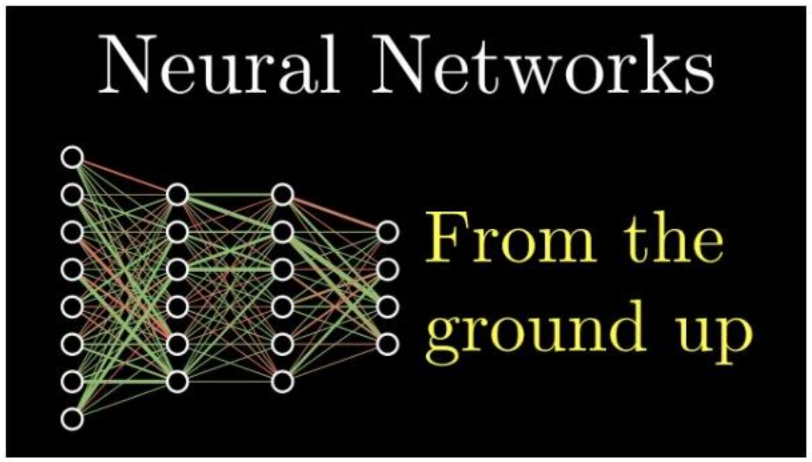
on Brightspace

Required Video (link on Brightspace)
(for today) *But What *is* a Neural Network?*
by 3blue1brown, Grant Sanderson

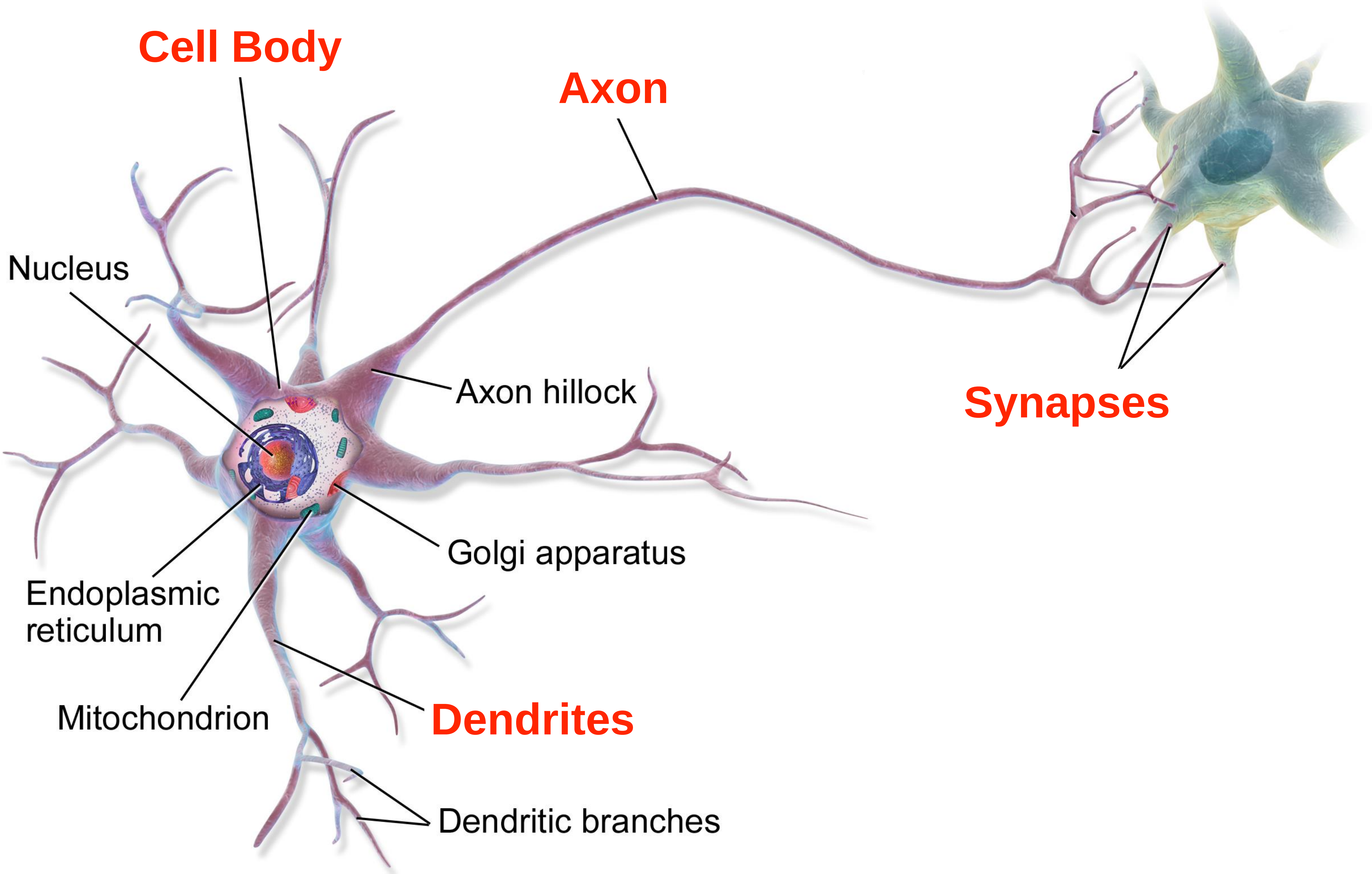


Homework 2 (on Brightspace, in today's slides)
Due Tue Sep 14

Simple Models of Neurons

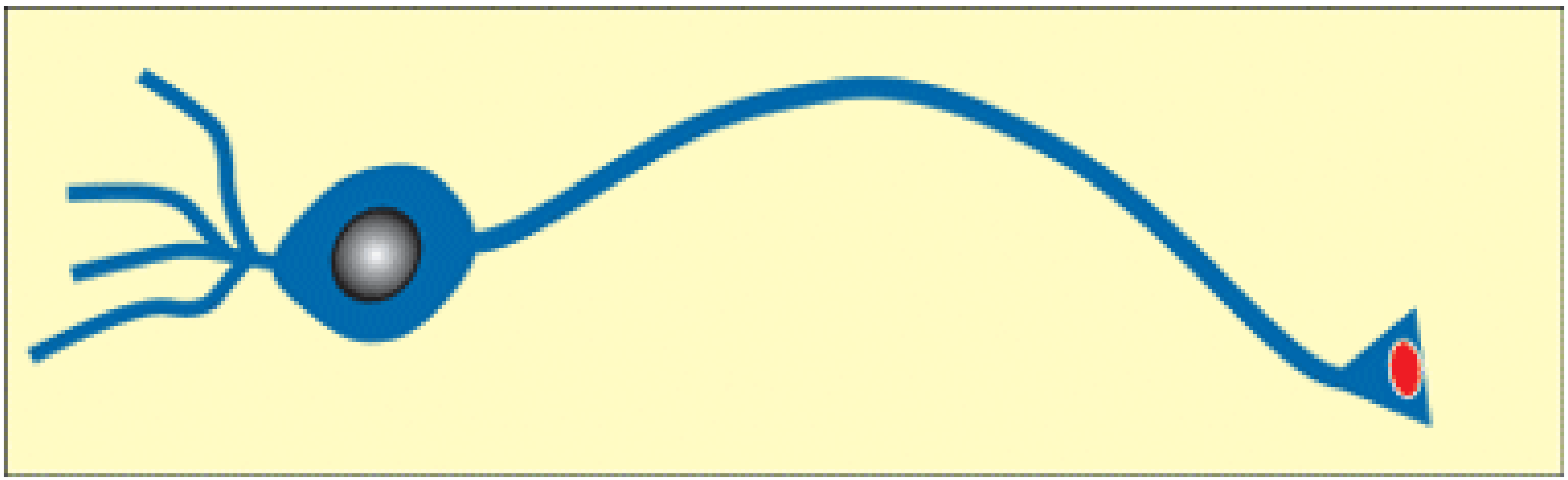
Brief Overview of Neuron Structure and Function

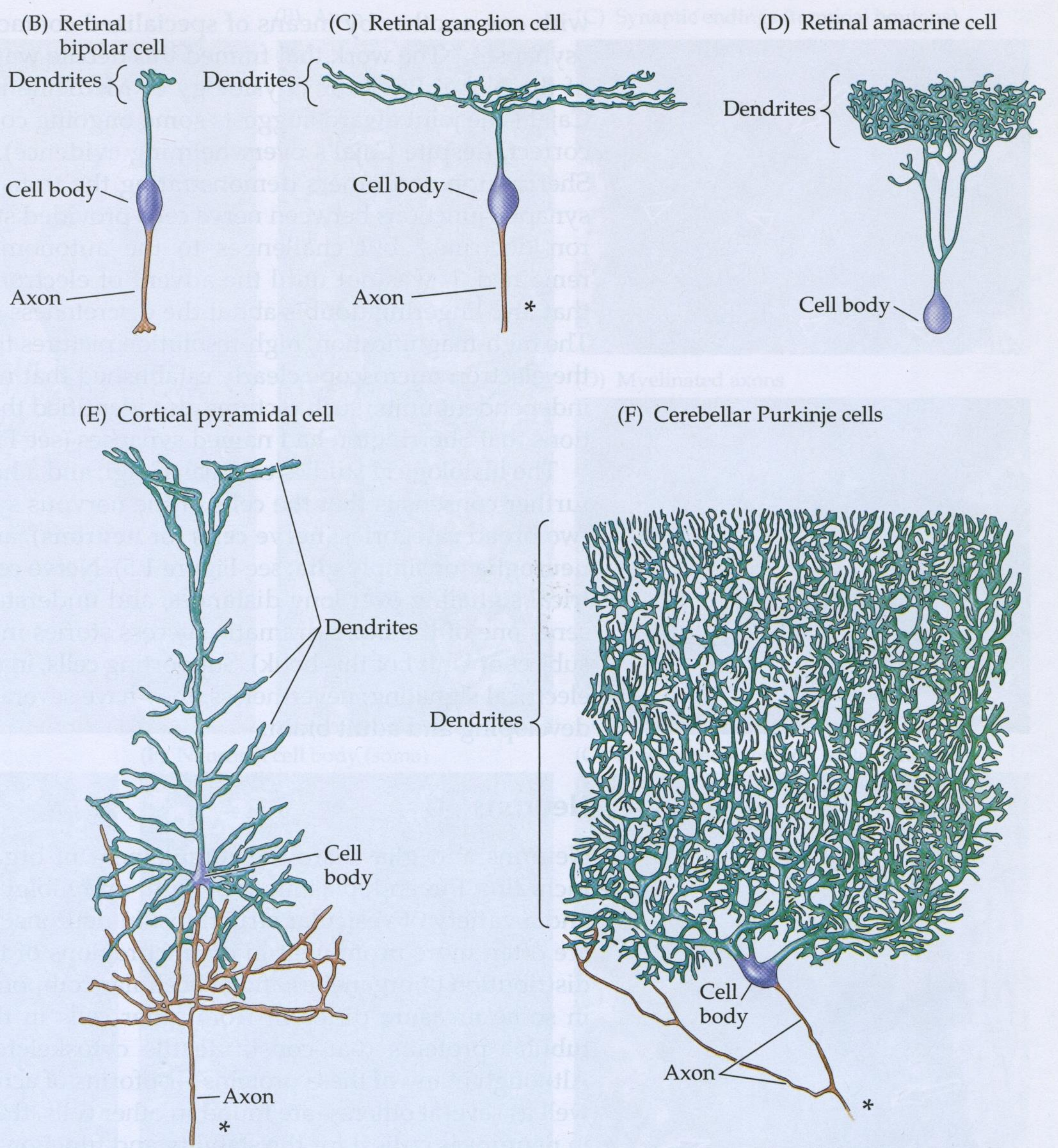
Neuron



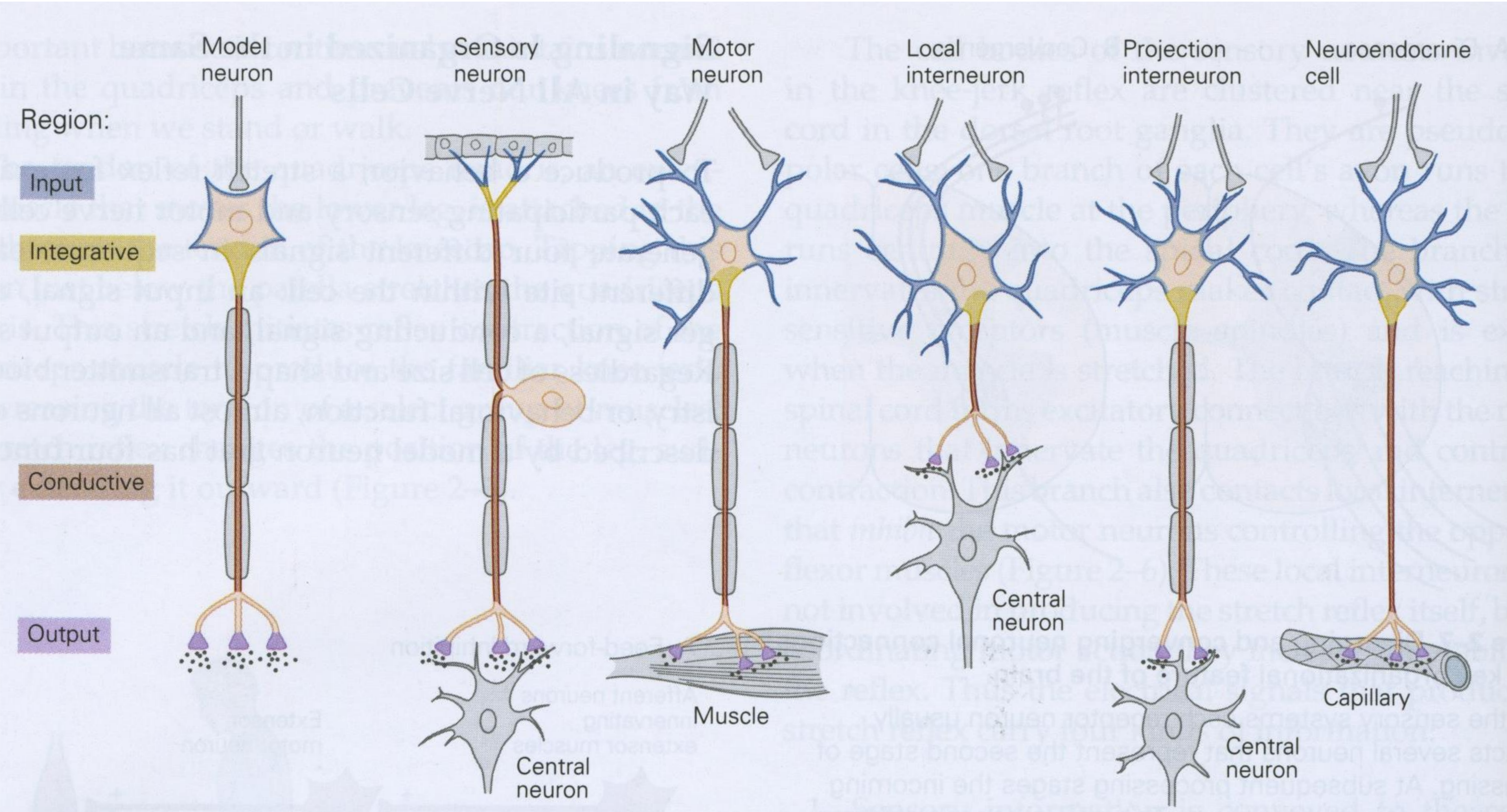
Cell Body
Axon
Nucleus
Axon hillock
Synapses
Golgi apparatus
Endoplasmic
reticulum
Mitochondrion
Dendrites
Dendritic branches

Dendrites Cell Body Axon Synapses

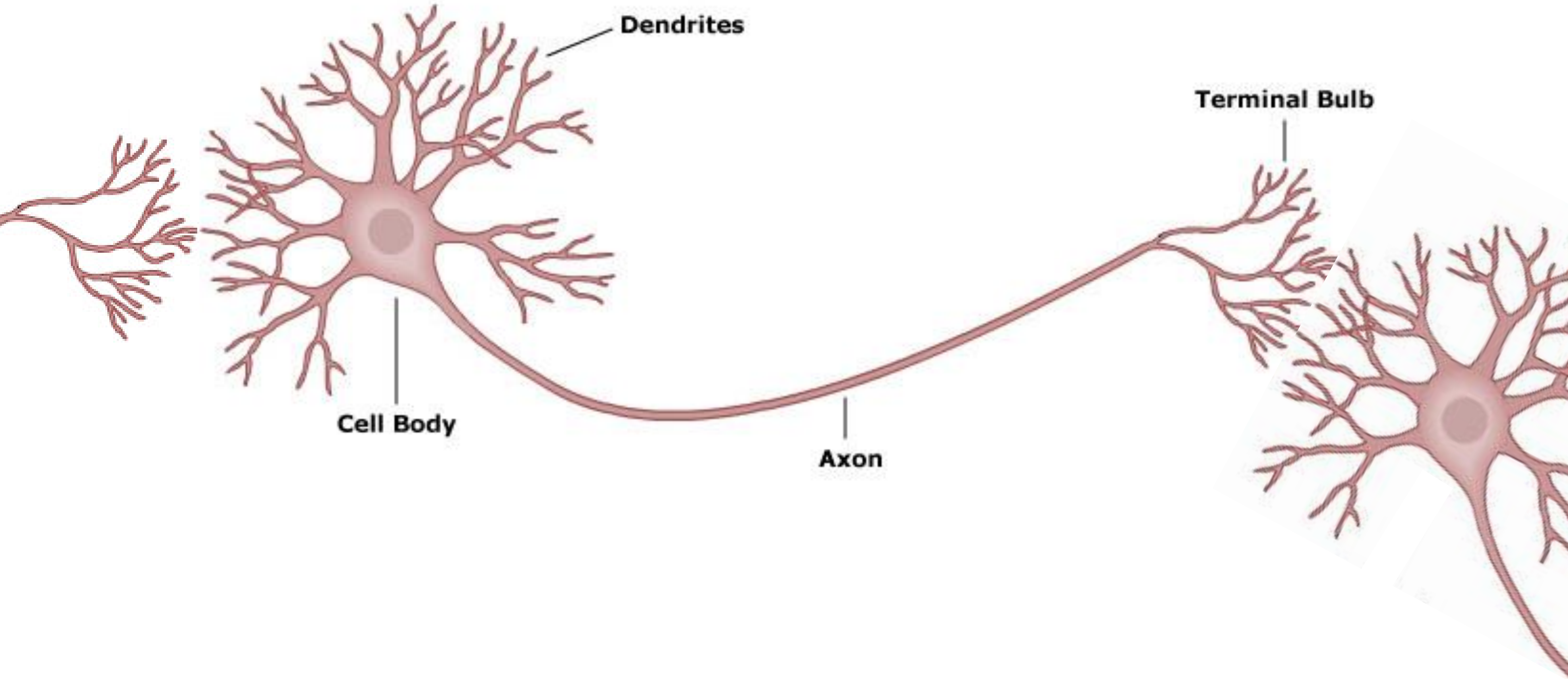


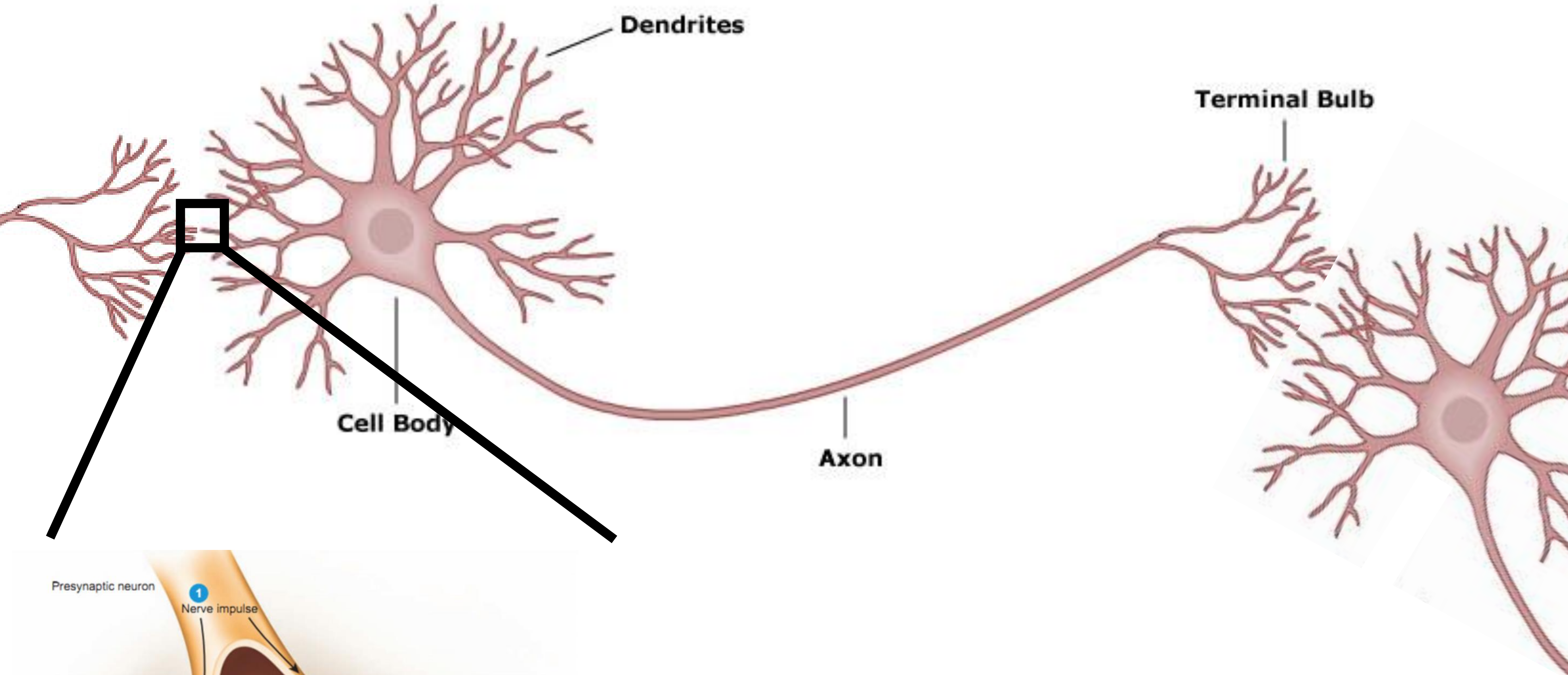

(B) Retinal bipolar cell
Dendrites
Cell body
Axon
(C) Retinal ganglion cell
Dendrites
Cell body
Axon
*
(D) Retinal amacrine cell
Dendrites
Cell body
(E) Cortical pyramidal cell
Dendrites
Cell body
Axon
*
(F) Cerebellar Purkinje cells
Dendrites
Cell body
Axon
*

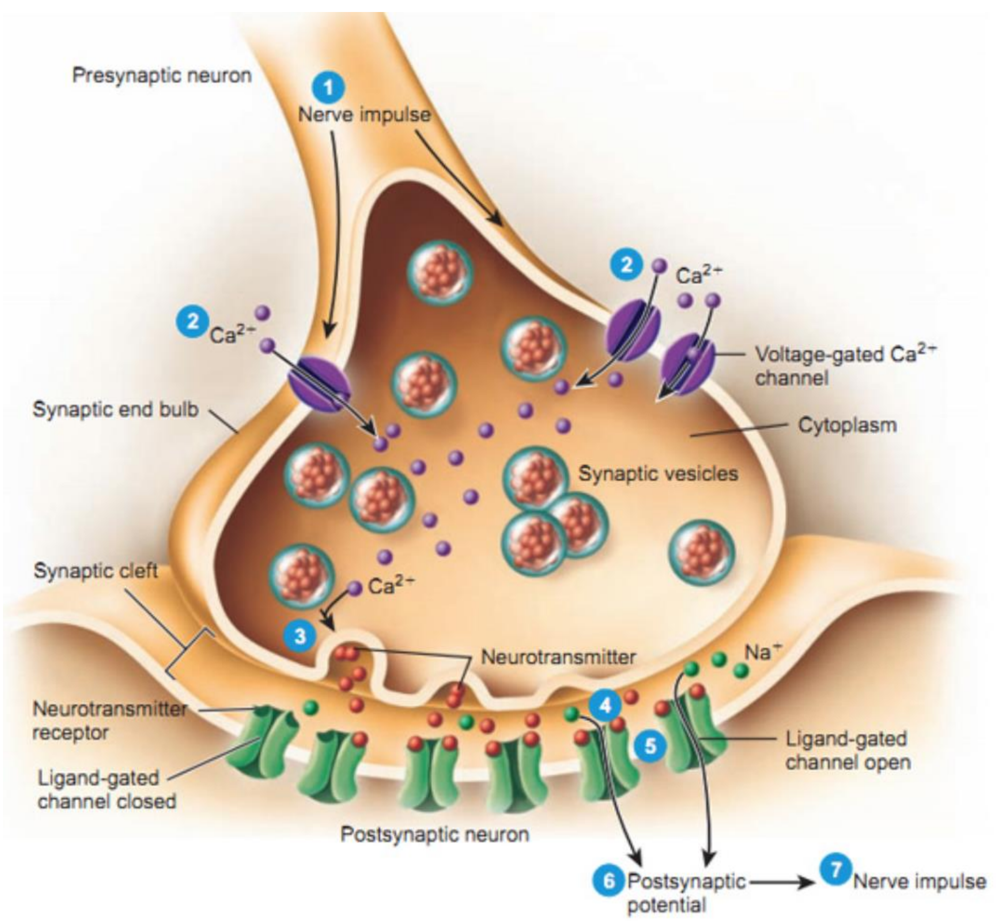
functional similarities despite morphological differences



we will not discuss different types of neurons, but instead model idealized neurons

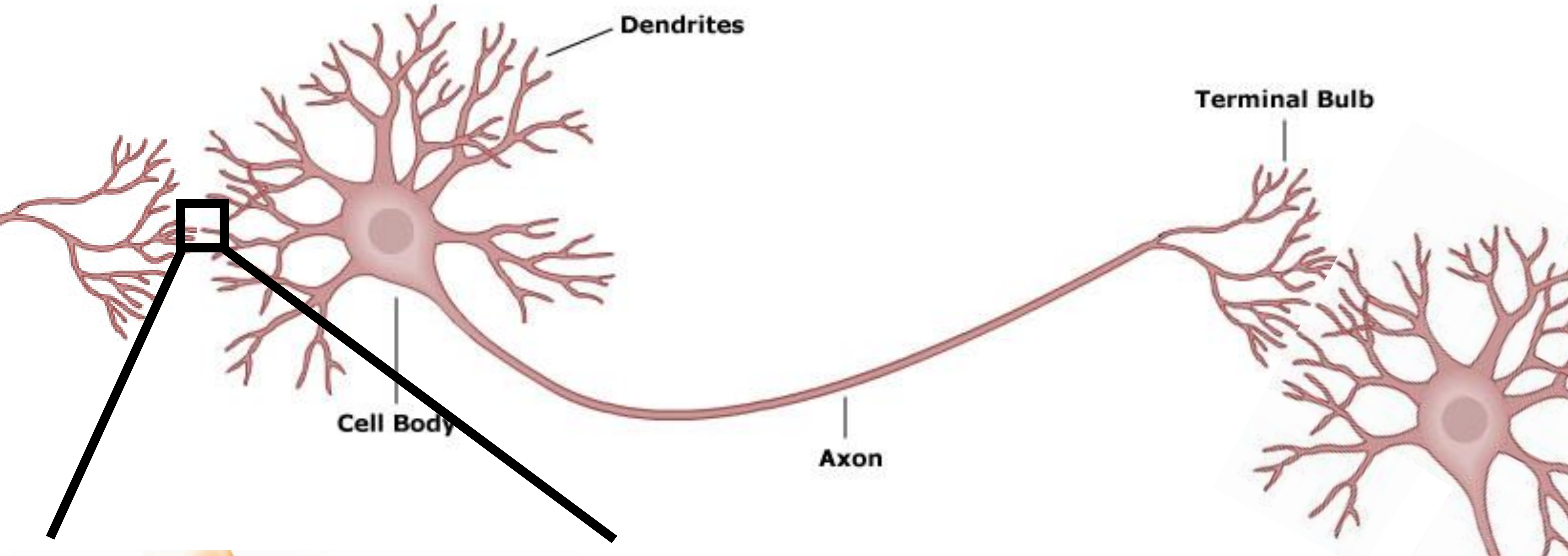

Dendrites
Terminal Bulb
Cell Body
Axon

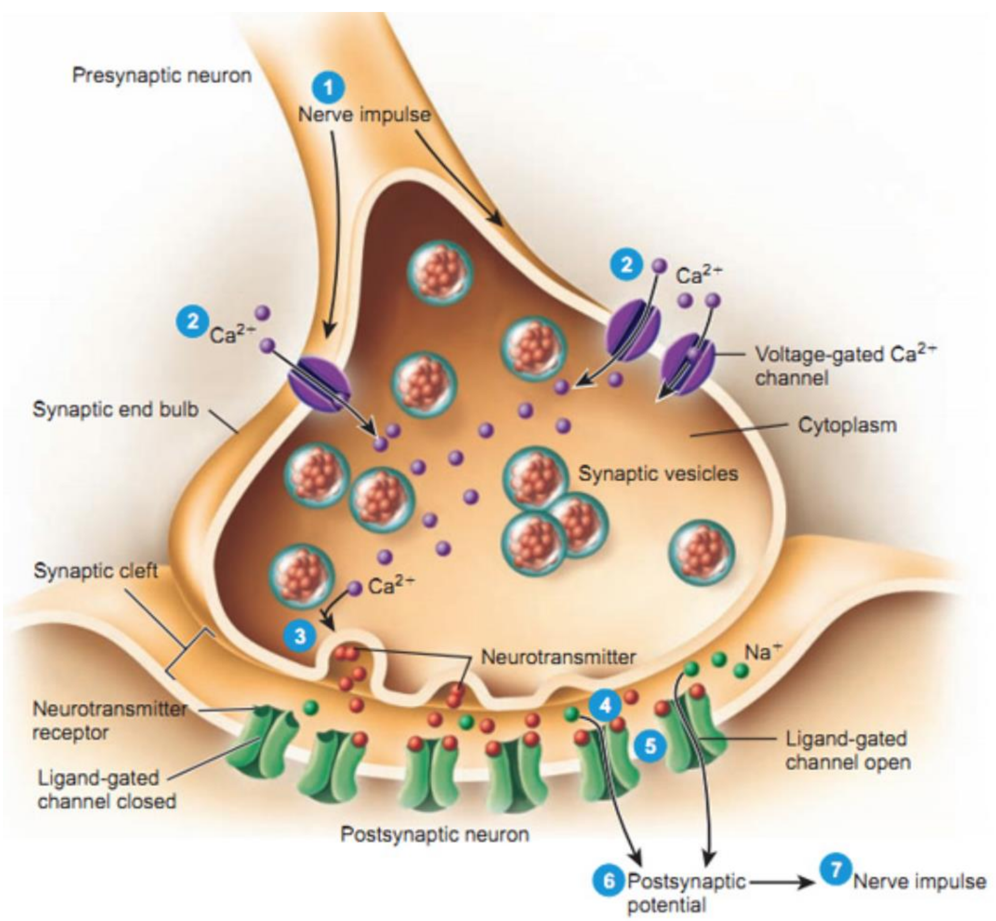




neurotransmitters cause certain channels to open, allowing charged ions to flow at certain times

ions flow, which changes the voltage of the neuron

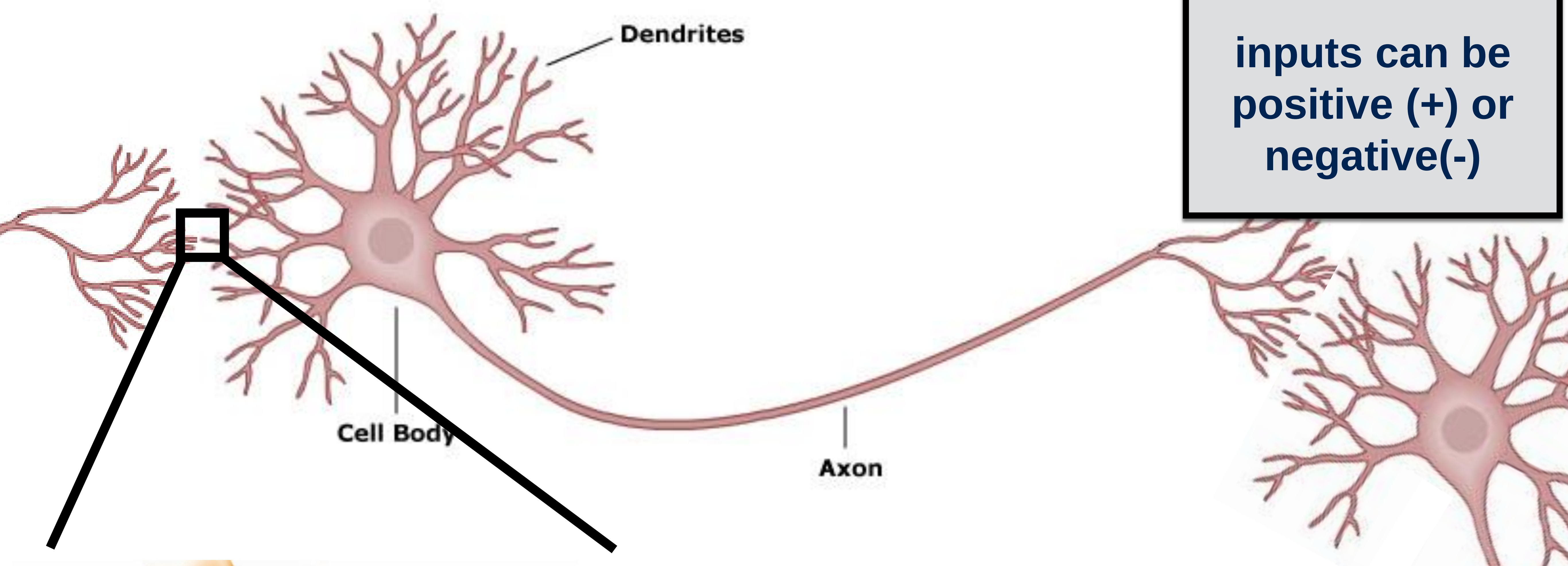


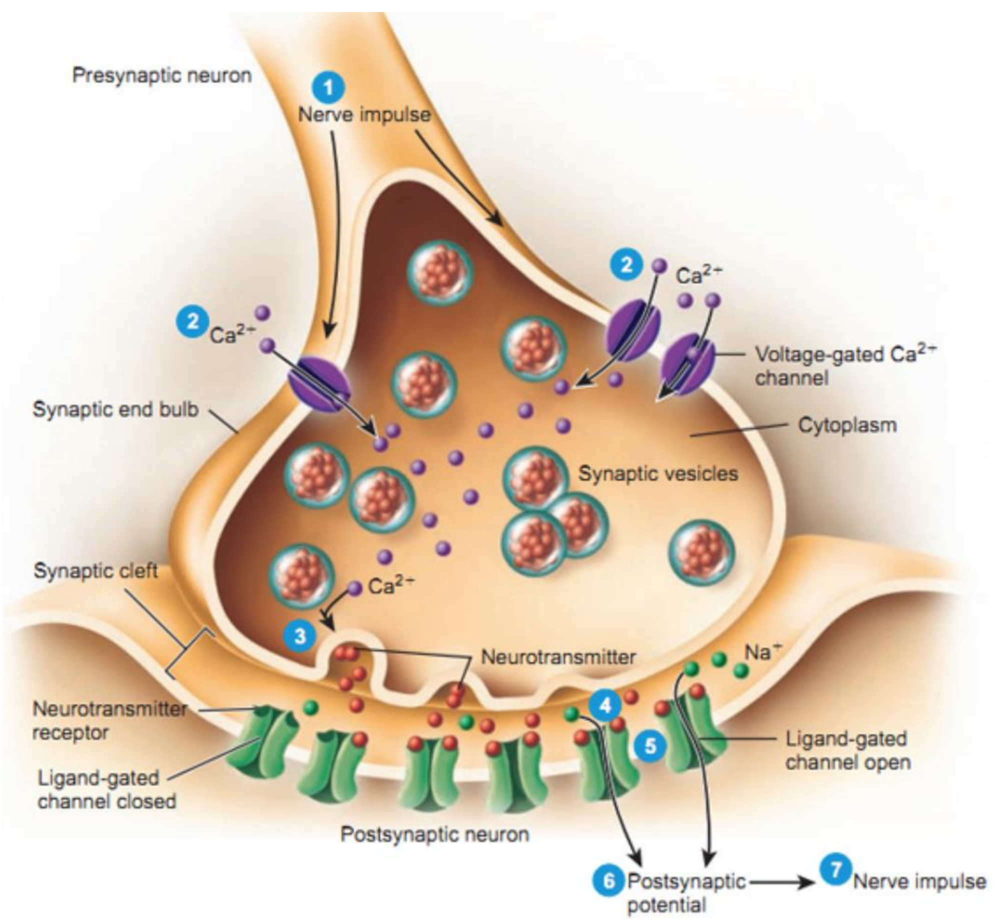


EPSPs (excitatory postsynaptic potentials) positive changes in membrane potential caused by excitatory pre-synaptic neurons (e.g., glutamate and NMDA receptors)

IPSPs (inhibitory postsynaptic potentials) negative changes in membrane potential caused by inhibitory pre-synaptic neurons (e.g., GABA and GABA receptors)

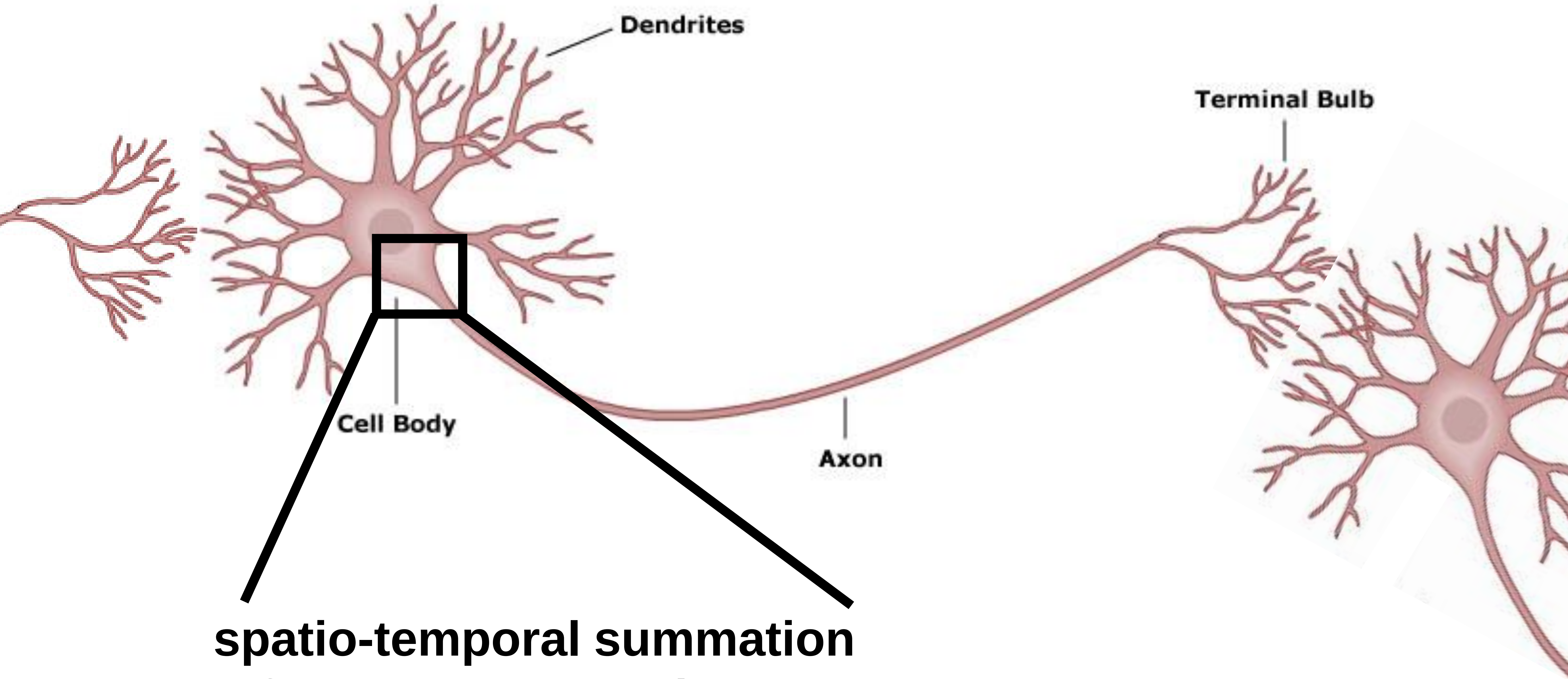
inputs can be positive (+) or negative(-)





EPSPs (excitatory postsynaptic potentials) positive changes in membrane potential caused by excitatory pre-synaptic neurons (e.g., glutamate and NMDA receptors)

IPSPs (inhibitory postsynaptic potentials) negative changes in membrane potential caused by inhibitory pre-synaptic neurons (e.g., GABA and GABA receptors)

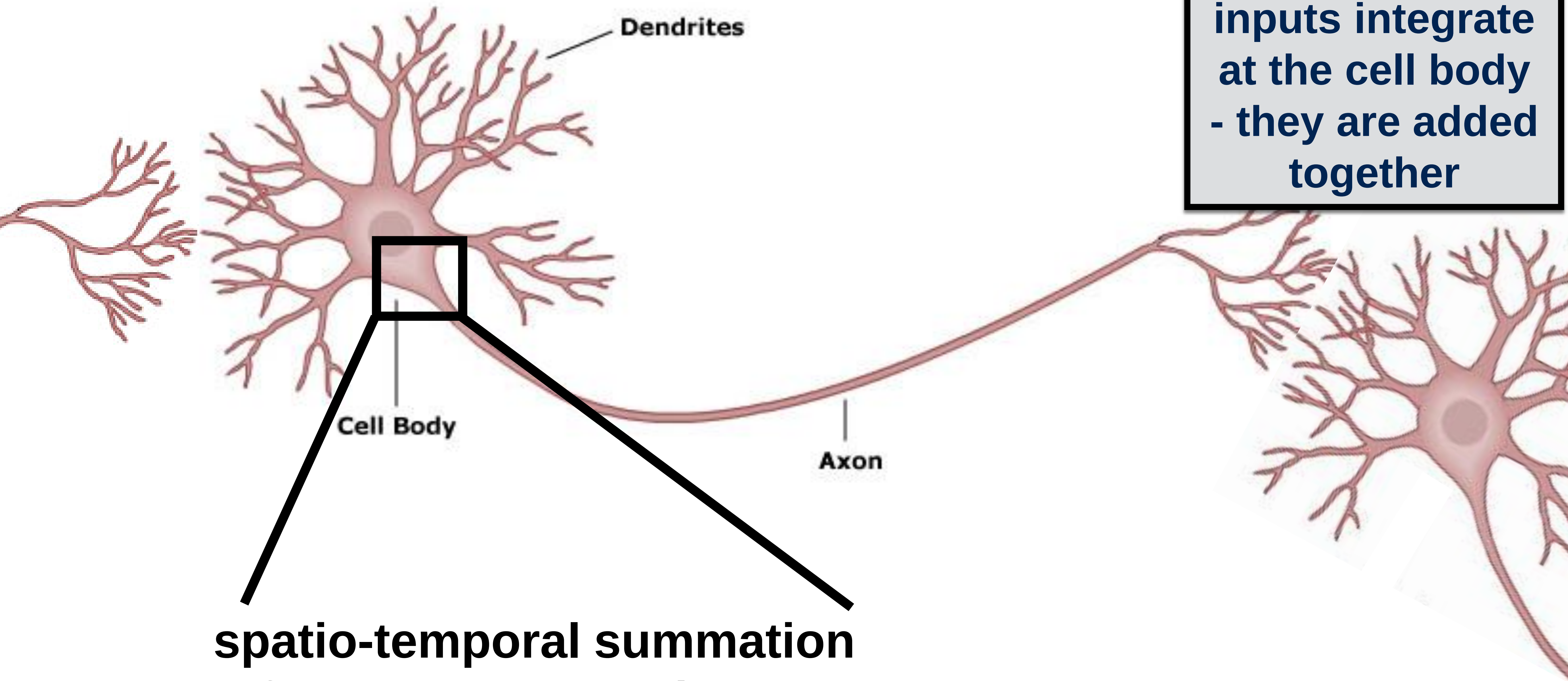


spatio-temporal summation of many EPSPs and IPSPs

from many of synaptic stimulations in close spatio-temporal proximity

potentials spread, but dissipate over space and time because neurons not good conductors

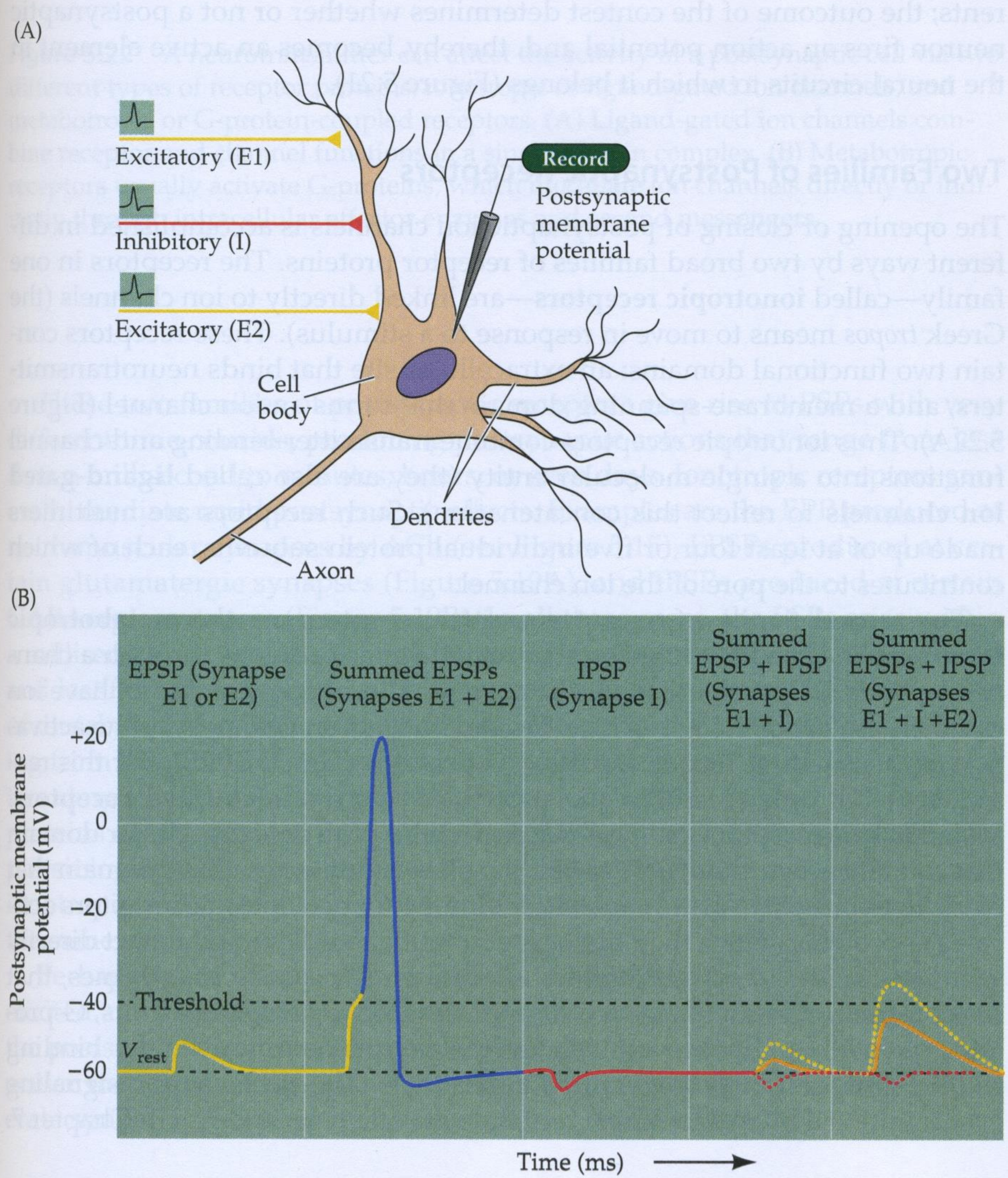
inputs integrate at the cell body - they are added together

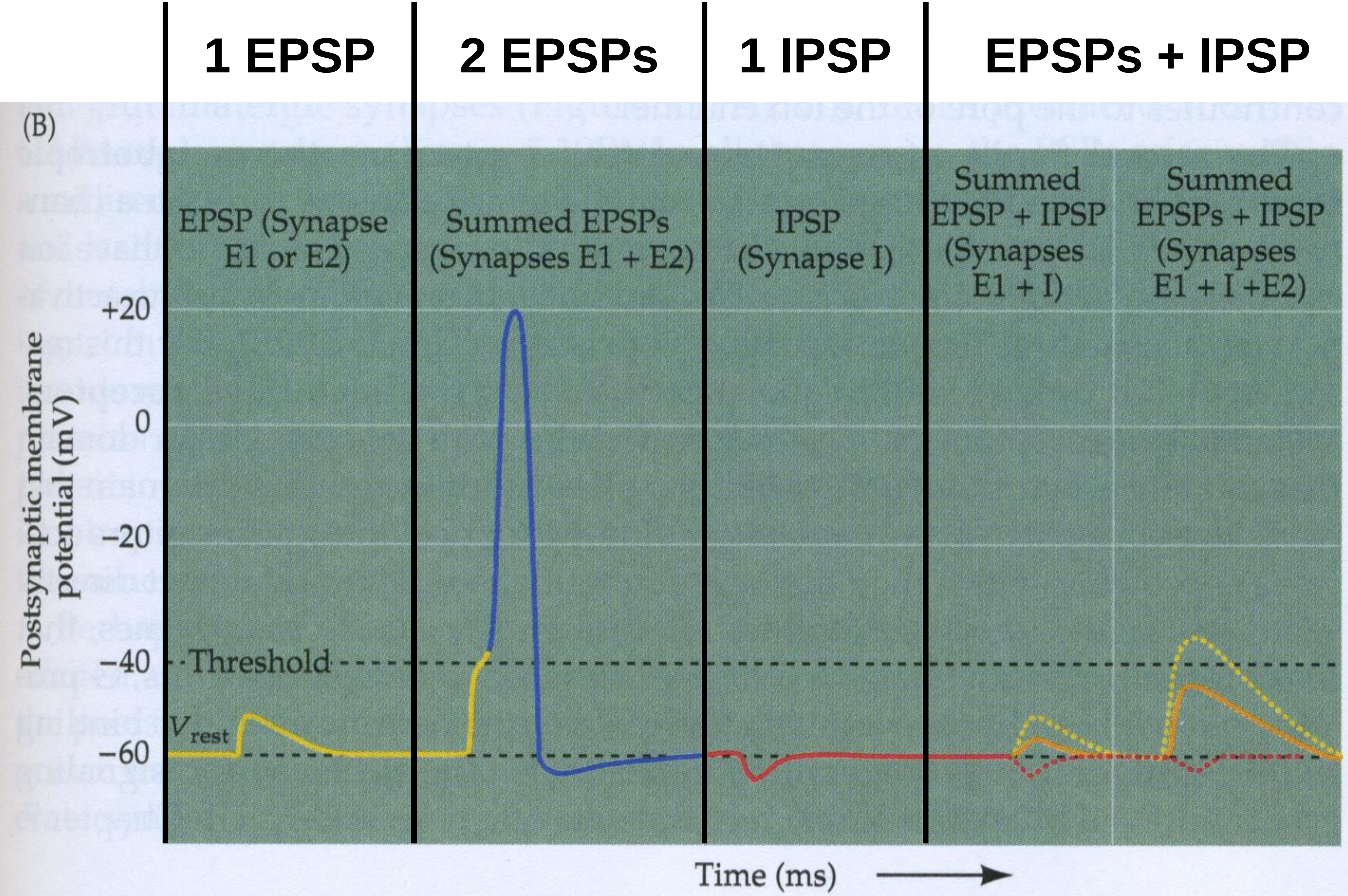


spatio-temporal summation of many EPSPs and IPSPs

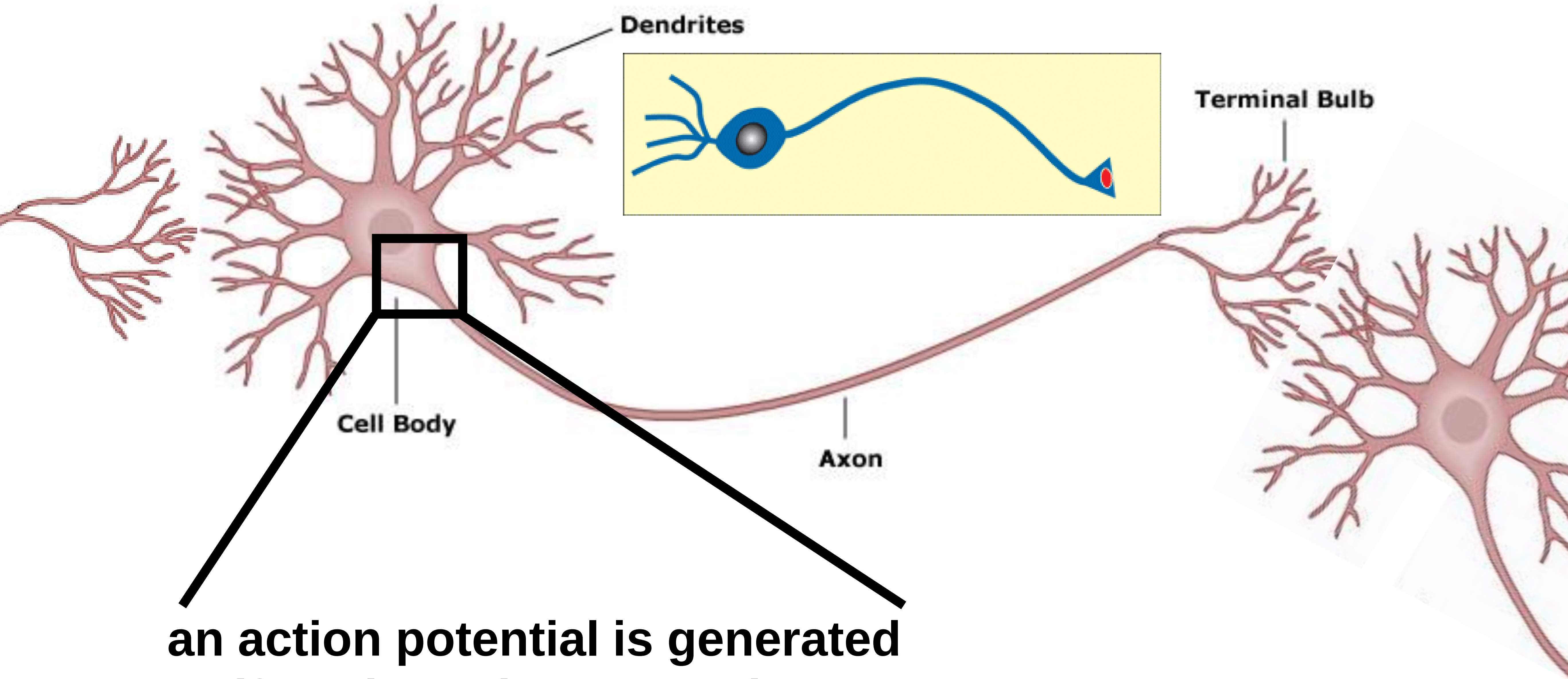
from many of synaptic stimulations in close spatio-temporal proximity

potentials spread, but dissipate over space and time because neurons not good conductors


(A)
Excitatory (E1)
Inhibitory (I)
Excitatory (E2)
Record
Postsynaptic
membrane
potential
Cell
body
Dendrites
Axon
(B)
EPSP (Synapse
E1 or E2)
Summed EPSPs
(Synapses E1 + E2)
IPSP
(Synapse I)
Summed
EPSP + IPSP
(Synapses
E1 + I)
Summed
EPSPs + IPSP
(Synapses
E1 + I +E2)
+20
0
−20
−40
Threshold
V_{rest}
−60
Postsynaptic membrane
potential (mV)
Time (ms)

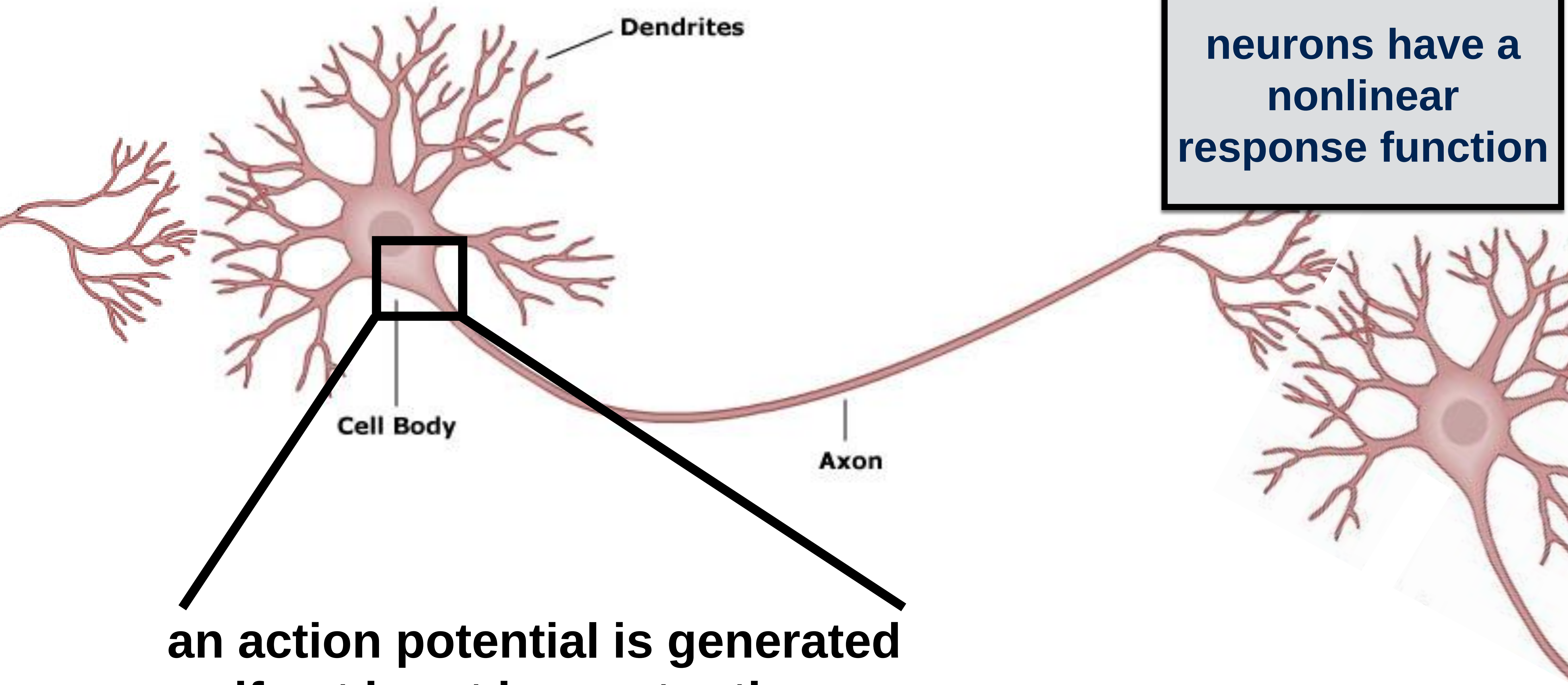


1 EPSP
2 EPSPs
1 IPSP
EPSPs + IPSP
(B)
EPSP (Synapse E1 or E2)
Summed EPSPs (Synapses E1 + E2)
IPSP (Synapse I)
Summed EPSP + IPSP (Synapses E1 + I)
Summed EPSPs + IPSP (Synapses E1 + I +E2)
+20
0
−20
−40
−60
Threshold
V_{rest}
Postsynaptic membrane potential (mV)
Time (ms)



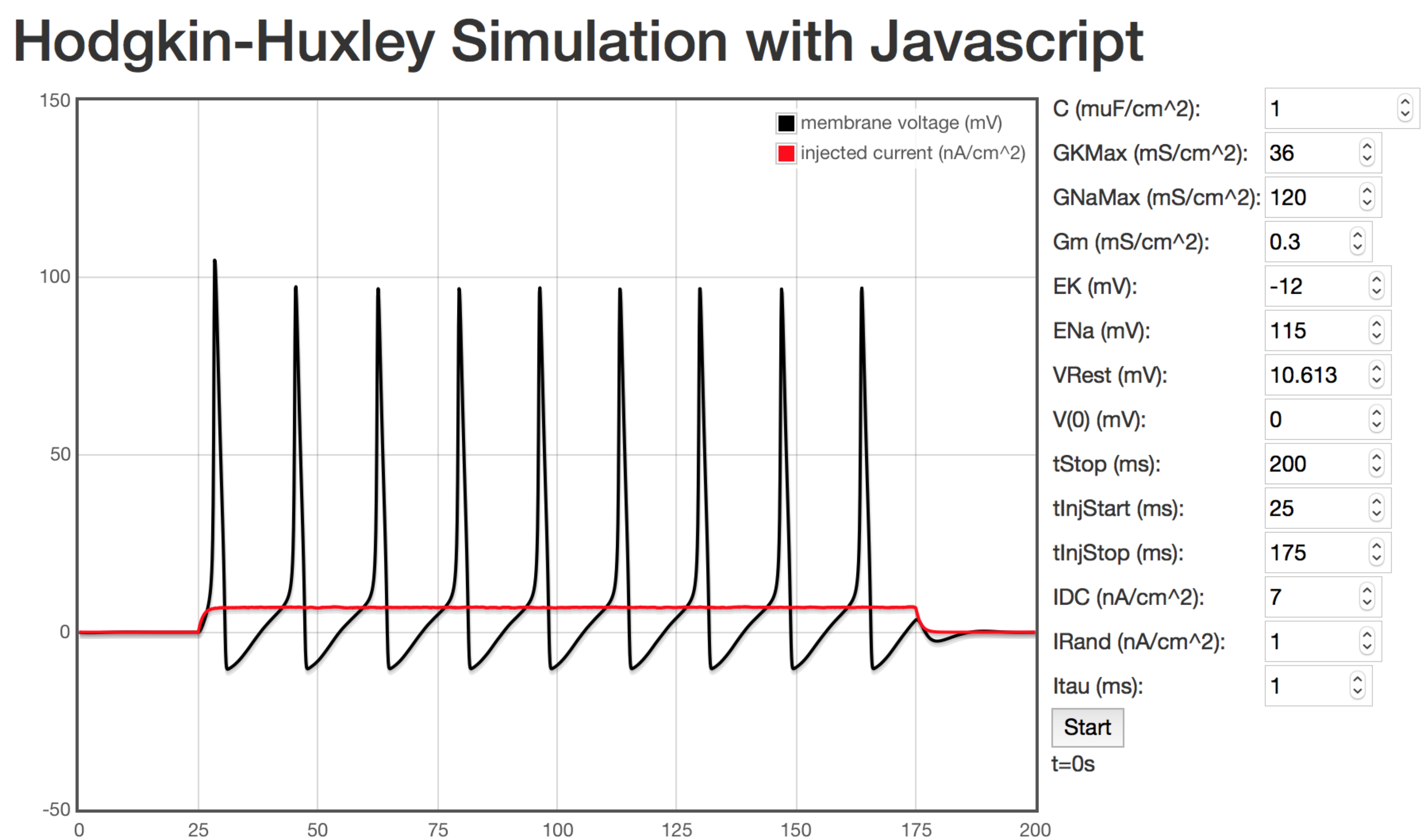
an action potential is generated if net input is greater than an activation threshold

neurons have a nonlinear response function



an action potential is generated
if net input is greater than
an activation threshold

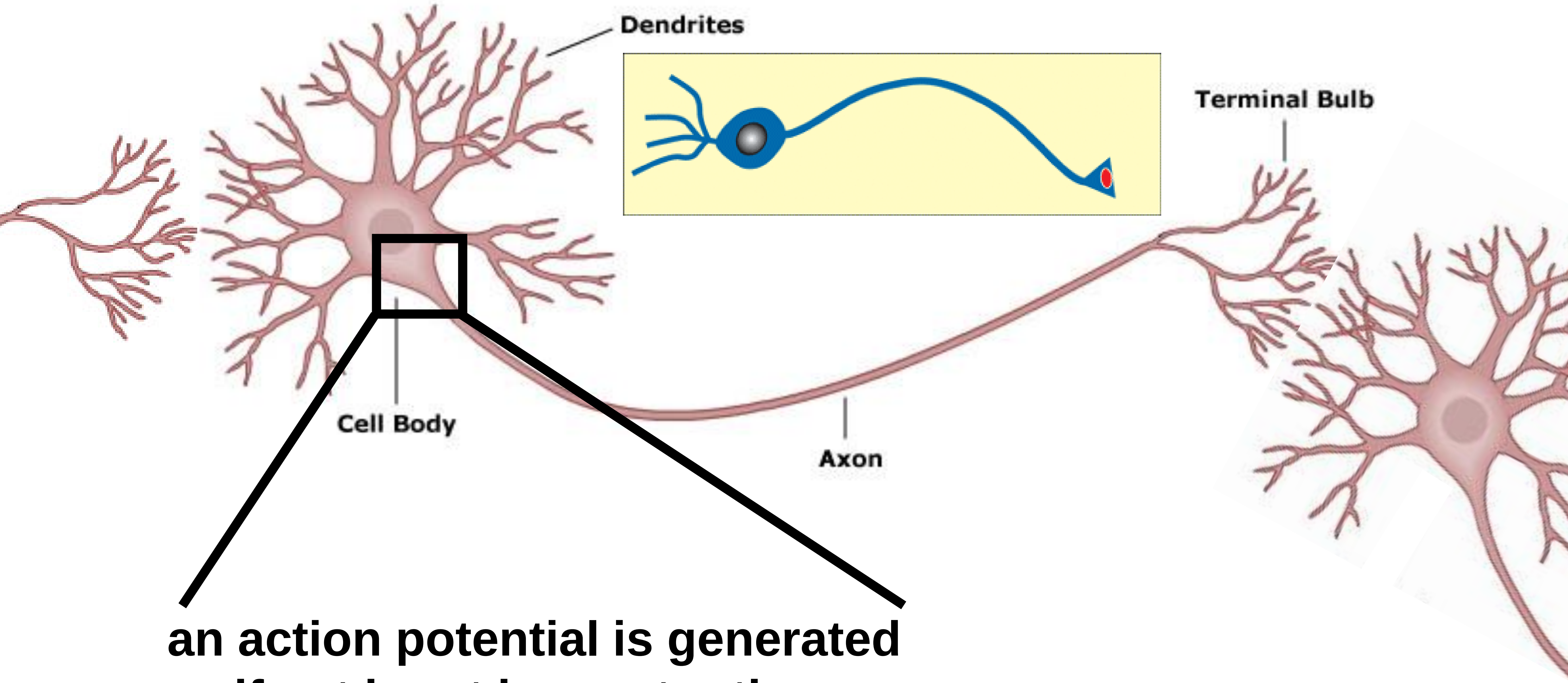
action potential simulator



http://myselph.de/hodgkinHuxley.html

IDC (input/injected current)
IRand (random variability in current)

try IDC=2.355 vs. IDC=2.356 with IRand=0

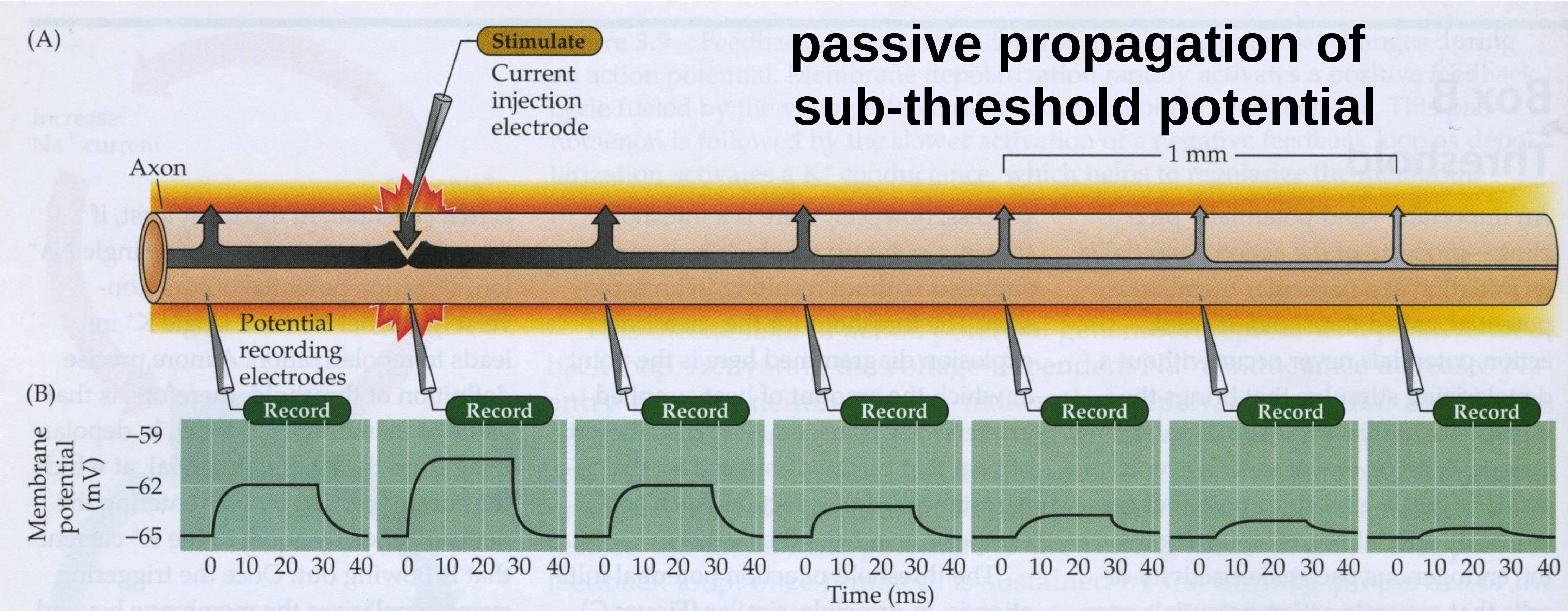


an action potential is generated if net input is greater than an activation threshold

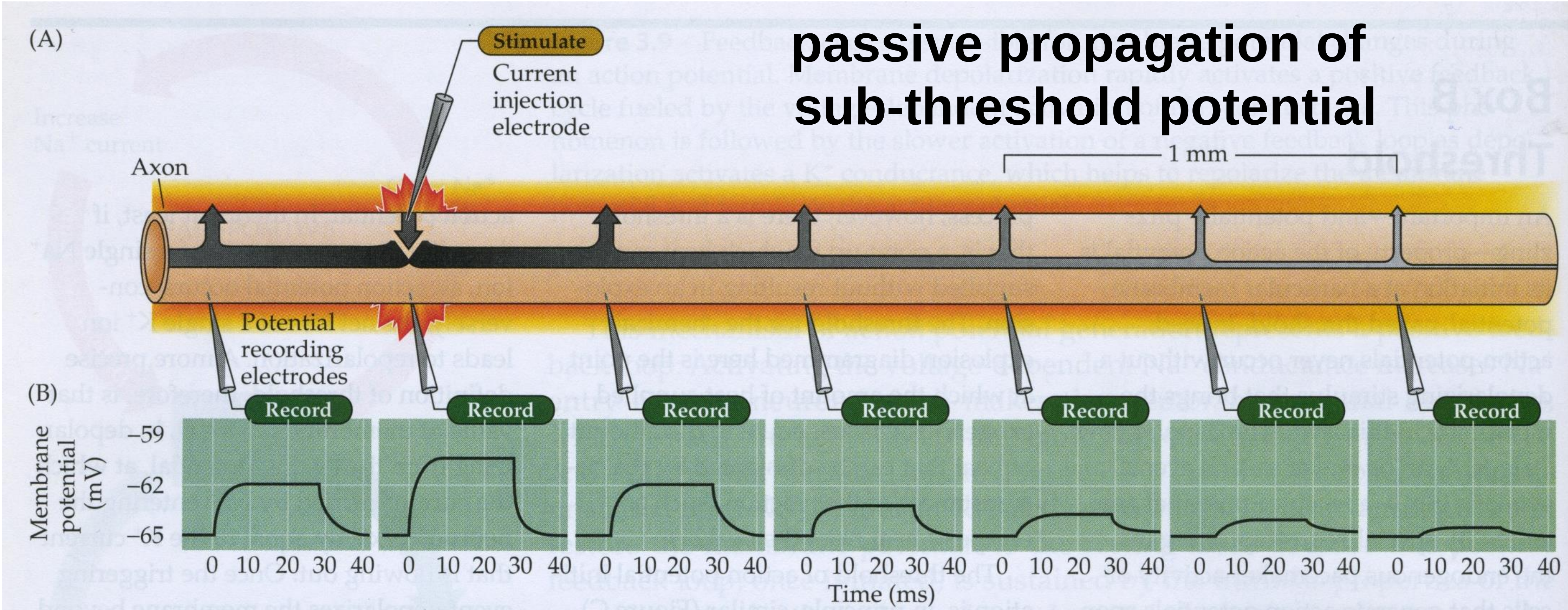
action potential is nature's way of making the axon conductive over relatively long distances

ion channels can open or close depending on the local voltage gradient

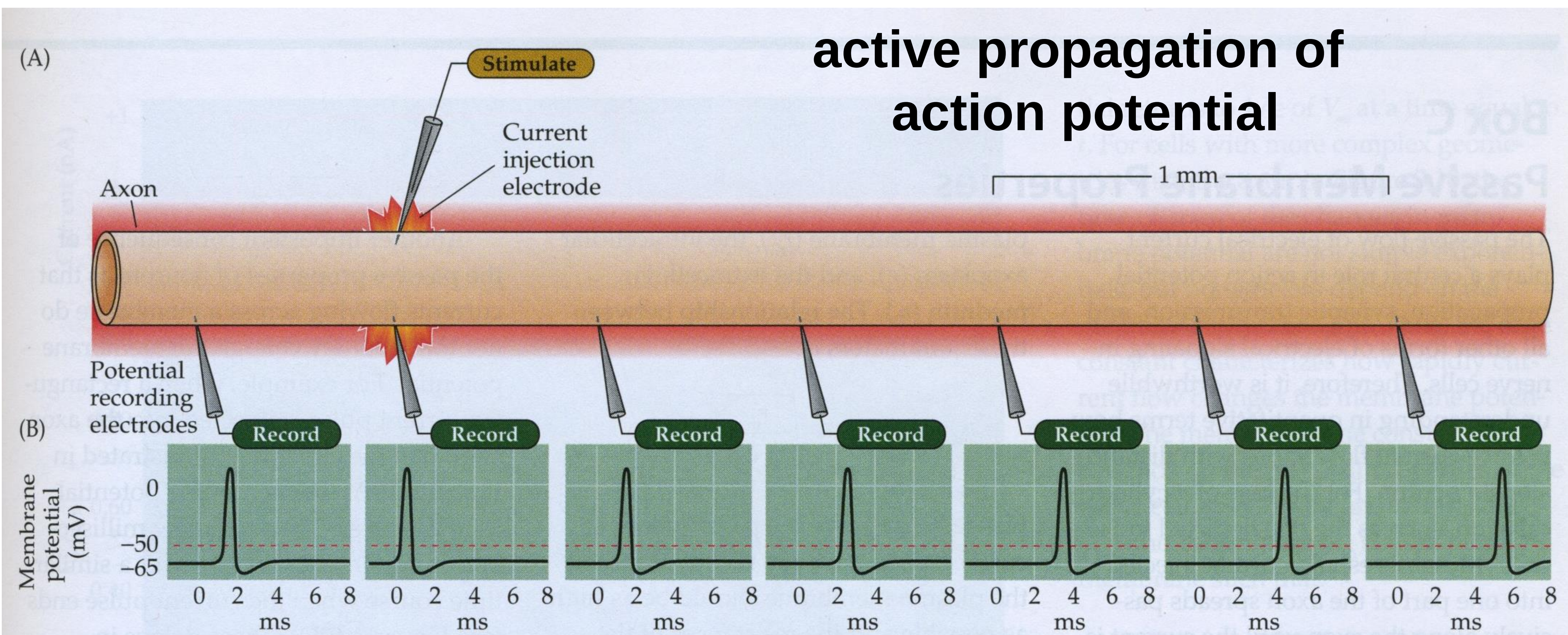
passive propagation of sub-threshold potential

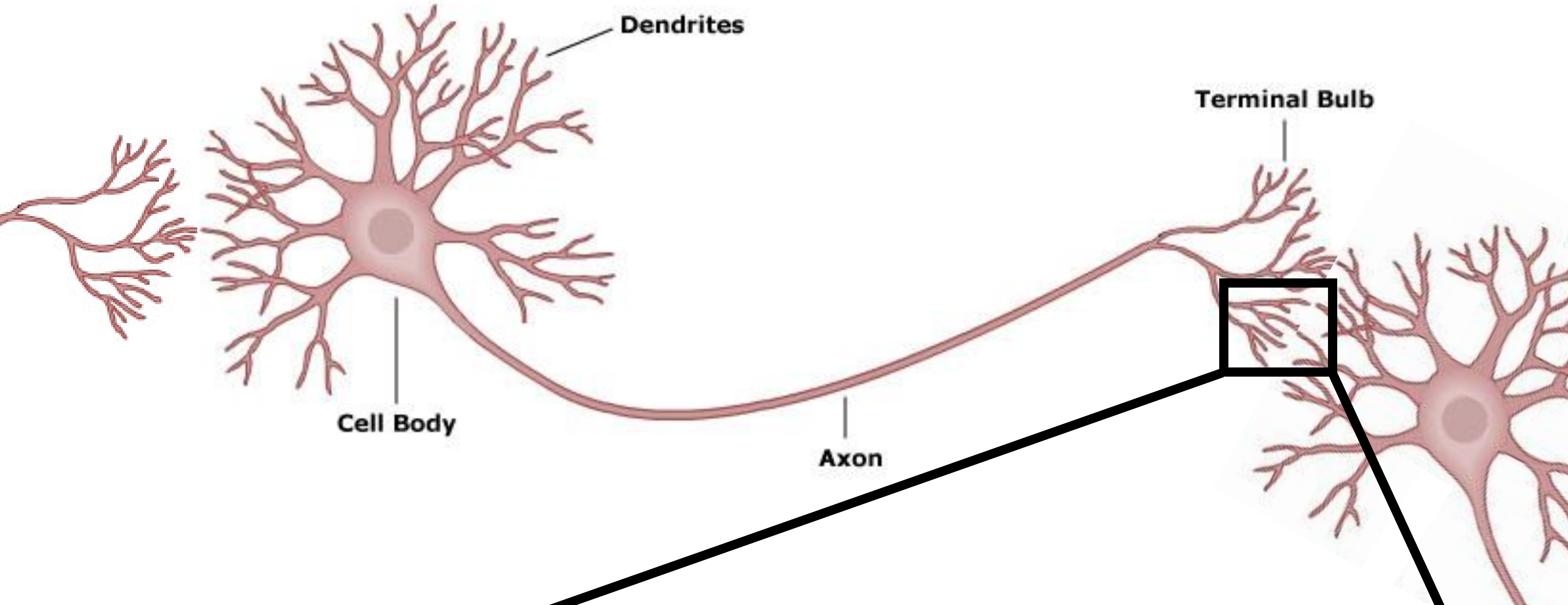


passive propagation of sub-threshold potential



active propagation of action potential



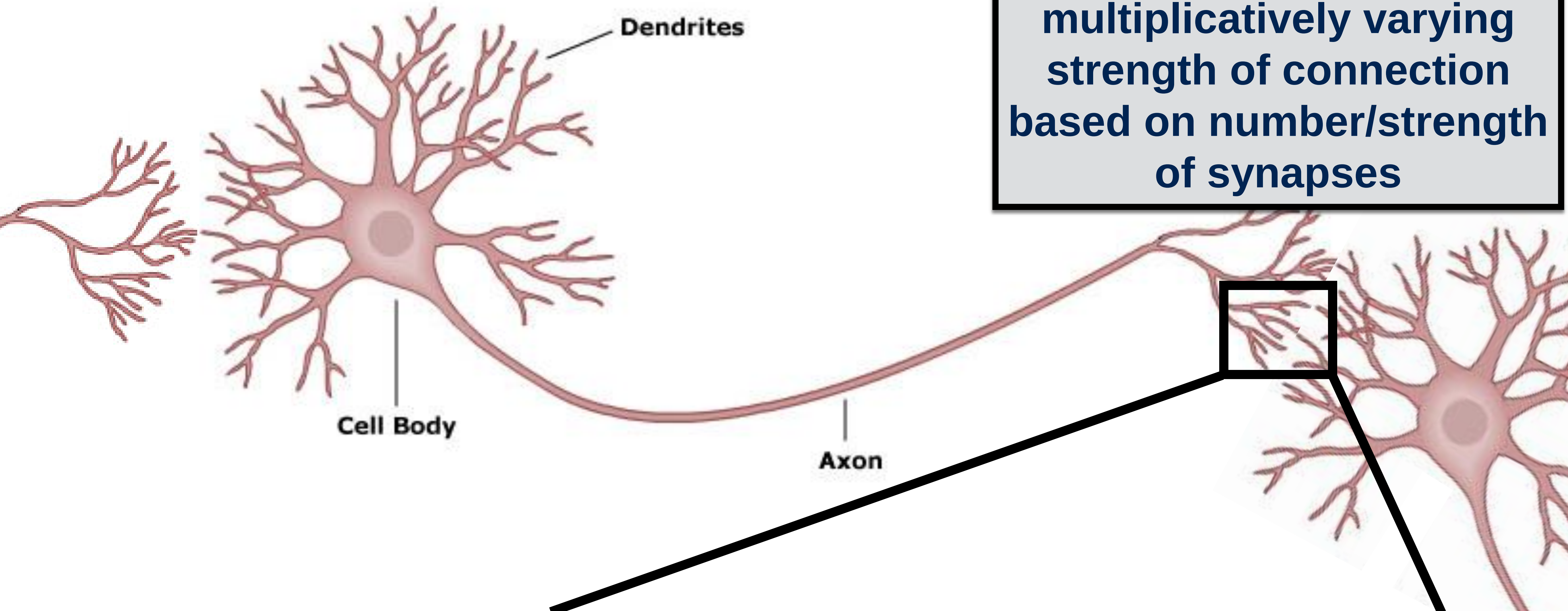


arrival of action potentials at synapse causes release of neurotransmitters to post-synaptic neuron

more and/or stronger synapses mean pre-synaptic neuron has more influence over post-synaptic neuron

synapses multiply effect of action potentials

multiplicatively varying strength of connection based on number/strength of synapses



arrival of action potentials at synapse causes release of neurotransmitters to post-synaptic neuron

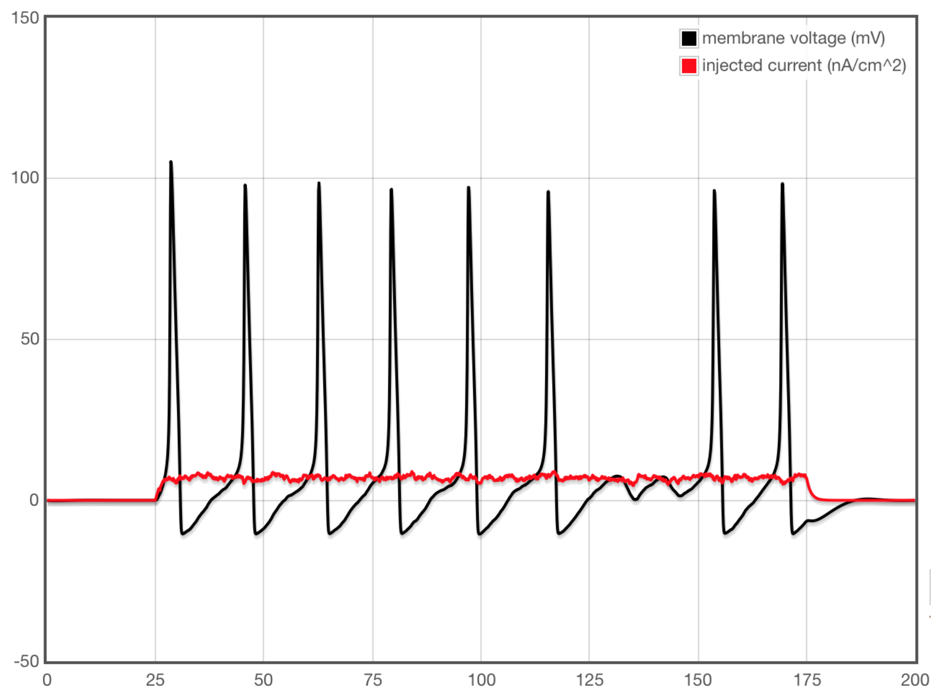
more and/or stronger synapses mean pre-synaptic neuron has more influence over post-synaptic neuron

synapses <u>multiply</u> effect of action potentials

Levels of Neuron Modeling

squid giant axon

- detailed model of action potentials (Hodgkin-Huxley model)



$$\frac{dV}{dt} = \frac{1}{C}(-I_{NA} - I_K - I_{leak} - I_{input})$$

$$\frac{dV}{dt} = \frac{1}{C}\left(-g_{NA}m^3h(V - E_{NA}) - g_K n^4(V - E_K) - g_{leak}(V - E_{leak}) + I_{input}\right)$$

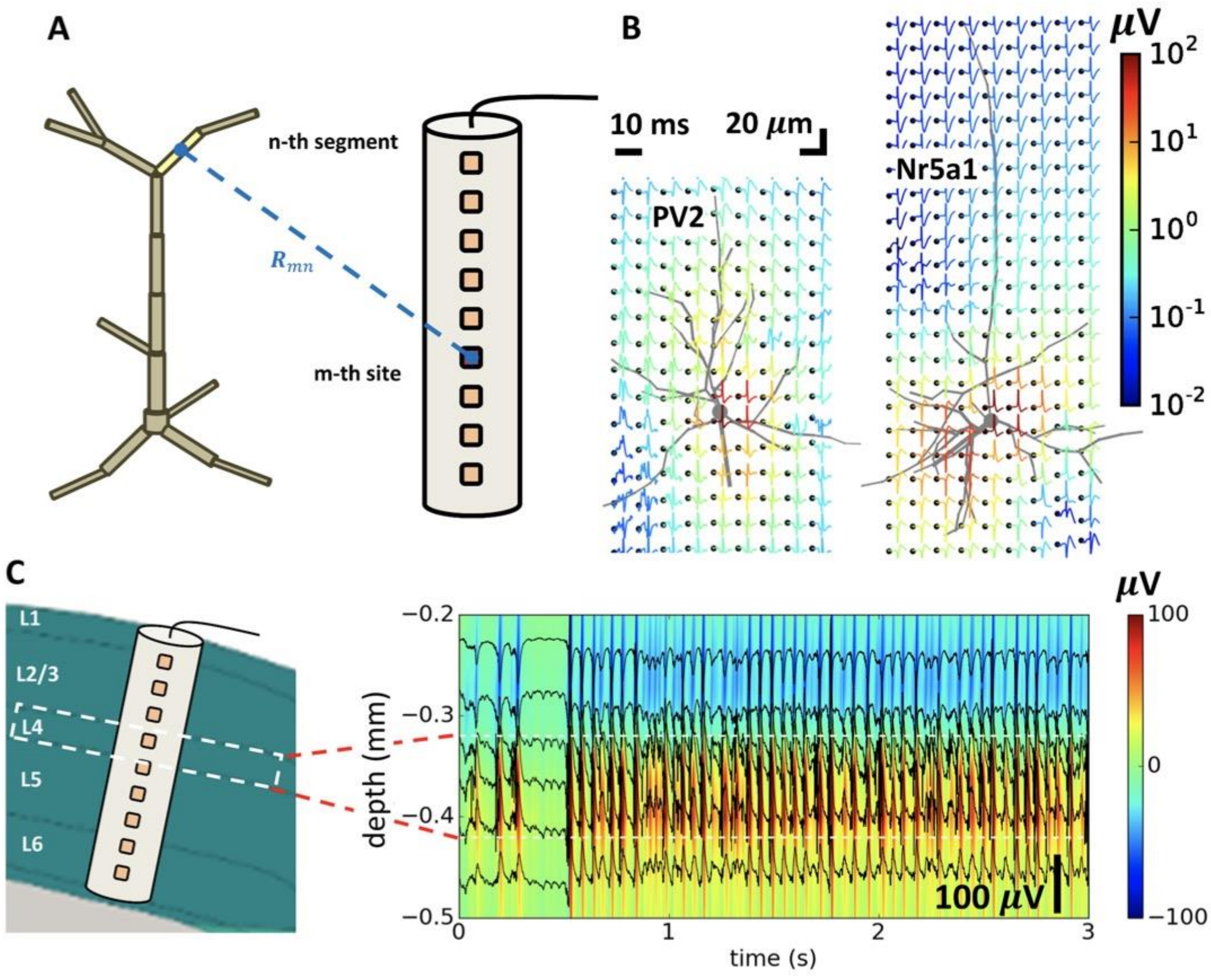
$$\frac{dm}{dt} = \frac{1}{\tau_m(V)}(-m + M(V))$$

$$\frac{dh}{dt} = \frac{1}{\tau_h(V)}(-h + H(V))$$

$$\frac{dn}{dt} = \frac{1}{\tau_n(V)}(-n + N(V))$$

Levels of Neuron Modeling

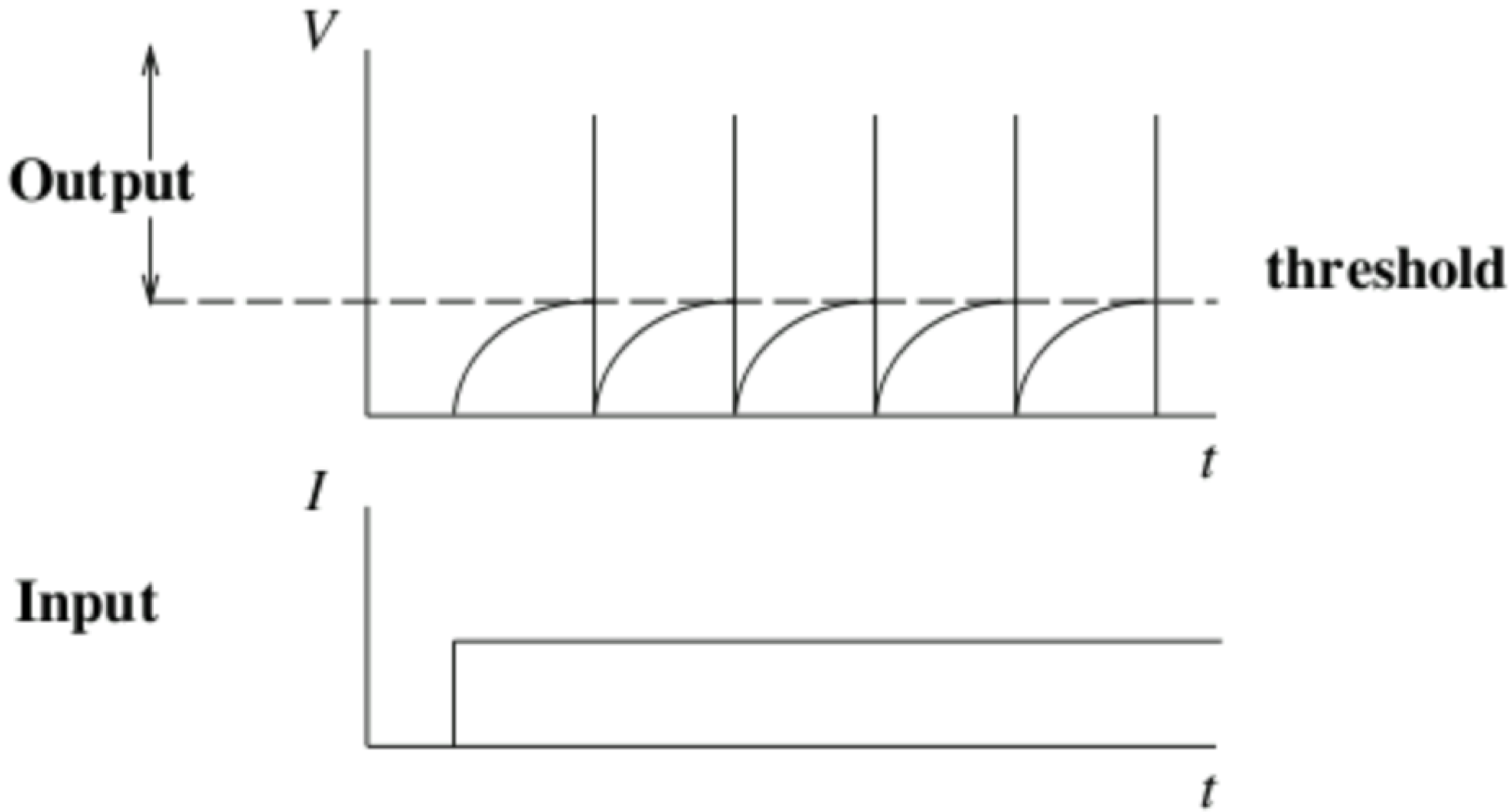
- more detailed model of neuron morphology (NEURON simulator)



Levels of Neuron Modeling

- detailed model of action potentials (Hodgkin-Huxley model)

- more abstract integrate-and-fire (spiking) neuron

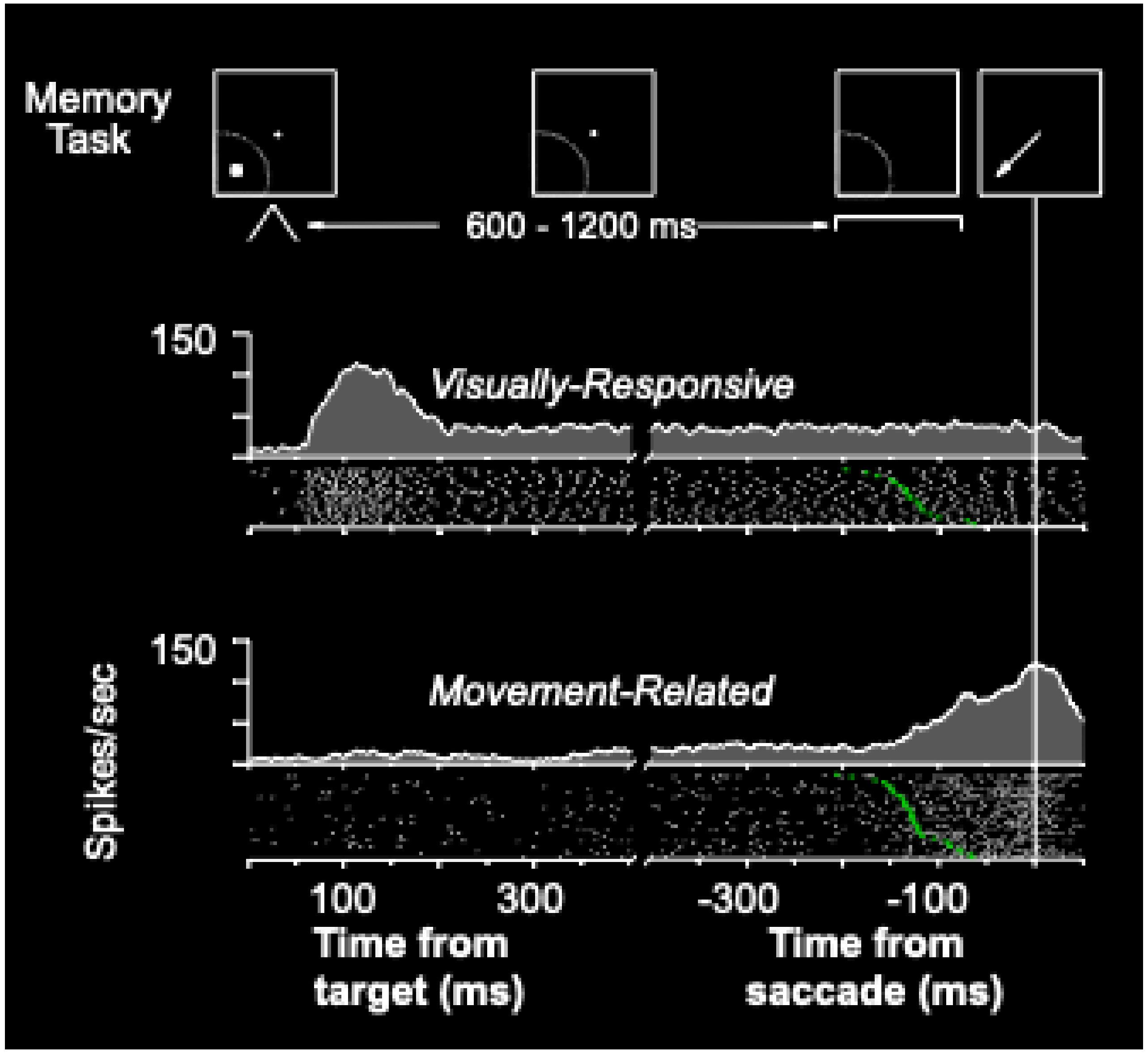


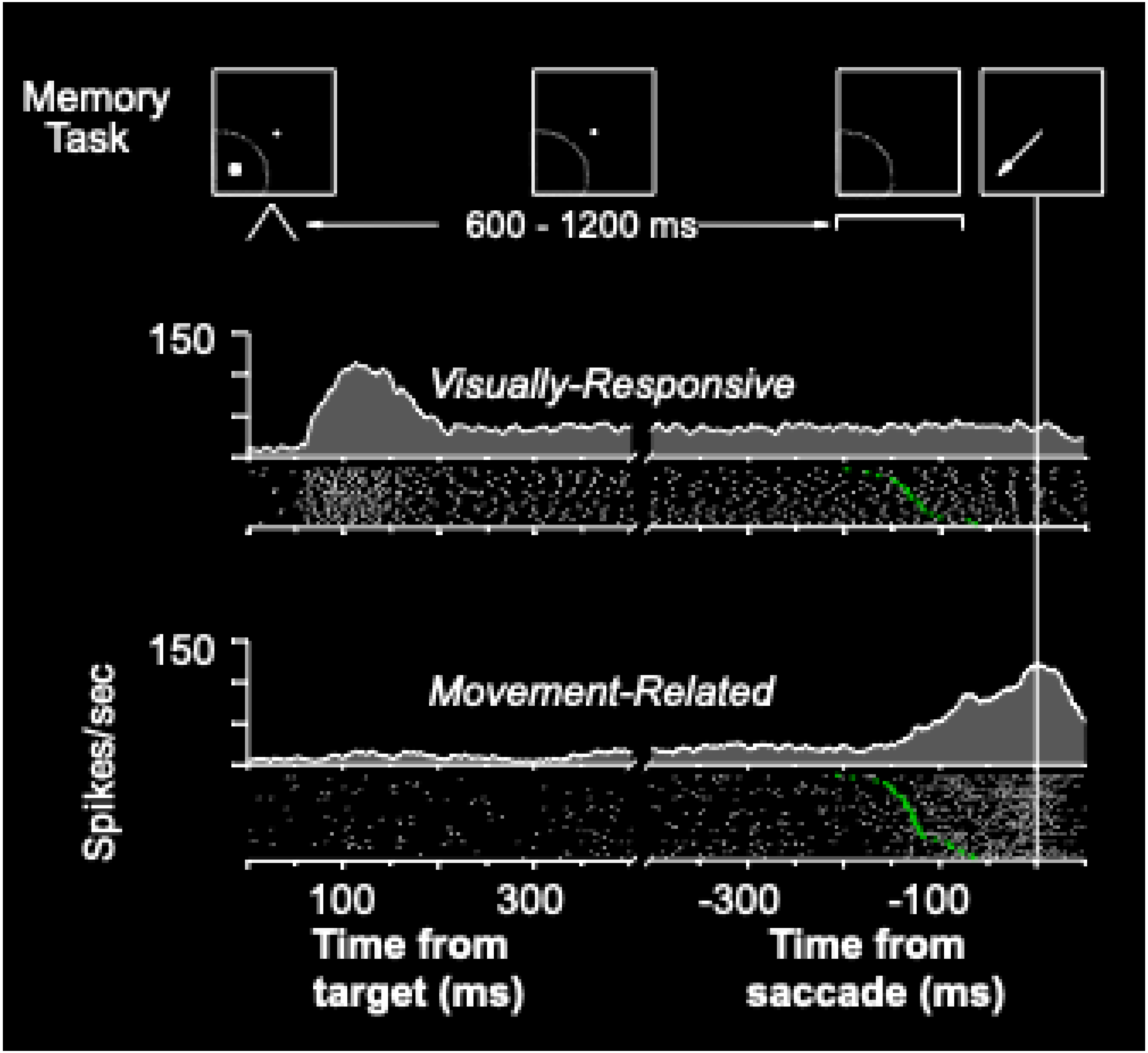
Levels of Neuron Modeling

- detailed model of action potentials (Hodgkin-Huxley model)

- more abstract integrate-and-fire (spiking) neuron

- rate-coded neuron




Memory Task
600 - 1200 ms
150
Visually-Responsive
150
Movement-Related
Spikes/sec
100
300
-300
-100
Time from target (ms)
Time from saccade (ms)

Levels of Neuron Modeling

- detailed model of action potentials (Hodgkin-Huxley model)

- more abstract integrate-and-fire (spiking) neuron

- rate-coded neuron

- activation (positive or negative) (collection of neurons)
  what we'll be focusing on

Levels of Neuron Modeling

- detailed model of action potentials (Hodgkin-Huxley model)

- more abstract integrate-and-fire (spiking) neuron

- rate-coded neuron

- activation (positive or negative) (collection of neurons)

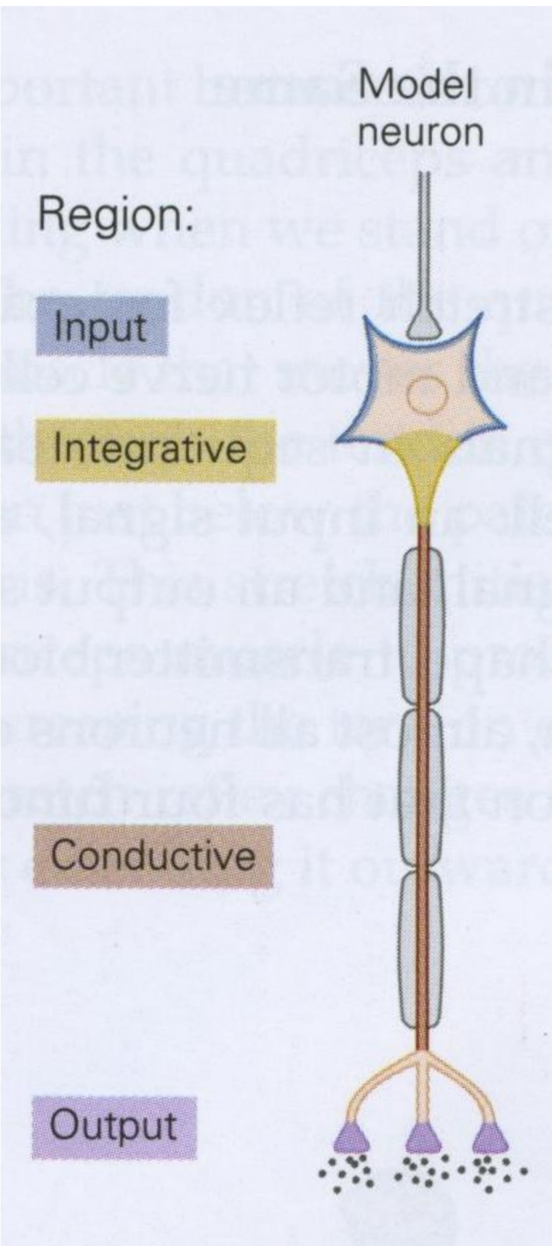
we'll start by considering neural network models that ignore spikes, and treat time in a “chunky” way

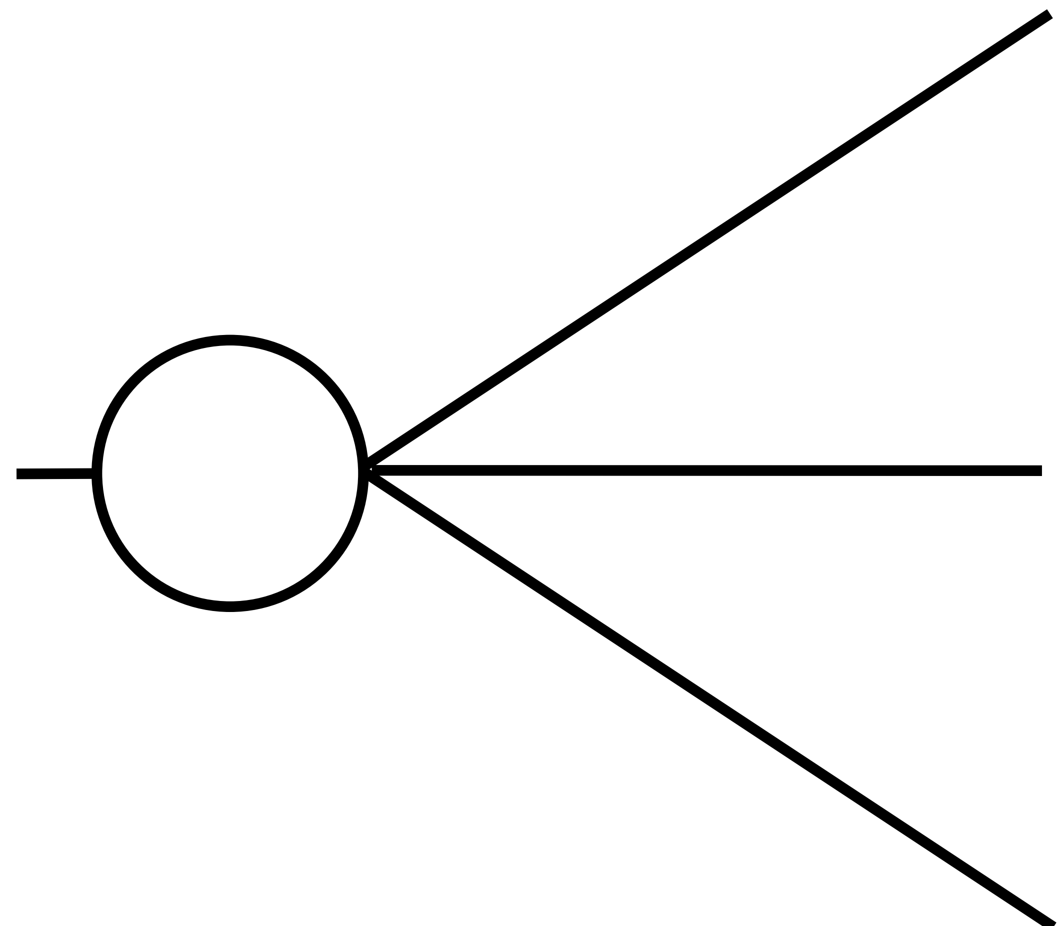
then consider models that learn, networks that change over time with experience

may consider models that allow for dynamics (recurrent networks, dynamical networks, spiking)

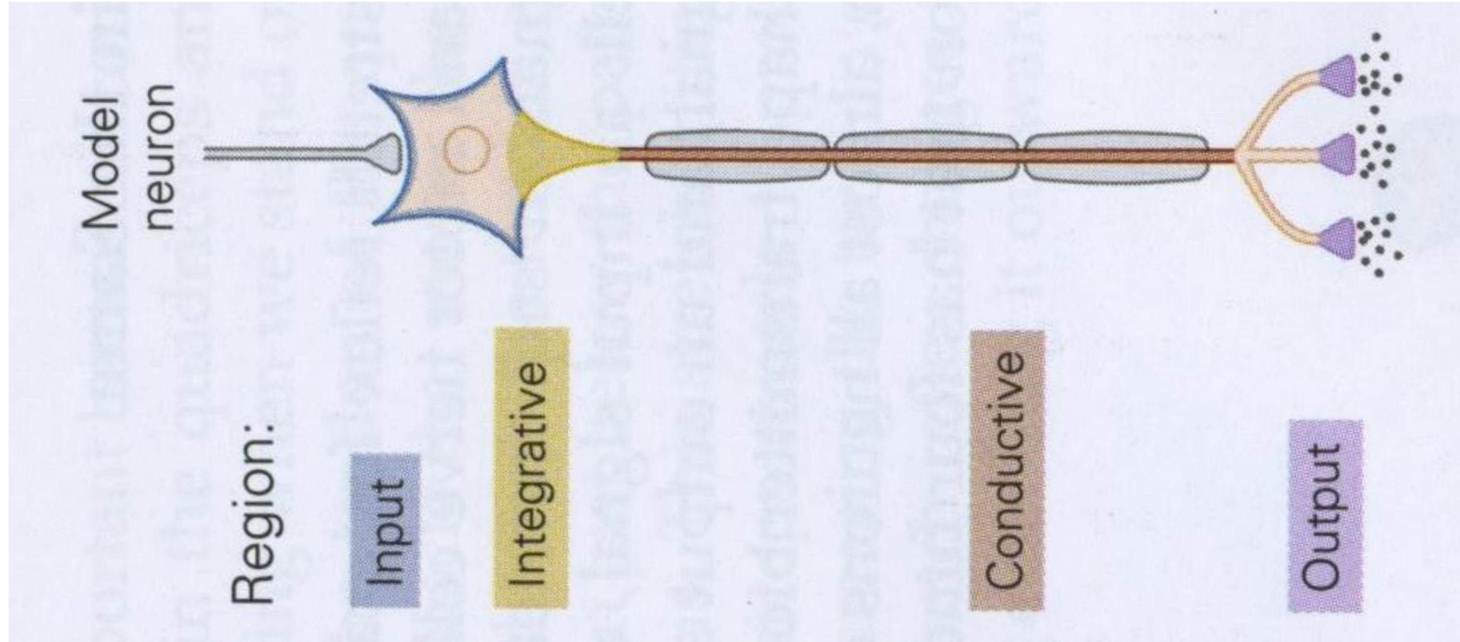
Introduction to Neural Networks

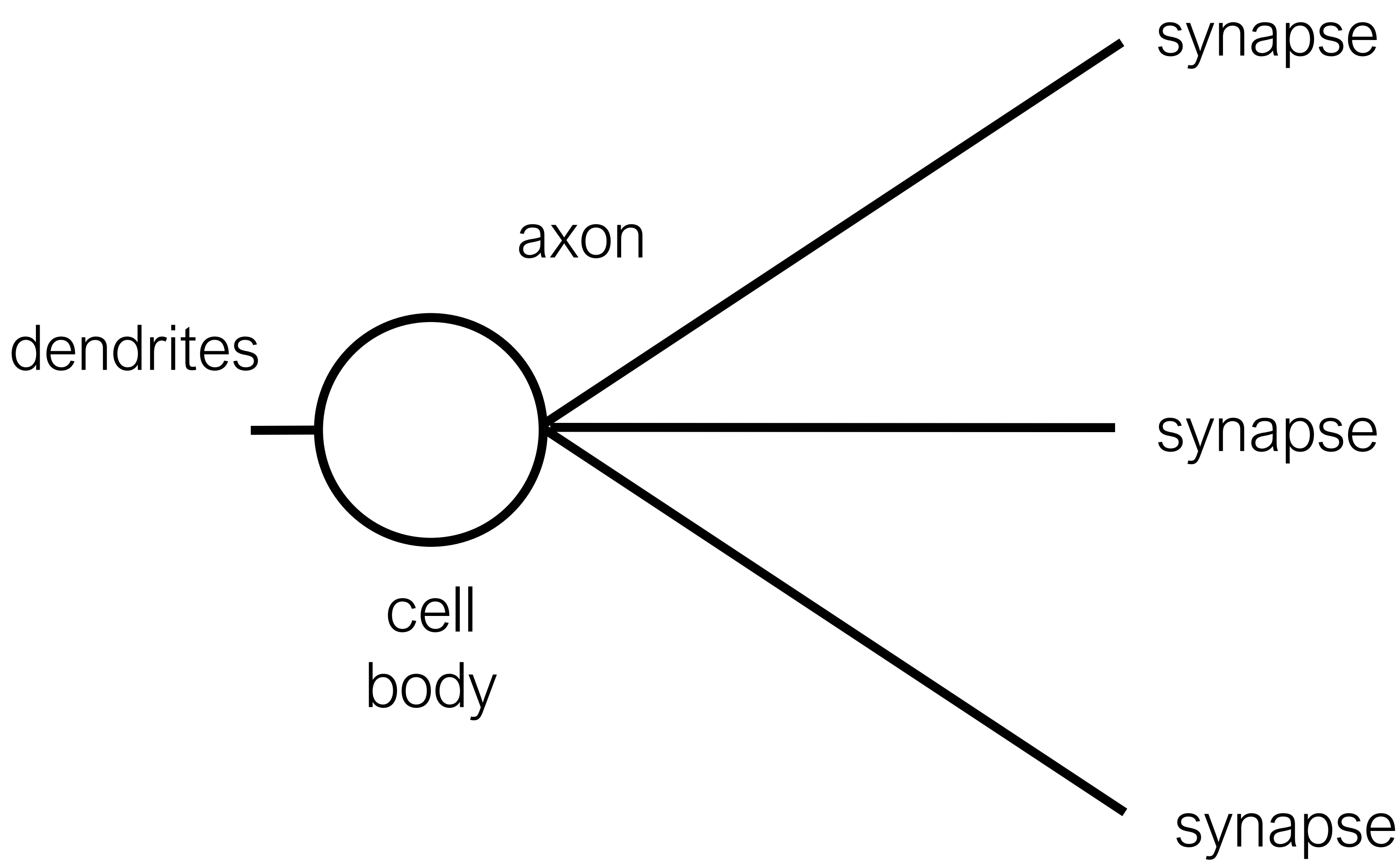
modeling an idealized neuron



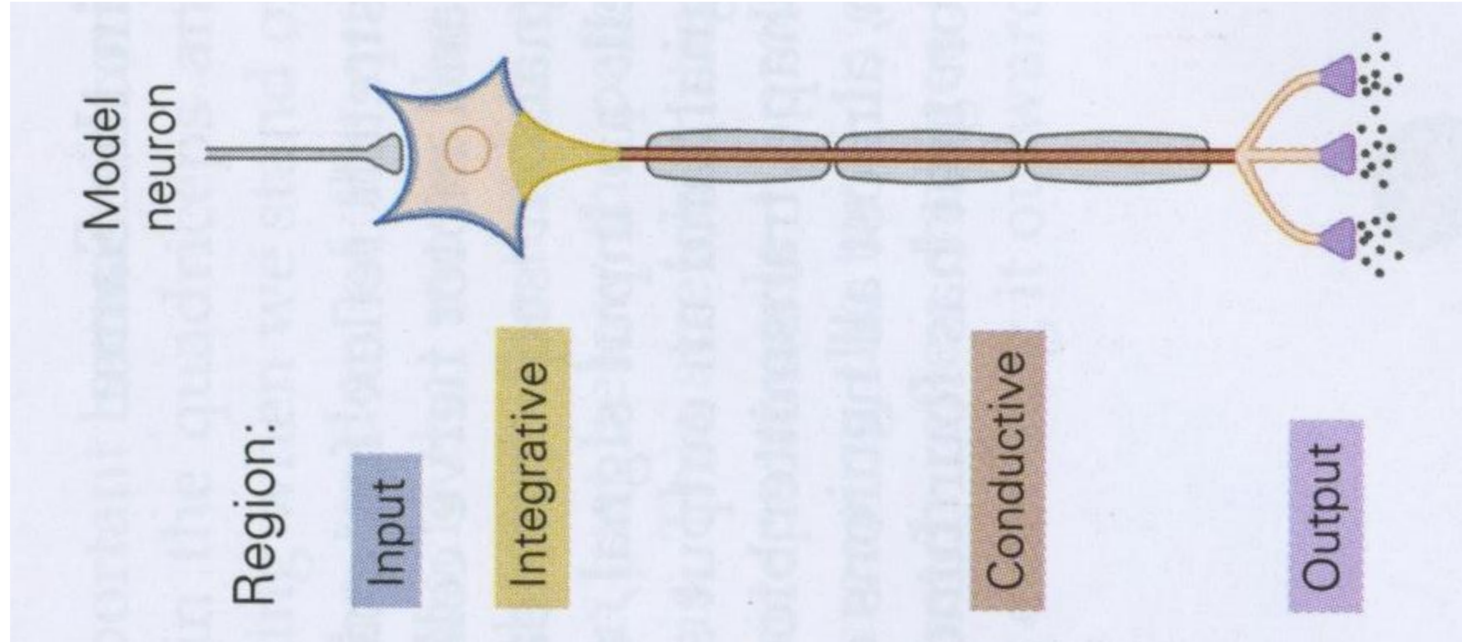


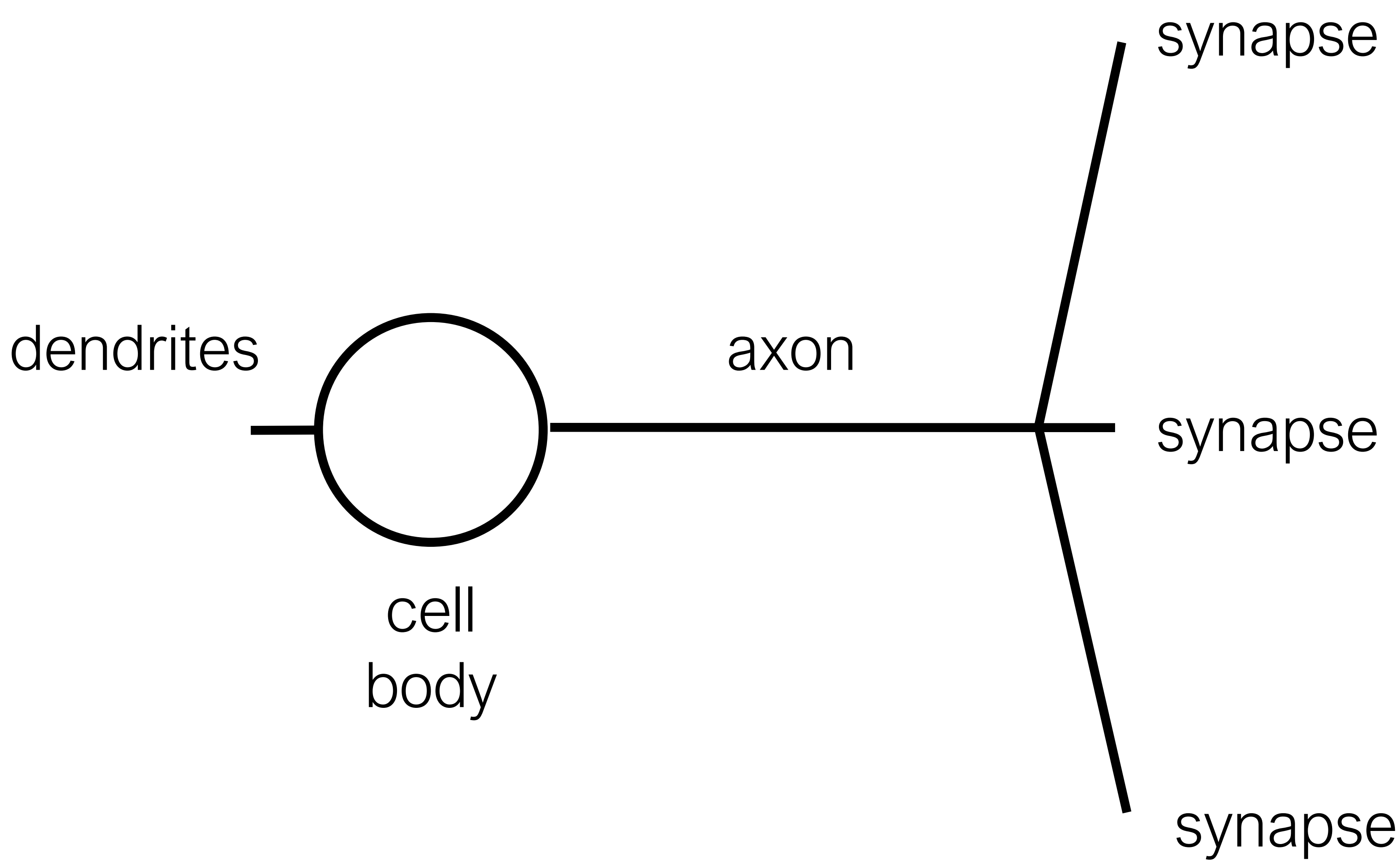
Idealized Neuron



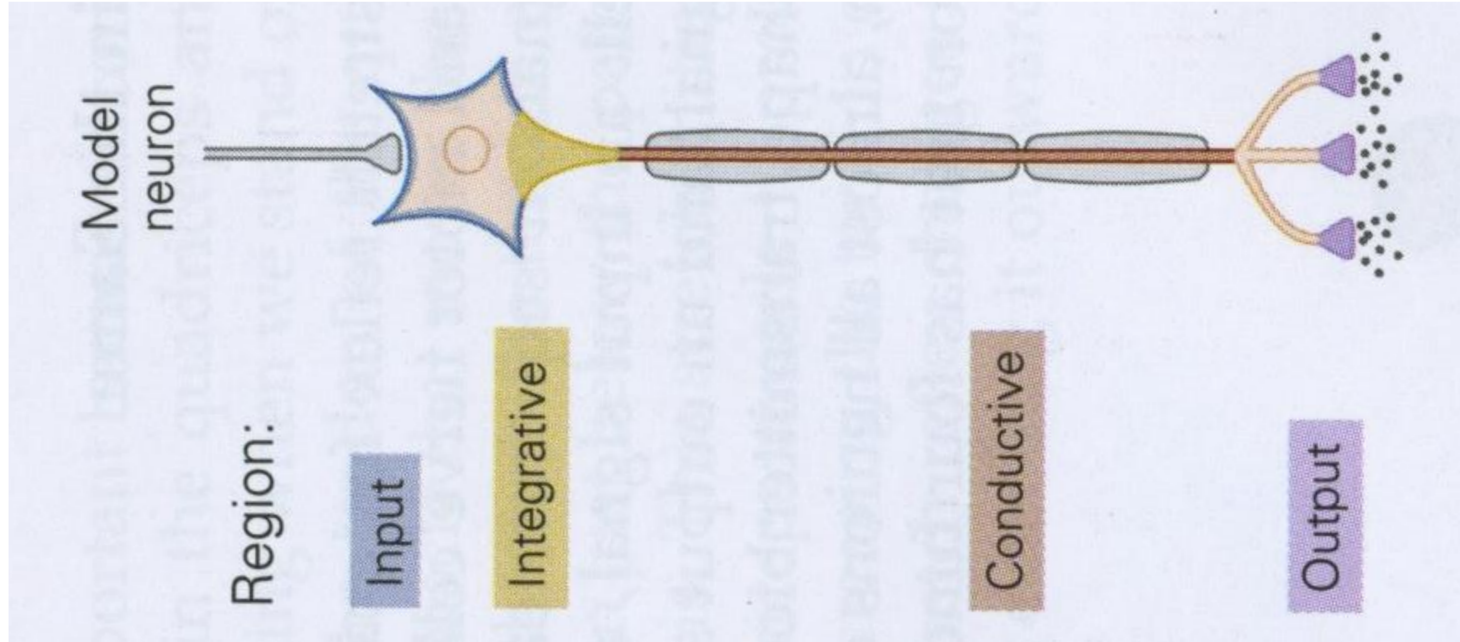


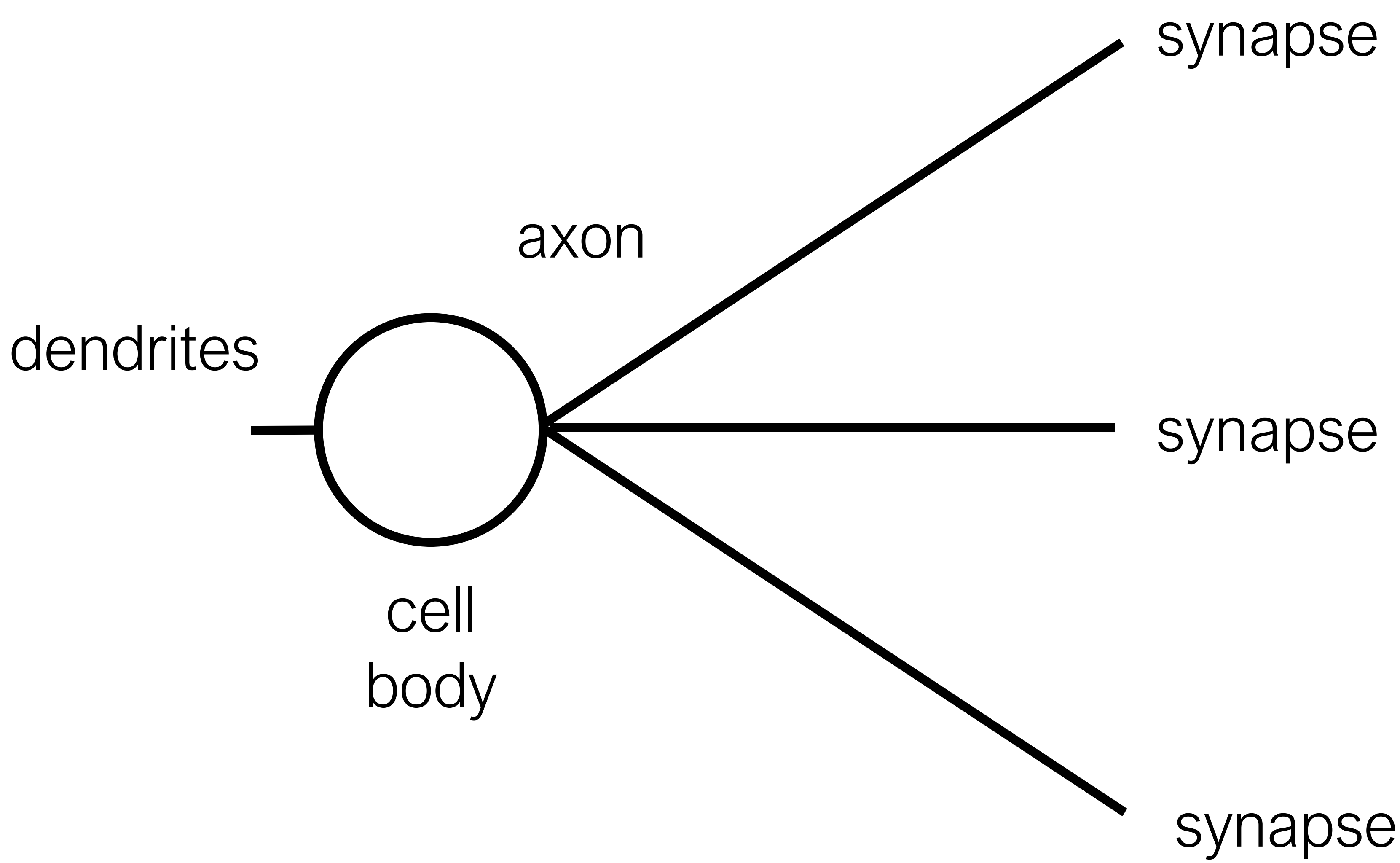
Idealized Neuron





Idealized Neuron

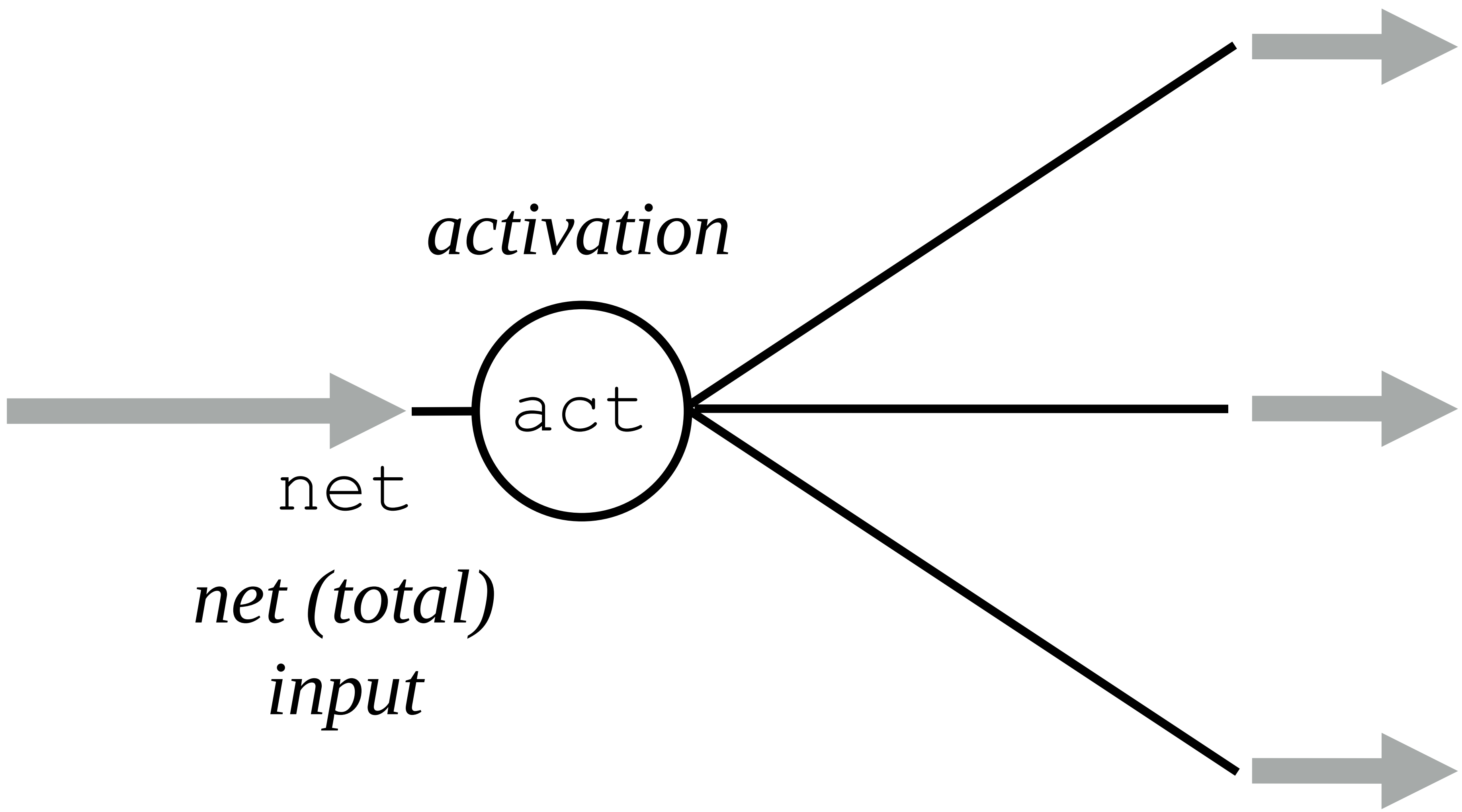




Idealized Neuron

"unit" or "node"

how is the activation of a *neuron* *related to its net input?*



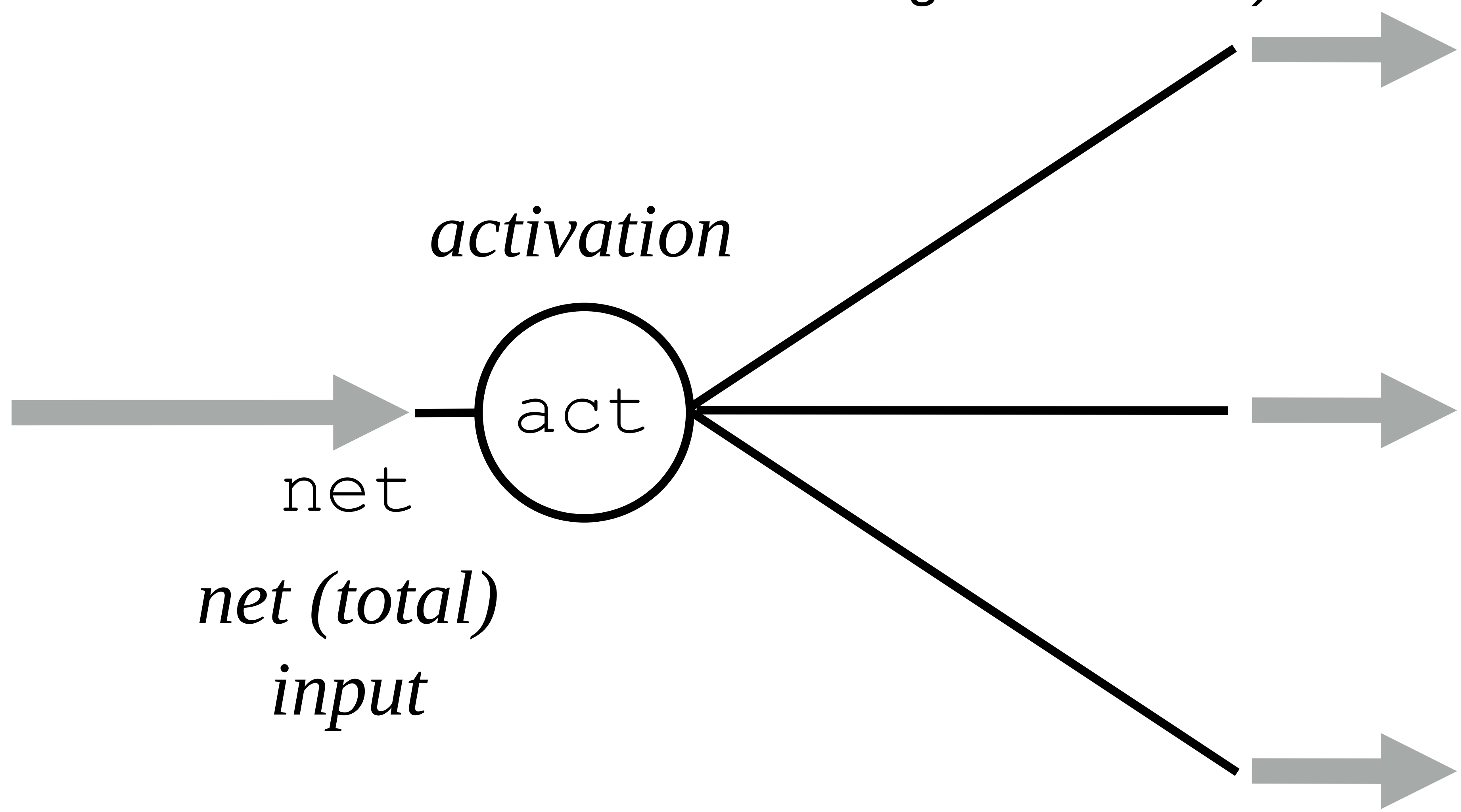
Idealized Neuron

"unit" or "node"

how is the activation of a neuron related to its net input?

where "net input" comes from other neurons (and their synapses)
(where net input can be positive or negative)

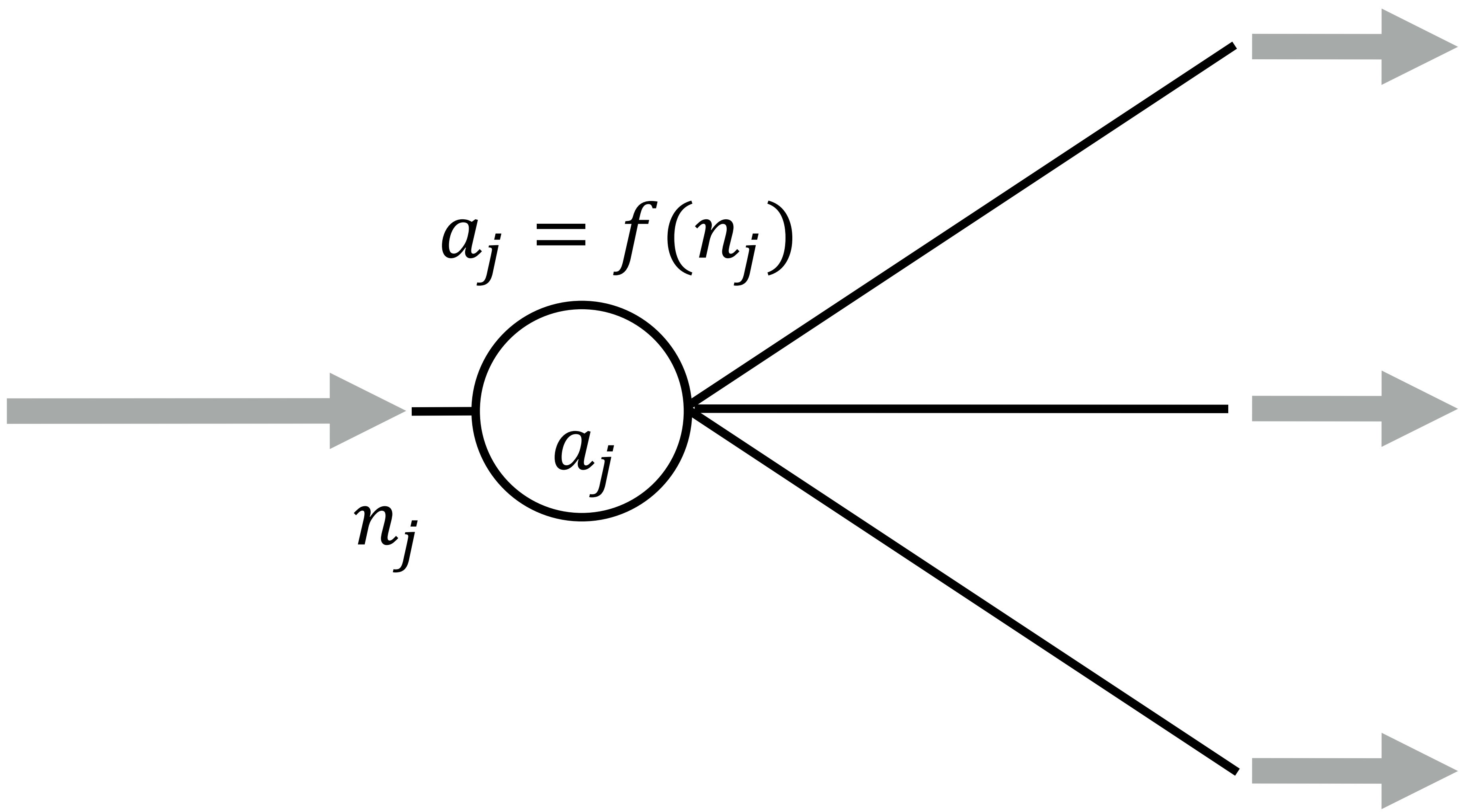
where "activation" is like spike rate
(but some activation functions allow negative values)



Idealized Neuron

a variety of activation functions are used
to model neurons in neural networks

because there are always multiple neurons (units/nodes)
in a neural network we will commonly use subscripts



Idealized Neuron

Keras, Tensorflow, and Python (plus toolkits)

Python
some programming will be
in base Python (with packages like
numpy, matplotlib, etc.)

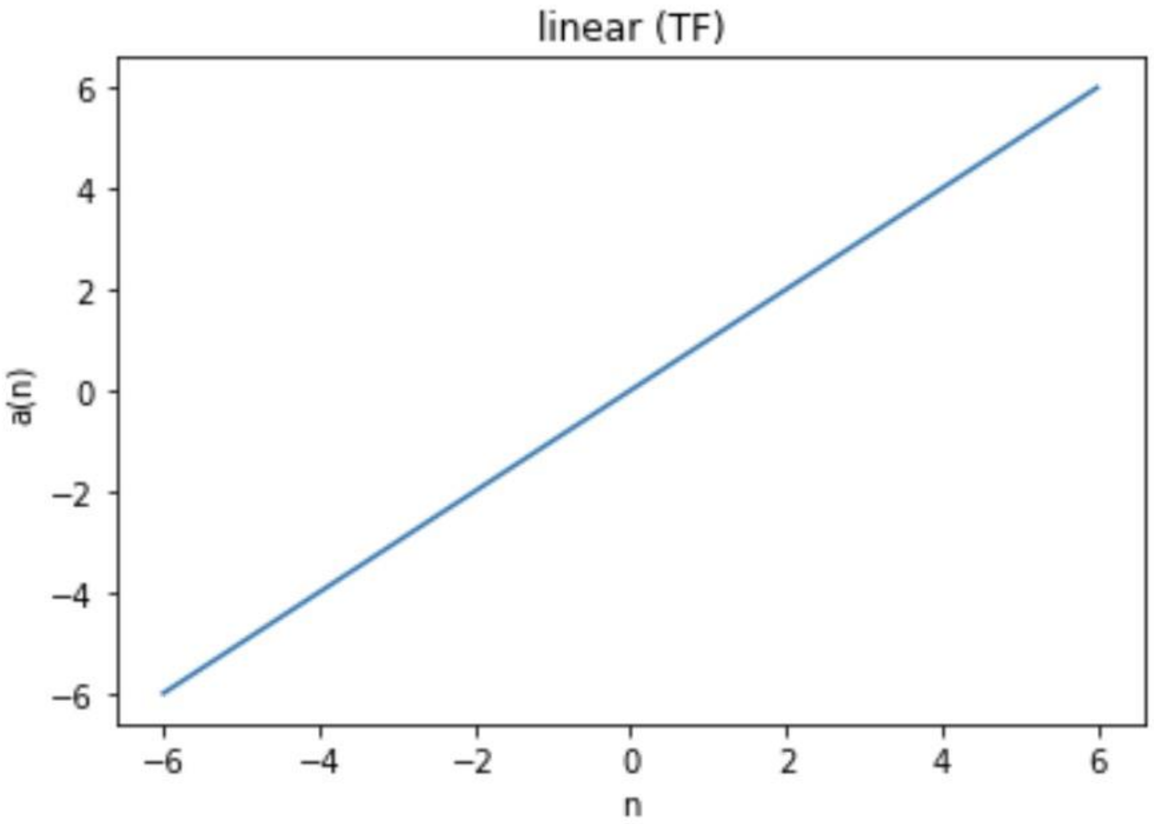
Keras, Tensorflow, and Python (plus toolkits)

Python
some programming will be in base Python (with packages like numpy, matplotlib, etc.)

Keras
high-level Python package for simulating neural networks

Tensorflow
high-level Python package for simulating neural networks

CPUs, GPUs, TPUs

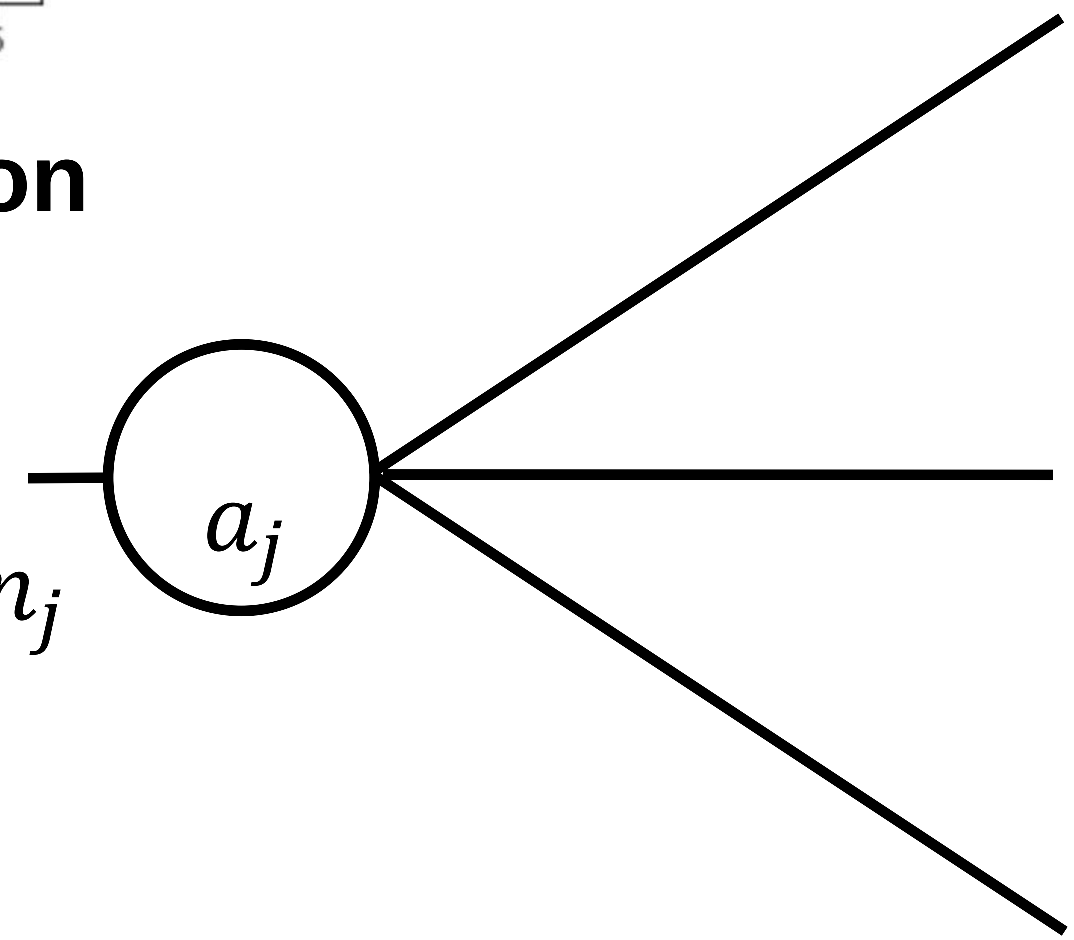


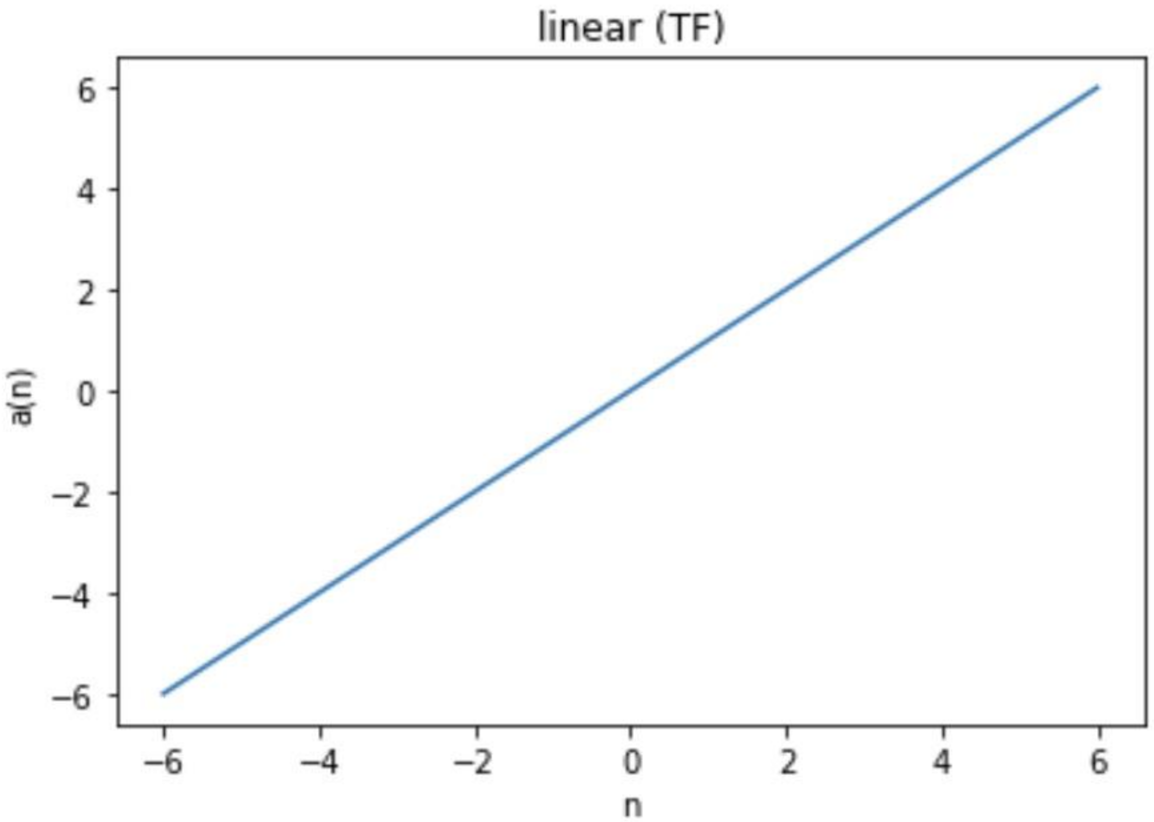
used as output activation
when modeling "regression"
problems

not used as activation
for "hidden" units
(we will discuss later)

linear activation function

$$a_j(n_j) = n_j$$





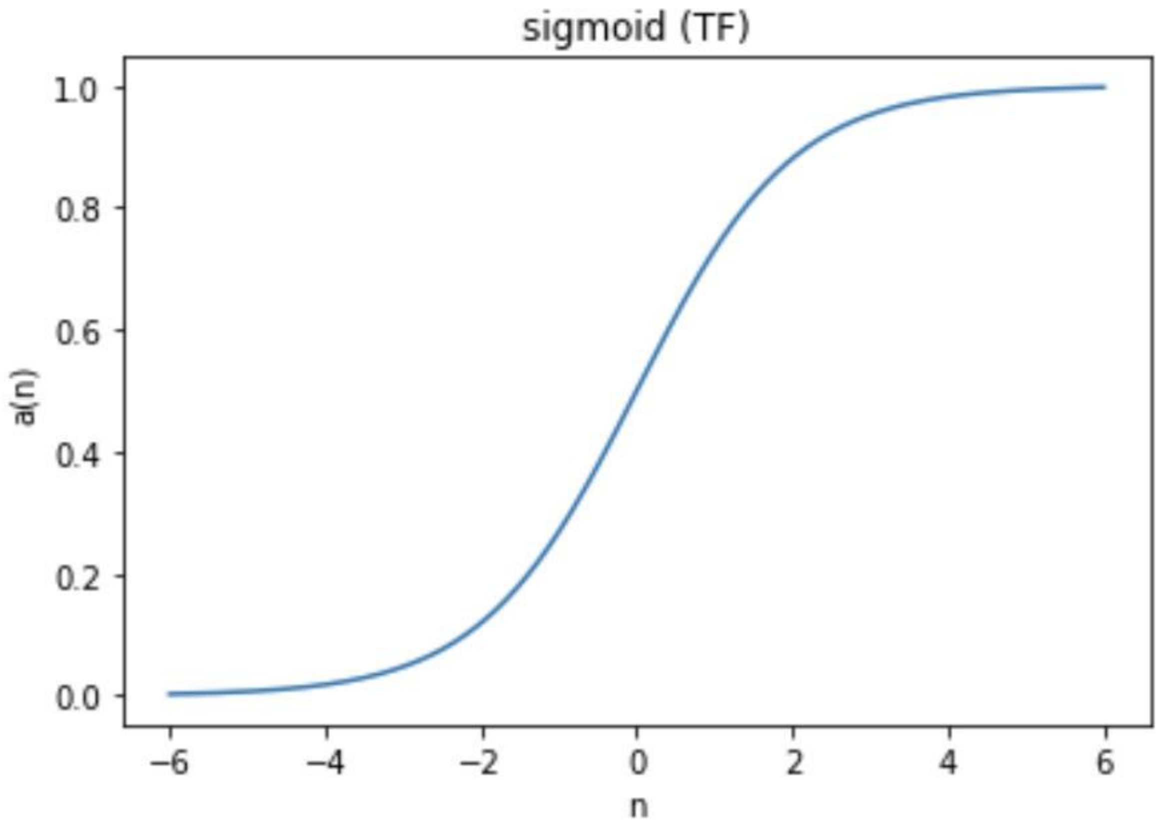
used as output activation
when modeling "regression"
problems

not used as activation
for "hidden" units
(we will discuss later)

linear activation function

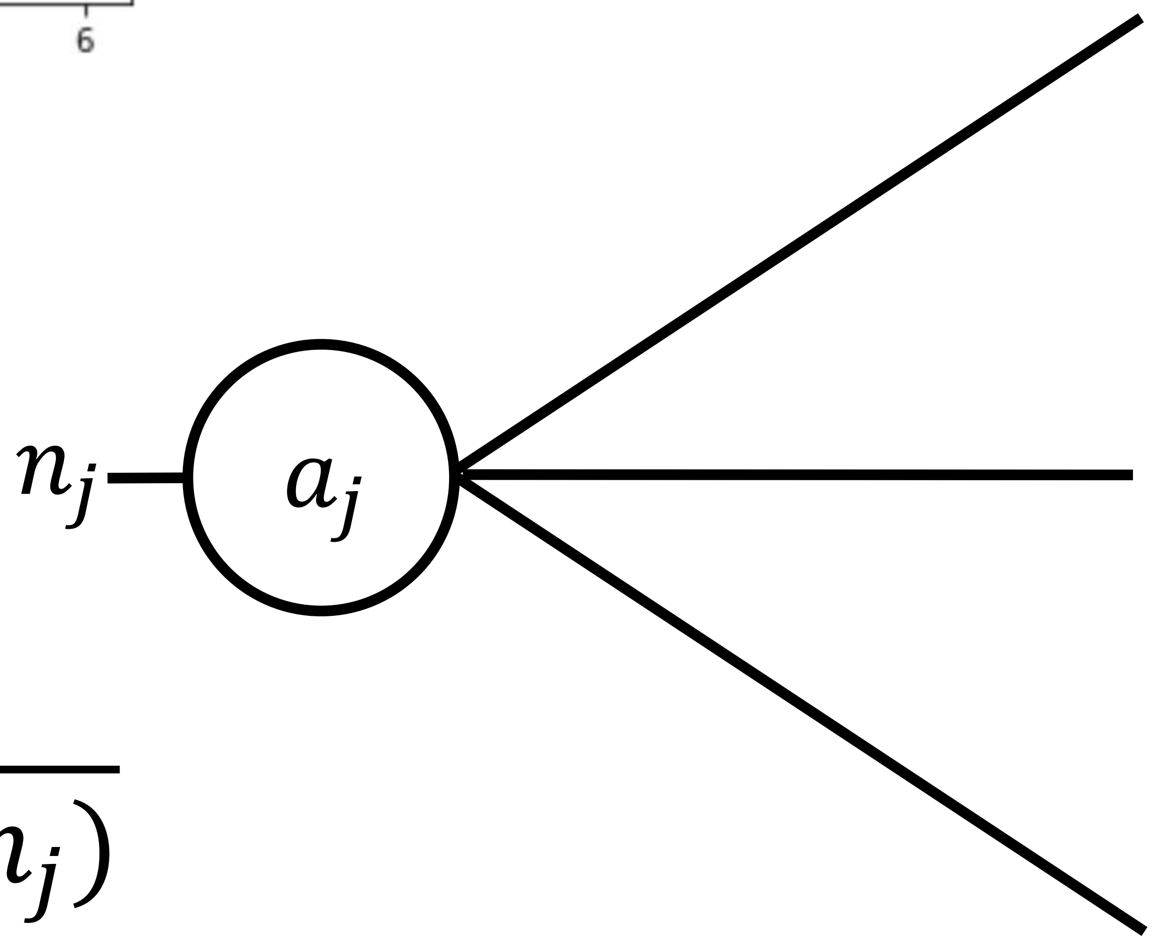
```
import numpy as np
import tensorflow as tf

n = np.linspace(-6, 6, 1000)
a = tf.keras.activations.linear(n)

my_singleplot(n, a, 'linear (TF)')
```

```
a = tf.keras.activations.linear(n)
```



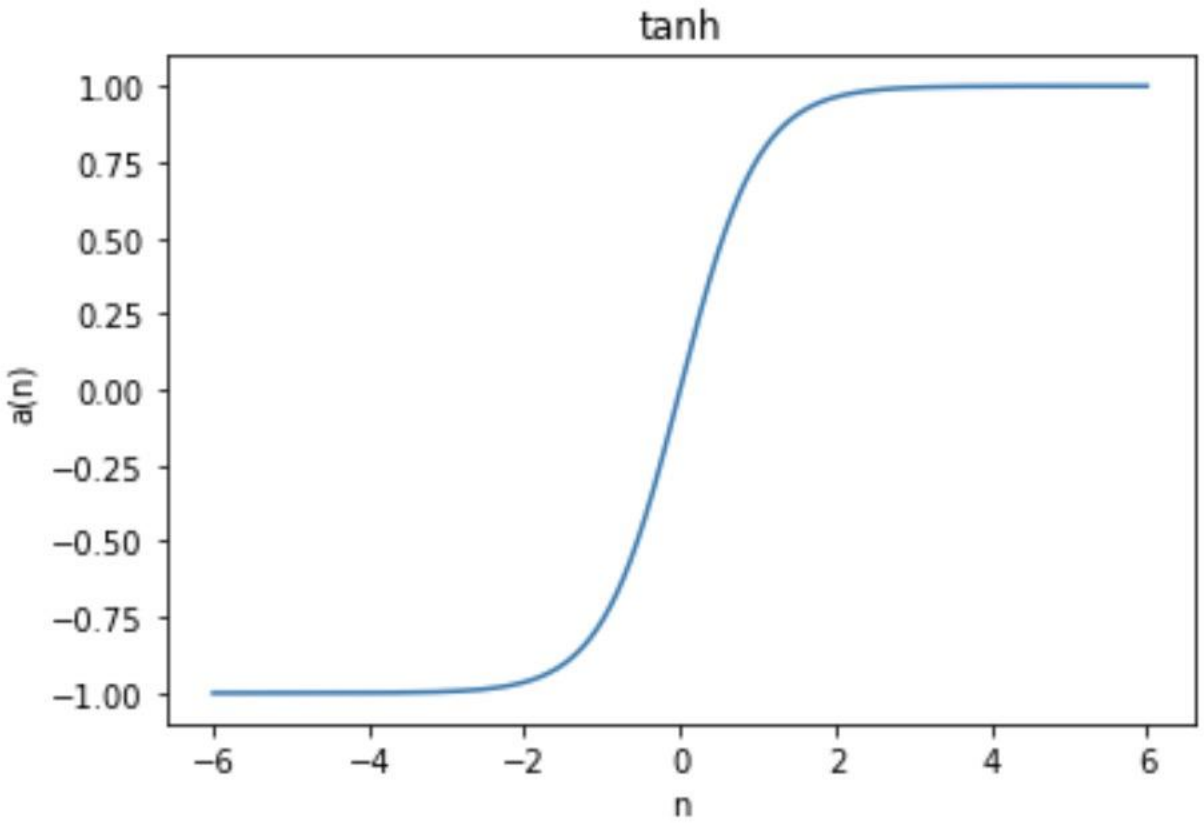
neurons have a nonlinear response function

sigmoid (logistic)



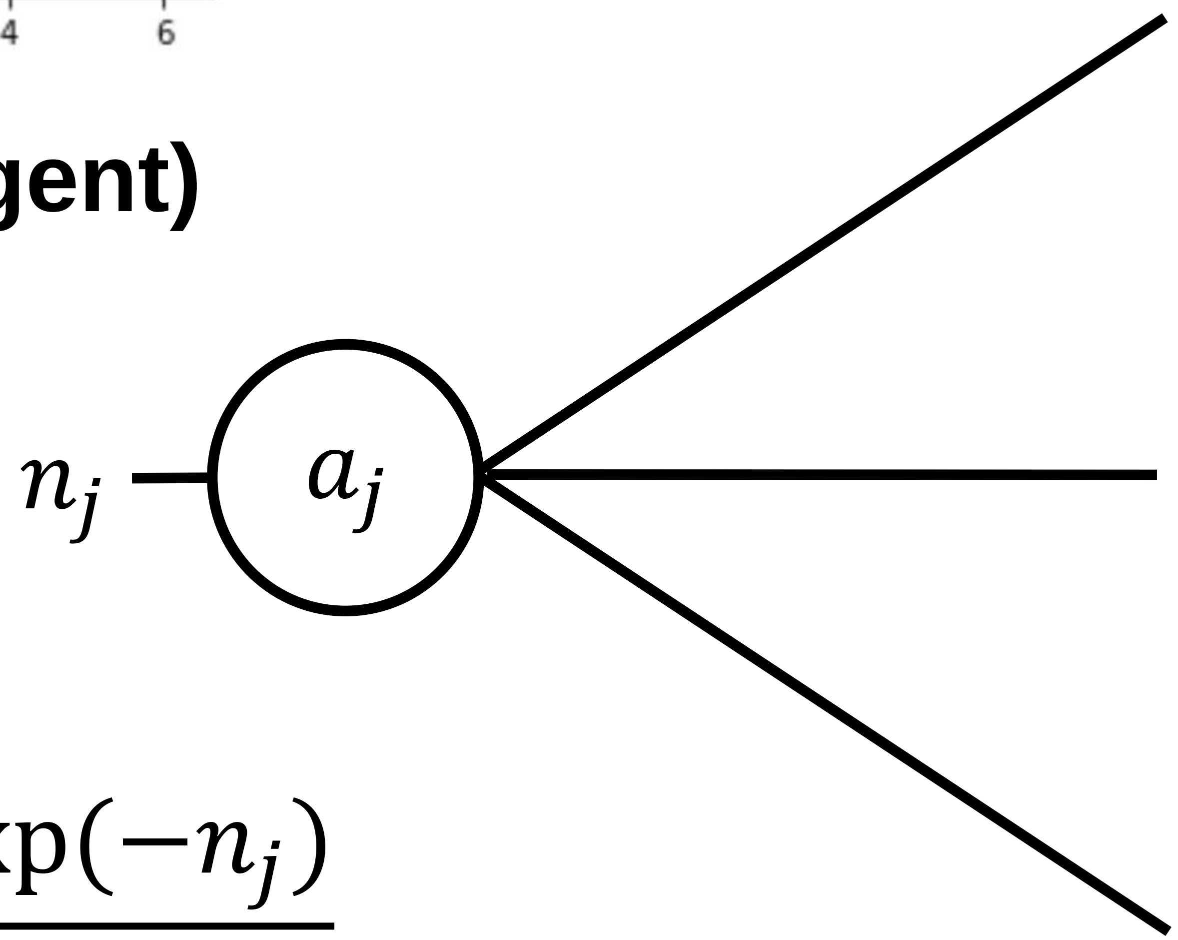
$$a_j(n_j) = \frac{1}{1 + \exp(-n_j)}$$

```
a = tf.keras.activations.sigmoid(n)
```



neurons have a nonlinear response function

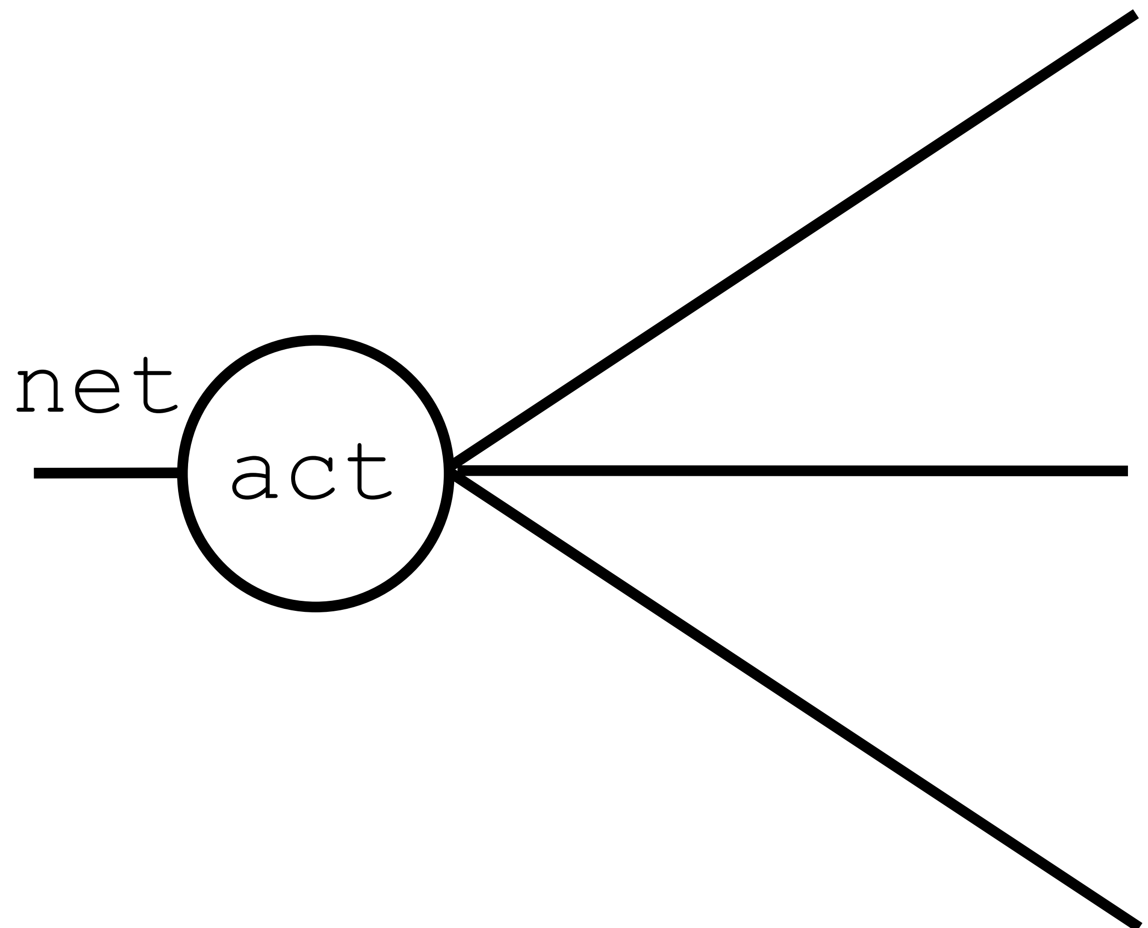
tanh (hyperbolic tangent)



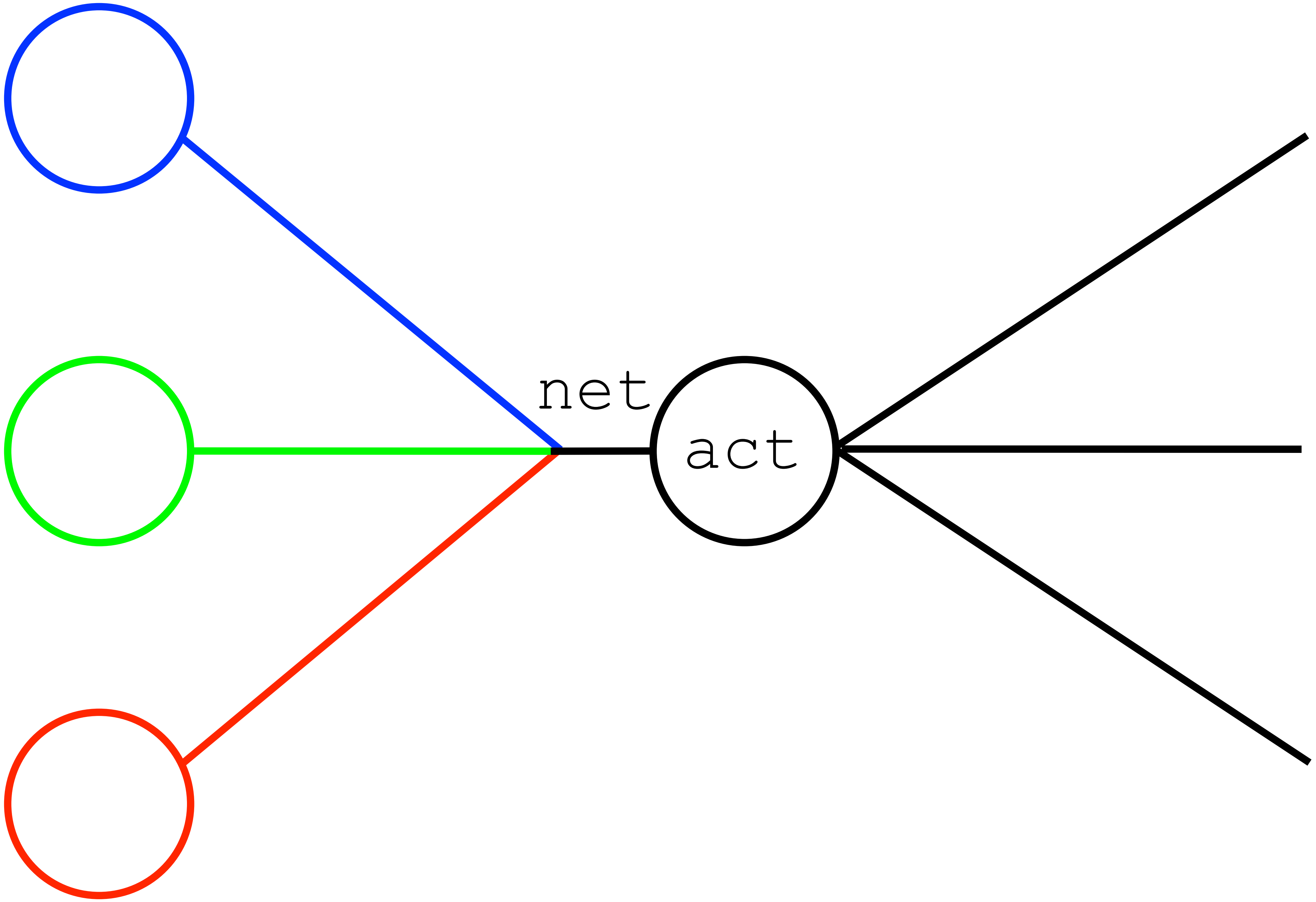
$$a_j(n_j) = \frac{\exp(n_j) - \exp(-n_j)}{\exp(n_j) + \exp(-n_j)}$$

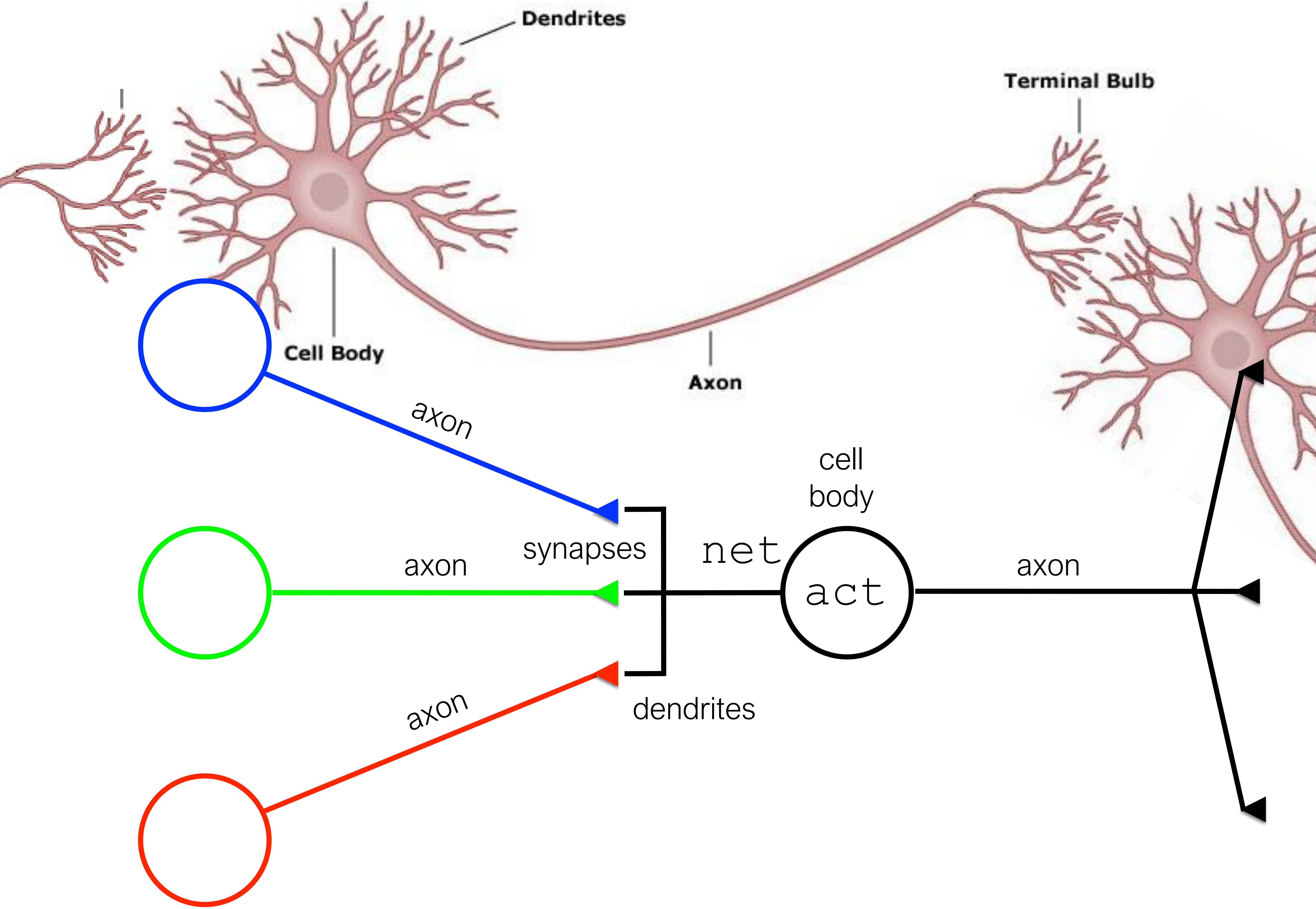
```
a = tf.keras.activations.tanh(n)
```

where does the net input come from?

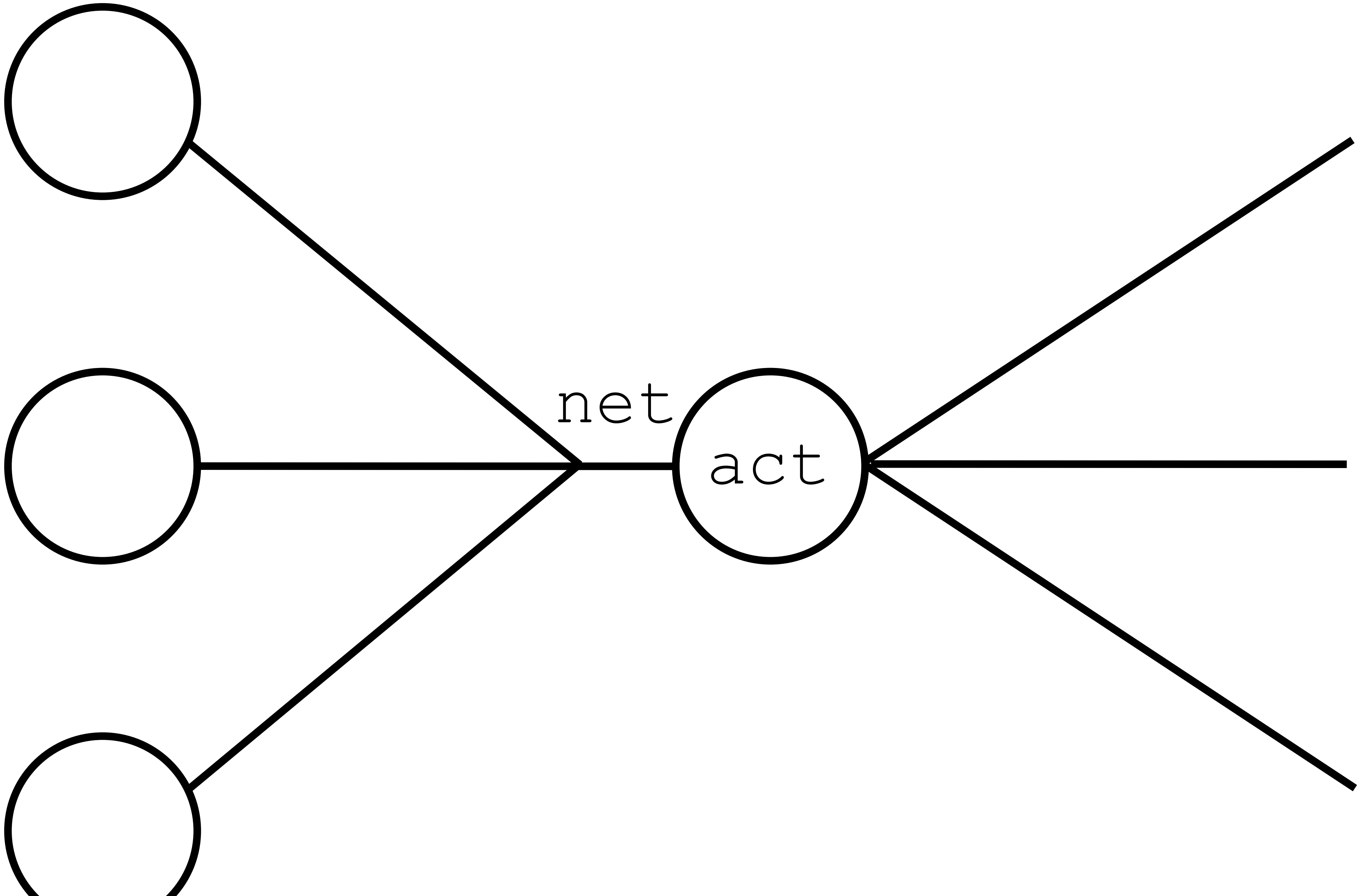


where does the net input come from?

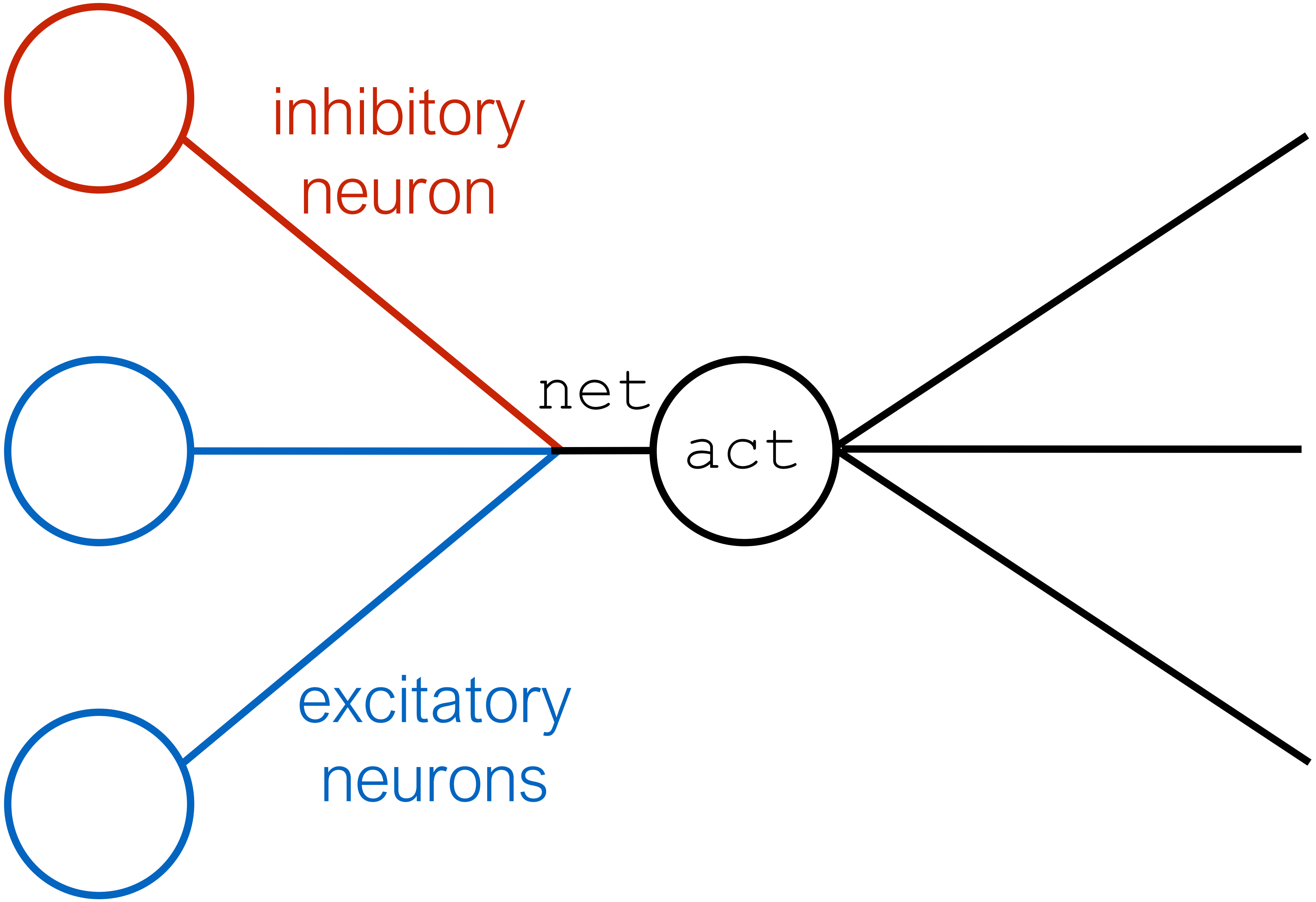



Dendrites
Terminal Bulb
Cell Body
Axon
axon
cell
body
synapses
net
act
axon
axon
axon
dendrites

where does the net input come from?



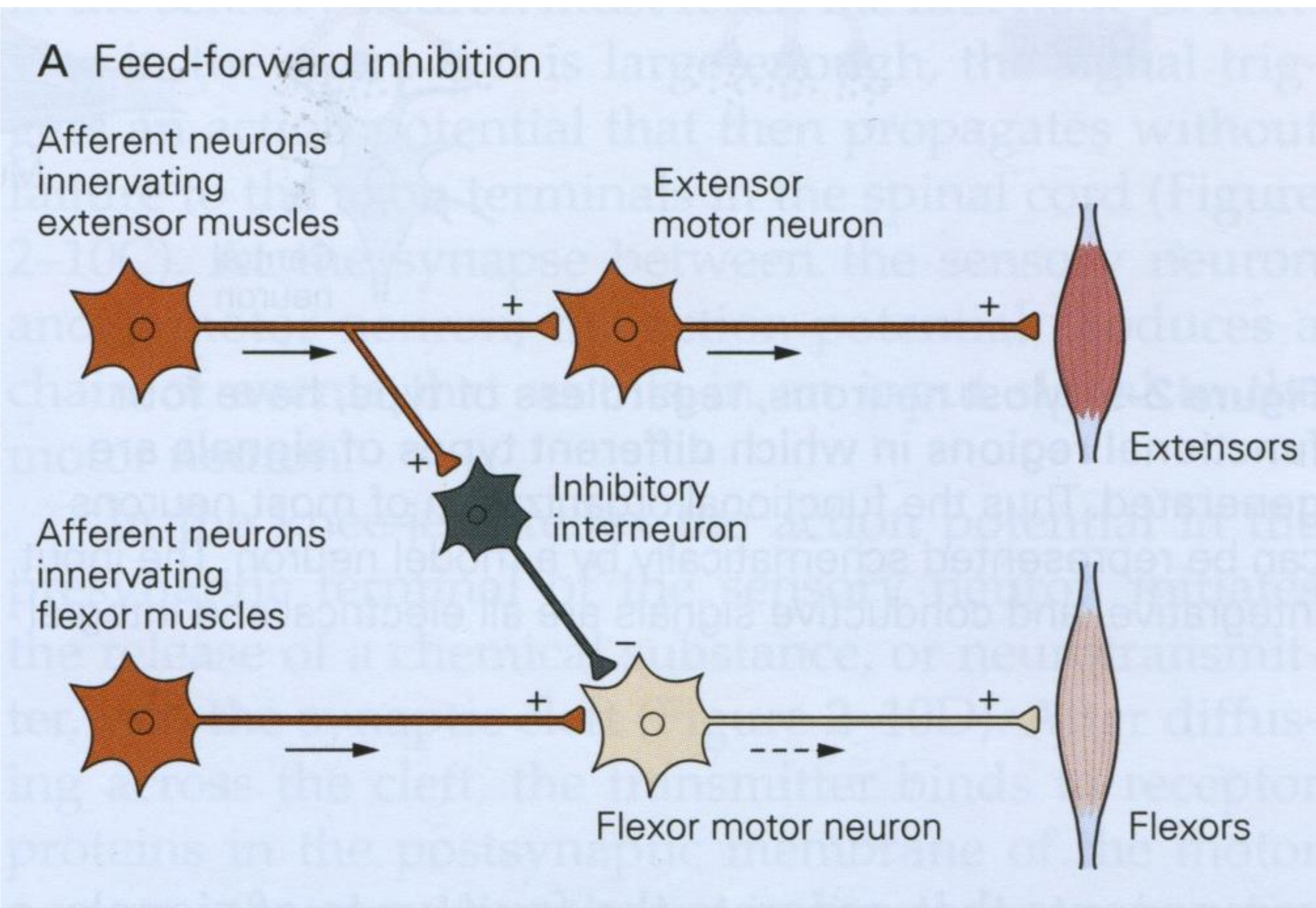
inputs can be positive (+) or negative(-)



while individual neurons are either excitatory or inhibitory, inhibitory interneurons can make an excitatory neuron inhibit indirectly

inputs can be positive (+) or negative(-)

so for simplicity, we allow the same unit to be either excitatory or inhibitory

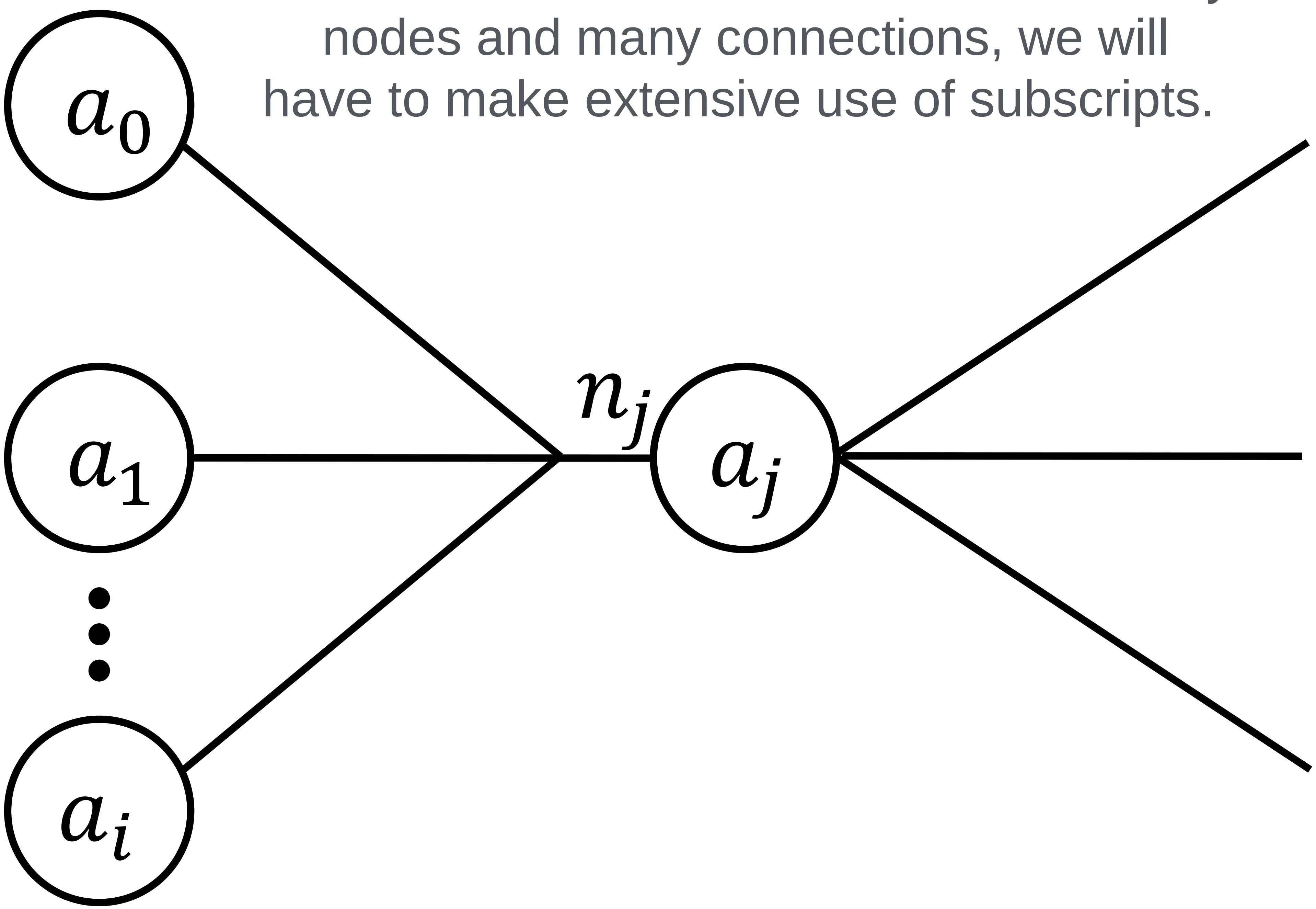


Two different implementations
Whether or not you allow the same unit to be excitatory and inhibitory.

This may not affect the kinds of computations a network can perform.

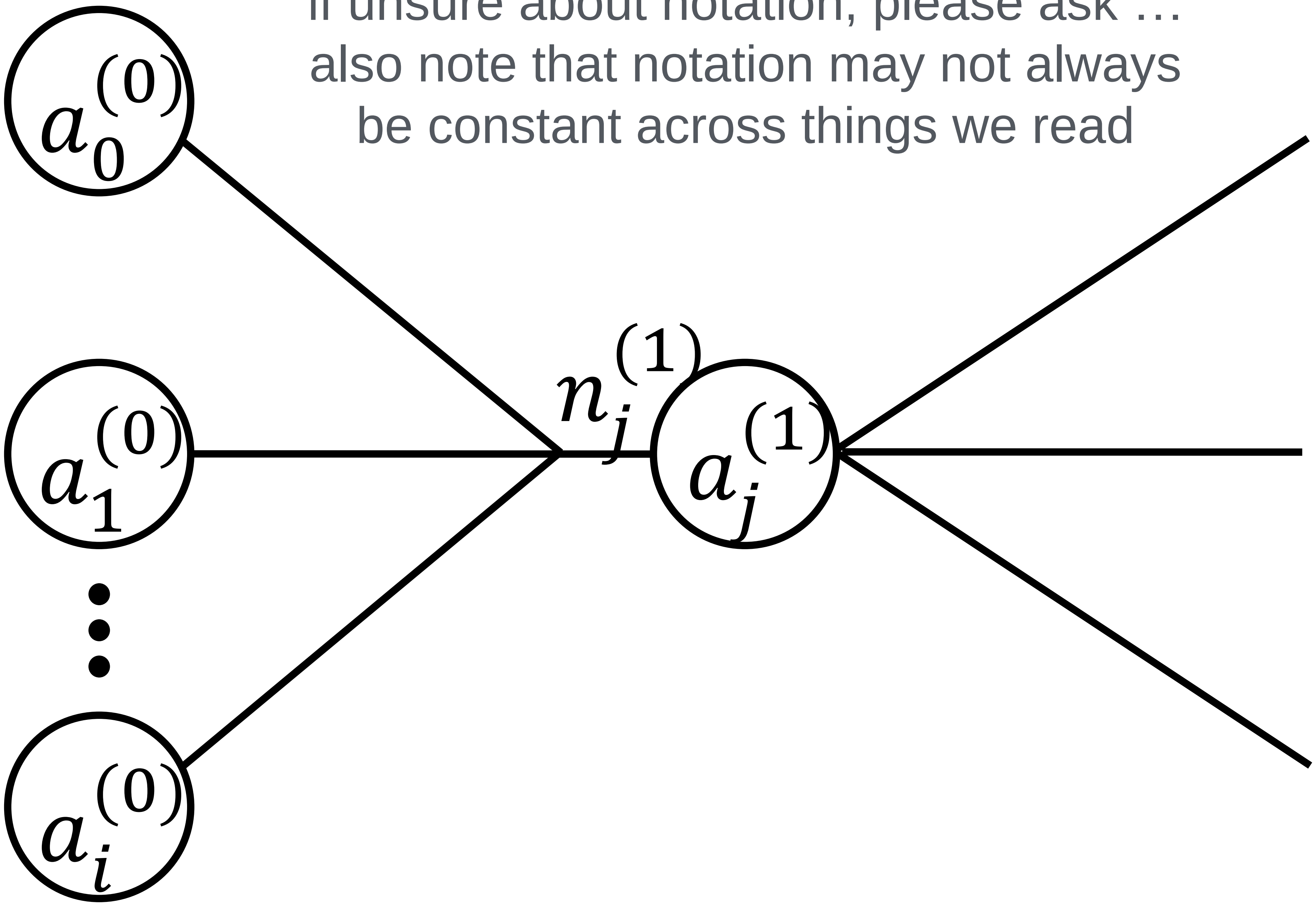
activation of
pre-synaptic
neurons

Note: because neural networks have many nodes and many connections, we will have to make extensive use of subscripts.

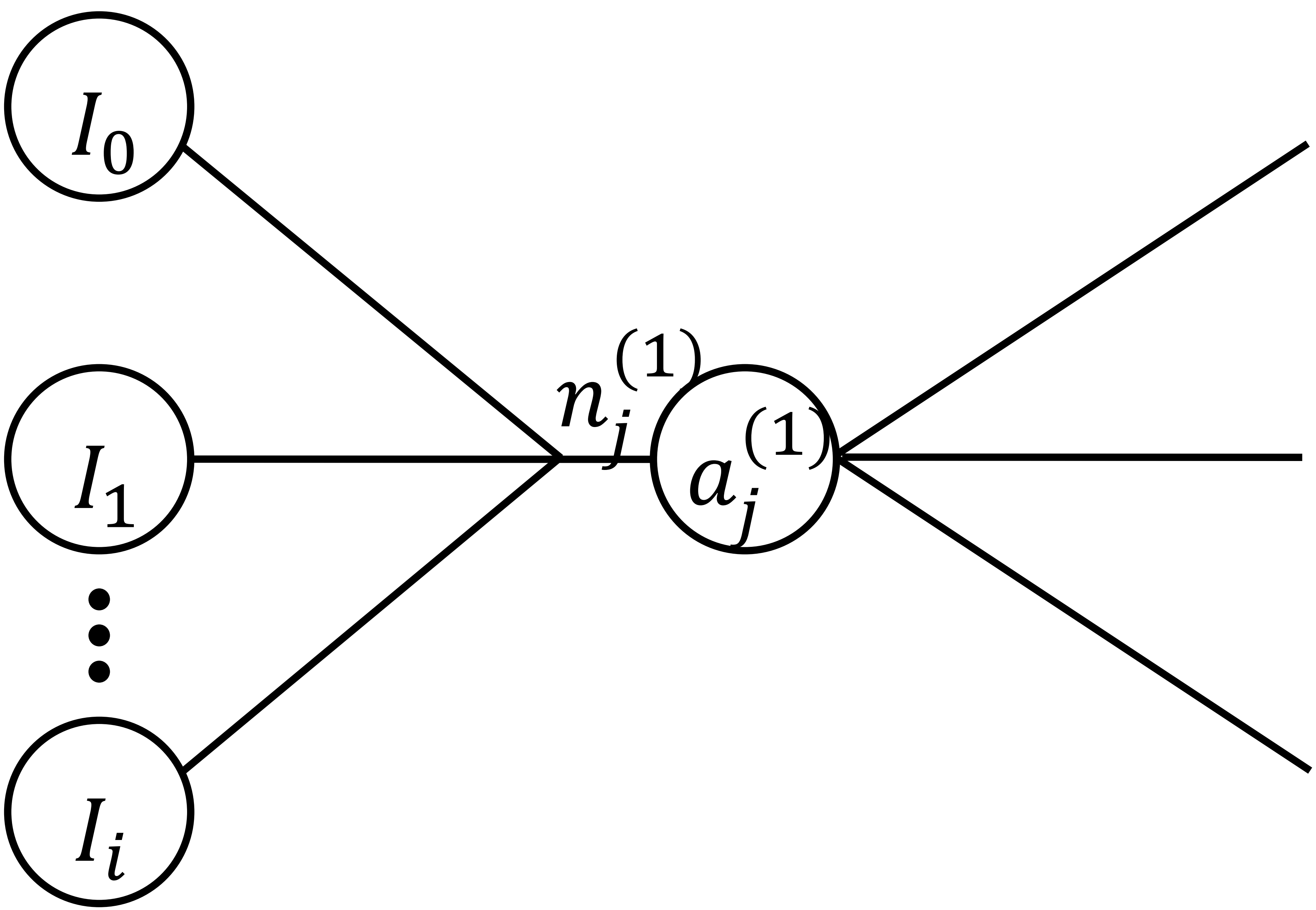


activation of pre-synaptic neurons

and sometimes superscripts as well …
if unsure about notation, please ask …
also note that notation may not always be constant across things we read



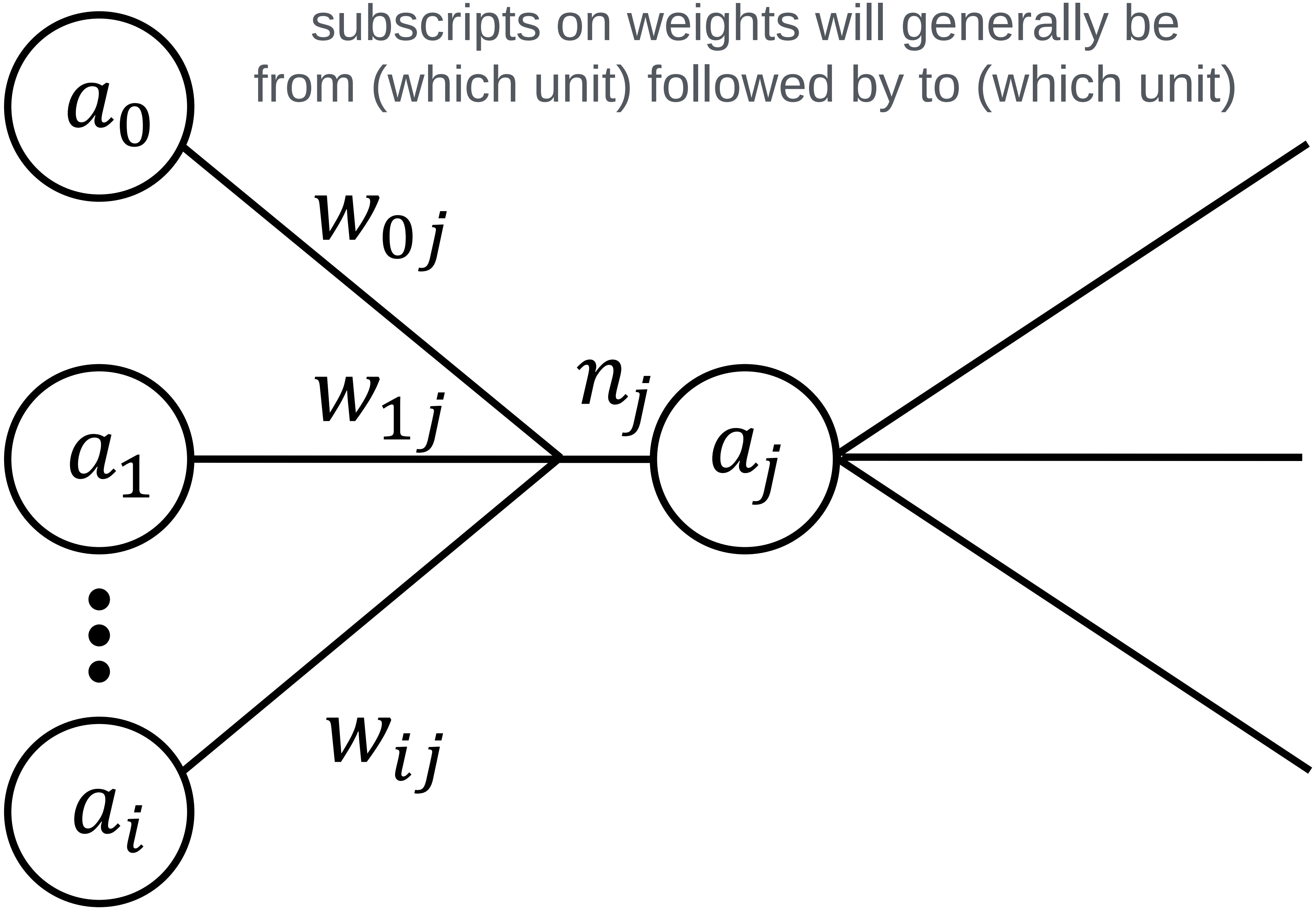
inputs clamped
onto network



$$w_{ij}$$

weight of synaptic connections

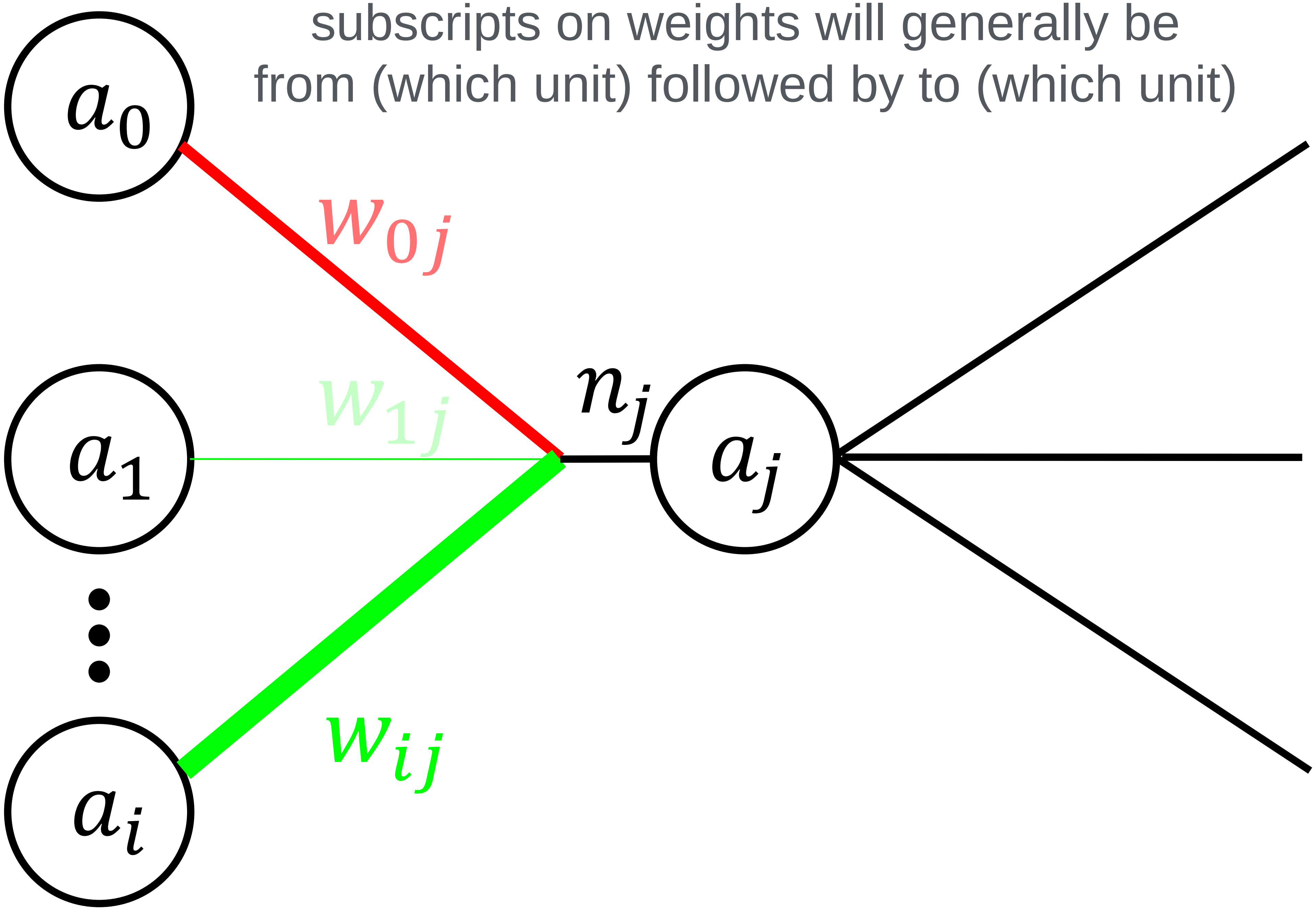
subscripts on weights will generally be
from (which unit) followed by to (which unit)



w_{ij}

weight of synaptic connections
can vary in strength (and sign)

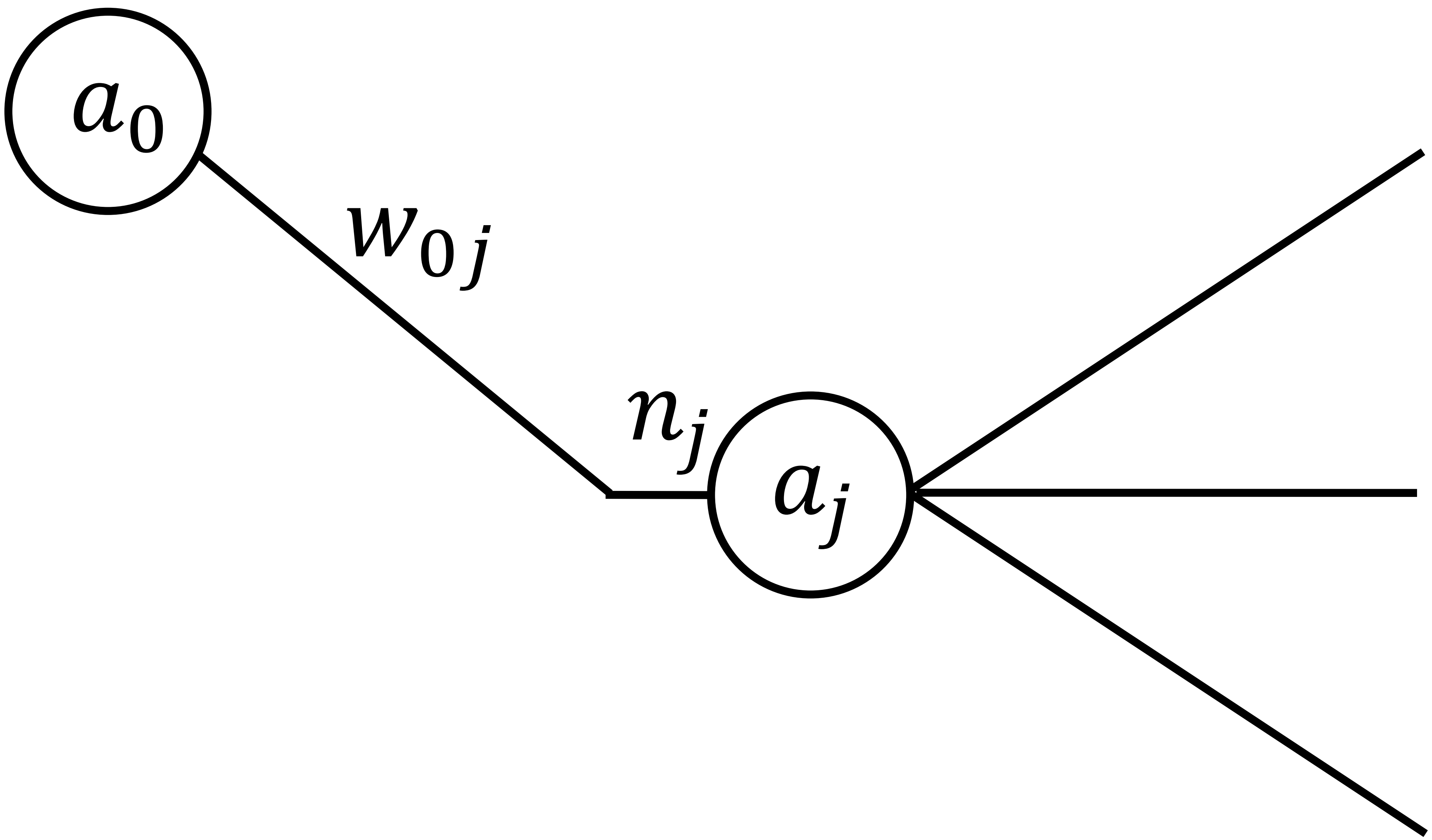
subscripts on weights will generally be
from (which unit) followed by to (which unit)



$$a_i w_{ij}$$

impact of one pre-synaptic neuron

multiplicatively varying strength of connection based on number/strength of synapses

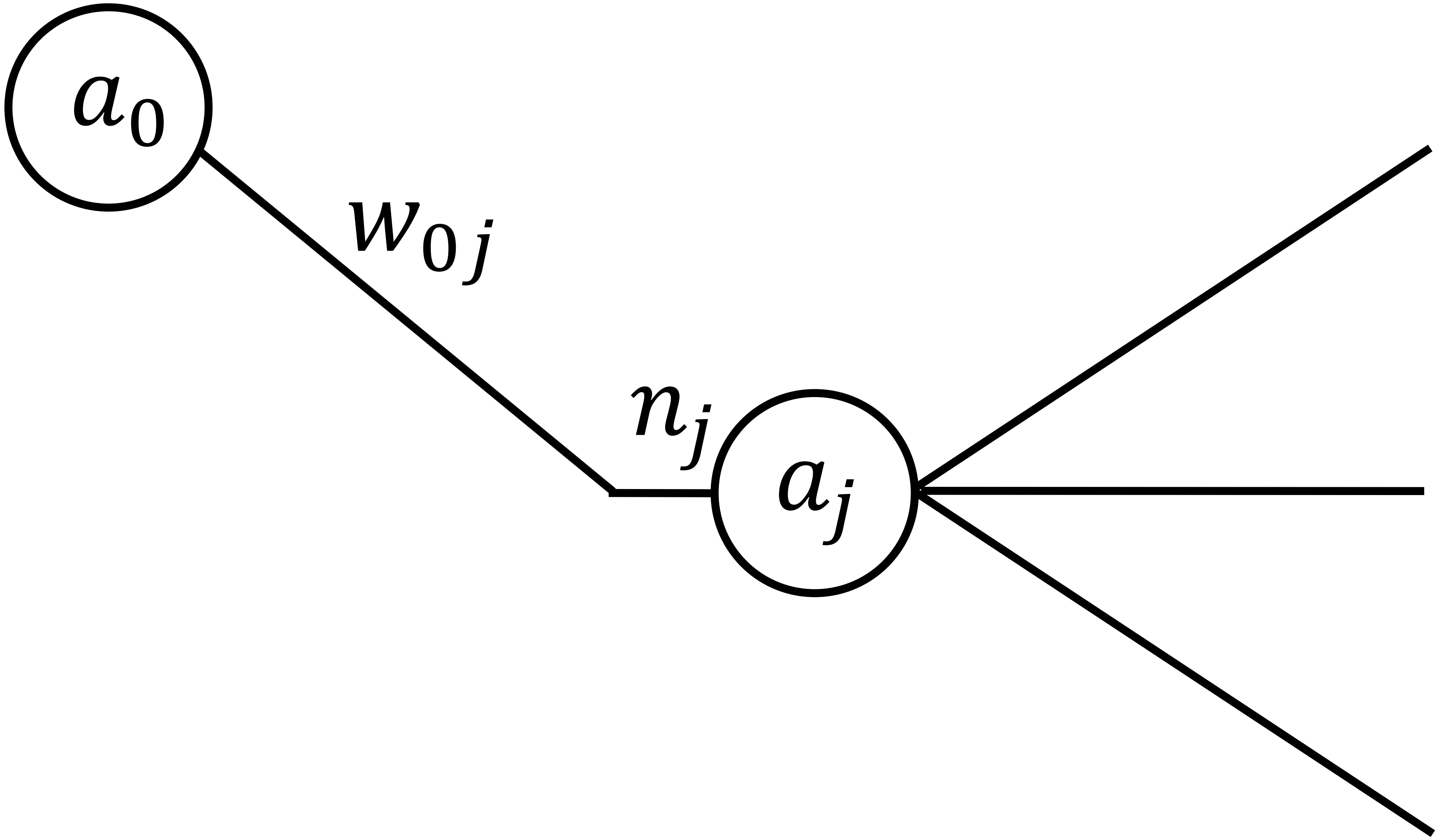


$$a_i w_{ij}$$

why is it multiplicative and not additive?

multiplication is more then just repeated addition

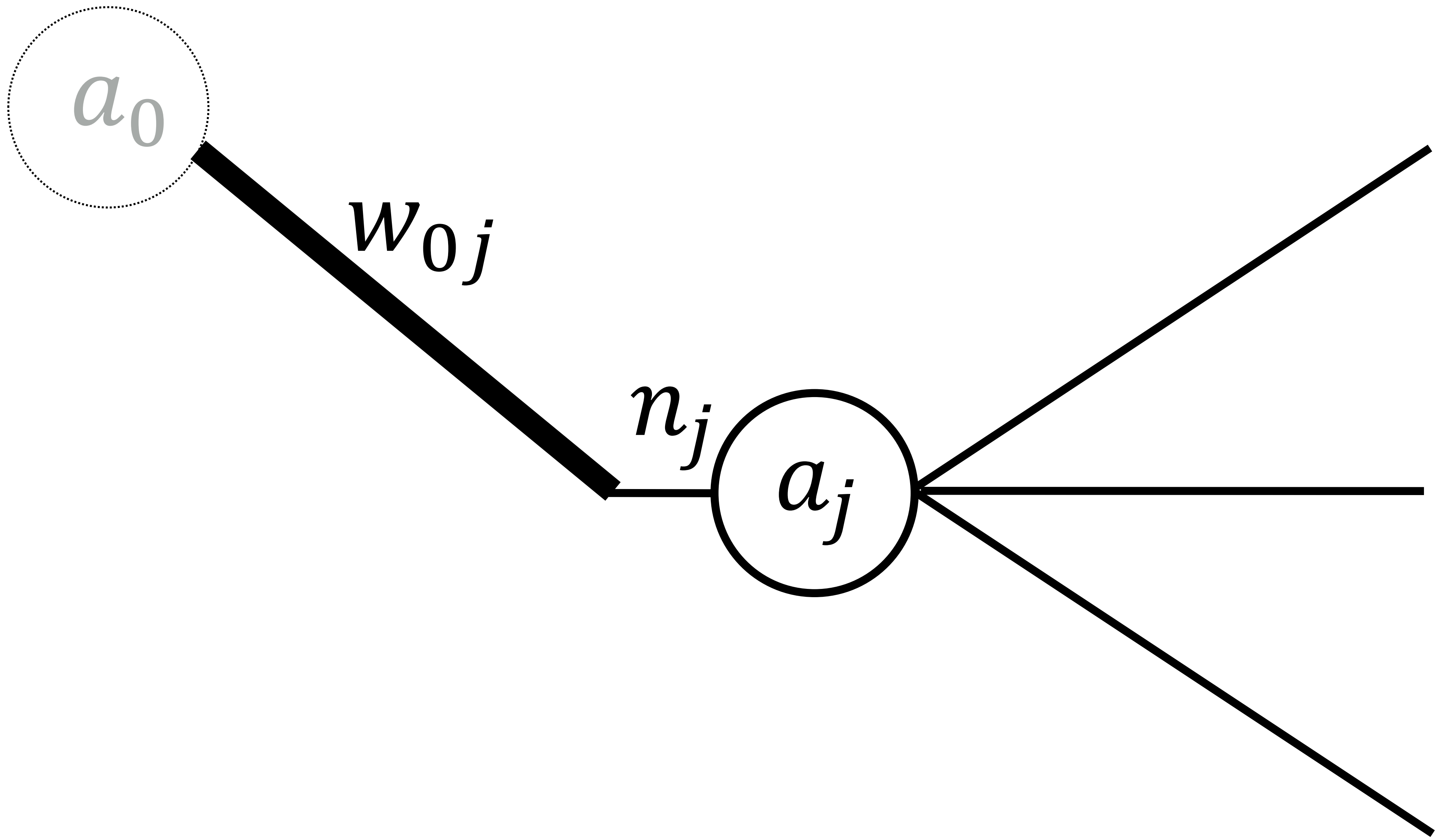
multiplicatively varying strength of connection based on number/strength of synapses



$$a_i w_{ij}$$

if the pre-synaptic neuron is inactive, it does not matter how strong the connection

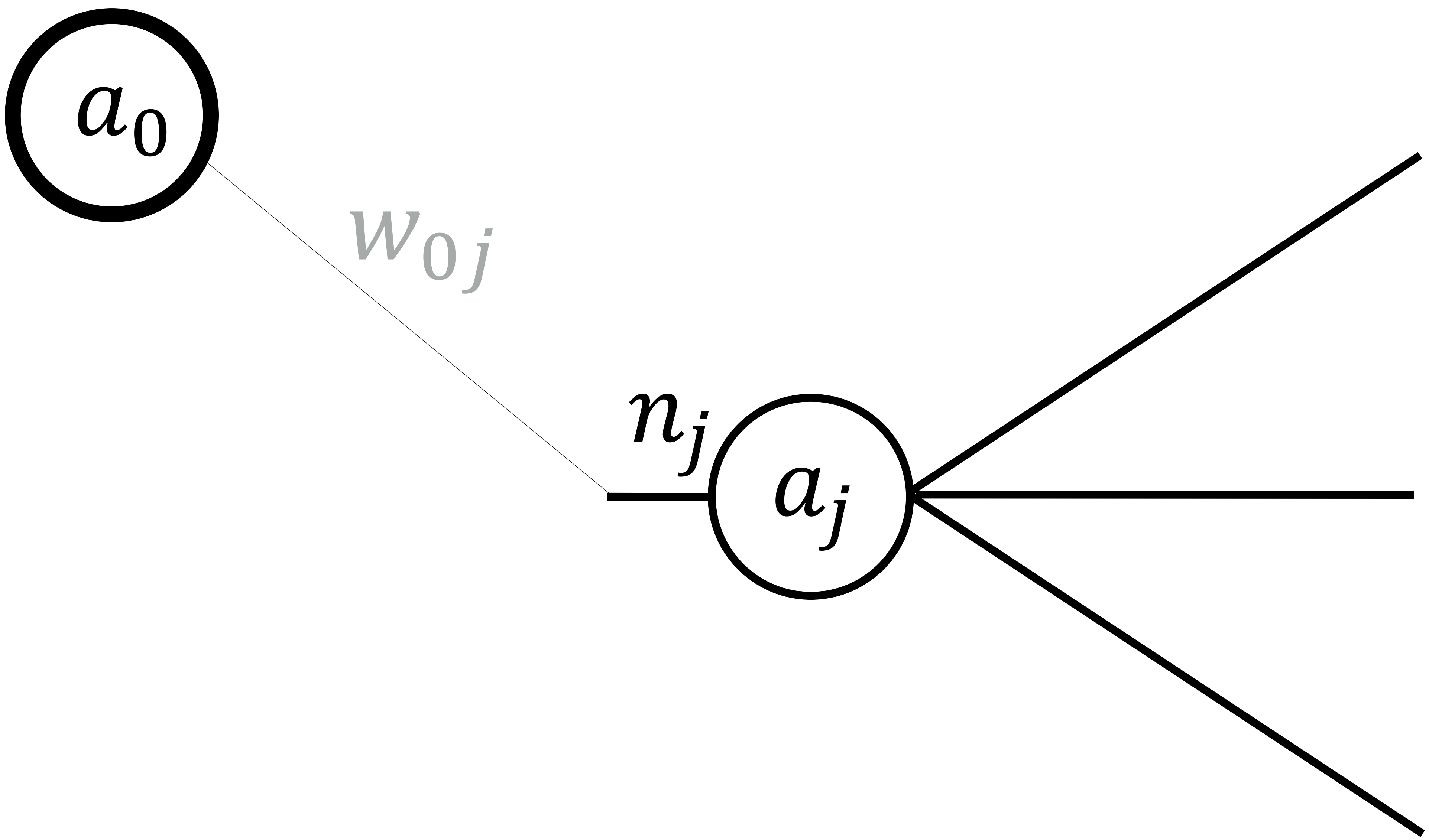
multiplicatively varying strength of connection based on number/strength of synapses



$$a_i w_{ij}$$

if the connection is weak or absent, it does not matter if pre-synaptic neuron is active or not

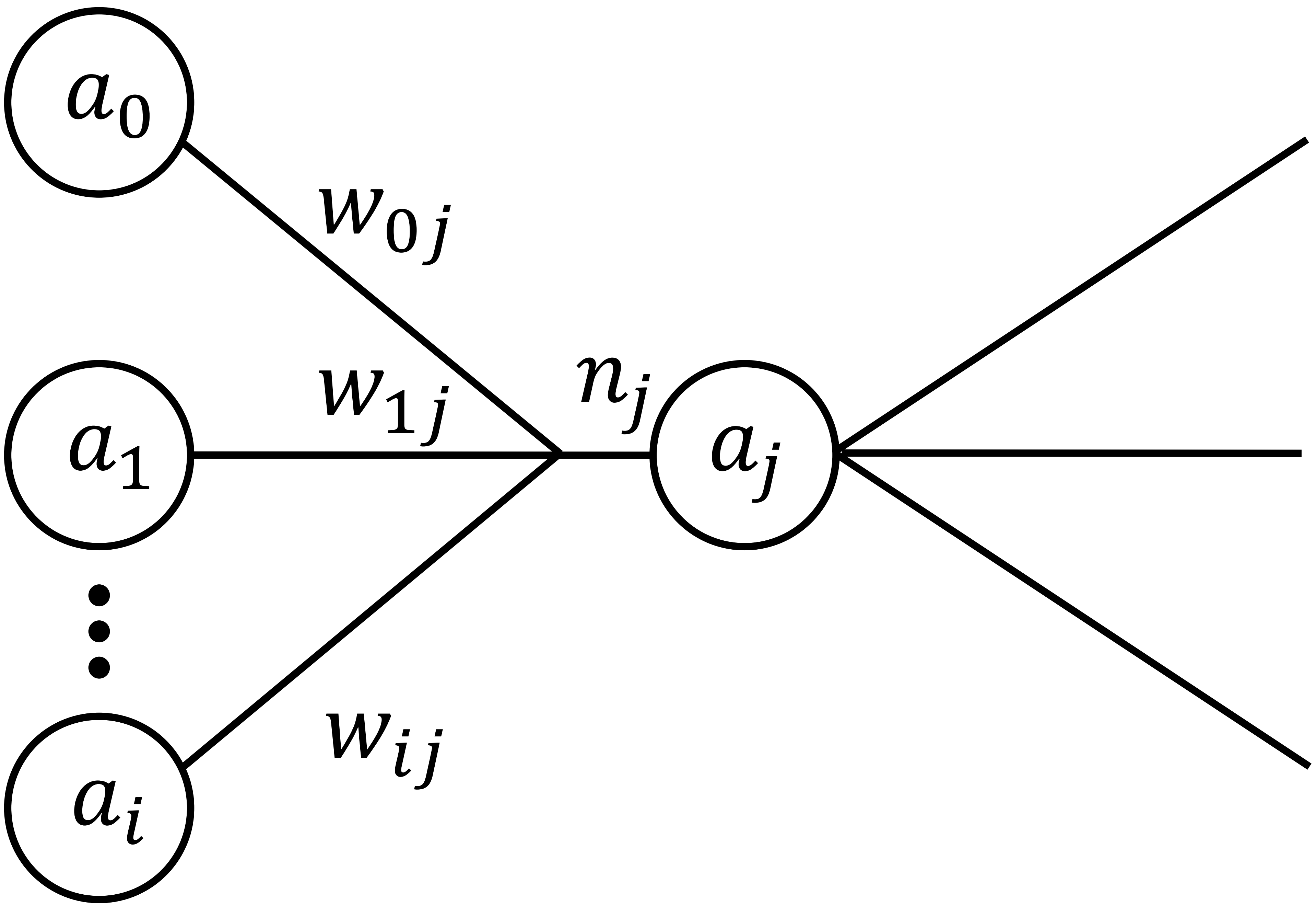
multiplicatively varying strength of connection based on number/strength of synapses



inputs integrate at the cell body - they are added together

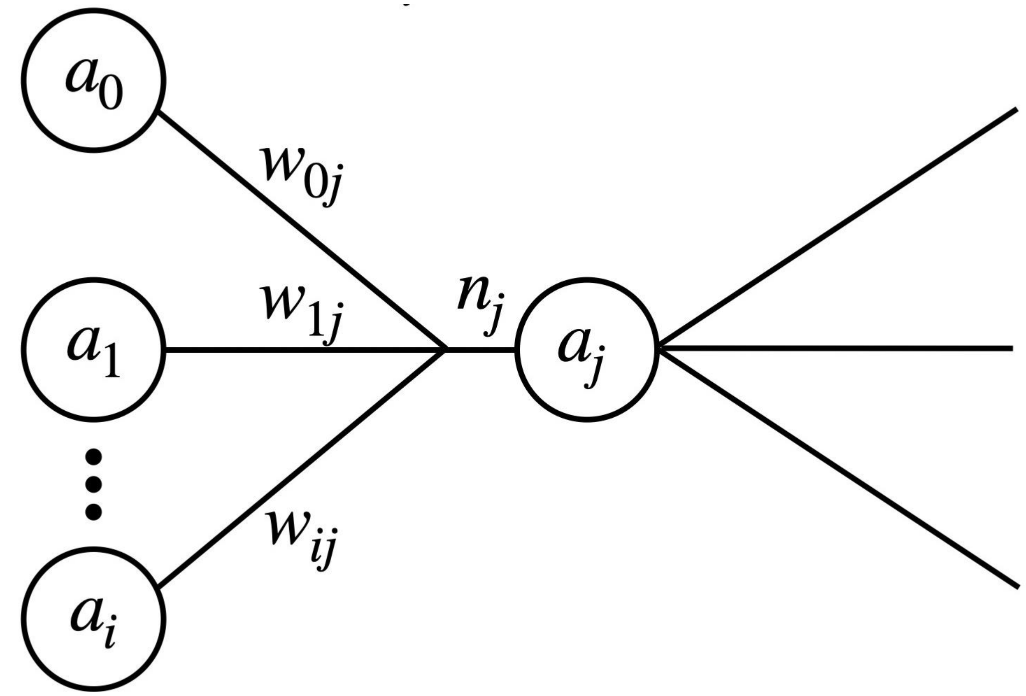
net input sums the weighted inputs

$$n_j = \sum_i a_i w_{ij}$$



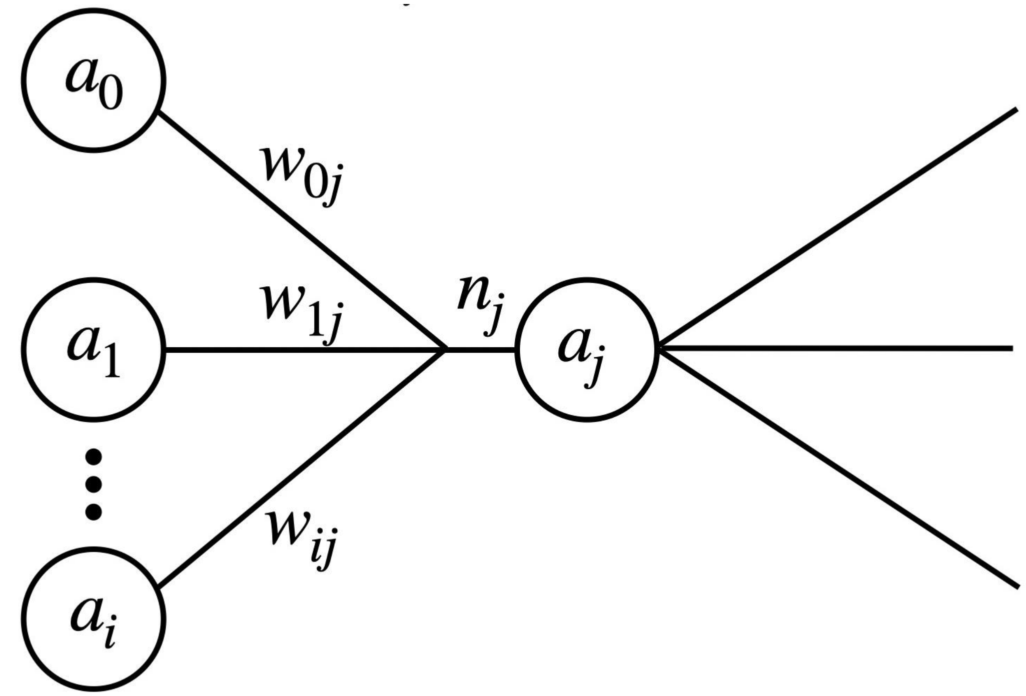
let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



let's take a look at this equation

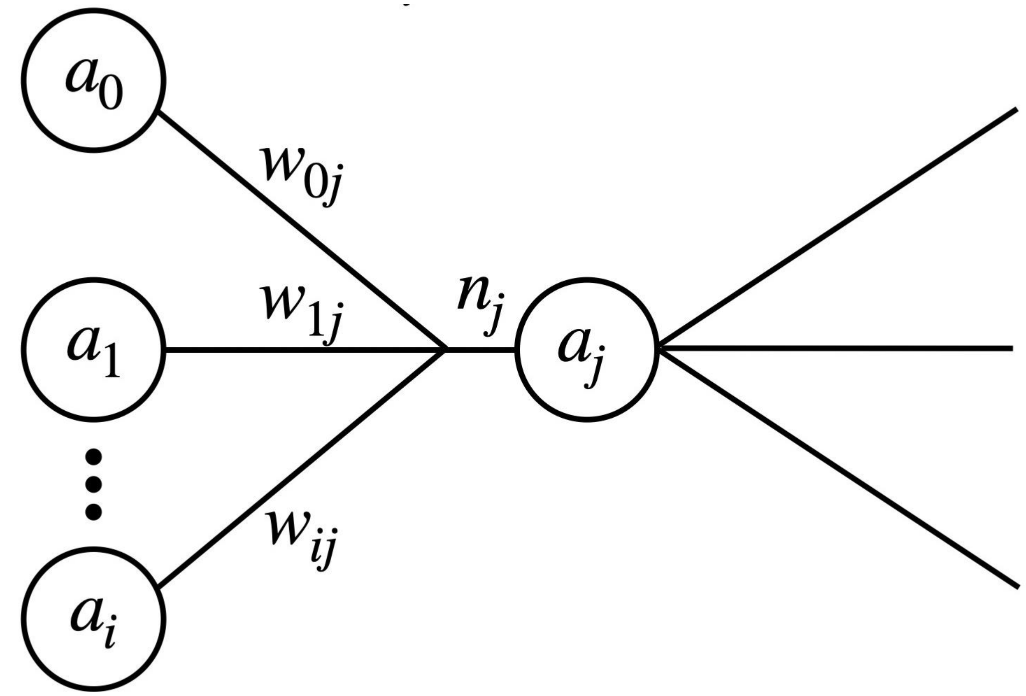
$$n_j = \sum_i a_i w_{ij}$$



$$n_j = \sum_{i=0}^{M-1} a_i w_{ij}$$

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



$$n_j = \sum_{i=0}^{M-1} a_i w_{ij} = a_0 w_{0j} + \ldots + a_i w_{ij} + \ldots + a_{M-1} w_{M-1,j}$$

$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

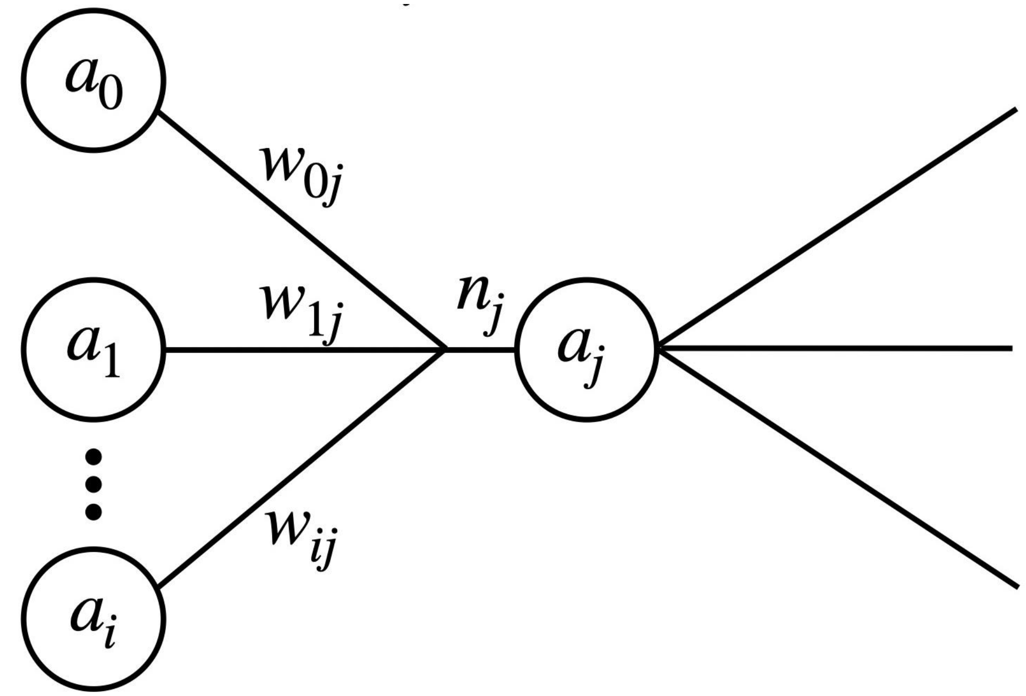
$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

vector of weights on connections to node j

see `Week3b.ipynb`

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$

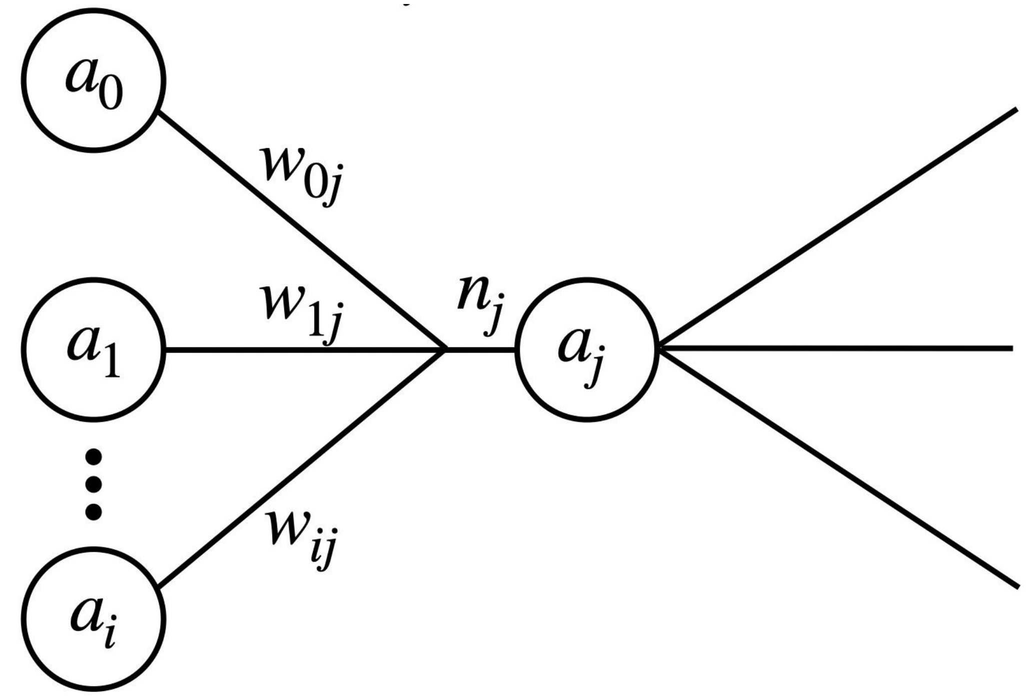


$$n_j = \sum_{i=0}^{M-1} a_i w_{ij} = \boxed{a_0} w_{0j} + \ldots + \boxed{a_i} w_{ij} + \ldots + \boxed{a_{M-1}} w_{M-1,j}$$

$$a = [a_0, \ldots, a_i, \ldots a_{M-1}] \qquad w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



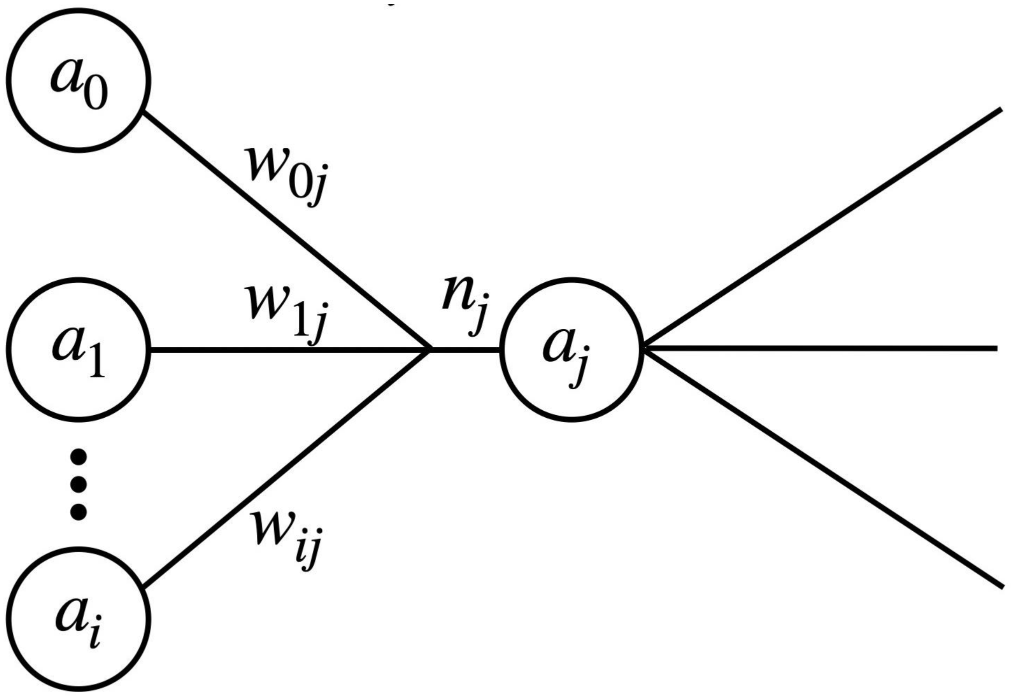
$$n_j = \sum_{i=0}^{M-1} a_i w_{ij} = a_0 \boxed{w_{0j}} + \ldots + a_i \boxed{w_{ij}} + \ldots + a_{M-1} \boxed{w_{M-1,j}}$$

$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,}$$

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

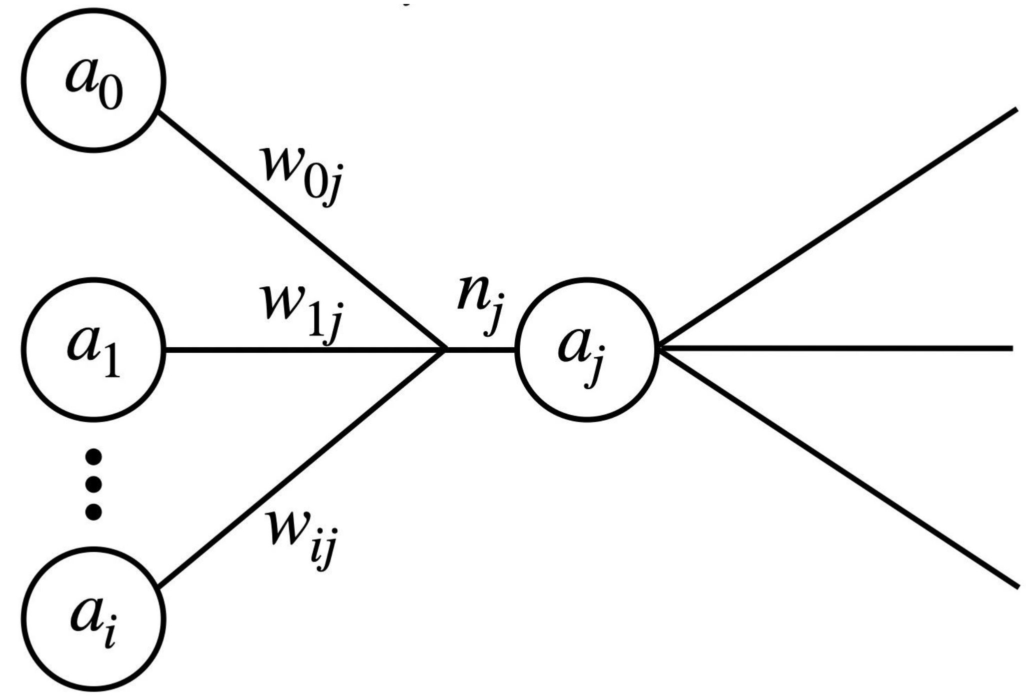
vector of weights on connections to node j

```
net = 0.0
for i in range(len(a)):
    net += a[i]*wj[i]
```

using for loops

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

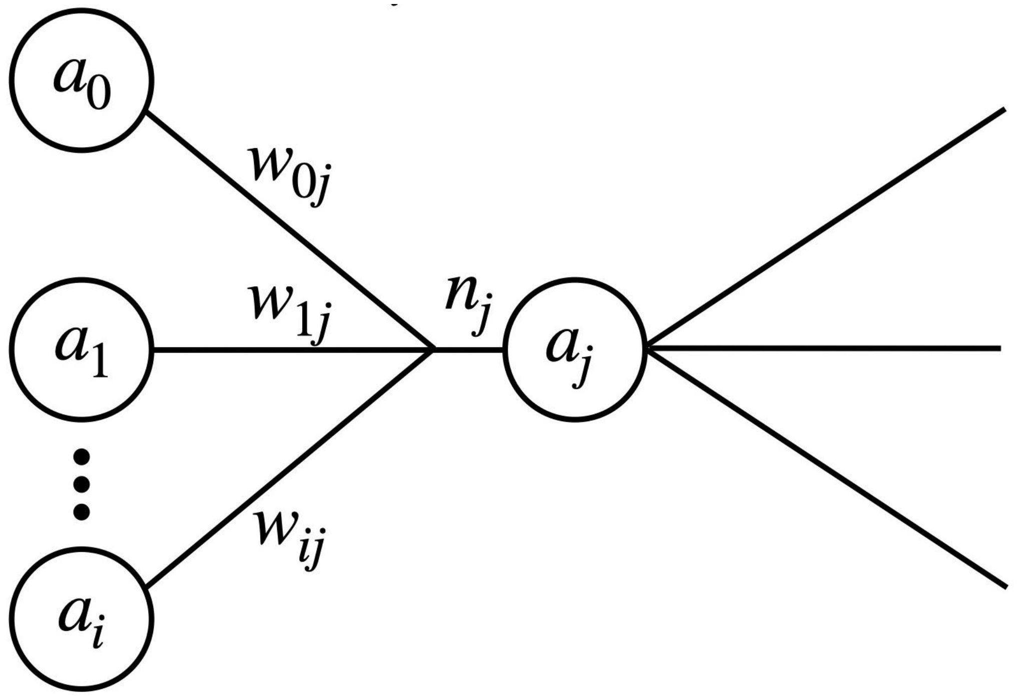
vector of weights on connections to node j

```
net = np.sum(a*wj)
```

vectorized

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

vector of weights on connections to node j

numpy array (from element-wise multiplication)

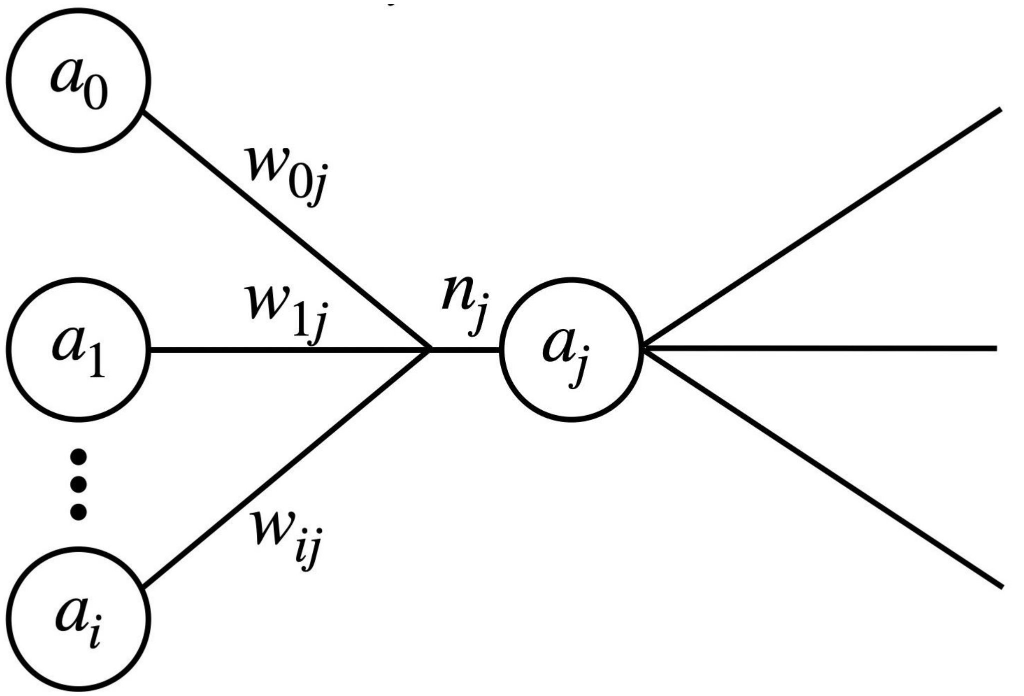
```
net = a*wj
net = np.sum(net)
```

float (sum of elements of numpy array)

vectorized

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$



$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

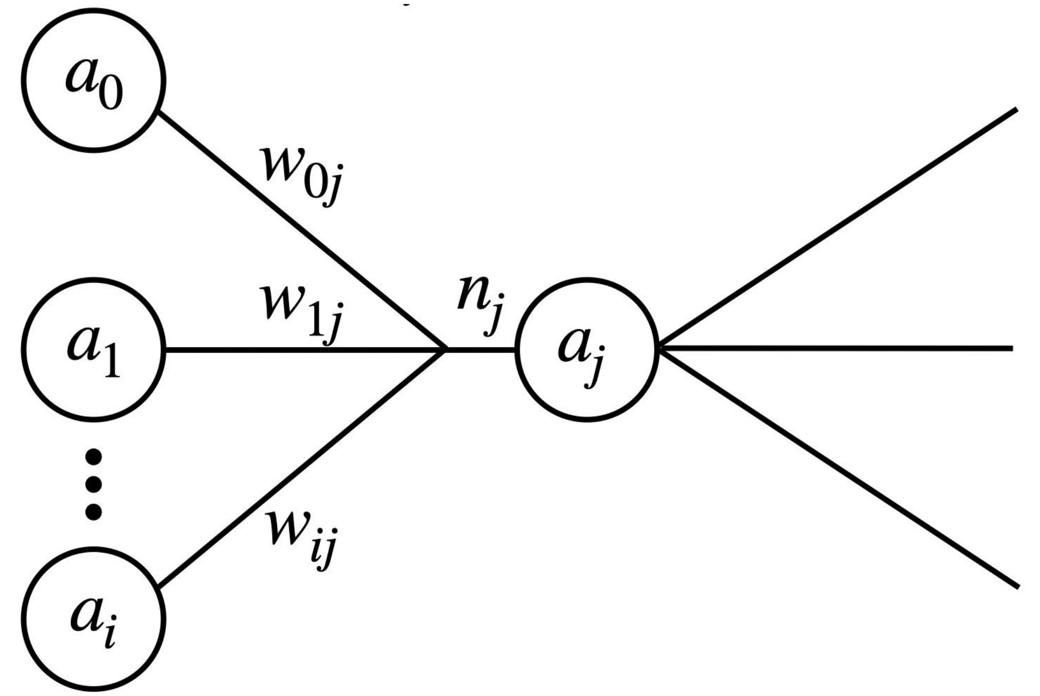
vector of weights on connections to node j

```
net = np.dot(a, wj)
```

as vector dot product

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij} = \boxed{a \cdot w_j}$$



$$a = [a_0, \ldots, a_i, \ldots a_{M-1}]$$

vector of activations of the input nodes

$$w_j = [w_{0j}, \ldots, w_{ij}, \ldots w_{M-1,j}]$$

vector of weights on connections to node j

```
net = np.dot(a, wj)
```

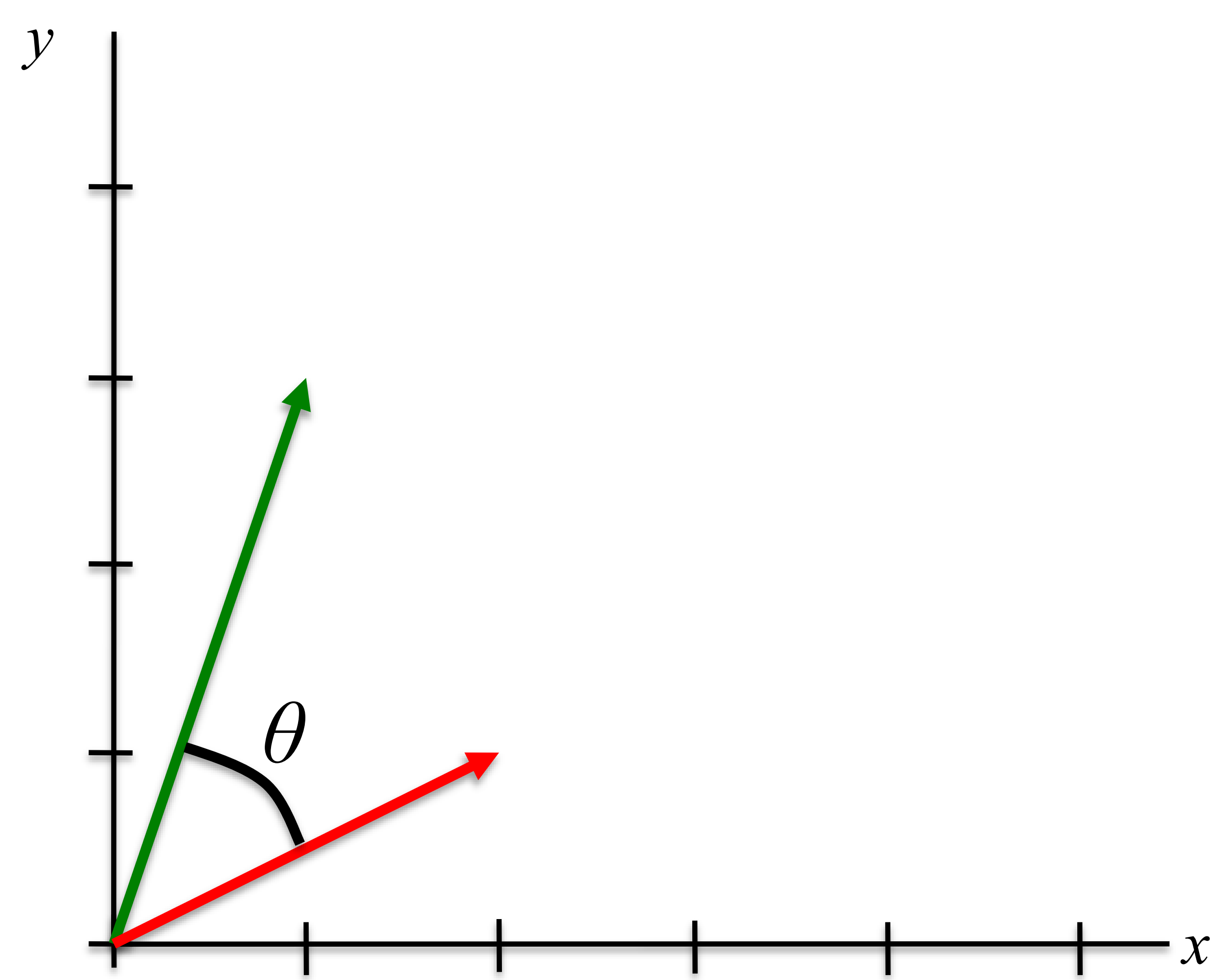
as vector dot product

linear algebra operation on vectors

dot product

computes a “similarity” between two vectors

```
a  = np.array([1, 3])
wj = np.array([2, 1])
```



dot product

angle between two vectors …

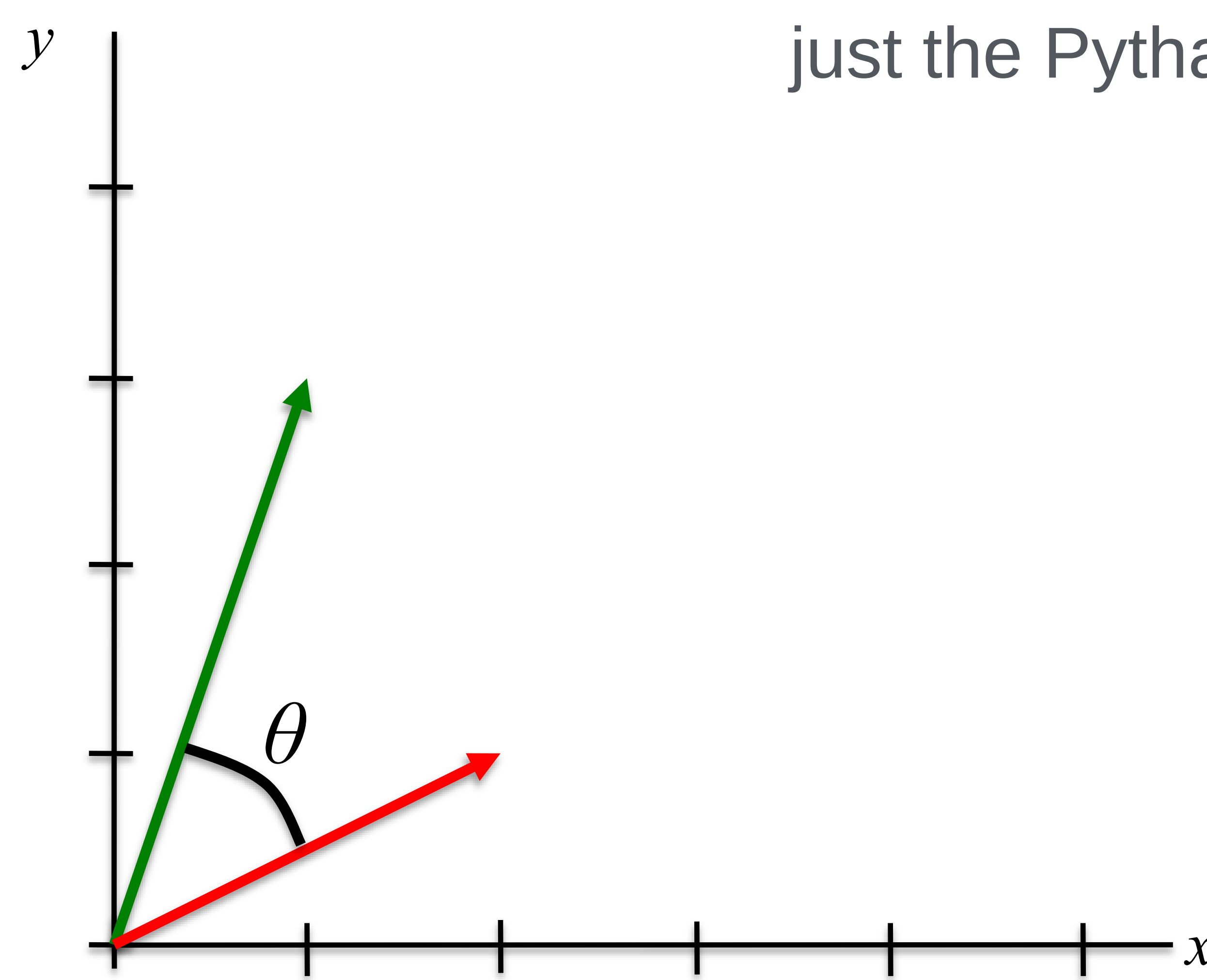
$$cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w \|}$$

dot product

norms (lengths)

$$\| a \| = \sqrt{a \cdot a}$$

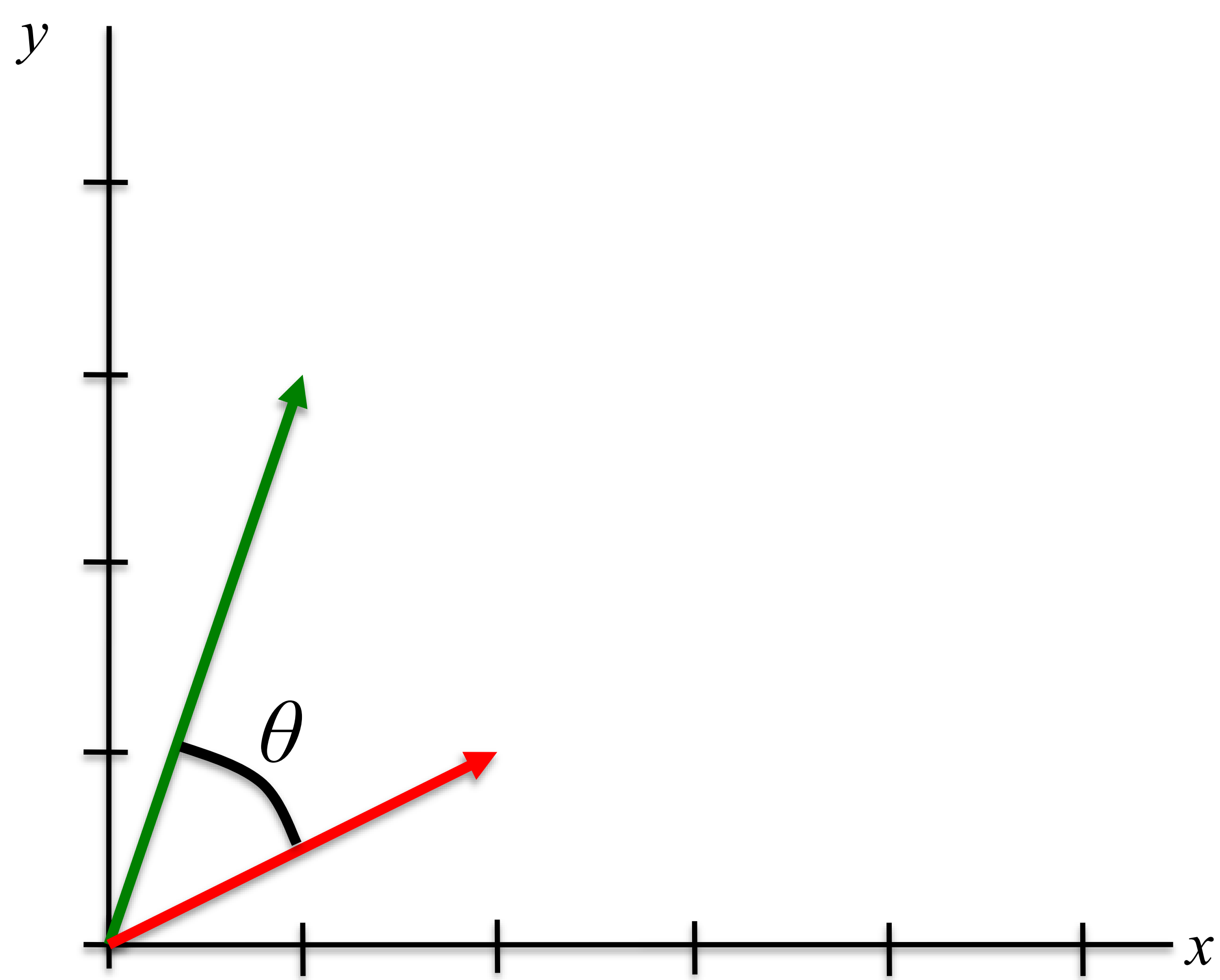
just the Pythagorean theorem



dot product

angle between two vectors …

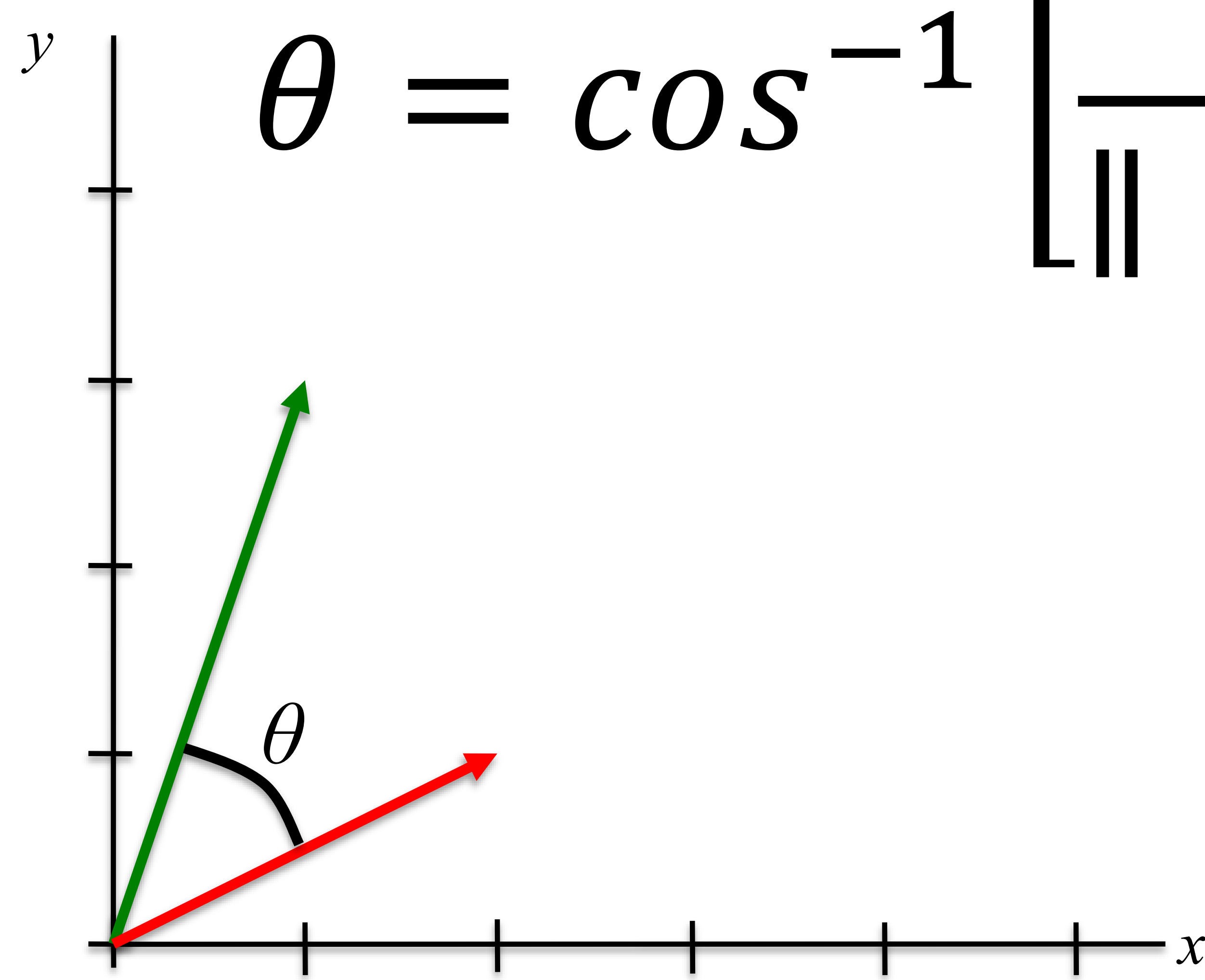
$$cos(\theta) = \frac{a}{\| a \|} \cdot \frac{w_j}{\| w \|}$$



dot product

```
costheta = np.dot(a,wj) /
                (np.sqrt(np.dot(a,a)) *
                 np.sqrt(np.dot(wj,wj)))
theta = np.arccos(costheta)
```

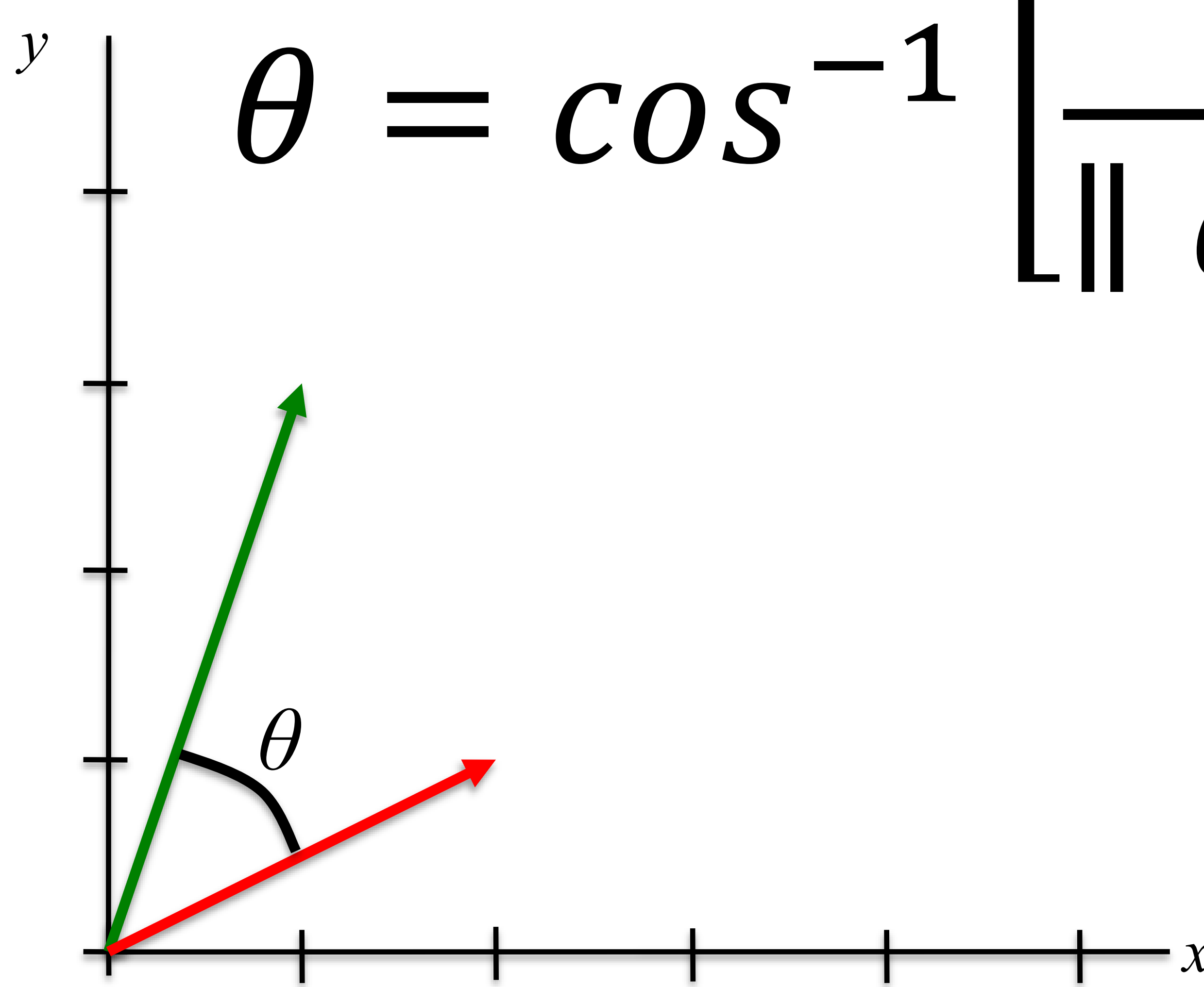
$$\theta = cos^{-1}\left[\frac{a \cdot w_j}{\parallel a \parallel \parallel w \parallel}\right]$$



dot product

```
costheta = np.dot(a,wj) /
                 (np.linalg.norm(a) *
                  np.linalg.norm(wj))
theta = np.arccos(costheta)
```

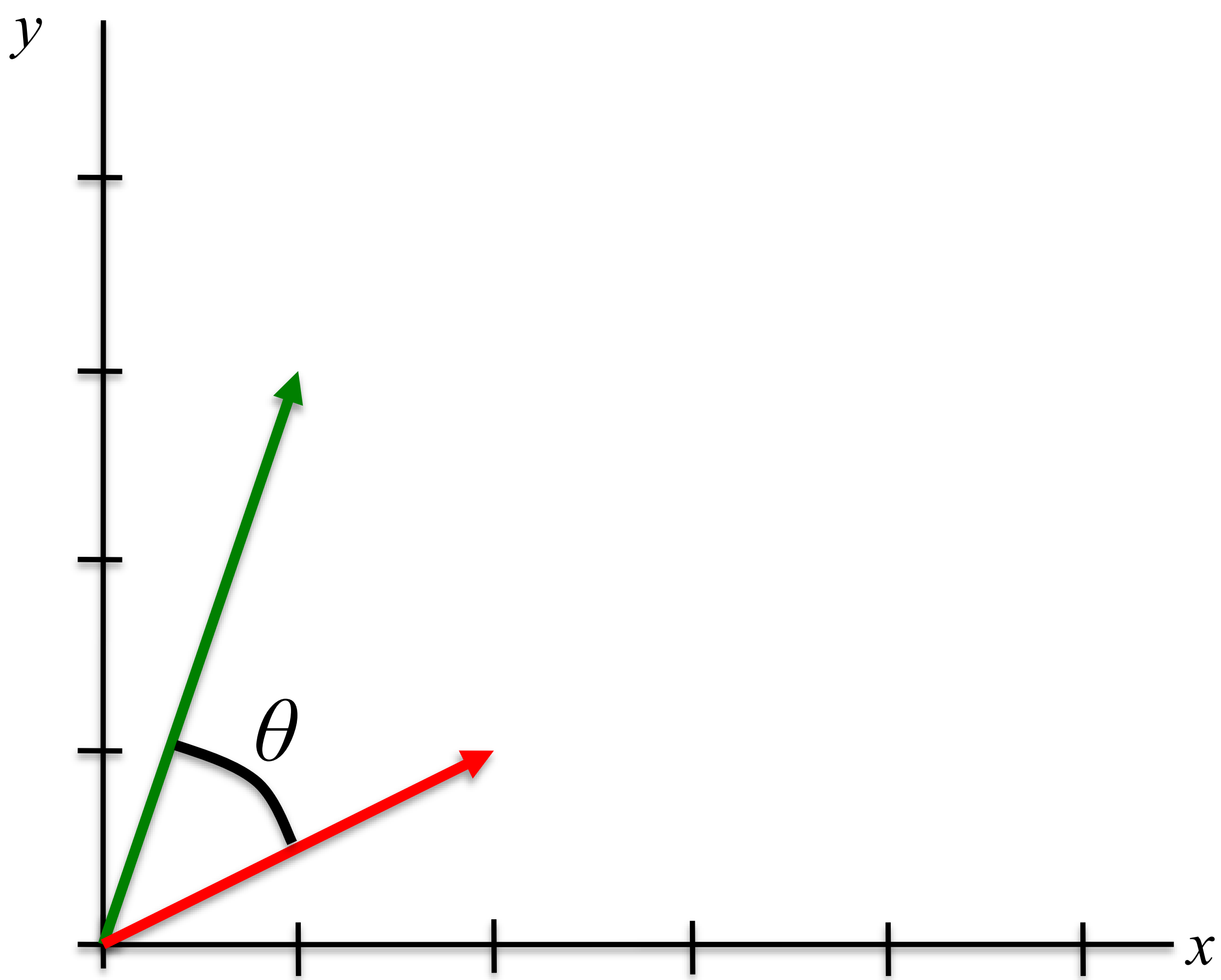
$$\theta = cos^{-1}\left[\frac{a \cdot w_j}{\| a \| \| w \|}\right]$$



dot product

another way to write this (rearrange terms)

$$cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w \|}$$

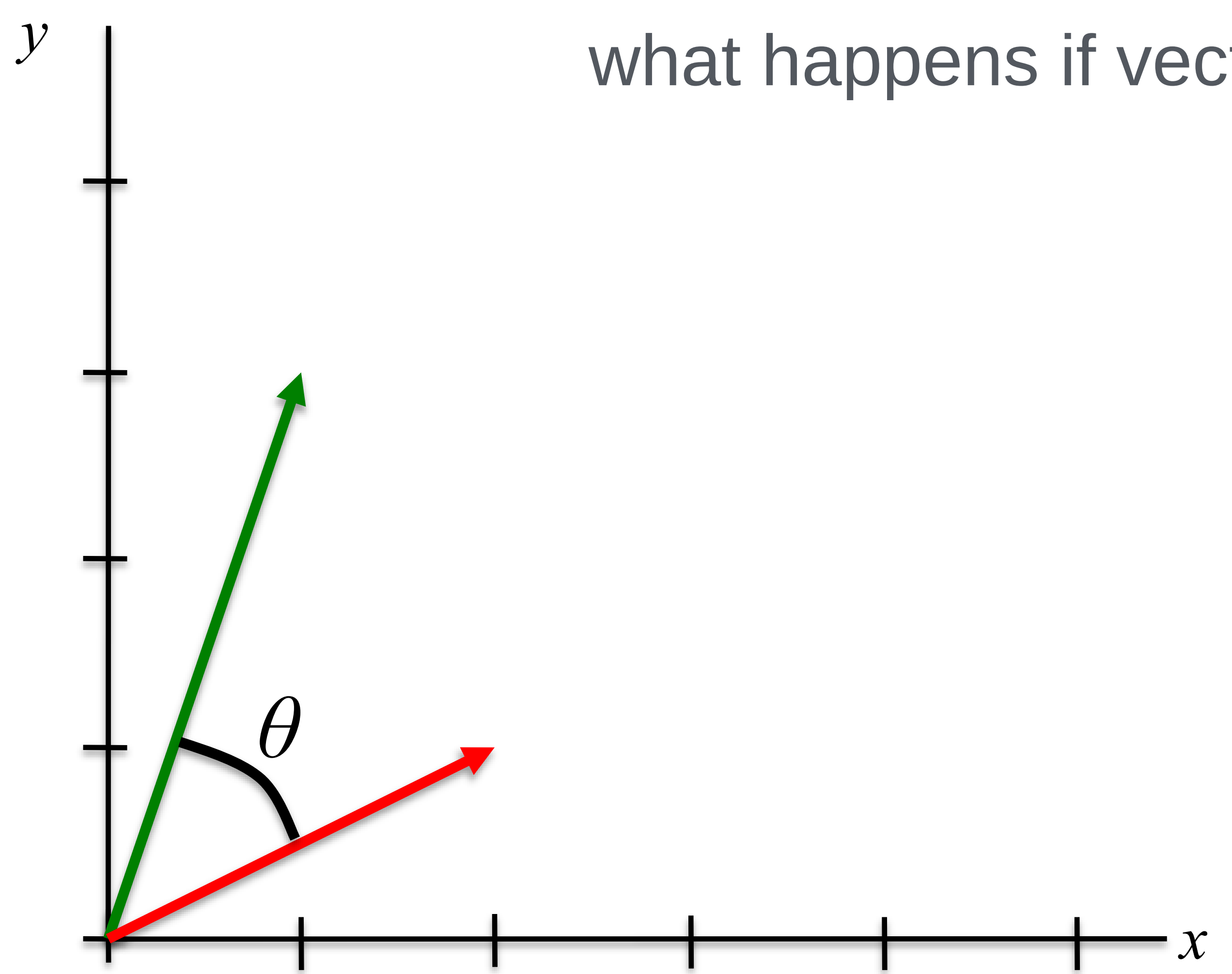


dot product

dot product equals product of the lengths and the angle

$$a \cdot w_j = \| a \| \| w \| \, cos(\theta)$$

dot product | length a | length w_j | cos angle



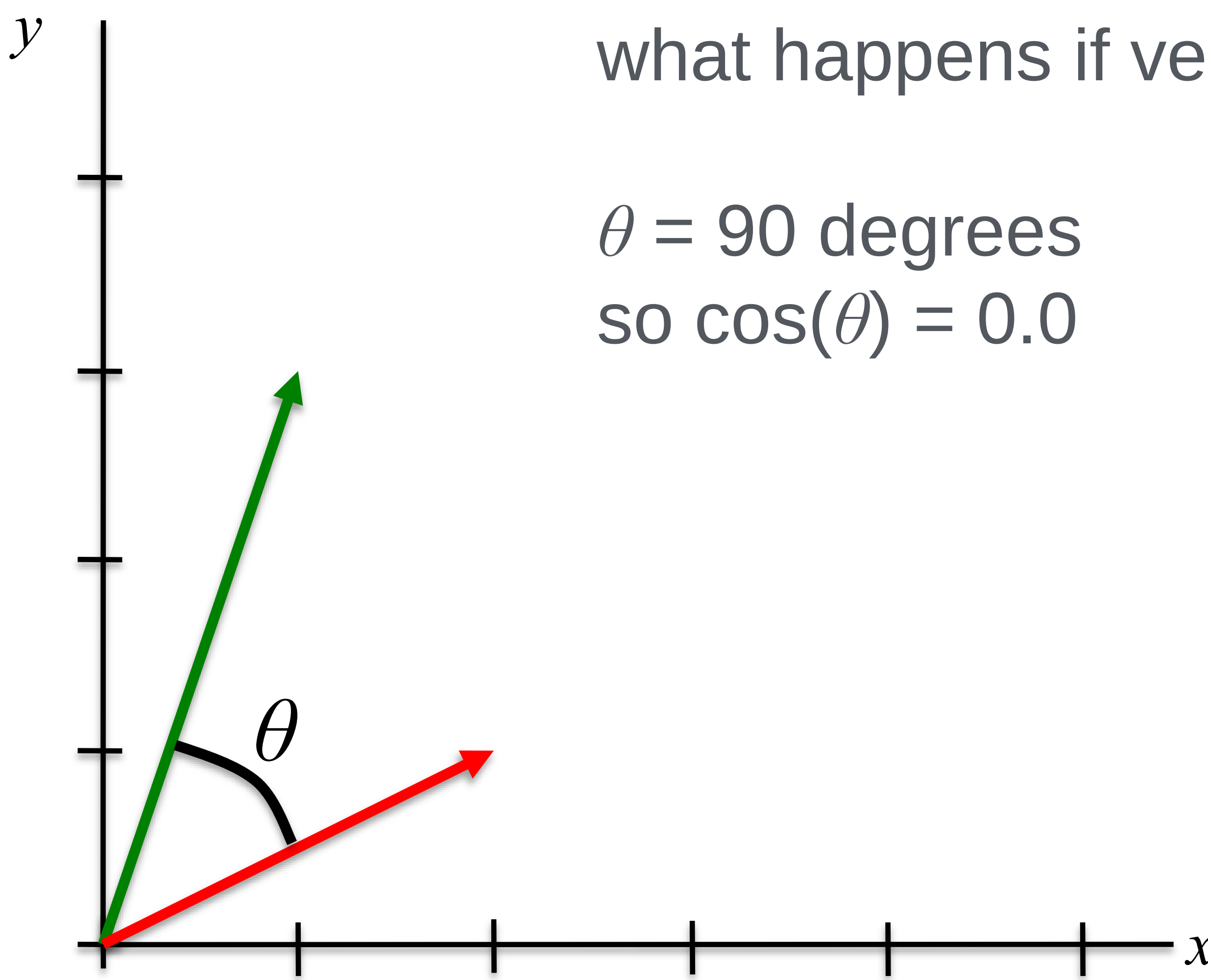
what happens if vectors are orthogonal?

dot product

dot product equals product of the lengths and the angle

$$a \cdot w_j = \| a \| \| w \| cos(\theta)$$

dot product	length a	length w_j	cos angle



what happens if vectors are orthogonal?

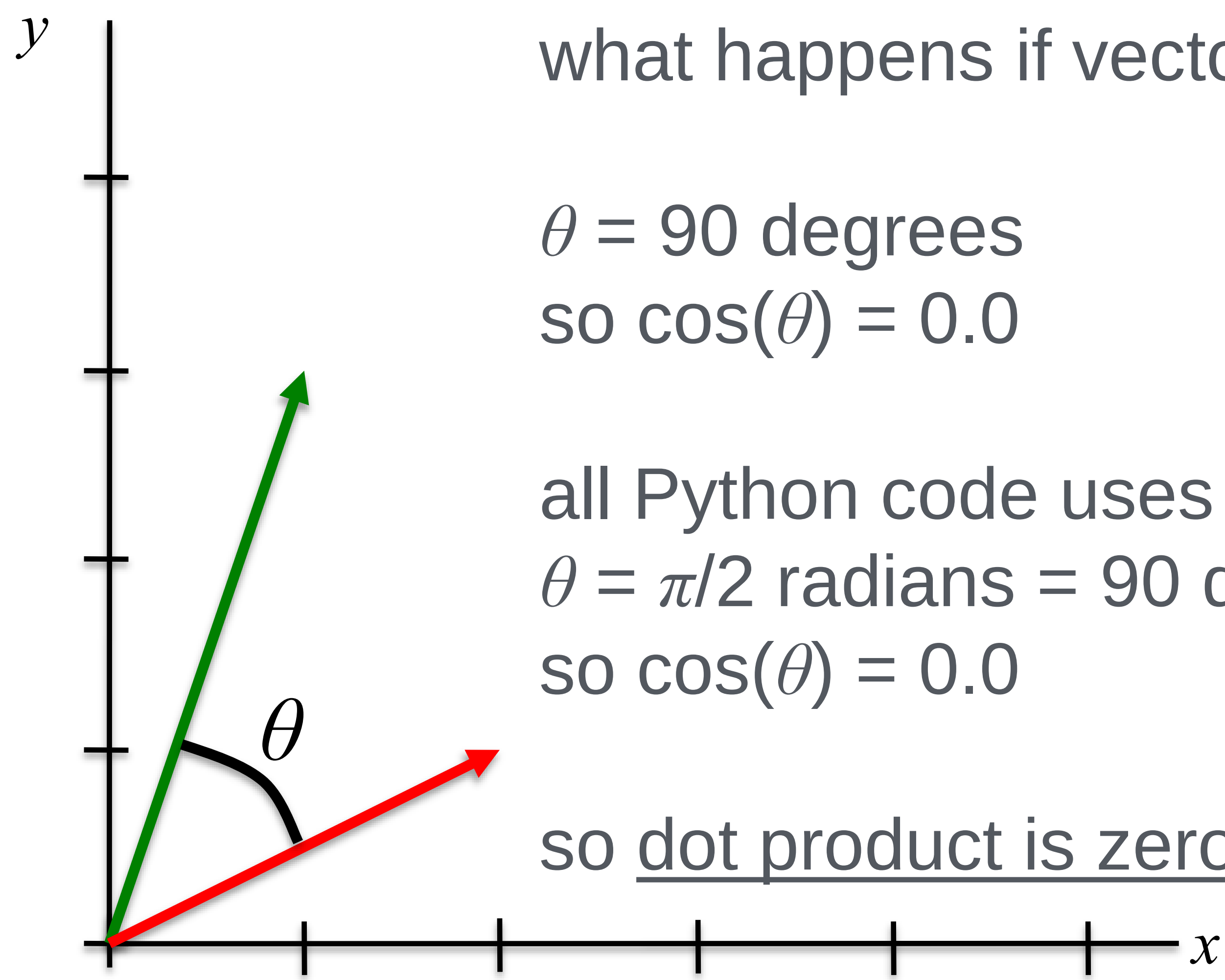
θ = 90 degrees
so cos(θ) = 0.0

dot product

dot product equals product of the lengths and the angle

$$a \cdot w_j = \| a \| \| w \| \, cos(\theta)$$

dot product | length a | length w_j | cos angle



what happens if vectors are orthogonal?

θ = 90 degrees
so cos(θ) = 0.0

all Python code uses radians (not degrees)
$\theta = \pi/2$ radians = 90 degrees
so cos(θ) = 0.0

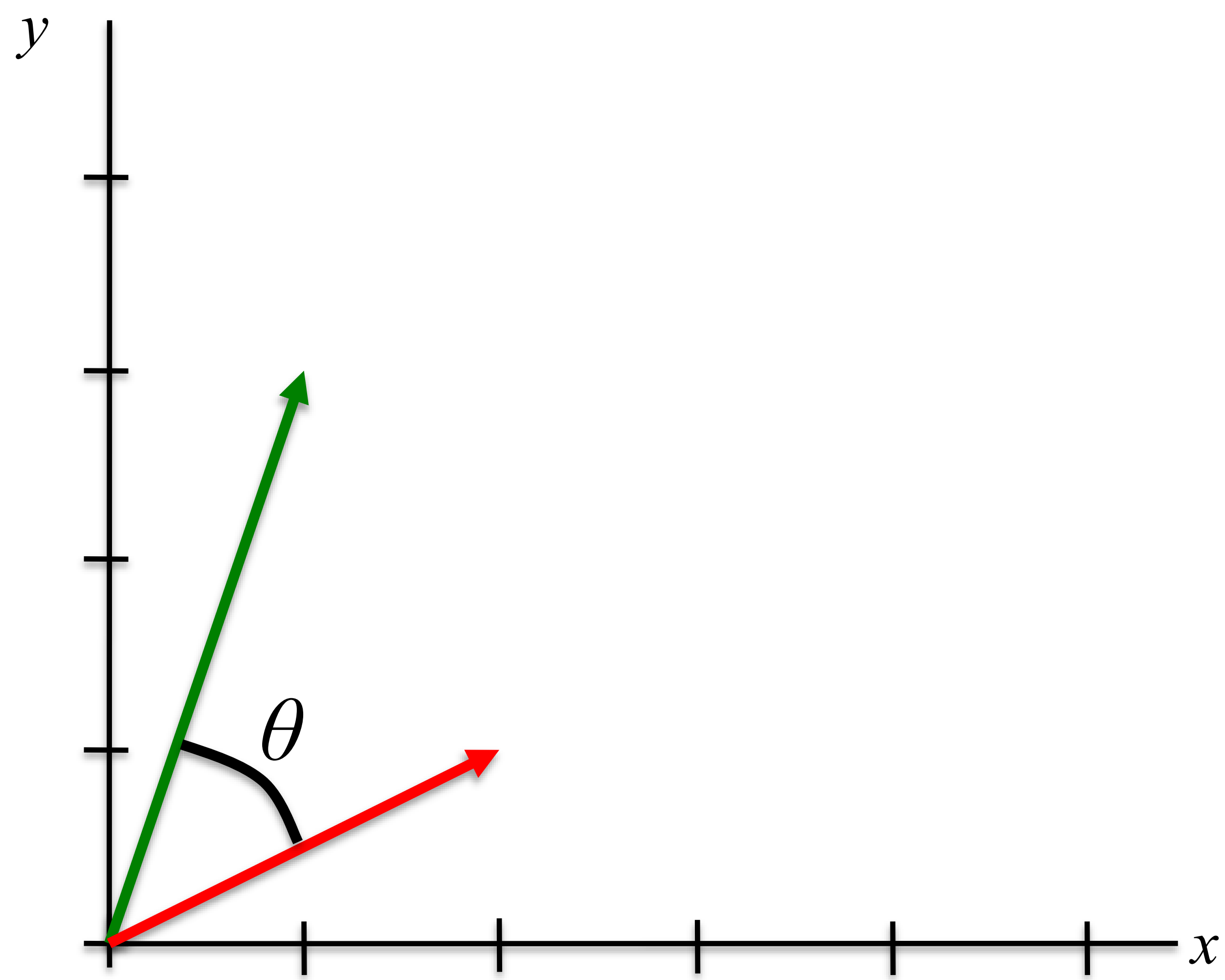
so dot product is zero

dot product

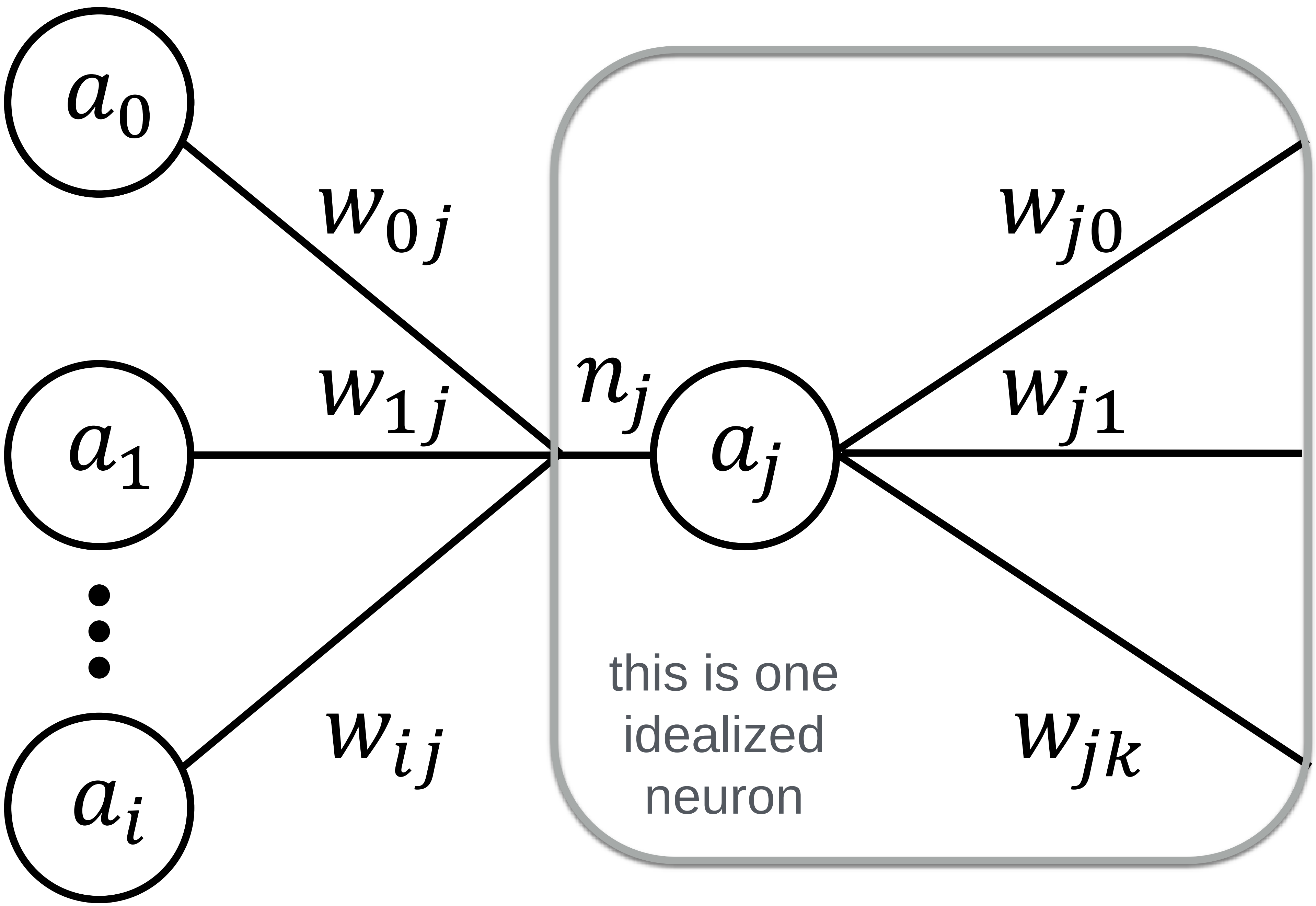
dot product is general

dot product lets you compute the angle (a measure of similarity) between vectors that are 2D, 10D, or 100D

$$cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w \|}$$



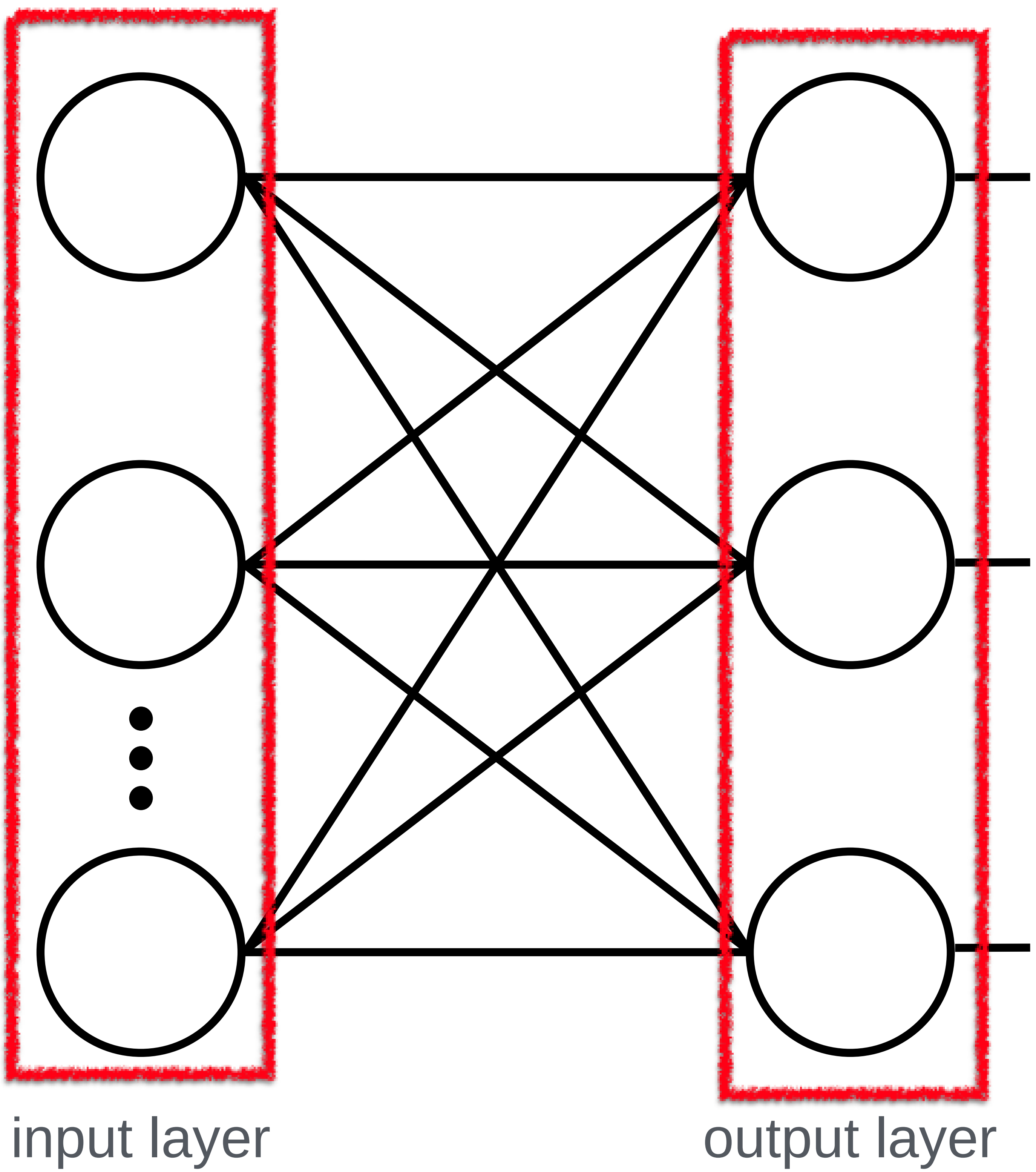
layers of a neural network



Idealized Neuron

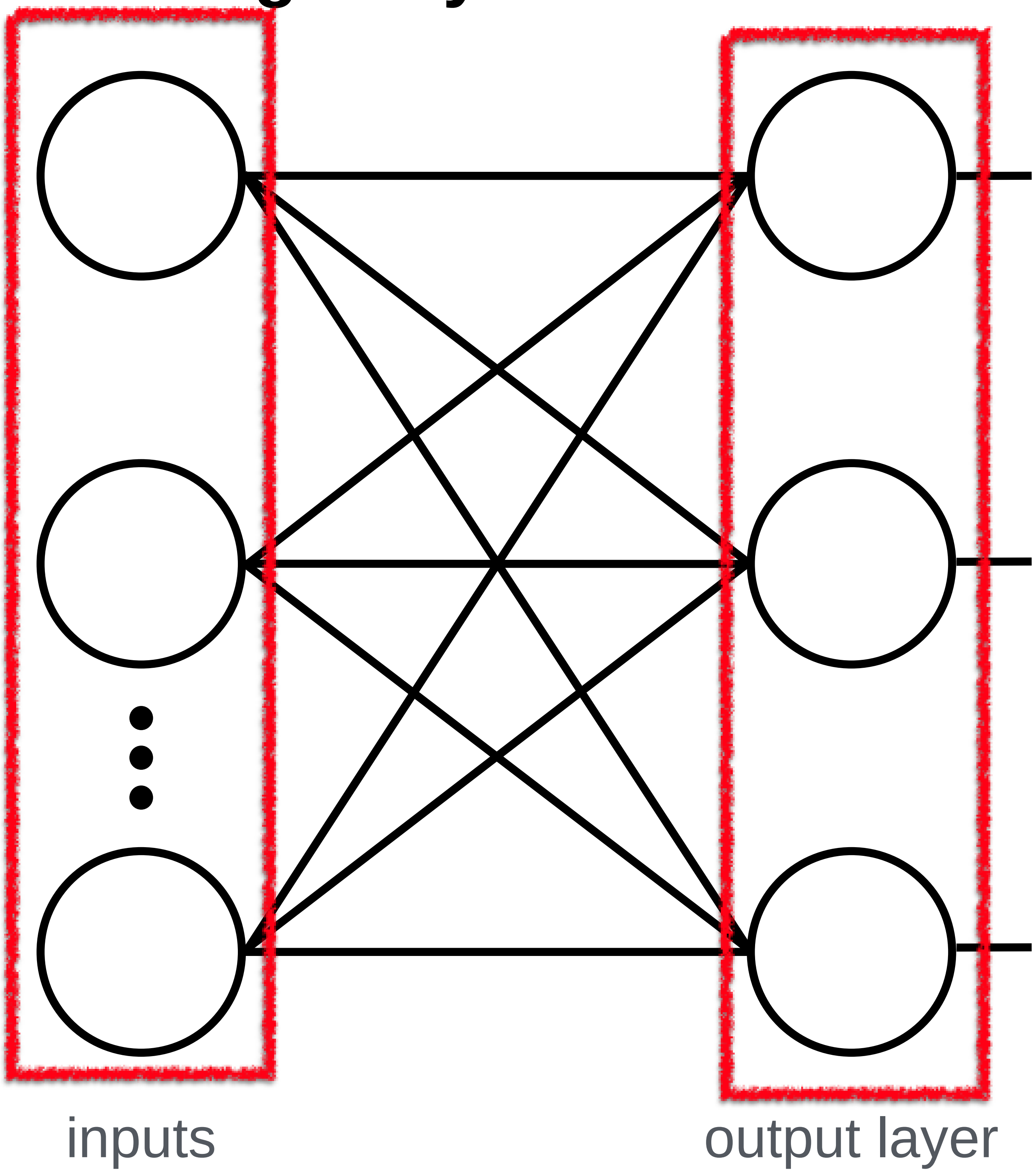
fully-interconnected two-layer network

this is how some readings will describe networks



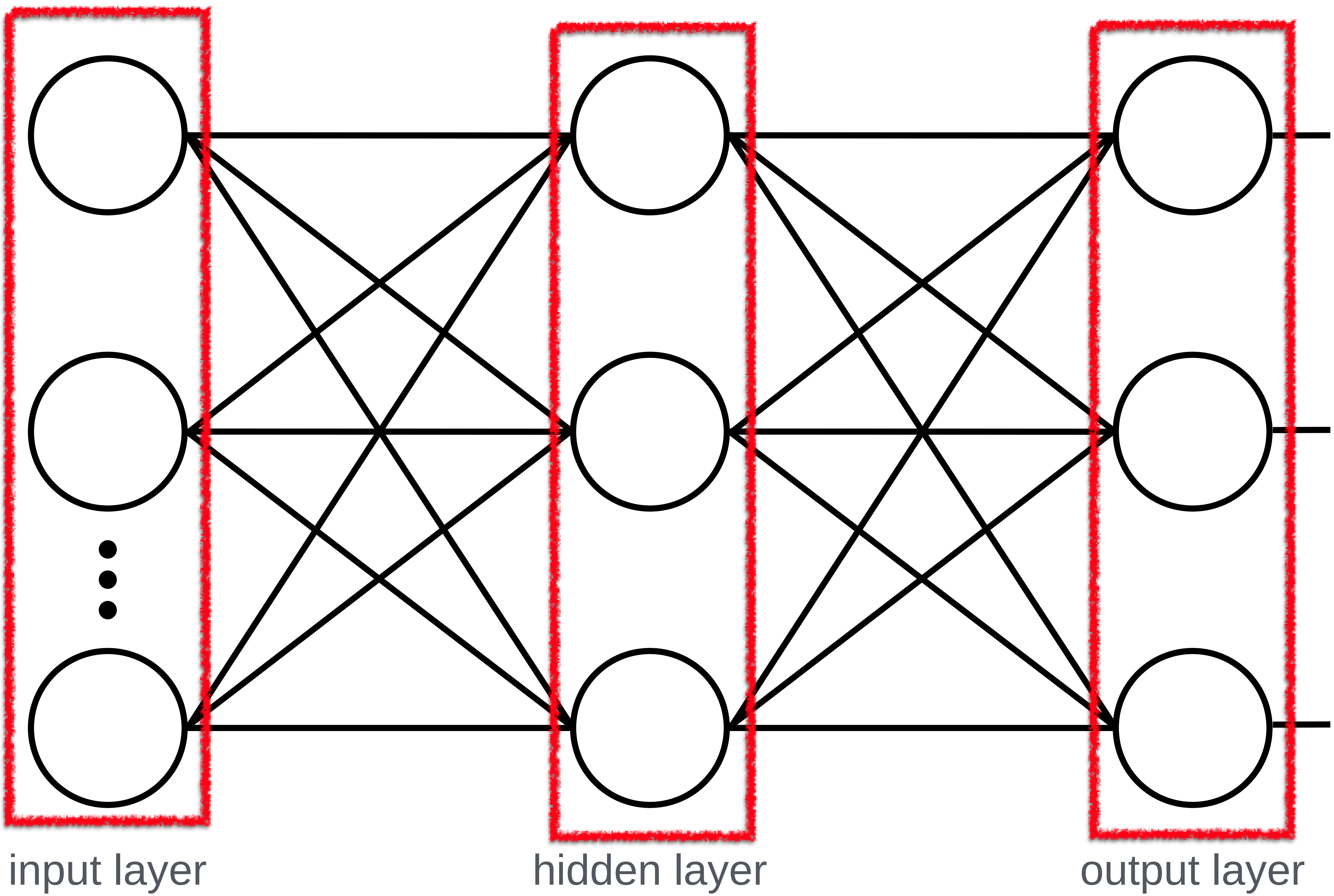
fully-interconnected
~~two-layer network~~
single-layer network

this is how **keras** will define networks



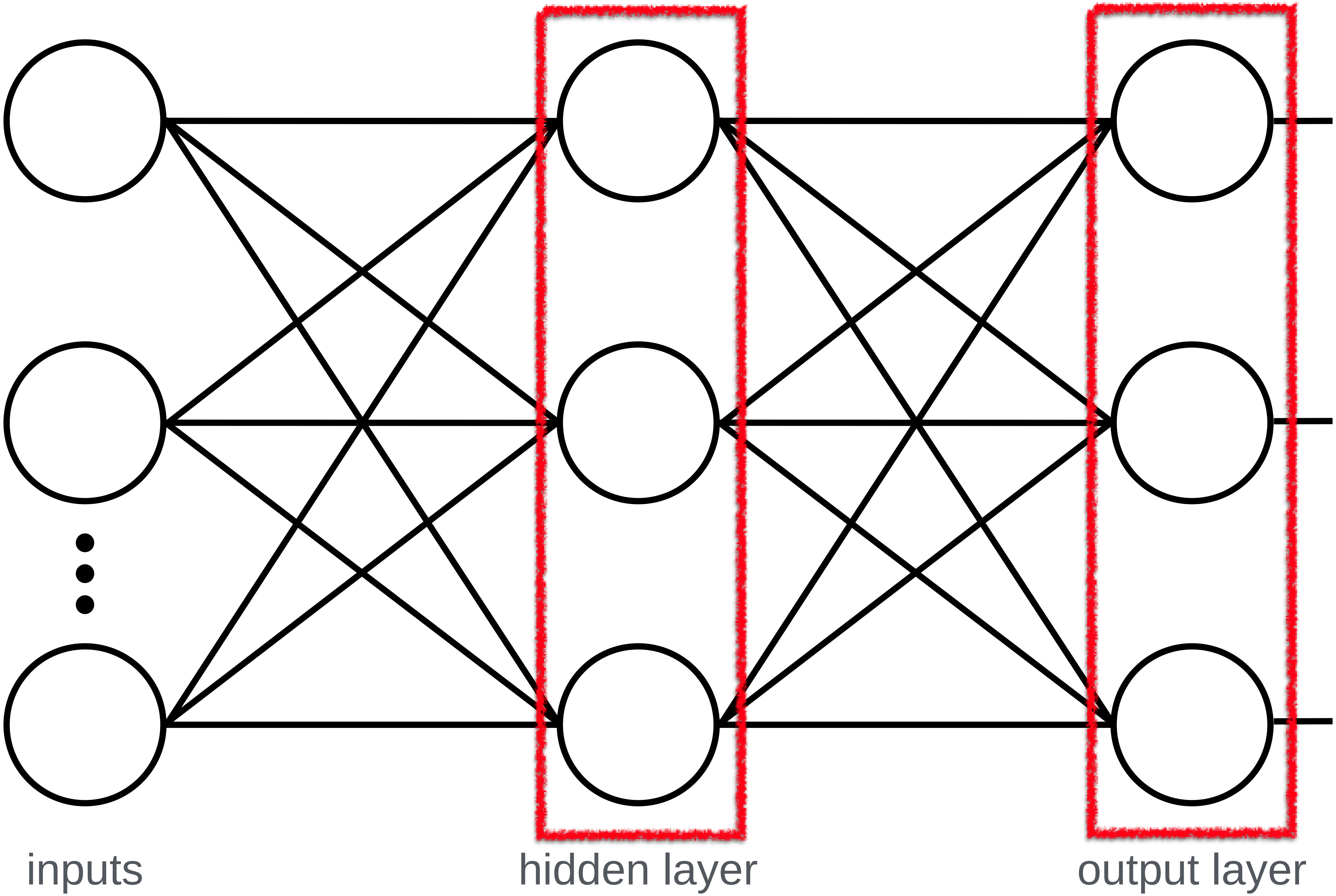
fully-interconnected three-layer network

this is how some readings will describe networks

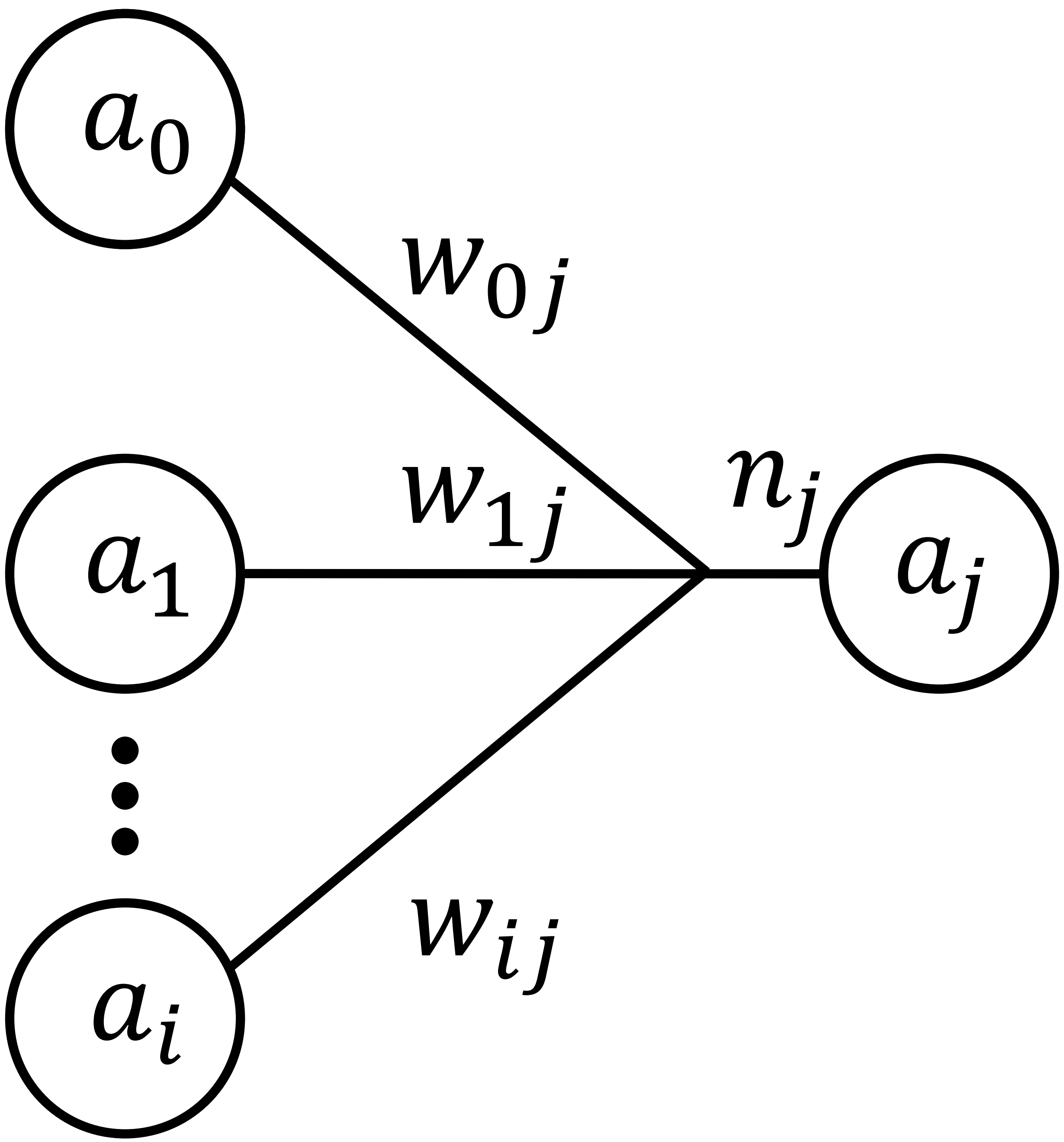


fully-interconnected
~~**three-layer network**~~
two-layer network

this is how **keras**
will define networks



bias term in net input



activation function

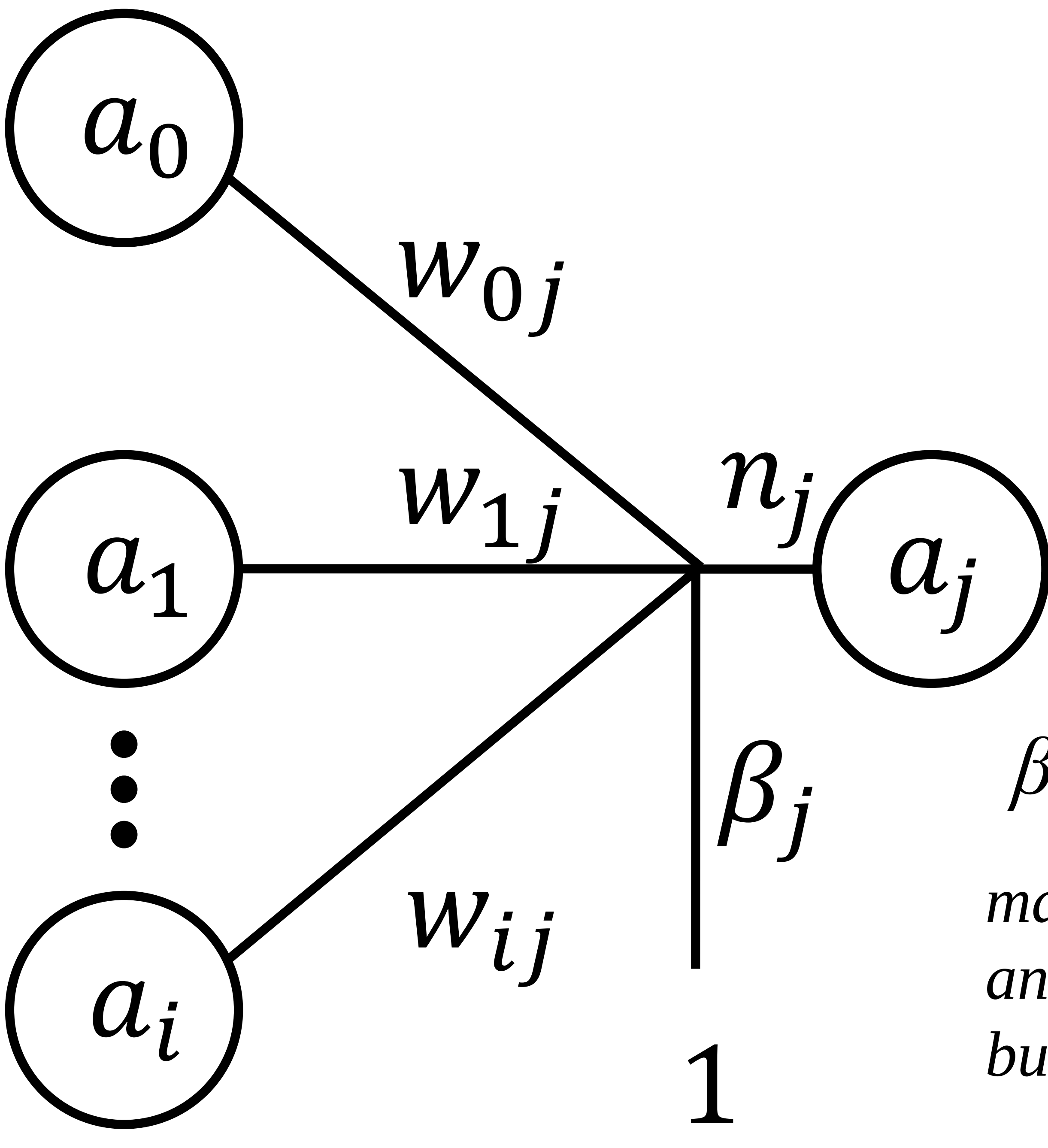
$$n_j = \sum_i a_i w_{ij}$$

$$a_j = f(n_j)$$

bias term in net input

$$n_j = \sum_i a_i w_{ij} + \beta$$

$$a_j = \frac{1}{1 + \exp(-n_j)}$$



β is the bias of a unit

mathematically, it can be treated like another weight contributing to net input, but it is kept separate in keras