EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Homework 10

This homework is due 8th December 2022, at 10:59PM.

Deliverables: Please submit the code/notebooks/saved model as a zip file to the code gradescope assignment. Submit your written answers in the written gradescope assignment, and attach a pdf printout of the notebooks.

1. Prompting Language Models

(a) **Exploring Pretrained LMs**

Play around with the GPT-3 web interface at https://beta.openai.com/playground. If you have not visited these sites before, you'll need to sign up for an account. If you use GPT-3, you will start with $18 of free credit. This should be more than enough to complete the assignment (the assignment will probably take <$1), but be careful not to run out if you run extra experiments. If you have already used up your free credit and do not want to pay for this assignment, you can alternatively use free models from Cohere https://os.cohere.ai/playground for this entire problem.

In the playground, turn "Show probabilities" on to "Full spectrum". Spend a while exploring prompting these models for different tasks. Here are some suggestions:

- Look through the 'Load a preset . . . ' button at the top of the page for example prompts.
- Ask the model to answer factual questions.
- Prompt the model to generate a list of 100 numbers sampled uniformly between 0 and 9. Are the numbers actually randomly distributed?
- Insert a poorly written sentence, and have the model correct the errors.
- Have the model brainstorm creative ideas (names for a storybook character, recipes, solutions to solve a problem, etc.)
- Chat with the model like a chatbot.

Answer the questions below:

i. Describe one new thing you learned by playing with these models. **Solution:** Answers may vary.

ii. How does the temperature parameter affect the outputs? Justify your answer with a few examples. **Solution:** When temperature = 0, the model is deterministic and outputs the greedy argmax answer every time you generate with the same prompt. When the temperature is higher, results are different every time, and the model is more likely to have weird, creative, and nonsensical outputs.

iii. Describe a task where the larger models significantly outperform the smaller ones. Paste in examples from the biggest and smallest model to show this. **Solution:** Answers may vary.

iv. Describe a task where even the largest model performs badly. Paste in an example to show this. **Solution:** Answers may vary

v. Describe a task where the model's outputs improve significantly with few-shot prompting compared to zero-shot prompting. **Solution:** Answers may vary, but this is often the case when you want anwers to be in a specific output format.

vi. (Optional) Click on sampled tokens to see other high-probability choices. Describe any interesting findings about the probability distributions you see or about the tokenization scheme. **Solution:** Answers may vary.

(b) **Using LMs for classification**

If you did not do part (a), you will still need to get an OpenAI account to complete this part. (Cohere is also acceptable). Run the notebook, then answer the following questions:

i. Analyze the GPT3 model's failures. What kinds of failures do you see with different prompting strategies? **Solution:** For SimplePrompt and SimpleQA prompt, many of the errors are the model outputting invalid solutions (especially newline characters with the SimplePrompt.)
For QAInstruction and the two FewShot variants, failures are mostly choosing the incorrect answer. A large portion of the incorrect answers are choices which seem reasonable even to a human.

ii. Does providing correct labels in few-shot prompting have a significant impact on accuracy? **Solution:** Answers may vary depending on how many data points you use, but you should see that the accuracy with incorrect labels in the prompt is similar to or slightly worse than with clean prompts. (Confidence decreases slightly too.)

iii. Observe the model's log probabilities. Does it seem more confident when it is correct than when it is incorrect? **Solution:** The model is on average more confident when it is correct, though which prompt strategy is being used is more correlated with confidence than correctness/incorrectness.

iv. Why do you think the GPT2 model performed so much worse than the GPT3 model on the question answering task? **Solution:** The GPT2 model is much smaller and trained on less data.

v. How did soft prompting compare to hard prompting on the pluralize task? **Solution:** At convergence, the soft prompt significantly outperforms hard prompts, even with several examples in the hard prompt.

vi. You should see that when the model fails (especially early in training of a soft prompt or with a bad hard prompt) it often outputs common but uninformative tokens such as the, ", or \n. Why does this occur? **Solution:** When the model is uncertain which token comes next, the most likely token is often a token which commonly occurs throughout most text corpuses.

2. Variational AutoEncoders

(Parts of this problem are adapted from Deep Generative Models, Stanford University)

For this problem we will be using PyTorch to implement the variational autoencoder (VAE) and learn a probabilistic model of the MNIST dataset of handwritten digits. Formally, we observe a sequence of binary pixels $\mathbf{x} \in \{0,1\}^d$ and let $\mathbf{z} \in \mathbb{R}^k$ denote a set of latent variables. Our goal is to learn a latent variable model $p_\theta(\mathbf{x})$ of the high-dimensional data distribution $p_{data}(\mathbf{x})$.

The VAE is a latent variable model with a specific parameterization $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$ Specifically, VAE is defined by the following generative process (often called **reparameterization trick**):

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I) \qquad \textit{(sample noise from standard Gaussian)}$$
$$p_\theta(\mathbf{x}|\mathbf{z}) = \text{Bern}(\mathbf{x}|f_\theta(\mathbf{z})) \qquad \textit{(decode noise to generate sample from real-distribution)}$$

That is, we assume that the latent variables $\mathbf{z}$ are sampled from a unit Gaussian distribution $\mathcal{N}(\mathbf{z}|0, I)$. The latent $\mathbf{z}$ are then passed through a neural network decoder $f_\theta(\cdot)$ to obtain the parameters of the d Bernoulli random variables that model the pixels in each image.

To learn the parameterized distibution we would like to maximize the marginal likelihood $p_\theta(\mathbf{x})$. However computing $p_\theta(\mathbf{x}) = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$ is generally intractable since this requires integrating over all possible values of $\mathbf{z} \in \mathbb{R}$. Instead, we consider a variational approximation to the true posterior

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\Big(\mathbf{z}|\mu_\phi(\mathbf{x}), \text{diag}(\sigma^2_\phi(\mathbf{x}))\Big)$$

In particular, we pass each image $\mathbf{x}$ through a neural network that outputs mean μ_ϕ and diagonal covariance $\text{diag}(\sigma^2_\phi(\mathbf{x}))$ of the multivariate Gaussian distribution that approximates the distribution over the latent variables $\mathbf{z}$ given $\mathbf{x}$. The high level intuition for training parameters (θ, ϕ) requires considering two expressions:

- **Decoding Latents** : Sample latents from $q_\phi(\mathbf{z})$, maximize likelihood of generating samples $\mathbf{x} \sim p_{data}$
- **Matching Prior** : A Kullback-Leibler (KL) term to constraint $q_\phi(\mathbf{z})$ to be close to the $p(\mathbf{z})$

Putting these terms together, gives us a lower-bound of the true marginal log-likehood, called the **evidence lower bound** (ELBO):

$$\log p_\theta(\mathbf{x}) \geq \text{ELBO}(\mathbf{x}; \theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Decoding Latents}} - \underbrace{D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{Matching Prior}}$$

(a) Implement the reparameterization trick in the function `sample_gaussian`. Specifically, your answer will take in the mean m and variance v of the Gaussian and return a sample $\mathbf{x} \sim \mathcal{N}(m, diag(v))$

Solution: Please see soln/utils.py

(b) Now, implement `negative_elbo_bound` loss function.

Note: We ask for the negative ELBO, as PyTorch optimizers minimze the loss function. Furthere, since we are computing the negative ELBO over a mini-batch of data $\{x^{(i)}\}_{i=1}^n$, make sure to compute the average of per-sample ELBO. Finally, note that the ELBO itself cannot be computed exactly since computation of the reconstruction term is intractable. Instead, you should estaimate the reconstruction term via Monte-Carlo sampling

$$-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \approx -\log p_\theta(\mathbf{x}|\mathbf{z}^{(1)})$$

where $\mathbf{z}^{(1)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ *denotes a single sample from the learned posterior.*

The `negative_elbo_bound` expects as output three quantities: *average* **negative ELBO, reconstruction loss, KL divergence.**

Solution: Please see soln/vae.py

(c) Test your implementation by training VAE with

```
python experiment.py --model vae
```

Once the run is complete (10000 iterations), it outputs : the *average*

- negative ELBO
- KL-Divergence term
- reconstruction loss

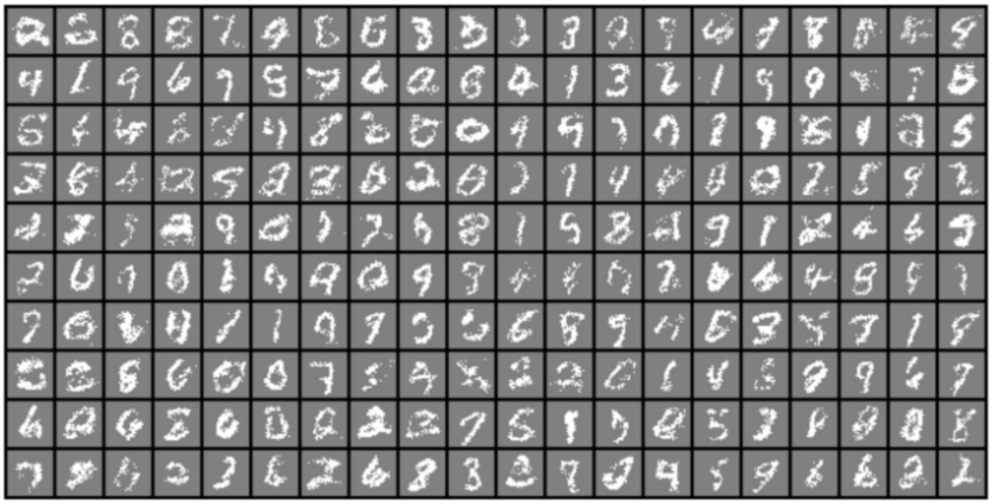
Since we're using stochastic optimization, you may wish to run the model multipple times and report each metric's mean and corresponding standard error *(Hint: the negative ELBO on the test subset should be* ~ 100*)*

Solution:

- **negative ELBO** : 98.12 ± 0.77
- **KL-Divergence** : 20.38 ± 0.38
- **reconstruction loss** : 77.93 ± 0.65

(d) Visualize 200 digits (generate a single image tiled in a grid of 10×20 digits) sampled from $p_\theta(\mathbf{x})$

Solution: Solutions should show visible digits, for example the img below (taken from a student submission).



3. Generative Adversarial Networks

(Parts of this problem are adapted from Deep Generative Models, Stanford University)

Unlike VAEs, that explicitily model data distributions with likelihood-based training, Generative Adversarial Networks (GANs) belong to the family of implicit generative models.

To model high-dimensional data distributions $p_{\text{data}}(\mathbf{x})$ (with $\mathbf{x} \in \mathbb{R}^n$), define

- a generator $G_\theta : \mathbb{R}^k \to \mathbb{R}^n$
- a discriminator $D_\phi : \mathbb{R}^n \to (0, 1)$

To obtain samples from the generator, we first sample a k-dimensional random vector $\mathbf{z} \sim \mathcal{N}(0, 1)$ and return $G_\theta(\mathbf{z}) \in \mathbb{R}^n$. The discriminator is effectively a classifer that judges how realistic the *fake* image $G_\theta(\mathbf{z})$ are, compared to *real* samples from the data distribution $x \sim p_{\text{data}}(\mathbf{x})$. Because its output is intended to be interpreted as a probability, the last layer of the discriminator is frequently the **sigmoid** function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

such that $\sigma(x) \in (0, 1)$. Therefore, for logits $h_\phi(\mathbf{x})$, discriminator output is $D_\phi(\mathbf{x}) = \sigma(h_\phi(\mathbf{x}))$.

For training GANs we define learning objectives $L_{\text{discriminator}}(\phi; \theta)$ and $L_{\text{generator}}(\theta; \phi)$ that are optimized iteratively in two-stages with gradient descent. In particular, we take a gradient step to minimize $L_{\text{discriminator}}(\phi; \theta)$ w.r.t discriminator parameters ϕ, followed by gradient step to minimize $L_{\text{generator}}(\theta; \phi)$ w.r.t. generator parameters θ. In lecture we've considered following versions of the losses:

$$L_{\text{discriminator}}(\phi; \theta) = \underbrace{-\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}\Big[\log D_\phi(\mathbf{x})\Big]}_{\text{Real Data}} - \underbrace{\mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\Big[\log\big(1 - D_\phi(G_\theta(\mathbf{z}))\big)\Big]}_{\text{Generated Data}}$$

$$L^{\text{minimax}}_{\text{generator}}(\theta; \phi) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\Big[\log\big(1 - D_\phi(G_\theta(\mathbf{z}))\big)\Big]$$

Training a GAN can be viewed as solving the following minimax optimization problem, for generator G_θ and discriminator D_ϕ:

$$\min_G \max_D V(G, D) \equiv \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}\Big[\log D_\phi(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\Big[\log\big(1 - D_\phi(G_\theta(\mathbf{z}))\big)\Big]$$

(a) **Vanishing Gradient with Minimax Objective**

Rewriting the above loss in terms of discriminator logits, sigmoid we have

$$L^{\text{minimax}}_{\text{generator}}(\theta; \phi) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\Big[\log\big(1 - \sigma(h_\phi(G_\theta(\mathbf{z})))\big)\Big]$$

Show that $\nabla_\theta L^{\text{minimax}}_{\text{generator}}(\theta; \phi) \to 0$ when discriminator output $D_\phi(G_\theta(\mathbf{z})) \approx 0$. Why is this problematic for training the generator when the discriminator is well-trained in identifying fake samples?

Solution: Recall that for sigmoid activation $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Taking the gradient w.r.t θ

$$\begin{aligned}
\frac{\partial L^{\text{minimax}}_{\text{generator}}}{\partial \theta} &= \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\left[\frac{-\sigma'(h_\phi(G_\theta(\mathbf{z})))}{1 - \sigma(h_\phi(G_\theta(\mathbf{z})))} \frac{\partial}{\partial \theta} h_\phi(G_\theta(\mathbf{z}))\right] \\
&= \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0,I)}\left[\frac{-\sigma(h_\phi(G_\theta(\mathbf{z})))(1 - \sigma(h_\phi(G_\theta(\mathbf{z}))))}{1 - \sigma(h_\phi(G_\theta(\mathbf{z})))} \frac{\partial}{\partial \theta} h_\phi(G_\theta(\mathbf{z}))\right]
\end{aligned}$$

$$= \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\Big[-\sigma(h_\phi(G_\theta(\mathbf{z})))\frac{\partial}{\partial\theta}h_\phi(G_\theta(\mathbf{z}))\Big]$$
$$= -\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\Big[D_\phi(G_\theta(\mathbf{z}))\frac{\partial}{\partial\theta}h_\phi(G_\theta(\mathbf{z}))\Big]$$

From the above derivation, it follows that for $D_\phi(G_\theta(\mathbf{z})) \approx 0$, suggesting that $\frac{\partial L^{\text{minimax}}_{\text{generator}}}{\partial\theta} \to 0$. As the generator update is proportional to the gradient, the vanishing gradient causes generator optimization to be slow.

(b) **GANs as Divergence Minimization**

To build intuition about the training objective, consider the distribution $p_\theta(\mathbf{x})$ corresponding to:

$$\mathbf{x} = G_\theta(\mathbf{z}) \qquad \text{where } \mathbf{z} \sim \mathcal{N}(0, I)$$

- **Optimal Discriminator**

  The discriminator minimizes the loss

$$L_{\text{discriminator}}(\phi;\theta) = -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\Big[\log D_\phi(\mathbf{x})\Big] - \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\big(1 - D_\phi(\mathbf{x})\big)\Big]$$

  For a fixed generator θ, show that the discriminator loss is minimized when $D^*_\phi = \frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})+p_{\text{data}}(\mathbf{x})}$.

  Solution: Rewriting the discriminator loss, we have

$$\begin{aligned} L_{\text{discriminator}}(\phi;\theta) &= -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\Big[\log D_\phi(\mathbf{x})\Big] - \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\big(1 - D_\phi(\mathbf{x})\big)\Big] \\ &= -\int p_{data}(\mathbf{x})\log D_\phi(\mathbf{x})d\mathbf{x} - \int p_\theta(\mathbf{x})\log\big(1 - D_\phi(\mathbf{x})\big)d\mathbf{x} \\ &= \int f(D_\phi(\mathbf{x}))d\mathbf{x} \end{aligned}$$

  where $f(t) = -p_{data}(x)\log t - p_\theta(x)\log(1-t)$ with $t = D_\phi(\mathbf{x})$. Note that the above function is a sum of two strictly convex functions, and is therefore convex, i.e. there exists a unique optimal solution t^* that minimizes $f(t)$.

$$\begin{aligned} f'(t) &= -\frac{p_{data}(x)}{t} + \frac{p_\theta(x)}{1-t} \equiv 0 \\ \implies tp_\theta(x) &= (1-t)p_{data}(x) \\ \therefore t^* &= \frac{p_{data}(x)}{p_\theta(x) + p_{data}(x)} \end{aligned}$$

  Therefore for each $\mathbf{x}$, we obtain the optimal discriminator by setting $t^* = D_\phi(\mathbf{x}) = \frac{p_{data}(x)}{p_\theta(x)+p_{data}(x)}$

- **Generator Loss**

  For a fixed generator θ, and corresponding optimal discriminator D^*_ϕ, show that the minimax objective $V(G, D^*)$ satisfies

$$V(G, D^*) = -\log 4 + 2D_{\text{JSD}}(p_{\text{data}}||p_\theta)$$

  where $D_{\text{JSD}}(p||q)$ is the Jenson-Shannon Divergence.

  Note: A divergence measures the distance between two distributions p, q. In particular, for distri-

butions p, q with common support $\mathcal{X}$, typically used divergence metrics include

$$D_{\mathrm{KL}}(p||q) = \mathbb{E}_{\mathbf{x}\sim p}\Big[\log\frac{p(x)}{q(x)}\Big] \qquad \textit{(Kullback-Leibler Divergence)}$$

$$D_{\mathrm{JSD}}(p||q) = \frac{1}{2}D_{\mathrm{KL}}\Big(p||\frac{p+q}{2}\Big) + \frac{1}{2}D_{\mathrm{KL}}\Big(q||\frac{p+q}{2}\Big) \qquad \textit{(Jensen-Shannon Divergence)}$$

Solution: Consider the learning objective

$$V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\log D_\phi(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\big(1 - D_\phi(\mathbf{x})\big)\Big]$$

For the optimal discriminator we know $D_\phi(\mathbf{x}) = \frac{p_{data}(x)}{p_\theta(x)+p_{data}(x)}$. Substituting the above in the learning objective, we have

$$\begin{aligned}
V(G,D) &= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\log\frac{p_{data}(x)}{p_\theta(x)+p_{data}(x)}\Big] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\Big(1 - \frac{p_{data}(x)}{p_\theta(x)+p_{data}(x)}\Big)\Big]\\
&= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\log\frac{p_{data}(x)}{p_\theta(x)+p_{data}(x)}\Big] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\frac{p_\theta(x)}{p_\theta(x)+p_{data}(x)}\Big]\\
&= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\log\frac{p_{data}(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}\cdot 2}\Big] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\frac{p_\theta(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}\cdot 2}\Big]\\
&= -\log 4 + \Big[\log\frac{p_{data}(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}}\Big] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\Big[\log\frac{p_\theta(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}}\Big]\\
&= -\log 4 + 2D_{JSD}(p_\theta||p_{data})
\end{aligned}$$

□

(c) **Training GANs on MNIST**

To mitigate vanishing gradients during training, (1) propose the non-saturating loss

$$L^{\mathrm{ns}}_{\mathrm{generator}}(\theta;\phi) = -\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\Big[\log D_\phi(G_\theta(\mathbf{z}))\Big]$$

For mini-batch approximation, we use Monte-Carlo estimates of the learning objectives, such that

$$\begin{aligned}
L_{\mathrm{discriminator}}(\phi;\theta) &\approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(\mathbf{x}^{(i)}) - \frac{1}{m}\sum_{i=1}^{m}\log\Big(1 - D_\phi(G_\theta(\mathbf{z}^{(i)}))\Big)\\
L^{\mathrm{ns}}_{\mathrm{generator}}(\phi;\theta) &\approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(G_\theta(\mathbf{z}^{(i)}))
\end{aligned}$$

for batch-size m, and batches of *real-data* $\mathbf{x}^{(i)} \sim p_{\mathrm{data}}(\mathbf{x})$ and *fake-data* $\mathbf{z}^{(i)} \sim \mathcal{N}(0,I)$. Following these details, implement training for GANs with above learning objectives by filling relevant snippets in `gan.py`. Test your implementation by running

```
python experiment.py --model gan
```

Visualize 200 digits (generate a single image tiled in a grid of 10 × 20 digits) sampled from $p_\theta(x)$

Solution: Please see soln/gan.py

4. Diffusion Models

The classes of generative models we've considered so far (VAEs, GANs), typically introduce some sort of bottleneck (*latent representation* $\mathbf{z}$) that captures the essence of the high-dimensional sample space $(\mathbf{x})$. An alternate view of representing probability distributions $p(\mathbf{x})$ is by reasoning about the *score function* i.e. the gradient of the log probability density function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$.

Given a data point sampled from a real data distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, let us define a *forward diffusion process* iteratively adding small amount of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $\mathbf{x}_1, ..\mathbf{x}_T$.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I) \qquad q(\mathbf{x}_{1:T}|x_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{1}$$

The data sample $\mathbf{x}_0$ gradually loses its distinguishable features as the step t becomes larger. Eventually when $T \to \infty$, $\mathbf{x}_T$ is equivalent to an isotropic Gaussian distribution. (You can assume $\mathbf{x}_0$ is Gaussian).

To generative model is therefore the *reverse diffusion process*, where we sample noise from an isotropic Gaussian, and iteratively refine it towards a realistic sample by reasoning about $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$.

(a) **Anytime Sampling from Intermediate Distributions**

Given $\mathbf{x}_0$ and the stochastic process in eq. (1), show that there exists a closed form distribution for sampling directly at the t^{th} time-step of the form

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)I)$$

Solution: Recall the reparameterization trick, where to sample from a Gaussian $\mathbf{x} \sim \mathcal{N}(\mu, \sigma^2)$, we could consider the following sampling process:

$$\mathbf{x} = \mu + \sigma\epsilon \qquad \text{where } \epsilon \sim \mathcal{N}(0, 1)$$

Therefore, defining $\gamma_t = 1 - \beta_t$, we have

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\gamma_t}\mathbf{x}_{t-1} + \sqrt{(1-\gamma_t)}\epsilon_{t-1} && \text{where } \epsilon_{t-1} \sim \mathcal{N}(0, I) \\ &= \sqrt{\gamma_t}\left(\sqrt{\gamma_{t-1}}\mathbf{x}_{t-2} + \sqrt{(1-\gamma_{t-1})}\epsilon_{t-2}\right) + \sqrt{(1-\gamma_t)}\epsilon_{t-1} && \text{where } \epsilon_{t-2} \sim \mathcal{N}(0, I) \end{aligned}$$

To simplify this, recall the following lemma, where mixing two Gausssians $\mathcal{N}(0, \sigma_1^2)$ and $\mathcal{N}(0, \sigma_2^2)$ gives a Gaussian $\mathcal{N}(0, \sigma_1^2 + \sigma_2^2)$. Therefore, mixing samples ϵ_1, ϵ_2. Building on this insight, we can combine the noise components ϵ_1, ϵ_2 into a new random variable:

$$\begin{aligned} \hat{\epsilon}_{t-2} &\sim \mathcal{N}(0, (\gamma_t(1-\gamma_{t-1}) + (1-\gamma_t))I) \\ &\sim \mathcal{N}(0, (1-\gamma_t\gamma_{t-1})I) \\ \therefore \mathbf{x}_t &= \sqrt{\gamma_t\gamma_{t-1}}\mathbf{x}_{t-2} + \sqrt{(1-\gamma_t\gamma_{t-1})}\hat{\epsilon}_{t-2} \end{aligned}$$

Unrolling this recursion, we would get the base case, where for $\mathbf{x}_0$ the samples are

$$\mathbf{x}_t = \sqrt{\Pi_{i=1}^t \gamma_i \mathbf{x}_0} + \sqrt{1 - \Pi_{i=1}^t \gamma_i \epsilon}$$

Therefore, by introducting $\alpha_t = \Pi_{i=1}^t \gamma_i$ we get that

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)I)$$

(b) **Reversing the Diffusion Process**

Reversing the diffusion process from *real* to *noise* would allow us to sample from the real data distribution. In particular, we would want to draw samples from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Show that given $\mathbf{x}_0$, the reverse conditional probability distribution is tractable and given by

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \mu(\mathbf{x}_t, \mathbf{x}_0), \hat{\beta}_t I)$$

(Hint: Use Bayes Rule on eq. (1) *, assuming that* $\mathbf{x}_0$ *is drawn from Gaussian* $q(\mathbf{x})$*)*

Solution: Applying Bayes rule on $q(x_t|x_{t-1}, x_0)$ we get the following expression

$$q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

From part (a) we know the densities as

$$\begin{aligned} q(x_t|x_0) &\sim \mathcal{N}(\sqrt{\alpha_t}x_0, (1-\alpha_t)I) \\ q(x_t|x_{t-1}, x_0) &\sim \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I) \end{aligned}$$

Therefore by plugging into the Bayes rule, we recover (upto proportionality constants)

$$\begin{aligned} q(x_{t-1}|x_t, x_0) &\propto \exp\Big(-\frac{1}{2}\Big\{\frac{(x_t - \sqrt{1-\beta_t}x_{t-1})^2}{\beta_t} + \frac{(x_{t-1} - \sqrt{\alpha_{t-1}}x_0)^2}{1-\alpha_{t-1}} - \frac{(x_t - \sqrt{\alpha_t}x_0)^2}{1-\alpha_t}\Big\}\Big) \\ &\propto \exp\Big(-\frac{1}{2}\Big\{\frac{x_t^2 - 2\sqrt{1-\beta_t}x_{t-1}x_t + (1-\beta_t)x_{t-1}^2}{\beta_t} + \\ &\qquad \frac{x_{t-1}^2 - 2\sqrt{\alpha_{t-1}}x_0x_{t-1} + \alpha_{t-1}x_0^2}{1-\alpha_{t-1}} - \frac{(x_t - \sqrt{\alpha_t}x_0)^2}{1-\alpha_t}\Big\}\Big) \end{aligned}$$

Simplifying the expression we get

$$q(x_{t-1}|x_t, x_0) \propto \exp\Big(-\frac{1}{2}\Big\{(\frac{1-\beta_t}{\beta_t} + \frac{1}{1-\alpha_t})x_{t-1}^2 - (\frac{2\sqrt{1-\beta_t}}{\beta_t}x_t + \frac{2\sqrt{\alpha_t}}{1-\alpha_t}x_0)x_{t-1} + H(x_t, x_0)\Big\}\Big)$$

where $H(x_t, x_0)$ is independent of x_{t-1} and therefore would be normalized out. Comparing to the expression for Gaussian $\mathcal{N}(\mu, \sigma^2)$

$$\mathcal{N}(\mu, \sigma^2) \propto \exp\Big(-\frac{1}{2}\Big\{\frac{x^2 - 2\mu x + \mu^2}{\sigma^2}\Big\}\Big)$$

we recover the expression for mean, variance of $q(x_{t-1}|x_t, x_0)$ as

$$\begin{aligned} \hat{\beta}_t &= 1/\Big(\frac{1-\beta_t}{\beta_t} + \frac{1}{1-\alpha_t}\Big) \\ &= \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t \qquad\qquad \Big(\text{recall } \alpha_t = \prod_{i=1}^{T}(1-\beta_t)\Big) \end{aligned}$$

$$
\begin{aligned}
\mu(x_t, x_0) &= \Big(\frac{\sqrt{1-\beta_t}}{\beta_t}x_t + \frac{\sqrt{\alpha_t}}{1-\alpha_t}x_0\Big)\Big/\Big(\frac{1-\beta_t}{\beta_t} + \frac{1}{1-\alpha_t}\Big) \\
&= \frac{\sqrt{1-\beta_t}(1-\alpha_t)}{1-\alpha_t}x_t + \frac{\beta_t\sqrt{\alpha_{t-1}}}{1-\alpha_t}x_0
\end{aligned}
$$

Therefore, under our assumptions, the distribution of $q(x_{t-1}|x_t, x_0) \sim \mathcal{N}(\mu(x_t, x_0), \hat{\beta}_t I)$. □

5. Continual Learning

We will explore some strategies that we can mitigate catastrophic forgetting when our neural network model sequentially learns the new tasks. Let's try and compare three classic methods: 1) naive 2) Elastic Weight Consolidation (EWC) and 3) Rehearsal.

(a) Naive approach

i. What do you observe? How much does the network forget from the previous tasks? Why do you think this happens?

Solution: The network forgets a lot from the previous tasks. This happens because the network is trained on each task separately, so it does not have access to the previous tasks when it is trained on the current task.

ii. (Open-ended question) We are using CNN. Does MLP perform better or worse than CNN? Try it out and report your results.

Solution: Any reasonable answer is correct. The example answer is: MLP will perform better than CNN. This is because CNNs utilize the spatial structure of the images, and the permuted MNIST images are not spatially structured.

(b) Elastic Weight Consolidation

i. Hyperparameter is underexplored in this assignment. Try different values of λ and report your results.

Solution: Every value students have tried is correct

ii. What is the role of λ? What happens if λ is too small or too large? Explain the results with plasticity and stability of the network.

Solution: λ controls plasticity and stability of the network. If it's small, the model becomes more plastic and vice versa

(c) Rehearsal

i. What would be the pros and cons of rehearsal? **Solution:** Pros: good performance, low levels of catastrophic forgetting; Cons: memory and computational cost of saving/re-training on past tasks.

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, 2014.

Contributors:

- Bryan Wu.
- Olivia Watkins.
- Kumar Krishna Agrawal.
- Anant Sahai.
- Aditya Grover.
- Stefano Ermon.
- Suhong Moon.