EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Homework 11

This homework is due on Friday, April 28, 2022, at 10:59PM.

1. Coding Question: Summarization (Part II)

Please follow the instructions in this notebook. You will learn how to efficiently enable the Transformer encoder-decoder model to generate sequences. Then, you will fine-tune another Transformer encoder-decoder model based on the pretrained language model T5 for the same task.

Note: this notebook takes **45 min to 1 hour** to run after you implemented everything correctly. Please start early.

- Download `submission_log.json` and submit it to "Homework 11 (Code) (Summarization)" in Gradescope.
- Answer the following questions in your submission of the written assignment:

(a) **What are the ROUGE-1, ROUGE-2, and ROUGE-L scores of the model trained from scratch on the summarization task?**

Solution: The model's ROUGE-1 score is 18.82, its ROUGE-2 score is 3.02, and its ROUGE-L score is 15.19. These results may vary slightly due to numerical errors, but deviations should not exceed 1.0.

(b) **In the T5 paper, which optimizer is employed for training T5, and what are the peak learning rates for pretraining and finetuning?**

Solution: As stated in Section 3.1.2 "Training" of the T5 paper, the optimizer used for training T5 is AdaFactor. The peak learning rate for pretraining is 0.01, while the peak learning rate for finetuning is 0.001.

(c) **What are the ROUGE-1, ROUGE-2, and ROUGE-L scores of the finetuned model, and how do these scores compare to those of the model trained from scratch?**

Solution: The model's ROUGE-1 score is 26.56, its ROUGE-2 score is 7.04, and its ROUGE-L score is 20.75. These results may vary slightly due to numerical errors, but deviations should not exceed 1.0. In comparison, the model finetuned from T5 has much higher performance according to all three types of ROUGE scores.

(d) Provide appropriate prompts for the summarization task, apply the prompt to the example documents, and fed them into an off-the-shelf large language model (e.g., ChatGPT). **Specify the model used, and share the generated outputs. Compare the outputs with your finetuned model's outputs qualitatively and identify the main reason for any differences.**

Solution: A reference prompt:

```
Summarize this document in one sentence:  [DOCUMENT]
```

Example output for the first document (from GPT-4 version of ChatGPT):

This document discusses the under-researched status of prostate cancer, advancements in radiotherapy, the discovery of genetic markers, and the improvement of surgical techniques using the DaVinci surgical robot to enhance treatment outcomes.

Example output for the second document (from GPT-4 version of ChatGPT):

The European Space Agency astronaut will conduct experiments on the International Space Station, focusing on metal physics, machine technology, and the impact of space on the human body, including body clock, nutrition, skin, and headaches.

Comparison:

Neither the model's output nor GPT-4's output matches the reference summary, because the summarization task is difficult and the evaluation is often subjective. Different summaries focus on various perspectives of the document. ChatGPT's answer is longer and more natural, as it was finetuned on human instructions.

(Please note that this is an open-ended question, and the answer may vary depending on the prompts, the large language model, and random seeds.)

2. Comparing Distributions

Divergence metrics provide a principled measure of difference between a pair of distributions (P, Q). One such example is the Kullback-Leibler Divergence, that is defined as

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P(x)}\Big[\log \frac{P(x)}{Q(x)}\Big]$$

(a) Technically D_{KL} is not a true distance since it is asymmetric, i.e. generally $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$. **Give an example of univariate distributions P and Q where $D_{\mathrm{KL}}(P||Q) \neq \infty$, $D_{\mathrm{KL}}(Q||P) = \infty$.**

Solution: To get $D_{\mathrm{KL}}(Q||P) = \infty$, the easiest way is to have $P(x) = 0$ somewhere where $Q(x) \neq 0$. For example, we can have P be a uniform random variable on $[0, +1]$ and Q be a uniform random variable on $[-2, 2]$. With this guess, we can now look at $D_{\mathrm{KL}}(P||Q)$ to verify that it is not infinity.

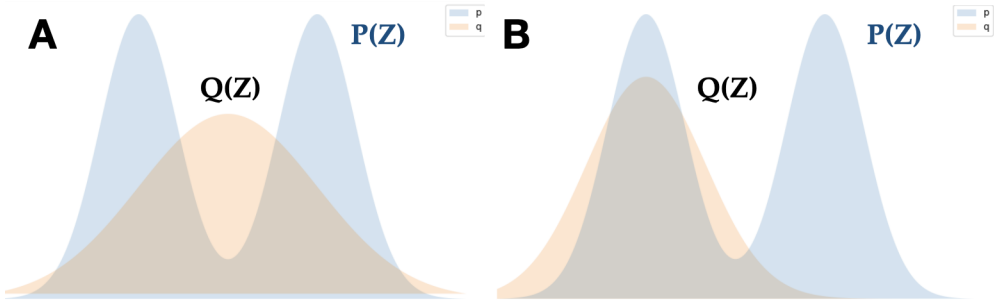
$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \mathbb{E}_{x\sim P(x)}\Big[\log \frac{P(x)}{Q(x)}\Big] \\ &= \int_0^1 \log \frac{P(x)}{Q(x)} dx \\ &= \int_0^1 \log \frac{1}{\frac{1}{4}} dx \\ &= \int_0^1 \log 4 dx \\ &= \log 4 \neq \infty. \end{aligned}$$

There are many examples that would work here. What would not work are proper Gaussian distributions since those are not zero anywhere.

(b) For a fixed target distribution P, we call $D_{\mathrm{KL}}(P||Q)$ the *forward-KL*, while calling $D_{\mathrm{KL}}(Q||P)$ the *reverse-KL*. Due to the asymmetric nature of KL, distributions Q that minimize $D_{\mathrm{KL}}(P||Q)$ can be different from those minimizing $D_{\mathrm{KL}}(Q||P)$.

From the following plots, **identify which of (A, B) correspond to minimizing forward vs. reverse KL. Give brief reasoning.** Here, only the mean and standard deviation of Q is allowed to vary during the minimization.

Solution: The easiest key to understanding this is to look at example B. In this, Q is extremely tiny in some place where P is big – this would make the forward-KL $D_{\mathrm{KL}}(P||Q)$ enormous. Meanwhile,



everwhere that Q is big, we see that P is also reasonably large in B. So the reverse-KL $D_{\mathrm{KL}}(Q||P)$ is not enormous. This tells us that B corresponds to minimizing reverse-KL.

Meanwhile, looking at A, we see that this corresponds to minimizing forward-KL $D_{\mathrm{KL}}(P||Q)$ because the Q is strongly trying to avoid being small where P is large and trying to strike an "average" type balance between the two modes of P. By doing so, it is putting a higher probability in a location where P is smaller — this would be costly in reverse-KL but is not that expensive in forward-KL.

3. Continual Learning (Optional)

Run this notebook and answer the questions. 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.

4. 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):

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

In this notebook, 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 $\mathbf{x} \sim \mathcal{N}(m, diag(v))$

Then, 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.**

Answer these questions:

(a) Test your implementation by training VAE with

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

Once the run is complete (10000 iterations), **report the following numbers** : 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

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

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

5. Generative Adversarial Networks (Optional)

(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

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

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

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

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

Solution: Consider the learning objective

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

For 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

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

$$= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{p_{data}(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}\cdot 2}\right] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\log \frac{p_\theta(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}\cdot 2}\right]$$
$$= -\log 4 + \left[\log \frac{p_{data}(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}}\right] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\log \frac{p_\theta(x)}{\frac{p_\theta(x)+p_{data}(x)}{2}}\right]$$
$$= -\log 4 + 2D_{JSD}(p_\theta||p_{data})$$

□

(c) **Training GANs on MNIST**

To mitigate vanishing gradients during training, **?** propose the non-saturating loss

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

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

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

for batch-size m, and batches of *real-data* $\mathbf{x}^{(i)} \sim p_{\text{data}}(\mathbf{x})$ 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

6. 0^{th} Order Optimization - Policy Gradient

We will now talk about 0^{th} order optimization, also known as Policy Gradient in a Reinforcement Learning context. Although this method is primarily used in an RL context we will be adapting this method to do 0^{th} order optimization on a Neural Network.

k^{th} order optimization means that in the optimization, we use a k^{th} order derivative ($\frac{\delta L^k}{\delta^k w}$) to do the optimization. So we can see that gradient descent is a first order optimization method, while Newton's method is a second order optimization method.

Polciy gradient is a 0^{th} order optimization method - which means that you use no derivative for the optimzation. This is used in contexts in which the loss is a **blackboxed** function, hence propogating a gradient through it is impossible.

Policy gradient at a high level approximates the gradient and then does gradient descent using this approximated gradient.

(a) **Prove** that

$$p_\theta(x)\nabla_\theta \log\big(p_\theta(x)\big) = \nabla_\theta p_\theta(x)$$

Solution: Using chain rule, we can see that the following is true

$$\begin{aligned} p_\theta(x)\nabla_\theta \log\big(p_\theta(x)\big) &= p_\theta(x)\frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} \\ &= \nabla_\theta p_\theta(x) \end{aligned}$$

(b) Let's say we have a neural network $f(x)$ which takes in a x and uses the weights(w) to output 2 logits ($P = [P(y = 0), P(y = 1)]$).
Let $p(x, y)$ be the joint distribution of the input and output data according to **our model**. Hence $p_w(x, y) = p(x)p_w(y|x)$, where p(x) is the ground distribution of x, while $p_w(y|x) = f(x)[y]$ is what our model predicts.

Similarly we have a **blackboxed** loss function $L(x, f(x))$ which outputs a loss. For example if I wanted to learn to classify $y = 1$ if $x > 5$ and $y = 0$ otherwise, $L(4, (0.1, 0.9))$ would be small while $L(4, (0.9, 0.1))$ would be very high. As we already discussed, since this loss is blackboxed we can't take the derivative through it.

We want to optimize the following objective function

$$w^* = argmin_w J(w)$$

where

$$J(w) = E_{(x,f(x))\sim p_w(x,y)}[L(x, f(x))].$$

To do this optimization we want to approximate $\nabla_w J(w)$ so that we could use an optimization method like gradient descent to find w^*

Prove that $\nabla_w J(w)$ can be approximated as $\frac{1}{N}\sum_{i=1}^{i=N}(\nabla_w \log\big(p_w(y_i|x_i)\big)L(x_i, f(x_i))$

Hints:

- Try creating a $\tau = (x, f(x))$
- $E[X] = \int_x xP(X = x)dx$
- Use the result from part a which was $p_\theta(x)\nabla_\theta \log\big(p_\theta(x)\big) = \nabla_\theta p_\theta(x)$
- $p_w(x, y) = p(x)p_w(y|x)$

Solution:

$$w^* = argmin_w E_{(x,f(x))\sim p_w(x,y)}[L(x, f(x))]$$

We call

$$J(w) = E_{(x,f(x))\sim p_w(x,y)}[L(x, f(x))]$$

and

$$\tau = (x, f(x))$$

$$\begin{aligned} J(w) &= E_{\tau \sim p_w(\tau)}[L(\tau)] \\ &= \int_\tau p_w(\tau) L(\tau) d\tau \end{aligned}$$

$$\begin{aligned} \nabla_w J(w) &= \int_\tau (\nabla_w p_w(\tau)) L(\tau) d\tau \\ &= \int_\tau (\nabla_w \log\big(p_w(\tau)\big)) p_w(\tau) L(\tau) d\tau \\ &= E_{\tau \sim p_w(\tau)}[(\nabla_w \log\big(p_w(\tau)\big) L(\tau)] \end{aligned}$$

We know that

$$\nabla_w \log\big(p_w(\tau)\big) = \nabla_w [\log\big(p(x)\big) + \log\big(p_w(y|x)\big)]$$

Since $p(x)$ does not depend on w we can simplify this to

$$\nabla_w \log\big(p_w(\tau)\big) = \nabla_w \log\big(p_w(y|x)\big)$$

Hence

$$\begin{aligned} \nabla_w J(w) &= E_{\tau \sim p_w(\tau)}[(\nabla_w \log\big(p_w(\tau)\big) L(\tau)] \\ &= E_{\tau \sim p_w(\tau)}[(\nabla_w \log\big(p_w(y|x)\big) L(x, f(x))] \end{aligned}$$

which can be approximated as

$$\frac{1}{N} \sum_{i=1}^{i=N} (\nabla_w \log\big(p_w(y_i|x_i)\big) L(x_i, f(x_i))$$

(c) The following two parts are based on this notebook, where you need to implement policy gradient according to your derivation before answering these questions.

Include the screenshot of the accuracy plot here. With a correct implementation, you should observe a test accuracy of approximately 75% after the final iteration.

Solution: See the solution notebook.

(d) **Compare the policy gradient and supervised learning approaches for this classification task, focusing on their convergence speed, stability, and final performance. Explain any observed differences.**

Solution: In the given classification task, the policy gradient method converges significantly slower compared to supervised learning.

Additionally, the policy gradient approach exhibits less stability during training.

Finally, the overall performance of the policy gradient method is generally worse than that of supervised learning.

Supervised learning is more direct in its approach, optimizing the loss function to increase its predicted probability of true labels, while policy gradients optimize through its own prediction with discrete rewards, which might not be as informative.

7. 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.**

Contributors:

- Linyuan Gong.
- Kumar Krishna Agrawal.
- Suhong Moon.
- Dhruv Shah.
- Anant Sahai.
- Aditya Grover.
- Stefano Ermon.
- Yashish Mohnot.

