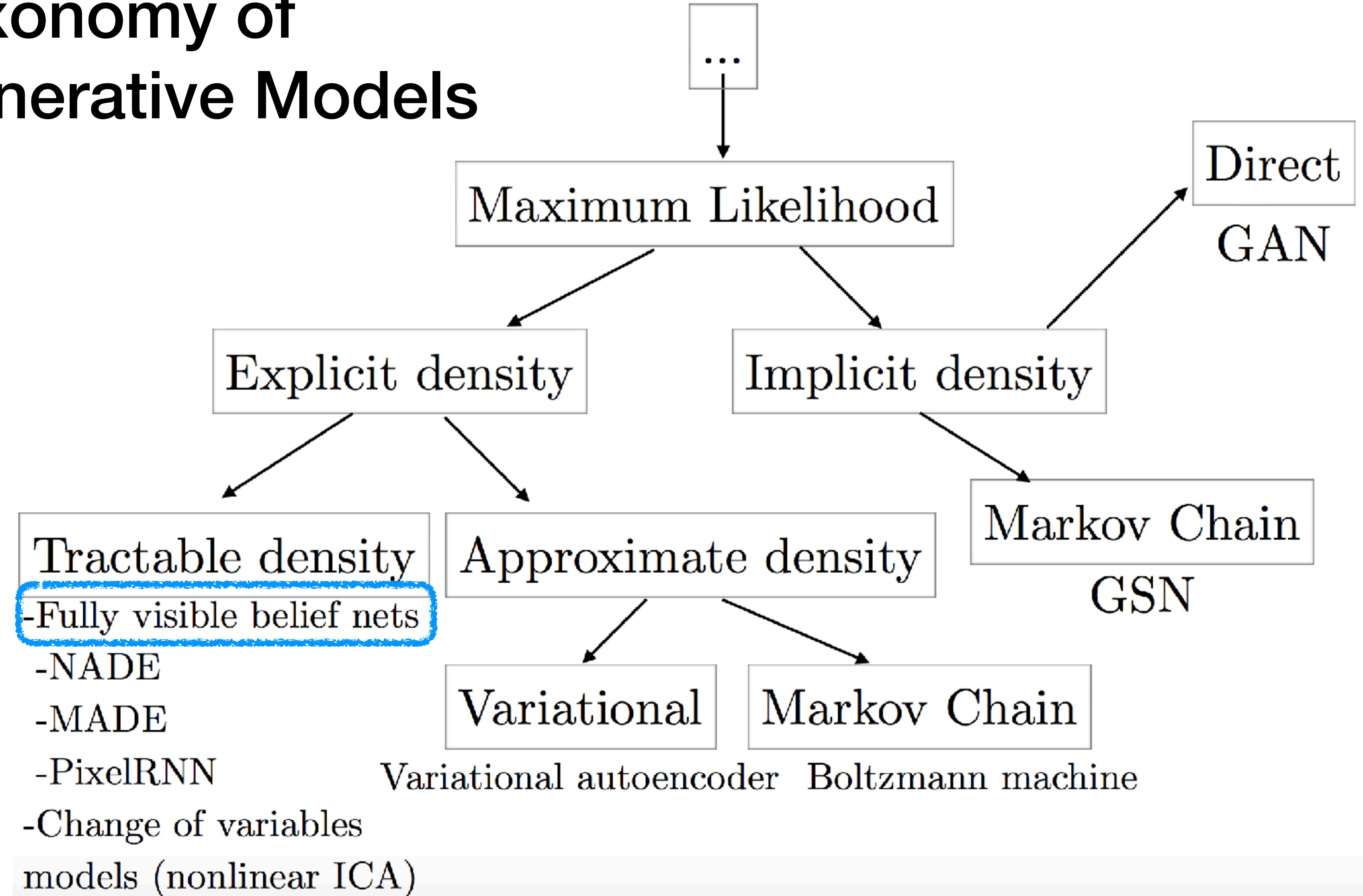


AutoRegressive Models

2021
Multicampus

Il Gu Yi
ModuLabs, Research Scientist
Soochul Park
Vivestudios, Research Scientist

Taxonomy of Generative Models



Autoregressive Models

- Autoregressive model은 하나의 data를 scalar값들의 sequence로 바라보고, 순차적으로 생성합니다.
- 하나의 data를 \mathbf{x} 라 두고, \mathbf{x} 가 scalar값의 집합 $\{x_1, x_2, \dots, x_D\}$ 으로 이루어져 있다고 할 때, \mathbf{x} 의 likelihood는 다음과 같이 표현됩니다.

$$p_{\theta}(\mathbf{x}) = \prod_{i=1}^D p_{\theta}(x_i | x_{<i})$$

- 즉, 현재시점의 scalar값 x_i 의 확률분포는 과거시점의 scalar값들 $x_{<i}$ 에 의해 결정됩니다.
- 이러한 autoregressive model은 주로 sequential data인 natural language, audio에 주로 사용되며, 2D 구조인 image도 sequential data로 인식하여 사용할 수 있습니다.



Autoregressive Models - loss functions

- Autoregressive model의 loss function은 모델의 확률분포 p_θ 에서 data \mathbf{x} 의 log-likelihood를 최대화하는 식으로 결정합니다.

$$\bullet \quad p_\theta(\mathbf{x}) = \prod_{i=1}^D p_\theta(x_i | x_{<i}),$$

- 양변에 log를 씌우면

$$\log p_\theta(\mathbf{x}) = \sum_{i=1}^D \log p_\theta(x_i | x_{<i})$$

- Neural network model이 구현하는 것은 과거의 scalar값들 $x_{<i}$ 를 입력으로 받아 현재의 데이터의 확률분포 $p_\theta(x_i | x_{<i})$ 를 구하는 것입니다.
- 확률분포 $p_\theta(x_i | x_{<i})$ 는 discrete distribution 또는 continuous distribution으로 결정할 수 있습니다.
- Discrete distribution으로는 주로 Bernoulli, categorical distribution을 사용하고, continuous distribution으로는 주로 Gaussian, Logistic distribution을 사용합니다.



Autoregressive Models - Bernoulli distribution

- Bernoulli distribution의 PMF (probability mass function)은 다음과 같이 주어집니다.

$$p(x | \mu) = Ber(x | \mu) = \mu^x (1 - \mu)^{(1-x)}$$

- 위에서 1이 일어날 확률을 가리키는 parameter μ 를 neural network로 결정합니다.
- 즉, neural network는 과거의 scalar값들 $x_{<i}$ 를 입력받아 현재의 값이 1이 될 확률을 출력합니다.

$$p_\theta(x_i | x_{<i}) = Ber(x_i | \mu_i = NeuralNetwork(x_{<i}))$$

- 위식에 log를 씌우고 정리하면, 다음과 같이 되고 이는 binary cross entropy로 구현이 됩니다.

$$\log p_\theta(x_i | x_{<i}) = \log Ber(x_i | \mu_i) = x_i \log \mu_i + (1 - x_i) \log(1 - \mu_i)$$

$$-\frac{1}{N} \sum_{n=1}^N \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

https://en.wikipedia.org/wiki/Cross_entropy



Autoregressive Models - categorical distribution

- Categorical distribution의 PMF (probability mass function)은 다음과 같이 주어집니다.
 \mathbf{x} 는 data가 어떤 category에 속하는지를 나타내는 one-hot vector이며, $\boldsymbol{\mu}$ 는 각 category에 속할 확률을 나타내는 vector입니다.

$$p(\mathbf{x} | \boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k}$$

- 위에서 각 category들이 일어날 확률을 가리키는 parameter $\boldsymbol{\mu}$ 를 neural network로 결정합니다.
- 즉, neural network는 과거의 scalar값들 $x_{<i}$ 를 입력받아 현재의 값 x_i 가 어떤 category에 속할 것인지 확률값을 출력합니다.

$$p_\theta(x_i | x_{<i}) = \text{cat}(x_i | \boldsymbol{\mu}_i = \text{NeuralNetwork}(x_{<i}))$$

- 위식에 \log 를 씌우고 정리하면, 다음과 같이 되고 이는 cross entropy로 구현이 됩니다.

$$\log p_\theta(x_i | x_{<i}) = \log \text{cat}(x_i | \boldsymbol{\mu}_i) = \sum_{k=1}^K x_{ik} \log \mu_k$$

$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x) \quad (\text{Eq.1})$$

https://en.wikipedia.org/wiki/Cross_entropy



Autoregressive Models - Gaussian distribution

- Gaussian distribution의 PDF (probability density function)은 다음과 같이 주어집니다.

$$\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma^2}(x - \mu)^2 \right\}$$

- 위에서 mean μ 와 standard deviation σ 를 neural network로 결정합니다.
- 즉, neural network는 과거의 scalar값들 $x_{<i}$ 를 입력받아 현재의 값 x_i 에 대한 distribution에 대한 파라미터 mean μ 와 standard deviation σ 를 출력합니다.
- $p_\theta(x_i | x_{<i}) = \mathcal{N}(x_i | \{\mu_i, \sigma_i\}) = \text{NeuralNetwork}(x_{<i})$
- 위식에 log를 씌우고 정리하면, 다음과 같이 되고, mean μ 를 구하는 것은 MSE loss에 해당합니다.
- $\log p_\theta(x_i | x_{<i}) = \log \mathcal{N}(x_i | \mu_i, \sigma_i) = -\frac{1}{2} \log 2\pi + \log \sigma_i + \frac{1}{2\sigma^2}(x_i - \mu_i)^2$



Fully Visible Sigmoid Belief Network

FVSBN

- Fully Visible Sigmoid Belief Network는 linear layer와 sigmoid activation을 이용하여 이미지 데이터를 생성하는 모델입니다.
- Linear layer만으로 구성되었기에 좋은 성능을 내지는 않지만 autoregressive model을 이해하기 좋은 예제입니다.
- MNIST data는 784개의 pixel로 이루어져 있습니다. 첫 pixel x_1 의 값이 1일 확률 $p(x_1)$ 를 모델의 parameter로 정합니다. $p(x_1)$ 는 다른 어떤 값을 조건으로하여 결정되지 않습니다.
- 이어서 두번째 pixel x_2 의 값이 1일 확률을 첫 pixel x_1 를 조건으로하여 정합니다. $p(x_2 | x_1)$ 즉, linear layer에 첫 pixel x_1 을 입력하여 나온 출력에 sigmoid activation을 적용한 값을 두번째 pixel x_2 의 값이 1일 확률로 정합니다. $p(x_2 | \sigma(\text{Linear}(x_1)))$
- 같은 방식으로 세번째 x_3 , 네번째 x_4 , 마지막 pixel x_{784} 의 값까지 autoregressive적으로 결정해나가면 한장의 MNIST data를 완성할 수 있습니다.

- $$p(\mathbf{x}) = p(x_1) \prod_{i=2}^{784} p(x_i | \sigma(\text{Linear}(x_{<i}))), \text{ where } \mathbf{x} = \{x_1, x_2, \dots, x_{784}\}$$



FVSBN Results

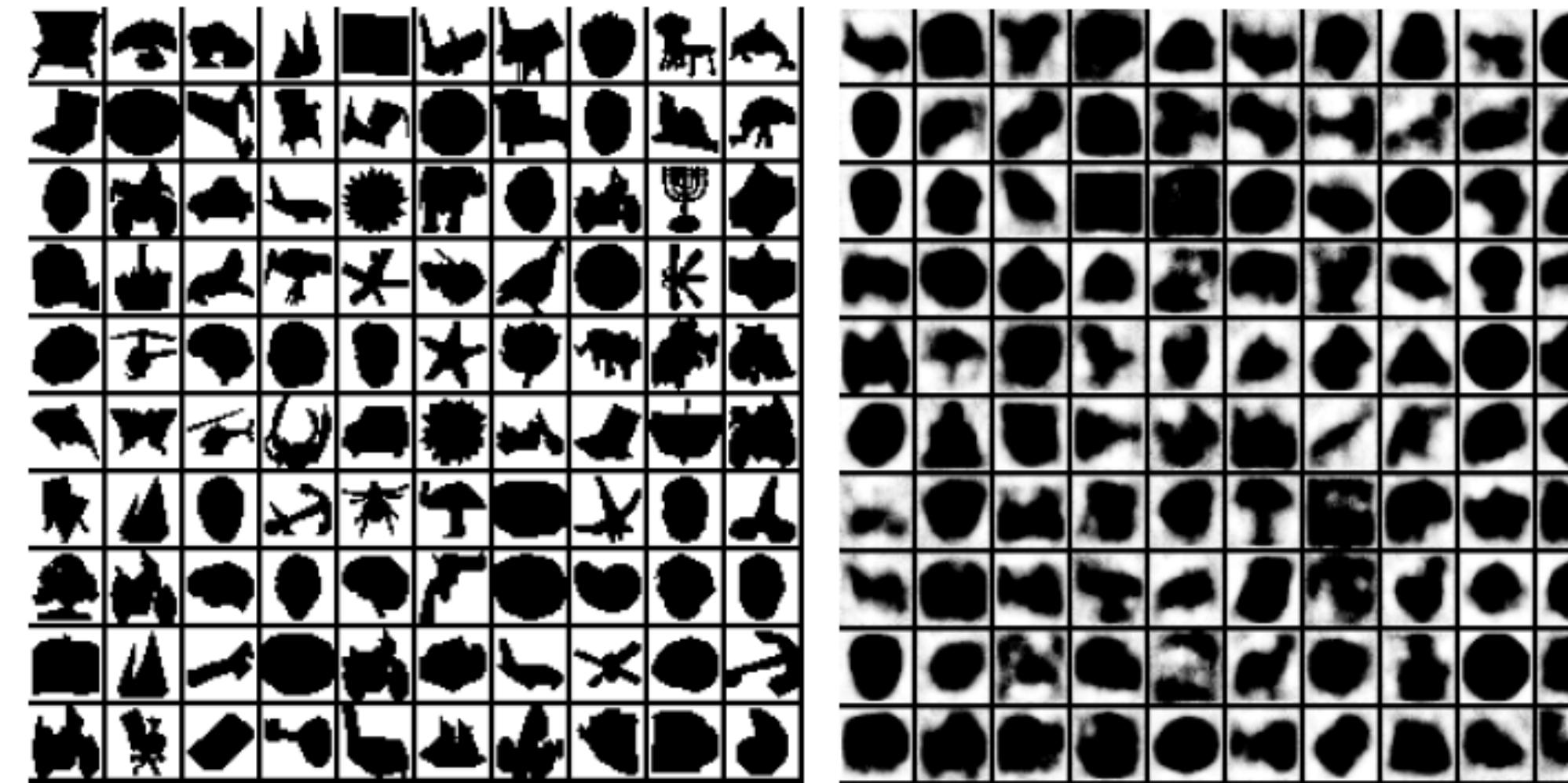


Figure from *Learning Deep Sigmoid Belief Networks with Data Augmentation*, 2015. Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right. Best performing model they tested on this dataset in 2015 (more on evaluation later).



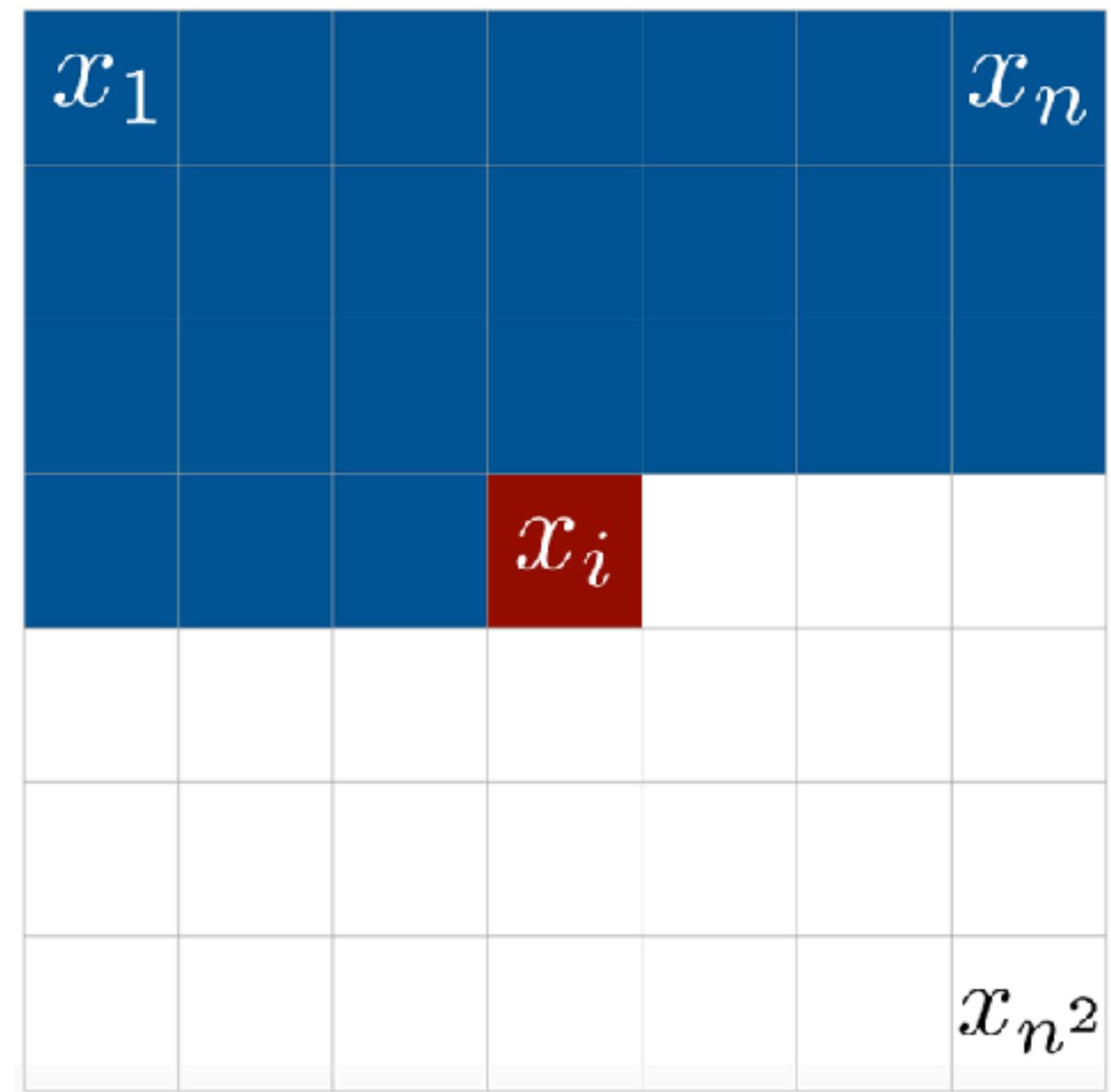
PixelCNN

PixelCNN

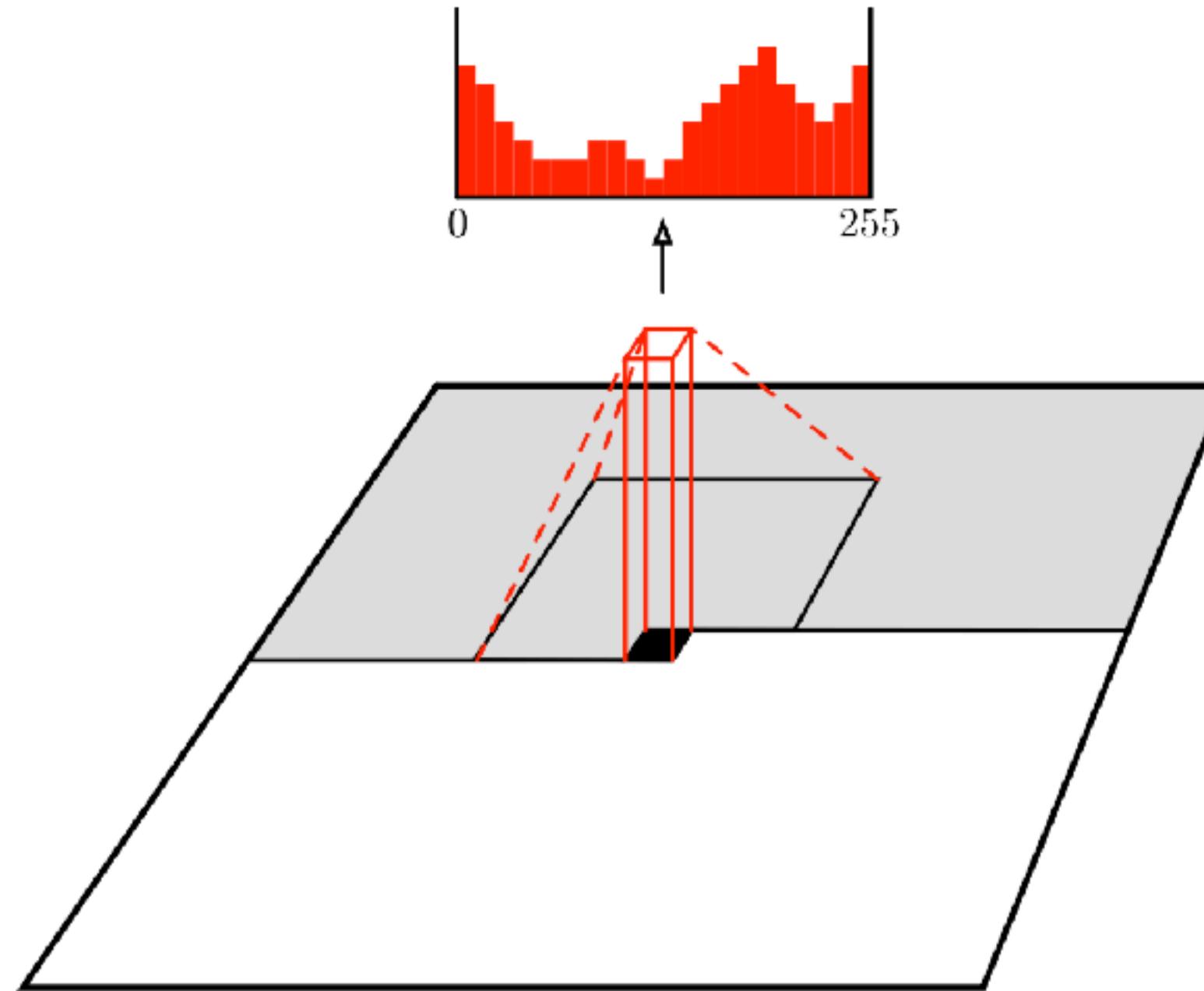
- PixelCNN은 Conditional Image Generation with PixelCNN Decoders논문에서 제안된 모델입니다.
- PixelCNN은 autoregressive적으로 생성하면서도 이미지의 2D 특성을 반영하기 위해 CNN을 도입합니다.
- 한 pixel씩 값을 생성하며 해당 pixel을 생성하기 위해 condition으로 주어지는 영역을 receptive field라고 부릅니다.
- Receptive field영역의 pixel값들이 CNN의 입력으로 들어가 다음 pixel의 값에 대한 확률분포의 파라메터를 출력합니다.
- Training시에는 이미지의 모든 영역의 값이 있는 상태이기 때문에 filter에 mask를 써워 현재와 미래에 prediction할 영역의 값을 읽지 못하도록 만듭니다.



PixelCNN



이미지의 pixel들을 좌상단에서 우하단까지
하나씩 생성하는 방식

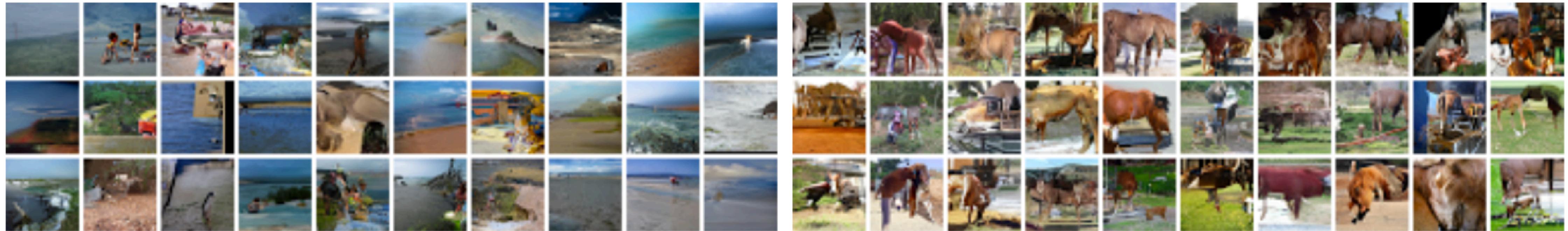


Receptive field를 CNN으로 분석하고
다음 pixel값의 확률 분포를 예측

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

Mask된 5x5 filter

Results via PixelCNN

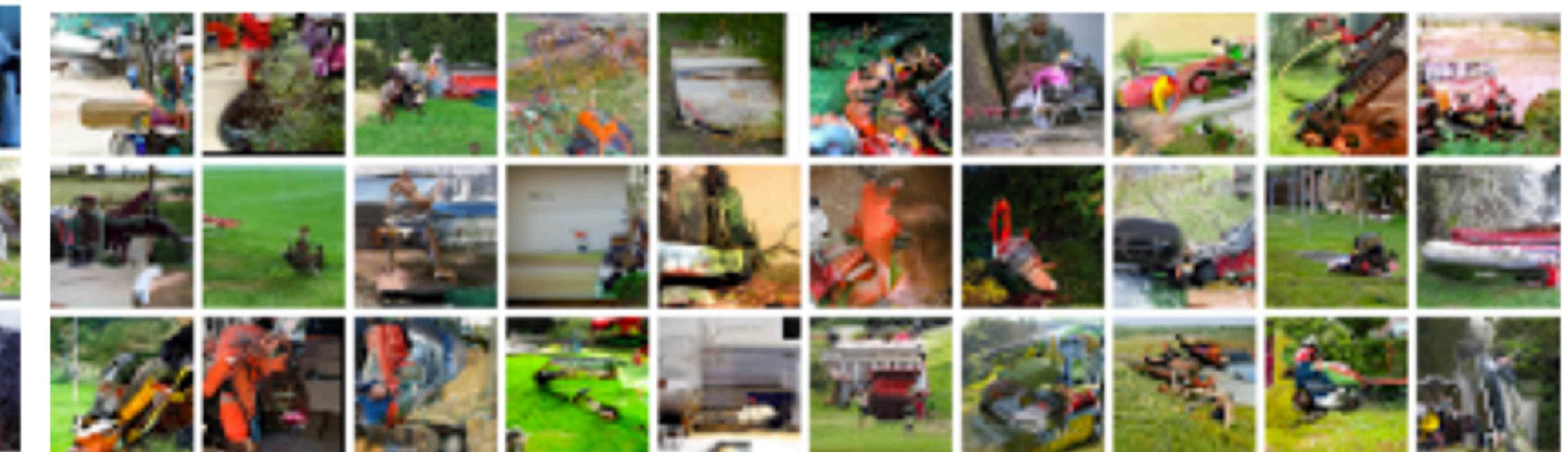


Sandbar

Sorrel horse



Lhasa Apso (dog)



Lawn mower



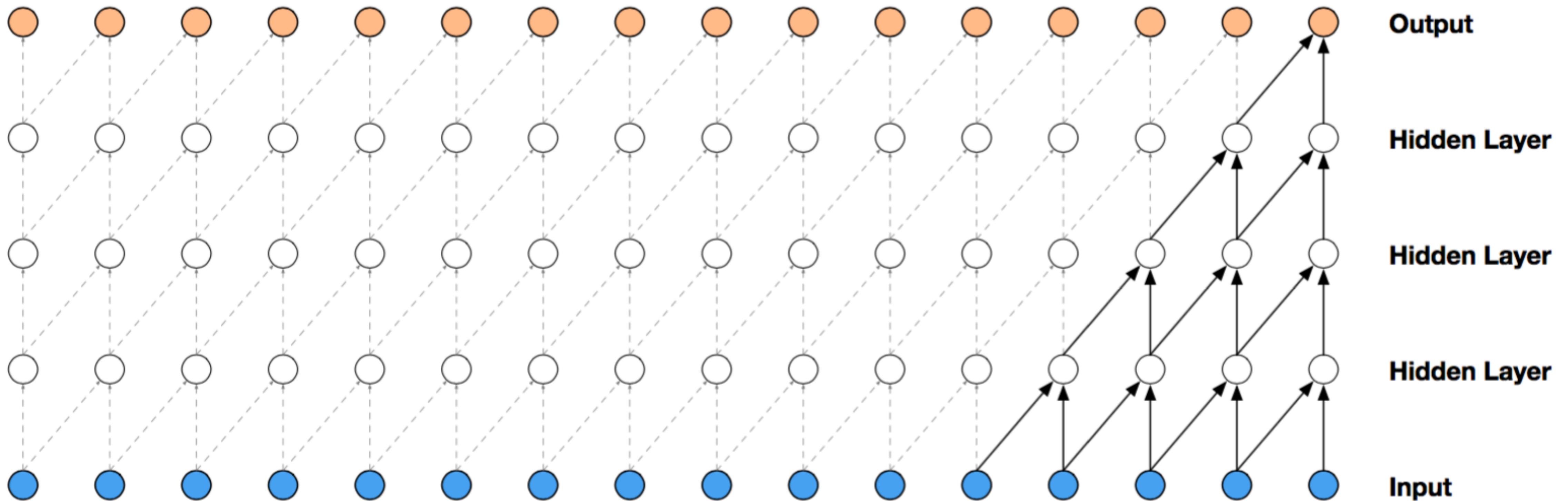
Wavenet

Wavenet

- Wavenet은 WaveNet: A Generative Model for Raw Audio 논문에서 제안된 모델입니다.
- PixelCNN과 같은 저자에 의해 연구된 모델이고 2D 이미지 대신 1D 오디오 데이터를 사용하였다는 점을 제외하면 거의 구조가 비슷합니다.
- Audio data는 매우 길이가 길기 때문에 dilated convolution을 사용하여 receptive field를 키우는 방식을 적용했습니다.
- Wavenet은 neural network를 이용하여 기존의 vocoder들의 품질을 뛰어넘는 고품질의 오디오를 생성한 최초의 모델이며, 아직까지 품질 면에서는 SOTA를 유지하고 있습니다.
- Autoregressive적으로 생성하므로 매우 느리다는 단점을 가지고 있어, 이후에 flow를 이용한 WaveGlow, GAN을 이용한 Parallel Wavenet 등이 등장했습니다.



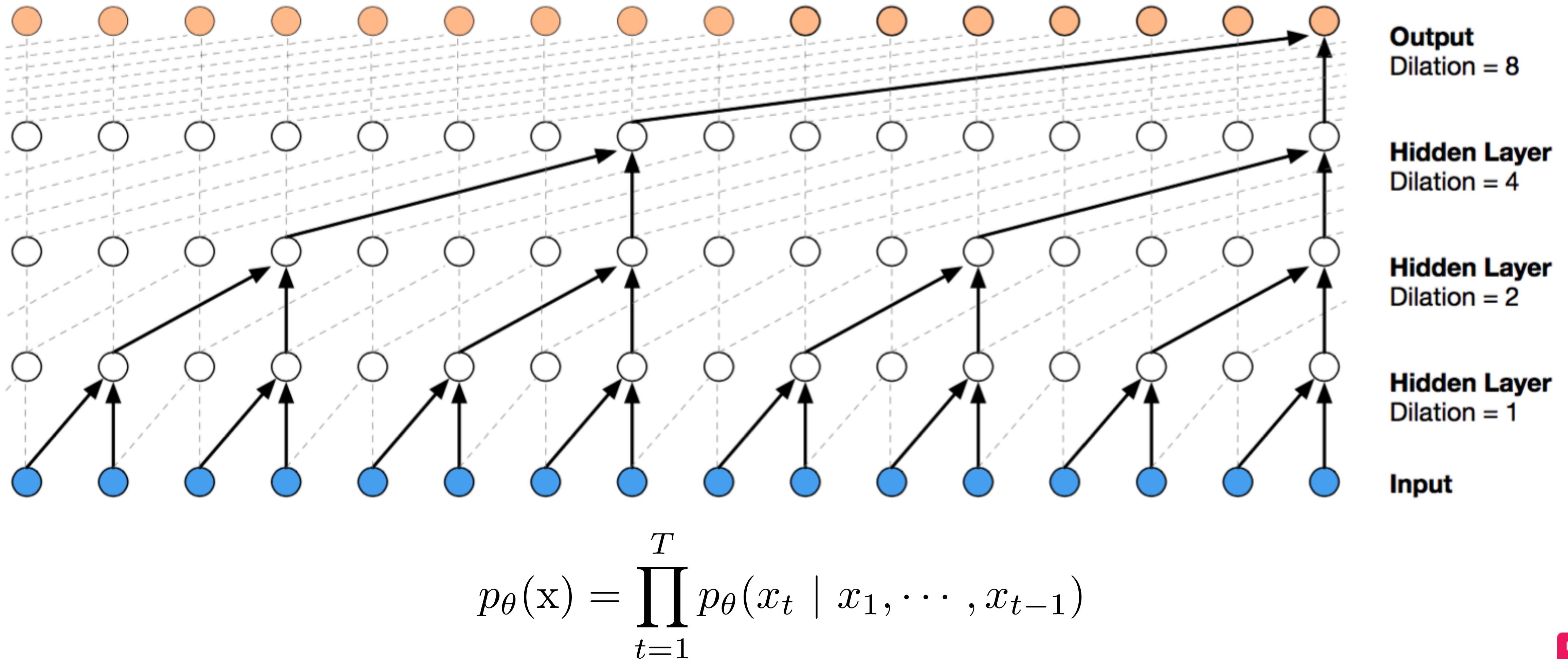
WaveNet (Naive Causal Convolution)



$$p_{\theta}(\mathbf{x}) = \prod_{t=1}^T p_{\theta}(x_t \mid x_1, \dots, x_{t-1})$$



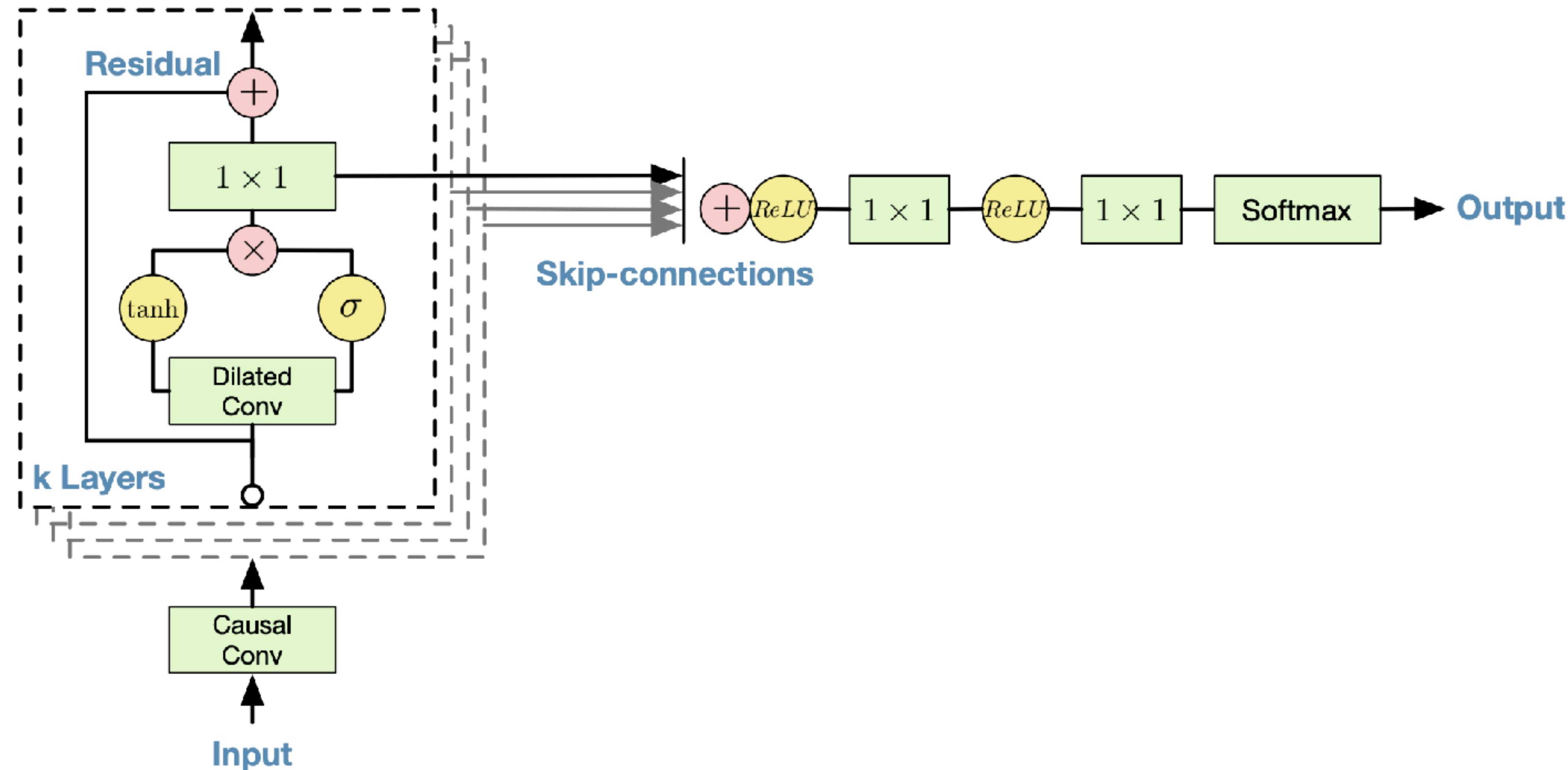
WaveNet



Demo: <https://www.deepmind.com/blog/wavenet-generative-model-raw-audio>

Figure credit: Aäron van den Oord, et al., Wavenet: A Generative Model for Raw Audio

WaveNet



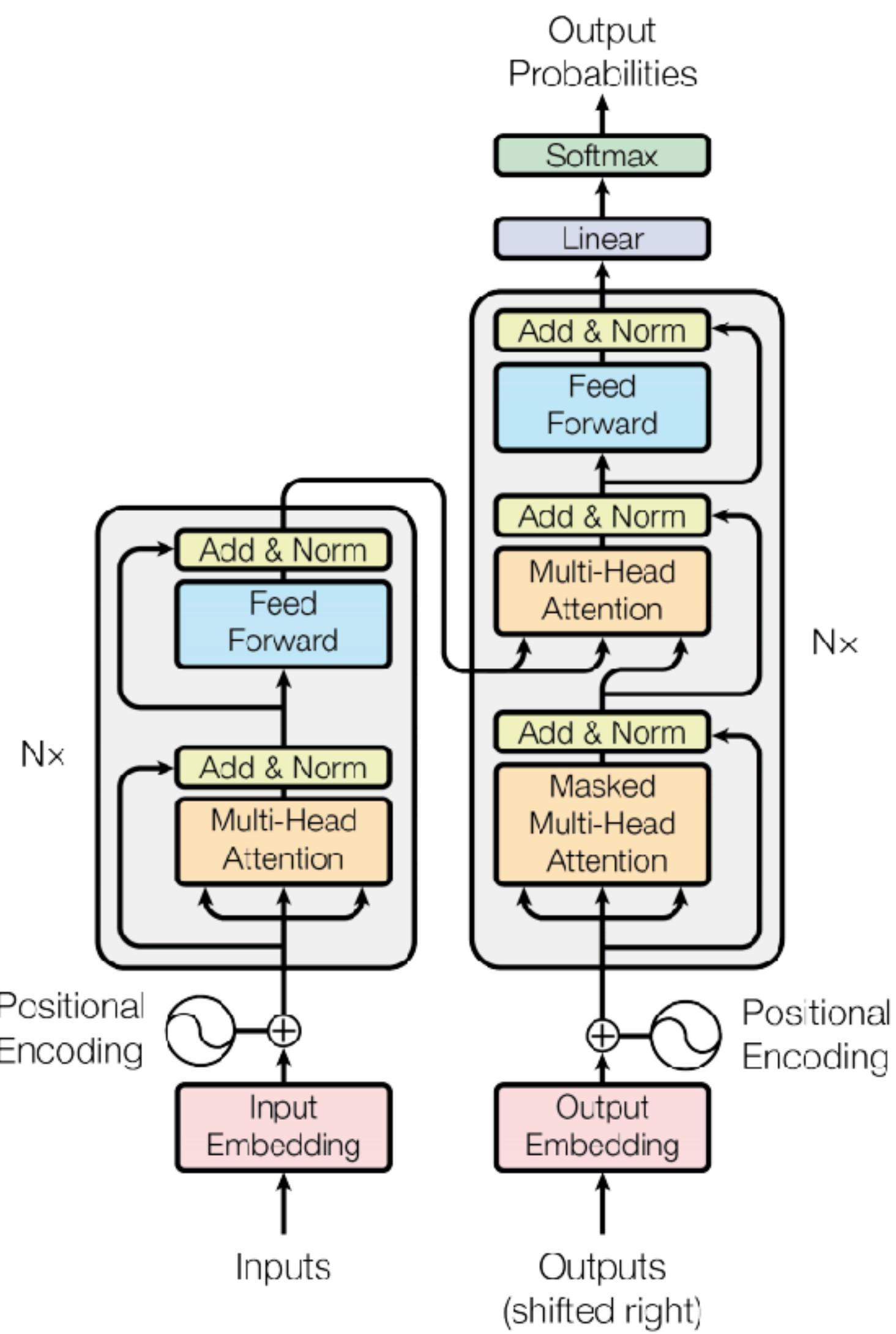
Generation of Wave

- Raw audio generation
- Neural autoregressive generative models that model complex distributions (ex. images and text)
- PixelRNN: thousands of random variables (e.g. 64x64)
- WaveNet: very high temporal resolution (16,000 samples)
- Based on the PixelCNN



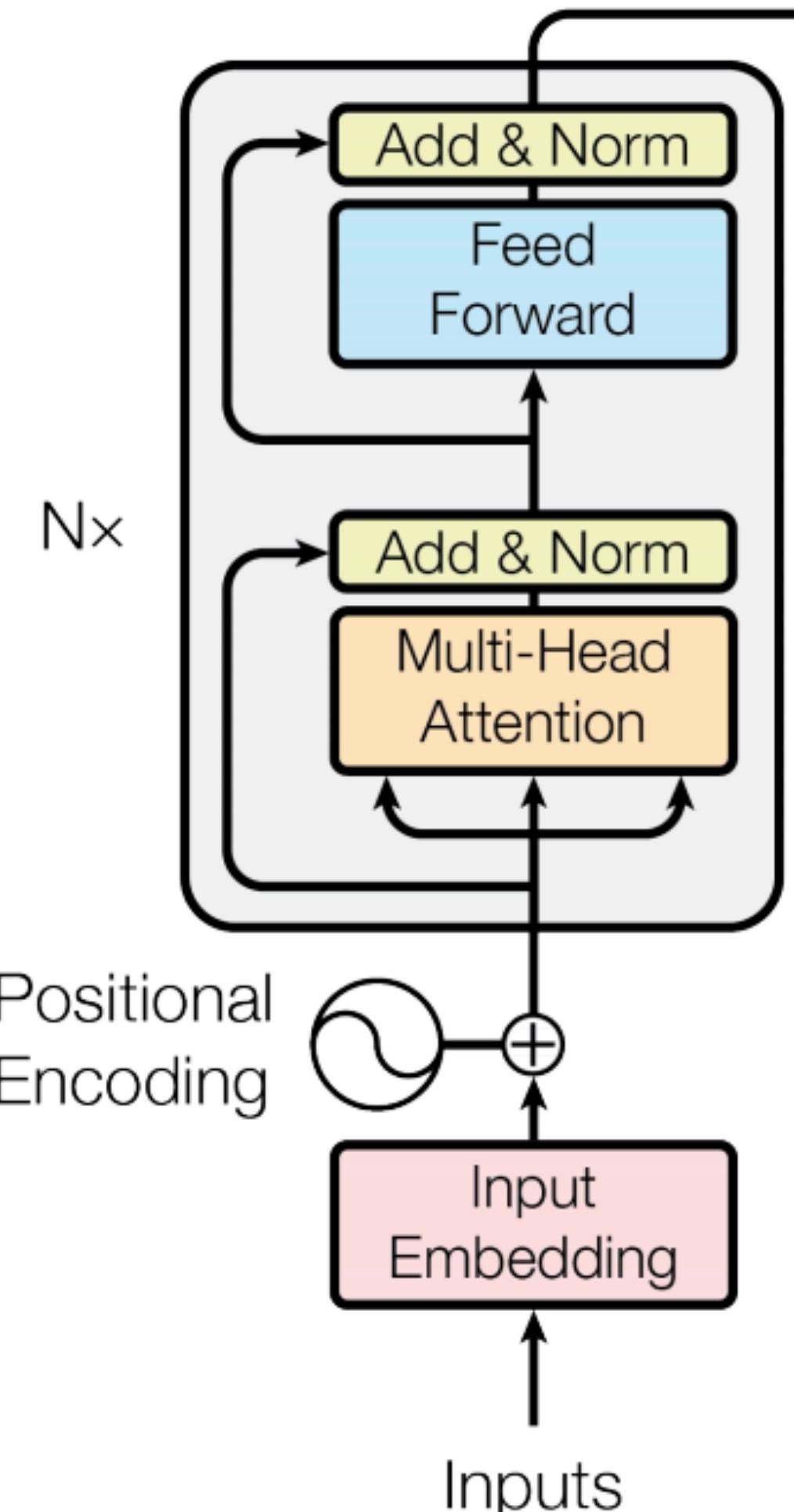
Transformer

Transformer



- Transformer는 크게 Inputs 데이터를 인코딩하는 Encoder와, 인코딩된 데이터를 기반으로 Outputs을 디코딩하여 Outputs의 확률분포 파라메터를 출력하는 Decoder로 구성됩니다.
- 이러한 구조는 seq2seq 모델이라고 불리며 원래 언어의 문장을 대상 언어의 문장으로 변환시키는 번역 작업이나, 질의 문장을 토대로 응답 문장을 생성해내는 질의 응답 작업 등에 사용됩니다.
- Encoder는 원래 언어의 문장의 토큰을 입력으로 받아 임베딩 테이블을 이용해 벡터로 변환하고 여러 인코딩 블록을 거쳐 최종적으로 Decoder에 들어갈 형태로 출력합니다.
- Decoder도 마찬가지로 대상 언어의 문장의 토큰을 입력으로 받아 임베딩 테이블을 이용해 벡터로 변환하고 여러 디코딩 블록을 거쳐 다음 토큰에 해당하는 확률 분포의 파라메터를 출력합니다.
- 이 때, 디코딩 블록 내에서 어텐션 메카니즘을 통해 Encoder에서 인코딩된 정보를 가져옵니다. 어텐션 시 여러 헤드로 분리하여 각각 다양한 부분의 정보를 가져올 수 있도록 하며, 가져온 정보들은 concatenation하여 다음 레이어로 전해집니다.

Transformer - Encoder

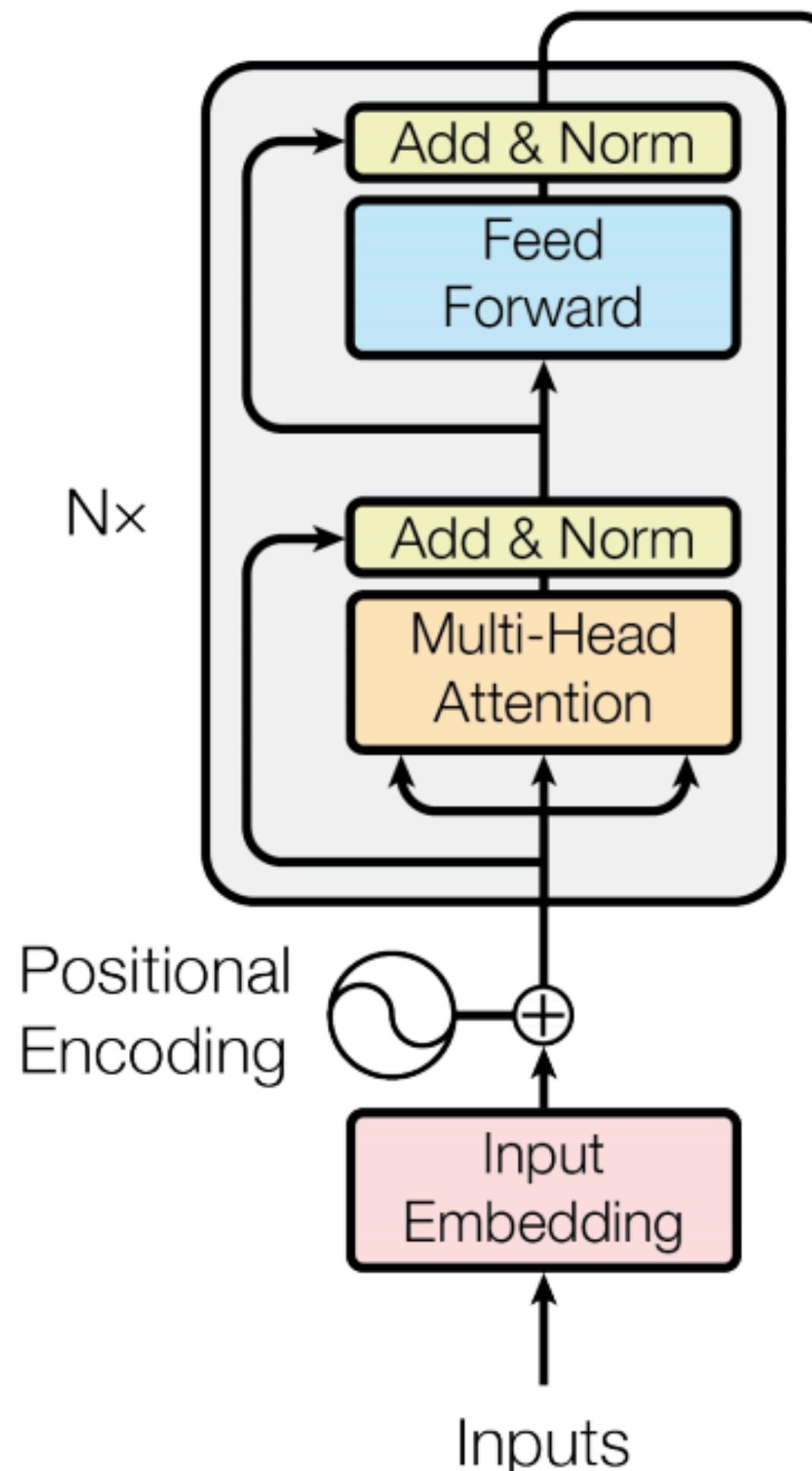


- Encoder는 Inputs, Input Embedding, Positional Encoding, Attention Blocks으로 구성되며, Attention Blocks내부는 Multi-Head Attention, Feed forward와 Layer Normalization 으로 구성됩니다.
- Inputs : 번역 작업의 경우 원본 문장, 질의 응답 작업의 경우 질의에 해당하는 입력 문자열에 해당합니다. 문자열에 대응하는 토큰이 입력으로 주어집니다.
- Input Embedding : 임베딩 테이블을 통해 각 토큰을 임베딩 벡터로 변환합니다. 임베딩 테이블은 그래디언트를 받아 트레이닝 가능한 상태로 둡니다.
- Positional Encoding : Attention 레이어는 CNN, RNN과 다르게 위치에 대한 정보가 출력에 반영되지 않으므로 별도로 위치 정보를 임베딩하는 것이 필요합니다. 위치 값을 아래와 같은 sin과 cos함수에 적용한 출력 값을 Input Embedding 결과에 더합니다.

$$PE_{(pos,2i)} = \sin\left(\text{pos}/10000^{2i/d_{\text{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\text{pos}/10000^{2i/d_{\text{model}}}\right)$$

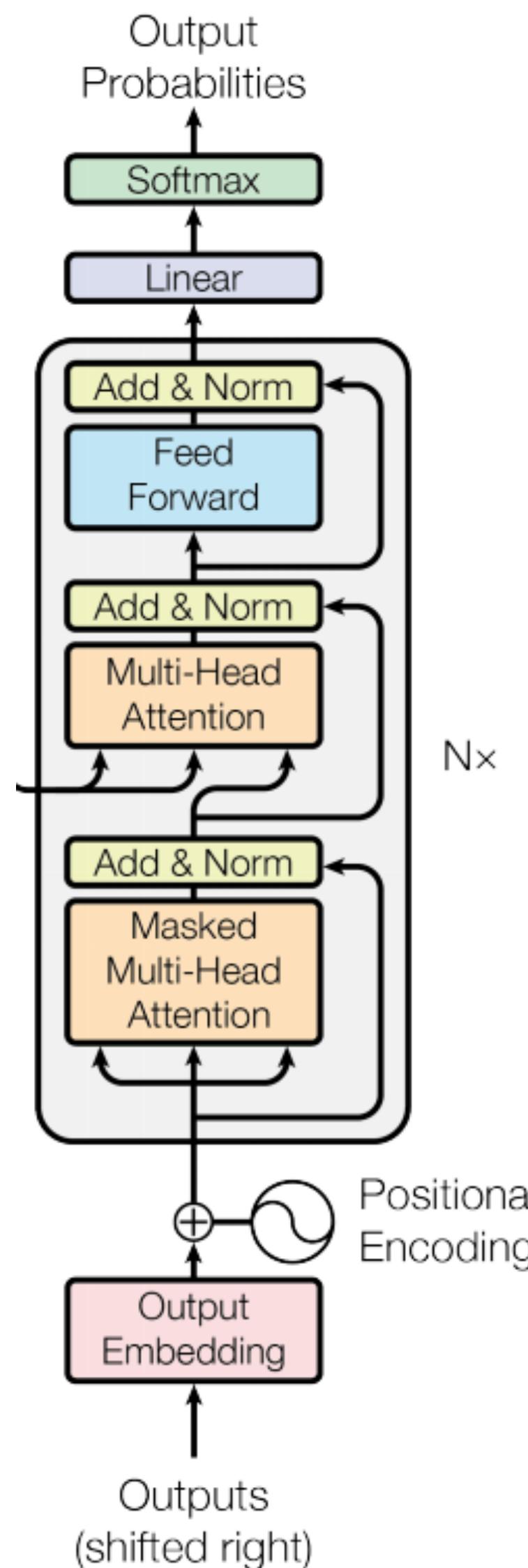
Transformer - Encoder



- Multi-Head Attention : 입력으로 들어온 벡터를 여러 Head로 나누어 각각 다른 어텐션 정보를 계산합니다. 이러한 작업으로 어텐션 영역 내에서 다양한 정보를 가져다 사용할 수 있는 능력을 부여합니다.
- Add & Norm : Multi-Head Attention의 결과와 입력을 더한다. 이러한 Residual Connection을 통해 깊은 네트워크를 구축하면서도 Backpropagation시 Gradient가 잘 전파되도록 돋습니다. 또한 Layer Normalization을 사용하여 Add 연산이 반복됨에도 안정된 값을 출력하도록 합니다.
- Feed Forward : 두 개의 Fully Connected 레이어와 ReLU Activation으로 구성됩니다. 내부 레이어의 차원은 입력보다 4배로 크게 설정됩니다.

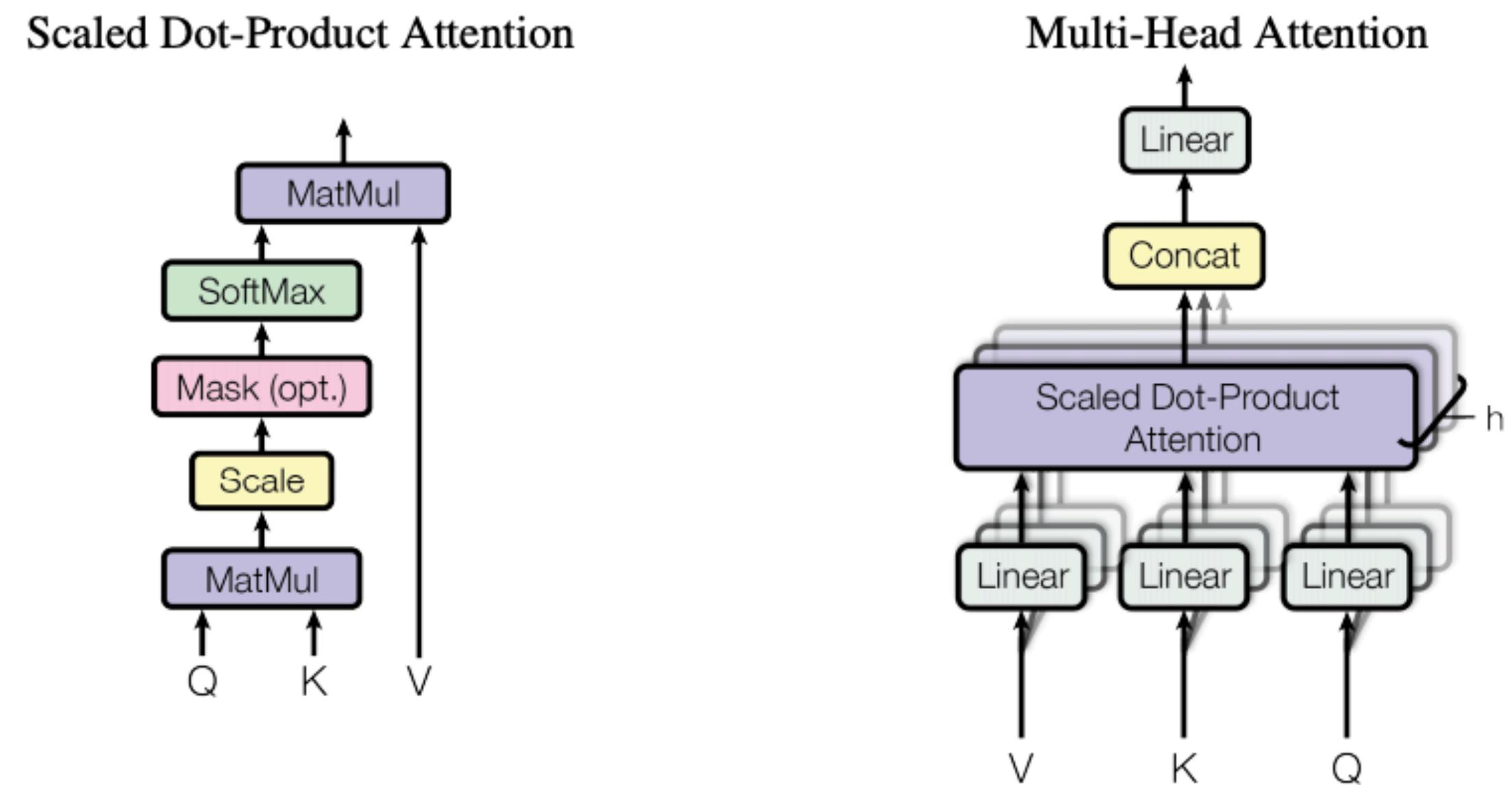
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Transformer - Decoder



- Decoder는 Outputs, Outputs Embedding, Positional Encoding, Attention Blocks으로 구성되며, Attention Blocks내부는 Masked Multi-Head Attention, Feed Forward와 Layer Normalization 으로 구성됩니다.
- Outputs : 번역 작업의 경우 대상 문장, 질의 응답 작업의 경우 응답문장에 해당합니다. 문자열에 대응하는 토큰이 입력으로 주어집니다.
- Outputs Embedding : Encoder의 Inputs Embedding과 같은 방식으로 작동합니다.
- Positional Encoding : Encoder의 Positional Encoding과 같은 방식으로 작동합니다.
- Masked Multi-Head Attention : Encoder의 Multi-Head Attention과 같지만 현재보다 과거의 정보들만을 참조하도록 Mask를 사용합니다.
- Add & Norm : Encoder의 Add & Norm과 같은 방식으로 작동합니다.
- Feed Forward : Encoder의 Feed Forward과 같은 방식으로 작동합니다.

Transformer - Multi-Head Attention



- Multi-Head Attention은 입력 벡터들을 선형 변환하여 Value, Key, Query 값을 얻고, 각 Query와 Key 값들 간에 dot-product를 취하여 얼마나 정보를 취할지 결정하는 Attention Score를 얻은 뒤, 이 값을 토대로 Value 값들을 선형 결합하여 최종적인 출력 벡터를 얻어냅니다. 이 때 입력 벡터를 각 Head로 분리하고 출력 벡터를 concatenation을 통해 다시 합쳐 다양한 어텐션 가능성을 얻을 수 있도록 합니다.

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

- Query와 Key의 내적을 각 Head의 차원 d_k 의 루트 값으로 나누어줌으로써 Head의 차원에 관계없이 내적값이 안정되도록 도와줍니다.

Image Transformer

- Image Transformer는 이미지를 생성하기 위해 Transformer를 사용합니다.
- 논문에서는 32x32 사이즈의 CIFAR-10 이미지를 dataset으로 사용합니다. 8x8 사이즈로 이미지를 줄인 뒤, 다시 32x32 사이즈로 키우는 super resolution과 unconditional generation 또는 class-conditional generation 작업을 수행합니다.
- 이미지는 기본적으로 2D 구조의 데이터이므로 2D attention을 사용하거나, flatten하여 펼쳐놓고 1D attention을 사용합니다.
- 이미지의 특성상 멀리 떨어져 있는 부분끼리는 관계가 적을 것을 가정하고 local attention을 사용합니다.
- 이를 위해 이미지를 query block과 이를 포함하는 memory block으로 나누고, query block에서는 Transformer에서 계산될 query를 만들고, memory block에서는 key와 value를 만듭니다.
- Autoregressive model이므로 Transformer의 출력은 다음 pixel의 값에 대한 확률 분포의 파라미터가 됩니다.

Image Transformer

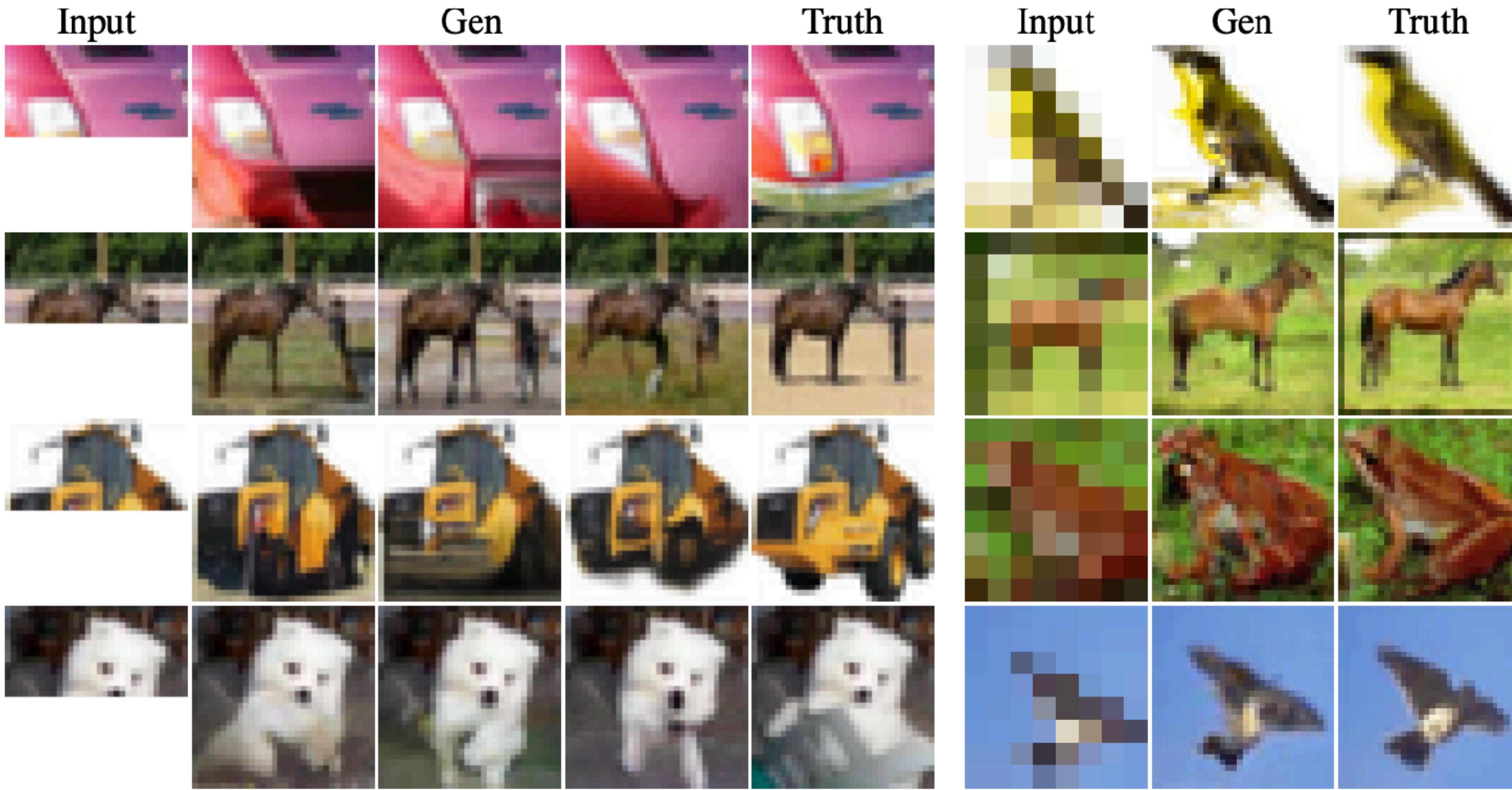
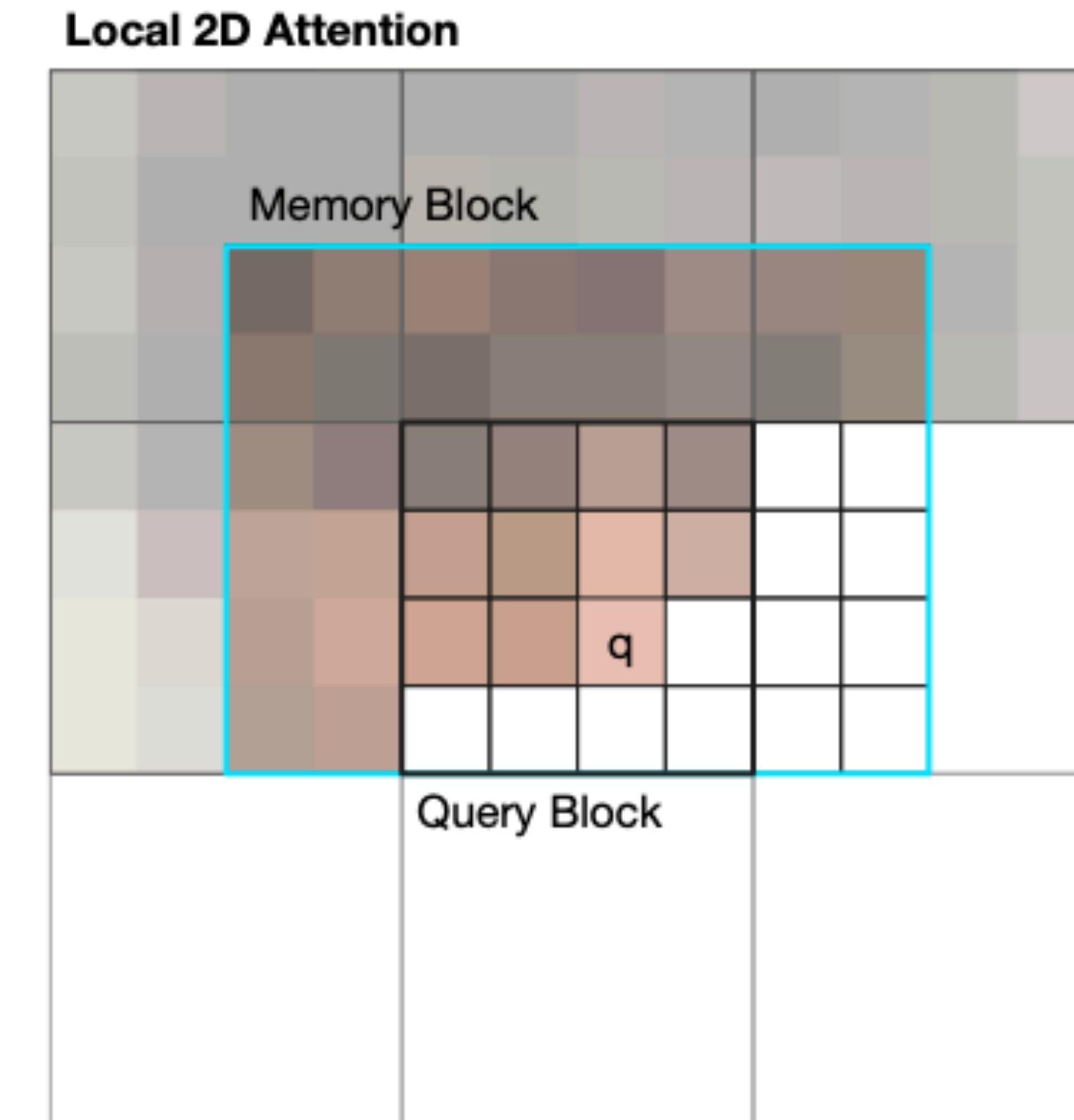
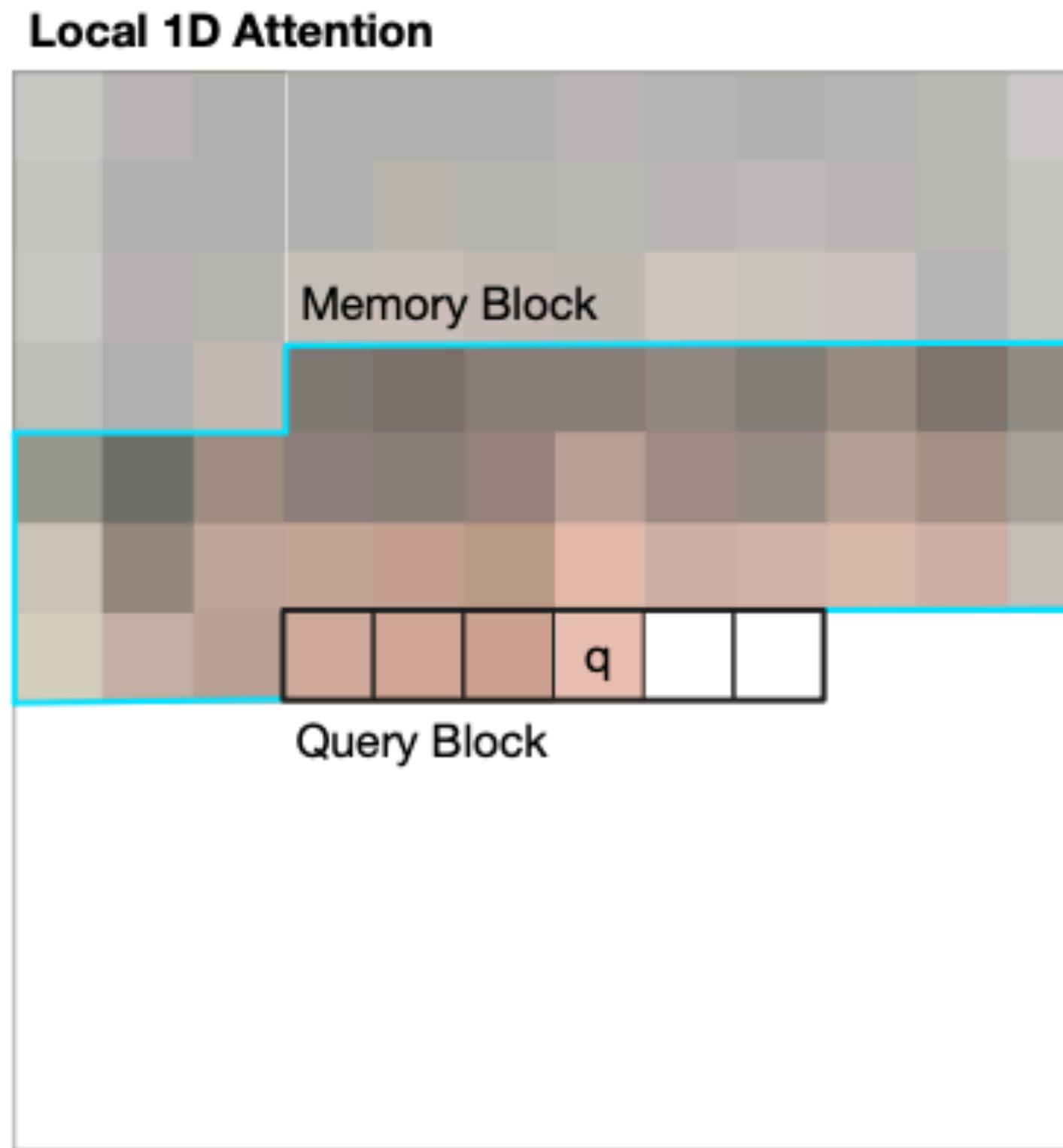


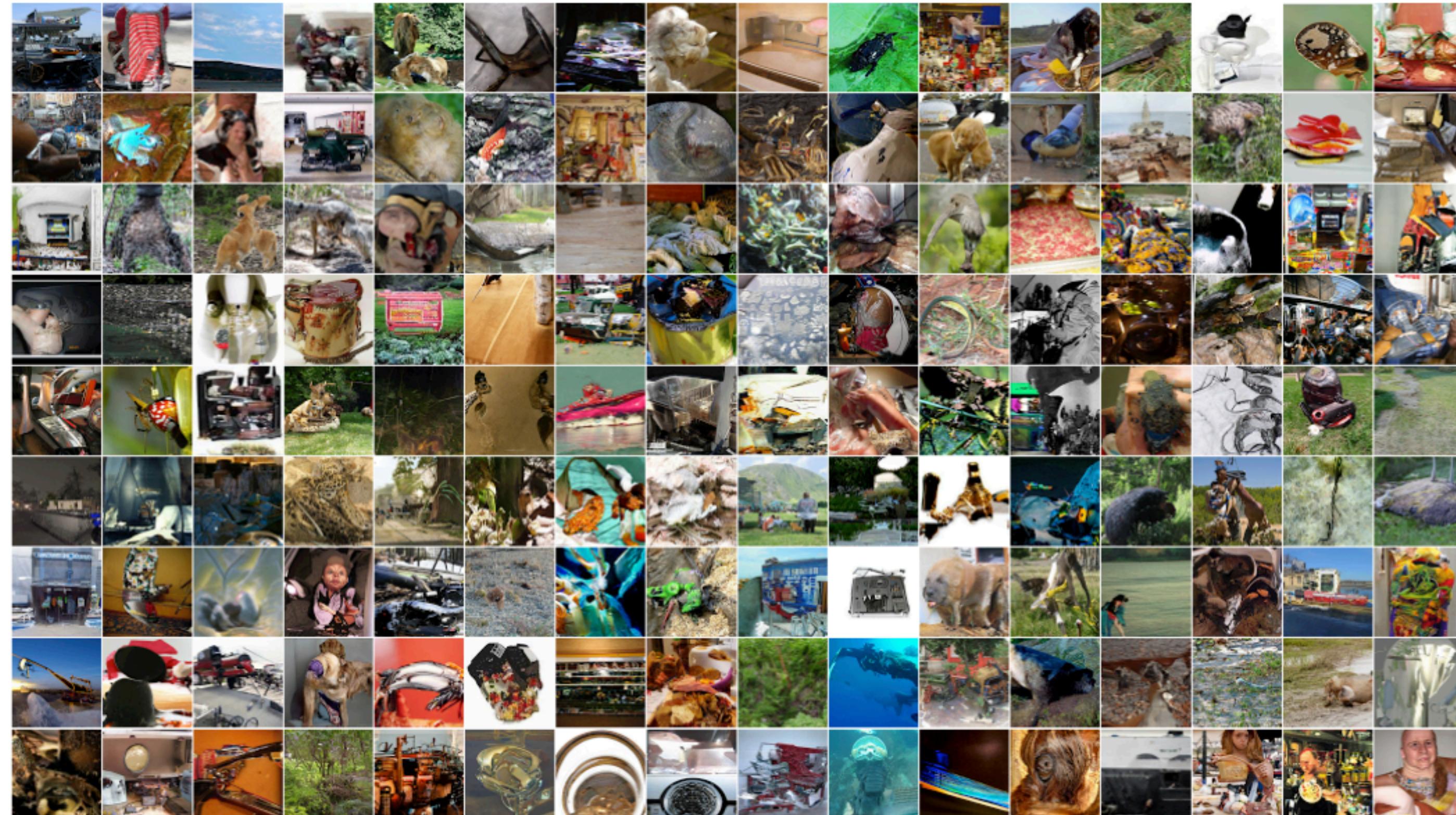
Image Transformer



Sparse Transformer

- Sparse Transformer는 Generating Long Sequences with Sparse Transformers 논문에서 제안되었습니다.
- Transformer는 모든 pixel을 촘촘하게 attention하기 때문에 속도가 느립니다.
- 자연어와 달리 이미지 데이터는 근접한 pixel끼리는 값이 비슷하기 때문에 촘촘하게 모든 값을 attention할 필요가 없습니다.
- 위와 같은 이유로 더 sparse하게 attention하여 속도와 memory를 줄일 수 있습니다.
- 대신 더 많은 수의 layer를 쌓아 네트워크의 분석 능력을 끌어올려 생성 능력을 향상 시켰습니다.

Sparse Transformer - Examples



Sparse Transformer로 생성한 이미지, 오디오 음악 데이터
<https://openai.com/blog/sparse-transformer/>

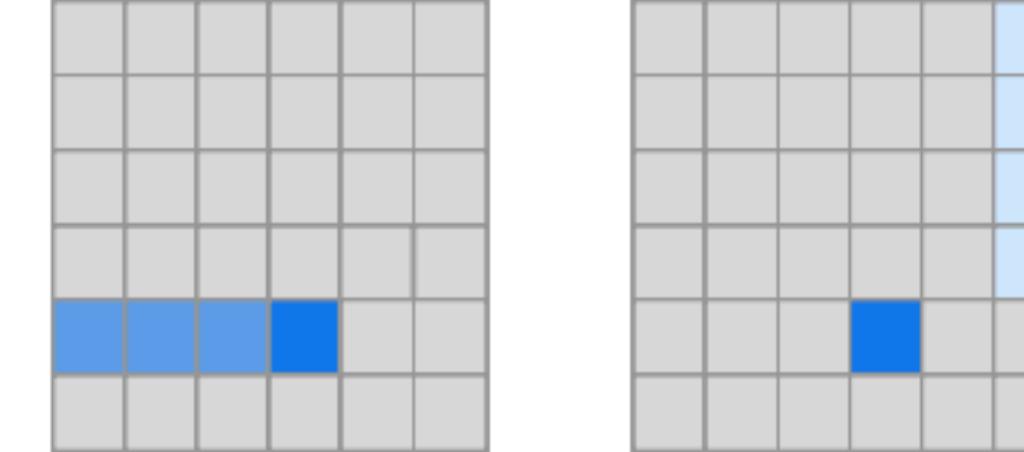
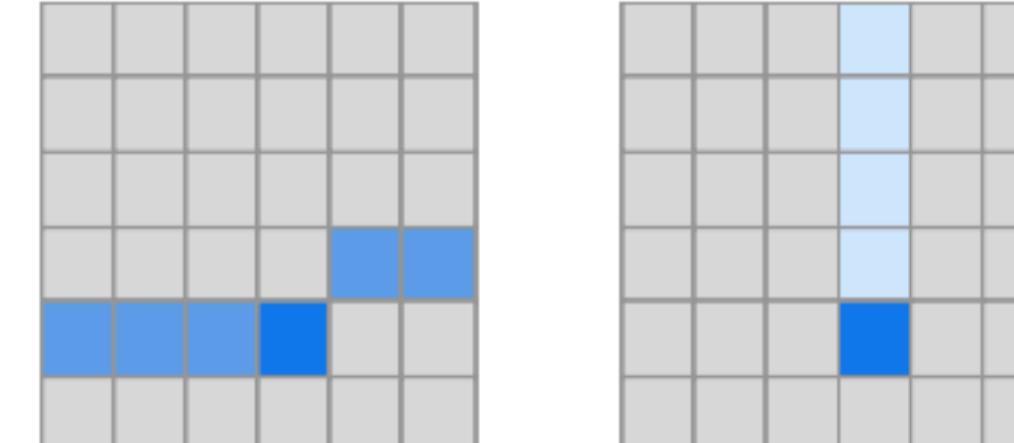
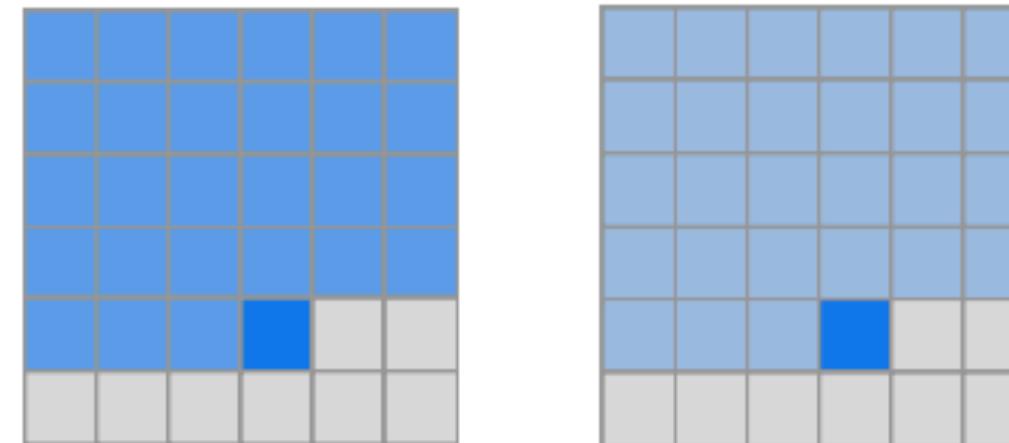
Sparse Transformer로 생성한 음악 데이터
<https://openai.com/blog/musenet/>

Sparse Transformer

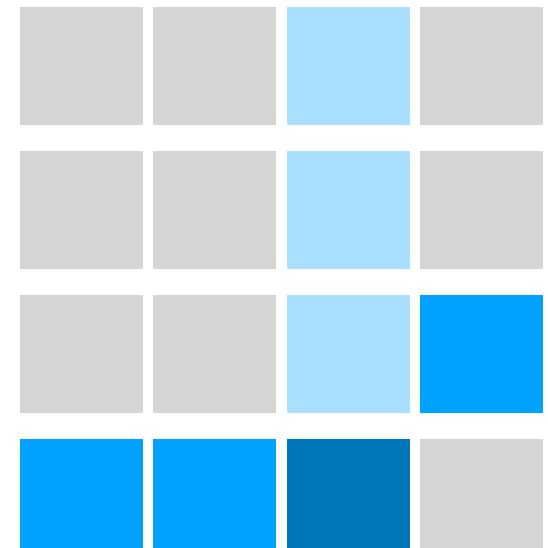
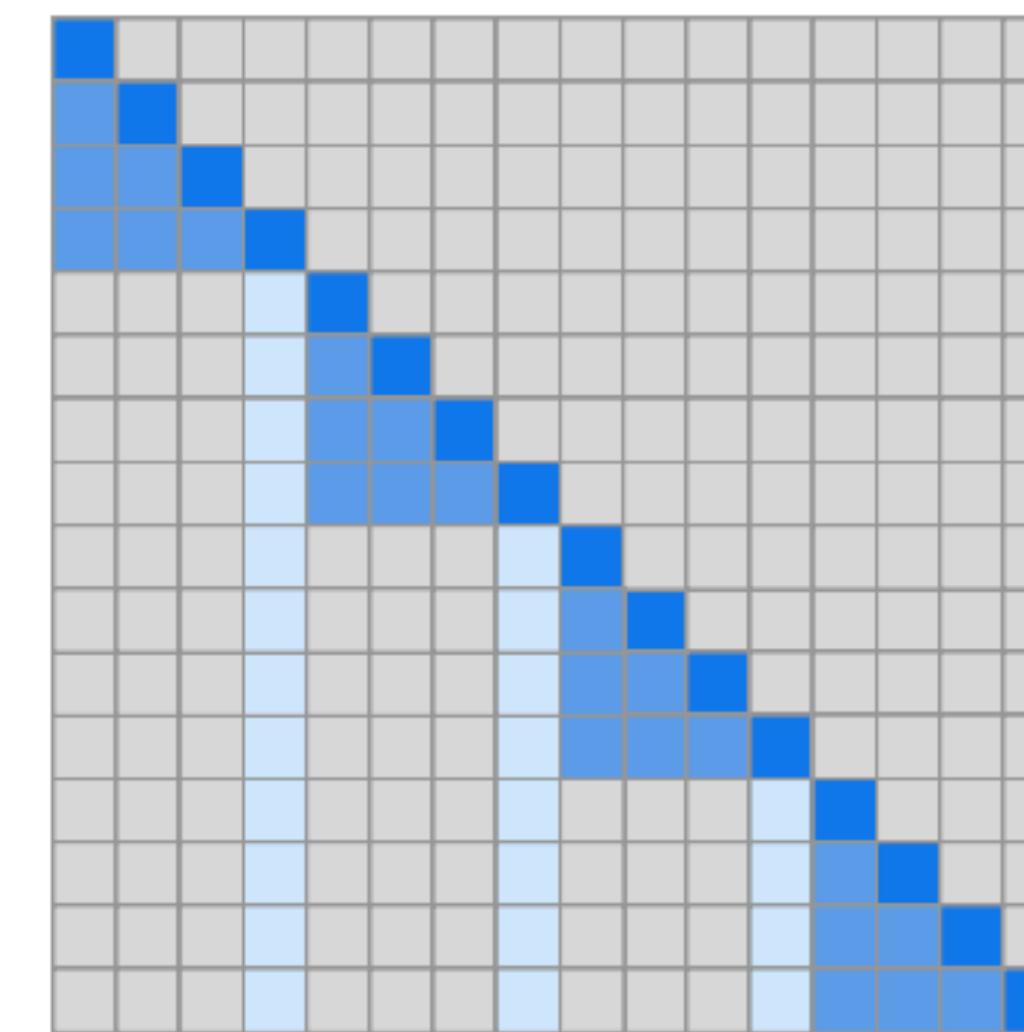
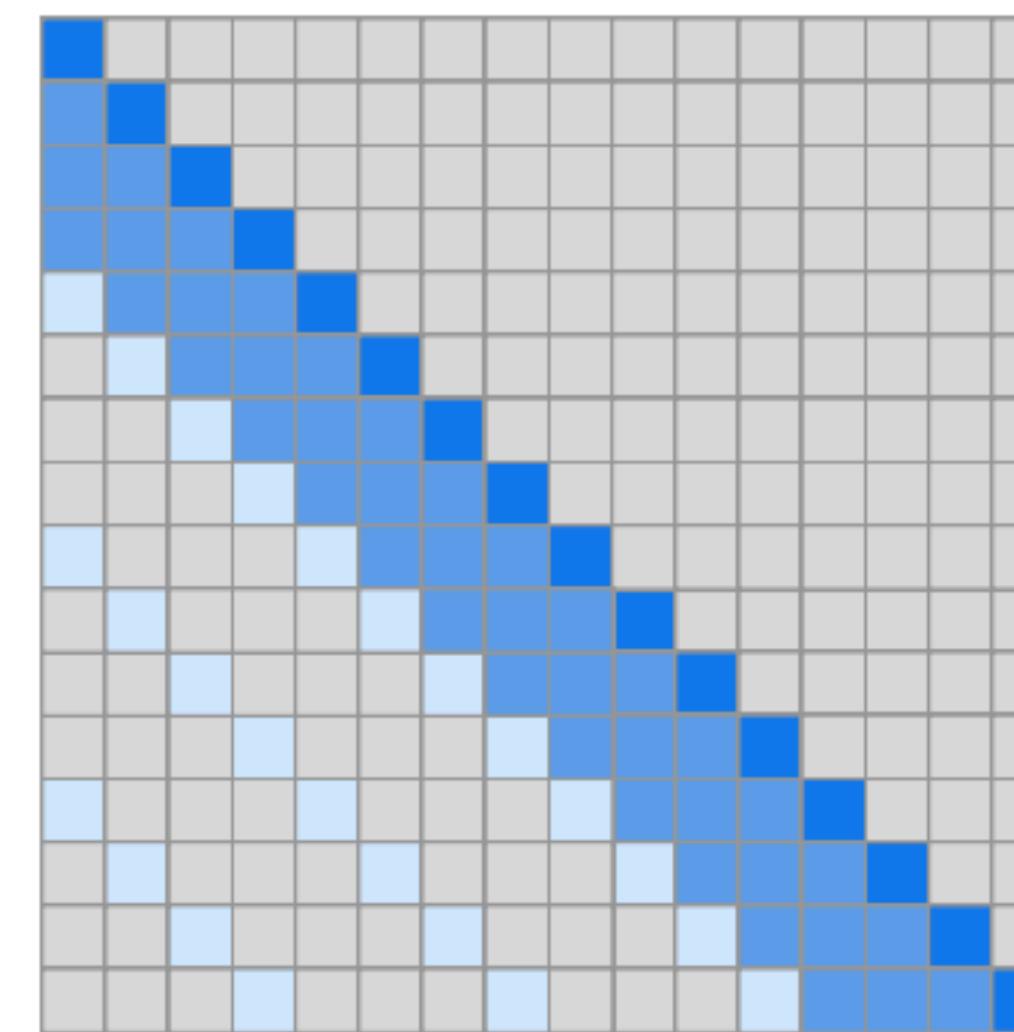
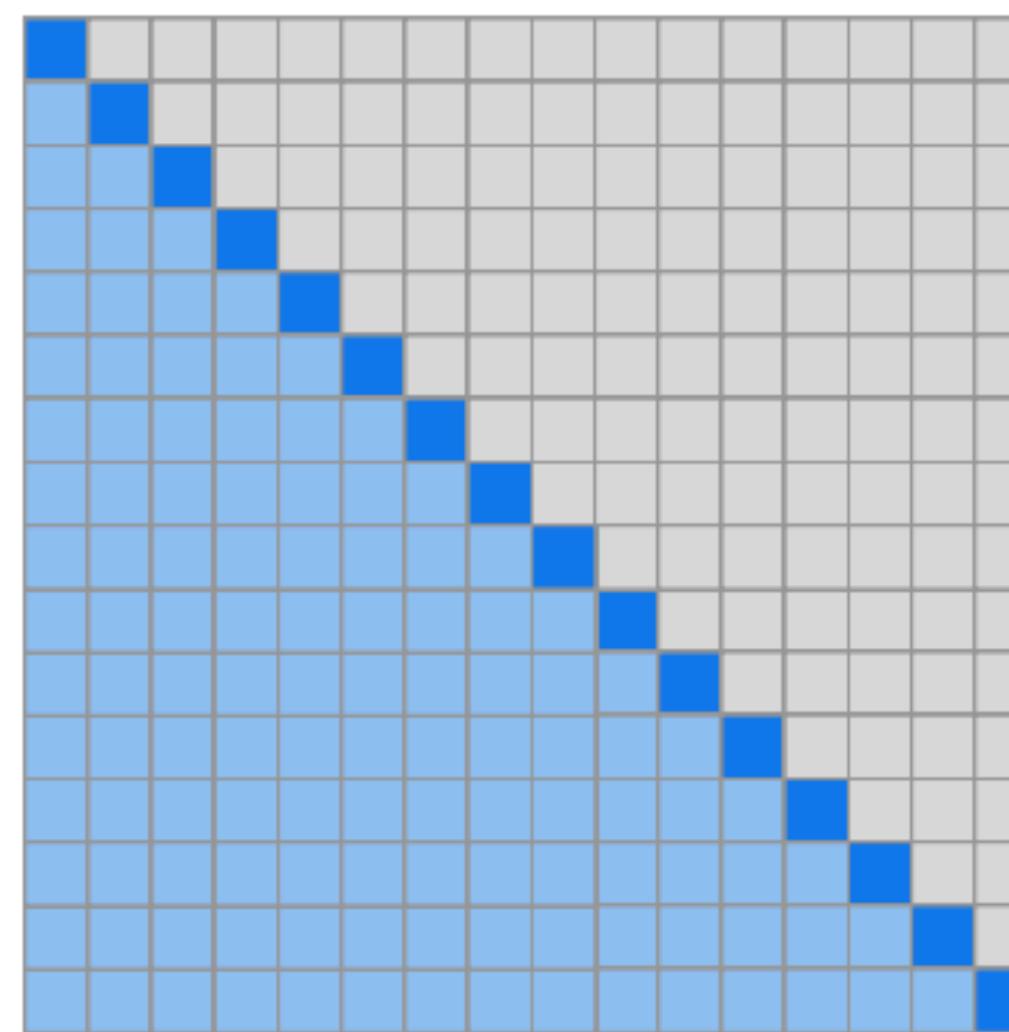


- CIFAR-10을 128-layer를 가진 full attention Transformer로 트레이닝한 그림
 - Layer마다 서로 다른 부분을 attention하려는 특징을 갖는 것을 발견할 수 있음

Sparse Transformer



**4x4 example
(strided)**



(a) Transformer

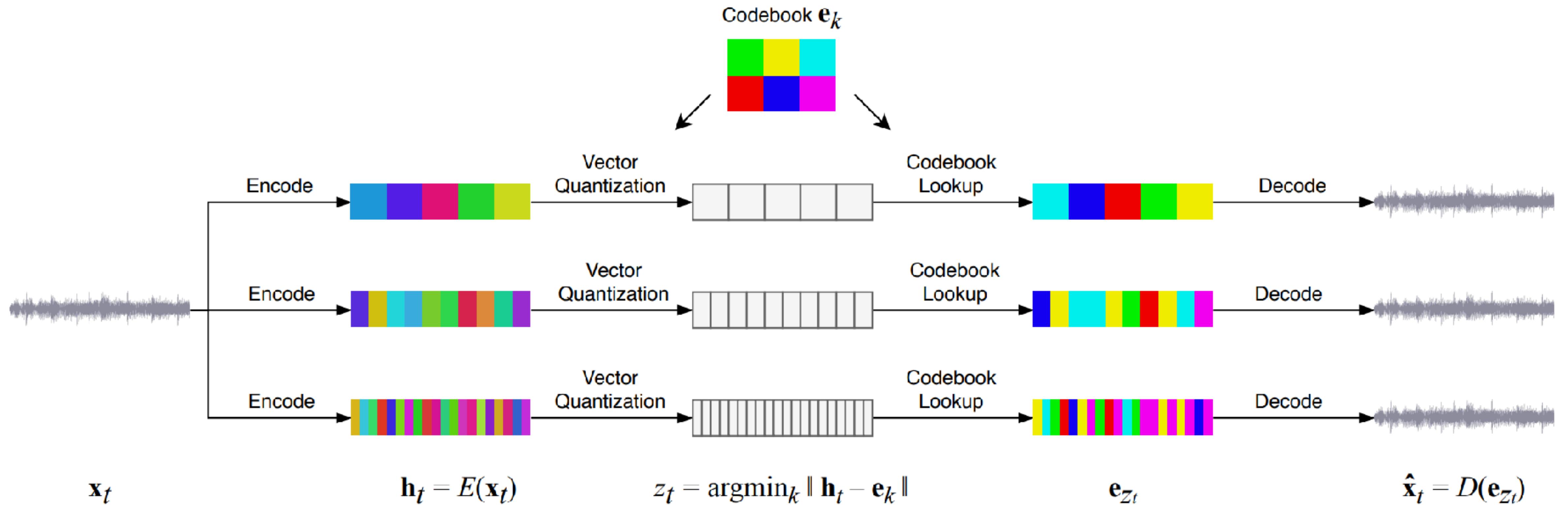
(b) Sparse Transformer (strided)

(c) Sparse Transformer (fixed)

Jukebox

- Jukebox는 OpenAI에서 Jukebox: A Generative Model for Music 논문을 통해 제안되었습니다.
- Wavenet처럼 음악 오디오 데이터를 생성할 수 있는 모델이며 long term dependency 문제를 해결하여 긴 패시지의 음악을 매우 자연스럽게 생성할 수 있습니다.
- Jukebox는 Wavenet, VQ-VAE, 그리고 Transformer 세가지 기술을 혼합하여 사용합니다.
- VQ-VAE는 vector를 압축시켜 integer에 대응 시키는 역할을 합니다. $\mathbb{R}^D \mapsto \mathbb{N}$ Wave 데이터를 frame 단위로 나누고, 각 frame을 Wavenet과 VQ-VAE로 인코딩하여 integer 값으로 변환합니다.
- 이렇게 변환된 integer 값들은 token들로 볼 수 있고 Transformer를 트레이닝하는데 이용할 수 있습니다.
- 트레이닝을 마친 뒤, Transformer로 생성한 token들은 다시 Wavenet을 이용하여 wave 데이터로 변환하여 사람이 들을 수 있는 음악으로 재현됩니다.

Jukebox



$$\begin{aligned}
 p(\mathbf{z}) &= p(\mathbf{z}^{\text{top}}, \mathbf{z}^{\text{middle}}, \mathbf{z}^{\text{bottom}}) \\
 &= p(\mathbf{z}^{\text{top}})p(\mathbf{z}^{\text{middle}}|\mathbf{z}^{\text{top}})p(\mathbf{z}^{\text{bottom}}|\mathbf{z}^{\text{middle}}, \mathbf{z}^{\text{top}})
 \end{aligned}$$