Latent Generative Memory

February 6, 2020

Latent Generative Memory (LGM)

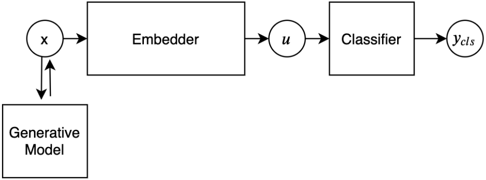

Figure 1: Generative Memory (i.e. a traditional one)

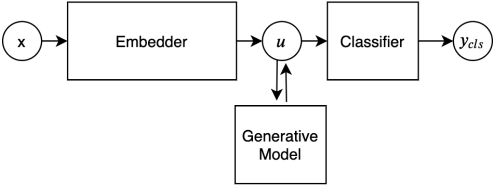

Figure 2: Latent Generative Memory

Remark: one can use any generative model (GAN/VAE/Flows/etc) for their generative memory.

LGM pros and cons

Pros:

- Much simpler computationally since training anything in a latent space is easier
- Should provide better scores than MeRGAN since huge GANs are difficult to train in an image space, especially continually

Cons:

- It is not obvious how to make Embedder not forget previous data

Making Embedder less forgetful. Strategy #1.

Strategy #1: Apply EWC (or MAS/AGEM/etc.) to Embedder.

- Pros: should be easier to implement
- Cons:
  - It is less novel.
  - Can work bad in scenarios where EWC (or MAS/AGEM/etc.) work bad. If this is true then things are bad.

Making Embedder less forgetful. Strategy #2.

Strategy #2: Apply autoencoding distillation loss:

- Train a decoder $D : u \to x$ to decode representations again into images
- Apply the following loss:

$$\mathcal{L}^{\mathsf{AE}} = \mathop{\mathbb{E}}_{p_{t-1}^{\mathsf{LGM}}} \left[\| u - E_t(D_{t-1}(u)) \|_2^2 \right]$$

- Intuition: $\mathcal{L}^{\mathsf{AE}}$ directly forces E_t not to change on previous data.
- Pros:
  - More novel
  - Should work well if decoder D is perfect
- Cons: we are not sure how good our decoder will be

LGM-VAE overview

Scheme is the same

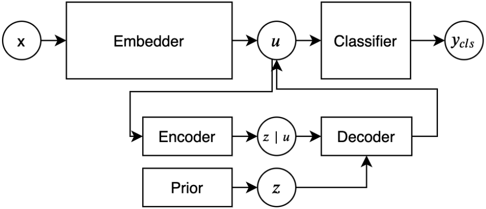

Figure 3: VAE-based LGM

VAE training specifics:

- We perform distillation for an encoder and a decoder independently:
  - For decoder we just sample from prior and optimize $\|D_t(z) - D_{t-1}(z)\|^2$
  - For encoder we sample u from the previous GM model and optimize $\|E_t(u) - D_{t-1}(u)\|^2$
- It should be beneficial to train a prior model to memorize data better

LGM-VAE pros and cons

Pros:

- training is more stable
- gives better scores in preliminary experiments
- gives better scores for “Three scenarios” paper

Cons:

- Distillation is a bit of ill-posed
- GANs should be better at memorizing data?
- Not obvious how to force hallucinated samples look real

Continual Learning by Memorization

Imagine we have a magic neural network M that:

- Takes a dataset X as an input and memorizes it
- Outputs the stored dataset X by demand
- Has "reasonable" number of parameters (i.e. less than HAT, for example).

We can "solve" CL by using M:

- For task 1 store dataset X_1 in M.
- For task t extract all the previous datasets $X_1, ..., X_{t-1}$, concat them and put into M
- At each timestep t we can extract all the stored data $X_1, ..., X_t$ and train a *joint* classifier (which is an upper bound!).

Will it be a fair approach?

Final thoughts

- ICCV19 model is not an LGM model since they do not use embedder
- LGM model, if done properly, seems to be an individual contribution that can be sold individually (i.e. applying a creativity loss can be a separate project after that)
- If one is going to build LGM through autoencoding loss than the project is likely to consist on 3 sequential projects:
  1. Take any autoencoder, train a generative model in its latent space. How good are the samples?
  2. Put this setup onto continual learning rails, i.e. it allows us to train LGM with autoencoding loss
  3. Apply creativity losses
- "CL through memorization" idea can also have potential for two projects:
  1. How to build a "Deep Memorizing Model"?
  2. Show that DMMs solve CL perfectly, implying that there is something wrong with our evaluation of CL.