Northeastern

CS 7180:

SPECIAL TOPICS IN AI

SOME RECENT HEADLINES

Stephen Hawking warns artificial intelligence could end mankind

By Rory Cellan-Jones
Technology correspondent

2 December 2014

› Technology Intelligence

AI is the biggest risk we face as a civilisation, Elon Musk says

share

Save 4

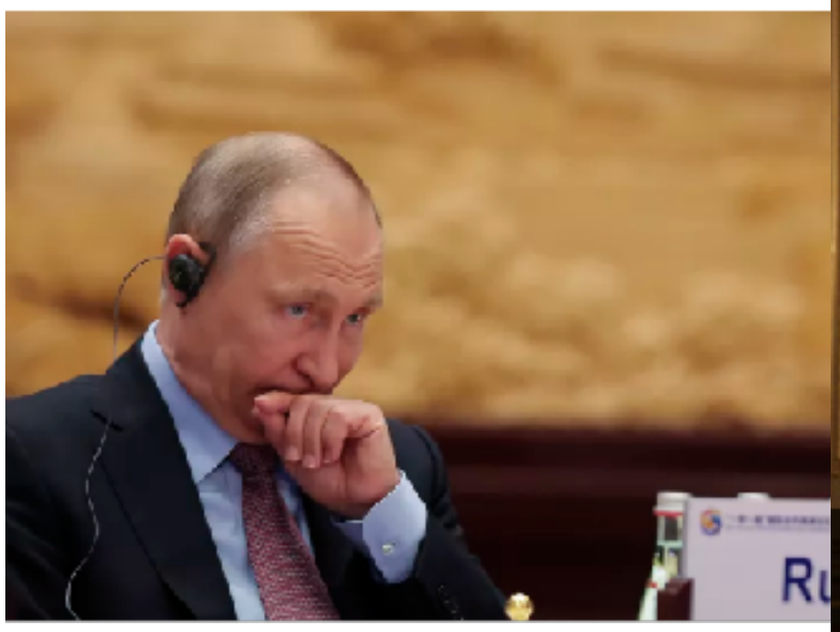
Musk was speaking to US governors. Credit: AP

SOME RECENT HEADLINES

Putin says the nation that leads in AI 'will be the ruler of the world'

The Russian president warned that artificial intelligence offers 'colossal opportunities' as well as dangers

By James Vincent | @jvincent | Sep 4, 2017, 4:53am EDT



French President Emmanuel Macron delivers a speech to French industrialists at the Elysee Palace in Paris | Yoan Valat/AFP via Getty Images

SOME HIGH VALUATIONS

Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Catherine Shu @catherineshu / 5 years ago

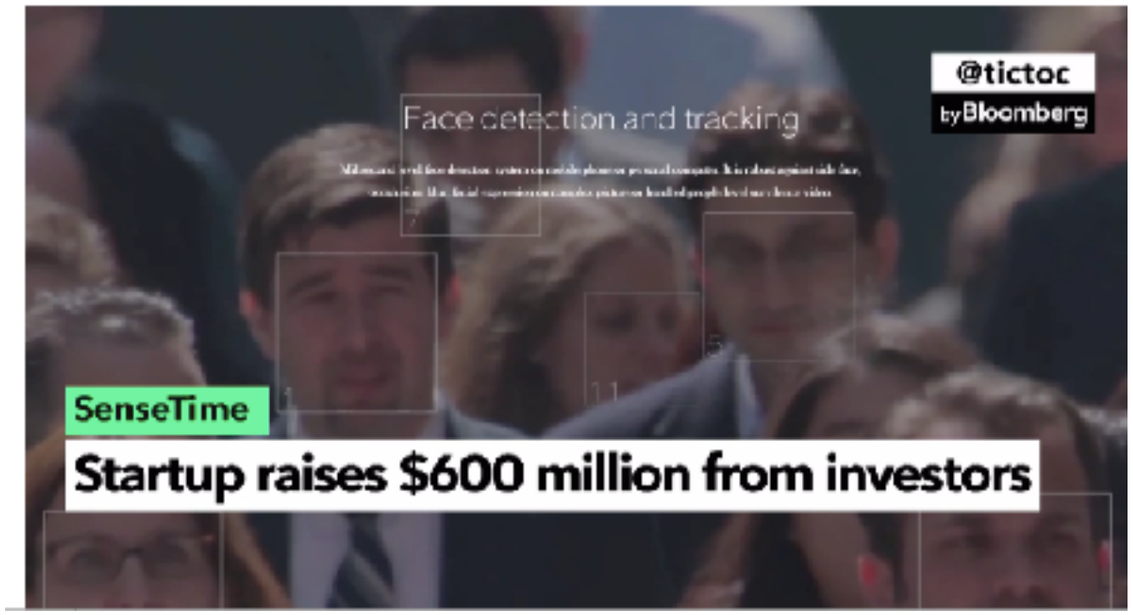
Comment

Technology

China Now Has the Most Valuable AI Startup in the World

Bloomberg News
April 8, 2018, 6:00 PM PDT

- It becomes the world's richest-valued private AI startup
- The company drives China's ambition to dominate global AI



SOME HIGH VALUATIONS

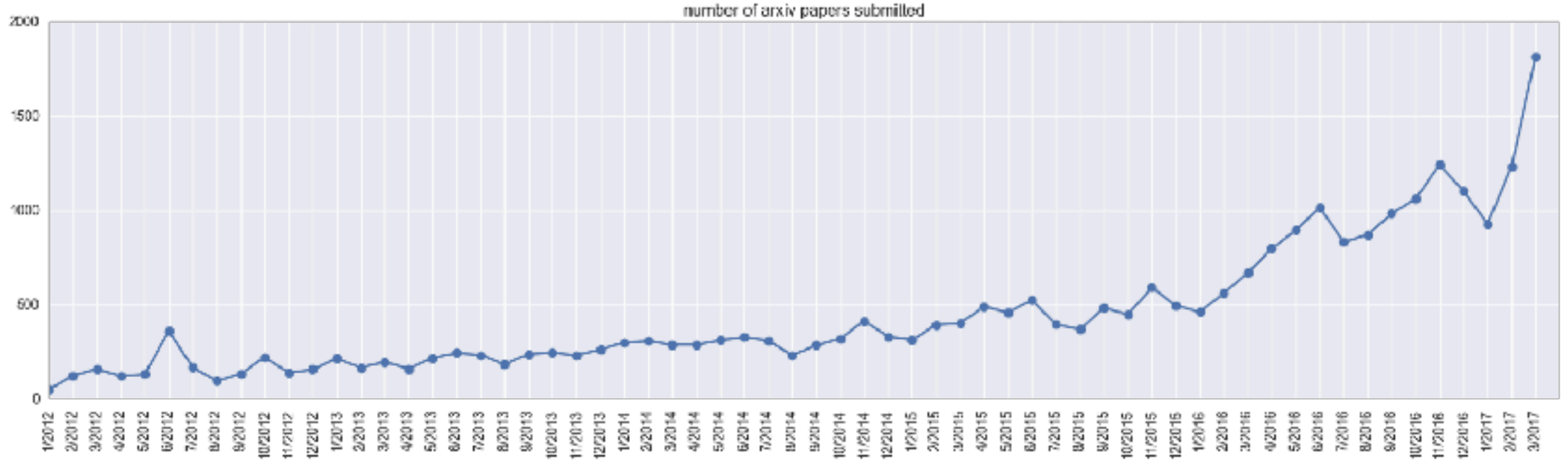
NIPS Tickets Sell Out in Less Than 12 Minutes

Synced · Follow
Sep 4, 2018 · 2 min read

The New York Times

Tech Giants Are Paying Huge Salaries for Scarce A.I. Talent

Nearly all big tech companies have an artificial intelligence project, and they are willing to pay experts millions of dollars to help get it done.



HYPE?

- Yes
- But also fundamental advances being made
- The most significant one is **Deep Learning**

ADMINISTRIVIA

COURSE BREAKDOWN

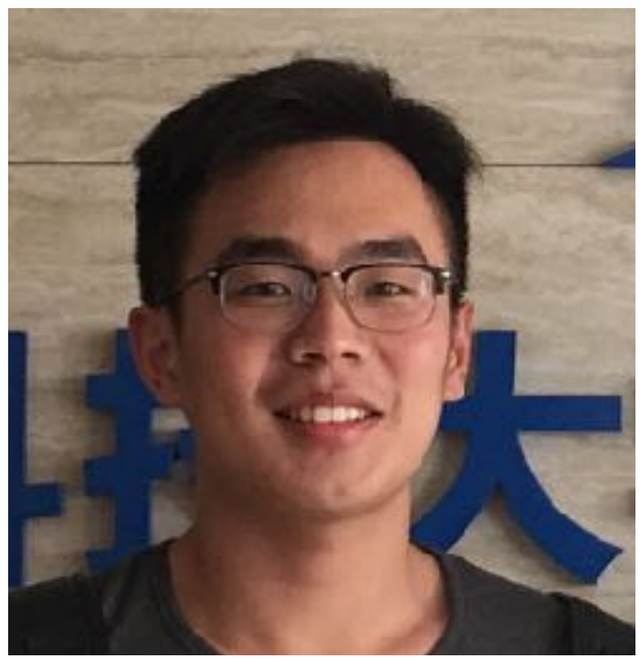
- 60 % Project
  - 5 % proposal, 10 % milestone 1, 15 % milestone 2,
  - 20 % report, 10 % poster presentation
    - Latex template
- 20 % Weekly Reading
  - Including hands-on practice
- 20 % Class Participation
  - 10% Presentation (20 min+5 Q&A)

COURSE ETIQUETTE

- Class participation
  - Show up
  - Ask questions
- Academic Integrity
  - Do NOT copy others' solutions
  - Cite properly

LOGISTICS

Lecturer: **Rose Yu**



TA: **Rui Wang**

- Lecture: 11:45 am - 1:30 pm MR SL 011
- Office hour: 2:00 pm - 3:00 pm M SL 045 or By Appointment

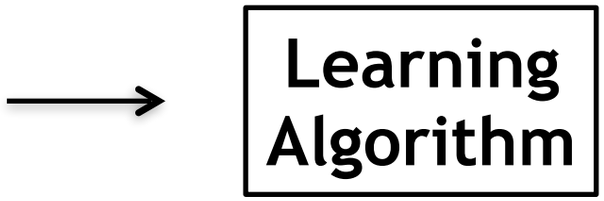
COURSE RESOURCES

- Course Website:
  - https://sites.google.com/view/cs-7180-spring-2019/home
  - Papers, Reading materials, Writing templates
- Blackboard:
  - Assignment submission
- Piazza:
  - https://piazza.com/northeastern/spring2019/cs7180
  - Lecture slides, Announcements, Q&A

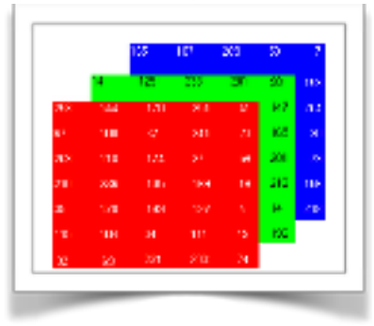
DEEP LEARNING

FEATURE ENGINEERING

- **Features**: representation of data



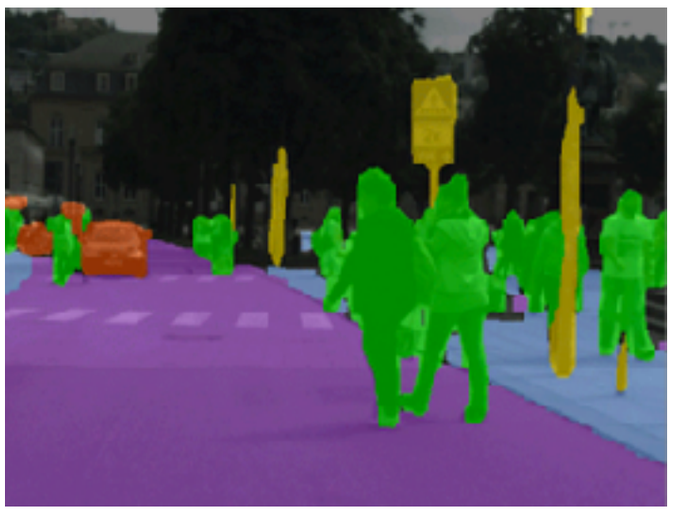
computer sees



- Feature Engineering
  - descriptors of predictive attributes
  - critical for good performance

WHY IS COMPUTER VISION HARD?

similarity measure

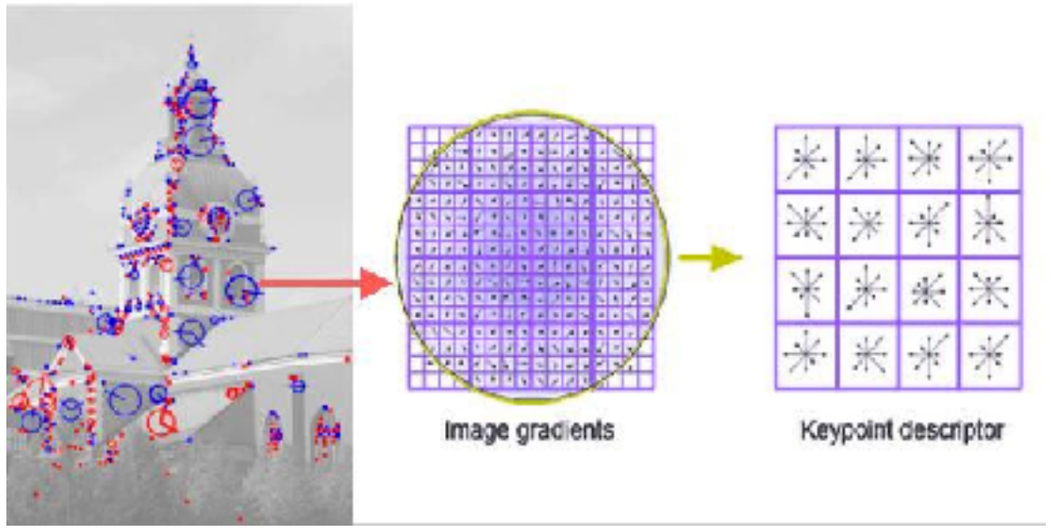


curse of dimensionality

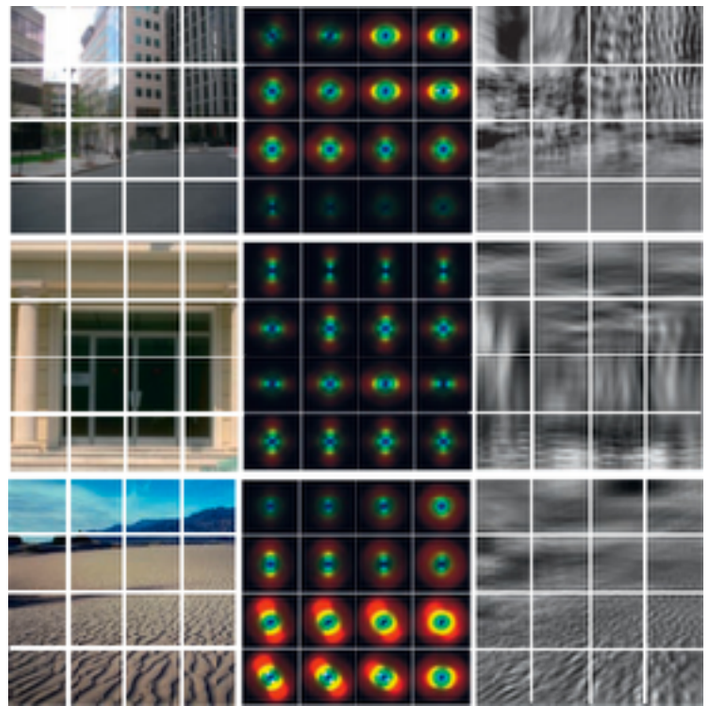
2D vs 3D

https://www.dailymail.co.uk/sciencetech/article-3145887/Google-apologises-Photos-app-tags-black-people-gorillas-Fault-image-recognition-software-mislabelled-picture.html

VISION FEATURES



SIFT

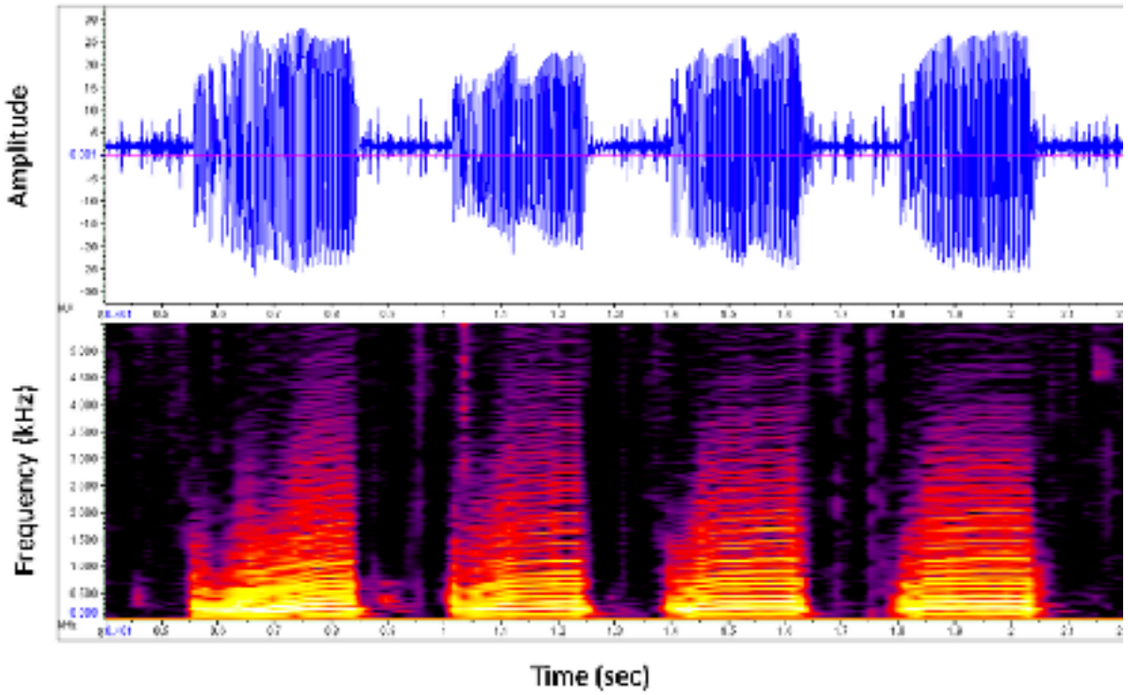


GIST

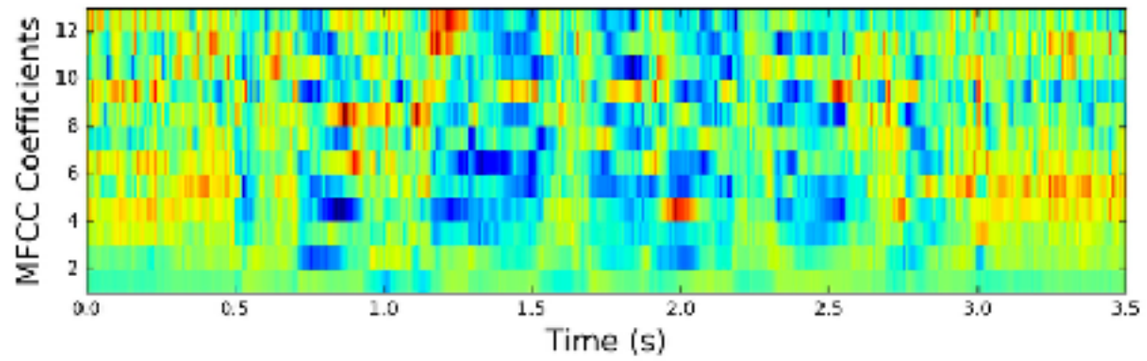
WHY IS SPEECH RECOGNITION HARD?

- Blind source separation
- Variability
- Phoneme boundaries

AUDIO FEATURES



Spectrogram



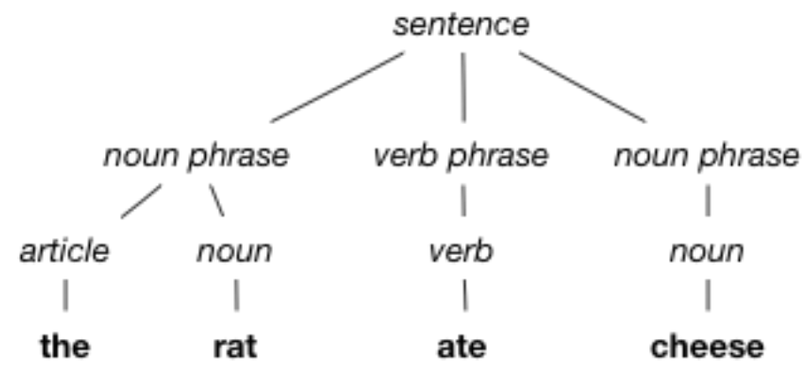
MFCC

TEXT FEATURES

the dog is on the table

0	0	1	1	0	1	1	1
are	cat	dog	is	now	on	table	the

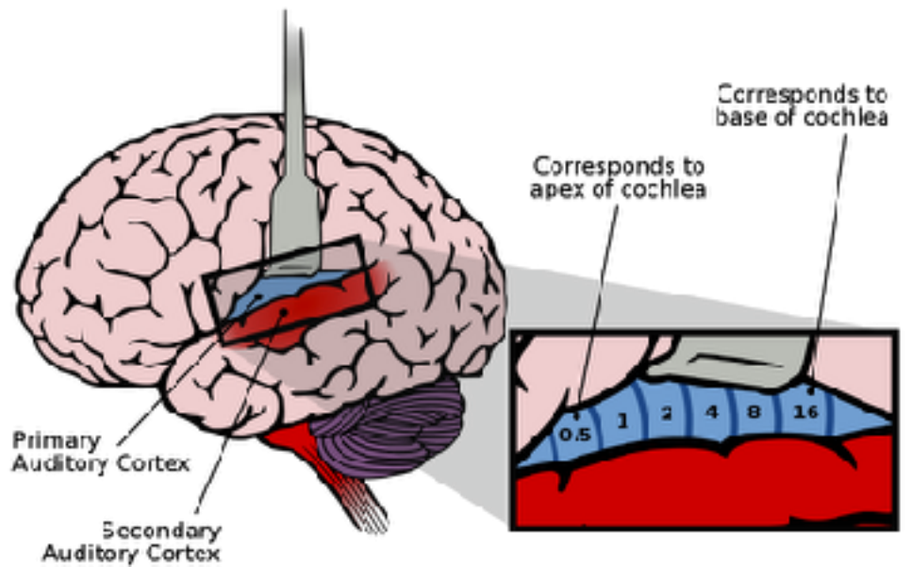
Bag of Words



Parse Tree

HUMAN BRAIN LEARNING

- Hypothesis: Human brain uses the same learning algorithm

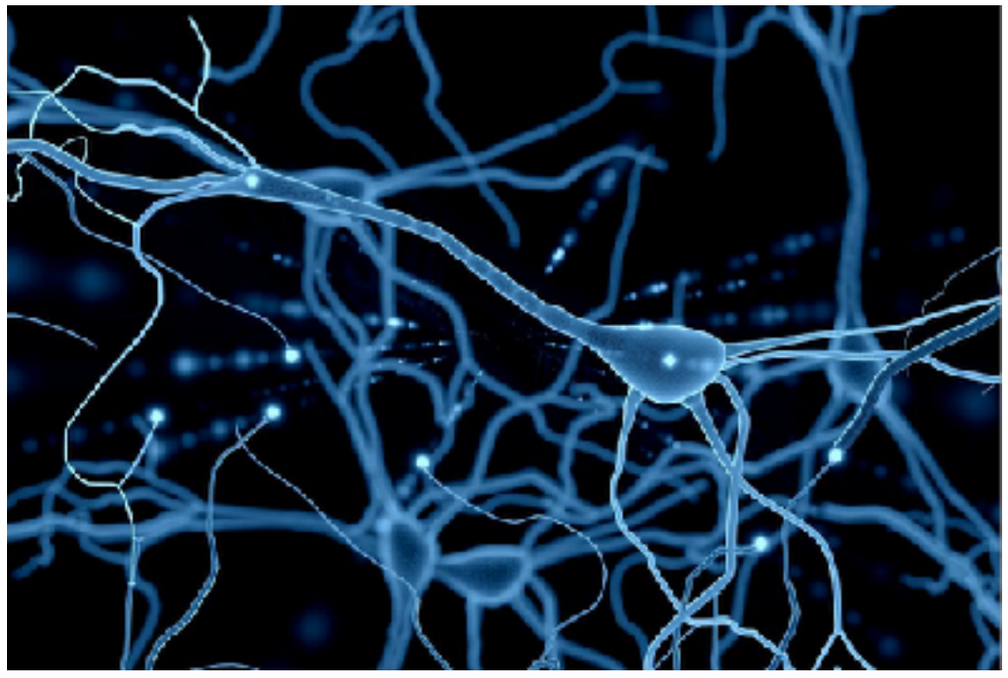


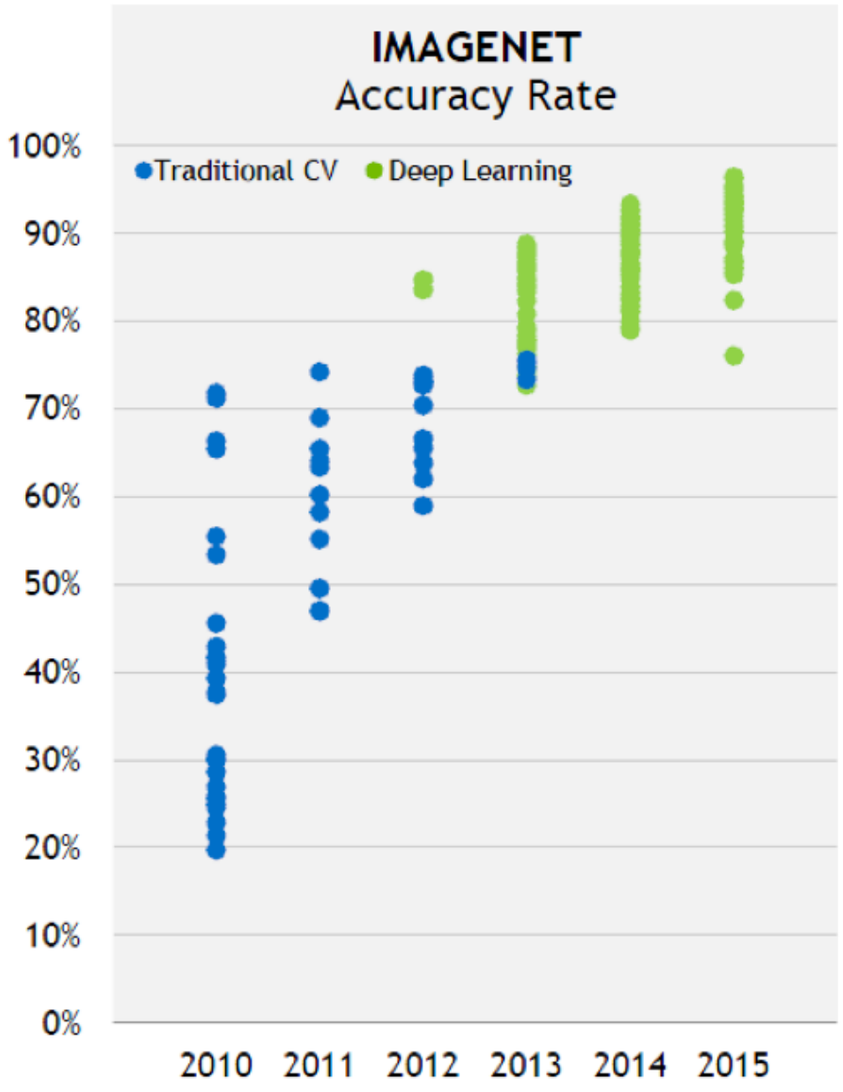
Auditory cortex learns to see

https://cs.stanford.edu/group/brain/index.html

MIMIC HUMAN BRAIN

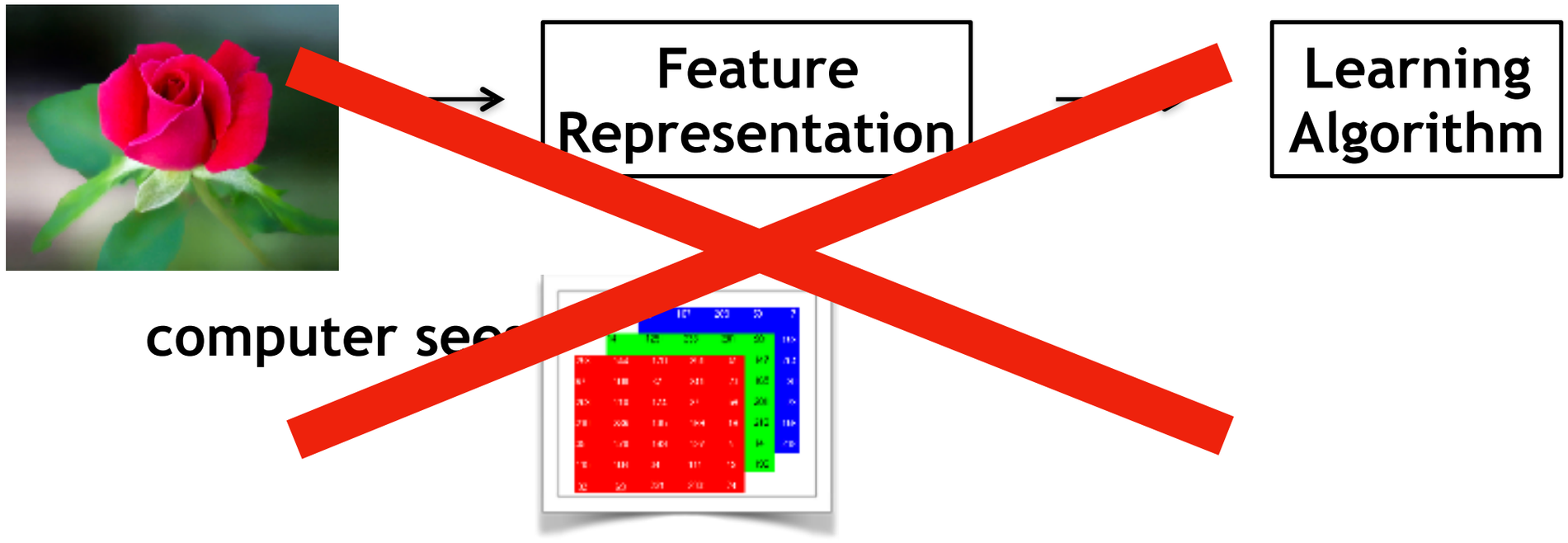
- Build learning algorithms that mimic the brain
- 100 billions neurons





FEATURE ENGINEERING

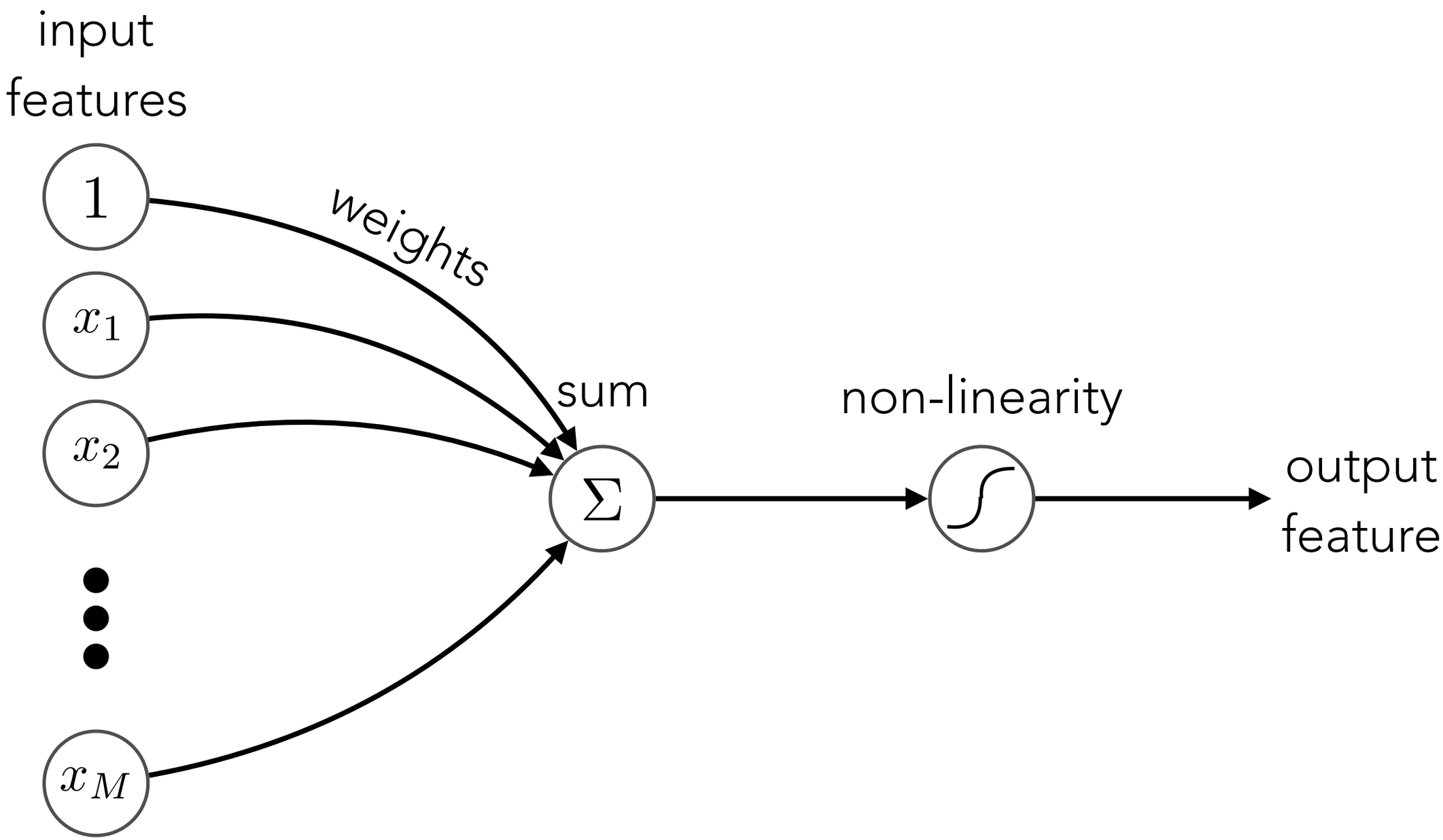
- Features: representation of data

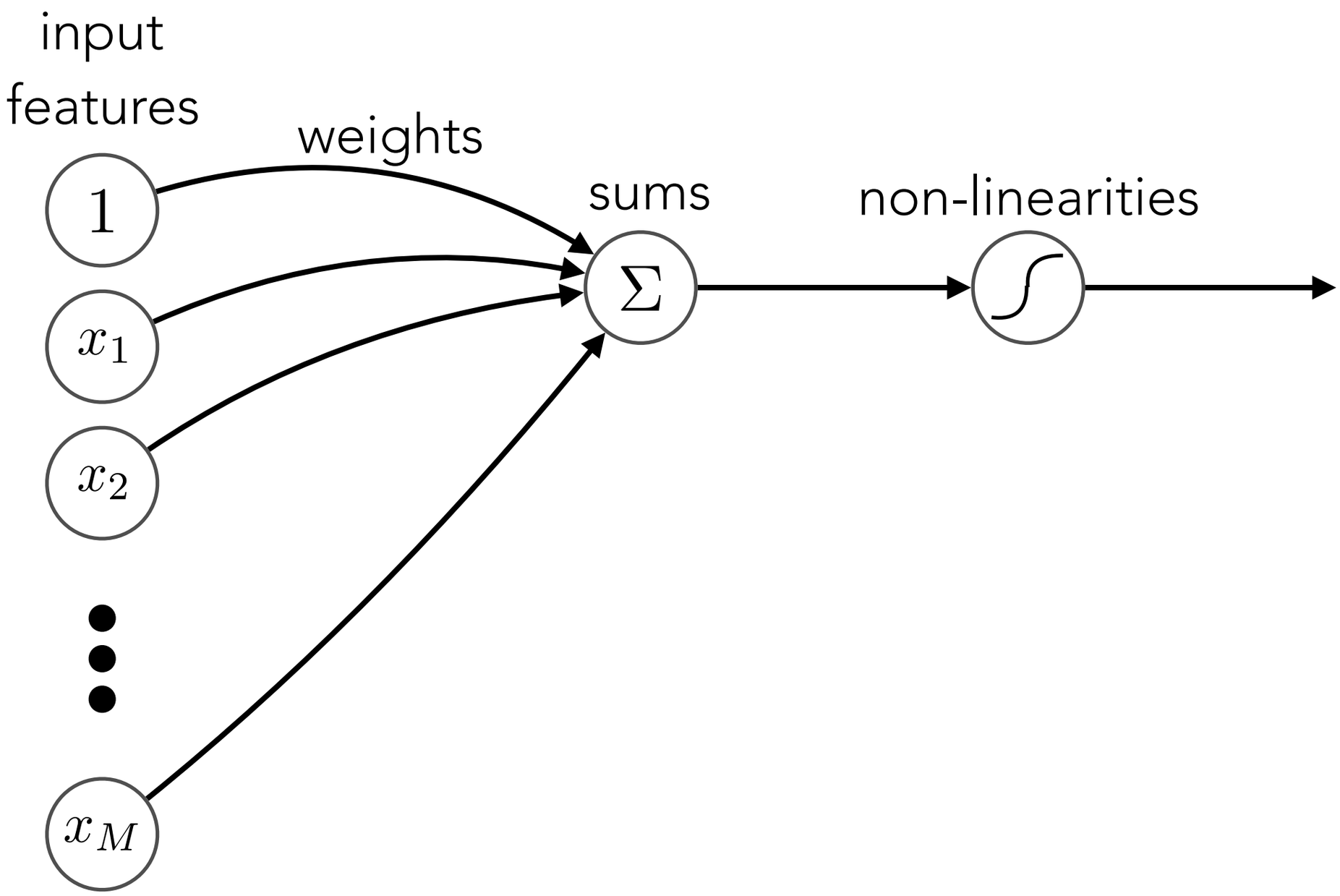


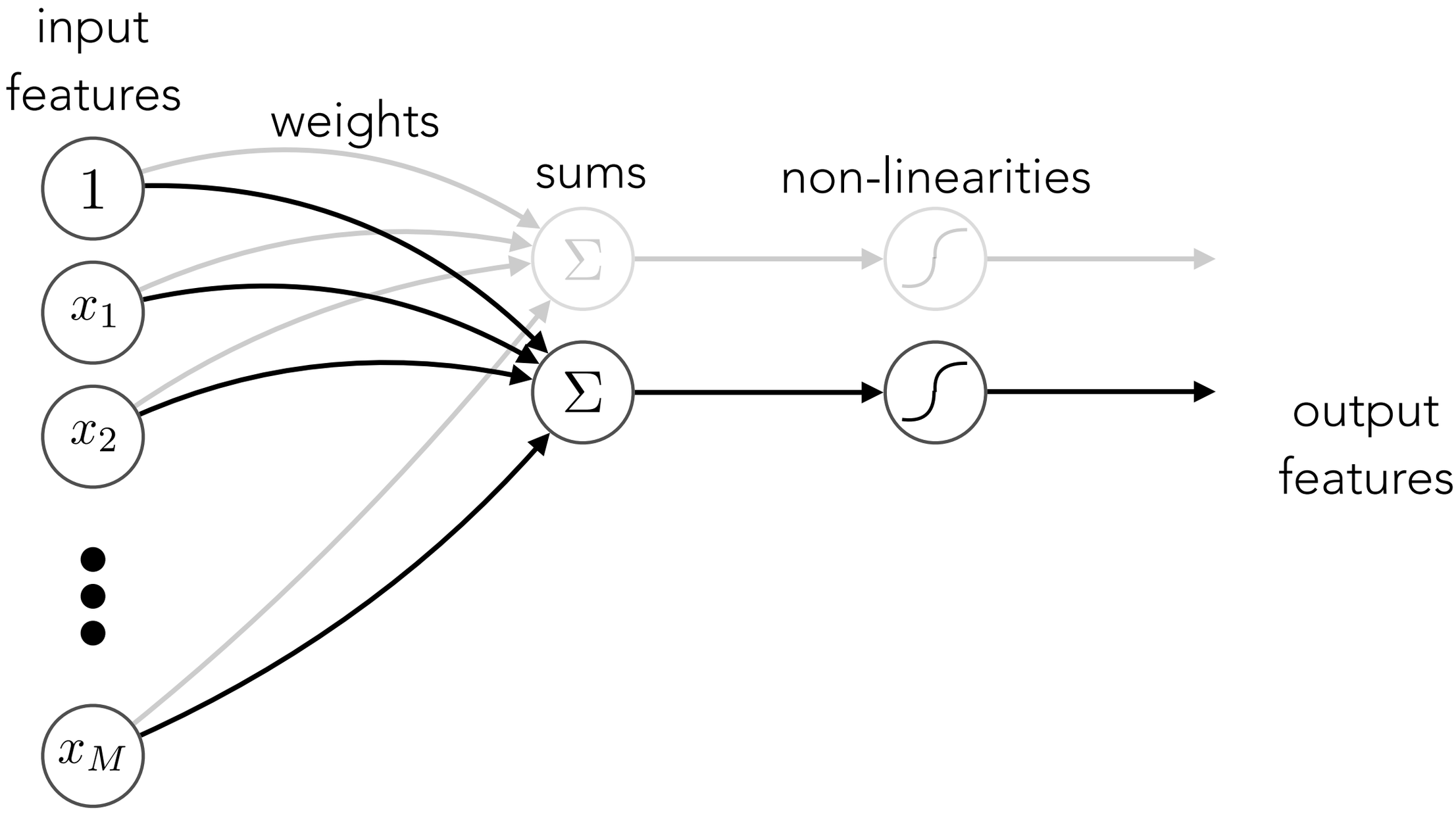
- Feature Engineering
  - descriptors of predictive attributes
  - critical for good performance

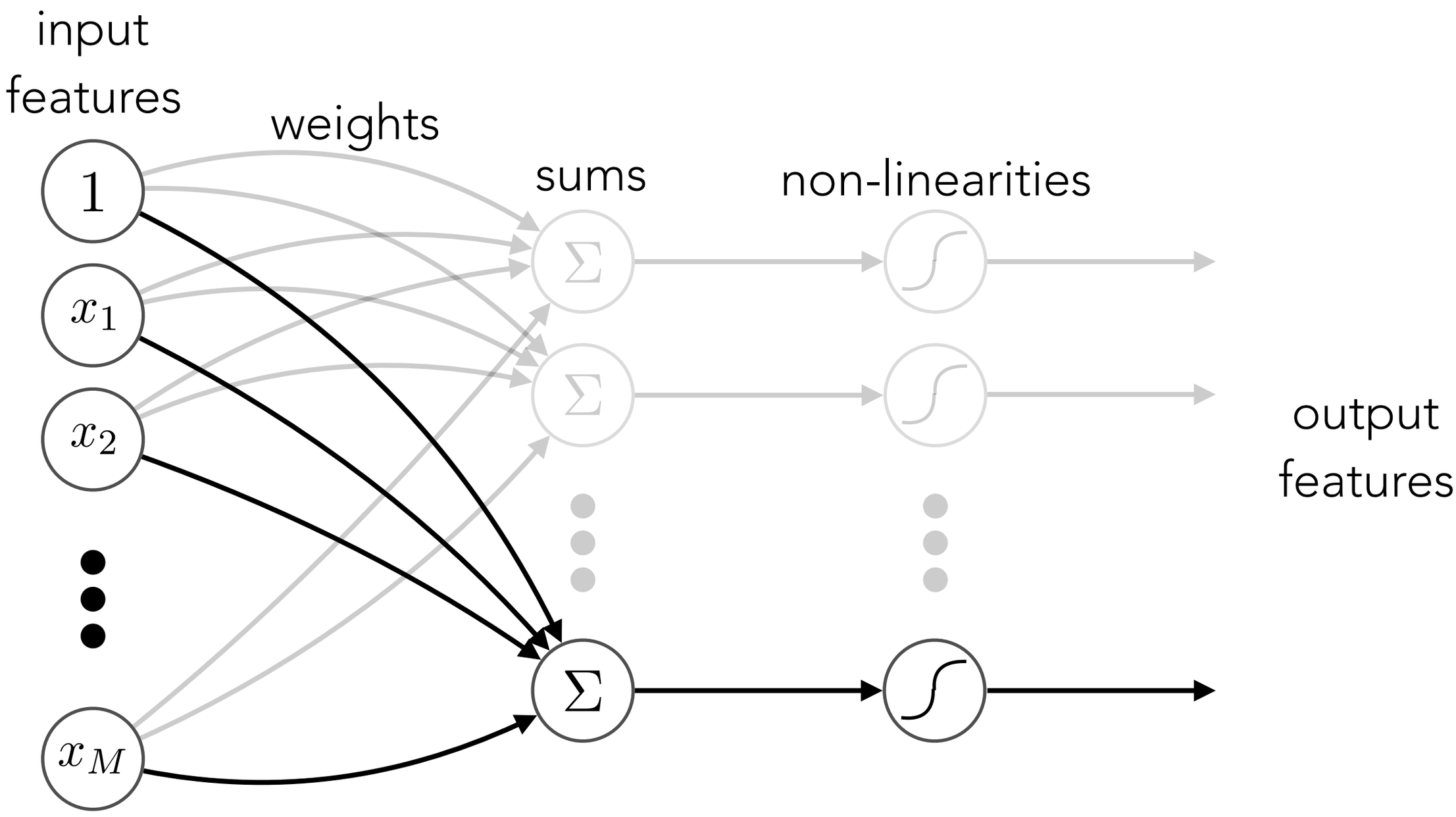
DEEP NEURAL NETWORKS

artificial **neuron**

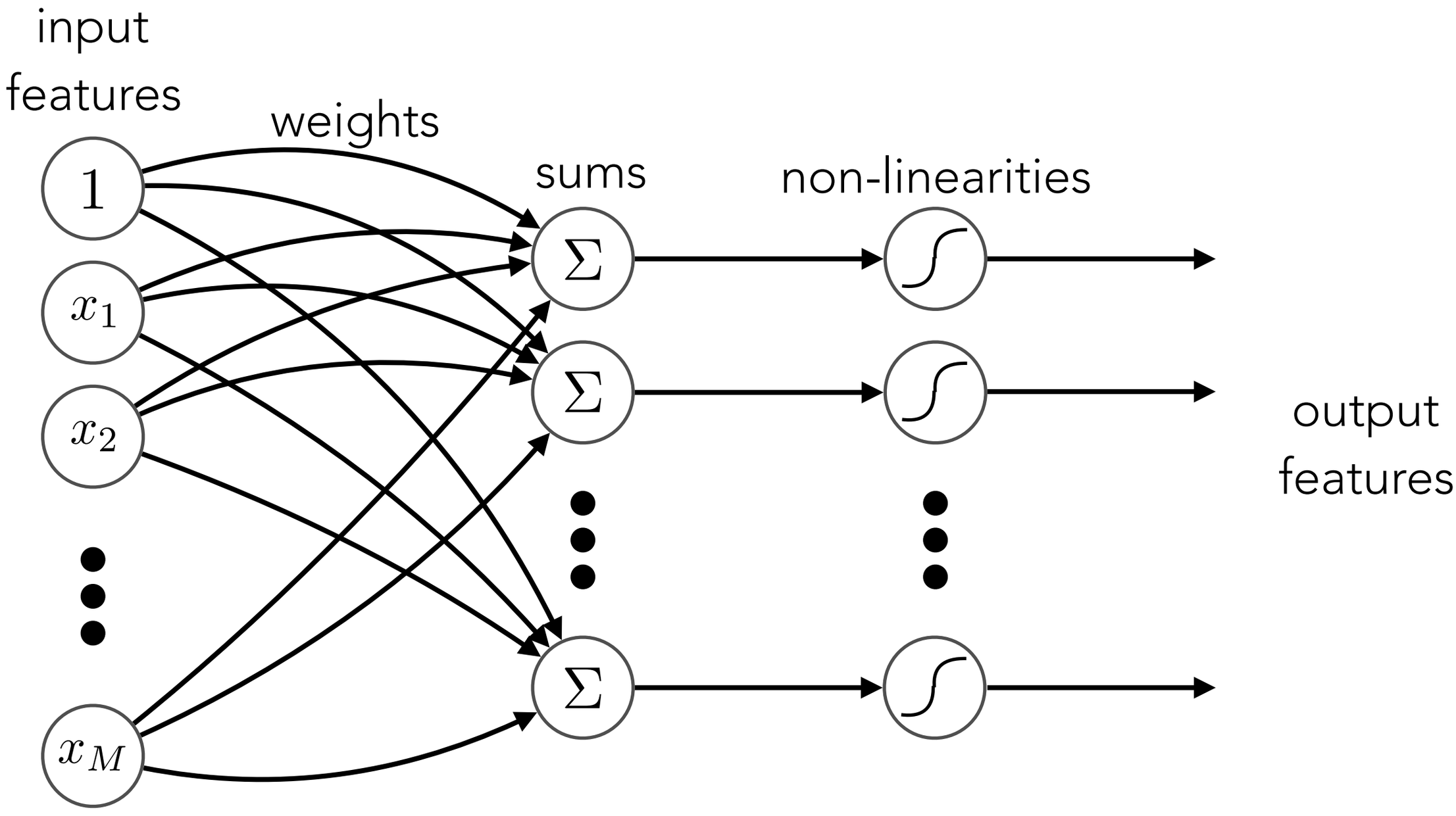


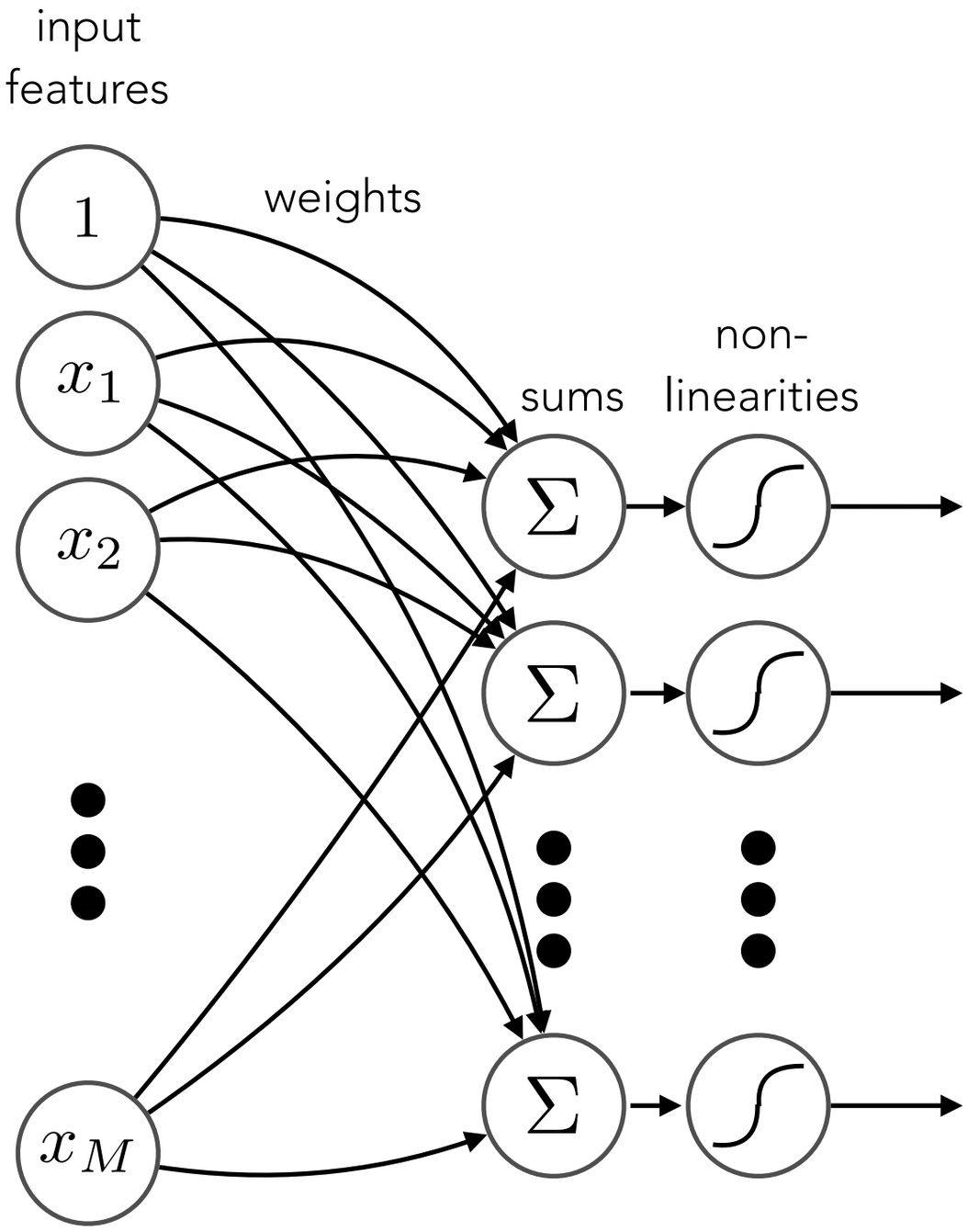

input
features
weights
sums
non-linearities
1
x_1
x_2
x_M
Σ

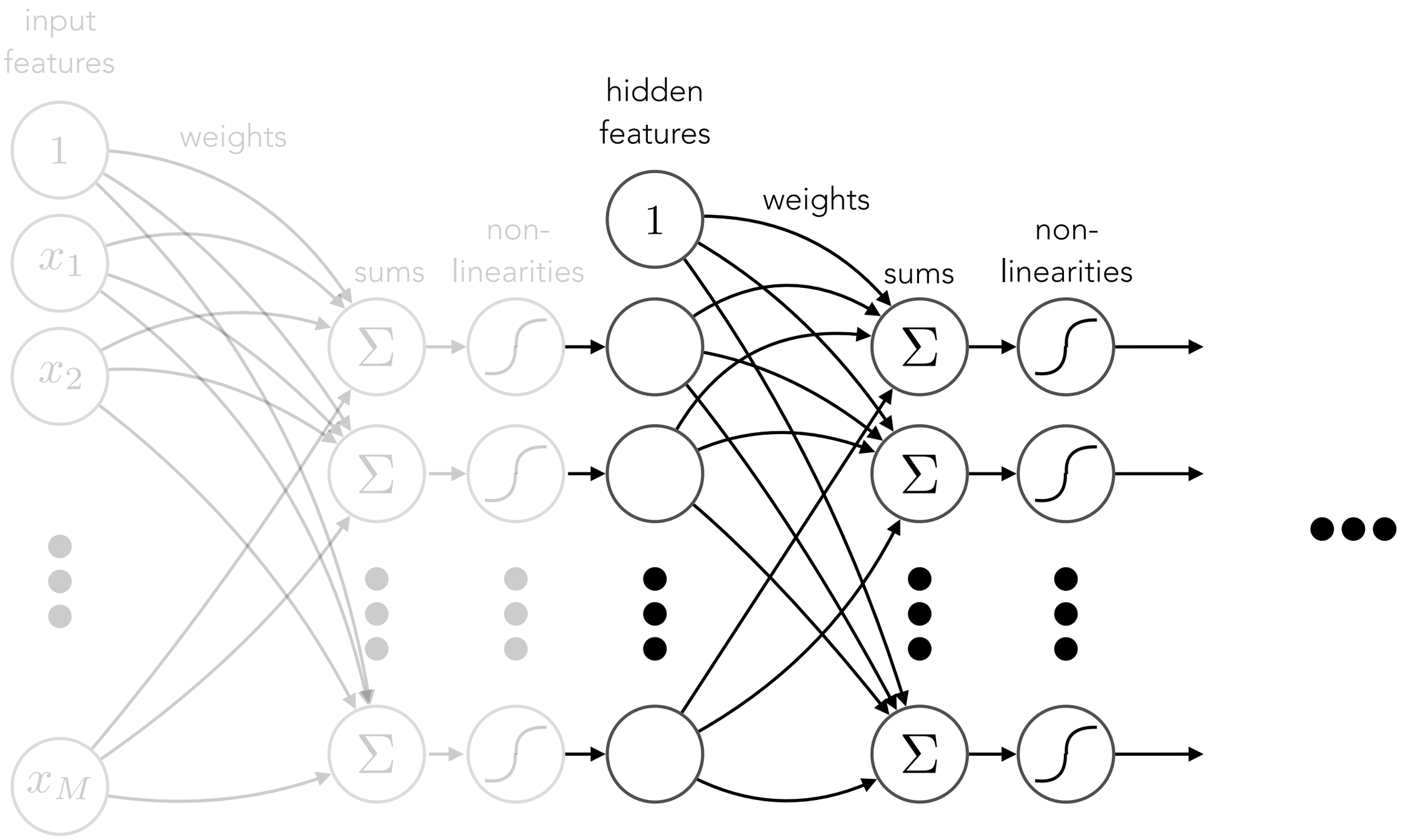

input
features
weights
sums
non-linearities
1
x_1
x_2
x_M
Σ
Σ
output
features


input
features
weights
sums
non-linearities
1
x_1
x_2
x_M
Σ
$\int$
output
features

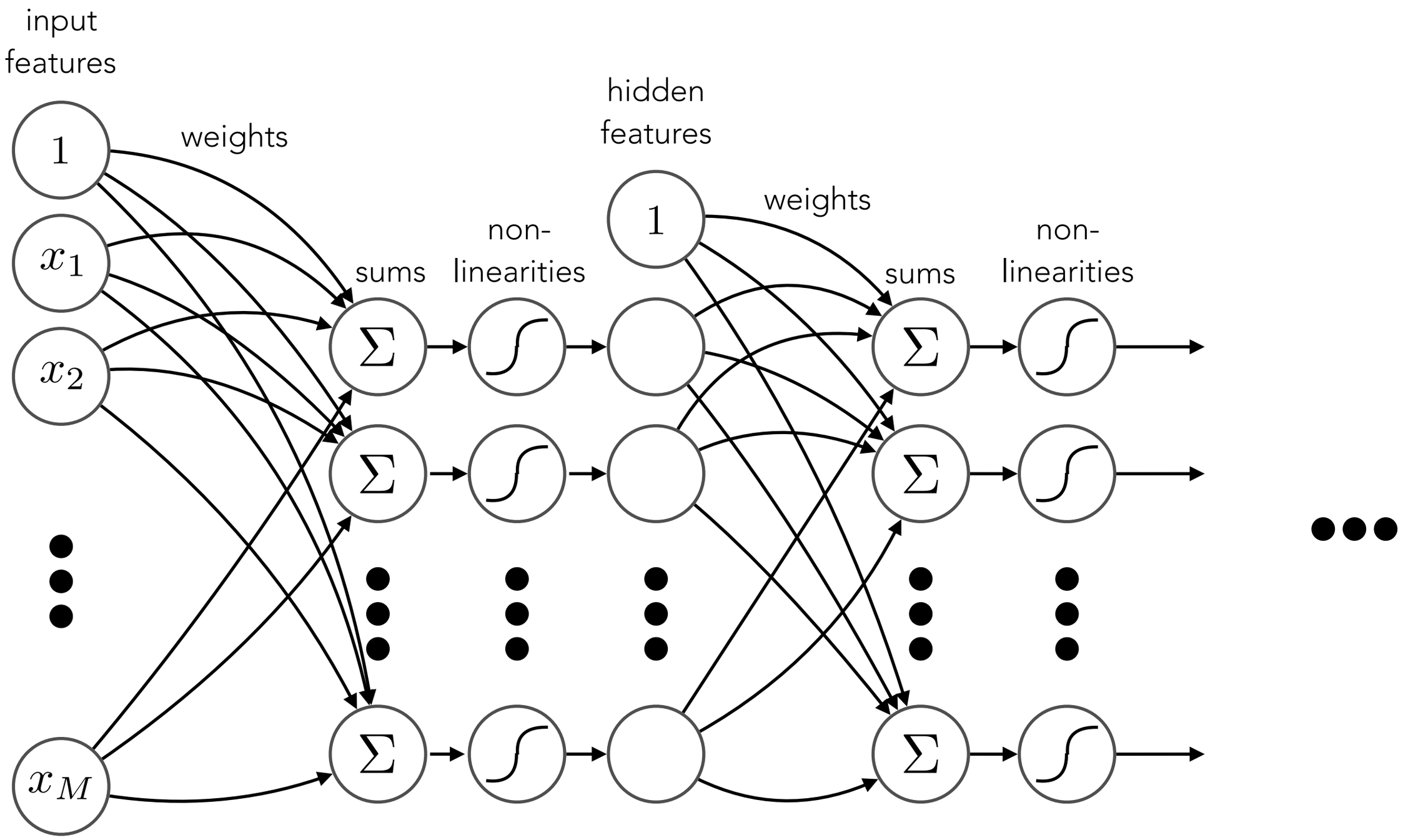
multiple neurons form a **layer**

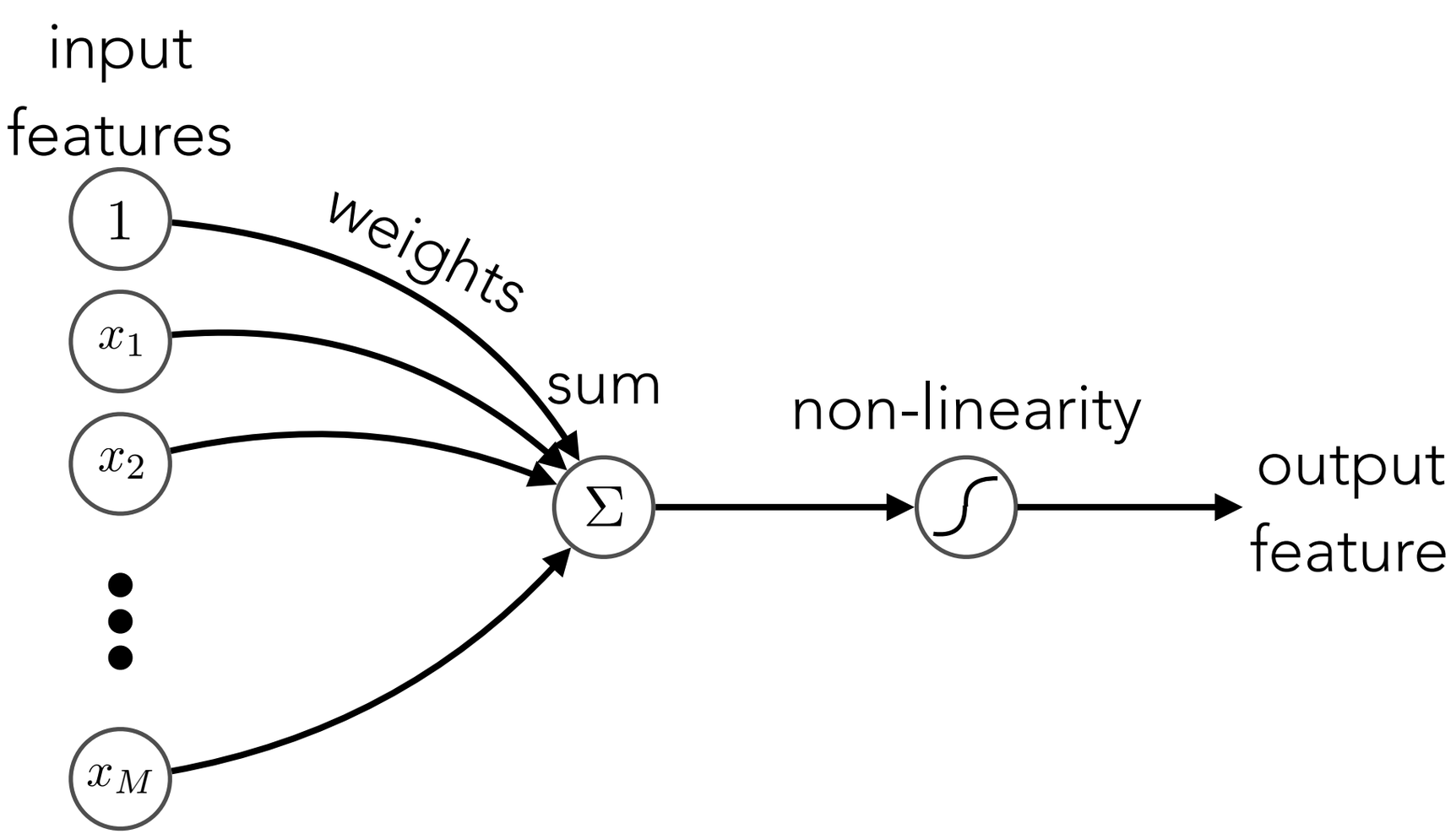



input
features
weights
non-
linearities
sums
1
x_1
x_2
x_M
Σ
Σ
Σ


input features
weights
sums
non-linearities
hidden features
weights
sums
non-linearities
1
x_1
x_2
x_M
Σ

multiple layers form a **network**





artificial neuron: *weighted sum and non-linearity*

bias

input features

$$s = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M = \mathbf{w}^\intercal \mathbf{x}$$

sum

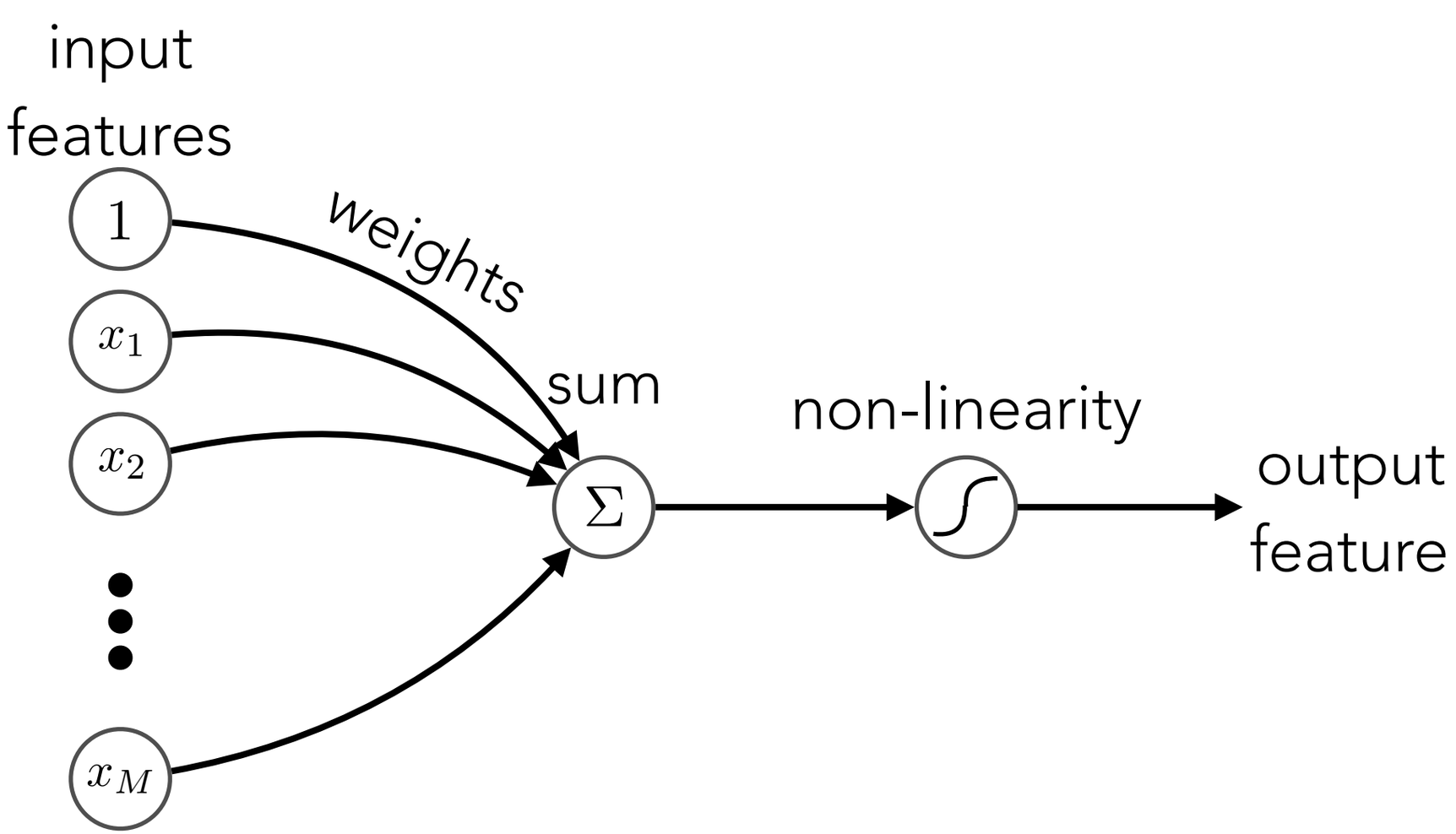
weights

$$h = \sigma(s)$$

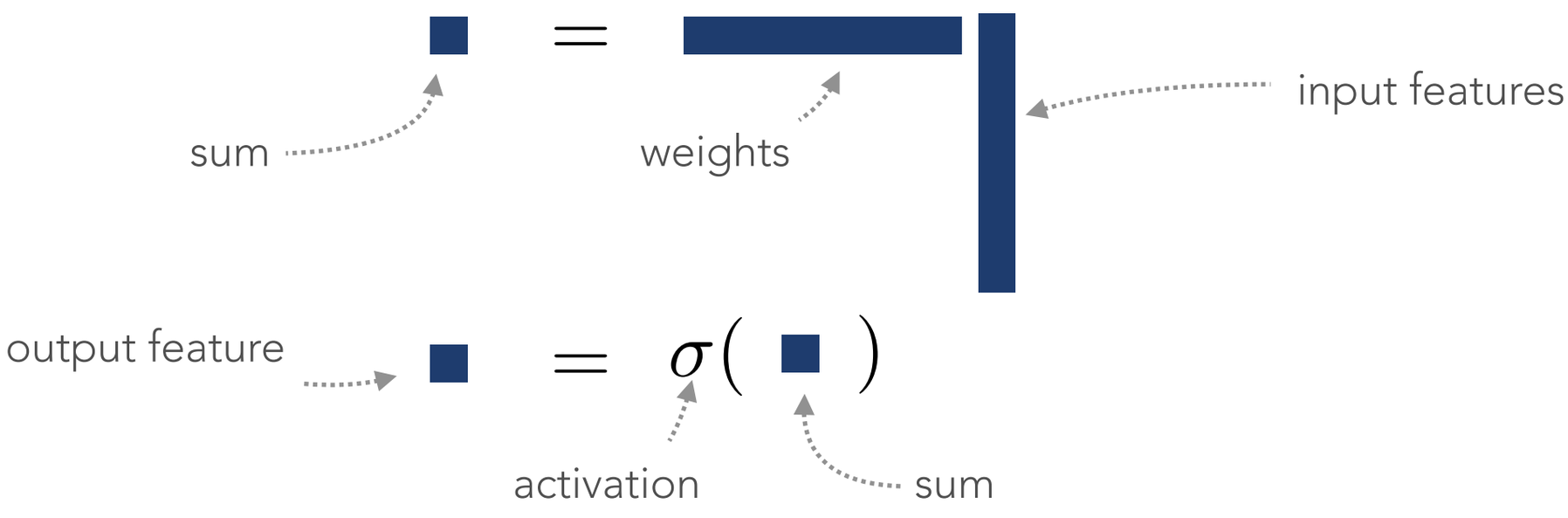
output feature

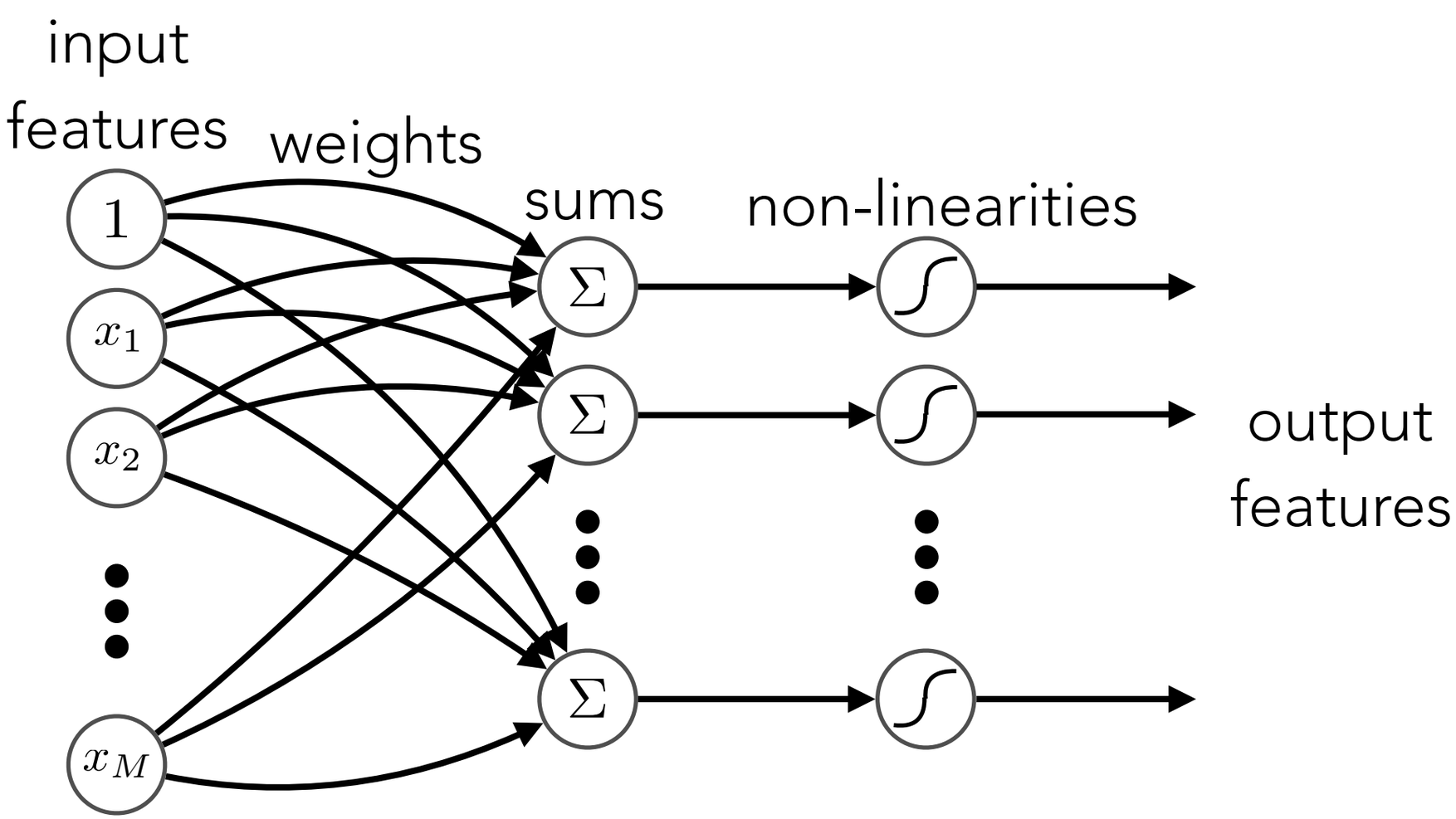
non-linearity

sum



artificial neuron: *weighted sum and non-linearity*

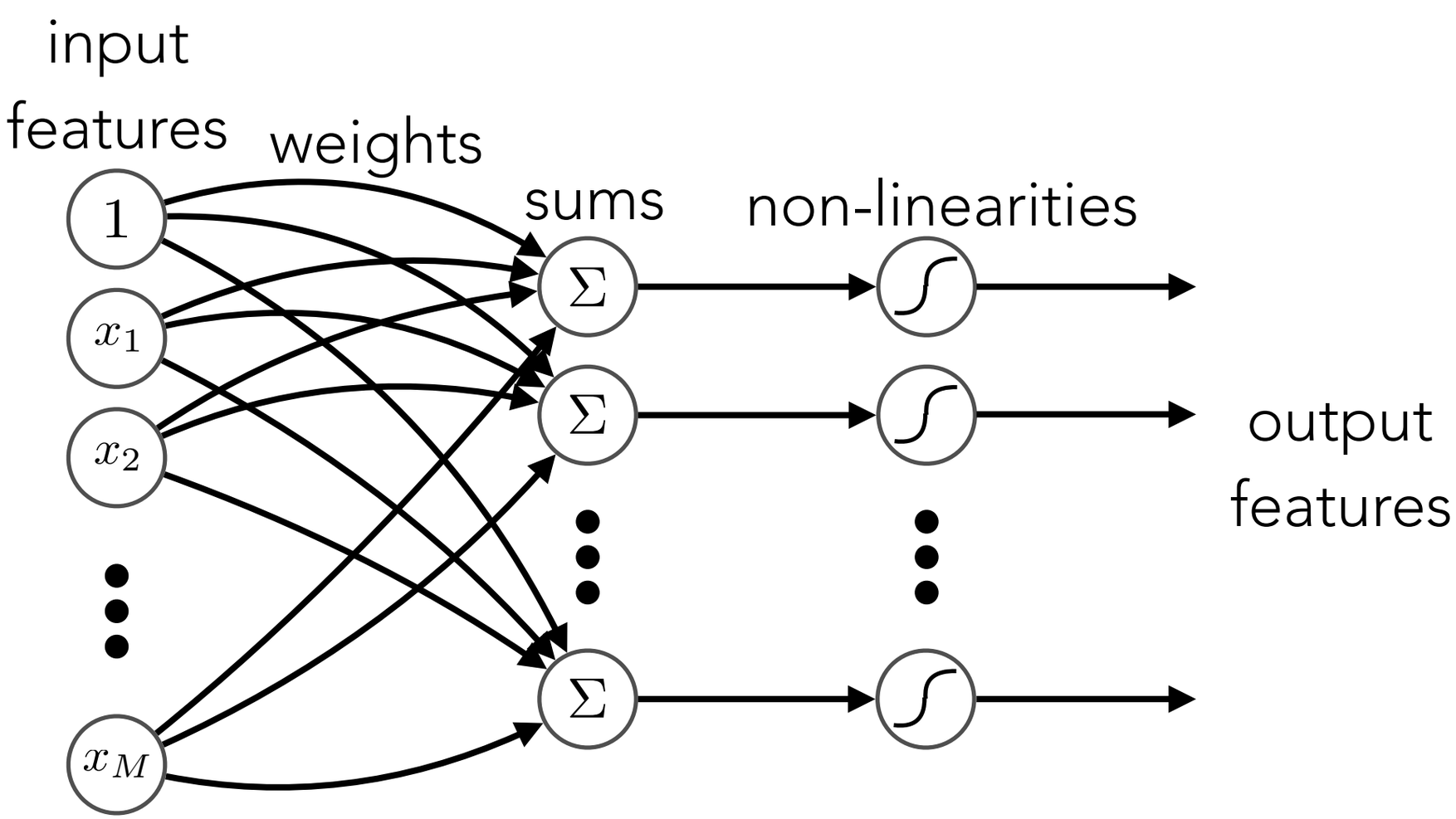




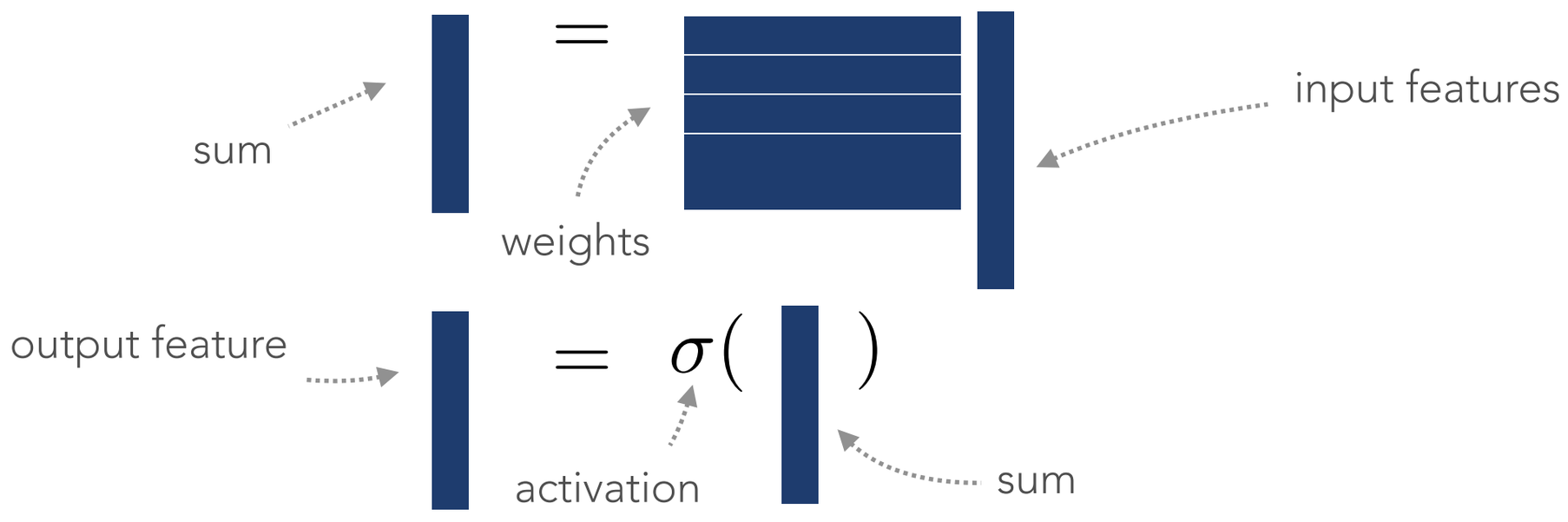
layer: *parallelized weighted sum and non-linearity*

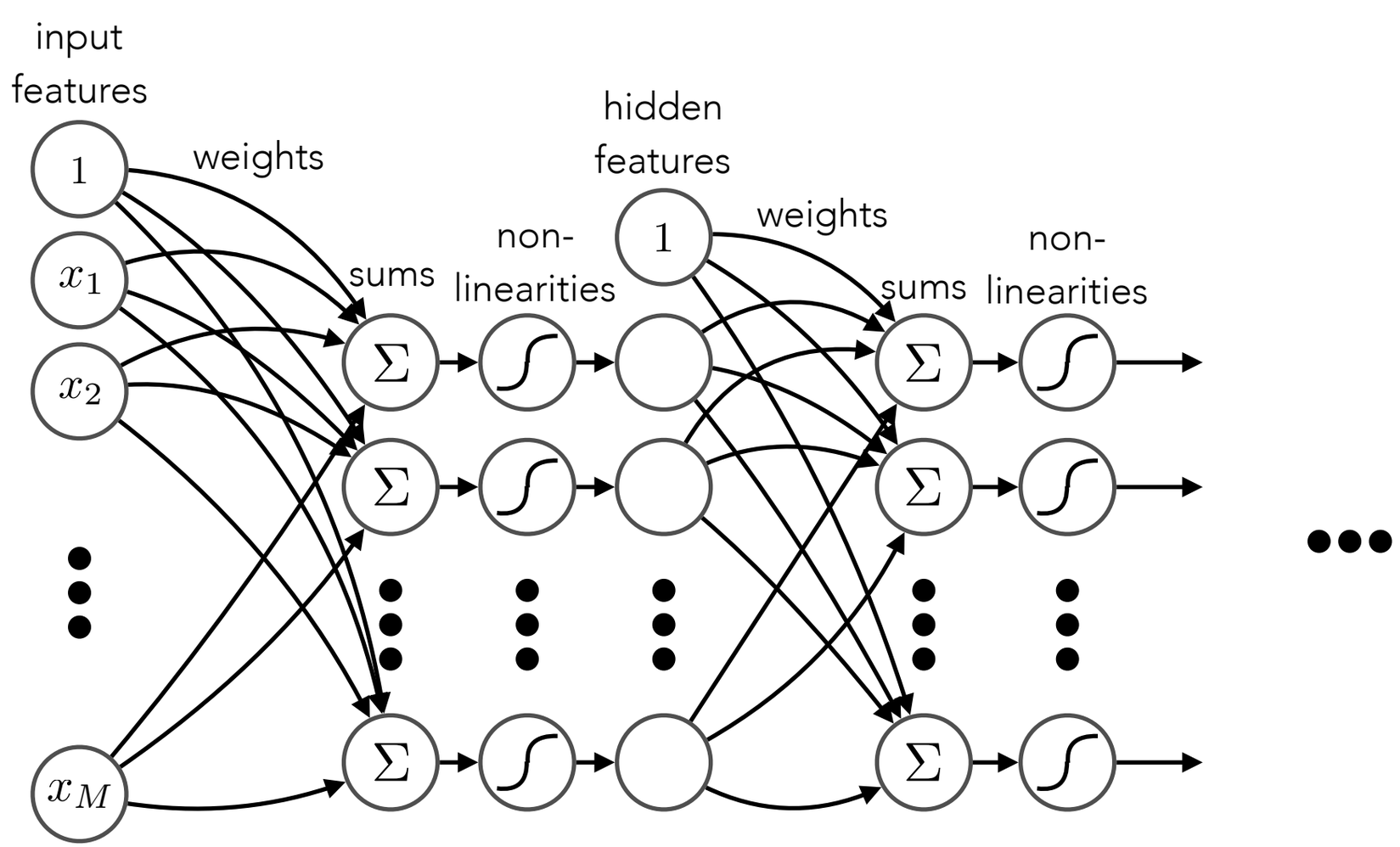
one sum per weight *vector* $s_j = \mathbf{w}_j^\intercal \mathbf{x} \longrightarrow \mathbf{s} = \mathbf{W}^\intercal \mathbf{x}$ vector of sums from weight *matrix*

$$\mathbf{h} = \sigma(\mathbf{s})$$



layer: *parallelized weighted sum and non-linearity*





network: *sequence of parallelized weighted sums and non-linearities*

DEFINE $\mathbf{x}^{(0)} \equiv \mathbf{x}$, $\mathbf{x}^{(1)} \equiv \mathbf{h}$, ETC.

1st layer

$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$$

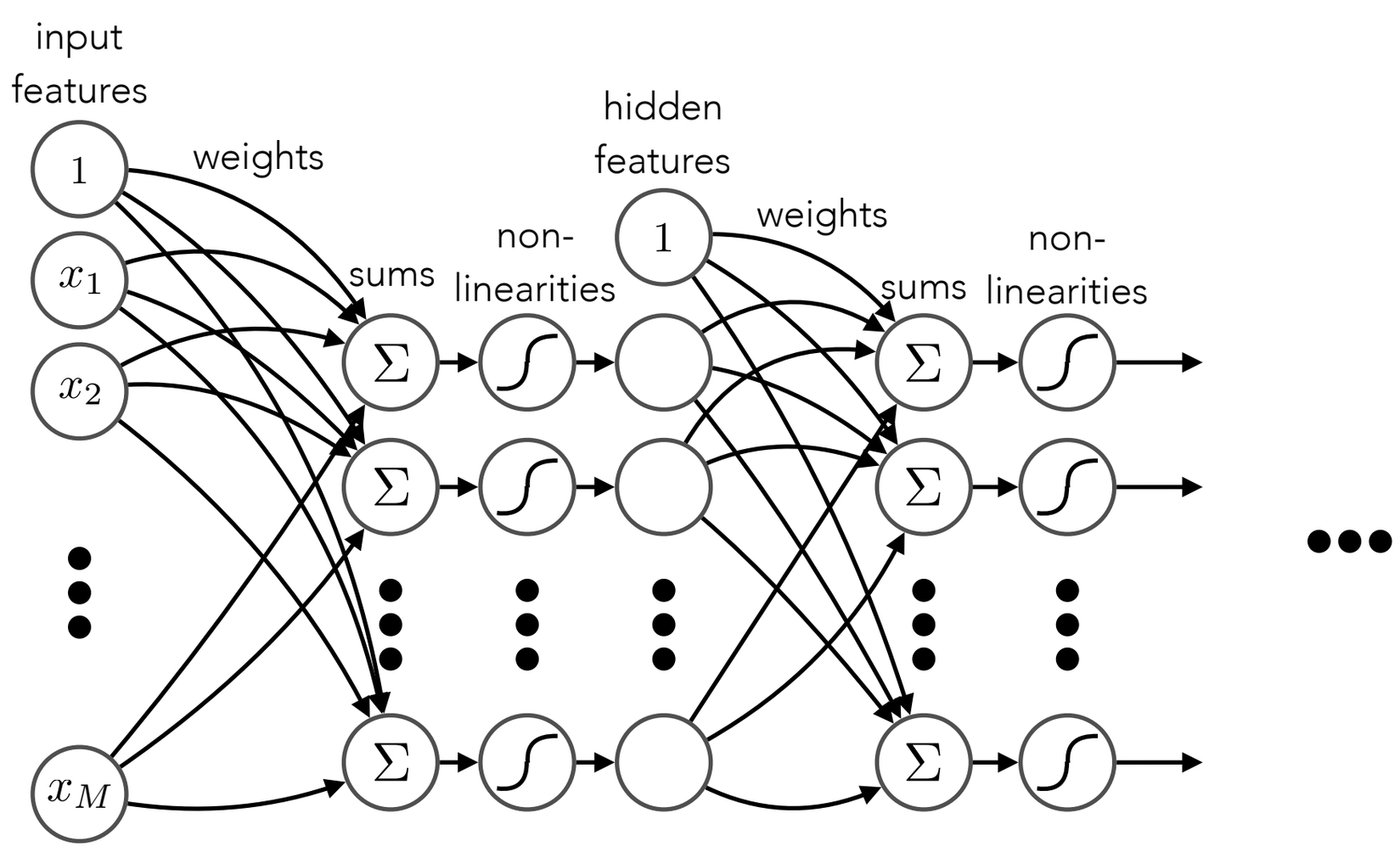
$$\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$$

2nd layer

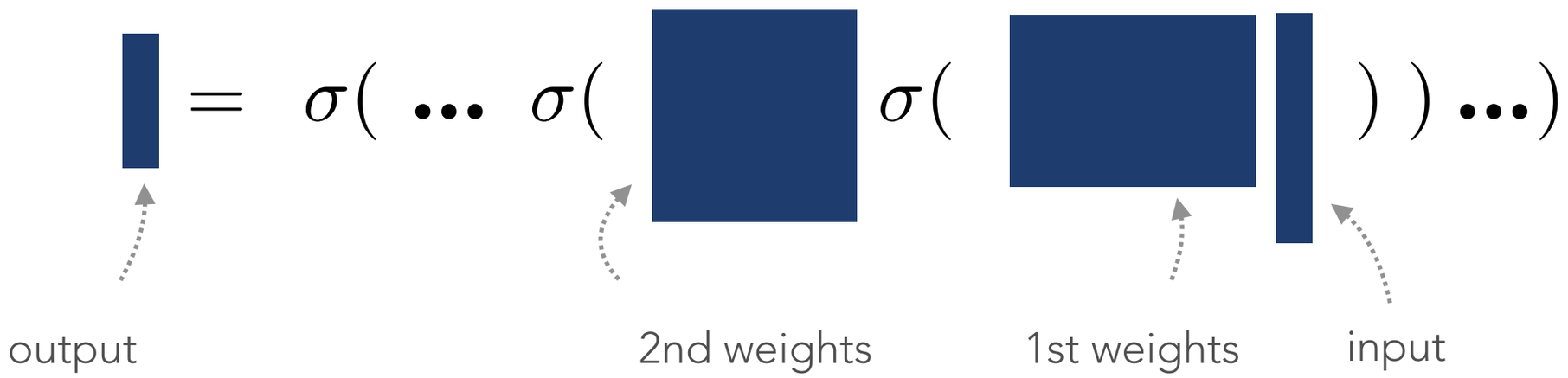
$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$$

$$\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$$

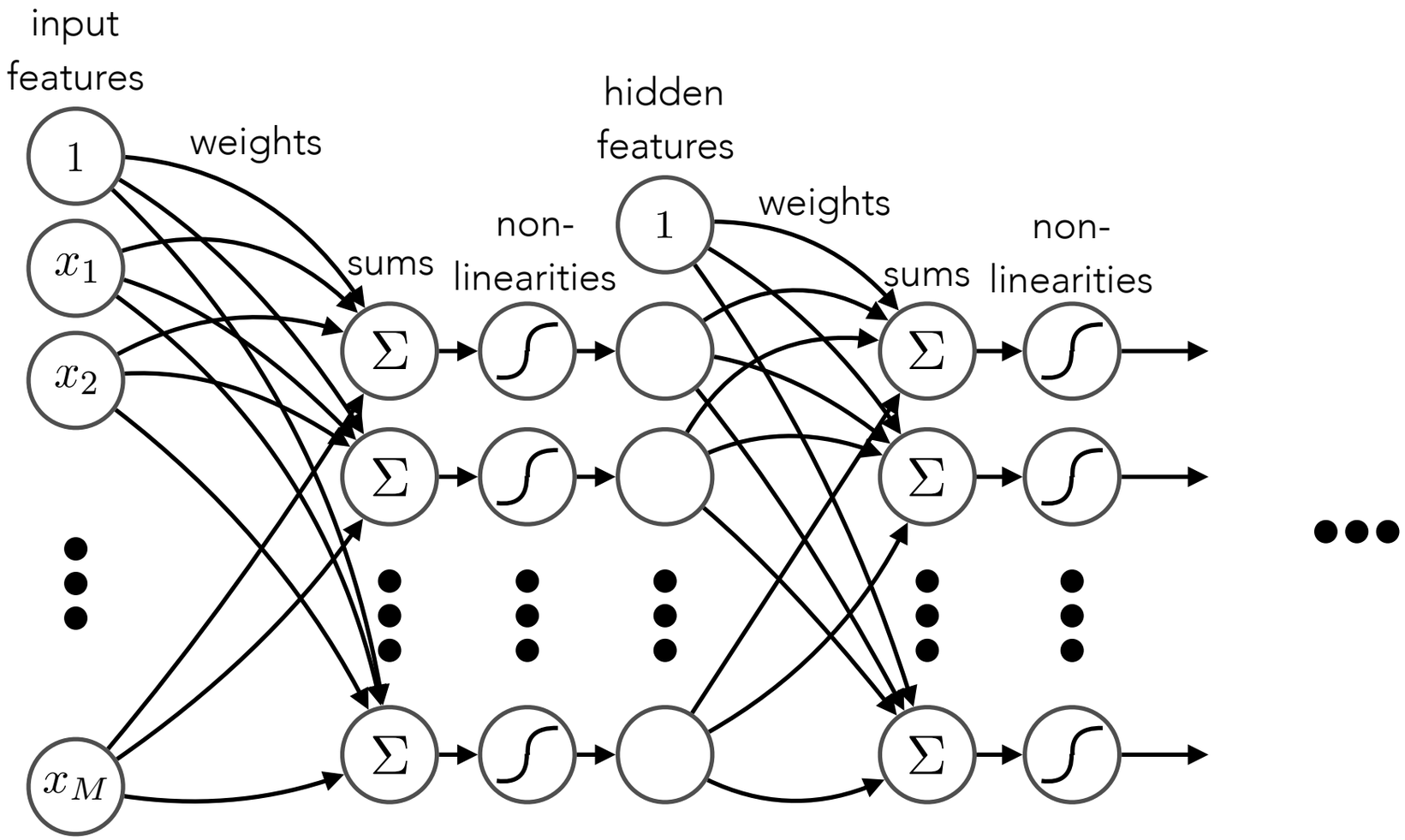
•••



network: *sequence of parallelized weighted sums and non-linearities*



recapitulation



we have a method for building ***expressive*** non-linear functions

deep networks are *universal function approximators* *(Hornik, 1991)*

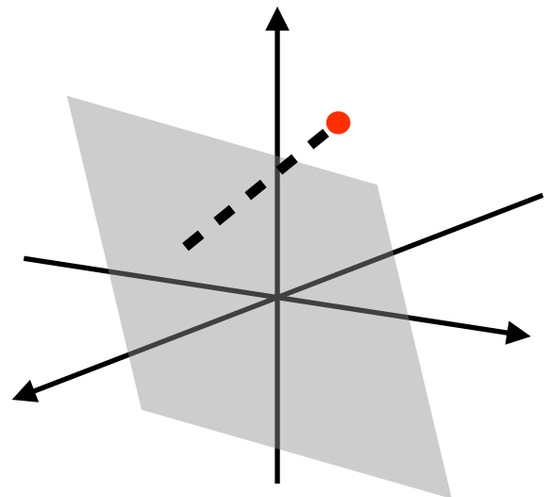
→ with enough units & layers, can approximate *any* function

reinterpretation

the dot product is the shortest distance between a point and a plane

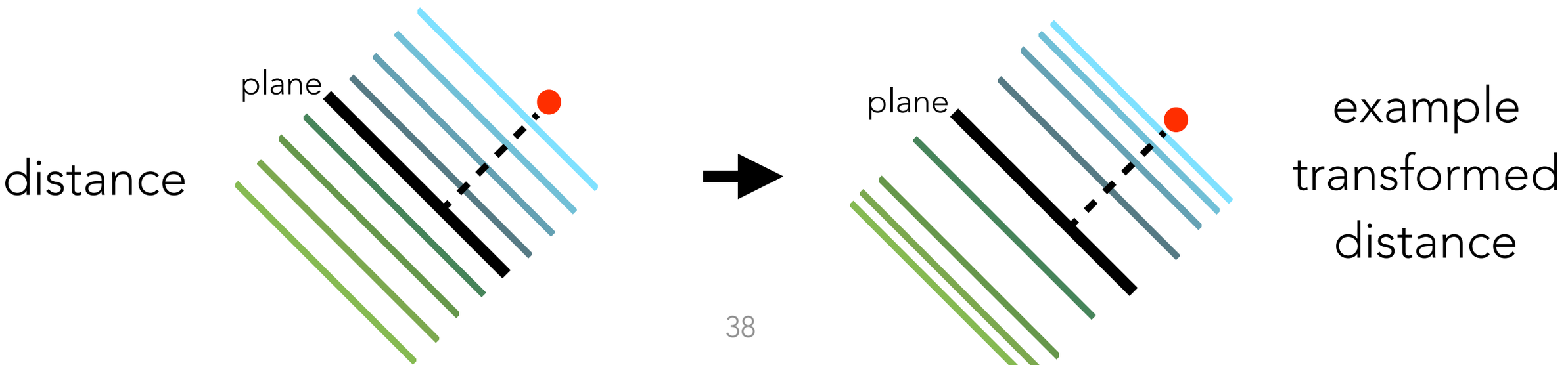
each artificial neuron defines a (hyper)plane:

$$0 = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_M x_M$$



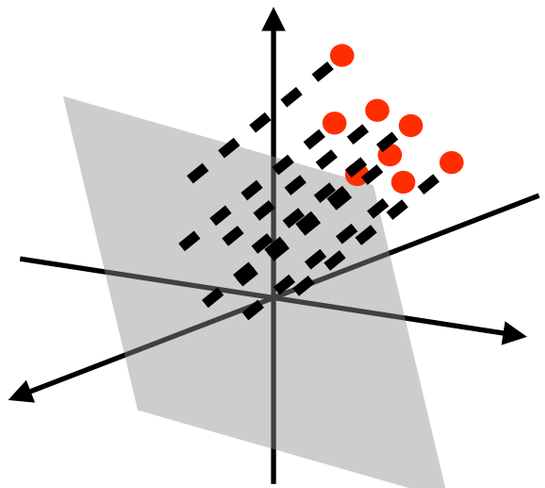
<u>calculating the sum</u> corresponds to finding the shortest distance between the input point and the weight hyperplane

<u>the non-linearity</u> transforms this distance,
creating a field that changes non-linearly with distance



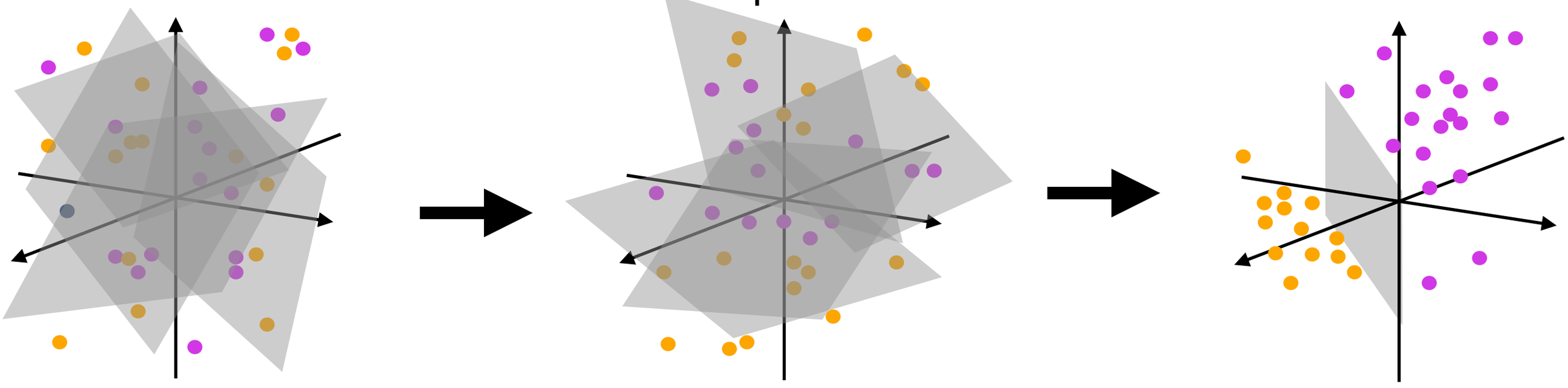
reinterpretation

a weight therefore becomes a *filter* if its hyperplane is facing a cluster of points within a region or subregion



the unit selects for the abstract feature shared by the cluster of points

reinterpretation



at each stage,

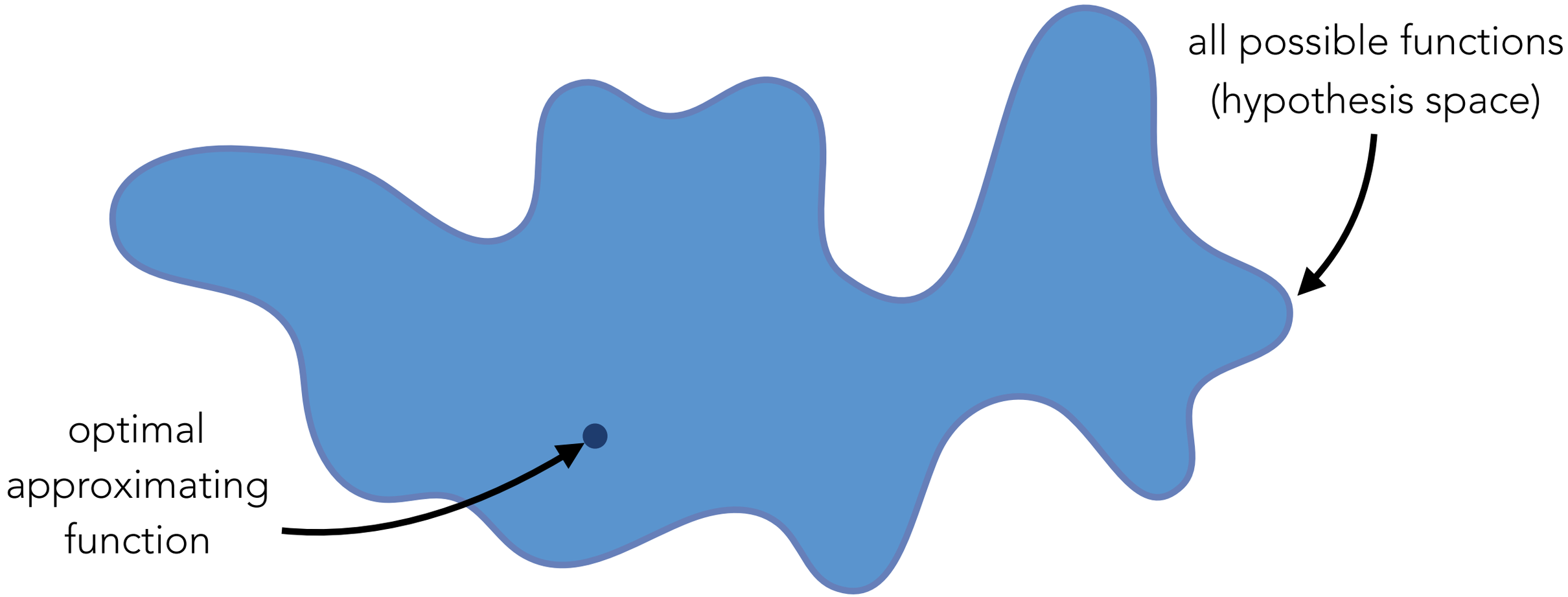
1. cut the space up with hyperplanes
2. evaluate distance of each point to each hyperplane
3. transform these distances according to non-linear function
4. transformed distances become points in new space

repeat until the data are sufficiently *linearized*

⟶ can separate class clusters with hyperplanes

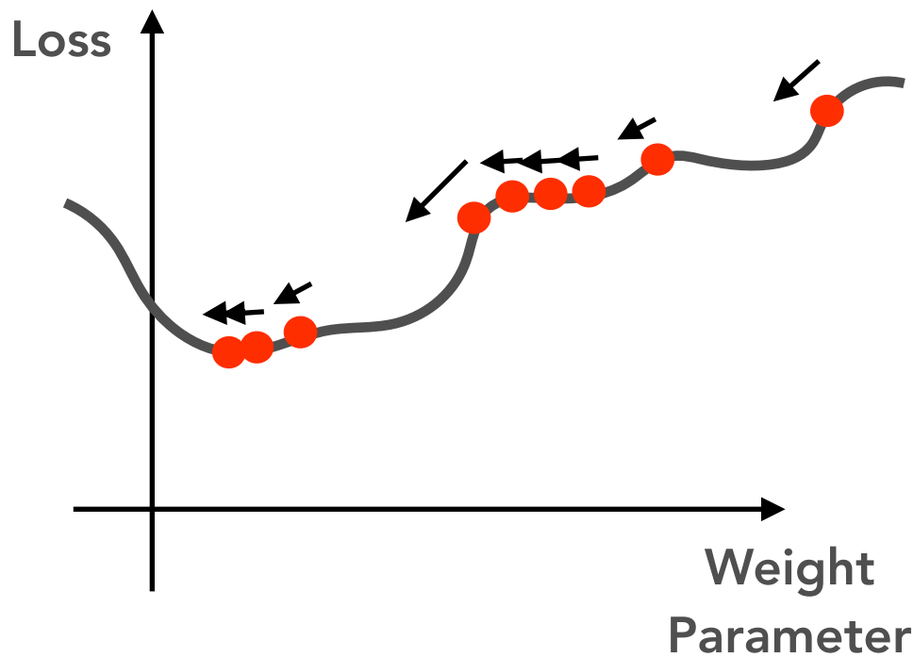
BACK-PROPAGATION

neural networks are universal function approximators,
but we still must find an optimal approximating function



we do so by adjusting the weights

learning as *optimization*



to learn the weights, we need the **derivative** of the loss w.r.t. the weight

i.e. "*how should the weight be updated to decrease the loss?*"

$$w = w - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

with multiple weights, we need the **gradient** of the loss w.r.t. the weights

$$\mathbf{w} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$$

backpropagation

a neural network defines a function of composed operations

$$f_L(\mathbf{w}_L, f_{L-1}(\mathbf{w}_{L-1}, \dots f_1(\mathbf{w}_1, \mathbf{x}) \dots))$$

and the loss $\mathcal{L}$ is a function of the network output

→ use chain rule to calculate gradients

chain rule example

$$y = w_2 e^{w_1 x}$$

input x output y parameters w_1, w_2

evaluate parameter derivatives: $\frac{\partial y}{\partial w_1}, \frac{\partial y}{\partial w_2}$

define

$$v \equiv e^{w_1 x} \longrightarrow y = w_2 v$$

$$u \equiv w_1 x \longrightarrow v = e^u$$

then

$$\frac{\partial y}{\partial w_2} = v = e^{w_1 x}$$

chain rule

$$\frac{\partial y}{\partial w_1} = \boxed{\frac{\partial y}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w_1}} = w_2 \cdot e^{w_1 x} \cdot x$$

back-propagation

recall

1st layer

$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$$
$$\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$$

2nd layer

$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$$
$$\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$$

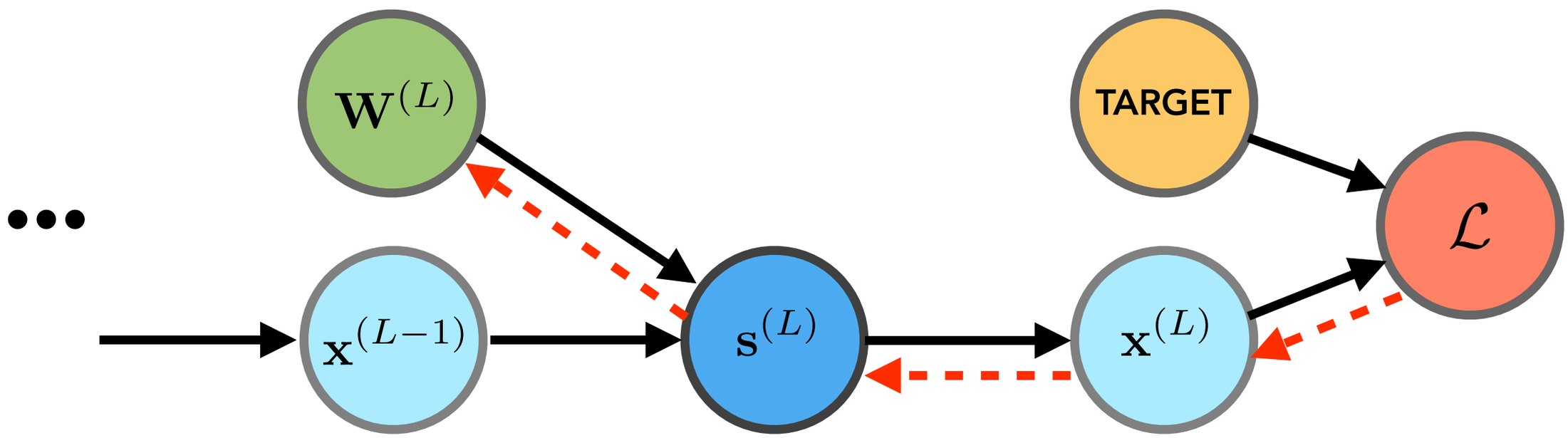
$\bullet\bullet\bullet$

Loss

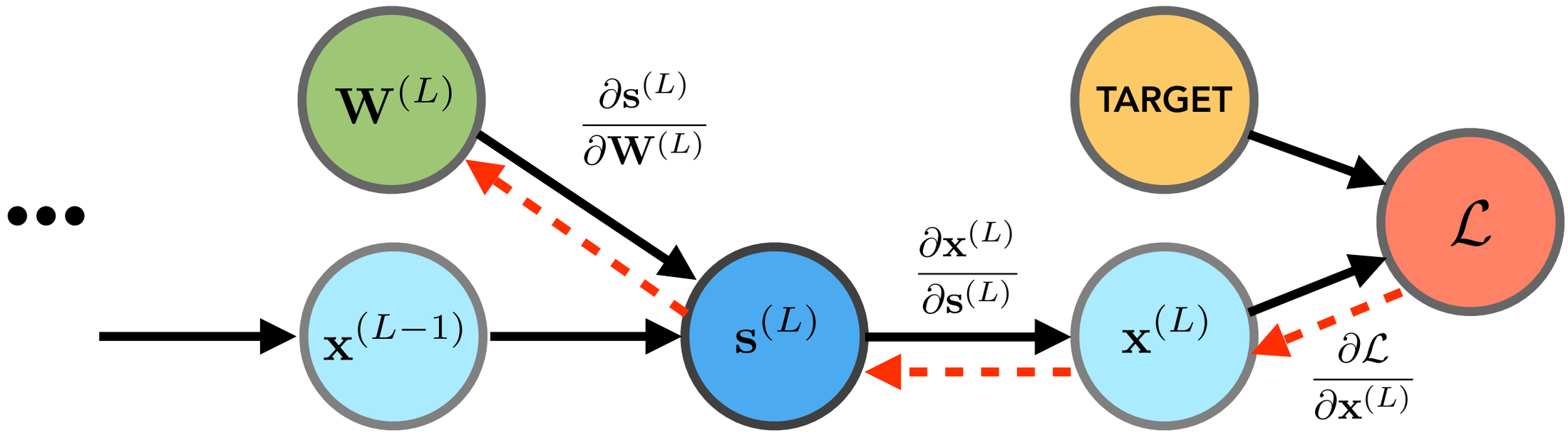
$$\mathcal{L}$$

calculate $\nabla_{W^{(1)}}\mathcal{L}, \nabla_{W^{(2)}}\mathcal{L}, \ldots$ let's start with the final layer: $\nabla_{W^{(L)}}\mathcal{L}$

to determine the chain rule ordering, we'll draw the dependency graph



backpropagation



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

depends on the form of the loss

derivative of the non-linearity

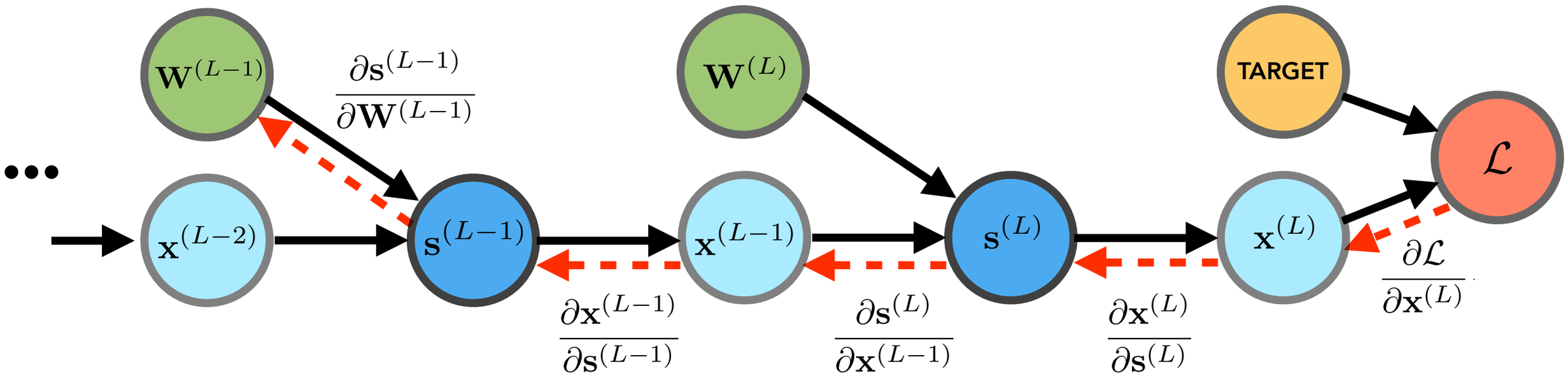
$$\frac{\partial}{\partial \mathbf{W}^{(L)}}(\mathbf{W}^{(L)\intercal}\mathbf{x}^{(L-1)}) = \mathbf{x}^{(L-1)\intercal}$$

note $\nabla_{\mathbf{W}^{(L)}}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}}$ is notational convention

backpropagation

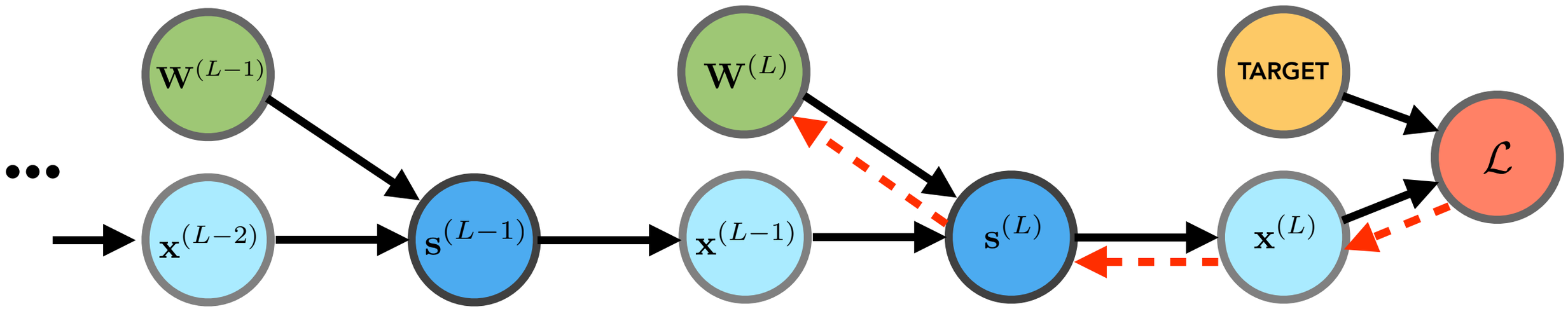
now let's go back one more layer…

again we'll draw the dependency graph:



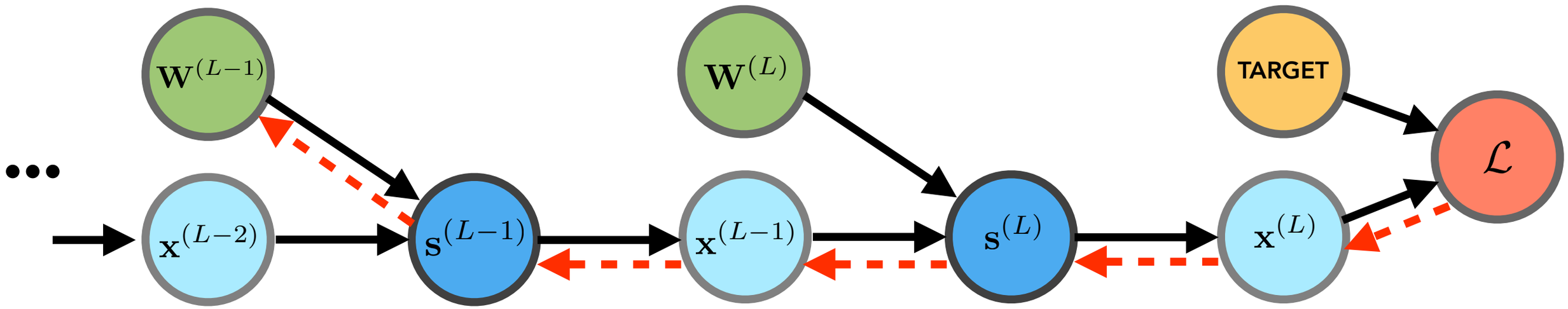
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{x}^{(L-1)}} \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{s}^{(L-1)}} \frac{\partial \mathbf{s}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}$$

backpropagation



notice that some of the same terms appear in both gradients

specifically, we can reuse $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(\ell)}}$ to calculate gradients in reverse order



backpropagation

BACKPROPAGATION ALGORITHM

calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}}$

store $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(L)}}$

for $\ell = [L-1, \ldots, 1]$

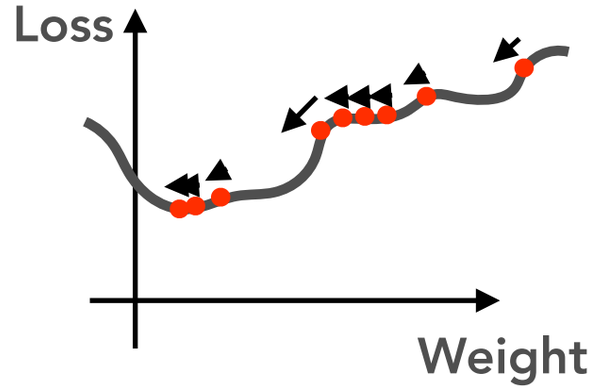
use $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(\ell+1)}}$ to calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(\ell)}}$

store $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(\ell)}}$

return $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}}$

recapitulation

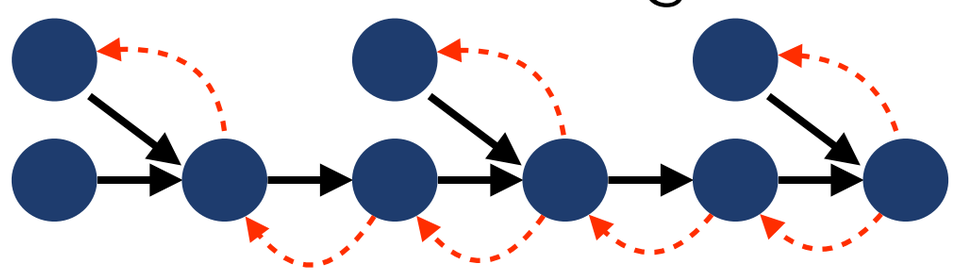
update weights using gradient of loss



backpropagation calculates the loss gradients w.r.t. internal weights

→ *"credit assignment"* via chain rule

gradient is *propagated backward* through the network



most deep learning software libraries automatically calculate gradients

→ *"automatic differentiation"* or *"auto-diff"*

→ can calculate gradients for any differentiable operation

TAKE HOME PRACTICE

- 60 Minutes Tutorial with Pytorch
  - Auto-differentiation
  - Neural Network
- https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html