Machine Learning – Overview

June 2023

Main Areas of Artificial Intelligence

from wikipedia

- **computer vision**
  (spatial structures, state-of-the-art: Convolutional Neural Networks)
- **natural language processing**
  (sequential structures, state-of-the-art: transformers)
- **automated decision making, robotics**
  (reinforcement learning)

agency:
perception – thought – action

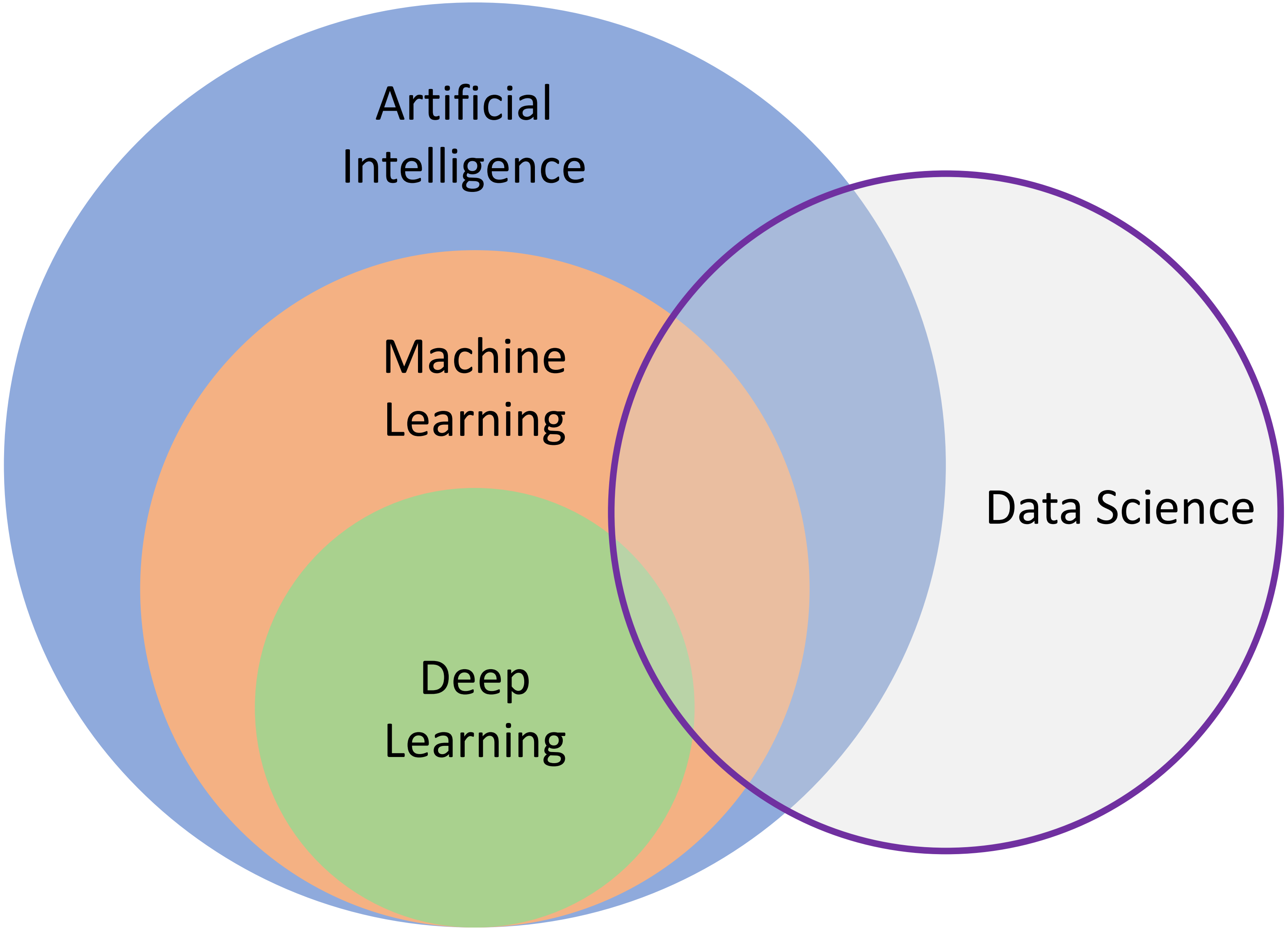
All of these are enabled by one key ingredient:

- *learning from experience* (**Machine Learning**)

- knowledge representation, automated reasoning

from *Artificial Intelligence: A Modern Approach* (Russell, Norvig)

Buzz Words ...



Deep Learning:

special kind of ML algorithms using (deep) neural networks

Data Science:

extract knowledge from data (by means of ML, among other things)

Traditional Algorithms and GOFAI

from wikipedia

traditional algorithms:

explicit (handcrafted) instructions for each situation

symbolic AI (aka GOFAI):

use knowledge by means of symbols (as representations), logic, search (e.g., expert systems like Deep Blue)

Public perception is changing over time: A modern chess program, nowadays disparaged as brute computing, would have been considered intelligent in the 50s.

ML: Learning from Experience/Data

mainly exploiting statistical dependencies with the aim of **generalization** to new (e.g., future) data

training (usually offline optimization):

ML algorithm + data = explicit algorithm (to be used at inference time)

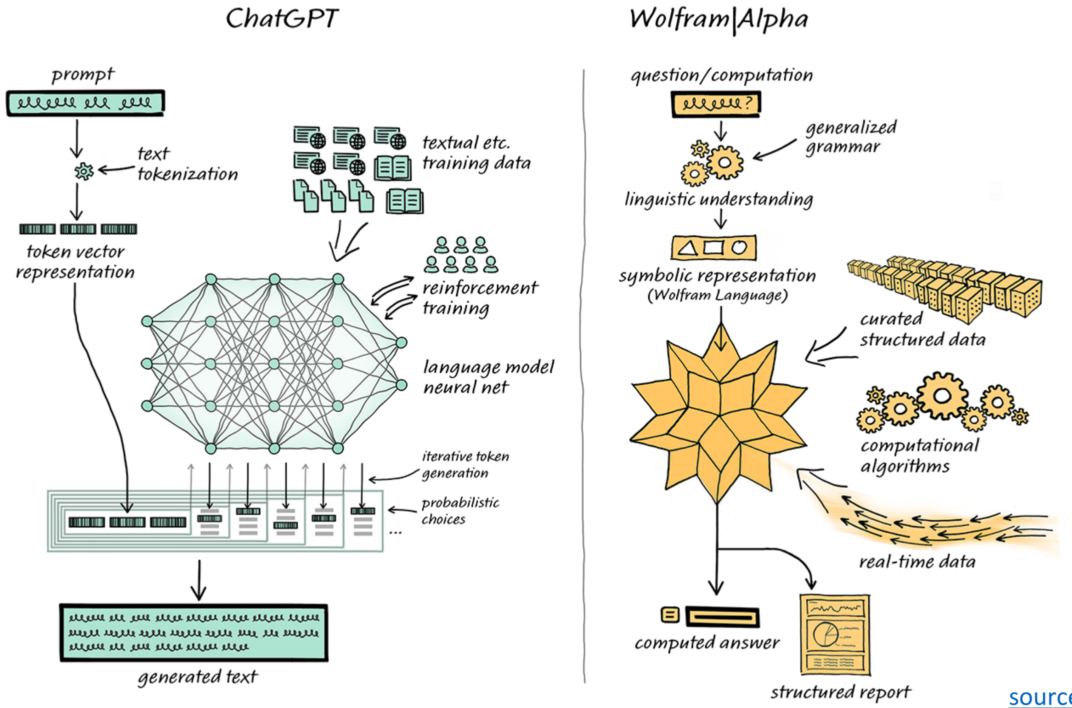
→ reduction of complexity and much better generalizability compared to handcrafted algorithms

analogy: Humans do not hit the ground running (storage capacity of DNA limited), but have learning capabilities.

Hybrid Approaches for ML and Symbolic AI

famous move 37
from Max Tegmark

Deep Reinforcement Learning
and Monte Carlo Tree Search



source

feature engineering for ML models also
kind of symbolic knowledge representation

tool usage:
LangChain

Supercharging the Scientific Method

use ML and data to replace or enhance explicit methods relying on detailed domain knowledge (Software 2.0)

→ overcome our evolutionary limitations in math with clever learning algorithms and collecting data

→ immediate impact on many aspects of industry, business, and science, formulated as narrow tasks with strictly defined inputs (aka weak AI)

more imminent than (still philosophical) long-term quest for human-level AI (aka strong AI, AGI), i.e., general-purpose intelligence

(although recent language models show multi-purpose capabilities)

When to apply ML?

complexity

- decisions under uncertainty, many influencing factors
- e.g., demand forecasting, DNA sequencing
- difficult for humans, direct model inexpressible

automation

- e.g., face and speech recognition, autonomous driving
- goal to reach human-level performance

… and of course you need data to learn from

and more recently:
generative tasks

- rather than predictive (or discriminative) ones
- e.g., image generation, conversational AI, new proteins or materials

Learning Paradigms

Supervised Learning

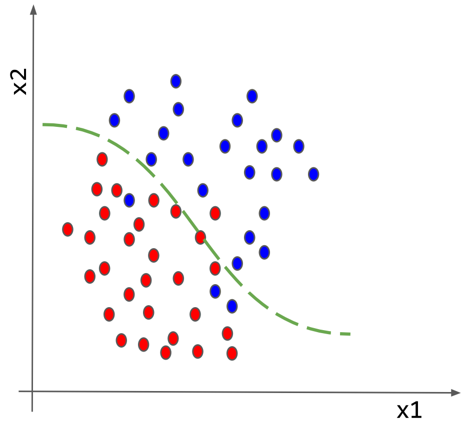
learning by teacher → usually rather narrow tasks

Target Quantity

- **known in training**: labeled samples or observations from past
- to be **predicted** for unknown cases (e.g., future values)

Features

input information that is
- correlated to target quantity
- known at prediction time



Example: Spam Filtering

Classify emails as spam or no spam

*use accordingly **labeled emails as training set***

use information like **occurrence of specific words** or **email length** as **features**

features x1 and x2
spam, no spam

Reinforcement Learning

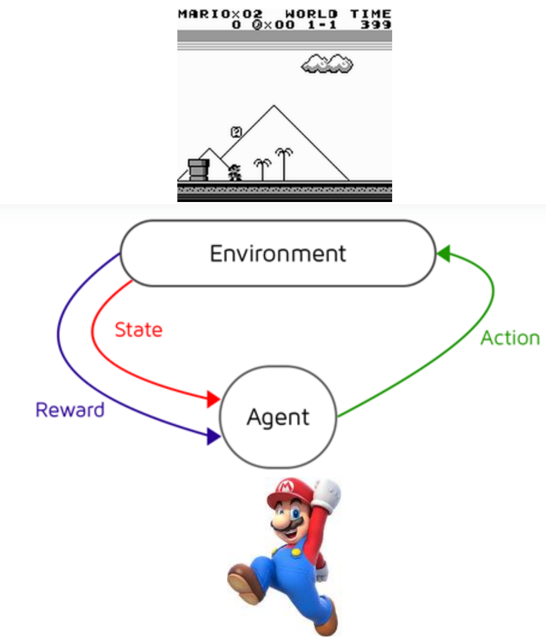
learning by trial-and-error (exploration and exploitation)

- goal-based approach → more generic than supervised learning (but sparse reward signals)
- receiving feedback from the environment, no supervision
- formalization of sequential decision making (delayed rewards)

corresponds to search for best action policy to reach a given goal (e.g., win a game)

using learning from examples (data) to guide the search

RL usually more difficult (e.g., non-differentiable as a whole) than supervised learning (which can be seen as "generalized optimization", often of proxy metric)

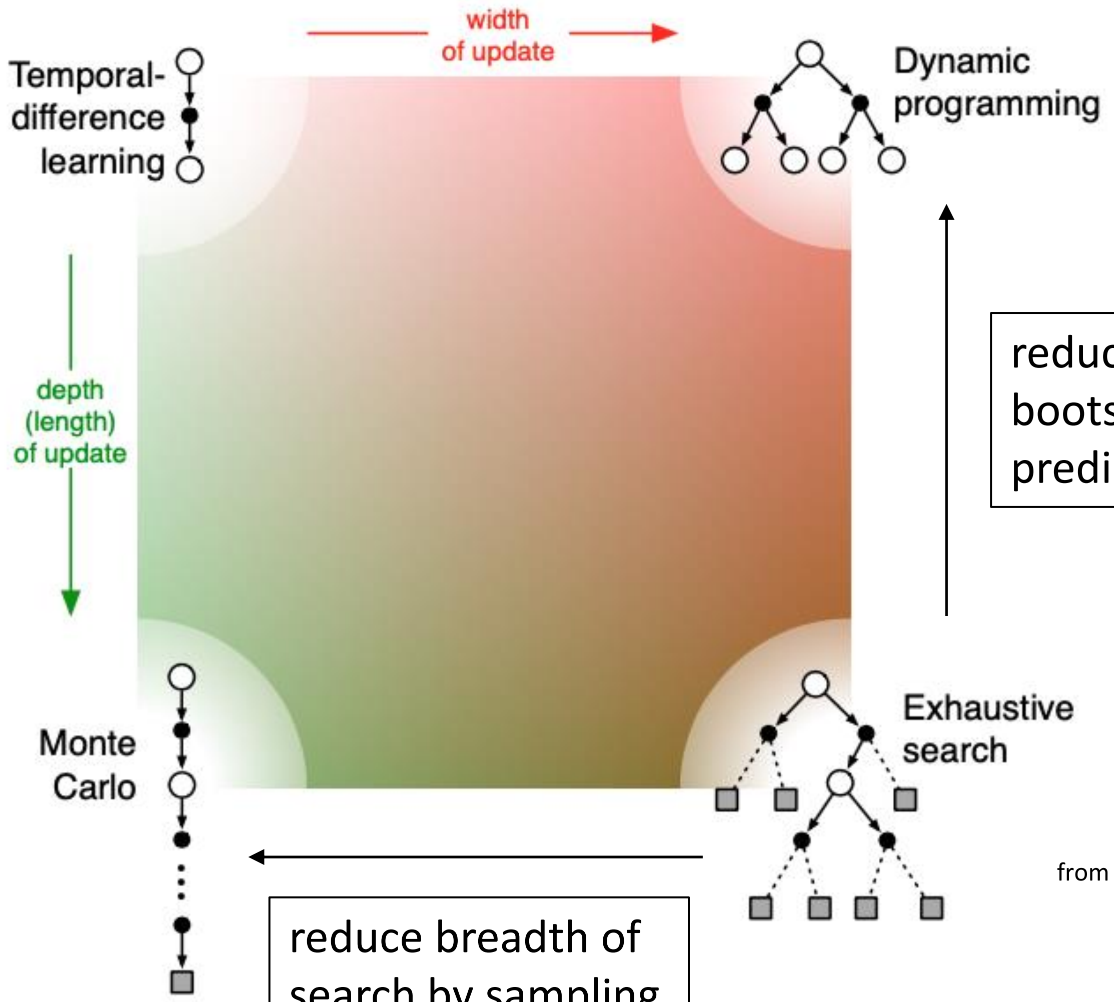


Reduction of Search Space

one-step TD

n-step TD

MC



reduce depth of search by bootstrapping with value function predicting future outcome

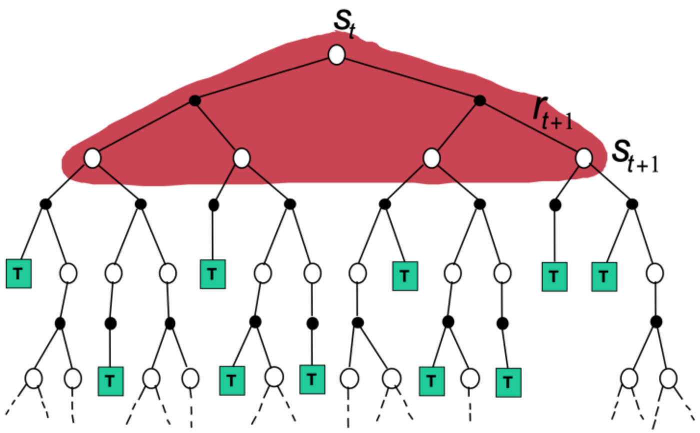
reduce breadth of search by sampling actions from policy

from Sutton

Main Concepts of Value-Based RL Methods

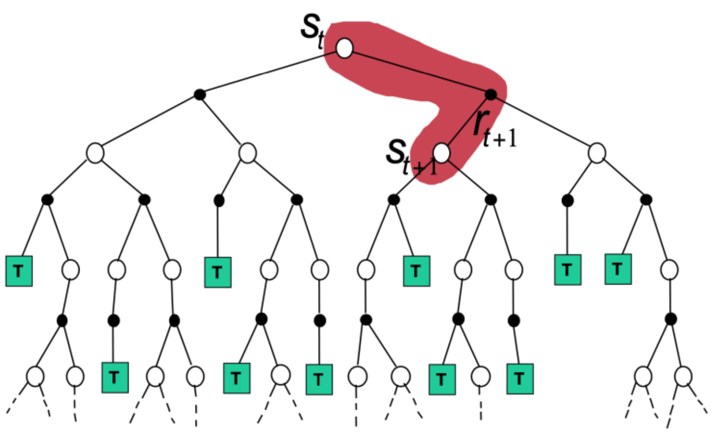
bootstrapping: update estimates of state values based on estimates of values of successor states
sampling: experience of sample sequences (no need for complete knowledge of environment)

Dynamic Programming



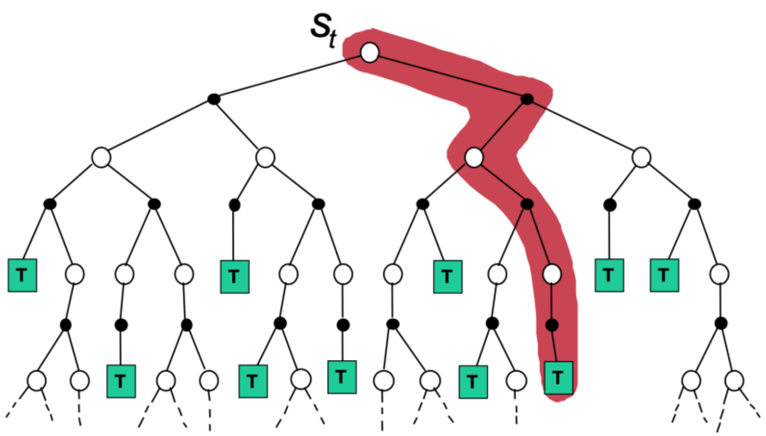
- bootstrapping
- no sampling → model-based (transition probabilities needed)

Temporal Difference (TD) Learning



- bootstrapping
- sampling → model-free

Monte Carlo (MC)



from Sutton

- no bootstrapping
- sampling → model-free

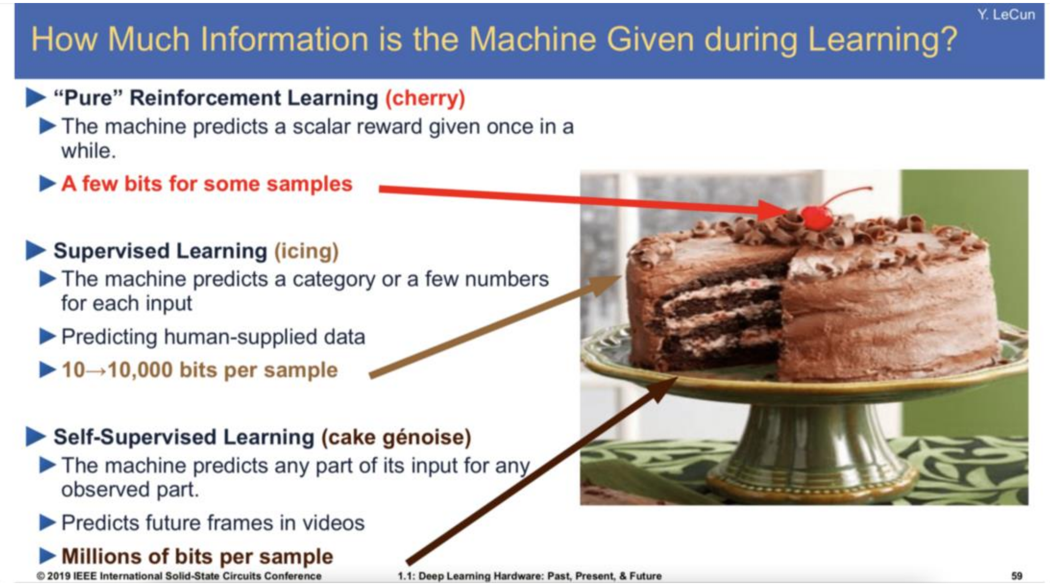
Unsupervised Learning

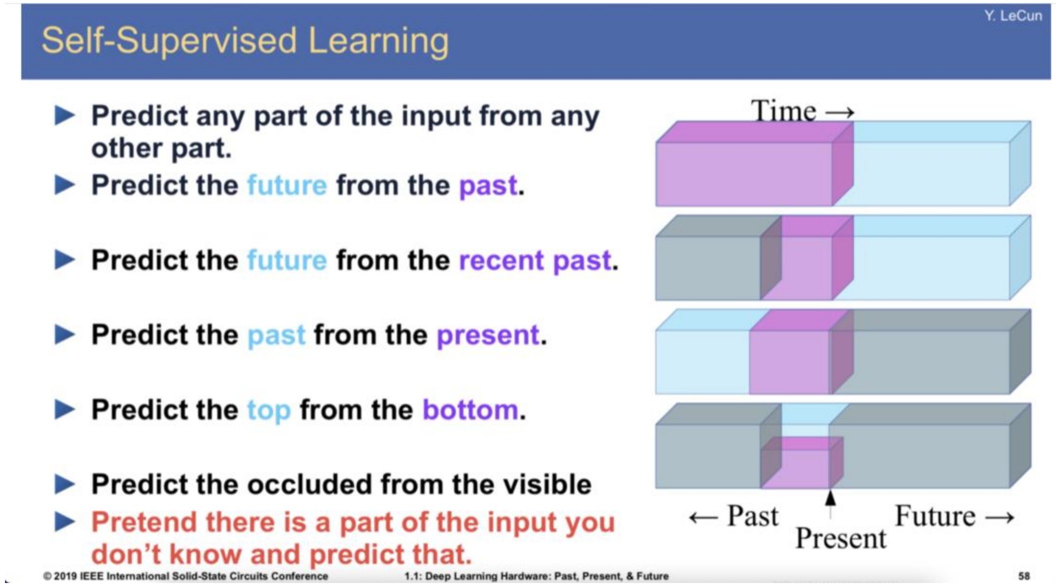
learning by observation

no target information → kind of "vague" pattern recognition (but plenty of data)

self-supervised:

- input-output mapping like supervised learning
- but generating labels itself from input information
- learning of semantic feature representations
- e.g., word2vec, BERT, GPT



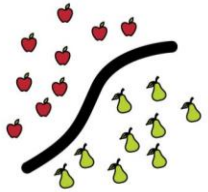


MACHINE LEARNING

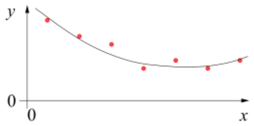
training target available (labeled or past data)

SUPERVISED

CLASSIFICATION



REGRESSION



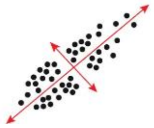
data not labeled in any way

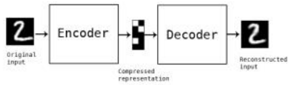
UNSUPERVISED

CLUSTERING

ASSOCIATION

DIMENSIONALITY REDUCTION





no supervision, but goal-based interaction with environment

REINFORCEMENT LEARNING

LEARN STATE OR ACTION VALUES

LEARN POLICY DIRECTLY

learning by teacher
(high-dimensional curve fitting)

learning by observation
(pattern recognition)

learning by trial-and-error
(sequential decision making)

Fitting and Generalization

... **ML** ...

Supervised Learning Scenario

ML domain:
no deterministic dependencies between input and output

map inputs to output: $y = f(\boldsymbol{x})$ (estimated: $\hat{f}(\boldsymbol{x})$)

random variables Y and $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)$ usually many dimensions

fit train data set of $(y_i, \boldsymbol{x}_i)$ pairs

(i.i.d. assumption: random samples from underlying data-generating process)

then apply learned statistical dependencies to test data set

classification:

categorical target: $y = 0$ or $y = 1$ (e.g., image of cat or not), predict probabilities

regression:

real-valued target: $Y \in [0, \infty)$ (e.g., demand forecasting) or $Y \in (-\infty, \infty)$

Generalization

core of ML:

empirical risk minimization (training error) as proxy for minimizing unknown population risk (test error, aka generalization error or out-of-sample error)

generalization gap: difference between test and training error

interpolation: to unencountered samples from training environment

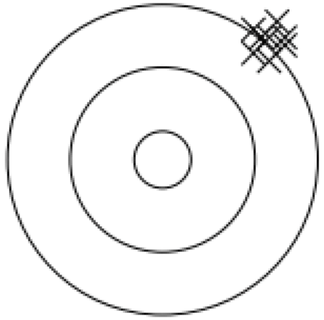
extrapolation: to testing conditions differing from training environment

curse of dimensionality: *“learning in high dimensions always amounts to extrapolation”*

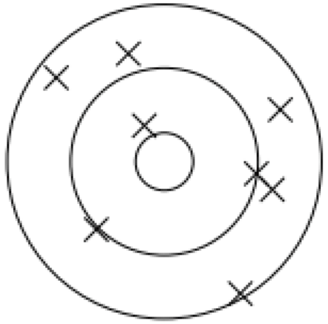
→ need for appropriate **inductive bias** (aka learning bias): set of assumptions of a learning algorithm to predict outputs of inputs not encountered during training (“data in disguise”)

Bias, Variance, Irreducible Error

think of fitting ML algorithms as repeatable processes with different (i.i.d.) data sets



bias:
due to too simplistic model
(same for all training data sets)
"underfitting"



variance:
due to sensitivity to specifics (noise)
of different training data sets
"overfitting"

irreducible error (aka Bayes error):

inherent randomness (target generated from random variable following probability distribution)

→limiting accuracy of ideal model

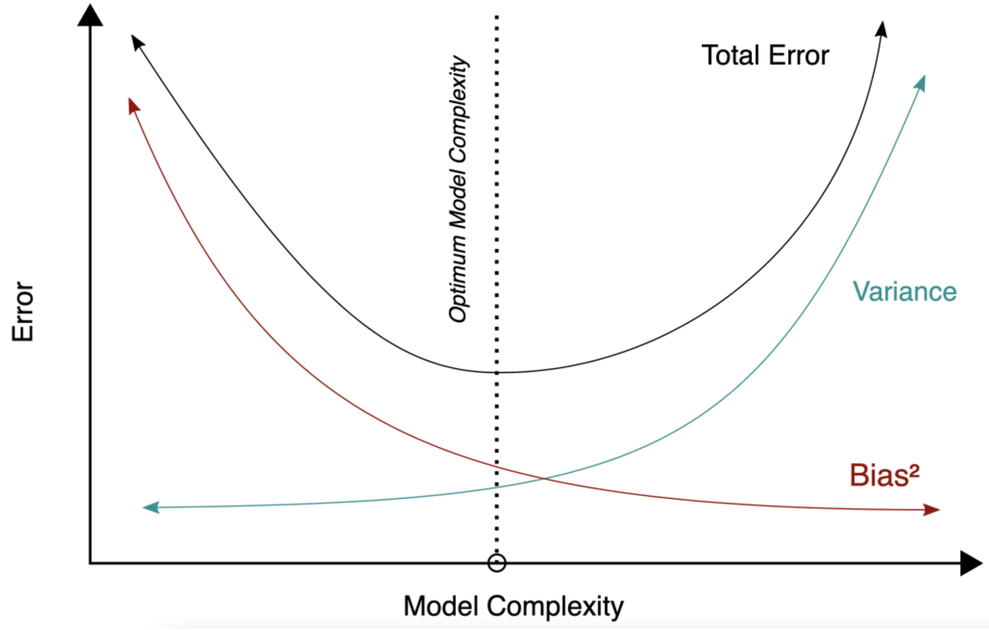
different potential reasons for inherent randomness (noise): complexity, missing information, …

Bias-Variance Tradeoff

- fundamental concept in classical statistical learning theory
- models of higher complexity have lower bias but higher variance (given the same number of training examples)

- generalization error follows U-shaped curve: overfitting once model complexity (number of parameters) passes certain threshold
- overfitting: variance term dominating test error

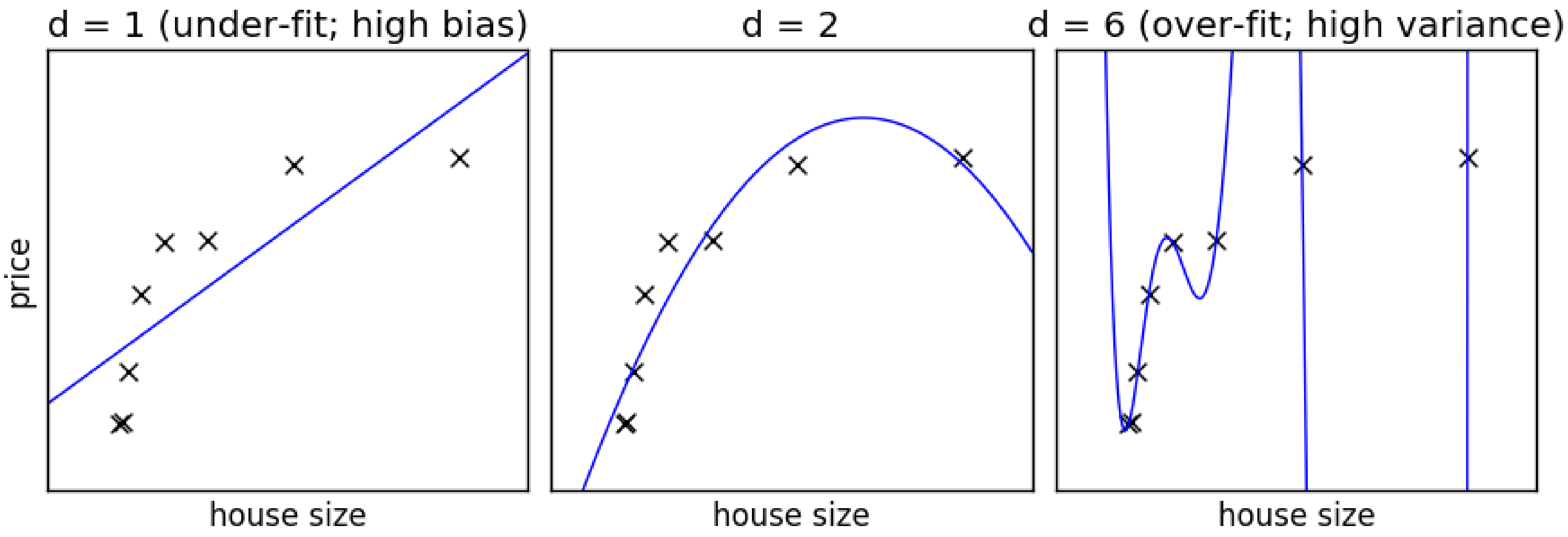
→ increasing model complexity increases test error



from wikipedia

Example: Non-Linear Function Approximation

degree of fitted polynomial



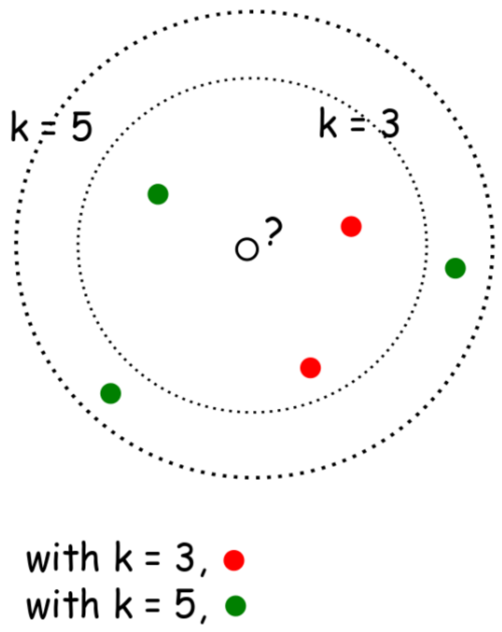
from scikit-learn documentation

Example: k-Nearest Neighbors

- local method, instance-based learning
- non-parametric
- distance defined by metric on $\boldsymbol{x}$ (e.g., Euclidean)

regression:

$$\hat{f}(\boldsymbol{x}_0) = \frac{1}{k}\sum_{j=1}^{k} y_j$$ with j running over k nearest neighbors of $\boldsymbol{x}_0$

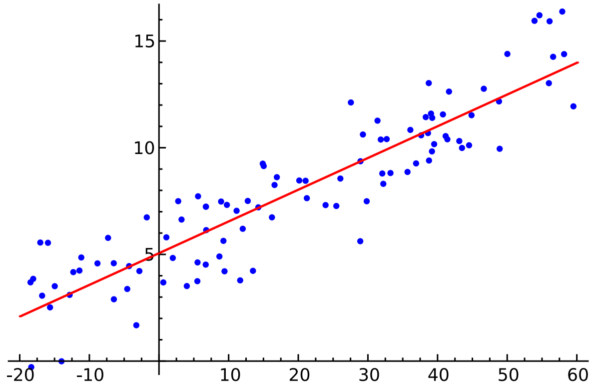


- low k: low bias but high variance
- high k: low variance but high bias

$$bias = f(\boldsymbol{x}) - \frac{1}{k}\sum_{j=1}^{k} y_j$$

$$var = \frac{\sigma^2}{k}$$

Linear Regression



from wikipedia

fit:

$$y_i = \overbrace{\hat{\alpha} + \sum_{j=1}^{p} \hat{\beta}_j \, x_{ij}}^{\hat{f}(\boldsymbol{x}_i)} + \varepsilon_i$$

(model)

error term (noise): reflects assumed data distribution (here: Gaussian with same variance σ^2 for all samples)

parameters to be estimated:

- $\hat{\alpha}, \widehat{\boldsymbol{\beta}}$

→ $\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\boldsymbol{x}_i) \right)^2$

(approximating assumed true $\alpha, \boldsymbol{\beta}$, σ)

predict:

$$\hat{y}_i = E[Y | \boldsymbol{X} = \boldsymbol{x}_i] = \hat{f}(\boldsymbol{x}_i)$$

- conditional mean for squared loss of least squares method
- predict arbitrary quantile by means of quantile loss

$$p(y | \boldsymbol{x}_i) = \mathcal{N}(y; \hat{y}_i, \hat{\sigma}^2)$$

Gaussian

mean

variance (reflected by ε_i in fit)

General Recipe of Statistical Learning

statistical learning algorithm by combining:

- **model** (e.g., linear function, Gaussian distribution)
- **objective function** (e.g., squared residuals)
- **optimization algorithm** (e.g., gradient descent)
- **regularization** (e.g., convolutions)

Loss Function

loss function L: expressing deviation between prediction and target

$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$

with $\widehat{\boldsymbol{\theta}}$ corresponding to parameters of model $\hat{f}(\boldsymbol{x})$

e.g., $\widehat{\alpha}, \widehat{\boldsymbol{\beta}}$ in linear regression

e.g., squared residuals (for regression problems):

$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right) = \left(y_i - \hat{f}\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}\right)\right)^2$

Cost Function

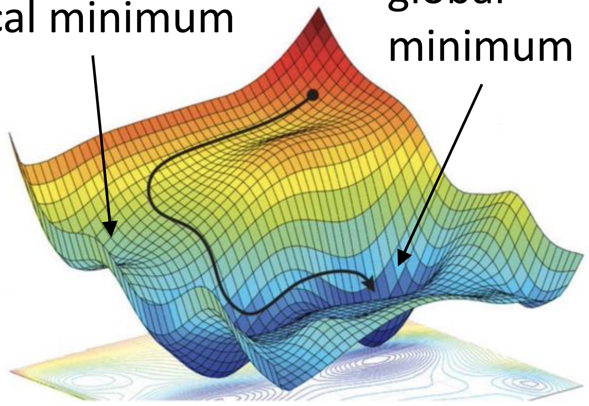
averaging losses over (empirical) training data set:

$$J(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$$

cost function to be minimized according to model parameters $\widehat{\boldsymbol{\theta}}$

→ objective function

Cost Minimization



minimize training costs $J(\widehat{\boldsymbol{\theta}})$ according to model parameters $\widehat{\boldsymbol{\theta}}$:

$$\nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}}) = 0$$

e.g., for mean squared error (aka least squares method):

$$\nabla_{\widehat{\boldsymbol{\theta}}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}) \right)^2 = 0$$

analytical solution for linear regression: ordinary least squares

in general: iterative, numerical optimization (e.g., **gradient descent**)

vector containing all partial derivatives

$$\boxed{\widehat{\boldsymbol{\theta}} \leftarrow \widehat{\boldsymbol{\theta}} - \eta \nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}})}$$

step size (learning rate)

maximum likelihood estimation (minimization of D_{KL} between probability distributions of true data-generating process and model: *make the model distribution match the empirical distribution*):

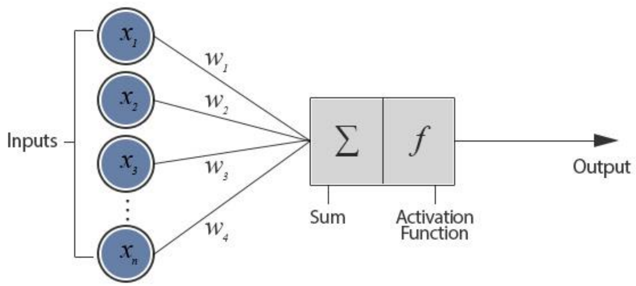
special objective function, estimate mode of assumed model distribution

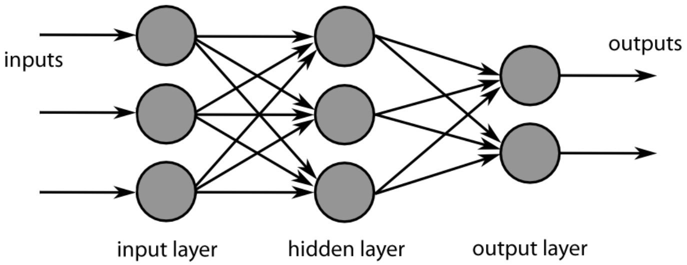
Algorithmic Families

linear (parametric) models

- linear regression
- Generalized Linear Models
- Generalized Additive Models

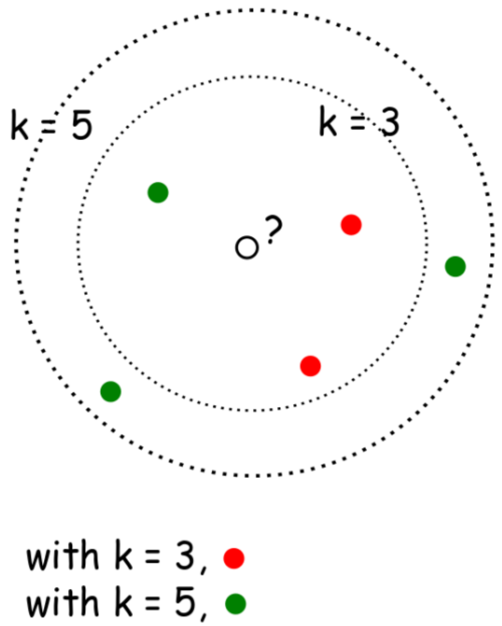
neural networks: non-linear just by means of activation functions





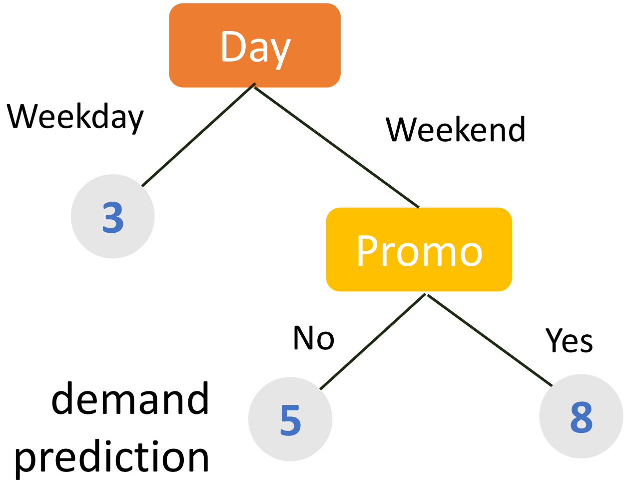
from wikipedia

nearest neighbors (local methods, instance-based learning) – non-parametric models



kernel/support-vector machines: linear model (maximum-margin hyperplane) with kernel trick

decision trees



often used in ensemble methods

- bagging: random forests
- boosting: gradient boosting

Most ML algorithms can be described by the general recipe of combining models, costs, optimization, and regularization methods.

including non-linear models like neural networks (backpropagation), support-vector machines (hinge loss in soft-margin SVM), or decision trees (impurity functions), and even unsupervised learning (e.g., maximum variance axes in PCA)

Most powerful ML algorithms are compound, with rather simple (often linear) building blocks.

To generalize well, one needs to find a method with an appropriate inductive bias for the task at hand (e.g., regularization method like convolutional layers or objective function).

Deep Learning

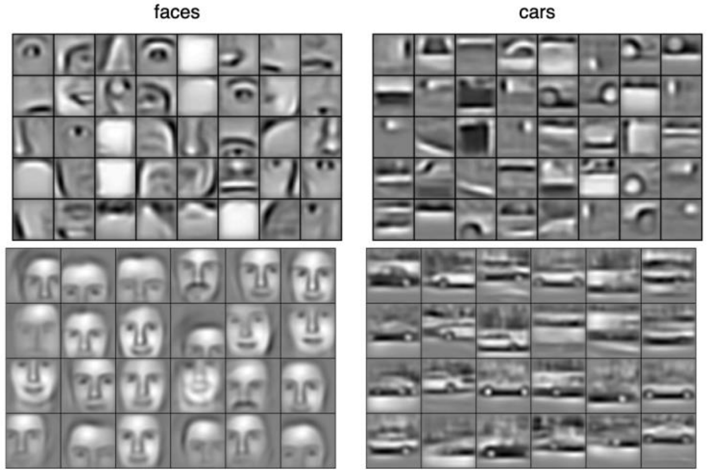
Recap: Goal of ML

generalization from optimization on training data set (approximation of true data generating probability distribution by empirical risk minimization)

- fitting: complex function approximation
- for generalization: learning of good abstraction/representation of data/concepts

→ deep learning methods (neural networks with many layers) optimal candidates

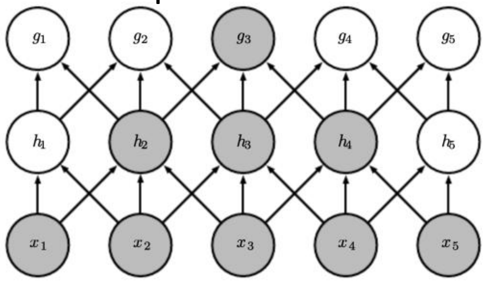
e.g., convolutional neural networks (CNN) can learn hierarchical representation by means of many convolutional and pooling layers



source

the deeper the better:
accuracy, hierarchical representation

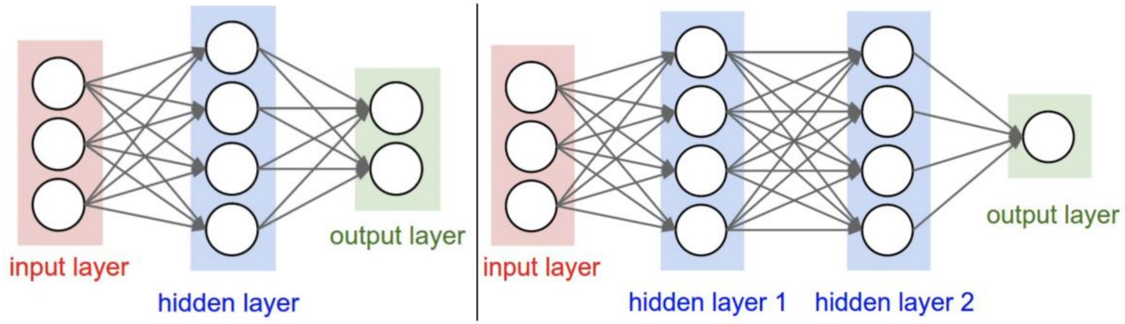
receptive fields:



source

Multi-Layer Perceptron (MLP)

fully-connected feed-forward network with at least one hidden layer



toward deep learning:
add more nodes and hidden layers …

classification:

- logistic regression in hidden nodes
- cross-entropy loss: $L_i\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{w}}\right) = -\sum_{k=1}^{K} y_{ik} \log \hat{f}_k(\boldsymbol{x}_i; \widehat{\boldsymbol{w}})$
- several output nodes k for multi-classification
- softmax output function: $g_k(\boldsymbol{t}_i) = \frac{e^{t_{ik}}}{\sum_{l=1}^{K} e^{t_{il}}}$

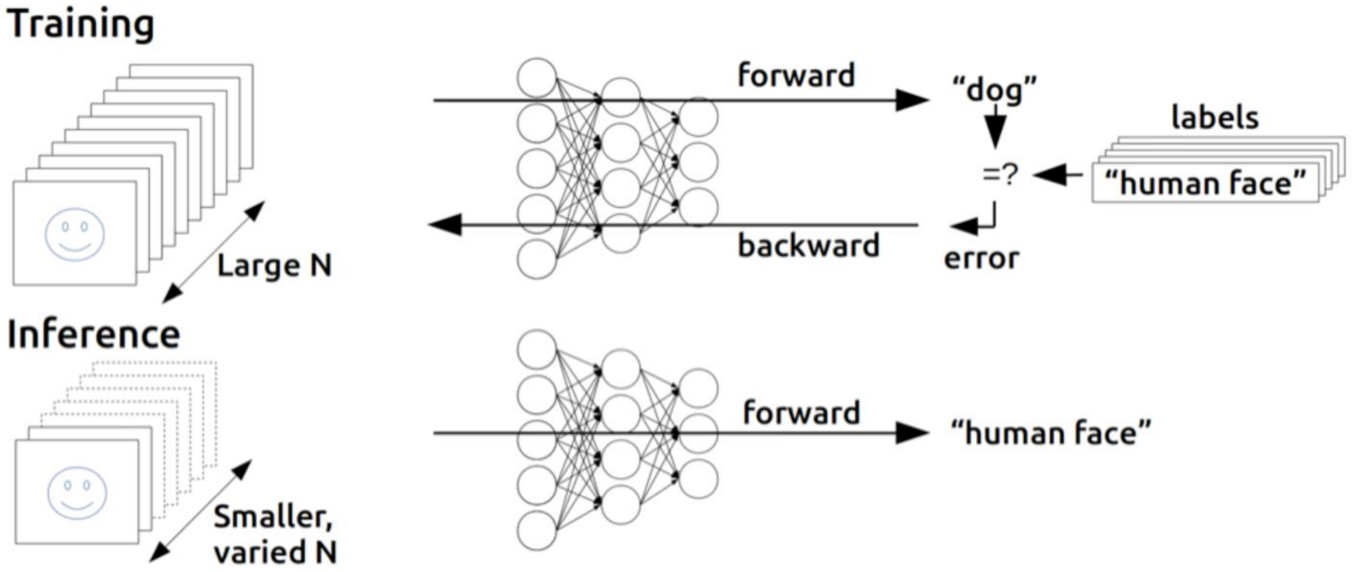
regression:

- squared error loss
- identity output function
- usually just one output node

Learning Mechanism: Back-Propagation

back-propagation of errors (gradients of cost function according to weights) through layers via chain rule of calculus (avoiding redundant calculations of intermediate terms)

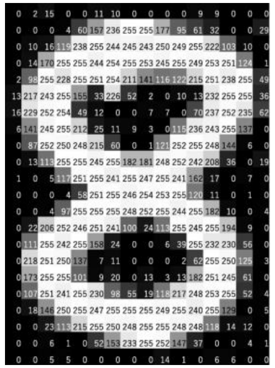
each node exchanges information only with directly connected nodes → enables efficient, parallel computation



source

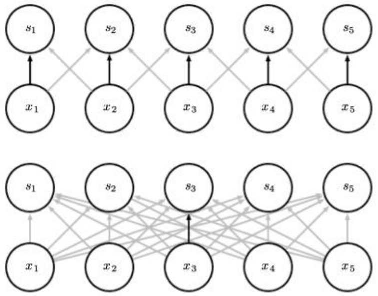
- forward pass: current weights fixed, predictions computed
- backward pass: errors computed from predictions and back-propagated → weights then updated according to loss gradients (via gradient descent)

Convolutional Neural Networks



source

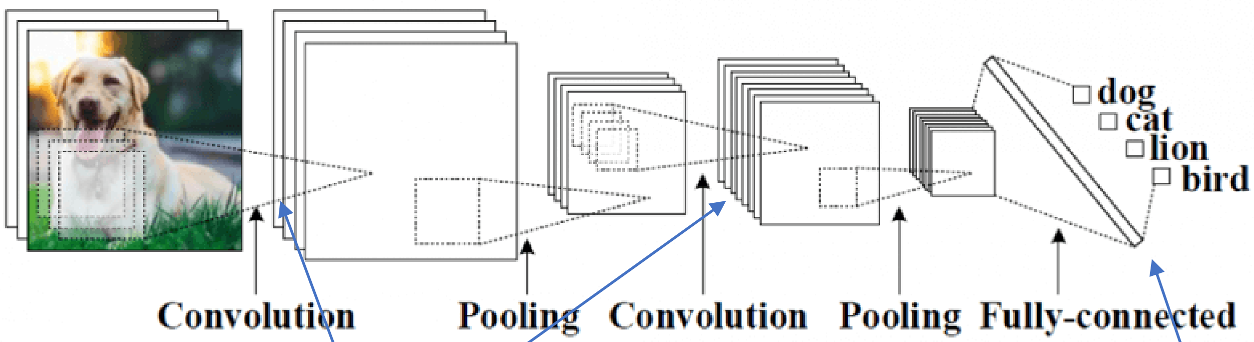
convolutional vs
fully-connected layer:



source

parameter sharing

kernel (learned
parameters)



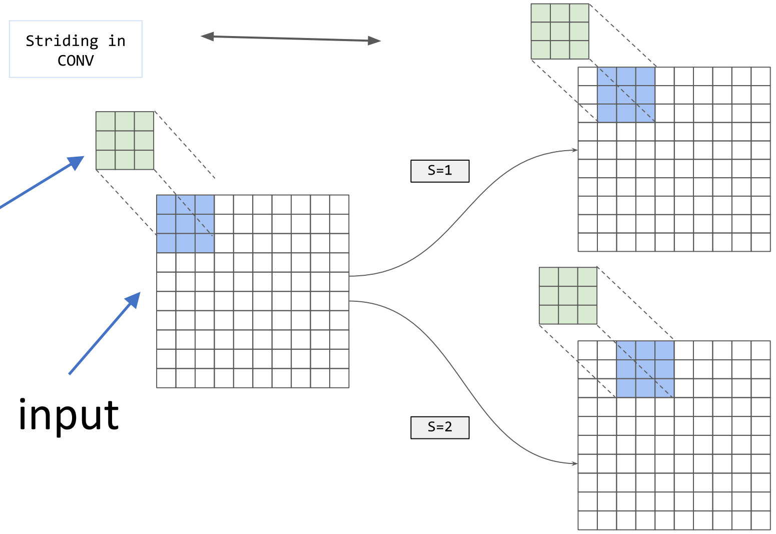
many images
(training examples),
potentially with
several channels

several kernels
producing several
feature maps

down-sampling
by convolutions
and pooling

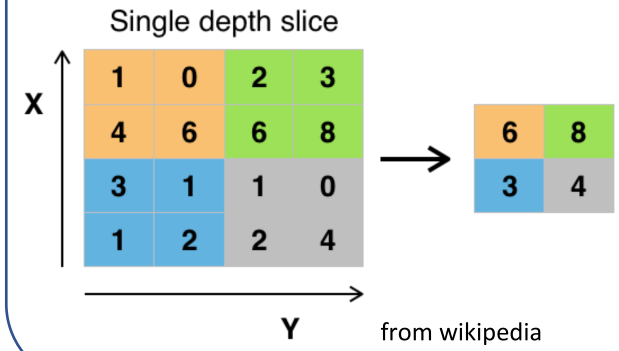
flatten dimensions
for final classification
or regression

convolution:



input

max pooling:



from wikipedia

highly regularized form of feed-forward neural networks

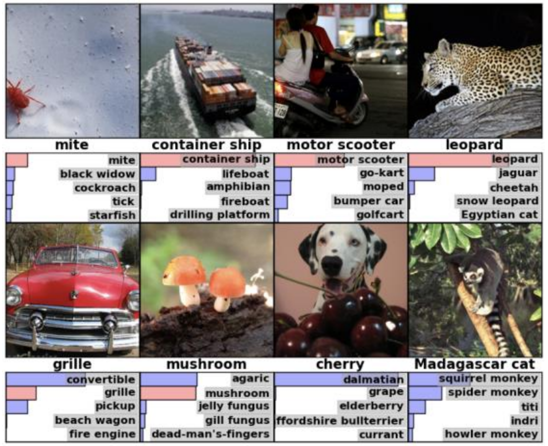
Rise of Deep Learning

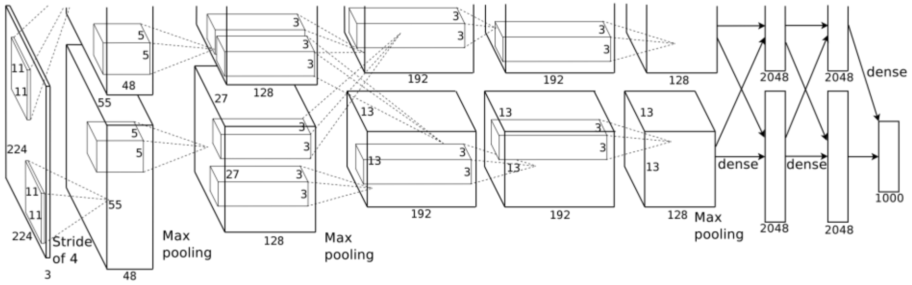
a little bit oversimplified:

deep learning = lots of training data + parallel computation + smart algorithms

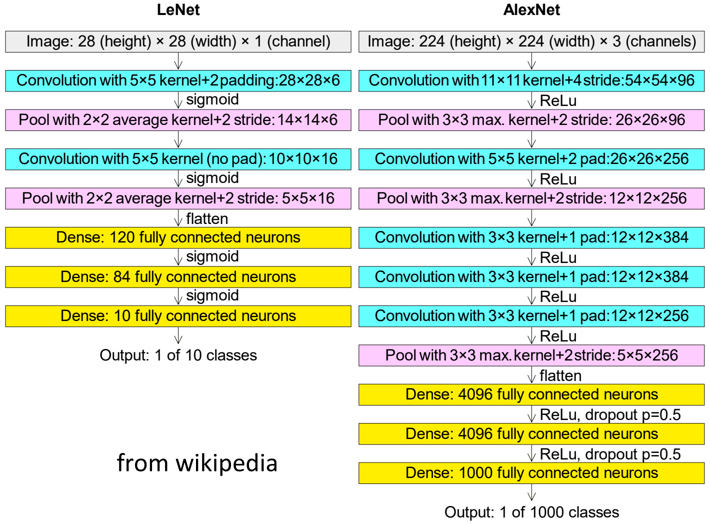
AlexNet: ImageNet + GPUs (allowing more layers) + ReLU, dropout

(pivotal moment for deep learning: ImageNet challenge 2012)





source



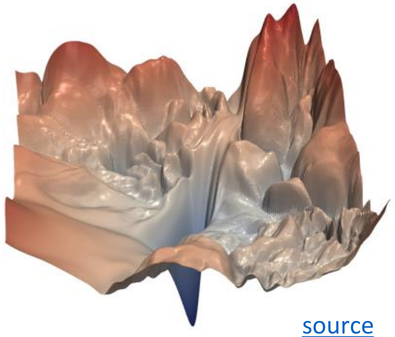
from wikipedia

Training Subtleties of Deep Neural Networks

optimization and regularization difficult
- non-convex optimization problem (e.g., local vs global minima), easily overfitting
- many hyperparameters to tune

many methods to get it working in practice (despite partly patchy theoretical understanding)

typical loss surface:



source

optimization
- activation and loss functions
- weight initialization
- stochastic gradient descent
- adaptive learning rate
- batch normalization

explicit regularization
- weight decay
- dropout
- data augmentation
- weight sharing

implicit regularization
- early stopping
- batch normalization
- stochastic gradient descent

Large Language Models (LLM)

natural language processing: dealing with sequential structures (e.g., text)

examples:

- machine translation (sequence-to-sequence model)
- sentiment classification
- chat bot (conversational AI)

context awareness via **embeddings** and (formerly) recurrent neural networks (**RNN**) or (nowadays) **self-attention**

LLMs: **transformer** models with hundreds of billions of parameters

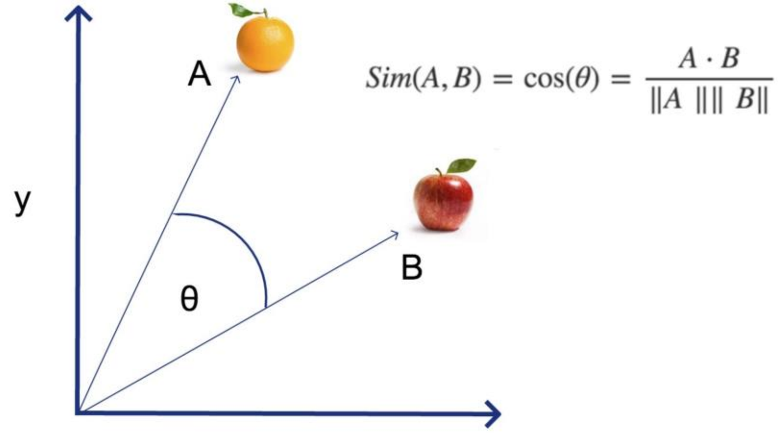
Embeddings

representation of entities by vectors

similarity between embeddings by, e.g., cosine similarity → semantic similarity

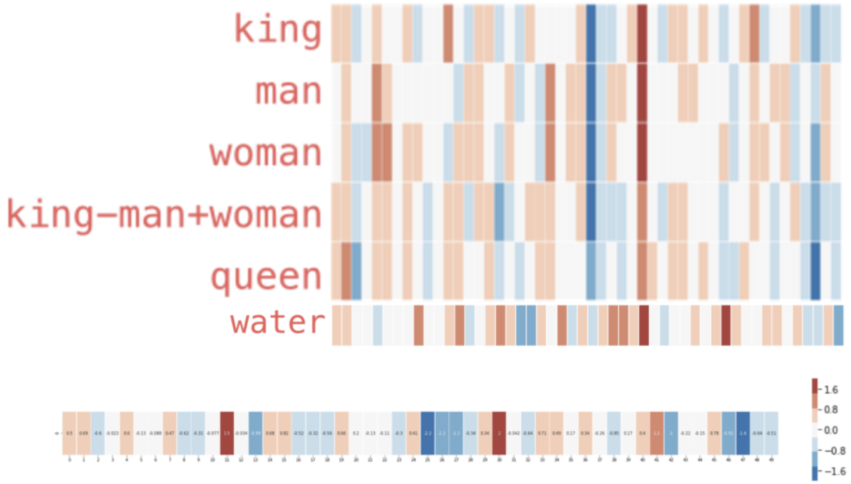
most famous application: word embeddings → associations (natural language processing)

but general concept: embeddings of (categorical) features (e.g., products in recommendation engines)



but also direction of difference vectors interesting (analogies):

`king – man + woman ~= queen`



source

Transformer

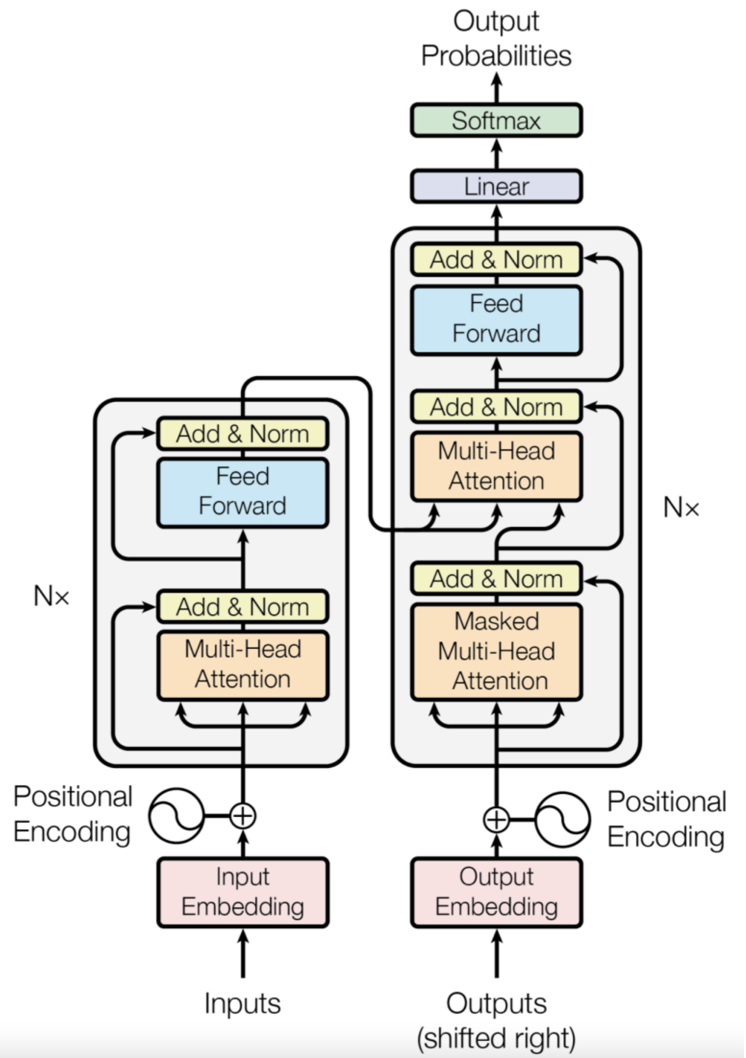
attention is all you need: getting rid of RNNs

replaced by multi-headed self-attention (implemented with matrix multiplications and feed-forward neural networks)

→ allowing for much more parallelization

→ allowing for deeper architecture (more parameters)

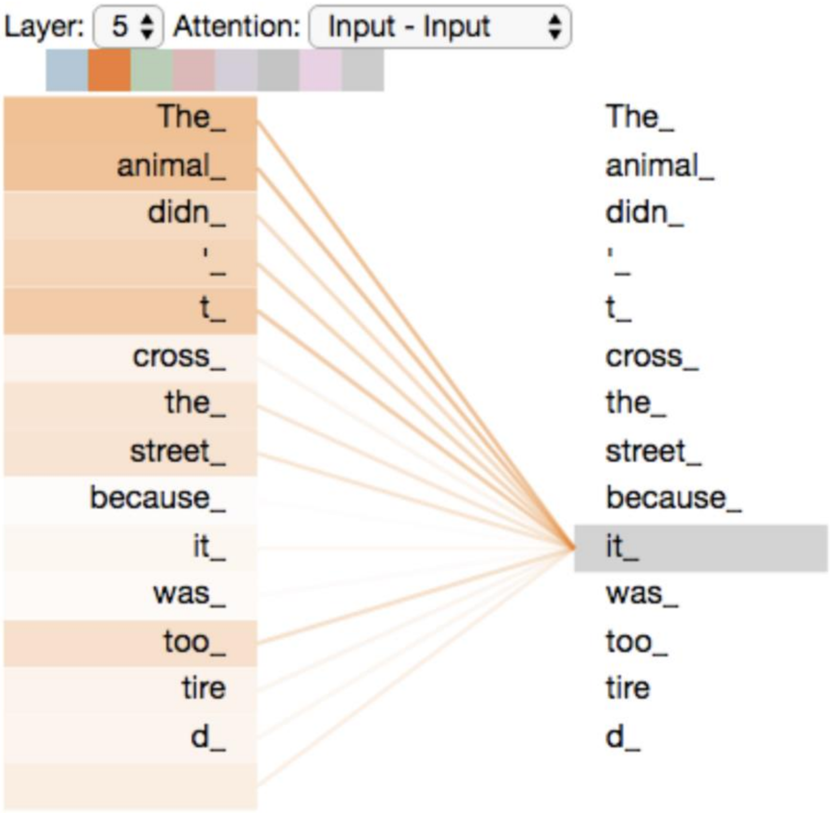
better long-range dependencies thanks to shorter path lengths in network (less sequential operations)



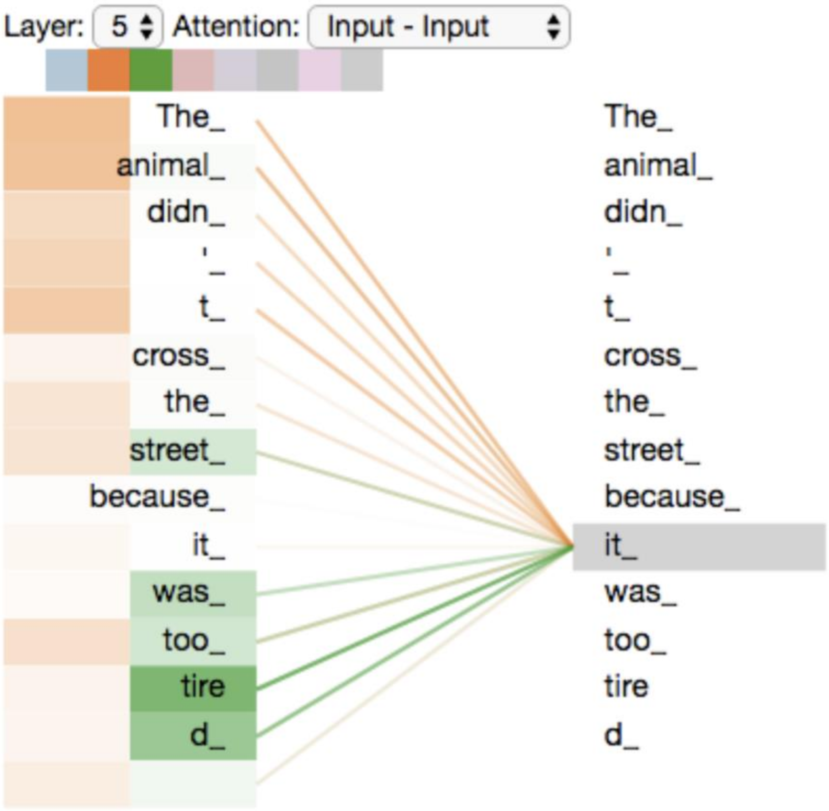
source

Self-Attention

evaluating other input words in terms of relevance for encoding of given word



multi-head attention: several attention layers running in parallel (considering different aspects of input)



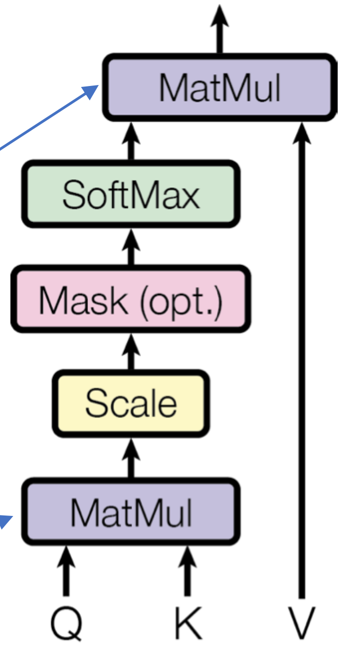
source

Mechanism: Scaled Dot-Products

softmax not scale invariant: largest inputs dominate output for large inputs (more embedding dimensions)

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention



source

3 abstract matrices created from inputs (e.g., word embeddings) by multiplying inputs with 3 different weight matrices

- query Q (embedding dimensions d_k)
- key K (embedding dimensions d_k)
- value V (embedding dimensions d_v)

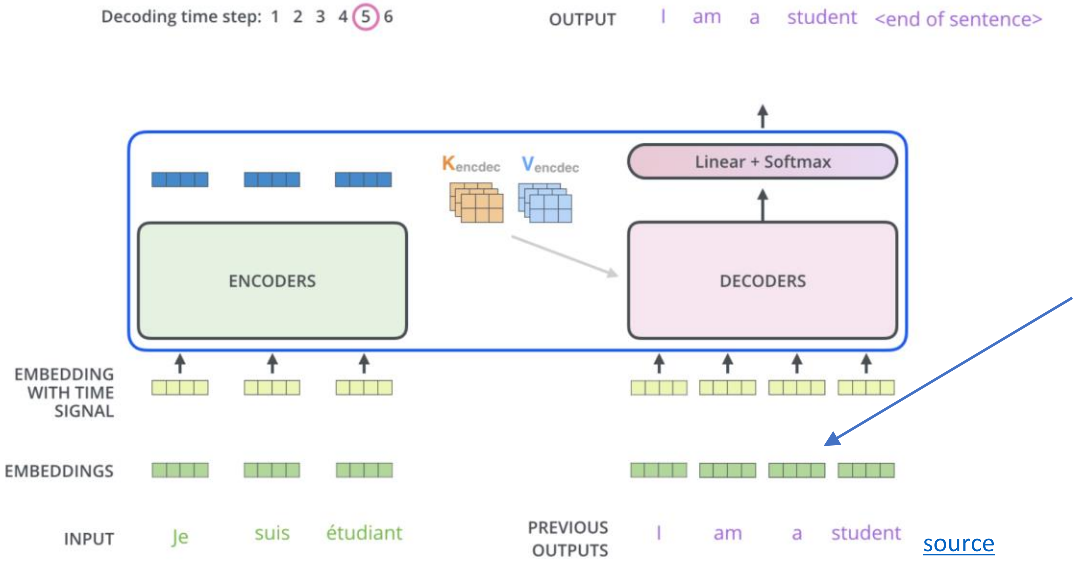
context

keep values of context words to attend to by multiplication of softmax scores with V

scoring each of the key words (context) with respect to current query word

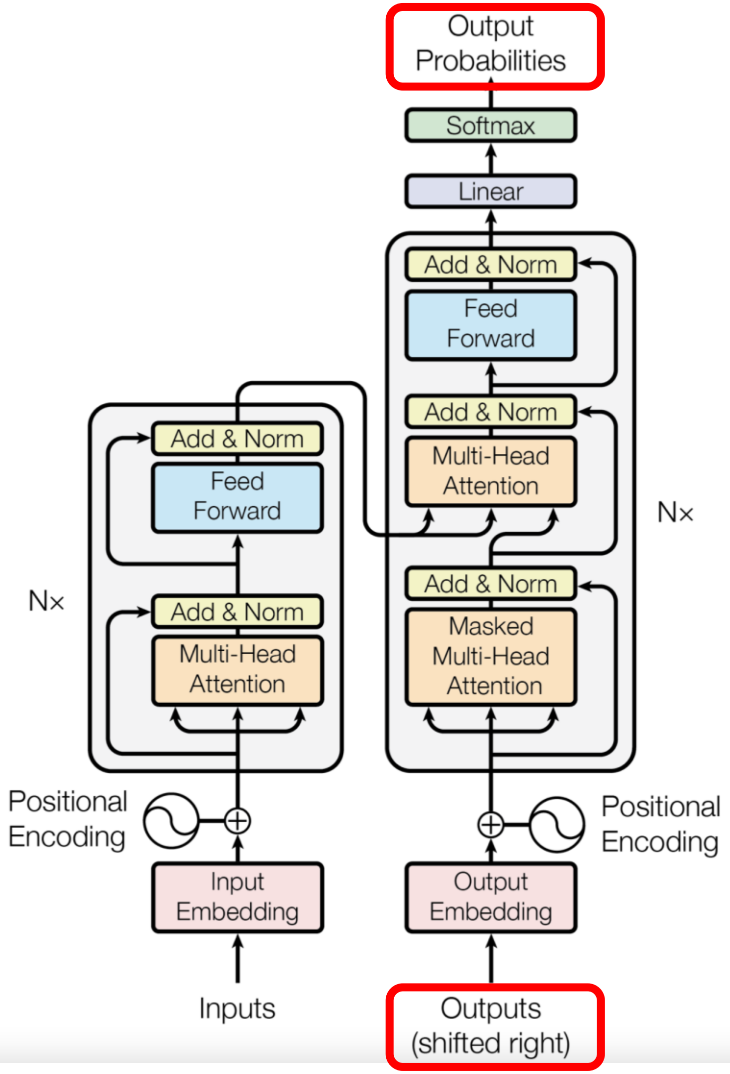
Sequence Completion

for each step/token (iteratively), choose one output token (e.g., greedily picking the one with highest probability or beam search) to add to decoder input sequence → increasing uncertainty



source

auto-regressive



source

prompt: externally given initial sequence for running start and context on which to build rest of sequence (prompt engineering)

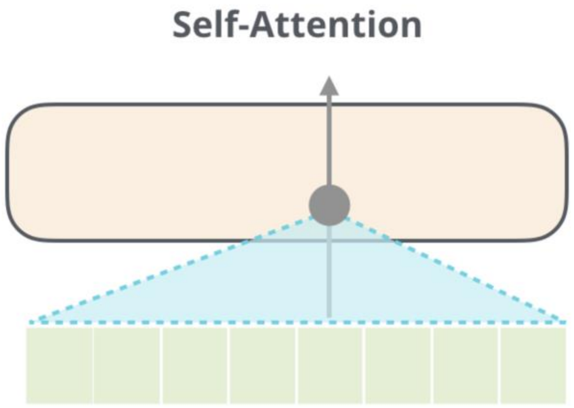
Typical Transformer Architectures for LLMs

encoder-decoder LLMs: sequence-to-sequence, e.g., machine translation

encoder-only LLMs:

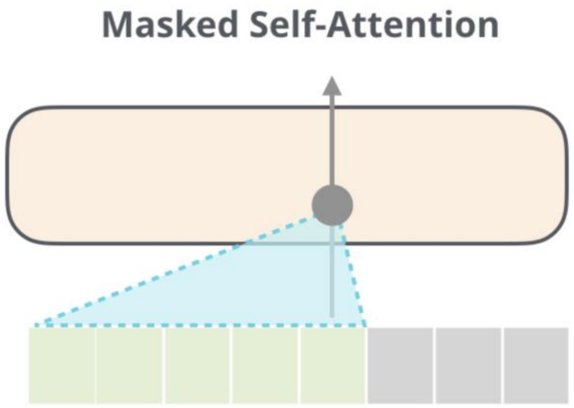
- representation learning (and subsequent fine-tuning)
- training: prediction of masked words
- incorporate context of both sides of token

example: Google's BERT (Bidirectional Encoder Representations from Transformers)



decoder-only LLMs:

- text generation (potentially in-context only), e.g., chat bot
- training: next-word prediction
- output one token at a time (auto-regressive)



source

example: OpenAI's GPT (Generative Pre-trained Transformer)

GPT-4 capabilities

Multi-Task Learning of LLMs

compositional nature of deep learning allows transfer learning in a semi-supervised way (also prominent for CNNs in computer vision):

- self-supervised **pre-training** (e.g., next-word prediction) on massive data sets (foundation models like GPT or BERT)
- subsequent supervised **fine-tuning** on specific tasks and (usually much smaller) data sets (by adapting parameters or/and adding layers)

in-context learning as alternative to fine-tuning: only using information fed into LLM via input prompt, no parameter updates (typically decoder-only)

typical prompt: instructions, context (potentially retrieved externally from, e.g., knowledge-base embeddings), query, output indicator

query with (few-shot) or without (zero-shot) providing explicit examples

prompt engineering

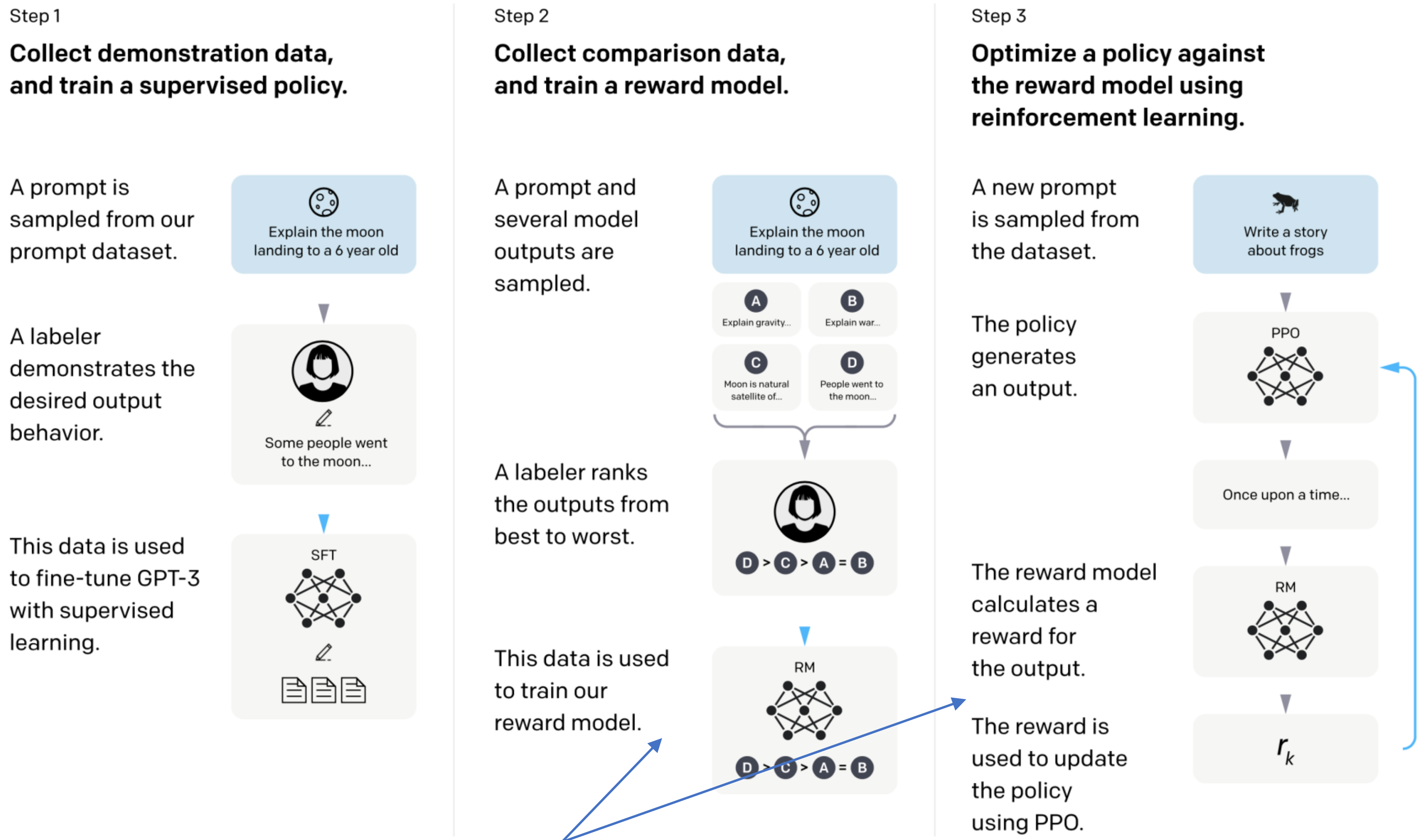
Conversational AI: RL from Human Feedback

example for supporting large language models (transformers) with RL

used in famous ChatGPT

goal: improve alignment with user intentions

→ learn from human preferences



source

RL looks at reward of text output passages as a whole (rather than token-level loss in supervised learning)

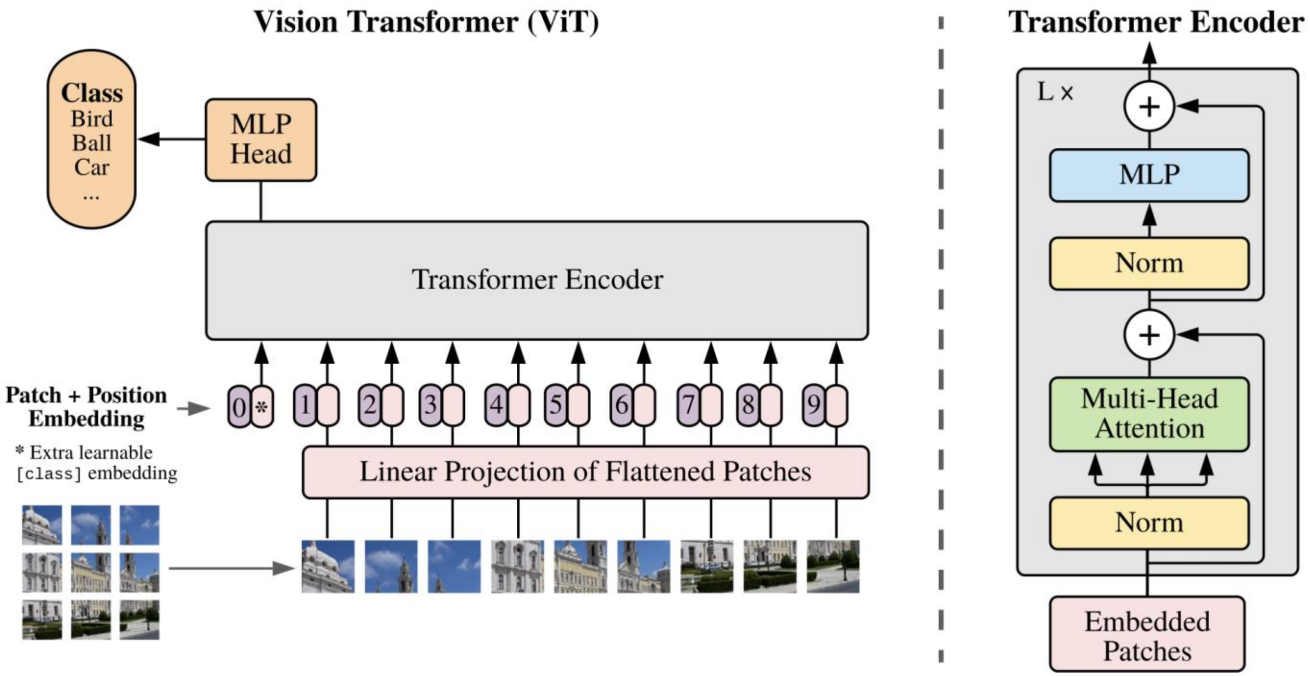
Transformer for Vision

formulation as sequential problem:

- split image into patches and flatten → use as tokens
- produce linear embeddings and add positional embeddings

processing by transformer encoder:

- pre-train with image labels
- fine-tune on specific data set

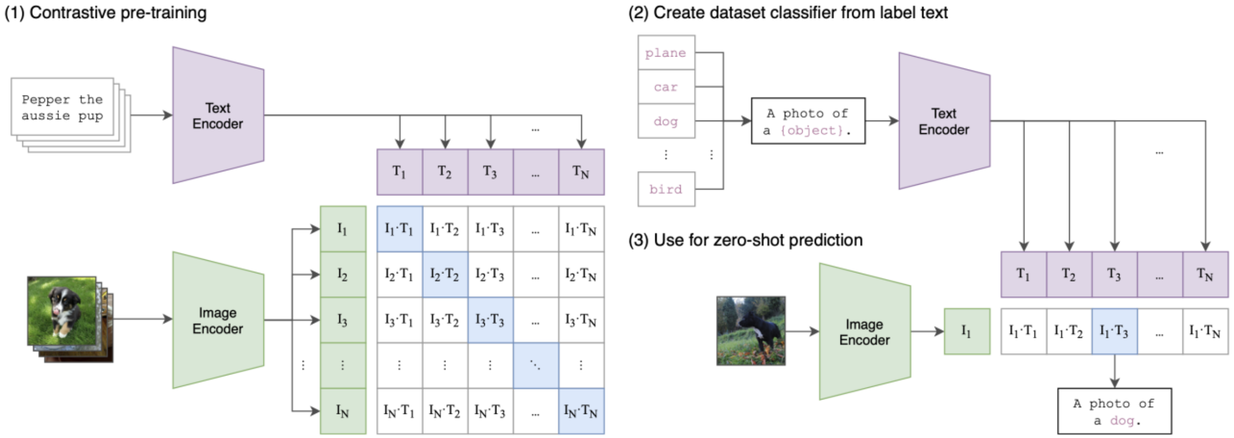


source

Combination of Vision and Text: Multi-Modality

example: CLIP (Contrastive Language-Image Pre-training)

- learn image representations by predicting which caption goes with which image (pre-training)
- zero-shot transfer (e.g., for object recognition)



multi-modal perception as input for large language models: KOSMOS-1

multi-purpose (multi-modal and multi-task) models as next generalization step of ML (e.g., Google's Pathways)

transformers good candidate (universal and flexible architecture, little task-specific inductive bias)

Generative Models

Generative vs Discriminative Models

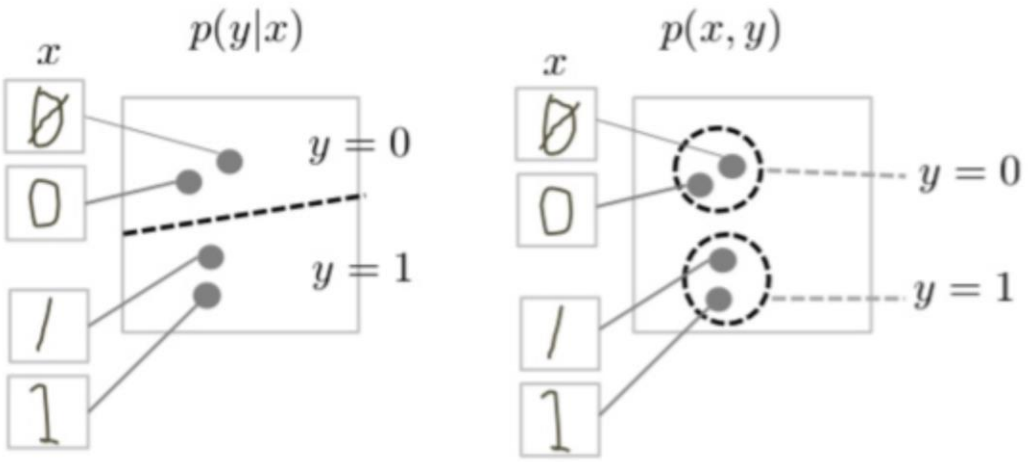
generative models: predict joint probability $P(Y, \boldsymbol{X})$ (what allows to create new data samples) or directly generates new data samples

discriminative models: predict conditional probability $P(Y|\boldsymbol{X})$ or directly output (label for classification, real value for regression)

task of generative models more difficult: model full data distribution rather than merely find patterns in inputs to distinguish outputs

discriminative model

generative model



source

Data Generation

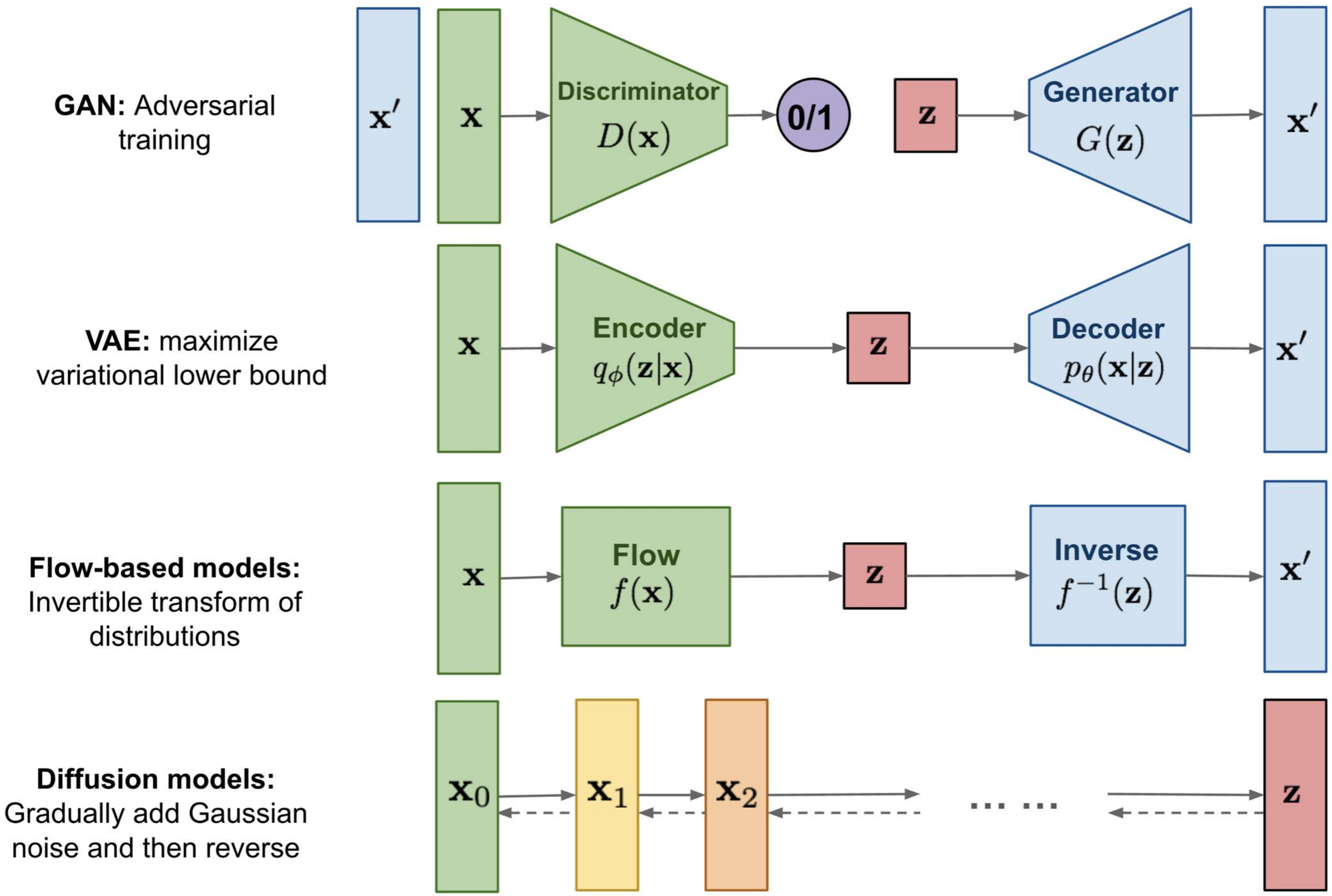
generative models can be used for discriminative tasks (although potentially inferior to direct discriminative methods)

but generative methods do more than discriminative ones: model full data distribution

→ allows generation of new data samples (can be images, text, [video](), [audio](), code like SQL or Python, proteins, materials, time series, structured data, …)

large (auto-regressive) language models examples of generative models

Different Types of Generative Models



source

Conditioning

tradeoff between diversity (unconditioned) and fidelity (guidance)

as discussed so far, generative methods give no control over what kind of data is generated (limited usability)

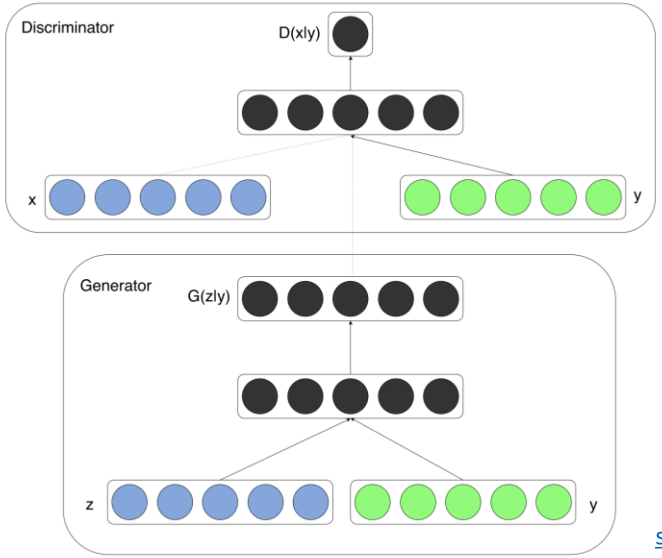
→ need for conditional approach (e.g., conditioning on describing text, for example by means of CLIP)

source

guided diffusion: “Pembroke Welsh corgi”

example GANs:

extend usual GAN in conditional model by feeding extra information y (e.g., class labels) as additional input layer into both generator and discriminator



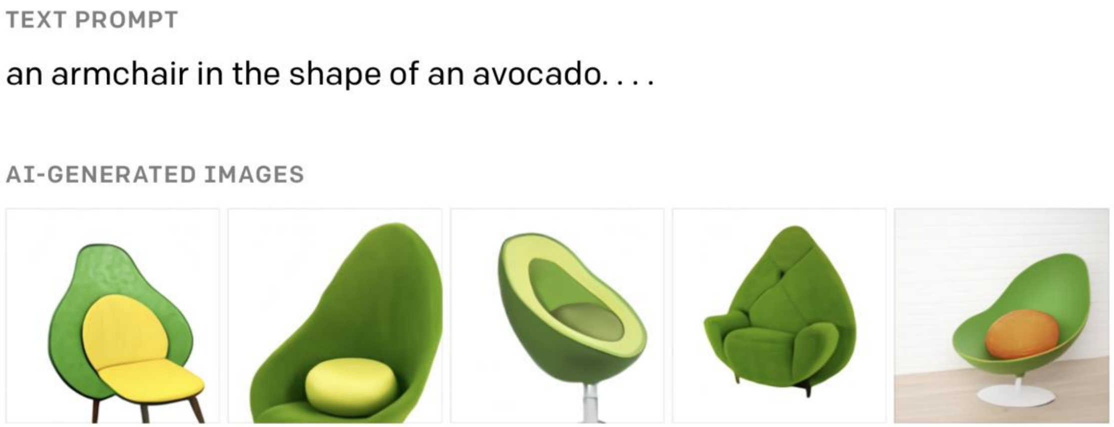
source

$$L(\boldsymbol{x}_i) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\ln D(\boldsymbol{x}_i|y_i)] + E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\ln\left(1 - D(\boldsymbol{x}_i|y_i)\right)\right]$$

Multi-Modal Generative Models

example: generate images from text descriptions

DALL-E (blend of WALL-E and Salvador Dalí): decoder-only transformer auto-regressively modeling text and image tokens as single data stream

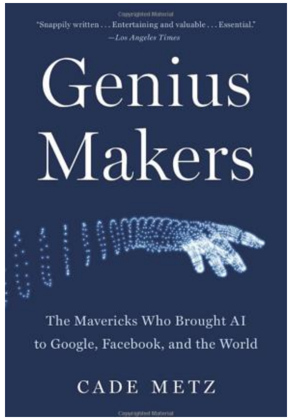


source

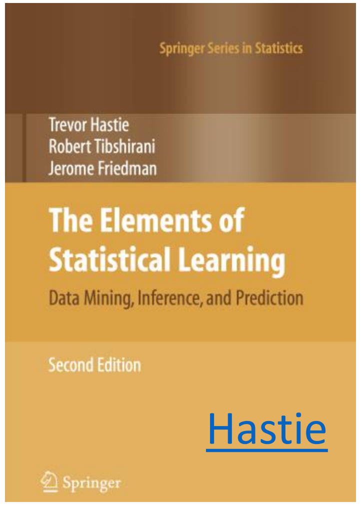
DALL-E 2: image generation conditioned on CLIP image embedding

Literature

historical overview of deep learning:



foundations of ML:

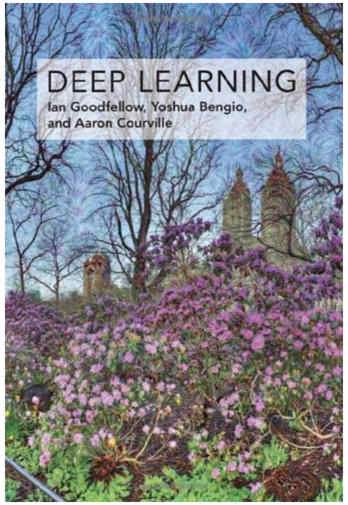


Hastie

a few seminal papers:

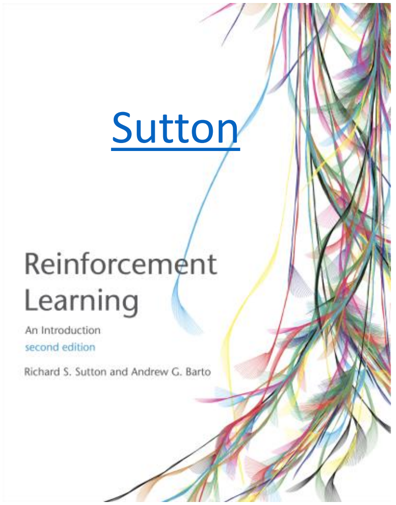
- back-propagation: one of the founding moments of deep learning
- CNN: neural networks work
- AlexNet: deep learning takes over
- transformer: SOTA

foundations of deep learning:
https://www.deeplearningbook.org/

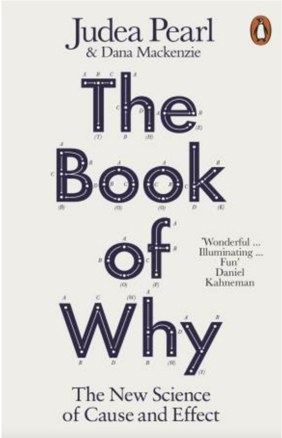


overview of reinforcement learning:

Sutton



gentle but geniune introduction to causality:



Some Philosophical Thoughts

computational theory of mind: mind from matter

just scaling up current methods (e.g., LLMs) enough to achieve general intelligence? or additional methods needed?

agency via goal-based approaches? (is reward enough?)

emergent capabilities of complex systems almost impossible to foresee: emotions or consciousness occur as emergent capabilities?