Vanishing (Exploding) Gradient Problems

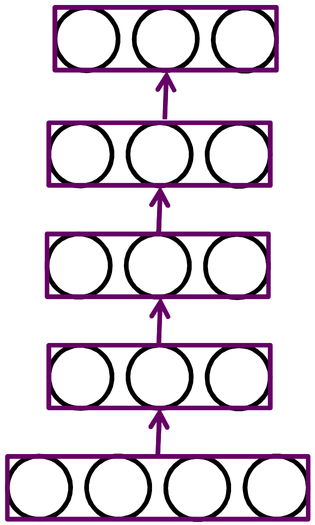
and the Solutions

Some slides were adated/taken from various sources, including Andrew Ng's Coursera Lectures, CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University CS Waterloo Canada lectures, Aykut Erdem, et.al. tutorial on Deep Learning in Computer Vision, Ismini Lourentzou's lecture slide on "Introduction to Deep Learning", Ramprasaath's lecture slides, and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and NOT to distribute it.

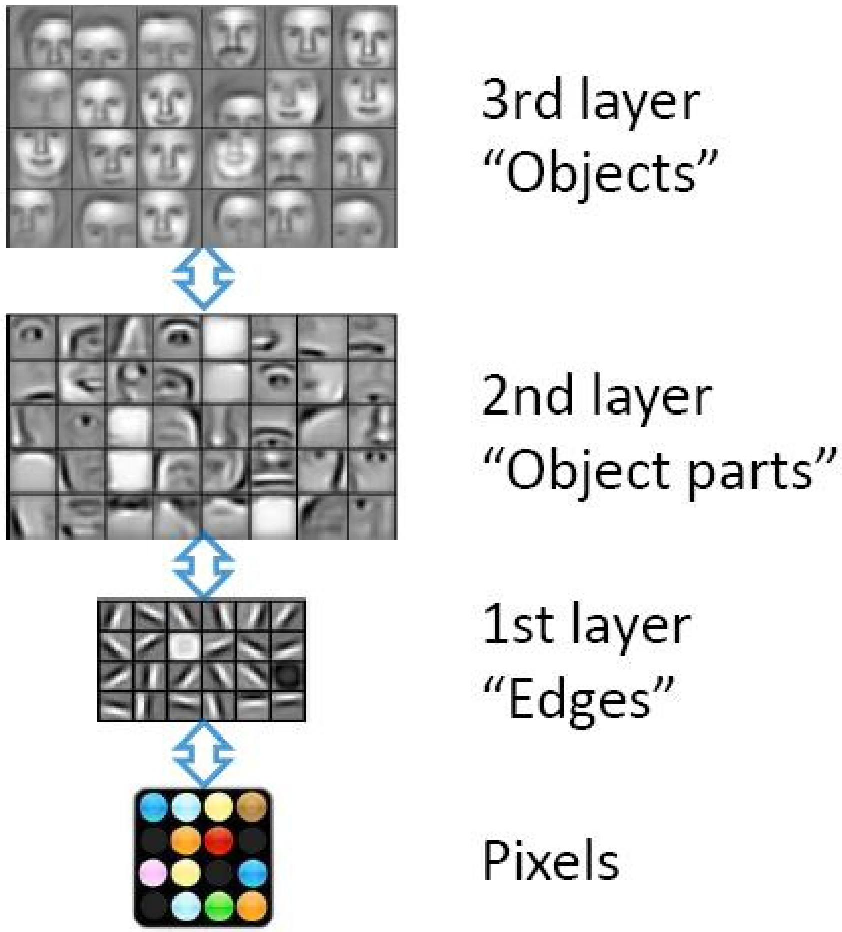
Why Stop At One Hidden Layer?

E.g., vision hierarchy for recognizing handprinted text

Word	output layer
Character	hidden layer 3
Stroke	hidden layer 2
Edge	hidden layer 1
Pixel	input layer



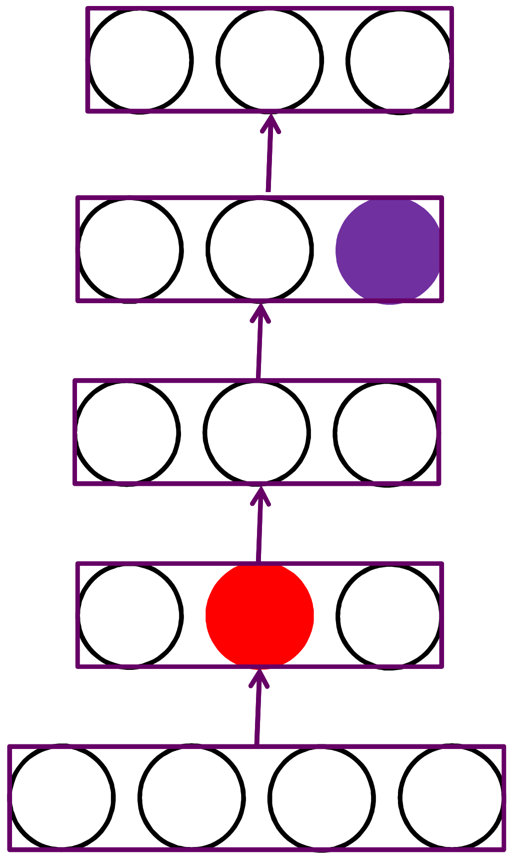
From Ng’s group



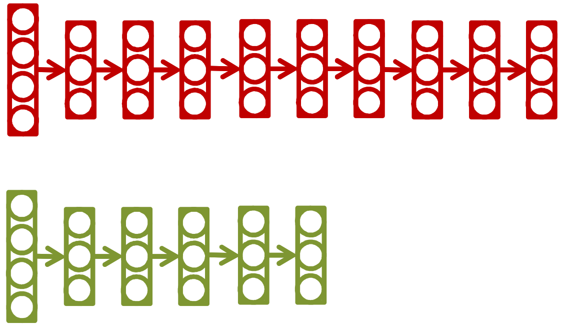
Why Deeply Layered Networks Fail

Credit assignment problem

- How is a **neuron in layer 2** supposed to know what it should output until all the neurons above it do something sensible?
- How is a **neuron in layer 4** supposed to know what it should output until all the neurons below it do something sensible?



Deeper Vs. Shallower Nets



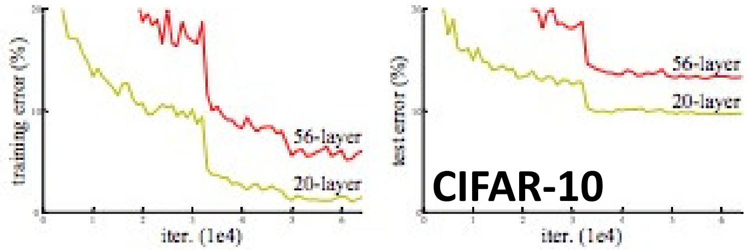
Deeper net can represent any mapping that shallower net can

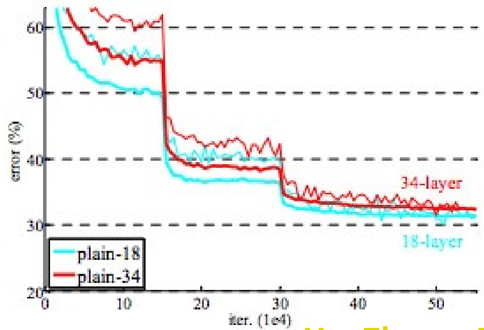
- **Use identity mappings for the additional layers**

Deeper net in principle is more likely to overfit

But in practice it often underfits *on the training set*

- **Degradation due to harder credit-assignment problem**
- **Deeper isn't always better!**





ImageNet
thin=train,
thick=valid.

He, Zhang, Ren, and Sun (2015)

Why Deeply Layered Networks Fail

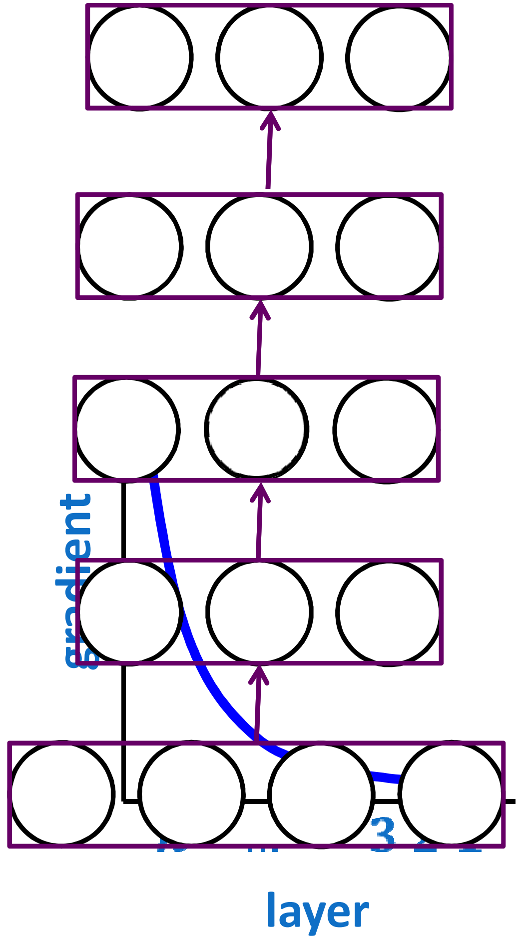
Vanishing gradient problem

- With logistic or tanh units

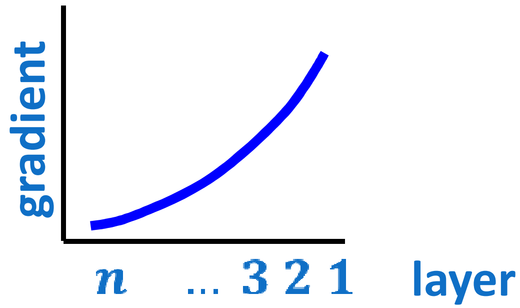
$$y_j = \frac{1}{1+\exp(-z_j)} \qquad \frac{\partial y_j}{\partial z_j} = y_j(1-y_j)$$

$$y_j = \tanh(z_j) \qquad \frac{\partial y_j}{\partial z_j} = (1+y_j)(1-y_j)$$

- Error gradients get squashed as they are passed back through a deep network



Why Deeply Layered Networks Fail

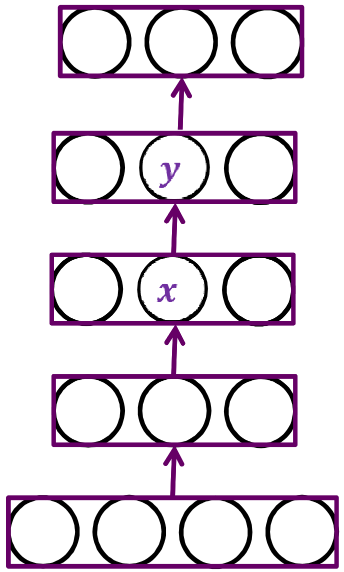


Exploding gradient problem

- with linear or ReLU units

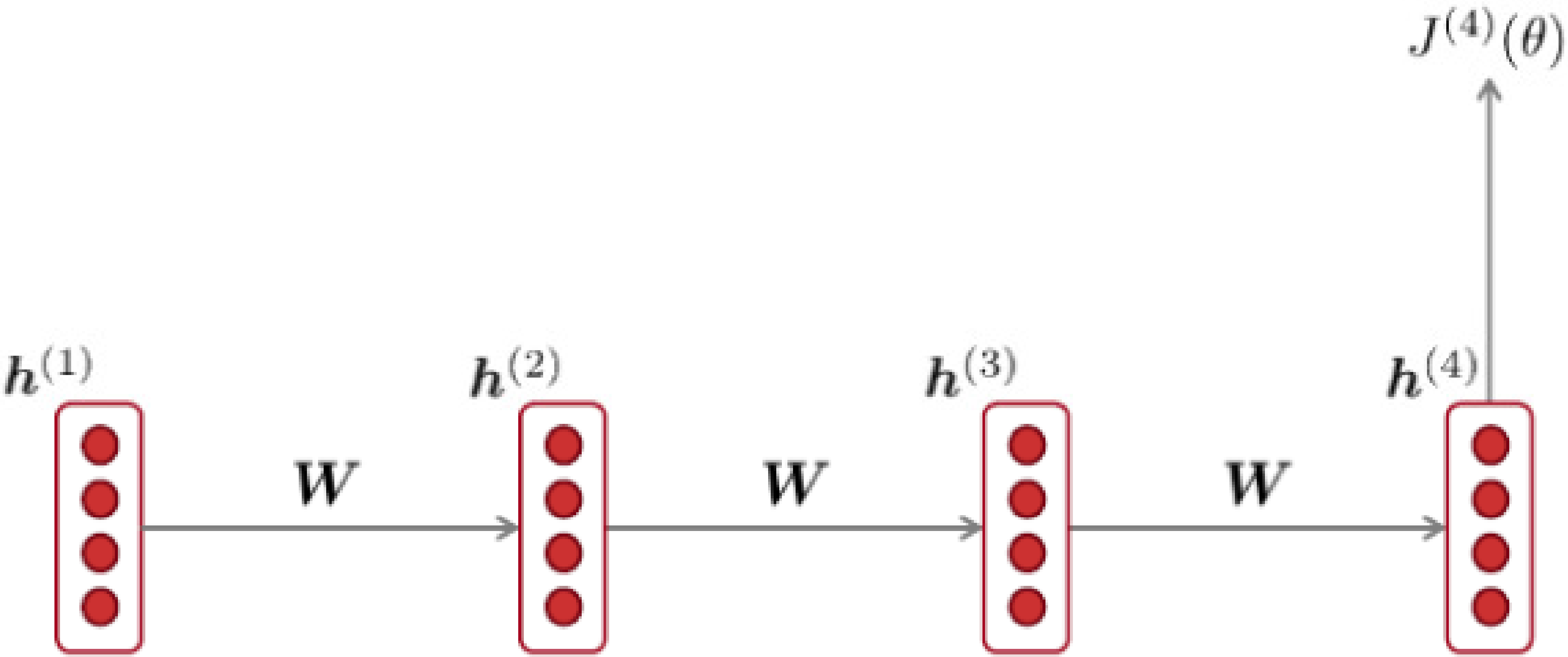
$$y = \max(\mathbf{0}, z) \qquad \frac{\partial y}{\partial z} = \begin{cases} \mathbf{0} & z \leq \mathbf{0} \\ \mathbf{1} & \textbf{otherwise} \end{cases}$$

$$\frac{\partial y}{\partial x} = \begin{cases} \mathbf{0} & y = \mathbf{0} \\ w_{yx} & \textbf{otherwise} \end{cases}$$

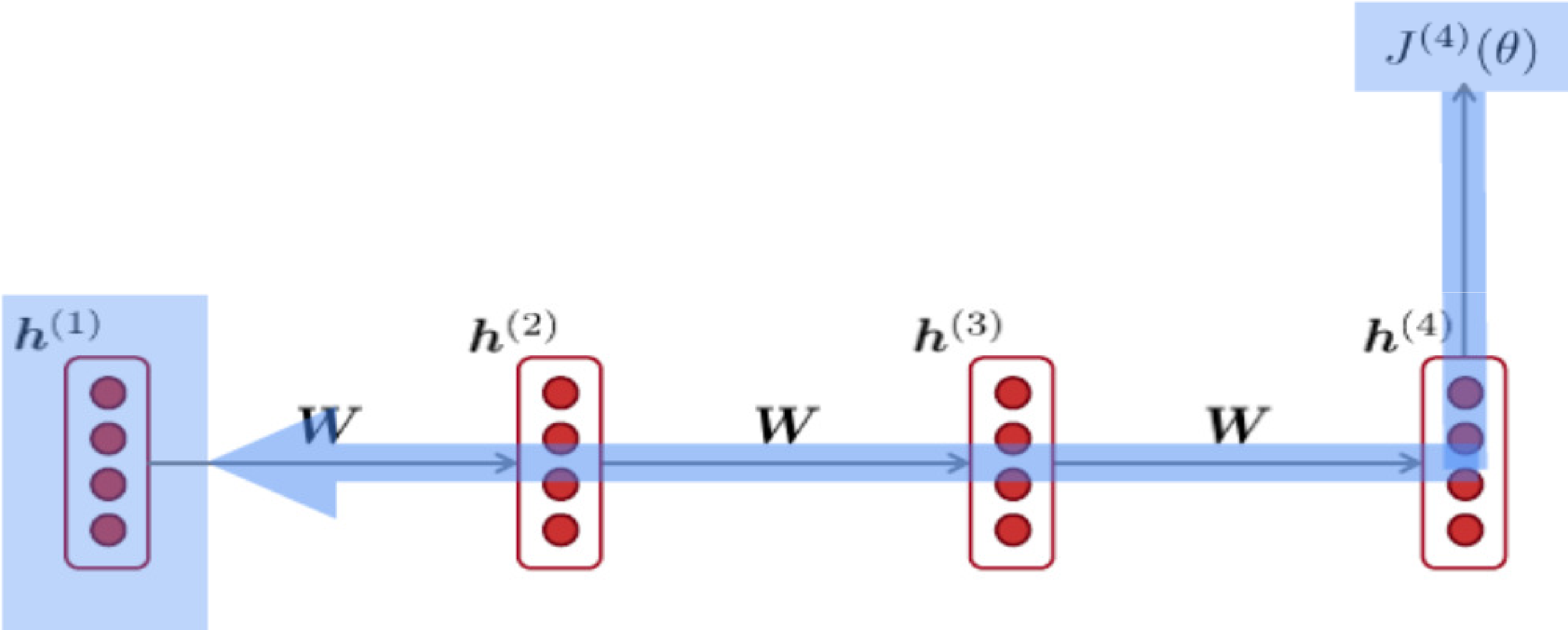


- can be a problem when $\left|\sum_y \frac{\partial y}{\partial x}\right| > 1$

Vanishing gradient intuition

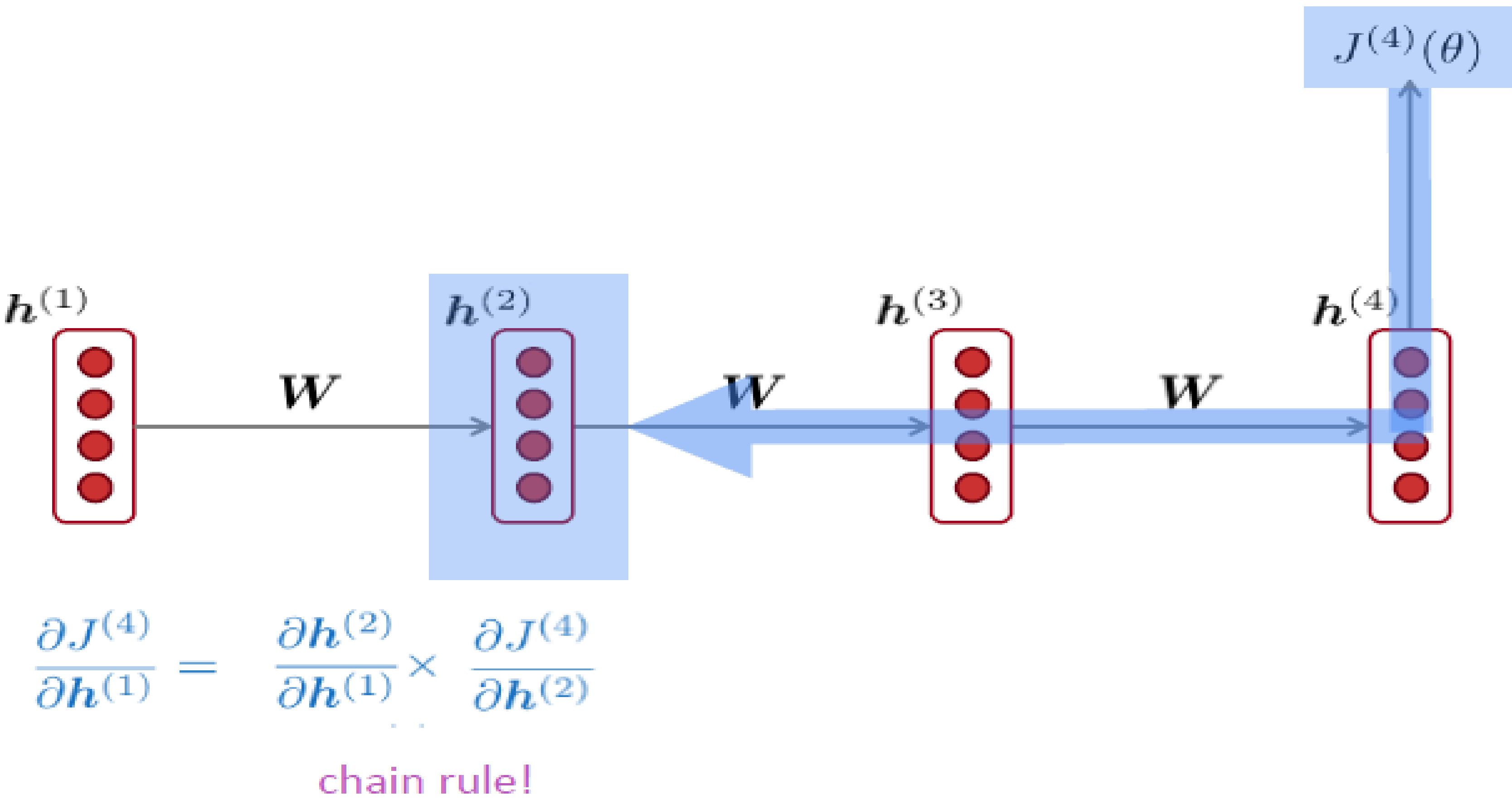


Vanishing gradient intuition

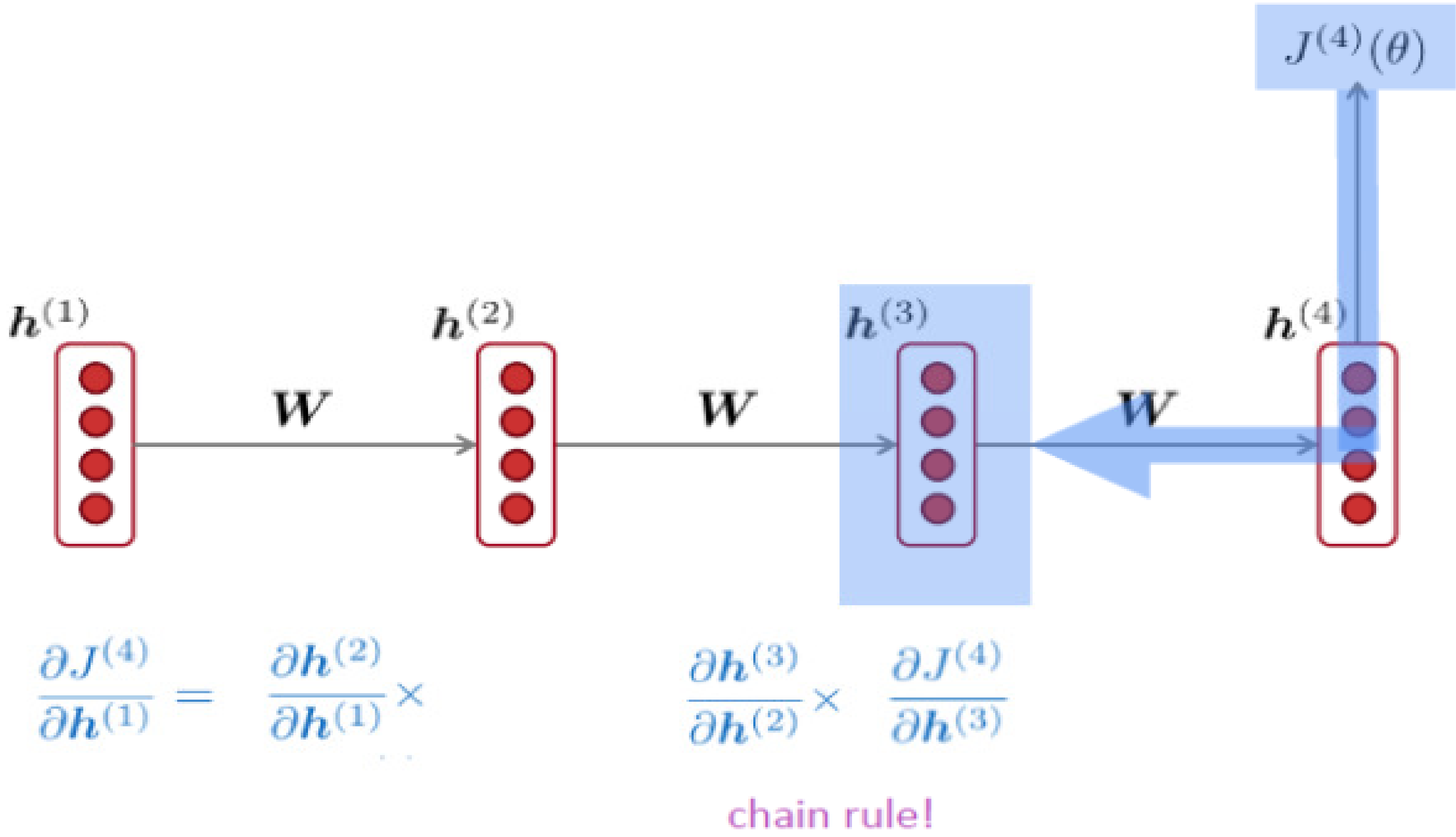


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

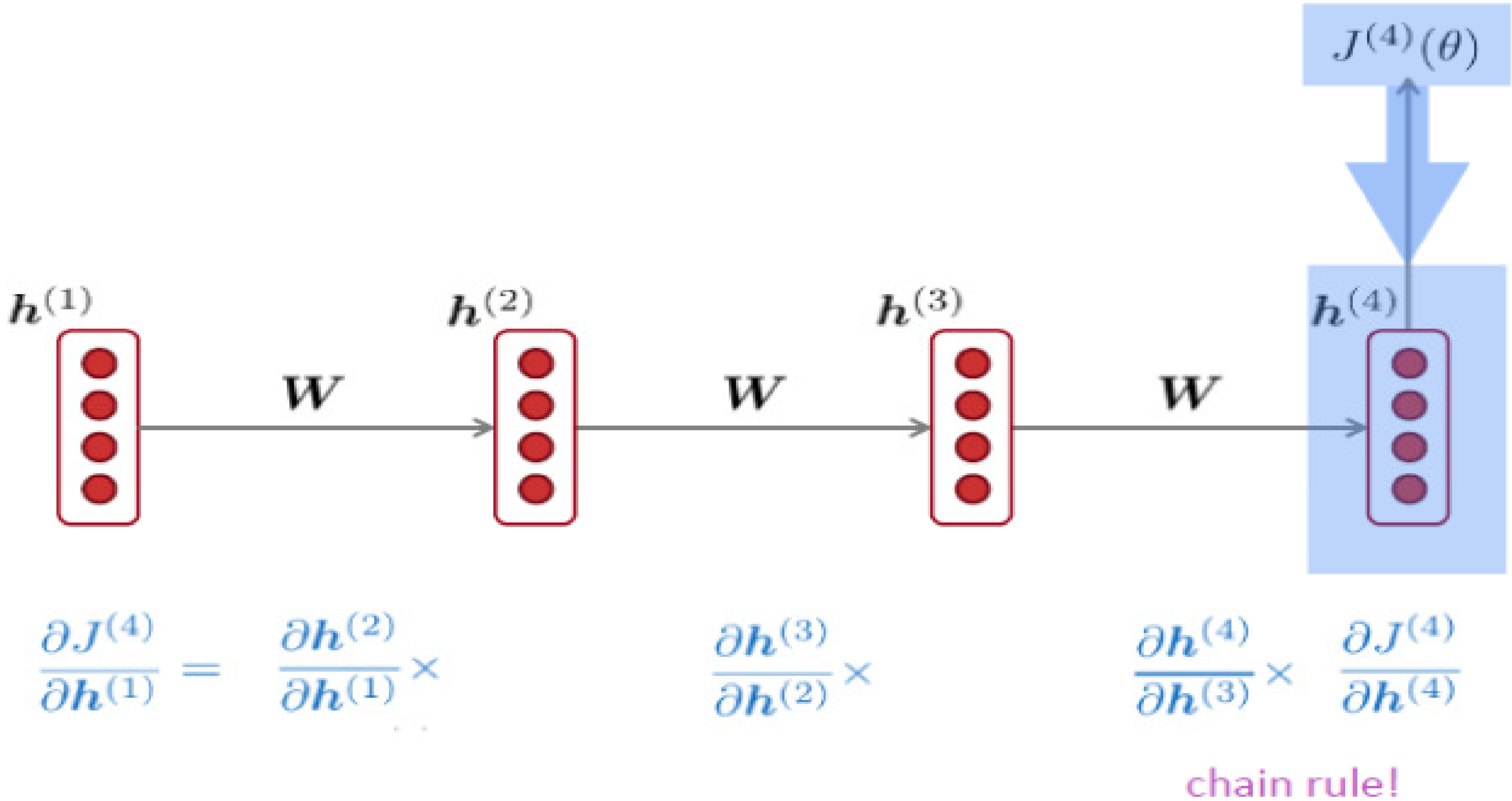
Vanishing gradient intuition



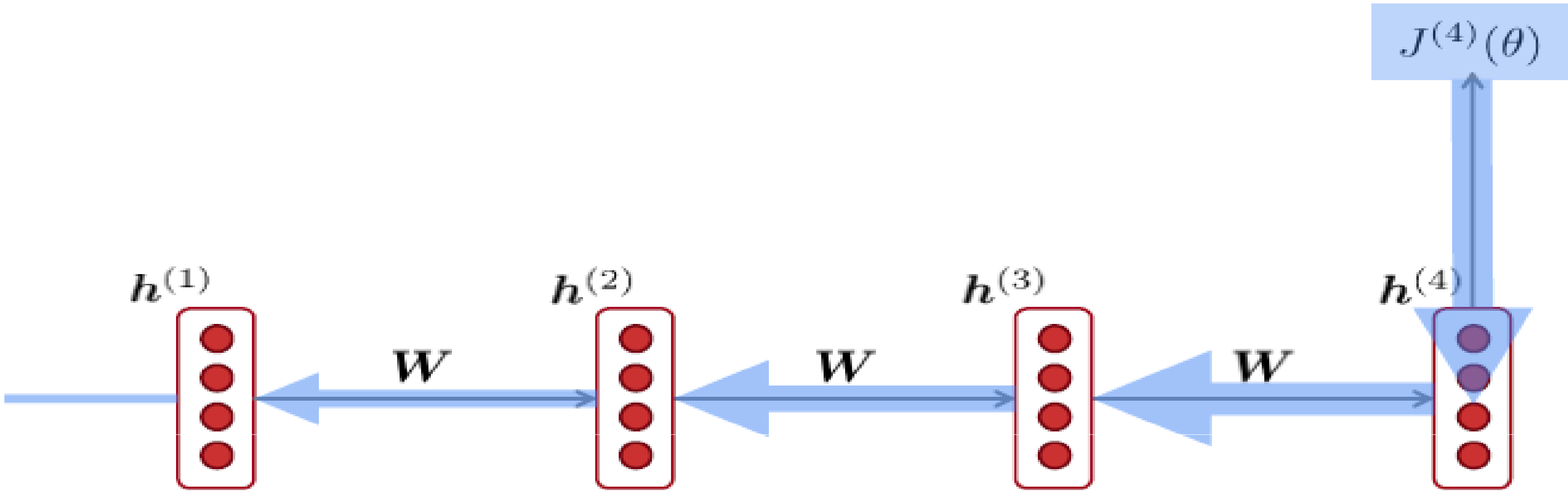
Vanishing gradient intuition



Vanishing gradient intuition



Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if these are small?

<u>Vanishing gradient problem:</u>
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Vanishing gradient proof sketch

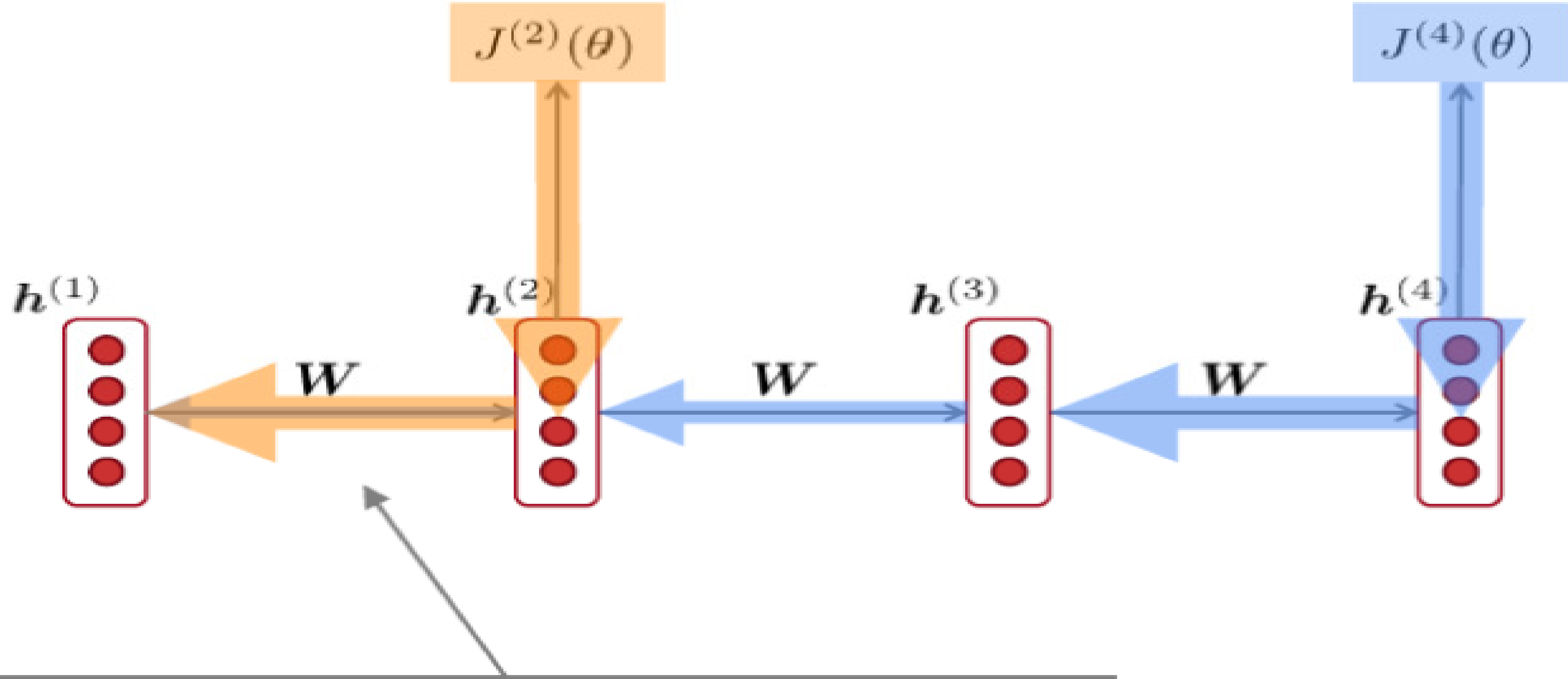
- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$
- Therefore: $$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h$$ (chain rule)
- Consider the gradient of the loss $J^{(i)}(\theta)$ on step *i*, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step *j*.

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$ (chain rule)

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{(i-j)}} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

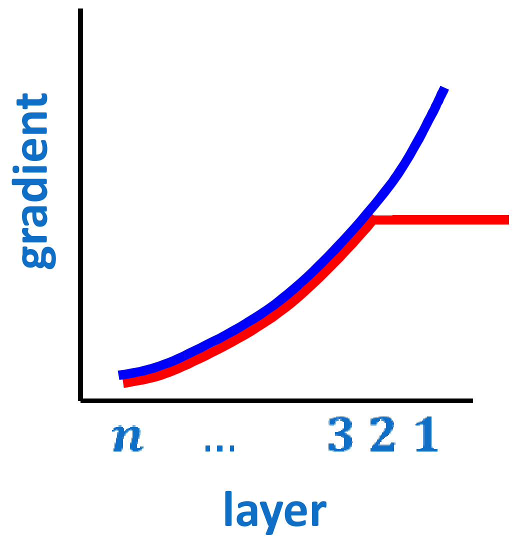
Why vanishing gradient is a problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Hack Solutions



Using ReLUs

- can avoid squashing of gradient

Use gradient clipping

- for exploding gradients

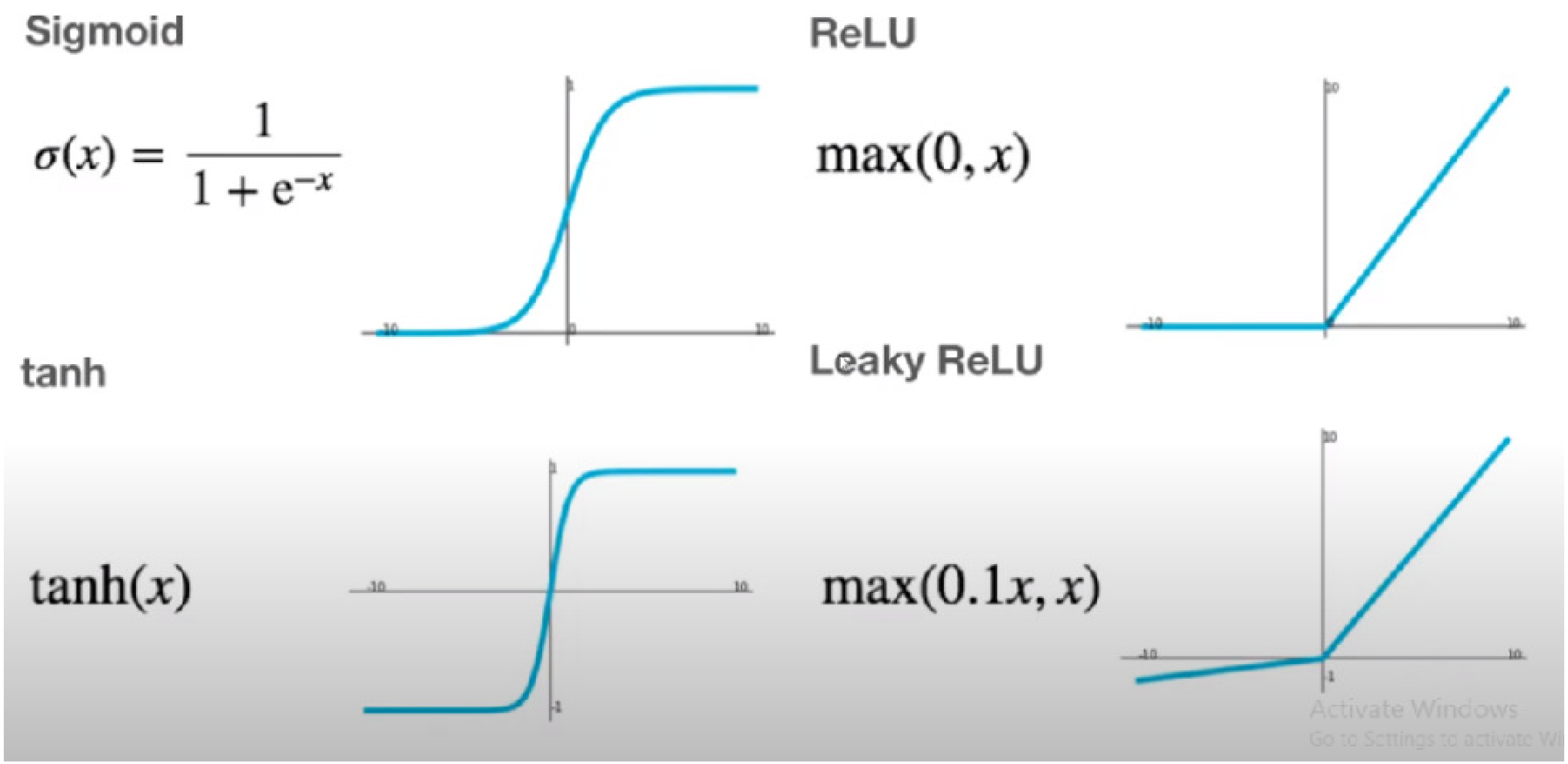
$$\Delta w_{xy} \sim \max\left(-\Delta_0, \min\left(\Delta_0, \frac{\partial E}{\partial w_{xy}}\right)\right)$$

Use gradient sign

- for exploding & vanishing gradients

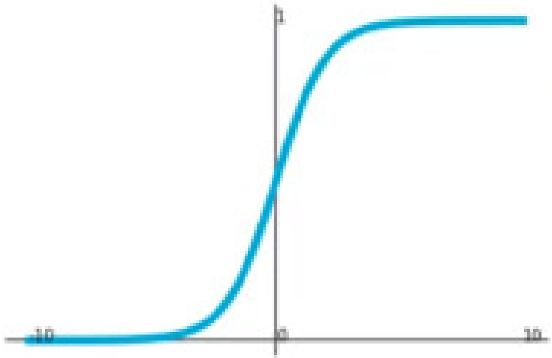
$$\Delta w_{xy} \sim \operatorname{sign}\left(\frac{\partial E}{\partial w_{xy}}\right)$$

Activation Functions



Activation Functions: Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

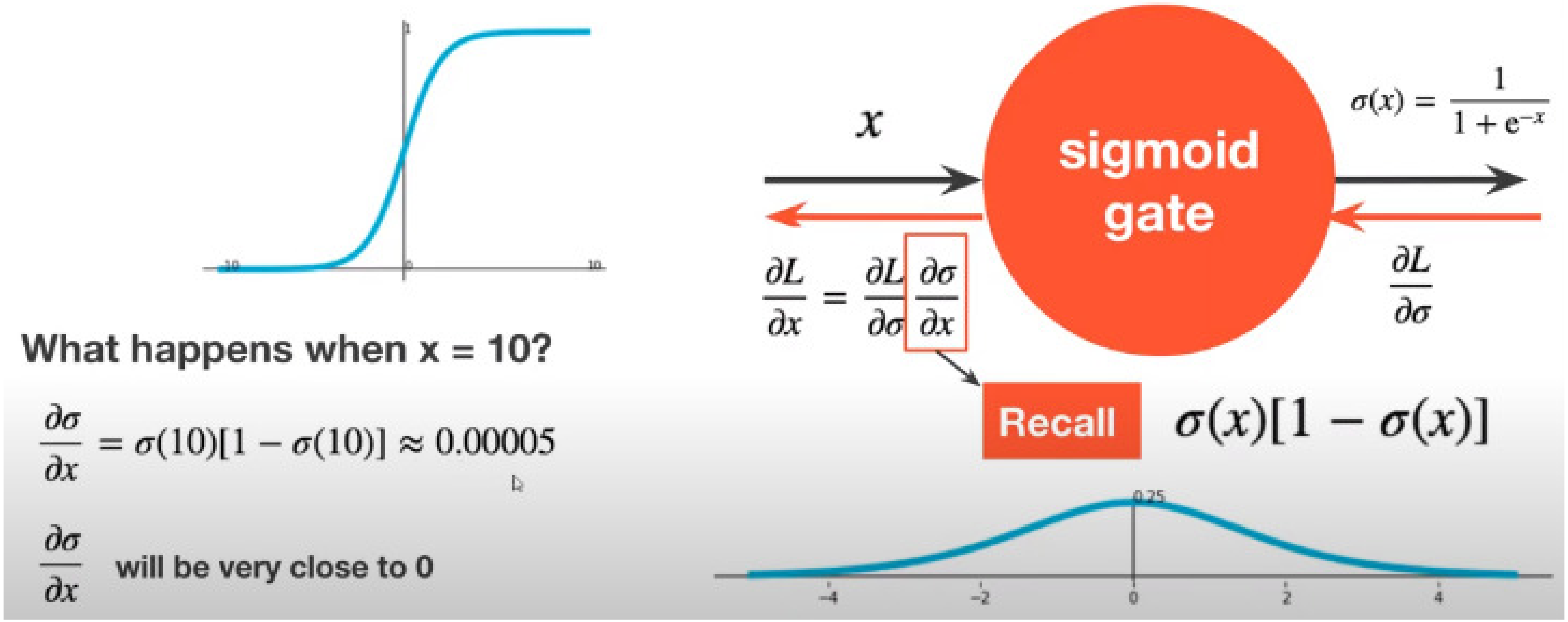


Squashes numbers to range [0, 1]

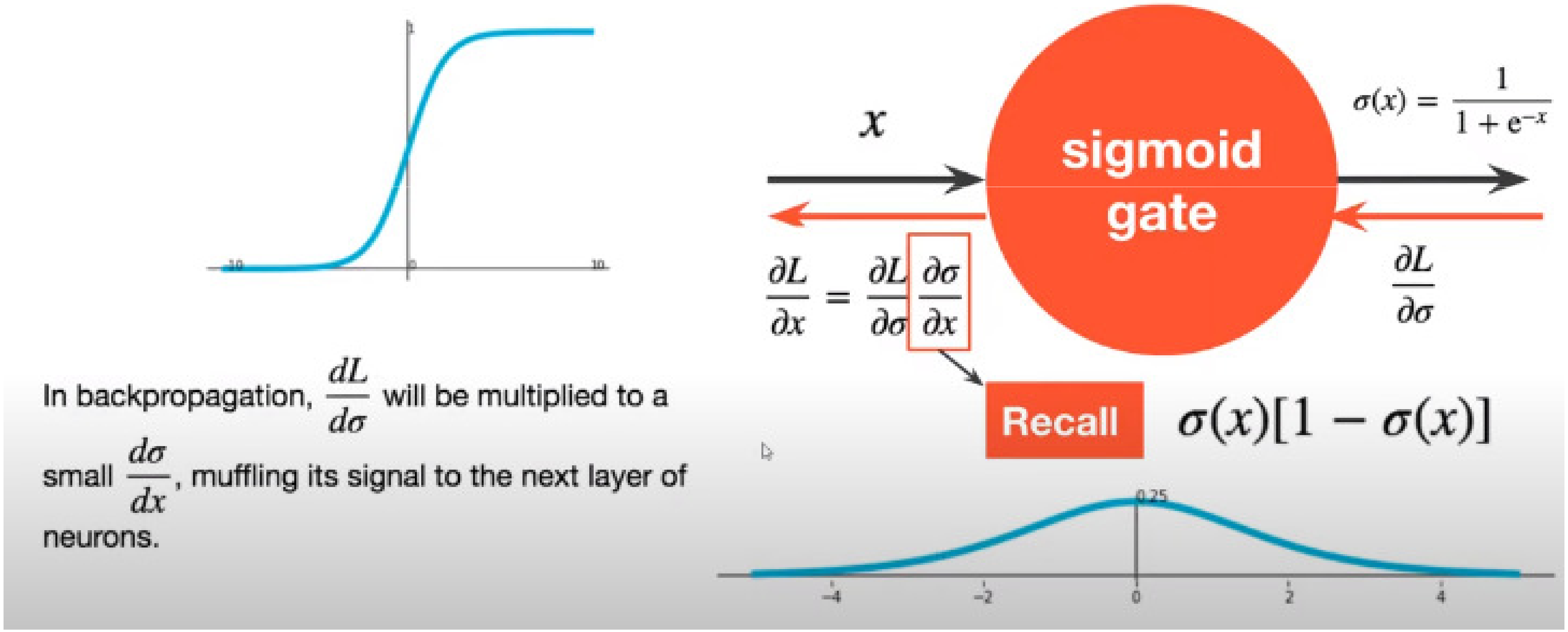
✖ Saturated neurons (neurons that output very close to 0 or very close to 1) "kill" the gradients during backpropagation

What does that mean?

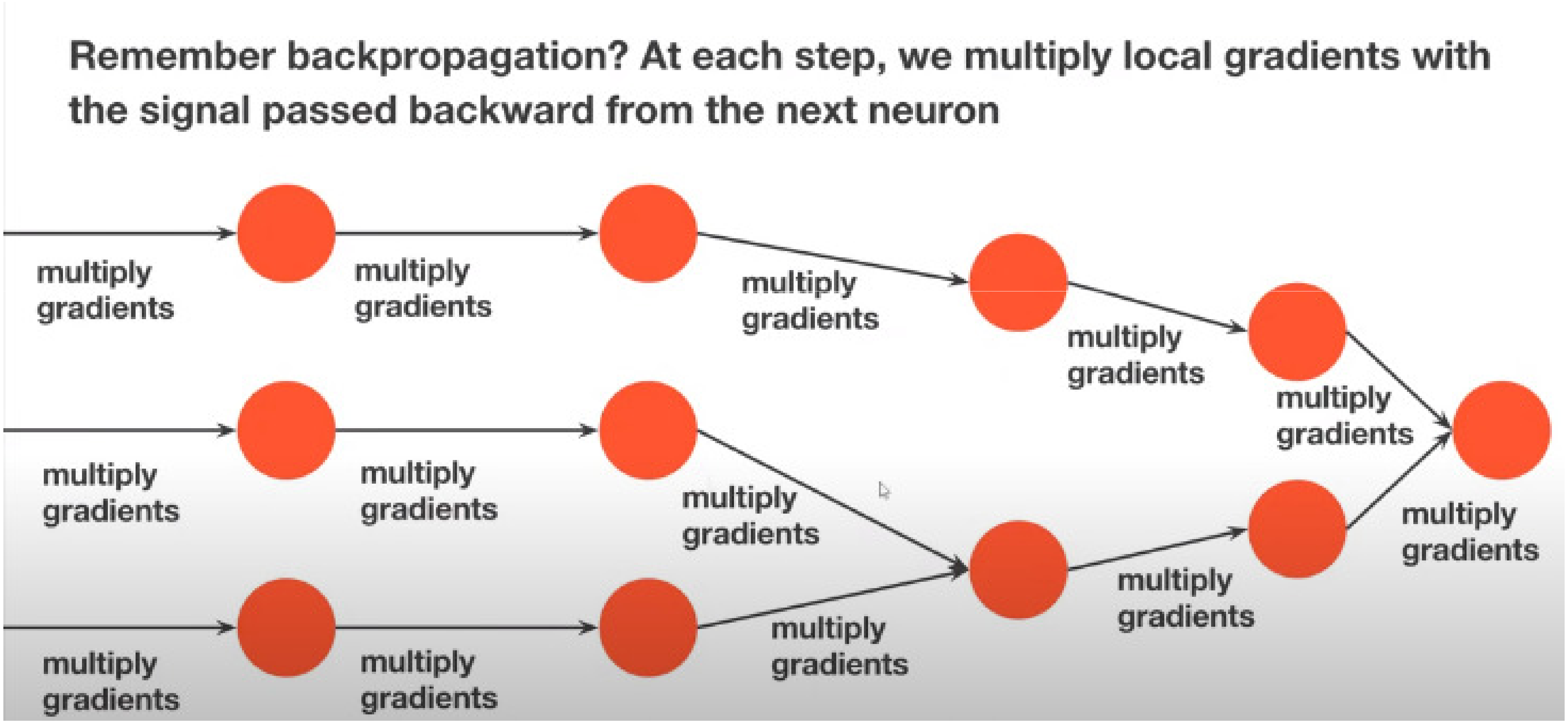
Activation Functions: Sigmoid



Activation Functions: Sigmoid

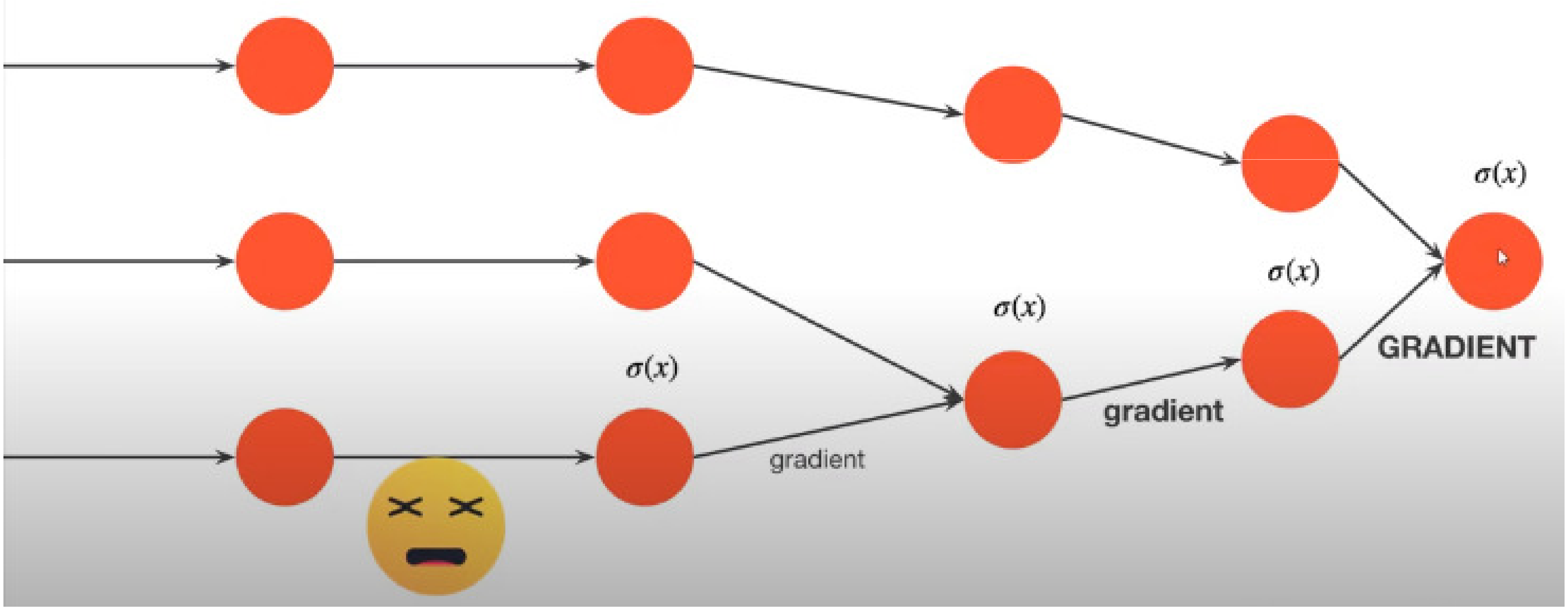


Activation Functions: on Backpropagation



Activation Functions: on Backpropagation

The gradients are going to slowly vanish until they simply die off. Once they do, the gradient flow stops and the neural network stops learning.



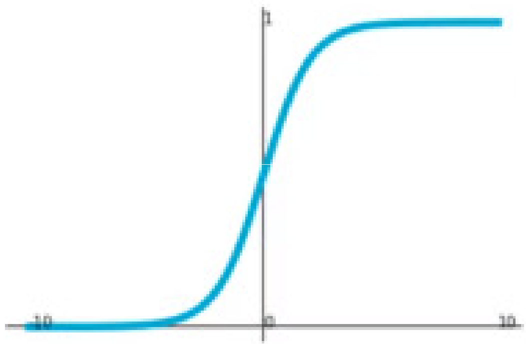
Why NN is taking longer time to train?

If the gradient at each “wiggle” step is too small, the training could get stuck. If the gradients are too small, then the NN stops training effectively.

- Many small steps means greater compute time needed to converge
- This small gradient problem is called the “vanishing gradient” problem.

Problems with Sigmoid Activation Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



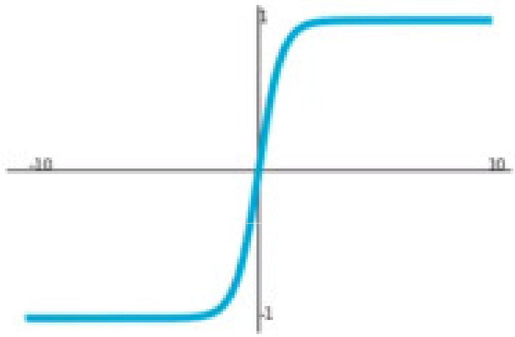
Squashes numbers to range [0, 1]

- ✖ Saturated neurons (neurons that output very close to 0 or very close to 1) "kill" the gradients during backpropagation
- ✖ Sigmoid outputs are not zero-centered
- ✖ e^x is computationally expensive

tanh Activation Function

tanh

$$\tanh(x)$$



Squashes numbers to range [-1, 1]

✔ Zero-centered (better than sigmoid!)

✖ Still kills gradients

Note that tanh is just a scaled sigmoid function

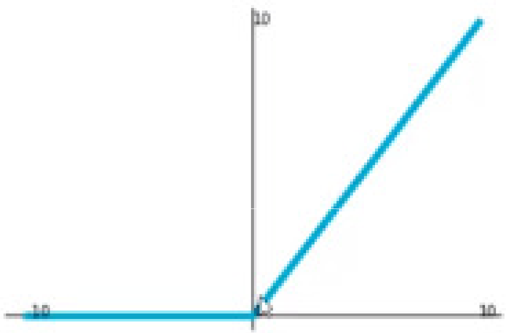
$$\frac{d}{dx}\tanh(x) = \operatorname{sech}^2 x$$

$$\tanh(x) = 2\sigma(2x) - 1$$

ReLU Activation Function

ReLU (Rectified Linear Unit)

$$\max(0, x)$$



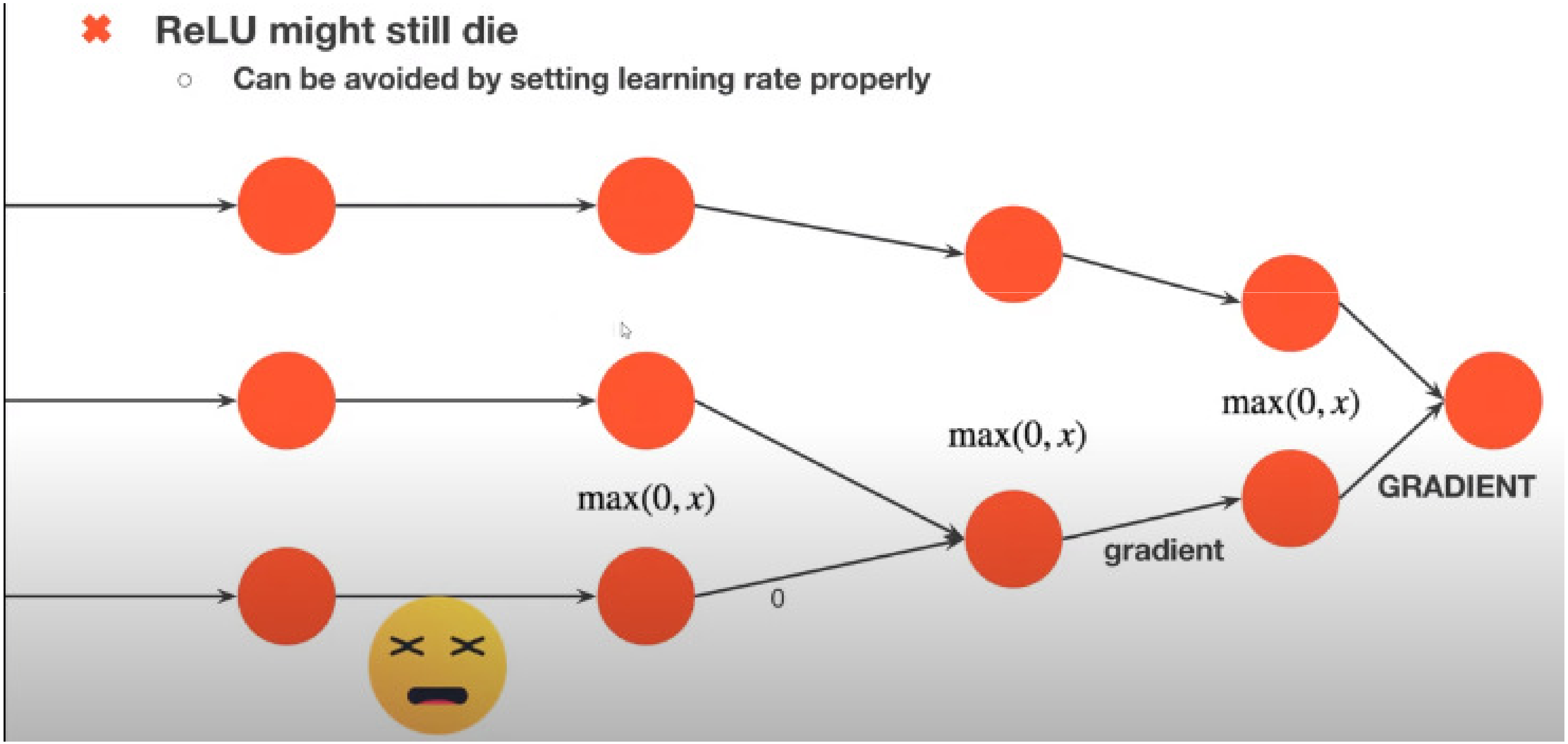
- ✔ Does not saturate in positive region
- ✔ Computationally efficient
- ✔ Converges faster than sigmoid and tanh in practice
- ✖ Might still die
  - Can be avoided by setting learning rate properly

$$\frac{d}{dx}\max(0, x) = 0 \text{ if } x < 0$$

$$\frac{d}{dx}\max(0, x) = 1 \text{ if } x > 0$$

Better than sigmoid and tanh!

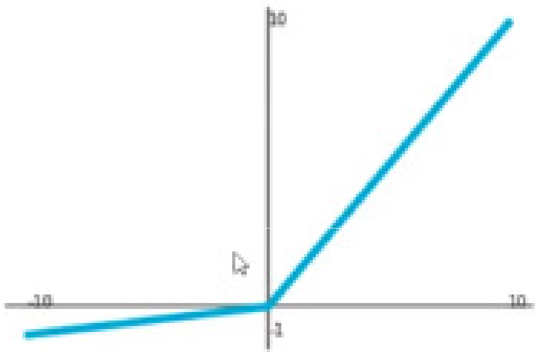
ReLU Activation Function on Backpropagation



Leaky ReLU Activation Function

Leaky ReLU

$$\max(0.1x, x)$$



$$\frac{d}{dx}\max(\alpha x, x) = \alpha \text{ if } x < 0$$

$$\frac{d}{dx}\max(\alpha x, x) = 1 \text{ if } x > 0$$

0.1 is an arbitrary value. The α in max(αx, x) can be set to any small value

- ✔ Solves ReLU's dying neuron problem

  Avoids 0 gradients by introducing a small slope

- ✔ Does not saturate
- ✔ Computationally efficient
- ✔ Converges faster than sigmoid and tanh in practice

Summary

- Use ReLU and be mindful of your learning rates
- Try Leaky ReLU
- Try tanh if choosing between sigmoid or tanh
- Don't use sigmoid!