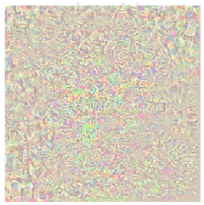
Deep Learning

Adversarial Examples

"golden retriever"
80.6 % confidence



added perturbation
(zoomed in by a factor of 10)

"plane"
99 % confidence

Learning goals

- adversarial robustness
- adversarial examples
- adversarial training
- projected gradient descent
- fast gradient sign method

ADVERSARIAL ROBUSTNESS

- It is critical to examine if a trained neural net is robust and reliable.
- **Adversarial robustness** of a model means that a model is robust to (test time) perturbations of its inputs.
- **Adversarial machine learning** studies techniques which attempt to fool machine learning models through malicious input.
- To make a model more robust, we can train our model on adversarially perturbed examples, called **adversarial examples**, derived from the training set. This is called **adversarial training**.
- This chapter summarizes high-level ideas in adversarial robustness with a particular emphasis on adversarial examples.
- For a deeper dive, ▸ click here.

Adversarial Examples

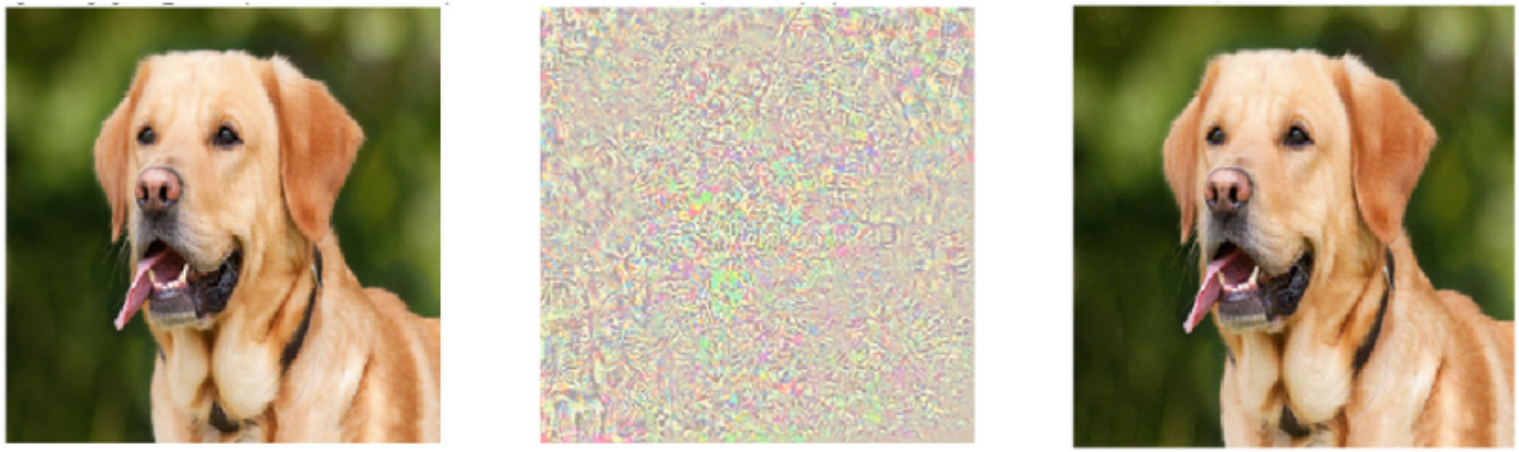
ADVERSARIAL EXAMPLES

- An adversarial example is an input to a model that is deliberately designed to "fool" the model into misclassifying it.
- The test error of a model is only an indicator of how well the model performs with respect to samples from the data-generating distribution.
- The performance of the same model can be drastically different on samples from a completely different distribution (on the same input space).
- It is possible to make changes to an image that makes a pretrained CNN (for example) output a completely different predicted class even though the change is imperceptible to the human eye.
- These examples suggest that even models that have very good test set performance do not have a human understanding of the underlying concepts that determine the correct output label.

ADVERSARIAL EXAMPLES

- Adversarial examples are *not* unique to deep neural nets. Many other models (such as logistic regression) are also susceptible to them.
- They pose serious security concerns in many areas.
- Example: Fooling autonomous cars into thinking that a stop sign is a 45 km/h sign.
- Example: Evading law enforcement by fooling facial recognition systems into misidentifying individuals.

ADVERSARIAL EXAMPLES



"golden retriever"
80.6 % confidence

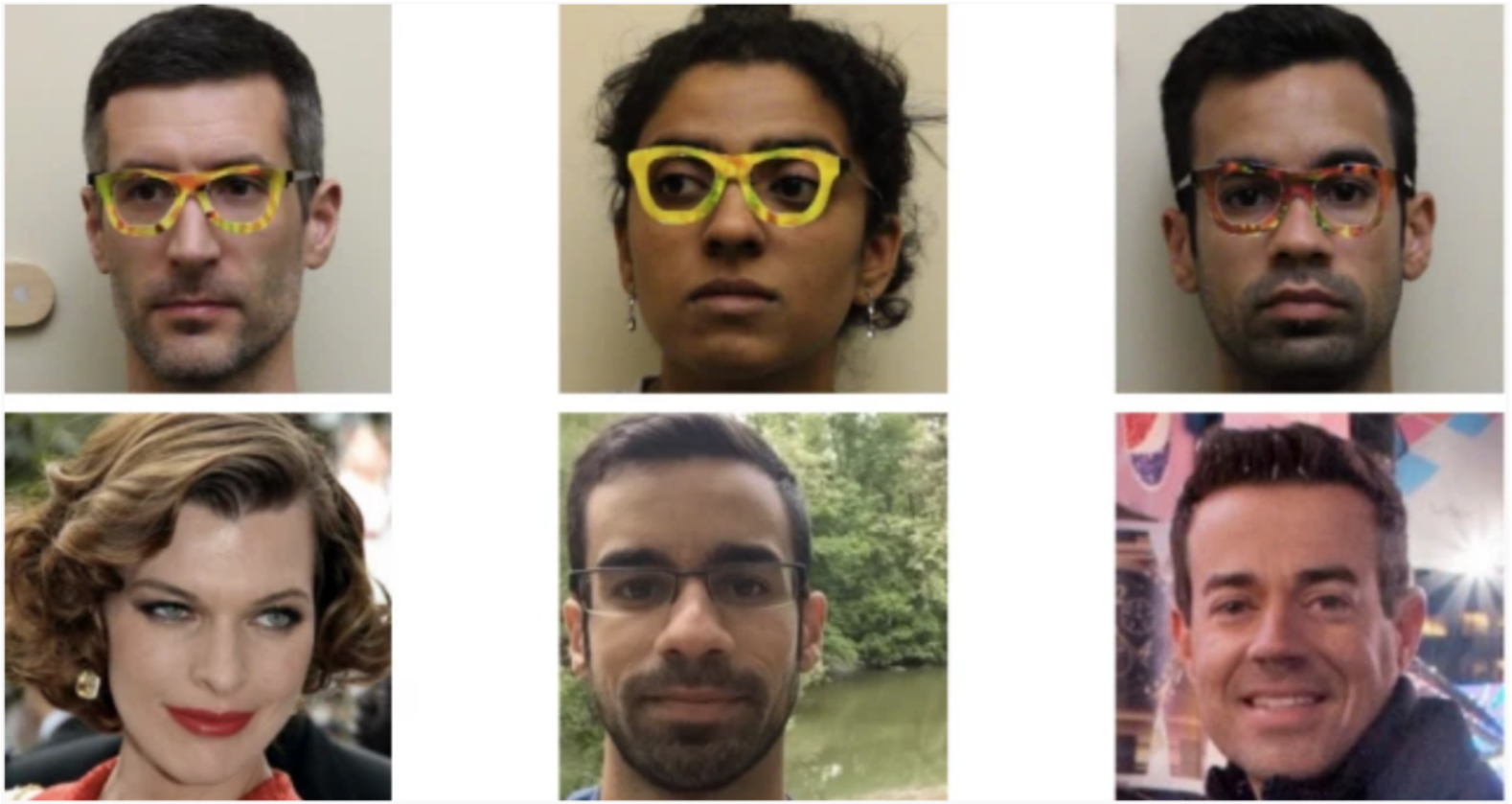
added perturbation
(zoomed in by a factor of 10)

"plane"
99 % confidence

Credit: Maxence Prevost

Figure: The difference between the left and the right golden retriever is imperceptible to humans. The last image was classified as a plane by ResNet50 with more than 99% confidence. The only difference between the left and right image are small pixel perturbations, shown in the second picture.

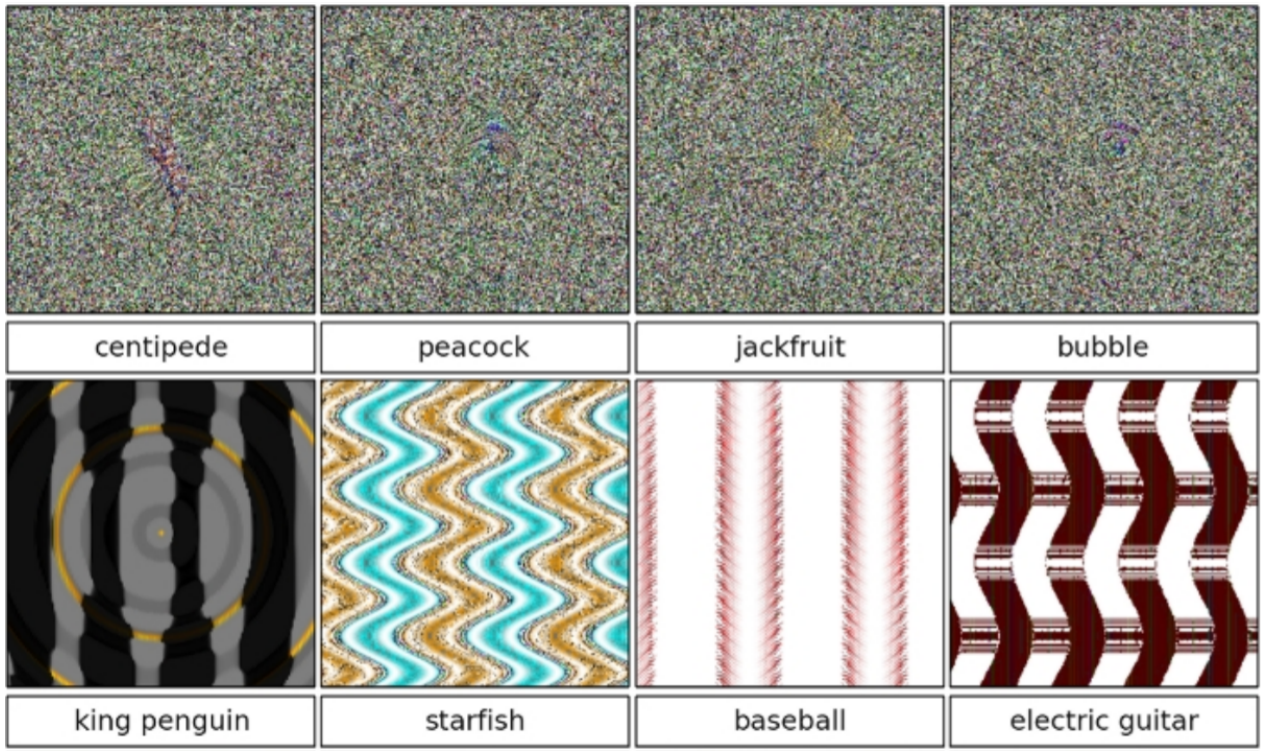
ADVERSARIAL EXAMPLES



Credit: Sharif et al.

Figure: A CNN misidentified each person in the top row (with the funky looking "adversarial" glasses) as the one in the corresponding position in the bottom row. The generated images do often contain some features of the target class.

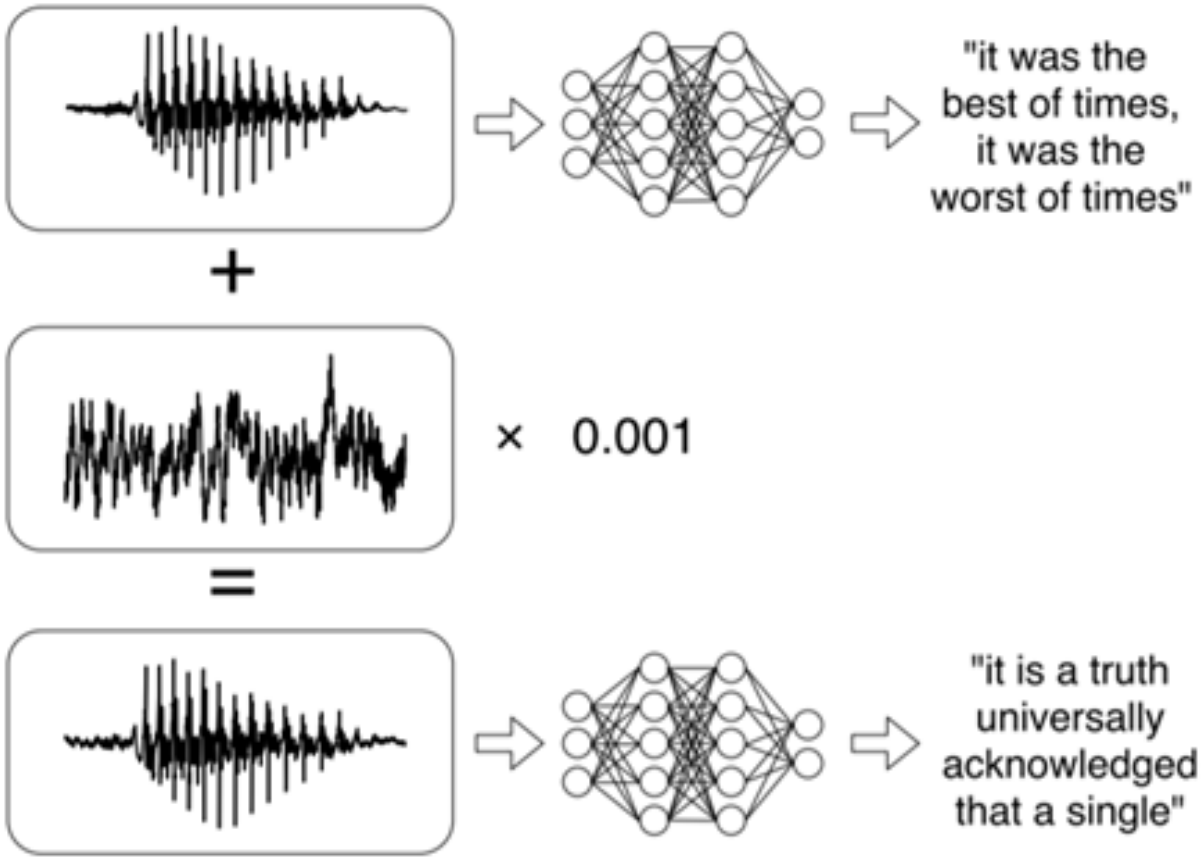
ADVERSARIAL EXAMPLES



Credit: Nguyen et al.

Figure: All 8 images above are unrecognizable to humans but are misclassified by a CNN with higher than 99% confidence. The CNN was trained by Krizhevsky et al. on the ImageNet dataset and consists of five convolutional layers, some of which are followed by max-pooling layers,and three fully-connected layers with a final softmax.

ADVERSARIAL EXAMPLES



Credit: Carlini et al

Figure: It is possible to add a small perturbation to any waveform in order to fool a speech-to-text neural network into transcribing it as any desired target phrase.

CREATION OF ADVERSARIAL EXAMPLES

- In the examples earlier, we saw that adversarial examples can seem recognizable to humans or seem like random noise/patterns.
- In the following, given a datapoint $\mathbf{x}$, we want to create an adversarial example $\tilde{\mathbf{x}}$ that is very similar to $\mathbf{x}$.
- Specifically, our goal is to find an input $\tilde{\mathbf{x}}$ close to the datapoint $\mathbf{x}$ such that a pretrained model (which accurately classifies $\mathbf{x}$), ends up misclassifying $\tilde{\mathbf{x}}$.
- When we train a neural network, we typically want to optimize the parameter θ, so that we minimize the loss.
- By contrast, to find an adversarial example $\tilde{\mathbf{x}}$ that is in the vicinity of $\mathbf{x}$, we want to optimize the *input* to *maximize* the loss.

CREATION OF ADVERSARIAL EXAMPLES

- To ensure that $\tilde{\mathbf{x}}$ is close to $\mathbf{x}$, we optimize over the perturbation of $\mathbf{x}$, denoted as $\boldsymbol{\delta}$, and define a feasible set of perturbations Δ.

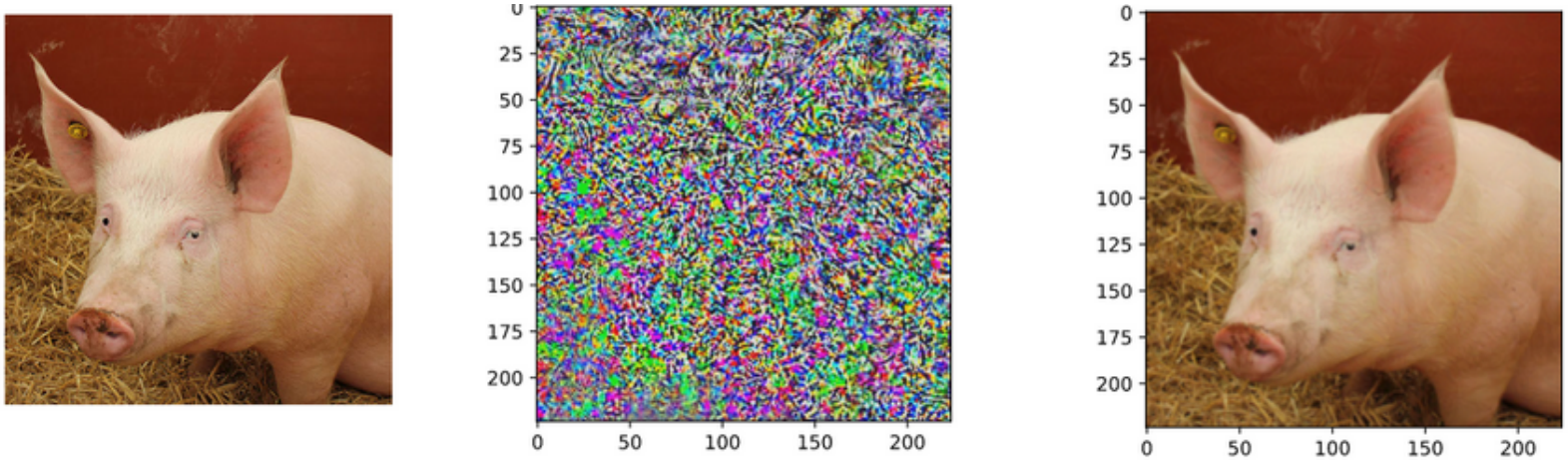
$$\arg\max_{\boldsymbol{\delta} \in \Delta} L(y, \hat{f}(\mathbf{x} + \boldsymbol{\delta} | \boldsymbol{\theta}))$$

- A common perturbation set is $\mathcal{B}_\epsilon^\infty$ which is the ϵ-ball measured by $\ell_\infty = \| \cdot \|_\infty$

$$\Delta = \mathcal{B}_\epsilon^\infty(\boldsymbol{\delta}) = \{\boldsymbol{\delta} : ||\boldsymbol{\delta}||_\infty \leq \epsilon\} \text{ with } ||\boldsymbol{\delta}||_\infty = \max_i |\delta_i|$$

  It allows each component of the perturbation $\boldsymbol{\delta}$ to lie between $-\epsilon$ and $+\epsilon$.
- In general, Δ can also depend on the input datapoint $\mathbf{x}$, denoted as $\Delta(\mathbf{x})$.

EXAMPLE: RESNET50



Credit: Kolter & Madry

Figure: Adversarial example for one datapoint of the ImageNet dataset and pre-trained ResNet50. By adding an imperceptibly small perturbation to the original image, an image was created that looks identical to our original image, but is misclassified.

TARGETED ATTACKS

- It is also possible to generate adversarial examples classified virtually as any desired class. This is known as a **targeted attack**.
- The only difference is that, instead of trying to just maximize the loss of the correct class, we maximize the loss of the correct class while also minimizing the loss of a target class y_{target}.

$$\arg\max_{\boldsymbol{\delta} \in \Delta} (L(y, \hat{f}(\mathbf{x} + \boldsymbol{\delta} | \boldsymbol{\theta})) - L(y_{target}, \hat{f}(\mathbf{x} + \boldsymbol{\delta} | \boldsymbol{\theta})))$$

Adversarial Training

ADVERSARIAL TRAINING

- To modify a trained model so that it is more resistant to such attacks, adversarial training can be performed.

- To do so, we minimize the **empirical adversarial risk** which measures the worst-case empirical loss of a model, if we are able to manipulate every input $\mathbf{x}$ in the training data set within the feasible set $\Delta(\mathbf{x})$:

$$\min_{\boldsymbol{\theta}} \mathcal{R}_{adv}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \max_{\boldsymbol{\delta} \in \Delta(\mathbf{x})} L(y^{(i)}, f(\mathbf{x}^{(i)} + \boldsymbol{\delta} | \boldsymbol{\theta}))$$

ADVERSARIAL TRAINING

- To solve the optimization problem, we use SGD over $\boldsymbol{\theta}$. In each SGD step $t \in \{1, 2, \ldots\}$ we repeatedly choose a minibatch of size m and repeat the following until a stopping criterion is met:
  1. For each $(\mathbf{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, we compute an adversarial example

$$\boldsymbol{\delta}^*(\mathbf{x}^{(i)}) = \underset{\boldsymbol{\delta} \in \Delta(\mathbf{x}^{(i)})}{\arg\max}\, L(y^{(i)}, f(\mathbf{x}^{(i)} + \boldsymbol{\delta} | \boldsymbol{\theta}^{[t]}))$$

  2. Then we compute the gradient of the empirical adversarial risk given $\boldsymbol{\delta}^* = (\boldsymbol{\delta}^*(\mathbf{x}^{(1)}), \ldots, \boldsymbol{\delta}^*(\mathbf{x}^{(m)}))$ and update $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{[t+1]} := \boldsymbol{\theta}^{[t]} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(y^{(i)}, f(\mathbf{x}^{(i)} + \boldsymbol{\delta}^*(\mathbf{x}^{(i)}) | \boldsymbol{\theta}^{[t]}))$$

- The first step is derived from Danskin's theorem, which states that the gradient of the inner function (maximization term) is simply given by the gradient of the function evaluated at its maximum.

Linear Models

LINEAR MODELS

- In case of linear models, the inner maximization problem can be solved exactly. We show this in the case of binary classification using linear models.
- Recall, the hypothesis space for logistic regression consists of models of the form:

$$\mathcal{H} = \left\{ f : \mathbb{R}^p \to [0,1] \;\middle|\; f(\mathbf{x}) = \tau\left(\sum_{j=1}^{p} \theta_j x_j + \theta_0\right), \boldsymbol{\theta} \in \mathbb{R}^p, \theta_0 \in \mathbb{R} \right\},$$

  where $\tau(z) = (1 + \exp(-z))^{-1}$ is the logistic sigmoid function.
- For class labels $y \in \{+1, -1\}$, the logistic loss is:

$$L(y, f(\mathbf{x} \mid \boldsymbol{\theta})) = \log(1 + \exp(-y(\underbrace{\sum_{j=1}^{p} \theta_j x_j}_{\boldsymbol{\theta}^T \mathbf{x}} + \theta_0))) \equiv \Psi(y(\boldsymbol{\theta}^T \mathbf{x} + \theta_0))$$

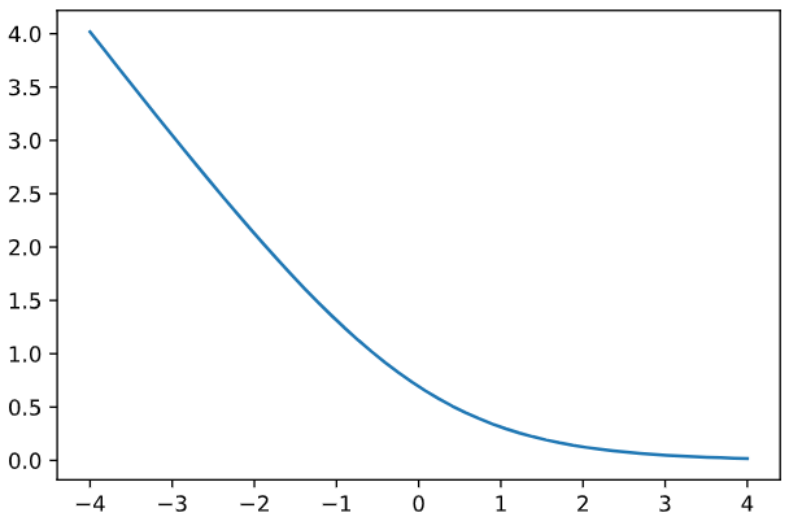
  where we define $\Psi(z) = \log(1 + \exp(-z))$.

LINEAR MODELS

- The inner maximization in the adversarial risk, which we saw earlier, can be written as:

$$\max_{\delta \in \Delta(\mathbf{x})} L(y, f(\mathbf{x} + \boldsymbol{\delta} | \boldsymbol{\theta})) = \max_{\delta \in \Delta(\mathbf{x})} \Psi(y(\boldsymbol{\theta}^T(\mathbf{x} + \boldsymbol{\delta}) + \theta_0))$$

- In this particular case, it is possible to solve the inner maximization exactly.
- First, note that Ψ is a monotonically decreasing function.



Credit: Kolter and Madry

LINEAR MODELS

- Maximizing such a monotonically decreasing function is equivalent to minimizing the argument.

- Therefore

$$\max_{\boldsymbol{\delta} \in \Delta(\mathbf{x})} \Psi(y(\boldsymbol{\theta}^T(\mathbf{x}+\boldsymbol{\delta}) + \theta_0)) = \Psi(\min_{\boldsymbol{\delta} \in \Delta(\mathbf{x})} y(\boldsymbol{\theta}^T(\mathbf{x}+\boldsymbol{\delta}) + \theta_0))$$
$$= \Psi(y(\boldsymbol{\theta}^T\mathbf{x} + \theta_0) + \min_{\boldsymbol{\delta} \in \Delta(\mathbf{x})} y(\boldsymbol{\theta}^T\boldsymbol{\delta}))$$

- We have to solve the problem

$$\min_{\boldsymbol{\delta} \in \Delta} y(\boldsymbol{\theta}^T\boldsymbol{\delta})$$

LINEAR MODELS

- To get a feel for the problem, let us consider the case where $y = +1$ and use $\Delta = \mathcal{B}_\epsilon^\infty$. The latter constrains each element of δ to lie between $-\epsilon$ and $+\epsilon$.

- The quantity $y(\boldsymbol{\theta}^T \boldsymbol{\delta})$ is then minimized when $\delta_j = -\epsilon$ for $\theta_j \geq 0$ and $\delta_j = \epsilon$ for $\theta_j < 0$.

- For $y = -1$, the signs would be flipped.

- The optimal solution then, is

$$\boldsymbol{\delta}^* = -y\epsilon \cdot \text{sign}(\boldsymbol{\theta})$$

- Note that the optimal solution does not explicitly depend on $\mathbf{x}$.

LINEAR MODELS

- The function value achieved by the solution is:

$$y \cdot \boldsymbol{\theta}^T \boldsymbol{\delta}^* = y \cdot \sum_j -y\epsilon \cdot \text{sign}(\theta_j)\theta_j = -y^2\epsilon \sum_j |\theta_j| = -\epsilon\|\boldsymbol{\theta}\|_1$$

- Therefore, we have analytically computed the solution to the inner maximization problem! The solution is:

$$\max_{\boldsymbol{\delta} \in \Delta(\mathbf{x})} \Psi(y(\boldsymbol{\theta}^T(\mathbf{x} + \boldsymbol{\delta}) + \theta_0)) = \Psi(y(\boldsymbol{\theta}^T(\mathbf{x} + \boldsymbol{\delta})) - \epsilon\|\boldsymbol{\theta}\|_1)$$

- As a result, the adversarial risk, which was a min-max problem, has now been converted to a pure minimization problem:

$$\min_{\boldsymbol{\theta}, \theta_0} \frac{1}{N} \sum_{i=1}^{N} \Psi\left(y^{(i)} \cdot \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} + \theta_0\right) - \epsilon\|\boldsymbol{\theta}\|_1\right)$$

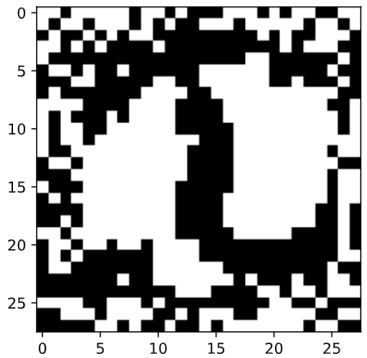
- This problem is convex in $\{\boldsymbol{\theta}, \theta_0\}$ and can be solved exactly. An iterative optimizer such as SGD will also approach the global minimum.

MNIST EXAMPLE

- As an example, we look at the MNIST dataset, but this time we perform logistic regression and focus only on the classification of 0s vs. 1s.
- The logistic regression classifier was trained for 10 epochs with SGD on the training set.
- This model obtained a low misclassification rate of 0.0004 on the test set.
- To generate adversarial examples, Δ is defined as $\mathcal{B}_{0.2}^{\infty}$.
- As we saw earlier, the optimal perturbation $\boldsymbol{\delta}^*$ is $-y\epsilon \cdot \text{sign}(\boldsymbol{\theta})$.

MNIST EXAMPLE

- As $\boldsymbol{\delta}^*$ does not directly depend on $\mathbf{x}$, it is the "same" (ignoring the value of label y) across all examples. This is what it looks like:

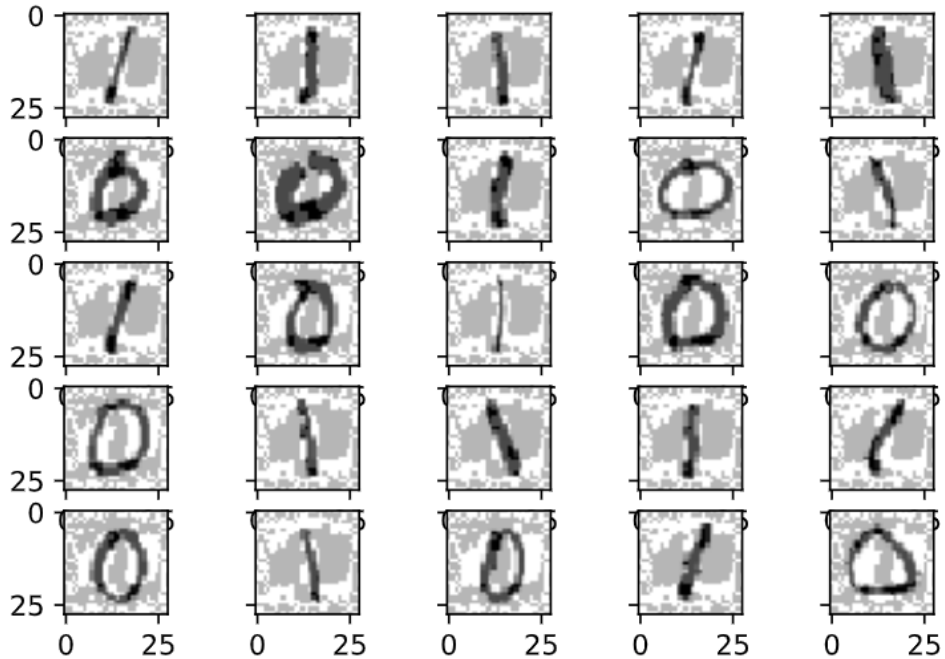


Credit: Kolter and Madry

Figure: The optimal perturbation for images that contain 0. For images that contain 1, the signs would be flipped. (The contrast between the black and white pixels is amplified for the sake of visualization.)

- The perturbation (*vaguely*) has a vertical line (like a 1) in black pixels, and a circle (like a 0) in white pixels. Intuition: When a given image is moved (translated) in the black direction, it is more likely to be classified as 1, whereas when moved in the white direction, it is more likely to be classified as 0.

MNIST EXAMPLE



Credit: Kolter and Madry

Figure: Perturbed images from the test set.

- When all the images in test set are perturbed, the misclassification error of the model jumps from 0.0004 to 0.845!
- Interestingly, when the model is trained on similarly perturbed images from the training set (that is, the empirical adversarial risk is minimized), the misclassification error on the perturbed test set drops to 0.025.

Neural Networks

PROJECTED GRADIENT DESCENT

- In contrast to logistic regression, neural networks can have a bumpier, non-convex loss surface.
- As a consequence, Danskin's theorem does not longer hold and the inner optimization problem must be solved approximately.
- One approximation method is projected gradient descent (PGD)[1].
- Let $\hat{f}$ be the pretrained model, $\mathbf{x}$ the input to the model, y the target and $L\left(y, \hat{f}(\mathbf{x}|\boldsymbol{\theta})\right)$ the loss function used to train the network.
- In each gradient descent step, the basic PGD algorithm updates $\boldsymbol{\delta}$ as:

$$\boldsymbol{\delta}^{[t+1]} = \mathcal{P}\left(\boldsymbol{\delta}^{[t]} + \alpha \nabla_{\boldsymbol{\delta}} L\left(y, \hat{f}(\mathbf{x} + \boldsymbol{\delta}^{[t]}|\boldsymbol{\theta})\right)\right)$$

  where $\mathcal{P}$ denotes the projection onto the ball of interest.

[1] Technically speaking, it is gradient *ascent* since we are maximizing a function, but for the sake of generality, we just refer to the process here as gradient descent.

PROJECTED GRADIENT DESCENT

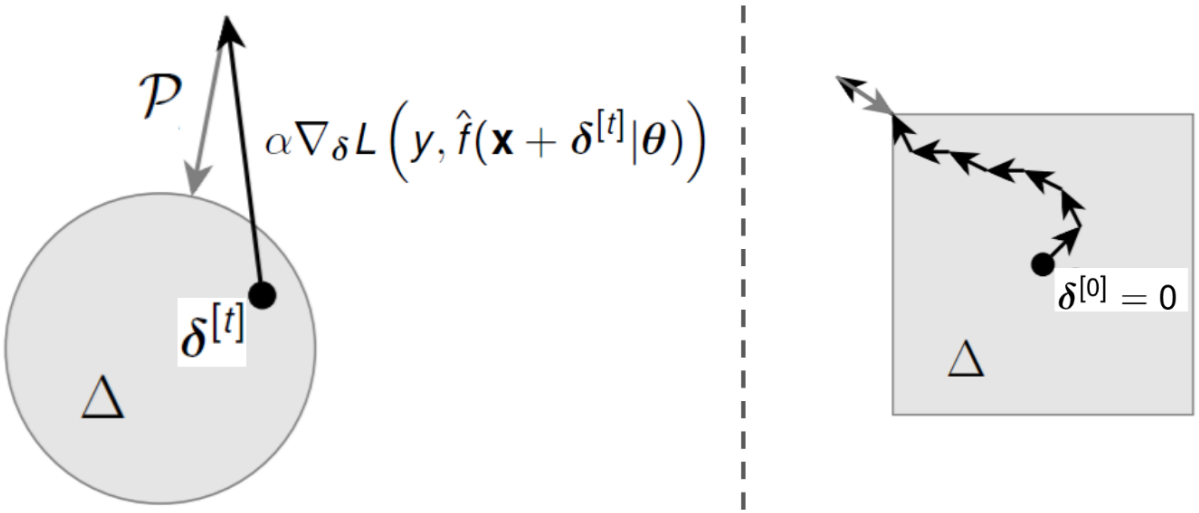
- In essence, $\boldsymbol{\delta}^{[t+1]}$ is obtained by moving from $\boldsymbol{\delta}^{[t]}$ (with a step size α) **in the direction** of the gradient of the loss with respect to $\boldsymbol{\delta}$ (evaluated at $\boldsymbol{\delta}^{[t]}$) and then projecting back to Δ.
- In case where the feasible set is $\mathcal{B}_\epsilon^\infty$, the projection of an arbitrary vector $\mathbf{z}$ is simply:

$$\mathcal{P}(\mathbf{z}) = \mathit{clip}(\mathbf{z}, [-\epsilon, \epsilon])$$

- Therefore, when Δ is $\mathcal{B}_\epsilon^\infty$, one gradient step is then defined as:

$$\boldsymbol{\delta}^{[t+1]} = \mathit{clip}\left(\boldsymbol{\delta}^{[t]} + \alpha \nabla_{\boldsymbol{\delta}} L\left(y, \hat{f}(\mathbf{x} + \boldsymbol{\delta}^{[t]} | \boldsymbol{\theta})\right), [-\epsilon, \epsilon]\right)$$

PROJECTED GRADIENT DESCENT



Credit: Kolter and Madry

Figure: *Left*: $\delta^{[t]} + \alpha \nabla_{\delta} L(y, \hat{f}(\mathbf{x} + \delta^{[t]}|\theta))$ is projected back to the perturbation set Δ using $\mathcal{P}$. Here, Δ is $\mathcal{B}_{\epsilon}^{2}$. *Right*: Multiple steps of PGD are shown in case of $\Delta = \mathcal{B}_{\epsilon}^{\infty}$; the projection must be only executed in the last step. (Note: A variant of PGD known as *normalized* PGD is shown here. This is why each step has the same size. See Kolter et al. (2019) for details.)

FAST GRADIENT SIGN METHOD

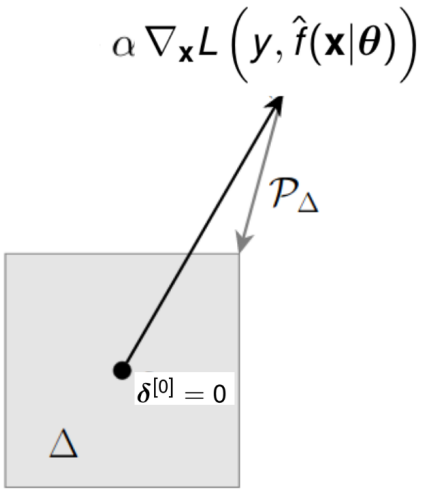
- Fast Gradient Sign Method (FGSM) is a special case of PGD when $\Delta = \mathcal{B}_\epsilon^\infty$ and $\alpha \to \infty$.
- As before, the projection of an arbitrary vector $\mathbf{z}$ onto $\mathcal{B}_\epsilon^\infty$ is $\mathcal{P}(\mathbf{z}) = \mathit{clip}(\mathbf{z}, [-\epsilon, \epsilon])$.
- As $\alpha \to \infty$, the elements of $\boldsymbol{\delta}$ are either set to $-\epsilon$ or ϵ depending on the sign of the corresponding component of the gradient.
- Thus, the FGSM algorithm computes $\boldsymbol{\delta}$ as

$$\boldsymbol{\delta} = \epsilon \mathit{sign}\left(\nabla_{\mathbf{x}} L\left(y, \hat{f}(\mathbf{x}|\boldsymbol{\theta})\right)\right)$$

- Note that for FGSM we only conduct one calculation and not multiple gradient descent steps.
- Furthermore, because we usually initialize $\boldsymbol{\delta}^{[0]}$ as 0

$$\nabla_{\boldsymbol{\delta}} L\left(y, \hat{f}(\mathbf{x} + \boldsymbol{\delta}|\boldsymbol{\theta})\right) = \nabla_{\mathbf{x}} L\left(y, \hat{f}(\mathbf{x}|\boldsymbol{\theta})\right)$$

FAST GRADIENT SIGN METHOD



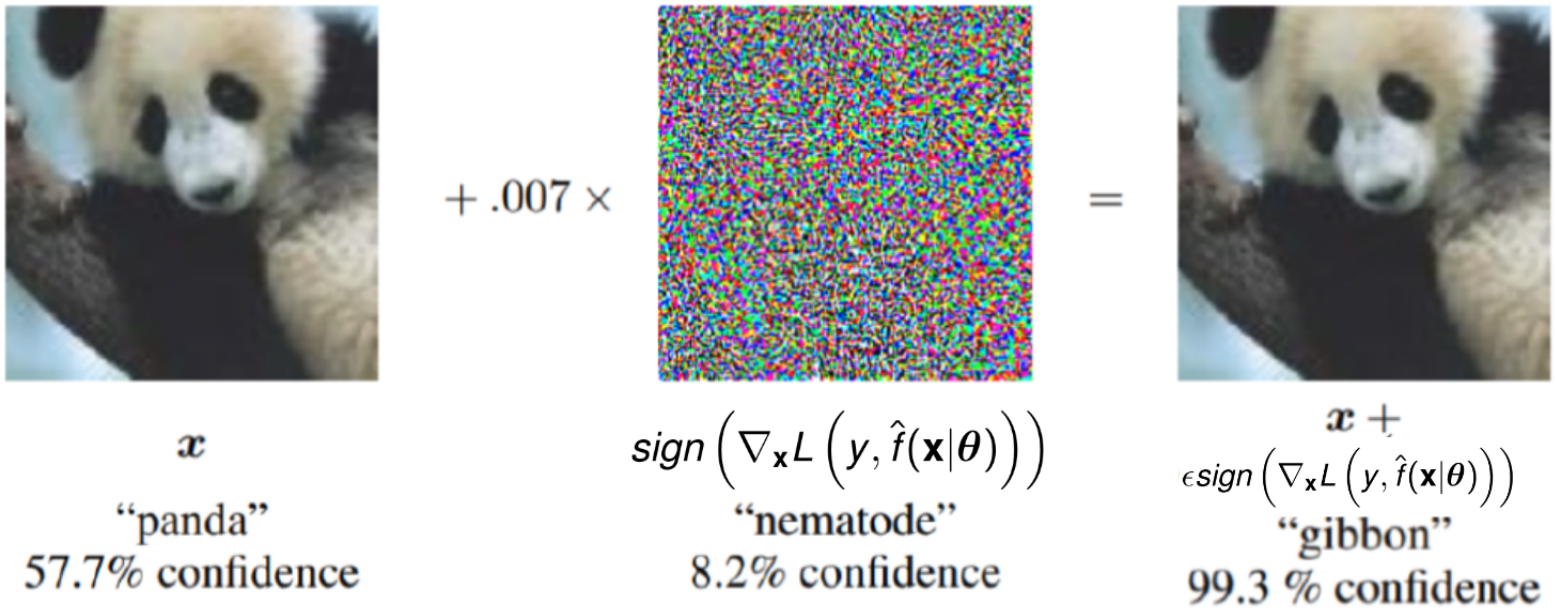
Credit: Kolter and Madry

Figure: $\boldsymbol{\delta}$ is obtained by setting each element of $\nabla_{\mathbf{x}} L\left(y, \hat{f}(\mathbf{x}|\boldsymbol{\theta})\right)$ to $-\epsilon$ or ϵ depending on its sign. Note that this slightly changes the direction of the step that is taken.

- Implicitly, the (element-wise) *sign* function is simply a way to constrain the size of the "step" that we take in the direction of the gradient. It is basically equal to a projection of the step back on Δ (which is $\mathcal{B}_{\epsilon}^{\infty}$).

FAST GRADIENT SIGN METHOD

- Note: The optimal attack against the linear binary classifier we saw in the last section was also FGSM!



Credit: Goodfellow

Figure: By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, GoogLeNet's classification of the image was changed from 'panda' to 'gibbon'. In this example, the ϵ is 0.007.

REFERENCES

Zico Kolter and Aleksander Madry (2019)

Adversarial Robustness - Theory and Practice

https://adversarial-ml-tutorial.org/

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Ian Goodfellow (2017)

Lecture 16 | Adversarial Examples and Adversarial Training

https://www.youtube.com/watch?v=CIfsB_EYsVI

Ian GoodfellowNicolas PapernotSandy HuangRocky DuanPieter AbbeelJack Clark (2017)

Attacking Machine Learning with Adversarial Examples

https://openai.com/blog/adversarial-example-research/

REFERENCES

Anh Nguyen, Jason Yosinski and Jeff Clune (2015)

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

https://arxiv.org/abs/1412.1897

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinto (2012)

ImageNet Classification with Deep Convolutional Neural Networks. NIPS.

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

Maxence Prevost (2018)

Adversarial ResNet50

http://arxiv.org/abs/1207.0580

Mahmood Sharif, Sruti Bhagavatula and Lujo Bauer (2016)

Accessorize to a Crime: Real and Stealthy Attacks onState-of-the-Art Face Recognition. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.

https://dl.acm.org/doi/10.1145/2976749.2978392

REFERENCES

Goodfellow, Shlens (2014)

Explaining and Harnessing Adversarial Examples

https://github.com/maxpv/maxpv.github.io/blob/master/notebooks/Adversarial_ResNet50.ipynb

Papernot , McDaniel, Goodfellow, Jha, Celik, Swamy (2016)

Practical Black-Box Attacks against Machine Learning

https://arxiv.org/abs/1602.02697

Athalye , Engstrom, Ilyas, Kwok (2017)

Synthesizing Robust Adversarial Examples

https://arxiv.org/abs/1707.07397

Tom B. Brown and Catherine Olsson, Research Engineers, Google Brain Team (2018)

Introducing the Unrestricted Adversarial Examples Challenge

https://ai.googleblog.com/2018/09/introducing-unrestricted-adversarial.html