

# Machine Learning – Overview

# Main Areas of Artificial Intelligence

- **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)



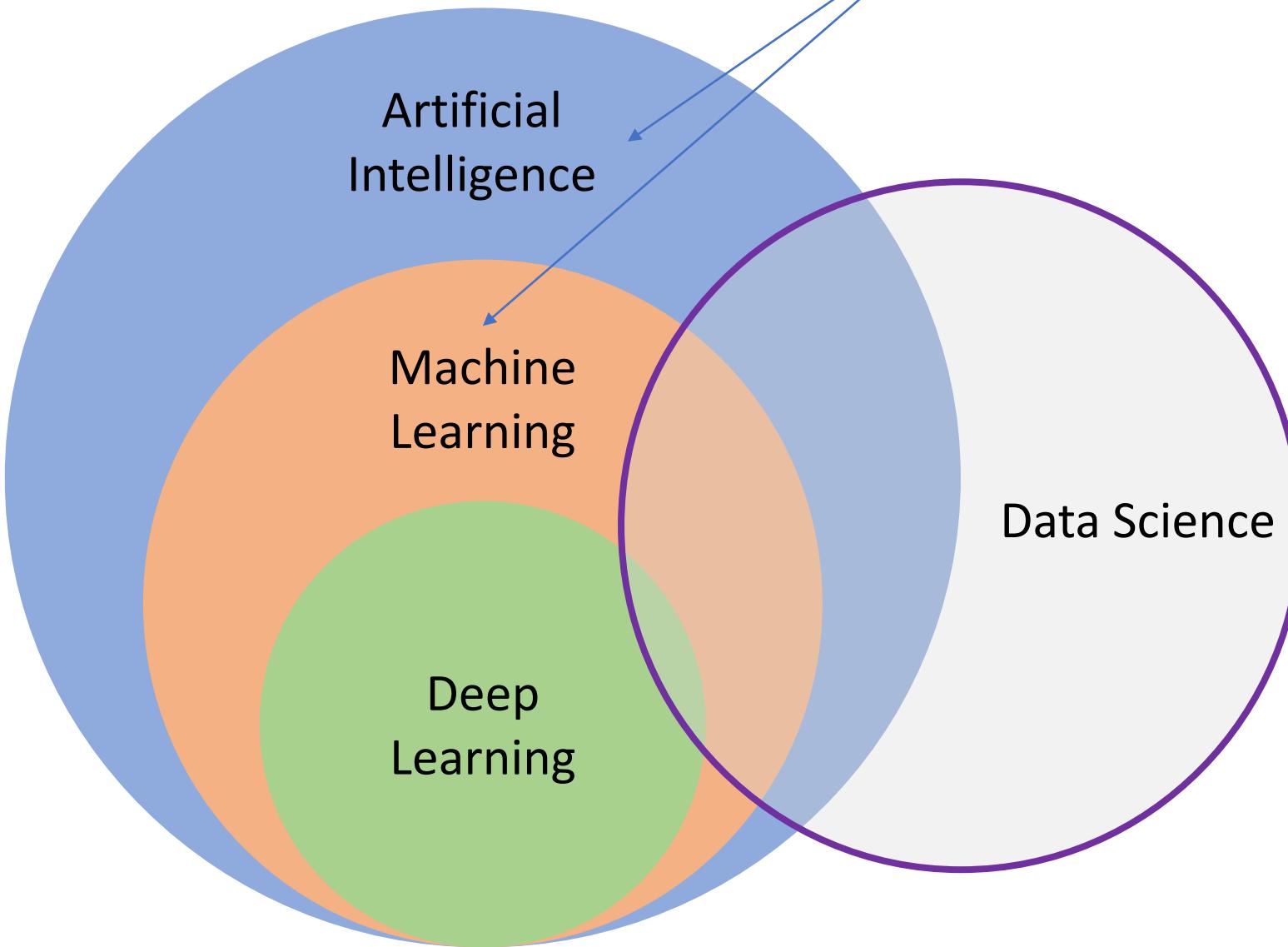
from wikipedia

All of these are enabled by one key ingredient:

agency:  
perception – thought – action

- *learning from experience (Machine Learning)*
- also: knowledge representation, automated reasoning (first indices in modern large language models)

# Buzz Words ...



blend of diverse components from different domains  
(statistics, optimization, computer science, ...)

*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

traditional algorithms:

explicit (handcrafted) instructions for each situation



from wikipedia

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 (compare with human reasoning by analogies)

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.

# 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)

# Most Famous Applications

recommendations



chatbots



ChatGPT



autonomous driving



translation



assistants (speech recognition)



robotics



OCR



and many more ...

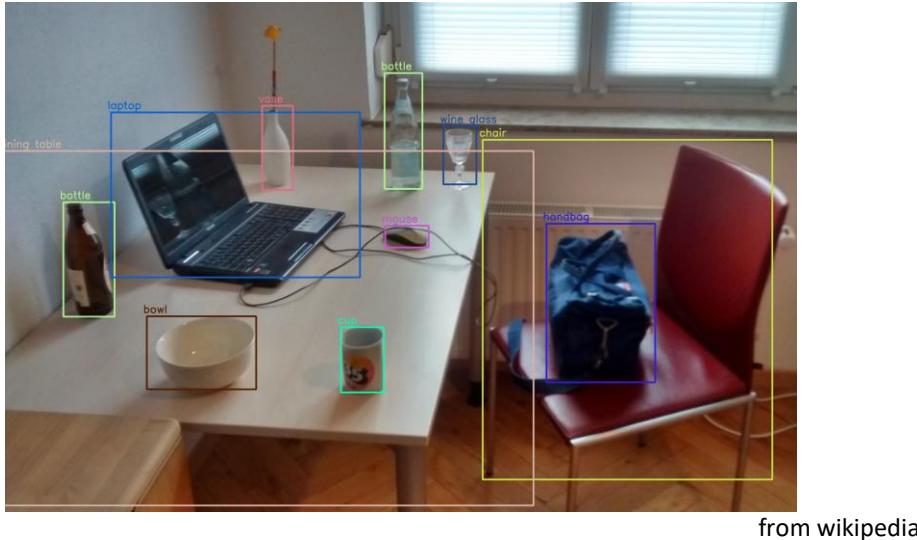
# When to Use ML (= Learning from Data)

## automation

too complex for rules

## complexity / uncertainty

too complex for humans



examples: object recognition, all applications from previous slide

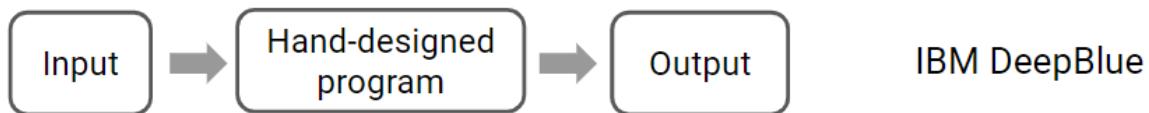
more scientific use cases: medicine (imaging, diagnosis, drug design), particle physics (analysis of collider experiments), material science (material properties and design of new materials), ...



examples: protein structure predictions (AlphaFold), demand forecasting

# Ladder of Generalization

## Rule-based systems



IBM DeepBlue

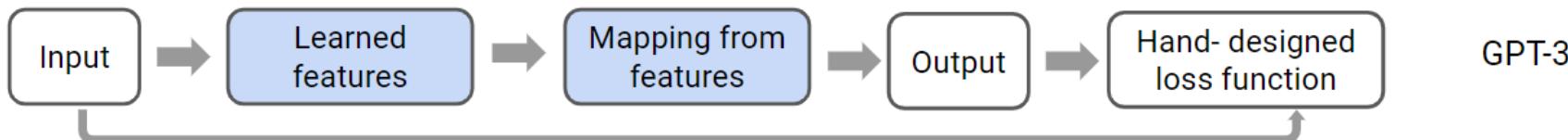
Learnable part of the system

## Classical machine learning



SVM

## Deep learning: (self-)supervised learning



GPT-3

## Deep learning: other RL formulations



???

e.g., ChatGPT (RLHF)

[source](#)

# Generative AI

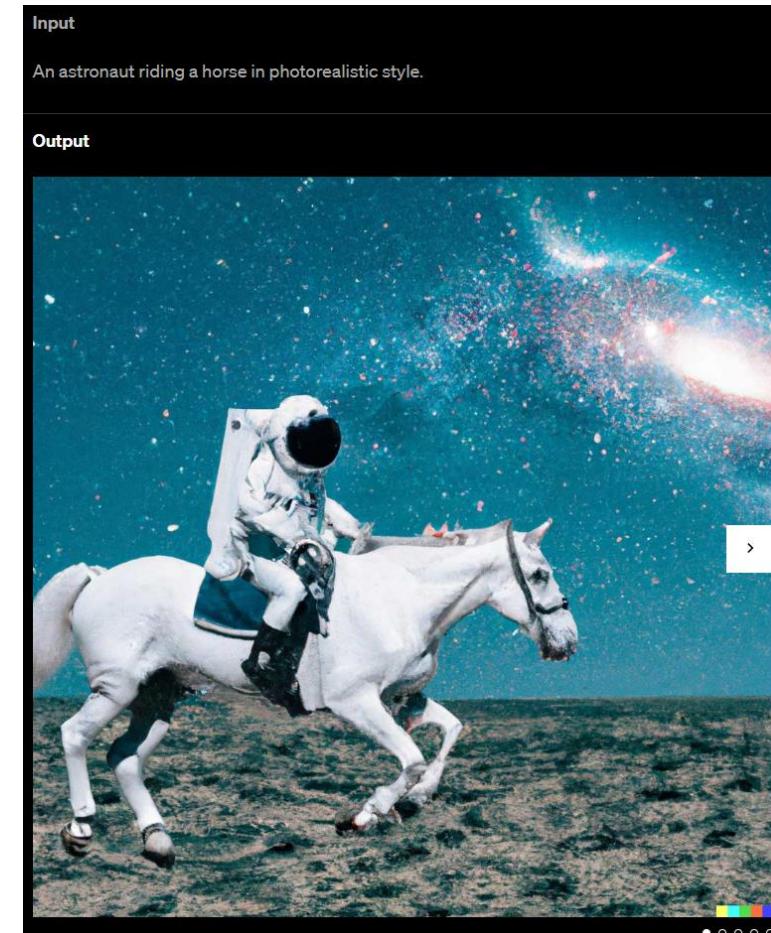
more recently: generative applications

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

Depending on the application, there are currently two dominant approaches:

- text generation: large language models (transformer)
- image synthesis: diffusion models (usually conditioned on text by transformers)

example: DALL-E 2



up next: video synthesis → dynamics/physics understanding/simulation

# Learning Paradigms

# Supervised Learning

**learning by teacher** → usually rather narrow tasks (passive approach)

## Target Quantity

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

## Features

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



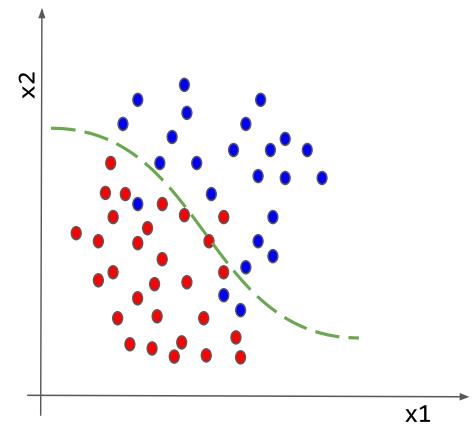
## Example: Spam Filtering

*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  $x_1$  and  $x_2$**   
**spam, no spam**



# Reinforcement Learning

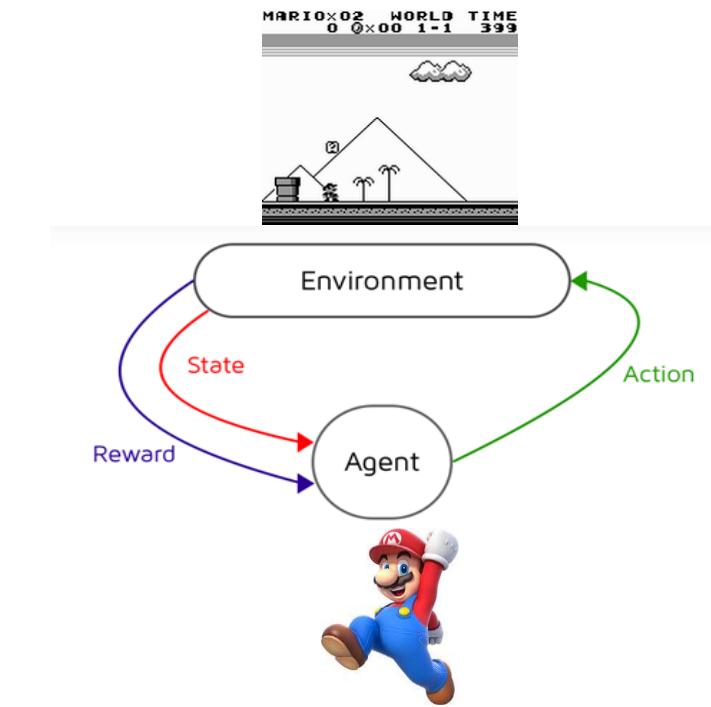
**learning by trial-and-error** (exploration and exploitation)

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

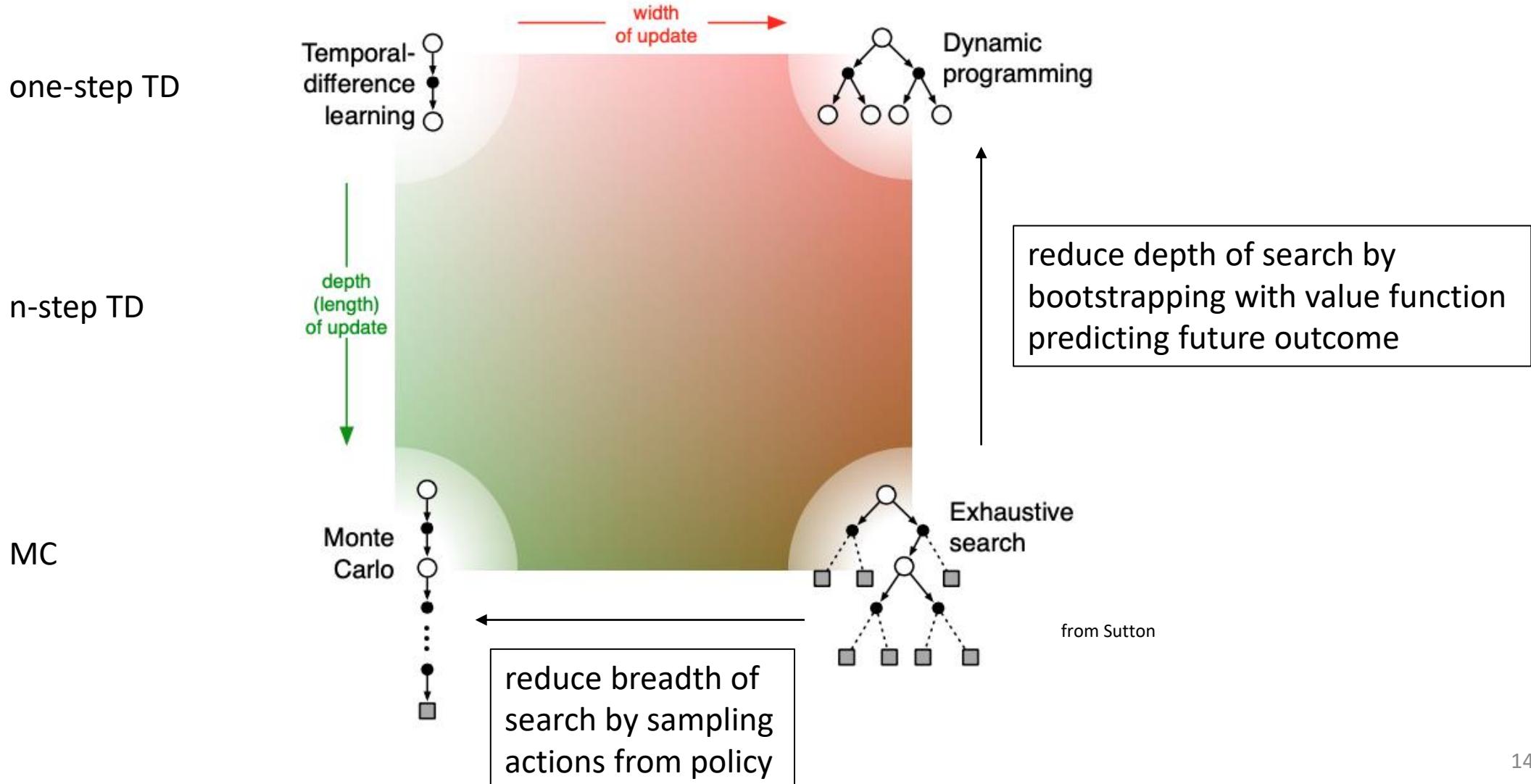
corresponds to search for best action policy to reach a given goal  
(e.g., win a game)

using learning from examples (data) to guide the search

RL setup usually more difficult (e.g., non-differentiable as a whole) than supervised learning one (which can be seen as “generalized optimization”, often of proxy metric)  
but RL can be cast as supervised-learning setup: express rewards by more intricate loss function



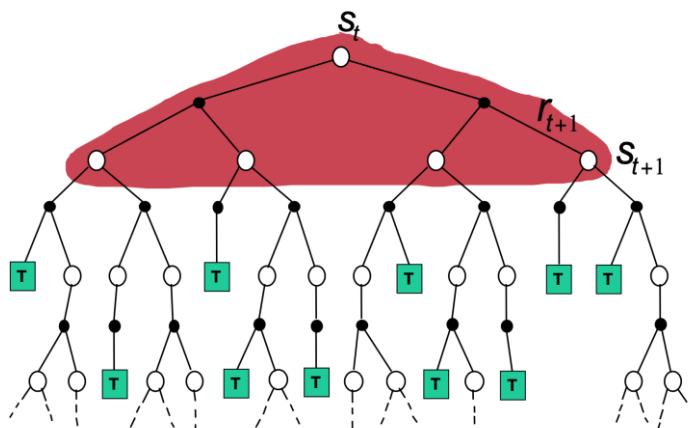
# Reduction of Search Space



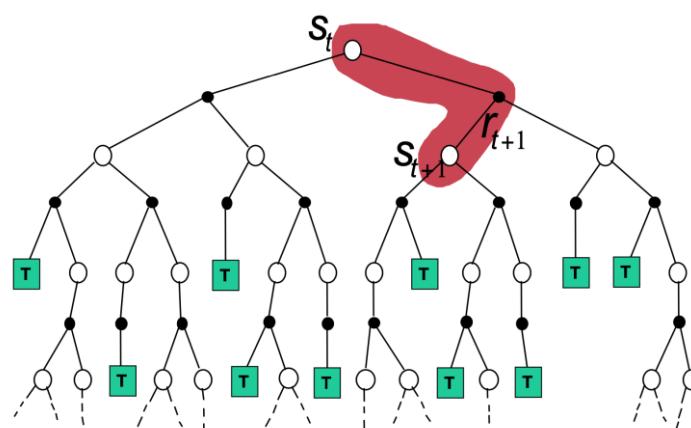
# Main Concepts of Value-Based RL Methods

**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)

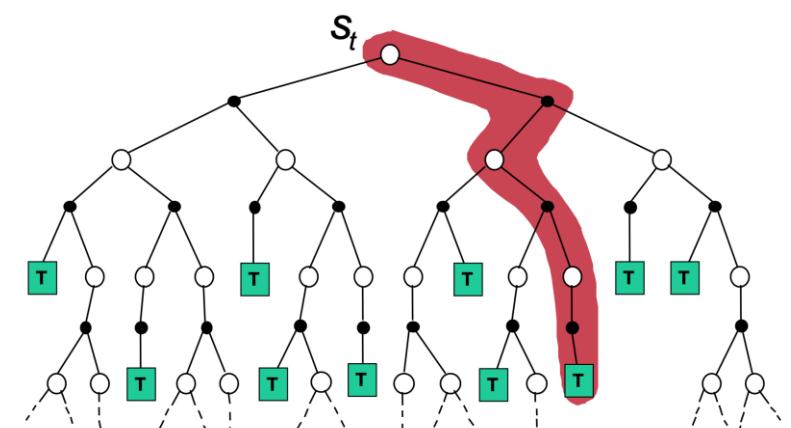
Dynamic Programming



Temporal Difference (TD) Learning



Monte Carlo (MC)



from Sutton

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

- bootstrapping
- sampling → model-free

- no bootstrapping
- sampling → model-free

# Unsupervised Learning

## learning by observation

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

can be cast as **self-supervised learning**:

- input-output mapping like supervised learning
- but generating labels itself from input information

generative AI as unsupervised learning: generate variations of training data

## A look at unsupervised learning

### ■ “Pure” Reinforcement Learning (cherry)

- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**



### ■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

### ■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**

■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

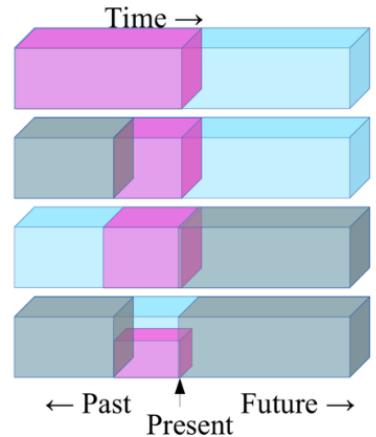
Original LeCun cake analogy slide presented at NIPS 2016, the highlighted area has now been updated.

[source](#)

Y. LeCun

## Self-Supervised Learning

- ▶ Predict any part of the input from any other part.
- ▶ Predict the **future from the past**.
- ▶ Predict the **future from the recent past**.
- ▶ Predict the **past from the present**.
- ▶ Predict the **top from the bottom**.
- ▶ Predict the **occluded from the visible**
- ▶ **Pretend there is a part of the input you don't know and predict that.**



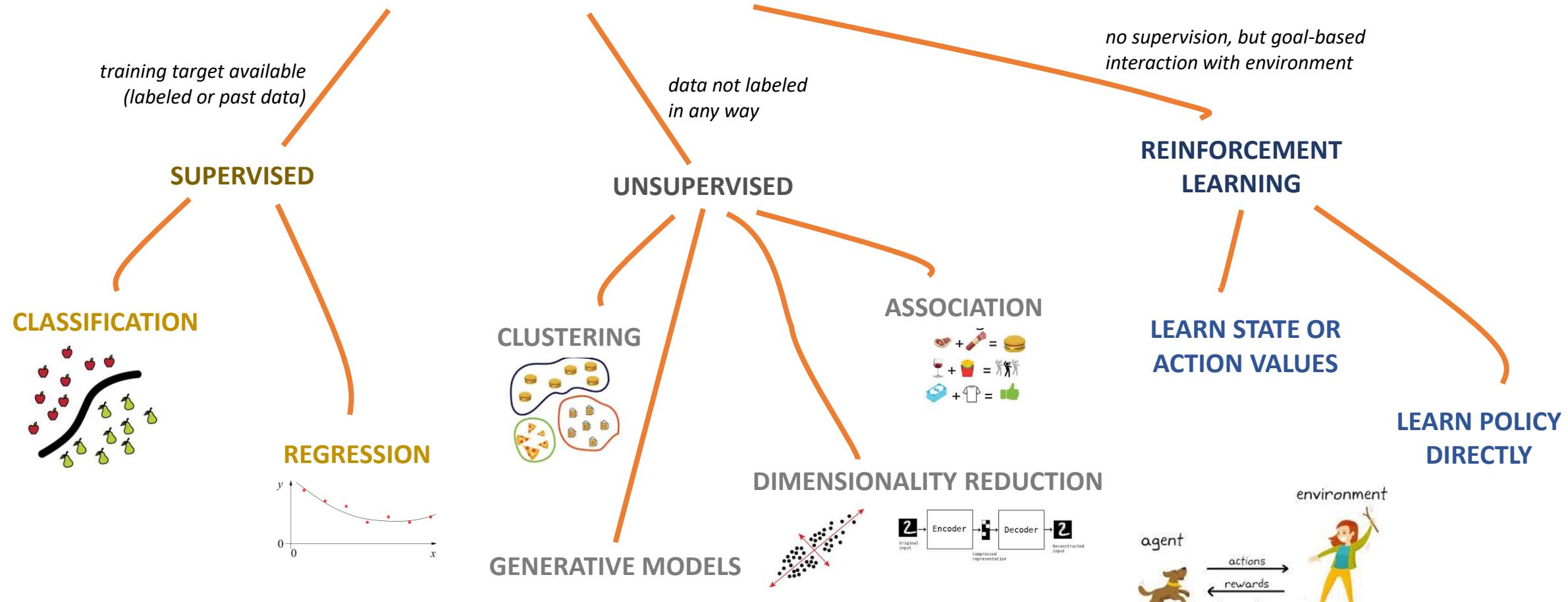
© 2019 IEEE International Solid-State Circuits Conference

1.1: Deep Learning Hardware: Past, Present, & Future

LeCun's self-supervised learning slide at ISSCC 2019

58

# MACHINE LEARNING



learning by teacher  
(high-dimensional curve fitting)

learning by observation  
(pattern recognition)

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

unsupervised and reinforcement learning can  
both be cast as supervised-learning setup

# Fitting and Generalization



... ML ...



# Supervised Learning Scenario

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

random variables  $Y$  and  $\mathbf{X} = (X_1, X_2, \dots, X_p)$  ← usually many dimensions

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

**generalization** as 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: many features (dimensions) → lots of data needed to densely sample volume

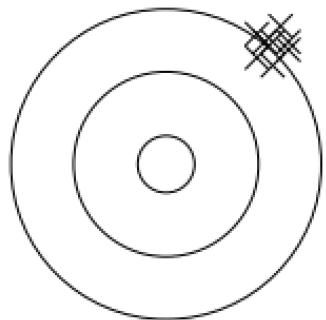
but reality is friendly: most high-dimensional data sets reside on lower-dimensional manifolds (manifold hypothesis) → enabling effectiveness of ML

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”)

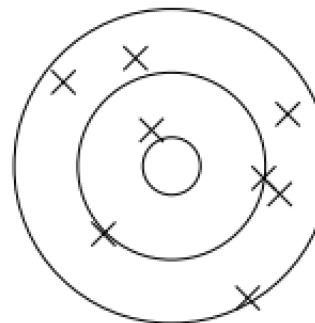
inductive bias comes in many different forms: model design (e.g., linear response), regularization (e.g., convolutions), optimization algorithms, ...

# 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):

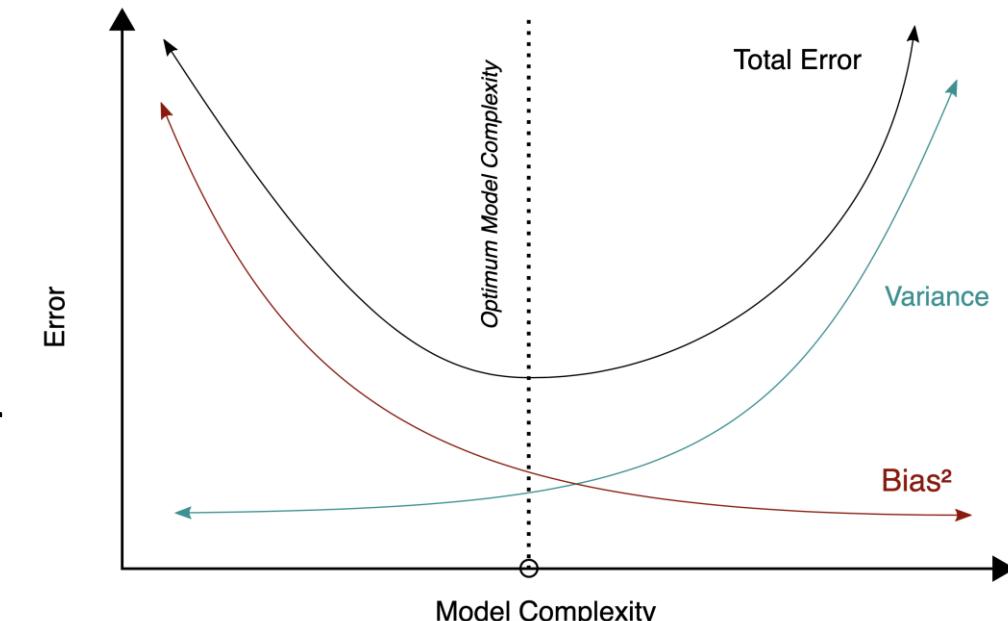
inherent randomness (target generated from random variable following probability distribution)

→ limiting accuracy of ideal model

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

# Bias-Variance Tradeoff

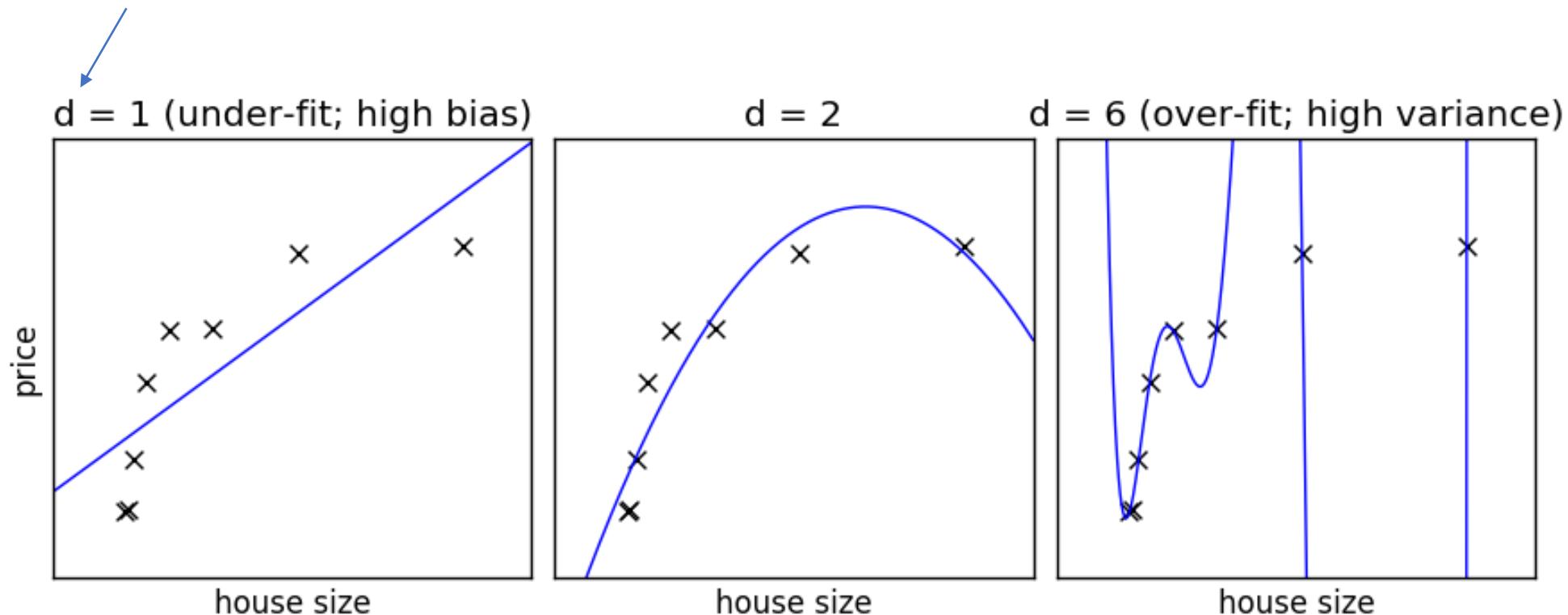
- fundamental concept in classical statistical learning theory
- models of higher complexity have lower bias but higher variance (given the same number of training examples)
- generalization error follows U-shaped curve:  
overfitting once model complexity (number of parameters) passes certain threshold
- overfitting: variance term dominating test error  
→ increasing model complexity increases test error



from wikipedia

# Example: Non-Linear Function Approximation

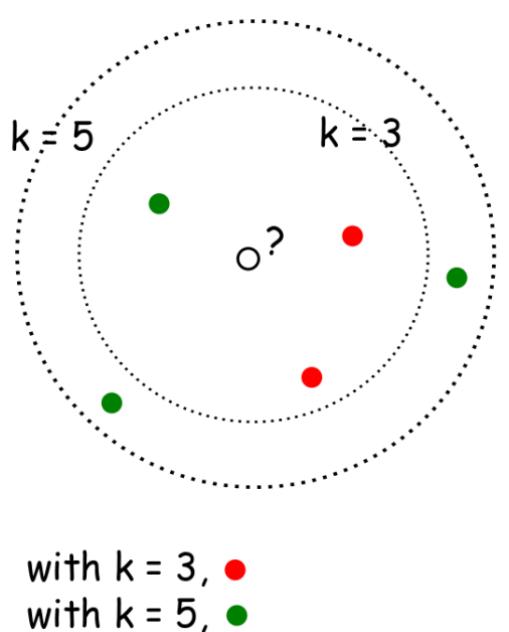
degree of fitted polynomial



from scikit-learn documentation

# Example: k-Nearest Neighbors

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



regression:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{j=1}^k y_j \quad \text{with } j \text{ running over } k \text{ nearest neighbors of } x_0$$

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

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

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

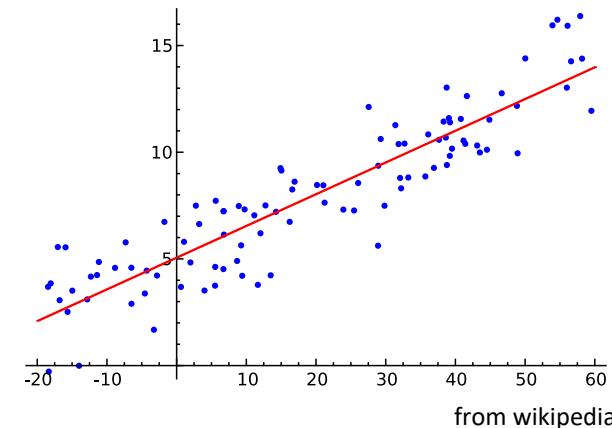
# Linear Regression

fit:

$$y_i = \hat{\alpha} + \sum_{j=1}^p \hat{\beta}_j x_{ij} + \varepsilon_i$$

(model)

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



parameters to be estimated:

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

$$\rightarrow \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

(approximating assumed true  $\alpha, \beta, \sigma$ )

predict:

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

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

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

Gaussian      mean      variance  
(reflected by  $\varepsilon_i$  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

$$L(y_i, \hat{f}(x_i); \hat{\theta})$$

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

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

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

$$L(y_i, \hat{f}(x_i); \hat{\theta}) = (y_i - \hat{f}(x_i; \hat{\theta}))^2$$

# Cost Function

averaging losses over (empirical) training data set:

$$J(\hat{\theta}) = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{f}(x_i); \hat{\theta})$$

cost function to be minimized according to model parameters  $\hat{\theta}$   
→ objective function

# Cost Minimization

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

$$\nabla_{\hat{\theta}} J(\hat{\theta}) = 0$$

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

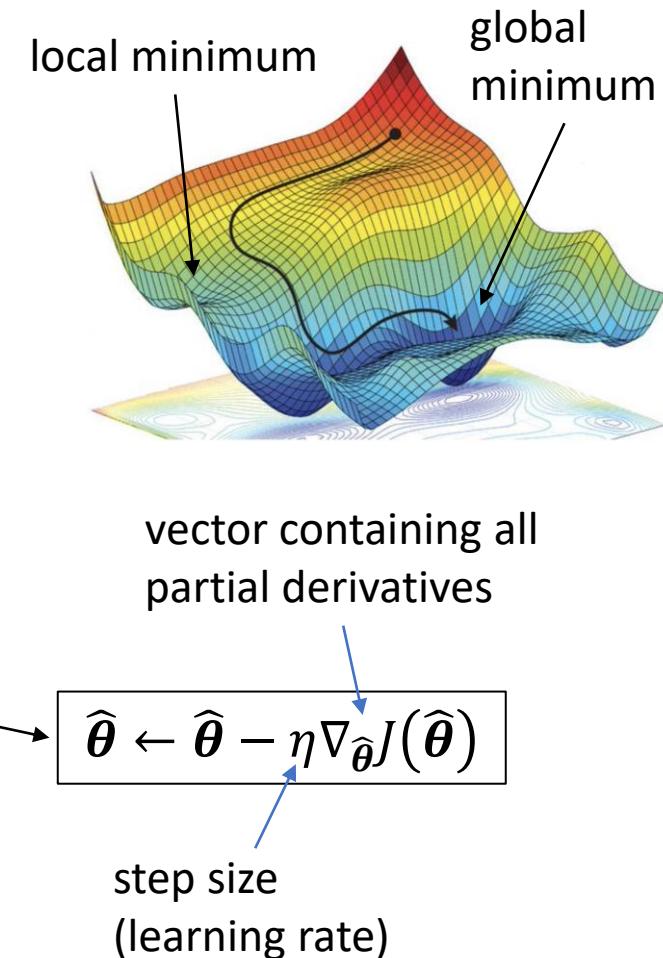
$$\nabla_{\hat{\theta}} \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i; \hat{\theta}))^2 = 0$$

analytical solution for linear regression: ordinary least squares

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

**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

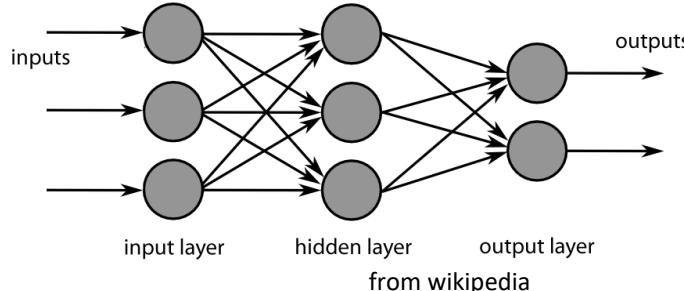
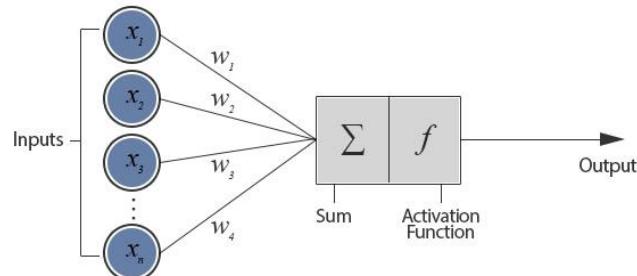


# Algorithmic Families

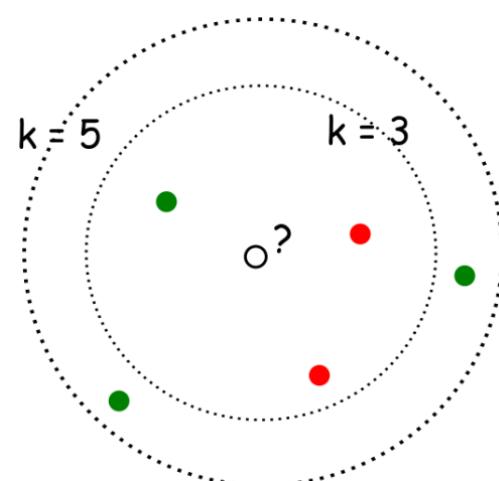
## linear (parametric) models

- linear regression
- Generalized Linear Models
- Generalized Additive Models

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



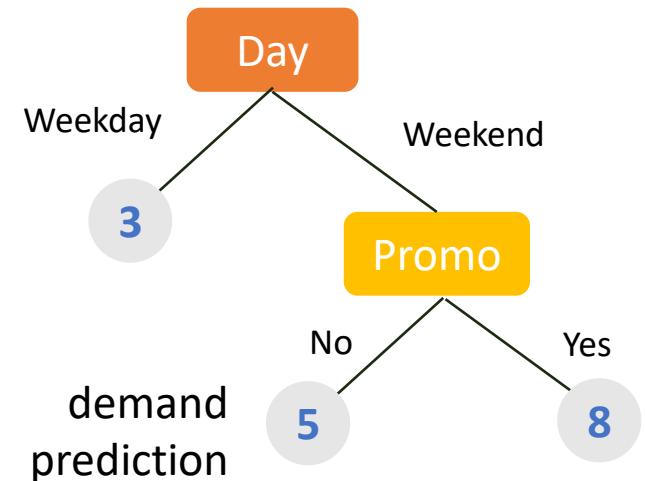
## nearest neighbors (local methods, instance-based learning) – non-parametric models



with  $k = 3$ , ●  
with  $k = 5$ , ●

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

## decision trees: rule learning



## often used in ensemble methods

- bagging: random forests
- boosting: gradient boosting

Most ML algorithms (even unsupervised and RL) 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)

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

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

# Deep Learning

# Recap: Goal of ML

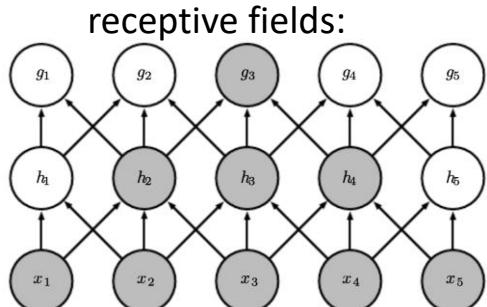
generalization from optimization on training data set (approximation of true data generating probability distribution by empirical risk minimization)

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

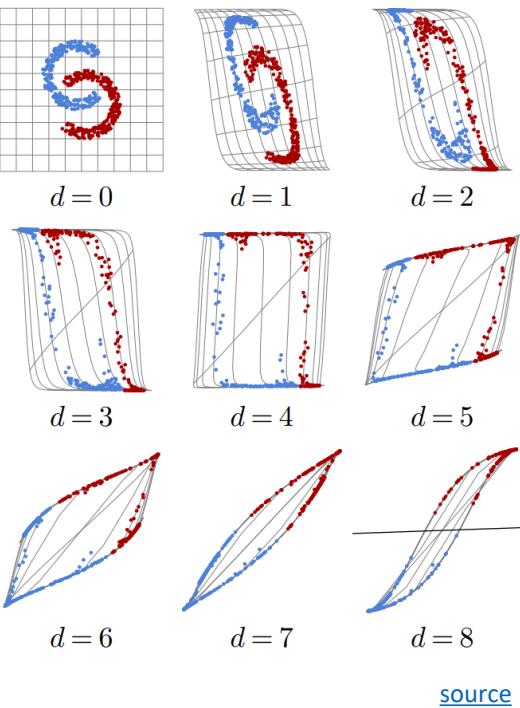
→ deep learning methods (neural networks with many layers) optimal candidates

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

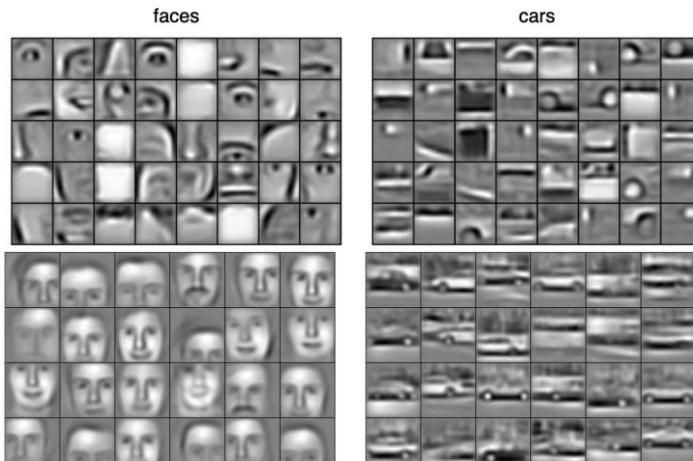
the deeper the better:  
accuracy, hierarchical representation (many aspects of nature hierarchical)



[source](#)



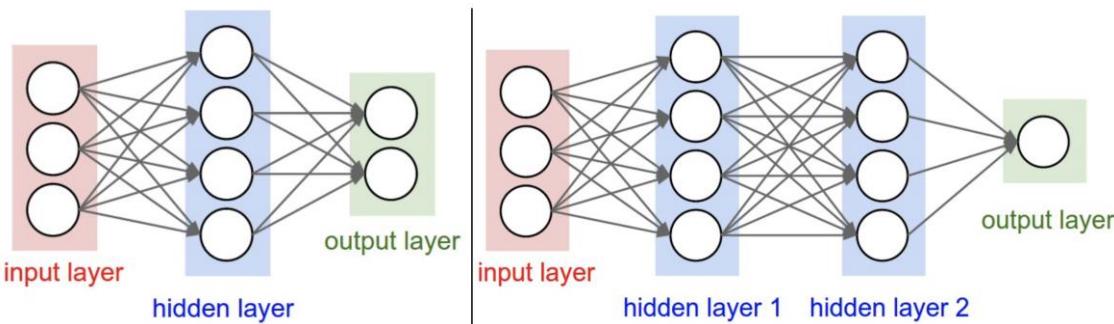
[source](#)



[source](#)

# Multi-Layer Perceptron (MLP)

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



toward deep learning: add hidden layers

more layers (depth) more efficient than just  
more nodes (width): less parameters needed  
for same function complexity

classification:

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

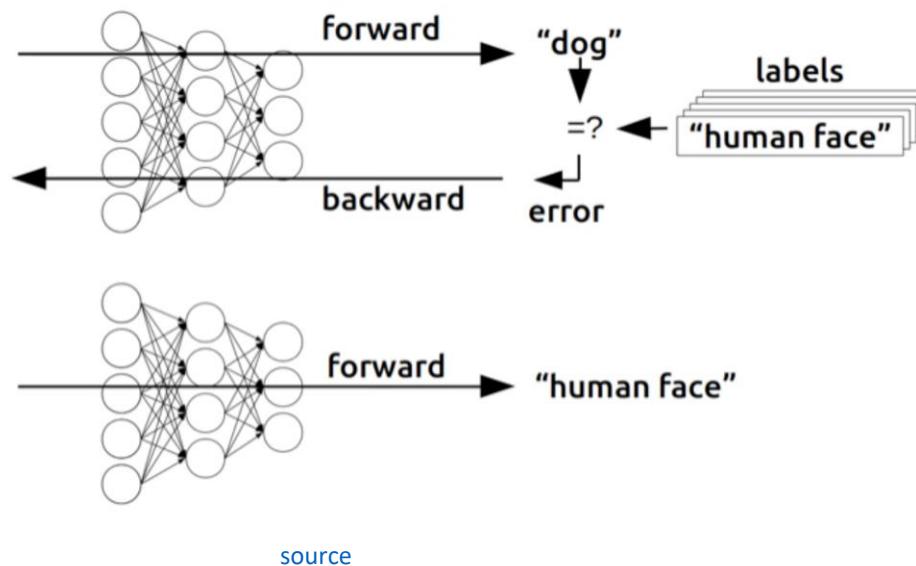
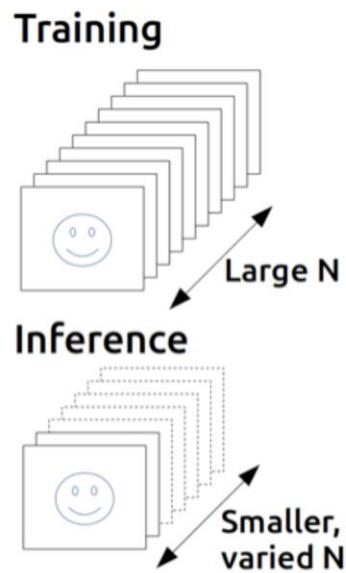
regression:

- 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



- 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)

**CNN** in short: local connections, shared weights, pooling, many layers

# Convolutional Neural Networks

## convolutional vs fully-connected layer:

convolutional vs  
fully-connected layer:

parameter sharing

kernel (learned  
parameters)

[source](#)

0	2	15	0	0	11	10	0	0	9	9	0	0
0	0	4	5	16	236	255	255	177	95	63	0	29
0	10	161	238	255	244	245	249	255	255	222	0	10
0	14	178	255	255	244	254	253	255	245	249	251	124
0	92	255	255	255	251	251	211	141	116	125	251	238
13	217	243	255	151	332	229	57	0	10	13	232	255
16	229	252	254	12	0	7	7	0	0	237	252	235
14	245	255	212	15	11	9	3	0	0	236	243	255
0	81	252	250	248	170	60	1	0	128	255	244	6
0	13	111	255	255	245	182	181	246	252	242	208	36
1	0	111	251	255	241	254	247	255	245	121	17	0
0	0	4	35	251	255	246	254	253	250	11	0	1
0	0	4	32	255	255	255	248	252	255	244	182	10
0	23	208	252	244	251	241	10	0	11	245	255	254
0	111	255	252	255	158	0	4	0	6	255	232	230
18	218	250	250	137	7	11	0	0	2	255	250	123
17	178	255	255	101	9	20	0	13	1	182	251	245
0	100	251	245	255	230	98	55	19	112	217	248	253
0	18	146	255	255	247	255	255	248	255	240	255	94
0	0	73	111	215	255	250	248	255	242	248	111	12
0	0	6	1	5	161	233	255	257	147	37	0	4
0	0	5	0	0	0	0	0	0	1	0	6	0

## source

many images  
(training examples),  
potentially with  
several channels

several kernels  
producing several  
feature maps

several kernels producing several feature maps

down-sampling by convolutions and pooling

flatten dimensions for final classification or regression

highly regularized form of feed-forward neural networks

The diagram illustrates the relationship between convolutional operations and spatial inductive bias, highlighting translation equivariance and locality.

**translation equivariance**

**locality**

**convolution:**

**Striding in CONV**

**input**

**spatial inductive bias**

The diagram shows two input grids representing spatial inductive bias. The top grid has stride  $S=1$ , where each green 2x2 receptive field is mapped to a single blue output unit. The bottom grid has stride  $S=2$ , where each green 2x2 receptive field is mapped to every second blue output unit. A label "Striding in CONV" points to the bottom grid. A label "S=1" is associated with the top grid, and a label "S=2" is associated with the bottom grid. A large bracket labeled "convolution:" spans both grids. A bracket labeled "translation equivariance" covers the top section, and a bracket labeled "locality" covers the right section.

# translation invariance

## max pooling:

Single depth slice				
X	1	0	2	3
4	6	6	8	
3	1	1	0	
1	2	2	4	

from wikipedia

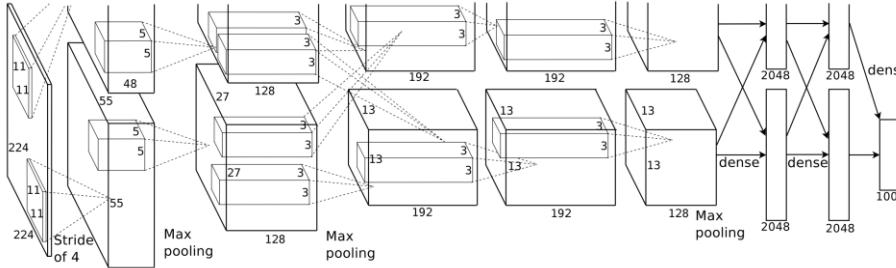
# Rise of Deep Learning

a little bit oversimplified:

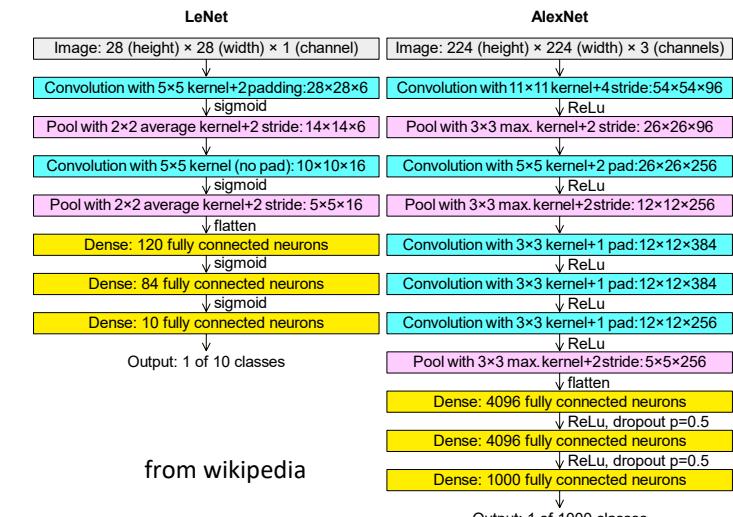
deep learning = lots of training data + parallel computation + smart algorithms

AlexNet:

ImageNet (with data augmentation) + GPUs (allowing more layers) + ReLU, dropout  
(pivotal moment for deep learning: ImageNet challenge 2012)



source



# Training Subtleties of Deep Neural Networks

optimization and regularization difficult

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

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

optimization

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

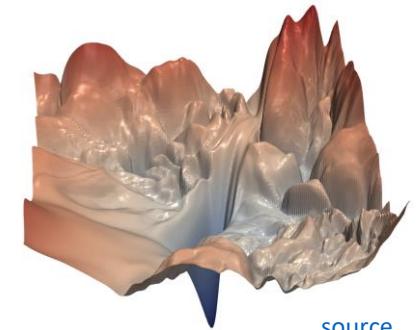
explicit regularization

- weight decay
- dropout
- data augmentation
- weight sharing

implicit regularization

- early stopping
- batch normalization
- stochastic gradient descent

typical loss surface:



[source](#)

# 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

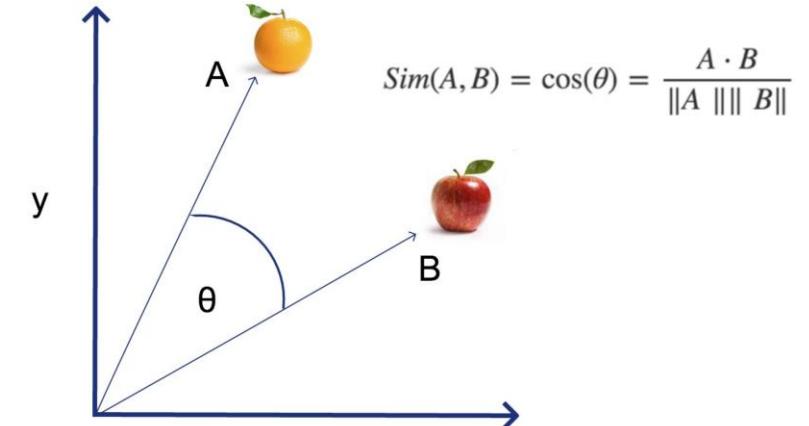
representation of entities by vectors

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

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

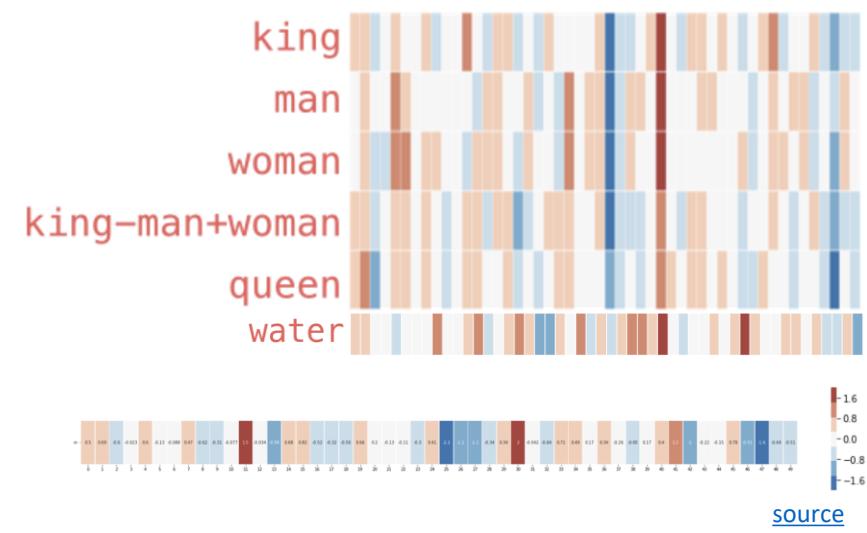
but general concept: embeddings of  
(categorical) features (e.g., products in  
recommendation engines)

learned via co-occurrence (e.g., [word2vec](#))



but also direction of difference  
vectors interesting (analogies):

king – man + woman ≈ queen

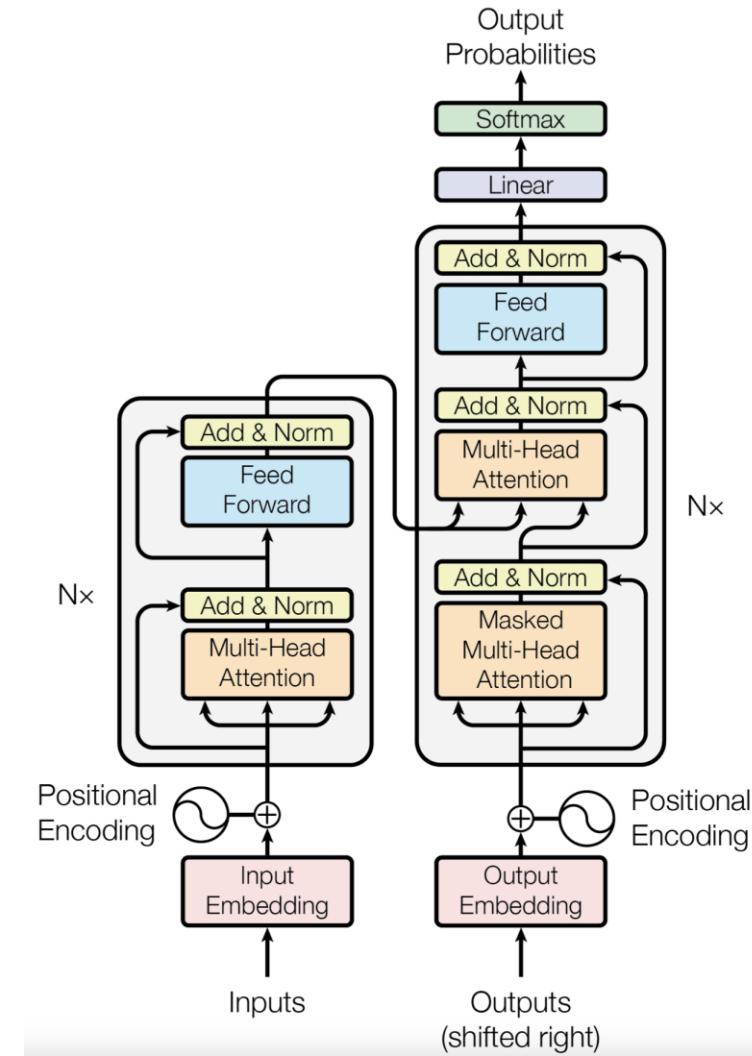


# Transformer

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 bigger models (more parameters)

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

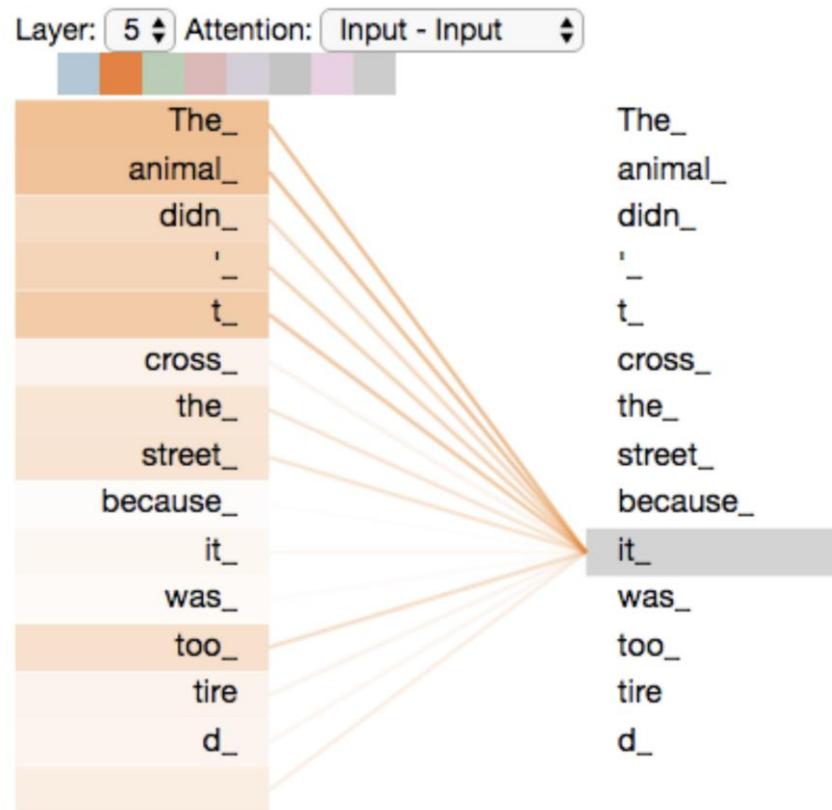


[source](#)

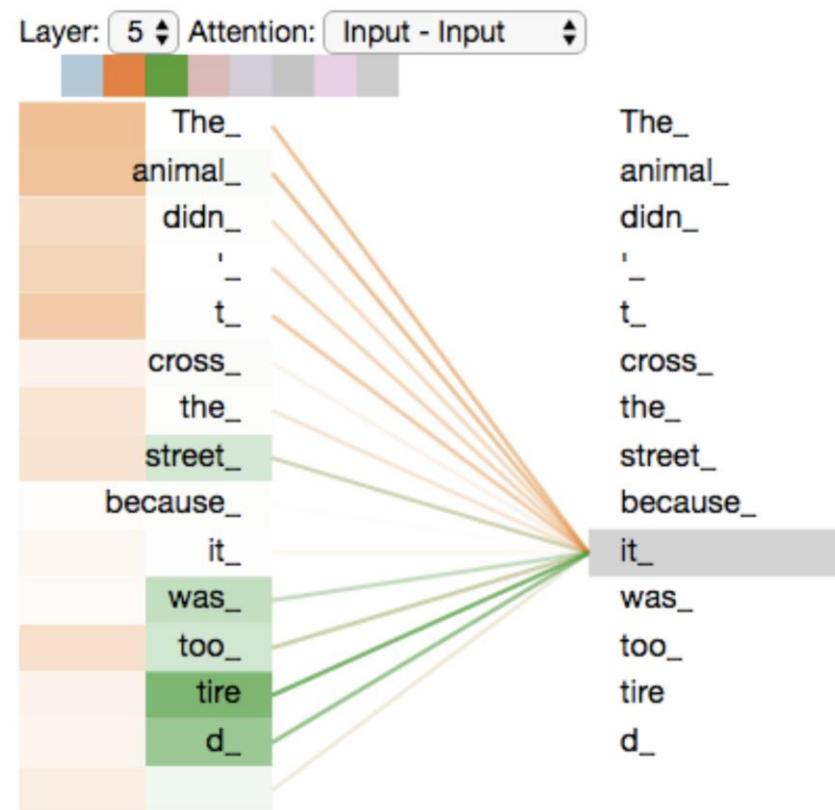
# Self-Attention

computational complexity  
quadratic in length of input (each token attends to each other token)

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)

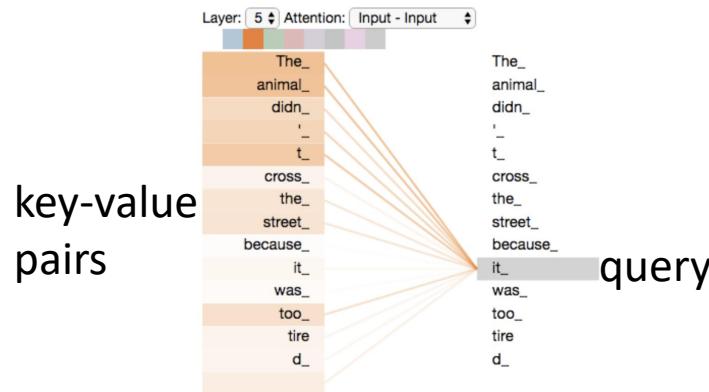


source

# Scaled Dot-Product Attention

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

- query Q
- key K
- value V



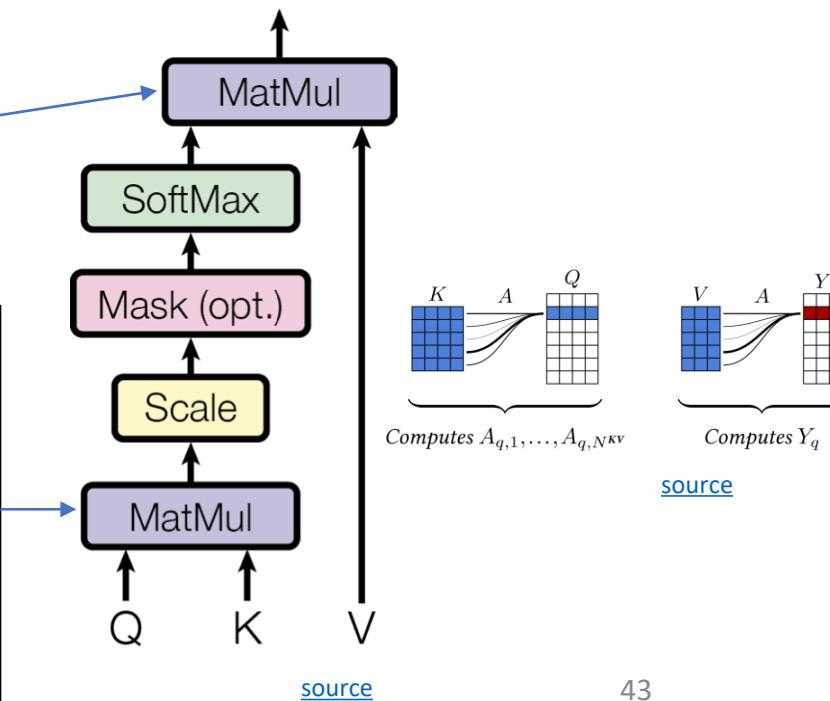
scoring each of the key words (context) with respect to current query word: multiplication of inputs (in contrast to inputs times weights in neural networks)

filtering: multiplication of attention probabilities with corresponding key word values

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

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

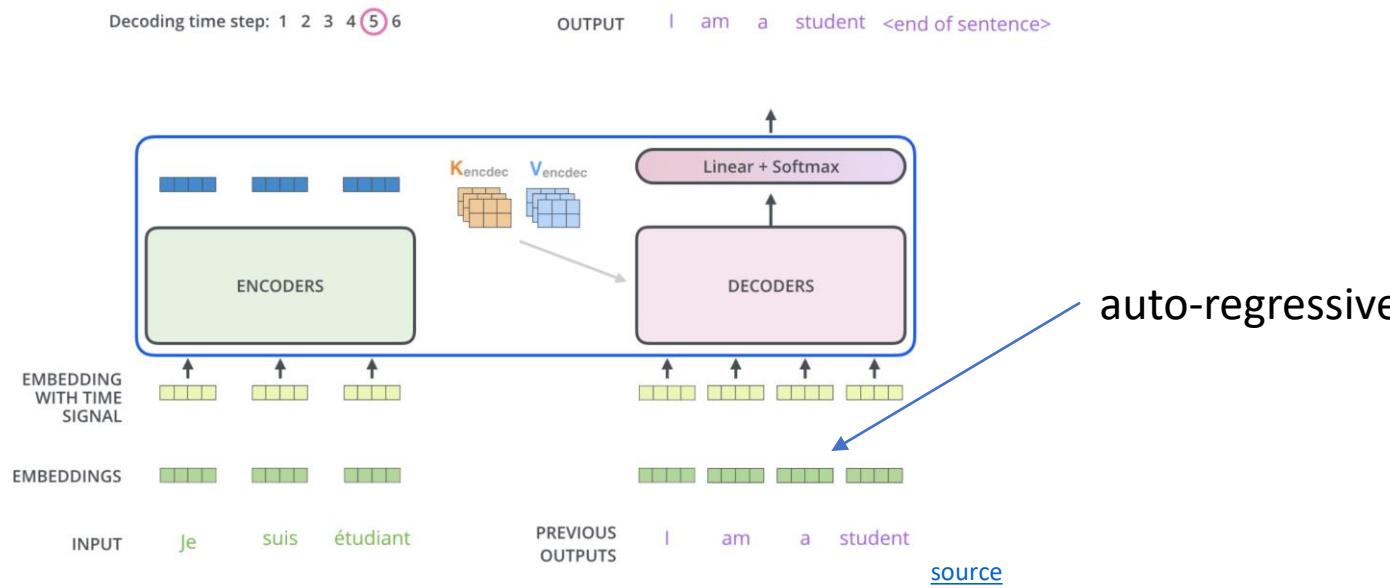
Scaled Dot-Product Attention



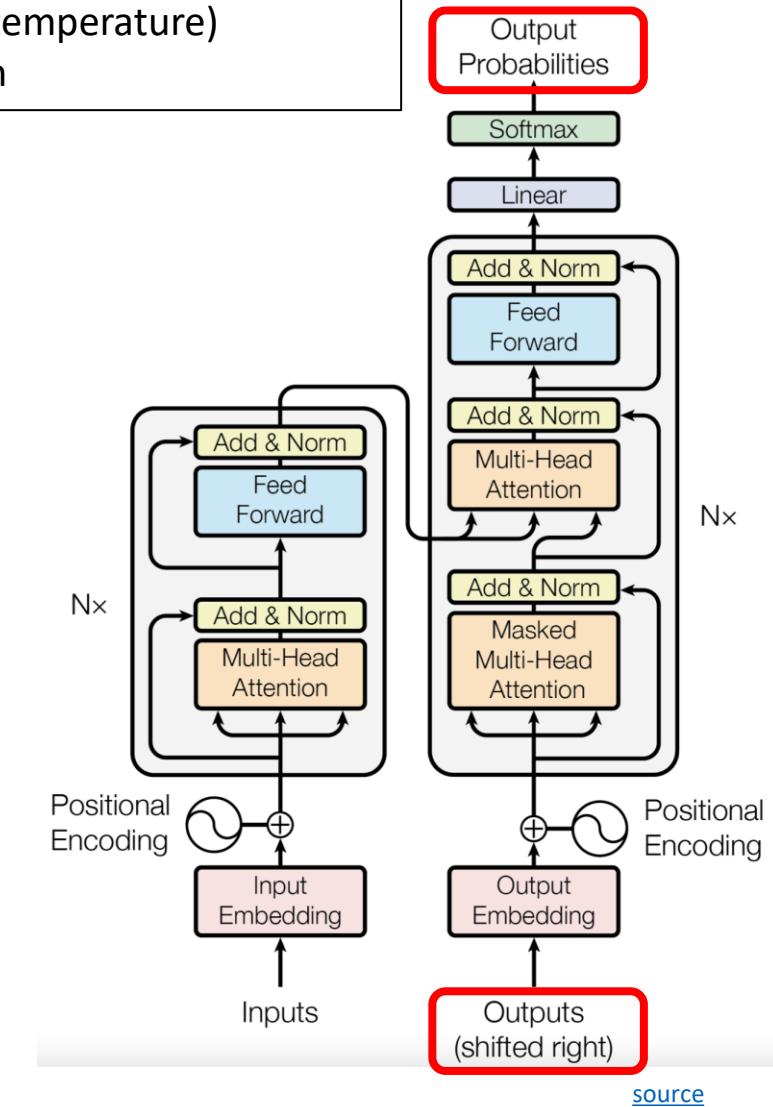
# Sequence Completion

- greedily picking the one with highest probability
- pick according to probabilities (degree of randomness controlled by softmax temperature)
- beam search

for each step/token (iteratively), choose one output token to add to decoder input sequence → increasing uncertainty



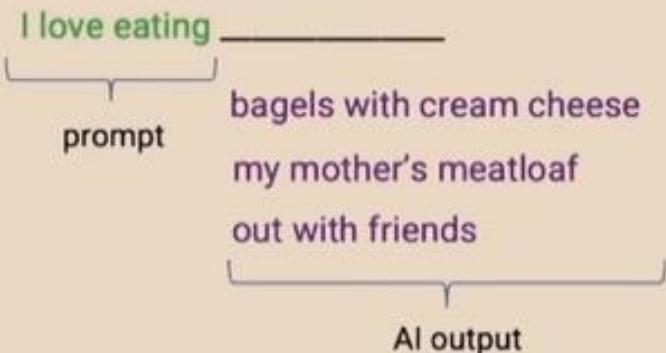
prompt: externally given initial sequence for running start and context on which to build rest of sequence ([prompt engineering](#))



[source](#)

# This decade: Generative AI

## Text generation process



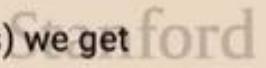
## How it works

Generative AI is built by using supervised learning ( $A \rightarrow B$ ) to repeatedly predict the next word.

*My favorite food is a bagel with cream cheese and lox.*

Input (A)	Output (B)
My favorite food is a	bagel
My favorite food is a bagel	with
My favorite food is a bagel with	cream

When we train a very large AI system on a lot of data (hundreds of billions of words) we get a Large Language Model like ChatGPT.



Andrew Ng



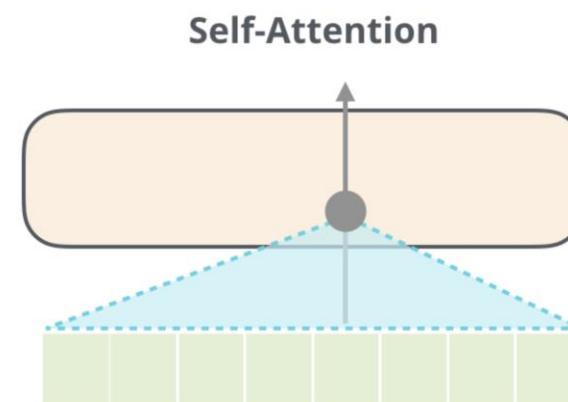
# Typical Transformer Architectures for LLMs

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

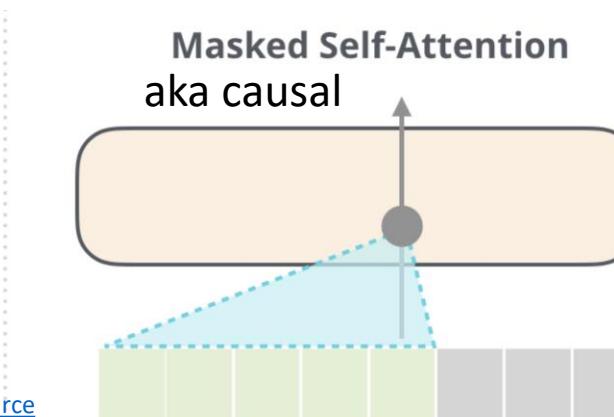
**encoder-only** LLMs:

- learning of contextual embeddings (and subsequent fine-tuning)
- training: prediction of masked words
- can't generate text, can't be prompted

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



source



source

**decoder-only** LLMs:

- text generation, e.g., chat bot (after instruction tuning)
- training: next-word prediction
- output one token at a time (auto-regressive: consuming its own output)

examples: OpenAI's [GPT](#) (Generative Pre-trained Transformer), Meta's [Llama2](#)

[GPT-4 capabilities](#)

# Multi-Task Learning of LLMs

compositional nature of deep learning (and homogeneity of unstructured data like text or images) 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 )

the new paradigm:

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

typical prompt: instructions, context (potentially retrieved externally from, e.g., knowledge-base embeddings), query, output indicator  
with (few-shot) or without (zero-shot) providing explicit examples

[prompt engineering](#)

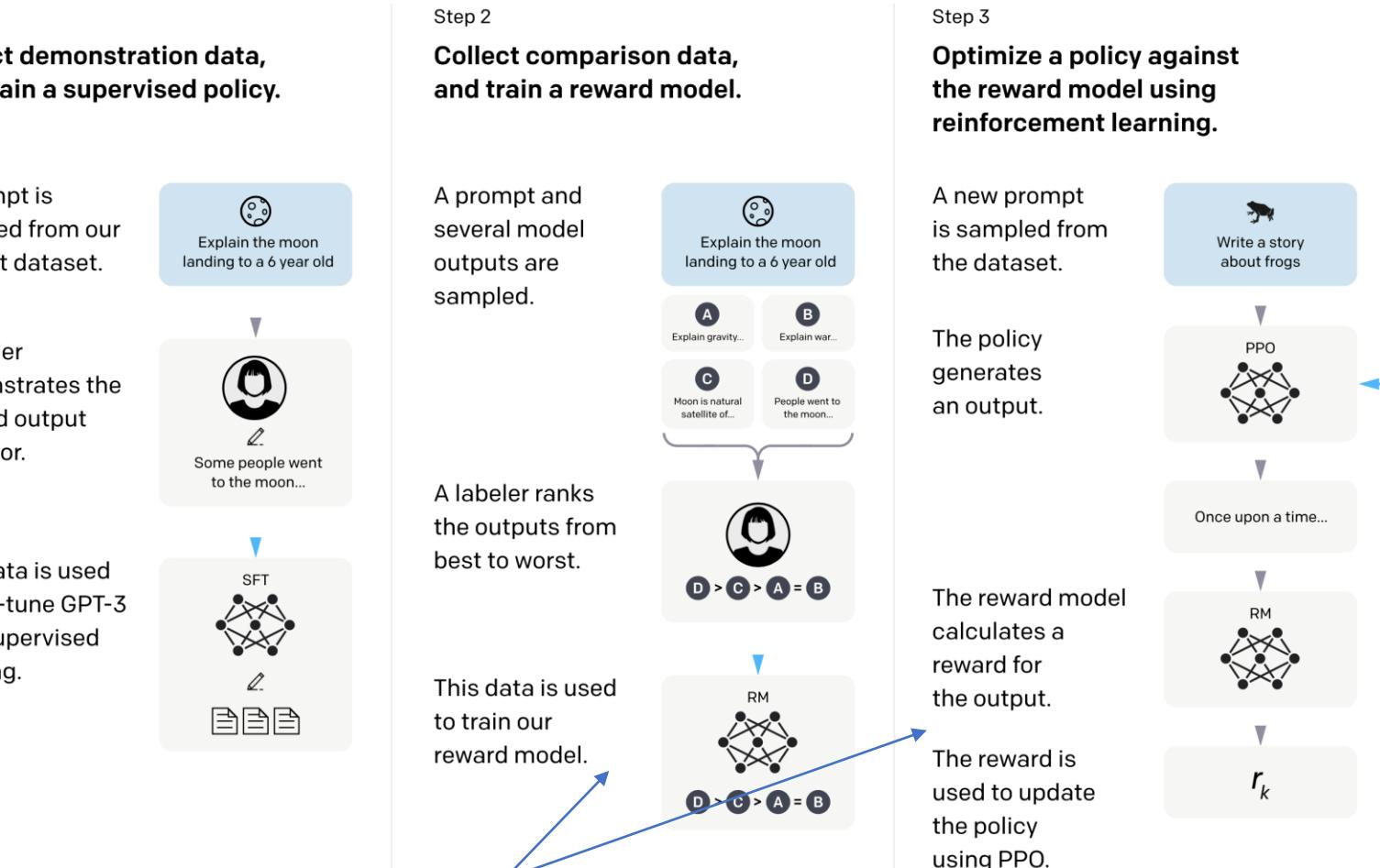
# Conversational AI: RL from Human Feedback

example for supporting large language models (transformers) with RL

used in famous ChatGPT

goal: improve alignment with user intentions

→ learn from human preferences



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

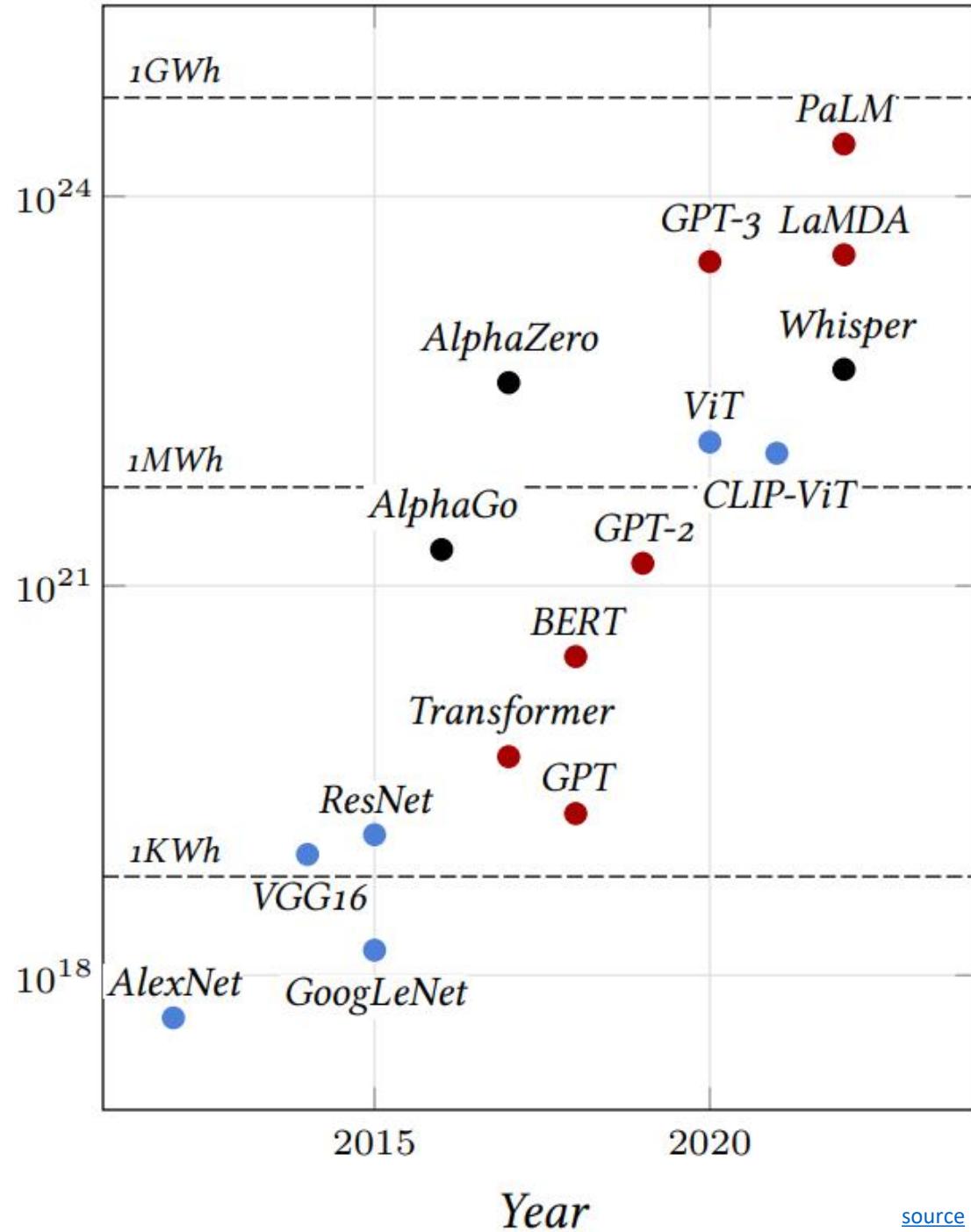
# LLMs in Plain Terms

foundation models:

- compression of the internet
- programming languages of new wave of AI applications (adapted to specific use cases and data)

→ These applications will make the internet more interactive.

*Training cost (FLOP)*



[source](#)

(open-source) SOTA (July 2024):  
Llama 3

### Closed-source vs. open-weight models

Llama 3.1 405B closes the gap with closed-source models for the first time in history.

MMU (5-shot)



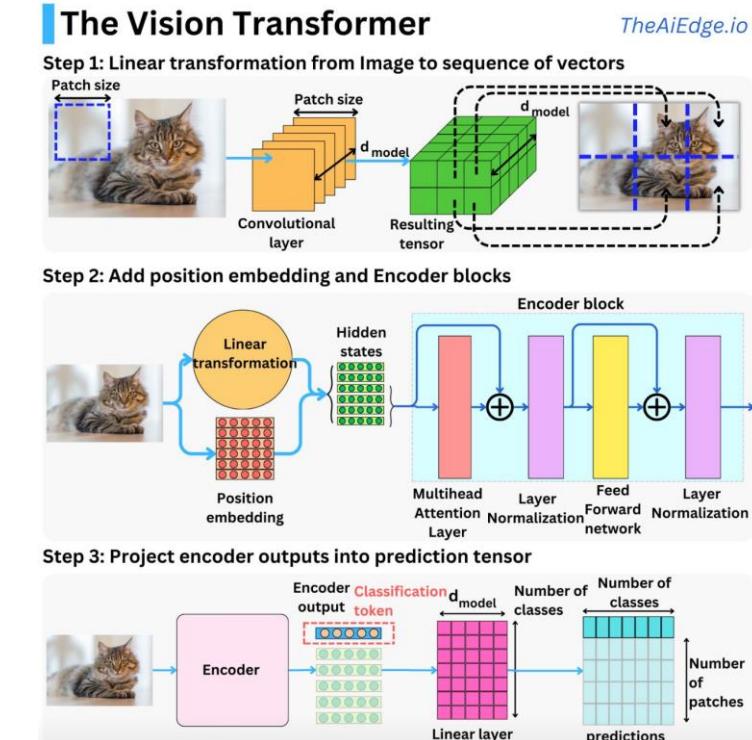
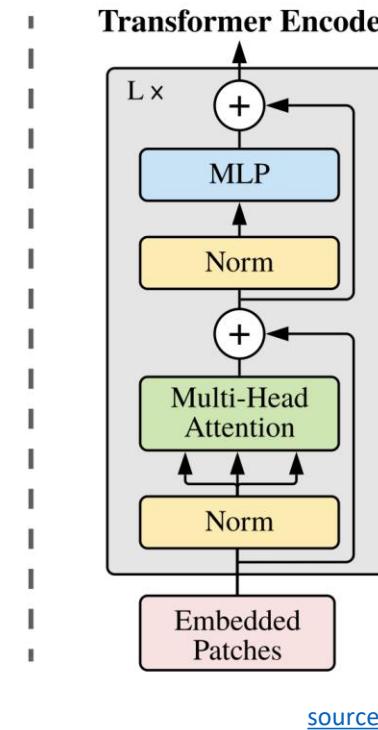
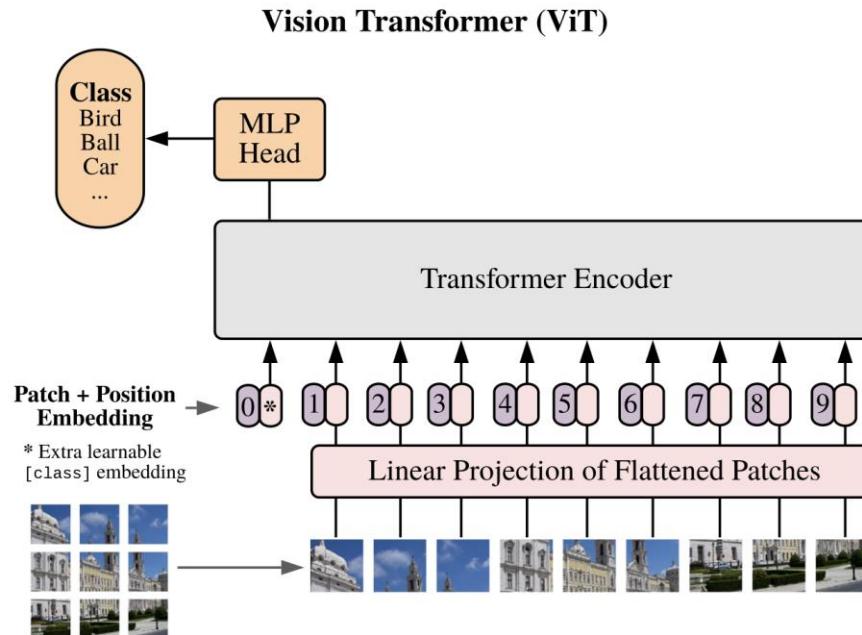
a few surveys: [1](#), [2](#), [3](#)

# Some LLM Numbers

example Llama 3 405B:

- vocabulary size (tokens): 128K
  - embedding/model dimensions: 16,384
  - parameters: 405B
  - training tokens: 15.6T
  - context length/window (tokens): 128K
  - training hardware: 16K GPUs (H100)
- factor less than 40  
→ a lot of memorizing

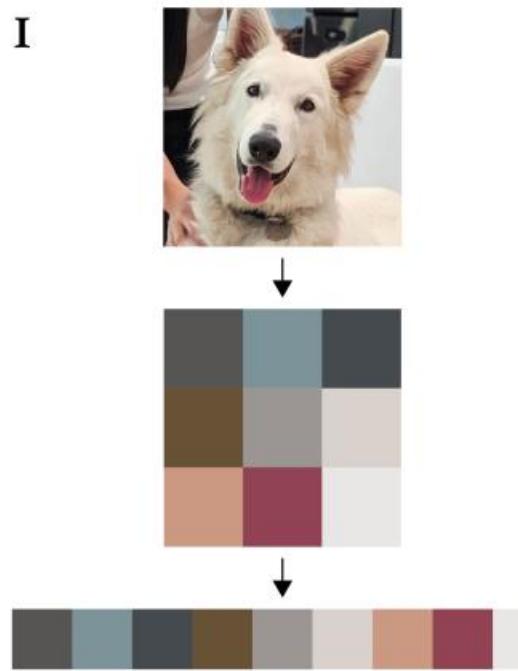
# Image Classification with Vision Transformer



formulation as sequential problem:  
split image into patches (tokens) and flatten,  
add positional embeddings

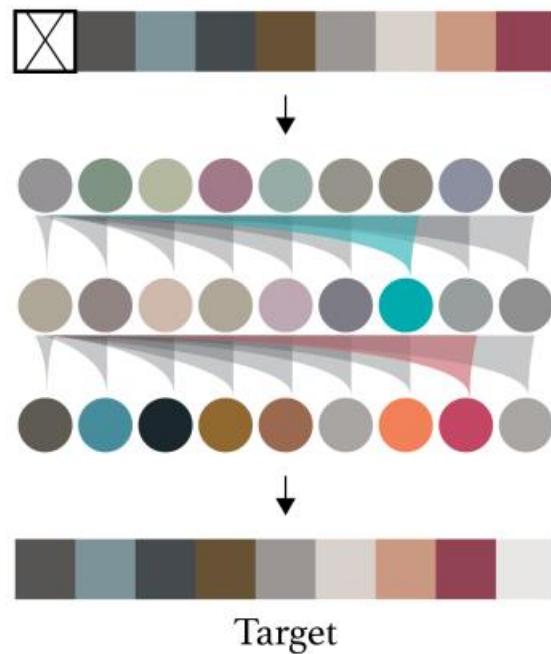
processing by transformer encoder:  
pre-train with image labels, fine-tune  
on specific data set

# Pixel Generation (iGPT)

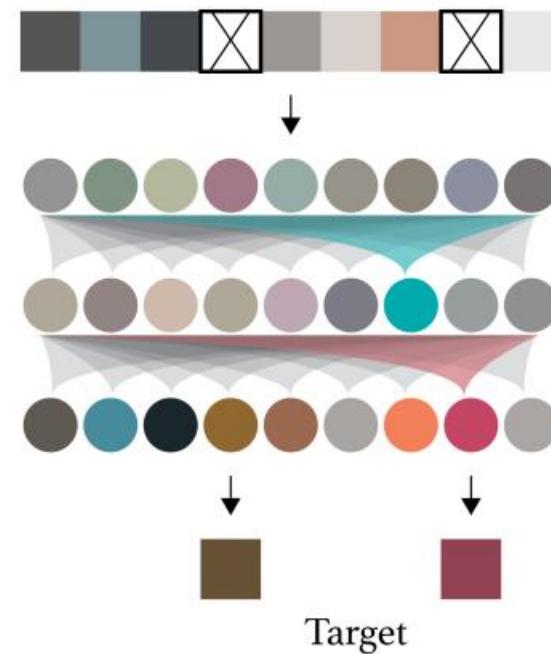


2

(a) Autoregressive



(b) BERT

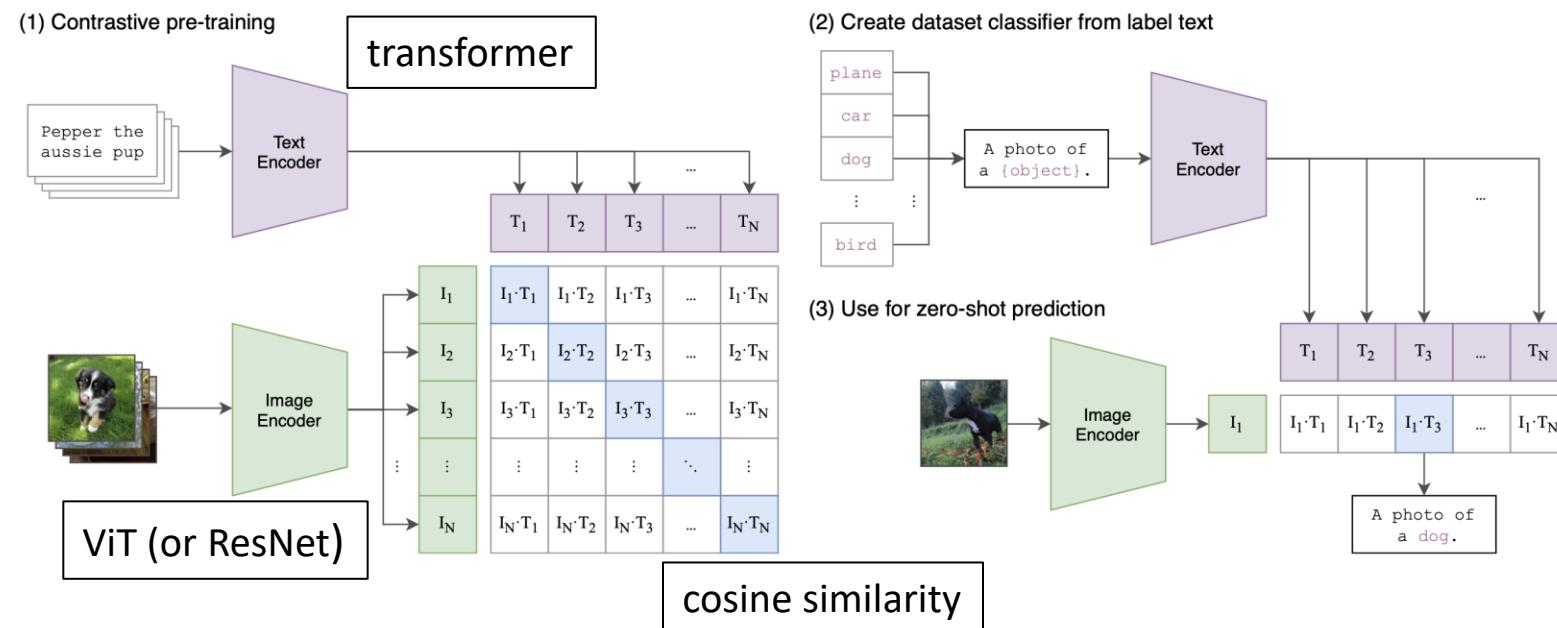


[source](#)

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

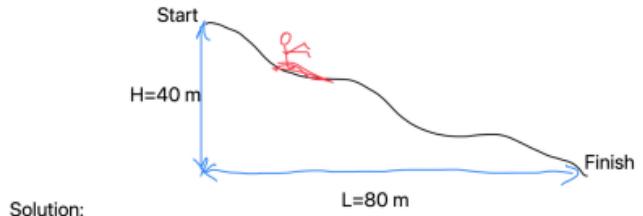
transformers good candidate (universal and flexible architecture, little task-specific inductive bias)

# Google's Gemini:

## Prompt

Here's a solution to a Physics problem by a student.

23. A skier slides down the frictionless slope as shown. What is the skier's speed at the bottom?



The conservation of energy means that the energy of the skier is same at the beginning and at the end.

$$\text{Start: } E = mgL$$

$$\text{End: } E = \frac{1}{2}mv^2$$

$$\Rightarrow mgL = \frac{1}{2}mv^2$$

$$\Rightarrow v = \sqrt{2gL} = \sqrt{2 \times 9.81 \times 80} = 39.6 \text{ m} \cdot \text{s}^{-1}$$

Try to reason about the question step by step.

Did the student get the correct answer? If the solution is wrong, please explain what is wrong and solve the problem.

Make sure to use LaTeX for math and round off the final answer to two decimal places.

## Response

The student did not get the correct answer. The student made a mistake in the calculation of the potential energy at the start of the slope. The potential energy at the start of the slope is given by  $mgh$ , where  $m$  is the mass of the skier,  $g$  is the acceleration due to gravity, and  $h$  is the height of the slope. The student used the length of the slope,  $L$ , instead of the height,  $H$ , in the calculation of the potential energy.

The correct solution is:

The conservation of energy means that the total energy of the skier is the same at the beginning and at the end of the slope.

At the start of the slope, the skier has only potential energy, which is given by  $mgh$ .

At the end of the slope, the skier has only kinetic energy, which is given by  $\frac{1}{2}mv^2$

Therefore, we can write:

$$mgh = \frac{1}{2}mv^2$$

Cancelling out the mass  $m$  on both sides, we get:

$$gh = \frac{1}{2}v^2$$

Solving for  $v$ , we get:

$$v = \sqrt{2gh}$$

Substituting the given values, we get:

$$v = \sqrt{2 \times 9.81 \times 40} = 28.01 \text{ m/s}$$

Therefore, the skier's speed at the bottom of the slope is 28.01 m/s.

# Hot LLM Research Topics

transformer efficiency

- (sparse) [mixture of experts](#) (e.g., [Mixtral 8x7B](#), [Gemini 1.5](#))
- sparse attention (e.g., [Longformer](#))
- minimize memory reads/writes ([FlashAttention](#))

or non-transformer architectures (e.g., [Mamba](#))

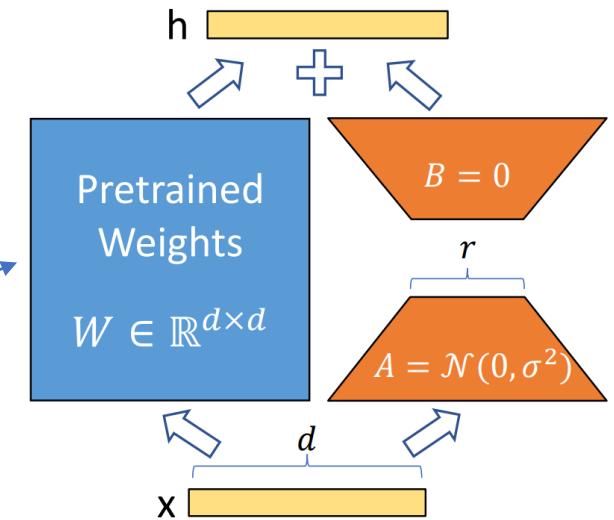
prompting strategies

- let LLM agents show reasoning/planning capabilities
- use tools (also embodiment/grounding)
- prompt optimization (e.g., [OPRO](#))

fine-tuning efficiency (e.g., [LoRA](#))

RAG

} make use of your own data



# Generative Models

# Generative vs Discriminative Models

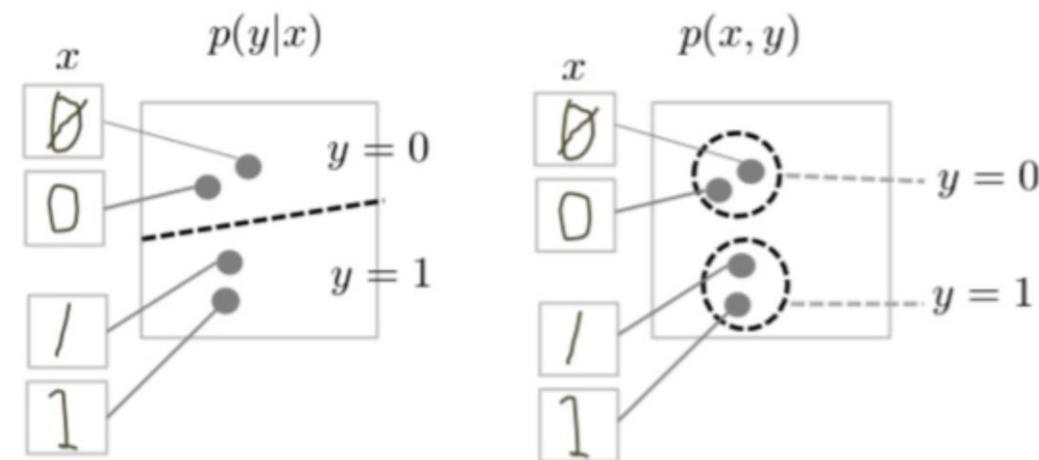
generative models: predict joint probability  
 $P(Y, X)$  (what allows to create new data samples) or directly generate new data samples

or just  $P(X) \rightarrow$  unsupervised (or self-supervised) learning

discriminative models: predict conditional probability (or probability distribution for regression)  $P(Y|X)$  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](#)

example: [DALL-E 2](#)

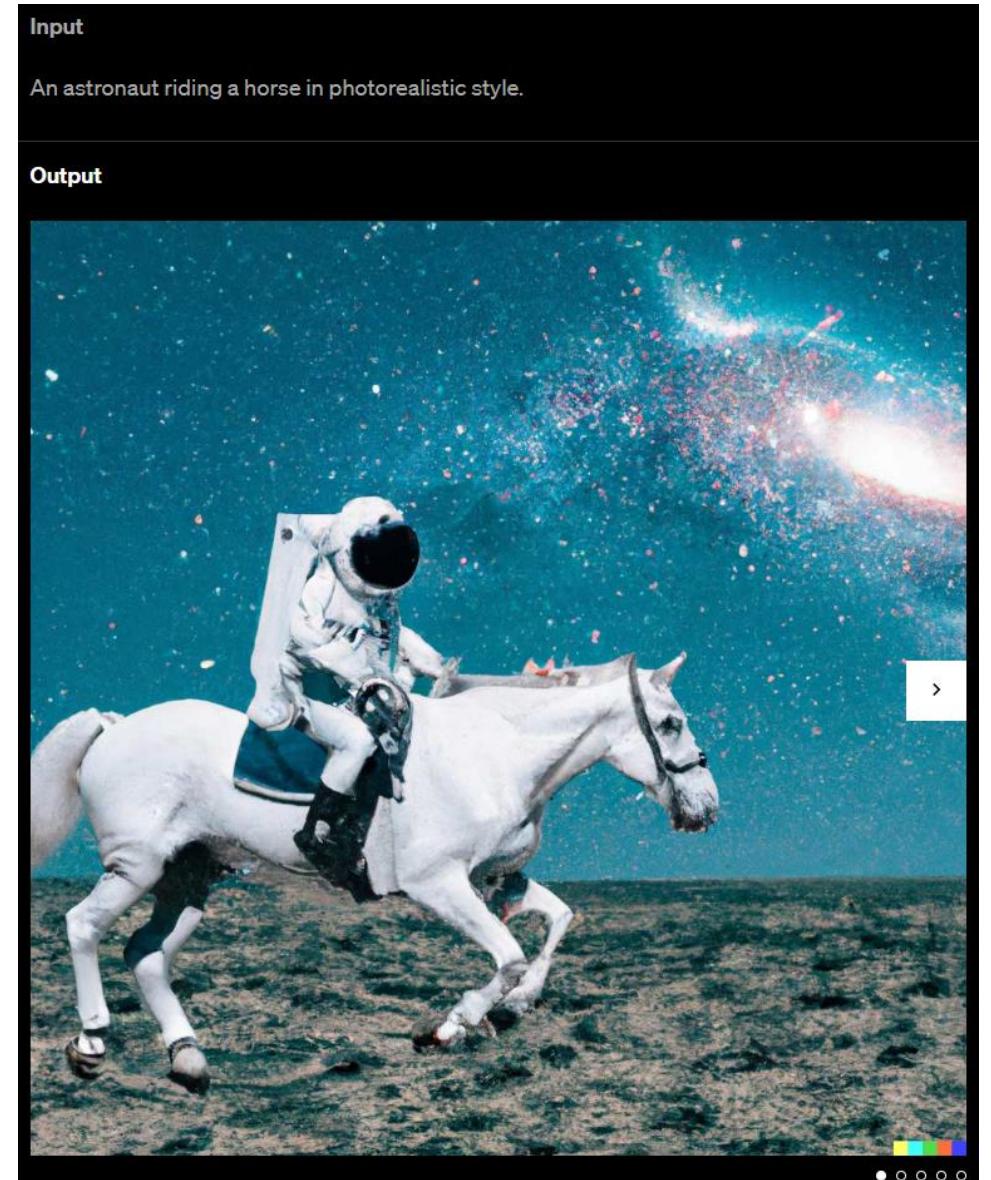
# Image Synthesis

idea: generate new images as variations  
of training data

condition generation on text prompts:  
text-to-image

trade-off between diversity and fidelity

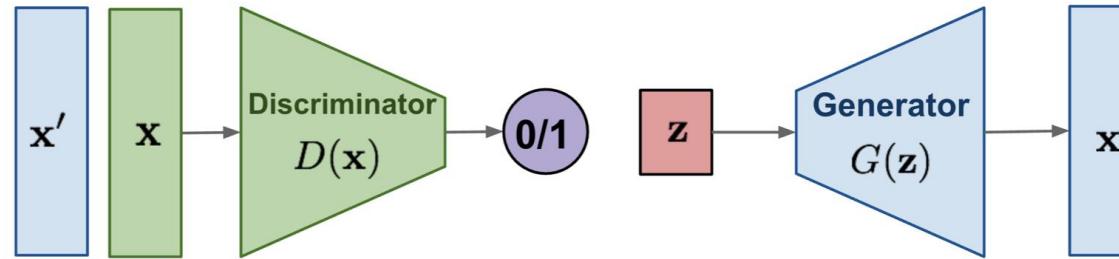
SOTA: (guided) diffusion models



conditioned on CLIP embeddings

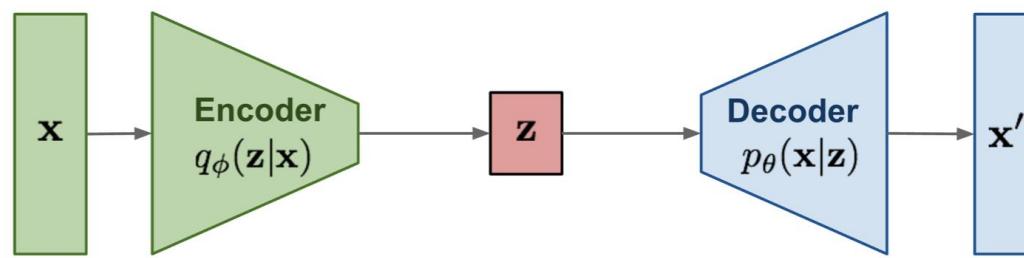
# Different Model Types for Image Synthesis

**GAN:** Adversarial training



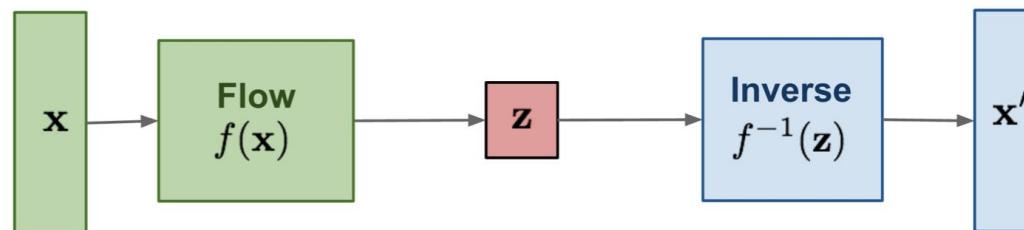
two neural networks playing a zero-sum game

**VAE:** maximize variational lower bound



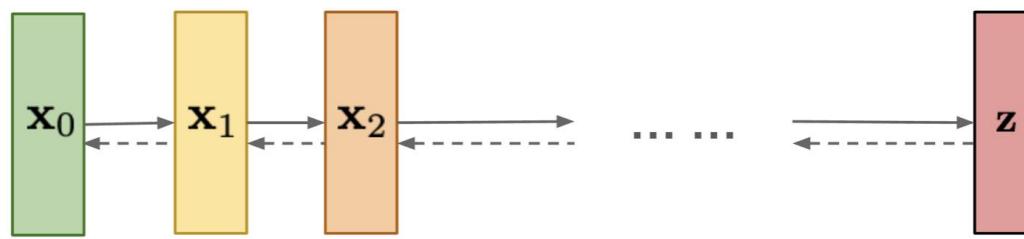
learn variational distribution (not just replicating inputs)

**Flow-based models:**  
Invertible transform of distributions



more complex distributions by applying change-of-variable technique (need for specialized architecture)

**Diffusion models:**  
Gradually add Gaussian noise and then reverse



chain of denoising autoencoders

[source](#)

→ generalization: [flow matching](#)

# Classic ML vs Deep Learning

# Generative vs Predictive Models

## text generation

```
done = False
total_reward = 0
while not done:
    state = torch.tensor(state, dtype=torch.float32)
    logits = policy_network(state)
    action = torch.argmax(logits).item()
    next_state, reward, done, _ = env.step(action)
    total_reward += reward
    state = next_state
    total_rewards.append(total_reward)

# Print average reward
print("Average Reward:", np.mean(total_rewards))
```

This code uses PyTorch to implement the policy gradient method (REINFORCE algorithm) to solve the CartPole problem. The policy network is defined as a simple feedforward neural network, and the training loop updates the policy network parameters to maximize the expected reward. Finally, it evaluates the learned policy by running it for 100 episodes and prints the average reward. Adjust hyperparameters and network architecture as needed for better performance.

Can you extend this to an actor-critic method?

Certainly! Here's an extension of the previous code using the actor-critic method to solve the CartPole problem:

Message ChatGPT

ChatGPT can make mistakes. Check important info.

ChatGPT

## BERT family



dmrc  
**XGBoost**

## tabular data

## image synthesis



Prompt: Epic anime artwork of a wizard atop a mountain at night casting a cosmic spell into the dark sky that says "Stable Diffusion 3" made out of colorful energy

[Stable Diffusion 3 — Stability AI](#)

## text-to-video



Prompt: A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She...

[Sora | OpenAI](#)

## computer vision

Classify



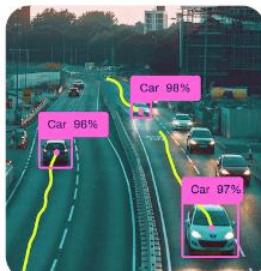
Detect



Segment

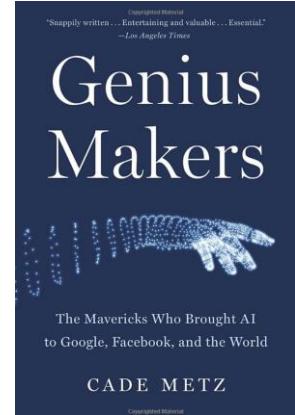


Track



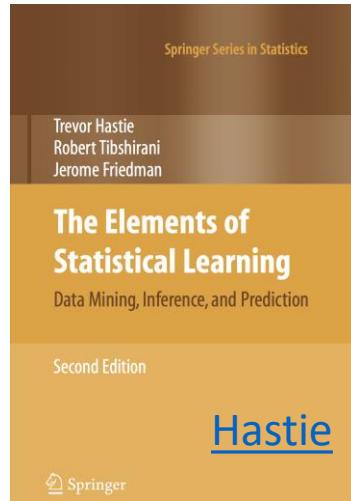
[YOLO](#)

historical overview  
of deep learning:

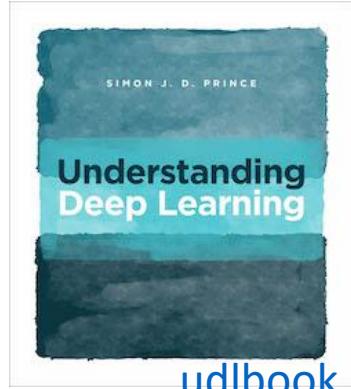


# Literature

foundations of ML:



covering newer topics:

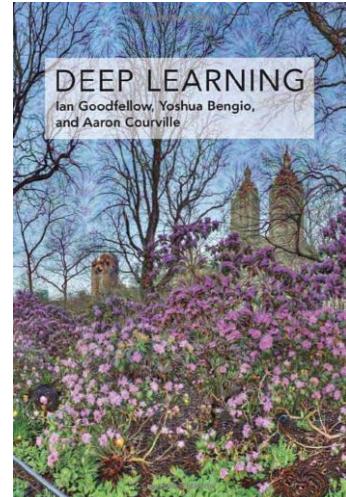


a few seminal papers:

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

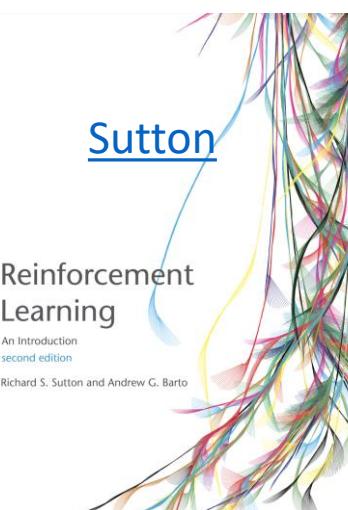
foundations of deep learning:

<https://www.deeplearningbook.org/>

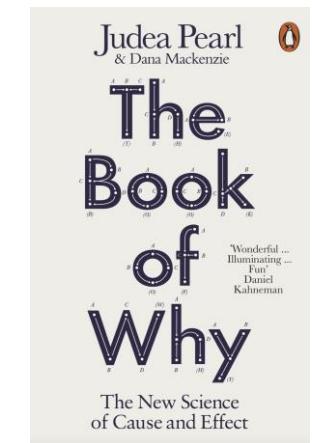


[The Little Book of Deep Learning](#)

overview of  
reinforcement learning:



Sutton



gentle but genuine  
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 difficult to foresee: emotions or consciousness occurring as emergent capabilities?