CS195: Computer Vision

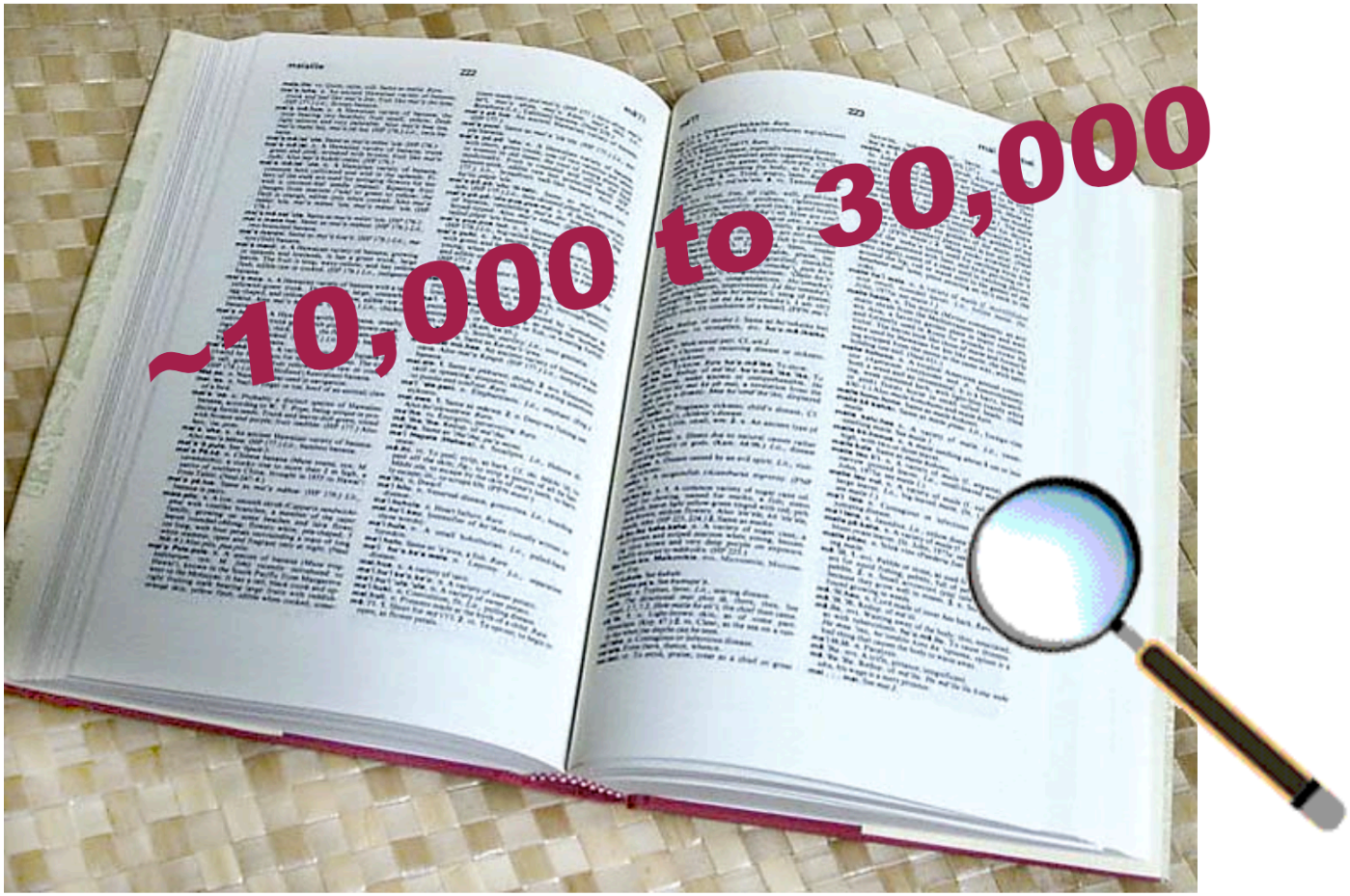
Object Recognition
Image Classification and Neural Networks (NN)
NN type: Multilayer Perceptron (MLP)
PyTorch Basics

Monday, September 9th, 2024

Drake
UNIVERSITY

Object Recognition

How many object categories are there?

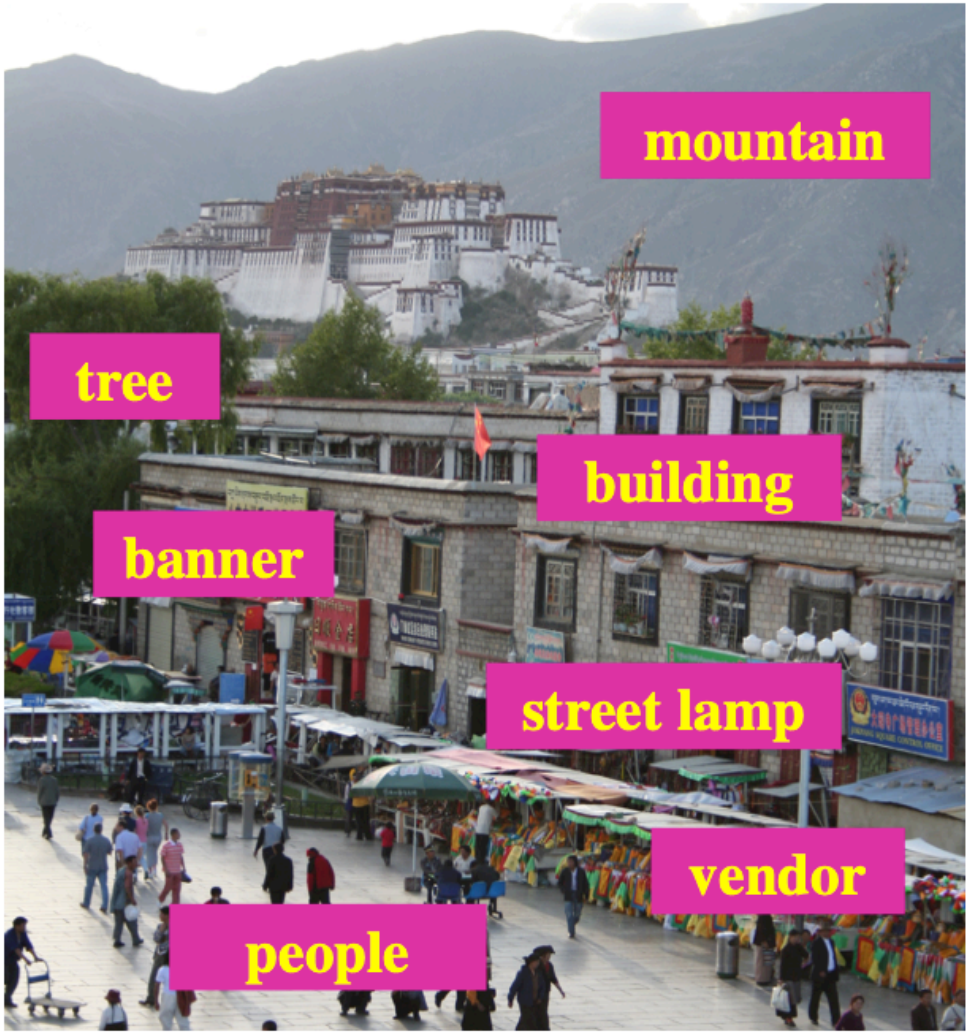


How many object categories are there?

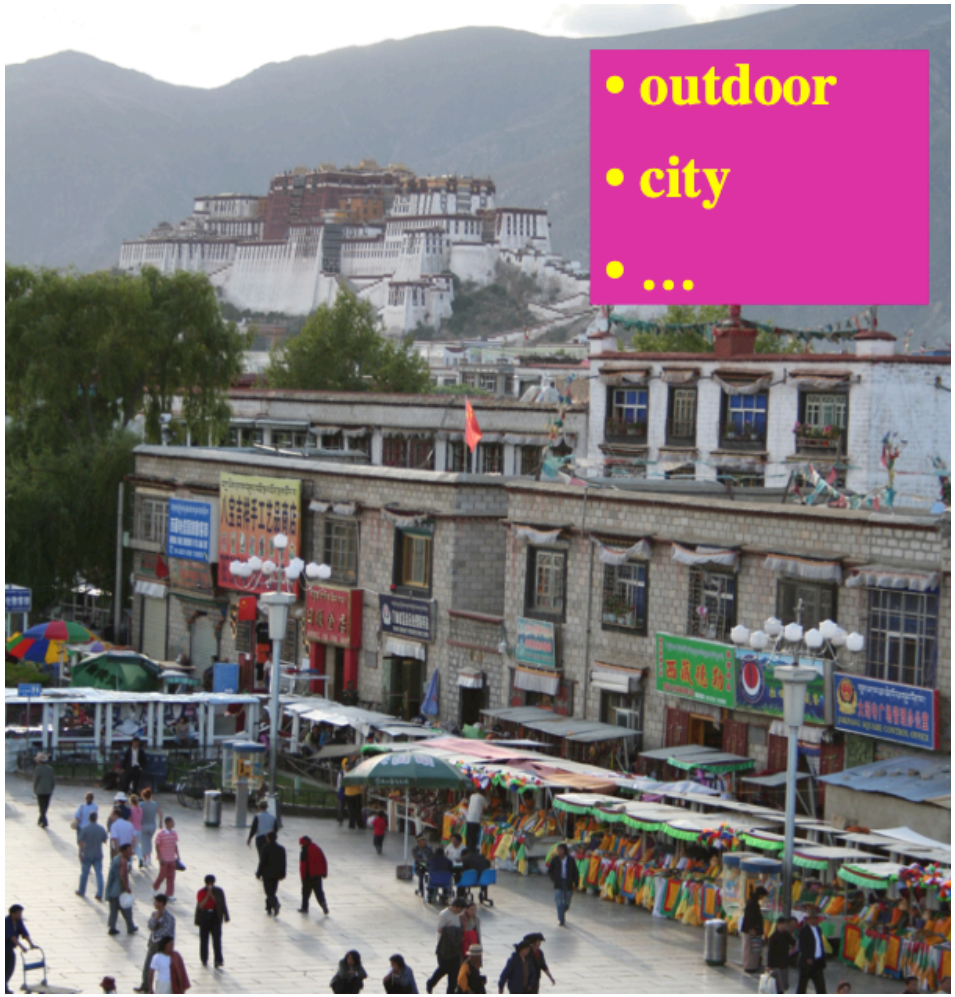
What does object recognition involve?

Classification: is that cropped portion a lamp post?

Multi-class classification: what objects are out there?

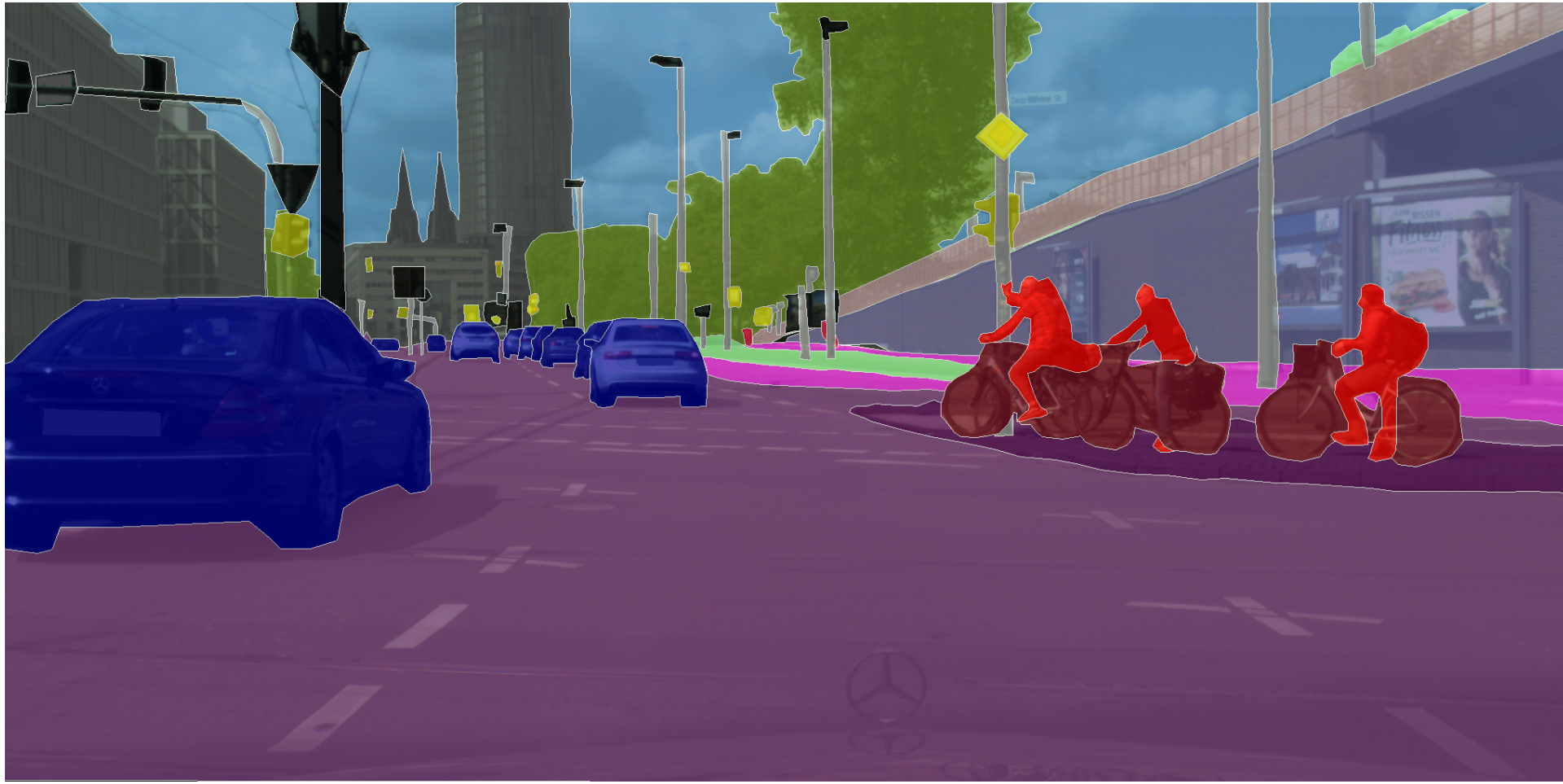


Scene and context classification



Detection: where are the people?

Semantic segmentation: what are the objects at the pixel level?

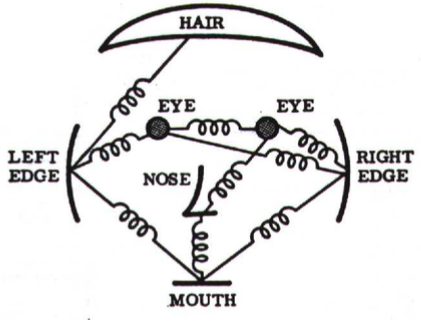


Recognition Models

- Different types of models can be used to for object recognition. Below are some classical object recognition models:
  - **Color histogram model:** is the simplest representing the objects by its color histogram

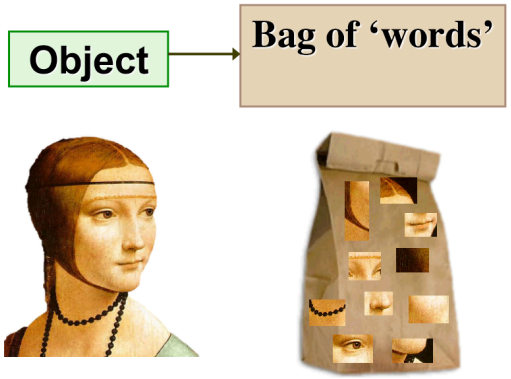
  - **Local feature model:** combining *local* appearance, spatial constraints, invariants, and classification techniques from machine learning

  - **Parts and shape model:**
    - modeling an object as a set of parts
    - relative locations between parts
    - appearance of part

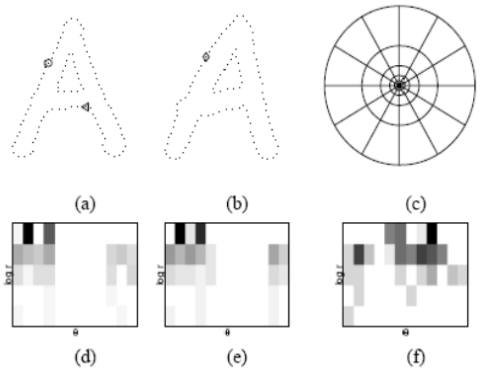


Recognition Models

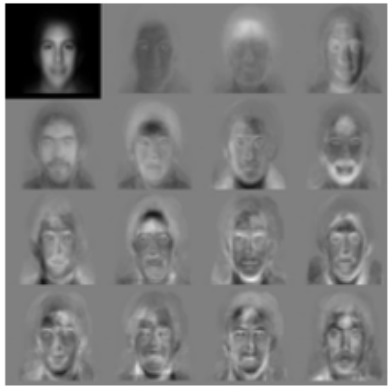
- Different types of models can be used to for object recognition. Below are some classical object recognition models:
  - **Bag of feature model:** an object is a collection of patches



  - **Shape context:**



  - **Appearance based object recognition:**
    - holistic representation of images, instead of features, use the whole image, eg, PCA-based techniques such as Eigen-Face



Current focus is on object classification using deep learning models

- Determine whether a given photo contains a ‘Dog’, ‘Cat’, ‘Horse’, or another specific object.

Current focus is on object classification using deep learning models

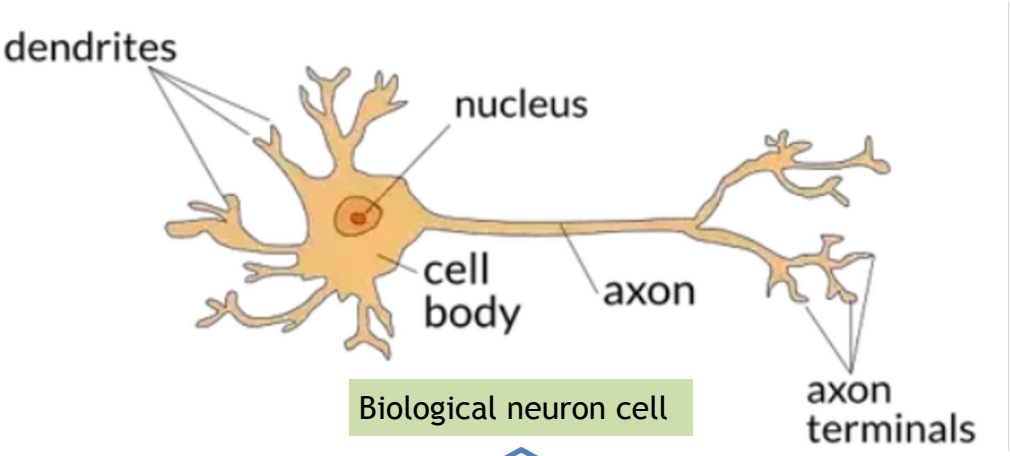
- Various types of **deep learning model based** — more specifically neural networks — can be employed to categorize an image into distinct classes

    - **Multilayer Perceptrons (MLP):** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers
    - **Convolutional Neural Network (CNN):** good for computer vision (CV) tasks
    - **Transformers:** rising star DL model; it had its inception in Natural Language Processing domain but is now gradually taking over all other AI domains such as Computer Vision, Audio/Speech, Robotics
    - **Very Recent additions (Early 2024):**
        - Mamba Network
        - Kolmogorov Arnold Networks (KAN)

Today's Agenda

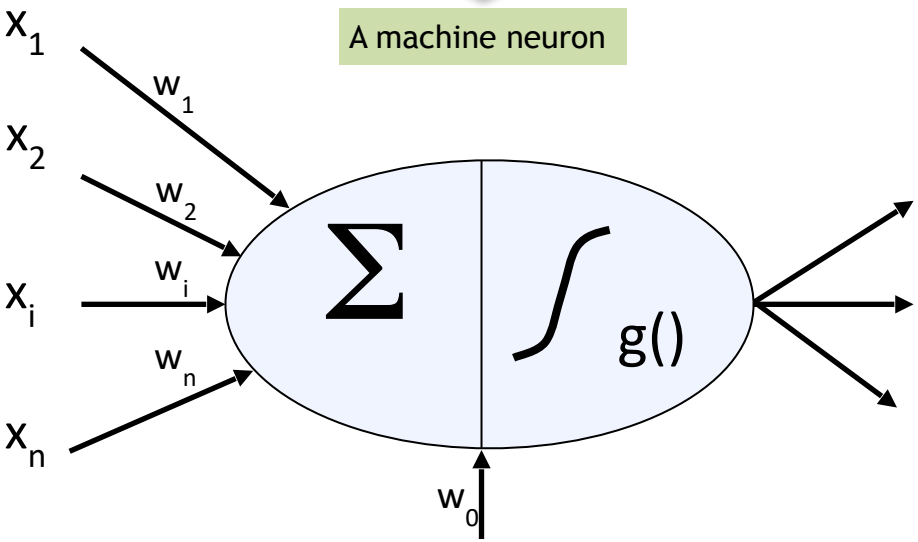
- Multilayer Perceptrons (MLP)

Inspiration: Neuron Cells



Biological neuron cell

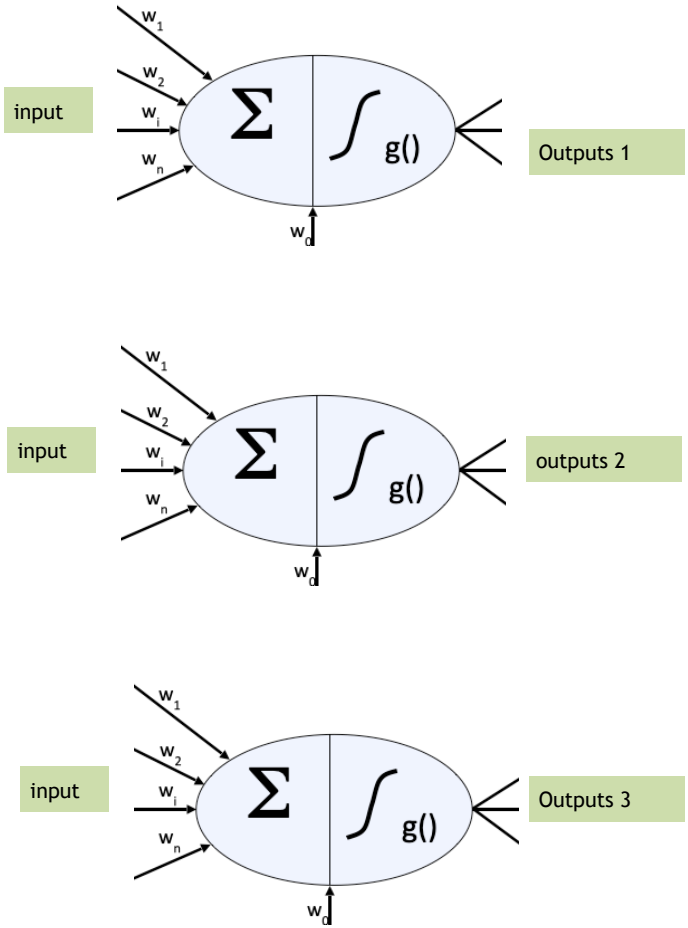
A machine neuron



Inputs

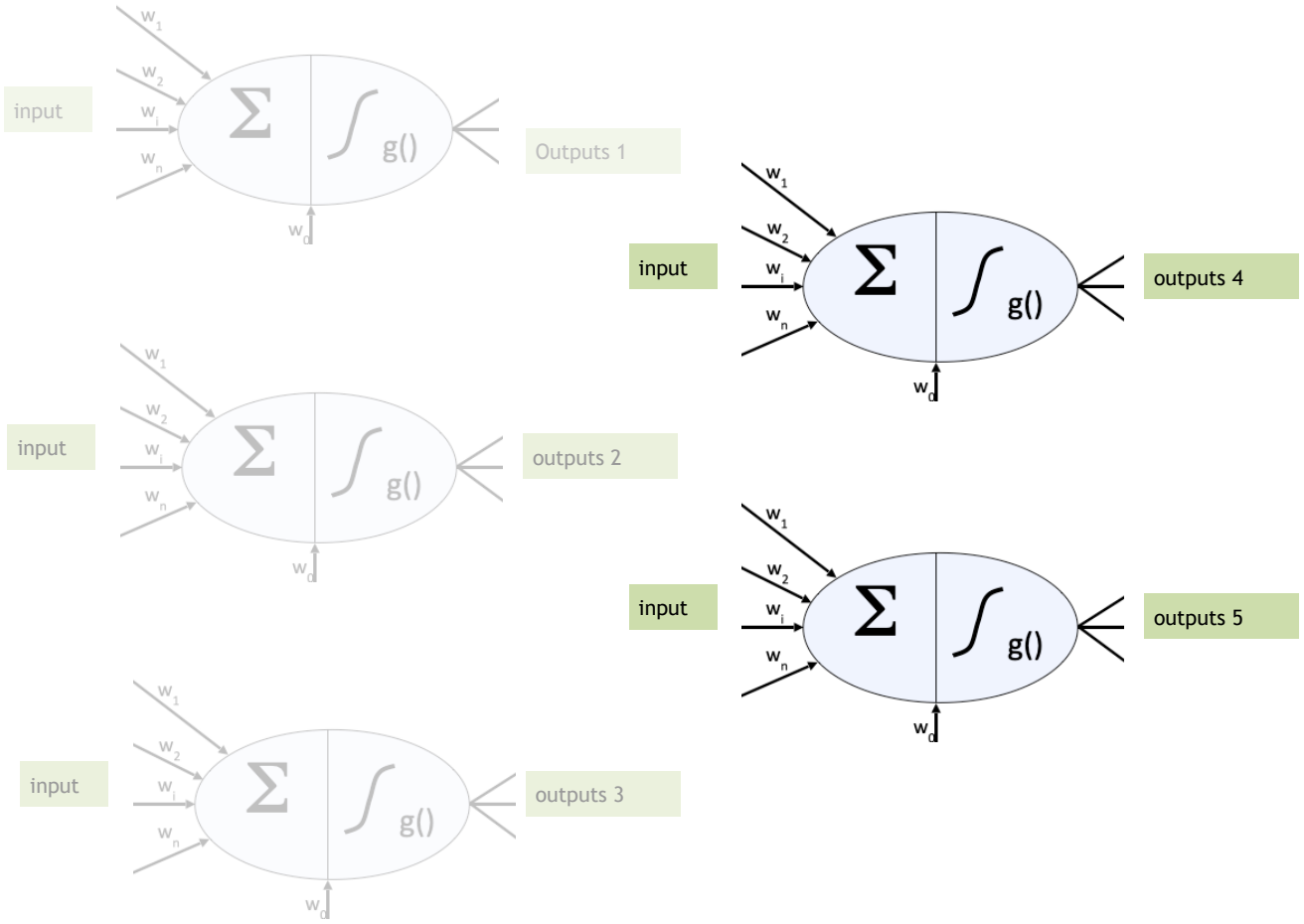
Outputs

Add three neurons in the first Layer

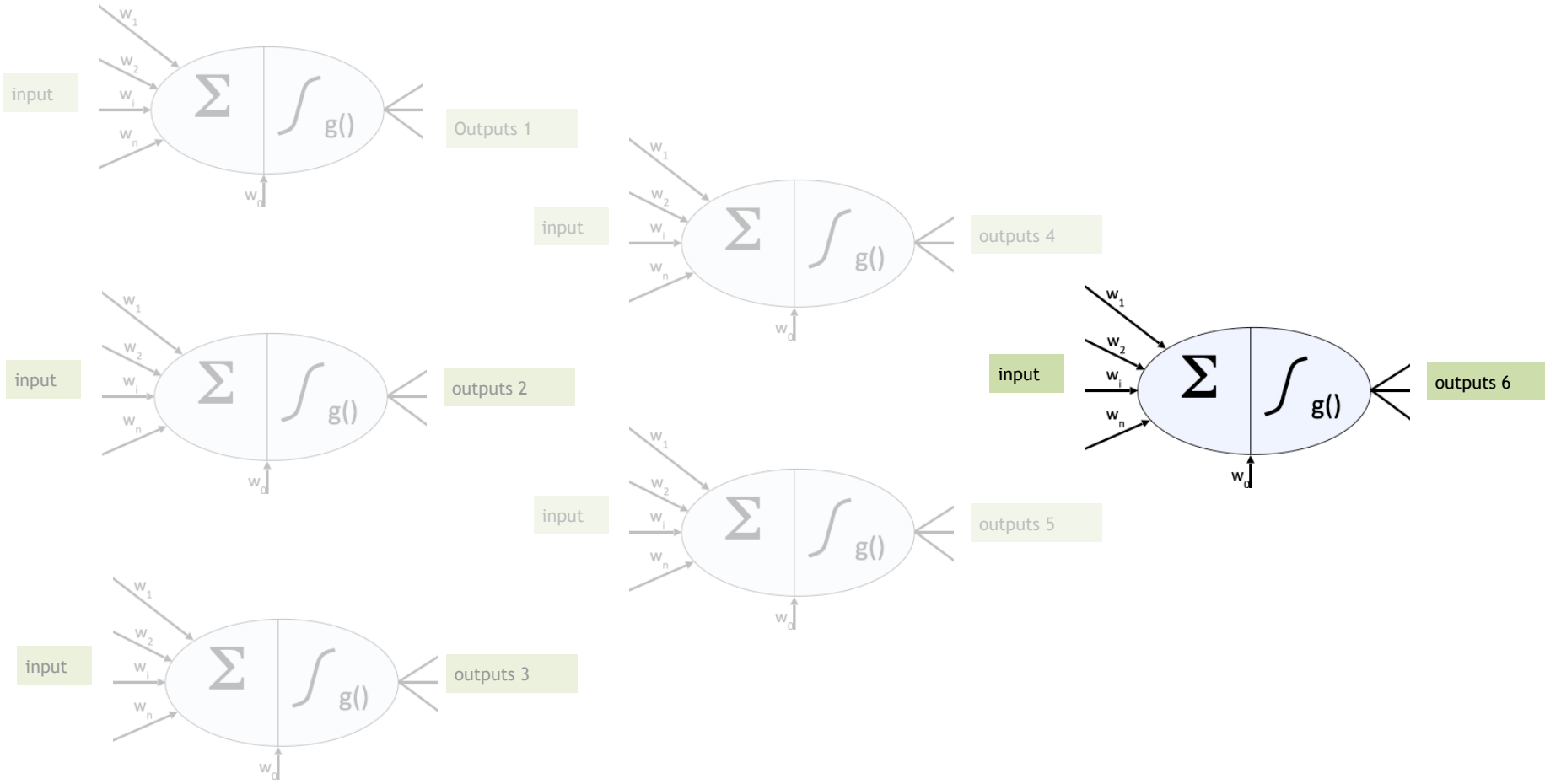


Layer 1

Add a two more neuron in the second layer

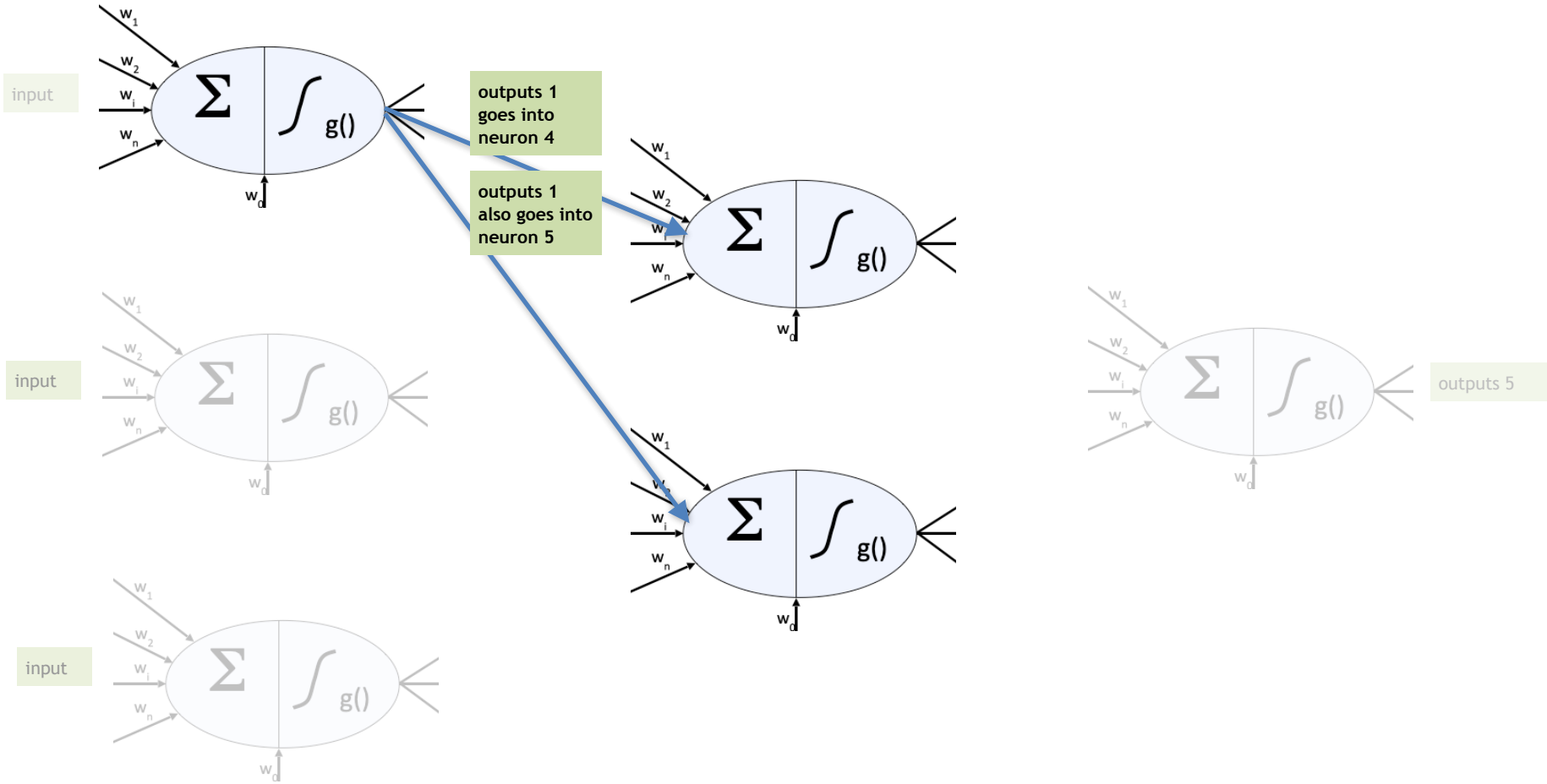


Add a two more neuron in the third layer



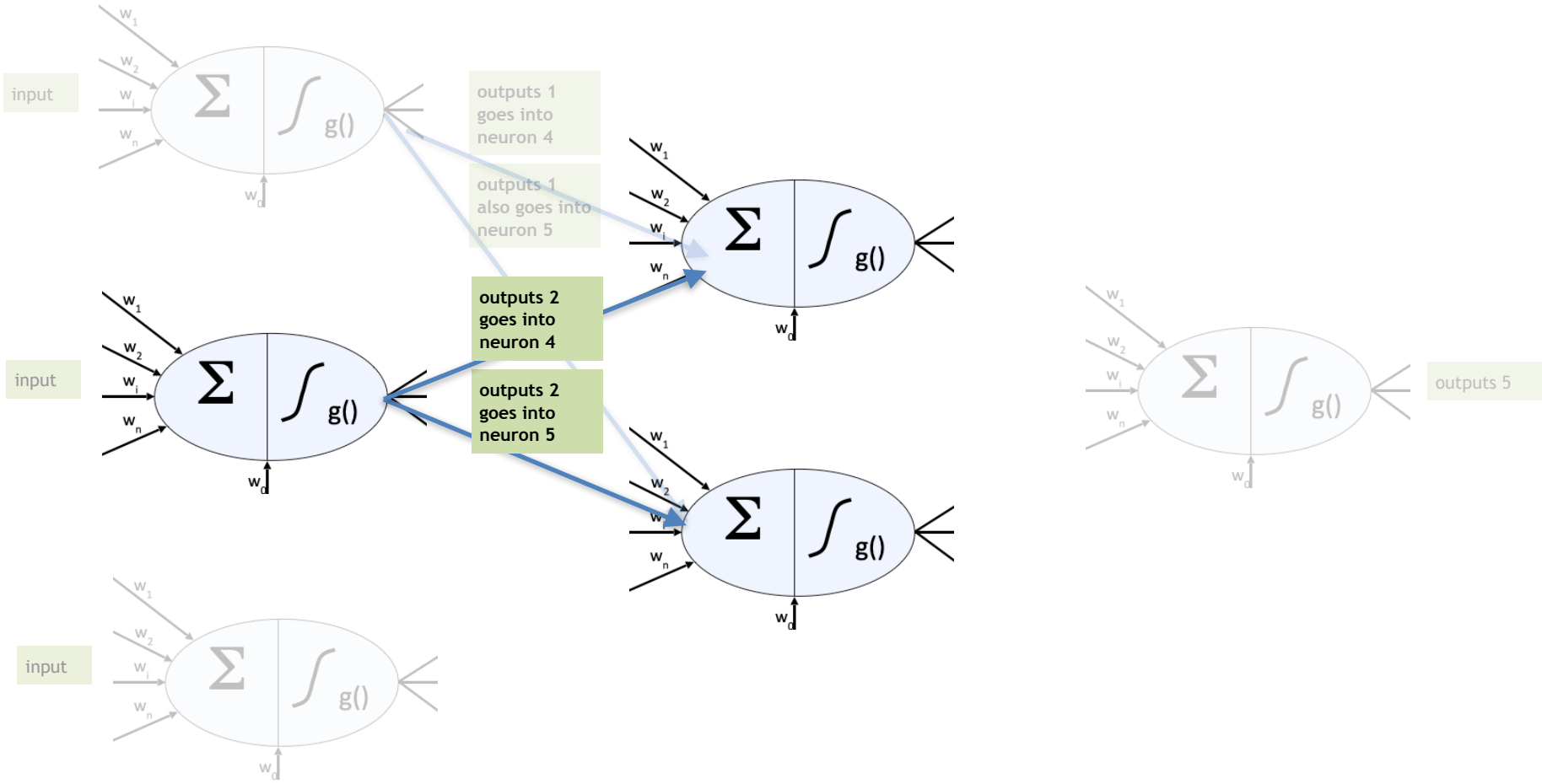
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



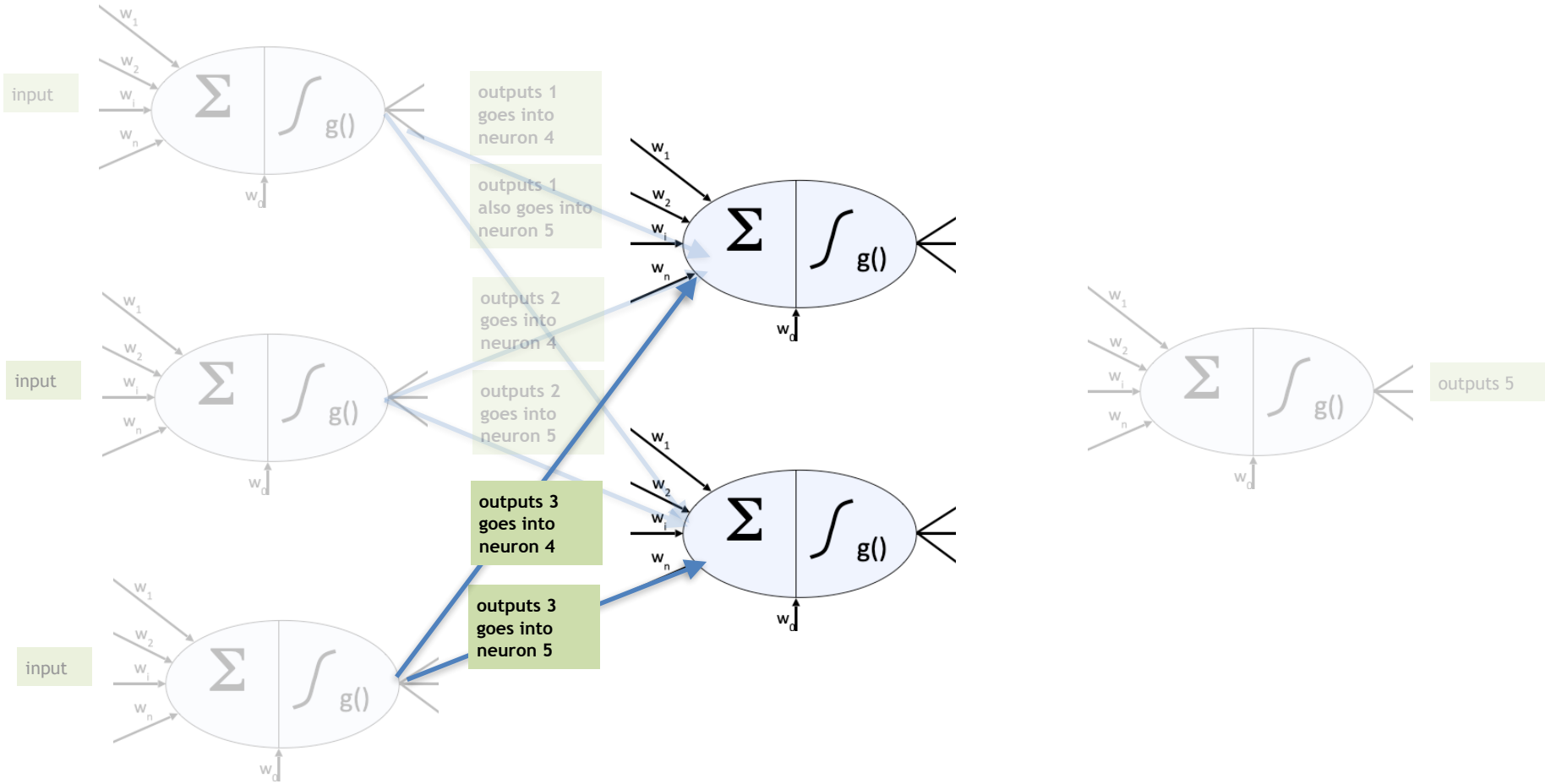
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



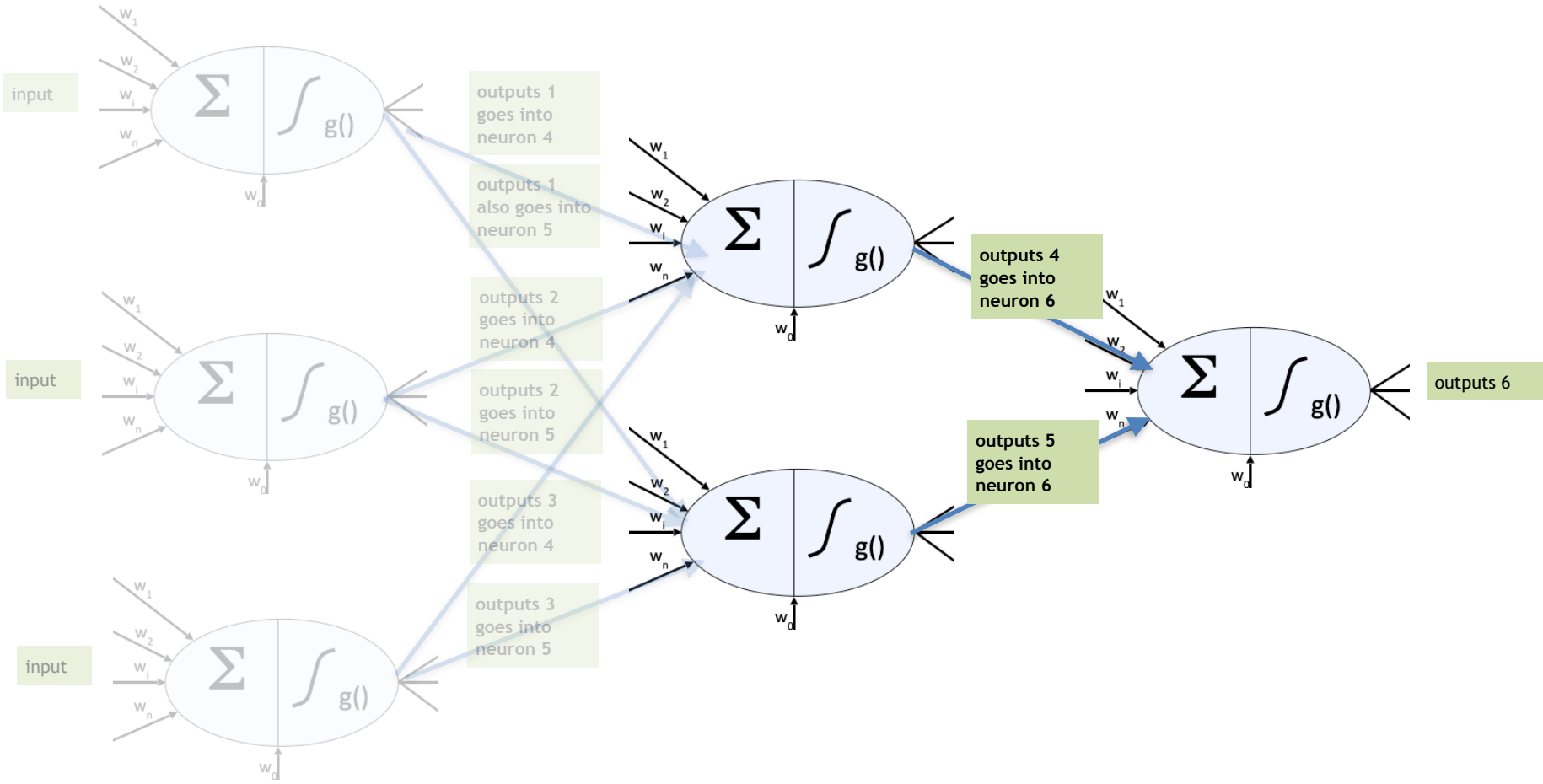
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



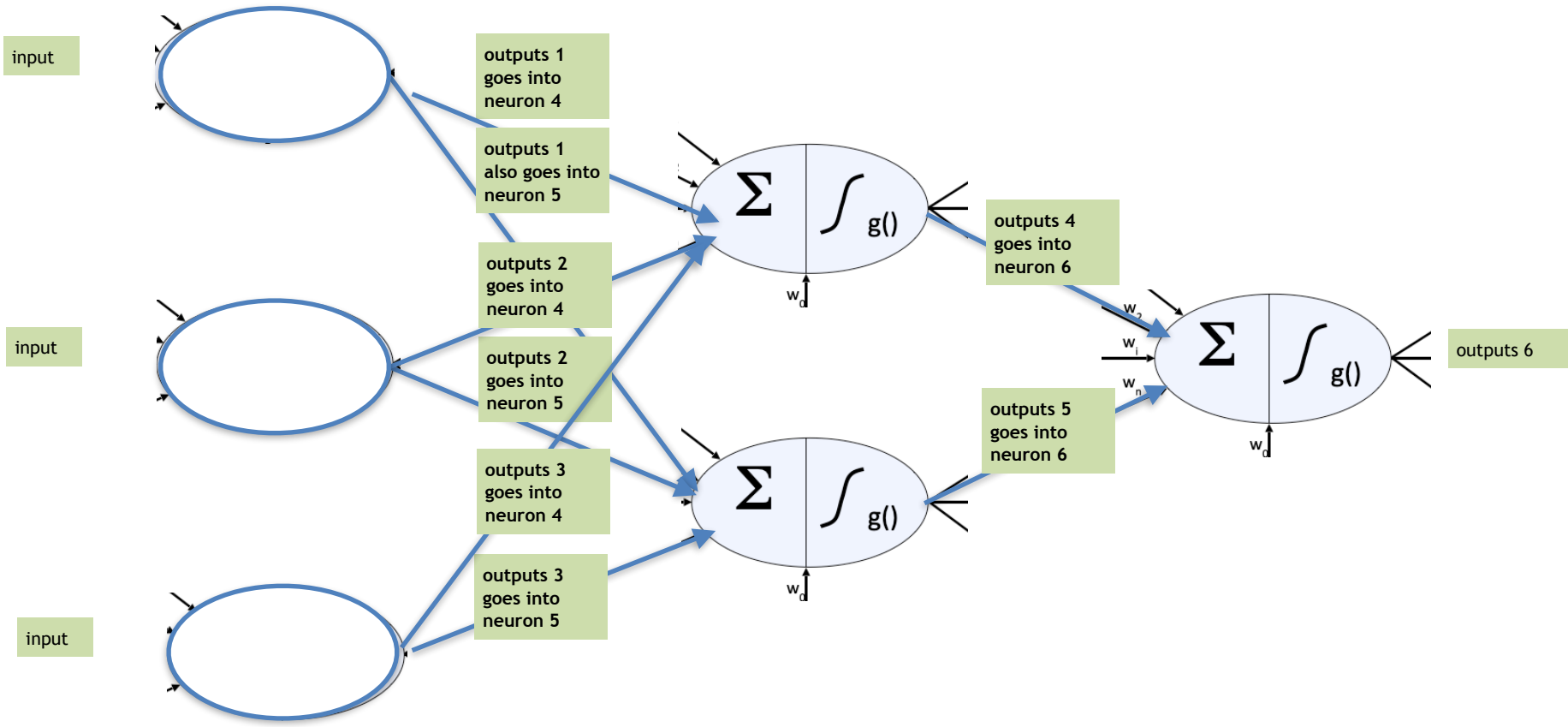
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 2** and **Layer 3**



1-Hidden Layer Neural Network

- We created our first multilayer perceptron (MLP)
- Any layers in between input layer and output layer are called hidden layers
- Hence this MLP can also be called 1-hidden layer neural network



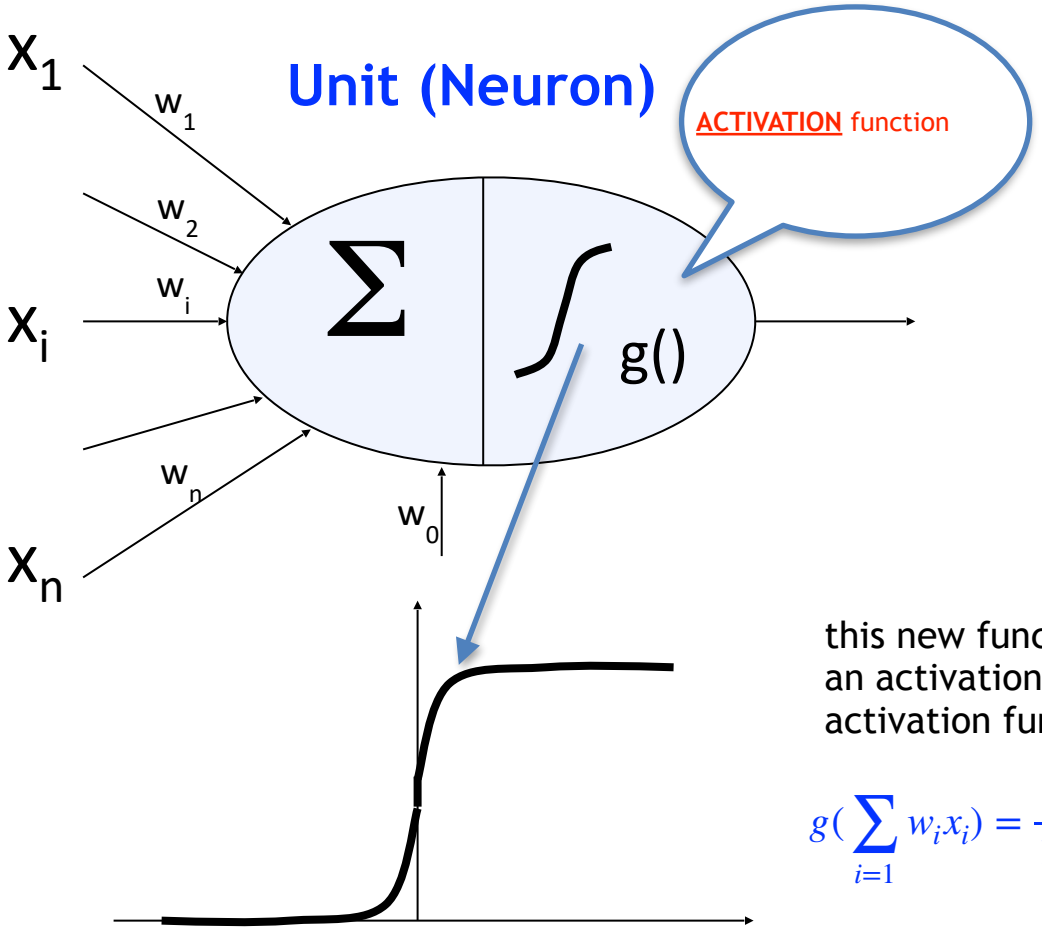
Neurons in the input layer do not have activation functions

Input Layer

Hidden Layer

Output Layer

Activation functions for a neuron



this new function is referred to as an activation function. eg, sigmoid activation function

$$g(\sum_{i=1} w_i x_i) = \frac{1}{1 + exp(-\sum_{i=1} w_i x_i)}$$

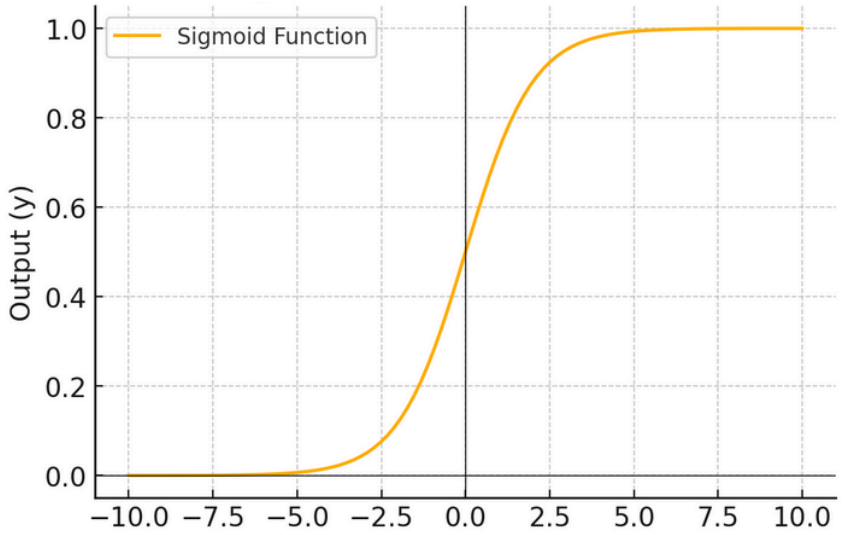
It is smooth and differentiable

$$y = f(\mathbf{x},\mathbf{w}) = g(\Sigma_{i=1,\ldots,n}\, w_i\, x_i)$$

Sigmoid (a.k.a. Logistic Function) Activation Function

It is smooth and differentiable

Sigmoid: $$g(u) = \frac{1}{1 + exp^{-u}}$$

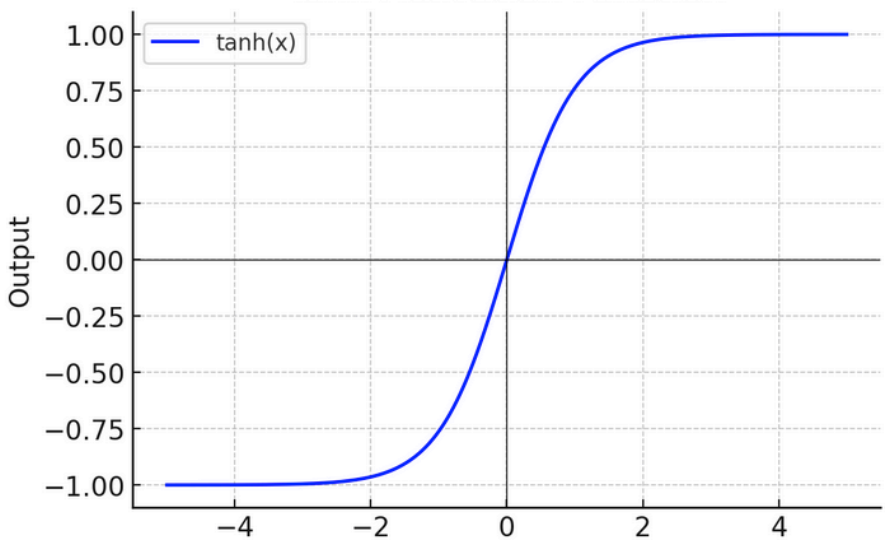


- Properties:
  - Activation value ranges between 0 to 1
  - Strictly increasing positive value for positive numbers with larger magnitude
  - For a large range of positive input numbers, it compresses them into a smaller range of values.

Tanh Activation Function

It is smooth and differentiable

tanh: $g(u) = \frac{exp^{u} - exp^{-u}}{exp^{u} + exp^{-u}}$

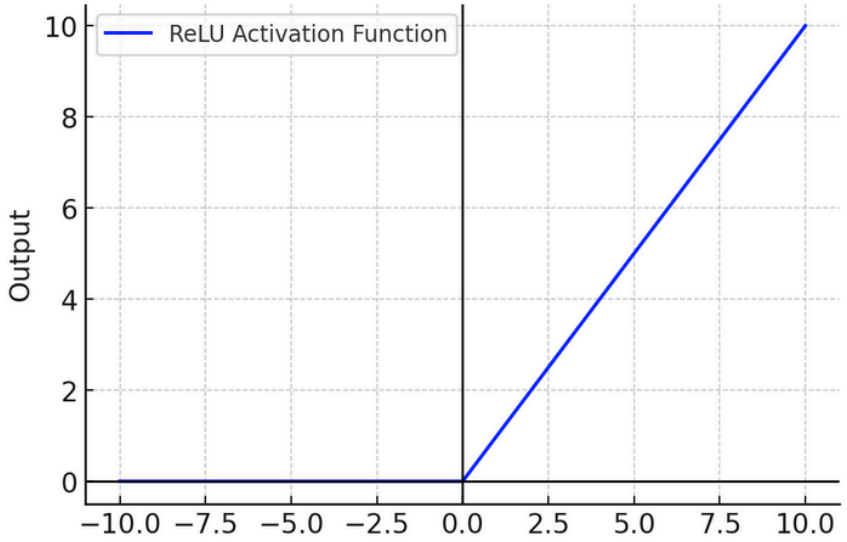


- Properties:
  - Activation value ranges between -1 to 1
  - Strictly increasing output values for any inputs
  - For a large range of positive or negative input numbers, it compresses them into a smaller range of values.

Rectified Linear Unit (ReLU) Activation Function

It is smooth and differentiable

ReLU: $g(u) = max(0,u)$



- Properties:
  - It always produces a positive output, reducing any negative input to zero
  - For positive values it will never saturate (proportionally go higher)
  - Strictly increasing output values for any inputs
  - Not symmetric around zero
  - Introduces sparse activity due to the squashing of zero for negative inputs
  - Better for gradient flow (during training a neural network), trains faster
  - Simplicity of computation (just max() function and no exponential calculation)

- Other variations: Parametric ReLU, Leaky ReLU

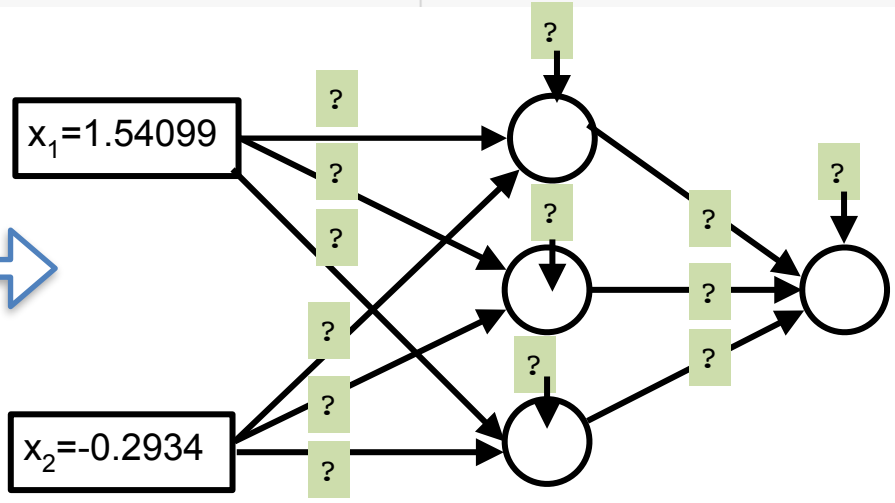
Generate Random Samples for the MLP Below

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

```
# let's generate 4 random samples of (x1, x2) for the above network
torch.manual_seed(0)
random_X = torch.randn(4,2) # you could imagine that these are pairs of (x1, x2) as shown in the above table
print('random_X = \n', random_X.numpy())
input_feature_size = random_X.shape[1] # number of columns corresponds to feature dimension
print('\n\ninput feature dimension: ', input_feature_size)
```

Sample#	x_1	x_2
1	1.5409961	-0.2934289
2	-2.1787894	0.56843126
3	-1.0845224	-1.3985955
4	0.40334684	0.83802634

Consider doing forward pass for this example



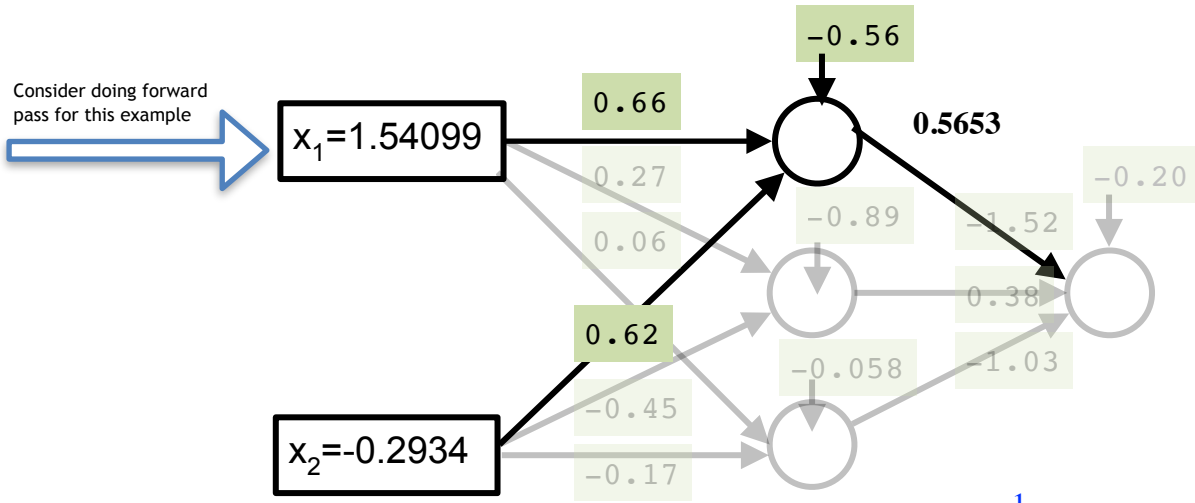
Important Design Questions for MLP

- Each of these questions need to be answered before you set up your **multilayer perceptron**
  - Q1: how many hidden layers should be there? (depth)
  - Q2: how many neurons should be in each layer? (width)
  - Q3: how many dense connections should be there in between each adjacent layers
  - Q4: what should the activation be at each of the intermediate layers?
    - `sigmoid(), tanh(), rectified-linear-unit(), etc`
  - Q5: what should be activation of the final layer
    - `depends the task` *`classification`* `(sigmoid(), softmax()) vs.` *`regression`*

Forward Pass in our Multilayer Perceptron (MLP)

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



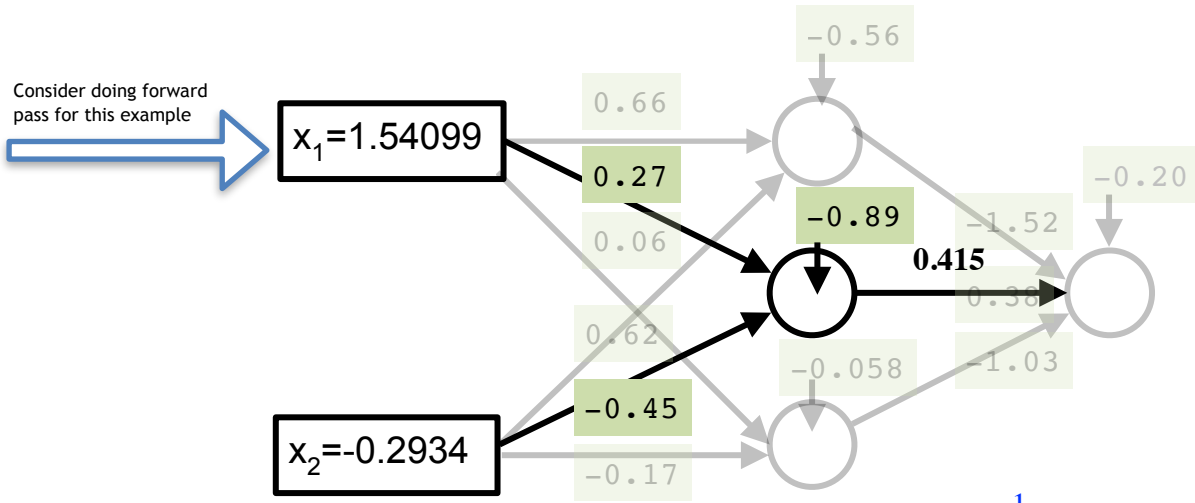
$$\mathbf{w}^T\mathbf{x} = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.56 \quad 0.66 \quad 0.62] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.56) + 1.54 * 0.66 + (-0.293) * 0.66 = 0.263$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^T\mathbf{x}}} = \frac{1}{1 + exp^{-0.263}} = 0.5653$$

Forward Pass in our Multilayer Perceptron (MLP)

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



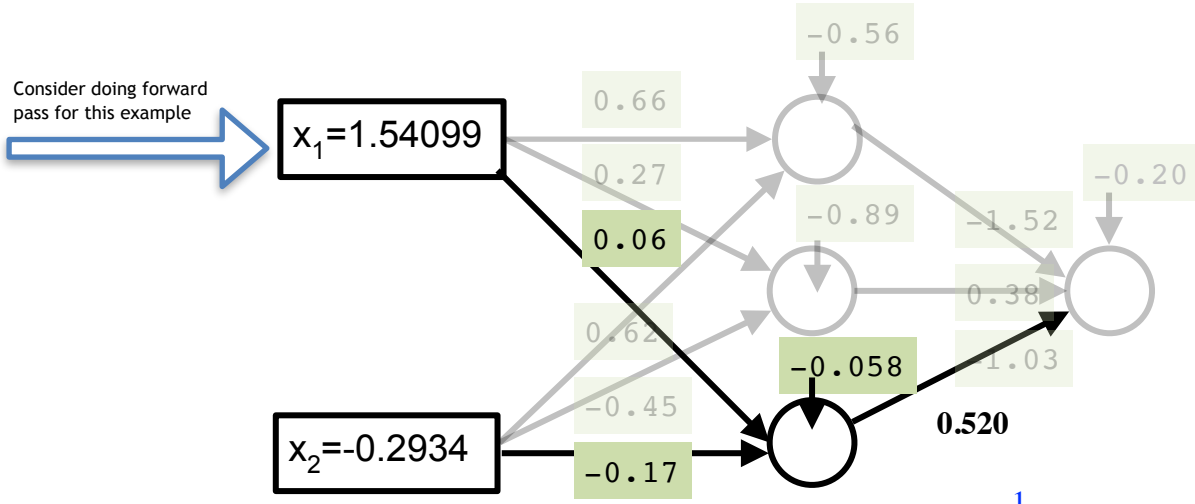
$$\mathbf{w}^T\mathbf{x} = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.89 \quad 0.27 \quad -0.45] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.89) + 1.54 * 0.27 + (-0.293) * (-0.45) = -0.34$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^T\mathbf{x}}} = \frac{1}{1 + exp^{-(-0.34)}} = 0.415$$

Forward Pass in our Multilayer Perceptron (MLP)

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



$$\mathbf{w^T x} = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.058 \quad 0.06 \quad -0.17] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.058) + 1.54 * 0.06 + (-0.293) * (-0.17) = 0.084$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w^T x}}} = \frac{1}{1 + exp^{-0.084}} = 0.520$$

Forward Pass in our Multilayer Perceptron (MLP)

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

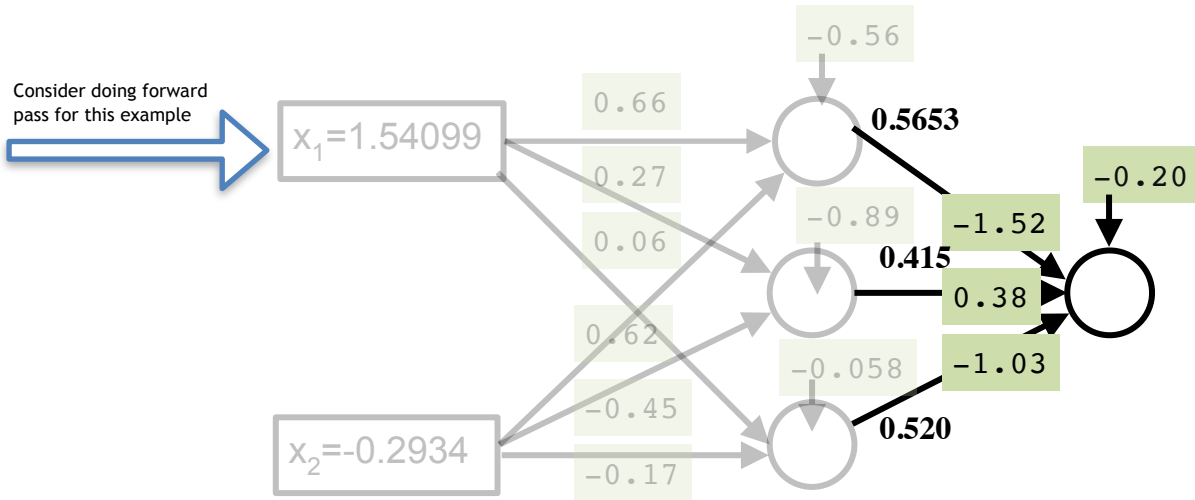
```
matrix_mult_X_and_W1 = torch.matmul(random_X[0,:], dense_connections_W1) + bias_terms_hidden
print('hidden layer input vector and weight vector dot products: \n', matrix_mult_X_and_W1.numpy())
output_hidden_layer = sigmoid_activation_hidden(matrix_mult_X_and_W1)
print('output of hidden layer: \n', output_hidden_layer.numpy())
```

```
hidden layer input vector and weight vector dot products:
 [ 0.27377588 -0.3483593   0.08554165]
output of hidden layer:
 [0.5680196  0.41378036 0.5213724 ]
```

Forward Pass in our Multilayer Perceptron (MLP)

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



$$\mathbf{w}^T\mathbf{x} = [w_0 \quad w_1 \quad w_2 \quad w_3] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = [1 \quad -1.52 \quad 0.38 \quad -1.03] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = 1*(-0.20) + (-1.52)*0.5653 + 0.38*0.415 + (-1.03)*(0.520)$$

$$= -1.437156$$

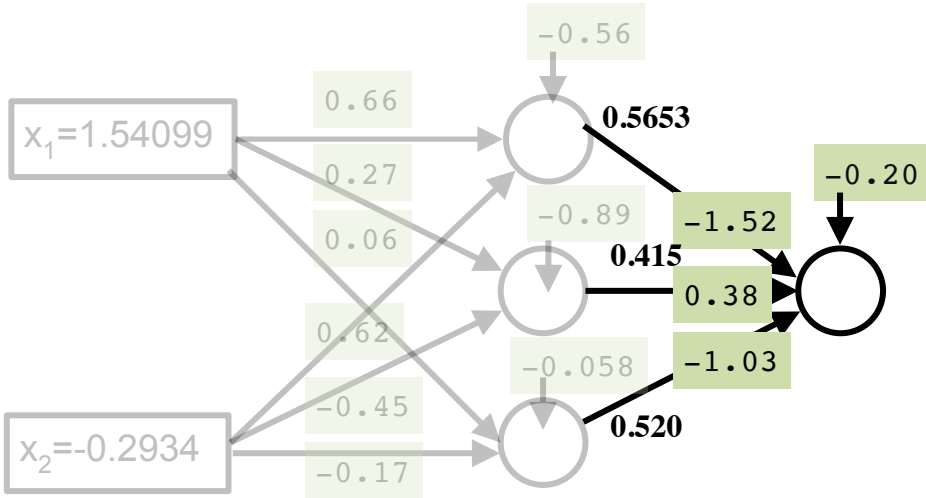
$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^T\mathbf{x}}} = \frac{1}{1 + exp^{-(-1.437156)}}$$

$$= 0.191$$

Forward Pass in our Multilayer Perceptron (MLP)

```
matrix_mult_hidden_and_W2 = torch.matmul(output_hidden_layer, dense_connections_W2) + bias_terms_output
print('output of output layer: \n', matrix_mult_hidden_and_W2)
final_output = sigmoid_activation_output(matrix_mult_hidden_and_W2)
print('output of hidden layer: \n', final_output.numpy())
```

```
output of output layer:
 tensor([-1.4383])
output of hidden layer:
 [0.1918079]
```



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2 \quad w_3] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = [1 \quad -1.52 \quad 0.38 \quad -1.03] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = 1*(-0.20) + (-1.52)*0.5653 + 0.38*0.415 + (-1.03)*(0.520)$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}} = \frac{1}{1 + exp^{-(-1.437156)}} = 0.191$$

Returning back to image classification

- Determine whether a given photo contains a ‘Dog’, ‘Cat’, ‘Horse’, or another specific object.

Multilayer Perceptron (MLP) for Image Classification

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers