Advanced Machine Learning (GR5242)
Fall 2020

Final Projects

Due: Monday 21 December at 7pm (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/2020/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 (`TF_agents_intro.ipynb` attached here) 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 (Model design for deep learning in practice)

In real-world applications, deep learning is complicated by the existence of several important design choices that go into a neural network model. These choices include (but are not limited to):

- Network architecture. This includes the number of layers, the type of layers, e.g., fully connected, convolutional, pooling, residual connections, etc., and the activation functions.
- Optimization. This includes the optimization algorithm itself, e.g., SGD, Adagrad, Adam, etc., as well as the minibatch size, number of epochs, the learning rate, and other hyperparameters.
- Initialization. Common choices include initializing the network weights with independent normal or uniform draws, distributed with a pre-specified mean and variance.
- Regularization. This includes techniques such as dropout, batch normalization, and weight decay, as well as implicit regularization such as early stopping or data augmentation.
- Loss function. For simplicity, we will stick with the standard choices: in our case we will use cross-entropy loss for classification.

These choices can have a substantial impact on the performance of trained neural network models. As a result, it is natural to ask how robust the models are to these design choices. The aim of this project is to investigate how these choices can affect out-of-sample performance and whether good choices for one dataset translate into good choices for another dataset.

- Start by training an unregularized MLP with 2 hidden layers of 16 hidden units and ReLU activations on the MNIST hand-written digits dataset. Train the model for 100 epochs using the Adam optimizer with a minibatch size of 128. Use the default optimization and initialization hyperparameters from keras (don't forget to explicitly state what these are in your project write-up, so that the reader will know exactly what you are using). Report the out-of-sample accuracy on the MNIST test set.

- For each of (i) network architecture, (ii) optimization, (iii), initialization, and (iv) regularization, pick at least 3 different design choices, and retrain your network under these new settings. Report out-of-sample accuracies and plot them as a function of training epoch. In light of your results, carefully discuss how performance depends on the hyperparameters, and any relevant observations that you find of interest. For example, for (ii) one might choose to run Adam with learning rates of 10^{-2}, 10^{-3}, and 10^{-4}.
- Take the best model from the previous part, and retrain it on the fashion-MNIST dataset; what is the out of sample accuracy? Now re-run the entire previous step on the fashion-MNIST dataset. Do you observe similar conclusions when running the analysis on this new dataset? Did the best performing model from the previous part correspond to the best performing model on the new dataset? Discuss the implications in terms of generalization and robustness (or lack thereof) of our models relative to hyperparameter choices.

Project 4 (Neural style transfer)

Neural style transfer is an optimization algorithm used to take a "content" image and a "style" image, and blend them together so the output image contains the same objects of the content image, but "rendered" in the style of the style image: `https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf` In this project, you will implement your own neural style transfer algorithm to mix content images with style images.

- Use a pre-trained image classification network (eg. 19-layer VGG network) to build the content and style representations, implement the weighted loss function (with content & style loss), and run gradient descent over the output image.
- Pick two content images and two style images (paintings, photographs, etc), and test your algorithm over the 4 resulting combinations of content and style.
- 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. Discuss any findings that you find of interest.
- 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 5 (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 you work with a custom dataset, we expect you to show the iterative process through which you designed your 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).