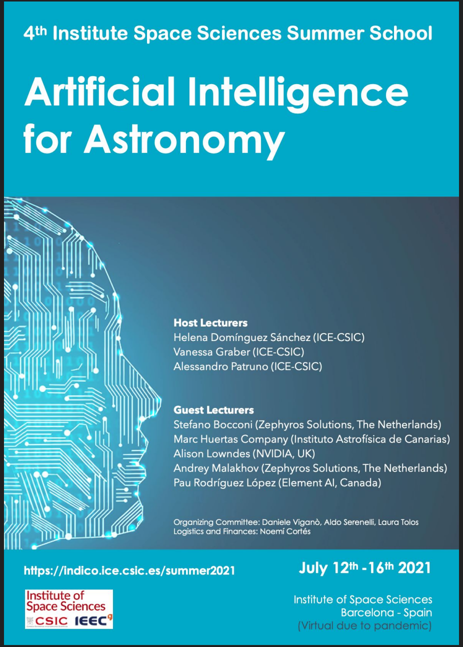

4th Institute Space Sciences Summer School

Artificial Intelligence for Astronomy

Host Lecturers
Helena Domínguez Sánchez (ICE-CSIC)
Vanessa Graber (ICE-CSIC)
Alessandro Patruno (ICE-CSIC)

Guest Lecturers
Stefano Bocconi (Zephyros Solutions, The Netherlands)
Marc Huertas Company (Instituto Astrofísica de Canarias)
Alison Lowndes (NVIDIA, UK)
Andrey Malakhov (Zephyros Solutions, The Netherlands)
Pau Rodríguez López (Element AI, Canada)

Organizing Committee: Daniele Viganò, Aldo Serenelli, Laura Tolos
Logistics and Finances: Noemí Cortés

https://indico.ice.csic.es/summer2021

July 12th - 16th 2021

Institute of Space Sciences
Barcelona - Spain
(Virtual due to pandemic)

Institute of Space Sciences CSIC IEEC

4th Institute of Space Sciences Summer School

AI for Astronomy

DEEP LEARNING (DL) & NEURAL NETWORKS (NNs)

Theory Day 2 - July 13th, 2021

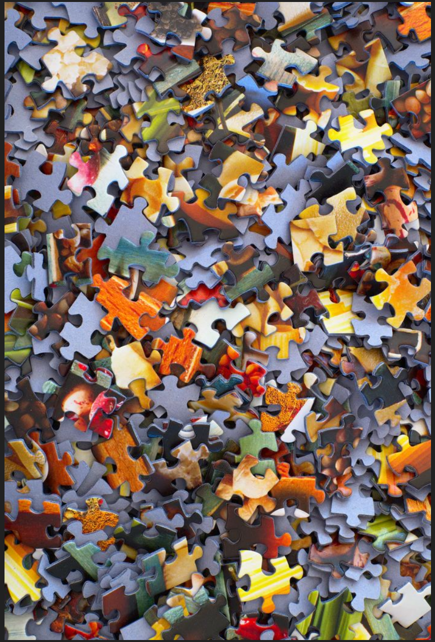
Vanessa Graber - graber@ice.csic.es

Institute of Space Sciences

CSIC IEEC

Learning outcomes:

- Learn about some of the **success stories** of deep learning and why this approach is so successful & popular right now.
- Get familiar with the field's **basic terminology**.
- Provide a **basis** for understanding the remaining lectures this week and applying deep learning in your own research.
- Understand the **theoretical background** of forward and backward propagation; the latter will form the basis for the second hands-on session this afternoon.


Credit: HP Gauster / Unsplash

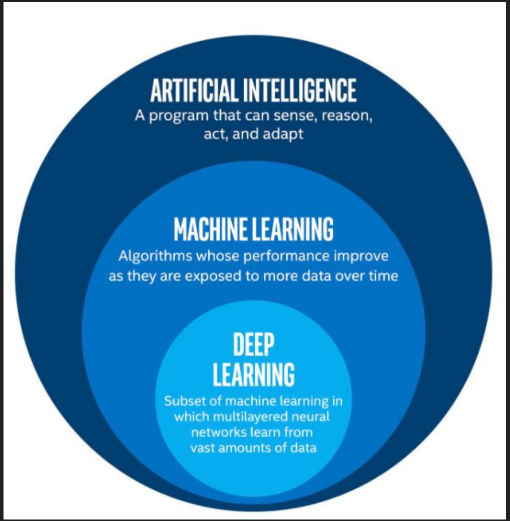
(A few) references:

- Online:
  - 3Blue1Brown: DL videos https://tinyurl.com/245cnnk8
  - reddit: https://www.reddit.com/r/MachineLearning/
- Books:
  - DL Illustrated: A Visual, Interactive Guide to AI by J. Krohn
  - DL by I. Goodfellow, Y. Bengio, and A. Courville
- Courses:
  - Elements of AI: ML basics https://www.elementsofai.com/
  - freeCodeCamp: DL with PyTorch https://tinyurl.com/3bs9ez3c
  - Coursera: ML by A. Ng https://tinyurl.com/4d6yx988
  - Udacity: Intro to DL with PyTorch https://tinyurl.com/4anysuwn

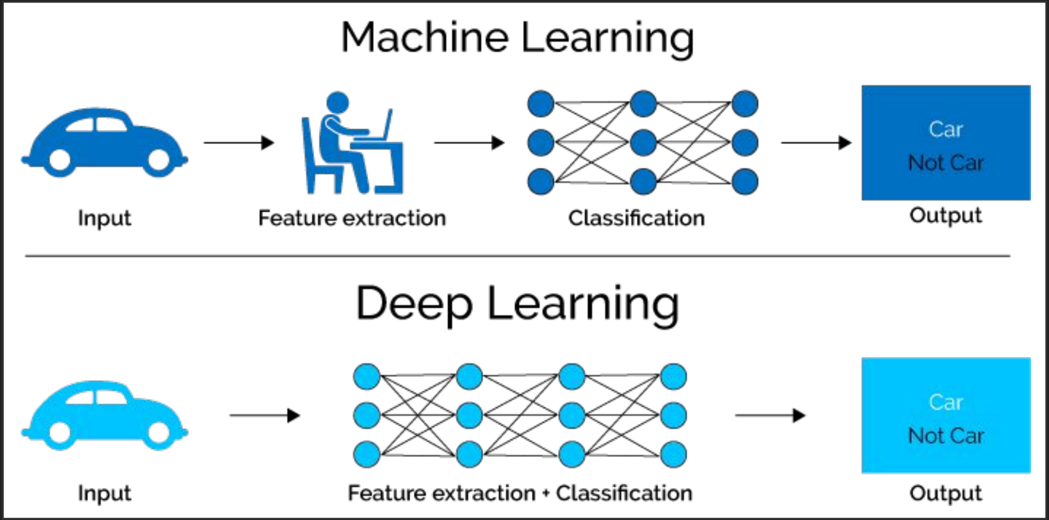
Overview:

1.1 What is DL? I

- DL is a **subfield of AI** and **ML**, focusing on using **multi-layered neural networks** to learn from large datasets. Different to standard ML approaches, DL does not require *external* **feature engineering**.


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

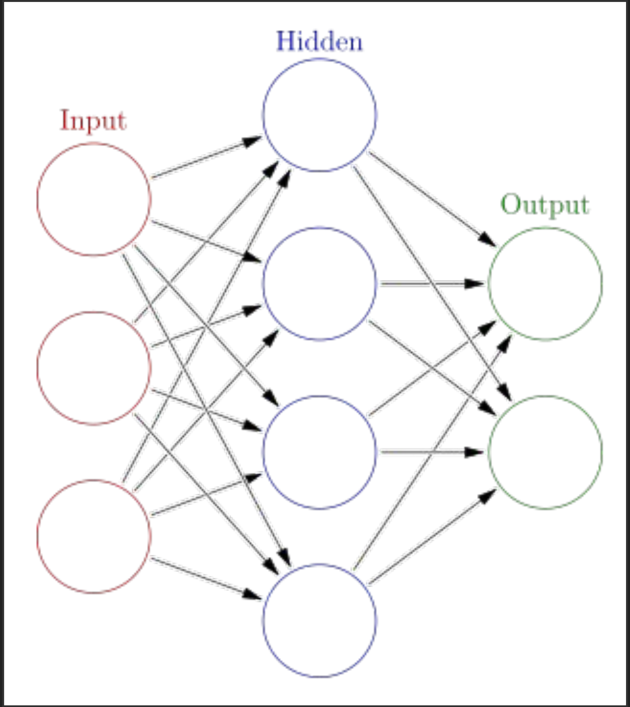
Credit: www.bigdata-insider.de



Credit: Acheron Analytics

1.1 What is DL? II

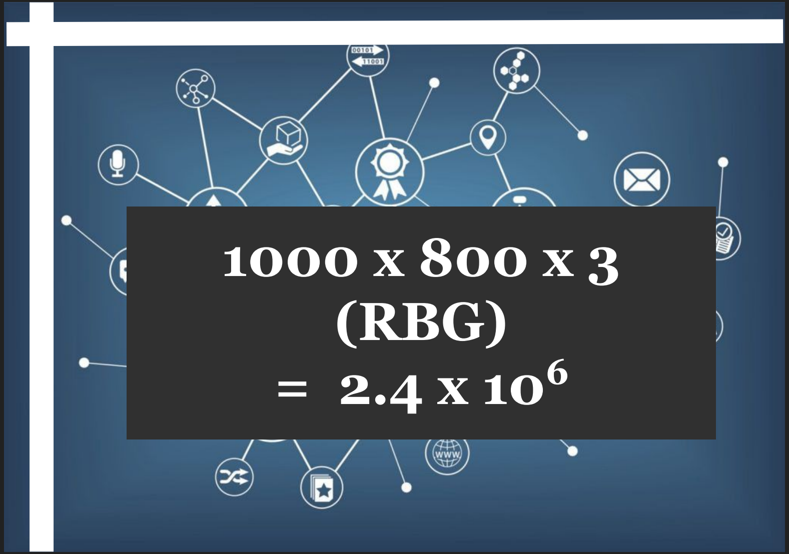
- A neural network is composed of **layers** (stacks of **neurons**). *Deep* in DL refers to the fact that **many hidden** layers are present between input and output layer.
- Each layer encodes a **simplified representation** of the input data.
- In analogy with **biological brains**, a DL algorithm **learns more and more** about the input as the data is passed through successive network layers; often in a **feedforward manner**.



Credit: Wikimedia Commons

1.2 Why DL?

- Recognising features in a **hierarchical way** allows deep NNs to model **complex non-linear relationships** for **large** input data.
- This makes DL powerful when working with **unstructured data** such as images, where the number of features / pixel can easily exceed millions.
- **BUT** (i) for successful application DL requires vast amounts of data; (ii) DL algorithms are more diffi-cult to interpret than other ML ones.



Credit: iconimage

1.3 Success stories: Computer vision I

- Computer vision aims at giving computers the ability to **gain understanding** from **digital images** very much like we do, to automate tasks such as image processing, analysing and then decision making.
- Such algorithms have a **wide range of applicability** including medicine, autonomous driving or surveillance.
- Early NN approaches were strongly influenced by **research in neurophysiology & signal processing** and modelled on biological vision.

Credit: @teenybiscuit

1.3 Success stories: Computer vision I

- Computer vision aims at giving computers the ability to **gain under-standing** from **digital images** very much like we do, to automate tasks such as image processing, analysing and then decision making.
- Such algorithms have a **wide range of applicability** including medicine, autonomous driving or surveillance.
- Early NN approaches were strongly influenced by **research in neuro-physiology & signal processing** and modelled on biological vision.

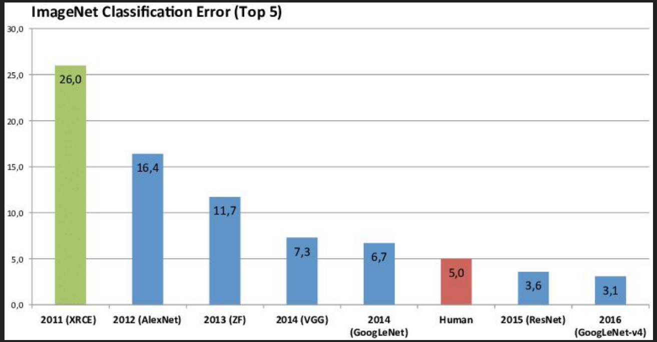
Credit: CLIPAREA.com

1.3 Success stories: Computer vision II

- The **first multi-layered NNs** include Neocognitron (1979), LeNet-5 (1998), and AlexNet (2012). NNs got/get better and better due to new computational techniques, hardware advances and larger datasets.

- **ImageNet** is a database with >14 million hand-annotated images separated into >20,000 categories.

Credit: karpathy.github.io/



Credit: Jason Chuang

- ILSVRC highlighted that DL (in particular CNNs) **outperforms standard ML approaches** and even humans.

1.3 Success stories: NLP I

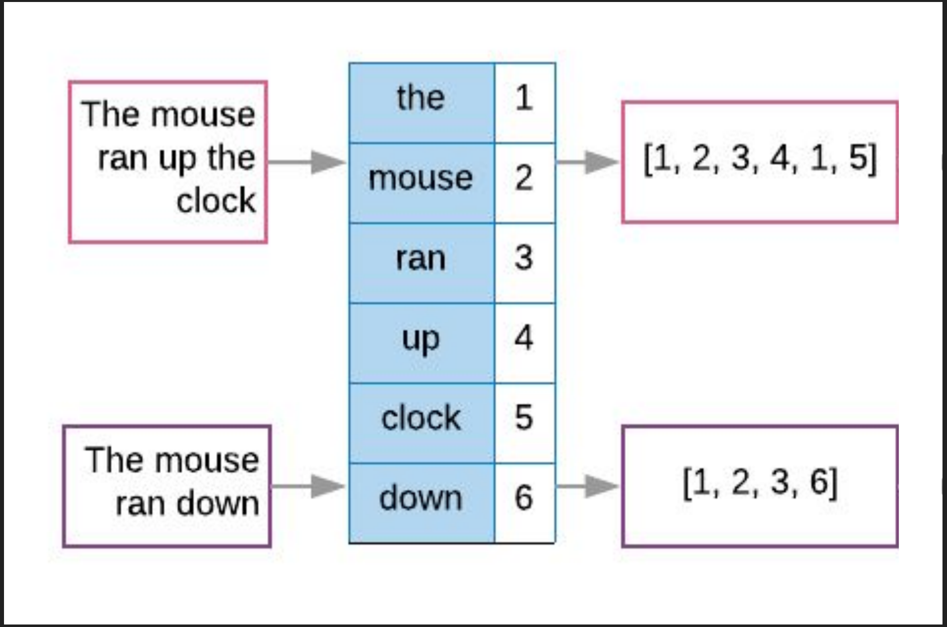
- **Natural language processing**: get a computer to **understand words & sentences** like humans do to solve tasks such as optical character recognition, machine translation, or speech recognition.

Credit: CBS/Merv Griffen

- (i) in 2011 **IBM Watson** beats previous champions on Jeopardy!; (ii) in 2018 **Google Duplex** powering Google's virtual assistant makes calls to hair salon and a restaurant.

1.3 Success stories: NLP II

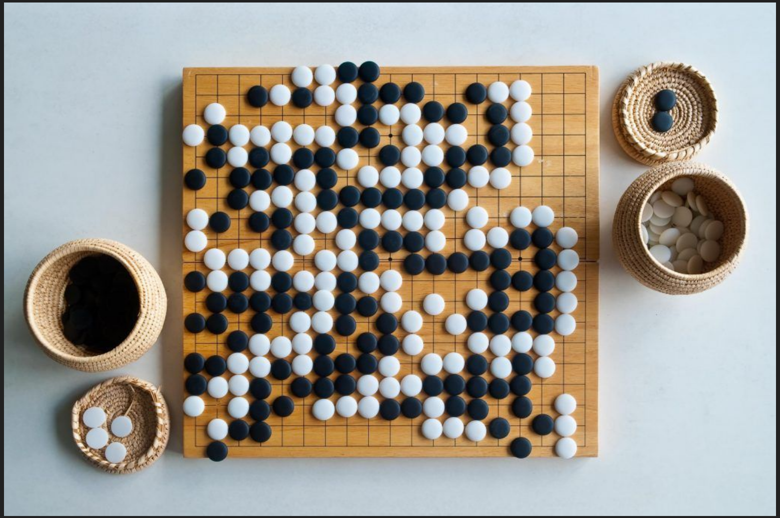
- NLP involves representing text into machine under-standable format such as matrices and **word vec-tors / sequences,** whose context and relations can be analysed by NNs (often **recurrent NNs**).
- A **simple example** on how to represent text is shown on the right.



Credit: Google

1.3 Success stories: Game playing

- Applying **reinforced learning** (interaction with environment to receive feedback), programs based on DL have been shown to meet or surpass human-level performance in (board and video) games.

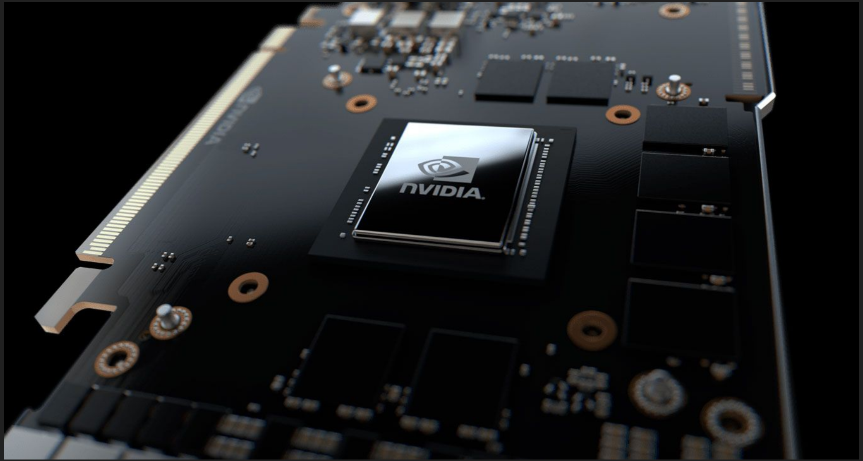

Credit: Saran Poroong

- An example: **AlphaGo** plays the two-player **board game Go** and was able to win against a professio-nal Go player in 2015.
- A **complicated game** with more than 10^{170} board positions, the NN learns from an initial set of games and then **playing against itself**.

1.4 Hardware: GPUs (& TPUs) I

- Although DL was already pioneered in the 1970s and 1980s, it took until the **late 2000s** for a **breakthrough**. Deep NNs with many free parameters require **many fast computations** for efficient training.

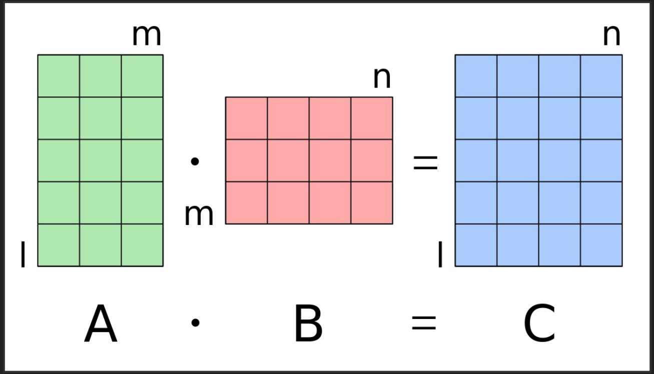
- The DL field was propelled forward by the usage of **graphics processing units (GPUs)**, which significantly accelerated the optimisation of NNs. E.g. AlexNet was a GPU-based algorithm.



Credit: Nvidia

1.4 Hardware: GPUs (& TPUs) II

- Training large NNs boils down to a lot of **matrix multiplications** with billions of parameters. This takes years on a standard CPU (central processing unit), where operations are executed sequentially.



Credit: Wikimedia Commons

- The advantage of GPUs is that operations can be **run in parallel** (at the same time) and **large** amounts of **data** can be handled, resulting in training **speed-ups** of about a factor 10 compared to CPUs.
- **Tensor processing units (TPUs)** originally developed by Google and available since 2018 are even more optimised to the needs of DL.

1. Questions

- Go to www.menti.com & enter 8245 3668.
  - 1. The *deep* in Deep Learning refers to the number of neurons stacked in each hidden layer of a neural network. Correct?
    - No
    - Yes
  - 2. The Deep Learning breakthrough was made possible by which of the following aspects?
    - Advances in hardware (GPUs)
    - New algorithms
    - Large training datasets

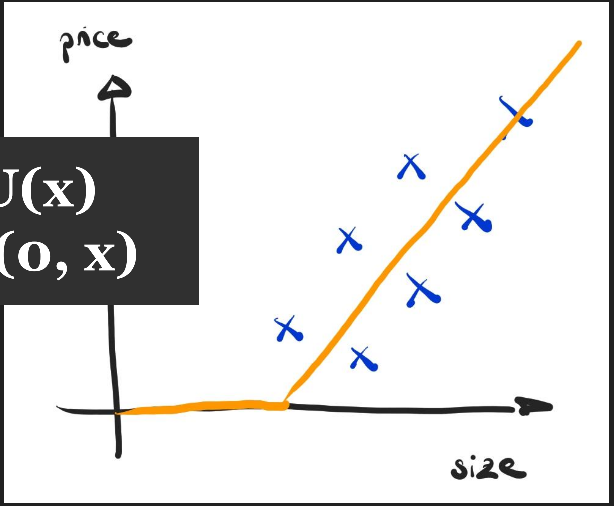
1. Questions

- Go to www.menti.com & enter 8245 3668.
  - 1. The *deep* in Deep Learning refers to the number of neurons stacked in each hidden layer of a neural network. Correct?
    - No (it's the numbers of hidden layers)
    - Yes
  - 2. The Deep Learning breakthrough was made possible by which of the following aspects?
    - Advances in hardware (GPUs)
    - New algorithms
    - Large training datasets

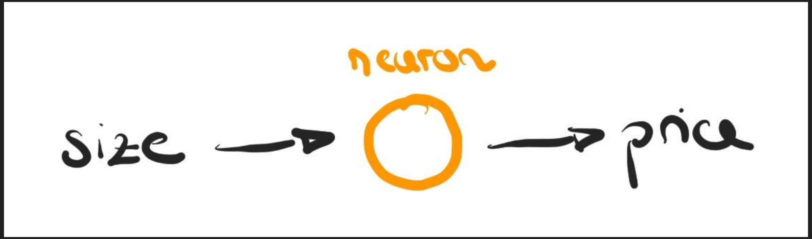
Overview:

2.1 NN examples: house prices I

- Let's look at a classic example: **housing price prediction**. We know the house sizes (**input vector**) and want to predict their prices (**output vector**) based on a few objects for which we know both. A **single neuron** is the simplest network between these two vectors.

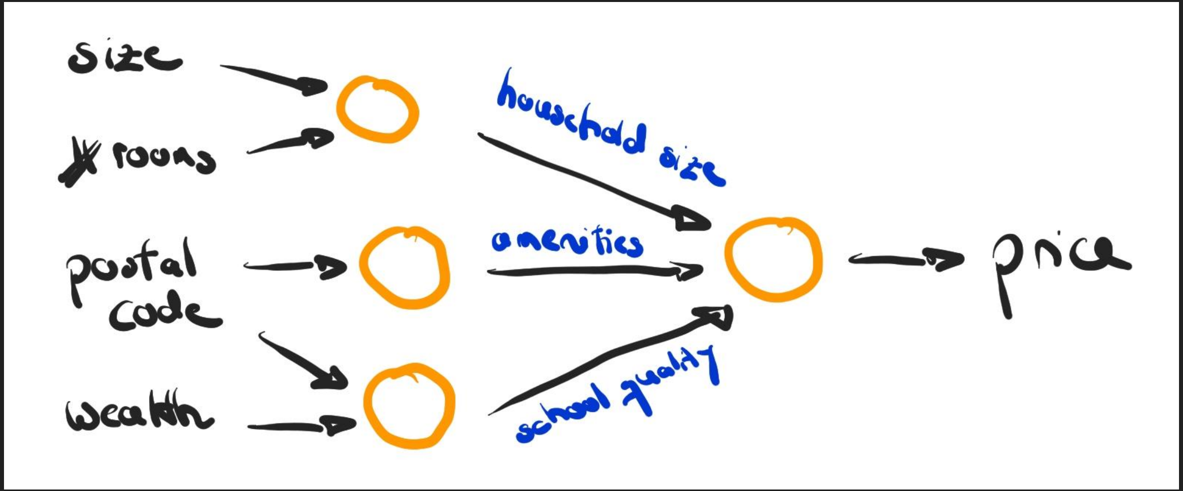


ReLU(x) = max(0, x)



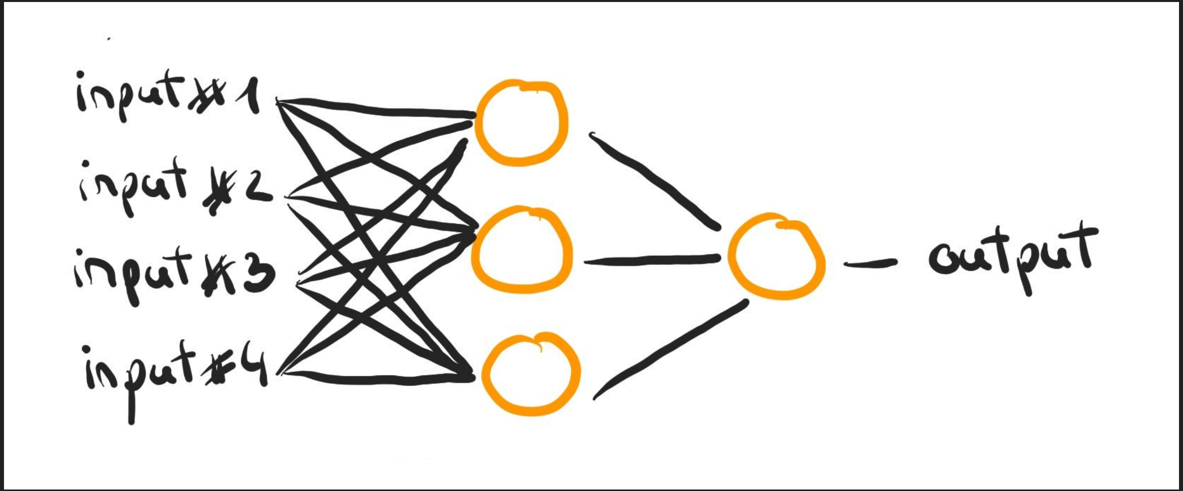
2.1 NN examples: house prices II

- To obtain better predictions, **additional features** that influence the housing price can be added. We e.g. imagine the following scenario including number of rooms, postal code & wealth of neighbourhood:



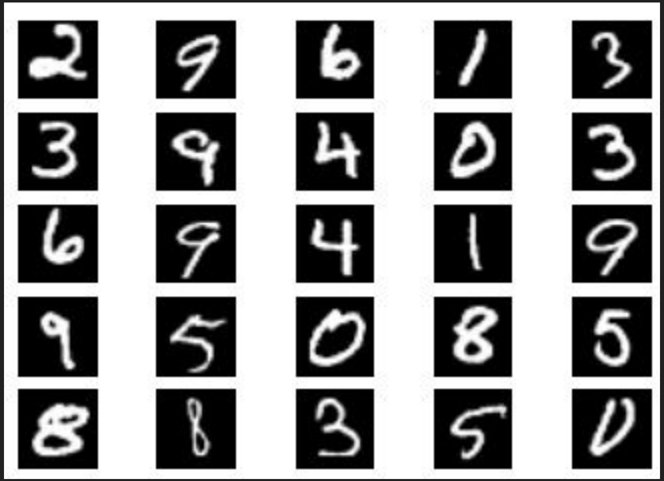
2.1 NN examples: house prices II

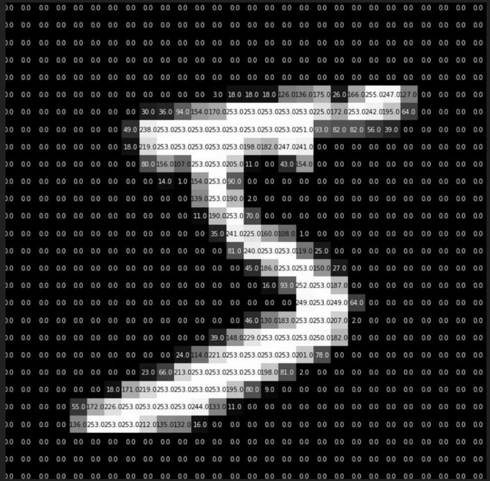
- To obtain better predictions, **additional features** that influence the housing price can be added. We e.g. imagine the following scenario including number of rooms, postal code & wealth of neighbourhood:



2.1 NN examples: MNIST dataset

- One of the **benchmark problems** in ML is image recognition using the labelled **MNIST dataset** of handwritten digits. A DL program has to determine the number from the 28 x 28 = **784 input pixel** containing a **grayscale value** between 0 (black) and 1 (white).

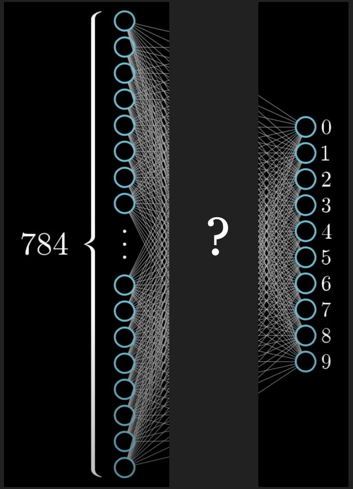

Credit: Andrew Gibiansky


Credit: Cecelia Shao

2.2 NN basics: the neuron

Credit: ktsdesign

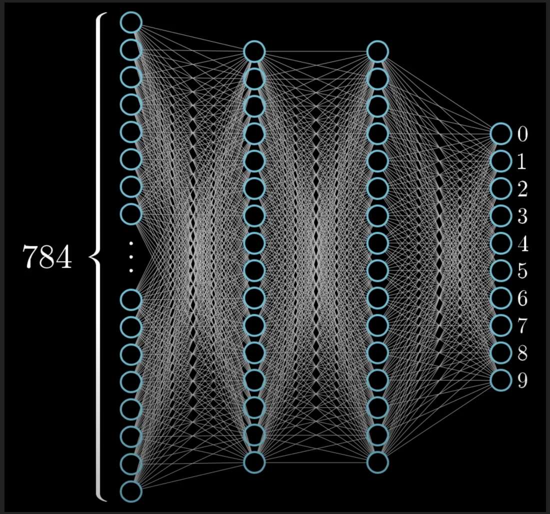
- By analogy with neuroscience, we interpret a **neuron** as an object holding a **numerical value** between 0 and 1, its **activation**.
- For digit recognition, our **input layer** consists of 784 **neurons** corresponding to the image pixels.
- **Output layer** has 10 neurons with activations correspon-ding to the probability of recognising the digits 0-9.



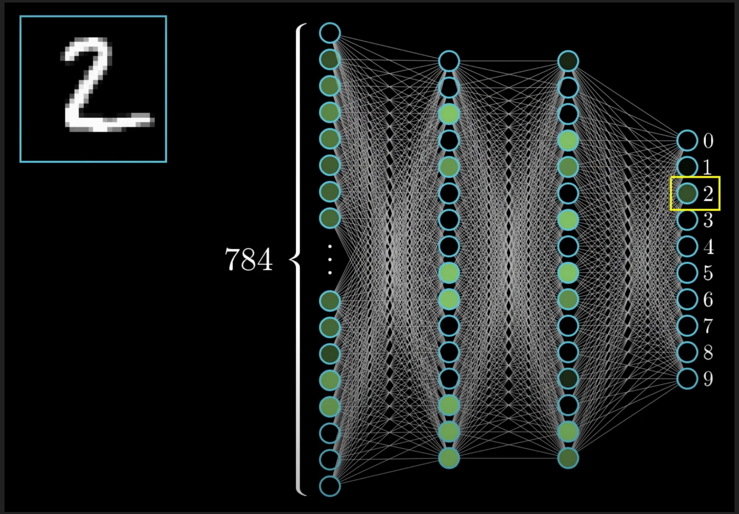
Credit: 3Blue1Brown, YT

2.2 NN basics: multilayer perceptron

- The multilayer perceptron is the **classic DL NN design** consisting of input layer, hidden layers and output layer that are **fully connected**.



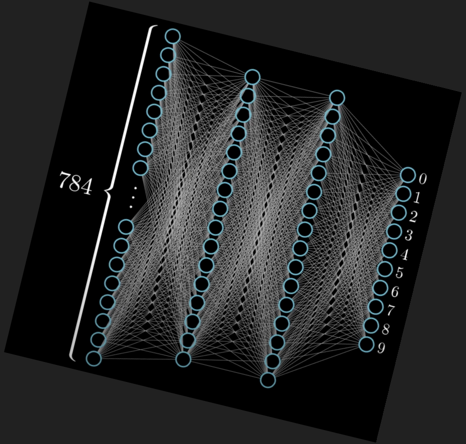
Credit: 3Blue1Brown, YT



Credit: 3Blue1Brown, YT

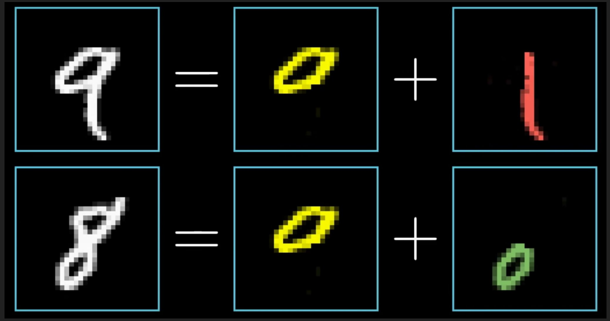
2.2 NN basics: feature recognition

- To extract features, we e.g. imagine: (i) the second hidden layer represent **patterns** such as **loops or lines**; (ii) the first hidden layer abstracts these subfeatures into **edges**.

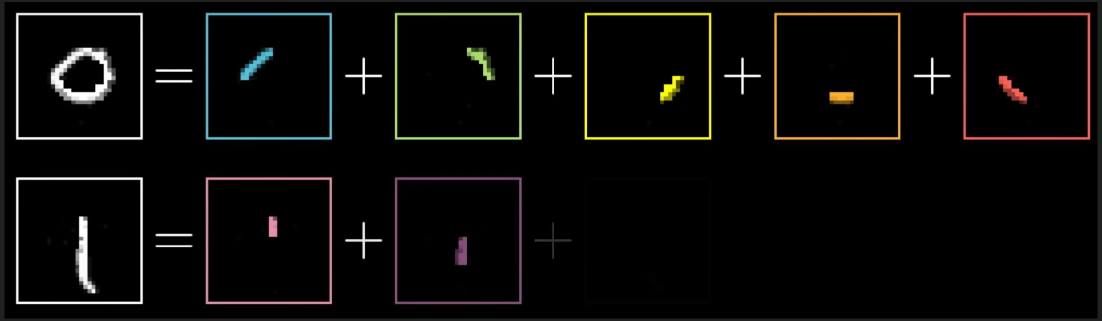


pixel → edges

→ patterns → digits



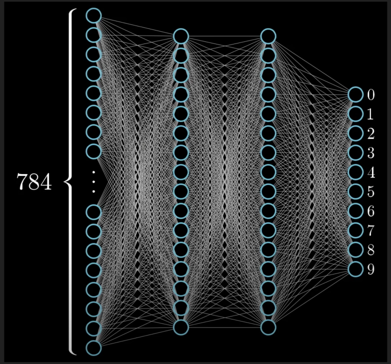
Credits: 3Blue1Brown, YT



2.2 NN basics: the components I

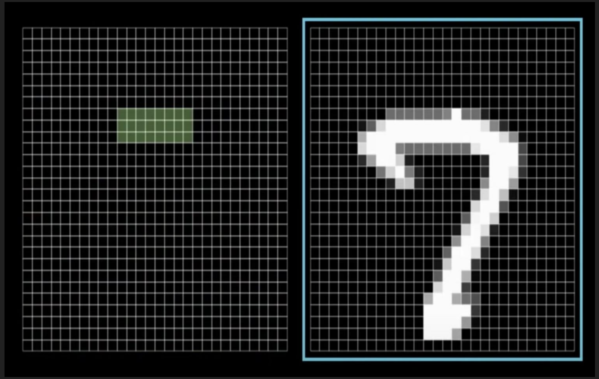
- The activations in the second network layer (first hidden layer) are obtained as a **weighted sum** of the activations in the first layer. For example:

$$w_1 a_1^{(1)} + w_2 a_2^{(1)} + \dots + w_n a_n^{(1)} = a_1^{(2)}$$



Credit: 3Blue1Brown, YT

Credit: 3Blue1Brown, YT

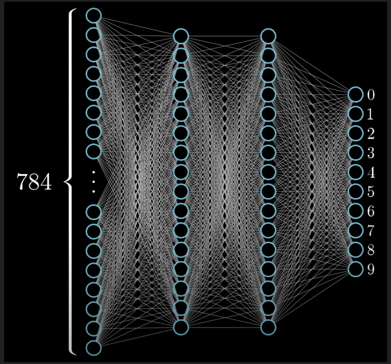


- The weights can be visualised as a matrix that we multiply with the input layer. To **recognise edges** in a certain part, those weights have to be positive while all others should vanish.

2.2 NN basics: the components I

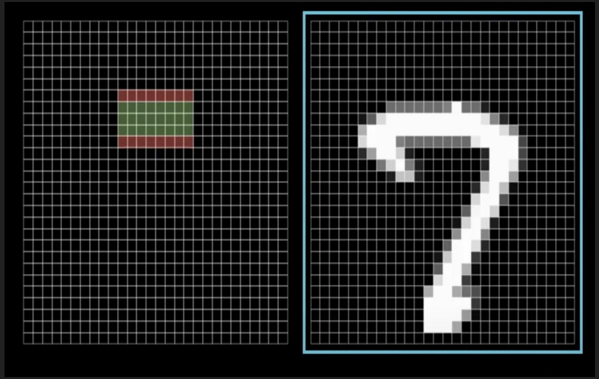
- The activations in the second network layer (first hidden layer) are obtained as a **weighted sum** of the activations in the first layer. For example:

$$w_1 a_1^{(1)} + w_2 a_2^{(1)} + \dots + w_n a_n^{(1)} = a_1^{(2)}$$



Credit: 3Blue1Brown, YT

Credit: 3Blue1Brown, YT

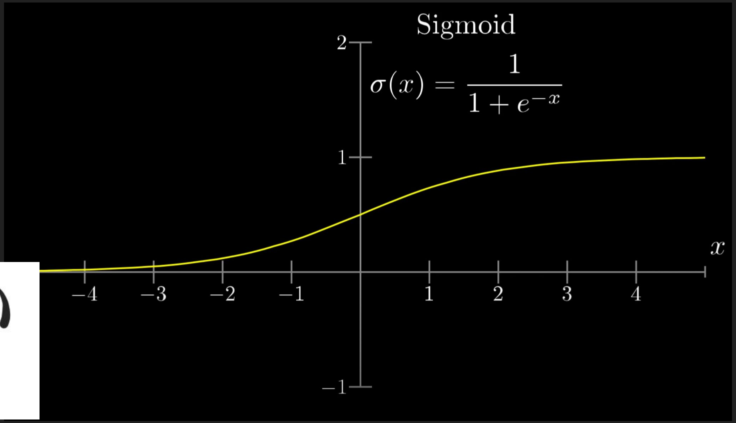


- The weights can be visualised as a matrix that we multiply with the input layer. To **recognise edges** in a certain part, those weights have to be positive while all others should vanish.

2.2 NN basics: the components II

- Our weighted sum can take any value, but for comparison is often rescaled by an **activation function**. Historically, the **sigmoid function** was used to obtain weighted sums in the range 0 to 1. Modern NNs typically use the **ReLU** as the activation function.

- To trigger a certain neuron above a **specific threshold**, we can add a **bias** to the sum, shifting the sigmoid.



$$\sigma\left(w_1 a_1^{(1)} + \dots + w_n a_n^{(1)} + b_1\right) = a_1^{(2)}$$

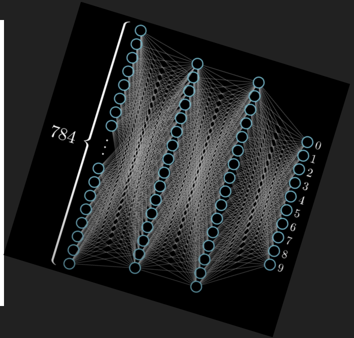
Credit: 3Blue1Brown, YT

2.2 NN basics: it’s all maths

- For our two-hidden layer NN with 16 neurons each, 784 input and 10 output neurons, we are left with **13,002 free parameters** to adjust.

neurons weights biases

$$\underbrace{784\times 16+16}_{\text{1st hidden layer}}+\underbrace{16\times 16+16}_{\text{2nd hidden layer}}+\underbrace{16\times 10+10}_{\text{output layer}}$$



- Everything we have seen is closely connected to **linear algebra and vector calculus** and we can take advantage of this when optimising the free parameters.

$$\vec{a}^{(2)}=\sigma(\underline{w}\,\vec{a}^{(1)}+\vec{b})$$

2. Questions I

- Go to www.menti.com & enter 1244 8724.
  - 1. The idea of a deep NN is that subsequent layers obtain successively abstract representations of the input data?
    - Yes
    - No
  - 2. Which of the following is optimised during NN training?
    - Network weights
    - Activation function
    - Network biases

2. Questions I

- Go to www.menti.com & enter 1244 8724.
  - 1. The idea of a deep NN is that subsequent layers obtain successively abstract representations of the input data?
    - Yes
    - No
  - 2. Which of the following is optimised during NN training?
    - Network weights
    - Activation function
    - Network biases

2.3 NN training

- Training a NN proceeds in the following steps:
    - **Forward pass:** calculate individual neuron activations for each network layer.
    - **Cost calculation:** determine the cost (error) of the prediction; how close is the prediction to the expected output (digit label from MNIST).
    - **Update step:** update all network parameters to produce a better prediction; this involves the crucial step of **backpropagation**.
- **Repeat** these steps until the **network converges**.

Credit: S Baker / Unsplash

2.4 Forward pass

- We have already seen earlier how **activations** are calculated for the first hidden layer. We can **generalise** this to all L network layers:

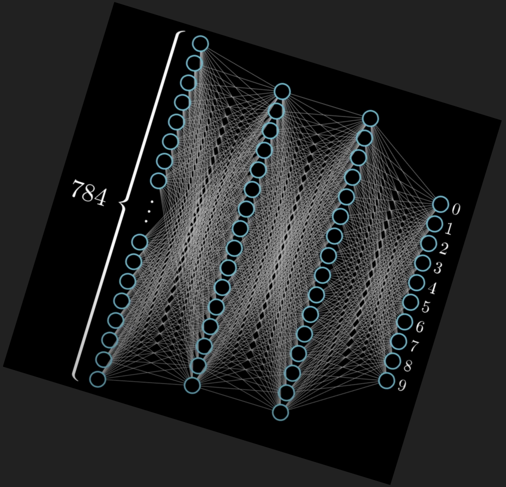
$$\vec{a}^{(2)} = \sigma\left(\underline{w}^{(2)}\vec{a}^{(1)} + \vec{b}^{(2)}\right)$$

$$\vec{a}^{(3)} = \sigma\left(\underline{w}^{(3)}\vec{a}^{(2)} + \vec{b}^{(3)}\right)$$

$$\vdots$$

$$\vec{a}^{(L)} = \sigma\left(\underline{w}^{(L)}\vec{a}^{(L-1)} + \vec{b}^{(L)}\right)$$

$$\Rightarrow \vec{a}^{(L)} = \sigma\left(\underline{w}^{(L)} \cdot \sigma\left(\underline{w}^{(L-1)} \cdot \sigma(\ldots) + \vec{b}^{(L-1)}\right) + \vec{b}^{(L)}\right)$$

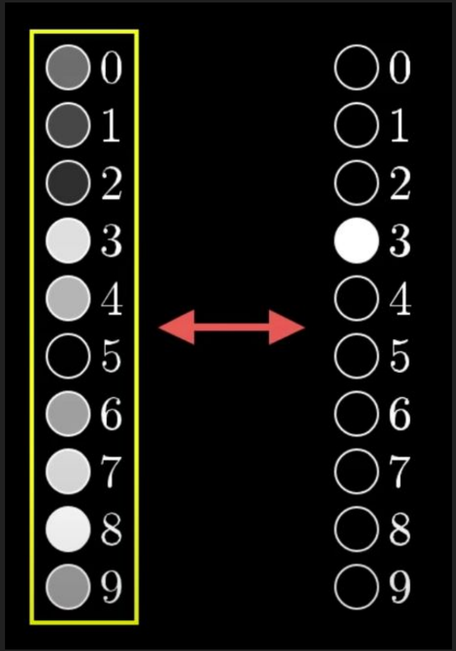


- For first forward pass, weights & biases are **randomly initialised**.

2.5 Optimisation: cost function I

- At the end of each forward pass, we obtain a **prediction** of the network. To determine quantitatively how good this prediction is w.r.t. to a **ground truth** (the digit label), we require a **cost function**.

- For each set of NN parameters this function gives a **single number** that represent how good our network is doing. The goal is to find the set of weights and biases that **minimises this cost function.**

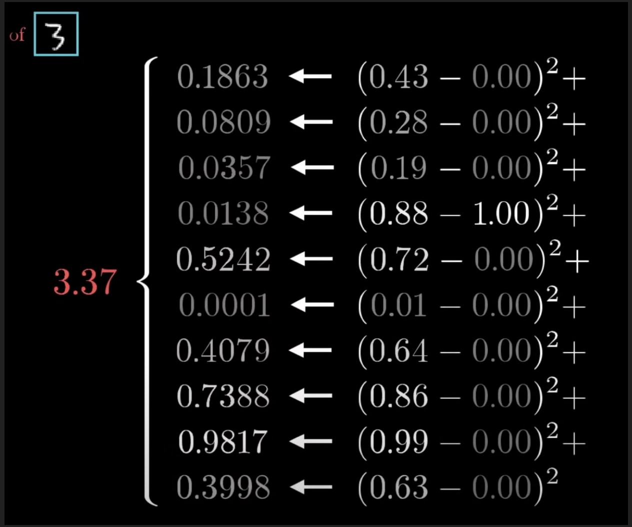


Credit: 3Blue1Brown, YT

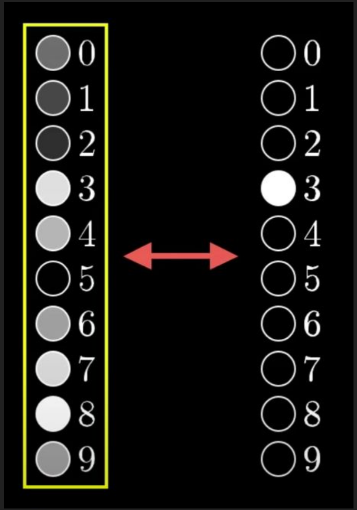
2.5 Optimisation: cost function II

- Selecting an appropriate cost function is crucial for NN training. A typical example is the **mean square error**, which we calculate between the activations of the final layer and the ground truths for a single image and then average for **all training examples**.

Credits: 3Blue1Brown, YT

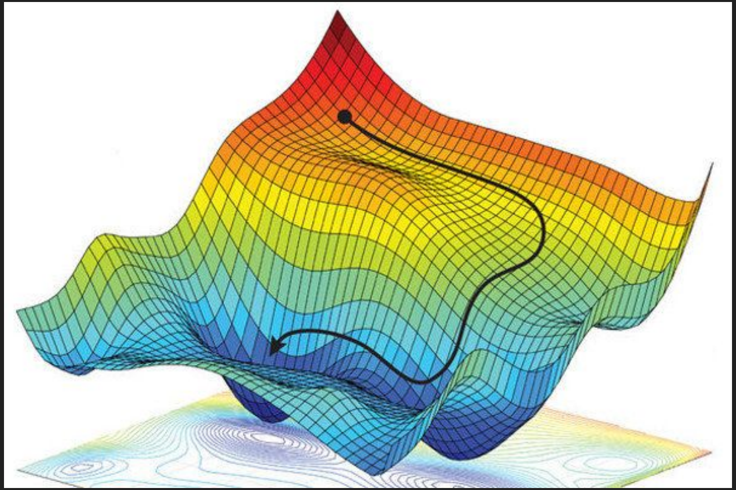


$$\text{of } \boxed{3} \qquad 3.37 \left\{ \begin{array}{lcl} 0.1863 & \leftarrow & (0.43 - 0.00)^2 + \\ 0.0809 & \leftarrow & (0.28 - 0.00)^2 + \\ 0.0357 & \leftarrow & (0.19 - 0.00)^2 + \\ 0.0138 & \leftarrow & (0.88 - 1.00)^2 + \\ 0.5242 & \leftarrow & (0.72 - 0.00)^2 + \\ 0.0001 & \leftarrow & (0.01 - 0.00)^2 + \\ 0.4079 & \leftarrow & (0.64 - 0.00)^2 + \\ 0.7388 & \leftarrow & (0.86 - 0.00)^2 + \\ 0.9817 & \leftarrow & (0.99 - 0.00)^2 + \\ 0.3998 & \leftarrow & (0.63 - 0.00)^2 \end{array} \right.$$



2.5 Optimisation: gradient descent I

- Like discussed yesterday, the standard way to minimise the cost function is via **gradient descent**, an algorithm that essentially helps us **find a minimum** (local or global) of the **cost landscape**.


Credit: Amini et al. (2018)

- To do so, we determine the direction of the **steepest gradient** at a given point and make a step in the opposite direction, or mathematically

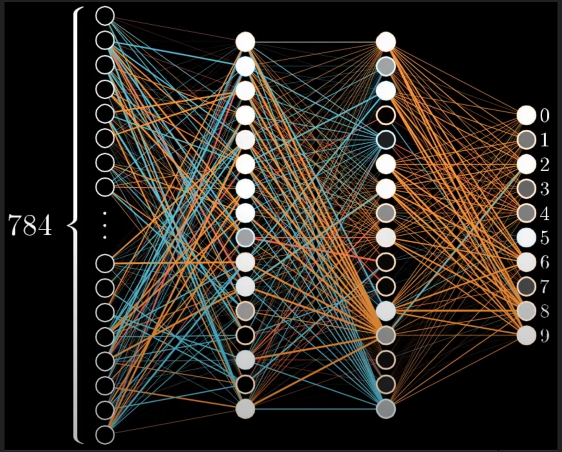
$$\vec{p} \rightarrow \vec{p} - \alpha \nabla C(\vec{p})$$

where α is the **learning rate**.

2.5 Optimisation: gradient descent II

- While it is possible to determine gradients for simple cost landscapes analytically and then update weights and biases to obtain better predictions, doing that even for our simple two-layer NN is **not feasible**.

$\propto \nabla C(\vec{p})$



Credit: 3Blue1Brown, YT

- Keep in mind that we need to **compute derivatives** w.r.t. to 13,002 parameters!!

- The crucial advancement in DL came in the form of backpropagation, an **algorithm** to efficiently compute the gradients of the cost function for a single input/output pair.

2.6 Backpropagation: intuition

- Before we turn to the actual mathematics of the backpropagation algorithm, let’s develop a bit of **intuition** for what is happening.

- To do so, please watch one of the videos from **3Blue1Brown** on Youtube dedicated to

  What is backpropagation really doing? | Chapter 3, Deep learning

  https://tinyurl.com/2f6aj9yk

2. Questions II

- Go to www.menti.com & enter 5563 4010.
  - 3. Which of the following adjustments will increase the activation of any given neuron?
    - Decreasing its bias
    - Increasing weights $\propto$ to activations of previous layer
    - Adjust activations of previous layer $\propto$ to weights
  - 4. What is the benefit of batched stochastic gradient descent?
    - The update steps are more accurate.
    - It is faster than standard gradient descent.

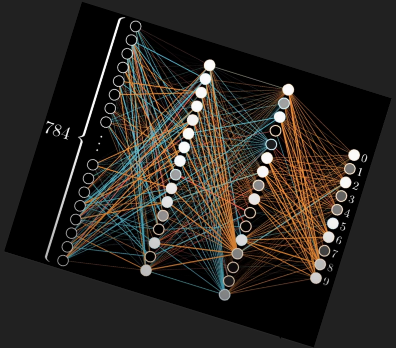
2. Questions II

- Go to www.menti.com & enter 5563 4010.
    - 3. Which of the following adjustments will increase the activation of any given neuron?
        - Decreasing its bias
        - Increasing weights $\propto$ to activations of previous layer
        - Adjust activations of previous layer $\propto$ to weights
    - 4. What is the benefit of batched stochastic gradient descent?
        - The update steps are more accurate.
        - It is faster than standard gradient descent.

2.6 Backpropagation: maths I

- To update the network parameters, we want to calculate

$$w^{(l)} \rightarrow w^{(l)} - \alpha \frac{\partial C}{\partial w^{(l)}}, \quad b^{(l)} \rightarrow b^{(l)} - \alpha \frac{\partial C}{\partial b^{(l)}}$$

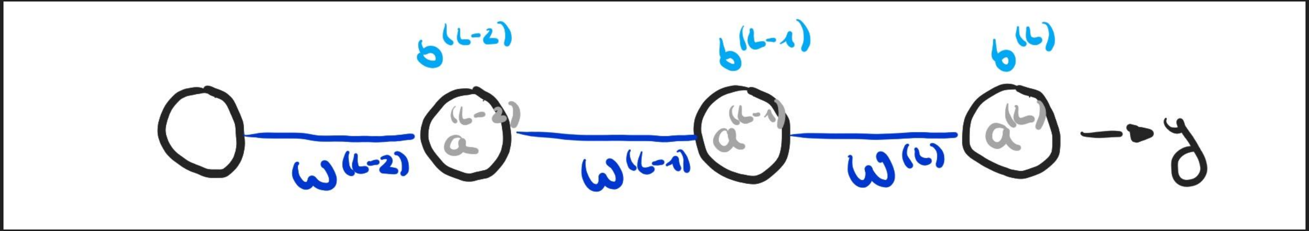


- The main idea of backpropagation is to compute the cost in our last layer and successively backpropagate it through all previous layers. To derive the corresponding equations, we will require the **chain rule**:

$$h(x) = f(g(x)) \Rightarrow h'(x) = f'(g) \cdot g'(x)$$

2.6 Backpropagation: maths II

- Let's look at a **simple example** to understand how the different network parameters actually affect our cost function:

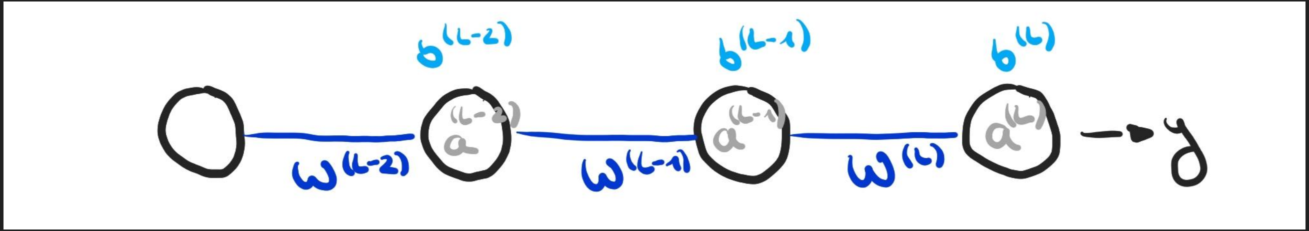


- The **cost for a single training example** is given by the following expression using the activation in the final neuron (L)

$$C = (a^{(L)} - y)^2 = [\sigma(w^{(L)} a^{(L-1)} + b^{(L)}) - y]^2$$

2.6 Backpropagation: maths II

- Let’s look at a **simple example** to understand how the different network parameters actually affect our cost function:



- The **cost for a single training example** is given by the following expression using the activation in the final neuron (L)

$$C = \left(a^{(L)} - y\right)^2 = \left[\sigma\left(\underbrace{w^{(L)} a^{(L-1)} + b^{(L)}}_{=: z^{(L)}}\right) - y\right]^2$$

2.6 Backpropagation: maths III

$$\frac{\partial C}{\partial w^{(L)}}$$

- Let's first focus on derivative of C w.r.t. to the final weight:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

- Similarly, to obtain the derivative of C w.r.t to the weight in L-1:

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L-1)}}$$

$$= \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

2.6 Backpropagation: maths IV

- We can **evaluate these partial derivatives** to obtain:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} = 2(a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot a^{(L-1)}$$

$$= 2(a^{(L)} - y) \cdot \sigma(z^{(L)})(1 - \sigma(z^{(L)})) \cdot a^{(L-1)}$$

$$= 2(a^{(L)} - y) \cdot a^{(L)}(1 - a^{(L)}) \cdot a^{(L-1)}$$

- Similarly, we obtain for the **derivative w.r.t. the last bias**

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial b^{(L)}} = 2(a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot 1$$

2.6 Backpropagation: maths V

- For the derivatives w.r.t. To parameters in the second to last layer:

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

$$= \frac{\partial C}{\partial z^{(L)}} \cdot w^{(L)} \cdot \sigma'(z^{(L-1)}) \cdot a^{(L-2)}$$

$$\frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial C}{\partial z^{(L)}} \cdot w^{(L)} \cdot \sigma'(z^{(L-1)}) \cdot 1$$

- Equivalent expression hold for **all hidden layers**. Backpropagation is efficient, since we can **reuse quantities** from the previous layer.

Summary:

- We have introduced the concept of **Deep Learning**, discussed some of the main **success stories** (image recognition, natural language processing and game playing) and addressed the reasons why this approach is so successful right now (data sets, hardware, algorithms).
- Subsequently, we addressed some classic examples of neural networks and discussed the **main components** of these structures (neurons, activation functions, weights and biases) and how the concept of **feature abstraction** works.
- We then talked about the **training process** of deep neural networks and looked at **forward passes**, **cost calculation** and finally the update step which involved the crucial **backpropagation** algorithm.