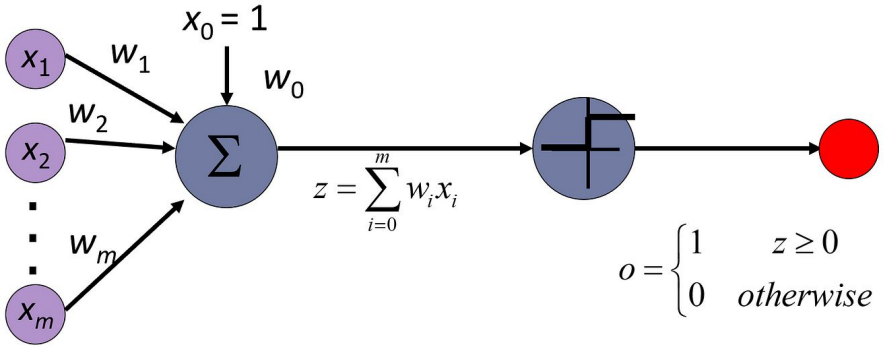
Artificial Neural Networks

Types and use cases
for AI models implementation

The Perceptron



A perceptron (or McCulloch-Pitts neuron) is an algorithm for supervised learning of binary classifiers. It is the most simple form of Artificial Neural Networks.

Invented in 1943 by Warren McCulloch and Walter Pitts, its first implementation was a machine built in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

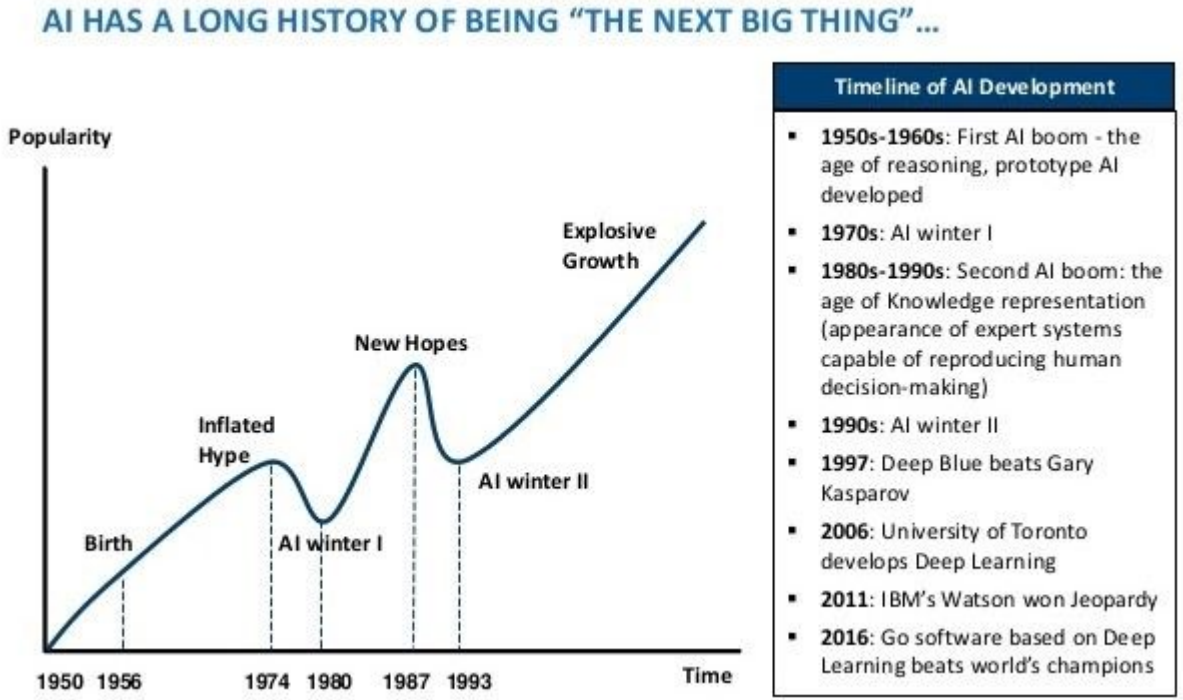
The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron".

This machine, modeled after the eye of a fly, was designed for image recognition and had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

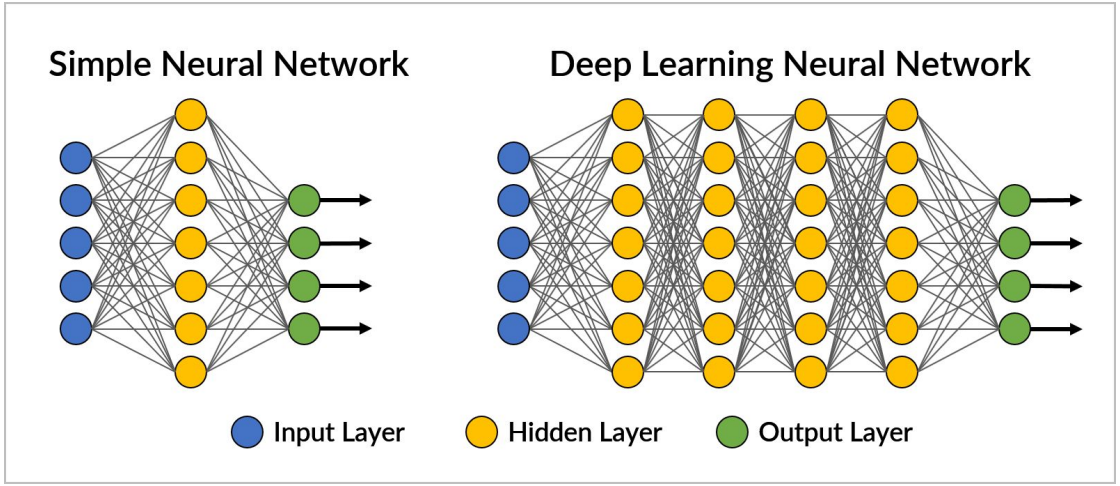
AI booms and winters

After the discovery of the Perceptron, the world of AI fell into its first winter, party in connection with the difficulty related to specific challenges (like the XOR problem)

AI and its winters



Deep Learning

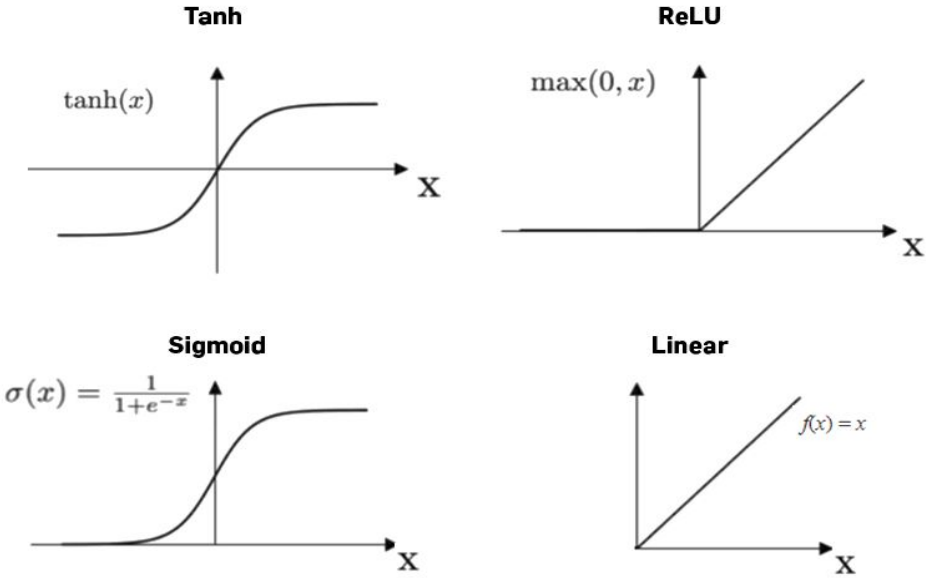


- Is a subset of machine learning, designed to predict non-linear models
- Neural Networks with three or more layers are considered DL models
- Imitates the way humans process data and make decisions
- Powered by advances in large scale data processing and inference

Activation Functions

In a neural network, after an input has been given a weight and a boas, for each layer it goes through an activation function that processes the information.

Here are some of the most common activation functions:



Gradient Descent (Optimization)

An optimizing algorithm used to iterate through different combinations of weights and bias to find the best values producing the lowest error

The loss function is used to quantify the error of a prediction when compared to the known value.

The learning rate decides the size of the steps to take at each iteration to reach a local minimum.

As gradient descent uses derivatives, it cannot work with step functions as they are not differentiable at certain points. Therefore when dealing with binary predictions as in logistic regression, we need to use a sigmoid function (usually with a cut-off mark of 0.5)

Loss Function

The most common Loss Function is the sum of squared errors.

When dealing with logistic regression and some other cases, Cross_Entropy is a better option

Backpropagation

Introduced in 1985 by Hinton, Rumelhart and Williams, it allows to efficiently calculate the parameters (weights and biases) for all layers in a Neural Network.

It calculates the gradient of the parameters of the last layer first and then proceed backwards all the way to the first layer of the neural network.

Parameters that are defined before the training of the network are called **Hyper-paramentes.**

Deep Learning applications and use cases

- Natural Language Processing (NLP)
- Speech Recognition and Synthesis
- Image Recognition
- Self-driving Cars
- Customer Experience, Healthcare, Robotics, etc.

Artificial Neural Network models

When designing a new ANN there a number of considerations to be made:

- What learning rate to chose for the optimization?
- How many hidden layers do we need?
- How many neurons for each layer?
- Which activation function should we use for each layer?

To experiment with some models for ANNs, try the Tensor Flow Playground tool:

https://playground.tensorflow.org/

Deep Learning Architectures

- **Convolutional Neural Networks (CNN)**

- **Recurrent Neural Networks (RNN)**. Variants include:
  - Gated Recurrent Unit (GRU)
  - Long/short term memory (LSTM)

Sequence Models

Basic deep learning models can predict output given set of inputs, but work independent of order in time and cannot model relationships across time.

By contrast, **Sequence Models** can

- Predict future sequences based on past sequence patterns
- Keep memory of what happened in the past,
- Make use of a **Lookback** period (predefined amount of past data used for prediction)
- Predict multiple future occurrences
- In the case of Bi-directional Models, predict based on occurrences before and after

RNN Architecture

- The same neural network is used in recurrent way over a series of input fed in chronological order.
- At each step one of the inputs is used through the same network, using the same layers, nodes weights, biases, and activation functions.
- Depending on the use case the network can be simple or deep
- **At each time step the network receives two inputs: the current input plus a hidden state produced by the previous input.**
- Each time step also produces two outputs: a current output plus the hidden state to be fed to the next step

Types of RNN architecture

Several types of RNN architectures are available, depending on the use case:

- Many-to-Many RNNs. Each time step has an input and an output.

  Ex: Speech Recognition

- Many-to-One RNNs. Each time step has an input but one single output is produced at the end of the sequence.

  Ex: Stock Price Prediction, Sentiment Analysis

- One-to-Many RNNs. A single input produces multiple outputs (rare).

  Ex: Music Synthesis

- Encoder-Decoder RNNs. Multiple input go into an encoder, then multiple outputs come out of a decoder. This is architecture is used for transformers

  Ex: Machine Translation, Text Summarization

RNN applications and use cases

- Time series forecasting
- Text classification
- Topic modeling
- Sentiment analysis
- Named entity recognition
- Speech to text
- Text to speech
- Machine translation
- Question answering
- Text summarization

Gated Recurrent Unit

- Regular RNNs pass on the previous hidden states to the next time step, with incremental decay
- GRUs use an “update gate” to decide ir the previous hidden state needs to be fully passed on or fully ignored. Some implementation use a separate “reset gate”
- Ensure long-term memory for certain features

Long Short Term Memory (LSTM)

- LSTM uses multiple gates (normally three: Update gate, Forward gate, Output gate) to decide if the previous hidden state need to be passed on or ignored
- In contrast with regular GRUs, LSTMs can used both the previous hidden state and the current input to compute the next hidden state, assigning a percentage weight to each of them
- Ensure long term memory for certain features with more accuracy, achieving better results for longer sequences
- Adds computational overhead, thus is less efficient than GRUs

Bidirectional RNNs

- A time step in a sequence may be dependent upon both:
  - What occurred in **previous time steps**
  - What occurs in **following time steps**

- BRNNs can pass hidden states in both directions:
  - Forward hidden state
  - Backwards hidden state

- Popular use cases include:
  - Named Entity Recognition
  - Speech Recognition
  - Grammar check

Time Series Patterns

A time series can show several overlaying patterns:

- Cyclic patterns: repeating over a relatively short time period
- Seasonal patterns composed of successive cycles
- Growth patterns: long term trends made of changes across seasonal patterns

When building a sequence model to analyze a time series, we need to ensure it will be able to capture multiple time patterns

Word Embeddings

- A 2D matrix of tokens (words) with corresponding features
- Tokens are assigned scores for each of the feature (ranging from -1 to 1)
- Tokens that are similar with respect to a feature are assigned similar scores
- Similar list of scores = semantically similar tokens
- A list of scores for a token (embedding vector) is used as its representation for machine learning

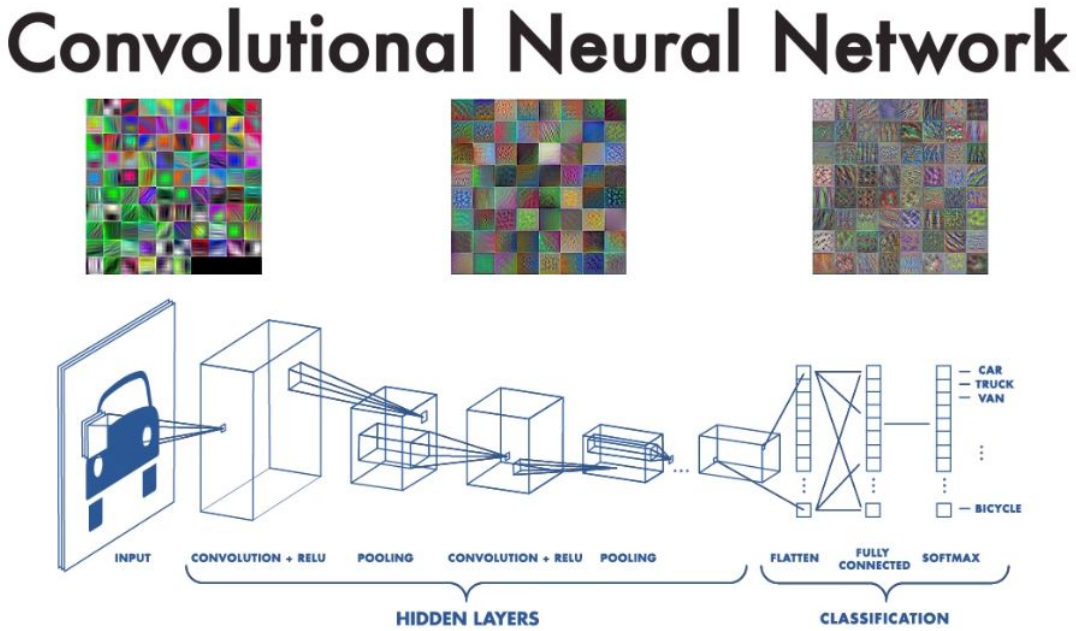
Pretrained Embeddings

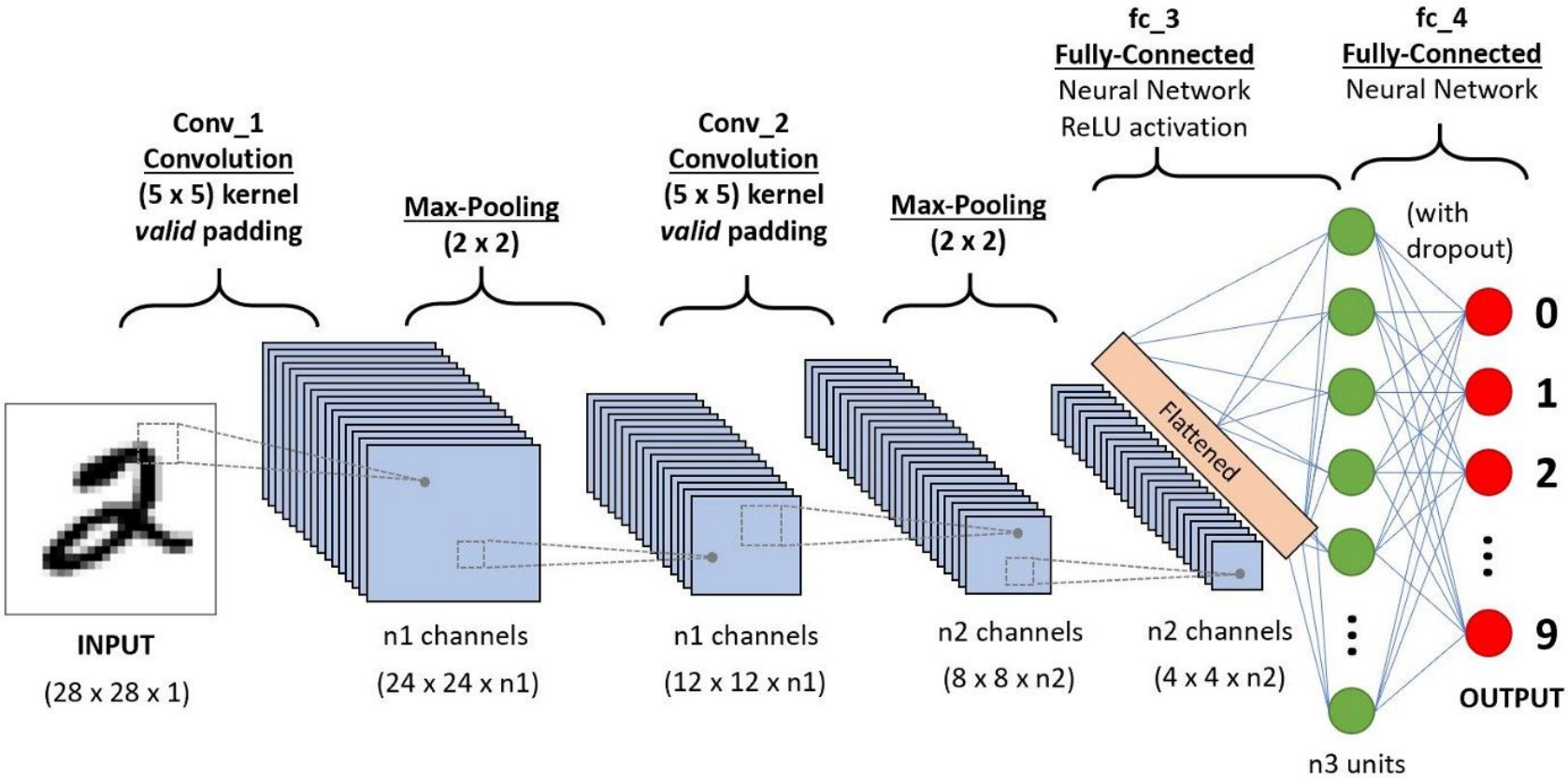
- Relationships between words are constant for a given language model and are usually not affected by the use case
- Embeddings created for one use can case can be reused for others
- Open-source embeddings (or word vectors) are available for reuse
- Multiple token (word) counts and feature (dimension) counts are available
- Popular embeddings include GloVe, Word2Vec, Elmo

nlp.stanford.edu/projects/glove/

Convolutional Neural Networks

- Convolution
- Pooling
- Image flattening
- Input into the NN
- Hidden Layers
- Output layer
- Classification




Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

Convolution operation

The convolution process analyzes the original image with a filter, using the dot product to output a reduced format image.

The formula below tell us the new image width equals the original width, minus the width of the filter plus 2x the padding size, divided by the size of each step taken by the filter, all incremented by 1.

$$W_{out} = \frac{W - F + 2P}{S} + 1$$