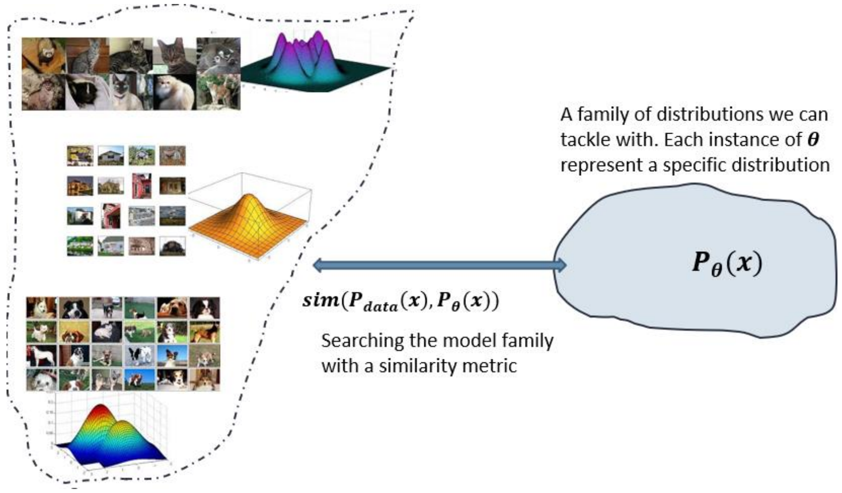
# Autoregressive models

22-808: Generative models
Sharif University of Technology
Fall 2025

Fatemeh Seyyedsalehi

## Recap


A family of distributions we can tackle with. Each instance of $\theta$ represent a specific distribution
$P_\theta(x)$
$sim(P_{data}(x), P_\theta(x))$
Searching the model family with a similarity metric

- We need a framework to interact with distributions for statistical generative models.
  - Probabilistic generative models
    - Representation – Inference – Sampling – Learning
    - Intro. to causal modeling
  - Deep generative models
    - **Autoregressive models**
      - **First family of models we talk about**

## PGMs Vs. Deep neural networks

- Chain rule

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)p(x_4 \mid x_1, x_2, x_3)$$

- PGMS
  - Assumes conditional independencies

$$p(x_1, x_2, x_3, x_4) \approx p_{\mathrm{CPT}}(x_1)p_{\mathrm{CPT}}(x_2 \mid x_1)p_{\mathrm{CPT}}(x_3 \mid \cancel{x_1}, x_2)p_{\mathrm{CPT}}(x_4 \mid x_1, \cancel{x_2, x_3})$$

- Neural networks
  - Assumes specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.

$$p(x_1, x_2, x_3, x_4) \approx p(x_1)p(x_2 \mid x_1)p_{\mathrm{Neural}}(x_3 \mid x_1, x_2)p_{\mathrm{Neural}}(x_4 \mid x_1, x_2, x_3)$$

# Deep neural networks for classification

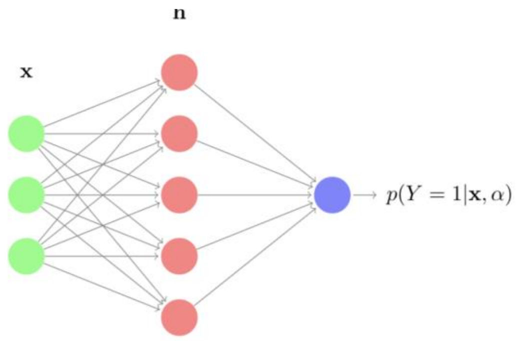
- Binary classification
  - We want to find

$$p(Y=1 \mid \mathbf{x};\alpha) = f(\mathbf{x},\alpha)$$

  - Logistic regression $z(\alpha,\mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i$

$$p_{\text{logit}}(Y=1 \mid \mathbf{x};\alpha) = \sigma(z(\alpha,\mathbf{x})),\ \text{where } \sigma(z) = 1/(1+e^{-z})$$

  - Neural networks $\mathbf{h}(A,\mathbf{b},\mathbf{x})$
    - More flexible $p_{\text{Neural}}(Y=1 \mid \mathbf{x};\alpha,A,\mathbf{b}) = \sigma(\alpha_0 + \sum_{i=1}^{h} \alpha_i \mathbf{h}_i)$
    - More parameters: A, b,α
    - Repeat multiple times to get a multilayer perceptron (neural network)



## Autoregressive models

- We can pick an ordering of all the random variables
  - Ordering of pixels in an image

- Without loss of generality, we can use chain rule for factorization

`

$$p(x_1, \cdots, x_{784}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

- We assume some parametric functions for CPDs

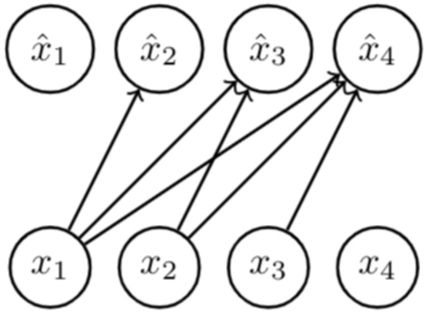
$$p(x_1, \cdots, x_{784}) = p_{\mathrm{CPT}}(x_1; \alpha^1) p_{\mathrm{logit}}(x_2 \mid x_1; \boldsymbol{\alpha}^2) p_{\mathrm{logit}}(x_3 \mid x_1, x_2; \boldsymbol{\alpha}^3) \cdots$$
$$p_{\mathrm{logit}}(x_n \mid x_1, \cdots, x_{n-1}; \boldsymbol{\alpha}^n)$$

- $p_{\mathrm{CPT}}(X_1 = 1; \alpha^1) = \alpha^1$, $p(X_1 = 0) = 1 - \alpha^1$
- $p_{\mathrm{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$
- $p_{\mathrm{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\alpha}^3) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$

# Fully Visible Sigmoid Belief Network (FVSBN)

- Each variable is a binary random variable given others

$$\hat{x}_i = p(X_i = 1 | x_1, \cdots, x_{i-1}; \boldsymbol{\alpha}^i) = p(X_i = 1 | x_{<i}; \boldsymbol{\alpha}^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$



FVSBN

- How to evaluate a joint probability

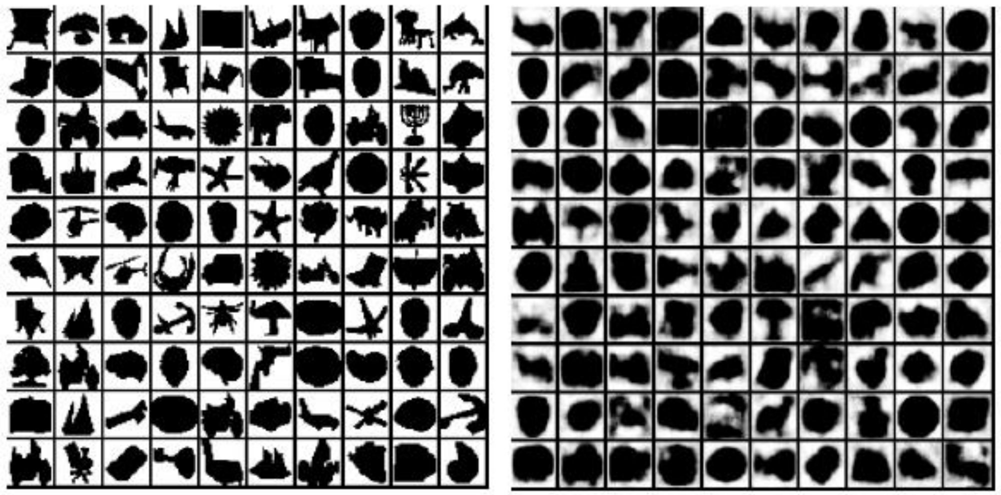
$$p(X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 0) = (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4)$$
$$= (1 - \hat{x}_1) \times \hat{x}_2(X_1 = 0) \times \hat{x}_3(X_1 = 0, X_2 = 1) \times (1 - \hat{x}_4(X_1 = 0, X_2 = 1, X_3 = 1))$$

- How to sample

1. Sample $\overline{x}_1 \sim p(x_1)$ (`np.random.choice([1,0],p=[`$\hat{x}_1, 1 - \hat{x}_1$`])`)
2. Sample $\overline{x}_2 \sim p(x_2 \mid x_1 = \overline{x}_1)$
3. Sample $\overline{x}_3 \sim p(x_3 \mid x_1 = \overline{x}_1, x_2 = \overline{x}_2) \cdots$

Ac

## Fully Visible Sigmoid Belief Network (FVSBN)



Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right.
Figure from *Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015*.

## NADE: Neural Autoregressive Density Estimation

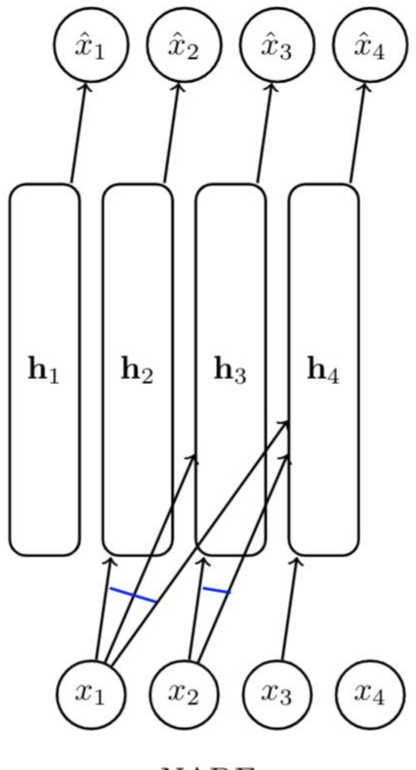
- To increase the power of the model we use neural networks instead of logistic regression

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{x}_i = p(x_i | x_1, \cdots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \alpha_i, b_i}_{\text{parameters}}) = \sigma(\alpha_i \mathbf{h}_i + b_i)$$

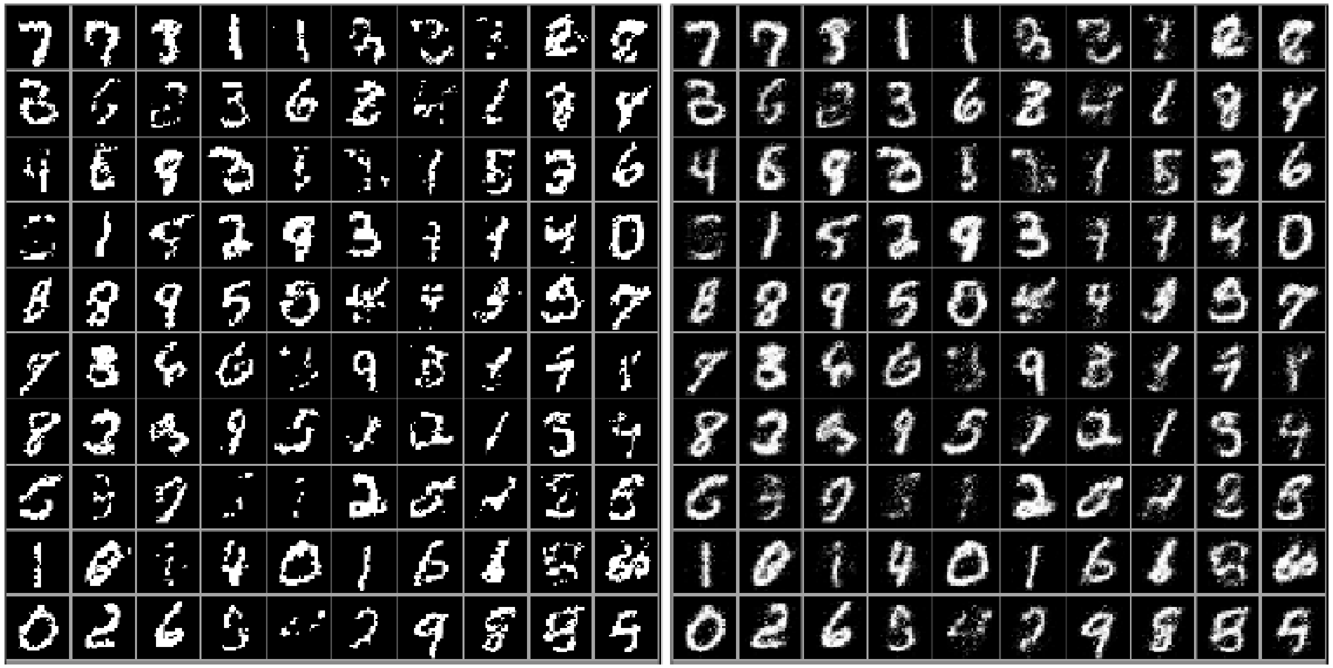
- O(n) parameters with parameter sharing

$$\mathbf{h}_2 = \sigma\left( \underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{W_{\cdot,<2}} x_1 + \mathbf{c} \right) \quad \mathbf{h}_3 = \sigma\left( \underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{W_{\cdot,<3}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) \quad \mathbf{h}_4 = \sigma\left( \underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{W_{\cdot,<4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right)$$



NADE

# NADE results



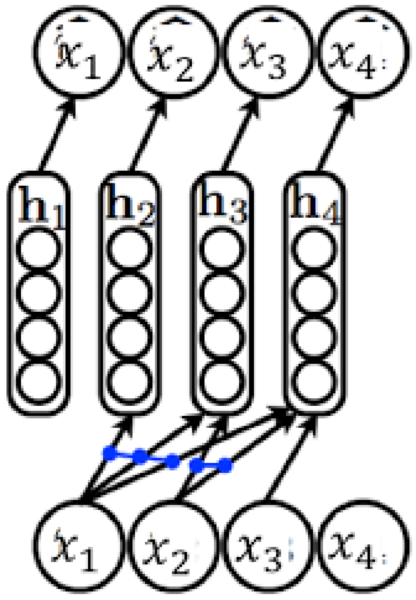
Samples from a model trained on MNIST on the left. Conditional probabilities $\hat{x}_i$ on the right.
Figure from *The Neural Autoregressive Distribution Estimator, 2011*

## General discrete distributions

- Model non-binary discrete random variables
  - Pixels from 0 – 255

- Imagine a categorical distribution

$$\begin{aligned} \mathbf{h}_i &= \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c}) \\ p(x_i|x_1,\cdots,x_{i-1}) &= Cat(p_i^1,\cdots,p_i^K) \\ \hat{\mathbf{x}}_i = (p_i^1,\cdots,p_i^K) &= softmax(A_i\mathbf{h}_i + \mathbf{b}_i) \end{aligned}$$



NADE

- Softmax operator

$$softmax(\mathbf{a}) = softmax(a^1,\cdots,a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)},\cdots,\frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$

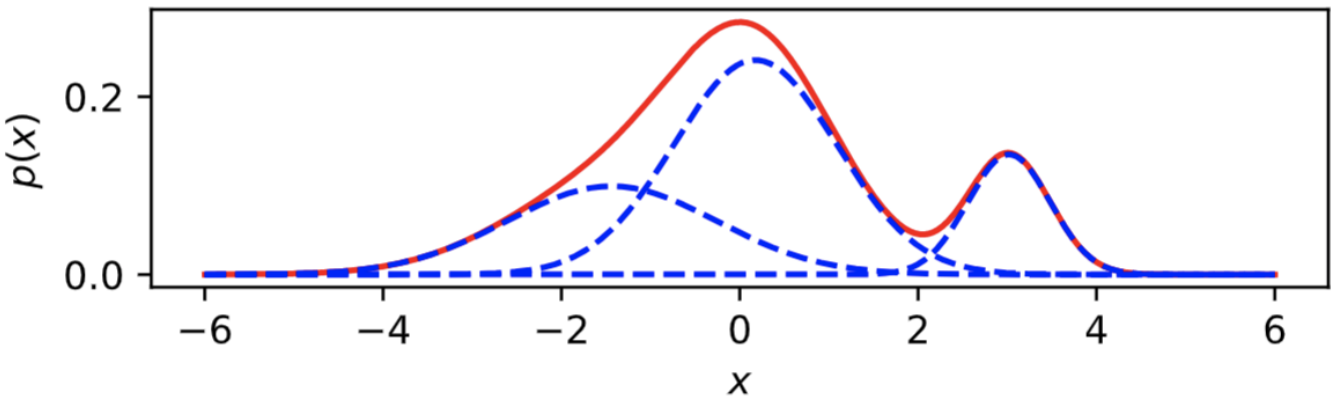
## RNADE

- Modeling continues random variables, e.g. speech signals by estimating the parameters of their distributions
  - For example, a mixture of k gaussians

$$p(x_i|x_1, \cdots, x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$

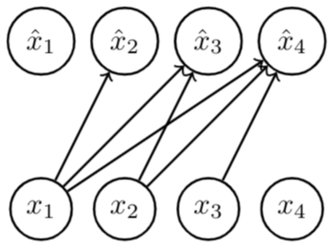
$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = (\mu_i^1, \cdots, \mu_i^K, \sigma_i^1, \cdots, \sigma_i^K) = f(\mathbf{h}_i)$$

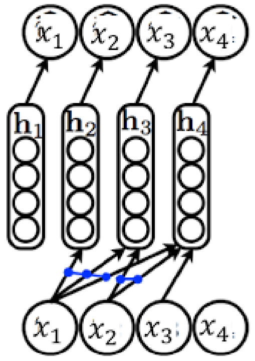


# Autoregressive models vs. autoencoders

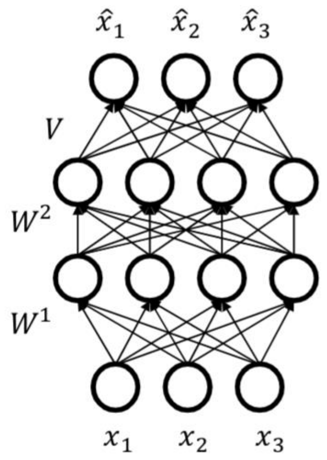
- It seems they are like to each other.



FVSBN


$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
$x_1$ $x_2$ $x_3$ $x_4$

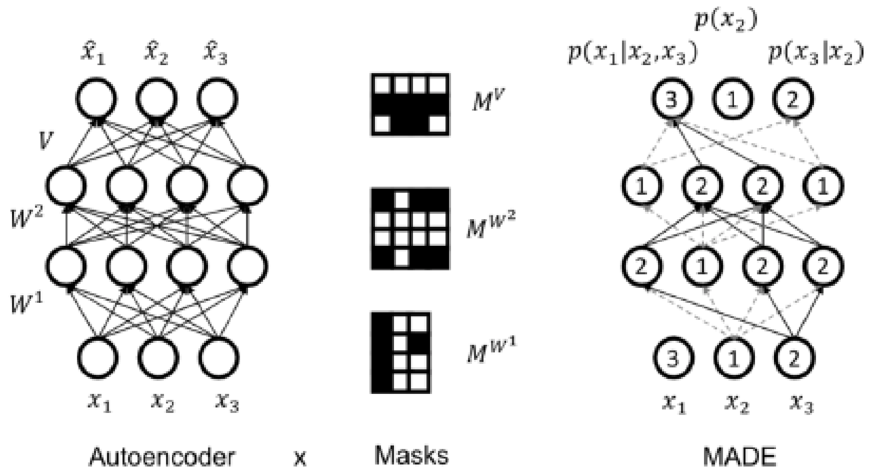
NADE



Autoencoder

- However, a vanilla autoencoder is not a generative model: it does not define a distribution over x we can sample from to generate new data points.

# MADE: Masked Autoencoder for Distribution Estimation



- Challenge: An autoencoder that is autoregressive

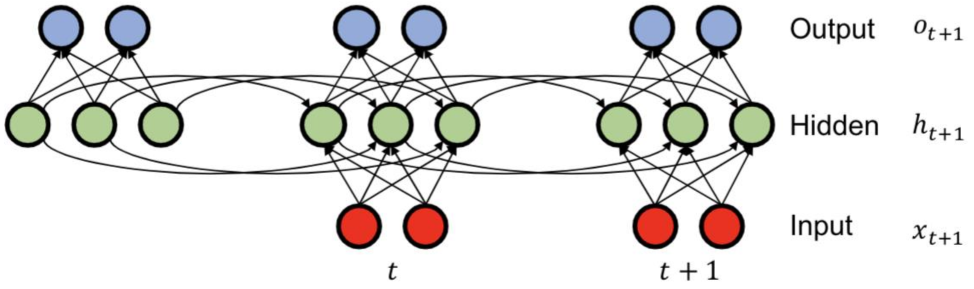
2. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is $x_2, x_3, x_1$, so $p(x_1, x_2, x_3) = p(x_2)p(x_3 \mid x_2)p(x_1 \mid x_2, x_3)$.
    1. The unit producing the parameters for $\hat{x}_2 = p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3|x_2)$ only on $x_2$. And so on...
    2. For each unit in a hidden layer, pick a random integer $i$ in $[1, n-1]$. That unit is allowed to depend only on the first $i$ inputs (according to the chosen ordering).
    3. Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer).

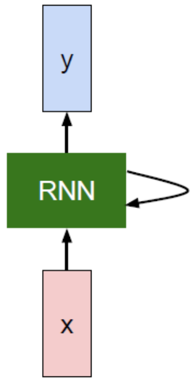
# RNN: Recurrent neural nets

- Main challenge of autoregressive models up to now:
  - History gets longer !!!

$$p(x_t | x_{1:t-1}; \alpha^t)$$

- RNN propose to keep a summary and recursively update it

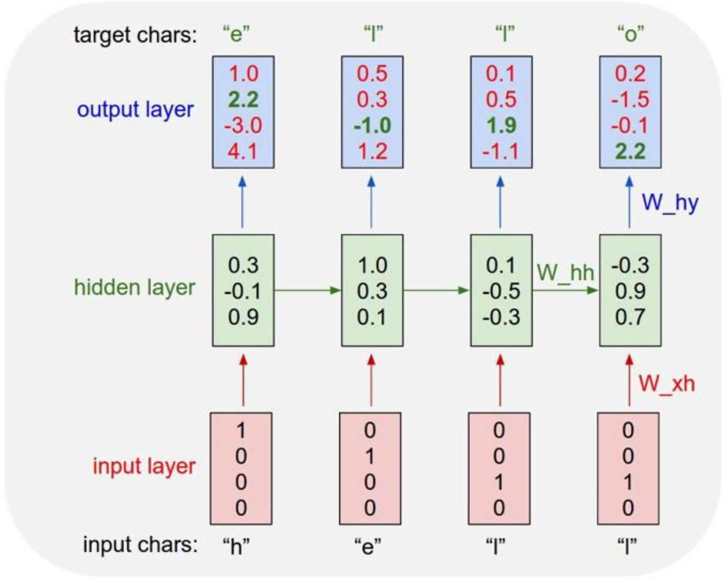




$$\begin{aligned} \text{Summary update rule:} \quad h_{t+1} &= tanh(W_{hh} h_t + W_{xh} x_{t+1}) \\ \text{Prediction:} \quad o_{t+1} &= W_{hy} h_{t+1} \\ \text{Summary initalization:} \quad h_0 &= \boldsymbol{b}_0 \end{aligned}$$

## RNN: Recurrent neural nets

- Example: imagine an alphabets with 4 letter
  - one-hot encoding

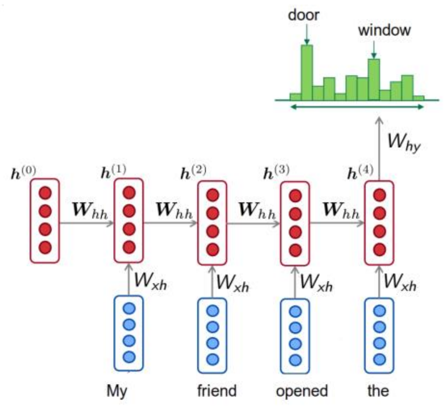


**Autoregressive**: $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$

$$\begin{aligned} p(x_2 = e|x_1 = h) &= softmax(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)} \\ o_1 &= W_{hy} h_1 \\ h_1 &= tanh(W_{hh} h_0 + W_{xh} x_1) \end{aligned}$$

# RNN: Recurrent neural nets

- Can be applied to sequences of arbitrary length, and are very general: For every computable function, there exists a finite RNN that can compute it.

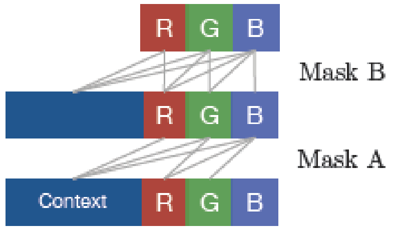


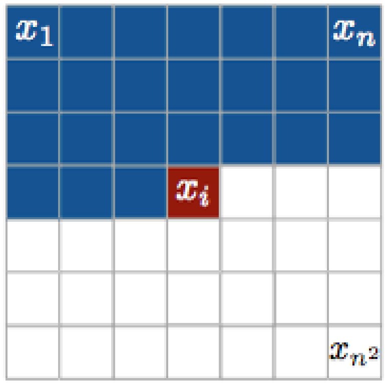
- Issues:
  - Requires an ordering.
  - A single hidden vector needs to summarize all the (growing) history.
  - They have sequential likelihood evaluation (very slow for training) and sequential generation (unavoidable in an autoregressive model) that can not be parallelized
  - Exploding/vanishing gradients when accessing information from many steps back

# PixelRNN

- Using RNN variants for generating images.
- We need an ordering assumption on pixels
  - An issue
- We also should consider an ordering for 3 channels: red, green and blue.

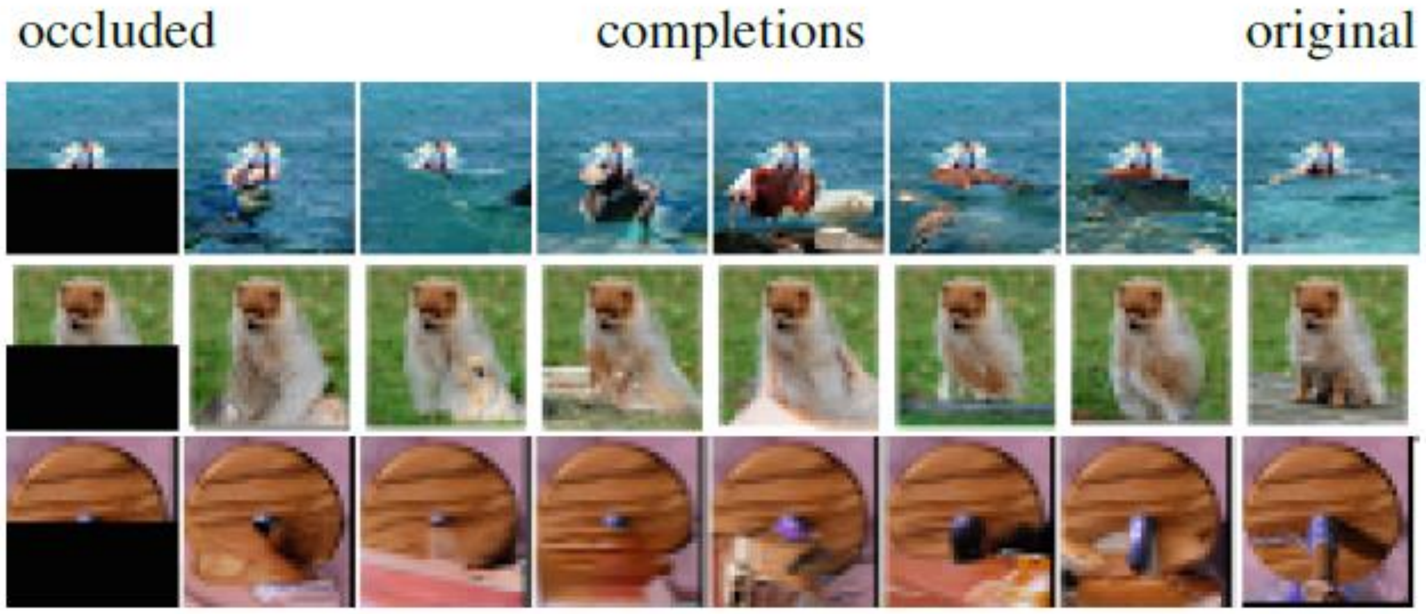
$$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$





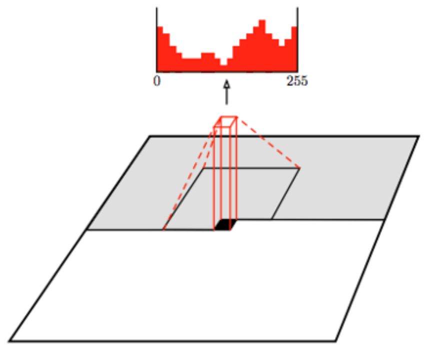
## PixelRNN

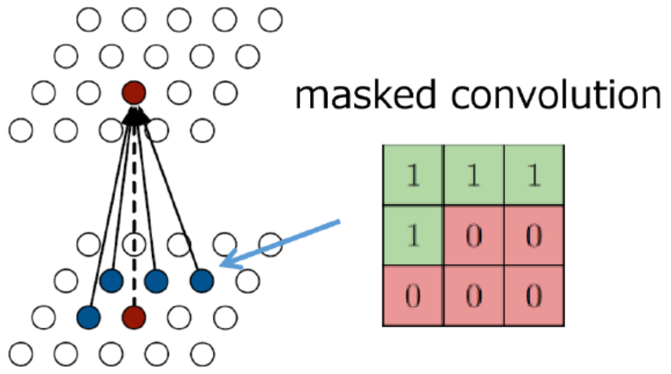
- Results on down sampled ImageNet.
- Very slow: sequential likelihood valuation.

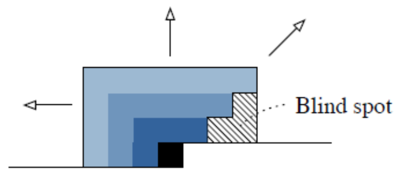


## PixelCNN

- Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
  - Has to be autoregressive.
  - Causal CNN
  - Masked convolutions preserve raster scan or
  - Additional masking for colors order.

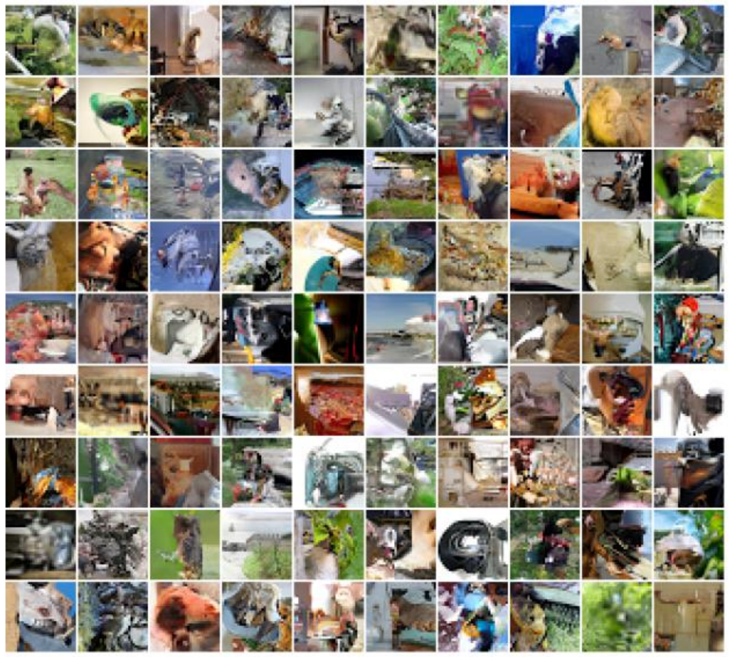






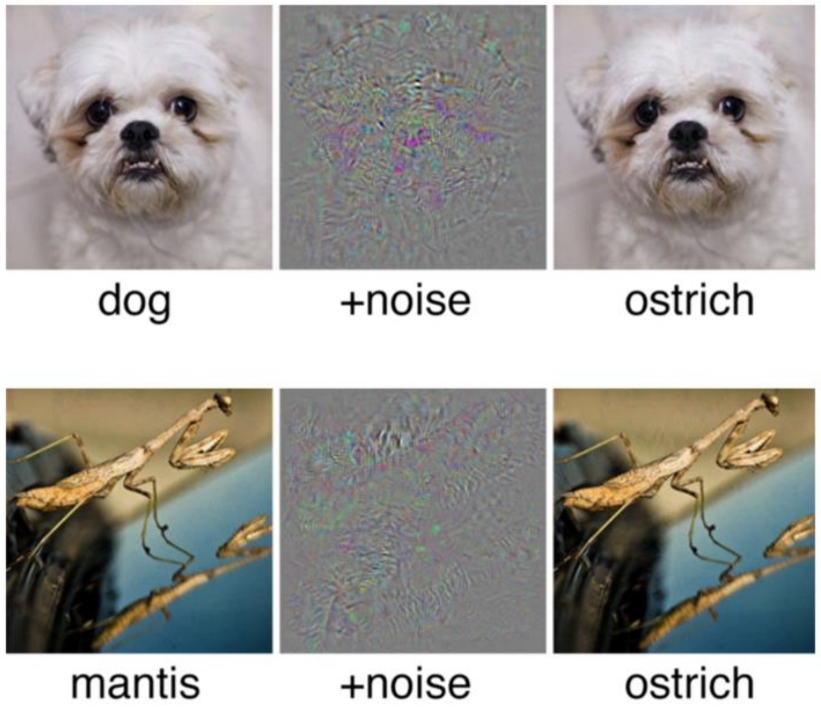
# PixelCNN

- Samples from the model trained on Imagenet (32 × 32 pixels).
- Similar performance to PixelRNN, but much faster.



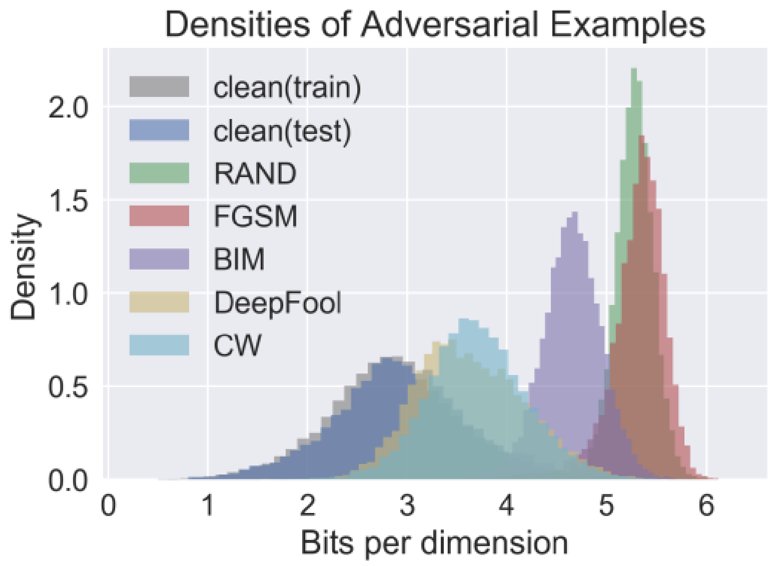
## Adversarial attack and anomaly detection

- Machine learning methods are vulnerable to adversarial examples.
- When a model can compute the likelihood function, we can use it for anomaly detection.
  - Corrupted images may have less likelihood



## PixelDefend

- Train a generative model $p(x)$ on clean inputs (PixelCNN)
- Given a new input $x$, evaluate $p(x)$
- Adversarial examples are significantly less likely under $p(x)$



## Attention mechanism

- Compare current hidden state (query) to all past hidden states (keys), e.g., by taking a dot product.
- Construct attention distribution

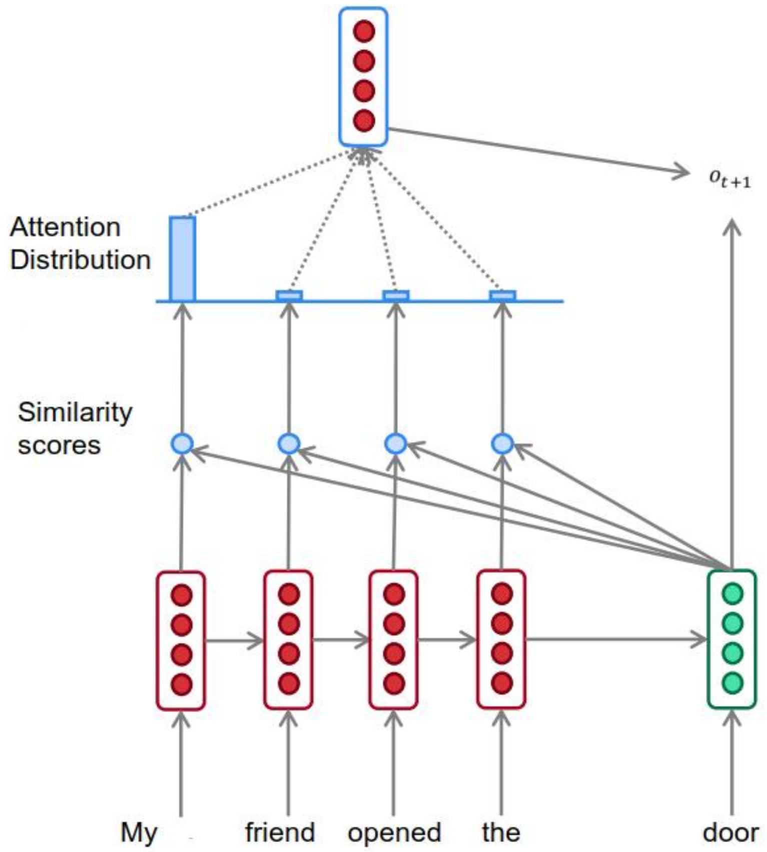
to figure out what parts of the history are relevant, e.g., via a Softmax.

- Construct a summary of the

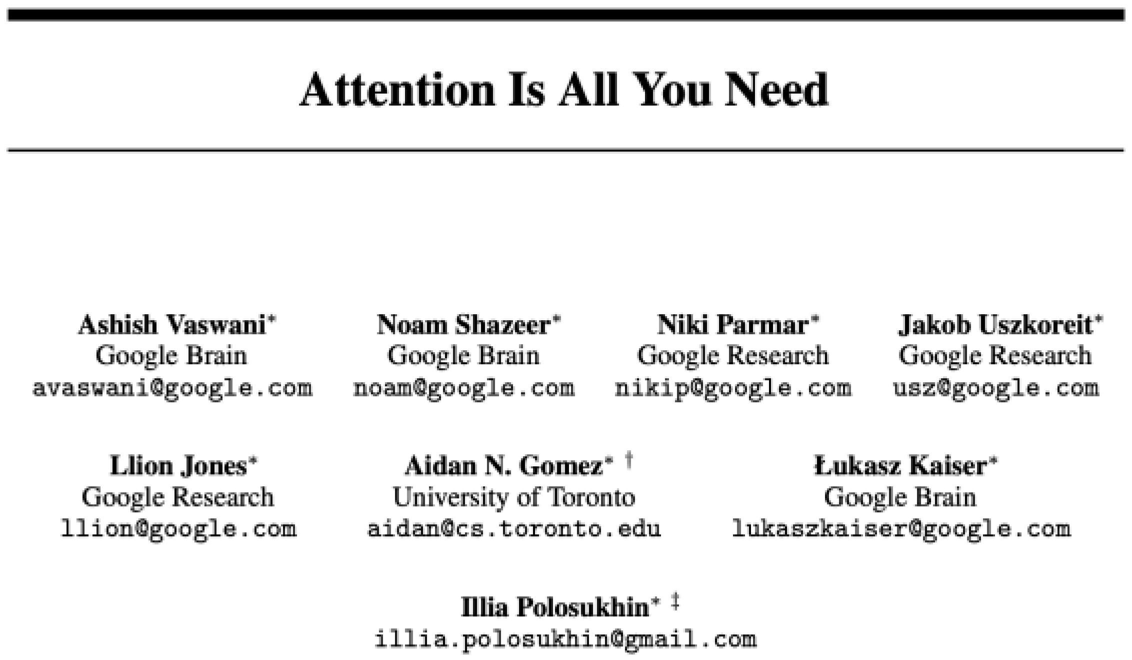
history, e.g., by weighted sum.

- Use summary and current hidden

state to predict next token/word.



## Transformer



**Attention Is All You Need**

**Ashish Vaswani***
Google Brain
avaswani@google.com

**Noam Shazeer***
Google Brain
noam@google.com

**Niki Parmar***
Google Research
nikip@google.com

**Jakob Uszkoreit***
Google Research
usz@google.com

**Llion Jones***
Google Research
llion@google.com

**Aidan N. Gomez*** †
University of Toronto
aidan@cs.toronto.edu

**Łukasz Kaiser***
Google Brain
lukaszkaiser@google.com

**Illia Polosukhin*** ‡
illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

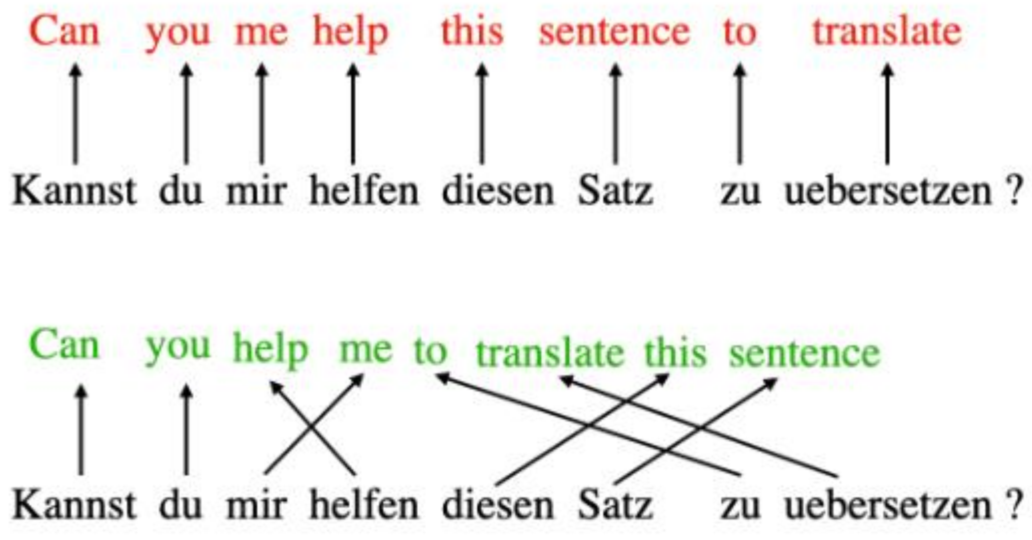
https://arxiv.org/abs/1706.03762

## Transformer

- Current state of the art (GPTs): replace RNN with Transformer.
- Avoid recursive computation. Use only self-attention to enable parallelization.
- Needs masked self-attention to preserve autoregressive structure.
- Demo:
  - https://transformer.huggingface.co/doc/gpt2-large
  - https://huggingface.co/spaces/huggingface-projects/llama-2-13b-chat

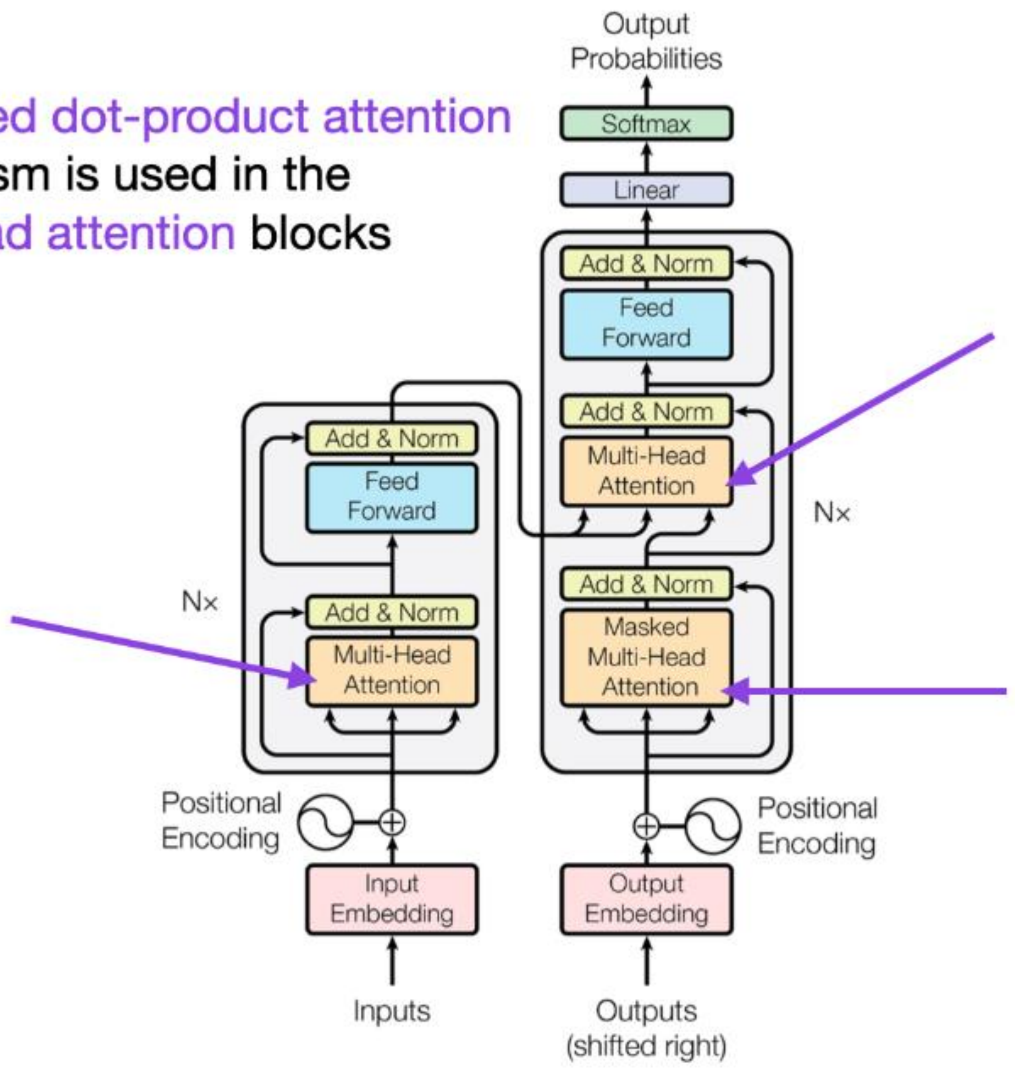
## Transformer

- Attention mechanisms were introduced to give access to all sequence elements at each time step and adaptively focus only on relevant context
  - Can handle longer sequences compared to RNNs.


Can you me help this sentence to translate
Kannst du mir helfen diesen Satz zu uebersetzen ?
Can you help me to translate this sentence
Kannst du mir helfen diesen Satz zu uebersetzen ?

# Transformer

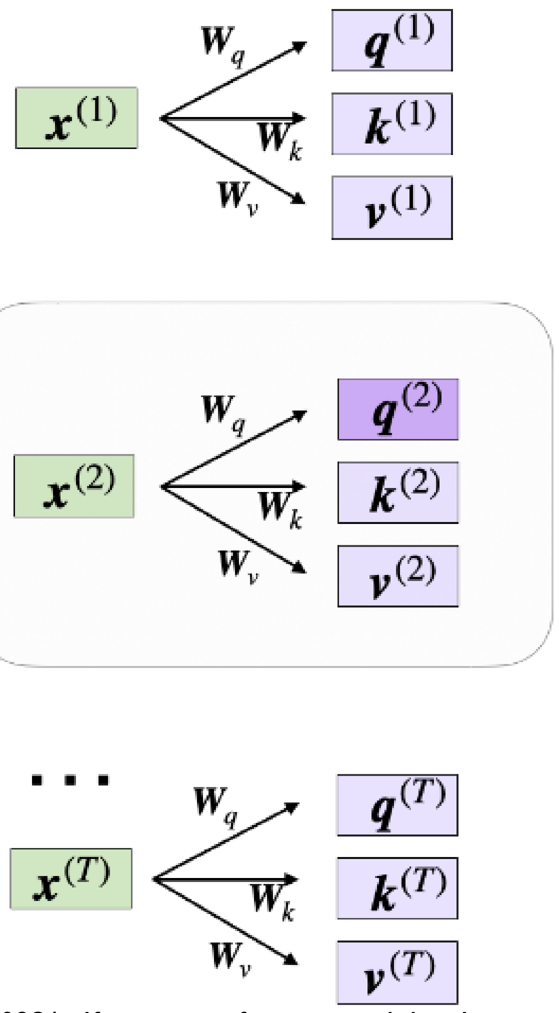
The scaled dot-product attention mechanism is used in the multi-head attention blocks



# Self-attention in transformers

- In each timestep, we compose 3 vectors from the input
  - Query
  - Key
  - Value

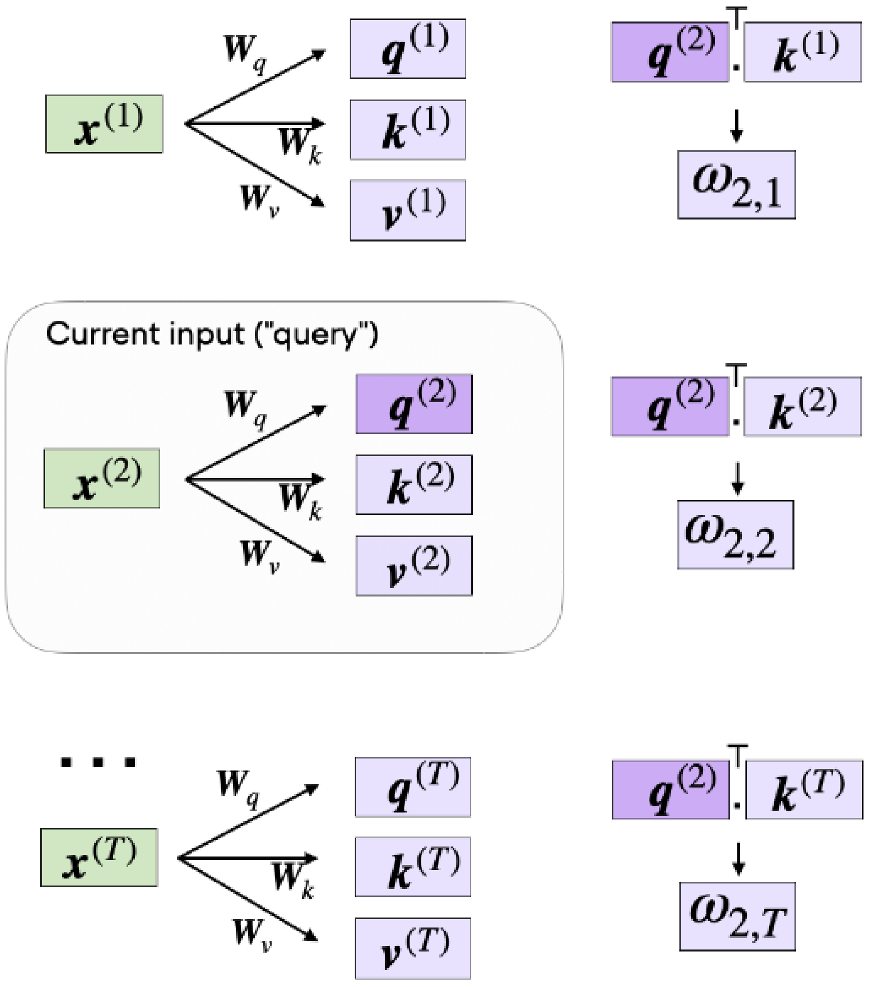
- These vectors are obtained with shared parameters $W_q, W_k, W_v$



https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

# Self-attention in transformers

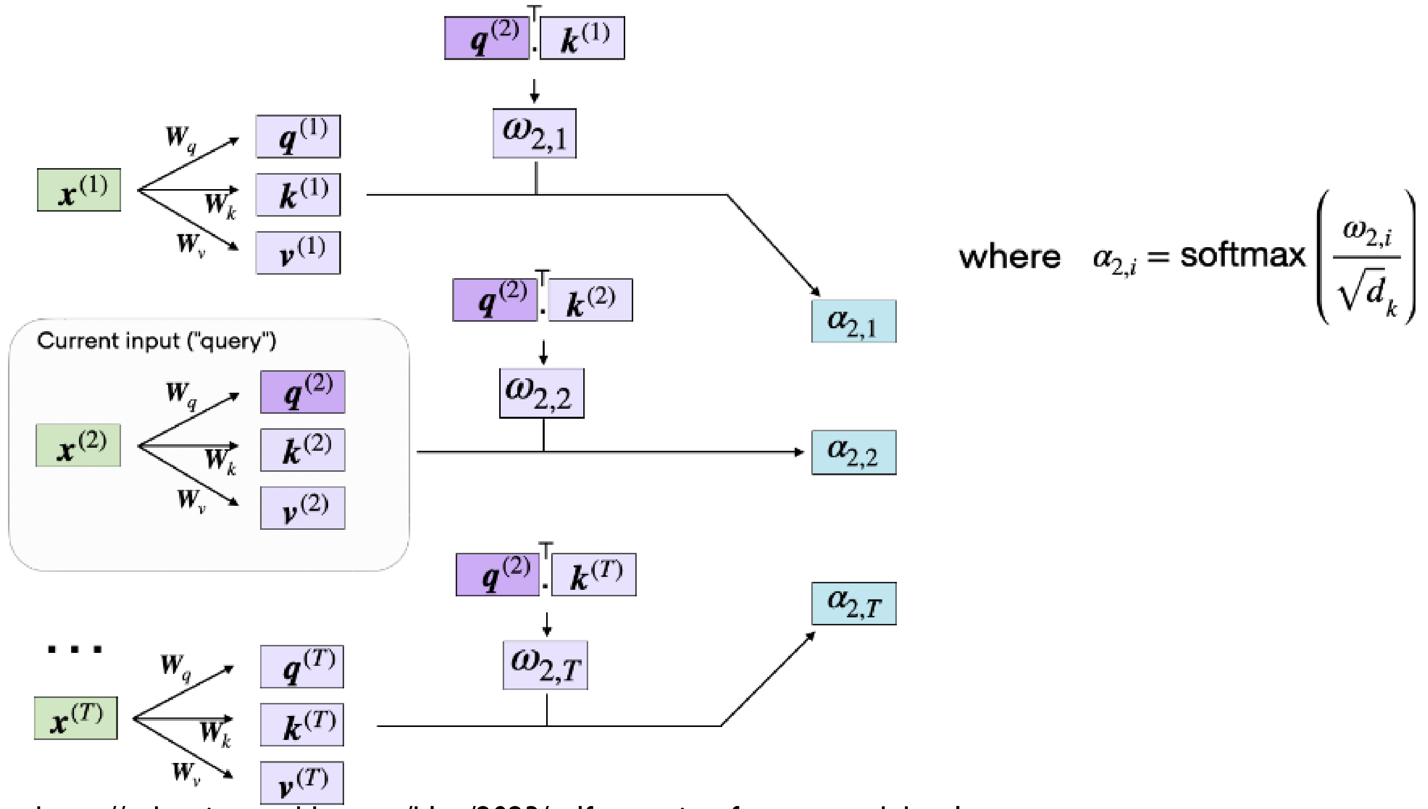
- Now, we compute unnormalized attention weights for each time step.



https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

# Self-attention in transformers

- The subsequent step is to normalize the unnormalized attention weights ω, to obtain the normalized attention weights $\alpha$
  - Scaling by $d_k$, dimension of key vectors, avoids the attention weighs to become to large or too small

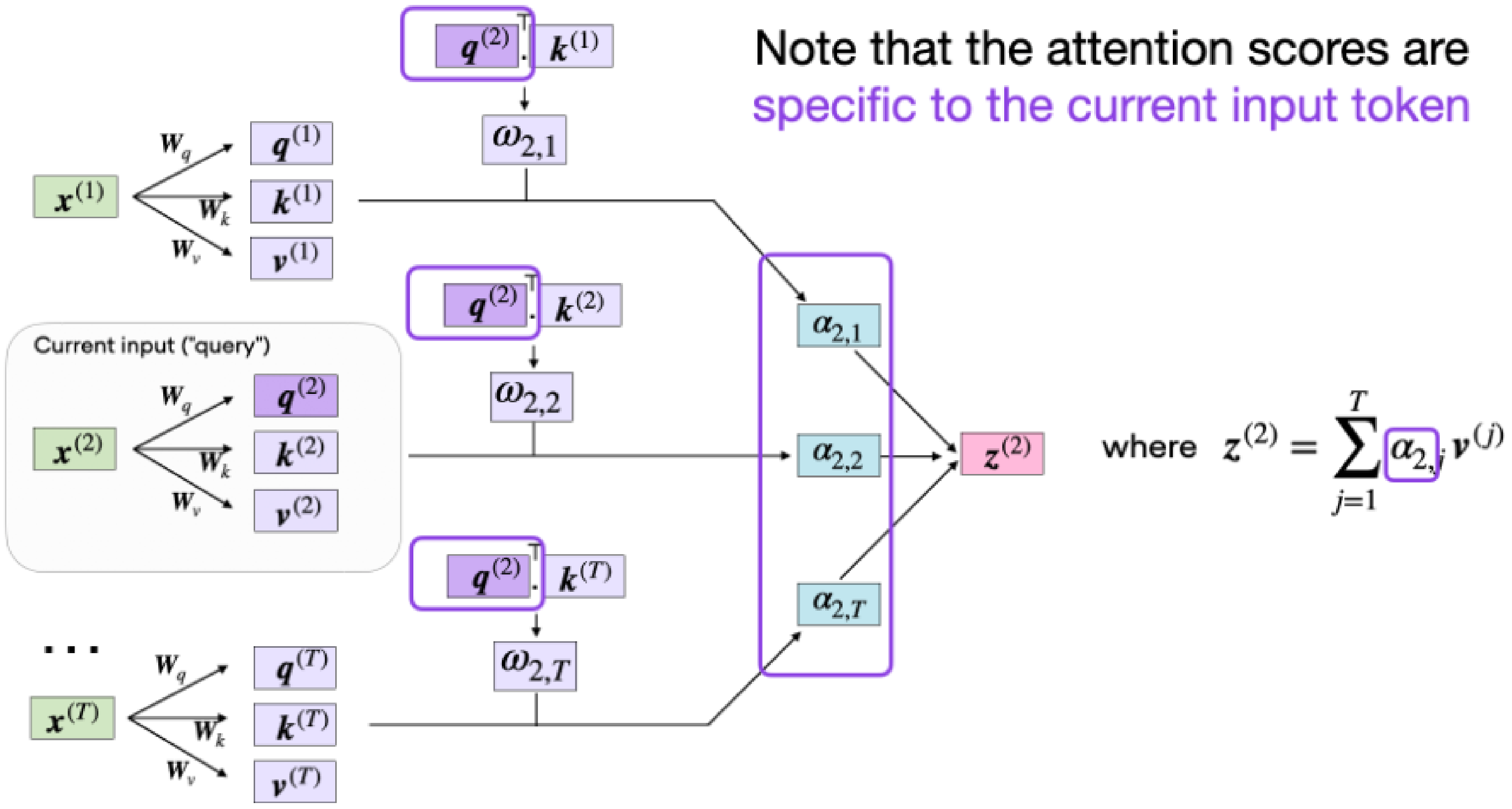


where $\alpha_{2,i} = \text{softmax}\left(\frac{\omega_{2,i}}{\sqrt{d_k}}\right)$

https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

## Self-attention in transformers

- In the last step, the context vector $z_i$ is obtained as an attention-weighted version of the input $x_i$

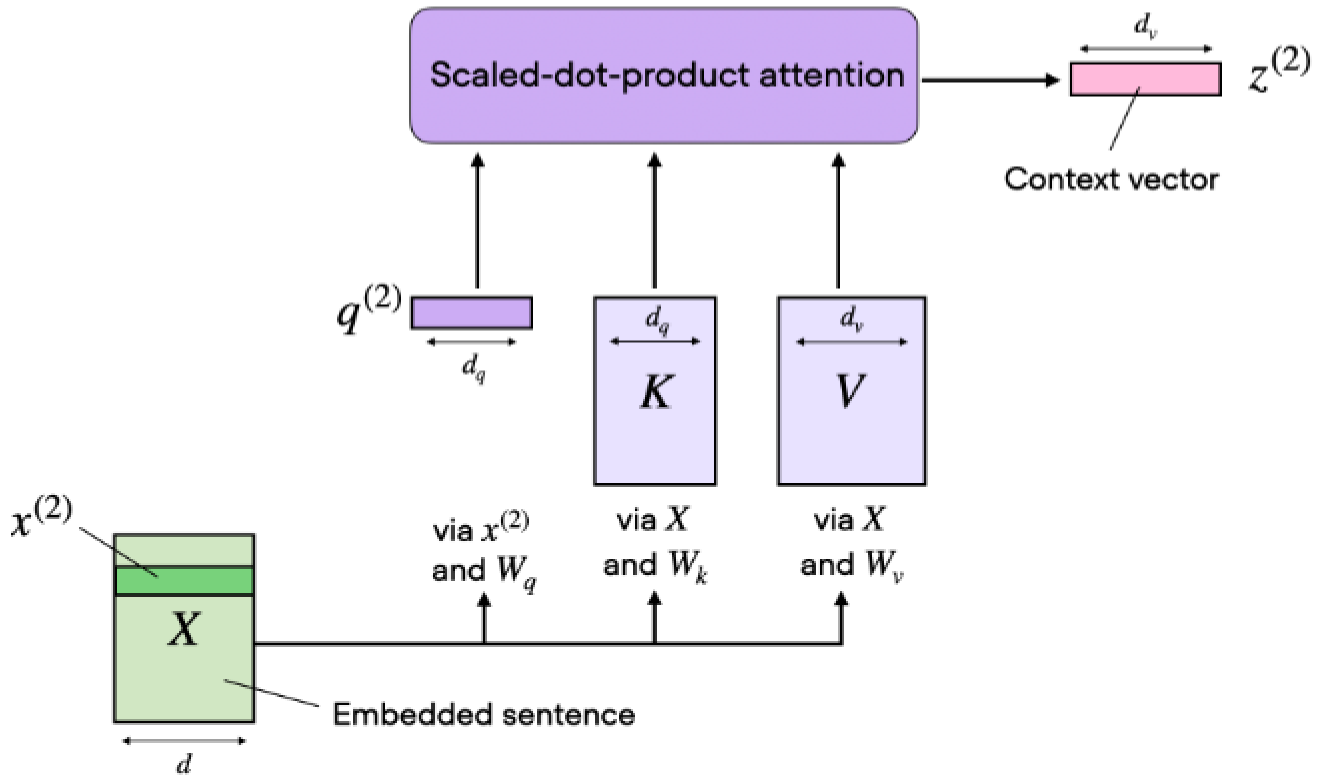


Note that the attention scores are specific to the current input token

where $z^{(2)} = \sum_{j=1}^{T} \boxed{\alpha_{2,j}} v^{(j)}$

## Multi-head attention

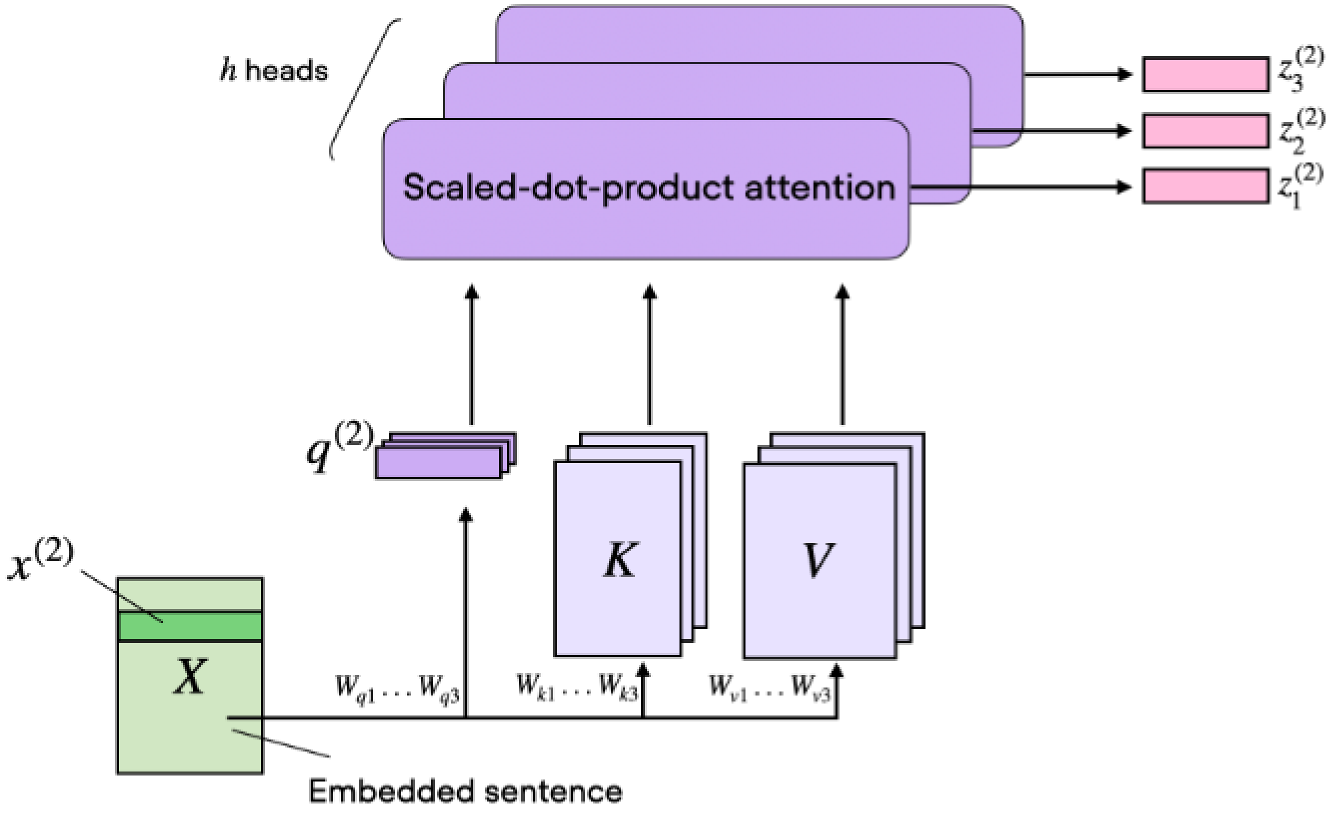
- A set of three matrices query, key, and value are considered as a single attention head in the context of **multi-head** attention.



https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

## Multi-head attention

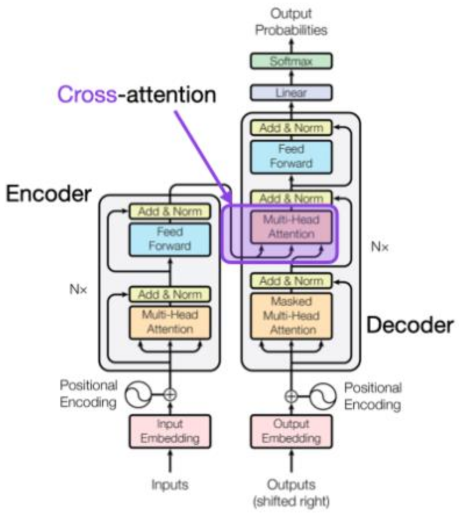
- The concept of multi-head attention is similar to using multiple kernels in CNNs.

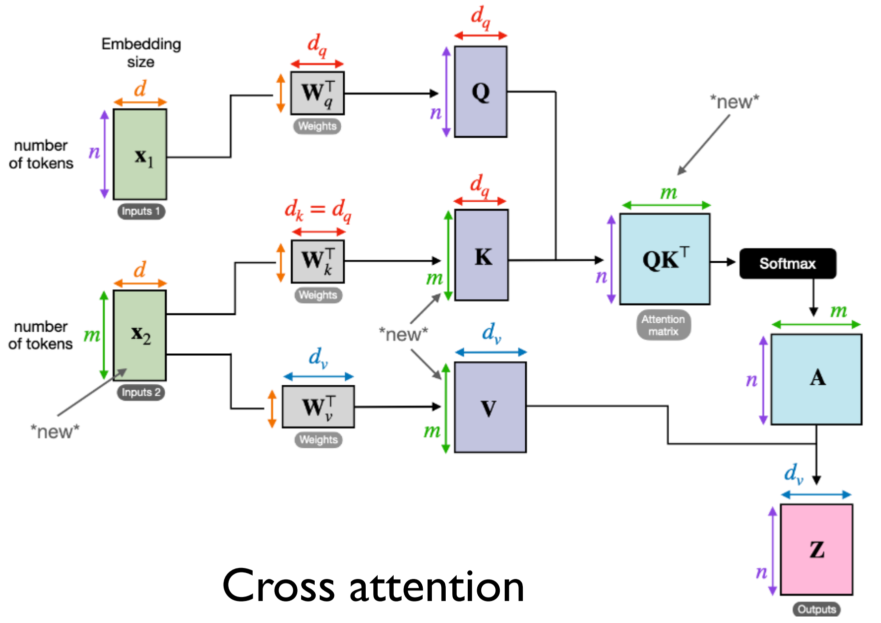


https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

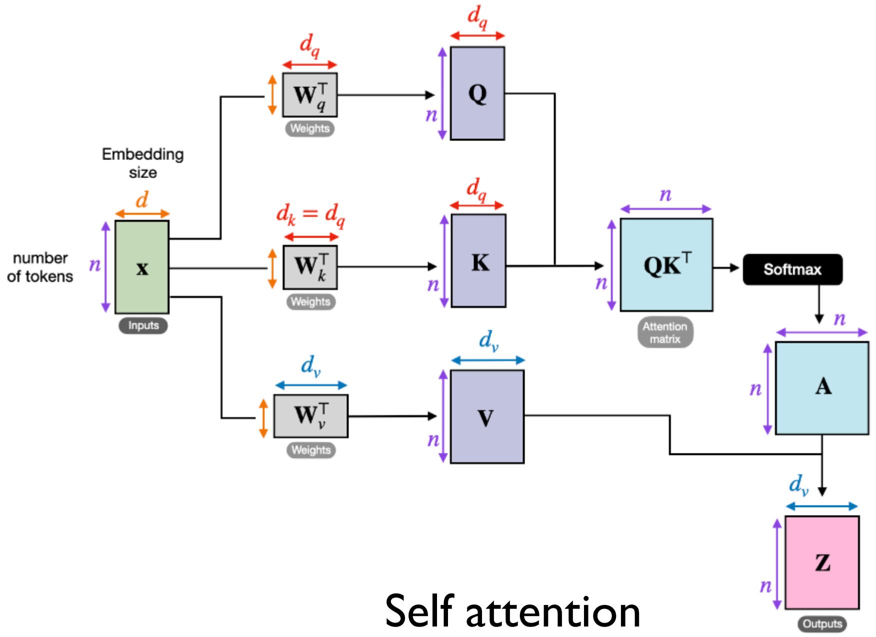
# Cross attention vs. self attention

- Mixing two input sequences
  - Can have different lengths
  - In encoder-decoder transformer, cross attention is used to relate the encoder output to decoder.





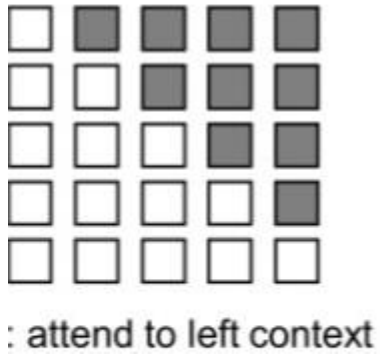
Cross attention

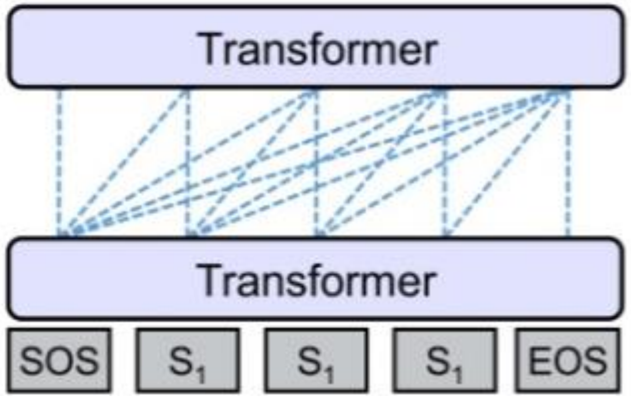


Self attention

## Masked attention

- In the decoder block, the self-attention mechanism is masked.
  - The causal self-attention
- To impose a causal structure to the model, the attention mechanism can only consider previous elements of the sequence.
  - Blocking paths to the future!



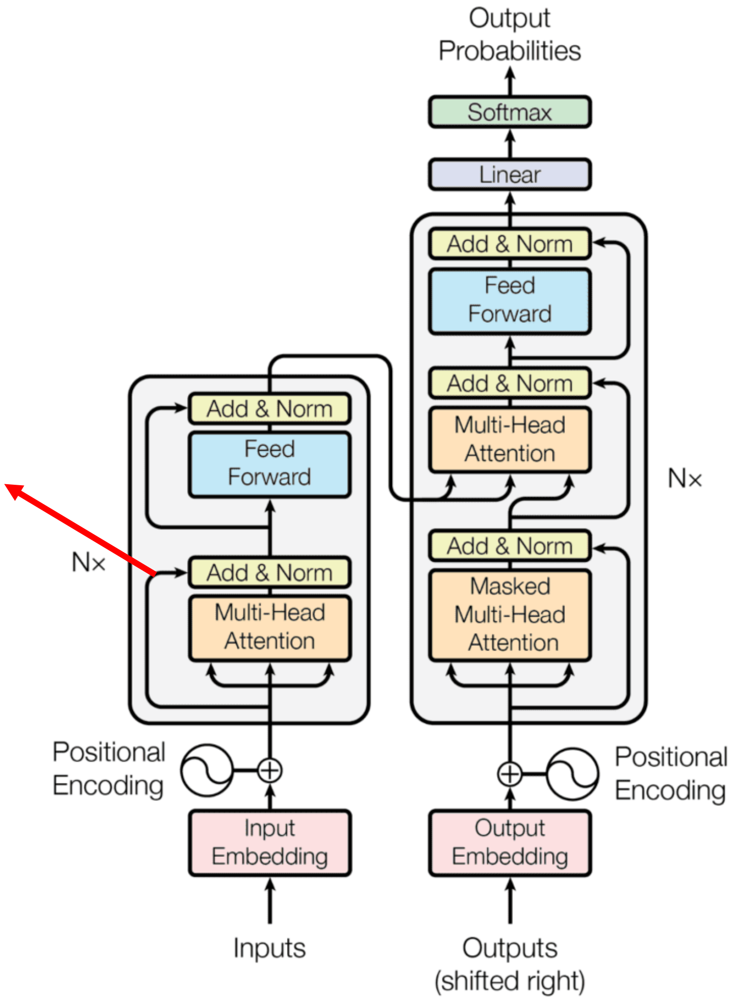


## Transformer

- Residual connection
- Layer normalization

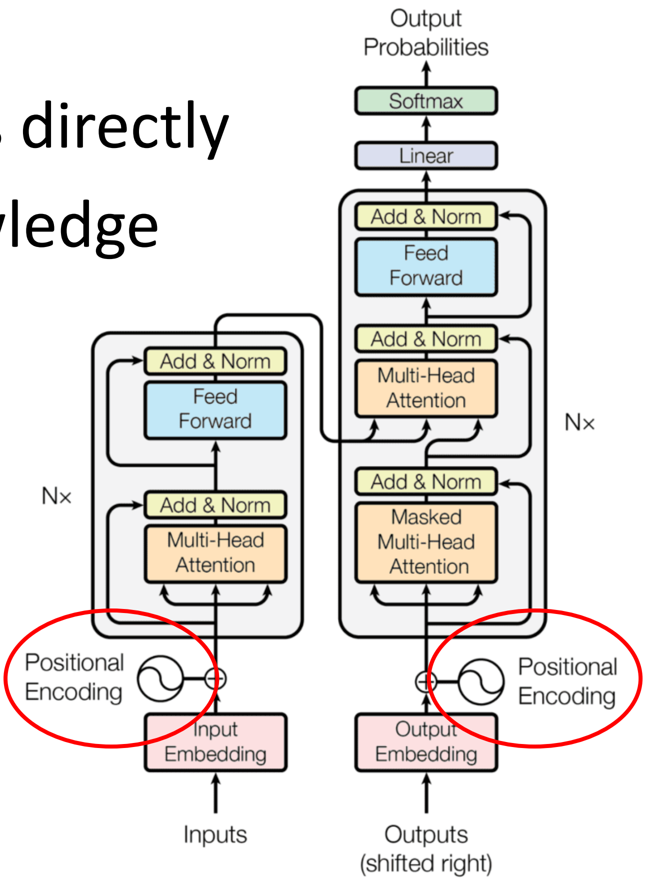
$$\mathrm{LayerNorm}(\mathbf{z};\gamma,\beta) = \gamma\frac{(\mathbf{z}-\mu_{\mathbf{z}})}{\sigma_{\mathbf{z}}} + \beta,$$

$$\mu_{\mathbf{z}} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{z}_i, \quad \sigma_{\mathbf{z}} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(\mathbf{z}_i-\mu_{\mathbf{z}})^2}.$$



## Positional encoding in transformers

- Up to now, we have considered the input as a bag of words.
- The transformer model does not have an inbuilt recurrent architecture like RNNs.
- Therefore, the positional information is directly added to the input to impose the knowledge of the order of objects in a sequence.
  - Learning an embedding
  - Using an embedding function



## Positional encoding in transformers

- Different ways:
  - One-hot encoding of the position in vector $e$

    $x \in R^{T\times d}, e \in R^{T} \quad \rightarrow \quad x' = (x, e), \qquad x' \in R^{T\times(T+d)}$

  - Learning a combined representation

    $x \in R^{T\times d}, e \in R^{T} \quad \rightarrow \quad x' = W^{T} ReLU(W_x^T x + W_e^T e)$

  - Building distinct representations of inputs and positions

    $x \in R^{T\times d}, e \in R^{T} \rightarrow x' = W_1^T ReLU(W_x^T x) + W_2^T ReLU(W_e^T e)$

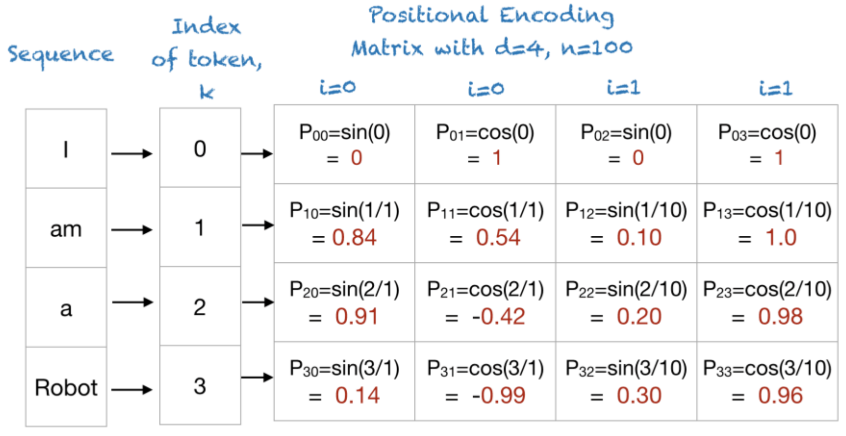
  - The original transformer paper proposes a fixed representation of positions $p$; using sin and cos functions

    $x \in R^{T\times d} \quad \rightarrow \quad x' = W_1^T ReLU(W_x^T x) + p$

# Positional encoding in transformers

- Sin and cos functions for fixed positional embedding:

  - $p(k, 2i) = \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right)$
  - $p(k, 2i+1) = \cos\left(\frac{k}{n^{\frac{2i}{d}}}\right)$
  - $k$: position an element
  - $d$: dimension of the embedding space
  - $n$: user defined scalar (10000 in the main paper)
  - $i$: used for mapping to column indices; $0 \leq i \leq \frac{d}{2}$



Positional Encoding Matrix with d=4, n=100

| Sequence | Index of token, k | i=0 | i=0 | i=1 | i=1 |
|---|---|---|---|---|---|
| I | 0 | $P_{00}$=sin(0) = 0 | $P_{01}$=cos(0) = 1 | $P_{02}$=sin(0) = 0 | $P_{03}$=cos(0) = 1 |
| am | 1 | $P_{10}$=sin(1/1) = 0.84 | $P_{11}$=cos(1/1) = 0.54 | $P_{12}$=sin(1/10) = 0.10 | $P_{13}$=cos(1/10) = 1.0 |
| a | 2 | $P_{20}$=sin(2/1) = 0.91 | $P_{21}$=cos(2/1) = -0.42 | $P_{22}$=sin(2/10) = 0.20 | $P_{23}$=cos(2/10) = 0.98 |
| Robot | 3 | $P_{30}$=sin(3/1) = 0.14 | $P_{31}$=cos(3/1) = -0.99 | $P_{32}$=sin(3/10) = 0.30 | $P_{33}$=cos(3/10) = 0.96 |

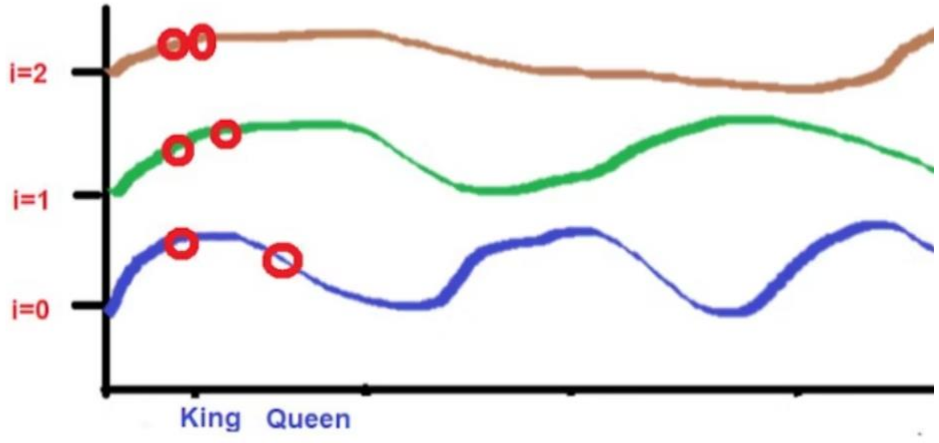
Positional Encoding Matrix for the sequence 'I am a robot'

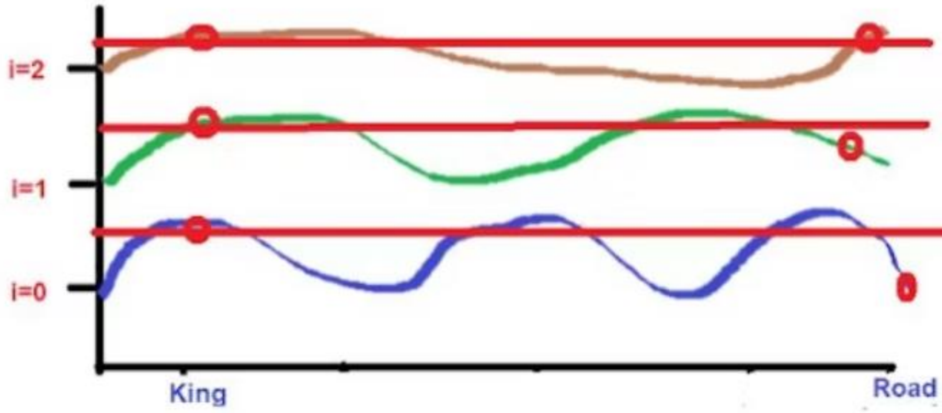
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

## Positional encoding in transformers

- Sin and cos functions for fixed positional embedding:
  - Different frequencies in this embedding helps the model to consider long and short term dependencies in the sequence.

Consider the sentence - ***"King and Queen are walking on the road."***





## The transformer function

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{p\times d} \to \mathbb{R}^{p\times d}$. If $\mathbf{x} \in \mathbb{R}^{p\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left( \frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}} \right), \tag{2}$$

$$\mathbf{u}'_i = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

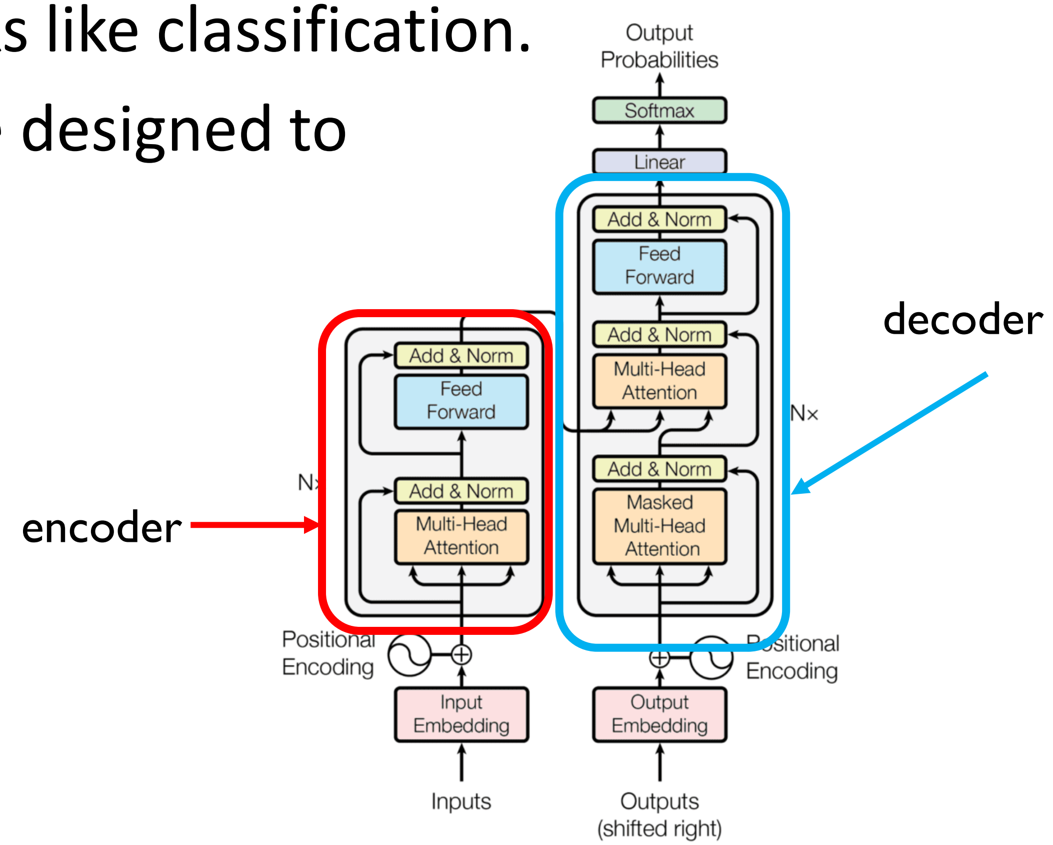
$$\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}'_i; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}, \tag{4}$$

$$\mathbf{z}'_i = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}'_i; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}. \tag{6}$$

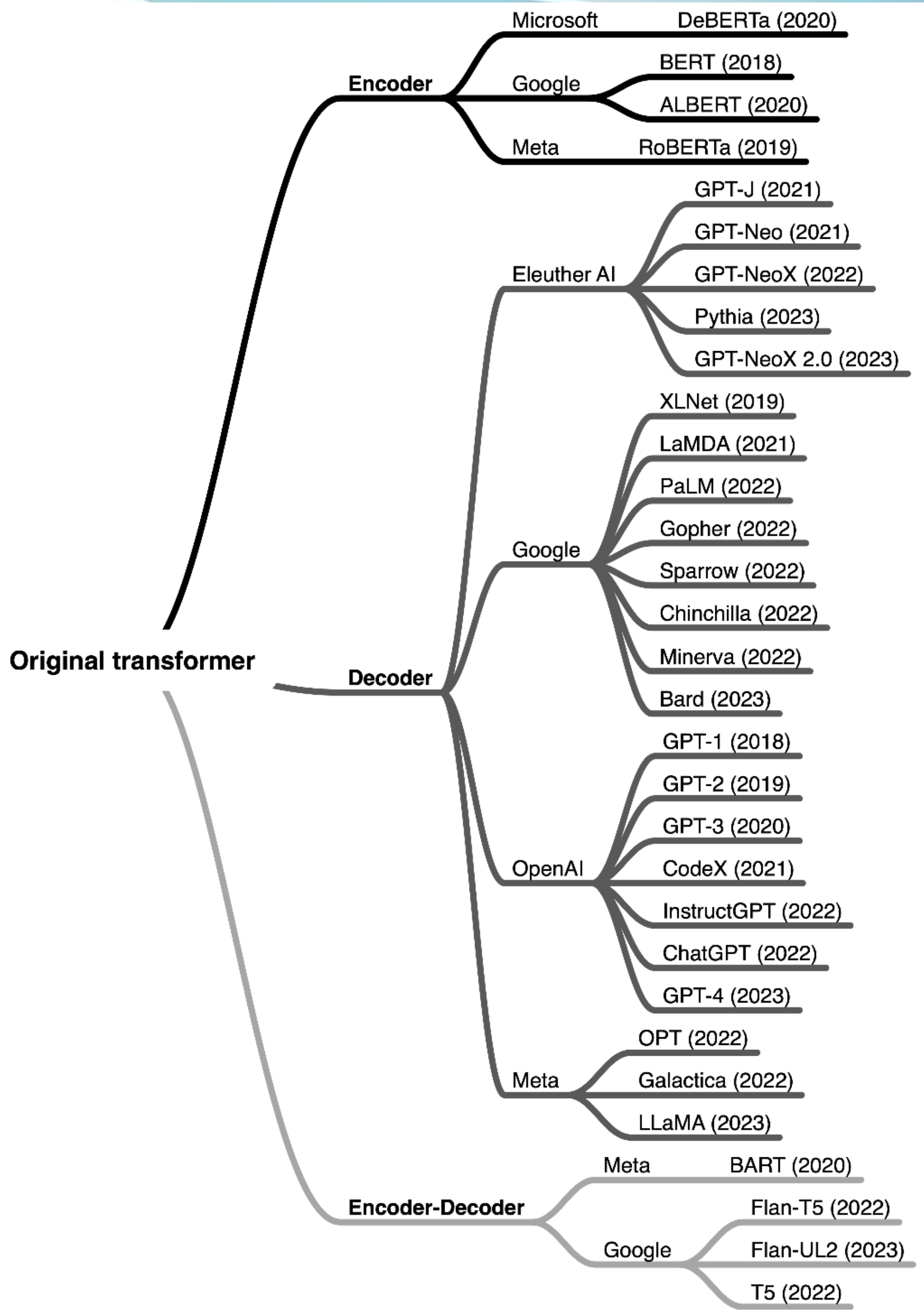
# Transformer based LLMs

- Both encoder and decoder style transformer use the same self-attention layers to encode tokens.
- Encoder is designed to learn embedding for predictive modeling tasks like classification.
- In contrast, decoders are designed to generate new texts, ex. answering user queries.



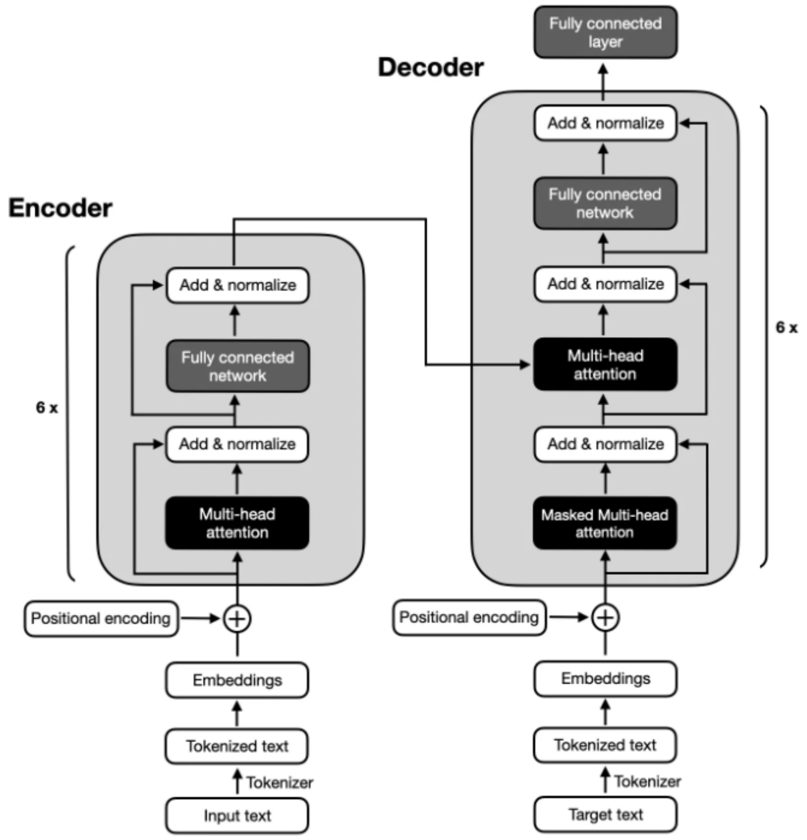
# Transformer based LLMs

Different developed LLMs:



## Encoder-decoder transformer example

- Main transformer paper
  - Developed for the translation task

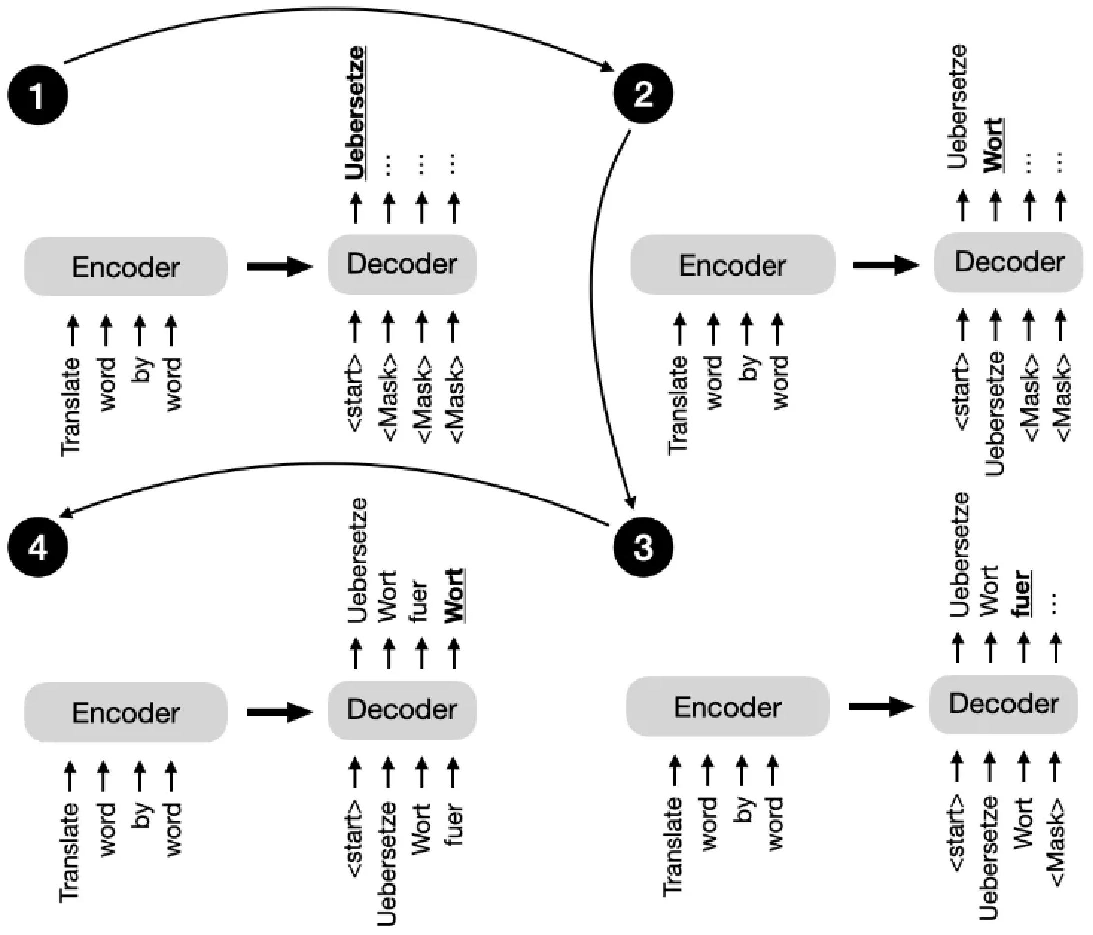


## Encoder-decoder transformer example

- Encoder-decoder models are typically used for natural language processing tasks that involve understanding input sequences and generating output sequences, often with different lengths and structures.
- They are particularly good at tasks where there is a complex mapping between the input and output sequences and where it is crucial to capture the relationships between the elements in both sequences.
- Some common use cases for encoder-decoder models include text translation and summarization.

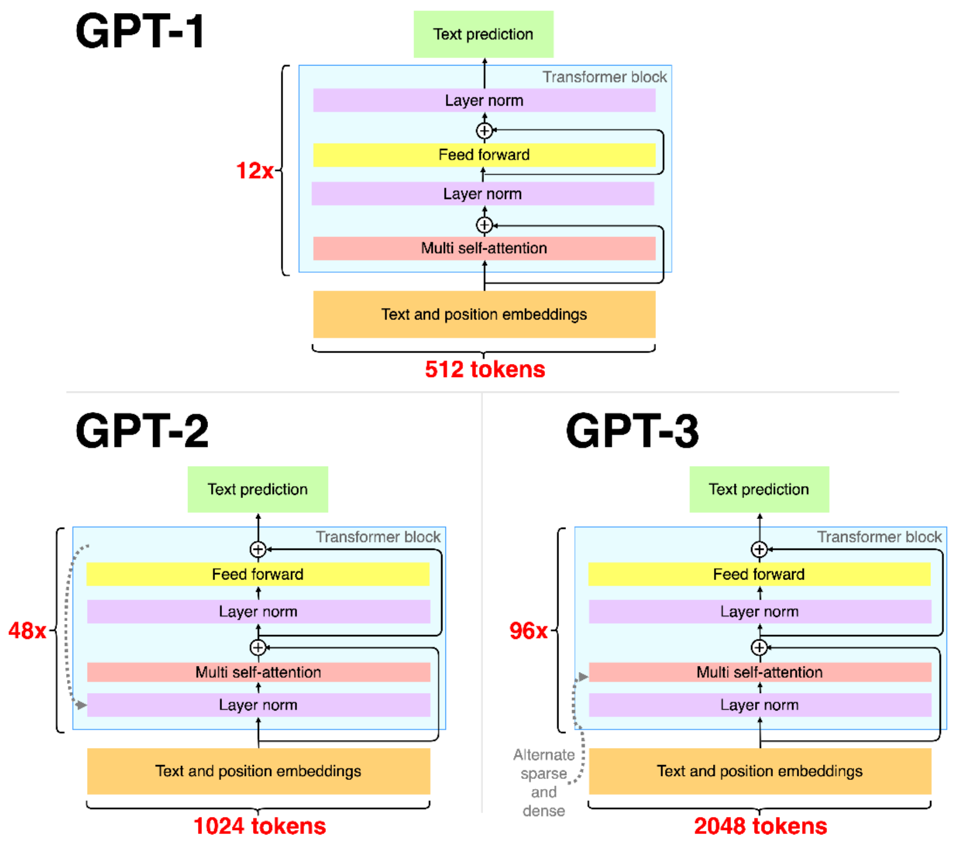
## Encoder-decoder transformer example

- Main transformer paper



# Decoder only transformer example

- GPT: **G**enerative **P**retrained **T**ransformer



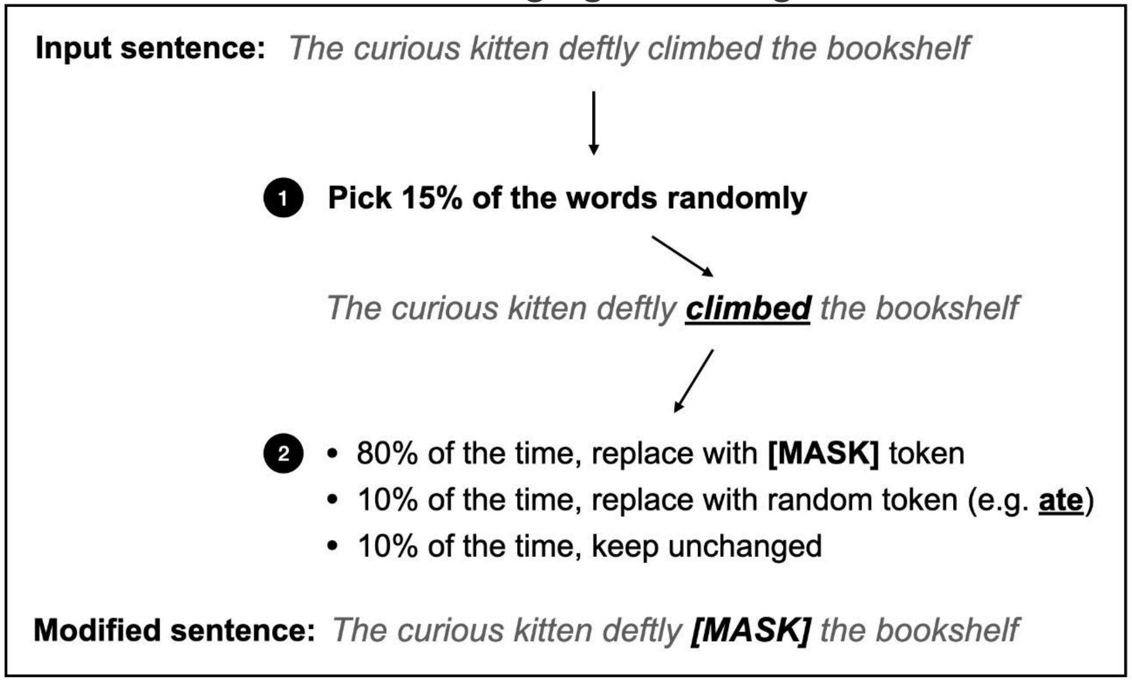
## Decoder only transformer example

- GPT: **G**enerative **P**retrained **T**ransformer
- One of the most notable aspects of GPT models is their emergent properties. Even though these models were only taught to predict the next word, the pretrained models are capable of text summarization, translation, question answering, classification, and more.

## Encoder only transformer example

- BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- Pretrained on a large text corpus using
  - Masked language modeling
  - Next-sentence prediction

Masked language modeling



**Input sentence:** *The curious kitten deftly climbed the bookshelf*

❶ **Pick 15% of the words randomly**

*The curious kitten deftly* ***climbed*** *the bookshelf*

❷
- 80% of the time, replace with **[MASK]** token
- 10% of the time, replace with random token (e.g. **ate**)
- 10% of the time, keep unchanged

**Modified sentence:** *The curious kitten deftly* ***[MASK]*** *the bookshelf*

# Encoder only transformer example

- Next-sentence prediction
  - Asks the model to predict whether the original document's sentence order of two randomly shuffled sentences is correct.

[CLS] Toast is a simple yet delicious food [SEP] It's often served with butter, jam, or honey.
[CLS] It's often served with butter, jam, or honey. [SEP] Toast is a simple yet delicious food.

- The [CLS] token is a placeholder token for the model, prompting the model to return a *True* or *False* label indicating whether the sentences are in the correct order or not.
- These two loss functions allow BERT to learn rich contextual representations of the input texts, which can then be finetuned for various downstream tasks like sentiment analysis, question-answering, and named entity recognition.