Deep Learning on Computational Accelerators
2367801 Mini-Project

Spring 2024

- **Submission Due**: 20/10/24, 23:59
- **TA In Charge**: Erez Koifman

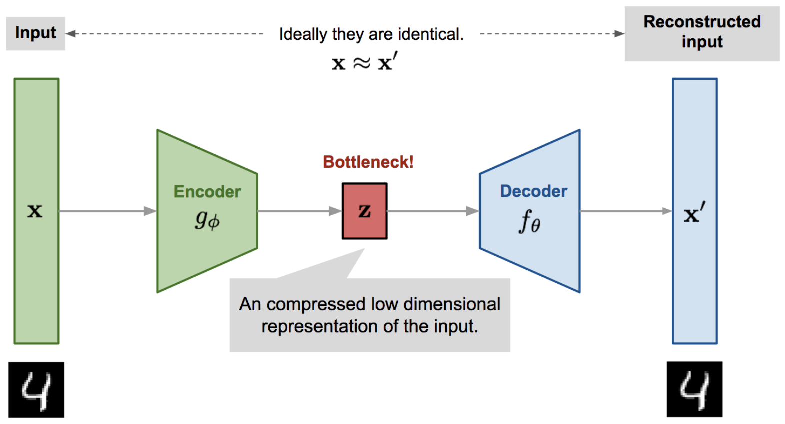
1 Wet Assignment

1.1 Introduction

In this project you will learn about the Variational Auto Encoder and the Variational Auto Decoder architectures, train generative models and experiment with their latent spaces.

1.2 Background

1.2.1 Auto-Encoders (AE)

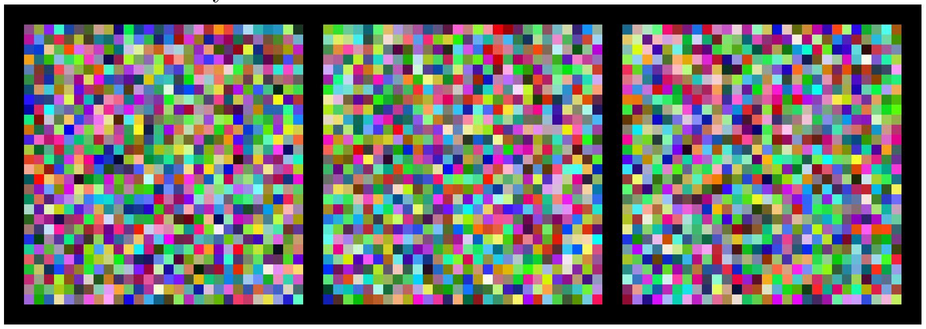

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Auto-encoders are a family of model architectures, which have 2 parts: encoder and decoder.
Given a dataset $X = \{x_1, \ldots, x_N\}$ sampled from a random variable $x \sim p(x)$, we will define a latent space with a random variable $z \sim p(z)$.

The encoder maps values from the data space x into the latent space z, meaning it learns $p(z \mid x)$, and the decoder maps values from the latent space z to the data space x meaning it learns $p(x \mid z)$.
This approach is self-supervised (as we can use the difference between the input and the reconstructed input as the loss) and the model learns to encode the data into compressed vectors while loosing as little information as possible.

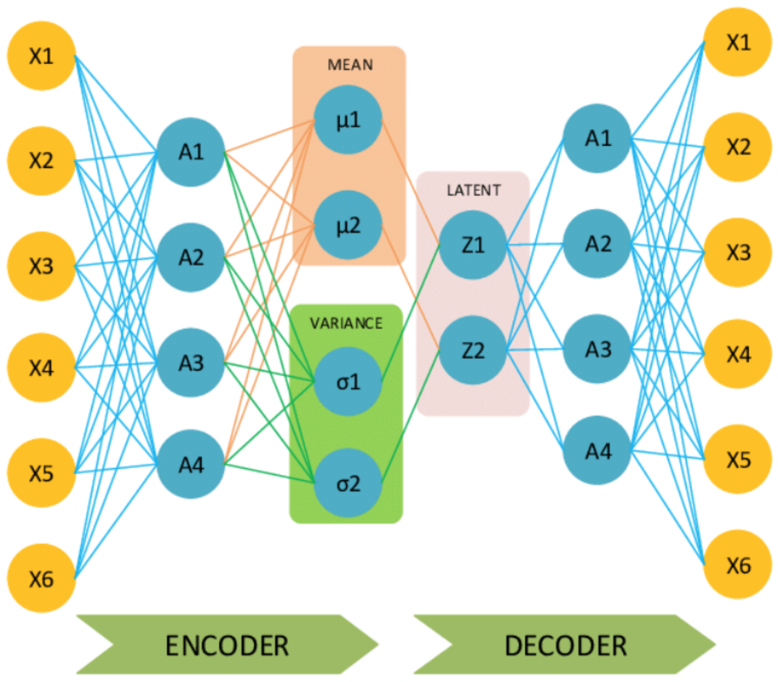
1.2.2 Latent Space

One question you may ask yourself at this point is why would this be possible? Why do we assume there exists a function that can compress data into a substantially smaller dimension without loosing information?
The answer to this question, is that most of the data doesn't represent anything with meaning.
In our case, we are going to be working with photos. If you generate random photos (like the ones attached below), the chances of them being anything other than noise are very low.



This property generalizes to other domains like text, voice, video, etc.
This means that all possible realistic images occupy only a small part of the total image space, and this small part can be mapped to and from a space with a much lower dimension.
Auto Encoders work because they learn to approximate this mapping function using neural networks.
It is important to note that there are many different mappings that each implicitly creates a latent space.
This means that without some kind of supervision, the structure of this latent space will be unpredictable and will not have any specific structure.
Because our mapping is not of the full latent space (as it is infinite and our neural network only approximates the function), we can map data into latents and mapped latents back into data, but we won't necessarily be able to map any data into latents (for example data that is not from the distribution the model was trained on) or map any latent into data. This may be a problem because our mapping is never for the full space (as it is infinite and we are approximating the mapping function) which means that

1.2.3 Variational Auto-Encoders (VAE)



One use for the Auto Encoder architecture is for generative tasks. For this purpose, the Variational Auto-Encoder was proposed. The idea in this architecture is to encode data into a known distribution instead of a vector, forcing the model to learn a range for each sample instead of single point in space.
In practice, this is done by taking the vector produced by the encoder, and using a layer to generate the parameters for the distribution we want for the latent space, sampling a latent vector from that distribution and decoding it.
For example, if the distribution we want is Gaussian (like in photo in this section), we take the output of the encoder, then use a network to calculate μ and a network to calculate σ^2 for each cell in the latent vector, we then draw values for each cell of the latent z_i from distribution $Z_i \sim \mathcal{U}(\mu_i, \sigma_i^2)$ and decode this latent vector to get the output of the model.
This approach allows us to generate new data by sampling a random latent vector from the learned distribution, and inputing it to the decoder.

1.2.4 The Reparameterization Trick

A main problem with this approach is that drawing values from random variables is a non derivable operator. This means we can't get a gradient for the latent vector which prevents us from training the encoder or the networks that predict the variables parameters.
To solve this, we use the reparameterization trick.

As you learned in probability, given a random variable $X \sim \mathcal{U}(\mu, \sigma^2)$, we can express it as $X = \sigma Z + \mu$ where $Z \sim \mathcal{U}(0, 1)$ (standard normal). Addition and multiplication are derivable, meaning if we sample Z and then multiply it by σ and add μ we can get a gradient for the latent vector with respect to μ and σ. This trick allows to train the network, and can work for many other distributions (which will be important later).

1.2.5 Training a Variational Auto Encoder

For the loss, the variational auto encoder uses the ELBO loss.
You have seen this loss in the tutorial about Diffusion, and it is recommended to go over it and understand it.

1.2.6 Variational Auto-Decoders (VAD)

As we explained in section 1.2.3, if we want to generate new data, we discrad the encoder after training, and only use the decoder.
This raises the question: can we train just the decoder?
The variational auto decoder is method for doing exactly this.
Instead of training an encoder, we initialize a random variable for each sample in our training set by initializing parameters that define a random variable from some known distribution.
We then optimize these parameters along with the decoder using gradient descent and the reparameterization trick we have seen in section 1.2.4. To test our model, we freeze the decoder's parameters then optimize a random variable for each sample in the test set and evaluate it based on the trained random variable.
When using the model we trained to generate new data, we only discard the random variables we trained for each sample, which is a lot less wasteful than discarding the entrie encoder.

1.2.7 Latent Space in Auto-Decoders

After training an Auto Decoder, we can get an expressive latent space which allows us to interpolate between different classes, and the model will generate interpolated data.

1.3 Assignment

1.3.1 Training an Auto-Decoder

In this part you will train an auto decoder on the Fashion MNIST dataset. This dataset contains photos of different clothing items like shirts, shoes and bags.
In total, it has 70K labeled samples, split into test and train sets but for this project we'll only use a small part of it.
You will be supplied with this dataset along with code to load and use it.

For this section, your task is to minimize the reconstruction loss for photos in the dataset.
Note that in this part, you are training an Auto Decoder and not a Variational Auto Decoder. This means that you are learning latent vectors for each sample, and not distributions. To train the model, initialise a random latent vector for each sample in the training set and optimize them along with the model's parameters.
After training the model:

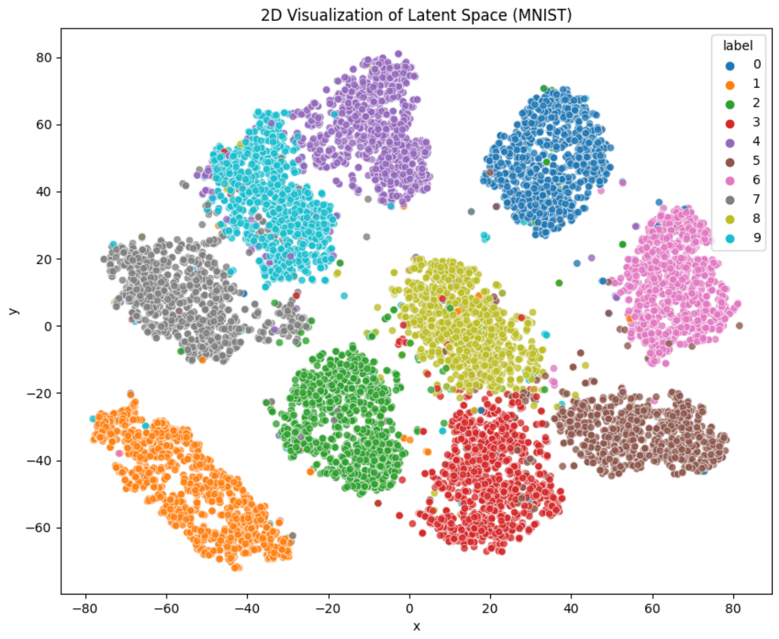
1. Write an overview of your Auto Decoder implementation. What architecture did you use and why? Explain your choices for the model parameters (layer parameters, model depth, etc.) and your choices for the training parameters (optimizer, optimizer parameters, etc.). Any choice is okay as long as long as you can justify it.

2. Evaluate your model on the training set and the test set and attach the results to your report.

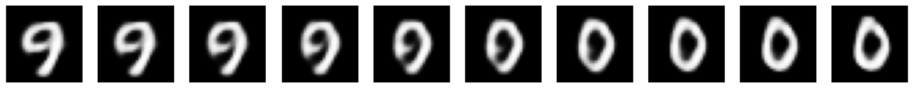
3. Sample 5 latent vectors from $\mathcal{U}(\mathbf{0}, \mathbf{I})$ and choose 5 latent vectors from the test set and decode them. Attach the results to your report.
   Compare the images you got from the test set latents to those you got from latents you sampled. Explain why the latents from the test set produced better results than the sampled latents.

1.3.2 Experimenting With Variational Auto Decoders and the Latent Space

In the previous section, we asked you to only minimize reconstruction loss.
This caused the models latent space to be disorganized, which is why you got bad results when trying to generate new data.
This can be visualized using t-distributed stochastic neighbor embedding (TSNE).
This technique projects the latent space to 2D, in a way that allows us to see how close different samples from the dataset are in the latent space.



To organize the latent space we will add a loss term that will normalize the random variables the model learns for each sample in the training set to some known distribution.
Then, when we sample from the distribution we normalize to, we will get good results more consistently.
Another nice property of a well structured latent space, is that we can interpolate latent vectors and get a mix of the sample they represent by decoding the interpolation.
For example, this is interpolation of samples of 0 and 9 from the MNIST dataset:



After implementing the model:

1. Write an overview of your Variational Auto Decoder implementation. What architecture did you use and why? Explain your choices for the model parameters (layer parameters, model depth, etc.) and your choices for the training parameters (optimizer, optimizer parameters, etc.). Any choice is okay as long as long as you can justify it.
2. Visualize the latent space of the model from section 1.3.1 using TSNE
3. Train a VAD on the dataset. Repeat the task from the previous section on this model.

4. Pick 2 different distributions you can use the reparametrization trick on. Train a VAD with each distribution. In your report, explain the distributions you picked and how you performed the reparametrization trick on them.

5. For each model, repeat the task from section 1.3.1 (with sampling from each model's matching distribution), and visualize the model's latent space using TSNE.

6. Pick 2 samples from the test set that are from different classes. Then, for each model:

   - Generate a latent vector for each sample
   - Calculate 5 evenly spaced linear interpolations of these vectors.
   - Decode the original vectors and their interpolations and plot them in a line (like in the attached image).

1.4 Code Base

- **utils.py**: this file includes the dataset object, along with a function that automatically creates the datasets and dataloaders for you. In addition, this file includes a function that creates tsne plots.
- **AutoDecoder.py**: you should implement your auto decoder in this file.
- **VariationalAutoDecoder.py**: you should implement your variational auto decoder in this file.
- **evaluate.py**: this file contains an evaluation function that tests the expressiveness of your model. You should use this function to evaluate your model on the test set in 1.3.1

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 want to use a 5-layer MLP to perform some regression task. Your friend claims that according to the Universal Approximation Theorem (UAT) for deep neural networks, any MLP with a single hidden layer can achieve optimal error over any dataset and loss criterion. He therefore concludes that there's never really a reason to use MLPs with more than one hidden layer.

   (a) Briefly explain what the UAT means, and how it supports the claim that you can achieve optimal error over any dataset and loss criteria.

   (b) Explain why his conclusion (never use more than one hidden layer) is wrong.

2. Alice and Bob want to perform image classification over a large, annotated dataset they found online. Alice suggests using an MLP, while Bob thinks it's better to use a CNN architecture.

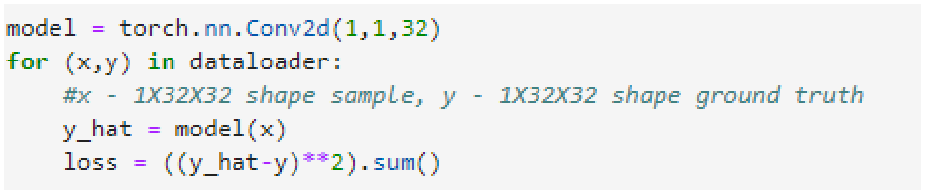
   (a) Help Bob convince Alice - what is the main advantage of using a CNN over an MLP in this case? What are the main difficulties Alice might encounter of she uses an MLP?

   (b) Alice heard your statements from the previous section, but is still not convinced. She claims that the convolution operation is in any case linear. Therefore, there shouldn't be a difference between using linear layers between activations (MLP) or convolutional operations between activations (CNN). Do you agree with her? Explain.

3. You are trying to solve an optimization problem where the overall loss w.r.t learnable parameters is convex (i.e. there's a single minimum value). Would you expect using momentum would help the optimization process? Explain.

4. Based on what we learned in the automatic differentiation mechanism tutorial, explain why Pytorch demands the loss tensor to be a scalar in order to perfrom backpropagation (i.e. run *loss.backward()*).

5. 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: Notice the loss term computation leverages the broadcasting mechanism.
   Hint 2: You don't have to understand how to strictly differentiate convolutional layers. Think about what the layer actually does in this specific case.



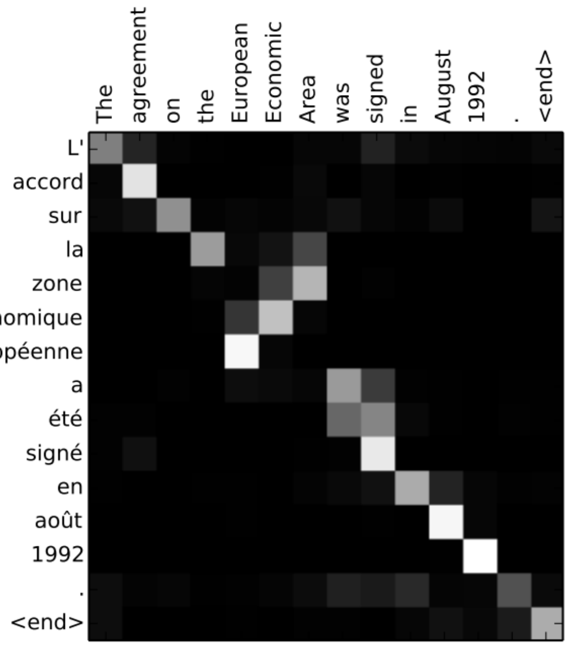
```
model = torch.nn.Conv2d(1,1,32)
for (x,y) in dataloader:
    #x - 1X32X32 shape sample, y - 1X32X32 shape ground truth
    y_hat = model(x)
    loss = ((y_hat-y)**2).sum()
```

6. 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.

7. Consider the following attention map, received during inference of a well-trained model performing English-French machine translation:



Explain the received attention map:

(a) What does each pixel in the map represent?

(b) What is the meaning of having rows with only one non-zero pixel?

(c) What is the meaning of rows that have several non-zero pixels?

(d) Explain why only rows with one non-zero pixel have a white pixels, while the rest have only gray pixels.

8. Explain what is the difference between the basic GAN we first presented in the GAN tutorial, and WGAN (Wasserstein-GAN) presented later on. What are the advantages of using WGAN over the basic GAN.

9. In the tutorial we've formulated:

$$\log p_\theta(x_0) = \mathbb{E}_q[\log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)} \cdot \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)})] = D_{KL}(q(x_{1:T}|x_0)||p_\theta(x_{1:T}|x_0)) + \mathbb{E}_q[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}]$$

In training we drop the KL-divergence term, and only optimize the ELBO term $\mathbb{E}_q[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}]$.

(a) What is the mathematical basis we rely on when we ignore the KL-divergence term?

(b) Why can't we compute the KL-divergence term?

(c) In the tutorial we further develop the ELBO term to

$$\mathcal{L}(\theta; x_0) = -D_{KL}(q(x_T|x_0)||p_\theta(x_T)) - \sum_{t>1} \mathbb{E}_q[D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}] + \mathbb{E}_q[\log p_\theta(x_0|x_1)]$$

In training we ignore the term $-D_{KL}(q(x_T|x_0)||p_\theta(x_T))$ Explain why.

10. (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.

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 AutoDecoder.py and VariationalAutoDecoder.py files and any additional files you used for their implementation.
- A PDF file named *wet.pdf* containing your wet part report.
- A PDF file named *dry.pdf* containing your dry part report.