Deep Learning on Computational Accelerators
236781 Mini-Project

Winter 2024-25

- **Submission Due**: 23/3/25, 23:59
- **TA In Charge**: Tamir Shor

1 Wet Assignment

In this project you will experiment with the task of autoencoders and latent representations over common computer-vision datasets.
In section 1.1 we supply some basic background about the topic. In section 1.2 we describe what you'll be doing. Section 1.3 elaborates the code we'll be supplying you with for this project. Section 1.4 specifies how we'll evaluate your work.

1.1 Background

1.1.1 Latent Spaces In Deep-Learning

Latent spaces are widely used in deep-learning for efficient data representation and processing.
Natural data (e.g. images, acoustic waveforms, 3D structures) can be large, or contain a lot of data that is irrelevant to the downstream task at hand. For instance - consider the MNIST dataset [1], which we widely used in our tutorials. The vast majority of pixels processed by the models we trained are 0, and contain no information relevant for our downstream task. This irrelevant information increases our computational resource consumption as it is still processed by our networks. In some cases this redundant information can also make the learning task more difficult.
One very common approach to address those challenges is representation learning. Namely, the input natural data at hand is projected from the original input dimension onto a lower dimension h, that should represent each datum's most distinct features, rather than hold the entire input signal. These projected representations are often termed *latent* (hidden) representations, and the

vector-space they form is termed a *latent space*. In this document we will refer the the original input-space representation space as *signal domain*, and to the compressed, latent representation as *latent space*.
The projection onto a smaller dimension, often termed *encoding*, can be done with classic methods (e.g. PCA), however nowadays it is prevalently done using neural networks.

1.1.2 Autoencoders

On common way to optimize for good latent representations is autoencoders (figure 1). Autoencoders are (usually) neural models composed of two sub-modules - and encoder and the decoder.
In training, the encoder receives the input data in signal-domain and encodes it onto its latent representation. The decoder receives the latent representation and decodes it back onto the signal domain (or, sometimes, to some other dimension, dependent on the downstream task). Training is usually self-supervised - we do not need ground-truth labels, but rather try to have the decoder reconstruct the original input signal from its latent representation. This objective forces the encoder to well-encapsulate input features in its learned latent space.

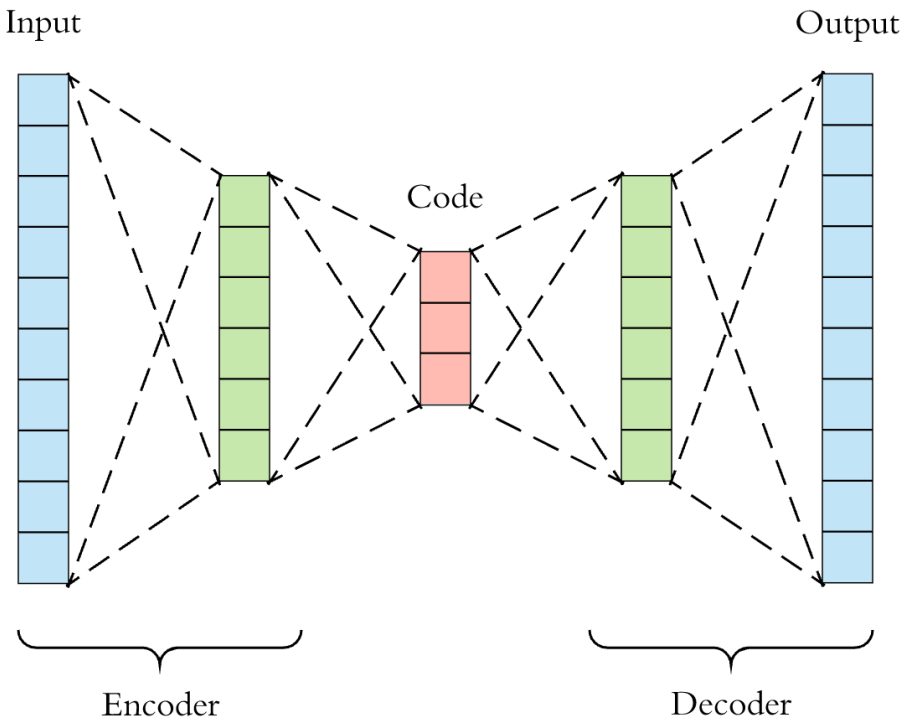


Figure 1: Autoencoder Model

At inference time, we simply use the encoder. The decoder is only required in training to train the encoder namely supply it with some feedback over the quality of its compression (much like the generator and discriminator in GANs, only in this case the two modules do not have conflicting objectives).

1.2 Assignment

In this assignment you will train a total of 6 autoencoder models to create meaningful latent representations for the task of image classification, over the MNIST [1] and CIFAR10 [2] datasets. For each of the two datasets you must complete three sub-tasks - 1.2.1, 1.2.2 and 1.2.3.

1.2.1 Self-Supervised Autoencoding

In this section you will train your latent space in a self-supervised manner, disjointly from the downstream classification task. Your will first train an autoencoder to project images onto a latent space (encoder) and reconstruct them back to the original input image (decoder). Then, you will use your pre-trained encoder by training a classifer over its learned latent representation.

1. Implement an autoencoder model (you may choose different implementations for both datasets) and train it to project your image data onto a single latent vector of dimension $h = 128$. Train with the self-supervised reconstruction objective (section 1.1.2). Namely, in training, the objective of the decoder should be to reconstruct the full input image x in the best possible way.

2. Train your classifier (in a fully-supervised manner) over the corresponding dataset. For each input image (or batch of images), you should first encode your image to a latent representation with your pre-trained encoder (from the previous step). Then, feed the 128-dimensional encoded input vector onto the classifier, to output class prediction. **Note** - make sure your encoder is frozen in this stage (not being optimized). You will probably still want to backpropagate through it in optimization.

1.2.2 Classification-Guided Encoding

In this section you will train your latent space jointly with the downstream classification task. Unlike in the previous section (1.2.1), in this section your decoder should not try to reconstruct the image from its latent representation, but directly try to perform classification over the latent code. The main difference here is that your encoder must be trained jointly with the classifier, to minimize classification error rather than reconstruction error.

1.2.3 Structured Latent Spaces

The definition of quality of learned latent spaces is elusive. This is partially because the corresponding latent encodings are high-dimensional and difficult

to interpret by a human, but also because this question is highly dependent of the specific designated use-case (in our case – image classification). Nonetheless, since latent spaces are expected to compress data features, it is common to expect that under some norm (say, cosine-similarity), the latent encodings of semantically similar data in signal domain would also be similar.
One issue with training autoencoders is that they often tend to converge to irregular latent spaces. Namely, in the task-formulation we defined previously for autoencoders, nothing actually enforces these latent representation to hold those desired traits of similarity and structure. This is likely to make the downstream classification task more difficult.

In this section you will implement some method of your choice to improve the latent structure of the latent space your encoder learns. For simplicity, in this exercise we evaluate the quality of latent-space structure only based on the separability of different latent encodings according to their ground-truth labels. Namely, in this section you will implement some method to be introduced during training of your encoder to make your encoded samples more easy to classify. To measure the success of your developed method we will measure both your classification accuracy (with and without the added logic), and based on visualizations as elaborated in section 1.4.
While you may use any strategy you think will be useful to promote easier downstream classification, in this section we introduce contrastive learning, which you may find especially useful.

Contrastive learning is a self-supervised learning technique used to learn meaningful representations of data, without relying on labels. Instead of directly training on classification labels or data reconstruction, the model learns by distinguishing between similar and dissimilar pairs of data points.

A key idea in contrastive learning is that in order to promote a well-structured latent space, similar samples (e.g. different augmented views of the same image) should have similar representations, while dissimilar samples (different images) should have distinct representations. This is often achieved by using a *contrastive loss function*. Such functions are designed to penalize similarity for latents originating from different signal-domain samples, whilst promoting similarity between samples we believe should have similar encodings. One common example is the **NT-Xent (Normalized Temperature-scaled Cross Entropy Loss)** used in **SimCLR** ([3]) - a widely used framework for contrastive learning.

For more information, refer to:

- SimCLR Paper: *A Simple Framework for Contrastive Learning of Visual Representations*
- A blog post on contrastive learning: Contrastive Representation Learning (Lilian Weng)
- *This* practical explanation.

Whether you choose to implement contrastive learning or any other mechanism for the designated task displayed here, in this section you must train the encoder with your newly-introduced logic (without training it specifically for the classification objective) and evaluate the learned representations by training a classifier on top of the enhanced pre-trained encoder. Make sure to use the same classifier architecture as section 1.2.1 or section 1.2.2.

1.3 Codebase

The codebase we provide consists of the two files:

- **main.py** - In this file you should implement and train your autoencoders and classifiers. We also include some basic examples and important code-related comments. If you wish to create new files (*main* files for other experiments, files for models, etc.) that is fine, however be sure to submit all of your code. **Note** - We've made all required data available for you on Lambda, on the path *"/datasets/cv_datasets/data"*. To avoid overcrowding the server (and make everything slower for everyone), please refrain from downloading the datasets to your personal partitions on Lambda.

- **utils.py** - This file includes the plot_tnse function required for some of your plots (see section 1.4). You may add other useful utility function to this file, however if you do, please include that file as well in your final submission zip.
  ‘

1.4 Evaluation

You will submit a report including the following components:

1. **Reasoning** - An overview of your solution - what autoencoder architectures did you use for each task (1.2.1,1.2.2)? Why did you think it was a good choice? Detail your considerations. Any choice is okay as long as you reason about it. Also describe how you performed network parameter choices (e.g. depth, hidden dimension, number of kernels etc) and hyperparameter tuning (learn rates, batch sizes, dropout etc). Also present your approach for section 1.2.3. Cite relevant sources from the literature (if applicable), and thoroughly explain your logic and motivation for using the selected method.
   Note - don't forget to use a validation set.

2. **Quantitative Results** - For each of the six encoders you trained, present downstream classification accuracy over your training, validation and test sets . For the two self-supervised training autoencoders, also report mean reconstruction error. You may train with any objective you choose, however you must report you reconstruction error in mean absolute error. Specifically, relate to whether your method from section 1.2.3 has been

successful, by comparing the performance of classifiers trained with the section 1.2.3 encoders to the performance of classifiers trained in previous sections.

3. **Qualitative Results** - For the two self-supervised autoencoders, randomly choose 5 images and plot each image next to its reconstruction outputted by your trained decoder.

4. **Linear Interpolation** - Out of the 5 images you have chosen from the MNIST test-set, choose 1 pairs of image. Perform a 10-step linear interpolation between the two encodings of the pair (from the self-supervised encoders, section 1.2.1), as given by the self-supervised encoder. Feed each interpolated latent through your pre-trained decoder (again, the one trained with the self-supervised objective) and plot its output (in image-domain). Analyze received interpolations - do they seem like hand-written digits that could interpolate your two images in image-domain or not. Why do you think that is?

5. Read about t-SNE projections here (or anywhere else). For each of the 6 autoencoders you trained, use the plot_tsne function we supplied you with to compare latent and image-domain representations. What is the general difference between latent spaces attained from self-supervised training (section 1.2.1) and classification-guided training (1.2.2)? What are the general differences between signal-domain representations and latent representations?
   Specifically, relate to whether (and how) your method from section 1.2.3 helped shape the learned latent space to be more separable. If it hadn't achieved the desired effect, explain why do you think that is.

Some Important Notes

:

- You must implement all models yourself. Any classifiers you train in this assignment are not allowed to directly receive images as input.

- You are encouraged to do some research (papers, blogs) to see how others solved the problem you are trying to solve and gain inspiration. That being said, we will not accept solutions completely copied from existing solutions, without any implementation on your part. You must implement your own model. Failing to do so would lead to disqualifying your solution. We also stress that the knowledge you gathered in this course alone is more than enough to solve this assignment and get perfect grade, without needing any external code or theory.

- We suggest you refrain from using overly large (more than 3 layers) models for your encoder and decoder - this is not necessary for good results, and will prolong optimization and evaluation times in both sections. If any of

the sections seem computationally heavy to you, reduce model size and use larger batch sizes.

- All plots *must* include a title and a short explanation of what's in the plot. You must also include axis labels and grids. Vague plots will incur grade penalty.
- Your report will be graded based on contents, but also based on presentation. Make sure to display your arguments in a clear, thorough and concise manner.
- Your report must address *all* requirements presented in the report description in this section.
- As a basic sanity check, you should be able to easily attain a classification accuracy of at least 60% in all experiments. We expect that at least in some experiments you reach accuracies much better than that.

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. In the above wet assignment we specified you should use latent vectors of dimension 128. Imagine we'd ask you to experiment with a larger range of latent dimensions, where for each dimension you'd be training a new latent space and then try to classify over that learned space. Assuming only the latent dimension changes in each experiment, how do you think the classification accuracy would change with respect to the hidden dimension? Namely, would it decrease up to a certain h value and then increase with growth in h, increase up to a certain h value and then decrease with growth in h, or would it be monotonically increasing/decreasing? Explain your answer.
2. 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.

3. 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.

4. In the optimization tutorial we learned about several optimization algorithms performing Exponential Moving Average (EMA). What would happened if instead of EMA, we'd be using linear averaging of accumulated gradients? Would you expect the algorithm's efficiency be compromised? Explain

5. 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 perform backpropagation (i.e. run *loss.backward()*).

6. (a) Explain the term *Inductive Bias*

   (b) What is the inductive bias of CNNs? Give at least 2 potential "biases".

   (c) Generally, what are the pros and cons of training a model with inductive bias? In what cases is more inductive bias good? In what cases would we prefer avoiding it?

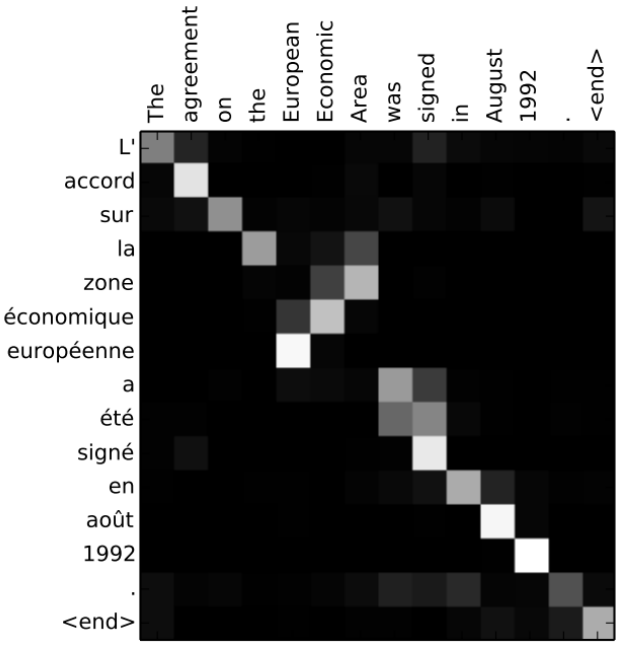
7. You are trying to solve some sequence modeling problem using self-attention. Your input sequence has n tokens, encoded to key, query and value matrices $Q, K, V \in \mathbb{R}^{n \times d}$.

   (a) What is the computational time complexity of computing the output of the attention layer - $Softmax(\frac{QK^T}{\sqrt{d}}) \cdot V$?

   (b) Assuming $d << n$, Bob offers to reduce the computational complexity by changing the attention operation to $Softmax(\frac{Q}{\sqrt{d}})(K^T \cdot V)$ (note the parenthesis around the $K^T \cdot V$, indicating this product is computed first). Why does this help reduce computational time complexity? Does this preserve the function of the original attention formulation?

8. 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.

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. When working on the wet assignment, Bob made the following distinction - *"In the self-supervised autoencoder trained over MNIST, the decoder component learns to map latent vectors onto real MNIST images. We can randomly draw random vectors, feed them to the decoder to get a generative model of handwritten digit images!"*.

    (a) Explain why Bob is wrong.

    (b) In spite of the issue with Bob's idea, his partner Alice became very excited about the notion of optimizing an autoencoder to pose as generative model. She came across *Variational Autoncoders* (VAEs) - explain what those are, how they differ from "standard" autoencoders (like the one you trained in the wet assignment), and how the modifications applied to them allow them to pose as generative models.
    You may use any source of information, however we specifically recommend you review our course's VAE tutorial (not covered this semester)

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, and any other code you may have added.
- A PDF file named *wet.pdf* containing your wet part report.
- A PDF file named *dry.pdf* containing your dry part report.

Good Luck!

References

[1] Li Deng. "The mnist database of handwritten digit images for machine learning research [best of the web]". In: *IEEE signal processing magazine* 29.6 (2012), pp. 141–142.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[3] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.