DEEP LEARNING

Deep Generative Models

Agenda

- What is Generative AI?
  - New kid on the block?
  - It has been around since 2017 – Attention is All You Need (in fact since 2014, Generative Adversarial Networks)
- Generative Adversarial Network (GAN)
- Variational Autoencoder
- Autoregressive Model
  - State of the Art Deep Learning model: Transformer

What is It?

Generative artificial intelligence (AI) is a capability to generate text, images, or other media in response to prompts.

Generative AI models learn the patterns and structure of their input training data by applying neural network models, and then generate new data that has similar attributes.

Some notable generative AI systems:

- ChatGPT, DALL-E, Bard, etc.

Google Bard

DALL·E 2

ChatGPT

OpenAI GPT-4

Generative vs. Discriminative Model

A **generative model** is a model of the conditional probability of the observable X, given a target y, symbolically, $P(X \mid Y = y)$

A **discriminative model** is a model of the conditional probability of the target Y, given an observation x, symbolically, $P(Y \mid X = x)$

A generative model can be used to "*generate*" random instances (outcomes), either of an observation and target (x,y), or of an observation x given a target value y.

While a discriminative model can be used to "*discriminate*" the value of the target variable Y, given an observation x.

Probability Example

- Data (x,y): (1,0), (1,0), (2,0), (2,1)

P(x,y):

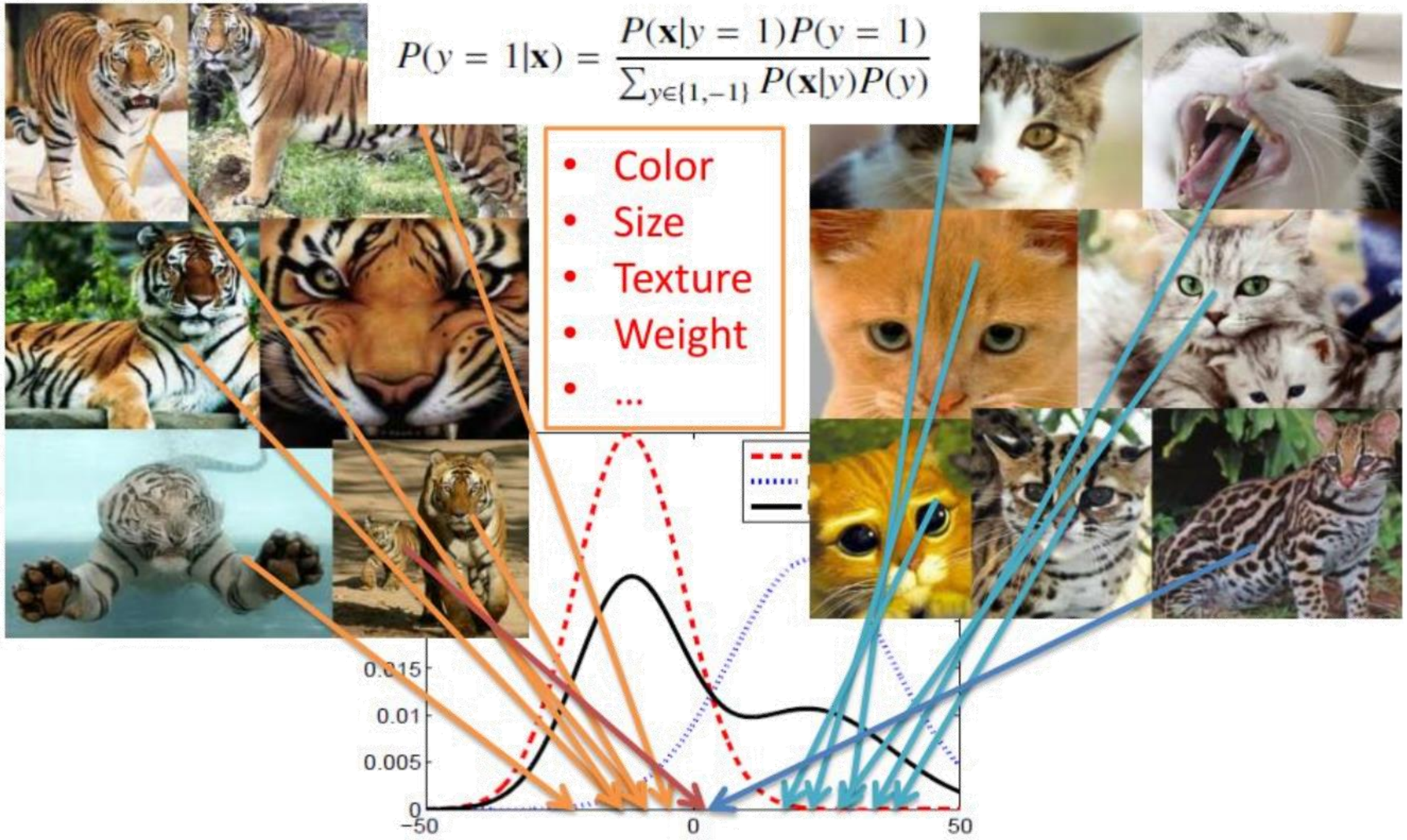
	y=0	y=1
x=1	1/2	0
x=2	1/4	1/4

P(y|x):

	y=0	y=1
x=1	1	0
x=2	1/2	1/2

$$P(x,y)=P(y|x)P(x)$$

Generative Model



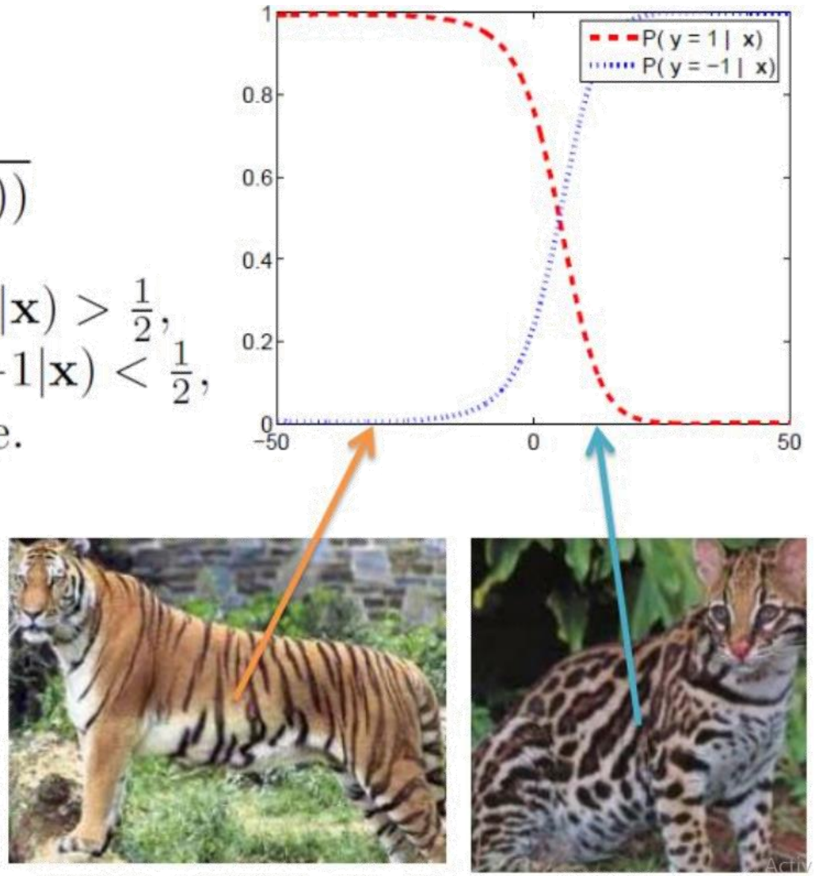
Discriminative Model

- Logistic regression

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(yf(\mathbf{x}))}$$

$$f^*(\mathbf{x}) = \begin{cases} +\infty & \Pr(y = 1|\mathbf{x}) > \frac{1}{2}, \\ -\infty & \Pr(y = -1|\mathbf{x}) < \frac{1}{2}, \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

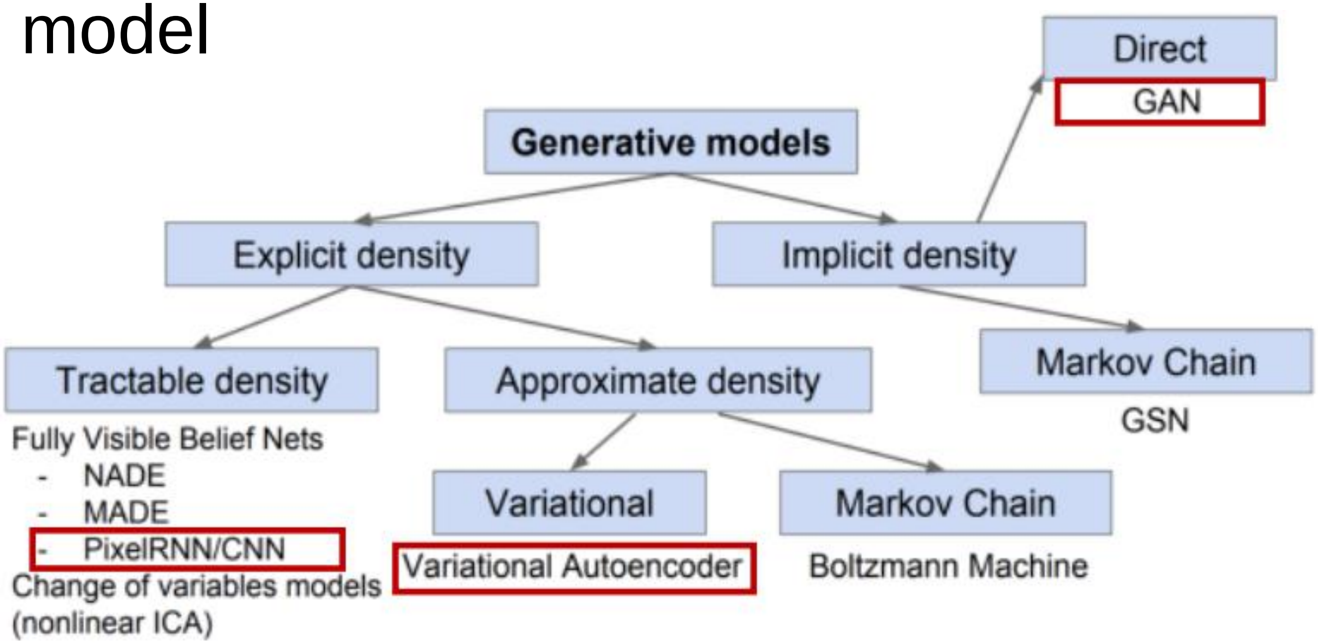
- Color
- Size
- Texture
- Weight
- ...



Deep Generative Models

- Generative adversarial networks (GAN)
- Variational autoencoder (VAE)
- Autoregressive model

Taxonomy ->



Generative Adversarial Network (GAN)

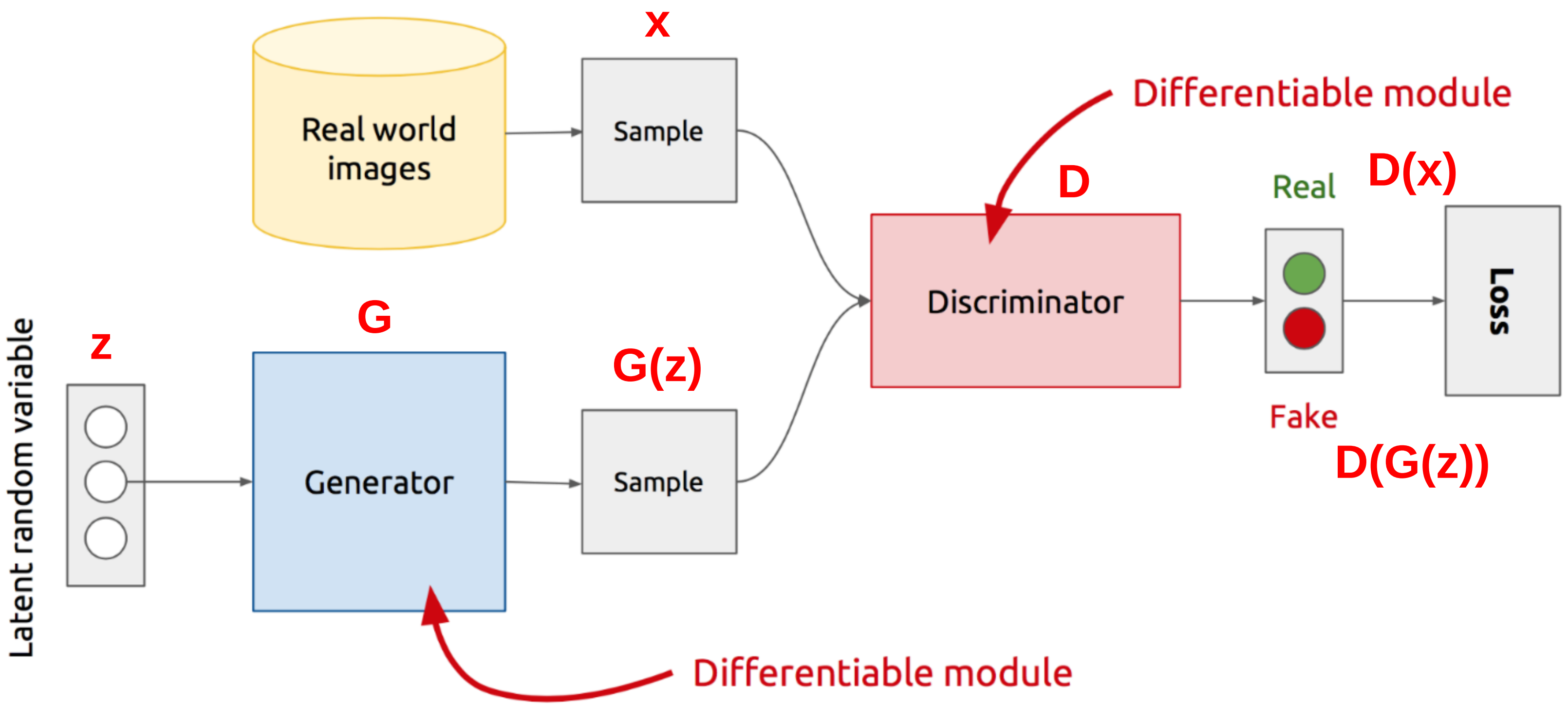
Adversarial Training

- GANs are generative models that are implemented using two stochastic neural network modules: Generator and Discriminator

- Generator tries to generate samples from random noise as input
  Discriminator tries to distinguish the samples from Generator and samples from the real data distribution

- Both networks are trained adversarially (in tandem) to fool the other component. In this process, both models become better at their respective tasks

GAN's Architecture



- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

Image source: www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Discriminator

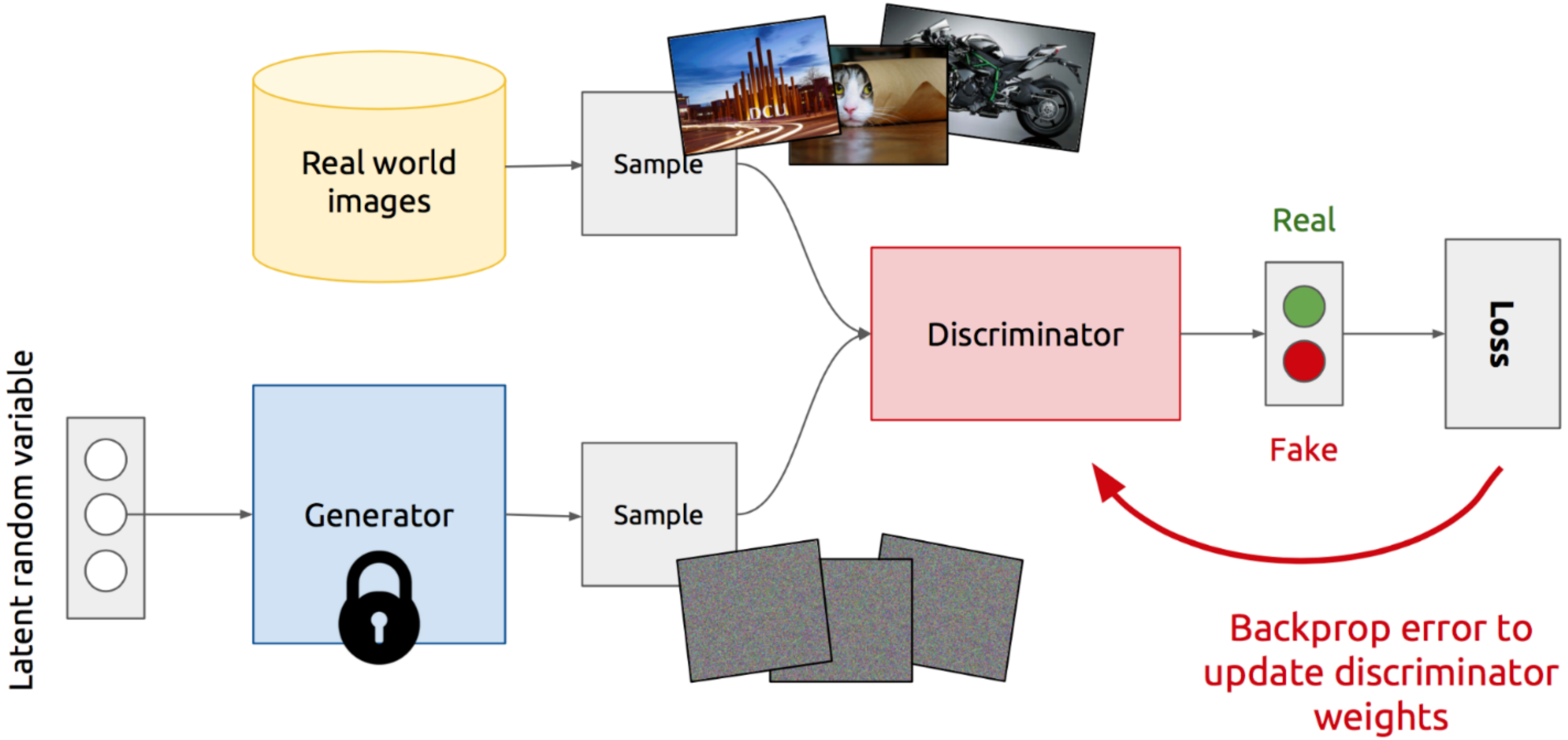


Image source: www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator

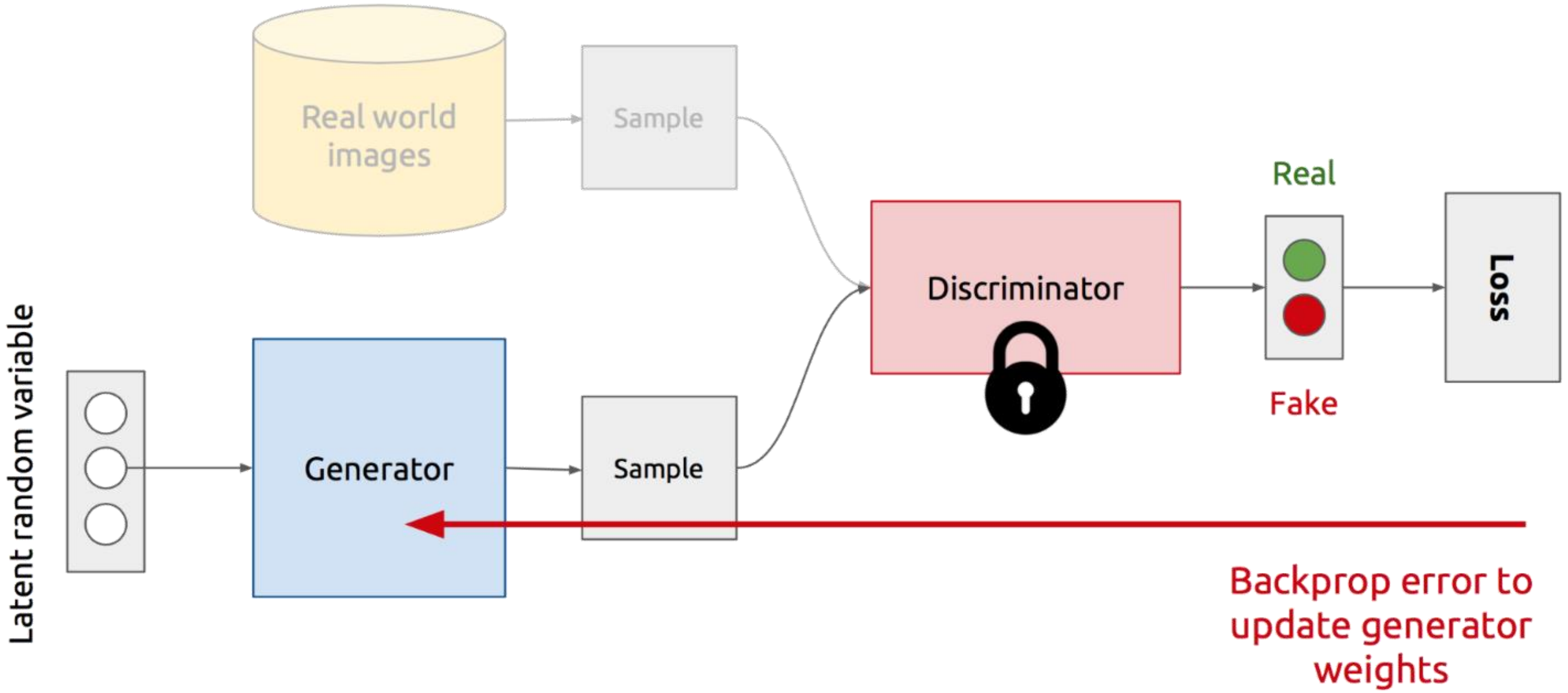


Image source: www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Generator In Action

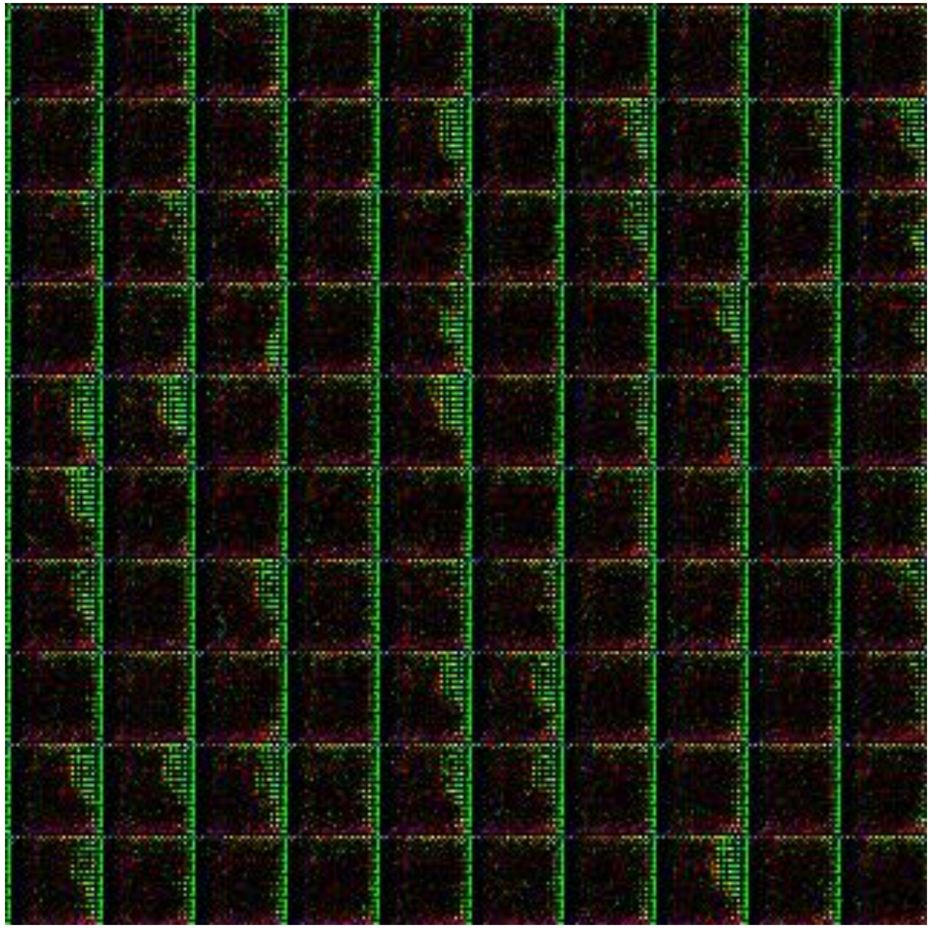


Image source:
openai.com/blog/generative-models/

GAN's Formulation

$$\min_{G} \max_{D} V(D, G)$$

- **It is formulated as a minmax game, where:**
  - The Discriminator is trying to maximize its reward $\boldsymbol{V(D, G)}$
  - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \boxed{\mathbb{E}_{x \sim p(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z \sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]}$$

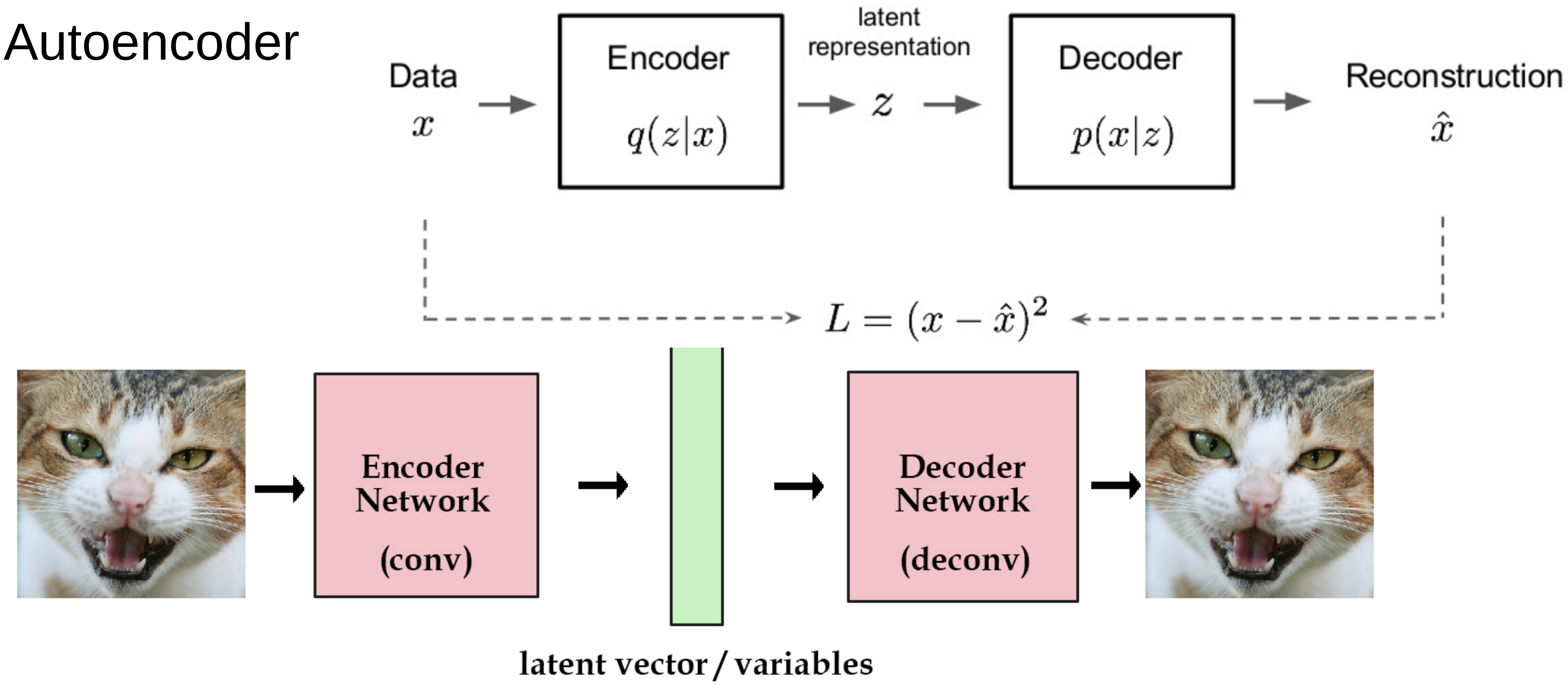
- **The Nash equilibrium of this particular game is achieved at:**
  - $P_{data}(x) = P_{gen}(x) \;\; \forall x$
  - $\mathrm{D}(x) = \frac{1}{2} \;\; \forall x$

- D(x) is the discriminator's estimate of the probability that real data instance x is real.
- E_x is the expected value over all real data instances.
- G(z) is the generator's output when given noise z.
- D(G(z)) is the discriminator's estimate of the probability that a fake instance is real.
- E_z is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
- The formula is derived from cross-entropy between

Variational Autoencoder

Variational Autoencoder (VAE)

- Autoencoder



Maximize Log-likelihood

Objective function:

$$\max \sum_i \log p(x_i)$$

$$p(x) = \int p(x|z)p(z)dz$$

Integrating over all possible z requires exponential time to compute. It is difficult to integrate in a neural network.

$$\log p_\theta(\mathbf{x}) \approx \log \frac{1}{N} \sum_j p_\theta(\mathbf{x}|\mathbf{z}_j)$$

Many sampled z will have a close-to-zero p(x|z)

Variational Autoencoder (VAE)

- Solution
  - Objective: maximize variational lower-bound
- Approximate the latent variable distribution
  - Approximate $p(z|x)$ using $q(z)$

Objective

$$\log p(x) - KL[q(z)] \parallel p(z|x)] = \int_z q(z) \log \frac{p(x|z)p(z)}{q(z)}$$

$$= \mathbb{E}_{z\sim q} \log p(x|z) - KL[q(z) \parallel p(z)]$$

- Maximize variational lower bound

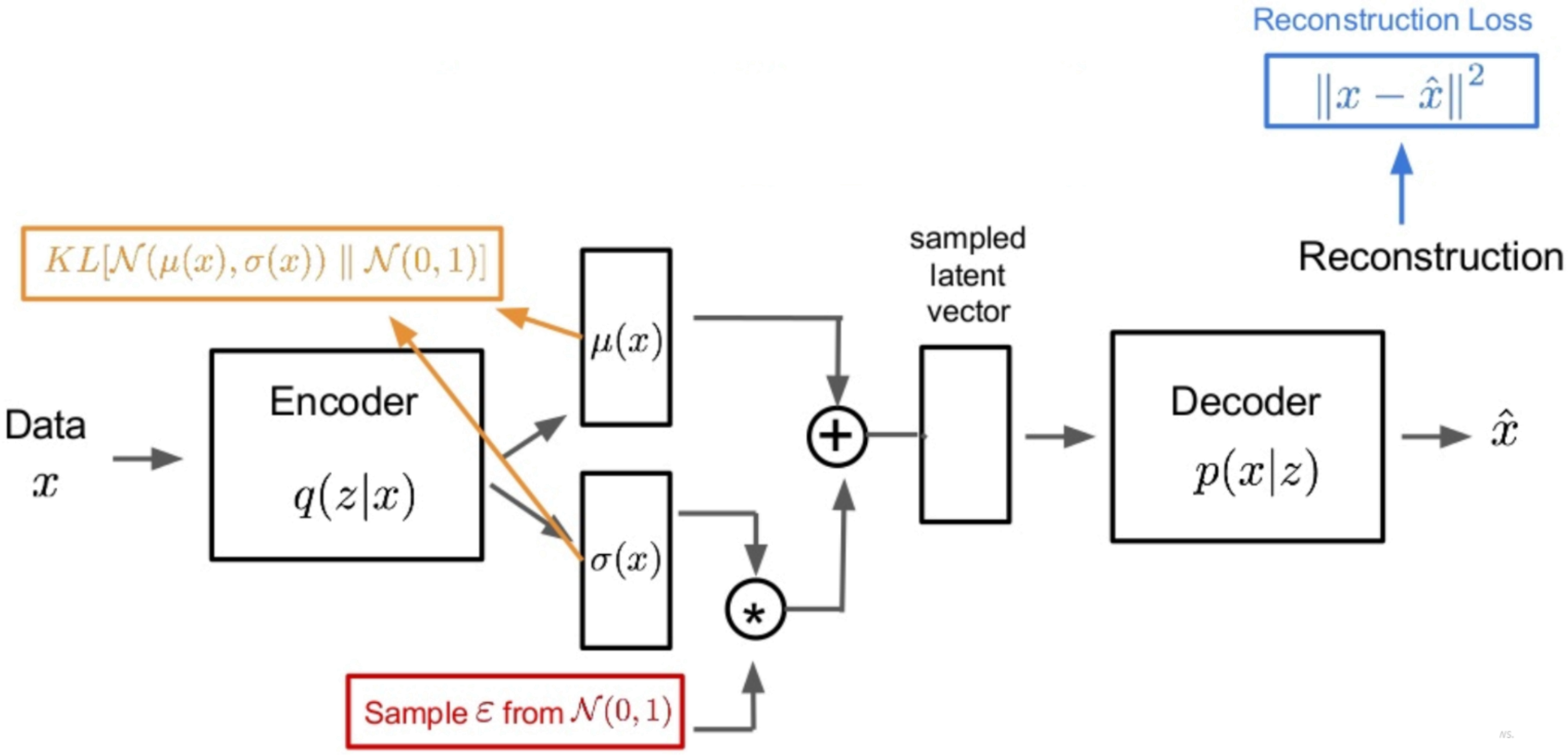
$$\log p(x) \geq \boxed{\mathbb{E}_{z\sim q} \log p(x|z)} - \boxed{KL[q(z) \parallel p(z)]}$$

Minimize reconstruction error:
Training samples have higher probability

Latent variable distribution
should be like the prior p(z)

Variational Autoencoder

- Maximize $\log p_\theta(\mathbf{x}_i) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i|\mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{x}_i)||p_\theta(\mathbf{z})]$



Variational Autoencoder

- Results

1st epoch

9th epoch

Training data

Autoregressive Model

Autoregressive Models (1/5)

- Generative model: given n examples $x^{(i)}$, recover $p(x)$
- Likelihood: $\prod_i p_\theta(x^i)$
- Model:

$$\begin{aligned}\theta^* &= arg\max_\theta \prod_i p_\theta(x^i)\\ &= arg\max_\theta \log \prod_i p_\theta(x^i)\\ &= arg\max_\theta \sum_i \log p_\theta(x^i)\end{aligned}$$

Autoregressive Models (2/5)

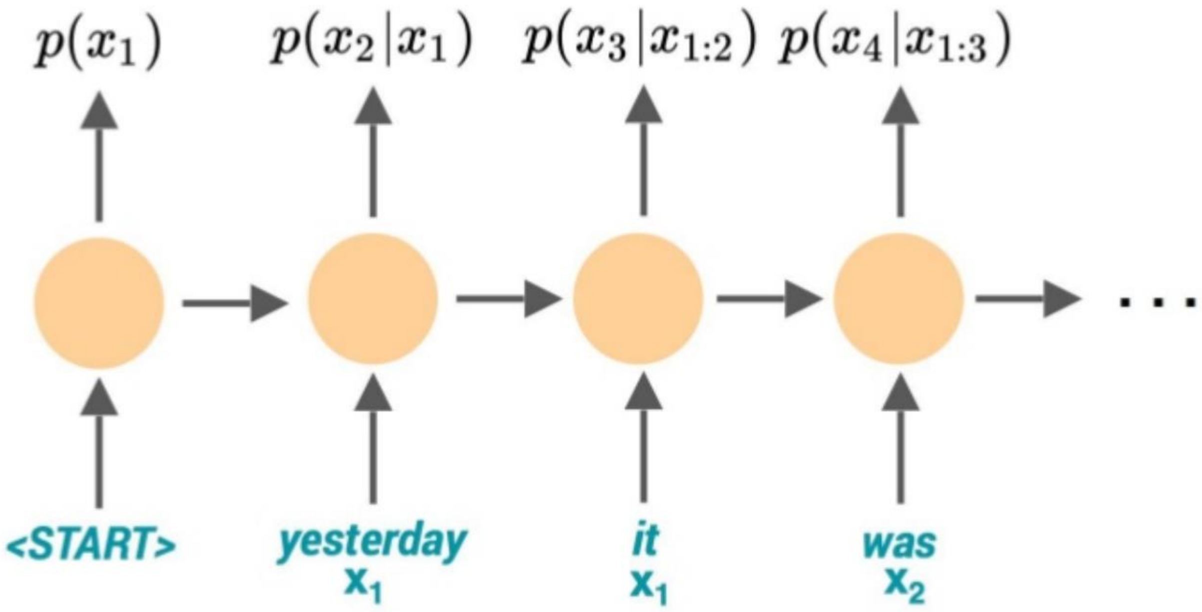
- Explicit formula based on chain rule

$$p_\theta(x) = p_\theta(x_1)\prod_{i=2}^{n} p_\theta(x_i|x_1, ..., x_{i-1})$$

- Generation:
  - Sample one step at a time, conditioned on all previous steps

Autoregressive Models (3/5)

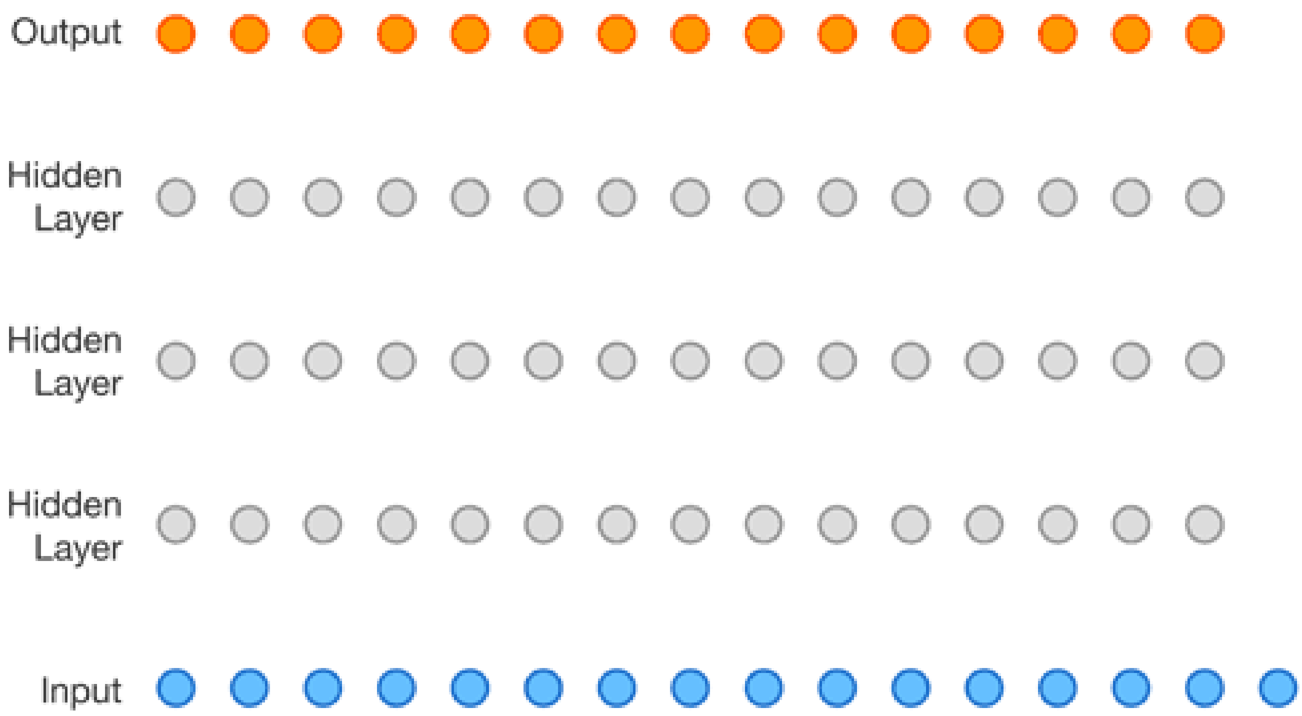
- Generate sentences



Autoregressive Models (4/5)

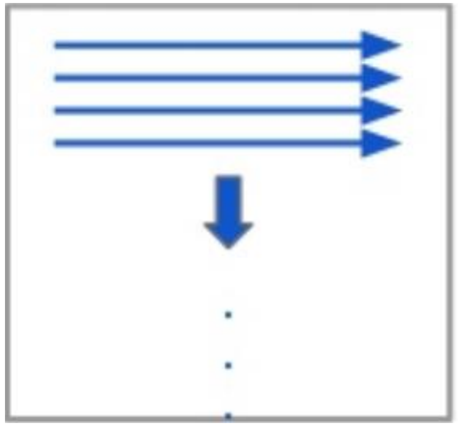
- Generate raw audio

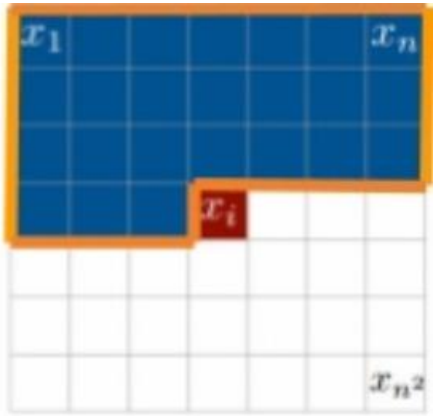
Demo

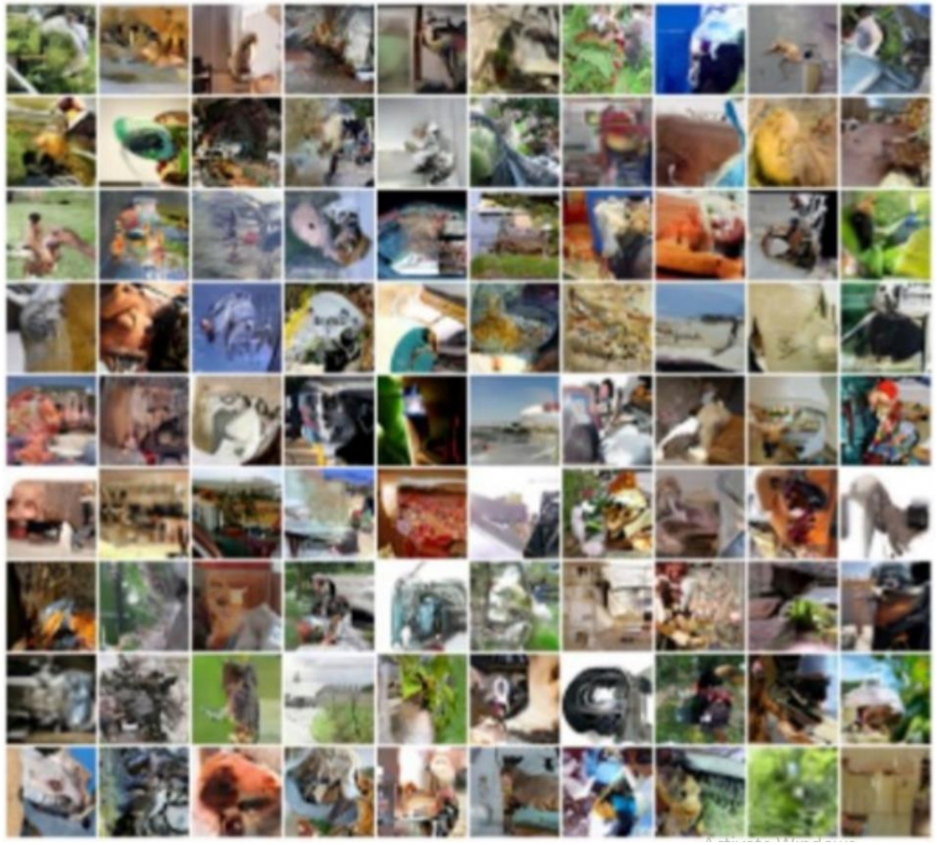


Autoregressive Models (5/5)

- Generate an image pixel by pixel







Oord *et al.*, Pixel Recurrent Neural Networks, 2016.

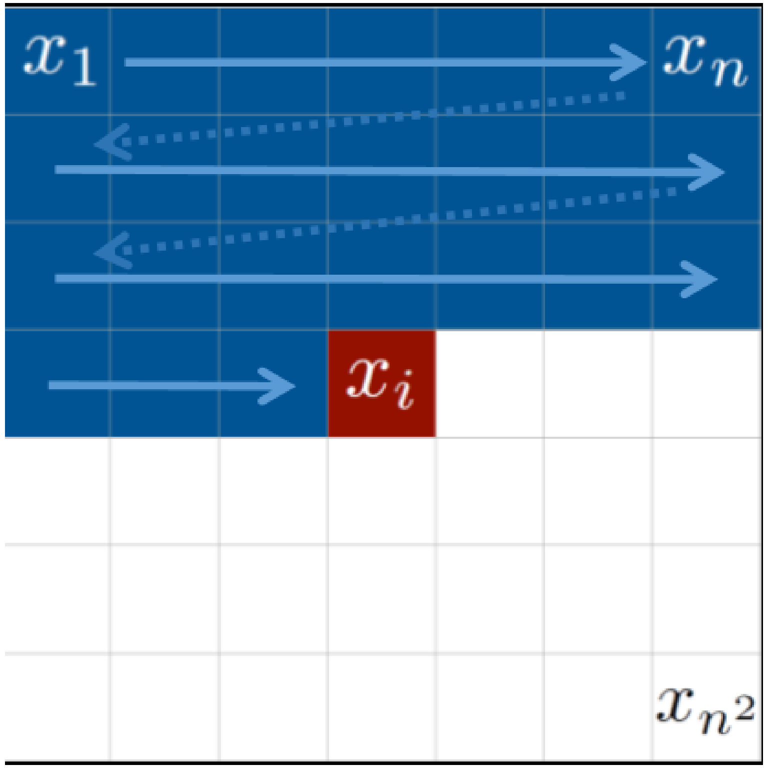
Intuition

$$p(\mathbf{x}) = p(x_1, x_2, ..., x_{n^2})$$

Likelihood:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

A sequential model!

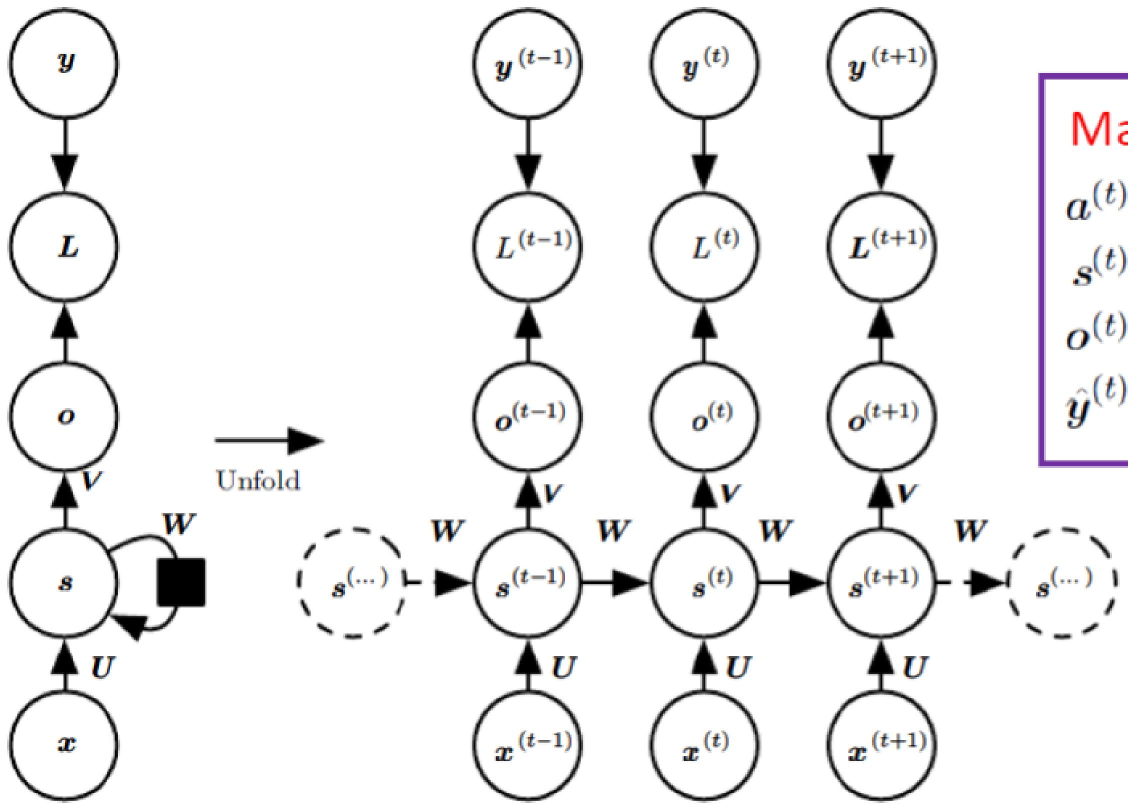


Oord *et al.*, Pixel Recurrent Neural Networks, 2016.

Transformer Model

(Autoregressive)

Recurrent Neural Networks (RNN)



Math formula:

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{s}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Figure from Deep Learning, Goodfellow, Bengio and Courville

Issues with RNN

- Vanishing/exploding gradients
  - When n hidden layers use an activation function like sigmoid, n small/large derivatives are multiplied together. So, the gradient increases/decreases exponentially as we propagate down to the initial layers.
- Short term memory
  - Difficulty accessing information from long time ago
- Slow computation for long sequences

Transformer Architecture

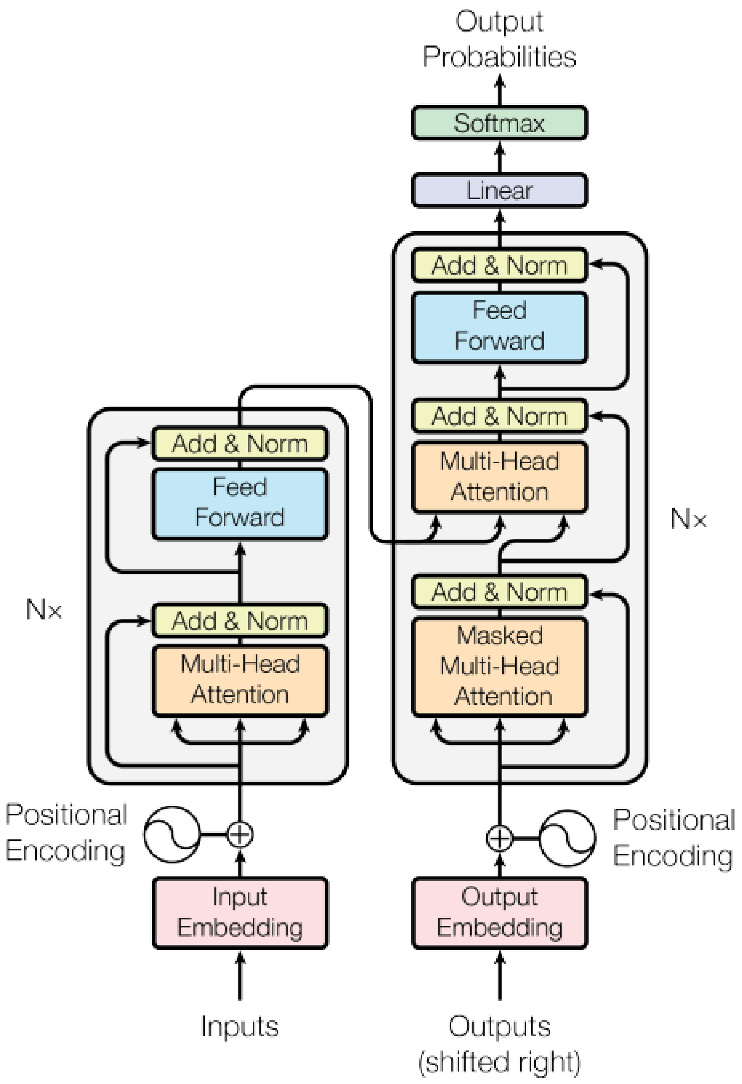
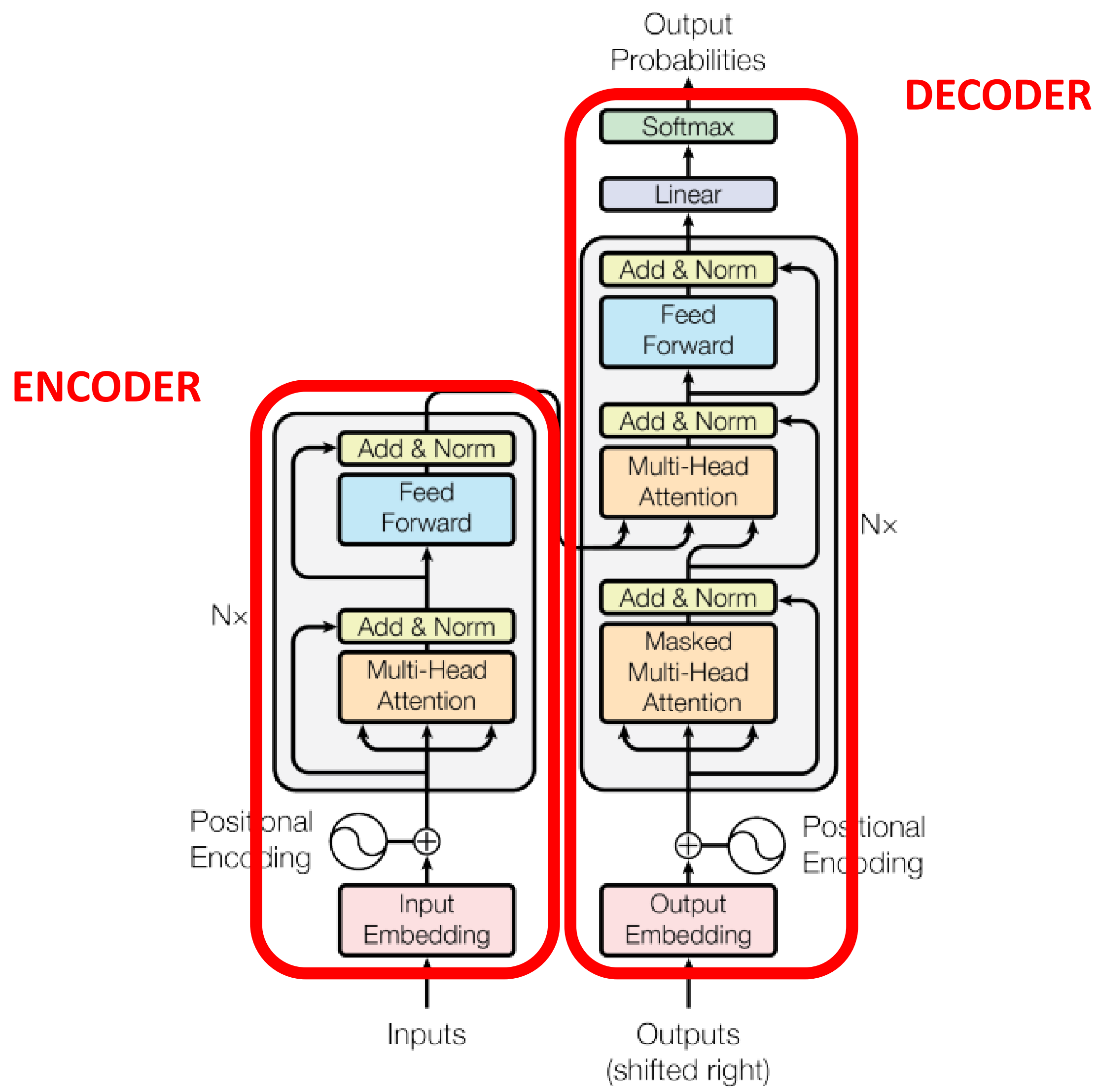
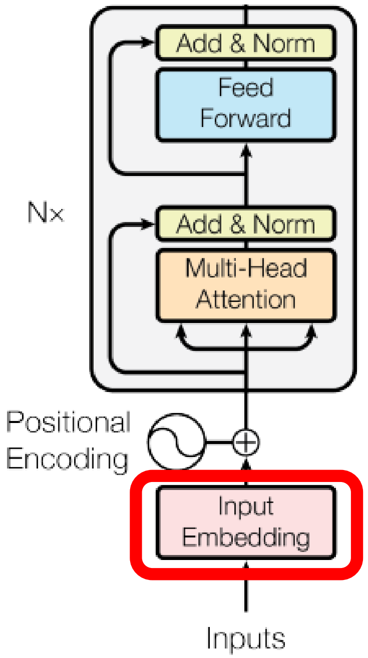


Figure from 'Attention is All You Need' paper by Vaswani et al.

Transformer Architecture



Input Embedding



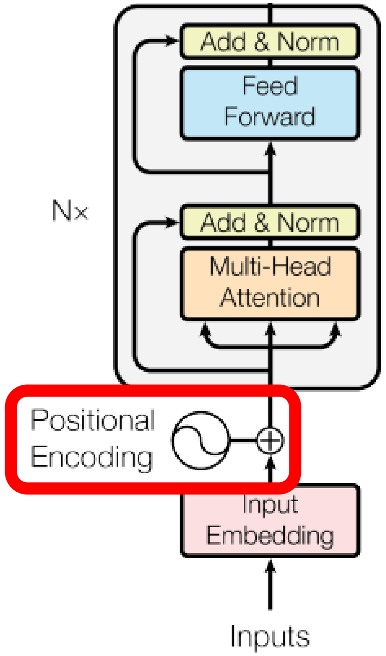
“This movie is great”

Input Embedding

Original sentence	THIS	MOVIE	IS	GREAT
Input ID (position in the vocabulary)	105	6587	5475	3578
Embedding (vector size of 512)	952.207 5450.84 1853.448 … 1.658 2671.428 659.014	171.411 3276.350 9192.819 … 3633.421 5478.325 4506.025	621.659 1304.051 0.565 … 8.764 5306.749 5119.949	6422.693 9358.778 2141.081 … 3068.778 3198.08 736.147

d_{model} = 512 is the size of the embedding vector for each word

Positional Encoding



Positional Encoding

- We want to store information about each word's position in the sentence
  - Treat words accordingly, i.e., based on their neighborhood with other words
- We want the positional encoding to represent a pattern that can be learned by the model

Input Embedding

Original sentence	THIS	MOVIE	IS	GREAT
Embedding	952.207 5450.84 … 1.658	171.411 3276.350 … 3633.421	621.659 0.565 … 8.764	6422.693 2141.081 … 3068.778
	+	+	+	+
Positional Embedding	2734.217 9250.814 … 1341.168	9433.411 8326.350 … 23.451	1431.659 1765.565 … 3212.767	9012.693 4211.081 … 32.779
	=	=	=	=
Encoder Input (vector of size 512)	---- ---- …	---- ---- …	---- ---- …	---- ---- …

Positional Encoding

- We compute positional encodings only once and reuse them

THIS	MOVIE	IS	GREAT

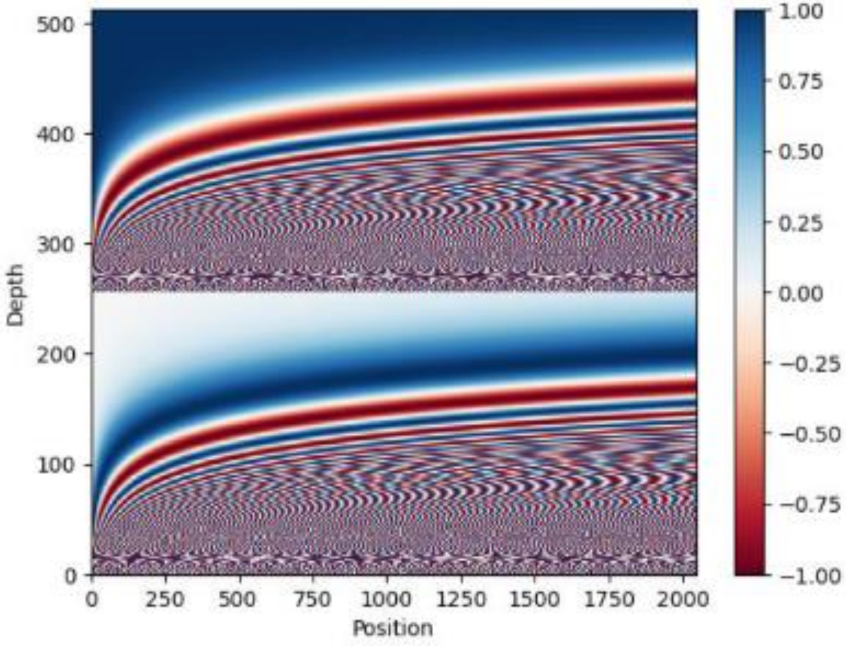
$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

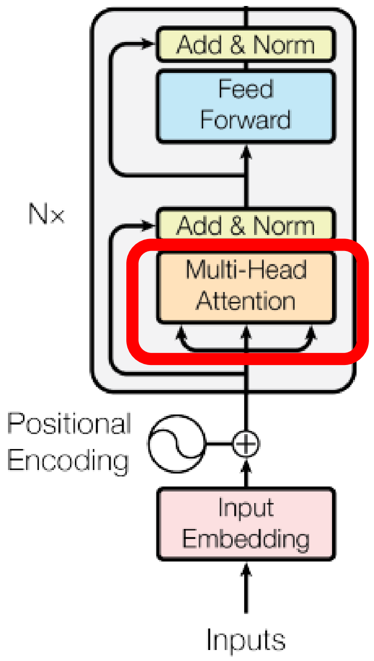
$$\begin{bmatrix} PE(0, 0) \\ PE(0, 1) \\ PE(0, 2) \\ PE(0, 3) \\ \dots \\ PE(0, 511) \end{bmatrix} \quad \begin{bmatrix} PE(1, 0) \\ PE(1, 1) \\ PE(1, 2) \\ PE(1, 3) \\ \dots \\ PE(1, 511) \end{bmatrix} \quad \begin{bmatrix} PE(2, 0) \\ PE(2, 1) \\ PE(2, 2) \\ PE(2, 3) \\ \dots \\ PE(2, 511) \end{bmatrix} \quad \begin{bmatrix} PE(3, 0) \\ PE(3, 1) \\ PE(3, 2) \\ PE(3, 3) \\ \dots \\ PE(3, 511) \end{bmatrix}$$

Positional Encoding

- Why trigonometric functions?
  - Sin and Cos represent a pattern that the model can recognize as continuous
  - Relative positions are easier to distinguish
  - Plot of these functions shows the pattern



Multi-Head Attention



Self Attention

- Allows the model to uncover the relationship between words
- This simple case considers the sequence length seq=4 and d_k=512
- The matrices *Q*, *K*, and *V* are just input sentences

$$\text{softmax}\left[\frac{\underset{(4,\,512)}{Q} \times \underset{(512,\,4)}{K^T}}{\sqrt{512}}\right] =$$

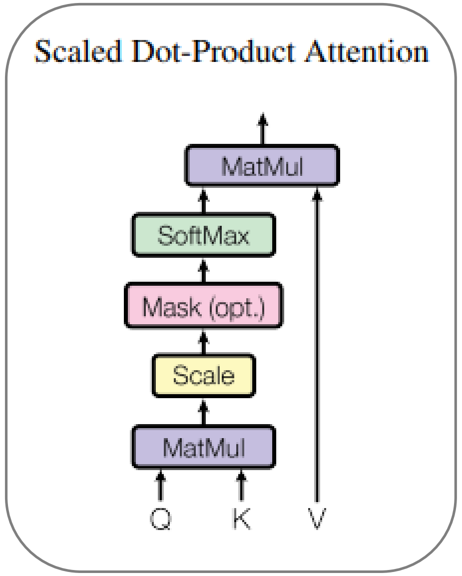
	THIS	MOVIE	IS	GREAT
THIS	0.568	0.264	0.212	0.039
MOVIE	0.368	0.564	0.012	0.139
IS	0.178	0.364	0.512	0.087
GREAT	0.103	0.264	0.112	0.539

(4, 4)

$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- All values are random
- All rows sum up to 1
- For the sake of simplicity, considering only one head

Self Attention



- Each row in the $Attention$ matrix captures not only the meaning (provided by the embedding) or the position (provided by the positional encoding) in the sentence, but also each word's relationship with others.

$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

	THIS	MOVIE	IS	GREAT
THIS	0.568	0.264	0.212	0.039
MOVIE	0.368	0.564	0.012	0.139
IS	0.178	0.364	0.512	0.087
GREAT	0.103	0.264	0.112	0.539

(4, 4)

x

V

(4, 512)

=

$Attention$

(4, 512)

Self Attention

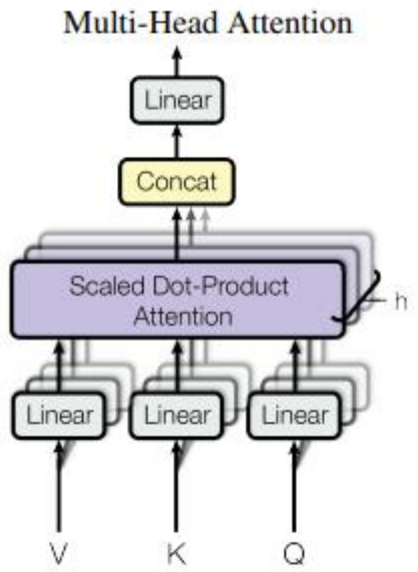
- So far, we haven’t used any parameters in self attention.
- The relationship between words has been driven by embeddings and the positional encodings, which will change later.
- Self attention is permutation invariant.
- To prevent interaction between words, we can set their values to -∞ before applying $softmax$ in this matrix. This will be used in the decoder to mask unseen/future words.
- Values along the diagonal in the matrix to be the highest.

Multi-head Attention

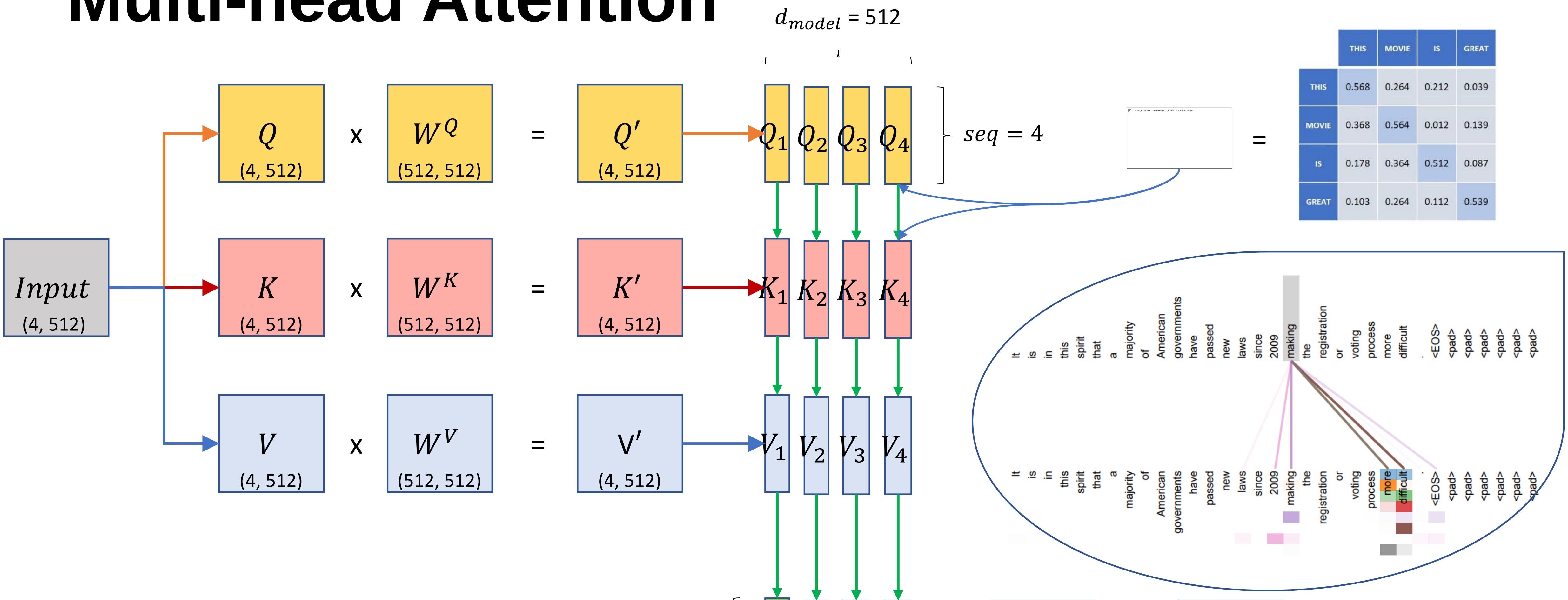
$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$MultiHead(Q, K, V) = Concat(head_1\ \ldots head_h)W^O$$

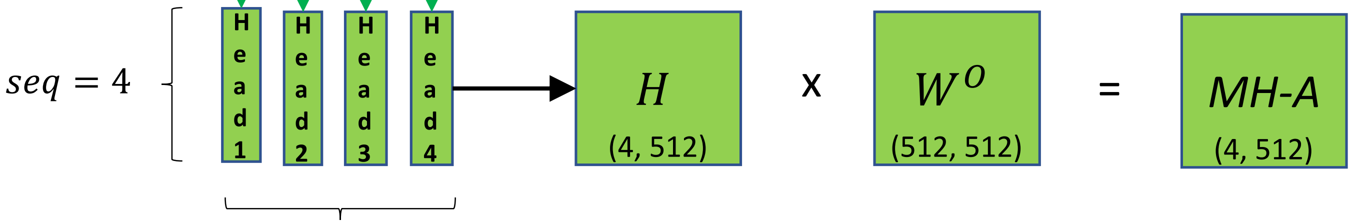
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



Multi-head Attention



	THIS	MOVIE	IS	GREAT
THIS	0.568	0.264	0.212	0.039
MOVIE	0.368	0.564	0.012	0.139
IS	0.178	0.364	0.512	0.087
GREAT	0.103	0.264	0.112	0.539



d_{model} = 512

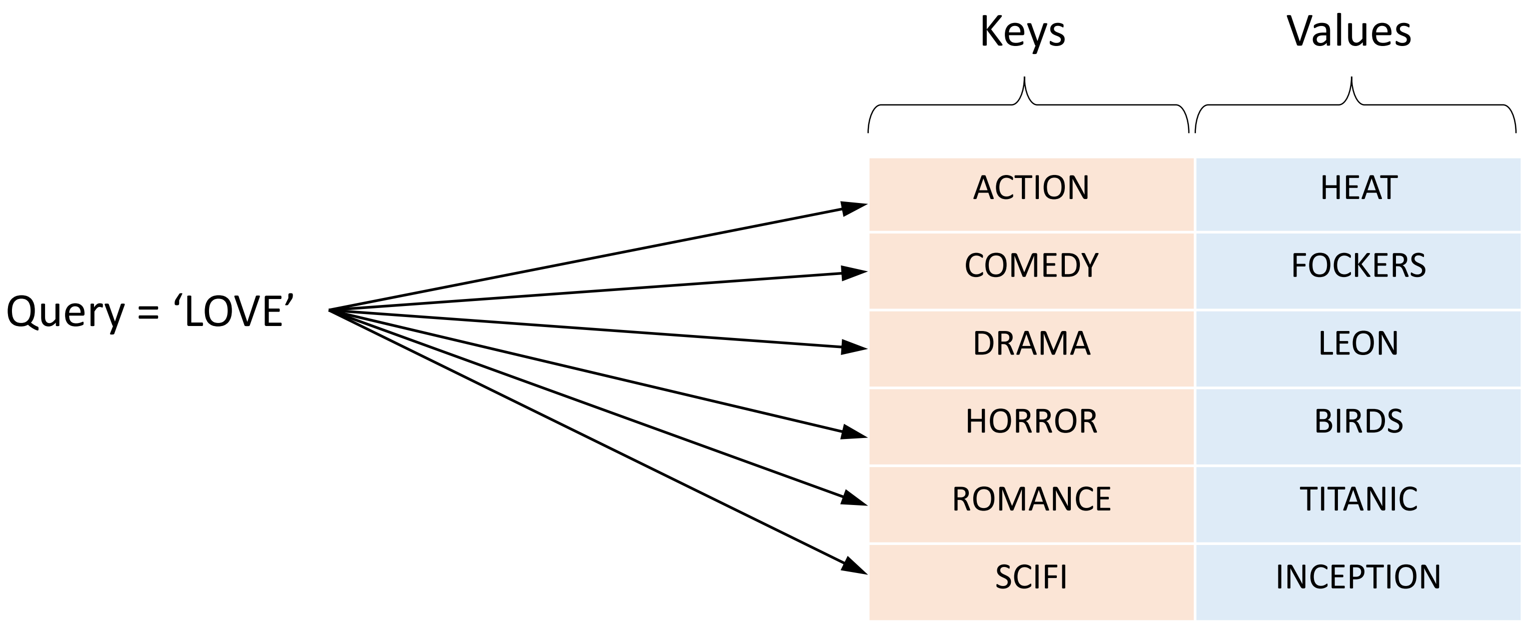
$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q,K,V) = Concat(head_1 \ \ldots head_h)W^O$$

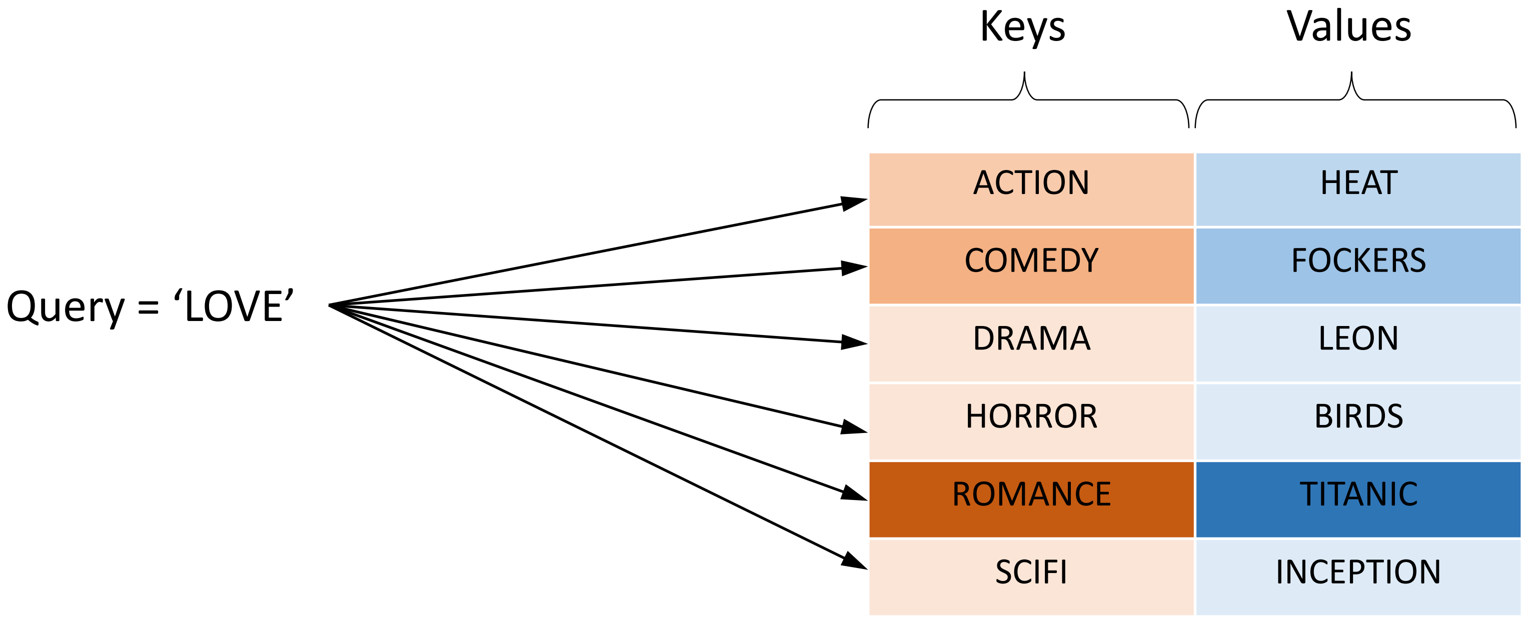
Query, Keys, and Values

- Concept borrowed from Information Retrieval, DB
  - Similar to Python-like dictionaries

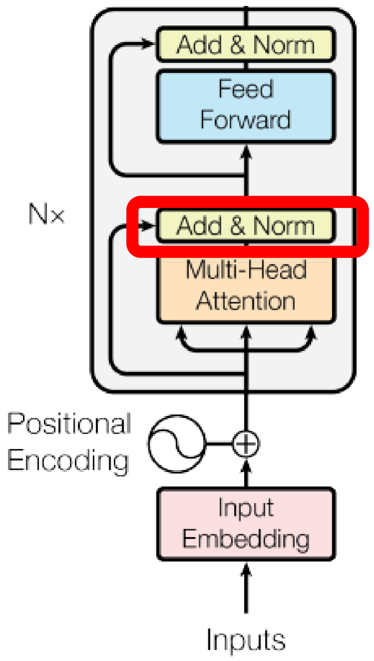


Query, Keys, and Values

- Concept borrowed from Information Retrieval, DB
  - Similar to Python-like dictionaries

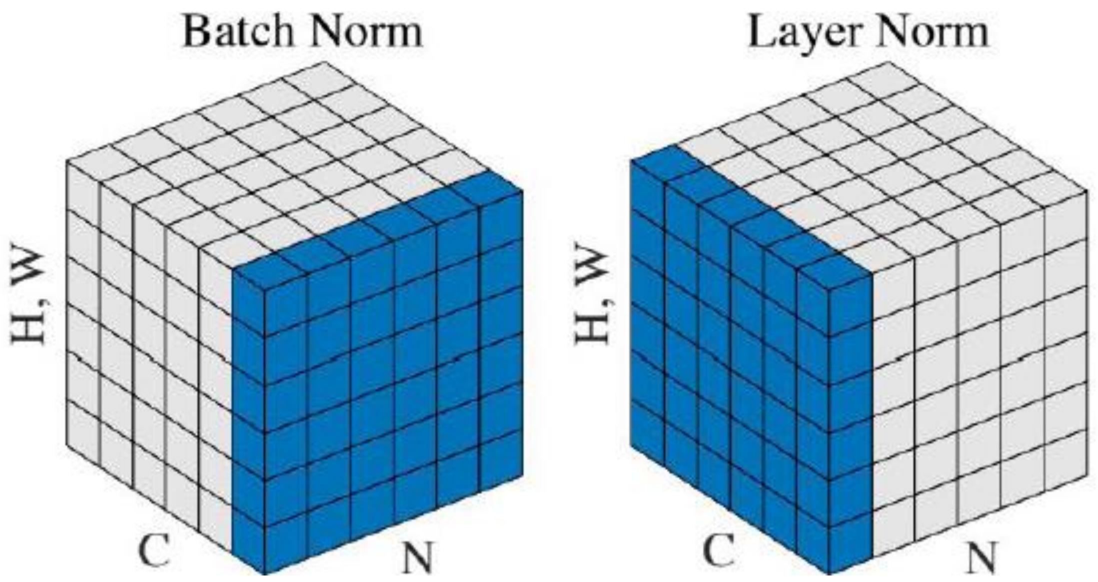


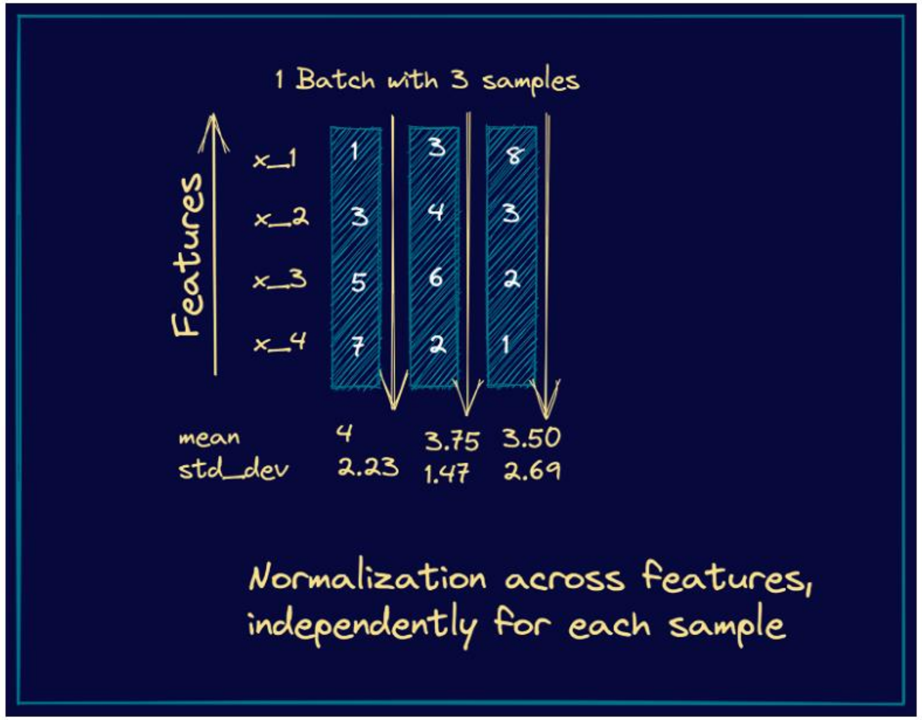
Add & Norm


Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs

Layer Normalization

https://www.pinecone.io/learn/batch-layer-normalization/





In “*Batch Normalization*”, mean and variance are calculated ***for*** each individual channel ***across*** all samples and both spatial dimensions.

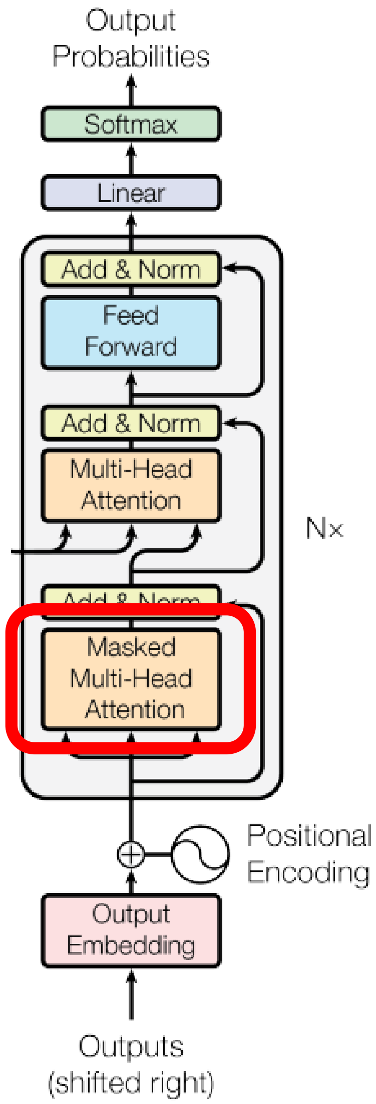
In “*Layer Normalization*”, mean and variance are calculated ***for*** each individual sample ***across*** all channels and both spatial dimensions.

$$\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i \quad (1)$$

$$\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2 \quad (2)$$

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2}} \quad (3)$$

Masked Multi-head Attention

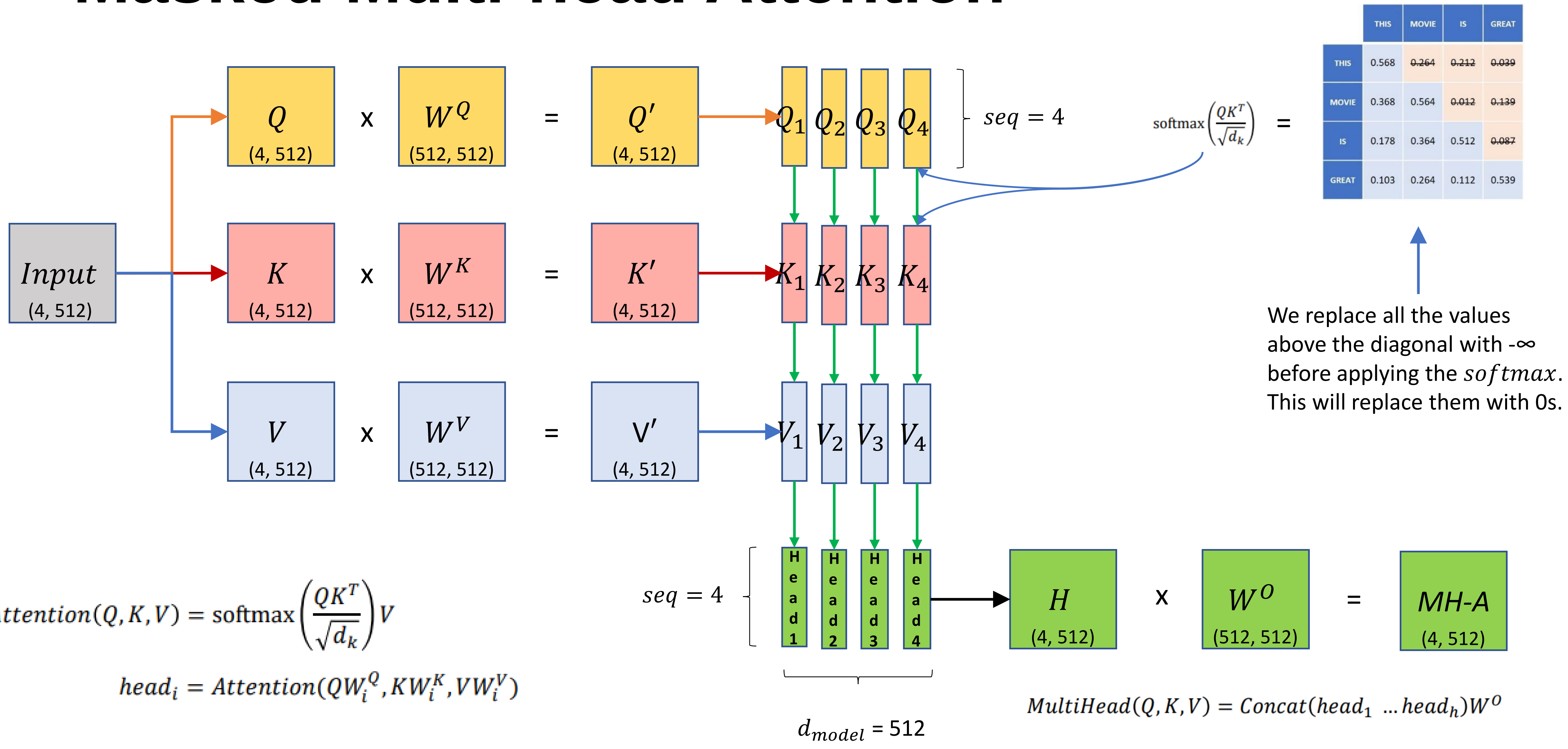


Masked Multi-head Attention

- We want to make the model causal, i.e., the output at a certain position should be based on the previous words the models has seen
- The future words must be masked from the model

	THIS	MOVIE	IS	GREAT
THIS	0.568	~~0.264~~	~~0.212~~	~~0.039~~
MOVIE	0.368	0.564	~~0.012~~	~~0.139~~
IS	0.178	0.364	0.512	~~0.087~~
GREAT	0.103	0.264	0.112	0.539

Masked Multi-head Attention



	THIS	MOVIE	IS	GREAT
THIS	0.568	~~0.264~~	~~0.212~~	~~0.039~~
MOVIE	0.368	0.564	~~0.012~~	~~0.139~~
IS	0.178	0.364	0.512	~~0.087~~
GREAT	0.103	0.264	0.112	0.539

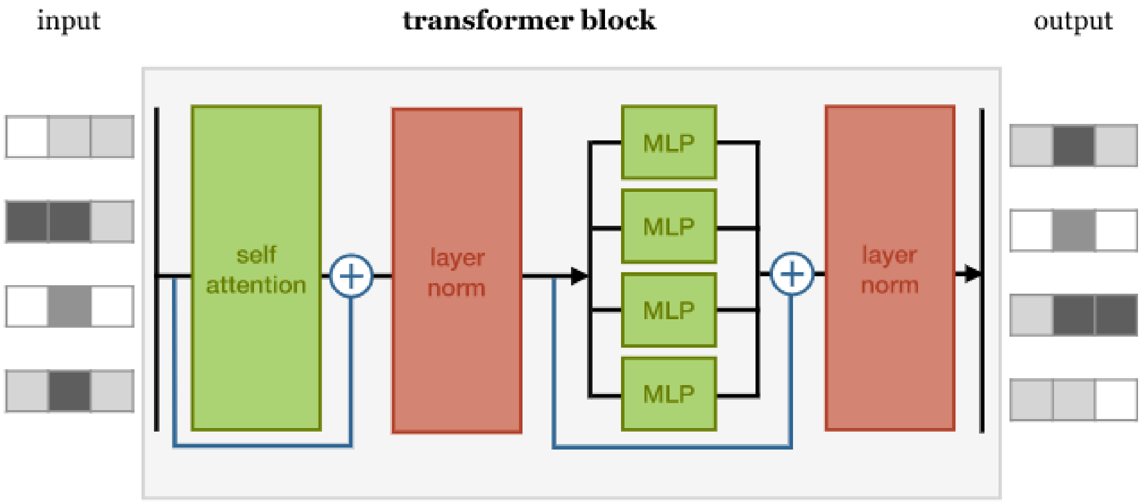
$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q,K,V) = Concat(head_1 \; ... head_h)W^O$$

Building a Transformer

- A transformer is not just a self-attention layer, it is an *architecture*.
- The block applies in sequence:
  - A self attention layer, layer normalization, feed forward layer (a single MLP applied independently to each vector), and another layer normalization.
  - Residual connections are added around both, before the normalization.



https://peterbloem.nl/blog/transformers

Training

All happens in parallel!
(in time step)

Dondurmayi kim sevmez<EOS>

Cross Entropy Loss

Softmax
(seq, vocab_size)

Linear
(seq, d_{model}) -> (seq, vocab_size)

Encoder Output
(seq, d_{model})

Decoder Output
(seq, d_{model})

Encoder

Decoder

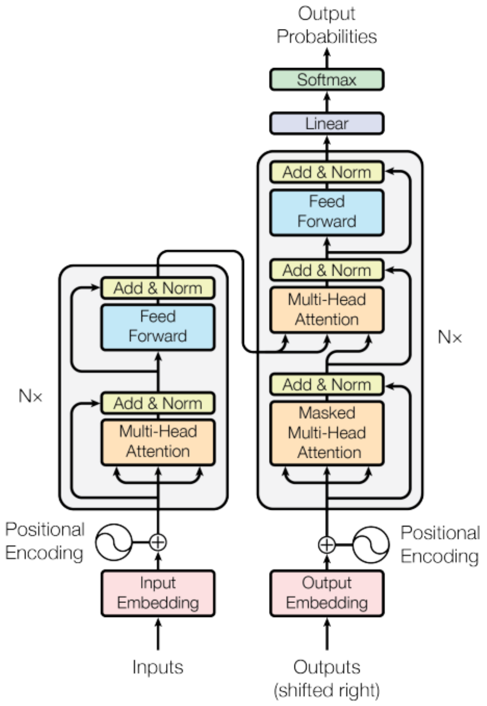
Encoder Input
(seq, d_{model})

Decoder Input
(seq, d_{model})

Decoder input is shifted right, i.e., we prepend <SOS> at the beginning

<SOS>Who does not like ice cream<EOS>

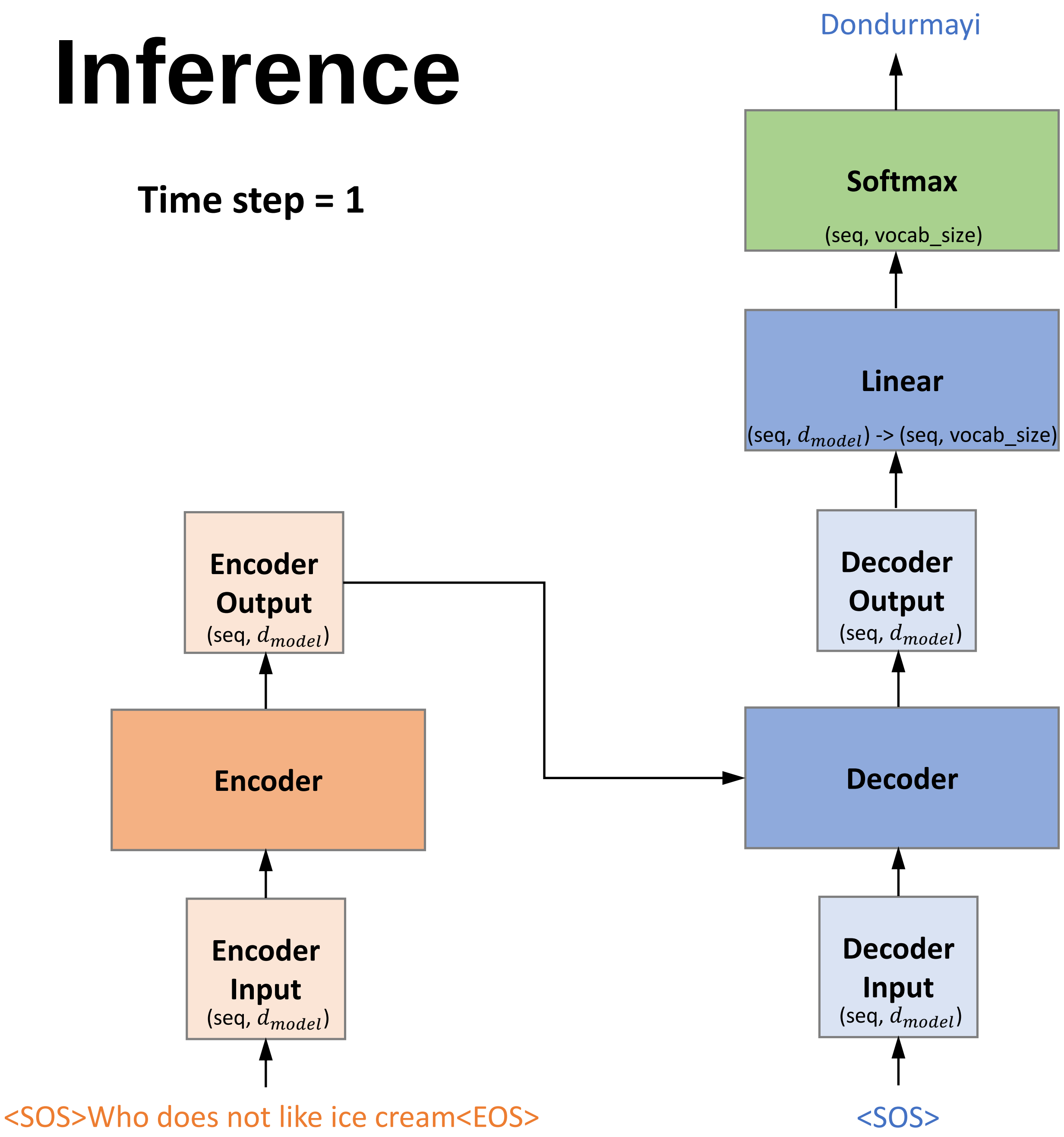
<SOS>Dondurmayi kim sevmez

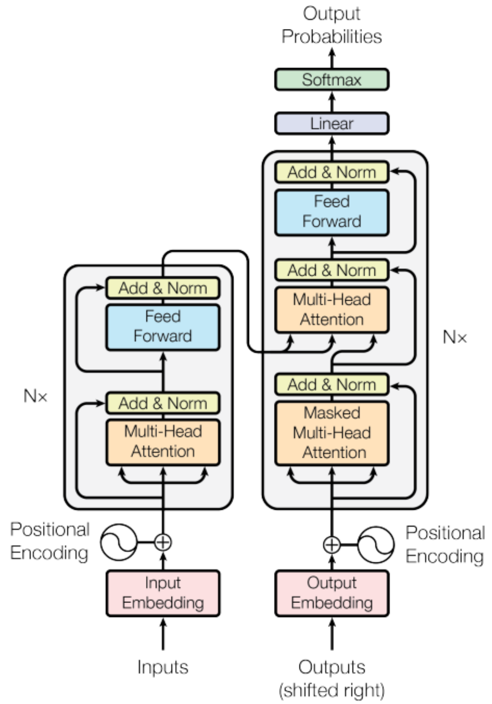


Inference

Time step = 1

Most probable word (token) is selected corresponding to the position from the vocabulary

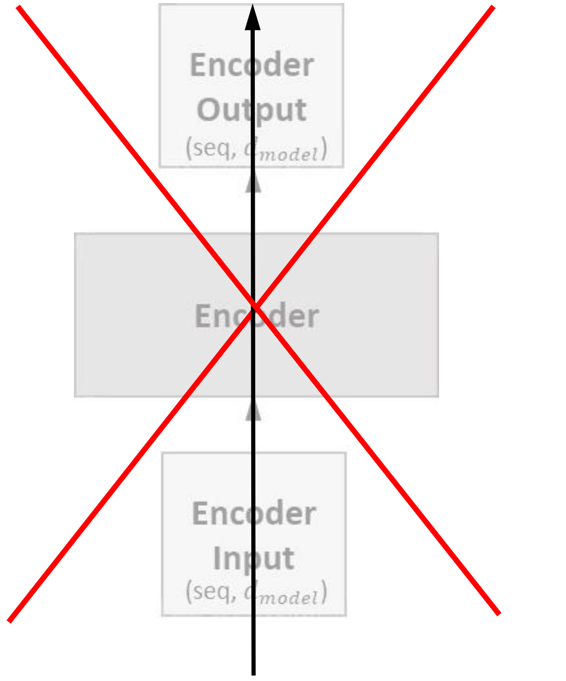




Inference

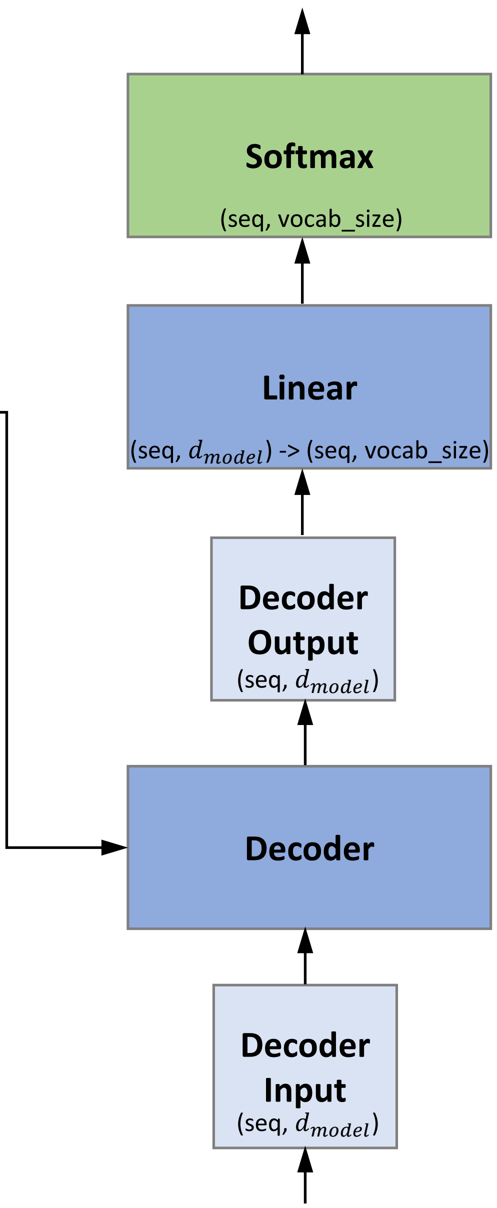
Time step = 2

We don't generate encoder output again. The output from the 1st time step is reused



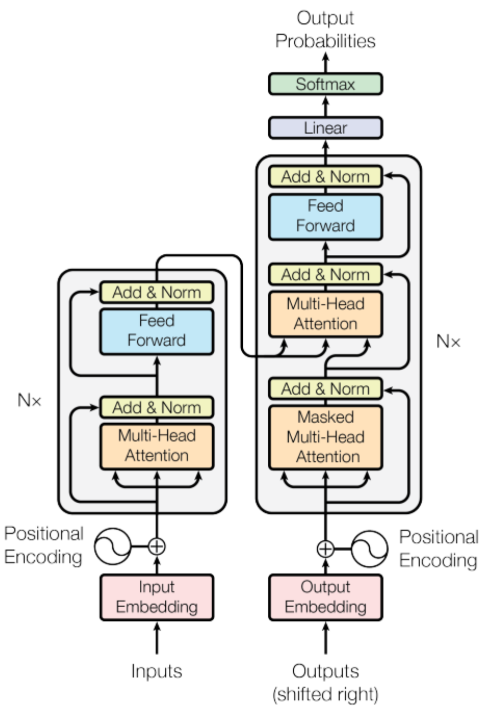
<SOS>Who does not like ice cream<EOS>

kim



<SOS>Dondurmayi

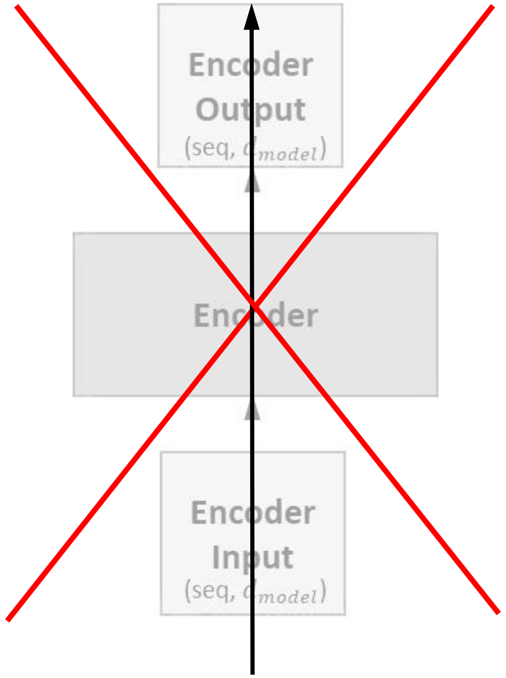
Most probable word (token) is selected corresponding to the position from the vocabulary

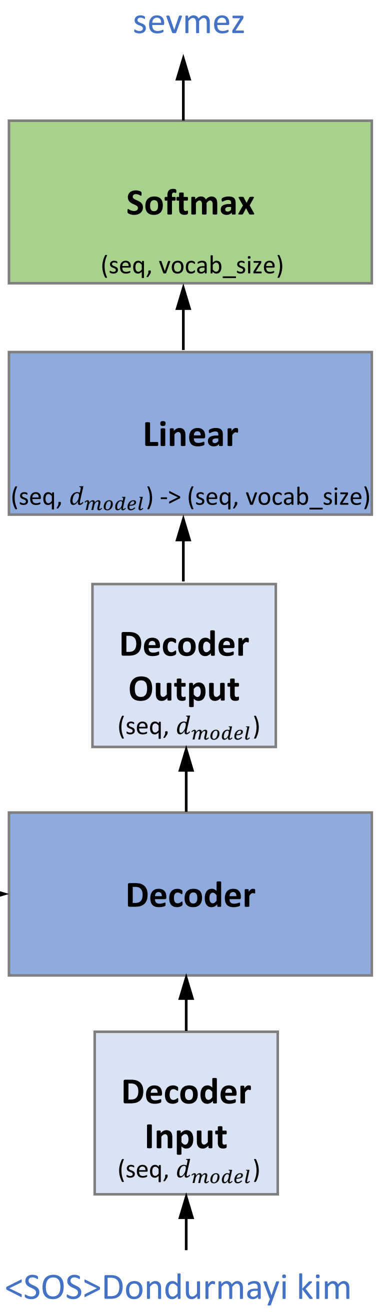


Inference

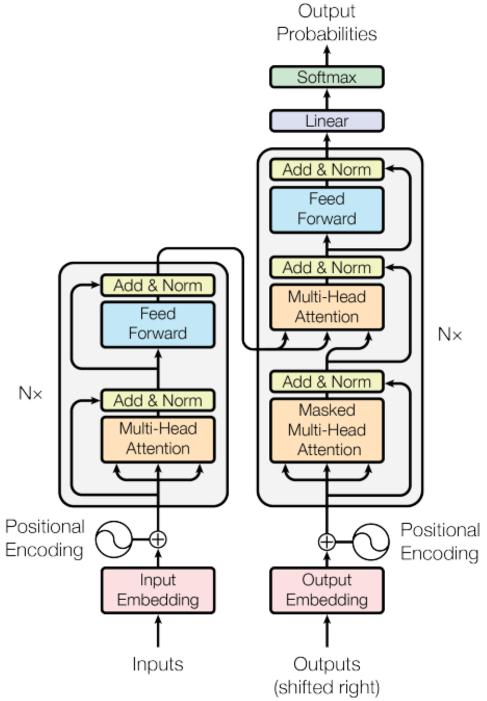
Time step = 3

We don't generate encoder output again. The output from the 1st time step is reused





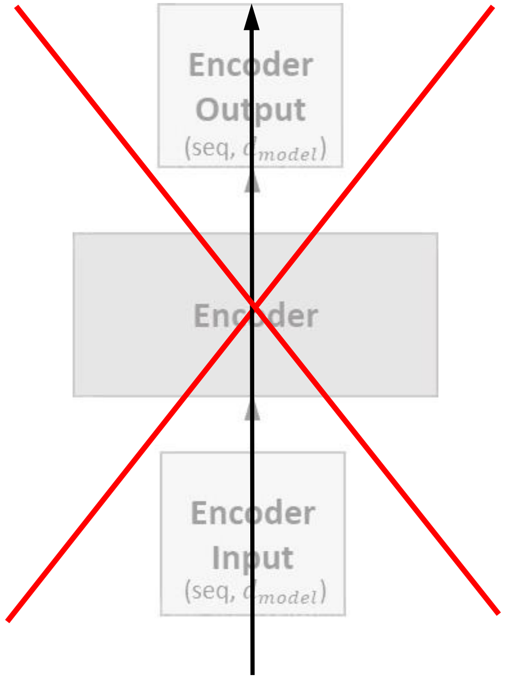
Most probable word (token) is selected corresponding to the position from the vocabulary



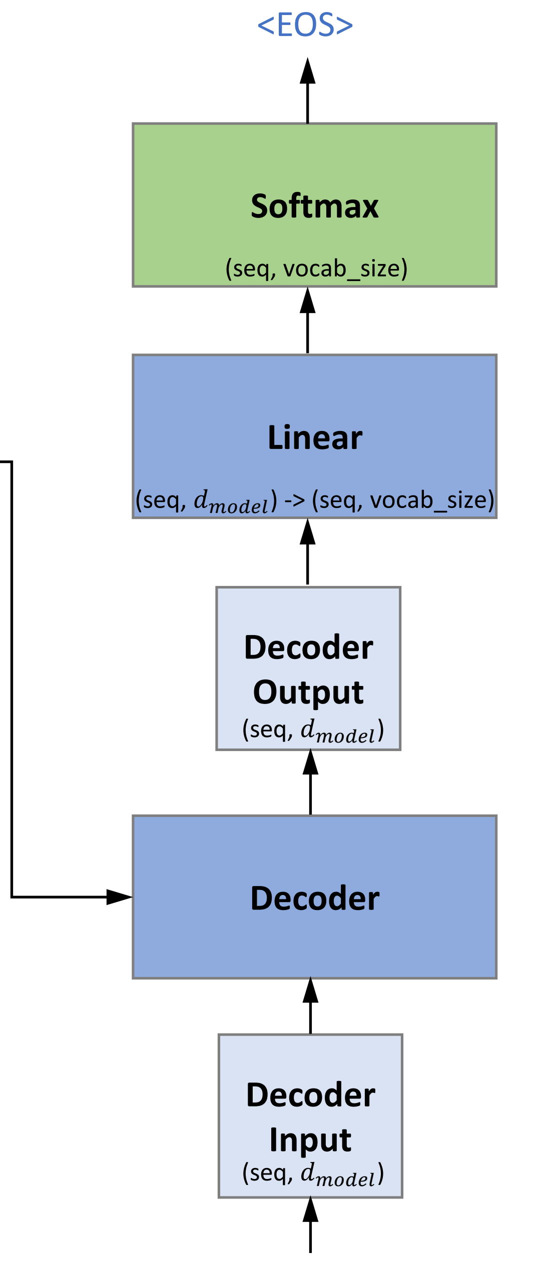
Inference

Time step = 4

We don't generate encoder output again. The output from the 1st time step is reused

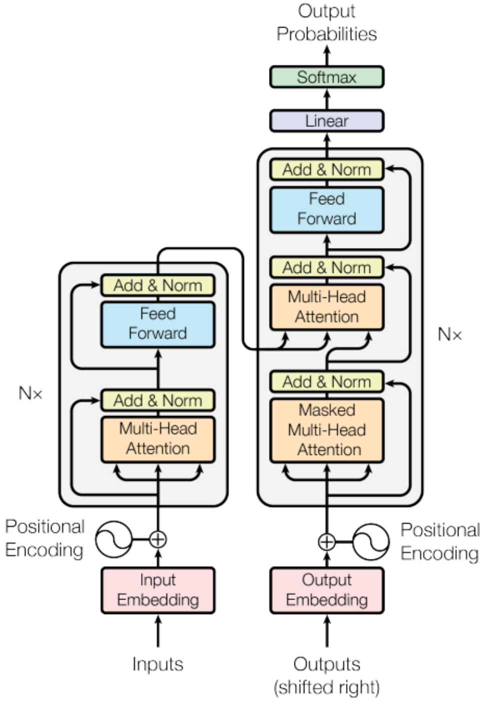


<SOS>Who does not like ice cream<EOS>



<SOS>Dondurmayi kim sevmez

Most probable word (token) is selected corresponding to the position from the vocabulary



Inference

- Selecting the most probable word (token) from the vocabulary at each time step may not yield the best translation. Why?
  - Greedy search
- An alternative to greedy search is Beam Search
  - Consider n-top probable words
  - Increased time complexity (slower), increased accuracy (performs better, overall)