CSS 699: Representation Learning - Fall 2019 Due: Sunday Nov 3, 2019

Rendering Images with Latent Semi-Supervision

Introduction

The purpose of the assignment is to provide you with experience training Variational Auto-Encoders (VAEs), for the purpose of discovering (decorrelated) latent factors in the data. In addition, we want to study the cases where it helps to supervise the latent representation with latent factor information (if/when such information is present). Finally, the assignment also proposes some related Beyond Assignments.

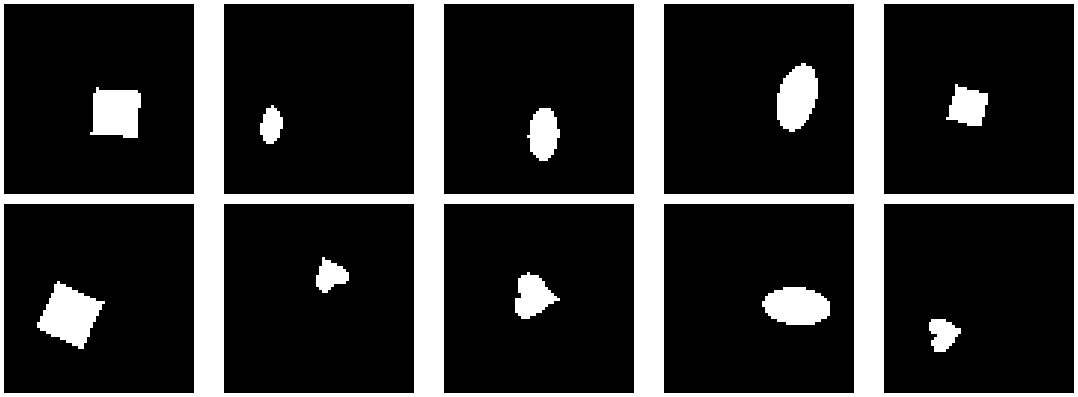
Latent Factors

Our dataset $\mathcal{D}$ consists of N binary images $x_1, \ldots, x_N \in \{0,1\}^{64\times 64}$. The data has been generated by uniformly sampling across some attributes:

- Color: white
- Shape: square, ellipse, heart
- Scale: 6 values linearly spaced in [0.5, 1]
- Orientation: 40 values in [0, 2 pi]
- Position X: 32 values in [0, 1]
- Position Y: 32 values in [0, 1]

Images

The images below have been randomly sampled from the dataset.



Rendering Images with Latent Semi-Supervision

Problem Setup

Unfortunately, for most samples $x \in \mathcal{D}$, their latent factors are hidden from us. Luckily, however, for a small fraction of images, we are given labels for their latent factors. We denote these latent factors with y, which is a set of 1-hot vectors. Specifically, y_j is a 1-hot vector for factor j with entry $y_{jc} = 1$ iff the attribute has value c. For example, $j = 1$ corresponds to "shape" and $c = 0$ corresponds to square. We denote images and their labeled factors as $(x, y) \in \mathcal{D}_L$ with $|\mathcal{D}_L| \ll |\mathcal{D}|$. The goal of the assignment is to create two sub-networks: an *attribute classifier* [i.e. an encoder] an *image renderer* [i.e. a decoder]. The output of one is the input to the other.

In the first task, you will train a Variational Auto Encoder (VAE) only on the images $x \in \mathcal{D}$. The VAE should discover the latent factors in an unsupervised way, though the correspondence between latent factors y (that generated the data) and the latent variables z (that the encoder learns to output) will not be pre-specified by the learning algorithm and such correspondence could be recovered by inspection after learning the VAE.

In the second task, you will use $(x, y) \in \mathcal{D}_L$ to supervise latent variables, in addition to using $x \in \mathcal{D}$, for training the VAE. This allows us to slice the z space with each segment of z capturing information for a certain latent factor y_j for $j \in \{1, 2, 3, 4, 5\}$

Finally, subsequent tasks are optional and can award you bonus credits, or can be folded as a *Beyond Assignment* if you choose to team-up with others. Briefly, the tasks are:

1. Standard β-VAE. Learn a model: image (x) $\rightarrow$ latent variables (z) $\rightarrow$ reconstructed image ($\widetilde{x}$), *without* latent supervision. That is, you only observe x. You will interpolate points by taking a convex combination of their latent variables.

2. Control the correspondence between latent variables and latent factors. Learn a model: image (x) $\rightarrow$ latent variables (z) $\rightarrow$ reconstructed image ($\widetilde{x}$), *with* latent supervision. In particular, we want to partition latent vector z into 5 partitions (5 subvectors, let us refer to them as $p_1, \ldots, p_5$), such that each partition can inform us of one latent factor.

3. [**Bonus choice 1; Beyond Assignment**] Modular Neural Network. You can hand-craft a neural network given your knowledge of the data-generating process. Injecting knowledge into the model architecture is referred to as *inductive bias* in deep learning. Since we know there are a discrete number of (simple) shapes, each can appear in a certain location and orientation, we propose to design a network consisting the following differentiable[1] modules. *Memory module*, for looking-up and storing object shapes[2]. Rotation and scaling network (for example, a single layer of the Spatial Transformer Network), used by the decoder. The encoder also consists of modules including *object localizer*, and an *object classifier*

[1] i.e. their gradient w.r.t. inputs and parameters is defined and continuous almost everywhere

[2] Exact shapes must not be memorized beforehand, but rather, memorized by the training process. Code should generalize to different shapes.

4. [**Bonus choice 2; Beyond Assignment**] Spatially Differentiable Spatial Transformer Networks by Annealed Gaussian Smoothing.

Dataset

We will be using the dSprites, which can be downloaded from github: dsprites-dataset. After downloading the dataset, you are expected to download the IDs file (which splits the dataset into train (representing $\mathcal{D}$, with $|\mathcal{D}| = 245,760$); supervised train (representing $\mathcal{D}_L$, with $|\mathcal{D}_L| = 16,384$); and test. Please follow these steps:

- Download the dataset from github: https://github.com/deepmind/dsprites-dataset
- Download the data split IDs: http://sami.haija.org/cs699/hw3/ids.npz
- You can access the images and latent variables following code snippet:

```
data = np.load('path/to/dsprites.npy')  # download from github
print(data.files)  #Prints: ['imgs', 'latents_classes', 'latents_values', ...]
ids = np.load('ids.npz')  # downloaded from course website.
all_imgs = data['imgs']

train_images = all_imgs[ids['train']]  # please only access imgs of 'train'

# You can access all attributes for supervised_train!
sup_train_images = all_imgs[ids['supervised_train']]
sup_train_factors = data['latents_values'][ids['supervised_train']]
sup_train_classes = data['latents_classes'][ids['supervised_train']]

test_reconstruct = all_imgs[ids['test_reconstruct']]  # for q1_a.pdf
test_interpolate = all_imgs[ids['test_interpolate']]  # for q1_b.pdf
```

[*60 points*] Task 1: β Variational Auto-Encoders

In this task, you should implement β-VAE accessing only '`train_images`' (according to the dataset snippet code). The construction of the β-VAE is in Prof. Greg's lecture slides (during week of Sept/30). Sami has also described the computation graph for training VAEs (remember: the *sampling* operator should be in the computation graph but not[3] on the critical path from trainable parameters to the loss node).

You are expected to "tune" your model, searching for a good architecture, optimization algorithm, learning rate schedule, and a good value of β. Online resources can help you complete the starter code.

[3] frameworks like tf probablity or keras offer sampling operators implementing the *reparametrization trick*

Rendering Images with Latent Semi-Supervision

The starter code is on http://sami.haija.org/cs699/hw3/starter.py. You are expected to complete the code and qualitatively show that your model is well-trained in two ways: (1) visualize test images and their reconstructions, and (2) by measuring how well your decoder can reconstruct images from interpolated points in the encoder output (latent variables) space z. We now list your deliverables.

(*0 pts*) Implement and train a β-VAE on training samples in file **starter.py**. *Major work. Grade points are folded onto the next 2 parts, as an implementation that produces no visualizations is worth zero credit.*

(30 pts) Create and upload to vocareum file **q1_a.pdf**, showing all images at indices '`test_reconstruct` and their reconstructions.

(30 pts) Create and upload to vocareum file **q1_b.pdf**, showing 5 pairs of images. For each pair $(x^{(a)}, x^{(b)})$, pass both $x^{(a)}$ and $x^{(b)}$ through the encoder to sample $z^{(a)}$ and $z^{(b)}$. Then, produce the image interpolations:

$$\text{interpolations} \;=\; \bigcup_{\alpha \in [0..1]_{0.2}} \left\{ \alpha z^{(a)} + (1-\alpha) z^{(b)} \right\},$$

where $[0..1]_{0.2}$ is the set of real numbers between 0 and 1 with 0.2 increments: $[0..1]_{0.2} = \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. For each pair, plot rows of 8 images: the left-most and the right-post are the original $x^{(a)}$ and $x^{(b)}$. Plot the 6 interpolations in the middle. Clearly indicate the originals and the interpolation values (e.g. indicate α).

Getting this part working properly is *crucial* for this assignment. You may search online for VAE model architectures, especially designed for dSprites. If you follow code on the internet, please add a file header comment linking to the source. You must understand the code very well (do not copy it blindly), as you need to modify it for the subsequent task.

[*80 points*] Task 2: Supervise Latent Representations

In this task, you are expected to designate variables in your latent space to be indicative of latent factors. Let d-dimensional $z \in \mathbb{R}^d$. Your goal is to partition the d dimensions among the latent 5 latent factors in y (we ignore color). Let us arbitrarily divide the two dimensions such that (w.l.o.g.): $z = [p_1\ p_2\ p_3\ p_4\ p_5]$ (i.e. z is a concatenation of p_i's). Your job is to maximize the mutual information terms, $I(y_1 : p_1), \ldots, I(y_5 : p_5)$.

Task 2.i (*45 points*): $\max I(y_i : p_i)$

This task will be evaluated on how-well we can predict y_i from p_i for $i \in \{1, 2, 3, 4, 5\}$. While learning your model, you are should[4] make p_i be a good predictor for y_i.

[4] How to do that, is up to you!

Rendering Images with Latent Semi-Supervision

This (sub-)task will be graded as follows:

$$\text{score} = 50 \times \frac{2}{N} \sum_{i=1}^{\frac{N}{2}} y^{(i)}_{jc^{(i)*}_j}$$

where $y^{(i)}_j$ is a one-hot label vector for example (i) and N is the number of test samples. The maximum achievable score for this part is 50. We denote binary value of class c as $y_{jc} \in \{0, 1\}$ with value set to 1 iff class c is the value for the latent factor j (e.g. $c \in \{\text{heart}, \text{square}, \text{oval}\}$, when factor $j = \text{shape}$), and the highest prediction:

$$c^{(i)*}_j = \arg\max_k \left[h^{(j)}(p^{(i)}_j) \right]_k ,$$

is the largest entry of vector $h^{(j)}(p^{(i)}_j)$, output by the model $h^{(j)}$ that the auto-grader trains[5] by optimizing the log-likelihood:

$$\max_{h^{(j)}} \log P(y_j \mid p_j; h) = \max_{h^{(j)}} \sum_{i=\frac{N}{2}+1}^{N} {y^{(i)}_j}^\top \log h^{(j)}(p_j) + (1 - y^{(i)}_j)^\top \log(1 - h^{(j)}(p_j)),$$

which is a sum of scalars, and the first (of the two) dot-product is between the ground-truth one-hot $y^{(i)}_j$ and the log model scores, assuming $0 \le h^{(j)}(p_j) \le 1$ (entry-wise).
Your deliverable in this task is to complete the class[6] `SSLatentVAE`. You must implement:

- The function `load`. It should accept no arguments and should restore the model parameters. Auto-grader will call this exactly once on each instance of `SSLatentVAE`.
- The function `sample_z_given_x`. It should accept batch of examples $\in \mathbb{R}^{b \times 64 \times 64}$, where b is the batch size, and return $z \in \mathbb{R}^{b \times d}$ matrix, with one d-dimensional vector per sample. If your function returns another shape e.g. $z \in \mathbb{R}^{b \times d_1 \times d_2 \times \cdots}$, then the autograder will flatten the dimensions except for the first.
- The function `get_partition_indices`. It should partition z (using index ranges) into 5 partitions, one partition for predicting each latent factor. Please follow the function comments.

The auto-grader is based on the following code snippet, which you might find helpful.

[5] We train sklearn's LogisticRegression classifier, specifically because its a one-layer neural network and therefore its the log-likehood is convex in the parameters. Its minimizer can be obtained up-to arbitrary precision ϵ, in computation time $\mathcal{O}(\log_2(\#\text{ bits required for representing } \epsilon))$. Nonetheless, you are allowed to substitute another sklearn linear model e.g. SVM, by modifying in starter code the variable `TASK_2A_AUTOGRADER_CLASSIFIER`.

[6] If you are using the first code version, this class is only a rename of `AdversarialLatentVAE`, to reflect the slight change of the task which we decided on, after posting the starter code.

```
model = SSLatentVAE()
model.load()
indices = model.get_partition_indices()
# ! check that len(indices) == 5 and non-overlapping.
# ... if check passes, continue:
X = ...  # loads portion of test
Y = ...  # loads portion of test
Z = model.sample_z_given_x(X)
Z = Z.reshape([len(Z), -1])  # Flatten

metric = [fit_then_eval(Z[:, idx[i][0]:idx[i][1]], Y[:, i]) for i in range(5)]
score = np.mean(metric) * 50.0
```

Task 2.ii (*35 points*): Rendering using Mixed Latent Vectors

In this task, you will produce visualizations, showing how well your VAE, when trained with semi-supervision that you crafted, can mix latent factors from different samples. Specifically, if you choose a pair of samples: $x^{(a)}$ and $x^{(b)}$. You can pass both through the encoder to sample $z^{(a)}$ and $z^{(b)}$. Since each z is partitioned i.e. $z^{(a)} = [p_1^{(a)}, p_2^{(a)}, p_3^{(a)}, p_4^{(a)}, p_5^{(a)}]$, and similarly for b, one should be able to mix the latent vectors e.g. let $z^{(\text{mixed})} = [p_1^{(b)}, p_2^{(b)}, p_3^{(a)}, p_4^{(a)}, p_5^{(b)}]$. Now, if we run $z^{(\text{mixed})}$ through the decoder, how does the rendered image looks like? Does it indeed produce an image which has the 3rd and 4th attribute from image (a) but the other attributes form image (b)?

Create file **q2_b.pdf** on your vocareum directory. Pick 5 pairs of images of your choice from dSprites. Favor diversity in your choices. Perhaps some pair could differ in 1 attribute, another in 2 attributes, etc. Show both source images on the pdf, and plot the reconstructed image, clearly indicating which latent variables you have taken from which.

[*120 points*] Bonus Task: Module Networks

Given the simple data generation process, we can tailor-design a module network for exactly solving the task: i.e. directly learn the latent factors (scale, object shape, location) and have the decoder render them. We propose to use the following modules:

- **Object Localizer** using Separable Convolutions [not trainable; but can be made trainable beyond this assignment]. Processes an image and outputs a bounding box (i.e. position and scale) of where the object lies in the image.
- **Extractor** by slicing the tensor. The slicing operation is equivalent to a (differentiable) attention module [not trainable]. Output should be a fixed-size image.

- **Object & Orientation Classifier**. Given a rescaled output, emits object ID and prediction on orientation. The object IDs need not to match the original IDs. In fact, you might create more IDs in this space than only the 3 we are given. Each object ID is mapped to an image by the memory network. [trainable]

- **Memory** It lives at the boundary of encoder and decoder. Specifically, The encoder outputs the "memory address" but the memory itself lives in the decoder which does a soft lookup (Graves, Wayne & Danihelka; 2014) the decoder retrieves the shape from memory shared between encoder and decoder); and finally a Spatial Transformer module, enabling the projection of the shape (from memory) onto the proper location Looks-up object ID from memory and returns it, giving its pixel values [trainable]

- **Renderer** Given scale, orientation, and pixel values from memory, this should produce an image of the desired object at the given location, orientation, and scale.

In fact, these are only suggestions. You can come up with your own modules (or own architecture), so long as you use some kind of inductive bias to design solve this task (i.e. use your knowledge about the data generation process).

If you do this part, you have two options: either make it part of this assignment (in which case, you'd get points for the assignment) or turn it into a "Beyond Assignment", in which case, you can work with collaborators on it. If you decide to capture the grades as part of this assignment, then please create a PDF outlining your findings (anything that proves you've done the work e.g. visualizations and/or metrics) and upload the file to vocareum, with name **task3.pdf**. If you want to make it a Beyond Assignment, then please **email it** to sami on haija@isi.edu with subject prefix "[Beyond Assignment Submission]" and CC your teammates.

[*120 points*] Bonus Task: Better differentiability through Spatial Transformations via Gaussian Smoothing

There is no write-up for this (open-ended) task, especially that it is not related to this specific assignment. If you'd like to attempt it, please discuss it with Sami through office hours or by appointment. For submission, follow the instructions in the previous paragraph.