Lecture 19: 07 April, 2022

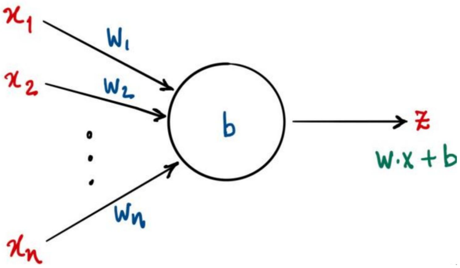
Madhavan Mukund

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

Data Mining and Machine Learning
January–May 2022

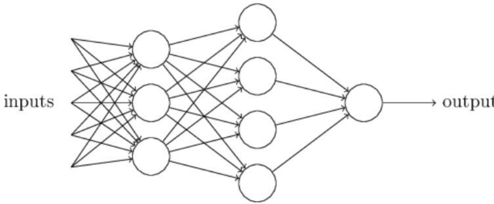
Linear separators and Perceptrons

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



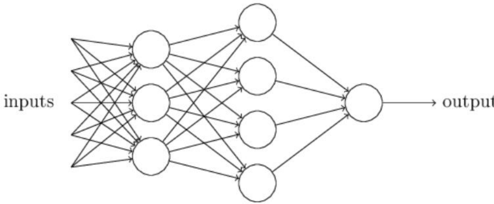
Linear separators and Perceptrons

- 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?



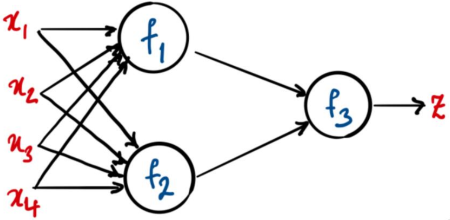
Linear separators and Perceptrons

- 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



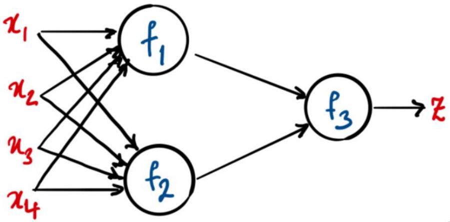
Linear separators and Perceptrons

- 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
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$



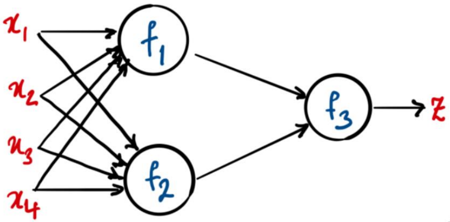
Linear separators and Perceptrons

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

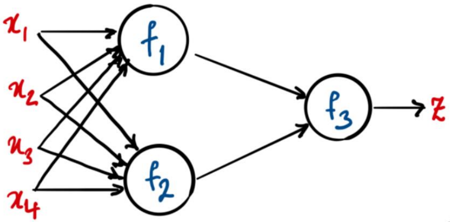


Linear separators and Perceptrons

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

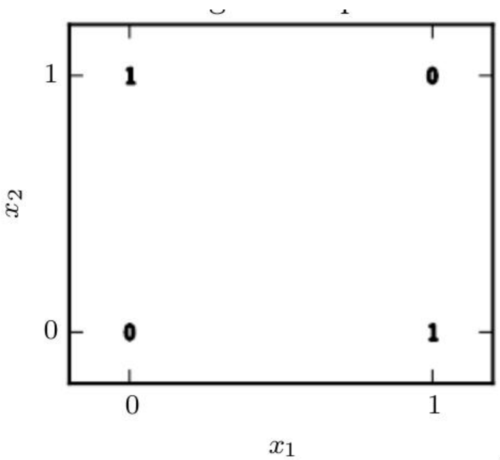


- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
  - $f_3 = w_3 \cdot \langle f_1,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

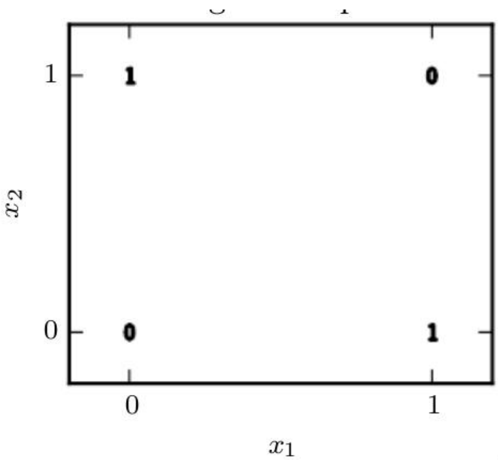


Limits of linearity

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

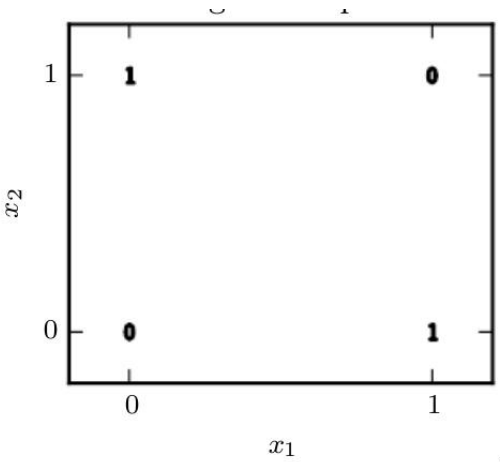


- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$

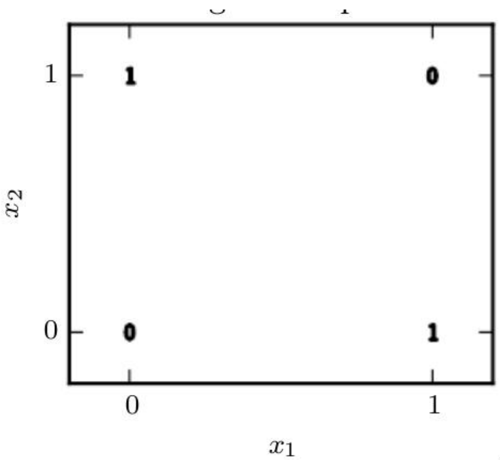


Limits of linearity

- 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 $u > 0$

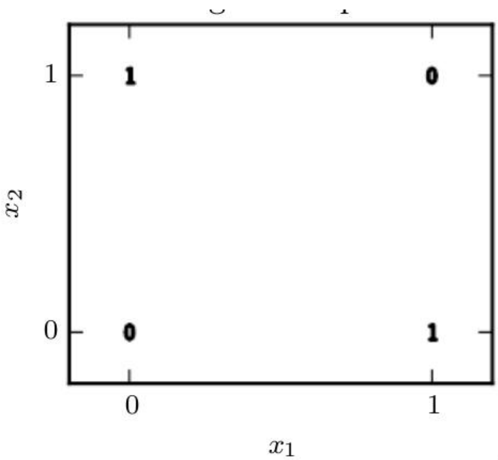


- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$
- $x_2 = 1$: As x_1 goes from 0 to 1, output goes from 1 to 0, so $u < 0$



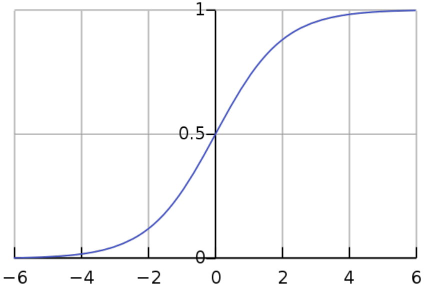
Limits of linearity

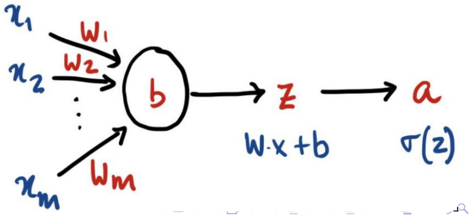
- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$
- $x_2 = 1$: As x_1 goes from 0 to 1, output goes from 1 to 0, so $u < 0$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



Non-linear activation

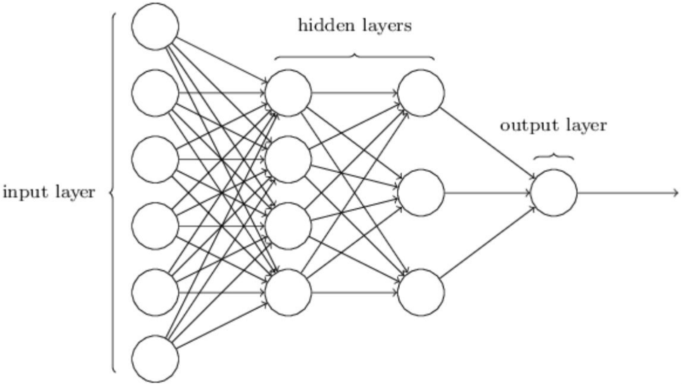
- Transform linear output z through a non-linear activation function
- Sigmoid function $\frac{1}{1+e^{-z}}$





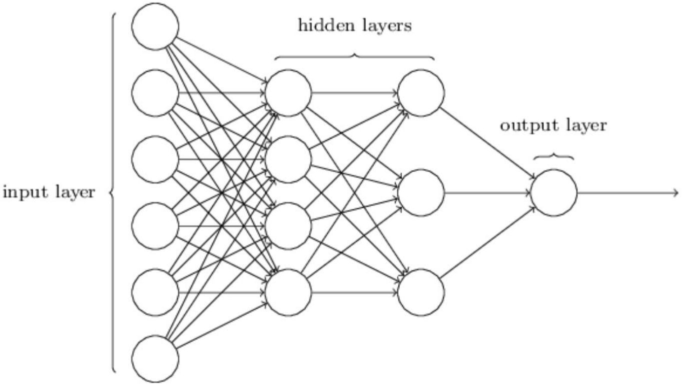
Structure of a neural network

- Acyclic
- Input layer, hidden layers, output layer



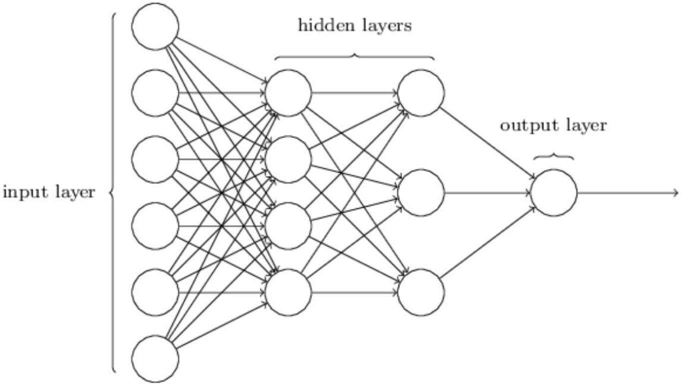
Structure of a neural network

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



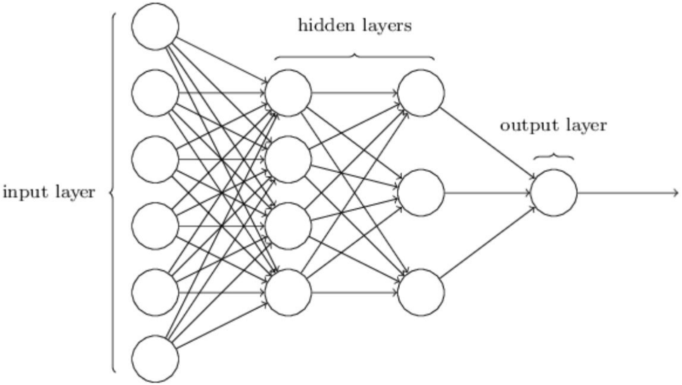
Structure of a neural network

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



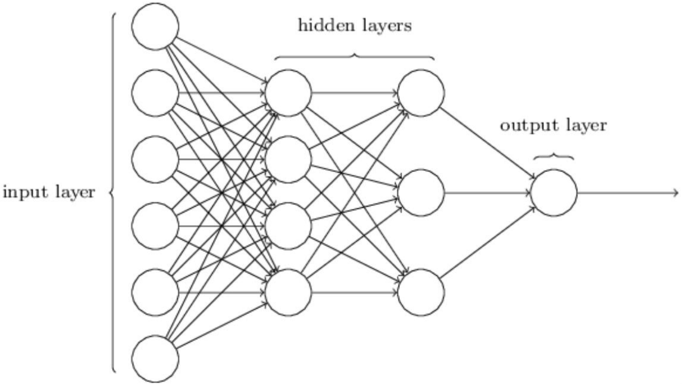
Structure of a neural network

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



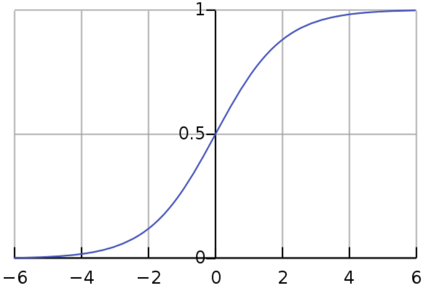
Structure of a neural network

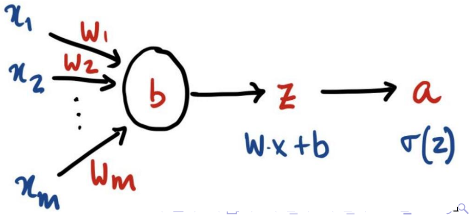
- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
  - Hidden neurons are arranged in layers
  - Each layer is fully connected to the next
  - Set weight to zero to remove an edge



Non-linear activation

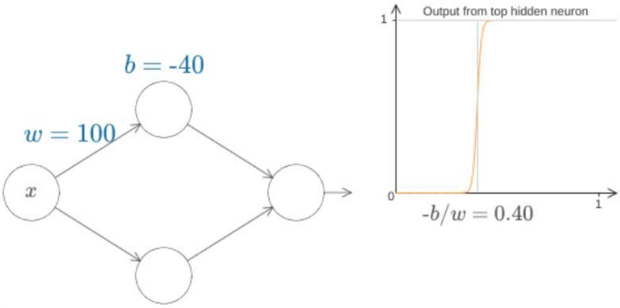
- Transform linear output z through a non-linear activation function
- Sigmoid function $\dfrac{1}{1+e^{-z}}$
- Step is at $z = 0$
  - $z = wx + b$, so step is at $x = -b/w$
  - Increasing w makes step steeper





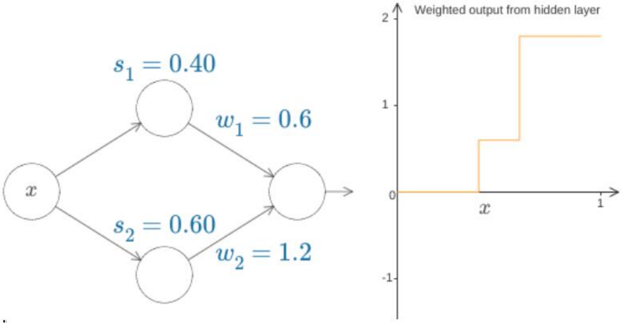
Universality

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



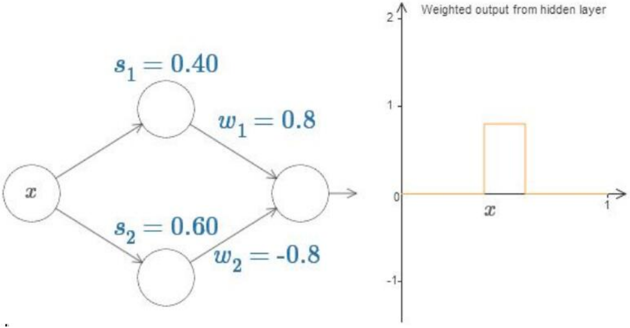
Universality

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



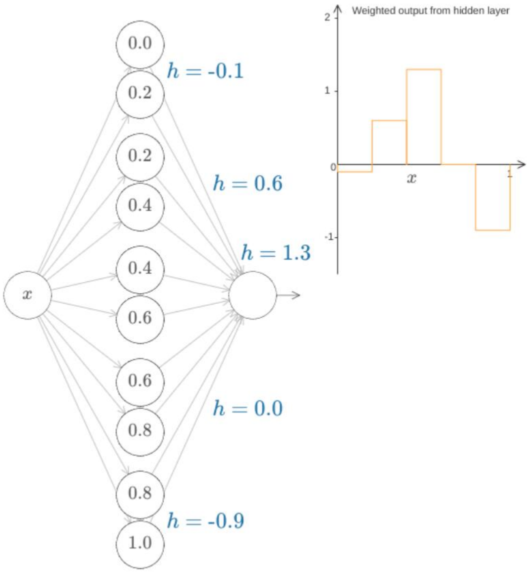
Universality

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



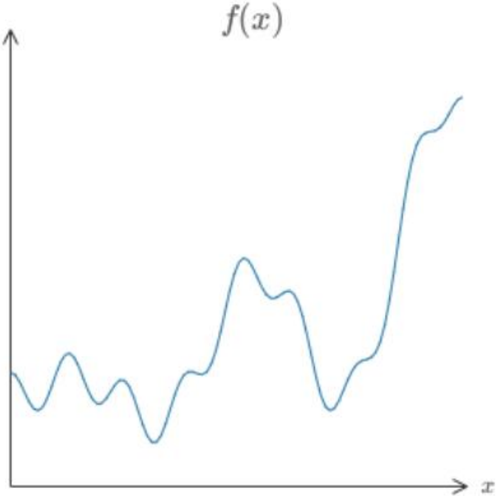
Universality

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



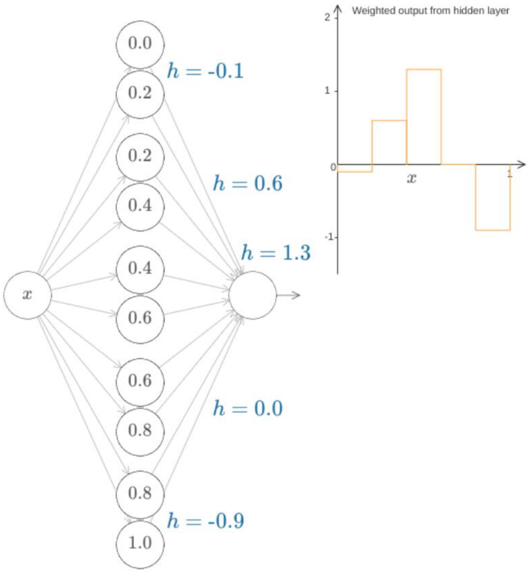
Universality

- 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!

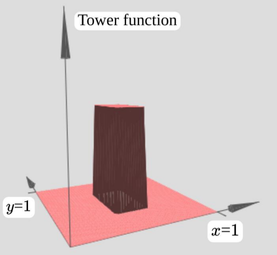


Non-linear activation

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

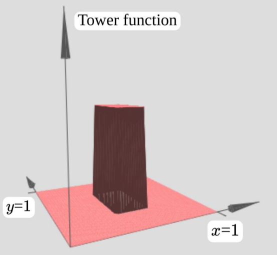
Non-linear activation

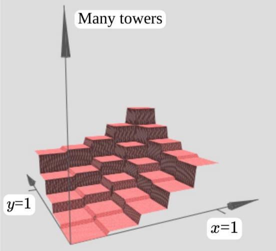
- With non-linear activation, network of neurons can approximate any function
  - Can build "rectangular" blocks



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

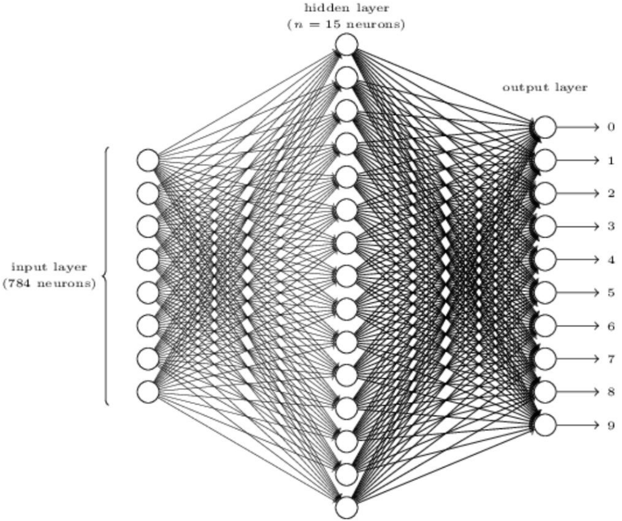
Example: Recognizing handwritten digits

- 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

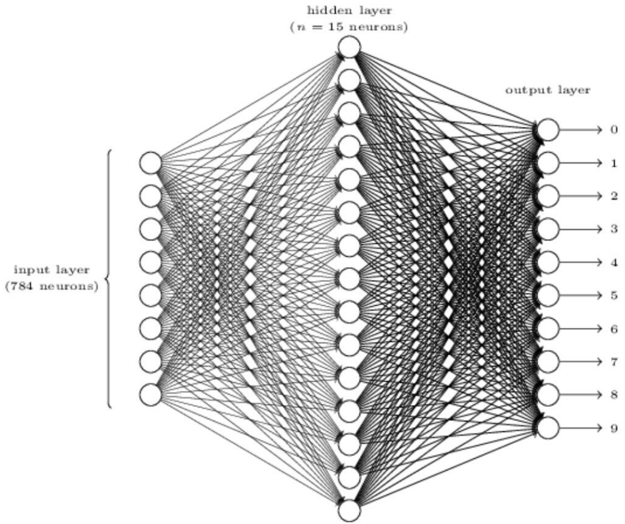
- MNIST data set
- 1000 samples of 10 handwritten digits
  - Assume input has been segmented
- Each digit is 28×28 pixels
  - Grayscale value, 0 to 1
  - 784 pixels
- Input $x = (x_1, x_2, \ldots, x_{784})$

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



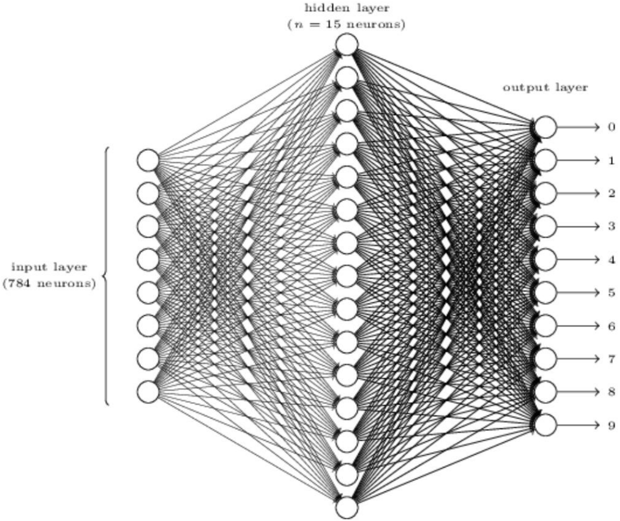
Example: Network structure

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



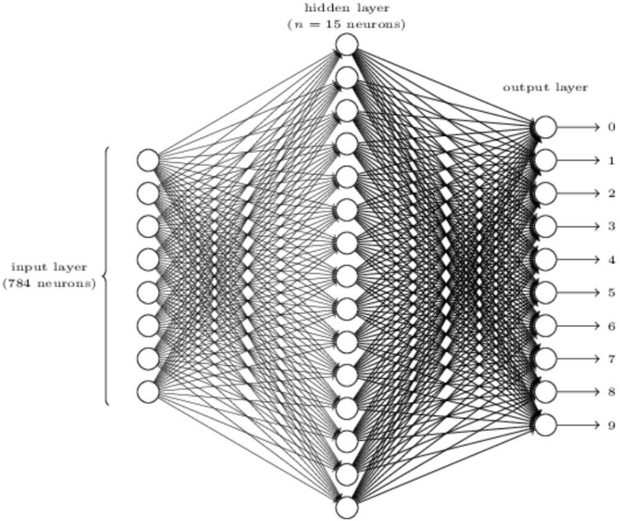
Example: Network structure

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



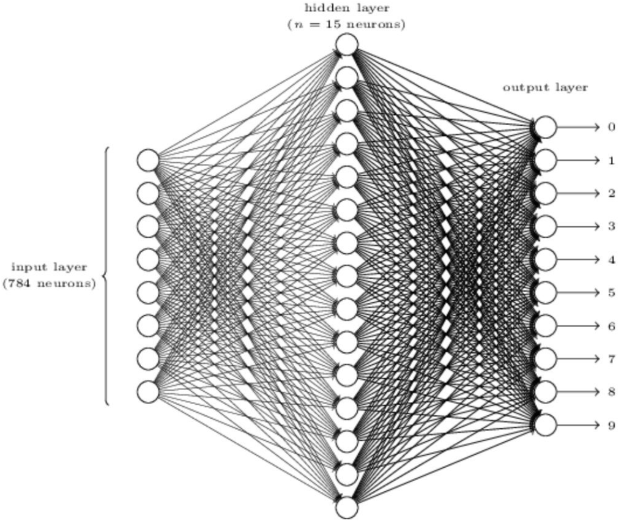
Example: Network structure

- 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



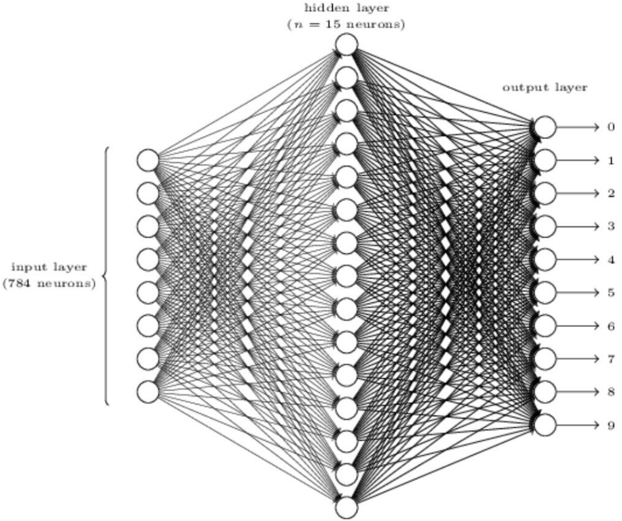
Example: Network structure

- 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, $\arg\max_j a_j$



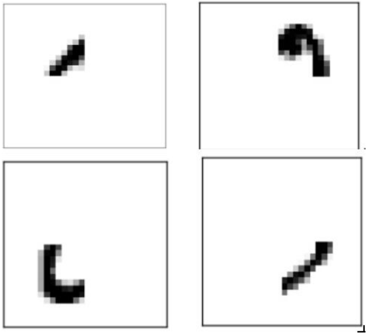
Example: Network structure

- 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, $\arg\max_j \frac{e^{a_j}}{\sum_j e^{a_j}}$
    - "Smooth" version of $\arg\max$



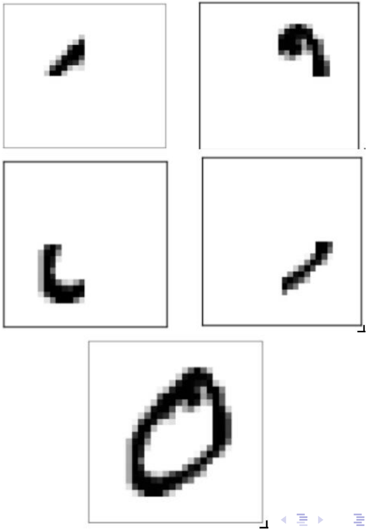
Example: Extracting features

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



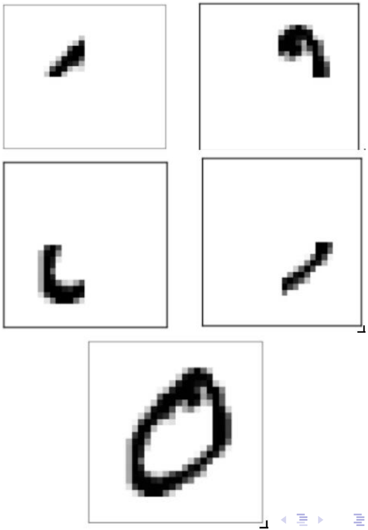
Example: Extracting features

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



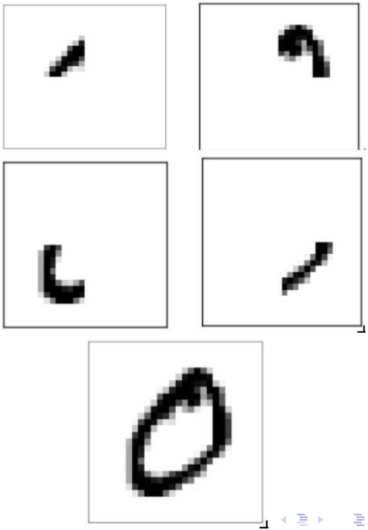
Example: Extracting features

- 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



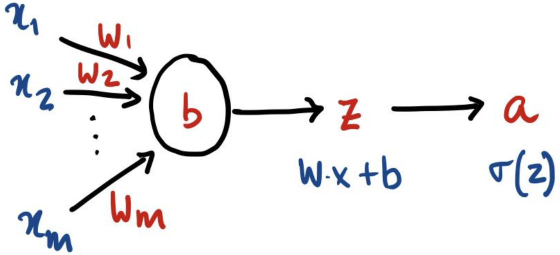
Neural networks

- 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 $w_1, \ldots, w_m$, bias b, output z, activation value a

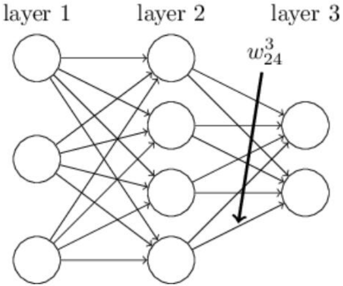


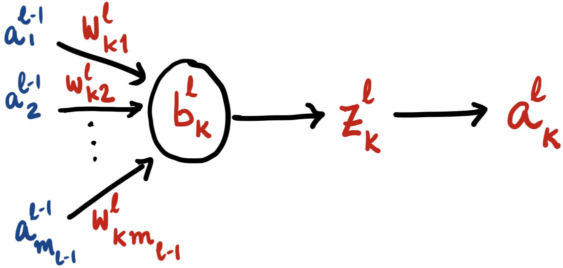
Notation

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

Notation

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





Notation

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

Notation

- Why the inversion of indices in the subscript w^{ℓ}_{kj}?
  - $z^{\ell}_k = w^{\ell}_{k1} a^{\ell-1}_1 + w^{\ell}_{k2} a^{\ell-1}_2 + \cdots + w^{\ell}_{km_{\ell-1}} a^{\ell-1}_{m_{\ell-1}}$
  - Let $\overline{w}^{\ell}_k = (w^{\ell}_{k1}, w^{\ell}_{k2}, \ldots, w^{\ell}_{km_{\ell-1}})$ and $\overline{a}^{\ell-1} = (a^{\ell-1}_1, a^{\ell-1}_2, \ldots, a^{\ell-1}_{m_{\ell-1}})$
  - Then $z^{\ell}_k = \overline{w}^{\ell}_k \cdot \overline{a}^{\ell-1}$
- Assume all layers have same number of nodes
  - Let $m = \max\limits_{\ell \in \{1.2,\ldots,L\}} m_\ell$
  - For any layer i, for $k > m_i$, we set all of $w^{\ell}_{kj}, b^{\ell}_k, z^{\ell}_k, a^{\ell}_k$ to 0
- Matrix formulation

$$\begin{bmatrix} \overline{z}^{\ell}_1 \\ \overline{z}^{\ell}_2 \\ \cdots \\ \overline{z}^{\ell}_m \end{bmatrix} = \begin{bmatrix} \overline{w}^{\ell}_1 \\ \overline{w}^{\ell}_2 \\ \cdots \\ \overline{w}^{\ell}_m \end{bmatrix} \begin{bmatrix} a^{\ell-1}_1 \\ a^{\ell-1}_2 \\ \cdots \\ a^{\ell-1}_m \end{bmatrix}$$

Learning the parameters

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

Learning the parameters

- Need to find optimum values for all weights w^{ℓ}_{kj}
- Use gradient descent
  - Cost function C, partial derivatives $\frac{\partial C}{\partial w^{\ell}_{kj}}, \frac{\partial C}{\partial b^{\ell}_{k}}$
- 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 $\dfrac{\partial C}{\partial w_{kj}^{\ell}}, \dfrac{\partial C}{\partial b_k^{\ell}}$
- 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^ℓ_{kj}
- Use gradient descent
  - Cost function C, partial derivatives $\frac{\partial C}{\partial w^\ell_{kj}}, \frac{\partial C}{\partial b^\ell_k}$
- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
     - For instance, for training input $(\boldsymbol{x}_i, y_i)$, sum-squared error is $(y_i - a^L_i)^2$
     - Note that $\boldsymbol{x}_i$, y_i are fixed values, only a^L_i is a variable
  2. Total cost is average of individual input costs
     - Each input $\boldsymbol{x}_i$ incurs cost $C(\boldsymbol{x}_i)$, total cost is $\frac{1}{n}\sum_{i=1}^{n} C(\boldsymbol{x}_i)$
     - For instance, mean sum-squared error $\frac{1}{n}\sum_{i=1}^{n}(y_i - a^L_i)^2$

Learning the parameters

- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ 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 $\dfrac{\partial C}{\partial w_{kj}^{\ell}}, \dfrac{\partial C}{\partial b_k^{\ell}}$ in terms of individual $\dfrac{\partial a_i^L}{\partial w_{kj}^{\ell}}, \dfrac{\partial a_i^L}{\partial b_k^{\ell}}$
  - Can extrapolate change in individual cost $C(x)$ 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 $\dfrac{\partial C}{\partial w^\ell_{kj}}, \dfrac{\partial C}{\partial b^\ell_k}$ in terms of individual $\dfrac{\partial a^L_i}{\partial w^\ell_{kj}}, \dfrac{\partial a^L_i}{\partial b^\ell_k}$
  - Can extrapolate change in individual cost $C(x)$ to change in overall cost C — stochastic gradient descent
- Complex dependency of C on w^ℓ_{kj}, b^ℓ_k
  - Many intermediate layers
  - Many paths through these layers
- Use chain rule to decompose into local dependencies
  - $y = g(f(x)) \Rightarrow \dfrac{\partial g}{\partial x} = \dfrac{\partial g}{\partial f}\dfrac{\partial f}{\partial x}$