Adversarial examples

- What are adversarial examples?
- Characteristics of adversarial examples.
- Adversarial examples arise from linear decision boundaries.
- How can we train to avoid these?

Adversarial examples

Many machine learning models (not just neural networks) are vulnerable to adversarial examples. In this lecture, we'll focus on adversarial examples in neural networks.

- Adversarial examples are examples that are specifically tailored so that the neural network misclassifies them.
- These specifically tailored images are only slightly different from correctly classified examples.
- In many cases, adversarial and non-adversarial examples are indistinguishable to the human eye.

Adversarial examples expose a blind spot in neural networks.

Adversarial example intuition

Let's take the examle of a simple binary classifier using logistic regression. Here, the network computes $z = \sigma(\mathbf{w}^T\mathbf{x} + b)$ and outputs 1 if $z > 0.5$ and 0 otherwise. Consider $b = 0$ and $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{10}$, with the following values:

$$\begin{aligned} \mathbf{w} &= \begin{bmatrix} -5 & 3 & -2 & 2 & -2 & -5 & -4 & 3 & 4 & -4 \end{bmatrix} \\ \mathbf{x} &= \begin{bmatrix} -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 - 1 & -1 \end{bmatrix} \end{aligned}$$

With these two settings of $\mathbf{w}$ and $\mathbf{x}$, we have that $z = 0.12$, i.e., it's fairly confident that it is class 0.

Example: say we wanted to add a small perturbation to "fool" the classifier that this is not class 0, but class 1. To do this change, we will allow you to either add 0.1 or -0.1 to every entry of $\mathbf{x}$. How will you modify $\mathbf{x}$?

Adversarial example intuition (cont.)

$$\begin{aligned} \mathbf{w} &= \begin{bmatrix} -5 & 3 & -2 & 2 & -2 & -5 & -4 & 3 & 4 & -4 \end{bmatrix} \\ \mathbf{x} &= \begin{bmatrix} -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1-1 & -1 \end{bmatrix} \end{aligned}$$

In this example, we want to make $\mathbf{w}^T\mathbf{x}$ less negative, so if $w_i < 0$, then we want to subtract 0.1 from x_i, and vice versa. With this intuition, we transform $\mathbf{x}$ to be:

$$\mathbf{x} = \begin{bmatrix} -1.1 & -0.9 & 0.9 & -0.9 & 0.9 & 0.9 & -1.1 & 1.1-0.9 & -1.1 \end{bmatrix}$$

Under this transformation, $\sigma(\mathbf{w}^T\mathbf{x}) = 0.8$, i.e., the classifier is confident that this is in class 1 instead of class 0.

Adversarial example intuition (cont.)

- Note that, with a small modification to the values, we were able to significantly change the classifier's output.
- The intuition is that, because we knew $\mathbf{w}$, we knew had to wiggle $\mathbf{x}$ in small but intentional ways to trick the classifier.
- Note that often times, inputs to neural networks are much higher dimensional than 10 dimensions (e.g., images are $224 \times 224 \times 3$). To this end, even smaller wiggles can result in the neural network outputting the incorrect class.

The fast gradient sign method

The perturbation we used made was a simplified form of the fast gradient sign method. Here, we want to find an $\tilde{\mathbf{x}}$ that is adversarial. By using the Taylor series, we have that:

$$\mathcal{L}(\tilde{\mathbf{x}}) \approx \mathcal{L}(\mathbf{x}) + (\tilde{\mathbf{x}} - \mathbf{x})^T \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})$$

An adversarial example $\tilde{\mathbf{x}}$ is one that drastically changes the loss. We'll constrain the perturbation that we make on $\mathbf{x}$ as well (so that no element changes by more than ϵ). This gives us the following optimization problem:

$$\begin{aligned} \max_{\tilde{\mathbf{x}}} \quad & \mathcal{L}(\mathbf{x}) + (\tilde{\mathbf{x}} - \mathbf{x})^T \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}) \\ \text{subject to} \quad & \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty \leq \epsilon \end{aligned}$$

The gradient is $\nabla_{\tilde{\mathbf{x}}} \mathcal{L}(\tilde{\mathbf{x}}) = \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})$. Hence, a gradient ascent step to update $\tilde{\mathbf{x}}$ would be

$$\tilde{\mathbf{x}} = \mathbf{x} + \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})$$

With our infinity norm constraint, this solution becomes:

$$\tilde{\mathbf{x}} = \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}))$$

Empirical results of the fast gradient sign method

Goodfellow et al., 2014, used the fast gradient sign method to fool neural networks. Here are some of their empirical results:

- Using $\epsilon = 0.25$, they caused a shallow softmax classifier to have an error rate of 99.9% with an average confidence of 79.3% on MNIST.
- Using $\epsilon = 0.255$, they caused a maxout network to misclassify 89.4% of examples with an average confidence of 97.6% on MNIST.
- Using $\epsilon = 0.15$, they caused a maxout network to misclassify 87.15% of examples with an average confidence of 96.6% on CIFAR-10.

Implications of adversarial examples

Adversarial examples have several implications on neural networks.

- The existence of adversarial examples show that even though networks achieve excellent test set performance, they are not learning true underlying concepts that classify optimally.
- In particular, they must be using features different than those learned by humans, as imperceptably small changes correspond to large differences in the network's representation.
- Goodfellow et al., 2014, use the analogy of the "Potemkin village." While neural networks can classify images with natural statistics, as soon as the statistics of the image change these networks can be tricked. E.g., see Fig 1 of their 2014 paper adding an imperceptibly small vector to a "panda" that changes it into a "gibbon".

Even worse news: adversarial examples are transferable

Interestingly, adversarial examples are transferable between networks and algorithms (Papernot et al., 2016).

- Concretely, an attacker may construct his own model, and from his model, create adversarial perturbations.
- The attacker can then transfer these adversarial perturbations to the victim, with minimal knowledge about the victim's model.
- As a result, we ought not think of these adversarial examples as precise perturbations that are model specific.
- Indeed it's possible to display adversarial noise in the real world and fool models that view these images through a camera.

Why do adversarial examples have these characteristics?

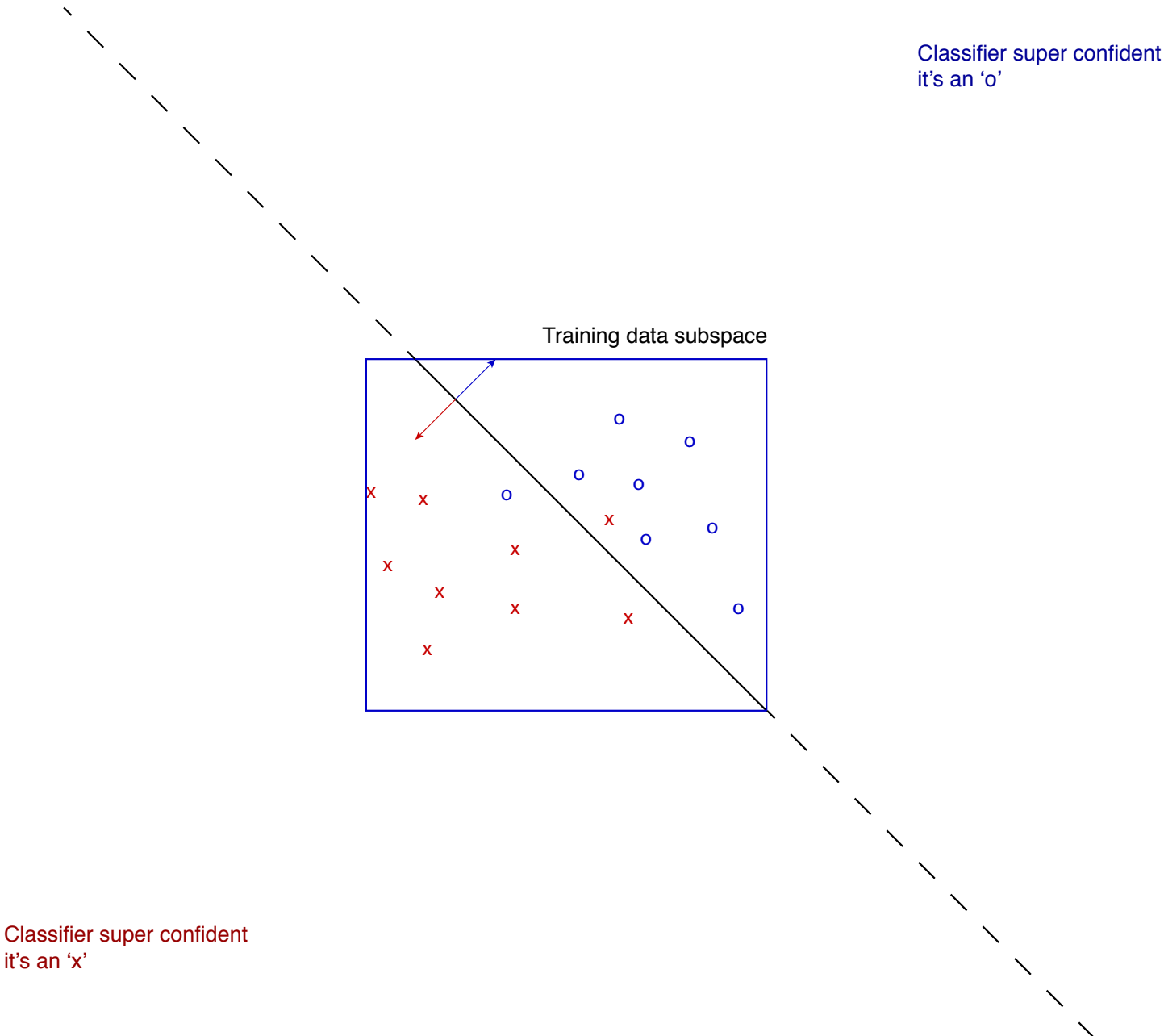
At this point, it can be confounding to explain why adversarial examples behave like this. One intuition is that adversarial examples may occur because the neural network is so high capacity that it overfits.

- By overfitting, but not adequately sampling the high-dimensional space (really, it only samples natural images), the neural network has potentially complex and somewhat arbitrary decision boundaries in subspaces corresponding to unnatural image statistics. Data in these points can be misclassified.
- But this can be explained away by the fact that adversarial examples are transferable. If adversarial examples resulted from overfitting, they would be specific to each instantiation of an overfit model.

Why do adversarial examples have these characteristics? (cont.)

Rather, Goodfellow and colleagues showed that the adversarial examples arise from excessive linearity (!) (Goodfellow et al., 2014). The idea is the following: we can draw a linear classifier, and it will classify well in the regions where the data is. But as you move further away from the linear boundary, the model becomes even more confident of one class. The figure on the next page is an appropriate one to have in mind:

Why do adversarial examples have these characteristics? (cont.)



But aren't neural networks nonlinear?

Yes, neural networks are nonlinear, but in practice, they're largely piecewise linear (which again, is nonlinear). Because they're piecewise linear, they're susceptible to this idea that as you move further away along a 1D manifold (given by the signed gradient) the network will gain extreme confidence in any given class.

- Why are neural networks piecewise linear?
- An intuition for this question can be answered by a common thread we've talked about all lecture: keeping gradients big.
- ReLU's are piecewise linear; sigmoids are linear in the region where we want to keep gradients big.
- Really, when you think about it, any time you can backpropagate a derivative and get back to the input without substantial vanishing or exploding gradients, you followed a largely linear path.
- So if you're able to train with backpropagation, you're probably susceptible to adversarial examples.

More evidence along these lines

To present even more evidence along these lines, Goodfellow and colleagues also evaluated a radial-basis function (RBF) classifier, which is a nonlinear classifier. They found that they were far more robust to adversarial examples.

- So why not put an RBF at the end of a neural network to classify?
- Well, the RBF is nonlinear, and so it has only small regions in parameter space where the gradients are large enough. Elsewhere, it dies to zero.
- Hence, training with RBFs (or other functions like it) is very difficult, since gradients can't be backpropagated well.

And here we run into a large problem: if we can train with gradient based methods, it largely relied on being able to backpropagate through locally linear activation functions. As soon as we move outside of this area, and have gradients that can die far more easily, we are more robust to adversarial examples, but training becomes harder.

Adversarial training

However, there are still ways to address this. One way to address the problem of adversarial examples is to incorporate it into training. While more research has to be done along these lines, some papers have shown some results to help networks become more robust to adversarial examples. In one example (Goodfellow et al., 2014), the objective is modified via an effective regularizer:

$$\alpha\mathcal{L}(\theta, \mathbf{x}) + (1-\alpha)\mathcal{L}(\theta, \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}}\mathcal{L}(\theta, \mathbf{x})))$$

Goodfellow and colleagues shows that the network trained via this objective is more robust to adversarial examples, although it could be much improved. Further, the regularization results in better performance on benchmarks than if one were to just use dropout.