The deep learning revolution

Ole Winther

Bioinformatics Centre, Department of Biology
University of Copenhagen (UCph)

Dept for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)

November 4, 2025

Part 0:
Practical information

Practical information

- When, who and ground rules

Practical information

- When, who and ground rules
- What

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.
- Compute running notebooks (.ipynb): own machine (e.g. Anaconda), Google Colab, Kaggle, Amazon, Azure, ...

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.
- Compute running notebooks (.ipynb): own machine (e.g. Anaconda), Google Colab, Kaggle, Amazon, Azure, . . .
- Week 1 + 2 we will learn the fundamentals both in lectures and hands-on.
- Questions?

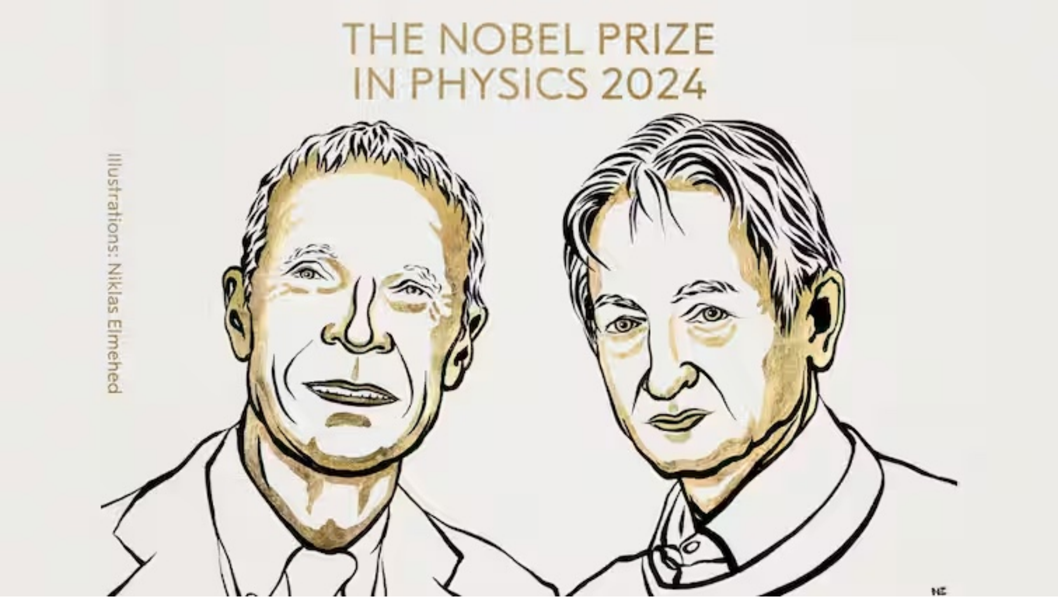
Part 1:
How it started and why we got here

Why we got here - Moore's law also holds for AI

- Rich Sutton, 2019 - The bitter lesson

- One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are search and learning.

Nobel prize 2024 part I



Generative $p(\mathbf{x})$ - Hopfield network and Boltzmann machine

Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. HOPFIELD

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

Generative $p(\mathbf{x})$ - Hopfield network and Boltzmann machine

Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. HOPFIELD

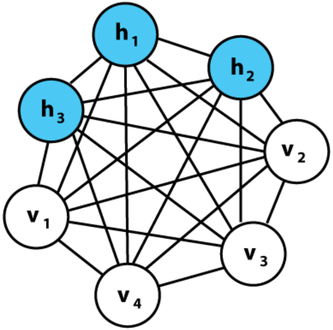
Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

COGNITIVE SCIENCE 9, 147–169 (1985)

A Learning Algorithm for Boltzmann Machines*

DAVID H. ACKLEY
GEOFFREY E. HINTON
Computer Science Department
Carnegie-Mellon University
TERRENCE J. SEJNOWSKI
Biophysics Department
The Johns Hopkins University



Source Wiki

Hopfield network and Boltzmann machine learning

- Define a joint distribution of spins $s_i = \pm 1$:

$$p(\mathbf{s}|\mathbf{W}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W}, \theta)} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

$$Z(\mathbf{W}, \boldsymbol{\theta}) = \sum_{s_1=\pm 1} \cdots \sum_{s_M=\pm 1} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

Hopfield network and Boltzmann machine learning

- Define a joint distribution of spins $s_i = \pm 1$:

$$p(\mathbf{s}|\mathbf{W},\boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W},\theta)} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i \right)$$

$$Z(\mathbf{W},\boldsymbol{\theta}) = \sum_{s_1=\pm 1} \cdots \sum_{s_M=\pm 1} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i \right)$$

- The variables $\mathbf{s} = \{\mathbf{x}, \mathbf{z}\}$ can either be visible (observed): $\mathbf{x}$ or hidden (latent): $\mathbf{z}$:

$$p(\mathbf{x},\mathbf{z}|\mathbf{W},\theta) = \frac{1}{Z(\mathbf{W},\theta)} \exp\left(\mathbf{x}^T \mathbf{W}_{(vv)} \mathbf{x} + \mathbf{x}^T \mathbf{W}_{(vh)} \mathbf{z} + \mathbf{z}^T \mathbf{W}_{(hh)} \mathbf{z} + \boldsymbol{\theta}_{(v)}^T \boldsymbol{x} + \boldsymbol{\theta}_{(h)}^T \mathbf{z} \right)$$

Hopfield network and Boltzmann machine learning

- Define a joint distribution of spins $s_i = \pm 1$:

$$p(\mathbf{s}|\mathbf{W}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W}, \theta)} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

$$Z(\mathbf{W}, \boldsymbol{\theta}) = \sum_{s_1=\pm 1} \cdots \sum_{s_M=\pm 1} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

- The variables $\mathbf{s} = \{\mathbf{x}, \mathbf{z}\}$ can either be visible (observed): $\mathbf{x}$ or hidden (latent): $\mathbf{z}$:

$$p(\mathbf{x}, \mathbf{z}|\mathbf{W}, \theta) = \frac{1}{Z(\mathbf{W}, \theta)} \exp\left(\mathbf{x}^T \mathbf{W}_{(vv)} \mathbf{x} + \mathbf{x}^T \mathbf{W}_{(vh)} \mathbf{z} + \mathbf{z}^T \mathbf{W}_{(hh)} \mathbf{z} + \boldsymbol{\theta}_{(v)}^T \boldsymbol{x} + \boldsymbol{\theta}_{(h)}^T \mathbf{z}\right)$$

- Learning from a collection of data: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ - think binary images
- Likelihood function - treat $\mathbf{z}$ as nuisance parameter:

$$p(\mathbf{x}|\mathbf{W}, \theta) = \sum_{\mathbf{z} \in \{-1,1\}^{d(h)}} p(\mathbf{x}, \mathbf{z}|\mathbf{W}, \theta)$$

Hopfield network and Boltzmann machine learning

- Define a joint distribution of spins $s_i = \pm 1$:

$$p(\mathbf{s}|\mathbf{W}, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{W}, \theta)} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

$$Z(\mathbf{W}, \boldsymbol{\theta}) = \sum_{s_1=\pm 1} \cdots \sum_{s_M=\pm 1} \exp\left(\sum_{i>j} W_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

- The variables $\mathbf{s} = \{\mathbf{x}, \mathbf{z}\}$ can either be visible (observed): $\mathbf{x}$ or hidden (latent): $\mathbf{z}$:

$$p(\mathbf{x}, \mathbf{z}|\mathbf{W}, \theta) = \frac{1}{Z(\mathbf{W}, \theta)} \exp\left(\mathbf{x}^T \mathbf{W}_{(vv)} \mathbf{x} + \mathbf{x}^T \mathbf{W}_{(vh)} \mathbf{z} + \mathbf{z}^T \mathbf{W}_{(hh)} \mathbf{z} + \boldsymbol{\theta}_{(v)}^T \boldsymbol{x} + \boldsymbol{\theta}_{(h)}^T \mathbf{z}\right)$$

- Learning from a collection of data: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ - think binary images
- Likelihood function - treat $\mathbf{z}$ as nuisance parameter:

$$p(\mathbf{x}|\mathbf{W}, \theta) = \sum_{\mathbf{z} \in \{-1,1\}^{d(h)}} p(\mathbf{x}, \mathbf{z}|\mathbf{W}, \theta)$$

- Maximum likelihood learning $\boldsymbol{W}_{\mathrm{ML}}, \boldsymbol{\theta}_{\mathrm{ML}} = \mathrm{argmax}_{\boldsymbol{W}, \boldsymbol{\theta}} \sum_{n=1}^N \log p(\mathbf{x}_n|\mathbf{W}, \theta) \rightarrow$ Boltzmann machine learning rule (involving sampling latent).
- Will meet latent variables model later: VAE and diffusion.

Supervised learning - $p(\mathbf{t}|\mathbf{x})$, $\mathbf{x}$ = input and $\mathbf{t}$ = target

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Supervised learning - $p(\mathbf{t}|\mathbf{x})$, $\mathbf{x}$ = input and $\mathbf{t}$ = target

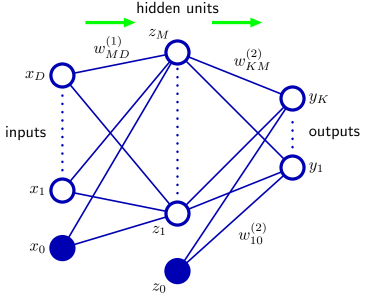
Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

- Rumelhart, Hinton and Williams, Influential work In Parallel Distributed Processing, Volume 1, 1985 and Nature, 1986
- "Just" the application of chain rule of differentiation (Leibniz 1676) to feed-forward neural networks:



- y = output
- Found before by Linnainmaa (1970) and Werbos (1981), Source wiki.

Supervised learning - $p(\mathbf{t}|\mathbf{x})$, $\mathbf{x} = \text{input}$ and $\mathbf{t} = \text{target}$

- Feed forward network first layer with activation function h_1:

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j^{(1)} = h_1(a_j^{(1)}) \ ,$$

- Feed forward network second layer:

$$a_j^{(2)} = \sum_{i}^{M} w_{ji}^{(2)} z_i^{(1)} \ .$$

- Output:

$$y_j = h_2(a_j^{(2)})$$

Supervised learning - $p(\mathbf{t}|\mathbf{x})$, $\mathbf{x} = \text{input}$ and $\mathbf{t} = \text{target}$

- Feed forward network first layer with activation function h_1:

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$
$$z_j^{(1)} = h_1(a_j^{(1)}) \ ,$$

- Feed forward network second layer:

$$a_j^{(2)} = \sum_{i}^{M} w_{ji}^{(2)} z_i^{(1)} \ .$$

- Output:

$$y_j = h_2(a_j^{(2)})$$

- Training data - input $\mathbf{x}$ and target $\mathbf{t}$ pairs: $(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)$

Supervised learning - $p(\mathbf{t}|\mathbf{x})$, $\mathbf{x} = \text{input}$ and $\mathbf{t} = \text{target}$

- Feed forward network first layer with activation function h_1:

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j^{(1)} = h_1(a_j^{(1)}) \ ,$$

- Feed forward network second layer:

$$a_j^{(2)} = \sum_{i}^{M} w_{ji}^{(2)} z_i^{(1)} \ .$$

- Output:

$$y_j = h_2(a_j^{(2)})$$

- Training data - input $\mathbf{x}$ and target $\mathbf{t}$ pairs: $(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)$
- Loss - could be squared loss:

$$E(\mathbf{w}) = \sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x}_n) - \mathbf{t}_n||_2^2$$

- We prefer a probabilistic apprach using minus the log likelihood as loss.

Part 2:
Scaling gradient computation

Automatic differentiation

- We will always use some sort of gradient based learning:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \frac{dE(\mathbf{w})}{d\mathbf{w}}$$

- This can be calculated efficiently in layered networks (Assignment 1 part 1 and 2)

Automatic differentiation

- We will always use some sort of gradient based learning:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \frac{dE(\mathbf{w})}{d\mathbf{w}}$$

- This can be calculated efficiently in layered networks (Assignment 1 part 1 and 2)
- Furthermore, with clever data structures (Assignment 1 part 2) we don't need to program the gradient calculation ourselves.

Automatic differentiation

- We will always use some sort of gradient based learning:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \frac{dE(\mathbf{w})}{d\mathbf{w}}$$

- This can be calculated efficiently in layered networks (Assignment 1 part 1 and 2)
- Furthermore, with clever data structures (Assignment 1 part 2) we don't need to program the gradient calculation ourselves.
- Consider the composite function $y = f(g(h(x)))$ and derivative $\frac{dy}{dx}$:

Automatic differentiation

- We will always use some sort of gradient based learning:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \frac{dE(\mathbf{w})}{d\mathbf{w}}$$

- This can be calculated efficiently in layered networks (Assignment 1 part 1 and 2)
- Furthermore, with clever data structures (Assignment 1 part 2) we don't need to program the gradient calculation ourselves.
- Consider the composite function $y = f(g(h(x)))$ and derivative $\frac{dy}{dx}$:
- Introduce intermediate results:

$$w_0 = x \;, \quad w_1 = h(w_0) \;, \quad w_2 = g(w_1) \;, \quad w_3 = f(w_2) = y$$
$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

- Now use chain rule of differentiation:

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dw_0}$$

Automatic differentiation

- We will always use some sort of gradient based learning:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \frac{dE(\mathbf{w})}{d\mathbf{w}}$$

- This can be calculated efficiently in layered networks (Assignment 1 part 1 and 2)
- Furthermore, with clever data structures (Assignment 1 part 2) we don't need to program the gradient calculation ourselves.
- Consider the composite function $y = f(g(h(x)))$ and derivative $\frac{dy}{dx}$:
- Introduce intermediate results:

$$w_0 = x\ , \quad w_1 = h(w_0)\ , \quad w_2 = g(w_1)\ , \quad w_3 = f(w_2) = y$$
$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

- Now use chain rule of differentiation:

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dw_0}$$

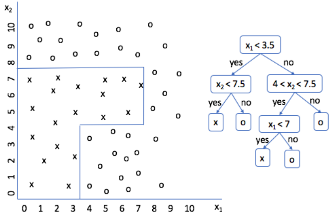
- If we want calculate $\frac{dy}{dx}$ for $x = 2$, we can build the function graph $x \to h \to g \to f$, compute $w_0, \ldots, w_3$ and follow the graph backwards to compute the derivative.

Part 3:

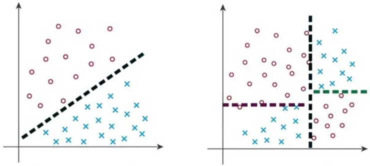
Inductive bias, equivariance and invariance

Inductive bias - why deep learning is not win every Kaggle competition

- Kaggle competition winners:
  - Tabular data - mixed bags of features - Tree methods (XGBoost and Random forest)
  - Text, images, speech, point clouds - Deep learning
  - Time series data - depends on case - spatial (e.g meteorology) or biomedial low dimensional sensor.



Source



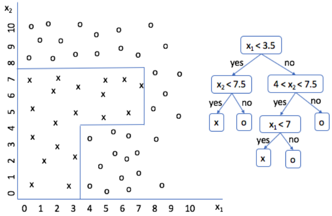
Linearly separable dataset

Linearly inseparable dataset

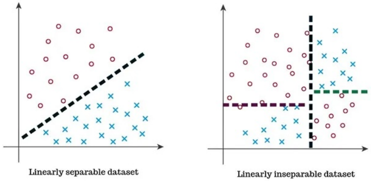
Source

Inductive bias - why deep learning is not win every Kaggle competition

- Kaggle competition winners:
  - Tabular data - mixed bags of features - Tree methods (XGBoost and Random forest)
  - Text, images, speech, point clouds - Deep learning
  - Time series data - depends on case - spatial (e.g meteorology) or biomedial low dimensional sensor.



Source

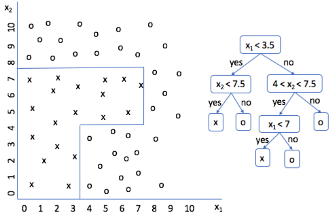


Source

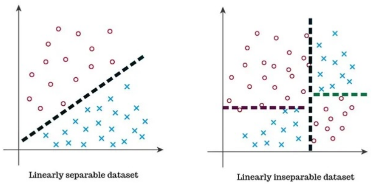
- Properties of data $\leftrightarrow$ favoured model choice. Pay attention throughout the course

Inductive bias - why deep learning is not win every Kaggle competition

- Kaggle competition winners:
  - Tabular data - mixed bags of features - Tree methods (XGBoost and Random forest)
  - Text, images, speech, point clouds - Deep learning
  - Time series data - depends on case - spatial (e.g meteorology) or biomedial low dimensional sensor.

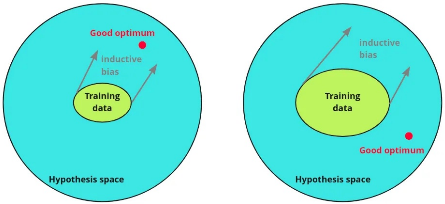


Source



Source

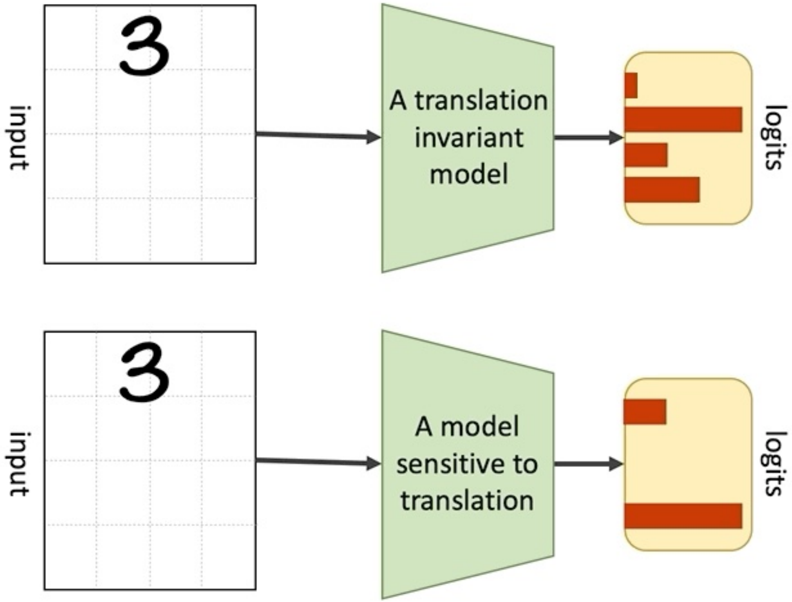
- The amount of data plays a role:



Source

- Properties of data $\leftrightarrow$ favoured model choice. Pay attention throughout the course

Inductive bias - classifier invariant to translation of input image



Source

Invariance and Equivariance

- A neural network layer maps from input data to representation:

$$\mathbf{h} = f(\mathbf{x})$$

- (The “input data” can also be the representation from the previous layer)

Invariance and Equivariance

- A neural network layer maps from input data to representation:

$$\mathbf{h} = f(\mathbf{x})$$

- (The "input data" can also be the representation from the previous layer)
- Some data comes with domain-specific transformations T_ϵ:

$$\mathbf{x} \to \mathbf{x}' = T_\epsilon(\mathbf{x})$$

  reflecting for example spatial-temporal properties.

Invariance and Equivariance

- A neural network layer maps from input data to representation:

$$\mathbf{h} = f(\mathbf{x})$$

- (The "input data" can also be the representation from the previous layer)
- Some data comes with domain-specific transformations T_ϵ:

$$\mathbf{x} \to \mathbf{x}' = T_\epsilon(\mathbf{x})$$

  reflecting for example spatial-temporal properties.
- Sometimes we want "bake" this into the neural network architecture so that $\mathbf{h}' = f(\mathbf{x}')$ behaves in a specific way:

Invariance and Equivariance

- A neural network layer maps from input data to representation:

$$\mathbf{h} = f(\mathbf{x})$$

- (The "input data" can also be the representation from the previous layer)
- Some data comes with domain-specific transformations T_ϵ:

$$\mathbf{x} \to \mathbf{x}' = T_\epsilon(\mathbf{x})$$

  reflecting for example spatial-temporal properties.
- Sometimes we want "bake" this into the neural network architecture so that $\mathbf{h}' = f(\mathbf{x}')$ behaves in a specific way:
- Invariance:

$$\mathbf{h}' = \mathbf{h} \quad \text{that is} \quad f(T_\epsilon(\mathbf{x})) = f(\mathbf{x})$$

Invariance and Equivariance

- A neural network layer maps from input data to representation:

$$\mathbf{h} = f(\mathbf{x})$$

- (The “input data” can also be the representation from the previous layer)
- Some data comes with domain-specific transformations T_ϵ:

$$\mathbf{x} \to \mathbf{x}' = T_\epsilon(\mathbf{x})$$

  reflecting for example spatial-temporal properties.
- Sometimes we want “bake” this into the neural network architecture so that $\mathbf{h}' = f(\mathbf{x}')$ behaves in a specific way:
- Invariance:

$$\mathbf{h}' = \mathbf{h} \quad \text{that is} \quad f(T_\epsilon(\mathbf{x})) = f(\mathbf{x})$$

- Equivariance:

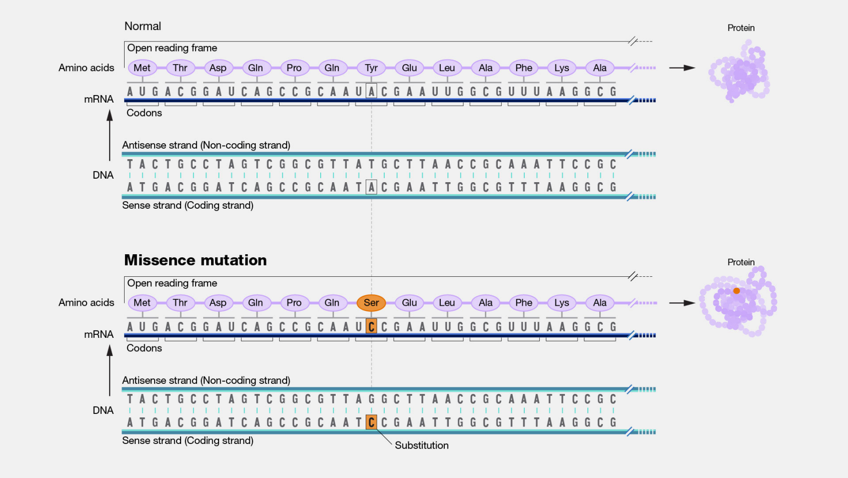
$$\mathbf{h}' = T(\mathbf{h}) \quad \text{that is} \quad f(T_\epsilon(\mathbf{x})) = T_\epsilon(f(\mathbf{x}))$$

Part 4:

Properties of data and how that translate into architectures

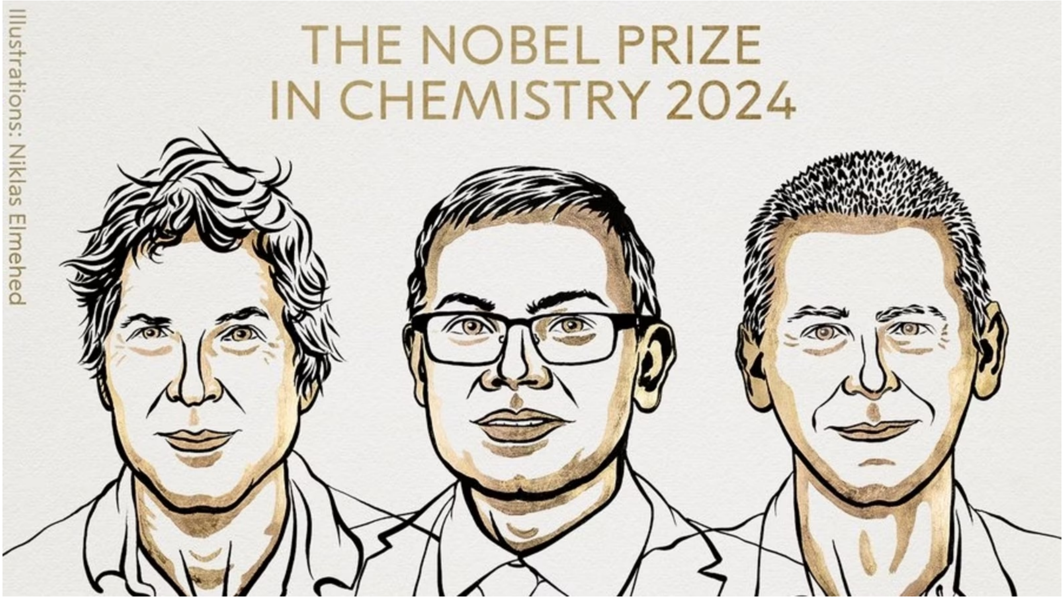
Example application areas

- Images, depth images, ...
- Point clouds, e.g. atom coordinates in molecules
- Text and other sequences, e.g. biological

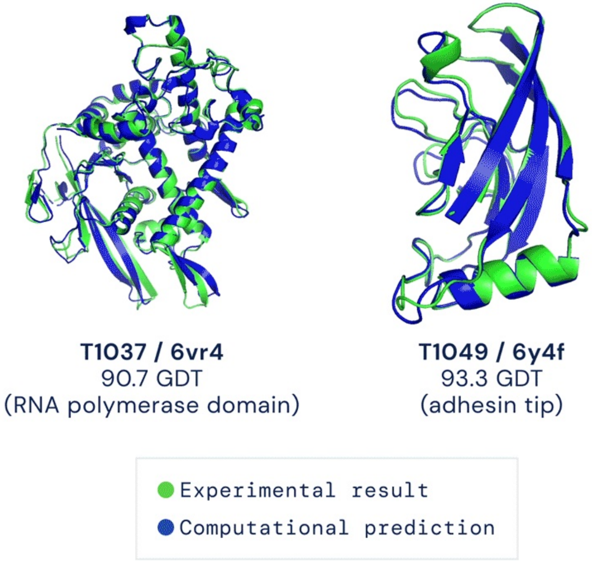


genome.gov

Nobel prize 2024 part II



2020 Protein structure prediction breakthrough - AlphaFold 2



Part 0 reprise:

Practical information

Practical information

- When, who and ground rules

Practical information

- When, who and ground rules
- What

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.
- Compute running notebooks (.ipynb): own machine (e.g. Anaconda), Google Colab, Kaggle, Amazon, Azure, . . .

Practical information

- When, who and ground rules
- What
- Prerequisites - DL is probability, algebra, programming and tinkering.
- Tinkering you will learn here.
- A good foundation in at least one of the other areas will be very helpful to succeed.
- Compute running notebooks (.ipynb): own machine (e.g. Anaconda), Google Colab, Kaggle, Amazon, Azure, . . .
- Week 1 + 2 we will learn the fundamentals both in lectures and hands-on.
- Questions?