Demystifying Deep Learning with Elixir

Marcel Tilly

Microsoft

+

Why Elixir?

- it is cool
- Move out my comfort zone
- It is functional y = f x, but fun

Why Deep Learning

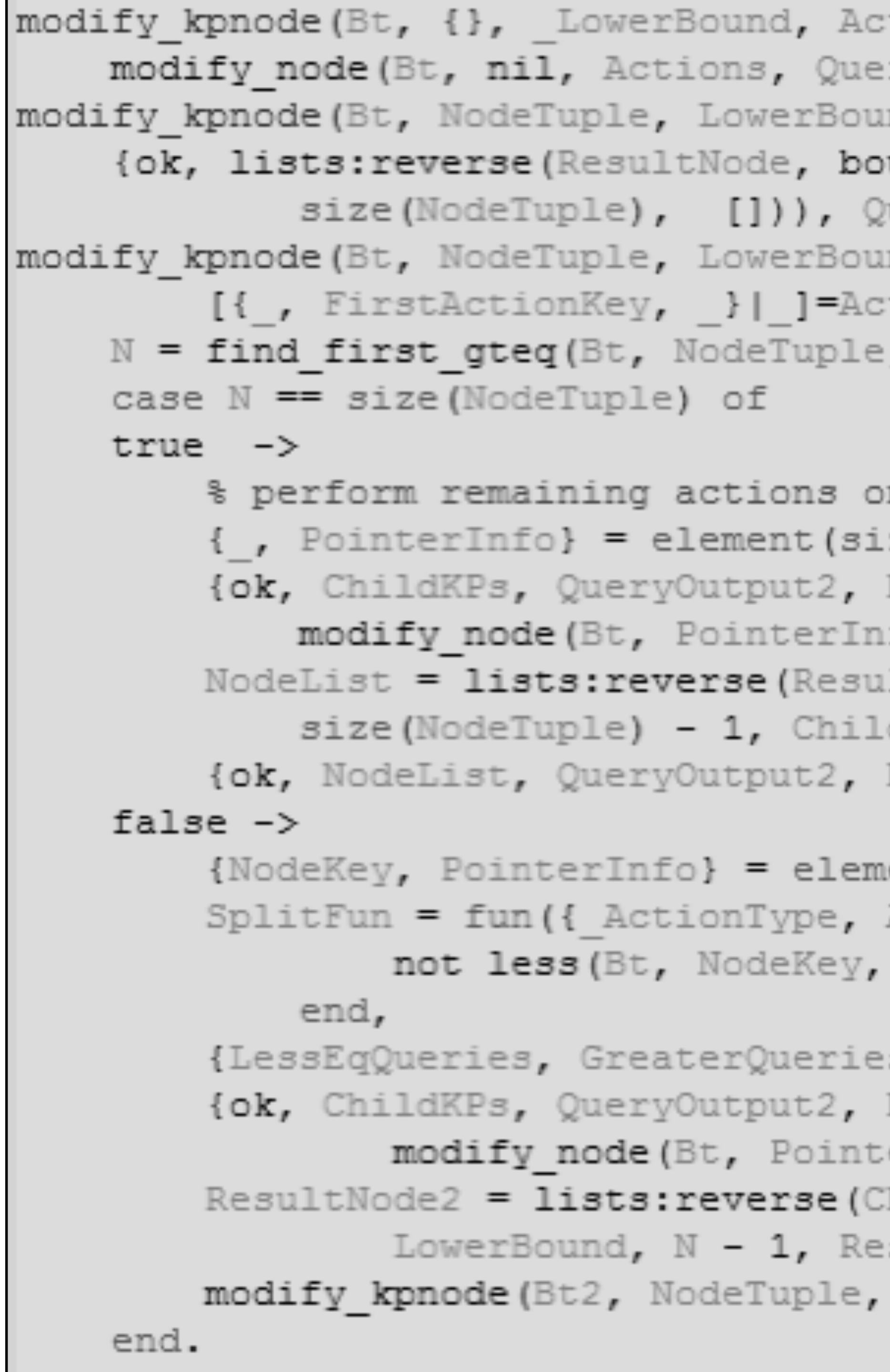
- it is cool
- the “one ring” to rule “everything”
- Demystify the magic

A brief history on Elixir (& Erlang)

- Elixir is a dialect based on Erlang
- Erlang was developed at Ericsson in 1986
  - Designed for telephony systems
  - Proprietary until 1998
  - Growing popularity: Amazon (SimpleDB), Call of Duty, GitHub, Goldman Sachs, Heroku, **WhatsApp**
- What is it good for:
  - Concurrency
  - Fault-tolerance
  - Soft real-time

…and what is Elixir?

- Created by Jose Valim ([Plataformatec]), 2012
- Compiles to Erlang bytecode
- Can call to any Erlang lib
- Friendlier syntax (inspired by Ruby)
- Modern tool chain (Mix)
- "No side effects" - Everything is a function
- Variables can't be reassigned (immutability)
- Lots of recursion
- REPL
- https://elixir-lang.org/

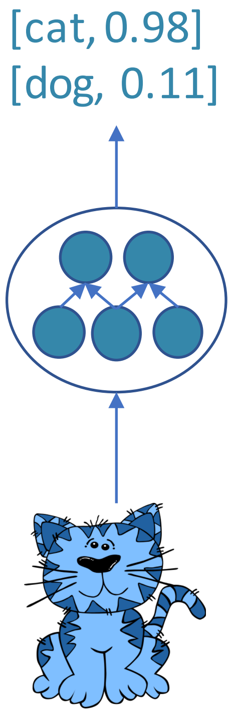


```
modify_kpnode(Bt, {}, _LowerBound, Ac
    modify_node(Bt, nil, Actions, Que
modify_kpnode(Bt, NodeTuple, LowerBou
    {ok, lists:reverse(ResultNode, bo
            size(NodeTuple),  [])), Q
modify_kpnode(Bt, NodeTuple, LowerBou
        [{_, FirstActionKey, _}|_]=Ac
    N = find_first_gteq(Bt, NodeTuple
    case N == size(NodeTuple) of
    true  ->
        % perform remaining actions o
        {_, PointerInfo} = element(si
        {ok, ChildKPs, QueryOutput2,
            modify_node(Bt, PointerIn
        NodeList = lists:reverse(Resu
            size(NodeTuple) - 1, Chil
        {ok, NodeList, QueryOutput2,
    false ->
        {NodeKey, PointerInfo} = elem
        SplitFun = fun({_ActionType,
                not less(Bt, NodeKey,
            end,
        {LessEqQueries, GreaterQuerie
        {ok, ChildKPs, QueryOutput2,
                modify_node(Bt, Point
        ResultNode2 = lists:reverse(C
                LowerBound, N - 1, Re
        modify_kpnode(Bt2, NodeTuple,
    end.
```

Elixir in action…

- **Value Types**: Integers, Floats, Atoms (Symbols), Ranges
- **Collections**: Tuples, Linked Lists, Binaries, Maps
- **System Types**: PIDs, Ports
- Functions
- Pattern Matching
- Guards
- The “magic” |>

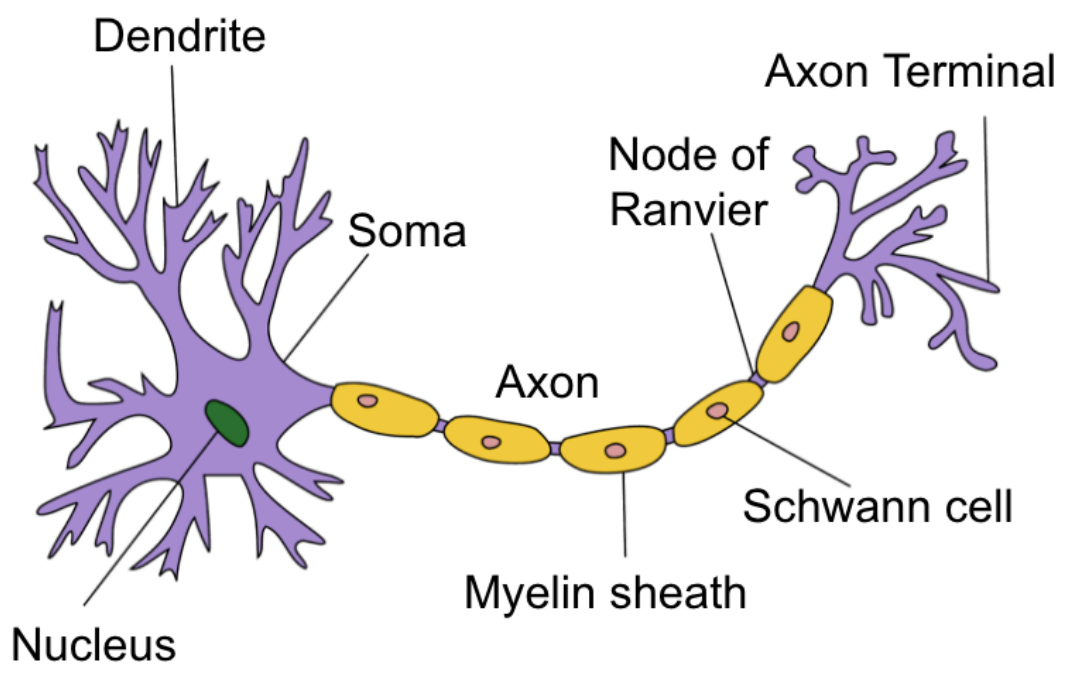
What is *Deep* Learning?



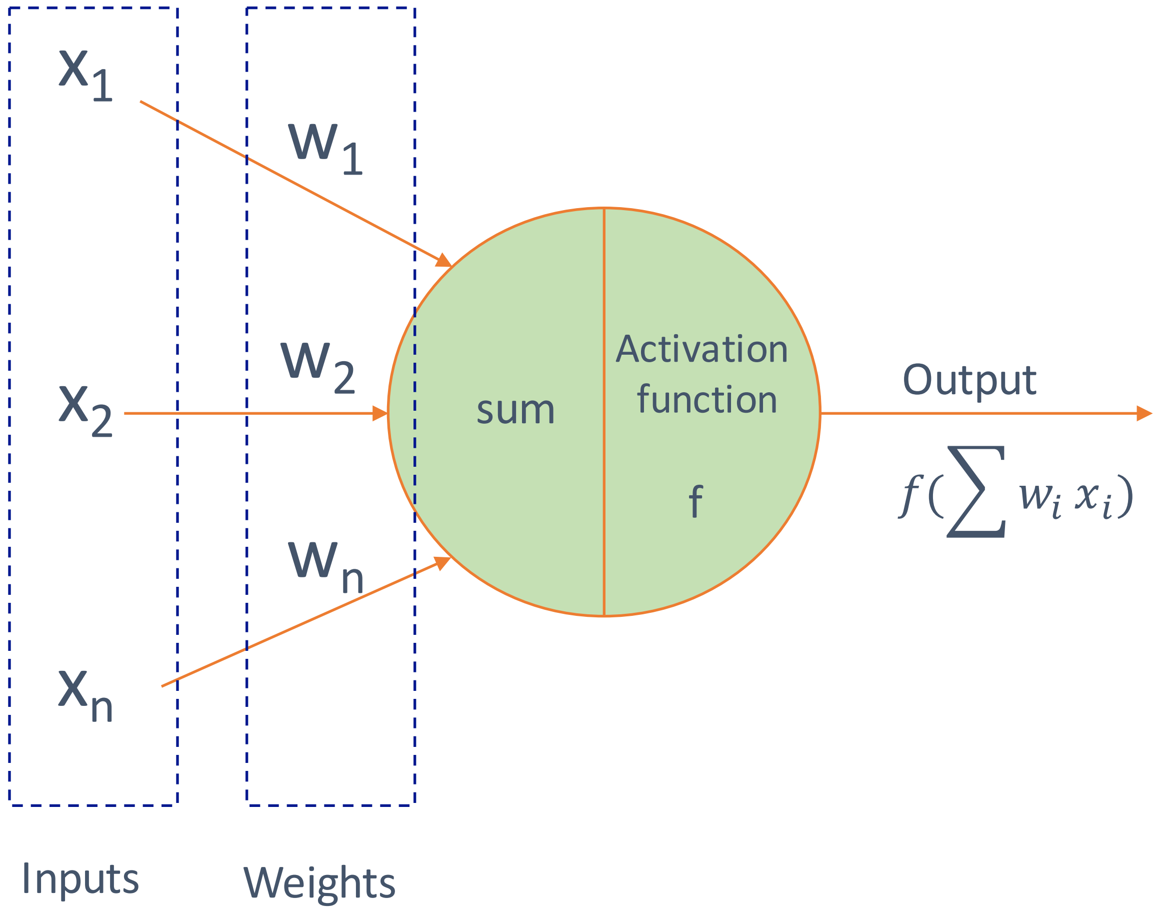
"In deep learning, the algorithms we use now are versions of the algorithms we were developing in the 1980s, the 1990s."

- Geoffrey Hinton

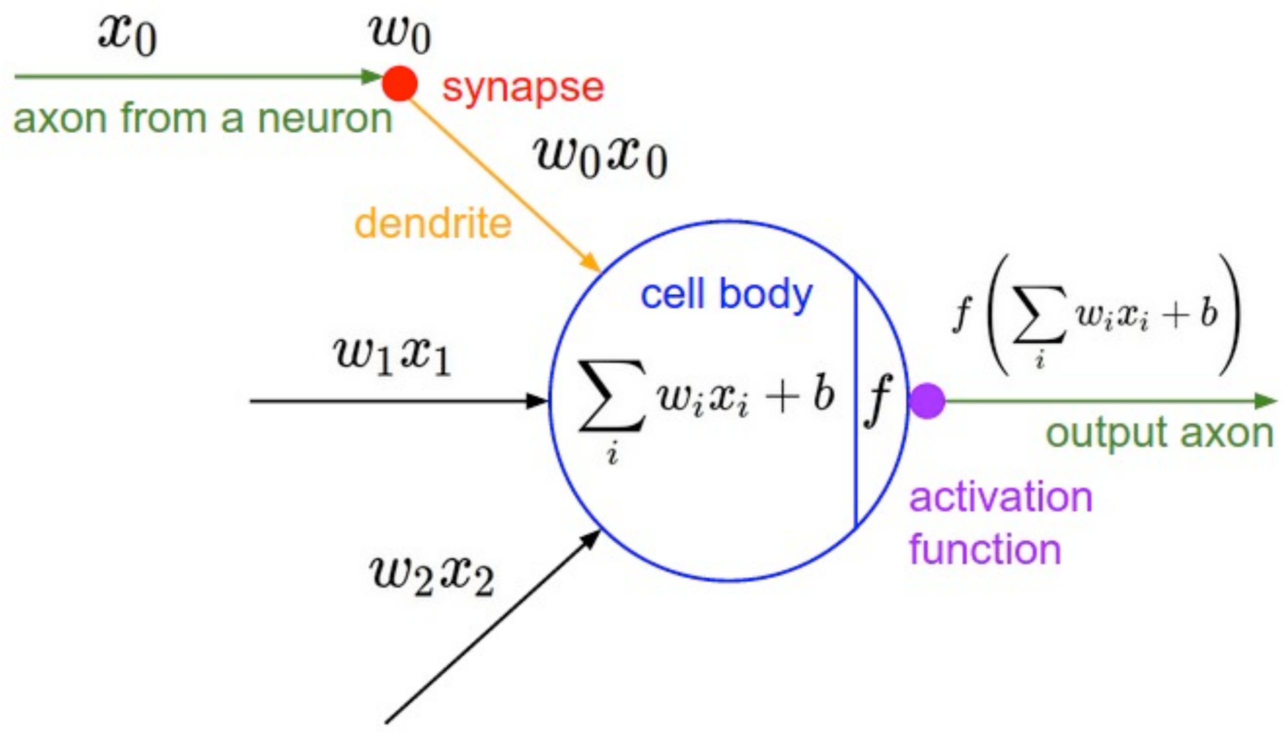
Let‘s start here:

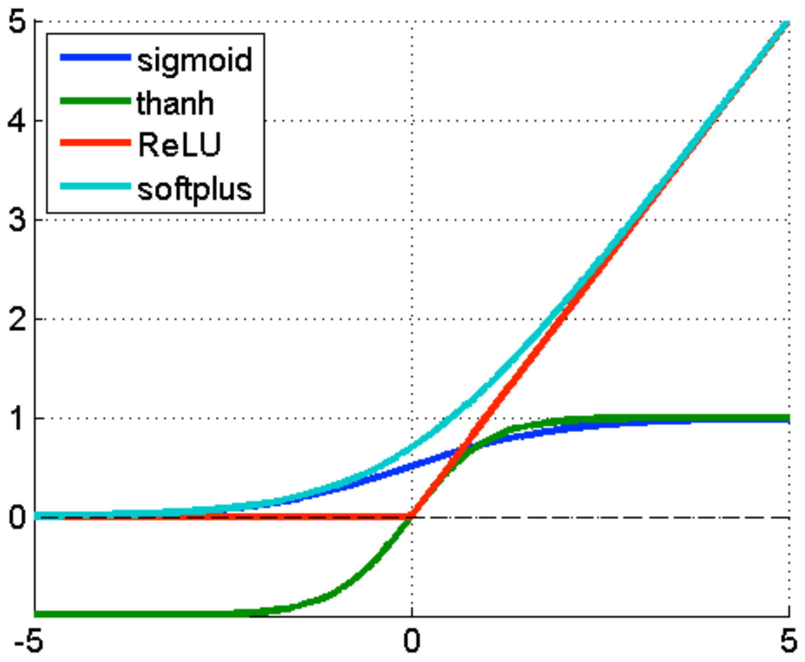


Structure of a typical neuron
(source: Wikipedia)

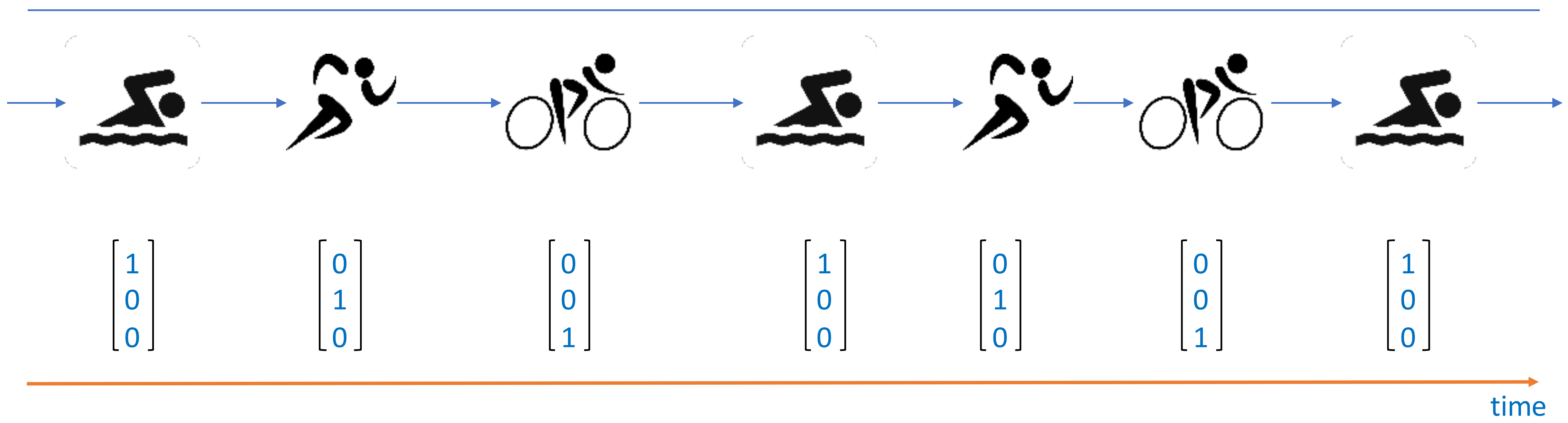


Why a functional programming language?



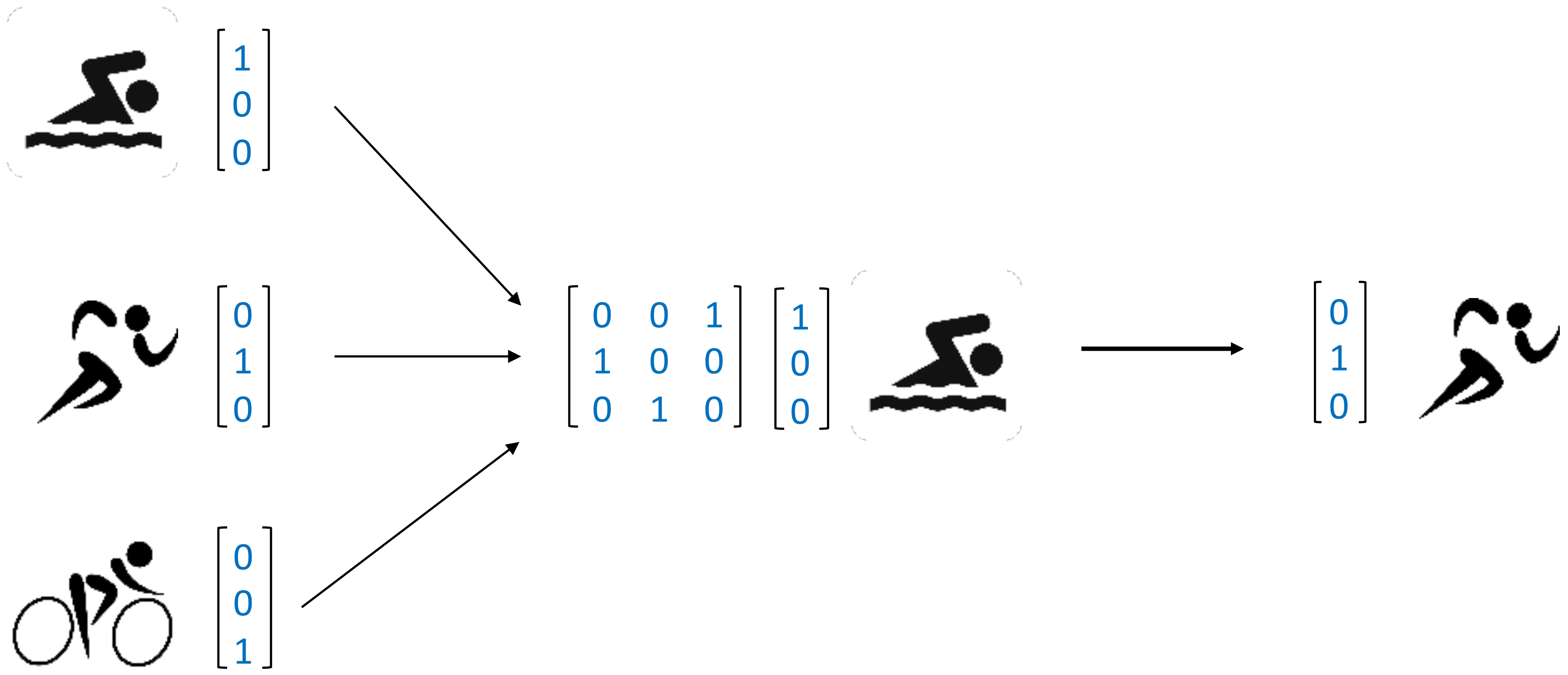


Example: Trainings Planning



$\begin{bmatrix}1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\\0\end{bmatrix}$
time

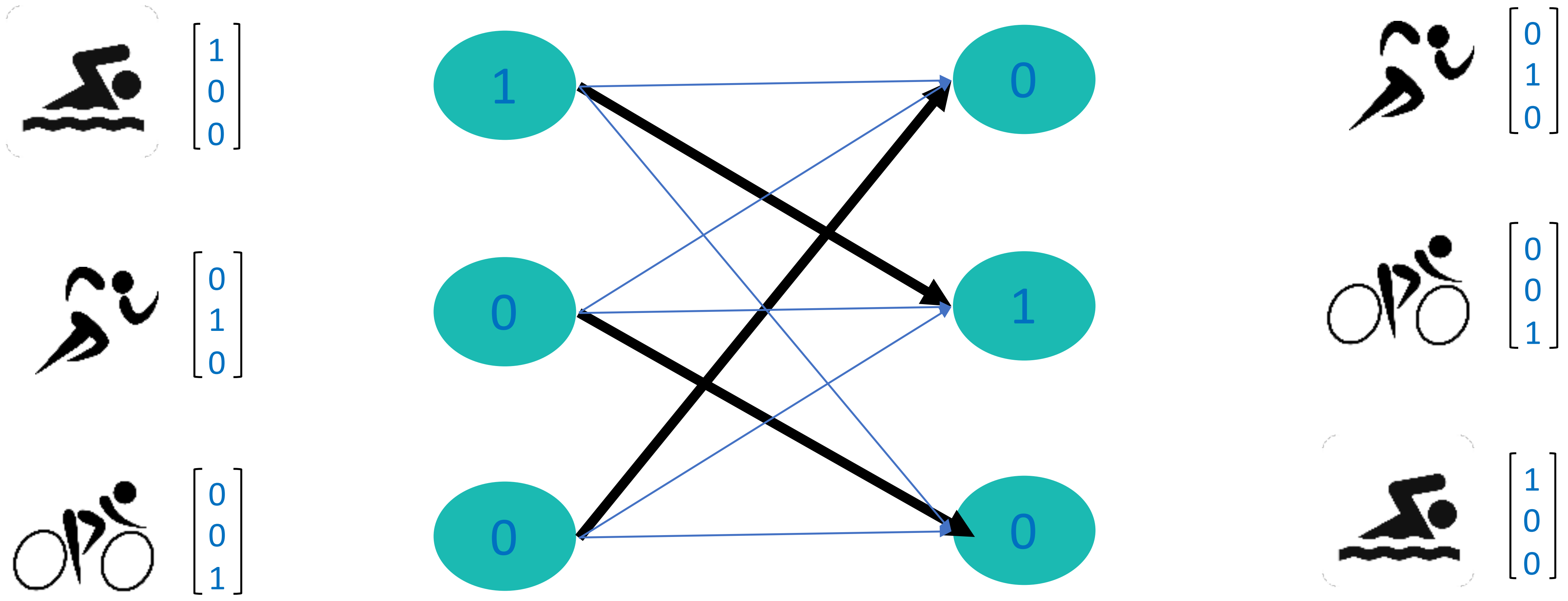
Towards a simple Neural Network



$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

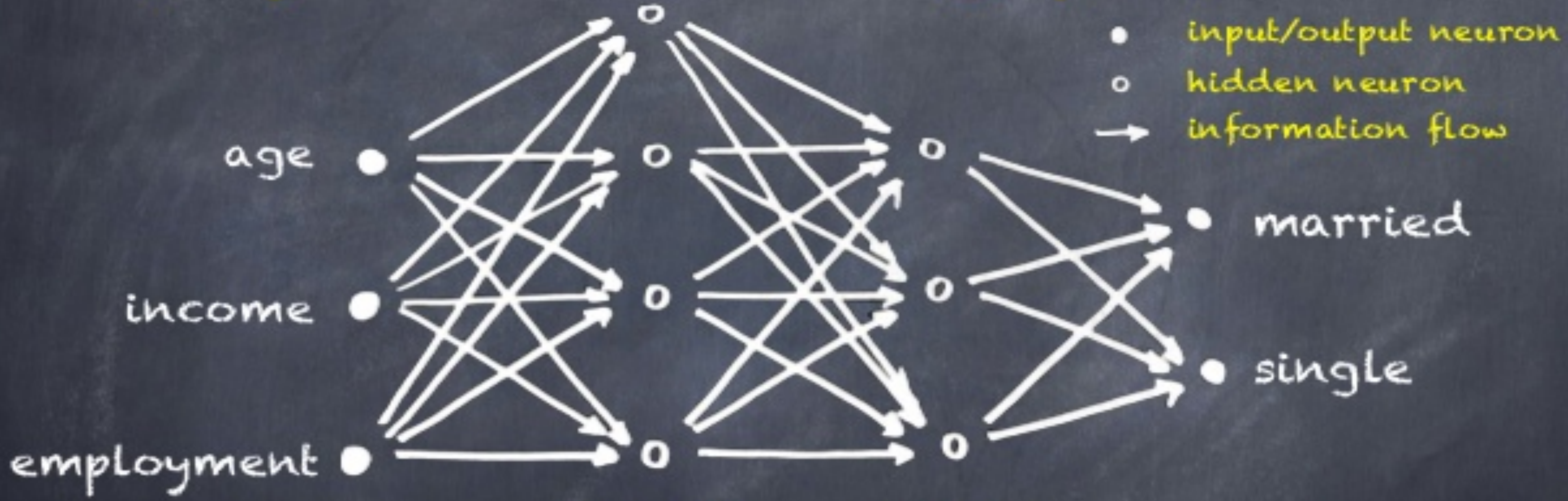
$$\begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Simple Neural Network



$\begin{bmatrix}1\\0\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
1
0
0
0
1
0
$\begin{bmatrix}0\\1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\\0\end{bmatrix}$

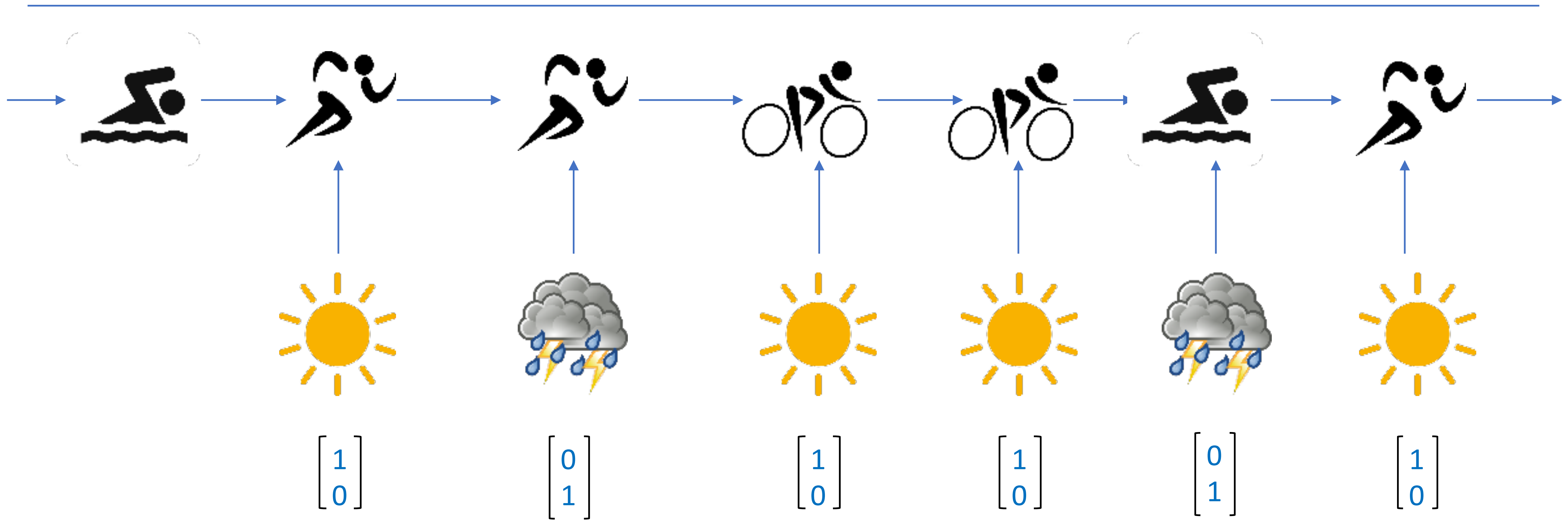
"fully connected" directed graph of neurons



	Input layer	Hidden layer 1	Hidden layer 2	Output layer
#neurons	3	4	3	2

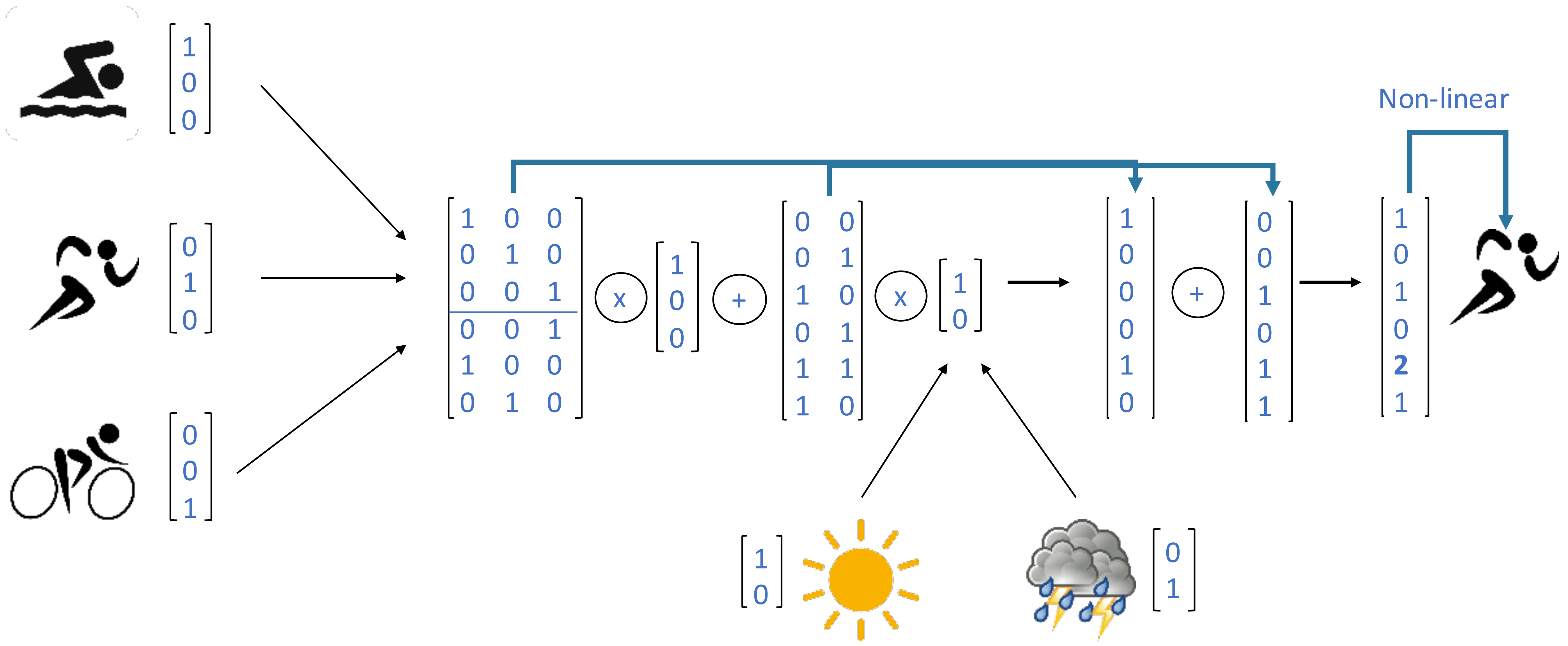
#connections 3x4 4x3 3x2

Motivation: Trainings Planning (incl. Weather)



$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$

Towards a more complex NN

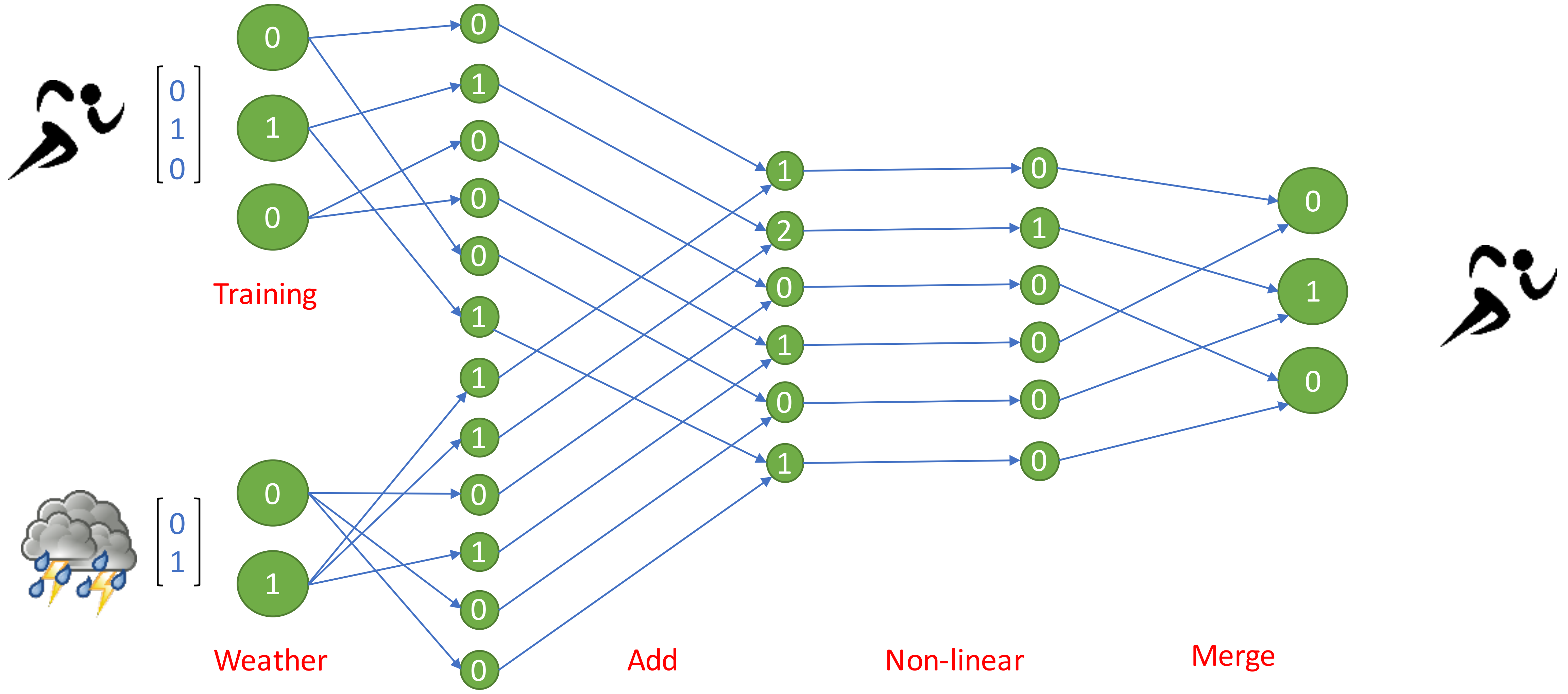


$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\\hline 0&0&1\\1&0&0\\0&1&0\end{bmatrix} \times \begin{bmatrix}1\\0\\0\end{bmatrix} + \begin{bmatrix}0&0\\0&1\\1&0\\0&1\\1&1\\1&0\end{bmatrix} \times \begin{bmatrix}1\\0\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\0\\0\\1\\0\end{bmatrix} + \begin{bmatrix}0\\0\\1\\0\\1\\1\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\1\\0\\\mathbf{2}\\1\end{bmatrix}$$

$$\begin{bmatrix}1\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\end{bmatrix}$$

Recurrent Neural Network *for my Training*



Elixir processes

- All Elixir code runs inside lightweight threads of execution (called processes) that are isolated and exchange information via messages
- Due to their lightweight nature, it is not uncommon to have hundreds of thousands of processes running *concurrently* in the same machine.
- Isolation allows processes to be garbage collected independently, reducing system-wide pauses, and using all machine resources as efficiently as possible (vertical scaling).
- Processes are also able to communicate with other processes running on different machines in the same network. This provides the foundation for distribution, allowing developers to coordinate work across multiple nodes (horizontal scaling).

Wrap-up

- Motivation for using Elixir
- Deep Learning is no magic

BUT

Deep Learning is risky – Data is always biased!

Thanks!

marcel.tilly@microsoft.com