Nilesh Kumar Srivastava

2022.03.11

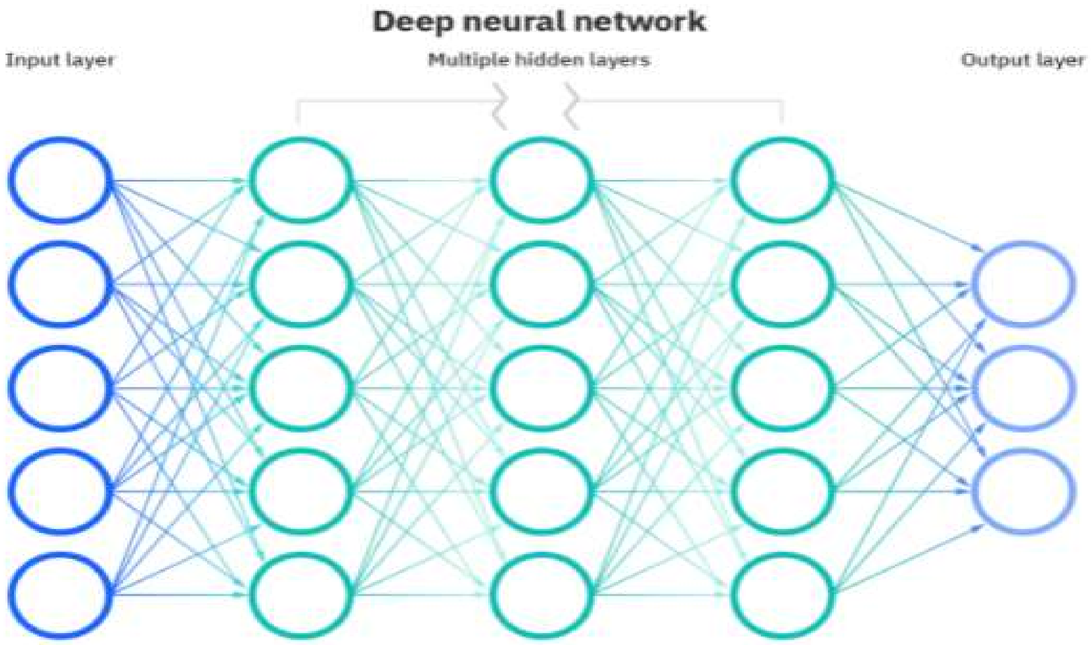
NEURAL NETWORK ARCHITECTURE

Contents

- Brief Intro of Neural Networks
- Biological Analogy
- Commonly used Activation Functions
- Architecture and Layers
- Feed Forward Computation
- Effect of Layers

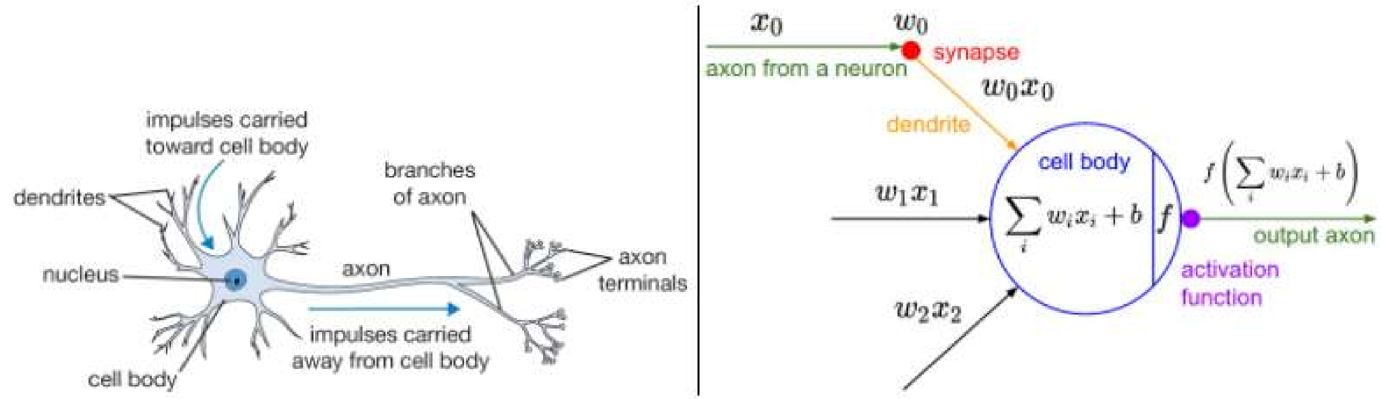
What are Neural Networks ?

- A subset of Machine Learning.
- Heart of Deep Learning Algorithms
- Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.



Biological Analogy

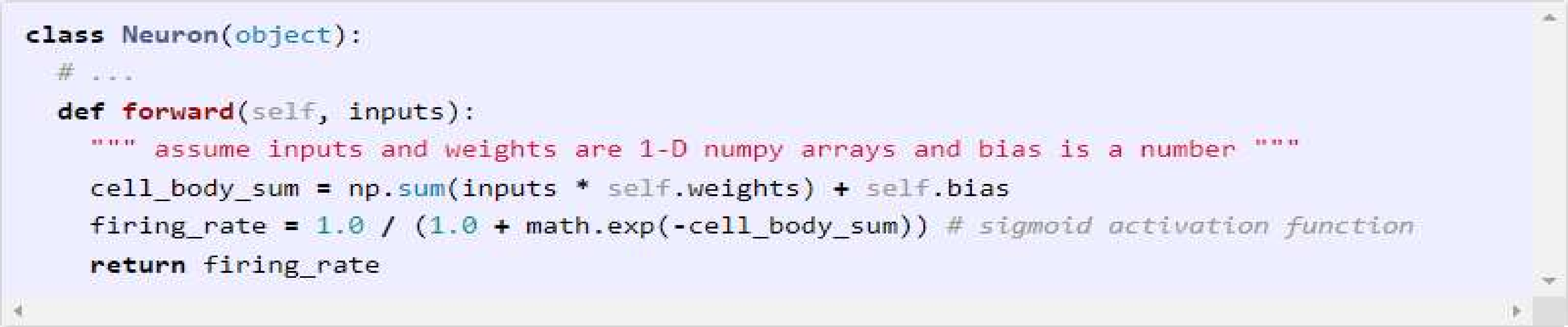
- The basic computational unit of the brain is a neuron.
- Each neuron receives input signals from its dendrites and produces output signals along its axon.
- The axon eventually branches out and connects via synapses to dendrites of other neurons.
- Input signal can be denoted as x and synaptic strength can be denoted as w. So signals travel along the axons (x) interact multiplicatively (wx) with the dendrites of the other neuron.
- We get so many such signal in a neuron and it all gets summed up. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. This firing rate can be inferred as activation function (f).



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Biological Analogy

An example code for forward-propagating a single neuron might look as follows:



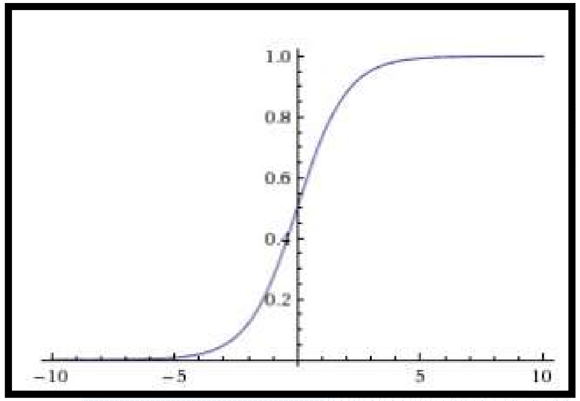
```
class Neuron(object):
 # ...
 def forward(self, inputs):
  """ assume inputs and weights are 1-D numpy arrays and bias is a number """
  cell_body_sum = np.sum(inputs * self.weights) + self.bias
  firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
  return firing_rate
```

- It’s important to stress that this model of a biological neuron is very coarse, because
  - ➢ there are many different types of neurons, each with different properties
  - ➢ The dendrites in biological neurons perform complex nonlinear computations
  - ➢ The synapses are not just a single weight, they’re a complex non-linear dynamical system

Commonly used Activation Functions

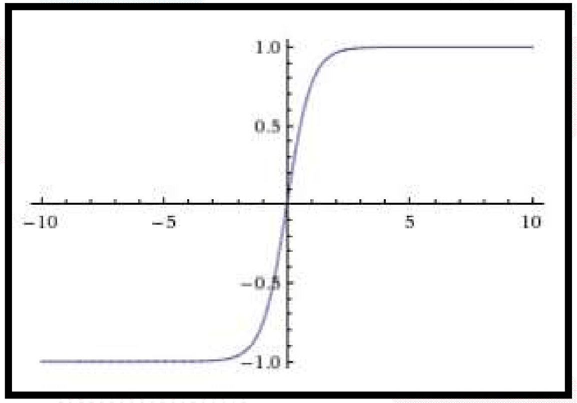
- Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it.

Sigmoid



- The sigmoid non-linearity as the mathematical form σ(x)=1/(1+e^−x)
- it takes a real-valued number and "squashes" it into range between 0 and 1
- It has two major drawbacks:
  - Sigmoid saturate and kill gradients
  - Sigmoid outputs are not zero-centered.

Tanh

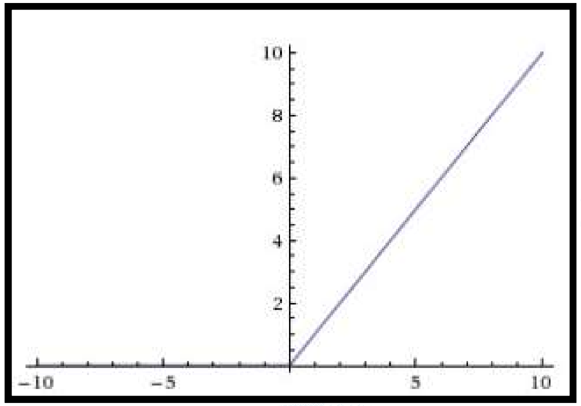


- Tanh activation is simply a scaled sigmoid activation as tanh(x)=2σ(x)−1
- Tanh squashes a real-valued number to the range [-1, 1].
- Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity.

Commonly used Activation Functions

- Rectified Linear Unit: f(x)=max(0,x)
  - (+) It was found to greatly accelerate the convergence of stochastic gradient desce nt compared to the sigmoid/tanh functions
  - (+) ReLU can be implemented by simply thresholding a matrix of activations at zero
  - (-) ReLU units can be fragile during training and can “die”
  - A large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again.

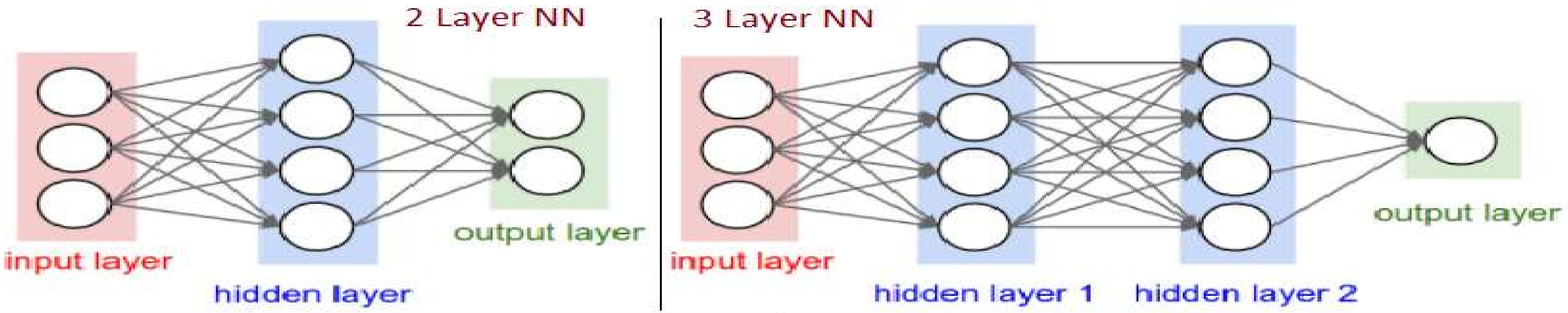
ReLU



- Leaky ReLU: f(x)=1(x<0)(αx)+1(x>=0)(x) where α is a small constant.
  - Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small positive slope.
- Maxout: max(w1x+b1,w2x+b2)
  - generalizes the ReLU and its leaky version.
  - both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have w1,b1=0). The Maxout neuro n therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its dra wbacks (dying ReLU).
  - It doubles the number of parameters for every single neuron, leading to a high total number of parameters.

Architecture and Layers

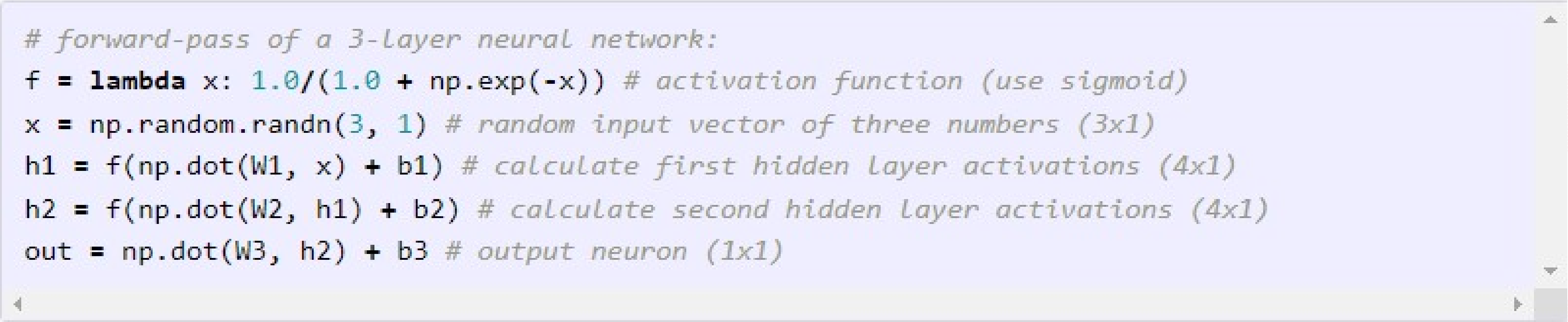
- Also referred as “Artificial Neural Networks” (ANN) or “Multi-Layer Perceptrons” (MLP).
- Neural Networks are modeled as collections of neurons that are connected in an acyclic graph
- Most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections



- Sizing neural networks:
  - The first network (left) has 4 + 2 = 6 neurons (not counting the inputs), [3 x 4] + [4 x 2] = 20 weights and 4 + 2 = 6 bia ses, for a total of 26 learnable parameters.
  - The second network (right) has 4 + 4 + 1 = 9 neurons, [3 x 4] + [4 x 4] + [4 x 1] = 12 + 16 + 4 = 32 weights and 4 + 4 + 1 = 9 biases, for a total of 41 learnable parameters.

Feed Forward Computation

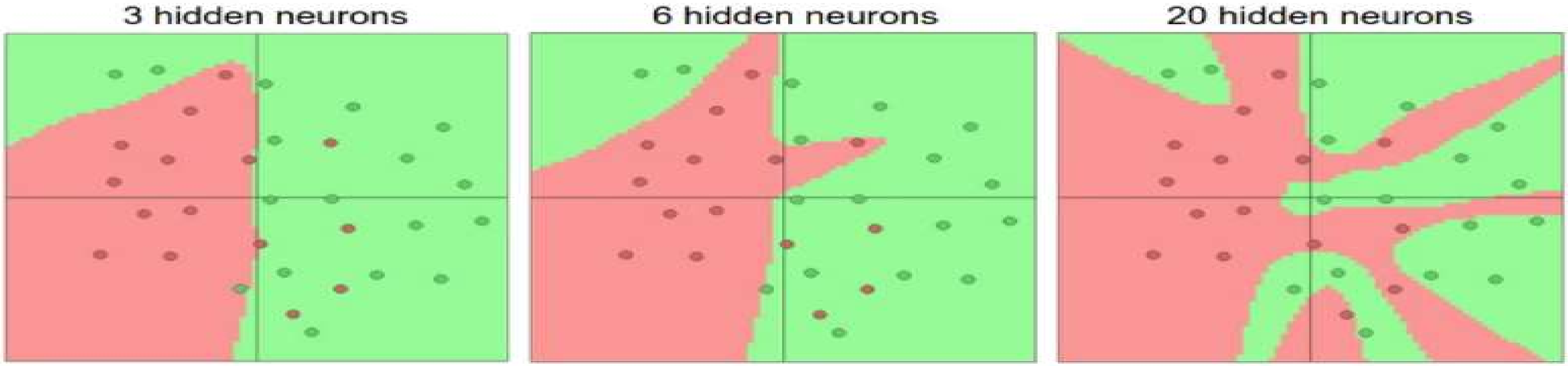
- Neural Networks have repeated matrix multiplications interwoven with activation function.
- Neural Network structure makes it very simple and efficient to evaluate Neural Networks using matrix vector operations
- Working with the example three-layer neural network
  - the input would be a [3x1] vector
  - first hidden layer's weights W1 would be of size [4x3], and the biases for all units would be in the vector b1, of size [4x1]
  - Similarly, W2 would be a [4x4] matrix that stores the connections of the second hidden layer, and W3 a [1x4] matrix for the last (output) layer.



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Effect of Layers

- As we increase the size and number of layers in a Neural Network, the capacity of the network increases.



- Neural Networks with more neurons can express more complicated functions.
- Overfitting occurs when a model with high capacity fits the noise in the data instead of the (assumed) underl ying relationship.
- Smaller neural networks can be preferred if the data is not complex enough to prevent overfitting.
- In practice, it is always better to use Regularization methods to control overfitting instead of the number of neurons.

Summary

- We introduced a very coarse model of a biological neuron.
- We discussed several types of activation functions that are used in practice, with ReLU being the most common choice.
- We introduced Neural Networks where neurons are connected with Fully-Connected layers where neurons in adjacent layers have full pair-wise connections, but neurons within a layer are not connected.
- We saw that this layered architecture enables very efficient evaluation of Neural Networks based on matrix multiplications interwoven with the application of the activation function.
- We discussed the fact that larger networks will always work better than smaller networks, but their higher model capacity must be appropriately addressed with regularization, or they might overfit.

References

- https://cs231n.github.io/neural-networks-1/
- https://www.ibm.com/cloud/learn/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another