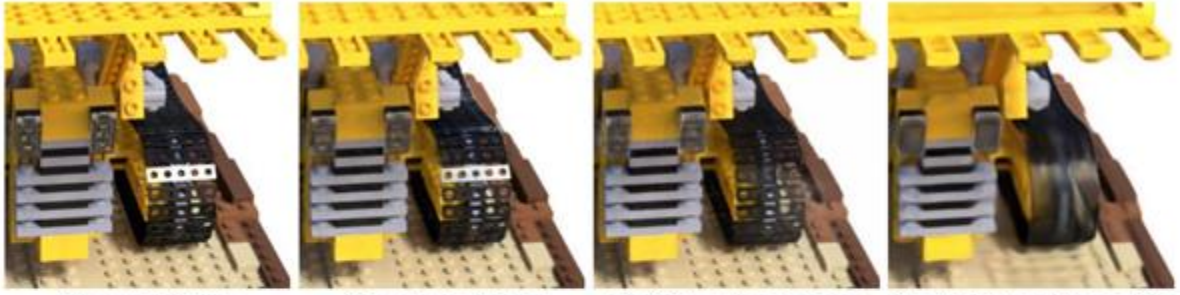
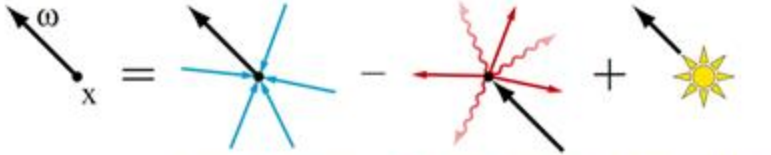
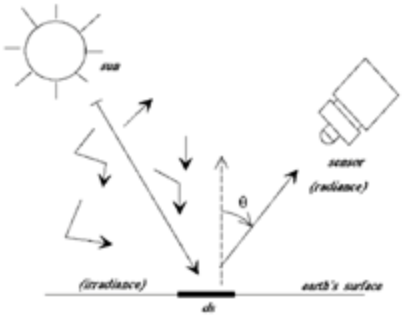


Fig. 4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance.

Radiative Transfer Equations - how to describe *the variation of the radiance L per unit distance along ω?* The equation of radiative transfer simply says that ***as a beam of radiation travels***, it loses energy to absorption, gains energy by emission processes, and redistributes energy by scattering.





$$(\omega \cdot \nabla) L(x, \omega) = \sigma_s(x) \int_{\mathbb{S}^2} f_p(x, \omega_i \to \omega) L(x, \omega_i) \, \mathrm{d}\omega_i - \sigma_t(x) L(x, \omega) + Q(x, \omega)$$

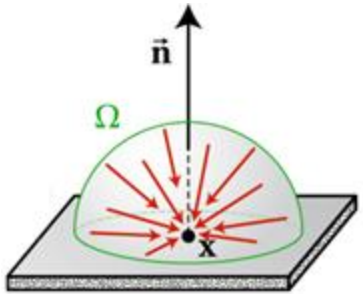
In-scattering | Out-scattering & absorption | Emission

Scattering coefficient: $\sigma_s(x) \in \mathbb{R}_{\geq 0}$,
Phase function: $f_p(x, \omega_i \to \omega)$, a probability density over $\mathbb{S}^2$ given x and ω_i

The ratio between σ_s and σ_t controls the fraction of radiant energy *not* being absorbed at each scattering and is also known as the *single-scattering albedo*

Extinction coefficient: $\sigma_t(x) \in \mathbb{R}_{\geq \sigma_s(x)}$
σ_t controls how frequently light scatters and is also known as the *optical density*

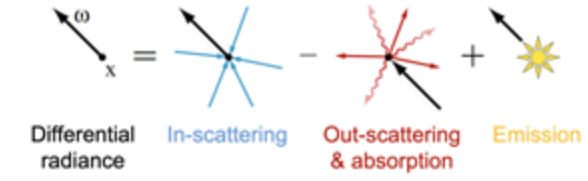
Source term: $Q(x, \omega) \in \mathbb{R}_{\geq 0}$

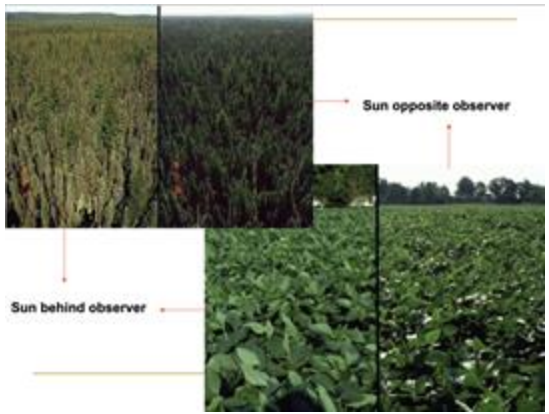


- Differential radiance

$$(\omega \cdot \nabla) L(x, \omega) = \frac{\mathrm{d} L(x + \tau\omega, \omega)}{\mathrm{d}\tau}\bigg|_{\tau=0} = \lim_{\tau \to 0} \frac{L(x + \tau\omega, \omega) - L(x, \omega)}{\tau}$$

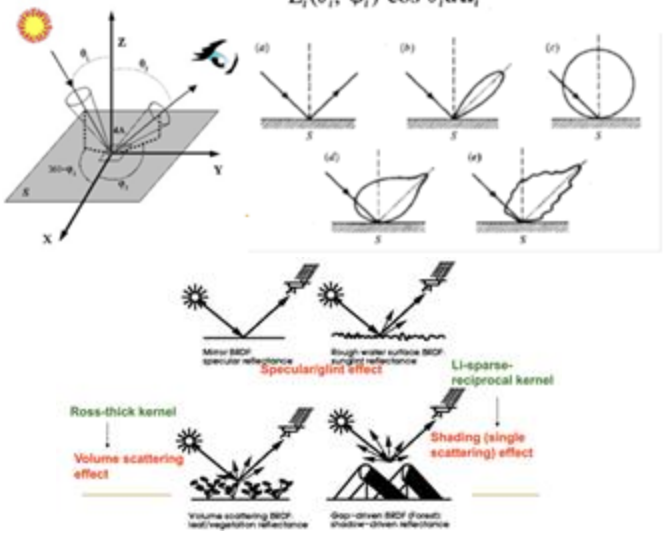
In-scattering - How to describe radiation directional properties? BRDF



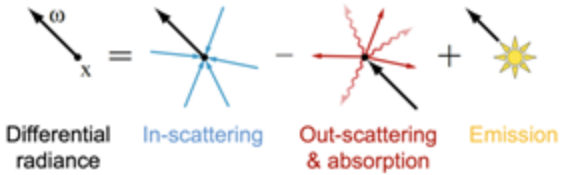


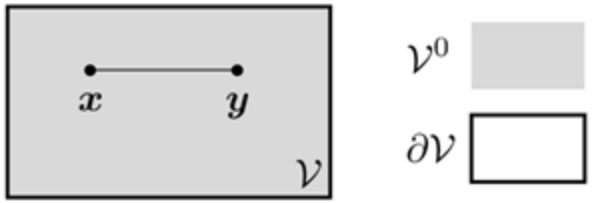
The Bi-directional Reflectance Distribution Function (BRDF) is used to describe the dependence of reflected radiation on the incident (i) and outgoing (v) directions (Nicodemus, 1977).

$$B(\theta_i, \phi_i, \theta_v, \phi_v) = \frac{dL_v(\theta_i, \phi_i, \theta_v, \phi_v)}{L_i(\theta_i, \phi_i)\cos\theta_i d\Omega_i} \text{ sr}^{-1}$$



Out-scattering & absorption - How to quantify ***attenuation***? Beer's Law





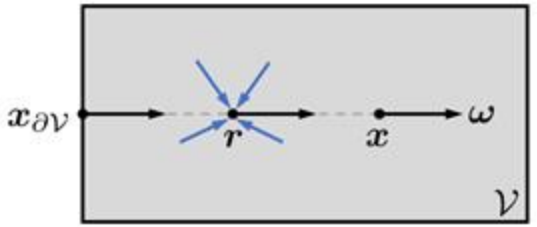
- For any $x, y \in \mathcal{V}$, the attenuation between ***x*** and ***y*** is

$$T(\boldsymbol{x} \leftrightarrow \boldsymbol{y}) := \exp\left(-\int_{(\boldsymbol{x},\boldsymbol{y})} \sigma_t(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}\right)$$

  - A line integral between ***x*** and ***y***
  - $0 \le T(\boldsymbol{x} \leftrightarrow \boldsymbol{y}) \le 1$ for all ***x*** and ***y***
  - For homogeneous media with $\sigma_t(\boldsymbol{x}) \equiv \sigma_t$,

$$T(\boldsymbol{x} \leftrightarrow \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|\sigma_t)$$

Solving Radiative Transfer Equations - Derive Integral form of RTEs



$\mathcal{V}^0$

$\partial\mathcal{V}$

Attenuation In-scattering Emission

$$L(\boldsymbol{x},\boldsymbol{\omega}) = \int_0^{h(\boldsymbol{x},\boldsymbol{\omega})} T(\boldsymbol{r}\leftrightarrow\boldsymbol{x})\left[\sigma_s(\boldsymbol{r})\int_{\mathbb{S}^2} f_p(\boldsymbol{r},\boldsymbol{\omega}_i\to\boldsymbol{\omega})L(\boldsymbol{r},\boldsymbol{\omega}_i)\,\mathrm{d}\boldsymbol{\omega}_i + Q(\boldsymbol{r},\boldsymbol{\omega})\right]\mathrm{d}\tau$$

$$+ T(\boldsymbol{x}_{\partial\mathcal{V}}\leftrightarrow\boldsymbol{x})\,L(\boldsymbol{x}_{\partial\mathcal{V}},\boldsymbol{\omega}) \quad \text{where } \boldsymbol{r} := \boldsymbol{x} - \tau\boldsymbol{\omega}$$

Attenuation Boundary cond.

(The second term vanishes when $h(\boldsymbol{x},\boldsymbol{\omega}) = +\infty$)

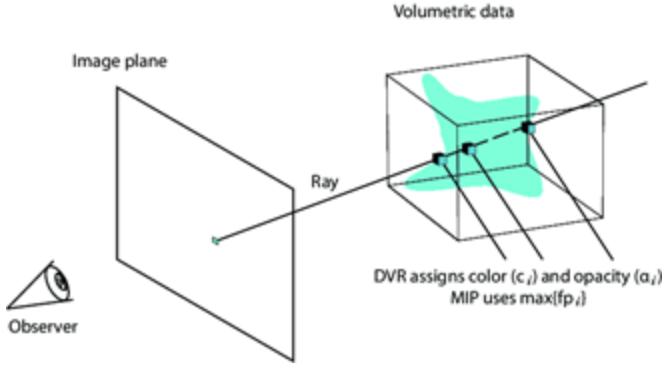
All RTMs follow this general form.

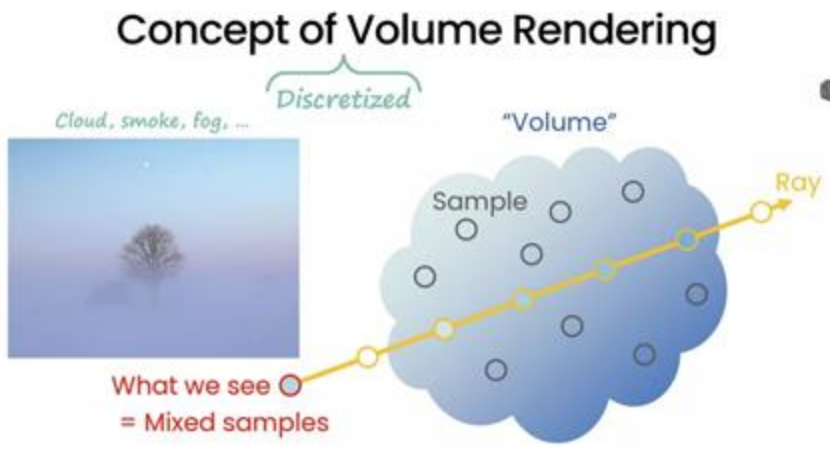
The differences however, are essentially due to the various forms for the emission and absorption coefficients.

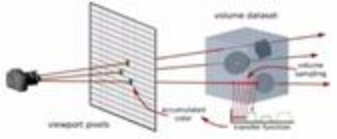
Rendering: 3D -> 2D

It does not consider the following factors:

1. The light source and its geometry
2. The in-scattering?
3. The bidirectional reflectance distribution function
4. The influence of the properties of the volume, e.g., biochemical properties







Improve Neural radiative field for Remote Sensing

- Encoder should output properties of the media, not just density and color.
- Encoder & Decoder should address not only the camera geometry, but also light source geometry
- Decoder should use bidirectional reflectance distribution function to model geometry dependance
- Decoder should integrate the properties of the media, e.g., biochemical properties, into radiative transfer, via RTMs, such as Prosail?