



**ANCHORMEN**  
data activators

# DEEP LEARNING

Raoul Grouls

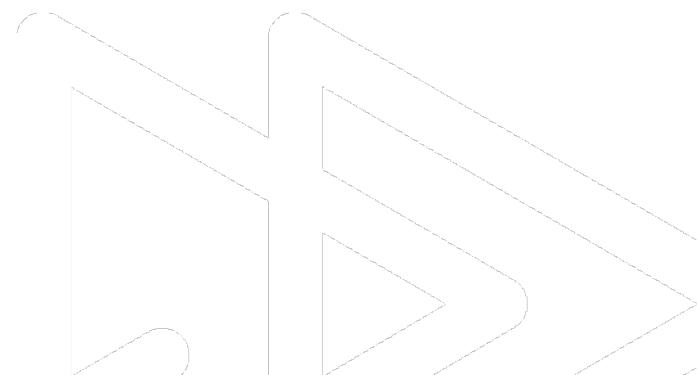
---

18-MAY-21



# INTRODUCTION DEEP LEARNING

- *Deep* refers to the many-layered computations
- At first **impossible** to feasibly train
- Around 2012, the ML community realized that with current resources this is no longer true
  - Computation power/GPU
  - Large datasets
  - Simple yet powerful cleverness



# COMPUTER VISION

- Object and activity recognition



Large-scale Video Classification with Convolutional Neural Networks, CVPR 2014

- Object segmentation



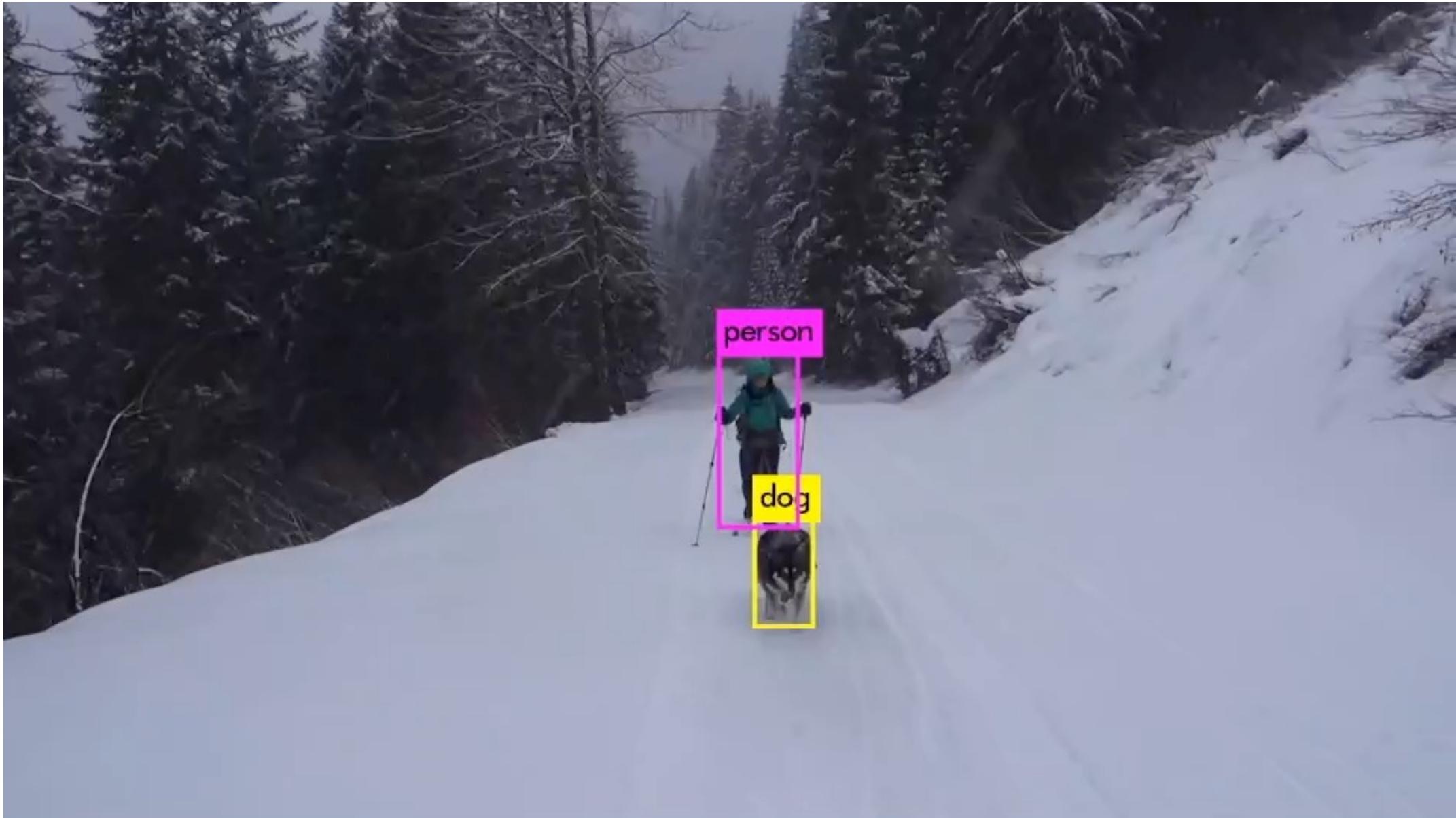
Microsoft Deep Learning Semantic Image Segmentation

- Image captioning



NeuralTalk and Walk, recognition, text description of the image while walking

# YOLO ALGORITHM



# ROBOTICS

- Self driving cars
- Drones & robots
- Games (Go, Atari, poker....)



Tip for watching: the alpha go documentary:

<https://youtu.be/WXuK6gekU1Y>



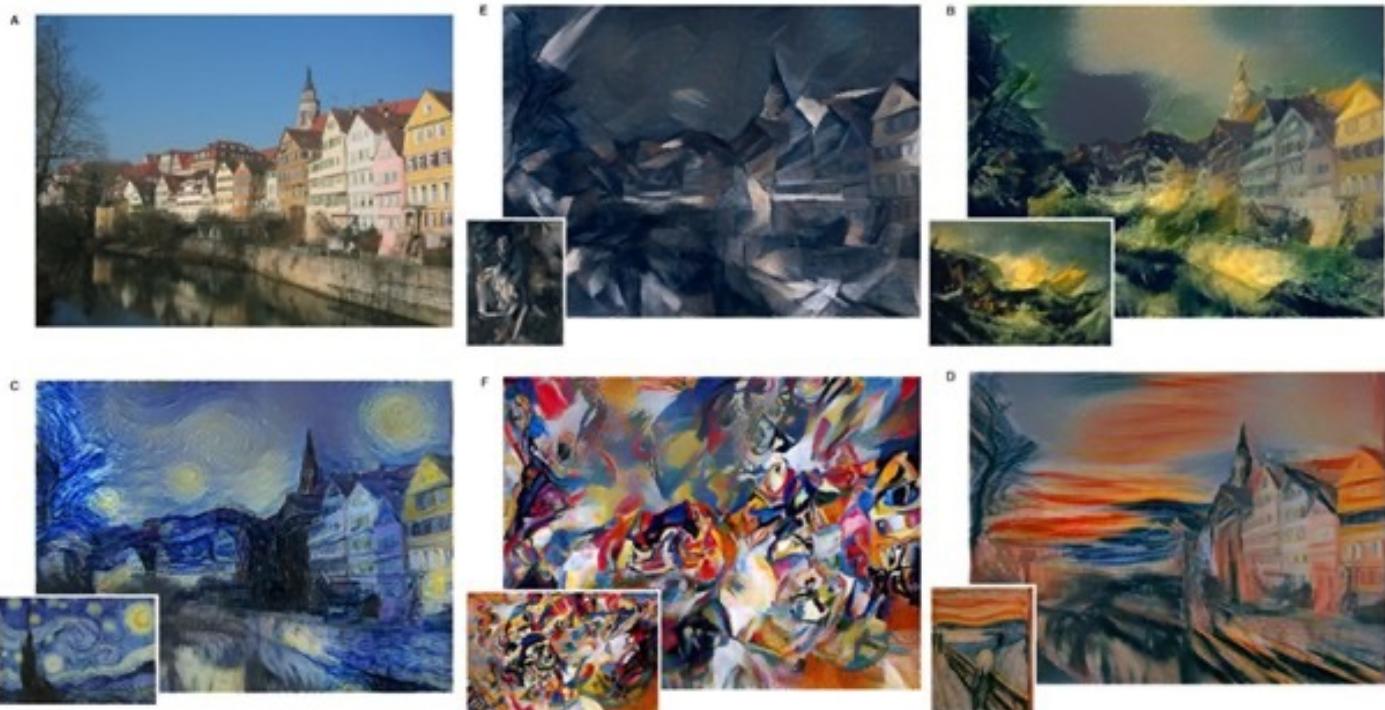
Autonomous Helicopter - Airshow #1

# NLP AND SEQUENCE PROCESSING

- Speech recognition
- Generating voices: <https://deepmind.com/blog/article/wavenet-generative-model-raw-audio>
- Machine translation or question answering

## OTHER COOL PROJECTS

- Imitating famous painters
- Create music
- Create handwriting



A haiku written by Ray Kurzweil's Cybernetic Poet  
after reading poems by John Keats and Wendy Dennis

*You broke my soul  
the juice of eternity,  
the spirit of my lips.*

Hi Motherboard readers!

This entire post was hand written by a neural network.  
(It probably writes better than you.)

Of course, a neural network doesn't actually have hands

and the original text was typed by me, a human.

So what's going on here?

A neural network is a program that can learn to follow a set of rules

But it can't do it alone. It needs to be trained.

This neural network was trained on a corpus of writing samples.

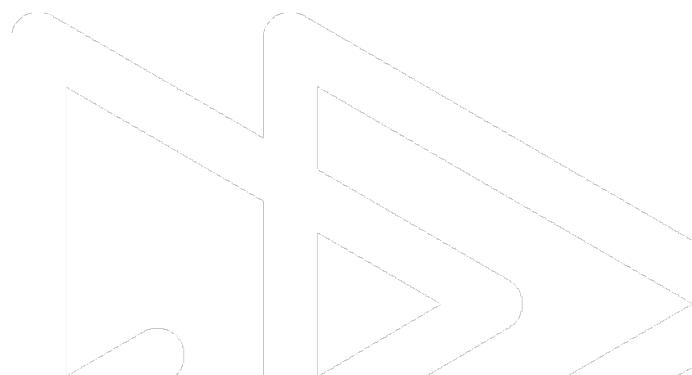
# THE ILSVC CHALLENGE

2012 Teams		%error	2013 Teams		%error	2014 Teams		%error	CNN based non-CNN based
Supervision (Toronto)		15.3	Clarifai (NYU spinoff)		11.7	GoogLeNet		6.6	
ISI (Tokyo)		26.1	NUS (singapore)		12.9	VGG (Oxford)		7.3	
VGG (Oxford)		26.9	Zeiler-Fergus (NYU)		13.5	MSRA		8.0	
XRCE/INRIA		27.0	A. Howard		13.5	A. Howard		8.1	
UvA (Amsterdam)		29.6	OverFeat (NYU)		14.1	DeeperVision		9.5	
INRIA/LEAR		33.4	UvA (Amsterdam)		14.2	NUS-BST		9.7	
			Adobe		15.2	TTIC-ECP		10.2	
			VGG (Oxford)		15.2	XYZ		11.2	
			VGG (Oxford)		23.0	UvA		12.1	

Figure taken from Y. LeCun's CVPR 2015 plenary talk

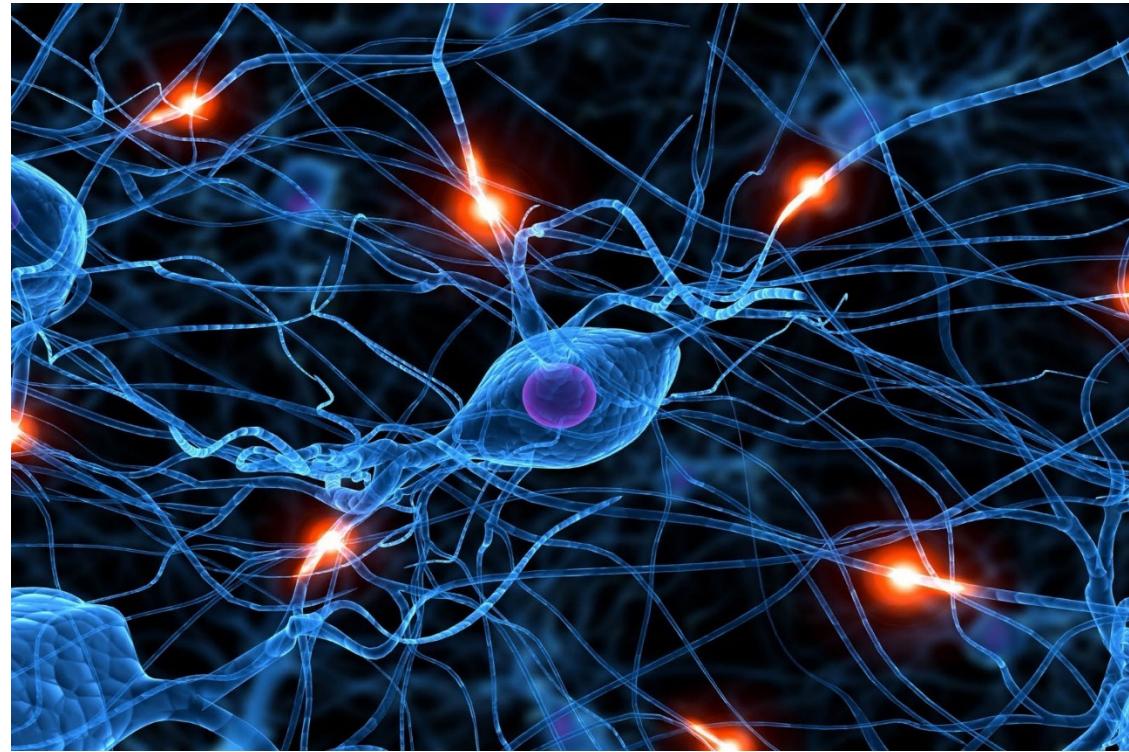
# SO, WHY NOW?

1. Better hardware
2. Bigger data
3. Better regularization methods, such as dropout
4. Better optimization protocols, such as batch normalization



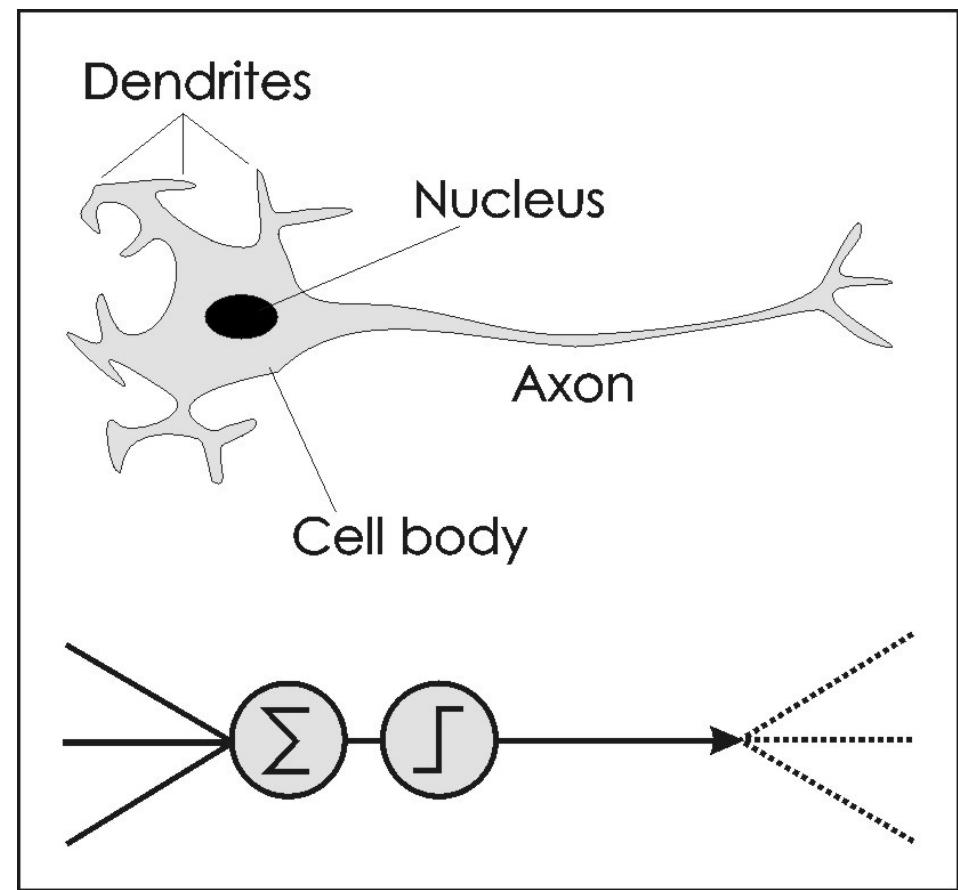
# THE BRAIN

- The brain has neurons
- Connected by synapses
- $10^{11}$  neurons per brain
- $10^4$  connections per neuron

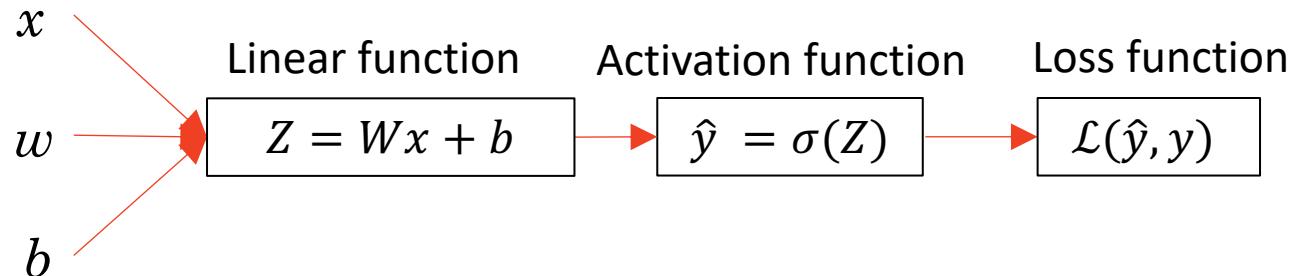
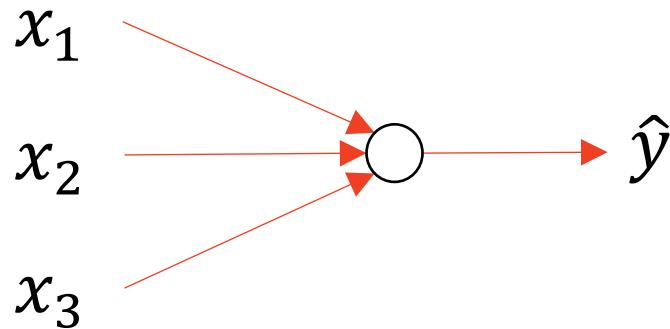


# THE BRAIN

- Neuron can have many input and many output connections
- Electrical signal is fired which is a complex time series of spikes.
- Mathematical simplification
  - Only focus on neuron signaling
  - One output value instead of series of spikes

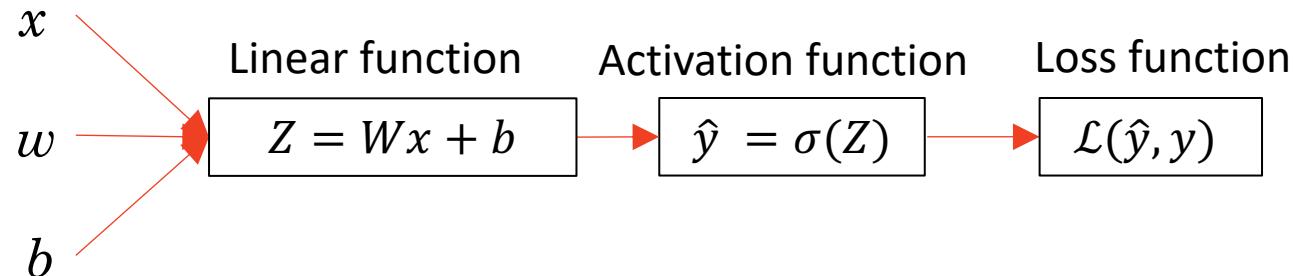
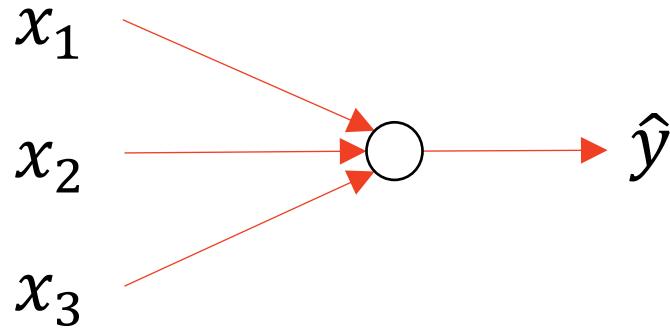


# NEURAL NETWORK



- The linear function is the linear model we have seen in simpler models
- The activation function  $\sigma$  is the innovation that makes sure stacking layers can solve non-linear data
- The loss function is what determines the performance.

# NEURAL NETWORK



How does the model learn the correct weights  $W$  (or  $\theta$ , if we use another symbol)? We will use gradients to update  $W$ .

Note this symbol:  $\partial$ . If you are unfamiliar with that symbol, you can read it as: take the derivative of the Loss function, with respect to variable  $w$ .

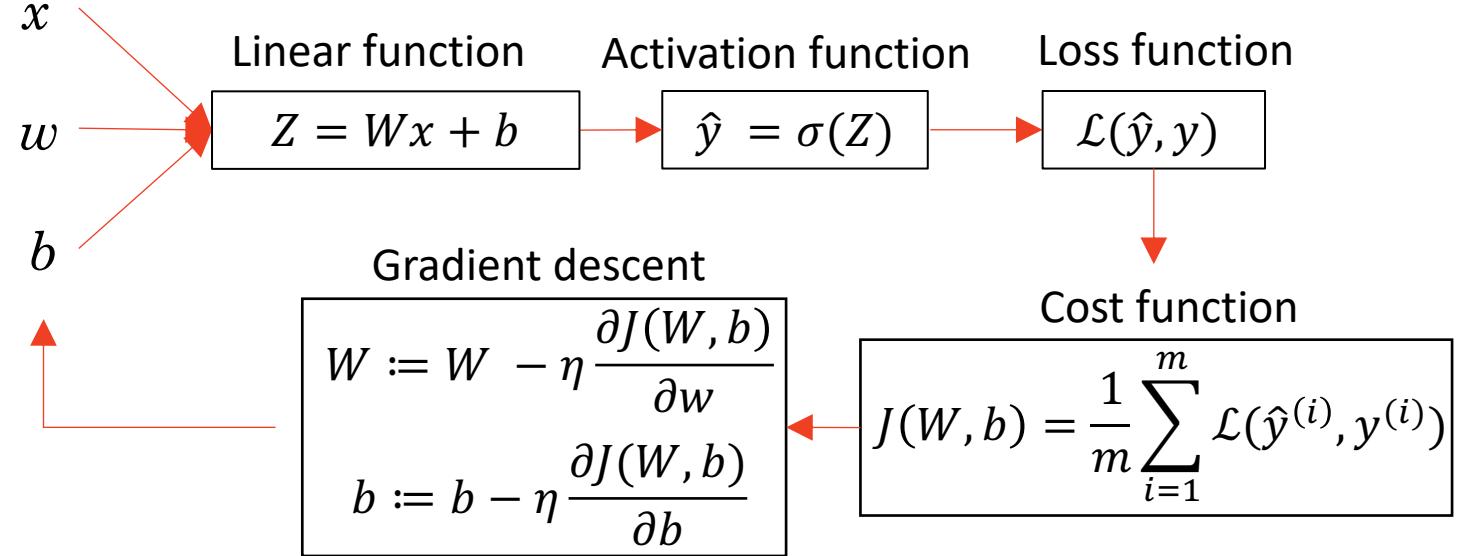
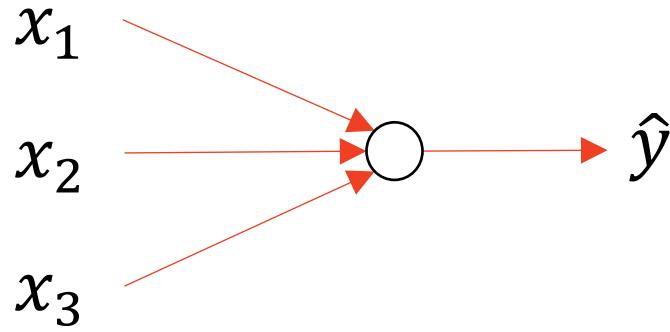
$\frac{\partial J(W,b)}{\partial w}$  means: find out to what extend the  $w$  variable is responsible for the error we make in the  $J(W, b)$  function.

$$W := W - \eta \frac{\partial J(W, b)}{\partial w}$$

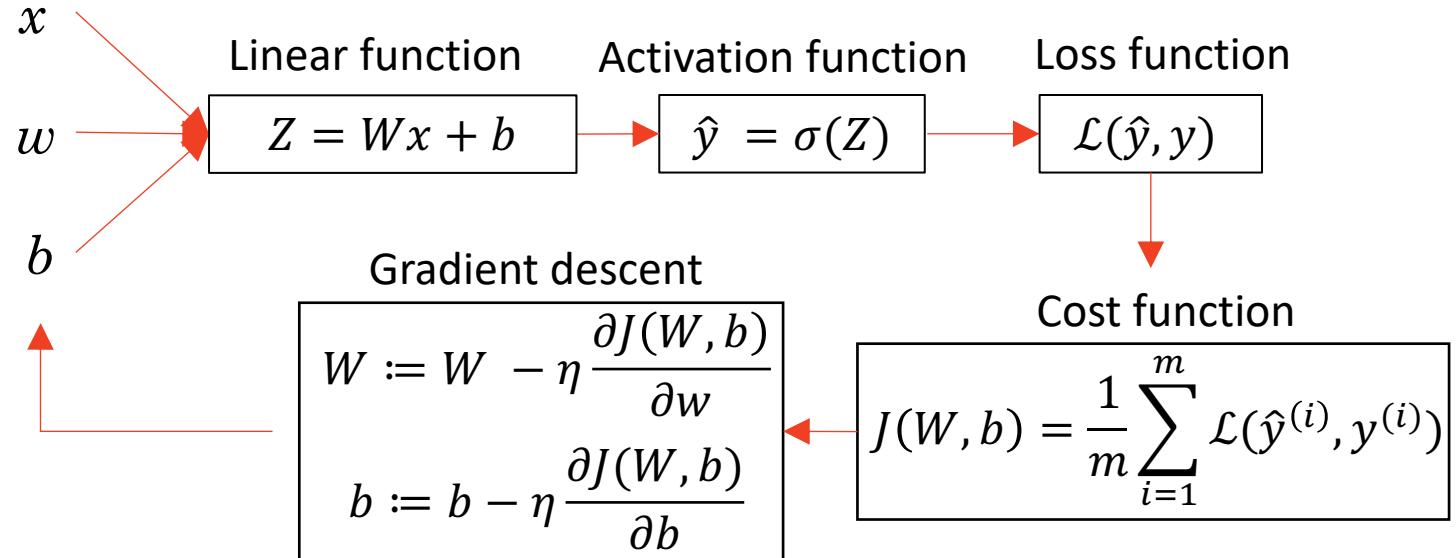
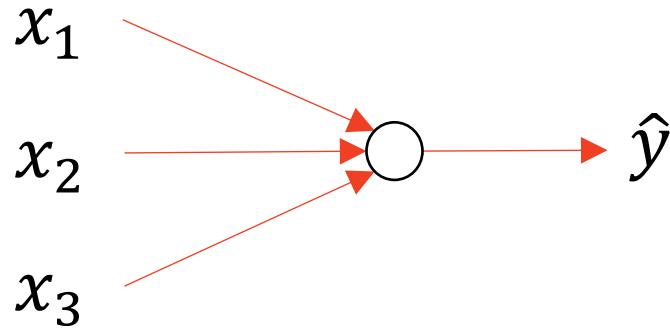
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

We will then use that answer to correct the weights we used by a fraction  $\eta$ . Variables that attribute more to the error, will be corrected more.

# NEURAL NETWORK



# NEURAL NETWORK



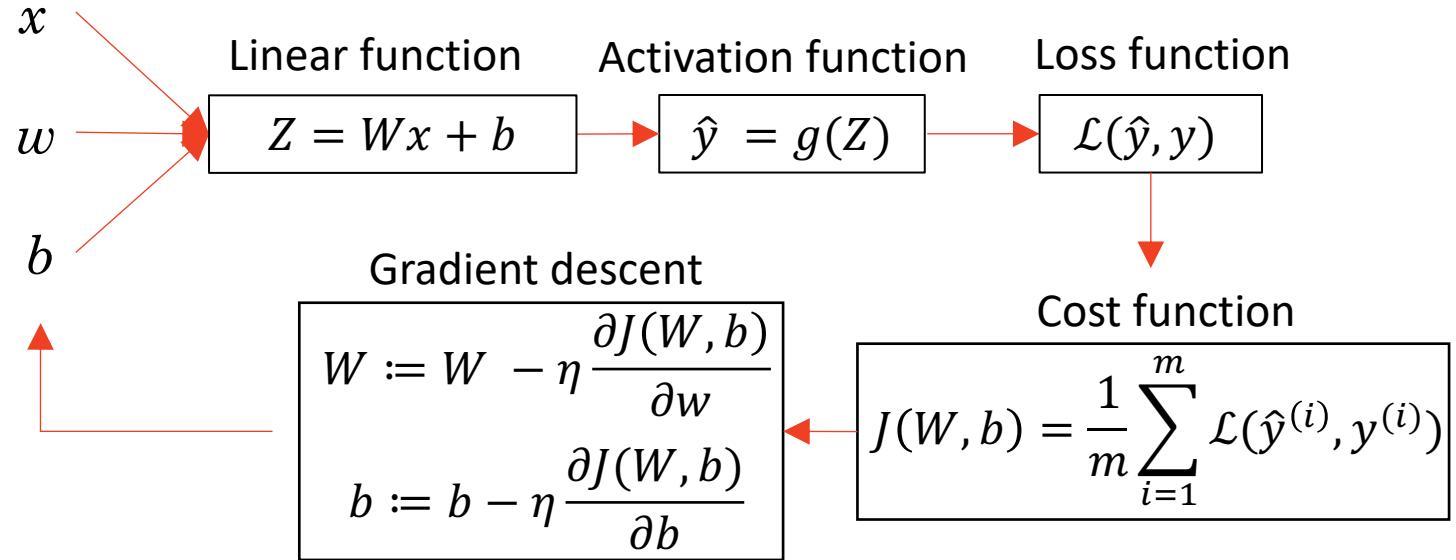
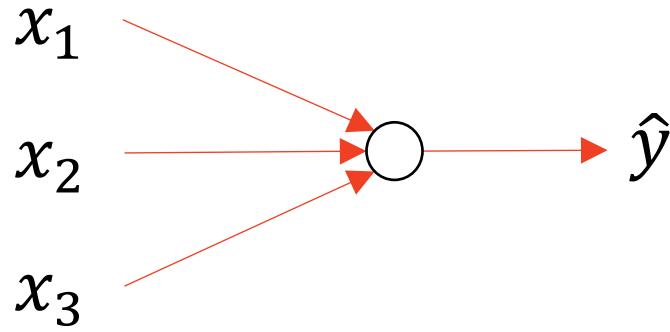
Activation functions:

$$\text{Sigmoid: } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\text{ReLU: } \sigma(z) = \max(0, z)$$

$$\text{tanh: } \sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

# NEURAL NETWORK



Activation functions:

$$\text{Sigmoid: } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\text{ReLU: } \sigma(z) = \max(0, z)$$

$$\text{tanh: } \sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

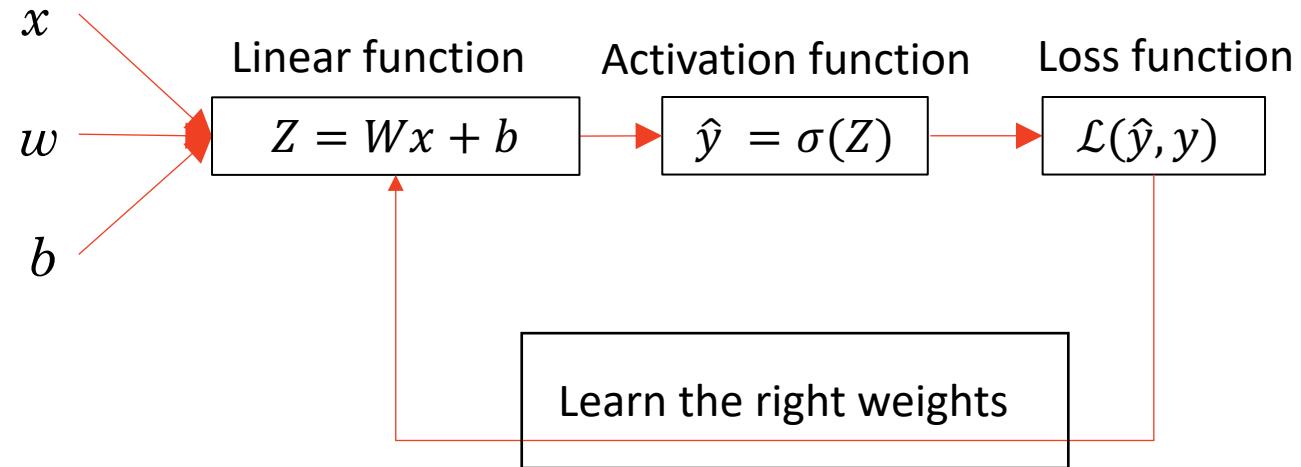
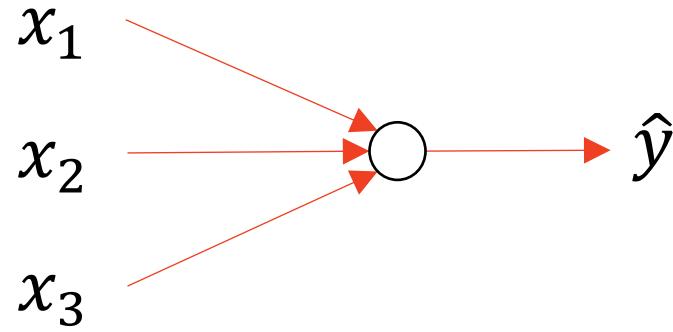
Loss functions:

$$\text{Cross entropy loss: } \mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

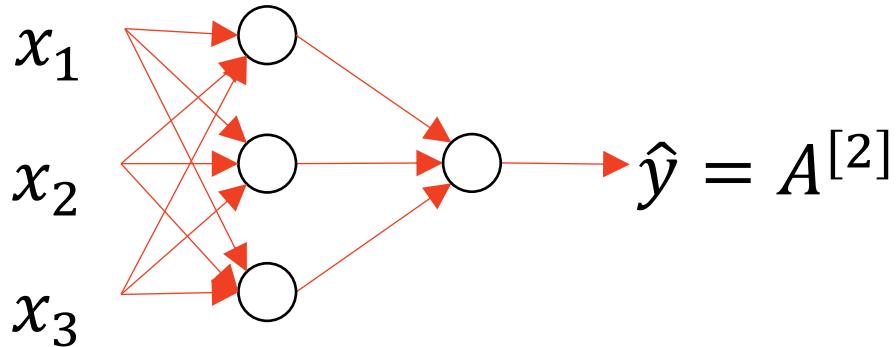
$$\text{Squared loss: } \mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$$

etc.

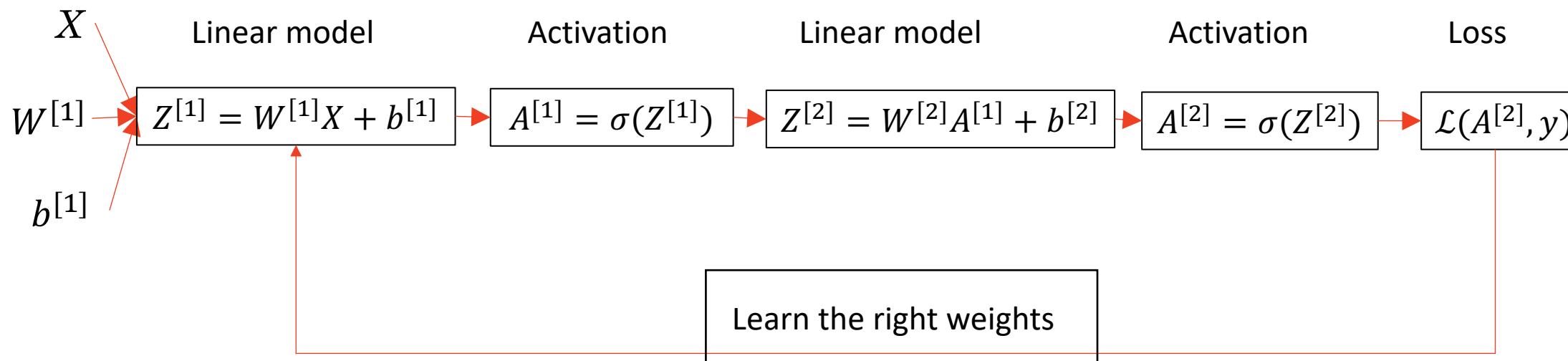
# NEURAL NETWORK



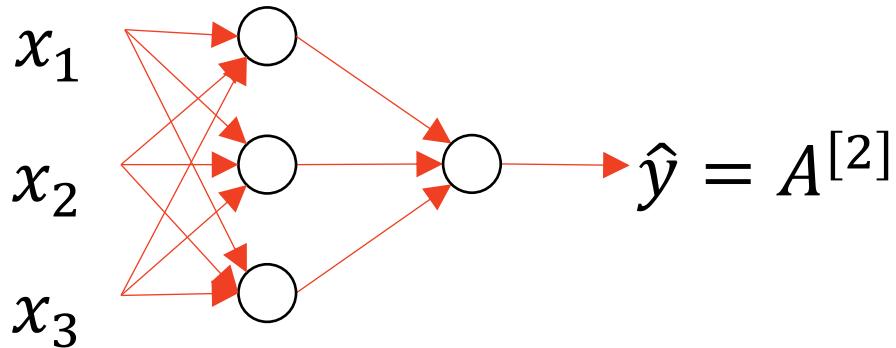
# NEURAL NETWORK



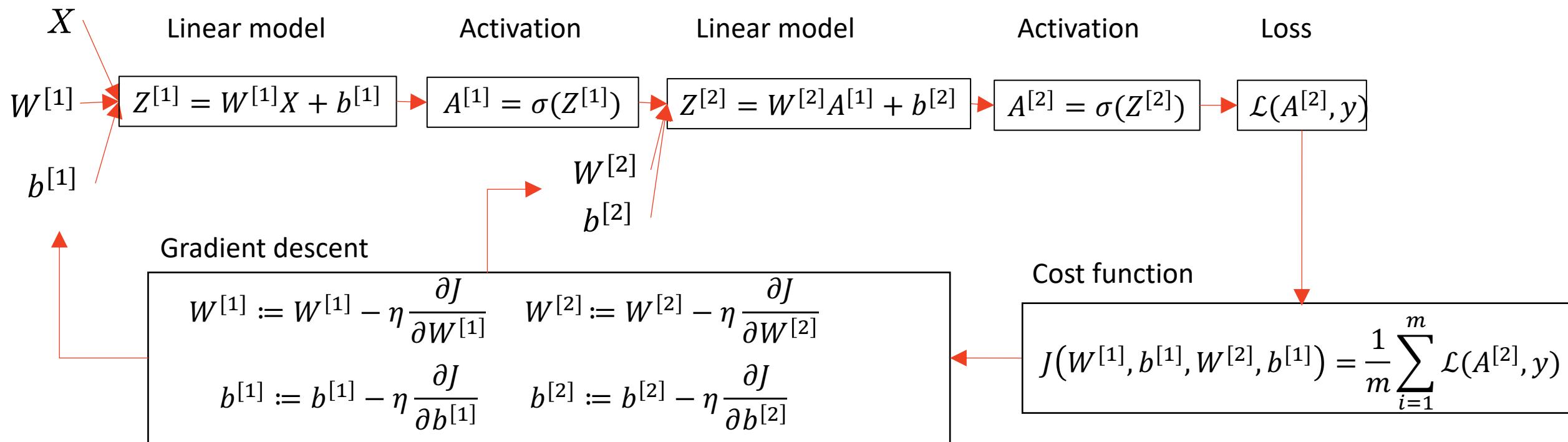
We can scale this up to more layers.



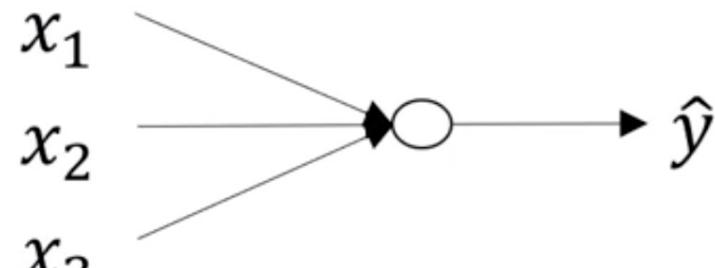
# NEURAL NETWORK



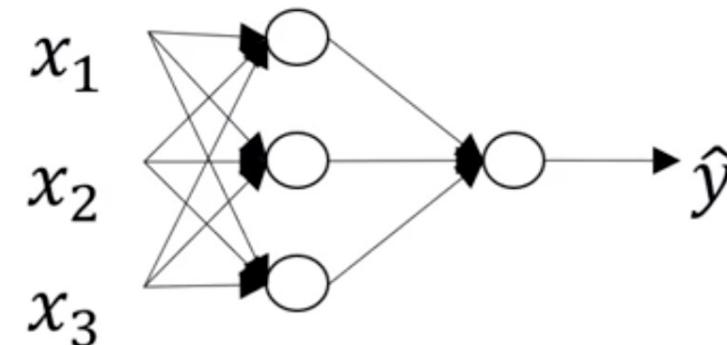
The learning will get more complex, but this is all handled by the tensorflow backend.



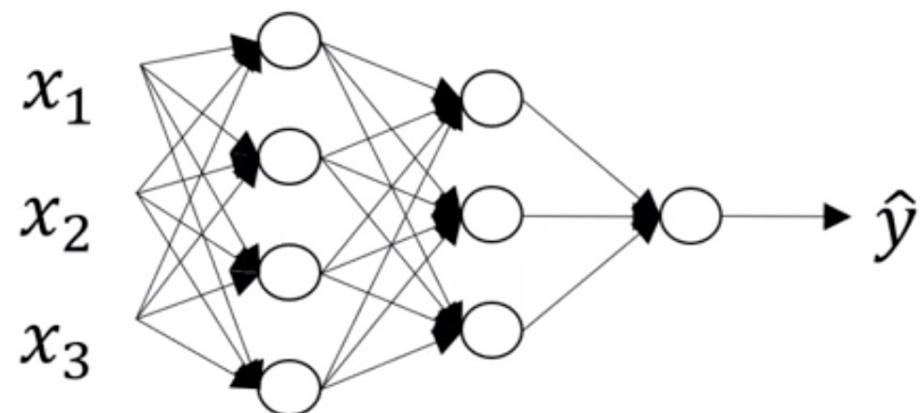
# SHALLOW AND DEEP NEURAL NETWORK



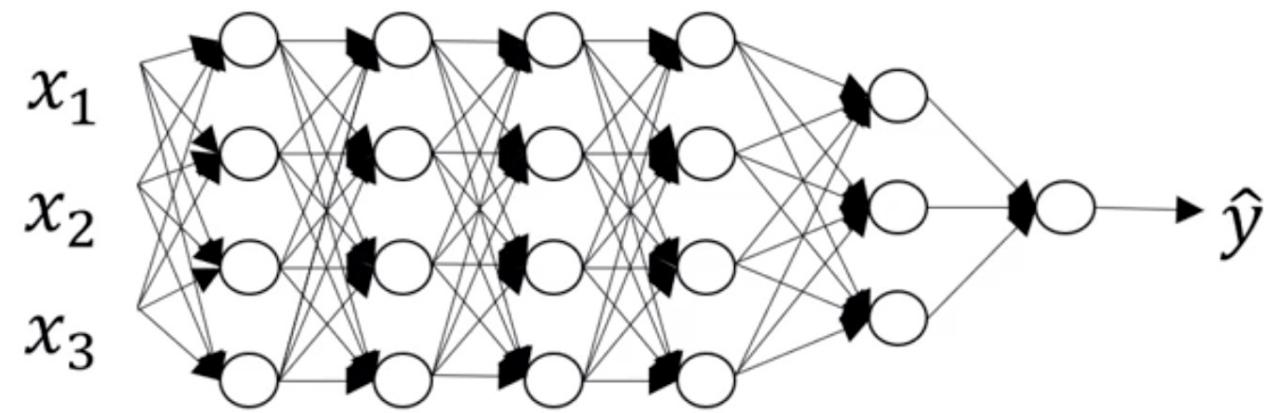
1-layer neural network



2-layer neural network



3-layer neural network



6-layer neural network

## SO, WHY DEEP AND NOT SHALLOW?

Although with two-layer (shallow) network, we can approximate all possible functions, given the network layers are wide enough:

- Deep architectures tend to be more efficient
- Also, deep and narrow architectures tend to **generalize** better than shallow and wide architectures

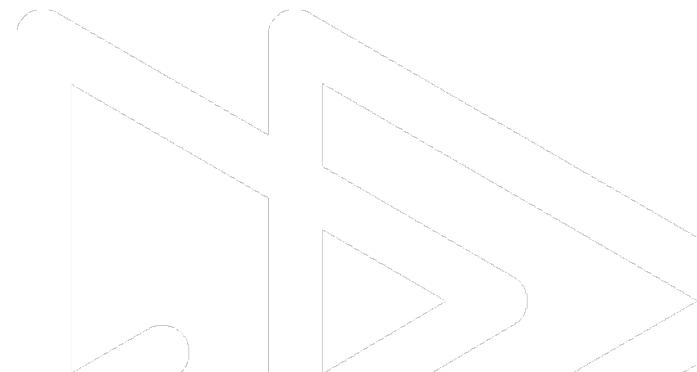
# PARAMETERS VS HYPERPARAMETERS

- Parameters
  - $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}, \dots$
- Hyperparameters
  - Learning rate  $\alpha$
  - Number of iterations
  - Number of hidden layers
  - Number of units per hidden layer
  - Choice of activation function

# ISSUES WITH TRAINING

# OVERVIEW OF ISSUES

- Gradient descent
- Learning rate
- Lots of layers and inputs
  - Demands more data
  - Overfitting
  - Vanishing gradients



# GRADIENT DESCENT IMPROVEMENTS

- A few slides back we had the following equations:

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

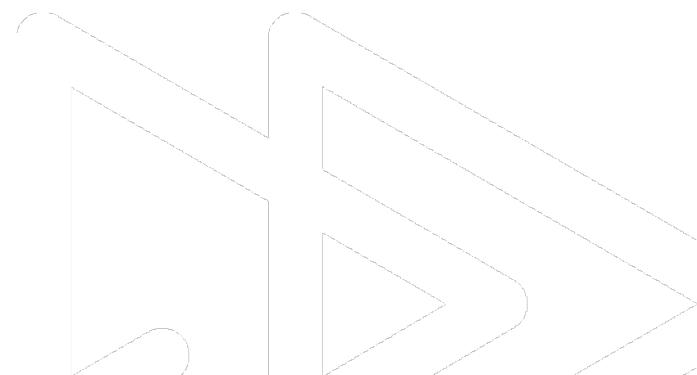
- Most (all?) neural net libraries use more advanced ways of implementing gradient descent.
- One concept is mini-batch gradient descent

## MINI-BATCH GRADIENT DESCENT

- Standard gradient descent computes error gradient over entire training set

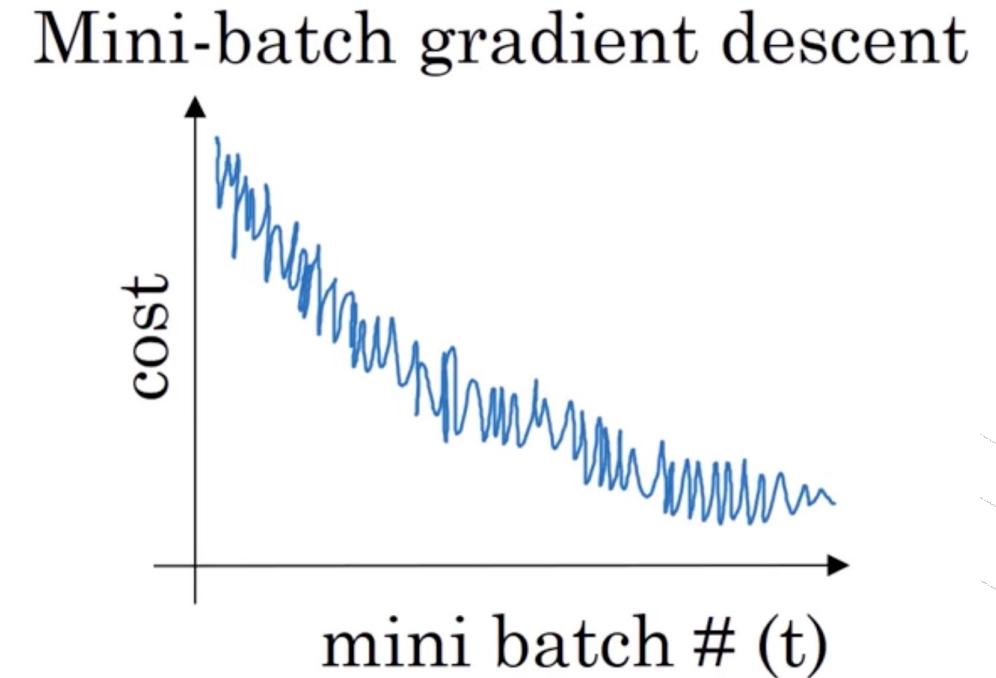
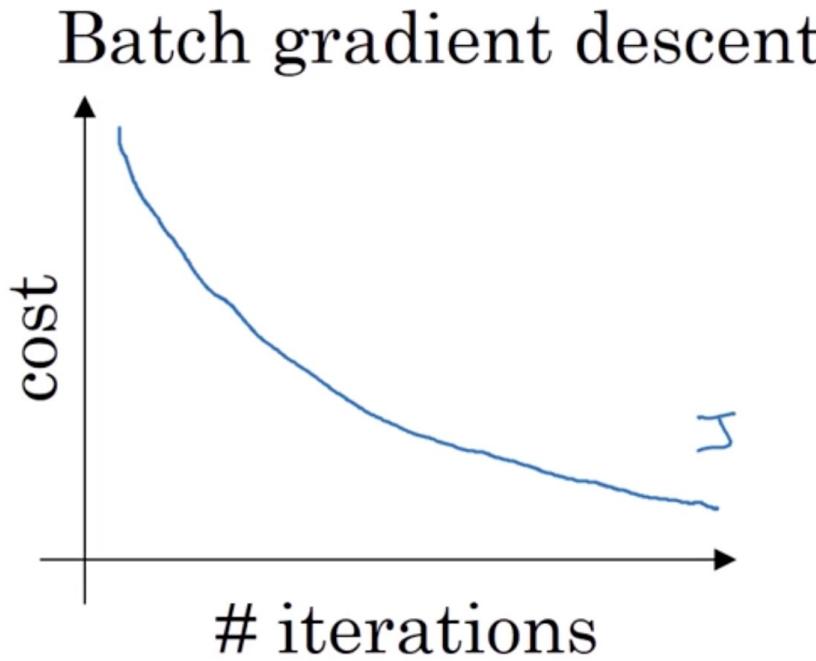
$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

- This can be very slow for large datasets and impossible for datasets which don't fit into memory!
- Mini-batch gradient descent (SGD) applies gradient descent on x training example



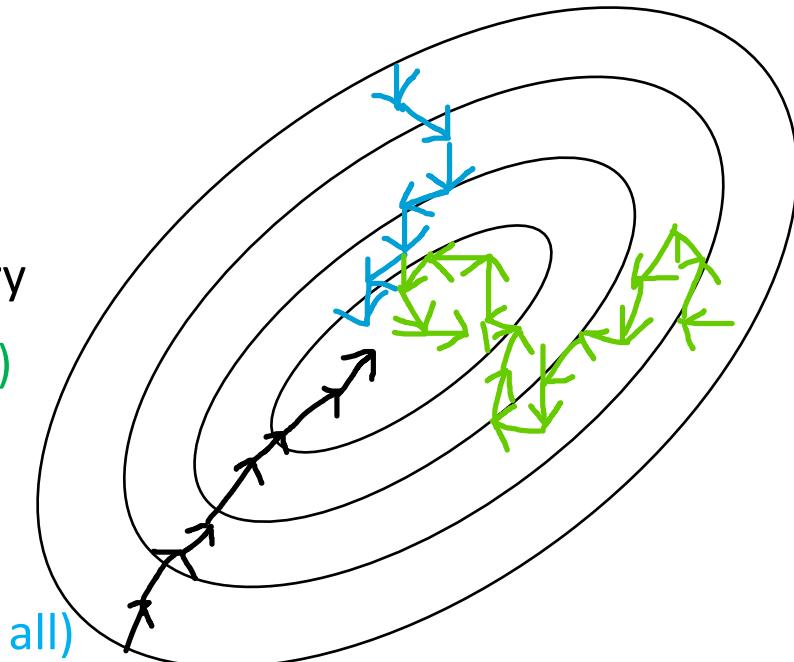
## MINI-BATCH GRADIENT DESCENT

- Mini-batch is an improvement over gradient descent in terms of speed.
- Complicates the convergence process (see figure).



## MINI-BATCH GRADIENT DESCENT

- Batch gradient descent (all data at once)
  - Too long per iteration to update parameters
  - Not able to process dataset that don't fit into memory
- Stochastic gradient descent (one observation at a time)
  - Introduces much noise
  - Not able to perform vectorized calculations
- Mini-batch gradient descent (more than one, less than all)
  - Able to use vectorization
  - Able to quickly update parameters



# MINI-BATCH GRADIENT DESCENT

- Mini-batch gradient descent solves this unstable convergence process by computing the gradient over a subset of the training data

$$J(w, b) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

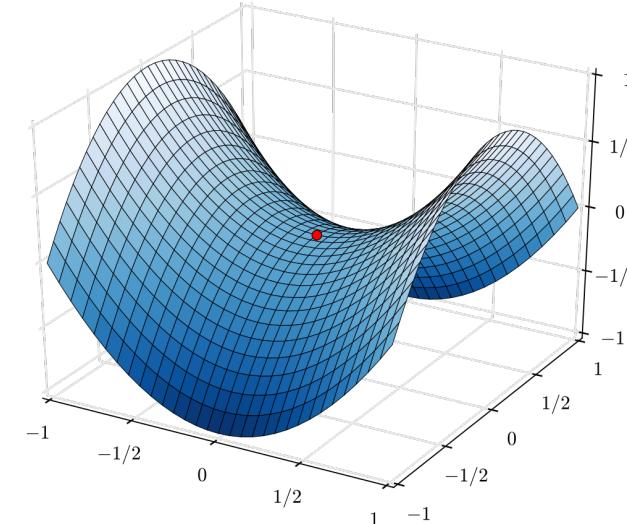
- Here, if  $N$  is the number of mini-batch training samples,  $n \ll N$ .
- A common mini-batch sizes is 32.
- In literature, SGD usually refers to mini-batch gradient descent.

# MOMENTUM - SPEED

- SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another.
- Momentum is a method which helps SGD move in the right direction by adding a fraction of the previous update to the current update. It is like a ball rolling down a hill; once it has speed, it doesn't stop suddenly at a tiny bump but continues because of the momentum.

$$v_{d\theta} = \beta_1 v_{d\theta} + (1 - \beta_1) \frac{\partial L}{\partial \theta}$$
$$\theta = \theta - \eta v_{d\theta}$$

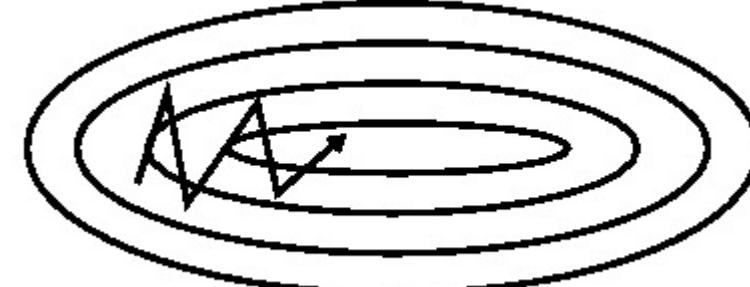
- $\beta_1$  is usually chosen to be around 0.9



SGD without momentum



SGD with momentum



# RMSPROP - DIRECTION

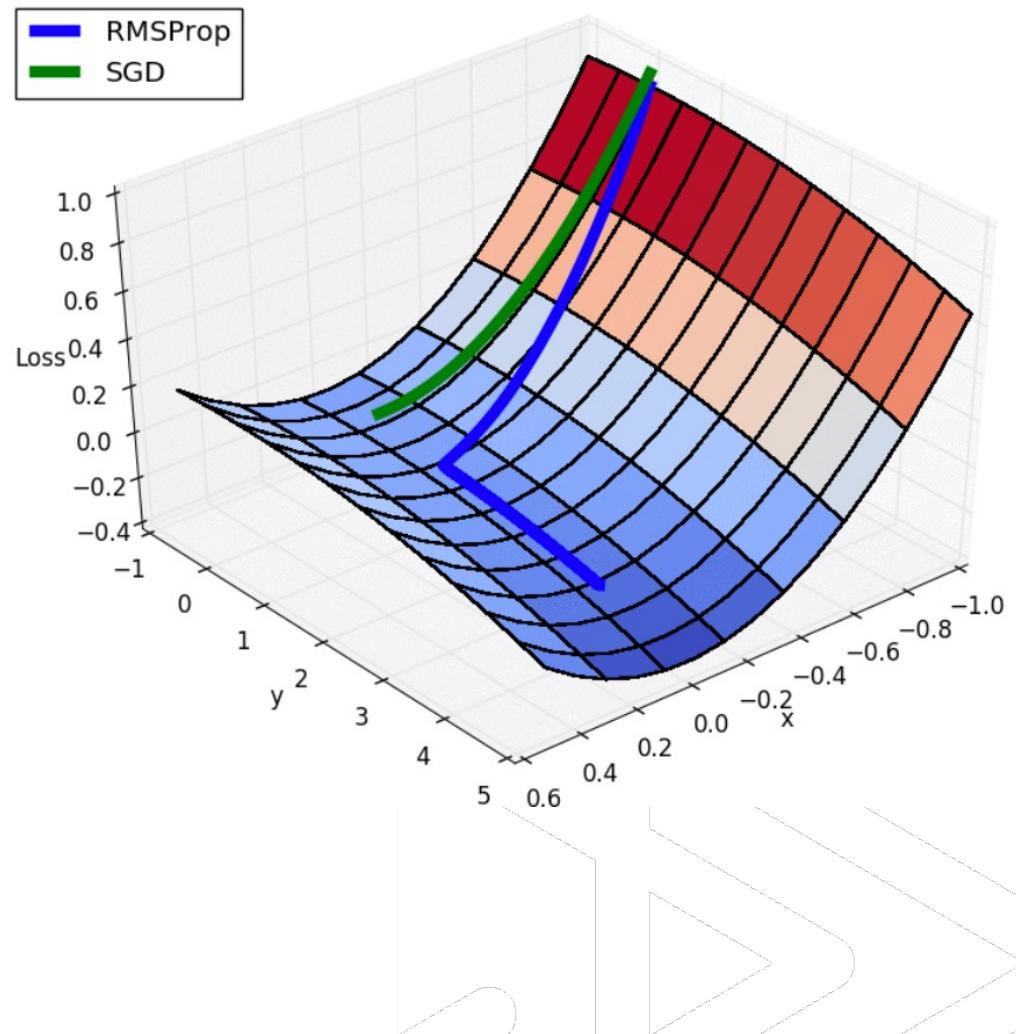
Momentum makes the speed adaptive. RMSPROP adjusts the direction in an early stage; it does so by scaling the gradient along the steepest dimensions.

$$s \leftarrow \beta s + (1 - \beta) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta - \frac{\eta \nabla_{\theta} J(\theta)}{\sqrt{s} + \epsilon}$$

1. First, the gradients are squared  $\nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$  and added to the vector  $s$ .  $\otimes$  stands for pair-wise multiplication (so, squaring every value in the vector).
2. Then, the gradient is scaled down by the square of  $s$ .  $\epsilon$  is a small term to avoid division by zero, e.g.  $10^{-10}$

This decays the learning rate, but does so faster for steep dimensions than for dimensions with a gentler slope. This helps to point earlier in the right direction.



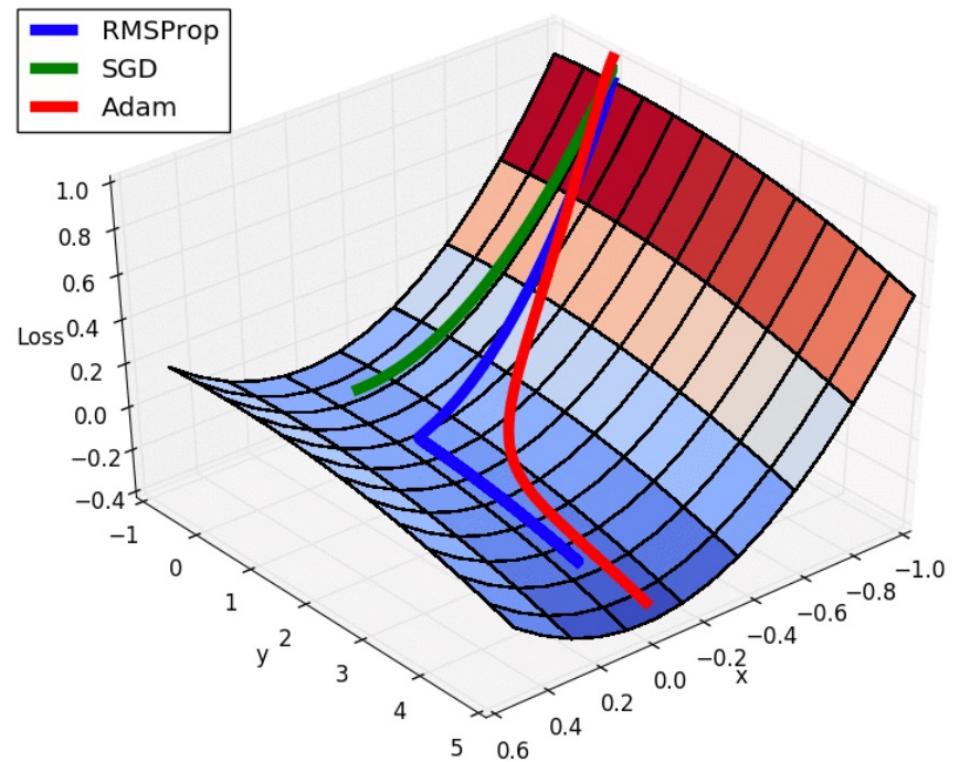
# ADAM

ADAM stands for adaptive moment estimation and combines the two ideas.

$$v_\theta = \beta_1 v_\theta + (1 - \beta_1) \frac{\partial L}{\partial \theta}$$

$$s_\theta = \beta_1 s_\theta + (1 - \beta_1) \left( \frac{\partial L}{\partial \theta} \right)^2$$

$$\theta \leftarrow \theta - \eta \frac{v_\theta}{\sqrt{s_\theta} + \text{eps}}$$



# OVERVIEW OF ISSUES

- Gradient descent
- Learning rate
- Lots of layers and inputs
  - Demands more data
  - Overfitting
  - Vanishing gradients

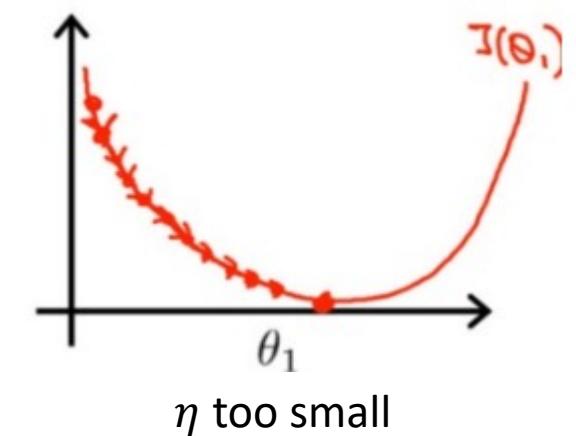
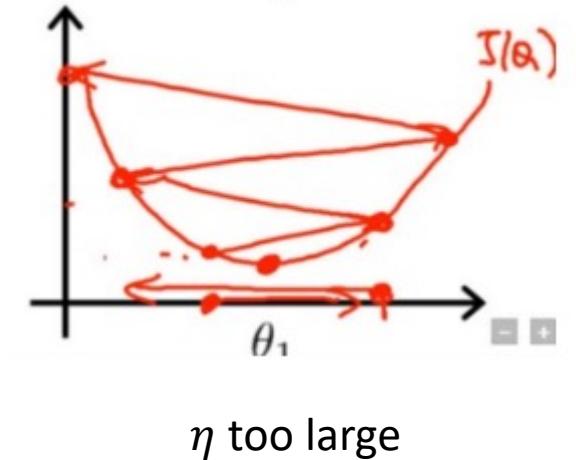
# LEARNING RATE

- Determining the learning rate  $\eta$  is an important part of our training process.
- Choosing a  $\eta$  that is too large will cause our gradient descent method to overshoot.
- If  $\eta$  is too small, gradient descent converges very slowly.

# LEARNING RATE DECAY

The rate starts high, and drops every few epochs with some rate (e.g. halves every 10 epochs)

$$\eta = \frac{1}{1 + decay\_rate * epoch\_num} \eta_0$$



# OVERVIEW OF ISSUES

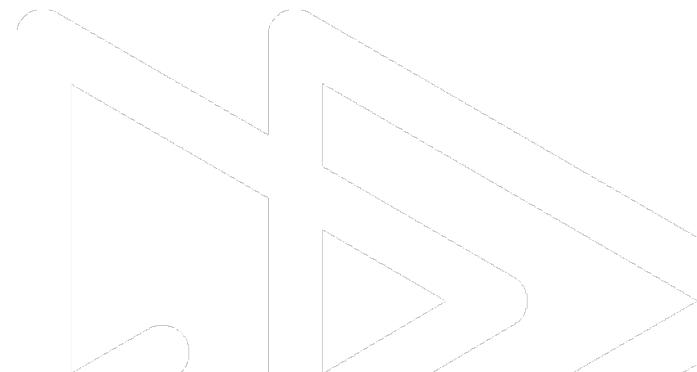
- Gradient descent
- Learning rate
- Lots of layers and inputs
  - Demands more data
  - Vanishing gradients
  - Overfitting

## FIRST ISSUE: THE DEMAND OF DATA



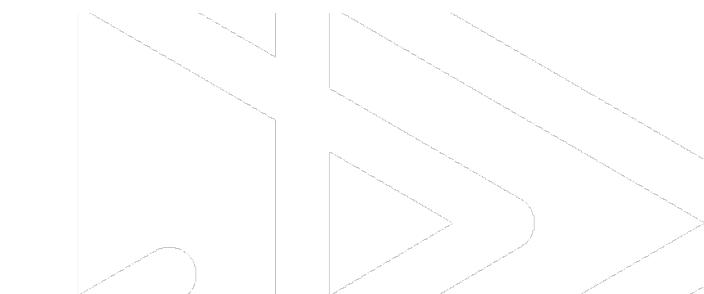
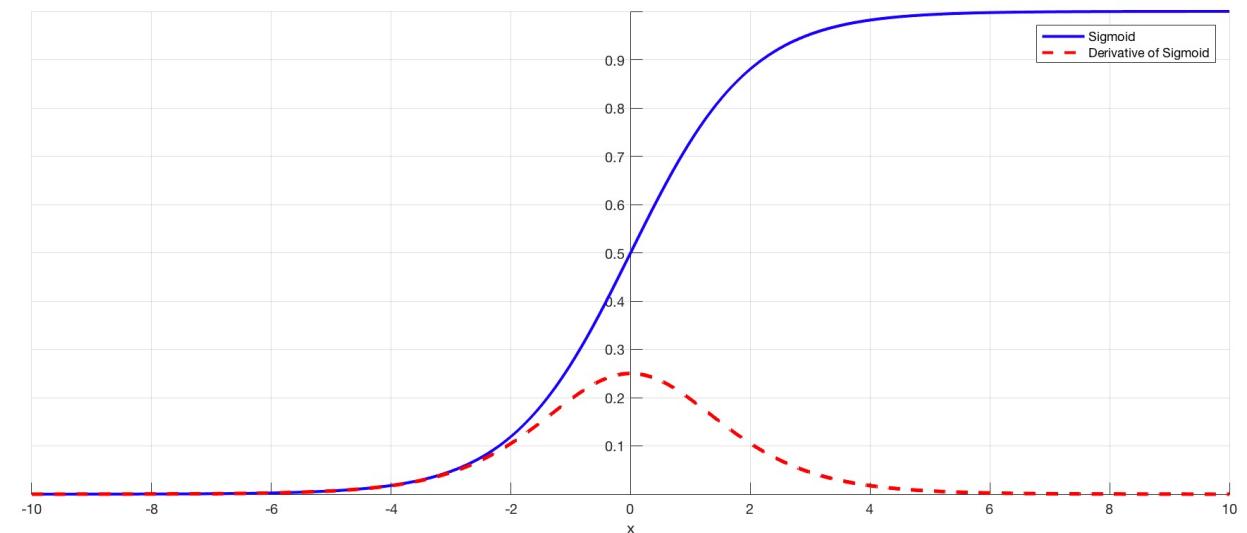
# OVERVIEW OF ISSUES

- Gradient descent
- Learning rate
- Lots of layers and inputs
  - Demands more data
  - **Vanishing gradients**
  - Overfitting

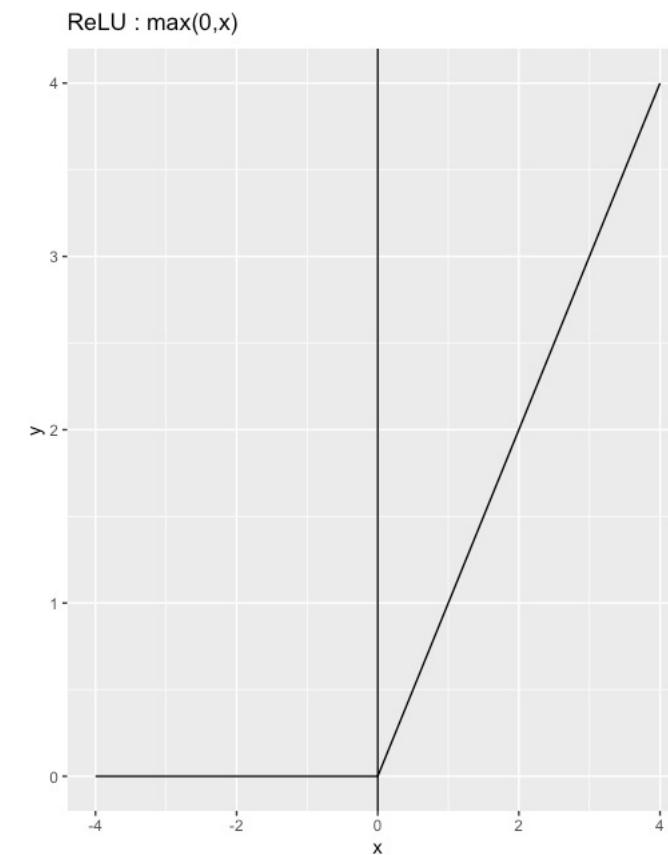
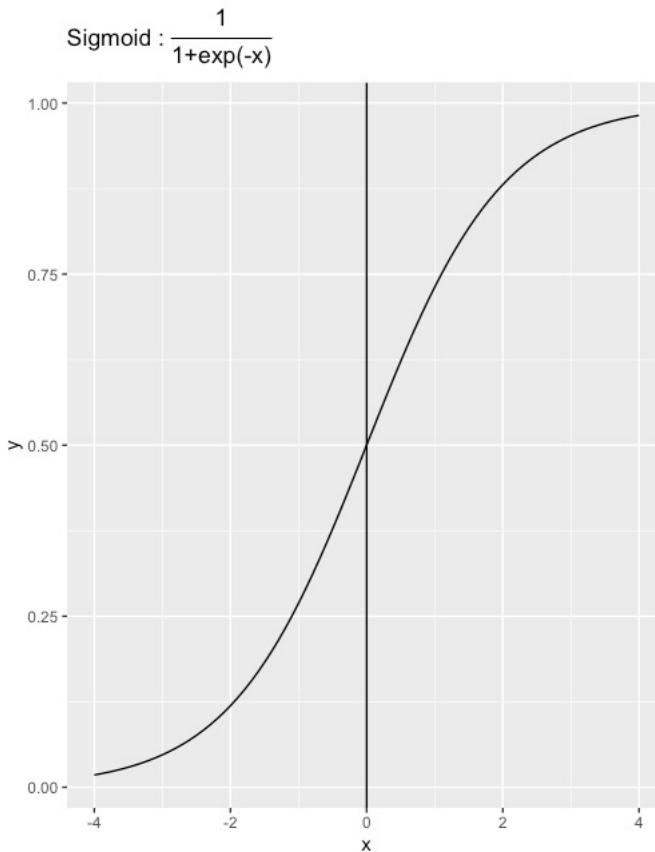


## SECOND ISSUE: VANISHING GRADIENTS

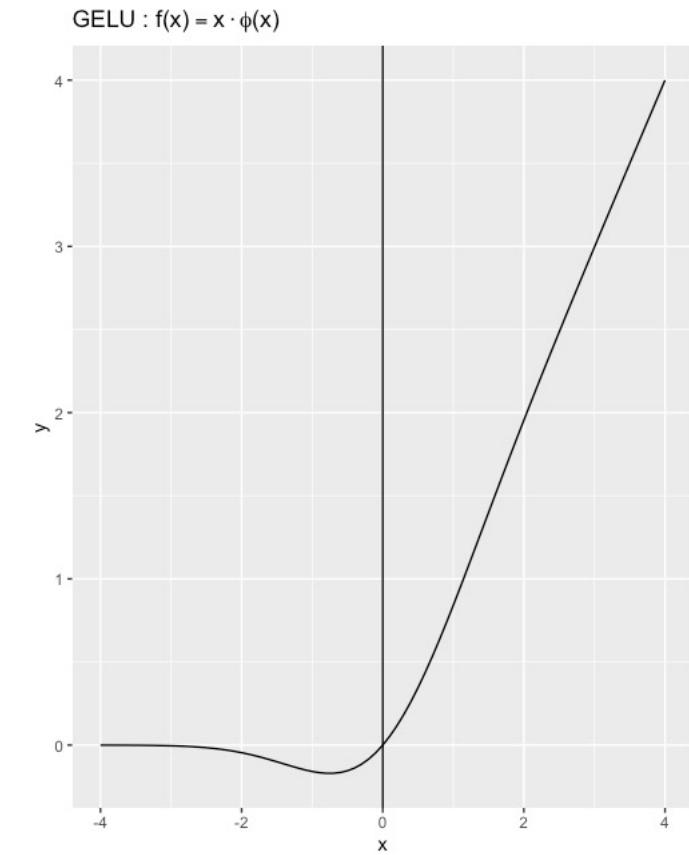
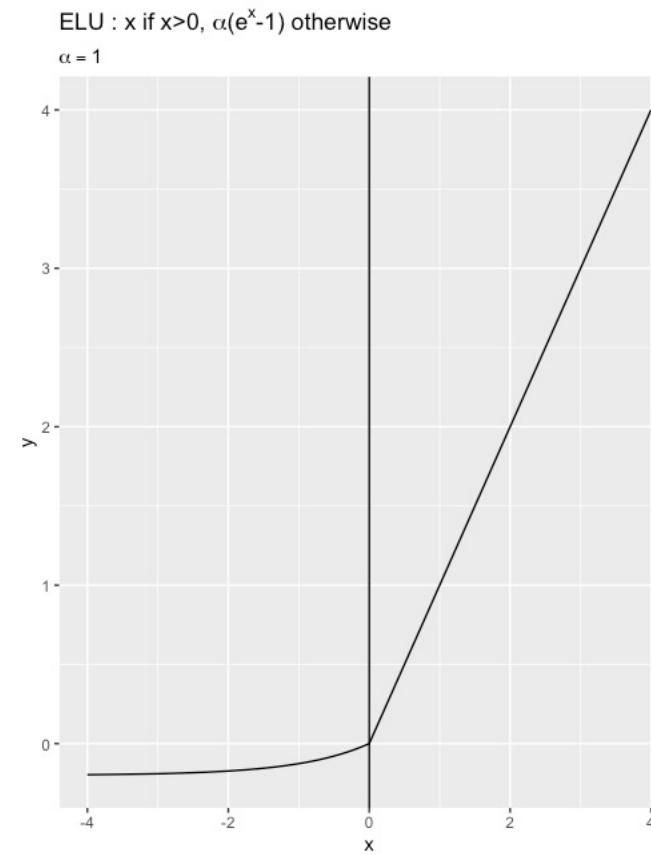
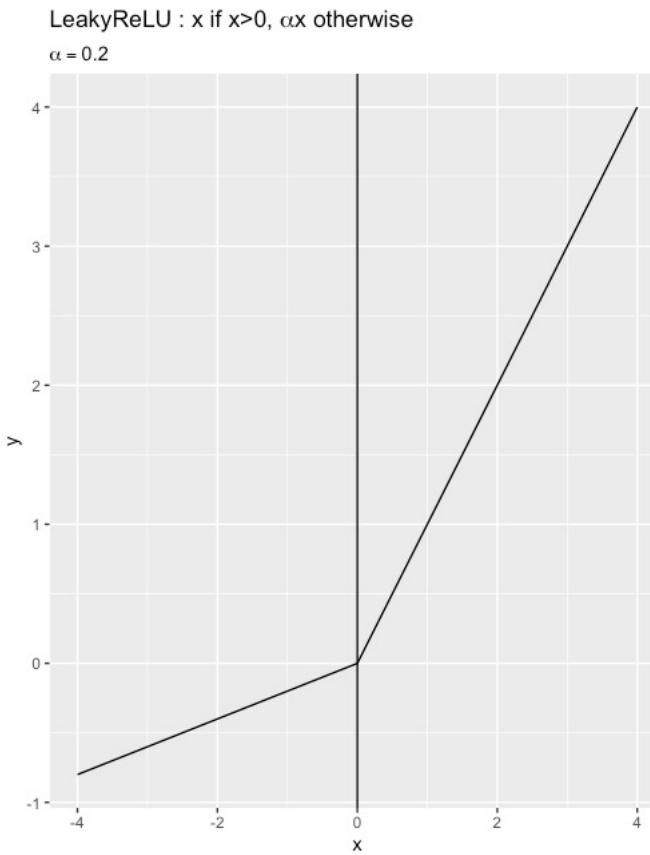
- More layers are added to the neural network, the gradients of the loss function approaches zero, making the network hard to train.
- Solution
  - Activation function
    - RELU instead of sigmoid
  - Better initialization of the weights
    - `he_normal` ->  $\text{stddev} = \sqrt{2 / N_{\text{in}}}$
    - `glorot_normal (Xavier)` ->  $\text{stddev} = \sqrt{2 / (N_{\text{in}} + N_{\text{out}})}$
  - Residual networks



## SECOND ISSUE: VANISHING GRADIENTS

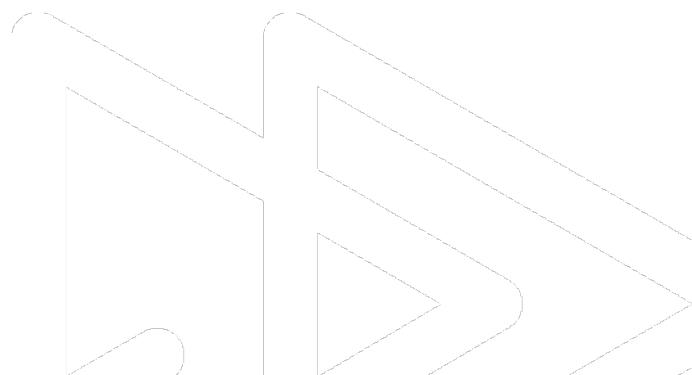


## SECOND ISSUE: VANISHING GRADIENTS



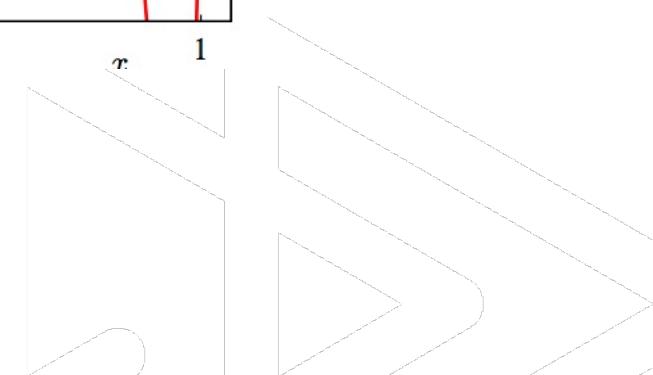
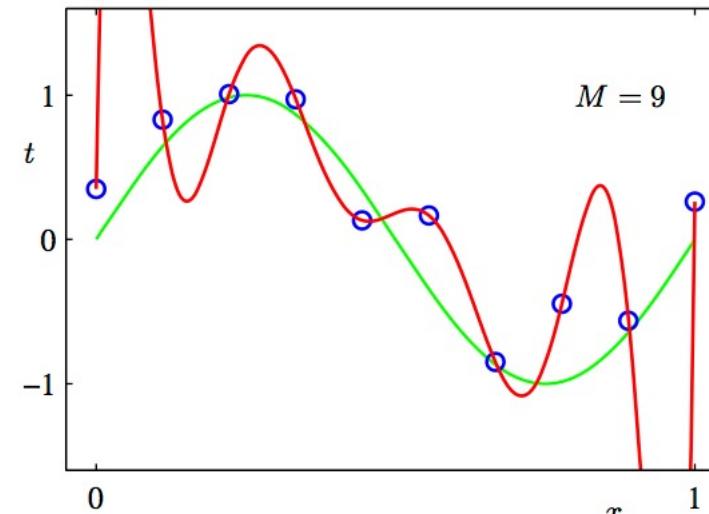
# OVERVIEW OF ISSUES

- Gradient descent
- Learning rate
- Lots of layers and inputs
  - Demands more data
  - Vanishing gradients
  - Overfitting



## THIRD ISSUE: OVERFITTING

- This happens due to the enormous amount of parameters
  - Yields high flexibility/nonlinearity
  - Potentially ‘remember’ outliers
  - Inability to generalize
- Countermeasures
  - Weight norm penalty
  - Early stopping
  - Dropout
  - ...



# WEIGHT PENALTIES

- A weight should only code for **regular** patterns
- Idea: force network to forget **irregularities**
- Accomplished by **augmenting loss function**
- Example
  - Classification → cross-entropy loss with L2 loss:

$$L = \sum_{i=1}^K y_i \log(\hat{y}_i) + \lambda \|\mathbf{w}\|_2^2$$

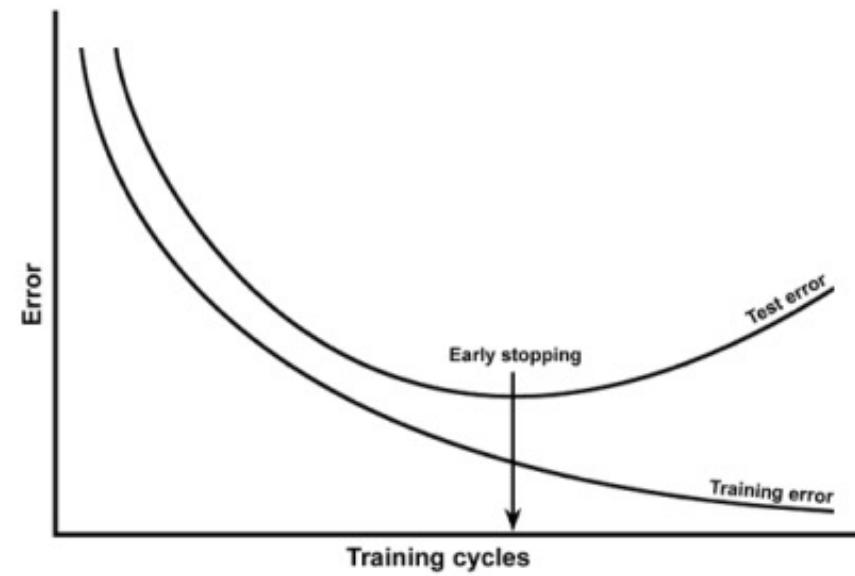
- Regression → sum of squared errors with L1 loss:

- 

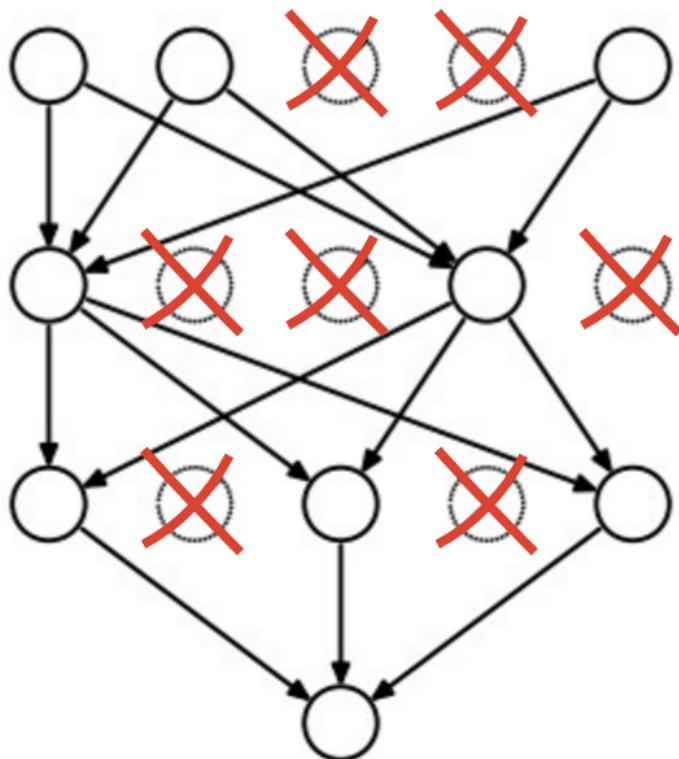
$$L = \sum_{i=1}^K (y_i - \hat{y}_i)^2 + \lambda \|\mathbf{w}\|_1$$

# EARLY STOPPING

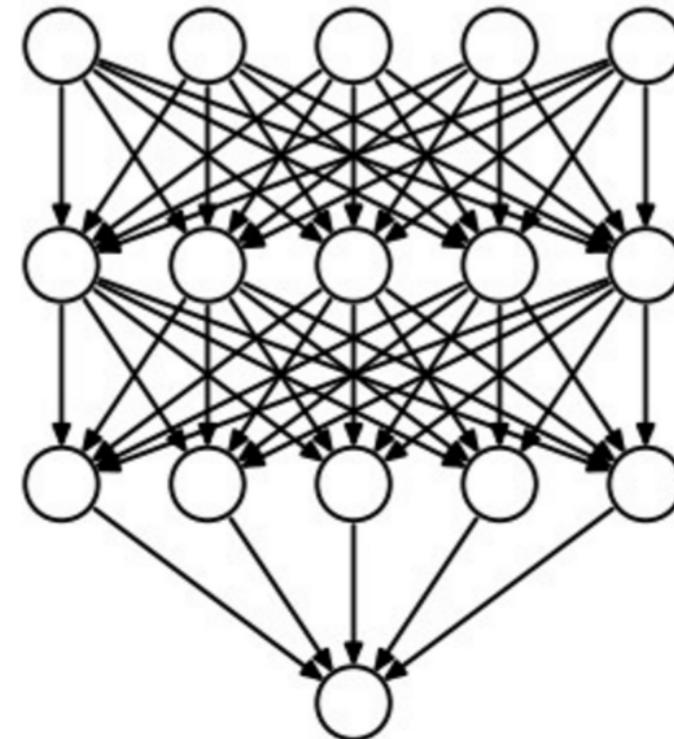
- When validation error starts increasing
- Protects against overfitting



## DROPOUT



Training  
 $p_{keep} = 0.50$



Test  
 $p_{keep} = 1.0$

# CONCLUSION

We have learned:

- What a (deep) neural network is
- Gradient descent and the different optimizers
- Learning rate decay
- Some problems of a deep neural network and how to solve them



**THANK YOU  
FOR YOUR  
ATTENTION**

Bas Huijben

**Pedro de Medinalaan 11,  
1086 XK Amsterdam**



020 - 773 1972



[www.anchormen.nl](http://www.anchormen.nl)

