Assignment 3

CMSC 678 — Introduction to Machine Learning

Due Wednesday May 3rd, 11:59 PM

Item	Summary
Assigned	Wednesday April 19th
Due	Wednesday May 3rd
Topic	Neural Networks
Points	105

In this assignment you will experiment with some small neural networks, become comfortable with the math of feed forward networks; and implement, experiment with, and compare neural network classifiers.

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Discord discussions).

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4
Points	35	20	20	30

For this assignment, you may use as a starting reference the following colab notebook:
`https://colab.research.google.com/drive/114o91vWoj5SBy9aiMryG5ew8a-2xGEn2?usp=sharing`

What To Turn In Turn in a **PDF** writeup that answers the questions; turn in all requested code **and models** necessary to replicate your results. Turn in models by serializing the learned models and parameters. Be sure to include specific instructions on how to build/compile (if necessary) and run your code. Answer the following questions in long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment on the submission site:

https://www.csee.umbc.edu/courses/graduate/678/spring23/submit.

Be sure to select "`Assignment 3`."

Motivation/Preamble

In class, we've so far studied neural networks from a classification perspective: given a D-dimensional input vector $x_i \in \mathbb{R}^D$, learn to predict the correct class y_i^* via new "features" h (the output of hidden layers). These features h are generally of a smaller dimension than the original input x_i. Most often, we learn to classify by optimizing K-dimensional cross entropy loss.

We've also studied dimensionality reduction techniques, such as PCA. In those techniques, we also learn to compute a lower dimensional representation z_i for each x_i. However, unlike learning the features to be effective for classification, in dimensionality reduction we learn to compute z_i so that it is able to *reconstruct* the original input x_i. As we saw with PCA, learning this reconstruction can be achieved by optimizing MSE loss (though this is not the only way).

While not necessarily obvious, it shouldn't be too surprising to ask if we can combine these two techniques and motivations: that is, can we formulate a neural network-like model that has the data reconstruction aims of dimensionality reduction techniques? Well, the answer is yes! Such a model is called a (neural) **autoencoder**. Very simply, we can formulate an autoencoder as an **encoder** network $\mathcal{E}$ followed by a **decoder** network $\mathcal{D}$. Each of $\mathcal{E}$ and $\mathcal{D}$ can be neural networks (or even something simpler like *just* linear layers). The job of the encoder $\mathcal{E}$ is to learn to compute a lower dimensional representation z of the input x, while the job of the decoder is to use z to compute a representation $\hat{x}$ that is "close" to the original x.

You will gain experience with neural autoencoders. Neural autoencoders form the basis for a *very* robust and active subfield of machine learning, and they form the basis for some pivotal work. In question 1, you will step through the formulation of a neural autoencoder. In question 2, you will then implement a reconstruction-based autoencoder and, in a pipelined manner, use it to construct features for a separate classification objective. In question 3 you will extend your autoencoder with a classification-based loss, and compare it with the approach in question 2. In question 4, you will see how encoder-decoder models can be used within a reinforcement learning setup (no implementation needed for question 4).

You will be using the MNIST dataset. This has both benefits and drawbacks: MNIST is a relatively "easy" and accessible dataset, which can make it very good for getting a good understanding of the core algorithms and methods. However, because it is easy, simpler techniques that would struggle on harder datasets can actually do quite well, while conversely more advanced and complex techniques that are necessary for better performance on those harder datasets don't do as well (basically, they're too complex for the simplicity of the problem). You want to be careful not to draw conclusions about comparative effectiveness of methods from these simpler datasets. The point of this assignment is to give you a deep understanding of how neural networks are implemented and trained, and for you to make connections across course topics. It is not to achieve stellar performance on MNIST.

Questions

1. (**35 points**) Let each input x_i be a D-dimensional input vector, $x_i \in \mathbb{R}^D$. Your overall goal in this question is to formulate the reconstruction loss for an autoencoder and then derive the partial derivatives of the loss function.

   Let f be an activation function. We're going to define our encoder $\mathcal{E}$ as computing z via the output of a linear layer passed through f:

   $$\mathcal{E} : z = f(Wx) \qquad \text{[Eq-1]}$$

   Specifically, the encoder takes in the D-dimensional input vector x and computes an E-dimensional representation z.

We're going to define the decoder $\mathcal{D}$ similarly, where g is also an appropriate activation function:

$$\mathcal{D} : \hat{x} = g(Uz). \qquad \text{[Eq-2]}$$

The decoder takes in the E-dimensional vector z and computes a D-dimensional representation $\hat{x}$. We set g based on the range of values in x: if each coordinate of x could be any real number, then g can be the identity. However, if x is a binary vector, then g can be the sigmoid function.

The goal is to learn to produce $\hat{x}$ such that it is "close" to the original x. In this approach, note that we are using the original values of x as the "labels." To properly formulate this, we'll need a loss function $\mathcal{L}(\text{true} = x, \text{predicted} = \hat{x})$.

(a) There are a number of ways to enforce this "closeness" criterion, i.e., choose the loss function. One approach could be to use MSE: $\mathcal{L}_{\text{MSE}}(x, \hat{x}) = \sum_d (x_d - \hat{x}_d)^2$. Another approach could be to use a binary cross entropy loss on each component/coordinate of x: $\mathcal{L}_{\text{xent}}(x, \hat{x}) = \sum_d \text{CE}(x_d, \hat{x}_d)$, where CE refers to the cross entropy loss, and we ensure each coordinate $\hat{x}_d$ is a number between 0 and 1. When would we want to use MSE, and when would we want to use the per-component binary cross entropy loss?

(b) If $x = (1, 0)^T$, $W = ((0.5, -0.5))$, $U = ((0.3), (-0.3))$, and f is the sigmoid function, compute z, $\hat{x}$, and $\mathcal{L}_{\text{xent}}(x, \hat{x})$. You must show your work, though you may use the A3 colab notebook as a way to *validate* your answers.

(c) What are the parameters that must be learned?

(d) If $D = 100, E = 50$, how many parameters are there in total?

(e) Now, let D and E be fixed but unknown. Let the loss $\mathcal{L}$ be component-wise binary cross entropy. For each variable ♣, derive the partial derivate $\frac{\partial \mathcal{L}}{\partial \clubsuit}$. Since you cannot assume any particular values for D or E, you may provide a general partial derivative formula for some subset of the variables, a second formula for another subset, etc. The formulas you do provide must give a way of computing the partial derivative for each variable.

(f) Neither [Eq-1] nor [Eq-2] include an explicit bias; describe how you can include the bias with minimal effort.

2. (**20 points**) Implement a neural autoencoder, according to [Eq-1] and [Eq-2]. Using a per-component binary cross entropy loss, train it on the MNIST data (e.g., train on `int-train` and evaluate it on `int-dev` from previous assignments). You should quantify experimental progress by tracking the value of the loss function (aggregated across batches within an epoch) for both training and dev.

(a) In your **report**, discuss your implementation and convergence criteria.

(b) In your **report**, experiment with at least two different *configurations* and document the internal development progress through quantifiable and readable means, i.e., graphs and/or tables showing the loss on training and the loss on the dev set that different model configurations yield. Provide your own analysis/discussion and summarization of these results.

(c) Use your trained autoencoder in a classification task on MNIST. Specifically, use your autoencoder to compute a lower dimensional representation z for an input x, and then use that representation z as features in a classification model. The implementation for *this question* should be pipelined, so that training the classifier does not impact the weights of the autoencoder. (This is outlined in the colab notebook.) Compare to a baseline of your choice. Evaluate your autoencoder-based classifier and the baseline on the original 10,000 image `test` set. Include these evaluation results in your **report** and discuss the results.

Configuration types include anything impacting the model's input, output, or internal computations. They include but aren't limited to:

- the size of the dimensionality-reduced representation (i.e., E);
- activation functions (e.g., sigmoid, tanh, ReLU);
- bias (present or not, learned or not) for each layer.

You may use Pytorch (or similar) to compute the gradient and to perform the gradient descent (including setting the learning rate).

3. (**20 points**) Now, extend the training of your neural autoencoder to include a classification loss $\mathcal{L}_c$. This classification loss should learn to predict the image's label from the learned encoding z, while simultaneously learning z to be good at reconstructing the original input image. If the loss from the previous question is $\mathcal{L}_{\text{autoenc}}$, then for this question you can optimize the joint loss $\mathcal{L}_c + \mathcal{L}_{\text{autoenc}}$. Evaluate this joint learning approach, compare it to the results you got from the previous question, and summarize your take-aways.

   For this question, you do *not* have to do multiple dev configurations.

4. (**30 points**) Read Zhang and Lapata (2017):

```
@InProceedings{zhang2017simplification_rl,
  title={Sentence Simplification with Deep Reinforcement Learning},
  author={Zhang, Xingxing and Lapata, Mirella},
  booktitle={EMNLP},
  year={2017}
}
```

   available at `https://aclanthology.org/D17-1062.pdf`. Then, write a 1/2-3/4 page reflection piece. Your reflection should

   - summarize the approach.
   - identify the components of an MDP in the DRESS formulation. If a particular component(s) are not described, explain your answer.
   - compare the loss functions between DRESS and the neural encoder-decoder approach described in Section 2.
   - contain a brief summary of the findings/conclusions of this paper, clearly stating the importance of these conclusions.