ICT303 – Advanced Machine Learning and Artificial Intelligence

Topic 3: Perceptron and Multilayer Perceptron (MLP)

Hamid Laga
H.Laga@murdoch.edu.au
Office: 245.1.020

How to Get in Touch with the Teaching Team

- Internal and External Students
  - Email: H.Laga@murdoch.edu.au.
- Important
  - In any communication, please make sure that you
    - Start the subject of your email with ICT303
    - Include your student ID, name, and the lab slot in which you are enrolled.
  - We will do all our best to answer your queries within 24 hrs.

In this Lecture

- From regression to classification
  - Concept
  - Activation functions
- Multilayer Perceptron (MLP)
  - Definition
  - Activation functions (again)
  - Hidden layers
- Training MLPs
  - Forward and backward propagation
- Summary

- Learning objectives
  - Understand and describe difference between classification and regression
  - Understand the meaning and importance of activation function
  - Understand multi-layer perceptrons and describe their advantage and limitations
  - Implement linear regression, classification and MLP using Python, NumPy and PyTorch
  - Train and apply MLP to practical problems
- Additional readings
  - Chapters 4 and 5 of the textbook, available at: https://d2l.ai/

Classification Problem

- Let's start with a simple problem
  - We would like to develop a machine learning model that takes as input a greyscale image and returns whether it is of a "cat" or "not a cat"

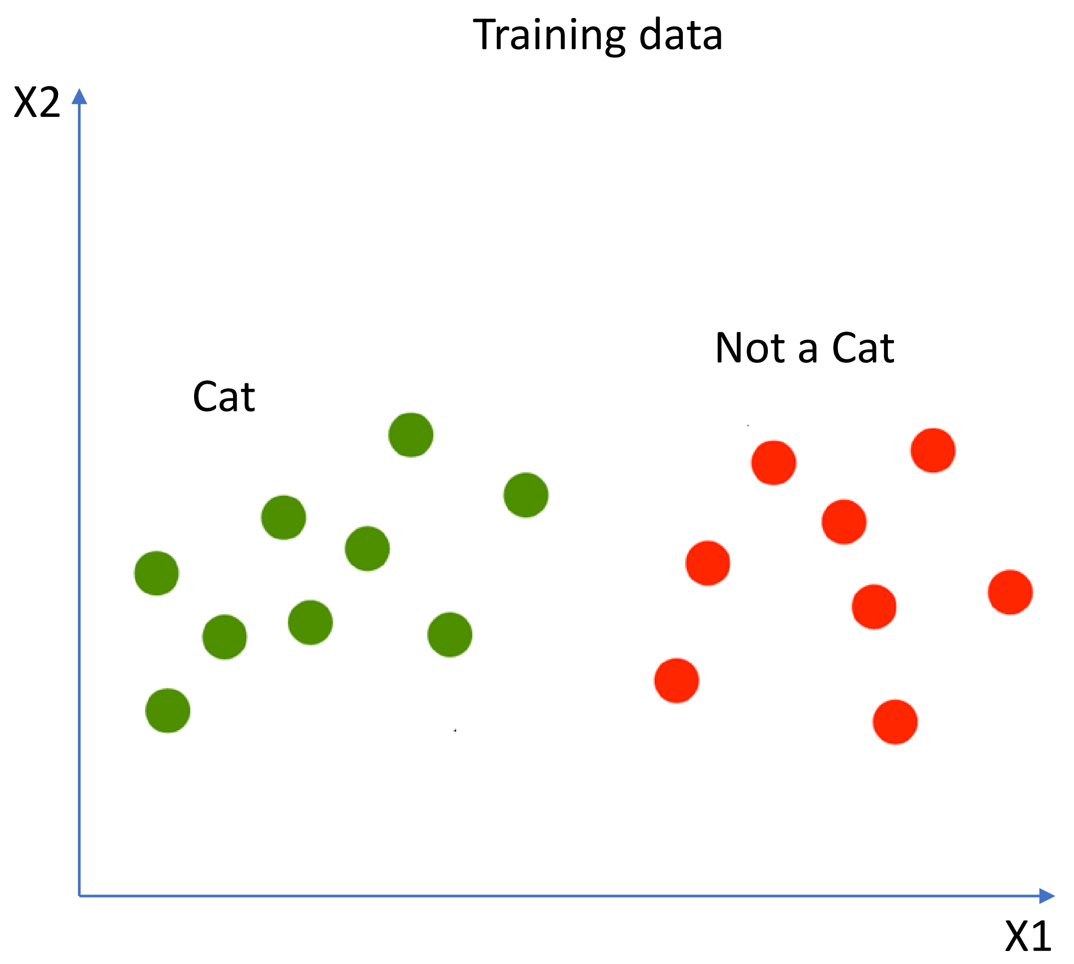
- This is a binary classification problem
  - Input to the model
    - A greyscale image
  - Output
    - A binary variable with value 1 to indicate a "cat" and 0 to indicate "not a cat"

Classification Problem

- What is classification
  - Given an input (e.g., an image), classify it into one of n categories (e.g., cat, horse, dog, …)
  - Example
    - Object classification from images
    - Face recognition
    - Suspicious behaviour detection
    - …
- Our objective
  - Extend the linear neuron we saw last week to classification problems

Classification Problem

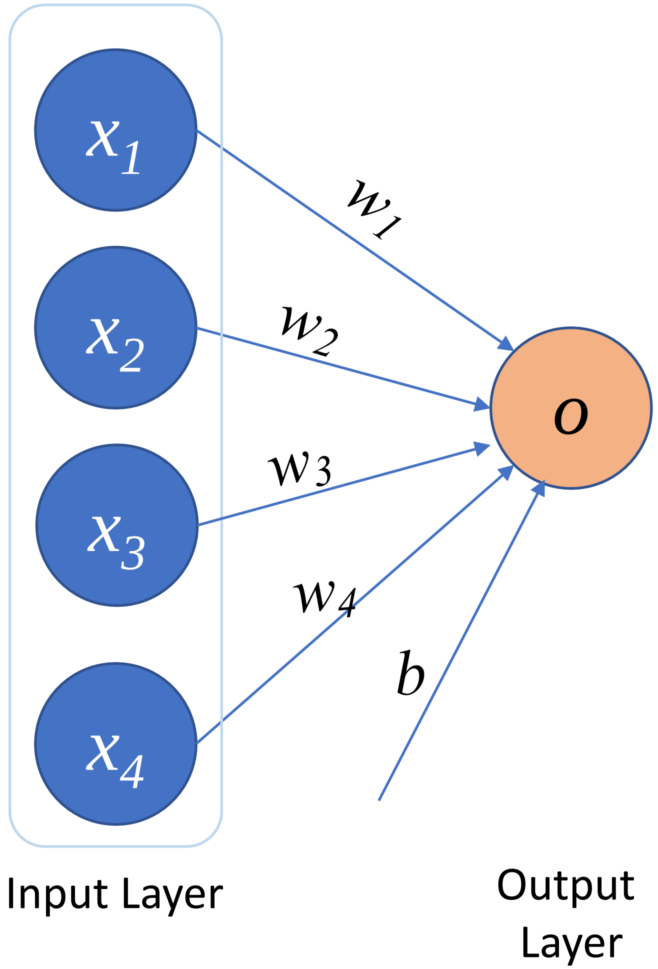
- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value bet x_1, x_2, x_3, x_4
  - Thus, the input is a vector of 4 values
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"



Classification Problem

- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"
- Let's use a linear model
  - Similar to linear regression:

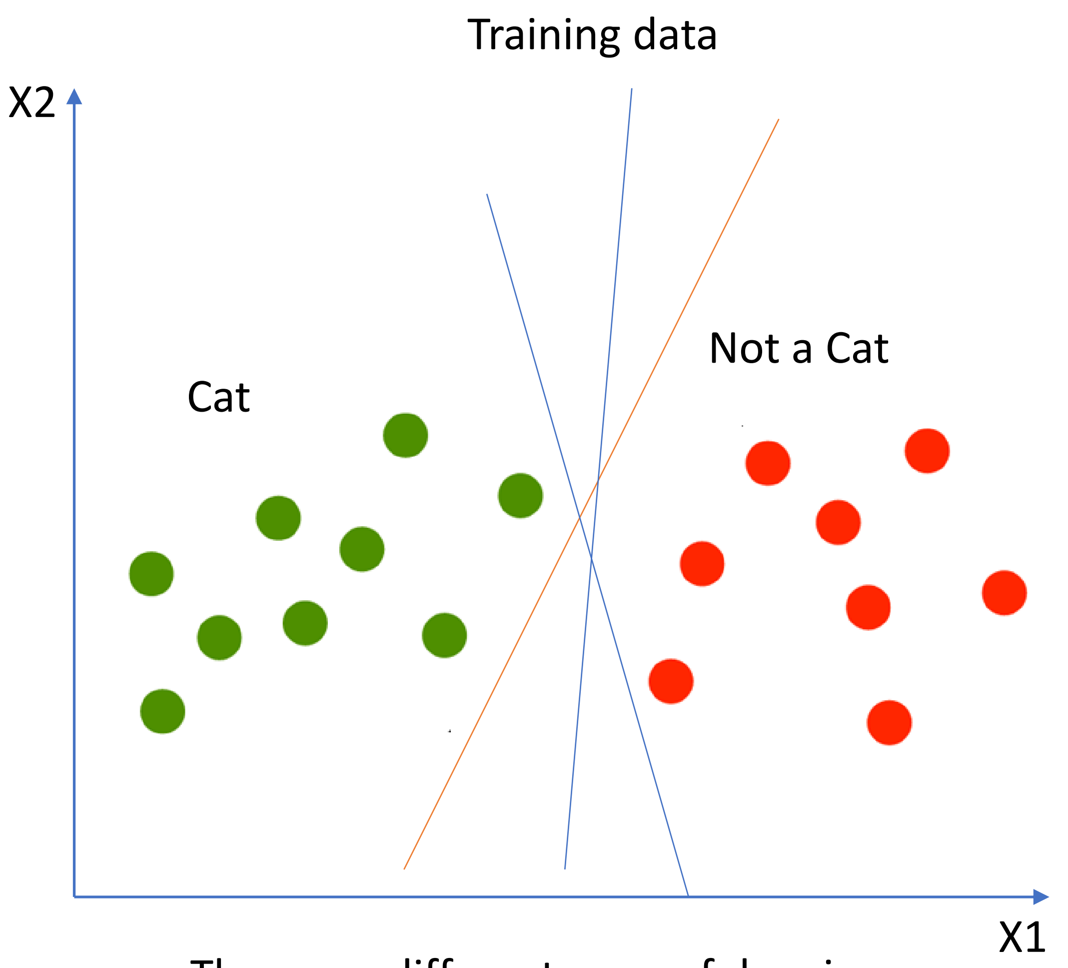
$$O = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$O = \langle \mathbf{W}, \mathbf{x} \rangle + b$$



Classification Problem

- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"
- Let's use a linear model
  - Similar to linear regression:

$$O = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$O = \langle \mathbf{W}, \mathbf{x} \rangle + b$$

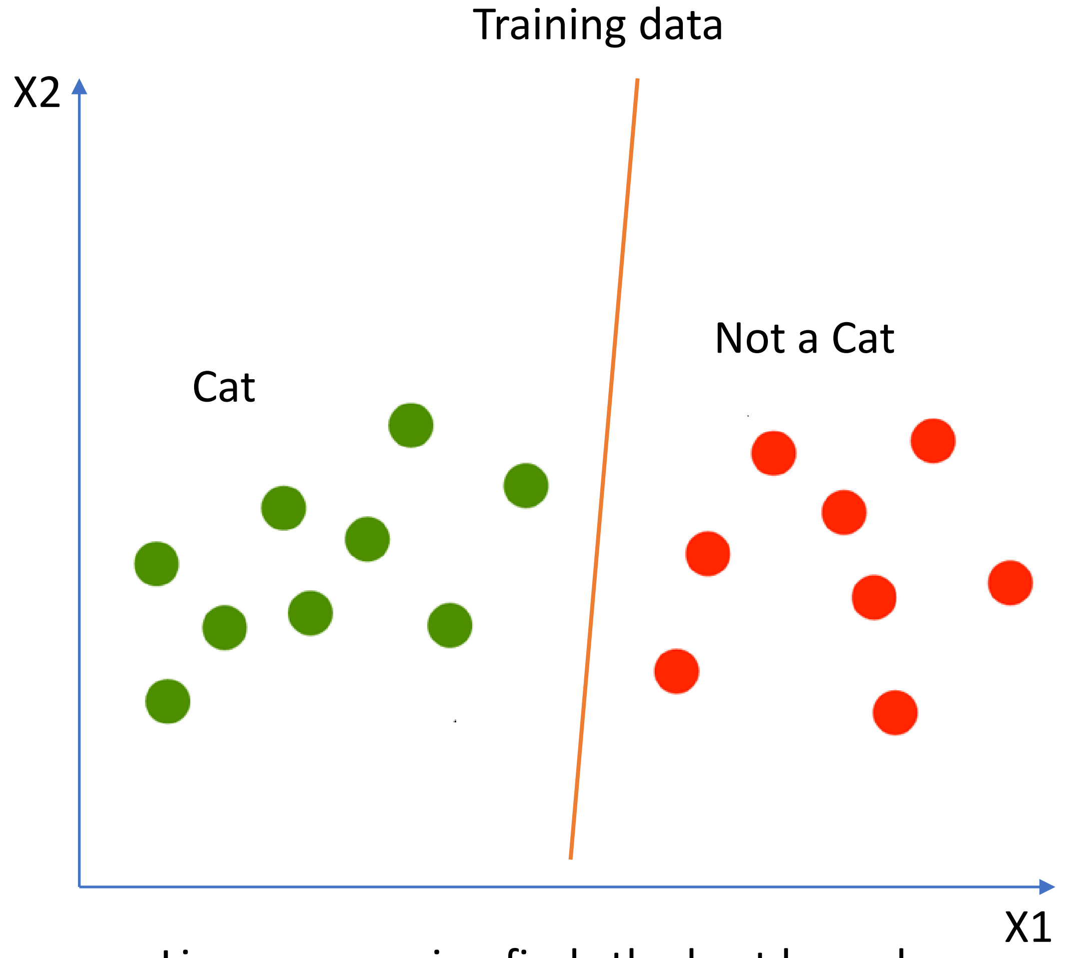


There are different ways of drawing a boundary between the two classes

Classification Problem

- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"
- Let's use a linear model
  - Similar to linear regression:

$$O = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$O = \langle \mathbf{W}, \mathbf{x} \rangle + b$$



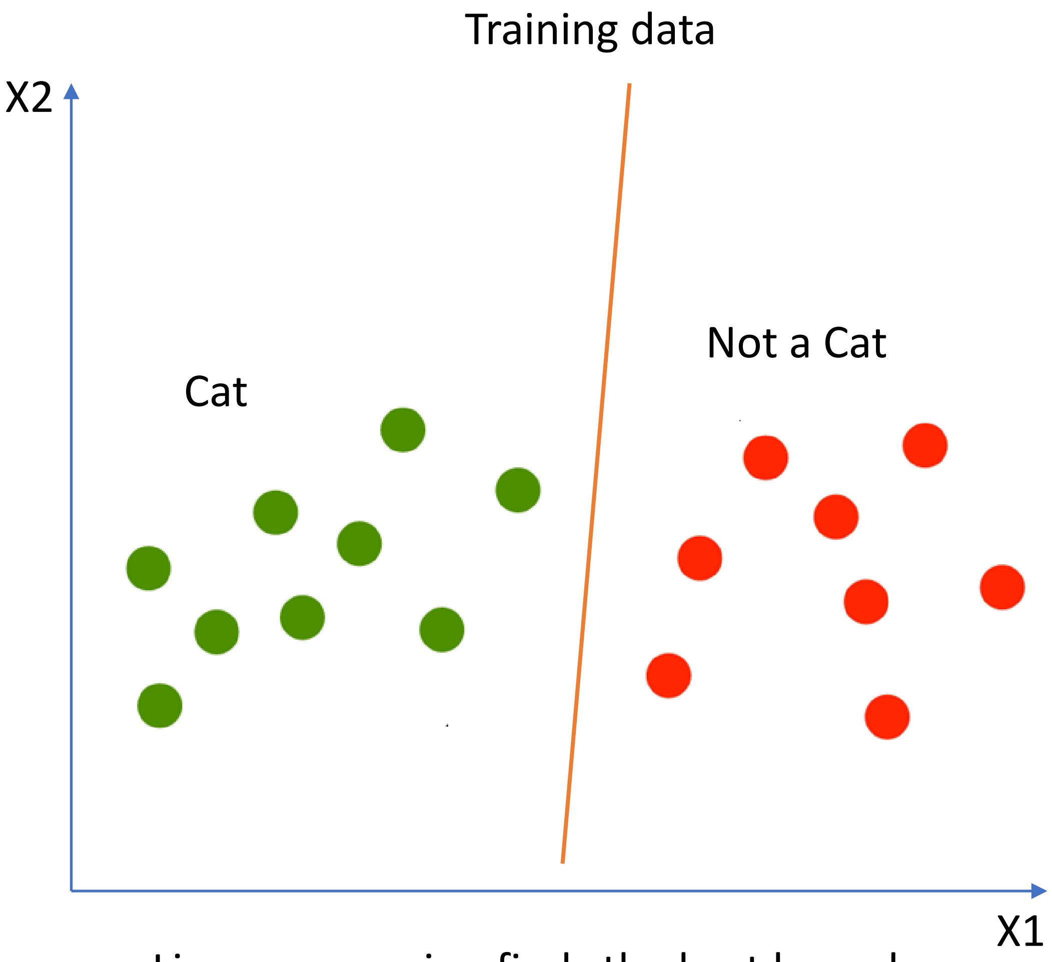
Linear regression finds the best boundary that separates the two classes

Classification Problem

- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"
- Let's use a linear model
  - Similar to linear regression:

$$O = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$O = \langle \mathbf{W}, \mathbf{x} \rangle + b$$

  - Linear regression finds the best boundary that separates the classes
  - It does not make a decision



Linear regression finds the best boundary that separates the two classes

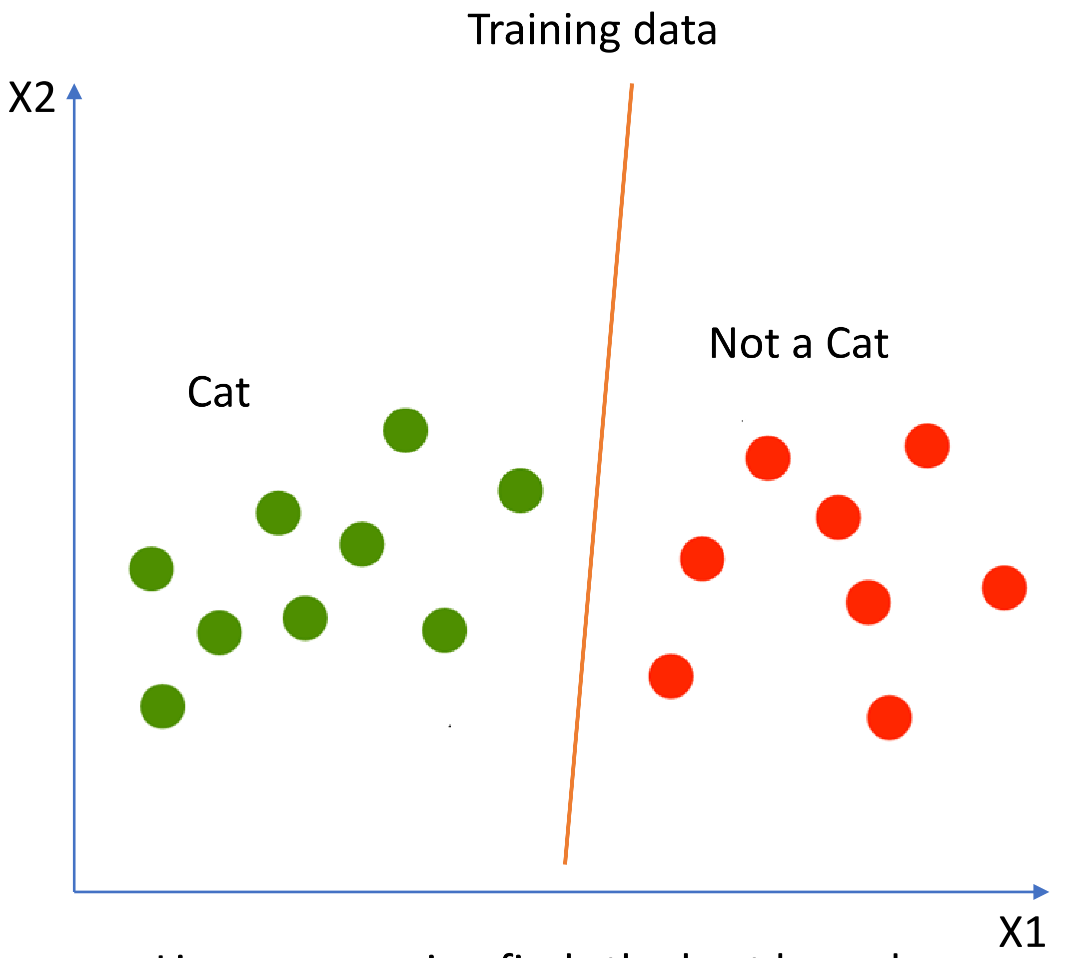
Classification Problem

- To start, let's assume that the images are of size 2 x 2
- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"
- Let's use a linear model
  - Similar to linear regression:

$$O = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$O = <\mathbf{W}, \mathbf{x}> + b$$

  - Decision:
    - transform the output values so that they are between 0 and 1 using an activation function σ

$$y = \sigma(O)$$



Linear regression finds the best boundary that separates the two classes

Classification Problem

- To start, let's assume that the images are of size 2 x 2

- Input - 2 x 2 grayscale image
  - Each pixel is a grayscale value between 0 and 1
  - Thus, the input is a vector of 4 values x_1, x_2, x_3, x_4

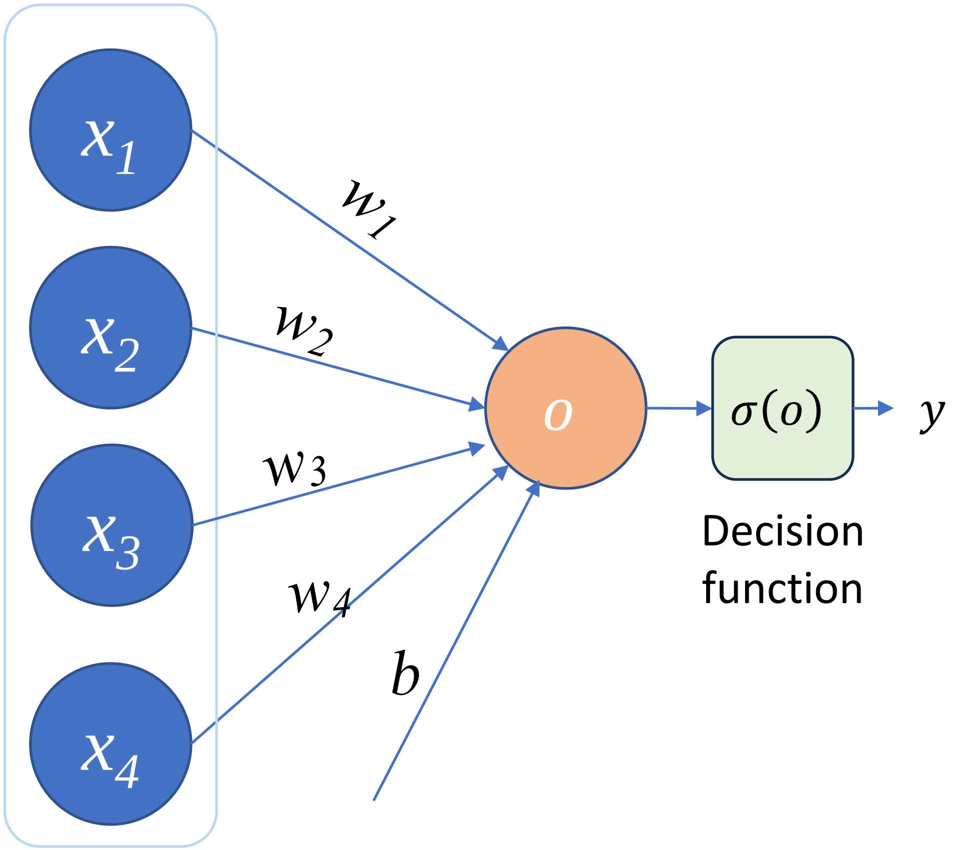
- Output - A binary variable with
  - value 1 to indicate that the image is of a "cat" and
  - Value 0 to indicate that it is "not a cat"

- Let's use a linear model
  - Similar to linear regression:

$$o = w_1x_1 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$
$$o = <\mathbf{W}, \mathbf{x}> + b$$

  - Decision:
    - transform the output values so that they are between 0 and 1 using an activation function σ

$$y = \sigma(O)$$



Example of decision, called also activation, functions

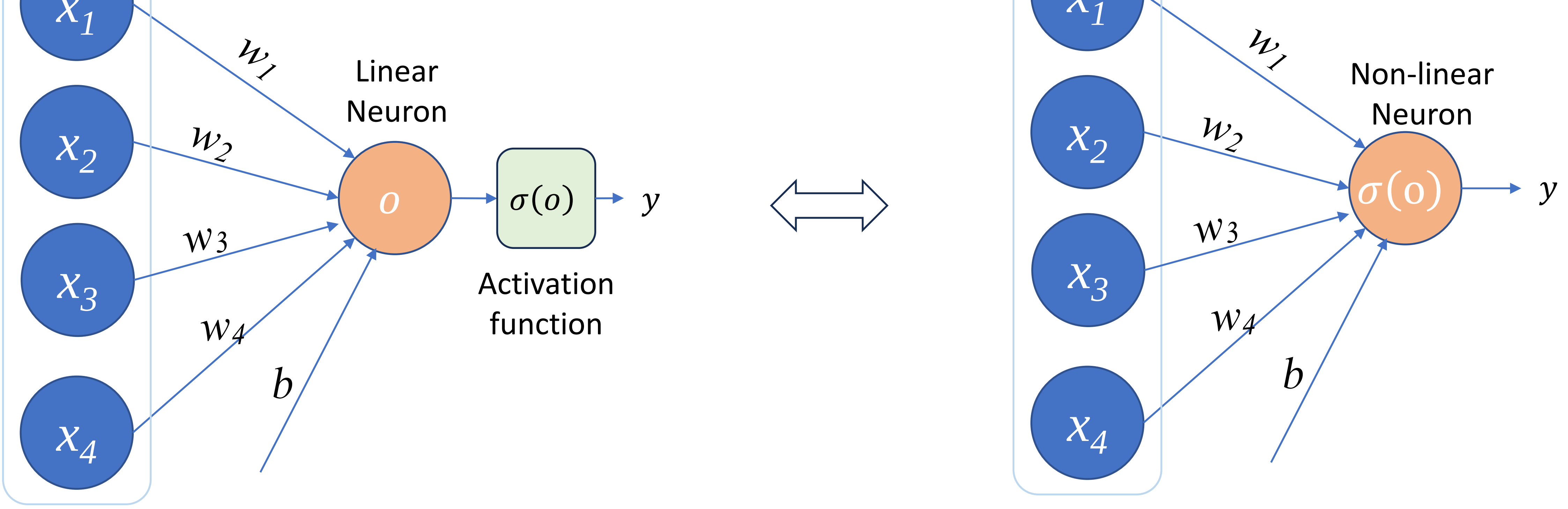
$$\sigma(\text{output}) = \begin{cases} 1 \text{ if output} > \epsilon \\ 0 \text{ otherwise} \end{cases}$$

ϵ is a threshold, e.g., 0.5

Perceptron

- Perceptron is just a linear neuron followed by an activation function
  - It introduces non-linearity – thus sometimes called non-linear neuron

Linear neuron followed by activation function forms a non-linear neuron or perceptron



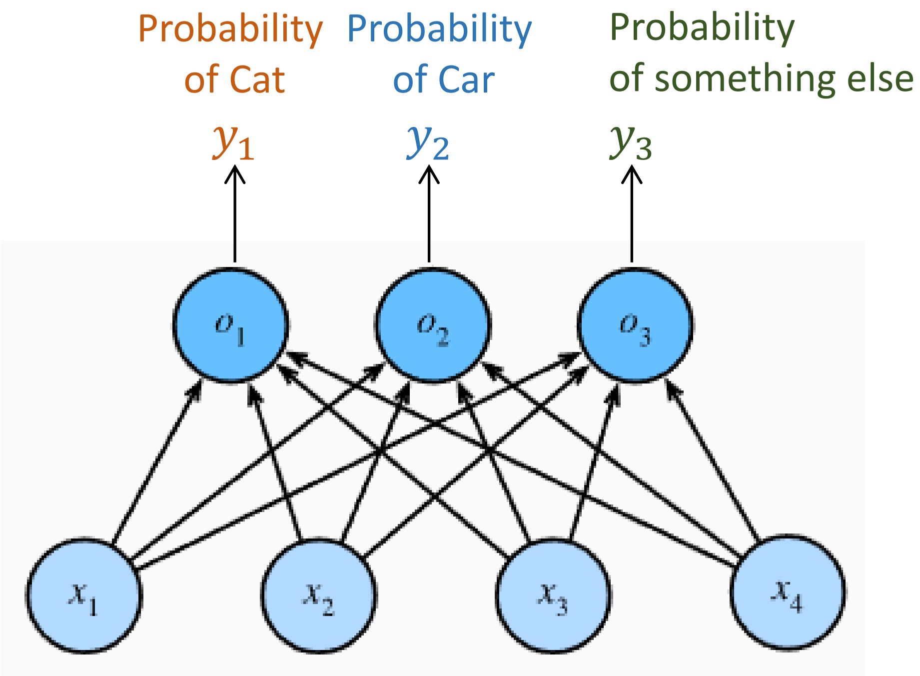
Extension to multiclass classification problem

- We would like to develop a model that takes a greyscale image and returns whether it is of a "cat", a "car", or "none of these"

https://d2l.ai/chapter_linear-classification/softmax-regression.html

Extension to multiclass classification problem

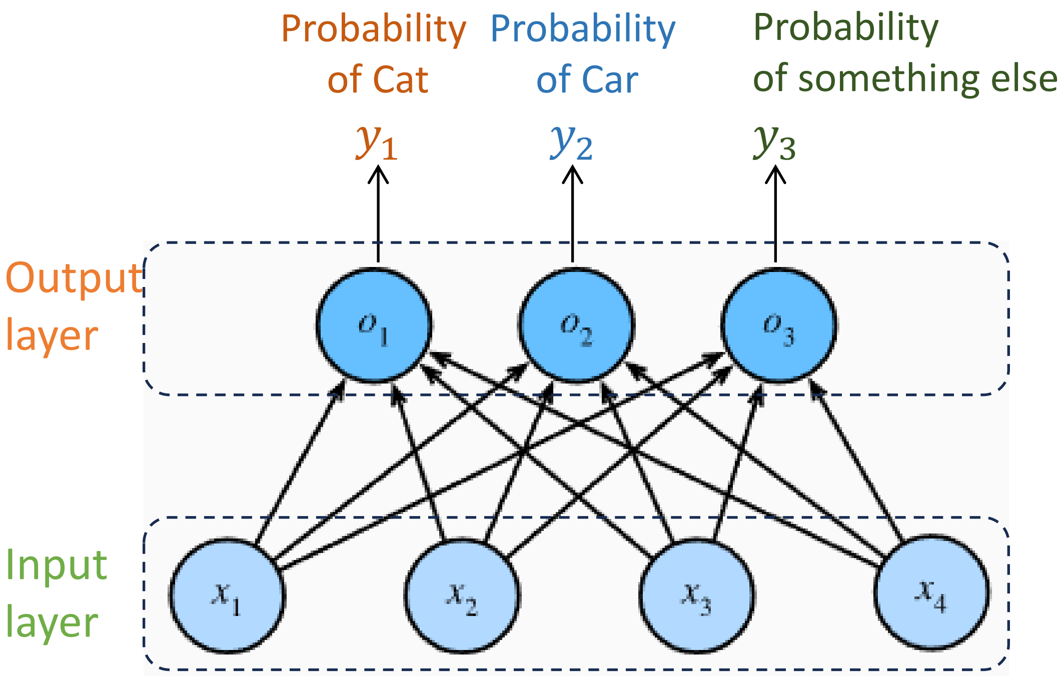
- We would like to develop a model that takes a greyscale image and returns whether it is of a "cat", a "car", or "none of these"
  - Input
    - A greyscale image of size 2 x 2
  - Output:
    - The probability of being a cat → y1
    - The probability of being a car → y2
    - The probability of being none of these → y3
  - Network architecture – two layers
    - Input layer has 4 neurons – one for each of the input values
    - The output layer has 3 neurons – one for each of the categories we would like to recognize



https://d2l.ai/chapter_linear-classification/softmax-regression.html

Extension to multiclass classification problem

- We would like to develop a model that takes a greyscale image and returns whether it is of a "cat", a "car", or "none of these"
  - Input
    - A greyscale image of size 2 x 2
  - Output:
    - The probability of being a cat → y1
    - The probability of being a car → y2
    - The probability of being none of these → y3
  - Network architecture – two layers
    - Input layer has 4 neurons – one for each of the input values
    - The output layer has 3 neurons – one for each of the categories we would like to recognize

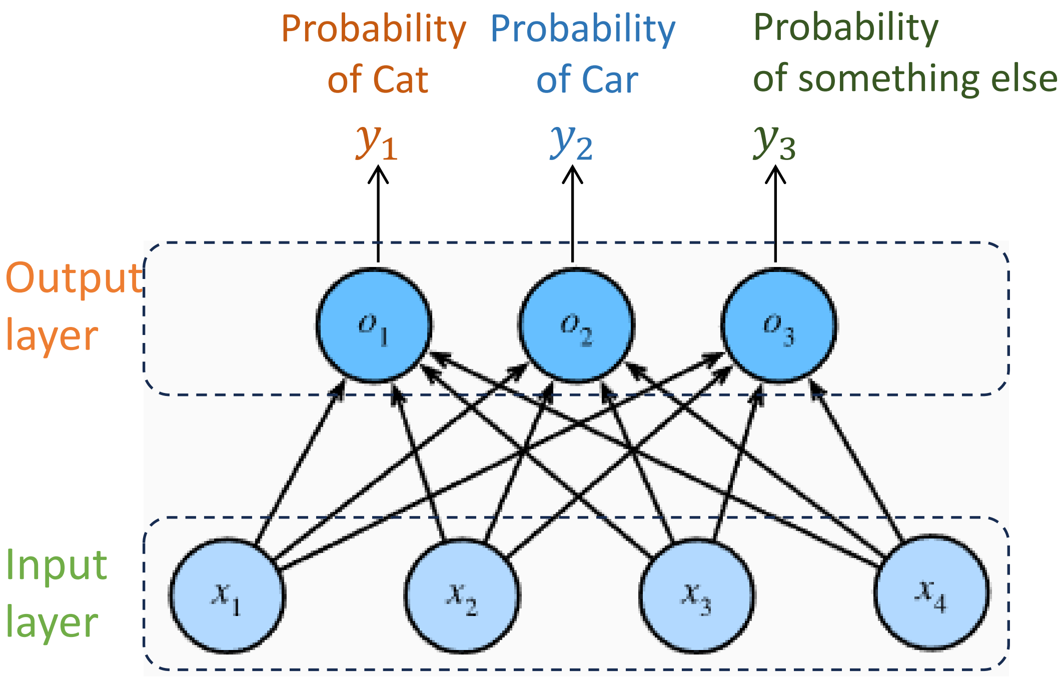


https://d2l.ai/chapter_linear-classification/softmax-regression.html

Extension to multiclass classification problem

- We would like to develop a model that takes a greyscale image and returns whether it is of a "cat", a "car", or "none of these"
  - Input
    - A greyscale image of size 2 x 2
  - Output:
    - The probability of being a cat → y1
    - The probability of being a car → y2
    - The probability of being none of these → y3
  - Network architecture – two layers
    - Input layer has 4 neurons – one per input value
    - The output layer has 3 neurons – one per class

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1, \\
o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2, \\
o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.
\end{aligned}
$$



https://d2l.ai/chapter_linear-classification/softmax-regression.html

Extension to multiclass classification problem

- Simple decision (activation) function
  - Can you explain why the simple decision function below won't work for the multiclass problem

$$\sigma(\text{output}) = \begin{cases} 1 \text{ if output} > 0 \\ 0 \text{ otherwise} \end{cases}$$

Extension to multiclass classification problem – Activation Functions

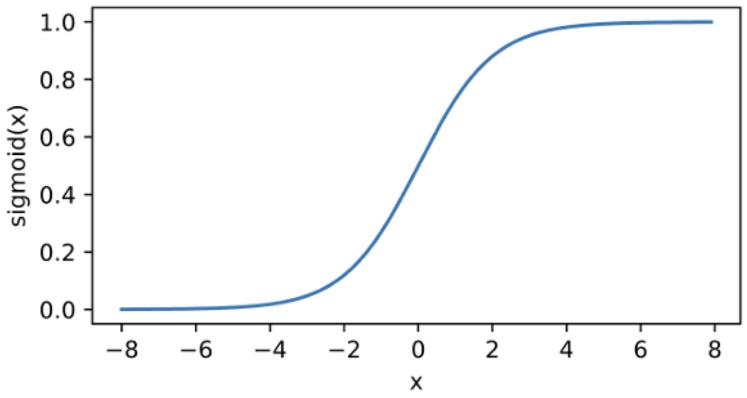
- Max function
  - Convert the output to values between 0 and 1
    - $y_1 = \frac{o_1}{o_1+o_2+o_3}$
    - $y_2 = \frac{o_2}{o_1+o_2+o_3}$
    - $y_3 = \frac{o_2}{o_1+o_2+o_3}$
  - The final class is the class that has the largest y.

- Softmax function
  - Map the output to probabilities by converting the output to values between 0 and 1
    - $y_i = \frac{\exp(o_i)}{\exp(o_1)+\exp(o_2)+\exp(o_3)}$
  - Indicates the probability of the input being of that class
  - The final class is the class that has the largest y.

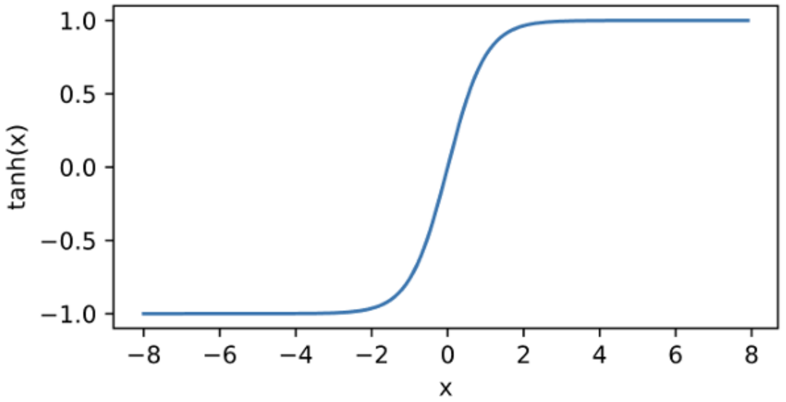
- These decision functions are also called activation function

Other Popular Activation Functions

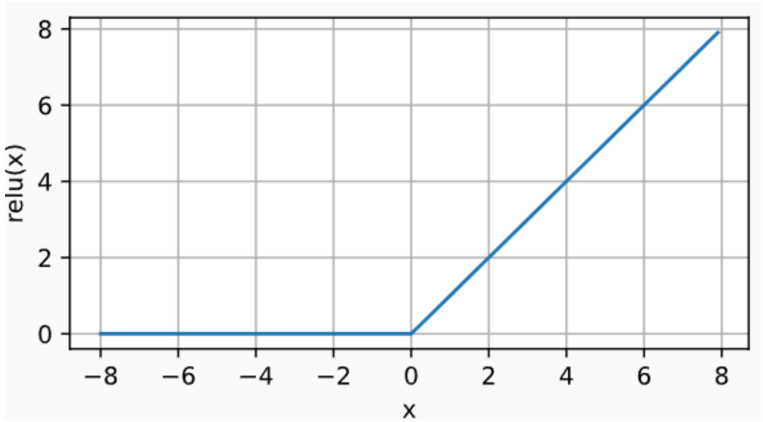
- Sigmoid activation
  - A soft version of the previous one
  - Maps output to [0, 1]
- Tanh activation
  - Maps output to [-1, 1]
- ReLU activation
  - Zeros negative outputs



$$\sigma(\text{output}) = \text{sigmoid}(\text{output}) = \frac{1}{1 + \exp(-\text{output})}$$



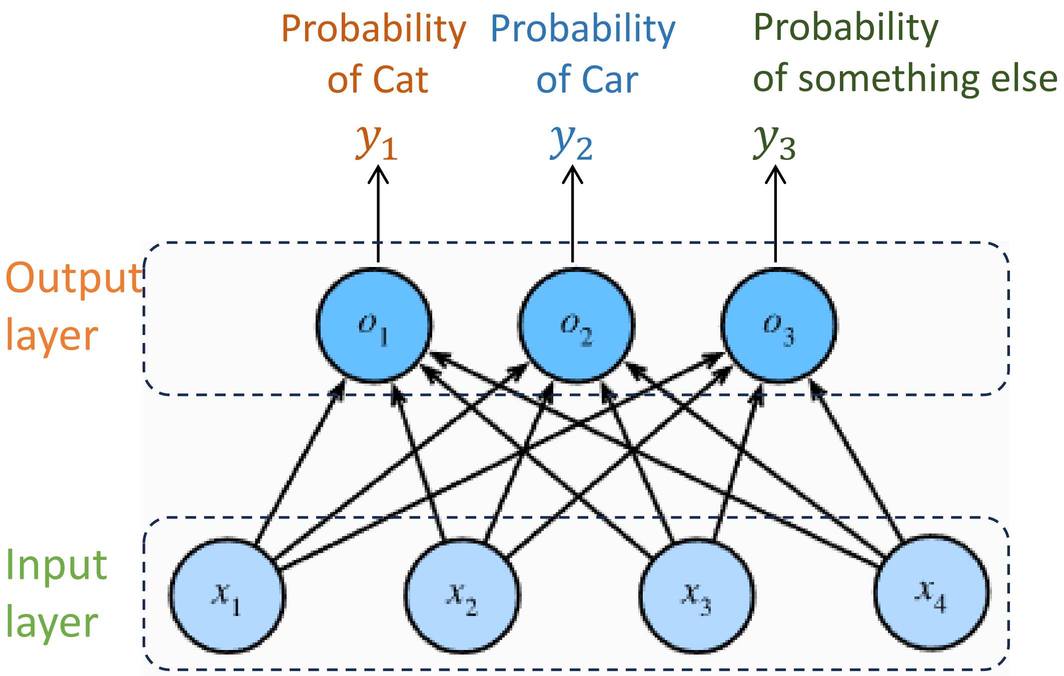
$$\sigma(\text{output}) = \tanh(\text{output}) = \frac{1 - \exp(-2\text{output})}{1 + \exp(-2\text{output})}$$



$$\sigma(\text{output}) = \text{ReLU}(\text{output}) = \max(\text{output}, 0)$$

Multiclass Classification Problem - Summary

- Input
  - Images of size W x H
- Output
  - $(y_1, y_2, \cdots, y_n, y_{n+1})$ where n is the number of classes we would like to recognize plus 1
  - The additional class corresponds to anything else, called background)
- Model - A network of two layers
  - Input layer has W X H neurons
  - Output layer has n neurons
- Activation function
  - Softargmax applied to the neurons in the output layer

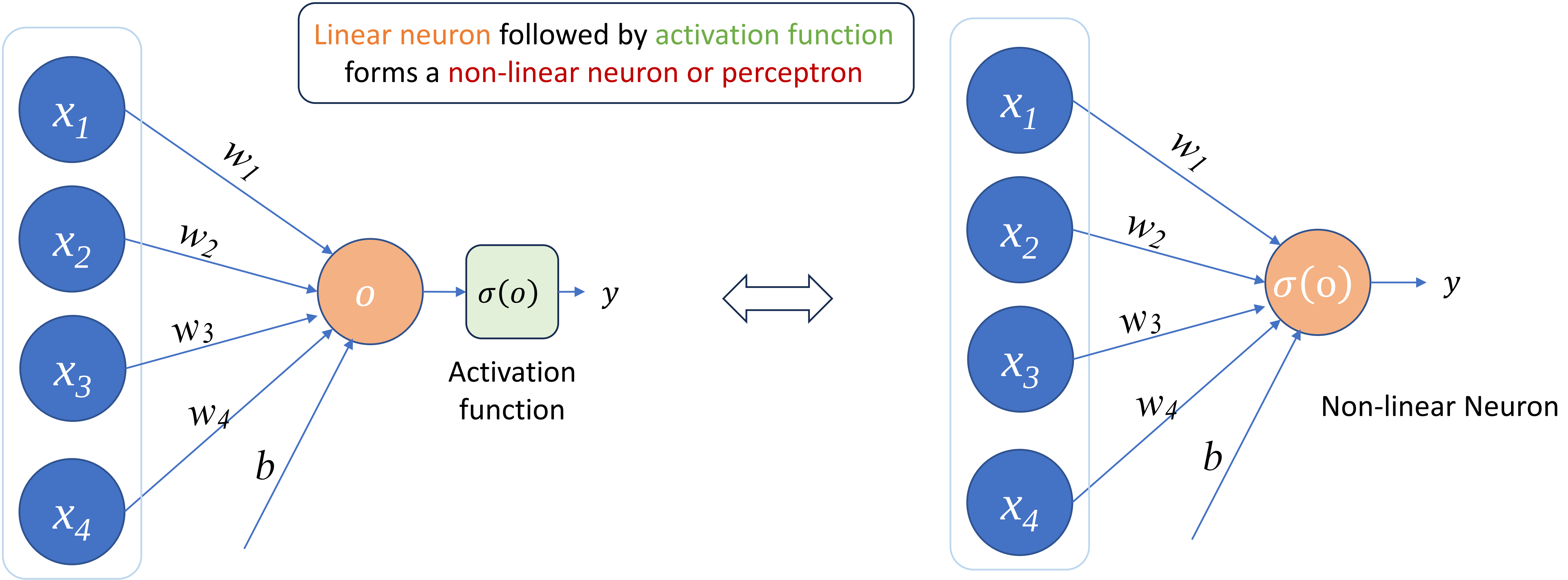


Implementation

- The full implementation will be discussed in the lab
- Questions to ask (and consider)
  - What is the computation time for a layer that has d inputs and q outputs?
  - How many parameters such layer will have?
  - Is it acceptable for real applications, e.g., when dealing with high resolution images and trying to recognize 1000 object categories?

Perceptron

- The activation function introduces non-linearity



Perceptron

- The activation function introduces non-linearity – why is it so important?
  - A single neuron cannot solve some (even simple) problems
  - To solve complex problems, we need to combine multiple neurons
  - Problem
    - Combining multiple linear neurons is the same as having one single linear neuron

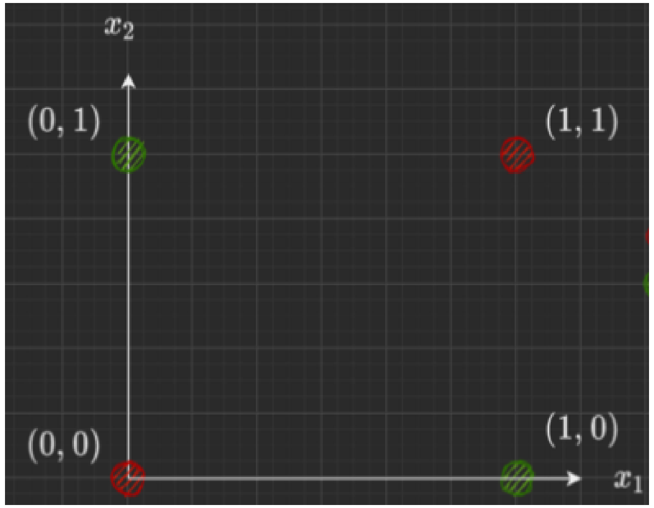
A Single Neuron cannot solve complex problems

- XOR Problem (Minsky & Papert, 1969)
  - Input
    - Two binary variables that take values either 0 or 1
  - Out put
    - Their XOR value (see the table on the right)

- This can be modelled as a binary classification problem
  - Class 0 represents the XOR value of 0
  - Class 1 represents the XOR value of 1

- Can you find a line that is able to separate the red class from the green class?

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Solution – Use Three Layers of Neurons

- Learning XOR
  - We can write

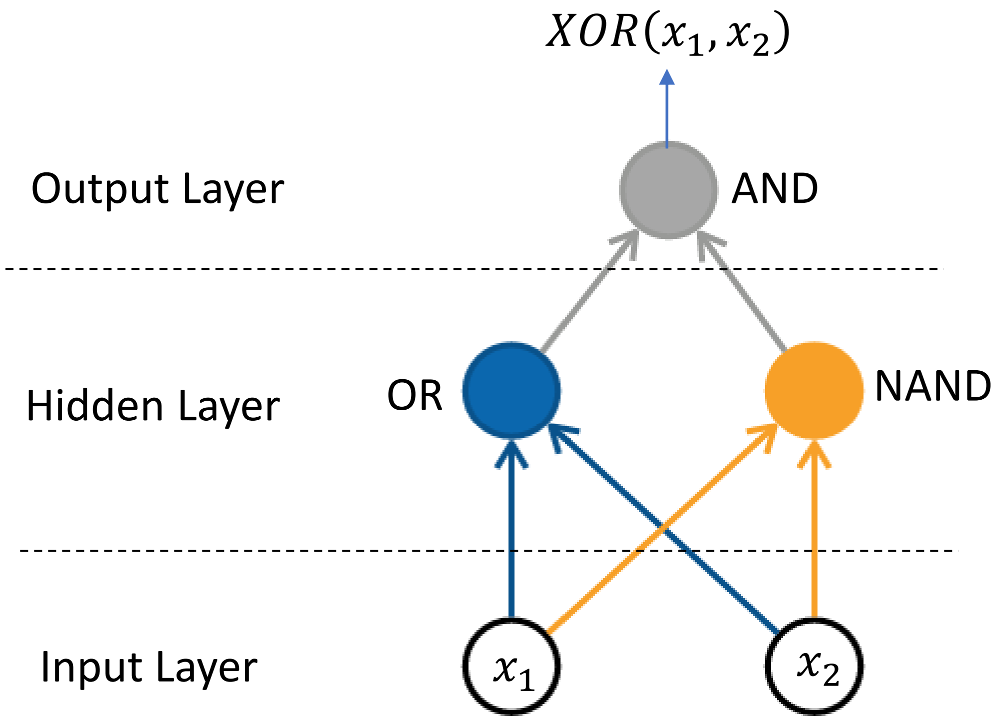
$$XOR(x_1, x_2) = \overbrace{\underbrace{(x_1 + x_2)}_{\text{OR}}\underbrace{(\overline{x_1 x_2})}_{\text{NAND}}}^{\text{AND}}$$

  - Show that OR, AND, and NAND (or Not AND) can be implemented using Perceptrons

Solution – Use Three Layers of Neurons

- Then XOR can be implemented using multiple layers
  - We can write

$$XOR(x_1, x_2) = \overbrace{\underbrace{(x_1 + x_2)}_{\text{OR}}\underbrace{(\overline{x_1 x_2})}_{\text{NAND}}}^{\text{AND}}$$

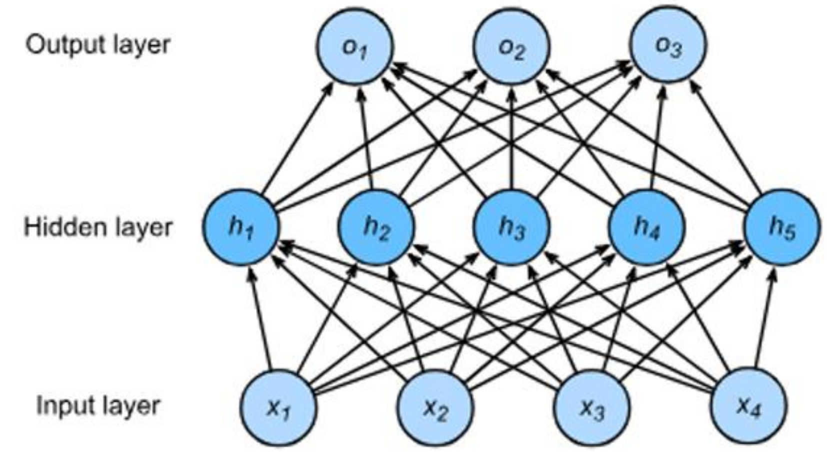


Multilayer Perceptron – Incorporating Hidden Layers

- Solve complex problems by incorporating one or multiple intermediate, called hidden, layers
  - Stack many fully connected layers on top of each other
  - Each layer feeds into the layer above until we generate outputs

Predictor layer

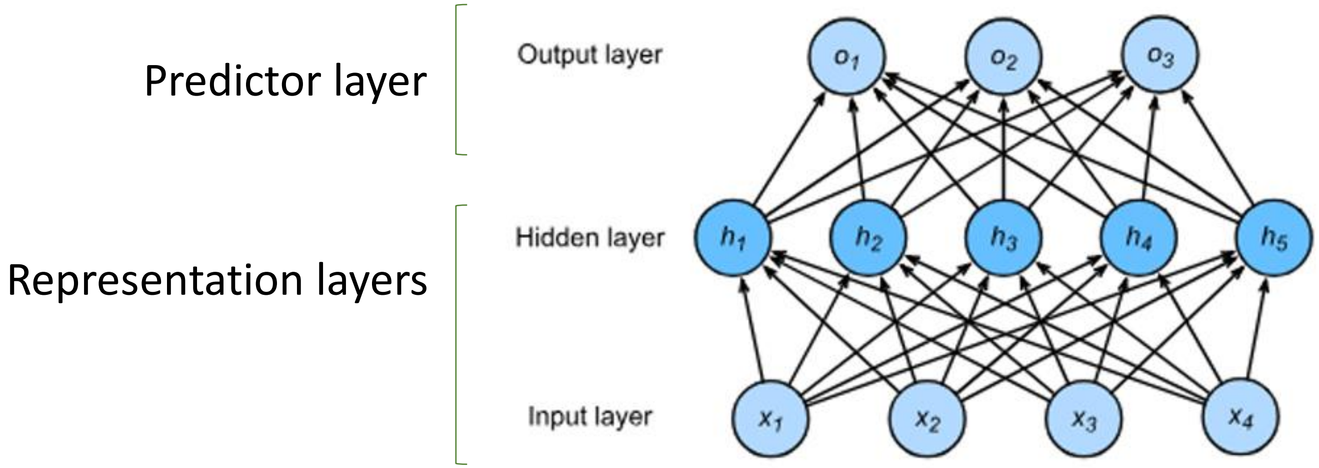
Representation layers



Hyperparameter - size m of hidden layer

Multilayer Perceptron – Incorporating Hidden Layer

- Problem
  - Show that cascading multiple linear layers is the same as having a network with just one layer of neurons (use a simple example)



Hyperparameter - size m of hidden layer

Multilayer Perceptron – Incorporating Hidden Layer

- Problem
  - Show that cascading multiple linear layers is the same as having a network with just one layer of neurons (use a simple example)
  - Let h be the output of the hidden layer

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

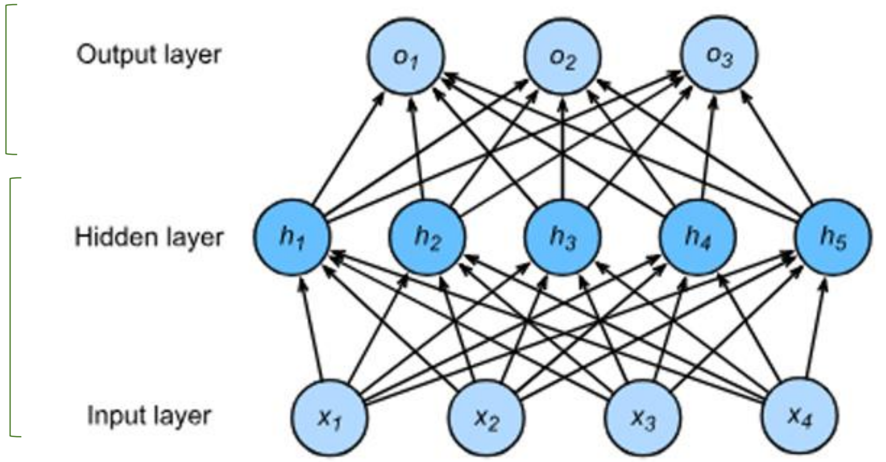
  - Let o be the output of the last layer

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

  - Hence:

$$o = \boxed{\mathbf{w}_2^\top \mathbf{W}_1}\mathbf{x} + b'$$

Combining linear layers is equivalent to on single linear layer



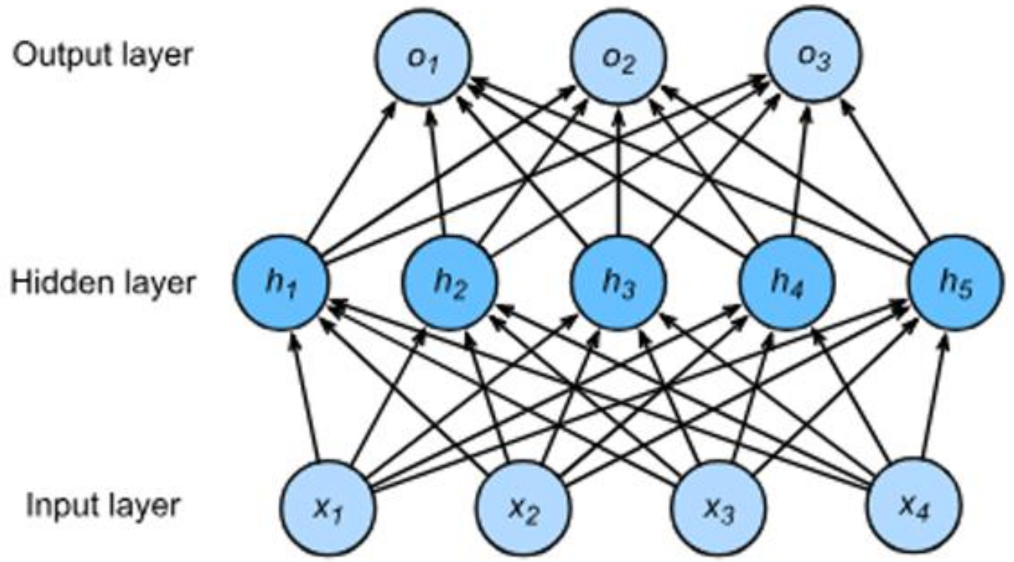
Hyperparameter - size m of hidden layer

Multilayer Perceptron (MLP) as Universal Approximators

- By applying nonlinear activation function σ to each hidden unit, MLPs become universal approximators
  - Input $\mathbf{x} \in \mathbb{R}^n$
  - Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
  - Output $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

σ is an element-wise activation function

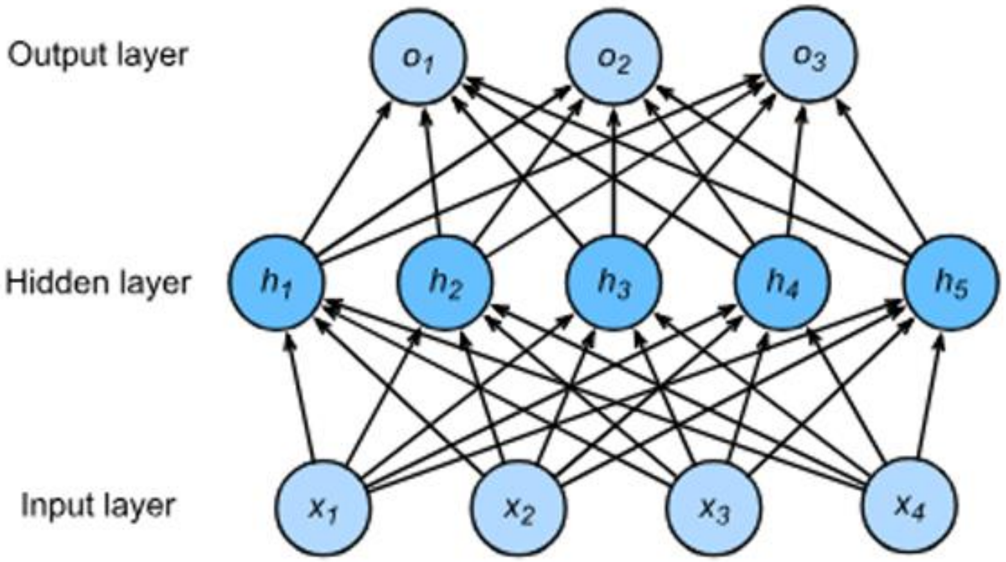


Hyperparameter - size m of hidden layer

Multilayer Perceptron (MLP) as Universal Approximators

- By applying nonlinear activation function σ to each hidden unit, MLPs become universal approximators

- Commonly used activation functions σ
  - In the input layer
    - None
  - In the intermediate (hidden) layer(s)
    - Rectified Linear Unit (ReLU)
  - In the last (output) layer
    - Softmax



Hyperparameter - size m of hidden layer

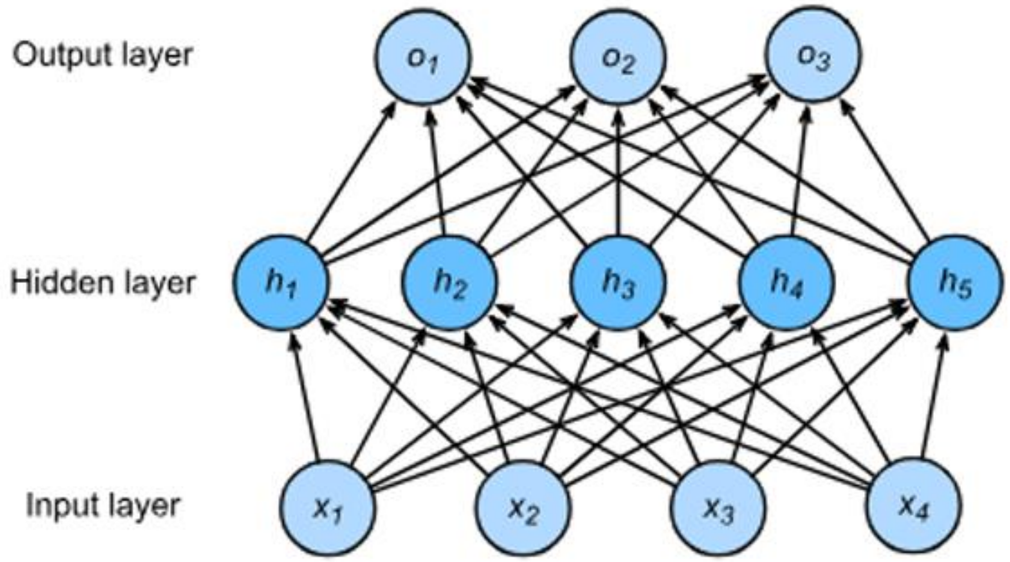
Multilayer Perceptron (MLP) as Universal Approximators

- By applying nonlinear activation function σ to each hidden unit, MLP become universal approximators
  - Input $\mathbf{x} \in \mathbb{R}^n$
  - Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
  - Output $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \mathrm{softmax}(o)$$

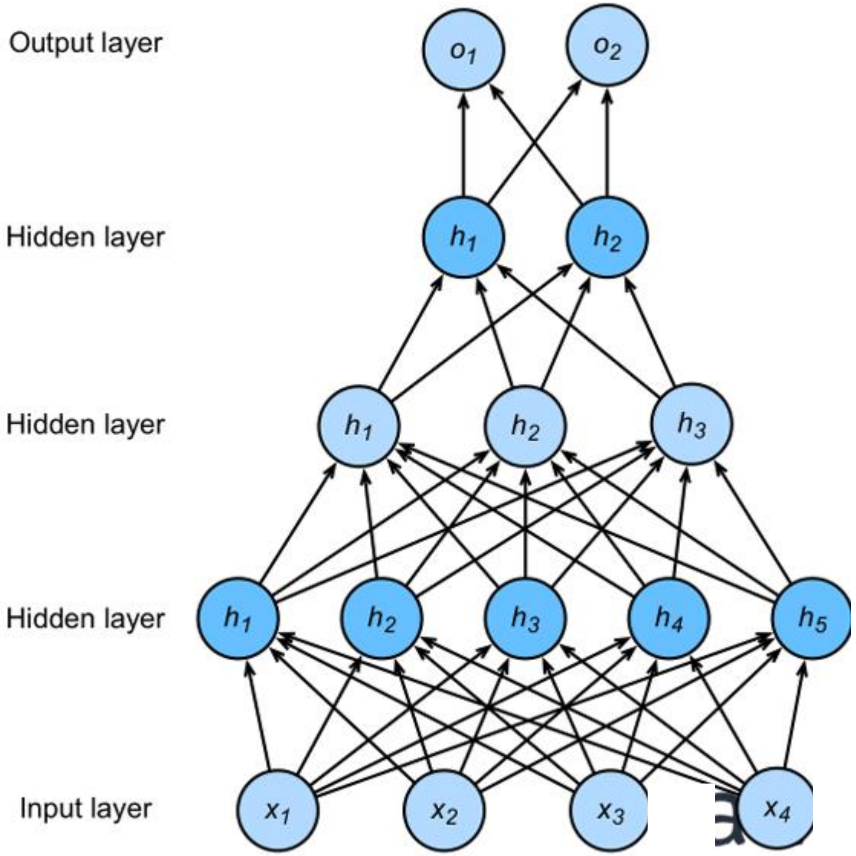


Hyperparameter - size m of hidden layer

Multilayer Perceptron (MLP) – Multiple Hidden Layers

- We can cascade multiple hidden layers to approximate complex functions

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$



- Hyper parameters
  - Number of hidden layers
  - Number of neurons in each hidden layer

Summary of what we have seen so far

- Linear neurons
  - Can be used for linear regression
  - To use them for classification, need to add an activation function (softmax) at the output layer
  - Cascading multiple layers of linear neurons is the same as having one single linear neuron
  - Cannot approximate many functions
- Multilayer Perceptrons (MLPs) as universal approximators
  - By adding nonlinear activation functions to the hidden layers, we can approximate any function
  - Can have one or multiple hidden layers (in addition to the input and output layers)
- Importance of activation functions
  - Introduce nonlinearity → allows to cascade multiple layers → made NN universal approximators

Training MLPs

- Training is the process of finding the best parameters of the network (i.e., the weights of the different connections)
  - Feed to the network an input, with known (desired) output
  - Compute, in a forward pass, the output of the network
  - Measure the discrepancy between the obtained output and the desired output
  - Update the network weights in such a way the discrepancy is reduced
  - Repeat the process until convergence, i.e., the change in the loss is minimal

Training MLPs

- Training is the process of finding the best parameters of the network (i.e., the weights of the different connections)
  - Feed to the network an input, with known (desired) output → Need training data
  - Compute, in a forward pass, the output of the network
  - Measure the discrepancy between the obtained output and the desired output → Need a loss function
  - Update the network weights in such a way the discrepancy is reduced → Gradient descent algorithm (or any of its latest variants)
  - Repeat the process until convergence, i.e., the change in the loss is minimal → Need to define a stopping criteria

Training – Loss Functions

- Let
  - $y = (y_1, y_2, \dots, y_q)$: the ground truth output
  - $\hat{y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_q)$: the output produced by the network

- L2 Loss
  - $l(y, \hat{y}) = \sqrt{\sum (y_i - \hat{y}_i)^2}$

- L1 Loss
  - $l(y, \hat{y}) = \sum |y_i - \hat{y}_i|$

- Cross entropy Loss
  - Suitable when using softmax activation

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

  - Its gradient:

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \mathrm{softmax}(\mathbf{o})_j - y_j.$$

Training – Loss Functions

- Use either of the losses defined in the previous slide. Let's call it L
- Update the network weights in such a way the discrepancy (loss) is reduced
- However,
  - We need to ensure that the weights do not grow undefinitely by imposing a constraint s on their norm (this is called L2 regularlization)

$$s = \frac{\lambda}{2} \|\boldsymbol{W}\|^2$$

- Overall loss (called also objective function)

$$J = L + s$$

Training MLPs

- Forward propagation (or forward pass)
  - It is the process of propagating the input of the network through all the layers, from the input layer to the output layer
  - It includes the calculation and storage of intermediate variables (including outputs)

- Once output is obtained for a given training sample,
  - Compute the objective function

$$h = \rho(z)$$

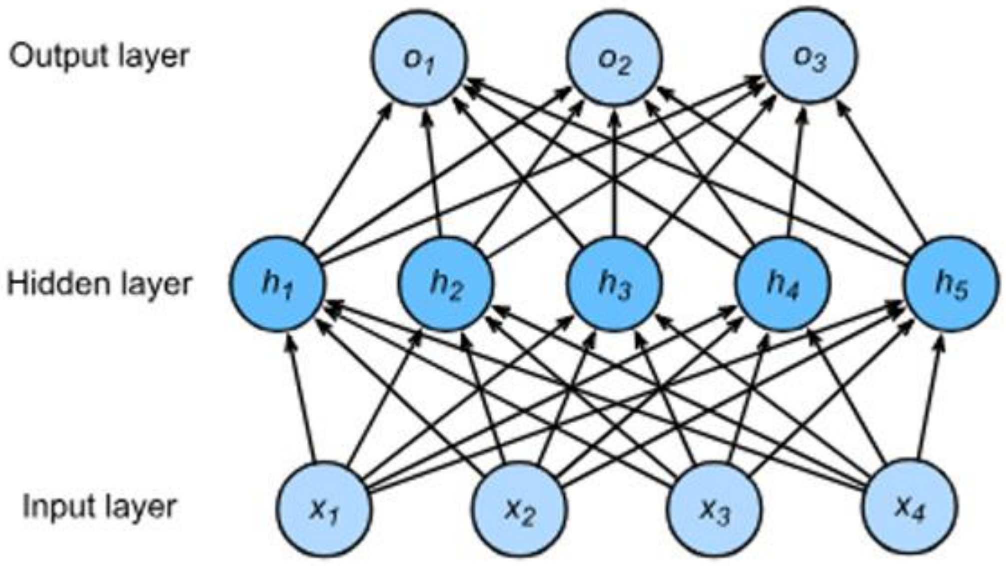
  - Compute its gradient with respect to the parameters
  - Update the parameters

$$o = W^{(2)}h$$

$$h = \rho(z)$$

$$z = W^{(1)}X$$

X



- $W^{(1)}$: weights of the connections from the input layer to the hidden layer
- $W^{(2)}$: weights of the connections from the hidden layer to the output layer

Training MLPs

- Backpropagation
  - This is the process of computing the gradient of the neural network parameters (i.e., the gradient of the loss function with respect to the NN parameters)
  - Introduced in 1980s and was considered as a major breakthrough in neural networks as it allows training, efficiently, neural networks composed of multiple layers

- When training neural networks
  - Initialize the parameters
  - Alternate between forward propagation and backward propagation until convergence
    - Forward propagation sequentially computes and stores intermediate variables (proceeds from the input to the output)
    - Backpropagation sequentially calculates and stores the gradients of intermediate variables and parameters in the reversed order (from output to input)

MLP – Implementation

- Will be covered in the lab

Summary

- Perceptron
  - Easy, simple but limited function complexity – cannot represent complex relations between the input and output
- Multilayer perceptron
  - Multiple layers add more complexity
  - Nonlinearity is needed → activation function at the hidden layers
- Activation functions and their importance
  - Without activation function an MLP is equivalent to a single neuron
- Training
  - Loss function
  - Training process

Next Week

- More about training (practical issues to take into account)

Questions