Computational Intelligence

LMU Munich
winter term 2024/2025

Thomas Gabor
Claudia Linnhoff-Popien

schedule update

49	2024-12-03 Lecture #8	2024-12-05 Lecture #9
50	2024-12-10 Writing Exercise #4	2024-12-12 Writing Exercise #5
51	2024-12-17 Lecture #10	2024-12-19 Reading Exercise #3

Algorithm 5 (gradient descent)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \mathcal{X} \to \mathcal{T}$ be the first derivative of τ. The process $\mathcal{D}$ continues via gradient descent (with update rate $\alpha \in \mathbb{R}^+$) if e is of the form

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha \cdot \tau'(x_t).$$

The learning rate α can also be given as a function, usually $\alpha : \mathbb{N} \to \mathbb{R}$ so that $e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha(t) \cdot \tau'(x_t)$.
If τ is stochastic, this process is called stochastic gradient descent (SGD).

recap

Algorithm 6 (gradient descent (policy))**.** Let π_θ be a policy π that depends on vector of continuous parameters $\theta \in \Theta$ such that usually $\Theta = \mathbb{R}^N$ for some N. Let $\tau : \Theta \to \mathcal{T}$ be a target function on the parameters θ of a policy π_θ. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \Theta \to \mathcal{T}$ be the first derivative of τ, i.e., $\tau'(\theta) = \frac{\partial\, \tau(\theta)}{\partial\, \theta}$. If $\mathcal{D} = (\Theta, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ is an optimization process that continues via gradient descent, $\mathcal{D}$ is a process of policy optimization via gradient descent.

Many Variants of Gradient Descent

recap

source:
https://optimization.cbe.cornell.edu/index.php?title=AdaGrad

Algorithm 1: AdaGrad general algorithm

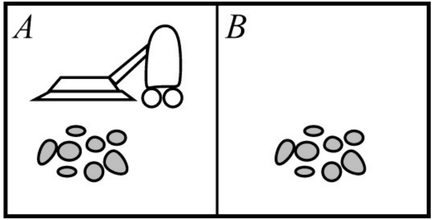
η: Stepsize ;
$f(x)$: Stochastic objective function ;
x_1: Initial parameter vector;
for $t = 1$ *to* T **do**
  Evaluate $f_t(x_t)$;
  Get and save g_t ;
  $G_t \leftarrow \sum_{\tau=1}^{t} g_\tau g_\tau^\top$;
  $x_{t+1} \leftarrow x_t - \eta G_t^{-1/2} g_t$;
end
return x_t

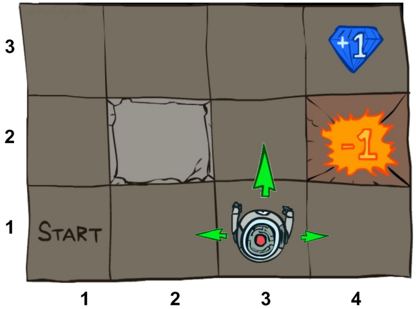
source: https://arxiv.org/pdf/1412.6980.pdf%5D

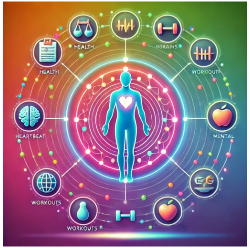
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

running examples







Russel, Norvig. Artificial Intelligence – A Modern Approach. Third Edition. 2016.
https://inst.eecs.berkeley.edu/~cs188/fa22/
chatgpt.com

encoding policies...

Differentiable Programming

Jax (Python)

https://github.com/google/jax

Zygote (Julia)

```
julia> using Zygote

julia> f(x) = 5x + 3

julia> f(10), f'(10)
(53, 5.0)
```

https://github.com/FluxML/Zygote.jl

Neural Networks

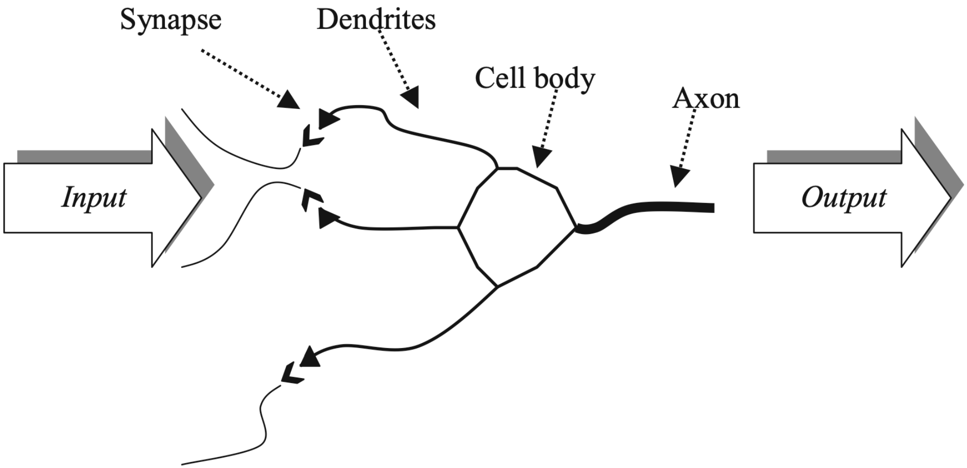
PyTorch

TensorFlow

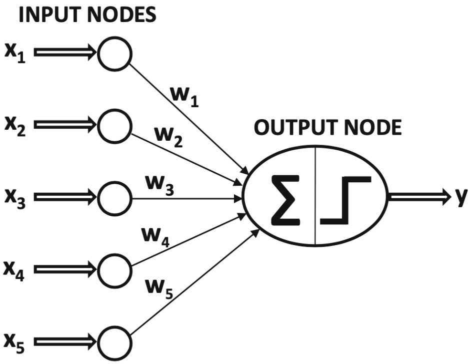
https://pytorch.org

https://www.tensorflow.org

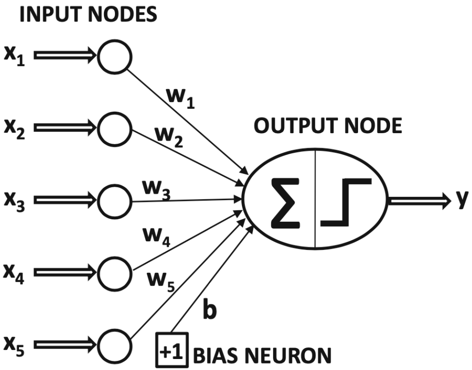
Neural Networks



source: Brabazon, O'Neill, McGarraghy. Natural Computing Algorithms. 2015.

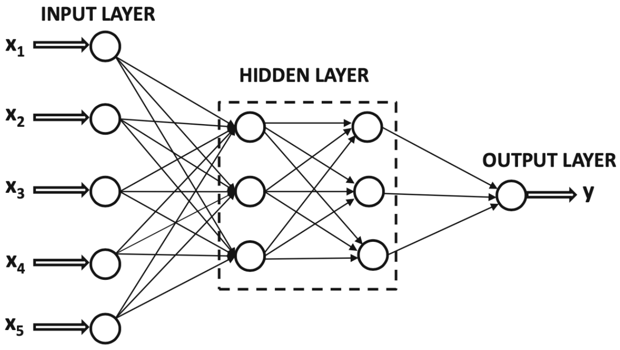


(a) Perceptron without bias

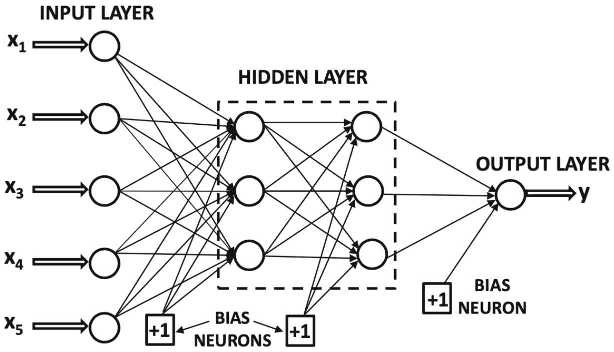


(b) Perceptron with bias

source: Aggarwal. Neural Networks and Deep Learning. 2

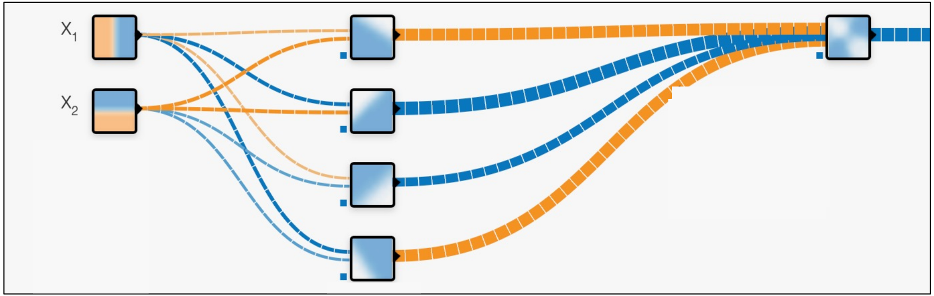


(a) No bias neurons



(b) With bias neurons

Let's try!



https://playground.tensorflow.org

Definition 8 (neural network)**.** A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers $L_1, ..., L_r$ where each layer L_l consists of $|L_l|$ cells $C_{l,1}, ..., C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, ..., |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}, e = 1, ..., |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \to \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{\, l=1,\dots,r,\;\; c=1,\dots,|L_l|,\;\; e=1,\dots,|L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{\, l=1,\dots,r,\;\; c=1,\dots,|L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \mathbin{+\!\!\!+} \mathbf{b}$ is called the network $\mathcal{N}$'s parameters.

Definition 8 (neural network)**.** A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers $L_1, ..., L_r$ where each layer L_l consists of $|L_l|$ cells $C_{l,1}, ..., C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, ..., |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}, e = 1, ..., |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{\,l=1,\ldots,r,\;\; c=1,\ldots,|L_l|,\;\; e=1,\ldots,|L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{\,l=1,\ldots,r,\;\; c=1,\ldots,|L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} + \mathbf{b}$ is called the network $\mathcal{N}$'s parameters.
A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(r, c) \rangle_{c=1,\ldots,|L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f\big(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)\big) & \text{otherwise.} \end{cases}$$

Definition 11 (training of a neural network)**.** Let $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ be a neural network with n weights $\overline{\mathcal{N}} = \mathbf{w} +\!\!+\, \mathbf{b} \in \mathbb{R}^n$ as in Definition 8. Note that thus $|\overline{\mathcal{N}}| = n$. Let $\tau : \mathbb{R}^n \to \mathbb{R}$ be a target function as in Definition 2. Note that thus $\mathcal{T} = \mathbb{R}$. The process of optimizing the network weights $\overline{\mathcal{N}}$ so that $\tau(\overline{\mathcal{N}})$ becomes minimal is called training.

- Let $\mathbb{T} = \{(\mathbf{x}_i, \mathbf{y}_i) \; : \; i = 1, ..., N\}$ be a set of N points of training data, where $\mathbf{x}_i \in \mathbb{R}^p, \mathbf{y}_i \in \mathbb{R}^q$ for all i. If τ is of the form

$$\tau(\overline{\mathcal{N}}) = \sum_{i=1}^{N} (\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i)^2$$

  or a similar form, the process of training $\mathcal{N}$ is called supervised learning.

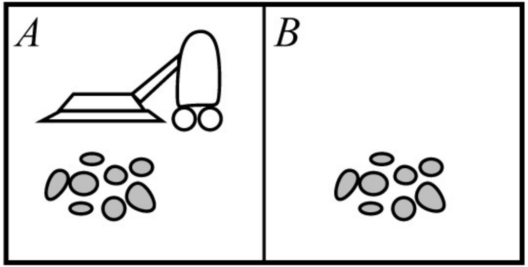
(to be continued)

Example: Vacuum World

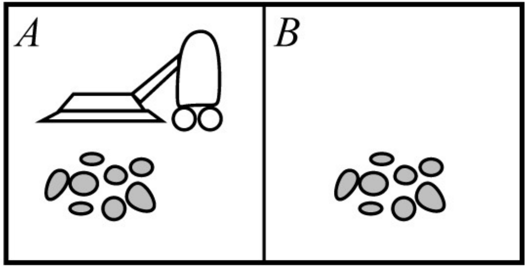
Let $\mathcal{O} = \big\{(\texttt{status\_A} = d_1, \texttt{status\_B} = d_2, \texttt{robot\_position} = p)$

$\mid (d_1, d_2) \in \{\texttt{dirty}, \texttt{clean}\}^2,\ p \in \{\texttt{A}, \texttt{B}\}\big\},$

let $\mathcal{A} = \{\texttt{vacuum}, \texttt{move}\}$.



Example: Vacuum World



$$\text{Let } \mathcal{O} = \Big\{(\texttt{status\_A} = d_1, \texttt{status\_B} = d_2, \texttt{robot\_position} = p) \;\Big|\; (d_1, d_2) \in \{\texttt{dirty}, \texttt{clean}\}^2,\ p \in \{\texttt{A}, \texttt{B}\}\Big\},$$

$$\text{let } \mathcal{A} = \{\texttt{vacuum}, \texttt{move}\}.$$

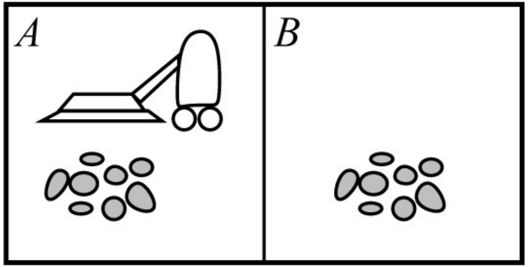
$$\text{Let } \texttt{encode} : \mathcal{O} \to \mathbb{R}^3 \text{ with}$$

$$\texttt{encode}(d_1, d_2, p) = (x_1, x_2, x_3) \text{ so that } d_1 = \texttt{dirty} \implies x_1 = 0 \text{ and so on...}$$

$$\text{Let } \texttt{decode} : \mathbb{R} \to \mathcal{A} \text{ with}$$

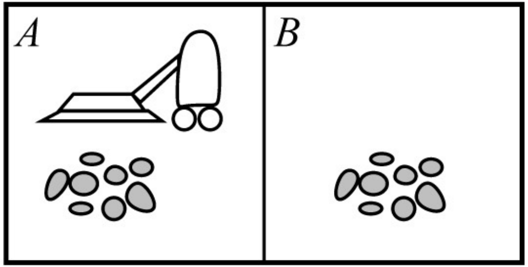
$$\texttt{decode}(y) = \begin{cases} \texttt{vacuum} & \text{if } y > 0.5, \\ \texttt{move} & \text{otherwise.} \end{cases}$$

Example: Vacuum World



$$\text{Let } \mathcal{O} = \big\{(\texttt{status\_A} = d_1, \texttt{status\_B} = d_2, \texttt{robot\_position} = p)$$
$$\mid (d_1, d_2) \in \{\texttt{dirty}, \texttt{clean}\}^2,\ p \in \{\texttt{A}, \texttt{B}\}\big\},$$
$$\text{let } \mathcal{A} = \{\texttt{vacuum}, \texttt{move}\}.$$
$$\text{Let } \texttt{encode} : \mathcal{O} \to \mathbb{R}^3 \text{ with}$$
$$\texttt{encode}(d_1, d_2, p) = (x_1, x_2, x_3) \text{ so that } d_1 = \texttt{dirty} \implies x_1 = 0 \text{ and so on...}$$
$$\text{Let } \texttt{decode} : \mathbb{R} \to \mathcal{A} \text{ with}$$
$$\texttt{decode}(y) = \begin{cases} \texttt{vacuum} & \text{if } y > 0.5, \\ \texttt{move} & \text{otherwise.} \end{cases}$$
$$\text{Now let } \mathcal{N} : \mathbb{R}^3 \to \mathbb{R} \text{ be a neural network, e.g., with}$$
$$\overline{\mathcal{N}} \in \mathbb{R}^{3\cdot 4+4\cdot 2+2\cdot 1} = \mathbb{R}^{22} \text{ for two hidden layers of sizes 4 and 2 and no biases.}$$
$$\text{Let } \Theta = \mathbb{R}^{22}.$$

Example: Vacuum World



Let $\mathcal{O} = \{(\texttt{status\_A} = d_1, \texttt{status\_B} = d_2, \texttt{robot\_position} = p)$
$\mid (d_1, d_2) \in \{\texttt{dirty}, \texttt{clean}\}^2,\ p \in \{\texttt{A}, \texttt{B}\}\},$

let $\mathcal{A} = \{\texttt{vacuum}, \texttt{move}\}$.

Let $\texttt{encode} : \mathcal{O} \to \mathbb{R}^3$ with

$\texttt{encode}(d_1, d_2, p) = (x_1, x_2, x_3)$ so that $d_1 = \texttt{dirty} \implies x_1 = 0$ and so on...

Let $\texttt{decode} : \mathbb{R} \to \mathcal{A}$ with

$$\texttt{decode}(y) = \begin{cases} \texttt{vacuum} & \text{if } y > 0.5, \\ \texttt{move} & \text{otherwise.} \end{cases}$$

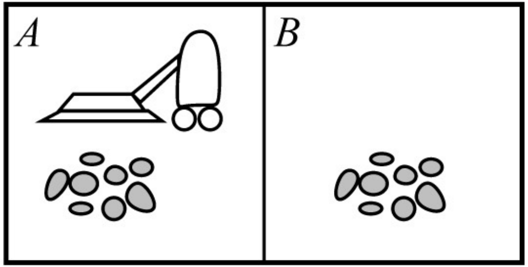
Now let $\mathcal{N} : \mathbb{R}^3 \to \mathbb{R}$ be a neural network, e.g., with

$\overline{\mathcal{N}} \in \mathbb{R}^{3\cdot 4 + 4\cdot 2 + 2\cdot 1} = \mathbb{R}^{22}$ for two hidden layers of sizes 4 and 2 and no biases.

Let $\Theta = \mathbb{R}^{22}$.

Thus, let $\pi_\theta((d_1, d_2, p)) = \texttt{decode}(\mathcal{N}(\texttt{encode}(d_1, d_2, p)))$ where $\overline{\mathcal{N}} = \theta$ for $\theta \in \Theta$.

Example: Vacuum World



Let $\mathcal{O} = \{(\texttt{status\_A} = d_1, \texttt{status\_B} = d_2, \texttt{robot\_position} = p)$

$\mid (d_1, d_2) \in \{\texttt{dirty}, \texttt{clean}\}^2,\ p \in \{\texttt{A}, \texttt{B}\}\}$,

let $\mathcal{A} = \{\texttt{vacuum}, \texttt{move}\}$.

Let $\texttt{encode} : \mathcal{O} \to \mathbb{R}^3$ with

$\texttt{encode}(d_1, d_2, p) = (x_1, x_2, x_3)$ so that $d_1 = \texttt{dirty} \implies x_1 = 0$ and so on...

Let $\texttt{decode} : \mathbb{R} \to \mathcal{A}$ with

$$\texttt{decode}(y) = \begin{cases} \texttt{vacuum} & \text{if } y > 0.5, \\ \texttt{move} & \text{otherwise.} \end{cases}$$

Now let $\mathcal{N} : \mathbb{R}^3 \to \mathbb{R}$ be a neural network, e.g., with

$\overline{\mathcal{N}} \in \mathbb{R}^{3\cdot4+4\cdot2+2\cdot1} = \mathbb{R}^{22}$ for two hidden layers of sizes 4 and 2 and no biases.

Let $\Theta = \mathbb{R}^{22}$.

Thus, let $\pi_\theta((d_1, d_2, p)) = \texttt{decode}(\mathcal{N}(\texttt{encode}(d_1, d_2, p)))$ where $\overline{\mathcal{N}} = \theta$ for $\theta \in \Theta$.

We can now learn a policy π_θ w.r.t. target function $\tau : \Theta \to \mathcal{T}$ by solving the optimization problem τ. If τ' is the derivative of τ w.r.t. Θ, we can use gradient-based optimization as well.

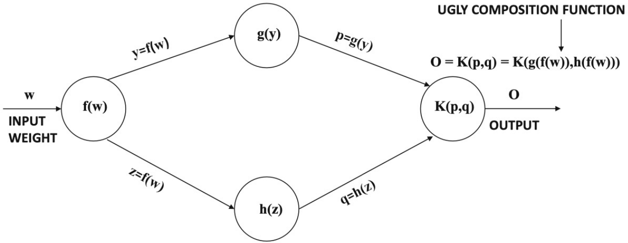
Why use neural networks?

Why use neural networks?

Theorem 2 (Kolmogorov-Arnold representation [2])**.** Any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ for some $n \in \mathbb{N}$ can be written as a finite composition of continuous functions of a single variable ($f_i : \mathbb{R} \to \mathbb{R}$ for $i \in \mathbb{R}$, $1 \leq i \leq n$ for some $n \in \mathbb{N}$) and addition ($\_ + \_ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$).

Backpropagation

Backpropagation



$$
\begin{aligned}
\frac{\partial o}{\partial w} &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial w} \quad \text{[Multivariable Chain Rule]} \\
&= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} \quad \text{[Univariate Chain Rule]} \\
&= \underbrace{\frac{\partial K(p,q)}{\partial p} \cdot g'(y) \cdot f'(w)}_{\text{First path}} + \underbrace{\frac{\partial K(p,q)}{\partial q} \cdot h'(z) \cdot f'(w)}_{\text{Second path}}
\end{aligned}
$$

Figure 1.13: **Illustration of chain rule in computational graphs:** The products of node-specific partial derivatives along paths from weight w to output o are aggregated. The resulting value yields the derivative of output o with respect to weight w. Only two paths between input and output exist in this simplified example.

The Goal Class Hierarchy

Goal Class 5: State Values

Goal Class 4: Rewards and Costs

Goal Class 3: Goal Direction

Goal Class 2: Goal Valuation

Goal Class 2.5: Multiple Goal Valuations

Goal Class 1: Goal Predicate

Goal Class 1.5: Multiple Goal Predicates

Goal Class 0: No Goals