INTRODUCTION

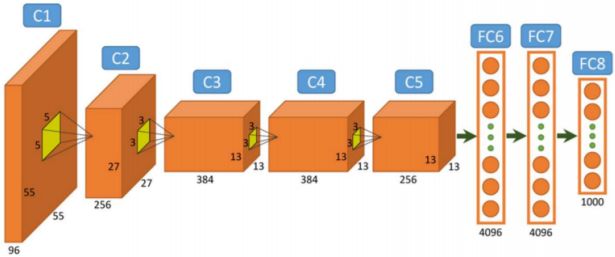
IN 5400— Network visualizaton and adverserial fooling

Anne Solberg

20.03.2019

University of Oslo

What does the layers learn?



- What does the intermediate features look like?
- How can we get confidence in what the network learns?

Plan for today: making new use of gradients

- Filter visualization
- Visualize features in the last layers
- Visualize what triggers a certain node in a given layer
- Salicency maps/class activation maps
- Layerwise relevance propagation
- Adverserial fooling

Reading material and relevant video links:

- No good text for this subject, research papers are the best source.
- Relevant papers are linked in the following slides.
- Visualization: partly covered by CS 231n Lecture 12 2017.
- Adverserial fooling: Ian Goodfellow's lecture `https://www.youtube.com/watch?v=CIfsB_EYsVI&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv`
- Focus: overview of the selected methods and how they use gradients

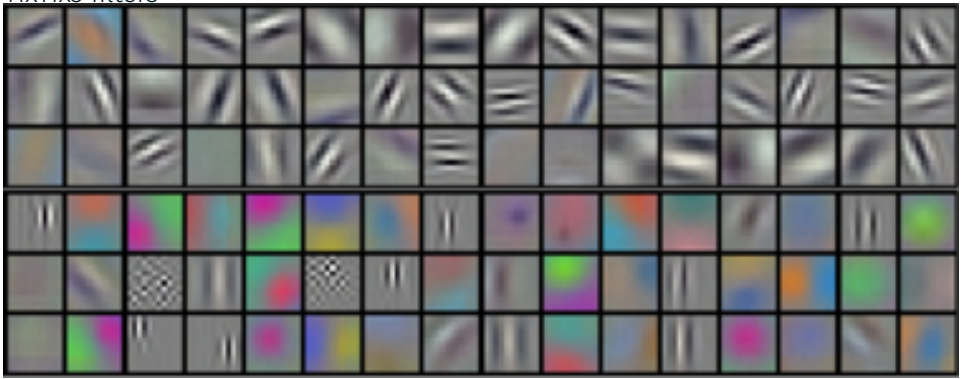
VISUALIZING THE FILTERS DIRECTLY

Can we visualize the filters directly?

- Useful for the first layer(s), but most useful for larger filter kernels like AlexNet
- More difficult deeper into the network, when we have a large number of small filter kernels in each layer.
- Check it out at https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Filters in the first layer of AlexNet

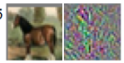
- Visualizing the filters of the first layer or AlexNet
- 11x11x3 filters



Visualizing filters in layer1 from LeNet trained on CIFAR layer

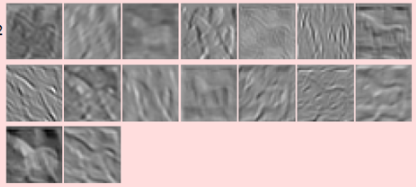
input (32x32x3)
max activation: 0.45686, min: -0.45295
max gradient: 0.01512, min: -0.01435

Activations:

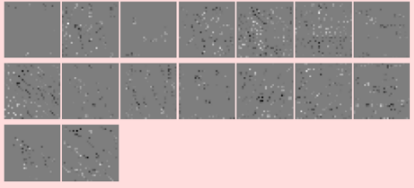


conv (32x32x16)
filter size 5x5x3, stride 1
max activation: 2.73235, min: -3.59482
max gradient: 0.00376, min: -0.0037
parameters: 16x5x5x3+16 = 1216

Activations:

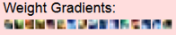


Activation Gradients:



Weights:

Weight Gradients:

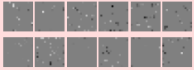


Visualizing filters in layer 2 from LeNet trained on CIFAR layer

conv (16x16x20)
filter size 5x5x16, stride 1
max activation: 3.9723, min: -7.03267
max gradient: 0.00261, min: -0.00266
parameters: 20x5x5x16+20 = 8020

Activations:

Activation Gradients:



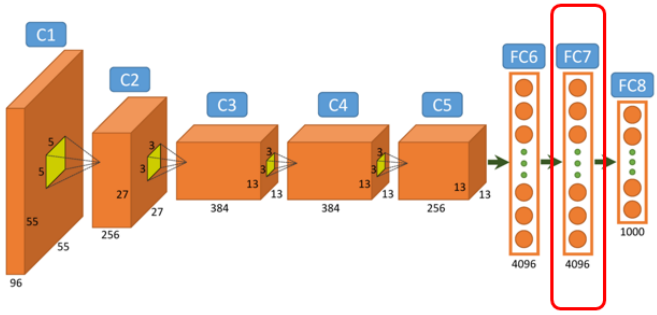
Weights:

Weight Gradients:

VISUALIZING FEATURES IN THE LAST LAYER

Visualizing the features in the final layer

Use AlexNet as an example:



- Can visualize the 4096 features in different ways:
  - Do a PCA (principal component analysis) down to 3 PCA components and visualize as RGB
  - Use t-SNE (later lecture) to visualize
  - Demonstrate similarity for an image by finding the nearest neighboring images in feature space.

Nearest neighbors in feature space for a given image

- Given a pretrained model and an example image
- Propagate the example image to the second last fully connected layer (before the classifier)
- Find the k nearest neighbors from the training set in the 4096-d feature space

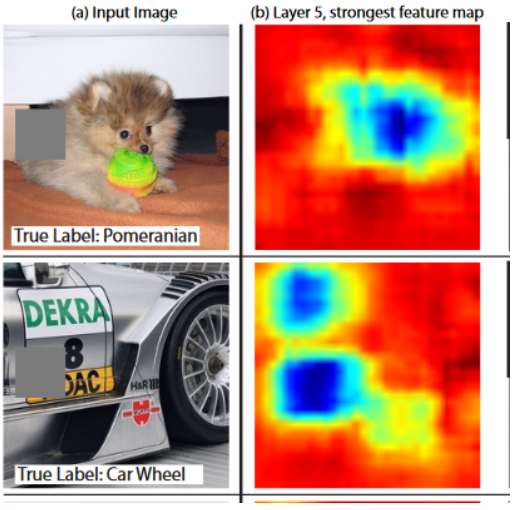
- This will tell about invariance etc.

OCCLUSION EXPERIMENTS

Occlusion experiments

- Create a small patch of zeros
- Slide this patch over the input image to zero out different parts of the image
- Classify each image
- Record how much the probability for the best class changes with and without a zero-pad at the location
- Reference: https://arxiv.org/pdf/1311.2901.pdf

Occlusion - examples



Figures from https://arxiv.org/pdf/1311.2901.pdf

SALIENCY/CLASS ACTIVATION MAPS

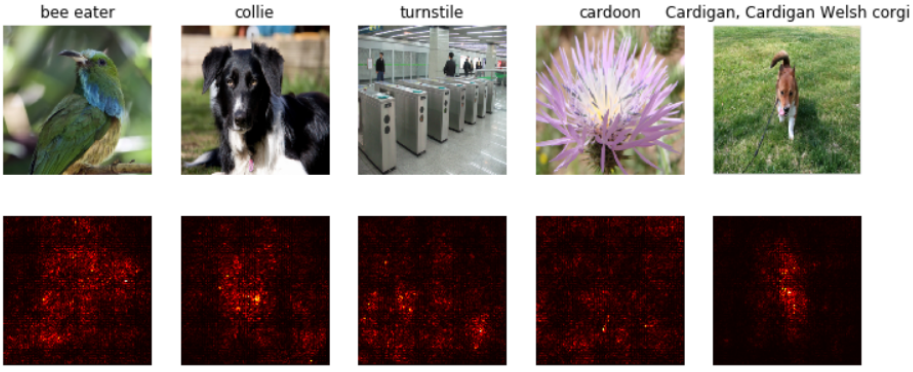
Introduction - backpropagation with respect to the image

- Training: given a likelihood function that measures the fit between the predicted class score and the true labels, use backpropagation to find the weights.
- Backpropagation with respect to the image pixels
  - Given a pretrained model and a given class
  - Keep the weights constant
  - Based on the likelihood with respect to the input image pixels, use gradient descent updates to find an input image that maximize the likelihood

Image saliency - by gradient explanations

- Given a pretrained model
- Given an input image
- Compute the score for that class (before normalization to probabilities)
- Keep the weights, zero out the gradients for the other classes.
- Backpropagate the gradients from the score for that class with respect to each input pixel.
- Take max over input channels to get a single scalar value for each pixel
- Image saliency gives a pixel-by-pixel view of how sensitive the class-specific score is to each pixel in the input image.
- Reference: https://arxiv.org/pdf/1312.6034.pdf

Saliency - examples



Saliency is a univariate number - but visualized using a heatmap colortable here.

GradCAM - visualizing class activation maps

- Main principle: visualize the gradient information in the last convolutional layer
  - Why - Fully connected layers loose the spatial information
- Given a class c
- Start with the score (non-normalized) s_c before softmax
- Compute the gradient of this with respect to the feature maps of the last conv-layer
- Use global average pooling for all locations to get a weight
- Sum these over all nodes in the layer
- Guided GradCAM combines this with Guided Backpropagation using multiplication
- Reference: https://arxiv.org/pdf/1610.02391.pdf

GradCAM - details

- Given a class c, a pretrained model, and an input image
- Select layer (normally the last convolutional)
- Forward propagate to the the score s_c before softmax.
- Zero out gradients for all other classes.
- Do backward pass to get the gradients of s_c with respcect to the activations for node k A_k , and save the gradient (hook layer in PyTorch)
- Get a score for how important node $k$is for class c by doing Global Average Pooling over the spatial dimensions of the tensor.

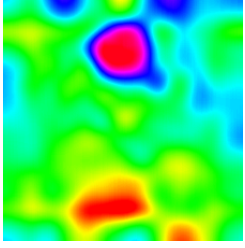
$$\alpha_k^c = \frac{1}{Z} \sum_h \sum_w \frac{\partial s_c}{\partial A_{hw}^k}$$

- Get the GradCAM score by summing over the nodes with positive activation:

$$GradCAM = ReLU\left(\sum_k \alpha_k^c A_k\right)$$

GradCAM - examples

Original

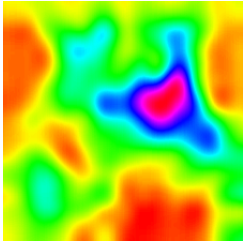


GradCAM heatmap

GradCAM overlaid image

GradCAM - examples

Original



GradCAM heatmap

GradCAM overlaid image

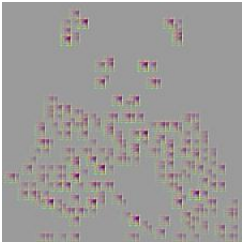
Guided GradCAM

Guided backpropagation with respect to a given node

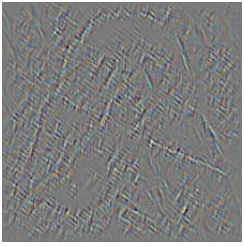
- Guided backprop: secton 3.4 in : https://arxiv.org/abs/1412.6806
- Select a layer and a node in that layer
- Forward propagate an image
- Set all gradient to zero, except for the selected node
- Backpropagate back to the input, but set negative gradients to zero

Guided backprop - one filter for several layers

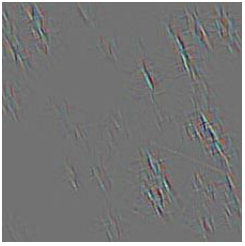
Original



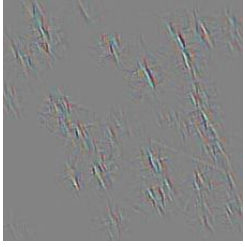
Filter 1 layer 2



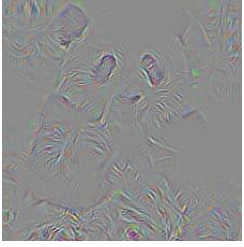
Filter 1 layer 3



Filter 1 layer 4



Filter 1 layer 5

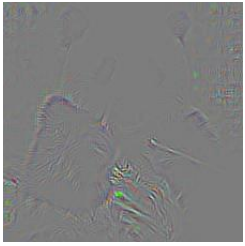


Filter 1 layer 7

Filter 1 layer 9

Filter 1 layer 15

Guided backprop - several filters for layer 15



Filter 1 layer 15

Filter 5 layer 15

Filter 15 layer 15

Filter 40 layer 15

Filter 15 layer 15

Filter 30 layer 15

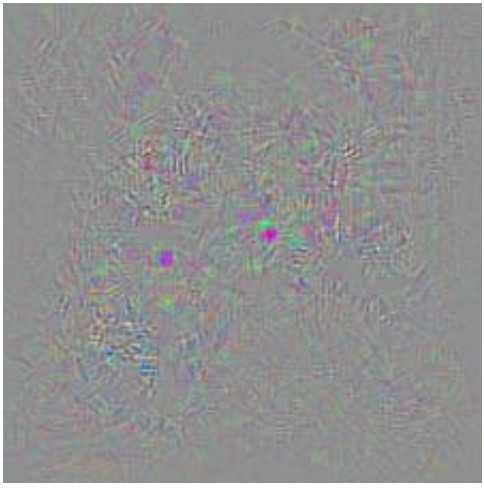
Filter 35 layer 15

Filter 45 layer 15

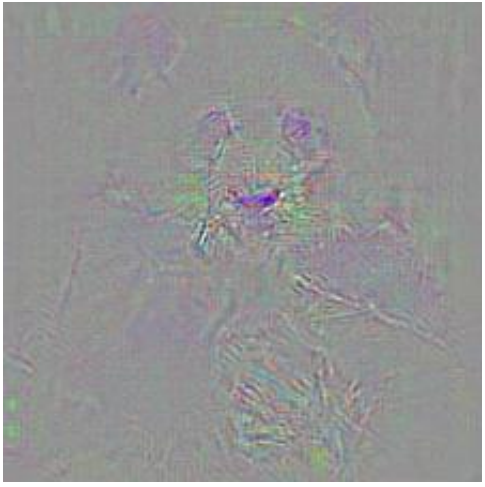
SmoothGrad - adding noise and averaging over gradients

- Very simple idea: add noise to a given gradient image/saliency image.
- Average the saliency map/gradient map over n noisy images.
- Reference: https://arxiv.org/pdf/1706.03825.pdf

Smoothgrad - results



Vanilla gradient backpropagation (saliency map)



Smooth Grad

Gradient ascent on the image for visualization

- Start from a random image
- Given a pretrained model and a target class
- Use gradient ascent to create an image $I*$ that maximize the score s (unnormalized probabilities) for the target class y

$$I* = \arg\max_I (s_y(I*) - R(I)) \tag{1}$$

- $R(I)$ is a regularization term to get the image to look like a natural image, and not just noise.
- ($R(I)$ should include L2-decay, Gaussian blur at certain iterations, and clipping pixels with small norm/contributions.
- See more details in https://arxiv.org/abs/1506.06579
- Implement this in the weekly exercise

LAYERWISE RELEVANCE PROPAGATION

Layerwise relevance propagation

- Layerwise backpropagation redistributes the class score back to the inputs in a somewhat more robust way than saliency or class activation maps
- Good sources for this:
  - Good overview paper: https://arxiv.org/abs/1706.07979
  - Tutorial for this method: `heatmapping.org`
- Consider the relevance R_k for node k in layer and how to backpropagate this to a node j in the previos layer R_j
- The relevance should be conserved: $\sum_j R_{j \leftarrow k} = R_k$
- Global conservation will sum this up over all layers.
- This technique can also be used for other types of data than images.

Introduction to layer propagation rules

- Let all neurons be described by the activation: $a_k = \sigma(\sum_j a_j w_{jk} + b_k)$ with activation function σ.
- Relevance backpropagation from layer k to j considers both positive and negative contributions:

$$R_j = \sum_k (\alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-}) R_k$$

- α and β are choosen as positive numbers such that $\alpha - \beta = 1$
- A simple variant is $\alpha_1 \beta_0$, which only considers positive contributions
- Another variant is $\alpha_2 \beta_1$
- Pooling layers can either use a winner-take-all strategy or distribute proportional to neuron activations in the pool
- Normalization layers can either be ignored or a special rule is used.

Layerwise relevance propagation - MNIST results



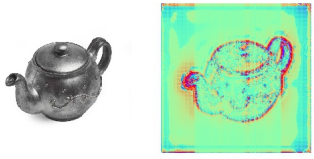




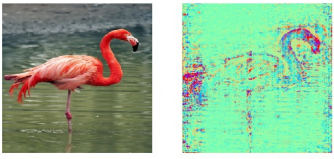



Layerwise relevance propagation - trained on VGG results

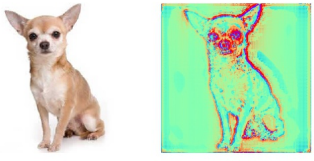
('n04398044', 'teapot', 0.5020969)



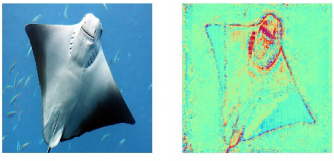
('n02007558', 'flamingo', 0.99997103)



('n02085620', 'Chihuahua', 0.99573576)



('n01498041', 'stingray', 0.4982025)



Why do we need explainable deep learning?

- Verification of the system
  - Example: to use deep learning for medical application the medical experts needs to trust it. Accuracy is not enough to decide to use a model.
- Understand weaknesses to improve the system
  - Example: are there any biases in the dataset? Is information included by a mistake?
- Learning from the system
  - Example: Has the game system (chess/go) learning something we do not know?
- Compliance to legislation?
  - Who is responsible for a decision?

Example of non-robust class activation maps

- This example of which part of the image is important illustrates the concept well!
- Reference: https://ieeexplore.ieee.org/document/7780687

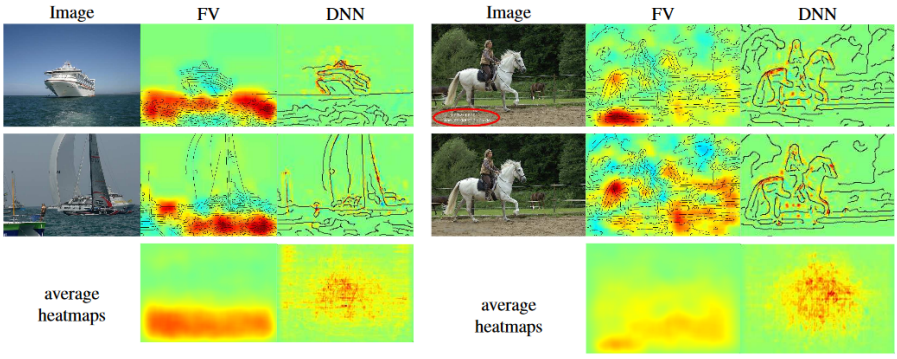


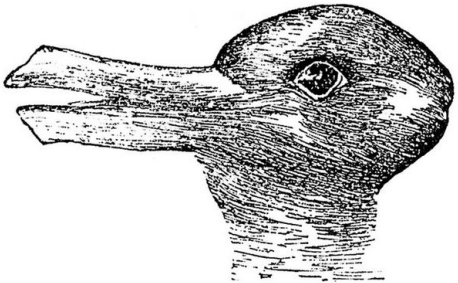
Figure 5. Top: Images of the classes "boat" and "horse", processed by the FV and DNN models and heatmapped using LRP. Bottom: Average heatmap scores over a random sample (of size between 47 and 177) of the distribution for each class and model. On the second image of class "horse", the copyright tag (marked by the red ellipse) has been removed.

ADVERSERIAL FOOLING

Introduction to adverserial fooling

- Adverserial images - what are they and why do they happen?
- How can they be used to compromise machine learning systems?
- How can we avoid this?
- How can they be used improve the performance, even without fooling the system.

Duck or rabbit?



Young or old lady?

See at lot more and learn about it here

Fooling the network

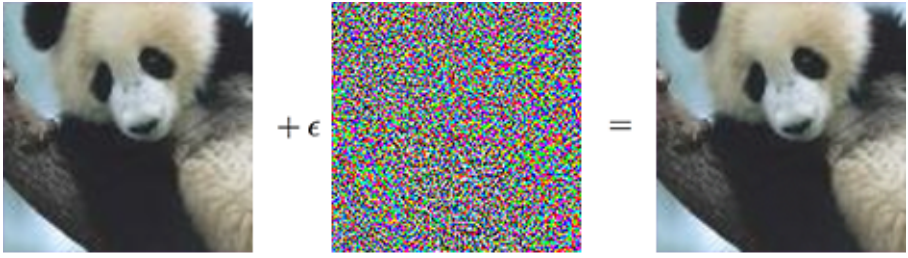


Image from https://arxiv.org/abs/1412.6572

- Original image: 8-bit RGB, internal representation 32bit float
- The differences between the images are so small that they are quantized into the same 8-bit representations, but they still fool the network

Linear explanation of adverserial examples

- Consider a sample x stored given a given precision (e.g. 8-bit image channels).
- Consider a small perturbation η of x, $\tilde{x} = x + \eta$ and assume η is so small that the representation of $\tilde{x}$ is the same as x.
- We would expect a classifier to assign x and $\tilde{x}$ to the same class as long as $||\eta||_\infty < \epsilon$, where ϵ is small enough to be discarded by the storage representation of x.
- Consider the dot product

$$w^T \tilde{x} = w^T x + w^T \eta$$

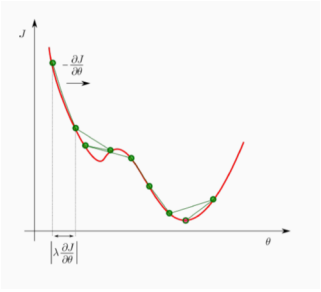
- The adverserial perturbation grows the activation by $w^T \eta$.
- For high dimensions, we can **add infinitesimal changes to the input that add up to large changes in the output** .
- For more details, see https://arxiv.org/abs/1412.6572
- This is valid for linear models trained with gradient descent, like softmax classification, logistic regression and support vector machines.

Discussion of non-linear models

- ReLU is close to a linear model
- Even tanh or LSTM-models are not that different from linear model
- By normalization we take care to operate close to the linear mode where gradient updates works bests

Adverserial perturbations of x

- Let w be the parameters of a model with input x and true class y_{true}.
- Given a likelihood function $L(x, w)$
- We want to create an adverserial perturbation of x in order to force the network to misclassify x as $y' \neq y_{\text{true}}$.
- We classify a sample to the class that maximize $L(w, x, y')$.



The sign of the gradient of L tells us if we should increase or decrease x to increase $L(w, x)$.

Fast gradient sign method for generating (FSGM) adverserial examples

- An adverserial example with ϵ-bound ϵ, given a norm p is classified to a different class than the target y and satisfies

$$||\tilde{x} - x||_p < \epsilon$$

- L^0-norm: bounds the number of pixels in $\tilde{x}$ that can be modified from x.
- L^2-norm: bounds the total squared distance between $\tilde{x}$ and x.
- L^∞-norm: bounds the change for every pixel in x.

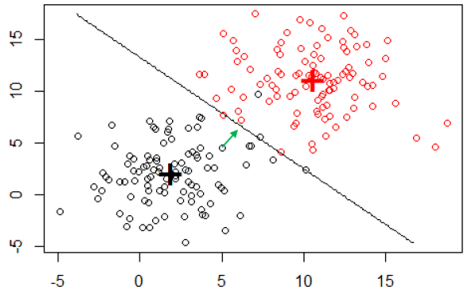
Fast gradient sign method

- When the input dimension is large, changing each element in x by ϵ yields a perturbation η (such that $||\eta||_\infty = \epsilon$) which can significantly change the inner product $w^T x$.
- The fast gradient sign method obtains the perturbations by setting

$$\eta = \epsilon sign(\nabla_x L(x, w))$$

- Update x given learning rate λ as:

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



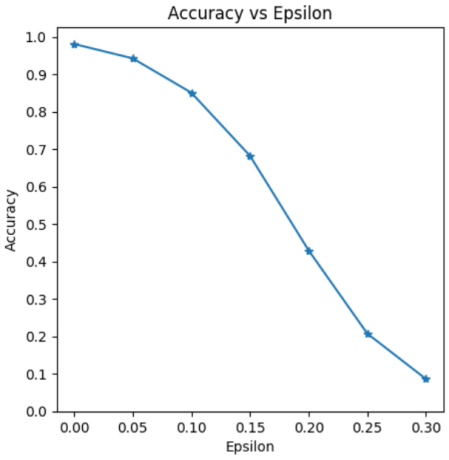
We want the given sample to be classifed to the red class. The weights are fixed, but we modify the sample.

Implementing FSGM

- Note that this can be computed by backpropagation with respect to the image, keeping the weights constant.
- Define the model/network architecture under attack, train it and set the flag for computing the gradients of the data.
- Select the image x_{fool} from the training set that we want to modify, and a target class y'.
- Iterate over the following steps:
  - Forward propage x_{fool} to get the current scores for all classes
  - Find the index of the current best class y_{curr}
  - If $y_{\text{curr}} = y'$, stop
  - Backpropagate x_{fool} with respect to the data to get g
  - Else update x_{fool} by taking a step $\lambda \epsilon sign(\nabla_x L(x, w))$

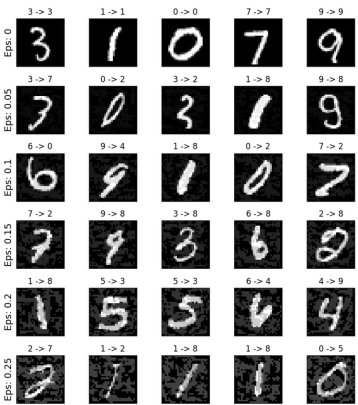
MNIST Adverserial attacks over a range of ϵ values

Try $\epsilon = 0, ..., 0.25$, attack the test set, and record the percentage of correctly classified images as a funciton of epsilon.



MNIST Adverserial attacks over a range of ϵ values

Example of MNIST adverserial images

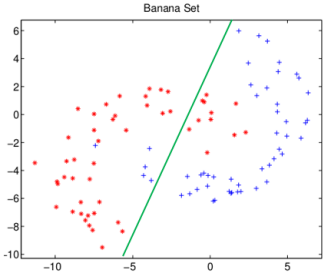


- MNIST images are small 32x32 images and we see the pertubations quite clear.
- With larger RGB-images (increasing dimension), the pertubation we need for each pixel to obtain a misclassification is much smaller
- For normal image sizes, adverserial images can look very similar to the original, while they have the same L2-distance to the original as the MNIST images here

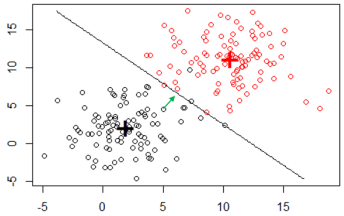
Some types of classification errors

Is any of these more likely to get adverserial errors?

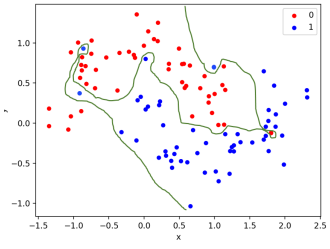
Underfitted model



Decent fit of model



Underfitted model



Adverserial training

- Regular data augmentation includes transformations like rotations, scaling etc. that naturally can occur in test sets.
- Training on adverserial examples add images that do not occur naturally, but expose the flaws in the network.
- A modified cost function is used:

$$\tilde{L}(w, x, y) = \alpha J(w, x, y) + (1 - \alpha) J(w, x + \epsilon sign(\nabla_x J(w, x, y))$$

- Increasing network capacity (number of nodes) and using early stopping might be useful

Adverserial examples for other domains

- Fooling OCR with text images https://arxiv.org/abs/1802.05385
- Fooling speech recognition
  https://arxiv.org/abs/1707.05373https://arxiv.org/abs/1707.05373

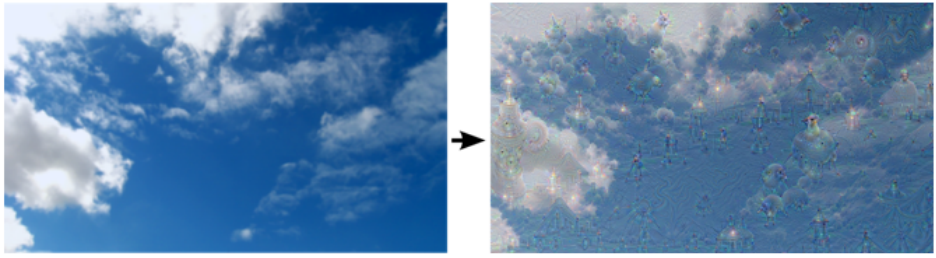
Discussion of adverserial fooling

- Requires access to the images to fool a trained network.
- Adverserial examples often generelize to similar models.
- Ideally we should also have access to the model, but we can also train a set of common CNN-models ourself, then attach an unknown model using an adverserial image that fools our trained models.
- Adverserial training is a type of regularization and can increase the robustness of the model.
- Adverserial fooling is also a problem in linear models, but adverserial training does not work well for linear models.

Just for fun: Deep Dream

- Read more: `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`
- Start with either a noise image or a natural image.
- Forward propagate the image to a given layer.
- Modify the gradient of this layer to equal its activation : see more of what the layer sees
- Add some tricks
- Backward propagate and update image

Deep dreams



"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

More fun: neural style transfer

- Texture synthesis using deep learning is really effective.
- Take e.g. an image and a painting.
- Combine the style of the painting onto the image by optimizing a content loss and a style loss
- Read `https://arxiv.org/abs/1508.06576`
- A faster variant `https://arxiv.org/abs/1603.08155`

Fun: neural style transfer

LEARNING GOALS

Learning goals

- Understand the need to be able to visualize the network
- Understand the limitations of visualizing the filters directly
- Understand heatmaps like class activation maps, saliency maps, and know about layerwise relevance propagation
- Know the principles and goals of guided backprop
- Understand how adverserial images are created, and what adverserial traning is, and what models are suspect to being fooled.

Questions?