Résultats bruts

LM_ResNet20 without noise:

ϵ	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**
Accuracy	0.9212	0.2488	0.1796	0.1586	0.1468	0.1407	0.1354

Files : "Lost"

LM_ResNet20 with noise 0.1 (no ensemble):
Files : "{accuracies/examples}_noise_0_1.npy"

ϵ	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**
Accuracy	0.9091	0.2559	0.1856	0.1629	0.1499	0.1378	0.1332

LM_ResNet20 with noise 0.1 and ensemble 3
Files : "{accuracies/examples}_ensemble_3_noise_0_1.npy"

ϵ	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**
Accuracy	0.9155	0.2298	0.1638	0.1442	0.1371	0.1296	0.1247

LM*ResNet20 with* $\epsilon = 0.05$ *and ensemble* $N \in \{1, 3, 5, 10\}$
Files : "new_exp{accuracies/examples}_ensemble_3_noise_0_1.npy"

N	**1**	**3**	**5**	**10**
Accuracy	0.2574	0.2284	0.2237	0.2171

epsilon 0.05
ensemble 3, noise 0.25: 0.2329
ensemble 1, noise 0.5: 0.4376 ensemble 3, noise 0.5: 0.3143
ensemble 5, noise 0.5: 0.2604
ensemble 3, noise 1: 0.2693
ensemble 3, noise 2: 0.0886

epsilon 0, noise 0.5, ensemble 1: 0.8034

Interpretation

My Idea

Start with a LM-ResNet20 trained for 160 epochs without PGD ($\simeq$ adversarial) training nor noise injection.
Inject noise only in testing.

Impact of ϵ

ϵ	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**
No noise	0.9212	0.2488	0.1796	0.1586	0.1468	0.1407	0.135
Noise 0.1	0.9091	0.2559	0.1856	0.1629	0.1499	0.1378	0.133
Noise 0.1 ensemble 3	0.9155	0.2298	0.1638	0.1442	0.1371	0.1296	0.124

We focus on $\epsilon = 0.05$ which is very legitimate (see pictures) and responsible for the biggest drop.

Impact of the scale of the noise injected

The bigger, the more robust to attacks but the less accurate.
$\epsilon = 0.05$, $N = 3$ copies of LM-ResNet20 in the ensemble.

Noise	**0.25**	**0.5**	**1**
Accuracy	0.2329	0.3143	0.2693

Remark: can also perform stronger attacks... here we're just iterating once.

Impact of ensembling with noise

Scaling factor of the noise injected: 0.5

Number of copies of the net	**1**	**3**	**5**
Accuracy	0.4376	0.3143	0.2604

Scaling factor of the noise injected: 0.1

Number of copies of the net	**1**	**3**	**5**	**10**
Accuracy	0.2574	0.2284	0.2237	0.2171

What's the best noise to protect against $\epsilon = 0.05$?

Analysis

My expectation:
More nets $\implies$ closer to MonteCarlo solution of Feynman-Kac $\implies$
Truer diffusion process $\implies$ Better robustness
Why doesn't it work?

- Less randomness because we average
- ?

En$_N$LM-ResNet20

Training: PGD (IFGSM^{10}, step size 0.031 (=8/255, standard)) joint training with noise 0.1 injected in each residual.

Model	**Epoch**	**Details**	**Attack**	**Accuracy**
En$_1$LM-ResNet20	~50	X	No attack	0.8465
En$_1$LM-ResNet20	~50	X	0.05	0.6712

Model	Epoch	Details	Attack	Accuracy
En_2LM-ResNet20	89	X	0.05	0.7094
En_2LM-ResNet20	200	Training took >6 hours	0.031	0.7788

BUT with stronger benchmark attacks accuracy fall to 0.30.
At the moment I'm confused about the paper's evaluation of robustness. Seems like they are not using strong attacks
I want to reevaluate that, and to start writing my thesis. Both can be done in parallel, especially since long running experiments are sent on cloud servers.