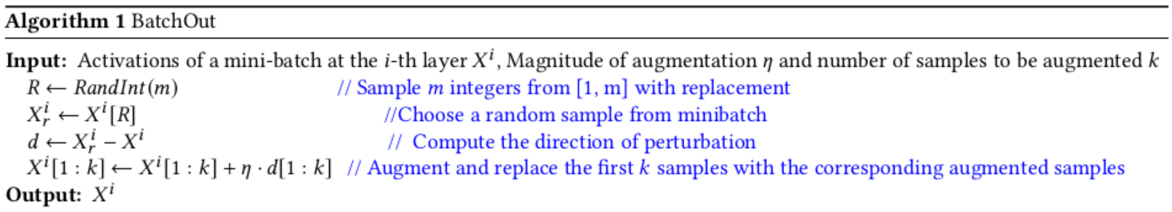
Report on Batchout

Batchout is an augmentation technique in the feature space. The main aim of batchout is to make the model adversarially robust. Batchout is a simple linear convex combination between two datapoints

x = x_1 + n * (x_2 - x_1) where x_1, x_2 are data-points and n is parameter.

The technique is called batchout because the above augmentation takes place with datapoints belonging to the same batch.

The original batchout paper implementation:



```
Algorithm 1 BatchOut
Input: Activations of a mini-batch at the $i$-th layer $X^i$, Magnitude of augmentation $\eta$ and number of samples to be augmented $k$
  $R \leftarrow RandInt(m)$                                   // Sample $m$ integers from [1, m] with replacement
  $X_r^i \leftarrow X^i[R]$                                   //Choose a random sample from minibatch
  $d \leftarrow X_r^i - X^i$                                  // Compute the direction of perturbation
  $X^i[1:k] \leftarrow X^i[1:k] + \eta \cdot d[1:k]$          // Augment and replace the first $k$ samples with the corresponding augmented samples
Output: $X^i$
```

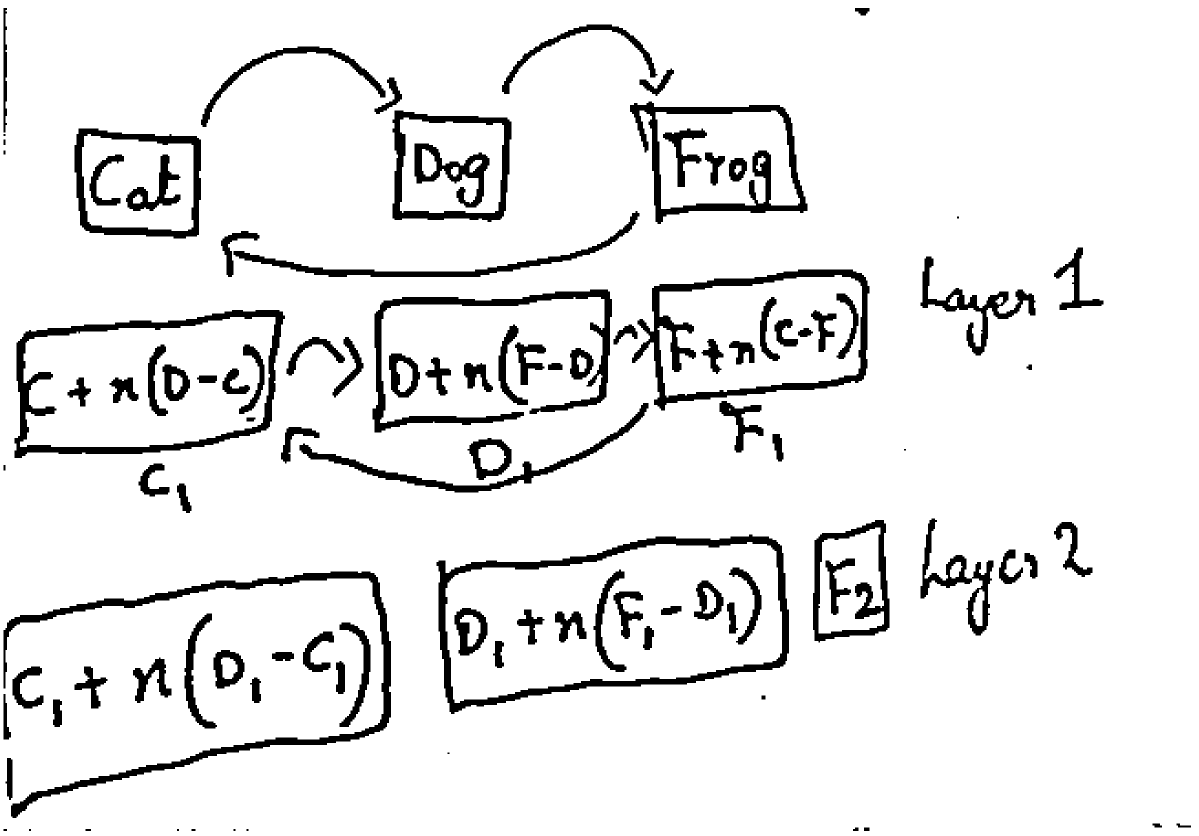
Note that in 3rd line, it's a typo, we don't choose a single random sample but random samples.

The main intuition of batchout is to push the decision boundary away from images making it more difficult to form adversaries.

There are two primary changes made to the algorithm:

1. Batchout will be performed to all samples unlike original algorithm where only k% of samples undergo.
2. Batchout will be mutual.

Let us consider a simple scenario where there are three images: Cat, Dog and Frog. Assume due to randomness the cat image is perturbed with dog, the dog with frog and frog with cat. In the 1st layer of batchout, we see normal convex combination. But in the 2nd layer, the perturbed image of cat also has some perturbations from frog class. This appears to be against intuition.



So the solution to this was thought to perturb the image of cat with dog and the dog with cat. The image of the frog will be perturbed with other image. Thus it is important for batchout to be mutual.

Using PyTorch, batchout can be applied by either:

1. Creating a new batchout class and change the forward class of model by adding the new class where we want. VGG models with simple forward function are best.
2. Using forward/pre-forward hooks. Models such as ResNet/DenseNet whose forward function is complex is best suitable.

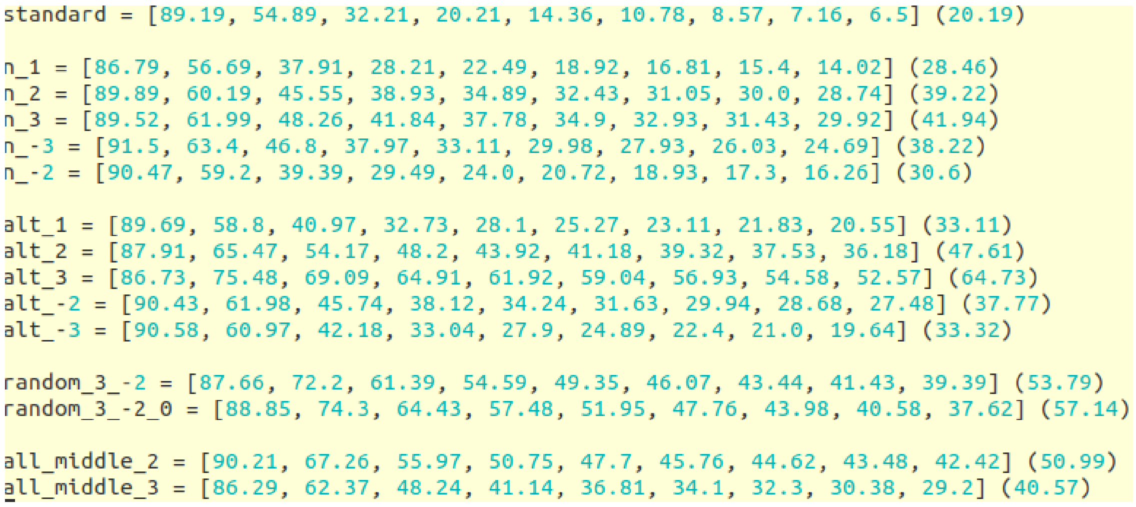
It is better not to have batchout on initial layers due to convergence issues. The depth after which batchout will be applied was based on results obtained from "Feature space perturbation yields more transferable adversarial examples" paper. Most of the ablation studies were done using VGG19 and CIFAR10. The major conclusions are:

Conclusions

- Applying batchout on all layers does not lead to convergence for large values of n such as 0.3. Thus it was decided to leave initial layers unperturbed.
- Applying batchout on alternative layers rather than all the layers gave better results.
- `n = Random[-2, 0, 3]` gave the best results. Here the values of n change for every batch. In the batchout paper, n = U[0, 0.15] was frequently used where U is uniform distribution.
- **The best PGD (200 steps) accuracy is 32.37%. The model gave 55.06% accuracy on PGD(10 steps). 37.62% FGSM accuracy.** However this model has very low accuracy 82.88%.
- Densenet and ResNet both of them failed to give any significant robust accuracies when batchout was applied on CIFAR10.

Few Points

Batchout is slightly different with mixup. We only take convex combination w.r.t images only. Intuitively in my opinion both are acceptable. Mixup tries to make smoother transition. Batchout tries to push the decision boundary away.

1. Usually value of n choosen was from 0.2 or 0.3. I applied negative values of -0.2 and -0.3 and the results were surprisingly good but not as good as 0.2 or 0.3. Why did I apply negative values? It was because of some misunderstanding why I was going through TRADES paper. Below we can see alt-*2 and alt*-3 gave 27% and 19% respectively, better than the standard model 6.5% at epsilon 8/255. The values in "[]" refers to accuracies from 0 to 8/255 epsilon and value in "()" is accuracy at l2 FGSM attack.


```
standard = [89.19, 54.89, 32.21, 20.21, 14.36, 10.78, 8.57, 7.16, 6.5] (20.19)

n_1 = [86.79, 56.69, 37.91, 28.21, 22.49, 18.92, 16.81, 15.4, 14.02] (28.46)
n_2 = [89.89, 60.19, 45.55, 38.93, 34.89, 32.43, 31.05, 30.0, 28.74] (39.22)
n_3 = [89.52, 61.99, 48.26, 41.84, 37.78, 34.9, 32.93, 31.43, 29.92] (41.94)
n_-3 = [91.5, 63.4, 46.8, 37.97, 33.11, 29.98, 27.93, 26.03, 24.69] (38.22)
n_-2 = [90.47, 59.2, 39.39, 29.49, 24.0, 20.72, 18.93, 17.3, 16.26] (30.6)

alt_1 = [89.69, 58.8, 40.97, 32.73, 28.1, 25.27, 23.11, 21.83, 20.55] (33.11)
alt_2 = [87.91, 65.47, 54.17, 48.2, 43.92, 41.18, 39.32, 37.53, 36.18] (47.61)
alt_3 = [86.73, 75.48, 69.09, 64.91, 61.92, 59.04, 56.93, 54.58, 52.57] (64.73)
alt_-2 = [90.43, 61.98, 45.74, 38.12, 34.24, 31.63, 29.94, 28.68, 27.48] (37.77)
alt_-3 = [90.58, 60.97, 42.18, 33.04, 27.9, 24.89, 22.4, 21.0, 19.64] (33.32)

random_3_-2 = [87.66, 72.2, 61.39, 54.59, 49.35, 46.07, 43.44, 41.43, 39.39] (53.79)
random_3_-2_0 = [88.85, 74.3, 64.43, 57.48, 51.95, 47.76, 43.98, 40.58, 37.62] (57.14)

all_middle_2 = [90.21, 67.26, 55.97, 50.75, 47.7, 45.76, 44.62, 43.48, 42.42] (50.99)
all_middle_3 = [86.29, 62.37, 48.24, 41.14, 36.81, 34.1, 32.3, 30.38, 29.2] (40.57)
```

   I tried a random combination of +3 and -2, but the results were somewhere in between as can be seen in random_3-*2 and random_3*-2_0.

2. I applied the batchout attack specifically on a pair of classes and implemented PGD targeted attack. The accuracy was poor here as well perhaps because our method is not strong with PGD.

3. The original implementation of batchout did not ensure that the two images that will be perturbed will be of same class. I applied batchout in both the cases, when it is ensured both classes are of same class and when it is not ensured. I see that strangely when it is not ensured, I achieve better accuracy(6% better accuracy). This can give some insight such as: **The linear space between images/features of same class is not necessarily classified as the class?** I tried to check whether mixup ensures both classes are different, but I didn't get any info.

4. Batchout was applied to all layers after a specific layer. As we can see in above figure, it didn't give the best result and also higher values of n gave poorer result.

5. Early stopping did give good results some time. But perhaps more experiments should be done to check whether this works (This is not related to batchout)

I used to think batchout or Mixup fail with PGD can perhaps be expalined by a paper Tramer et al(2017). I have written more about this later in other file. The gist is that Tramer et al finds around 25 directions in which Adversarial examples exist for MNIST. This number I think will be larger as dataset size increases. But Batchout or Mixup can at max consider 9 direction (each direction in the direction of other class) for CIFAR10. Perhaps this is the main reason why it is giving good (and sometimes better than PGD training) on FGSM but poor on PGD.

Papers and Resources Followed

https://adversarial-ml-tutorial.org/ for hands-on Introduction

Papers

Deep neural networks are easily fooled: High confidence predictions for unrecognizable images This paper uses Genetic algorithm to create images that fool deep nets to 99% accuracy. Nothing important but gives an idea that deep nets can be fooled to high accuracy.

Intriguing properties of neural networks The first paper to introduce Adversarial examples. Generates AE using some Newton Optimizer method

Explaining and harnessing adversarial examples Introduces FGSM and FGSM based AT

Universal adversarial perturbations

The papers from Madry's Lab and Zico Kolter's Lab are excellent and top rated.

Papers such as "Adversarial Examples Are Not Bugs, They Are Features", "PGD AT" from Madry's Lab are important, they also maintain a blog https://gradientscience.org/. "Fast is better than Free", "Certified Defence via Randomized Smoothing", "Overfitting in Adversarial Domain" papers from zico's lab are necessary. This paper however claims there are some shortcomings and other advantages of Fast is better than Free paper.

Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples

Adversarial Logit Pairing This paper from Goodfellow is very good but it's submission was retracted from NeurIPS (not sure why). The paper proposes a new method to make models robust. However a subsequent paper was able to break the defence.

TRADES One of the most important papers that breaks away from min-max optimisation technique. The paper is also very interesting to read.

Unlabeled Data improves AR showed great improvement in robustness.

The space of adversarial transferability I never actually read the paper completely because it is mathematically heavy. I feel something very important related to adversarial directions is discussed.

Auto Attack (ensemble of 4 attacks) is the most commonly used benchmark

SOTA no new algorithm is proposed. Just extensive experiments are performed and new best is obtained.

Youtube Videos:

https://www.youtube.com/watch?v=aK7DJ8XCWCI&t=447s Konda Reddy Sir's Introduction

https://www.youtube.com/watch?v=4rFOkpI0Lcg Intro to Adversarial Examples by Arxiv Insights

https://www.youtube.com/watch?v=df_NZyGeVXg&t=1s Lipschitz constant explaination from Intriguing properties of Neural Networks paper.

https://www.youtube.com/watch?v=CIfsB_EYsVI&t=1883s GoodFellow's Stanford Lecture wrote about it

https://www.youtube.com/watch?v=UHs2mGBH0Fg Zico Kolter's Lecture on Randomized Smoothing https://www.youtube.com/watch?v=fzusr-VdPxw&t=385s Talk from author on Bugs not Feature paper.

https://www.youtube.com/watch?v=hMO6rbMAPew Yannic Kilcher on Features not Bugs Paper