COMPSCI 389
Introduction to Machine Learning

Days: Tu/Th. **Time:** 2:30 – 3:45 **Building:** Morrill 2 **Room:** 222

Topic 9.0: Neural Networks

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Linear Parametric Models (review)

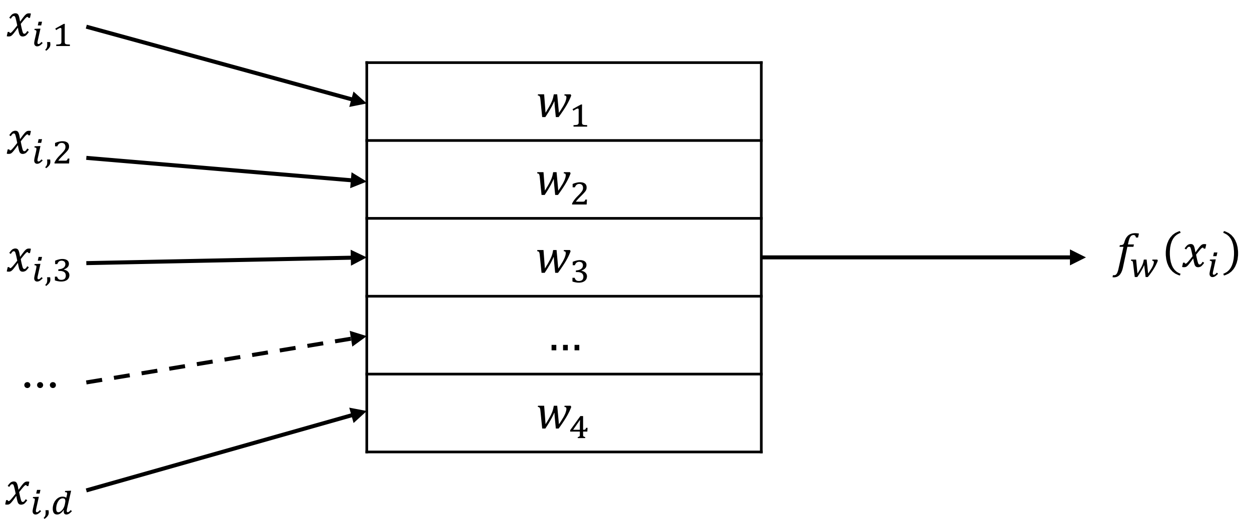
- Linear parametric models f_w are linear with respect to the weights, w, of the model.

$$f_w(x_i) = \sum_{j=1}^{m} w_j \phi_j(x_i)$$

Linear Model (Graphical Representation)

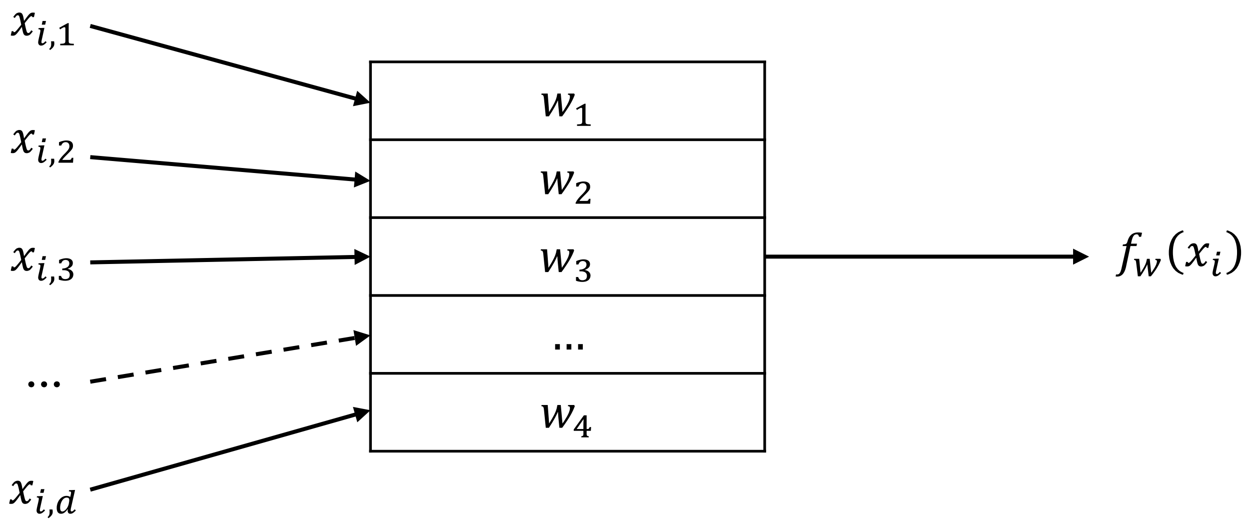
- Consider a linear parametric model **without a basis function** (feature generator, ϕ)

$$f_w(x_i) = \sum_{j=1}^{d} w_j x_{i,j}$$



Question: How can we make this model non-linear w.r.t. the model parameters (weights w)?

$$f_w(x_i) = \sum_{j=1}^{d} w_j x_{i,j}$$

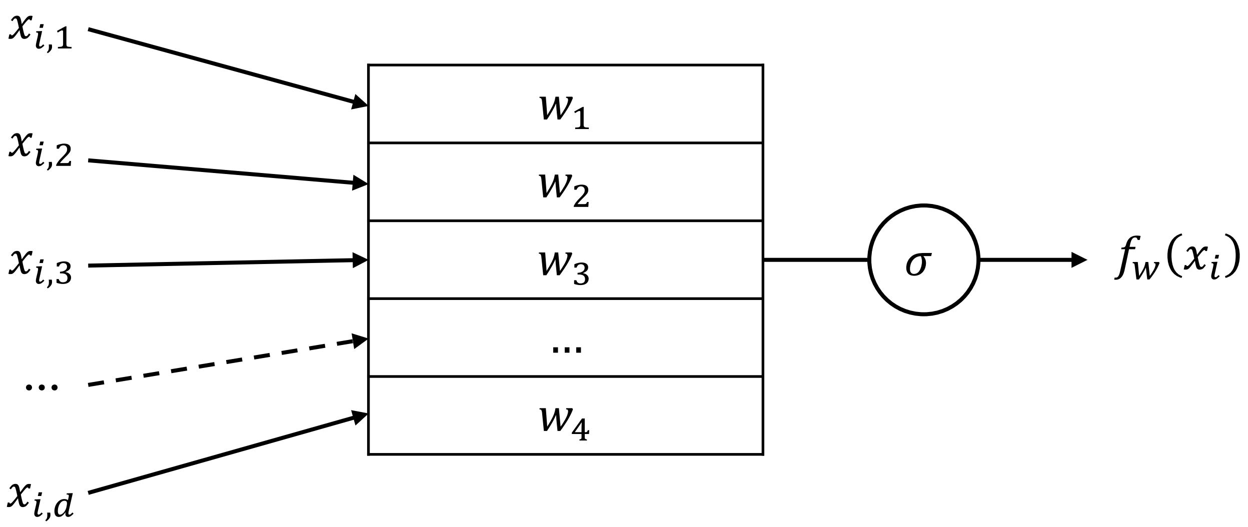


Answer: One way is to apply a non-linear function, σ, to the output.

Question: Would $\sigma(z) = 5z$ work?
Answer: No, this is a linear function. This would be equivalent to multiplying each weight by 5. It doesn't change the functions that can be represented.

Question: Would $\sigma(z) = z^2$ work?
Answer: Yes, this would result in a non-linear parametric model.

$$f_w(x_i) = \sigma\left(\sum_{j=1}^{d} w_j x_{i,j}\right) \qquad f_w(x_i) = \left(\sum_{j=1}^{d} w_j x_{i,j}\right)^2$$

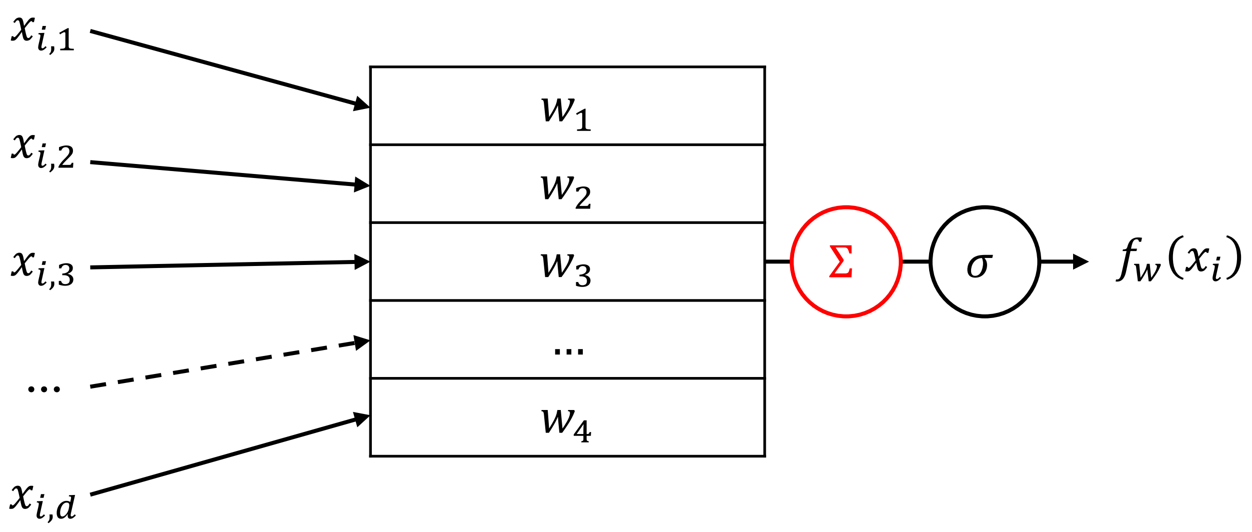


Note: The function σ is often called an **activation function, nonlinearity, threshold function**, or **squashing function**.

Note: This parametric model (with any nonlinear σ) is called a **perceptron**.

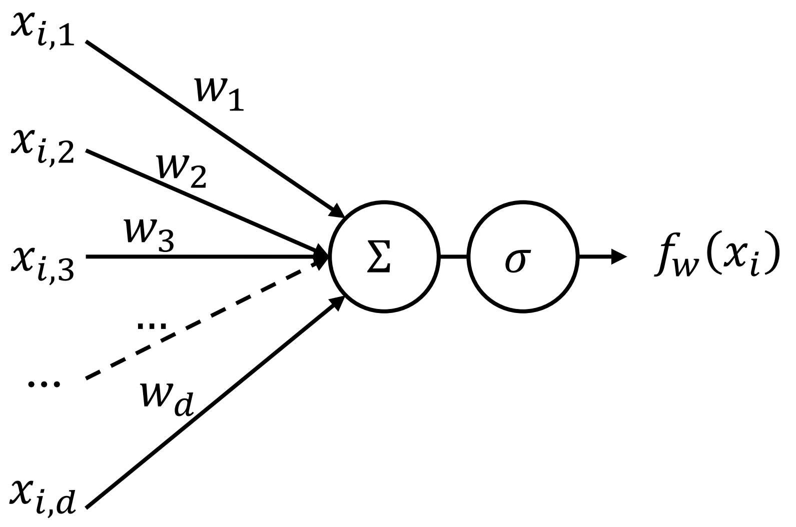
Alternative Perceptron Graphics

$$f_w(x_i) = \sigma\left(\sum_{j=1}^{d} w_j x_{i,j}\right)$$



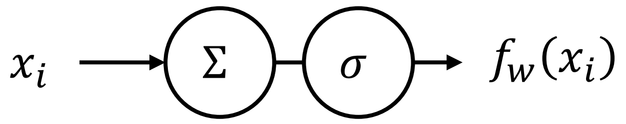
Alternative Perceptron Graphics

$$f_w(x_i) = \sigma\left(\sum_{j=1}^{d} w_j x_{i,j}\right)$$



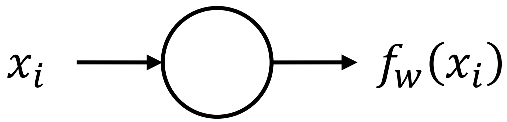
Alternative Perceptron Graphics

$$f_w(x_i) = \sigma\left(\sum_{j=1}^{d} w_j x_{i,j}\right)$$



Alternative Perceptron Graphics

$$f_w(x_i) = \sigma\left(\sum_{j=1}^{d} w_j x_{i,j}\right)$$

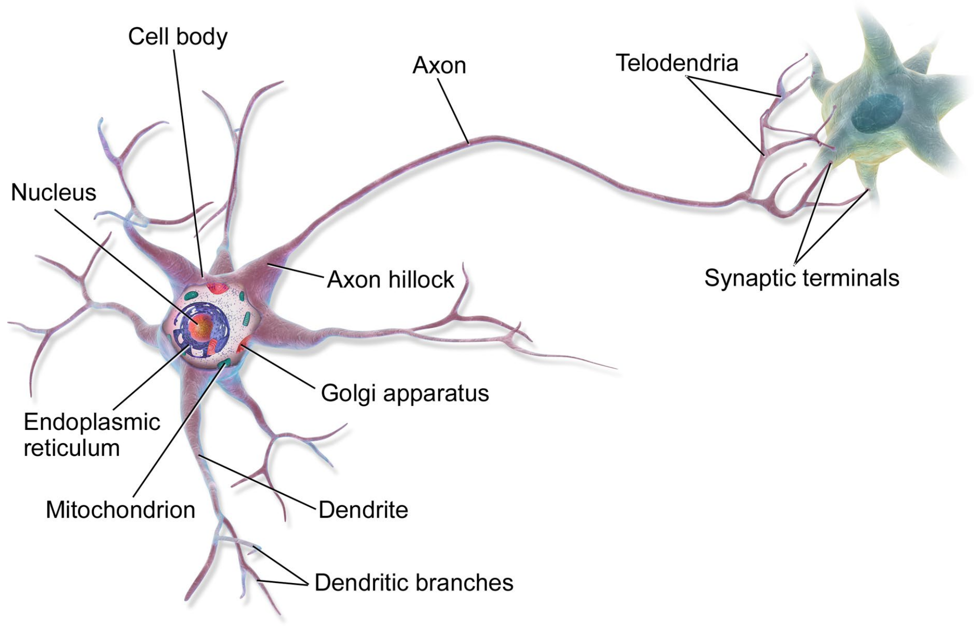


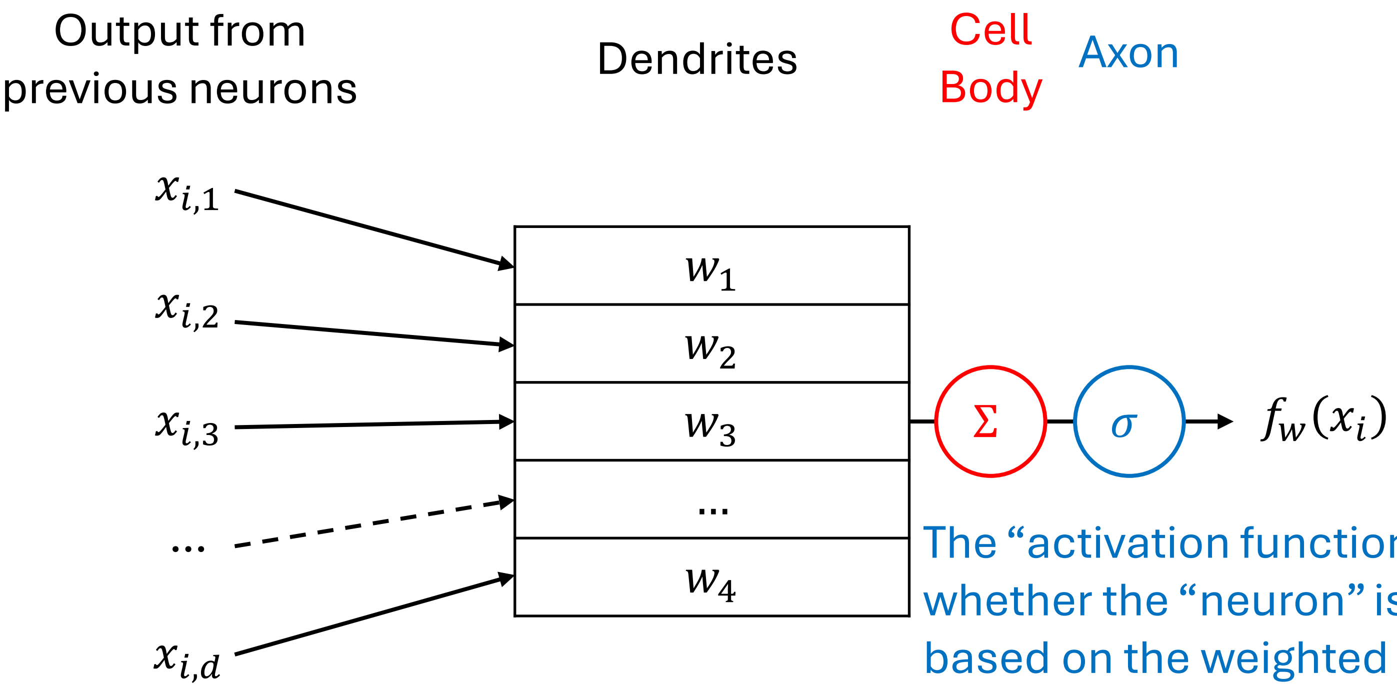
Note: This is most common when working with many perceptrons, connected together in some way.

Perceptron

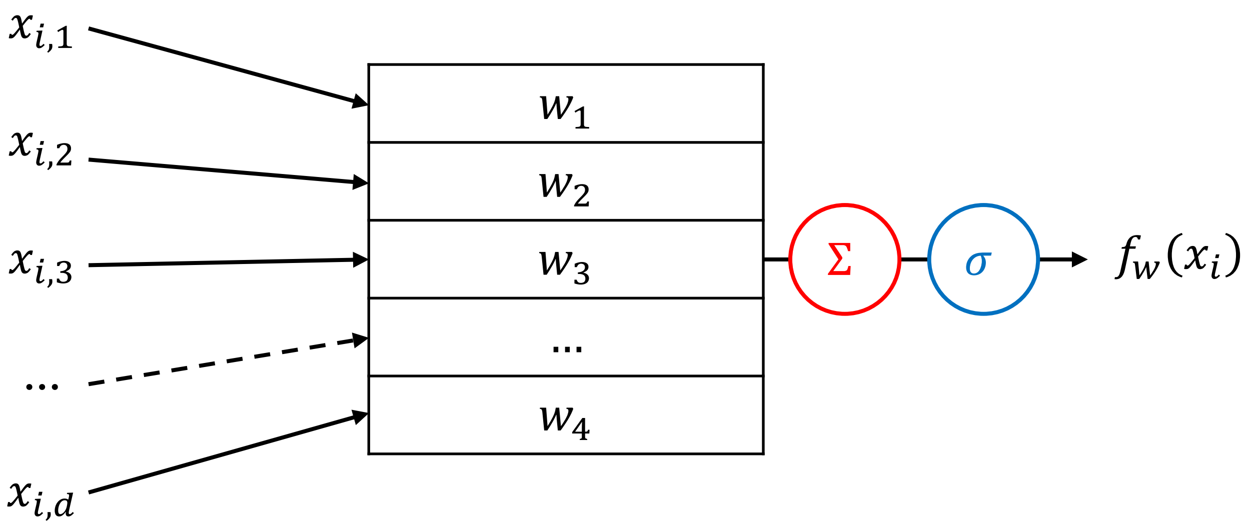
Perceptrons can be viewed as **extremely crude** simulations of neurons.

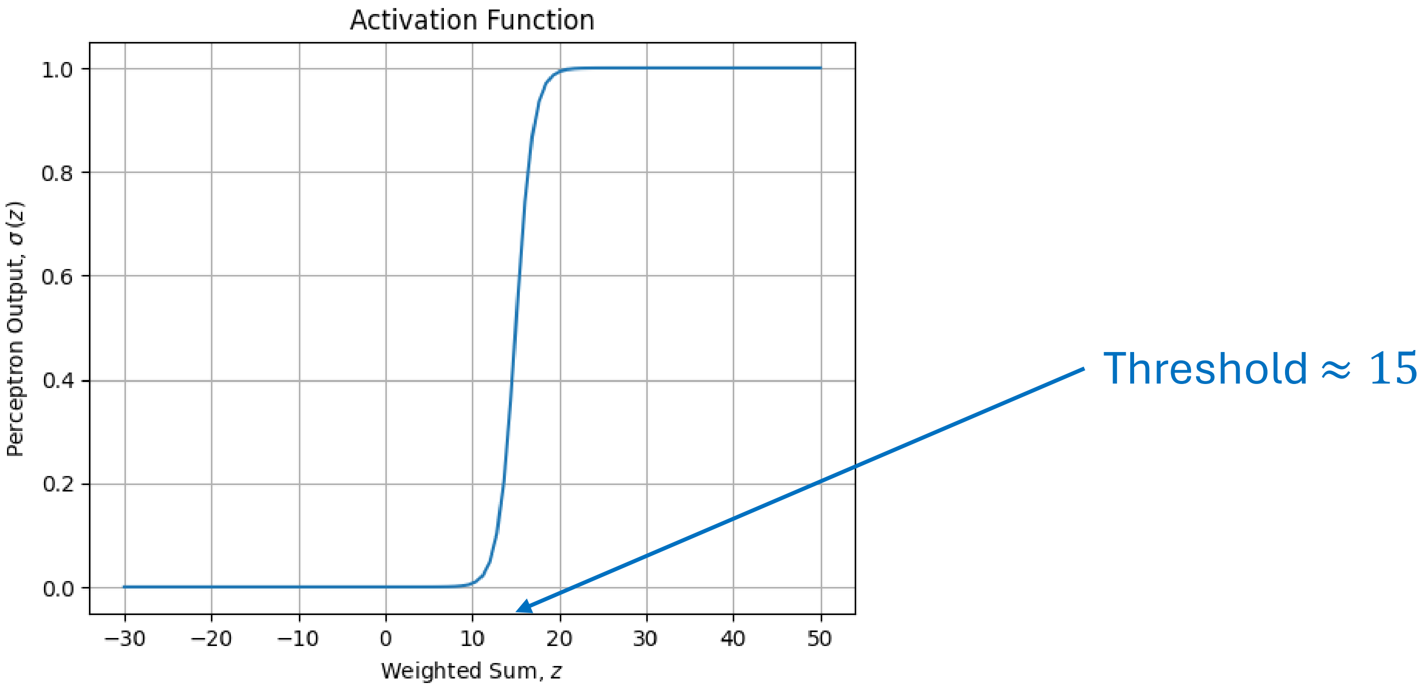
- Roughly speaking (ignoring important aspects of biology and neuroscience), when enough of the inputs to a neuron are activated, the neuron becomes sufficiently stimulated and "fires" (it becomes activated).
- We can select σ to be similar to a threshold function.
  - If the weighted sum is below some threshold for the neuron to be activated, σ outputs 0 (not firing).
  - If the weighted sum is above the threshold, σ outputs 1 (firing).





The "activation function" decides whether the "neuron" is firing based on the weighted sum.





Note: This model typically outputs 0 or 1, which may not be what we want for our parametric model. We will revisit this later.

Note: σ *squashes* the output to the range [0,1], hence the name *squashing function*.

Perceptron vs Neuron

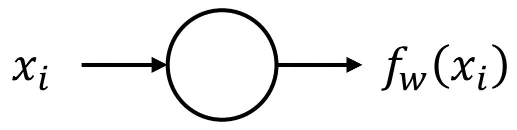
- Perceptrons are **extremely crude** models of real neurons.
- Real neurons do not switch between firing and not firing, but instead change the rate at which they fire.
  - More realistic parametric models of neurons are called “spiking neuron models”
- Real neurons don’t just compute a weighted sum of the inputs.
  - They consider the timing of different inputs arriving.
  - Complex calculations can result from dendritic morphology.
- Neurons experience fatigue.
  - Roughly speaking, when a neuron fires at a high rate for too long, chemical changes force it to fire less frequently.
- And much, much more…

Training Non-Linear Parametric Models

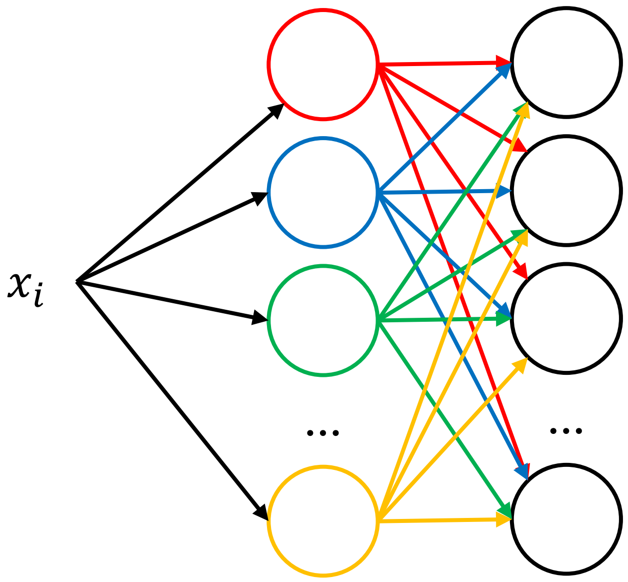
- We train non-linear parametric models using **gradient descent**!
- Later we will discuss how the necessary derivatives can be computed.

Neural Networks: Parametric Models Comprised of Many Perceptrons

- Recall the graphical representation:

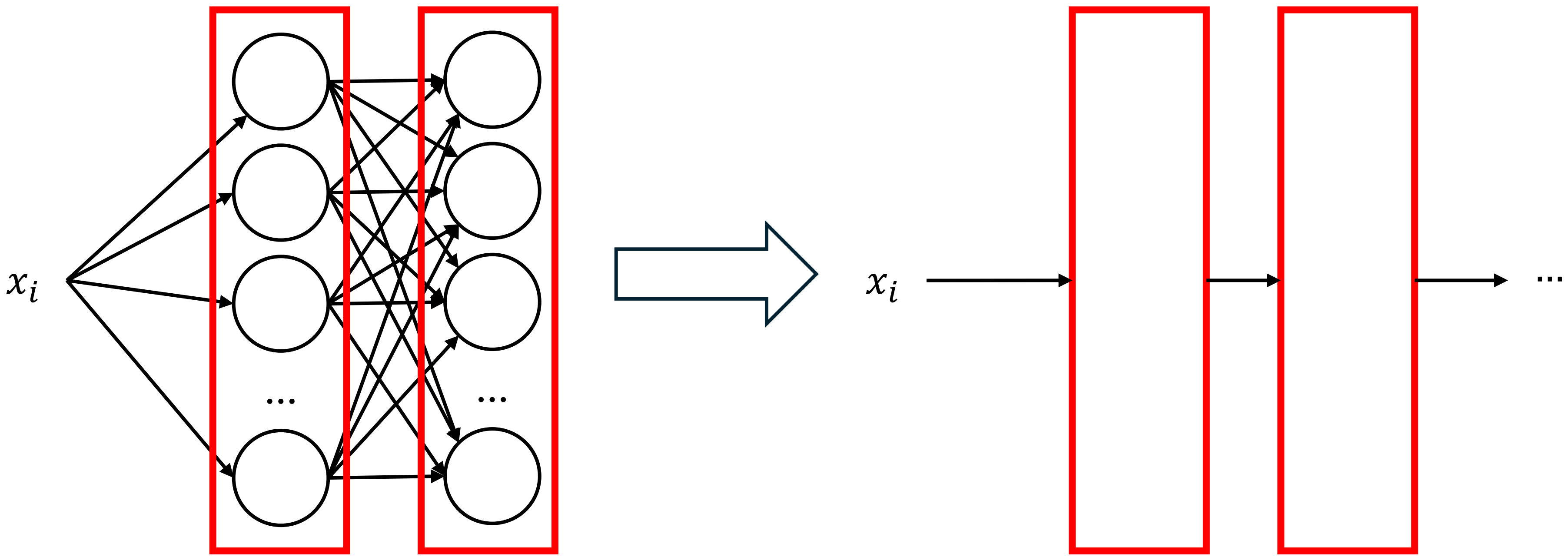


- **Idea**: Connect many perceptrons together.



This is tedious and
too many arrows!

Neural Network Graphical Depiction

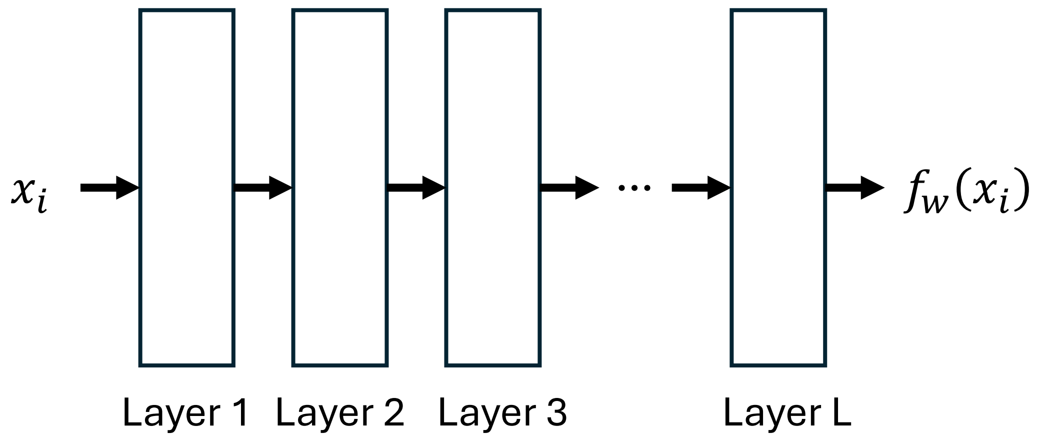


Idea: Use boxes to represent **layers** (columns) of perceptrons.

Here arrows between boxes denote **fully connected layers**.

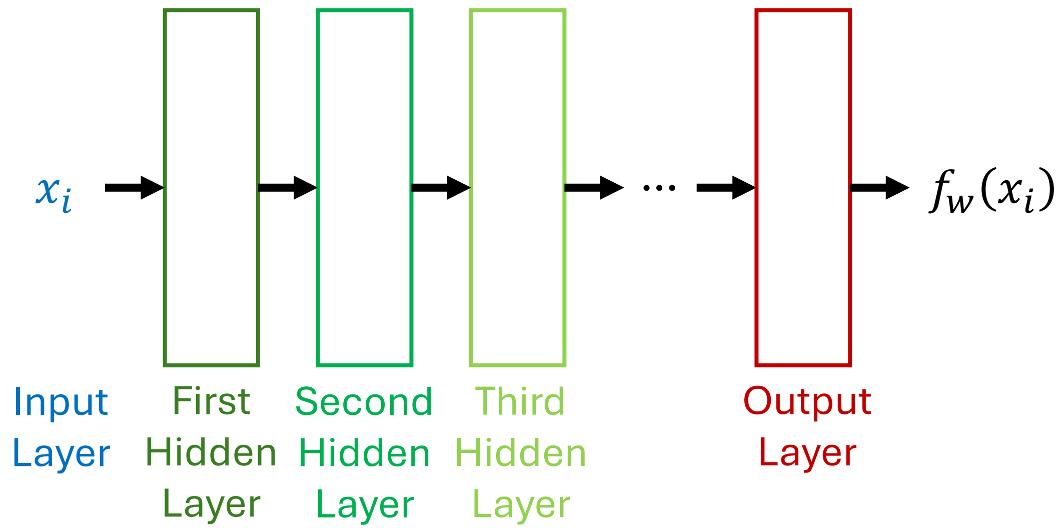
- Each perceptron in the right-layer takes the output of each perceptron in the left-layer as input.

Neural Network (Graphical Depiction)



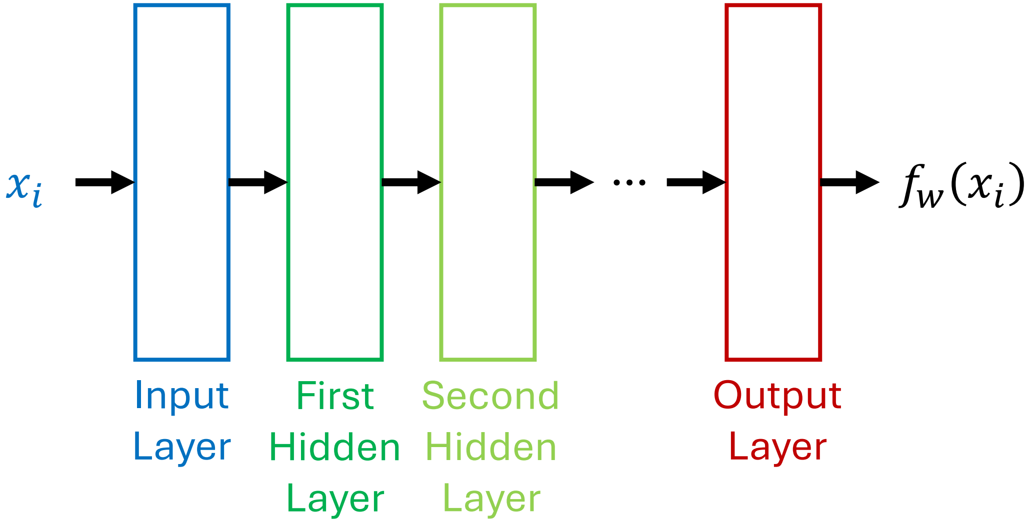
- In the context of neural networks, perceptrons are often called **units**.
- Each layer can have different numbers of units.
  - The number of units in a layer is often called the "size" of the layer.

Neural Network (Graphical Depiction)



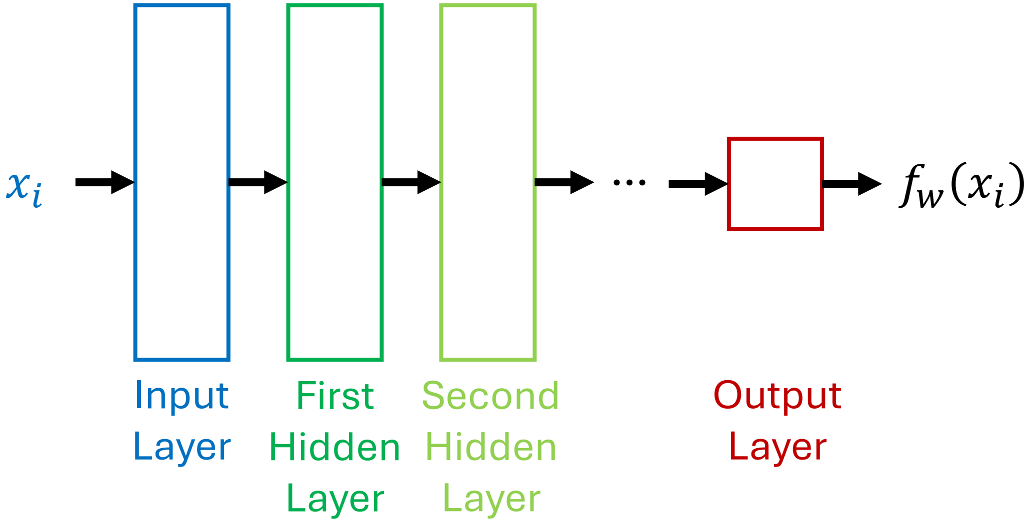
- The input, x_i is called the **input layer**.
- The last layer is called the **output layer**.
- All layers between the input and output layers are called **hidden layers**.

Neural Network (Graphical Depiction)



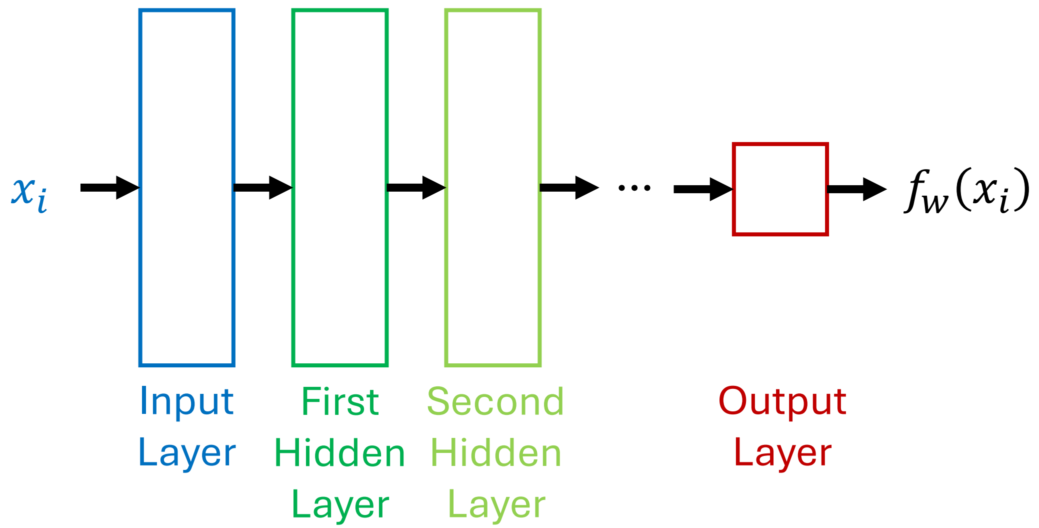
- Sometimes the input layer is represented by its own rectangle.
- This layer simply outputs x_i.

Neural Network (Graphical Depiction)



- The number of units in the output layer should equal the number of outputs of $f_w(x_i)$
  - For the GPA-prediction task, $x_i \in \mathbb{R}^9$ and $y_i \in \mathbb{R}$.
  - So, the output layer should have one unit.

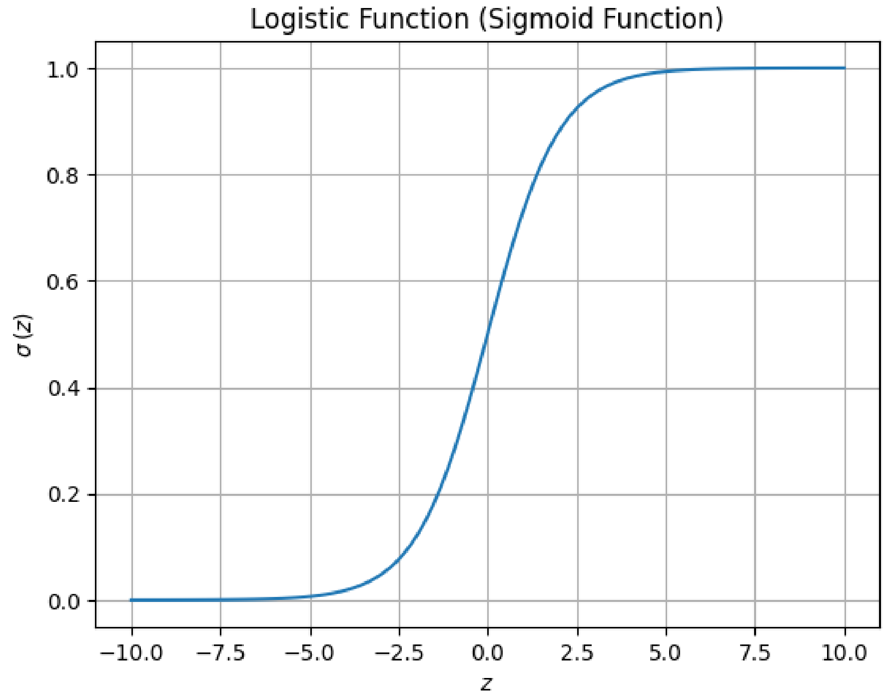
Neural Network (Graphical Depiction)



- If the output of the parametric model should not be "squashed" to [0,1], the squashing function (activation function) can be omitted from the output layer.

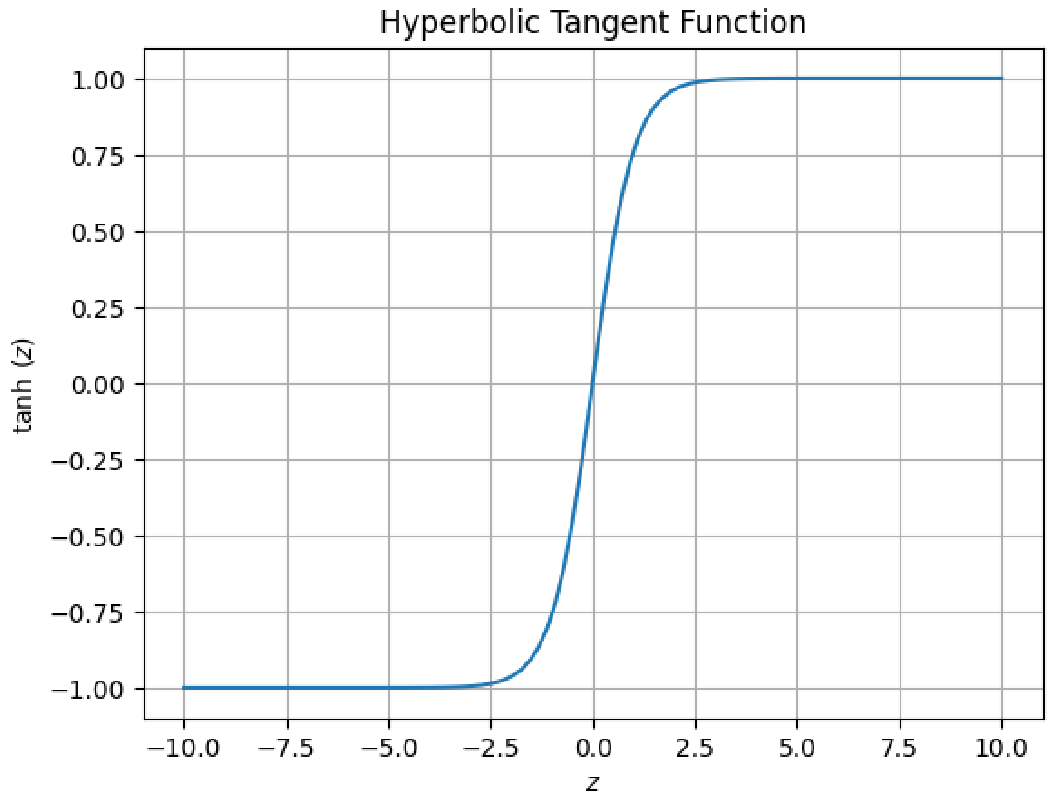
Activation Function: Sigmoid

- **Sigmoid functions** are a class of S-shaped functions.
- The most common one is called the **logistic function.**
  - It is so common that it is often called “the” sigmoid function.
- $\sigma(z) = \frac{1}{1+e^{-z}}$



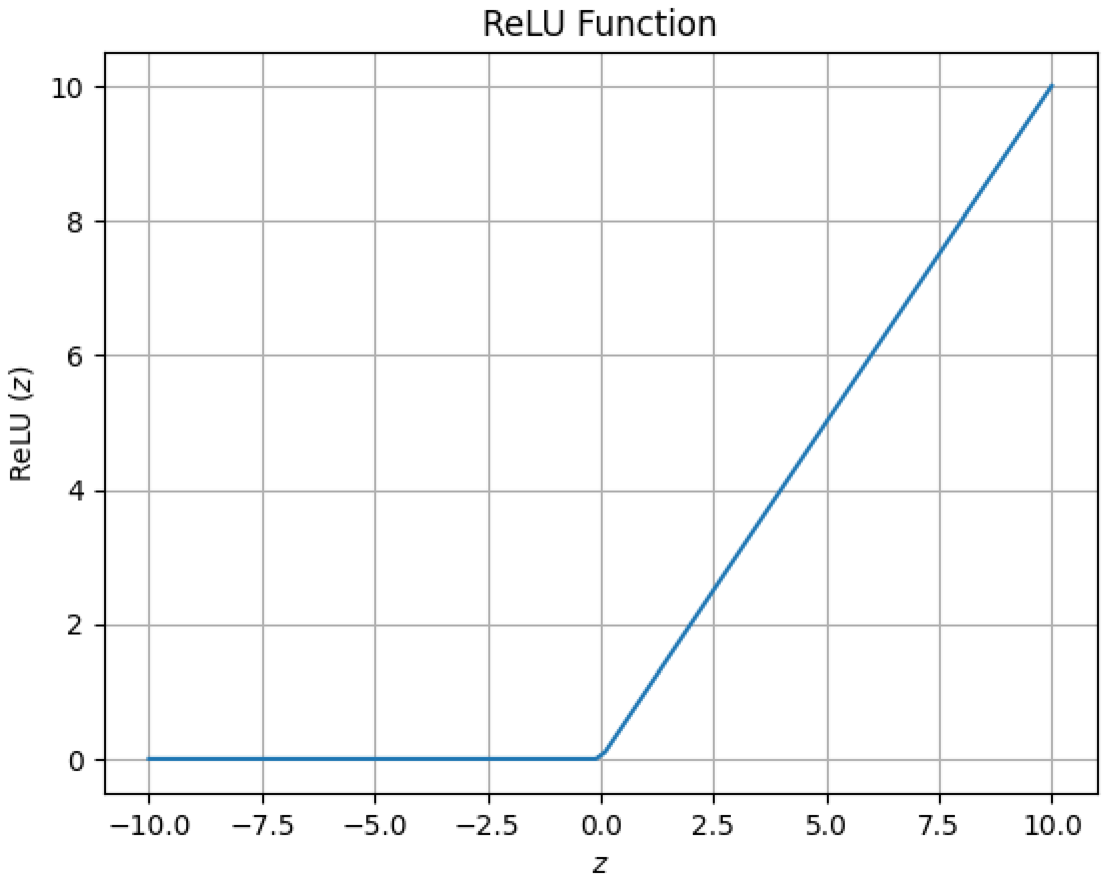
Activation Function: Hyperbolic Tangent Function (tanh)

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



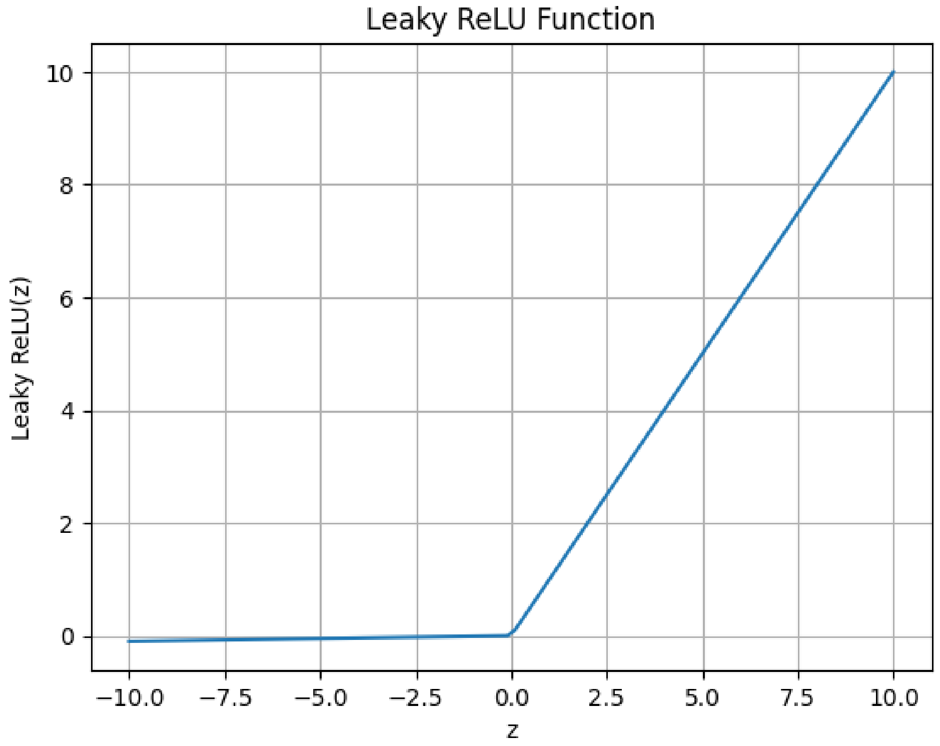
Activation Function: Rectified Linear Unit (ReLU)

- $\mathrm{ReLU}(z) = \max(0, z)$



Activation Function: Leaky ReLU

- Leaky ReLU$(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$
  - Here α is a small constant, typically 0.01.

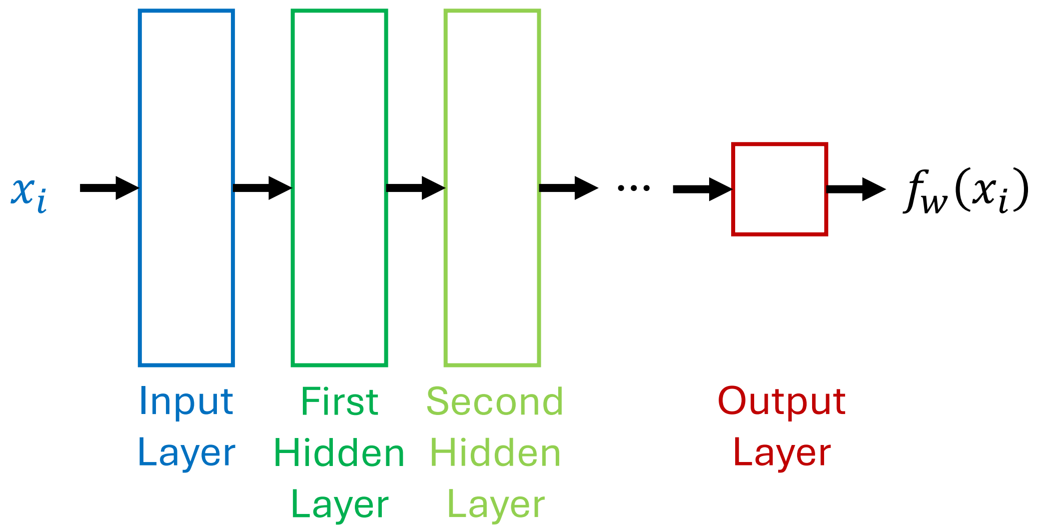


Terminology

- You will see both **neural network** and **artificial neural network** (ANN) used to describe these parametric models.
  - ANN emphasizes that these parametric models are very different from biological neural networks.
  - We will use both phrases, but will use the abbreviation ANN to differentiate from nearest neighbor (NN).

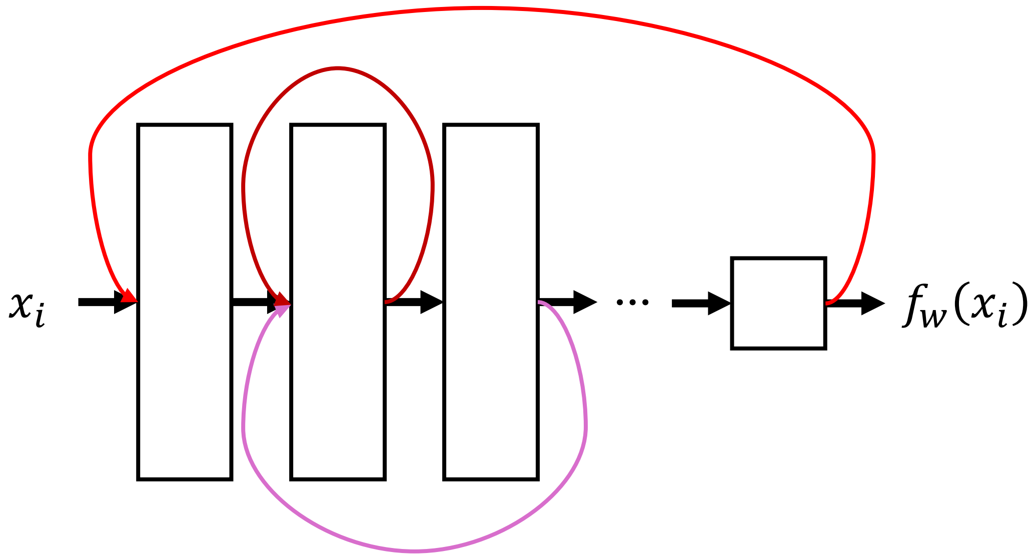
Fully-Connected Feed-Forward Networks

- A fully-connected feed-forward ANN is one where each unit in the i^{th} layer:
  - Takes the output of each unit in the $(i-1)^{\text{th}}$ layer as input.
  - Provides its output to each unit in the $(i+1)^{\text{th}}$ layer.



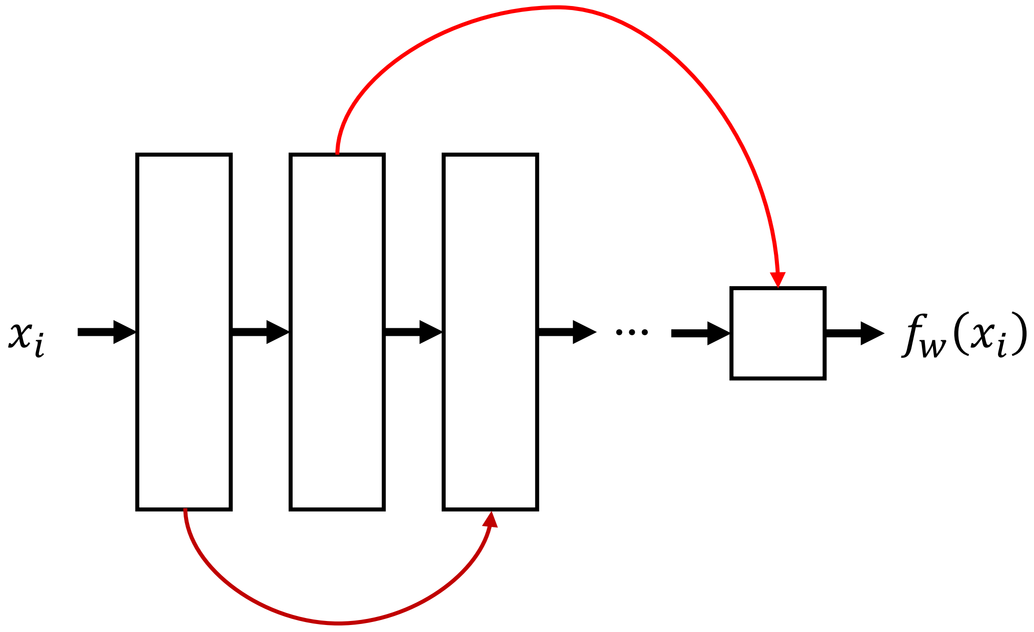
Recurrent Neural Network (RNN)

- Recurrent neural networks can have backwards connections between layers.
- These networks are typically run several times on the same input, and recurrent (backwards) edges provide values from the previous runs.
  - Recurrent connections provide a form of "memory"



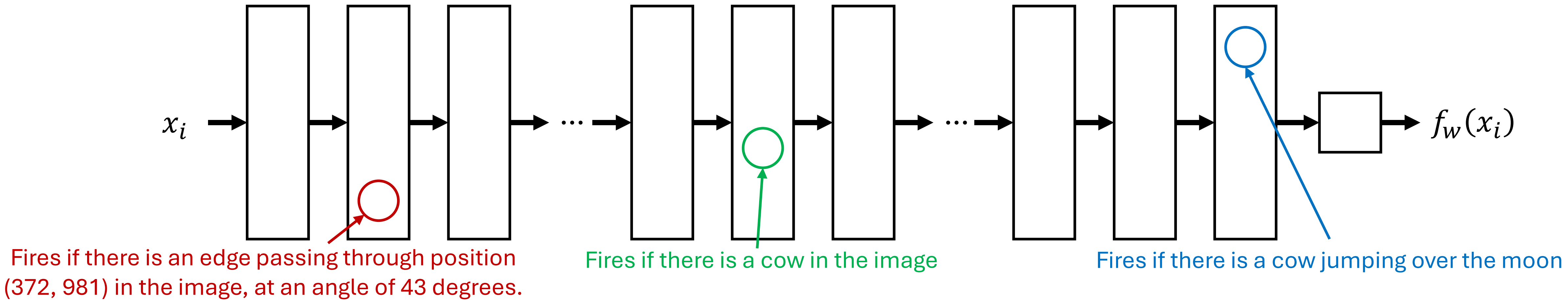
Skip Connections

- Skip connections are connections that skip over one or more layers.



What do different layers learn?

- Consider parametric models that take images as input.
- The layers closer to the input tend to learn low-level visual features.
- Later layers use these low-level features to learn about higher-level features and concepts.

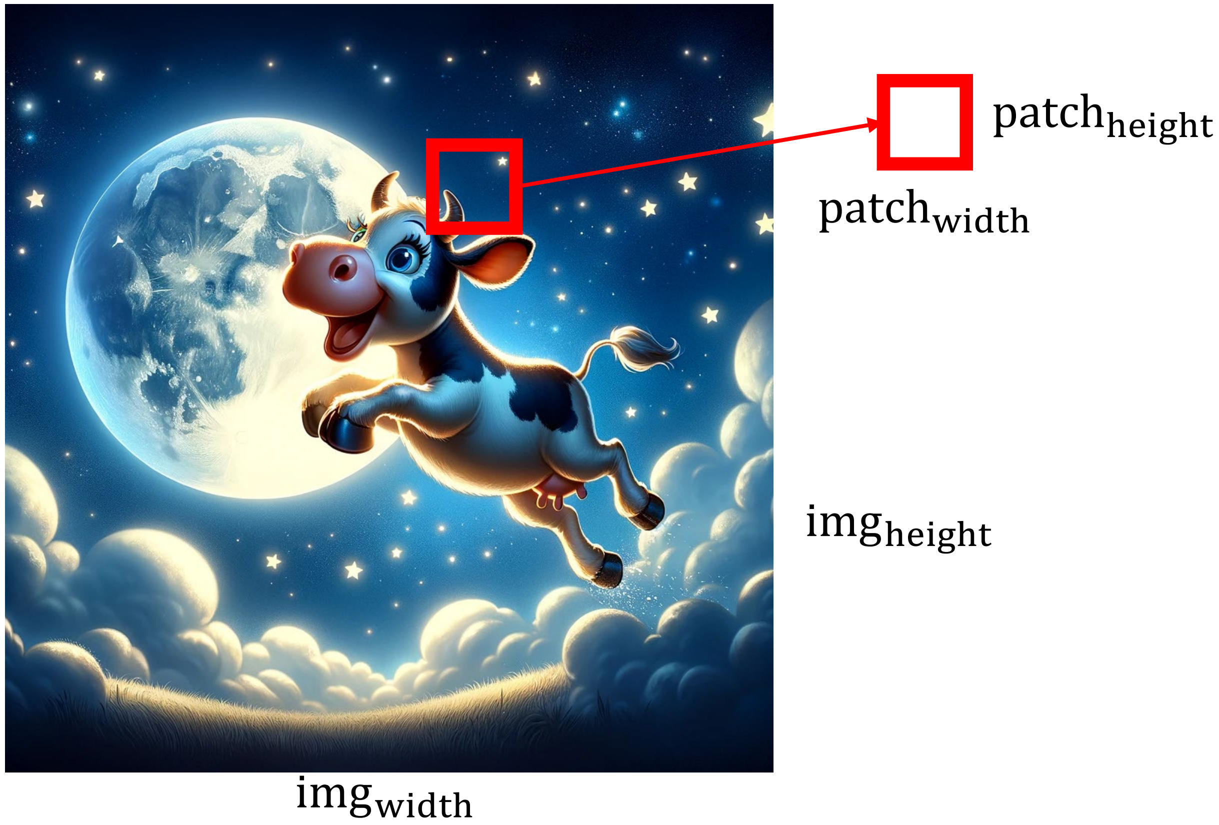


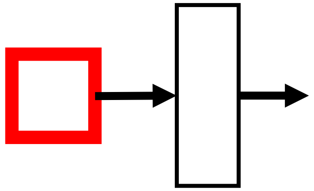
Learning Low-Level Features

- An ANN might use early layers to detect low-level features of an image
  - One unit early in the network might “fire” when there is an edge at position (x,y) in the image, and the edge is vertical.
  - Another unit might fire when there is an edge at position (x,y) at an angle of 80 degrees (nearly vertical).
  - There may be different units for all of these features at each (x,y) coordinate in the image!
- Learning to separately detect the same feature at each location in the image is wasteful.
- **Idea**: Create a parametric model (layer for ANNs) that learns to find and represent features *anywhere* in the image.

Convolutional Layer

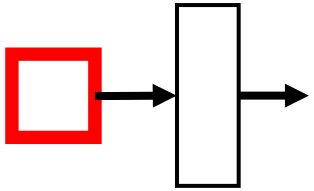
- If an image is of size $\mathrm{img}_{\mathrm{width}} \times \mathrm{img}_{\mathrm{height}}$, create a parametric model, called a **filter,** that takes as input a small subregion of the image, called a **patch**.
- This filter (small parametric model) is run on each patch in the image.
    - The patches can overlap.
    - Each patch is a fixed number of pixels over from the previous patch. This number is called the **stride**.



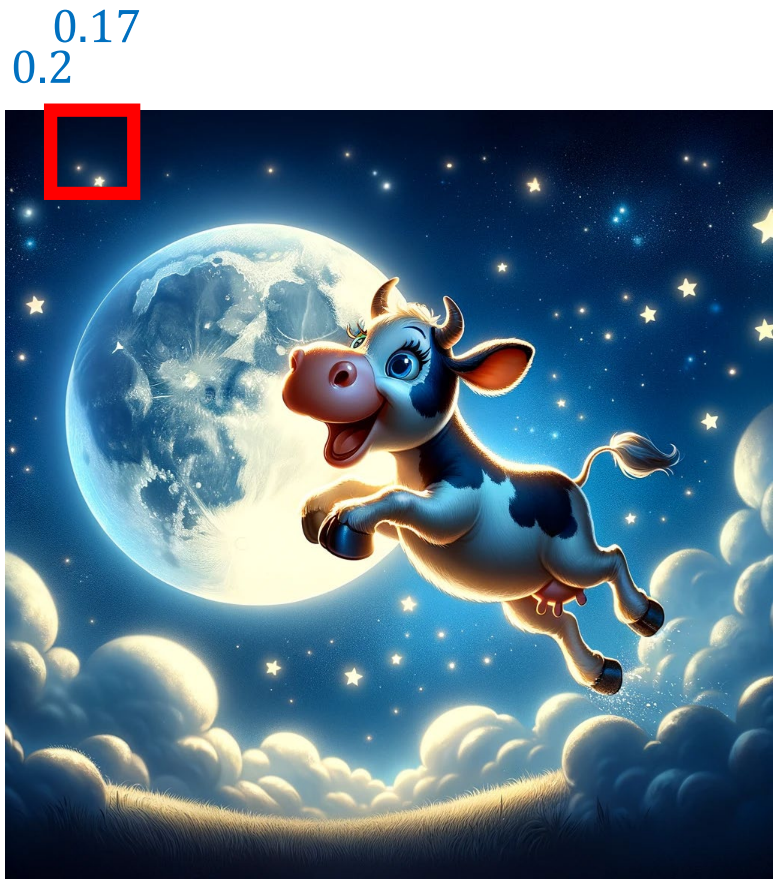


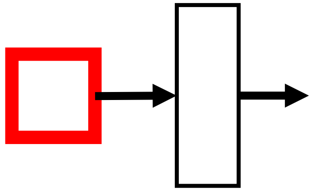
One number, the “feature” value for this patch.

0.2

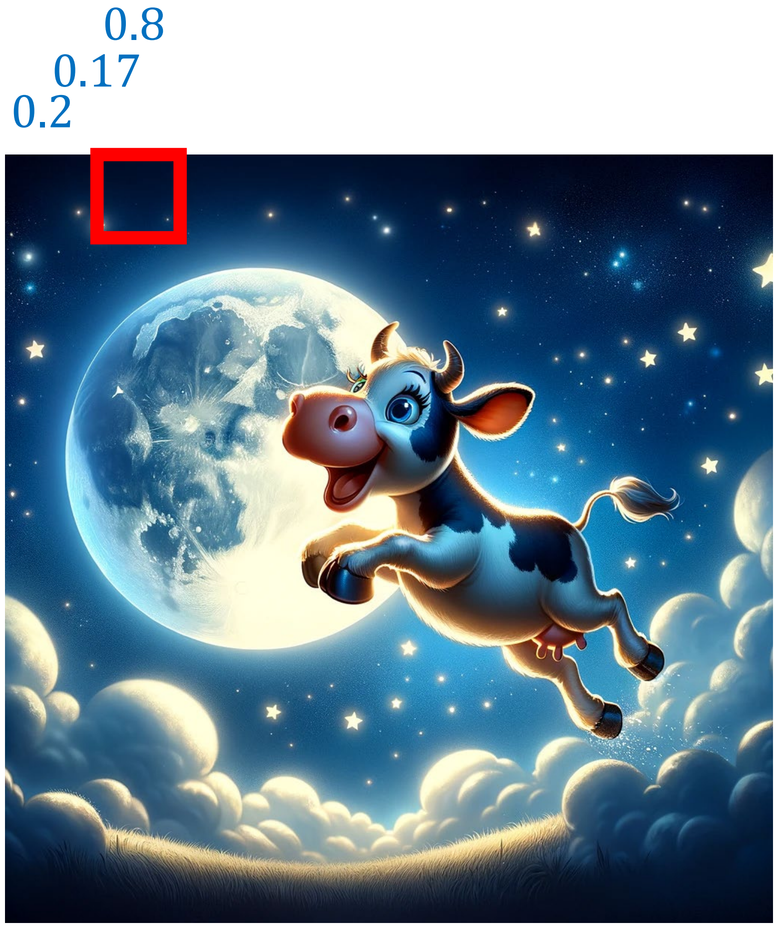


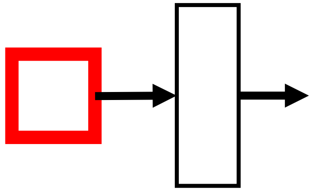
One number, the “feature” value for this patch.





One number, the “feature” value for this patch.

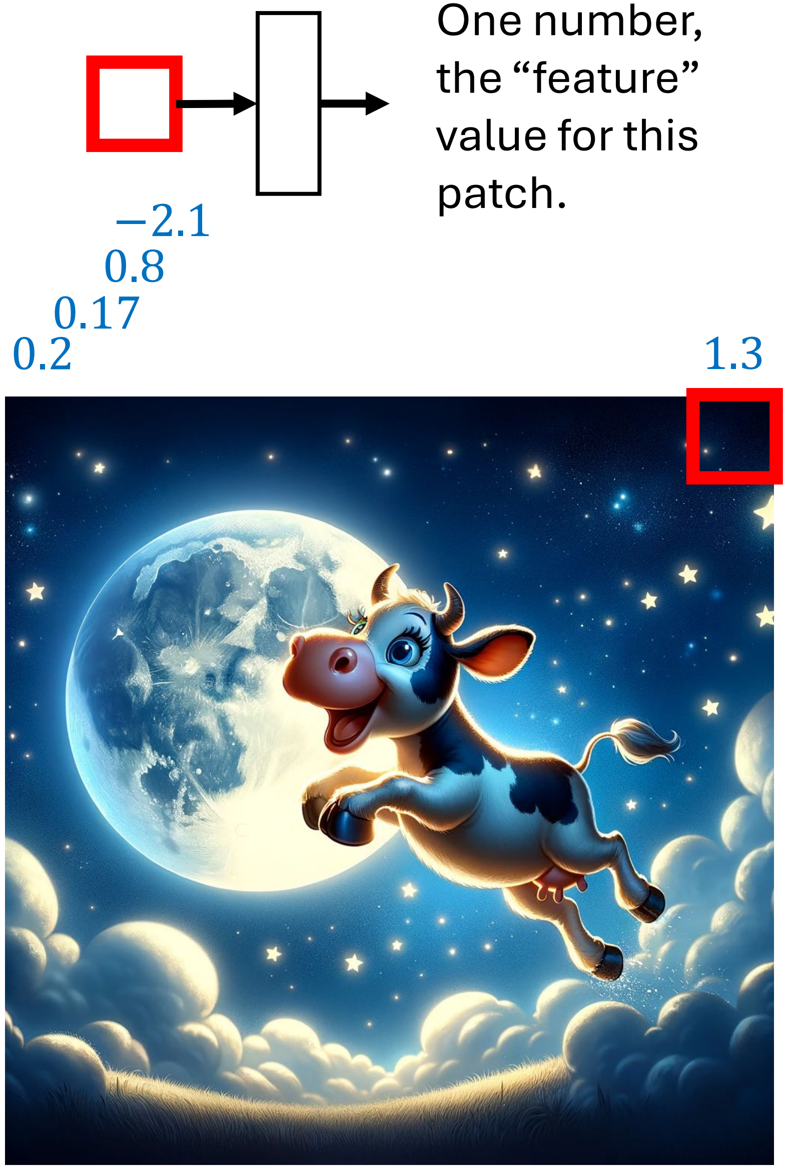




One number, the “feature” value for this patch.

-2.1
0.8
0.17
0.2

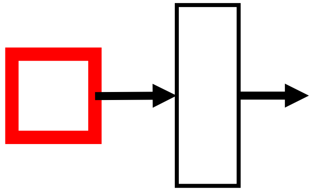
The patch is shifted over by **stride** number of pixels each time.



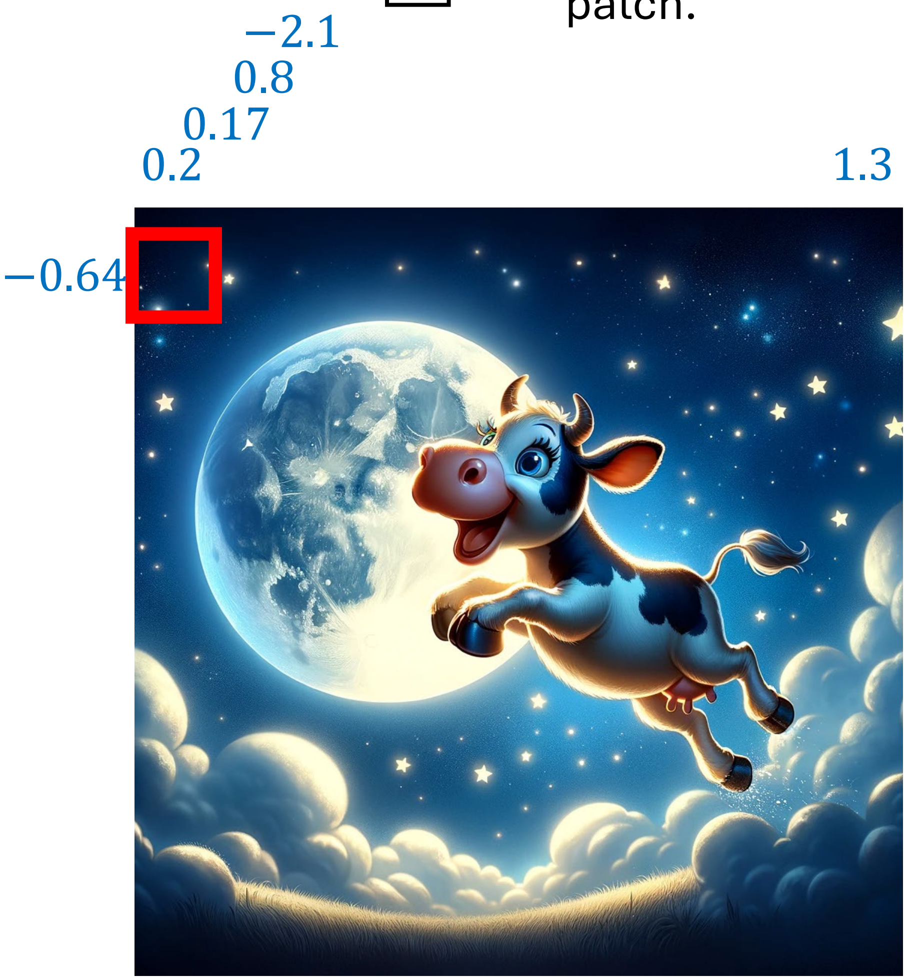
One number, the “feature” value for this patch.

The patch is shifted over by **stride** number of pixels each time.

When the patch reaches the end, it shifts down by **stride** pixels and starts over.

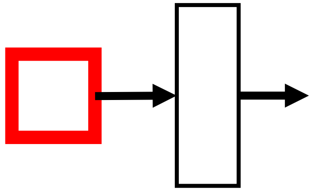


One number, the “feature” value for this patch.



The patch is shifted over by **stride** number of pixels each time.

When the patch reaches the end, it shifts down by **stride** pixels and starts over.



One number, the “feature” value for this patch.

-2.1

0.8

0.17

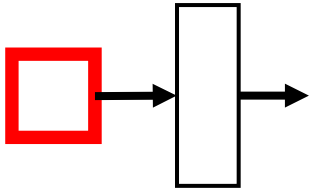
0.2 1.3

-0.64

The patch is shifted over by **stride** number of pixels each time.

When the patch reaches the end, it shifts down by **stride** pixels and starts over.

At the end, the **convolutional layer** outputs all the computed values:
$(0.2, 0.17, 0.8, -2.1, \ldots, 1.3, -0.64, \ldots)$



One number, the “feature” value for this patch.

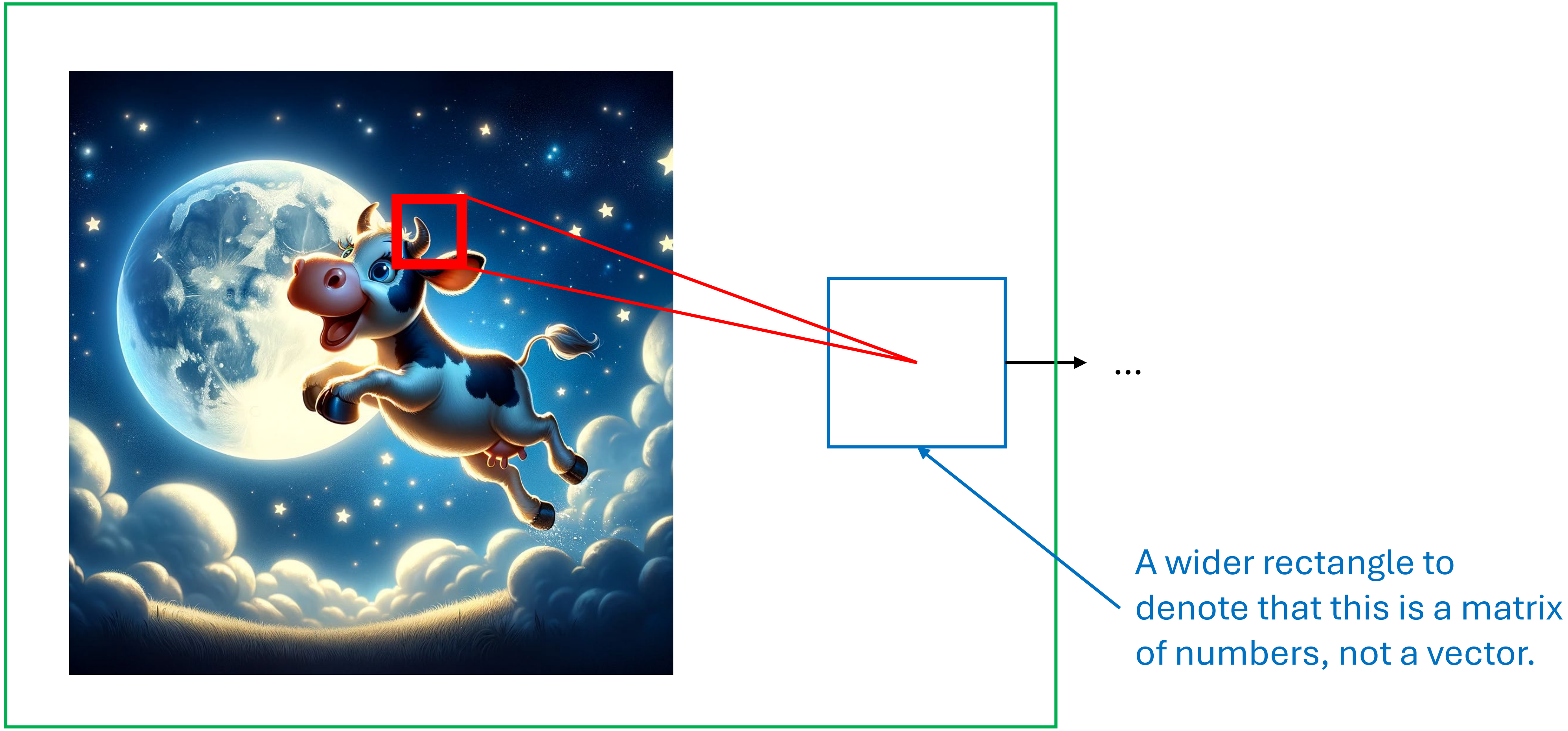
The patch is shifted over by **stride** number of pixels each time.

When the patch reaches the end, it shifts down by **stride** pixels and starts over.

At the end, the **convolutional layer** outputs all the computed values:
$(0.2, 0.17, 0.8, -2.1, \dots, 1.3, -0.64, \dots)$

These values are usually represented as a matrix to track the position of the patch they were computed from.

Convolutional Layer (Graphical Depiction)



This represents a convolutional layer (blue) applied to an image.

Convolutional Layer (summary)

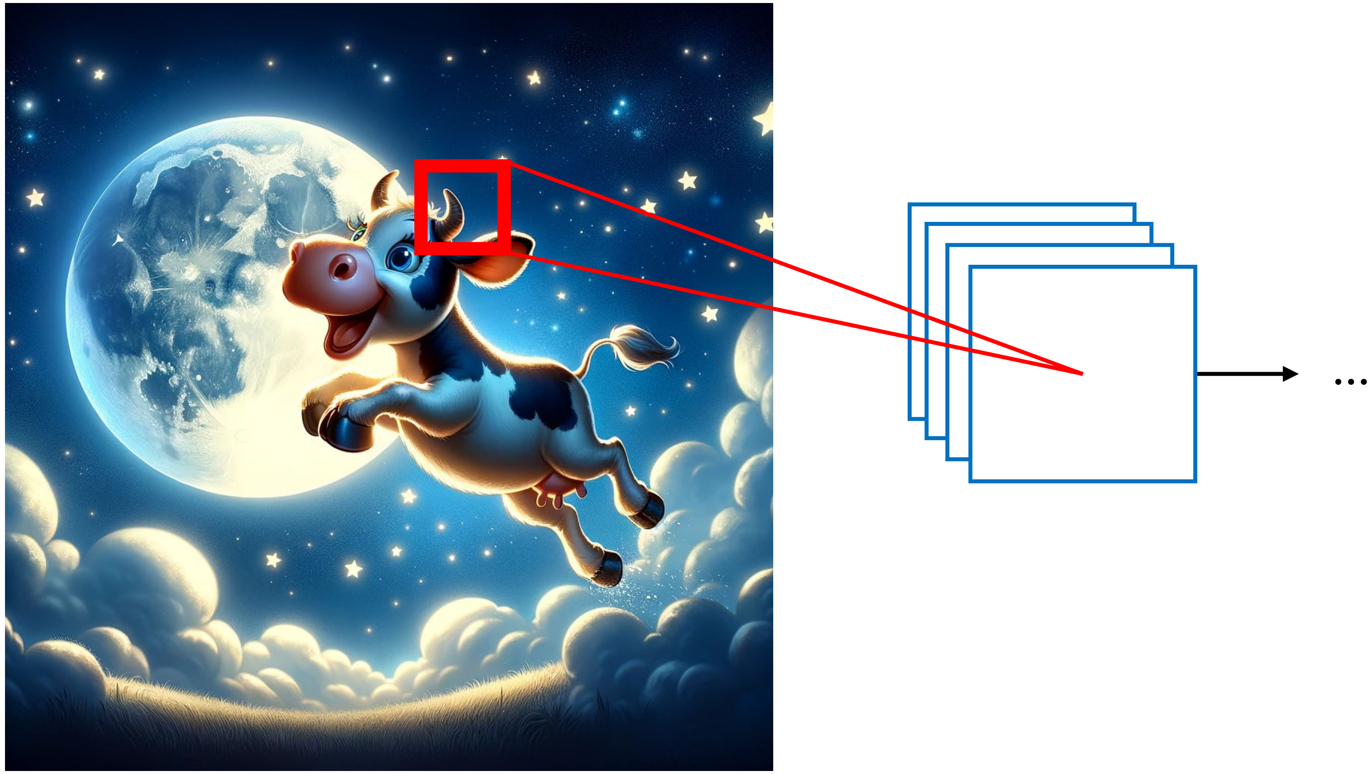
- A convolutional layer can be viewed as a small parametric model (within the main parametric model) that has a relatively small number of parameters.
    - This model is called a **filter**.
- The filter is applied to patches of an image.
- The outputs of the filter, for all patches, is viewed as the output of the convolutional layer.
    - These outputs are represented as a matrix.
    - The position in the matrix represents the position of the patch in the image.
- A single filter can learn features like “do two edges meet to form a corner in this patch?” or “is there a line at a specific angle in this patch?”

Convolutional Layer (Multiple Filters)

- We typically want to learn more than one feature for each patch.
  - For example, line detectors for lines at different angles.
- A convolutional layer, as described so far, learns only one feature.
- Convolutional layers can learn k features by applying k different filters (small parametric models) to each patch.
  - Each filter produces one number for each patch.
  - The outputs for each filter are stored as separate matrices, one per filter.

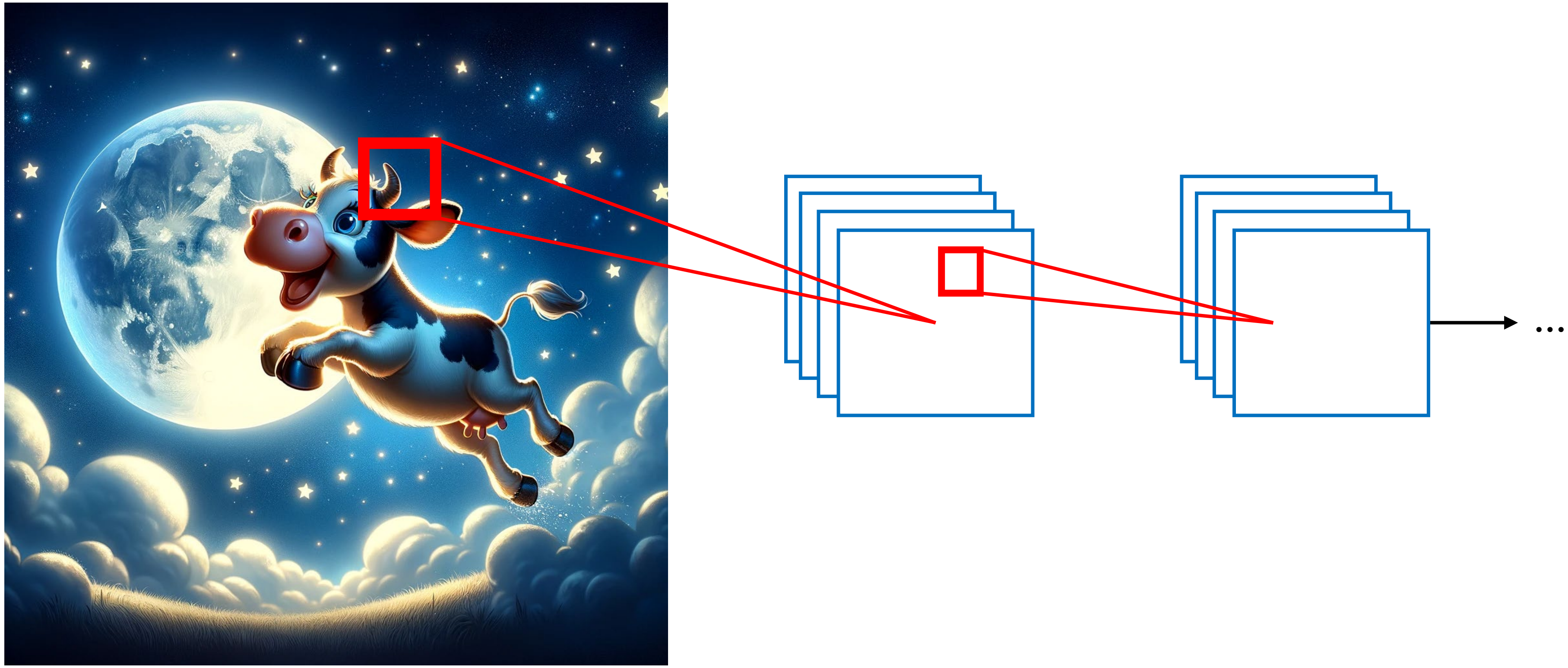
Convolutional Layer

- A convolutional layer with multiple filters is represented using many stacked boxes:



Convolutional Layer

- Convolutional layers can be applied in a sequence!



Max Pooling Layers

- When using convolutional layers with many filters, you can end up with more outputs from the convolutional layer than there were pixels in the original image!
- To make the number of values more manageable, a **max pooling** layer can be used to downsample (reduce) the number of features.
- A max pooling layer acts like a convolutional layer, but without any parameters.
  - For each patch, it returns the maximum value within the patch.
  - Other pooling layers (e.g., average pooling layers) compute other fixed functions of a patch (e.g., the average value in the patch)
  - A max pooling layer typically has a relatively wide stride and/or patch.
    - For example, a 2x2 patch with no overlap between patches quarters the number of values.