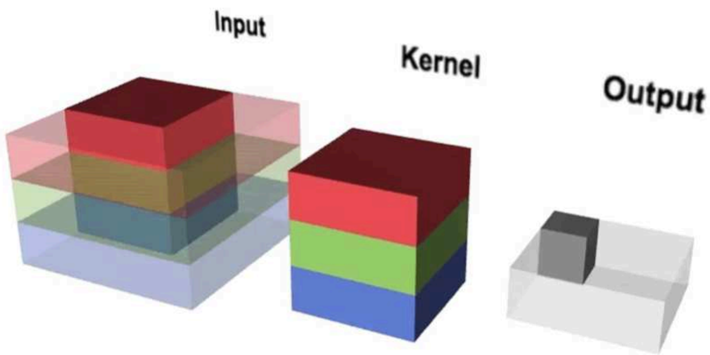
Biomedical Data Science: Mining and Modeling

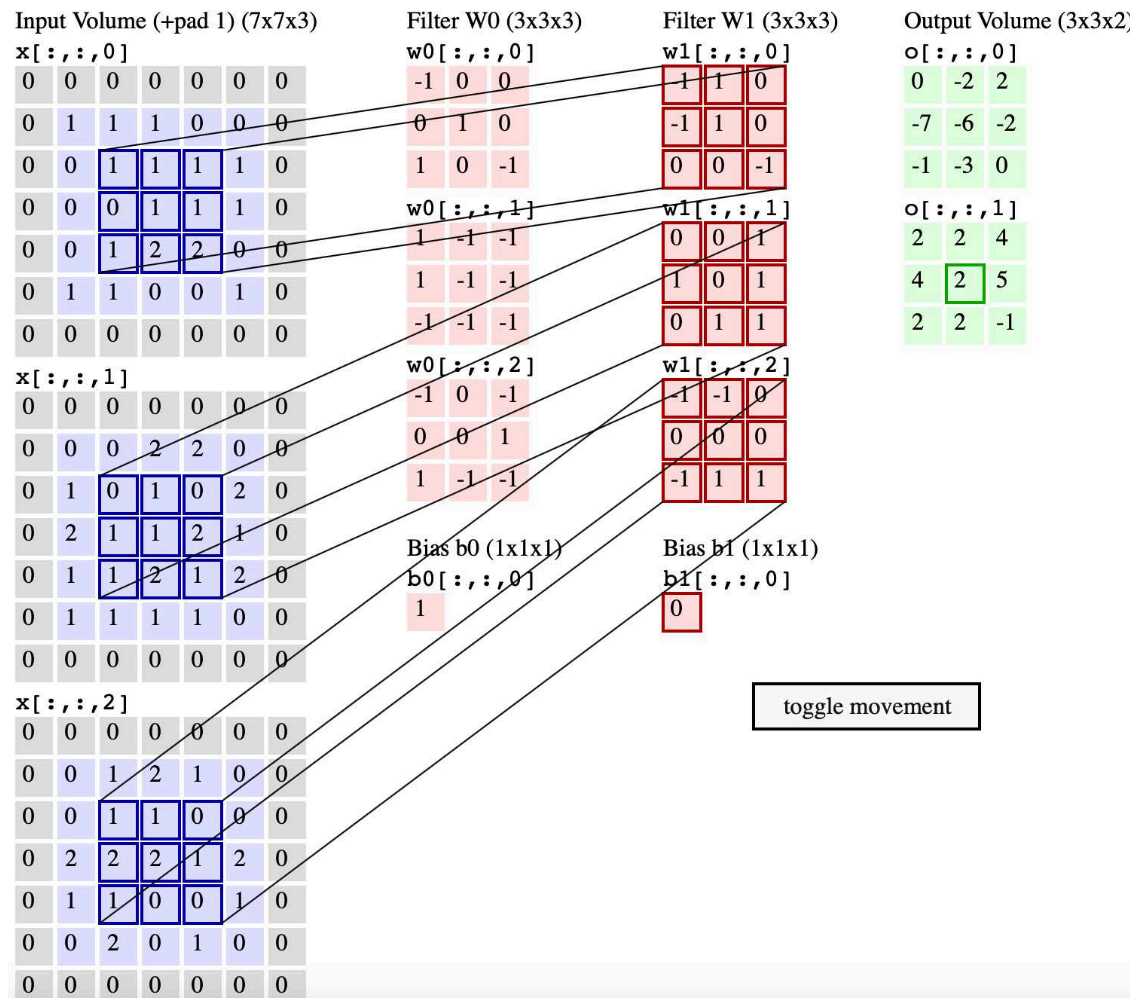
Deep Learning: Part II
Deep Generative Models

Dr. Martin Renqiang Min
NEC Laboratories America
Mar 29, 2023

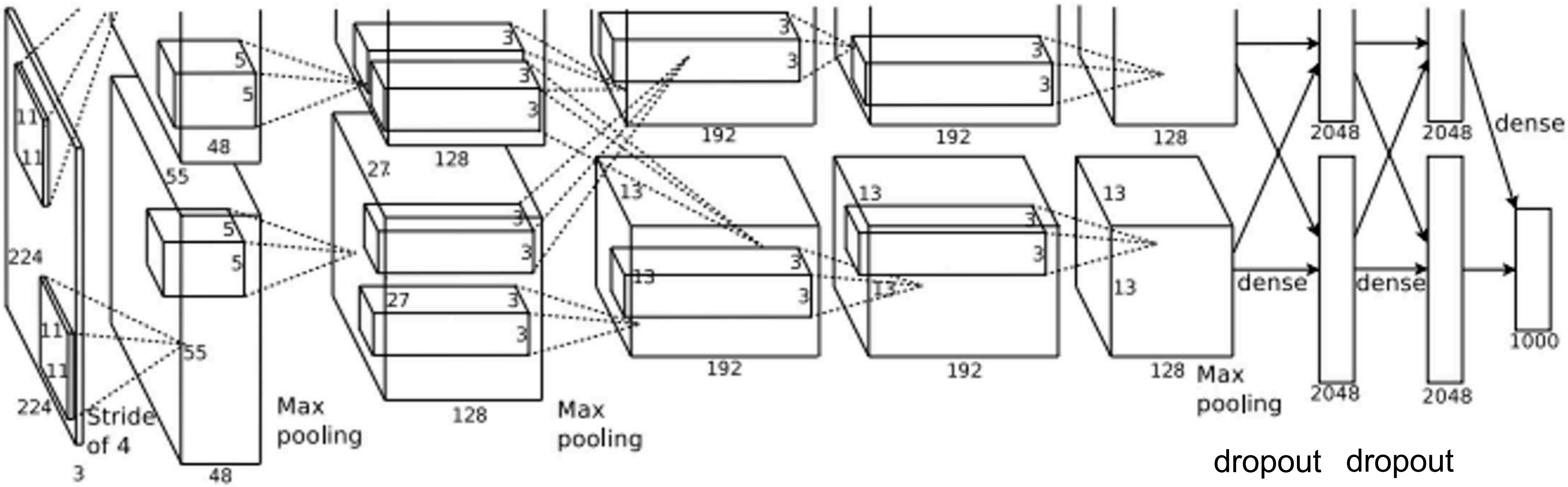
2D 3x3 Convolution Applied to RGB Input of Size 5x5



Picture credit:
https://thomelane.github.io/convolutions/2DConvRGB.html



AlexNet Network Structure



dropout 0.5 dropout 0.5

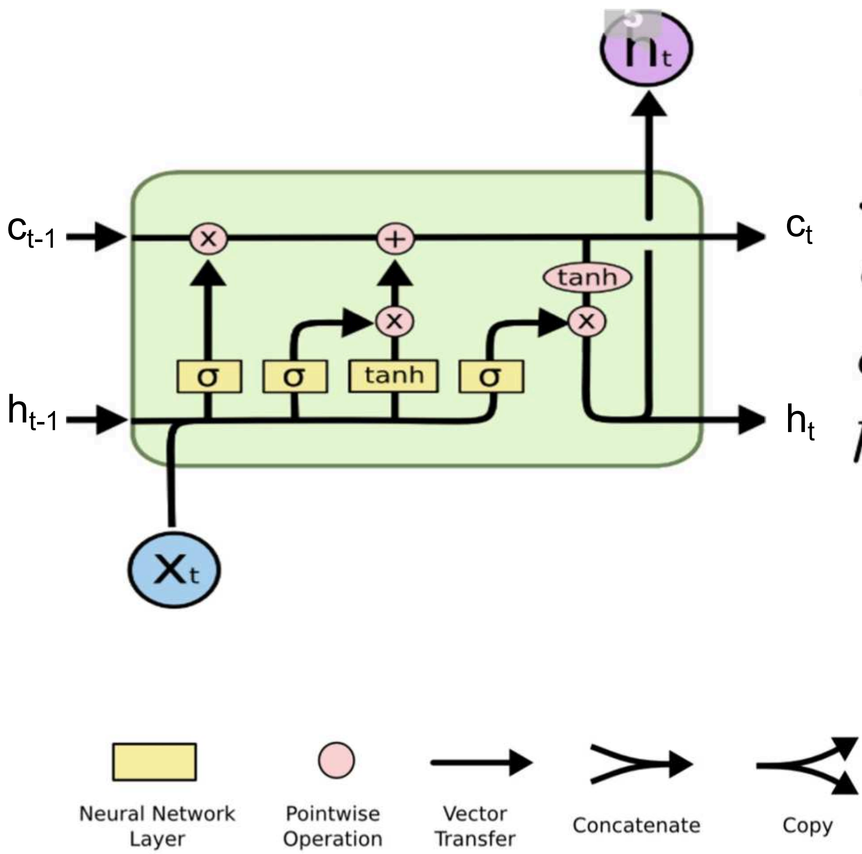
padding
227X227X3

3x3
stride 2

Output Size = $(W - F + 2P)/S + 1$

Pay attention to the output Size and the number of parameters

Long Short-Term Memory



$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$

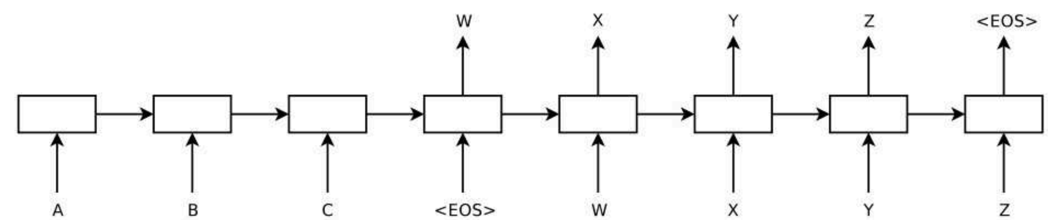


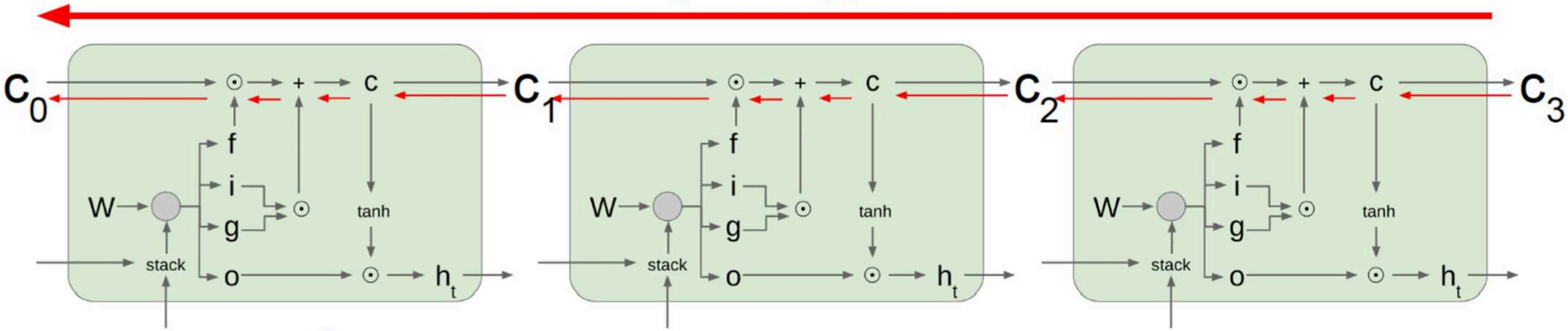
Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Picture Credit:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

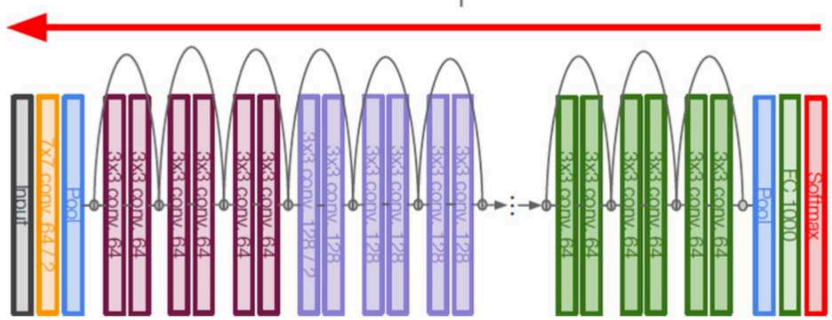
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

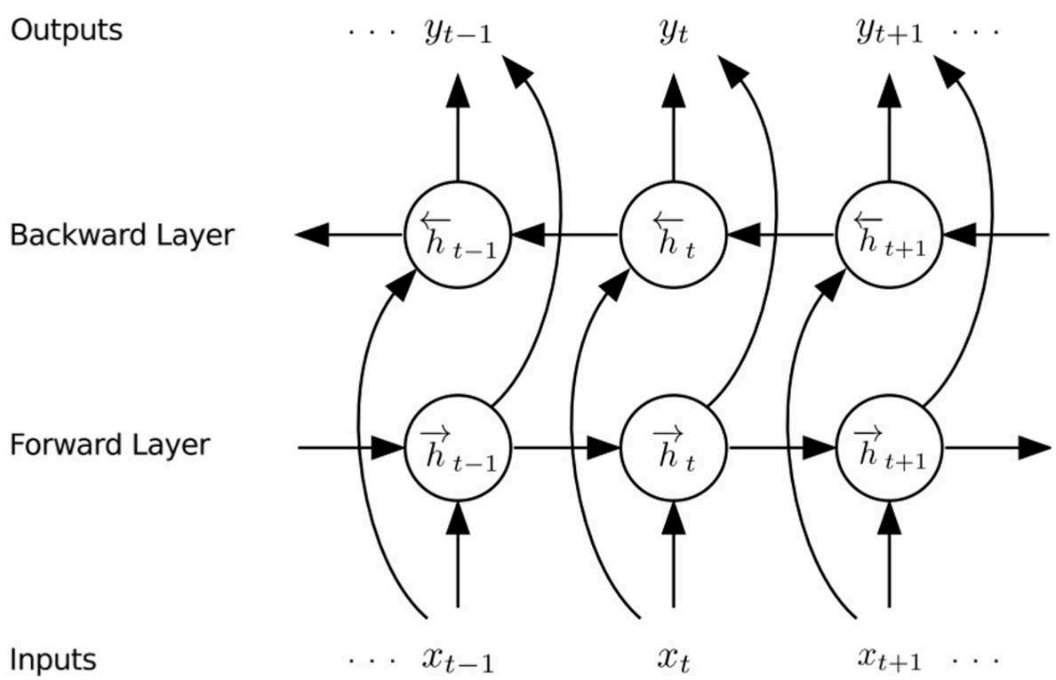
$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

Bidirectional LSTM



$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

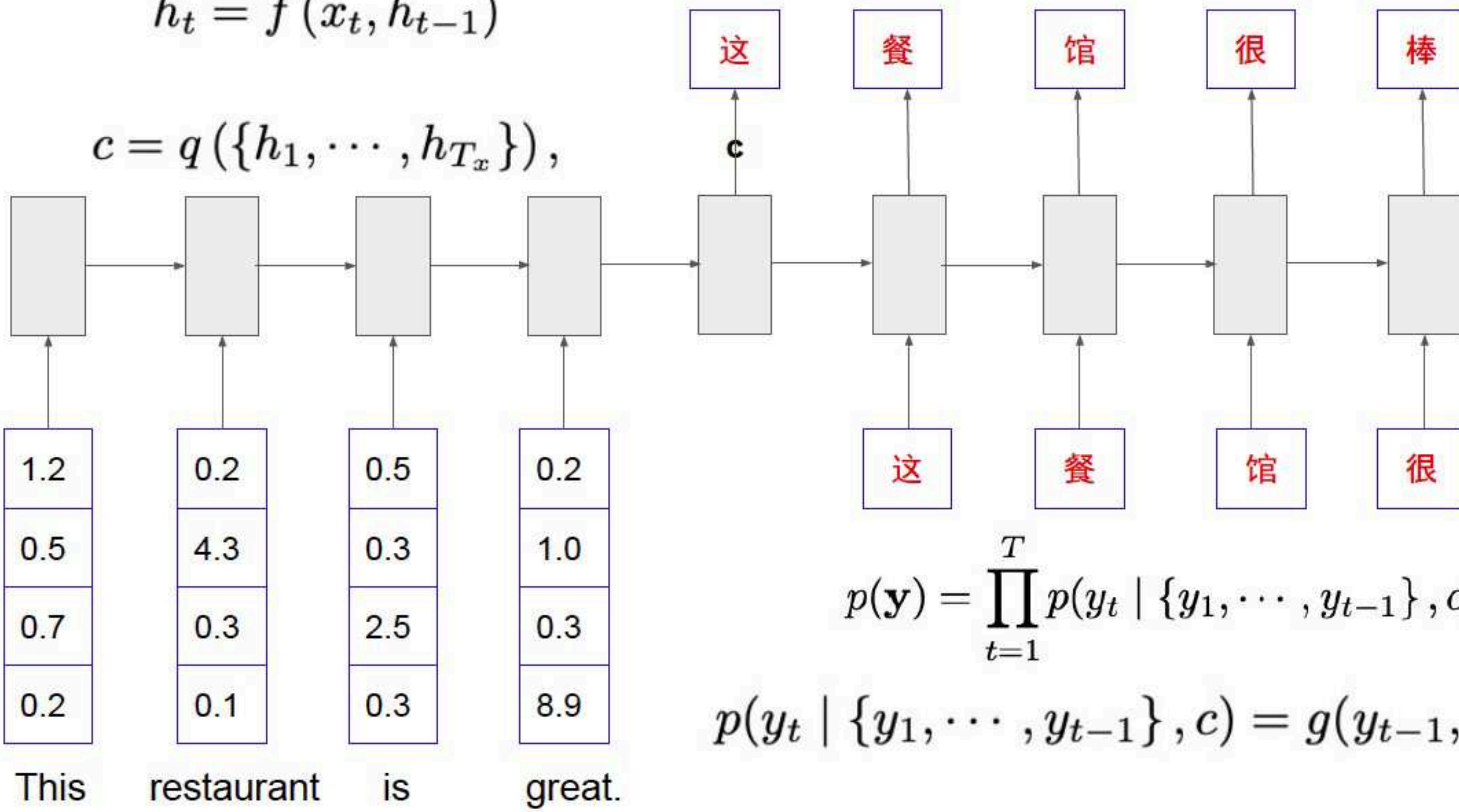
$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y$$

Picture Credit: https://www.cs.toronto.edu/~graves/asru_2013.pdf

Sequence-to-Sequence Model for Machine Translation

$$h_t = f(x_t, h_{t-1})$$

$$c = q(\{h_1, \cdots, h_{T_x}\}),$$



$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c),$$

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c),$$

Encoder-Decoder with Attention for Machine Translation

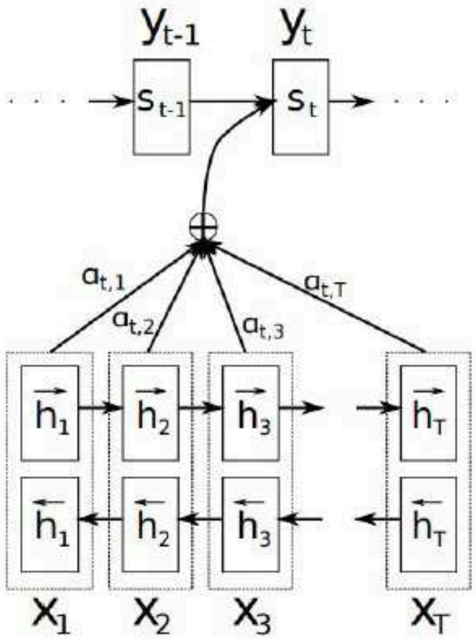


Figure 1: The graphical illus- the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

Encoder:

$$\overrightarrow{h}_i = \begin{cases} (1 - \overrightarrow{z}_i) \circ \overrightarrow{h}_{i-1} + \overrightarrow{z}_i \circ \overrightarrow{\underline{h}}_i & , \text{if } i > 0 \\ 0 & , \text{if } i = 0 \end{cases}$$

$$\overrightarrow{\underline{h}}_i = \tanh\left(\overrightarrow{W} \overline{E} x_i + \overrightarrow{U}\left[\overrightarrow{r}_i \circ \overrightarrow{h}_{i-1}\right]\right)$$
$$\overrightarrow{z}_i = \sigma\left(\overrightarrow{W}_z \overline{E} x_i + \overrightarrow{U}_z \overrightarrow{h}_{i-1}\right)$$
$$\overrightarrow{r}_i = \sigma\left(\overrightarrow{W}_r \overline{E} x_i + \overrightarrow{U}_r \overrightarrow{h}_{i-1}\right).$$

$$h_i = \begin{bmatrix} \overrightarrow{h}_i \\ \overleftarrow{h}_i \end{bmatrix}$$

Decoder:

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

$$\tilde{s}_i = \tanh\left(W E y_{i-1} + U\left[r_i \circ s_{i-1}\right] + C c_i\right)$$
$$z_i = \sigma\left(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i\right)$$
$$r_i = \sigma\left(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i\right)$$

$$t_i = U_o s_{i-1} + V_o E y_{i-1} + C_o c_i.$$

$$p(y_i | s_i, y_{i-1}, c_i) \propto \exp\left(y_i^\top W_o t_i\right),$$

$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$e_{ij} = a(s_{i-1}, h_j)$$

Bahdanau *et al.*, NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE. ICLR 2015.

Self Attention: Transformer

- Self Attention: use word representations in a sequence to attend to word representations at different positions in the same sequence
  - Capture long-range dependencies in a sequence more efficiently

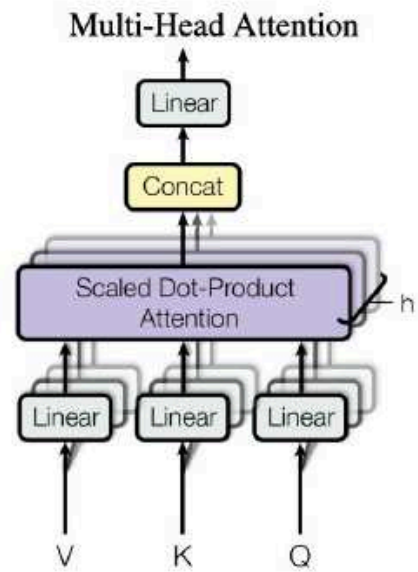
- Scaled dot-product attention
  - Transformer views encoder representations of an input sequence as Key-Value (K, V) pairs and employs multi-head scaled dot-product attention

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{n}}\right)\mathbf{V}$$

Vaswani et al., Attention Is All You Need. NIPS 2017.

Multi-Head Scaled Dot-Product Attention in Transformer

Multi-head attention jointly attend to information from different representation subspaces at different positions.

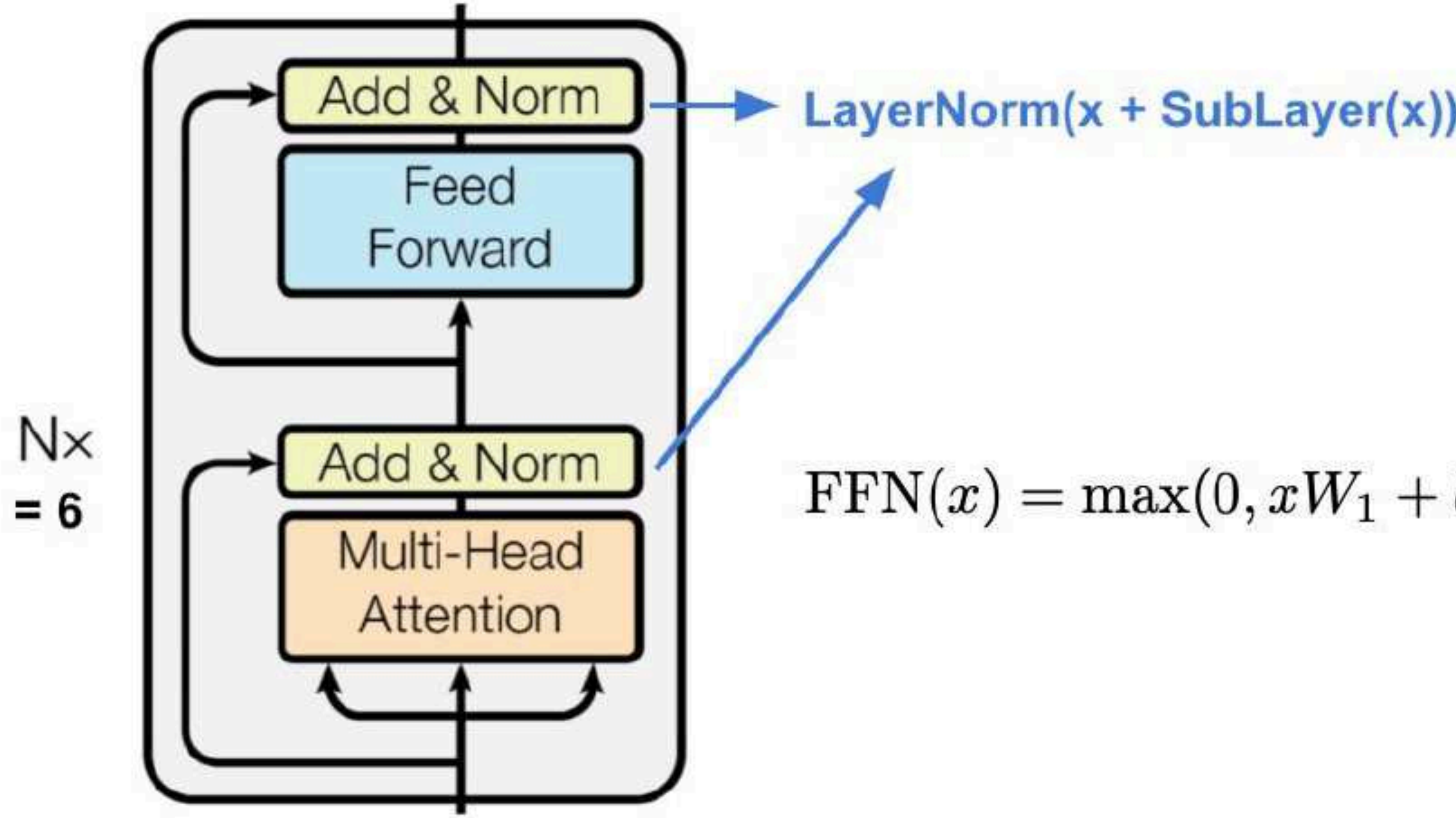


$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{QK}^\top}{\sqrt{n}})\mathbf{V}$$

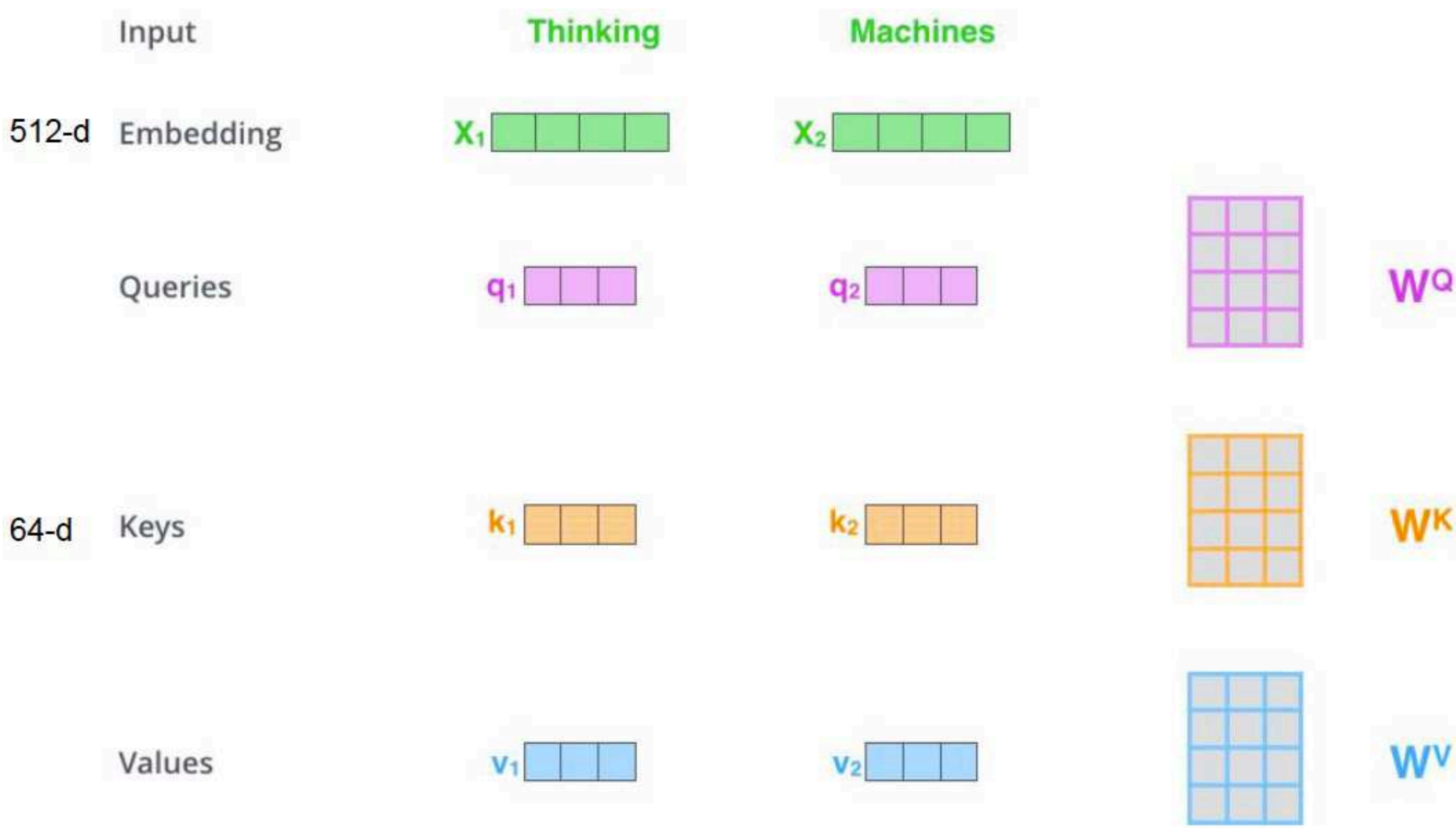
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

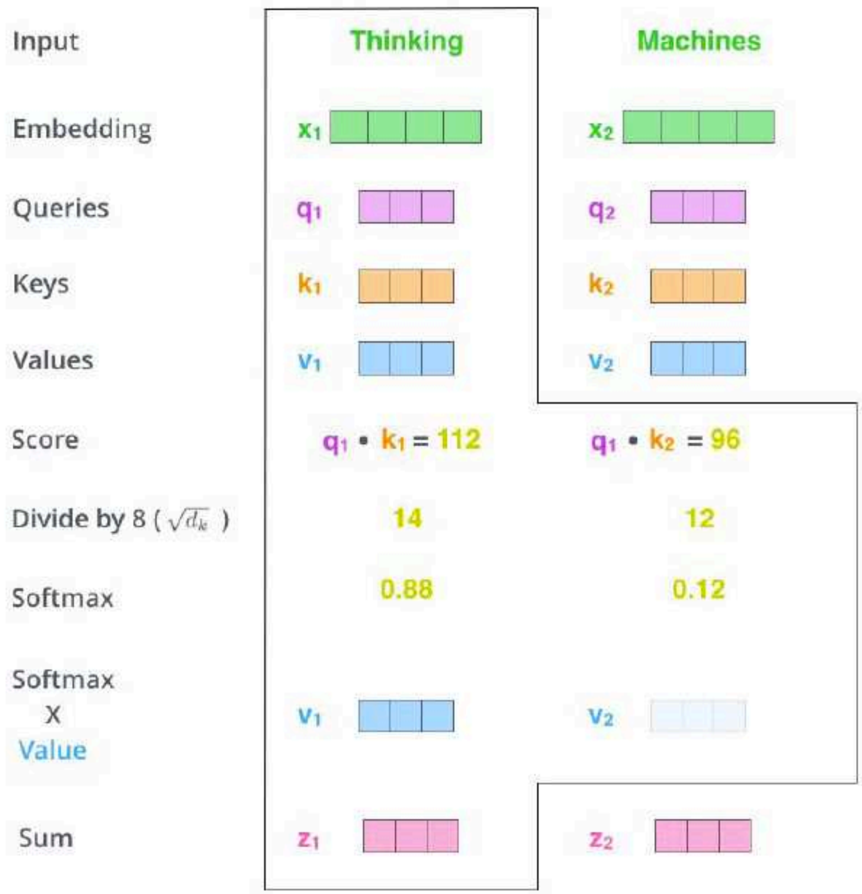
Encoder of Transformer

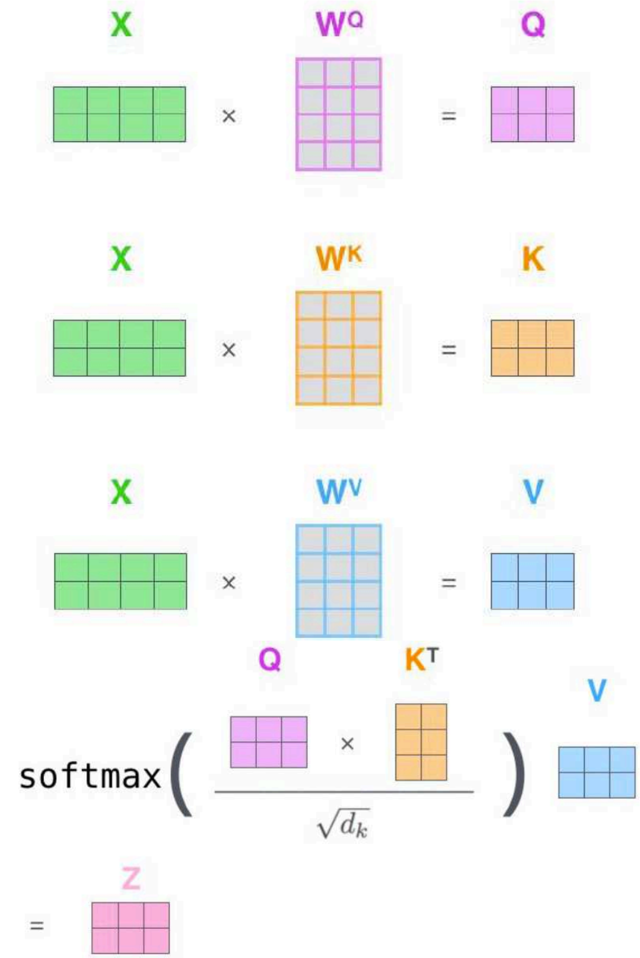


$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



picture credit: https://jalammar.github.io/illustrated-transformer/





picture credit: https://jalammar.github.io/illustrated-transformer/

Transformer-based Large Language Models

$p(s_{n-k}, ..., s_n | s_1, ..., s_{n-k-1})$ $p(output|input, task)$

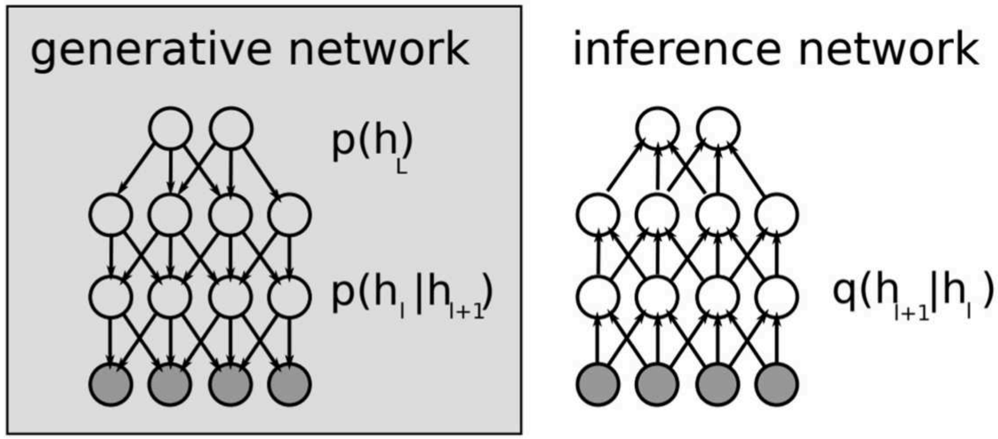
Random	25.0%
Average human rater	34.5%
GPT-3 5-shot	43.9%
Gopher 5-shot	60.0%
***Chinchilla* 5-shot**	**67.6%**
Average human expert performance	*89.8%*
June 2022 Forecast	57.1%
June 2023 Forecast	63.4%

Table 6 | **Massive Multitask Language Understanding (MMLU).** We report the average 5-shot accuracy over 57 tasks with model and human accuracy comparisons taken from Hendrycks et al. (2020). We also include the average prediction for state of the art accuracy in June 2022/2023 made by 73 competitive human forecasters in Steinhardt (2021).

https://arxiv.org/pdf/2203.15556.pdf

Directed Probabilistic Generative Models with Hidden Units

We want to train a directed generative model p



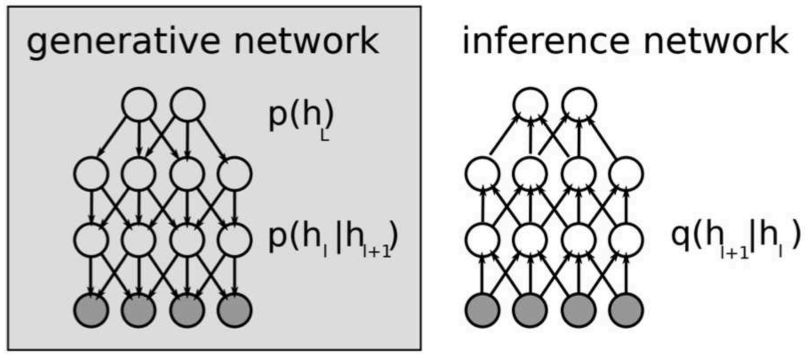
$$p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h}_1)p(\mathbf{h}_1|\mathbf{h}_2)...p(\mathbf{h}_L)$$
$$q(\mathbf{h}|\mathbf{x}) = q(\mathbf{h}_1|\mathbf{x})q(\mathbf{h}_2|\mathbf{h}_1)...q(\mathbf{h}_L|\mathbf{h}_{L-1})$$

- Our goal is to learn the model parameters to maximize the log-probability of data x
  - Learning: learn the model parameters maximizing log p(x)
  - Inference: infer the hidden states from p(h | x)

Variational Inference

We want to train a directed generative model p

Variational Bound of Log-Likelihood P(x)



$$p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h}_1)p(\mathbf{h}_1|\mathbf{h}_2)...p(\mathbf{h}_L)$$
$$q(\mathbf{h}|\mathbf{x}) = q(\mathbf{h}_1|\mathbf{x})q(\mathbf{h}_2|\mathbf{h}_1)...q(\mathbf{h}_L|\mathbf{h}_{L-1})$$

$$\max_\theta \mathbb{E}_{\hat{p}(x)} \ln p_\theta(x) = \max_\theta \mathbb{E}_{\hat{p}(x)} \ln \int_z p_\theta(x, z) dz.$$

$$\max_\theta \mathbb{E}_{\hat{p}(x)} \left[\ln p_\theta(x) - \min_{q \in \mathcal{Q}} D(q(z) \parallel p_\theta(z \mid x)) \right] = \max_\theta \mathbb{E}_{\hat{p}(x)} \left[\max_{q \in \mathcal{Q}} \mathbb{E}_{q(z)} \ln \frac{p_\theta(x, z)}{q(z)} \right]$$

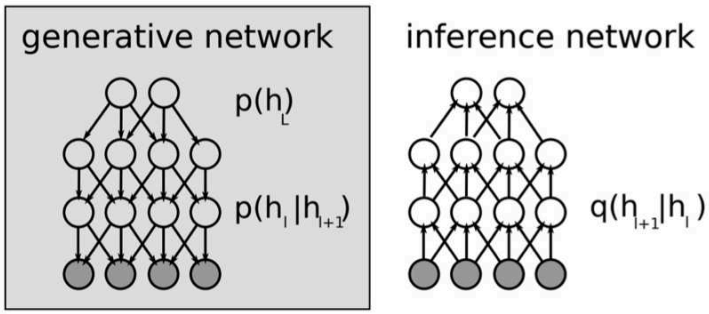
Every data point x has its own variational parameters (q(z)): flexible but not scalable.

Amortized Variational Inference

All data points share a variational inference network Q parameterized by a neural network.

Variational Bound of Log-Likelihood P(x)

We want to train a directed generative model p



$$p(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}|\mathbf{h}_1)p(\mathbf{h}_1|\mathbf{h}_2)...p(\mathbf{h}_L)$$

$$q(\mathbf{h}|\mathbf{x}) = q(\mathbf{h}_1|\mathbf{x})q(\mathbf{h}_2|\mathbf{h}_1)...q(\mathbf{h}_L|\mathbf{h}_{L-1})$$

$$\begin{aligned}
\log P_\theta(x) &= \log \sum_h P_\theta(x, h) \\
&\geq \sum_h Q_\phi(h|x) \log \frac{P_\theta(x, h)}{Q_\phi(h|x)} \\
&= E_Q[\log P_\theta(x, h) - \log Q_\phi(h|x)] \\
&= \mathcal{L}(x, \theta, \phi).
\end{aligned}$$

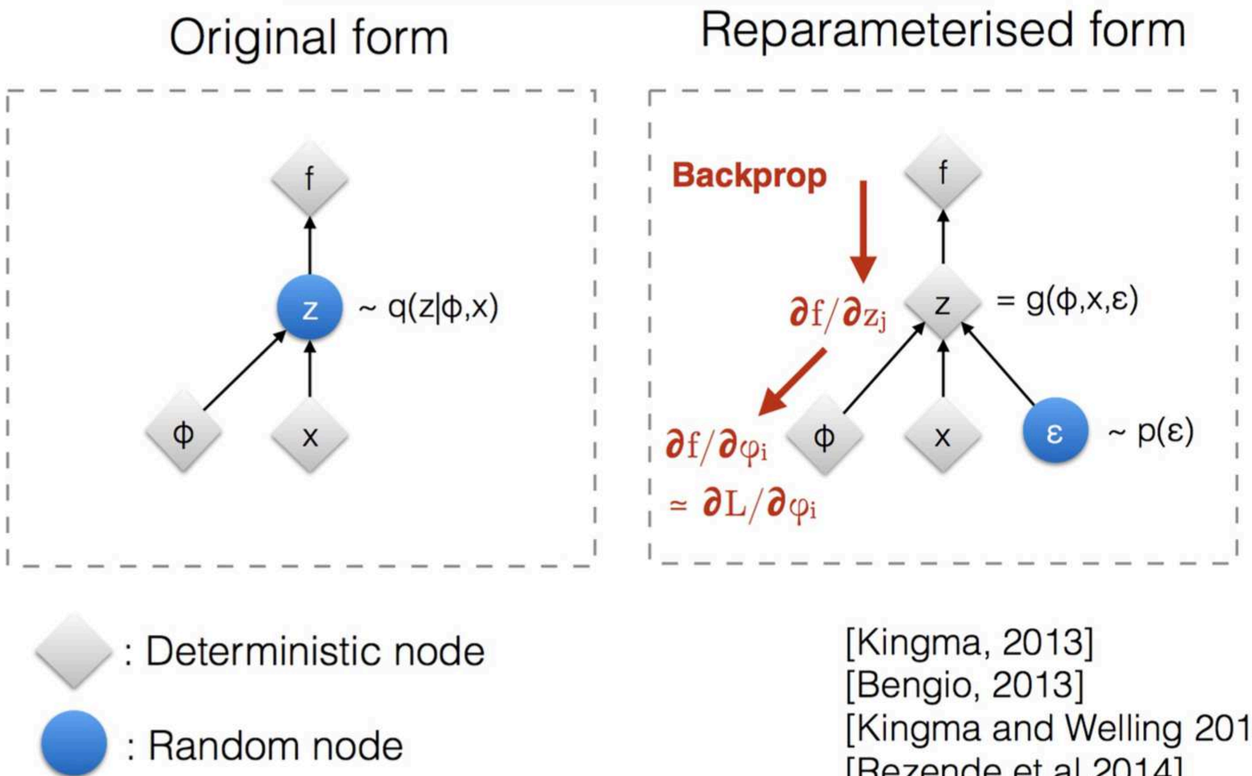
By rewriting the bound as

$$\mathcal{L}(x, \theta, \phi) = \log P_\theta(x) - KL(Q_\phi(h|x), P_\theta(h|x)),$$

The Reparameterization Trick Using a Deterministic Function Mapping

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$$



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

Variational Inference with the Reparameterization Trick

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

ELBO:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

Variational Autoencoder with a Isotropic Multivariate Gaussian Prior

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

$$p(z) \equiv \mathcal{N}(0, I)$$

$$p(x|z) \equiv \mathcal{N}(f(z), cI) \qquad f \in F \qquad c > 0$$

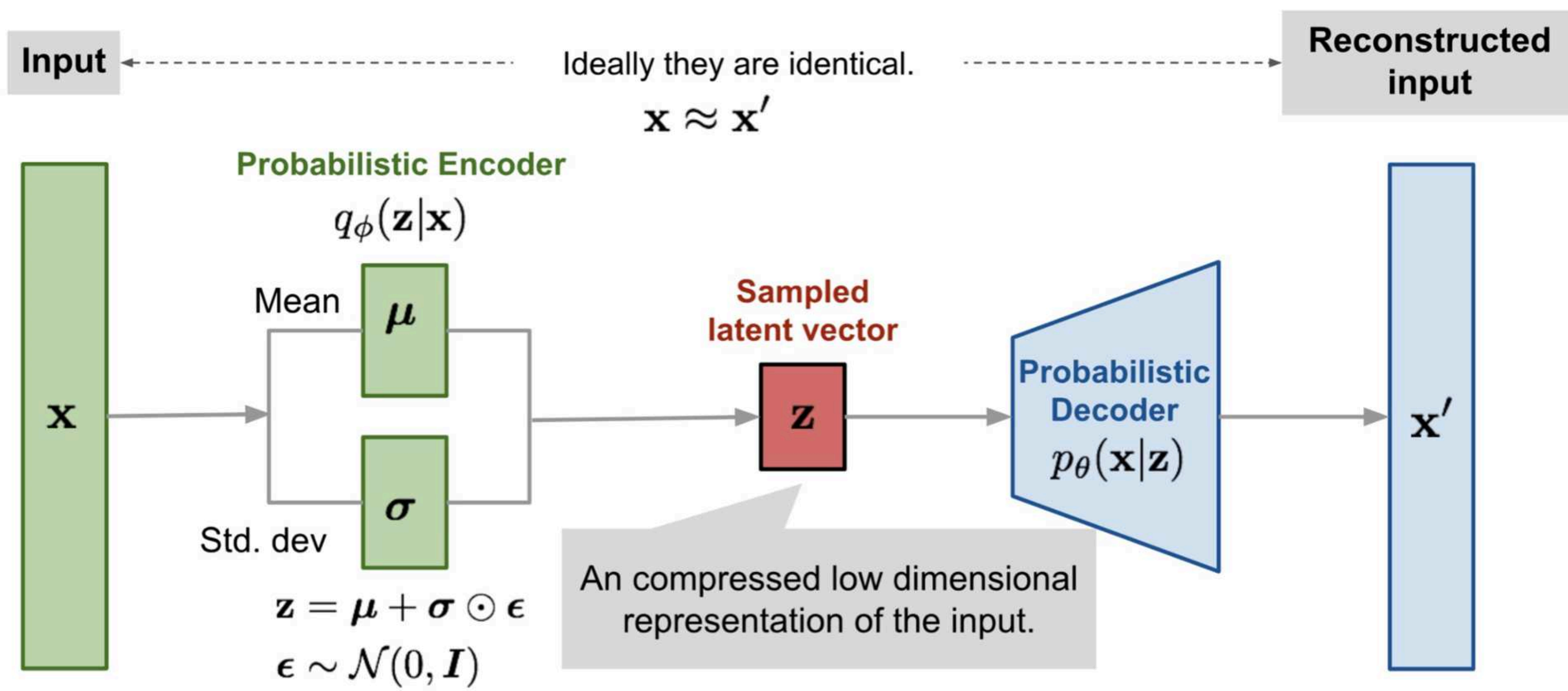
$$\begin{aligned} f^* &= \arg\max_{f \in F} \mathbb{E}_{z \sim q_x^*}(\log p(x|z)) \\ &= \arg\max_{f \in F} \mathbb{E}_{z \sim q_x^*}\left(-\frac{||x - f(z)||^2}{2c}\right) \end{aligned}$$

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

$$\text{where} \quad \mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$$

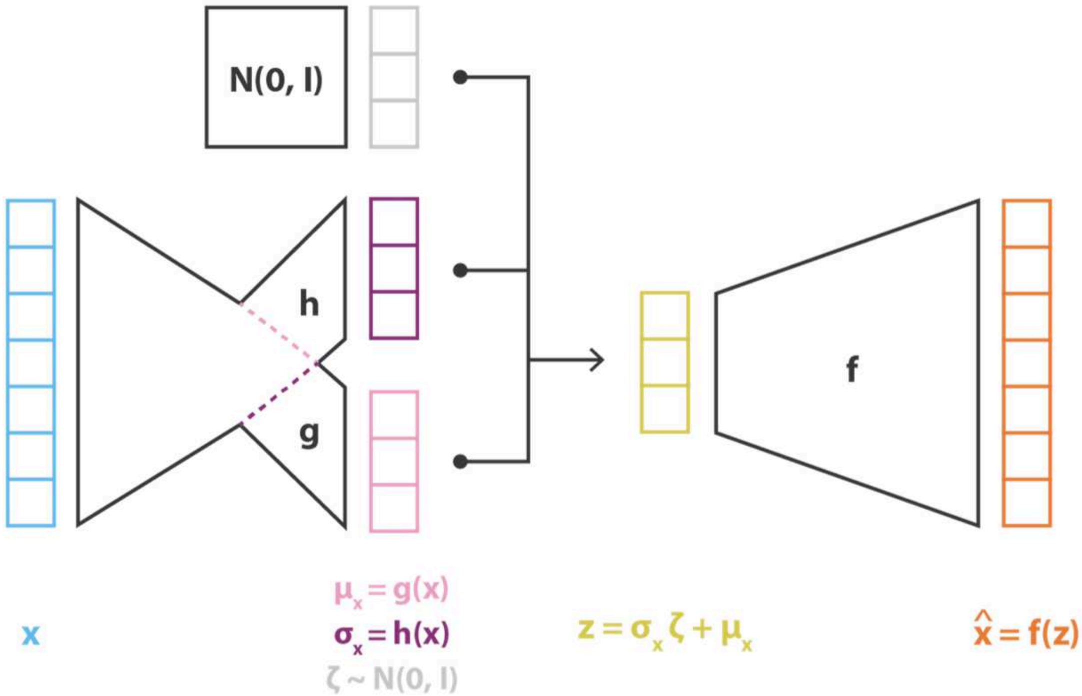
Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

Variational Autoencoder with a Isotropic Multivariate Gaussian Prior



Picture Credit: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

VAE Loss



$$\text{loss} = C \,\|\, x - \hat{x} \,\|^2 + \text{KL}[\, N(\mu_x, \sigma_x), N(0, I)\,] = C \,\|\, x - f(z) \,\|^2 + \text{KL}[\, N(g(x), h(x)), N(0, I)\,]$$

Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Training VAE Using Mini-batch Variational Inference with the Reparameterization Trick

---

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initialize parameters
repeat
  $\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)
  $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
  $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^M, \boldsymbol{\epsilon})$ (Gradients of minibatch estimator (8))
  $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using gradients $\mathbf{g}$ (e.g. SGD or Adagrad [DHS10])
until convergence of parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$
return $\boldsymbol{\theta}, \boldsymbol{\phi}$

Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

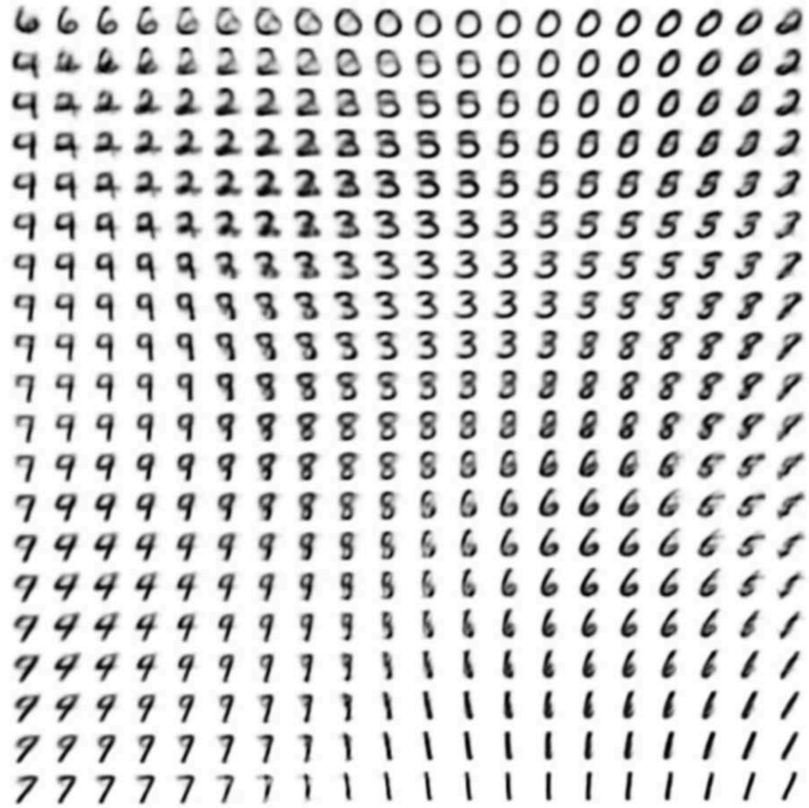
VAE for Generating MNIST Digits

left: 1st epoch, middle: 9th epoch, right: original

Picture Credit: http://kvfrans.com/variational-autoencoders-explained/

Learned 2D Manifold by VAE

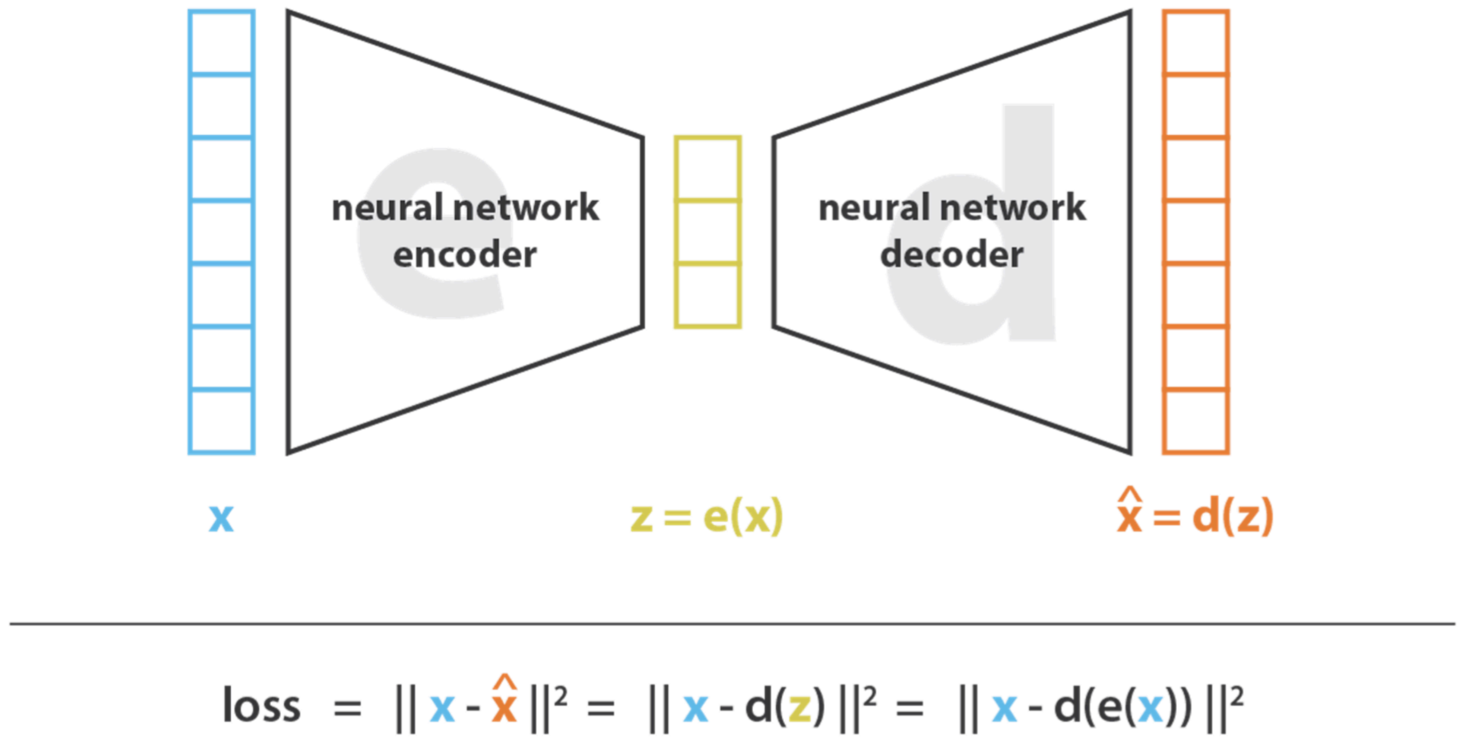
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

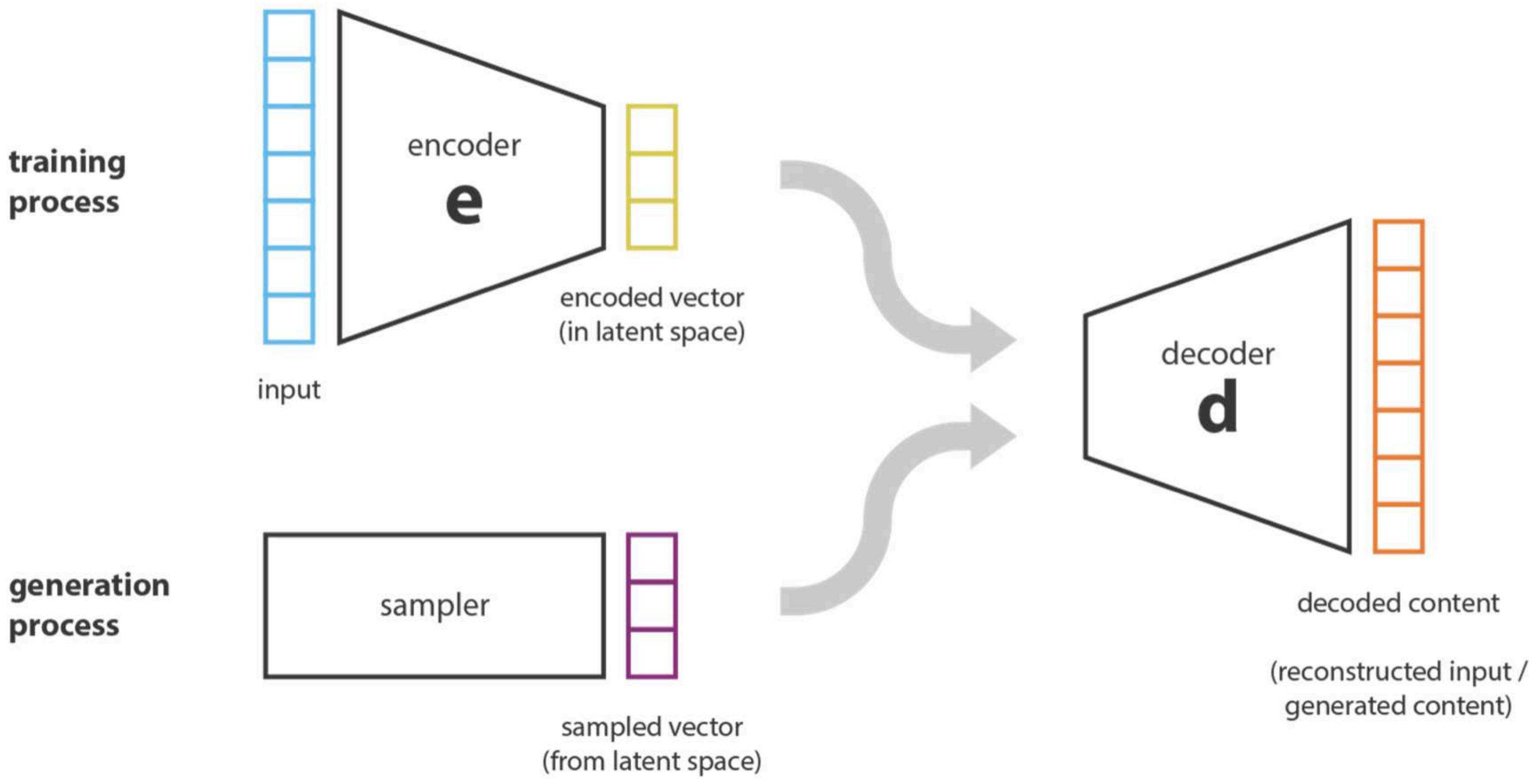
Kingma and Welling, Auto-Encoding Variational Bayes. ICLR 2014

Autoencoder vs. Variational Autoencoder



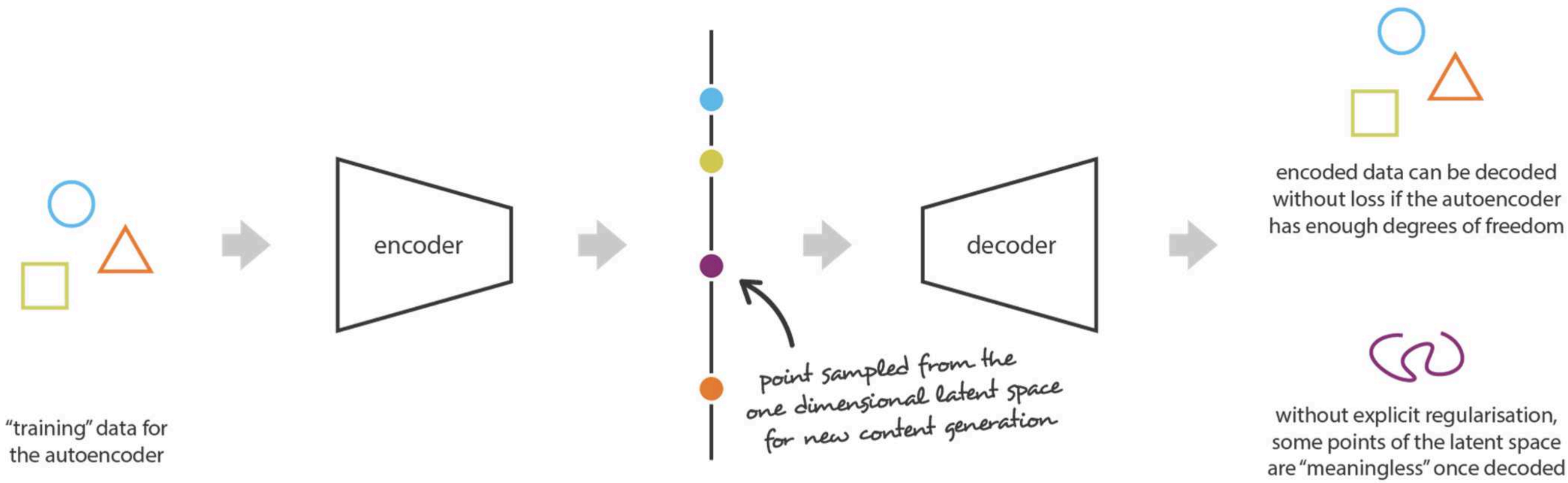
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Autoencoder vs. Variational Autoencoder



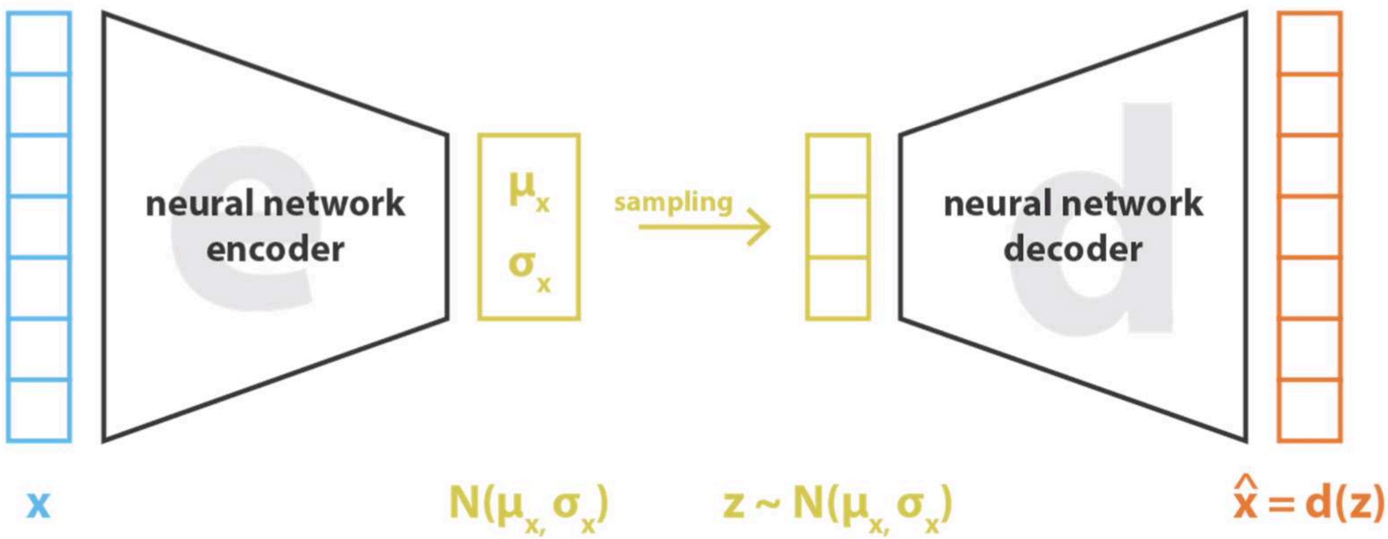
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Autoencoder vs. Variational Autoencoder



Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

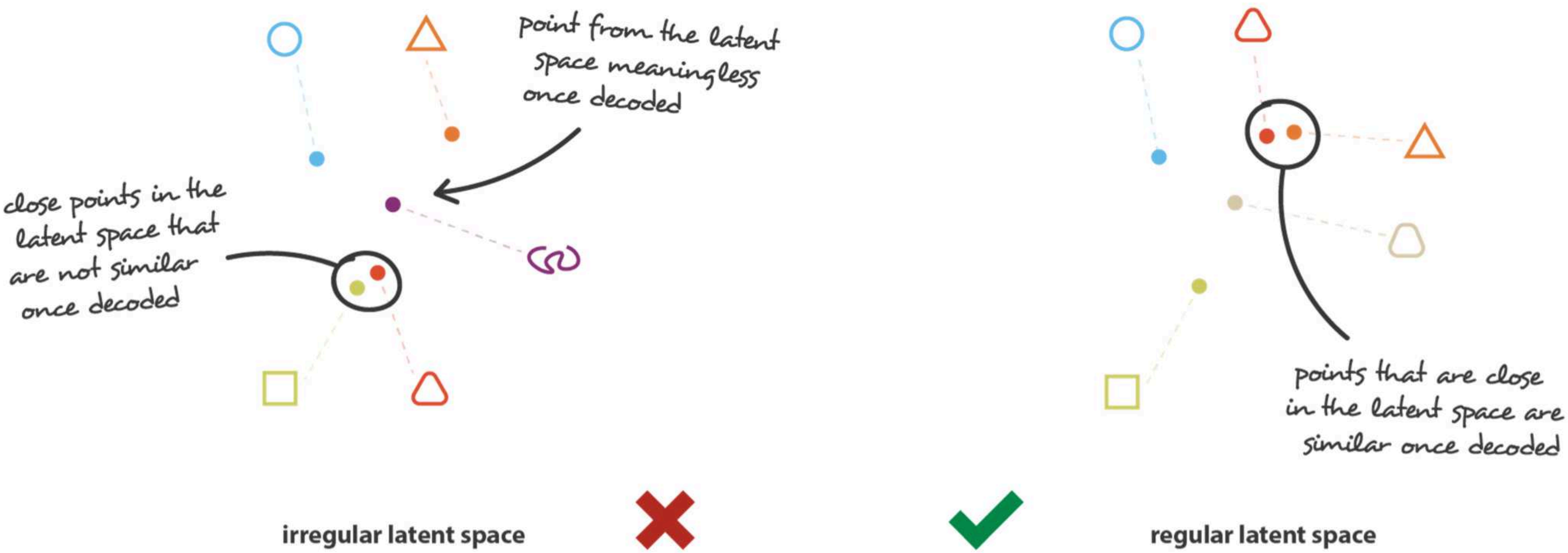
Autoencoder vs. Variational Autoencoder



$$\text{loss} = \| x - \hat{x} \|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \| x - d(z) \|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

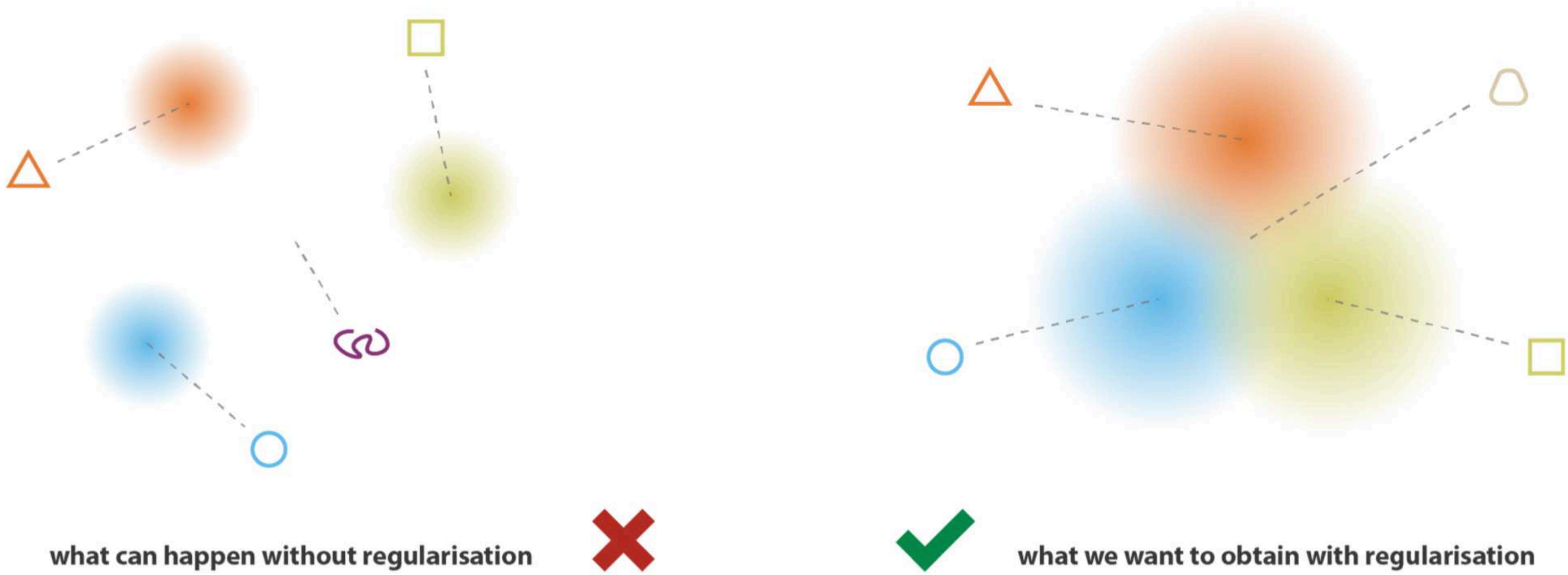
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Autoencoder vs. Variational Autoencoder



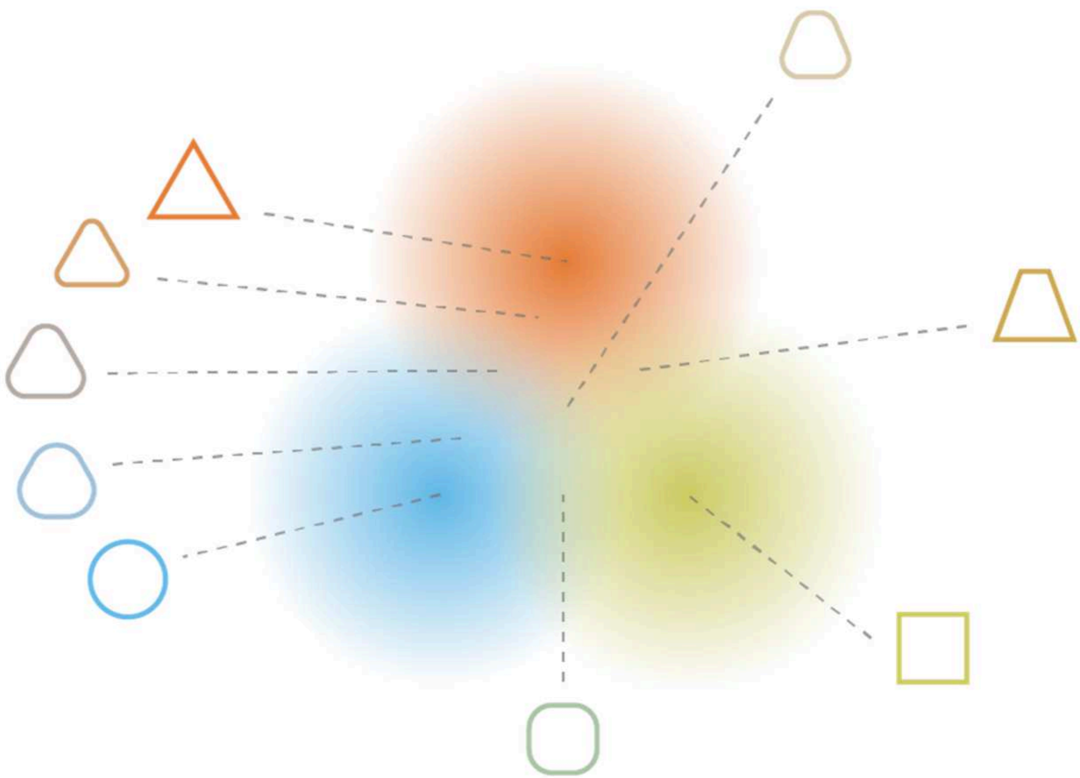
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Autoencoder vs. Variational Autoencoder



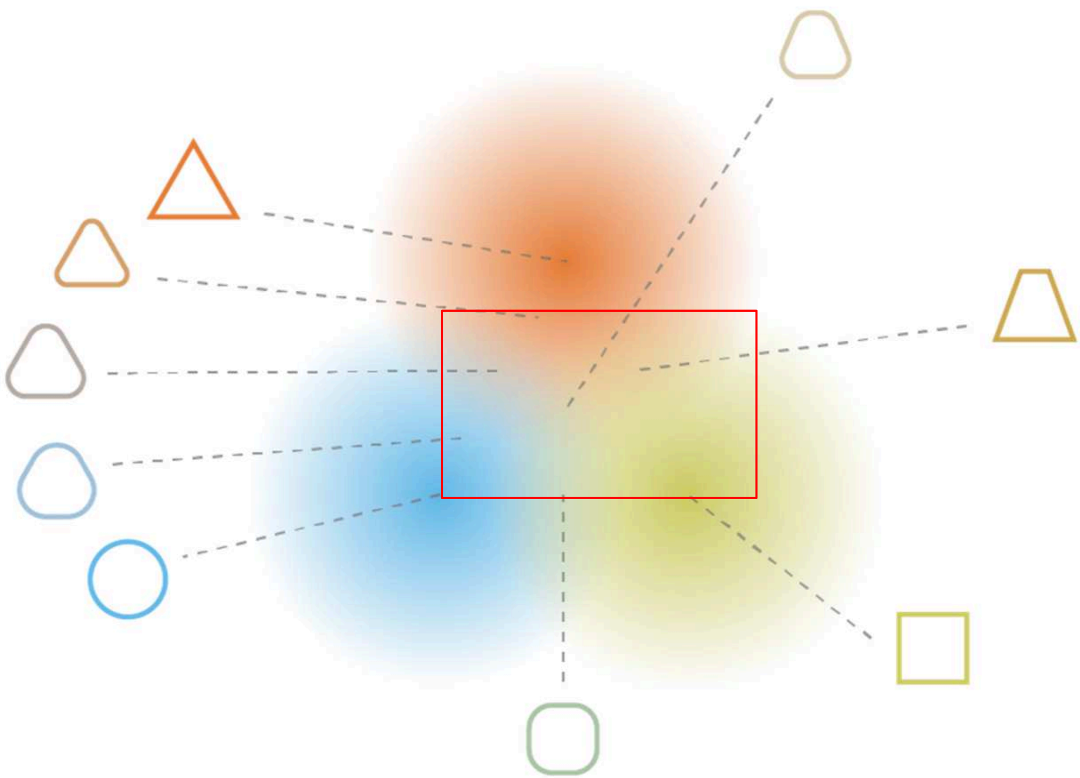
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Autoencoder vs. Variational Autoencoder



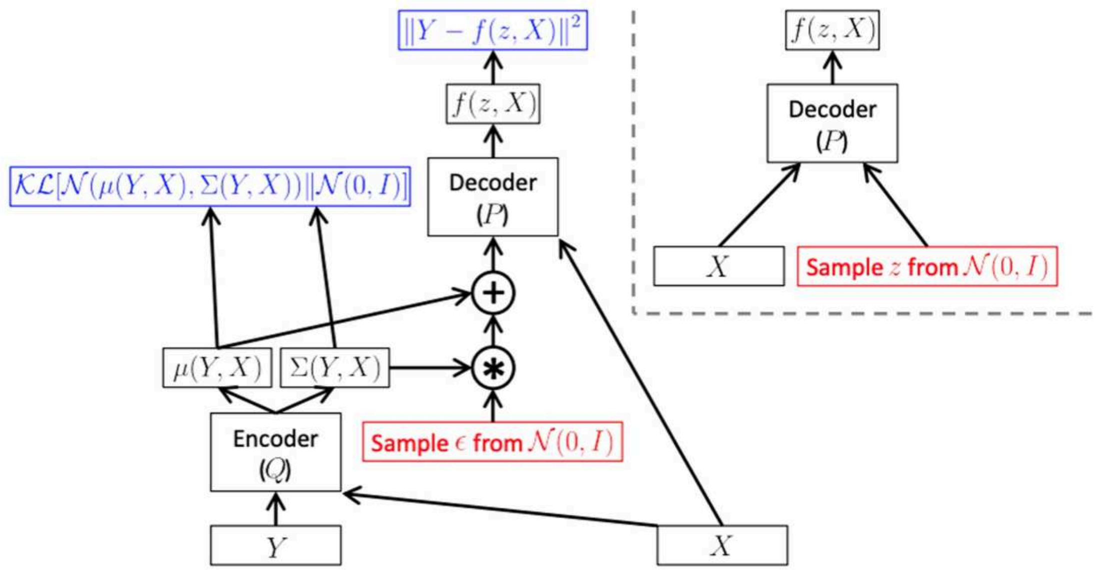
Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Problems of VAE: Overlapping Latent Space



Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Conditional VAE (There Are Other Conditioning Priors)



Left: a training-time conditional variational autoencoder implemented as a feedforward neural network

Right: the same model at test time, when we want to sample from $P(Y|X)$.

Picture Credit: https://arxiv.org/pdf/1606.05908.pdf

Conditional VAE (There Are Other Conditioning Priors)

$$\log p_\theta(\mathbf{y}|\mathbf{x}) \geq -KL\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\|p_\theta(\mathbf{z}|\mathbf{x})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z})\right]$$

and the empirical lower bound is written as:

$$\widetilde{\mathcal{L}}_{\text{CVAE}}(\mathbf{x},\mathbf{y};\theta,\phi) = -KL\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\|p_\theta(\mathbf{z}|\mathbf{x})\right) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{y}|\mathbf{x},\mathbf{z}^{(l)}),$$

$\mathbf{z}^{(l)} = g_\phi(\mathbf{x},\mathbf{y},\epsilon^{(l)}),\ \epsilon^{(l)} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and L is the number of samples.

Sohn *et al.*, Learning Structured Output Representation using Deep Conditional Generative Models. NIPS 2015.

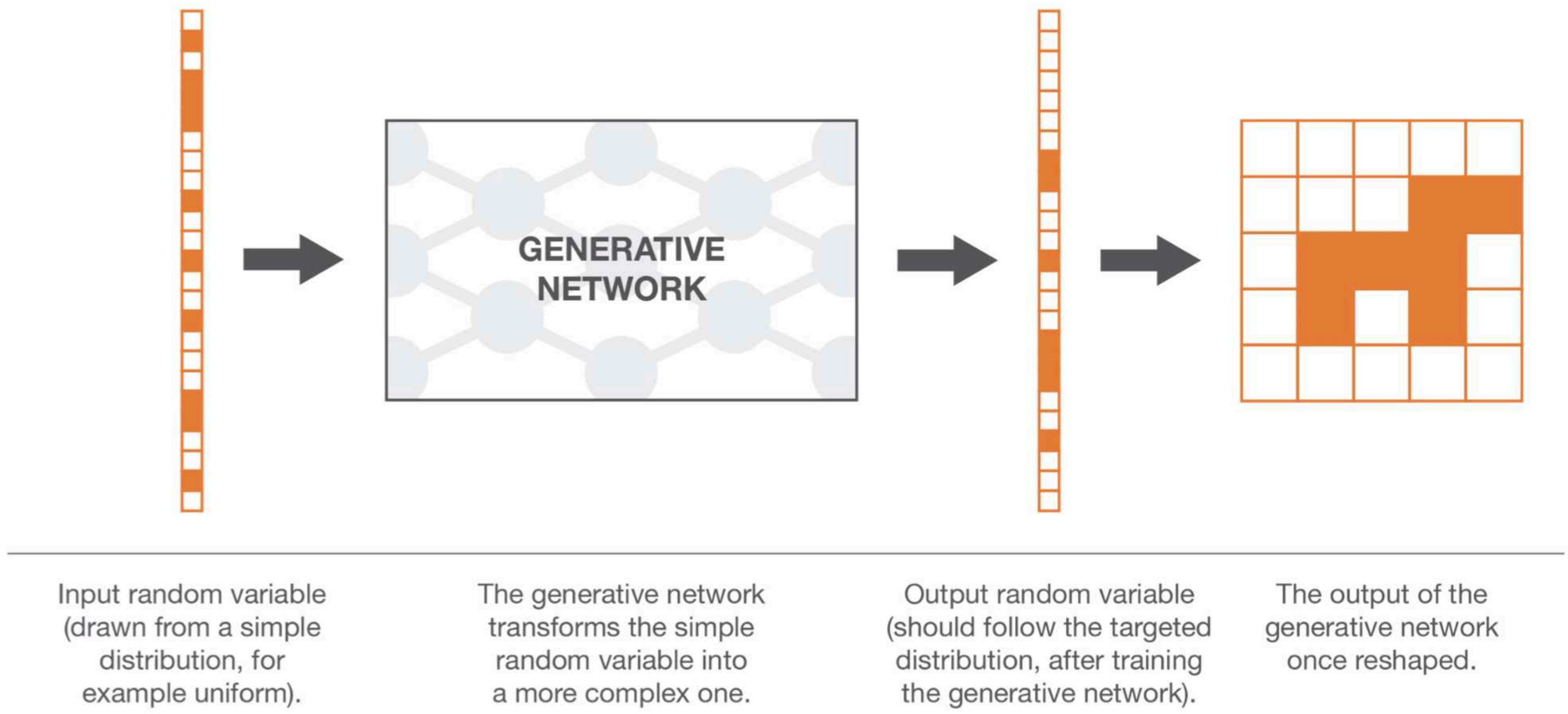
The Reparameterization Trick in VAE

$$p(z) \equiv \mathcal{N}(0, I)$$
$$p(x|z) \equiv \mathcal{N}(f(z), cI) \qquad f \in F \qquad c > 0$$

Let's forget about variational inference for maximizing log p(x) but focus on the probability distribution of p(x|z) itself, we can easily sample from p(x|z), which leads to a nice GENERATIVE model and transforms a simple Gaussian distribution to a complex data distribution $p_g(x)$ through a one-to-one mapping f: z → x

A direct approach to aligning our generated data distribution $p_g(x)$ with real data distribution $p_r(x)$ is to perform moment matching, for e.g., minimizing maximum mean discrepancy in a high-dimensional feature space induced by a kernel (kernel MMD).

Transform a Simple Distribution to a Complex Distribution



Picture Credit:https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

An Indirect Approach for Comparing Distributions

$$p(z) \equiv \mathcal{N}(0, I)$$
$$p(x|z) \equiv \mathcal{N}(f(z), cI) \qquad f \in F \qquad c > 0$$

- Transform a simple Uniform/Gaussian distribution p(z) to a complex data distribution $p_g(x)$ through a one-to-one mapping f: z → x

- An indirect approach is to assume that we have an oracle discriminator that can perfectly discriminates whether or not a data point is from the real data distribution. We can make use of this oracle discriminator to improve our generative network such that our generated data distribution perfectly aligns with the real data distribution.

- In practice, we don't have this oracle discriminator, but we can treat it as a deep neural network and learn it from data.

Generative Adversarial Network (GAN)

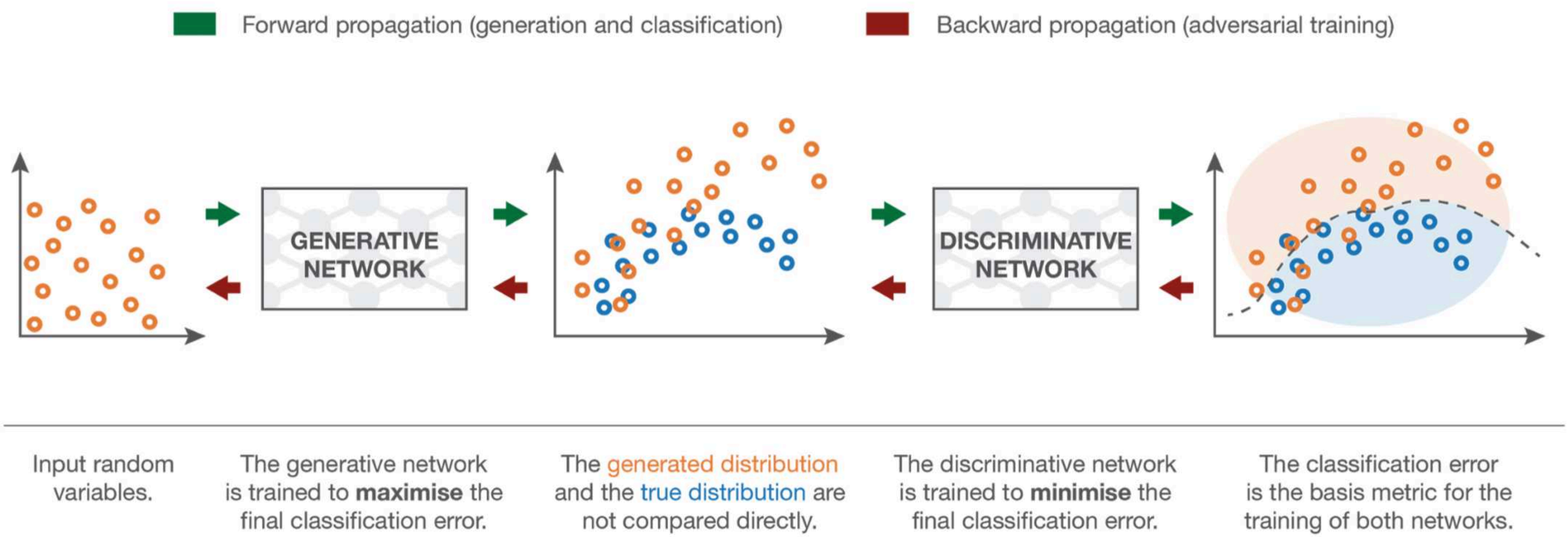
- The goal of the discriminator D is to discriminate whether a sample comes from the real data distribution (training data) or the generated data distribution (generated data).

- The goal of the generator G is to transform a simple (e.g., Gaussian, Uniform) distribution to a real data distribution such that the generated sample will fool the discriminator.

- This is a minmax two-player game. In a global optimum, D will output ½ everywhere and $p_g(x) = p_r(x)$

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

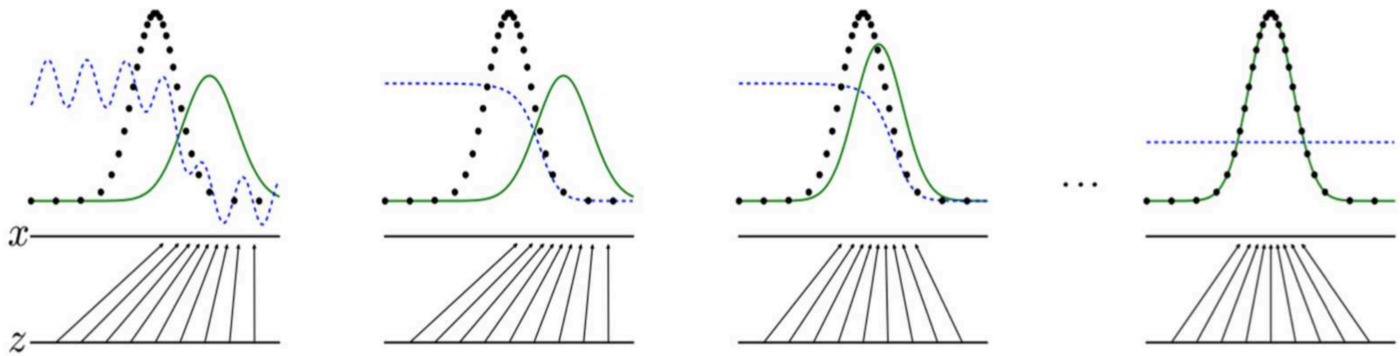
Generative Adversarial Network (GAN)



Picture Credit:https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Generative Adversarial Network (GAN)



(a) (b) (c) (d)

Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Optimal D of Generative Adversarial Networks

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

if $y = a\log(y) + b\log(1-y)$, the optimal y is

$$\Longrightarrow y^* = \frac{a}{a+b}$$

$$y = a\log(y) + b\log(1-y)$$
$$y' = \frac{a}{y} - \frac{b}{1-y}$$
$$\frac{a}{y^*} = \frac{b}{1-y^*}$$

Find optimal y^* by setting $y' = 0$.

$$\frac{1-y^*}{y^*} = \frac{b}{a}$$
$$\frac{1}{y^*} = \frac{a+b}{a}$$
$$y^* = \frac{a}{a+b}$$

Optimize $D(x) = p_r(x)\log D(x) + p_g(x)\log(1 - D(x))$, we get

$$\Longrightarrow D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Optimal Solution of Generative Adversarial Networks

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

With $p = q$, ***the optimal value for D and V is***

$$D^*(x) = \frac{p}{p+q} = \frac{1}{2}$$

$$\begin{aligned}\min_G \max_D V(D, G) &= \mathbb{E}_{x \sim p_r(x)}[\log \frac{1}{2}] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - \frac{1}{2})] \\ &= -2 \log 2\end{aligned}$$

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Training Algorithm of GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

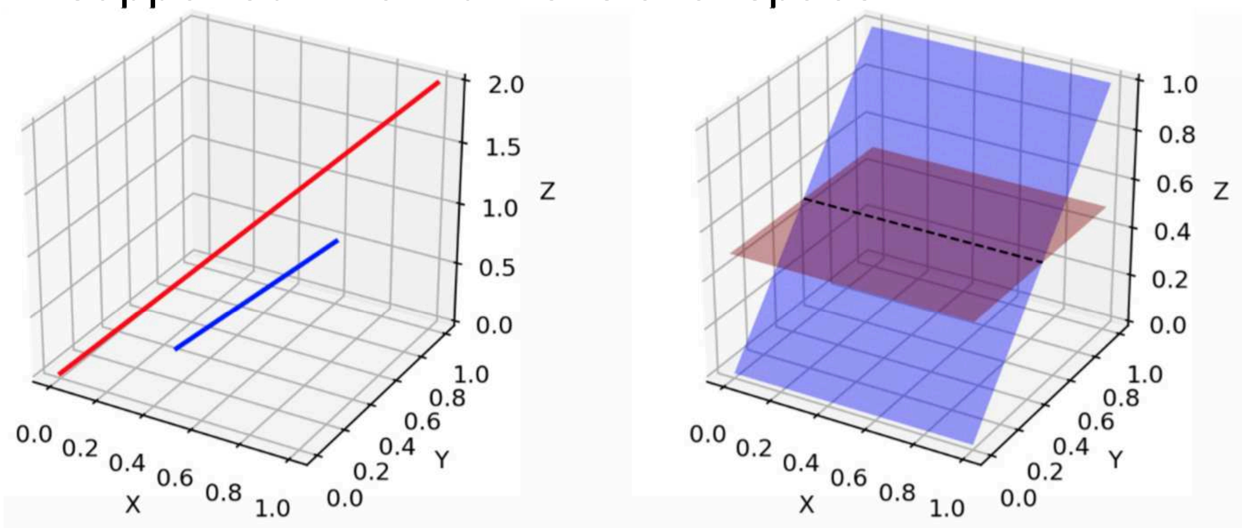
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Real Image/Video Data is often Supported in a Low-D Manifold

For e.g. MNIST digits, ImageNet Images, Videos, although the pixel space is very high-dimensional.

It's easy to find a perfect discriminator to separate high-dimensional data supported in low-dimensional space.



$$-\nabla_{\theta_g} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \rightarrow \boldsymbol{0}$$

orignal GAN generator's gradient

Picture Credit:https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

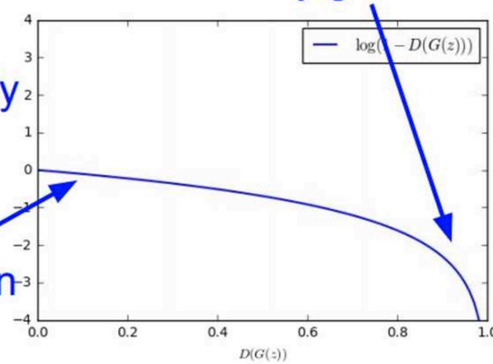
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Slide Credit: Fei-Fei Li, Justin Johnson, and Serena Yeung, cs231n 2017

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

Alternate between:

1. **Gradient ascent** on discriminator

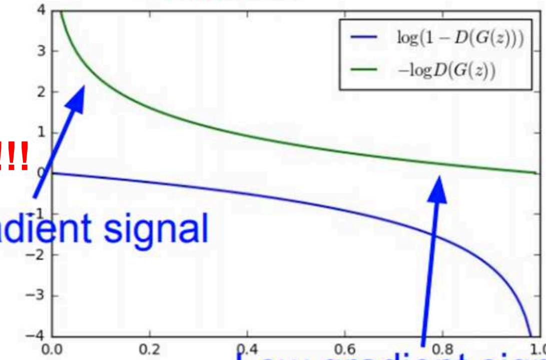
$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

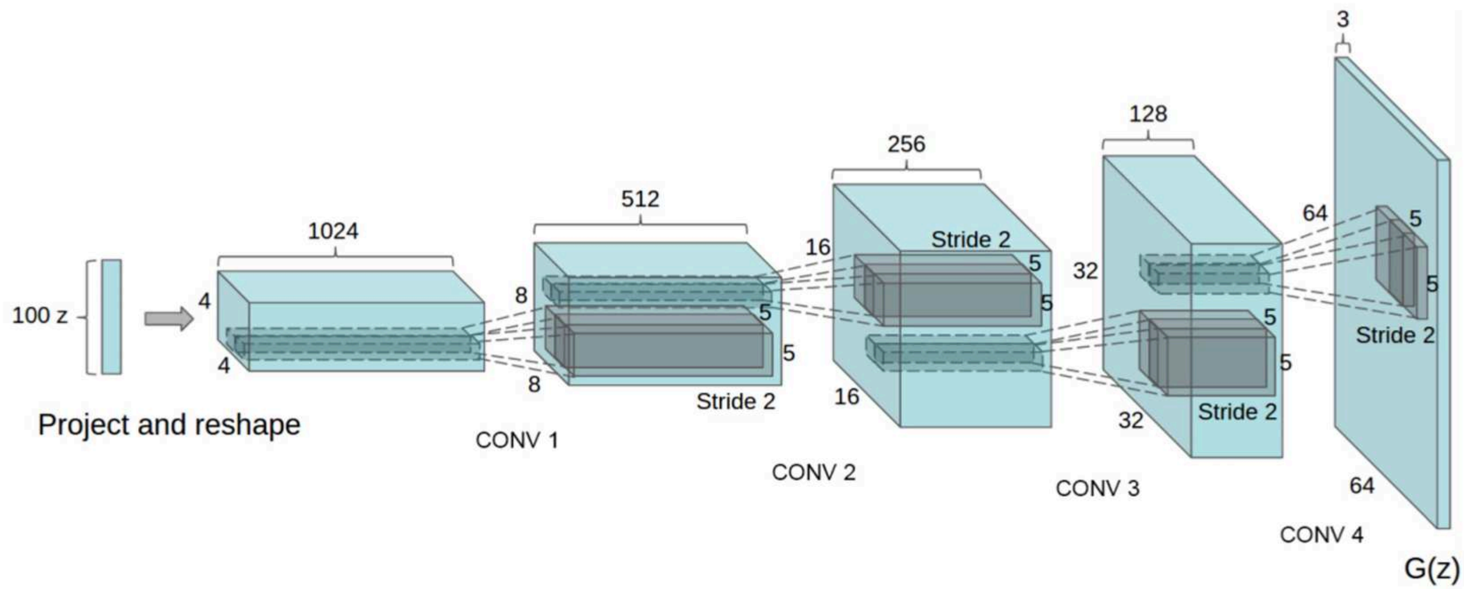
This is unstable with large variance of gradient!!!

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Slide Credit: Fei-Fei Li, Justin Johnson, and Serena Yeung, cs231n 2017

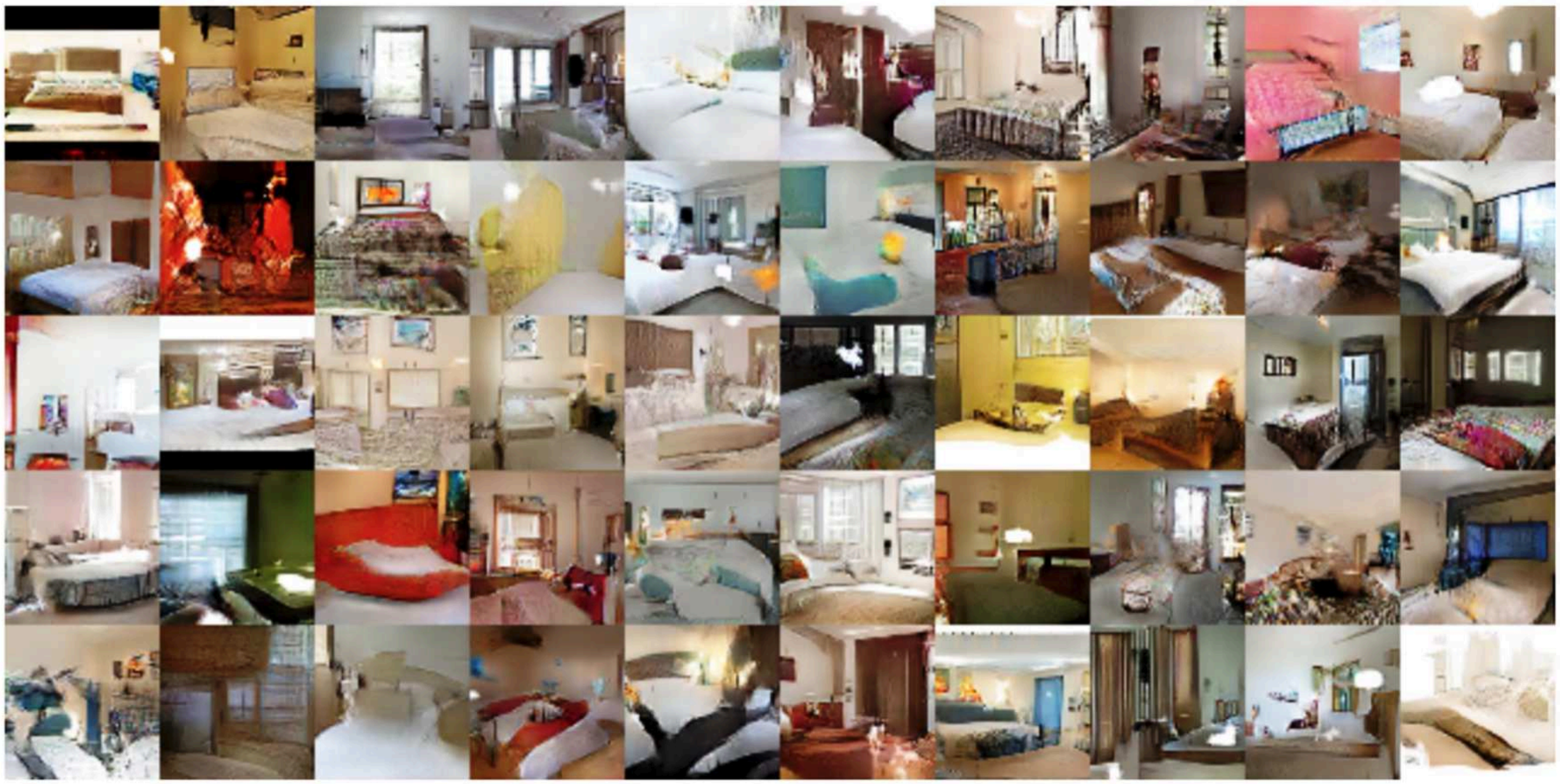
Deep Convolutional GAN (DCGAN): CNN Generator



DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

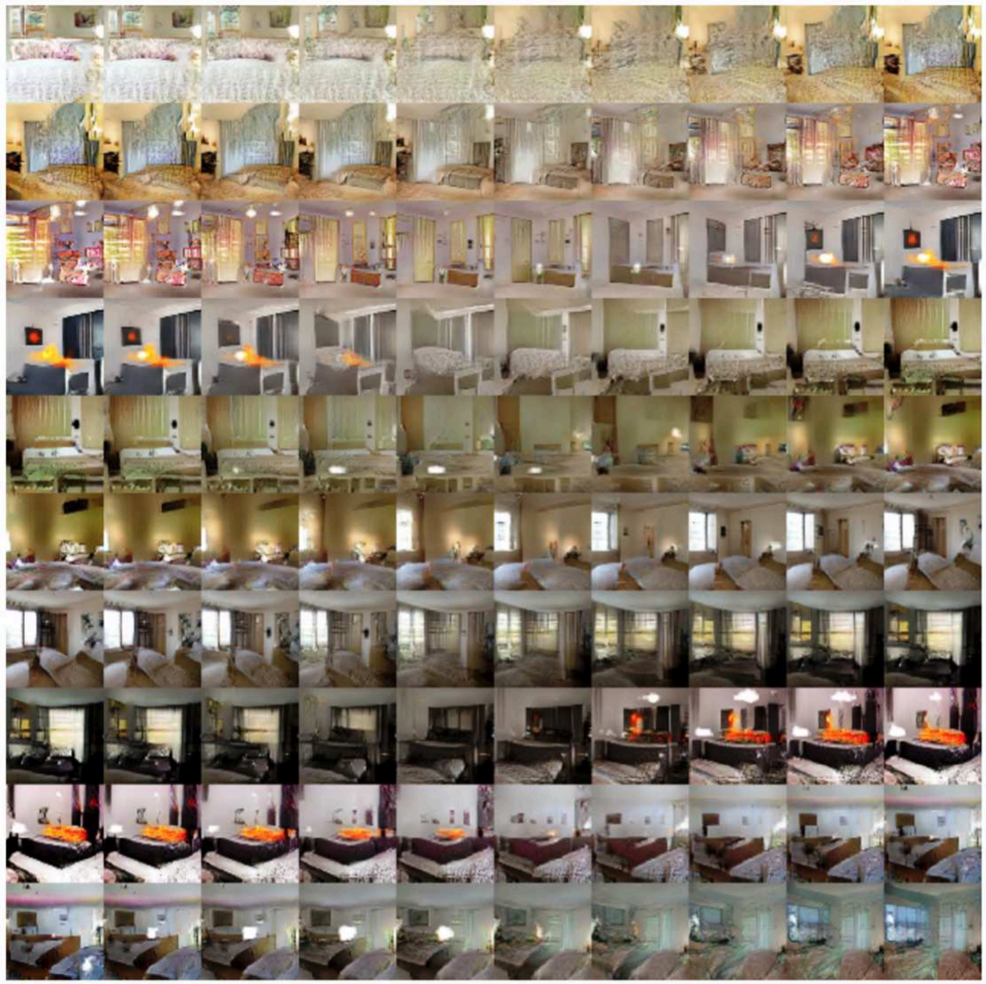
Radford *et al*., UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS. ICLR 2016

Generated Samples of DCGAN

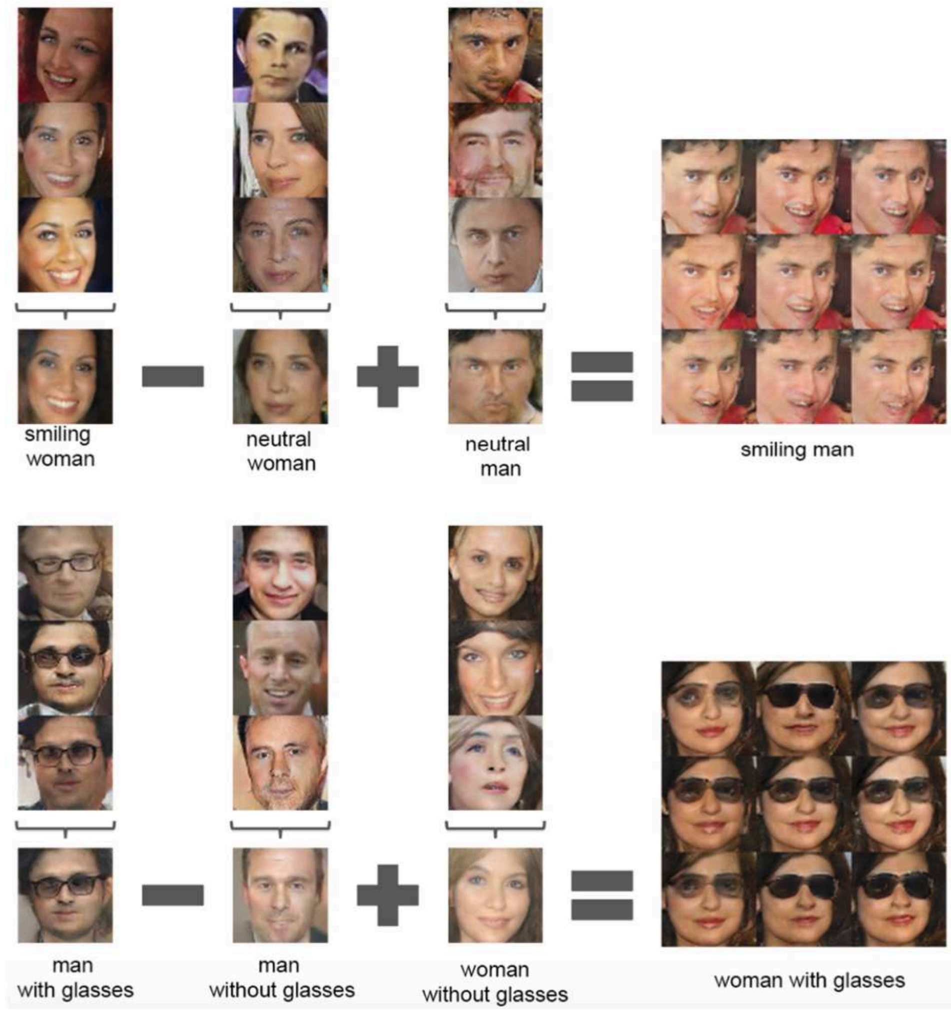


Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

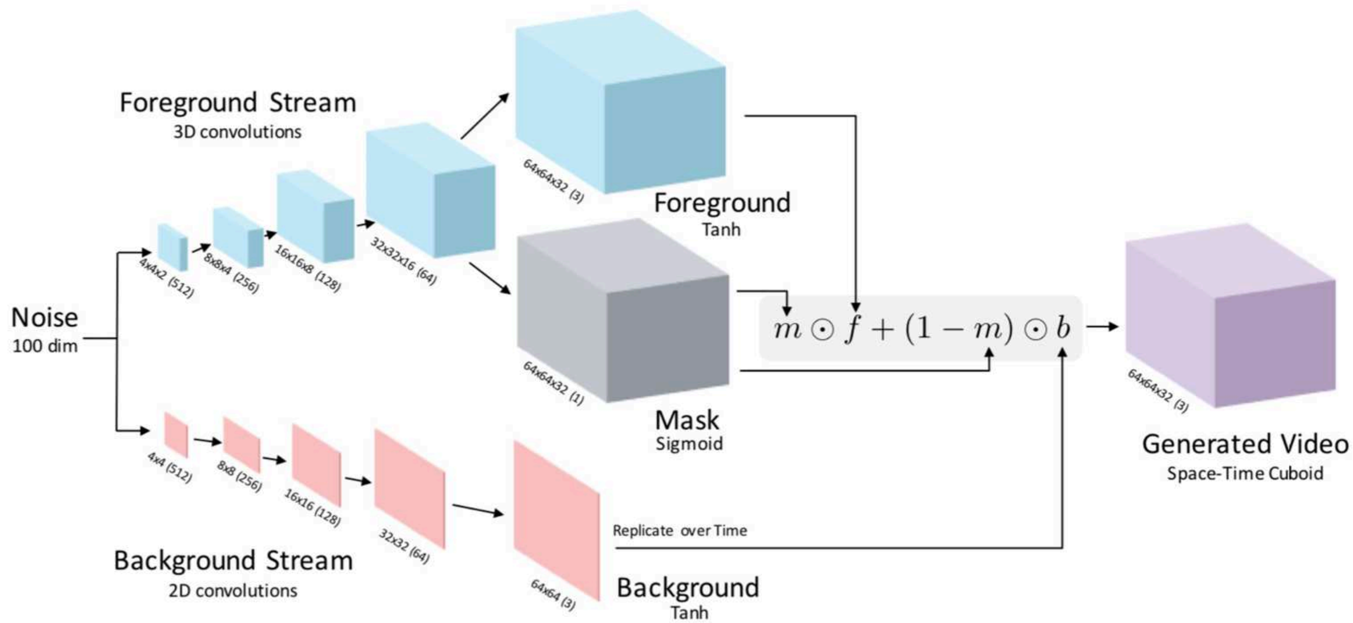
Interpolation Results of DCGAN



Latent Vector (z) Manipulation Results of DCGAN



GAN for Video Generation



Video Generator Network: We illustrate our network architecture for the generator. The input is 100 dimensional (Gaussian noise). There are two independent streams: a moving foreground pathway of fractionally-strided spatio-temporal convolutions, and a static background pathway of fractionally-strided spatial convolutions, both of which up-sample. These two pathways are combined to create the generated video using a mask from the motion pathway. Below each volume is its size and the number of channels in parenthesis.

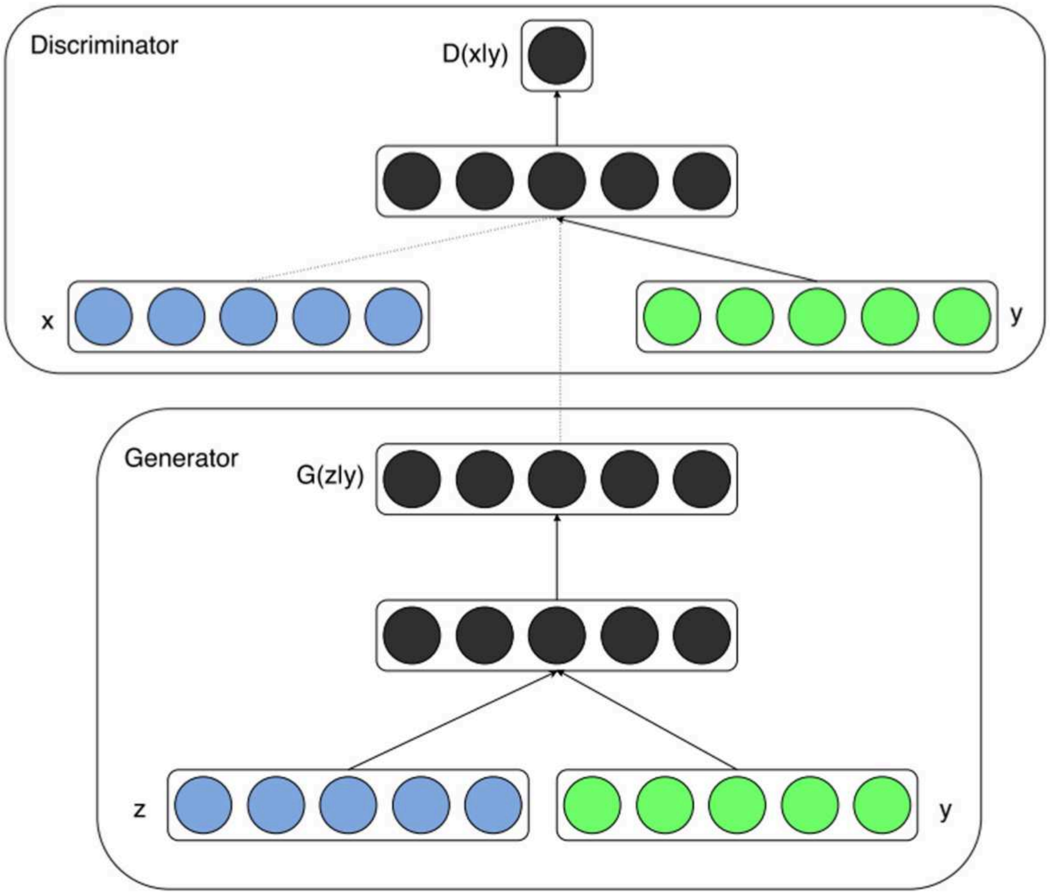
Vondrick et al., Generating Videos with Scene Dynamics, NIPS 2016.

GAN for Music Generation

Engel et al., GANSYNTH: ADVERSARIAL NEURAL AUDIO SYNTHESIS. ICLR 2019. https://openreview.net/pdf?id=H1xQVn09FX

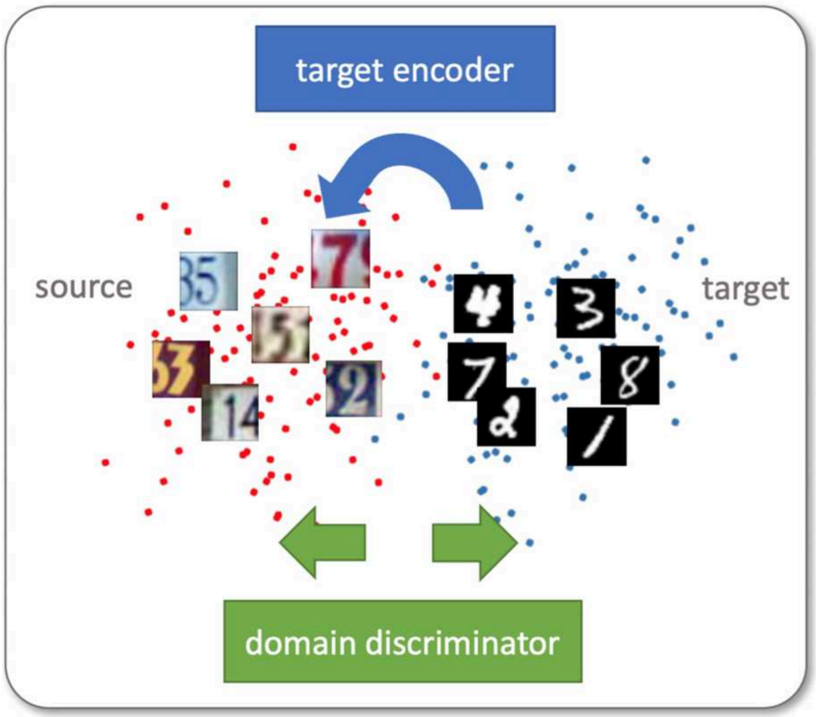
Generated Music Samples: https://magenta.tensorflow.org/gansynth

Conditional GAN



https://arxiv.org/pdf/1411.1784.pdf

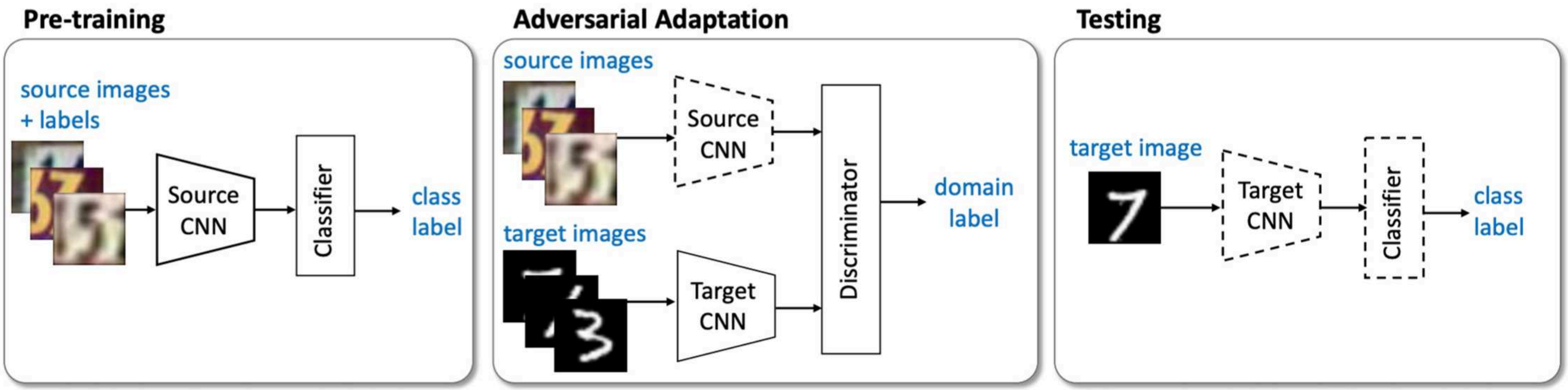
Domain Adaptation



We have a lot of (labeled) training data in a source domain, and we plan to deploy our learned model in the source domain to a target domain that has a different data distribution from the one in the source domain.

Picture Credit: Tzeng *et al.*, Adversarial Discriminative Domain Adaptation, CVPR 2017.

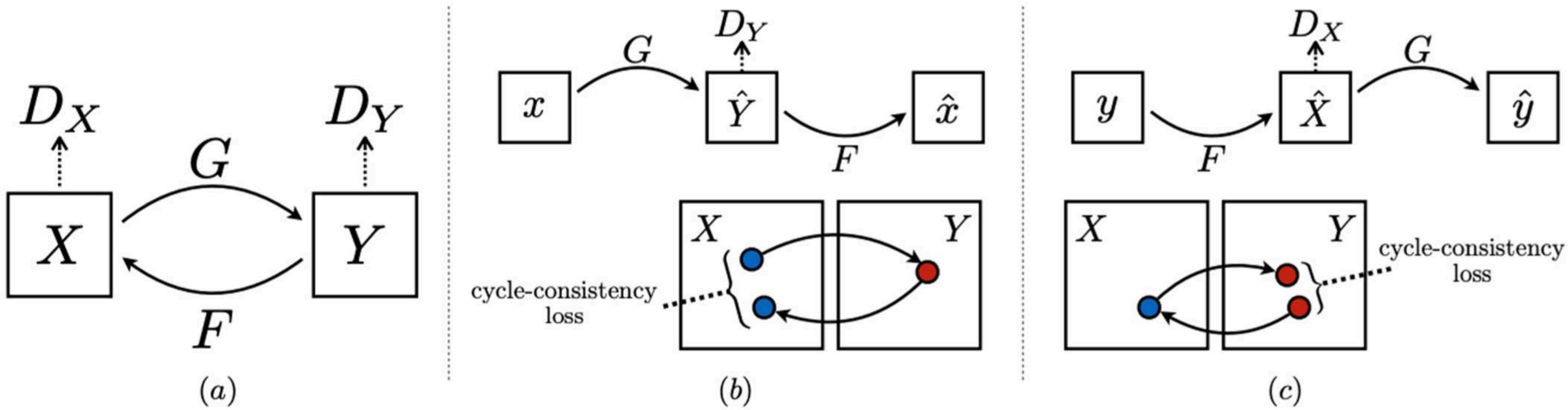
Adversarial Feature Learning for Domain Adaptation



An overview of our proposed Adversarial Discriminative Domain Adaptation (ADDA) approach. We first pre-train a source encoder CNN using labeled source image examples. Next, we perform adversarial adaptation by learning a target encoder CNN such that a discriminator that sees encoded source and target examples cannot reliably predict their domain label. During testing, target images are mapped with the target encoder to the shared feature space and classified by the source classifier. Dashed lines indicate fixed network parameters.

Tzeng *et al.*, Adversarial Discriminative Domain Adaptation, CVPR 2017.

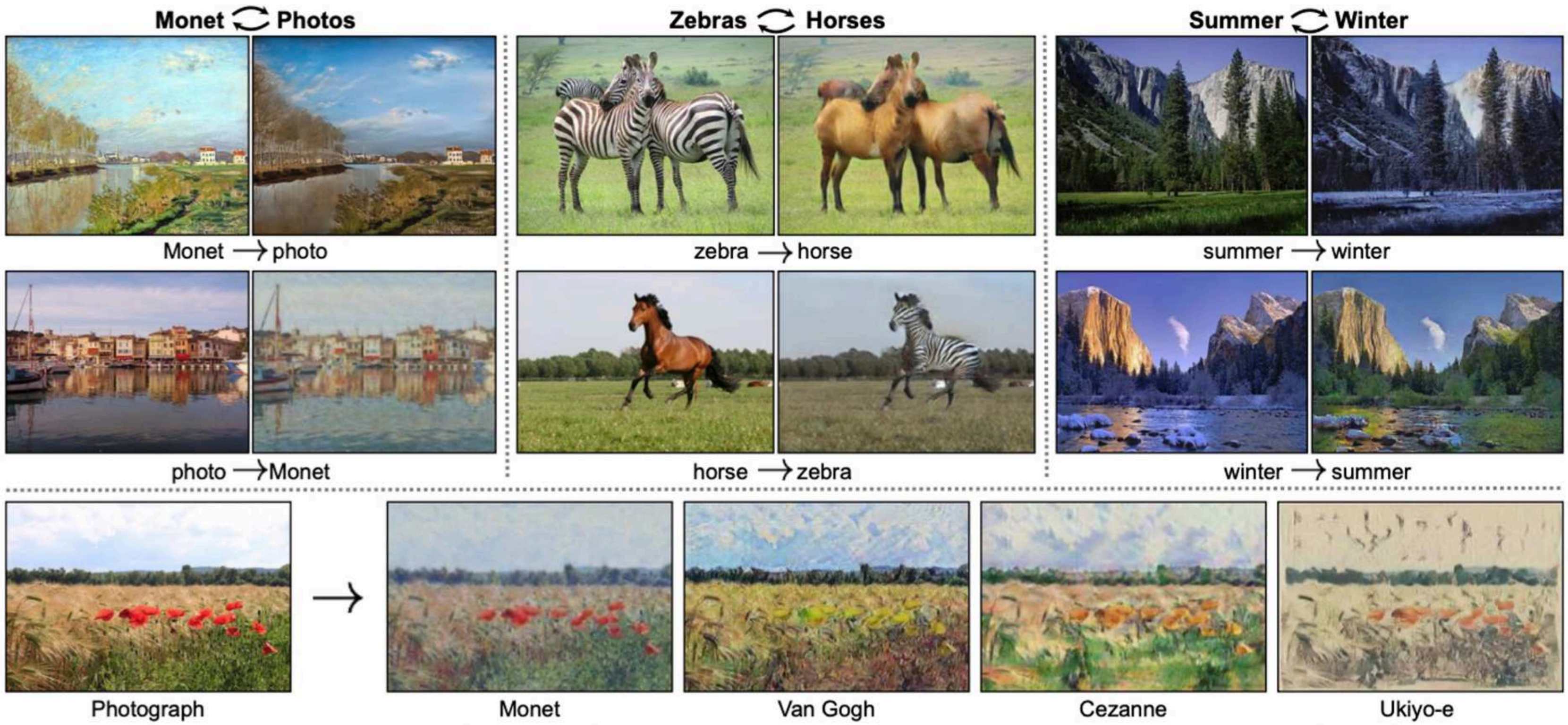
CycleGAN



(a) Our model contains two mapping functions $G : X \to Y$ and $F : Y \to X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \to G(x) \to F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \to F(y) \to G(F(y)) \approx y$

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.

CycleGAN Results



Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017.

Text2Video: Goals and Challenges

Build a conditional generative model to generate videos from text capturing different contextual semantics of natural language descriptions

Capable of capturing both static content and dynamic motion features of videos

Challenges

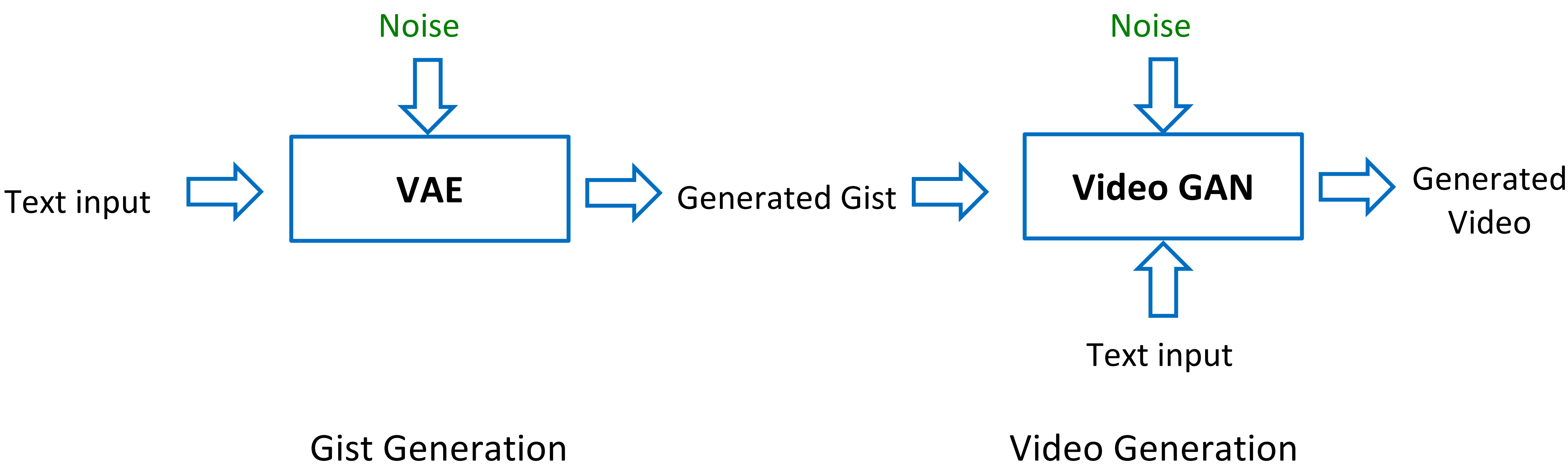
- It's hard to condition on text, a big gap
- It is hard to build powerful video generator
- No publicly available dataset

How? Integrating VAE and GAN

https://www.cs.toronto.edu/~cuty/Text2VideoAAAI2018.pdf

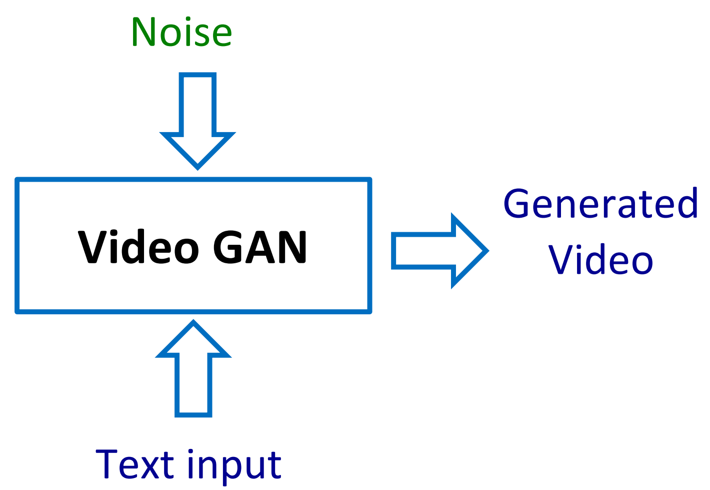
Model Overview

- We introduce an intermediate step called 'Gist' Generation.
- The model is trained end-to-end.

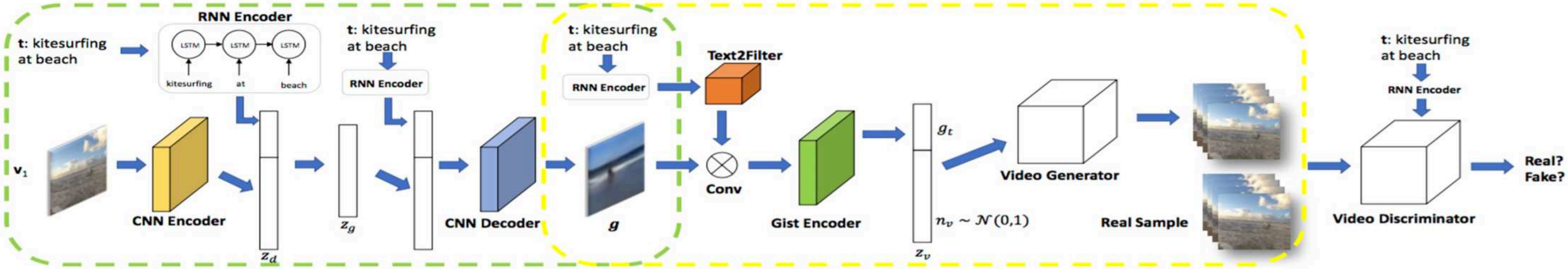


What does the Gist do?

- Gist captures the static features of a video.
- Gist generation gives a sketch.



The Complete Text2Video Model



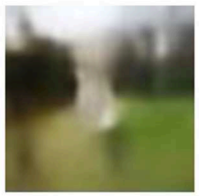
Framework of the proposed text-to-video generation method. The gist generator is within the green box. The encoded text is concatenated with the encoded frame to form the joint hidden representation z_d, which is further transformed into z_g. The video generator is within the yellow box. The text description is transformed into a filter kernel (Text2Filter) and applied to the gist. The generation uses the feature z_g. Following this point, the flow chart forms a standard GAN framework with a final discriminator to judge whether a video and text pair is real or synthetic. After training, the CNN image encoder is ignored.

Li, Min, Shen, and Lawrence, AAAI 2018
https://www.cs.toronto.edu/~cuty/Text2VideoAAAI2018.pdf

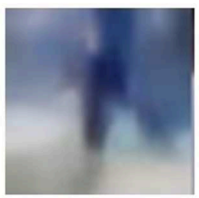
Generated Video Samples

Text input | **Generated gist** | **Generated video**

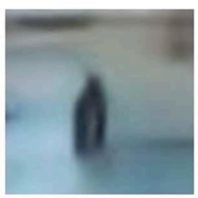
Play golf on grass



Play golf on snow



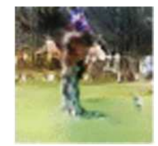
Play golf on water



More Examples

Text: Playing Golf on | Gist | Video

– grass field



– snow

– water

More Examples

Playing golf

Playing golf in swimming pool

Swimming in swimming pool

Sailing on the sea

Sailing on snow

Sailing on grass

Running on the sea

Running on sand

More Examples

Kitesurfing on the sea

Kitesurfing on grass

An Improved Text2Video Model

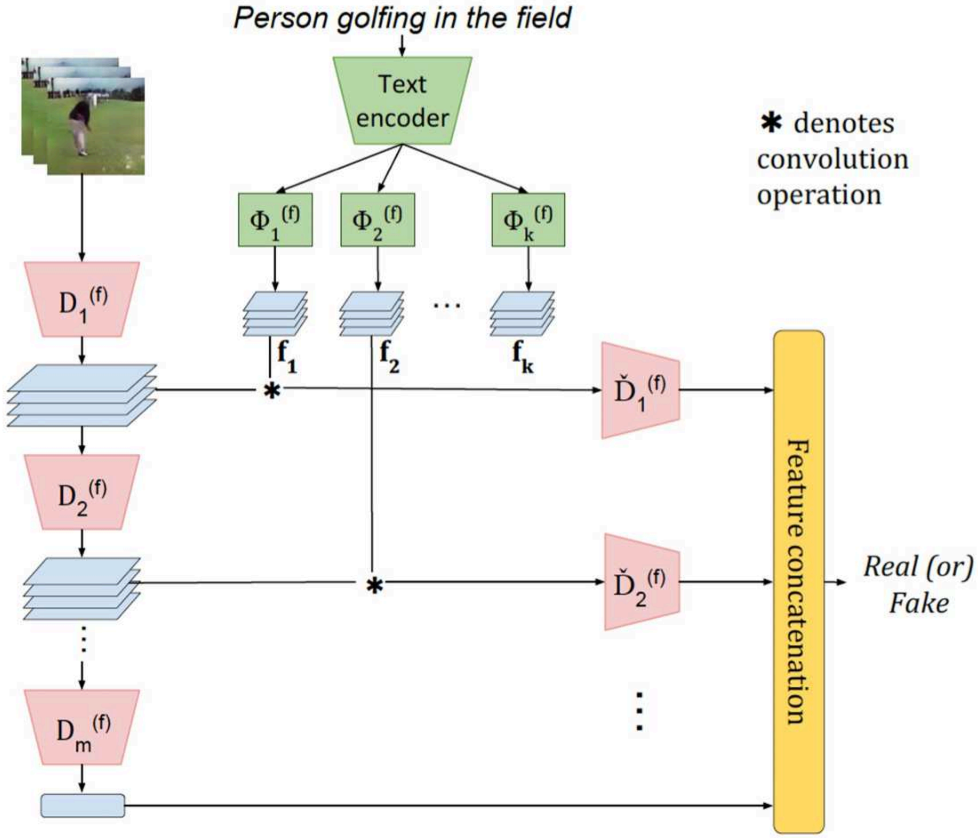


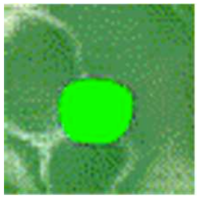
Illustration of our Text-Filter conditioning strategy.

Balaji, Min, Bai, Chellappa, and Graf. Conditional GAN with Discriminative Filter Generation for Text-to-Video Synthesis. IJCAI 2019.

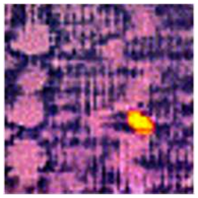
Generated Videos

TFGAN

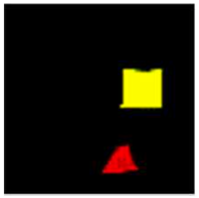
Baseline



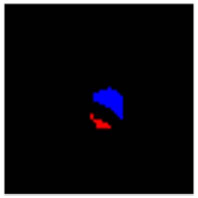
A large green circle is moving in a zigzag path towards east



A large yellow square is moving in a diagonal path in the northeast direction

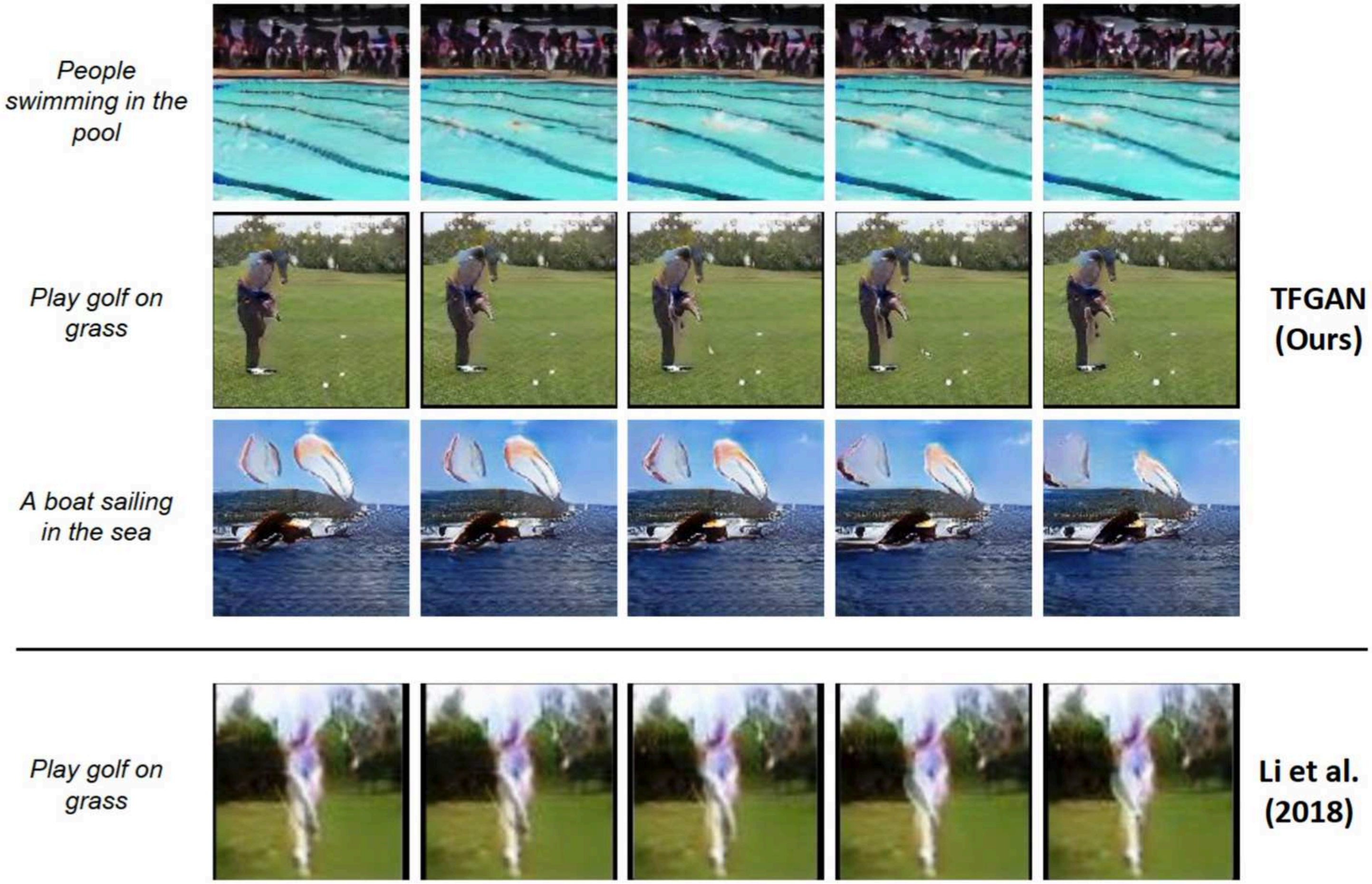


A large red triangle is moving in a straight line towards north and a large yellow square is moving in a zigzag path towards west



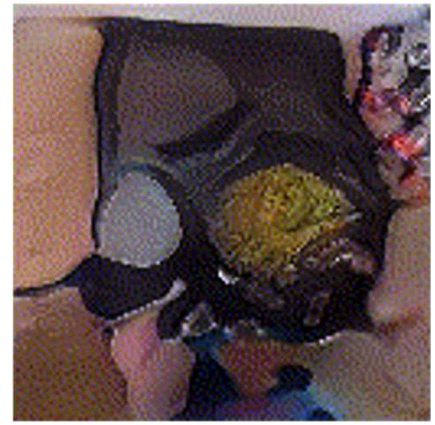
A large red triangle is moving in a zigzag path towards south and a large blue triangle is moving in a zigzag path towards west

Generated Videos



Previous Model

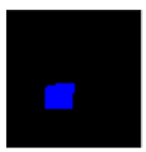
Generated Videos

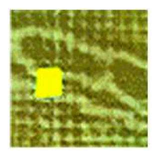


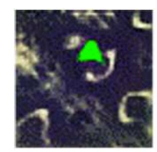
People swimming in pool

Person skiing in ocean

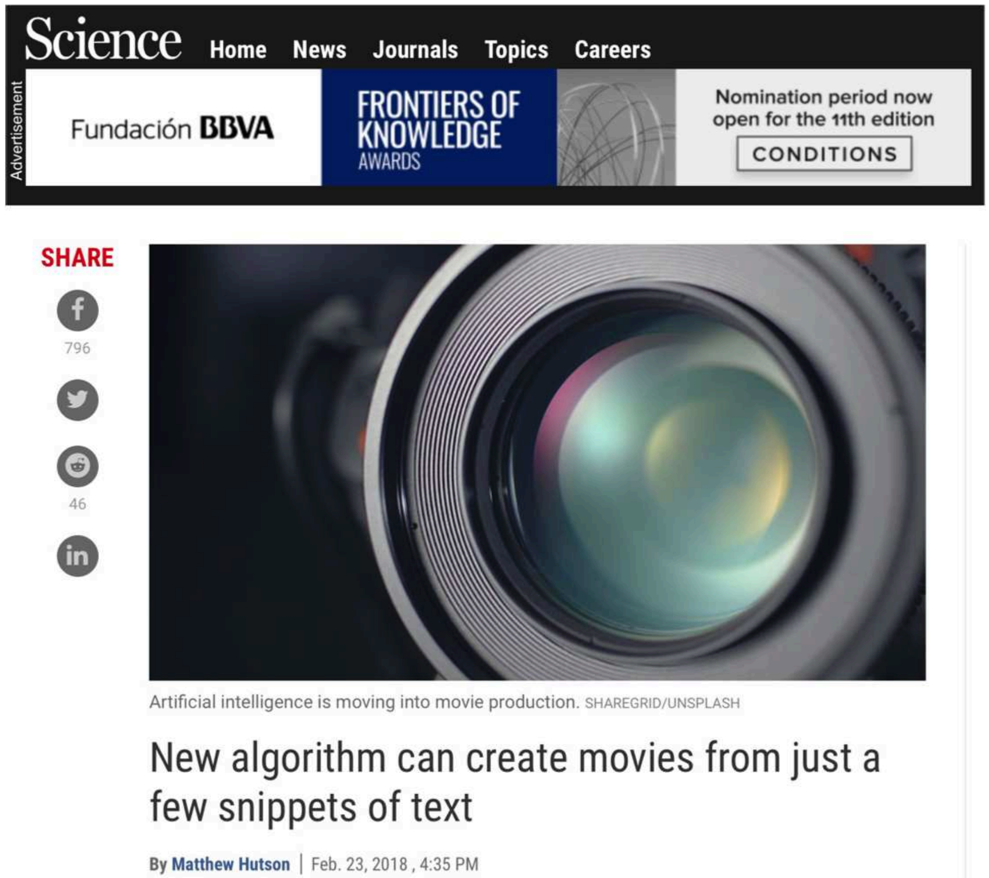
Stir vegetables







Media Reports from Science, MIT Technology Review, Communications of ACM, etc.



Li, Min, *et al.*, AAAI 2018

Problems of GAN

The minmax training of GAN doesn’t necessarily converge in practice:

If we have a perfect discriminator in the beginning, the gradient of the loss function with respect to generator parameters is close to zero and the learning is very slow

If we have a very bad discriminator, we don’t get much useful feedback from the discriminator.

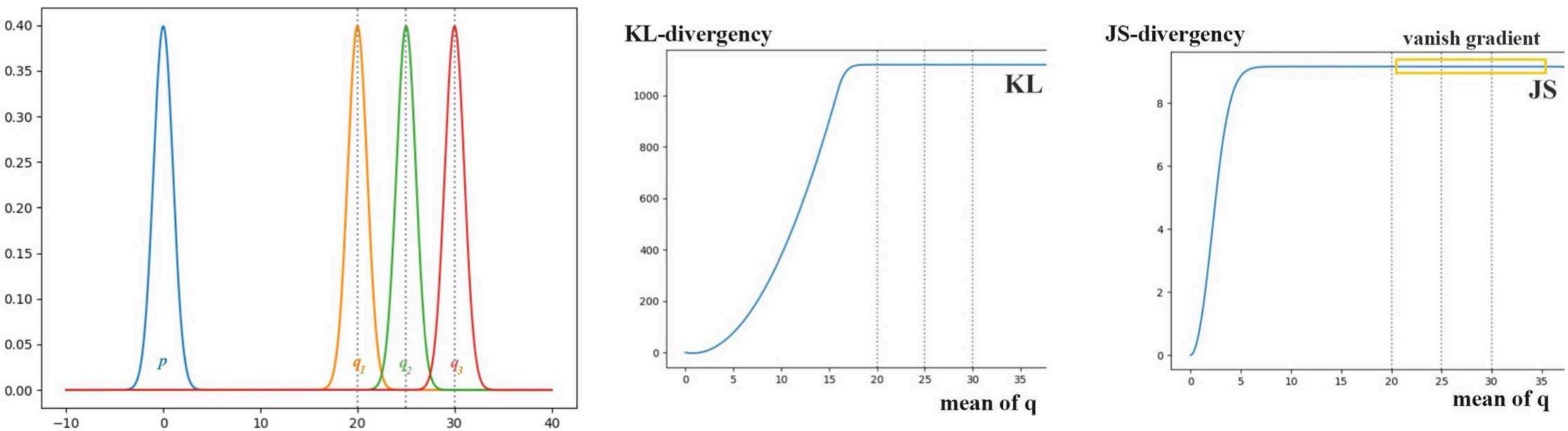
Training can be unstable.

Mode collapse: the generator only generates a subset of training data distribution modes to fool the discriminator and fails to explore other modes.

GAN Minimizes JS-Divergence to Update G

$$D_{KL}(P||Q) = \sum_{x=1}^{N} P(x) \log \frac{P(x)}{Q(x)}$$ for VAE

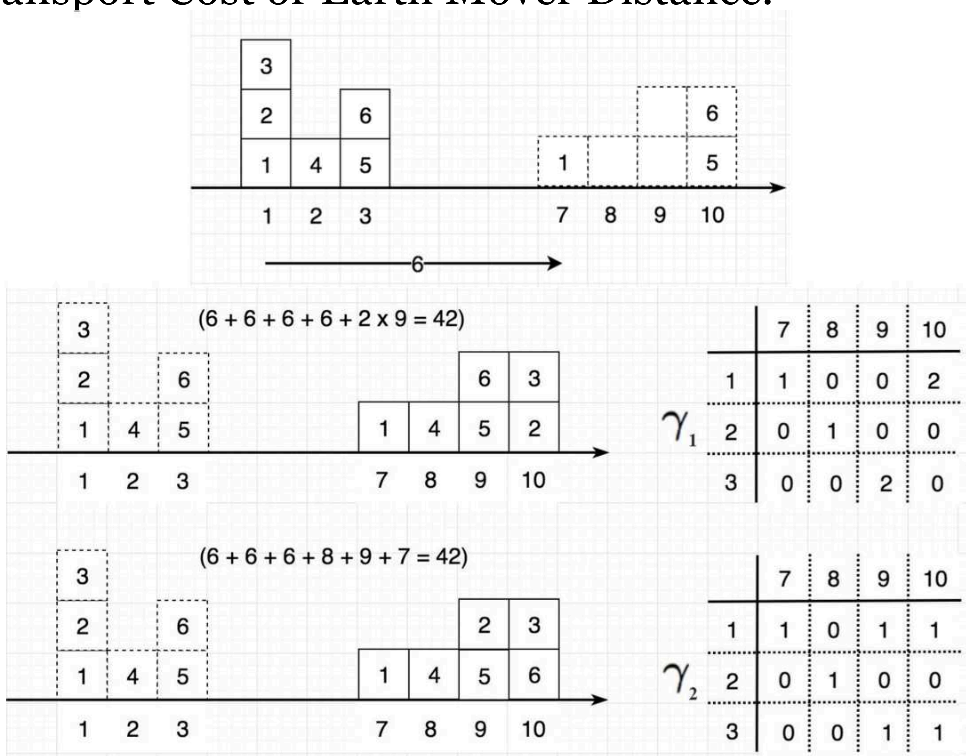
$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$ for GAN



Picture Credit: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Wasserstein Distance

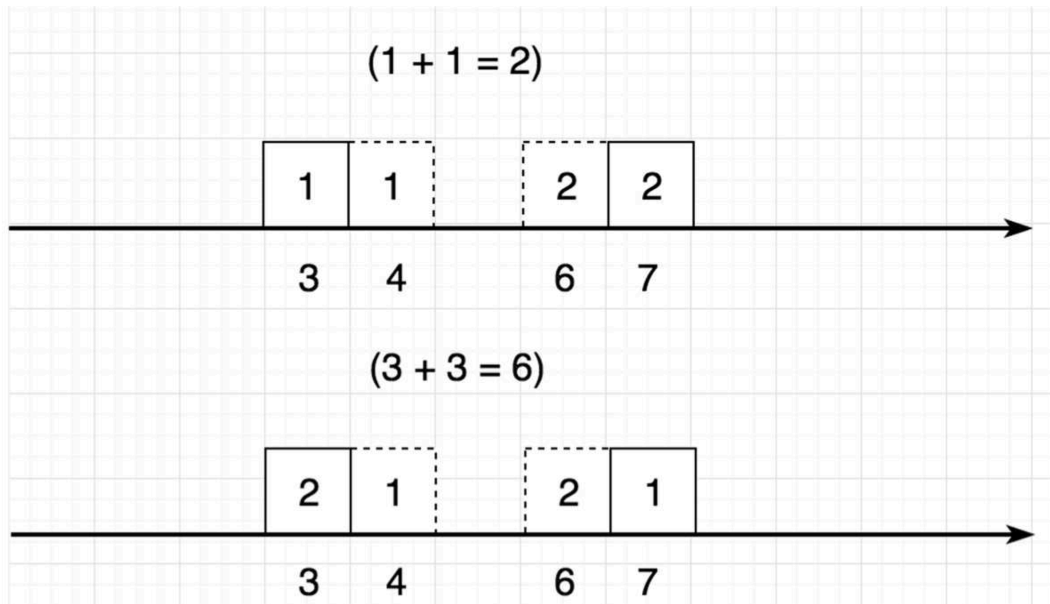
The Wasserstein distance of $\boldsymbol{p}$ and $\boldsymbol{q}$ is the minimum cost of transporting mass in converting the shape of a data distribution $\boldsymbol{q}$ to the shape of a data distribution $\boldsymbol{p}$. It is also called Optimal Transport Cost or Earth Mover Distance.



Picture Credit: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Wasserstein Distance

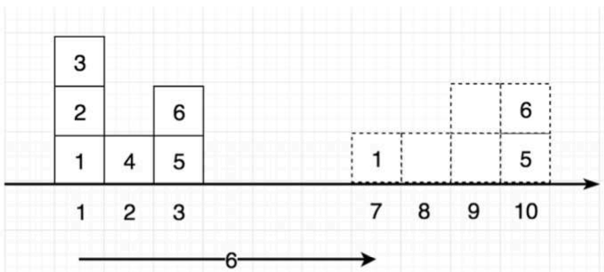
The Wasserstein distance of $\boldsymbol{p}$ and $\boldsymbol{q}$ is the minimum cost of transporting mass in converting the shape of a data distribution $\boldsymbol{q}$ to the shape of a data distribution $\boldsymbol{p}$. It is also called Optimal Transport Cost or Earth Mover Distance.



Picture Credit: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

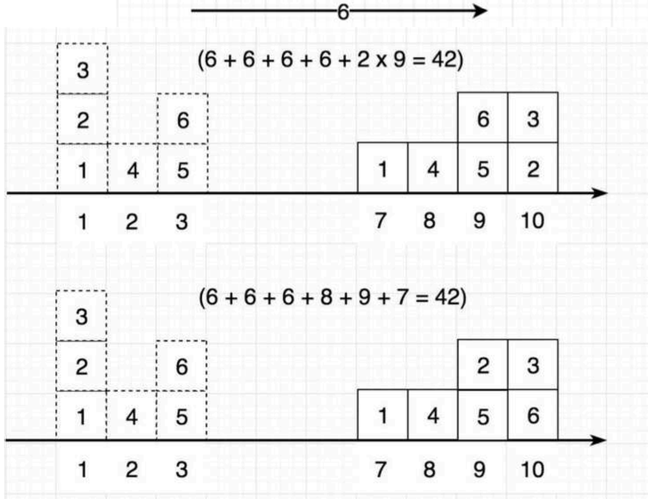
Wasserstein Distance

The Wasserstein distance of $\boldsymbol{p}$ and $\boldsymbol{q}$ is the minimum cost of transporting mass in converting the shape of a data distribution $\boldsymbol{q}$ to the shape of a data distribution $\boldsymbol{p}$. It is also called Optimal Transport Cost or Earth Mover Distance.



$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big] \,,$$

Π(Pr, Pg) denotes the set of all joint distributions γ(x, y) whose marginals are respectively Pr and Pg.



γ_1

	7	8	9	10
1	1	0	0	2
2	0	1	0	0
3	0	0	2	0

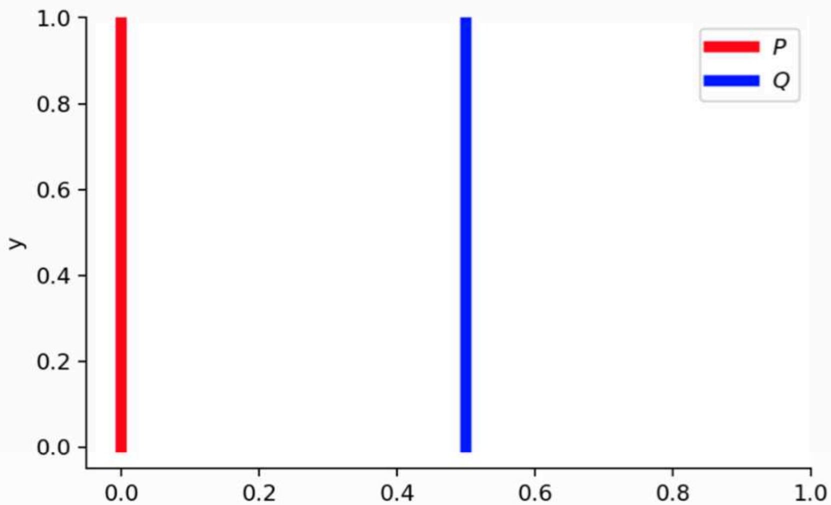
γ_2

	7	8	9	10
1	1	0	1	1
2	0	1	0	0
3	0	0	1	1

Picture Credit: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Comparing Wasserstein Distance with KLD and JSD

$$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0, 1)$$
$$\forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)$$



When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$
$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$
$$D_{JS}(P, Q) = \frac{1}{2}\Big(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\Big) = \log 2$$
$$W(P, Q) = |\theta|$$

But when $\theta = 0$, two distributions are fully overlapped:

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P, Q) = 0$$
$$W(P, Q) = 0 = |\theta|$$

Picture Credit:https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

Wasserstein GAN (WGAN) Minimizing Wasserstein Distance between p_g and p_r

Using the Kantorovich-Rubinstein duality, we can simplify the calculation to

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

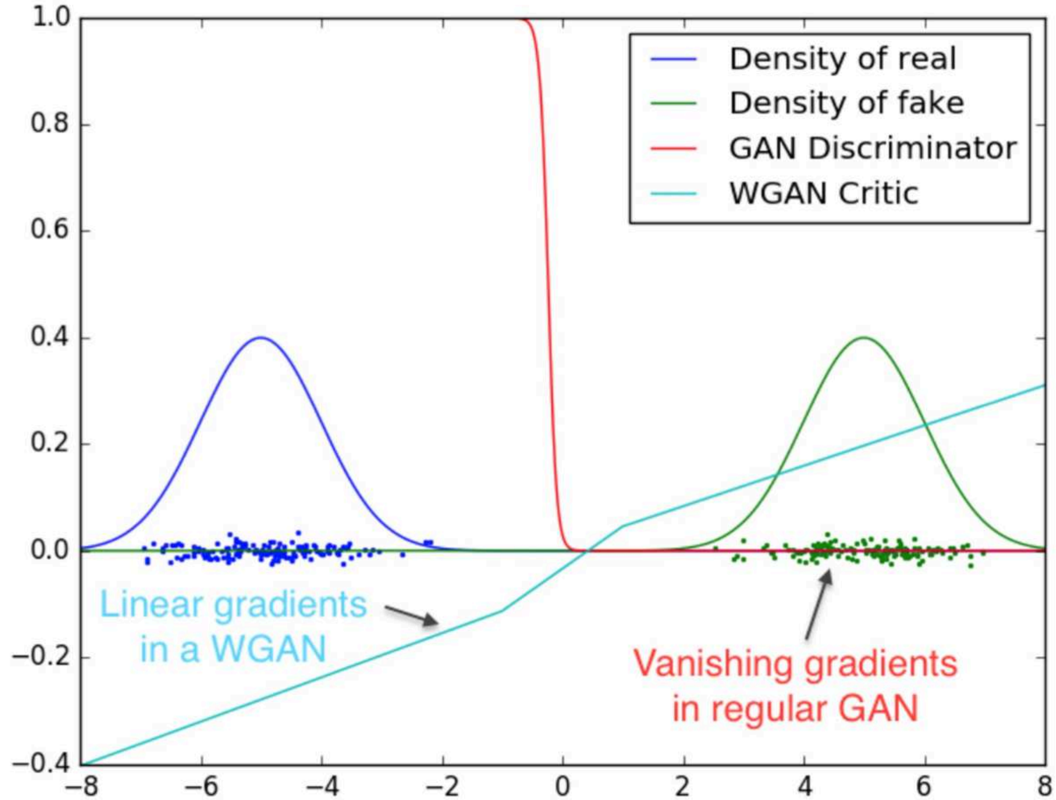
Arjovsky *et al.*, Wasserstein Generative Adversarial Networks. ICML 2017.

WGAN vs. GAN

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right)\right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right)$

In WGAN, we have a critic with a scalar output without log

WGAN vs. GAN



Arjovsky *et al.,* Wasserstein Generative Adversarial Networks. ICML 2017.

Training Algorithm of WGAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
    for $t = 0, ..., n_{\text{critic}}$ **do**
        Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
        Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
        $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
        $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
        $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

The Latest GAN Architectures - StyleGAN2 & StyleGAN-XL

Figure 11. Four hand-picked examples illustrating the image quality and diversity achievable using StylegGAN2 (config F).

https://arxiv.org/abs/1912.04958

https://arxiv.org/pdf/2202.00273.pdf

Denoising Diffusion Probabilistic Models and Flows (Not Covered)

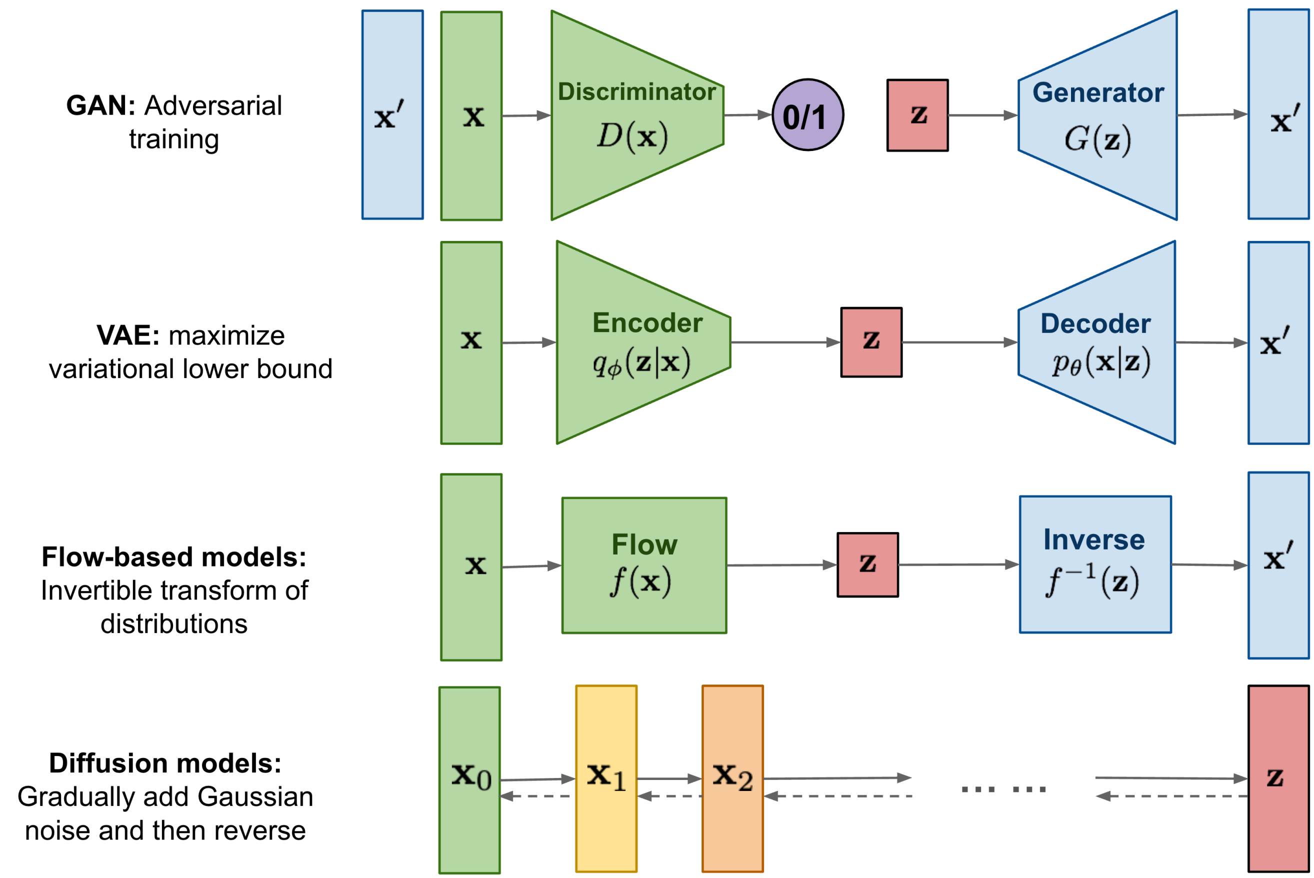


Image Credit: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Summary of Topics Discussed

- Attention: Transformer
- VAE
- GAN
- Adversarial Domain Adaptation, CycleGAN
- Text2Video Synthesis
- Wasserstein GAN

The End

Thank You!