**CNN-Lytical**

**Mentors : Vaibhav Raj, Ashwin Ramachandran, Akshat Kumar**

**Keywords :** CNNs, Adversarial Attacks, DeepFool, FGSM, MNIST, Neural Networks, ML

| **Mentee** | **Roll Number** | **Email-Id** |
| --- | --- | --- |
| Soham Joshi | 210051004 | sohamjoshitab2@gmail.com |

**Adversarial Attacks on Computer Vision Models**

In this project, I explored different methods to attack neural networks, mainly centred upon introducing perturbations to misclassify examples in a trained neural network by the generation of adverserial examples. Implement FGSM, DeepFool Methods of Attack. Use one method for the generation of adversarial examples while the other provides input to compare the effectiveness of these methods

**Progress**

In this project, I did the following Programming Assignments :

1. Assignment 1 - Building a single layer neural network that classifies the MNIST dataset from scratch
2. Assignment 2 - Building a neural network using PyTorch for classifying the MNIST dataset
3. Assignment 3 - A CNN for classifying the CIFAR-10 dataset.

In addition, I studied some lectures for the Project-Selection Assignment belonging to the CS231n lectures by Andrej Karpathy

The implementation project consisted of study and implementation of the DeepFool and FGSM algorithms. I studied the following papers :

1. Advances in adversarial attacks and defenses in computer vision: A survey by Naveed Akhtar, Ajmal Mian, Navid Kardan and Mubarak Shah
2. EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES by Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
3. DeepFool: a simple and accurate method to fool deep neural networks by Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard Ecole Polytechnique F ́ed ́erale de Lausanne

For implementation of algorithms, I used Google Colab for coding and implemented :

1. FGSM (Fast Gradient Sign Method) Attack on Neural Networks : In this implementation (FGSM.ipynb), I have used the first programming assignment and in the same spirit of doing everything from scratch, have implemented the FGSM algorithm using only the numpy library. The implementation contains cells which show :

i. Generation of adverserial examples

ii. Normal example generation and predictions

iii. Accuracy upon some fixed maximum perturbation using the FGSM attack

1. DeepFool Algorithm : In this implementation (DeepFool.ipynb), I have again used the first programming assignment and have used only the numpy library. The implementation contains :

i. Generation of adverserial examples

ii. Normal example generation and predictions

iii. Accuracy upon some fixed maximum perturbation using the FGSM attack

The workflow of the assignment checkpoints involved watching the video lectures of CS231n followed by solving the assignments on Google Colab Notebooks taking help from the official PyTorch documentation, while asking difficulties encountered on the discord server. Implementing a higher level of abstraction using the PyTorch API was often filled with searching the net and at times time consuming, however, that effort in searching up the documentation and understanding the underlying math had to be put.

Calculations largely involved a strong understanding of matrix calculus and linear algebra. Implementing networks actually can be done without understanding any math and just dealing with a layer of abstraction but I found that understanding the underlying math concepts was extremely rewarding.

In the implementation of adversarial networks, I first read the papers mentioned above and then implemented the two algorithms mentioned above using pure numpy (no API). That is, I implemented all the matrix perturbations from scratch and indeed my findings matched with the results published in the above papers.

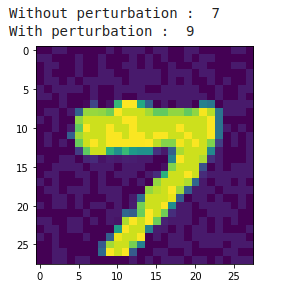
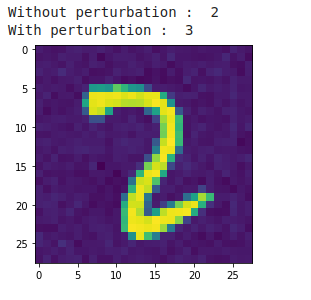
**Results**

Let us start with the FGSM method of attacking neural networks. In the FGSM method, we compute the gradient of the loss function with respect to the input features and multiply it with “epsilon” and add it to the input feature vector.

Now, I have observed the following trend with %accuracy of the neural net and the value of epsilon.

| **epsilon** | **accuracy(%)** |
| --- | --- |
| 0.01 | 83.5 |
| 0.05 | 62.4 |
| 0.1 | 24.6 |
| 0.25 | 0.1 |

In the DeepFool algorithm, I have implemented a loop which runs until the image is pushed out of the “decision boundary” of the classifier. In this scenario, the image is indistinguishable from the “unperturbed” image in majority of cases. The perturbed images in both cases are shown below.

****

DeepFool FGSM

As you can see, the matrix perturbations change the original MNIST images slightly so that they look okay to the human eye, but are misclassified by neural networks. The main causes for this are mentioned in the research papers attached in the github repo linked below. The papers contain the survey, history and mathematical analysis of what is causing such misclassification and why do adversarial examples work for such a wide variety of neural networks. The resources I used are listed under references/citations.

**Learning Value**

In this project, I have learned a lot of skills such as :

1. Looking up documentation and figuring out APIs
2. Applying concepts of continuous math in discrete code
3. Reading research papers and implementing algorithms as described in the paper and cross-verify the results as obtained in the paper
4. Proper Documentation of your work
5. Tacking a topic which I had no idea about beforehand.

This project gave me a good push to explore a lot of math and ML myself, and actively read papers atleast the ones which I can understand.

**Software Used**

1. Google Colab Notebooks
2. VS Code (text editor)

**Suggestions for Others**

I would advise anyone starting with this project, or ML in general, to implement a neural network from scratch at least once before switching to an API and dealing with neural networks abstractly. This is one of the biggest reasons why I implemented even the adversarial algorithms using only numpy. Also, matrix calculus can be rather tiring so derive the formulas once then use them as axioms of sorts. Also, it is very important to brush up on your linear algebra since vectorisation of operations is used heavily in ML implementations. Once you get a grip upon how to implement basic neural networks and are able to solve the assignments (which will take a lot of time, so be patient), start reading the papers (link in the “citations” section). Also, don’t be afraid of ML, its difficulty is hyped a bit, if you are comfortable with math , you can probably understand what is happening under the hood.

**References / Citations**

Github repository : <https://github.com/Ihsoj-Mahos/SoC-Soham_Joshi>

Research papers :

1. <https://arxiv.org/pdf/1412.6572.pdf> (FGSM Method)
2. <https://arxiv.org/pdf/1511.04599.pdf> (DeepFool)
3. <https://arxiv.org/pdf/2108.00401.pdf> (Survey of Adversarial attacks)

Assignments - <https://github.com/CNN-lytical>

CS231n Lectures Link : <https://www.youtube.com/watch?v=i94OvYb6noo&list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC&index=5>

Slides of CS231n : <http://cs231n.stanford.edu/slides/2016/>

ConvNet demo : <https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

60 minute blitz through PyTorch - <https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html>

Extremely well-written PyTorch documentation-

<https://pytorch.org/docs/stable/index.html>

Torchvision, because we are dealing with images - <https://pytorch.org/vision/stable/index.html>

PyTorch transforms for augmentation and processing of images - <https://pytorch.org/vision/stable/transforms.html>

Sample FGSM implementation-

<https://pytorch.org/tutorials/beginner/fgsm_tutorial.html>

Final presentation video -

**Disclaimer**

Copyright Disclaimer : All the materials used are openly available, and all use of material in this project and in the creation of documentation is within the limits of allowed use.

Fair Use Disclaimer : Fair use of all research papers attached in this document has been done, they have all been cited under the references/citations section