ARTIFICIAL INTELLIGENCE NEEDS REAL INTELLIGENCE

Linear Neural Network I

Professor: Tianyuan ZHANG
tianyuan.zhang@kedgebs.com

KEDGE BUSINESS SCHOOL

Recap of Ensemble Learning

- Ensemble Learning
  - A machine learning paradigm where multiple models (often called "**weak learners**") are trained to solve the same problem and combined to get better results.
- Rationale
  - No single model can capture all the patterns in the data perfectly.
  - Leverage the strength and compensate for the weaknesses of individual models.
  - A group of "weak learners" can come together to form a "strong learner".
  - **Wisdom of the crowd**

Recap of Ensemble Learning

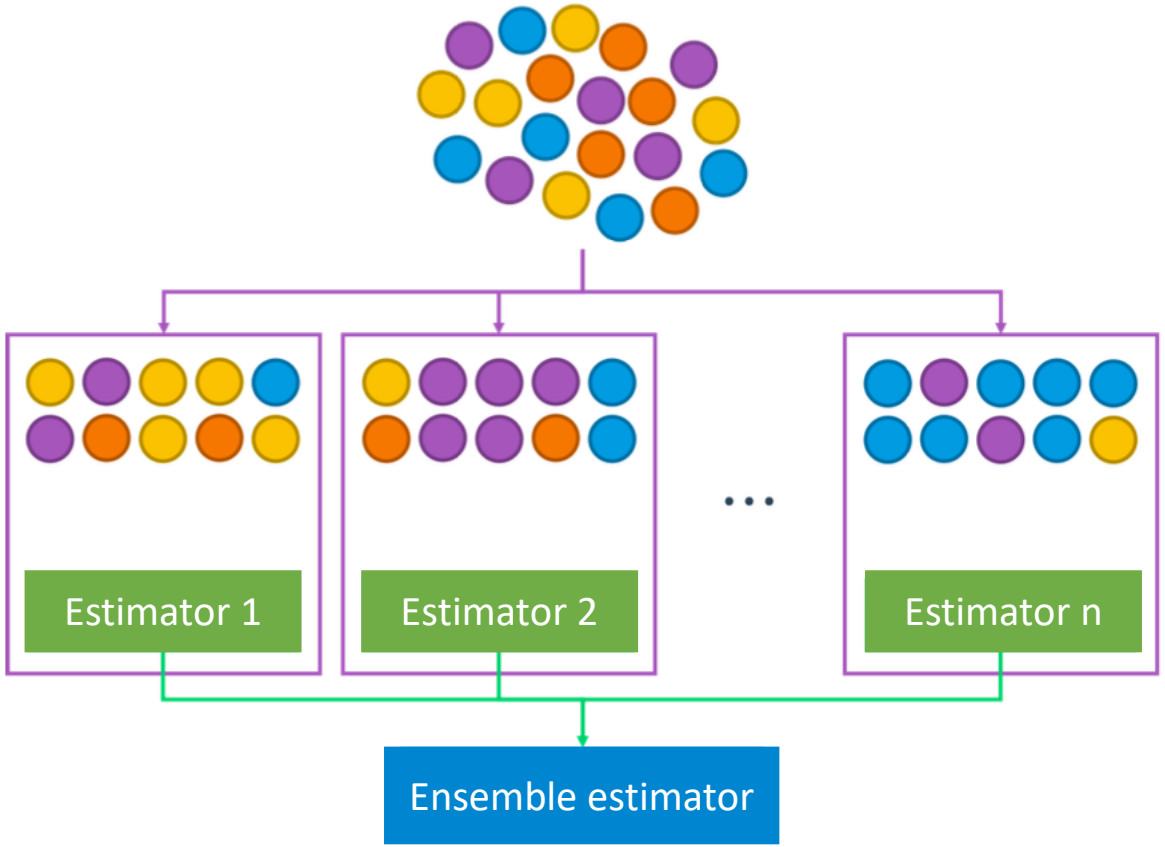
- Two questions to answer when designing ensemble learning methods
  1. How to generate multiple base models (estimators)?
  2. How to integrate / combine their predictions to make final prediction?
- Different decisions lead to different **types** of ensemble learning methods
  - Bagging
  - Boosting
  - Stacking

Recap of Ensemble Learning

- **Bootstrap Aggregating → Bagging**
- An ensemble learning technique
  - **Parallel model training**
    - Multiple models / estimators are trained simultaneously on different subsets of the original training dataset.
  - **Bootstrap sampling**
    - Each subset of training data is generated by randomly sampling with replacement.
  - **Aggregation of predictions**
    - Regression: Averaging the predictions
    - Classification: Majority voting

Recap of Ensemble Learning

- **Bagging**



Original dataset

↓

Bootstrap sampling

↓

Parallel training

↓

Aggregation of predictions

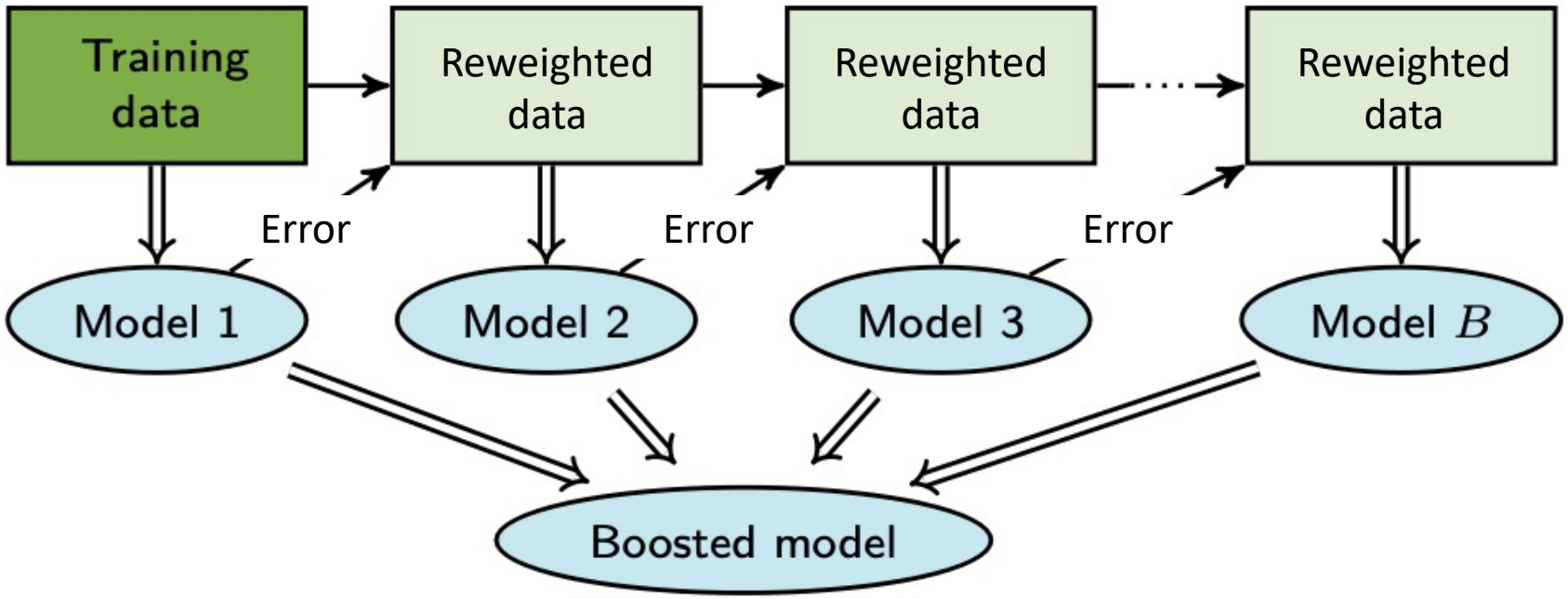
Image source: https://en.wikipedia.org/wiki/Bootstrap_aggregating

Recap of Ensemble Learning

- **Boosting**
  - A type of ensemble learning methods that boosts weak learners to strong learners
  - Key features
    - **Sequential model training**
      - Each model in the sequence is trained on the dataset with a focus on the instances that previous models misclassified.
    - **Error correction focus**
      - The algorithm identifies the errors or mispredictions of the previous model and increases their weight, making them more significant in the training of the next model.
    - **Cumulative learning**
      - Unlike parallel ensemble methods, boosting involves cumulative learning where each model builds on the knowledge of its predecessors.

Recap of Ensemble Learning

- **Boosting**



- An **iterative** process, where the training set is **reweighted** in each iteration

Recap of Ensemble Learning

- **Stacking**
  - Same training set + different algorithms → different weak learners
  - Integrate base predictions using a meta-learner
  - Process
    1. Choose M different algorithms, train M different base learners $\left\{\hat{y}^j(\mathbf{x})\right\}_{j=1}^n$ on the training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
    2. Form a new training data set using the base predictions
       - New training data set $\{(\mathbf{x}_i', y_i)\}_{i=1}^n$
       - $\mathbf{x}_i' = \left\{\hat{y}_i^1(\mathbf{x}_i), \ldots, \hat{y}_i^M(\mathbf{x}_i)\right\}$
    3. Train a meta-learner $\hat{Y}(\mathbf{x})$ on the new training data set $\{(\mathbf{x}_i', y_i)\}_{i=1}^n$

Recap of Ensemble Learning

- **Stacking**

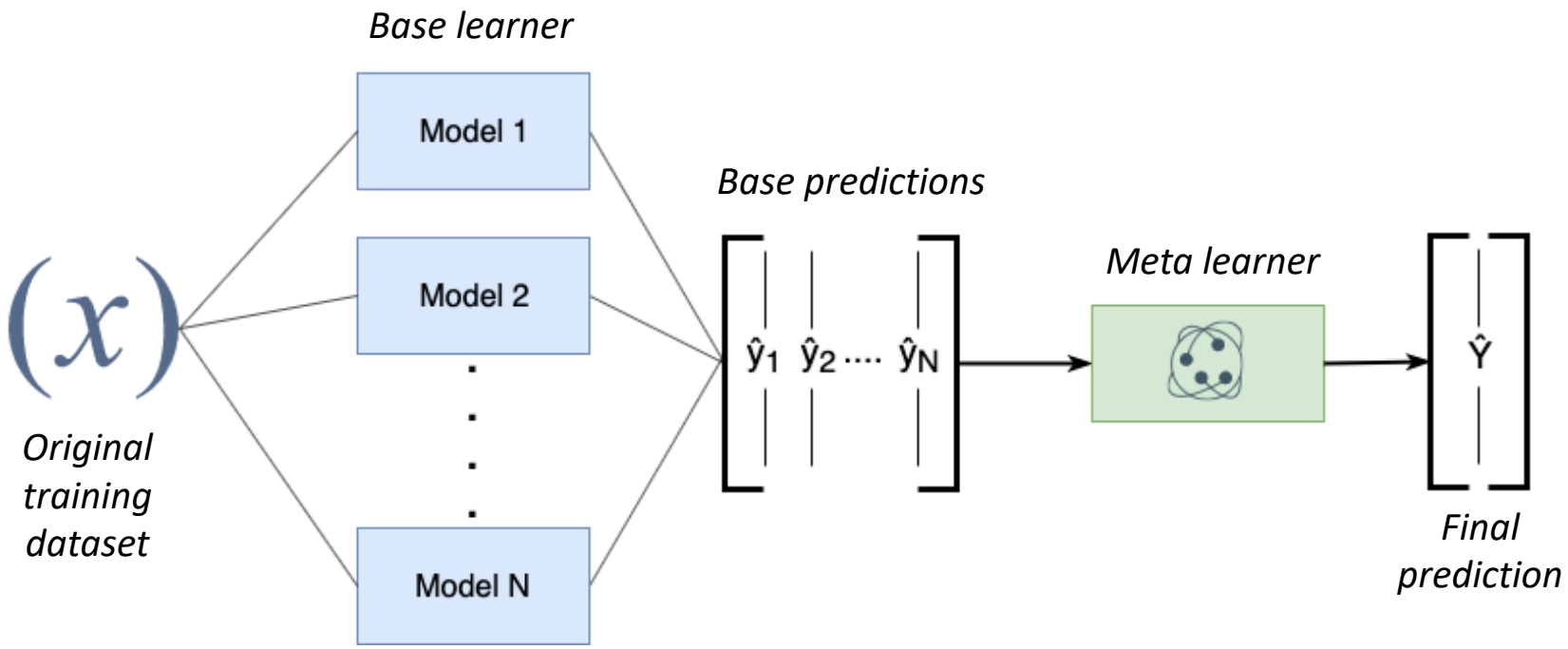


Image source: https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html#stacking

Recap of Ensemble Learning

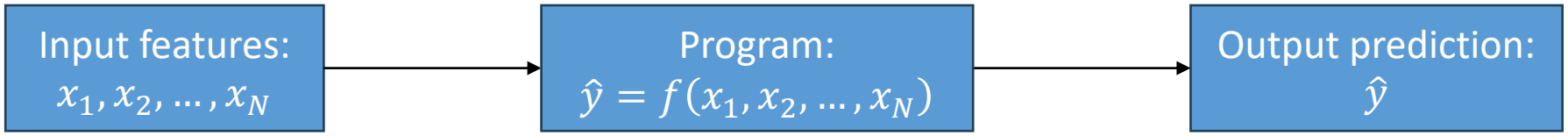
- Ensemble Learning
  - A way to combine a group of “weak learners” into one “strong learner”
  - Reduce the prediction errors of a single model
    - Bias: under-fitting model
    - Variance: over-fitting model
  - If the available machine learning algorithms fail to solve the complex problem, ensemble learning can help.
- What if **a single model** can be **complex enough** to avoid high bias and **generalized well** to avoid high variance?
  - **Human being**
  - **Artificial Neural Network → Deep Learning**

Outline

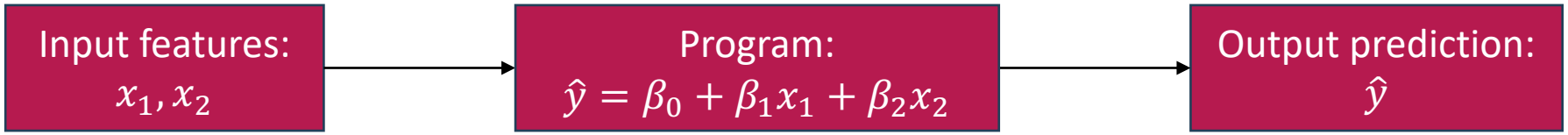
- **Classical Machine Learning vs. Deep Learning**
- Artificial Neural Network
- Implement an ANN
- Preliminaries with PyTorch

Classical Machine Learning vs. Deep Learning

- Break down classical ML into several key concepts
  - **Parameter**
    - The **behavior** of a **flexible program** is determined by a number of parameters



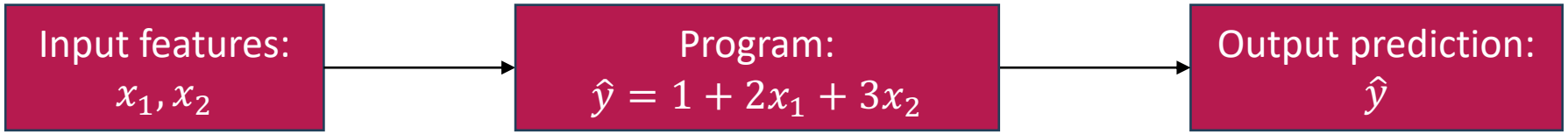
    - Example: Linear regression



      - $\beta_0, \beta_1, \beta_2$ are parameters that determine the behavior of the program
      - Given the same inputs, different values of parameters lead to different outputs

Classical Machine Learning vs. Deep Learning

- Break down classical ML into several key concepts
  - **Parameter**
    - The behavior of a flexible program is determined by a number of parameters
  - **Model**
    - Once the **parameters are fixed**, the flexible program becomes a model
    - Example: Linear regression



    - $\beta_0, \beta_1, \beta_2$ are fixed at 1, 2, 3
    - Then we can use this program to calculate the outputs given the inputs
    - It becomes a ML model that can make predictions given the inputs

Classical Machine Learning vs. Deep Learning

- Break down classical ML into several key concepts
  - **Parameter**
    - The behavior of a flexible program is determined by a number of parameters
  - **Model**
    - Once the parameters are fixed, the flexible program becomes a model
  - **A family of model**
    - The **set of all distinct models (input-output mapping)** that we can produce by manipulating the parameters of the same flexible program
    - Example: Linear regression
      - Model 1 $\rightarrow \hat{y} = 1 + 2x_1 + 3x_2$
      - Model 2 $\rightarrow \hat{y} = 4 - 1x_1 + 2x_2$
      - Model 3 $\rightarrow \hat{y} = 0 + 7x_1 + 5x_2$

Classical Machine Learning vs. Deep Learning

- Break down classical ML into several key concepts
  - **Parameter**
    - The behavior of a flexible program is determined by a number of parameters
  - **Model**
    - Once the parameters are fixed, the flexible program becomes a model
  - **A family of model**
    - The **set of all distinct models (input-output mapping)** that we can produce by manipulating the parameters of the same flexible program
  - **Learning process**
    - The "meta-program' that uses training data to **choose the best values of parameters**, which will **produce a model**
    - Example: Linear regression, use ordinary least squares method to estimate $\beta_0, \beta_1, \beta_2$

Classical Machine Learning vs. Deep Learning

- Break down classical ML into several key concepts
  - **Parameter**
  - **Model**
  - **A family of model**
  - **Learning process**
    - The “meta-program’ that uses training data to choose the best values of parameters, which will produce a model
    - **Objective function**
      - A formal measure of how good (or bad) the models are.
      - Example: RMSE for regression, Accuracy for classification
    - **Optimization algorithm**
      - Search for the best values of parameters for minimizing / maximizing the objective function

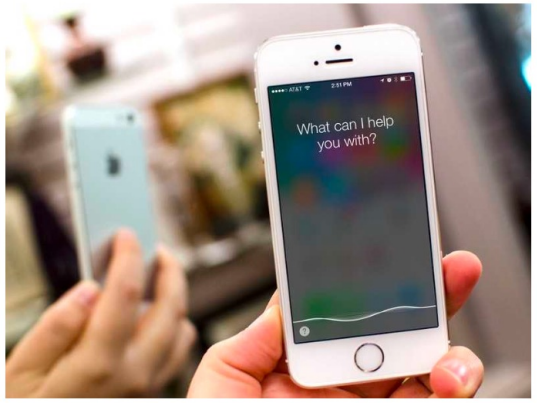
Classical Machine Learning vs. Deep Learning

- Classical ML → **Deep Learning**
  - Parameter → Parameter of **artificial neuron**
  - Model → **Artificial neural networks** (consists of artificial neurons)
  - Learning process
    - Objective function → **Loss function**
      - **Conventionally lower is better**
    - Optimization algorithm → **Any optimization algorithm**
      - The dominant approach is based on **gradient descent**
- **Advantages** of deep learning
  - Artificial neural network and its variants can form very complex models (**deep**)
  - Various techniques to avoid over-fitting and ensure generalization

Classical Machine Learning vs. Deep Learning

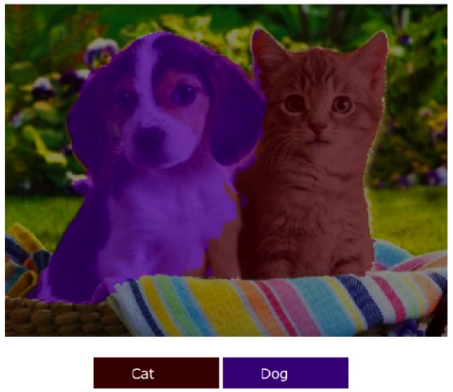
- Examples of deep learning applications

Speech recognition
Natural language processing



Deep reinforcement learning

Computer vision
Object recognition
Semantic segmentation



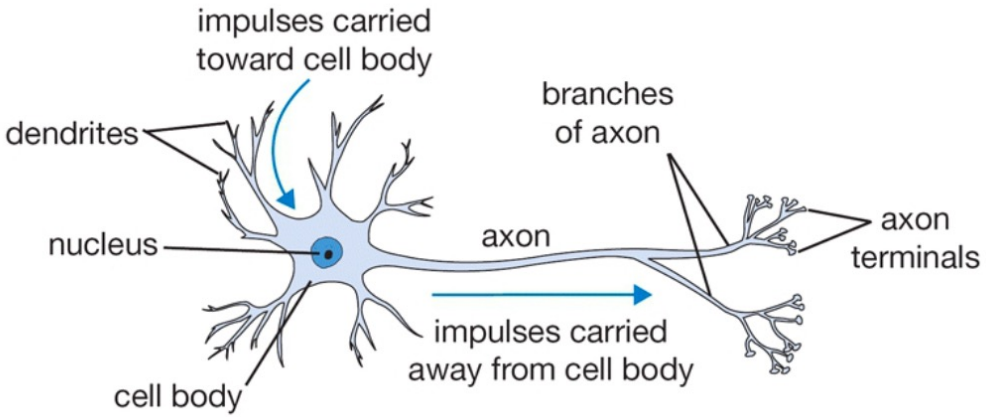
Generative AI
Style transfer

Outline

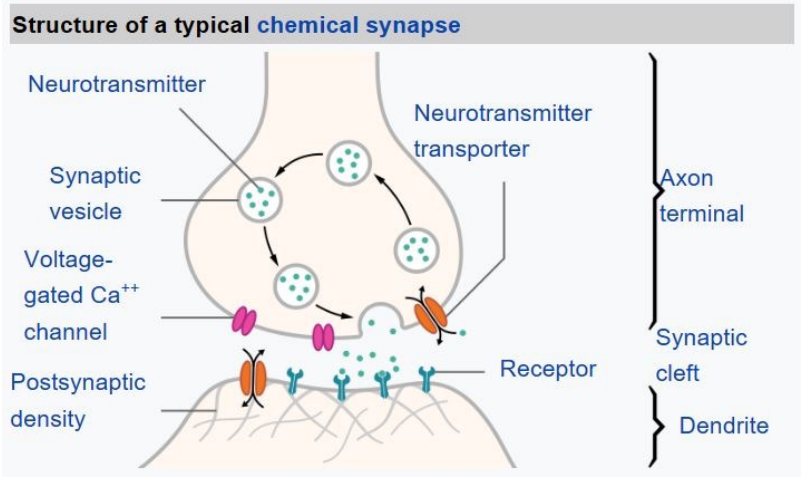
- Classical Machine Learning vs. Deep Learning
- **Artificial Neural Network**
- Implement an ANN
- Preliminaries with PyTorch

Artificial Neural Network

- Inspiration: The Brain
  - Human brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^{4}$ other neurons



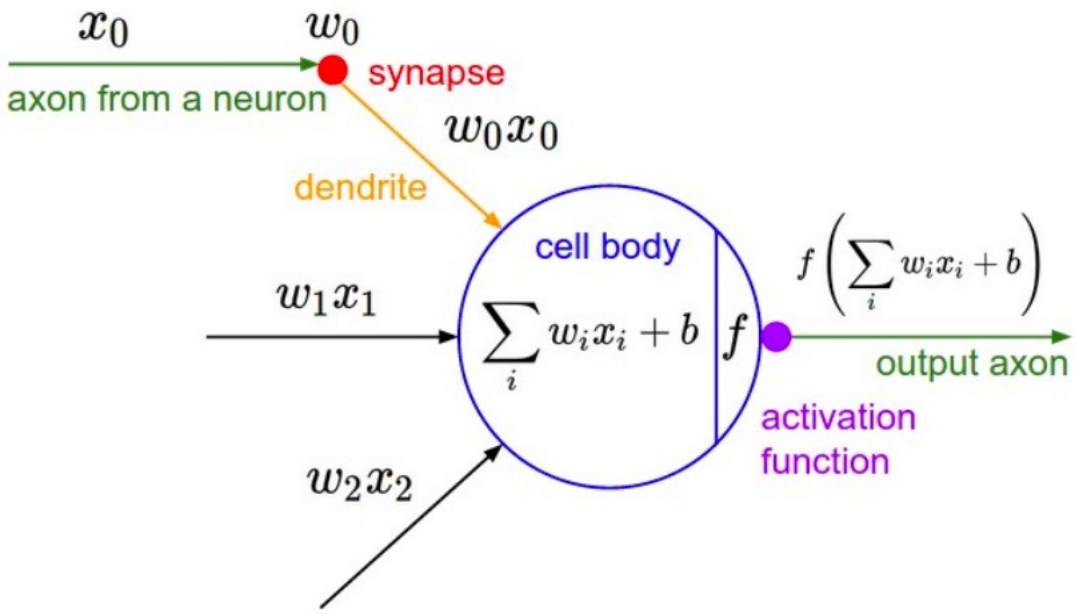
The basic computation unit of the brain: Neuron



Connections between axons and dendrites

Artificial Neural Network

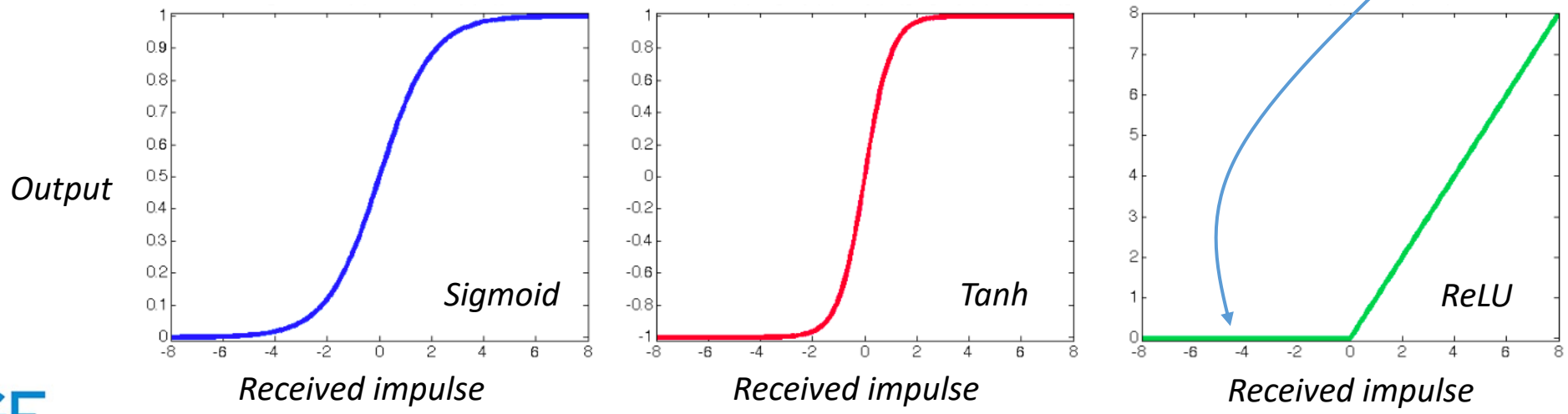
- **Artificial Neuron**
  - A mathematical model of a neuron
    - $f(\sum_{i=1}^{n} w_i x_i + b)$ is the **output** of a neuron
    - $(x_1, \dots, x_n)$ are an **input** data point with n features
    - $(w_1, \dots, w_n)$ are the **weights** of each feature
    - b is the **bias** added to the weighted sum
    - $(\sum_{i=1}^{n} w_i x_i + b)$ can be seen as the **impulse received** by the neuron
    - f is the **activation function**, determining whether the neuron is activated by the received impulse or not



Artificial Neural Network

- Commonly used **activation functions**
  - **Linear**: $f(z) = z$
  - Non-linear:
    - **Sigmoid**: $\sigma(z) = 1/(1 + e^{-z})$
    - **Tanh**: $\tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$
    - **ReLU** (Rectified Linear Unit): $\mathrm{RELU}(z) = \max(0, z)$

If the output of a neuron is 0 or close to 0, we can say the neuron is not activated



Artificial Neural Network

- Example Python scripts of a neuron with a sigmoid activation function

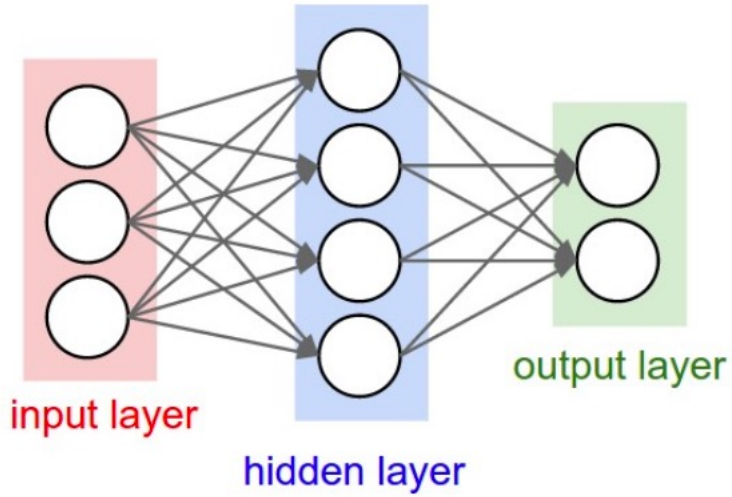
```python
class Neuron(object):
  # ...
  def forward(inputs):
    """assume inputs and weights are 1-D numpy arrays"""
    """assume bias is a number"""
    received_impulses = np.sum(inputs * self.weights) + self.bias
    output = 1.0 / (1.0 + math.exp(-received_impulses)) # sigmoid activation function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

Forward *means the artificial neuron is performing* ***inference***

- *Take features of an instance as inputs*
- *Compute the received impulse using weights and bias*
- *Transform the impulse into output through the activation function*

Artificial Neural Network

- **Artificial Neural Network (ANN)**
  - A network consists of multiple connected artificial neurons arranged into different layers

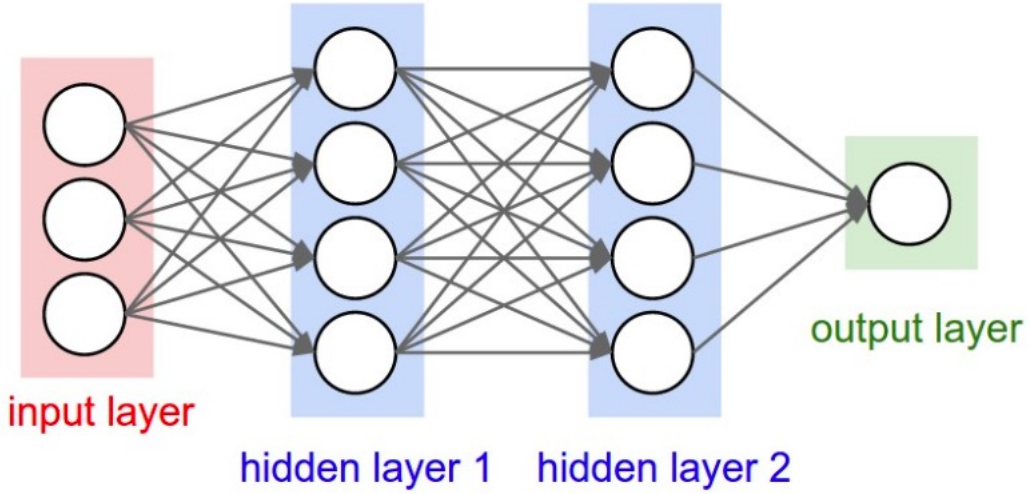


Terminology:

- input nodes
- output nodes
- hidden layers
- connections
- weights
- activation function

Artificial Neural Network

- **Artificial Neural Network (ANN)**
  - A network consists of multiple connected artificial neurons arranged into different layers



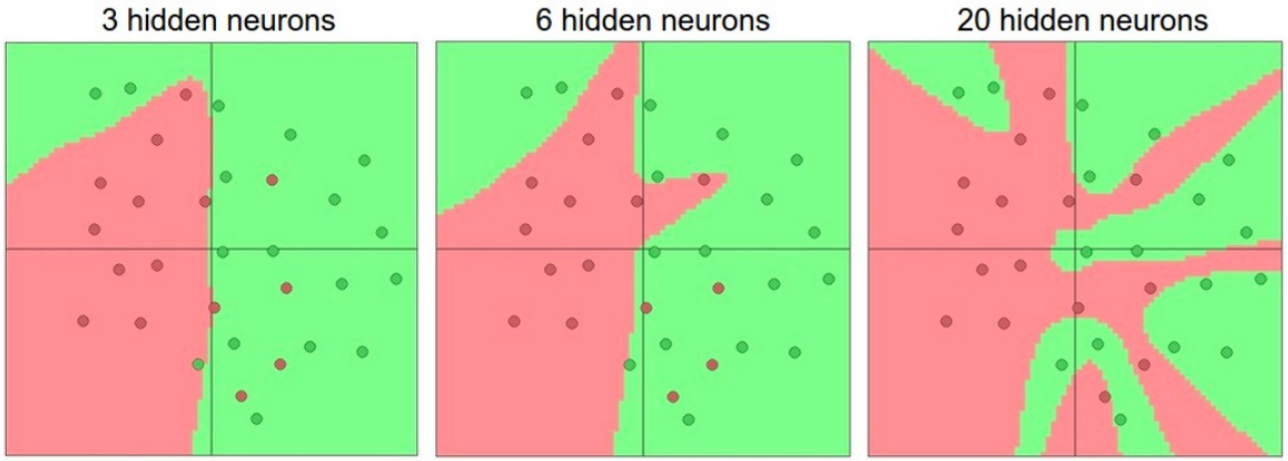
A 3-layer neural network with two hidden layers

Naming conventions: a N-layer neural network

- N-1 hidden layers
- 1 output layer
- We do not count the input layer

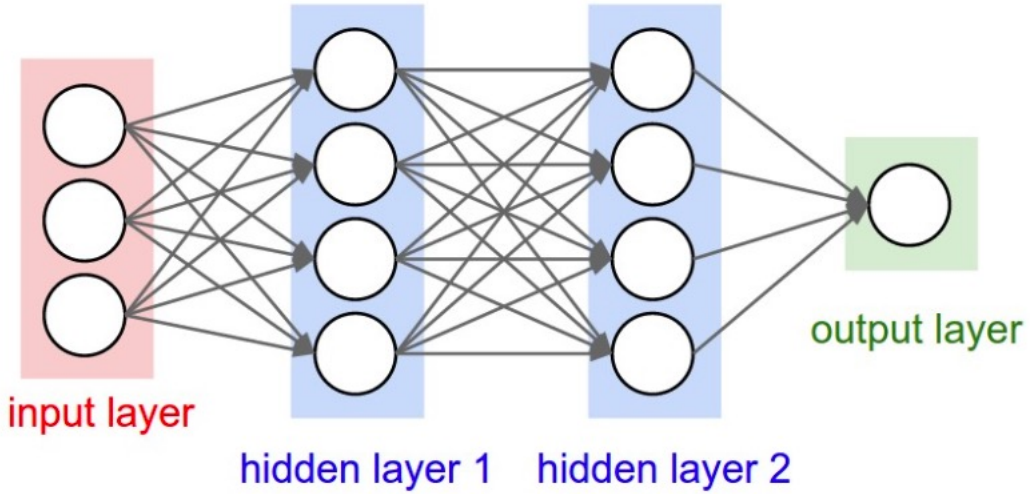
Artificial Neural Network

- **Artificial Neural Network (ANN)**
  - ANN is a family of model with excellent **representational power**
    - Depends on the number of hidden neurons and hidden layers, the network can by very simple or complex
  - A 2-layer ANN can be **a universal approximator**
    - By adjusting the number of neurons in the hidden layer and their parameters, it can represent any function



Artificial Neural Network

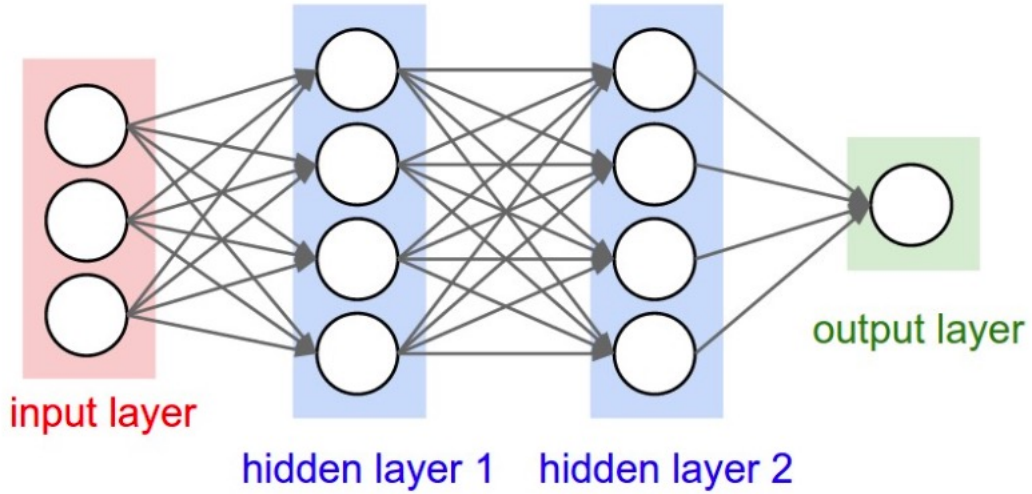
- **Artificial Neural Network (ANN)**
  - **Hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *The first three determined the structure of the ANN*
- *Pre-defined when we create an ANN*
- *Remain unchanged during the training process*

Artificial Neural Network

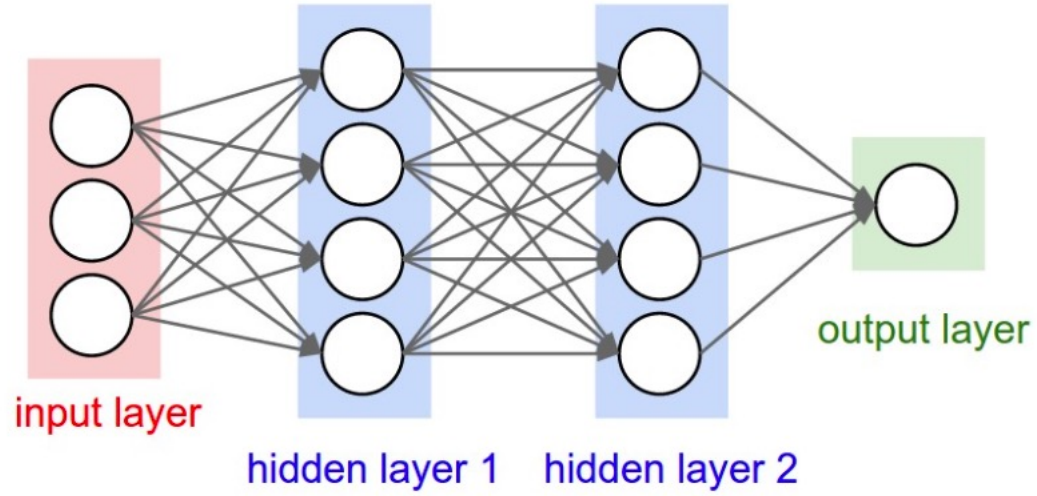
- **Artificial Neural Network (ANN)**
  - Hyper-parameters
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron
  - **Parameters → Learning parameters**
    - The weight of each connection
    - The bias of each neuron added to the weighted sum



- *Adjusted during the training process*
- *The values of parameters are learned from the training data*
- *More connections will have more weights*
- *More neurons will have more bias*

Artificial Neural Network

- **Artificial Neural Network (ANN)**
  - Hyper-parameters
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron
  - Parameters → Learning parameters
    - The weight of each connection
    - The bias of each neuron added to the weighted sum
  - The **capacity** (complexity, representational power) of a network can be assessed by **the number of learning parameters**



Outline

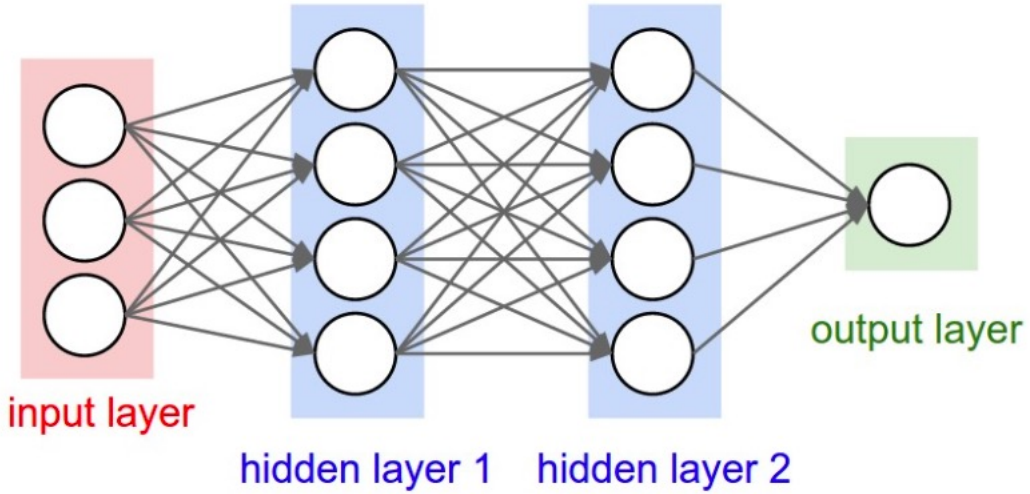
- Classical Machine Learning vs. Deep Learning
- Artificial Neural Network
- **Implement an ANN**
- Preliminaries with PyTorch

Implement an ANN

- How to **create** an ANN?
- How does ANN perform **inference**?
- How to **train** an ANN?

Implement an ANN

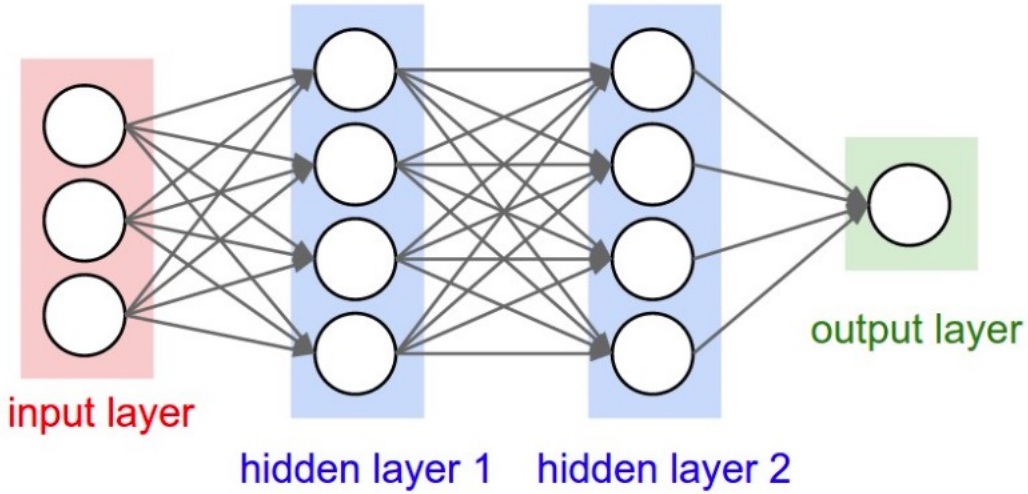
- How to **create** an ANN?
  - **Declare the hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *3 layers (2 hidden layer + 1 output layer)*
- *3 neurons in the input layer*
- *4 neurons in the 1st hidden layer, 4 in the 2nd*
- *1 neuron in the output layer*
- *Fully connected between layers*
- *Sigmoid activation function*

Implement an ANN

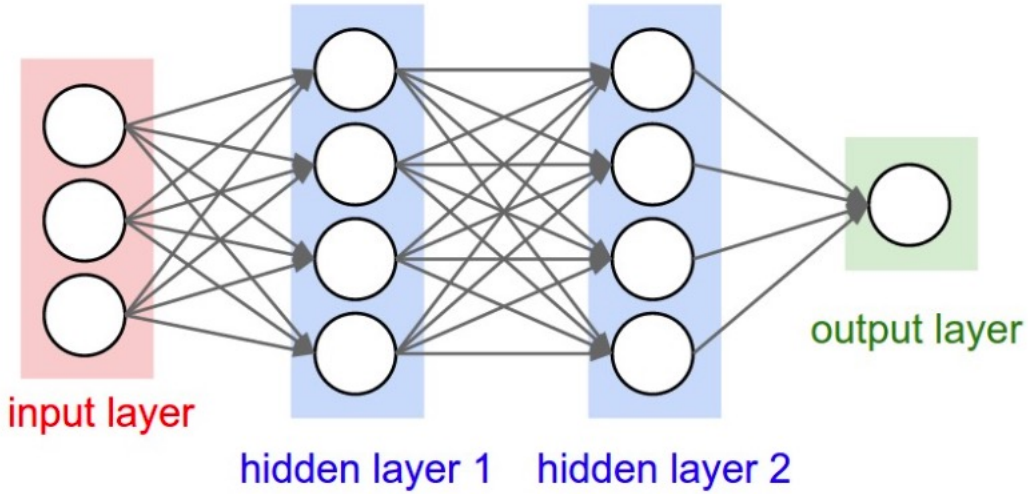
- How to **create** an ANN?
  - **Declare the hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *The structure of the ANN determines its capacity:*
  - *Be careful to ensure your ANN having enough capacity given the problem to be solved*
  - *If you ANN is under-fitted, consider to increase the capacity*
    - *Add more layers → Deeper*
    - *Add more neurons*
    - *Add more connections*

Implement an ANN

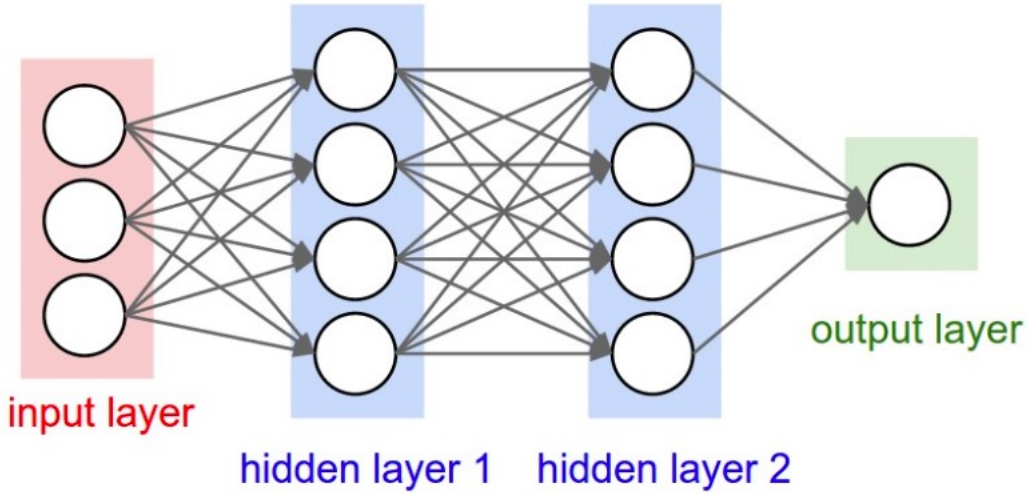
- How to **create** an ANN?
  - **Declare the hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *The structure of the ANN determines its capacity:*
  - *Be careful to ensure your ANN having enough capacity given the problem to be solved*
  - *If you ANN is over-fitted, consider to decrease the capacity*
    - *Delete some layers → Shallower*
    - *Delete (or deactivate) some neurons*
    - *Delete some connections (from fully connected to sparse connected)*

Implement an ANN

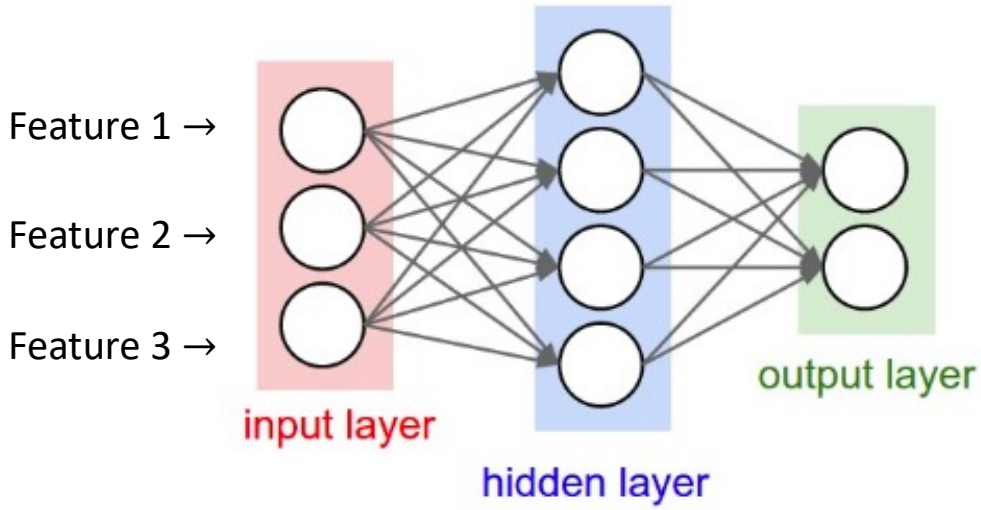
- How to **create** an ANN?
  - **Declare the hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *The structure of the ANN determines its capacity:*
  - *Be careful to ensure your ANN having enough capacity given the problem to be solved*
  - *In practice:*
    - *Always start with an over-fitted model which ensure your ANN having enough capacity*
    - *Then try to decrease the capacity by simplify the ANN*
    - *Or in most cases, applying other technique to prevent the ANN from overfitting*
      - *For example, early stopping*

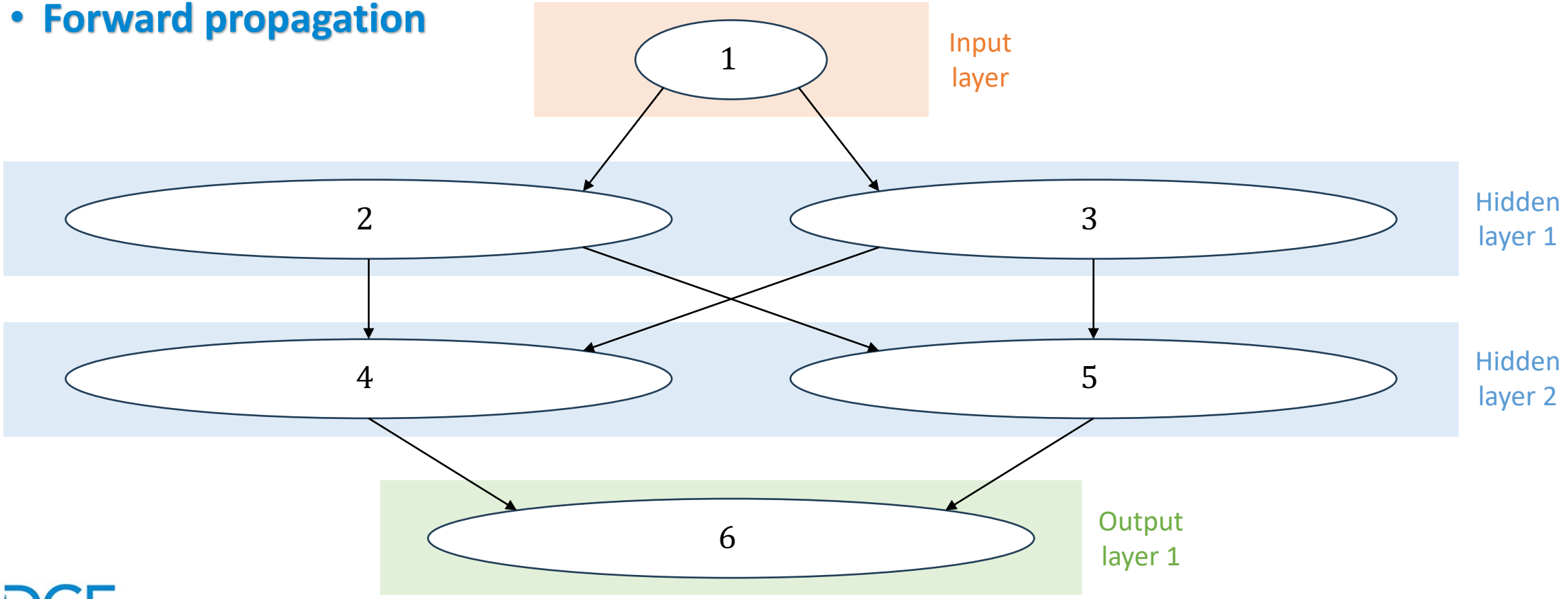
Implement an ANN

- How does ANN perform **inference**?
  - Given an input instance
    - Feed the n features to the n input nodes
      - **The number of features should equal to the number of input nodes**
    - Calculate the output of each neuron
      - Layer by layer
      - Weights, bias, activation function
    - The output of the output node is the prediction made by the ANN
  - **Forward propagation**



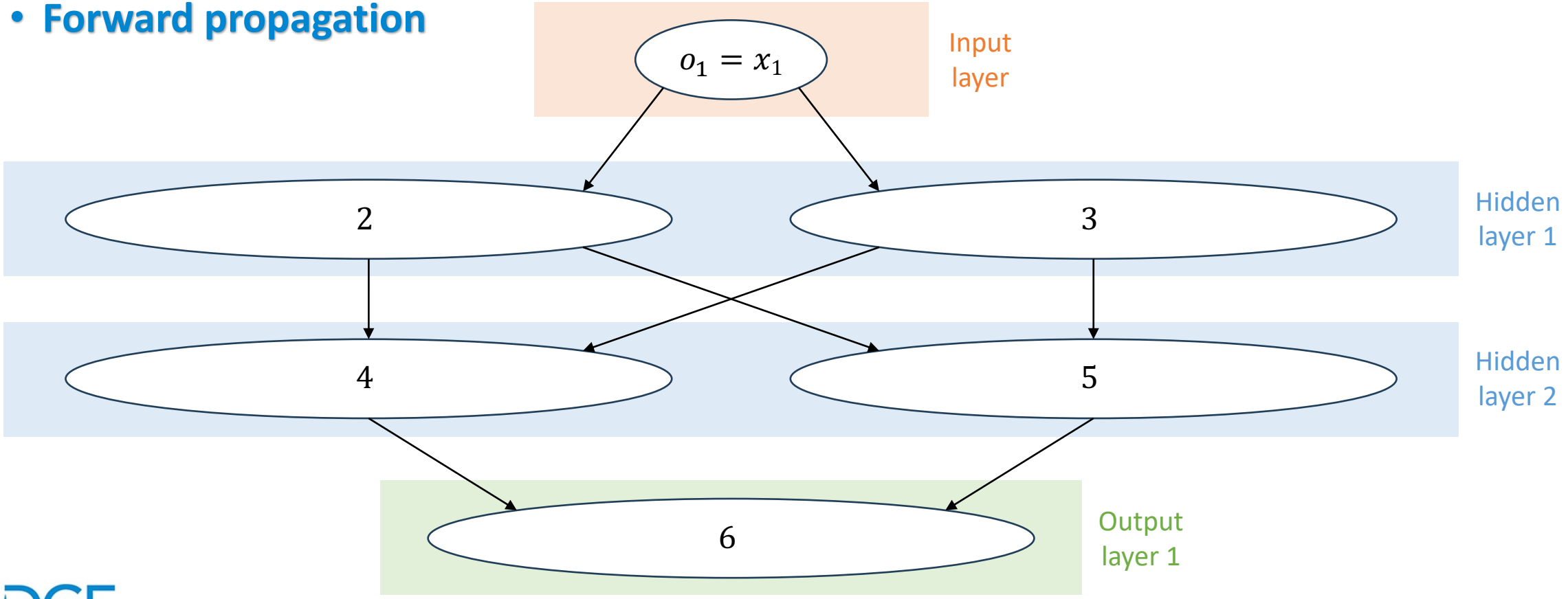
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



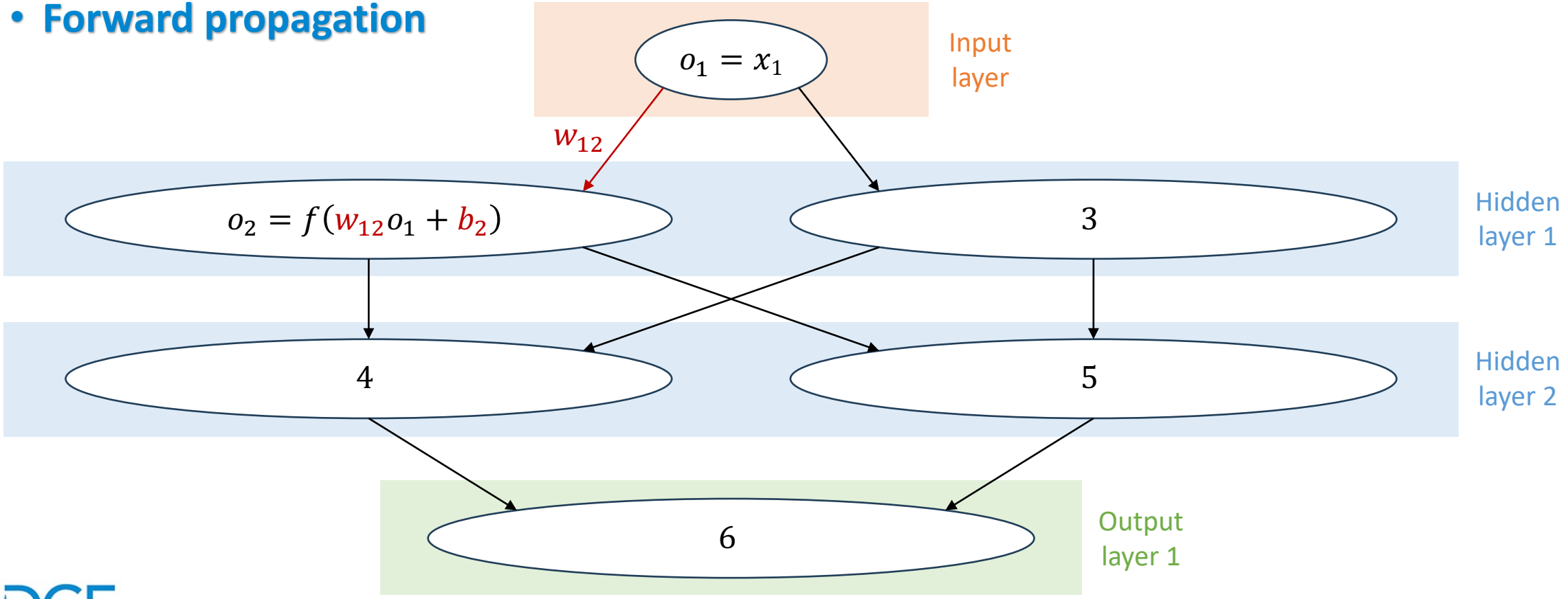
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



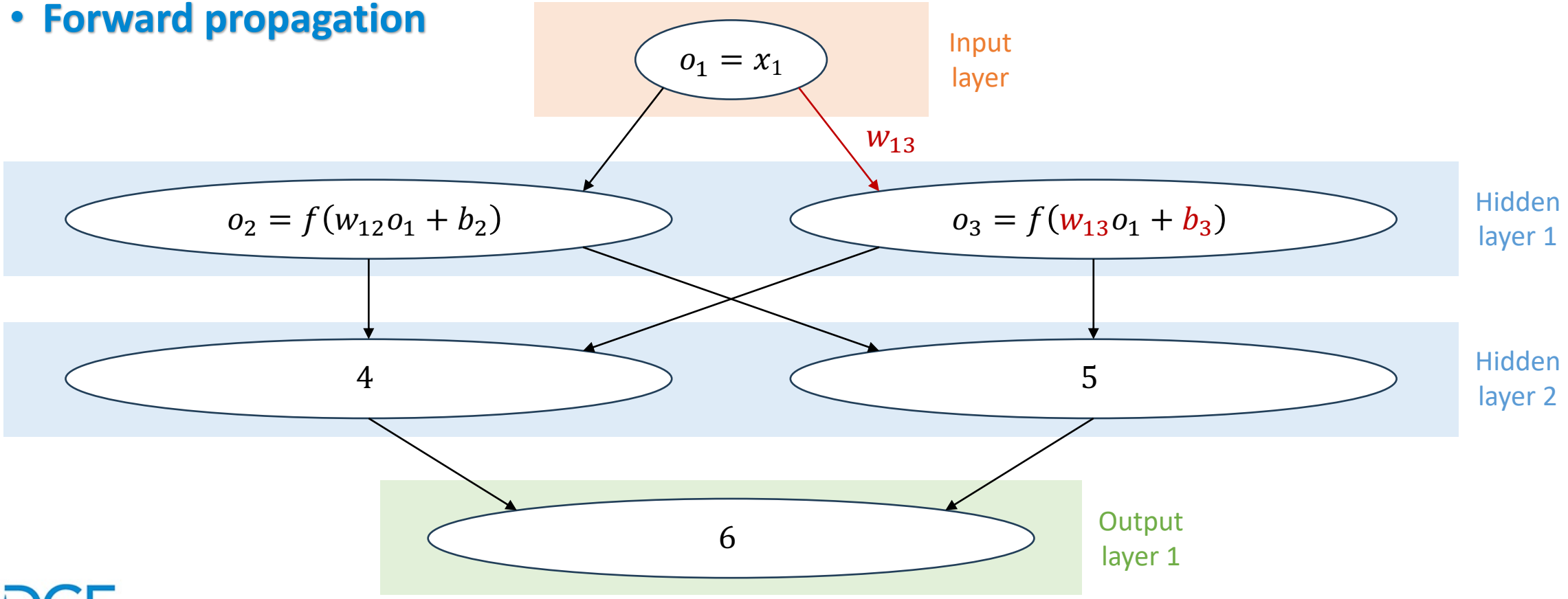
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



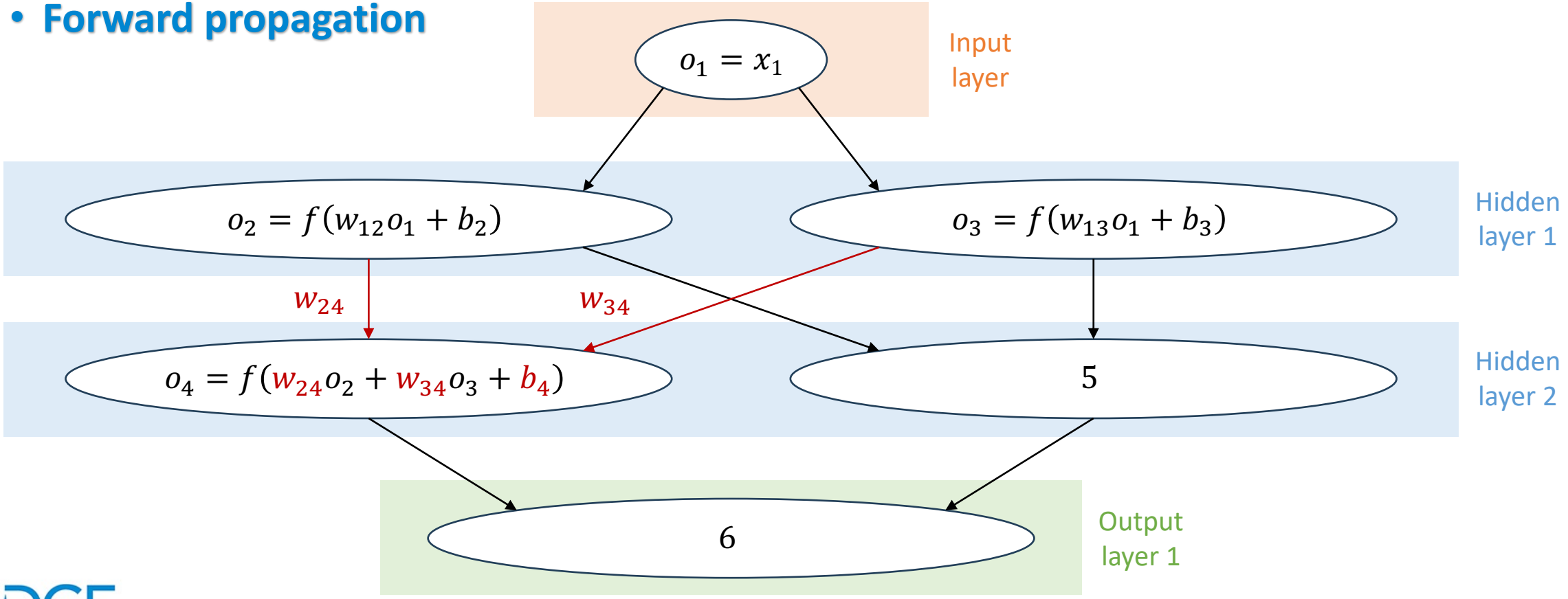
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



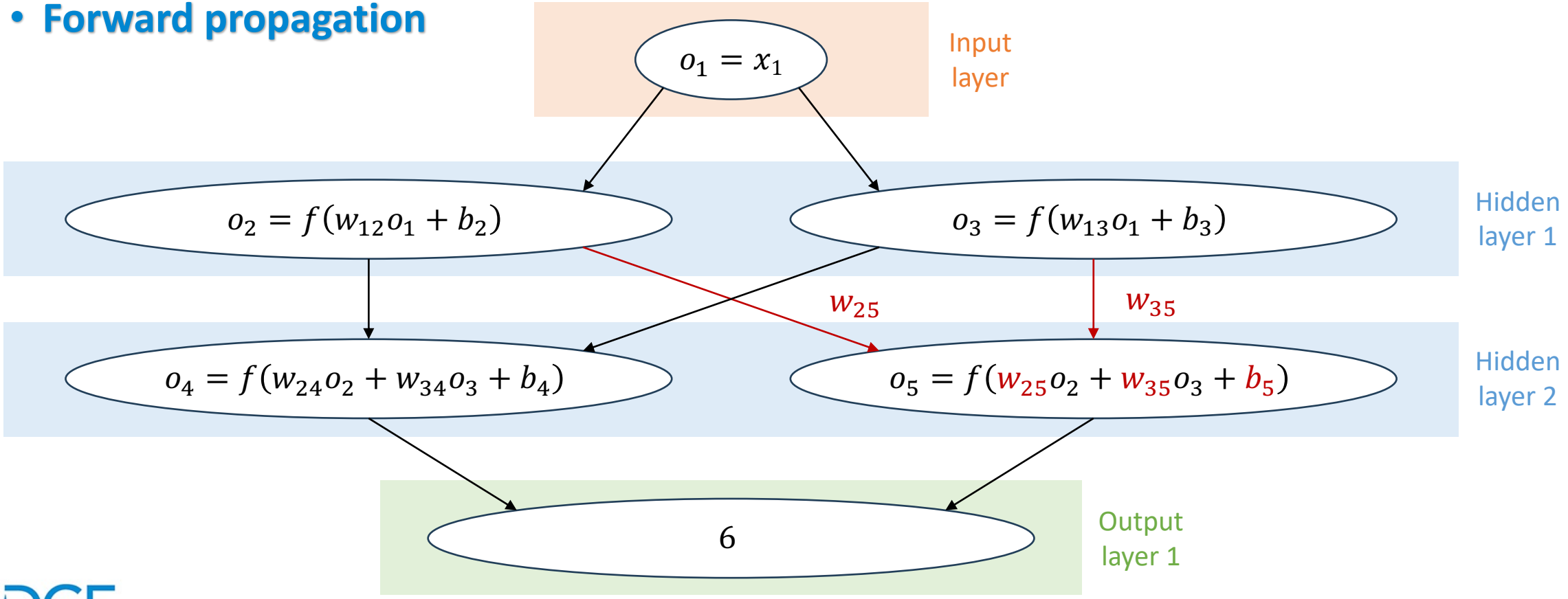
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



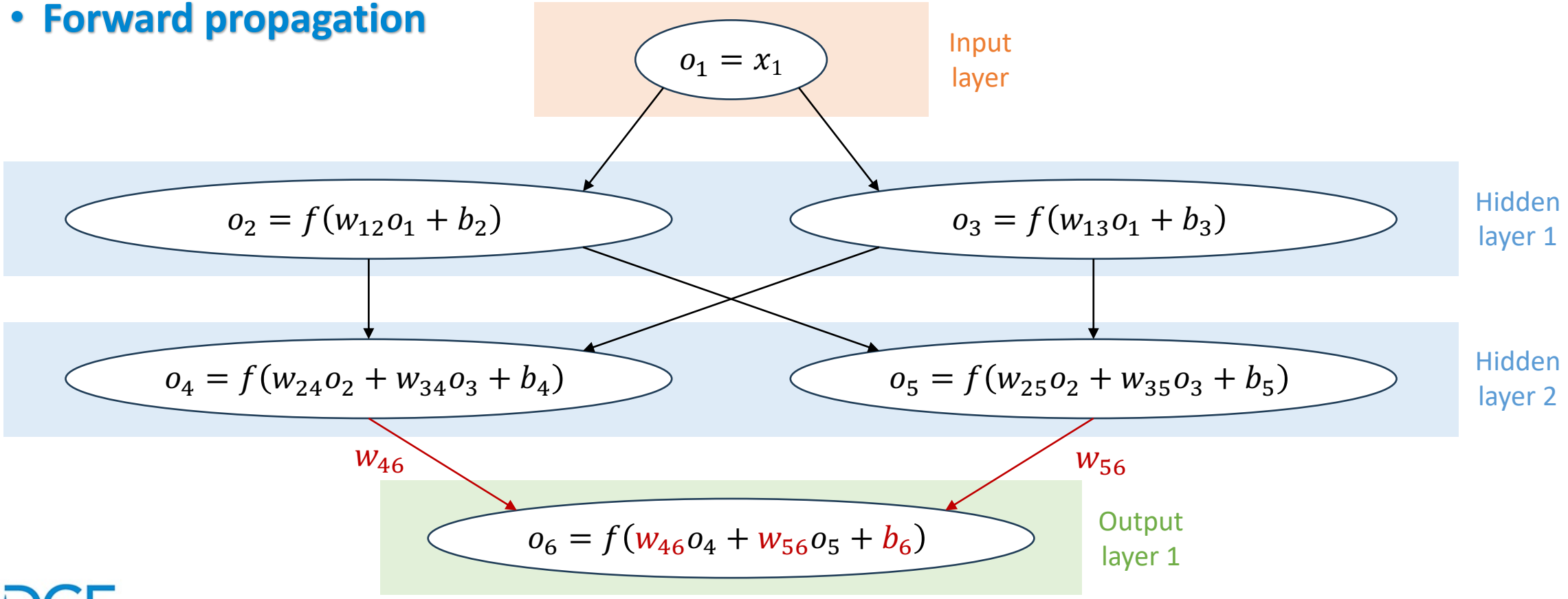
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



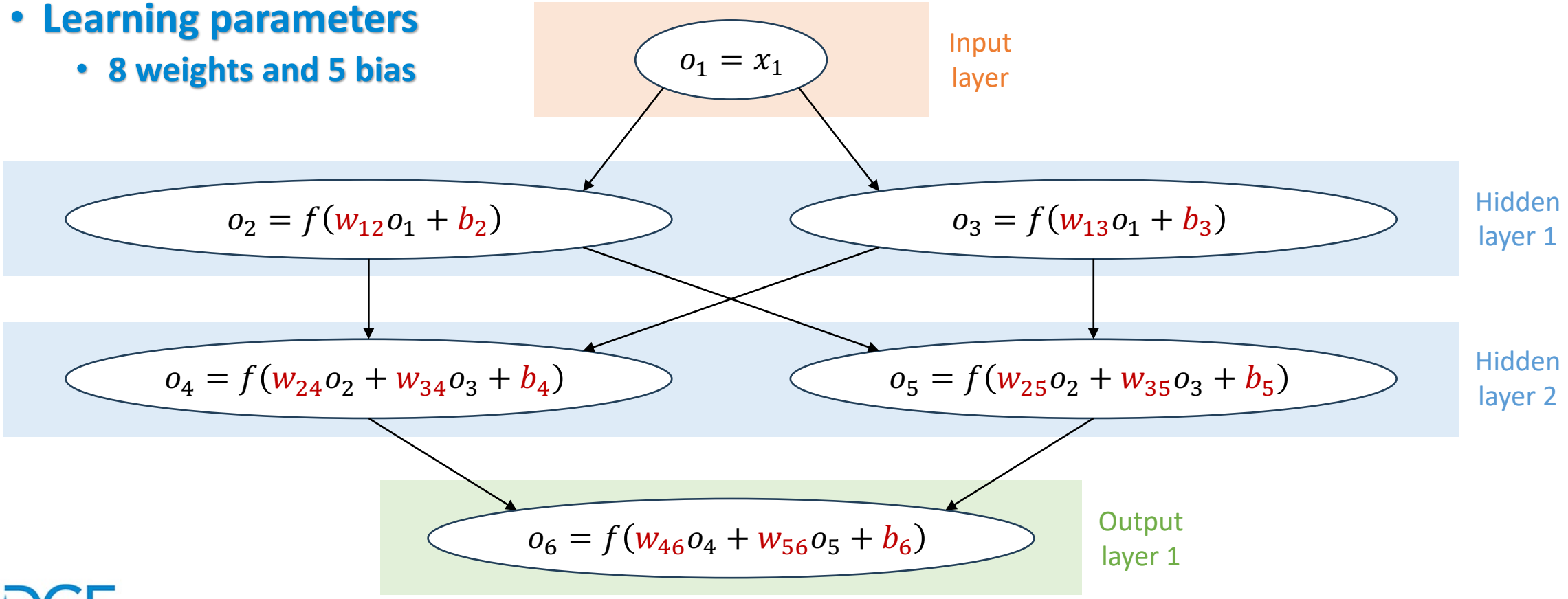
Implement an ANN

- How does ANN perform **inference**?
  - **Forward propagation**



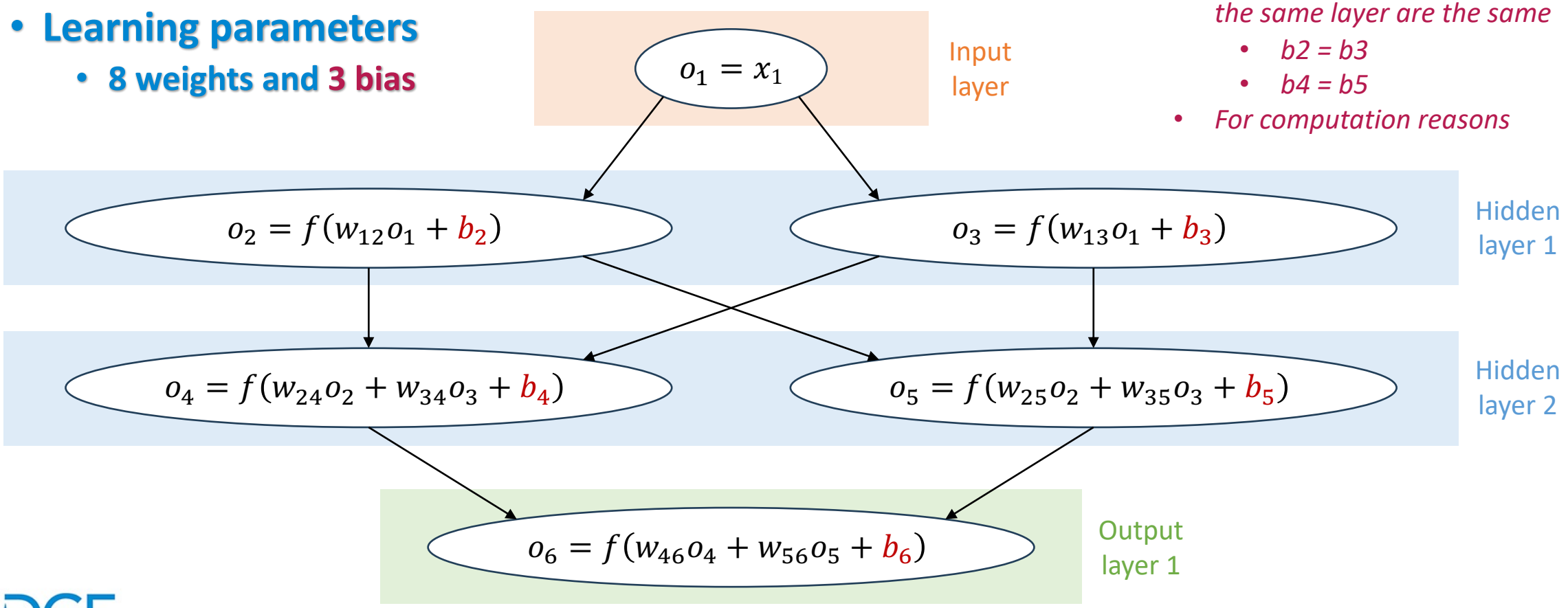
Implement an ANN

- How does ANN perform **inference**?
  - **Learning parameters**
    - **8 weights and 5 bias**



Implement an ANN

- How does ANN perform **inference**?
  - **Learning parameters**
    - **8 weights and 3 bias**

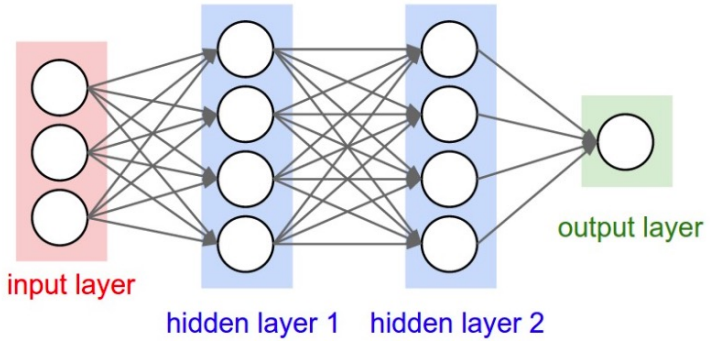


In today's practice:

- *The bias values of neurons in the same layer are the same*
  - *b2 = b3*
  - *b4 = b5*
- *For computation reasons*

Implement an ANN

- How does ANN perform **inference**?
  - Example Python scripts for a 3-layer network to perform inference

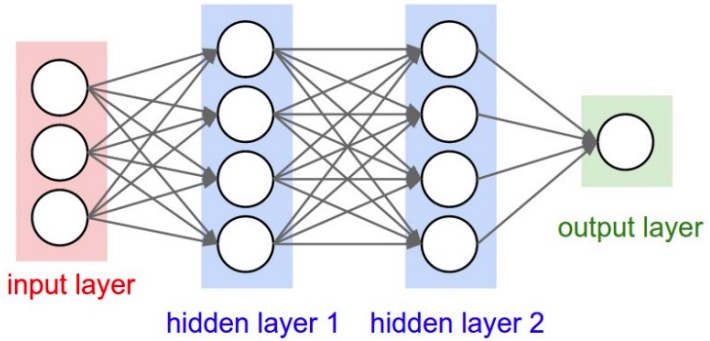


→

```
# forward propagation of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.rand(3, 1) # random input features of an instance
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer outputs
h2 = f(np.dot(W2, h1) + b2) # calculate second hidder layer outputs
out = f(np.dot(W3, h2) + b3) # output prediction
```

Implement an ANN

- How does ANN perform **inference**?
  - Example Python scripts for a 3-layer network to perform inference



→

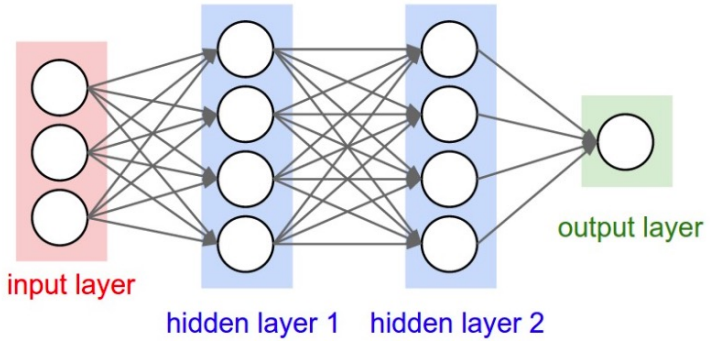
```
# forward propagation of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.rand(3, 1) # random input features of an instance
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer outputs
h2 = f(np.dot(W2, h1) + b2) # calculate second hidder layer outputs
out = f(np.dot(W3, h2) + b3) # output prediction
```

Activation function f:

- *In the example scripts, activation functions are the same for all hidden and output neurons*
- *In practice:*
  - *Activation functions of the neurons in the same layer are the same*
  - *Activation functions for different layers can be different*

Implement an ANN

- How does ANN perform **inference**?
  - Example Python scripts for a 3-layer network to perform inference



→

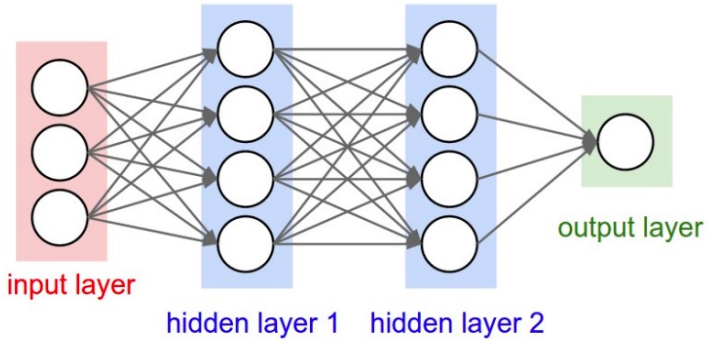
```
# forward propagation of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.rand(3, 1) # random input features of an instance
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer outputs
h2 = f(np.dot(W2, h1) + b2) # calculate second hidder layer outputs
out = f(np.dot(W3, h2) + b3) # output prediction
```

np.dot() computes the dot production of two NumPy arrays

- *The calculation of forward propagation can be implemented efficiently using matrix operation*
- *Matrix operation can be parallelized → Use GPU to accelerate*

Implement an ANN

- How does ANN perform **inference**?
  - Example Python scripts for a 3-layer network to perform inference



→

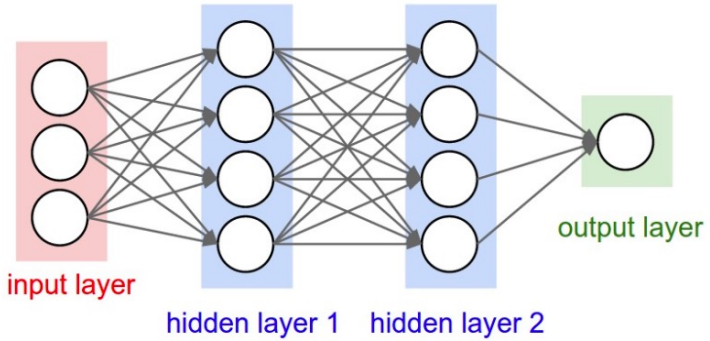
```
# forward propagation of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.rand(3, 1) # random input features of an instance
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer outputs
h2 = f(np.dot(W2, h1) + b2) # calculate second hidder layer outputs
out = f(np.dot(W3, h2) + b3) # output prediction
```

Be careful of matrix operation, the dimensions should be compatible:

shape(x) = (3, 1)
shape(h_1) = (4, 1)
shape(h_2) = (4, 1)
shape(out) = (1, 1)

Implement an ANN

- How does ANN perform **inference**?
  - Example Python scripts for a 3-layer network to perform inference



→

```
# forward propagation of a 3-layer neural network:
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.rand(3, 1) # random input features of an instance
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer outputs
h2 = f(np.dot(W2, h1) + b2) # calculate second hidder layer outputs
out = f(np.dot(W3, h2) + b3) # output prediction
```

Be careful of matrix operation, the dimensions should be compatible:

shape(W_1) = (4, 3)
shape(W_2) = (4, 4)
shape(W_3) = (4, 1)

shape(x) = (3, 1)
shape(h_1) = (4, 1)
shape(h_2) = (4, 1)
shape(out) = (1, 1)

Implement an ANN

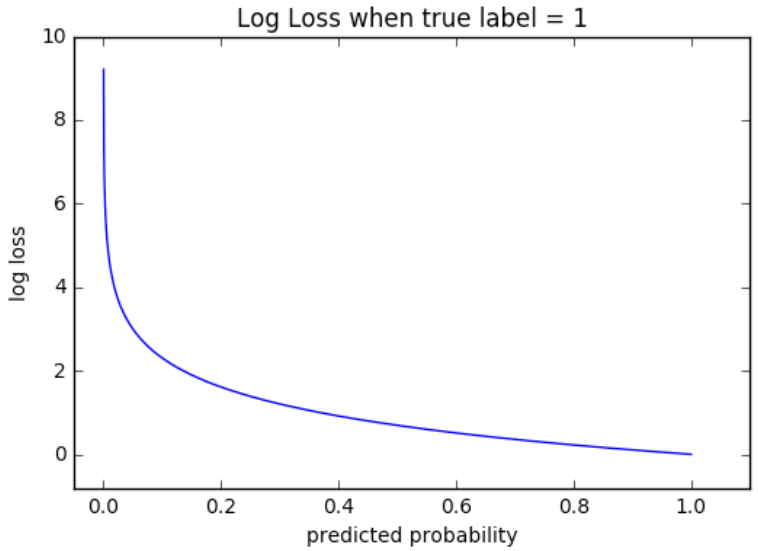
- How to **create** an ANN?
  - Number of layers, neurons and connections, activation functions
- How does ANN perform **inference**?
  - Forward propagation
- How to **train** an ANN?
  - **Learning process** → Decide the values of **learning parameters**
    - **Loss / Objective function**
      - A formal measure of how good (or bad) the models are.
    - **Optimization algorithm**
      - Search for the best values of learning parameters for minimizing the loss function

Implement an ANN

- How to **train** an ANN?
  - **Learning process** → Decide the values of **learning parameters**
    - **Loss / Objective function**
      - A formal measure of how good (or bad) the models are.
      - 'Loss' means lower value is better
      - Regression
        - **MSE, RMSE, MAE**
      - Classification
        - **Cross-entropy loss** (also called log loss)

Implement an ANN

- How to **train** an ANN?
  - **Learning process** → Decide the values of **learning parameters**
    - **Cross-entropy loss** (also called log loss)
      - Measures the performance of a classifier whose output is a probability value between 0 and 1
      - Lower is better
      - For binary classification, one instance
        - $L = -(y\log(p) + (1-y)\log(1-p))$
      - For M class, one instance
        - $L = -\sum_{c=1}^{M} y_c \log(p_c)$
      - For M class, N instance
        - $L = -\sum_{i=1}^{N} \sum_{c=1}^{M} y_c \log(p_c)$



Implement an ANN

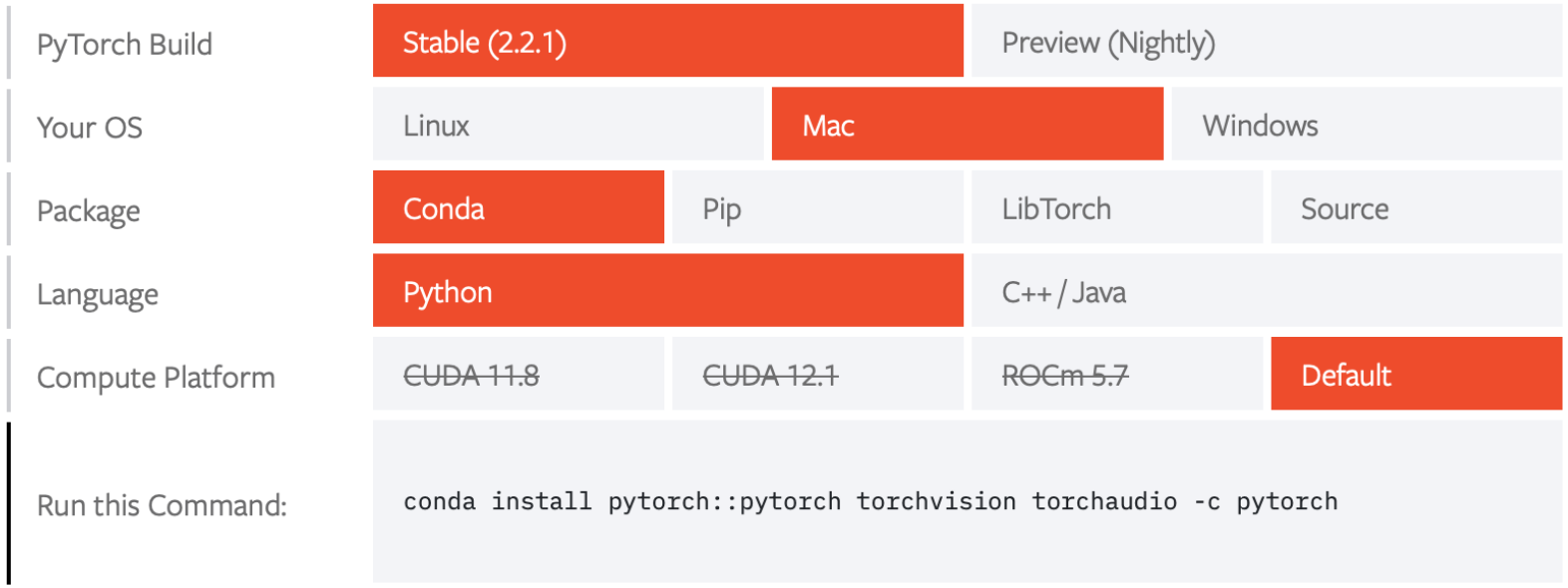
- How to **train** an ANN?
  - **Learning process** → Decide the values of **learning parameters**
    - **Loss / Objective function**
    - **Optimization algorithm**
      - Search for the best values of learning parameters for minimizing the loss function
      - **Gradient descent**

Outline

- Classical Machine Learning vs. Deep Learning
- Artificial Neural Network
- Implement an ANN
- **Preliminaries with PyTorch**

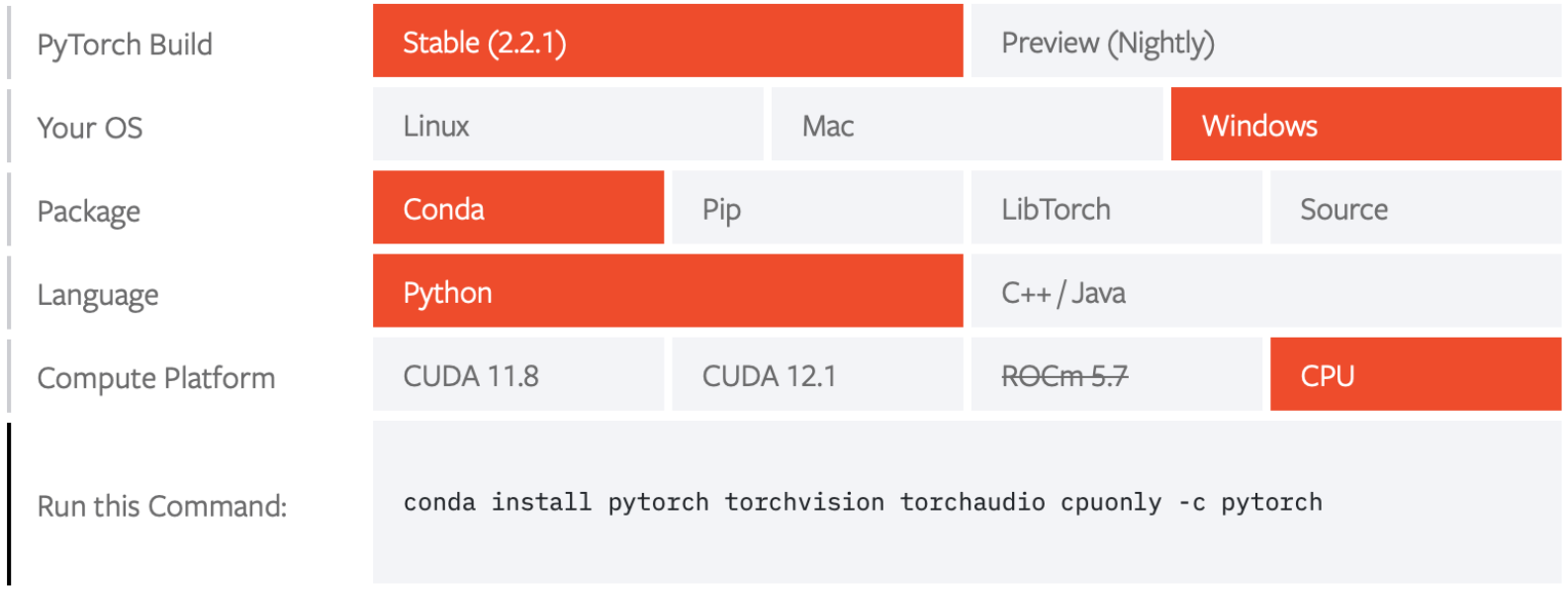
Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - For Mac



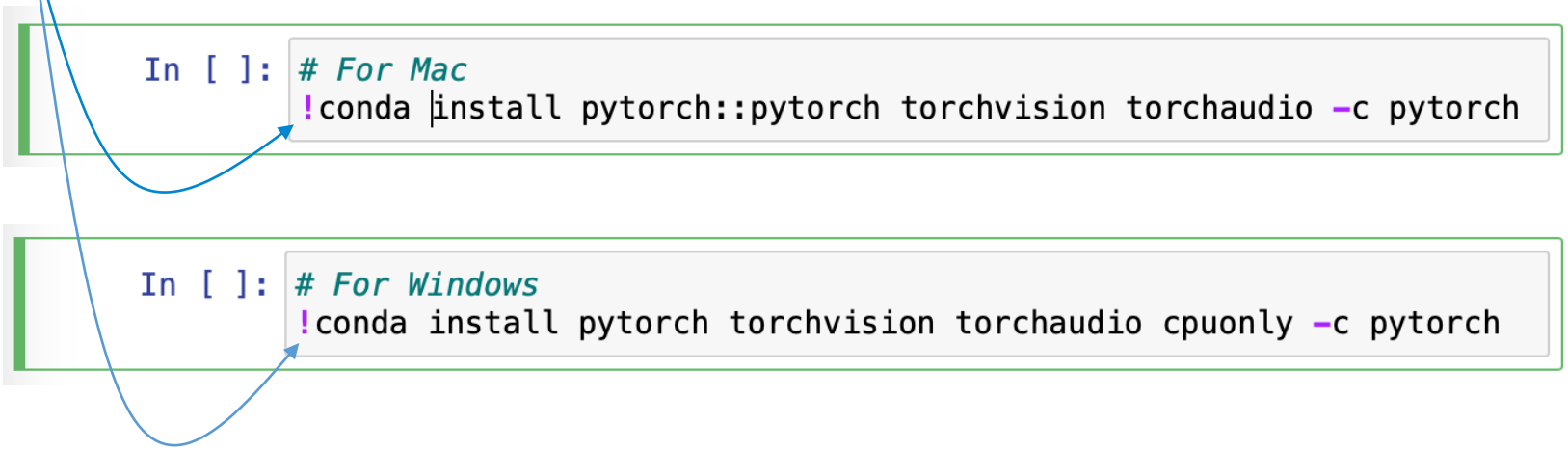
Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - For Windows



Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - Execute the corresponding command in a Jupyter Notebook Cell
    - Add ! in front of the command



Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - Execute the corresponding command in a Jupyter Notebook Cell
    - Add ! in front of the command

```
In [*]: # For Mac
        !conda install pytorch::pytorch torchvision torchaudio -c pytorch
```

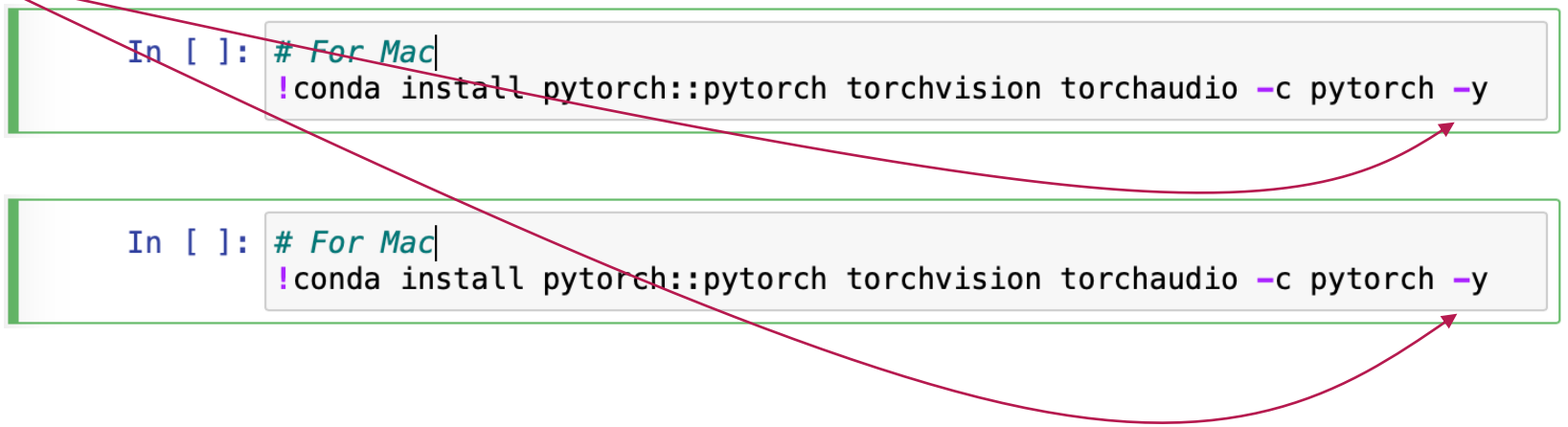
```
  mpc                pkgs/main/osx-arm64::mpc-1.1.0-h8c48613_1
  mpfr               pkgs/main/osx-arm64::mpfr-4.0.2-h695f6f0_1
  mpmath             pkgs/main/osx-arm64::mpmath-1.3.0-py311hca03da5_0
  networkx           pkgs/main/osx-arm64::networkx-3.1-py311hca03da5_0
  numpy              pkgs/main/osx-arm64::numpy-1.26.4-py311he598dae_0
  numpy-base         pkgs/main/osx-arm64::numpy-base-1.26.4-py311hfbfe69c_0
  openjpeg           pkgs/main/osx-arm64::openjpeg-2.3.0-h7a6adac_2
  pillow             pkgs/main/osx-arm64::pillow-10.2.0-py311h80987f9_0
  pysocks            pkgs/main/osx-arm64::pysocks-1.7.1-py311hca03da5_0
  pytorch            pytorch/osx-arm64::pytorch-2.2.1-py3.11_0
  requests           pkgs/main/osx-arm64::requests-2.31.0-py311hca03da5_1
  sympy              pkgs/main/osx-arm64::sympy-1.12-py311hca03da5_0
  torchaudio         pytorch/osx-arm64::torchaudio-2.2.1-py311_cpu
  torchvision        pytorch/osx-arm64::torchvision-0.17.1-py311_cpu
  urllib3            pkgs/main/osx-arm64::urllib3-2.1.0-py311hca03da5_1
  zstd               pkgs/main/osx-arm64::zstd-1.5.5-hd90d995_0


Proceed ([y]/n)?
```

There is a double check to run this command

Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - Execute the corresponding command in a Jupyter Notebook Cell
    - Add ! in front of the command
    - Add -y at the end of the command to skip the double check



Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - Execute the corresponding command in a Jupyter Notebook Cell
    - Add ! in front of the command
    - Add -y at the end of the command to skip the double check

You only need to install once.

```
In [1]: # For Mac
        !conda install pytorch::pytorch torchvision torchaudio -c pytorch -y

        pytorch-2.2.1        | 55.5 MB   | ###3                                |   9%

        torchaudio-2.2.1     | 5.0 MB    | ##############################7     |  86%
        pytorch-2.2.1        | 55.5 MB   | ###7                                |  10%

        torchaudio-2.2.1     | 5.0 MB    | ################################### |  95%
        pytorch-2.2.1        | 55.5 MB   | ####                                |  11%
        pytorch-2.2.1        | 55.5 MB   | ####6                               |  13%
        pytorch-2.2.1        | 55.5 MB   | #####                               |  14%

        pytorch-2.2.1        | 55.5 MB   | #######7                            |  21%
        torchvision-0.17.1   | 6.8 MB    | ################################### | 100%



        Preparing transaction: done
        Verifying transaction: done
        Executing transaction: done
```

These 3 lines indicate you successfully install PyTorch

Preliminaries with PyTorch

- Install PyTorch use Conda
  - https://pytorch.org/get-started/locally/
  - Execute the corresponding command in a Jupyter Notebook Cell
    - Add ! in front of the command
    - Add -y at the end of the command to skip the double check
  - Verify whether PyTorch is installed correctly

```
In [2]: import torch
        x = torch.rand(5, 3)
        print(x)

        tensor([[0.6496, 0.9073, 0.0164],
                [0.2405, 0.4017, 0.6177],
                [0.1217, 0.6217, 0.0389],
                [0.4661, 0.9980, 0.1216],
                [0.9674, 0.0551, 0.5011]])
```

Reading Assignment

- Reading assignments
  - https://d2l.ai/
  - Chapter 2. Preliminaries
    - 2.1 Data Manipulation
    - 2.3 Linear Algebra
    - 2.4 Calculus
    - 2.5 Automatic Differentiation
  - Read the content and execute the code in the Jupyter Notebook