HW11

Mengying Lin SID: 3038737132.

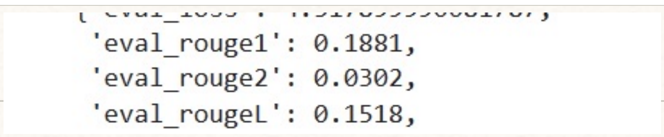
1. Coding Question: Summarization (Part II)

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?**

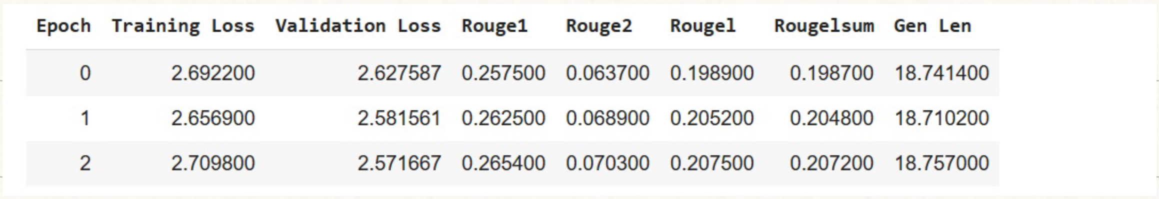


```
 'eval_rouge1': 0.1881,
 'eval_rouge2': 0.0302,
 'eval_rougeL': 0.1518,
```

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

Adafactor. 0.00005.

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?**

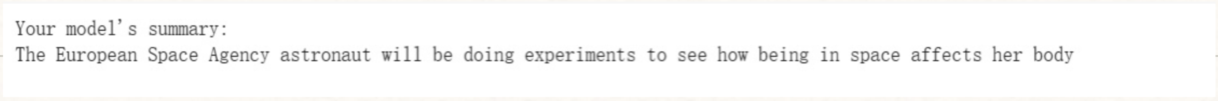


Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
0	2.692200	2.627587	0.257500	0.063700	0.198900	0.198700	18.741400
1	2.656900	2.581561	0.262500	0.068900	0.205200	0.204800	18.710200
2	2.709800	2.571667	0.265400	0.070300	0.207500	0.207200	18.757000

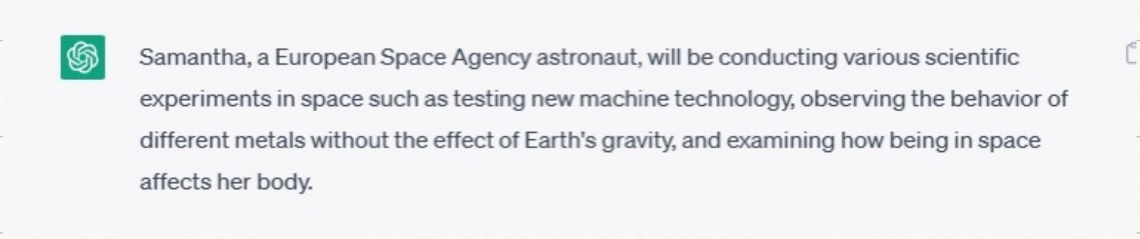
The scores are higher.

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.**

I used chatGPT, and before feeding in the document, I specify " Summarize the article in one sentence.



```
Your model's summary:
The European Space Agency astronaut will be doing experiments to see how being in space affects her body
```



Samantha, a European Space Agency astronaut, will be conducting various scientific experiments in space such as testing new machine technology, observing the behavior of different metals without the effect of Earth's gravity, and examining how being in space affects her body.

chatGPT manages to keep more critial info (e.g the name of the Astronaut) and uses a way more complex sentence structure, because it has higher expressive power and is able to capture more complex context and structure its expression in a more organized way.

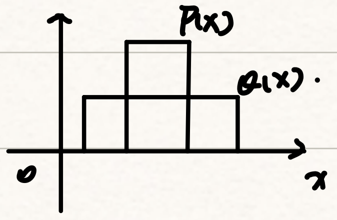
2. Comparing Distributions

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

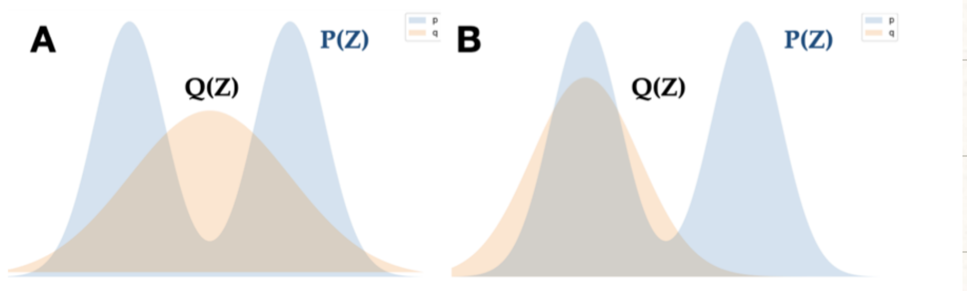
$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P(x)}\left[\log\frac{P(x)}{Q(x)}\right]$$

(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 $D_{\mathrm{KL}}(P||Q) \neq \infty$, $D_{\mathrm{KL}}(Q||P) = \infty$.**

a)



(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.



$D_{KL}(P||Q) = \int p(x) \log \frac{P(x)}{Q(x)} dx. = \int p(x) \log P(x) dx + \int p(x) \log \frac{1}{Q(x)} dx$

To minimize it, we need to minimize $\int P(x) \log \frac{1}{Q(x)} dx$

(assuming P(x) is fixed).

When P(x) is large, we hope $\log \frac{1}{Q(x)}$ is as insignificant as possible, which means Q(x) is large. Hence B is forward KL while A is the backward one

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.

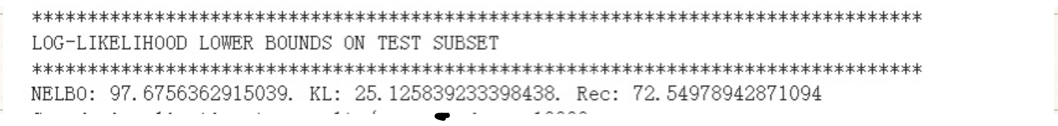
(a) Test your implementation by training VAE with

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

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

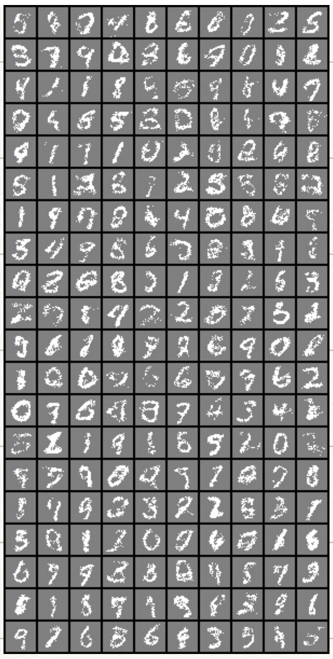
- negative ELBO
- KL-Divergence term
- reconstruction loss

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* $\sim$ 100*)*



```
********************************************************************************
LOG-LIKELIHOOD LOWER BOUNDS ON TEST SUBSET
********************************************************************************
NELBO: 97.6756362915039. KL: 25.125839233398438. Rec: 72.54978942871094
```

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



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

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

a) $p_\theta(x) \frac{1}{p_\theta(x)} \nabla_\theta p_\theta(x) = \nabla_\theta p_\theta(x).$

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

Hints:

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

$$J_w(w) = \sum_i^N p_w(x_i, y_i) L(x_i, f(x_i))$$

$$= \sum_i p_w(y_i|x_i)\, p(x_i)\, L(x_i, f(x_i))$$

$$\nabla_w J(w) = \sum_i p(x_i)\, \nabla_w p_w(y_i|x_i)\, L(x_i, f(x_i))$$

$$= \sum_i p(x_i)\, p_w(y_i|x_i)\, \nabla_w \log p_w(y_i|x_i)\, L(x_i, f(x_i))$$

$$= \sum_i p_w(x_i, y_i)\, \nabla_w \log p_w(y_i|x_i)\, L(x_i, f(x_i))$$

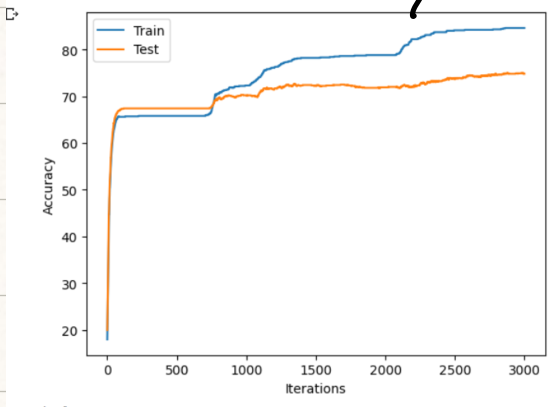
$$= \frac{1}{N} \sum_i \nabla_w \log p_w(y_i|x_i)\, L(x_i, f(x_i)).$$

$p_w(x, y) = p(x)p_w(y|x)$

(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.

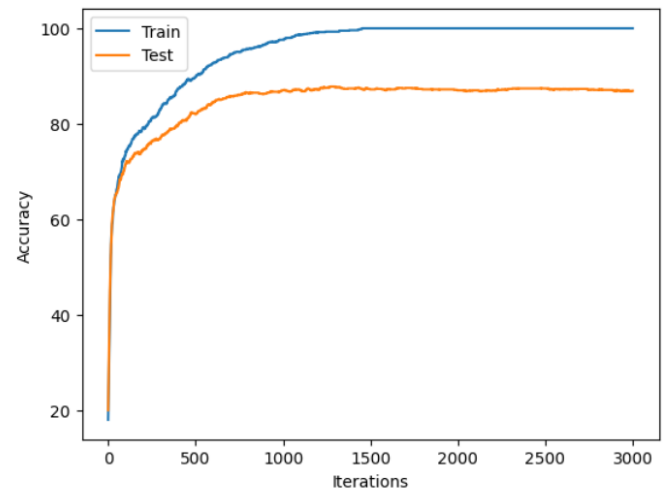
Include the screenshot of the accuracy plot here. With a correct implementation, you should observe an accuracy of approximately ~~90%~~ after the final iteration.

75%..



```
Final test accuracy: 74.8
```

(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.**



← supervised.

The former converges in a faster and more stable manner and ends up with higher acc

Policy gradient converges slowlier probably due to the undifferentiable reward fn in loss, which also accounts for the unstability. It performs worse probably because its reward function is not well designed.