NEURAL NETWORKS – PART 4

DATA/MSML 603: Principles of Machine Learning

UNIVERSITY OF MARYLAND 1856 SCIENCE ACADEMY

Last time

- Ensemble methods
  - Bagging (bootstrap aggregating)
  - Boosting & AdaBoost (adaptive boosting)
- Recurrent Neural Networks
  - Training RNNs – backpropagation through time (BPTT)
  - LSTM & GRU
  - Bidirectional RNNs

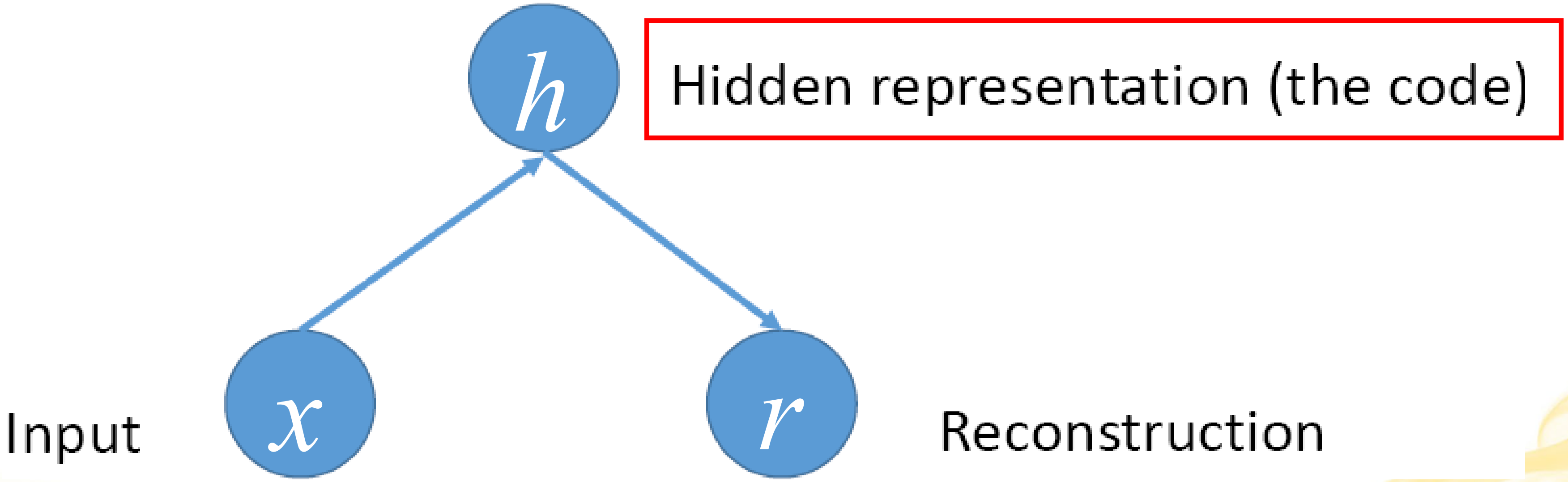
Autoencoder

Autoencoder

- **Unsupervised learning** technique that utilizes neural networks for **representation learning**
- **A special encoder-decoder**
  - Neural networks trained to attempt to copy its input to its output
  - Expecting the output is the same as the input
- Contains two parts
  - **Encoder:** map the input to a **hidden representation** **(called latent space representation)**
  - **Decoder:** map the **hidden representation** to the output

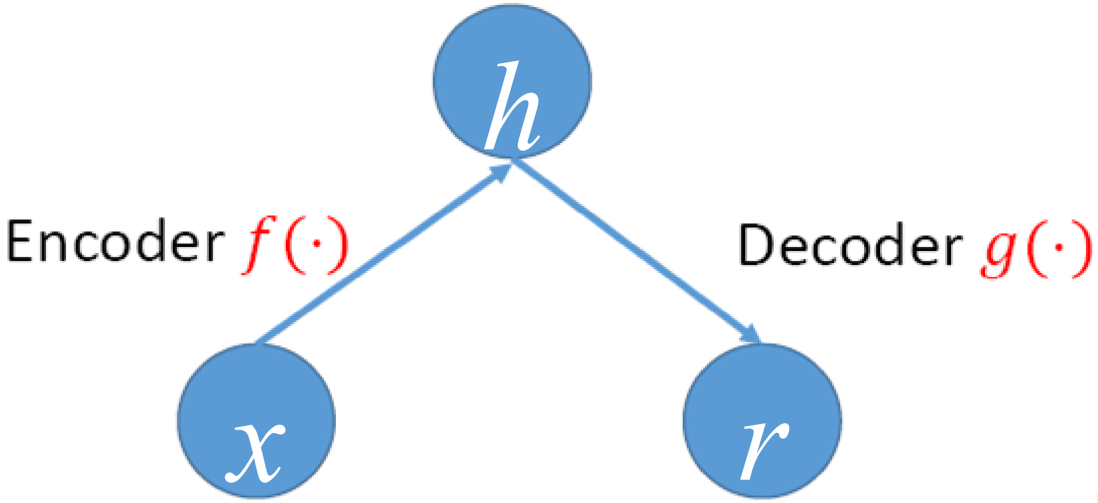
Autoencoder

- Discovers latent variables and effective representation of the input data (without label)
  - Learns which latent variables are useful for accurately reconstructing the original data (most essential information in the original input)



Autoencoder

- Encoder - transforms input data into a smaller dimensional representation (called the code)
  - Learns mapping f
- Decoder - rebuilds initial input from the code
  - Learns mapping g



$$h = f(x), r = g(h) = g(f(x))$$

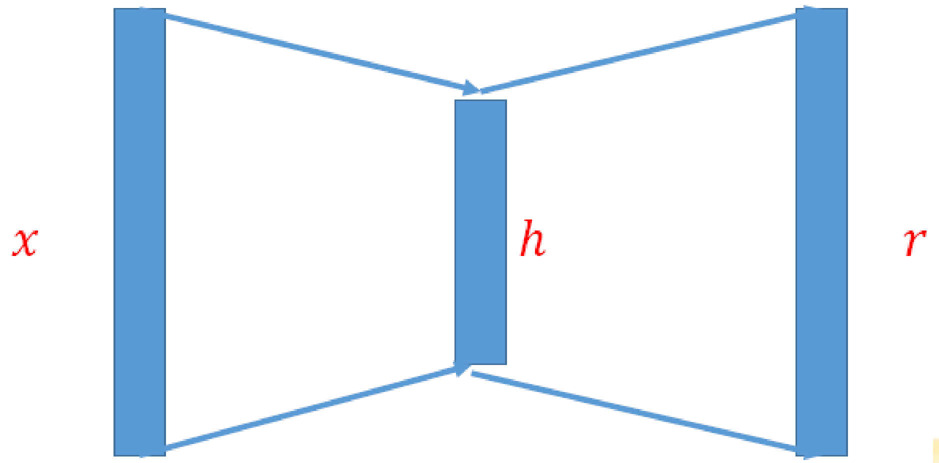
Why Reconstruct Input

- Not really interested in copying
- Interesting case: NOT able to copy exactly but strive to do so by looking for good representation of input (representation learning)
- Autoencoder forced to select which aspects to preserve and thus hopefully can learn useful properties of the data
  - Helps learn and construct important features (feature engineering/selection)
- Historical note: goes back to (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1994).

Undercomplete Autoencoder

- Constrain the code to have smaller dimension than the input
  - **Dimensionality reduction** - learns a compressed representation of input data
- Training: minimize a loss function (typically mean squared error (MSE))
  - If the encoder cannot learn effective representation, it will incur larger costs

$$L(x, r) = L(x, g(f(x)))$$

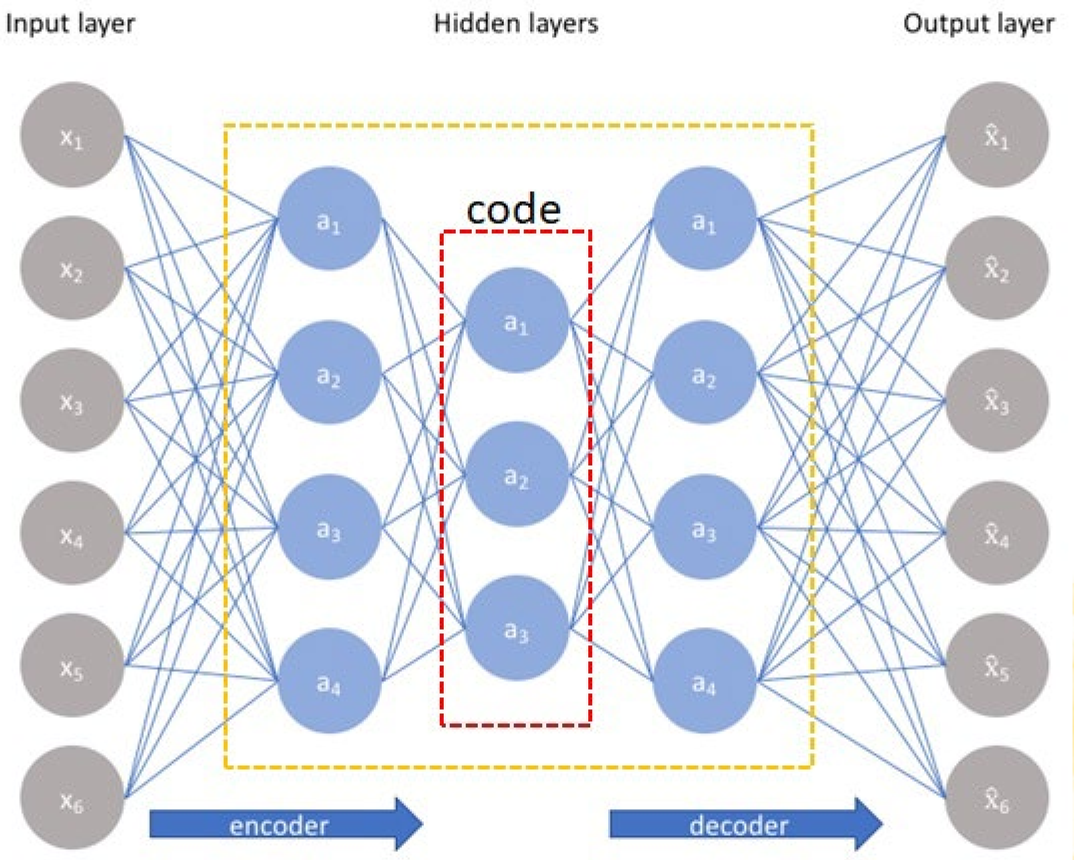


Undercomplete Autoencoder

- Constrain the code to have smaller dimension than the input
- Training: minimize a loss function

$$L(x,r) = L(x, g(f(x)))$$

- Special case: f, g linear, L mean squared error
- Reduces to Principal Component Analysis (PCA)

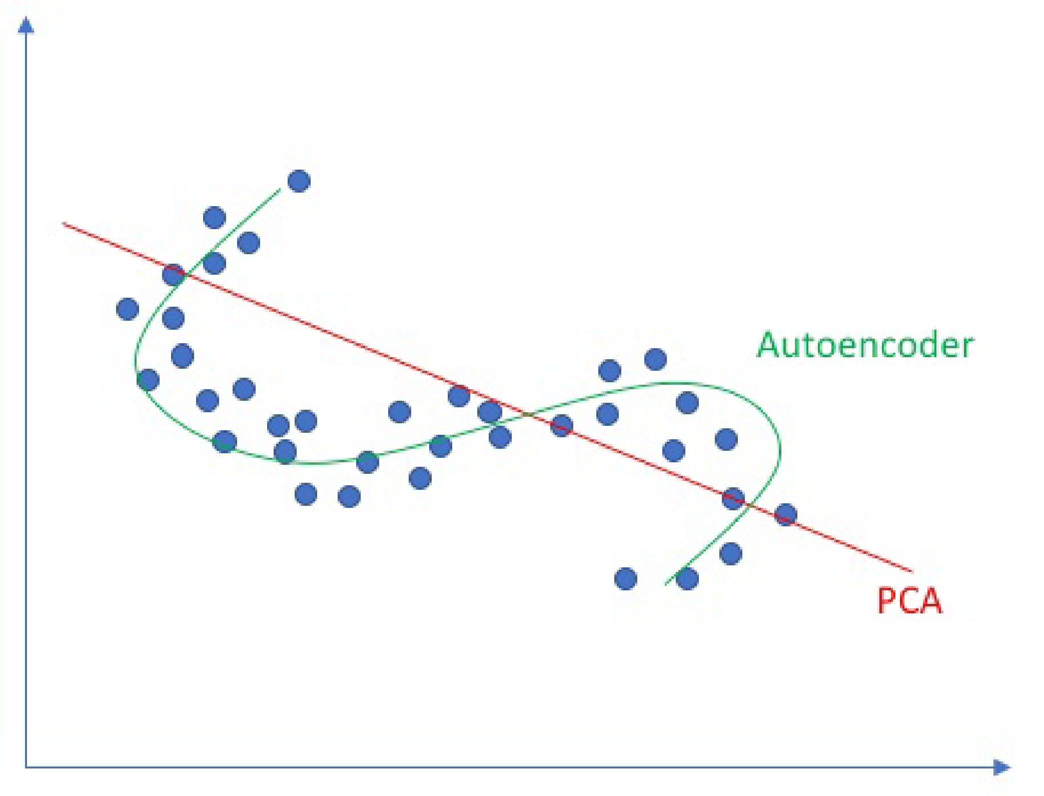


[source: www.jeremyjordan.me]

Undercomplete Autoencoder

- Nonlinear generalization of PCA
  - Nonlinear activation functions
  - Learns nonlinear manifolds (i.e., continuous nonintersecting surface)

Linear vs nonlinear dimensionality reduction



[source: www.jeremyjordan.me]

Autoencoder with a Single Hidden Layer

- Single layer with linear functions

$$\hat{x} = g(f(\mathbf{x})) = \mathbf{V}(\mathbf{W}\mathbf{x})$$

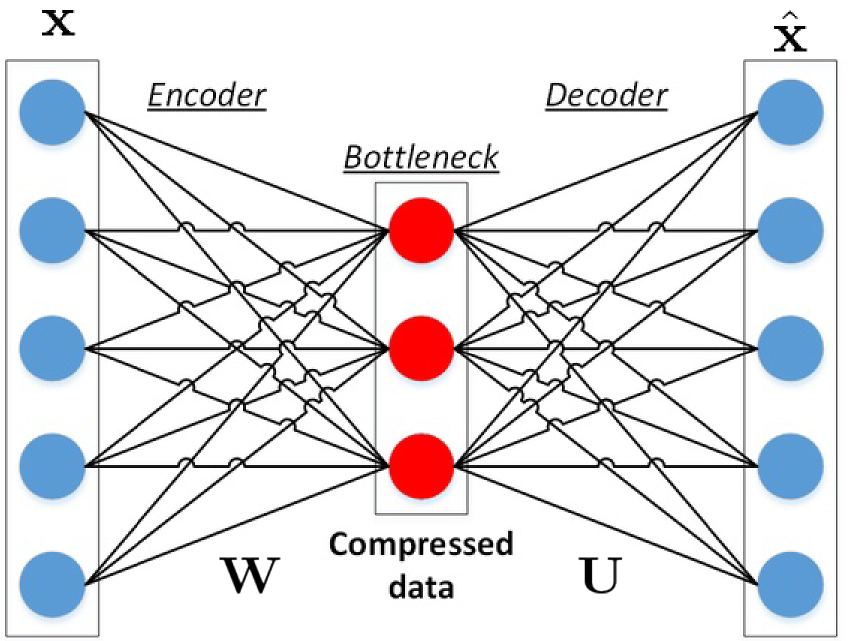
- Let $\mathbf{X} = \begin{bmatrix} \mathbf{x}^1 & \mathbf{x}^2 & \cdots & \mathbf{x}^n \end{bmatrix}$ be $d \times n$ matrix containing training data
- Wish to minimize the mean squared error

$$\sum_{i=1}^{n} \|\mathbf{x}^i - \hat{\mathbf{x}}^i\|_2^2 = \|\mathbf{X} - \mathbf{VWX}\|_F^2$$

- Solution given by Singular Value Decomposition

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

  - Columns of matrix $\mathbf{U}$ consist of left singular vectors
  - Columns of matrix $\mathbf{V}$ consist of right singular vectors



[source: medium.com

Autoencoder with a Single Hidden Layer

- Singular Value Decomposition: $\mathrm{rank}(\mathbf{X}) = r$

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

- Properties of left and right singular vectors

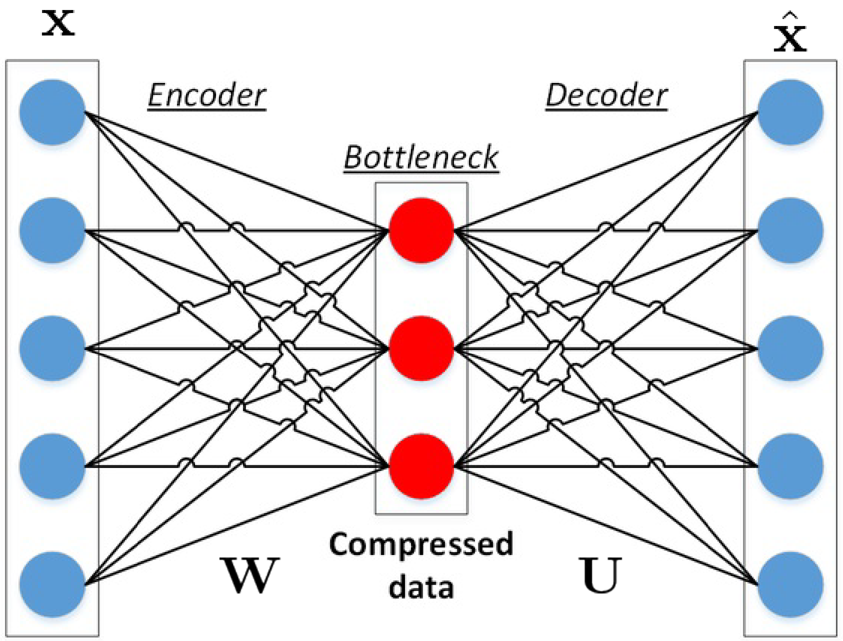
$$\mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r} = \mathbf{V}^T\mathbf{V}$$

- Pseudo-inverse of $\mathbf{U} : \mathbf{U}^\dagger = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T = \mathbf{U}^T$

- Suppose we choose $\mathbf{W} = \mathbf{U}^T$

$$\hat{\mathbf{X}} = \mathbf{U}\mathbf{U}^T\mathbf{X} = \mathbf{U}\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{U}\Sigma\mathbf{V}^T = \mathbf{X}$$

- When the dimension of compressed data is smaller than r, use truncated singular value decomposition

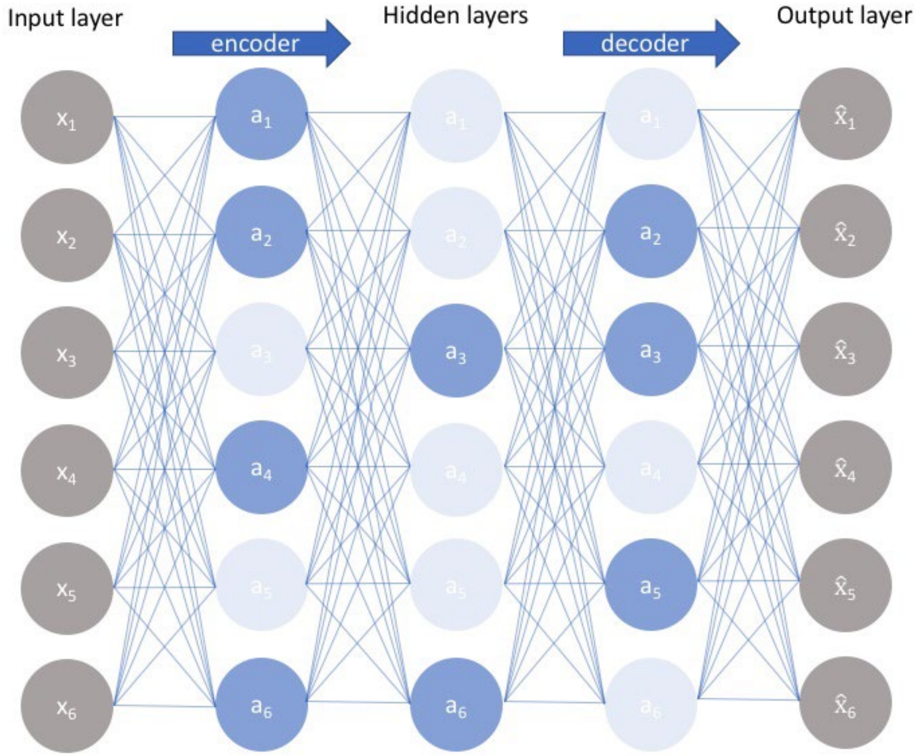


Denoising Autoencoder

- Traditional autoencoder: encourage to learn $g(f(\cdot))$ to be identity function
  - Produces output equal to input
- Denoising : Add noise to the input data while maintaining the original data as target output
  - Helps avoid overfitting by making it harder to memorize input data (rather than learning important features and effective representation)
  - Improves generalization
  - Can be used to remove noise in images

Sparse Autoencoder

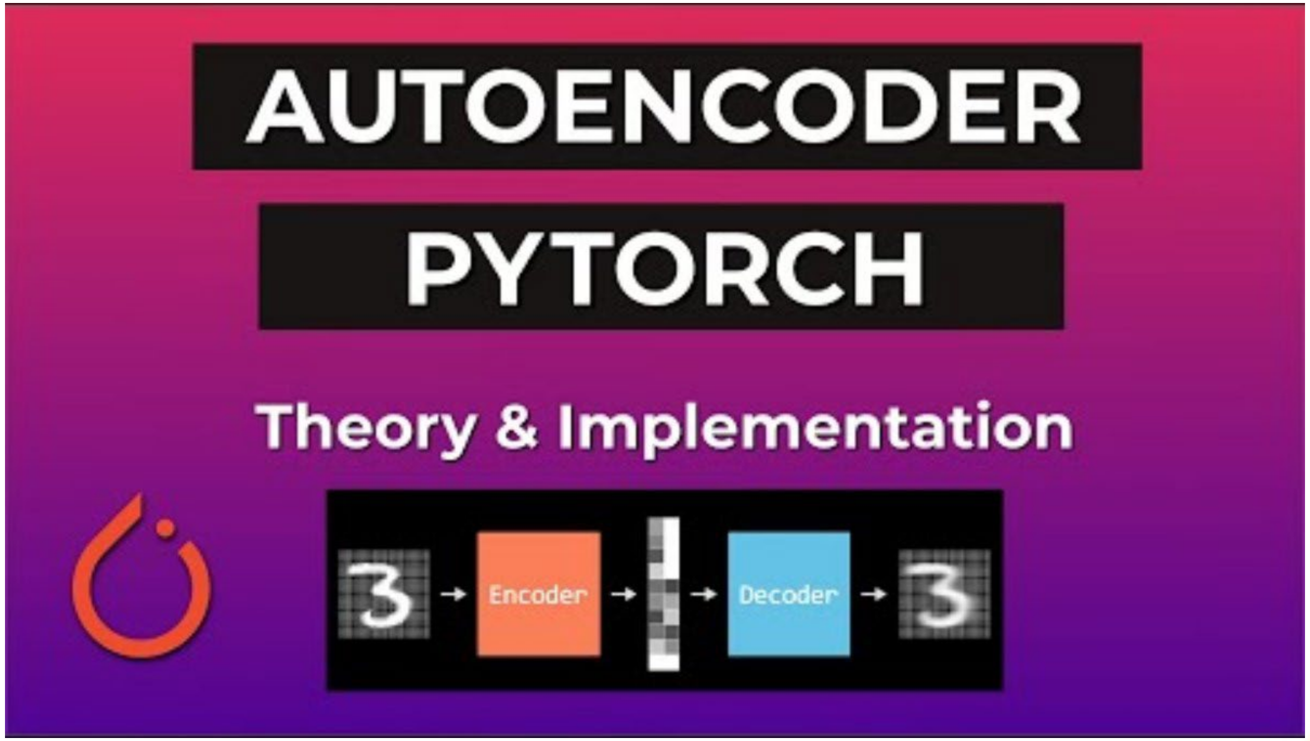
- Instead of requiring reduction in the number of nodes in hidden layers, use a loss function that activates only a small number of nodes
  - Similar to regularizing weights, but here regularize activations



[source: www.jeremyjordan.me]

Autoencoder

- [Building autoencoder in PyTorch]



Generative Adversarial Networks (GANs)

Generative Adversarial Networks

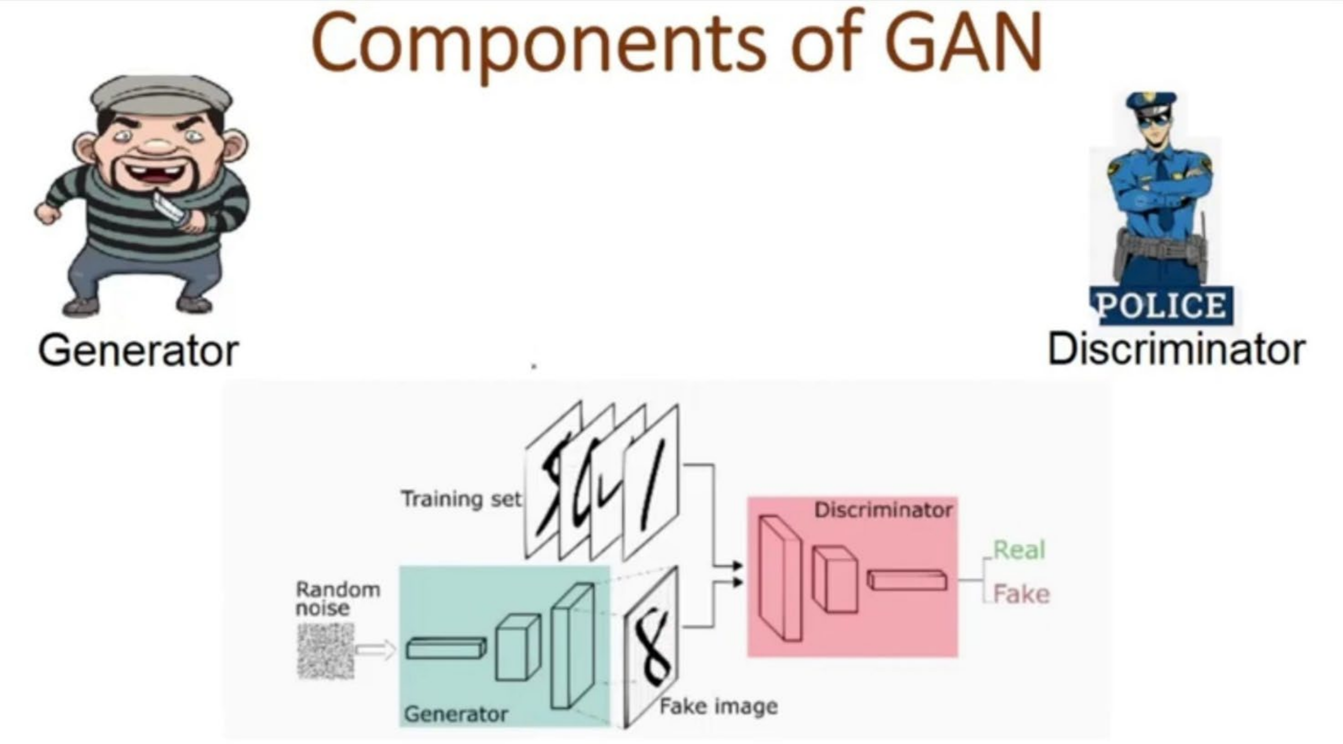
- Machine learning model for data generation
  - Creates generative model of a base dataset by using an adversarial game between two players – generator and discriminator
- Motivation
  - Data synthesis
  - Find a mapping between spaces
  - Image in-painting
- Approach
  - Collect a lot of data, use it to train a model to generate similar data from scratch
- Intuition
  - Number of parameters of the model << amount of data

Generative Adversarial Networks

- Generator takes (Gaussian) noise as input and produces an output, which is a "generated" sample similar to the base data
- Discriminator is (typically) a probabilistic classifier, such as logistic regression, whose job is to distinguish real samples from the base dataset and the generated sample
- Adversarial game (minimax learning problem)
  - Generator tries to fool the discriminator by creating samples that are as realistic as possible
  - Discriminator tries to identify the fake samples irrespective of how well the generator tries to fool it

Generative Adversarial Networks

- Nash equilibrium of minimax game provides the final trained model
  - Neither Generator nor Discriminator can improve its performance



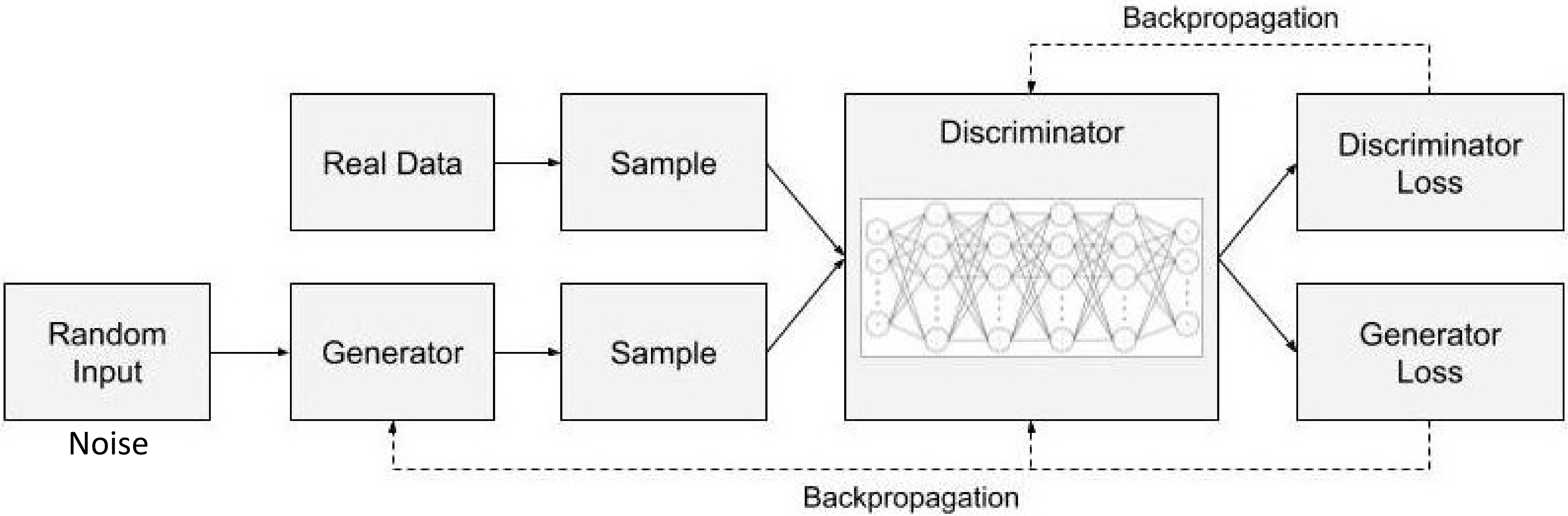
[source: www.analyticsvidhya.com]

Generative Adversarial Networks

- Discriminative model: directly estimate the conditional probability $\mathbf{P}(y|\mathbf{x})$ of label y given the feature values $\mathbf{x}$
  - Example: logistic regression
- Generative model: estimate the joint probability $\mathbf{P}(\mathbf{x}, y)$, which is a generative probability of a data instance
  - Recall that conditional probability of label y given $\mathbf{x}$ can be estimated from the joint probability

$$\mathbf{P}(y|\mathbf{x}) = \frac{\mathbf{P}(\mathbf{x}, y)}{\mathbf{P}(\mathbf{x})} = \frac{\mathbf{P}(\mathbf{x}, y)}{\sum_z \mathbf{P}(\mathbf{x}, z)}$$

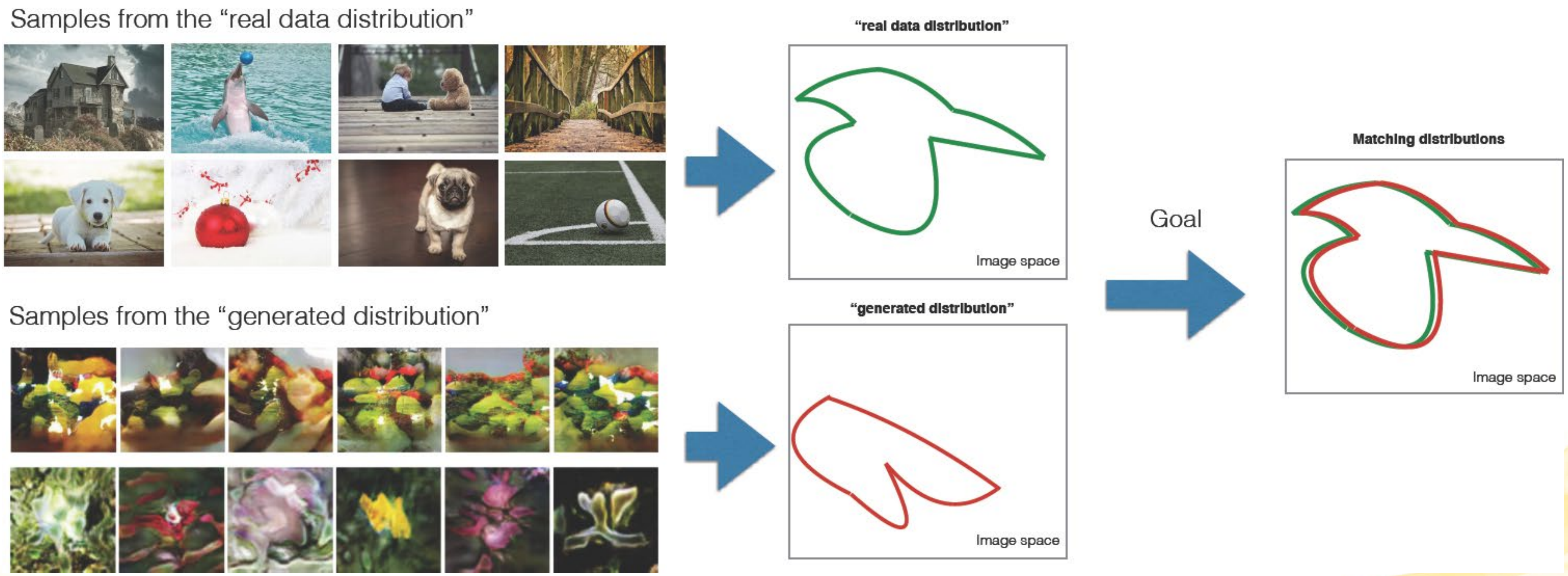
Generative Adversarial Networks



[source: www.ml-science.com]

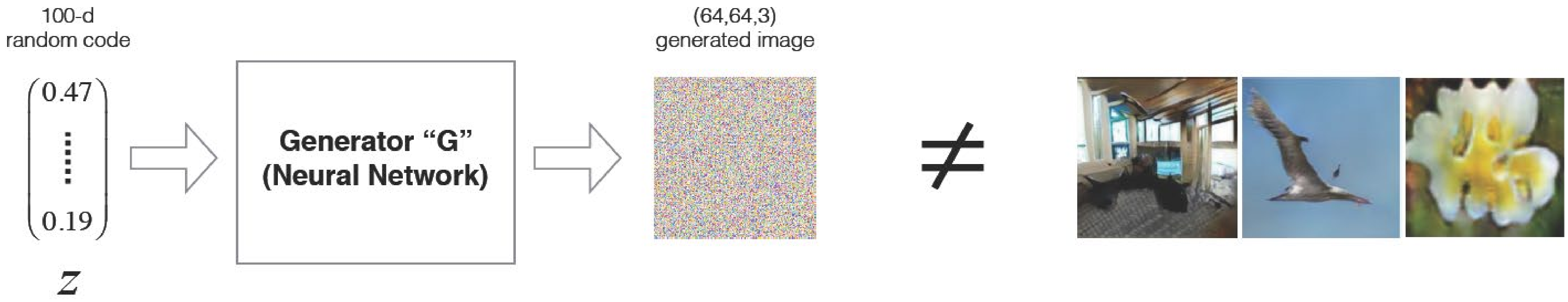
Generative Adversarial Networks (2/3)

Probability distribution

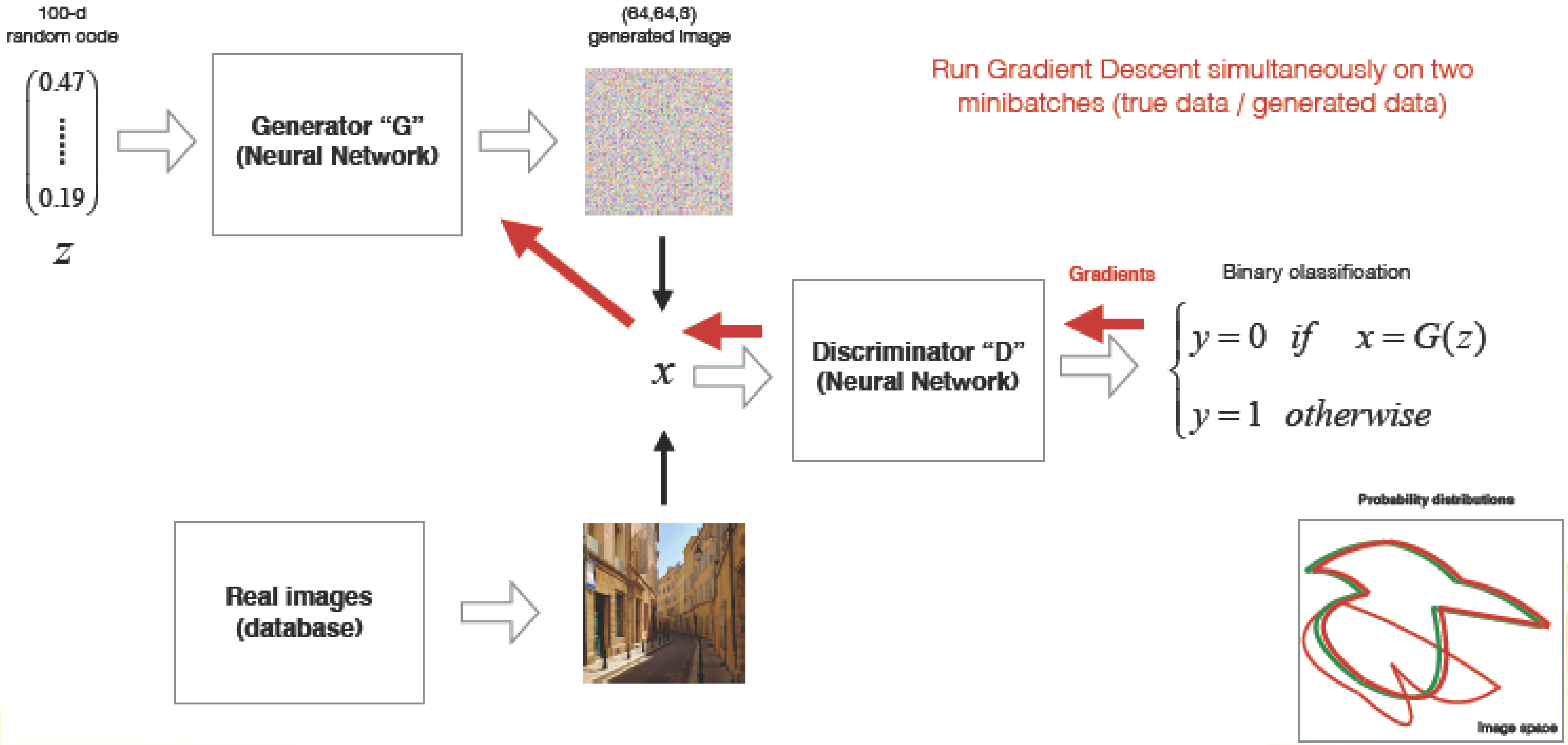


Generative Adversarial Networks (3/3)

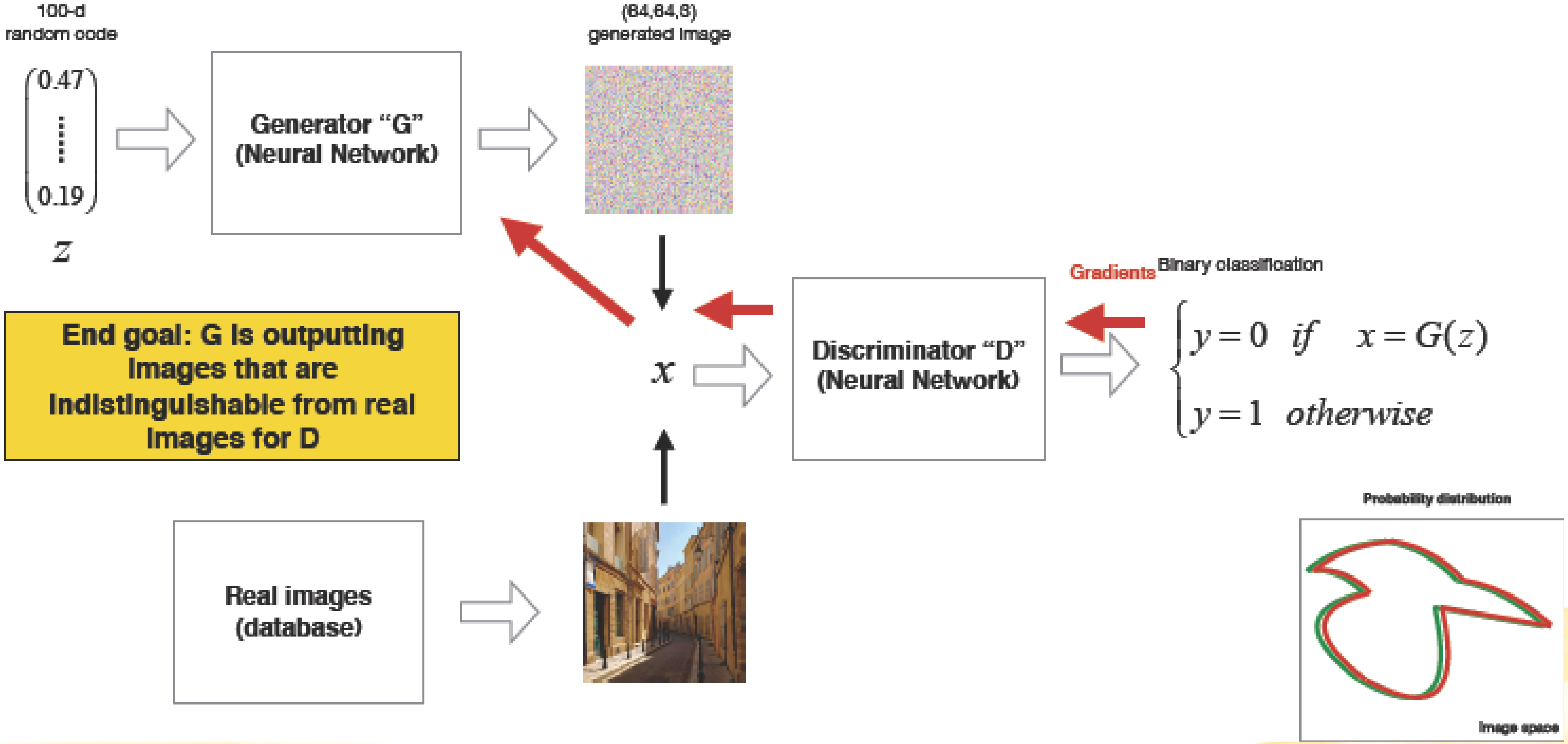
How can we train G to generate images from the true data distribution?



Generator/Discriminator Game (1/2)



Generator/Discriminator Game (2/2)



GANs Formulation

$$\min_G \max_D V(D,G)$$

- **It is formulated as a minimax 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)

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

  - The **Nash equilibrium** of this particular game is achieved at:

$$P_{data}(x) = P_{gen}(x) \;\; \forall x$$

$$D(x) = \frac{1}{2} \;\; \forall x$$

Variations

Table 1: Generator and discriminator loss functions. The main difference whether the discriminator outputs a probability (MM GAN, NS GAN, DRAGAN) or its output is unbounded (WGAN, WGAN GP, LS GAN, BEGAN), whether the gradient penalty is present (WGAN GP, DRAGAN) and where is it evaluated. We chose those models based on their popularity.

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\|\nabla D(\alpha x + (1-\alpha\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d + \mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \mathrm{AE}(x)\|\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$

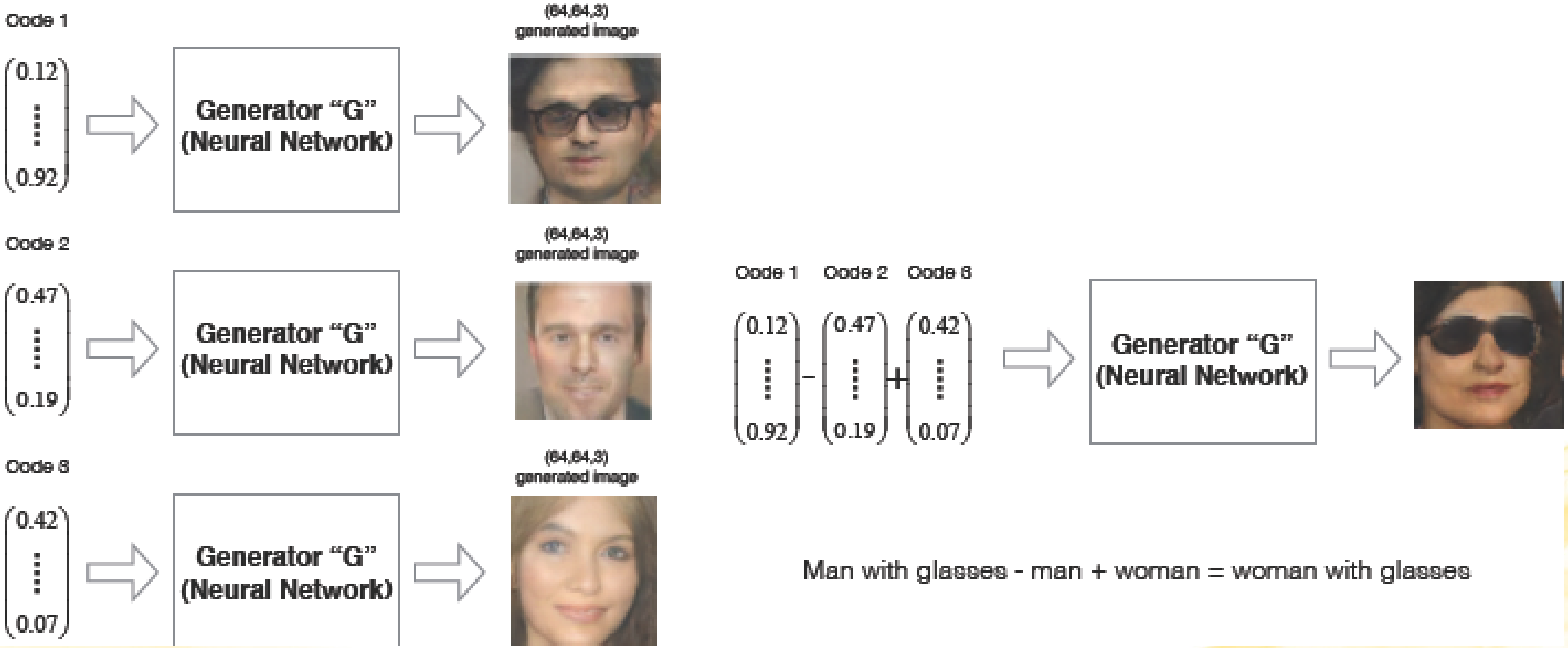
[Lucic, Kurach et al. (2018): Are GANs Created Equal? A Large-Scale Study]

Generative Adversarial Networks

[Building GAN from scratch in PyTorch]

Nice Results

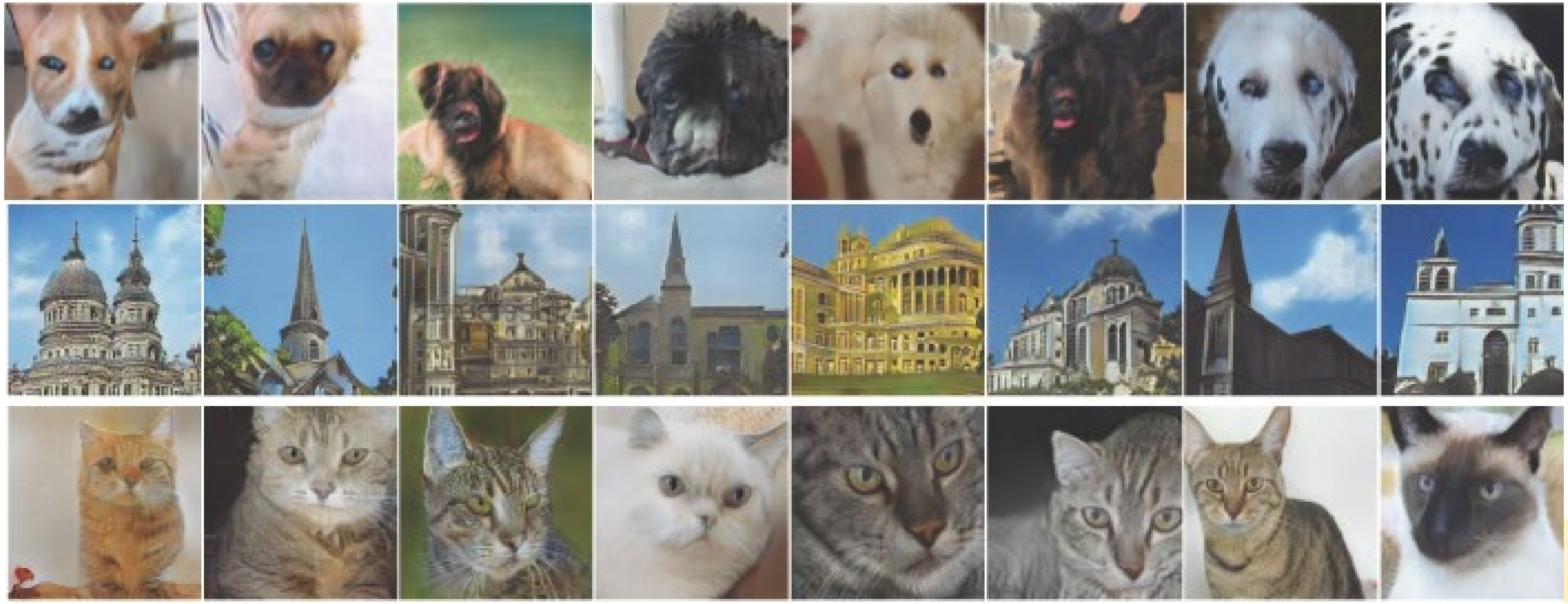
Operation on codes



Nice Results (2/3)

Image generation

Samples from the "generated distribution"



[Zhang et al. (2017): StackGAN++]

Nice Results (3/3)

Image generation

Image source: Lin et. Al. (2017): Unsupervised Image-to-Image Translation Networks

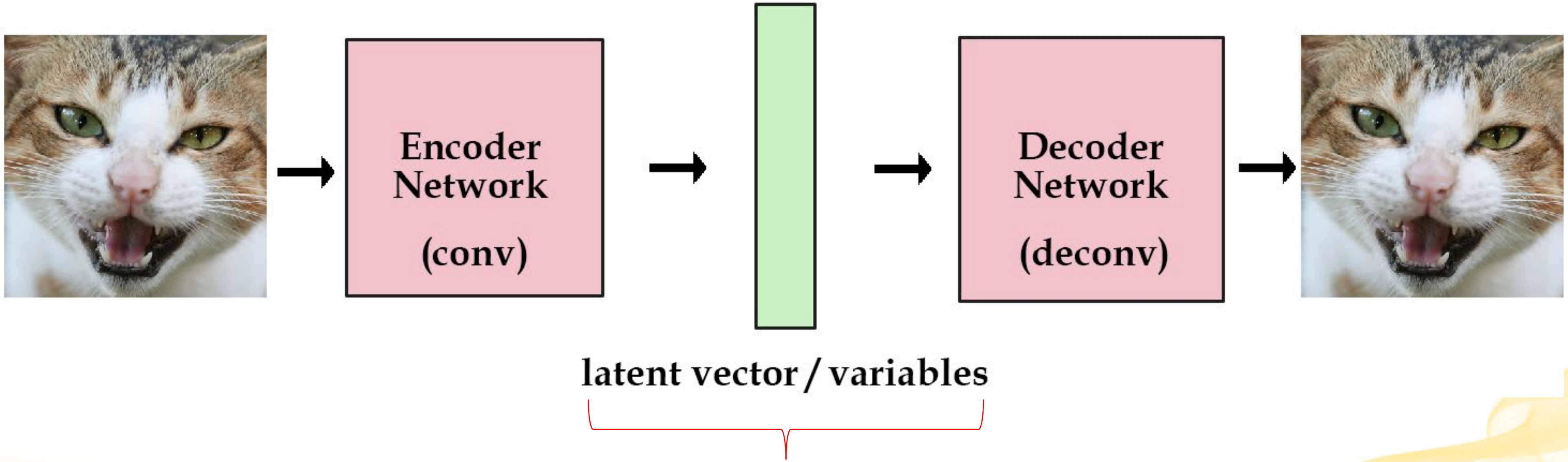
Figure 3: Street scene image translation results. For each pair, left is input and right is the translated image.

age-to-Image Translation Networks]

Variational Autoencoder

Variational Autoencoder (1/4)

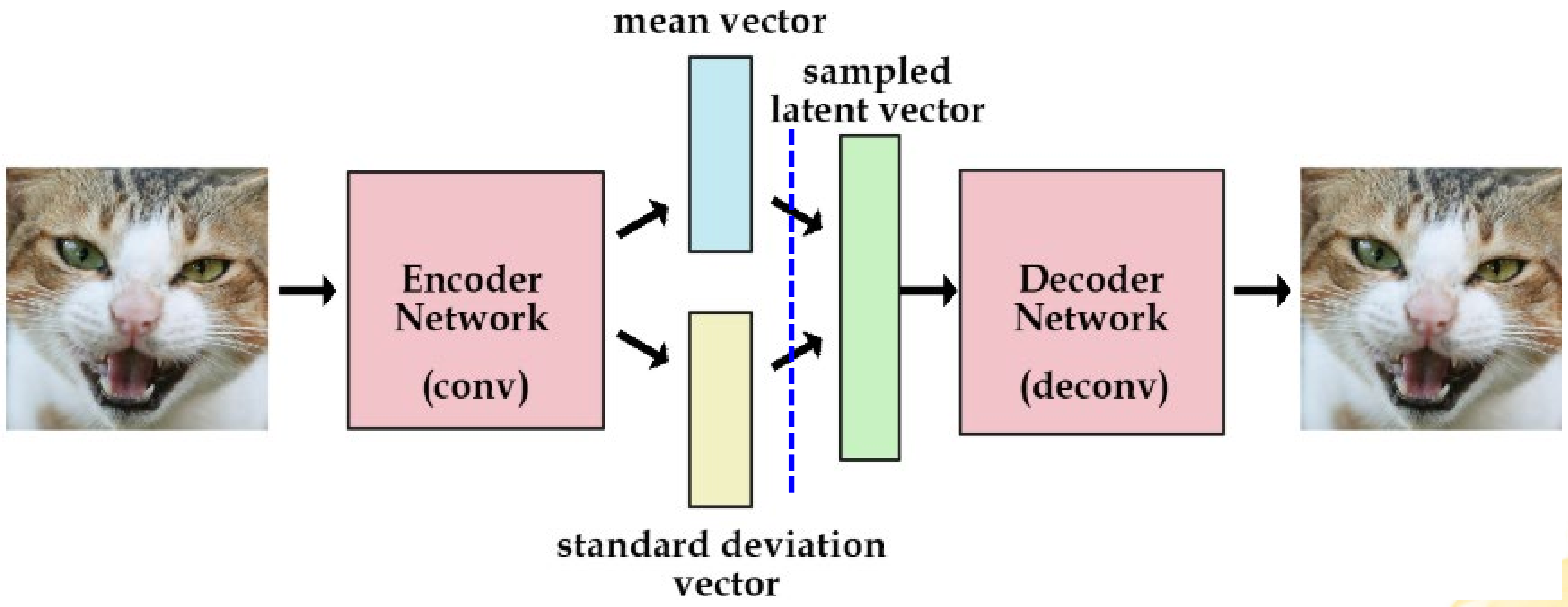
- Traditional autoencoder – tries to separate codes far away from each other for reconstruction
  - Discrete points in latent space



single point in latent space

Variational Autoencoder (3/4)

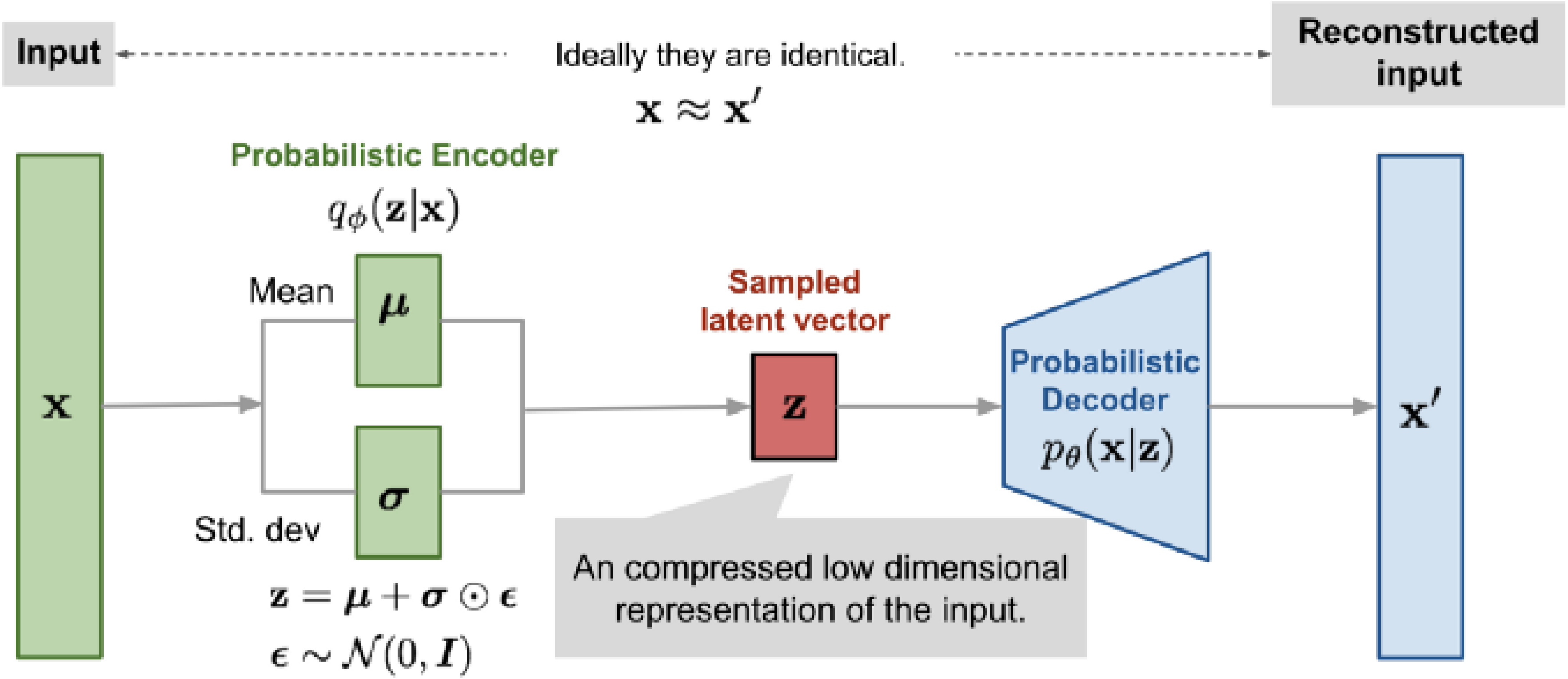
- Variational autoencoder



Encoder output = distribution

Decoder input = sample from distribution

Variational Autoencoder (4/4)



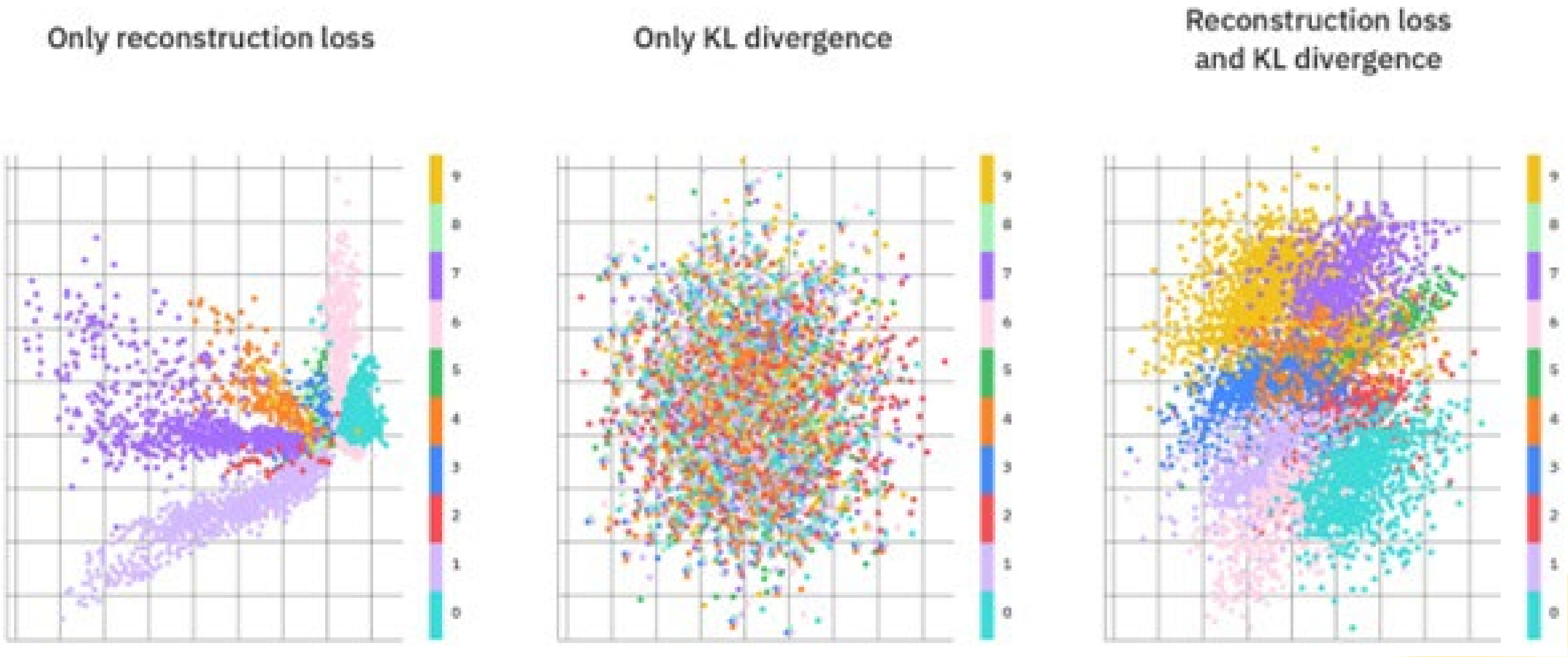
- VAE learns to model the underlying probability distribution of input data

Variational Autoencoder: Differences Between VAEs and autoencoders

- Variational Autoencoder
  - Latent space: continuous and smooth
    - Allows interpolation and transitions of representations
  - Loss: reconstruction loss + (Kullback-Leibler) KL divergence
    - Regularizes latent space, forcing it to following a prior distribution (usually Gaussian distribution)

- Autoencoder
  - Latent space: discrete with gaps in representations
  - Loss: primarily reconstruction loss for accurate reproduction of input

Variational Autoencoder: Loss + KL divergence



[latent space for handwritten digits of 0-9 from the MNIST dataset]

Variational Autoencoder: Results

1st epoch

9th epoch

Training data

Variational Autoencoder

[Building variational autoencoder from scratch in PyTorch]

SCIENCE ACADEMY

Transformer Networks

High Level View…

- Essentially large encoder/decoder blocks that process data
- Key features
  - Input embedding
  - Positional encoding
  - Self-attention
  - Multi-head attention

Transformer Networks

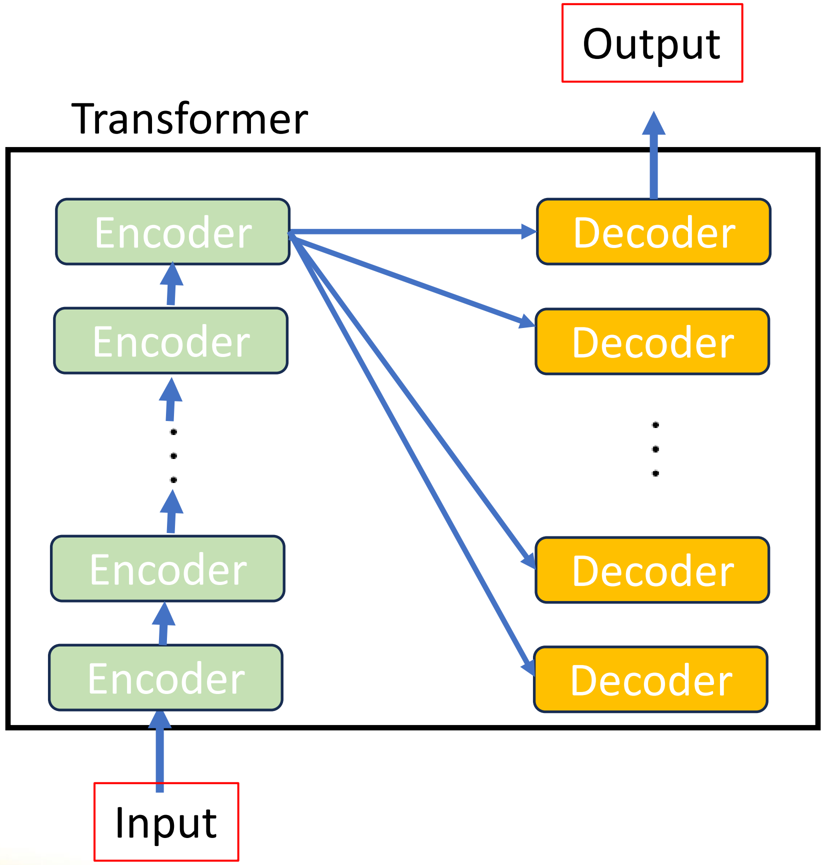
[Building a Transformer with PyTorch]
[Building your own transformer rom scratch using PyTorch]

- Applications
  - Natural language processing (NLP)
    - Language translation, speech recognition, speech translation
  - Computer vision
  - Time series forecasting
- Examples: Generative Pre-trained Transformer (GPT-3), Bidirectional Encoder Representations from Transformers (BERT), Robustly Optimized BERT Pretraining Approach (RoBERTa)

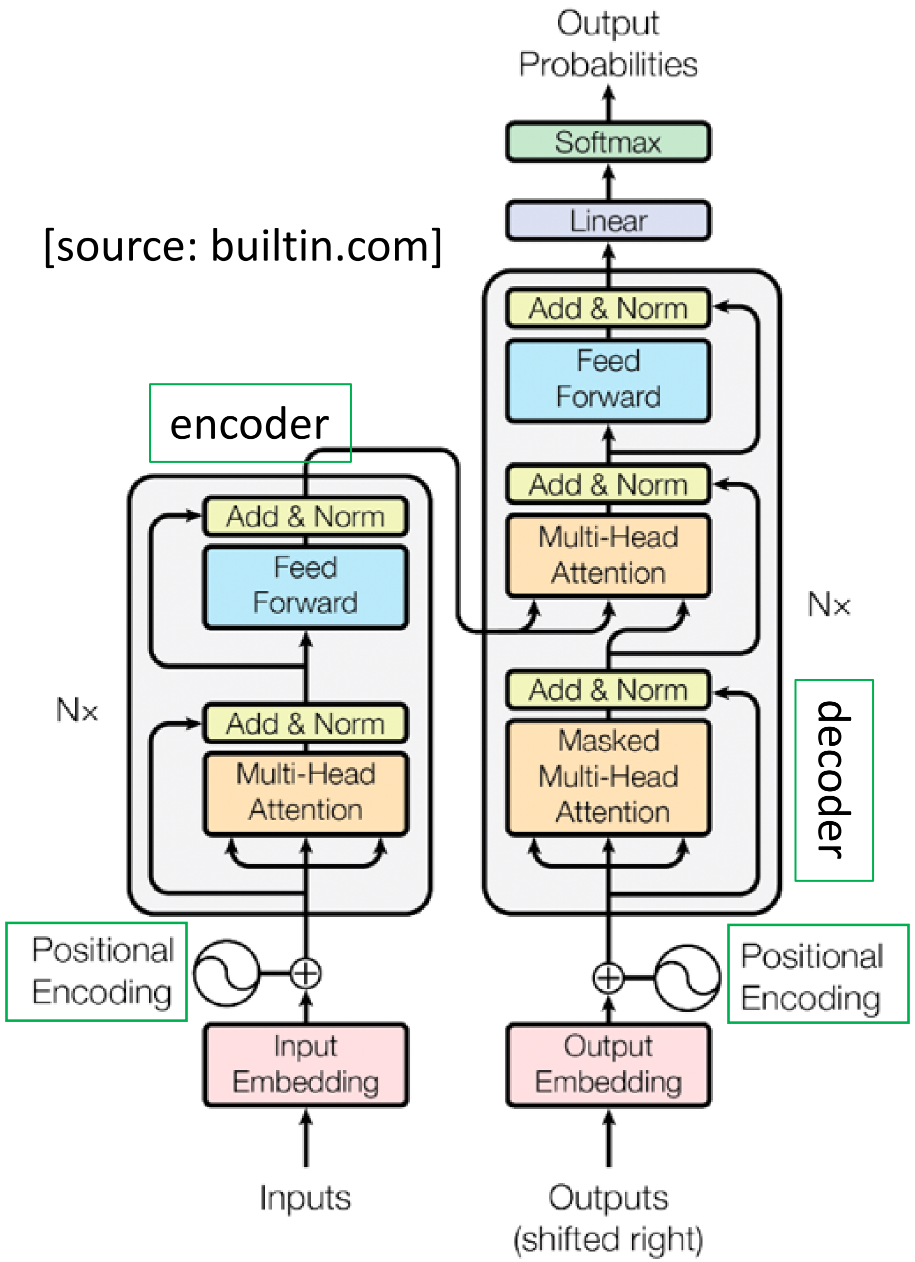
Key Advantages

- Semi-supervised learning
  - It uses a large set of “unlabeled” data for pre-training (self-supervised) and then fine-tune on a smaller “labeled” dataset for specific tasks
- Lends itself to parallel processing
  - Sequential nature of data handled using “positional encoding”
  - Speeds up training

Architecture of Transformer Model

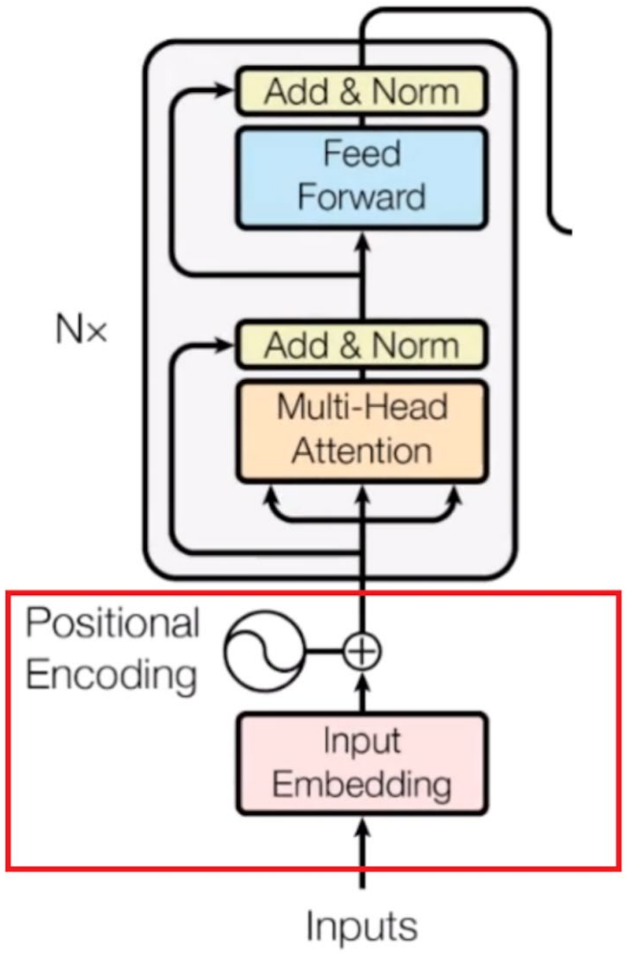


[source: builtin.com]



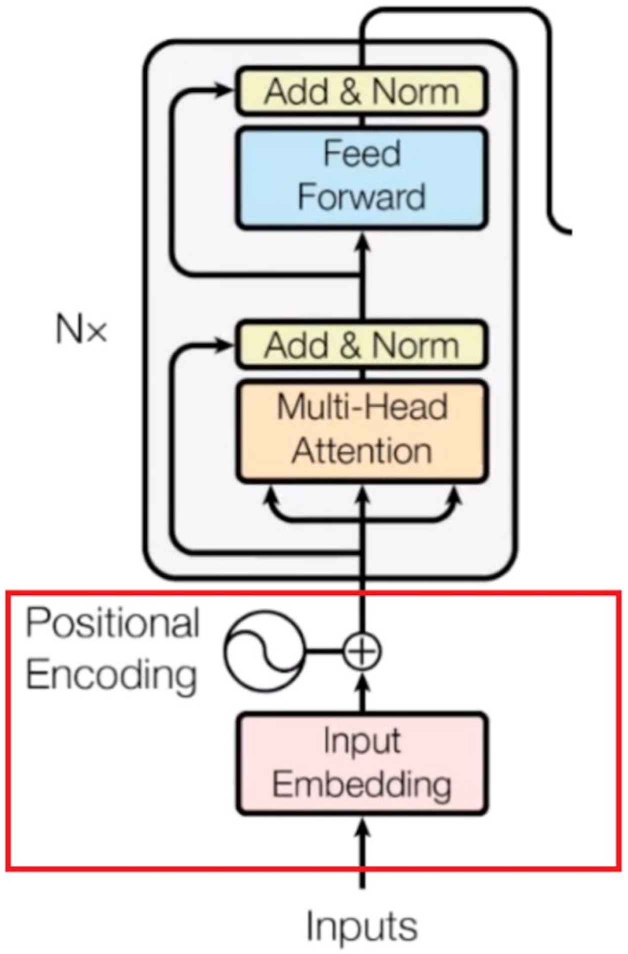
Input Embedding & Positional Embedding

- Tokenization: Input text split into individual tokens (e.g., words)
- Input embedding: A vector corresponding to the token in the embedding matrix (size 512)
  - Words with similar meaning have similar vectors
  - Learned during training
- Positional embedding: positional encoding vector added to the input embedding to indicate its position in the input sequence
  - Eliminates the need for recurrence
  - Encoding of sinusoidal functions (sine and cosine) allows the network to determine if words are close



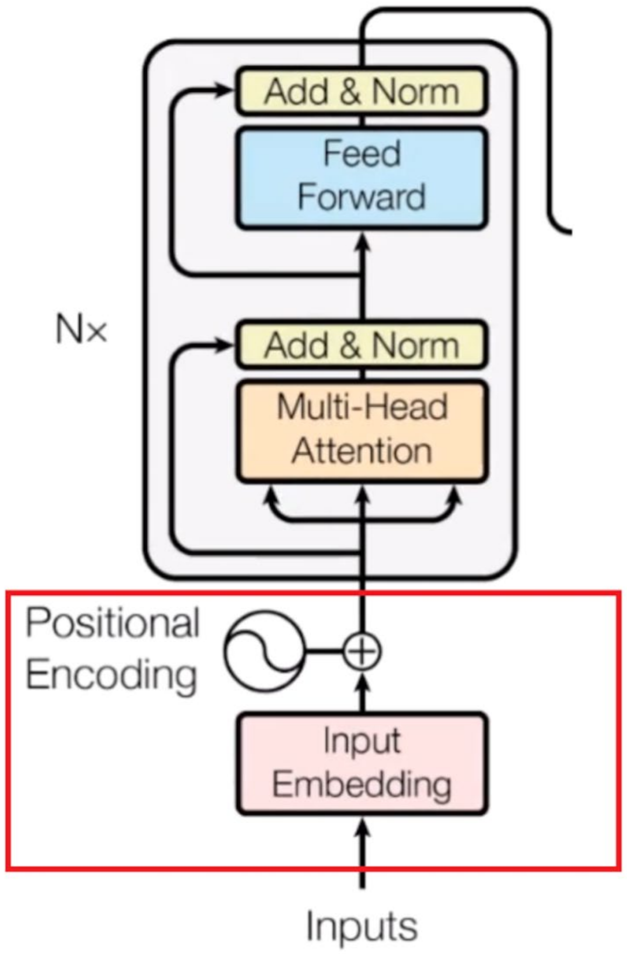
Encoder

- Transforms input tokens (e.g., words in a sentence) into contextualized representation
  - Captures the context of each token in relation to entire sentence
- Typically consists of multiple encoder layers
  - 6 in the original paper (“Attention Is All Your Need” by Vaswani et al, NIPS 2017)



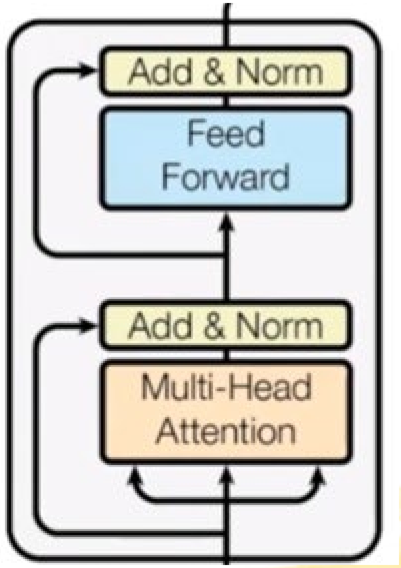
Encoder

- Encoder layer – encapsulates the information in the “entire” input sequence in a continuous, abstract representation with the help of two main modules
  - Multi-headed attention mechanism
  - Fully connected network



Encoder Layer

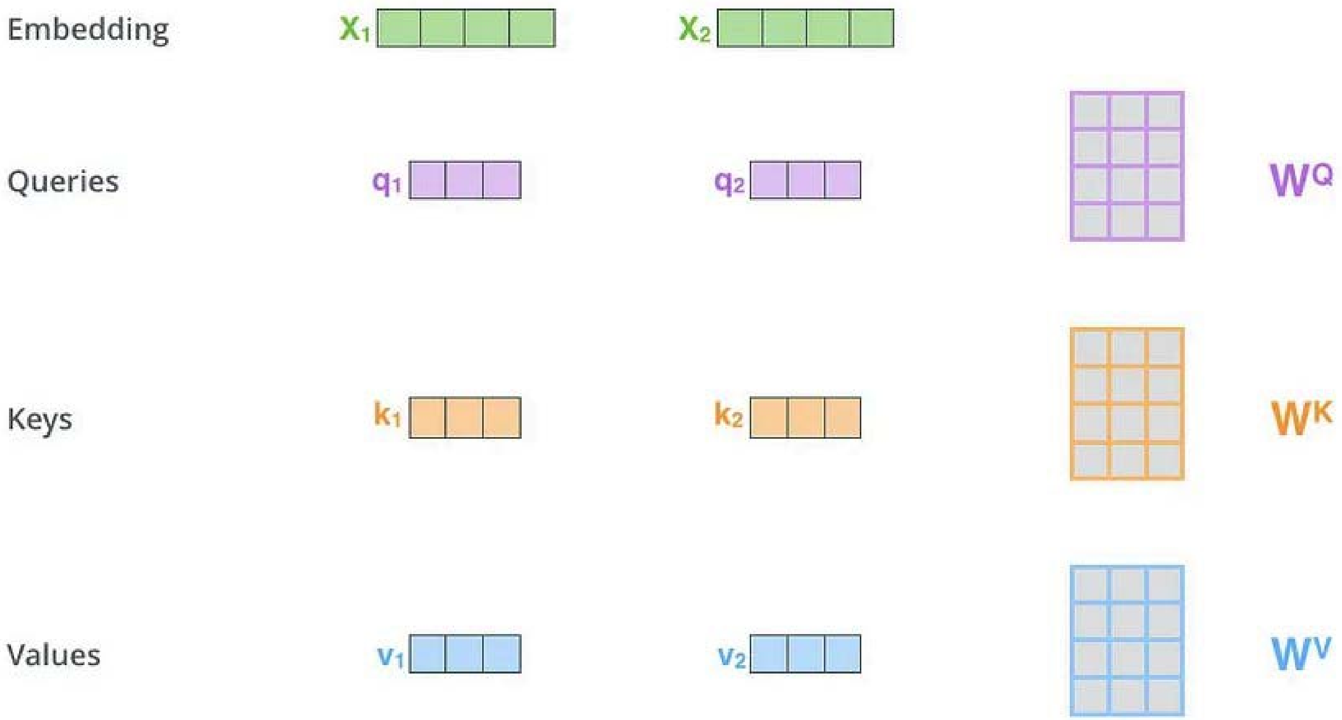
- Self-attention
  - Multi-head attention utilizes self-attention that relates each word in the input text with all other words
    - Captured by attention scores
  - Allows the encoder to focus on different parts of the input text as it processes each token
  - Attention scores computed using query vector, key vector, and value vector
    - Created by multiplying the embedding by three matrices trained during the training process



Encoder Layer

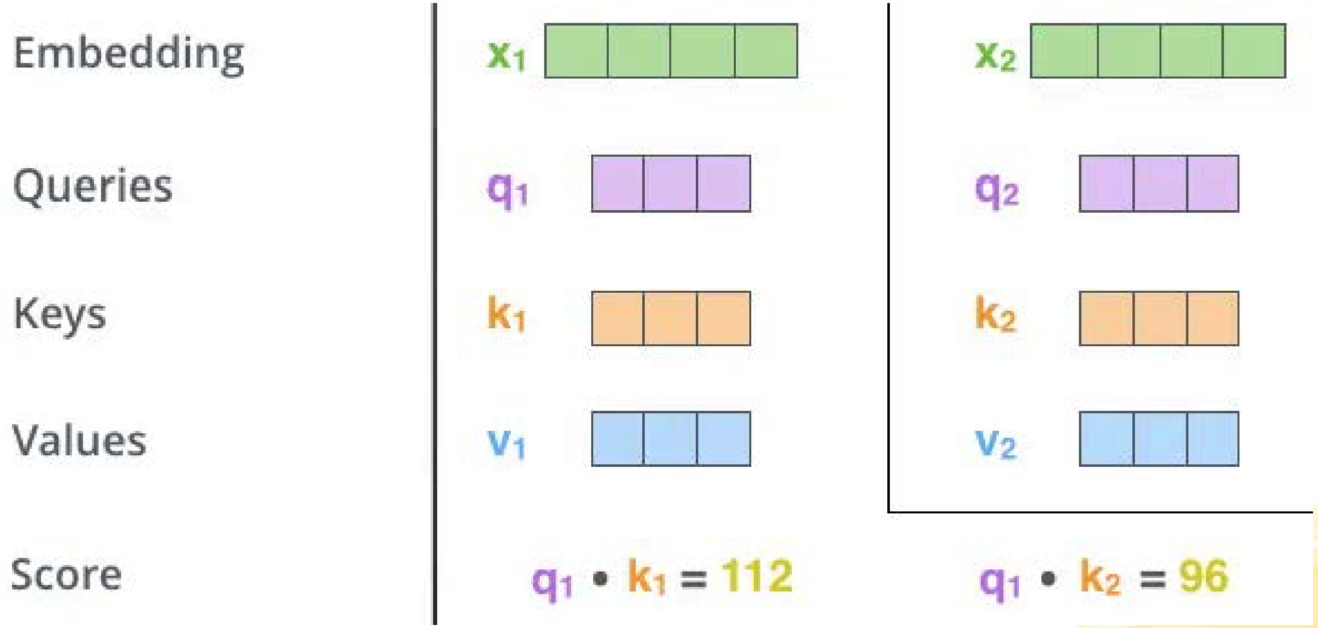
- Query vector, key vector, and value vector created by multiplying the embedding by three matrices trained during the training process
  - Have smaller dimensionality of 64 (as opposed to 512 for embedding)

[source: towardsdatascience.com]



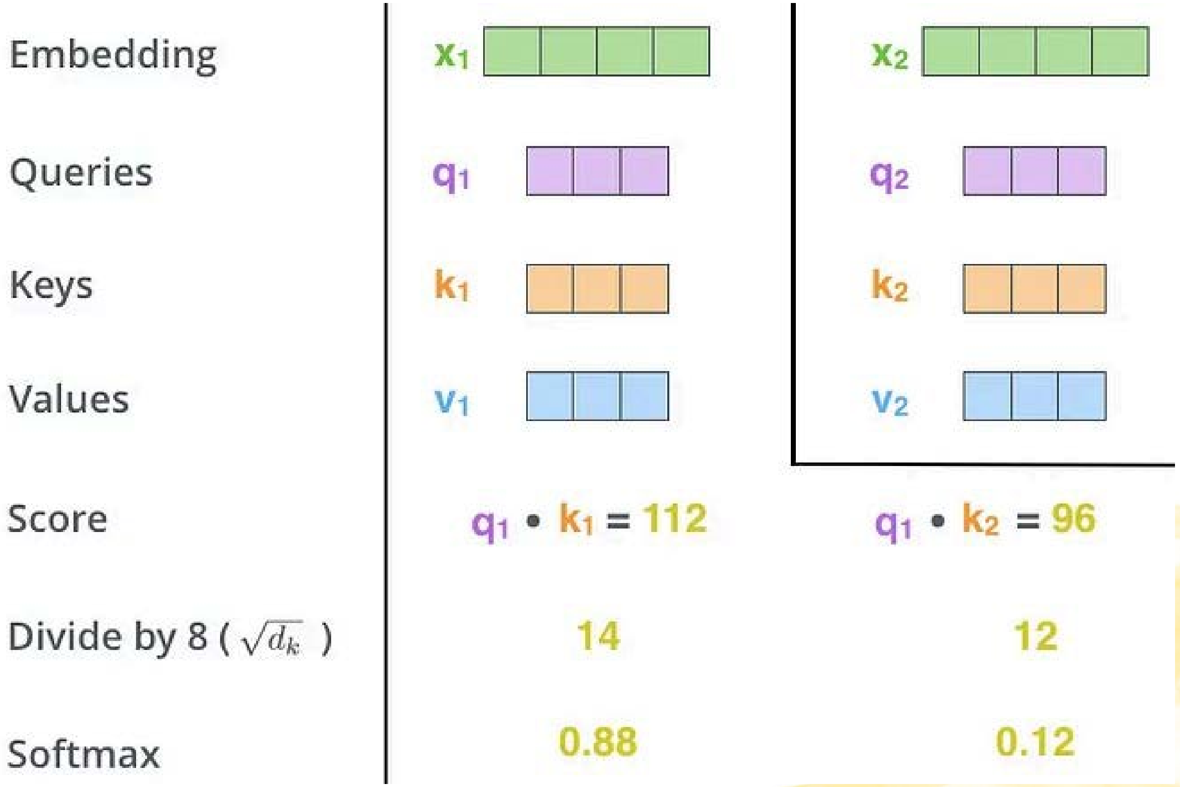
Encoder Layer

- For each embedding, compute the scores for all the words in input text
  - Determines how much focus to assign to other parts of input text
  - Score of a token given by the dot product of the query vector with the key vector of the word being scored



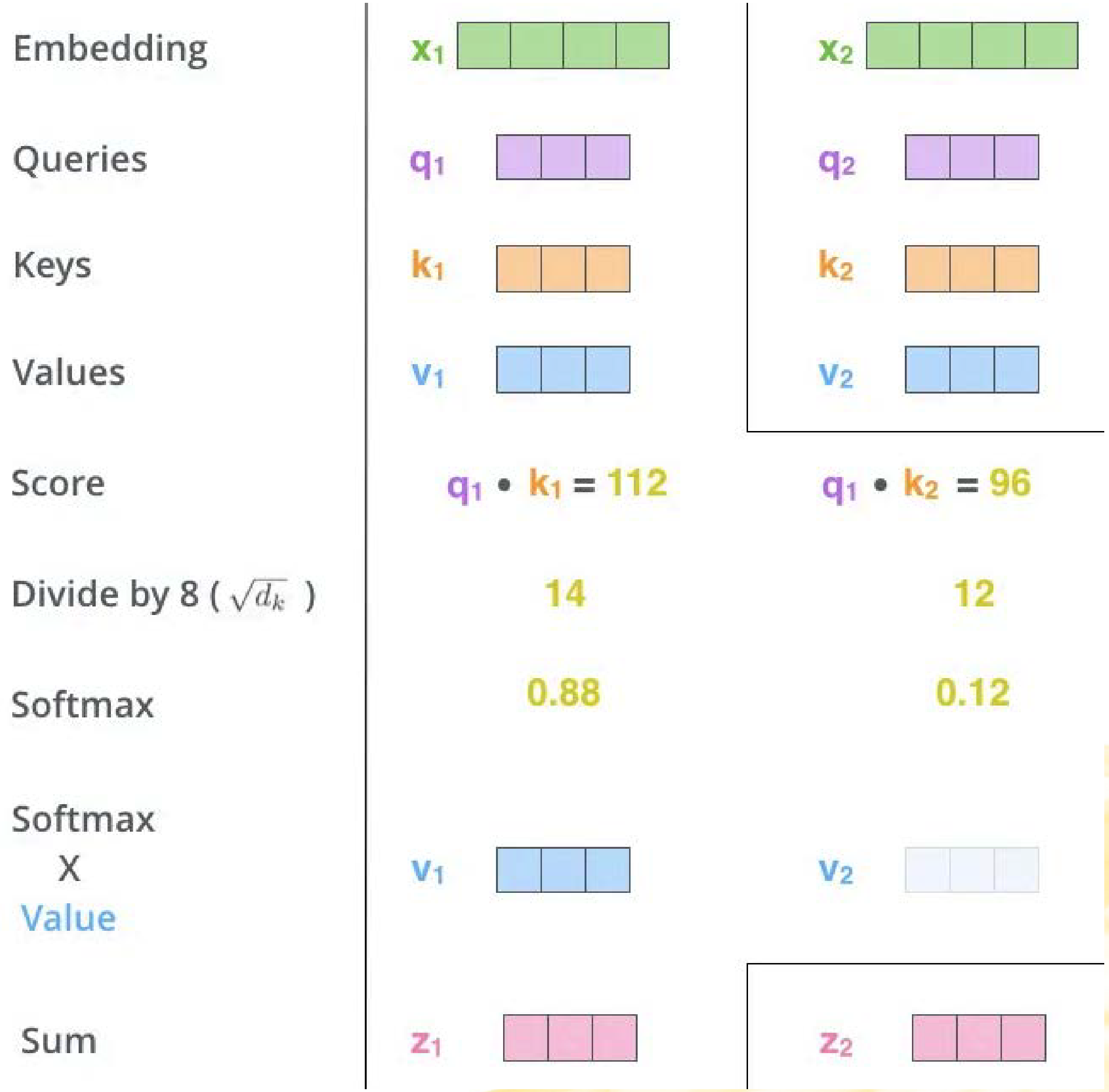
Encoder Layer

- Divide the scores by 8 (square root of dimension 64)
  - For stable gradient
- Pass through softmax
  - Normalizes the scores



Encoder Layer

- Multiply each value vector by softmax score
  - Keep the values of tokens we want to focus on intact and discount irrelevant tokens
- Sum up the weighted value vectors
- Output of self-attention layer for the embedding
  - Forwarded to the feedforward network

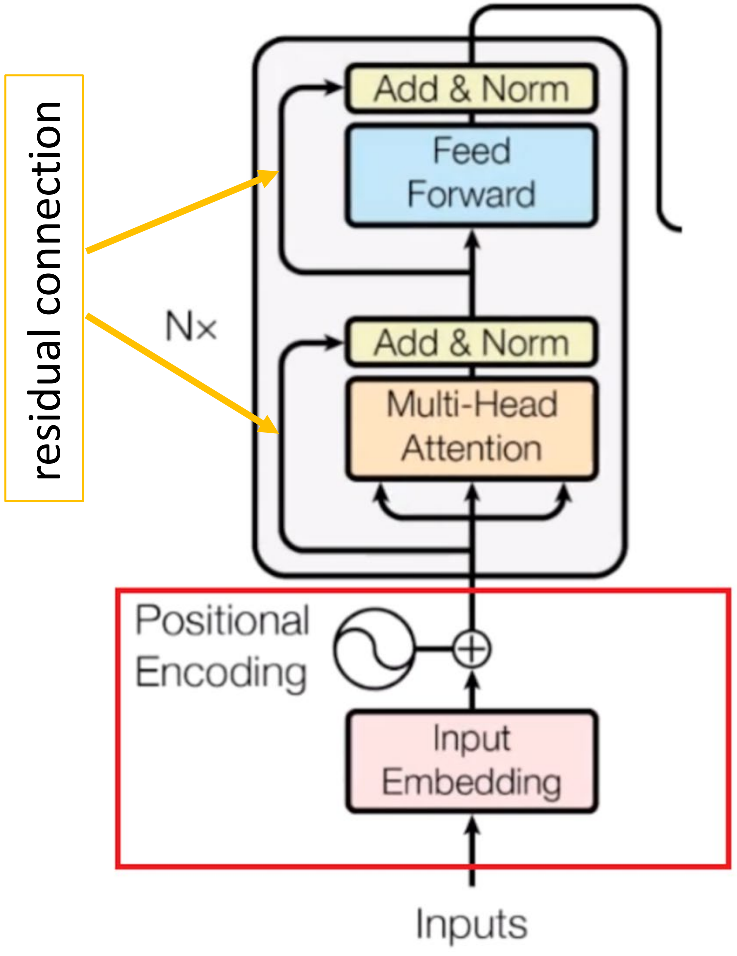


Encoder Layer

- Multi-head attention
  - Transformer repeats the computation of scores multiple times in parallel (each is called “attention head”)
  - Each attention head uses different matrices for computing query, key and value vectors
    - Provides richer interpretation of input text
  - Scores from all attention heads merged

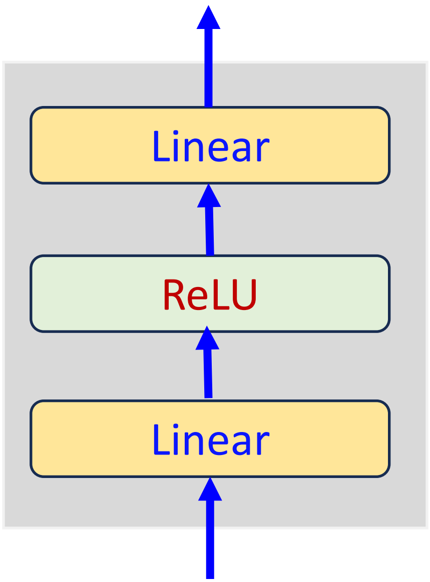
Normalization Layer

- Adjust the output vectors of the Multi-Head Self-Attention mechanism so that they have similar ranges of values
- Each output vector forward to feedforward layer separately



Feedforward Network & Output of Encoder

- Feedforward neural network
  - Dual linear layers with a ReLU activation between them



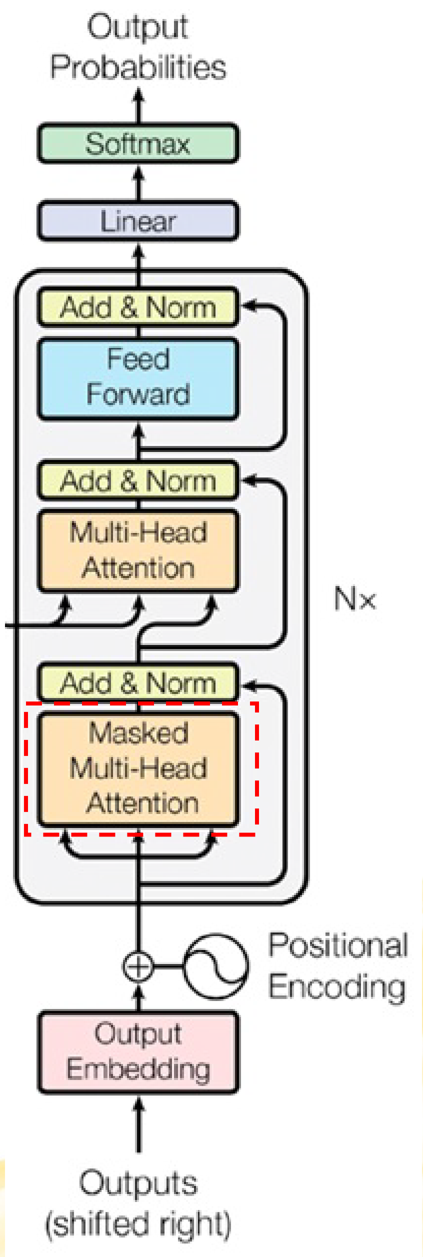
- Output of encoder
  - A set of vectors
  - Each vector represents the encoded representation of a word in the input sequence and is used as an input to decoder

Decoder

- Masked self-attention mechanism
  - Prevents positions from attending to subsequent positions so that each work is not influenced by future tokens

$$\underset{\text{scaled scores}}{\begin{bmatrix}0.5 & 0.3 & 0.1\\ 0.3 & 0.6 & 0.2\\ 0.1 & 0.2 & 0.3\end{bmatrix}} + \underset{\text{Look-ahead Mask}}{\begin{bmatrix}0 & -\mathrm{inf} & -\mathrm{inf}\\ 0 & 0 & -\mathrm{inf}\\ 0 & 0 & 0\end{bmatrix}} = \underset{\text{Masked Scores}}{\begin{bmatrix}0.5 & -\mathrm{inf} & -\mathrm{inf}\\ 0.3 & 0.6 & -\mathrm{inf}\\ 0.1 & 0.2 & 0.3\end{bmatrix}}$$

  - Masking ensures that predictions for a particular position depends only on known outputs at positions before it (e.g., constructed outputs so far)



Decoder

- Encoder-decoder multi-head attention
  - Outputs from **encoder** act as both keys and values
  - Output from **first multi-head attention layer** serve as queries
  - Tries to align encoder's input with that of decoder
- Linear layer and softmax serve as a classifier
  - Size of output equal to the size of the vocabulary
  - Index corresponding to the highest probability points to the word predicted by the model