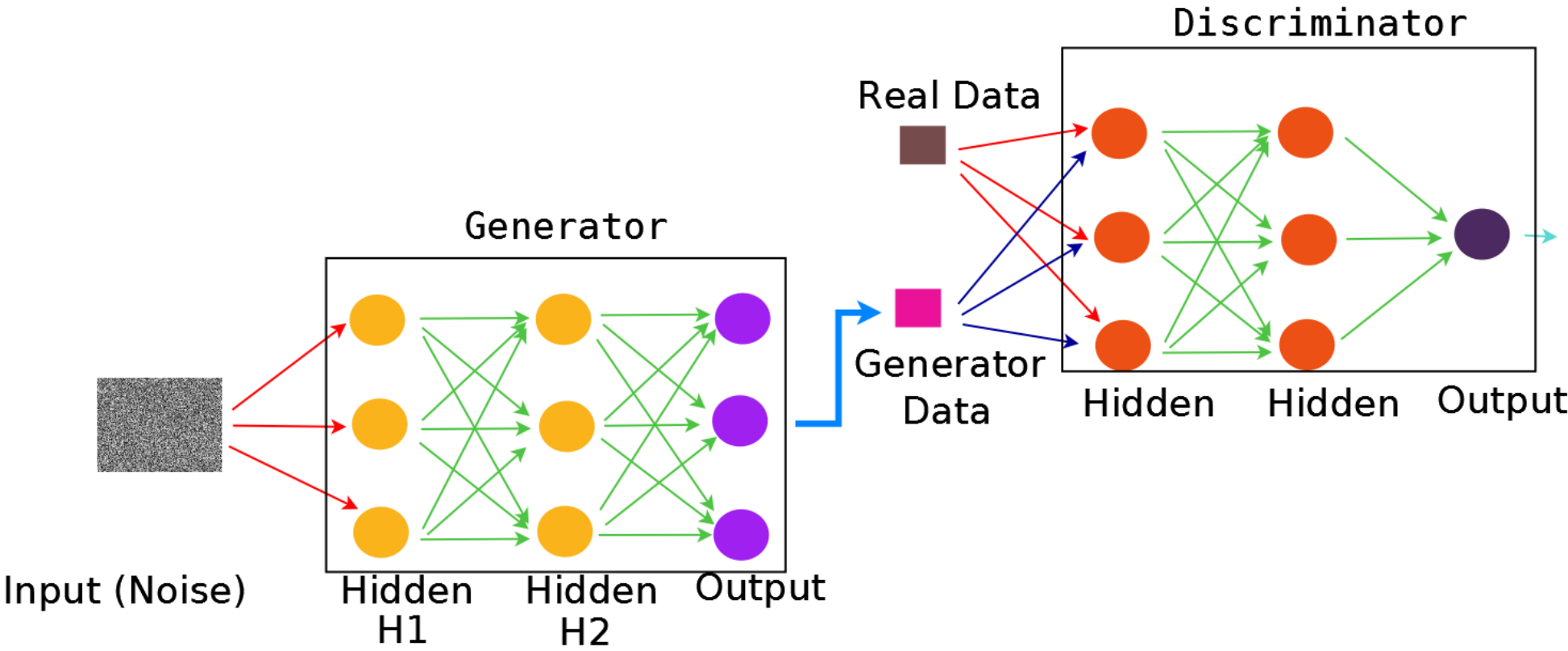
Adversarial Training

IE643 - Lecture 18

October 12, 2022.

GAN


Discriminator
Real Data
Generator
Data
Hidden
Hidden
Output
Generator
Input (Noise)
Hidden
H1
Hidden
H2
Output

GAN

- Discriminator component D
- Generator component G

GAN

- Aim of Discriminator component D:
  - Maximize likelihood of real data from distribution p_{data}
  - Minimize likelihood of fake data $G(z)$ obtained through generator G, where z comes from noise distribution p_{noise}.

GAN

- Objective of Discriminator component D: (straightforward)

$$\max_{\theta_d} \; \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G(z))).$$

Assumption: $D(x)$ is a probability (or likelihood)

GAN

- Aim of Generator component G:
  - Fool the discriminator by providing $G(z)$ as if it comes from p_{data}.
  - How to do that?

GAN

- Aim of Generator component G:
  - Fool the discriminator by providing $G(z)$ as if it comes from p_{data}.
  - How to do that?
- Idea:

$$\min_{\theta_g} \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))).$$

GAN

- Overall Objective of GAN:

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \min_{\theta_g} \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right].$$

GAN

- Overall Objective of GAN (seems to be):

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \min_{\theta_g} \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right].$$

- Looks like a max-min problem

GAN

- Overall Objective of GAN (used in paper):

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_{noise}} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right].$$

- This is a min-max problem

GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN

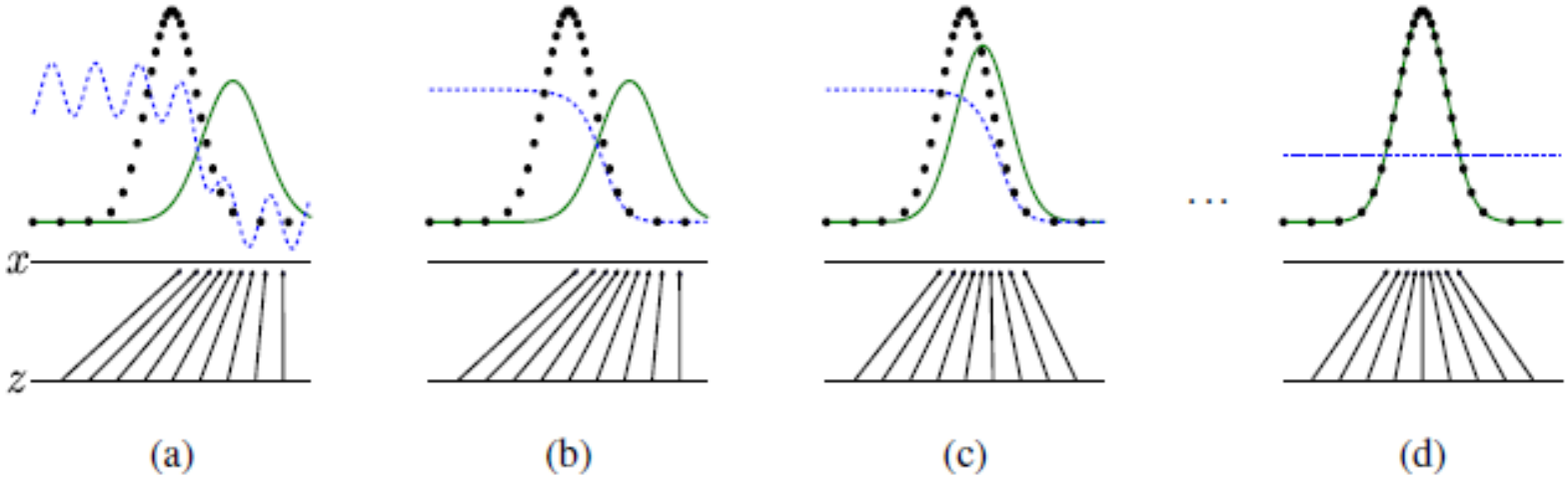


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

GAN

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

GAN

Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

GAN - Mode Collapse Problem

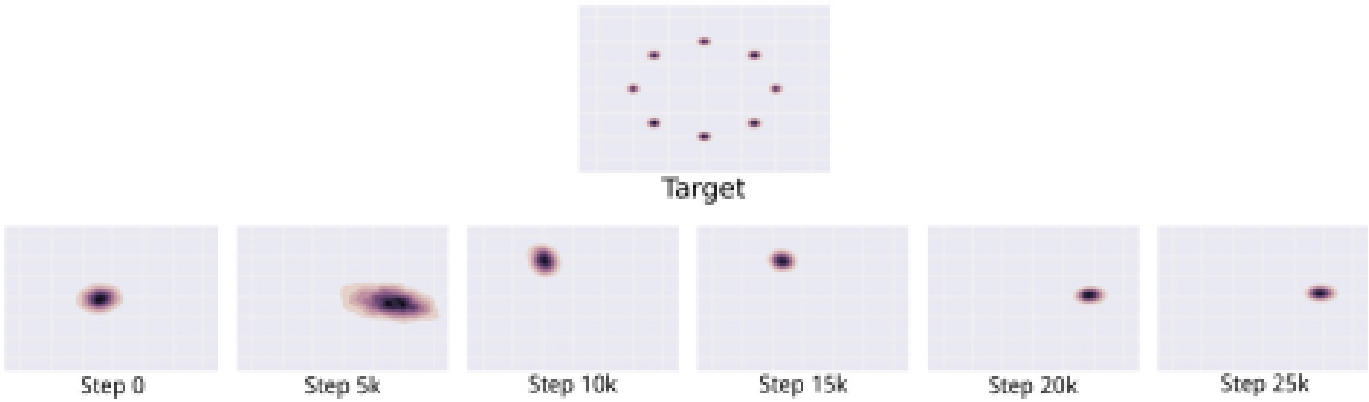


Figure 22: An illustration of the mode collapse problem on a two-dimensional toy dataset. In the top row, we see the target distribution p_{data} that the model should learn. It is a mixture of Gaussians in a two-dimensional space. In the lower row, we see a series of different distributions learned over time as the GAN is trained. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. Images from Metz *et al.* (2016).

Adversarial Training

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

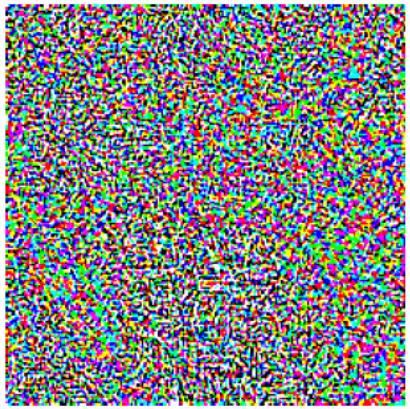
Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

Adversarial Training

x

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Note:

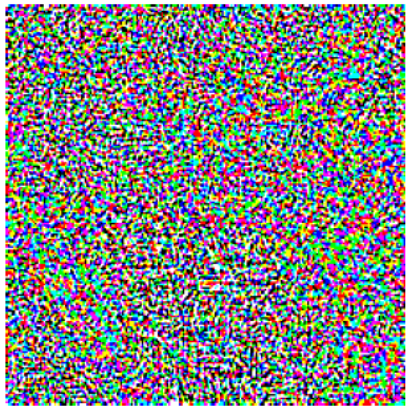
- $J(\boldsymbol{\theta}, \mathbf{x}, y)$ is the loss function.
- $\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)$ is the gradient of loss function with respect to the variations in input $\mathbf{x}$.

Adversarial Training

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

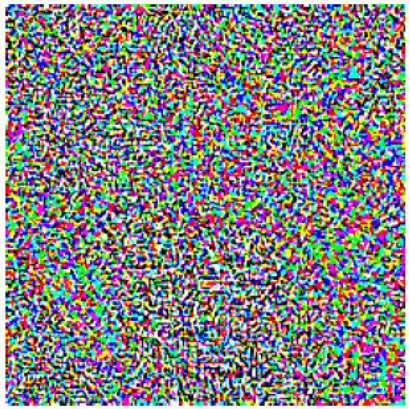
- Generating adversarial images: By using a perturbation $\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta} = \epsilon\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$.

Adversarial Training

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

- Generating adversarial images: By using a perturbation $\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta} = \epsilon\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$.
- This is called fast gradient sign method of generating adversarial examples.

Adversarial Training

- Adversarial training done using original images and adversarial images using an adversarial objective function:

$$\bar{J}(\boldsymbol{\theta}, \mathbf{x}, y) = \alpha J(\boldsymbol{\theta}, \mathbf{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)), y).$$

- $\alpha = 0.5$ was used in experiments.

Adversarial Training

- Adversarial training done using original images and adversarial images using an adversarial objective function:

$$\bar{J}(\boldsymbol{\theta}, \mathbf{x}, y) = \alpha J(\boldsymbol{\theta}, \mathbf{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)), y).$$

- $\alpha = 0.5$ was used in experiments.
- Adversarial training helped in better regularization.
- Adversarial training did not reach zero error rate on adversarial images.

Adversarial Training

- Adversarial training done using original images and adversarial images using an adversarial objective function:

$$\bar{J}(\boldsymbol{\theta}, \mathbf{x}, y) = \alpha J(\boldsymbol{\theta}, \mathbf{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y)), y).$$

- $\alpha = 0.5$ was used in experiments.
- Adversarial training helped in better regularization.
- Adversarial training did not reach zero error rate on adversarial images.
- Larger models with more units in hidden layers, early stopping on adversarial validation set error, helped to decrease error rate from 89.4% on adversarial examples to 17.9%.

Adversarial Training

Other Experiments:

- Control experiments were performed by adding random uniform noise $\pm\epsilon$ to each pixel of an image to generate adversarial samples.
- This model achieved 86.2% error rate on the adversarial examples created by adding random uniform noise.
- But the model achieved 90.4% error rate on adversarial examples created by fast gradient sign method.

Adversarial Training

Other Experiments:

- Control experiments were performed by generating adversarial examples using small rotations or by addition of scaled gradient.
- Such experiments lead to smooth objective functions.
- However training on such adversarial examples was found to be ineffective.

Adversarial Training

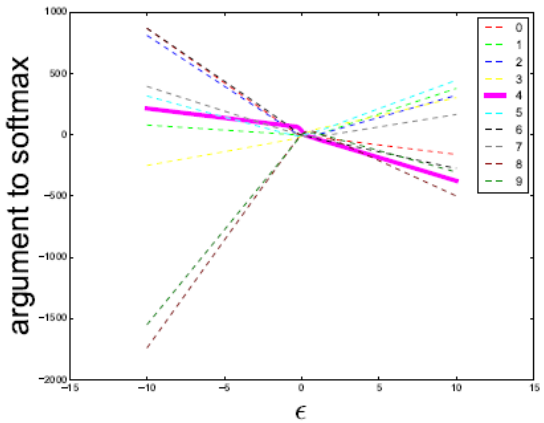
Other Experiments:

- Experiments were performed by perturbing the activations of hidden layers (but not the output layer).
- When hidden layers have sigmoid activations, perturbing hidden layer activations helped to improve regularization.
- When hidden layer activations have ReLU type activation functions, the perturbations did not give much improvements.
- Exercise: Check why last layer was not considered for perturbations!

Adversarial Training

Observations:

- An adversarial example generated for one model is misclassified by other models too.



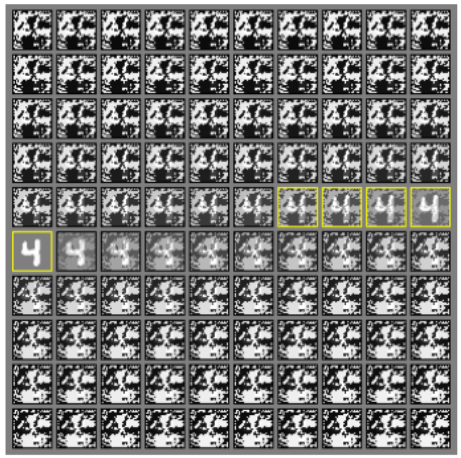


Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where $\boldsymbol{x}$ occurs in the data. Most of $\mathbb{R}^n$ consists of adversarial examples and *rubbish class examples* (see the appendix). This plot was made from a naively trained maxout network. Left) A plot showing the argument to the softmax layer for each of the 10 MNIST classes as we vary ϵ on a single input example. The correct class is 4. We see that the unnormalized log probabilities for each class are conspicuously piecewise linear with ϵ and that the wrong classifications are stable across a wide region of ϵ values. Moreover, the predictions become very extreme as we increase ϵ enough to move into the regime of rubbish inputs. Right) The inputs used to generate the curve (upper left = negative ϵ, lower right = positive ϵ, yellow boxes indicate correctly classified inputs).

Adversarial Training

Observations:

- When other models misclassify an adversarial example, the class labels given by different models are almost the same.
- One hypothesis to validate this observation: most neural networks try to approximate to a reference linear model on the training set.

Adversarial Training

Optimization problem:

- Adversarial training can be posed as the following training problem:

$$\min_{\theta} \sum_{i=1}^{N} \max_{\delta \in \Delta} \ell(f_\theta(x^i + \delta), y^i)$$

where $\Delta = \{\vartheta : \|\vartheta\|_\infty \leq \epsilon\}$ for some fixed $\epsilon > 0$.

Adversarial Training

Optimization problem:

- Adversarial training can be posed as the following training problem:

$$\min_{\theta} \sum_{i=1}^{N} \max_{\delta \in \Delta} \ell(f_\theta(x^i + \delta), y^i)$$

where $\Delta = \{\vartheta : \|\vartheta\|_\infty \leq \epsilon\}$ for some fixed $\epsilon > 0$.

Note: FGSM can be seen to consider an inaccurate approximate solution to the inner problem as $\delta^* = \epsilon \text{sign}(\nabla_x \ell(f(x), y))$.

Adversarial Training

Towards Deep Learning Models Resistant to Adversarial Attacks

Aleksander Mądry*
MIT
madry@mit.edu

Aleksandar Makelov*
MIT
amakelov@mit.edu

Ludwig Schmidt*
MIT
ludwigs@mit.edu

Dimitris Tsipras*
MIT
tsipras@mit.edu

Adrian Vladu*
MIT
avladu@mit.edu

Adversarial Training

PGD Method:

Algorithm 1 PGD adversarial training for T epochs, given some radius ϵ, adversarial step size α and N PGD steps and a dataset of size M for a network f_θ

for $t = 1 \dots T$ **do**
  for $i = 1 \dots M$ **do**
    // Perform PGD adversarial attack
    $\delta = 0$ *// or randomly initialized*
    for $j = 1 \dots N$ **do**
      $\delta = \delta + \alpha \cdot \text{sign}(\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$
      $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
    end for
    $\theta = \theta - \nabla_\theta \ell(f_\theta(x_i + \delta), y_i)$ *// Update model weights with some optimizer, e.g. SGD*
  end for
end for

Adversarial Training

Optimization problem:

- Adversarial training can be posed as the following training problem:

$$\min_{\theta} \sum_{i=1}^{N} \max_{\delta \in \Delta} \ell(f_\theta(x^i + \delta), y^i)$$

  where $\Delta = \{\vartheta : \|\vartheta\|_\infty \leq \epsilon\}$ for some fixed $\epsilon > 0$.

- Landscape of local maxima of the inner maximization problem was investigated to check the behavior.
- Done by restarting PGD from different points in the ℓ_∞ balls around the data points.
- Existence of multiple maxima; in each case, the loss plateaus quickly.
- Exercise: How to check for distinct maxima?

Adversarial Training

Loss values for different restarts:

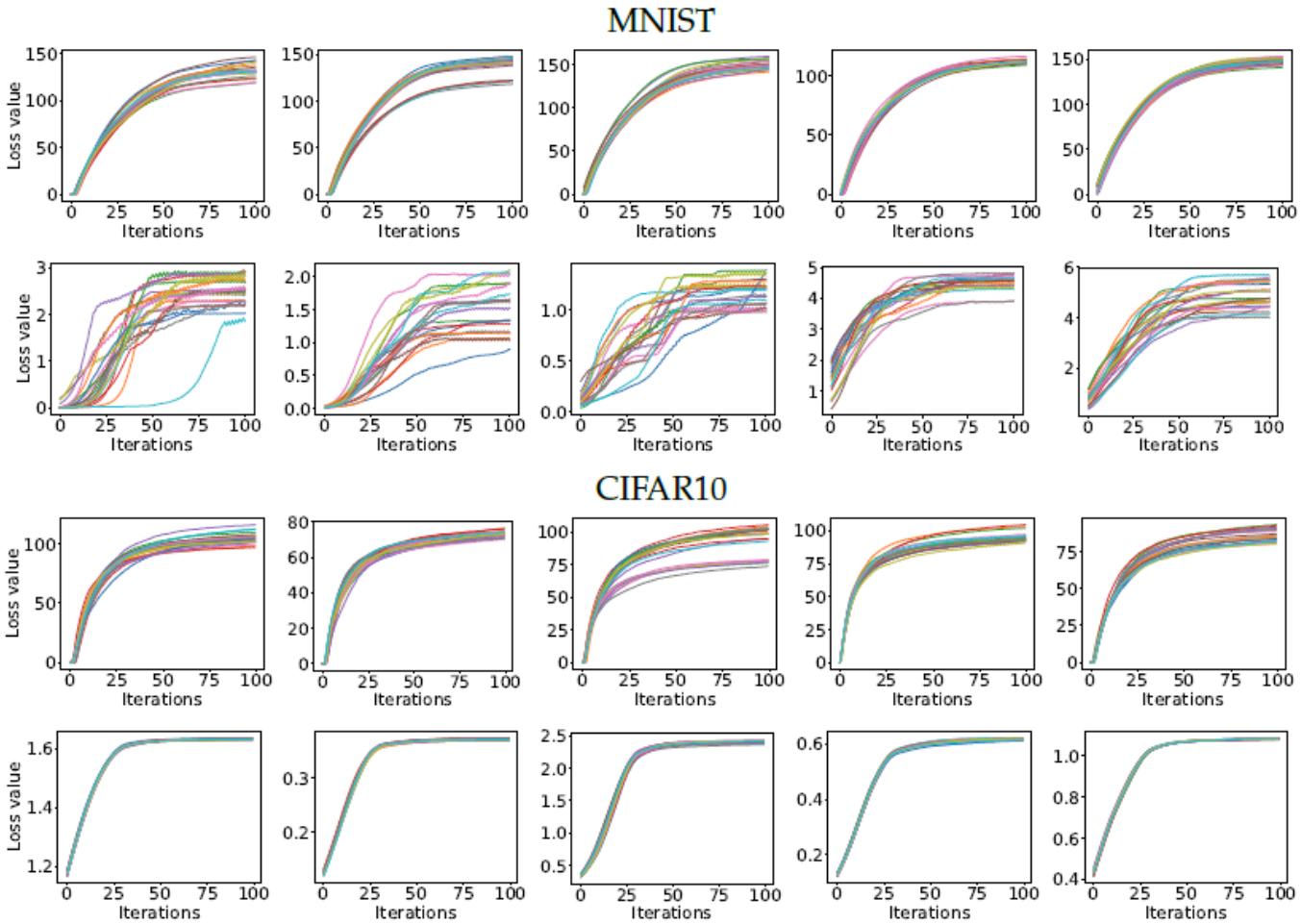


Figure : Loss function value over PGD iterations for 20 random restarts on random examples. The 1st and 3rd rows correspond to standard networks, while the 2nd and 4th to adversarially trained ones.

Adversarial Training

Loss value concentration over multiple restarts:

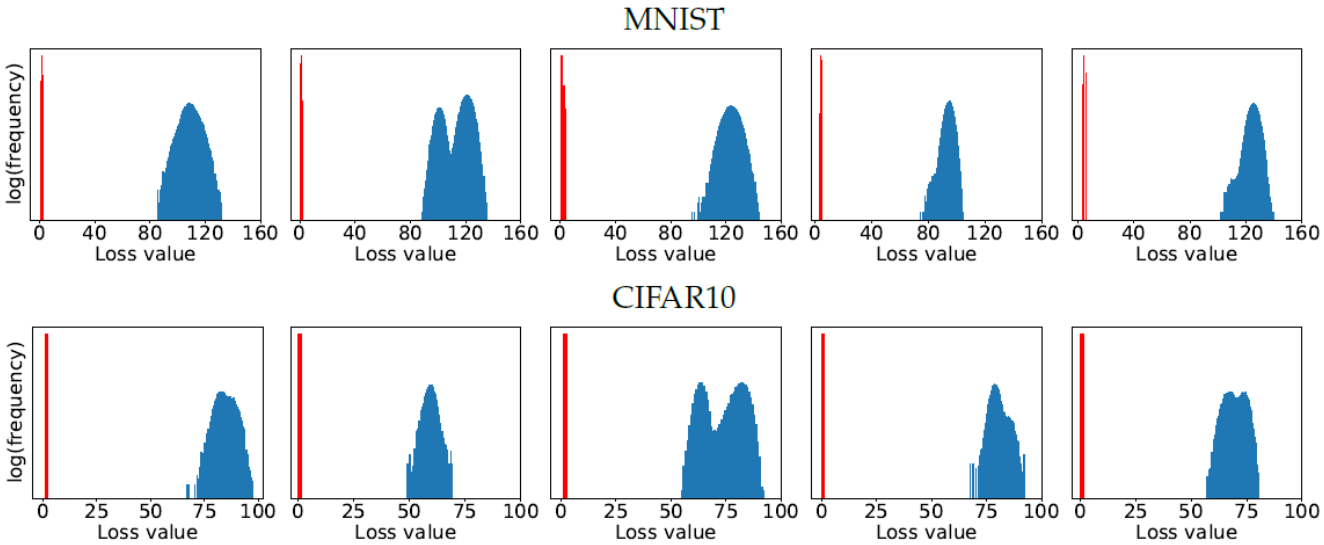
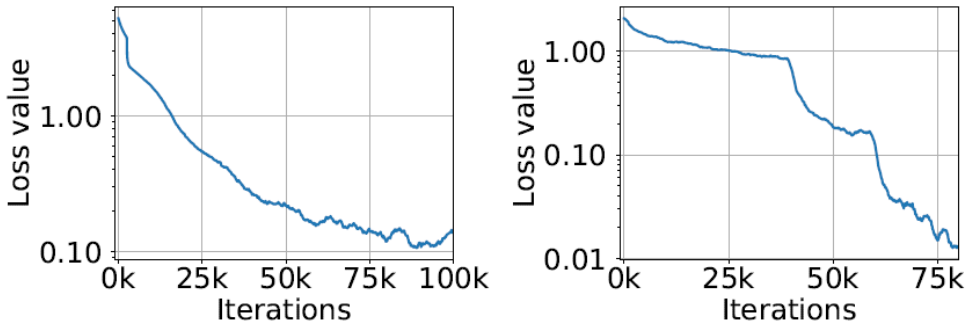


Figure : Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, we start projected gradient descent (PGD) from 10^5 uniformly random points in the ℓ_∞-ball around the example and iterate PGD until the loss plateaus. The blue histogram corresponds to the loss on a standard network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

The average loss of final iterate over 10^5 restarts follows a concentrated distribution without extreme outliers.

Adversarial Training

Loss behavior on adversarial examples:



(a) MNIST (b) CIFAR10

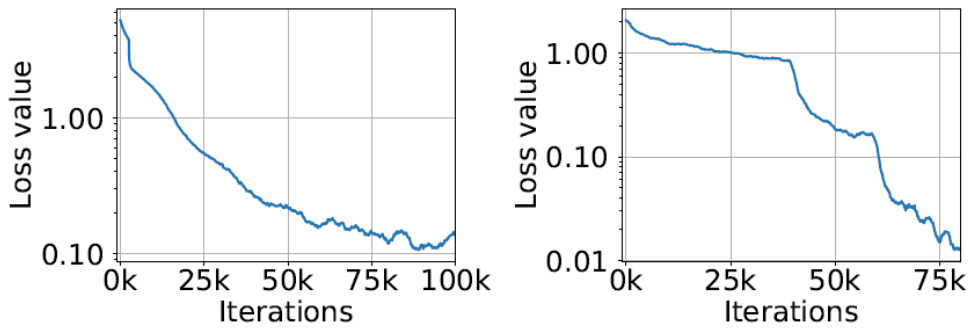
Figure : Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

MNIST:

- Training was done on CNN with two conv layers with 32 and 64 filters respectively, followed by 2×2 max-pooling and fully connected layer of size 1024.
- $\epsilon = 0.3$
- 40 iterations of PGD with step size 0.01.

Adversarial Training

Loss behavior on adversarial examples:



(a) MNIST (b) CIFAR10

Figure : Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (2.1), thus producing an increasingly robust classifier.

CIFAR:

- Training was done on Resnet.
- $\epsilon = 8$
- 40 iterations of PGD with step size 0.02.

Adversarial Training

Types of adversarial attakcs:

- Black box attacks: when the adversary has no knowledge of the architecture used for training.
- White box attacks: when the adversary has complete knowledge of the architecture used for training.

Adversarial Training

Types of adversarial examples:

- Non-targeted: add perturbation to an image so that the network predicts a different class label other than the original class label.
- Targeted: add perturbation to an image so that the network predicts a particular class label of the adversary's choice different from the original class label.