Rewriting a Deep Generative Model[1]

September 23, 2020

[1] "Rewriting a Deep Generative Model" by Bau et al., 2020

Overview

- *Rewriting a generator* means changing its weights in such a way that it now generates a different distribution

Figure: An example of rewriting: generator now generates churches with trees on top of their domes

- Authors provide a different perspective on NN layers as key-value memory banks
- They rewrite G by inserting a new key-value pair into one of its layers

An old way to do rewriting

- Imagine that we want to rewrite $G(z, \theta_0)$
- For this, we first generate several $x_i = G(z_i, \theta_0)$ for different z_i
- Then we update each $x_i \mapsto x_{*i}$ to describe our new rule
  - For example, photoshop a hat on a horse' head

(a) Copy

(b) Paste

An old way to do rewriting (continued)

Now, find the "rewrited" G parameters θ_1 by minimizing:

$$\theta_1 = \arg\min_{\theta} \mathcal{L}_{\text{smooth}}(\theta) + \lambda \mathcal{L}_{\text{constraint}}(\theta) \tag{1}$$

$\mathcal{L}_{\text{smooth}}$ helps us not to deviate from old images:

$$\mathcal{L}_{\text{smooth}}(\theta) \triangleq \mathbb{E}_z \left[\ell \left(G(z; \theta_0), G(z; \theta) \right) \right] \tag{2}$$

$\mathcal{L}_{\text{constraint}}$ helps us to enforce the rule:

$$\mathcal{L}_{\text{constraint}}(\theta) \triangleq \sum_i \ell \left(x_{*i}, G(z_i; \theta) \right) \tag{3}$$

Unfortunately, G overfits and does not generalize too well
But authors found that this strategy works much better if we optimize over a single layer instead of all the parameters

NN layers are memory banks

- We can see a linear layer $y = Wx$ as an associative memory (a 50-year old idea):

$$v_i \approx W k_i \tag{4}$$

- So, instead of input-output pairs we now have key-value pairs
- I.e., given an input k_i, layer W "recalls" output v_i.
- From this perspective, rewriting can be seen as overwriting key k_i with new value v_{*i} (or inserting a new one)
- For a convolutional layer, we just convert $n_{\text{in}} \times k \times k \times n_{\text{out}}$ tensor into $(n_{\text{in}} \cdot k \cdot k) \times (n_{\text{out}})$ matrix

Rewriting by memory update in a linear case

- We now can rewrite G by overwriting one of its keys k_* in layer W^ℓ
- For this, we solve the optimization problem:

$$W_1 = \arg\min_W \|V - WK\|^2 \qquad \text{s.t. } v_* = W_1 k_* \tag{5}$$

- It has an analytical solution, but authors develop a different perspective to be able to generalize to non-linear transforms

Rewriting by memory update in a *non*-linear case

- NN layers are usually non-linear transforms: $y = f(x, W)$
- This still can be seen as an associative memory $v_i = f(k_i, W)$
- In this case, there is no analytical solution to update the memory
- But for a linear case, we can shown that

$$W_1 = W_0 + \Lambda \left(C^{-1} k_*\right)^\top \tag{6}$$

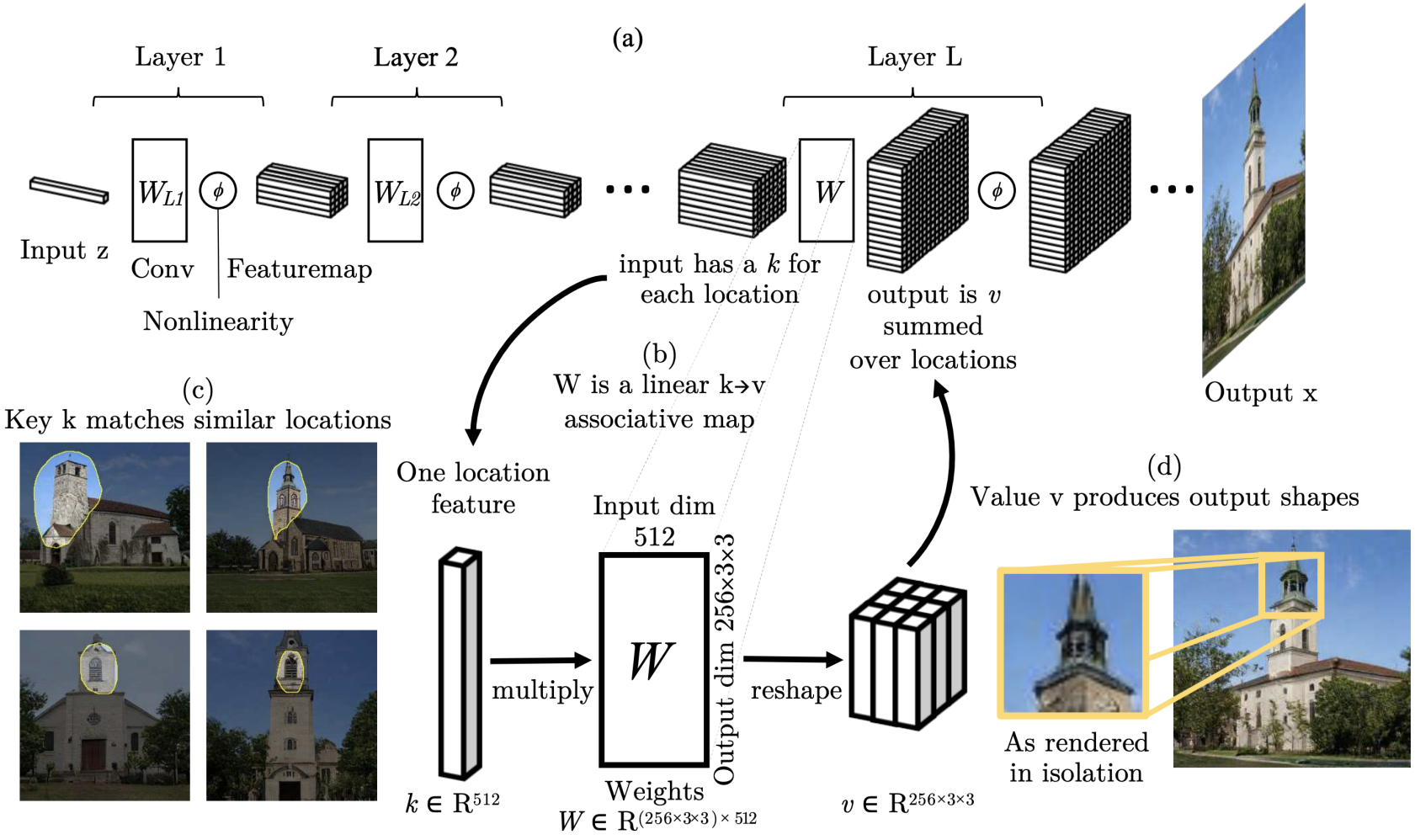
where

$$\Lambda_1 = \arg\min_{\Lambda \in \mathbb{R}^M} \left\| v_* - [W_0 + \Lambda (C^{-1} k_*)^\top] k_* \right\| \tag{7}$$

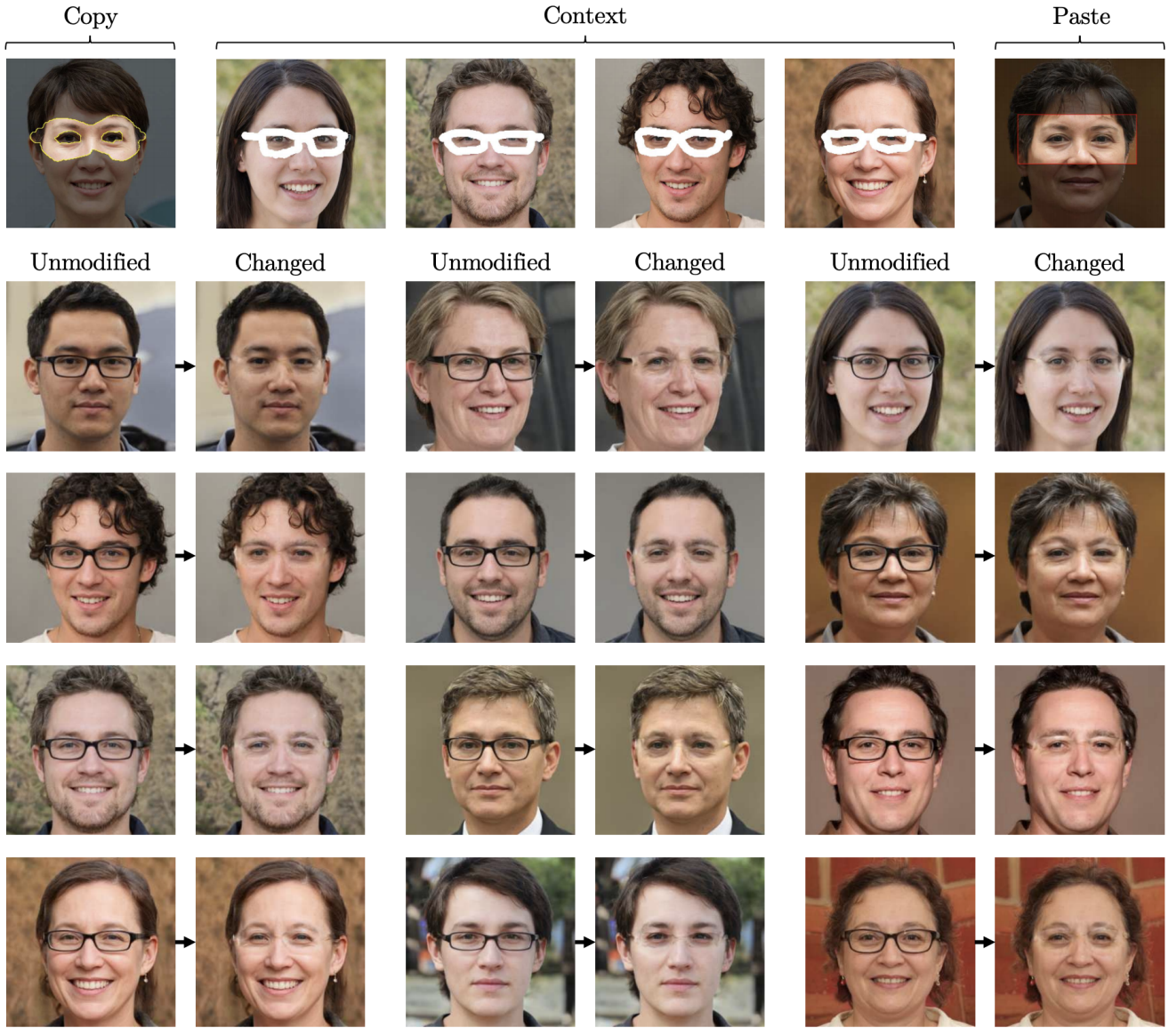
- Authors generalize this procedure to a non-linear case by finding Λ_1 via:

$$\Lambda_1 = \arg\min_{\Lambda \in \mathbb{R}^M} \left\| v_* - f\left(k_*; W_0 + \Lambda (C^{-1} k_*)^\top\right) \right\| \tag{8}$$

Rewriting procedure visualization



Qualitative results



Final thoughts

- Rewriting shows a powerful property of modern GANs
- But generator does not know how to blend a new value nicely with the novel context
  - That's why for many samples results look clumsily "copy-pasted"
  - How can we alleviate this?
  - (Results still look impressive)
- Associative memory perspective is very interesting since this is how human memory works