Deep Learning on Computational Accelerators
236781 Mini-Project

Winter 23-24

- **Submission Due**: 26/05/24, 23:59
- **TA In Charge**: Tamir Shor

1 Wet Assignment

In this project you will experiment with the task of MRI compressed sensing.
In section 1.1 we supply some basic terminology needed for this assignment. In section 1.2 we give some information about what compressed sensing is. This is mainly for you to understand the task you'll be solving if you find this interesting. You don't need this background to do the project.
In section 1.3 we describe what you'll be doing. Section 1.5 elaborates the code we'll be supplying you with for this project. Section 1.6 specifies how we'll evaluate your work.

1.1 A bit of Basic Background

What we do in this assignment makes use of Fourier transforms. If you don't know what those are, don't worry. No signal processing background is required for this assignment - just think of it as a constant linear transformation transforming an $H \times W$ image to another representation of the image with shape $H \times W \times 2$. Just like the linear layers we learned about do (only pre-chosen and constant).
You will not need to worry about any of these transformations yourself - we will supply code that uses them and will refer to them when explaining what compressed sensing is and what the code we give you does. For this purpose we will define some related terminology - When we say "the image in frequency domain" we just mean the image after being been applied this transformation, giving back the $H \times W \times 2$ image. When we say "the image in image domain" we mean the original image after being applied the reverse transform (also a constant linear operation) over the frequency domain image, giving back the

$H \times W$ image. In this document we'll denote images represented in image domain by $x \in \mathcal{X}$ and frequency domain images by $\tilde{x} \in \tilde{\mathcal{X}}$

1.2 Compressed Sensing

MRI scans take a long item. This makes MR imaging more expensive, less pleasant for patients and sometimes causes imaging artifacts. One approach to reduce the time they take is compressed sensing - MR images (Images attained from MRI) are acquired in the frequency domain. This is basically a decomposition of the image to the different frequency components that it is made of. Unlike the image domain, if we remove some of the data in frequency domain, the changes as induced in image domain can often be imperceptible to the human eye, or at least easier to reconstruct (say, using a neural network).
This is essentially what we do in compressed sensing - first, the image is subsampled in the frequency domain (say, random values are zeroed-out), giving us some image. Then, the subsampled image is transferred to the image domain, and then fed into a neural network to reconstruct the original image.
This is useful because at inference we can only acquire the frequencies we didn't remove in subsampling (which could take much less time in an MRI scan), and then use our network to reconstruct the image (that is hopefully similar to the original image we'd get if we'd acquire the all data in frequency domain). You can see a visualization in the plot below 1.2.

Figure 1: Fully-acquired image

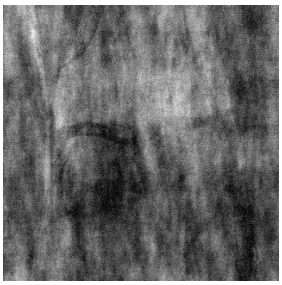

Figure 2: Subsampled Image

Figure 3: Image reconstructed from image 2

In this project we will experiment with random compressed sensing masks (i.e. pixels for removal from the frequency domain image are randomly selected). It's important to note (though we ignore this for the project) that this is not how real compressed sensing systems work - many other factors must be taken into account. For example, the sampling trajectories must be smooth and kinematically feasible for the imaging machine. If you're interested in how actual feasible systems work you may refer to some work from our lab. You can also contact us if you wish to do project courses related to compressed sensing (or

to other topics, for that matter).

1.3 Assignment

Your assignment in this project will be to design and train a neural network to perform the reconstruction of subsampled images. We will supply you with a codebase (detailed in 1.5) that handles data loading. Each data sample you receive from our dataloaders will be a batch of MR images in frequency domain $\tilde{x}$. You will implement a model which takes these image batches, subsamples them according to a random binary mask $\mathcal{M}$ (with code we give you) to receive the subsampled version $\mathcal{M}(\tilde{x}) = \tilde{x}_{subsampled}$, converts the data to $x_{subsampled}$ (in image domain) and performs reconstruction using a model m_θ to receive $m_\theta(x_{subsampled}) = x_{reconst}$. The fully-acquired target image x is also provided by the dataloaders. A naive, vanilla reconstruction model would be provided as reference. Your goal is to make $m_\theta(\mathcal{M}(\tilde{x})$ resemble x as much as possible.

You will train and evaluate your algorithm for various level of image subsampling (i.e. amount of data subsampled) in two scenarios - under a constant random subsampling mask $\mathcal{M}$, and for a model that jointly optimizes your reconstruction network m_θ and a parametric subsampling mask $\mathcal{M}_\psi$ (ψ parameterizes which frequencies to subsample). For the latter case, our code handles mask learning, however training stability and performance depend on your reconstruction model and optimization parameters. Gradients will flow from your model to the mask optimization.

1.4 Data

We'll be using the fastMRI dataset of 1500 fully-sampled knee MRI scans, published be Meta and NYU. You can read about it here. The dataset is mounted on course servers.

1.5 Codebase

The codebase we provide consists of the following files:

- **main.py** - This is the main file to run training. You can pass the *learn-mask* argument to enable subsampling mask optimization and the *drop-rate* argument to control extent of subsampling (refer to the argparsing function for full argument documentation). We supply train, validation and test sets. You must use them appropriately according to their respective roles.

- **models directory** - You should implement your reconstruction model in a python file located in this folder. The folder contains two files - **subsampling.py** implements the subsampling layer maintaining the subsampling mask. There's no reason to change this file. **vanilla.py** gives you a basic example of how to incorporate the subsampling layer in your model.

In your own model's file, you need to make sure to initialize this layer as done in the vanilla model's init function, and run your input through the subsampling layer in the forward function before doing anything else (follow documentation in vanilla.py). This will subsample your input and return the subsampled data in image domain.

- **data directory** - Here we implement the dataset class and some needed data transformations used in the main file. You do not need to change these files.
- **utils.py** - Located in the utils folder, here we implement some utilities for the main file. You may add your own utility functions there, however you shouldn't change existing ones.

Some important coding notes

- After calling your optimizer's step, you must use this line: *model.subsample.mask_grad(args.mask_lr)*. If you don't, your mask learning wouldn't work. The reason we do this is that in our setting we must manually apply the gradient over the mask - we are optimizing strictly binary masks, therefore torch's continuous optimization, as is, wouldn't be applicable. your optimizer only optimizes the reconstruction network.
- Our subsampling layer handles all image/frequency domain transitions. However, if you further want to convert frequency domain image to image domain (say, for visualization purposes), you can use the *freq_to_image* function, implemented in the utils.py file. View documentation if needed.

1.6 Evaluation

You will submit a report including the following components:

- An overview of your solution - Explain your solution - what reconstruction architectures did you consider? Which network did you eventually use? Why did you think it was a good choice? Detail your considerations. Any choice is okay as long as you reason about it. Also state which loss criteria you used, and whether you had a specific reason to use it.
- For each of the two scenarios mentioned in section 1.3 (learned and non-learned subsampling mask), train and evaluate your model on all drop rates between 0.1 and 0.7 in jumps of 0.1 (inclusive, meaning $0.1, 0.2, 0.3, ..., 0.7$). Report mean and std of PSNR (see here) across the train and test sets (separately). PSNR should be evaluated between target and output image for each of the total of 14 test cases - so overall your report should include 14 PSNR means and stds for the train set, and 14 means and stds for the test set. Make sure to explain what we see in results and what conclusions we can draw from them. Also, based on the comparison between train and test results, relate to whether your results indicate underfitting, overfitting, or none of the two.

- Your report should also include some visualization of results - for each of the 14 subsampling rates, plot any one pair of corresponding target (fully sampled) image and reconstructed output image **from the test set**. You are encouraged to include any other visualizations you find relevant.
- Explain your results - for which cases (drop rates, learned vs. non-learned masks) were results better and in which worse? Explain some reasons to why that could be. How are the model and optimization choices you made affected these results?
- Assuming you had a year to work on this project and unlimited hardware resources, what else would you have tried and what would you do differently?

You will be evaluated based on your algorithm's results. Nonetheless, the most important factor we will be grading this segment on is your ability to explain and justify your choices. **Your algorithm doesn't have to produce good results on all cases**. As a basic sanity check, however, at the very least you should receive coherent output images when dropping less than 30% of the data.

2 Dry Assignment

In this section you will answer general theoretical questions concerning various topics we've seen in the course. Be sure to provide full and elaborate answers to all questions.

1. You are given a classification model $y = \sigma(Wx + b)$. x is some input, W, b are learnable weight and bias parameters. Classification is positive if y is larger/equal to some scalar threshold τ, else negative.
   (a) Is this classification model linear w.r.t to the input? Prove your answer.
   (b) Assume your dataset is $X = \{(1,1),(1,-1),(-1,1),(-1,-1)\}$, where color represents the sample class. What would be your classifier's optimal classification accuracy over this dataset. Explain.
   (c) Now you expand your classifier to be $y = W_2 \cdot \sigma(W_1 x + b_1) + b_2$. W_1, W_2 are weight matrices and b_1, b_2 are bias vectors. Would your answer to the previous sections be any different? How come?
   (d) Generally speaking, in what cases would we prefer to use linear models over non-linear models? Give at least 2 specific scenarios.
2. Your friend is trying to optimize an MLP model to fit some labeled dataset. His optimizer is an instance of some optimizers class in *torch.optim*. He is using the following snippet:

   Explain what this code is currently doing, and how it is different from the correct optimization process we've learned. Find the error in the code

```
for (x,y) in dataloader:
    #x - sample, y - ground truth
    loss = loss_fn(model(x),y) #model is some mlp
    loss.backward()
    optimizer.step()
```

(assuming the loss function, model and data have no issues). How would you fix it?

3. (a) Explain the term "Receptive Field" in the context of CNNs.
   (b) Give one example of a scenario where we'd want to have large receptive fields, and one case we're we'd prefer to have smaller ones. Explain.
   (c) Offer and explain 3 different ways to control the size of the receptive field.

4. Your friend has a labeled dataset of images of cats, dogs and horses. She wants to train some classification model over this dataset.
   (a) She decides to use the MSE loss as her training criteria - for each classification her model outputs (a class number - 0, 1 or 2), she'd compare it to the ground truth (a scalar from the same value set) and compute the MSE.
   Would you say this is a good choice? Why or why not.
   (b) Your friend decided to take a different road and solve a semantic segmentation problem - now, her model would predict a single class (out of the 3) for each pixel. Assume the dataset is labeled accordingly (for each image we have a per-pixel label). Her model computes the output per-pixel prediction, and she offers to use the L1 loss (absolute error) to compute the loss used for optimization. Explain why this might not be a suitable loss criteria.
   (c) Specify a better criteria for section b.

5. (a) Explain the terms "vanishing gradients" and "exploding gradients".
   (b) Provide a numerical example demonstrating each.
   (c) For each of the following architectures - MLP, CNN, RNN, offer one **distinct** way to combat vanishing or exploding gradients (do not use the same technique for two different architectures). Briefly explain how your technique helps reducing these phenomena.

6. You are given an image x of shape $1 \times 32 \times 32$. You are using the following convolutional layer to predict a ground-truth label y, which is an image of the same size as x. Given the convolutional layer and loss in the snippet, derive an analytical expression for the gradients of the loss w.r.t each learnable parameter in the model.

Hint: You don't have to understand how to strictly differentiate convolutional layers. Think about what the layer actually does in this specific case.

```
model = torch.nn.Conv2d(1,1,32)
for (x,y) in dataloader:
    #x - 1X32X32 shape sample, y - 1X32X32 shape ground truth
    y_hat = model(x)
    loss = ((y_hat-y)**2).sum()
```

7. (a) Explain the back-propagation through time algorithm (BPTT). How does it work (briefly)? why is it needed in sequence models (rather than using the same backpropagation as we've seen in tutorial 5)?
   (b) What are the main issues with BPTT and how does Truncated BPTT (TBPTT) help combat them?
   (c) True or False - TBPTT of timespan S allows the model to learn relations between input elements that are at most S timesteps apart. Explain your answer.

8. Your friend wants to process some set of sequential data samples with an RNN model. She learned about positional embeddings in transformers and decided to apply them in RNNs as well.
   (a) Is it possible to add positional embeddings when using an RNN model? Explain.
   (b) If you think it is possible, explain why it would probably not be necessary in the case of RNNs. If you don't think it's possible, offer some adaptation to the RNN model we learned in class so that adding positional embeddings would be applicable.

9. You are given the input sentence "Attention's all you need". The sentence is tokenized into four input word embeddings $[[0.3, 0.25, 0.45], [0.75, 0.1, 0.15], [0.6, 0.2, 0.2], [0.05, 0.4, 0.55]]$. Assume We are using the query/key/value casting matrices.

$$W_Q = \begin{pmatrix} 0.1 & 0.3 & 0.6 \\ 0.4 & 0.05 & 0 \\ 0 & 0.7 & 0.5 \end{pmatrix}, W_K = \begin{pmatrix} 0 & 0.15 & 0.9 \\ 0.8 & 0.35 & 0.75 \\ 0.25 & 0.4 & 0 \end{pmatrix}, W_V = \begin{pmatrix} 0.7 & 0.15 & 0.1 \\ 0 & 0.1 & 0.4 \\ 0.2 & 0.4 & 0.95 \end{pmatrix}$$

   Manually calculate the query matrix Q, key matrix K and attention score matrix A. Use 2 decimal point precision. Show your steps.

10. (a) For the same input sentence from question 7 and same embeddings, what would be the positional encoding of the last token in the sequence? No need to show steps. Use the sinusoidal encoding we've learned in the tutorials.

(b) Why is positional encoding needed? Give a specific example where not using positional encoding might harm model performance.

(c) The use of transformers doesn't always require positional embeddings. Give an example of some task related to processing sequential data where we would not have to add positional embeddings, even when using a transformer.
Hint: Think about what the positional encoding does - are there any tasks where we don't need this affect?

(d) In the tutorials we've learned about sinusoidal positional encodings. One simpler option would be to encode each token's position by its absolute position. i.e. The encoding of the ith token in the sequence would be an appropriately-sized vector with values all equal to $i-1$. Why would that not be a good choice?

11. We are training a basic GAN architecture with binary cross-entropy criterion, as we've seen in tutorials. Assuming training goes well and the generator eventually manages to produce real-looking outputs - what should be the discriminator's final loss? Explain.

12. Your friend wants to create a generative model for his favorite dataset. He wants to use an encoder-decoder model, however he thinks our modeling from the VAE tutorial is unnecessarily complicated. Instead, he proposed to take each input image x, encode it with our encoder to a single latent vector z, and then decode the same latent vector z by the decoder to receive an output image $\hat{x}$. These $x, \hat{x}$ and z would be used to compute the ELBO like we defined in the VAE tutorial.

    - This proposal is inherently different from the VAE model we defined in the tutorial. How?
    - Do you think the new proposed method would work (i.e. will your friend manage to generate good images)? Why or why not?

3 Submission

Your submission must include a single zip file named $< id1 > - < id2 > .zip$, where id1,id2 are ids of you and your partner. This folder must include the following files:

- Your wet part's main.py file (also add utils.py if you added code to it).
- A python file model.py containing your reconstruction network implementation for the wet part (similar to vanilla.py).
- A PDF file named *wet.pdf* containing your wet part report.
- A PDF file named *dry.pdf* containing your dry part report.

Good Luck!