Multilayer Perceptron Architecture

Understanding Neural Network Layers

Sina Moghimi, Autumn 2024

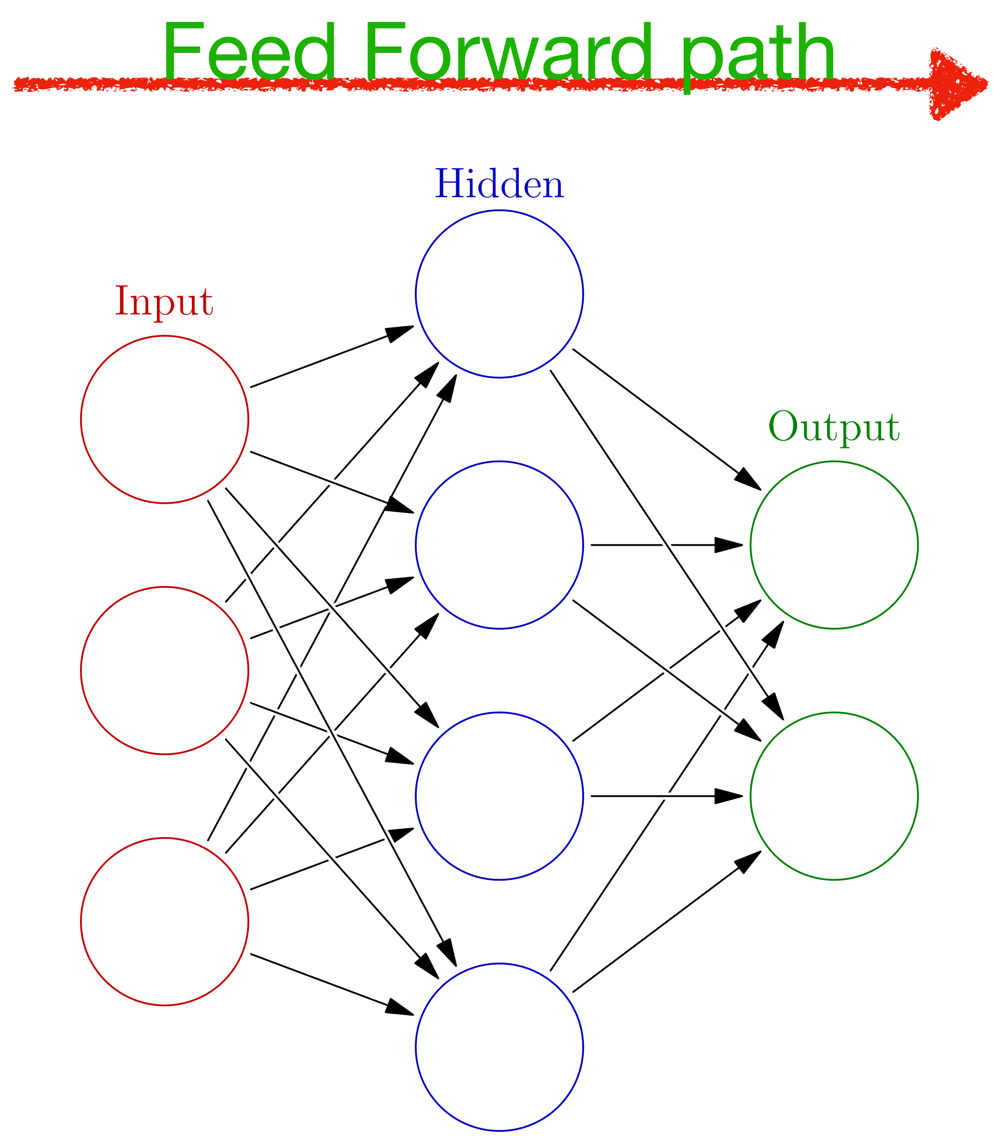
МФТИ

Outline

- Feedforward Network Basics
- Input Layer Fundamentals
- Hidden Layers Exploration
- Output Layer Design
- Network Complexity Analysis
- Mathematical Representation
- Architectural Considerations
- Practical Implementation

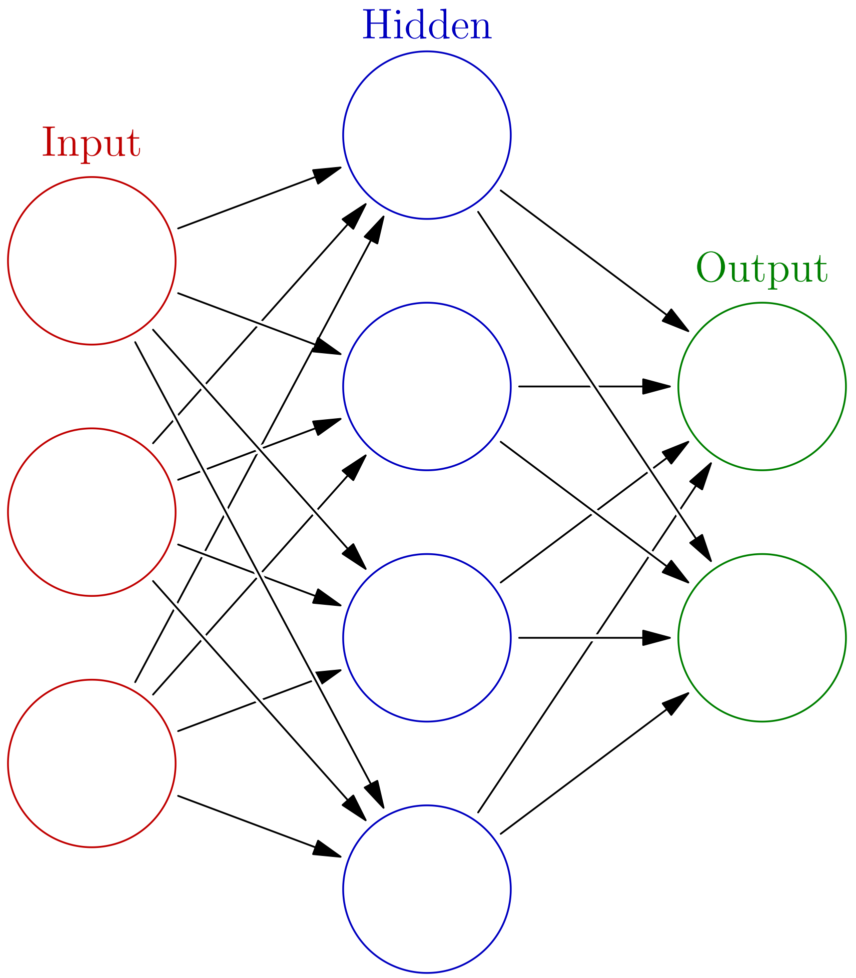
Feedforward Network Basics

- Information flows from input to the output (unidirectional)
- No Loops/Cycles
- No Memory of Past Inputs(unlike RNNs)



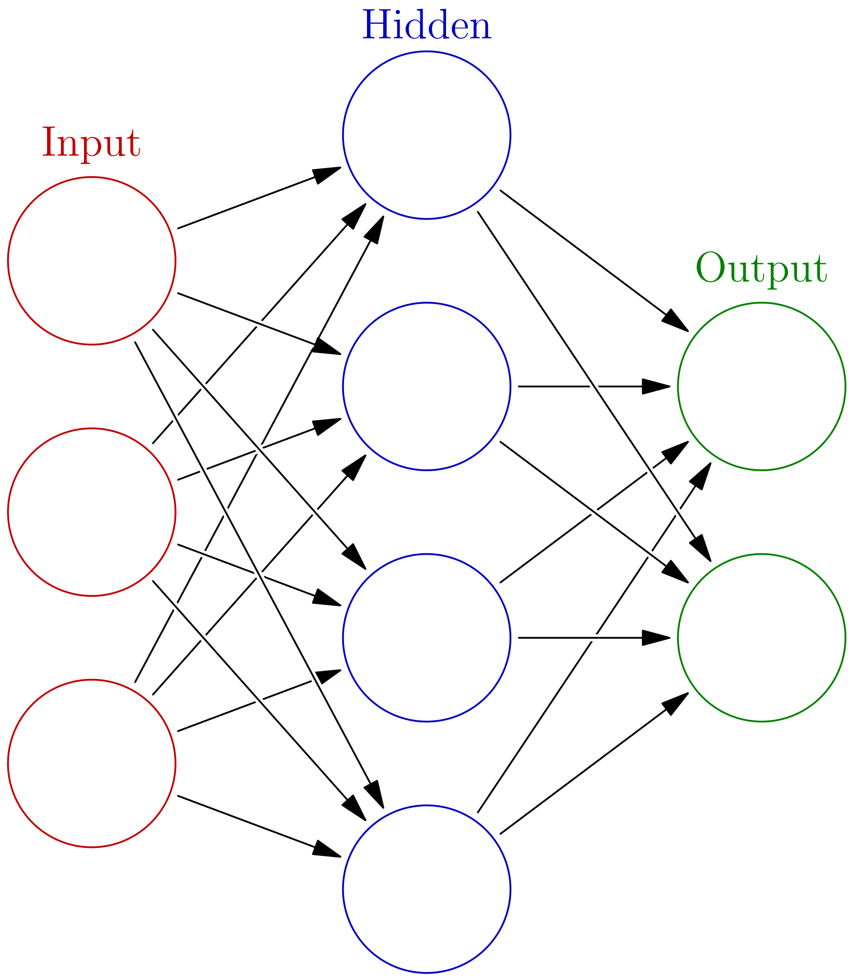
Input Layer

- Main Purpose:
  - Take in the raw data
  - Ensures input shape is compatible with the architecture
- Doesn’t transform the features
- Each feature/dimension of the input data corresponds to a node in input layer
- It distributes the input data to all the nodes in the first hidden layer (each input node is linked to every node in the first hidden layer)



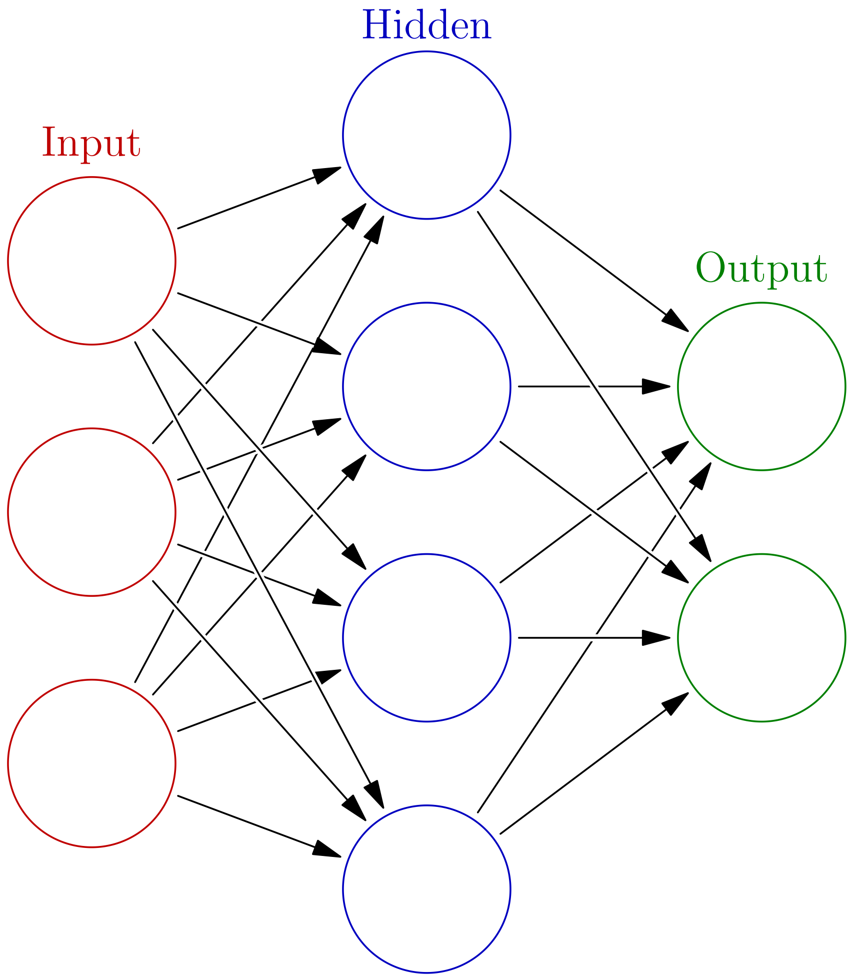
Hidden Layer

- Lies between the input and the output layer
- Feature Extraction
    - The first hidden layer captures the basic features (e.g. edges in an image)
    - Deeper layers, extract more complex, abstract features (e.g. shapes, eyes, faces)
    - Eliminates the need for manual feature engineering
- *# hidden layers* = depth of the network
- *# neurons/nodes* per layer = Width of the network
- Networks with many hidden layers = Deep neural networks (DNNs)



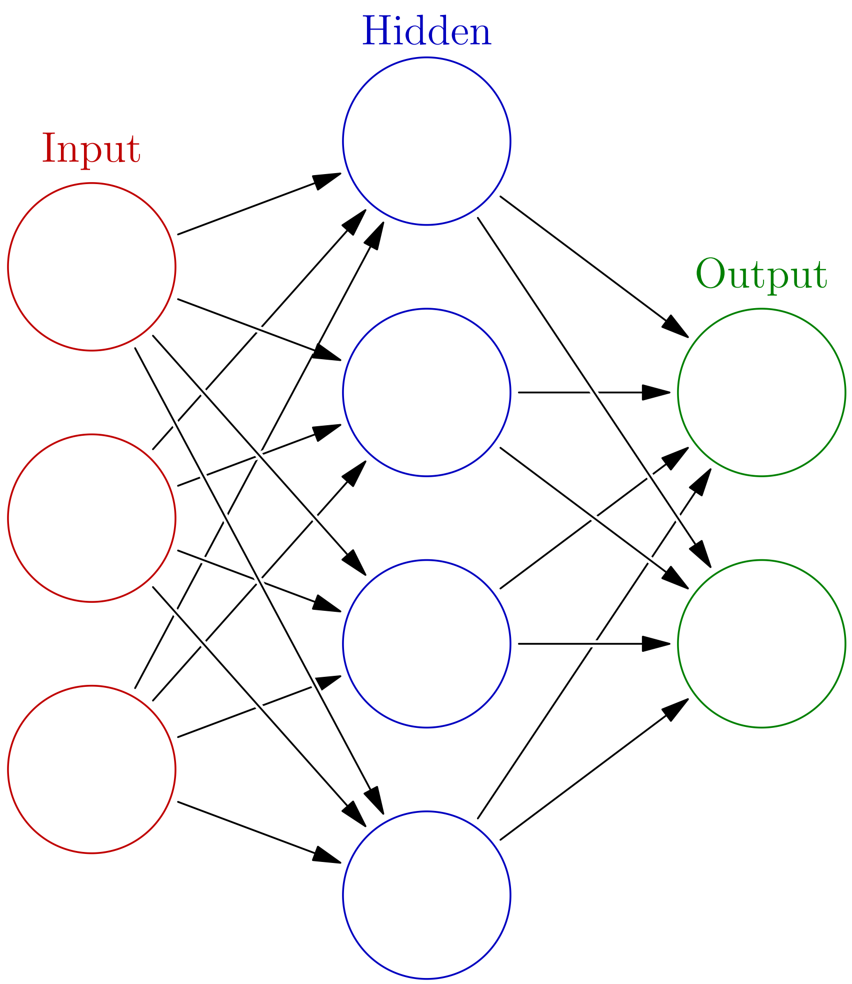
Hidden Layer

- Tasks requiring abstract reasoning
  - ***Depth*** is important
- Tasks requiring fine-grained feature analysis
  - ***Width*** is important



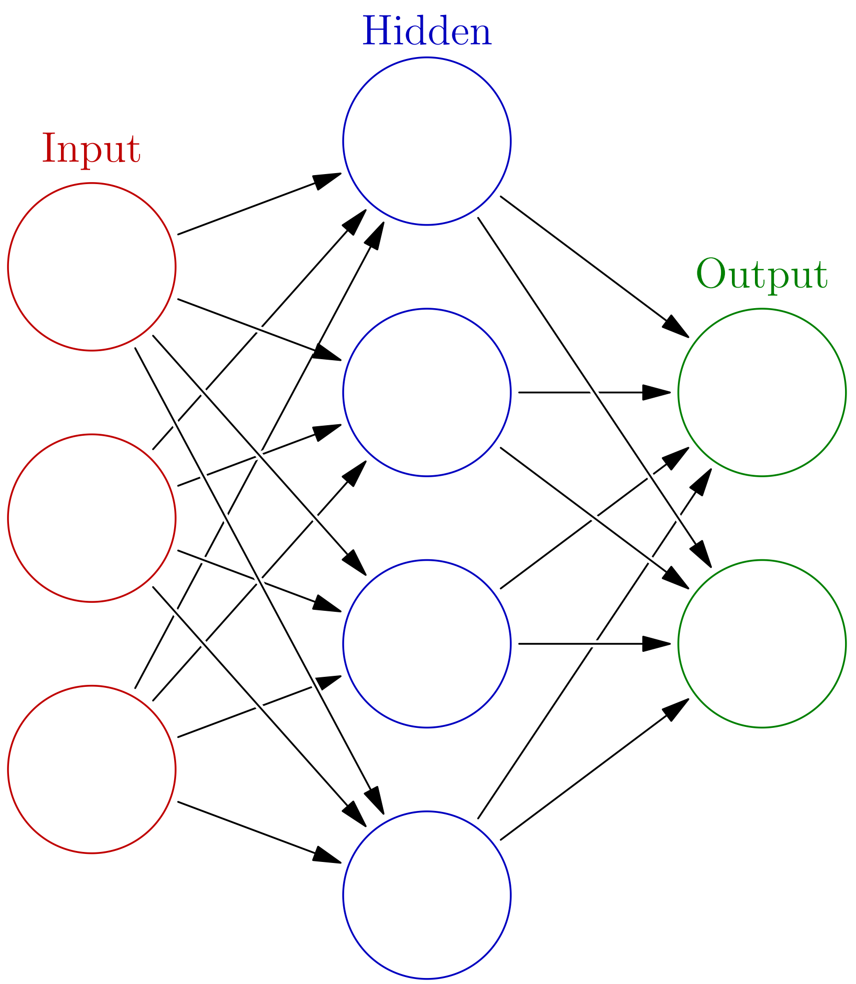
Output Layer

- Final Layer of the network
- Produces the network predictions
- Structure
  - **Regression**
    - **Scalar Output:** Single Node
    - **Vector Output**: Multiple Nodes
  - **Classification**
    - # Nodes = # Classes
- Output values must often be post-processed to get a meaningful result

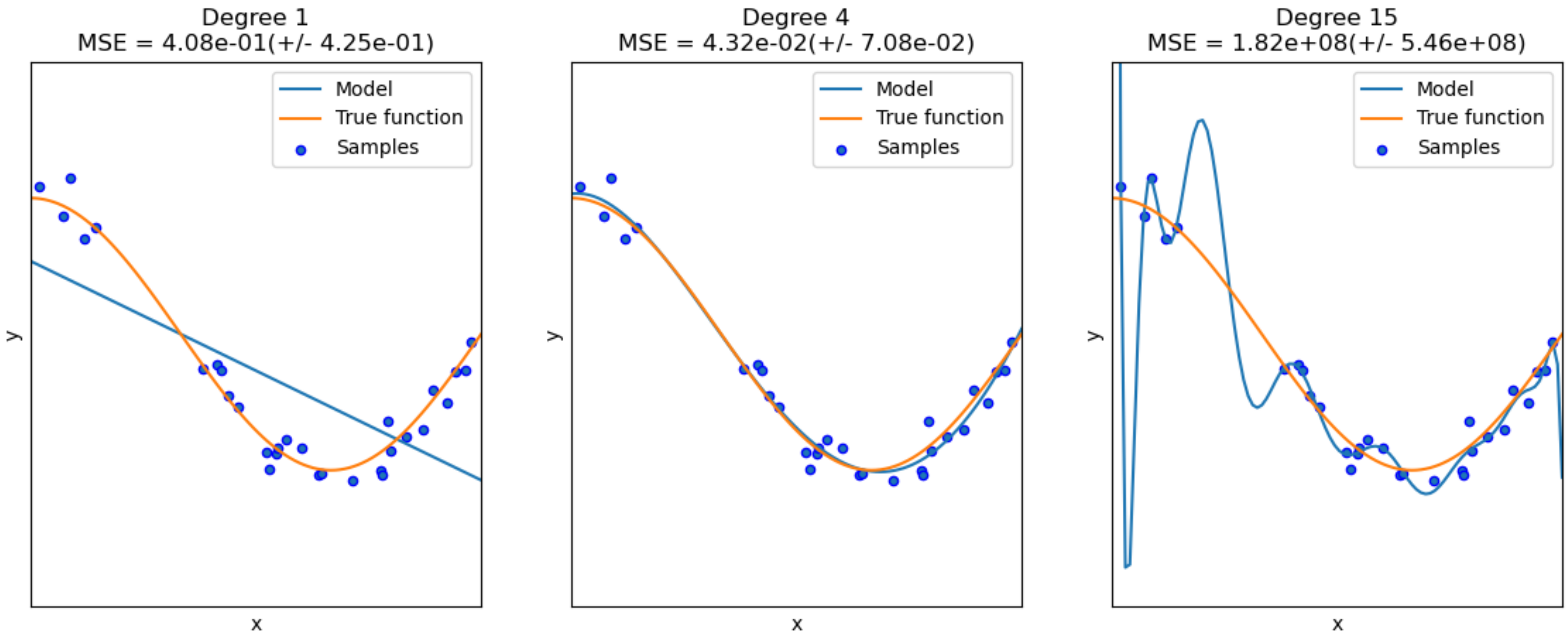


Output Layer

- Some Designs:
  - **Single Node** (Regression)
  - **Single Node** (Binary Classification)
  - **Multi-Node** (Multi-Class Classification)
  - **Multi-Node** (Multi-Label Classification)
  - **Custom**



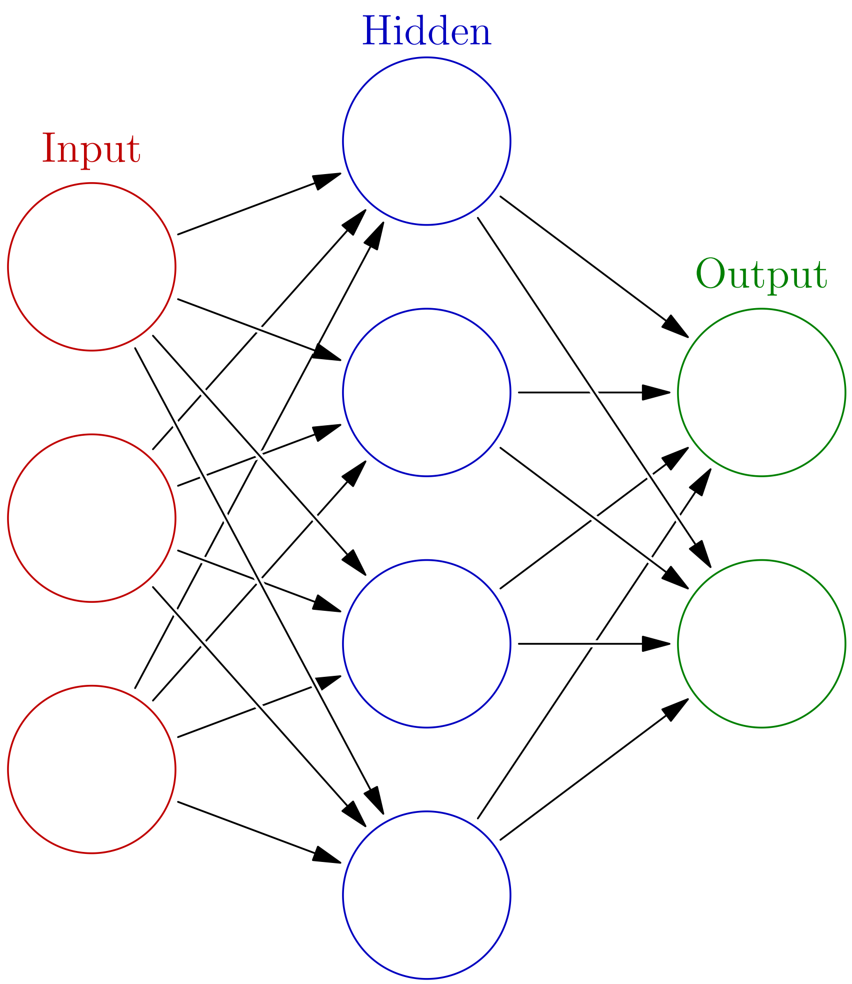
Overfitting vs. Underfiting



https://scikit-learn.org/dev/auto_examples/model_selection/plot_underfitting_overfitting.html

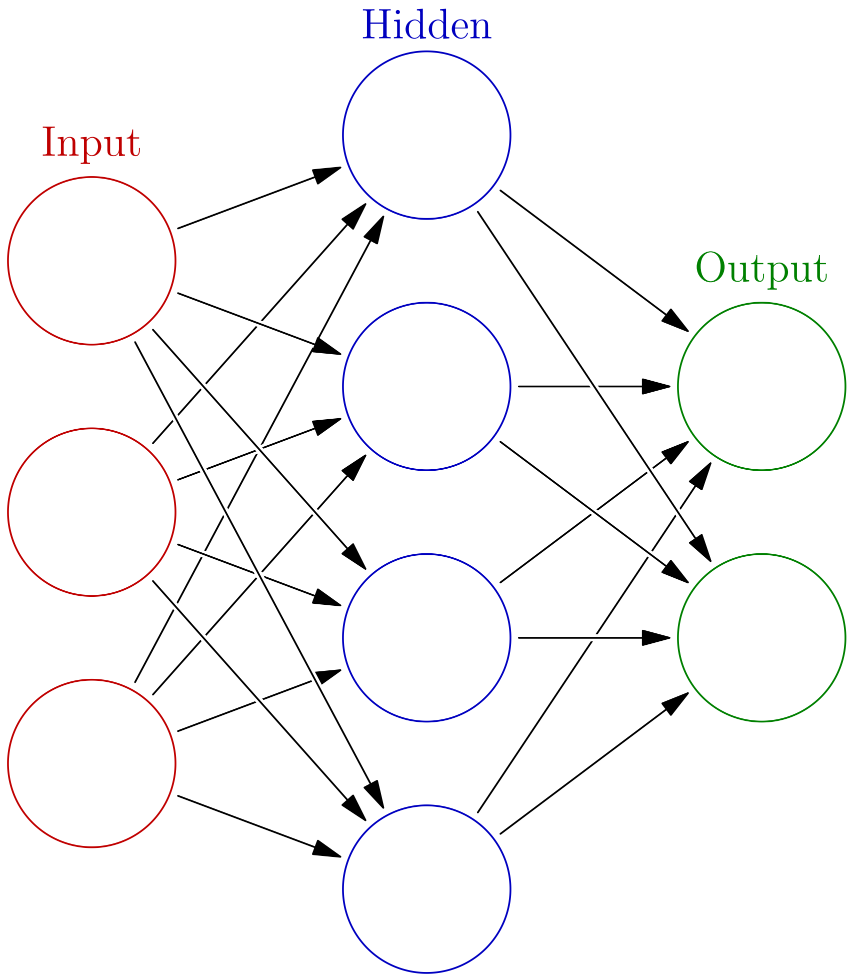
Complexity Trade-Offs

- Between the ability to learn and practical limits (i.e. Computation, Generalization, Interpretability)
- **Complexity vs. Generalization**
    - **High Complexity:** Learns well, Risk of overfitting
    - **Low Complexity**: Learns poorly, Risk of underfitting
- **Depth vs. Width**
    - **Deeper Networks:** Learn complex features, Risk of Vanishing/Exploding Gradients
    - **Wider Network:** Excel in capturing fine-grained patterns, But can be computationally expensive, Risk of Overfitting



Complexity Trade-Offs

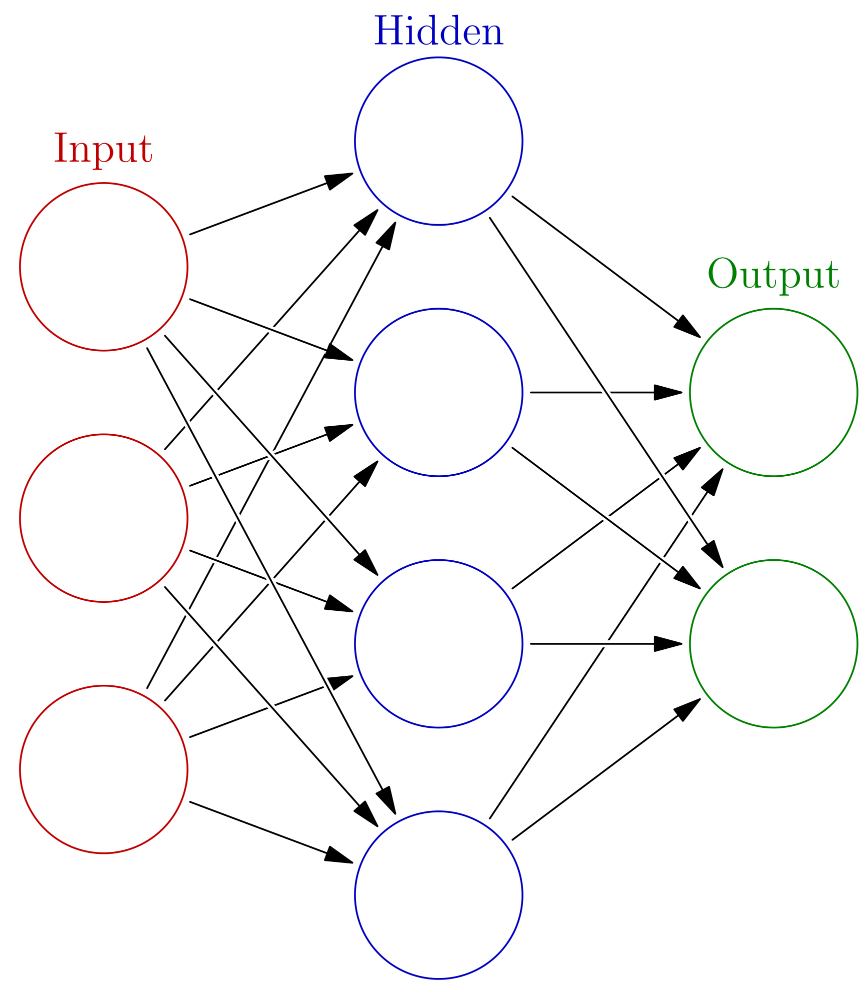
- Between the ability to learn and practical limits (i.e. Computation, Generalization, Interpretability)
- **Accuracy vs. Efficiency**
  - **High Accuracy:** Requires larger models, Increase computation, memory usage and energy consumption
  - **Efficiency**: Cut some part of the model, lose some accuracy in favor of faster inference and reduce resource requirments
- **Interpretability vs. Complexity**
  - **Complex model:** Nice but like a Black box
  - **Simple model:** Weak in complex tasks but interpretable



МФТИ

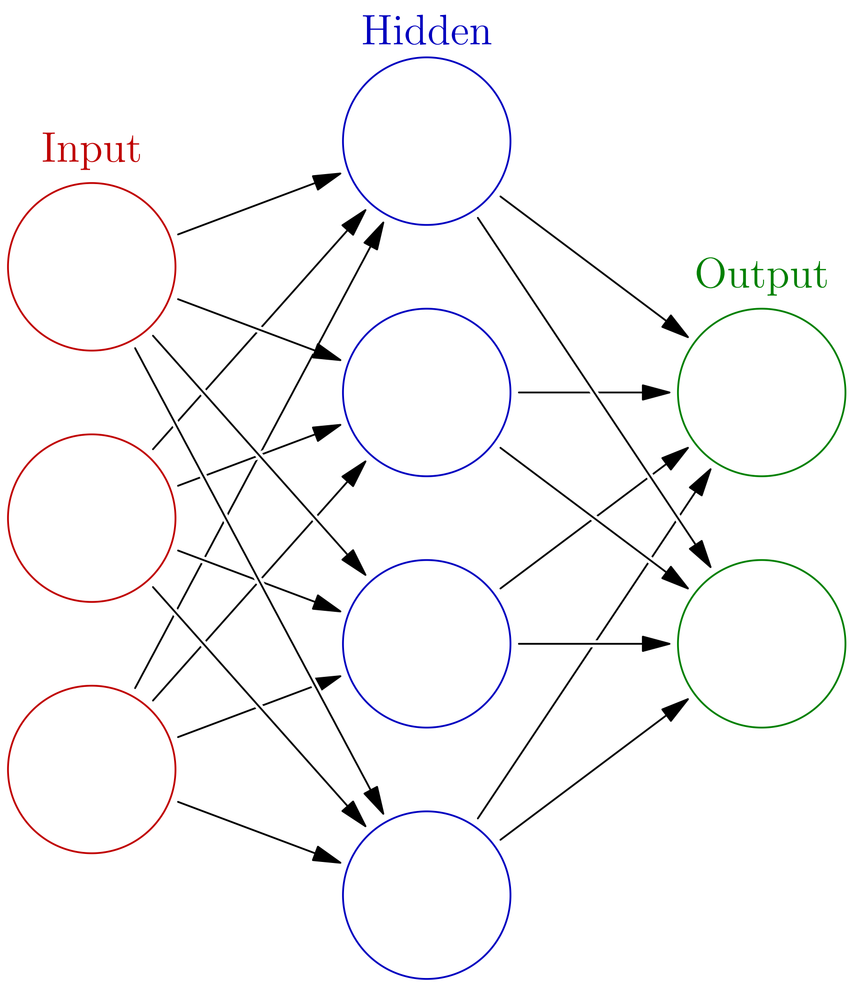
Complexity Trade-Offs

- Between the ability to learn and practical limits (i.e. Computation, Generalization, Interpretability)
- **Robustness vs. Simplicity**
  - **Robust Model:** Account for noisy or adversarial data, but computationally intensive and complex
  - **Simple Model**: Struggles with Noisy data but faster and easier to implement



Strategies in Managing the Trade-Offs

- **Regularization**
- **Model Pruning**
- **Early Stopping**
- **Transfer Learning**
- **Automated Optimization**
  - **Neural Architecture Search (NAS)**



https://www.youtube.com/watch?v=wL-p5cjDG64

МФТИ

Mathematics of MLP

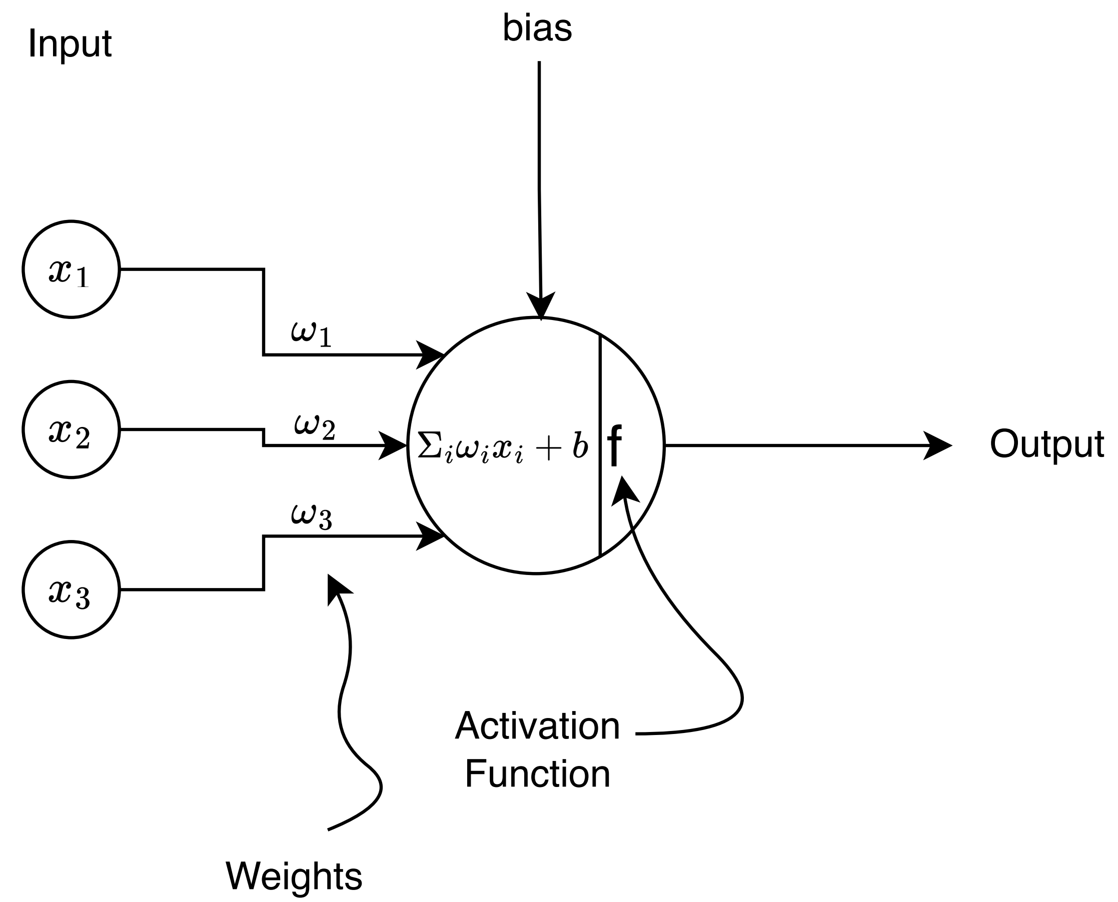
- Each layer has neurons, with every neuron in that layer, linked to the neurons of the next

- For a given layer ***"l"***

  - Input: x^{l-1}

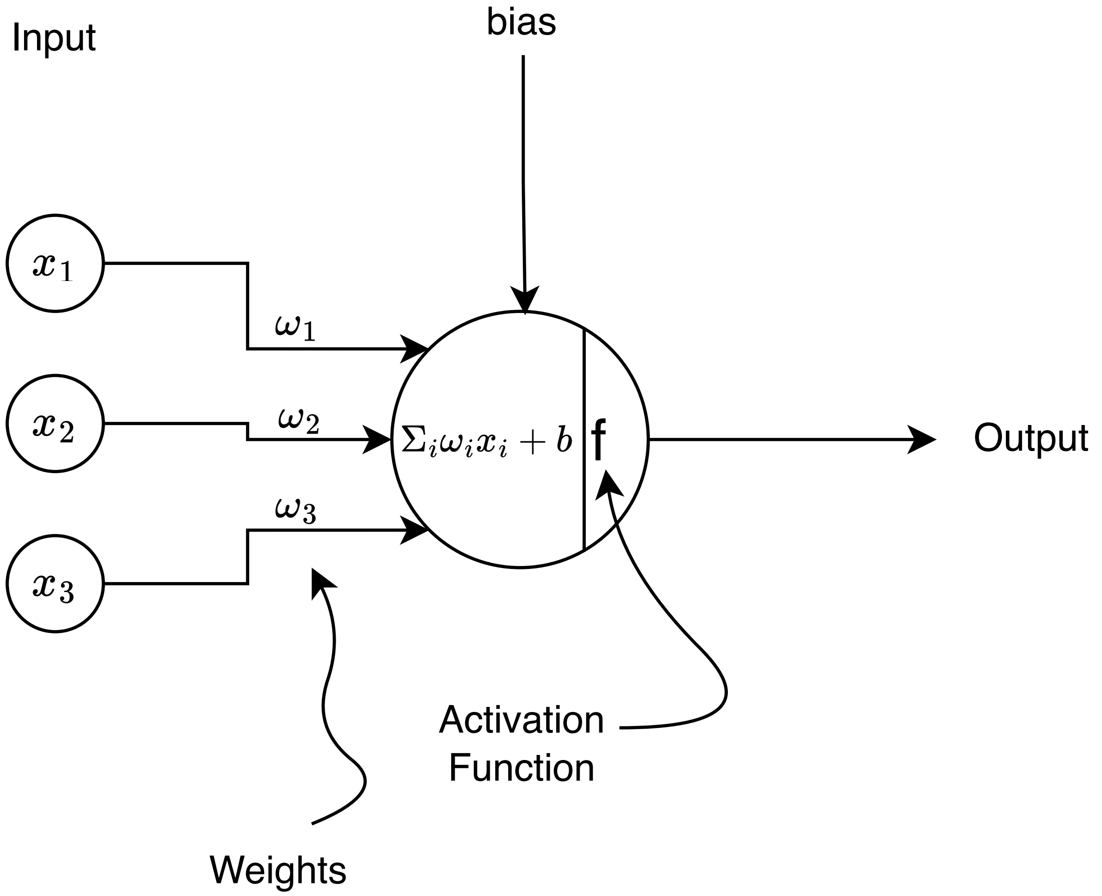
  - Linear Transformation: $z^l = \omega^{(l)} x^{(l-1)} + b^{(l)}$

  - Non-Linear Transformation: $x^{(l)} = \phi(z^{(l)})$



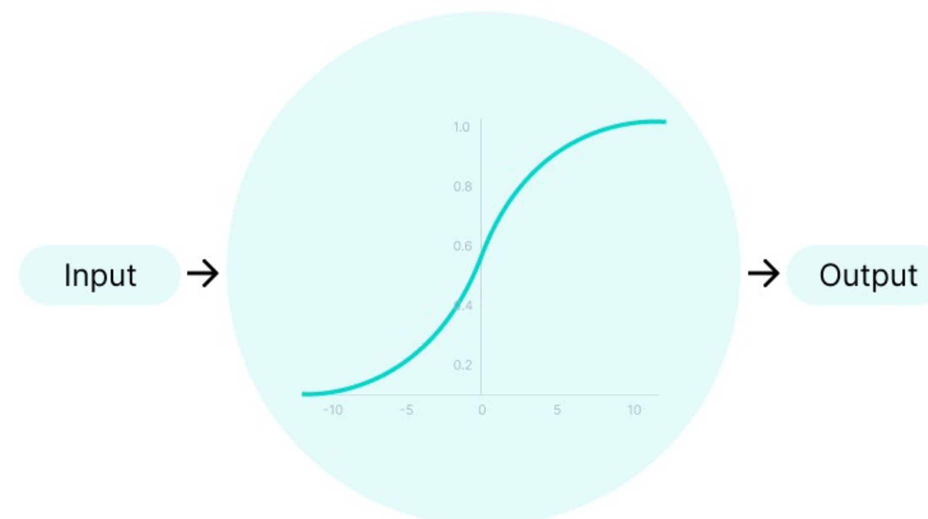
Mathematics of MLP

- **Weights**:
    - A matrix of learnable parameters $\omega^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$
    - Represents the strength of connections between neurons in layer *“l-1”* and *“l”*
- **Biases:**
    - A vector of learnable parameters, where $b^{(l)} \in \mathbb{R}^{n_l}$
    - Allows the layer to shift the activation values → Enhancing flexibility



Mathematics of MLP

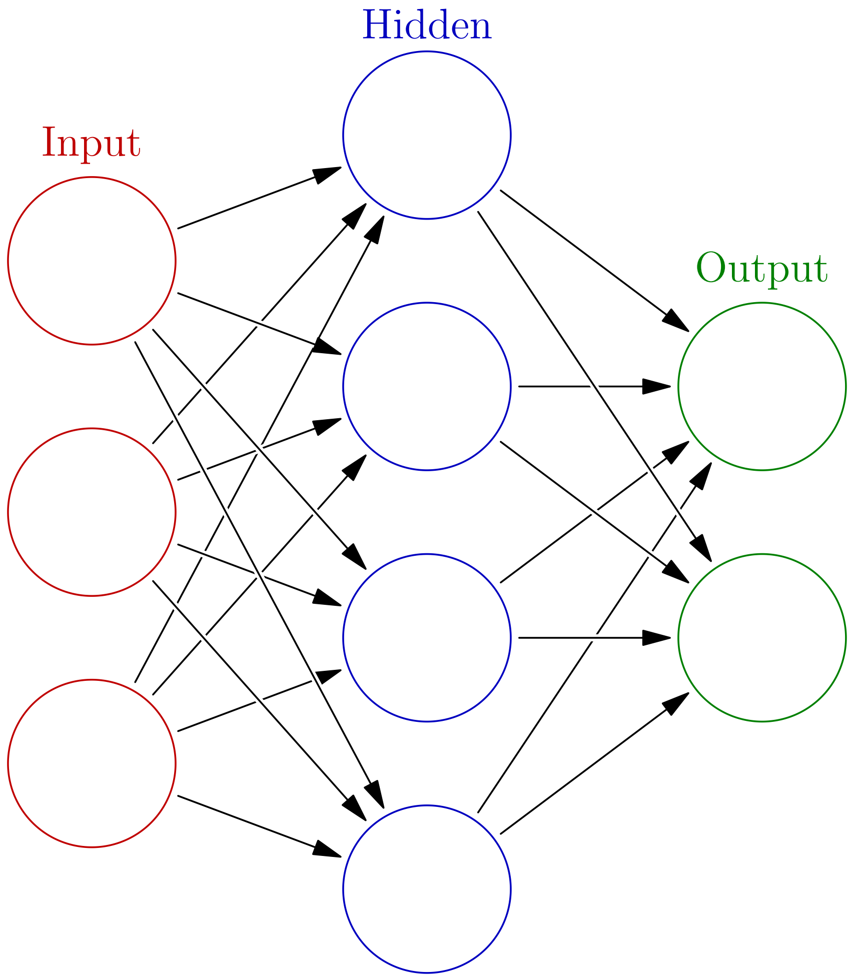
- **Activations:**
  - ***ReLU***: $\phi(x) = max(0, x)$
  - ***Sigmoid:*** $\phi(x) = \dfrac{1}{1 + e^{-x}}$
  - ***Tanh:*** $\phi(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
  - ***Softmax:*** $\phi(x_i) = \dfrac{e^{x_i}}{\sum_j e^{x_j}}$



Activation Function

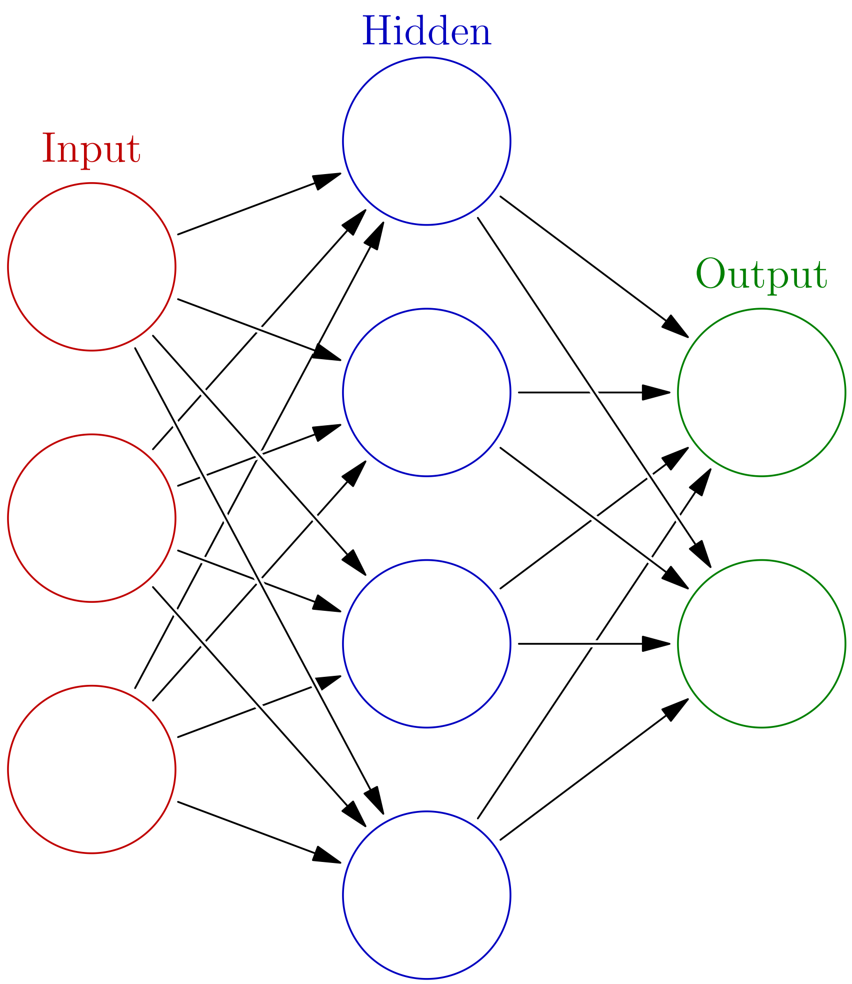
Mathematics of MLP

- **Forward Propagation:**
  - ***Input Layer***
    - $x^{(0)} = input$
  - ***Hidden Layer:***
    - $x^l = \phi(\omega^{(l)} x^{(l-1)} + b^l)$
  - ***Output Layer***
    - $\hat{y} = \phi(\omega^{(L)} x^{(L-1)} + b^L)$



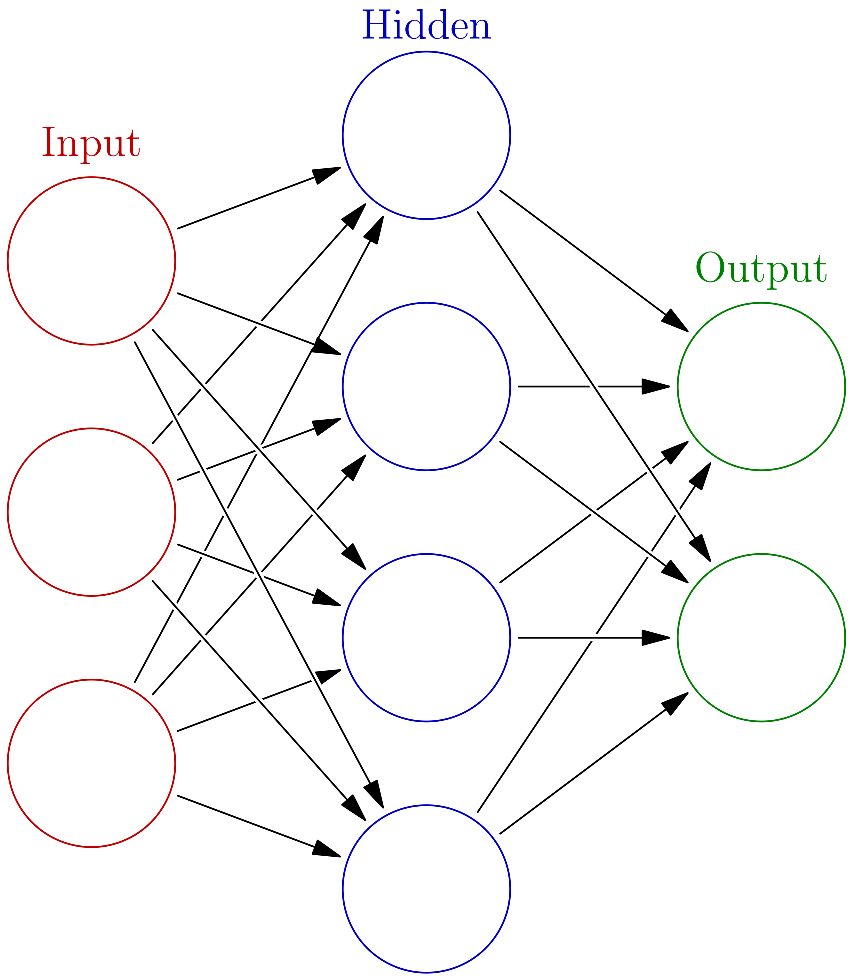
Considerations

- **Input Layer**
    - **Size**: ~# features in input data
- ***Hidden Layer***
    - *Determine the network's depth and width*
    - ***Shallow Networks**: Suitable for simpler tasks or smaller datasets.*
    - ***Deep Networks**: Capture hierarchical relationships but require careful regularization and optimization.*
- ***Output Layer:***
    - **Size**: # output classes
    - *Activation functions should align with the task type (e.g., softmax for classification, linear for regression).*



Considerations

- **Neurons per Layer**
    - **Too Few Neurons:** Can lead to underfitting, due to lack of ability in learning complex patterns
    - **Too Many Neurons:** Can lead to overfitting, model memorizes the training data instead of generalizing
    - **Practice**: For a funnel-shaped structure with fewer neurons in deeper layers
- ***Regularization***
    - ***Dropout***
    - ***Weight Decay (L2 Regularization)***
    - ***Batch Normalization***
    - ***Early Stopping***
    - ***Data Augmentation***



Practice

```python
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, output_size)
        self.activation = nn.ReLU()
        self.output_activation = nn.Softmax(dim=1)  # For multi-class probabilities

    def forward(self, x):
        x = self.activation(self.layer1(x))
        x = self.output_activation(self.layer2(x))
        return x
```