Berkeley Engineering | BerkeleyHaas

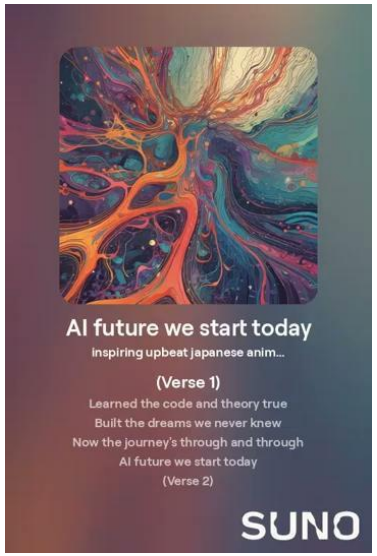
PROFESSIONAL CERTIFICATE IN MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Let’s give everyone a couple of minutes to join…

Module 22

Deep Neural Networks

Office Hours with Viviana Márquez
February 27, 2025



AGENDA

- Capstone meetings and FAQ
- Where have we come from and what do we have left?
- Content review: Deep Neural Networks
- Questions

AGENDA

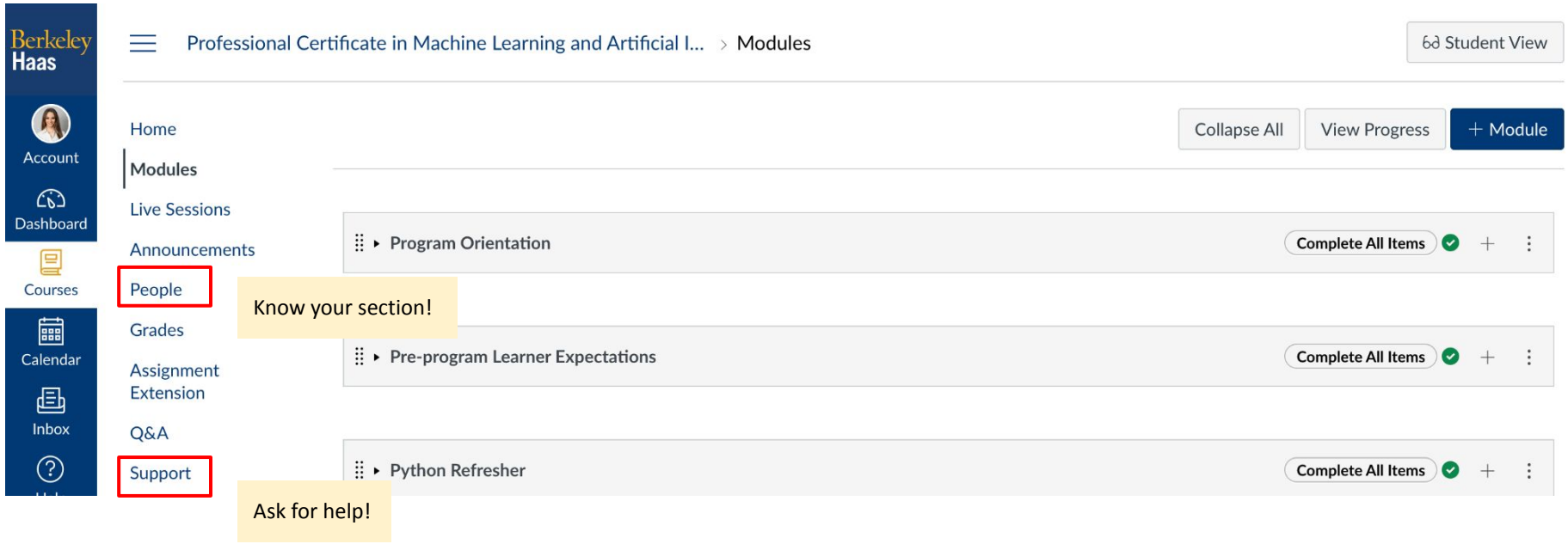
- Capstone meetings and FAQ
- Where have we come from and what do we have left?
- Content review: Deep Neural Networks
- Questions

CAPSTONE PROJECT

- **Schedule your second 1:1 with your Learning Facilitator**
  - **Vikesh Koul (Section A)**: https://calendly.com/vkoul/vikesh-bh-pcmlai-24-08-capstone-consultation-session-2?back=1&month=2025-02
  - **Jessica Cervi (Section B)**: https://calendly.com/jessicacervi/bh-pcmlai-24-08-capstone-consultation-2
  - **Viviana Márquez (Section C)**: https://calendly.com/vivianamarquez/bh-pcmlai-24-08-capstone-consultation-2-clone
  - **Francesca Vera (Section D)**: https://calendly.com/envision-learn/
  - **Mani Kannapan (Section E):**
  - https://calendly.com/mani-kannappan/mani-bh-pcmlai-24-08-capstone-consultation-session-1?month=2025-02

Only one 30 min consultation per learner during
Mod 21 - Mod 23
February 19, 2025 - March 11, 2025

CANVAS


Berkeley Haas
Account
Dashboard
Courses
Calendar
Inbox
Professional Certificate in Machine Learning and Artificial I... > Modules
Student View
Home
Modules
Live Sessions
Announcements
People
Grades
Assignment Extension
Q&A
Support
Collapse All
View Progress
+ Module
Program Orientation
Complete All Items
Pre-program Learner Expectations
Complete All Items
Python Refresher
Complete All Items
Know your section!
Ask for help!

CAPSTONE PROJECT

- **Roadmap**
  - **Module 6**
    Draft the Problem Statement 👩‍💻
  - **Modules 12 to 15**
    First 1:1 With Your Learning Facilitator
  - **Module 16**
    Finalizing Your Problem Statement
  - **Module 20**
    Initial Report and EDA 👩‍💻
  - **Modules 21 to 23**
    Second 1:1 With Your Learning Facilitator
  - **Module 24**
    Final Analysis and Report 👩‍💻

An outline of the BH-PCMLAI passing criteria

Category	Module #	Weight
Codio Activities	Modules 1–4, 6–10, 12–16, 18, 19, 21–23	15%
Knowledge Checks and Discussions	Modules 1-4, 6–10, 12–16, 18, 19, 21–23	15%
Practical Application Problems	Modules 5, 11, 17	20%
Capstone Project Part 1	Modules 6, 16, 20	20%
Capstone Project Part 2	Module 24	30%

CAPSTONE PROJECT

- If you want to go the extra mile…

Dash by plotly

Streamlit

github:pages

AGENDA

- ✅ Capstone meetings
- Where have we come from and what do we have left?
- Content review: Deep Neural Networks
- Questions

LEARNING JOURNEY – PROGRAM OVERVIEW

Three Content Areas:

Section 1 (Mod 1-5) Foundations (probability, statistics, data analysis, and code)
Section 2 (Mod 6-17,20) Machine Learning
Section 3 (Mod 21-24) NLP, Recommenders, Deep Learning

Structured data vs unstructured data

Usually Machine Learning

Mass (g)	Extension 1 (mm)	Extension 2 (mm)	Average Extension (mm)
0	0	1	0.5
100	5	6	5.5
200	9	9	9
300	15	15	15
400	20	21	20.5
500	24	25	24.5
600	30	31	30.5

- **Structured data** is highly organized and easily readable by machines. It is typically stored in tabular formats, such as spreadsheets (CSV, Excel) or relational databases (SQL).

  Each observation is in a row, and its features are in predefined columns, making it easier to process and analyze.

Usually Deep Learning

- **Unstructured data** does not follow a specific format or structure, making it more challenging to organize and analyze. This type of data includes free text, images, videos, audio, and other multimedia formats.

  Due to its nature, unstructured data often requires advanced techniques such as Natural Language Processing (NLP) or Convolutional Neural Networks (CNN).

DEEP LEARNING

Deep Learning modules:

- **Mod 21:** Deep Neural Networks I
  - Foundations of NN
  - Hyperparameter tuning
- **Mod 22:** Deep Neural Networks II
  - CNNs for image classification
  - LSTM for time series
  - FCNN for regression
- **Mod 23:** GenAI
  - GANs
  - Diffusion models
  - RNNs to generate text
  - Transformers to generate text

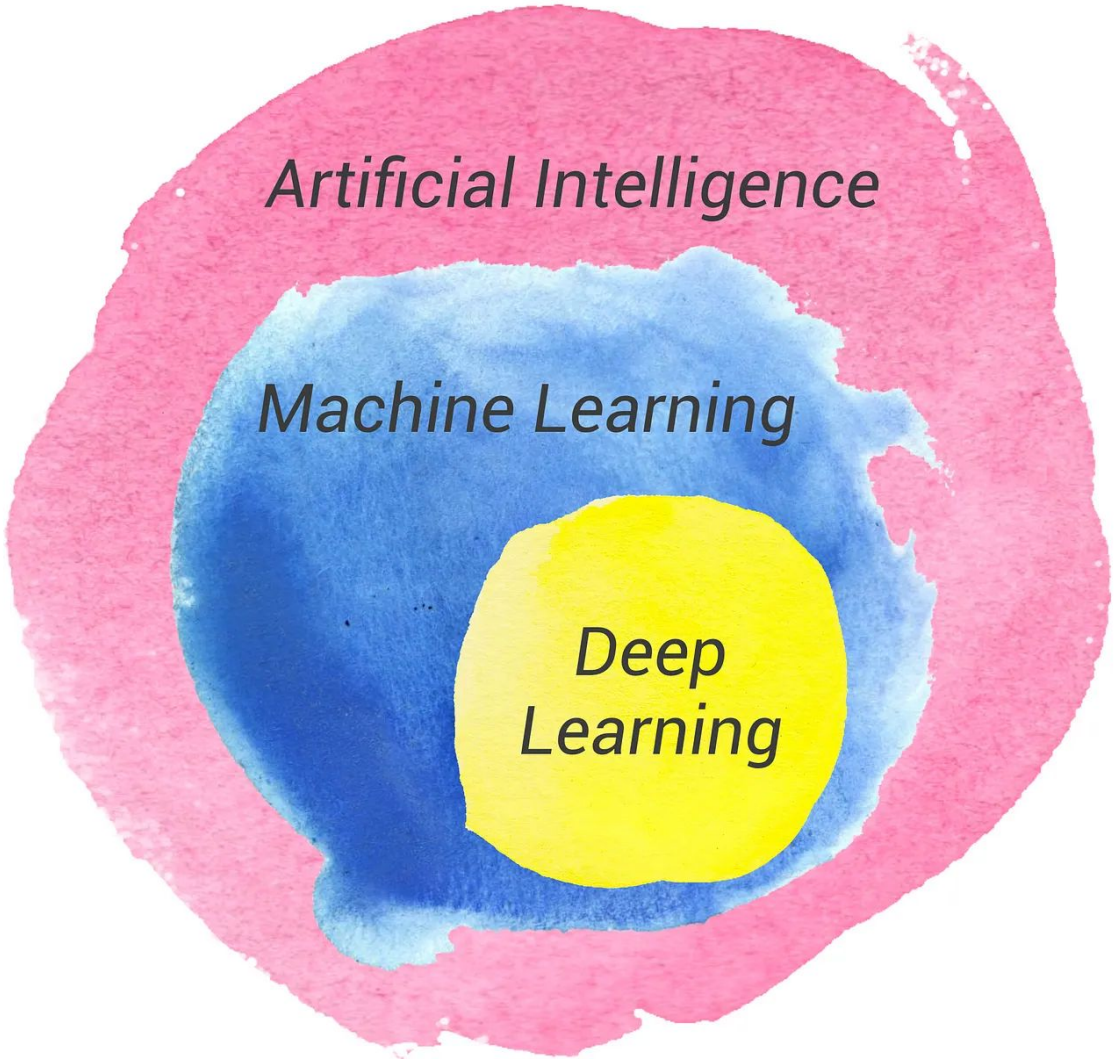
DEEP LEARNING

Deep Learning modules:

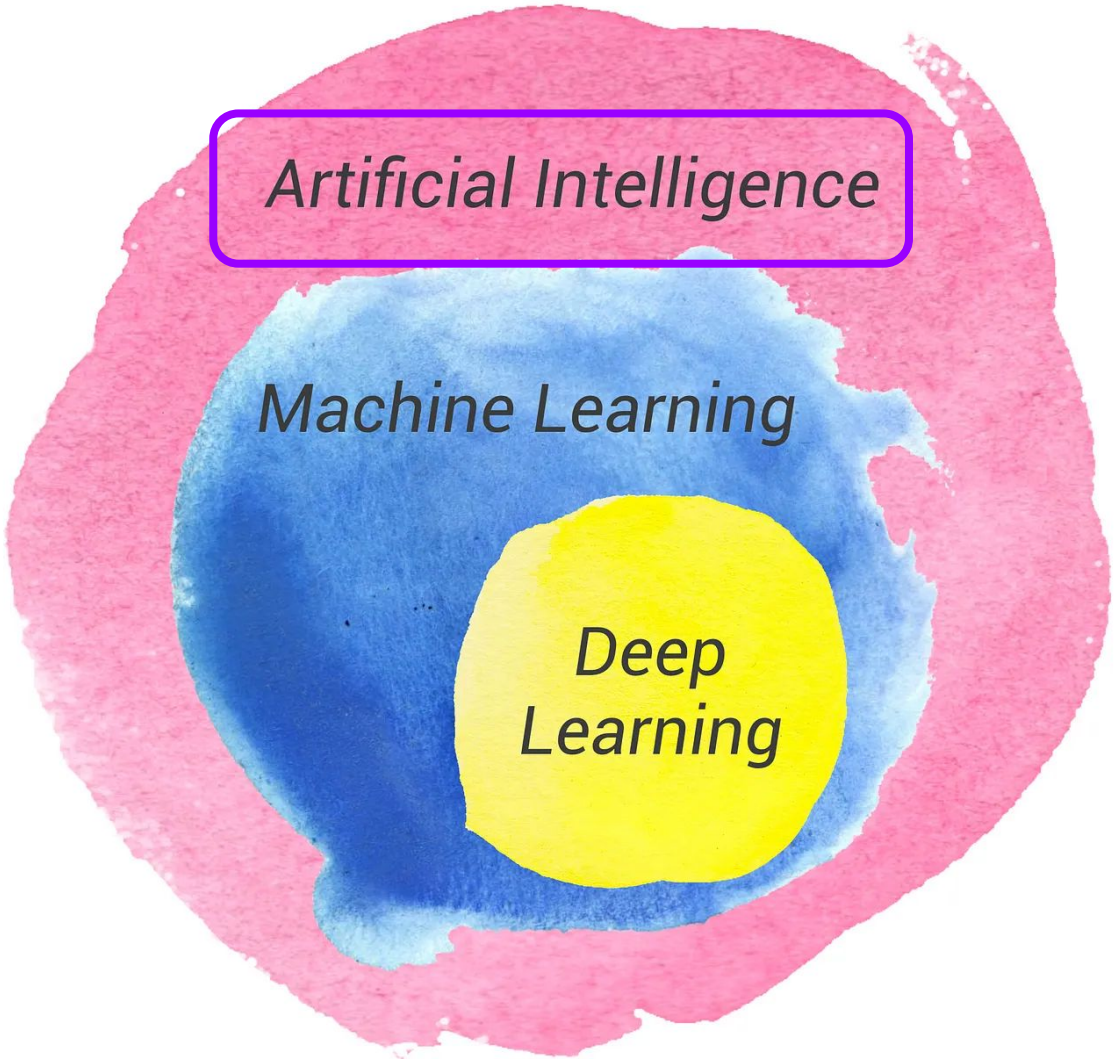
- **Mod 21:** Deep Neural Networks I
  - **Foundations of NN**
  - Hyperparameter tuning
- **Mod 22:** Deep Neural Networks II
  - **CNNs for image classification**
  - LSTM for time series
  - FCNN for regression
- **Mod 23:** GenAI
  - GANs
  - Diffusion models
  - RNNs to generate text
  - Transformers to generate text

AGENDA

- ✅ Capstone meetings
- ✅ Where have we come from and what do we have left?
- Content review: Deep Neural Networks
- Questions


Artificial Intelligence
Machine Learning
Deep
Learning

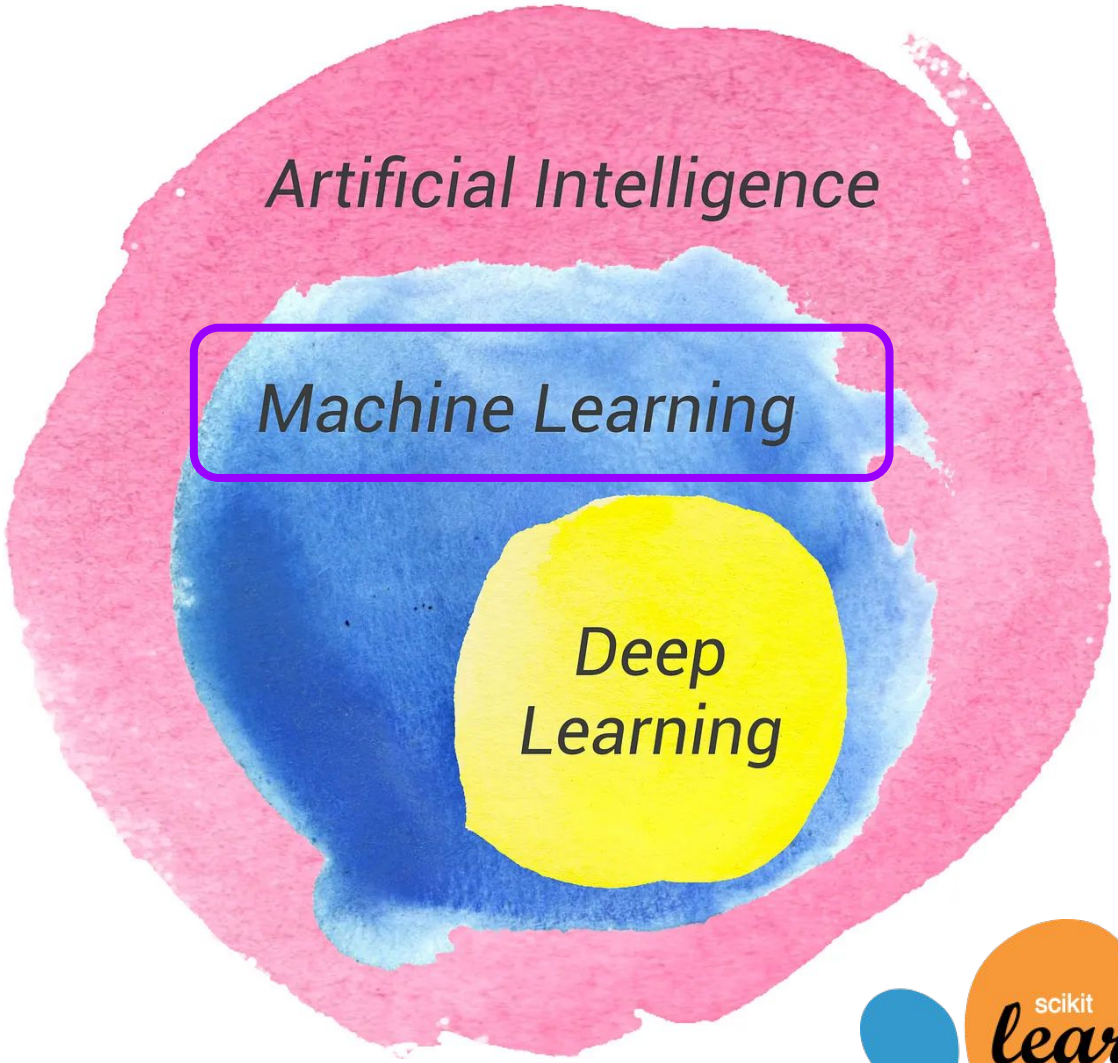

AI
AI everywhere



Artificial Intelligence

AI is the broad field where machines are designed to mimic human intelligence, enabling them to perform tasks like decision-making, language understanding, and problem-solving.

Scope: Broad. AI encompasses everything that allows computers to imitate human intelligence, including robotics, natural language processing, and problem-solving.



Machine Learning

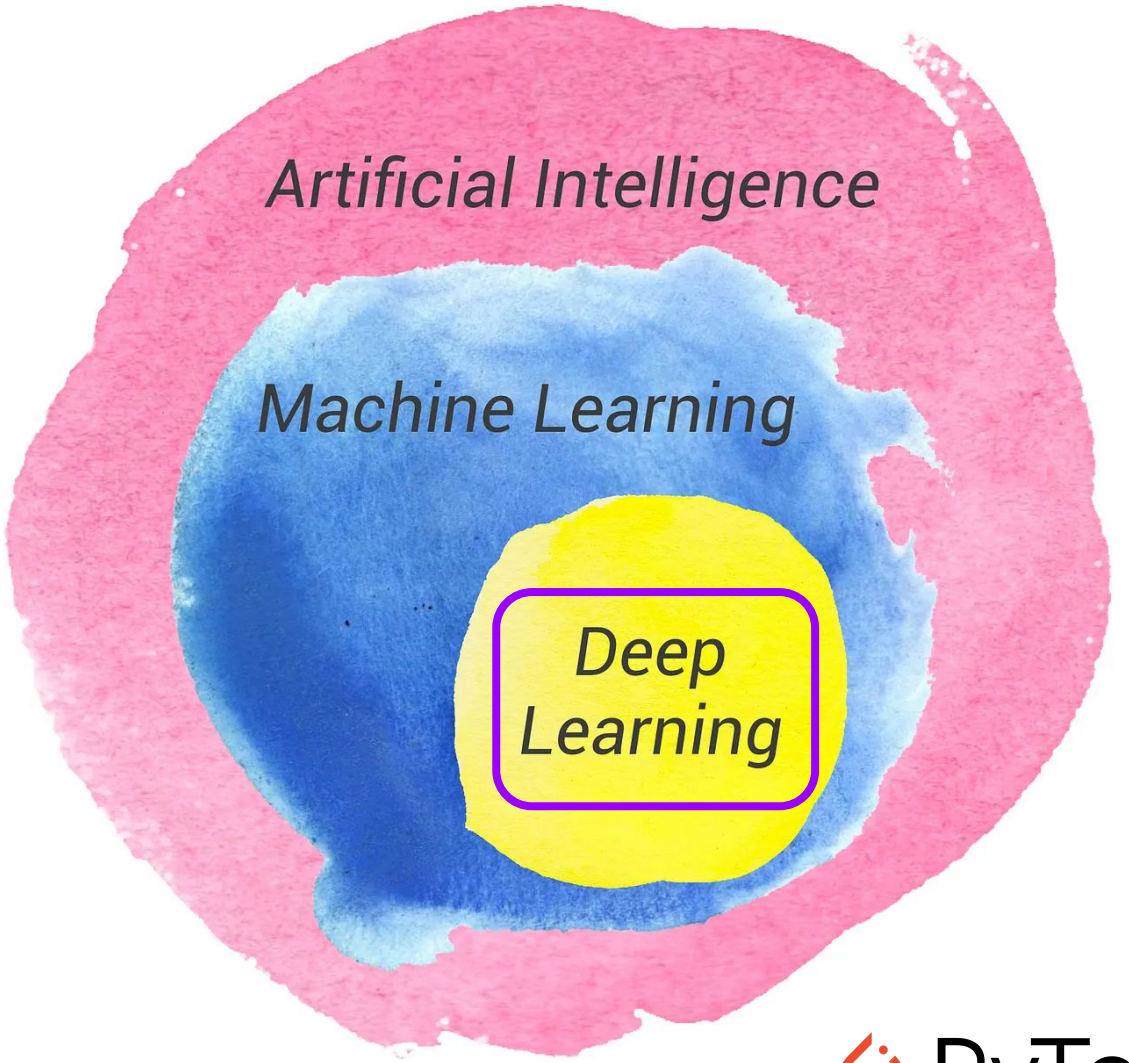
ML is a subset of AI that focuses on creating systems that can learn and improve from experience or data, without being explicitly programmed for every task.

Scope: Moderate. It includes various techniques such as regression, classification, clustering, and ensemble models.

Mass (g)	Extension 1 (mm)	Extension 2 (mm)	Average Extension (mm)
0	0	1	0.5
100	5	6	5.5
200	9	9	9
300	15	15	15
400	20	21	20.5
500	24	25	24.5
600	30	31	30.5

scikit learn

Usually structured data



Deep Learning

DL is a specialized type of machine learning that uses neural networks with many layers to analyze vast amounts of data and learn complex patterns, often achieving results comparable to human performance in areas like image recognition or language translation.

Scope: Narrow. DL is a specific, yet powerful, form of machine learning.

PyTorch

TensorFlow

Usually unstructured data

When to use Deep Learning vs Machine Learning?

Machine Learning

- Limited Data
- Interpretability
- Less Computational Power
- Feature Engineering
- Faster Training
- Problem Domains with Proven ML Success

Deep Learning

- Large Amounts of Data
- Complex Patterns in Data
- High Computational Power
- Problem Domains with Proven DL Success
- End-to-end Learning

Cool examples of DL

- https://chat.openai.com/
- https://www.suno.ai/
- https://www.opus.pro/
- https://elevenlabs.io/app/speech-synthesis
- https://leonardo.ai/
- … what others do you know?

Cool examples of DL

- https://chat.openai.com/
- https://www.suno.ai/
- https://www.opus.pro/
- https://elevenlabs.io/app/speech-synthesis
- https://leonardo.ai/
- … what others do you know?

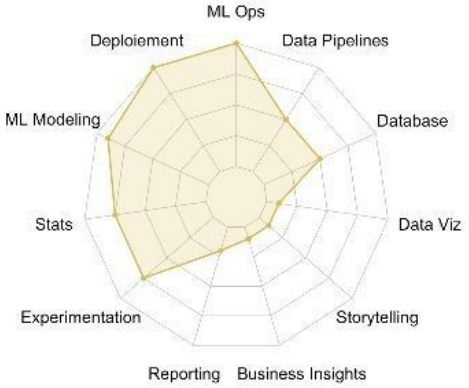
According to OpenAI, the training process of Chat GPT-3 required 3.2 million USD in computing resources alone. This cost was incurred from running the model on 285,000 processor cores and 10,000 graphics cards, equivalent to about 800 petaflops of processing power.

Types of Data Professionals

Data Engineer

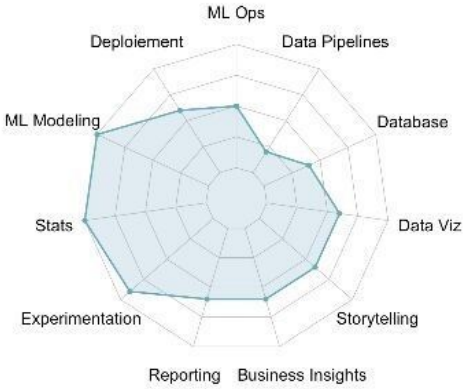
Data Scientists
ML Ops

ML Engineer

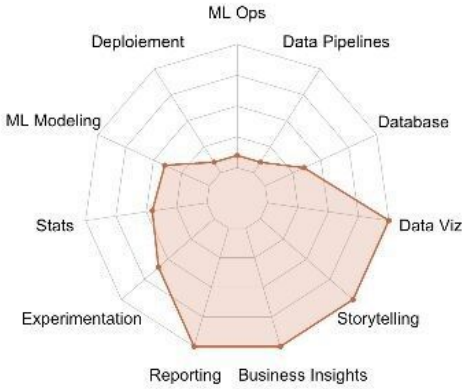


Data Analysts
ML Ops

Data Scientist



Data Analyst




STILL WAITING
FOR MY ML MODEL TO TRAIN
makeameme.org

GPU vs CPU

GPUs (Graphics Processing Units) and **CPUs (Central Processing Units)** are designed with different architectures and for different purposes, which makes one more suitable than the other for specific tasks. Here's why GPUs are considered more powerful than CPUs for certain operations:

GPU vs CPU

Parallelism:

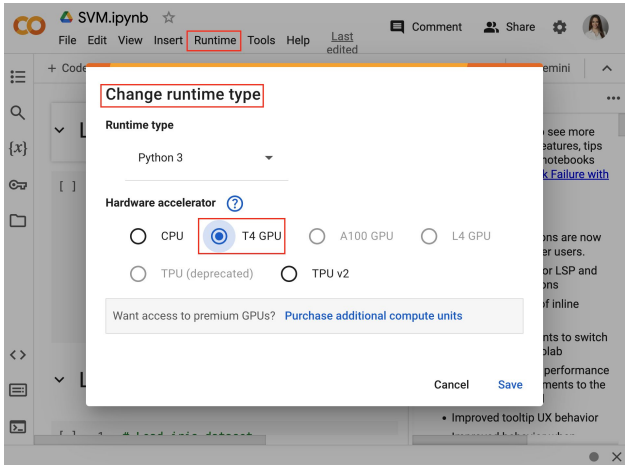
- GPUs: Originally designed to render graphics, GPUs are built to handle multiple tasks simultaneously. They have thousands of smaller cores designed for parallel processing, making them adept at handling multiple computations concurrently.
- CPUs: Typically have fewer cores (e.g., 4, 8, or 16 cores in modern CPUs). They are optimized for tasks that require sequential processing and are better suited for general-purpose tasks and managing system operations.

GPU vs CPU

Task Specialization:

- GPUs: Specialized for compute-intensive tasks. They excel at matrix operations and floating-point arithmetic, which are common in graphics rendering and deep learning.
- CPUs: Designed as general-purpose processors, capable of handling a variety of tasks, from managing I/O operations to running complex algorithms. They're more versatile but may not be as efficient as GPUs for specific parallel tasks.

What is Google Colab?



aws Azure Google Cloud

EC2 VM GCE

TensorFlow vs Keras vs Pytorch

TensorFlow:

- **Description**: TensorFlow is an open-source deep learning framework developed by the Google Brain team. It is designed to help researchers and developers work with large-scale machine learning and deep learning models.
- **Features**:
  - Eager Execution: Allows operations to be executed immediately as they are called, which aids in debugging and prototyping.
  - TensorBoard: A visualization tool for understanding, debugging, and optimizing TensorFlow programs.
  - TensorFlow Lite: Enables deployment of lightweight ML models on mobile and embedded devices.
  - Distributed Training: TensorFlow can run on multiple GPUs and TPUs, enabling faster model training.

TensorFlow vs Keras vs Pytorch

Keras:

- **Description**: Keras is an open-source high-level neural network API that's built on top of TensorFlow, Theano, or CNTK. It is designed to be user-friendly and modular, allowing for easy and fast prototyping.
- **Features**:
  - Modularity: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
  - Easy to Extend: You can write custom building blocks to express new ideas for research.
  - User-friendly: Designed for human beings, not machines, making it easy to learn and use.

In practice, with the integration of Keras into TensorFlow 2.x and the strong support and updates provided by TensorFlow, most developers and researchers now use the tf.keras module within TensorFlow rather than standalone Keras.

TensorFlow vs Keras vs Pytorch

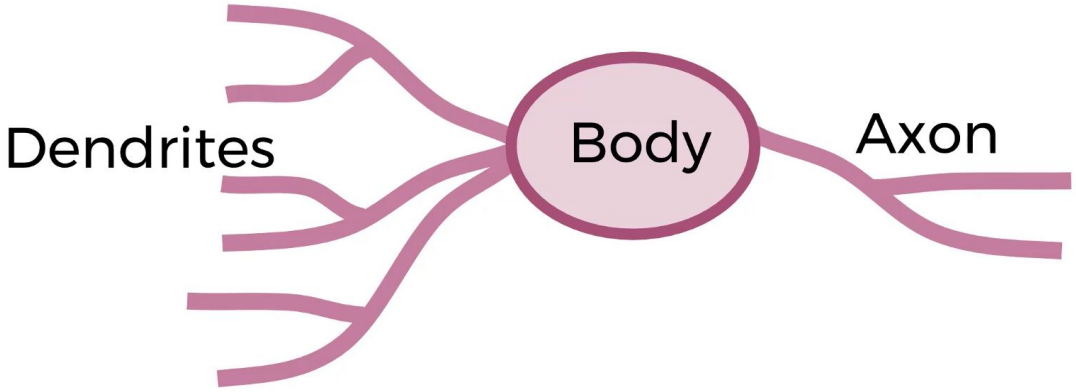
PyTorch:

- **Description**: PyTorch is an open-source deep learning platform developed by Facebook's AI Research lab. It is known for providing two of the most essential features for research - flexibility and dynamic computation graphs, which makes it particularly useful for deep learning research.
- **Features**:
  - Dynamic Computation Graphs: Known as Autograd, it allows for dynamic modification of the graph on the fly. This is particularly useful for models that require iterative computation.
  - Native Python Support: It blends seamlessly with the Python programming ecosystem.
  - TorchServe: For serving PyTorch models in production.
  - Distributed Training: Like TensorFlow, PyTorch supports multi-GPU training.

Neural Network

- A neural network is composed of neurons
- Artificial Neural Networks (ANN) are inspired by biological neural networks

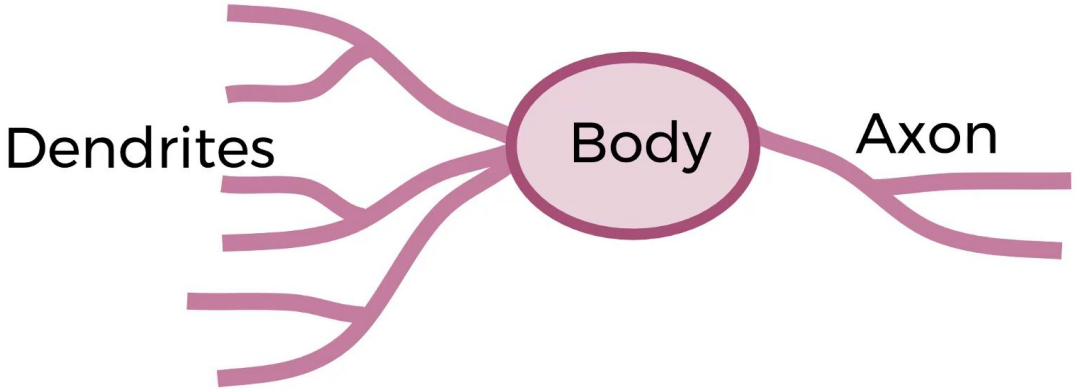
A biological neuron



In a simplified way:

- Dendrites feed the cell body through electrical signals
- The response is then passed through the axon

A biological neuron



In a simplified way:

- Dendrites feed the cell body through electrical signals
- The response is then passed through the axon

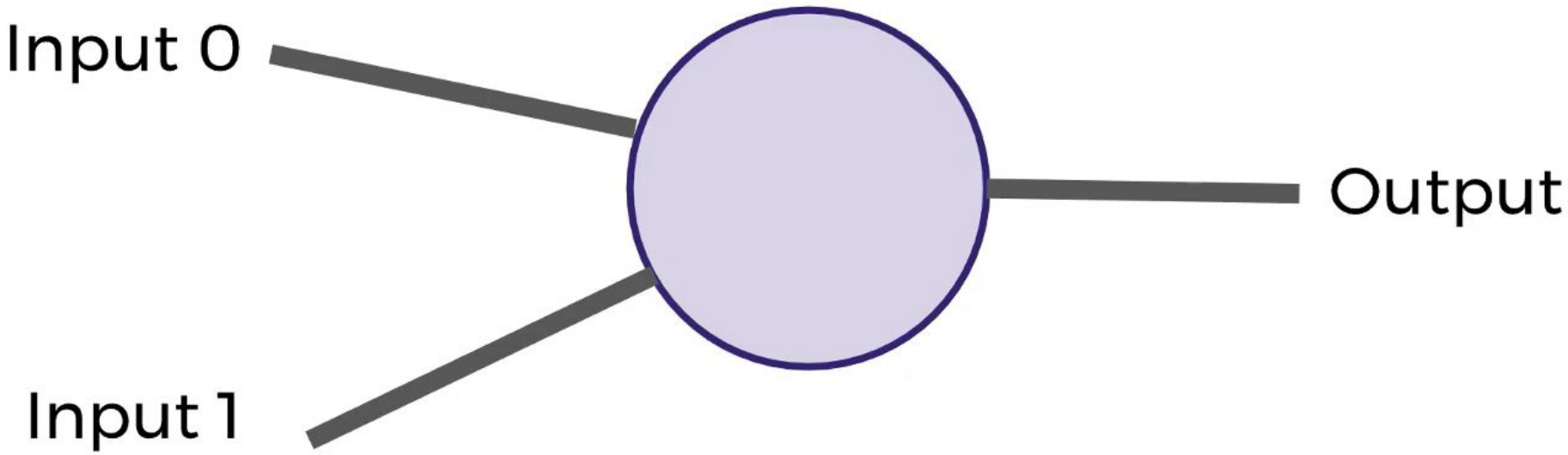
Perceptron

An artificial neuron is called a **Perceptron**

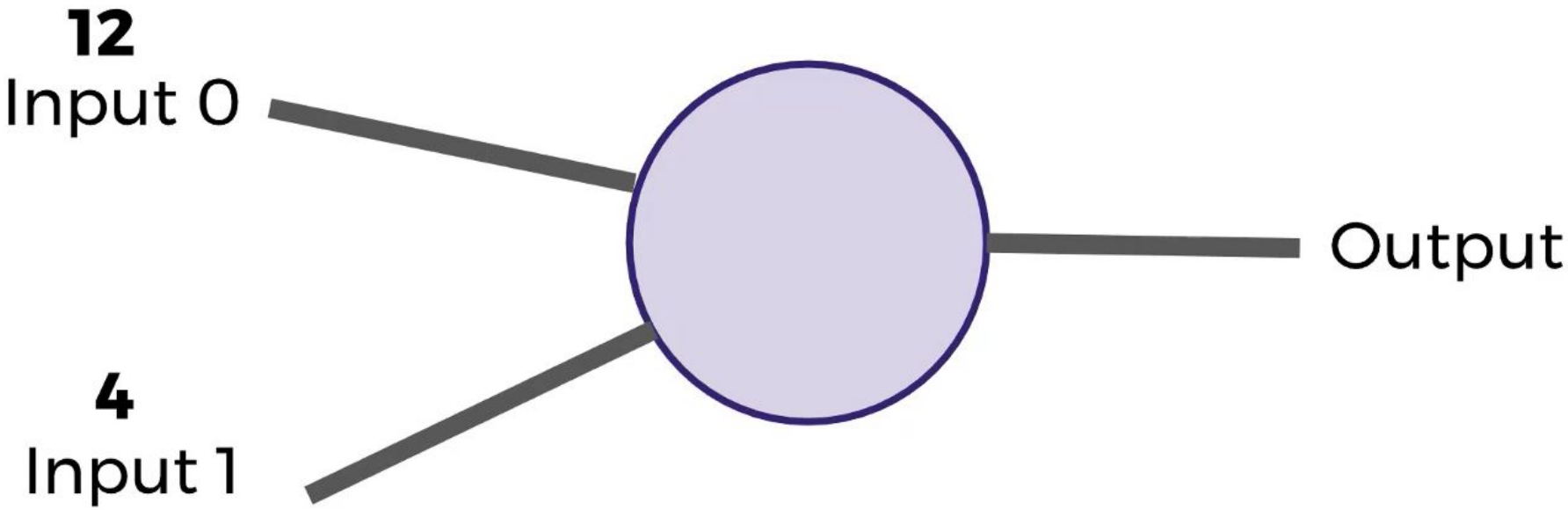
Perceptron

- A perceptron is one of the simplest forms of a neural network
- It can be considered the building block for more complex networks

Perceptron

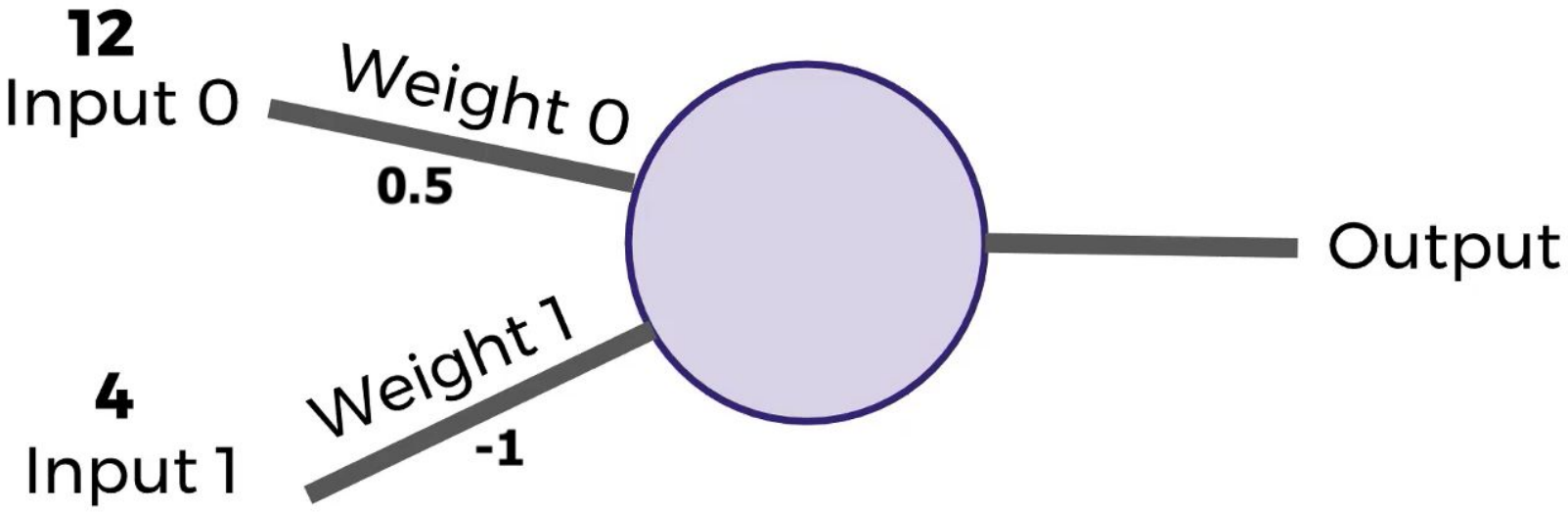


Perceptron



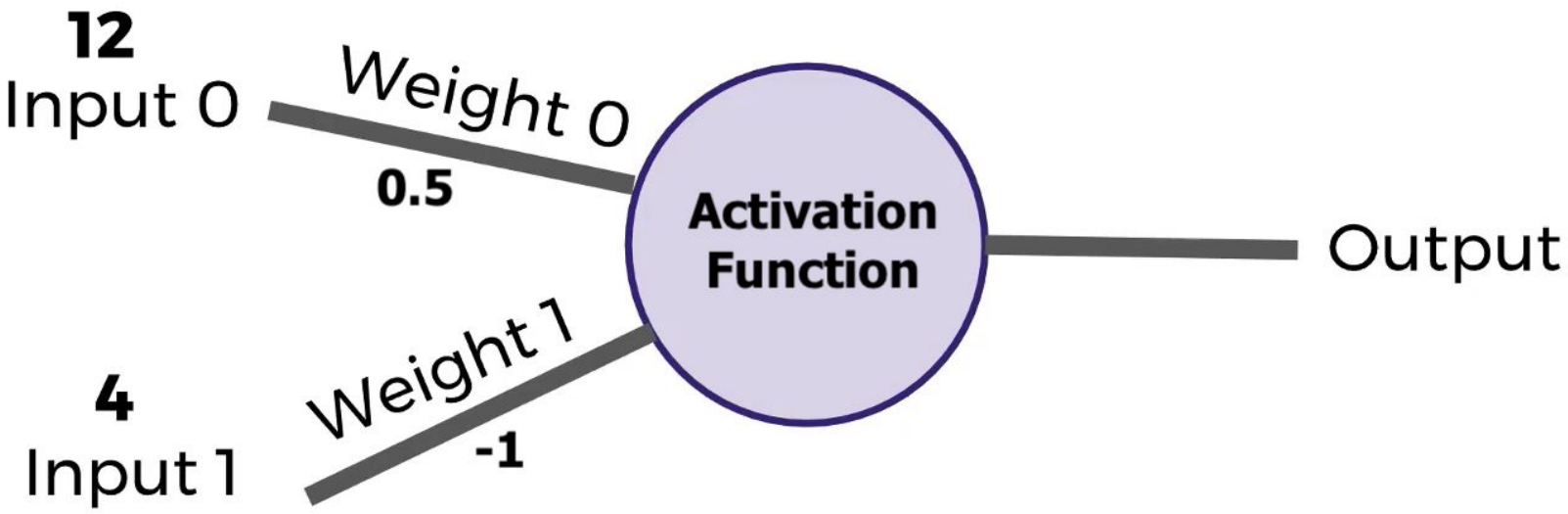
- Like any ML model, we have input values

Perceptron



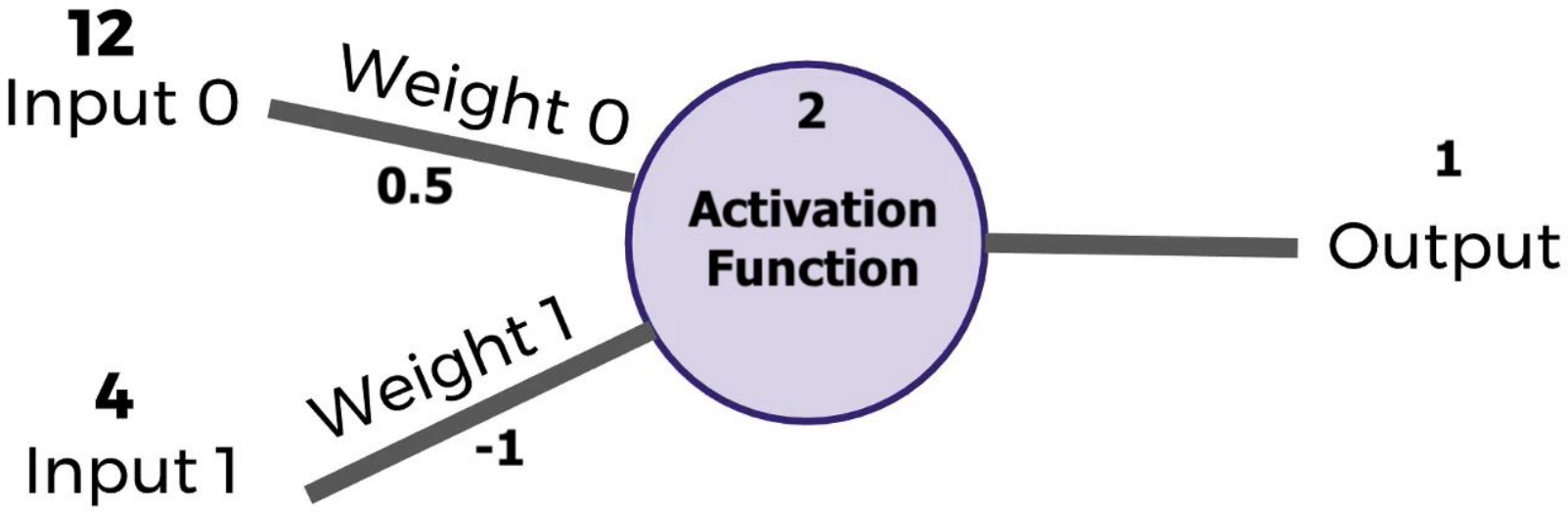
- Then, the input values are multiplied by weights
- These weights are initialized randomly

Perceptron



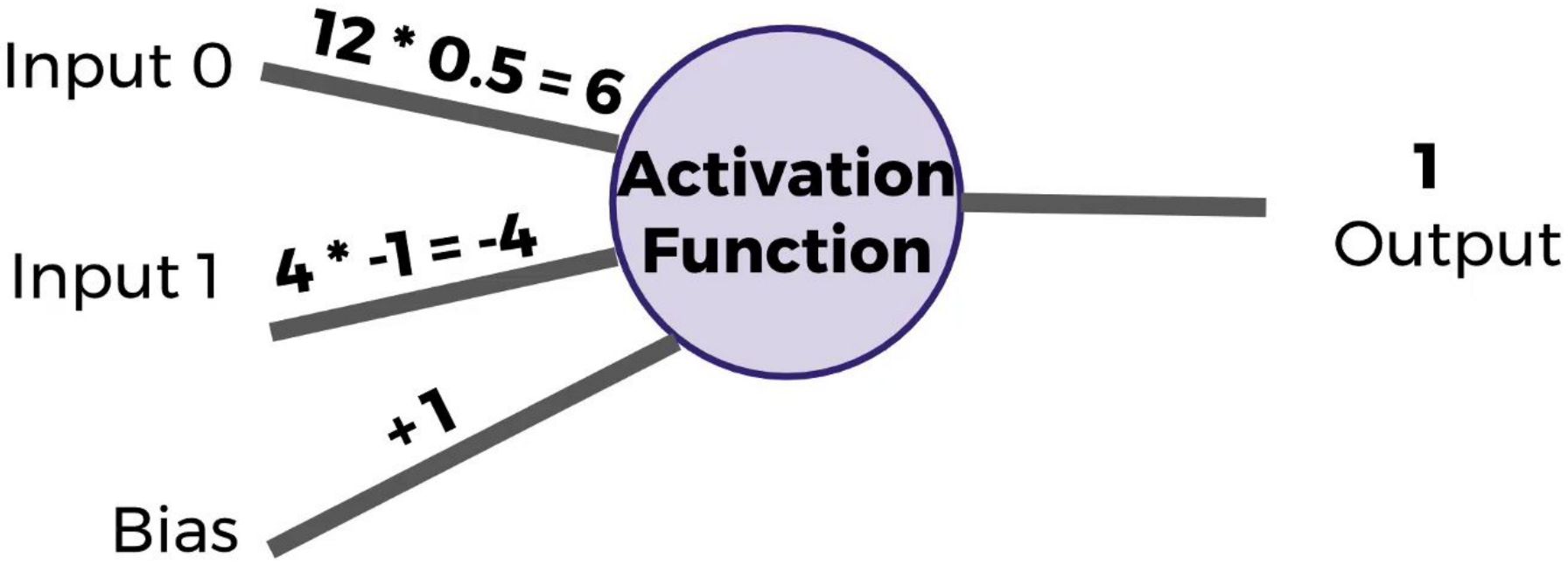
- The result (in this example 12*0.5+4*-1=2) is passed through an activation function
- There are many activation functions

Perceptron



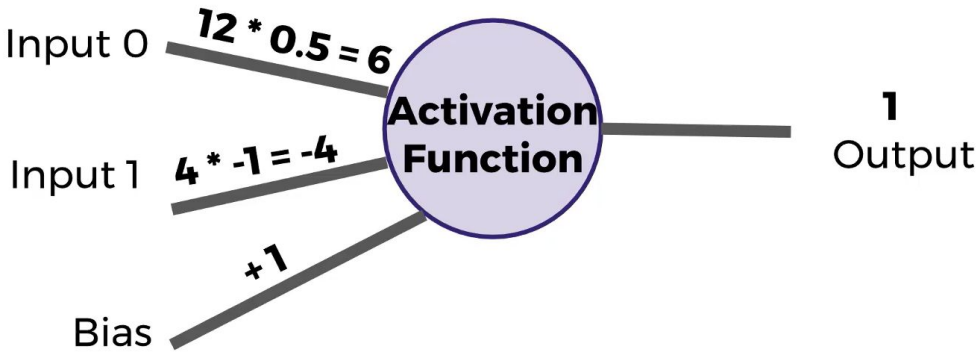
- The result (in this example 12*0.5+4*-1=2) is passed through an activation function
- There are many activation functions - **Example:** Positive = 1, Negative = 0

Perceptron



- Bias added to avoid mathematical problems
- Additional parameter that provides flexibility to the model

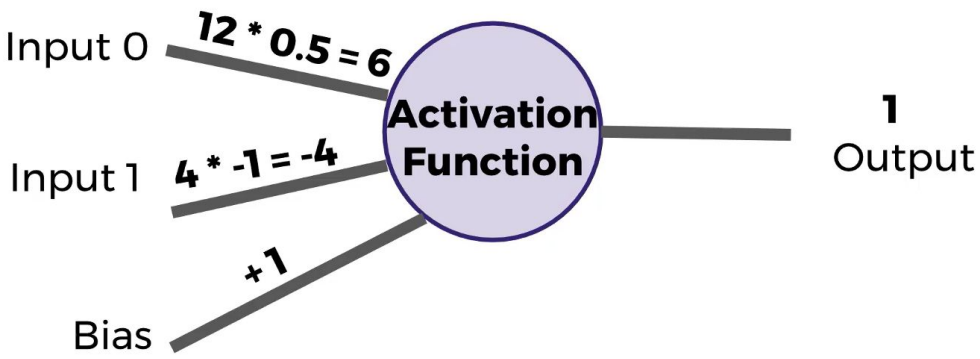
Perceptron



- In math terms, we $\sum_{i=0}^{n} w_i x_i + b$

When we have a neural network composed of several perceptrons, it is extended to matrix form

Perceptron

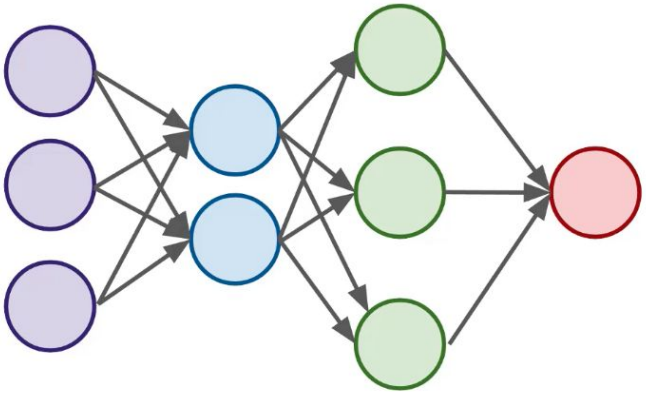


A perceptron is simply a Linear model with a twist: Activation Function

- In math terms, we $\sum_{i=0}^{n} w_i x_i + b$

When we have a neural network composed of several perceptrons, it is extended to matrix form

An artificial neural network

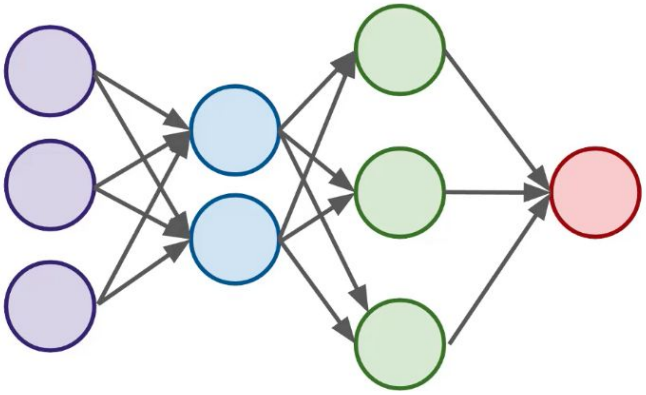


Parts:

- Input layer: Actual data values
- Hidden layers (2 in this case): Deep network
- Output Layer: Final estimate

As the number of layers increases, the level of abstraction increases

An artificial neural network



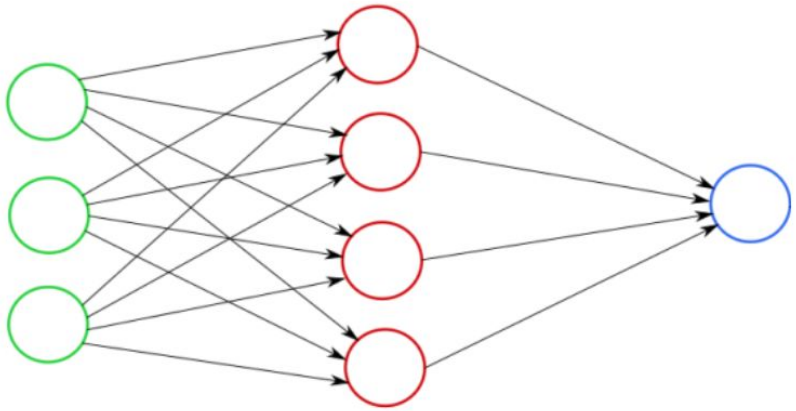
NN send data to a space where everything is linearly separable

Parts:

- Input layer: Actual data values
- Hidden layers (2 in this case): Deep network
- Output Layer: Final estimate

As the number of layers increases, the level of abstraction increases

👮‍♀️ Checkpoint



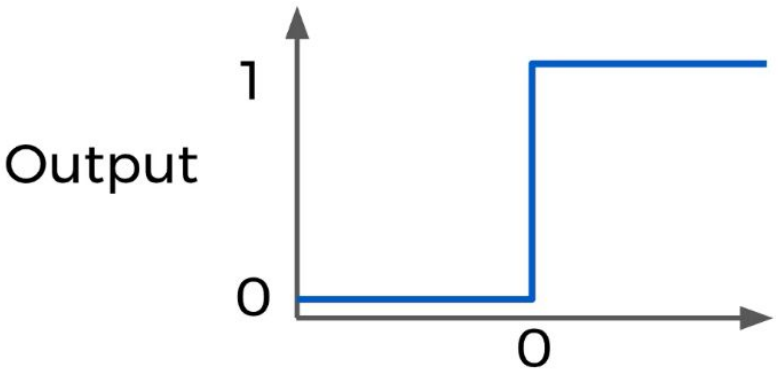
How many hidden layers are there in this NN?

Activation function

- A mathematical function applied to the output of each neuron or node
- It introduces non-linearity into the model, allowing the neural network to represent more complex functions and learn from the error

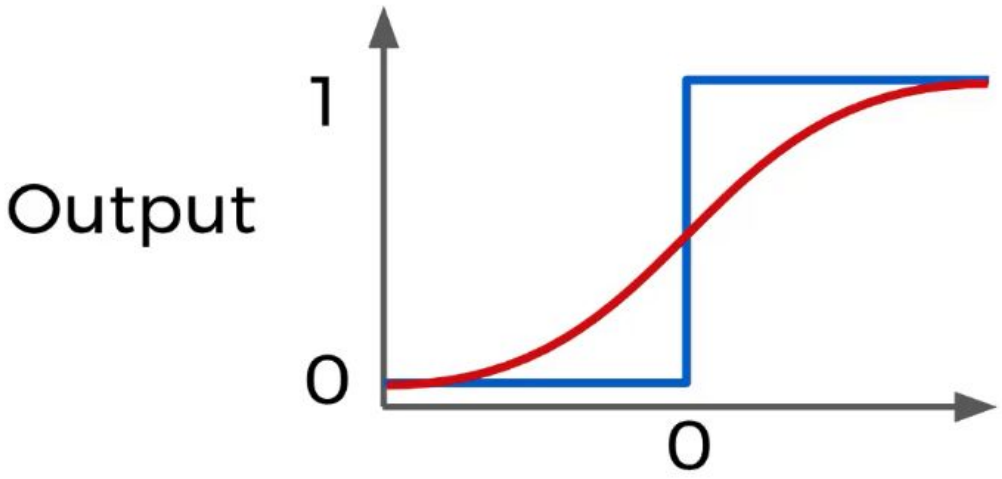
Activation function

- A mathematical function applied to the output of each neuron or node
- It introduces non-linearity into the model, allowing the neural network to represent more complex functions and learn from the error



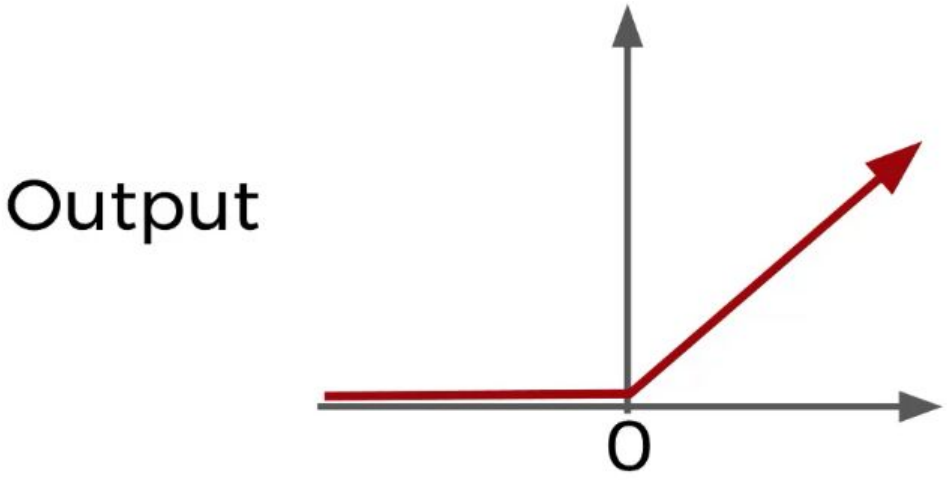
- Assigns 0 if the value is less than 0, assigns 1 if the value is greater than or equal to zero
- Very drastic, small changes are not reflected

Example: Sigmoid function



- Sigmoid function:
  f(x) = 1/1+e^(-x)
- Very useful depending on the task

Example: ReLu function



- ReLU (Rectified Linear Unit): f(x) =max(0,x)
- Although it is simple, it performs very well in many situations

Common Activation Functions for Output layers

Binary classification

- **Neurons**: 1
- **Activation Function**: Sigmoid (to squash output between 0 and 1, representing the probability of the positive class).
- **Example**: Email spam detection (spam or not spam).

Multi-class Classification (Single Label)

- **Neurons**: Equal to the number of classes.
- **Activation Function**: Softmax (to output a probability distribution over multiple classes).
- **Example**: Handwritten digit recognition (0 to 9).

Multi-label Classification

- **Neurons**: Equal to the number of classes.
- **Activation Function**: Sigmoid for each neuron (since each class prediction is treated as an independent binary classification).
- **Example**: Image tagging (an image can have multiple tags like "beach", "sunset", "people").

Regression

- **Neurons**: 1 (for single-output regression) or more (for multi-output regression).
- **Activation Function**: Usually linear or none. However, if the regression output has a known bounded range, activation functions like sigmoid or tanh might be used to bound the output.
- **Example**: Predicting house prices.

Time Series Forecasting

- **Neurons**: Can vary depending on the forecasting horizon (e.g., next value, next 10 values).
- **Activation Function**: Often linear, especially if the values can range widely. For bounded values, other activations might be used.
- **Example**: Stock price prediction for the next week.

Common Activation Functions for Output layers

Autoencoders (for dimensionality reduction, denoising, etc.)

- **Neurons**: Varies based on the desired reduced dimensionality or the structure of the data.
- **Activation Function**: Could be linear, sigmoid, tanh, etc., depending on the nature of the data and the specific use-case.
- **Example**: Image denoising.

Generative Models (like GANs)

- **Neurons**: Varies based on the structure and size of the data being generated.
- **Activation Function**: Can vary, but for image generation tasks, the tanh activation is commonly used for the generator's output layer.
- **Example**: Generating art images.

Sequence-to-Sequence Problems (like translation)

- **Neurons**: Often depends on the vocabulary size of the target language (for tasks like translation) or other sequence length details.
- **Activation Function**: Softmax, especially when predicting tokens from a vocabulary.
- **Example**: Translating English to French.

Common Activation Functions for Hidden layers

ReLU (Rectified Linear Unit)

- **Formula**: f(x) = max(0,x)
- **Pros**: Fast to compute, reduces vanishing gradient problem, generally works well in practice for deep networks.
- **Cons**: Can suffer from "dead neurons" where some neurons never activate and therefore never update.

Leaky ReLU

- **Formula**: f(x) = x for x>0 and f(x) = ax for x<=0
- **Pros**: Addresses the "dead neuron" problem of ReLU by allowing a small gradient for negative values.

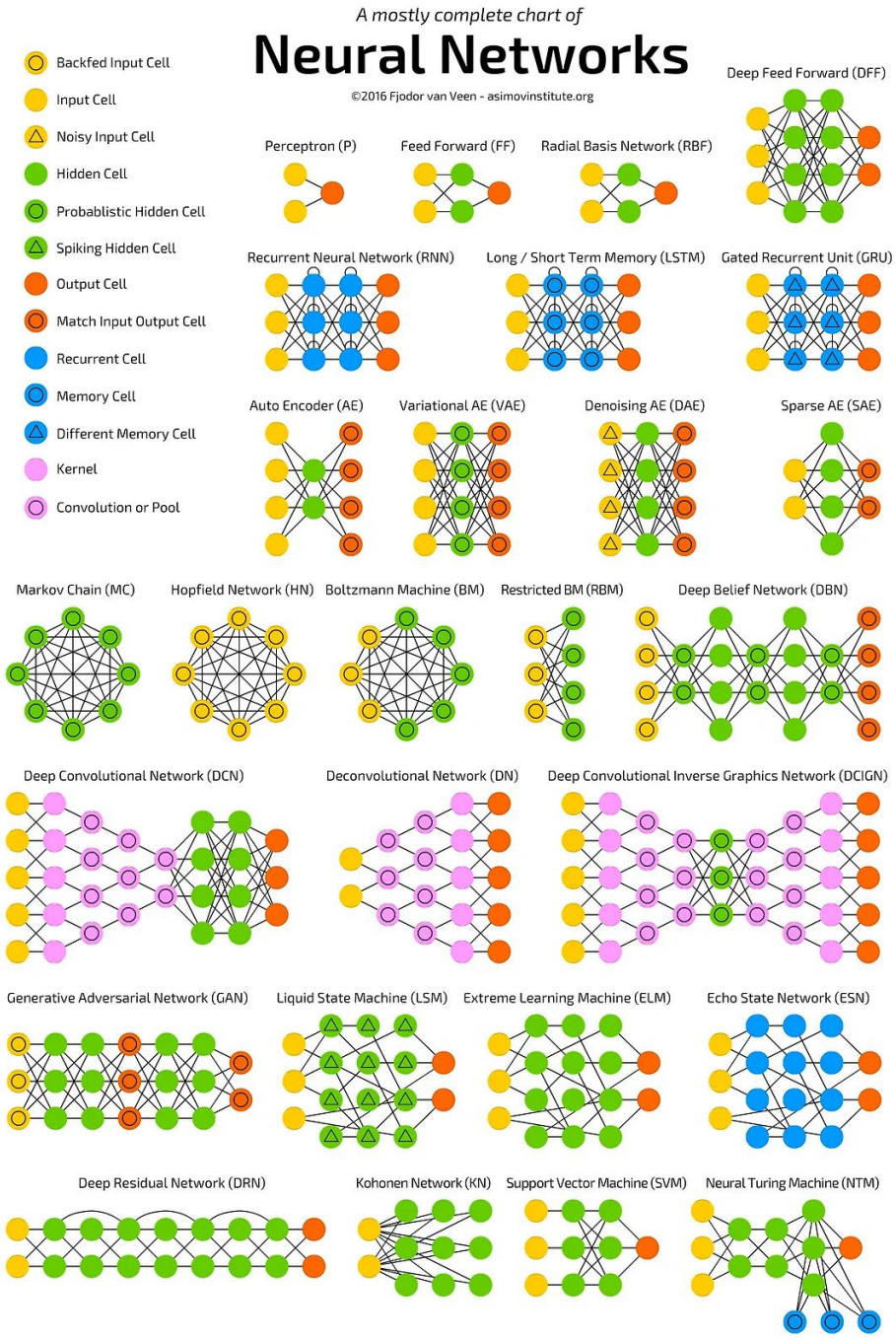
Common Activation Functions for Hidden layers

ReLU (Rectified Linear Unit)

- **Formula**: f(x) = max(0,x)
- **Pros**: Fast to compute, reduces vanishing gradient problem, generally works well in practice for deep networks.
- **Cons**: Can suffer from "dead neurons" where some neurons never activate and therefore never update.

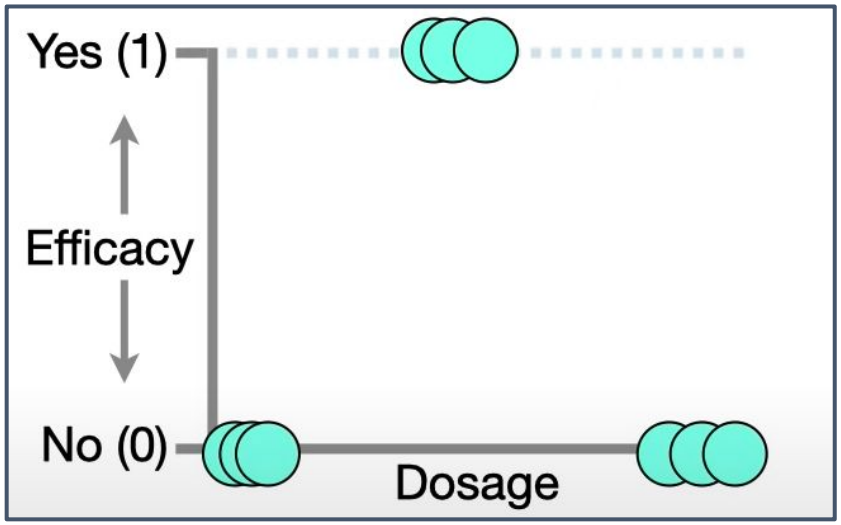
Leaky ReLU

- **Formula**: f(x) = x for x>0 and f(x) = ax for x<=0
- **Pros**: Addresses the "dead neuron" problem of ReLU by allowing a small gradient for negative values.
    - **Consider Training Dynamics:** Some activation functions can impact training speed and convergence. For example, dead neurons in ReLU might slow down training, and you might prefer Leaky ReLU in such cases.
    - **Computational Considerations:** ReLU is computationally simpler than tanh or sigmoid, which might matter for very large models or datasets.
- **General Rule of Thumb:** ReLU and its variants (like Leaky ReLU) are often good starting points for feedforward and convolutional neural networks due to their computational efficiency and performance in practice.
- **Problem Nature:** If you know your data distribution or the kind of relationships (linear/non-linear) your network needs to model, you can make informed decisions. For instance, if you expect non-linearities, ReLU, tanh, or sigmoid might be useful.
- **Network Depth:** For very deep networks, ReLU and its variants might be preferred due to the vanishing gradient problem with sigmoid and tanh.
- **Historical/Previous Success:** If an activation function has been shown to work well on similar problems or datasets, it's worth trying it out.
- **Experimentation: Often,** the choice of activation function is empirical. Running experiments with different activation functions and evaluating the performance on validation data can guide the choice.



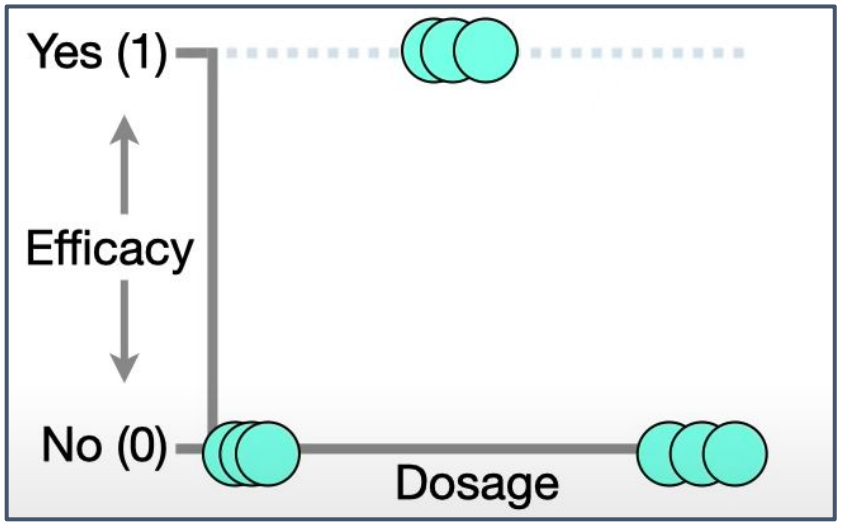
- [Wikipedia]
- [Tensorflow Neural Network Playground]

When to use Deep Learning?



- Complex data
- High dimensional data

When to use Deep Learning?



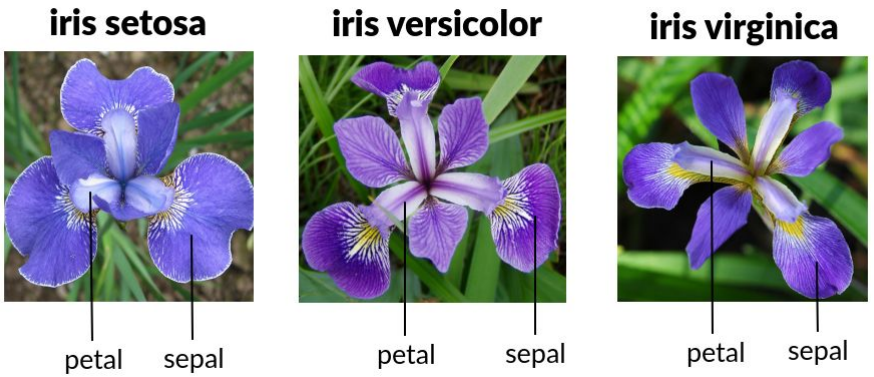
- Complex data
- High dimensional data

A perceptron is simply a Linear classifier with a twist: Activation Function

NN send data to a space where everything is linearly separable

Me using neural network for simple regression problem

An artificial neural network



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

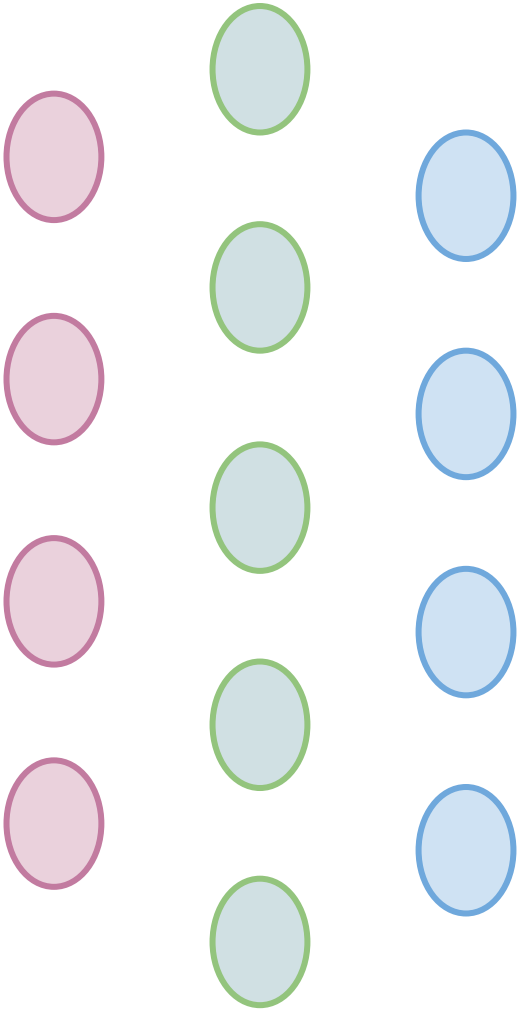
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Input layer: Four neurons corresponding to the four features

An artificial neural network

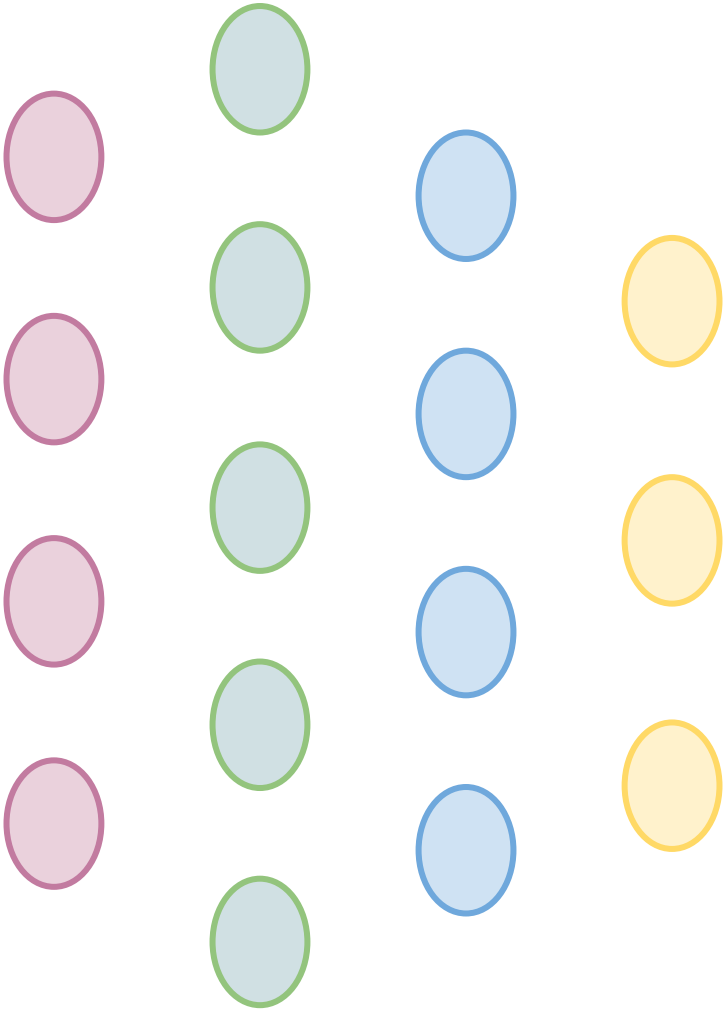
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



Hidden layers: Number and size of hidden layers can vary based on the complexity of the model

An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



Output layer: Three, corresponding the three classes of Iris flowers

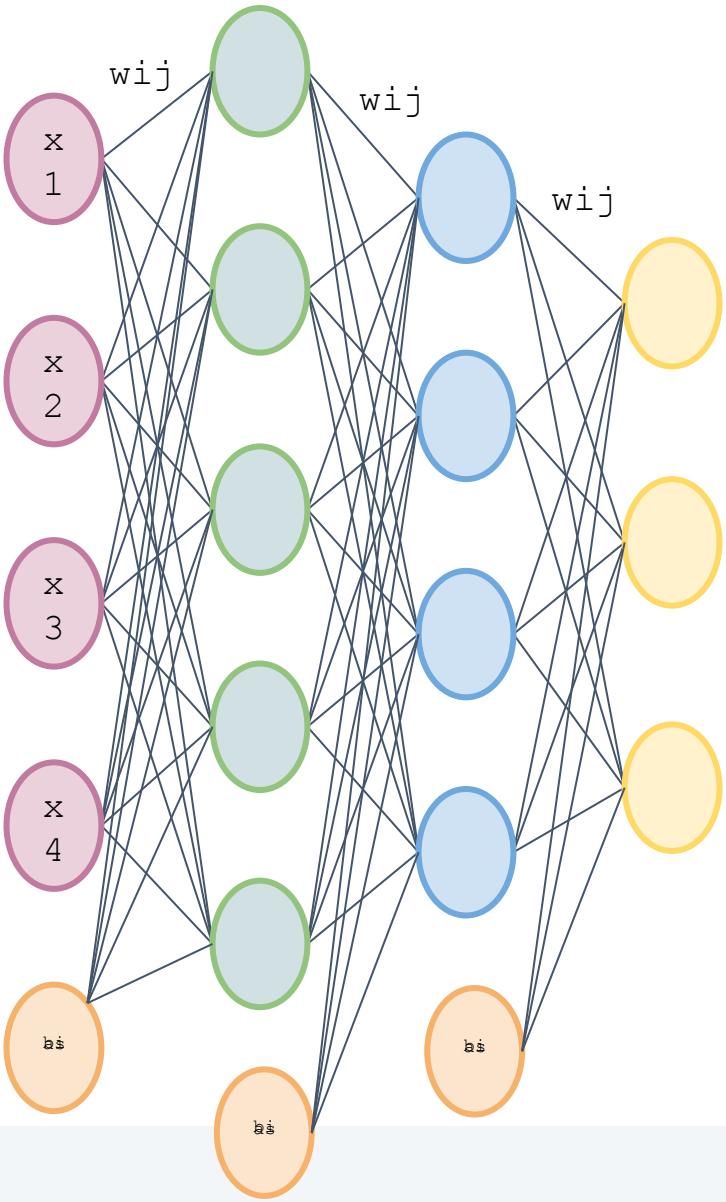
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 1:

Initialization

Weights and biases in the network are usually initialized, often with small random values



Bias: When illustrating a neural network, the bias is often represented as an additional node in each layer, except for the output layer. In PyTorch, when you use nn.Linear, the bias is automatically added for you.

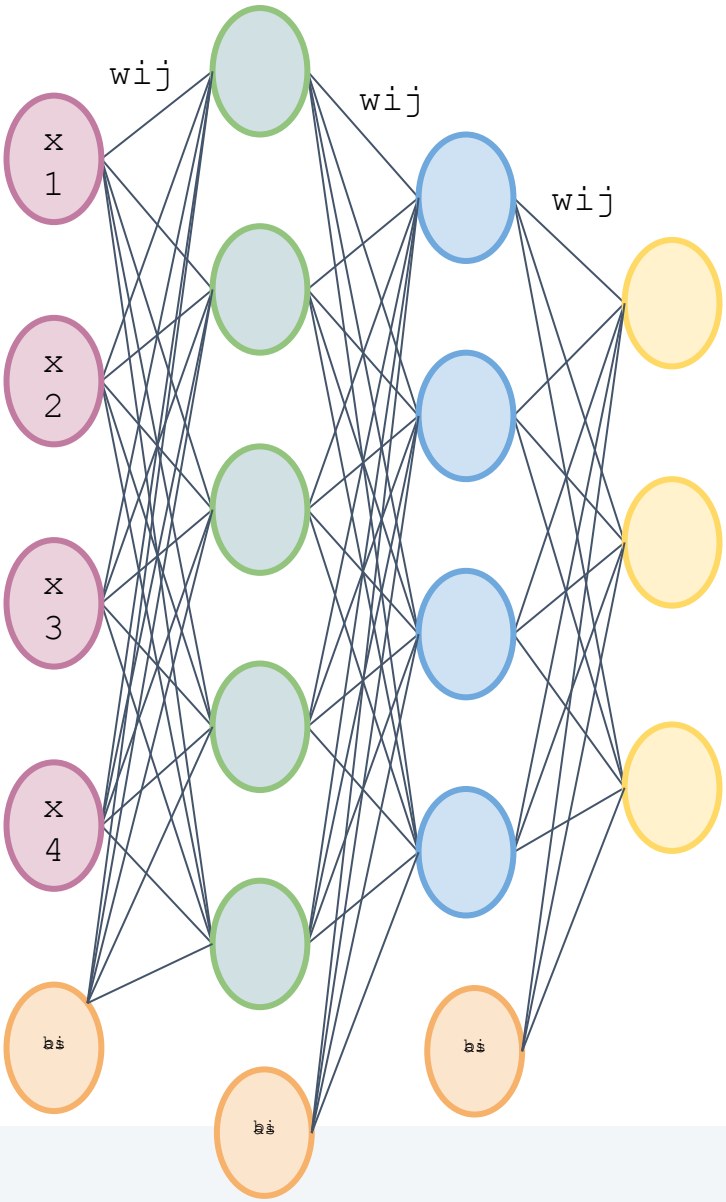
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 2:

Input

Data is fed into the network



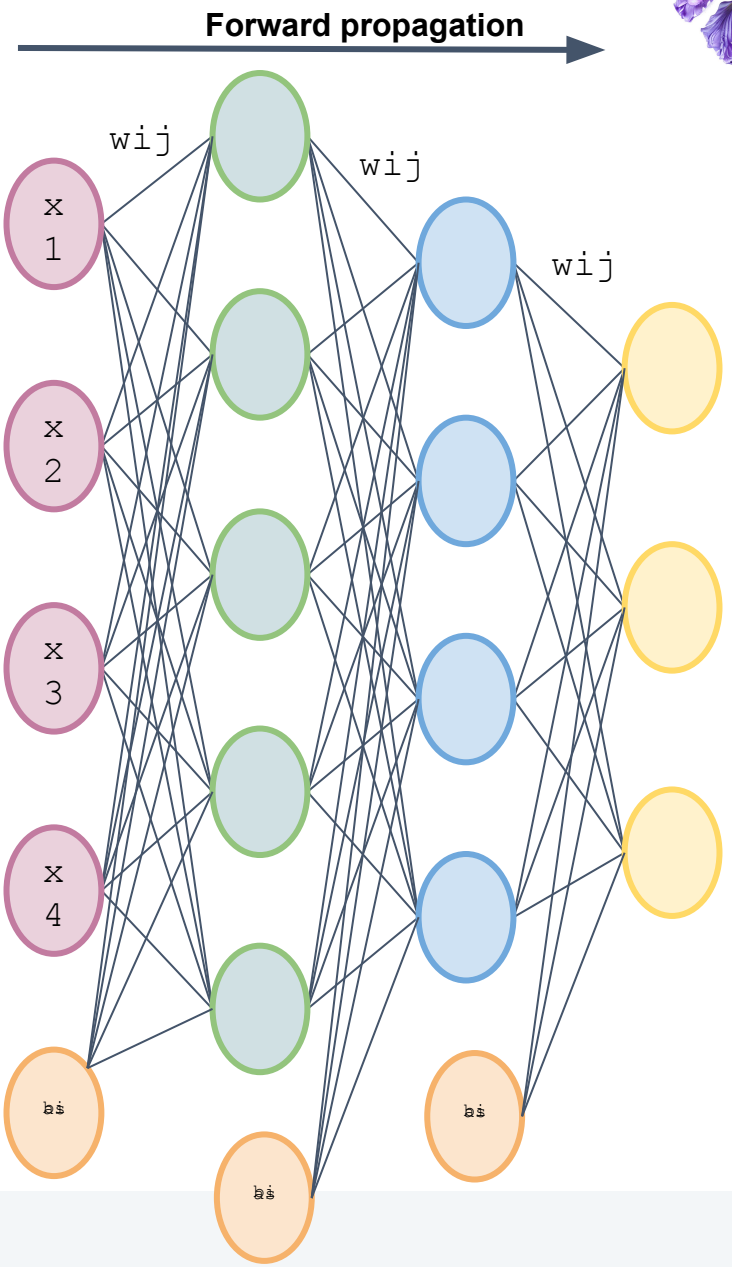
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3:

Forward propagation

- Starting from the input layer, each neuron computes a weighted sum of inputs
- This sum is then passed through an activation function to produce the neuron's output
- This output becomes the input for the next layer, and the process is repeated for each subsequent layer in the network



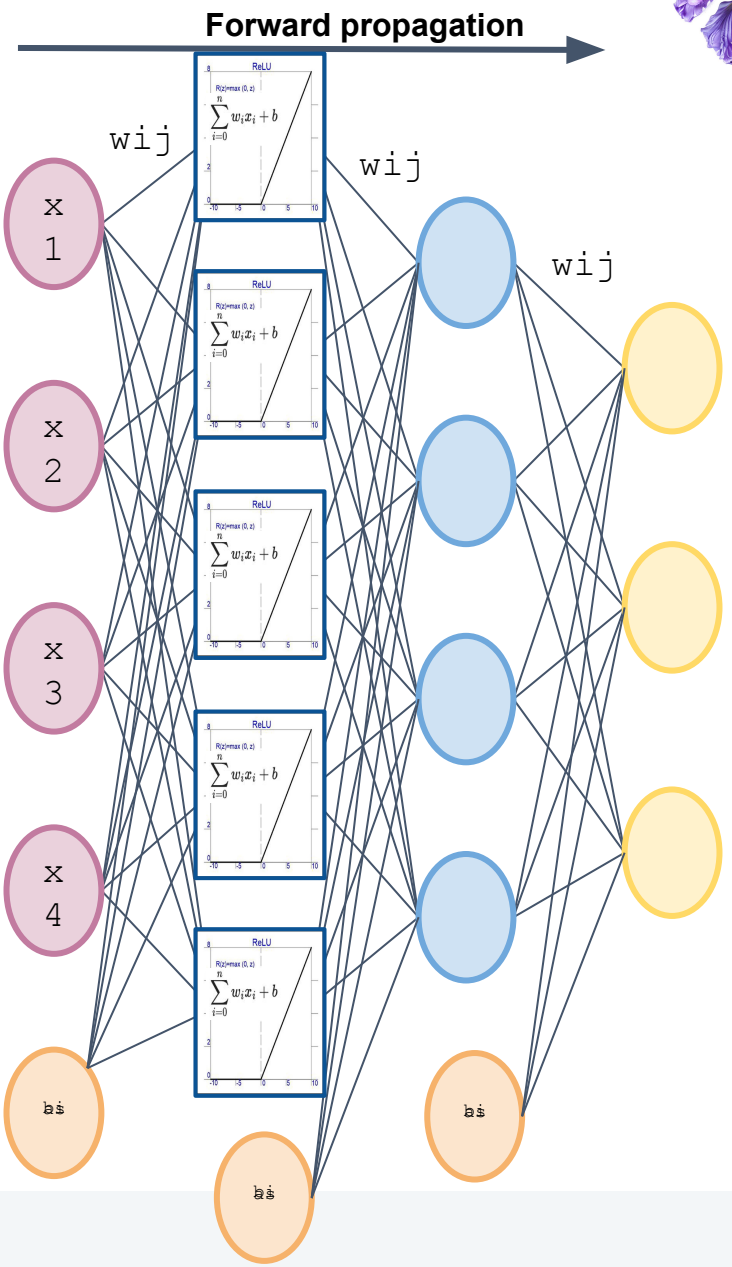
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3:

Forward propagation

- Starting from the input layer, each neuron computes a weighted sum of inputs
- This sum is then passed through an activation function to produce the neuron's output
- This output becomes the input for the next layer, and the process is repeated for each subsequent layer in the network



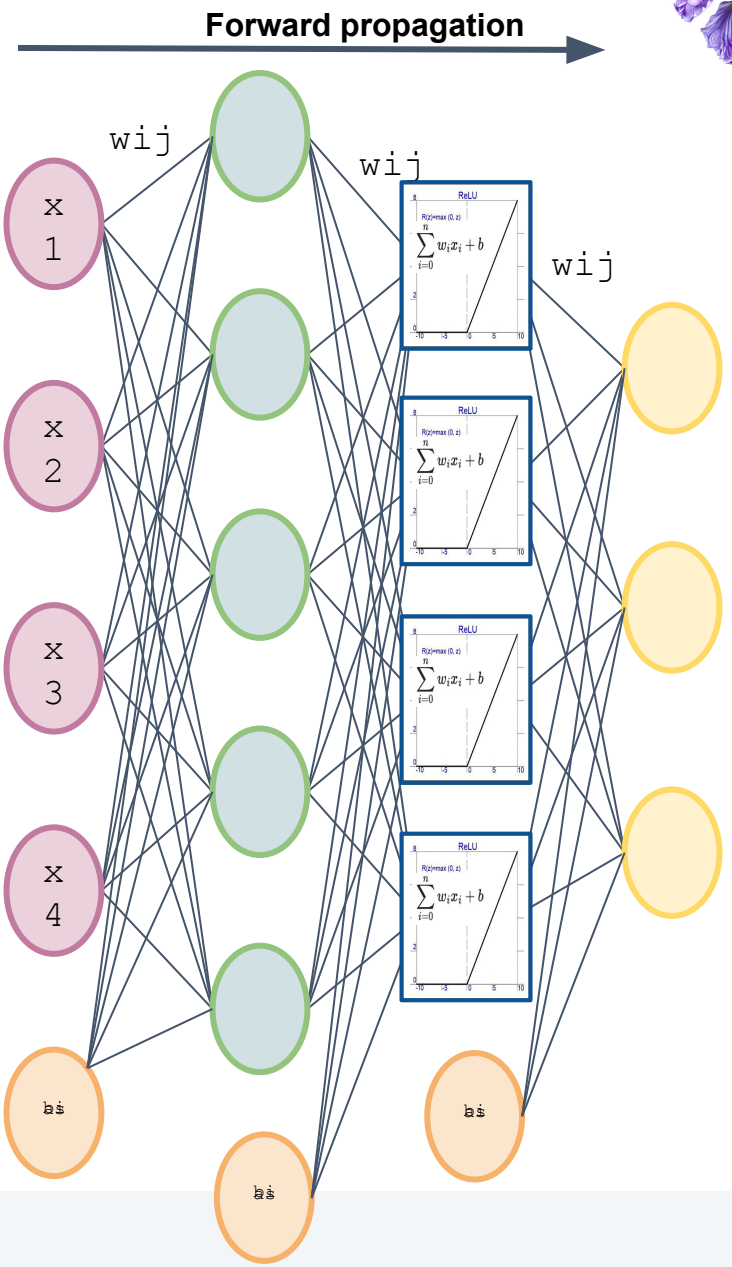
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3:

Forward propagation

- Starting from the input layer, each neuron computes a weighted sum of inputs
- This sum is then passed through an activation function to produce the neuron's output
- This output becomes the input for the next layer, and the process is repeated for each subsequent layer in the network



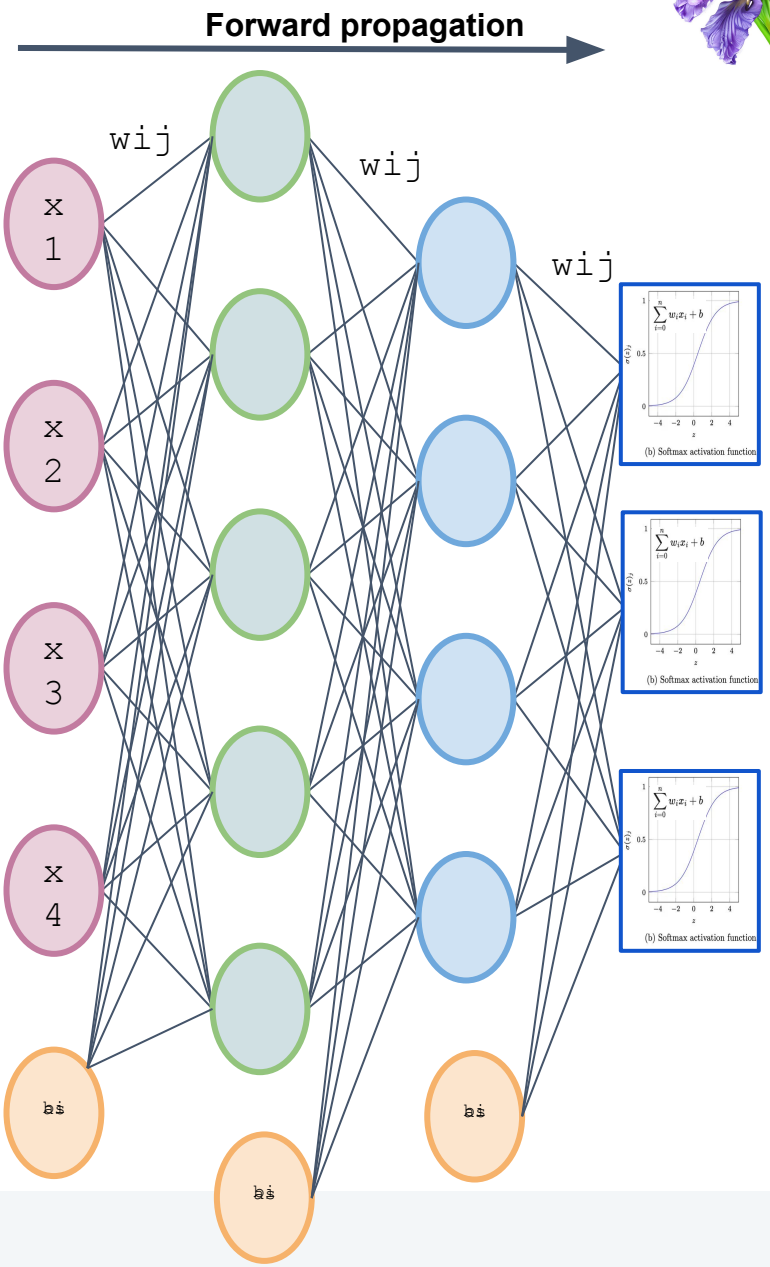
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3:

Forward propagation

- Starting from the input layer, each neuron computes a weighted sum of inputs
- This sum is then passed through an activation function to produce the neuron's output
- This output becomes the input for the next layer, and the process is repeated for each subsequent layer in the network



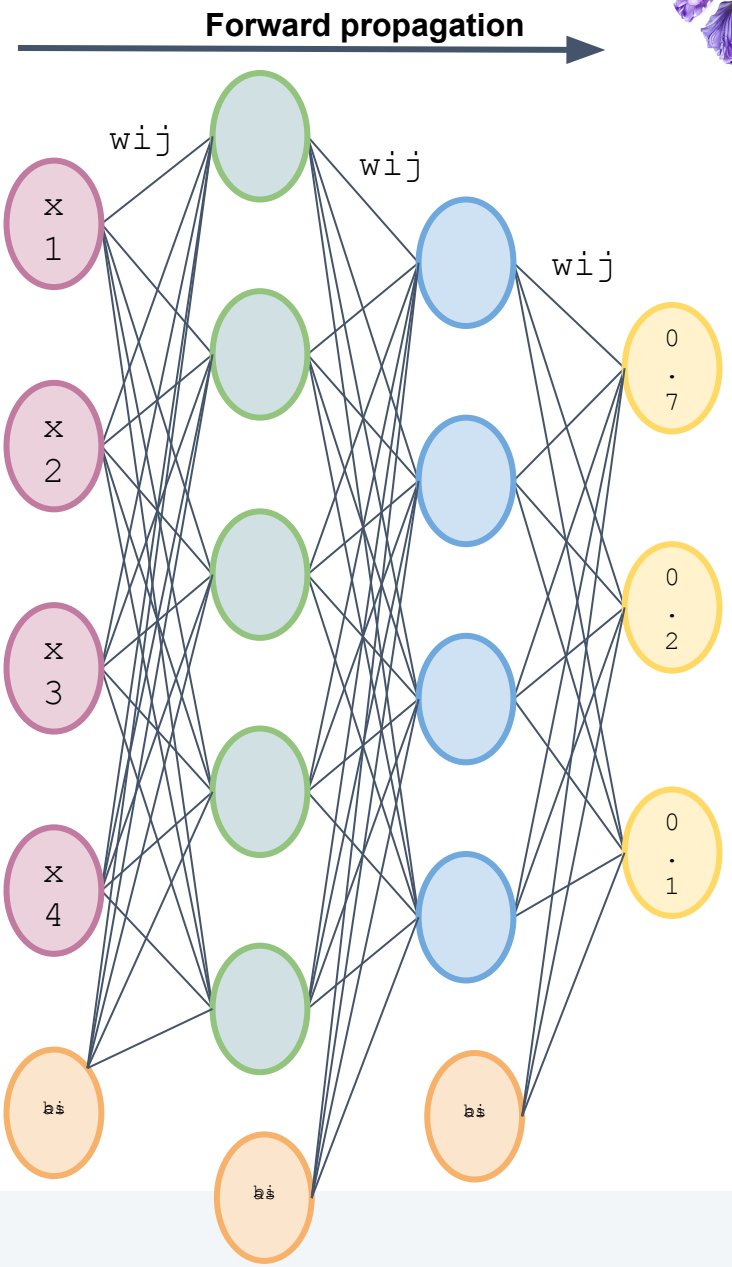
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3:

Forward propagation

- Starting from the input layer, each neuron computes a weighted sum of inputs
- This sum is then passed through an activation function to produce the neuron's output
- This output becomes the input for the next layer, and the process is repeated for each subsequent layer in the network



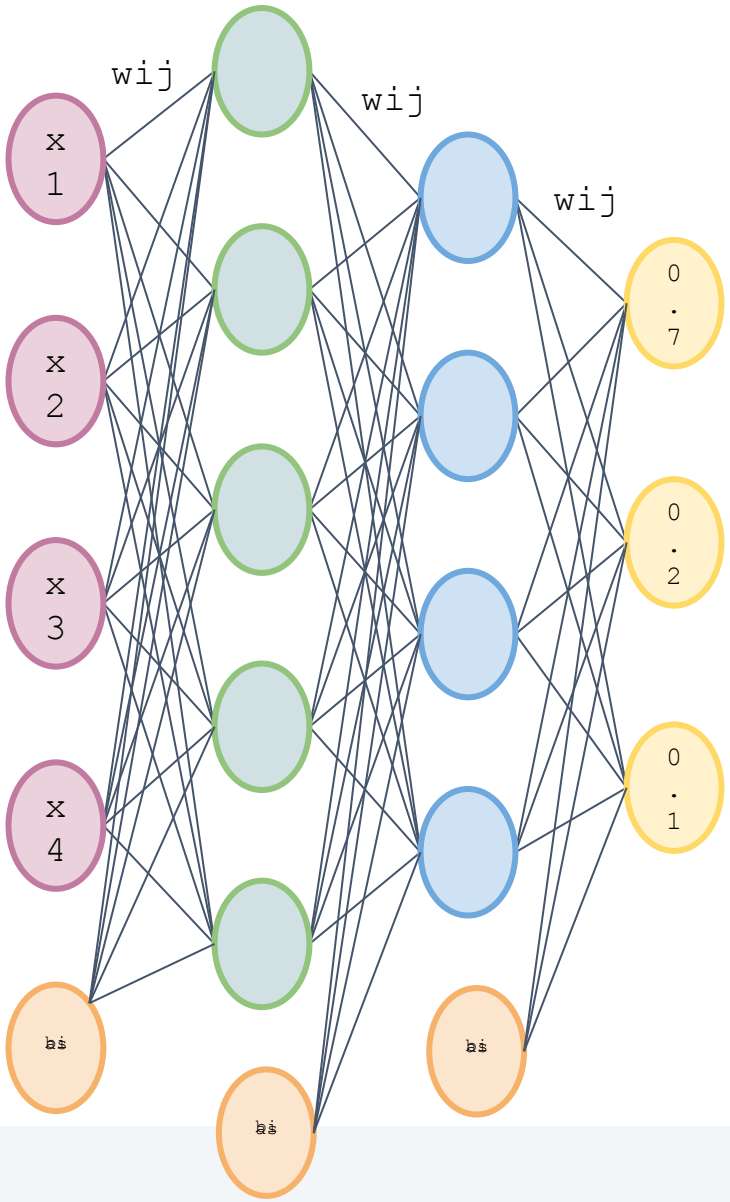
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 4:

Compute the loss

- Once you have the network's prediction (output of the last layer), you can compute the loss (or the error)
- The loss function measures the difference between predicted and target
- Regression: MSE
  Classification: Cross-Entropy



$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

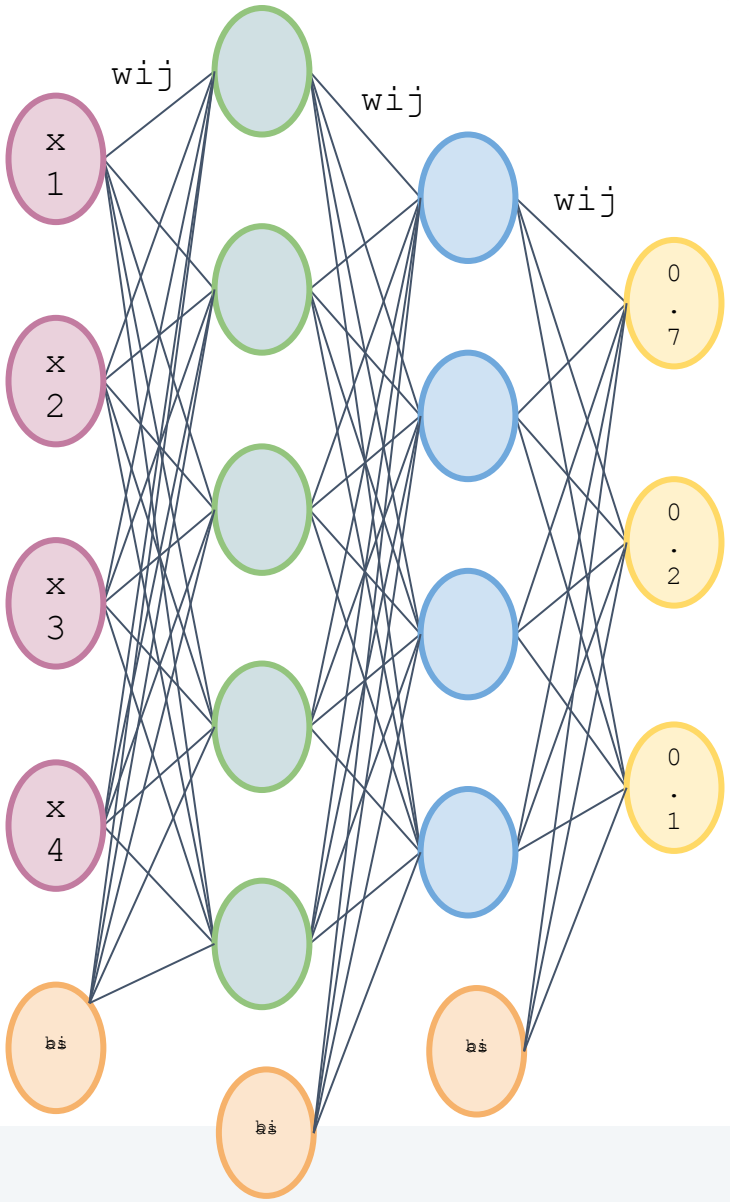
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 4:

Compute the loss

- Once you have the network's prediction (output of the last layer), you can compute the loss (or the error)
- The loss function measures the difference between predicted and target
- Regression: MSE
  Classification: Cross-Entropy



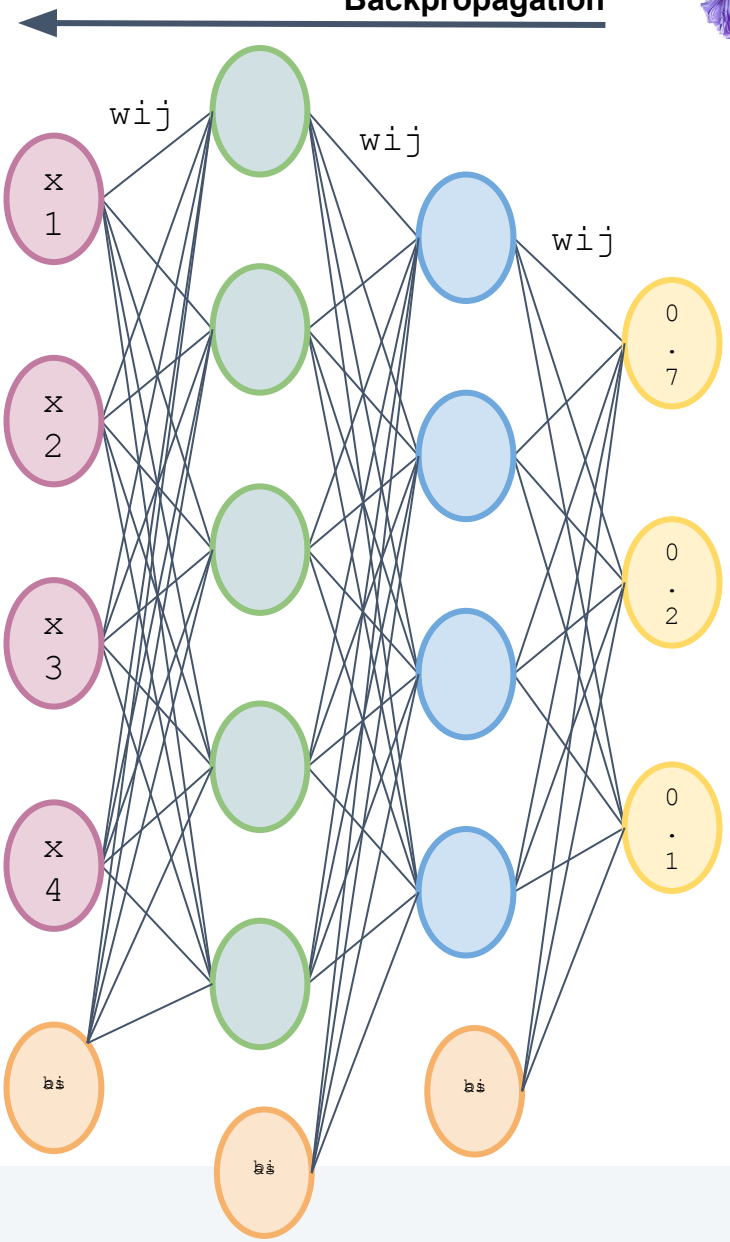
$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Objective of a Neural Network
Find weights and biases that minimizes the loss function

An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



Step 5:

Backpropagation

- Compute the **gradient** of the loss function with respect to each weight and bias in the network, ie, how much each weight and bias contributed to the error
- The computation starts from the output layer and moves backwards through the network (using the chain rule to get the gradient of each parameter)
- This process gives us a direction in which to adjust each weight and bias to minimize loss

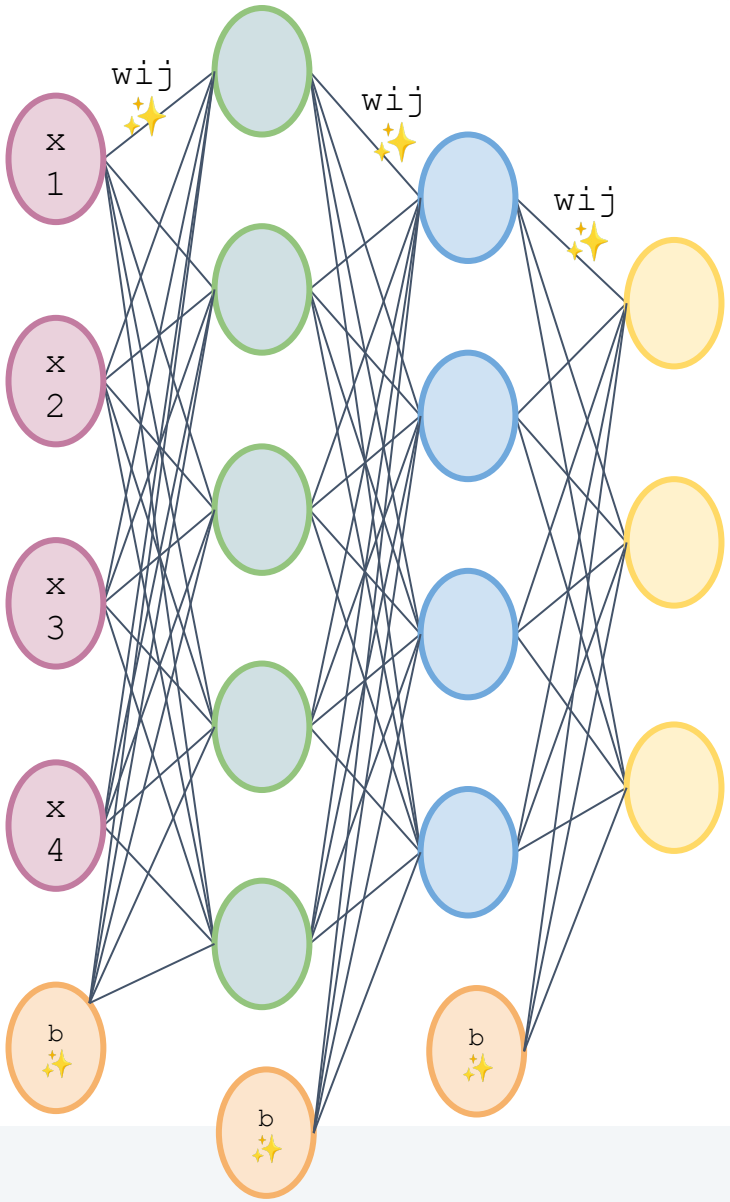
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 6:

Update weights and biases

- Using **gradient descent** or variants (Stochastic gradient descent, Adam, etc.) update the value for the weights and biases



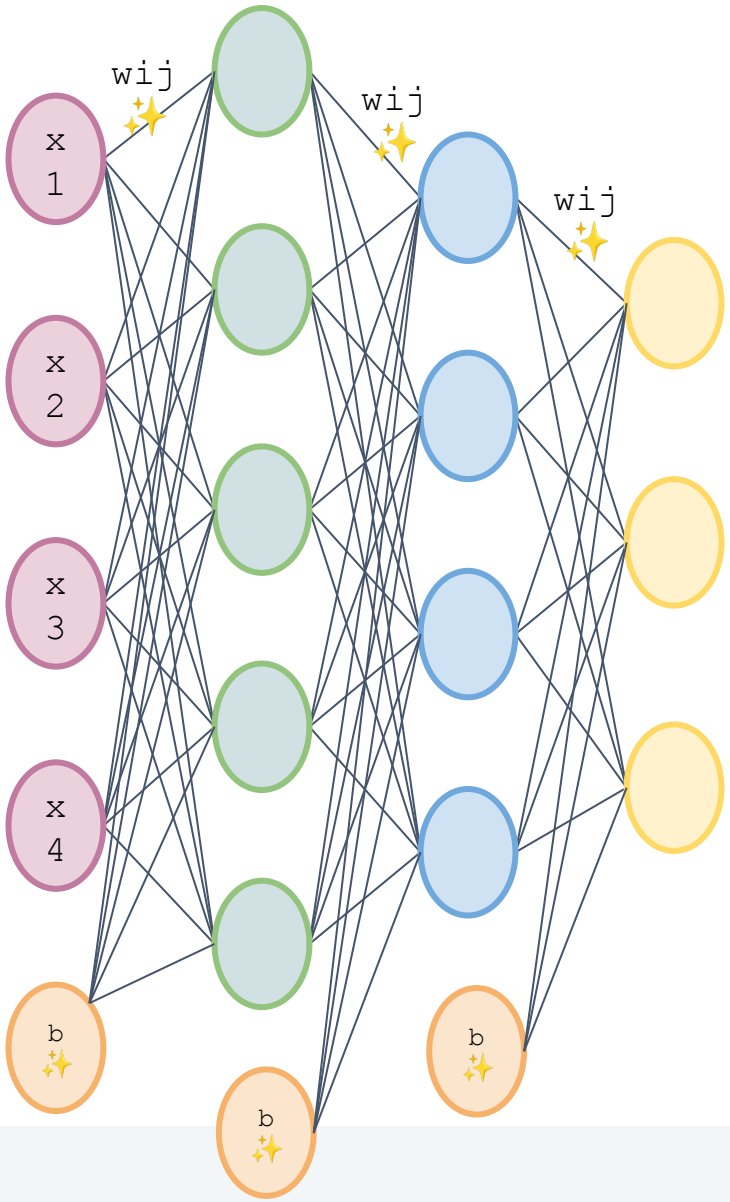
An artificial neural network

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 7:

Iteration

- Repeat steps 3-6 (forward propagation, compute loss, backpropagation, update weights and biases) until the loss in validation stops improving/increases.



Vocab

An **epoch** refers to one complete forward and backward pass of all the training examples

An artificial neural network

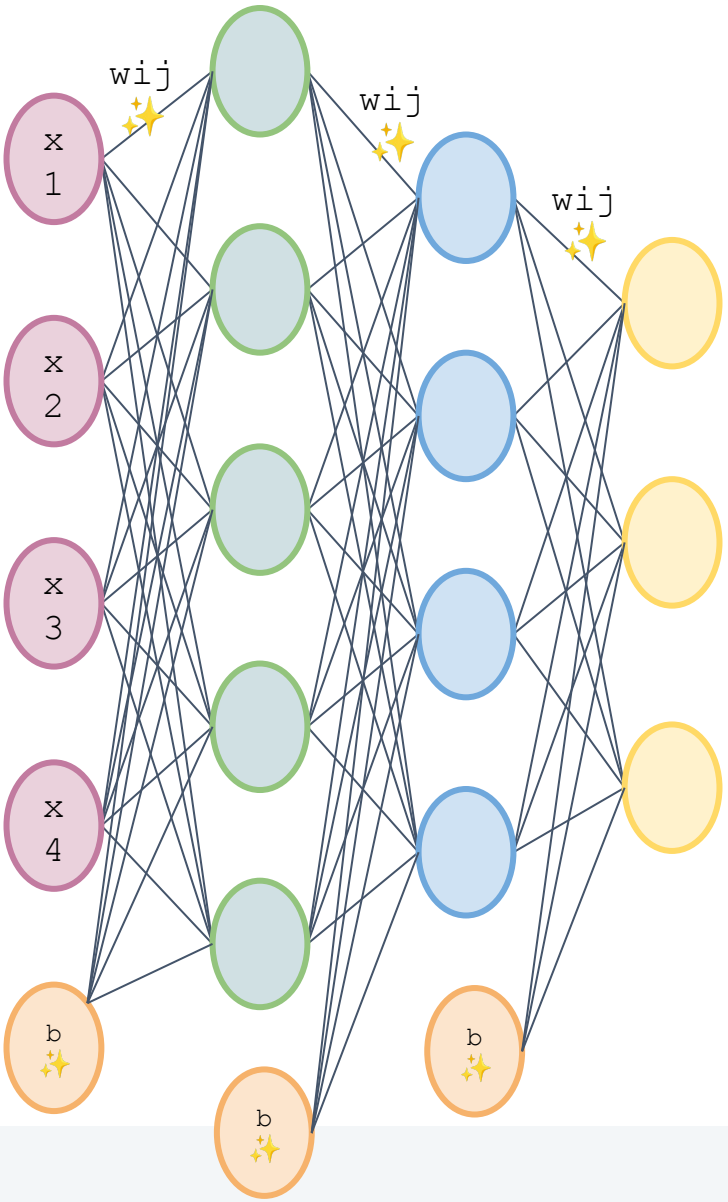
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 8:

Evaluation

Step 9:

Deploy



Common Losses in NN

Regression tasks

- MSE
  MSE calculates the average squared difference between the predicted values and actual values.

$$\text{MSE} = \frac{1}{n}\sum(y_{\text{true}} - y_{\text{pred}})^2$$

Classification tasks

- Cross-Entropy Loss
  Measures the difference between the true label distribution and the predicted probability distribution

$$L = -\sum y_{\text{true}} \log(y_{\text{pred}})$$

An artificial neural network

Code:

- https://colab.research.google.com/drive/15teX61eEkIIZjevCcEhdUhv96cxw-pox?authuser=1
- (Tensorflow) https://colab.research.google.com/drive/1kK2Q0Z55A5qA6xQJ87OkdAVqIlR1Dktn?authuser=1

Additional resources

- Neural Networks by StatQuest
- PyTorch documentation

Hyperparameters

We talked about hyperparameters in previous modules

Hyperparameters

Parameters:

Hyperparameters:

Hyperparameters

Parameters:

- Configuration variable that is internal to the model and is used to make predictions on new data
- Model parameters are learned from training data
- They are often not set manually by you
- Examples:
  - The coefficients of linear regression models
  - Support vectors in SVM
  - Split points in a decision tree

Hyperparameters:

Hyperparameters

Parameters:

- Configuration variable that is internal to the model and is used to make predictions on new data
- Model parameters are learned from training data
- They are often not set manually by you
- Examples:
  - The coefficients of linear regression models
  - Support vectors in SVM
  - Split points in a decision tree

Hyperparameters:

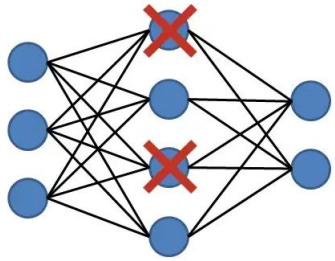
- Configuration variables that are set before the training process begins
- They control the behavior of the learning algorithm and the model itself (ie determine parameters)
- Cannot be learned directly from the training data
- They are often specified by you
- Examples:
  - Learning rate in gradient descent
  - Maximum depth of the tree
  - Number of trees in a random forest

Hyperparameters in Neural Networks

Hyperparameters are parameters whose values are set before the learning process begins, contrasting with the weights of the network, which are learned during training

- **Learning Rate**:
  - One of the most critical hyperparameters
  - Determines how much the weights should be updated concerning the loss gradient
  - Too high can cause the model to converge quickly but inaccurately (possibly overshooting the global minimum)
  - Too low might result in a long training time or the model getting stuck in a local minimum.

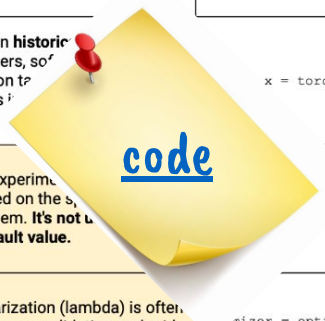
- Regularization Techniques
  - **Dropout rate**



    - Prevents overfitting in neural networks
    - During training, some number of layer outputs a ignored or "dropped out."
    - By “dropping out” some units (i.e., setting them to zero), the neural network becomes less sensitive to the specific weights of neurons
    - This in turn results in a network that is capable of better generalization and is less likely to overfit the training data

  - **L1/L2 regularization**
    - Adds an additional term to the loss function, which penalizes certain parameter values to prevent the coefficients from fitting so perfectly to overfit the training data

Category	Hyperparameter	What is it	Why is it important	How to optimize	PyTorch code
Optimization	**Learning rate**	The size of the steps taken during optimization	Controls how fast or slow a model learns. Too high might miss minima, too low might take too long.	Often found through trial and error, grid search, or algorithms like learning rate annealing/scheduling, or adaptive learning methods (like in Adam optimizer). **Not typically left at default, as it's critical for model performance.**	`optimizer = optim.Adam(model.parameters(), lr=0.01)`
	Momentum	The amount of "velocity" carried from previous gradients during optimization	Helps accelerate gradients vectors in the right directions, thus leading to faster converging	**Often kept at a default** (like 0.9) for optimizers like SGD with momentum.	`optimizer = optim.Adam(model.parameters(), lr=0.01, momentum=0.9)`
	Optimizer	The method used to update the model's weights based on the data's gradients	Different optimizers may converge faster and more reliably depending on the task.	Choice is based on the specific task, the size of the data, or **historical precedence**. Adam is very popular due to its performance across a wide range of tasks.	`optimizer = optim.Adam(model.parameters(), lr=0.01, momentum=0.9)`
Training configuration	**Number of epochs**	Number of times the entire training dataset is passed forward and backward through the neural network	Determines how long the model trains, which affects the learning detail and potential for overfitting.	Set based on when the model stops improving on a validation set, **often using techniques like early stopping rather than fixing a number upfront**.	`for epoch in range(num_epochs): # training loop...`
	Batch size	The number of training samples used in one iteration to update the model's weights	Larger batches provide more accurate gradient estimates but require more memory and often lead to poorer generalization, while smaller batches use less memory and can generalize better but may lead to less stable convergence.	It's often chosen based on memory limitations and **empirically** set by trying several different values.	`train_loader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)`
Network architecture	**Number of hidden layers and units**	The layers within the neural network that perform computations and hold weights	They capture features and complexities within the data. Too many can lead to overfitting.	Generally found through experimentation, **historical precedence**, or sometimes heuristics.	`self.fc1 = nn.Linear(4, 12)`
	Activation functions	Non-linear functions that decide if a neuron should be activated	They introduce non-linearity, helping NNs learn complex patterns that linear models cannot.	Often chosen based on **historic** ReLU for hidden layers, so[f] layers in classification ta[sks] **intensively** unless [you're] specializ[ing]	`x = torch.relu(self.fc1(x))`
Regularization	**Dropout rate**	The probability of temporarily removing a neuron from the network during training	Prevents over-reliance on any one neuron, promoting robustness and reducing overfitting.	Typically found via experim[entation] vary significantly based on the [model] architecture and problem. **It's not [recommended] a default value.**	`= nn.Dropout(p=0.5)`
	L1/L2 regularization	Additional term in the loss function that penalizes certain parameter values to prevent the coefficients from overfitting	Adds penalties on weight size, encouraging simpler models that are less likely to overfit.	The strength of regularization (lambda) is often determined through cross-validation and grid search. It's problem-specific and thus **not left at default.**	`[optim]izer = optim.Adam(model.parameters(), [lr]=0.01, momentum=0.9, weight_decay=1e-5) #L2`



https://docs.google.com/spreadsheets/d/1DwM_Wm5B9VxXzGd-AlDhrV3SJ2bi0CjX/edit?usp=sharing&ouid=104266955540125590646&rtpof=true&sd=true

Types of NN

- **Fully Connected Neural Networks (FCNNs)**

  Also known as Multilayer Perceptrons (MLPs), these are the simplest form of neural networks, where each neuron in one layer is connected to all neurons in the next layer.
- **Convolutional Neural Networks (CNN)**

  Highly effective in processing data that has a grid-like topology, such as images. CNNs have revolutionized the field of computer vision, providing substantial improvements in image recognition, image classification, object detection, and many other areas
- **Recurrent Neural Networks (RNNs)**

  Designed to handle sequential data such as time series, speech, or text. They can maintain state information by feeding the output of a neuron back into the network as an input for the next step, essentially 'remembering' previous information

  Variants include: Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), which are designed to capture long-term dependencies and solve the vanishing gradient problem common in basic RNNs.
- **Autoencoders (AE)**

  Used for unsupervised learning tasks, primarily dimensionality reduction and feature learning.
- **Generative Adversarial Networks (GANs)**

  Consist of two networks, a generator and a discriminator, that are trained simultaneously in a zero-sum game framework.

  The generator learns to create data that's similar to the training data, and the discriminator learns to differentiate between the generated data and real data. They are widely used for generating realistic images, style transfer, and more.
- **Transformer Networks**

  Introduced in the paper "Attention is All You Need", transformers are designed to handle sequential data without the need for recurrence. They rely on a mechanism called 'attention' to weigh the influence of different parts of the input data. Foundation of models like BERT, GPT, etc.
- and more…

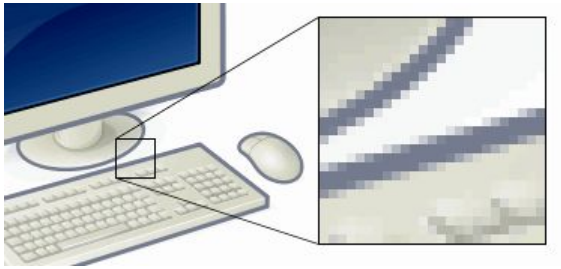
CNN

What is an image?
According to computers → Computer Vision

What We See

What Computers See

What is an image?
According to computers → Computer Vision



- **Pixel**: (Picture element) The basic unit of a digital image
- Each pixel represents a tiny area of the picture and is assigned a color value
- The most direct measure of an image's resolution is its width and height in pixels

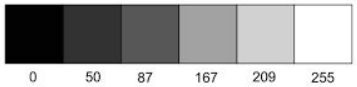
What is an image?
Grayscale images vs color images

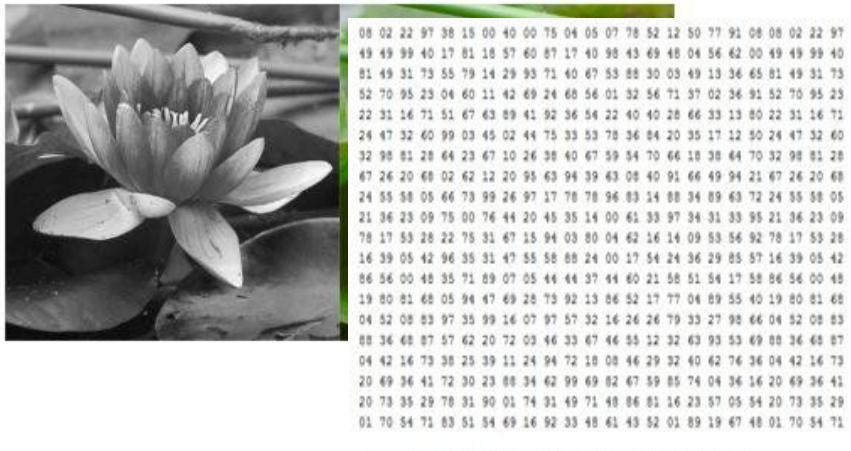
(a) (b)

What is an image?
Grayscale images vs color images

- Each pixel in a black and white image can be represented by a single value which indicates the intensity of the gray at that particular point.

What is an image?
Grayscale images vs color images



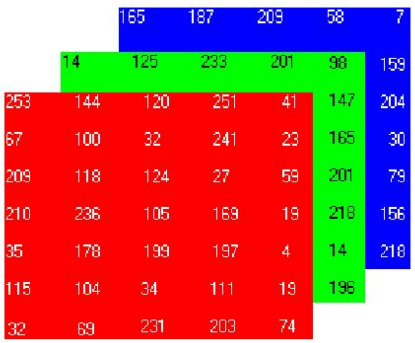


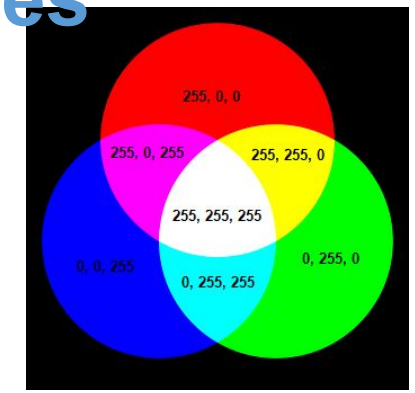
- Each pixel in a black and white image can be represented by a single value which indicates the intensity of the gray at that particular point.

What is an image?
Grayscale images vs color images

- RGB color model: Each color is made up of three values (red, green, blue)
- Each of the three color values can range from 0 to 255

What is an image?
Grayscale images vs color images



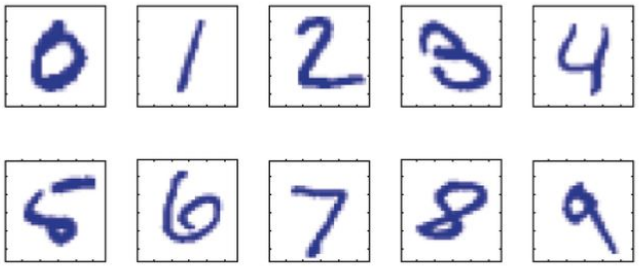


- RGB color model: Each color is made up of three values (red, green, blue)
- Each of the three color values can range from 0 to 255

What is Machine Learning?

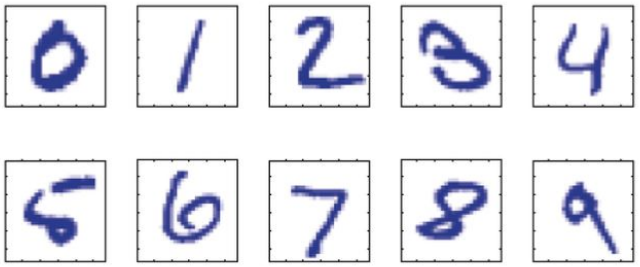
What is Machine Learning?

Question: How can one recognize handwritten digits?



What is Machine Learning?

Question: How can one recognize handwritten digits?

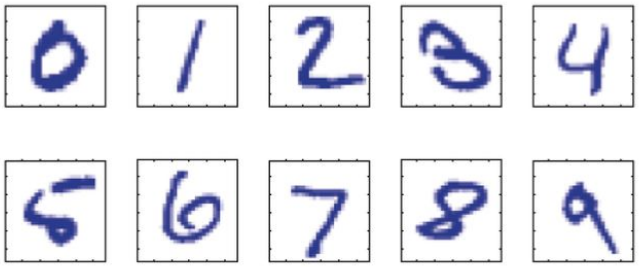


Potential answer: Design your own rules?

- A series of aligned pixels ⇒ ‘1’
- A circle of pixels ⇒ ‘0’
- Etc…

What is Machine Learning?

Question: How can one recognize handwritten digits?



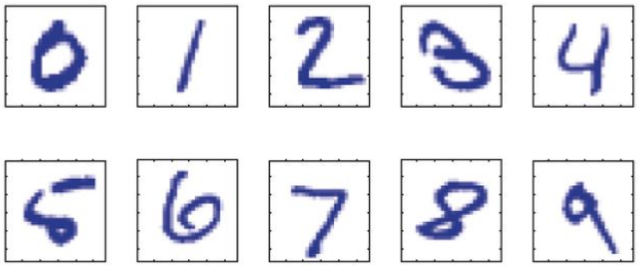
Potential answer: Design your own rules?

- A series of aligned pixels ⇒ ‘1’
- A circle of pixels ⇒ ‘0’
- Etc…

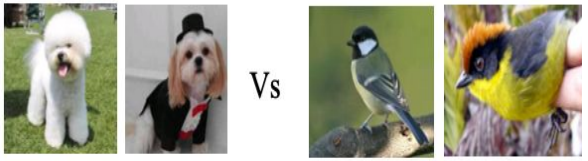
Bad generalization

What is Machine Learning?

Question: How can one recognize handwritten digits?



Often difficult



Dogs Birds

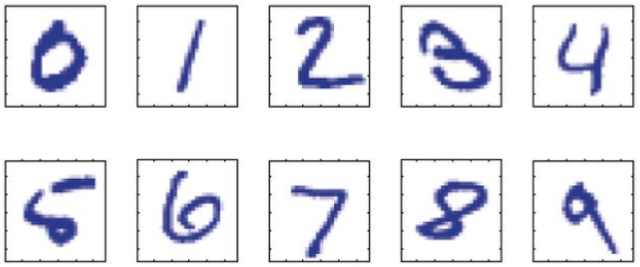
Potential answer: Design your own rules?

- A series of aligned pixels ⇒ ‘1’
- A circle of pixels ⇒ ‘0’
- Etc…

Bad generalization

What is Machine Learning?

Question: How can one recognize handwritten digits?

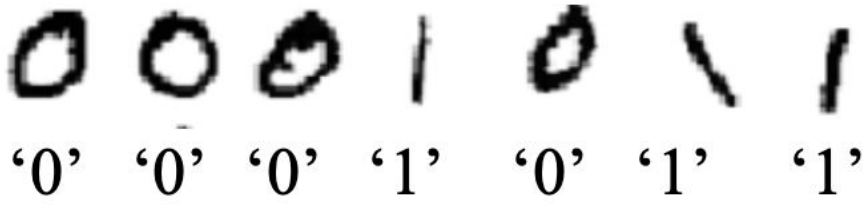


Answer: Let the computer **learn** the rules

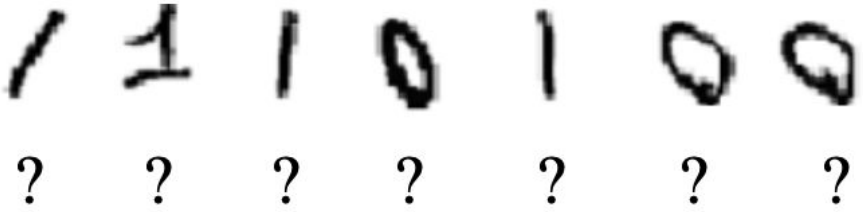
Main goal of machine learning!

What is Machine Learning?

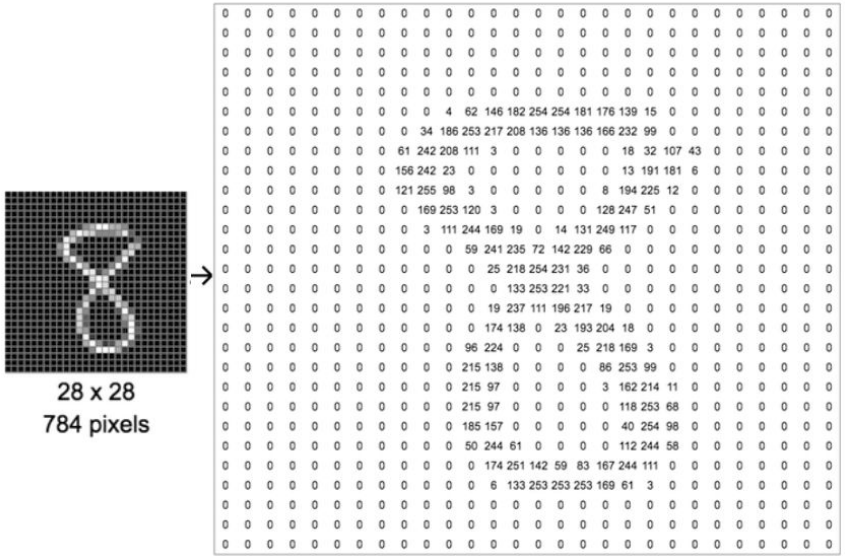
Provide the algorithm with annotated training data…



… and the algorithm returns a function capable of **generalizing** on new data



Example of a classification dataset
MNIST



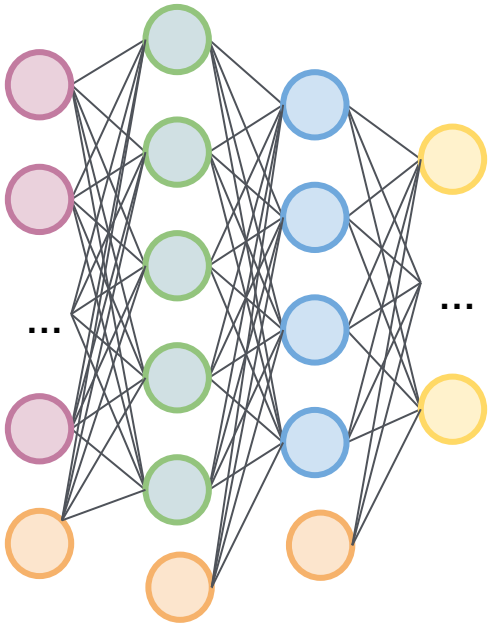
- Large database of handwritten digits, commonly used for training
- 10 classes
- 70,000 images
- Images are grayscale

  28x28 = 784 dimensions

Example of a classification dataset

MNIST

If we were going to do this with a Fully Connected Neural Network (FCNN), also known as Multilayer Perceptron (MLPs)...



- Input layer

  ____ neurons

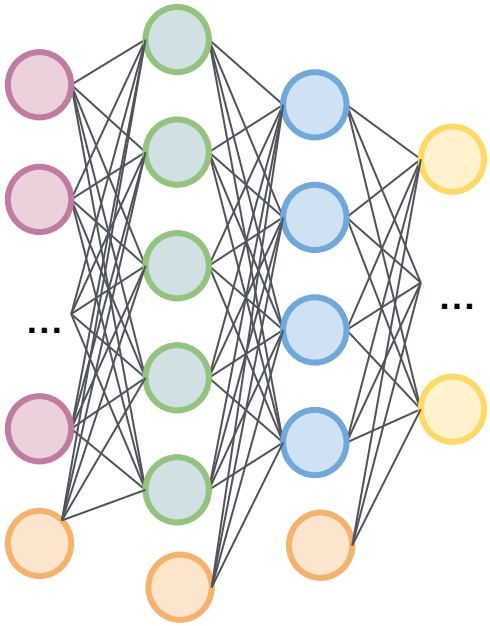
- Output layer

  ____ neurons

Example of a classification dataset

MNIST

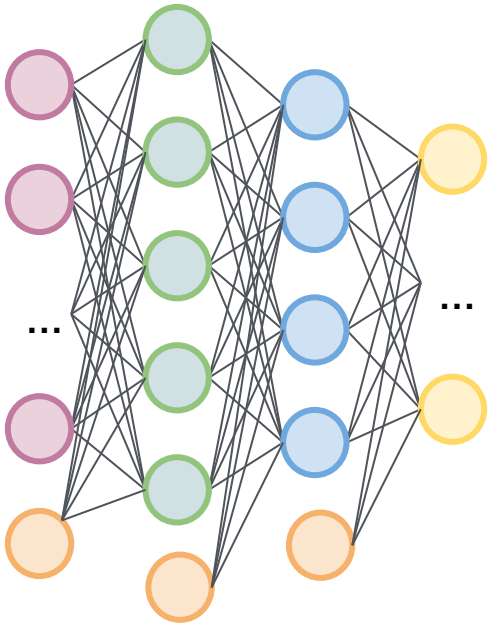
If we were going to do this with a Fully Connected Neural Network (FCNN), also known as Multilayer Perceptron (MLPs)...



- Input layer
  784 neurons
- Output layer
  10 neurons

Example of a classification dataset
MNIST

If we were going to do this with a Fully Connected Neural Network (FCNN), also known as Multilayer Perceptron (MLPs)...



- Input layer
  784 neurons
- Output layer
  10 neurons

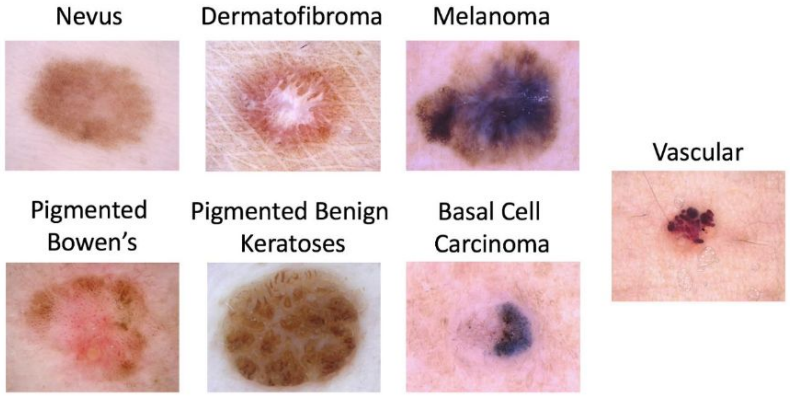
Suppose 1 hidden layer with 256 neurons

- Weights: 784 (input neurons) x 256 (hidden neurons) = 200,704
- Biases: 256 (one per hidden neuron)
- Weights: 256 (hidden neurons) x 10 (output neurons) = 2,560
- Biases: 10 (one per output neuron)
- **Grand total: 203,530 parameters!!!**

Example of a classification dataset

ISIC melanoma classification challenge dataset

https://challenge.isic-archive.com/landing/2018/



- 7 classes
- 11,527 images
- Images are in color

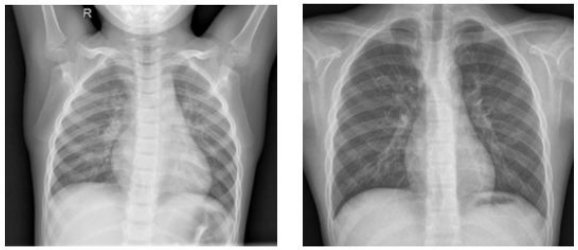
  628x417x3 = 785,628 dimensions

Example of a classification dataset

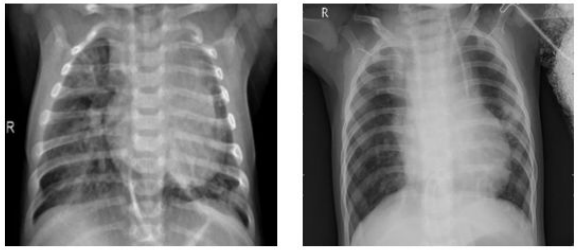
Chess X-Ray Pneumonia

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

Healthy

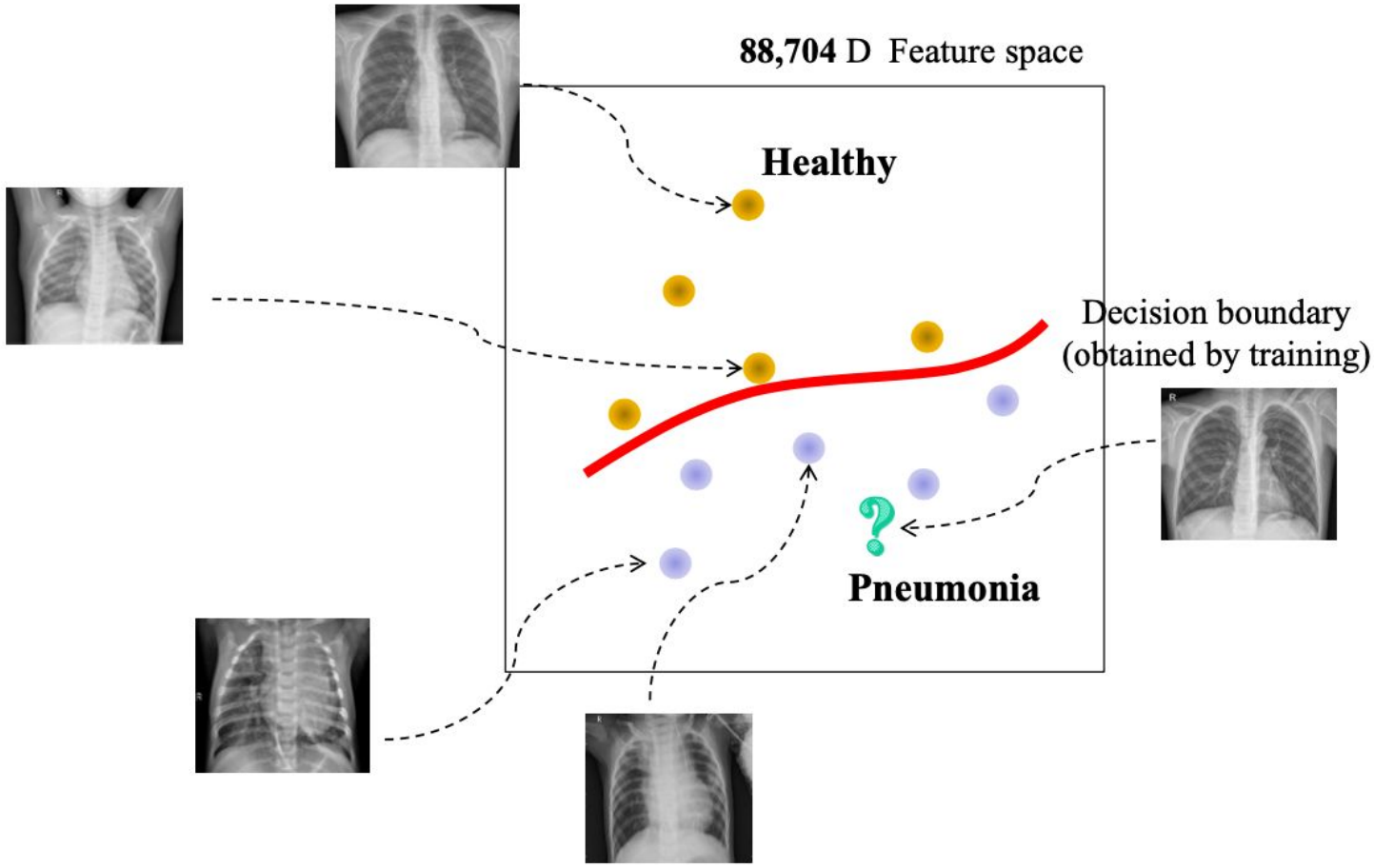


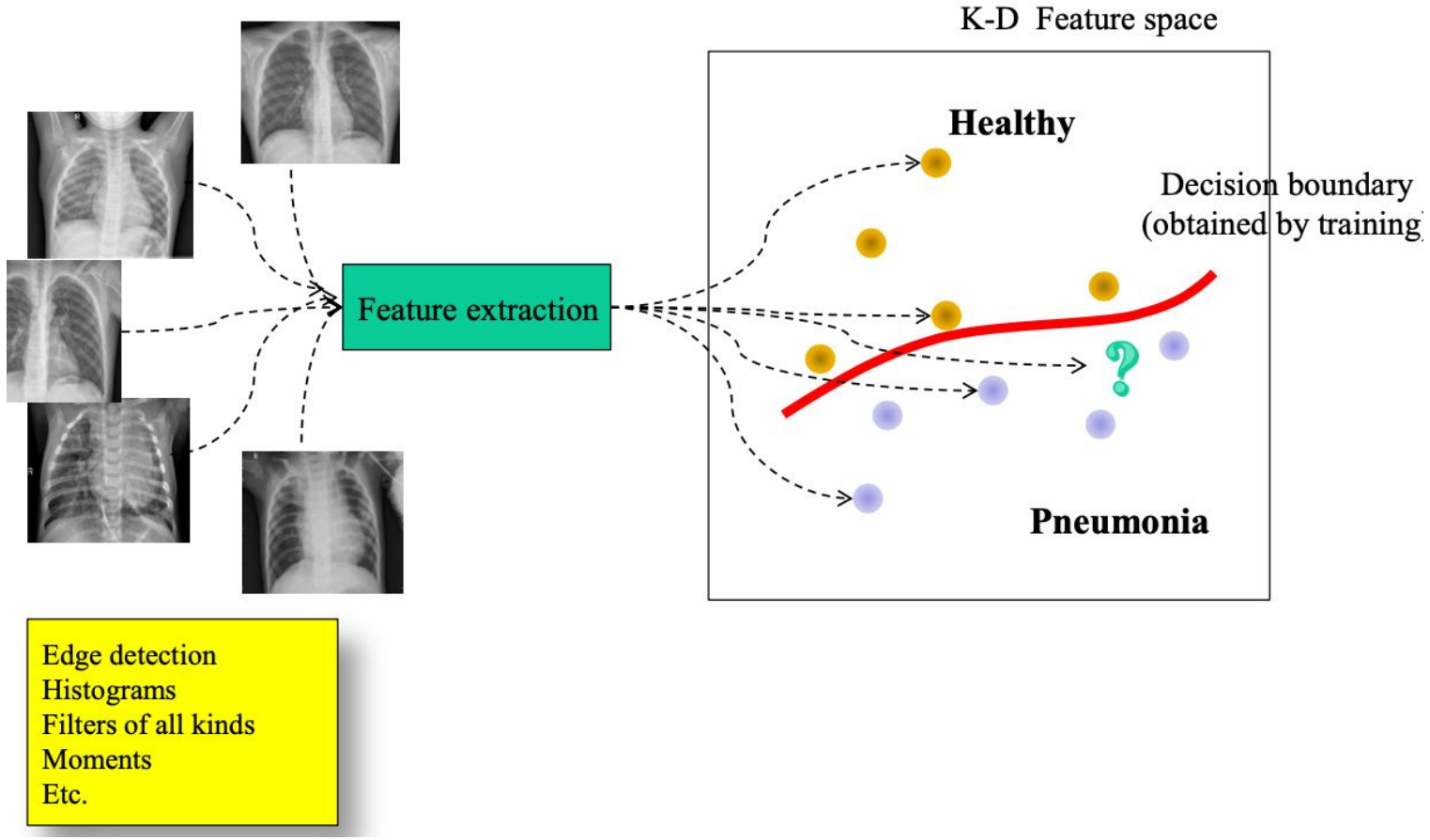
Pneumonia

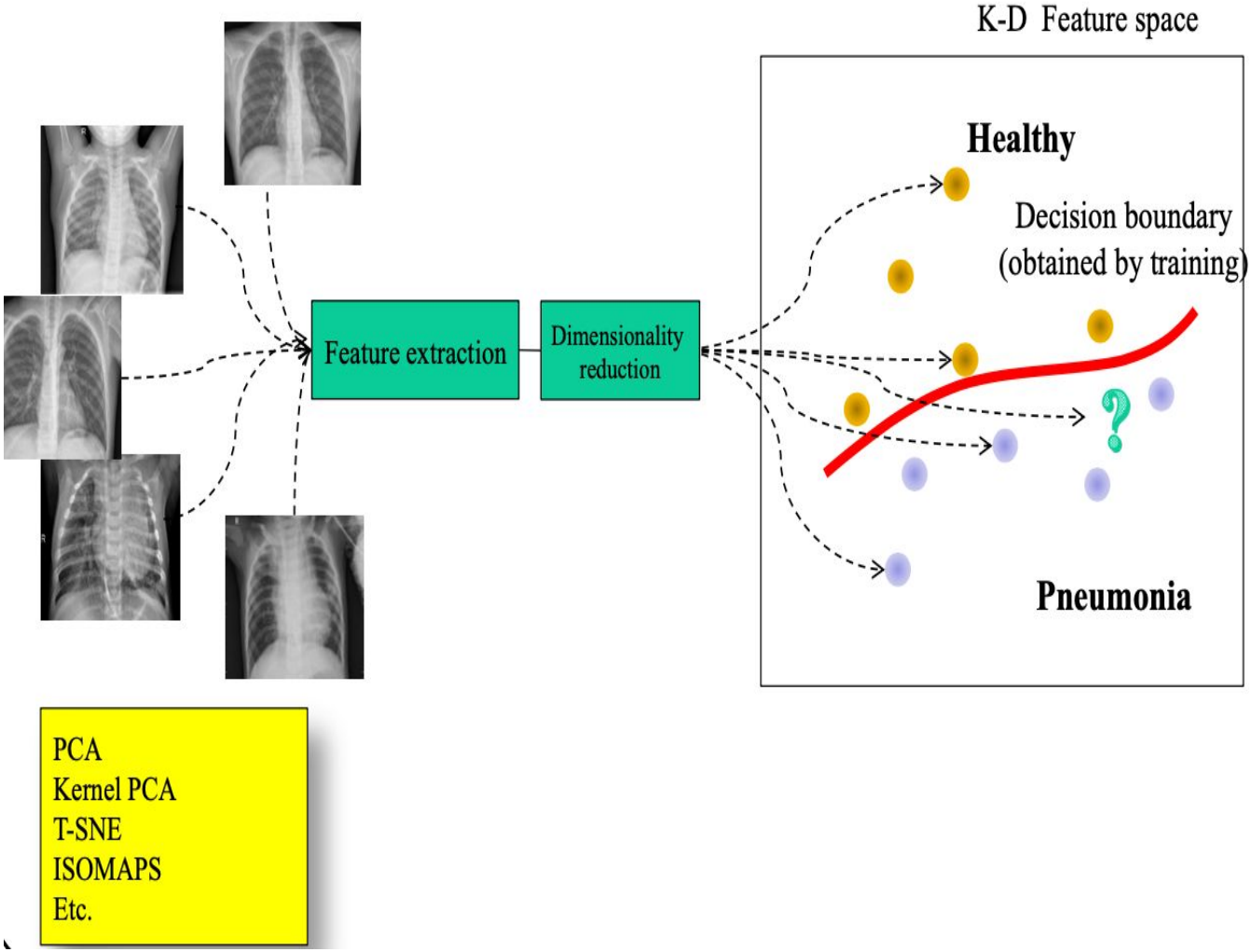


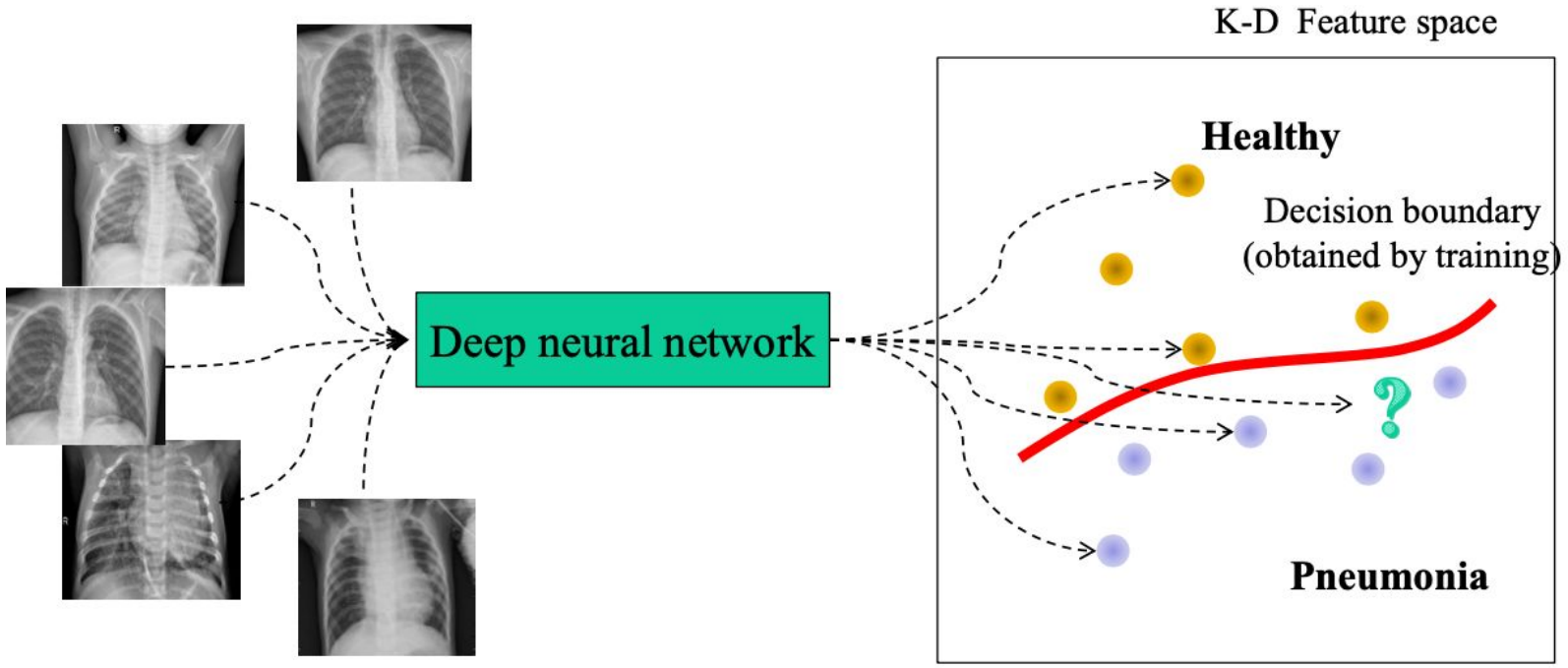
- 2 classes
- 5,840 images
- Images are grayscale, after rescaling

  336 x 264 = 88,704 dimensions


88,704 D Feature space
Healthy
Decision boundary
(obtained by training)
Pneumonia


K-D Feature space
Healthy
Decision boundary
(obtained by training
Feature extraction
Pneumonia
Edge detection
Histograms
Filters of all kinds
Moments
Etc.


K-D Feature space
Healthy
Decision boundary
(obtained by training)
Pneumonia
Feature extraction
Dimensionality reduction
PCA
Kernel PCA
T-SNE
ISOMAPS
Etc.


K-D Feature space
Healthy
Decision boundary
(obtained by training)
Deep neural network
Pneumonia

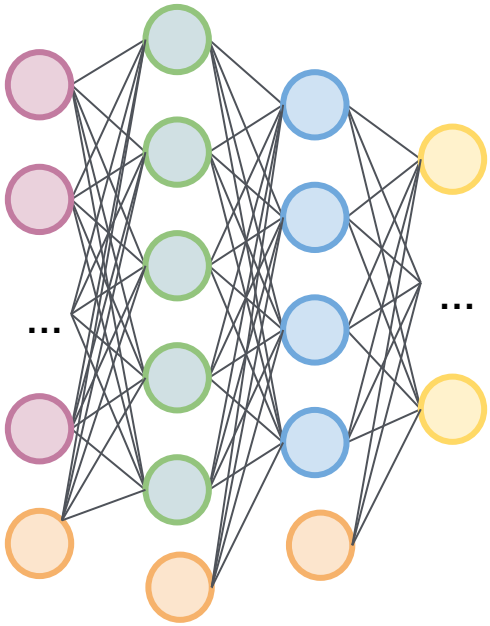
Very large feature spaces (like **88,704 dim) are problematic**.

- Computing/time expensive
- Increased # of neurons/layers → overfitting

Very large feature spaces (like **88,704 dim) are problematic**.

Example of a classification dataset
MNIST

If we were going to do this with a Fully Connected Neural Network (FCNN), also known as Multilayer Perceptron (MLPs)...



- Input layer
  784 neurons
- Output layer
  10 neurons

Suppose 1 hidden layer with 256 neurons

- Weights: 784 (input neurons) x 256 (hidden neurons) = 200,704
- Biases: 256 (one per hidden neuron)
- Weights: 256 (hidden neurons) x 10 (output neurons) = 2,560
- Biases: 10 (one per output neuron)
- **Grand total: 203,530 parameters!!!**

How to classify an image?

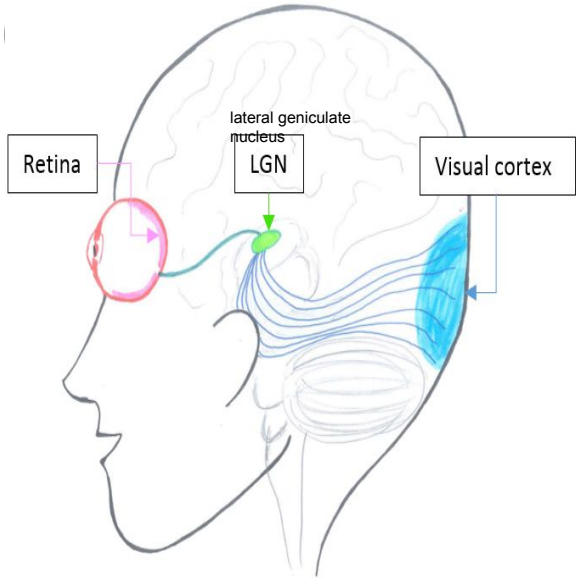
How to classify an image?

Convolutional Neural Networks!

Convolutional Neural Network | CNN | CovNets

- Type of Neural Network
- Typically used to analyze visual imagery
- Also used for NLP, Audio/Speech processing, and more…

Convolutional Neural Networks | CNN |



- CNNs are inspired by the biological processes of the human visual cortex
- **Visual Cortex Mimicry**: CNNs are inspired by the organization of the animal visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field
- **Hierarchical Processing**: Similar to how the visual cortex processes visual information in a hierarchical manner, CNNs also process data through multiple layers where each layer captures a different level of abstraction
- **Pattern Recognition**: Just as the human visual system is able to recognize patterns (like shapes and edges), CNNs are designed to automatically and hierarchically learn spatial hierarchies of features
- **Shared Weights and Repetition:** Biological neurons often reuse the same functionality across different spatial locations. Similarly, CNNs use shared weights in convolutional filters across the entire input, which also leads to translation invariance — detecting features irrespective of their position in the visual field

Functionality of a CNN

- **Feature Learning:** The initial layers of a CNN automatically learn to detect features such as edges and simple textures. As the data passes through the network, subsequent layers learn to detect more complex features.

- **Spatial Hierarchy**: CNNs exploit the spatial hierarchical pattern in data by constructing a complex pattern using smaller and simpler patterns. Thus, the network learns from the general to the specific.

- **Translation Invariance**: Once a feature is learned, the CNN can recognize that feature anywhere in the image, making CNNs robust to the location of the feature in the input.

Characteristic components of a CNN

- **Convolutional Layers**: The core building blocks of a CNN. These layers perform the convolution operation, which involves sliding a filter or kernel over the input image and computing the dot product of the filter elements and the input pixels in a local region (receptive field). The result is a feature map that emphasizes certain features from the input.

- **Pooling Layers**: Pooling (also known as subsampling or downsampling) reduces the spatial size of the feature map, which decreases the number of parameters and computation in the network, thereby also controlling overfitting. The most common pooling operation is max pooling, which takes the maximum value in a local patch of neurons.

LeNet-5

- Pioneering type of CNN developed by Yann LeCun et al
- Primarily designed for classification tasks, specifically for handwritten digit recognition on the MNIST dataset
- Since then, many other types of CNN architectures have been developed, each designed to tackle specific challenges in computer vision or to improve upon the performance of previous models
- Modern CNNs are more complex and deeper, but understanding LeNet-5 provides an essential historical and architectural perspective
- Architecture: https://medium.com/codex/lenet-5-complete-architecture-84c6d08215f9

Other notable CNN architectures

- **AlexNet:** Developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, AlexNet won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 by a substantial margin. It was deeper and wider than LeNet and introduced ReLU activations.

- **VGGNet:** Developed by Visual Graphics Group from Oxford (hence VGG), this network is known for its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth.

- **GoogLeNet (Inception):** This network introduced a novel inception module, which allows the network to choose from multiple filter sizes. GoogLeNet also significantly reduced the number of parameters compared to AlexNet and VGGNet.

- **ResNet (Residual Network):** Developed by Kaiming He et al., ResNet introduced residual connections that allow gradients to flow through the network more effectively, enabling the training of very deep networks (networks with more than 100 layers).

- **U-Net:** Originally designed for biomedical image segmentation, U-Net has an architecture that includes a contracting path to capture context and a symmetric expanding path that enables precise localization.

- **DenseNet (Densely Connected Convolutional Networks):** This architecture connects each layer to every other layer in a feed-forward fashion. DenseNets are known for their efficiency in terms of parameters and computation.

https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

- and more…

ImageNet

https://www.image-net.org/

- ImageNet is a large-scale, diverse database of images that has been one of the most influential resources in the field of computer vision and deep learning
- ImageNet, with its millions of labeled images, provides a standard dataset to train and test new algorithms, especially deep learning models
- Pre-trained models on ImageNet have been widely used for transfer learning. Transfer learning involves taking a model trained on a large dataset (like ImageNet) and fine-tuning it on a smaller, domain-specific dataset

Additional resources

- ML4A
- CNN by StatQuest
- Feature visualization
- Grad-CAM

Characteristic components of a CNN

- **Convolutional Layers**: The core building blocks of a CNN. These layers perform the convolution operation, which involves sliding a filter or kernel over the input image and computing the dot product of the filter elements and the input pixels in a local region (receptive field). The result is a feature map that emphasizes certain features from the input.

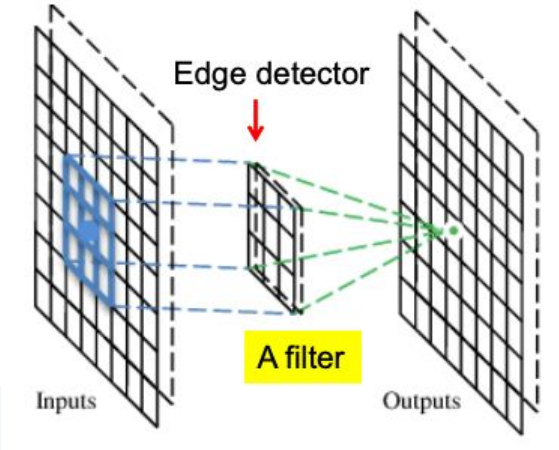
- **Pooling Layers**: Pooling (also known as subsampling or downsampling) reduces the spatial size of the feature map, which decreases the number of parameters and computation in the network, thereby also controlling overfitting. The most common pooling operation is max pooling, which takes the maximum value in a local patch of neurons.

Characteristic components of a CNN

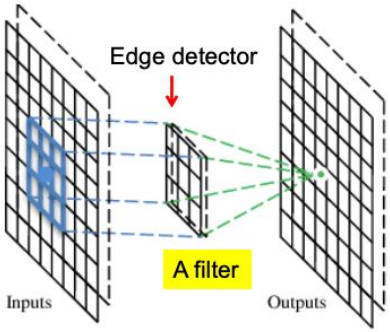
- Convolutional Layer
  - Filter/Kernel
  - Convolution Operation
  - Stride
  - Padding
  - Feature Map
  - Parameter sharing
- Pooling Layer

A convolutional layer

- **Key idea:** Replace matrix multiplication in neural networks with convolution
- Core building block of CNNs
- In CNNs, convolution involves a filter or kernel (a small matrix of weights) that passes over the input data (like an image) in a sliding window fashion
- As the filter slides over the input data, it performs element-wise multiplication with the part of the input it is currently on, and the results of these multiplications are summed up to form a single output pixel in the feature map



A convolutional layer



- **Filter/Kernel:** small matrix used to apply effects such as blurring, sharpening, and edge detection to images
- These are the parameters to be learned

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

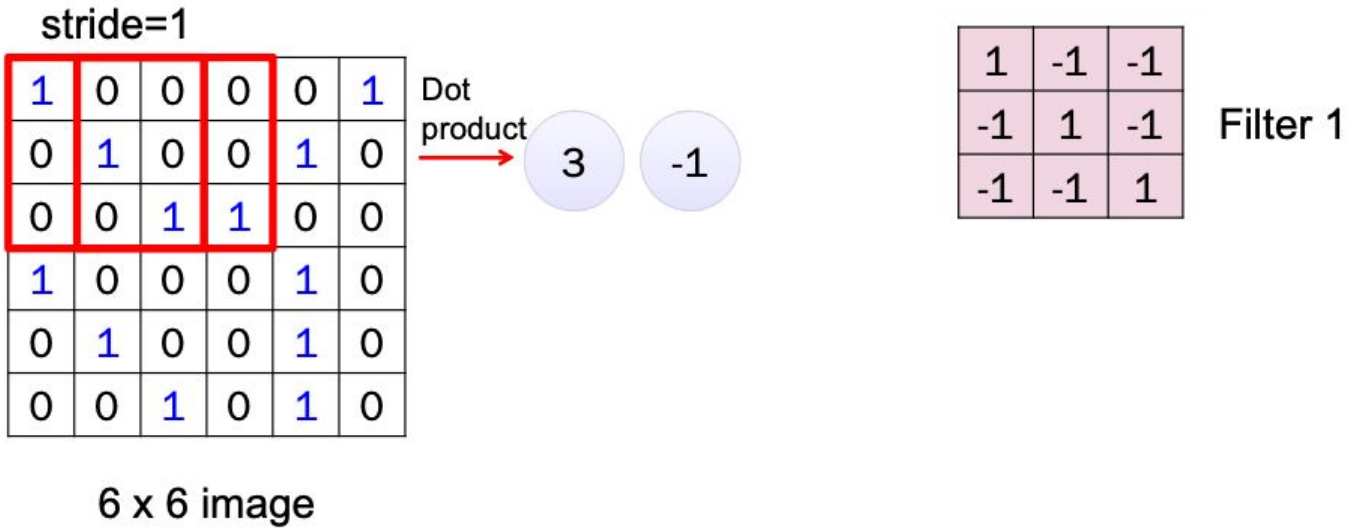
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

A convolutional layer

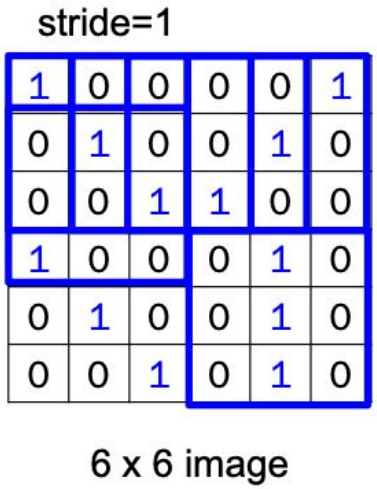


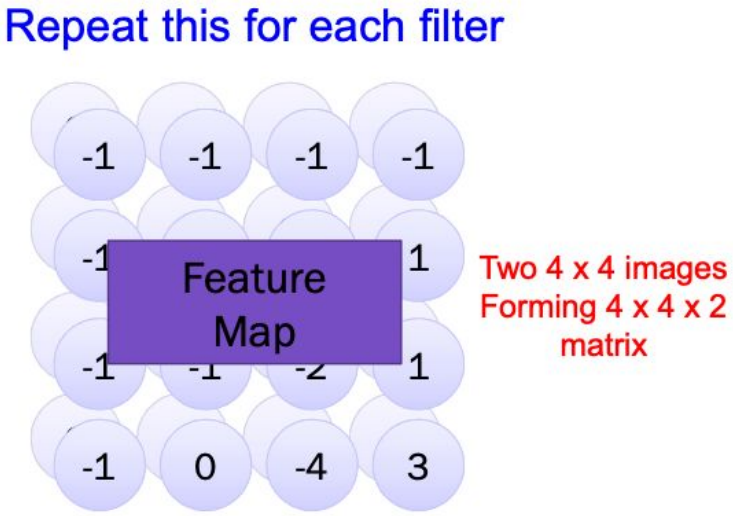
- **Strides:** Number of pixels by which the filter matrix moves across the input image

A convolutional layer

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

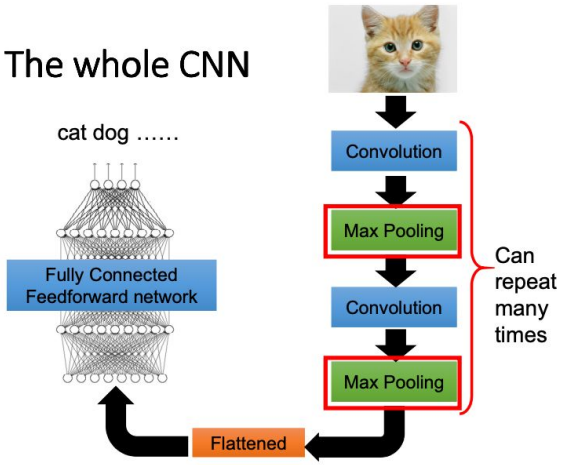




- **Feature map:** The output of the convolution operation is a feature map that represents certain features of the input image, like edges, corners, or textures.

Traditional CNNs

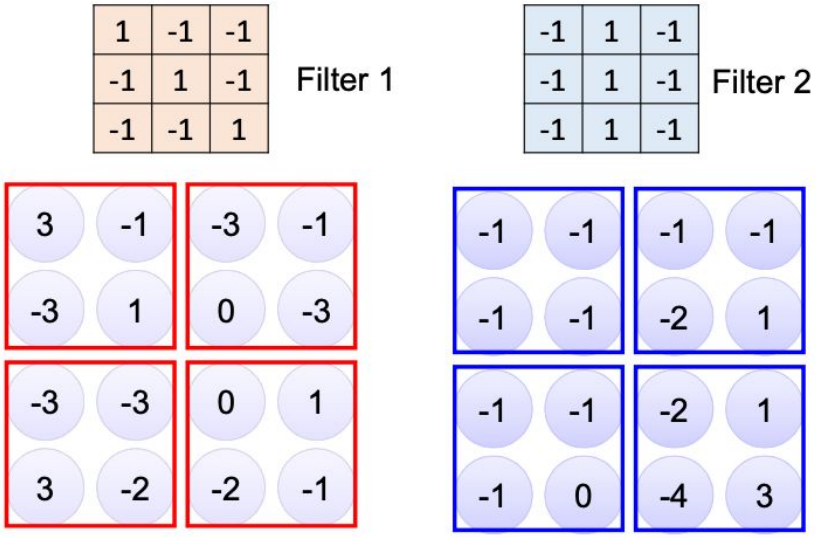
- In traditional CNN architectures, after several convolutional and pooling layers, the high-dimensional output (the feature maps) is flattened into a one-dimensional vector to be used as input for the fully connected layers that follow.
- This flattening step is necessary because fully connected layers expect a fixed number of neurons as input, and the flattening process converts the 2D feature maps into a 1D feature vector.

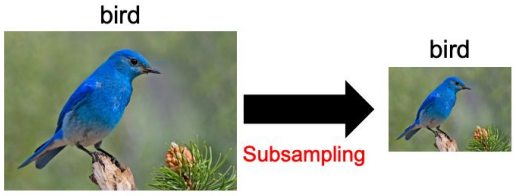


A pooling layer

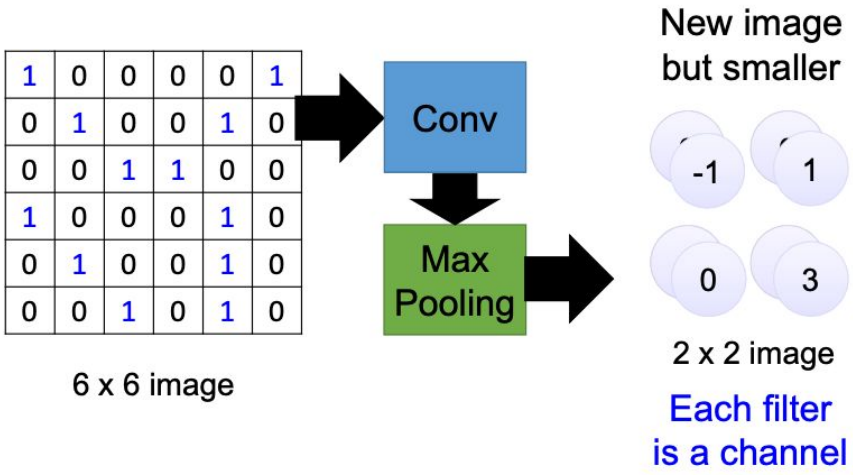
They follow the convolutional layers and serve several purposes in the architecture of a CNN:

- Dimensionality reduction
- Reduced overfitting
- Pooling layers do not have weights that are learned during training. They are fixed operations that only depend on the chosen pooling strategy (max, average, etc.).

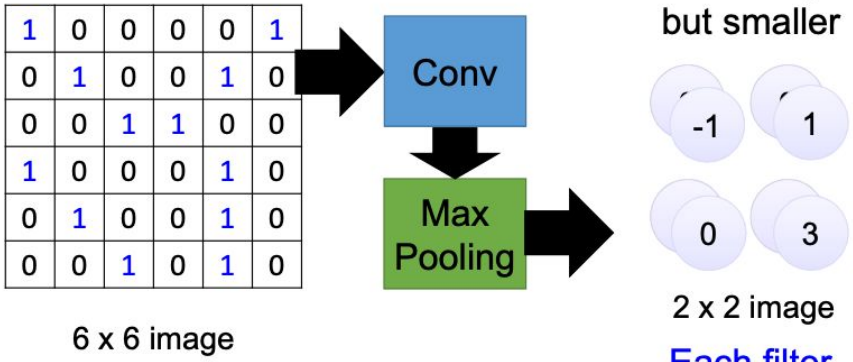


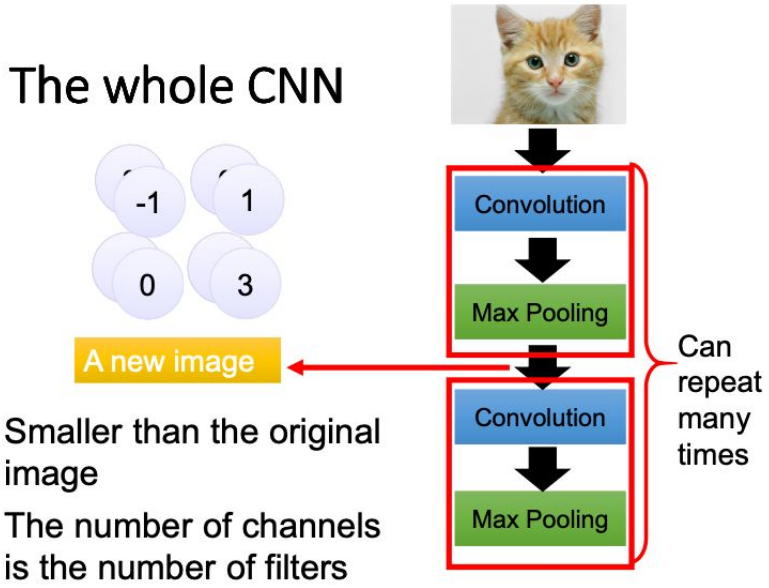


A pooling layer



A pooling layer





The whole CNN

Convolution

Max Pooling

A new image

Convolution

Max Pooling

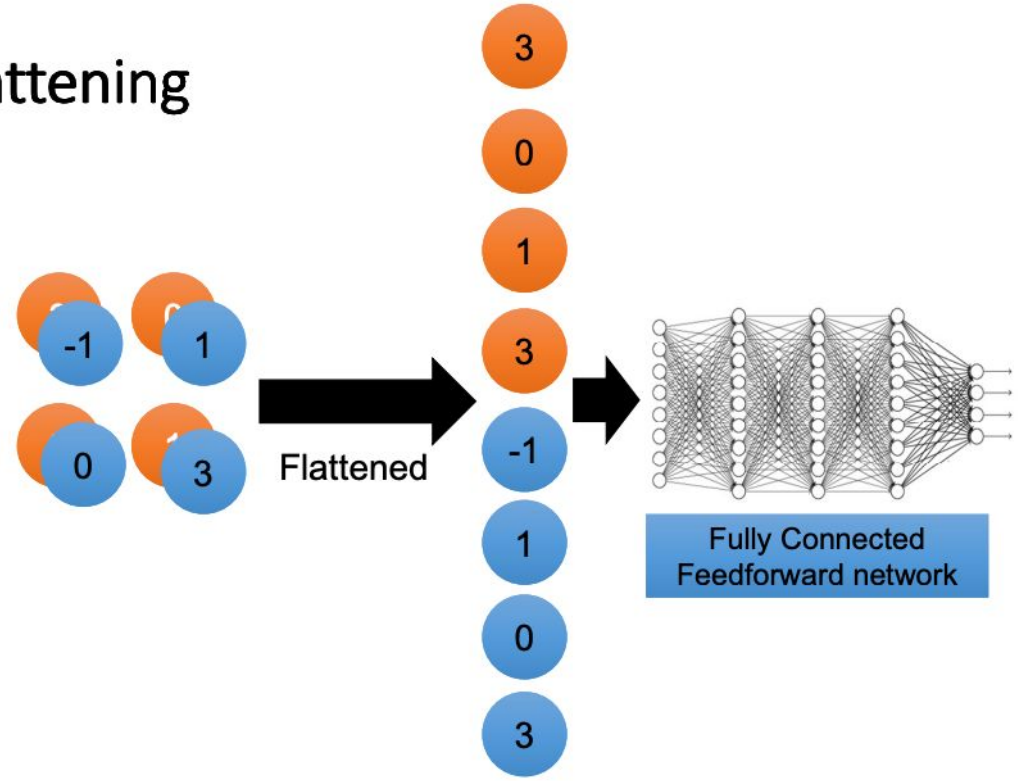
A new image

cat dog ……

Fully Connected Feedforward network

Flattened

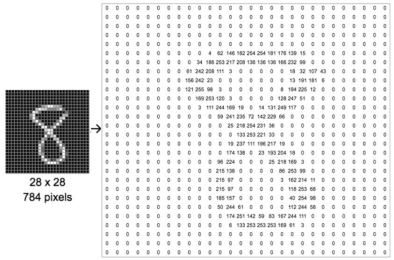
Flattening


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

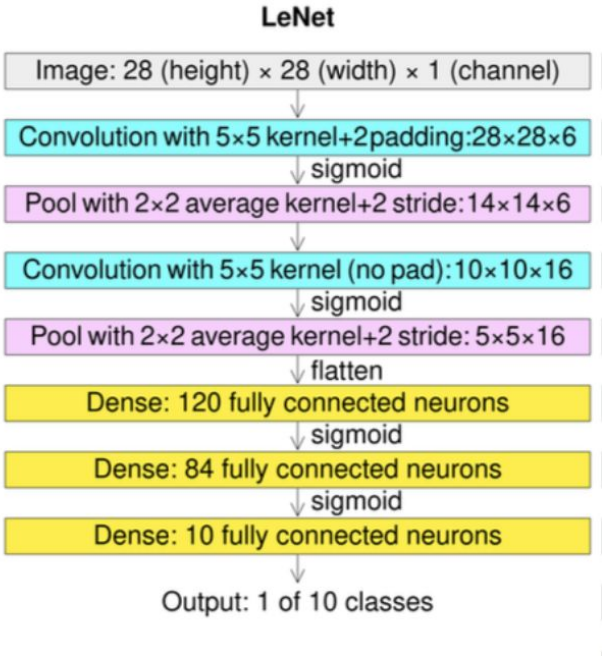
LeNet-5

- Pioneering type of CNN developed by Yann LeCun et al
- Primarily designed for classification tasks, specifically for handwritten digit recognition on the MNIST dataset
- Since then, many other types of CNN architectures have been developed, each designed to tackle specific challenges in computer vision or to improve upon the performance of previous models
- Modern CNNs are more complex and deeper, but understanding LeNet-5 provides an essential historical and architectural perspective
- Architecture: https://medium.com/codex/lenet-5-complete-architecture-84c6d08215f9

LeNet-5



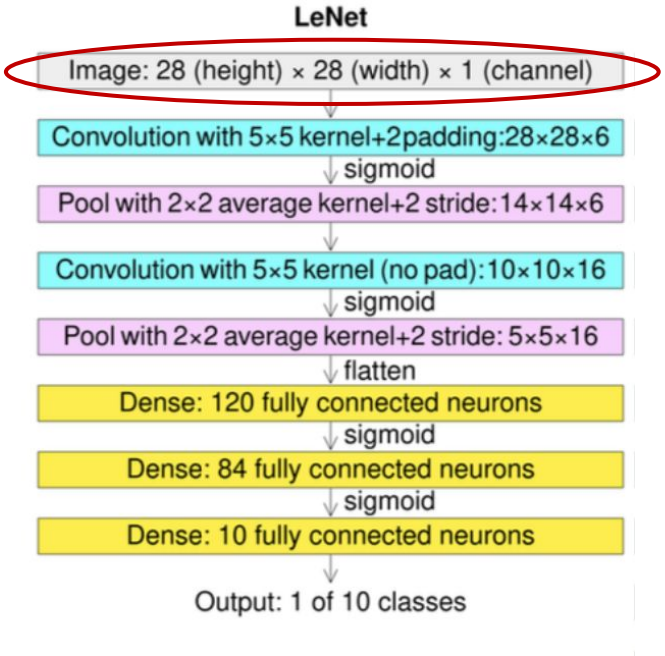
MNIST dataset

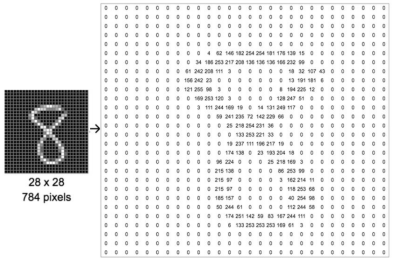


LeNet-5

Input Image:

- The input is a single-channel (grayscale) image with a dimension of 28x28 pixels.





MNIST dataset

```python
# Convolutional layer 1: 1 input channel (for grayscale images), 6 output channels, 5x5 kernel
self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
```

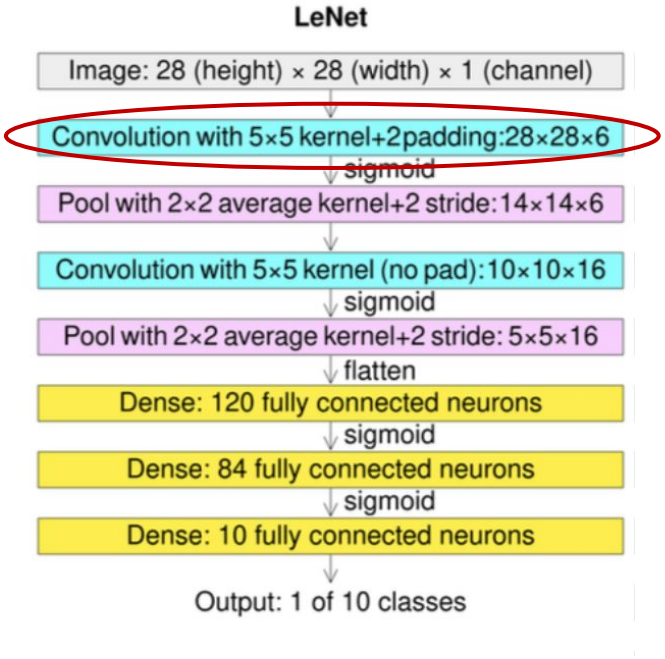
LeNet-5

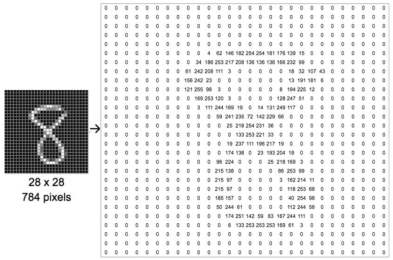
First Convolutional Layer:

- This layer applies a 5x5 kernel (or filter) with 2 pixels of padding to the input image.
- Padding helps maintain the size of the output feature map. In this case, it ensures the feature map remains at 28x28 pixels after the convolution.
- The depth of the output is 6, indicating that there are 6 filters being applied, each generating its own feature map. Thus, the output from this layer is a 28x28x6 volume.

Activation Function:

- After the convolution operation, a sigmoid activation function is applied to each pixel in the feature map, introducing non-linearity to the model.





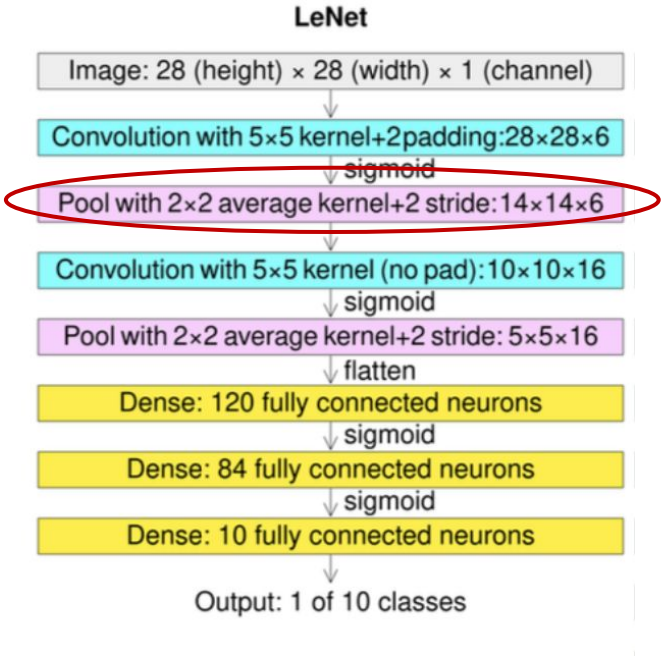
MNIST dataset

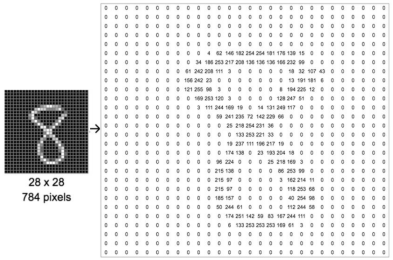
```
# Subsampling layer (MaxPooling) with 2x2 kernel and stride 2
self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
```

LeNet-5

First Pooling Layer:

- This layer uses a 2x2 kernel to perform average pooling with a stride of 2.
- Pooling reduces the spatial dimensions by half (since the stride is 2), resulting in a 14x14x6 volume.





MNIST dataset

```
# Convolutional layer 2: 6 input channels, 16 output channels, 5x5 kernel
self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
```

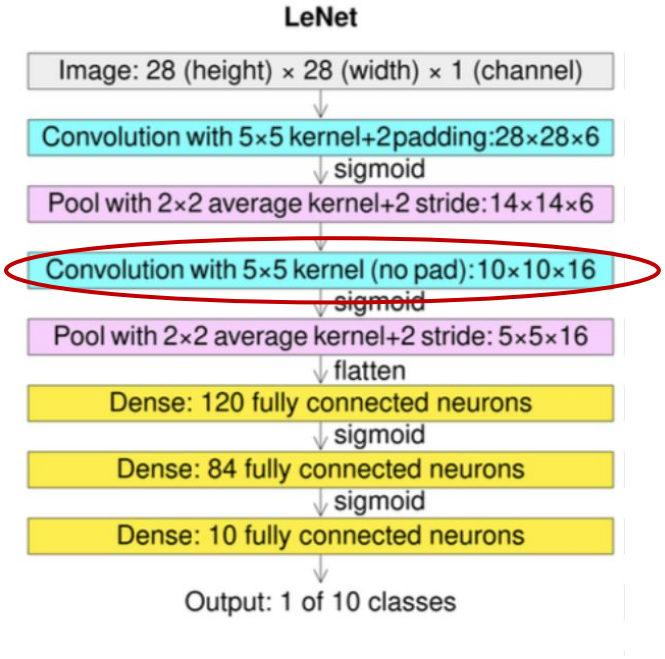
LeNet-5

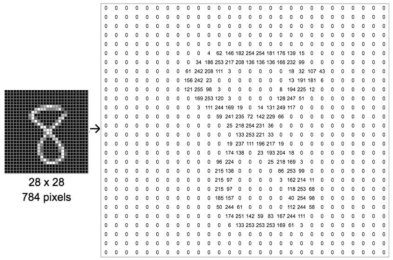
Second Convolutional Layer:

- Another set of 5x5 kernels is used for the second convolution, this time without padding.
- As no padding is applied, the spatial dimensions of the feature maps are reduced, and the output volume is a 10x10x16 feature map (assuming the stride is 1).
- The depth is 16, implying 16 filters are used, creating 16 separate feature maps.

Activation Function:

- Again, a sigmoid activation function is applied after the convolution to add non-linearity.



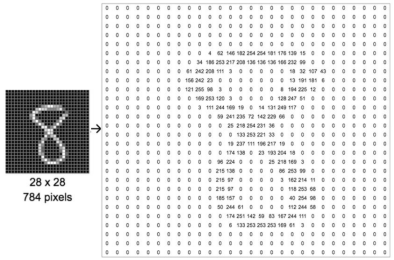


MNIST dataset

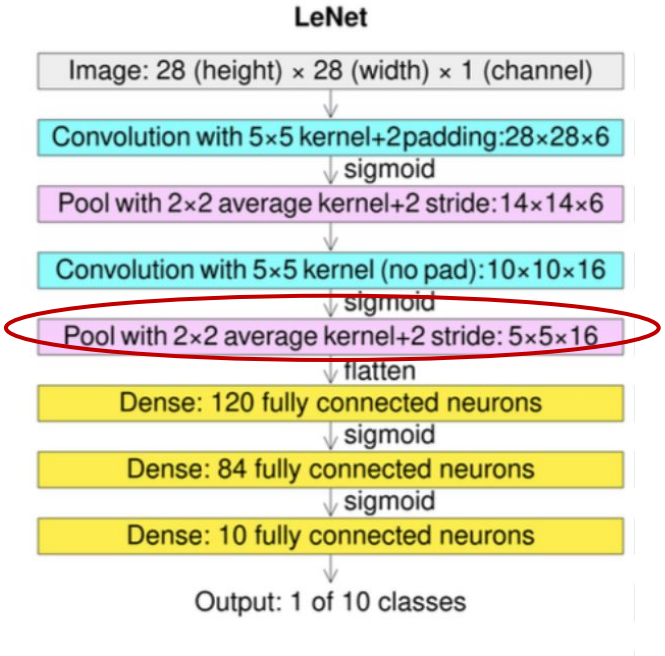
```
# Subsampling layer (MaxPooling) with 2x2 kernel and stride 2
self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
```

LeNet-5

```
# Flatten the output for the fully connected layer
x = x.view(x.size(0), -1)  # Flatten the tensor
```



MNIST dataset



Second Pooling Layer:

- The second pooling layer also uses a 2x2 kernel with a stride of 2 for average pooling.
- The output volume after this pooling step is 5x5x16, as the spatial dimensions are halved again.

Flatten:

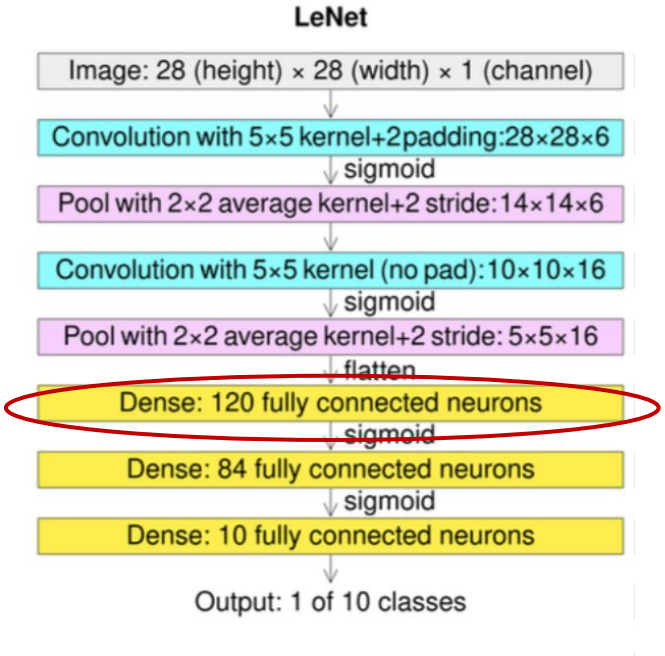
- The pooled feature maps are then flattened into a single vector of values, converting the 3D volume into a 1D vector to serve as input for the subsequent fully connected layers.

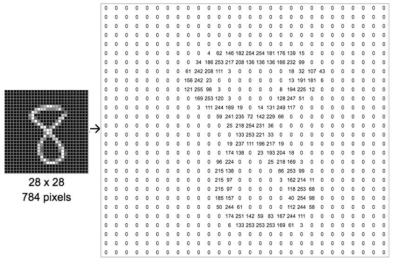
```
# Fully connected layer 1: 16*5*5 input features, 120 output features
self.fc1 = nn.Linear(16*5*5, 120)
```

LeNet-5

First Fully Connected Layer:

- The first dense layer has 120 neurons.
- Each neuron is connected to every element of the flattened vector from the previous layer.
- A sigmoid activation function is used to allow the network to learn complex patterns.





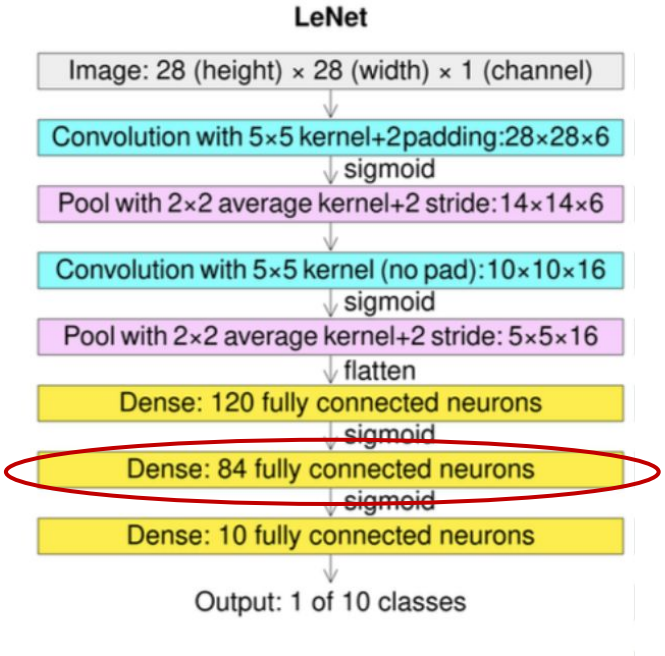
MNIST dataset

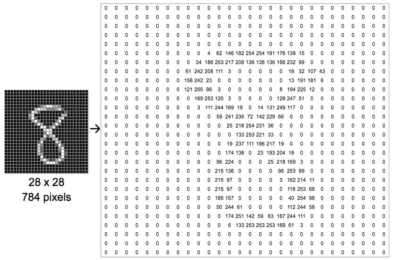
```python
# Fully connected layer 2: 120 input features, 84 output features
self.fc2 = nn.Linear(120, 84)
```

LeNet-5

Second Fully Connected Layer:

- This layer consists of 84 neurons and also uses a sigmoid activation function.
- It takes the output from the first fully connected layer and processes it to identify higher-level features.





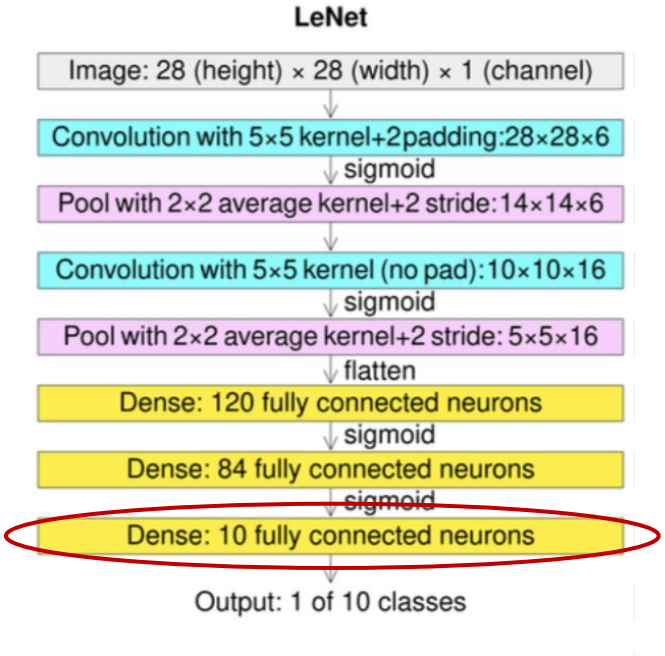
MNIST dataset

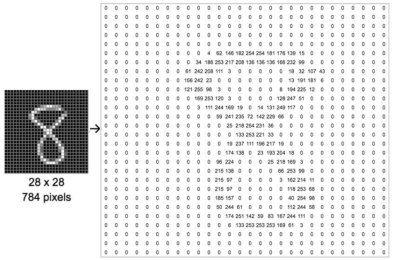
```
# Output layer: 84 input features, 10 output features for 10 classes
self.fc3 = nn.Linear(84, 10)
```

LeNet-5

Output Fully Connected Layer:

- The final dense layer has 10 neurons, corresponding to the 10 possible classes (digits 0-9).
- This layer typically uses a softmax activation function to output a probability distribution over the 10 classes.

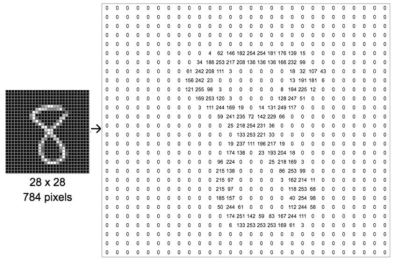




MNIST dataset

```
# Loss function
criterion = nn.CrossEntropyLoss()
```

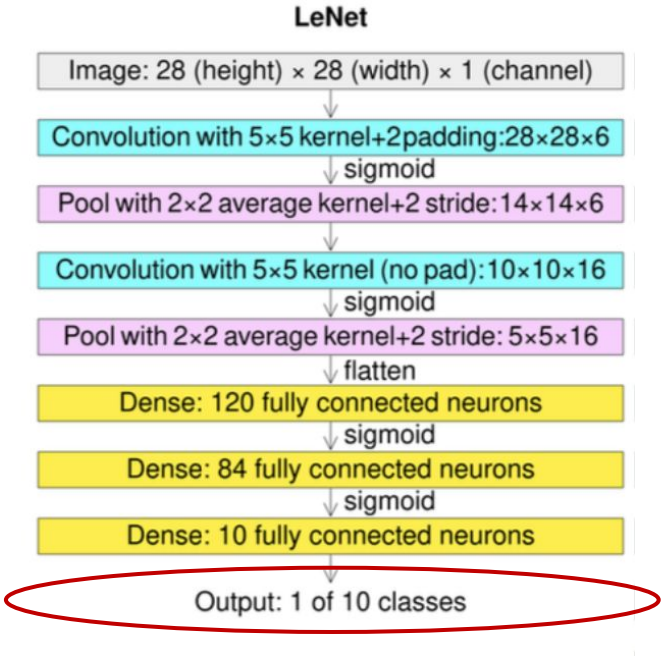
LeNet-5



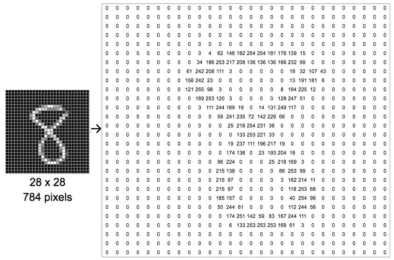
MNIST dataset

Output:

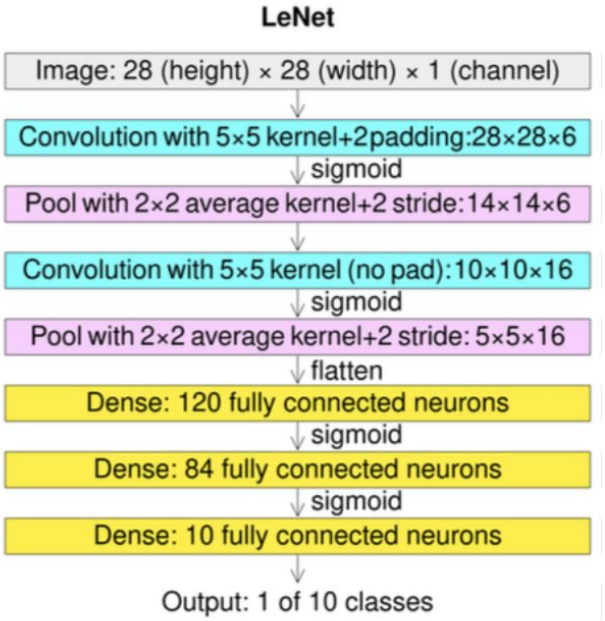
- The output is a probability distribution across the 10 classes. The highest probability indicates the network's prediction for the digit's class.



LeNet-5



MNIST dataset

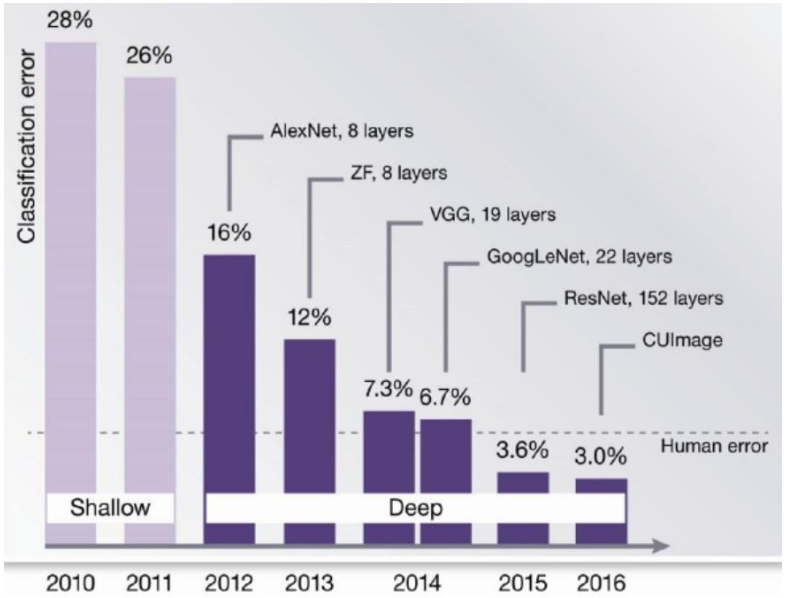


https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/

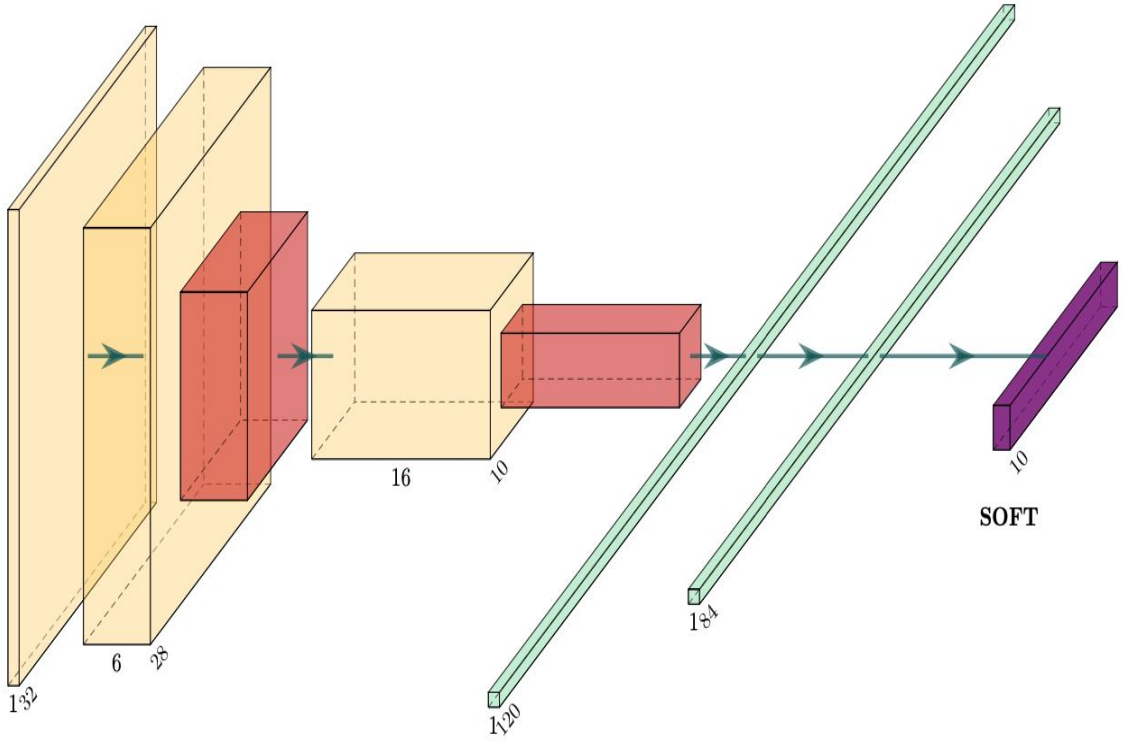
ImageNet

- Large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated.
  https://www.image-net.org/

ImageNet for CNN classifier architectures

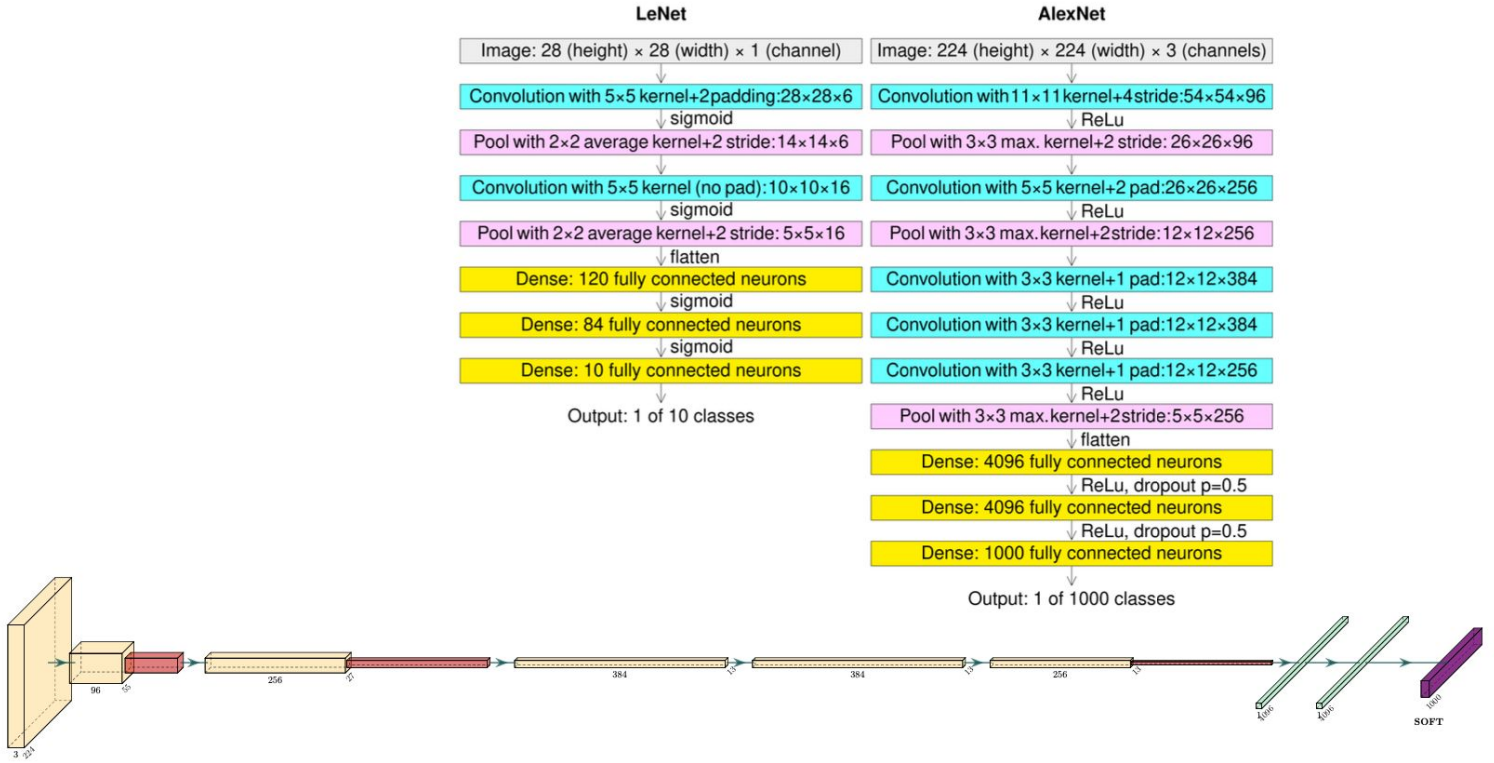


LeNet-5



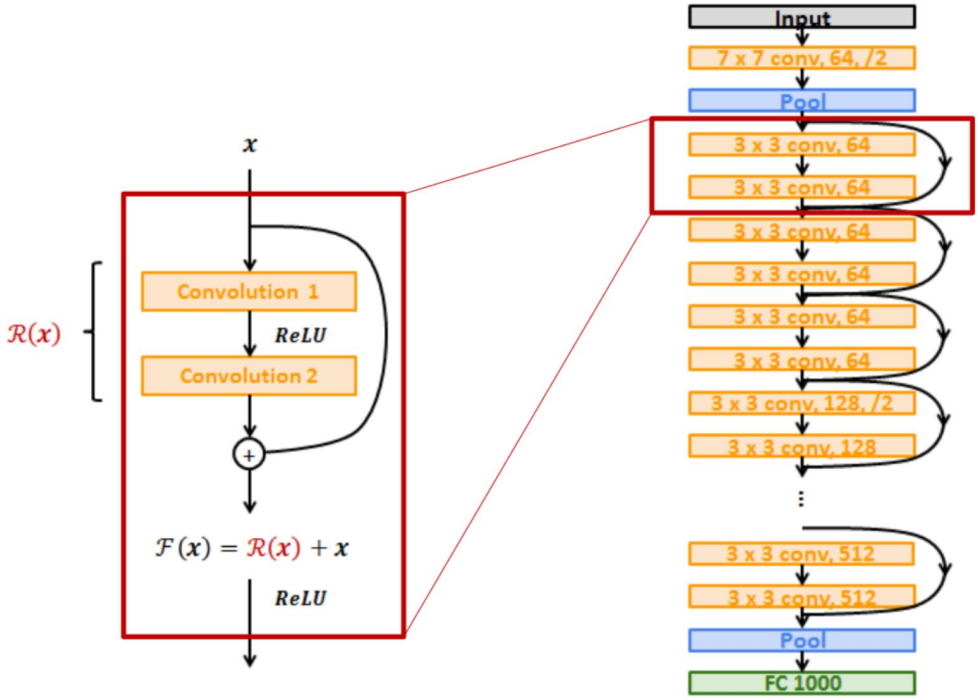
LeCun et al., Neural Computation 1989, "Backpropagation Applied to Handwritten Zip Code Recognition"
LeCun et al., 1998, Proceedings of the IEEE, Gradient-based learning applied to document recognition.

LeNet-5 vs AlexNet



Krizhevsky etal. ImageNet classification with deep convolutional neural networks

ResNet



He et al, CVPR 2016, Deep Residual Learning for Image Recognition

Take home message

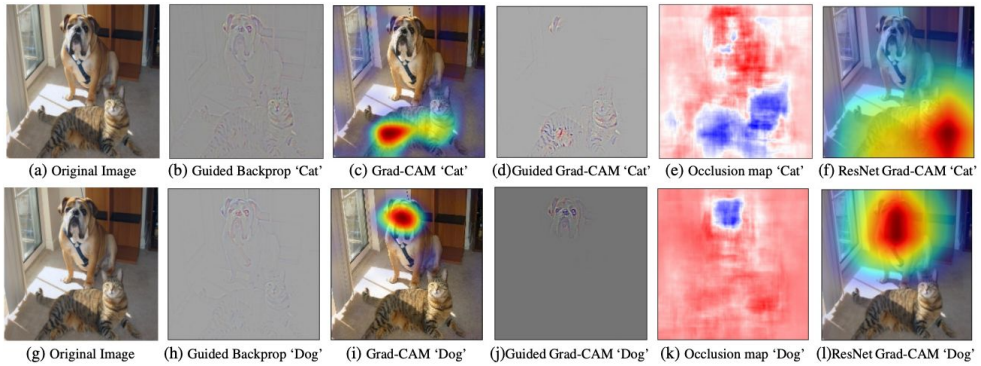
- ⚠️ Don’t start your new network from scratch!
  - Use a model pre-trained on a large dataset (like ImageNet)
  - Adapt a pre-trained model for your specific task with a smaller dataset

Take home message

- ⚠️ Don’t start your new network from scratch!
  - Use a model pre-trained on a large dataset (like ImageNet)
  - Adapt a pre-trained model for your specific task with a smaller dataset

BONUS | Grad-CAM

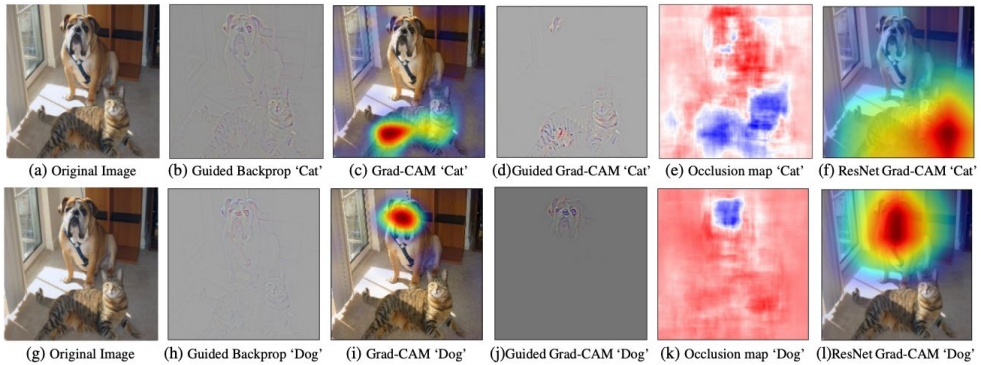
- Gradient-weighted Class Activation Mapping
- Technique for making Convolutional Neural Network (CNN) models more transparent by visualizing the regions of input that are important for predictions from these models
- It helps in understanding which parts of a given image lead the CNN to its final classification decision.



BONUS | Grad-CAM

- Gradient-weighted Class Activation Mapping
- Technique for making Convolutional Neural Network (CNN) models more transparent by visualizing the regions of input that are important for predictions from these models
- It helps in understanding which parts of a given image lead the CNN to its final classification decision.

code



Code - CNN

- Intro to computer vision
  https://drive.google.com/file/d/1OjGTc_qJu4TKMfernC72pE14FAzr0F6p/view?usp=sharing
- LeNet5
  https://drive.google.com/file/d/1Rnc9uNCxLrupZB52-aBgMxO9qIFkwsr2/view?usp=sharing
- Pre-trained ResNet (Transfer learning)
  https://drive.google.com/file/d/1c2v3zScmOCEswbHFWOoPJa3aXl29gUrM/view?usp=sharing
- GradCAM
  https://drive.google.com/file/d/1nHDSDtcDodCJ6lN-BHMeRXaH-uC2GGPf/view?usp=sharing
- ResNet with hyperparameter tuning
  https://drive.google.com/file/d/1leJH1xNIfRsuNnYuPsFsiKqjMdo0Je1w/view?usp=sharing

APPENDIX

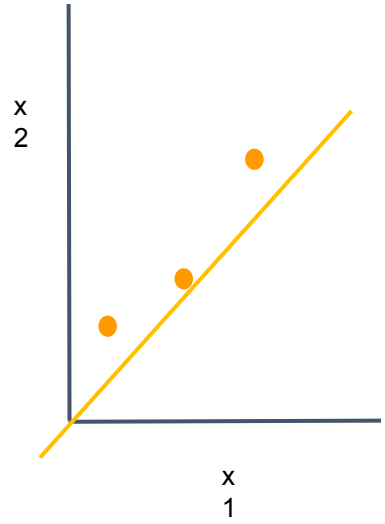
Gradient Descent

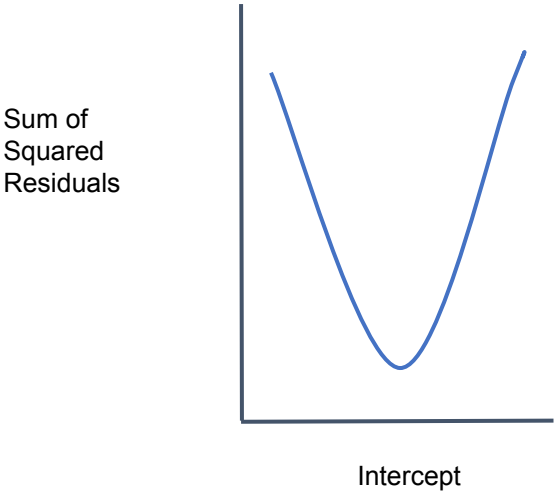
Objective of a Neural Network:

Find weights and biases that minimizes the loss function

Objective of a Neural Network:

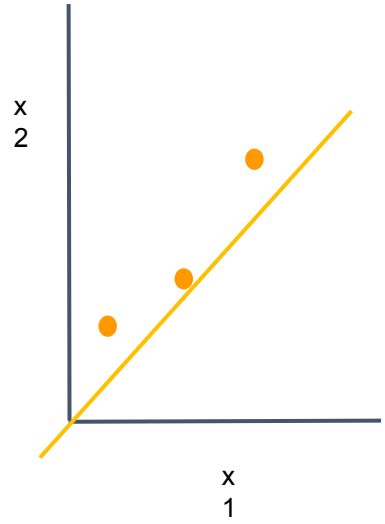
Find weights and biases that minimizes the loss function

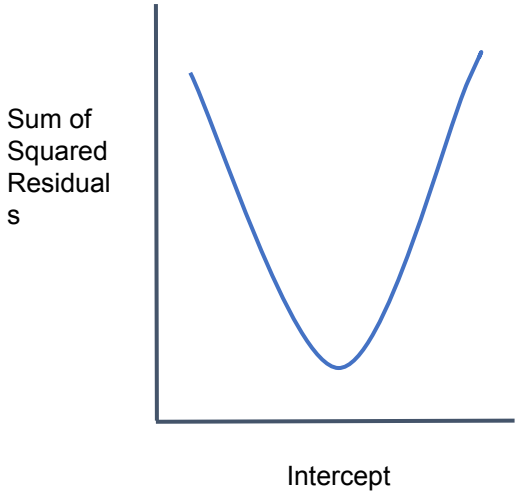




Objective of a Neural Network:

Find weights and biases that minimizes the loss function

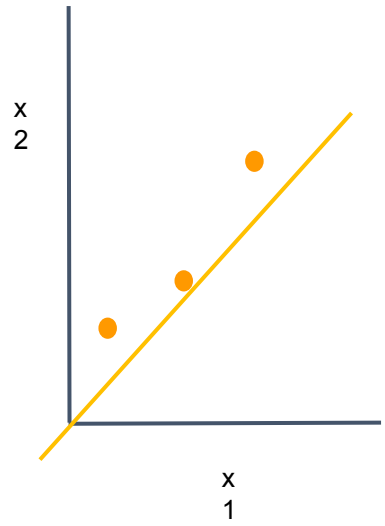


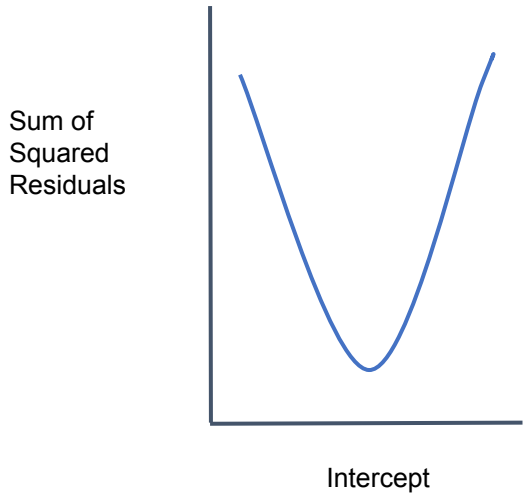


Derivative: Slope of a curve at a specific point

Objective of a Neural Network:

Find weights and biases that minimizes the loss function





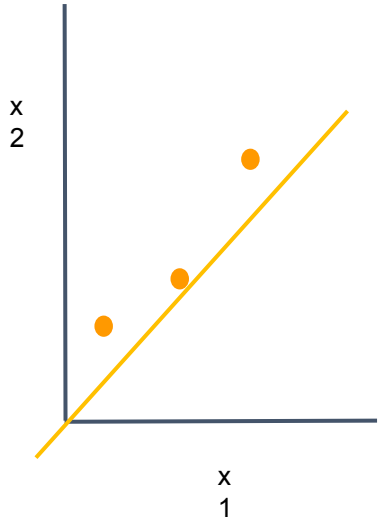
Derivative: Slope of a curve at a specific point

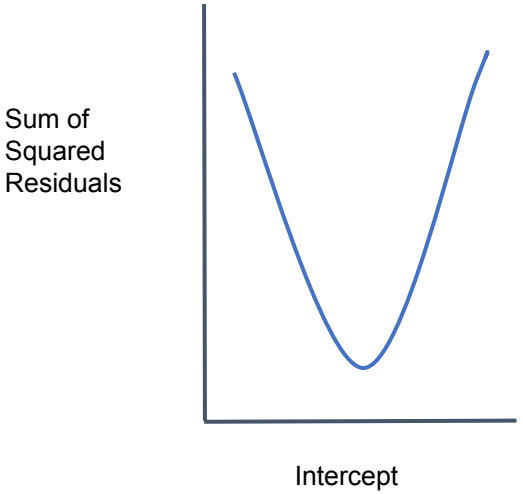
Gradient: Generalization of the derivative. It extends this idea to function of multiple variables.

A gradient is like an arrow that points in the direction of the steepest increase on a surface.

Objective of a Neural Network:

Find weights and biases that minimizes the loss function





If you imagine being on a hill and you want to know which way is uphill, the gradient would point in that direction. If you want to go downhill (like in gradient descent), you'd go in the opposite direction of the gradient.

Derivative: Slope of a curve at a specific point

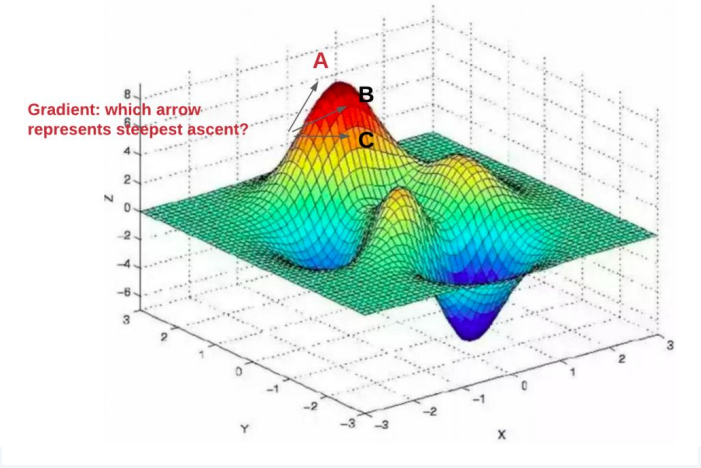
Gradient: Generalization of the derivative. It extends this idea to function of multiple variables.

Gradient descent

Objective of a Neural Network:

Find weights and biases that minimizes the loss function

- The **gradient** of the loss gives you the direction of the steepest ascent
- To minimize, you move the opposite way (towards the steepest descent)



Gradient descent

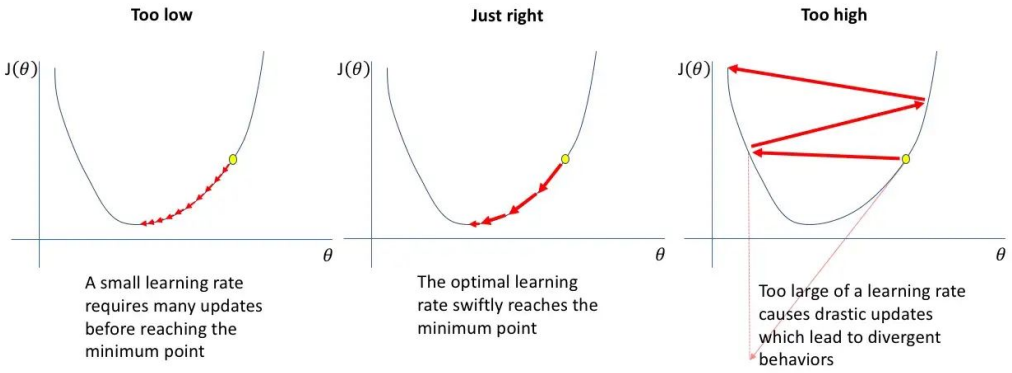
Update rule:

$$w_{new} = w_{old} - \alpha \cdot \nabla_w L$$

Where,

- w_{new} is the new weight
- w_{old} is the old weight
- α is the learning rate
- $\nabla_w L$ is the gradient of the loss L with respect to weight w

Gradient descent



Update rule:

$$w\_new = w\_old - \alpha \cdot \nabla\_wL$$

Where,

- w_new is the new weight
- w_old is the old weight
- α is the learning rate
- ∇_wL is the gradient of the loss L with respect to weight w

Learning rate

It's a hyperparameter, it determines the step size at each iteration while moving towards a minimum of the loss function.

Gradient descent

Backpropagation

- Compute the gradient of the loss with respect to the network's output.
- For each layer, starting from the output layer and moving backward to the input layer:
    - Calculate the gradient of the loss with respect to the layer's output (before activation).
    - Calculate the gradient of the loss with respect to the layer's weights.
    - Use the gradients to update the weights using the update rule.
    - Propagate the gradient backward to the previous layer to continue the process.

The key to computing gradients in the hidden layers lies in the **chain rule of differentiation**. During backpropagation, gradients are propagated backward through the network by multiplying the gradient from the subsequent layer by the derivative of the current layer's activation function.

In practice, deep learning frameworks like TensorFlow or PyTorch handle the backpropagation process automatically, making it easier to train complex neural network architectures.

Types of gradient descent

- **Batch Gradient Descent**: Computes the gradient using the entire dataset. This is computationally expensive and infeasible for large datasets.

- **Stochastic Gradient Descent (SGD)**: Computes the gradient using just one example at each iteration. It's faster and can escape local minima because of its inherent noise, but it's also much noisier and might not converge as smoothly.

- **Mini-Batch Gradient Descent**: A compromise between the two. It computes the gradient using a small random sample (mini-batch) of the dataset. This is the most common method in practice, especially in the context of deep learning.

Types of gradient descent

In practice, **Adam** (Adaptive Moment Estimation) is often preferred over simple mini-batch gradient descent

- Adaptive learning rates
- Momentum
- Bias correction
- Easier to tune
- Efficiency
- Widely used and tested

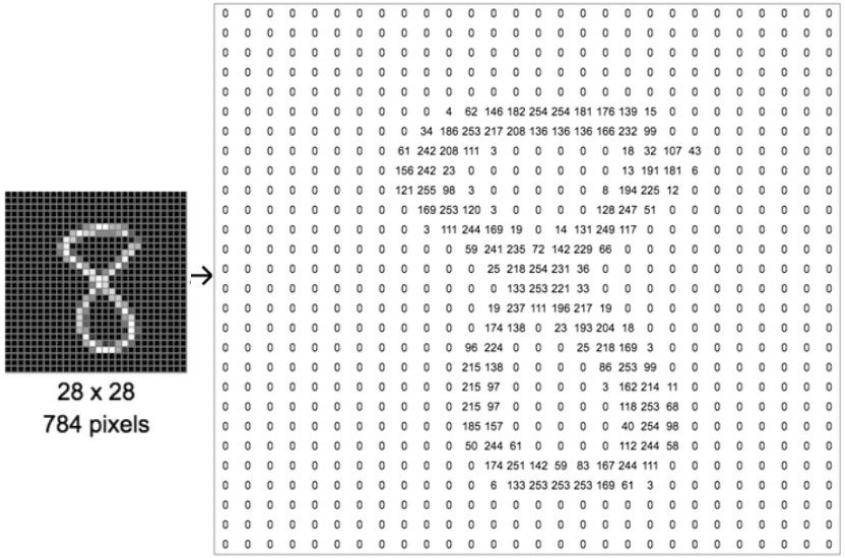
How do we avoid getting stuck in a local minimum?

The problem of local minima is particularly prominent in neural networks due to their complex loss landscapes with high dimensionality.

Some strategies:

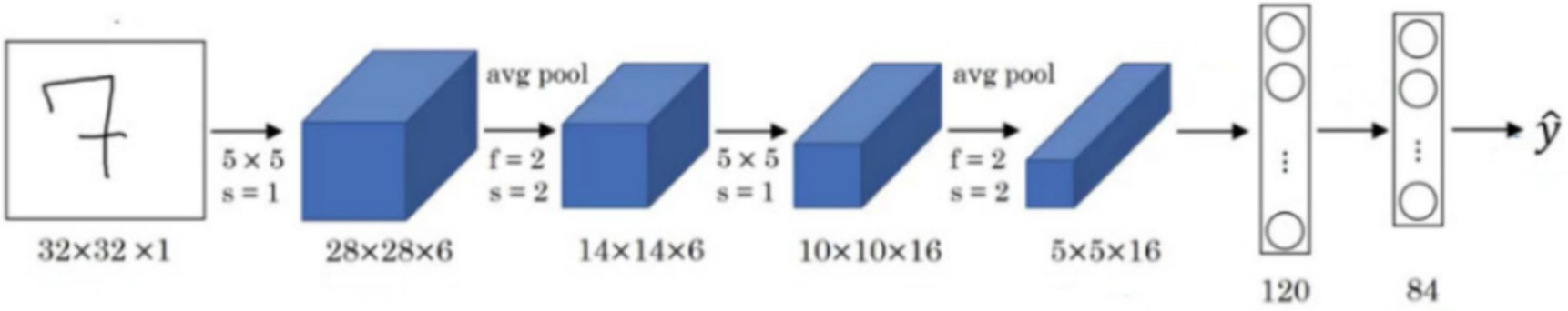
- Use of Momentum-Based Optimizers
- Adaptive Learning Rates
- Stochastic Gradient Descent (SGD)
- Random Restarts
- and more…

Example of a classification dataset
MNIST



- Large database of handwritten digits, commonly used for training
- 10 classes
- 70,000 images
- Images are grayscale

  28x28 = 784 dimensions

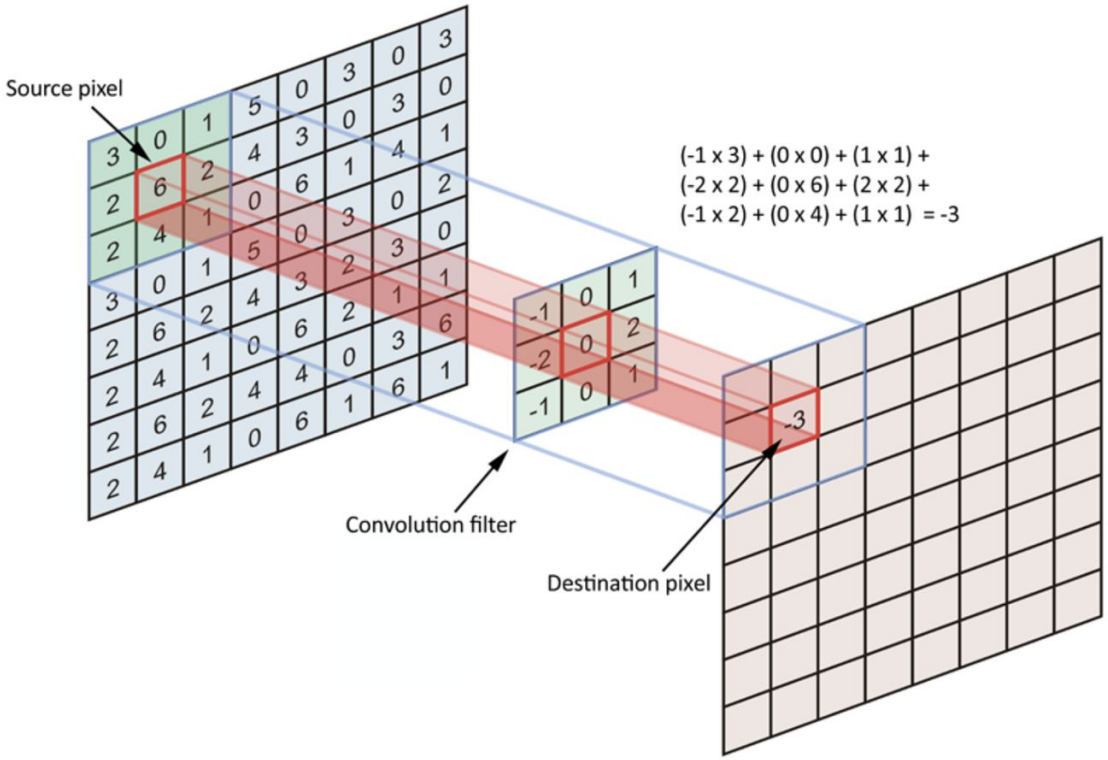


- **Parameters:** 60k
- **Layers flow:** Conv → Pool → Conv → Pool → FC → FC → Output
- **Activation functions:** Sigmoid/tanh and ReLU

When working with the MNIST dataset, this means that the images need to be padded to fit the expected input size of the network.

Convolutional Neural Networks

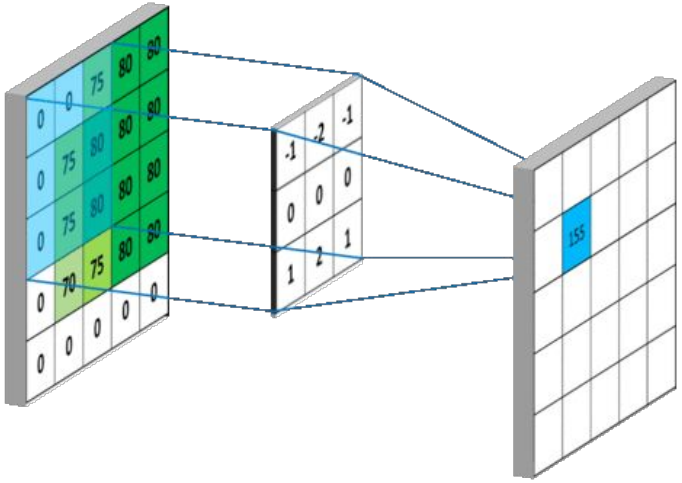
Convolution



Adapted from Harvard CS109B

Convolutional Neural Networks

Convolution



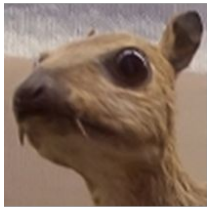
Adapted from Harvard CS109B

Convolutional Neural Networks

"Convolution" operations in action

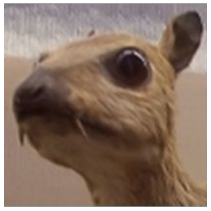
Edge detection

Kernel



$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

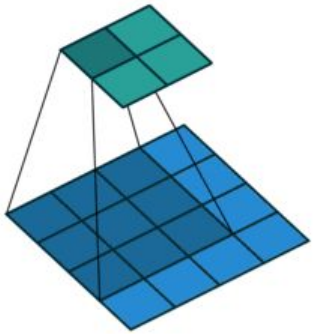
Sharpen



$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

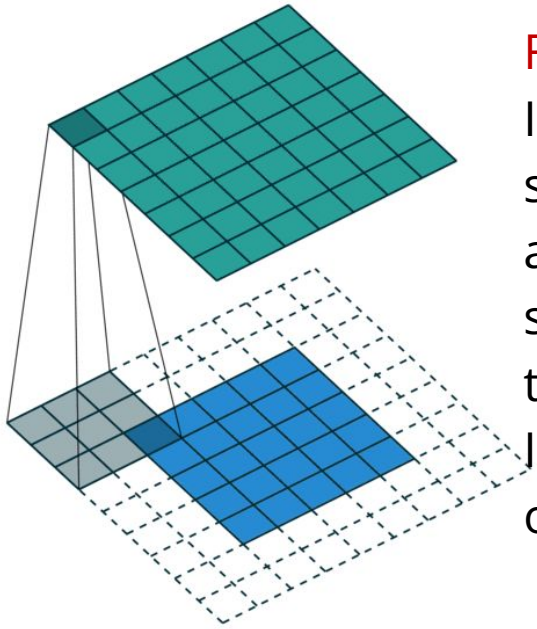
Padding

If we apply convolutions on a normal image, the result will be down-sampled by an amount depending on the size of the filter.

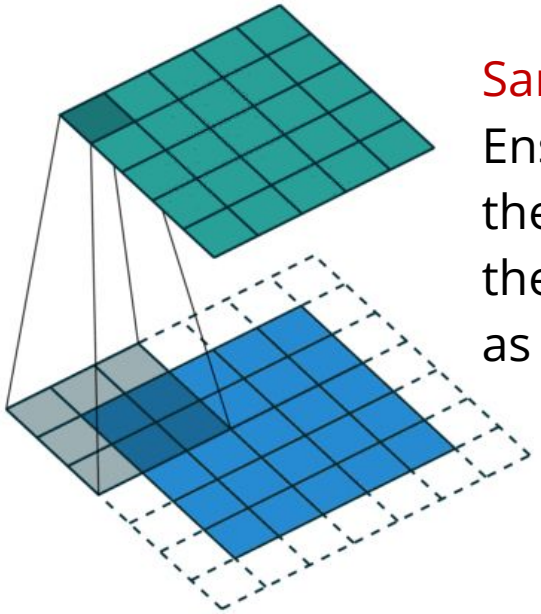


We can avoid this by padding the edges in different ways.

Padding



Full padding.
Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



Same padding.
Ensures that the output has the same size as the input.

Convolutional Neural Networks

Stride

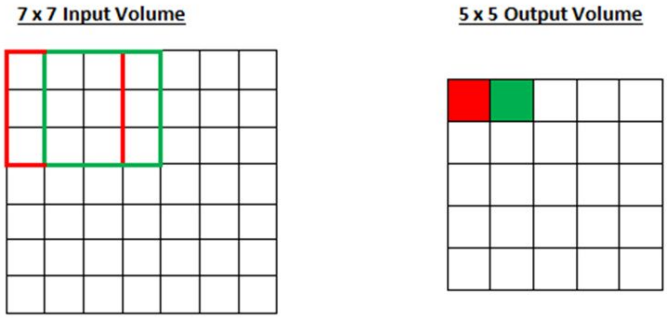
Stride controls how the filter convolves around the input volume.

The formula for calculating the output size is:

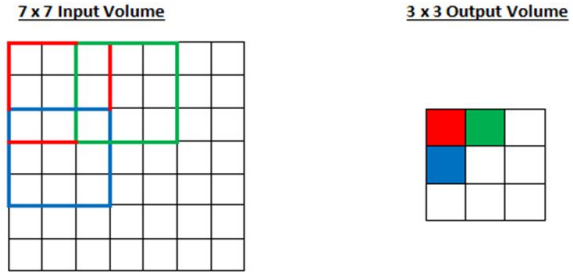
$O=(W-K+2P)/S+1$

Where O is output dim, W is the input dim, K is the filter size, P is padding and S the stride

Stride 1



Stride 2



Adapted from Harvard CS109B

Convolutional Neural Networks

CNN

(Co

Adapted from Harvard CS109B

What do CNN layers learn?

Each CNN layer learns filters of increasing complexity.

The first layers learn basic feature detection filters: edges, corners, etc.

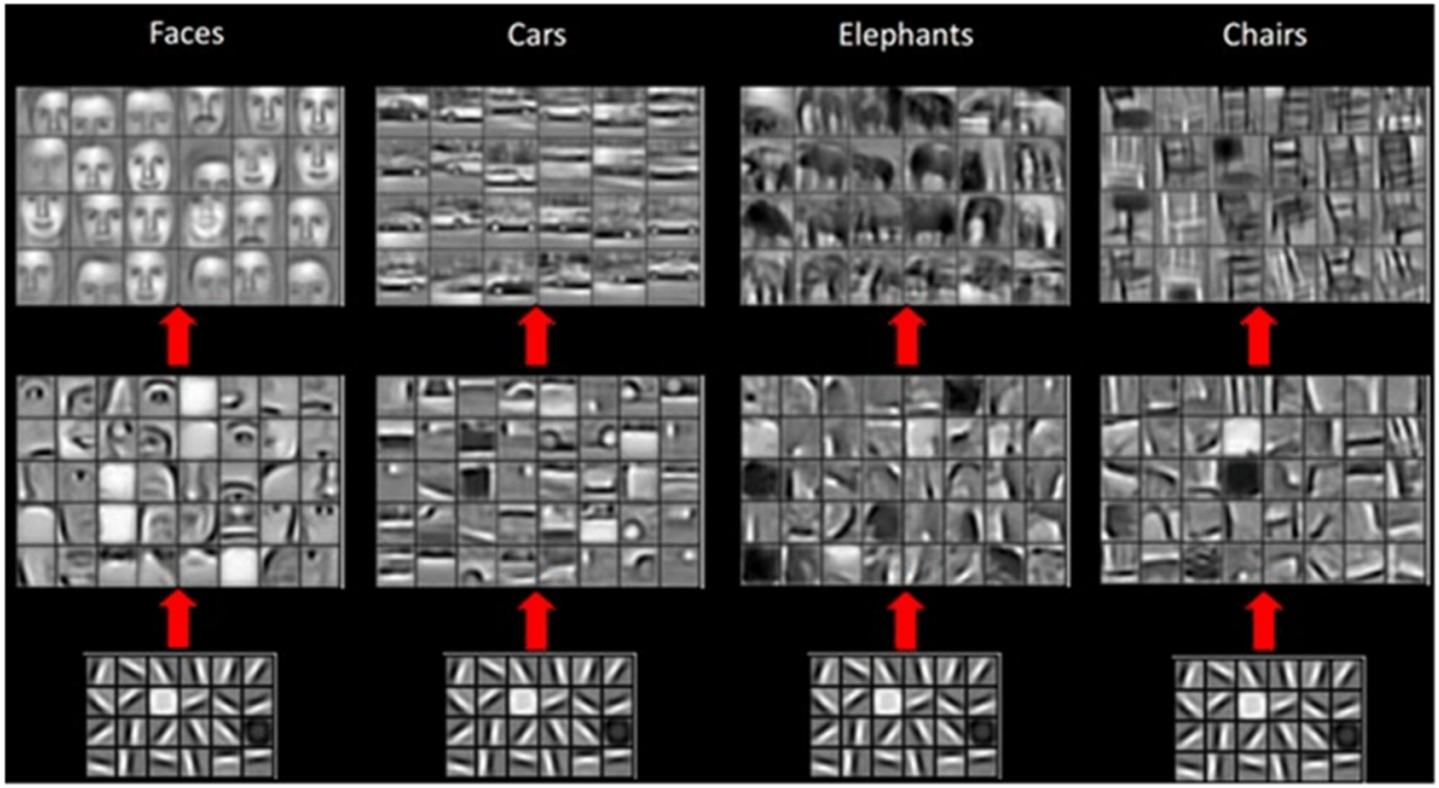
The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.

The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.

Adapted from Harvard CS109B

Convolutional Neural Networks

Fully Built CNN (VGG)



Adapted from Harvard CS109B

Code - LSTM

Generating text
https://colab.research.google.com/drive/14vOI8DdfUEPZ3jf0_fpzpSR8m9NfSfsm?usp=sharing