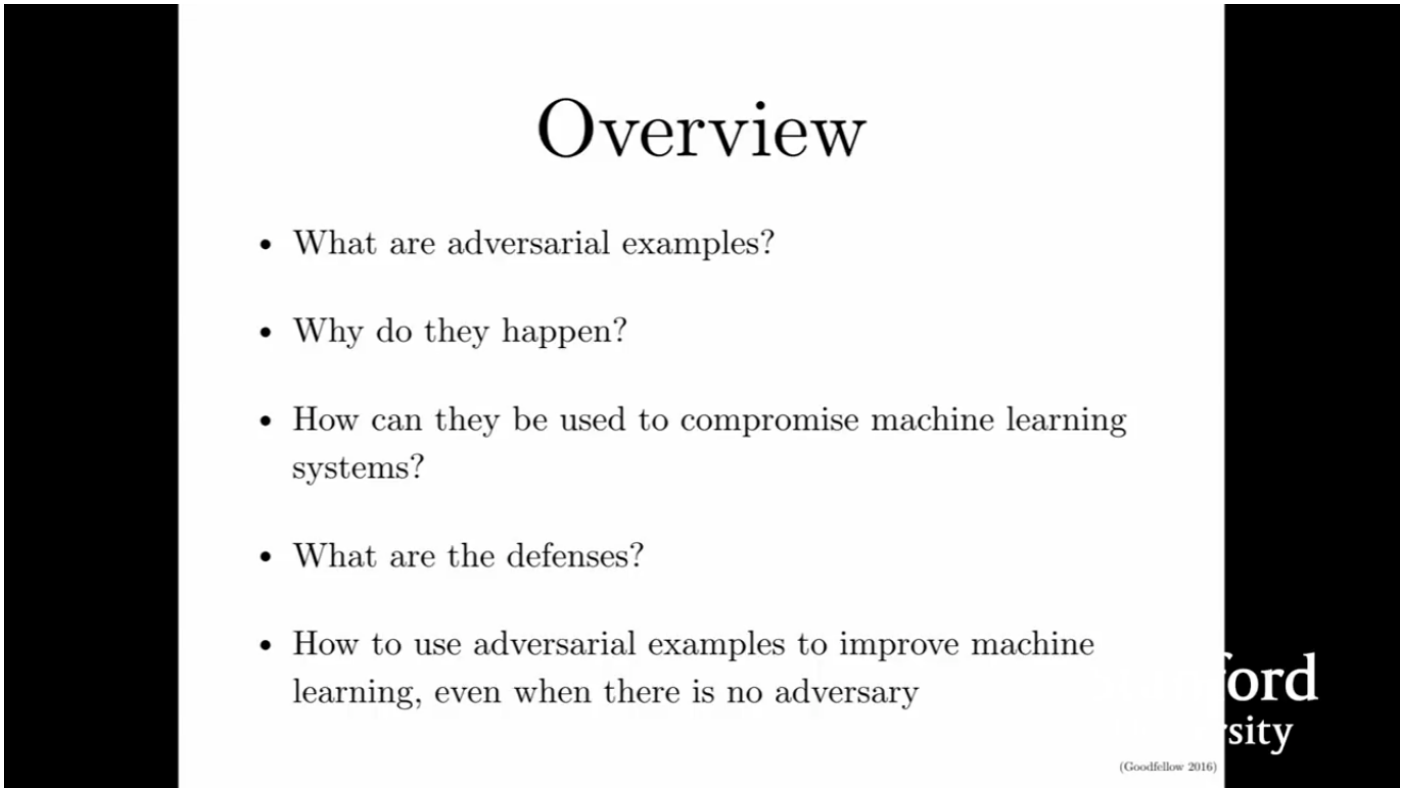
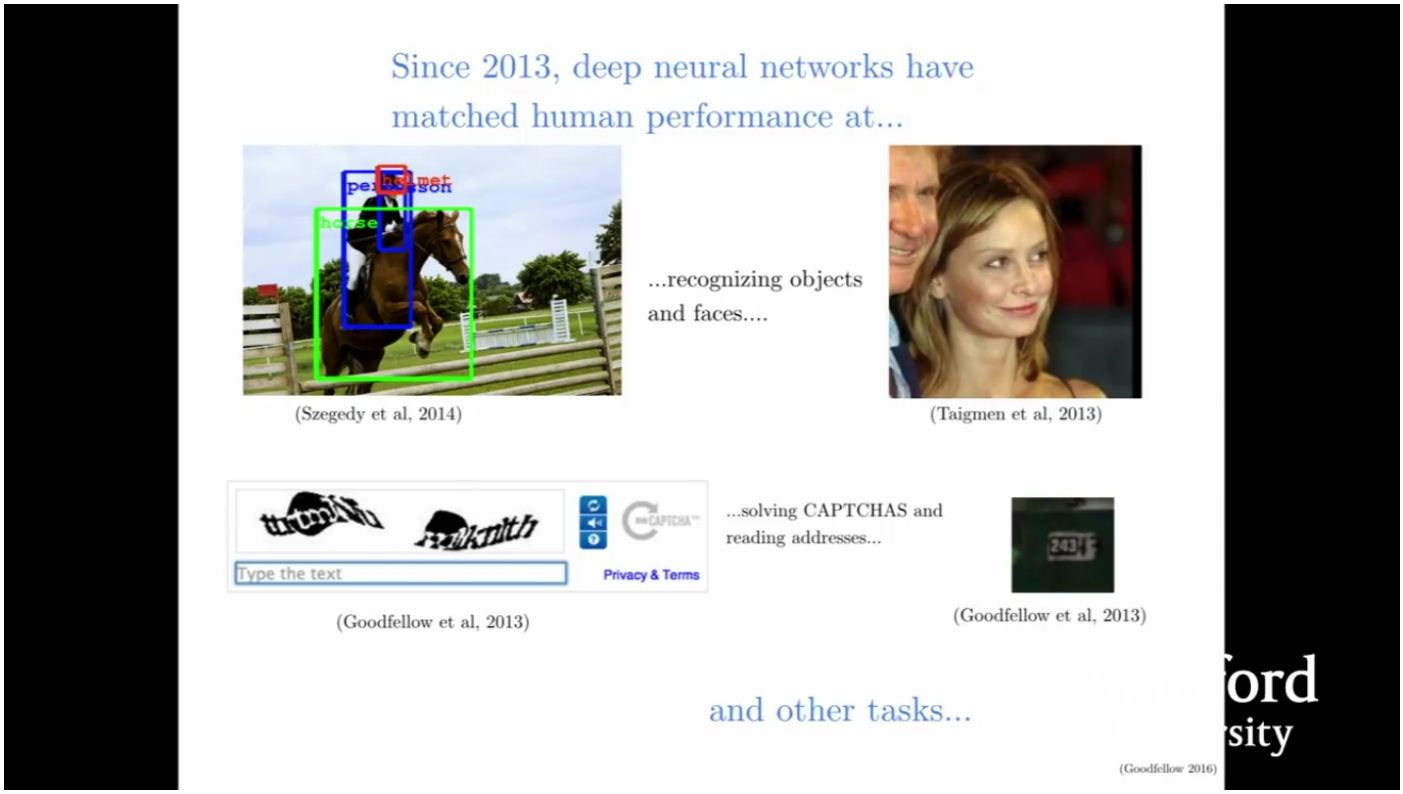
# lec16

Creation Date: 27/01/2020 17:31
Last Modified Date: 12/02/2020 19:26

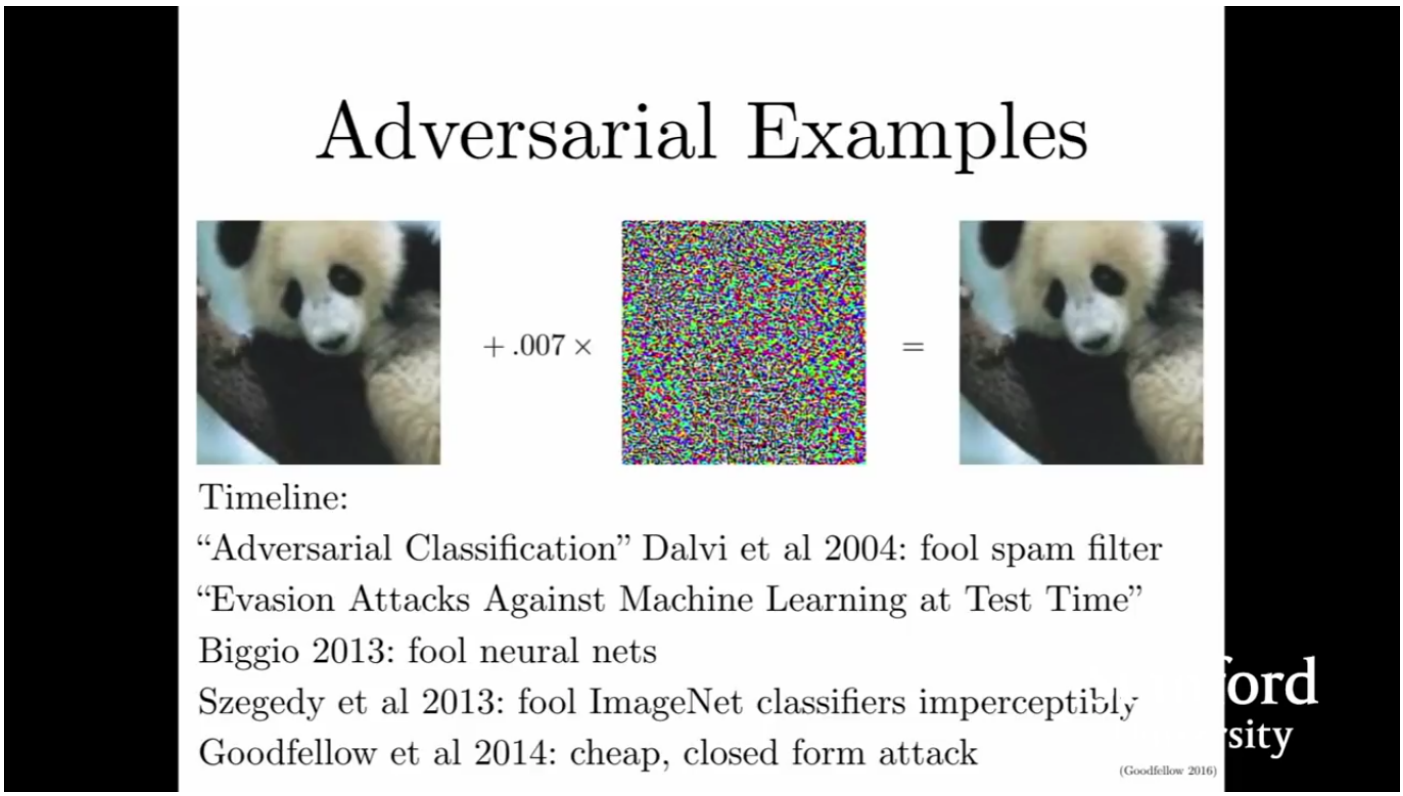
Lec 16: Adversarial Examples and Adversarial Training



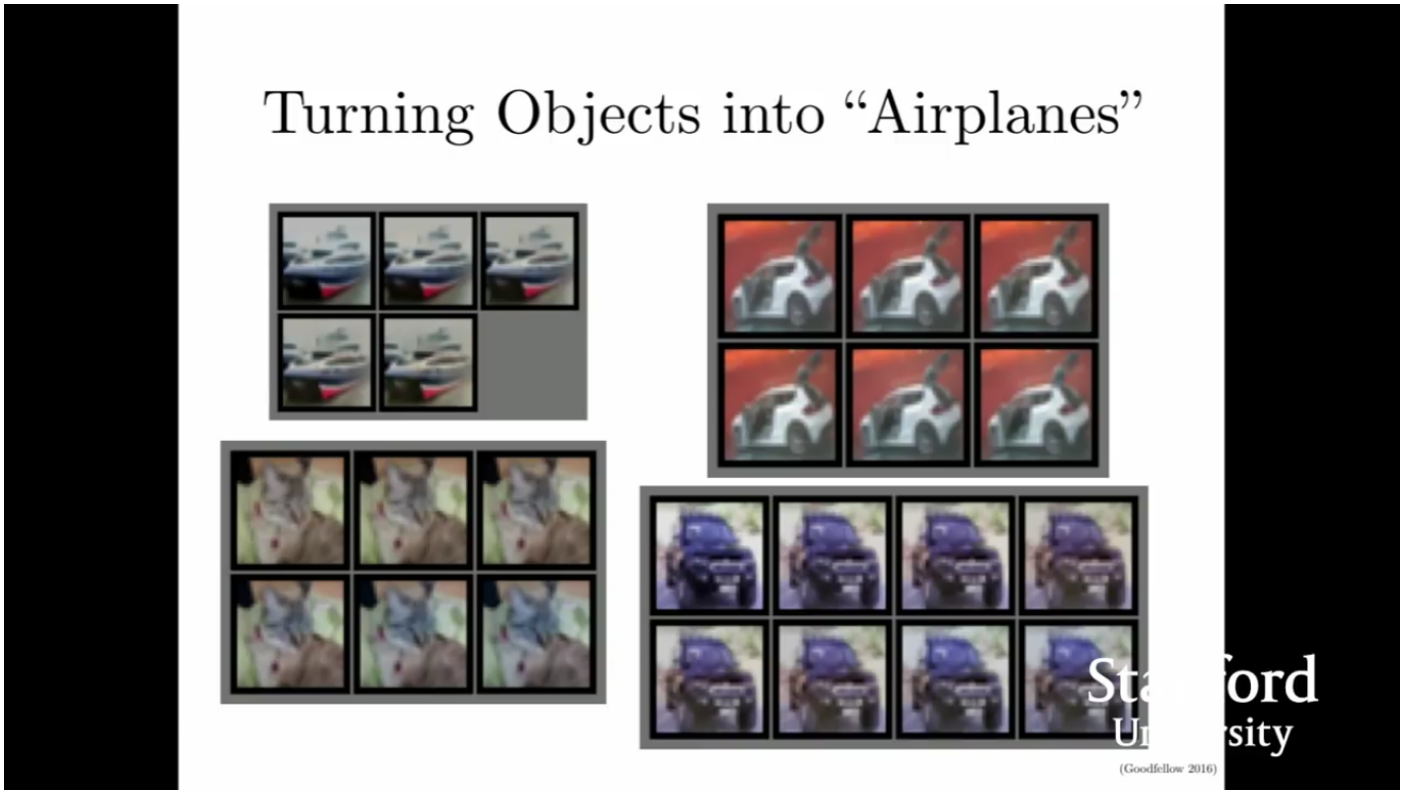




- Machine recognizes images better than humans because humans cannot distinguish intraclass variations (Eg. Siberian Husky and Alaskan Husky)

- Even captchas were being learnt in 2013.
- **Adversarial example is an example which is carefully computed to be misclassified.**

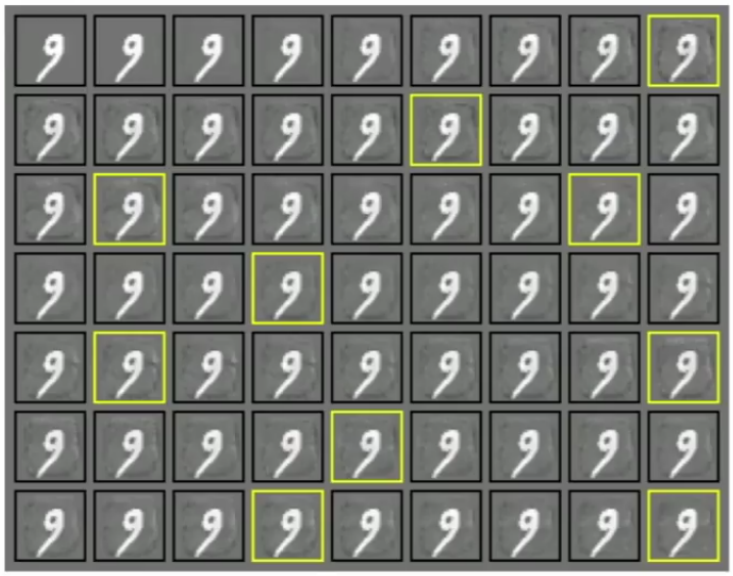



- The image on the left is of a panda with a 60% probability but after adding a carefully crafted noise it gets classified into a Gibbon with a 99.9% probability
- **We should notice that it was not just a boundary case where only a small perturbation made the misclassification but the model is very confident about the misclassification.**




- Using gradient descent these images are being classified as 'airplanes' but they dont possess the characteristics of an airplane.

# Attacking a Linear Model



(Goodfellow 2016)

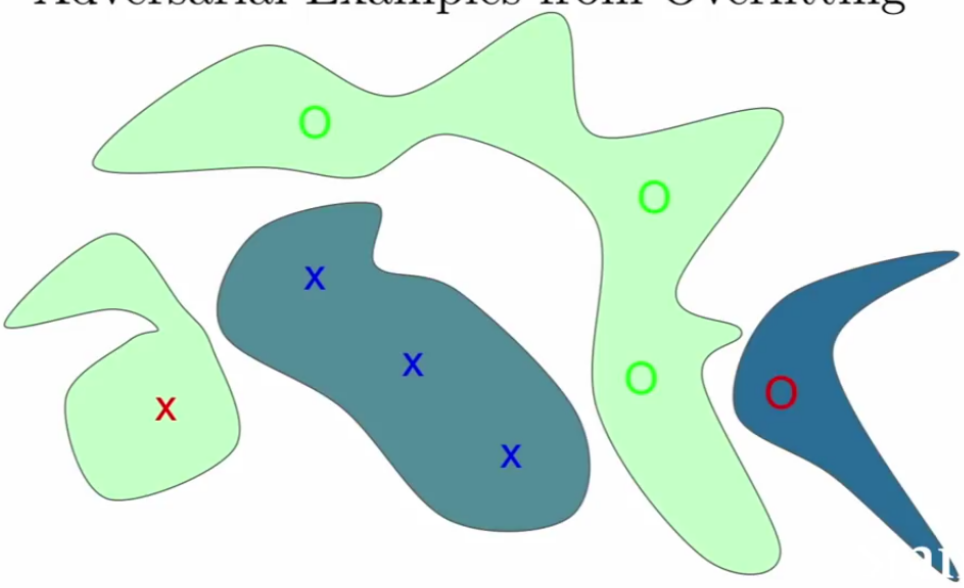
- Each row in this grid gets classified as a different number.

# Not just for neural nets

- Linear models
  - Logistic regression
  - Softmax regression
  - SVMs
- Decision trees
- Nearest neighbors

(Goodfellow 2016)

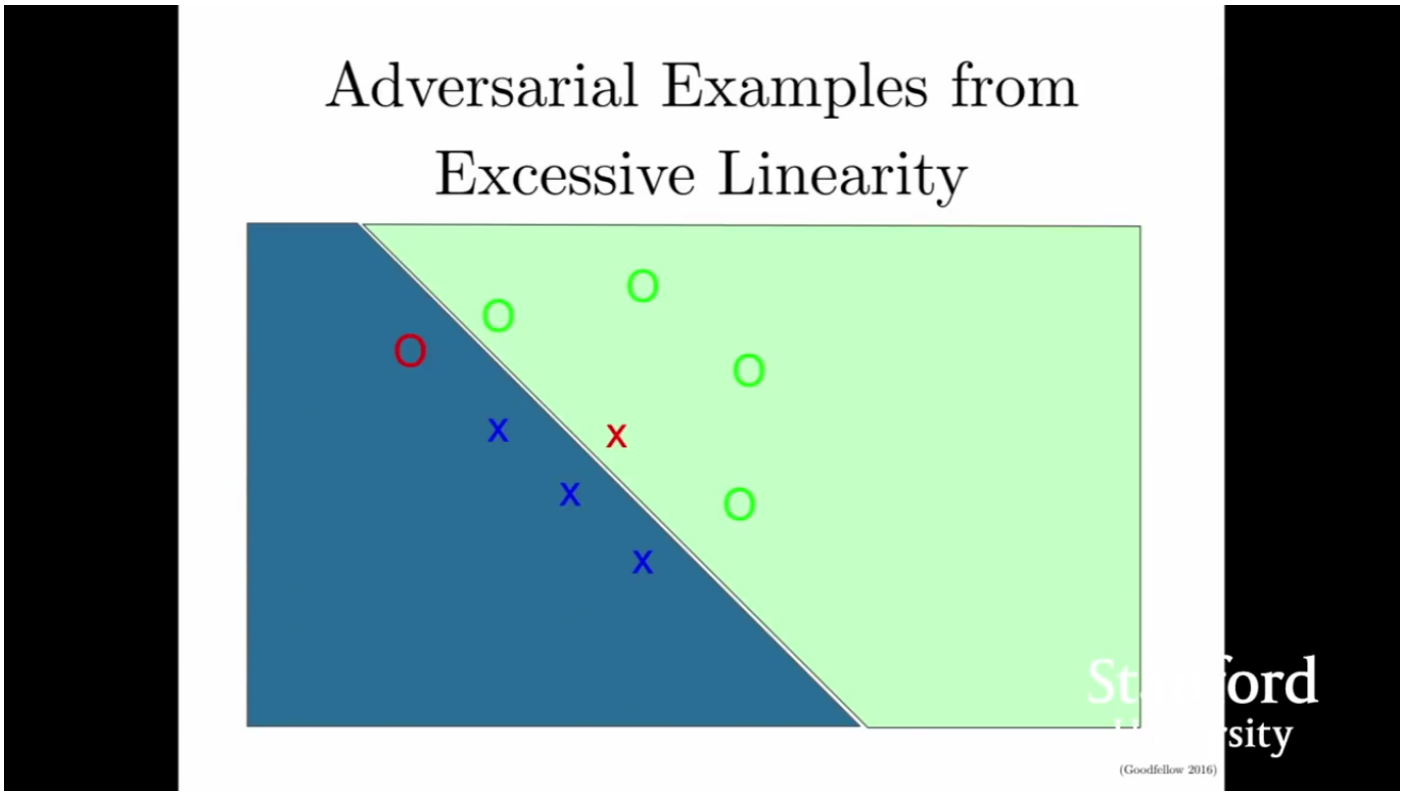
# Adversarial Examples from Overfitting




(Goodfellow 2016)

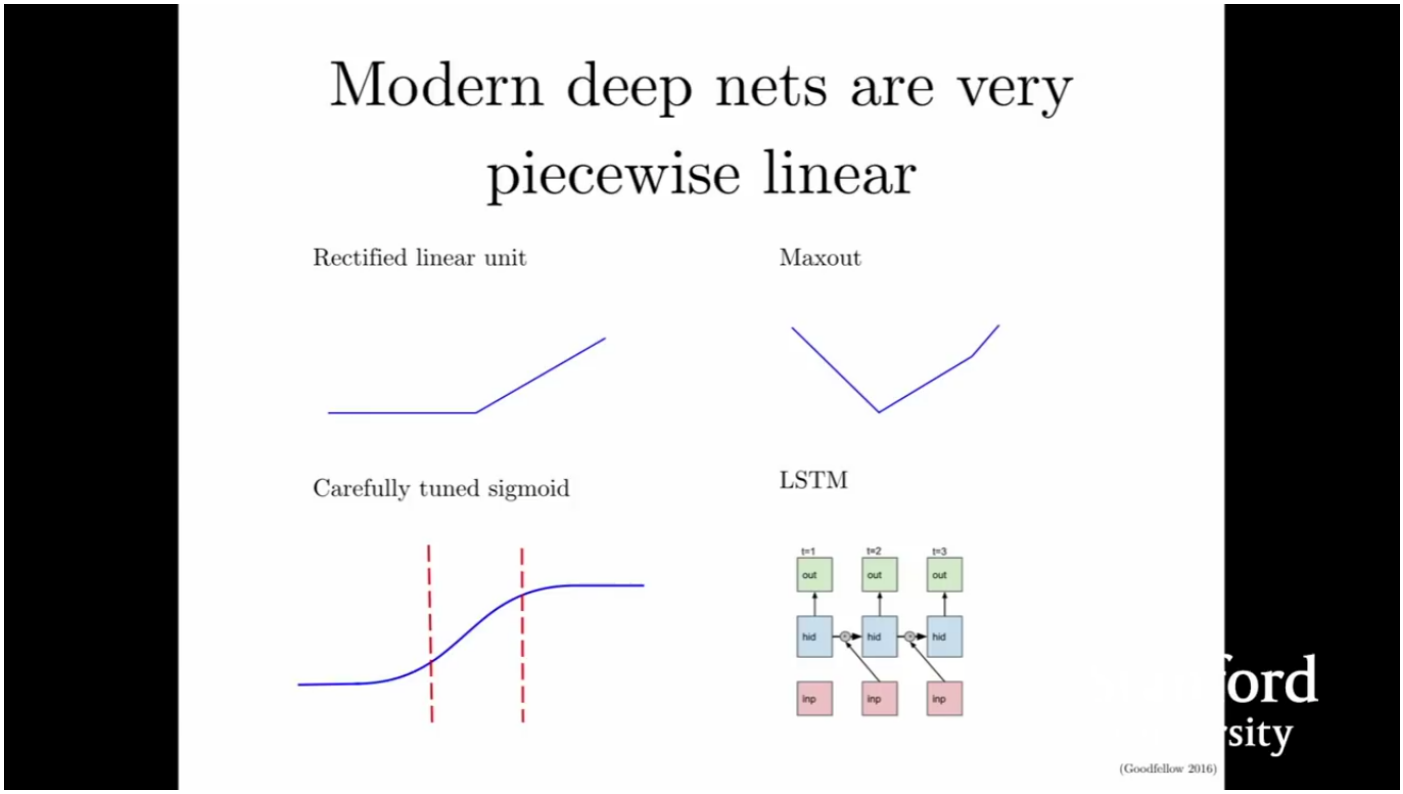
- Initial thoughts were that the adversarial attacks happen due to overfitting.
- The model had fit very well for the training set, **and since this a very complicated function and it has way too many parameters than it needs to represent the training task, it throws little mass of probability around the rest of space randomly.**
- We have a green blob where there are Xs and a blue blob where there are Os. **These blobs will give rise to adversarial examples**
- **But if overfitting was the cause then, the results would have been random and would differ for different functions.**

- But it was found that same adversarial examples can be used for different models and these models would all classify them to the same wrong class.
- If we take the difference between the original example and an adversarial example then we would get a direction in the input space, and we could add that offset vector to any clean example and would usually get an adversarial example as a result.
- **So this shows that adversarial attacks are systematic and not just a random event**

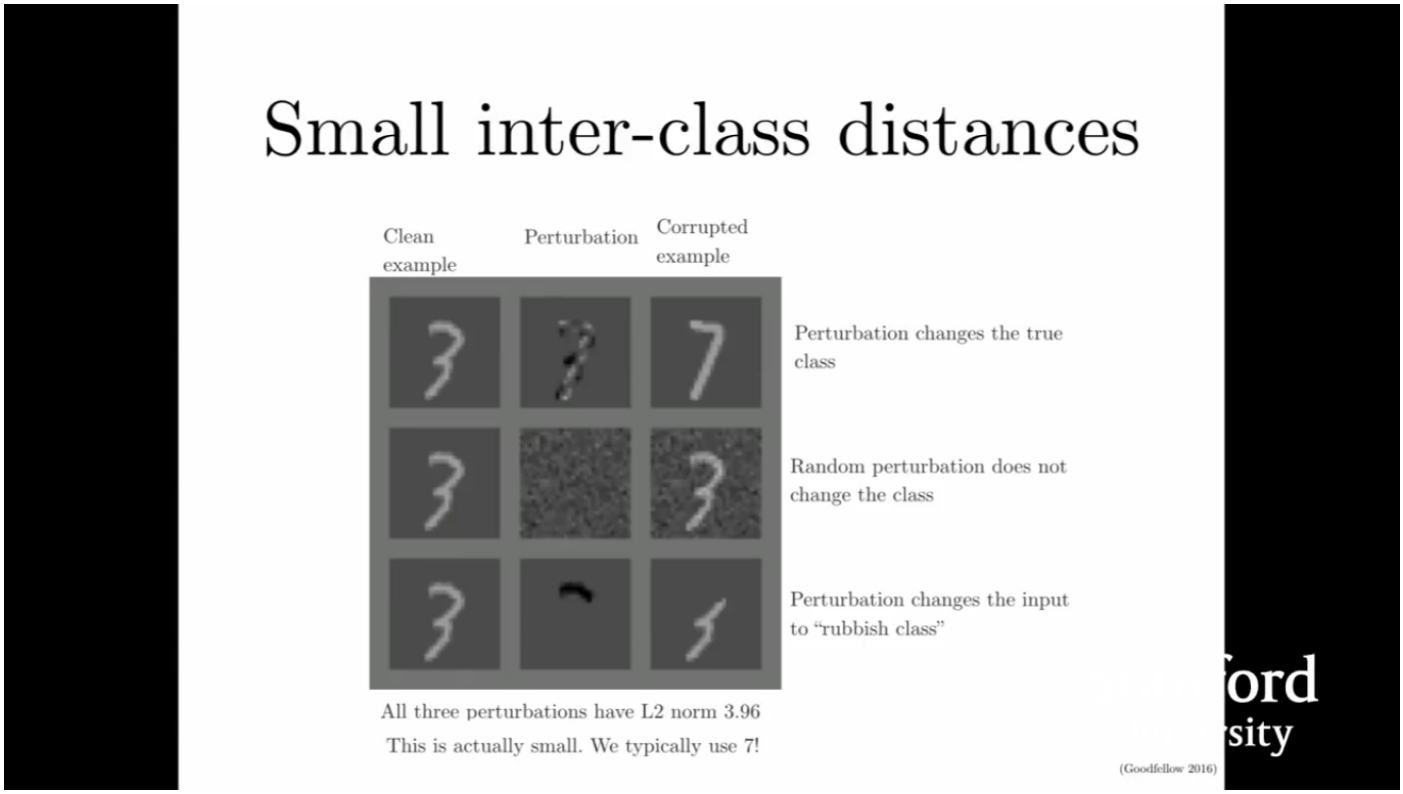



Another idea:

- Maybe the adversarial attacks happen due to underfitting, the model being too linear.
- *We can notice that the upper right and lower right region will be classified as one of the classes with a very high probability even though the model may not have seen any samples from that space*

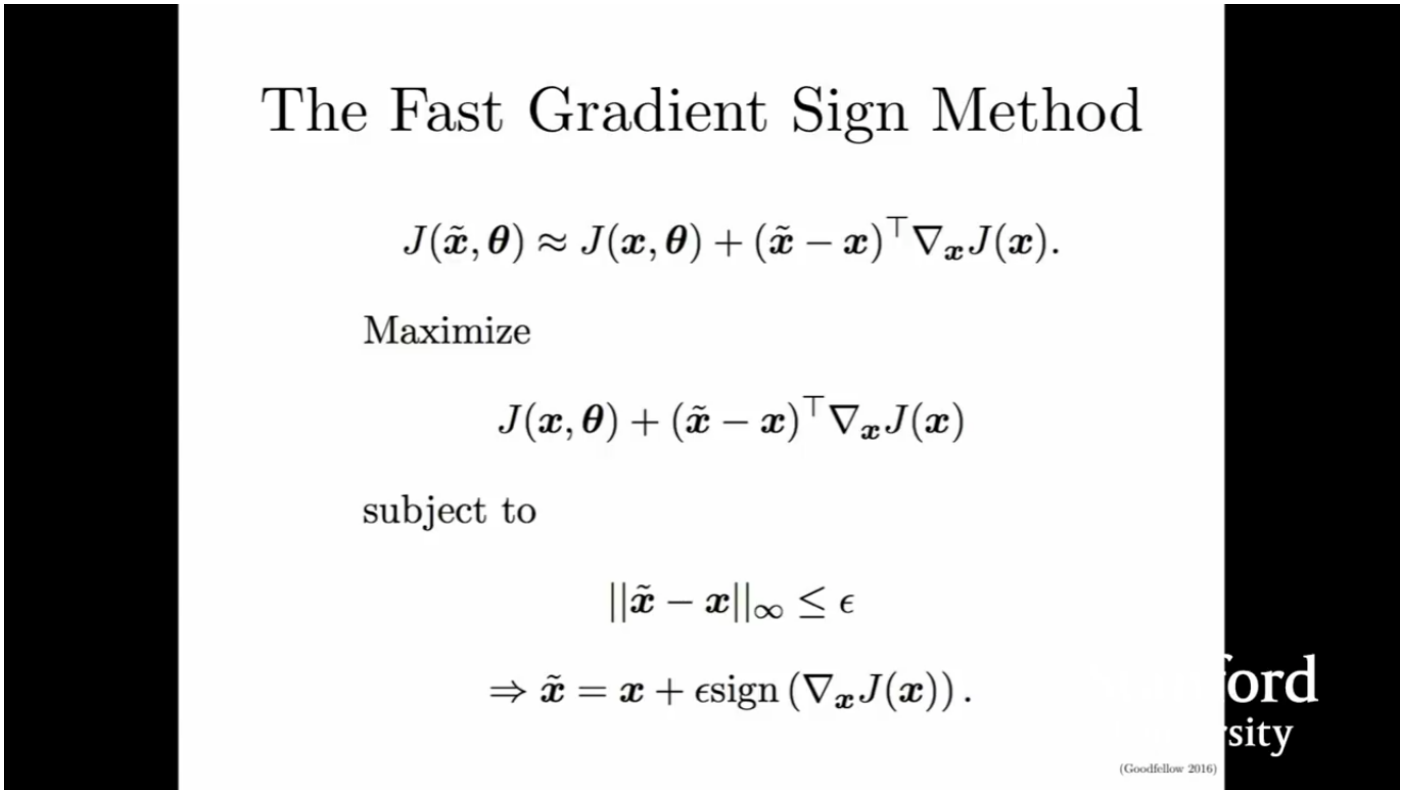



- So we have seen that linear models assign unusually high confidence values to samples which are away from the decision boundary, even if there isn't any data in those regions.
- **But are deep-neural-networks anything like linear models? Could linear models explain anything about how it is that deep neural networks fail?**
- Ans: It turns out that modern DNN are actually **piece-wise linear**, so rather than being a single linear function they are piece-wise linear with maybe not that many linear pieces. If we use ReLU then the mapping from the input image to the output logits(unnormalized log probabilities of the output before we apply the softmax) is **literally** a piece-wise linear function.
- There are other networks like MaxOut Networks which are also piece-wise linear.
- Before ReLU became mainstream, people used Sigmoid which needed careful initialization, and the center region of the Sigmoid is piece-wise linear.
- In LSTM, we use addition to propagate the information and addition is one of the simplest form of linearity.
- Linear in the sense of mapping of the input space to the output space and not the parameters space. Because as we go deeper the weight matrices are multiplied with each other forming non-linear functions.
- **Since the mapping of input to output is linear and the mapping of weights to the output is non-linear, it is easy to optimize a problem which models the input to the model than to optimize the weights.**



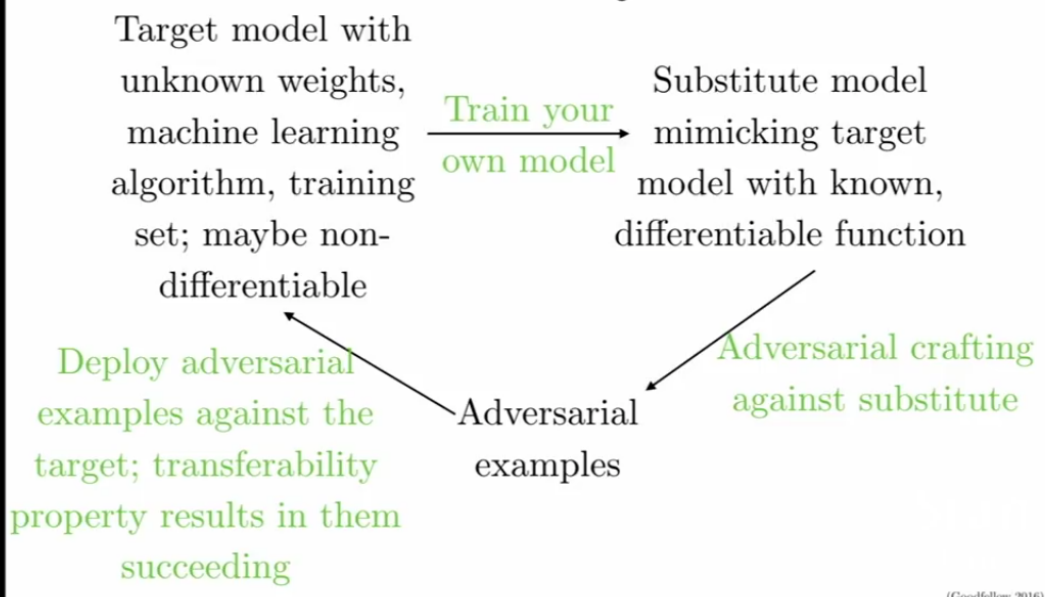

- **When we construct adversarial examples we need to take care that we are able to get quite a large perturbation without changing the image very much, as far as human being is concerned.**
- In the image above we are changing the image of 3 with a L2 Norm perturbation.
- In the first row we are changing 3 to 7 by looking at the nearest seven in the training set. The difference between these 2 images is an image which looks like a 7 wrapped by black lines(middle image of first row). The middle column shows the perturbation added to the image of 3.
- Here each of the 3 perturbation(middle column) has the L2-Norm of 3.96.
- The second row adds a perturbation but in a random direction, so it didn't really change the class label.
- The third row erases a part of the image using the perturbation resulting in an image which doesn't belong to any of the class.
- **We should notice that all the 3 changes happen with the same L2-Norm perturbation which measures 3.96**
- ***Lot of the times adversarial attacks perform larger perturbations.***
- ***There are so many pixels in the image that just small changes to the individual pixels can add up to relatively large vectors.***
- For larger datasets like ImageNet where there are more pixels for a given image, just a small amount of change to each pixel will result in change of vector to a very far distance in the vector space as measured by the L2-Norm.
- **This means that we can perform peturbations which are not perceptible by humans but have adversarial attacks**

- We should be able to generate adversarial examples which are really adversarial attacks. In the top row, we changed 3 to 7, and then the new image gets classified as 7, which is not an adversarial attack, since we have modified the input class to get classified into 7.
- **During adversarial attacks, we want to fool the network and not just change the input class. E.g. The image should look like 3 but should get classified as 7.**




- One of the way to create an adversarial example is to add the gradient of the input.
- We can constraint the perturbation by using **Max Norm**, which says that no pixel can change more than the set perturbation $\epsilon$.
- We take the gradient of the cost wrt to the input and then take the sign of the gradient.
- The sign is essentially enforcing the maxnorm constraint, and the sign signifies whether we have to add epsilon or subtract epsilon in order to hurt the network.
- This approach is similar to Taylor Series Approximation of first order where we want to maximize the cost following the maxnorm constraint, this technique is called as Fast Gradient Sign Method.
- **Adversarial Examples generalize over cross-model, cross-dataset** because machine learning model learn a generalized function for a task.

# Transferability Attack






# Practical Attacks

- Fool real classifiers trained by remotely hosted API (MetaMind, Amazon, Google)
- Fool malware detector networks
- Display adversarial examples in the physical world and fool machine learning systems that perceive them through a camera

# Failed defenses

Generative pretraining

Removing perturbation with an autoencoder

Adding noise at test time

Ensembles

Confidence-reducing perturbation at test time

Error correcting codes

Multiple glimpses

Weight decay

Double backprop

Adding noise at train time

Various non-linear units

Dropout



# Adversarial Training of other Models

- Linear models: SVM / linear regression cannot learn a step function, so adversarial training is less useful, very similar to weight decay
- $k$-NN: adversarial training is prone to overfitting.
- Takeway: neural nets can actually become more secure than other models. *Adversarially trained neural nets have the best empirical success rate on adversarial examples of any machine learning mode*.

# Conclusion

- Attacking is easy
- Defending is difficult
- Adversarial training provides regularization and semi-supervised learning
- The out-of-domain input problem is a bottleneck for model-based optimization generally

Stanford University