Deep Learning

Ulf Brefeld

build: June 25, 2024

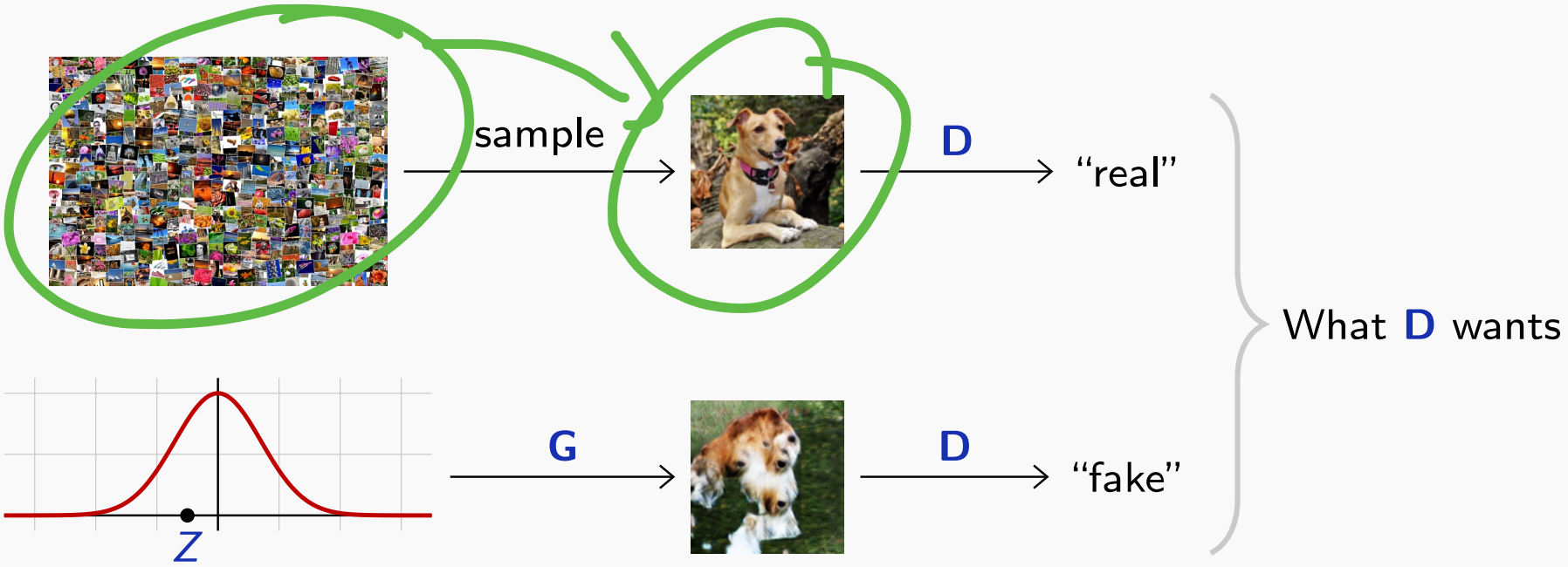
Machine Learning Group
Leuphana University of Lüneburg

Generative Adversarial Networks

A different approach to learn high-dimensional generative models are Generative Adversarial Networks (Goodfellow et al., 2014)

The idea behind GANs is to train two networks simultaneously:

- A discriminator $\mathbf{D}$ to classify samples as *real* or *fake*
- A generator $\mathbf{G}$ to create samples that fool $\mathbf{D}$



It is adversarial since the two networks have antagonistic objectives.

A bit more formally, let $\mathcal{X}$ be the signal space and D the dimension of the latent space

- The generator is trained so that [ideally] if it gets a random normal-distributed Z as input, it produces a sample following the data distribution as output

$$\mathbf{G} : \mathbb{R}^D \to \mathcal{X}$$

- The discriminator is trained so that if it gets a sample as input, it predicts if it is genuine.

$$\mathbf{D} : \mathcal{X} \to [0, 1]$$

If $\mathbf{G}$ is fixed, to train $\mathbf{D}$ given a set of real examples

$$x_n \sim \mu, n = 1, \ldots, N$$

we create

$$z_n \sim \mathcal{N}(0, 1), 1 \leq n \leq N$$

build a binary data-set

$$\mathcal{D} = \Big\{ \underbrace{(x_1, 1), \ldots, (x_N, 1)}_{\text{real samples} \sim p_X}, \underbrace{(\mathbf{G}(z_1), 0), \ldots, (\mathbf{G}(z_N), 0))}_{\text{fake samples} \sim p_G} \Big\}$$

and minimize the binary cross-entropy

$$E(\mathbf{D}) = -\frac{1}{2N} \left(\sum_{n=1}^{N} \log \mathbf{D}(x_n) + \sum_{n=1}^{N} \log(1 - \mathbf{D}(\mathbf{G}(z_n))) \right)$$

$$\approx -\frac{1}{2} \left(\hat{\mathbb{E}}_{X \sim p_X} [\log \mathbf{D}(X)] + \hat{\mathbb{E}}_{X \sim p_G} [\log(1 - \mathbf{D}(X))] \right)$$

where p_X is the true distribution of the data, and p_G is the distribution of $\mathbf{G}(Z)$ with $Z \sim \mathcal{N}(0, 1)$

The situation is slightly more complicated since we also want to optimize $\mathbf{G}$ to maximize $\mathbf{D}$'s loss.

Goodfellow et al. (2014) provide an analysis of the resulting equilibrium of that strategy.

$$E(\mathbf{D}) = \mathbb{E}_{X\sim\mu}\left[\log \mathbf{D}(X)\right] + \mathbb{E}_{X\sim\mu_G}\left[\log(1-\mathbf{D}(X))\right]$$

is high if $\mathbf{D}$ is doing a good job (low cross entropy), and low if $\mathbf{G}$ fools $\mathbf{D}$

Thus, the theoretically best $\mathbf{G}^*$ that fools any $\mathbf{D}$ is

$$\mathbf{G}^* = \operatorname*{argmin}_{\mathbf{G}} \max_{\mathbf{D}} V(\mathbf{D}, \mathbf{G})$$

Similarly, the optimal discriminator for a given generator $\mathbf{G}$ is

$$\mathbf{D}^*_{\mathbf{G}} = \operatorname*{argmax}_{\mathbf{D}} V(\mathbf{D}, \mathbf{G})$$

The joint objective becomes

$$\mathbf{G}^* = \operatorname*{argmin}_{\mathbf{G}} V(\mathbf{D}^*_{\mathbf{G}}, \mathbf{G})$$

We have

$$V(\mathbf{D}, \mathbf{G}) = \mathbb{E}_{X \sim \mu}[\log \mathbf{D}(X)] + \mathbb{E}_{X \sim \mu_G}[\log(1 - \mathbf{D}(X))]$$
$$= \int_X \mu(x) \log \mathbf{D}(x) + \mu_{\mathbf{G}}(x) \log(1 - \mathbf{D}(x)) dx$$

Since

$$\operatorname*{argmax}_d \mu(x) \log d + \mu_{\mathbf{G}}(x) \log(1 - d) = \frac{\mu(x)}{\mu(x) + \mu_{\mathbf{G}}(x)}$$

and

$$\mathbf{D}^*_{\mathbf{G}} = \operatorname*{argmax}_{\mathbf{D}} V(\mathbf{D}, \mathbf{G})$$

we get (if there is no additional regularization!)

$$\forall x,\ \mathbf{D}^*_{\mathbf{G}}(x) = \frac{\mu(x)}{\mu(x) + \mu_{\mathbf{G}}(x)}$$

$$d^* = \underset{D(x)}{\operatorname{argmax}} \underbrace{\mu(x)}_{A} \underbrace{\log D(x)}_{d} + \underbrace{\mu^{(2)}}_{B} \log(1 - \underbrace{D(x)}_{d})$$

$$\Rightarrow \frac{\partial}{\partial d} A \cdot \log(d) + B \cdot \log(1-d)$$

$$= A \frac{1}{d} + B \frac{1}{1-d}(-1)$$

$$= A \frac{1}{d} - B \frac{1}{1-d} \quad | \cdot d$$

$$= A - B \frac{d}{1-d}$$

$$= A - B \frac{1}{\frac{1}{d} - 1} \stackrel{!}{=} 0$$

$$\Rightarrow A = +B \frac{1}{\frac{1}{d} - 1} \quad | \cdot \left(\frac{1}{d} - 1\right)$$

$$\Rightarrow \frac{1}{\alpha} A - A = +B$$

$$\frac{1}{\alpha} A = A + B$$

$$\frac{1}{\alpha} = \frac{A+B}{A}$$

$$\alpha^* = \frac{A}{A+B}$$

$$\Rightarrow y(x) = \frac{\mu_x(x)}{\mu_x(x) + \mu_G(x)}$$

So, since

$$\forall x,\ \mathbf{D}^*_{\mathbf{G}}(x) = \frac{\mu(x)}{\mu(x) + \mu_{\mathbf{G}}(x)}$$

we have

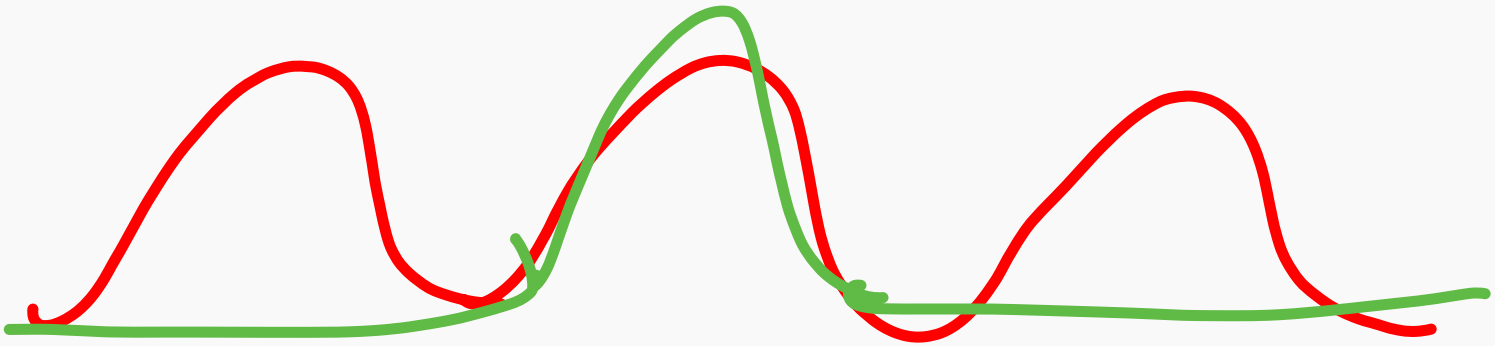
$$\begin{aligned}
V(\mathbf{D}^*_{\mathbf{G}}, \mathbf{G}) &= \mathbb{E}_{X\sim\mu}\left[\log \mathbf{D}^*_{\mathbf{G}}(X)\right] + \mathbb{E}_{X\sim\mu_{\mathbf{G}}}\left[\log(1 - \mathbf{D}^*_{\mathbf{G}}(X))\right] \\
&= \mathbb{E}_{X\sim\mu}\left[\log \frac{\mu(X)}{\mu(X) + \mu_{\mathbf{G}}(X)}\right] + \mathbb{E}_{X\sim\mu_{\mathbf{G}}}\left[\log \frac{\mu_{\mathbf{G}}(X)}{\mu(X) + \mu_{\mathbf{G}}(X)}\right] \\
&= KL\left(\mu \middle\| \frac{\mu + \mu_{\mathbf{G}}}{2}\right) + KL\left(\mu_{\mathbf{G}} \middle\| \frac{\mu + \mu_{\mathbf{G}}}{2}\right) - \log 4 \\
&= JSD(\mu, \mu_{\mathbf{G}}) - \log 4
\end{aligned}$$

where JSD is the Jensen-Shannon Divergence, a dissimilarity measure between distributions.

Training a standard GAN often results in two pathological behaviors:

- Oscillations without convergence: we have no guarantee that the loss will actually decrease
- The infamous *mode collapse*, when $\mathbf{G}$ models very well a small part of the data, concentrating on a few modes

Additionally, performance is hard to assess, often being a "beauty contest".

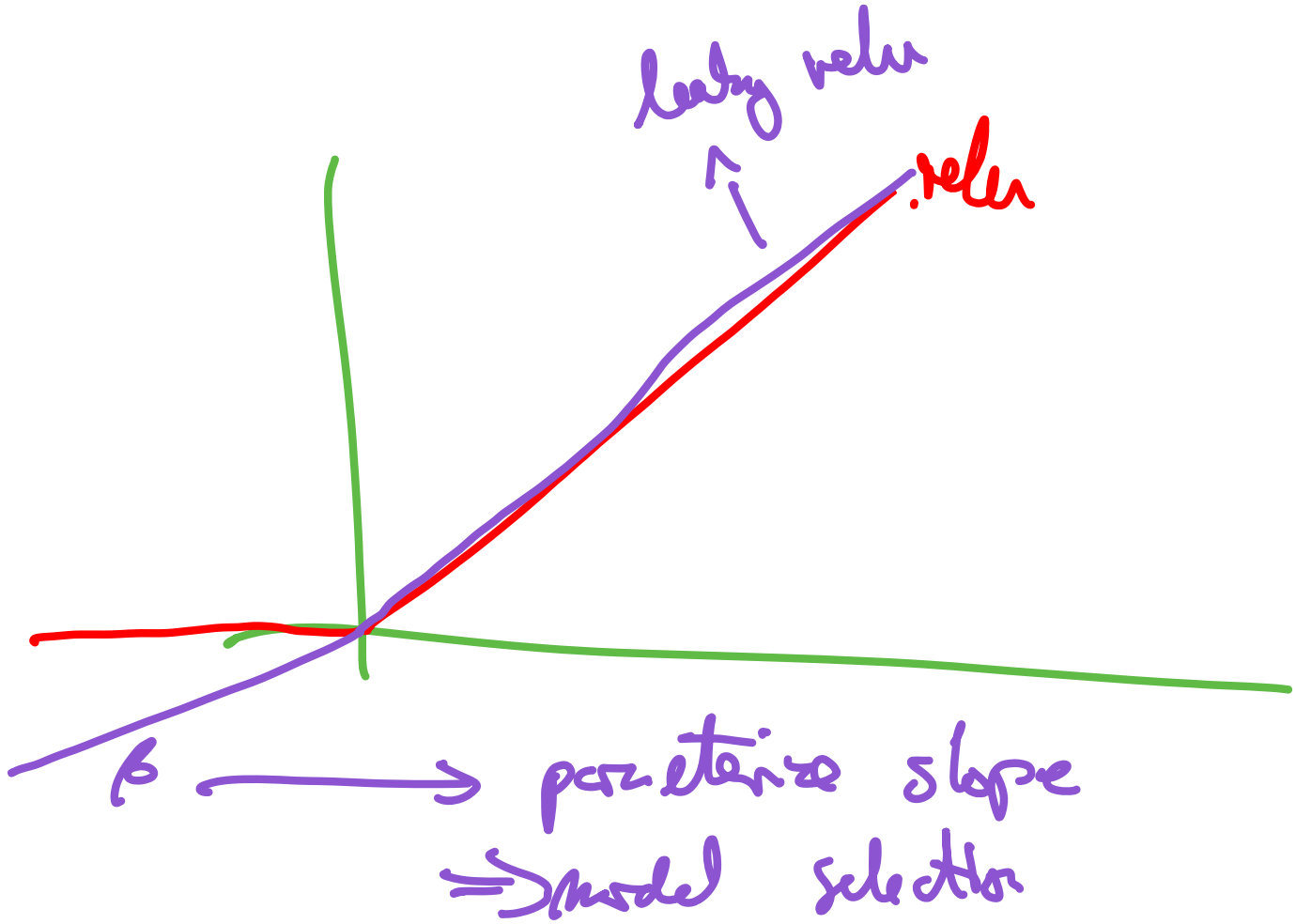


We also encountered difficulties attempting to scale GANs using CNN architectures commonly used in the supervised literature. However, after extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models.

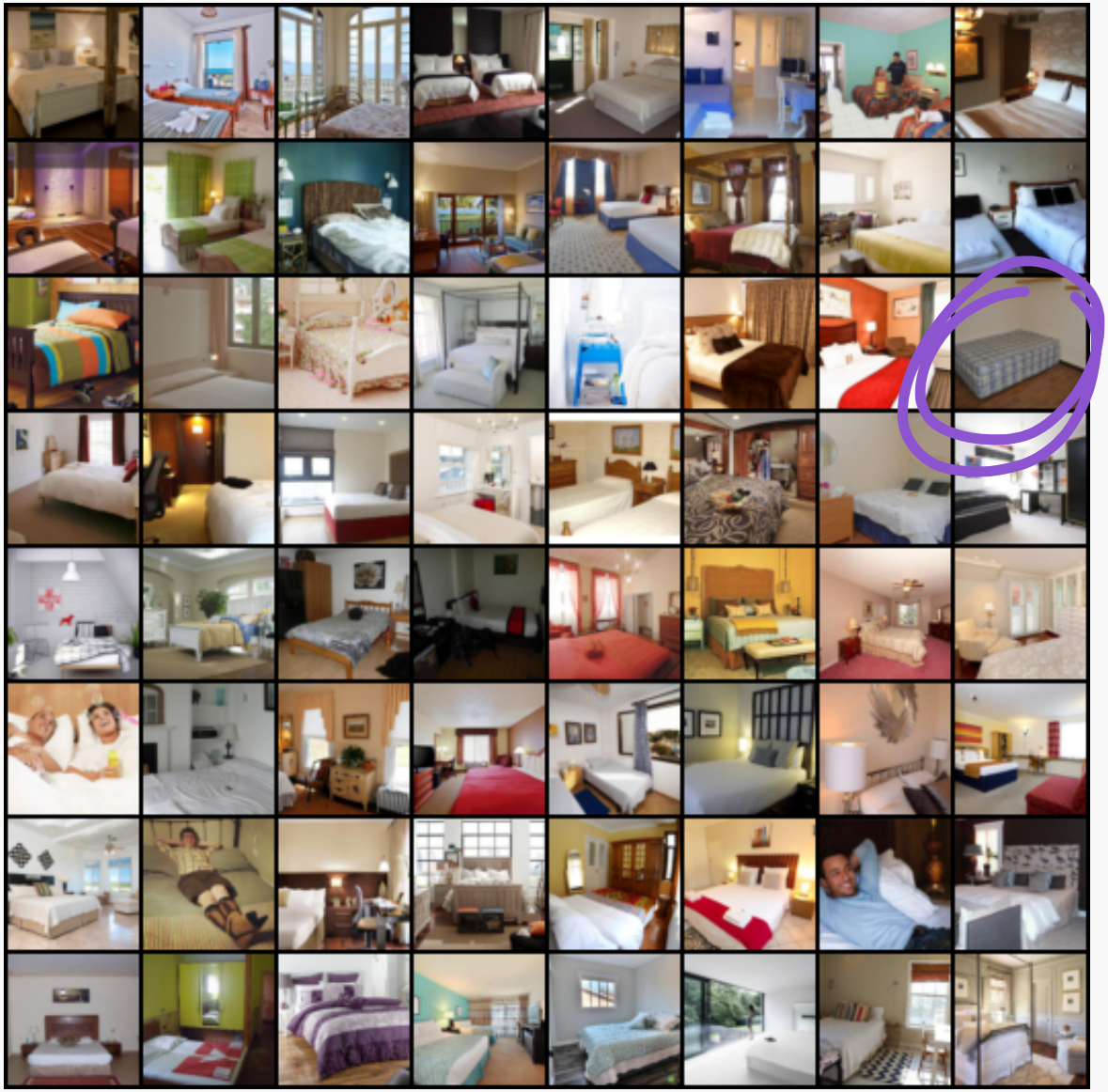
Radford et al. (2015)

Radford et al. (2015) adopted the following "rules":

- Replace pooling with
  - Strided convolutions in $\mathbf{D}$
  - Transposed convolutions in $\mathbf{G}$
- Use BatchNorm in both $\mathbf{D}$ and $\mathbf{G}$
- No fully connected layers
- Output of $\mathbf{G}$ uses $\tanh$, then ReLU elsewhere
- LeakyReLUs in $\mathbf{D}$

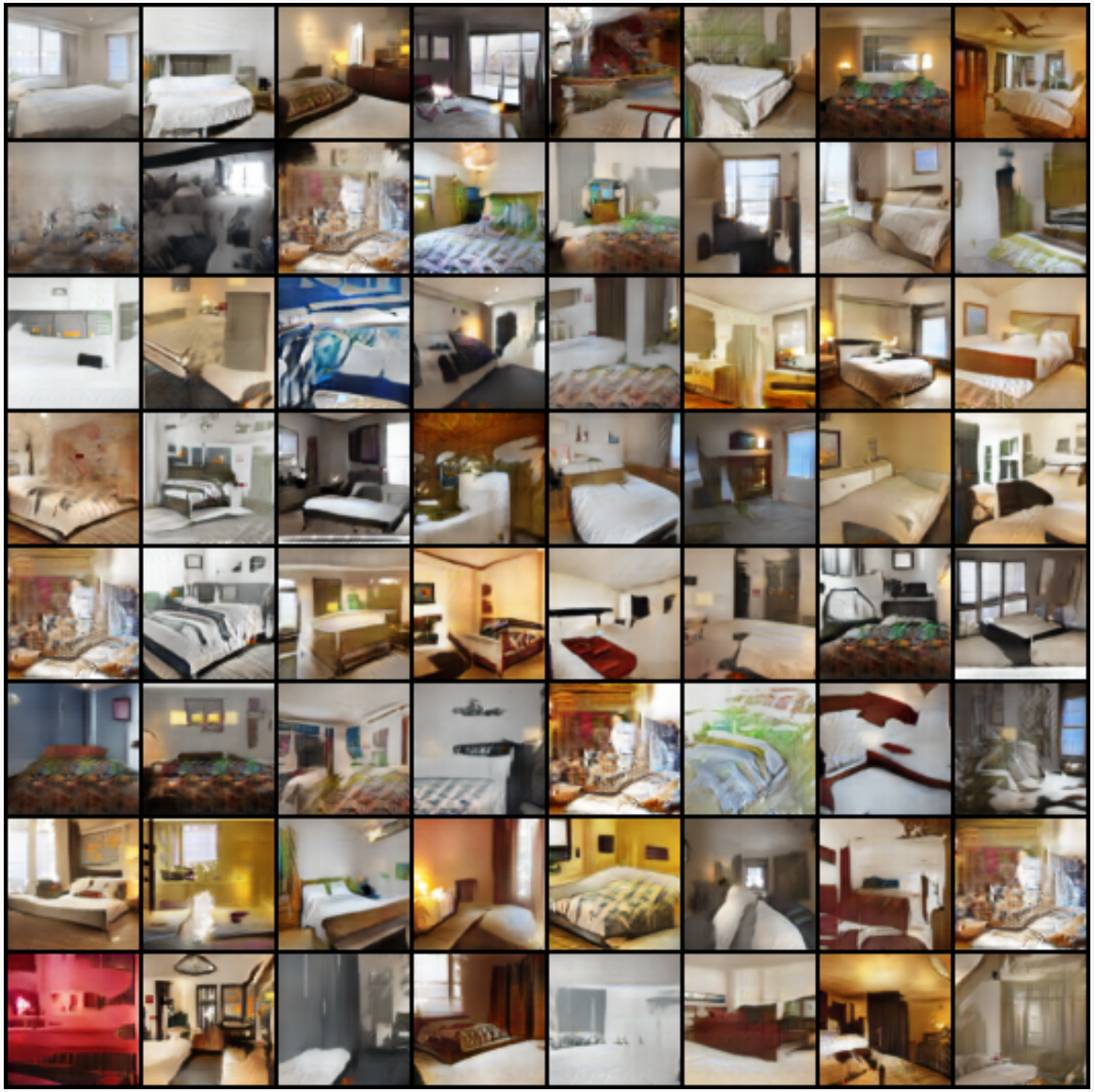

leaky relu
relu
α ⟶ parameter slope
⇒ model selection

“bedroom” class of the LSUN data set



Real samples

“bedroom” class of the LSUN data set



Fake samples (20 epochs)

Replace discriminator by statistical test

G → data set → D(data set)
(real data set)

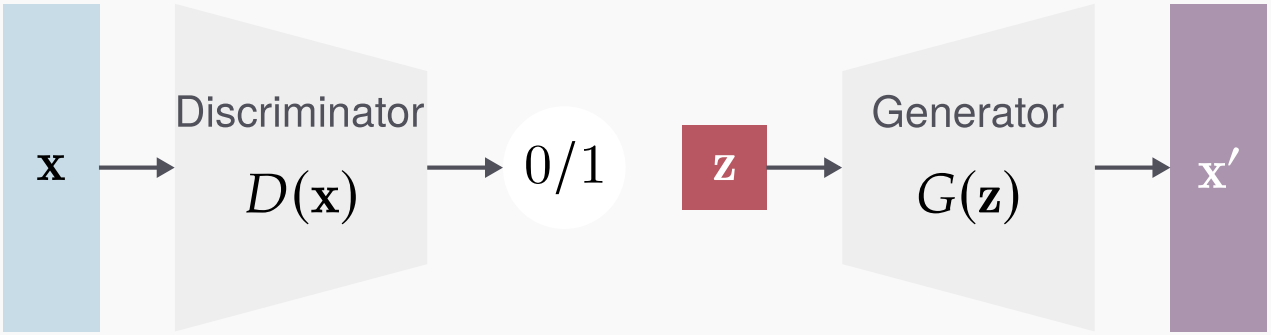
↓

significantly diff
or not

Deep Generative Modeling

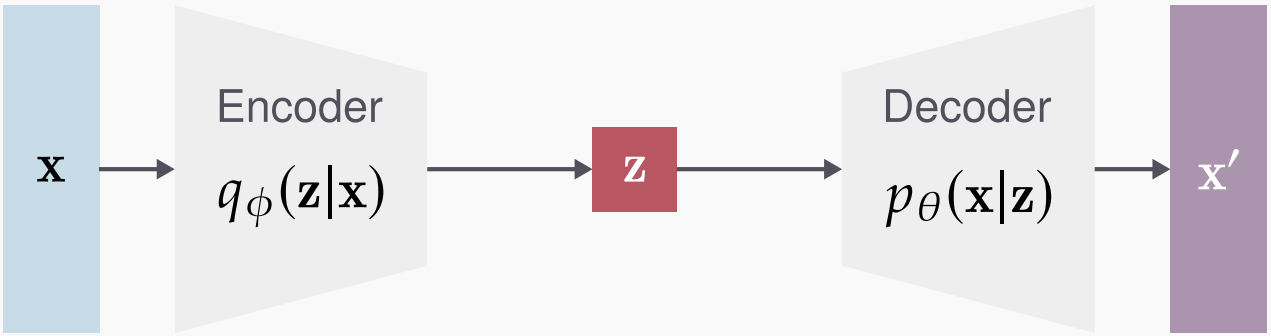
GAN

Minimax classification loss



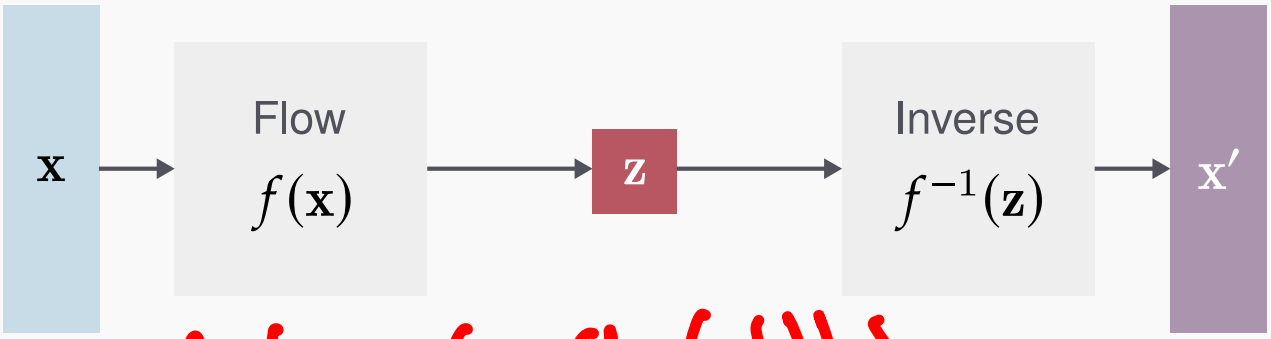
VAE

Maximize ELBO



Flow

Minimize NLL

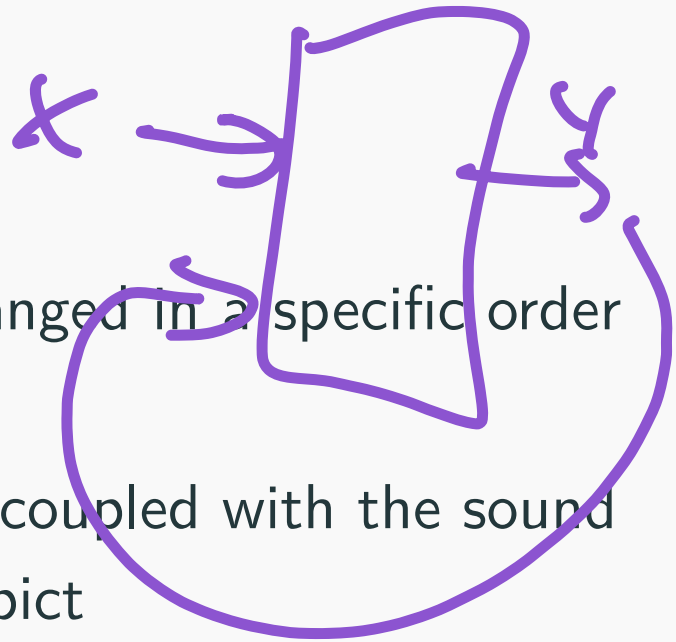


$f_k(f_{k-1}(\dots(f_1(x))\dots))$

Recurrent Neural Networks

TD-Gammon

Temporal or sequential data are different from simple vector data because the order in which they are arranged is part of the underlying phenomena they represent.



Examples include

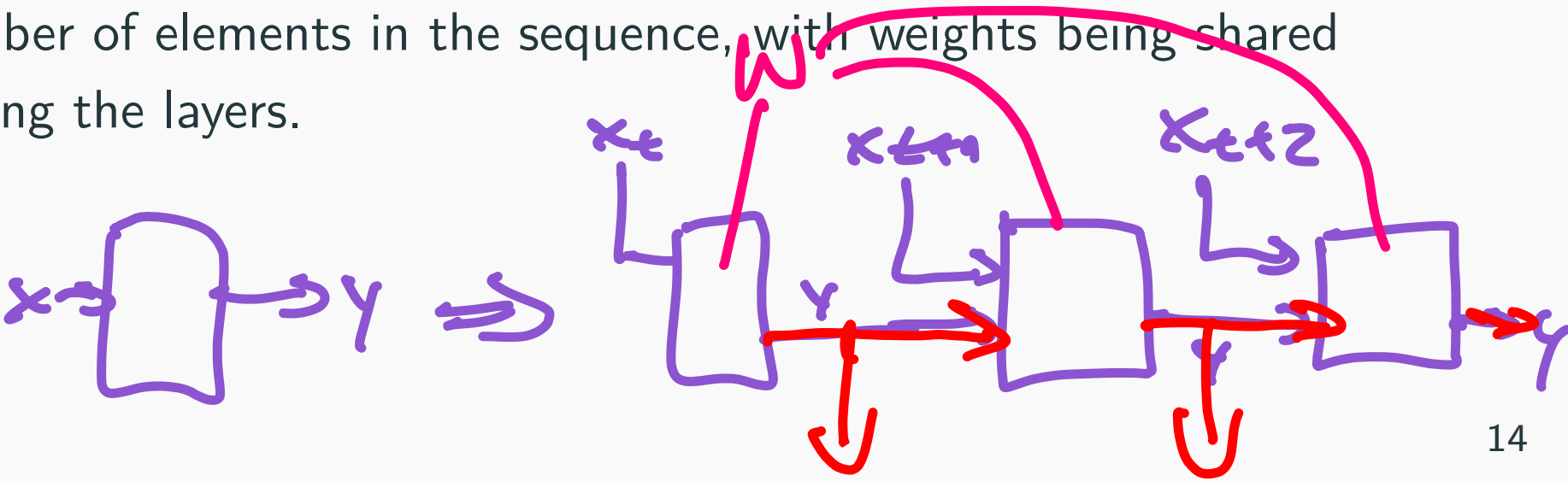
- text, where words and letters are arranged in a specific order to convey a certain meaning
- video, where the sequence of images coupled with the sound signal determine the content they depict

Recurrent neural networks are a class of neural networks designed specifically for handling sequential data.

Neural network architectures usually follow a specific “forward” direction: activations are always further from the inputs as it is transformed.

In recurrent architectures, however, this is not the case. The outputs of some neurons can be connected to the inputs of neurons that came before.

Such recurrent models can be seen as a MLP as deep as the number of elements in the sequence, with weights being shared among the layers.

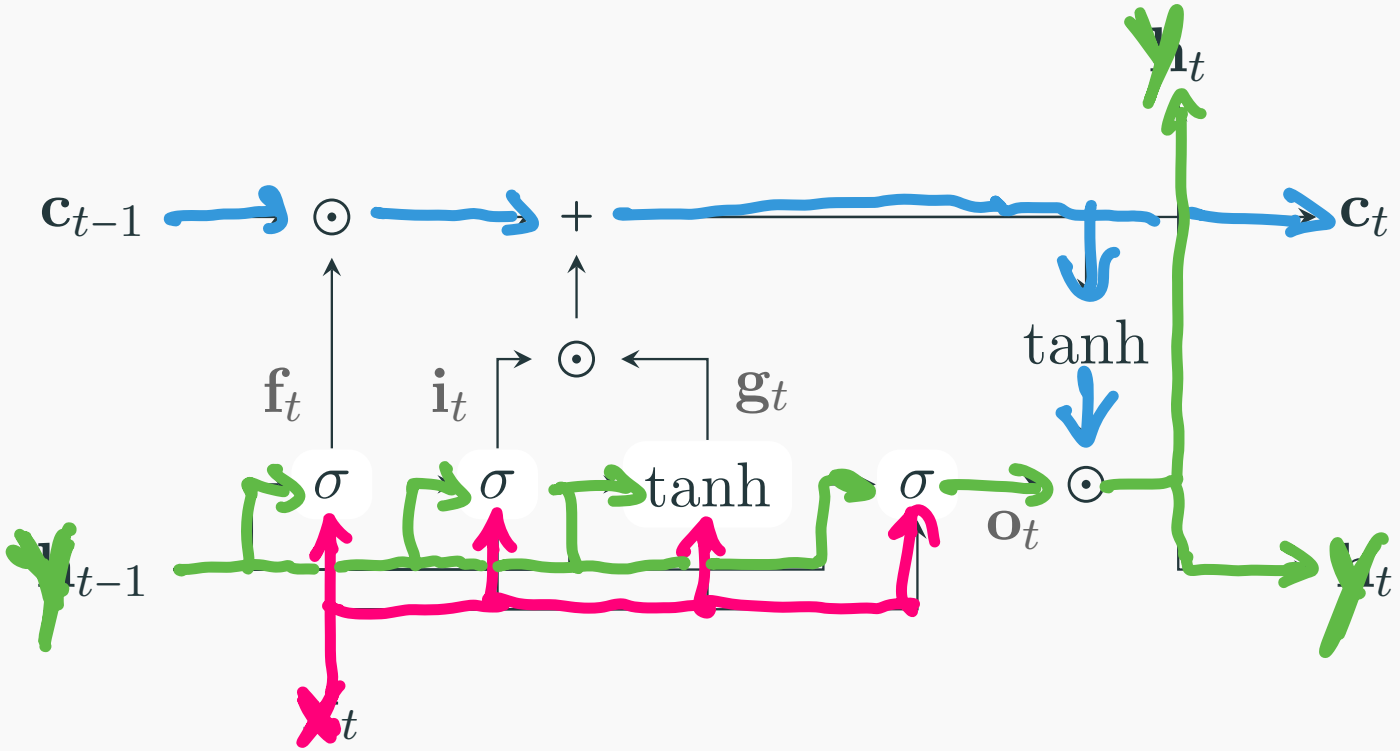


Similarly to regular deep architectures, connecting neurons in a recurrent fashion can easily cause vanishing gradients.

In fact, some of the first analyses on the vanishing gradients problem were conducted while studying recurrent architectures.

There are two main recurrent architectures designed to work with very long sequences: the *long short-term memory* (LSTM) and its simplification the *gated recurrent unit* (GRU).

Long Short-Term Memory (LSTM) cell



The value of the cell state $\mathbf{c}_{t-1}$ is increased by $\mathbf{i}_t \odot \mathbf{g}_t$. The forget gate $\mathbf{f}_t$ allows the LSTM to easily reset the value of the cell. The output is typically $\mathbf{h}_t$, and $\mathbf{c}_t$ is used internally.

The full updates are calculated as:

$$\mathbf{i}_t = \sigma\left(\mathbf{z}^{(ii)}(\mathbf{x}_t) + \mathbf{z}^{(hi)}(\mathbf{h}_{t-1})\right) \quad \text{(input gate)}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{z}^{(if)}(\mathbf{x}_t) + \mathbf{z}^{(hf)}(\mathbf{h}_{t-1})\right) \quad \text{(forget gate)}$$

$$\mathbf{g}_t = \tanh\left(\mathbf{z}^{(ig)}(\mathbf{x}_t) + \mathbf{z}^{(hg)}(\mathbf{h}_{t-1})\right) \quad \text{(full input gate)}$$

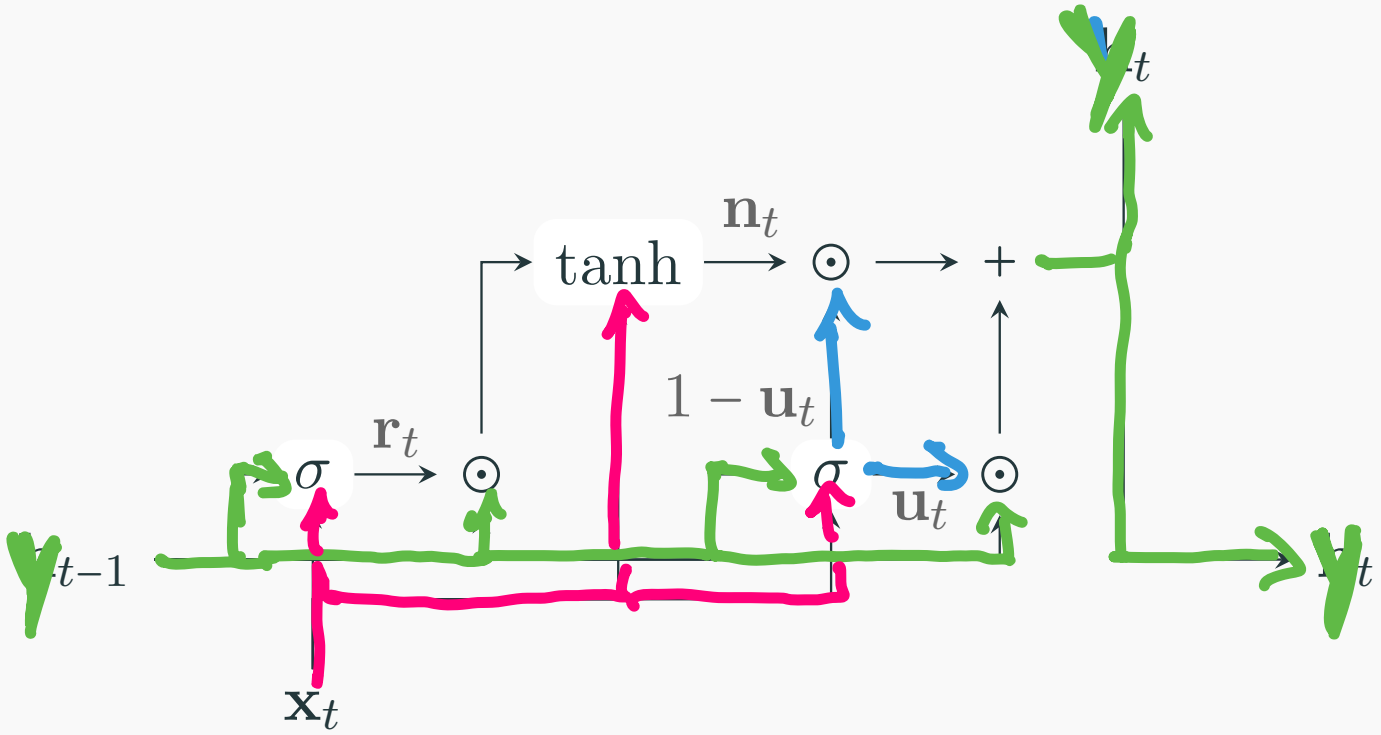
$$\mathbf{o}_t = \sigma\left(\mathbf{z}^{(io)}(\mathbf{x}_t) + \mathbf{z}^{(ho)}(\mathbf{h}_{t-1})\right) \quad \text{(output gate)}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \quad \text{(cell state)}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad \text{(output)}$$

where σ is a sigmoid function and $z^{(\cdot)}(\cdot)$ are linear transformations.

Gated Recurrent Unit (GRU) cell



$$\mathbf{r}_t = \sigma\left(\mathbf{z}_{\mathsf{ir}}(\mathbf{x}_t) + \mathbf{z}_{\mathsf{r}}(\mathbf{h}_{t-1})\right) \qquad \text{(reset gate)}$$

$$\mathbf{u}_t = \sigma\left(\mathbf{z}_{\mathsf{iz}}(\mathbf{x}_t) + \mathbf{z}_{\mathsf{hz}}(\mathbf{h}_{t-1})\right) \qquad \text{(update gate)}$$

$$\mathbf{n}_t = \tanh\left(\mathbf{z}_{\mathsf{in}}(\mathbf{x}_t) + \mathbf{r}_t \odot \mathbf{z}_{\mathsf{hn}}(\mathbf{h}_{t-1})\right)$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{n}_t + \mathbf{u}_t \odot \mathbf{h}_{t-1} \qquad \text{(output)}$$

References

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. CoRR, abs/1406.2661, 2014.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR, abs/1511.06434, 2015.