INTRODUCTION TO MACHINE LEARNING COMPSCI 4ML3

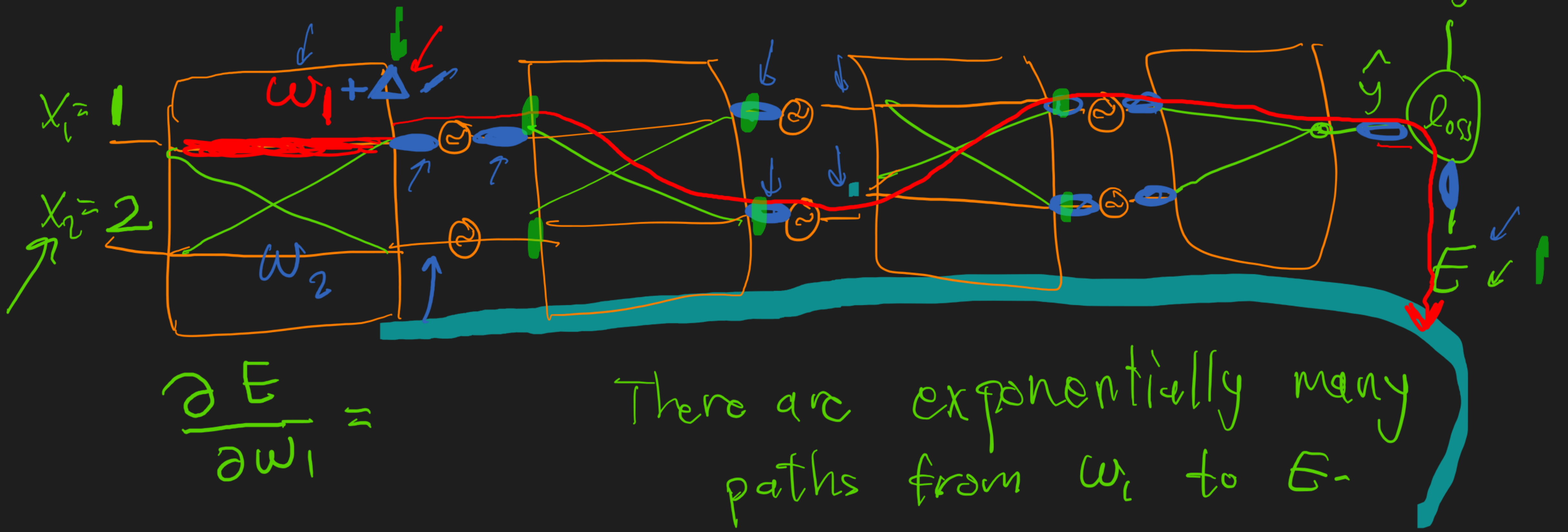
LECTURE 23

HASSAN ASHTIANI

COMPUTING THE GRADIENT EFFICIENTLY

- $\nabla_{\mathrm{w}}\big(\mathrm{L}(w,b)\big) = \sum_i \nabla_{\mathrm{w}}\Big(l\big(f_{w,b}(x^i), y^i\big)\Big)$
  - How to calculate $\nabla_{\mathrm{w}}\Big(l\big(f_{w,b}(x^i), y^i\big)\Big)$?
- This can be computationally expensive

NAÏVE APPROACH



$$\frac{\partial E}{\partial w_1} =$$

There are exponentially many paths from w_1 to E.

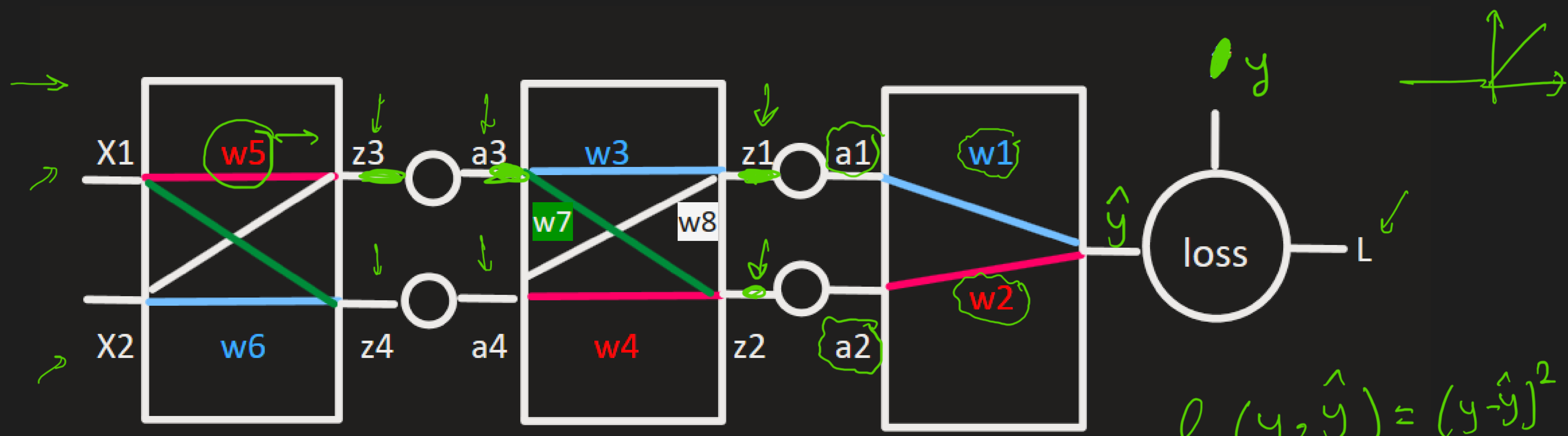
* the computations involved in computing $\frac{\partial E}{\partial w_1}$ and $\frac{\partial E}{w_2}$ are redundant

COMPUTING THE GRADIENT?

- Naïve approach:
  - Computationally expensive for deep models
  - Some of the computations are repetitive
- What to do?
  - A kind of "dynamic programming"
  - Back propagation

BACK-PROPAGATION

- Use chain rule (for vector-valued functions)
- Linear time in terms of the number of weights!
- Forward phase
  - Compute the input/outputs of activation functions
- Backward phase
  - Compute the gradients, layer-by-layer



$\ell(y, \hat{y}) = (y - \hat{y})^2$

ReLU activation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = 2(\hat{y} - y) \cdot a_1$$

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} = 2(\hat{y} - y) \cdot w_1 \cdot 1\{z_1 > 0\}$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_5} =$$

$$\left(\frac{\partial L}{z_1} \cdot \frac{\partial z_1}{\partial a_3} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_3} \right) \cdot \frac{\partial a_3}{\partial z_3} \cdot x_1$$

$=$

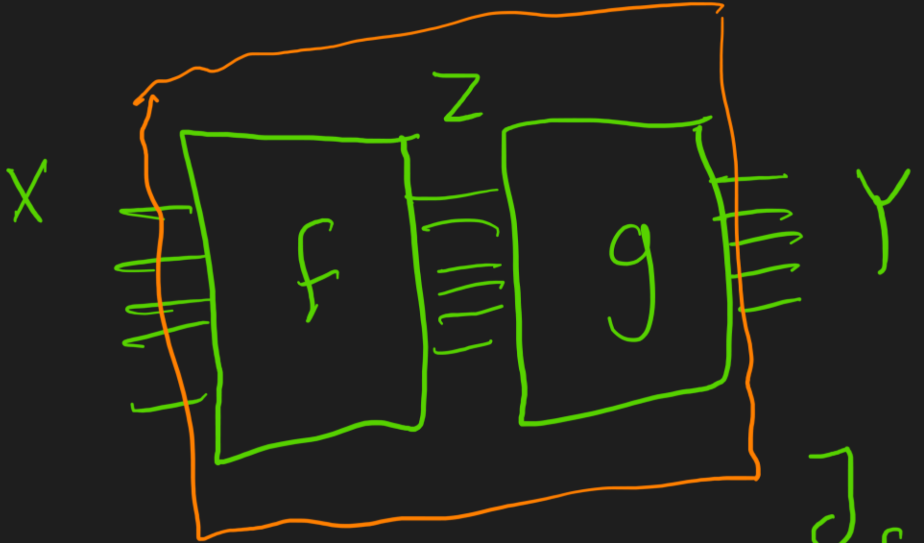
How to do these using matrix/vector calculus?

MULTI-VARIATE CHAIN RULE AND A MODULAR APPROACH

$A \cdot B \cdot C \cdot D \cdot E$

$f: \mathbb{R}^n \rightarrow \mathbb{R}^m$

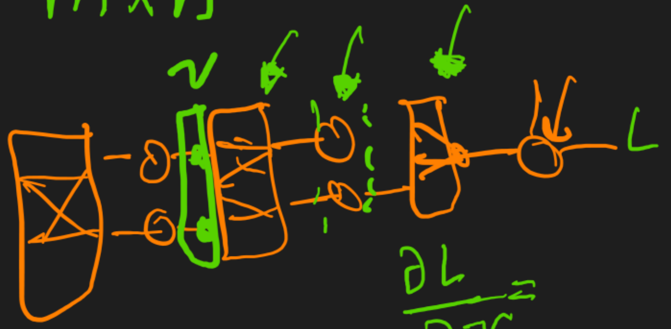
$g: \mathbb{R}^m \rightarrow \mathbb{R}^d$



$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{x_n} \\ \vdots & & \\ \frac{\partial f_m}{\partial x_1} & & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{m \times n} \checkmark$$

$$\nabla_x f \circ g(x) \overset{?}{=} [\nabla g]_{d \times m} \cdot [\nabla f]_{m \times n}$$

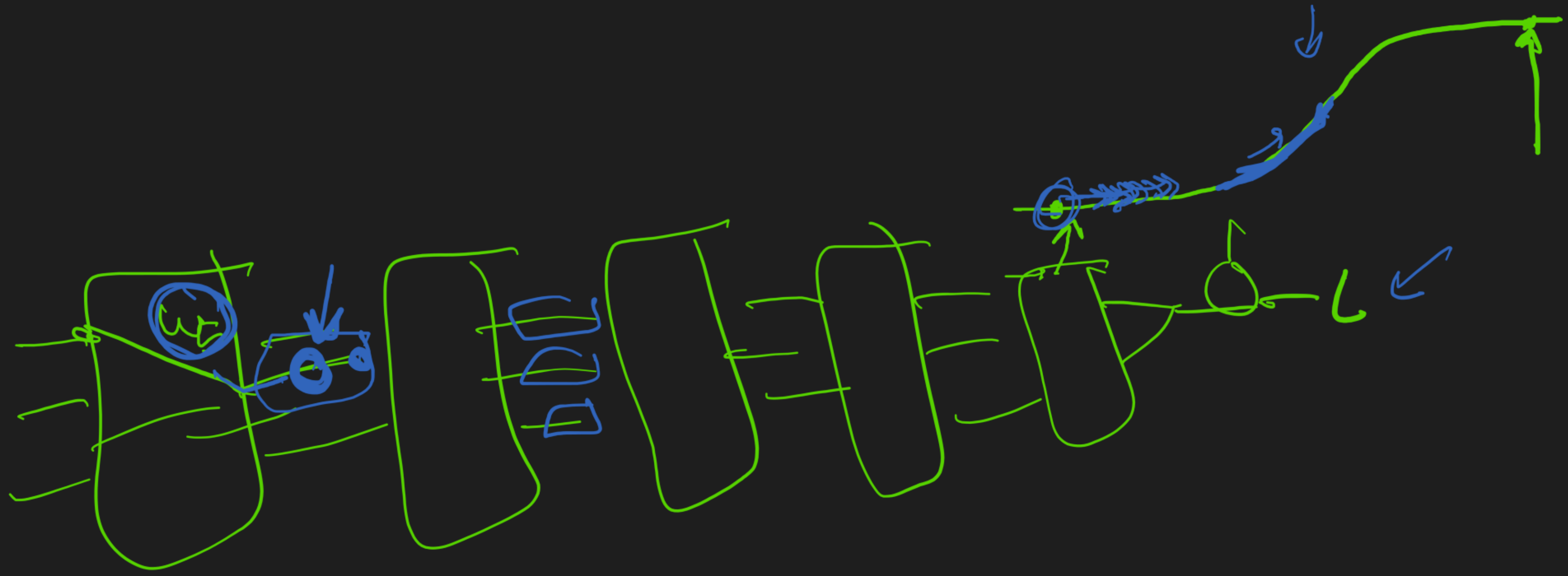
$$\text{if} \quad f = Ax \quad \text{then} \quad \nabla_f = A$$



$\frac{\partial L}{\partial r} =$

VANISHING GRADIENT

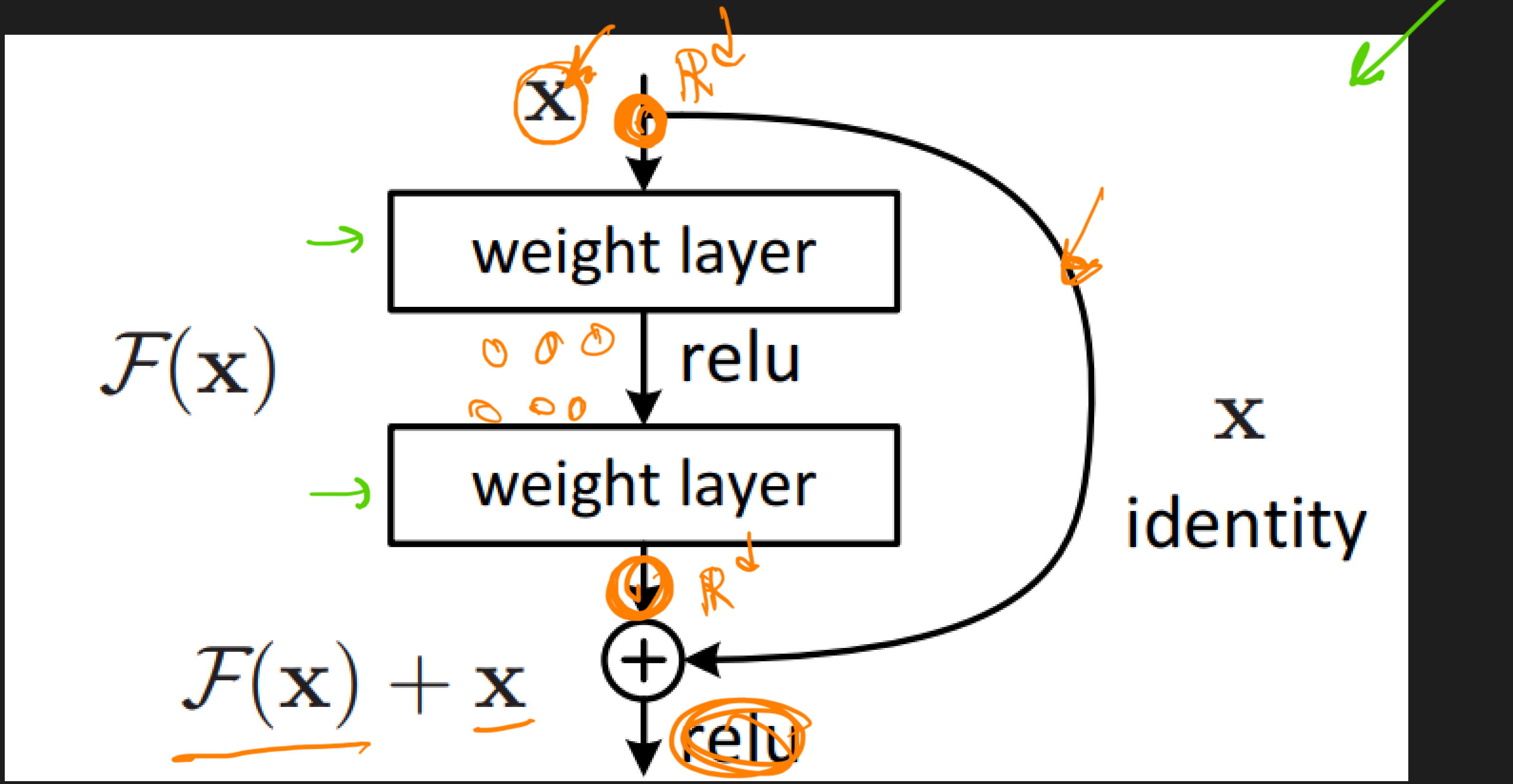
- For deeper networks, the partial derivatives
  - For the output layer vs the input layer?
- For sigmoid activation functions
  - $\sigma'(x) \in [0,1]$
  - For "saturated" neurons $\sigma'(x) \approx 0$
  - Gradient "does not reach" the first layers
- What can we do?

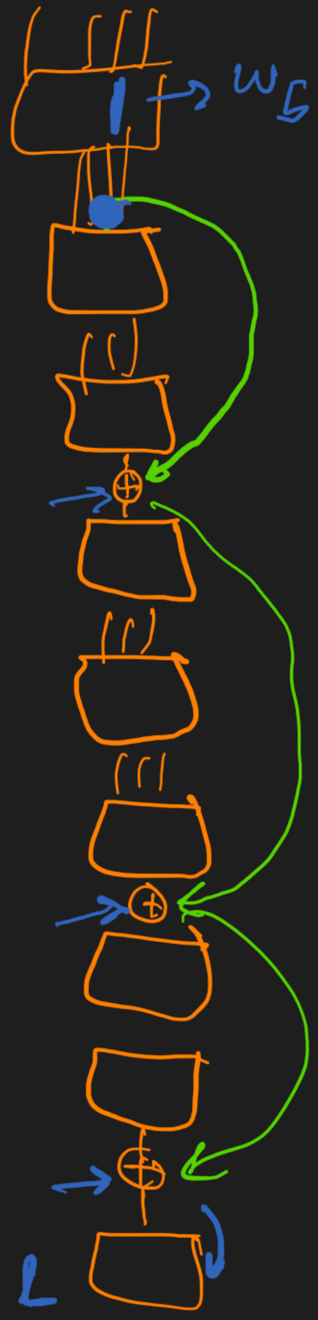

L

VANISHING GRADIENT

- Some activation functions are better
  - Leaky ReLU (also ReLU/max-out)
  - But gradient still can vanish after a couple layers
- Better initialization
- A possible workaround
  - Use "shortcuts" for the gradient to flow
- ResNets can have 100s of layers!

DEEP RESIDUAL NETWORKS





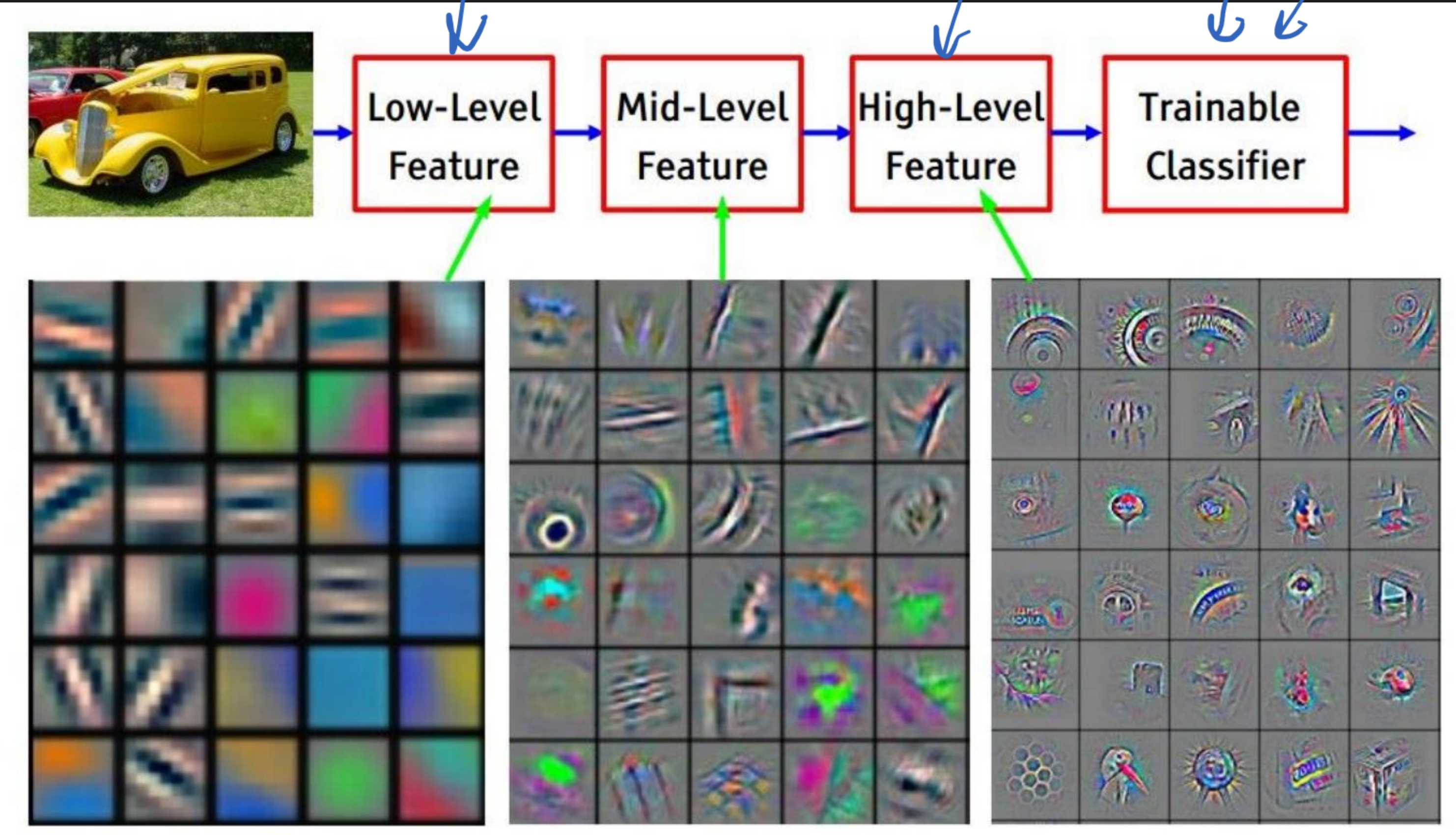
- THE INPUT CAN DIRECTLY GO TO DEEPER NEURONS, SO THE GRADIENT CAN FLOW

UNIVERSAL APPROXIMATION THEOREM

- FEED-FORWARD NETWORKS WITH SIGMOID ACTIVATION FUNCTIONS CAN APPROXIMATE ANY BOUNDED CONTINUOUS FUNCTION UP TO DESIRABLE ACCURACY
  - **ONLY A SINGLE HIDDEN LAYER IS NEEDED!**
  - GEORGE CYBENKO, 1989
  - ALSO HOLDS FOR OTHER USUAL ACTIVATION FUNCTIONS
- IS THERE A POINT IN HAVING MORE THAN ONE HIDDEN LAYER?

DEEP VS SHALLOW NETWORKS

- Low-level to high-level computations/detections

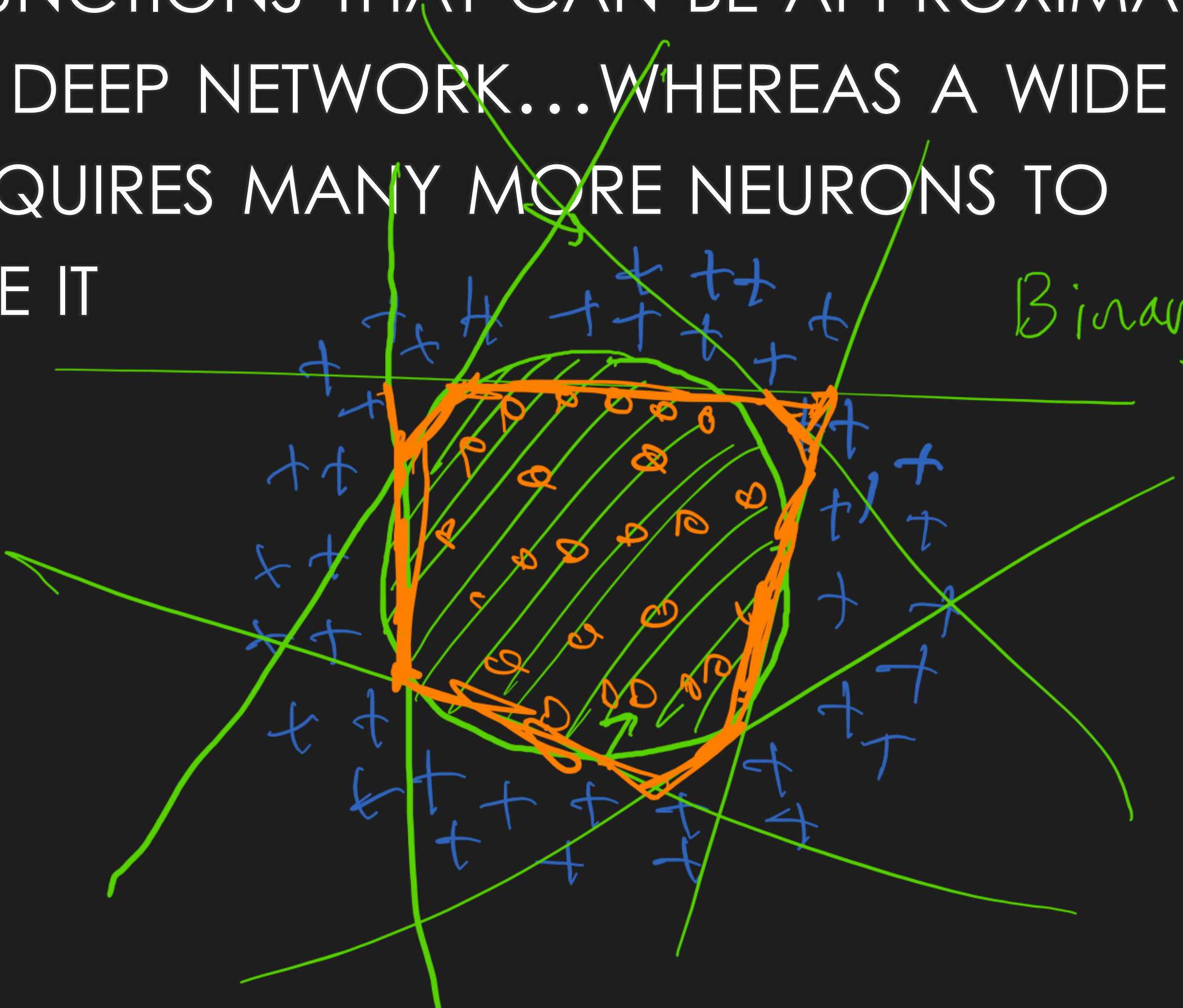


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

DEEP VS SHALLOW NETWORKS

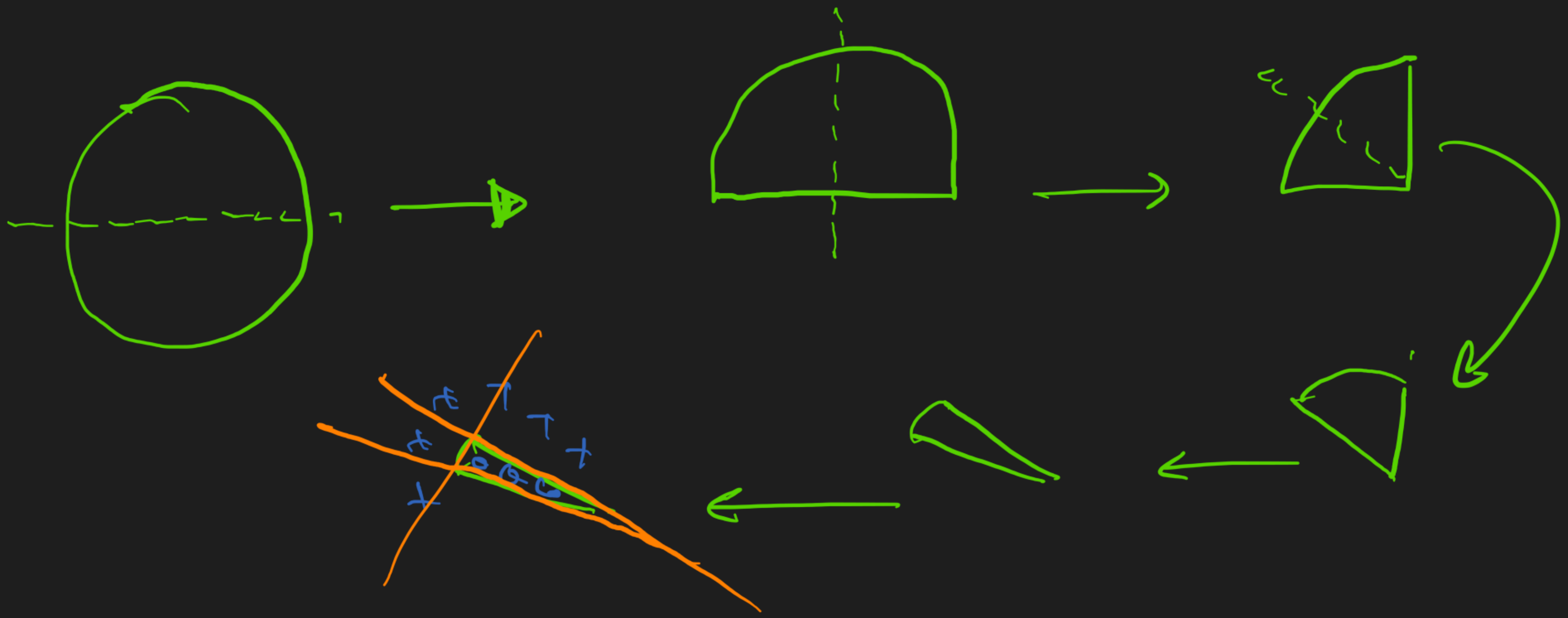
- There are functions that can be approximated with a small but deep network…whereas a wide shallow network requires many more neurons to approximate it

Binary classification



$$\text{error} \approx \frac{1}{\#\text{neurons}}$$

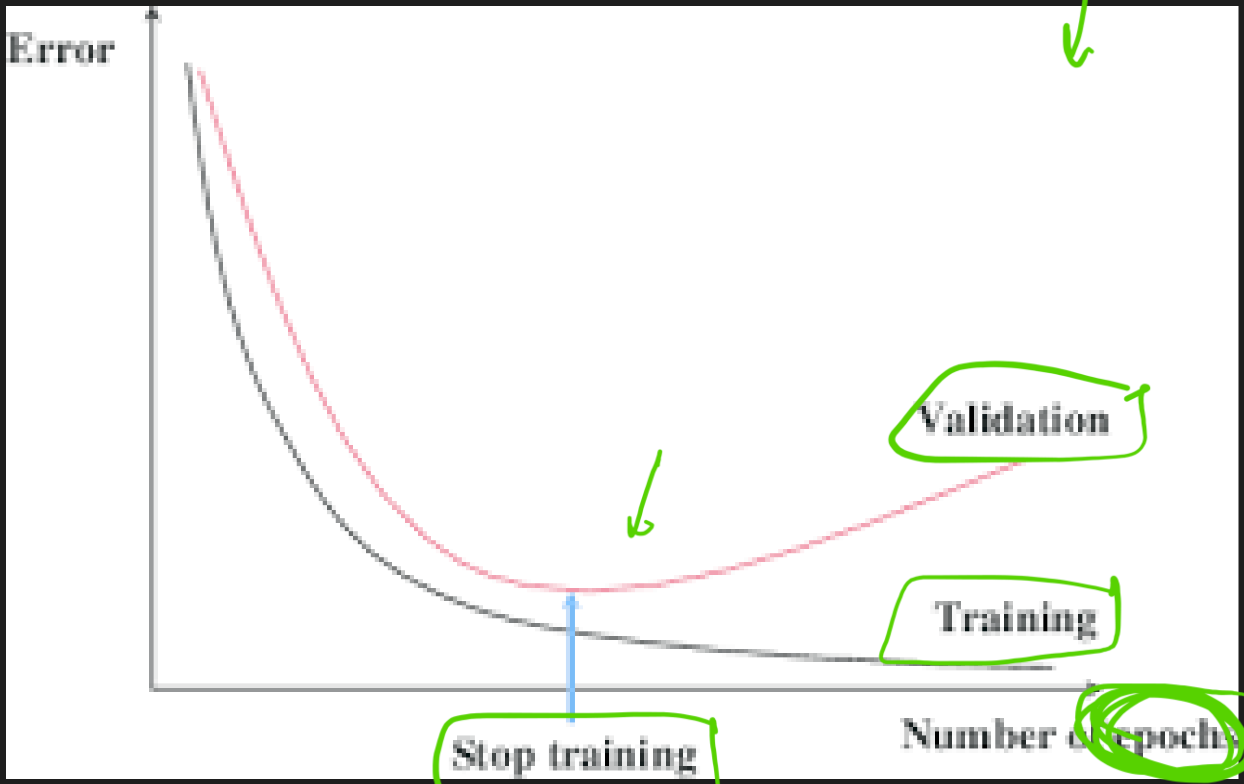
$$\text{error} \lesssim \frac{1}{(\#\text{neurons})^{\text{layers}}}$$



REGULARIZING NNS

- Good neural networks are often over-parametrized!
  - Prone to over-fitting

1. Adding regularization terms to the objective function
   - $E(w) + ||w||^2$
2. Early stopping
3. Adding noise
4. Structural regularization



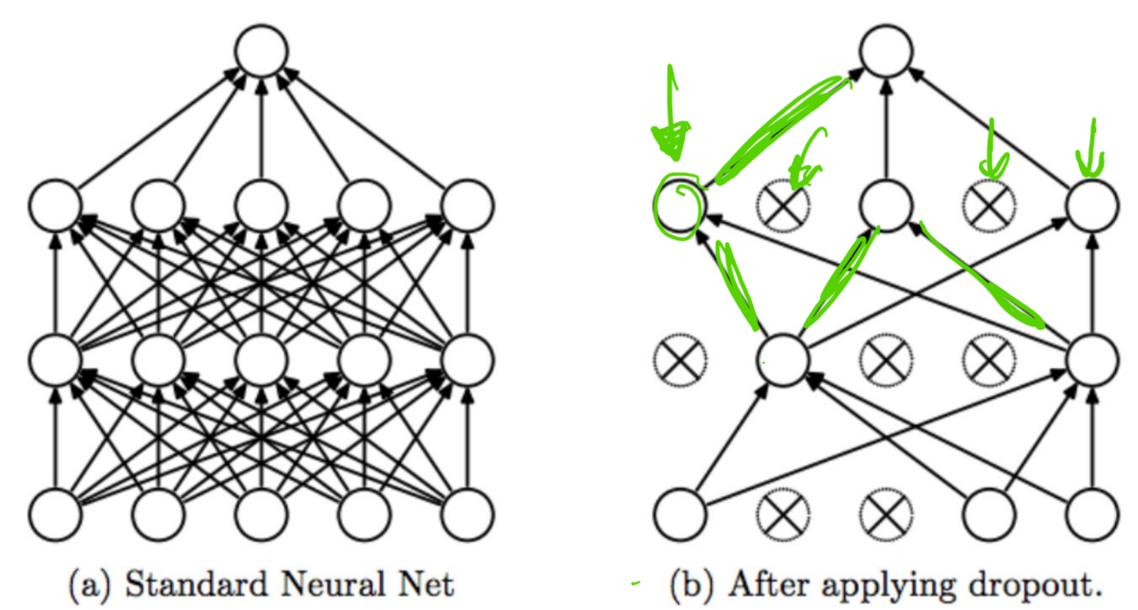
REGULARIZING NNS WITH DROPOUT

TRAINING

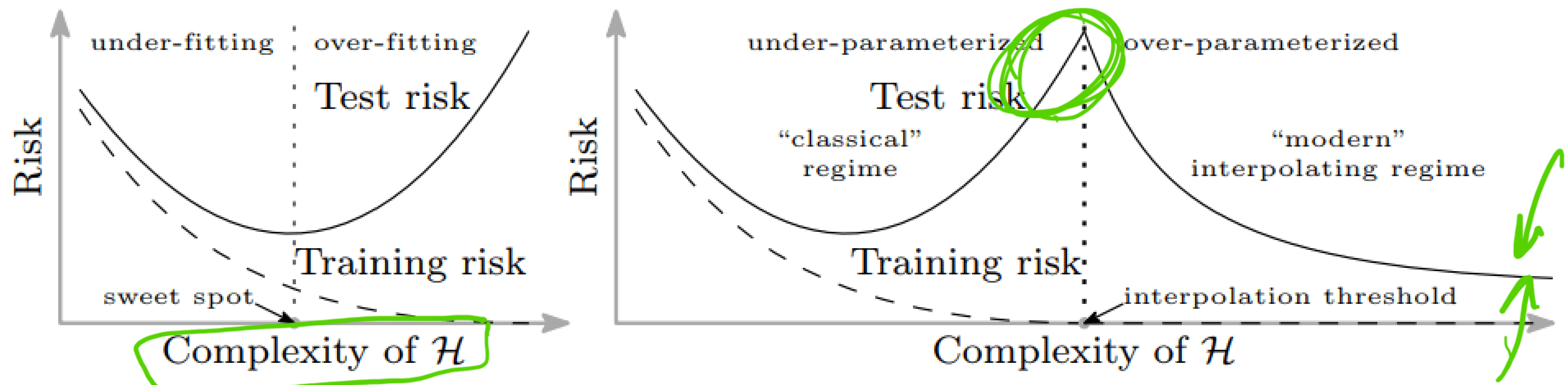
- FOR EACH ITERATION OF STOCHASTIC GRADIENT DESCENT AND FOR EACH TRAINING DATA POINT DO:
  - DROP EACH NODE WITH PROBABILITY p

TESTING

- DON'T DROP THE NODES, BUT FOR ALL NODES, BUT MULTIPLY THE VALUE OF ACTIVATIONS BY $(1-p)$



DOUBLE DESCENT?



(a) U-shaped "bias-variance" risk curve

(b) "double descent" risk curve