Neural Networks

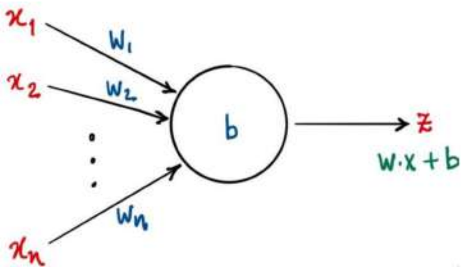
Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Data Mining and Machine Learning
August–December 2020

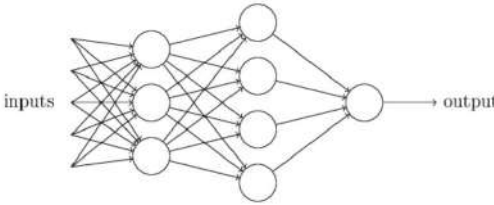
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)



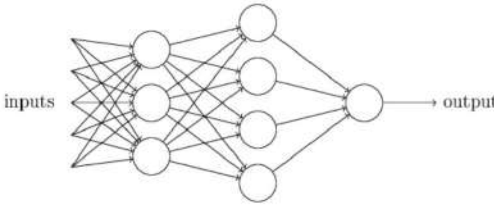
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?



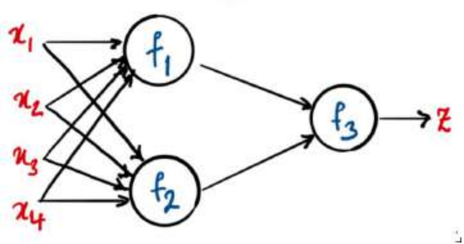
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator

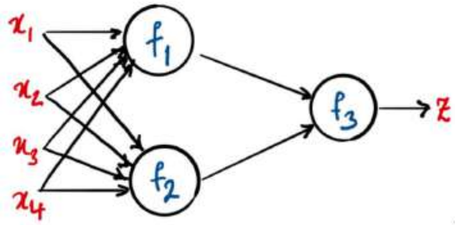


Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$

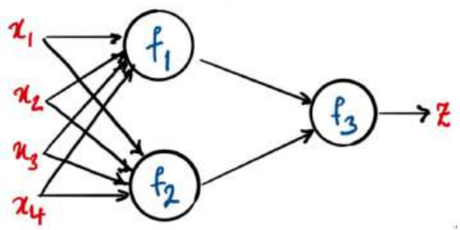


- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
  - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$



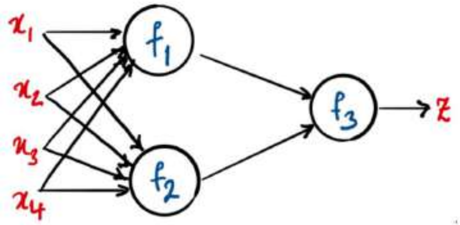
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
  - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$
  - $f_3 = w_3 \cdot \langle w_1 \cdot x + b_1, w_2 \cdot x + b_2 \rangle + b_3$

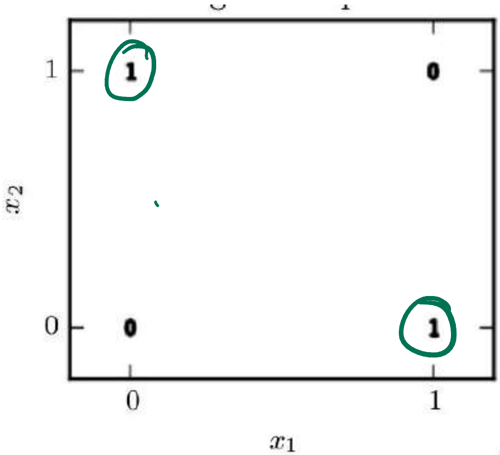


Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
  - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$
  - $f_3 = w_3 \cdot \langle w_1 \cdot x + b_1, w_2 \cdot x + b_2 \rangle + b_3$
  - $f_3 = \sum_{i=1}^{4} (w_{3_1} w_{1_i} + w_{3_2} w_{2_i}) \cdot x_i$
    $+(w_{3_1} b_1 + w_{3_2} b_2 + b_3)$

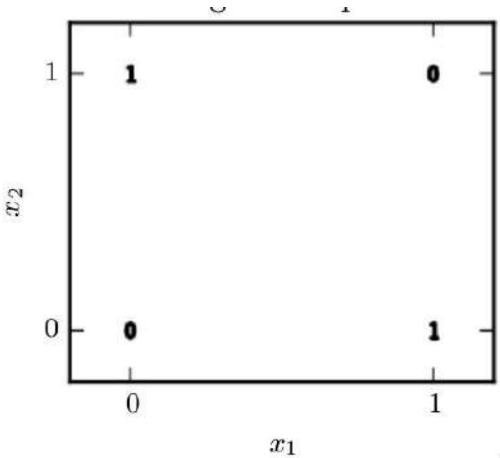


- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)



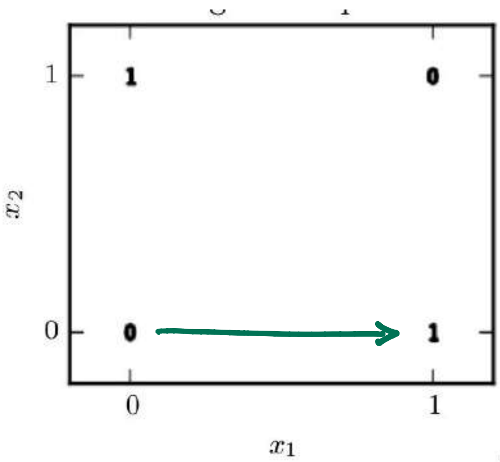
Limits of linearity

- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$

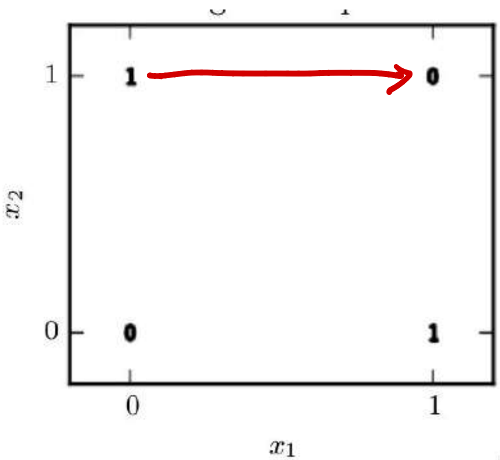


Limits of linearity

- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$

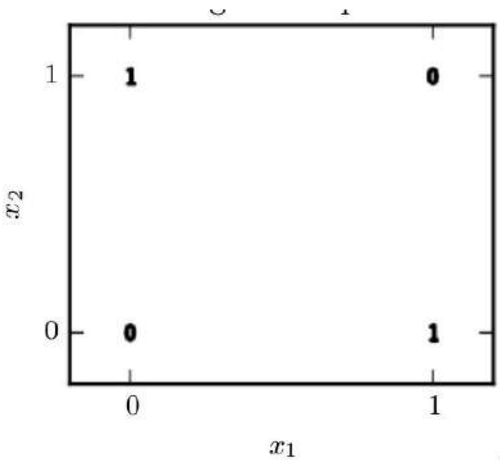


- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$
- $x_2 = 1$: As x_1 goes from 0 to 1, output goes from 1 to 0, so $u < 0$

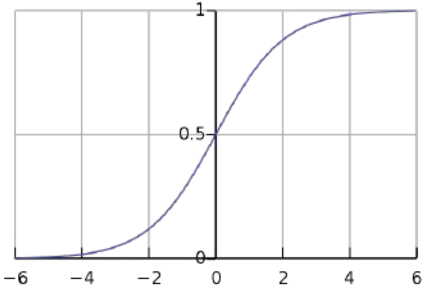


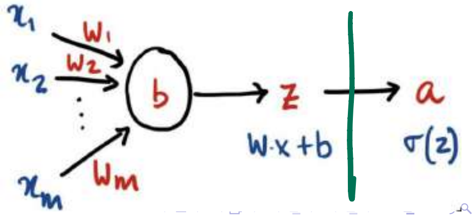
Limits of linearity

- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$
- $x_2 = 1$: As x_1 goes from 0 to 1, output goes from 1 to 0, so $u < 0$
- Observed by Minsky and Papert, 1969, first "AI Winter"



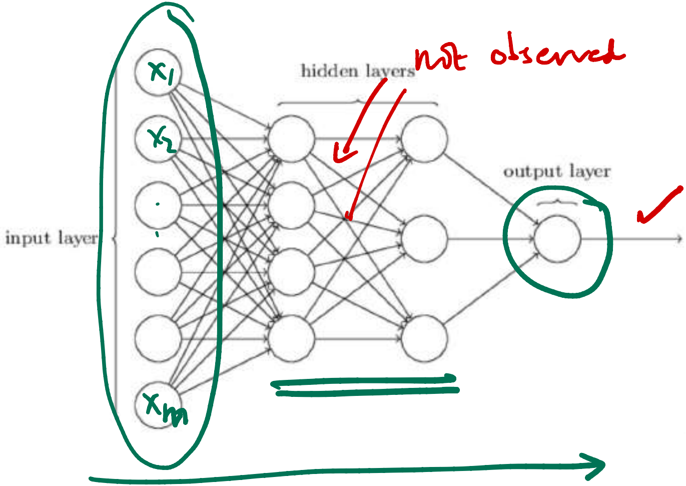
- Transform linear output z through a non-linear activation function
- Sigmoid function $\dfrac{1}{1+e^{-z}}$





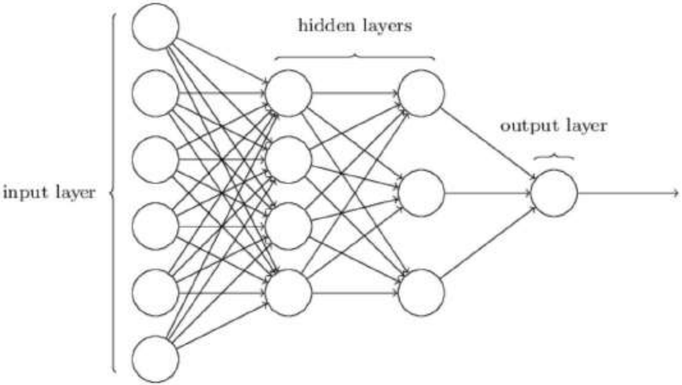
Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer



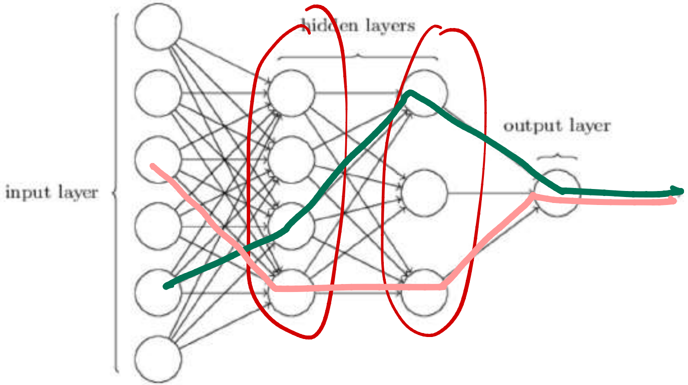
Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer
- Assumptions



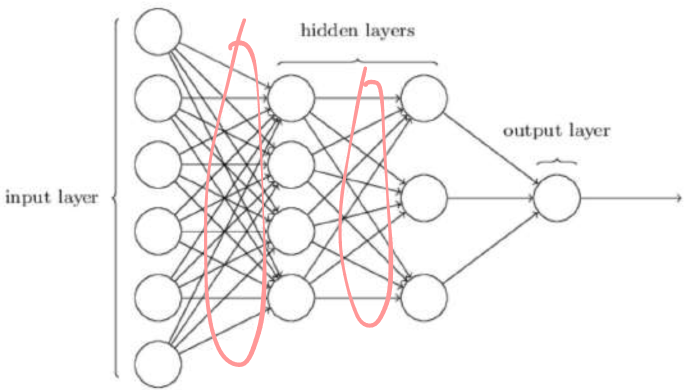
Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
  - Hidden neurons are arranged in layers

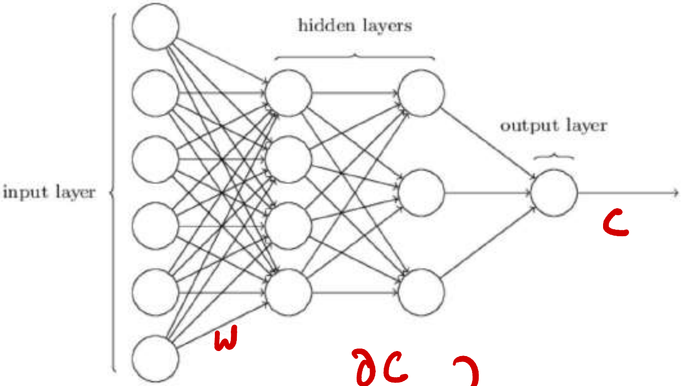


Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
  - Hidden neurons are arranged in layers
  - Each layer is fully connected to the next



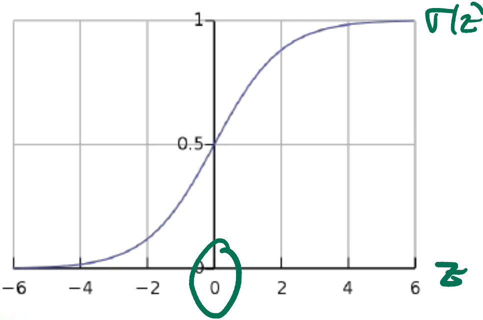
- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
  - Hidden neurons are arranged in layers
  - Each layer is fully connected to the next
  - Set weight to zero to remove an edge

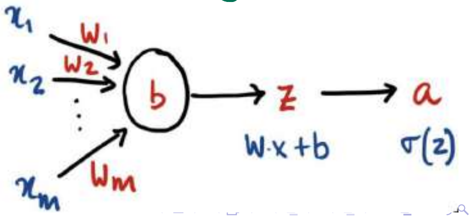


$\frac{\partial C}{\partial w}$?

Non-linear activation

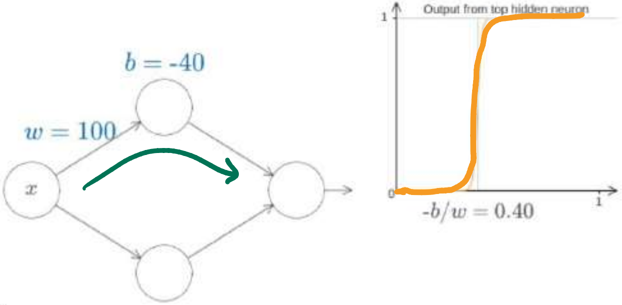
- Transform linear output z through a non-linear activation function
- Sigmoid function $\dfrac{1}{1+e^{-z}}$ $\dfrac{1}{1+1} = \dfrac{1}{2}$
- Step is at $z = 0$
  - $z = wx + b$, so step is at $x = \cancel{-w/b}$ $-\dfrac{b}{w}$
  - Increasing w makes step steeper





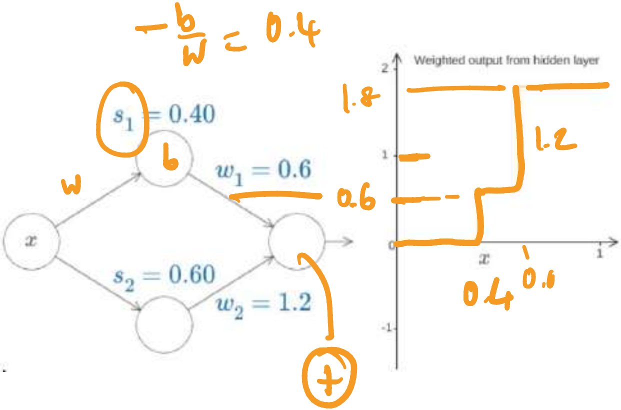
Universality

- Create a step at $x = -\cancel{w/b}$ b/w



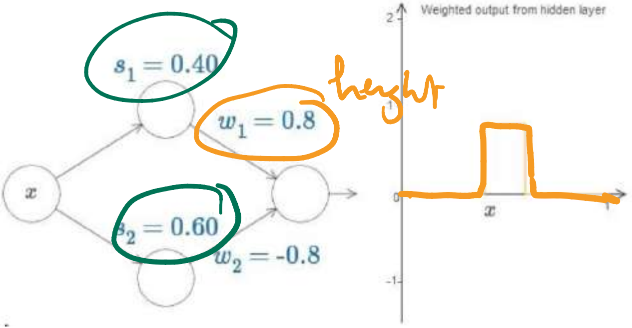
Universality

- Create a step at $x = -w/b$
- Cascade steps



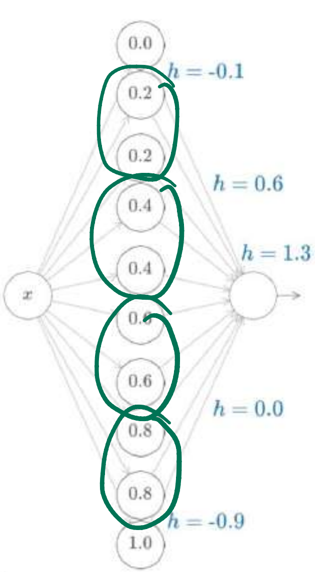
Universality

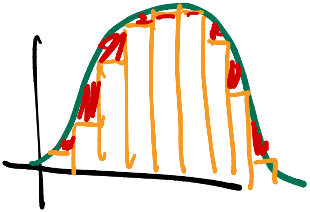
- Create a step at $x = -w/b$
- Cascade steps
- Subtract steps to create a box



Universality

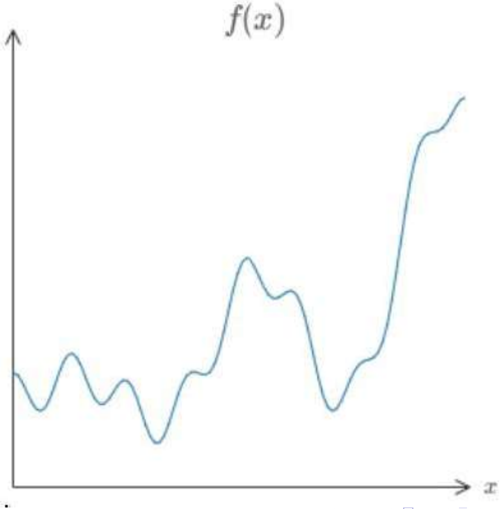
- Create a step at $x = -w/b$
- Cascade steps
- Subtract steps to create a box
- Create many boxes





Universality

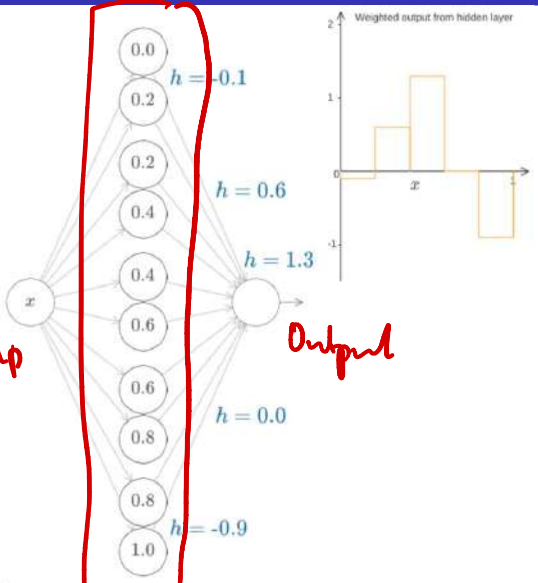
- Create a step at $x = -w/b$
- Cascade steps
- Subtract steps to create a box
- Create many boxes
- Approximate any function



Universality

- Create a step at $x = -w/b$
- Cascade steps
- Subtract steps to create a box
- Create many boxes
- Approximate any function
- Need only one hidden layer!

Inp



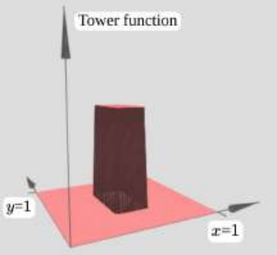
Output

Non-linear activation

- With non-linear activation, network of neurons can approximate any function

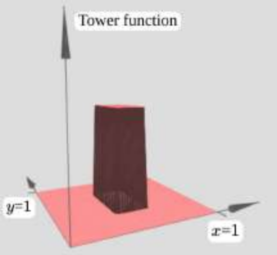
Non-linear activation

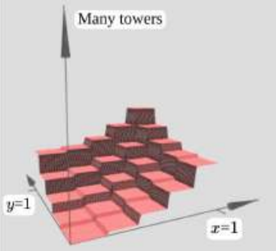
- With non-linear activation, network of neurons can approximate any function
  - Can build "rectangular" blocks



Non-linear activation

- With non-linear activation, network of neurons can approximate any function
  - Can build “rectangular” blocks
  - Combine blocks to capture any classification boundary





Example: Recognizing handwritten digits

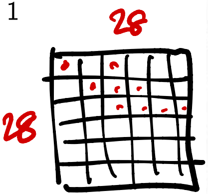
- MNIST data set

Example: Recognizing handwritten digits

- MNIST data set
- 1000 samples of 10 handwritten digits
  - Assume input has been segmented

Example: Recognizing handwritten digits

- MNIST data set
- 1000 samples of 10 handwritten digits
  - Assume input has been segmented
- Each digit is 28×28 pixels
  - Grayscale value, 0 to 1
  - 784 pixels



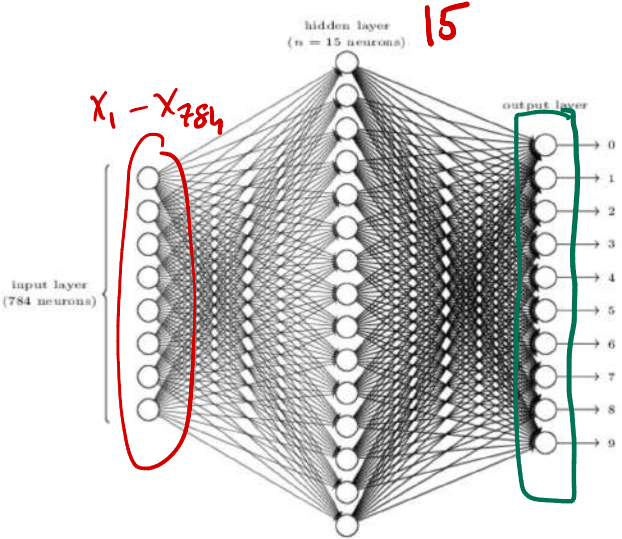
Example: Recognizing handwritten digits

- MNIST data set
- 1000 samples of 10 handwritten digits
  - Assume input has been segmented
- Each digit is 28×28 pixels
  - Grayscale value, 0 to 1
  - 784 pixels
- Input $x = (x_1, x_2, \ldots, x_{784})$

0-1 0-1

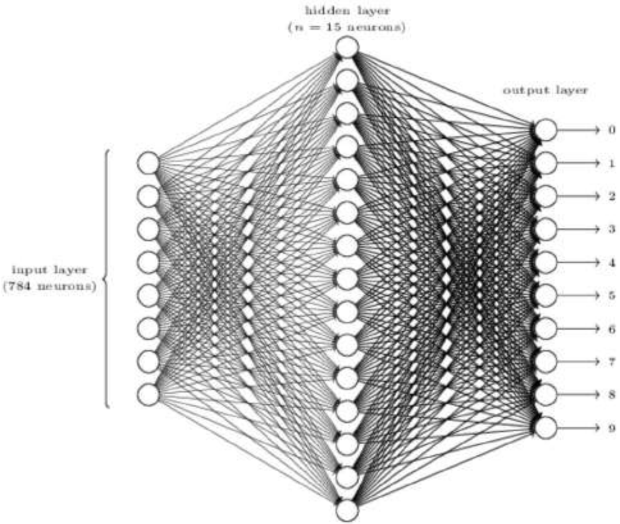
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$



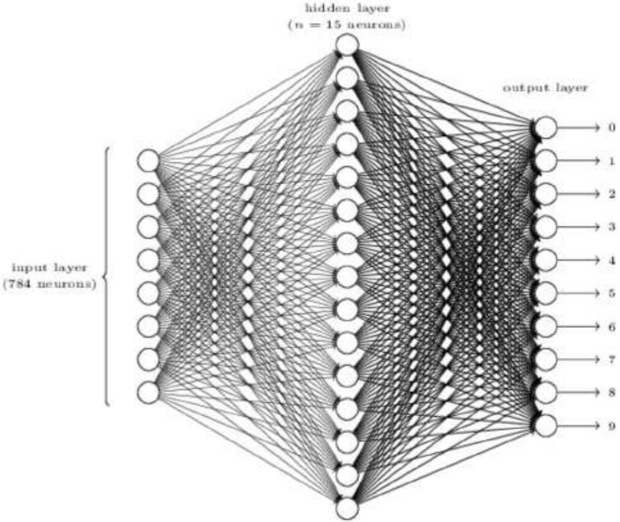
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes



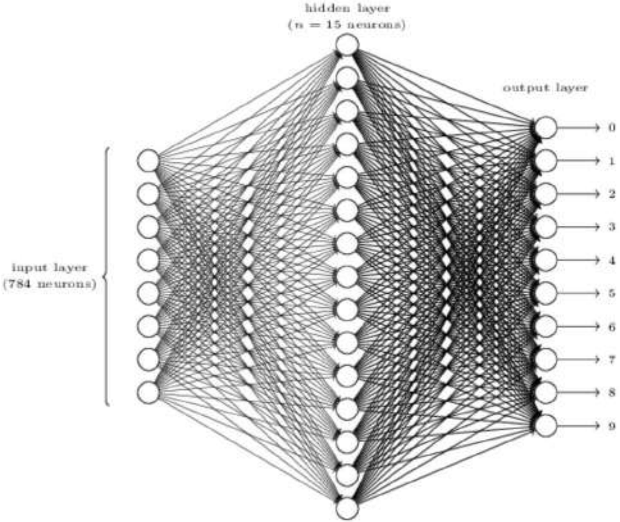
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes
- Output layer, 10 nodes
  - Decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$



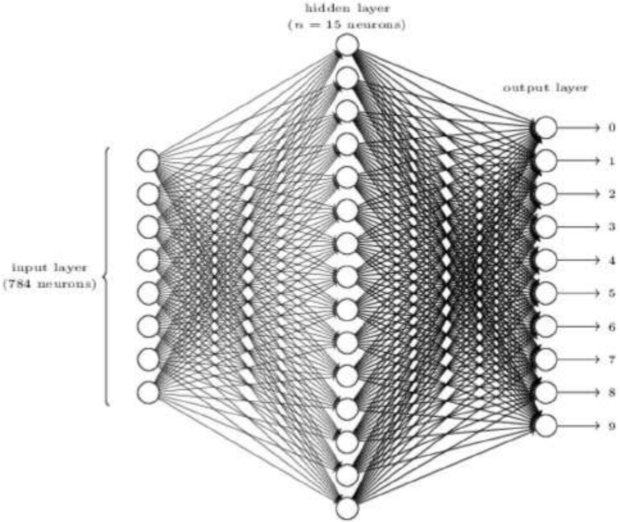
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes
- Output layer, 10 nodes
  - Decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$
- Final output is best a_j



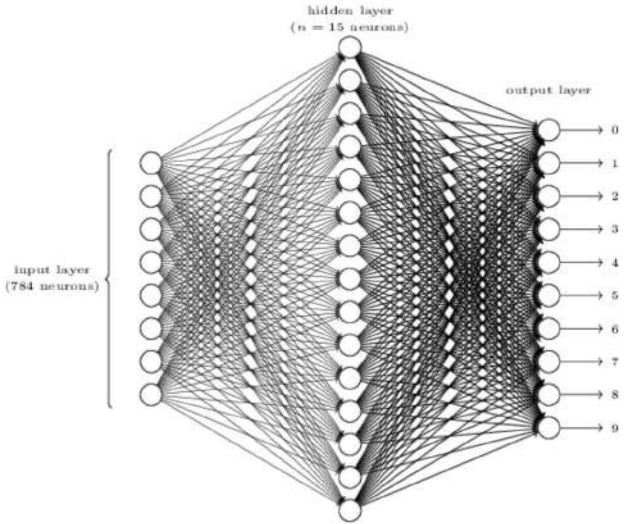
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes
- Output layer, 10 nodes
  - Decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$
- Final output is best a_j
  - Naïvely, $\arg\max_j a_j$



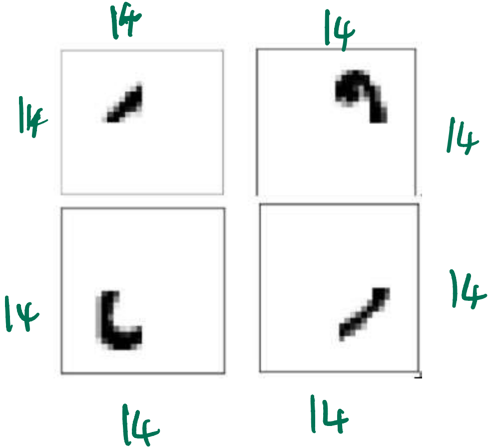
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes
- Output layer, 10 nodes
  - Decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$
- Final output is best a_j
  - Naïvely, $\arg\max_j a_j$
  - Softmax, $\arg\max_j \dfrac{e^{a_j}}{\sum_j e^{a_j}}$
    - "Smooth" version of $\arg\max$



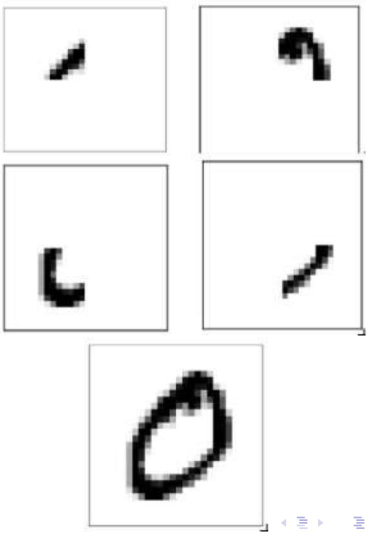
Example: Extracting features

- Hidden layers extract features
  - For instance, patterns in different quadrants



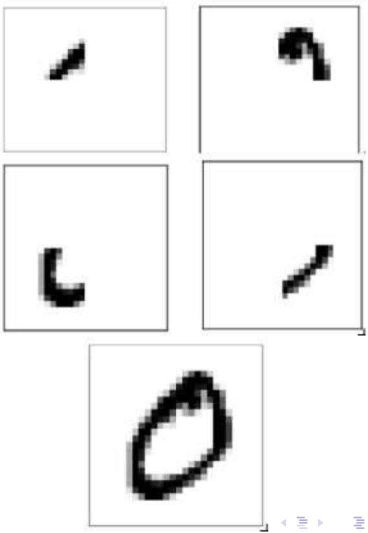
Example: Extracting features

- Hidden layers extract features
  - For instance, patterns in different quadrants
- Combination of features determines output



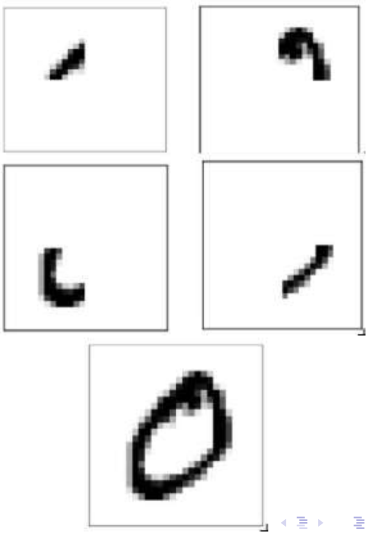
Example: Extracting features

- Hidden layers extract features
  - For instance, patterns in different quadrants
- Combination of features determines output
- Claim: Automatic identification of features is strength of the model



Example: Extracting features

- Hidden layers extract features
  - For instance, patterns in different quadrants
- Combination of features determines output
- Claim: Automatic identification of features is strength of the model
- Counter argument: implicitly extracted features are impossible to interpret
  - Explainability



Example: Extracting features

- Hidden layers extract features
    - For instance, patterns in different quadrants
- Combination of features determines output
- Claim: Automatic identification of features is strength of the model
- Counter argument: implicitly extracted features are impossible to interpret
    - Explainability