Transformers

CISC 7026 - Introduction to Deep Learning

Steven Morad

University of Macau

Review

Review

Last time, we derived various forms of **attention**

Review

Last time, we derived various forms of **attention**

We started with composite memory

Review

Last time, we derived various forms of **attention**

We started with composite memory

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_i$$

Review

Last time, we derived various forms of **attention**

We started with composite memory

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_i$$

Given large enough T, we will eventually run out of storage space

Review

Last time, we derived various forms of **attention**

We started with composite memory

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_i$$

Given large enough T, we will eventually run out of storage space

The sum is a **lossy** operation that can store a limited amount of information

Review

So we introduced a forgetting term γ

Review

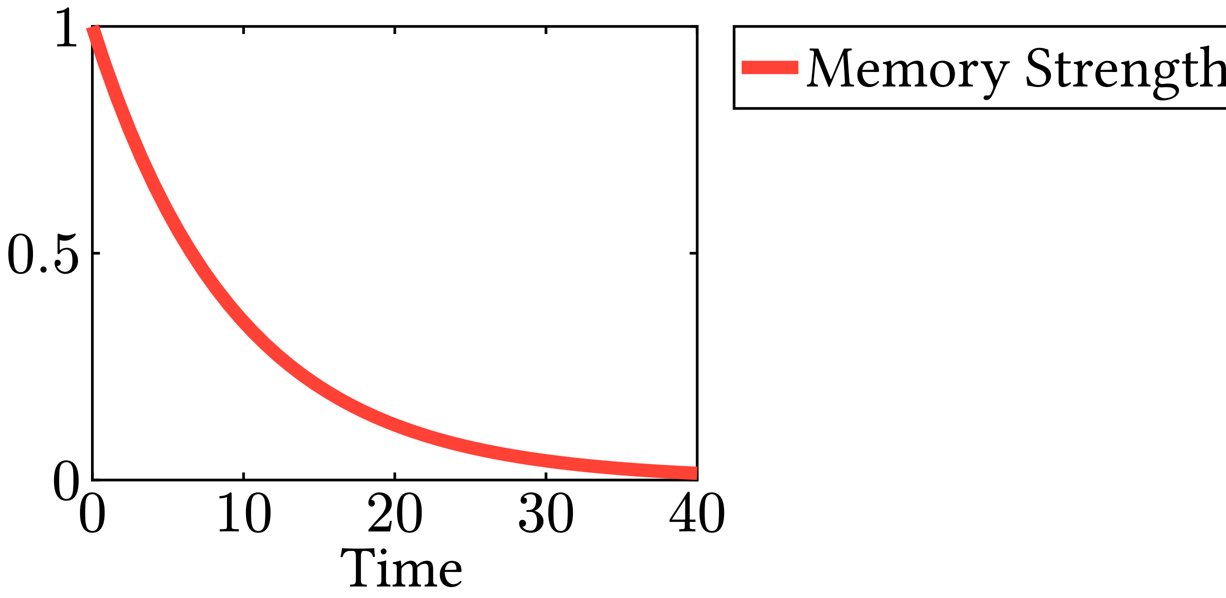
So we introduced a forgetting term γ

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{T} \gamma^{T-i} \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_i$$

Review

So we introduced a forgetting term γ

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=1}^{T} \gamma^{T-i} \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_i$$



Review

We went to a party and the forgetting seemed ok

Review

We went to a party and the forgetting seemed ok

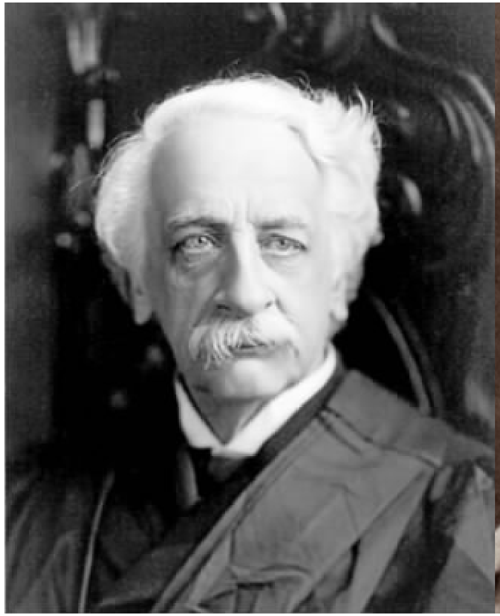
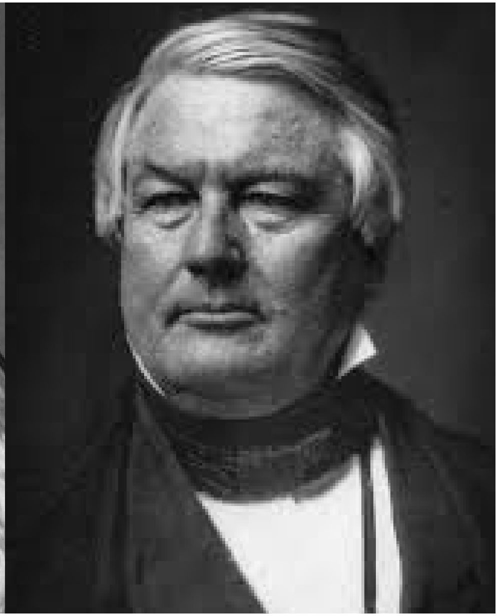
Review

We went to a party and the forgetting seemed ok

10 PM

Review

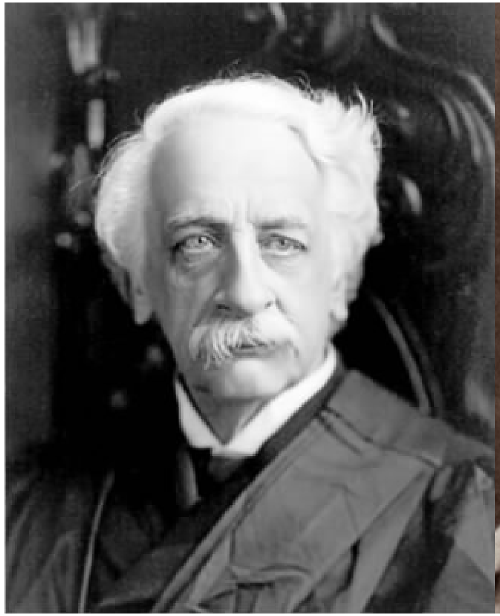
We went to a party and the forgetting seemed ok

10 PM   11 PM

Review

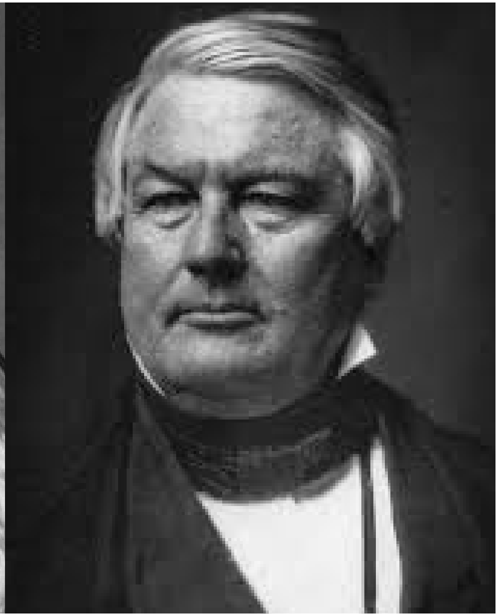
We went to a party and the forgetting seemed ok



10 PM

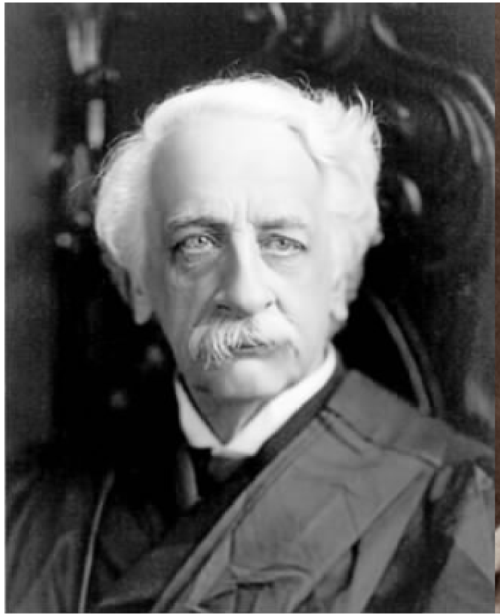
11 PM

12 AM



Review

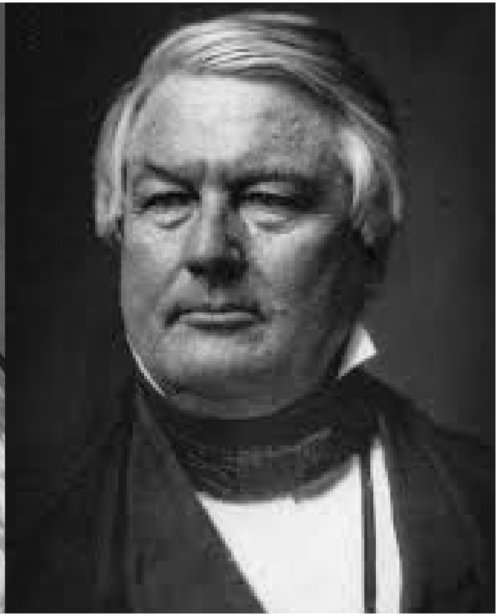
We went to a party and the forgetting seemed ok



10 PM

11 PM

12 AM



1 AM

Review

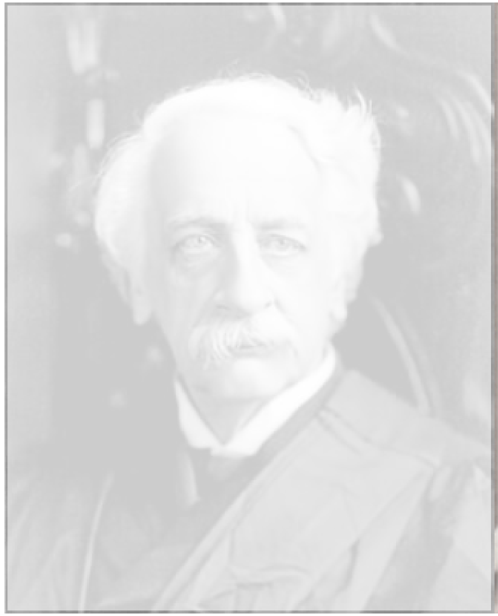
Review

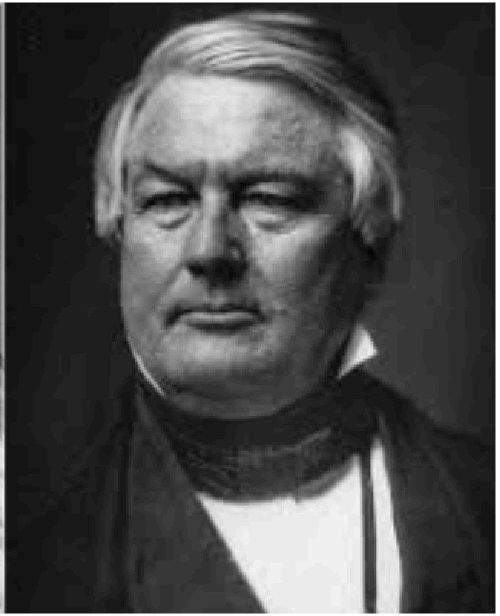
$$\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$$

Review

$\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$ $\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$

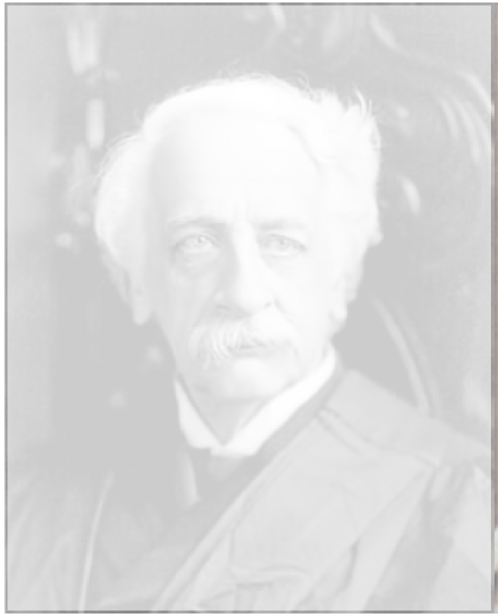
Review



$\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$ $\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$ $\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$

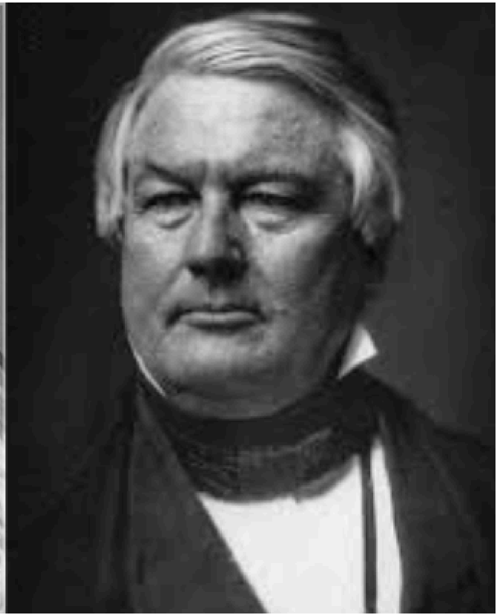
Review



$$\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$$

$$\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$$

$$\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$$



$$\gamma^0 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$$

Review

But we encountered problems when Taylor Swift arrived at the party

Review

But we encountered problems when Taylor Swift arrived at the party

Review

But we encountered problems when Taylor Swift arrived at the party

$\gamma^4 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$ $\quad$ $\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$ $\quad$ $\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$ $\quad$ $\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$ $\quad$ $\gamma^0 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$

Review

$\gamma^4 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$ $\quad$ $\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$ $\quad$ $\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$ $\quad$ $\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$ $\quad$ $\gamma^0 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$

Review

$$\gamma^4 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad \gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \qquad \gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \qquad \gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4 \qquad \gamma^0 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$$

With composite memory, we forget Taylor Swift!

Review

$\gamma^4 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$ $\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$ $\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$ $\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$ $\gamma^0 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$

With composite memory, we forget Taylor Swift!

Our model of human memory was incomplete

Review

So we introduced **attention**

Review

So we introduced **attention**

The attention we pay to person i is

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i = \operatorname{softmax}\left(\begin{bmatrix} \boldsymbol{\theta}_\lambda^\top \overline{\boldsymbol{x}}_1 \\ \vdots \\ \boldsymbol{\theta}_\lambda^\top \overline{\boldsymbol{x}}_T \end{bmatrix}\right)_i = \frac{\exp\left(\boldsymbol{\theta}_\lambda^\top \overline{\boldsymbol{x}}_i\right)}{\sum_{j=1}^{T} \exp\left(\boldsymbol{\theta}_\lambda^\top \overline{\boldsymbol{x}}_j\right)}$$

Review

Review

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_1$$

$$\cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$$

Review

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_1 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_2$$

$$\cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$$

Review

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_1 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_2 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_3$$

$$\cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$$

Review

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_1 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_2 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_3 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_4$$

$$\cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$$

Review

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_1 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_2 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_3 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_4 \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_5 \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_5$$

$$\cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4 \qquad \cdot\, \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$$

Review

Review

$$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$$

Review

$$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \qquad 0.04 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$$

Review

$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \quad 0.04 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$

Review

$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \quad 0.04 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \quad 0.20 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4$

Review

$$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \quad 0.04 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \quad 0.20 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$$

Review

$$0.70 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1 \quad 0.04 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3 \quad 0.20 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_4 \quad 0.03 \cdot \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_5$$

$$0.70 + 0.04 + 0.03 + 0.20 + 0.03 = 1.0$$

Review

Then, we introduced **keys** and **queries**

Review

Then, we introduced **keys** and **queries**

Query: Which person will help me on my exam?

Review

Then, we introduced **keys** and **queries**

Query: Which person will help me on my exam?

Review

Then, we introduced **keys** and **queries**

Query: Which person will help me on my exam?

Musician Lawyer Shopkeeper Chef Scientist

Review

Then, we introduced **keys** and **queries**

Query: Which person will help me on my exam?

Musician Lawyer Shopkeeper Chef Scientist

Review

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \dots \ \ \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right], \quad \boldsymbol{K} \in \mathbb{R}^{d_h \times T}$$

Review

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \dots \ \ \boldsymbol{k}_T] = [\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T], \quad \boldsymbol{K} \in \mathbb{R}^{d_h \times T}$$

$$\boldsymbol{q} = \boldsymbol{\theta}_Q^\top \boldsymbol{x}_q, \quad \boldsymbol{\theta}_Q \in \mathbb{R}^{d_x \times d_h}, \quad \boldsymbol{q} \in \mathbb{R}^{d_h}$$

Review

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right], \quad \boldsymbol{K} \in \mathbb{R}^{d_h \times T}$$

$$\boldsymbol{q} = \boldsymbol{\theta}_Q^\top \boldsymbol{x}_q, \quad \boldsymbol{\theta}_Q \in \mathbb{R}^{d_x \times d_h}, \quad \boldsymbol{q} \in \mathbb{R}^{d_h}$$

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}(\boldsymbol{q}^\top \boldsymbol{K}) = \operatorname{softmax}(\boldsymbol{q}^\top [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{k}_T])$$

$$= \operatorname{softmax}\left(\left[\boldsymbol{q}^\top \boldsymbol{k}_1 \ \ \boldsymbol{q}^\top \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{q}^\top \boldsymbol{k}_T\right]\right)$$

Review

$$\boldsymbol{K} = [\boldsymbol{k}_1 \;\; \boldsymbol{k}_2 \;\; \dots \;\; \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \;\; \dots \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right], \quad \boldsymbol{K} \in \mathbb{R}^{d_h \times T}$$

$$\boldsymbol{q} = \boldsymbol{\theta}_Q^\top \boldsymbol{x}_q, \quad \boldsymbol{\theta}_Q \in \mathbb{R}^{d_x \times d_h}, \quad \boldsymbol{q} \in \mathbb{R}^{d_h}$$

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \mathrm{softmax}(\boldsymbol{q}^\top \boldsymbol{K}) = \mathrm{softmax}(\boldsymbol{q}^\top [\boldsymbol{k}_1 \;\; \boldsymbol{k}_2 \;\; \dots \;\; \boldsymbol{k}_T])$$

$$= \mathrm{softmax}([\boldsymbol{q}^\top \boldsymbol{k}_1 \;\; \boldsymbol{q}^\top \boldsymbol{k}_2 \;\; \dots \;\; \boldsymbol{q}^\top \boldsymbol{k}_T])$$

We call this **dot-product attention**

Review

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right], \quad \boldsymbol{K} \in \mathbb{R}^{d_h \times T}$$

$$\boldsymbol{q} = \boldsymbol{\theta}_Q^\top \boldsymbol{x}_q, \quad \boldsymbol{\theta}_Q \in \mathbb{R}^{d_x \times d_h}, \quad \boldsymbol{q} \in \mathbb{R}^{d_h}$$

$$\lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}(\boldsymbol{q}^\top \boldsymbol{K}) = \operatorname{softmax}(\boldsymbol{q}^\top [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{k}_T])$$

$$= \operatorname{softmax}\left([\boldsymbol{q}^\top \boldsymbol{k}_1 \ \ \boldsymbol{q}^\top \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{q}^\top \boldsymbol{k}_T]\right)$$

We call this **dot-product attention**

Then we add attention back to the composite model

Review

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \boldsymbol{x}_i \cdot \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i$$

Review

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \boldsymbol{x}_i \cdot \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i$$

We relabel $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_V$

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \sum_{i=1}^{T} \boldsymbol{\theta}_{V}^\top \boldsymbol{x}_i \cdot \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i$$

Review

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \sum_{i=1}^{T} \boldsymbol{\theta}^\top \boldsymbol{x}_i \cdot \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i$$

We relabel $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_V$

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \sum_{i=1}^{T} \boldsymbol{\theta}_{V}^\top \boldsymbol{x}_i \cdot \lambda\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}_\lambda\right)_i$$

In dot-product attention, we call $\boldsymbol{\theta}_V^\top \boldsymbol{x}_i$ the **value**

Review

In **dot product self attention** we create queries for all inputs

Review

In **dot product self attention** we create queries for all inputs

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \ \ \boldsymbol{q}_2 \ \ \dots \ \ \boldsymbol{q}_T] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T\right], \quad \boldsymbol{Q} \in \mathbb{R}^{T \times d_h}$$

Review

In **dot product self attention** we create queries for all inputs

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \ \ \boldsymbol{q}_2 \ \ \dots \ \ \boldsymbol{q}_T] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T\right], \quad \boldsymbol{Q} \in \mathbb{R}^{T \times d_h}$$

Review

Attention looks messy, but we can rewrite it in matrix form

Review

Attention looks messy, but we can rewrite it in matrix form

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \ \ \boldsymbol{q}_2 \ \ \ldots \ \ \boldsymbol{q}_T] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \ldots \ \ \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{V} = [\boldsymbol{v}_1 \ \ \boldsymbol{v}_2 \ \ \ldots \ \ \boldsymbol{v}_T] = \left[\boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_T\right]$$

Review

Attention looks messy, but we can rewrite it in matrix form

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \ \boldsymbol{q}_2 \ \dots \ \boldsymbol{q}_T] = [\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \ \dots \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T]$$

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \boldsymbol{k}_2 \ \dots \ \boldsymbol{k}_T] = [\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \dots \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T]$$

$$\boldsymbol{V} = [\boldsymbol{v}_1 \ \boldsymbol{v}_2 \ \dots \ \boldsymbol{v}_T] = [\boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_2 \ \dots \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_T]$$

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V}$$

Review

Attention looks messy, but we can rewrite it in matrix form

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \ \ \boldsymbol{q}_2 \ \ \dots \ \ \boldsymbol{q}_T] = [\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T]$$

$$\boldsymbol{K} = [\boldsymbol{k}_1 \ \ \boldsymbol{k}_2 \ \ \dots \ \ \boldsymbol{k}_T] = [\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_K^\top \boldsymbol{x}_T]$$

$$\boldsymbol{V} = [\boldsymbol{v}_1 \ \ \boldsymbol{v}_2 \ \ \dots \ \ \boldsymbol{v}_T] = [\boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \ \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_2 \ \ \dots \ \ \boldsymbol{\theta}_V^\top \boldsymbol{x}_T]$$

$$\mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{V}$$

With attention, we can create the **transformer**

Review

Attention looks messy, but we can rewrite it in matrix form

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \;\; \boldsymbol{q}_2 \;\; \ldots \;\; \boldsymbol{q}_T] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{K} = [\boldsymbol{k}_1 \;\; \boldsymbol{k}_2 \;\; \ldots \;\; \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{V} = [\boldsymbol{v}_1 \;\; \boldsymbol{v}_2 \;\; \ldots \;\; \boldsymbol{v}_T] = \left[\boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_T\right]$$

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V}$$

With attention, we can create the **transformer**

Review

Why should we care about the transformer?

Review

Why should we care about the transformer?

It is arguably the most powerful neural network architecture today

Review

Why should we care about the transformer?

It is arguably the most powerful neural network architecture today

- AlphaFold (Nobel prize)

Review

Why should we care about the transformer?

It is arguably the most powerful neural network architecture today

- AlphaFold (Nobel prize)
- ChatGPT, Qwen, LLaMA, etc

Review

Why should we care about the transformer?

It is arguably the most powerful neural network architecture today

- AlphaFold (Nobel prize)
- ChatGPT, Qwen, LLaMA, etc
- DinoV2

Review

Why should we care about the transformer?

It is arguably the most powerful neural network architecture today

- AlphaFold (Nobel prize)
- ChatGPT, Qwen, LLaMA, etc
- DinoV2

Going Deeper

Going Deeper

Modern transformers can be very deep

Going Deeper

Modern transformers can be very deep

Very deep networks require two new training tricks

Going Deeper

Modern transformers can be very deep

Very deep networks require two new training tricks

We must understand these tricks before implementing the transformer

Going Deeper

Modern transformers can be very deep

Very deep networks require two new training tricks

We must understand these tricks before implementing the transformer

Trick 1: Residual connections

Going Deeper

Modern transformers can be very deep

Very deep networks require two new training tricks

We must understand these tricks before implementing the transformer

Trick 1: Residual connections

Trick 2: Layer normalization

Going Deeper

Modern transformers can be very deep

Very deep networks require two new training tricks

We must understand these tricks before implementing the transformer

Trick 1: Residual connections

Trick 2: Layer normalization

We will start with the **residual connection**

Going Deeper

Remember that a two-layer MLP is a universal function approximator

Going Deeper

Remember that a two-layer MLP is a universal function approximator

$$| \; f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \; | < \varepsilon$$

Going Deeper

Remember that a two-layer MLP is a universal function approximator

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

This is only as the width of the network goes to infinity

Going Deeper

Remember that a two-layer MLP is a universal function approximator

$$| \, f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \, | < \varepsilon$$

This is only as the width of the network goes to infinity

It is often more efficient to create deeper networks instead

Going Deeper

Remember that a two-layer MLP is a universal function approximator

$$| \; f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \; | < \varepsilon$$

This is only as the width of the network goes to infinity

It is often more efficient to create deeper networks instead

But there is a limit!

Going Deeper

Making the network too deep can hurt performance

Going Deeper

Making the network too deep can hurt performance

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_{\mathbf{2}}), ..., \boldsymbol{\theta}_k)$$

Going Deeper

Making the network too deep can hurt performance

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

At each layer, we lose a little bit of information

Going Deeper

Making the network too deep can hurt performance

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_{\mathbf{2}}), ..., \boldsymbol{\theta}_k)$$

At each layer, we lose a little bit of information

With enough layers, all the information in $\boldsymbol{x}$ is lost!

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Question: We have seen a similar model, what was it?

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Question: We have seen a similar model, what was it?

Answer: Autoencoder! But it was just two functions, not k functions

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Question: We have seen a similar model, what was it?

Answer: Autoencoder! But it was just two functions, not k functions

Question: Do you agree or disagree with the claim?

Going Deeper

$$\boldsymbol{y} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Claim: If the input information is present in all layers of the network, then we should be able to learn the identity function $f(x) = x$

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Question: We have seen a similar model, what was it?

Answer: Autoencoder! But it was just two functions, not k functions

Question: Do you agree or disagree with the claim?

https://colab.research.google.com/drive/1qVIbQKpTuBYIa7FvC4IH-kJq-E0jmc0d#scrollTo=bg74S-AvbmJz

Going Deeper

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Going Deeper

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Very deep networks struggle to learn the identity function

Going Deeper

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Very deep networks struggle to learn the identity function

If the information from $\boldsymbol{x}$ is present in every layer, then learning the identity function should be very easy!

Going Deeper

$$\boldsymbol{x} = f_k(...f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta_2}), ..., \boldsymbol{\theta}_k)$$

Very deep networks struggle to learn the identity function

If the information from $\boldsymbol{x}$ is present in every layer, then learning the identity function should be very easy!

Question: How can we prevent $\boldsymbol{x}$ from getting lost?

Going Deeper

We can feed $\boldsymbol{x}$ to each layer

Going Deeper

We can feed x to each layer

The first approach is called the **DenseNet** approach

Going Deeper

We can feed $\boldsymbol{x}$ to each layer

The first approach is called the **DenseNet** approach

$$\begin{aligned}
\boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\
\boldsymbol{z}_2 &= f_2\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \end{bmatrix}, \boldsymbol{\theta}_2\right) \\
&\vdots \\
\boldsymbol{z}_k &= f_k\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_{k-1} \end{bmatrix}, \boldsymbol{\theta}_k\right)
\end{aligned}$$

Going Deeper

$$\begin{aligned} \boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\ \boldsymbol{z}_2 &= f_2\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \end{bmatrix}, \boldsymbol{\theta}_2\right) \\ &\vdots \\ \boldsymbol{z}_k &= f_k\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_{k-1} \end{bmatrix}, \boldsymbol{\theta}_k\right) \end{aligned}$$

Going Deeper

$$\boldsymbol{z}_1 = f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)$$

$$\boldsymbol{z}_2 = f_2\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \end{bmatrix}, \boldsymbol{\theta}_2\right)$$

$$\vdots$$

$$\boldsymbol{z}_k = f_k\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_{k-1} \end{bmatrix}, \boldsymbol{\theta}_k\right)$$

Question: Any issues with the DenseNet approach?

Going Deeper

$$\begin{aligned}
\boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\
\boldsymbol{z}_2 &= f_2\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \end{bmatrix}, \boldsymbol{\theta}_2\right) \\
&\vdots \\
\boldsymbol{z}_k &= f_k\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_{k-1} \end{bmatrix}, \boldsymbol{\theta}_k\right)
\end{aligned}$$

Question: Any issues with the DenseNet approach?

Answer: Very deep networks require too many parameters!

Going Deeper

The next method is called the **ResNet** approach

$$\begin{aligned}
\boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\
\boldsymbol{z}_2 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_2) + \boldsymbol{x} \\
\boldsymbol{z}_3 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_3) + \boldsymbol{z}_2 \\
&\vdots \\
\boldsymbol{z}_k &= f_k(\boldsymbol{x}, \boldsymbol{\theta}_k) + \boldsymbol{z}_{k-1}
\end{aligned}$$

Going Deeper

The next method is called the **ResNet** approach

$$\begin{aligned} \boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\ \boldsymbol{z}_2 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_2) + \boldsymbol{x} \\ \boldsymbol{z}_3 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_3) + \boldsymbol{z}_2 \\ &\vdots \\ \boldsymbol{z}_k &= f_k(\boldsymbol{x}, \boldsymbol{\theta}_k) + \boldsymbol{z}_{k-1} \end{aligned}$$

This allows information to flow around the layers

Going Deeper

The next method is called the **ResNet** approach

$$\begin{aligned}
\boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\
\boldsymbol{z}_2 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_2) + \boldsymbol{x} \\
\boldsymbol{z}_3 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_3) + \boldsymbol{z}_2 \\
&\vdots \\
\boldsymbol{z}_k &= f_k(\boldsymbol{x}, \boldsymbol{\theta}_k) + \boldsymbol{z}_{k-1}
\end{aligned}$$

This allows information to flow around the layers

Fewer parameters than a DenseNet, but performs slightly worse

Going Deeper

The next method is called the **ResNet** approach

$$
\begin{aligned}
\boldsymbol{z}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) \\
\boldsymbol{z}_2 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_2) + \boldsymbol{x} \\
\boldsymbol{z}_3 &= f_2(\boldsymbol{x}, \boldsymbol{\theta}_3) + \boldsymbol{z}_2 \\
&\vdots \\
\boldsymbol{z}_k &= f_k(\boldsymbol{x}, \boldsymbol{\theta}_k) + \boldsymbol{z}_{k-1}
\end{aligned}
$$

This allows information to flow around the layers

Fewer parameters than a DenseNet, but performs slightly worse

For very deep networks, we use ResNets over DenseNets

Going Deeper

ResNets use a **residual connection**

Going Deeper

ResNets use a **residual connection**

$$\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{x}$$

Going Deeper

ResNets use a **residual connection**

$$\boldsymbol{z} = \underbrace{f(\boldsymbol{x}, \boldsymbol{\theta})}_{\text{Residual}} + \boldsymbol{x}$$

Going Deeper

ResNets use a **residual connection**

$$z = \underbrace{f(\boldsymbol{x}, \boldsymbol{\theta})}_{\text{Residual}} + \boldsymbol{x}$$

Think of the residual as a small change to $\boldsymbol{x}$

Going Deeper

ResNets use a **residual connection**

$$\boldsymbol{z} = \underbrace{f(\boldsymbol{x}, \boldsymbol{\theta})}_{\text{Residual}} + \boldsymbol{x}$$

Think of the residual as a small change to $\boldsymbol{x}$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{x} = \boldsymbol{\varepsilon} + \boldsymbol{x}$$

Going Deeper

ResNets use a **residual connection**

$$\boldsymbol{z} = \underbrace{f(\boldsymbol{x}, \boldsymbol{\theta})}_{\text{Residual}} + \boldsymbol{x}$$

Think of the residual as a small change to $\boldsymbol{x}$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{x} = \boldsymbol{\varepsilon} + \boldsymbol{x}$$

Each layer is a small perturbation of $\boldsymbol{x}$

Going Deeper

ResNets use a **residual connection**

$$\boldsymbol{z} = \underbrace{f(\boldsymbol{x}, \boldsymbol{\theta})}_{\text{Residual}} + \boldsymbol{x}$$

Think of the residual as a small change to $\boldsymbol{x}$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{x} = \boldsymbol{\varepsilon} + \boldsymbol{x}$$

Each layer is a small perturbation of $\boldsymbol{x}$

This prevents $\boldsymbol{x}$ from getting lost in very deep networks

Going Deeper

The second trick is called **layer normalization**

Going Deeper

The second trick is called **layer normalization**

With parameter initialization and weight decay, we ensure the parameters are fairly small

Going Deeper

The second trick is called **layer normalization**

With parameter initialization and weight decay, we ensure the parameters are fairly small

But we can still have very small or very large outputs from each layer

Going Deeper

The second trick is called **layer normalization**

With parameter initialization and weight decay, we ensure the parameters are fairly small

But we can still have very small or very large outputs from each layer

And the magnitude of the outputs impacts the gradient

Going Deeper

The second trick is called **layer normalization**

With parameter initialization and weight decay, we ensure the parameters are fairly small

But we can still have very small or very large outputs from each layer

And the magnitude of the outputs impacts the gradient

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} \cdot x_i$$

Going Deeper

The second trick is called **layer normalization**

With parameter initialization and weight decay, we ensure the parameters are fairly small

But we can still have very small or very large outputs from each layer

And the magnitude of the outputs impacts the gradient

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} \cdot x_i$$

Question: If all $x_i = 1$, $\theta_{1,i} = 0.01$ and $d_x = 1000$, what is the output?

Going Deeper

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} x_i$$

Going Deeper

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} x_i$$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{1000} 0.01 \cdot 1 = 10$$

Going Deeper

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} x_i$$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{1000} 0.01 \cdot 1 = 10$$

What if we add another layer with the same d_x and θ?

Going Deeper

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} x_i$$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{1000} 0.01 \cdot 1 = 10$$

What if we add another layer with the same d_x and θ?

$$f_2(\boldsymbol{z}, \boldsymbol{\theta}_2) = \sum_{i=1}^{1000} 0.01 \cdot 10 = 100$$

Going Deeper

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{d_x} \theta_{1,i} x_i$$

$$f_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \sum_{i=1}^{1000} 0.01 \cdot 1 = 10$$

What if we add another layer with the same d_x and θ?

$$f_2(\boldsymbol{z}, \boldsymbol{\theta}_2) = \sum_{i=1}^{1000} 0.01 \cdot 10 = 100$$

Question: What is the problem?

Going Deeper

Let us look at the gradient

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) =$$

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1}[f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1}[f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta_1}} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1}[f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1}[f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$

$$\approx 100 \cdot 10$$

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1} [f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1} [f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$

$$\approx 100 \cdot 10$$

Can cause exploding or vanishing gradient

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1}[f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1}[f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$
$$\approx 100 \cdot 10$$

Can cause exploding or vanishing gradient

Deeper network $\Rightarrow$ worse exploding/vanishing issues

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1}[f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1}[f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$

$$\approx 100 \cdot 10$$

Can cause exploding or vanishing gradient

Deeper network $\Rightarrow$ worse exploding/vanishing issues

Question: What can we do?

Going Deeper

Let us look at the gradient

$$\nabla_{\boldsymbol{\theta}_1} f_2(f_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2) = \nabla_{\boldsymbol{\theta}_1}[f_2](f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)) \cdot \nabla_{\boldsymbol{\theta}_1}[f_1](\boldsymbol{x}, \boldsymbol{\theta}_1)$$
$$\approx 100 \cdot 10$$

Can cause exploding or vanishing gradient

Deeper network $\Rightarrow$ worse exploding/vanishing issues

Question: What can we do?

Answer: We can normalize the output of each layer

Going Deeper

Layer normalization normalizes the output of a layer

Going Deeper

Layer normalization normalizes the output of a layer

First, layer normalization **centers** the output of the layer

Going Deeper

Layer normalization normalizes the output of a layer

First, layer normalization **centers** the output of the layer

$$\mu = \frac{1}{d_y} \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

Going Deeper

Layer normalization normalizes the output of a layer

First, layer normalization **centers** the output of the layer

$$\mu = \frac{1}{d_y} \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Going Deeper

Layer normalization normalizes the output of a layer

First, layer normalization **centers** the output of the layer

$$\mu = \frac{1}{d_y} \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Question: What does this do?

Going Deeper

Layer normalization normalizes the output of a layer

First, layer normalization **centers** the output of the layer

$$\mu = \frac{1}{d_y} \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Question: What does this do?

Answer: Creates zero-mean output (both positive and negative values)

Going Deeper

$$\mu = d_y \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

$$f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Going Deeper

$$\mu = d_y \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i \qquad\qquad f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Then, layer normalization **rescales** the output by standard deviation

Going Deeper

$$\mu = d_y \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i \qquad\qquad f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Then, layer normalization **rescales** the output by standard deviation

$$\sigma = \frac{\sqrt{\sum_{i=1}^{d_y} \left(f(\boldsymbol{x}, \boldsymbol{\theta})_i - \mu\right)^2}}{d_y}$$

Going Deeper

$$\mu = d_y \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i \qquad\qquad f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Then, layer normalization **rescales** the output by standard deviation

$$\sigma = \frac{\sqrt{\sum_{i=1}^{d_y} \left(f(\boldsymbol{x}, \boldsymbol{\theta})_i - \mu\right)^2}}{d_y}$$

$$\mathrm{LN}(f(\boldsymbol{x}, \boldsymbol{\theta})) = \frac{f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu}{\sigma}$$

Going Deeper

$$\mu = d_y \sum_{i=1}^{d_y} f(\boldsymbol{x}, \boldsymbol{\theta})_i \qquad\qquad f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu$$

Then, layer normalization **rescales** the output by standard deviation

$$\sigma = \frac{\sqrt{\sum_{i=1}^{d_y} \left(f(\boldsymbol{x}, \boldsymbol{\theta})_i - \mu\right)^2}}{d_y}$$

$$\mathrm{LN}(f(\boldsymbol{x}, \boldsymbol{\theta})) = \frac{f(\boldsymbol{x}, \boldsymbol{\theta}) - \mu}{\sigma}$$

Layer norm makes the output normally distributed, $y_i \sim \mathcal{N}(0, 1)$

Going Deeper

If the output is normally distrbuted:

Going Deeper

If the output is normally distrbuted:

- 99.7% of outputs $\in [-3, 3]$

Going Deeper

If the output is normally distrbuted:

- 99.7% of outputs $\in [-3, 3]$
- 99.99% of outputs $\in [-4, 4]$

Going Deeper

If the output is normally distrbuted:

- 99.7% of outputs $\in [-3, 3]$
- 99.99% of outputs $\in [-4, 4]$
- 99.9999% of outputs $\in [-5, 5]$

Going Deeper

If the output is normally distrbuted:

- 99.7% of outputs $\in [-3, 3]$
- 99.99% of outputs $\in [-4, 4]$
- 99.9999% of outputs $\in [-5, 5]$

This helps prevent vanishing and exploding gradients

Going Deeper

Now, let's combine residual connections and layer norm and try our very deep network again

https://colab.research.google.com/drive/1qVIbQKpTuBYIa7FvC4IH-kJq-E0jmc0d#scrollTo=iQtXjGYiz5CD

Transformers

Transformers

Now we have everything we need to implement a transformer

Transformers

Now we have everything we need to implement a transformer

- Attention

Transformers

Now we have everything we need to implement a transformer

- Attention
- MLP

Transformers

Now we have everything we need to implement a transformer

- Attention
- MLP
- Residual connections

Transformers

Now we have everything we need to implement a transformer

- Attention
- MLP
- Residual connections
- Layer normalization

Transformers

Now we have everything we need to implement a transformer

- Attention
- MLP
- Residual connections
- Layer normalization

A transformer consists of many **transformer layers**

Transformers

```python
class TransformerLayer(nn.Module):
    def __init__(self):
        self.attn = Attention()
        self.mlp = Sequential(
            Linear(d_h, d_h), LeakyReLU(), Linear(d_h, d_h))
        self.norm = nn.LayerNorm(
            d_h, elementwise_affine=False)

    def forward(self, x):
        # Residual connection and layer norm
        x = self.norm(self.attn(x) + x)
        x = self.norm(self.mlp(x) + x)
        return x
```

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.layer1 = TransformerLayer()
        self.layer2 = TransformerLayer()
        self.layer3 = TransformerLayer()

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x
```

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.layer1 = TransformerLayer()
        self.layer2 = TransformerLayer()
        self.layer3 = TransformerLayer()

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x
```

Question: What are the input/output shapes of the transformer?

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.layer1 = TransformerLayer()
        self.layer2 = TransformerLayer()
        self.layer3 = TransformerLayer()

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x
```

Question: What are the input/output shapes of the transformer?

Answer: $f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$

Positional Encoding

Positional Encoding

Our transformer maps T inputs to T outputs

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence
- T pixels in an image

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence
- T pixels in an image
- T amino acids in a protein

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence
- T pixels in an image
- T amino acids in a protein

Question: Do we care about the order of the T inputs?

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence
- T pixels in an image
- T amino acids in a protein

Question: Do we care about the order of the T inputs?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \stackrel{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Positional Encoding

Our transformer maps T inputs to T outputs

$$f : \mathbb{R}^{T \times d_x} \times \Theta \mapsto \mathbb{R}^{T \times d_y}$$

- T words in a sentence
- T pixels in an image
- T amino acids in a protein

Question: Do we care about the order of the T inputs?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \stackrel{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Answer: It depends. In some tasks, yes. In others, no

Positional Encoding

Question: Detecting birds in an image of T pixels, does order matter?

Positional Encoding

Question: Detecting birds in an image of T pixels, does order matter?

Positional Encoding

Question: Detecting birds in an image of T pixels, does order matter?

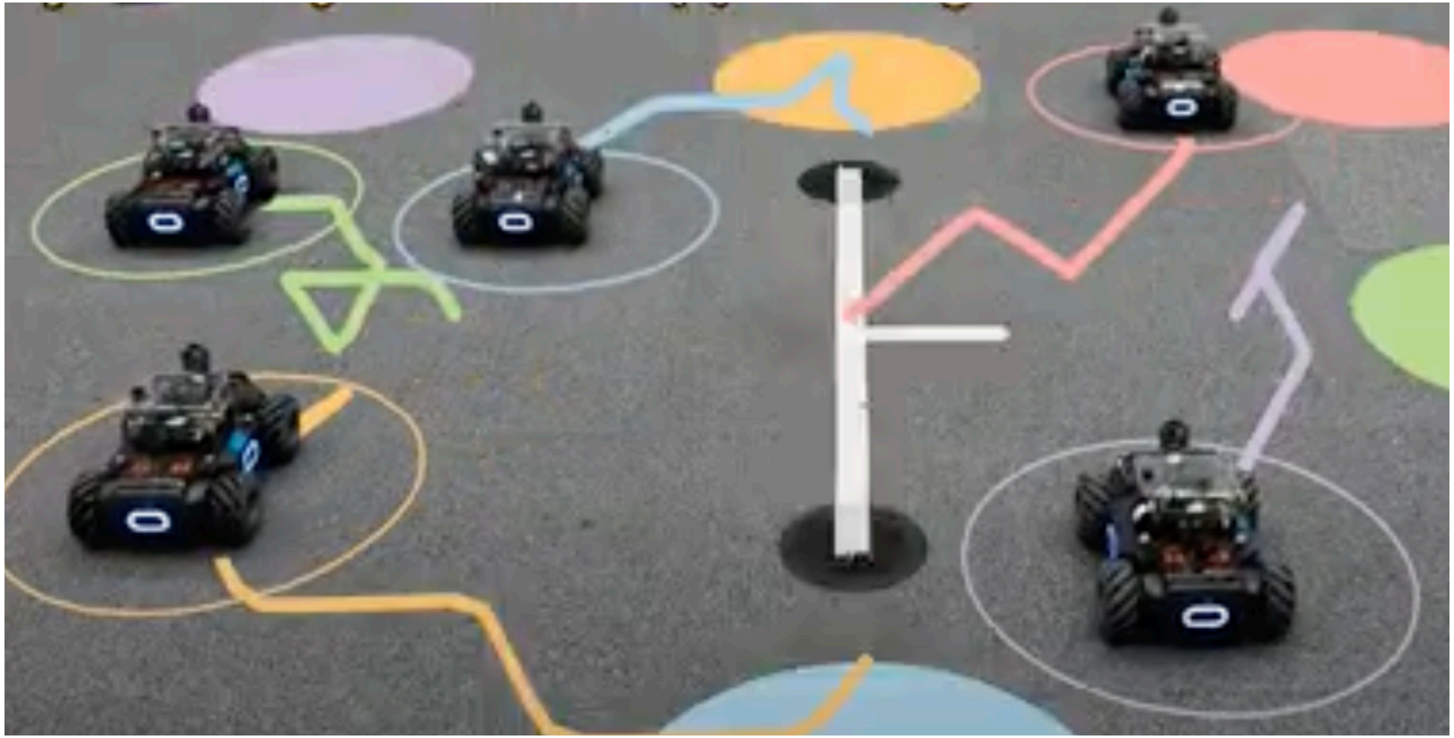
Answer: Yes!

Positional Encoding

Question: T robots searching for an object. Does order matter?

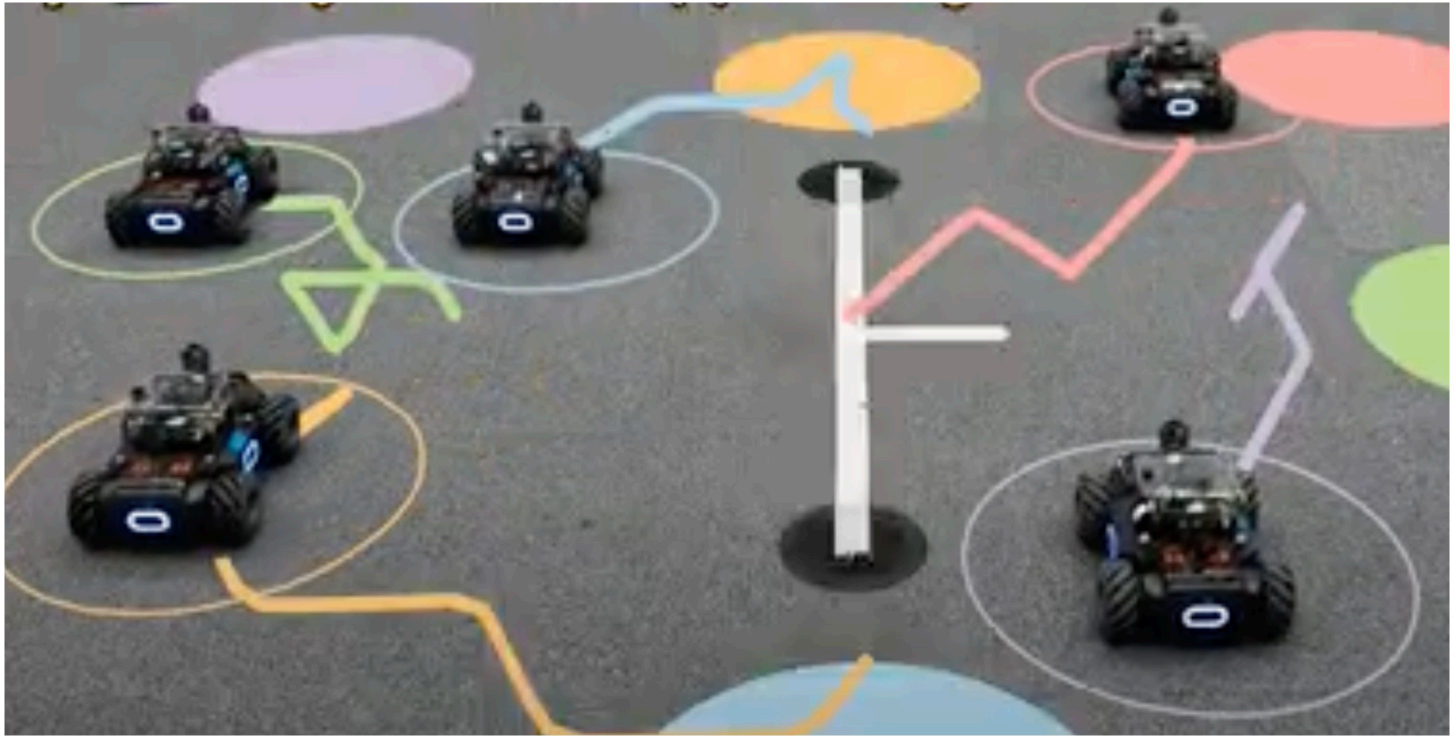
Positional Encoding

Question: T robots searching for an object. Does order matter?



Positional Encoding

Question: T robots searching for an object. Does order matter?



Answer: No!

Positional Encoding

Question: We translate a sentence containing T words. Does order matter?

Positional Encoding

Question: We translate a sentence containing T words. Does order matter?

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix}$$

Positional Encoding

Question: We translate a sentence containing T words. Does order matter?

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{x}_1 \\ \color{red}{\boldsymbol{x}_5} \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \color{red}{\boldsymbol{x}_2} \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{owner} \\ \text{licks} \\ \text{the} \\ \text{dog} \end{bmatrix}$$

Positional Encoding

Question: We translate a sentence containing T words. Does order matter?

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{x}_1 \\ {\color{red}\boldsymbol{x}_5} \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ {\color{red}\boldsymbol{x}_2} \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{owner} \\ \text{licks} \\ \text{the} \\ \text{dog} \end{bmatrix}$$

Answer: Yes!

Positional Encoding

For certain tasks, the order of inputs matters

Positional Encoding

For certain tasks, the order of inputs matters

Question: Does the order matter to the transformer?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \stackrel{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Positional Encoding

For certain tasks, the order of inputs matters

Question: Does the order matter to the transformer?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \overset{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Let us find out!

Positional Encoding

For certain tasks, the order of inputs matters

Question: Does the order matter to the transformer?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \overset{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Let us find out!

First, define a **permutation matrix** $\boldsymbol{P} \in \{0, 1\}^{T \times T}$ that reorders the inputs

Positional Encoding

For certain tasks, the order of inputs matters

Question: Does the order matter to the transformer?

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \boldsymbol{\theta}\right) \stackrel{?}{=} f\left(\begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{x}_1 \end{bmatrix}, \boldsymbol{\theta}\right)$$

Let us find out!

First, define a **permutation matrix** $\boldsymbol{P} \in \{0, 1\}^{T \times T}$ that reorders the inputs

Positional Encoding

Example 1:

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix};$$

Positional Encoding

Example 1:

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}; \quad \boldsymbol{P}\boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

Positional Encoding

Example 1:

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}; \quad \boldsymbol{P}\boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

Example 2:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix};$$

Positional Encoding

Example 1:

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}; \quad \boldsymbol{P}\boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

Example 2:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{a} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}; \quad \boldsymbol{P}\boldsymbol{a} = \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix}$$

Positional Encoding

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1\\ \vdots\\ \boldsymbol{x}_n\end{bmatrix}\right) \neq \boldsymbol{P} f\left(\begin{bmatrix}\boldsymbol{x}_1\\ \vdots\\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Positional Encoding

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) \neq \boldsymbol{P} f\left(\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does** matter (not equivariant)

Positional Encoding

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) \neq \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) = \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1\\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does** matter (not equivariant)

Positional Encoding

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) \neq \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does** matter (not equivariant)

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) = \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does not** matter (equivariant)

Positional Encoding

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) \neq \boldsymbol{P} f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does** matter (not equivariant)

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) = \boldsymbol{P} f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Order **does not** matter (equivariant)

Which is a transformer?

Positional Encoding

Recall dot product self attention

Positional Encoding

Recall dot product self attention

$$\mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{V}$$

Positional Encoding

Recall dot product self attention

$$\mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{V}$$

$$\boldsymbol{Q} = [\boldsymbol{q}_1 \;\; \boldsymbol{q}_2 \;\; \ldots \;\; \boldsymbol{q}_T] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{K} = [\boldsymbol{k}_1 \;\; \boldsymbol{k}_2 \;\; \ldots \;\; \boldsymbol{k}_T] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_T\right]$$

$$\boldsymbol{V} = [\boldsymbol{v}_1 \;\; \boldsymbol{v}_2 \;\; \ldots \;\; \boldsymbol{v}_T] = \left[\boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_2 \;\; \ldots \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_T\right]$$

Positional Encoding

Permuting the inputs reorders $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$

Positional Encoding

Permuting the inputs reorders $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \operatorname{softmax}\left(\frac{(\boldsymbol{P}\boldsymbol{Q})(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}} \right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

Permuting the inputs reorders $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\frac{(\boldsymbol{P}\boldsymbol{Q})(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\boldsymbol{P}\boldsymbol{Q} = [\boldsymbol{q}_T \;\; \boldsymbol{q}_1 \;\; \ldots \;\; \boldsymbol{q}_2] = \left[\boldsymbol{\theta}_Q^\top \boldsymbol{x}_T \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_1 \;\; \ldots \;\; \boldsymbol{\theta}_Q^\top \boldsymbol{x}_2\right]$$

$$\boldsymbol{P}\boldsymbol{K} = [\boldsymbol{k}_T \;\; \boldsymbol{k}_1 \;\; \ldots \;\; \boldsymbol{k}_2] = \left[\boldsymbol{\theta}_K^\top \boldsymbol{x}_T \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_1 \;\; \ldots \;\; \boldsymbol{\theta}_K^\top \boldsymbol{x}_2\right]$$

$$\boldsymbol{P}\boldsymbol{V} = [\boldsymbol{v}_T \;\; \boldsymbol{v}_1 \;\; \ldots \;\; \boldsymbol{v}_2] = \left[\boldsymbol{\theta}_V^\top \boldsymbol{x}_T \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_1 \;\; \ldots \;\; \boldsymbol{\theta}_V^\top \boldsymbol{x}_2\right]$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \operatorname{softmax}\left(\frac{(\boldsymbol{P}\boldsymbol{Q})(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}} \right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\mathrm{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \mathrm{softmax}\left(\frac{(\boldsymbol{P}\boldsymbol{Q})(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\mathrm{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \mathrm{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\frac{(\boldsymbol{P}\boldsymbol{Q})(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}(\boldsymbol{P}\boldsymbol{K})^\top}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}(\boldsymbol{K}^\top\boldsymbol{P}^\top)}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \operatorname{softmax}\left(\boldsymbol{P} \frac{\boldsymbol{Q}(\boldsymbol{K}^\top \boldsymbol{P}^\top)}{\sqrt{d_h}} \right) (\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}(\boldsymbol{K}^\top \boldsymbol{P}^\top)}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\boldsymbol{P}^\top\right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}(\boldsymbol{K}^\top \boldsymbol{P}^\top)}{\sqrt{d_h}}\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \operatorname{softmax}\left(\boldsymbol{P}\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\boldsymbol{P}^\top\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\boldsymbol{P}^\top\right)(\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\mathrm{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \,\mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \boldsymbol{P}^\top \right) (\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\mathrm{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \, \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \boldsymbol{P}^\top \right) (\boldsymbol{P}\boldsymbol{V})$$

$$\mathrm{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \, \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{P}^\top (\boldsymbol{P}\boldsymbol{V})$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \boldsymbol{P}^\top \right) (\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{P}^\top (\boldsymbol{P}\boldsymbol{V})$$

$\boldsymbol{P}$ swaps row i and j

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \boldsymbol{P}^\top \right) (\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{P}^\top (\boldsymbol{P}\boldsymbol{V})$$

$\boldsymbol{P}$ swaps row i and j $\quad$ $\boldsymbol{P}^T$ swaps row j and i

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \boldsymbol{P}^\top \right) (\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{P}^\top (\boldsymbol{P}\boldsymbol{V})$$

$\boldsymbol{P}$ swaps row i and j $\quad$ $\boldsymbol{P}^T$ swaps row j and i $\quad$ $\boldsymbol{P}^\top \boldsymbol{P} = \boldsymbol{I}$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\boldsymbol{P}^\top\right)(\boldsymbol{P}\boldsymbol{V})$$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{P}^\top(\boldsymbol{P}\boldsymbol{V})$$

$\boldsymbol{P}$ swaps row i and j $\quad$ $\boldsymbol{P}^T$ swaps row j and i $\quad$ $\boldsymbol{P}^\top\boldsymbol{P} = \boldsymbol{I}$

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V}$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{V}$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V}$$

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) = \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T\end{bmatrix}, \boldsymbol{\theta}\right) = \boldsymbol{P}\operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}}\right)\boldsymbol{V}$$

$$f\left(\boldsymbol{P}\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right) = \boldsymbol{P}f\left(\begin{bmatrix}\boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n\end{bmatrix}\right)$$

Question: What does this mean?

Positional Encoding

$$\operatorname{attn}\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \boldsymbol{P} \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_h}} \right) \boldsymbol{V}$$

$$f\left(\boldsymbol{P} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} \right) = \boldsymbol{P} f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} \right)$$

Question: What does this mean?

Answer: Attention/transformer **does not** understand order.

Equivariant, order **does not** matter to the transformer.

Positional Encoding

This makes sense, in our party attention example we never consider the order

Positional Encoding

This makes sense, in our party attention example we never consider the order

Positional Encoding

This makes sense, in our party attention example we never consider the order

Transformer cannot determine order of inputs! **Equivariant** to ordering

Positional Encoding

The following sentences are the same to a transformer

Positional Encoding

The following sentences are the same to a transformer

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix}$$

Positional Encoding

The following sentences are the same to a transformer

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{x}_1 \\ \color{red}{\boldsymbol{x}_5} \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \color{red}{\boldsymbol{x}_2} \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{owner} \\ \text{licks} \\ \text{the} \\ \text{dog} \end{bmatrix}$$

Positional Encoding

The following sentences are the same to a transformer

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{x}_1 \\ \color{red}{\boldsymbol{x}_5} \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \color{red}{\boldsymbol{x}_2} \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{owner} \\ \text{licks} \\ \text{the} \\ \text{dog} \end{bmatrix}$$

This is a problem! For some tasks, we care about input order

Positional Encoding

The following sentences are the same to a transformer

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \boldsymbol{x}_5 \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{dog} \\ \text{licks} \\ \text{the} \\ \text{owner} \end{bmatrix} \qquad \begin{bmatrix} \boldsymbol{x}_1 \\ \color{red}{\boldsymbol{x}_5} \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \\ \color{red}{\boldsymbol{x}_2} \end{bmatrix} = \begin{bmatrix} \text{The} \\ \text{owner} \\ \text{licks} \\ \text{the} \\ \text{dog} \end{bmatrix}$$

This is a problem! For some tasks, we care about input order

Can we make the transformer care about order?

Positional Encoding

Question: What are some ways we can introduce ordering?

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting (function of time)

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting (function of time)

ALiBi: *Press, Ofir, Noah Smith, and Mike Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation." International Conference on Learning Representations.*

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting (function of time)

ALiBi: *Press, Ofir, Noah Smith, and Mike Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation." International Conference on Learning Representations.*

RoPE: *Su, Jianlin, et al. "Roformer: Enhanced transformer with rotary position embedding." Neurocomputing.*

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting (function of time)

ALiBi: *Press, Ofir, Noah Smith, and Mike Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation." International Conference on Learning Representations.*

RoPE: *Su, Jianlin, et al. "Roformer: Enhanced transformer with rotary position embedding." Neurocomputing.*

Answer 2: We can modify the inputs based on their ordering

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting (function of time)

ALiBi: *Press, Ofir, Noah Smith, and Mike Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation." International Conference on Learning Representations.*

RoPE: *Su, Jianlin, et al. "Roformer: Enhanced transformer with rotary position embedding." Neurocomputing.*

Answer 2: We can modify the inputs based on their ordering

We will focus on answer 2 because it is more common

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right)$$

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) \qquad \operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ \vdots \\ f_{\text{pos}}(T) \end{bmatrix}, \boldsymbol{\theta} \right)$$

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) \qquad \operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ \vdots \\ f_{\text{pos}}(T) \end{bmatrix}, \boldsymbol{\theta}\right)$$

Add ordering information to the inputs

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) \qquad \operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ \vdots \\ f_{\text{pos}}(T) \end{bmatrix}, \boldsymbol{\theta}\right)$$

Add ordering information to the inputs

Even if we permute the inputs, we still know the order!

Positional Encoding

$$\text{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) \qquad \text{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ \vdots \\ f_{\text{pos}}(T) \end{bmatrix}, \boldsymbol{\theta}\right)$$

Add ordering information to the inputs

Even if we permute the inputs, we still know the order!

$$\text{attn}\left(\begin{bmatrix} \boldsymbol{x}_T \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_2 \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(T) \\ f_{\text{pos}}(1) \\ \vdots \\ f_{\text{pos}}(2) \end{bmatrix}, \boldsymbol{\theta}\right)$$

Positional Encoding

What is f_{pos}?

Positional Encoding

What is f_{pos}?

Easy solution is just some random parameters

Positional Encoding

What is f_{pos}?

Easy solution is just some random parameters

$$f_{\text{pos}}(i) = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_T \end{bmatrix}_i$$

Positional Encoding

What is f_{pos}?

Easy solution is just some random parameters

$$f_{\text{pos}}(i) = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_T \end{bmatrix}_i$$

We call this a **positional encoding**

Positional Encoding

What is f_{pos}?

Easy solution is just some random parameters

$$f_{\text{pos}}(i) = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_T \end{bmatrix}_i$$

We call this a **positional encoding**

In `torch`, this is called `nn.Embedding`

Positional Encoding

What is f_{pos}?

Easy solution is just some random parameters

$$f_{\text{pos}}(i) = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_T \end{bmatrix}_i$$

We call this a **positional encoding**

In `torch`, this is called `nn.Embedding`

Now, let us rewrite the transformer with the positional encoding

Positional Encoding

So, our final transformer is

```
class Transformer(nn.Module):
    def __init__(self):
        self.f_pos = nn.Embedding(d_x, T)
        self.layer1 = TransformerLayer()
        self.layer2 = TransformerLayer()

    def forward(self, x):
        x = x + f_pos(torch.arange(x.shape[0]))
        x = self.layer1(x)
        x = self.layer2(x)
        return x
```

Positional Encoding

Let us code up the transformer in colab

https://colab.research.google.com/drive/1qVIbQKpTuBYIa7FvC4IH-kJq-E0jmc0d#scrollTo=iQtXjGYiz5CD

Applications

Applications

Now that we understand the transformer, how do we use it?

Applications

Now that we understand the transformer, how do we use it?

We can apply a transformer to many types of problems

Applications

Now that we understand the transformer, how do we use it?

We can apply a transformer to many types of problems

But today we will focus on text and images

Applications

Now that we understand the transformer, how do we use it?

We can apply a transformer to many types of problems

But today we will focus on text and images

Let us see how to create inputs for our transformers

Text Transformers

Text Transformers

Consider a dataset of sentences

$$\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix}$$

Text Transformers

Consider a dataset of sentences

$$\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix}$$

This is a vector of sentences, but a transformer input is $\mathbb{R}^{T \times d_x}$

Text Transformers

Consider a dataset of sentences

$$\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix}$$

This is a vector of sentences, but a transformer input is $\mathbb{R}^{T \times d_x}$

What if we represent a sentence like this

$$\underbrace{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}\right\} T$$

Text Transformers

$$\underbrace{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}\right\} T$$

Text Transformers

$$\underbrace{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}\right\} T$$

But these are words, the transformer needs the input to be numbers

Text Transformers

$$\underbrace{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}\right\} T$$

But these are words, the transformer needs the input to be numbers

What if we create a vector to represent each word?

Text Transformers

$$\underbrace{\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}\right\} T$$

But these are words, the transformer needs the input to be numbers

What if we create a vector to represent each word?

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \end{bmatrix}}_{d_x} \left.\vphantom{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}\right\} T$$

Text Transformers

Step 1: Find all unique words in the dataset

Text Transformers

Step 1: Find all unique words in the dataset

$$\text{unique}\left(\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix} \right)$$

$$= [\text{John} \;\; \text{likes} \;\; \text{movies} \;\; \text{Mary} \;\; I \;\; \text{dogs}]$$

Text Transformers

Step 1: Find all unique words in the dataset

$$\text{unique}\left(\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix} \right)$$

$$= [\text{John} \ \ \text{likes} \ \ \text{movies} \ \ \text{Mary} \ \ I \ \ \text{dogs}]$$

Step 2: Create a vector representation for each unique word

Text Transformers

Step 1: Find all unique words in the dataset

$$\text{unique}\left(\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix}\right)$$

$$= [\text{John} \ \ \text{likes} \ \ \text{movies} \ \ \text{Mary} \ \ I \ \ \text{dogs}]$$

Step 2: Create a vector representation for each unique word

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

Text Transformers

Step 1: Find all unique words in the dataset

$$\text{unique}\left(\begin{bmatrix} \text{John likes movies} \\ \text{Mary likes movies} \\ \text{I like dogs} \end{bmatrix}\right)$$

$$= [\text{John} \;\; \text{likes} \;\; \text{movies} \;\; \text{Mary} \;\; I \;\; \text{dogs}]$$

Step 2: Create a vector representation for each unique word

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

Step 3: Replace words with vector representations

Text Transformers

Example: Convert the sentence to vector representations

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

$$\text{John likes movies} =$$

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

$$\text{John likes movies} = \begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_3 \end{bmatrix}^\top$$

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

$$\text{John likes movies} = \begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_3 \end{bmatrix}^\top$$

$$\text{Mary likes John} =$$

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

$$\text{John likes movies} = \begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_3 \end{bmatrix}^\top$$

$$\text{Mary likes John} = \begin{bmatrix} \boldsymbol{\theta}_4 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 \end{bmatrix}^\top$$

Text Transformers

Example: Convert the sentence to vector representations

$$\begin{bmatrix} \text{John} \\ \text{likes} \\ \text{movies} \\ \text{Mary} \\ I \\ \text{dogs} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \\ \boldsymbol{\theta}_4 \\ \boldsymbol{\theta}_5 \\ \boldsymbol{\theta}_6 \end{bmatrix}$$

$$\text{John likes movies} = \begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_3 \end{bmatrix}^\top$$

$$\text{Mary likes John} = \begin{bmatrix} \boldsymbol{\theta}_4 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_1 \end{bmatrix}^\top$$

Now, let us write some pseudocode

Text Transformers

```
unique_words = set(sentence.split(" ") for sentence in xs)
word_index = {word: i for i, word in enumerate(unique_words)}
embeddings = nn.Embedding(len(tokens), d_x)
# Convert from words to parameters
xs = []
for sentence in sentences:
    xs.append([])
    for word in sentence:
        index = word_index[word]
        representation = embeddings[word_index]
        xs.append(representation)

print(xs)
>>> [[Tensor(...), Tensor(...), ...]]
```

Text Transformers

Now, feed our dataset to the transformer

Text Transformers

Now, feed our dataset to the transformer

```
model = Transformer()
for sentence_representation in xs:
    # Convert list to tensor
    x = torch.stack(tokenized_sentence)
    y = model(x)
```

Image Transformers

Image Transformers

In image transformers, we treat a **patch** of pixels as an $\boldsymbol{x}$

Image Transformers

In image transformers, we treat a **patch** of pixels as an $\boldsymbol{x}$

$$X \in [0, 1]^{3 \times 16 \times 16}$$

Image Transformers

In image transformers, we treat a **patch** of pixels as an $\boldsymbol{x}$

$$X \in [0, 1]^{3 \times 16 \times 16}$$

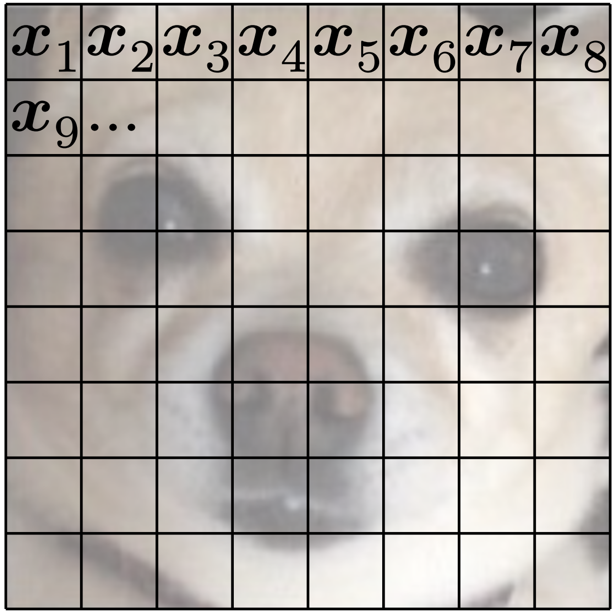
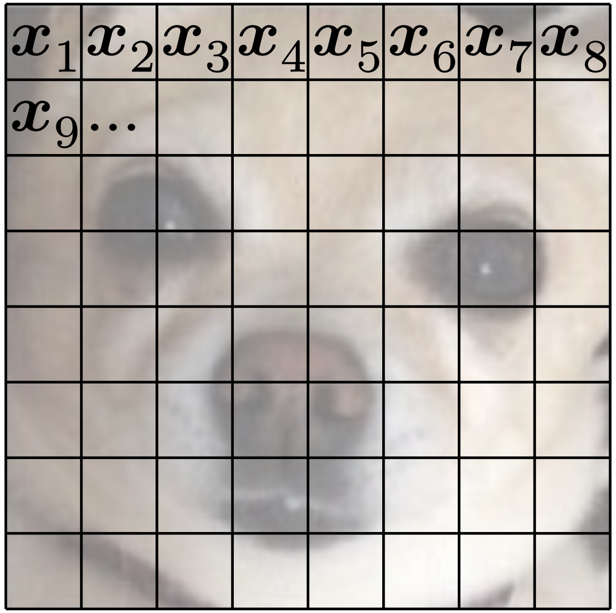
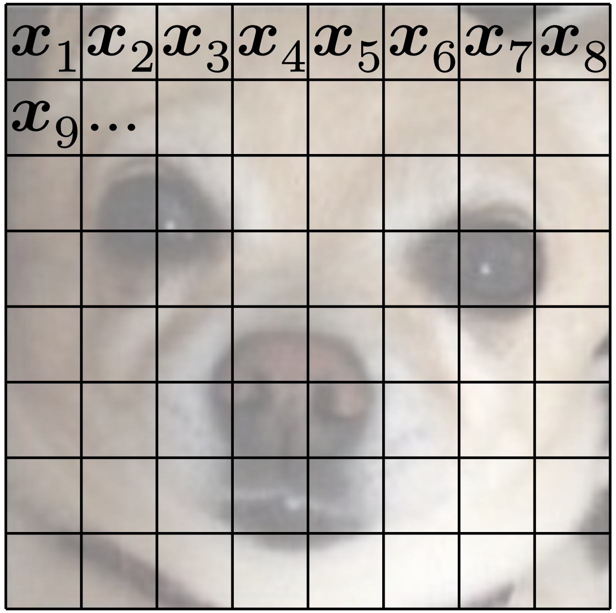


Image Transformers



Then, feed a sequence of patches to the transformer

Image Transformers



Then, feed a sequence of patches to the transformer

$$f\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right)$$

Image Transformers

```
# Convert image into patches
patches = []
for i in range(0, x.shape[0] - k + 1, k):
    for j in range(0, x.shape[1] - k + 1, k):
        patches.append(
            x[i: i + k , j: j + k]
        )
patches = stack(patches, axis=0)
print(patches.shape)
>>> (T, k, k)

model = Transformer()
y = model(patches)
```

Generative Pre-Training

Generative Pre-Training

Question: How do we train transformers?

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

In practice, transformers require lots of training data

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

In practice, transformers require lots of training data

Transformer scale very well – add more data, model becomes stronger

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

In practice, transformers require lots of training data

Transformer scale very well – add more data, model becomes stronger

Today, dataset size limits transformers

Generative Pre-Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

In practice, transformers require lots of training data

Transformer scale very well – add more data, model becomes stronger

Today, dataset size limits transformers

There are not enough graduate students to label training data!

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

Mask/hide part of the input, train the model to predict the missing part

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

Mask/hide part of the input, train the model to predict the missing part

Another name for this is **generative pre-training** (GPT)

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

Mask/hide part of the input, train the model to predict the missing part

Another name for this is **generative pre-training** (GPT)

We **generate** the missing data

Generative Pre-Training

Question: How can we train transformers with finite students/datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

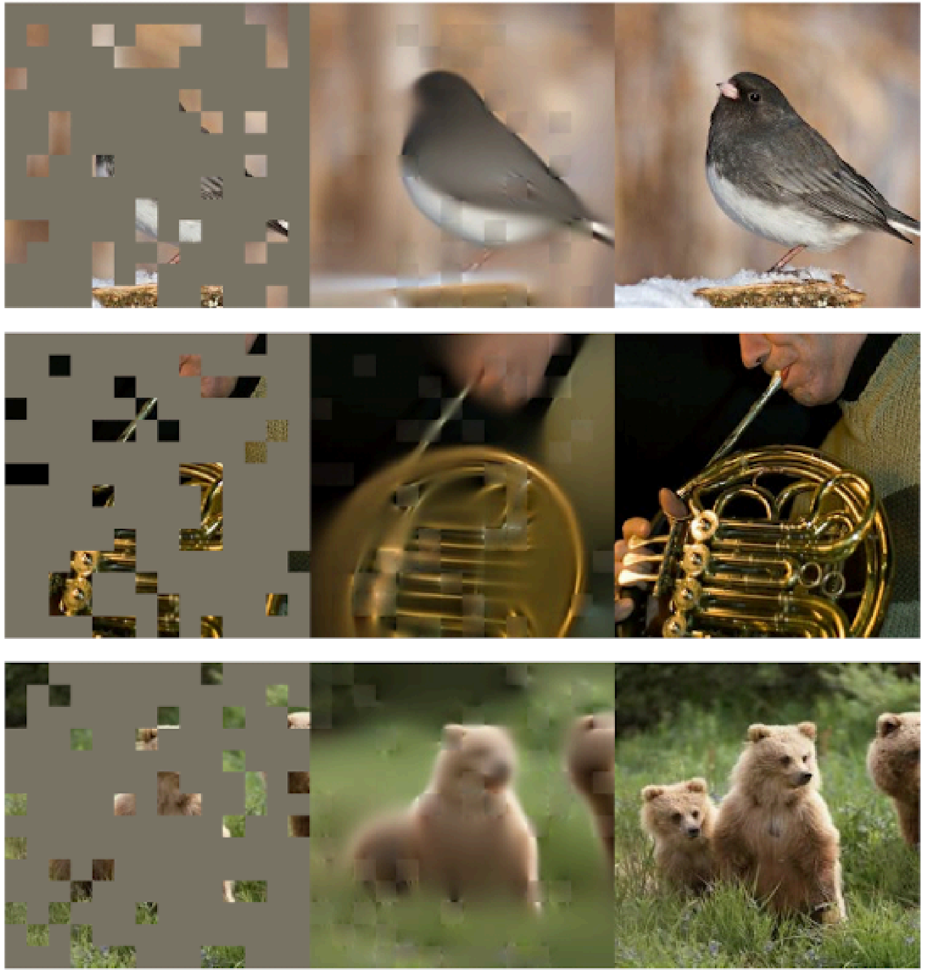
Mask/hide part of the input, train the model to predict the missing part

Another name for this is **generative pre-training** (GPT)

We **generate** the missing data

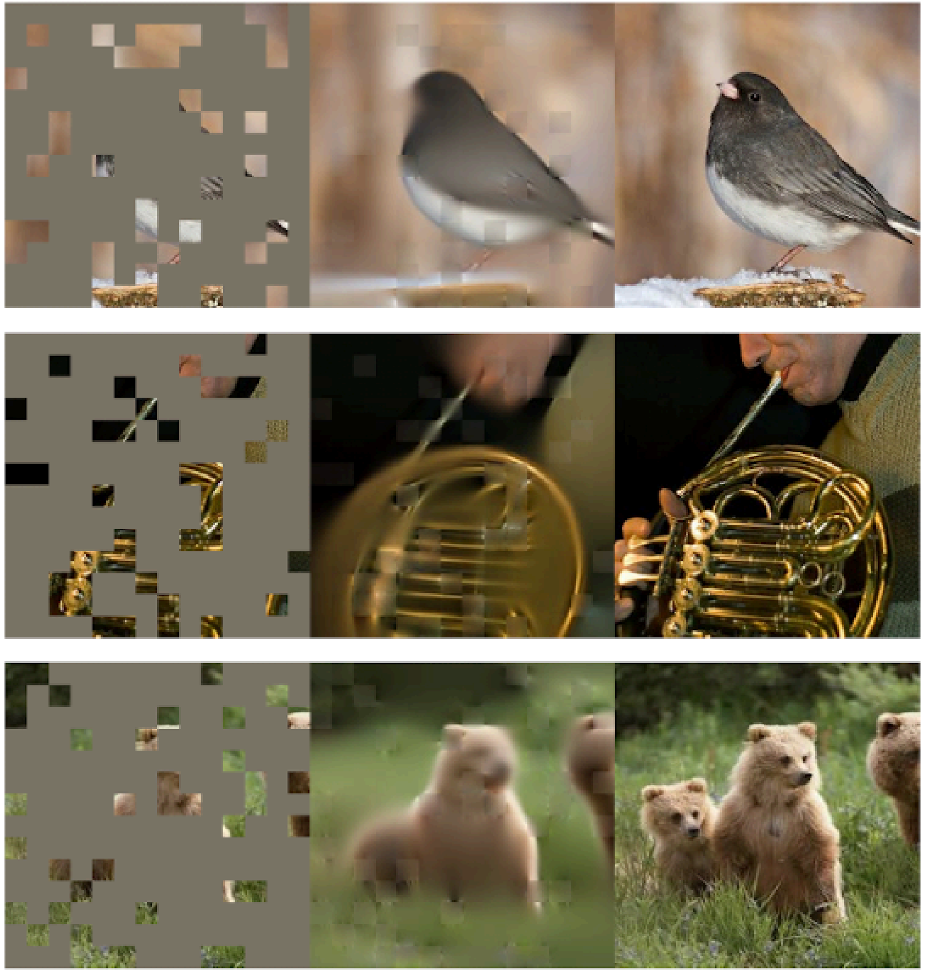
This method is **extremely** powerful

Generative Pre-Training



He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Generative Pre-Training



He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Generative Pre-Training

In GPT, we optimize the following objective

Generative Pre-Training

In GPT, we optimize the following objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

Generative Pre-Training

In GPT, we optimize the following objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$P\left(\begin{bmatrix} \text{movies} \\ \text{Mary} \\ \text{dogs} \end{bmatrix} \middle| \text{John, likes}; \boldsymbol{\theta}\right) = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

Generative Pre-Training

In GPT, we optimize the following objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$P\left(\begin{bmatrix} \text{movies} \\ \text{Mary} \\ \text{dogs} \end{bmatrix} \middle| \text{John, likes}; \boldsymbol{\theta}\right) = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

$\boldsymbol{x}_T$ = “movies”

Generative Pre-Training

In GPT, we optimize the following objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$P\left(\begin{bmatrix} \text{movies} \\ \text{Mary} \\ \text{dogs} \end{bmatrix} \middle| \text{John, likes}; \boldsymbol{\theta} \right) = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

$\boldsymbol{x}_T$ = "movies"

$$P(\text{movies} \mid \text{John, likes}; \boldsymbol{\theta}) = 0.5$$

Generative Pre-Training

In GPT, we optimize the following objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$P\left(\begin{bmatrix} \text{movies} \\ \text{Mary} \\ \text{dogs} \end{bmatrix} \middle| \text{John, likes}; \boldsymbol{\theta} \right) = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

$\boldsymbol{x}_T$ = "movies" $\quad\quad$ $P(\text{movies} \mid \text{John, likes}; \boldsymbol{\theta}) = 0.5$

Update $\boldsymbol{\theta}$ so that $P(\text{movies} \mid \text{John, likes}; \boldsymbol{\theta}) = 0.6$

Generative Pre-Training

What about for images?

Generative Pre-Training

What about for images?

Use the same objective

Generative Pre-Training

What about for images? Use the same objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

Generative Pre-Training

What about for images?

Use the same objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$= \arg\min_{\boldsymbol{\theta}} [-\log P(\text{pixel\_3} \mid \text{pixel\_1}, \text{pixel\_2}; \boldsymbol{\theta})]$$

Generative Pre-Training

What about for images? Use the same objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$= \arg\min_{\boldsymbol{\theta}} [-\log P(\text{pixel\_3} \mid \text{pixel\_1}, \text{pixel\_2}; \boldsymbol{\theta})]$$

The square error represents a normal distribution over pixel values

Generative Pre-Training

What about for images?

Use the same objective

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) = \arg\min_{\boldsymbol{\theta}} [-\log P(\boldsymbol{x}_T \mid \boldsymbol{x}_1, ..., \boldsymbol{x}_{T-1}; \boldsymbol{\theta})]$$

$$= \arg\min_{\boldsymbol{\theta}} [-\log P(\text{pixel\_3} \mid \text{pixel\_1}, \text{pixel\_2}; \boldsymbol{\theta})]$$

The square error represents a normal distribution over pixel values

$$= \arg\min_{\boldsymbol{\theta}} \left(f\left(\begin{bmatrix} \text{pixel\_1} \\ \text{pixel\_2} \end{bmatrix}, \boldsymbol{\theta}\right) - \text{pixel\_3}\right)^2$$

Generative Pre-Training

Why does this work so well?

Generative Pre-Training

Why does this work so well?

Let us see what the models learn with a GPT loss

Generative Pre-Training

Why does this work so well?

Let us see what the models learn with a GPT loss

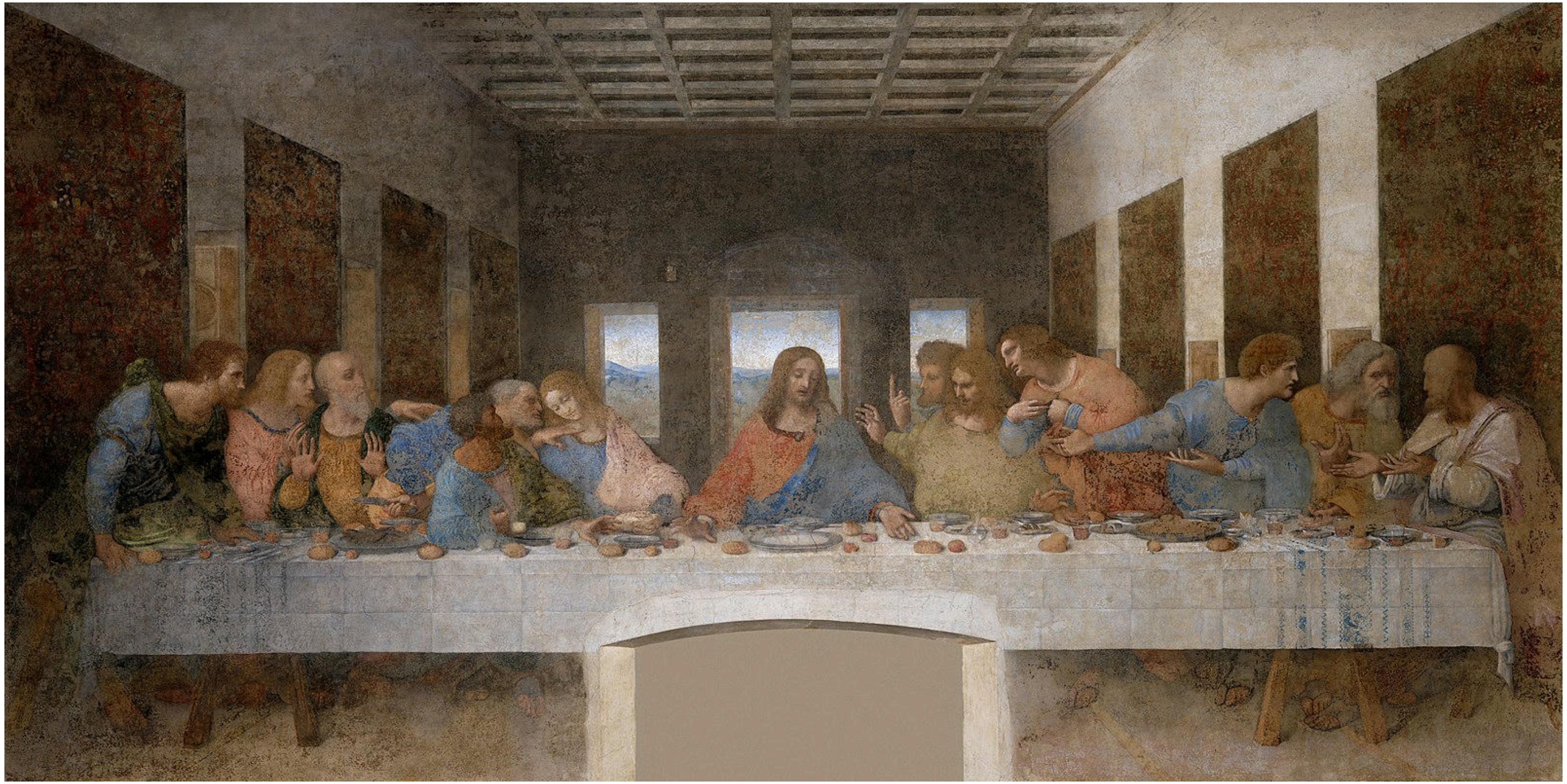
Let us start with the image transformer

Generative Pre-Training

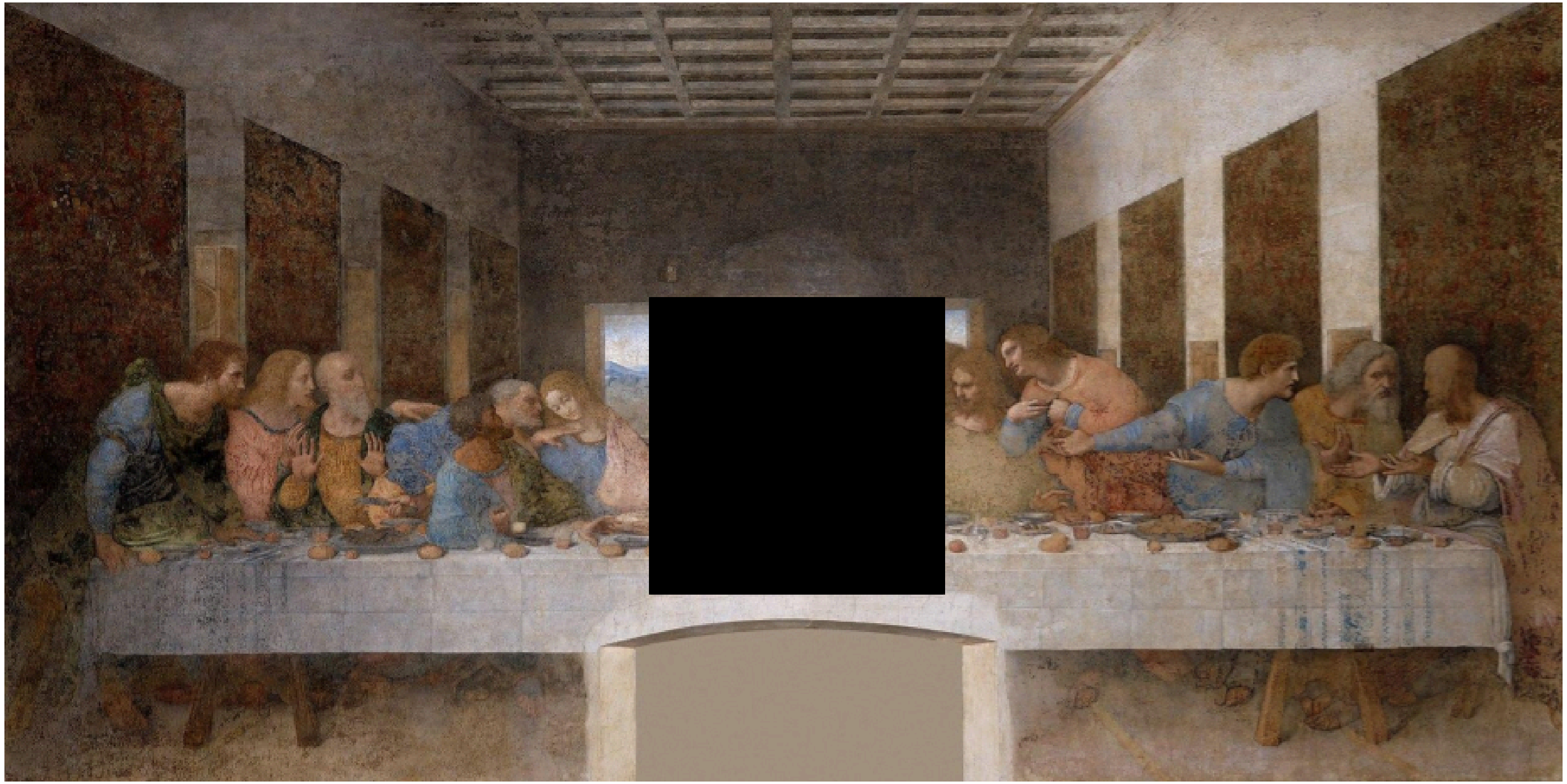
Anyone familiar with Da Vinci's painting *The Last Supper*?

Generative Pre-Training

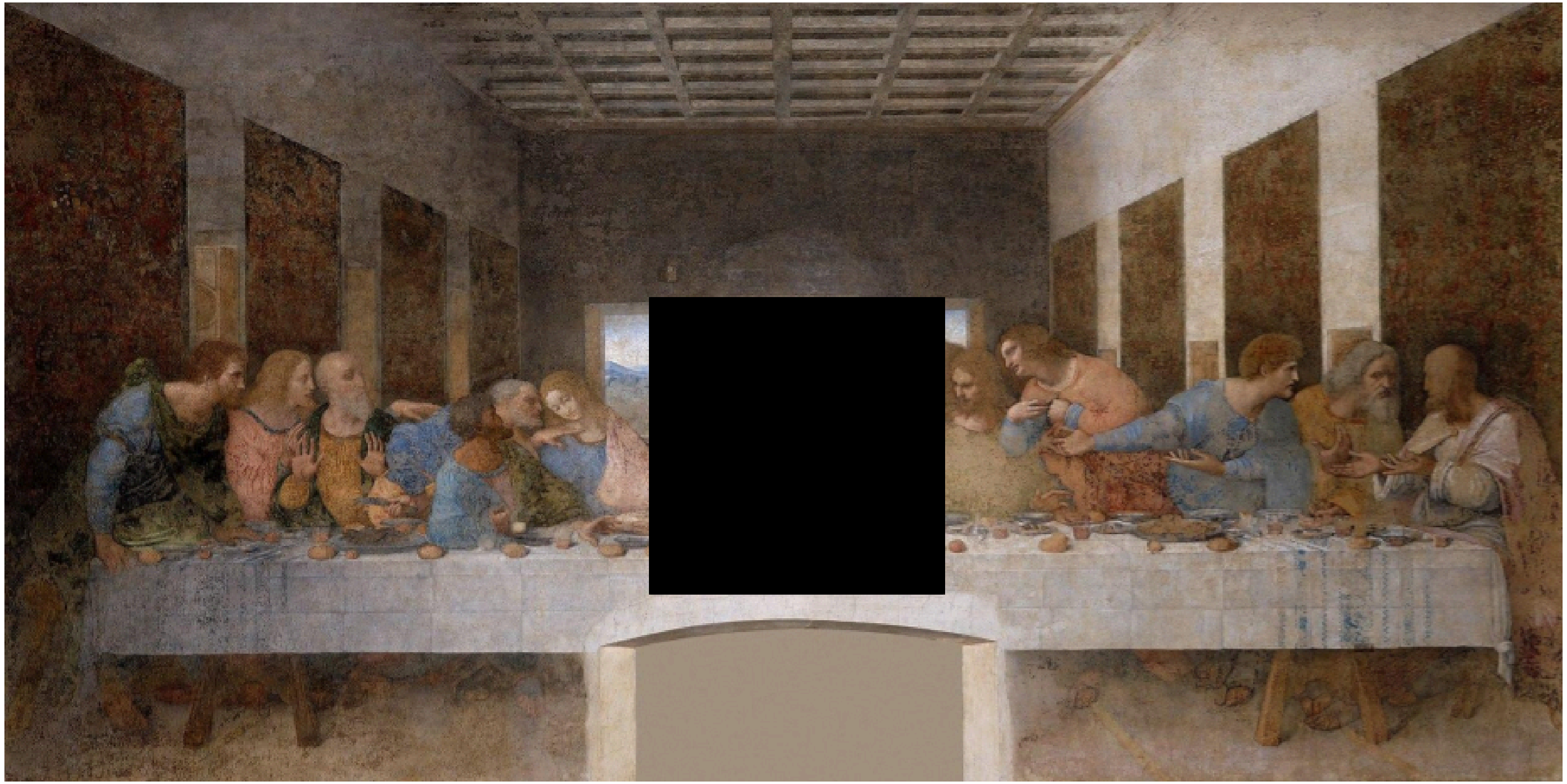
Anyone familiar with Da Vinci's painting *The Last Supper*?



Generative Pre-Training



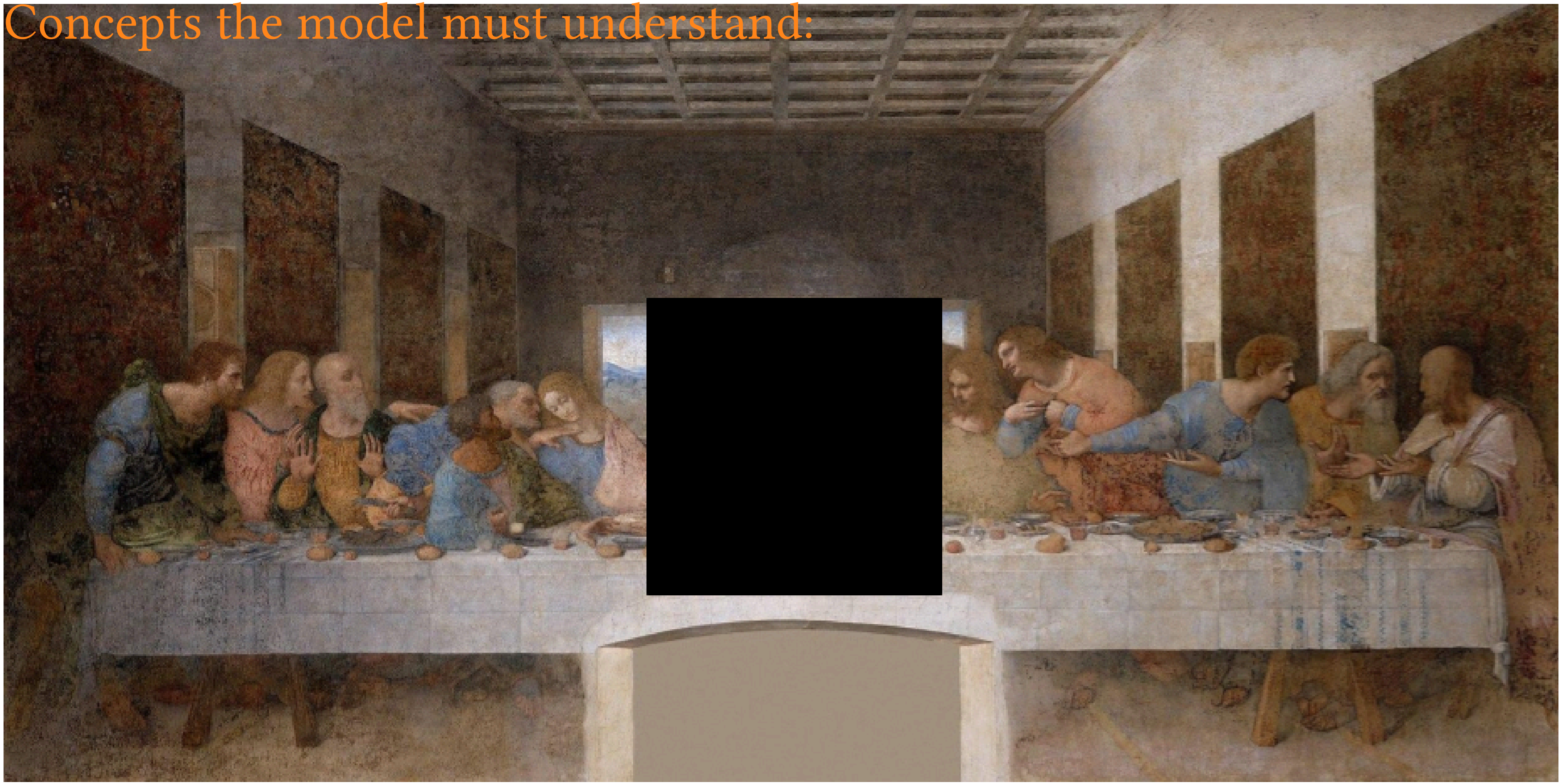
Generative Pre-Training



Question: What concepts does the vision transformer need to understand to predict the missing pixels?

Generative Pre-Training

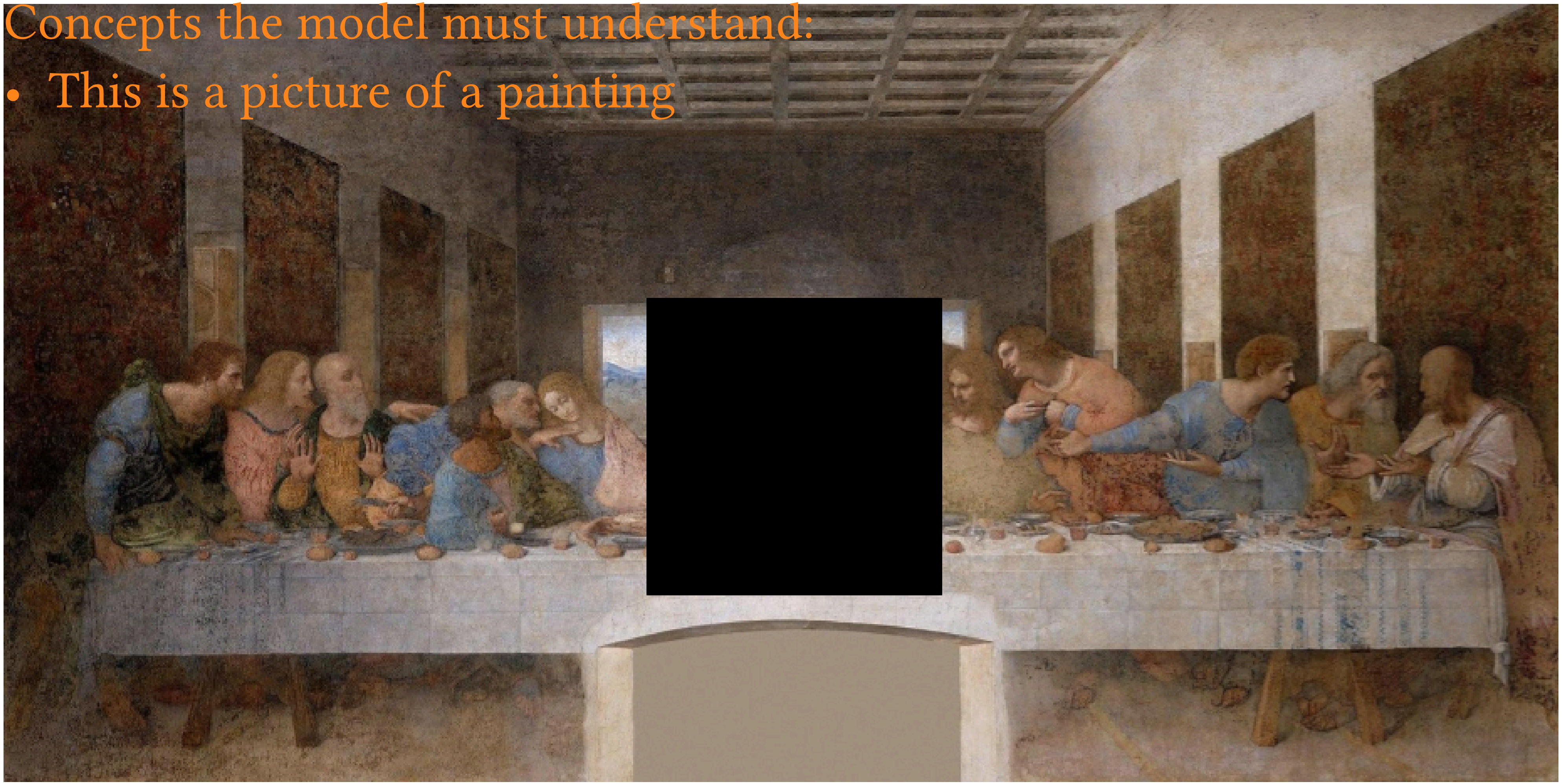
Concepts the model must understand:



Generative Pre-Training

Concepts the model must understand:

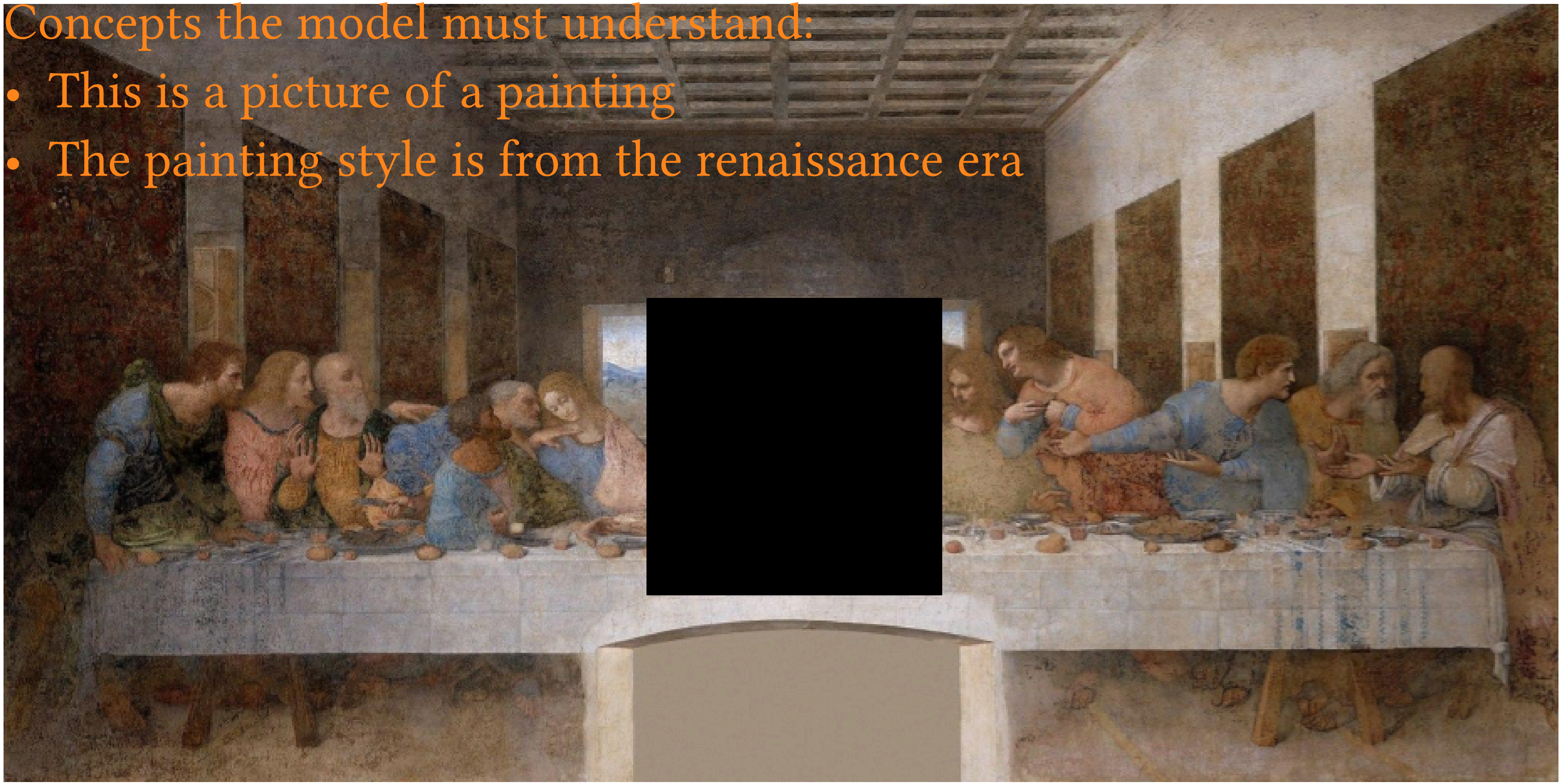
- This is a picture of a painting



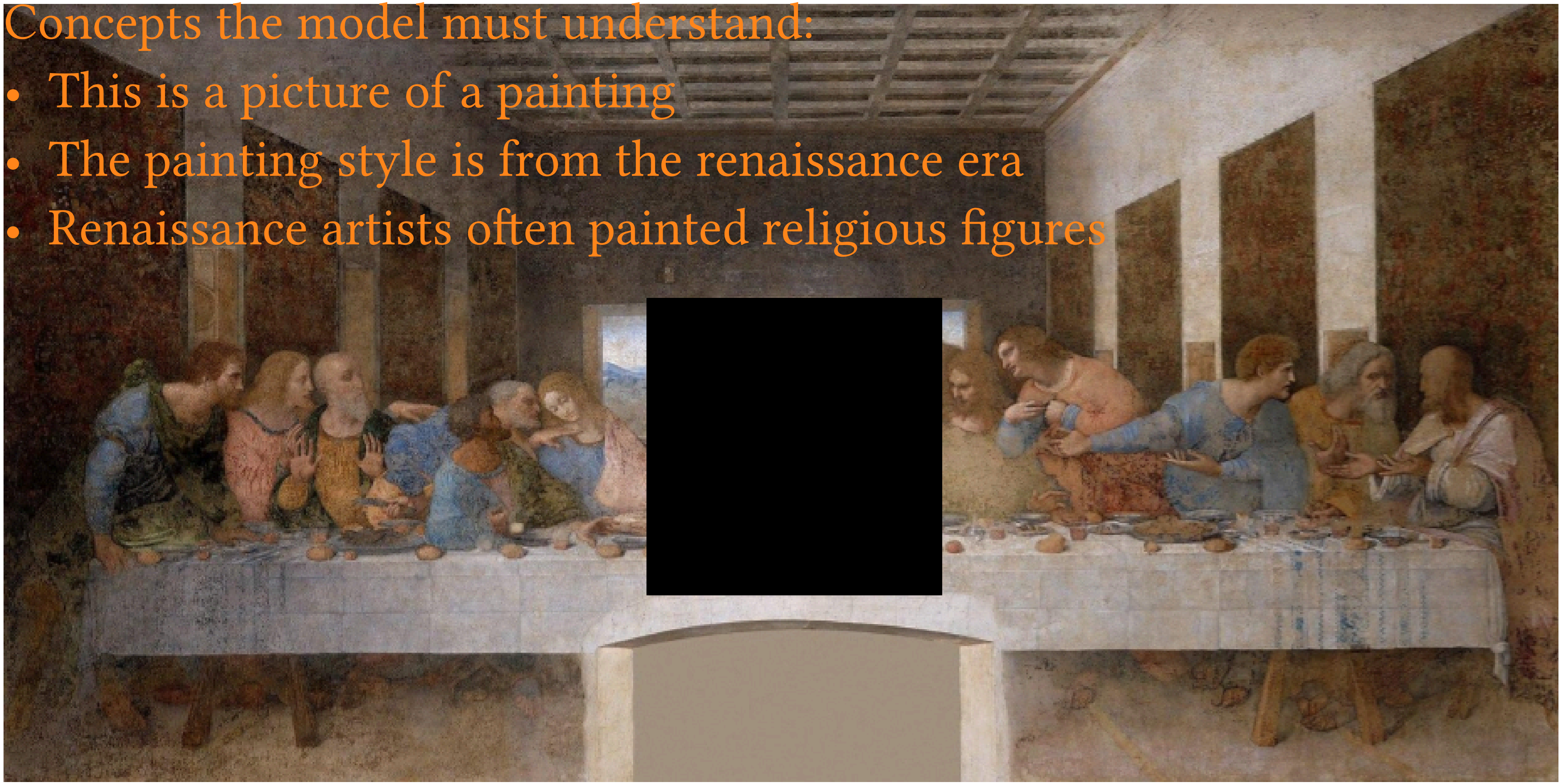
Generative Pre-Training

Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era

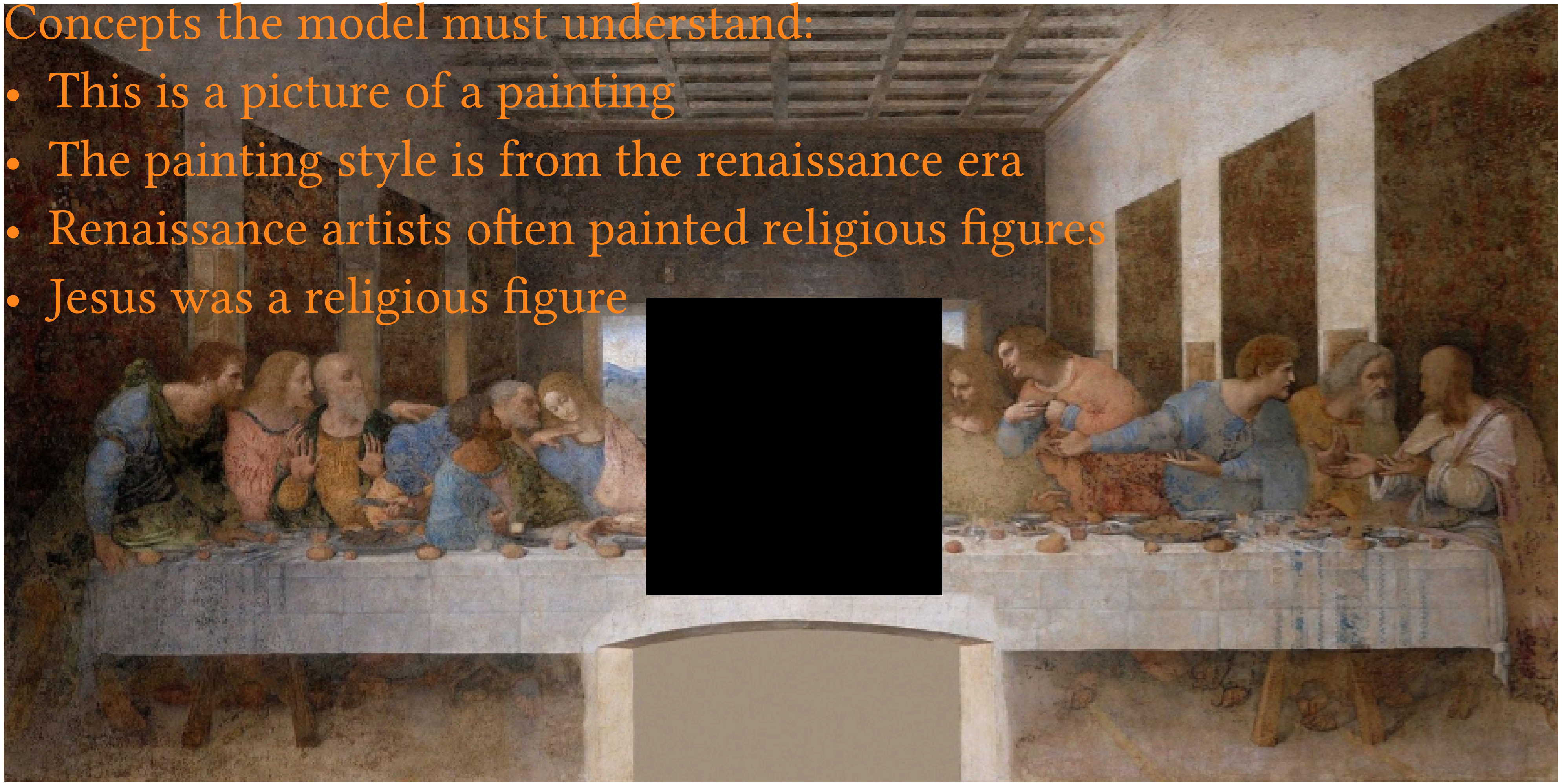


Generative Pre-Training



Concepts the model must understand:
- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures

Generative Pre-Training



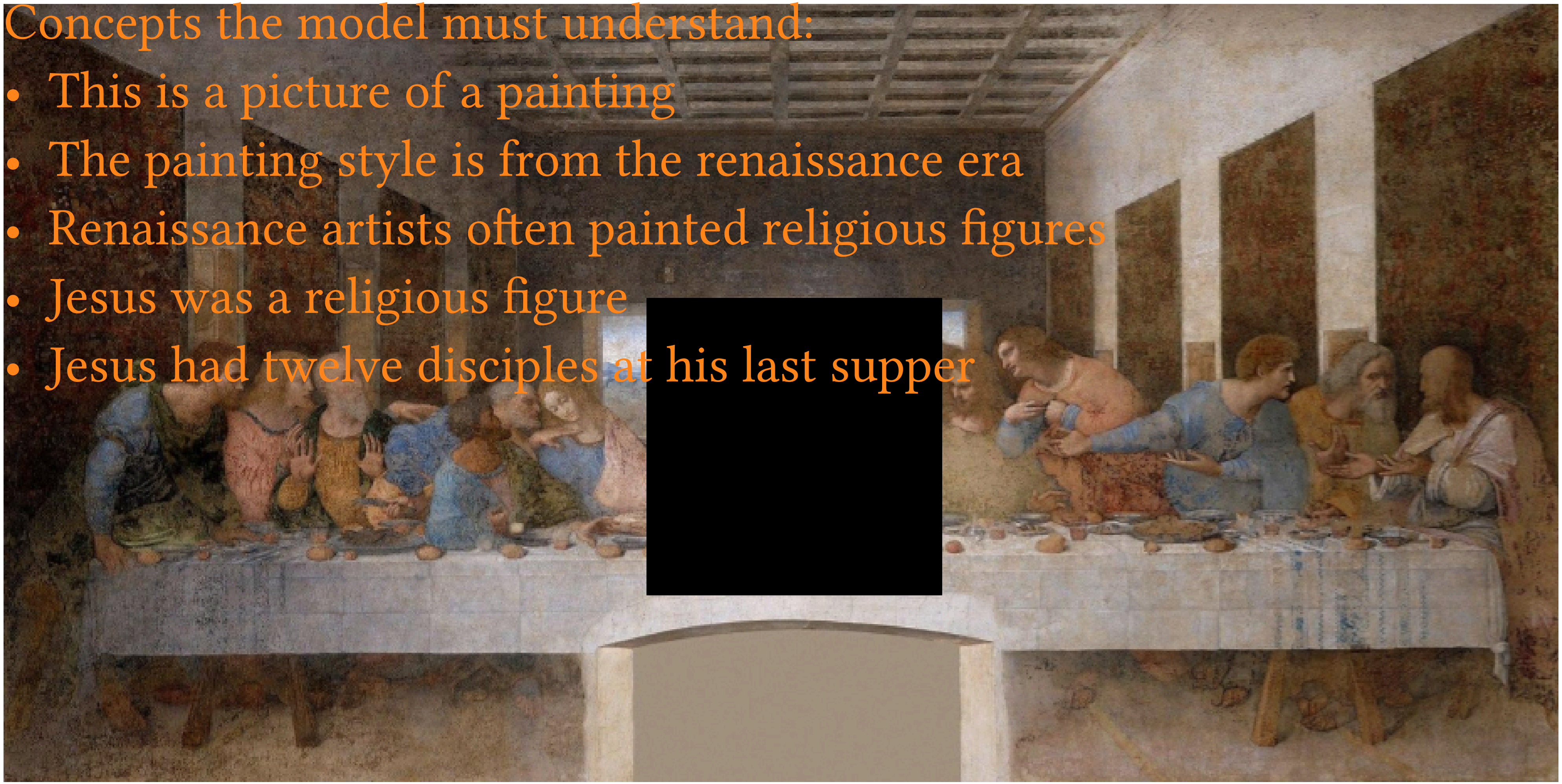
Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures
- Jesus was a religious figure

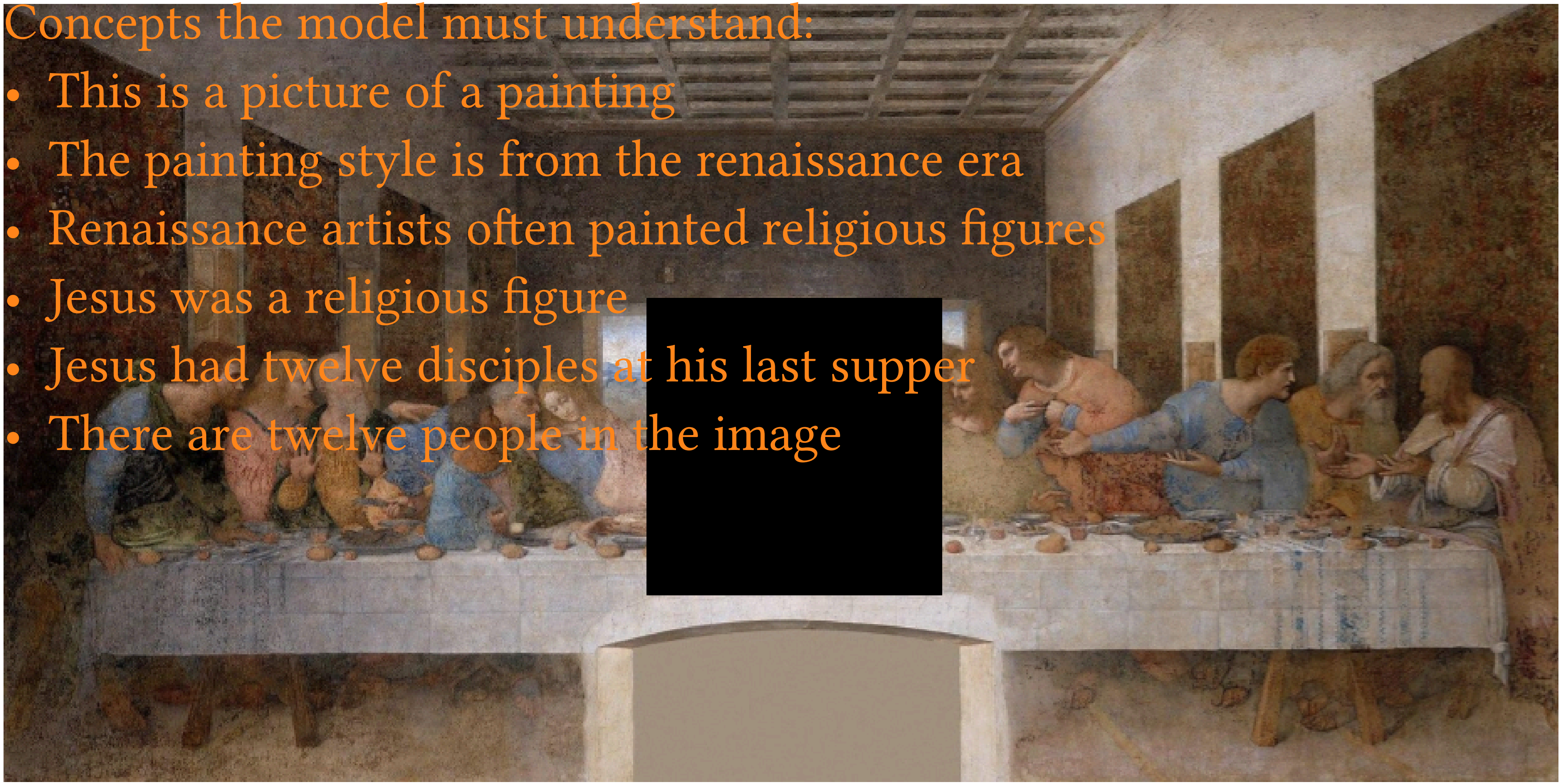
Generative Pre-Training

Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures
- Jesus was a religious figure
- Jesus had twelve disciples at his last supper



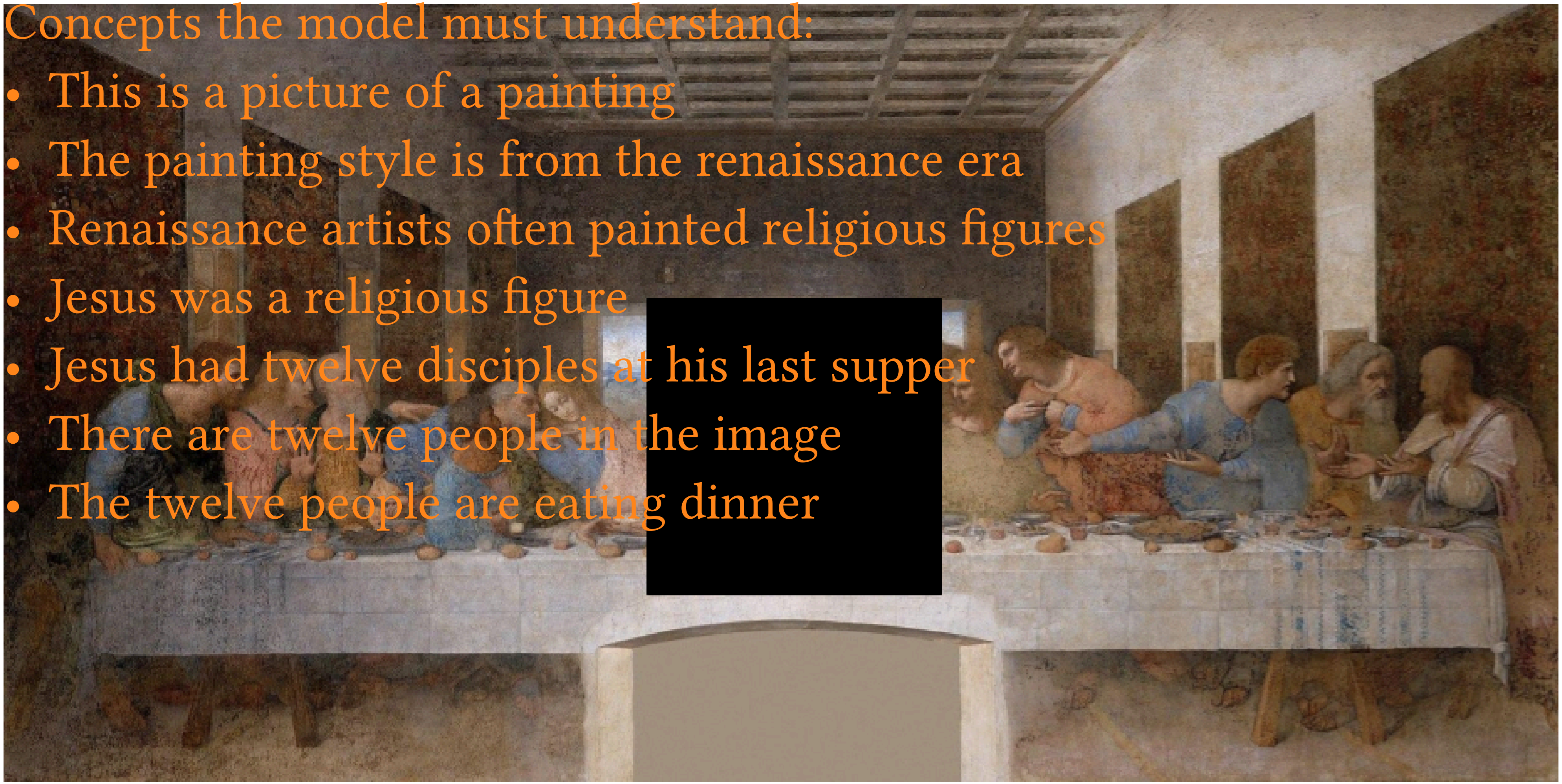
Generative Pre-Training



Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures
- Jesus was a religious figure
- Jesus had twelve disciples at his last supper
- There are twelve people in the image

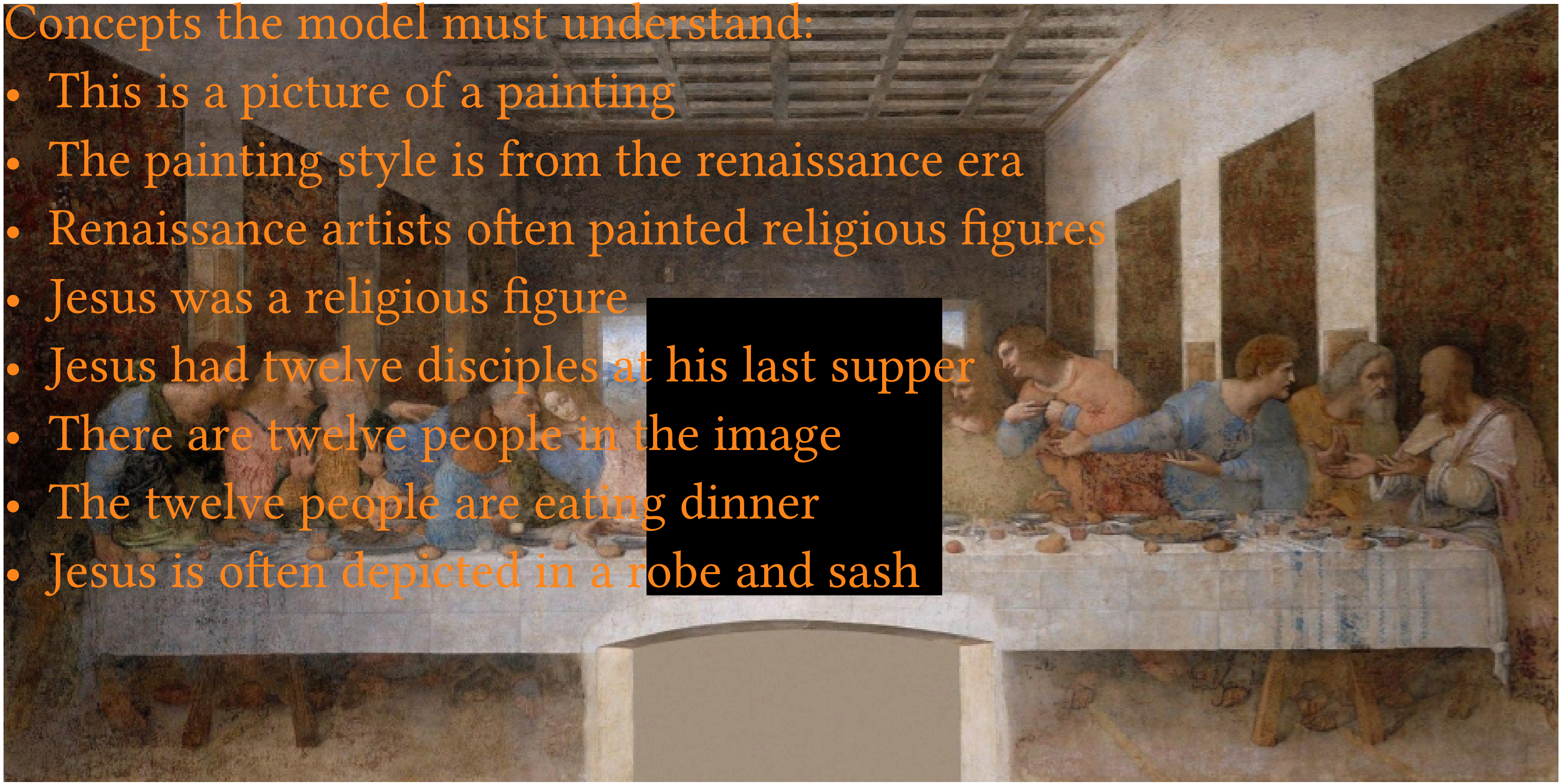
Generative Pre-Training



Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures
- Jesus was a religious figure
- Jesus had twelve disciples at his last supper
- There are twelve people in the image
- The twelve people are eating dinner

Generative Pre-Training



Concepts the model must understand:

- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures
- Jesus was a religious figure
- Jesus had twelve disciples at his last supper
- There are twelve people in the image
- The twelve people are eating dinner
- Jesus is often depicted in a robe and sash

Generative Pre-Training

We train the model to fix the image

Generative Pre-Training

We train the model to fix the image

To fix the image, the model must understand so much of our world

Generative Pre-Training

We train the model to fix the image

To fix the image, the model must understand so much of our world

This is the power of generative pre-training

Generative Pre-Training

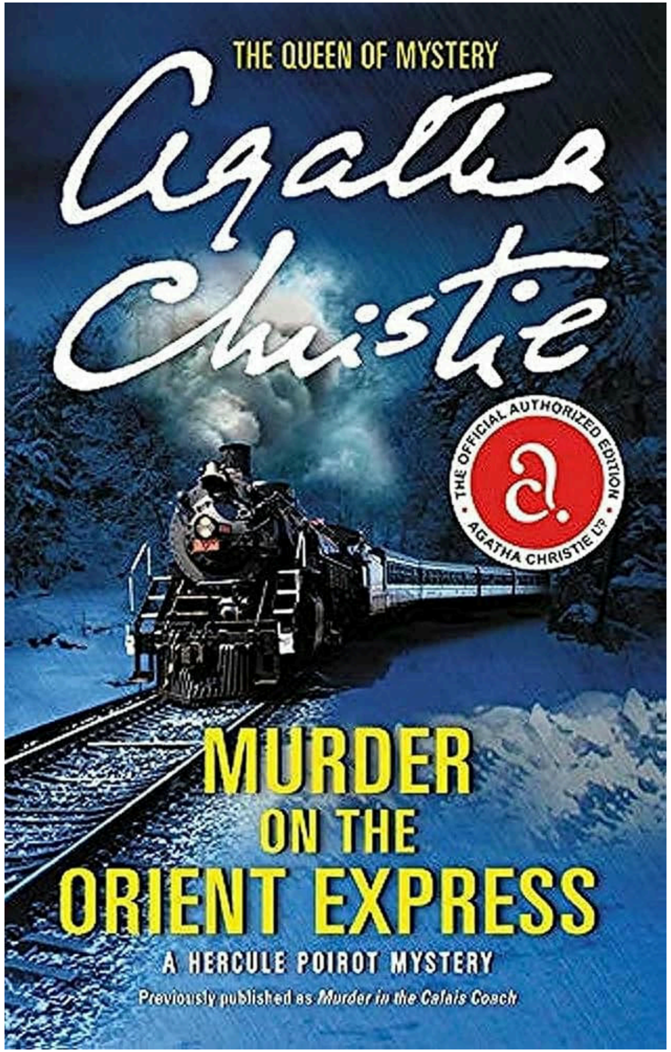
We train the model to fix the image

To fix the image, the model must understand so much of our world

This is the power of generative pre-training

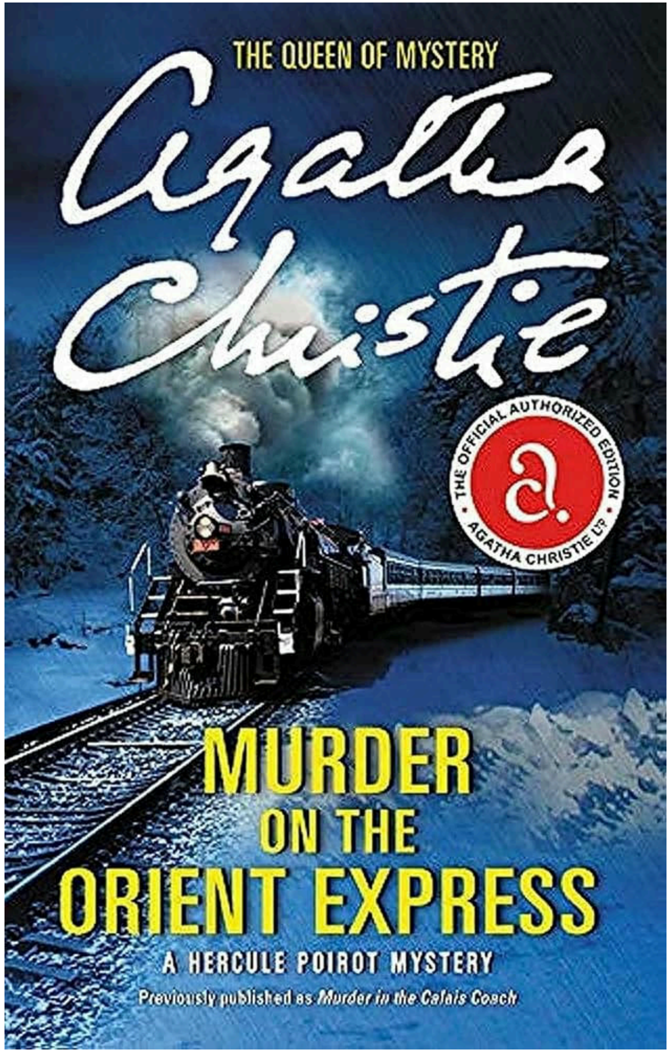
What about text transformers?

Generative Pre-Training



This is a mystery novel

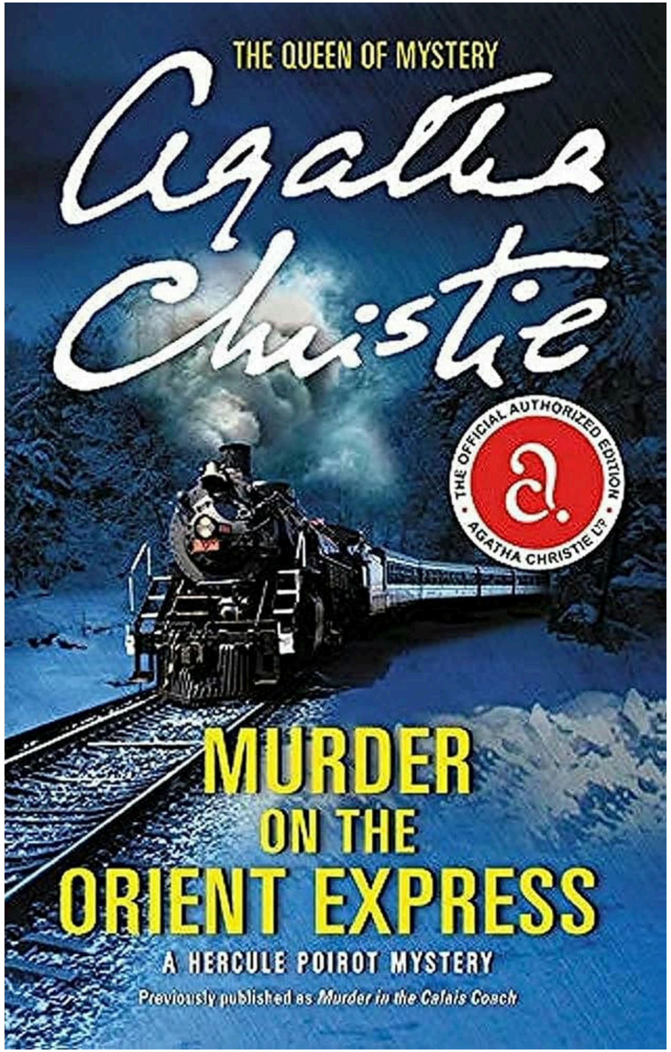
Generative Pre-Training



This is a mystery novel

Clues, intrigue, murder, etc

Generative Pre-Training

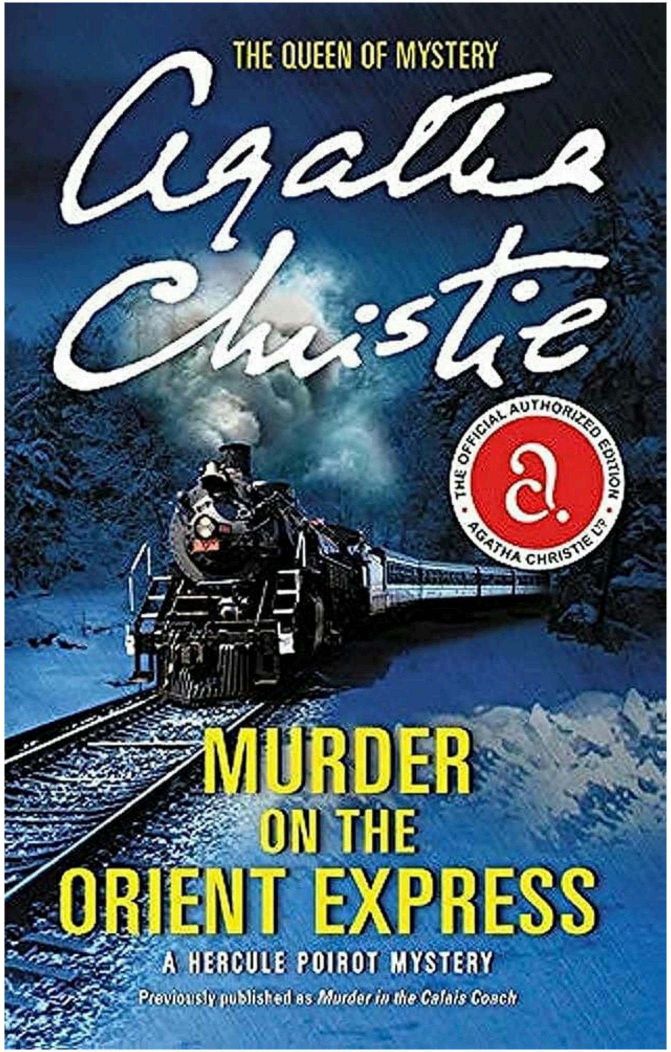


This is a mystery novel

Clues, intrigue, murder, etc

"Ah, said inspector Poirot, the murderer must be ________."

Generative Pre-Training



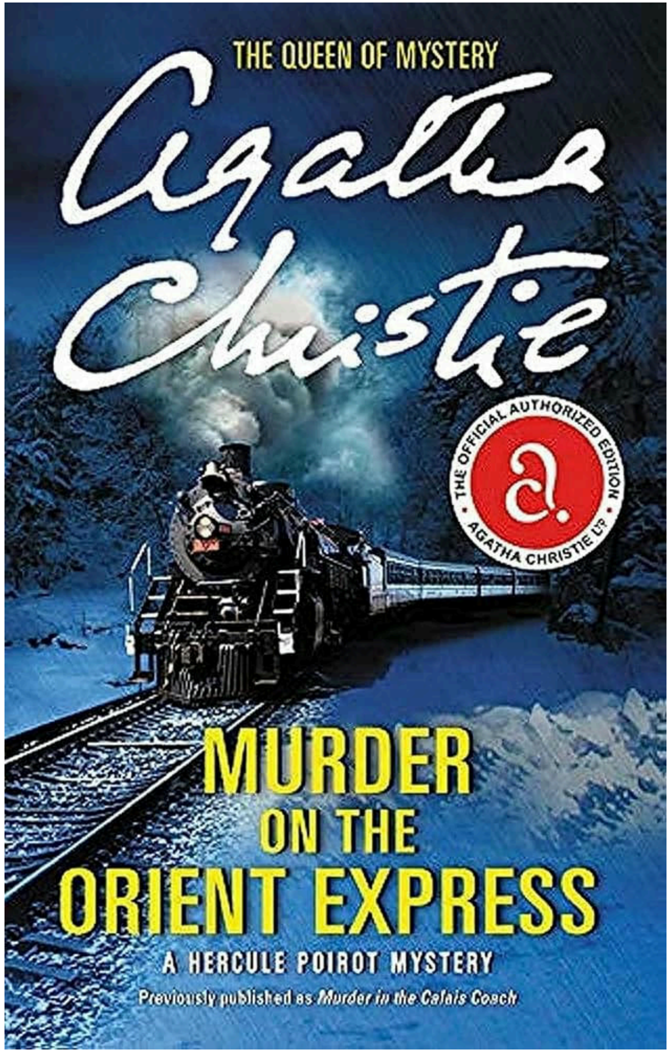
This is a mystery novel

Clues, intrigue, murder, etc

“Ah, said inspector Poirot, the murderer must be ________.”

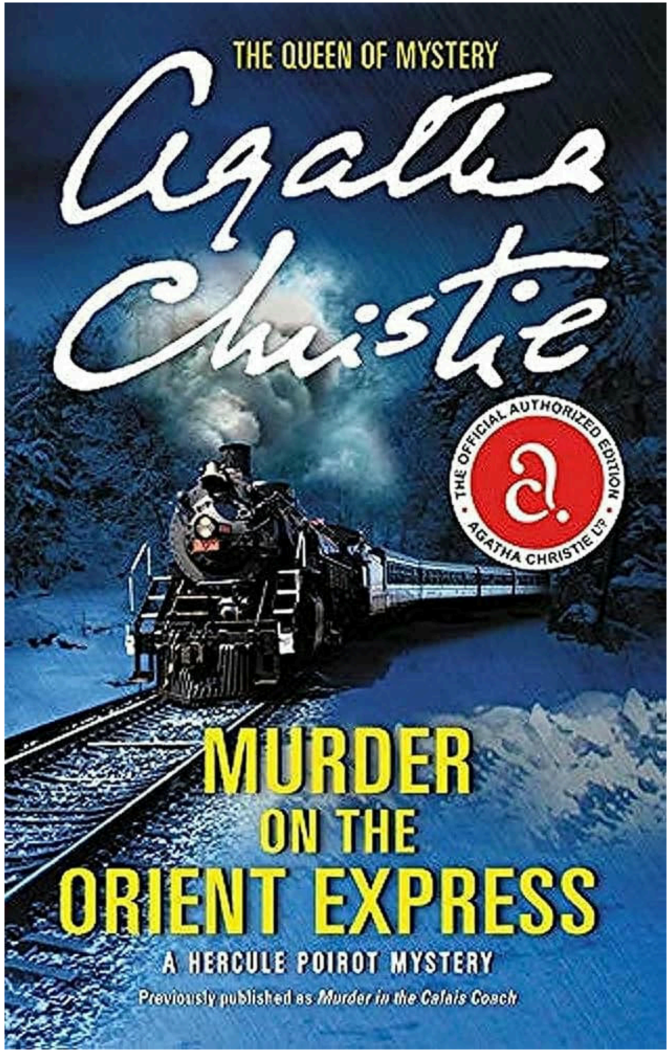
To complete the sentence, the model must understand:

Generative Pre-Training



To complete the sentence, the model must understand:

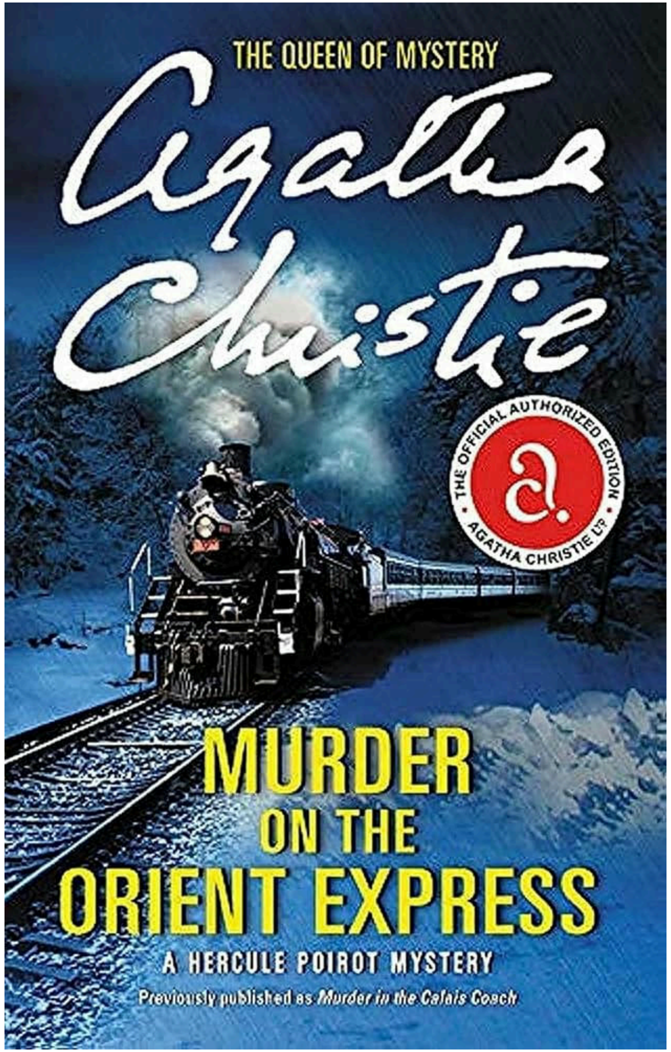
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is

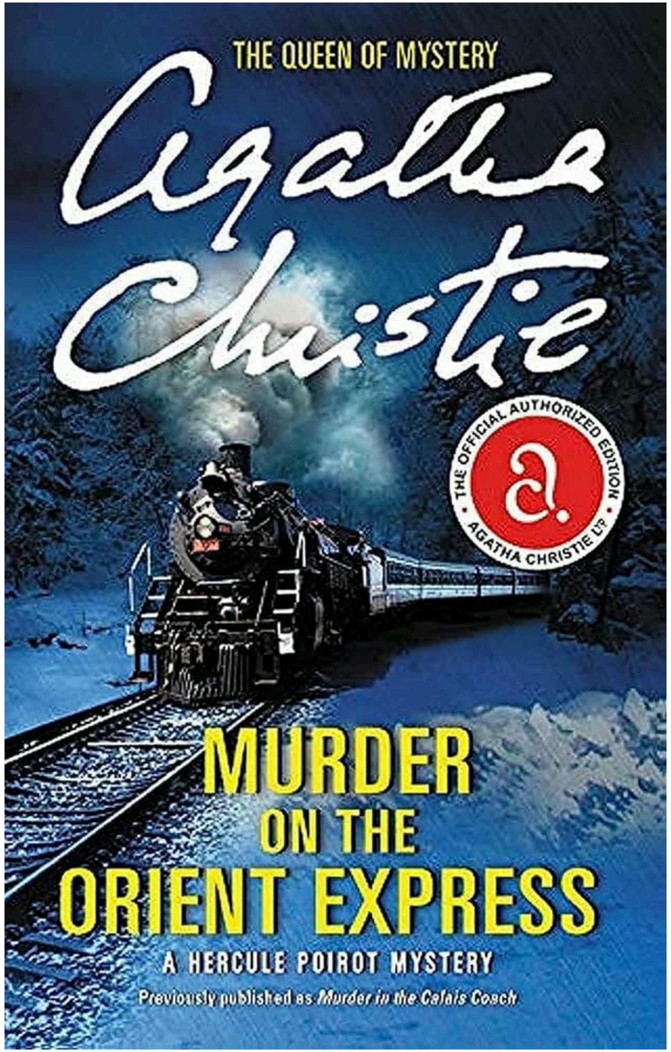
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is
- What it means to be alive

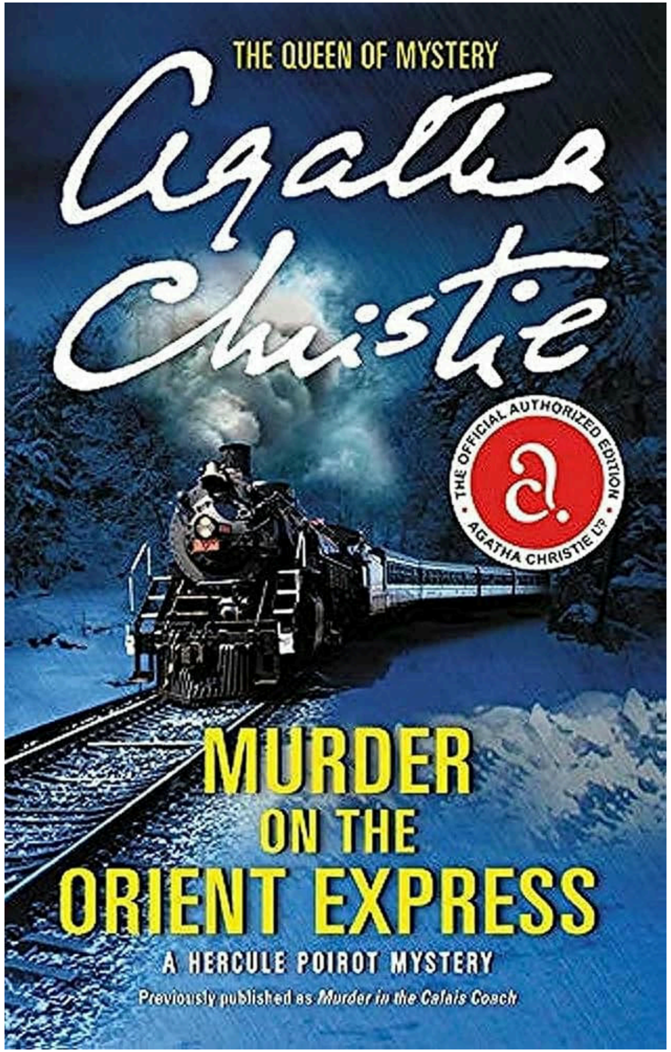
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love

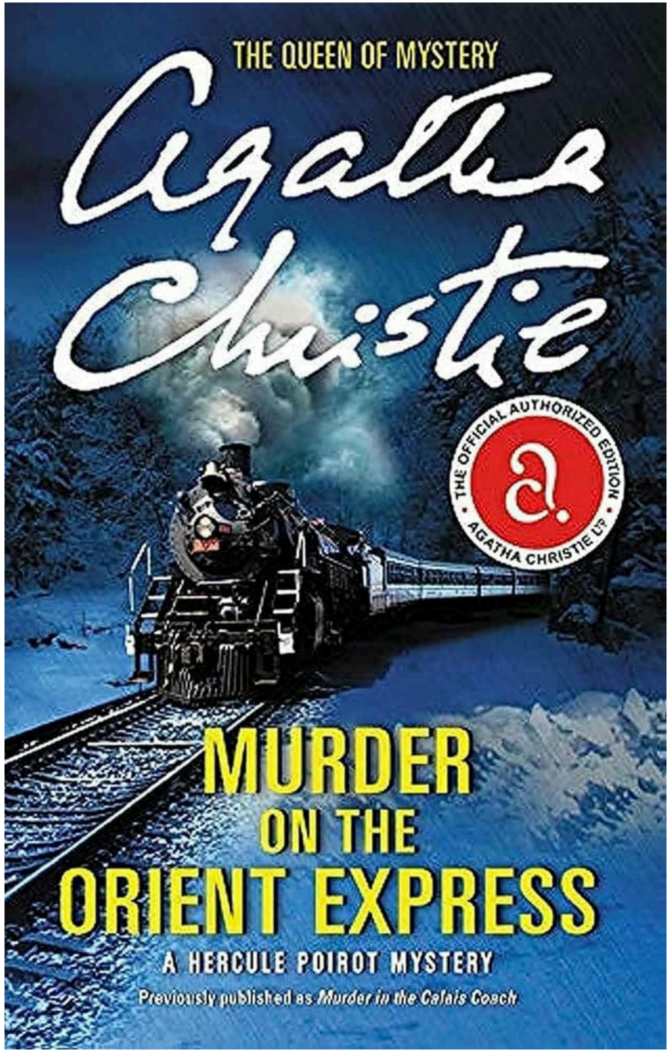
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love
- Personalities of each character

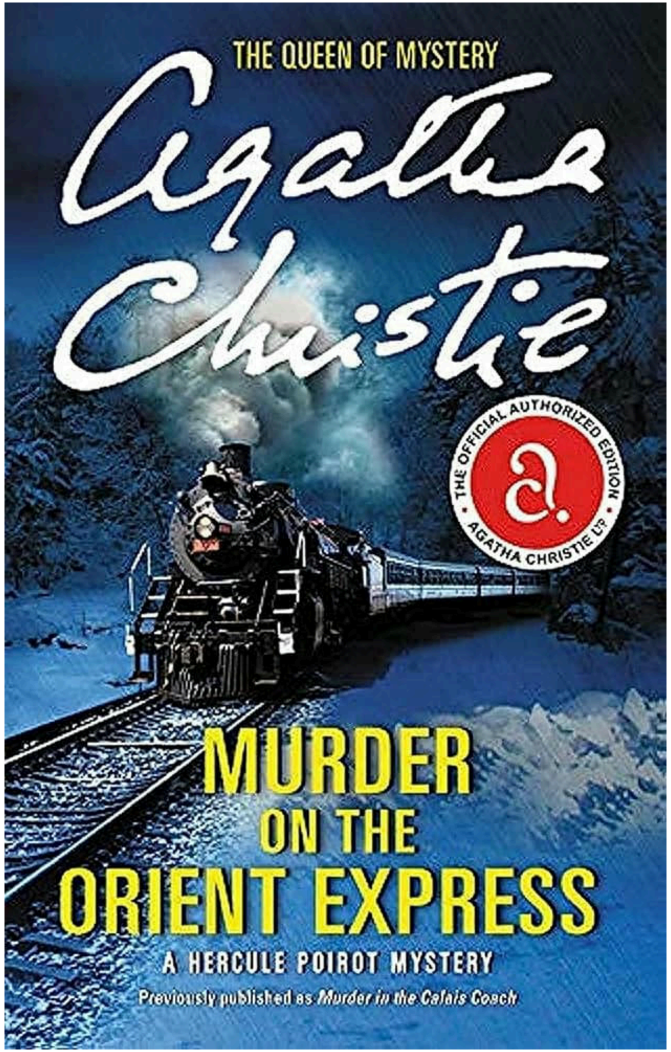
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love
- Personalities of each character
- Why a human would murder another human

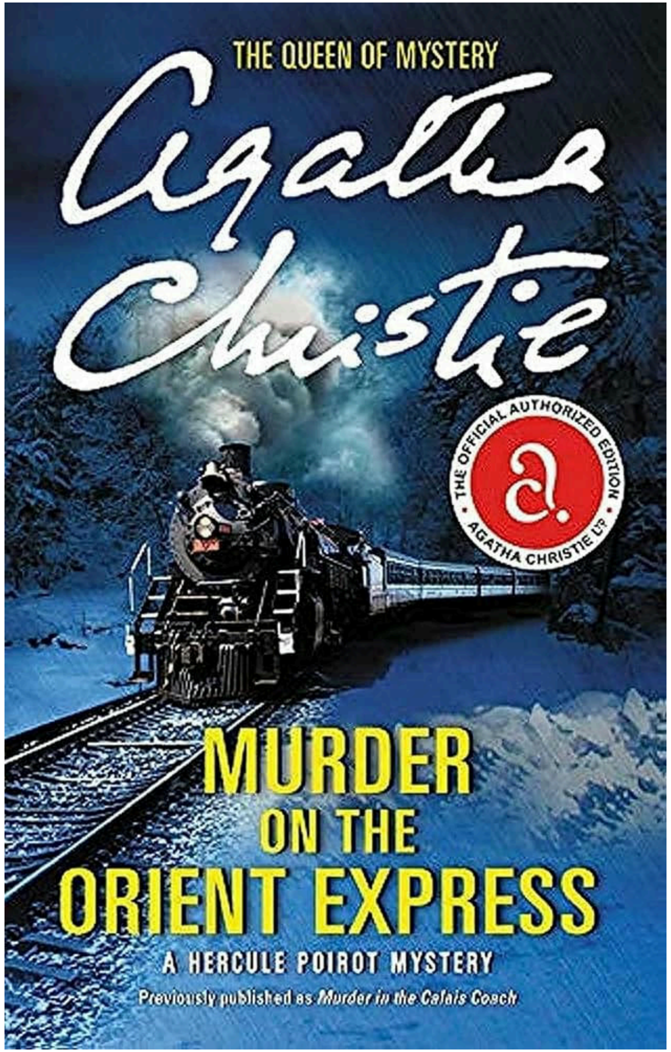
Generative Pre-Training



To complete the sentence, the model must understand:

- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love
- Personalities of each character
- Why a human would murder another human
- How humans react to emotions

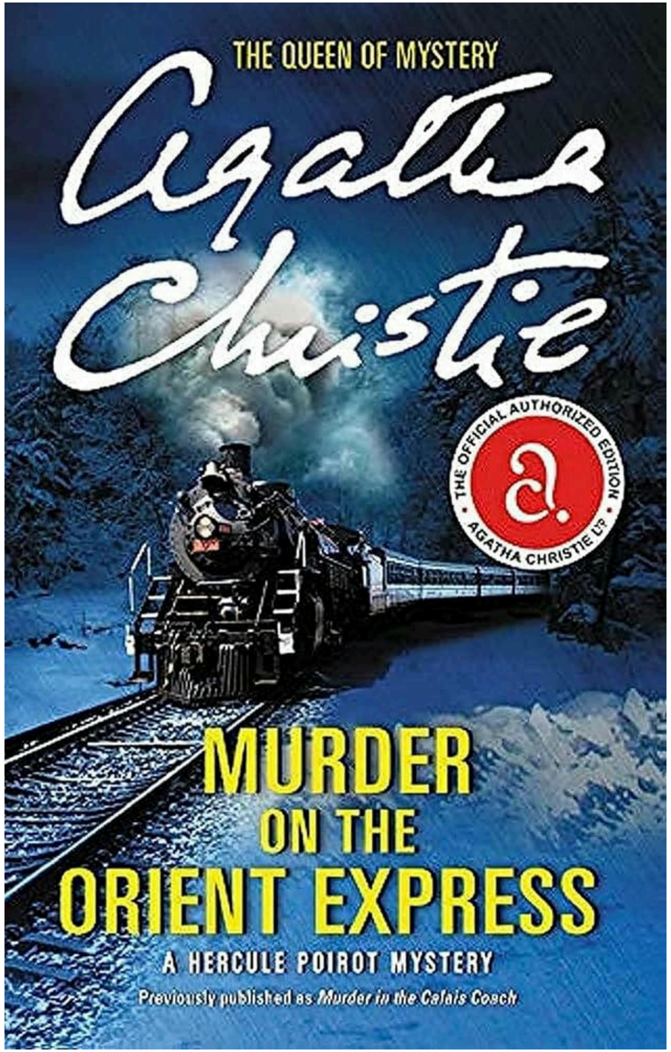
Generative Pre-Training



To complete the sentence, the model must understand:

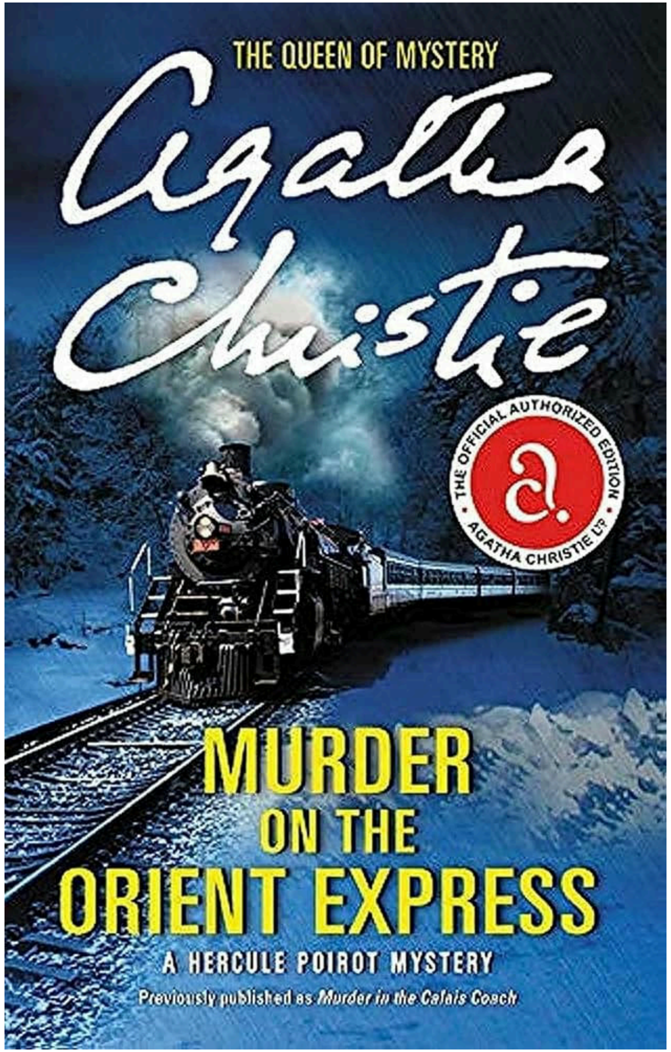
- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love
- Personalities of each character
- Why a human would murder another human
- How humans react to emotions
- How to tell if someone lies

Generative Pre-Training



To predict the murderer, the model must understand so much about humans and our society

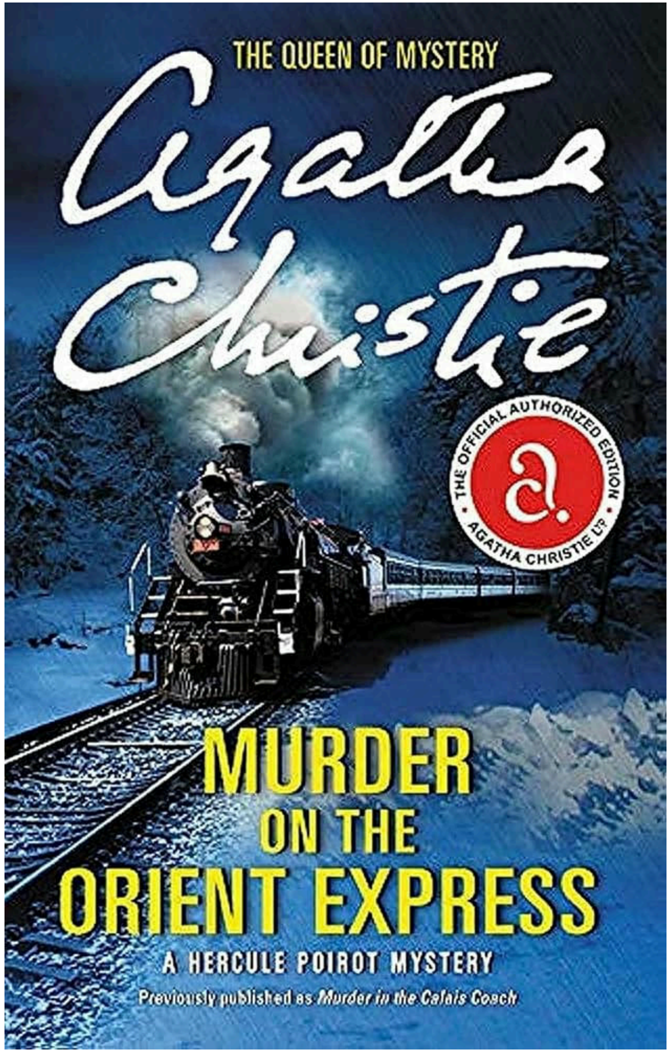
Generative Pre-Training



To predict the murderer, the model must understand so much about humans and our society

The Books3 dataset contains 200,000 books

Generative Pre-Training



To predict the murderer, the model must understand so much about humans and our society

The Books3 dataset contains 200,000 books

We train the model to predict the ending of all these books

Generative Pre-Training

We can apply this same concept to:

Generative Pre-Training

We can apply this same concept to:

- Predict missing base pairs in a strand of DNA

Generative Pre-Training

We can apply this same concept to:

- Predict missing base pairs in a strand of DNA
- Predict missing audio from a song

Generative Pre-Training

We can apply this same concept to:

- Predict missing base pairs in a strand of DNA
- Predict missing audio from a song
- Predict the outcome of particle collisions at the Large Hadron Supercollider

Generative Pre-Training

We can apply this same concept to:

- Predict missing base pairs in a strand of DNA
- Predict missing audio from a song
- Predict the outcome of particle collisions at the Large Hadron Supercollider

All we need is a large enough dataset!

Generative Pre-Training

What if we put the model in a robot?

Generative Pre-Training

What if we put the model in a robot?

Give the model what the robot sees and what the robot does

Generative Pre-Training

What if we put the model in a robot?

Give the model what the robot sees and what the robot does

Predict what the robot will see next

Generative Pre-Training

What if we put the model in a robot?

Give the model what the robot sees and what the robot does

Predict what the robot will see next

Call this a **world model** because it models the structure of our world

Generative Pre-Training

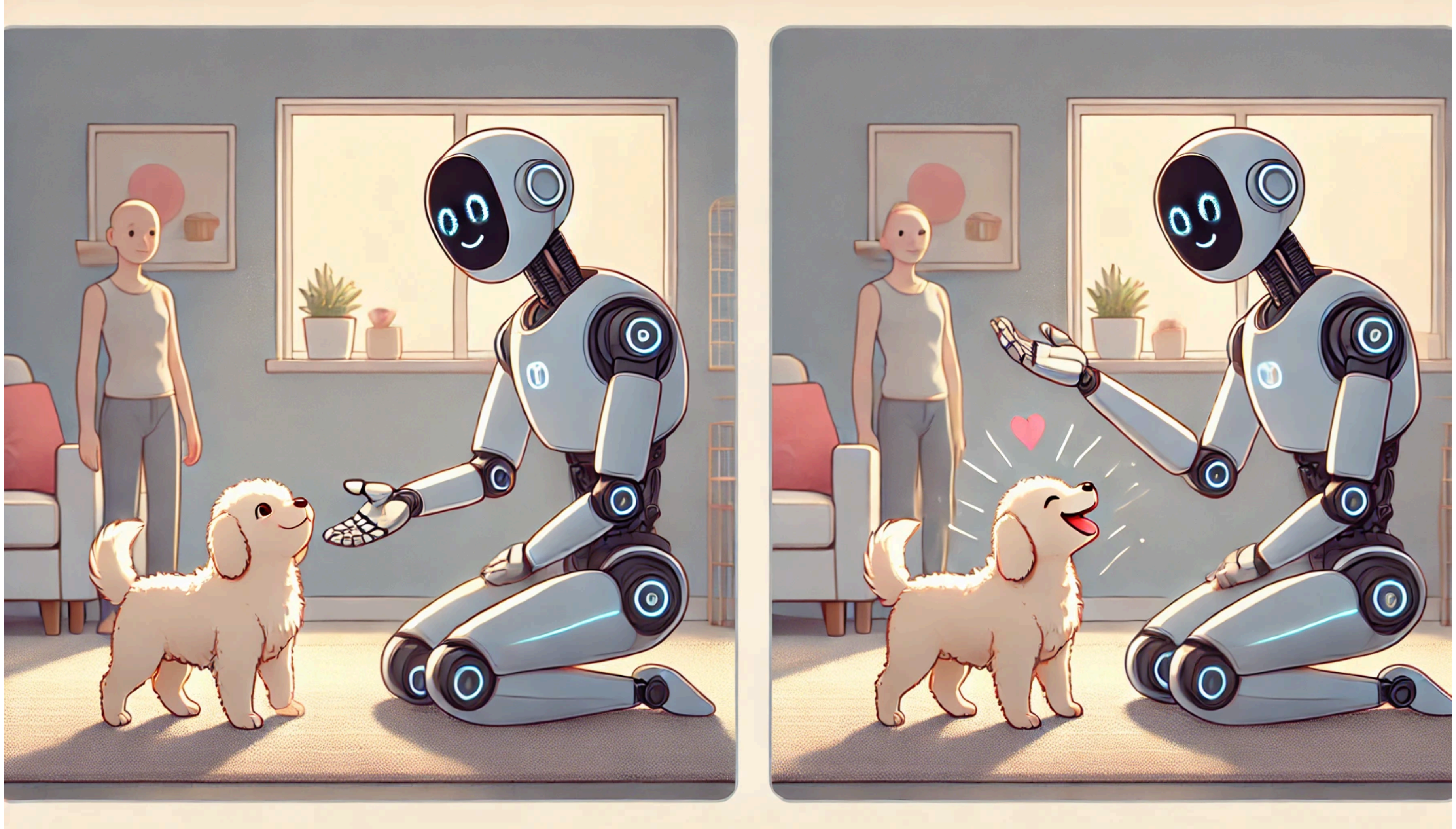
What if we put the model in a robot?

Give the model what the robot sees and what the robot does

Predict what the robot will see next

Call this a **world model** because it models the structure of our world

Generative Pre-Training



Generative Pre-Training

The world model must understand:

Generative Pre-Training

The world model must understand:

- How to control the robot

Generative Pre-Training

The world model must understand:

- How to control the robot
- How to pet a dog

Generative Pre-Training

The world model must understand:

- How to control the robot
- How to pet a dog
- Dogs have feelings

Generative Pre-Training

The world model must understand:

- How to control the robot
- How to pet a dog
- Dogs have feelings
- Petting makes dogs happy

Generative Pre-Training

The world model must understand:

- How to control the robot
- How to pet a dog
- Dogs have feelings
- Petting makes dogs happy
- Dogs smile when happy

Generative Pre-Training

Soon, I will apply for a grant to train a world model

Generative Pre-Training

Soon, I will apply for a grant to train a world model

If I win, I will need help creating a robot dataset

Generative Pre-Training

Soon, I will apply for a grant to train a world model

If I win, I will need help creating a robot dataset

I will need some humans to control our robots in the world

Generative Pre-Training

Soon, I will apply for a grant to train a world model

If I win, I will need help creating a robot dataset

I will need some humans to control our robots in the world

If you are interested, give me your email after class

Generative Pre-Training

These transformers learn and understand the structure of our world

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

How can we use this strong understanding to help humans?

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

How can we use this strong understanding to help humans?

- Identify pictures of cancer

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

How can we use this strong understanding to help humans?

- Identify pictures of cancer
- Make scientific discoveries

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

How can we use this strong understanding to help humans?
- Identify pictures of cancer
- Make scientific discoveries
- Minimize human suffering

Generative Pre-Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

How can we use this strong understanding to help humans?

- Identify pictures of cancer
- Make scientific discoveries
- Minimize human suffering

Today, we use **reinforcement learning**

We will formally introduce reinforcement learning next lecture

Closing Remarks

Closing Remarks

This is the last in-person lecture

Closing Remarks

This is the last in-person lecture

I will record a video on reinforcement learning next week

Closing Remarks

This is the last in-person lecture

I will record a video on reinforcement learning next week

I will be here from 7:00PM on December 2 for questions/discussin on reinforcement learning

Closing Remarks

In this course, we started from Gauss in 1795

Closing Remarks

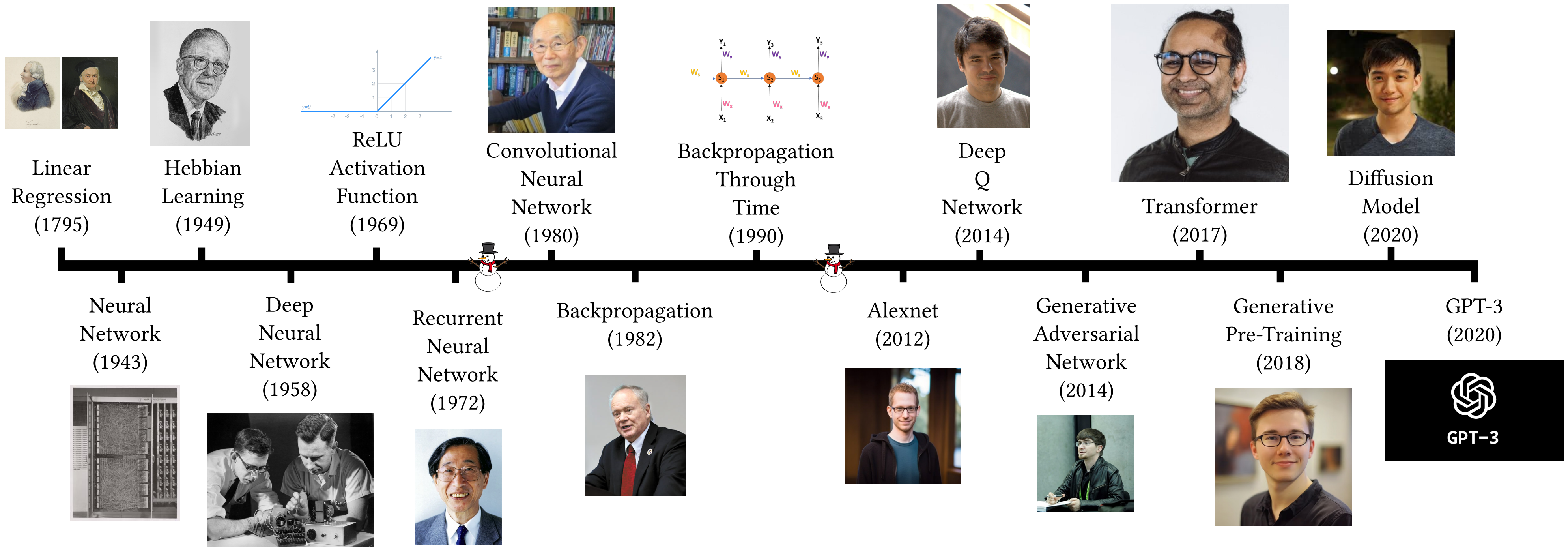
In this course, we started from Gauss in 1795

We built up concepts until we reached the modern age

Closing Remarks

In this course, we started from Gauss in 1795

We built up concepts until we reached the modern age



Closing Remarks

We learned about:

Closing Remarks

We learned about:

- Linear regression

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks
- Autoencoders

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks
- Autoencoders
- Variational autoencoders

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks
- Autoencoders
- Variational autoencoders
- Graph neural networks

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks
- Autoencoders
- Variational autoencoders
- Graph neural networks
- Attention and transformers

Closing Remarks

We learned about:

- Linear regression
- Polynomial regression
- Biological neurons
- Artificial neurons
- Perceptrons
- Backpropagation
- Gradient descent
- Classification
- Parameter initialization
- Many activation functions
- Stochastic gradient descent
- RMSProp and Adam
- Convolutional neural networks
- Composite memory
- Recurrent neural networks
- Autoencoders
- Variational autoencoders
- Graph neural networks
- Attention and transformers
- Generative pre-training

Closing Remarks

I hope you enjoyed the course!

Closing Remarks

I hope you enjoyed the course!

But there are many more topics to learn!

Closing Remarks

I hope you enjoyed the course!

But there are many more topics to learn!

Now, you have the tools to study deep learning on your own

Closing Remarks

I hope you enjoyed the course!

But there are many more topics to learn!

Now, you have the tools to study deep learning on your own

You have the tools to train neural networks for real problems

Closing Remarks

I hope you enjoyed the course!

But there are many more topics to learn!

Now, you have the tools to study deep learning on your own

You have the tools to train neural networks for real problems

Closing Remarks

In the first lecture, I asked everyone in this class for something

Closing Remarks

In the first lecture, I asked everyone in this class for something

Question: Do you remember what it was?

Closing Remarks

Deep learning is a powerful tool

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art
- Autonomous driving

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art
- Autonomous driving

- Making DeepFakes

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art
- Autonomous driving

- Making DeepFakes
- Weapon guidance systems

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art
- Autonomous driving

- Making DeepFakes
- Weapon guidance systems
- Discrimination

Closing Remarks

Deep learning is a powerful tool

Like all powerful tools, deep learning can be used for good or evil

- COVID-19 vaccine
- Creating art
- Autonomous driving

- Making DeepFakes
- Weapon guidance systems
- Discrimination

Before training a model, think about whether it is good or bad for the world

Course Evaluation

Course Evaluation

Department instructed me to ask you for course feedback

Course Evaluation

Department instructed me to ask you for course feedback

We take this feedback seriously

Course Evaluation

Department instructed me to ask you for course feedback

We take this feedback seriously

Your feedback will impact future courses (and my job)

Course Evaluation

Department instructed me to ask you for course feedback

We take this feedback seriously

Your feedback will impact future courses (and my job)

If you like the course, please say it!

Course Evaluation

Department instructed me to ask you for course feedback

We take this feedback seriously

Your feedback will impact future courses (and my job)

If you like the course, please say it!

Please be specific on what you like and do not like

Course Evaluation

Department instructed me to ask you for course feedback

We take this feedback seriously

Your feedback will impact future courses (and my job)

If you like the course, please say it!

Please be specific on what you like and do not like

Your likes/dislikes will change your future courses

Course Evaluation

I must leave the room to let you fill out this form

Course Evaluation

I must leave the room to let you fill out this form

Please scan the QR code and complete the survey

Course Evaluation

I must leave the room to let you fill out this form

Please scan the QR code and complete the survey

I will return in 10 minutes to see if everyone has finished

Course Evaluation

I must leave the room to let you fill out this form

Please scan the QR code and complete the survey

I will return in 10 minutes to see if everyone has finished

https://isw.um.edu.mo/siaweb

Course Evaluation

If you participated in class come see me after class

Course Evaluation

If you participated in class come see me after class

- Answered a question

Course Evaluation

If you participated in class come see me after class

- Answered a question
- Asked a question