EE660 Mini-Project

Assigned: 11/13/25, Due: 12/5/25

1 Description

The mini-project is a potentially open-ended project to give students the opportunity to apply the topics discussed in the course to problems of interest. Both projects which lean towards the theoretical side and/or experiment side are encouraged. Students may work either alone or in groups of two; **groups larger than two will not be allowed**. Below is a non-exhaustive list of both theoretical and experimental mini-projects which would be considered in scope for the course. Note that as this is a one month mini-project, research novelty is **not** required for a project to receive full credit (although will certainly not be discouraged). Finally, we note that we have prepared in detail a default project based on generative modeling: if you are not interested in doing an open-ended project, you may **skip directly to Section 2**.

Theoretical projects: Below is a non-exhaustive list of project ideas which are more on the theoretical side. Note that "incremental" extensions/modifications of existing results are in-scope, in addition to proving known results using alternative techniques. Furthermore, writing your own proofs based on existing literature is also acceptable as long as credit is properly attributed.

(a) Conduct a literature survey and describe the sharpest known bounds on the Rademacher complexity of deep neural networks. Include discussions on fully connected and CNNs, and also the activation function. References to consult: [1, 2, 3]

(b) Derive uniform concentration bounds, in addition to concentration bounds on error of kernel matrices versus their random feature approximations. Discuss when the resulting bounds provide computational advantages to using random features versus kernels. Evaluate the sharpness of the bounds in practice. References to consult: [4, 5].

(c) Use Bernstein's inequality to derive generalization bounds in the setting of finite hypothesis classes. Demonstrate that your bounds allow for *fast-rates*, i.e., rates of the form $1/n$ instead of $1/\sqrt{n}$, in certain favorable situations. References to consult: [6].

(d) Write a short survey paper about the Neural Tangent Kernel (NTK). Be sure to discuss topics such as NTK vs. standard initialization, lazy training, and connections to Gaussian processes. References to consult: [7, 8].

(e) Derive finite sample bounds for maximum likelihood estimation over finite function classes. Extend the result to parametric classes using covering numbers. References to consult: [9, Section B.4], [10, Section E].

(f) Consider simple Gaussian distributions to conduct a detailed, closed-form comparison between DDPM, DDIM, and the SDE version of diffusion models. References to consult: [11, 12, 13].

Experimental projects: Below is a non-exhaustive list of project ideas which are more on the experimental side. Note that re-implementations of existing algorithms/models are in-scope, as well as applications of specific algorithms/models to a problem domain:

(a) Perform an empirical study regarding sharpness of the generalization bounds we studied in this class. Consider both under and over-parameterized regimes. Compare the bounds to other holdout set methods [14, 15].

(b) Implement PAC-Bayes methods [16, 17] for deriving generalization bounds, and compare the PAC-Bayes bounds to traditional Rademacher/VC-dimension bounds, in addition to other holdout set methods [14, 15].

(c) Use sufficiently wide neural networks to fit data with arbitrary (random) labels to zero training error. Discuss the implications of this finding on deriving a generalization theory for neural networks. References to consult: [18]

(d) Compare the performance of kernel methods to their random feature counterparts on a few sample problems. Do there exists configurations where random features obtain nearly identical performance to using kernels, while providing clear computational gains?

(e) Investigate the impact of optimization algorithm on generalization performance in neural networks. Consider both first, second, and possibly third order methods (e.g., full GD, SGD, Adam, L-BFGS, BFGS, and cubic regularized Newton's method). Are you able to find situations where all algorithms achieve zero training error but some generalize noticeably better than others?

(f) Construct simple latent variable models where the exact posterior $p(z \mid x)$ is tracable to compute. Run various algorithms (such as EM and variational inference) applied to these simple latent variable models. Do these algorithms reliably recover the true latent parameters (up to unrecoverable symmetries)? Does the optimization method used in e.g., variational inference matter for generalization?

Deliverables: The main deliverable for this project is a project report in the form of a PDF. Please keep the main body of the project report to at most 6 pages inclusive (1 inch margins, 11 point font); there are no page limits for the references and any appendices. If your project involves any code/notebooks, please also submit these artifacts as well.

Deadlines:

(a) **11/17, 11:59pm Pacific Time:** Submit a short (at most one page) writeup describing your project at a high level. If you are working in a group, only one submission per group is necessary; please include the name of all groups members in your submission. The course staff will take a look at your proposed project and provide any necessary feedback/adjustments to keep it within scope. Note: if you are doing the default project (Section 2), do **not** submit a project proposal.

(b) **12/5, 11:59pm Pacific Time:** The project report (as a PDF), as well as any project (code, notebooks, etc.) artifacts. Note that the same deadline here applies to students who choose to do the default generative modeling project.

2 Default Project

If none of the topics above in Section 1 sound appealing, and if you are not able to think of another idea that you would be interested in working on for the mini-project, Section 3 contains a detailed description of a generative modeling project which we have already designed that you may choose to work on. If done correctly, this project will receive full credit. **However, if you choose to work on the project described in Section 3, you must work alone.** Also, if you choose to do this project, you do **not** need to submit a one page writeup outlining your proposed project as described in Section 1.

3 Generative Modeling Project

Your task is to implement two of the generative models listed below, and provide both a quantitative and qualitative assessment comparing the performance of the models you implemented. This project has two deliverables: (a) the code you wrote to complete the assignment (notebooks, modules, etc.), and (b) a short, write-up which will be described shortly.

Here is the list of models which you may choose from to implement:

(a) Energy-based models (EBM),

(b) Denoising score matching (DSM),

(c) Variational autoencoder (VAE),

(d) Denoising diffusion probabilistic model (DDPM),

(e) Continuous normalizing flow (CNF),

(f) Generative adversarial network (GAN).

(Note that score matching (SM) is excluded from this list above, because we will be providing you with an example notebook containing an implementation of SM.) Some of these methods will be covered in class, the others you can learn about by reading the lecture notes.

Dataset wise, the following two-dimensional dataset generators are provided for you in the notebook `dataset_generators.ipynb`:

(a) Spiral,

(b) Pinwheel,

(c) Checkerboard,

(d) Gaussian mixtures.

Here is a detailed breakdown of what is expected in this assignment, along with a grading breakdown (note that this assignment is out of 100 points):

Implement two generative models (50 points): Choose two generative models from the list above, and implement both the model's *training* procedure, in addition to the model's *sampling* procedure.

In all of these generative models, there is a point where a general purpose function approximator is needed. For example, in energy models, where $p_\theta(x) \propto \exp(f_\theta(x))$, the function approximator is

$f_\theta(x)$. As another example, in DDPM, we need to learn a denoiser $f_\theta(x, t)$ using a general function approximator. In this assignment, you are free to use any general function approximator you want (e.g., kernel machine, neural networks, random features, etc.). However, if you do not have any particular affinity for a specific approximator, we recommend using a simple fully connected multi-layer perceptron (MLP).

Furthermore, you are also free to use any software stack you wish. But again if you have no particular affinity, we recommend that you use `jax`,[1] in particular the `flax` library[2] to define neural networks, and the `optax` library[3] to implement gradient-based optimization. This is the same software ecosystem that the notebook we provide you is written in.

Regarding code reuse: the reuse of code snippets that you find online is permissible given proper attribution, **for code not directly related to the generative model**. For example, if you find a snippet online for defining a 3 layer neural network in `flax`, this is an acceptable piece of code reuse. However, copying an existing implementation of a VAE off the internet and submitting it as yours is **strictly prohibited**. You are of course allowed to look at somebody else's implementation to get an idea of how they e.g., defined the training loss, but you need to write your own original implementation. Another thing which is prohibited is directly calling a library which implements a generative model. For example, using `diffusers.DDPMPipeline` from huggingface's API is **not allowed**. Please exercise common sense in this manner, and if there is a piece of code reuse you are unsure about, feel free to come to office hours and discuss it with the course staff.

Full credit here involves turning in original, working code which trains on the datasets from the dataset generators and generates new samples. The easiest way to demonstrate this to perform the required analysis (described subsequently in the document) in a Jupyter notebook which clearly imports your model/train code (we recommend keeping the model/train code separate from the analysis notebook), and submit this notebook (with the cell outputs kept) along with the rest of your code. Note that the *quality* of the training/sampling is assessed later: this step is simply about producing original implementations which actually run.

Hints: we recommend selecting DSM as one of your model implementations, since it is very closely related to the SM example provided in the notebook. But this is not a requirement. Also, if you choose to implement a continuous normalizing flow, we recommend using `diffrax`[4] to define the log-probability ODE, since `diffrax` takes care of auto-differentiation through ODEs.

Train both models on two datasets listed above (20 points): Choose two datasets listed above and train both your generative models on $n = 2000$ training samples from each dataset (that is, you should have *four* model parameters in total: the cross product of two models and two dataset). In your writeup, include the training curves, showing the progress of training for each model/dataset. These training curves should typically be of the form where a loss function is plotted on the y-axis, and the number of gradient steps (if you are using gradient based optimization) is plotted on the x-axis. The precise value of the loss function plotted will be different for each model. For example, for VAE the loss would be the empirical ELBO, for DSM the loss would be the denoising score matching loss, etc. Also, in your write-up, plot the value of the loss function on $n = 500$ sample hold-out set, alongside with your training loss values, to show that your model

[1] `https://jax.readthedocs.io/en/latest/`

[2] `https://github.com/google/flax`

[3] `https://github.com/google-deepmind/optax`

[4] https://docs.kidger.site/diffrax/

is not overfitting.

Regarding hyperparameter values (e.g., the step sizes for gradient based optimization, width of the neural networks used, dimensionality of the latent variable in a VAE, etc.), we do *not* expect you to conduct an exhaustive hyperparameter sweep. Simply document in your write-up the various hyperparameter values that you played with, if necessary to improve the training performance and/or reduce overfitting; you only need to plot the train/hold-out curves for your best hyperparameter values.

Full credit for this step involves demonstrating that both your model implementations are able to stably decrease their training loss (or increase its objective depending on how you define it) on both datasets. In other words, the training curves should trend in the right direction (they certainly do not need to be monotonic, but the trend should be clear). There are two exceptions to this: (1) the training process of energy based model is quite difficult to stabilize, and (2) for GANs, since a min/max game is being optimized, the best we can hope for in the training curves is that the value stabilizes to a number (ideally this number is $-2\log 2$, but it is OK if your implementation does not converge precisely to $-2\log 2$). Therefore, for EBM and GAN training curves, we will be fairly lenient on the grading; feel free to check with the course staff in OH if you are wondering whether your implementation is behaving reasonably.

Qualitative comparison of model sample quality (15 points): Qualitatively compare the quality of the learned model samplers by drawing samples from each of your learned models, and comparing it to the samples from the true distribution. Specifically, for each (model, dataset) pair, draw N samples (N is a free parameter for you to decide) from both the model and the dataset generator, and visualize all the $2N$ cumulative samples on the same 2D plot (using one color for the model and another color for the dataset generator). Include this comparison in your write-up, and make an assessment based on your plots about which model generates samples with higher fidelity. Comment on any abnormalities in the samples generated by your models.

Note that there is no right answer here: depending on a combination of various factors including learning rates, function approximator capacity, etc., since the datasets we consider are relative simple, we do not expect there to be a clear-cut winner in every situation.

Quantitative comparison of model sample quality (15 points): The last part we ask is to give a *quantitative* comparison of the model sample quality. In general, this is actually quite difficult to do. But the fact that our datasets are all two dimensional, and the fact that we have a dataset generator available to us (not just a fixed training set) affords us some opportunities.

Here are a few ideas. For instance, imagine we are comparing a normalizing flow model to a VAE. For the normalizing flow, we could simple evaluate the average log-likelihood under our trained model of a hold out dataset, and that would give us an unbiased estimate of the true log-likelihood. On the other hand, for the VAE, we could evaluate a Monte-Carlo estimate of the ELBO on a holdout dataset, and use that as a lower bound of the true log-likelihood. Of course this methodology is not 100% satisfying since we not exactly comparing apples to apples by ignoring the gap between the ELBO and the log-likelihood, but for our purposes it will suffice. What if, on the other hand, our model does not have a likelihood function? Since our distributions are two dimensional, we can turn to kernel density estimation (KDE). We can fit a KDE to samples from our model, using say `scipy.stats.gaussian_kde`,[5] and use the KDE model to compute the

[5] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html`

log-likelihood (see the `logpdf` function). This can be done across models to get an apples-to-apples numerical comparison. See [19, Section 7.2] for an example of both a qualitative comparison, and a quantitative comparison between different generative models on different datasets.[6]

Of course, there are other methods. Again, there is no right method here. A full credit answer can use any method (including the ones described in the previous paragraph) that reasonably allows us to compare models numerically, as long as it is explained clearly what you are doing. If you come up with another method and want to discuss it with the course staff, again feel free to come to OH. Also, if you come across another evaluation methodology in a research paper that seems sound, feel free to cite it and use it (with proper explanation).

3.1 Deliverables

Here is a list of the items to submit on Brightspace by **12/5, 11:59pm Pacific Time**:

(a) Code (notebooks, modules, etc.).

(b) A writeup (in PDF format) describing which models you implemented, which datasets you trained your models on, the training curves (along with the hyperparameters you used for your model and training), the qualitative comparisons (figures and explanations), and finally the quantitative comparisons (tables containing the quantitative metrics and explanations). As described in Section 1, there is a 6 page main text limit (1 inch margins, 11pt font). However, for this generative modeling project, we encourage you to write concisely and target a 3 page writeup.

3.2 Questions and Answers

Can I implement another type of generative model not on this list? If you are interested in doing so, please email the course staff and we will discuss it.

What if I want to train my generative models on image datasets (e.g., MNIST/CIFAR10) instead of these 2D datasets? We chose to do 2D datasets for this assignment to ease computation requirements; none of the models trained for this assignment should require access to GPUs in order to train in a reasonable amount of time. However, if you have access to GPUs and are interested in working with image data, come speak with the course staff. While you will not earn any additional points for training on image datasets, this is something you are passionate about pursuing, we will be happy to accommodate your request.

3.3 Hints

Energy based models. If you plan to implement EBM training, we encourage to you take a look at [20], specifically Section 3, which discusses some of the difficulties which may arise during training. If you choose to implement some of these techniques to improve EBM training, please make note of it in your report.

[6] Note that the quantitative comparison here is a bit more complex than described in the main problem description, since it uses a KDE to approximate the integral defining the Bhattacharyya distance (`https://en.wikipedia.org/wiki/Bhattacharyya_distance`) between the true data distribution and the model's sampling distribution.

Denoising score matching. We found the following implementation details helped our DSM model train better:

(a) *Rewrite the loss:* The original DSM loss is

$$\hat{L}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \| f_\theta(\tilde{x}_i) - (x_i - \tilde{x}_i)/\sigma^2 \|^2.$$

One issue is that for small σ, the RHS can become quite large. Thus, we found it beneficial to use the following identity:

$$f_\theta(\tilde{x}_i) - (x_i - \tilde{x}_i)/\sigma^2 = \frac{1}{\sigma^2}\left[\sigma^2 f_\theta(\tilde{x}_i) - (x_i - \tilde{x}_i)\right],$$

which makes our loss:

$$\hat{L}_n(\theta) = \frac{1}{n\sigma^4}\sum_{i=1}^{n} \| \tilde{x}_i + \sigma^2 f_\theta(\tilde{x}_i) - x_i \|^2.$$

Furthermore, when implementing a batch gradient, we remove the scaling factor $1/(n\sigma^4)$.

(b) *Use multiple noise realizations for every x_i:* Fix a positive integer k. We let $\tilde{x}_i^j = x_i + \sigma w_i^j$, where $\{w_i^j\}_{i,j=1}^{n,k}$ are iid $N(0, I)$ noise vectors. With this notation, we consider the following variance-reduced loss (building on top of our previous modification):

$$\hat{L}_n(\theta) = \frac{1}{nk\sigma^4}\sum_{i=1}^{n}\sum_{j=1}^{k} \| \tilde{x}_i^j + \sigma^2 f_\theta(\tilde{x}_i^j) - x_i \|^2.$$

Furthermore, in forming a stochastic gradient, while we dropped the $1/(n\sigma^4)$ scaling factor, we kept the $1/k$ scaling factor (so that the scale of the batch gradient remains the same regardless of our choice of k).

(c) *Noisy training curve:* Due to the added noise to the loss function, the training curves for DSM will be substantially noiser than the training curves for regular score matching. This is to be expected. You may also plot (on top of the original training curve) a filtered version of the training loss running through an exponential moving average to smooth out the noise (this is not required, but may help in understanding if your model is actually training).

References

[1] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

[2] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Conference On Learning Theory*, pages 297–299. PMLR, 2018.

[3] Lan V Truong. On rademacher complexity-based generalization bounds for deep learning. *arXiv preprint arXiv:2208.04284*, 2022.

[4] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[5] Joel A. Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

[6] Patrick Rebeschini. Bernstein's concentration inequalities. fast rates, 2021. Accessed: 2024-11-04.

[7] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[8] Lilian Weng. Some math behind neural tangent kernel. *Lil'Log*, Sep 2022.

[9] Dylan J Foster, Adam Block, and Dipendra Misra. Is behavior cloning all you need? understanding horizon in imitation learning. *arXiv preprint arXiv:2407.15007*, 2024.

[10] Alekh Agarwal, Sham Kakade, Akshay Krishnamurthy, and Wen Sun. Flambe: Structural complexity and representation learning of low rank mdps. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20095–20107. Curran Associates, Inc., 2020.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[12] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

[13] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[14] John Langford. Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research*, 6(10):273–306, 2005.

[15] Anastasios Nikolas Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification, 2021.

[16] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

[17] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Nonvacuous generalization bounds at the imagenet scale: a PAC-bayesian compression approach. In *International Conference on Learning Representations*, 2019.

[18] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115, February 2021.

[19] Sumeet Singh, Stephen Tu, and Vikas Sindhwani. Revisiting energy based models as policies: Ranking noise contrastive estimation and interpolating energy models. *arXiv preprint arXiv:2309.05803*, 2023.

[20] Yang Song and Diederik P Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.