Homework 5

CSCI 5525: Machine Learning

Due on Dec 13 (Friday) 11:55 pm

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to fill in above to specify which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online.
- Look up things/post on sites like Quora, StackExchange, etc.

Submission. Submit a PDF using this LaTeX template for written assignment part and submit .py files for all programming part. You should upload all the files on Canvas.

Written Assignment

Instruction. For each problem, you are required to write down a full mathematical proof to establish the claim.

Problem 1. PCA.

(**Total 7 points**)

- (**3 point**) Consider the full SVD of $X = USV^\intercal$. Define

$$S_1 := \{D \in \mathbb{R}^{d\times k} : D^\intercal D = I\}, S_2 := \{V^\intercal D : D \in S_1\}.$$

  Show that $S_1 = S_2$.

- (**4 points**) Now use the fact above to show that

$$\max_{D\in S_1} \|XD\|_F^2 = \max_{D\in S_1} \|SD\|_F^2$$

(a) For the SVD, we know that $VV^T = I$. Thus:

$$(V^T D)^T (V^T D) = D^T V V^T D = D^T D = I$$

Which means: $\forall X \in S_2, X^T X = I$, we get $S_2 \subset S_1$.

Then, $\forall D \in S_1$, $D = V^T(VD^T) = V^TD' \in S_2$ (Apparently, $D'^TD' = I$, thus $D' \in S_1$), we get $S_1 \subset S_2$.
Because $S_1 \subset S_2$ and $S_1 \subset S_2$, we get the proof that $S_1 = S_2$.

(b)For the SVD, we know that $VV^T = I$ and $UU^T = I$.

$$\begin{aligned}\max_{D\in S_1} \|XD\|_F^2 &= \max_{D\in S_1} tr((XD)^TXD) \\ &= \max_{D\in S_1} tr(D^TX^TXD) \\ &= \max_{D\in S_1} tr(D^TVS^TU^TUSV^TD) \\ &= \max_{D\in S_1} tr(D^TVS^TSV^TD)\end{aligned}$$

Let v_i be the row i in matrix V, s_i be the i_{th} element of the the diagonal matrix S. Because $VV^T = I$, when $i = j, v_iv_j^T = 1$, otherwise, $v_iv_j = 0$. As we only consider the trace of the matrix in the above formula, that is to say, in this case, the multiplication of V parts all are 1, so we have:

$$\begin{aligned}\max_{D\in S_1} \|XD\|_F^2 &= \max_{D\in S_1} tr(D^TVS^TSV^TD) \\ &= \max_{D\in S_1} tr(D^TS^TSD) \\ &= \max_{D\in S_1} \|SD\|_F^2\end{aligned}$$

Problem 2. MLE .

(**5 points**) Consider that the n samples are drawn from a uniform distribution , $X_i \sim \text{unif}(a, b)$ and the likelihood function is given as $\mathcal{L}(a, b) = \frac{1}{(b-a)^n}$. Derive the expression for MLE.

Your answer. Apparently, when $b > a$, $\frac{\partial \mathcal{L}}{\partial a} = \frac{n}{(b-a)^{n+1}} > 0$, $\frac{\partial \mathcal{L}}{\partial b} = \frac{-n}{(b-a)^{n+1}} < 0$, so if we want to minimize the likelihood function, we need to make a as small as possible while b as big as possible. Then, we get $a = \min\{X_i\}$, $b = \max\{X_i\}$

Problem 3. Perceptron convergence .

(**10 points**)
In this problem you need to show that when the two classes are linearly separable, the perceptron algorithm will converge. Specifically, for a binary classification dataset of N data points, where every x_i has a corresponding label $y_i \in \{-1, 1\}$ and is normalized: $\|x_i\| = \sqrt{x_i^Tx_i} = \mathbf{1}, \forall i \in \{1, 2, .., N\}$, the perceptron algorithm proceeds as below:

In other words, weights are updated right after the perceptron makes a mistake (weights remain unchanged if the perceptron makes no mistakes). Let the (classification) margin for a hyperplane $\mathbf{w}$ be $\gamma(\mathbf{w}) = \min_{i\in[N]} \frac{|\mathbf{w}^T\mathbf{x_i}|}{\|\mathbf{w}\|}$ (convince yourself that $\gamma(\mathbf{w})$ is the smallest distance of any data point from the hyperplane). Let $\mathbf{w}_{opt}$ be the optimal hyperplane, i.e. it linearly separates the classes with maximum margin. Note that since data is linearly separable, there will always exist some $\mathbf{w}_{opt}$. Let $\gamma = \gamma(\mathbf{w}_{opt})$.

Algorithm 1 Perceptron

while no converged **do**
  Pick a data point $\mathbf{x_i}$ randomly
  Make a prediction $y = \text{sign}(\mathbf{w}^T\mathbf{x_i})$ using current $\mathbf{w}$
  if $y \neq y_i$ **then**
    $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x_i}$
  end if
end while

Following the steps below, you will show that the perceptron algorithm makes a finite number of mistakes that is at most γ^{-2}, and therefore the algorithm must converge.

Step 1: Show that if the alorithm makes a mistake, the update rule moves it towards the direction of the optimal weights $\mathbf{w}_{opt}$. Specifically, denoting explicitly the updating iteration index by k, the current weight vector by $\mathbf{w}_k$, and the updated weight vector by $\mathbf{w}_{k+1}$, show that, if $y_i\mathbf{w}_k^T\mathbf{x}_i < 0$, we have

$$\mathbf{w}_{k+1}^T\mathbf{w}_{opt} \geq \mathbf{w}_k^T\mathbf{w}_{opt} + \gamma\|\mathbf{w}_{opt}\|$$

Hint: Consider $(\mathbf{w}_{k+1} - \mathbf{w}_k)^T\mathbf{w}_{opt}$ and consider the property of $\mathbf{w}_{opt}$.

Step 2: Show that the length of updated weights does not increase by a large amount. Mathematically show that, if $y_i\mathbf{w}_k^T\mathbf{x}_i < 0$

$$\|\mathbf{w}_{k+1}\|^2 \leq \|\mathbf{w}_k\|^2 + 1$$

Hint: Consider $\|\mathbf{w}_{k+1}\|^2$ and substitute $\mathbf{w}_{k+1}$.

Step 3: Assume that the initial weight vector $\mathbf{w}_0 = 0$ (all-zero vector). Using results from step 1 and step 2, show that for any iteration $k+1$, with M being the total number of mistakes the algorithm has made for the first k iterations, we have

$$\gamma M \leq \|\mathbf{w}_{k+1}\| \leq \sqrt{M}$$

Hint: Use Cauchy-Schwartz inequality $\mathbf{a}^T\mathbf{b} \leq \|\mathbf{a}\|\|\mathbf{b}\|$ and telescopic sum.

Step 4: Using result of step 3, conclude $M \leq \gamma^{-2}$.

Your answer. Step 1:

$$\begin{aligned}
&\mathbf{w}_{k+1}^T\mathbf{w}_{opt} \geq \mathbf{w}_k^T\mathbf{w}_{opt} + \gamma\|\mathbf{w}_{opt}\| \\
&\iff (\mathbf{w}_{k+1}^T - \mathbf{w}_k^T)\mathbf{w}_{opt} \geq \gamma\|\mathbf{w}_{opt}\| \\
&\iff (\mathbf{w}_{k+1}^T - \mathbf{w}_k^T)\frac{\mathbf{w}_{opt}}{\|\mathbf{w}_{opt}\|} \geq \gamma = \min_{i\in[N]} \frac{|\mathbf{w}_{opt}^T\mathbf{x_i}|}{\|\mathbf{w}_{opt}\|} \\
&\iff \frac{y_i\mathbf{x}_i^T\mathbf{w}_{opt}}{\|\mathbf{w}_{opt}\|} \geq \min_{i\in[N]} \frac{|\mathbf{w}_{opt}^T\mathbf{x_i}|}{\|\mathbf{w}_{opt}\|}
\end{aligned}$$

Apparently, with the optimal w, $y_i\mathbf{x}_i^T\mathbf{w}_{opt} > 0$, so we have:

$$\frac{y_i\mathbf{x}_i^T\mathbf{w}_{opt}}{\|\mathbf{w}_{opt}\|} = \frac{|\mathbf{x_i}^T\mathbf{w}_{opt}|}{\|\mathbf{w}_{opt}\|} = \frac{|\mathbf{w}_{opt}^T\mathbf{x_i}|}{\|\mathbf{w}_{opt}\|} \geq \min_{i\in[N]} \frac{|\mathbf{w}_{opt}^T\mathbf{x_i}|}{\|\mathbf{w}_{opt}\|}$$

Step 2:

$$\begin{aligned}\|\mathbf{w}_{k+1}\|^2 &= \mathbf{w}_{k+1}^T\mathbf{w}_{k+1}\\ &= (\mathbf{w}_k + y_i\mathbf{x}_i)^T(\mathbf{w}_k + y_i\mathbf{x}_i)\\ &= \mathbf{w}_k^T\mathbf{w}_k + y_i\mathbf{w}_k^T\mathbf{x}_i + y_i\mathbf{x}_i^T\mathbf{w}_k + y_i^2\mathbf{x}_i^T\mathbf{x}_i\\ &= \|\mathbf{w}_k\|^2 + 2y_i\mathbf{x}_i^T\mathbf{w}_k + 1\\ &\leq \|\mathbf{w}_k\|^2 + 1\end{aligned}$$

Step 3:
For $\|\mathbf{w}_{k+1}\| \leq \sqrt{M}$, with the conclusion in the step 2, we know that if $y_i\mathbf{w}_k^T\mathbf{x}_i < 0$, $\|\mathbf{w}_{k+1}\|^2 \leq \|\mathbf{w}_k\|^2 + 1$. When $y_i\mathbf{w}_k^T\mathbf{x}_i < 0$, the algorithm makes mistake. Because the total number of mistakes is M, we have:

$$\|\mathbf{w}_{k+1}\|^2 \leq \|\mathbf{w}_0\|^2 + M = M$$

Thus, $\|\mathbf{w}_{k+1}\| \leq \sqrt{M}$.
For $\|\mathbf{w}_{k+1}\| \geq \gamma M$, with the conclusion in the step 1, when $y_i\mathbf{w}_k^T\mathbf{x}_i < 0$, $(\mathbf{w}_{k+1}^T - \mathbf{w}_k^T)\mathbf{w}_{opt} \geq \gamma\|\mathbf{w}_{opt}\|$, and the Cauchy-Schwartz inequality, we have:

$$\begin{aligned}\|\mathbf{w}_{k+1}\|\|\mathbf{w}_{opt}\| &\geq \mathbf{w}_{k+1}^T\mathbf{w}_{opt}\\ &= (\mathbf{w}_{k+1}^T - \mathbf{w}_k^T)\mathbf{w}_{opt} + (\mathbf{w}_k^T - \mathbf{w}_{k-1}^T)\mathbf{w}_{opt} + ... + (\mathbf{w}_1^T - \mathbf{w}_0^T)\mathbf{w}_{opt} + \mathbf{w}_0^T\mathbf{w}_{opt}\\ &\geq M\gamma\|\mathbf{w}_{opt}\|\end{aligned}$$

Thus, $\|\mathbf{w}_{k+1}\| \geq \gamma M$.
Step 4:
With Step 3:

$$\begin{aligned}\gamma M \leq \|\mathbf{w}_{k+1}\| \leq \sqrt{M}\\ \iff \gamma^2 M^2 \leq M\\ \iff M \leq \gamma^{-2}\end{aligned}$$

Problem 4. GAN .

(**5 points**) Consider the standard GAN objective

$$\min_\theta \max_\gamma V(G_\theta, D_\gamma) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\ln D_\gamma(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\ln(1 - D_\gamma(G_\theta(\mathbf{z})))]$$

Show that for a given generator, G, the optimal discriminator is given by

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

Hint: it might be useful to show that the function $p\ln(a) + q\ln(1-a)$ is concave and has maximum $p/(p+q)$ for any $p, q > 0$.

Your answer. Firstly find the second derivative of the function in the hint(let l the function):

$$\frac{\partial^2 l}{\partial a^2} = -\frac{p}{a^2} - \frac{q}{(1-a)^2}$$

Apparently, $\forall p, q > 0$, $\frac{\partial^2 l}{\partial a^2} < 0$, that is to say, function l is concave downward.
Then find the first derivative of l and let it equal 0. As l is concave downward, we can get an a which maximizes l:

$$\begin{aligned} \frac{\partial l}{\partial a} = \frac{p}{a} - \frac{q}{1-a} &= 0 \\ &\Longleftrightarrow \\ a &= \frac{p}{p+q} \end{aligned}$$

For a given G, let $D_G^* = \arg\max_D V(G, D)$. The GAN objective is equivalent to:

$$\begin{aligned} V(G, D_G^*) &= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\ln D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\ln(1 - D_G^*(G(\mathbf{z})))] \\ &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \ln D_G^*(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{z}} p(\mathbf{z}) \ln(1 - D_G^*(G(\mathbf{z}))) d\mathbf{z} \\ &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \ln D_G^*(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{x}} p_G(\mathbf{x}) \ln(1 - D_G^*(\mathbf{x})) d\mathbf{x} \\ &= \int_{\mathbf{x}} [p_{\text{data}}(\mathbf{x}) \ln D_G^*(\mathbf{x}) + p_G(\mathbf{x}) \ln(1 - D_G^*(\mathbf{x}))] d\mathbf{x} \end{aligned}$$

Thus, according to function l and conclusion above, to maximize $V(G, D_G)$:

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

Programming Assignment

Instruction. For each problem, you are required to report descriptions and results in the PDF and submit code as python file (.py) (as per the question).

- **Python** version: Python 3.
- Please follow PEP 8 style of writing your Python code for better readability in case you are wondering how to name functions & variables, put comments and indent your code
- **Packages allowed**: numpy, pandas, matplotlib, pytorch
- **Submission**: Submit report, description and explanation of results in the main PDF and code in .py files.
- Please **PROPERLY COMMENT** your code in order to have utmost readability otherwise **1 MARK** would be deducted

Programming Common

This programming assignment focuses on implementing generative models for MNIST - you have already used MNIST in previous assignments.

This assignment needs to be implemented in python/pytorch and it has to be for the CPU version only.

Note Report the results, explanation or plots in the PDF (e.g. plots outside the pdf will not be accepted) and submit the files as asked in the respective questions.

Problem 4. GAN.

(**Total 18 Points**) Fig 1 shows a simple GAN model. In this simple version, both the generator and discriminator are fully connected neural networks and you will see that this simple variant is also able to generate recognizable digits. The generator takes random inputs and created samples matching the dimension of the training samples. The discriminator takes samples from training set as well as generated samples and attempts to recognize if a sample is real (i.e. coming from training set) or fake (i.e. generated one). Use MNIST dataset similar to hw3.

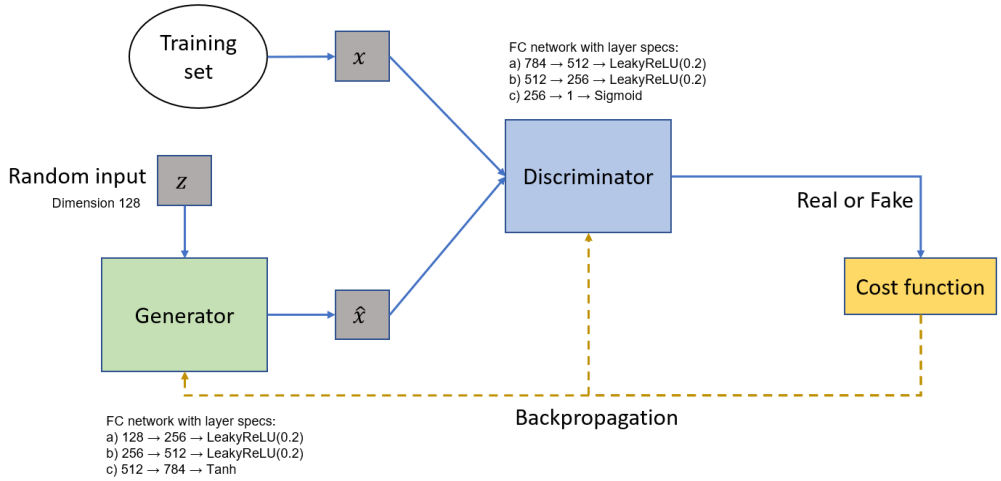


Figure 1: GAN.

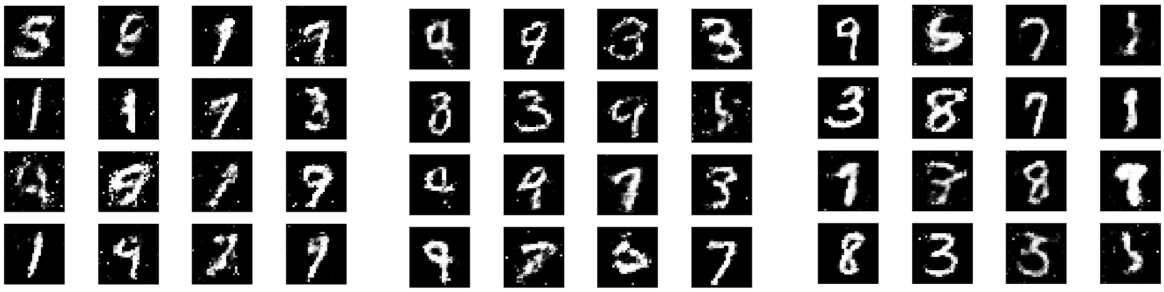
a) (**14 Points**) Implement a simple GAN model consisting of fully connected networks. Consider the dimension of z to be 128. The required specifications are clearly mentioned in the fig 1. In HW3, you have already seen and used cross entropy loss and optimizer. Here, you will use the function binary_cross_entropy or BCE loss (refer Link to BCE loss documentation). Remember, here you have to optimize for both the generator and the discriminator parameters. A rough sketch of training would be first generate fake images, get real images then train discriminator and generator one after the other. In this homework, we will follow this basic approach only for simplicity. Use number of epochs as a stopping criteria. Also, here you will use a leaky ReLU with slope given in specifications (some of you already wanted to use it in convolutional network, here it is!). Use ADAM optimizer you are already familiar with. Consider the batch size as 100. This information is more than sufficient for you to implement a basic GAN (There are hundreds of types of GANs to emphasize on improving the quality and diversity of generated images, but training time will increase. Therefore, this basic variant should give you an idea of its working). Keep in mind that this is in initial few epochs, GAN training losses could be inherently unstable, however, you should be able to see stability and improvement as the training progresses.

b) (**4 Points**) Run it for 50 epochs and show a plot of generator loss and discriminator loss for epochs. Every 10 epochs, generate 16 images from generator and show them a 4x4 grid. Report each of these generated image grids in PDF. Save your trained models with the names (hw5_gan_dis.pth and hw5_gan_gen.pth). Submit the saved model.

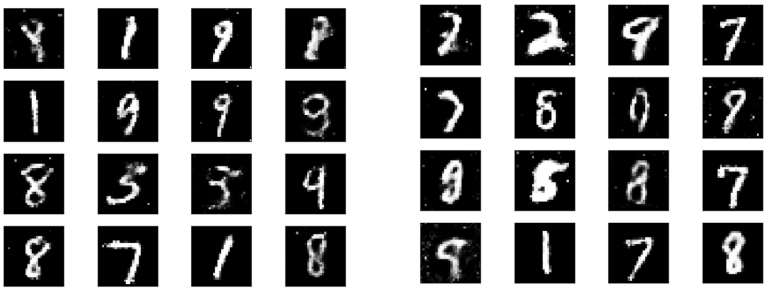
Note: The hints provided are more than enough to answer the question.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw5_gan.py).

In this problem, the learning rates for both generator and discriminator are 0.0002, the generated images are show in figure 2, we can see that when epoch increases, the noise in the image decreases.
As for the GAN loss in the figure 3, the generator loss increases first then suddenly decreases, while the discriminator loss decreases at the same time when generator loss increases. We can see that these two networks are competing each other, one loss increases, the other will decrease. This confirms the characteristic of 'Adversarial'.

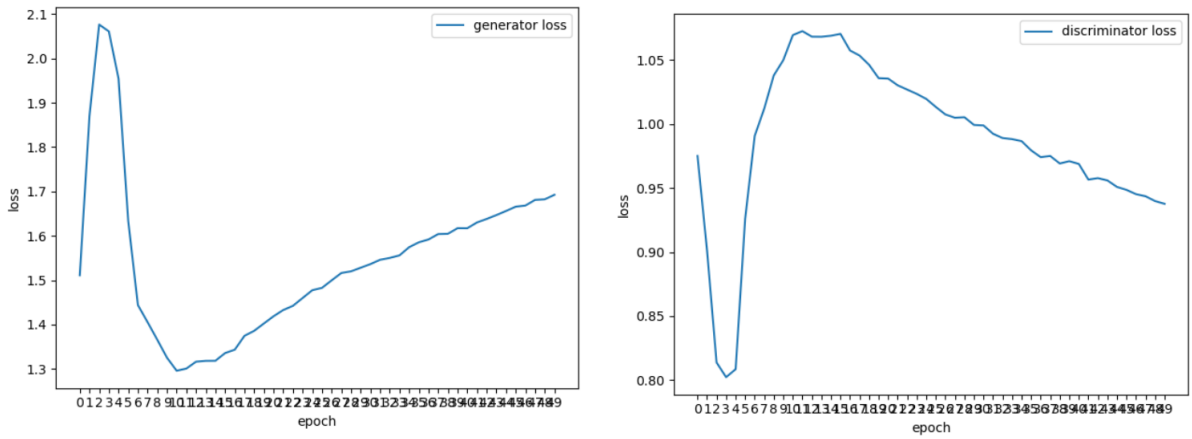


(a) epoch=10 (b) epoch=20 (c) epoch=30



(d) epoch=40 (e) epoch=50

Figure 2: Generated images



(a) Generator loss

(b) Discriminator loss

Figure 3: GAN Loss

Problem 5. Denoising AE.

(**Total 15 Points**)
As shown in the schema in fig 4, the auto encoder encodes the input x through hidden layers to find a dense representation of the data and then this dense representation is decoded resulting in a reconstruction of the original inputs reconstructed features. Now, in order to learn interesting features, one common tactics is to introduce some noise to the input data points before encoding them and then compare the resulting reconstruction to the original. By introducing the artificial noise, we ensure that the network does not learn an identity mapping but rather the network will learn to ignore the noise and learn useful features. This approach is referred to as a denoising autoencoder (dAE). Use MNIST dataset similar to hw3.

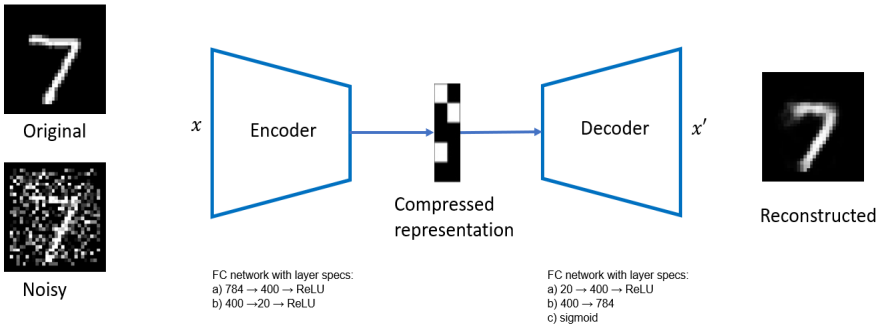


Figure 4: dAE.

a) (**10 Points**) Implement a basic dAE model consisting of fully connected networks. The network specifications are given in fig 4. Add random noise to the input images and use the cross entropy loss. Use ADAM optimizer. In HW3, you have already seen and used cross entropy loss and optimizer. Here, you will use the function binary_cross_entropy or BCE loss (refer Link to BCE loss documentation). Consider the batch size as 64. This information is sufficient for you to implement the dAE.

   Run it for 10 epochs and show a plot of average loss for epochs. (Please do not show loss per iteration). Save your model with the name (hw5_dAE.pth). Submit the saved model.

b) (**3 Points**) Now, consider test images. Make a 2x5 grid and in the first row show 5 noisy test images (by adding the noise as above) and in the second row show corresponding reconstructed images from your model. You should be able to see that this simple auto-encoder has denoised the images.

c) (**2 Points**) Try to briefly explain why the autoencoder architecture can finish denoising job. Give a mathematical sense.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw5_dAE.py) and saved model named (hw5_dAE.pth).

(a) In the code, the learning rate is 0.001. The average loss is shown in figure 5.

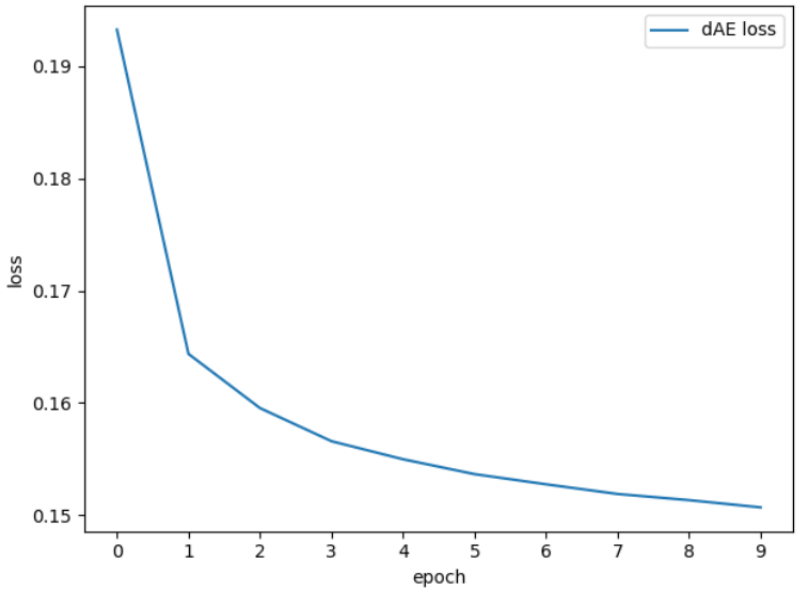


Figure 5: dAE loss

(b)The comparison before and after denoising is shown in figure 6, I generate Gaussian noise and add the noise to the original images directly to produce noising images.

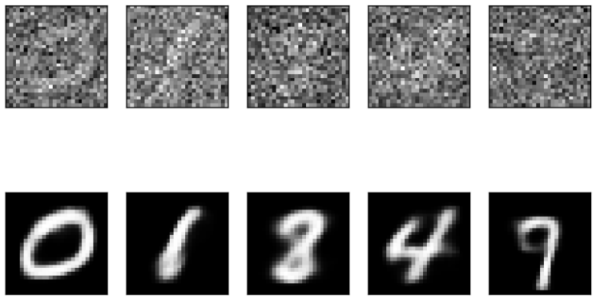


Figure 6: Denoise

(c) The encoding method will lose information, so during the training the encoder learns to keep and compress the necessary information while discard the others. The decoder learns to decompress the information.

Problem 6. VAE.

(**Extra Credits: Total 15 Points**)
Your goal here is to implement a variational auto-encoder (VAE). Let z be the hidden state representation. Encoder learns a mapping $x \to z$ and decoder learns how to reconstruct x' from z. For VAE, the loss function has two terms: the reconstruction loss between x and x' and KL-divergence to make our approximate learned distribution $q(z|x)$ similar to the true distribution $p(z)$, assumed to be unit Gaussian. The reconstruction loss is given as the cross entropy between x, x' and the KL-divergence term can be written as $\frac{1}{2}\sum_j(1+\log((\sigma_j^2))-(\mu_j^2)-(\sigma_j^2))$ so you can implement it in this architecture. (refer original paper on VAE - Auto-Encoding Variational Bayes). Both the encoder and decoder are implemented using neural networks. A high level view is given in the fig 7 Use MNIST dataset similar to hw3.

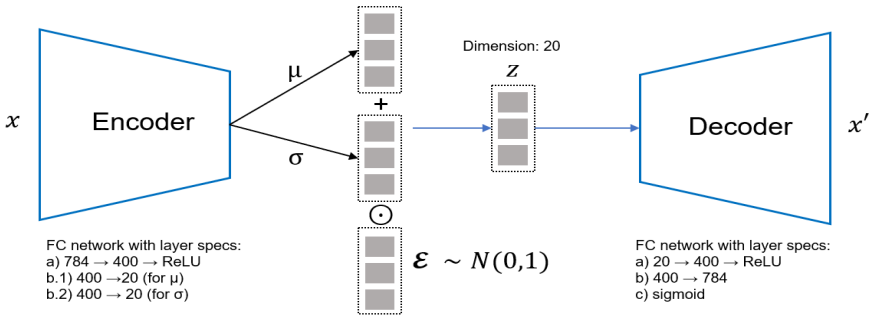


Figure 7: VAE.

a) (**12 Points**) Implement a simple VAE model consisting of fully connected networks. Consider the dimension of z to be 20. The layer specifications are given in the fig 7. Use ADAM optimizer. In HW3, you have already seen and used cross entropy loss and optimizer. Here, you can use the function binary_cross_entropy or BCE loss (refer Link to BCE loss documentation) with reduction='sum'. Consider the batch size as 64. This information is sufficient for you to implement a simple VAE.

   Run it for 10 epochs and show a plot of average loss for epochs. (Please do not show loss per iteration). Save your model with the name (hw5_vae.pth). Submit the saved model.

b) (**3 Points**) For the test set, randomly generate 16 new Z from unit Gaussian, see if we can generate images similar in MNIST dataset. Show a 4×4 grid.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw5_vae.py) and your saved model (hw5_vae.pth).

(a) In the code, the learning rate is 0.0005. The average loss is shown in figure 8, it decreases with epoch.

(b) The generated images are shown in figure 9.

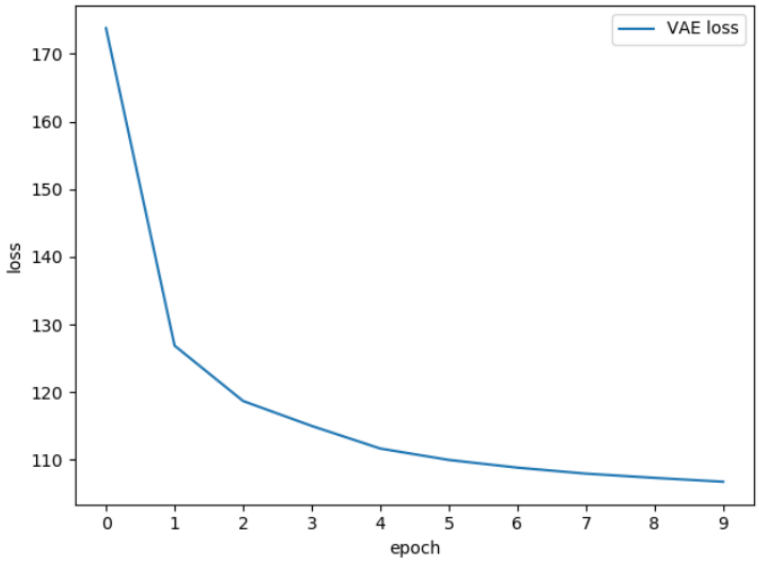


Figure 8: VAE loss

Figure 9: Generated images

Problem 7. Adversarial examples.

(**Extra Credits: 10 Points**)
Background In last few years, deep learning had made strides in many different areas with wide ranging claims of super human performances. Although the performance of deep learning

algorithms is impressive, it is far from robustness seen in humans. For example, you have already learned about convolution based deep learning classifiers for images and their performance in object recognition. One of the popularity of deep learning comes from the provision that you can use pre-trained models. However, a key question is are these models robust enough to be compared with humans? Can we create "examples" by perturbing the test images in a way that it can actually fool these state of the art classifier? Such examples are called adversarial examples.

In this problem, you will use a popular pre-trained classification network - resnet. It can be loaded using

```
from torchvision.models import resnet50
model = resnet50(pretrained=True)
pred_vector = model(test_input_image)
```

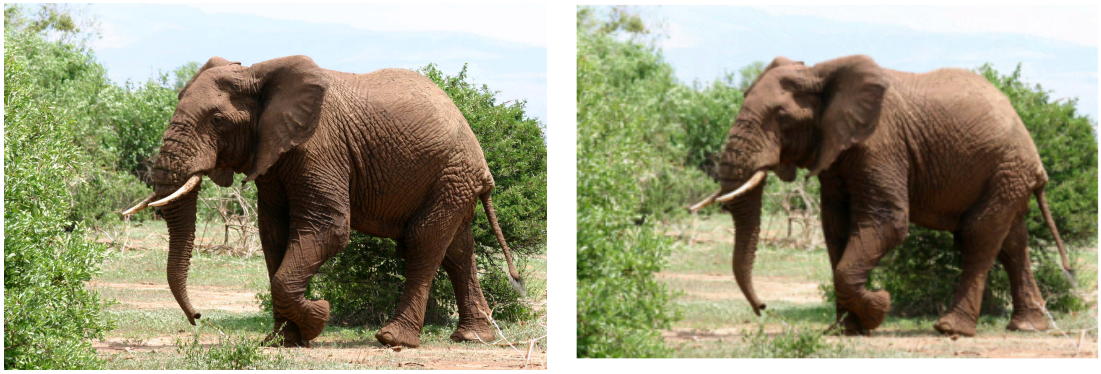
To predict, you can use this model now by passing a test image tensor ("Elephant2.jpg", you would need to resize the image to 224 and convert to tensor in pytorch so you can use it with resnet). This can be done as

```
preprocess = transforms.Compose([
transforms.Resize(224),
transforms.ToTensor(),
])
my_tensor = preprocess(my_img)[None,:,:,:]
```

Now, you normalize it (use imagenet mean [0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]) and ask resnet to classify it.

```
pred_vector = model(normalized_my_tensor)
```

which gives you a prediction vector. To find the label of the class, you can use the index for the max value in the predicted vector obtained from the resnet model for your test image and create a lookup function in the json. The json file has indices and labels. For example, for index 101, the label is 'tusker'. That is your image label from the classifier.



((a)) Original image ((b)) Adversarial image

Figure 10: Adversarial examples: Original example is in (a). The example in (b) is an adversarial example (with some small perturbations as you will implement in this assignment). Although to humans, both the examples look like 'tusker', however, the classifier labels the second image as a bullet_train.

As you know, machine learning models are trained by finding model parameters that minimize the empirical loss on the training set and use gradient descent to make updates to the

model parameters. Using the model predicted output and index of the true class, you can use cross entropy function in PyTorch to get the loss value. Now, your goal in this assignment is to devise and implement a way to create adversarial examples, thereby, a targeted attack so that the pre-trained network starts classifying the given example as something else. (Note, pre-trained network itself will not be modified and the image should not be completely changed). For example, the elephant image in fig 10a is classified as "tusker" which is good but manipulated image in fig 10b is classified as a bullet_train.

Assignment Problem:

a) Devise and implement a method so you can fool the classifier by modifying the image in some way. Show the image of your final adversarial example and report what the classifier labels it.
(*Hint*: Now, the question you have to think is - how can you manipulate the image so that the pre-trained network thinks it will be something else? Remember, model parameters, θ, are updated using gradients w.r.t θ. But that's not the only thing available to you. Provided you can compute a loss value, you can think of using optimization for some small perturbation in x also. You can use cross entropy loss, for example:
`nn.CrossEntropyLoss()(model(some_normalized_my_tensor)),torch.LongTensor([101])).item()`
gives you loss with tusker class itself. You can use SGD to optimize for small perturbation but clamp this perturbation within some ϵ - that you choose. Because, to create adversarial examples, you do not want to completely change your image too (that's not useful) but it should be something that looks almost the same to the humans but classifier makes a mistake.)

b) Extend on the method in part (a) further to devise and implement a method so you can make the classifier label the 'tusker' image (given image) as a 'bullet_train'. Show the image of your final adversarial example and report what the classifier labels it.
(*Hint*: Think of a mechanism of how much to manipulate your image using a new loss function such that it allows the prediction to be closer to the targeted class i.e. bullet_train but farther to the original class. The loss function should now consist of two parts. Similar to the above, find small perturbations to the image now optimized for the new criteria.)

Note: The hints provided are more than enough to answer the question. You have to create only one adversarial example. You are given two files, one elephant image and one json file.
Submission: Submit all plots requested and explanation in latex PDF. Submit both the steps of your program in a file named (hw5_adv_examples.py).

(a) The idea to fool the classifier is quite simple, we generate a trainable noise with the same size of the original image, then add this noise to the original image and put the mixed image into the classier to get a prediction, after that, we use the cross entropy loss to compute the prediction with true label (in this case, the true label is a long vector that only the index for the 'tusker' is set 1). As we want to fool the classifier, instead of gradient descent, we implement gradient ascent to the noise tensor to make the differences between the prediction and the true label bigger, so we can do it by multiple the loss by -1. Also, to make the adversarial image look same as the original one, we need to use clamp to bound the noise. In my code, I use SGD as the optimizer, the learning rate is 1e-3, the bound for the noise is [-1e-2, 1e-2]. The result is shown in figure 11.

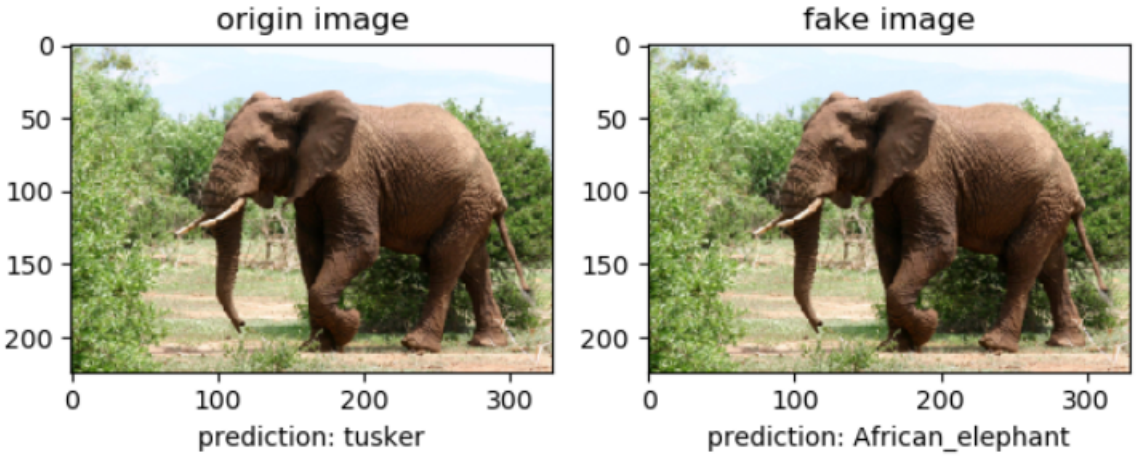


Figure 11: fool the Resnet

(b) The idea is almost same as the former question, in the former question, we multiple the loss by -1. Here, we add another positive cross entropy loss w.r.t the prediction and the true label of the 'bullet_train'. This infers an idea that we do the gradient descent to the label 'bullet_train' while do the gradient ascent to the label 'tusker'. In my code, my learning rate is 2e-2. The result is shown in figure 12.

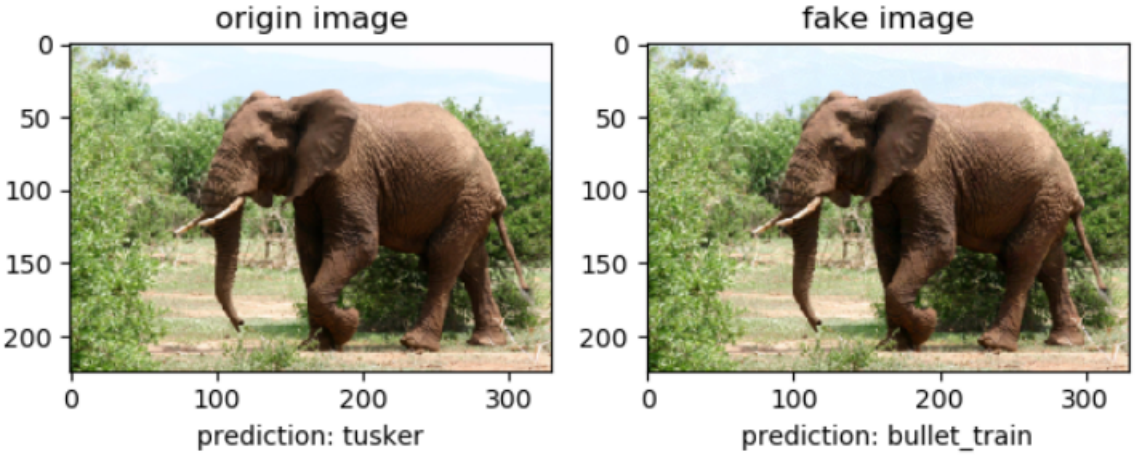


Figure 12: fool the Resnet with a target