

Lecture 19: 07 April, 2022

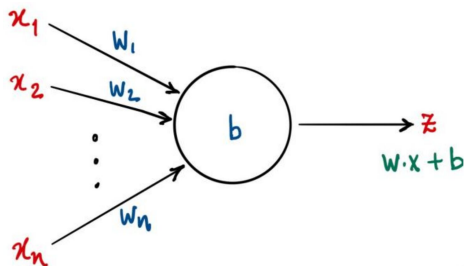
Madhavan Mukund

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

Data Mining and Machine Learning
January–May 2022

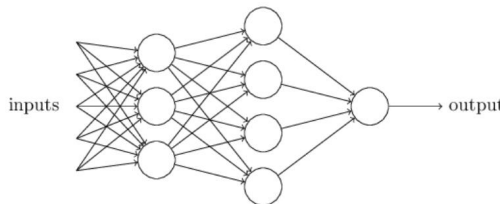
Linear separators and Perceptrons

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



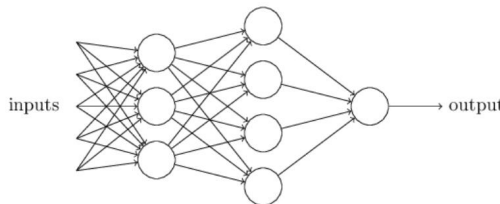
Linear separators and Perceptrons

- 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

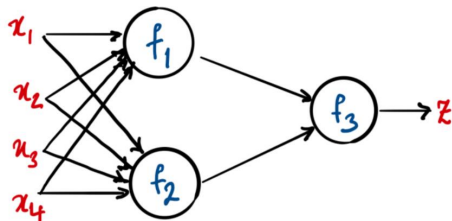
- Perceptrons define linear separators $w \cdot x + b$
 - $w \cdot x + b > 0$, classify Yes (+1)
 - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator



Linear separators and Perceptrons

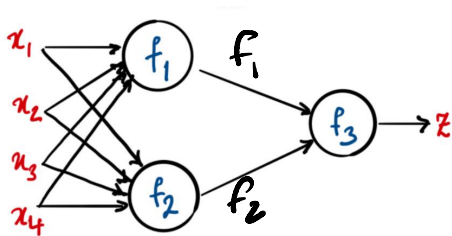
- Perceptrons define linear separators $w \cdot x + b$
 - $w \cdot x + b > 0$, classify Yes (+1)
 - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
 - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$

w_1, w_2, w_3, w_4
 f_1, x_2, x_3, x_4



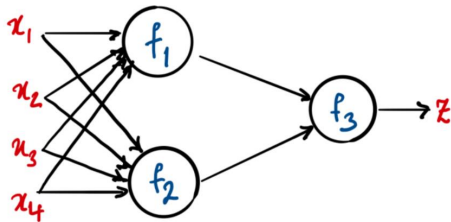
Linear separators and Perceptrons

- 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
 - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
 - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$



Linear separators and Perceptrons

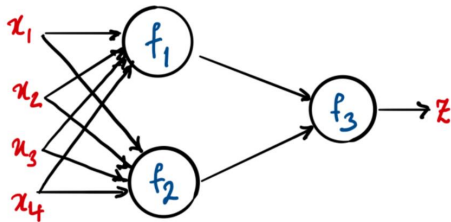
- Perceptrons define linear separators $w \cdot x + b$
 - $w \cdot x + b > 0$, classify Yes (+1)
 - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
 - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
 - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle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



1

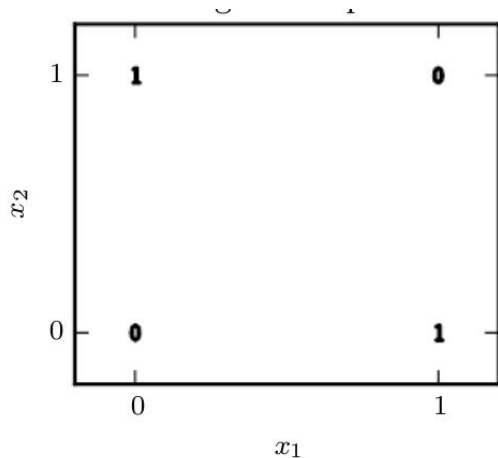
Linear separators and Perceptrons

- 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?
- Result is still a linear separator
 - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
 - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$
 - $f_3 = w_3 \cdot \langle w_1 \cdot x + b_1, w_2 \cdot x + b_2 \rangle + b_3$
 - $f_3 = \sum_{i=1}^4 (w_{31} w_{1i} + w_{32} w_{2i}) \cdot x_i + (w_{31} b_1 + w_{32} b_2 + b_3)$



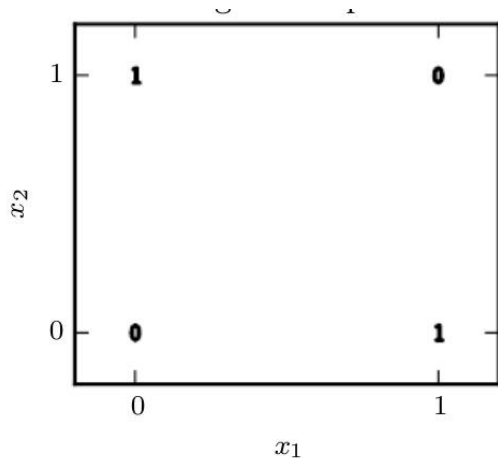
Limits of linearity

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



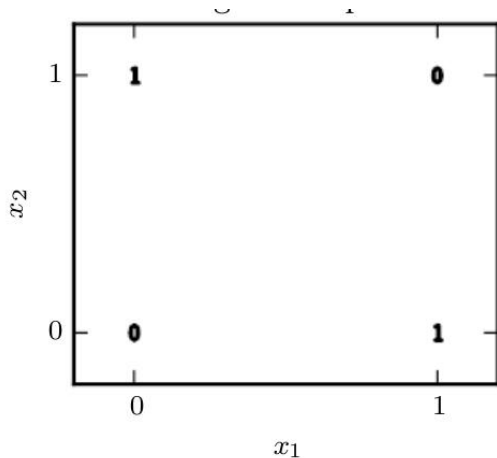
Limits of linearity

- 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 $XOR(x_1, x_2) = ux_1 + vx_2 + b$



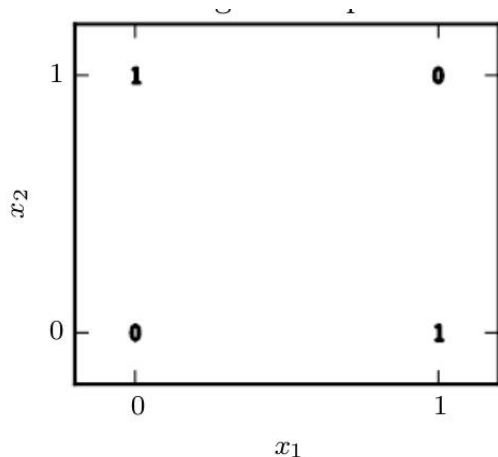
Limits of linearity

- 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 $u > 0$



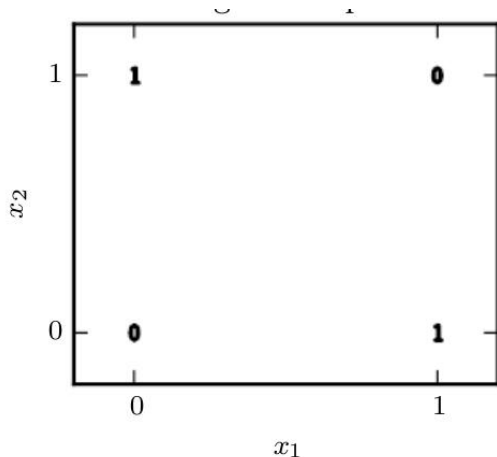
Limits of linearity

- 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 $u < 0$



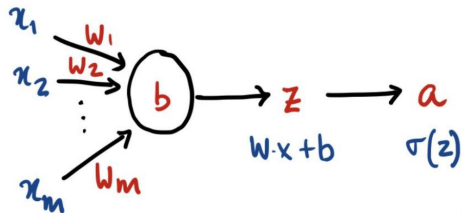
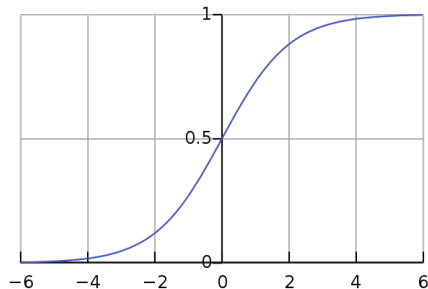
Limits of linearity

- 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”



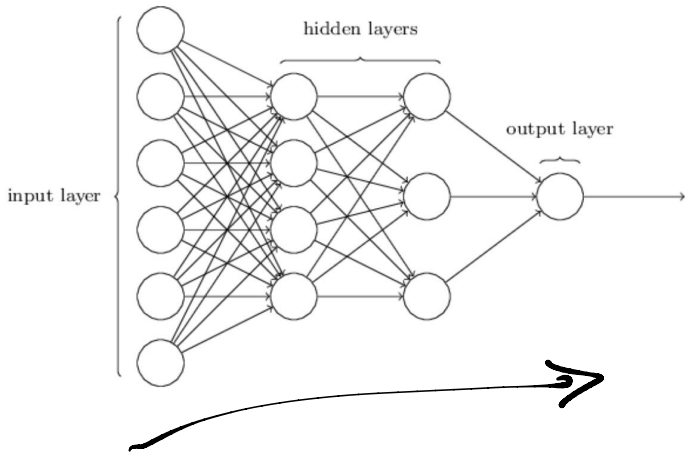
Non-linear activation

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



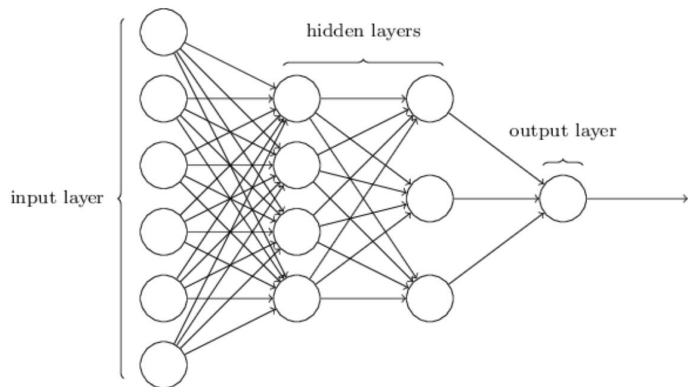
Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer



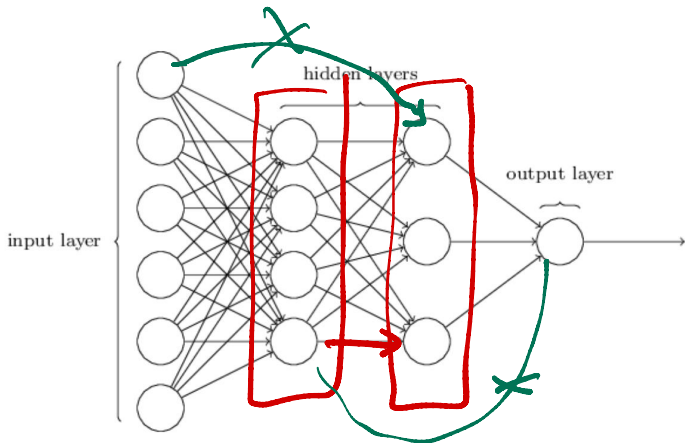
Structure of a neural network

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



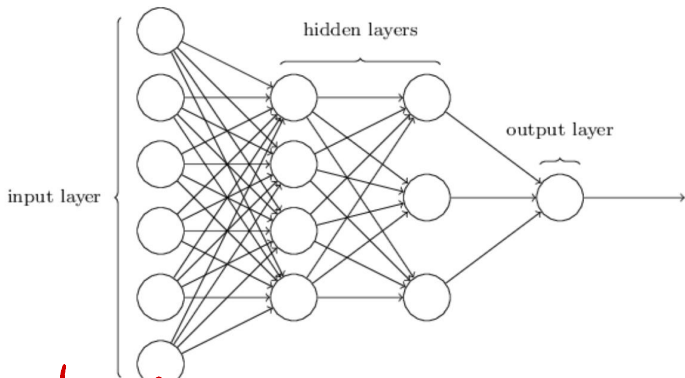
Structure of a neural network

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



Structure of a neural network

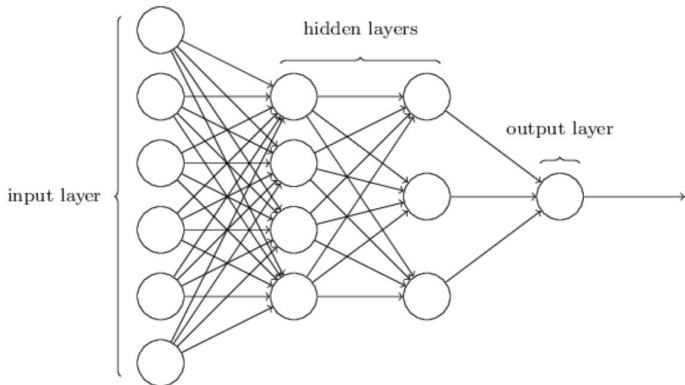
- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
 - Hidden neurons are arranged in layers
 - Each layer is fully connected to the next



*How many layers?
Nodes per layer*

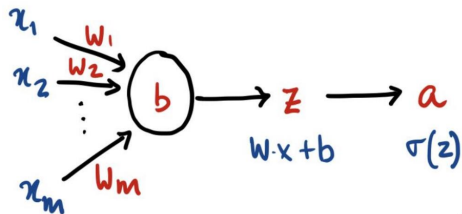
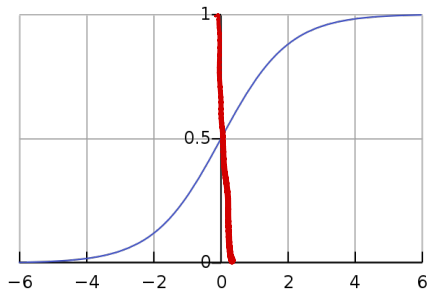
Structure of a neural network

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

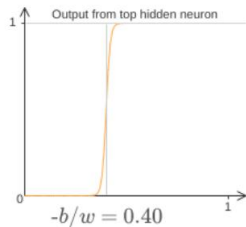
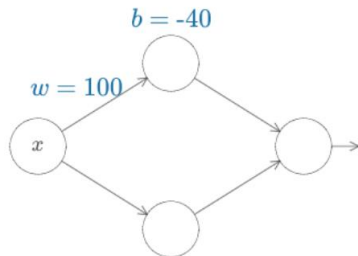


Non-linear activation

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

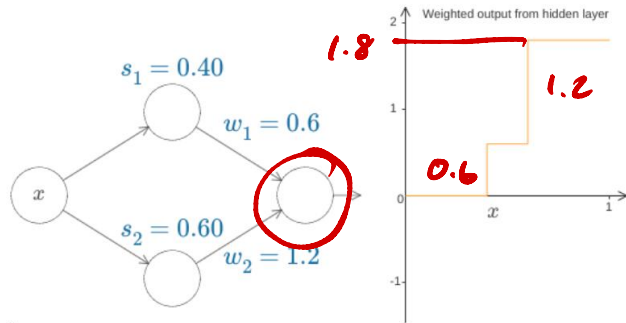
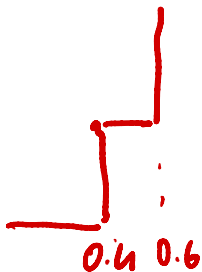


- Create a step at $x = -b/w$



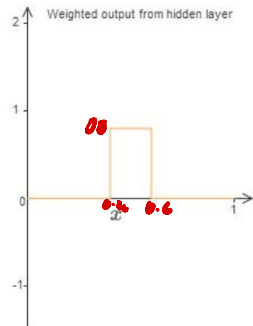
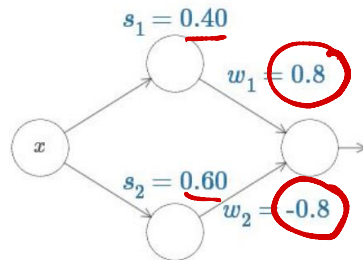
Universality

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



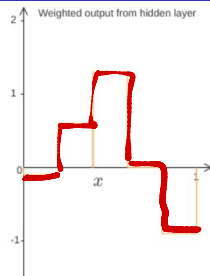
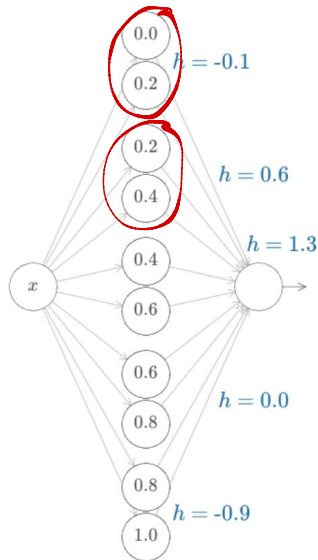
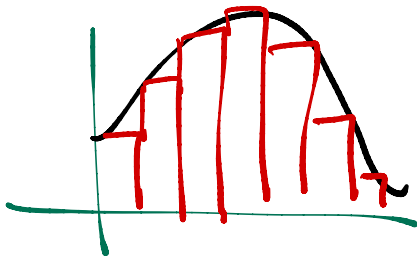
Universality

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



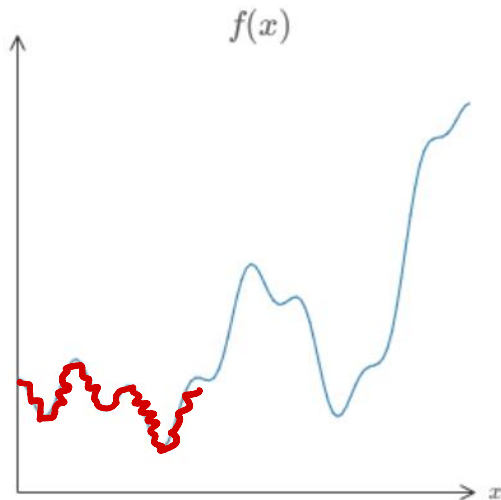
Universality

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



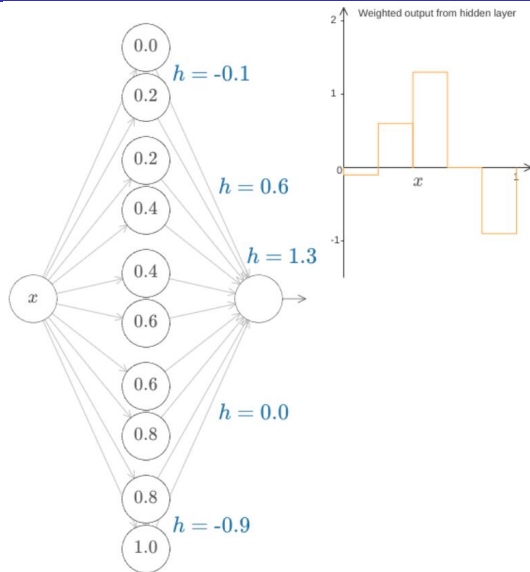
Universality

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



Universality

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

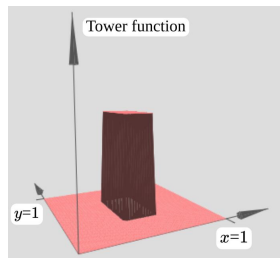


Non-linear activation

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

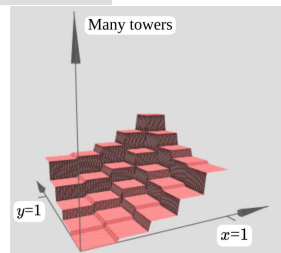
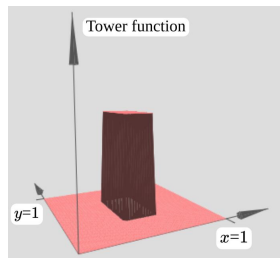
Non-linear activation

- 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

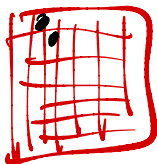
- MNIST data set
- 1000 samples of 10 handwritten digits
 - Assume input has been segmented



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



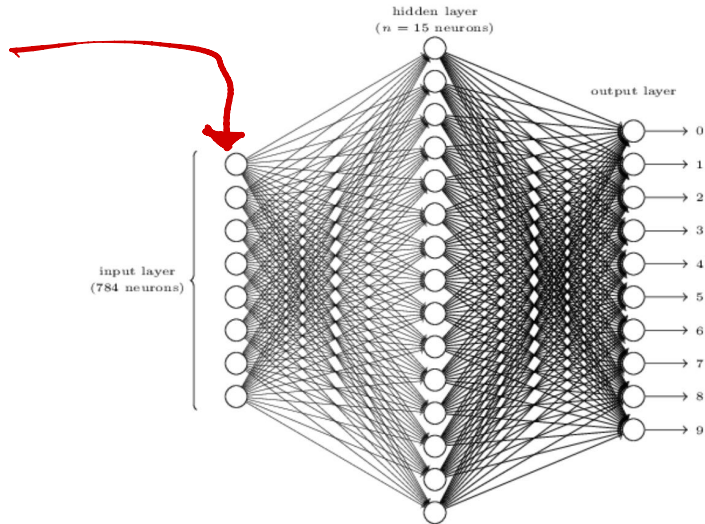
Example: Recognizing handwritten digits

- 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 $x = (x_1, x_2, \dots, x_{784})$



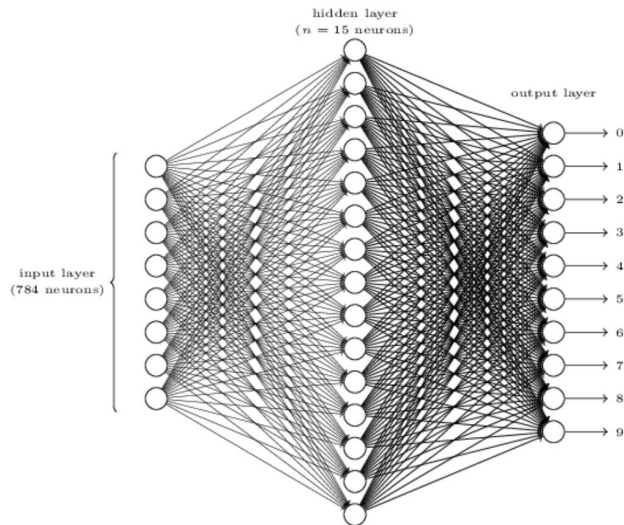
Example: Network structure

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



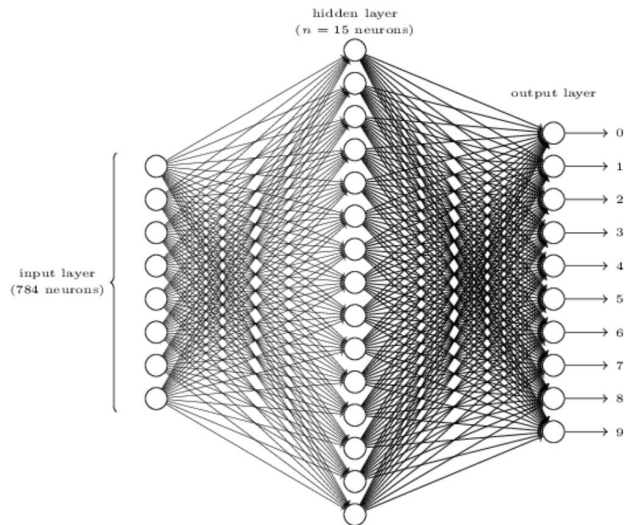
Example: Network structure

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



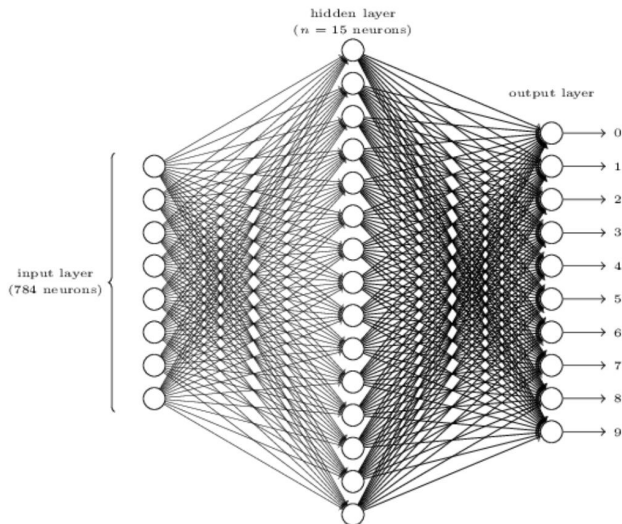
Example: Network structure

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



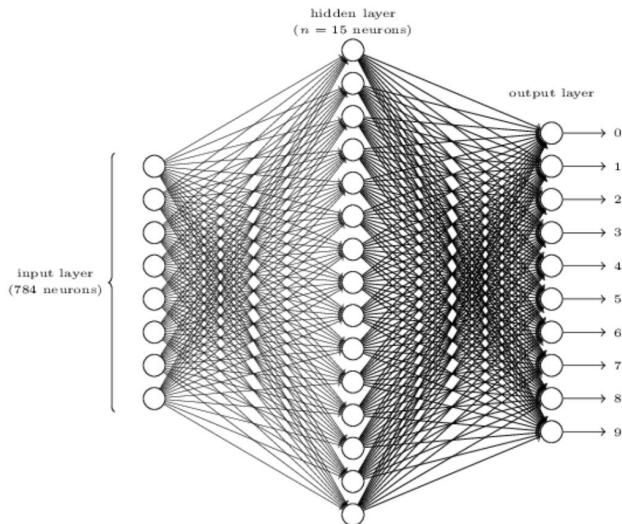
Example: Network structure

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



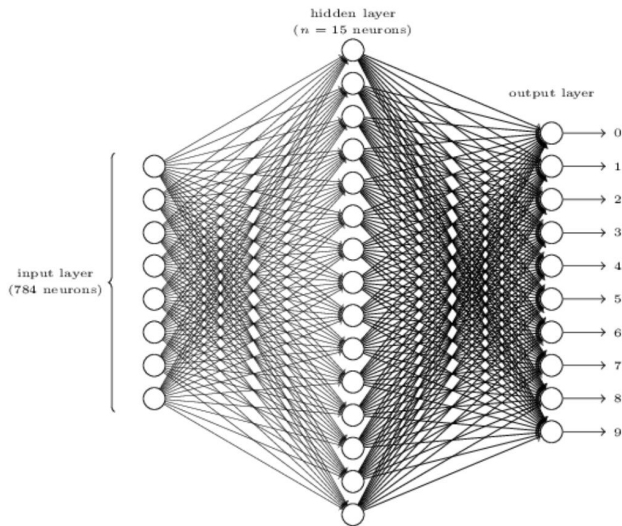
Example: Network structure

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



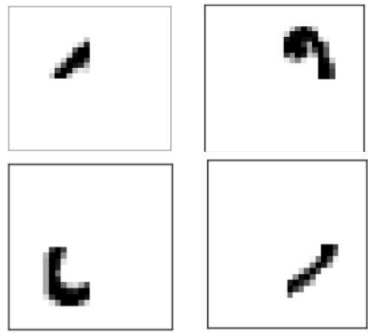
Example: Network structure

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



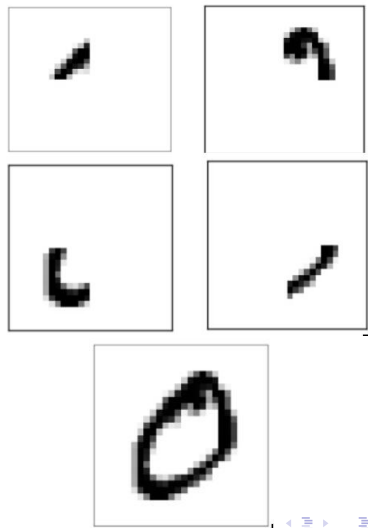
Example: Extracting features

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



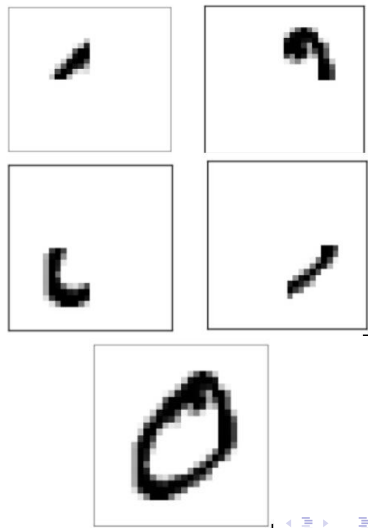
Example: Extracting features

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



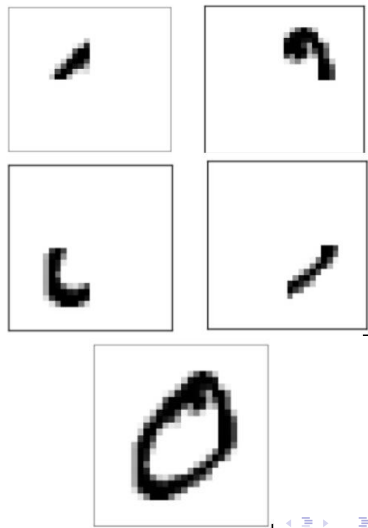
Example: Extracting features

- 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

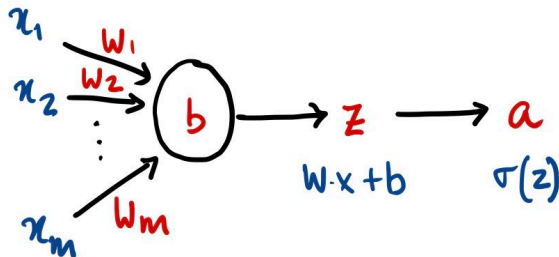


Neural networks

- Without loss of generality,
 - Assume the network is layered
 - All paths from input to output have the same length
 - Each layer is fully connected to the previous one
 - Set weight to 0 if connection is not needed

Neural networks

- Without loss of generality,
 - Assume the network is layered
 - All paths from input to output have the same length
 - Each layer is fully connected to the previous one
 - Set weight to 0 if connection is not needed
- Structure of an individual neuron
 - Input weights w_1, \dots, w_m , bias b , output z , activation value a

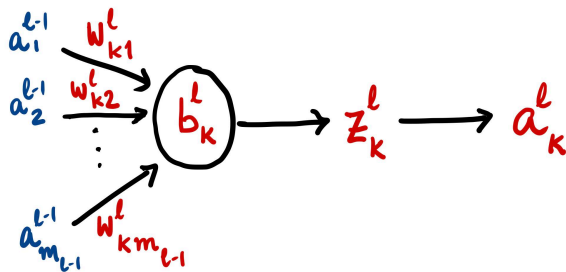
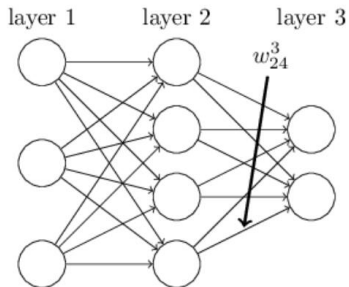


Notation

- Layers $\ell \in \{1, 2, \dots, L\}$
 - Inputs are connected first hidden layer, layer 1
 - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \dots, m_\ell$

Notation

- Layers $\ell \in \{1, 2, \dots, L\}$
 - Inputs are connected first hidden layer, layer 1
 - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \dots, m_\ell$
- Node k in layer ℓ has bias b_k^ℓ , output z_k^ℓ and activation value a_k^ℓ
- Weight on edge from node j in level $\ell-1$ to node k in level ℓ is w_{kj}^ℓ



- Why the inversion of indices in the subscript w_{kj}^l ?
 - $z_k^l = w_{k1}^l a_1^{l-1} + w_{k2}^l a_2^{l-1} + \dots + w_{km_{l-1}}^l a_{m_{l-1}}^{l-1}$
 - Let $\bar{w}_k^l = (w_{k1}^l, w_{k2}^l, \dots, w_{km_{l-1}}^l)$
and $\bar{a}^{l-1} = (a_1^{l-1}, a_2^{l-1}, \dots, a_{m_{l-1}}^{l-1})$
 - Then $z_k^l = \bar{w}_k^l \cdot \bar{a}^{l-1}$

- Why the inversion of indices in the subscript w_{kj}^l ?

- $z_k^l = w_{k1}^l a_1^{l-1} + w_{k2}^l a_2^{l-1} + \dots + w_{km_{l-1}}^l a_{m_{l-1}}^{l-1}$

- Let $\bar{w}_k^l = (w_{k1}^l, w_{k2}^l, \dots, w_{km_{l-1}}^l)$
and $\bar{a}^{l-1} = (a_1^{l-1}, a_2^{l-1}, \dots, a_{m_{l-1}}^{l-1})$

- Then $z_k^l = \bar{w}_k^l \cdot \bar{a}^{l-1}$

- Assume all layers have same number of nodes

- Let $m = \max_{\ell \in \{1, 2, \dots, L\}} m_\ell$

- For any layer i , for $k > m_i$, we set all of $w_{kj}^l, b_k^l, z_k^l, a_k^l$ to 0

- Matrix formulation

$$\begin{bmatrix} \bar{z}_1^l \\ \bar{z}_2^l \\ \dots \\ \bar{z}_m^l \end{bmatrix} = \begin{bmatrix} \bar{w}_1^l \\ \bar{w}_2^l \\ \dots \\ \bar{w}_m^l \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \dots \\ a_m^{l-1} \end{bmatrix}$$

Learning the parameters

- Need to find optimum values for all weights w_{kj}^l
- Use gradient descent
 - Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$

Learning the parameters

- Need to find optimum values for all weights w_{kj}^l
- Use gradient descent
 - Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$
- Assumptions about the cost function

Learning the parameters

- Need to find optimum values for all weights w_{kj}^{ℓ}
- Use gradient descent
 - Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^{\ell}}$, $\frac{\partial C}{\partial b_k^{\ell}}$
- Assumptions about the cost function
 - 1 For input \mathbf{x} , $C(\mathbf{x})$ is a function of only the output layer activation, a^L
 - For instance, for training input (\mathbf{x}_i, y_i) , sum-squared error is $(y_i - a_i^L)^2$
 - Note that \mathbf{x}_i, y_i are fixed values, only a_i^L is a variable

Learning the parameters

- Need to find optimum values for all weights w_{kj}^ℓ

- Use gradient descent

- Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$

- Assumptions about the cost function

- 1 For input \mathbf{x} , $C(\mathbf{x})$ is a function of only the output layer activation, a^L

- For instance, for training input (\mathbf{x}_i, y_i) , sum-squared error is $(y_i - a_i^L)^2$
- Note that \mathbf{x}_i, y_i are fixed values, only a_i^L is a variable

- 2 Total cost is average of individual input costs

- Each input \mathbf{x}_i incurs cost $C(\mathbf{x}_i)$, total cost is $\frac{1}{n} \sum_{i=1}^n C(\mathbf{x}_i)$
- For instance, mean sum-squared error $\frac{1}{n} \sum_{i=1}^n (y_i - a_i^L)^2$

Learning the parameters

- Assumptions about the cost function

- 1 For input \mathbf{x} , $C(\mathbf{x})$ is a function of only the output layer activation, a^L
- 2 Total cost is average of individual input costs

- With these assumptions:

- We can write $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$ in terms of individual $\frac{\partial a_i^l}{\partial w_{kj}^l}$, $\frac{\partial a_i^l}{\partial b_k^l}$
- Can extrapolate change in individual cost $C(\mathbf{x})$ to change in overall cost C — **stochastic gradient descent**

Learning the parameters

- Assumptions about the cost function

- 1 For input \mathbf{x} , $C(\mathbf{x})$ is a function of only the output layer activation, a^L
- 2 Total cost is average of individual input costs

- With these assumptions:

- We can write $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$ in terms of individual $\frac{\partial a_i^L}{\partial w_{kj}^\ell}$, $\frac{\partial a_i^L}{\partial b_k^\ell}$
- Can extrapolate change in individual cost $C(\mathbf{x})$ to change in overall cost C — **stochastic gradient descent**

- Complex dependency of C on w_{kj}^ℓ , b_k^ℓ

- Many intermediate layers
- Many paths through these layers

- Use **chain rule** to decompose into local dependencies

- $y = g(f(x)) \Rightarrow \frac{\partial g}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x}$