EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Homework 12

This homework is due on Wednesday, May 3, 2023, at 10:59PM.

1. Denoising diffusion models for generation

In lecture, we talked about denoising diffusion models to get samples from a continuous distribution. This problem is about the potentially simpler binary case. We will assume that we have an unknown distribution of black-and-white images $P(\mathbf{x})$ together with a very large number of example images $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. Formally, each image can be viewed as a binary vector of length m, i.e. $\mathbf{x} \in \{-1, +1\}^m$.

The first conceptual step in setting up a diffusion model is to choose the easy-to-sample distribution that we want to have at the end of the forward diffusion. For this, we choose m iid fair coin tosses (here we think of a fair coin as having a 50% chance of being +1 and a 50% chance of being -1) arranged into a vector.

Next, we need to choose a way to incrementally degrade the images. Let $\mathbf{Y}_0$ start with whatever image sample $\mathbf{x}$ we want to start with. At diffusion stage t, we generate $\mathbf{Y}_t$ from $\mathbf{Y}_{t-1}$ by randomly flipping each pixel of $\mathbf{Y}_{t-1}$ independently with probability δ where δ is a small positive number.

It turns out that this process of rare pixel-flipping can be reinterpreted for easier analysis. For the j-th pixel at diffusion stage t, this process can alternatively be viewed as first flipping an independent coin $R_t[j]$ with a probability 2δ of coming up $+1$ and then, if $R_t[j] = +1$ replacing $Y_{t-1}[j]$ with a freshly drawn independent fair coin $F_t[j]$ that is equally likely to be -1 or $+1$. If $R_t[j] \neq +1$, we leave that pixel alone i.e. $Y_t[j] = Y_{t-1}[j]$.

(a) We need to verify that if we do this and diffuse for sufficiently many stages T, that the resulting distribution is close to looking like m i.i.d. fair coins. **Show that the probability that pixel j has been replaced at some point by an independent fair coin by time T goes to 1 as $T \to \infty$.**

(HINT: It might be helpful to look at the probability that this has not happened...)

Solution: After T stages, consider the probability that the j-th pixel has not been replaced. This requires the T i.i.d. coin tosses to all have come up tails (or -1 if you prefer) and by independence, the probability of that happening is $(1-2\delta)^T$. Since $\delta > 0$, we know that $|1-2\delta| < 1$ and by stability, this implies that $\lim_{T\to\infty}(1-2\delta)^T = 0$. Since the probability of the complementary event is going to zero, we know that the probability that the pixel *has* been replaced by time T is going to 1.

(b) To efficiently do diffusion training, we need a way to be able to quickly sample a realization of $\mathbf{Y}_t$ starting from $\mathbf{Y}_0 = \mathbf{x}_i$. **Give a procedure to sample a realization of $\mathbf{Y}_t$ given $\mathbf{Y}_0$ without having to generate $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_{t-1}$.**

(HINT: This should involve flipping at most two (potentially biased) coins for each pixel.)

Solution: We know from the previous part that that the probability that j-th pixel has not been replaced by a fair coin by time t is $(1-2\delta)^t$. Consequently, we flip a coin with a probability $(1-2\delta)^t$ of coming up tails. (Note: this can be done by sampling a uniform random variable on $[0,1]$ and seeing if it is greater than $(1-2\delta)^t$ and calling that a head.) If it comes up heads, replace the j-th pixel of $\mathbf{Y}_0$ with a freshly tossed independent fair coin (-1 or +1 equally likely) for $y_t[j]$. Otherwise, have $y_t[j] = y_0[j]$.

(c) For the reverse diffusion process that will be used during image generation and is being learned during training, our goal is to approximate $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$ with a neural net that has learnable parameters θ. Suppose I give you a neural net whose input is a binary image $\mathbf{Y}_t$ and whose output is m real numbers that could each in principle be from $-\infty$ to $+\infty$ (for example, these could be the outputs of a linear layer). **Which of the following nonlinear activation functions would be most appropriate to convert them into a probability that we could use to sample whether the pixel in question should be a $+1$?**

◯ Sigmoid $\frac{1}{1+\exp(-x)}$
Solution: The sigmoid is the right answer since since it always returns a number between 0 to 1 which is required for a valid probability.

◯ ReLU $\max(0, x)$
Solution: This is the wrong answer since it can be greater than 1.

◯ Tanh $\tanh(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$
Solution: This is the wrong answer since it can be negative.

(d) The goal of training is to approximate a probability distribution for random denoising. However, we do not actually have access to $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$ and decide to use $P(\mathbf{Y}_{t-1}|\mathbf{Y}_t, \mathbf{Y}_0)$ instead as a proxy.
What is $P(Y_{t-1}[j] = +1|\mathbf{Y}_t = \mathbf{y}, \mathbf{Y}_0 = \mathbf{x})$?
For simplicity, just do this calculation for the case $x[j] = +1$. To further help you save some time, you may use the following helper result that comes from Bayes' Rule. If A and B are both binary random variables where the prior probabilities are $P(A = +1) = \rho$ and $P(A = -1) = 1 - \rho$, with B being bit-flipped from A with independent probability δ — that is, $P(B = +1|A = +1) = 1 - \delta$, $P(B = -1|A = +1) = \delta$, $P(B = +1|A = -1) = \delta$, and $P(B = -1|A = -1) = 1 - \delta$ — then the conditional probabilities for A conditioned on B are given by:

$$P(A = +1|B = +1) = \frac{(1-\delta)\rho}{(1-\rho)\delta + (1-\delta)\rho} \tag{1}$$

$$P(A = -1|B = +1) = \frac{(1-\rho)\delta}{(1-\rho)\delta + (1-\delta)\rho} \tag{2}$$

$$P(A = +1|B = -1) = \frac{\delta\rho}{\rho\delta + (1-\delta)(1-\rho)} \tag{3}$$

$$P(A = -1|B = -1) = \frac{(1-\delta)(1-\rho)}{\rho\delta + (1-\delta)(1-\rho)} \tag{4}$$

(HINT: What is the distribution for $Y_{t-1}[j]$ given $Y_0[j]$?)

Solution: Following the strategy urged by the hint, we let A be $Y_{t-1}[j]$ and B be $Y_t[j]$ so we can use the formulas above. Since we are interested in the conditional probability of $Y_{t-1}[j] = +1$, the relevant formulas we want to invoke are (1) and (3).

Conditioning on $\mathbf{Y}_0 = \mathbf{x}$ just tells us that $x[j] = 1$ and by the underlying Markovianity of the evolution from $\mathbf{Y}_{t-1} \to \mathbf{Y}_t$, we know that $P(Y_t[j]|\mathbf{Y}_{t-1}, \mathbf{Y}_0) = P(Y_t[j]|\mathbf{Y}_{t-1}) = P(Y_t[j]|Y_{t-1}[j])$. This means that we are exactly in the setup for the calculation given above. This means that all we need is the $\rho = P(Y_{t-1}[j] = +1|Y_0[j] = x[j] = +1)$.

Since ρ is the probability of a 1, there are two disjoint ways we could have a 1. The original 1 could have survived or the final fair coin toss that replaced it was itself a 1.

$$\rho = (1-2\delta)^{t-1} + (1-(1-2\delta)^{t-1})\frac{1}{2} \tag{5}$$

$$= \frac{1 + (1-2\delta)^{t-1}}{2} \tag{6}$$

and plugging this into (1) and (3) gives us:

$$P(Y_{t-1}[j] = +1 | \mathbf{Y}_t = \mathbf{y}, \mathbf{Y}_0 = \mathbf{x}) = \begin{cases} \frac{(1-\delta)\rho}{(1-\rho)\delta + (1-\delta)\rho} & \text{if } y[j] = +1 \\ \frac{\delta\rho}{\rho\delta + (1-\delta)(1-\rho)} & \text{if } y[j] = -1 \end{cases}.$$

Note that if we had $x[j] = -1$, what would happen is that ρ would be $\frac{1-(1-2\delta)^{t-1}}{2}$.

(e) Let the (conditional) probability distribution (on whether each pixel is $+1$) output by our neural net with nonlinearity be $Q_t(\mathbf{Y}_t)$. For training the denoising diffusion model $q(\mathbf{Y}_{t-1}|\mathbf{Y}_t)$, we choose to use SGD loss $D_{KL}(P(\mathbf{Y}_{t-1}|\mathbf{Y}_t, \mathbf{Y}_0 = \mathbf{x}_i)||Q_t(\mathbf{Y}_t))$ where $\mathbf{x}_i$ is the random training image drawn, t is the random time drawn from 1 to T, and $\mathbf{Y}_t$ is the randomly sampled realization of the forward diffusion at time t starting with the image $\mathbf{x}_i$ at time 0. This ends up being a loss on the vector of probabilities coming out of $Q_t(\mathbf{Y}_t)$ that can be written as a sum over the m entries in the vector of probabilities.

Given what you know about KL Divergence, what does this loss penalize most strongly? What does this loss look like at $t = 1$ in particular?

Solution: The KL divergence penalizes most strongly the probability that the neural net says that something is very unlikely when the proxy target distribution has significant probability. This is due to the Q (from the neural net) being the second term in the KL divergence.

It is informative to look at $t = 1$ as a starting point. For $t = 1$, the target distribution we are using is $P(\mathbf{Y}_{1-1}|\mathbf{Y}_1, \mathbf{Y}_0 = \mathbf{x}_i) = P(\mathbf{Y}_0|\mathbf{Y}_1, \mathbf{Y}_0 = \mathbf{x}_i) = P(\mathbf{Y}_0|\mathbf{Y}_0 = \mathbf{x}_i)$ which is always going to be putting all of its probability on $\mathbf{x}_i$. Using the formula for KL divergence, we see that there is only a single term in the expectation and so:

$$D_{KL}(P(\mathbf{Y}_0|\mathbf{Y}_0 = \mathbf{x}_i)||Q_1(\mathbf{Y}_1)) = \log \frac{1}{Q_1(\mathbf{x}_i|\mathbf{Y}_1)} \tag{7}$$

$$= \sum_{j=1}^{m} -\log Q_1(\mathbf{x}_i[j]|\mathbf{Y}_1) \tag{8}$$

which is identical to binary cross-entropy loss for denoising the original binary image, treating every pixel as a binary classification problem.

This insight allows us to, in the binary case, reinterpret the losses at every stage of the denoising diffusion model. After all, in the previous problem we can see that the target distribution for the KL loss is approaching a fair coin toss as $t \to \infty$. This can be interpreted as binary cross-entropy loss for reconstructing the original binary image, except using a label-smoothing schedule where the target labels are increasingly smoothed as t grows.

Under this interpretation, every stage of the reverse diffusion is very much an image denoiser. However, as we increase the level of the noise, we increasingly smooth the labels for the cross-entropy loss function.

This interpretation should help you understand more deeply why substantial weight-sharing across different t makes so much sense — they are all reflecting the same basic pattern which is being learned.

2. Diffusion Models

In the previous question we considered sampling from a discrete distribution. Let's now see how iteratively adding Gaussian noise to a data point leads to a noisy sequence, and how the reverse process refines noise to generate realistic samples.

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 $\mathbf{x}_1, ..\mathbf{x}_T$.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I) \qquad q(\mathbf{x}_{1:T}|x_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{9}$$

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. (9), **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

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

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. (9) , 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

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

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

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

$$\mu(x_t, x_0) = \left(\frac{\sqrt{1-\beta_t}}{\beta_t}x_t + \frac{\sqrt{\alpha_t}}{1-\alpha_t}x_0\right) \Big/ \left(\frac{1-\beta_t}{\beta_t} + \frac{1}{1-\alpha_t}\right)$$
$$= \frac{\sqrt{1-\beta_t}(1-\alpha_t)}{1-\alpha_t}x_t + \frac{\beta_t\sqrt{\alpha_{t-1}}}{1-\alpha_t}x_0$$

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

3. TinyML - Early Exit

As models get deeper and deeper, we spend a lot of compute on inference, passing each batch through the entirety of a deep model.

Early exit comes from the idea that the computationally difficulty of making predictions on some inputs is easier than others. In turn, these "easier" inputs won't need to be processed through the entire model before a prediction can be made with reasonable confidence. These easier examples will exit early, and examples that are more difficult/have more variability in structure will need to be processed through more layers before making a reasonably confident prediction.

In short, we offer samples the option to be classified early, thus saving on the extra compute that would've been exhausted if full inference had been executed.

Early exit serves to save compute, decrease inference latency, all while maintaining a sufficient standard of accuracy.

(a) We consider a toy model of early exit, a series of cascading probability distributions.

We sample from each distribution in a sequence and add the result to a partial sum of all previously sampled values. The ith distribution is sampled from $N(0, \frac{1}{2^{i-1}})$. Denote this as X_i. All X_i are independent. Denote $Y_k = \sum_{i=1}^{k} X_i$ and $Y = \sum_{i=1}^{\infty} X_i$. **Leave your answers in terms of Φ, the CDF of the normal.**

i. **Calculate $\mathbf{P(Y \leq 0 | Y_k = M)}$**

That is, if the value of after summing up the first k samples of our partial sum is M, what is the probability that our final sum will be less than 0. The kth partial sum can be seen as the value of the feature map at the kth layer in the neural network. Each sequential layer provides less new information, as each sequential distribution has a smaller variance.

Solution: Variance of the remaining distribution is the sum of the variances of the remaining distributions. Since all the distributions have zero mean, we don't need to consider the mean.
$\sum_{i=1}^{\infty} Var(X_i) = \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = \frac{1}{1-\frac{1}{2}} = 2$

$Var(Y - Y_k) = Var(Y) - Var(Y_k) = 2 - \sum_{i=1}^{k} \frac{1}{2^{i-1}} = 2 - \frac{(1-\frac{1}{2}^k)}{\frac{1}{2}} = 2 - 2(1 - \frac{1}{2}^k)$

Call this distribution Z_k.

$P(Z_k < -M) = P(Z < \frac{-M}{\sqrt{2-2(1-\frac{1}{2}^k)}}) = \Phi(\frac{-M}{\sqrt{2-2(1-\frac{1}{2}^k)}})$

ii. **Calculate $\mathbf{P(Y \leq 0 | Y_1 = 5)}$. Speculate why if we have only sampled the first distribution, but got a 5, we are pretty sure that the final value will not be less than 0.**

Solution: $Var(Y - Y_1) = Var(Y) - Var(Y_1) = 2 - 2(1 - (1 - (\frac{1}{2})^1)) = 1$

$P(Y < 0|Y_1 = 5) = P(Y - Y_1 < -5) = P(Z_1 < -5) = P(Z < \frac{-5}{\sqrt{Var(Y-Y_1)}}) = P(Z < -5) = \Phi(-5) = 2.9e-7$

We are very sure because we have a lot of information. We have sampled 20 times and the variances are getting exponentially smaller. Thus even if we are close to 0, it doesn't matter because the variance of the remaining sum is tiny.

iii. **Calculate $\mathbf{P(Y \leq 0|Y_{40} = 0.00001)}$. Speculate why even if we are so close to 0, after $\mathbf{k = 40}$ we are very sure that the final value will not be less than 0.**

Solution:
$Var(Y - Y_{40}) = Var(Y) - Var(Y_{40}) = 2 - 2(1 - (1 - (\frac{1}{2})^{40})) = 1.8189894e-12$

$P(Y < 0|Y_{40} = 0.00001) = P(Y - Y_{40} < -0.00001) = P(Z_{40} < -0.00001) = P(Z < \frac{-0.00001}{\sqrt{Var(Y-Y_{40})}}) = P(Z < -7.4146023178) = \Phi(-7.4146023178) < 10^{-10}$

If we sample 5 on the first distribution, it is unlikely the rest of the values will end up pulling that 5 down to a 0. The reason is because 5 quite large, and since the variances are getting smaller, this signal will likely not be overriden by samples from further distributions.

iv. **How does this relate to early exit?**

Solution:
Thus, given a certain value obtained after a certain number of samples, we have little reason to continue sampling. In neural networks, after achieving a sufficient confidence before propagating to the end of the network, there is little reason to waste compute sending the information through the whole network.

(b) Please complete `hw12_early_exit.ipynb` notebook on early exit then answer the following questions. **Training should take less than 10 minutes per model. Please select a GPU runtime on colab. You should work on the analytic portion while training.**

i. How does the baseline ResNet perform on the validation set? **Solution:** 55-65 percent

ii. What is the validation accuracy of the early exit model? **Solution:** 46 - 56 percent

iii. How often is the model exiting early? How confident is it when it exits early? How Confident is the model when it passes through the entire model?

Solution: Exiting Early: 60-80 percent
Early Exit Confidence: 35-45 percent
No Early Exit: 50-60 percent

iv. What is the MAC Ratio between the baseline model and the early exit model? Can you find a threshold that has a spike in change?

Solution: 50-60 percent

v. What is the validation accuracy of the smaller resnet model?

Solution: 45-55 percent

vi. Find the minimum threshold where the early exit accuracy is better than the small net accuracy. Please report your findings of hyperparameter search. **Solution:** Students values may vary. Generally they should be between 39 and 41 percent. The MACs should decrease about 20-30 percent

4. Reinforcement Learning from Human Feedback

As the next chapter of our "Transformer for Summarization" series, we will delve into the application of reinforcement learning from human feedback (RLHF) for natural language processing tasks, as introduced in the InstructGPT paper (https://arxiv.org/pdf/2203.02155.pdf). Building on the foundations laid in our previous assignments, we will implement the RLHF algorithm to tackle the news summarization task.

First, we'll explore the application of policy gradients for training sequence generation models. In every generation step of a sequence generator, it produces a probability distribution for the next token, given the prior tokens (and the source sequence, if it's a sequence-to-sequence model):

$$\mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1})$$

Considering the sequence generation model as an RL agent's policy network, we can represent it as follows:

State Prior tokens $y_1,\ldots,y_{i-1}$, along with the source sentence $\mathbf{x}$ in a sequence-to-sequence context.

Action Generating the next token y_i.

Action space The entire token vocabulary.

Transition By producing token y_i, the state transitions from $y_1,\ldots,y_{i-1}$ to $y_1,\ldots,y_i$.

Agent The policy network $\mathbf{P}_\theta$, which outputs a probability distribution over the vocabulary (action space) at each step.

Upon generating a sequence $\mathbf{y}=[y_1,\ldots,y_n]$, it is evaluated (we will discuss evaluation methods later) to obtain a **reward** value $r(\mathbf{y})$ (or $r(\mathbf{x},\mathbf{y})$ in sequence-to-sequence generation).

(a) **Prove that the policy gradients $\nabla_\theta\mathcal{L}(\theta)$ for the sequence-to-sequence generation task are given by:**

$$-\mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\mathbf{P}_\theta(\mathbf{x})}\left(r(\mathbf{x},\mathbf{y})\sum_{i=1}^{n}\nabla_\theta\log\mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1},\mathbf{x})\right)$$

Solution: Use the formula for policy gradients, we have

$$\nabla_\theta\mathcal{L}(\theta)=-\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left(\left(\sum_{i=1}^{n}\nabla_\theta\log\mathbf{P}_\theta(a_i|s_i)\right)\left(\sum_{i=1}^{n}\text{Reward}(s_i,a_i)\right)\right) \tag{10}$$

Given the definition of states and actions in this problem, we have $s_i=(\mathbf{x},[y_1,\ldots,y_{i-1}])$ and $a_i=y_i$

$$\nabla_\theta\mathcal{L}(\theta)=-\mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\mathbf{P}_\theta(\mathbf{x})}\left(\left(\sum_{i=1}^{n}\nabla_\theta\log\mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1},\mathbf{x})\right)\left(\sum_{i=1}^{n}\text{Reward}((\mathbf{x},[y_1,\ldots,y_{i-1}]),y_i)\right)\right) \tag{11}$$

Because the reward is given only after the last step, we have

$$\text{Reward}((\mathbf{x},[y_1,\ldots,y_{i-1}]),y_i)=\begin{cases}r(\mathbf{x},\mathbf{y}),\ \text{If } i=n\\ 0.\ \text{Otherwise}\end{cases} \tag{12}$$

So we have:

$$\nabla_\theta \mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\mathbf{P}_\theta(\mathbf{x})}\left(r(\mathbf{x},\mathbf{y})\sum_{i=1}^{n}\nabla_\theta \log \mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1},\mathbf{x})\right) \tag{13}$$

(b) In the last part, we established that the loss function

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\mathbf{P}_\theta(\mathbf{x})}\left(r(\mathbf{x},\mathbf{y})\sum_{i=1}^{n}\log \mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1},\mathbf{x})\right)$$

can be differentiated to obtain the policy gradients for sequence-to-sequence generation.
What is the relationship between the policy gradient loss and the cross-entropy loss in supervised sequence-to-sequence training?

Solution:
The cross-entropy loss in supervised sequence-to-sequence training is:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{\text{train}}}\sum_{i=1}^{n}\log \mathbf{P}_\theta(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) \tag{14}$$

The part inside the expectation is the same when we set $r(\mathbf{x},\mathbf{y}) = 1$, i.e., the reward is 1 for all input-output pairs in the training dataset.
The distinction lies in the subscript of the expectation. In supervised training, input-output pairs are ground truths sampled from a labeled training dataset. In reinforcement learning, the input is typically sampled from an unlabeled training set (since we don't have access to ground truth outputs when we have to apply reinforcement learning), and the output $\mathbf{y}$ is sampled from the model's output probabilities at each step.

The remainder of this assignment involves coding. Please follow the instructions in this notebook to learn how to implement RLHF for summarization. Once you have completed the coding portion, download `submission_log.json` and submit it to "Homework 12 (Code) (RLHF)" in Gradescope.

5. 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:

- Anant Sahai.
- Kumar Krishna Agrawal.
- Liam Tan.
- Linyuan Gong.

