Advanced Machine Learning (GR5242)
Fall 2019

Final Projects

Due: Monday 09 December at 4pm (for both sections of the class)

Project Submission: Complete one of the following projects. Please submit your completed project by publishing a colab notebook that cleanly displays your code, results and plots to pdf or html. You should also include a pdf file containing typeset or neatly scanned description of your project goals and results. You should use the NeurIPS tex template and style, found here `https://neurips.cc/Conferences/2019/PaperInformation/StyleFiles` (note: ensure the author names are not anonymized!).

Project 1 (GAN)

Given some data, understanding its distribution and generating samples are not trivial, especially for complex data such as images or audio. Generative Adversarial Networks (GAN) provide a method to do so by learning a generative process. GAN consists of two networks, a generator G and a discriminator D.

G is responsible for understanding the distribution of data and generating a sample. Let $p_g(x)$ be the generator's distribution over data x and $p_z(z)$ be a noise distribution where samples can be easily obtained (for example, Gaussian distribution). Then, $G(z)$ is a differentiable function with parameters, θ_G that maps z to x. Thus, $G(z) \stackrel{d}{=} x$, where $z \sim p_z(z)$ and $x \sim p_g(x)$.

D is responsible for discriminating real data and synthetic data generated from G. Thus, $D(x)$ is another differentiable function with parameters, θ_D that outputs a probability that x is from real data rather than p_g.

Then, the goal is straightforward. D is trained such that $D(x)$ to be 1 if x is real data and 0 if $x \sim p_g(x)$. G is trained to confuse D. In summary, the objective function, $V(G, D)$ will be

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

G is trained to minimize $V(G, D)$ and D is trained to maximize it.

- From the paper. `https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`, understand core ideas of GAN. Make sure to understand Figure 1 and Algorithm 1 of the paper. Also, why we want G to minimize and D to maximize $V(G, D)$.
- Implement your own GAN with CNN layers on MNIST data. Please describe the architectures of your generator and discriminator and also any hyper-parameters chosen. Post a plot of training process from tensorboard to make sure that the networks are trained as expected. To help guide you, an example of GAN on MNIST can be found in `https://www.tensorflow.org/tutorials/generative/dcgan`, but importantly, you must develop your own code and your own neural network models.
- Visualize samples from your model. How do they look compared to real data? How is it compared to Figure 2a in the paper?
- Implement your own GAN with SVHN data. Explore different architecture of neural networks and hyperparameters. Compare samples from your model to real data. How is the quality compared to your GAN on MNIST? If the training does not go well, what failure modes do you see?
- **(Optional)** There are several improved versions of GAN such as Wasserstein GAN (WGAN). Train your own WGAN `https://arxiv.org/abs/1701.07875` on MNIST and SVHN instead of the plain GAN.

Project 2 (Deep reinforcement learning with TF-Agents)

In this project, you will implement a reinforcement agent that is able to excel one of the OpenAI Gym games. In class, we saw three types of agents — random, strict policy, and deep Q-learning agent. With the availability of the new Tensorflow library tf-agents, now you can explore and implement other types of reinforcement agents with less efforts on coding, but more on improving the policy. Follow the steps below for completing this project:

- Choose one environment from OpenAI Gym (`https://gym.openai.com/envs/#classic_control`). You may want to start with the beginner "Classic control" section if you are relatively new to this area. Nevertheless, feel free to choose other more advanced environments to challenge yourself.
- Choose one agent from tf.agents library (`https://github.com/tensorflow/agents`).
- Study and explain the policy learning method for the particular agent of your choice.
- Use the training procedure provided in the tutorial (`https://github.com/lydiahsu/AML_Fall_2019/blob/master/TF_agents_intro.ipynb`) as a reference and implement your own agent.
- Show the performance of your agent by letting the trained agent to play several random games and compare the total rewards with the maximum possible rewards.
- Notes: (1) You should implement your agent based on tf-agents. Although there are numerous examples of reinforcement learning codes on the internet, few of them use tf-agents (since it's new!) Thus you are expected to do your own work for this project. Though there are only a few examples you can find online, if you source any key ideas from elsewhere, you must reference the source of those ideas. (2) This project may be time and computation consuming, please plan ahead and be prepared to allocate adequate amount of time.

Project 3 (VAE for downstream classification)

A natural Bayesian representation of a large variety of real-world representations is that of a latent variable model, whereby the data we observe (e.g. the pixels of a photo) depend on some unknown latent feature (e.g. the type of object captured by a photo).

However, these latent models can be hard to describe (e.g. for a hand-written digits, one might not only consider the digit, but also the stroke, the angle, etc.), and even if we were able to do so, inference over large latent variable models can be challenging, as we have seen in the first part of the course.

Variational auto-encoders (VAE; `https://arxiv.org/pdf/1312.6114.pdf`) can be thought of as a nonlinear version of PCA, or in other words, a nonlinear dimensionality reduction technique. Recall that PCA finds a linear projection of our data onto a low-dimensional space, that gives the optimal reconstruction of the original data (in the L2 sense). A VAE is a generalization of such a method where instead of a linear projection we use a non-linear mapping (a neural network), and we define the reconstruction error in a probabilistic sense of our choosing (for image data, it is typically cross-entropy). Thus, one of the practical uses of a VAE is to map high-dimensional data into a space of much lower dimension, in a way that captures most of the sources of variation in the data. This low-dimensional representation can be used for downstream tasks such as classification (classifiers may generally benefit from having a small number of inputs relative to the number of datapoints in the training set). This will be particularly relevant in the setting where we have access to a large sample of unlabelled datapoints, but acquiring the relevant labels is expensive, so we only have access to a small number of labelled samples.

- Implement your own VAE for the fashion-MNIST data `https://github.com/zalandoresearch/fashion-mnist`. Your VAE should consist of a generative model (decoder), capable of generating new samples that look similar to the actual images in the dataset, as well as an inference network (encoder), that approximates the conditional distribution of the hidden variables given the observed image.
- As a sanity check that your generative model has learnt the correct distribution, show some generated samples, together with some actual samples from the data. Verify that they look similar.

- As a sanity check that your inference network is also correct, show some reconstructed samples, together with the originals, and verify that they look similar. In other words, take samples from the original data, run them through the inference network to obtain the corresponding hidden variables, and feed those back through the decoder network to obtain the reconstructed images.
- Now that we know our VAE is behaving reasonably well, we want to evaluate the usefulness of the learned representations for a downstream classification task. In this section, our VAE stays fixed, and we train some additional classifiers:
  - Take the images in your dataset, and feed them through the inference network to obtain the corresponding hidden variables.
  - Next, use the values of those hidden variables as the input to a logistic regression that predicts the class labels. You should repeat this process in two regimes: one where you train the logistic regression classifier on a small subset of the data, and another where you use a larger subset. Evaluate the resulting test error against a held-out dataset.
  - Do this again using a more flexible classifier: use a simple CNN (one convolutional layer and one fully connected layer) instead of logistic regression.
  - Compare the results of this process to directly training a multilayer CNN classifier on the original data, both in the setting where we train only on the small subset of the data, and the larger subset.
  - Discuss how the performance of the three methods (VAE + logistic regression, VAE + simple CNN, and multilayer CNN) varies across different data regimes. Which is better in which setting? In the 'small labelled data' regime, where we trained our classifier on a small number of samples, did the dimensionality reduction from the VAE help the simpler classifiers to achieve a better test error than the multilayer CNN classifier?

Project 4 (Neural style transfer)

Neural style transfer `https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf` is an optimization algorithm used to take a content image and a style image (such as a masterpiece by a famous artist), and blend them together so the output image looks like the content image, but "painted" in the style of the style image. In this project, you will implement your own neural style transfer algorithm to mix the content image with style image.

- Use pretrained image classification network (eg. 19-layer VGG network) to build the content and style representations, calculate the weighted loss function of content & style loss, and run gradient descent over the output image.
- Pick at least three images and two different styles (paintings, photographs, etc), test your algorithm on them.
- Study how the results would change with respect to training iterations, different random seeds and relative weight of content & style loss in the loss function.
- This project can be done without a GPU, and an advanced version of this project would involve a GPU.
- There are numerous examples of code doing similar things on the internet, though you are expected to do your own work for this project. If you source any key ideas from elsewhere, you must reference the source of those ideas.

Project 6 (Choose your own adventure)

You are also given the flexibility to design your own deep learning project, drawing on any material covered in the course (and perhaps some topics not covered).

Notes:

- This project should be understood to be harder than the other choices listed here; not only must you solve the problem, but you must identify the problem as well. We expect more effort to be put into this type of project than others.
- We are interested in solutions to real problems; for example, if you have a particularly exciting dataset from your research or a job or similar, that would be relevant.
- If work with a custom dataset, we expect you to show the iterative process through which you designed you model and how you evaluated its performance at each step. Only reporting the final model and results will not suffice.
- You are strongly encouraged to contact the TAs to refine the scope of your own project. However, before contacting the TAs you must flesh out your idea for the project and acquire all relevant materials (i.e. they will not be able to allocate resources to helping you find a dataset or identify relevant literature).