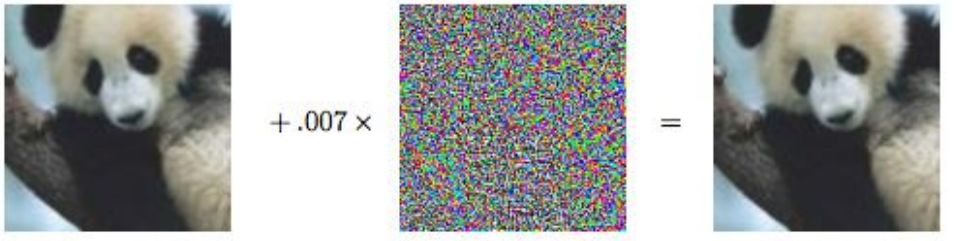
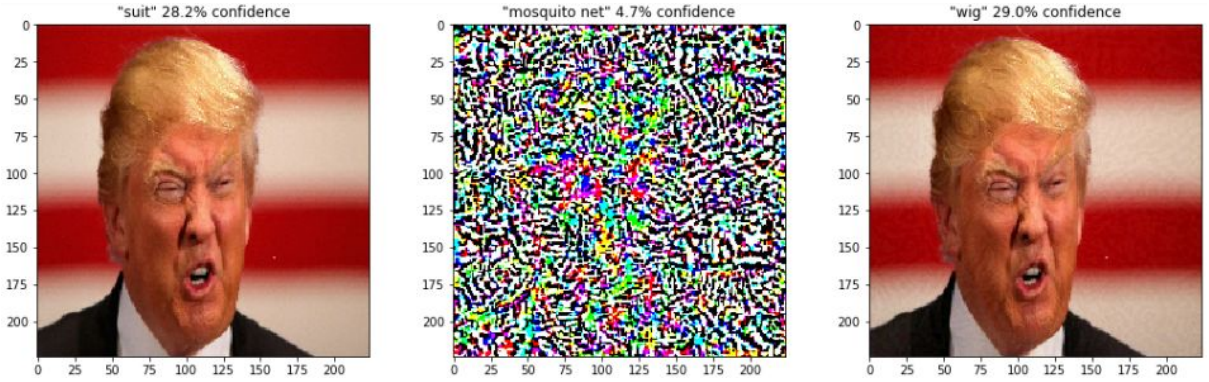
ADVERSARIAL EXAMPLES:

Neural networks have demonstrated robust performance in several computer vision tasks with new algorithms reported to even surpass human performance. But the latest studies have also shown that such networks are defenseless when it comes to the attacks of adversarial examples

Adversarial examples are inputs to a neural network that result in an incorrect output from the network. It's probably best to show an example.
We can start with an image of a panda on the left which some network thinks with 57.7% confidence is a "panda."But then by adding a very small amount of *carefully constructed* noise you can get an image that looks exactly the same to a human, but that the network thinks with 99.3% confidence is a "gibbon."



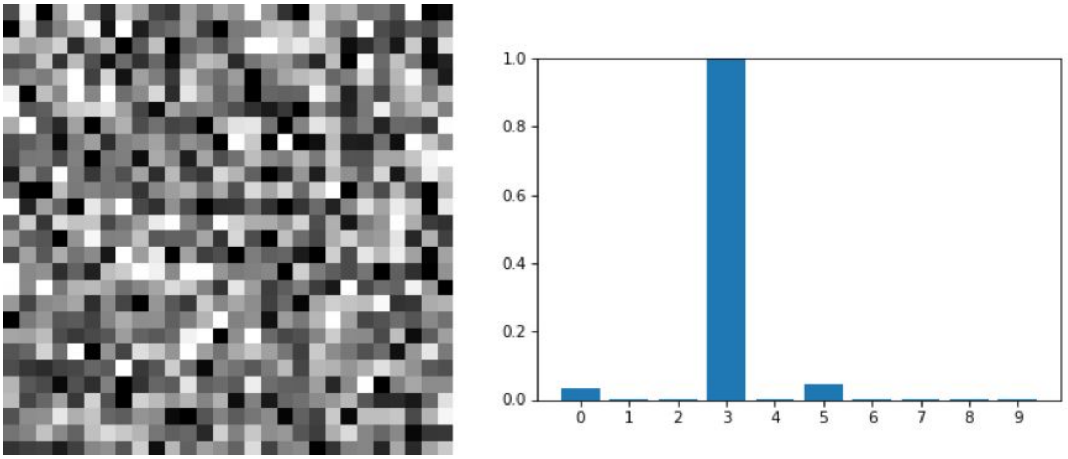
From Explaining and Harnessing Adversarial Examples by Goodfellow et al



An image classified as "suit" by the VGG16 neural network (left), a perturbation created specifically using the image on the left (middle) and the resulting perturbed image classified as a wig (right).

The above adversarial examples are targeted examples . A small amount of carefully constructed noise was added to an image that caused a neural network to misclassify the image, despite the image looking exactly the same to a human.

There are also non-targeted examples which simply try to find *any* input that tricks the neural network. This input will probably look like white noise to a human, but because we aren't constrained to find an input that resembles something to a human the problem is a lot easier.



White-Box & Black-Box attacks

Depending on the knowledge level of the attacker, the adversarial attacks can be classified as either white-box or black-box attacks. White-box attacks are where the adversary has complete knowledge about the model being attacked like weights, biases, hyper parameters used etc. Black-box attacks are where the adversary is a normal user who knows only the output of the model.

Linear behavior in high-dimensional spaces is sufficient to cause adversarial examples.

Explanation:

In many problems, the precision of an individual input feature is limited. For example, digital images often use only 8 bits per pixel.so they discard all information below 1/255 of the dynamic range. Because the precision of the features is limited, it is not rational for the classifier to respond differently to an input x than to an adversarial input :

$$\tilde{x} = x + \eta$$

if every element of the perturbation η is smaller than the precision of the features. Formally, for problems with well-separated classes, we expect the classifier to assign the same to x and ˜ x so long as$||\eta||_\infty < \text{epsilon}$, where epsilon is small enough to be discarded by the sensor or data storage apparatus associated with our problem.

Consider the dot product between a weight vector w and an adversarial example ˜ x:
w.T x = w.Tx + w.Tη.
The adversarial perturbation causes the activation to grow by w.Tη. Since ||η||∞ does not grow with the dimensionality of the problem but the change in activation caused by perturbation by η can grow linearly with n, then for high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output.
This explanation shows that a simple linear model can have adversarial examples if its input has sufficient dimensionality.

How to generate adversarial examples?

Non-Targeted Examples:

$$y_{goal} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

That is, we want to come up with an image such that the neural network's output is the above vector. In other words, find an image such that the neural network thinks the image is a 5 (remember, we're zero indexing). It turns out we can formulate this as an

optimization problem in much the same way we train a network. Let's call the image we want to make $\vec{x}$ (a 784 dimensional vector, because we flatten out the 28 × 28 pixel image to make calculations easier). We'll define a cost function as:

$$C = \frac{1}{2}\|y_{goal} - \hat{y}(\vec{x})\|_2^2$$

Where the *y_goal* is our goal label from above. The output of the neural network given our image is $y(\vec{x})$. You can see that if the output of the network given our generated image $\vec{x}$ is very close to our goal label *y_goal*, then the corresponding cost is low. If the output of the network is very far from our goal then the cost is high. Therefore, finding a vector $\vec{x}$ that minimizes the cost CC results in an image that the neural network predicts as our goal label. Our problem now is to find this vector $\vec{x}$.

This problem is incredibly similar to how we train a neural network, where we define a cost function and then choose weights and biases (a.k.a. parameters) that minimize the cost function.

In the case of adversarial example generation, instead of choosing weights and biases that minimize the cost, we hold the weights and biases constant (in essence hold the entire network constant) and choose an $\vec{x}$ input that minimizes the cost.

we'll use gradient descent!!

There might be something bugging you at this point. If we want to make an adversarial example corresponding to a five then we want to find a $\vec{x}$ that when fed into the neural network gives an output as close as possible to the one-hot vector representing "5". However, why doesn't gradient descent just find an image of a "5"? After all, the neural network would almost certainly believe that an image of a "5" was actually a "5" (because it *is* actually a "5").

A possible theory as to why this happens is the following:

The space of all possible 28×28 images is utterly massive. There are 256^(28×28)≈10^1888 possible different 28×28 pixel black and white images.
And out of all these photos only an essentially insignificant fraction actually look like numbers to the human eye. Whereas given that there are so many images, a good amount of them would look like numbers to a neural network . So there is very less Probability that neural network will choose an image which looks like numbers to humans.

TARGETED Examples

What we can do is add a term to the cost function that we're minimizing. Our new cost function will be:

$$C = \frac{1}{2}\|y_{goal} - \hat{y}(\vec{x})\|_2^2 + \lambda\|\vec{x} - x_{target}\|_2^2$$

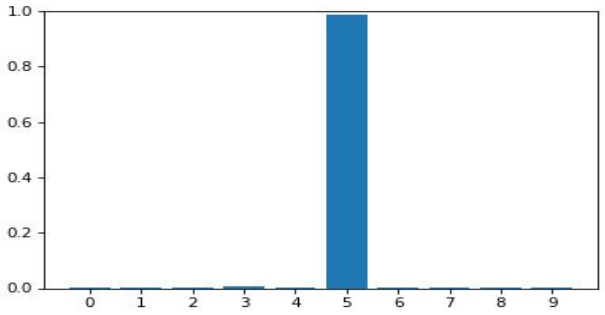
Where x_target is what we want our adversarial example to look like (*x_target* is therefore a 784 dimensional vector, the same dimension as our input). So what we're doing now is we're simultaneously minimizing two terms. The left term we've seen already. Minimizing this will make the neural network output y_goal when given $\vec{x}$. Minimizing the second term will try to force our adversarial image *x* to be as close as possible to *x_target* as possible (because the norm is smaller when the two vectors are closer), which is what we want!

The extra λ out front is a hyperparameter that dictates which of the terms is more important. As with most hyperparameters we find after a lot of trial and error that .05 is a good number to set λ to.

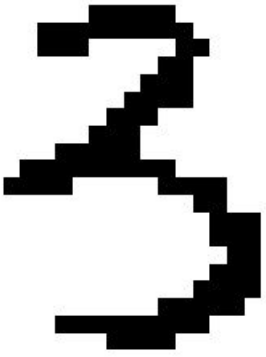
Protecting Against Adversarial Attacks

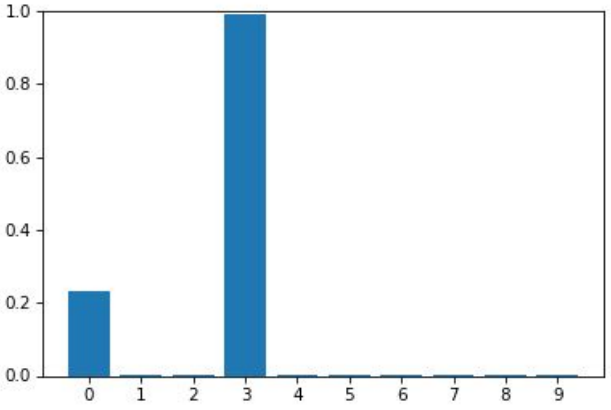
If you look closely at the original images and the adversarial examples you'll see that the adversarial examples have some sort of grey tinged background.

One naive thing we could try is to use binary thresholding to completely white out the background



3 classified as 5





3 classified as 3

Turns out binary thresholding works! But this way of protecting against adversarial attacks is not very good. Not all images will always have an all white

background. For example look at the image of the panda at the very beginning of this post. Doing binary thresholding on that image might remove the noise, but not without disturbing the image of the panda a ton. Probably to the point where the network (and humans) can't even tell it's a panda.

Another simple method is *ADVERSARIAL TRAINING*

***Adversarial Training* is when adversarial examples are used during training to reduce misclassification. Doing this results in a significant increase in accuracy over adversarial test examples.**

Training with an adversarial objective function based on the *fast gradient sign method*(FGSM) was an effective regularizer:

$$\tilde{} J(\theta,x,y) = \alpha J(\theta,x,y) + (1-\alpha)J(\theta,x + \text{sign}(\nabla x J(\theta,x,y)).$$

While this strategy improves robustness to *FGSM* attacks, it has multiple downside:

:It does not help against more sophisticated white-box attacks
:It does not help against black-box attacks either.

Let's talk more about black -box attacks:

An interesting and important observation of adversarial examples is that they generally are not model or architecture specific. Adversarial examples generated for one neural network architecture will transfer very well to another architecture. In other words, if you wanted to trick a model you could create your own model and adversarial examples based off of it. Then these same adversarial examples will most probably trick the other model as well.

Moreover, when these different models misclassify an adversarial example, they often agree with each other on its class. This behavior is especially surprising from the view of the hypothesis that adversarial examples finely tile space like the rational numbers among the reals, because in this view adversarial examples are common but occur only at very precise locations

But Actually adversarial examples occur in contiguous regions of the 1-D subspace defined by the fast gradient sign method, not in fine pockets

.....