Transformers

CISC 7026 - Introduction to Deep Learning

Steven Morad

University of Macau

Review

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

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

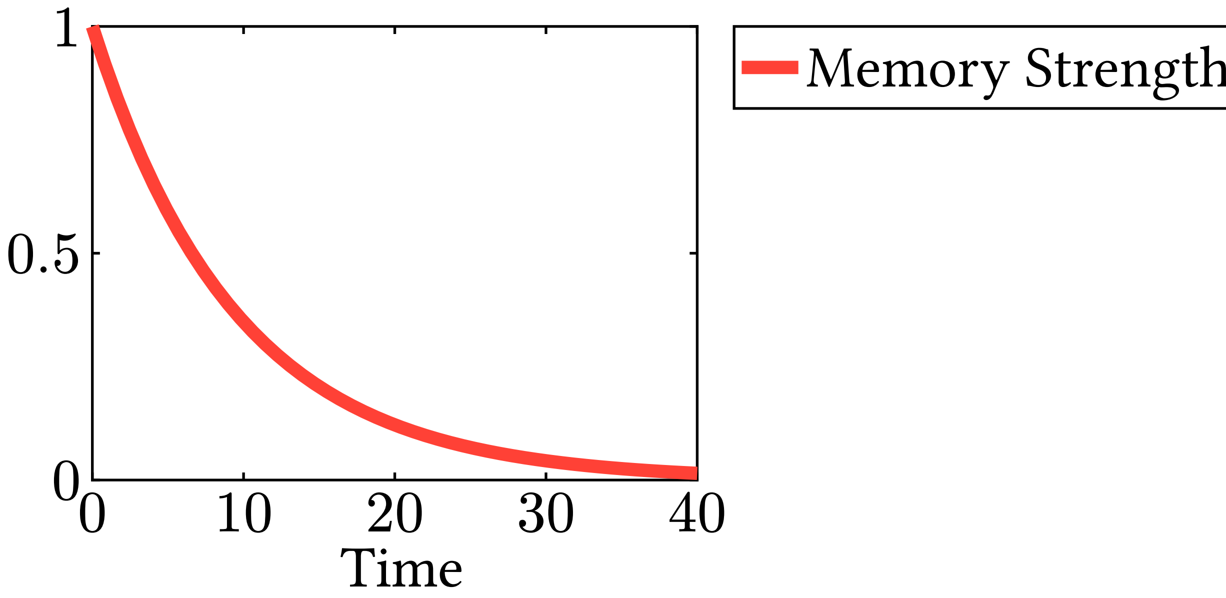
So we introduced a forgetting term γ

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

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

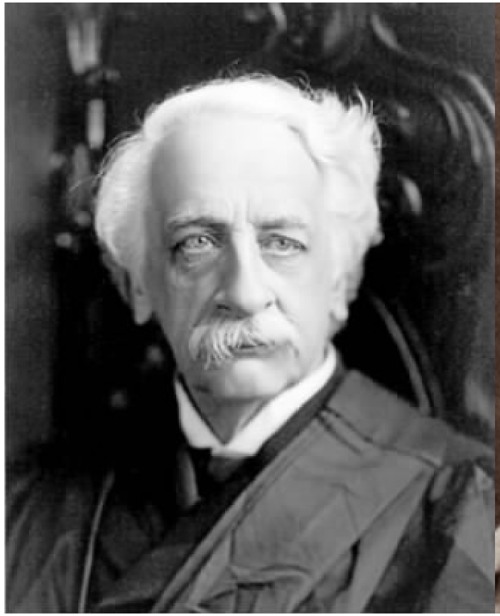
Review

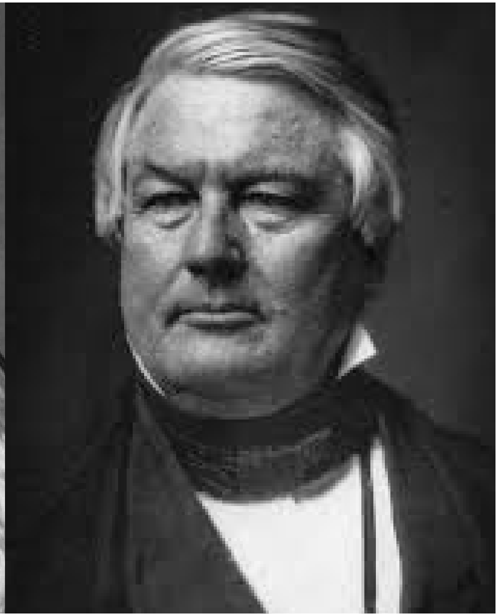
We went to a party and the forgetting seemed ok

10 PM

Review

We went to a party and the forgetting seemed ok



10 PM

11 PM

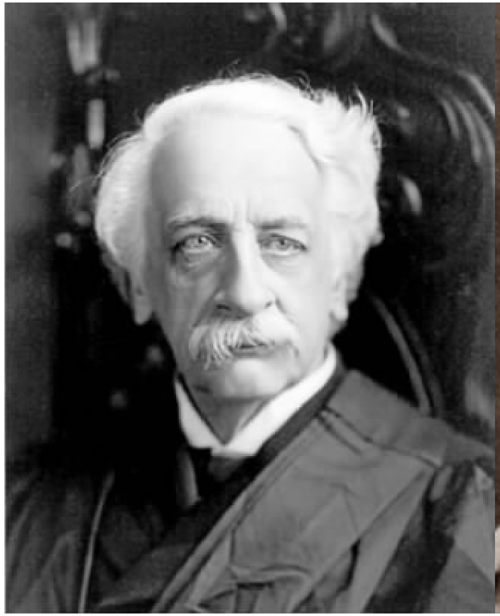
Review

We went to a party and the forgetting seemed ok

10 PM 11 PM 12 AM

Review

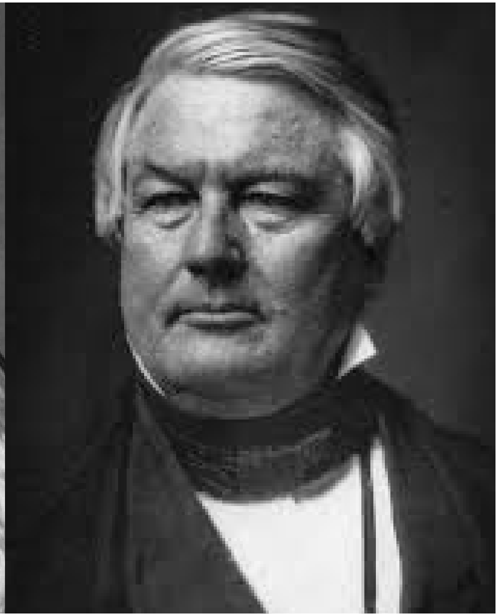
We went to a party and the forgetting seemed ok



10 PM

11 PM

12 AM



1 AM

Review

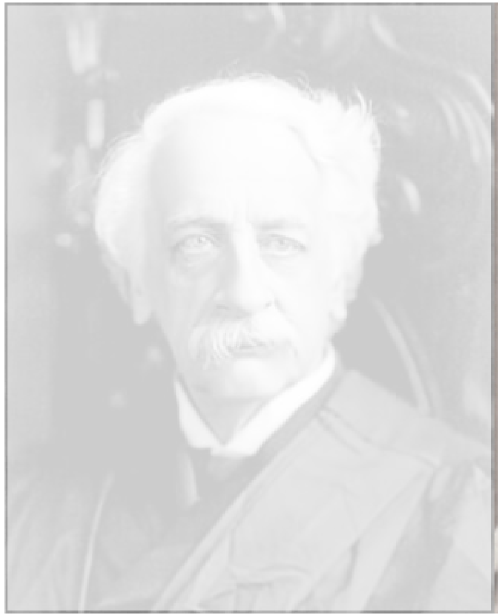
Review

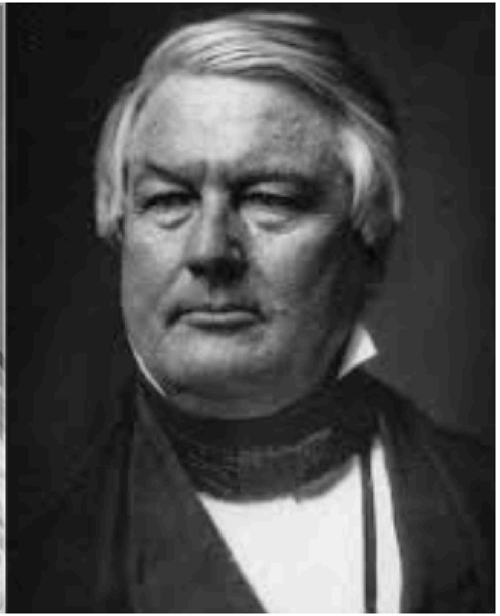
$$\gamma^3 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_1$$

Review

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

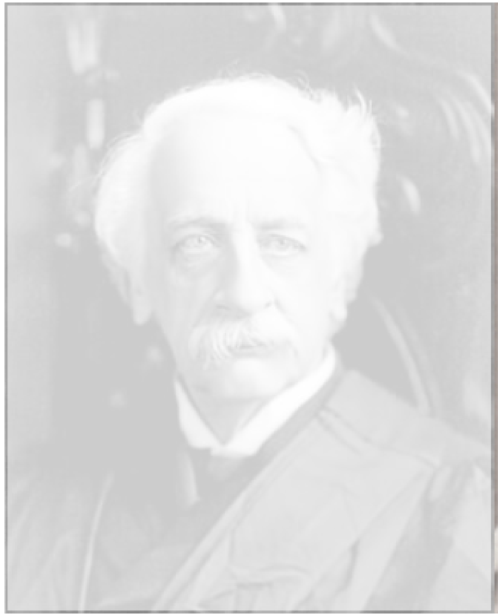
Review





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

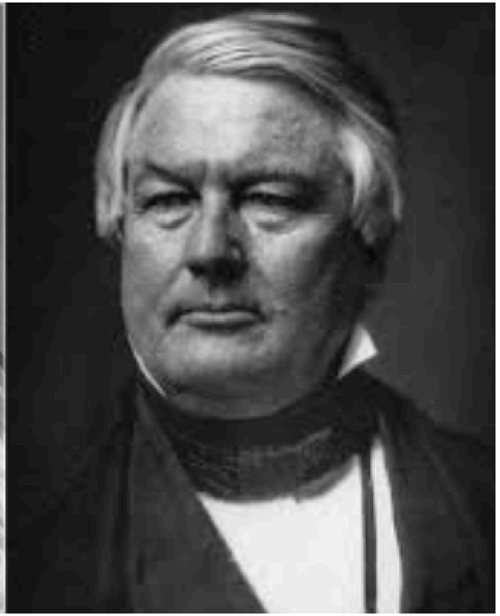
Review



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

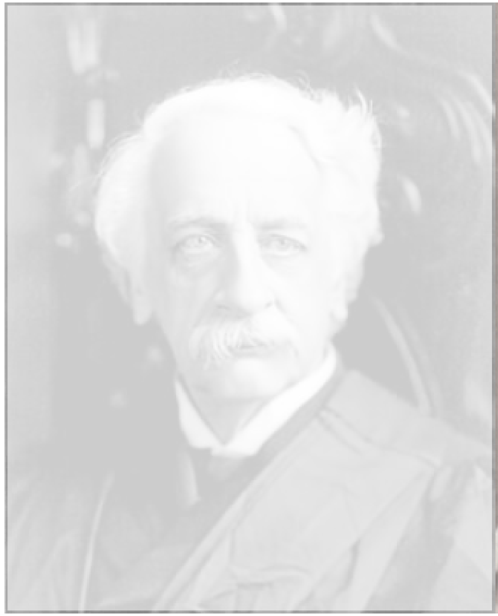
$$\gamma^2 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_2$$

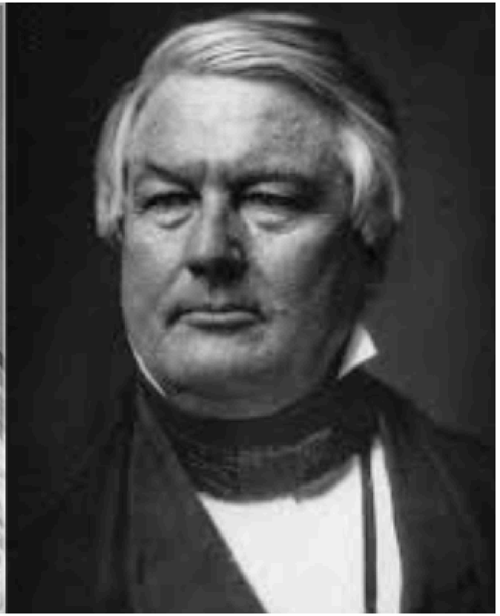
$$\gamma^1 \boldsymbol{\theta}^\top \overline{\boldsymbol{x}}_3$$



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

Review



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

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

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

Review

But we encountered problems when Taylor Swift arrived at the party

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

Review

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

Review

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

With our current model, 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 our current model, we forget Taylor Swift!

Our model of human memory is incomplete

Review

Review

Last time we studied attention

Review

Last time we studied attention

Overview of transformer application and domains

Going Deeper

Going Deeper

We previously reviewed training tricks

Going Deeper

We previously reviewed training tricks

- Deeper networks

Going Deeper

We previously reviewed training tricks

- Deeper networks
- Parameter initialization

Going Deeper

We previously reviewed training tricks

- Deeper networks
- Parameter initialization
- Stochastic gradient descent

Going Deeper

We previously reviewed training tricks

- Deeper networks
- Parameter initialization
- Stochastic gradient descent
- Adaptive optimization

Going Deeper

We previously reviewed training tricks

- Deeper networks
- Parameter initialization
- Stochastic gradient descent
- Adaptive optimization
- Weight decay

Going Deeper

We previously reviewed training tricks

- Deeper networks
- Parameter initialization
- Stochastic gradient descent
- Adaptive optimization
- Weight decay

These methods empirically improve performance, but we do not always understand why

Going Deeper

Modern transformers can be very deep

Going Deeper

Modern transformers can be very deep

For this reason, they use two new training tricks to enable very deep models

Going Deeper

Modern transformers can be very deep

For this reason, they use two new training tricks to enable very deep models

- Residual connections

Going Deeper

Modern transformers can be very deep

For this reason, they use two new training tricks to enable very deep models

- Residual connections
- Layer normalization

Going Deeper

Modern transformers can be very deep

For this reason, they use two new training tricks to enable very deep models

- Residual connections
- Layer normalization

Let us introduce these tricks

Going Deeper

Modern transformers can be very deep

For this reason, they use two new training tricks to enable very deep models

- Residual connections
- Layer normalization

Let us introduce these tricks

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

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

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

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

For certain problems, we need deeper networks

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

For certain problems, we need deeper networks

Going Deeper

But there is a limit!

Going Deeper

But there is a limit!

Making the network too deep can hurt performance

Going Deeper

But there is a limit!

Making the network too deep can hurt performance

The theory is that the input information is **lost** somewhere in the network

Going Deeper

But there is a limit!

Making the network too deep can hurt performance

The theory is that the input information is **lost** somewhere in the network

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

Going Deeper

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

Claim: If the input information is present throughout 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 throughout 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 throughout 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?

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 throughout 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?

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

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

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 input information is available, then learning the identity function should be very easy!

Question: How can we prevent the input from getting lost?

Going Deeper

We can feed the input to each layer

Going Deeper

We can feed the input to each layer

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

Going Deeper

We can feed the input 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

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

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

$$\vdots$$

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

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?

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

It requires much fewer parameters than a dense net, but also does not work as well

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

It requires much fewer parameters than a dense net, but also does not work as well

Going Deeper

We call $f(\boldsymbol{x}) + \boldsymbol{x}$ a **residual connection**

Going Deeper

We call $f(\boldsymbol{x}) + \boldsymbol{x}$ a **residual connection**

Instead of learning how to change $\boldsymbol{x}$, f learns what to add to $\boldsymbol{x}$

Going Deeper

We call $f(\boldsymbol{x}) + \boldsymbol{x}$ a **residual connection**

Instead of learning how to change $\boldsymbol{x}$, f learns what to add to $\boldsymbol{x}$

For example, for an identity function we can easily learn

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = 0; \quad f(\boldsymbol{x}, \boldsymbol{\theta}) + x = x$$

Going Deeper

We call $f(\boldsymbol{x}) + \boldsymbol{x}$ a **residual connection**

Instead of learning how to change $\boldsymbol{x}$, f learns what to add to $\boldsymbol{x}$

For example, for an identity function we can easily learn

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = 0; \quad f(\boldsymbol{x}, \boldsymbol{\theta}) + x = x$$

This helps prevent information 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**

Recall that with parameter initialization and weight decay, we ensure the parameters are fairly small

Going Deeper

The second trick is called **layer normalization**

Recall that 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**

Recall that 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

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

Going Deeper

The second trick is called **layer normalization**

Recall that 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

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

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?

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

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

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

$$\approx 100 \cdot 10$$

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

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?

Going Deeper

We can use **layer normalization**

Going Deeper

We can use **layer normalization**

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

Going Deeper

We can use **layer normalization**

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

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

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

Question: What does this do?

Going Deeper

We can use **layer normalization**

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

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

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

Question: What does this do?

Answer: Makes output have zero mean (both positive and negative values)

Going Deeper

We can use **layer normalization**

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

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

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

Question: What does this do?

Answer: Makes output have zero mean (both positive and negative values)

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 outputs

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

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

Now, the output of the layer is normally distributed

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

TODO COLAB

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

A deep neural network consists of many layers

Transformers

Now we have everything we need to implement a transformer

- Attention
- MLP
- Residual connections
- Layer normalization

A deep neural network consists of many layers

A transformer consists of many **transformer blocks**

Transformers

```
class TransformerBlock(nn.Module):
    def __init__(self):
        self.attn = Attention()
        self.mlp = Sequential(
            Linear(d_h, d_h), LeakyReLU(), Linear(d_h, d_h))
        self.norm1 = nn.LayerNorm(d_h)
        self.norm2 = nn.LayerNorm(d_h)

    def forward(self, x):
        x = self.norm1(self.attn(x) + x)
        x = self.norm2(self.mlp(x) + x)
        return x
```

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.block1 = TransformerBlock()
        self.block2 = TransformerBlock()
        self.block3 = TransformerBlock()

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        return x
```

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.block1 = TransformerBlock()
        self.block2 = TransformerBlock()
        self.block3 = TransformerBlock()

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        return x
```

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

Transformers

```python
class Transformer(nn.Module):
    def __init__(self):
        self.block1 = TransformerBlock()
        self.block2 = TransformerBlock()
        self.block3 = TransformerBlock()

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        return x
```

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

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

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?

Answer: 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?

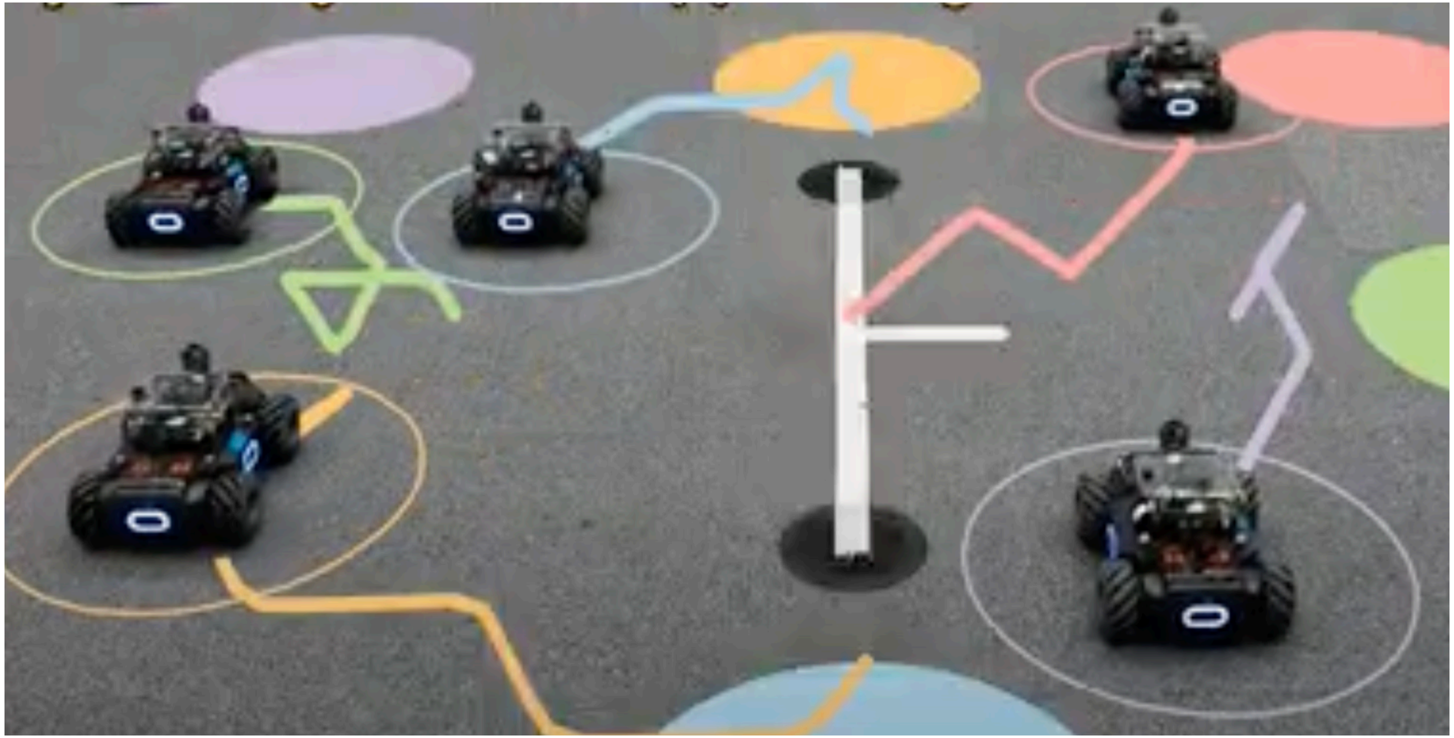
Answer: Yes!

Positional Encoding

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

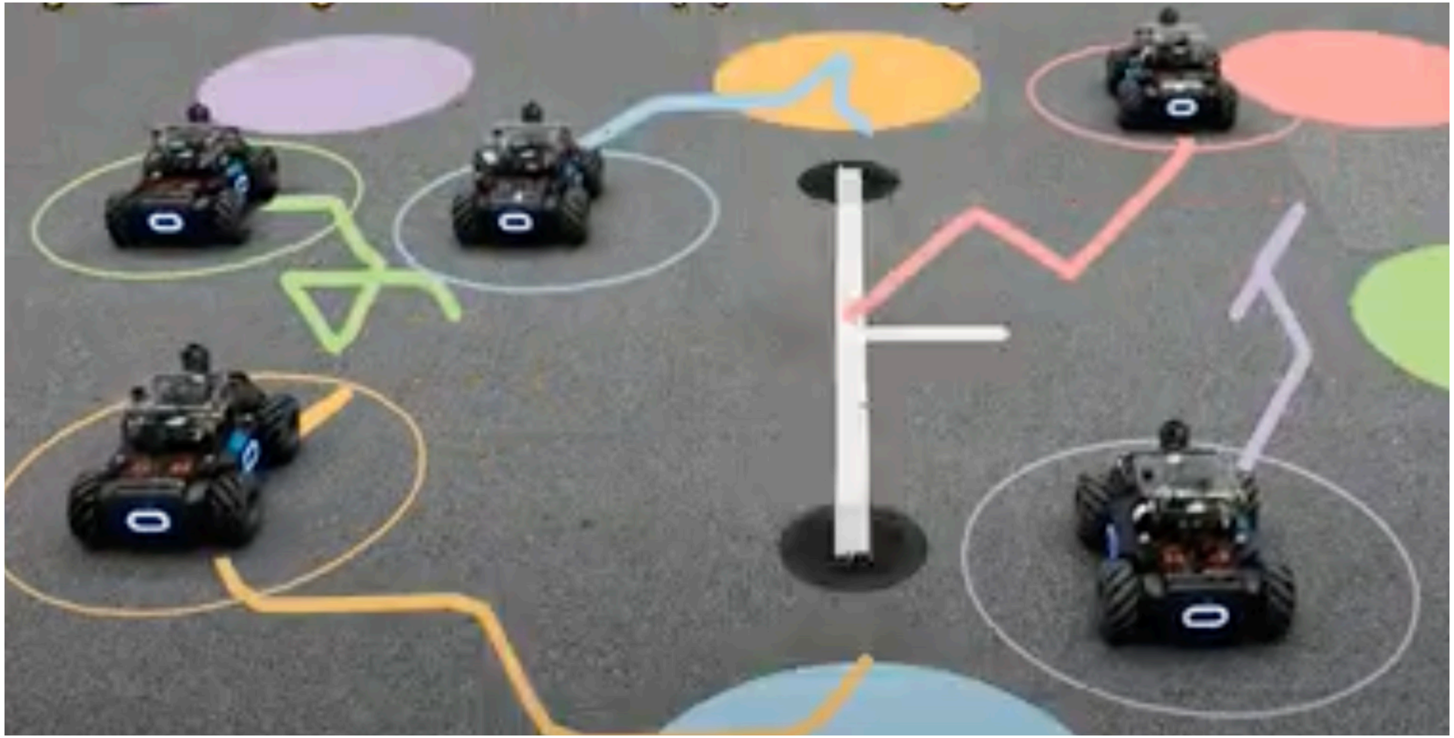
Positional Encoding

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?

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

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 some tasks, we must know the order of the inputs

Positional Encoding

For some tasks, we must know the order of the inputs

Question: Does the transformer know the order of the T inputs?

Positional Encoding

For some tasks, we must know the order of the inputs

Question: Does the transformer know the order of the T inputs?

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

Positional Encoding

For some tasks, we must know the order of the inputs

Question: Does the transformer know the order of the T inputs?

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

Positional Encoding

Example 1:

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

Positional Encoding

Example 1:

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

Positional Encoding

Example 1:

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

Example 2:

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

Positional Encoding

Example 1:

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

Example 2:

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

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

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

f is equivariant, order **does** matter

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 is equivariant, order **does** matter

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

Positional Encoding

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

f is equivariant, order **does** matter

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

f is equivariant, order **does not** matter

Which is a transformer?

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 is equivariant, order **does** matter

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

f is equivariant, order **does not** matter

Which is a transformer?

MLP is equivariant, but what about attention?

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 is equivariant, order **does** matter

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

f is equivariant, order **does not** matter

Which is a transformer?

MLP is equivariant, but what about attention?

Positional Encoding

Recall dot product self attention

Positional Encoding

Recall dot product self attention

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

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

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

Positional Encoding

Recall dot product self attention

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

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

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

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

Positional Encoding

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

Positional Encoding

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

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

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

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

Positional Encoding

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

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

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

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

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

Positional Encoding

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

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

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

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

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

Positional Encoding

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

Positional Encoding

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

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

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

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

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

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

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

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

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

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

Positional Encoding

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

Positional Encoding

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

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

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

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

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

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 is equivariant, order **does not** matter

Positional Encoding

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

Positional Encoding

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

Positional Encoding

This makes sense, in our party 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

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

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

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

Positional Encoding

Question: What are some ways we can introduce ordering?

Answer 1: We can introduce forgetting

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

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

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

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

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right)$$

Positional Encoding

$$\mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) \qquad \mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\mathrm{pos}}(1) \\ f_{\mathrm{pos}}(2) \\ \vdots \\ f_{\mathrm{pos}}(3) \end{bmatrix}, \boldsymbol{\theta} \right)$$

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta} \right) \qquad \operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ f_{\text{pos}}(2) \\ \vdots \\ f_{\text{pos}}(3) \end{bmatrix}, \boldsymbol{\theta} \right)$$

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

Positional Encoding

$$\mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) \qquad \mathrm{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\mathrm{pos}}(1) \\ f_{\mathrm{pos}}(2) \\ \vdots \\ f_{\mathrm{pos}}(3) \end{bmatrix}, \boldsymbol{\theta}\right)$$

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

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

Positional Encoding

$$\operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}, \boldsymbol{\theta}\right) \qquad \operatorname{attn}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix} + \begin{bmatrix} f_{\text{pos}}(1) \\ f_{\text{pos}}(2) \\ \vdots \\ f_{\text{pos}}(3) \end{bmatrix}, \boldsymbol{\theta}\right)$$

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

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

Positional Encoding

What is f_{pos}?

Positional Encoding

What is f_{pos}?

Easiest solution is just some random parameters

Positional Encoding

What is f_{pos}?

Easiest solution is just some random parameters

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

Positional Encoding

What is f_{pos}?

Easiest solution is just some random parameters

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

If ordering matters, learn $\boldsymbol{\theta}$ to be different

Positional Encoding

What is f_{pos}?

Easiest solution is just some random parameters

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

If ordering matters, learn $\boldsymbol{\theta}$ to be different

If order does not matter, learn $\boldsymbol{\theta}$ to be the same/zero

Positional Encoding

What is f_{pos}?

Easiest solution is just some random parameters

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

If ordering matters, learn $\boldsymbol{\theta}$ to be different

If order does not matter, learn $\boldsymbol{\theta}$ to be the same/zero

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

Positional Encoding

So, our final transformer is

```python
class Transformer(nn.Module):
    def __init__(self):
        self.position_embedding = nn.Embedding(d_x)
        self.block1 = TransformerBlock()
        self.block2 = TransformerBlock()

    def forward(self, x):
        x = x + embedding(torch.arange(x.shape[0]))
        x = self.block1(x)
        x = self.block2(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 it to many domains, but today we will focus on text and images

Applications

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

We can apply it to many domains, but today we will focus on text and images

Text Transformers

Text Transformers

In text transformers, we create a parameter for each word

Text Transformers

In text transformers, we create a parameter for each word

We call these parameters **tokens**

Text Transformers

In text transformers, we create a parameter for each word

We call these parameters **tokens**

The simple way is to create one parameter for each unique word in the dataset

Text Transformers

In text transformers, we create a parameter for each word

We call these parameters **tokens**

The simple way is to create one parameter for each unique word in the dataset

Find unique words, and assign each one a parameter

Text Transformers

In text transformers, we create a parameter for each word

We call these parameters **tokens**

The simple way is to create one parameter for each unique word in the dataset

Find unique words, and assign each one a parameter

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

Text Transformers

Then assign a parameter to each word

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

$$\begin{bmatrix} \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \boldsymbol{\theta}_3 \end{bmatrix} =$$

Text Transformers

Then assign a parameter to each word

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

$$[\boldsymbol{\theta}_1 \;\; \boldsymbol{\theta}_2 \;\; \boldsymbol{\theta}_3] = \text{John likes movies}$$

$$[\boldsymbol{\theta}_4 \;\; \boldsymbol{\theta}_2 \;\; \boldsymbol{\theta}_1] =$$

Text Transformers

Then assign a parameter to each word

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

$$[\boldsymbol{\theta}_1 \ \ \boldsymbol{\theta}_2 \ \ \boldsymbol{\theta}_3] = \text{John likes movies}$$

$$[\boldsymbol{\theta}_4 \ \ \boldsymbol{\theta}_2 \ \ \boldsymbol{\theta}_1] = \text{Mary likes John}$$

Text Transformers

```python
unique_words = set(sentence.split(" ") for sentence in xs)
tokens = {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 = embeddings[word]
        token = tokens[index]
        xs.append(token)

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

Text Transformers

```
model = Transformer()
for tokenized_sentence 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 x

Image Transformers

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

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

Image Transformers

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

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

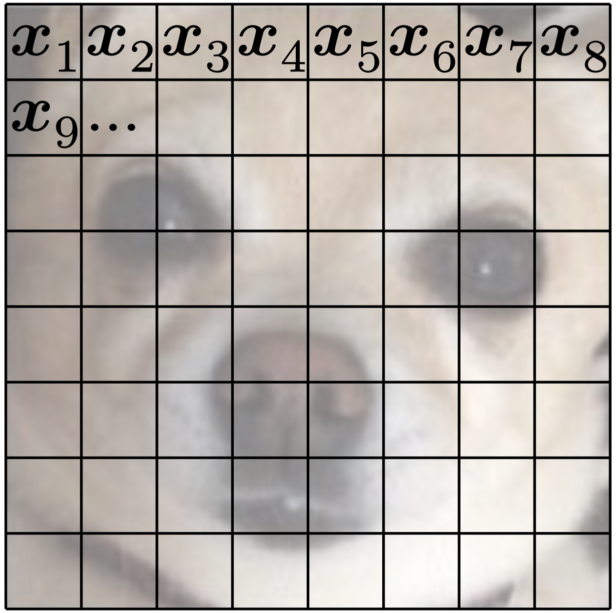


Image Transformers

```python
# 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)
```

Unsupervised Training

Unsupervised Training

Question: How do we train transformers?

Unsupervised Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

Unsupervised Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss

Unsupervised Training

Question: How do we train transformers?

Answer: Can train just like other neural networks

- Classification loss
- Regression loss

Unsupervised 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

Unsupervised 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

There is almost infinite empirical scaling – add more data, model becomes stronger

Unsupervised 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

There is almost infinite empirical scaling – add more data, model becomes stronger

Transformers are limited by the size of datasets today

Unsupervised 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

There is almost infinite empirical scaling – add more data, model becomes stronger

Transformers are limited by the size of datasets today

There are not enough students to label training data!

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Answer: We can use **unsupervised learning**

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

We can mask/hide part of the input, and train the model to predict the missing parts

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Answer: We can use **unsupervised learning**

The internet contains billions of unlabeled sentences and images

We can mask/hide part of the input, and train the model to predict the missing parts

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

Unsupervised Training

Question: How can we train transformers if we cannot create big enough datasets?

Answer: We can use **unsupervised learning**

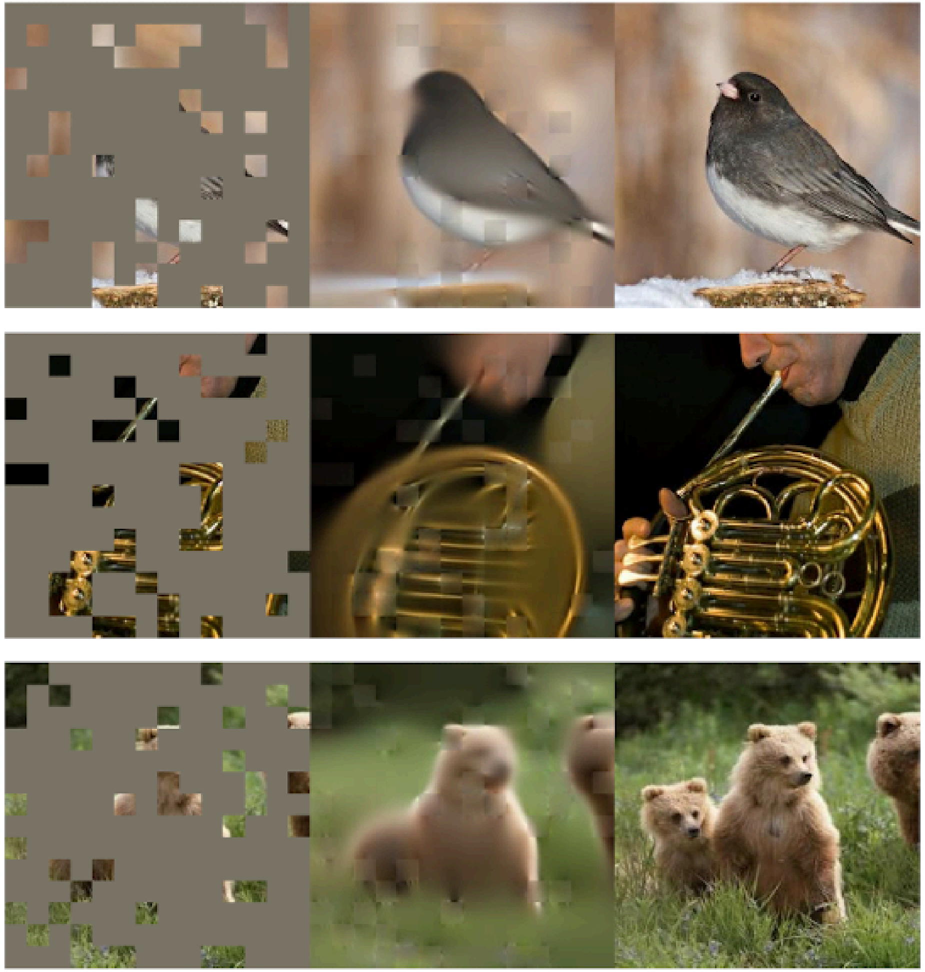
The internet contains billions of unlabeled sentences and images

We can mask/hide part of the input, and train the model to predict the missing parts

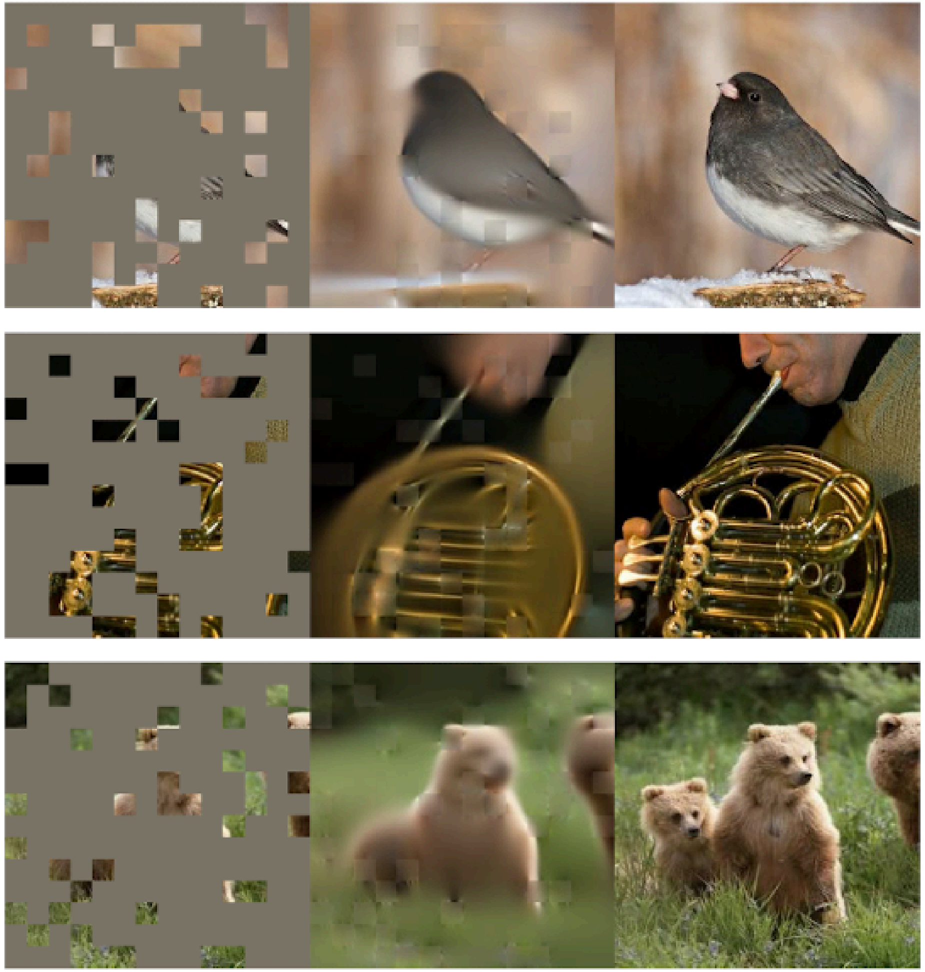
Another name for this is **generative pre-training** (GPT)

This method is **extremely** powerful

Unsupervised Training

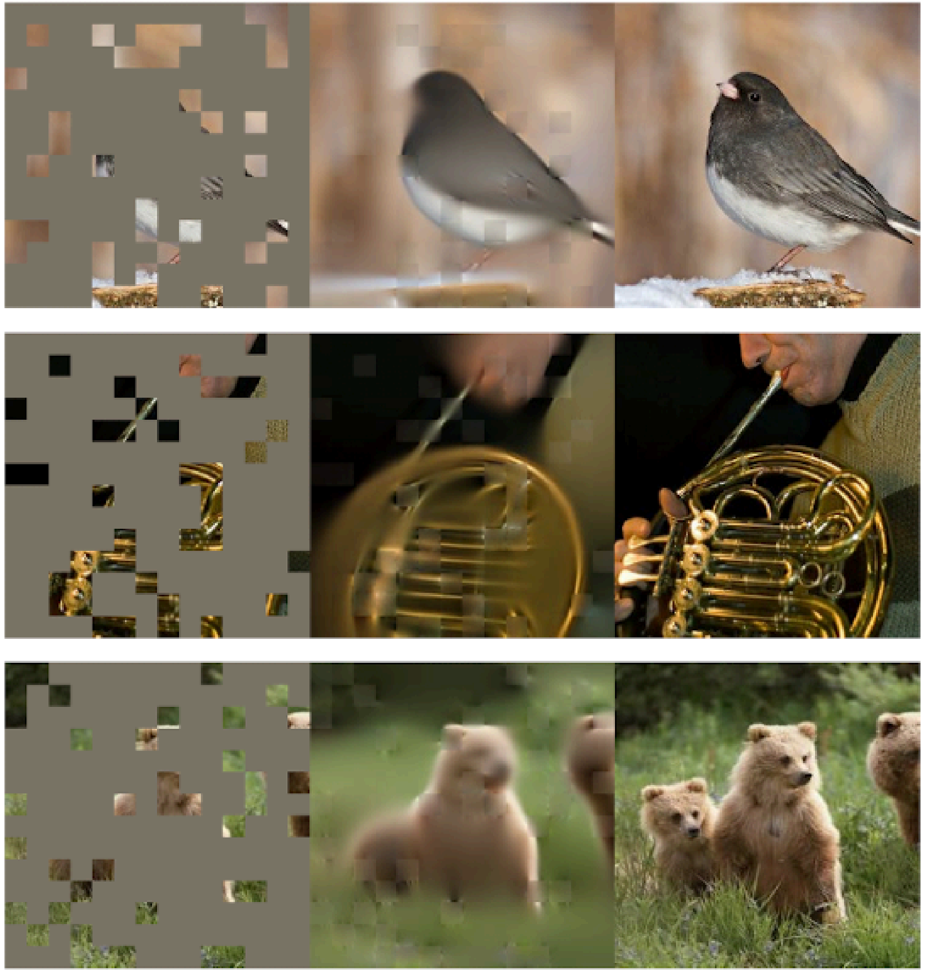


Unsupervised Training



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

Unsupervised Training



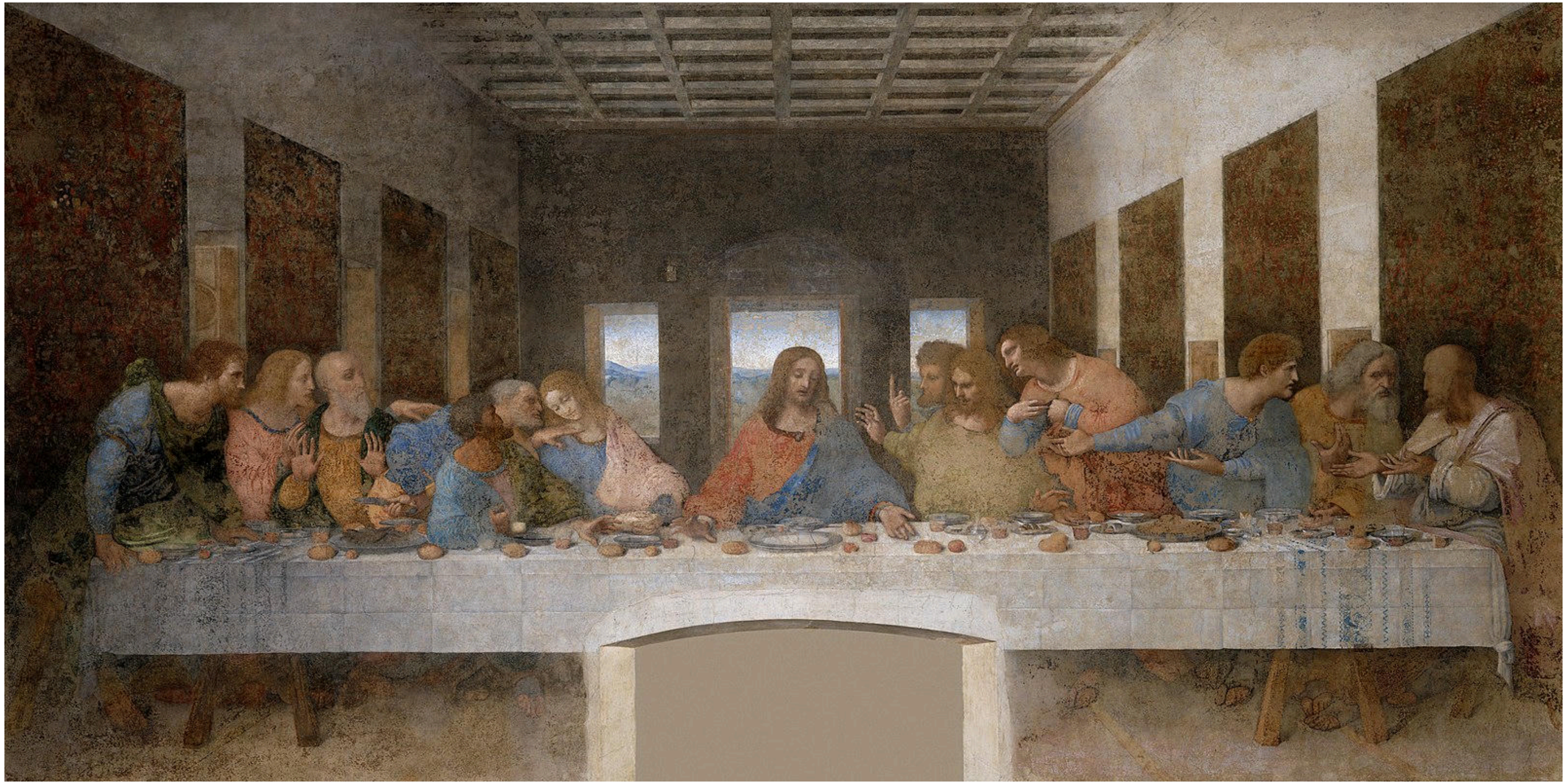
He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Unsupervised Training

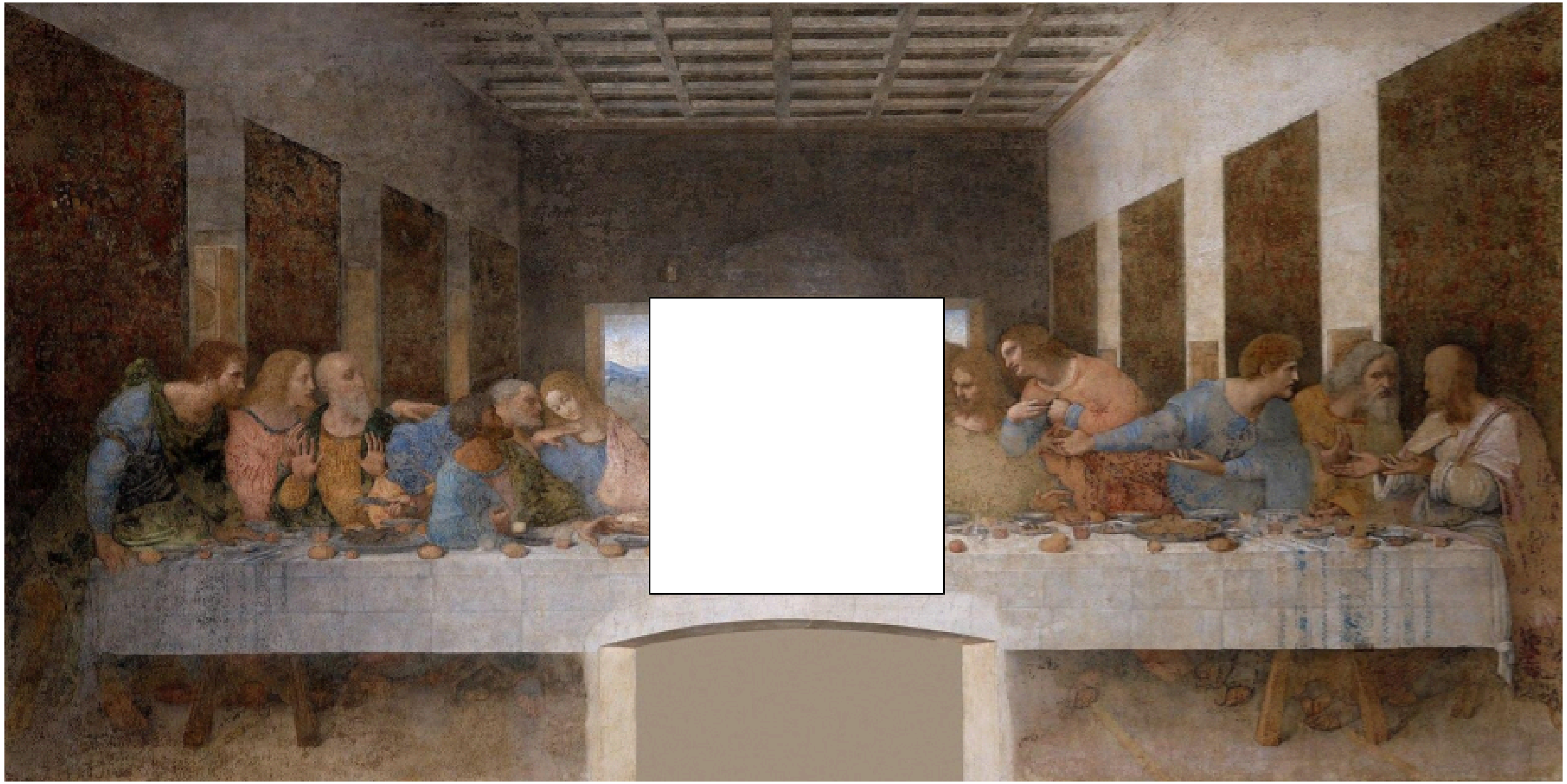
Anyone familiar with Da Vinci's painting *The Last Supper*?

Unsupervised Training

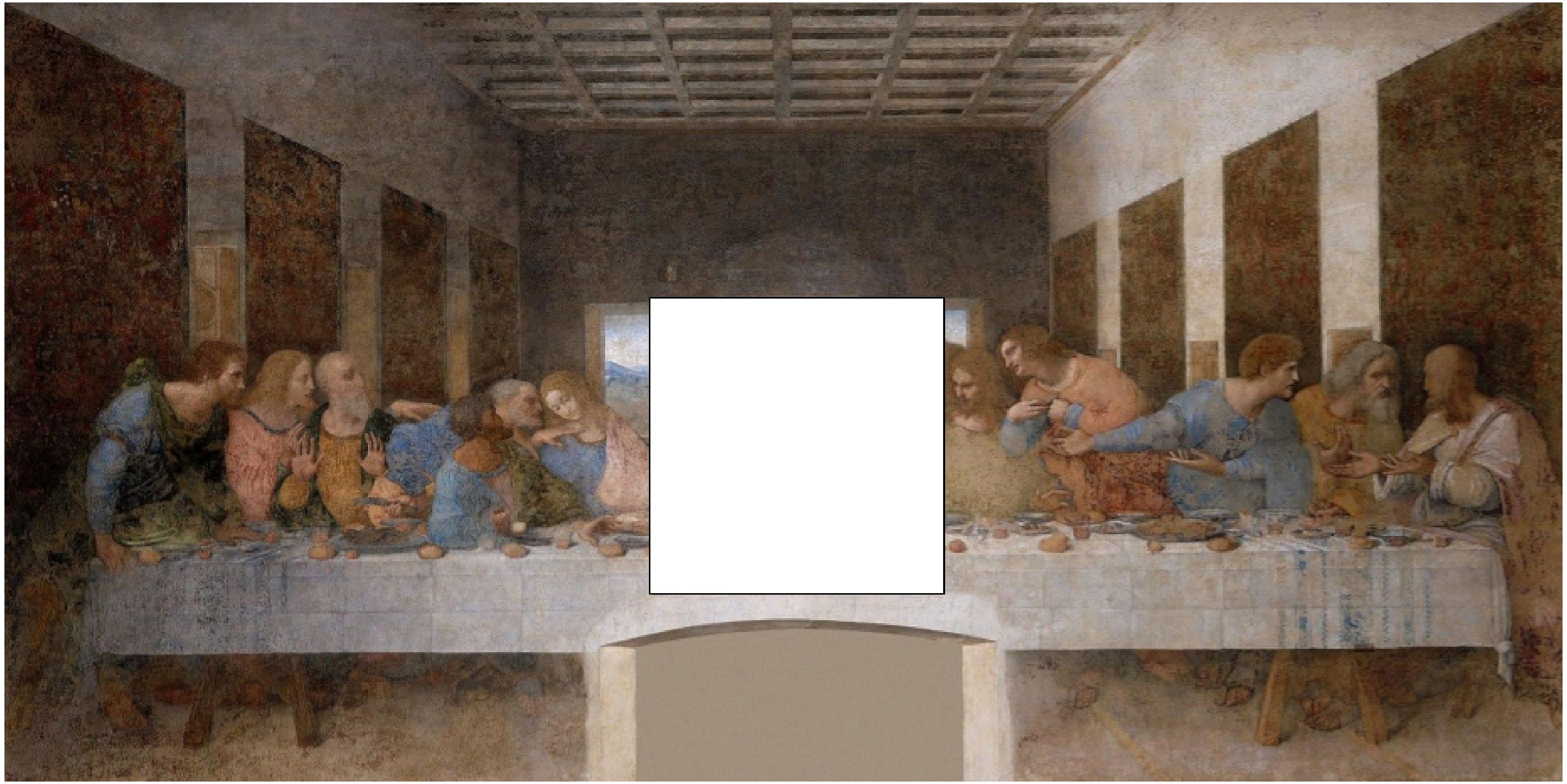
Anyone familiar with Da Vinci's painting *The Last Supper*?



Unsupervised Training



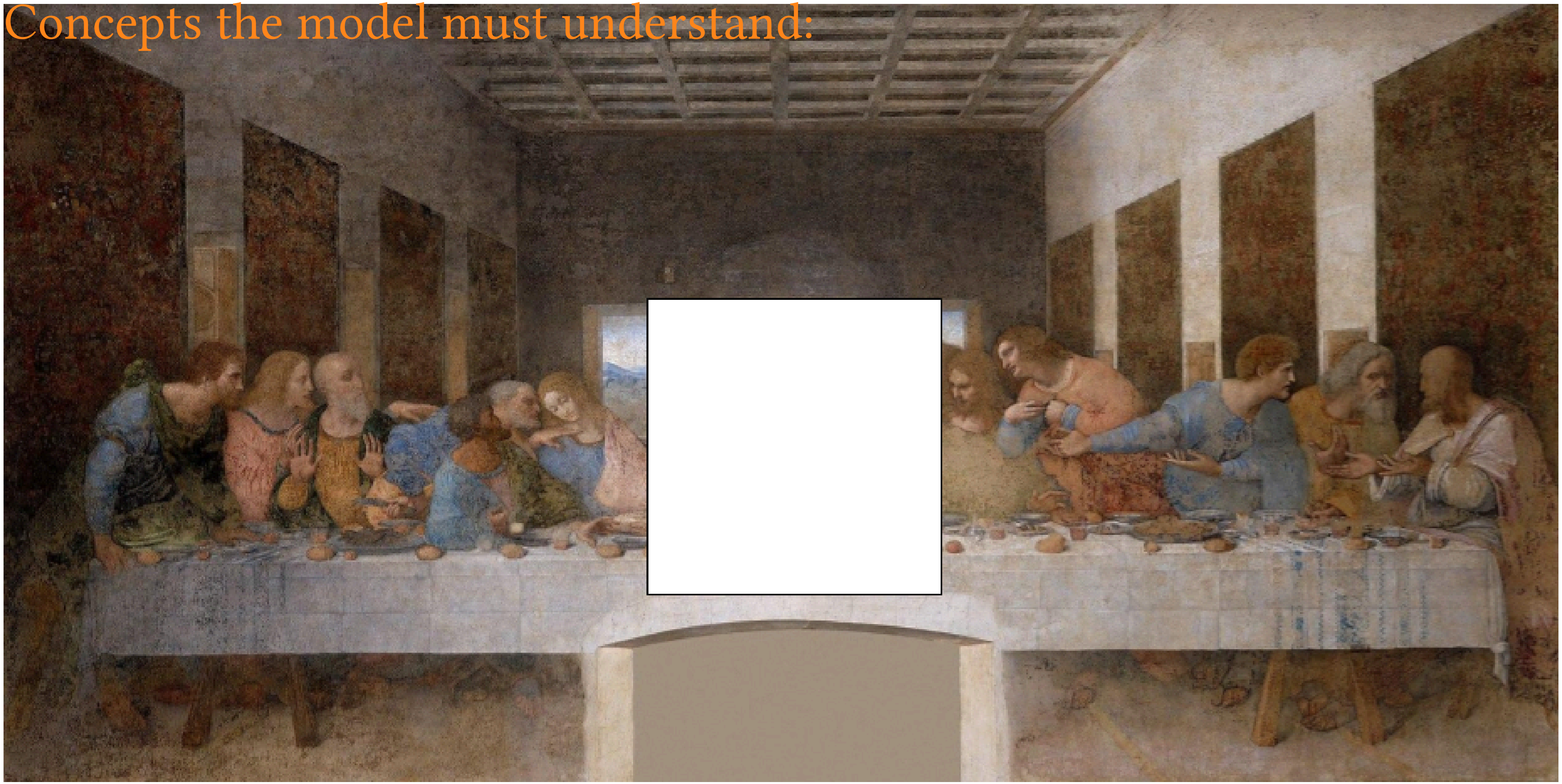
Unsupervised Training



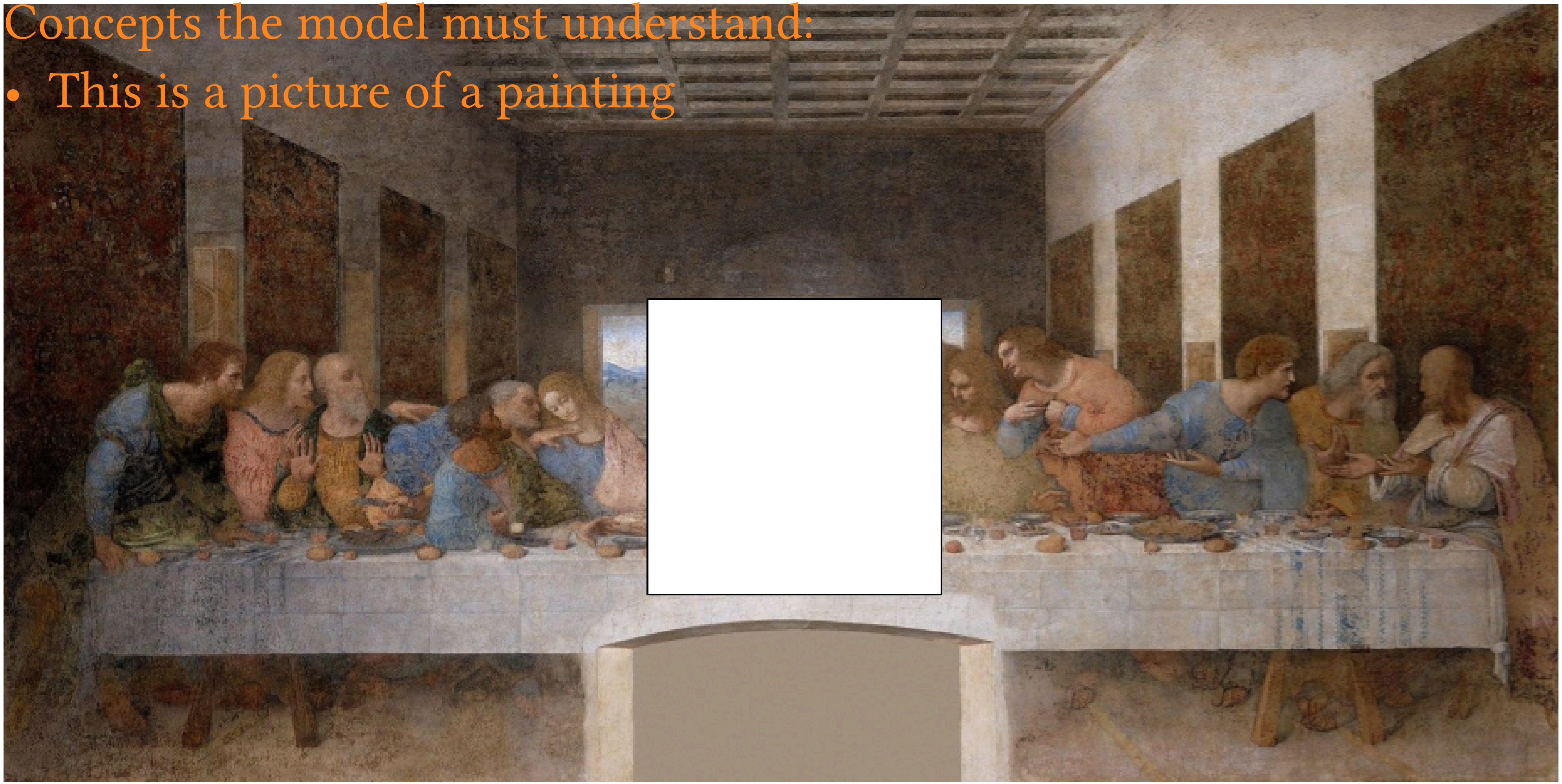
Question: What concepts does the vision transformer need to understand to predict the missing pixels?

Unsupervised Training

Concepts the model must understand:



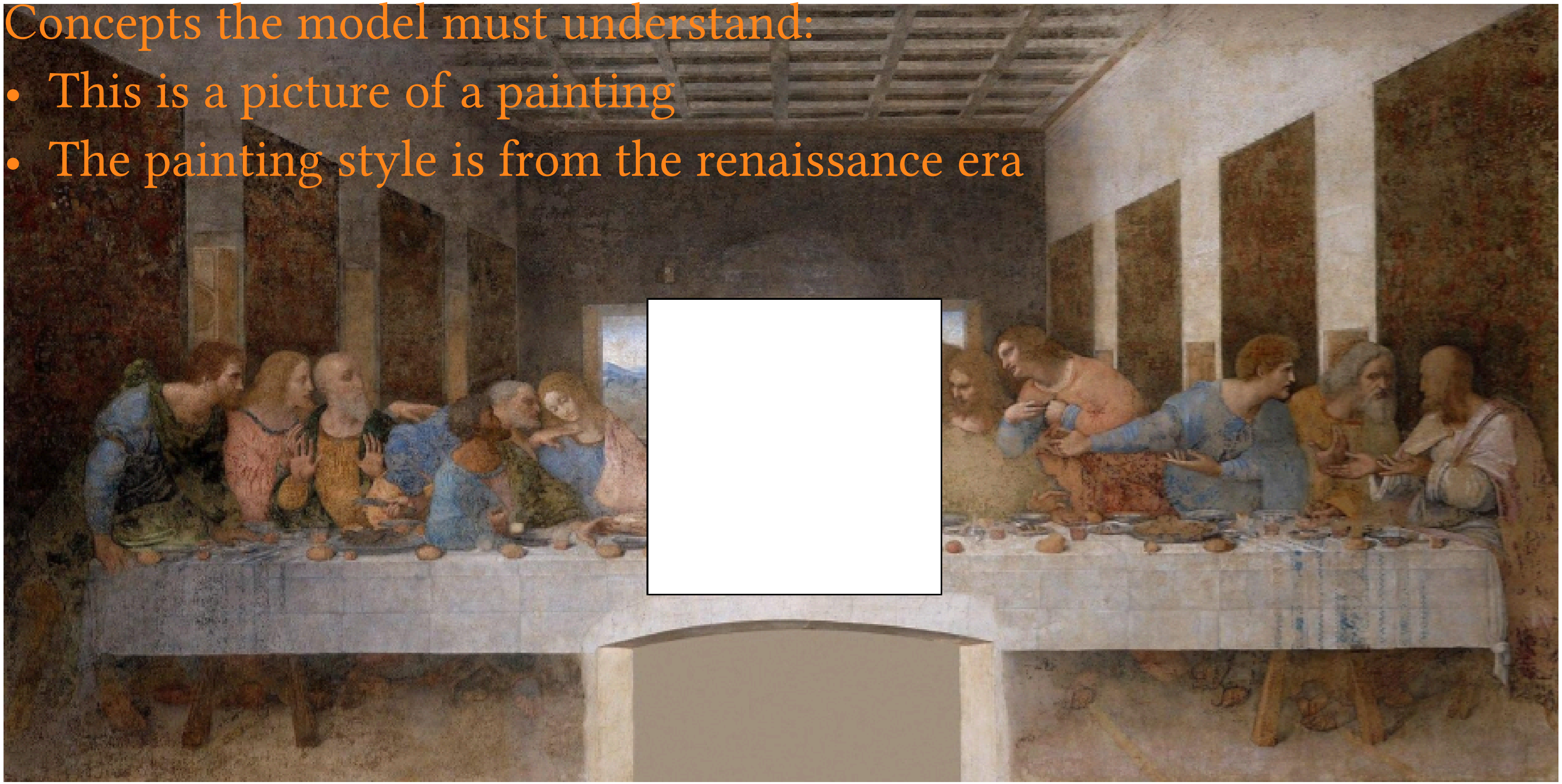
Unsupervised Training



Unsupervised Training

Concepts the model must understand:

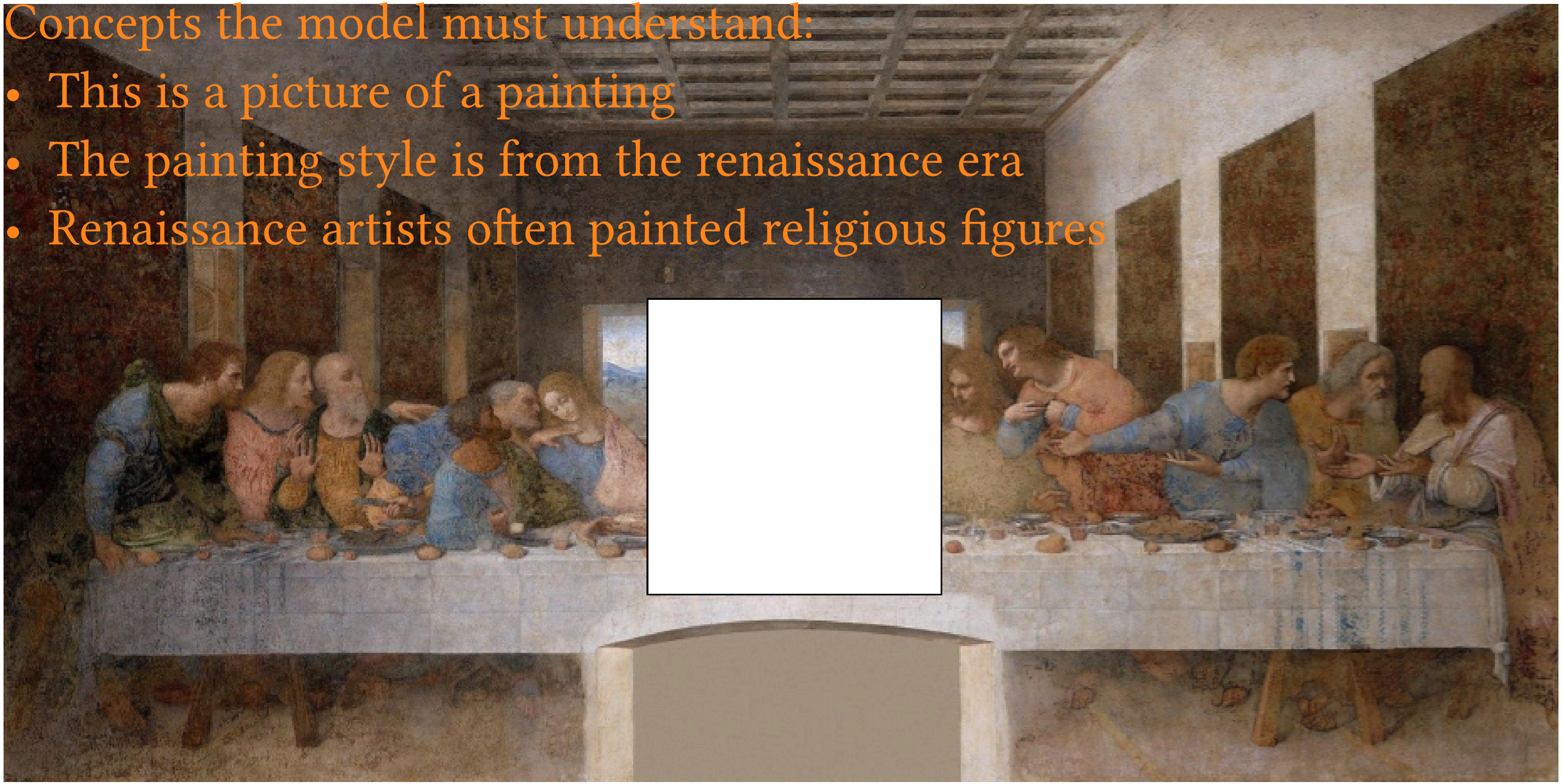
- This is a picture of a painting
- The painting style is from the renaissance era



Unsupervised Training

Concepts the model must understand:

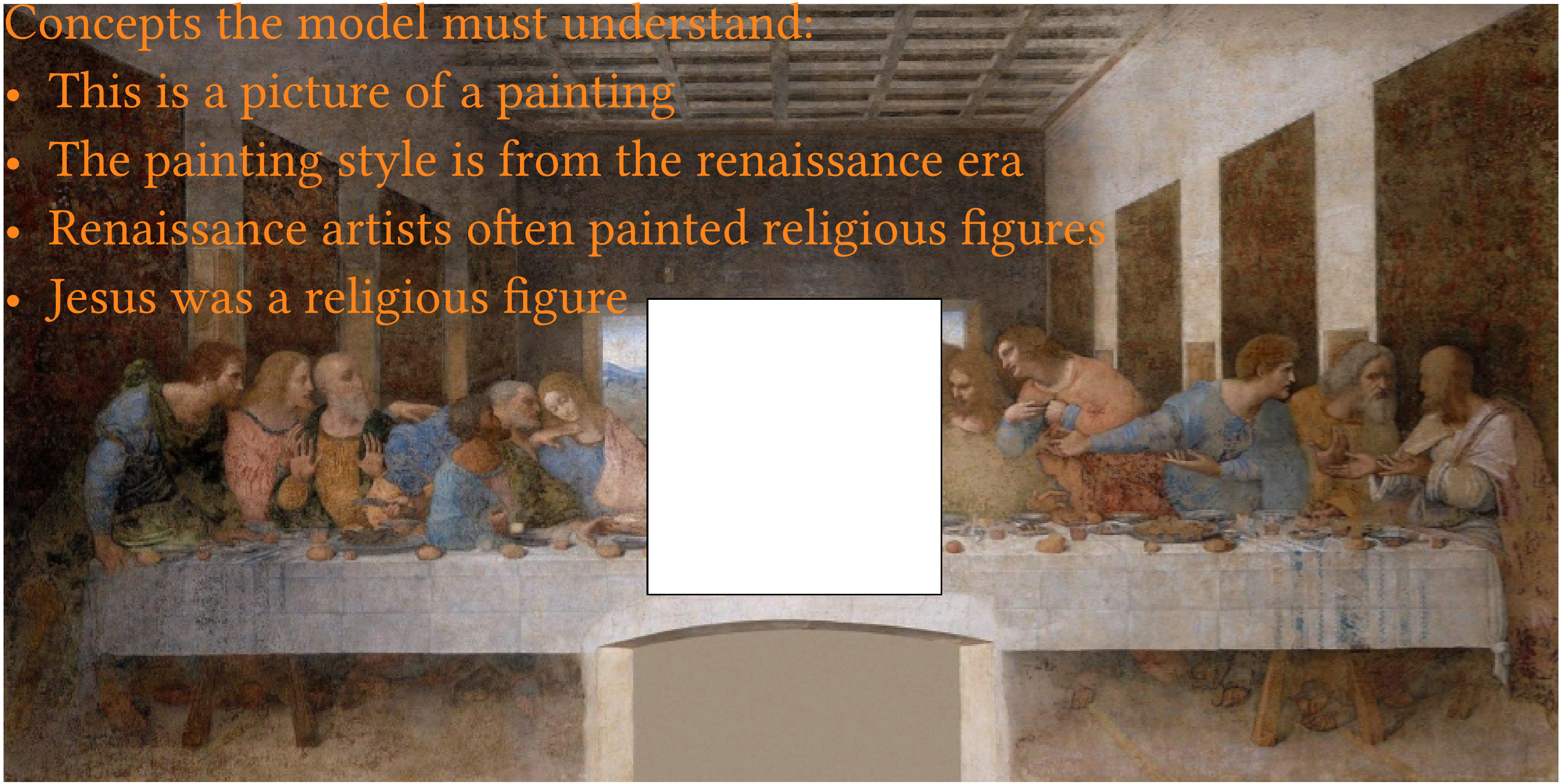
- This is a picture of a painting
- The painting style is from the renaissance era
- Renaissance artists often painted religious figures



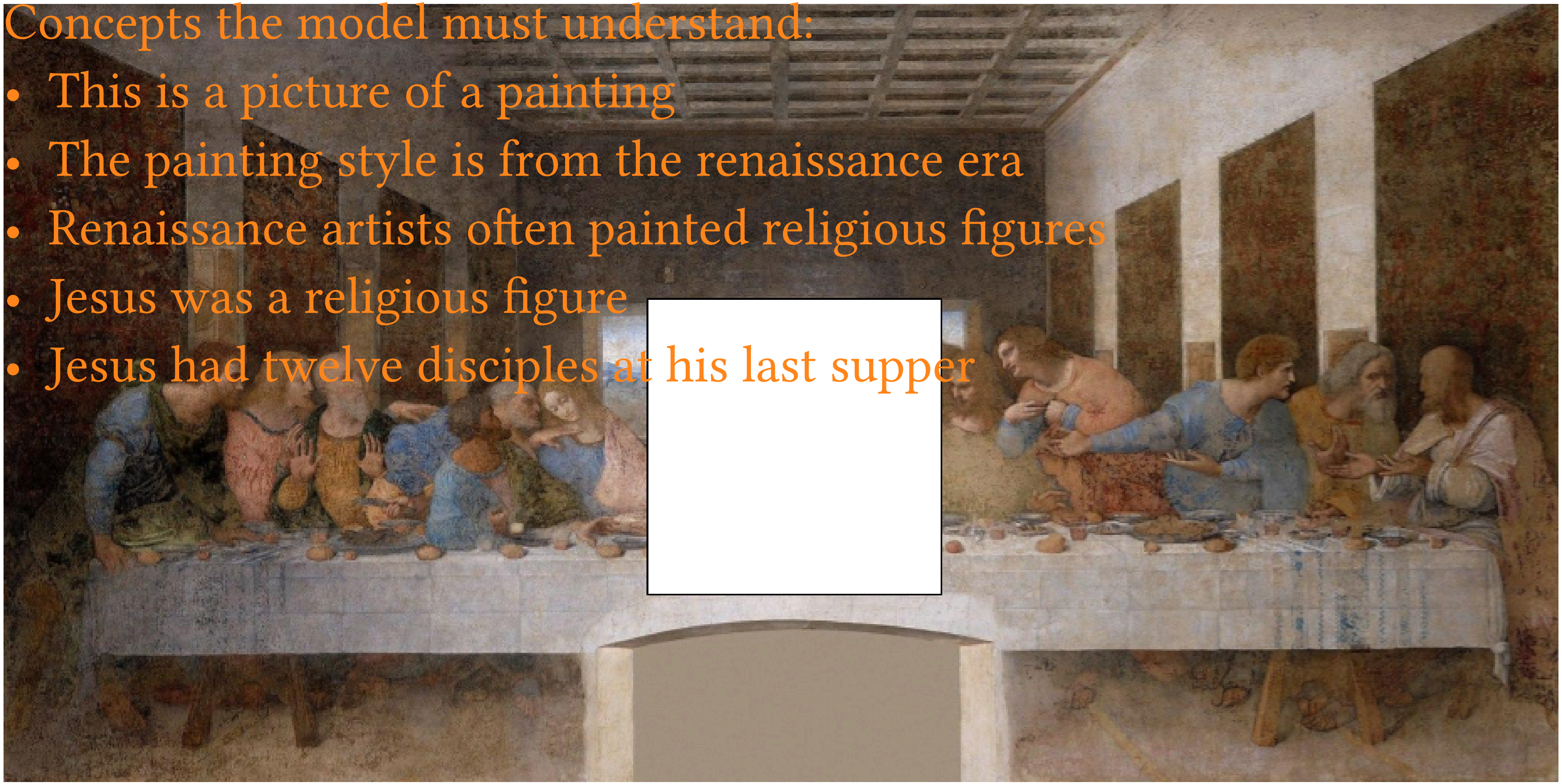
Unsupervised Training

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



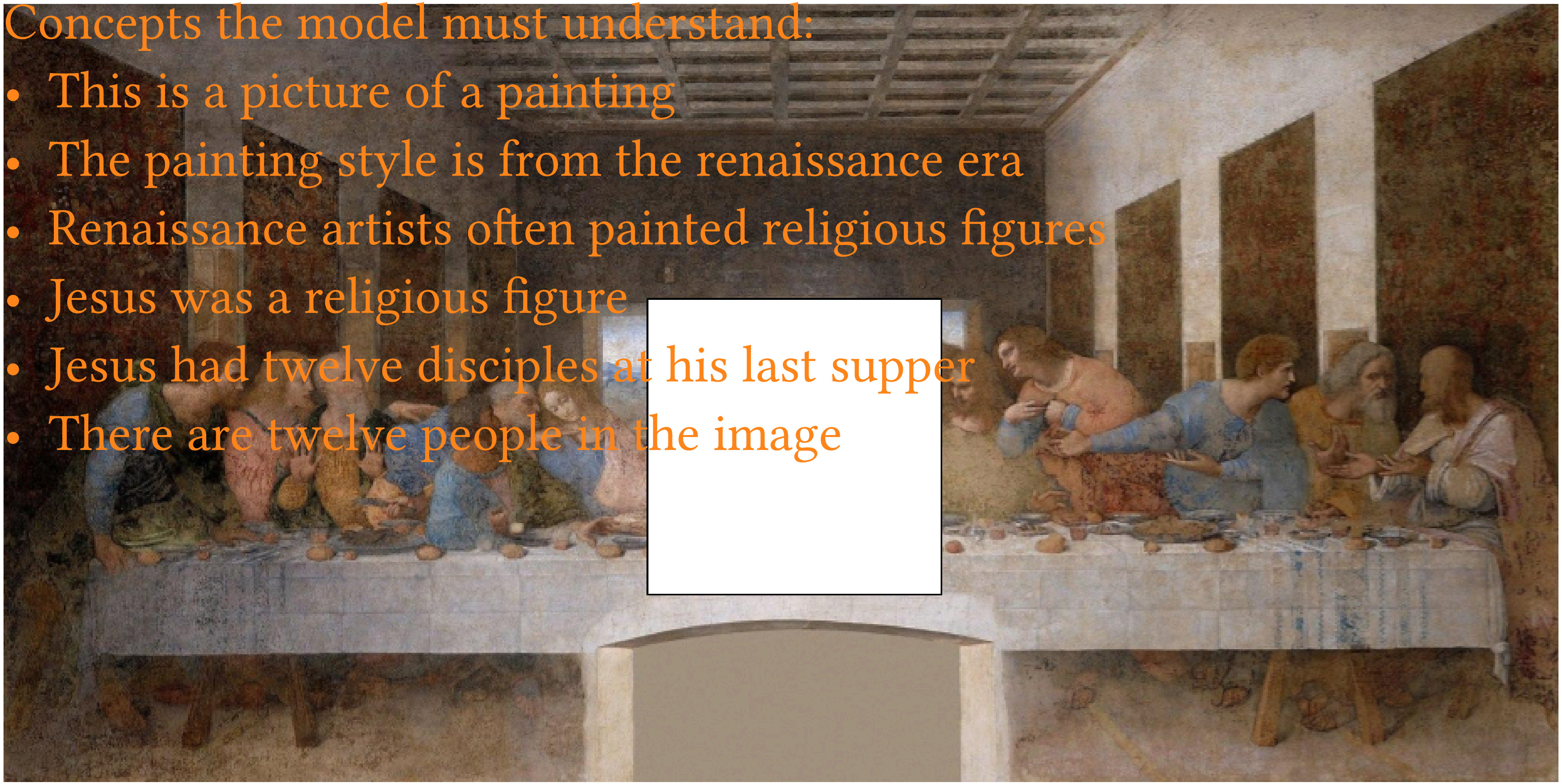
Unsupervised Training



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

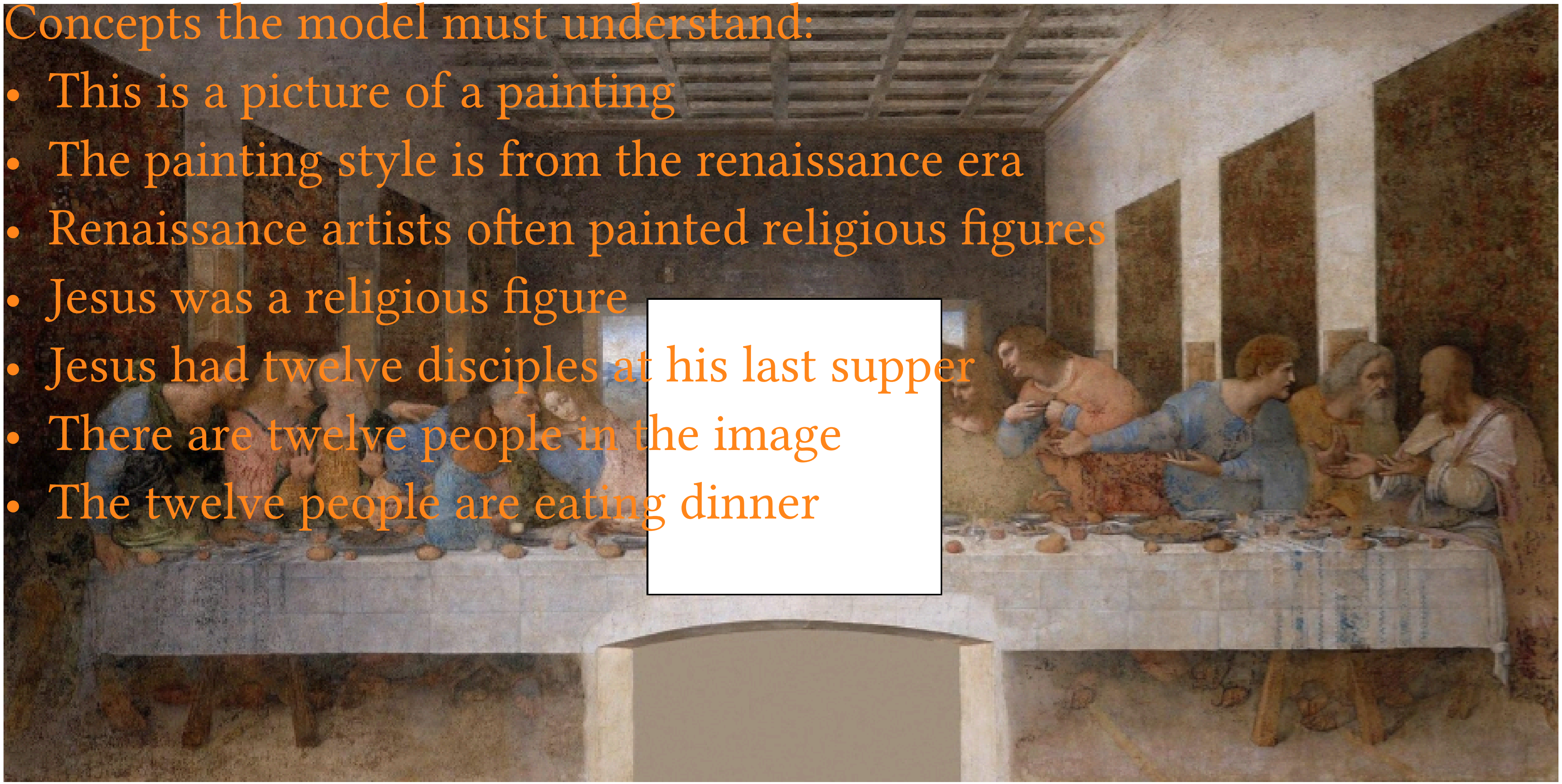
Unsupervised Training



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

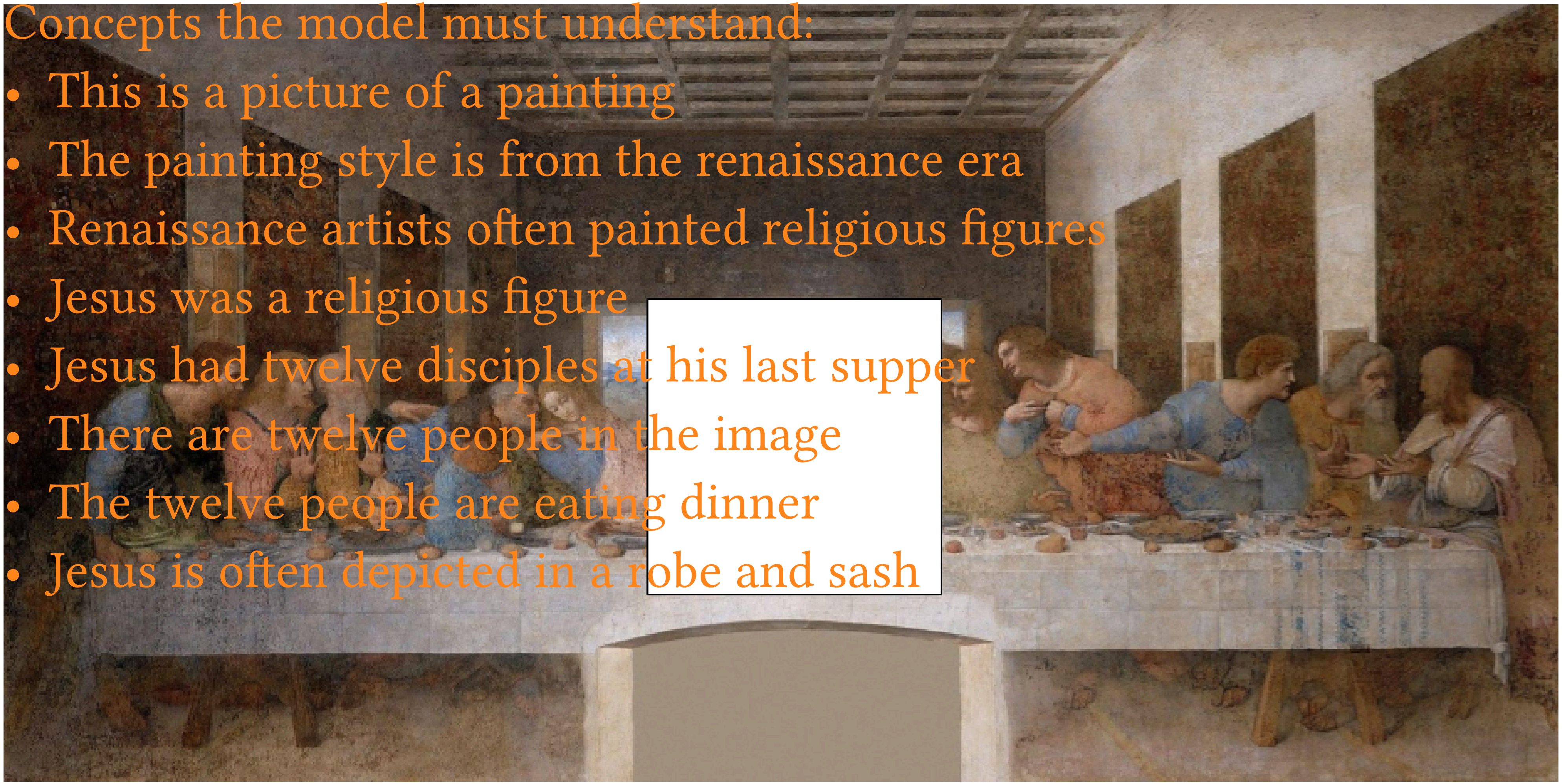
Unsupervised Training



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

Unsupervised 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

Unsupervised Training

We train the model to fix the image

Unsupervised Training

We train the model to fix the image

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

Unsupervised 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

Unsupervised Training

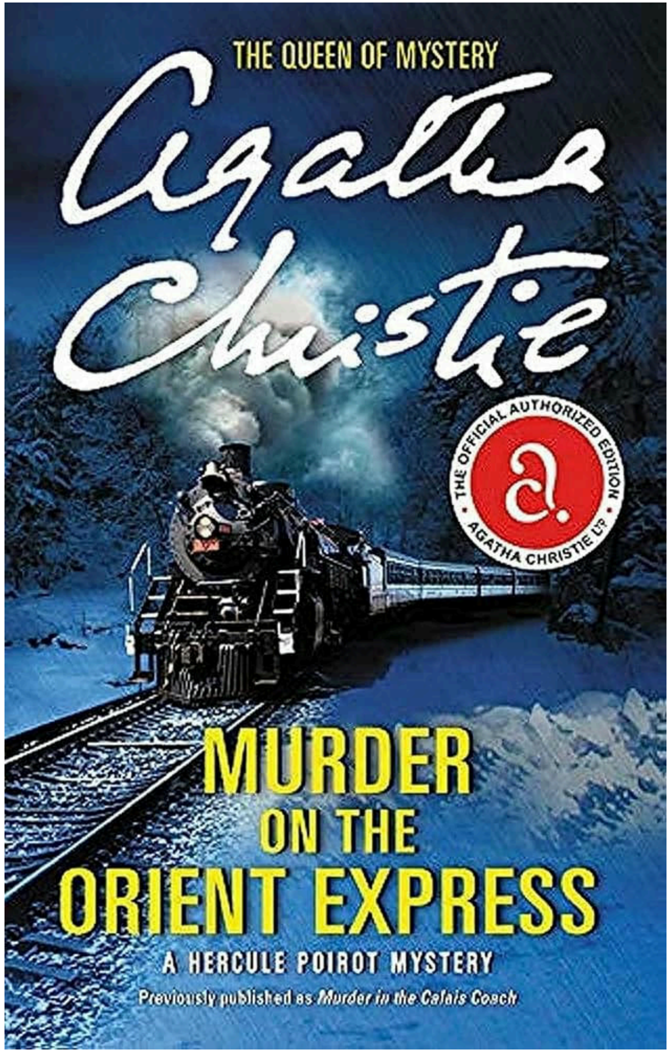
We train the model to fix the image

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

This is the power of generative pre-training

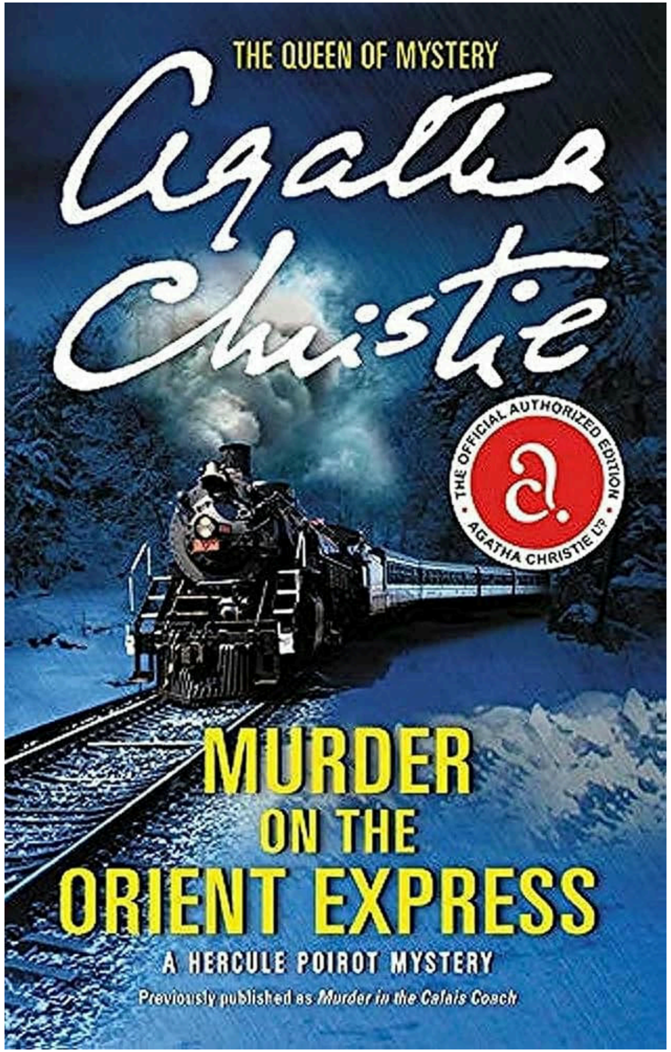
What about text transformers?

Unsupervised Training



This is a mystery novel

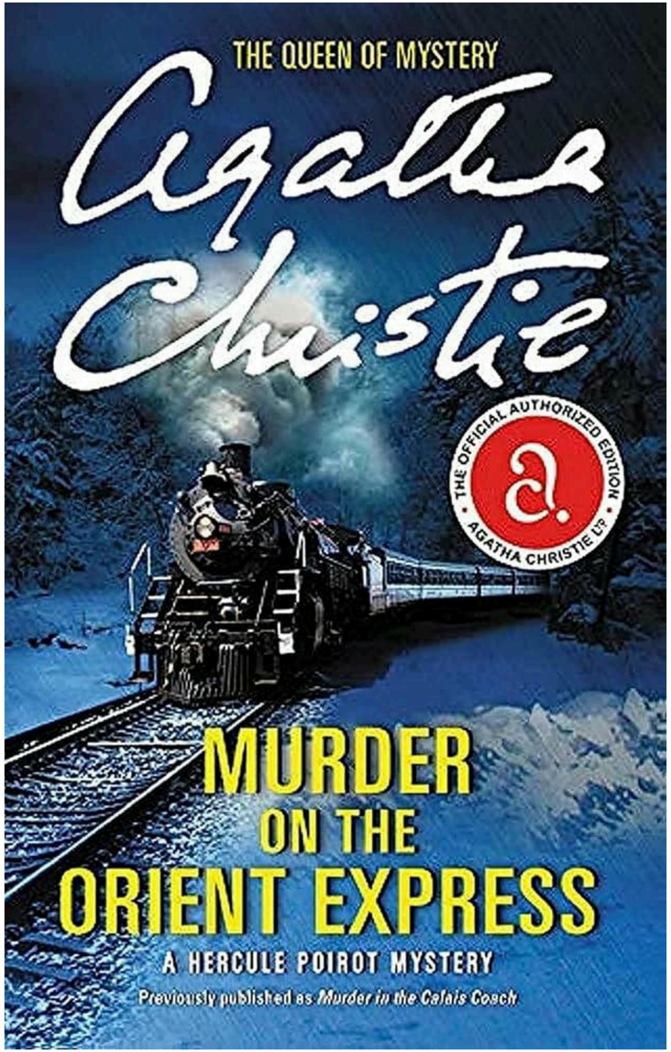
Unsupervised Training



This is a mystery novel

Clues, intrigue, murder, etc

Unsupervised Training

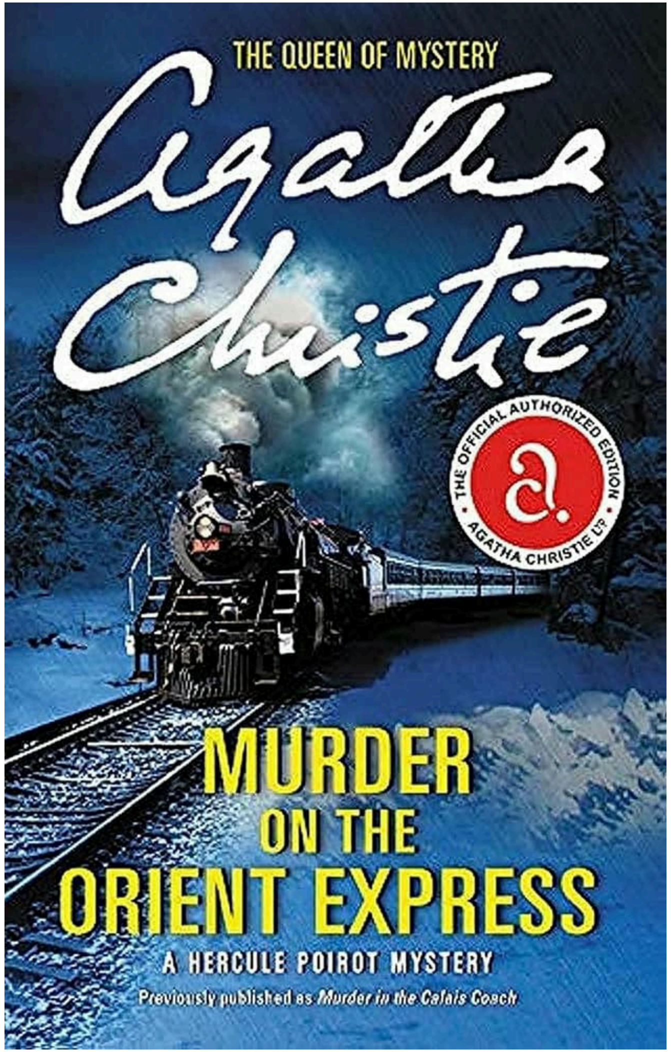


This is a mystery novel

Clues, intrigue, murder, etc

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

Unsupervised Training



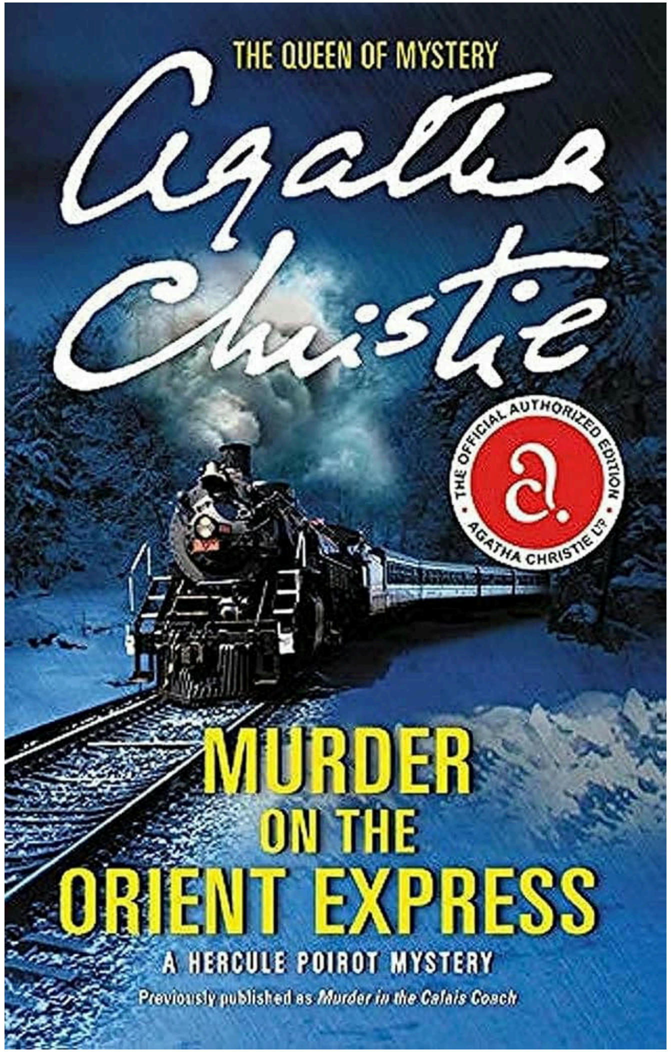
This is a mystery novel

Clues, intrigue, murder, etc

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

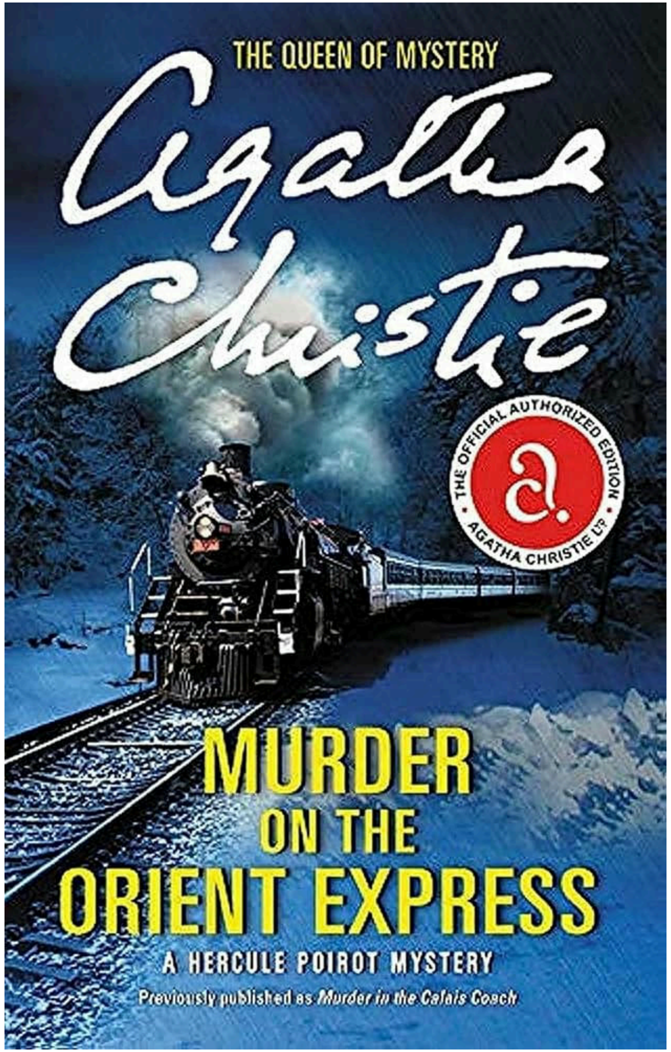
To complete the sentence, the model must understand:

Unsupervised Training



To complete the sentence, the model must understand:

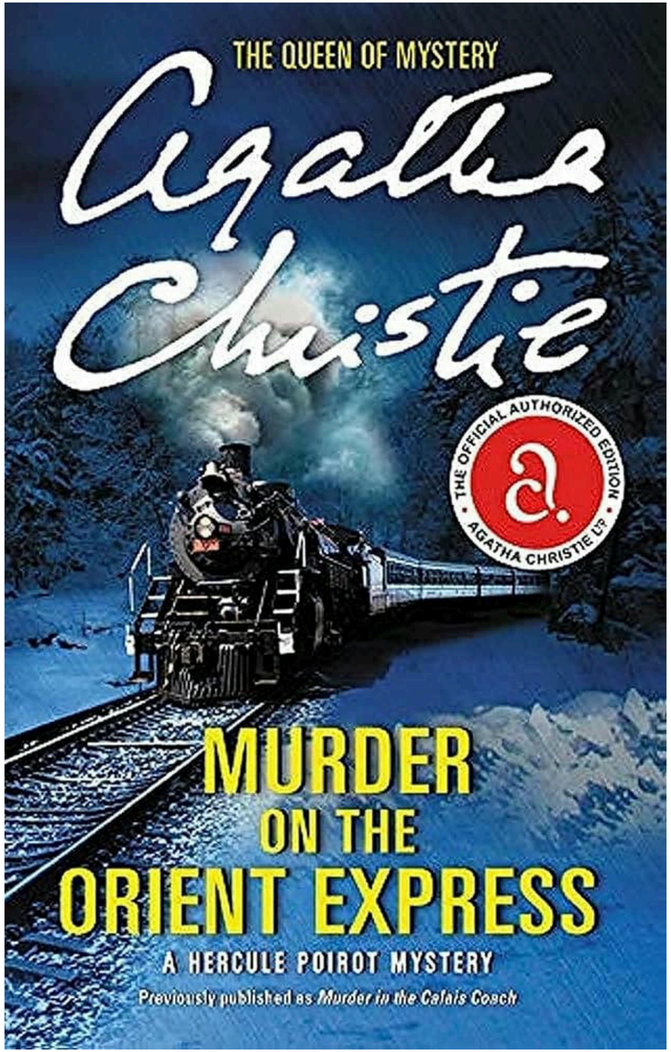
Unsupervised Training



To complete the sentence, the model must understand:

- What a murder is

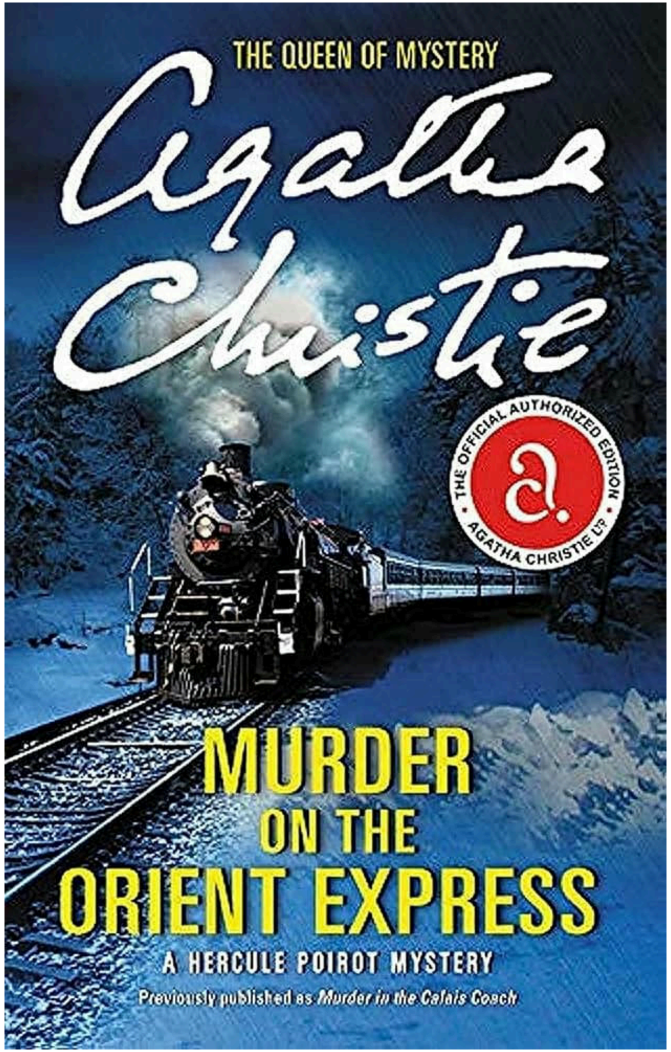
Unsupervised Training



To complete the sentence, the model must understand:

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

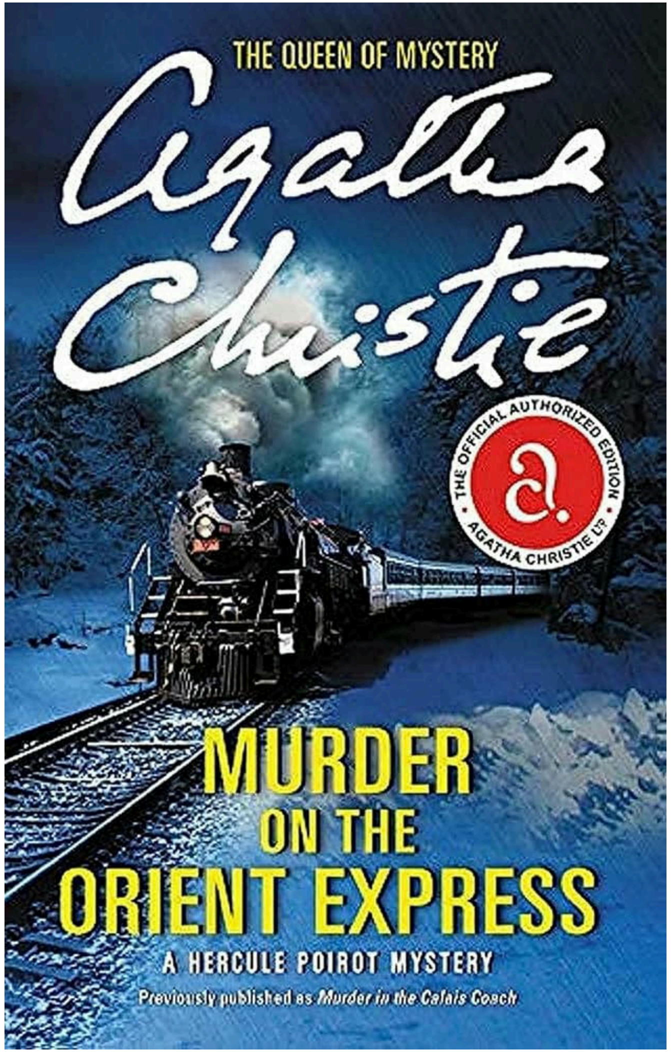
Unsupervised Training



To complete the sentence, the model must understand:

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

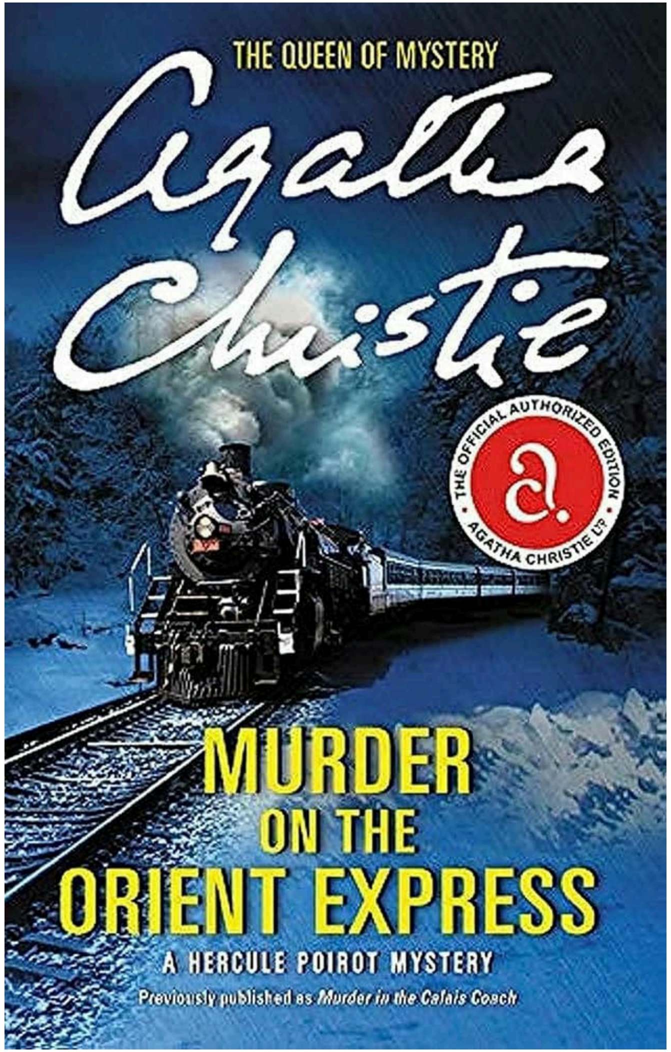
Unsupervised Training



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

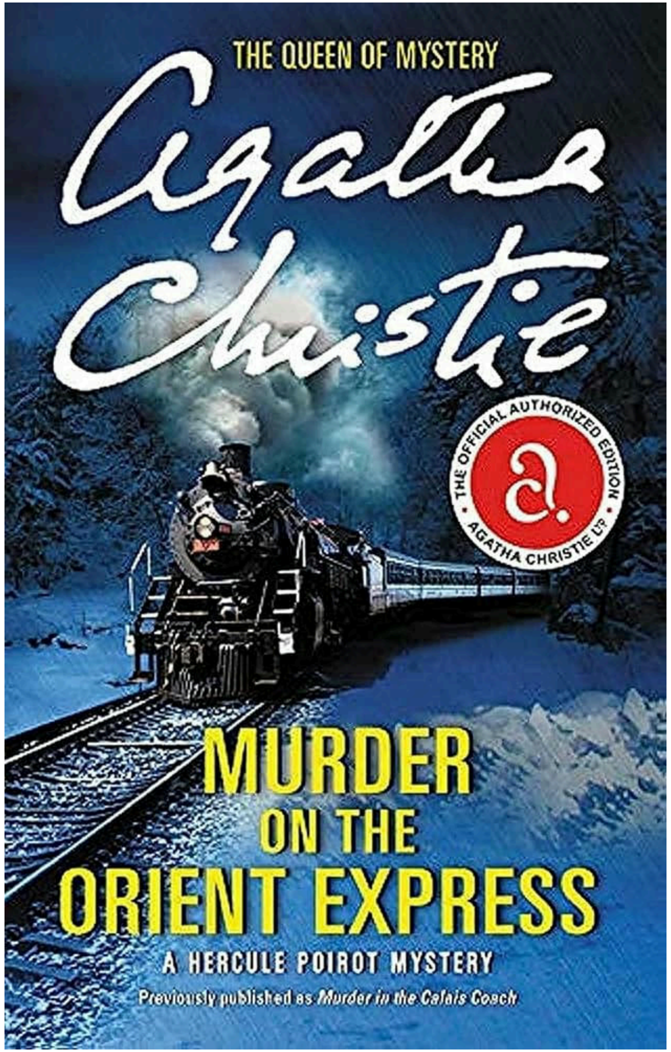
Unsupervised Training



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

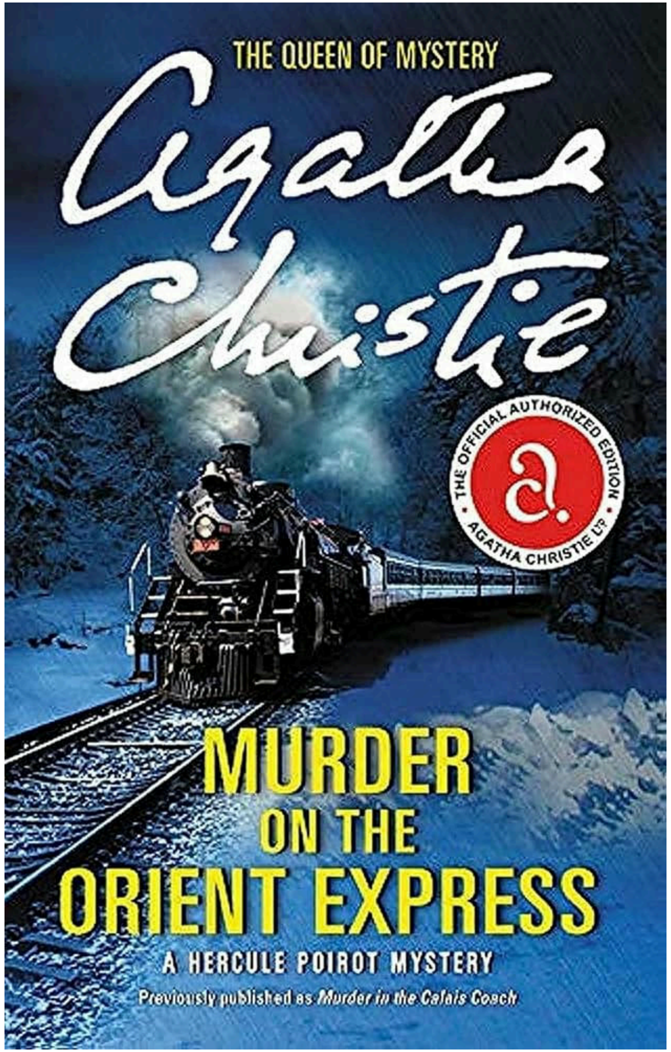
Unsupervised Training



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

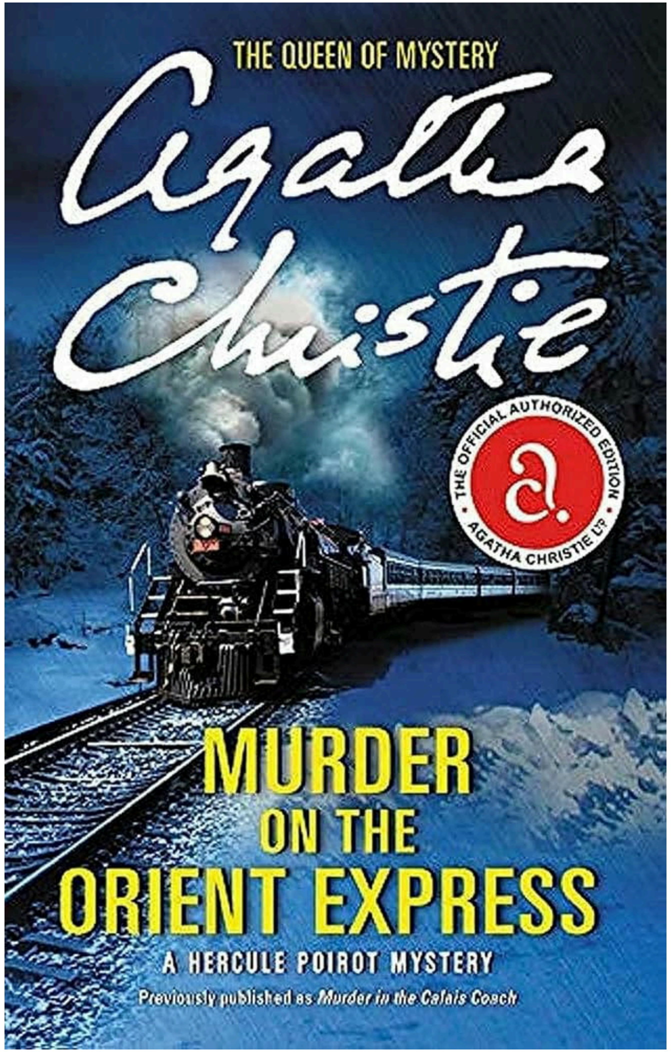
Unsupervised Training



To complete the sentence, the model must understand:

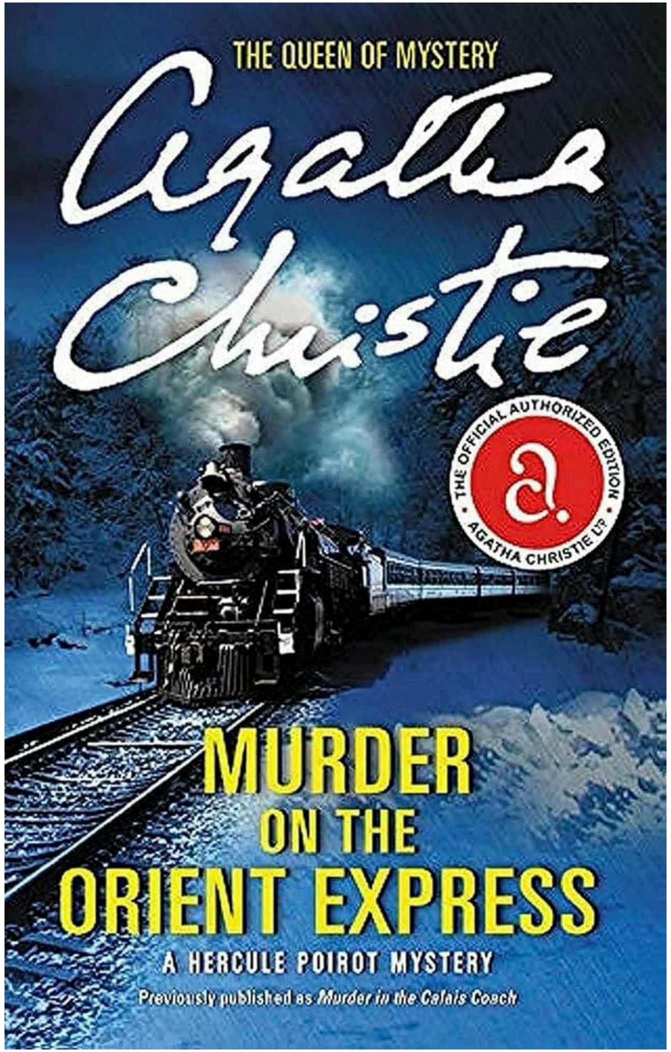
- What a murder is
- What it means to be alive
- Emotions like anger, jealousy, betrayal, love
- Personalities of each character
- Why a human would murder another human
- How humans react to emotions
- How to tell if someone lies

Unsupervised Training



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

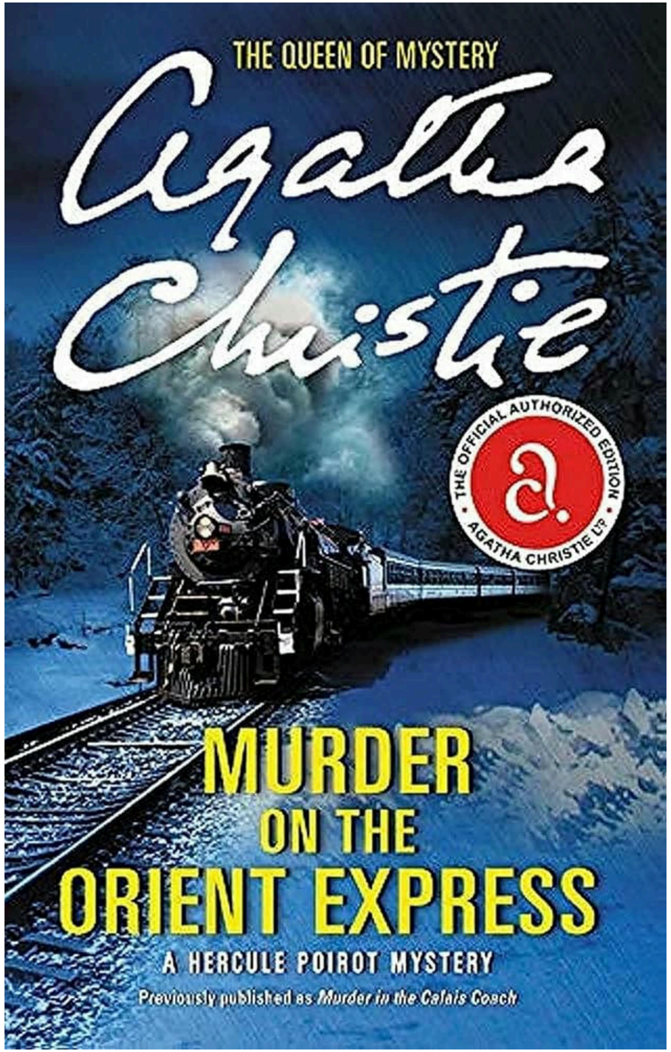
Unsupervised Training



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

The Books3 dataset contains 200,000 books

Unsupervised 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

Unsupervised Training

We can apply this same concept to:

Unsupervised Training

We can apply this same concept to:

- Predict missing base pairs in a strand of DNA

Unsupervised Training

We can apply this same concept to:

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

Unsupervised 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

Unsupervised 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!

Unsupervised Training

What if we put the model in a robot?

Unsupervised Training

What if we put the model in a robot?

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

Unsupervised 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

Unsupervised 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

We call this a **world model**

Unsupervised Training

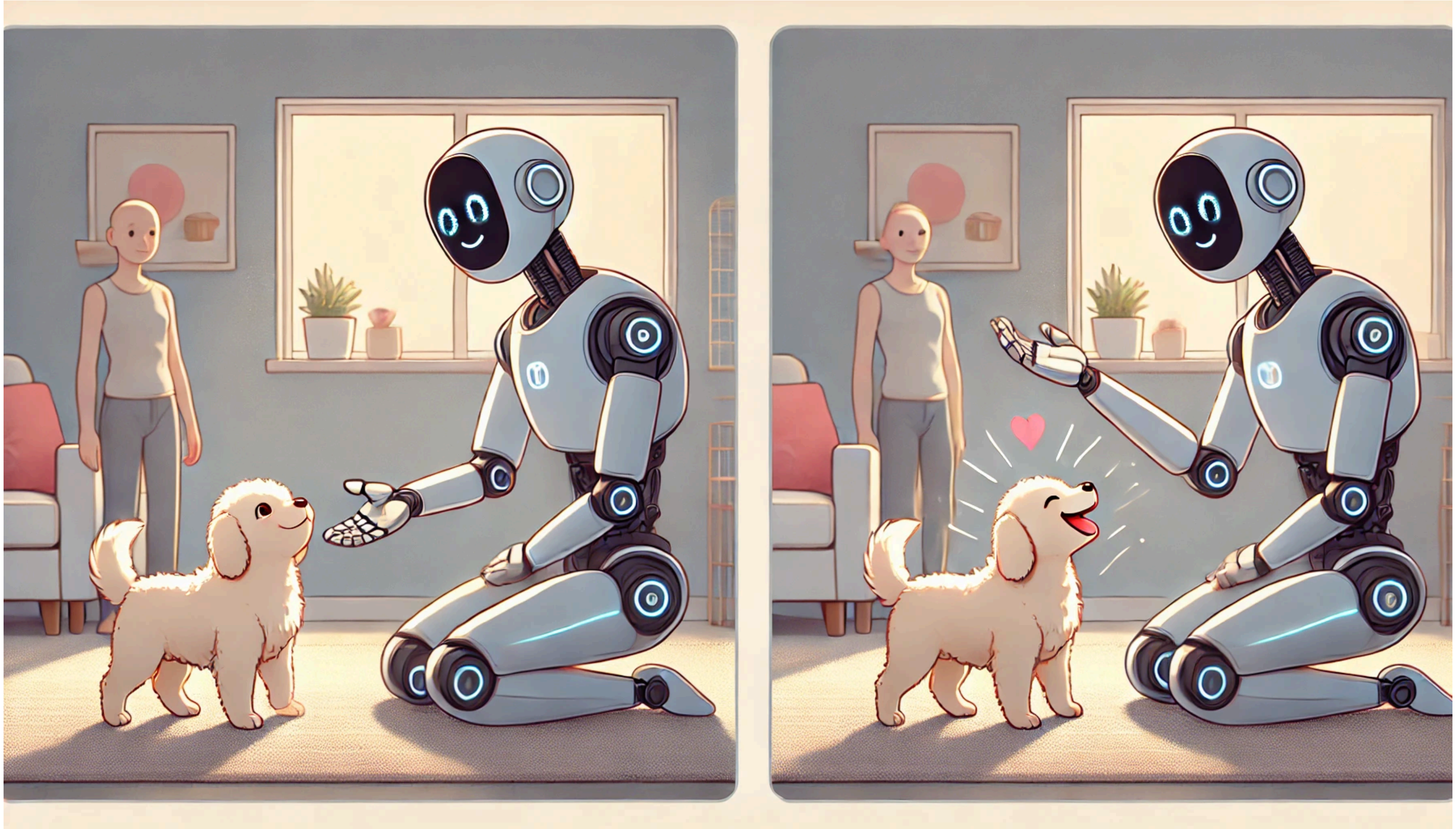
What if we put the model in a robot?

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

Predict what the robot will see next

We call this a **world model**

Unsupervised Training



Unsupervised Training

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

Unsupervised Training

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

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

Unsupervised 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

Unsupervised 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

Unsupervised Training

These transformers learn and understand the structure of our world

Unsupervised Training

These transformers learn and understand the structure of our world

But their understanding is trapped

Unsupervised Training

These transformers learn and understand the structure of our world

But their understanding is trapped

They can only finish sentences or complete pictures

Unsupervised 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?

Unsupervised 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

Unsupervised 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

Unsupervised 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

Unsupervised 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

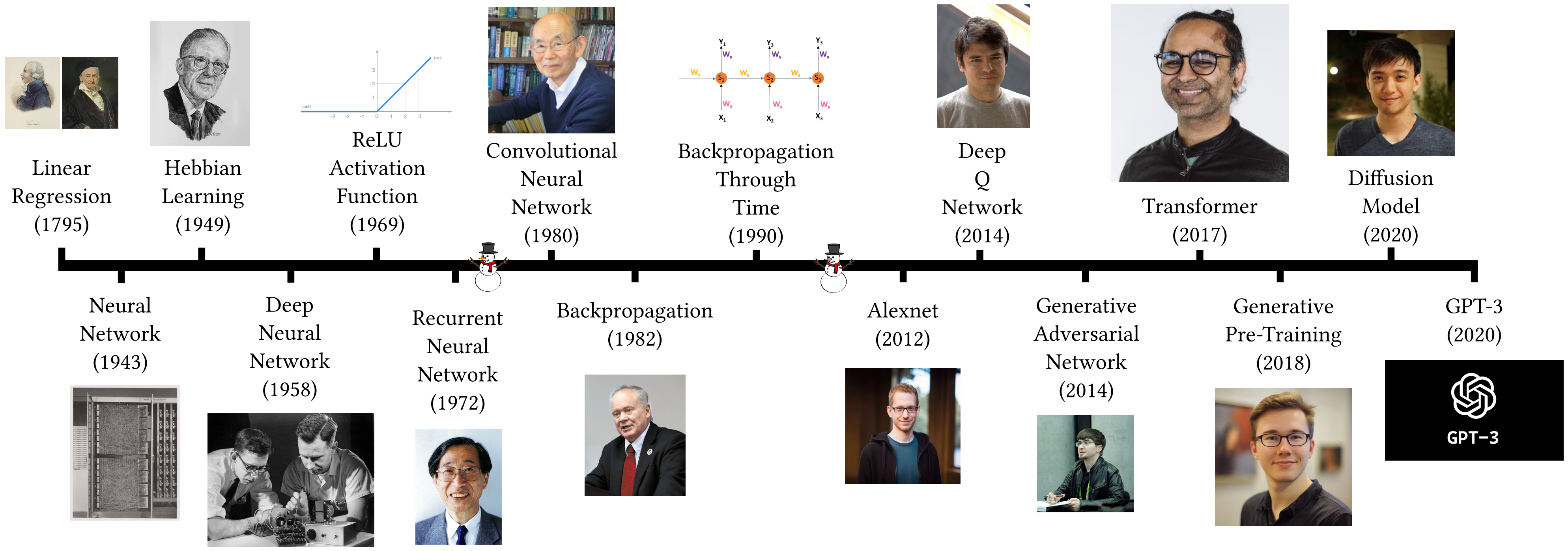
In this course, we started from Gauss in 1795

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

There are many more topics to cover!

Closing Remarks

There are many more topics to cover!

Now, you have the tools to study deep learning

Closing Remarks

There are many more topics to cover!

Now, you have the tools to study deep learning

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

Department has suggested 10 minutes

Course Evaluation

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

Please scan the QR code and complete the survey

Department has suggested 10 minutes

I will return in 10 minutes

Course Evaluation

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

Please scan the QR code and complete the survey

Department has suggested 10 minutes

I will return in 10 minutes

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