COMP250: Artificial Intelligence

10: Deep learning

Neural networks

Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains

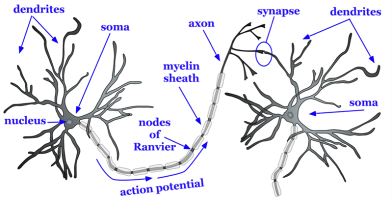
Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains
- Idea has been around since the 1950s

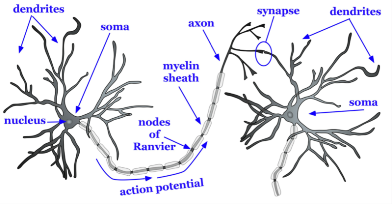
Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains
- Idea has been around since the 1950s
- Recent resurgence of interest: today's powerful CPUs and GPUs allow much larger ANNs to be used

Real neurons

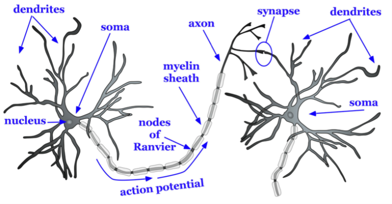


Real neurons



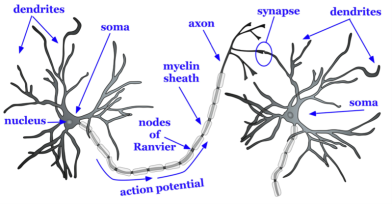
- An **electrically excitable** cell

Real neurons



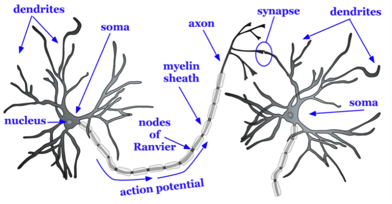
- An **electrically excitable** cell
- Neurons are **connected together**

Real neurons



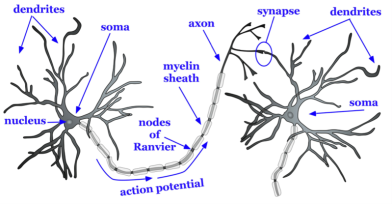
- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**

Real neurons



- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**
- If enough excitatory signals are received, the neuron **fires** — sends an electrical signal to the connected neurons

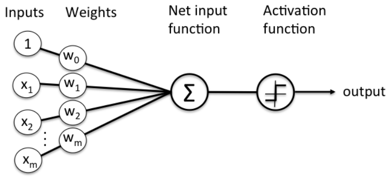
Real neurons



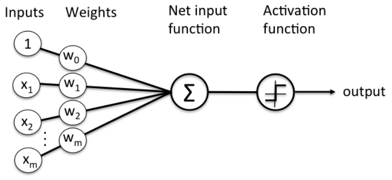
- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**

- If enough excitatory signals are received, the neuron **fires** — sends an electrical signal to the connected neurons
- Human brain contains approximately **100 billion** neurons

An artificial neuron

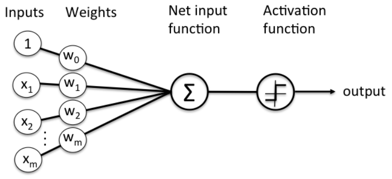


An artificial neuron



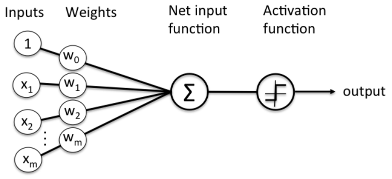
- A **perceptron**

An artificial neuron



- A **perceptron**
- Inputs $x_1, \ldots, x_m$ are outputs from **other perceptrons**

An artificial neuron



- A **perceptron**
- Inputs $x_1, \ldots, x_m$ are outputs from **other perceptrons**
- Each input has a **weight** w_i between -1 and $+1$

Perceptron activation

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

- This goes through an **activation function**

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

- This goes through an **activation function**
- Simplest: **step function**

$$\textbf{output} = \begin{cases} 1 & \text{if sum} \geq \text{threshold} \\ 0 & \text{if sum} < \text{threshold} \end{cases}$$

Perceptron activation

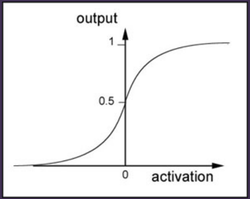
- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

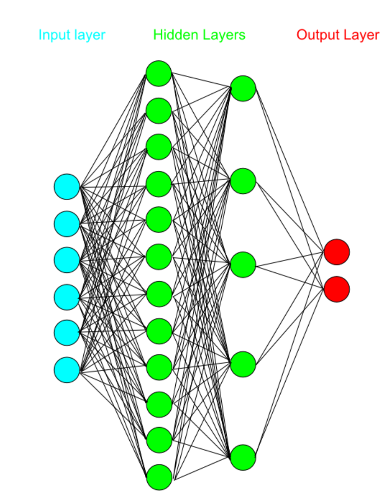
- This goes through an **activation function**
- Simplest: **step function**

$$\textbf{output} = \begin{cases} 1 & \text{if sum} \geq \text{threshold} \\ 0 & \text{if sum} < \text{threshold} \end{cases}$$

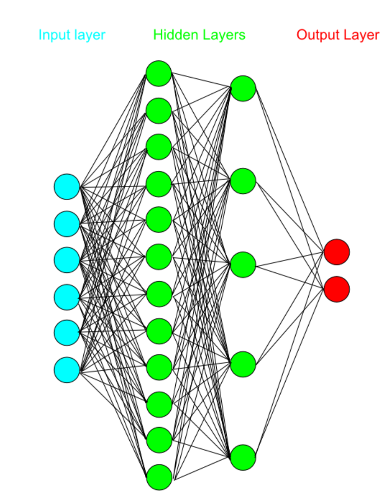
- More common: **sigmoid function**



An artificial neural network

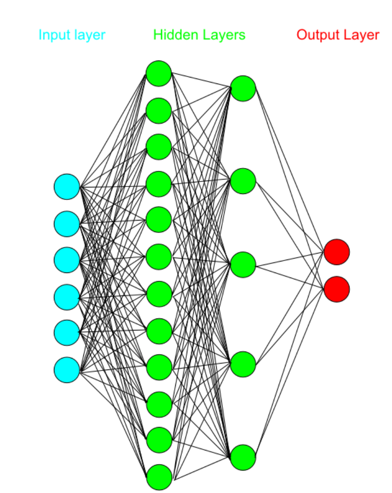


An artificial neural network



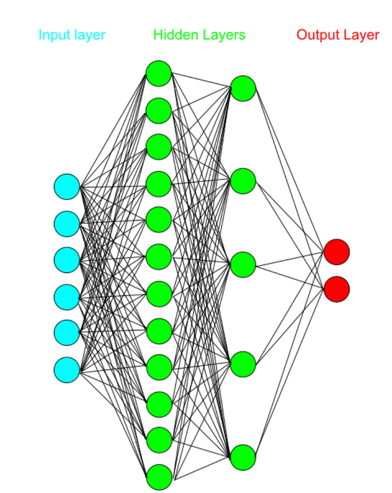
- A **multilayer perceptron (MLP)**

An artificial neural network



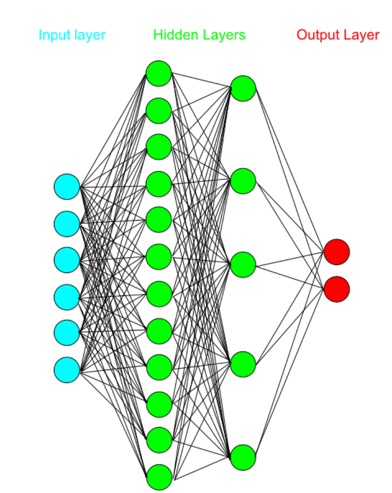
- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**

An artificial neural network



- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**
- Each layer is an array of **perceptrons**

An artificial neural network



- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**
- Each layer is an array of **perceptrons**
- Each perceptron's output is connected to **every** perceptron in the next layer

Image classification

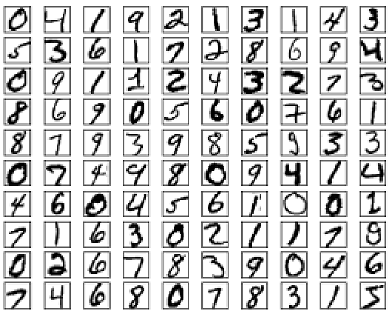
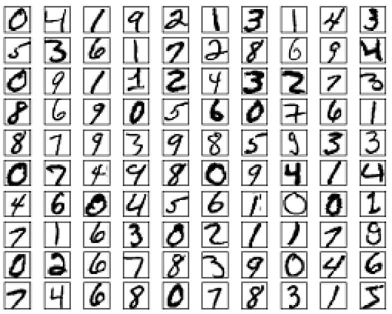
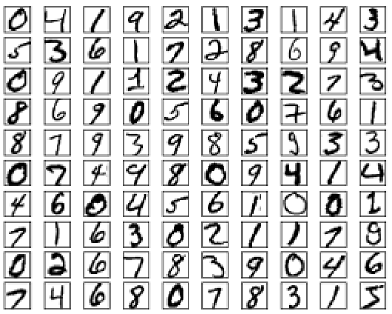


Image classification



- Classic example: **handwritten digit recognition**

Image classification



- Classic example: **handwritten digit recognition**
- Given a **raster image**, which of the digits 0 to 9 does it represent?

https://twitter.com/NaughtThought/status/846262063827730432

MLPs for image classification

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???
  - Parameters to tune

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???
  - Parameters to tune
- **Weights**: ???

How to set the weights?

How to set the weights?

- We need to **train** the network

How to set the weights?

- We need to **train** the network
- Idea:

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights
  - Repeat until a desired **accuracy** is obtained

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights
  - Repeat until a desired **accuracy** is obtained
- Note: this requires a large amount of training data that is **tagged**, i.e. for which we already know the correct answer

Stochastic gradient descent

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**
- Each epoch uses a randomly sampled **subset** of the training data

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**
- Each epoch uses a randomly sampled **subset** of the training data
- This reduces computation time, and helps to escape local optima

ANN example

http://playground.tensorflow.org

Overfitting

Overfitting

- ANN learns **patterns** in the training data

Overfitting

- ANN learns **patterns** in the training data
- Insufficient training data might result in the network learning "patterns" that are actually random anomalies

Deep learning

Deep learning

Deep learning

- Basically, the use of large ANNs with **many layers**

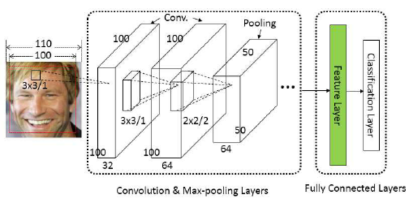
Deep learning

- Basically, the use of large ANNs with **many layers**
- Often uses **large training sets**

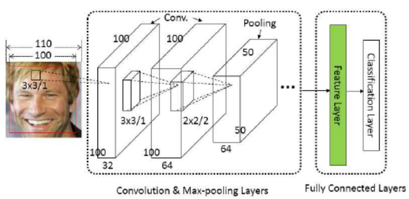
Deep learning

- Basically, the use of large ANNs with **many layers**
- Often uses **large training sets**
- Training often uses powerful **GPUs** — many times faster than training on the CPU

Convolutional Neural Networks (ConvNets)

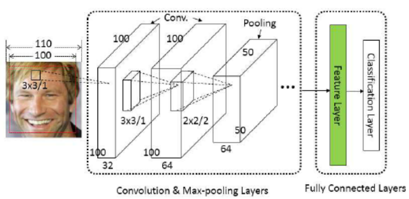


Convolutional Neural Networks (ConvNets)



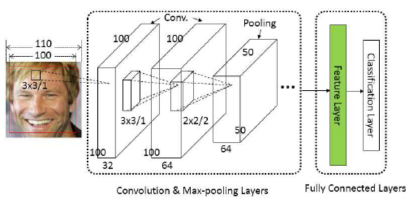
- Layers are **2D arrays**

Convolutional Neural Networks (ConvNets)

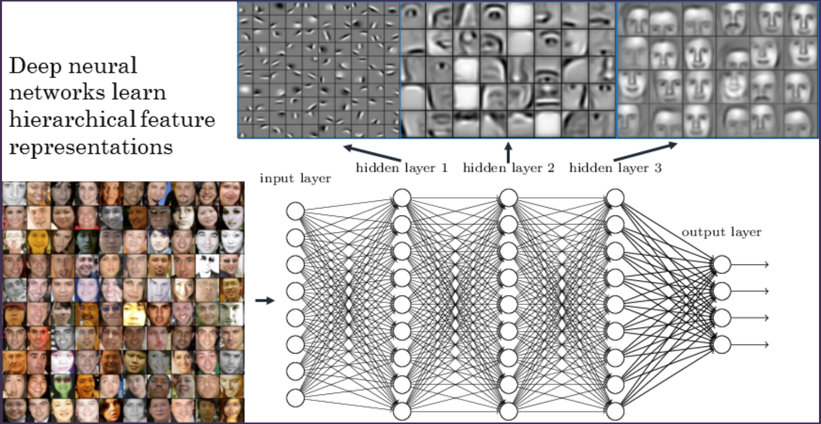


- Layers are **2D arrays**
- Neurons in convolutional layers are only connected to nearby neurons

Convolutional Neural Networks (ConvNets)



- Layers are **2D arrays**
- Neurons in convolutional layers are only connected to nearby neurons
- There are also fully connected layers


Deep neural
networks learn
hierarchical feature
representations
input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

DeepDream

DeepDream

- Train a ConvNet to recognise something (e.g. faces, objects, animals)

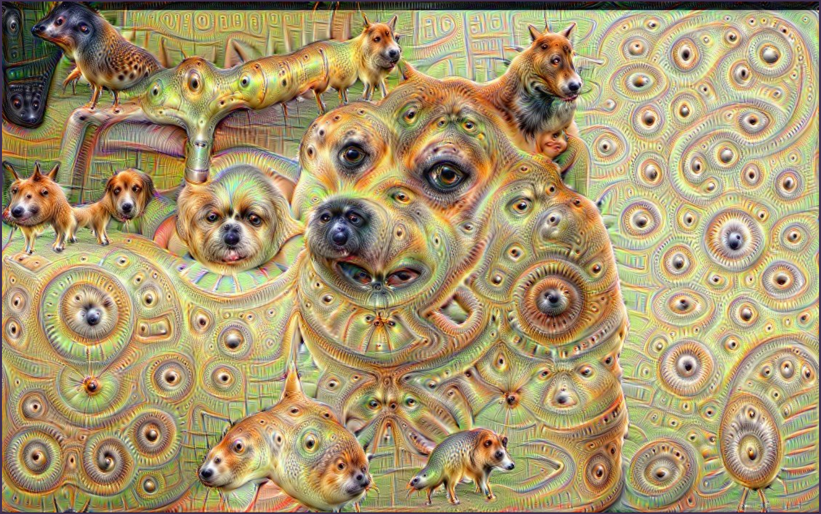
DeepDream

- Train a ConvNet to recognise something (e.g. faces, objects, animals)
- Run the network in “reverse”

DeepDream

- Train a ConvNet to recognise something (e.g. faces, objects, animals)
- Run the network in “reverse”
  - Adjust the image (e.g. via gradient ascent) so that it is more strongly recognised by the network

DeepDream



Style transfer

Style transfer

- Train a ConvNet to recognise a particular artistic style

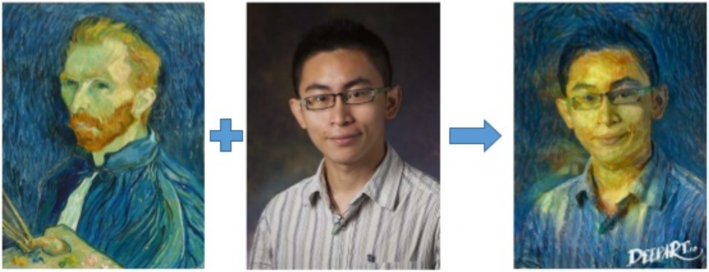
Style transfer

- Train a ConvNet to recognise a particular artistic style
- Run the network in “reverse” on an input image

Style transfer

- Train a ConvNet to recognise a particular artistic style
- Run the network in “reverse” on an input image
  - Adjust the image (e.g. via gradient ascent) so that it is more strongly recognised by the network

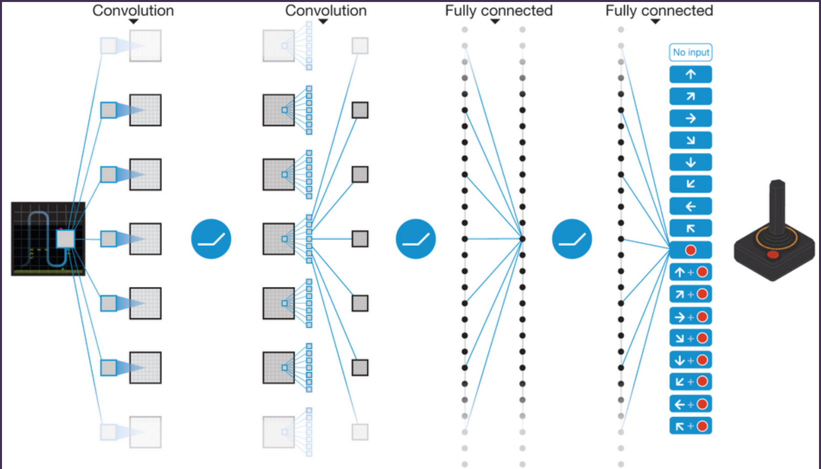
Style transfer



Source image (**Style**) | Target image (**Content**) | Output (deepart)

A Neural Algorithm of Artistic Style [Gatys et al. 2015]

Learning to play Atari games (Mnih et al, 2015)



FALMOUTH UNIVERSITY

Deep learning for PCG

https://www.youtube.com/watch?v=3wcpLwvBTYo