# Understanding DNNs

Presented by
Adel bibi & Modar Alfadly
Prepared by Modar

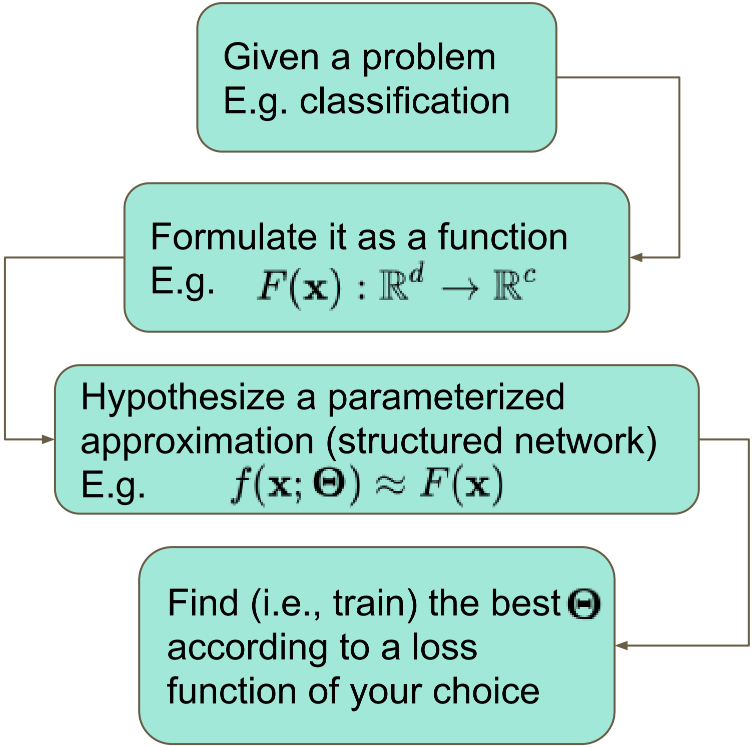
# Outline

- Introduction
- Geometrical Study [with Modar]
- Probabilistic Study [with Adel]
- References

# Introduction

- Standard Workflow Pipeline in Deep Learning
- Training DNNs Using Gradient Descent
- Universal Approximation Theorem
- Deeper vs. Wider Argument

# Standard Workflow Pipeline in Deep Learning




- The input has some distribution i.e., $\mathbf{x} \sim \mu$ (e.g., natural images)
- A **good enough** parameterized model should approximate the original function for most samples in the domain
- Most DNNs are constructed as a **hierarchy of layers**
- **Each layer** is a small parameterized function that might be followed by an activation function (e.g., ReLU or Sigmoid)

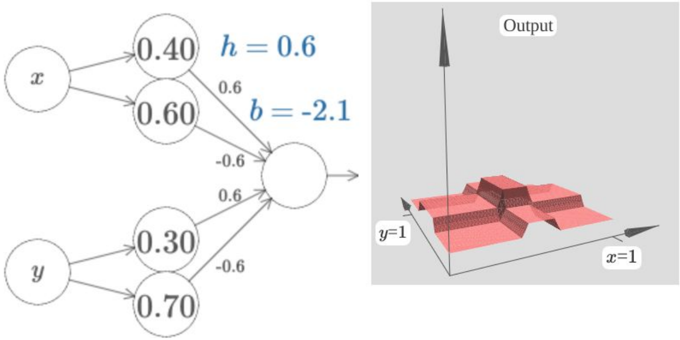
# Training DNNs Using Gradient Descent

1. Start with initial parameters $\mathbf{\Theta}_0$ and learning rate $\alpha$
2. Let $k \leftarrow 0$
3. Compute the loss of all the training data $\delta(\mathbf{X}, \mathbf{y}; \mathbf{\Theta}_k)$
4. Compute the partial subgradients of all the parameters $\frac{\partial}{\partial \mathbf{\Theta}_k} \delta(\mathbf{X}, \mathbf{y}; \mathbf{\Theta}_k)$
5. Backpropagate to all the parameters $\mathbf{\Theta}_{k+1} \leftarrow \mathbf{\Theta}_k - \alpha \frac{\partial}{\partial \mathbf{\Theta}_k} \delta(\mathbf{X}, \mathbf{y}; \mathbf{\Theta}_k)$
6. Let $k \leftarrow k + 1$
7. Change the learning rate if desired
8. Repeat steps 3-7 until some stopping criteria

**Linearization:** $f(\mathbf{x}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a})^T (\mathbf{x} - \mathbf{a})$
Linearize the loss around the parameters $\delta(\mathbf{X}, \mathbf{y}; \mathbf{\Omega}_k) \approx \delta(\mathbf{X}, \mathbf{y}; \mathbf{\Theta}_k) + \frac{\partial}{\partial \mathbf{\Theta}_k} \delta(\mathbf{X}, \mathbf{y}; \mathbf{\Theta}_k)^T (\mathbf{\Omega}_k - \mathbf{\Theta}_k)$
Then, move opposite to the gradient to decrease the loss

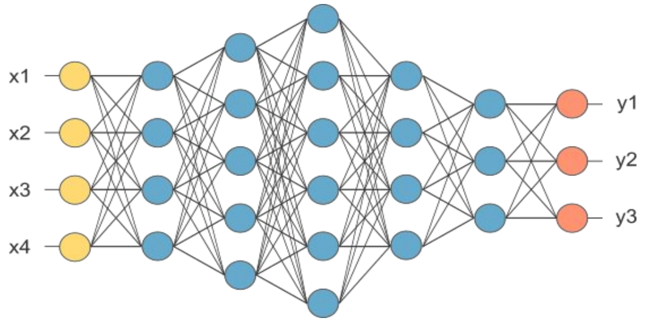
# Universal Approximation Theorem

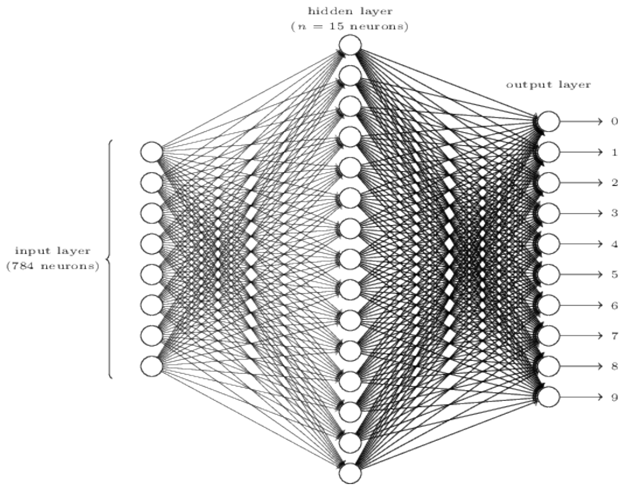
- Any arbitrary continuous function can be **approximated** effectively using a feed-forward neural network (i.e., a multilayer perceptron) with a single hidden layer, under mild assumptions on the activation function [1].
- A visual and interactive proof to this theorem is presented in [2,3].



# Deeper vs. Wider Argument

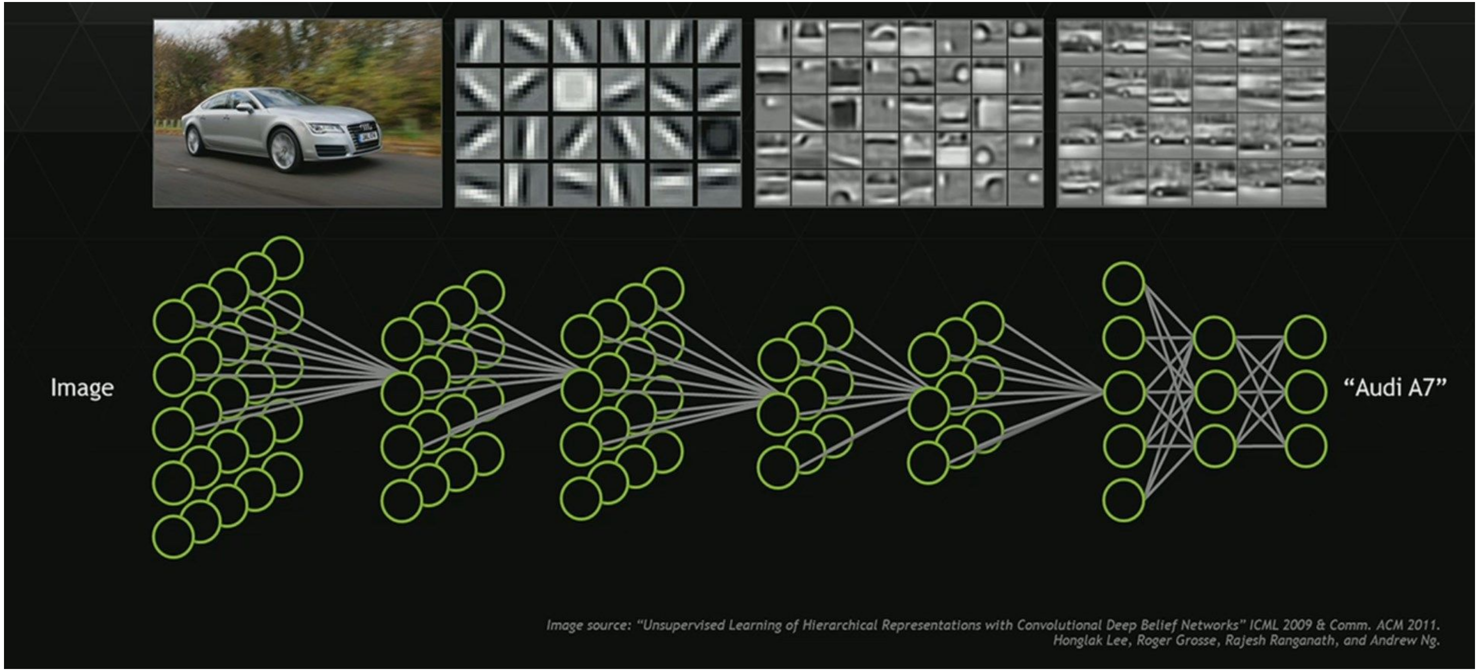
- Is it better to go deeper or wider? [4]
  - **Training difficulty** (e.g., vanishing gradients)
  - **Deployment restrictions** (e.g. high input dimensionality)
- Try different structures with Tensorflow Playground [here]



# Deeper vs. Wider Argument

Example of high dimensional input and proposed solution (CNNs)

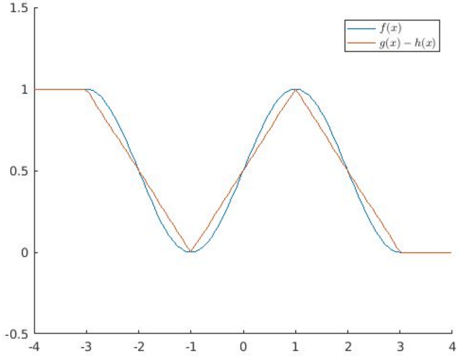


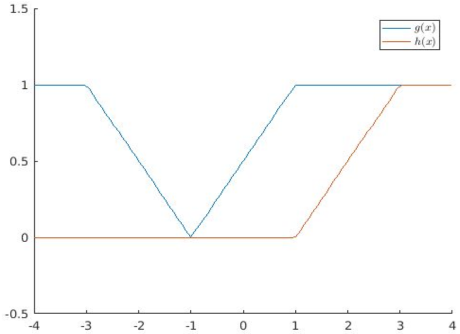
# Geometrical Study [with Modar]

- Continuous Piecewise Linear DNNs
- Gradient Images for PL-DNNs
- Sensitivity Analysis of PL-DNNs
- Adversarial Examples for PL-DNNs

# Continuous Piecewise Linear DNNs

Any Continuous PL function can be written as **a difference of only two** convex PL functions



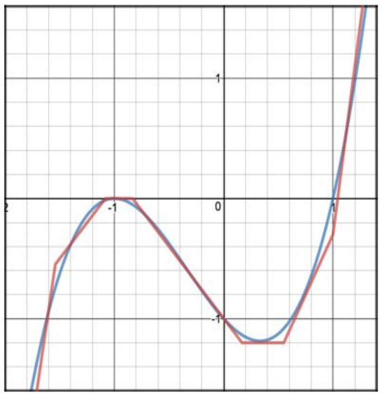

$$f(x) = \begin{cases} 1 & \text{if } x < -3 \\ \frac{1}{2}\sin(\frac{n}{2\pi}) + \frac{1}{2} & \text{if } -3 \le x < 3 \\ 0 & \text{if } 3 \le x \end{cases}$$

$$g(x) = \begin{cases} 1 & \text{if } x < -3 \\ \frac{2 - x - 3}{2} + 1 & \text{if } -3 \le x < -1 \\ \frac{x+1}{2} & \text{if } -1 \le x < 1 \\ 1 & \text{if } 1 \le x \end{cases}$$

$$h(x) = \begin{cases} 0 & \text{if } x < -1 \\ \frac{x+1}{2} & \text{if } -1 \le x < 1 \\ 1 & \text{if } 1 \le x \end{cases}$$

# Continuous Piecewise Linear DNNs

$$f(x) = x^3 + x^2 - x - 1$$



$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$\begin{aligned} Z(x) = &-n_1(x) - n_2(x) - n_3(x) \\ &+ n_4(x) + n_5(x) + n_6(x) \end{aligned}$$

Example of a single hidden layer network

- Convex piecewise linear functions are defined as $f(\mathbf{x}) = \max_{i \in [1,m]} \{\mathbf{a}_i^T \mathbf{x}\}$
- Most **DNN layers** are piecewise linear [5] E.g., ReLU, MaxPool, Conv, FC
- The **composition** of two PL functions is PL
- Thus, Most DNNs are piecewise linear
- Note that, **Softmax is not PL**

# Continuous Piecewise Linear DNNs

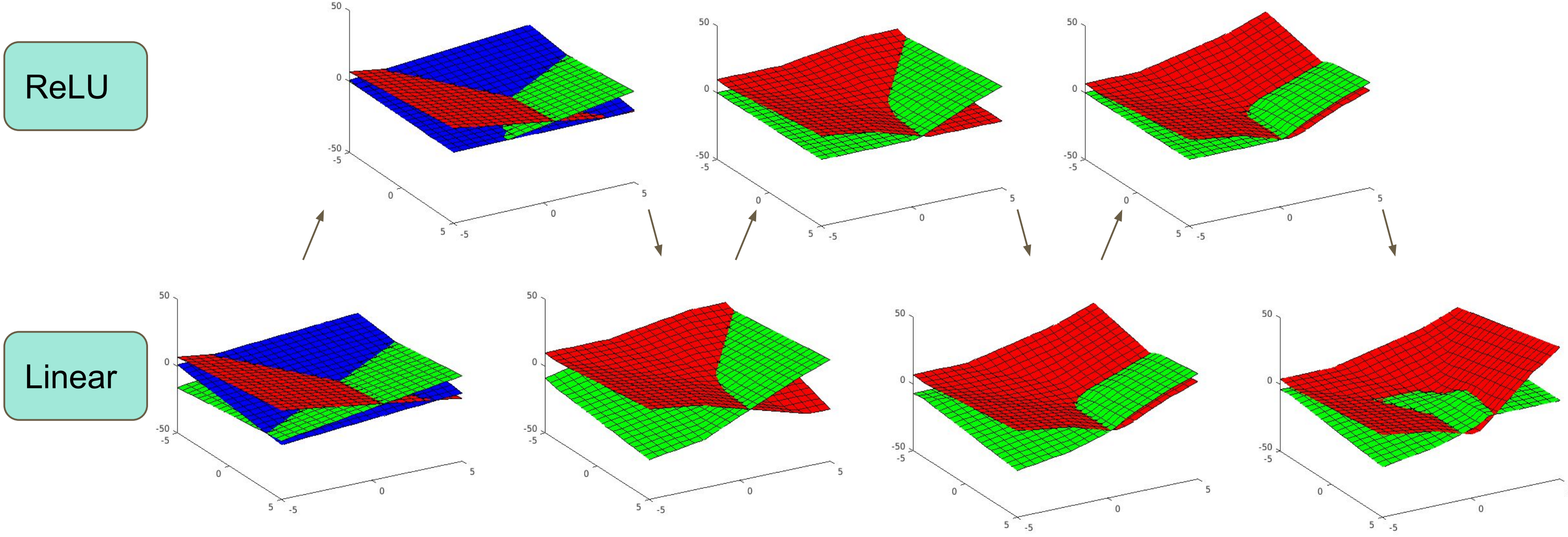
- For a classifier PL-DNN, let us recursively define $\forall i \in (0, L]$

| | |
|---|---|
| The network as a function | $f : \mathbb{R}^{m^0} \rightarrow \mathbb{R}^{m^L} \Rightarrow f(\mathbf{x}; \mathbf{\Theta}) = l^L(\mathbf{x})$ |
| Layer output: *activation(linear(input))* | $\Lambda^i(\mathbf{x}) = A^i(l^i(\mathbf{x}))$ |
| Linear layer | $l^i(\mathbf{x}) = \mathbf{W}^i \Lambda^{i-1}(\mathbf{x}) + \mathbf{b}^i$ |
| Base cases | $\mathbf{W}^0 = \mathbf{I}_{m_0}, \mathbf{b}^0 = \mathbf{1}_{m_0} \Rightarrow \Lambda^0(\mathbf{x}) = \mathbf{x}$ |
| Such that | $\Lambda^i : \mathbb{R}^{m^{i-1}} \rightarrow \mathbb{R}^{m^i}, \mathbf{W}^i \in \mathbb{R}^{m^i \times m^{i-1}}, \mathbf{b}^i \in \mathbb{R}^{m^i}$ |

- The parameters $\mathbf{\Theta}$ is the set $\{\mathbf{W}^i, \mathbf{b}^i | \forall i \in (0, L]\}$

# Continuous Piecewise Linear DNNs

- Example of three hidden-layers network on 2D input



# Continuous Piecewise Linear DNNs

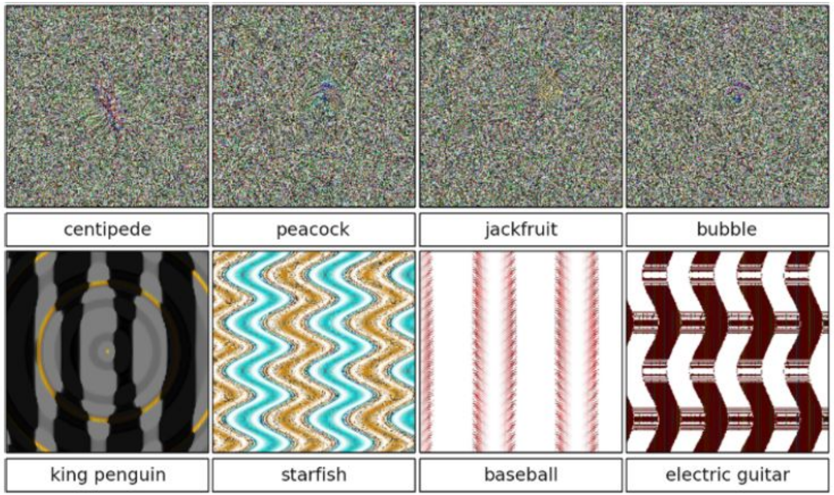
- PL-DNNs divides the input space into convex polyhedrons.
- There is a strong influence of the number of nonlinear layers and the number of wide layers on the complexity and expressivity of the network.
- A tight upper bound exists for number of knots neural networks that has input dimension of one and ReLU activations [15].

$$N \leq \sum_{i=1}^{L} m^i \prod_{j=i+1}^{L} (m^j + 1)$$

- The number of piecewise linear regions grows exponentially with the number of layers [16].

# Continuous Piecewise Linear DNNs

- It is possible to generate unnatural looking images that are classified with high confidence to be belonging to a certain class
- Using genetic algorithm with direct and indirect encoding [14]



# Gradient Images for PL-DNNs

- Gradients are the directions of the steepest change
- Let us define the gradient with respect to the input recursively

| | |
|---|---|
| Gradient of images of subnetwork | $\frac{\partial \Lambda^i}{\partial \mathbf{x}} = \frac{\partial A^i}{\partial l^i} \frac{\partial l^i}{\partial \mathbf{x}}$ (Chain Rule) |
| Gradient images of subnetwork | $\frac{\partial l^i}{\partial \mathbf{x}} = \frac{\partial l^i}{\partial \Lambda^{i-1}} \frac{\partial \Lambda^{i-1}}{\partial \mathbf{x}}$ (Chain Rule) |
| Gradient of linear layer | $\frac{\partial l^i}{\partial \Lambda^{i-1}} = \mathbf{W}^i$ |
| Base cases | $\frac{\partial \Lambda^0}{\partial \mathbf{x}} = \mathbf{I}_{m^0}$ |
| Such that | $\frac{\partial \Lambda^i}{\partial \mathbf{x}} \in \mathbb{R}^{m^i \times m^0}, \frac{\partial A^i}{\partial l^i} \in \mathbb{R}^{m^i \times m^i}, \frac{\partial l^i}{\partial \mathbf{x}} \in \mathbb{R}^{m^i \times m^0}$ |

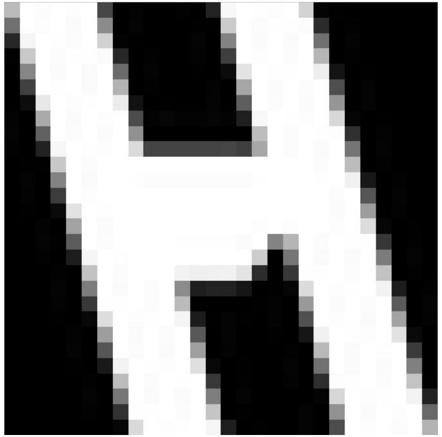
- Different activation functions have different gradients

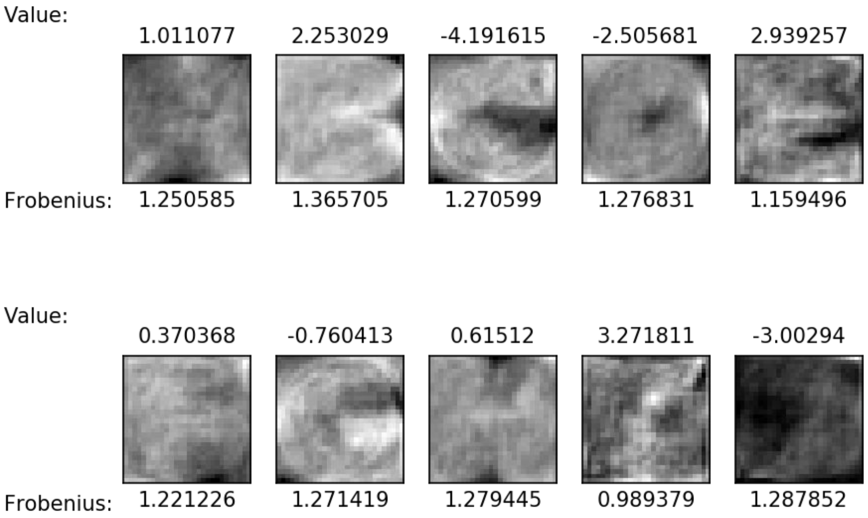
# Gradient Images for PL-CNNs

- Conv layers can be converted to FC layers (i.e., matrix-vector product)
- MaxPooling for a certain input can be converted as matrix-vector product
- We will consider the default activation for FC and Conv layers to be ReLU
- The default activation function for MaxPooling is the identity function
- For ReLU: $\frac{\partial A^i}{\partial l^i} = \mathbf{V}^i = diag(\mathbf{v}^i) \text{ s.t. } v^i_j = \begin{cases} 1 & \text{if } l^i_j(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$
- Therefore, the gradient images are given by $\frac{\partial f}{\partial \mathbf{x}} = \mathbf{W}^L \mathbf{V}^{L-1} \mathbf{W}^{L-1} \ldots \mathbf{V}^1 \mathbf{W}^1$
- The Vs job is to select specific rows of the Ws and make them zeros
- The Vs are functions of the input image while the Ws are constants
- This product contains the gradients of the linearization around the input

# Gradient Images for PL-CNNs

- Example of gradient images with Not-MNIST [6] and a Single layer ANN
- The output is 10 classes





# Sensitivity Analysis of PL-DNNs

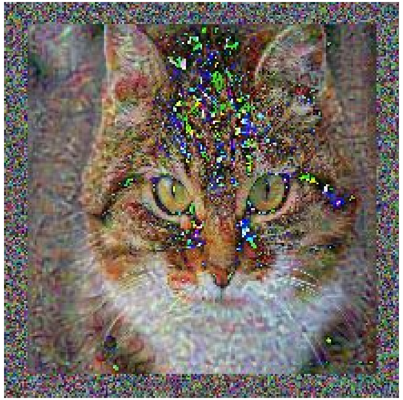
- How much can we change the input without changing the class label?
  - Move in a direction **orthogonal to all gradients** (i.e.,vector in the null space of $\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}, \boldsymbol{\Theta})$)
    - Any right singular vector that correspond to a zero singular value in the SVD
    - Minimum-energy solution $\mathbf{G} = \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}, \boldsymbol{\Theta}) \rightarrow \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}\mathbf{x}_0 - \mathbf{x}_0$ for any random $\mathbf{x}_0$
  - Move while keeping the **ordering** of final layer functions the same
    - Form this **convex polyhedral cone**
      Where $j_k = \begin{cases} k & \text{if } k < i \\ k+1 & \text{if } k > i \end{cases}$
      Find a point in this polyhedron

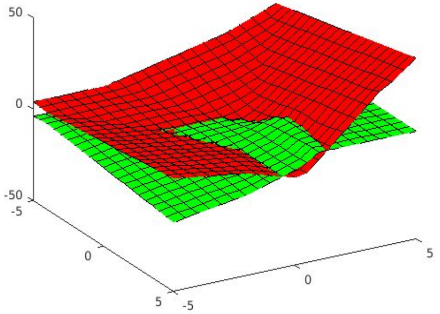
      $$\begin{bmatrix} (\nabla f_{j_1}(\mathbf{x}) - \nabla f_i(\mathbf{x}))^T \\ \vdots \\ \left(\nabla f_{j_{m^0-1}}(\mathbf{x}) - \nabla f_i(\mathbf{x})\right)^T \end{bmatrix} \mathbf{v} \leq \begin{bmatrix} f_i(\mathbf{x}) - f_{j_1}(\mathbf{x}) \\ \vdots \\ f_i(\mathbf{x}) - f_{j_{m^0-1}}(\mathbf{x}) \end{bmatrix} \Rightarrow \mathbf{A}\mathbf{v} \leq \mathbf{b}$$

      - Using Linear Programming (very expensive)
      - Start from the **intersection** then move inside $\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}(\mathbf{b} - \mathbf{c})$ s.t. $\mathbf{c} \geq \mathbf{0}$
    - With these techniques you have multiple points in a convex polyhedron
      Taking **any convex combination** of those points that is close enough to the original point will yield a point that has the same label as the original image

# Sensitivity Analysis of PL-DNNs

- Example of moving inside the convex polyhedral cone





# Sensitivity Analysis of PL-DNNs

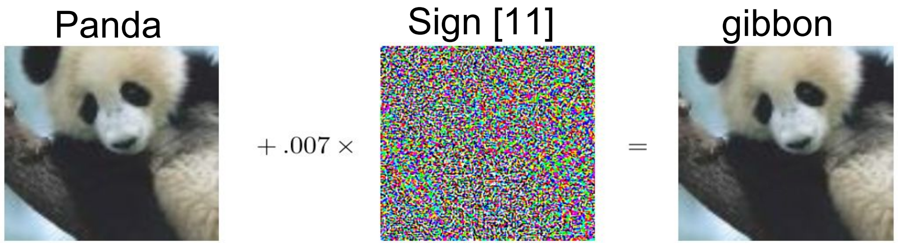
- **Lipschitz constant** tells us what is the effect of a small change in the input of a function to the output [7] $\forall \mathbf{x}, \mathbf{r} \| f(\mathbf{x}) - f(\mathbf{x} - \mathbf{r}) \|_2 \leq L \|\mathbf{r}\|_2$
- The smaller the constant the more smaller the change is going to be
- The Lipschitz constant of an FC layer bounded from above by the maximum singular value of the weights matrix
- The Lipschitz constant of a network is the product of its layers
- For a trained AlexNet [8] on imagenet dataset [9], Lipschitz constants are

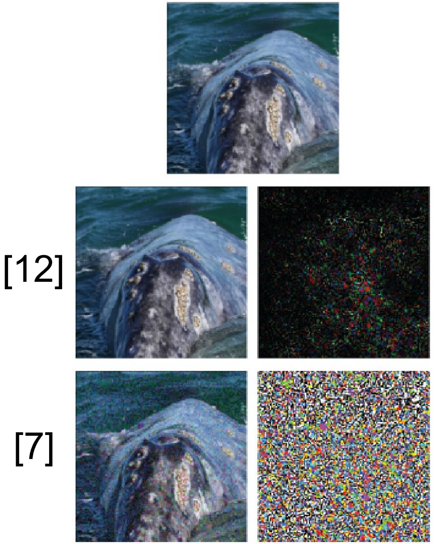
| Conv1 | Conv2 | Conv3 | Conv4 | Conv5 | FC6 | FC7 | FC8 |
|---|---|---|---|---|---|---|---|
| 2.75 | 10 | 7 | 7.5 | 11 | 3.12 | 4 | 4 |

- There is a way to train a network such that the Lipschitz constant is less than or equal to one for each layer to increase its robustness [10].

# Adversarial Examples for PL-DNNs

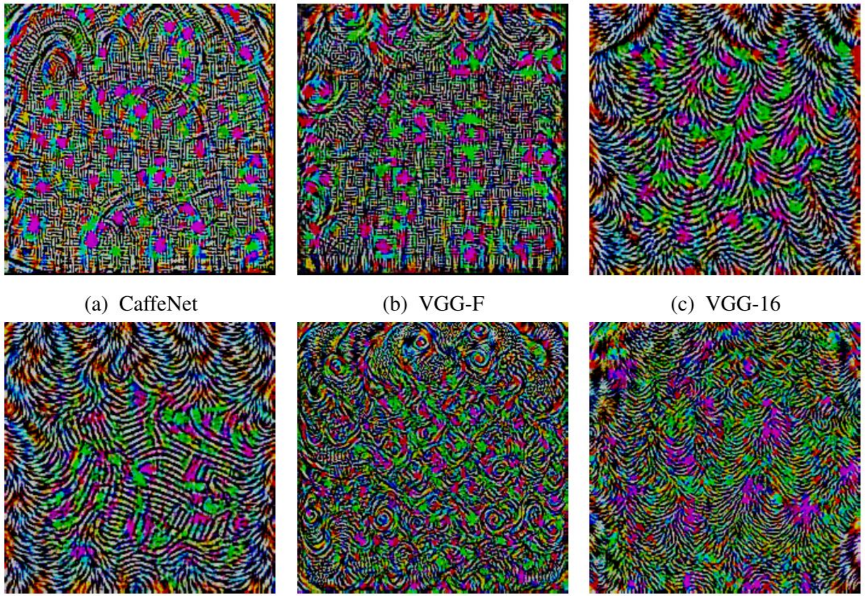
- Given an input image, add small perturbation to it to change its label.
  - Minimize the loss to a different label [7] (the minimization is done in very few steps)
  - Move along the sign of the gradient of the loss [11]
  - DeepFool: go outside the convex polyhedral cone [12]
  - Add an adversarial universal perturbation [13]



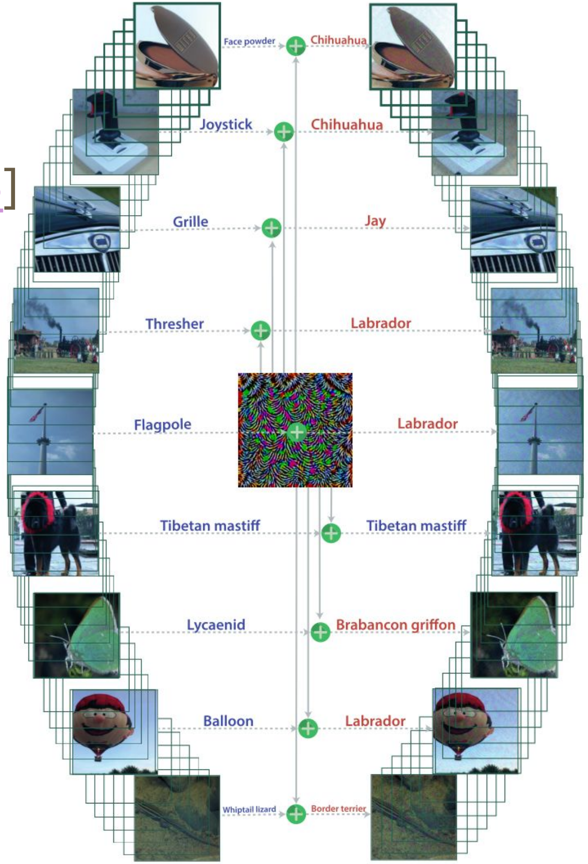
# Adversarial Examples for PL-DNNs

- More about universal adversarial perturbation [13]



(a) CaffeNet (b) VGG-F (c) VGG-16

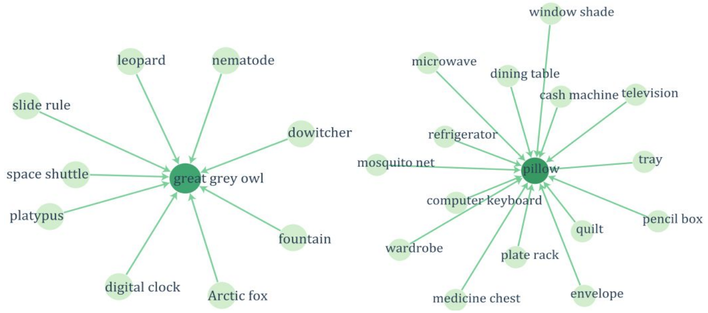
(d) VGG-19 (e) GoogLeNet (f) ResNet-152



# Adversarial Examples for PL-DNNs

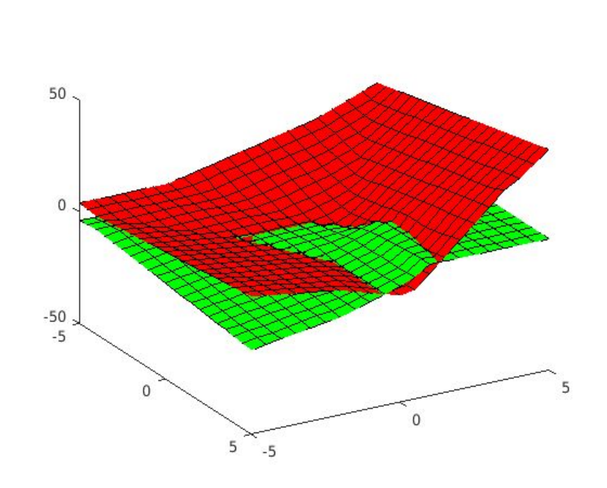
- These perturbation happen to be network-agnostic and there is even a high chance for an image to fool different networks by the same label
- Retraining with perturbed images doesn't help like filtering, JPEG compression and adversarial examples detector DNNs
- By studying the geometrical curvature of DNNs we can detect its adversarial examples [16].

| | VGG-F | CaffeNet | GoogLeNet | VGG-16 | VGG-19 | ResNet-152 |
|---|---|---|---|---|---|---|
| VGG-F | **93.7%** | 71.8% | 48.4% | 42.1% | 42.1% | 47.4 % |
| CaffeNet | 74.0% | **93.3%** | 47.7% | 39.9% | 39.9% | 48.0% |
| GoogLeNet | 46.2% | 43.8% | **78.9%** | 39.2% | 39.8% | 45.5% |
| VGG-16 | 63.4% | 55.8% | 56.5% | **78.3%** | 73.1% | 63.4% |
| VGG-19 | 64.0% | 57.2% | 53.6% | 73.5% | **77.8%** | 58.0% |
| ResNet-152 | 46.3% | 46.3% | 50.5% | 47.0% | 45.5% | **84.0%** |



# Suggested Reading

- Can neural networks solve any problem [2]?
- Universal adversarial perturbations [13].
- Classification regions of deep neural networks [16].



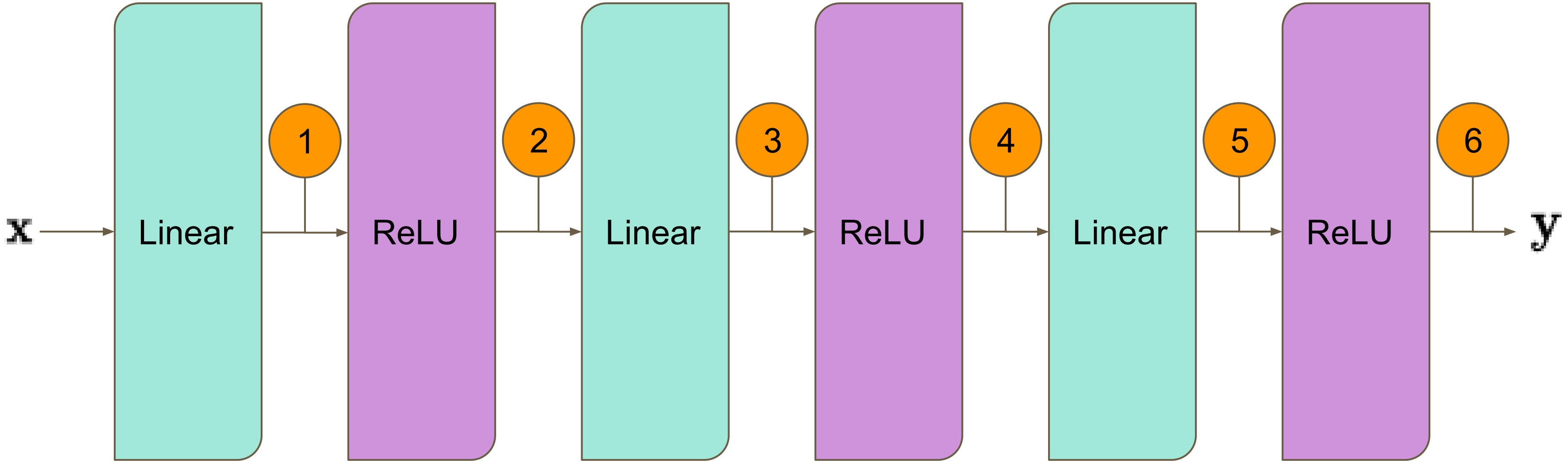
# Geometric Study ~ Conclusions

- DNNs are parametrized models that have high capacity to approximate continuous functions using different architectural choices (deep vs. wide).
- These models need to be trained with enough data from a certain distribution to be able to generalize well to new unseen examples.
- With the current training techniques there still appear blindspots to the model were adversarial examples live even after fine tuning on them.
- These adversarial samples appear to be universal with different DNNs.
- By studying the geometry of these constructions (i.e., PL-DNNs) we get insights on why these phenomenon occur and how to avoid them.
- We hope that we can use this knowledge to understand the capabilities and shortcoming of DNNs and how best to construct and train them.

# Probabilistic Study [with Adel]

- Statistical Analysis of Fully Connected Networks

# Statistical Analysis of Fully Connected Networks



# Thank You
# for
# Listening!

References list is on the next slide

# References


1. Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." Neural networks 4.2 (1991): 251-257.
2. Fortuner, Brendan. "Can neural networks solve any problem?" Medium. Towards Data Science, 07 Mar. 2017. Web. 21 June 2017.
3. Nielsen, M. A. (1970, January 01). Neural Networks and Deep Learning. Retrieved June 20, 2017, from http://neuralnetworksanddeeplearning.com/chap4.html
4. Pandey, Gaurav, and Ambedkar Dukkipati. "To go deep or wide in learning?." AISTATS. 2014.
5. Berrada, Leonard, Andrew Zisserman, and M. Pawan Kumar. "Trusting SVM for Piecewise Linear CNNs." arXiv preprint arXiv:1611.02185 (2016).
6. Bulatov, Yaroslav. "NotMNIST dataset." Machine Learning, etc. N.p., 01 Sept. 2011. Web. 21 June 2017.
7. Szegedy, Christian, et al. "Intriguing properties of neural networks." arXiv preprint arXiv:1312.6199 (2013).
8. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
9. Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.
10. Cisse, Moustapha, et al. "Parseval networks: Improving robustness to adversarial examples." arXiv preprint arXiv:1704.08847 (2017).
11. Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014).
12. Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
13. Moosavi-Dezfooli, Seyed-Mohsen, et al. "Universal adversarial perturbations." arXiv preprint arXiv:1610.08401 (2016).
14. Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
15. Chen, Kevin K. "The upper bound on knots in neural networks." arXiv preprint arXiv:1611.09448 (2016).
16. Fawzi, Alhussein, et al. "Classification regions of deep neural networks." arXiv preprint arXiv:1705.09552 (2017).