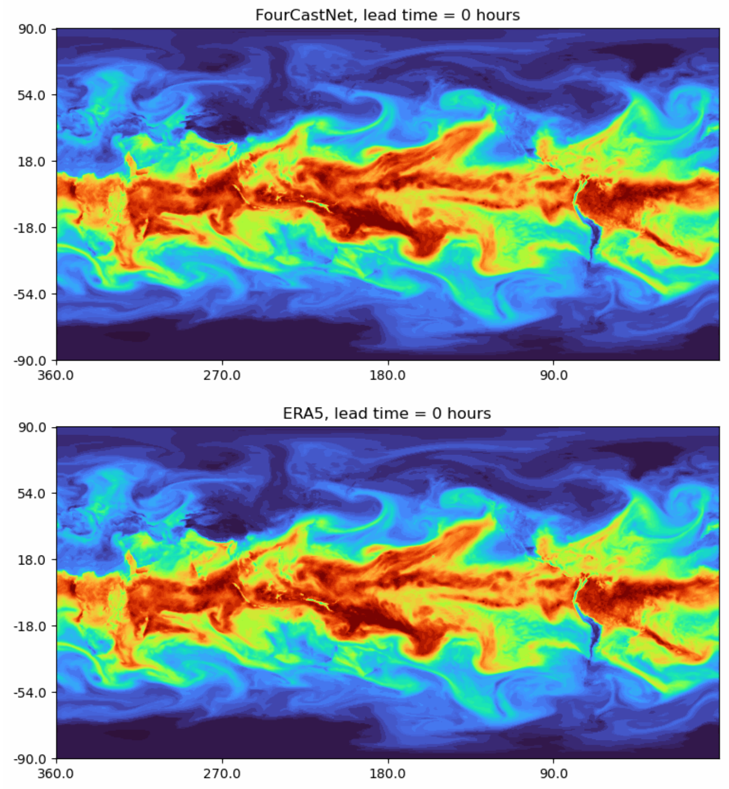
AI in the Sciences and Engineering

Introduction to Deep Learning – Part 1

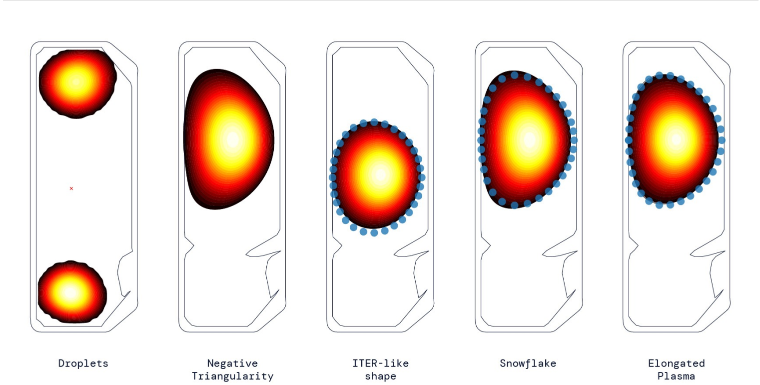
Spring Semester 2024

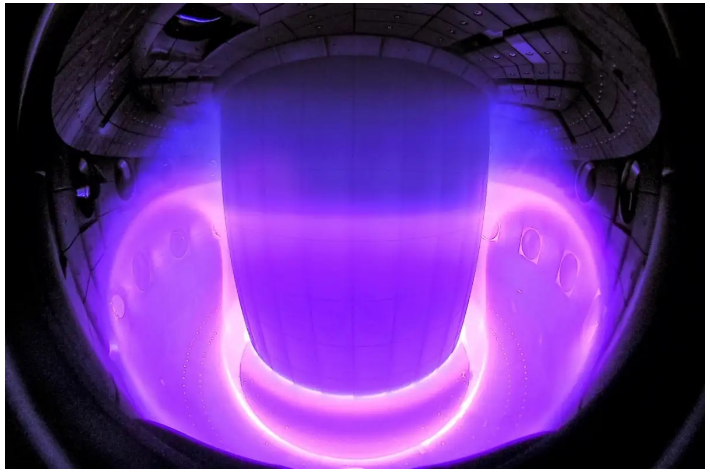
Siddhartha Mishra
Ben Moseley

Recap – AI for science

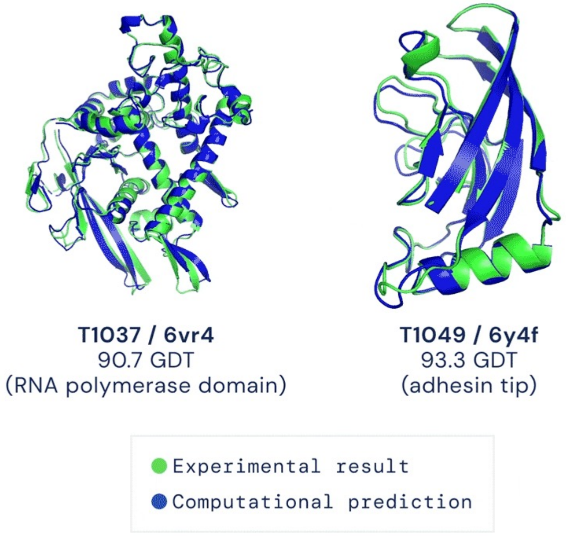


Pathak et al, FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, ArXiv (2022)





Degrave et al, Magnetic control of tokamak plasmas through deep reinforcement learning, Nature (2022)

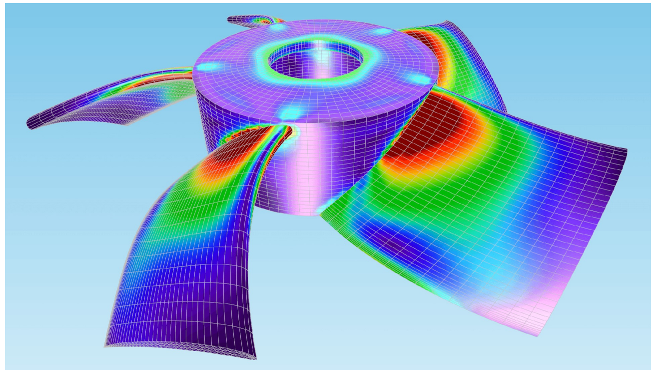


Jumper et al, Highly accurate protein structure prediction with AlphaFold, Nature (2021)

Recap – key scientific tasks

Simulation

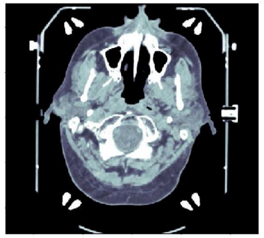
$$b = F(a)$$



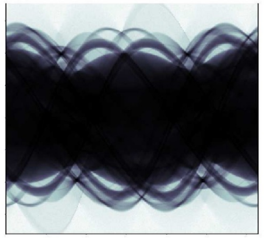
Mesh for finite element method
Source: COMSOL

Inverse problems

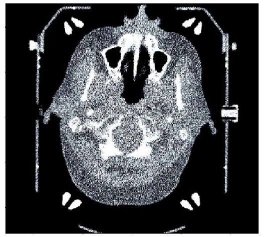
$$b = F(a)$$



Ground truth computed tomography image



Resulting tomographic data (sinogram)

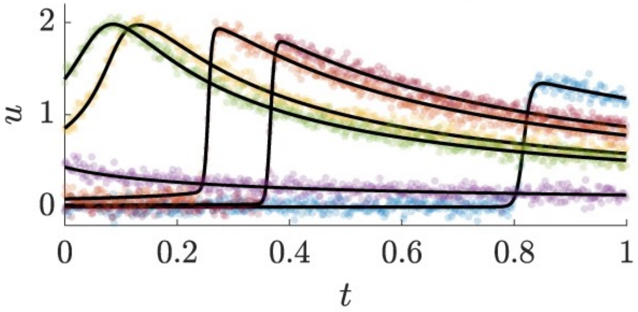


Result of inverse algorithm (filtered back-projection)

Adler et al, Solving ill-posed inverse problems using iterative deep neural networks, Inverse Problems (2017)

Equation discovery

$$b = F(a)$$

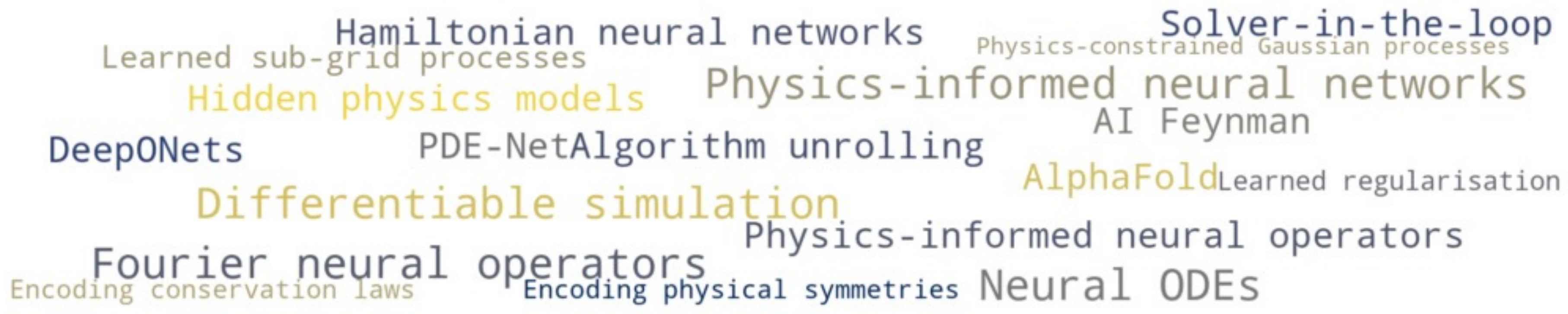


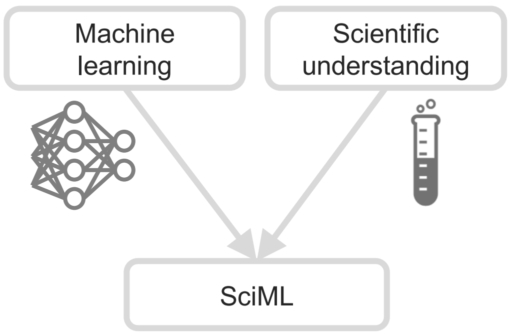
Ground truth: $u_t + uu_x - 0.0032u_{xx} = 0$

Discovered: $u_t + 1.002uu_x - 0.0032u_{xx} = 0$

Chen et al, Physics-informed learning of governing equations from scarce data, Nature communications (2021)

Recap – scientific machine learning





more **powerful,**
robust, interpretable
models

Course timeline

Tutorials		Lectures			
Mon 12:15-14:00 HG E 5		*Wed 08:15-10:00 ML H 44*		*Fri 12:15-13:00 ML H 44*	
19.02.		21.02.	~~Course introduction~~	23.02.	**Introduction to deep learning I**
26.02.	Introduction to PyTorch	28.02.	Introduction to deep learning II	01.03.	Importance of PDEs in science
04.03.	CNNs and surrogate modelling	06.03.	Introduction to physics-informed neural networks	08.03.	PINNs – limitations and extensions
11.03.	Implementing PINNs I	13.03.	PINNs – extensions and theory	15.03.	PINNs – theory
18.03.	Implementing PINNs II	20.03.	Introduction to operator learning	22.03.	DeepONets and spectral neural operators
25.03.	Operator learning I	27.03.	Fourier- and convolutional- neural operators	29.03.	
01.04.		03.04.		05.04.	
08.04.	Operator learning II	10.04.	Operator learning – limitations and extensions	12.04.	Introduction to transformers
15.04.		17.04.	Foundational models for operator learning	19.04.	Graph neural networks for PDEs
22.04.	GNNs	24.04.	GNNs for PDEs / introduction to diffusion models	26.04.	Introduction to diffusion models
29.04.	Transformers	01.05.		03.05.	Diffusion models - applications
06.05.	Diffusion models	08.05.	Introduction to differentiable physics	10.05.	Hybrid workflows
13.05.	Coding autodiff from scratch	15.05.	Neural differential equations	17.05.	Introduction to JAX
20.05.		22.05.	Symbolic regression and equation discovery	24.05.	Course summary and future trends
27.05.	Introduction to JAX / NDEs	29.05.	Guest lecture: ML in chemistry and biology	31.05.	Guest lecture: ML in chemistry and biology

Lecture overview

- What is deep learning?
  - Multilayer perceptrons
  - Universal approximation
- Popular deep learning tasks
  - Supervised learning
  - Unsupervised learning
- Training deep neural networks
  - Backpropagation & autodifferentiation

Lecture overview

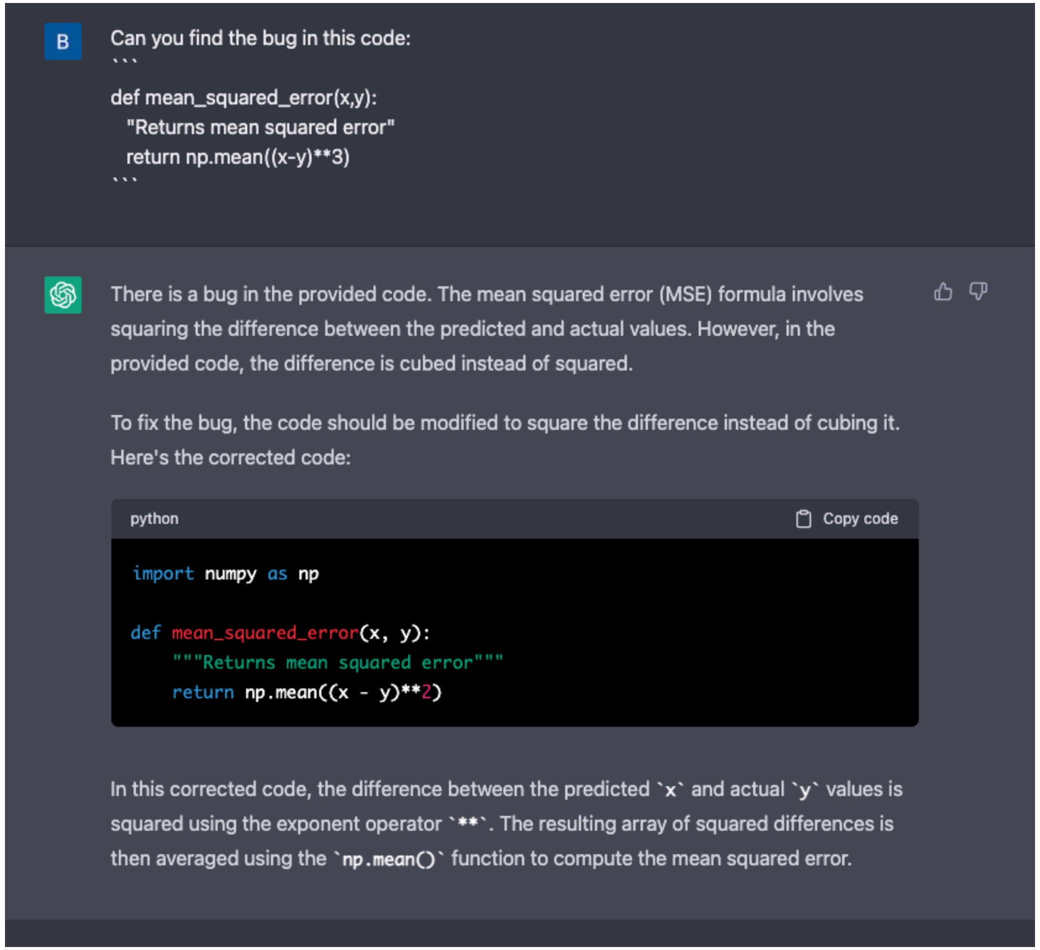
- What is deep learning?
  - Multilayer perceptrons
  - Universal approximation
- Popular deep learning tasks
  - Supervised learning
  - Unsupervised learning
- Training deep neural networks
  - Backpropagation & autodifferentiation

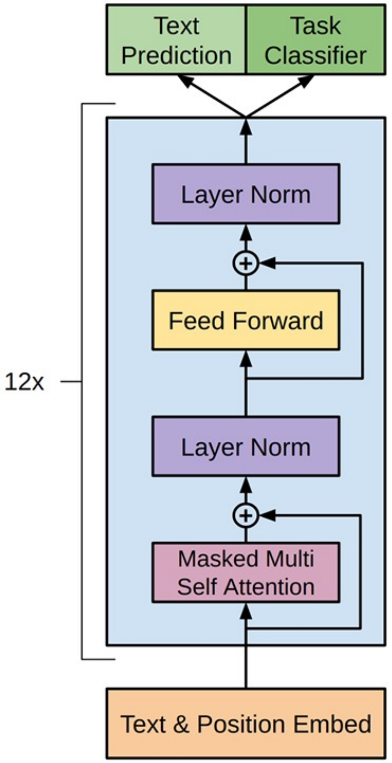
Learning objectives

- Be able to mathematically define a deep neural network
- Understand the typical tasks neural networks are used for
- Explain how neural networks are trained

State-of-the-art

- Inside ChatGPT – by end of these two lectures, you will understand how this works!





Radford et al, Improving Language Understanding by Generative Pre-Training, ArXiv (2018)
Brown et al, Language Models are Few-Shot Learners, NeurIPS (2020)

The rise of AI

You
how powerful is AI today?

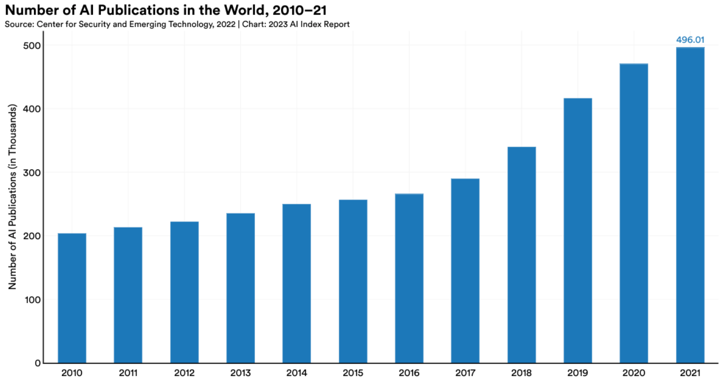
ChatGPT
As of my last update in January 2022, AI has made significant strides in various domains, demonstrating remarkable capabilities in tasks such as natural language processing, image recognition, speech synthesis, and more. Here are some aspects of AI's current power:

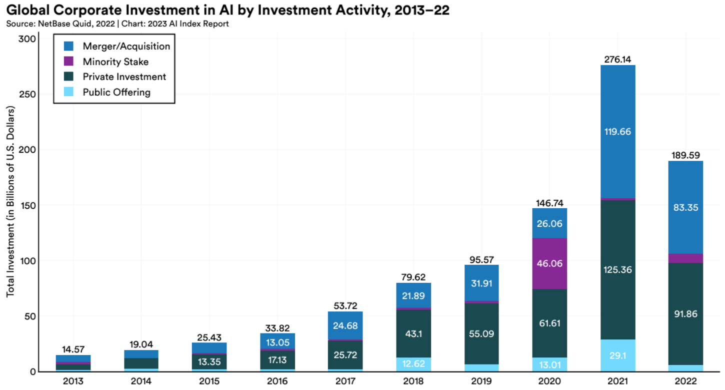
1. **Natural Language Processing (NLP)**: AI models like GPT-3 (the architecture I'm based on) have demonstrated the ability to generate human-like text, answer questions, summarize information, and even engage in basic forms of conversation. These models have advanced to the point where they can generate coherent and contextually relevant text across a wide range of topics.
2. **Computer Vision**: AI-powered systems are now capable of accurately recognizing and classifying objects, people, and activities in images and videos. They are used in various applications such as autonomous vehicles, medical imaging, surveillance, and augmented reality.
3. **Speech Recognition and Synthesis**: AI technologies can accurately transcribe spoken language into text and generate human-like speech. Virtual assistants like Siri, Alexa, and Google Assistant rely on these capabilities to understand and respond to user commands.

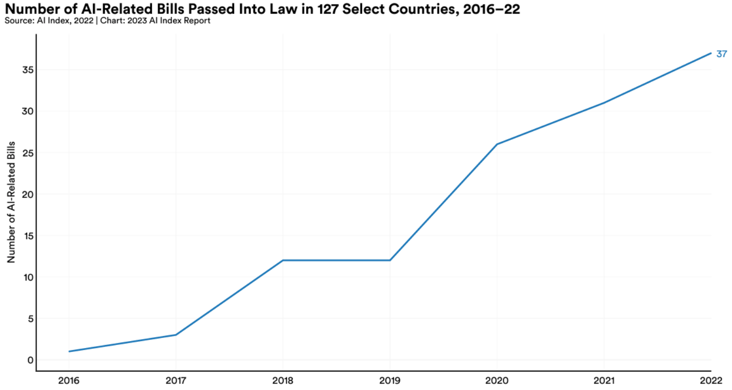
Brown et al, Language Models are Few-Shot Learners, NeurIPS (2020)

"a photograph of an astronaut riding a horse"

Source: Stable Diffusion Rombach et al, High-Resolution Image Synthesis with Latent Diffusion Models, CVPR (2022)







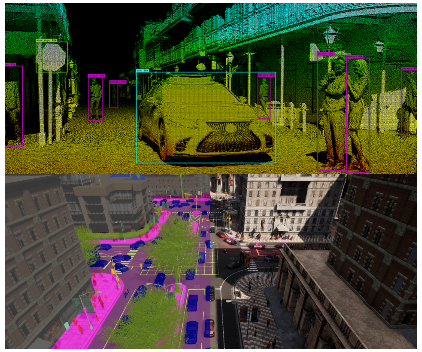
Source: AI Index Report, Stanford University

Reed et al., A Generalist Agent, TMLR (2022)

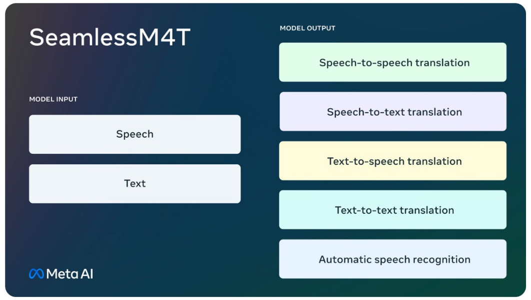
```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```

Copilot

Source: GitHub Copilot



Source: Machine Learning for Autonomous Driving Workshop, NeurIPS (2023)

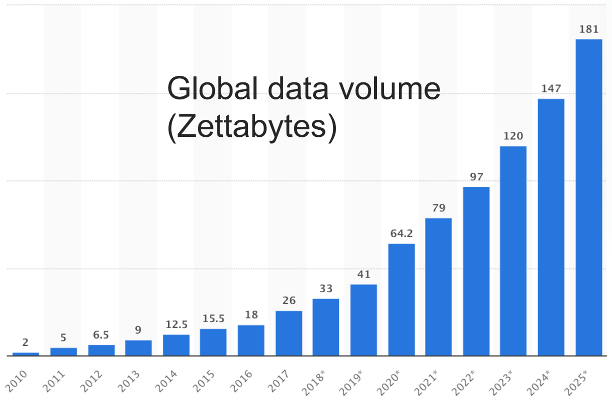


Barrault et al., SeamlessM4T: Massively Multilingual & Multimodal Machine Translation, ArXiv (2023)

Why now?

Neural networks date back to the 1950's – so why is deep learning so popular today?

Rapidly increasing amounts of data

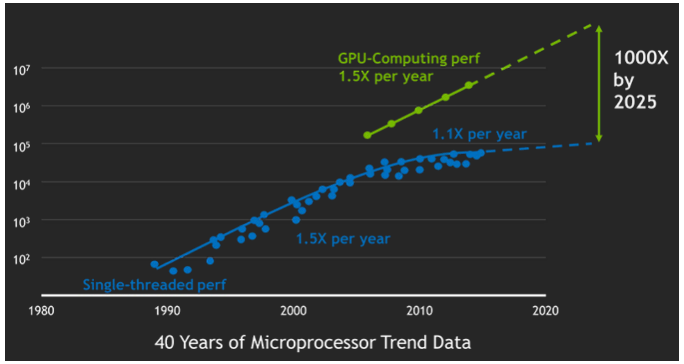


Source: Statista

IMAGENET

WIKIPEDIA
The Free Encyclopedia

Hardware improvements



Source: NVIDIA

- Graphical processing units (GPUs)
- Highly optimised for deep learning (massively parallel)

Software improvements

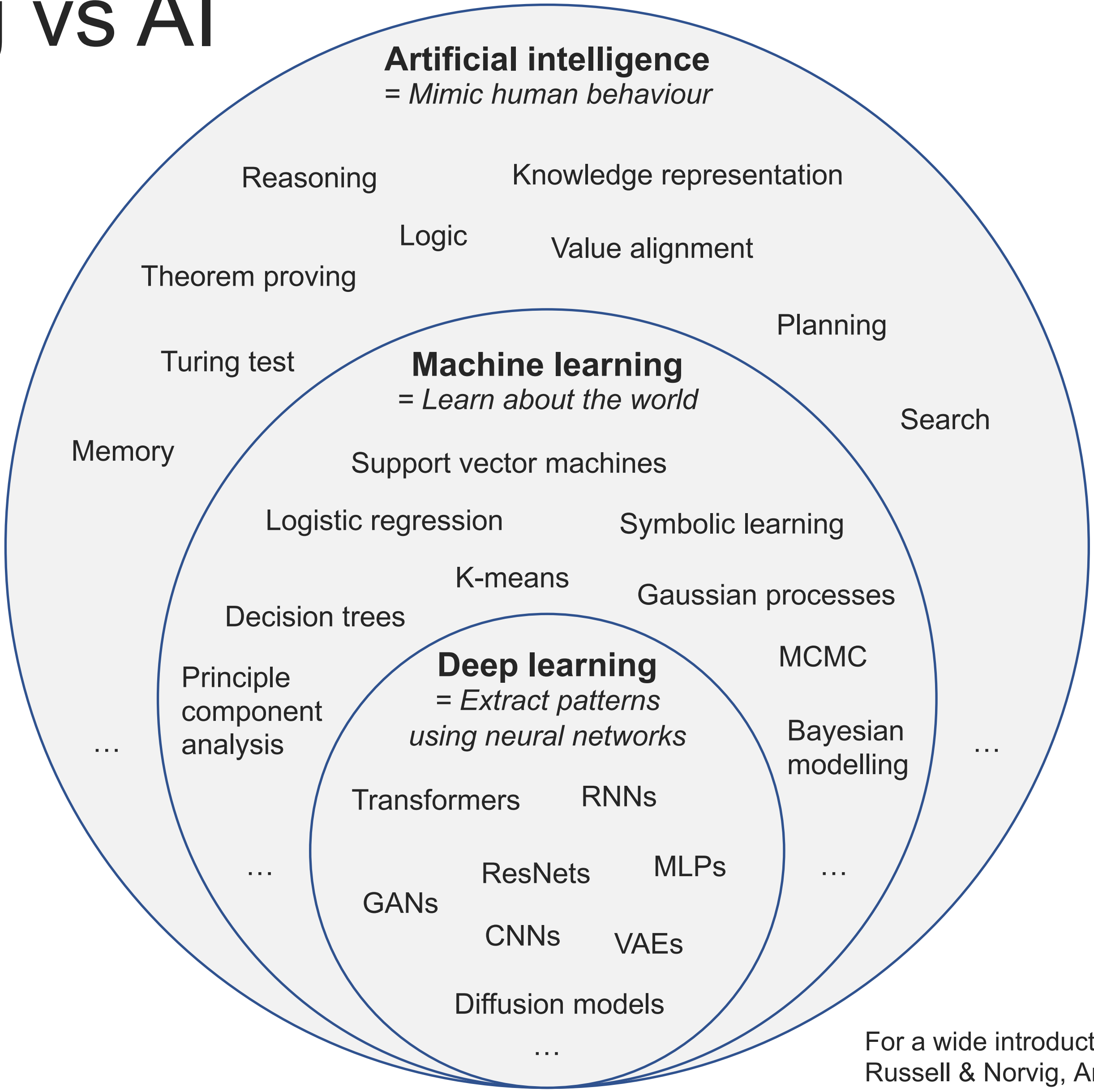
PyTorch

TensorFlow

Keras

- Mature deep learning frameworks
- Better training algorithms
- Deeper and more sophisticated architectures

Deep learning vs AI

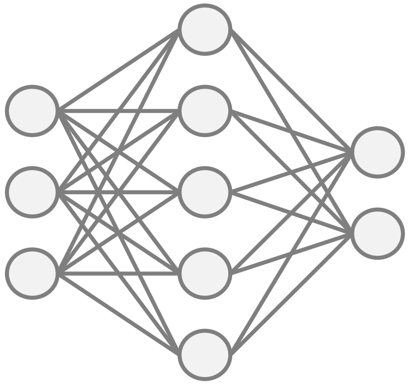


For a wide introduction to AI, see for example:
Russell & Norvig, Artificial Intelligence: A Modern Approach

What is a neural network?

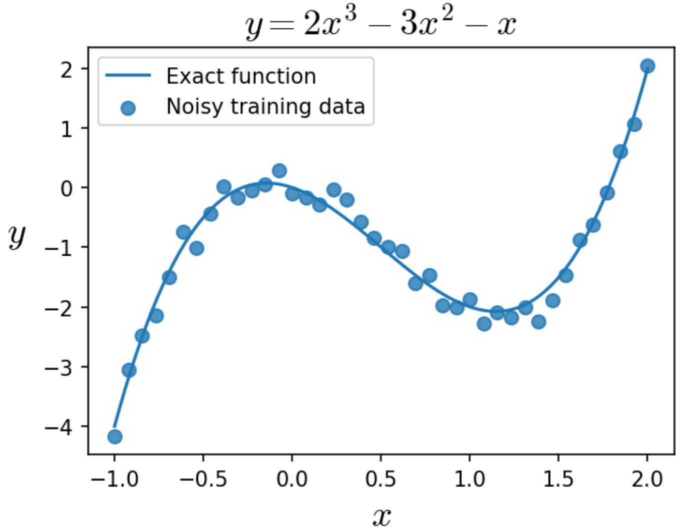
Neural networks are simply **flexible functions** fit to data

x



$\hat{y} = NN(x; \theta)$

Example dataset:



Goal: given training data, find a function (with flexible parameters θ) which approximates the true function,

$$\hat{y} = NN(x; \theta) \approx y(x)$$

Function fitting

Simple polynomial regression

$$\hat{y}(x;\theta) = \theta_4 x^3 + \theta_3 x^2 + \theta_2 x + \theta_1$$

To fit, use least-squares:

$$\theta^* = \min_{\theta} \sum_i^N (\hat{y}(x_i;\theta) - y_i)^2 \qquad (1)$$

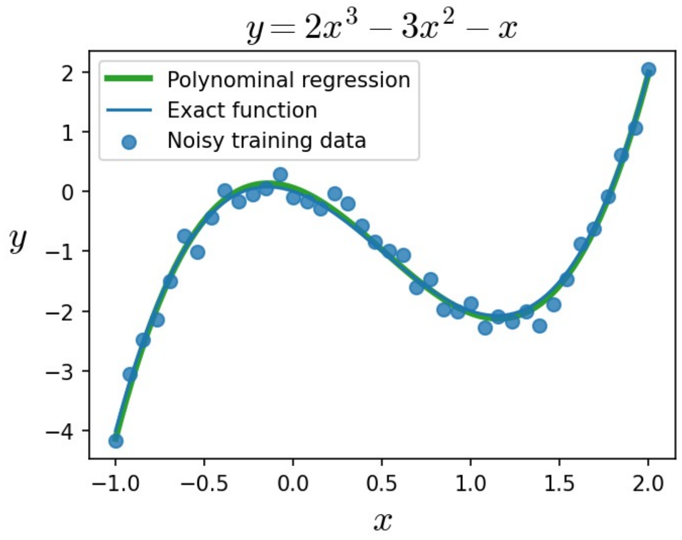
Re-write using linear algebra:

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \dots \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix} \text{ or } \hat{Y} = \Phi^T \theta$$

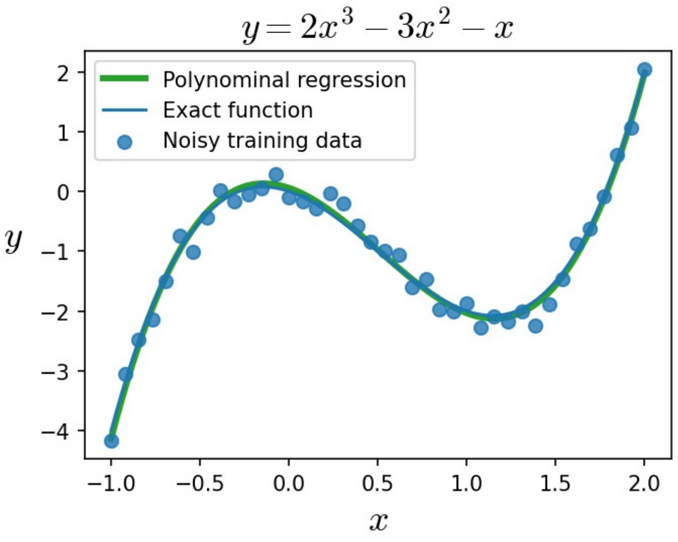
$$\theta^* = \min_{\theta} \|\Phi^T \theta - Y\|^2$$

In this case, it can be shown (1) has an analytical solution:

$$\theta^* = (\Phi^T \Phi)^{-1} \Phi^T Y$$



Function fitting



Simple polynomial regression

$$\hat{y}(x;\theta) = \theta_4 x^3 + \theta_3 x^2 + \theta_2 x + \theta_1$$

To fit, use least-squares:

$$\theta^* = \min_\theta \sum_i^N (\hat{y}(x_i;\theta) - y_i)^2 \qquad (1)$$

Re-write using linear algebra:

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \ldots \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix} \text{ or } \hat{Y} = \Phi^T\theta$$

$$\theta^* = \min_\theta \|\Phi^T\theta - Y\|^2$$

In this case, it can be shown (1) has an analytical solution:

$$\theta^* = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Neural network regression

$$\hat{y}(x;\theta) = NN(x;\theta)$$

To fit, use least-squares:

$$\theta^* = \min_\theta \sum_i^N (NN(x_i;\theta) - y_i)^2 \qquad (2)$$

In general, no analytical solution to (2) exists, so we must use **optimisation**

For example, gradient descent:

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial \sum_i^N (NN(x_i;\theta) - y_i)^2}{\partial \theta_j}$$

or equally

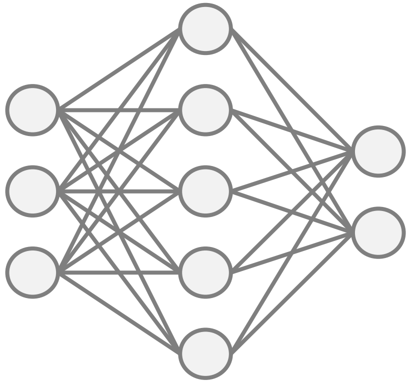
$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial L(\theta)}{\partial \theta_j}$$

Where γ is the learning rate and $L(\theta)$ is the **loss function**

Neural network architecture

So, what exactly is $\hat{y} = NN(x; \theta)$?

This depends on the network **architecture** you choose (CNN, ResNet, Transformer, … etc)



2-layer MLP

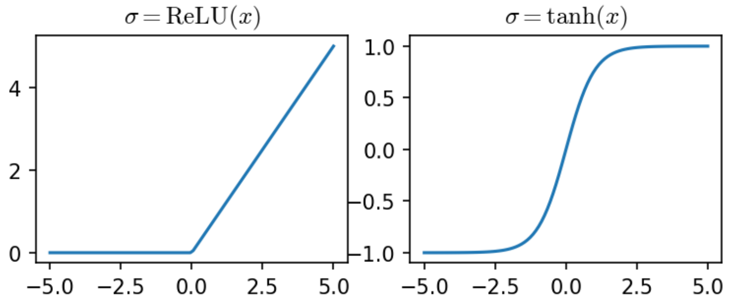
The most basic architecture is the **multilayer perceptron** (MLP) (aka **fully connected network**)

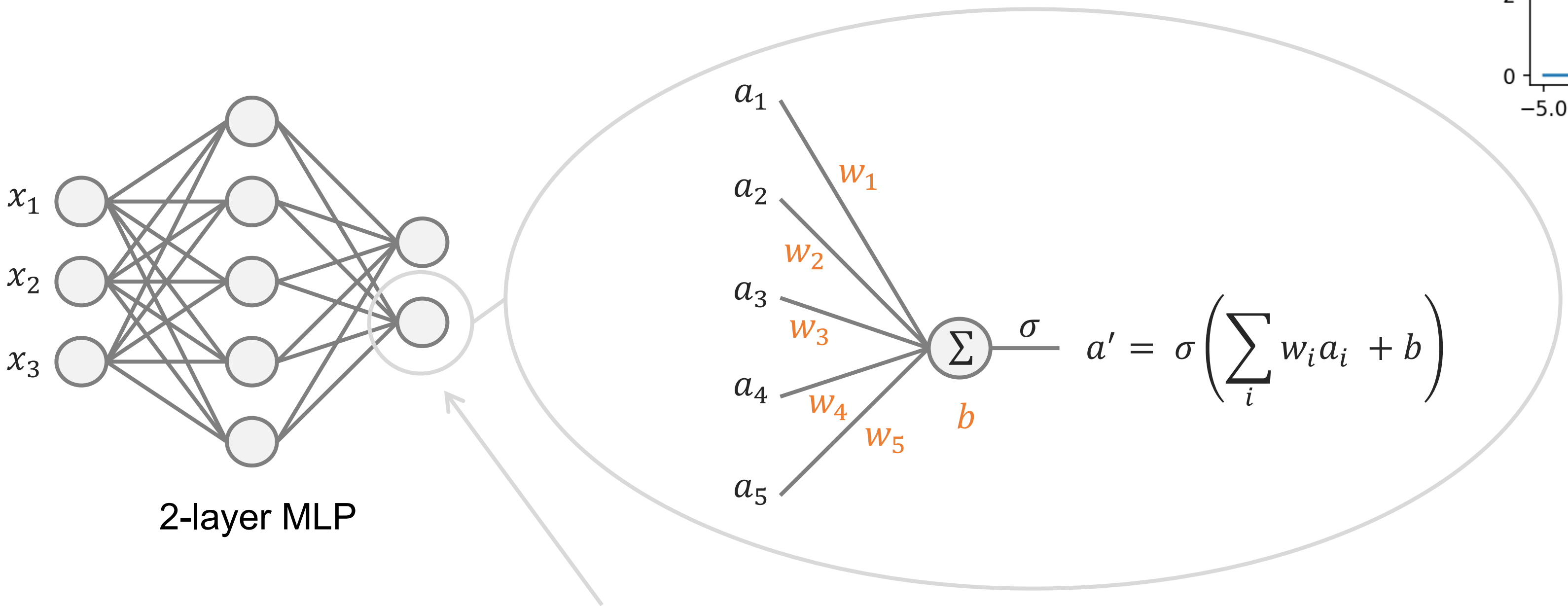
For example, a 2-layer MLP is defined as:

$$NN(x; \theta) = W_2 \sigma(W_1 x + b_1) + b_2$$

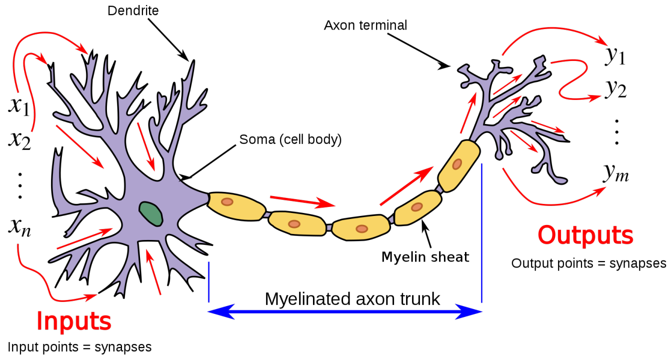
Where x is an input vector, W_1 and W_2 are learnable weight matrices, b_1 and b_2 are learnable bias vectors, and σ is an activation function, for example, $\sigma = \tanh(\cdot)$

Biological inspiration





2-layer MLP



Biological neuron
(Source: Wikipedia)

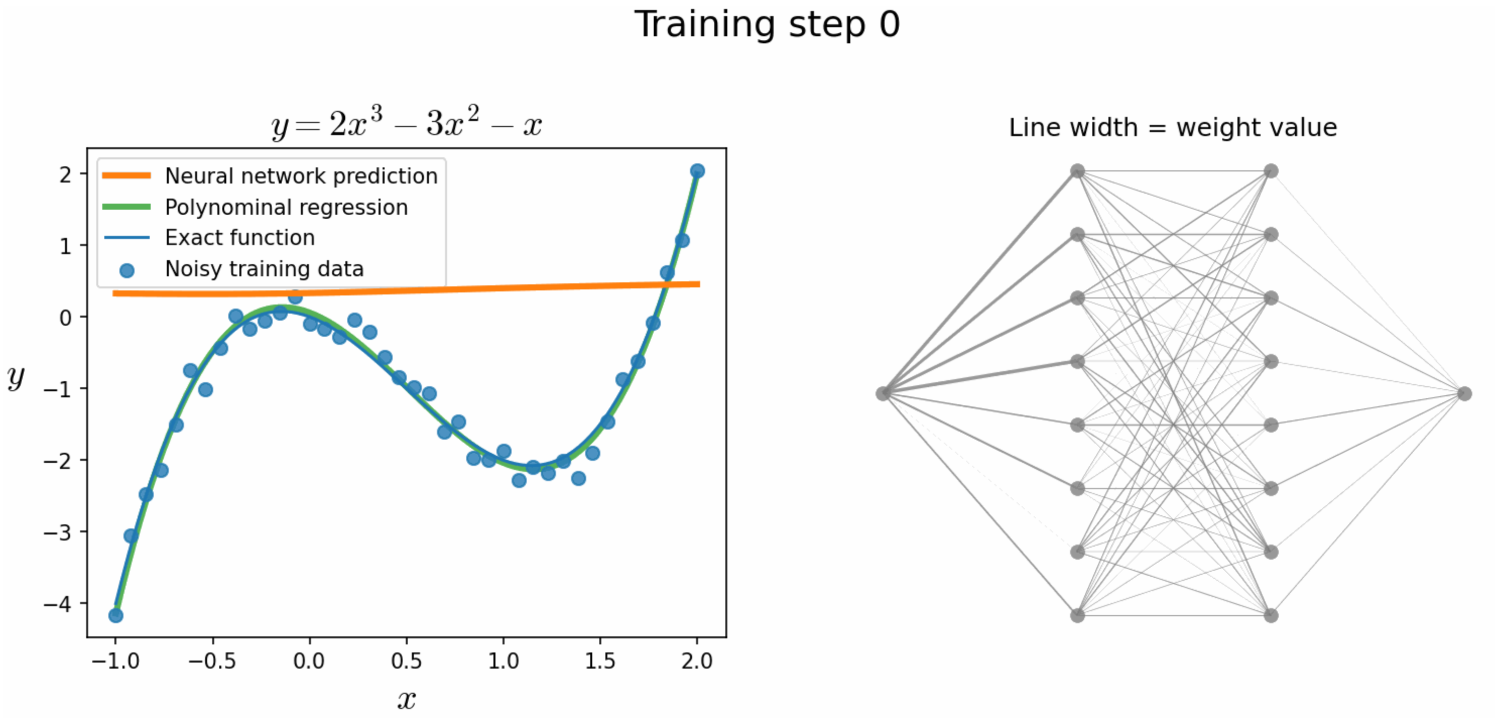
For last layer:

$$\begin{pmatrix} a_1' \\ a_2' \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)$$

Entire network:

$$NN(\boldsymbol{x};\theta) = \sigma(W_2\sigma(W_1\boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2}) = \boldsymbol{f} \circ \boldsymbol{g}\ (\boldsymbol{x};\theta)$$

Polynomial regression example



$$NN(x;\theta) = W_3(\sigma(W_2\sigma(W_1x + b_1) + b_2) + b_3$$

Trained using gradient descent

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial \sum_i^N (NN(x_i;\theta) - y_i)^2}{\partial \theta_j}$$

Universal approximation

So why not just use linear regression?

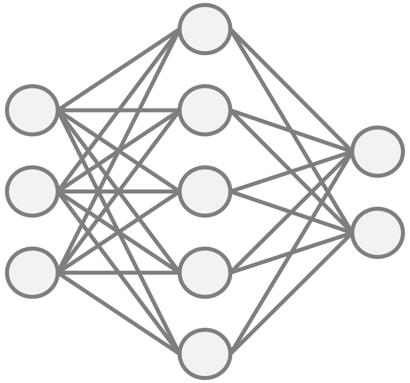
Universal approximation

So why not just use linear regression?

Neural networks are simply **flexible functions** fit to data

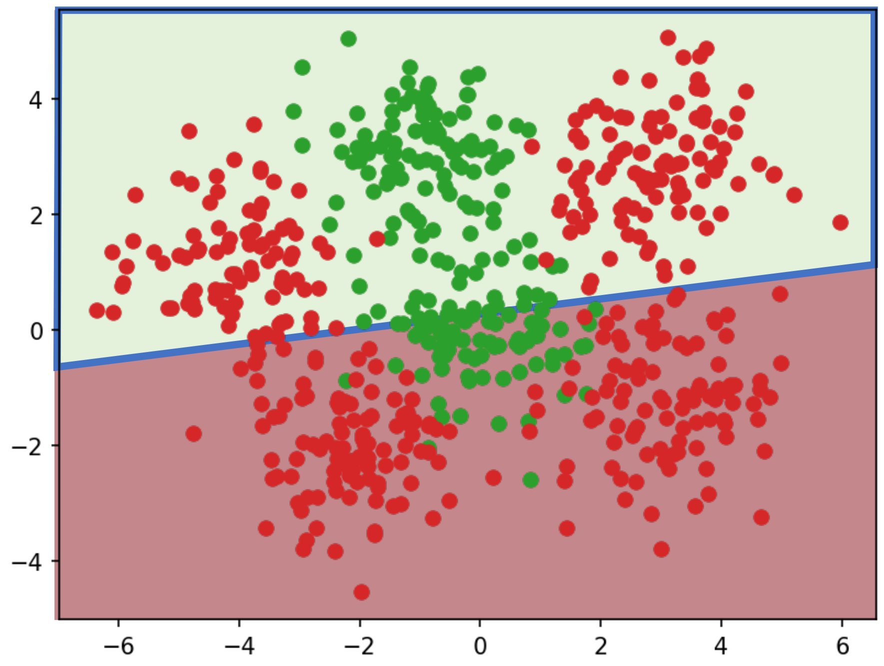
With enough parameters, neural networks can approximate any* arbitrarily complex function
= **universal approximation**

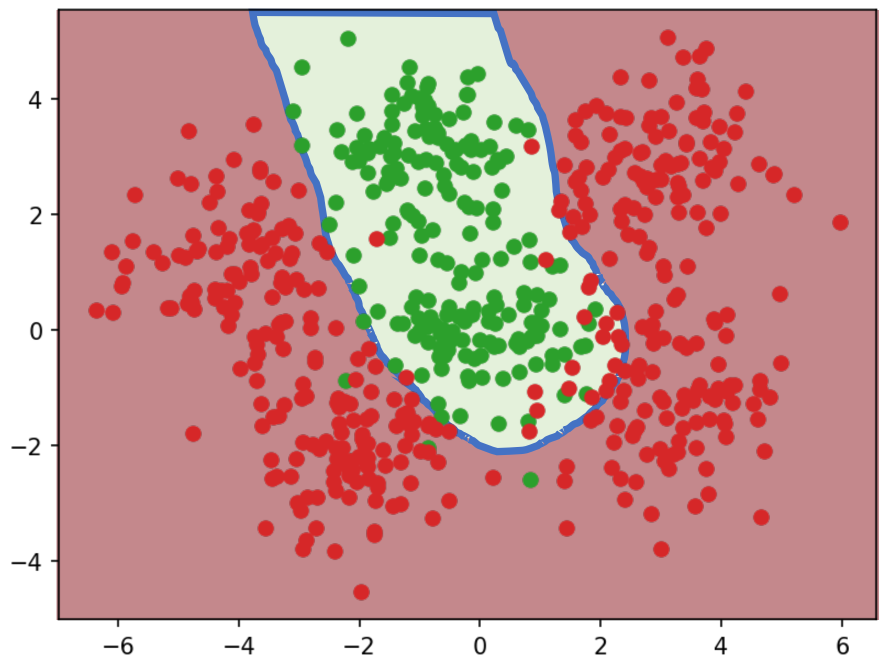
x = array of RGB values



$\hat{y} = P(\text{dog} \mid x) = 1$

Importance of activation functions

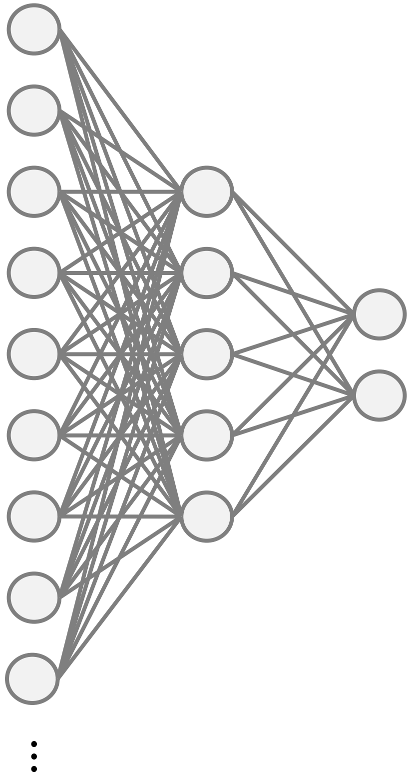




Non-linearities allow us to approximate arbitrary **non-linear** functions

MLPs use lots of parameters

=> Flatten to 1D =>



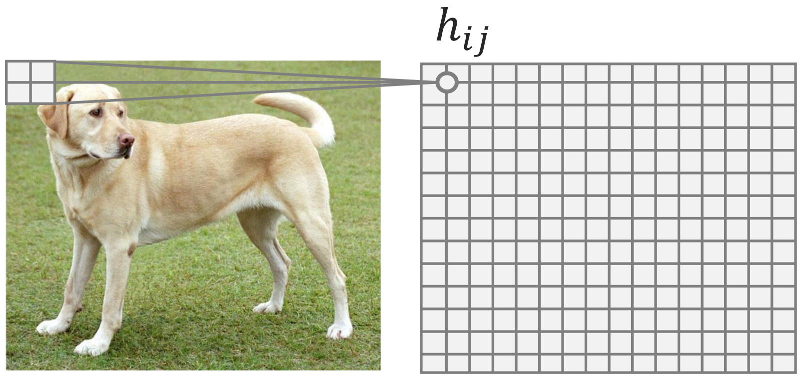
$$NN(x;\theta) = W_3(\sigma(W_2\sigma(W_1 x + b_1) + b_2) + b_3$$

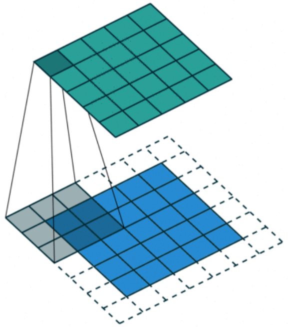
Assume the image has shape 128 x 128, and we have 100 hidden units in the first layer, then W_1 has shape (100 x (128 x 128)) = (100 x 16,384)

= 1.6M parameters!

=> A simple MLP image classifier can have millions of parameters

Convolutional neural network (CNN)





$$NN(x;\theta) = W_3 \star (\sigma(W_2 \star \sigma(\underbrace{W_1 \star x + b_1}_{h}) + b_2) + b_3$$

$$h_{ij} = \sum_{i'}^{l} \sum_{j'}^{m} W_{i'j'} x_{i+i',j+j'} + b$$

Convolutional neural networks honor the **spatial correlations** in their inputs

Each neuron;

- Has a **limited** field of view
- **Shares** the same weights as the other neurons in the layer
- Mathematically, CNNs use cross-correlation

CNNs have translation equivariance (an inductive bias)

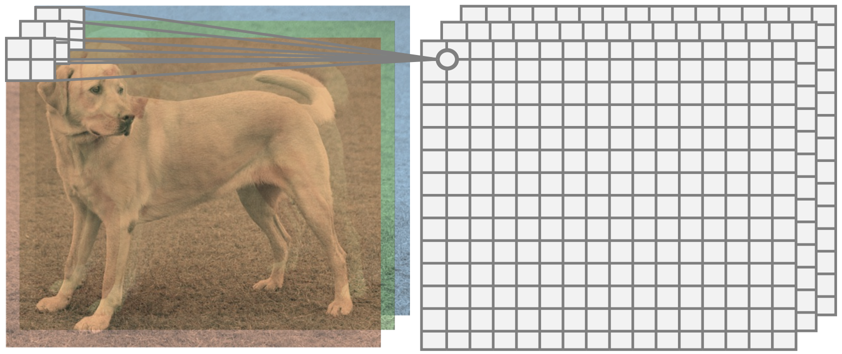
Let the size of the convolutional filter be 3 x 3

Then W_1 has shape (3 x 3)

= 9 parameters! (much, much smaller than a MLP)

Image source:
github/vdumoulin/conv_arithmetic

Convolutional neural network (CNN)



Then the convolutional layer is defined by:

$$h_{ijc} = \sum_{i'}^{l} \sum_{j'}^{m} \sum_{c'}^{C_{\text{in}}} W_{i'j'c'c}\, x_{i+i',j+j',c'} + b_c$$

In practice, CNNs are usually extended so they can have multiple **channels** in the inputs and outputs of each layer

e.g. (R,G,B) image as input, where each channel is a color

Also:
- 1D and 3D CNNs follow analogously
- And we can add dilations and strides too

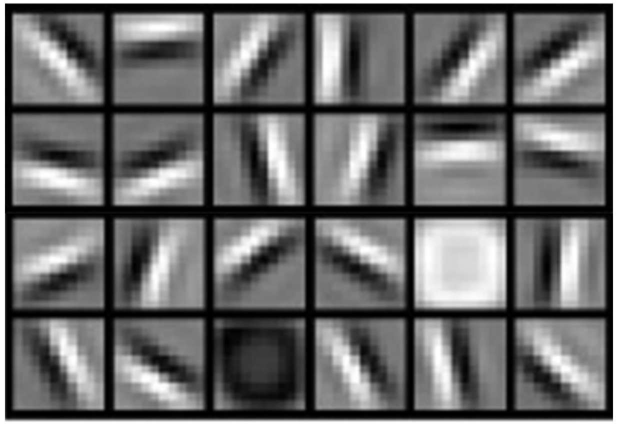
Let the size of the convolutional filter be 3 x 3

Then W has shape (3 x 3 x C_{in} x C_{out})

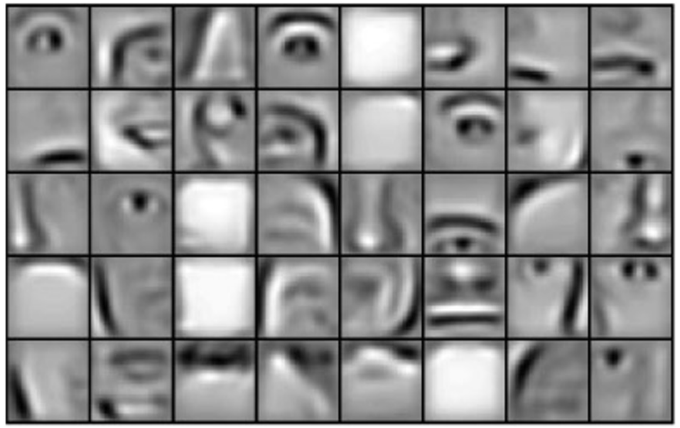
= 81 parameters for 3 input and 3 output channels

Deep CNNs

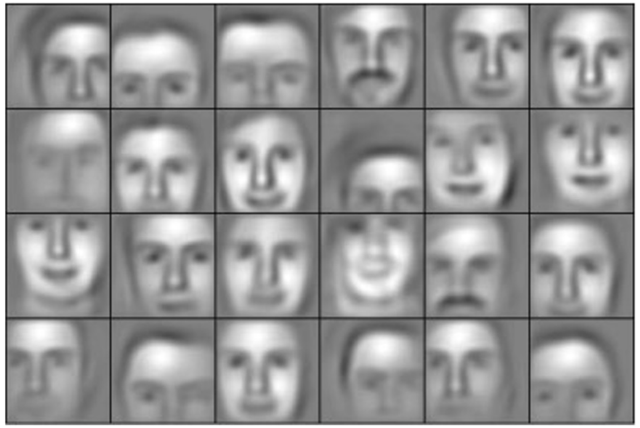
First layer



Second layer



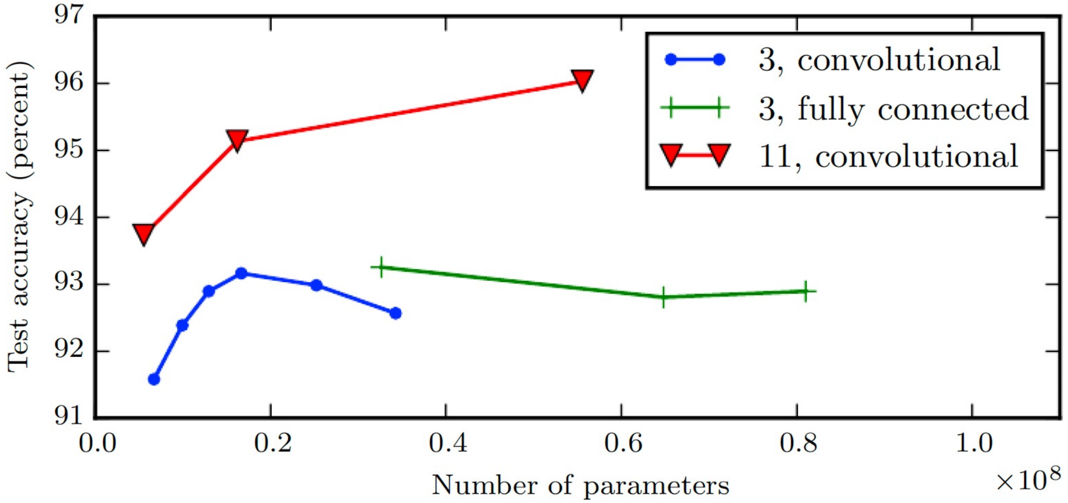
Third layer



Deep CNNs learn **hierarchical** features

Lee et al, Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks, Communications of the ACM (2011)

Depth is key



Goodfellow et al, Multi-digit number recognition from street view imagery using deep convolutional neural networks, ICLR (2014)

Empirically, deep neural networks perform better than shallow neural networks

=> encode a very general belief that the true function is **composed** of simpler functions

Popular deep learning tasks

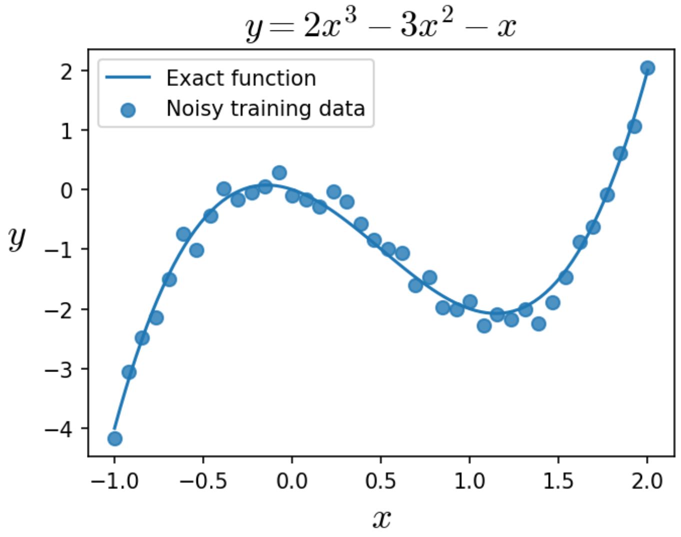
Popular deep learning tasks

- Supervised learning
  - Regression
  - Classification

- Unsupervised learning
  - Feature learning
  - Autoregression
  - Generative models

- …but in all cases, the neural network is still a function fit to data!

Supervised learning - regression



Supervised learning - regression:

Given a set of example inputs and outputs (labels) $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ from some true function $y(x)$ where $x \in \mathbb{R}^n, y \in \mathbb{R}^m$

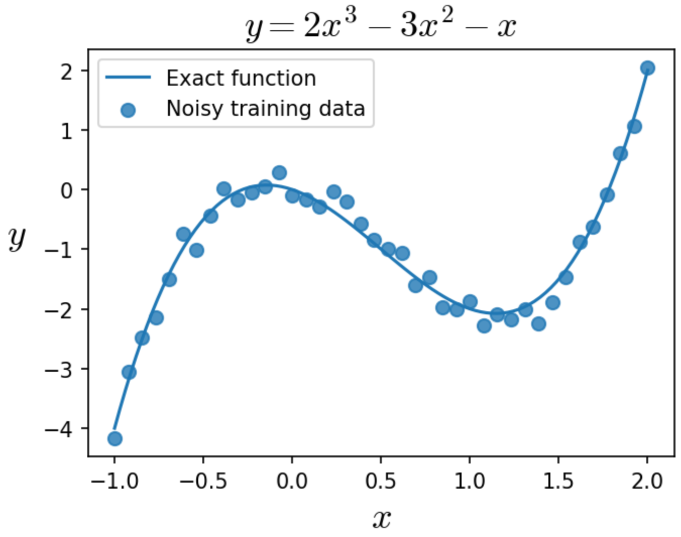
Find

$$\hat{y} = NN(x; \theta) \approx y(x)$$

Loss function (mean squared error)

$$L(\theta) = \frac{1}{N} \sum_{i}^{N} (NN(x_i; \theta) - y_i)^2$$

Supervised learning - regression



Supervised learning - regression:

Given a set of example inputs and outputs (labels) $\{(x_1, y_1), \dots, (x_N, y_N)\}$ from some true function $y(x)$ where $x \in \mathbb{R}^n, y \in \mathbb{R}^m$

Find

$$\hat{y} = NN(x; \theta) \approx y(x)$$

Probabilistic perspective:

Given a set of example inputs and outputs (labels) $\{(x_1, y_1), \dots, (x_N, y_N)\}$ drawn from the probability distribution $p(y|x)$

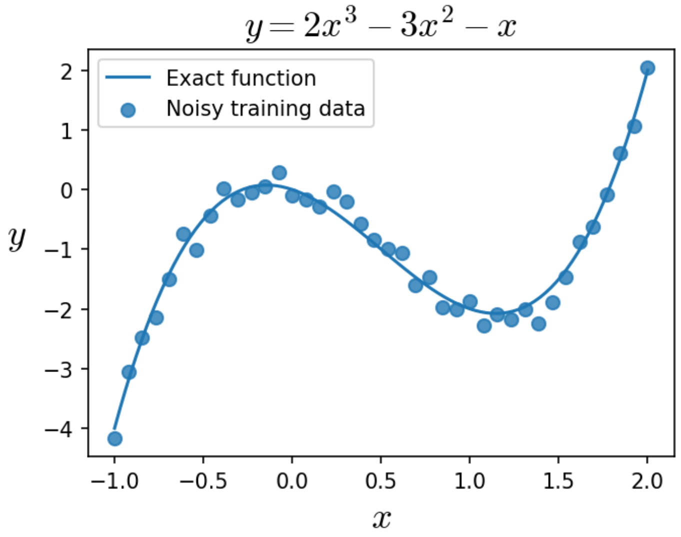
Find

$$\hat{p}(y|x, \theta) \approx p(y|x)$$

Loss function (mean squared error)

$$L(\theta) = \frac{1}{N} \sum_{i}^{N} (NN(x_i; \theta) - y_i)^2$$

Supervised learning - regression



Loss function (mean squared error)

$$L(\theta) = \frac{1}{N}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$

Probabilistic perspective:

Assume $\hat{p}(y|x,\theta)$ is a **normal** distribution:

$$\hat{p}(y|x,\theta) = \mathcal{N}(y; \mu = NN(x;\theta), \sigma = 1)$$

Then, assume each training datapoint is independently and identically distributed (iid), then the **data likelihood** can be written as:

$$\hat{p}(D|\theta) = p(x_1, y_1, \ldots, x_N, y_N|\theta) = \prod_{i}^{N}\hat{p}(y_i|x_i,\theta)$$

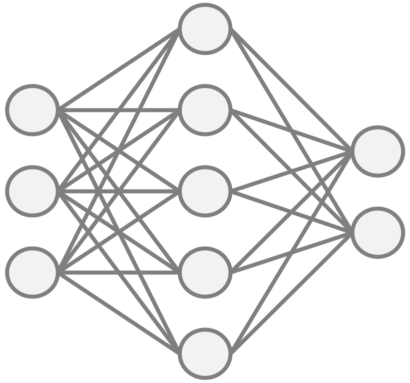
Then use **maximum likelihood estimation** (MLE) to estimate θ^*:

$$\theta^* = \max_{\theta}\hat{p}(D|\theta)$$

$$= \max_{\theta}\prod_{i}^{N} e^{-\frac{1}{2}\left(\frac{y_i - NN(x_i;\theta)}{1}\right)^2}$$

$$= \min_{\theta}\sum_{i}^{N}(NN(x_i;\theta) - y_i)^2$$

Supervised learning - classification



$P(\text{dog} \mid x) = 0.99$

$P(\text{cat} \mid x) = 0.01$

Supervised learning - classification:

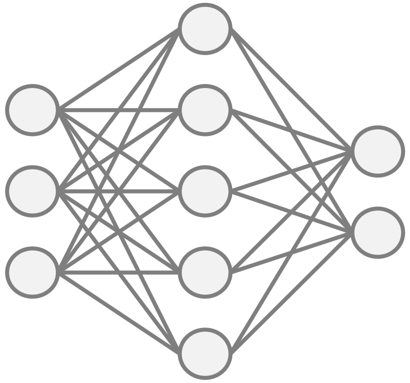
Given a set of example inputs and outputs (labels) $\{(x_1, y_1), \dots, (x_N, y_N)\}$ drawn from the discrete probability distribution $P(y|x)$

where $y \in Y$, for example, $Y = \{\text{dog}, \text{cat}\}$

Find

$$\hat{P}(y|x, \theta) \approx P(y|x)$$

Supervised learning - classification



$P(\text{dog} \mid x) = 0.99$

$P(\text{cat} \mid x) = 0.01$

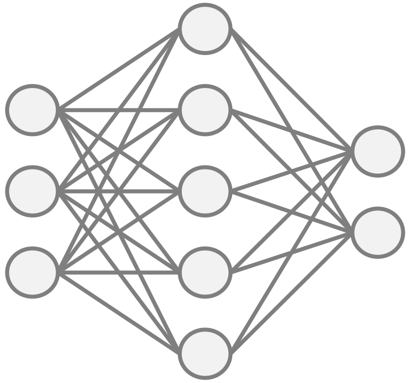
Let each class be encoded as a one-hot vector of length C, e.g

$y = (0,1)$ (dog) or
$y = (1,0)$ (cat)

Then assume

$$\hat{P}(y|x,\theta) = \prod_{j}^{C} NN(x;\theta)_j^{y_j}, \qquad \sum_{j}^{C} NN(x;\theta)_j = 1$$

Supervised learning - classification



$P(\text{dog} \mid x) = 0.99$

$P(\text{cat} \mid x) = 0.01$

Let each class be encoded as a one-hot vector of length C, e.g

$y = (0{,}1)$ (dog) or
$y = (1{,}0)$ (cat)

Then assume

$$\hat{P}(y|x,\theta) = \prod_{j}^{C} NN(x;\theta)_j^{y_j}, \qquad \sum_{j}^{C} NN(x;\theta)_j = 1$$

Then, assume each training datapoint is independently and identically distributed (iid), then the **data likelihood** can be written as:

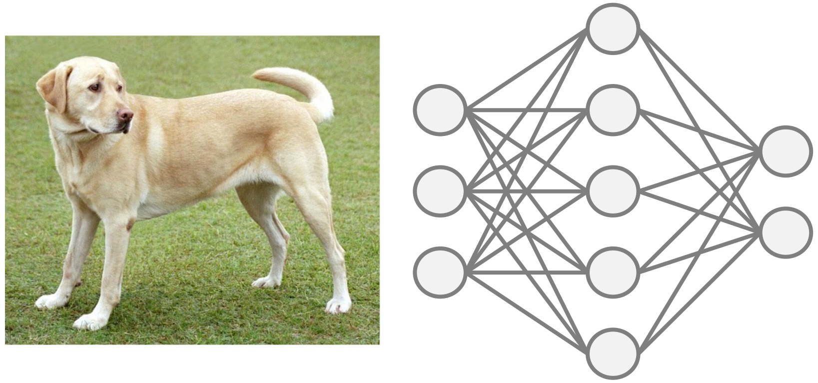
$$\hat{P}(D|\theta) = \hat{P}(x_1, y_1, \dots, x_n, y_n|\theta) = \prod_{i}^{N} \hat{P}(y_i|x_i,\theta)$$

Then use **maximum likelihood estimation** (MLE) to estimate θ^*:

$$\begin{aligned}\theta^* &= \max_{\theta} \hat{P}(D|\theta) \\ &= \max_{\theta} \prod_{i}^{N} \prod_{j}^{C} NN(x_i;\theta)_j^{y_{ij}} \\ &= \min_{\theta} -\sum_{i}^{N}\sum_{j}^{C} y_{ij} \log NN(x_i;\theta)_j\end{aligned}$$

Also known as the **cross-entropy loss**

Supervised learning - classification



Let each class be encoded as a one-hot vector of length C, e.g

$$y = (0,1) \quad \text{(dog) or}$$
$$y = (1,0) \quad \text{(cat)}$$

Typically, we use a softmax output layer to assert $\sum_j^C NN(x;\theta)_j = 1$;

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j^C e^{z_j}}$$

Then assume

$$\hat{P}(y|x,\theta) = \prod_j^C NN(x;\theta)_j^{y_j}, \qquad \sum_j^C NN(x;\theta)_j = 1$$

Then, assume each training datapoint is independently and identically distributed (iid), then the **data likelihood** can be written as:

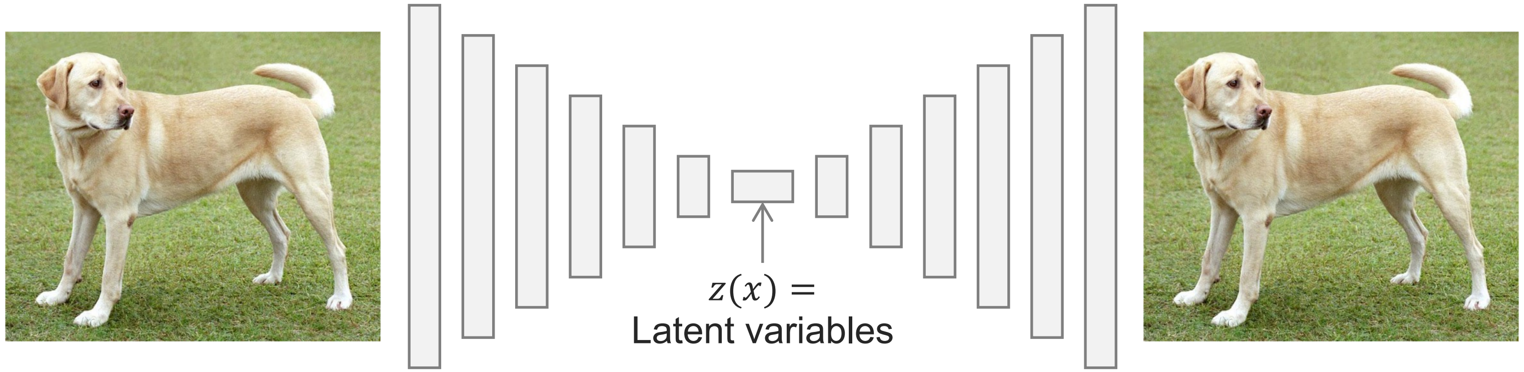
$$\hat{P}(D|\theta) = \hat{P}(x_1, y_1, \dots, x_n, y_n|\theta) = \prod_i^N \hat{P}(y_i|x_i,\theta)$$

Then use **maximum likelihood estimation** (MLE) to estimate θ^*:

$$\theta^* = \max_\theta \hat{P}(D|\theta)$$
$$= \max_\theta \prod_i^N \prod_j^C NN(x_i;\theta)_j^{y_{ij}}$$
$$= \min_\theta - \sum_i^N \sum_j^C y_{ij} \log NN(x_i;\theta)_j$$

Also known as the **cross-entropy loss**

Unsupervised learning - feature learning



Loss function

Many different possibilities, a simple choice is

$$L(\theta) = \sum_{i}^{N} (NN(x_i; \theta) - x_i))^2$$

Unsupervised learning – feature learning

Given a set of examples $\{x_1, \ldots, x_N\}$, find some features $z(x)$

Which are salient descriptors of $x,$ where $x \in \mathbb{R}^n, z \in \mathbb{R}^d$

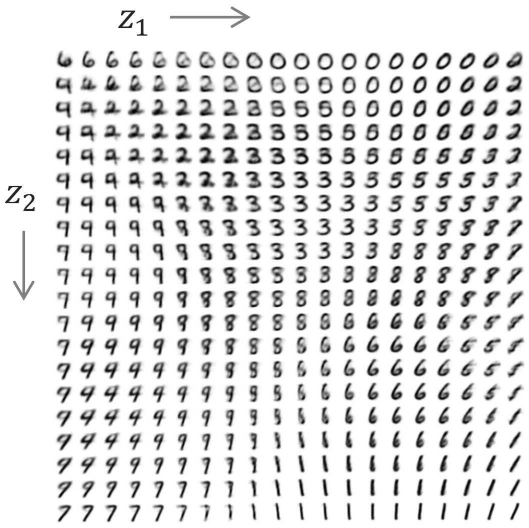
Typically, $d \ll n$ (= compression)

z can be used for downstream tasks, e.g. clustering / classification

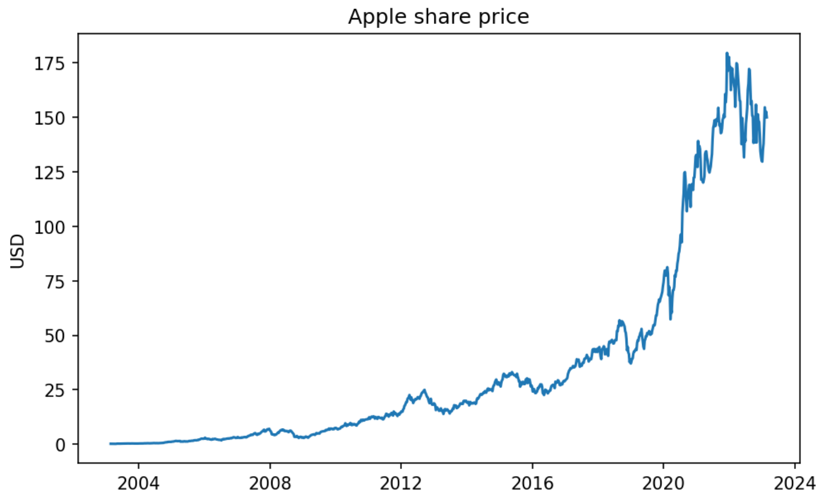
For example:

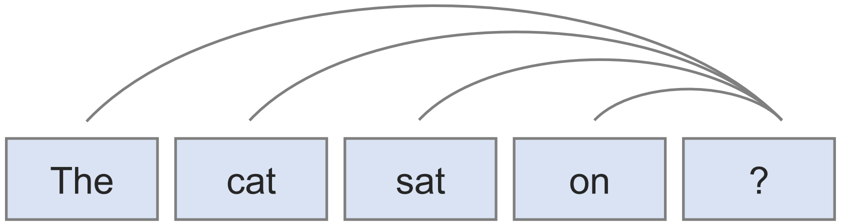
Variational autoencoders (VAEs)

Kingma et al, 2014



Unsupervised learning - autoregression

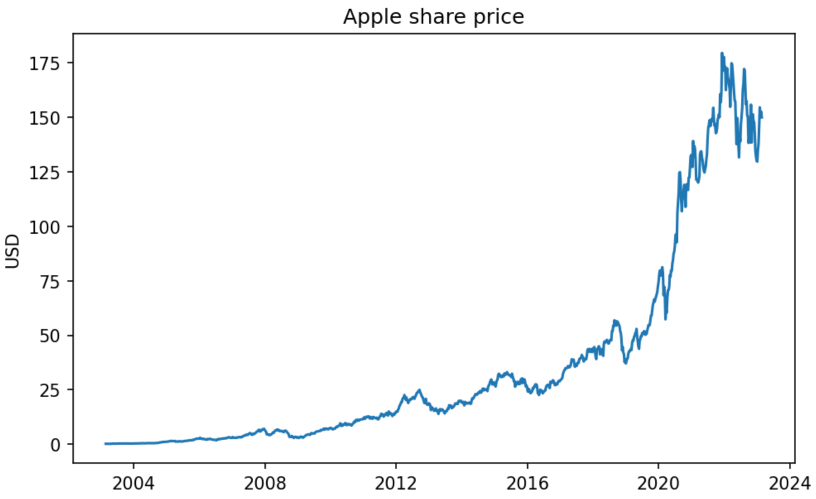


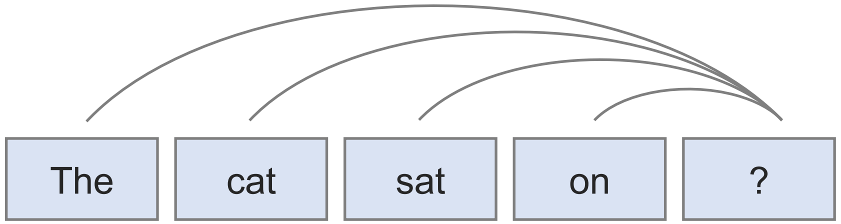


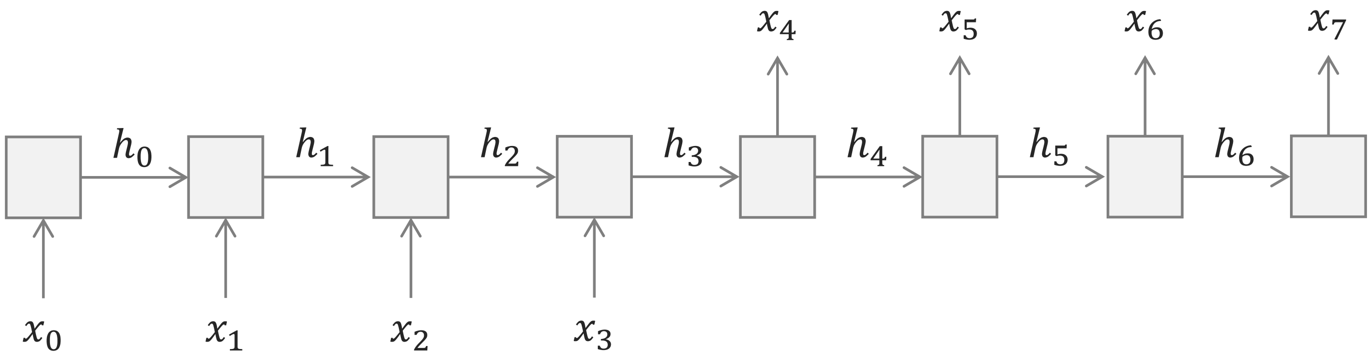
Unsupervised learning – autoregression

Given many examples sequences, train a model to predict future values from past values

Unsupervised learning - autoregression

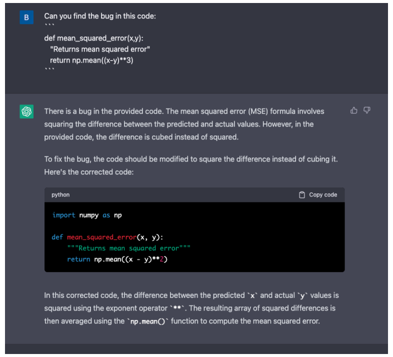






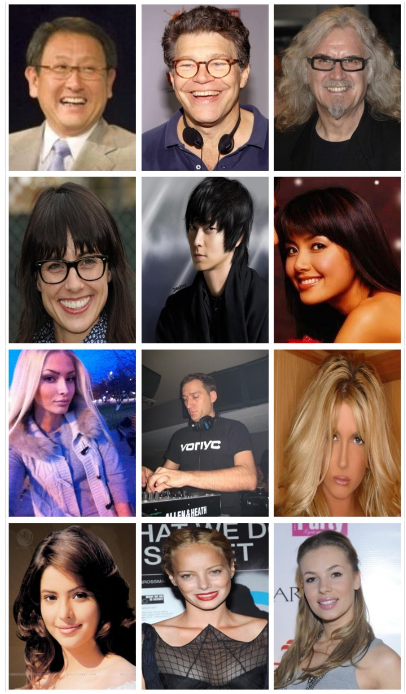
For example:

ChatGPT



Unsupervised learning - generative modelling

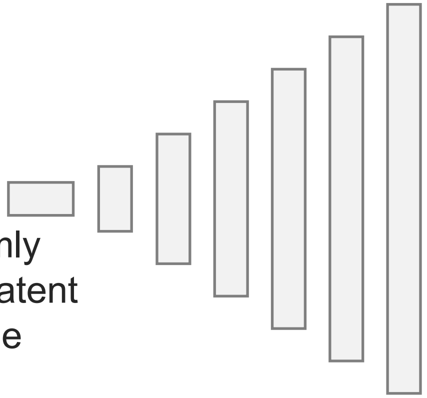
Training dataset



Source: CelebA

Generative model

$z =$ Randomly sampled latent variable

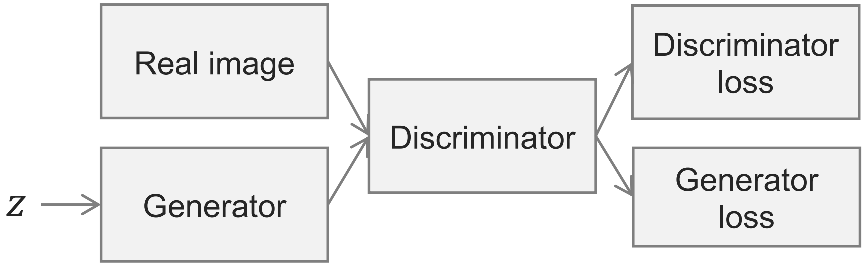


$x =$ generated image

For example:

Generative adversarial networks (GANs)

Goodfellow et al, 2014



Unsupervised learning – generative modelling

Given many examples $\{x_1, \ldots, x_N\}$ sampled from some distribution $p(x)$, learn to sample from $p(x)$

Training deep neural networks

How do we train neural networks?

Gradient descent

$$\hat{y}(x; \theta) = NN(x; \theta)$$

To fit, use least-squares:

$$\theta^* = \min_{\theta} \sum_i^N (NN(x_i; \theta) - y_i)^2 \qquad (2)$$

In general, no analytical solution to (2) exists, so we must use **optimisation**

For example, gradient descent:

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial \sum_i^N (NN(x_i; \theta) - y_i)^2}{\partial \theta_j}$$

or equally

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial L(\theta)}{\partial \theta_j}$$

Where γ is the learning rate and $L(\theta)$ is the **loss function**

How do we train neural networks?

Gradient descent

$$\hat{y}(x;\theta) = NN(x;\theta)$$

To fit, use least-squares:

$$\theta^* = \min_{\theta} \sum_i^N (NN(x_i;\theta) - y_i)^2 \qquad (2)$$

In general, no analytical solution to (2) exists, so we must use **optimisation**

For example, gradient descent:

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial \sum_i^N (NN(x_i;\theta) - y_i)^2}{\partial \theta_j}$$

or equally

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial L(\theta)}{\partial \theta_j}$$

Where γ is the learning rate and $L(\theta)$ is the **loss function**

Note that

$$\frac{\partial L(\theta)}{\partial \theta_j} = \sum_i^N 2(NN(x_i;\theta) - y_i)\frac{\partial NN(x_i;\theta)}{\partial \theta_j}$$

Let's consider a fully connected network

$$NN(\boldsymbol{x};\theta) = W_3(\sigma(\overbrace{W_2\sigma(\underbrace{W_1\boldsymbol{x} + \boldsymbol{b_1}}_{\boldsymbol{h}}) + \boldsymbol{b_2}}^{\boldsymbol{g}}) + \boldsymbol{b_3} = \boldsymbol{f} \circ \boldsymbol{g} \circ \boldsymbol{h}(\boldsymbol{x};\theta)$$

How do we calculate $\frac{\partial NN(\boldsymbol{x}_i;\theta)}{\partial W_1}$?

How do we train neural networks?

Gradient descent

$$\hat{y}(x;\theta) = NN(x;\theta)$$

To fit, use least-squares:

$$\theta^* = \min_\theta \sum_i^N (NN(x_i;\theta) - y_i)^2 \qquad (2)$$

In general, no analytical solution to (2) exists, so we must use **optimisation**

For example, gradient descent:

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial \sum_i^N (NN(x_i;\theta) - y_i)^2}{\partial \theta_j}$$

or equally

$$\theta_j \leftarrow \theta_j - \gamma \frac{\partial L(\theta)}{\partial \theta_j}$$

Where γ is the learning rate and $L(\theta)$ is the **loss function**

Note that

$$\frac{\partial L(\theta)}{\partial \theta_j} = \sum_i^N 2(NN(x_i;\theta) - y_i)\frac{\partial NN(x_i;\theta)}{\partial \theta_j}$$

Let's consider a fully connected network

$$NN(\boldsymbol{x};\theta) = W_3(\sigma(\overbrace{W_2\sigma(\underbrace{W_1\boldsymbol{x} + \boldsymbol{b_1}}_{\boldsymbol{h}}) + \boldsymbol{b_2}}^{\boldsymbol{g}}) + \boldsymbol{b_3} = \boldsymbol{f} \circ \boldsymbol{g} \circ \boldsymbol{h}(\boldsymbol{x};\theta)$$

How do we calculate $\frac{\partial NN(\boldsymbol{x}_i;\theta)}{\partial W_1}$?

Note $\boldsymbol{f}, \boldsymbol{g},$ and $\boldsymbol{h}$ are vector functions =>

Use the **multivariate chain rule** (= matrix multiplication of **Jacobians**)

$$\frac{\partial NN}{\partial W_1} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{g}}\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial W_1}$$

$$J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{g}} = \begin{pmatrix} \frac{\partial f_1}{\partial g_1} & \cdots & \frac{\partial f_1}{\partial g_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial g_1} & \cdots & \frac{\partial f_m}{\partial g_n} \end{pmatrix}$$

Evaluating the chain rule

$$NN(x;\theta) = W_3(\sigma(W_2\overbrace{\sigma(\underbrace{W_1\boldsymbol{x} + \boldsymbol{b_1}}_{\boldsymbol{h}}) + \boldsymbol{b_2}}^{\boldsymbol{g}}) + \boldsymbol{b_3} = \boldsymbol{f} \circ \boldsymbol{g} \circ \boldsymbol{h}(\boldsymbol{x};\theta)$$

One can show (exercise for the reader!)

$$\frac{\partial NN}{\partial W_1} = \frac{\partial f}{\partial \boldsymbol{g}}\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial W_1} = W_3 \text{ diag}(\sigma'(\boldsymbol{g}))W_2 \text{ diag}(\sigma'(\boldsymbol{h})) \otimes \boldsymbol{x}$$

and therefore

$$\frac{\partial L}{\partial W_1} = \sum_i^N 2(f_i - y_i)\ W_3 \text{ diag}(\sigma'(\boldsymbol{g}_i))\ W_2 \text{ diag}(\sigma'(\boldsymbol{h}_i)) \otimes \boldsymbol{x}_i$$

Backpropagation

Forward pass:

$$\boldsymbol{x}_i \to \boldsymbol{h}_i = W_1 \boldsymbol{x}_i + \boldsymbol{b}_1 \to \boldsymbol{g}_i = W_2 \sigma(\boldsymbol{h}_i) + \boldsymbol{b}_2 \to f_i = W_3 \sigma(\boldsymbol{g}_i) + \boldsymbol{b}_3$$

Save layer outputs in forward pass

Backward pass:

$$\frac{\partial L}{\partial W_1} = \sum_{i}^{N} 2(f_i - y_i)\ W_3\ \mathrm{diag}(\sigma'(\boldsymbol{g}_i))\ W_2\ \mathrm{diag}(\sigma'(\boldsymbol{h}_i)) \otimes \boldsymbol{x}_i$$

Evaluate from left to right (reverse-mode) for efficiency

Similar equations for other weight matrices and bias vectors

Backpropagation

Forward pass:

$$\boldsymbol{x}_i \rightarrow \boldsymbol{h}_i = W_1 \boldsymbol{x}_i + \boldsymbol{b}_1 \rightarrow \boldsymbol{g}_i = W_2 \sigma(\boldsymbol{h}_i) + \boldsymbol{b}_2 \rightarrow f_i = W_3 \sigma(\boldsymbol{g}_i) + \boldsymbol{b}_3$$

Backward pass:

In practice:

PyTorch

TensorFlow

Autodifferentiation tracks all your forward computations and their gradients and applies the chain rule automatically for you, so you don't have to worry!

Lecture summary

- (Deep) neural networks are simply **flexible functions** fit to data
- **Universal approximation** means they can be applied to many different tasks
- DNNs are trained using chain rule (backpropagation) and **gradient descent**