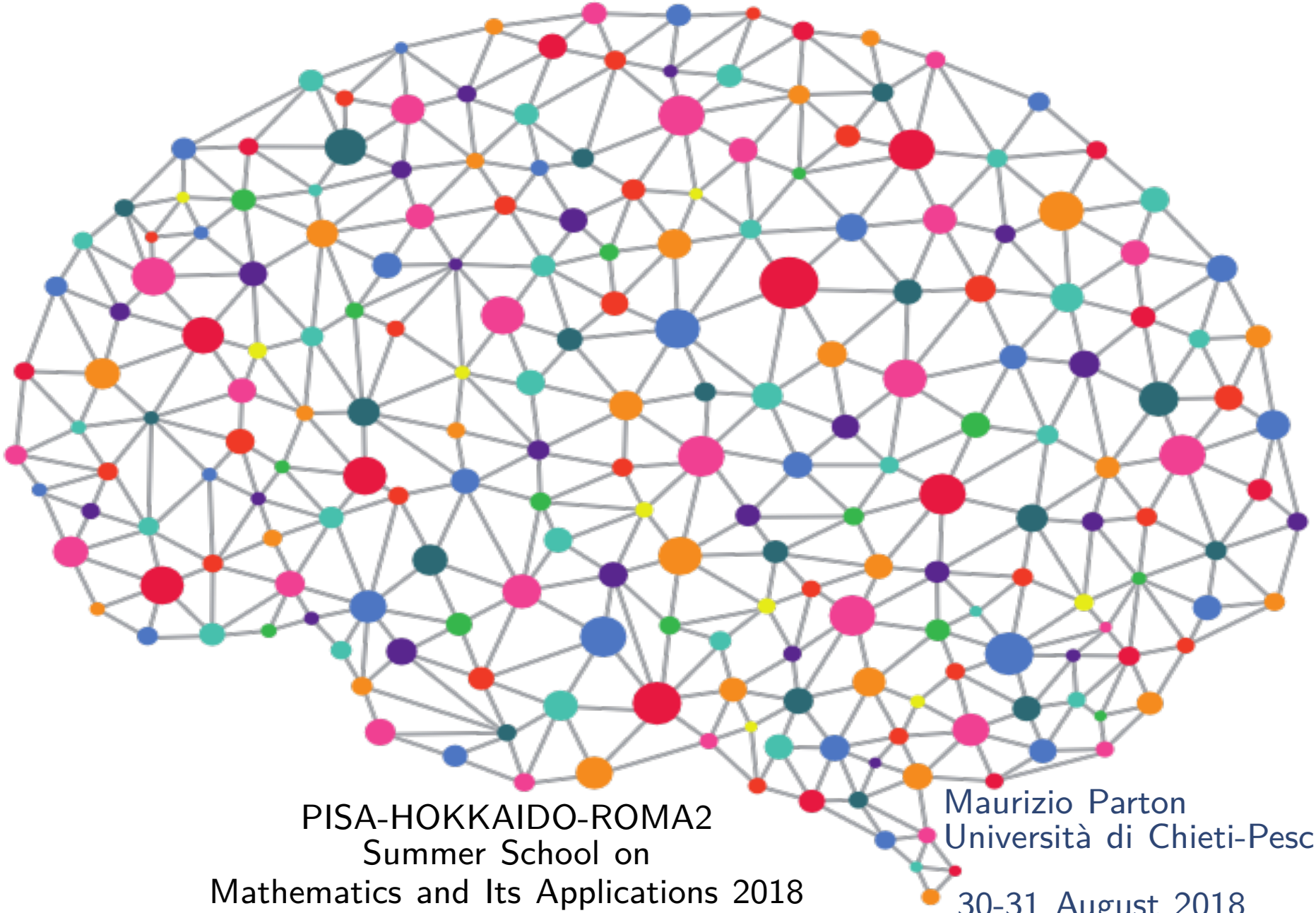
Introduction to neural networks for mathematicians



PISA-HOKKAIDO-ROMA2
Summer School on
Mathematics and Its Applications 2018

Maurizio Parton
Università di Chieti-Pescara

30-31 August 2018

We have a plan!

1. What are AI, ML, NN?
2. (Very quick) historical introduction to AI
3. Functions and neural networks as (computational) graphs
4. Representational power of neural networks
5. Looking for weights aka Learning
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

What to expect?

Time available: $60 * 4 - 15 * 4 = 180$ minutes. Few details, lot of info. Basis and motivation to start studying NN.

What are AI, ML, NN?

This was easy!

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI
3. Functions and neural networks as (computational) graphs
4. Representational power of neural networks
5. Looking for weights aka Learning
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

John McCarthy, Dartmouth Conference on AI, 1956

We propose [. . .] a 2-month, 10-men study of artificial intelligence [. . .] [We conjecture] that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.

An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.

We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

- The term *Artificial Intelligence* (AI) is born, together with the idea of *learning*. Big expectations!

First big AI successes

- Logic Theorist, 1956: proves theorems in Logic.
- Kolmogorov's Representation Theorem, 1957: the only continuous functions of several variable is the addition.
- First *Neural Network* (NN): *Perceptron*, 1957.

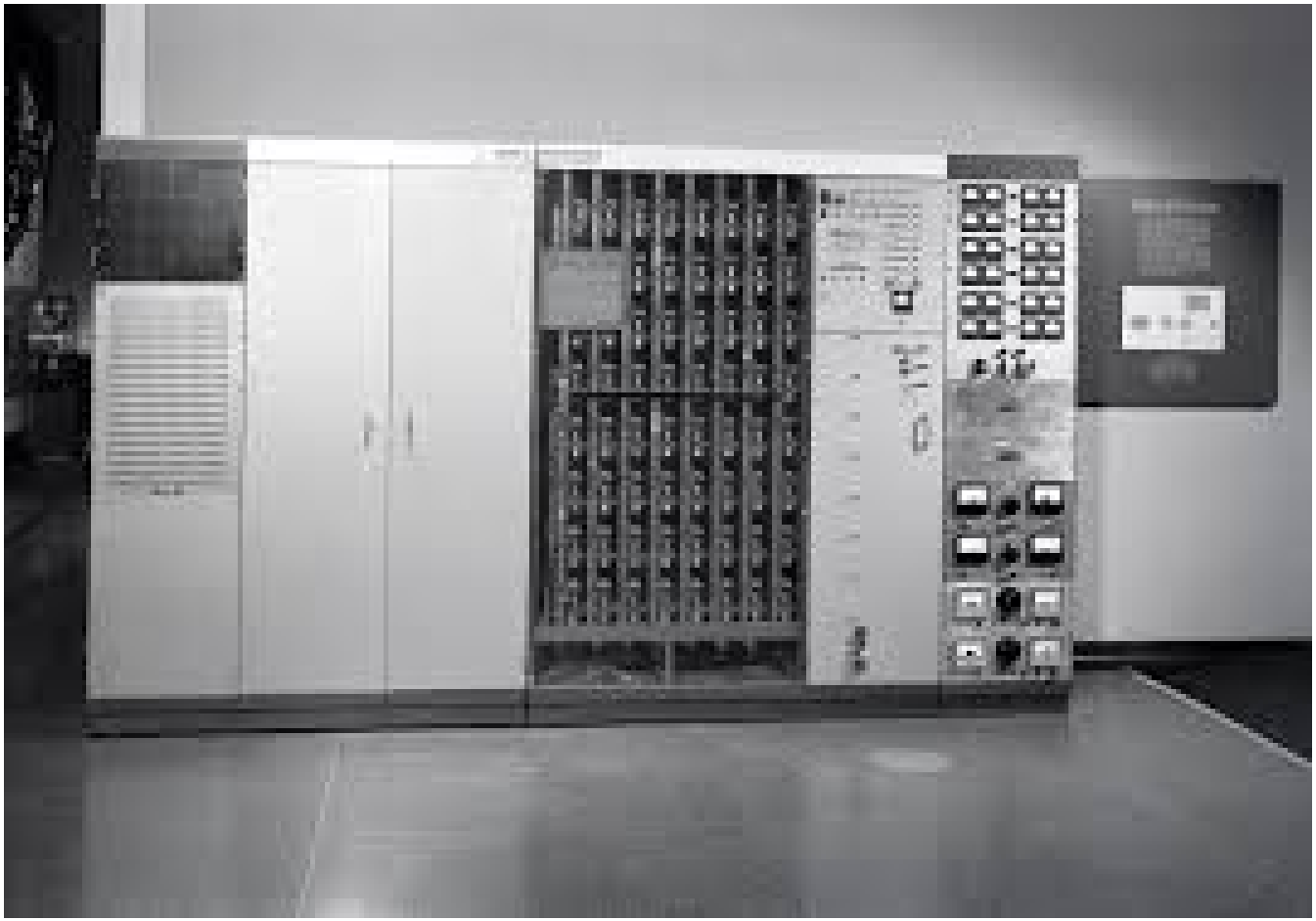
New York Times, 8 July 1958

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

First big AI successes

New York Times, 8 July 1958

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.



First big AI successes

- General Problem Solver, 1959: proves general theorems.

Jerome Wiesner, `https://youtu.be/aygSMgK3BEM`, 1961

I suspect if you come back in 4-5 years I'll say surely they really do think.

Herbert Simon, 1965

Machines will be capable, within twenty years, of doing any work a man can do.

Self-improving machines

John Good, Advances in Computers, 1965

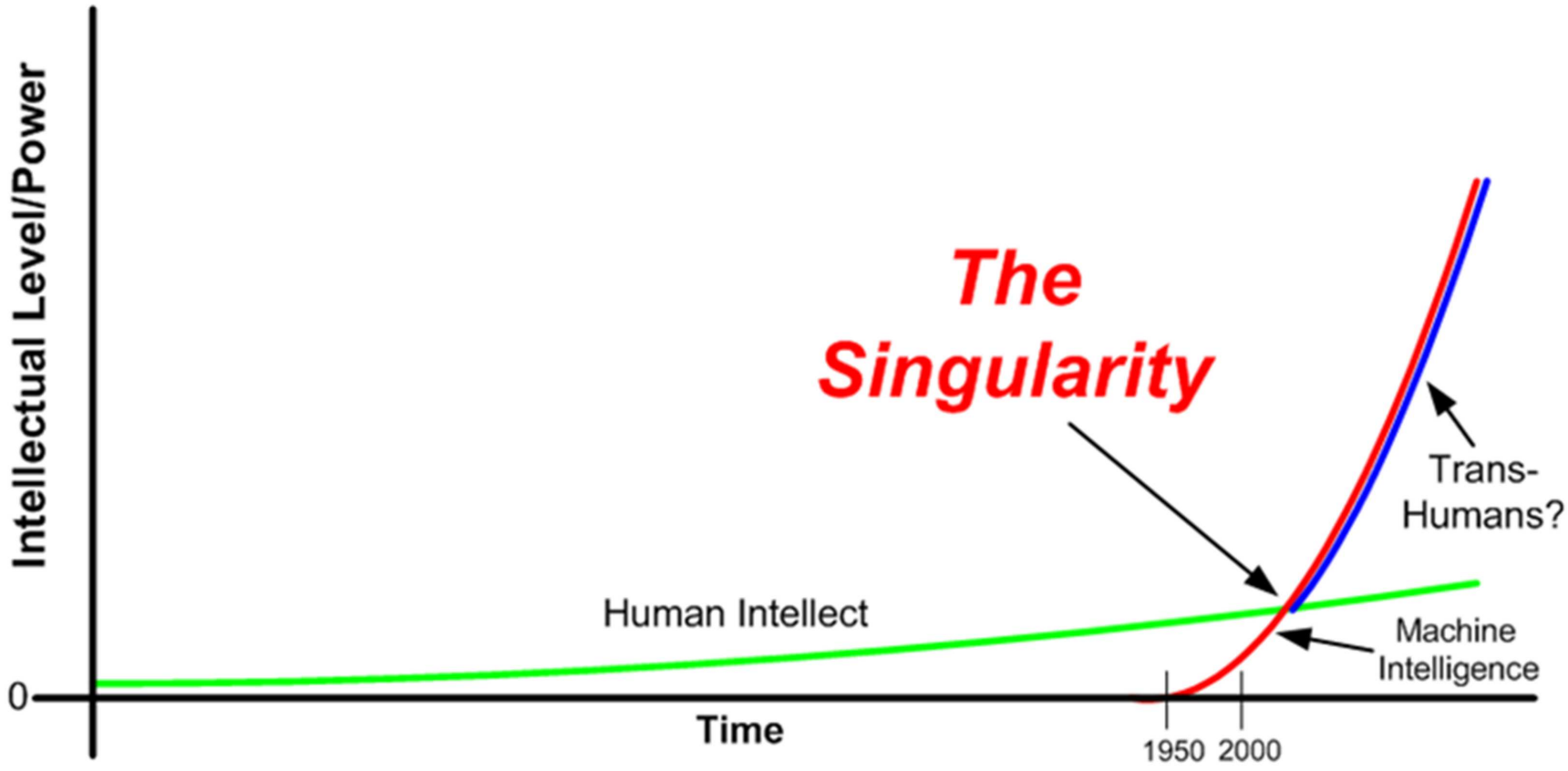
Let an ultraintelligent machine be defined as a machine that can far surpass all the intellectual activities of any man however clever. Since the design of machines is one of these intellectual activities, an ultraintelligent machine could design even better machines; there would then unquestionably be an 'intelligence explosion', and the intelligence of man would be left far behind. Thus the first ultraintelligent machine is the last invention that man need ever make, provided that the machine is docile enough to tell us how to keep it under control...

- *the intelligence of man would be left far behind* $\rightarrow$ Singularity.
- *provided that the machine is docile enough* $\rightarrow$ Need for control: the *Red Button*.

Self-improving machines

Marvin Minsky, 1967

Within a generation [. . .] the problem of creating 'artificial intelligence' will substantially be solved.



The “Perceptrons” book, a milestone, 1969

- “Perceptrons” Marvin Minsky, Seymour Papert, 1969. First mathematical analysis of a neural network. Beautiful reading, strongly suggested!
- It describes *failures* (XOR), *merits* (AND, OR, NOT) and *future* (multilayer networks) of perceptrons.

The “Perceptrons” book, a milestone, 1969

First AI winter: 1974-1980

- Limited computing power: exhaustive search doesn't work.
- Too much faith in advances in computing power.
- Cold war, failure in automatic translations Russian-English: "the spirit is willing but the flesh is weak" $\leftrightarrow$ "the vodka is good but the meat is rotten" (this citation is probably a myth, but gives the idea).

Failures $\Rightarrow$ Pessimism in AI community $\Rightarrow$ Pessimism in the press $\Rightarrow$ No more funding $\Rightarrow$ End of scientific research.

After the winter: 1980-1987

- *Expert systems*: AI specialized in one task.
- Mycin: identify bacterial infections and recommend antibiotics with the correct dosage.
- Dendral: identify unknown organic molecules.
- SID: CPU design.

After the winter: 1980-1987

- Paul Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, 1974: *backpropagation* algorithm.
- With the backpropagation, the perceptron is now able to learn!
- *Deep NN* (DNN): NN with many layers can be trained.
- New successes $\Rightarrow$ New trust $\Rightarrow$ New money $\Rightarrow$ New research.

After the winter: 1980-1987

- Paul Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, 1974: *backpropagation* algorithm.
- With the backpropagation, the perceptron is now able to learn!
- *Deep NN* (DNN): NN with many layers can be trained.
- New successes $\Rightarrow$ New trust $\Rightarrow$ New money $\Rightarrow$ New research.

Ask the question, please!

After the winter: 1980-1987

- Paul Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, 1974: *backpropagation* algorithm.
- With the backpropagation, the perceptron is now able to learn!
- *Deep NN* (DNN): NN with many layers can be trained.
- New successes $\Rightarrow$ New trust $\Rightarrow$ New money $\Rightarrow$ New research.

Ask the question, please!

Q: "Is 1974 a typo?".
A: "No. It is just a very bad choice of time!".

After the winter: 1980-1987

Paul Werbos, 2006

In the early 1970s, I did in fact visit Minsky at MIT. I proposed [...] a way that uses the reverse method, which we now call backpropagation in the ANN field. But Minsky was not interested. In fact, no one at MIT or Harvard or any place else I could find was interested at the time.

- Werbos published his PhD thesis only in 1982, because of the AI winter!
- The backpropagation algorithm was rediscovered several times from 1980 to 1987.
- David Rumelhart, Geoffrey Hinton, and Ronald Williams rediscovered the technique in 1985-1986. They succeeded in making it widely known.

New approaches to learning

- Successes in speech recognition, with *Recurrent NN* (RNN).
- *Reinforcement Learning* (RL): a new learning technique that does not require a training set, 1989.
- *TD-gammon*, 1995: a reinforcement learning NN playing backgammon at world-champion level.

Second AI winter: 1987-(1994)-2006

- Sepp Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen*, Diploma thesis, 1991: backpropagation applied to DNN suffers of the *vanishing or exploding gradients* problem. *Perceptron-like effect* on fundings!
- *Shallow* NN can be used only for very simple tasks.
- The techniques used for Backgammon doesn't work for Chess and Go.
- Deep Blue, 1997: great success in Chess *without* using any AI technique!
- Other more promising techniques appear: random forests and support vector machines.

Second AI winter: 1987-(1994)-2006

The Star, 17 April 2015

in 2004, Hinton asked to lead a new program on neural computation. The mainstream machine learning community could not have been less interested in neural nets.

The Star, 17 April 2015

"It was the worst possible time," says Bengio [...] "Everyone else was doing something different."

The Deep Learning conspiracy

Table: Geoffrey Hinton, Yann LeCun, Yoshua Bengio

- They perseverate, thanks to *Canadian Institute for Advanced Research* (CIFAR).
- They rebrand the topic under the term *Deep Learning*.

Geoffrey Hinton, Simon Osindero, Yee-Whye Teh, 2006

A breakthrough paper that *solves the vanishing or exploding gradients problem*: a new era for DNN begins!

After 2006

- MNIST digits recognition with 1% error rate (2006).
- DNN are more efficient for difficult problems than shallow methods (2007).
- Speech recognition with 23% error rate (2011).
- Image recognition with *Convolutional NN* (CNN): win the ILSVRC-2012 (Olympics of computer vision) with 15.3% error rate - second place got 26.2%!
- Speech Recognition and instant translation: `https://www.youtube.com/watch?v=Nu-nlQqFCKg`, 2012.
- Skype real time translation: 2014.

After 2006

- Atari Breakout: `https://www.youtube.com/watch?v=V1eYniJ0Rnk`, 2015.
- AlphaGo, supervised learning: 2016. Wins against best human players.
- AlphaGo Zero, reinforcement learning: 2017. Wins against AlphaGo.
- Alpha Zero, plays Go, Chess and Shogi with reinforcement learning: 2017. Wins against best Chess software.

Why this is happening now?

- Backpropagation and ReLU.
- Moore's law: $x2$ every 18 months.
- Graphics cards (GPU): cheap computing power.
- Facebook, Google and Microsoft entered the game.
- 1998: MNIST, $60 * 10^3$ images. 2012: ImageNet, $1.2 * 10^6$ images.
- Big Data now available.
- A feedback of exciting new successes.

Is all of this just fuss and hype?

Q: Is another AI or DL winter coming?

Is all of this just fuss and hype?

Q: Is another AI or DL winter coming?
A: No.

Summing up

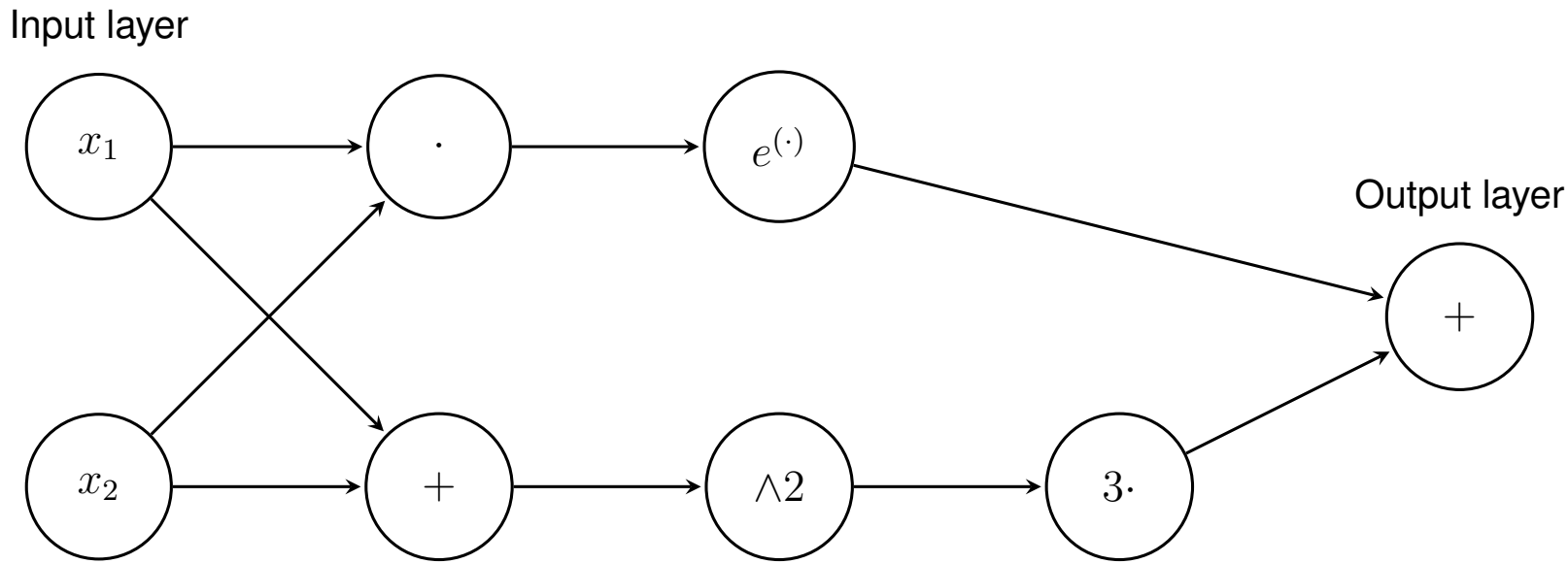
- AI research depends very much on funding.
- Fuss or not, today there is a lot of money for research in NN and DL.
- I strongly suggest you consider doing research in NN and DL.

Not really *very* quick...

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs
4. Representational power of neural networks
5. Looking for weights aka Learning
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

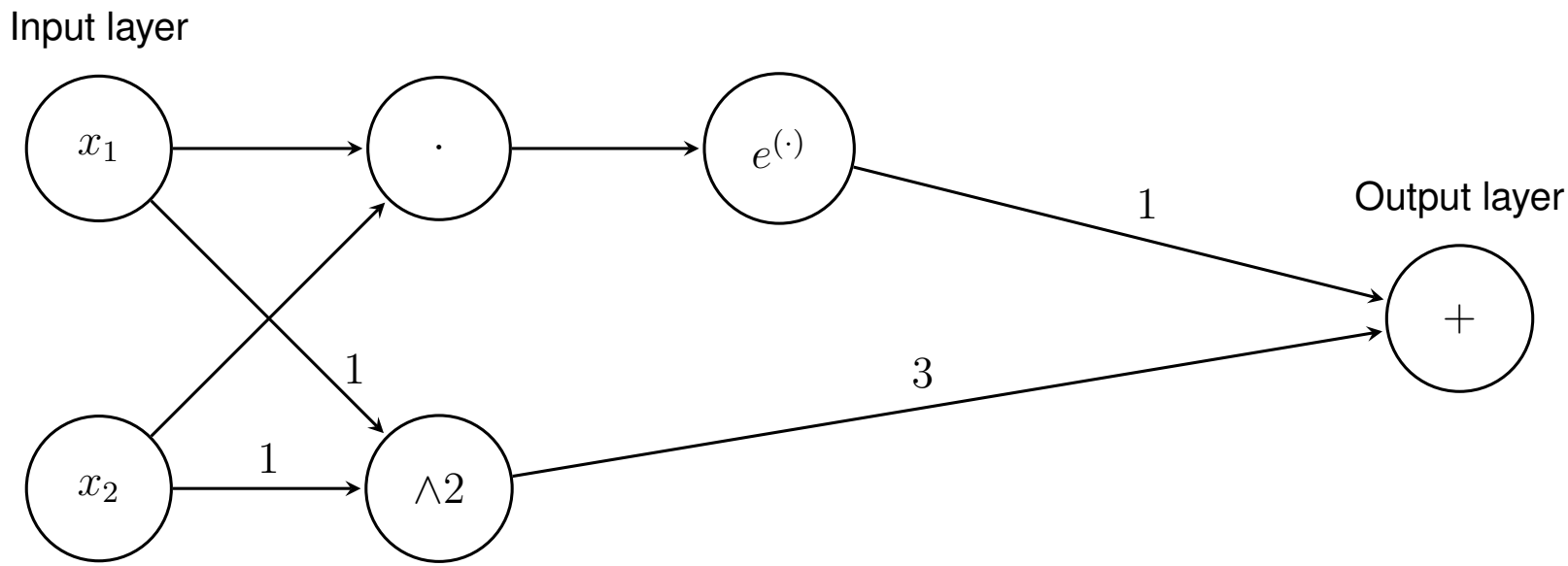
Computational graphs

- *Computational graph* of $e^{x_1 x_2} + 3(x_1 + x_2)^2$.



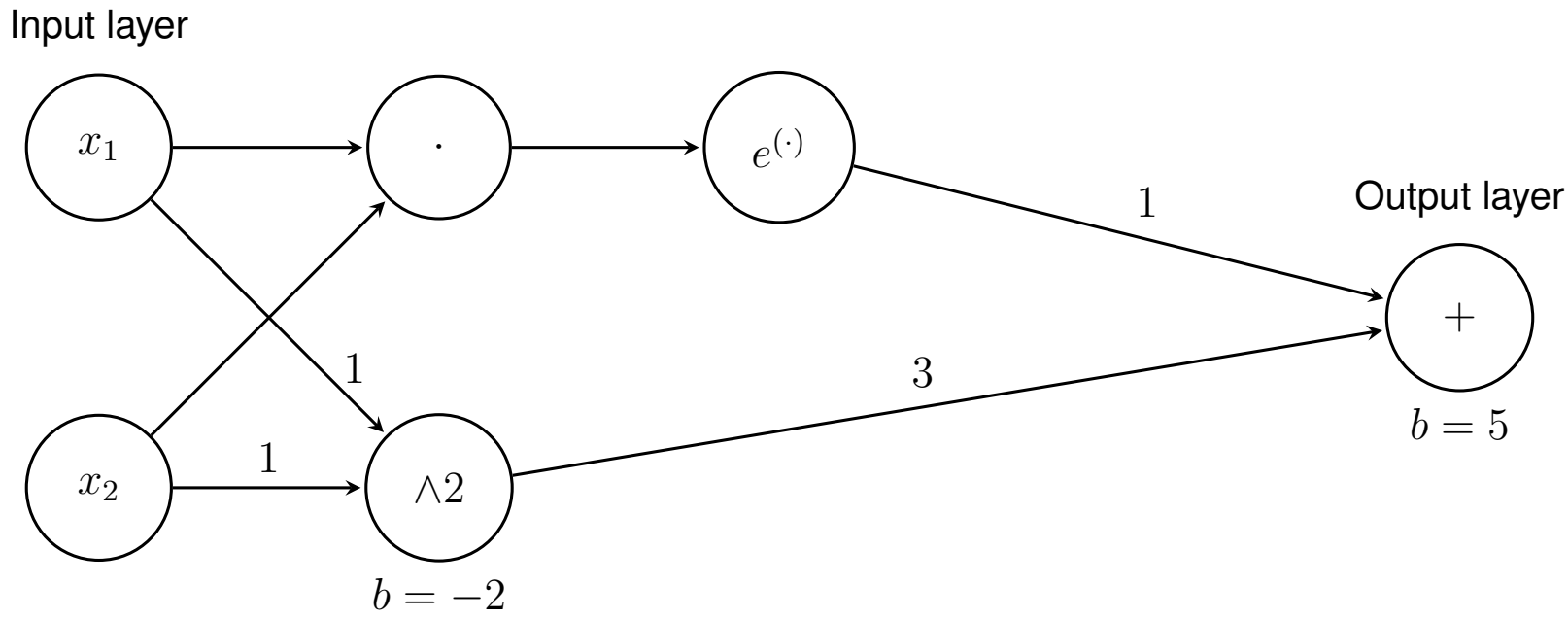
Computational graphs

- Notation: numbers on arrows are used for linear (affine) combinations of inputs. $f(x_1, x_2) = e^{x_1 x_2} + 3(x_1 + x_2)^2$.



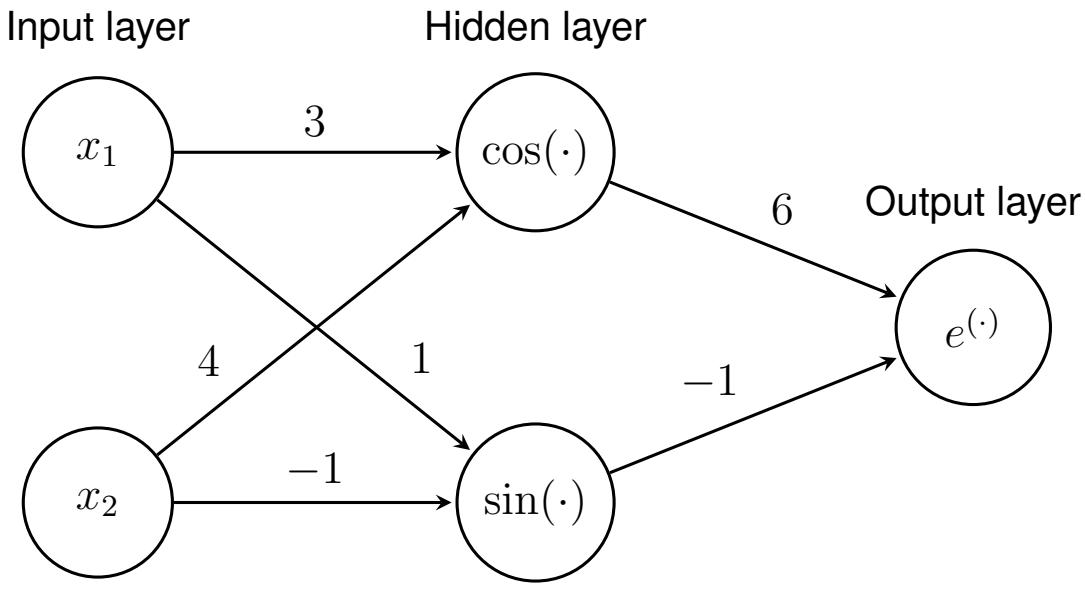
Computational graphs

- Notation: translations by a *bias* b are labels on the node. Usually omitted. $f(x_1, x_2) = e^{x_1 x_2} + 3(x_1 + x_2 - 2)^2 + 5$.
- Alternative: b is given as *weight* by an additional input $= 1$.



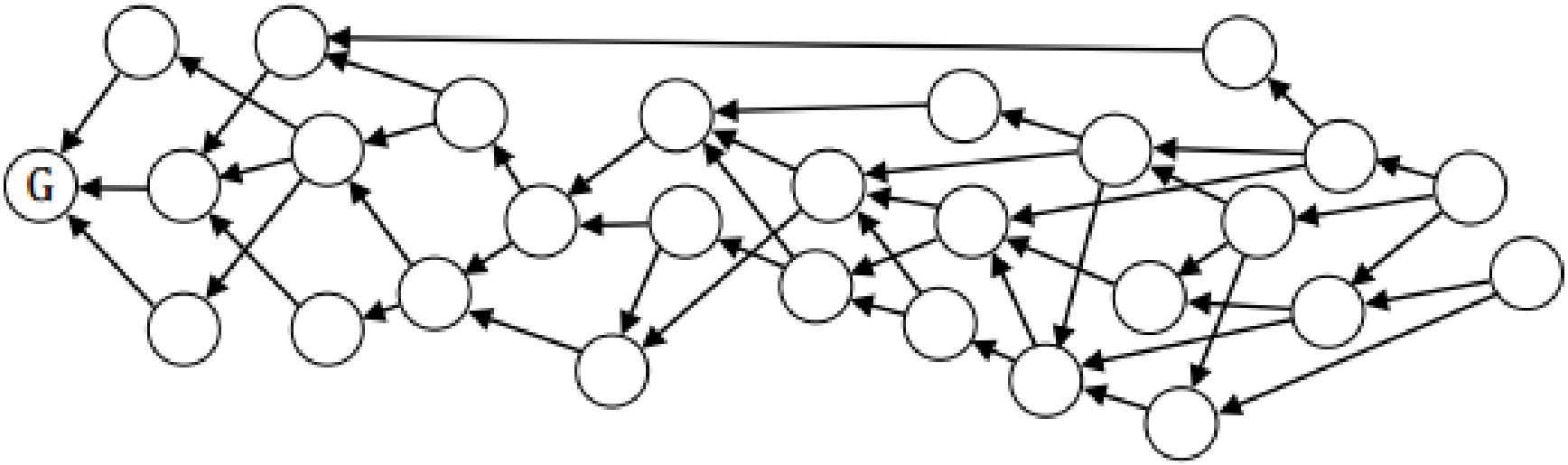
Computational graphs

- cos, sin, exp are called *activation functions*.



Neural networks

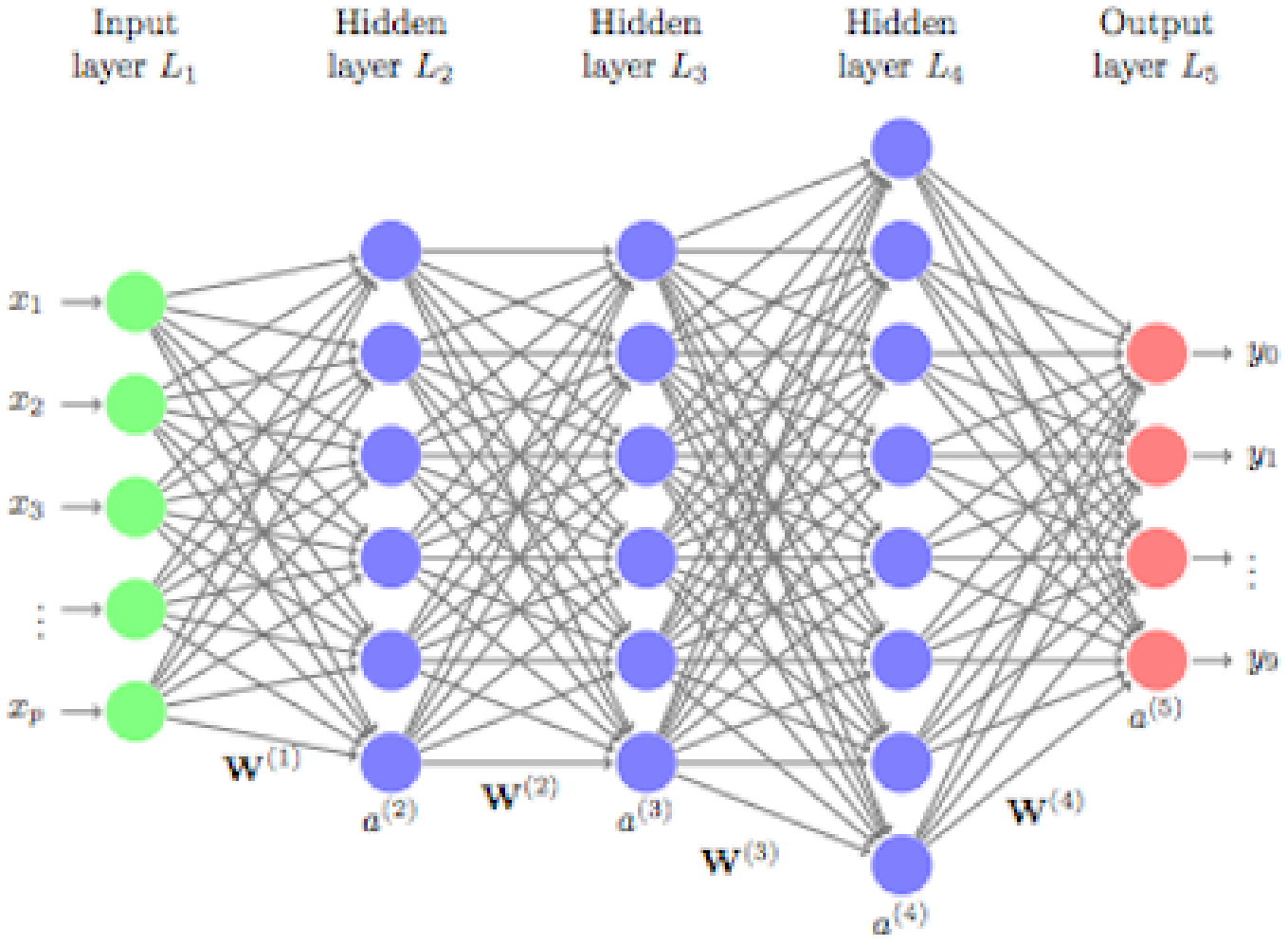
- Very general definition: a *neural network* (NN) is a function described by a computational graph.
- A *Feedforward Neural Network* (FNN) is a NN without cycles.



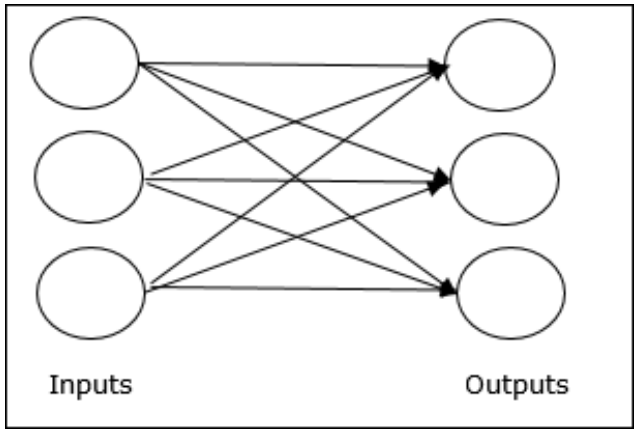
- In this picture, every node contains an *activation function*, but the two rightmost that are input nodes. The output node contains the activation function G.

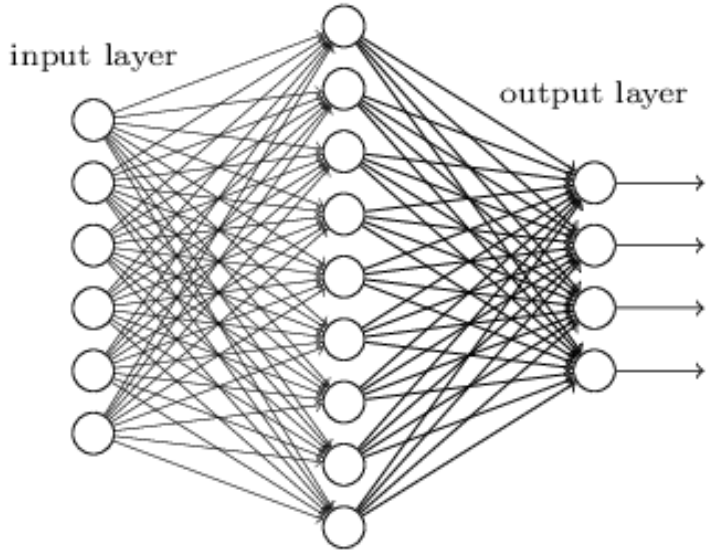
Layered FNN

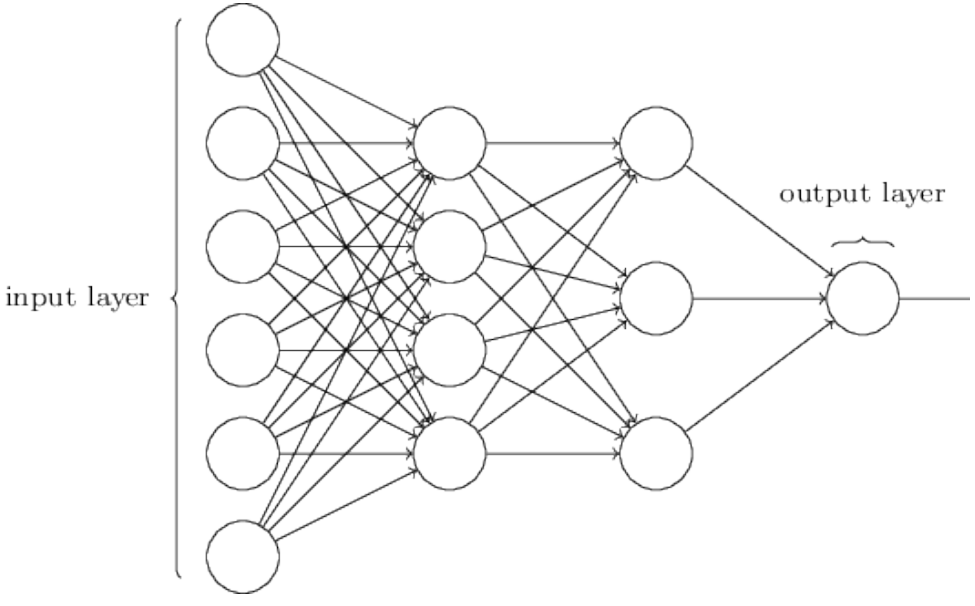
- Often, by FNN one means a *layered* FNN, that is, with well-defined layers: the *input layer*, the *output layer* and the *hidden layers*.

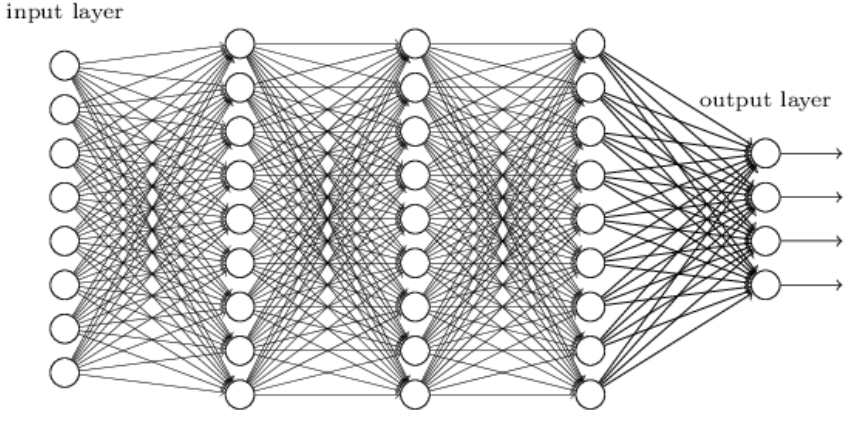


Layered FNN


Inputs
Outputs

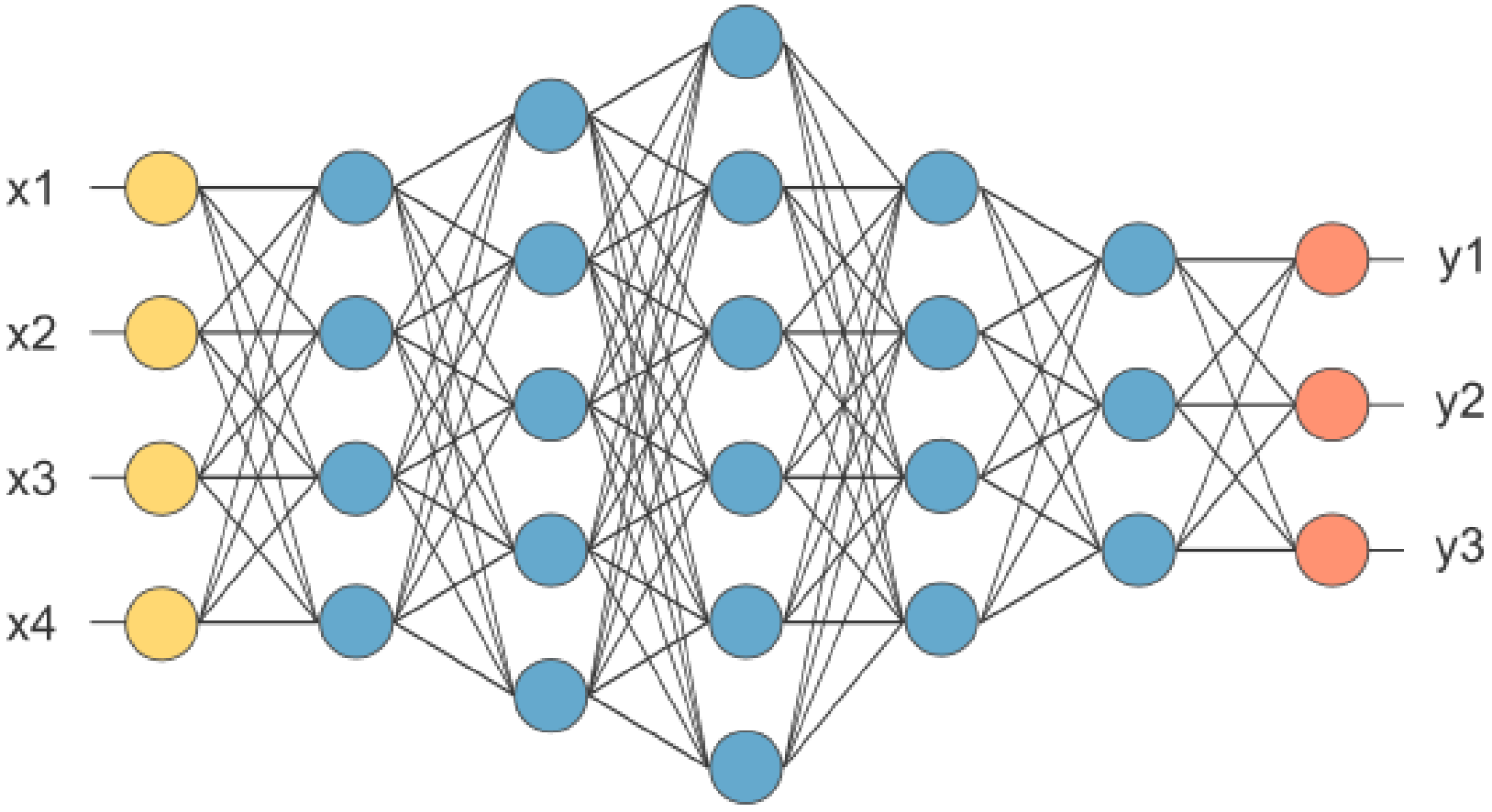

input layer
output layer


input layer
output layer

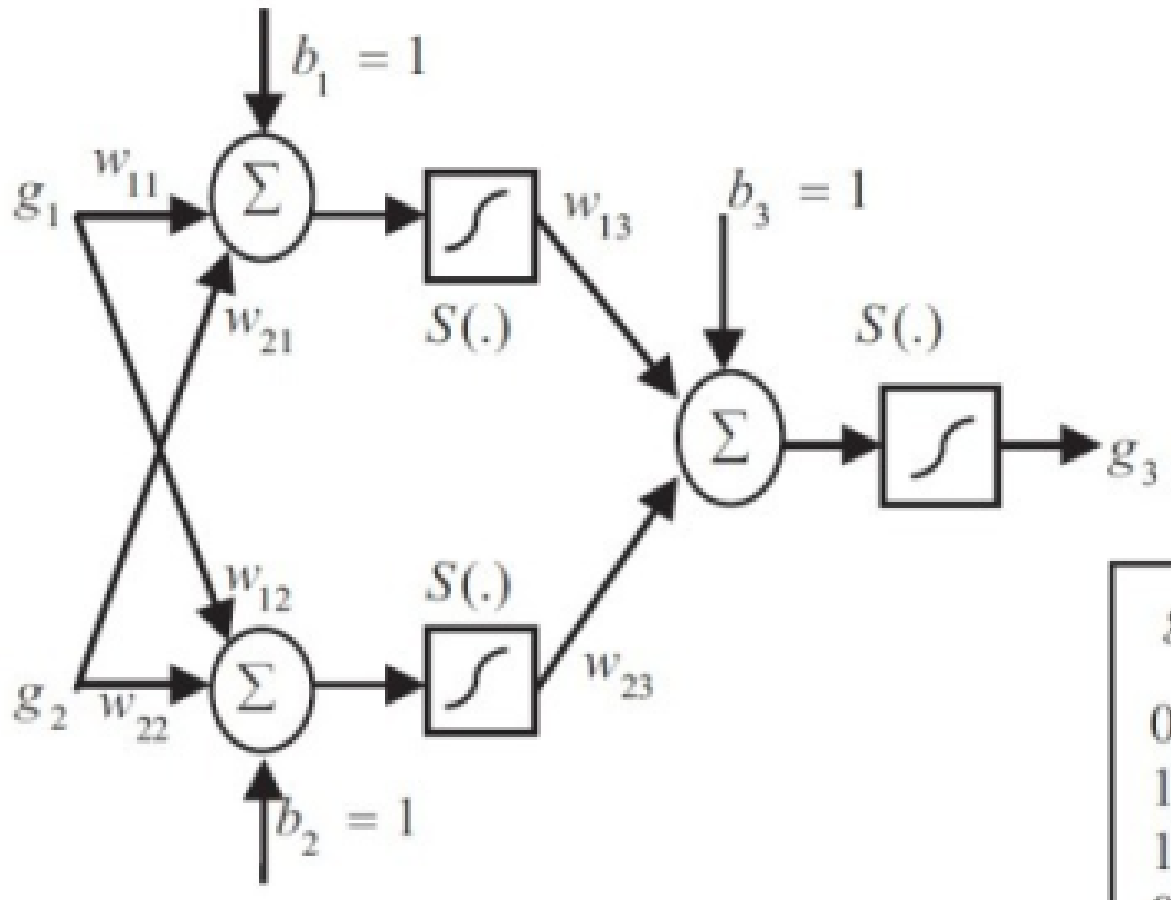

input layer
output layer

Layered FNN

- A *Deep NN* (DNN) is a layered NN with several hidden layers. Often, several means ≥ 3.

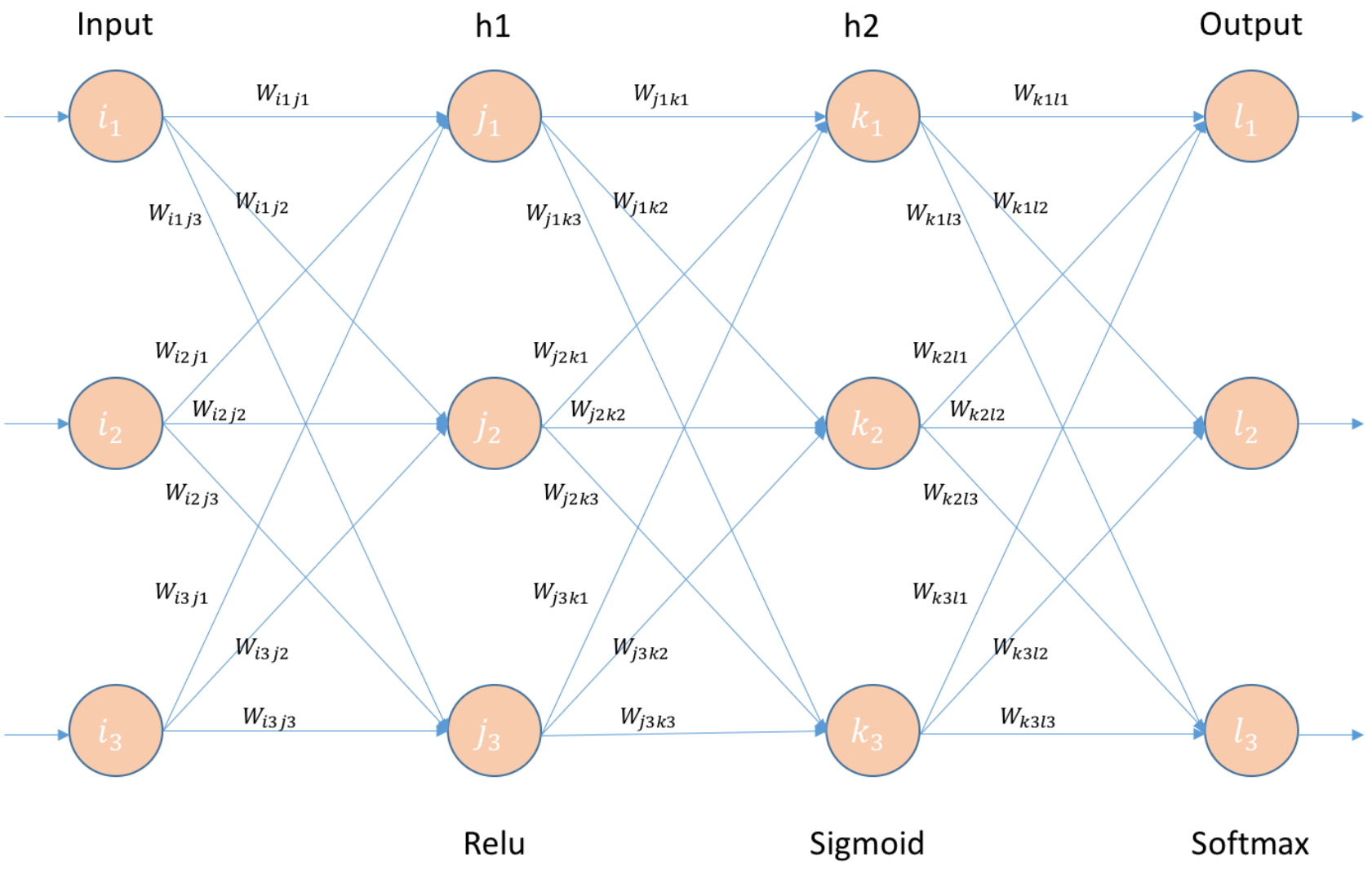


Examples

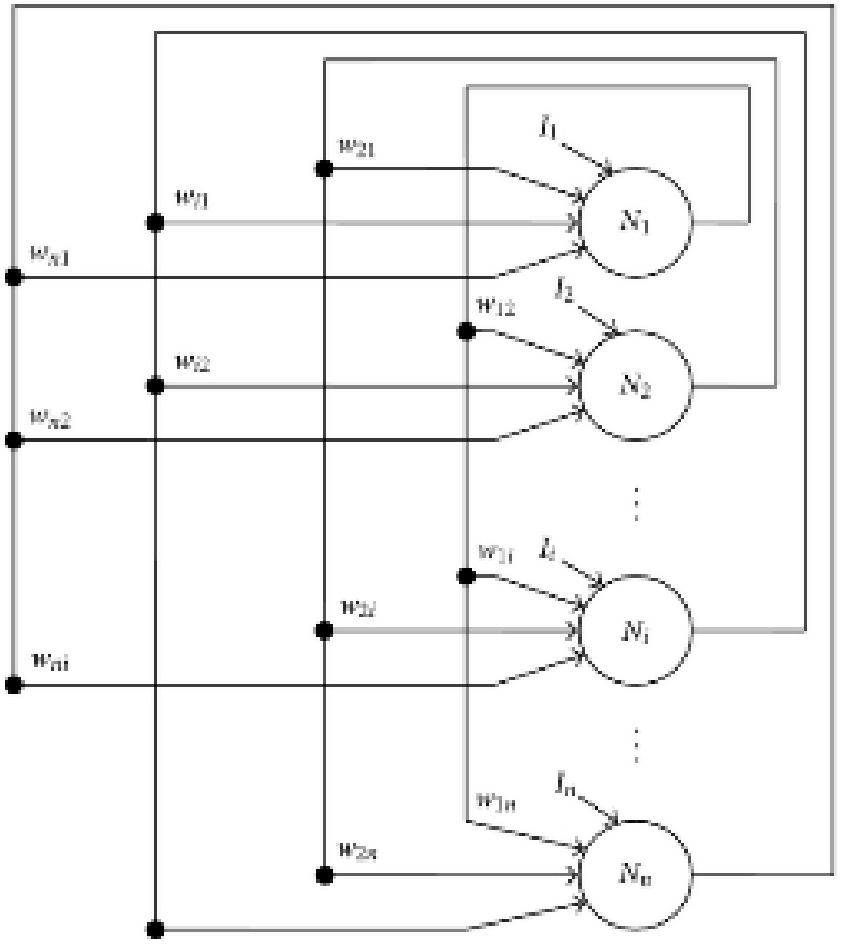


g_3	g_1	g_2
0	0	0
1	0	1
1	1	0
0	1	1

Examples

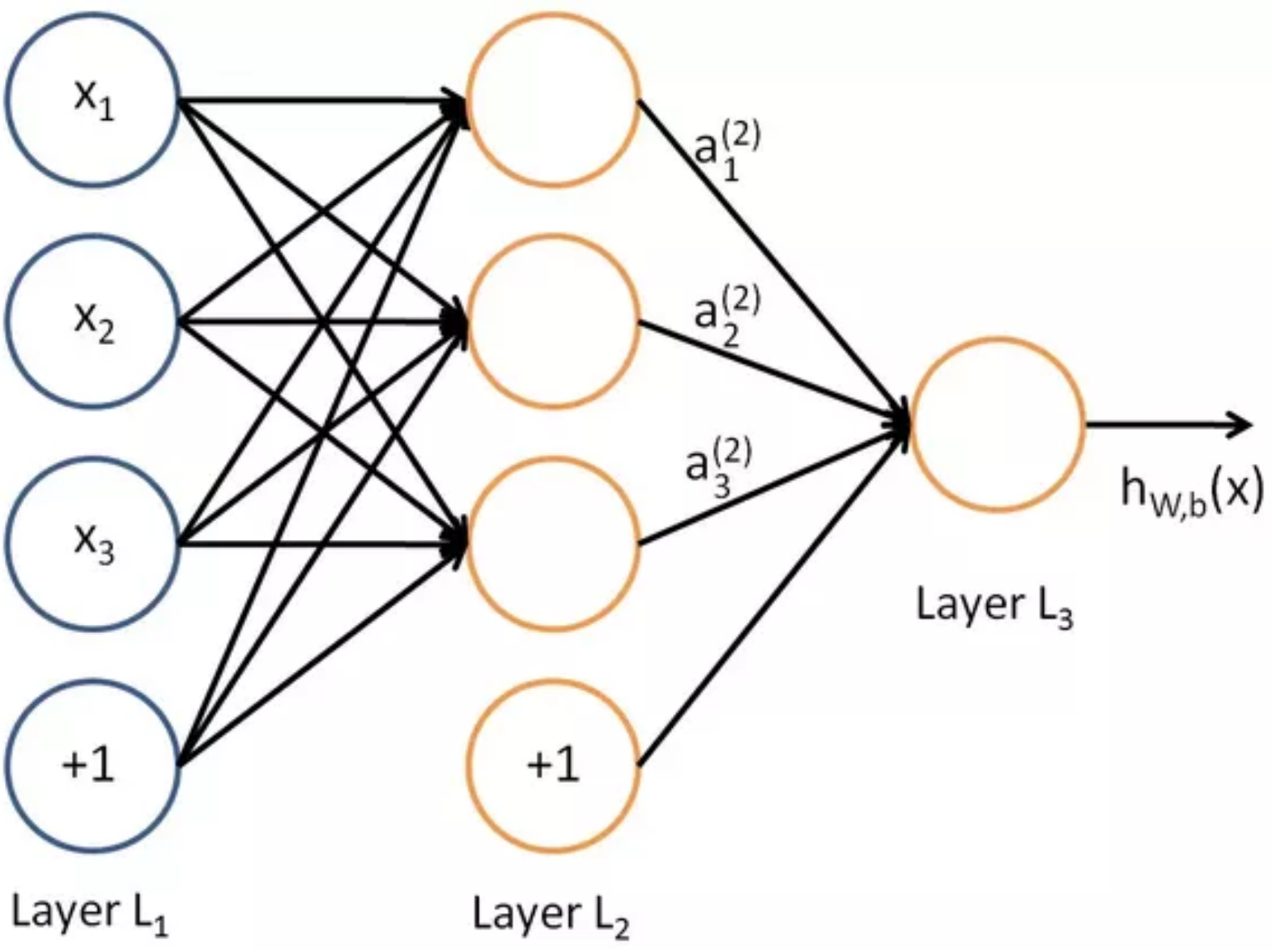

Input
h1
h2
Output
i_1
i_2
i_3
j_1
j_2
j_3
k_1
k_2
k_3
l_1
l_2
l_3
W_{i1j1}
W_{i1j2}
W_{i1j3}
W_{i2j1}
W_{i2j2}
W_{i2j3}
W_{i3j1}
W_{i3j2}
W_{i3j3}
W_{j1k1}
W_{j1k2}
W_{j1k3}
W_{j2k1}
W_{j2k2}
W_{j2k3}
W_{j3k1}
W_{j3k2}
W_{j3k3}
W_{k1l1}
W_{k1l2}
W_{k1l3}
W_{k2l1}
W_{k2l2}
W_{k2l3}
W_{k3l1}
W_{k3l2}
W_{k3l3}
Relu
Sigmoid
Softmax

Examples

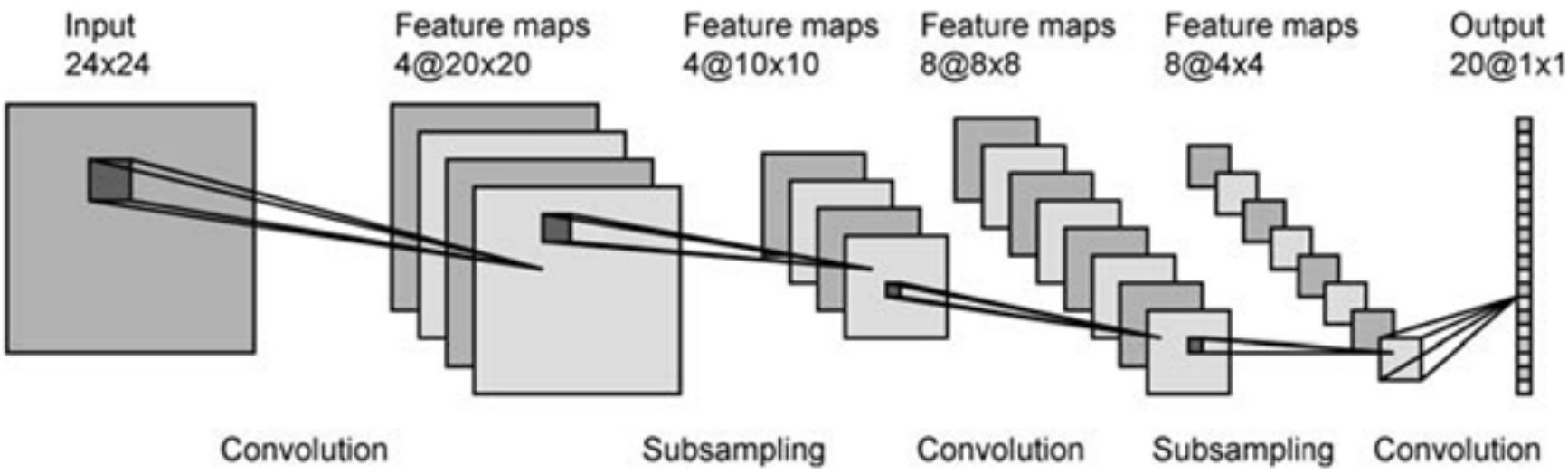


Hopfield Neural Network

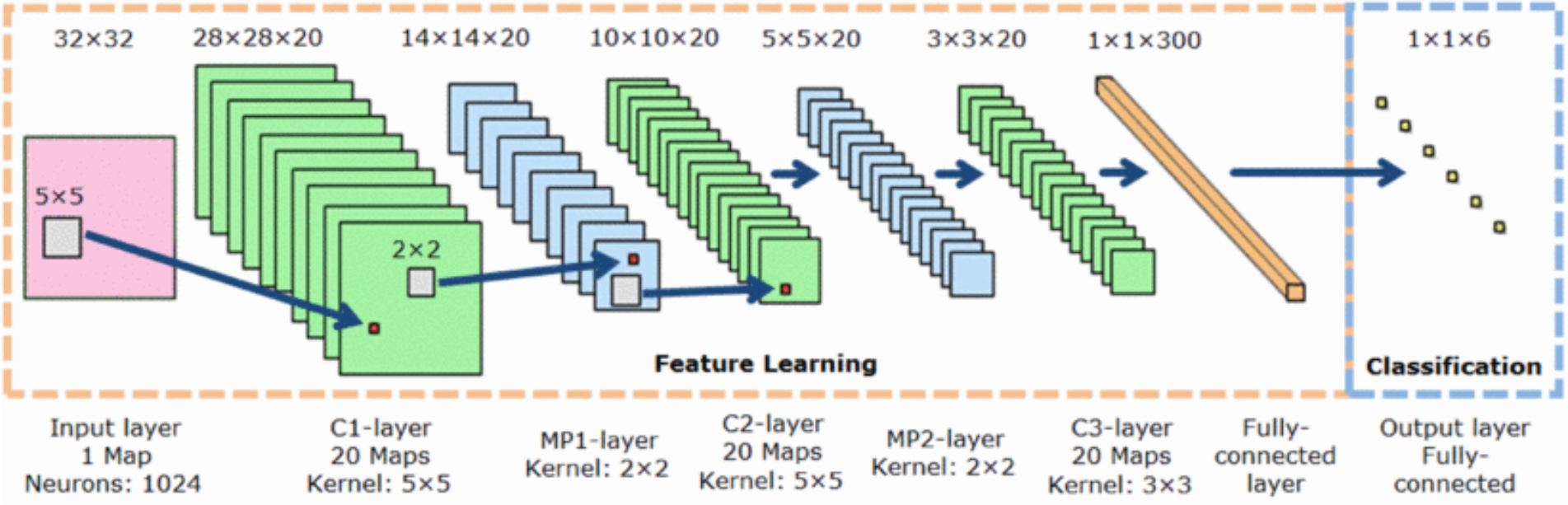
Examples

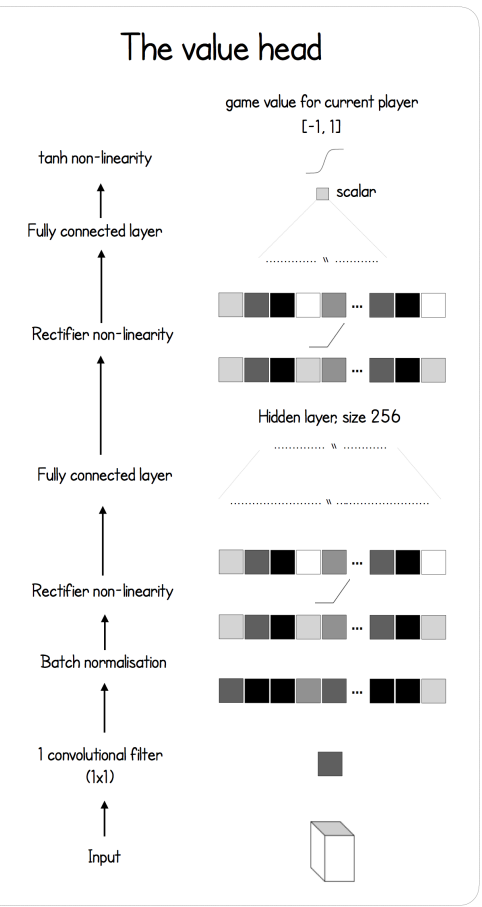


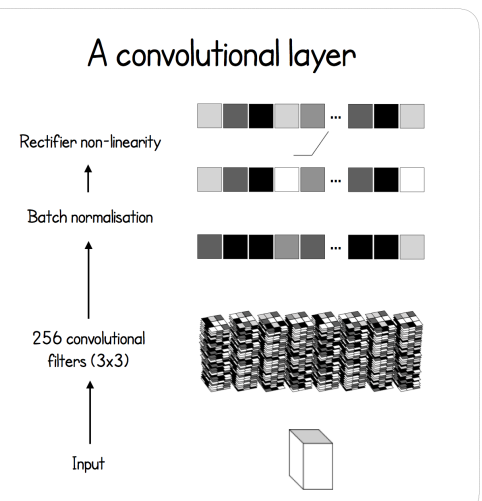
Examples

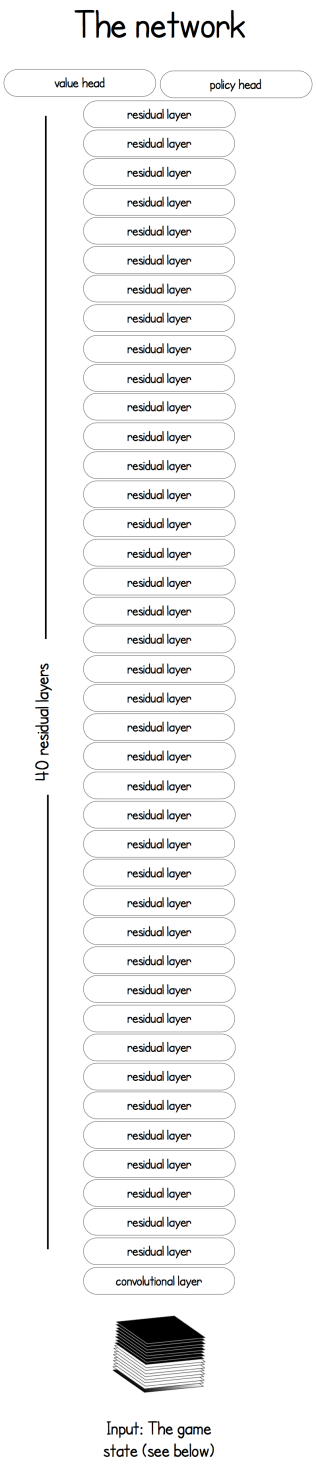

Input
24x24
Feature maps
4@20x20
Feature maps
4@10x10
Feature maps
8@8x8
Feature maps
8@4x4
Output
20@1x1
Convolution
Subsampling
Convolution
Subsampling
Convolution

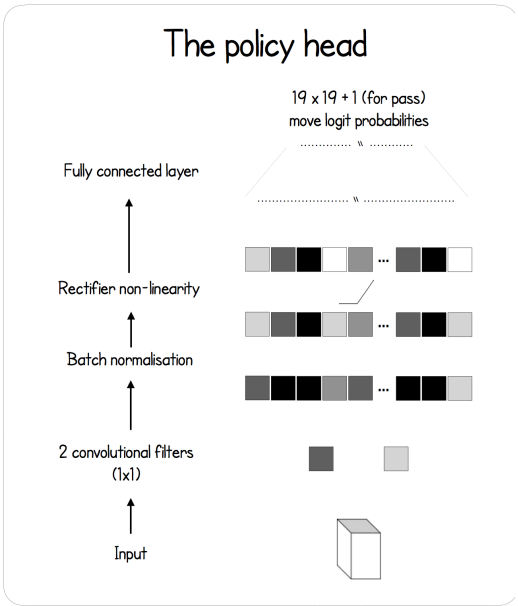
Examples

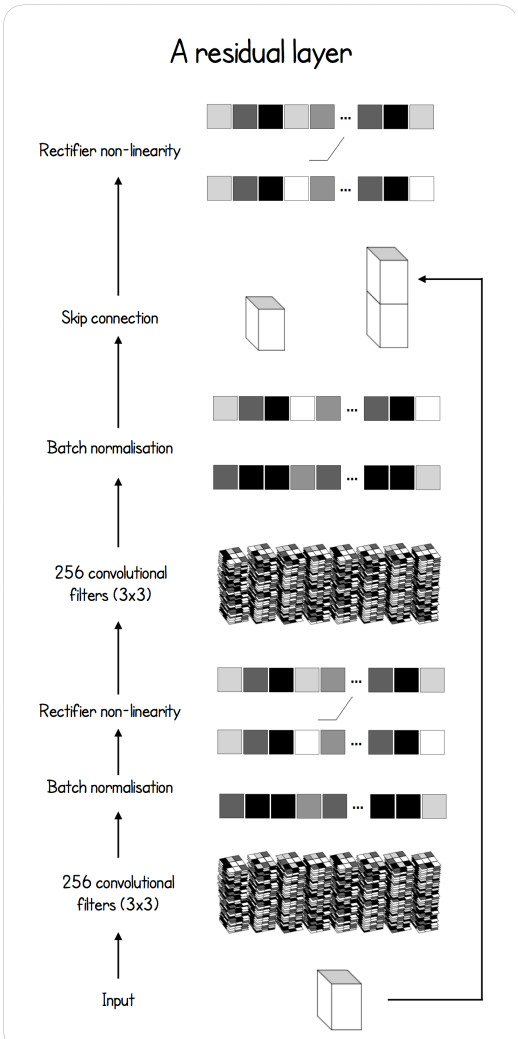

32×32
28×28×20
14×14×20
10×10×20
5×5×20
3×3×20
1×1×300
1×1×6
5×5
2×2
Feature Learning
Classification
Input layer
1 Map
Neurons: 1024
C1-layer
20 Maps
Kernel: 5×5
MP1-layer
Kernel: 2×2
C2-layer
20 Maps
Kernel: 5×5
MP2-layer
Kernel: 2×2
C3-layer
20 Maps
Kernel: 3×3
Fully-
connected
layer
Output layer
Fully-
connected


The value head
game value for current player
[-1, 1]
tanh non-linearity
scalar
Fully connected layer
Rectifier non-linearity
Hidden layer, size 256
Fully connected layer
Rectifier non-linearity
Batch normalisation
1 convolutional filter
(1x1)
Input


A convolutional layer
Rectifier non-linearity
Batch normalisation
256 convolutional
filters (3x3)
Input

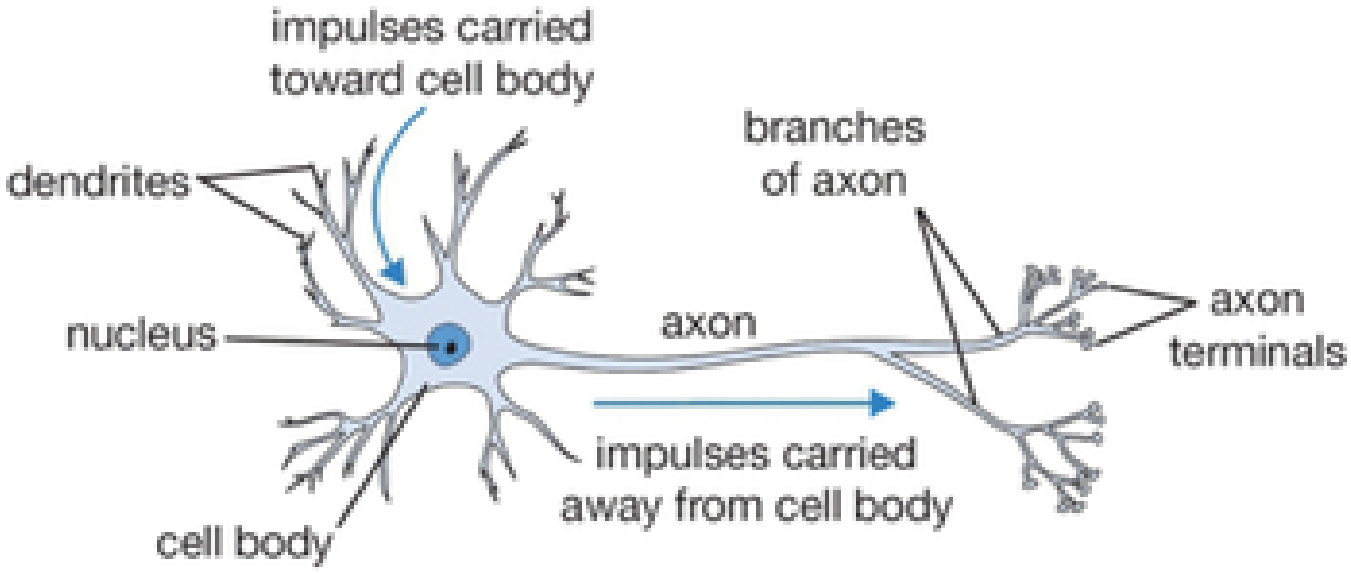

The network
value head
policy head
residual layer
40 residual layers
convolutional layer
Input: The game
state (see below)


The policy head
19 x 19 + 1 (for pass)
move logit probabilities
Fully connected layer
Rectifier non-linearity
Batch normalisation
2 convolutional filters
(1x1)
Input

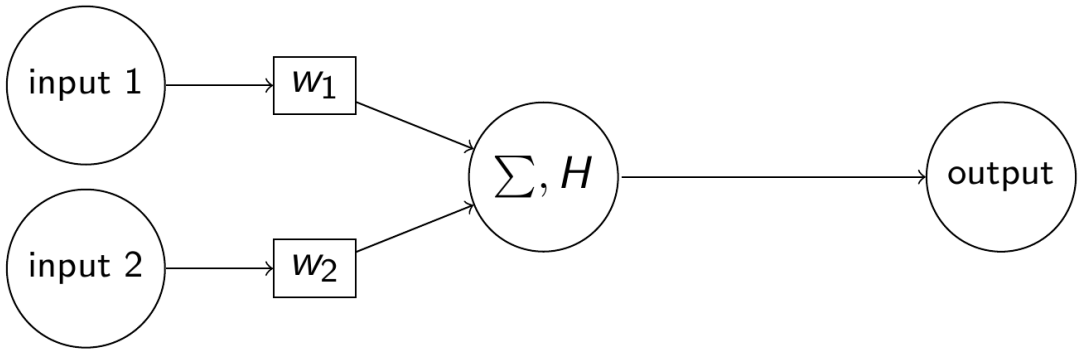

A residual layer
Rectifier non-linearity
Skip connection
Batch normalisation
256 convolutional
filters (3x3)
Rectifier non-linearity
Batch normalisation
256 convolutional
filters (3x3)
Input

Biological neuron/Perceptron

- The biological neuron.

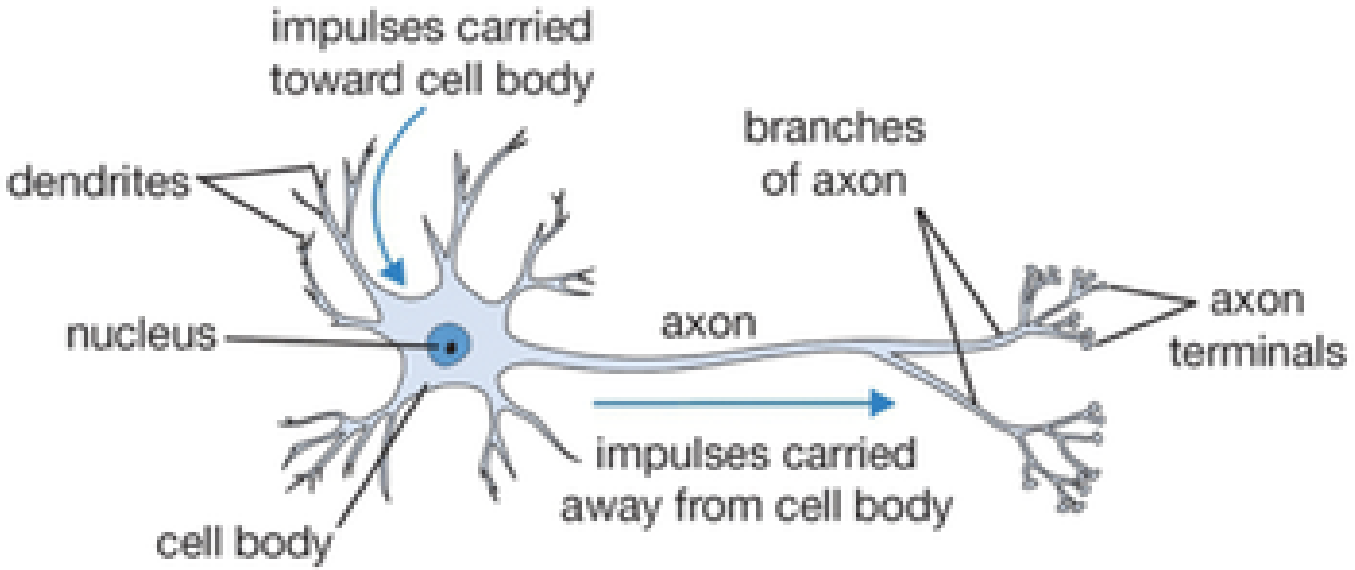


- The perceptron, detailed notation. H is the *Heaviside* activation function.

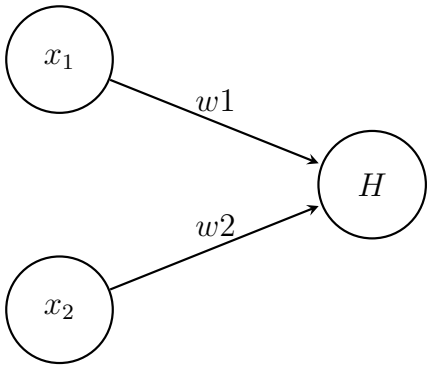


Biological neuron/Perceptron

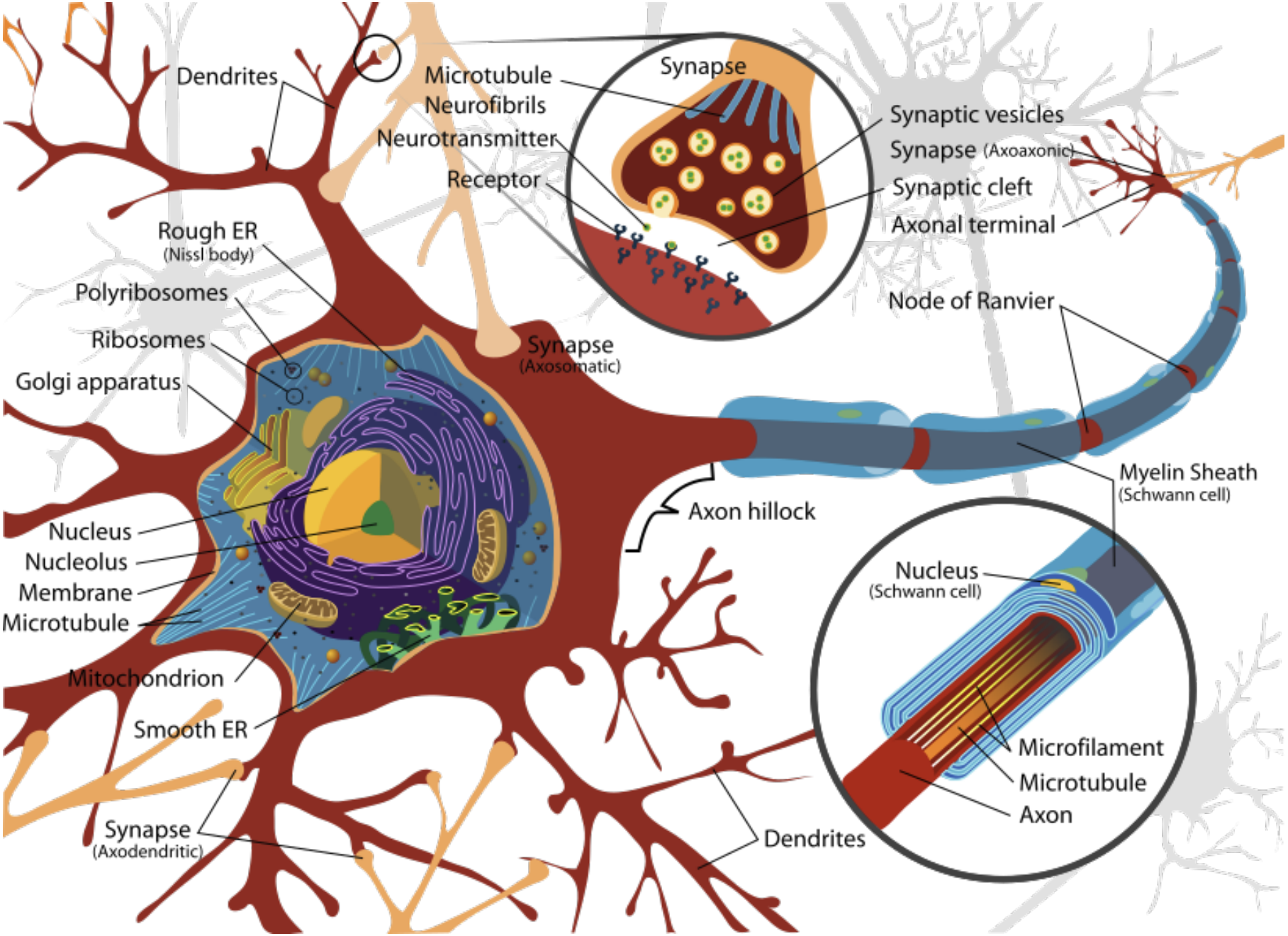
- The biological neuron.



- The perceptron, concise notation. *H* is the *Heaviside* activation function.



The real neuron



Summing up

- Learn these terms: AI, ML, DL, NN, FNN, CNN, DNN, activation, weight, bias, input layer, hidden layer, output layer, sigmoid, Heaviside, shallow NN, deep NN.
- NN are a way to represent parametric functions.
- Parameters are called *weights*. The bias is a weight weighting the input 1.
- In a NN, every node contains an activation function.
- Input->Hidden->Output. 1H is shallow. 3 or more H is deep.

2nd try: what is a NN?

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks
5. Looking for weights aka Learning
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

Hilbert's thirteenth problem

David Hilbert, 1900

Question: is the solution $f(a, b, c)$ of $x^7 + ax^3 + bx^2 + cx + 1 = 0$ a superposition of finitely many algebraic functions of at most 2 variables? It holds true for solutions of equations of degree up to 6.

Related question: is every continuous function $f(a, b, c)$ a superposition of finitely many continuous functions of at most 2 variables?

Hilbert's thirteenth problem

David Hilbert, 1900

Question: is the solution $f(a, b, c)$ of $x^7 + ax^3 + bx^2 + cx + 1 = 0$ a superposition of finitely many algebraic functions of at most 2 variables? It holds true for solutions of equations of degree up to 6.

Related question: is every continuous function $f(a, b, c)$ a superposition of finitely many continuous functions of at most 2 variables?

- Hilbert's conjecture: no. Why? Because he tried very hard without success.

Hilbert's thirteenth problem

David Hilbert, 1900

Question: is the solution $f(a, b, c)$ of $x^7 + ax^3 + bx^2 + cx + 1 = 0$ a superposition of finitely many algebraic functions of at most 2 variables? It holds true for solutions of equations of degree up to 6.

Related question: is every continuous function $f(a, b, c)$ a superposition of finitely many continuous functions of at most 2 variables?

- Hilbert's conjecture: no. Why? Because he tried very hard without success.

Exercise

Prove or disprove the following conjecture: every continuous function $f(a, b, c)$ is a superposition of finitely many functions (no continuity required) of at most 2 variables.

Kolmogorov and Arnold representation theorems

Theorem Kolmogorov, 1956

Let I^n be the n-dimensional cube, $n \geq 3$. Let $f : I^n \to \mathbb{R}$ be continuous. Then there exist continuous functions $g_r : \mathbb{R}^3 \to \mathbb{R}$, $\psi_{1r}, \psi_{2r} : I^{n-1} \to \mathbb{R}$, $r = 1, \ldots, n$ such that

$$f(x_1, \ldots, x_n) = \sum_{r=1}^{n} g_r(\psi_{1r}(x_1, \ldots, x_{n-1}), \psi_{2r}(x_1, \ldots, x_{n-1}), x_n)$$

Corollary

On I^n, any continuous function of $n \geq 4$ variables is a superposition of continuous functions of at most 3 variables.

Kolmogorov and Arnold representation theorems

Theorem Kolmogorov, 1956

Let I^n be the n-dimensional cube, $n \geq 3$. Let $f : I^n \to \mathbb{R}$ be continuous. Then there exist continuous functions $g_r : \mathbb{R}^3 \to \mathbb{R}$, $\psi_{1r}, \psi_{2r} : I^{n-1} \to \mathbb{R}$, $r = 1, \ldots, n$ such that

$$f(x_1, \ldots, x_n) = \sum_{r=1}^{n} g_r(\psi_{1r}(x_1, \ldots, x_{n-1}), \psi_{2r}(x_1, \ldots, x_{n-1}), x_n)$$

Corollary

Only any continuous function of $n \geq 4$ variables is a superposition of continuous functions of at most 3 variables.

Kolmogorov and Arnold representation theorems

Theorem Arnold, 1956

Let $f : I^3 \to \mathbb{R}$ be continuous. Then there exist continuous functions $g_{ij} : \mathbb{R}^2 \to \mathbb{R}$, $\psi_{ij} : I^2 \to \mathbb{R}$, $i, j = 1, 2, 3$, such that

$$f(x_1, x_2, x_3) = \sum_{i=1}^{3} \sum_{j=1}^{3} g_{ij}(\psi_{ij}(x_1, x_2), x_3)$$

Corollary, disproving Hilbert's conjecture

On I^n, any continuous function of $n \geq 3$ variables is a superposition of continuous functions of at most 2 variables.

Kolmogorov and Arnold representation theorems

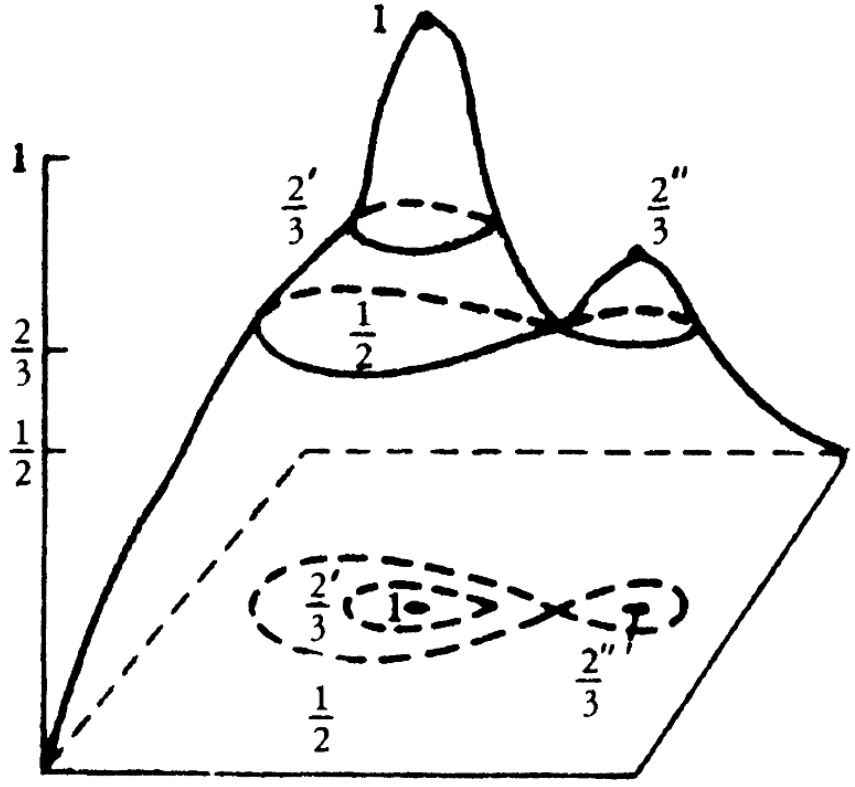


Figure 1. The level sets for 0, 1/2, 2/3 and 1 are designated, by the values to which they correspond. $2/3'$ and $2/3''$ are the two components of the level set for 2/3.

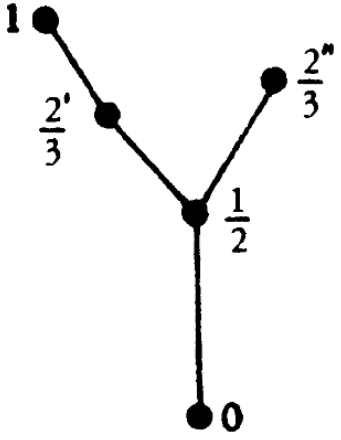


Figure 2. The tree of components of the level sets of the function of Figure 1. The components are designated as in Figure 1.

Kolmogorov and Arnold representation theorems

Theorem Arnold, 1956

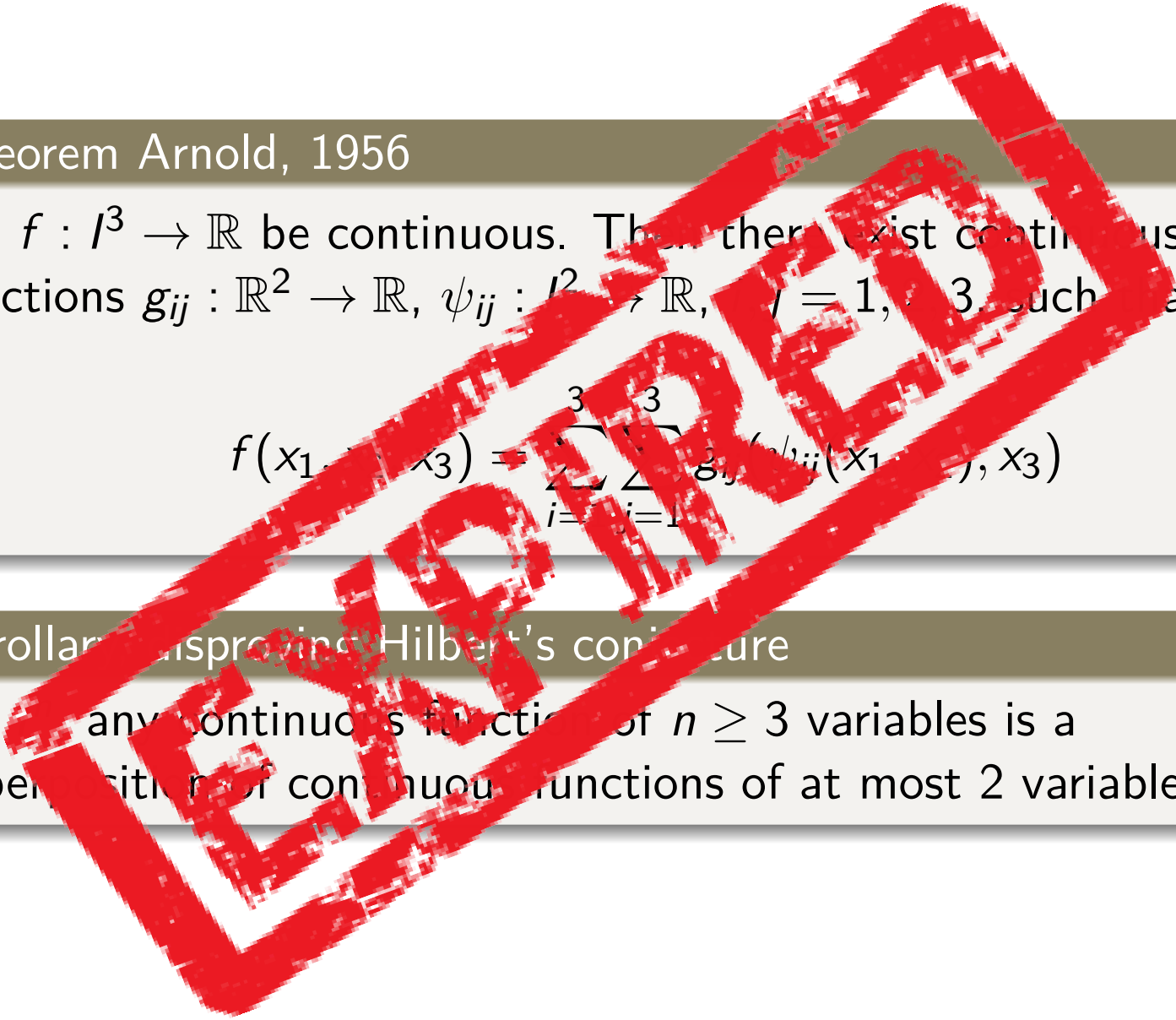
Let $f : I^3 \to \mathbb{R}$ be continuous. Then there exist continuous functions $g_{ij} : \mathbb{R}^2 \to \mathbb{R}$, $\psi_{ij} : I^2 \to \mathbb{R}$, $i, j = 1, 2, 3$, such that

$$f(x_1, x_2, x_3) = \sum_{i=1}^{3} \sum_{j=1}^{3} g_{ij}(\psi_{ij}(x_1, x_2), x_3)$$

Corollary, disproving Hilbert's conjecture

On I^n, any continuous function of $n \geq 3$ variables is a superposition of continuous functions of at most 2 variables.

Kolmogorov and Arnold representation theorems



Theorem Arnold, 1956

Let $f : I^3 \to \mathbb{R}$ be continuous. Th... ther... exist c...ti...us functions $g_{ij} : \mathbb{R}^2 \to \mathbb{R}$, $\psi_{ij} : I^2 \to \mathbb{R}$, ... $= 1, ... 3$. such ... at

$$f(x_1, \ldots, x_3) = \sum_{i=\ldots}^{3} \sum_{j=1}^{3} g_{ij}(\psi_{ij}(x_1, \ldots), x_3)$$

Corolla... ...ispro...ing Hilbe...'s con...ture

On... any ...ontinuo...s f...ncti... ...f $n \geq 3$ variables is a super...siti... ...f con...nuou... ...unctions of at most 2 variables.

Kolmogorov and Arnold representation theorems

Theorem Kolmogorov, 1957

Let $f : I^n \to \mathbb{R}$ be continuous. Then there exist continuous functions $g_i : \mathbb{R} \to \mathbb{R}$, $\psi_{ij} : I \to \mathbb{R}$, $i = 1, \ldots, 2n+1$, $j = 1, \ldots, n$, such that

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{2n+1} g_i\left(\sum_{j=1}^{n} \psi_{ij}(x_j)\right)$$

Kolmogorov and Arnold representation theorems

Theorem Kolmogorov, 1957

There exist continuous functions $\psi_{ij} : I \to \mathbb{R}$, $i = 1, \ldots, 2n+1$, $j = 1, \ldots, n$ such that for every continuous $f : I^n \to \mathbb{R}$ there exist continuous functions $g_i : \mathbb{R} \to \mathbb{R}$ such that

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{2n+1} g_i\left(\sum_{j=1}^{n} \psi_{ij}(x_j)\right)$$

The functions $\psi_{ij} : I \to \mathbb{R}$ can be chosen independently on f.

Kolmogorov and Arnold representation theorems

Theorem Sprecher, 1965

There exist continuous functions $\psi_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, 2n+1$, and real numbers λ_j, $j = 1, \ldots, n$, such that for every continuous $f : I^n \to \mathbb{R}$ there exist a continuous function $g : \mathbb{R} \to \mathbb{R}$ such that

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{2n+1} g\left(\sum_{j=1}^{n} \lambda_j \psi_i(x_j)\right)$$

Corollary

On I^n, any continuous function is a superposition of continuous functions of one variable and addition. Moreover, all the functions in the superposition but one does not depend on f.

Exercise

Consider the map $\phi : I^n \to \mathbb{R}^{2n+1}$ given by

$$\phi(x_1, \ldots, x_n) = (\sum_{j=1}^{n} \lambda_j \psi_1(x_j), \ldots, \sum_{j=1}^{n} \lambda_j \psi_{2n+1}(x_j))$$

Prove that $\phi : I^n \to \phi(I^n)$ is a homeomorphism.

Kolmogorov Theorem: useful or not?

Arnold, 1958

[. . .] are of purely theoretical interest, since they use essentially non-smooth functions; therefore for practical purposes these representations are useless.

Lorentz, 1962

[. . .] is very important in principle. Will it have useful applications? Theoretically, one could hope to derive by means of it results concerning functions of several variables from corresponding results about functions of one variable.

Kolmogorov and neural networks

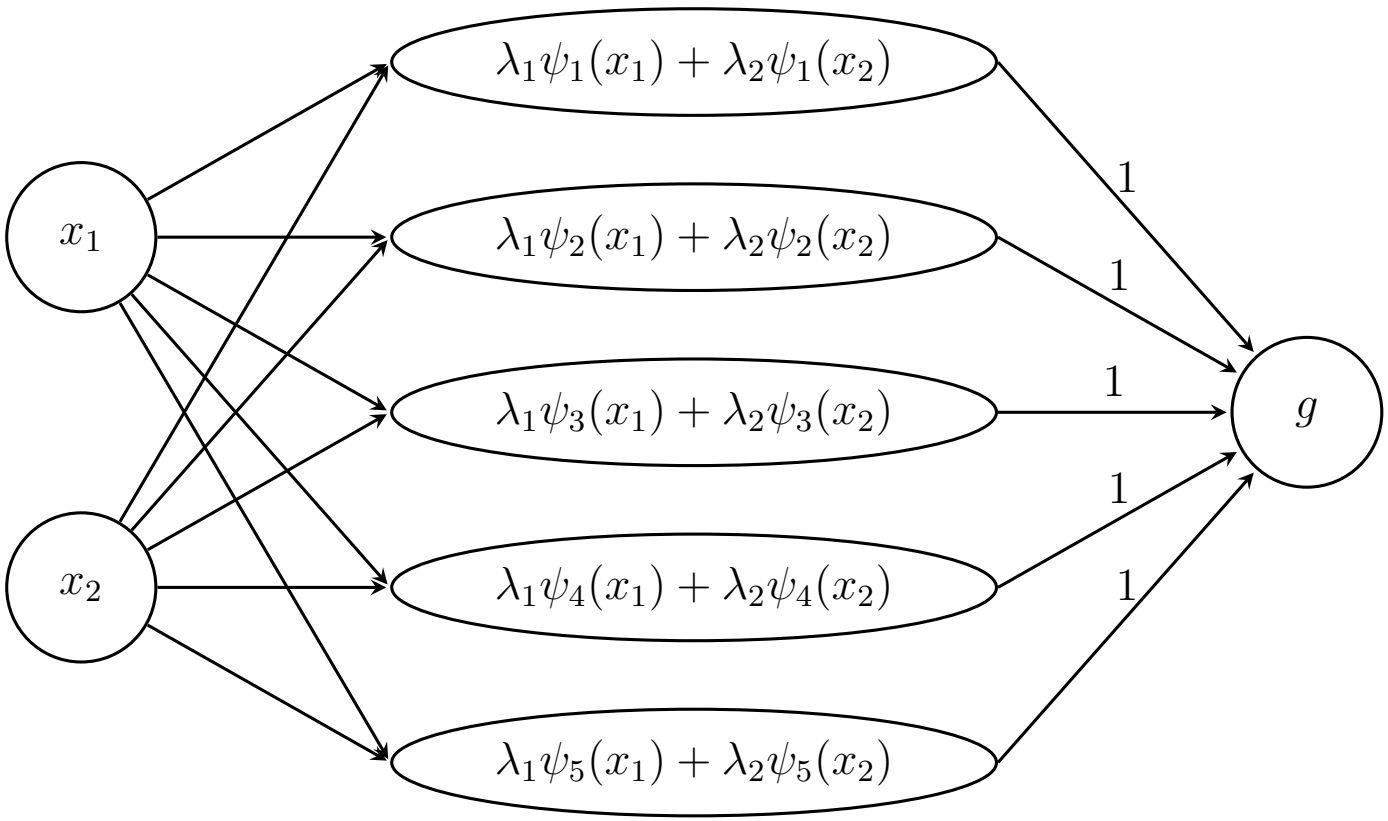
Hecht-Nielsen, 1987

[. . .] In mathematical terms, no one has found a significant use for it. The point of this paper is that this is not the case in neurocomputing!

Kolmogorov and neural networks

Hecht-Nielsen, 1987

[. . .] In mathematical terms, no one has found a significant use for it. The point of this paper is that this is not the case in neurocomputing!



Universal Approximation Theorems

- From the point of view of actual computation, functions in Kolmogorov and Hecht-Nielsen Theorems suffer from several problems.
- g depends on f, thus cannot be written in a suitable parametric form.
- The ψs are ugly: their graph is a fractal.

Cybenko, 1989

Let σ be a sigmoidal function. The set of functions $F : I^n \to \mathbb{R}$ given by

$$\{F(x_1, \ldots, x_n) = \sum_{j=1}^{N} \lambda_j \sigma(\mu_{1j} x_1 + \cdots + \mu_{nj} x_n + b_j), N \in \mathbb{N}, \mu_{ij}, \lambda_j \in \mathbb{R}\}$$

is dense in $C(I^n)$.

Universal Approximation Theorems

Hornik, 1991

Let σ be a bounded non-constant function. The set of functions $F : \mathbb{R}^n \to \mathbb{R}$ given by

$$F(x_1, \ldots, x_n) = \sum_{j=1}^{N} \lambda_j \sigma(\mu_{1j} x_1 + \cdots + \mu_{nj} x_n + b_j)$$

is dense in $L^p(\mu)$ for any finite measure μ on $\mathbb{R}^n$.

Hornik, 1991

Let σ be a continuous, bounded non-constant function. The set of functions $F : R^n \to \mathbb{R}$ given by

$$F(x_1, \ldots, x_n) = \sum_{j=1}^{N} \lambda_j \sigma(\mu_{1j} x_1 + \cdots + \mu_{nj} x_n + b_j)$$

is dense in $C(K)$, for any compact $K \subset \mathbb{R}^n$.

Summing up

- Learn these terms: ReLU, universal approximation theorem.
- Arnold was very unlucky. But he recovered very well.
- A shallow NN with sigmoidal or ReLU activation function can approximate any continuous function on compact set. Bounds on number of nodes are known
- For real problems, the number of nodes is way too high!

This wasn't easy

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks ✓
5. Looking for weights aka Learning
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

Learning: useless case

Exercise

Write a neural network approximating e^x in a neighbourhood of $x = 3$.

Learning: useless case

Exercise

Write a neural network approximating e^x in a neighbourhood of $x = 3$.

Solution

Taylor polynomial.

Is this NN useful?

Learning: useless case

Exercise

Write a neural network approximating e^x in a neighbourhood of $x = 3$.

Solution

Taylor polynomial.

Is this NN useful? Not really: we need an automagic way to compute parameters! That is, we need a *learning algorithm*!

Supervised Learning (SL)

- Goal: find an unknown map $f(x) = t$ from a set of observations (x_i, t_i).
- Supervised: the teacher knows the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher.
- *Classification*: when t is a finite set, that is, a "type".
- *Regression*: when $t \in \mathbb{R}$.

Question

Do you know any famous supervised learning algorithm?

Unsupervised learning

- You have only inputs x_i. No correct answer t_i, and thus no teacher.
- Goal: to model the underlying structure or distribution in the data in order to learn more about the data.
- Clustering: you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- Association: you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Reinforcement Learning (RL)

- A greedy teacher!
- An agent receives rewards by performing correctly and penalties by performing incorrectly.
- The agent learns by maximizing its reward and minimizing its penalty.
- Can be used together with supervised learning: the agent produces the data (x_i, t_i).

Question

Do you know any famous software where reinforcement learning is used?

Reinforcement Learning (RL)

- A greedy teacher!
- An agent receives rewards by performing correctly and penalties by performing incorrectly.
- The agent learns by maximizing its reward and minimizing its penalty.
- Can be used together with supervised learning: the agent produces the data (x_i, t_i).

Question

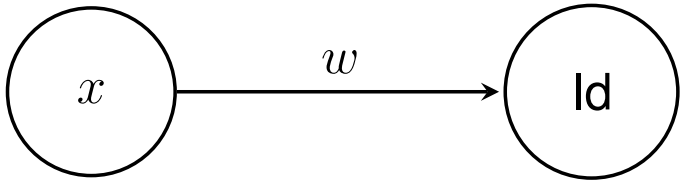
Do you know any famous software where reinforcement learning is used? You will!

Learning: the “trivial” case

- Consider the simplest NN ever! Call F the function it defines.

Learning: the "trivial" case

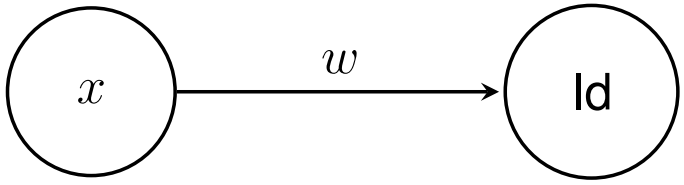
- Consider the simplest NN ever! Call F the function it defines.



- The *network function* is the parametric function $F(x) = wx + b$.

Learning: the "trivial" case

- Consider the simplest NN ever! Call F the function it defines.



- The *network function* is the parametric function $F(x) = wx + b$.
- Consider a function $f : \mathbb{R} \to \mathbb{R}$ that you would like to approximate.
- You don't know f, just its values $t = f(x)$ on a finite set. $T = \{(x_1, t_1), \ldots, (x_n, t_n)\}$ is the *training set*.
- Look for *the best* function F approximating f on T: we need a method to say when F is "the best".

Learning: the “trivial” case

- Choose an *error function* to minimize: $E = \sum_{i=1}^{n} |F(x_i) - t_i|^2$

Learning: the "trivial" case

- Choose an *error function* to minimize: $E = \sum_{i=1}^{n} |F(x_i) - t_i|^2$

Yes, you are right

This is linear regression, we are looking for the best-fitting line through T.

Learning = find the best w and b.

Learning: the "trivial" case

- Choose an *error function* to minimize: $E = \sum_{i=1}^{n} |F(x_i) - t_i|^2$

Yes, you are right

This is linear regression, we are looking for the best-fitting line through T.

Learning = find the best w and b.

- Training set $\rightarrow$ Validation set $\rightarrow$ Test set.
- Possible problem of *overfitting*: F is very good on the training set, but very bad on the test set.

Question

How can this happen?

Learning: Gradient Descent (GD)

- Learning: find the weights w and the biases b that minimize the error function E.
- With non-linear activation functions and big training data, explicitly solving $\nabla E(w) = 0$ is not feasible. We need numerical methods.
- We use the *gradient descent* algorithm (*optimizer*): start from a random w_0; at step i, update $w_i = w_{i-1} - \gamma \nabla E(w_{i-1})$. The *hyperparameter* γ is called *learning rate*.
- Stop when the error $E(w_{i-1})$ is sufficiently small.

Question

Do you see any bottleneck in the above algorithm?

Learning: BackPropagation (BP)

- We need an efficient way to compute ∇E
- This efficient way is the famous *backpropagation algorithm*.

The backpropagation algorithm explained

Graphical approach: reduction to a graph labeling problem. More general and easier to follow. Really cool!

Blackboard time!

Learning: BackPropagation (BP)

- The proof with the graph labeling works for every FNN, not only the layered ones.
- For a layered FNN, BP can be optimized.
- BP can be adapted to *recurrent NN* (RNN) by an unfolding strategy: this is called *BP through time* (BPTT).

Learning: BackPropagation (BP)

- The addition is the only integration function preserving the *locality* of BP.

Exercise

Give an example of a NN with multiplication used instead of addition as integration function. Explain what is the meaning of "locality of BP" and show that multiplication as integration function doesn't preserve locality of BP.

Exercise *

Prove that addition is the only integration function preserving locality of BP.

Backpropagation: terminology

- *Epoch*: one cycle of F+BP for the whole training set.
- *Batch training* or *offline training*: F+BP with the whole training set.
- *Mini-batch training*: F+BP with subsets of the training set.
- *Batch size*: number of training items in the mini-batch.
- *Iteration*: one cycle of F+BP.
- *Online training*: F+BP with one training item.

Backpropagation: terminology

Example

If card(training set) $= 1000$ and batch size $= 250$, then 1 epoch $=$ 4 iterations. If we train for 10 epochs, we are doing 40 iterations of training.

Underfitting, overfitting and capacity

Capacity of a NN

Representational power, that is, how many functions the NN can approximate. Low capacity risks underfitting, high capacity risks overfitting.

- *Underfitting*: when the network function doesn't fit the training set T. We say the NN *doesn't learn*, meaning that we are not able to find the correct weights.
- Improve T or the capacity to reduce underfitting.

Example

Blackboard.

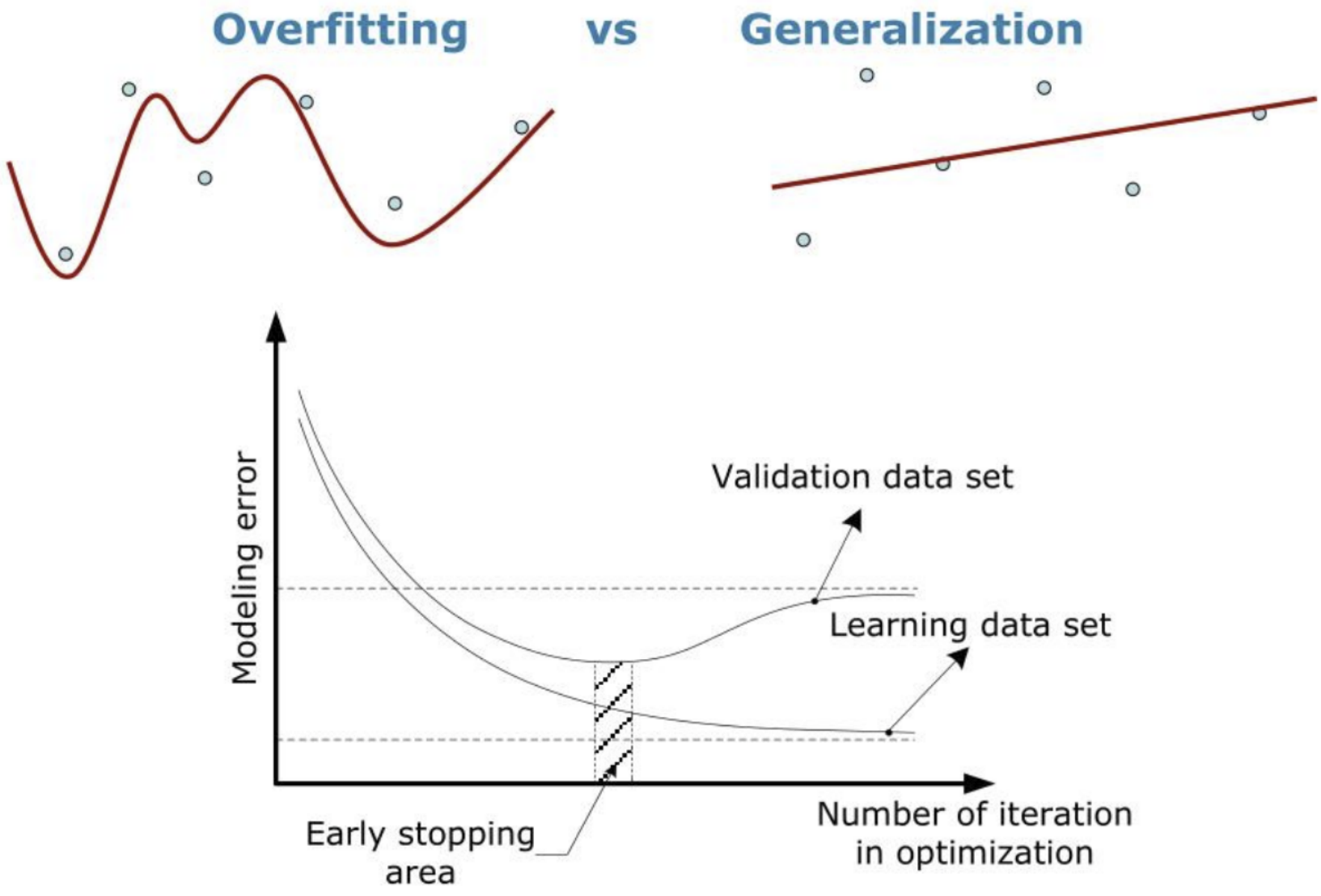
Underfitting, overfitting and capacity

- After every iteration, we check the NN against a *validation set*.
- The validation set is used to tune the *hyperparameters*, and to decide when to stop training.
- When the learning phase ends, we check the trained NN against a *test set*.
- A NN *generalizes* if the error against the test set is small.
- If not: *overfitting*.

Example

Blackboard.

Training-validation graph



Controlling overfitting

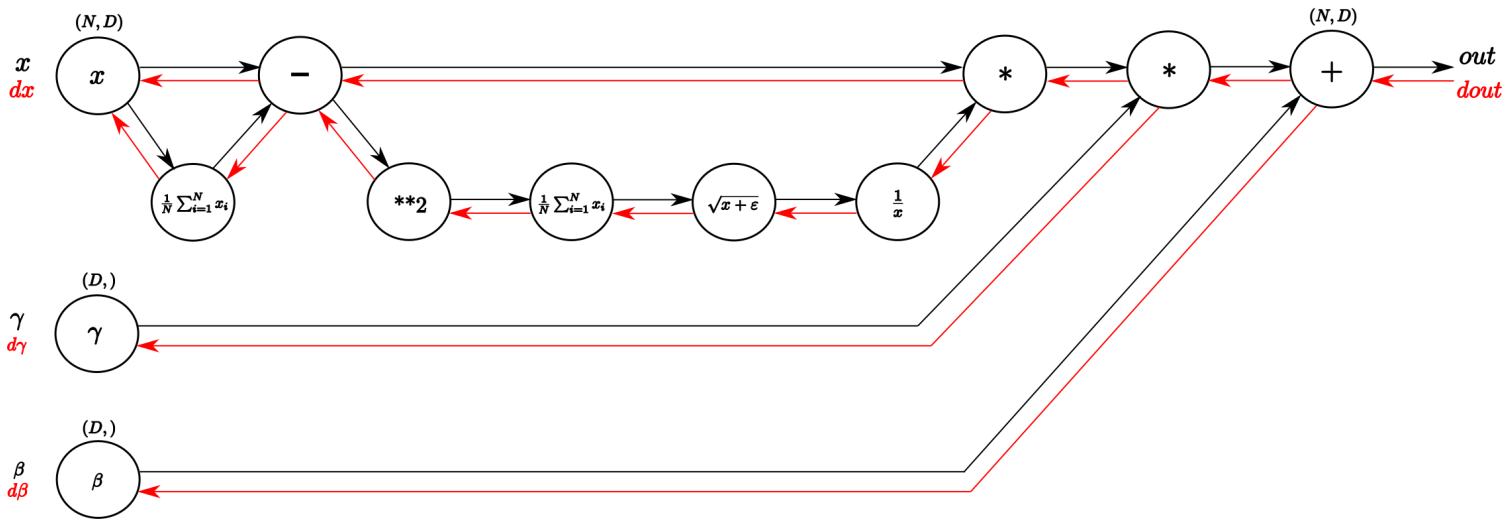
- T must be representative (statistical methods for choosing T).
- Adequate *capacity*: too few or too much is bad.
- l^2-*Regularization*: force the weights to stay small during training with $E = \cdots + \lambda \sum_i |w_i|^2$.
- λ is the *regularization* hyperparameter. It measures the relative importance of the regularization versus the error.
- Idea: make the network prefer to learn small weights.
- in this way, the model is less sensitive to noise during gradient descent.

Controlling overfitting: other techniques

- l^1-regularization: just a different metric. Why? It's better. Sometimes.
- *Dropout*: at each epoch or mini-batch, depending on the optimizer, drop half of the nodes. Sounds crazy, but works.
- *Data augmentation*: artificially increase the training set size (shift in every direction, grayscale conversion, fourier transform, small rotations, blur, sharpening, etc. . .).

Controlling overfitting: other techniques

- *Batch normalization* (BN): 2015, a breakthrough after ReLU, today used in almost every NN. A BN layer is put between hidden layers to normalize data. It improves training and stability.



Summing up

- Learn these terms: SL, RL, error function, loss function, cost function, training set, validation set, test set, holdout set, underfitting, overfitting, GD, optimizer, learning rate, BP, RNN, BPTT, epoch, offline training, batch training, mini-batch training, batch, online training, iteration, capacity, to generalize, regularization, dropout, data augmentation, BN.
- Learning for a NN consists in a sequence of small updates of weights (iterations), made using GD+BP.
- The training-validation graph is a pervasive tool when training a NN. One uses it to check that a) the NN is learning and b) the NN is not overfitting.

Very nice proof of BP!

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks ✓
5. Looking for weights aka Learning ✓
6. Deep neural networks
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

Deep NN: why?

- Shallow and 2-layers NN can in theory approximate every function: why one should use more layers?
- Divide et Impera: first layers face a small portion of the problem (divide), and the final layers gather all the information together (impera).

Example

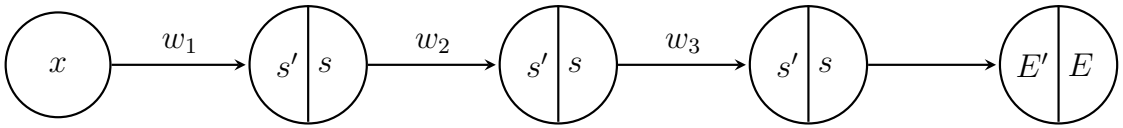
Computing the parity of a set of bits. A shallow NN requires a direct computation. A DNN computes the parity of pairs of bits, then use those results to compute the parity of pairs of pairs of bits, and so on.

The vanishing gradient problem for DNN

Experimental fact

Early layers get stuck during training. Often, already during the first epochs of training.

- This means that ∇E becomes smaller and smaller during BP. Why?
- Reasons: activation function, chain rule and choice for weights initialization.



The vanishing gradient problem for DNN

- $s'(0) = 1/4$ is a maximum.
- Initial weights are chosen with a standard normal distribution, thus usually $|w_i| < 1$.
- The factors in BP will usually satisfy $|w_i s'(o_i)| < 1/4$. Thus, at first layer we have $|\partial E/\partial b_1| < 1/64$: the gradient is vanishing!
- Sometimes the gradient can explode, instead. In general, the multiplications in the chain rule make BP unstable.

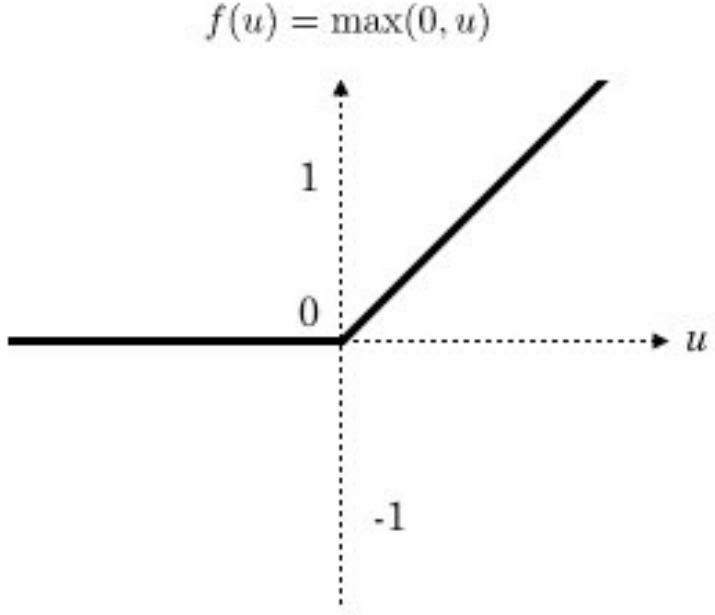
The vanishing gradient problem for DNN

Solution 1 Hinton, Osindero, Teh, 2006

Ad hoc training, before applying BP.

Solution 2 Glorot, Bordes, Bengio, 2011

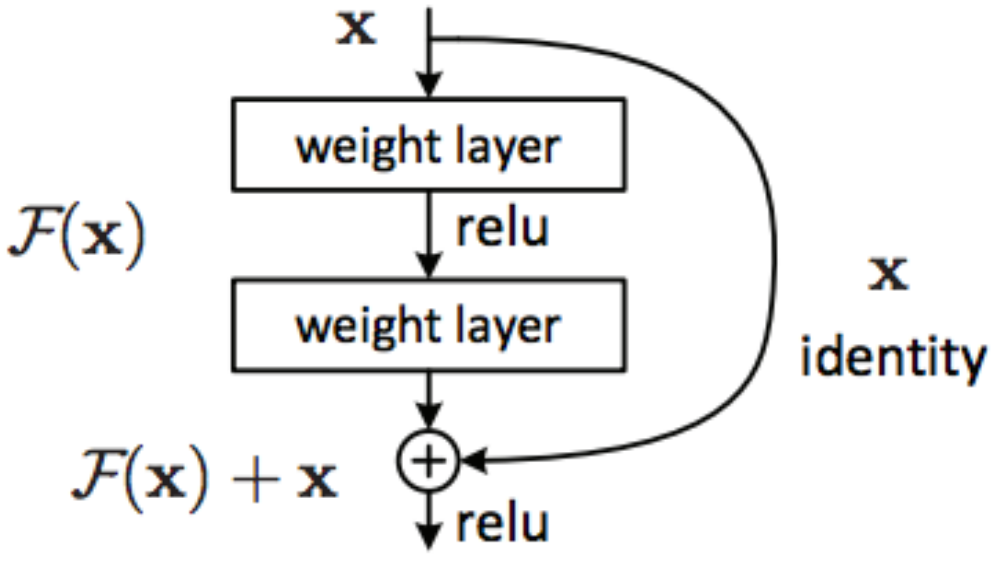
ReLU instead of sigmoid: ReLU doesn't saturate towards $+\infty$, unlike sigmoid neurons, and this helps rectified linear units continue learning. A breakthrough.



The vanishing gradient problem for DNN

Solution 3 He, Zhang, Ren, Sun, 2015

Residual connections, also known as *skip connections*: they provide BP with a shortcut. A breakthrough.



- With this technique, the authors have been able to train a DNN with 1001 layers!

How today works for DNN

http://neuralnetworksanddeeplearning.com/chap6.html

What makes the rectified linear activation function better than the sigmoid or tanh functions? At present, we have a poor understanding of the answer to this question.

http://neuralnetworksanddeeplearning.com/chap6.html

In an ideal world we'd have a theory telling us which activation function to pick for which application. But at present we're a long way from such a world. I should not be at all surprised if further major improvements can be obtained by an even better choice of activation function. And I also expect that in coming decades a powerful theory of activation functions will be developed.

Summing up

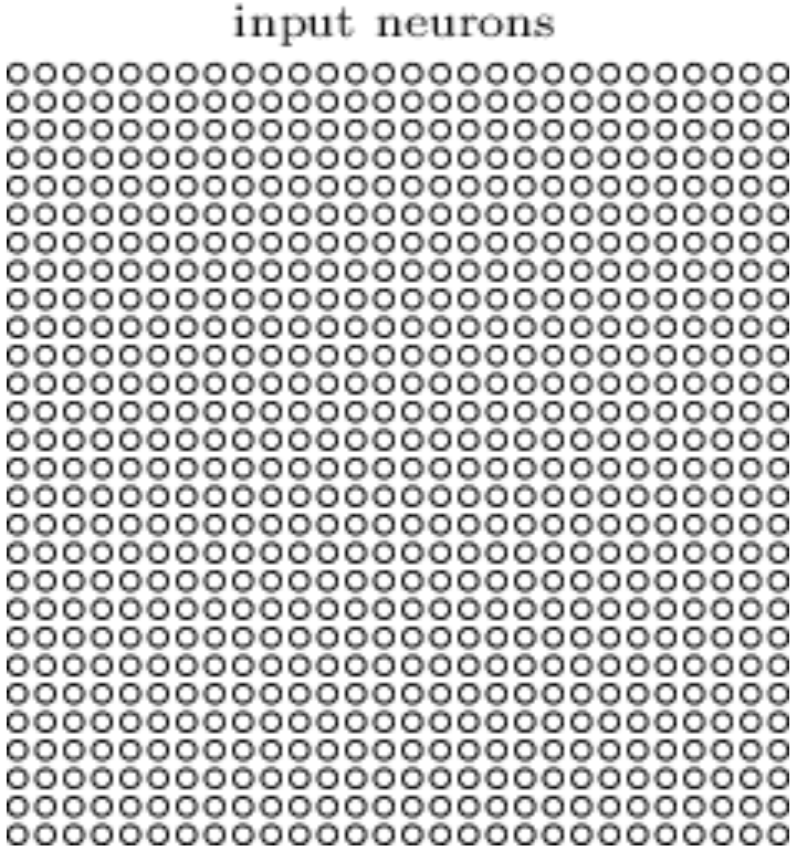
- Learn these terms: DNN, vanishing gradient problem, ReLU, residual connection, skip connection.
- BP is unstable for DNN, essentially because of the chain rule. First layers gets a very small (often) or a very big (sometimes) gradient, making the corresponding weight update too small or too big.
- Nowadays, the unstable gradient issue is solved using a residual connection, that is, a direct connection between non-consecutive layers.
- The DNN world is waiting for mathematicians and theorems!

This was deep

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks ✓
5. Looking for weights aka Learning ✓
6. Deep neural networks ✓
7. Examples of NN
8. Monte Carlo Tree Search
9. AlphaGo and his family

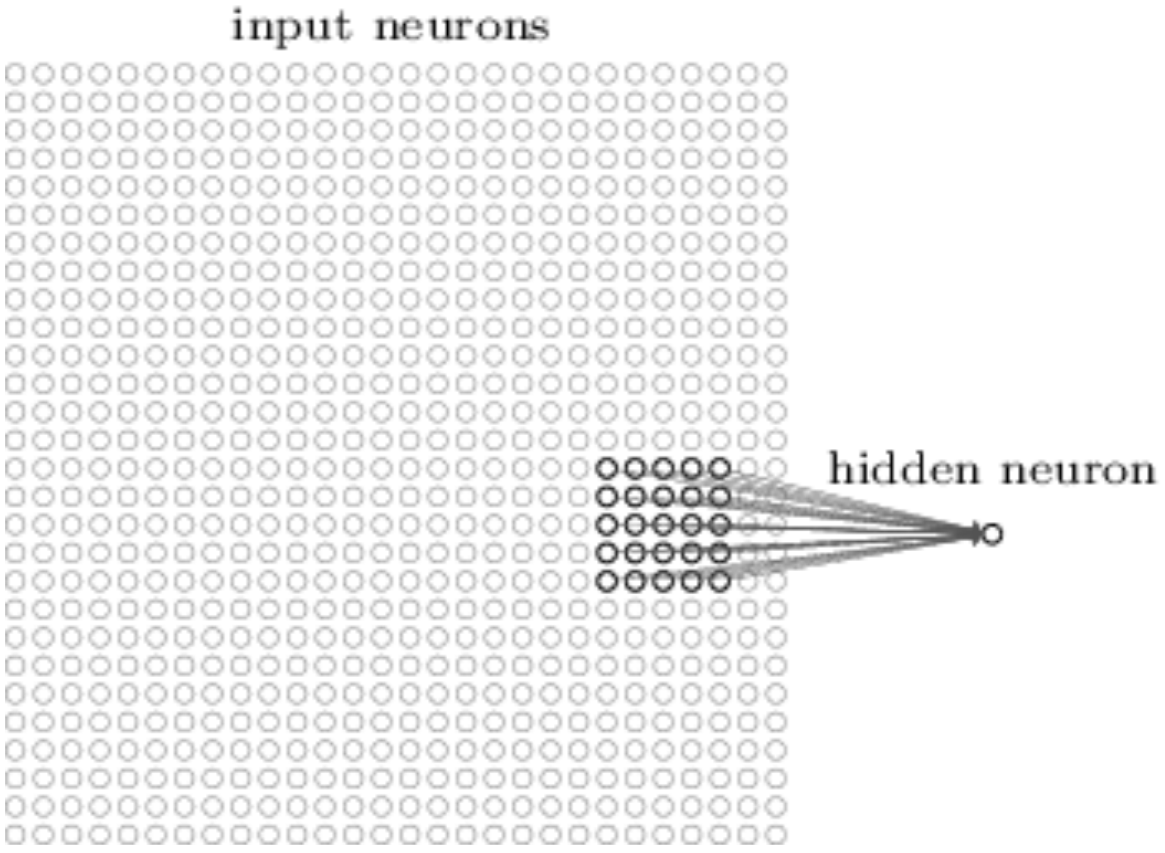
Convolutional layer

- Problem: standard NN do not scale well for images.
- Idea: mimic what is the brain doing to recognize an image.
- The image contains spatial relations.



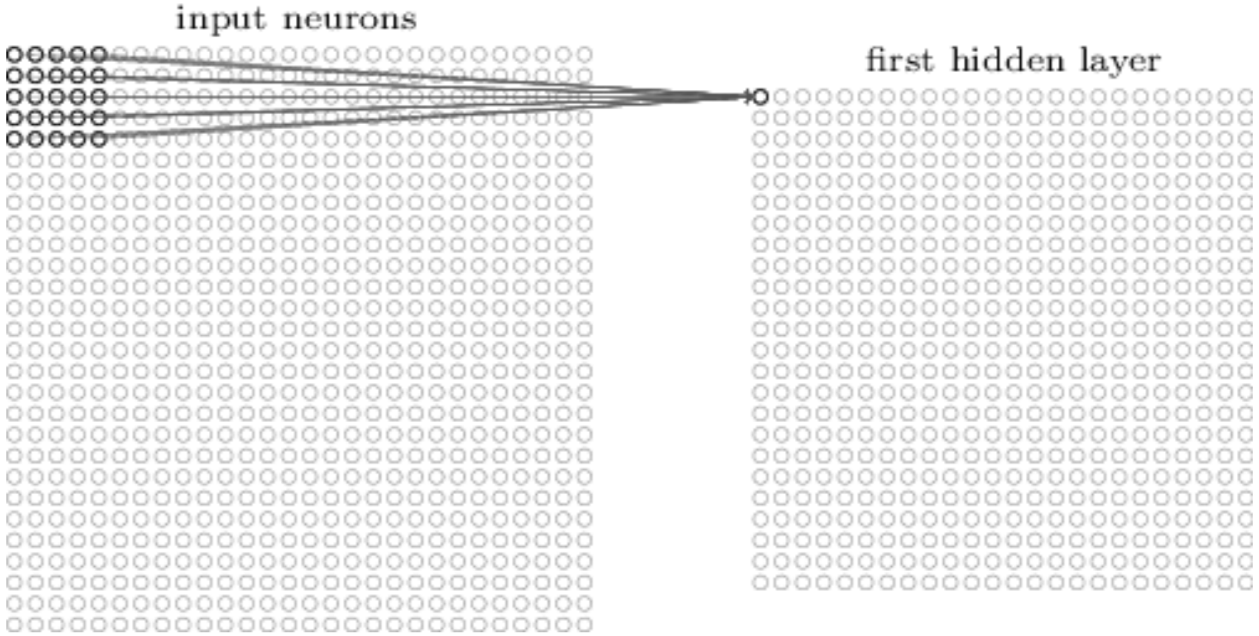
Convolutional layer

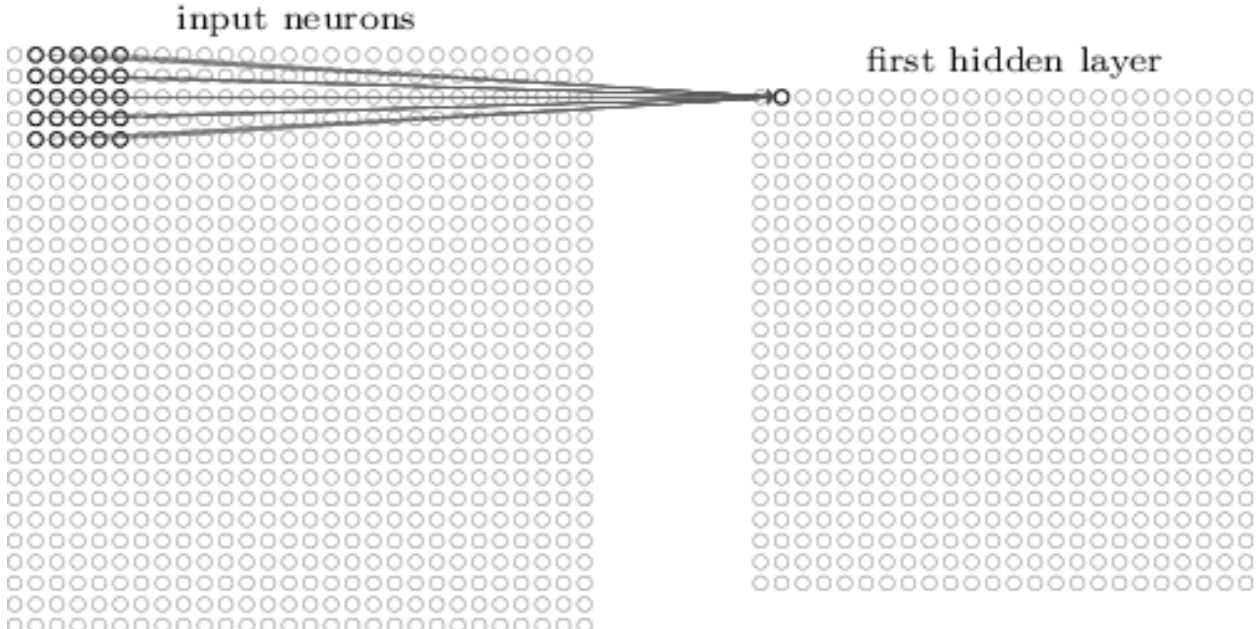
- The human vision uses a *local receptive field*.



Convolutional layer

- The receptive field is moved around according to its *stride*.





Convolutional layer

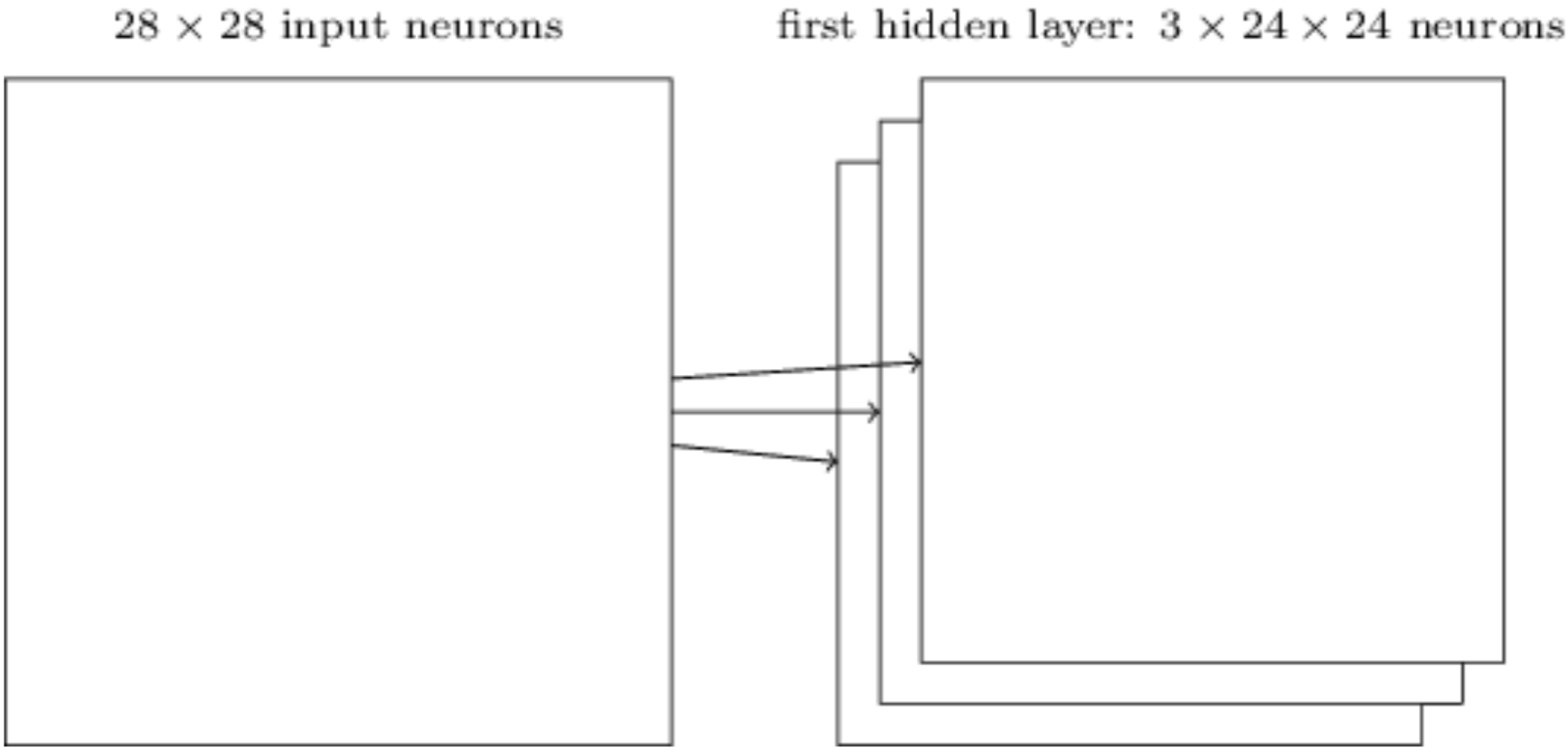
- Weights and biases sharing: the receptive field parameters doesn't change when sliding around.
- In the example, the output of the j, k hidden neuron is:

$$s(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m})$$

- In formula, s is the activation function (ReLU nowadays), b is the *shared bias*, $w_{l,m}$ is a 5x5 array of *shared weights*, $a_{x,y}$ is the input arriving from position x, y in the input layer.
- This makes all neurons in the hidden layer detect the same *feature*. It makes sense, for images.
- In technical terms: a convolutional layer behaves well wrt translations.
- The network map input $\rightarrow$ hidden is the *feature map*.
- The set of shared weights and bias are called *kernel* or *filter*.

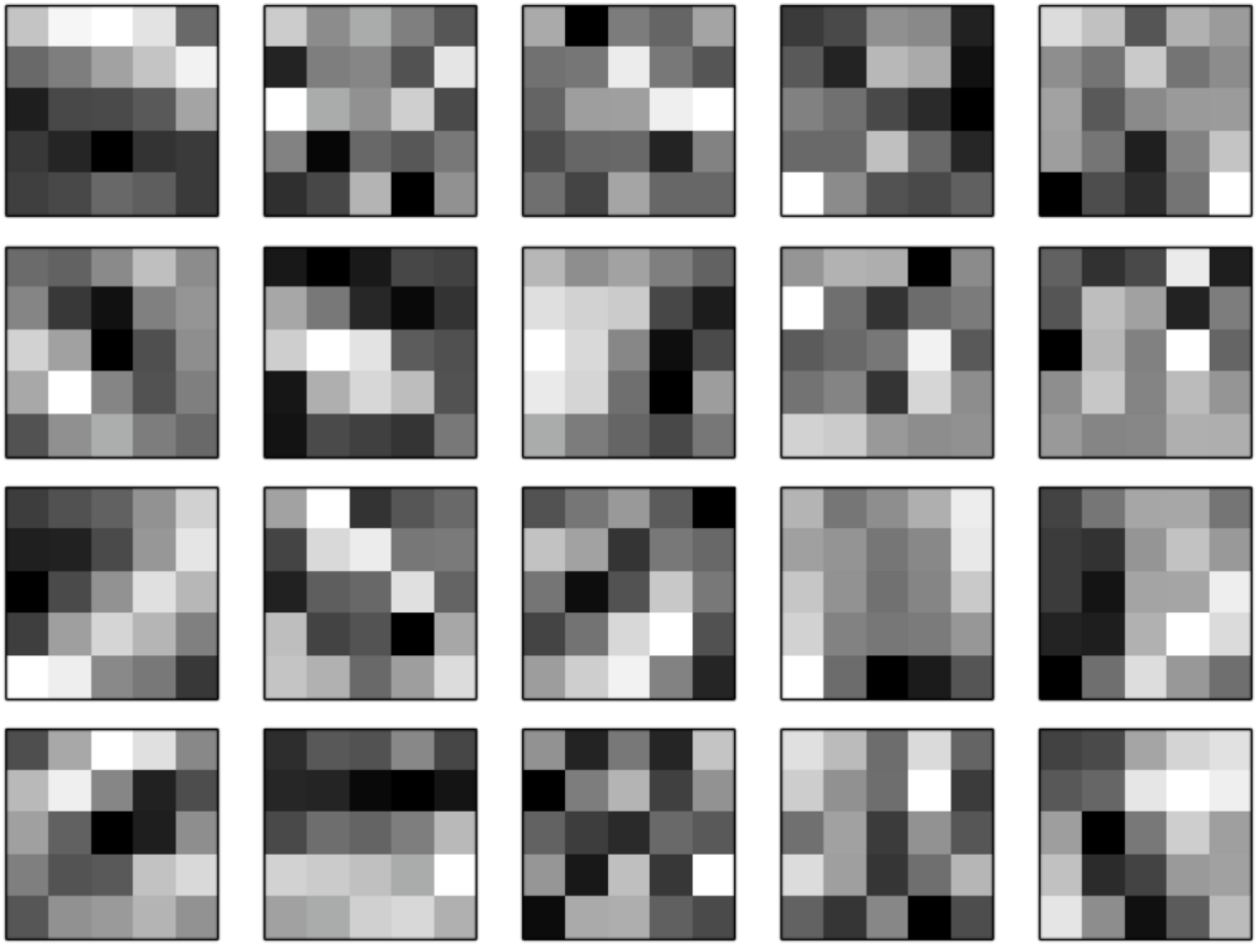
Convolutional layer

- To recognize different features of the image, we need more filters and features maps. We arrange them spatially in this way:



Convolutional layer

- The features learned in a real example (20 feature maps) look like this:



Convolutional layer

- This technique clearly reduces the number of parameters.

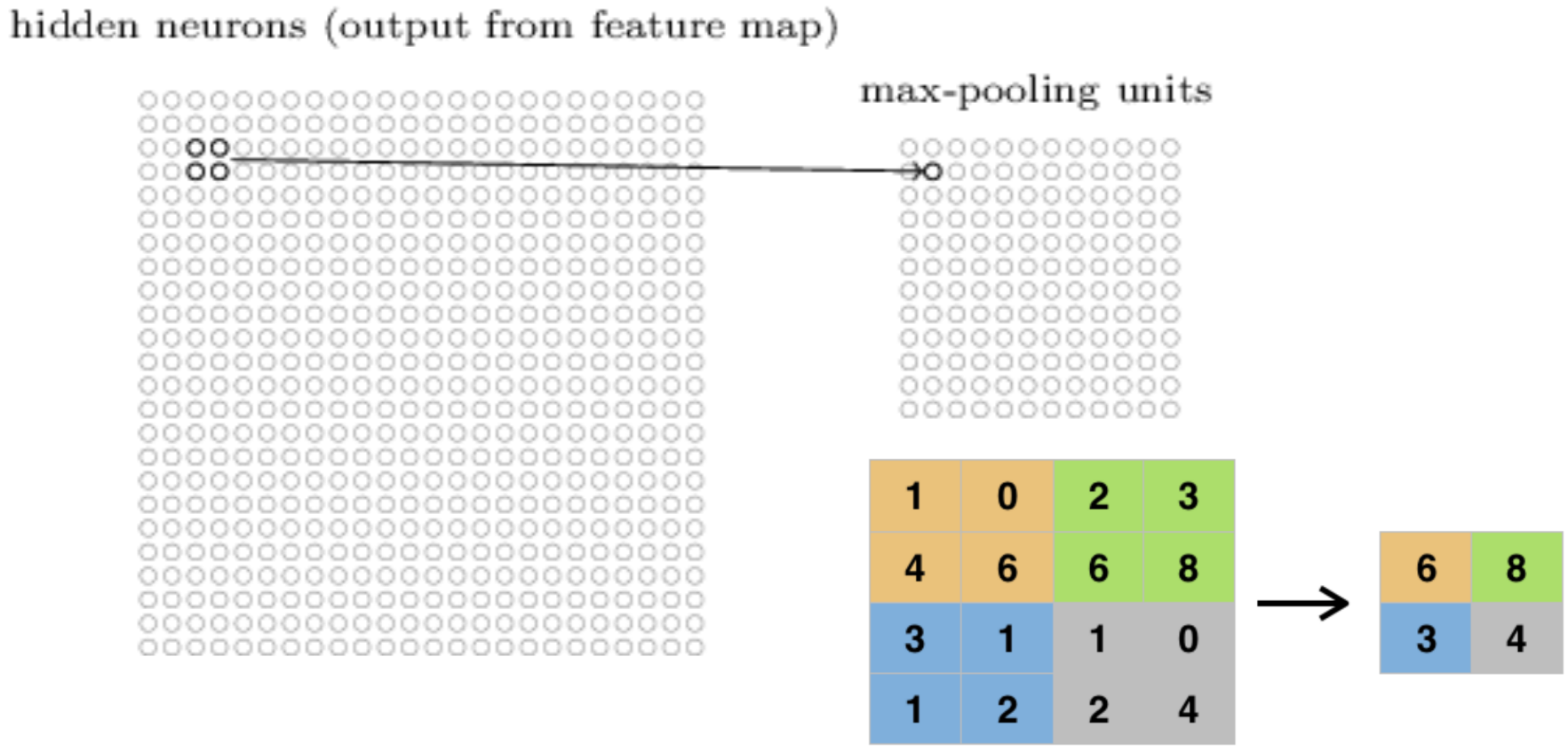
Exercise

Compute the number of parameters in a convolutional layer filtering a 28×28 image with 20 kernels of size 5×5 and stride 1, and compare with the fully connected case.

- The name convolution comes from the analogous mathematical operation.

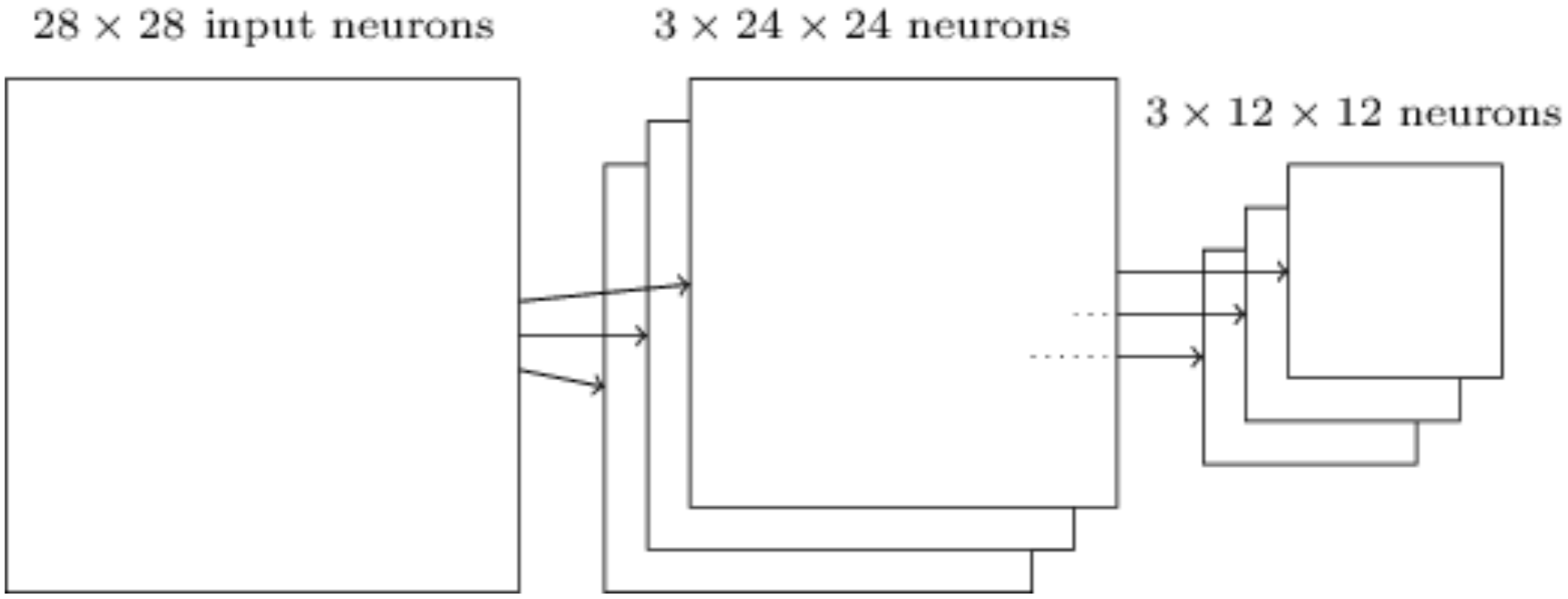
Pooling layer

- After the convolutional layer, one often puts a *pooling* layer. Aim: condense information with a certain pooling procedure (max, L^2,...).
- The *max-pooling* procedure takes a region and output the maximum.



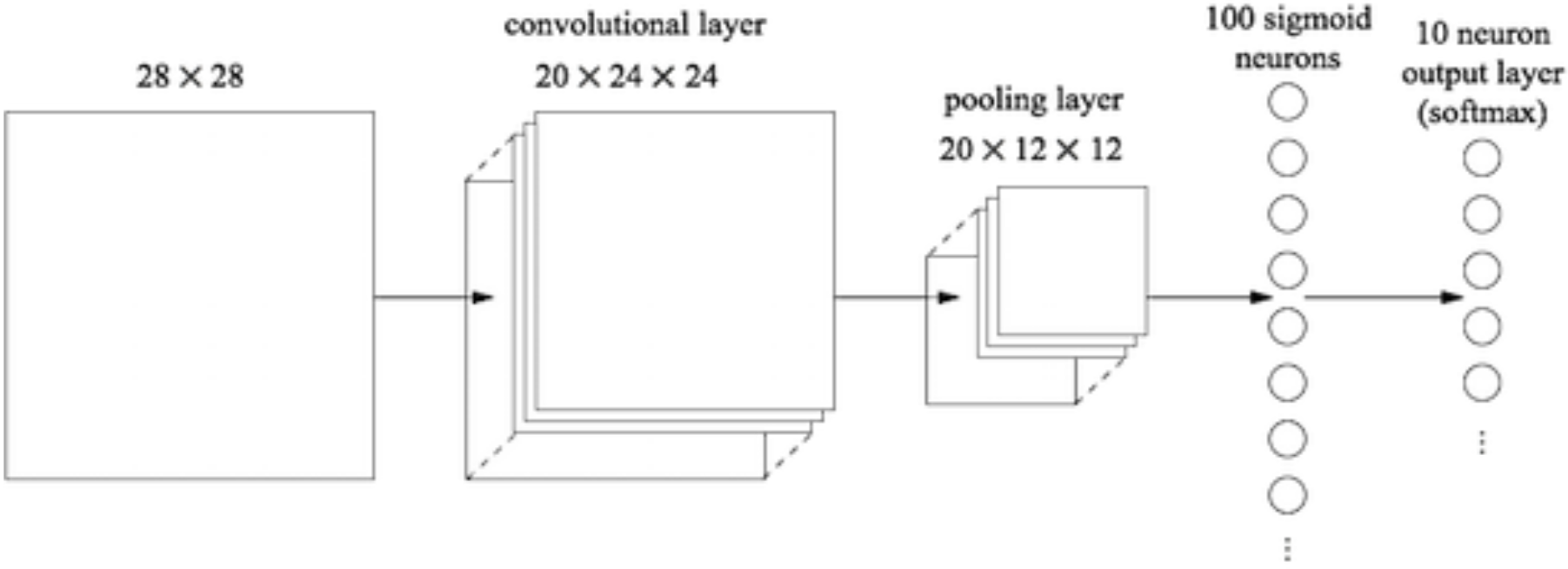
Pooling layer

- If the *pooling size* is 2×2, and its input is 24×24, the pooling output is 12×12.



Convolutional NN

- Adding a bigger fully-connected layer can help in collecting all the local features together, before the final output.



- CNN = (convolutional + pooling) +. . . + (convolutional + pooling) + fully connected + output (often softmax).

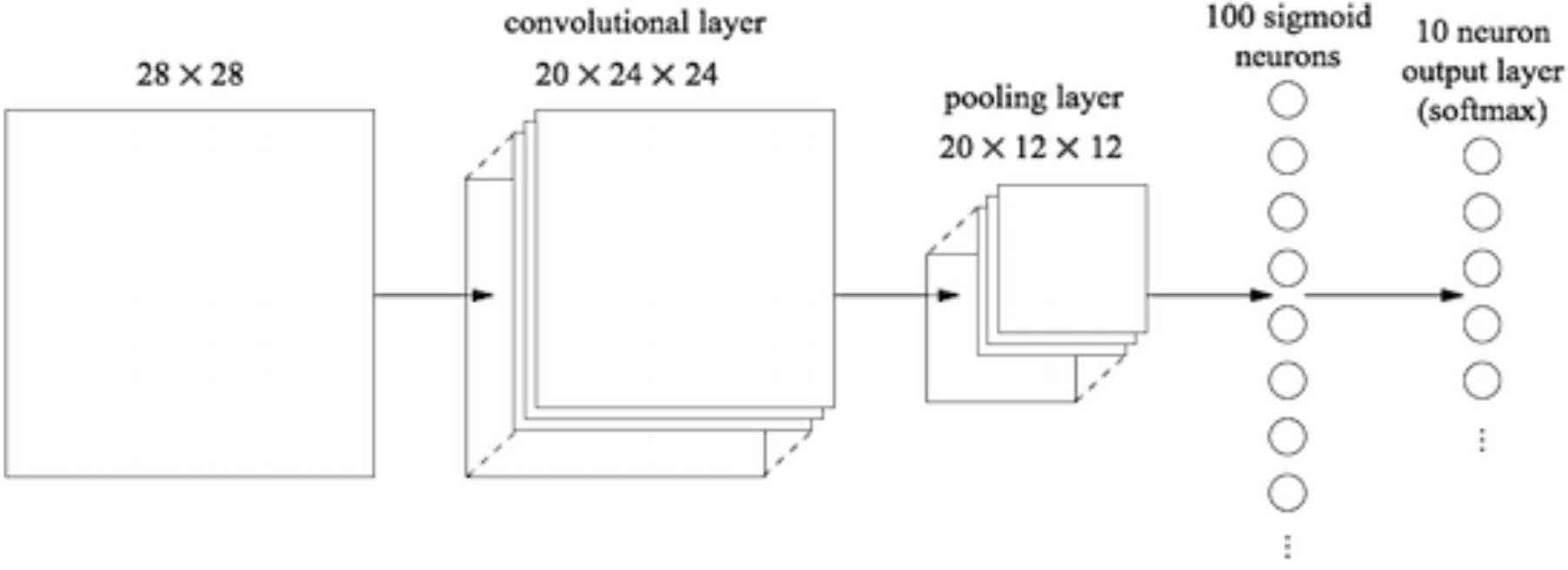
Convolutional NN

Exercise

Compute how many additional parameters are given by a convolutional layer following the $20 \times 12 \times 12$ pooling layer.

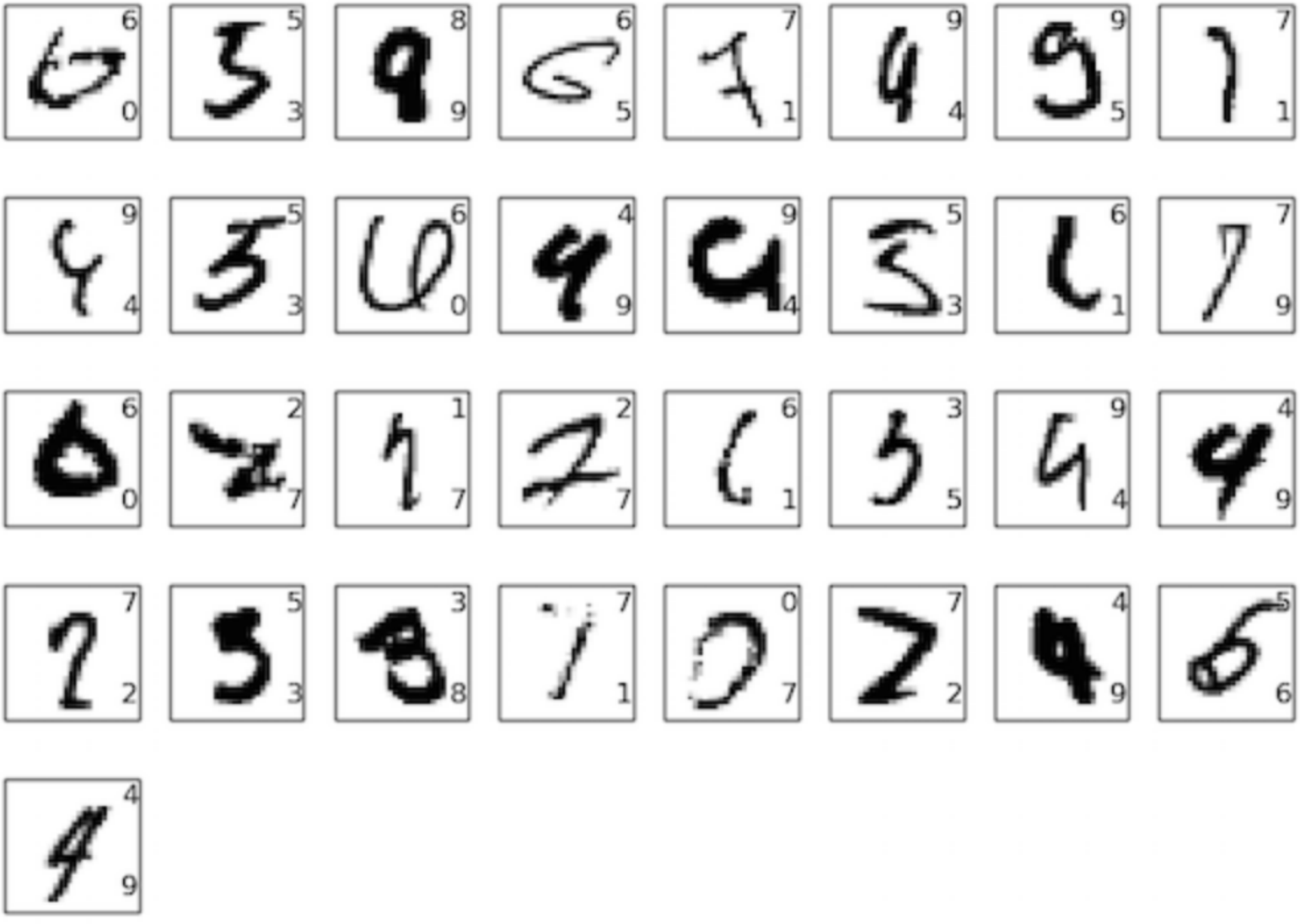
Exercise

Compute how many parameters we have in this CNN.



The final classification

- In the real case described, only 33 out of 10^4 images have not been recognized:



Softmax layer

- When classification in K classes is needed, the output layer should consist of K nodes and use the *softmax* $s : \mathbb{R}^K \to \mathbb{R}^K$ as activation function.

$$s(x_1, \ldots, x_K) = (e^{x_1}, \ldots, e^{x_K}) / \sum_{i=1}^{K} e^{x_i}$$

- Softmax outputs a probability distribution.

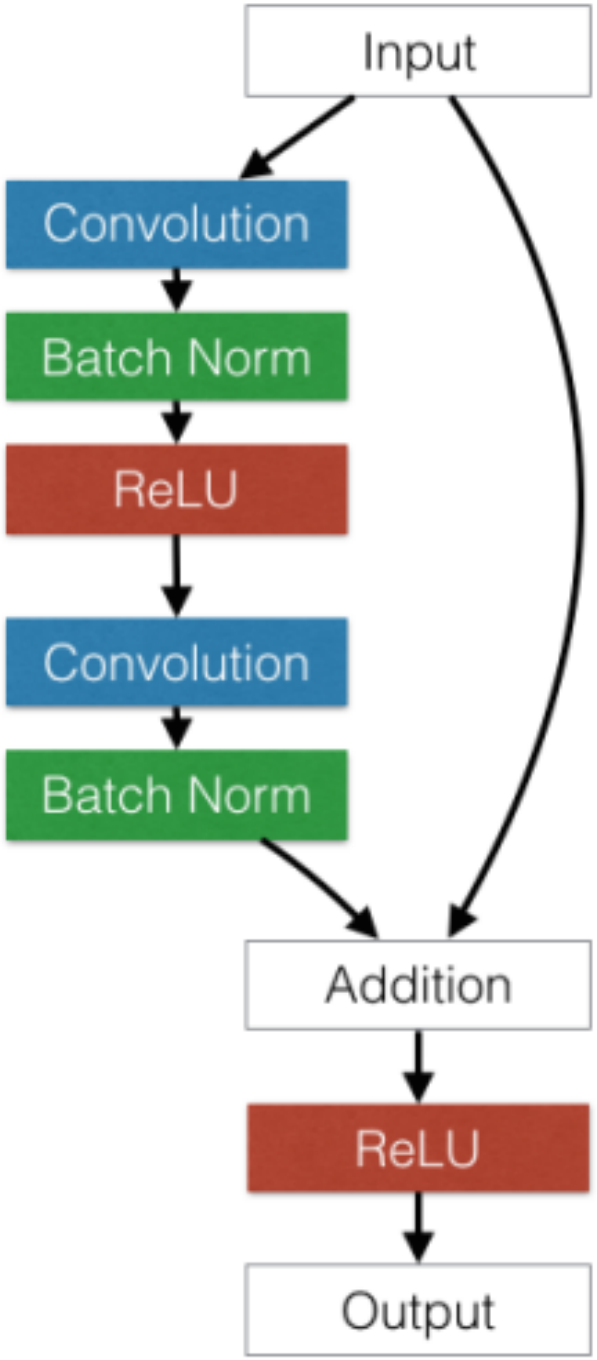
Exercise

Show that increasing x_j will increase $s_j(x_1, \ldots, x_K)$ and decrease all other components.

Exercise

Compute $\lim_{c \to +\infty} e^{cx_j} / \sum_{i=1}^{K} e^{cx_i}$. Why is softmax called this way?

Residual layer



Summing up: activation functions

- Heaviside, or Threshold or Step (used originally perceptron).
- Sigmoid
- Logistic (a particular sigmoid).
- Hyperbolic Tangent or Symmetric Sigmoid ($2s(2x) - 1$).
- ReLU. Does not saturate at -1 or $+1$. Most used nowadays, because it solves the vanishing gradient problem.
- Softmax (used to output a probability distribution).

Summing up

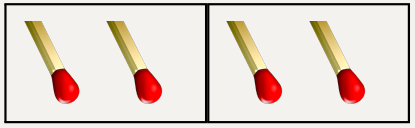
- Learn these terms: CNN, convolutional layer, local receptive field, stride, feature map, kernel, filter, features, pooling, max-pooling, softmax layer, cross-entropy, residual layer.
- NN means learning a lot of vocabulary, but a lot of things are "easy" when you know what you are talking about.

I know a lot of things now!

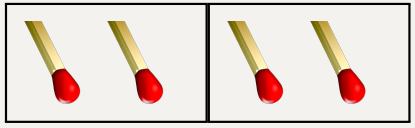
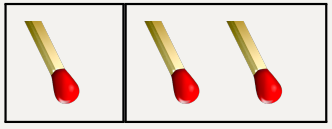
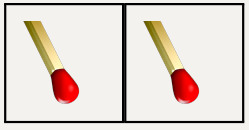
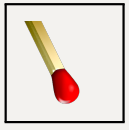
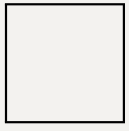
1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks ✓
5. Looking for weights aka Learning ✓
6. Deep neural networks ✓
7. Examples of NN ✓
8. Monte Carlo Tree Search
9. AlphaGo and his family

The Nim: one game to rule them all

Rules

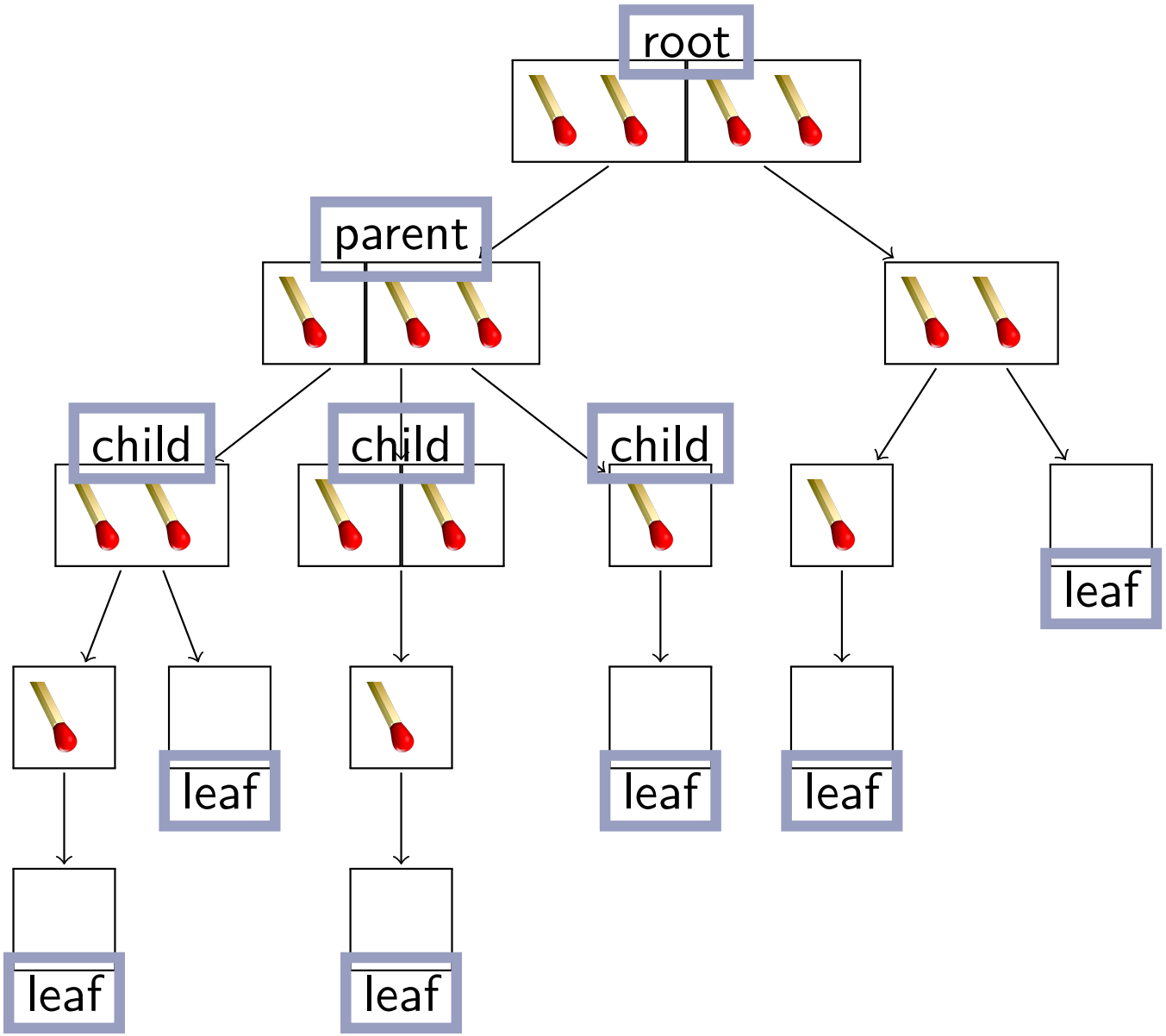
- N heaps of items: matches, in the picture: 
- On each turn, a player must remove at least one item. He may remove any number of items provided they all come from the same heap.
- Normal rule: you lose if you can't move.

Example

- Two players: Alice and Bob.
- Starting position: 
- Bob takes 1 item from left heap: 
- Alice takes 1 item from right heap: 
- Bob takes 1 item from left heap - he must move: 
- Alice takes the last item and wins: 

A mathematician tree

Nim is a tree!



Who plays

Bob

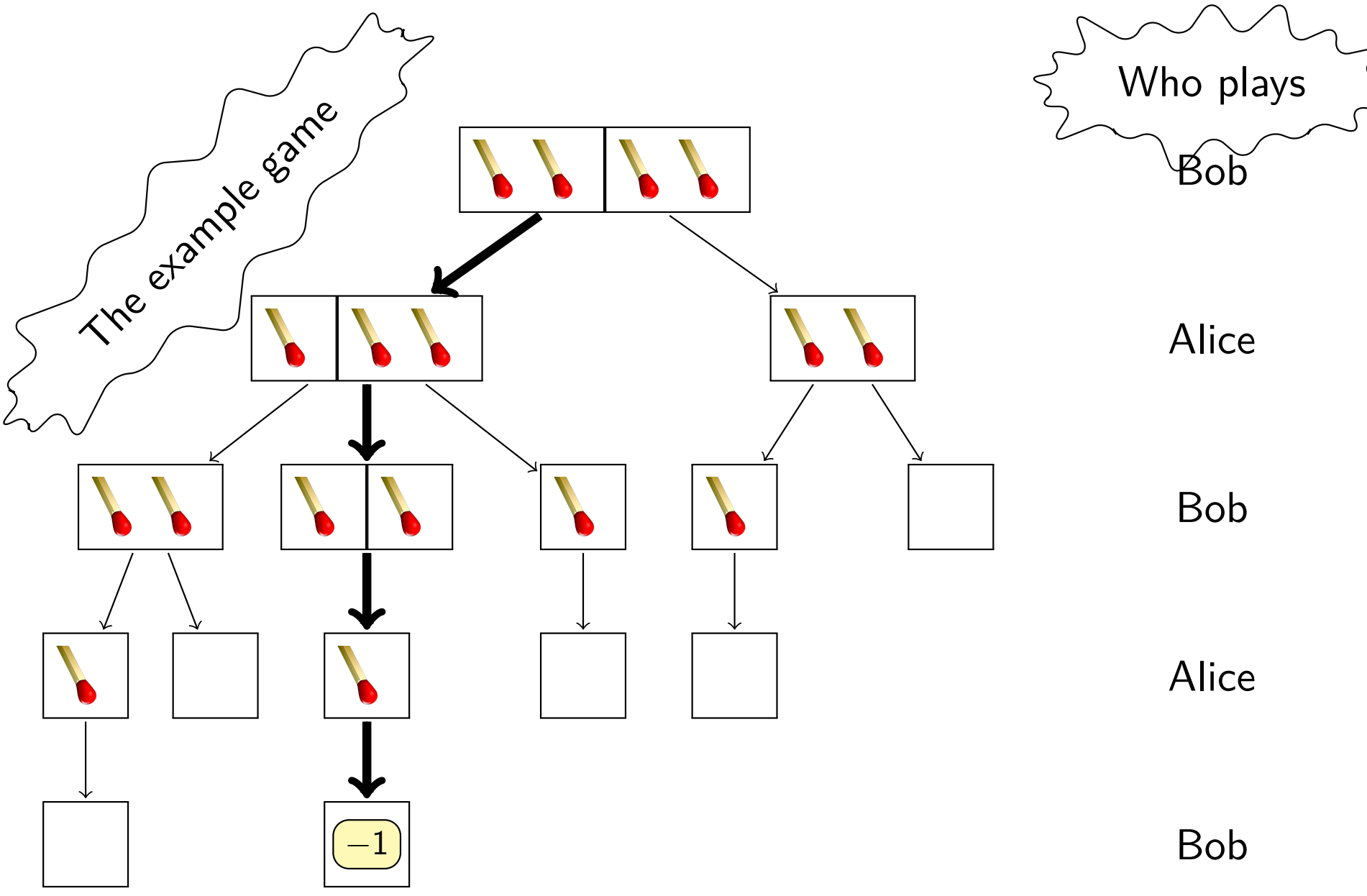
Alice

Bob

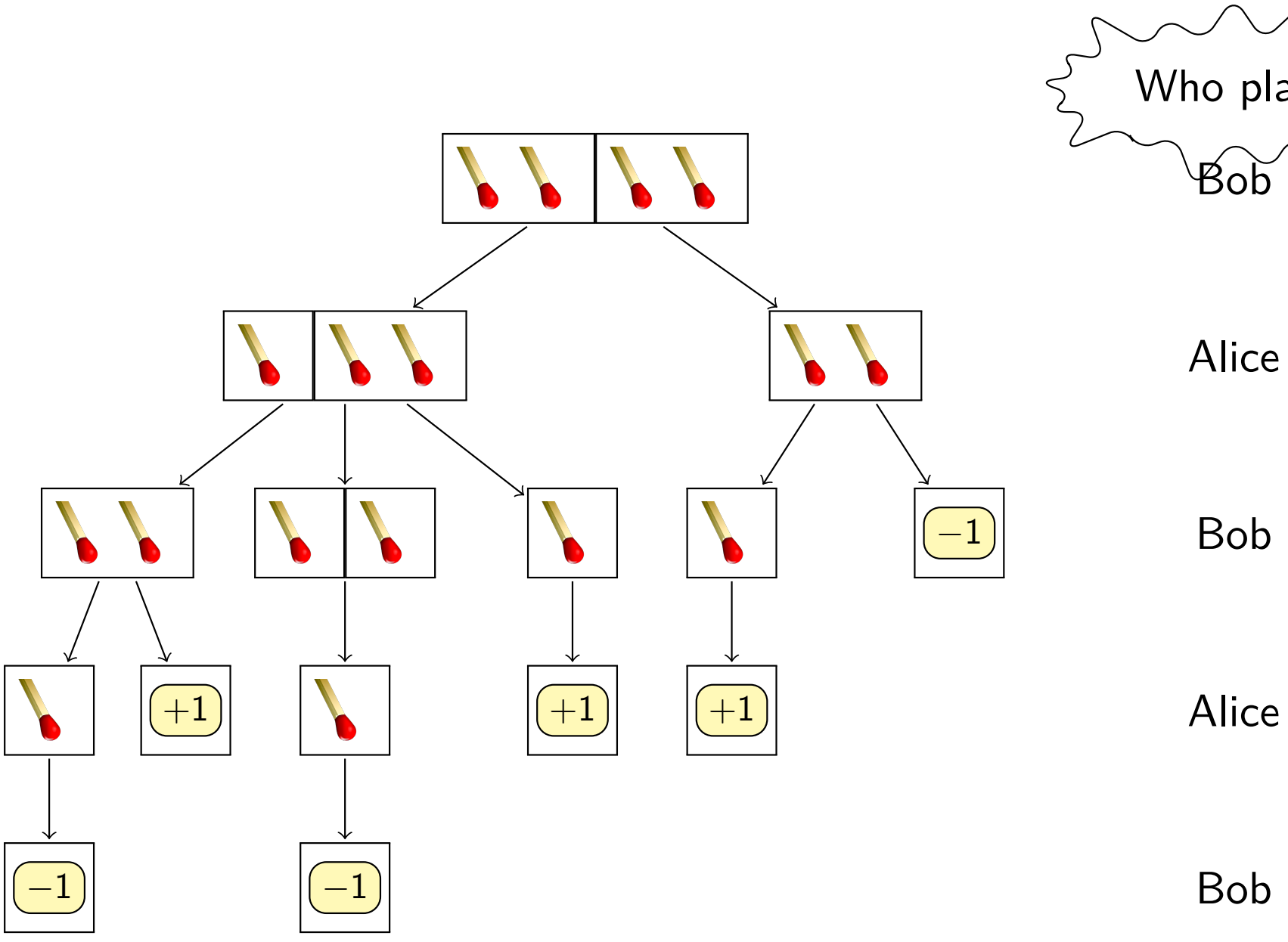
Alice

Bob

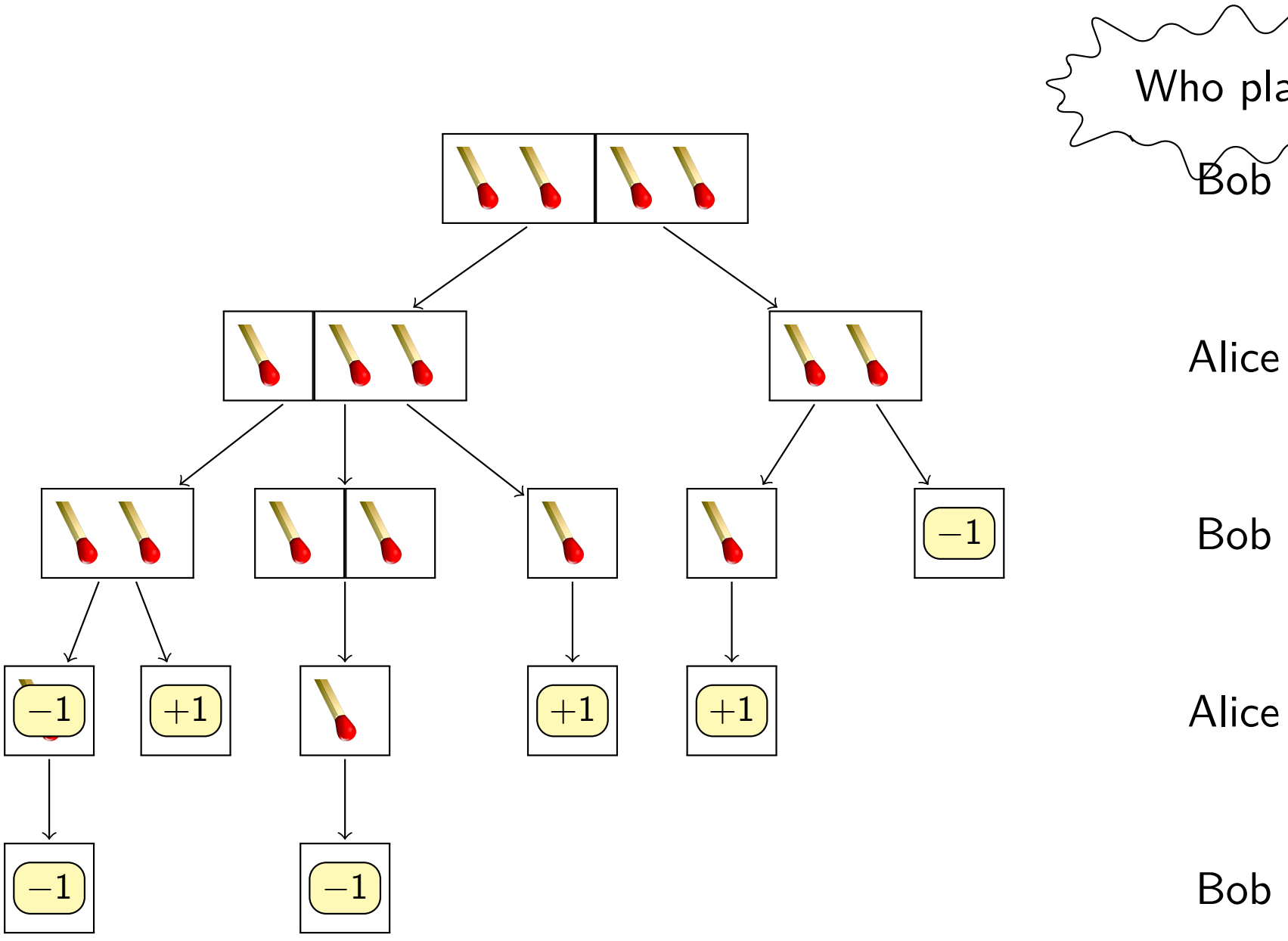
Nim is a tree!



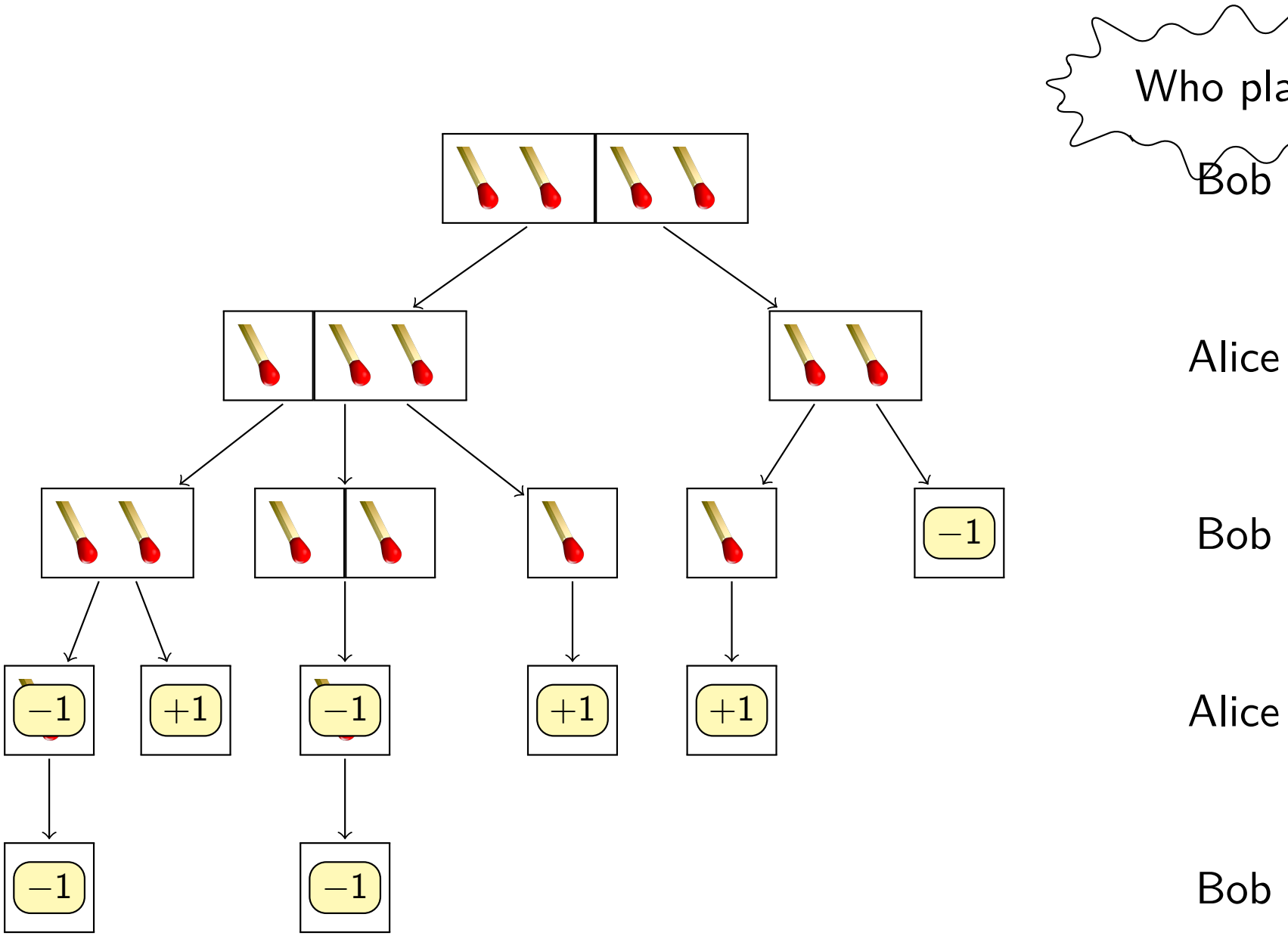
Nim is a tree!



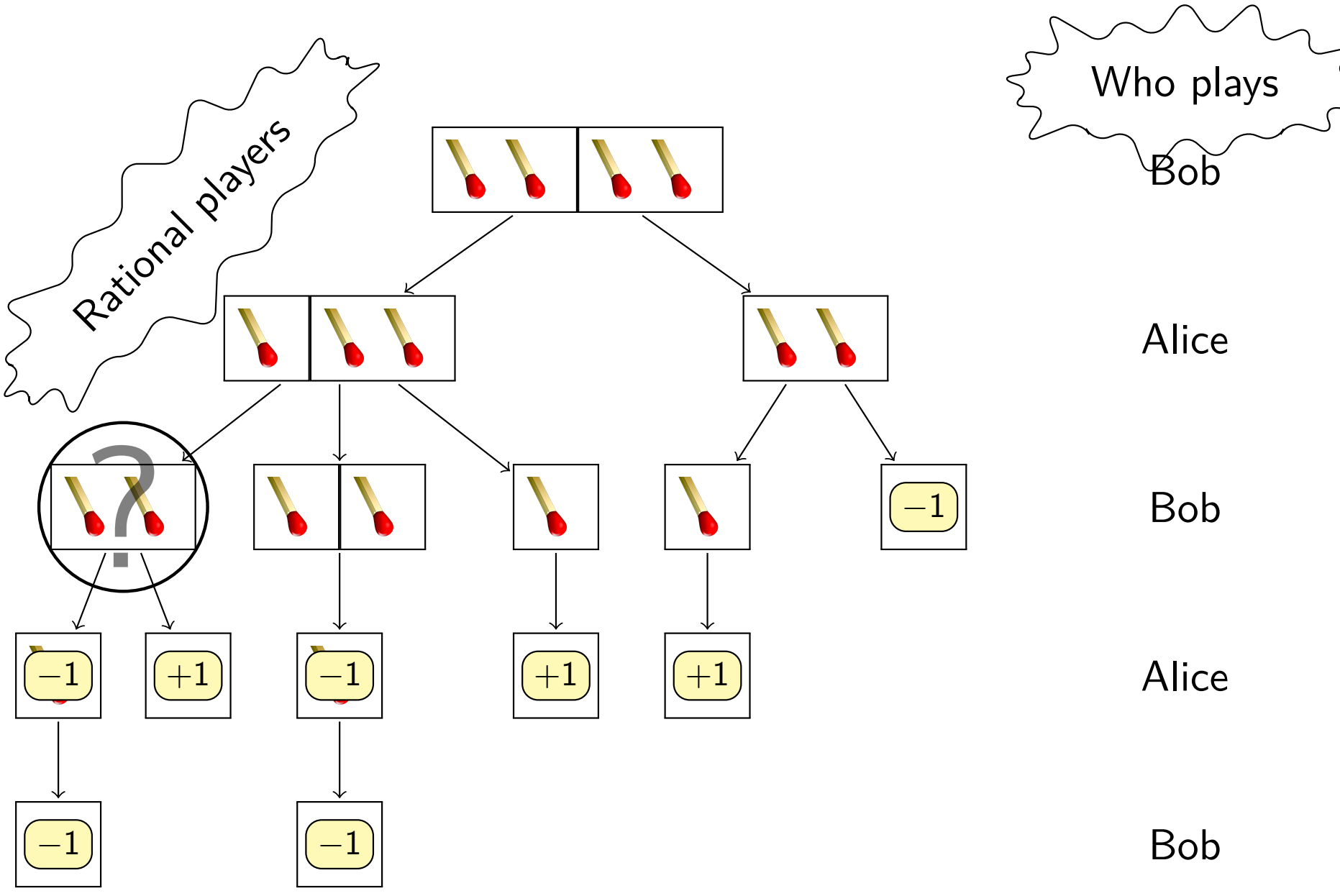
Nim is a tree!



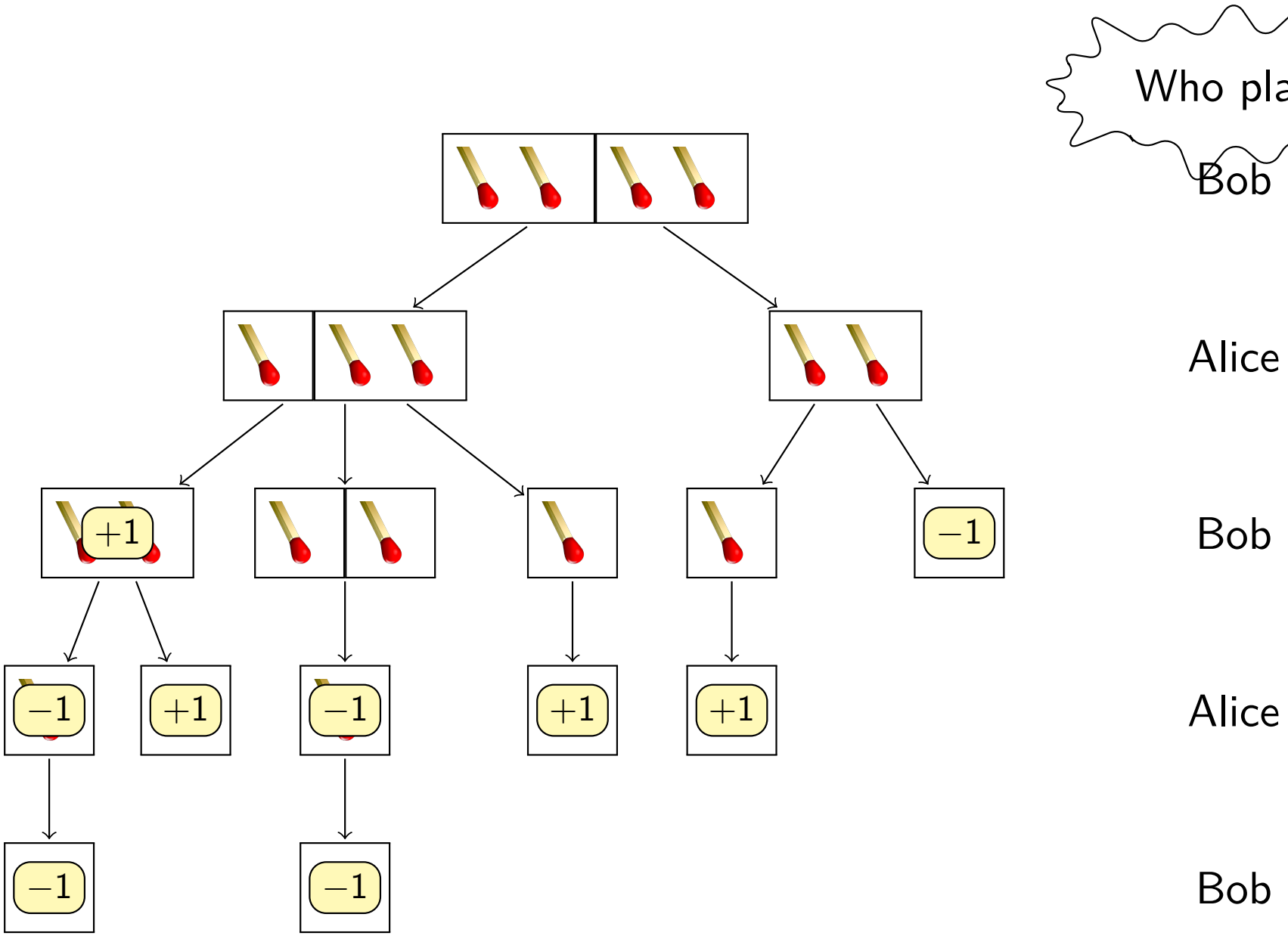
Nim is a tree!



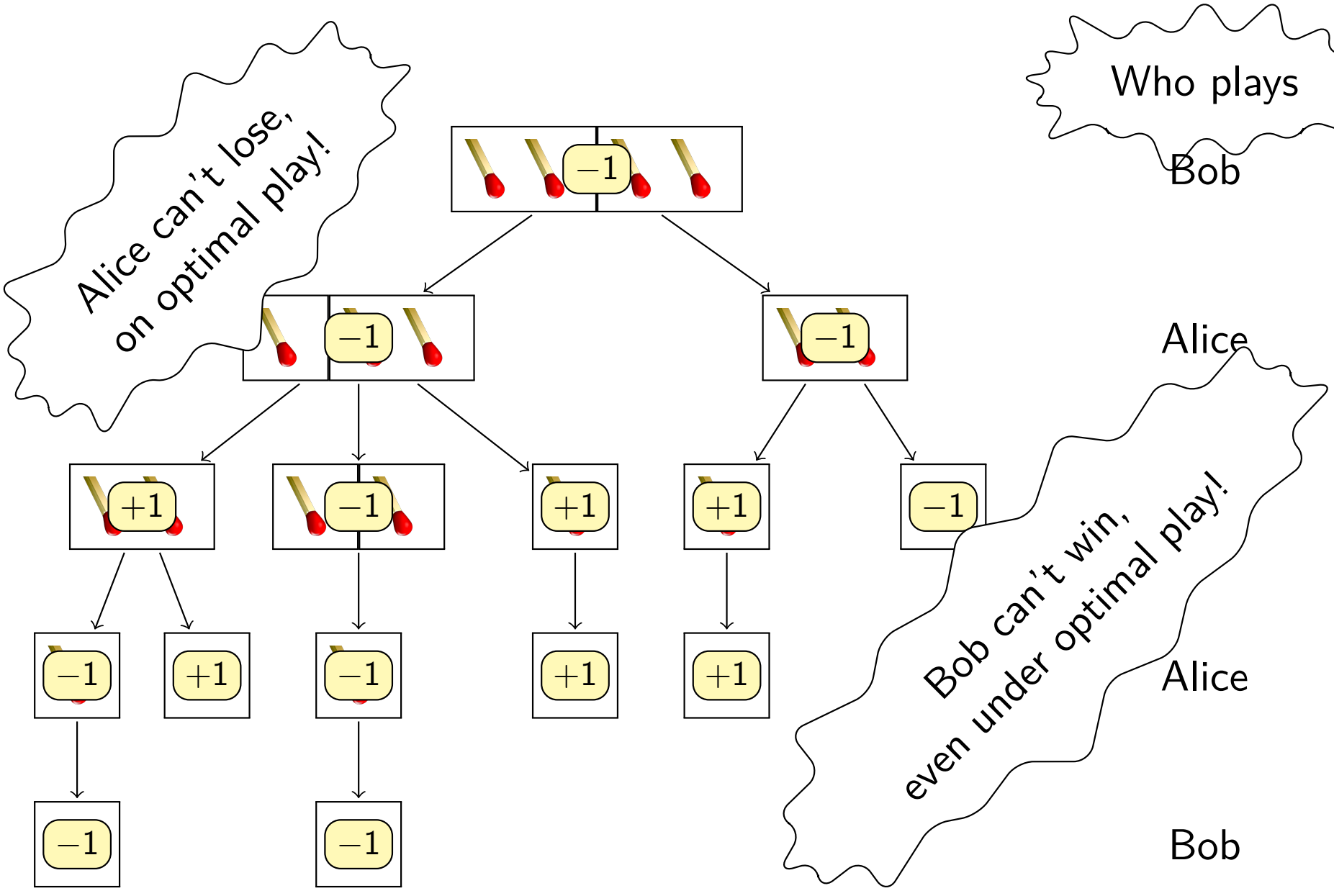
Nim is a tree!


Rational players
Who plays
Bob
Alice
Bob
Alice
Bob
?
−1
−1
+1
−1
+1
+1
−1
−1

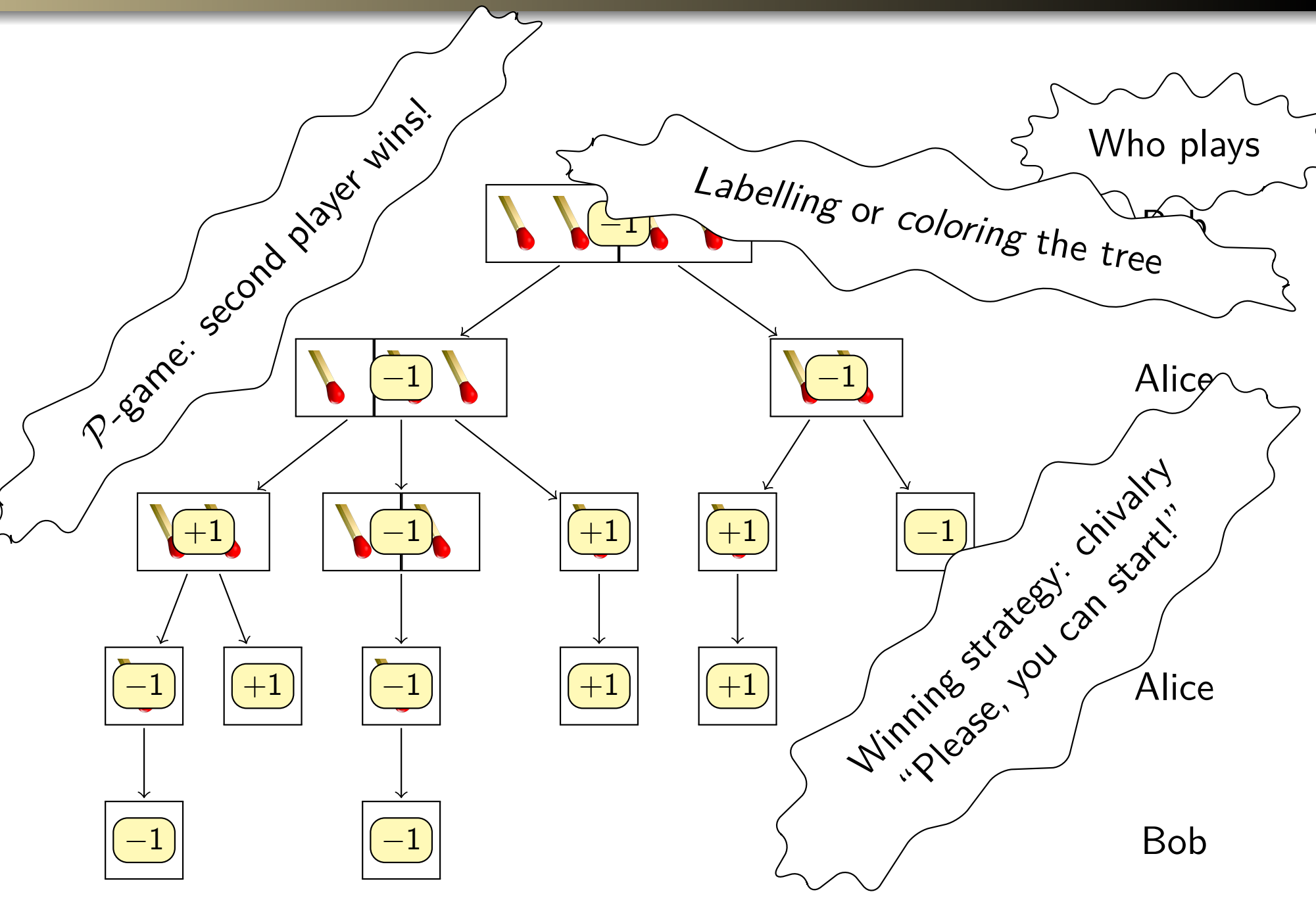
Nim is a tree!



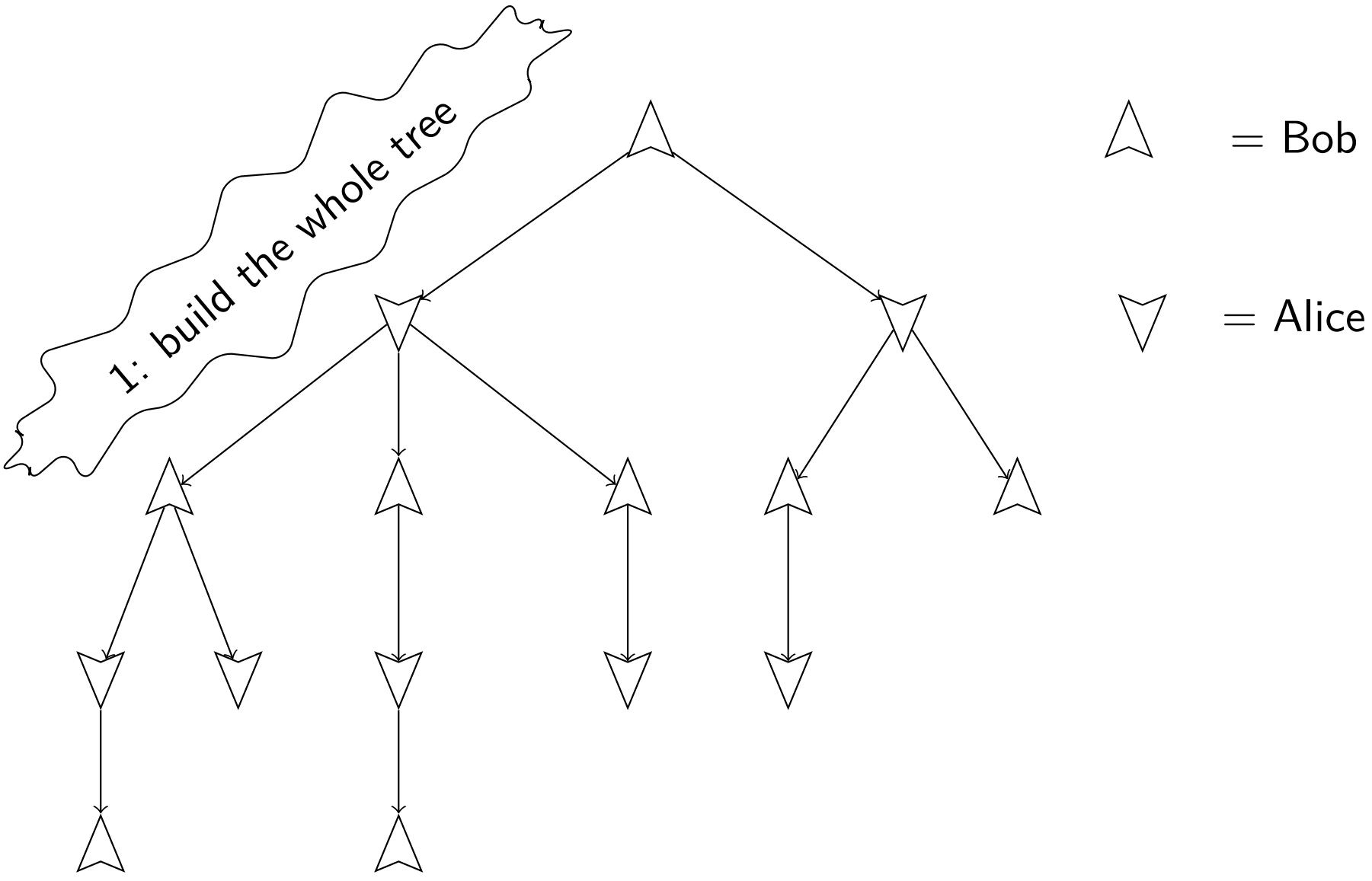
Nim is a tree!


Who plays
Bob
Alice
Alice
Bob
Alice can't lose,
on optimal play!
Bob can't win,
even under optimal play!
−1
−1
−1
+1
−1
+1
+1
+1
−1
−1
+1
−1
+1
+1
−1
−1

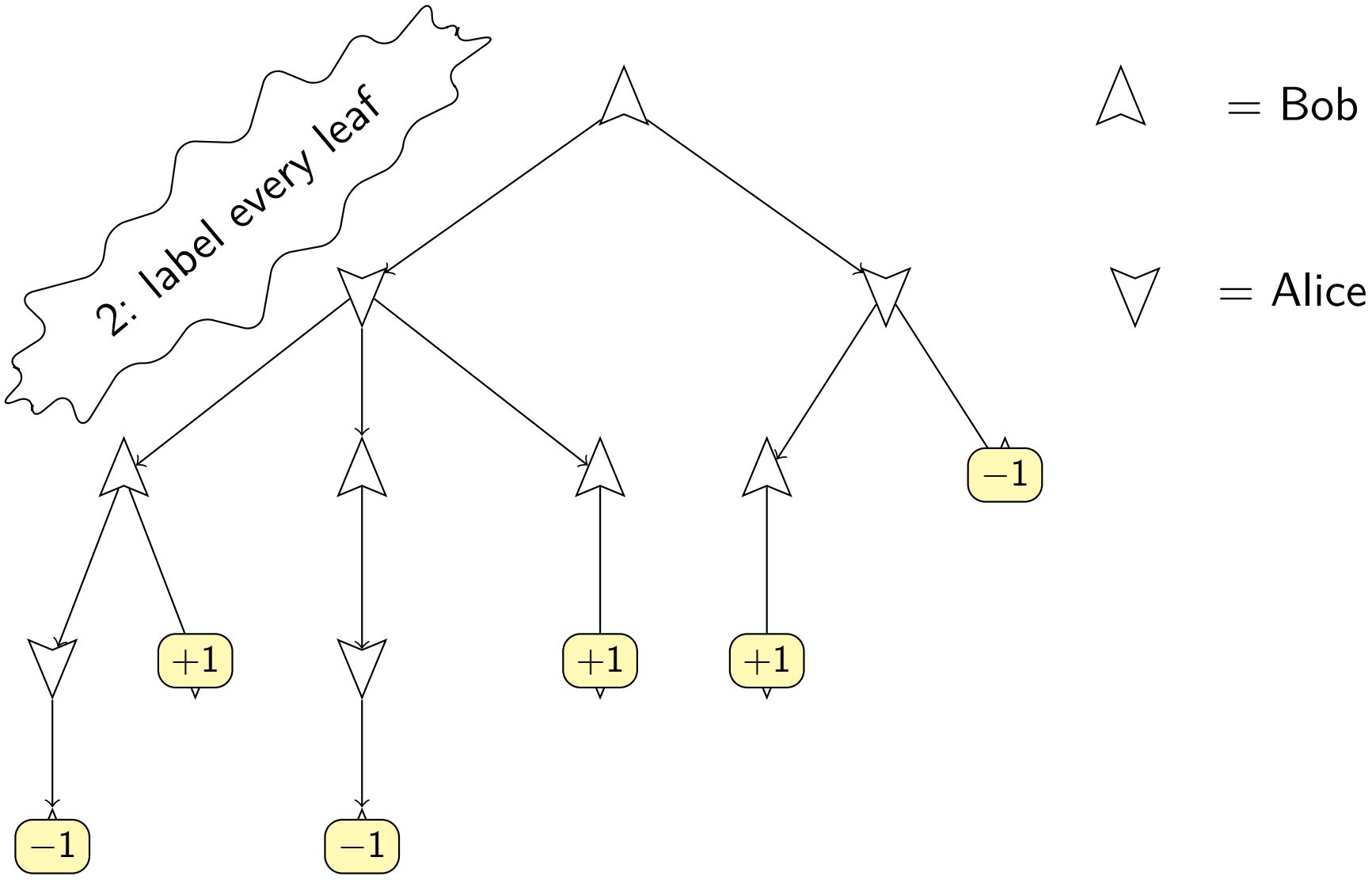
Nim is a tree!


$\mathcal{P}$-game: second player wins!
Labelling or coloring the tree
Who plays
Alice
Winning strategy: chivalry
"Please, you can start!"
Alice
Bob
−1
−1
−1
+1
−1
+1
+1
+1
−1
−1
+1
−1
+1
+1
−1
−1

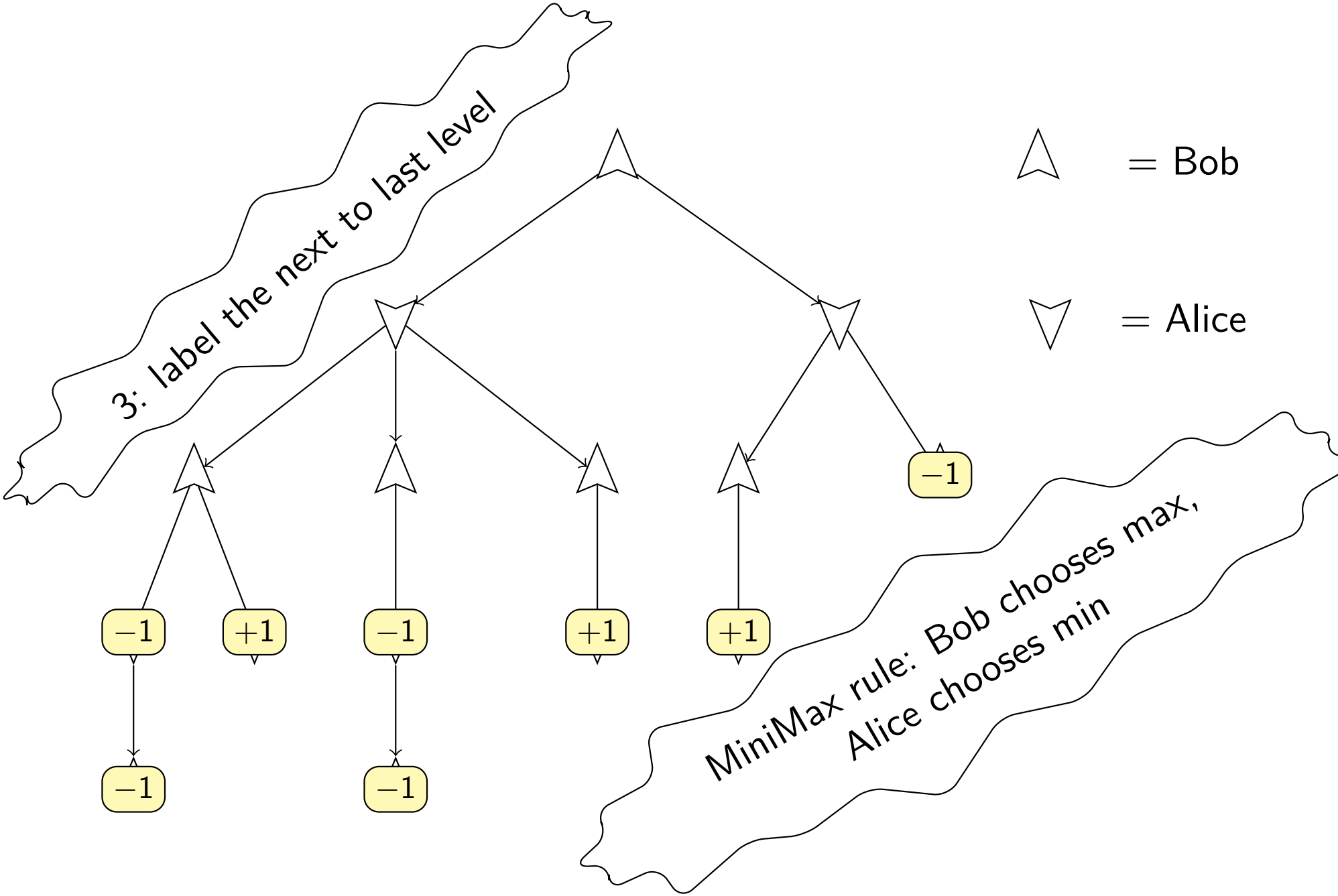
MiniMax algorithm



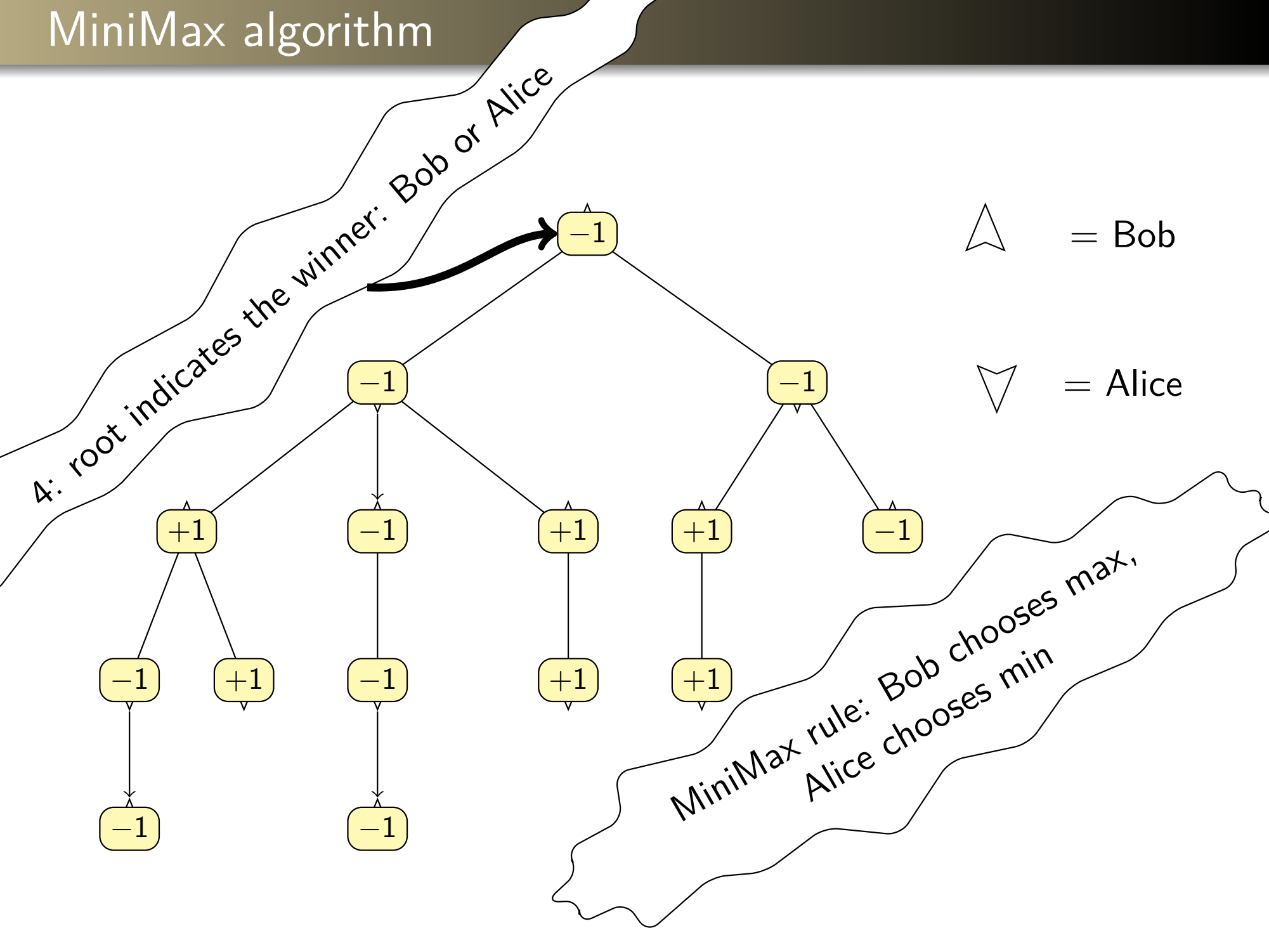
MiniMax algorithm



MiniMax algorithm

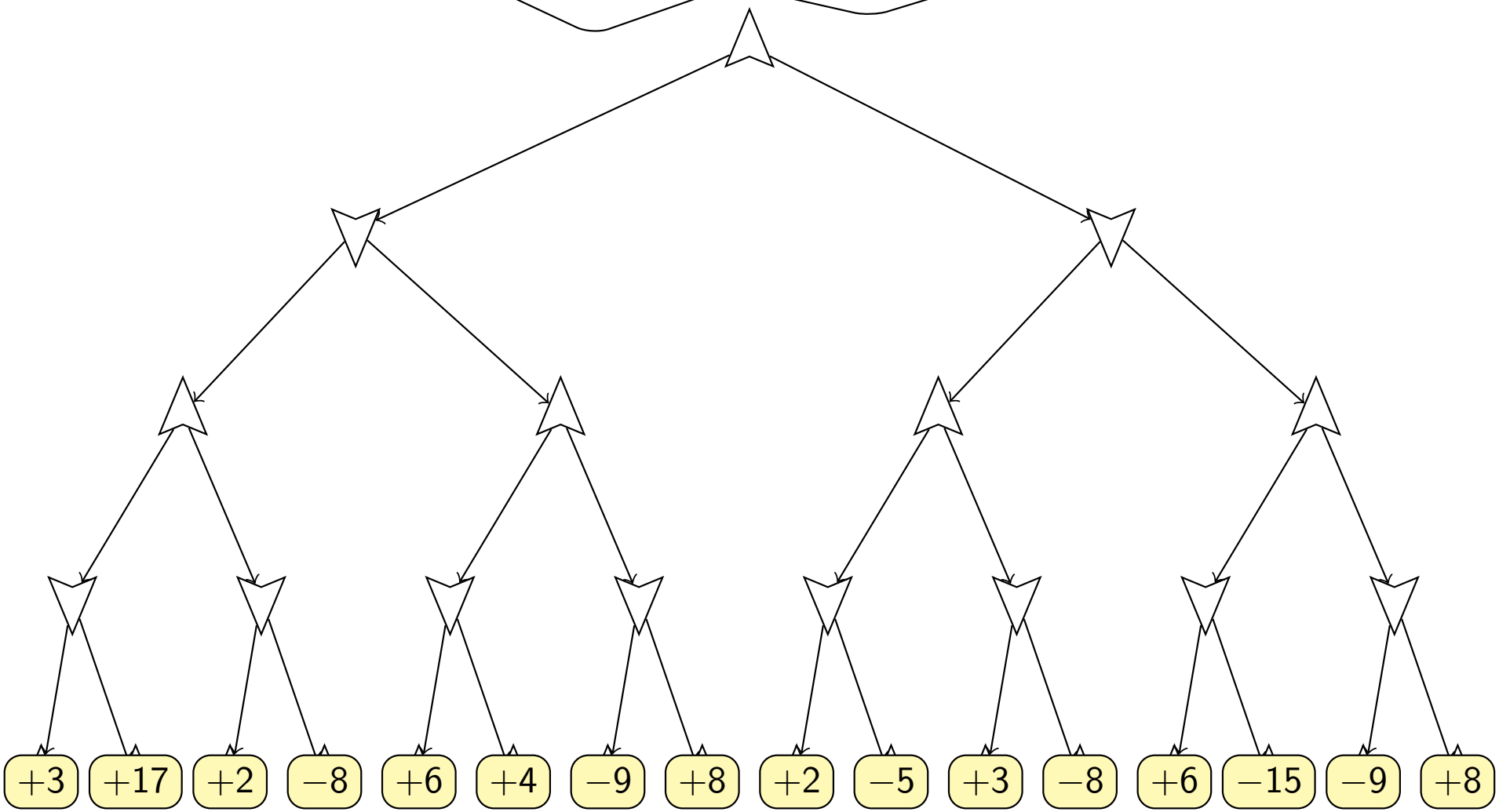


MiniMax algorithm



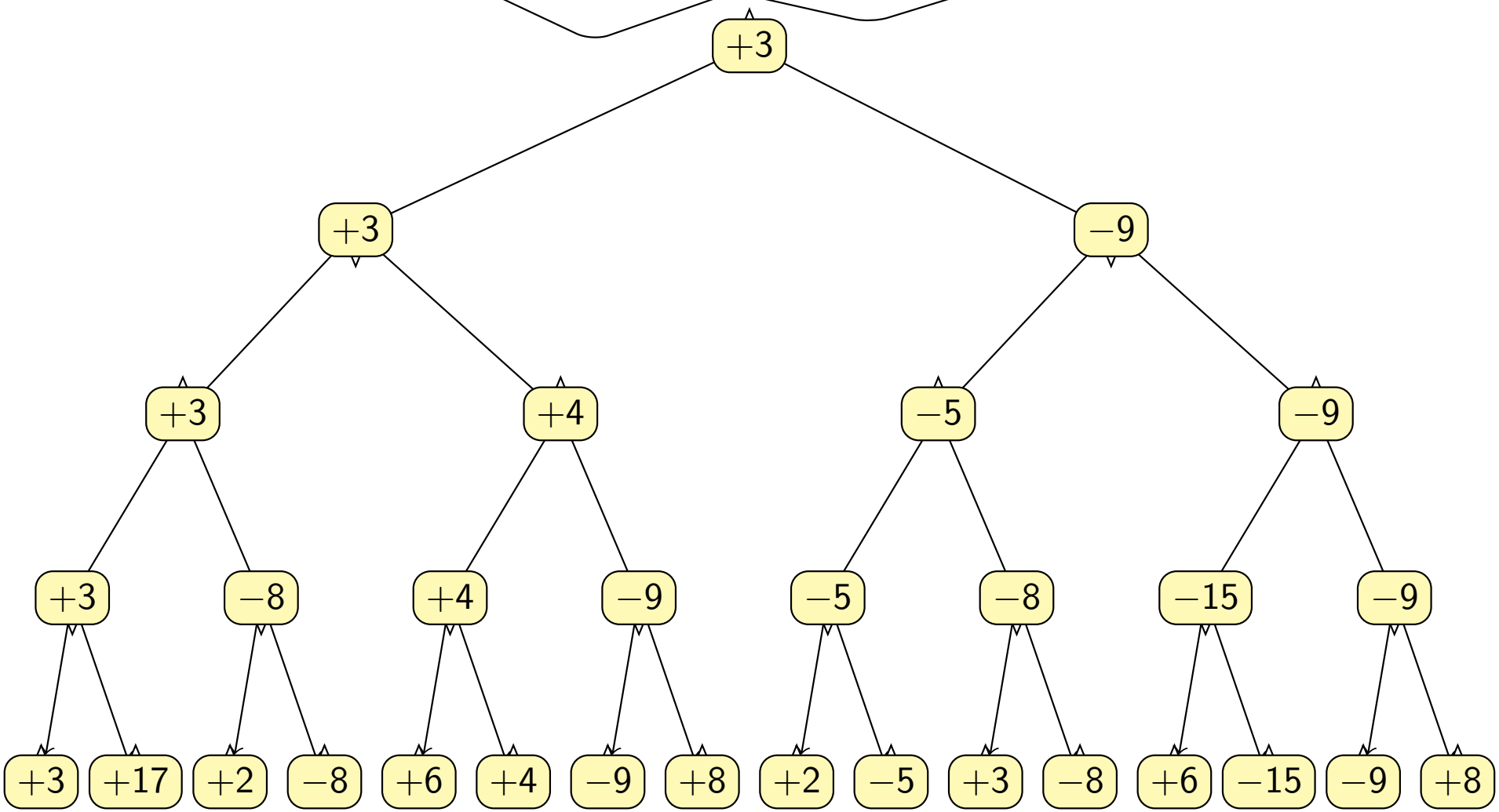
Who is the winner?

Who wins? Bob or Alice? How much score difference?



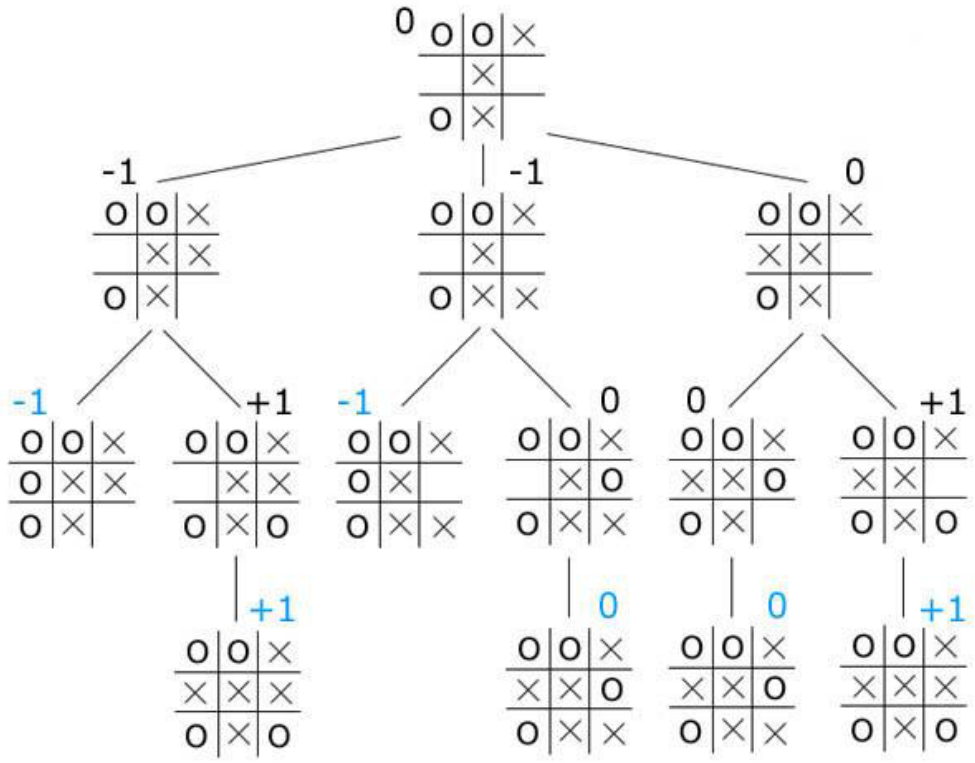
Bob wins of (at least) 3 points

By labelling the tree, we can play the *perfect game* from every position



Finite combinatorial games

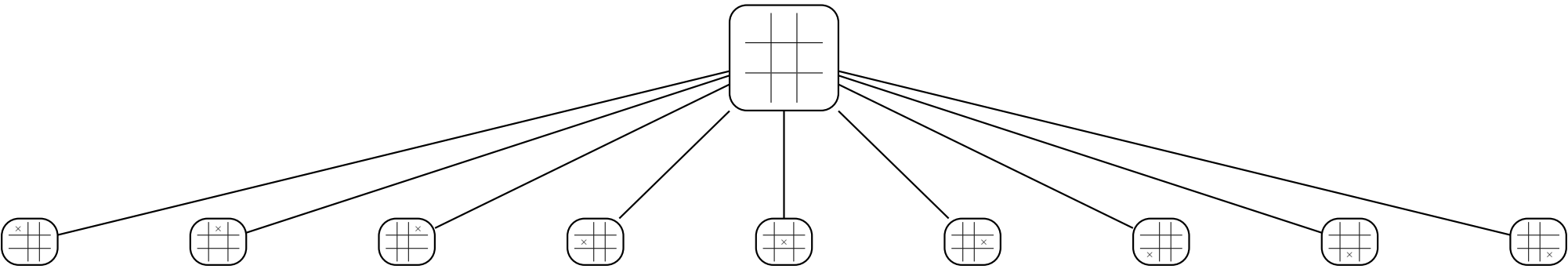
Every “Nim-like” game can be described by its complete game tree. . .



. . . and then they can be solved using MiniMax.

Finite combinatorial games

Problem: the tree grows very fast, and Mini-Max needs every leaf!



$9 \cdot 8$ nodes

$9 \cdot 8 \cdot 7$ nodes

$$\text{Leaves}=9! = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 362880$$

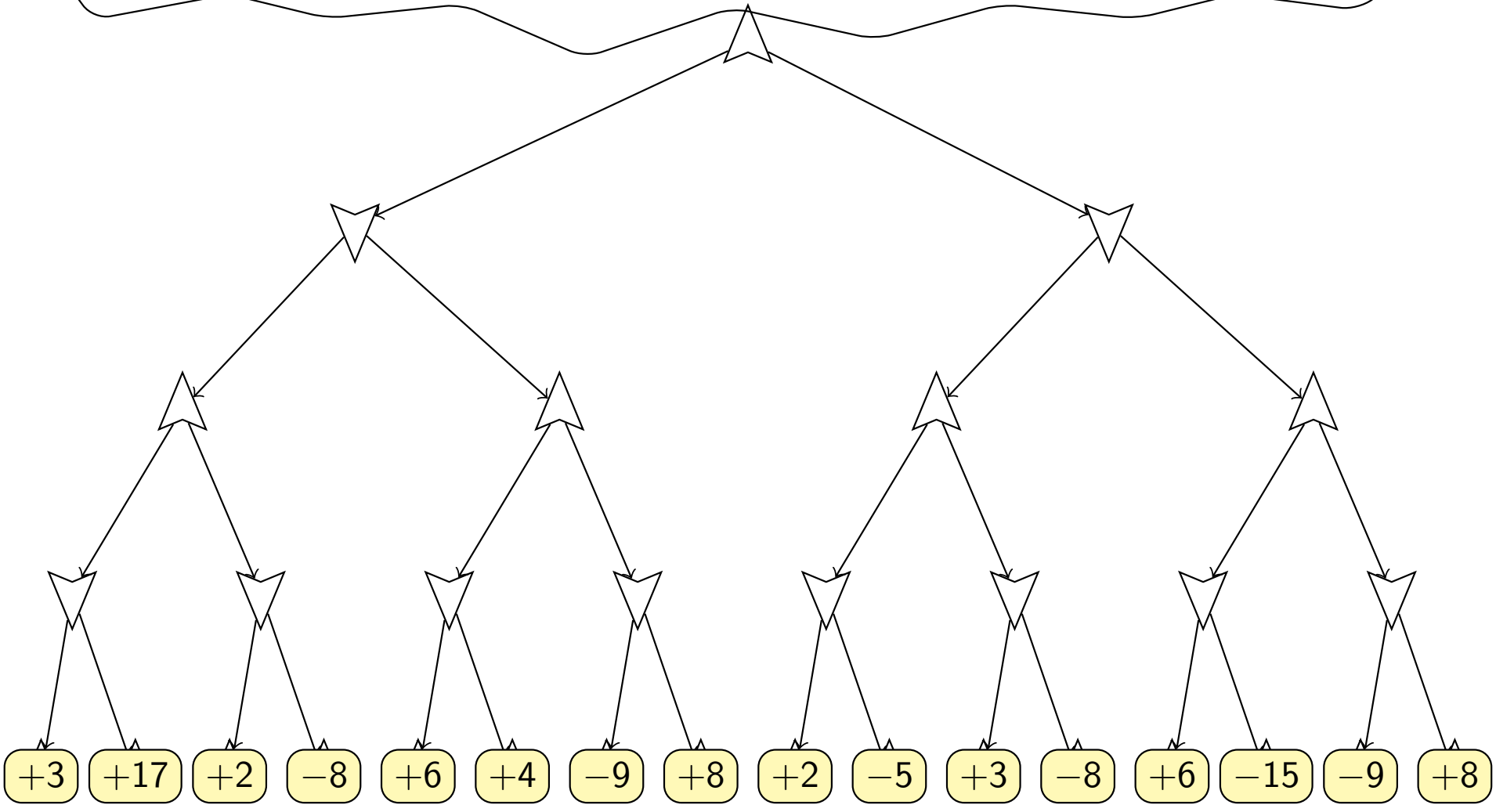
Alpha-Beta pruning

We need an algorithm to build only the part of the tree that we need!

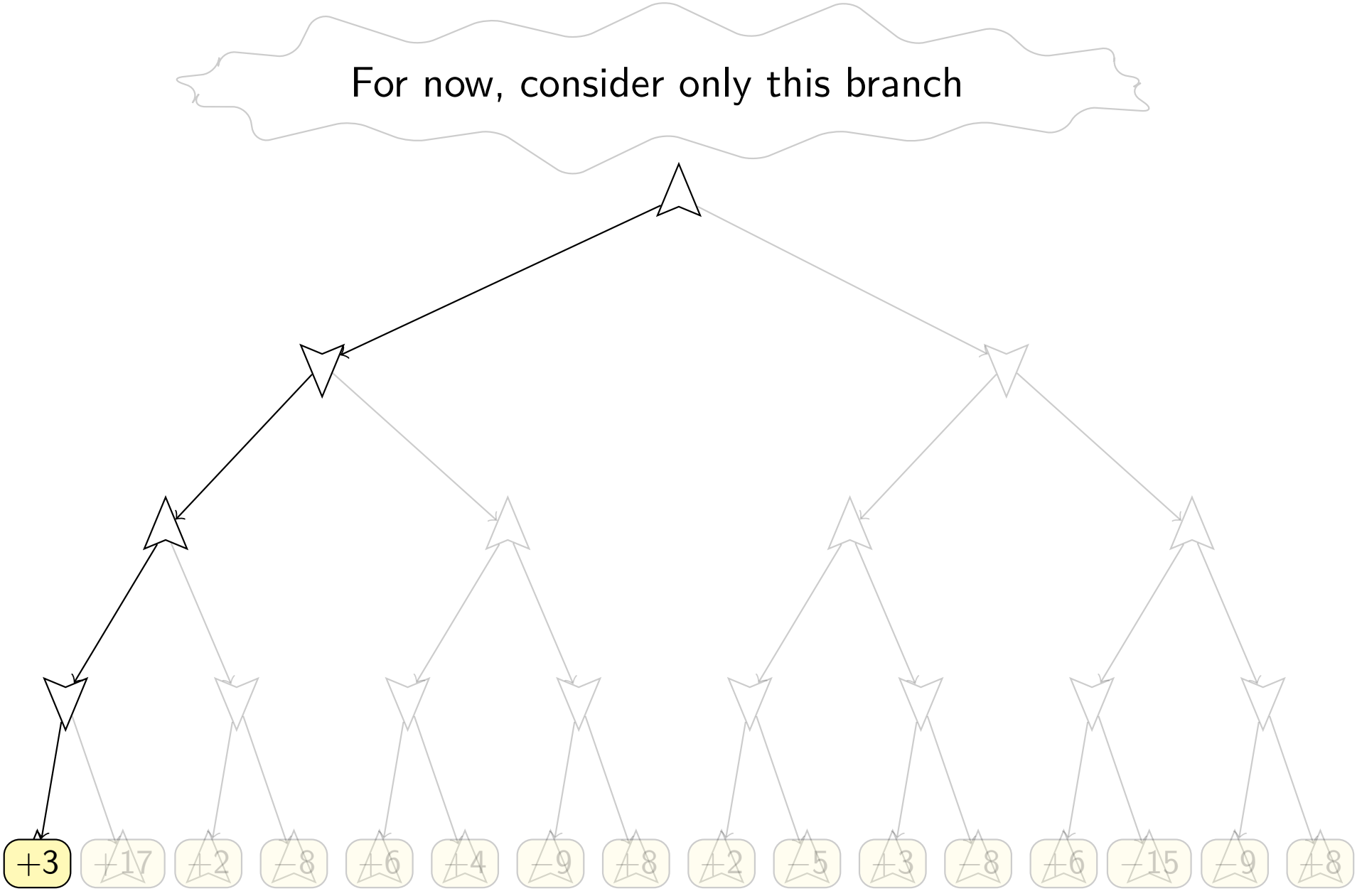
- Idea: cut off drying branches.
- "If I play here, I will gain at least 3 points. But, if I play there, my opponent can play in such a way that I will gain no more than 2 points. Thus, I will not consider anymore playing there."
- "I will not consider"="I will not build the subtree starting from there".
- *Alpha-Beta pruning* is an algorithm formalizing the above argument.

Alpha-Beta pruning

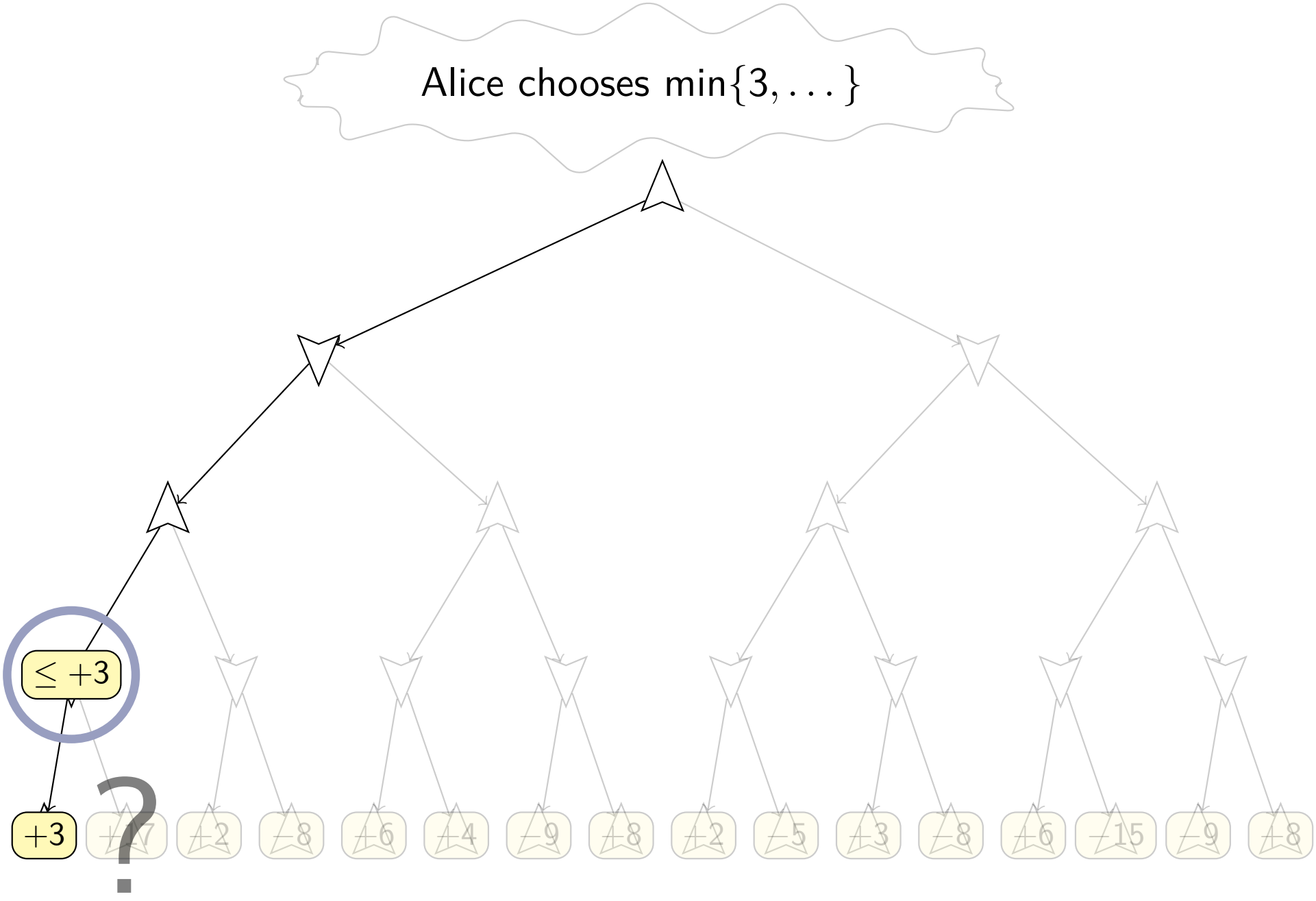
Example: binary tree, all leaves at the same depth.



Alpha-Beta pruning

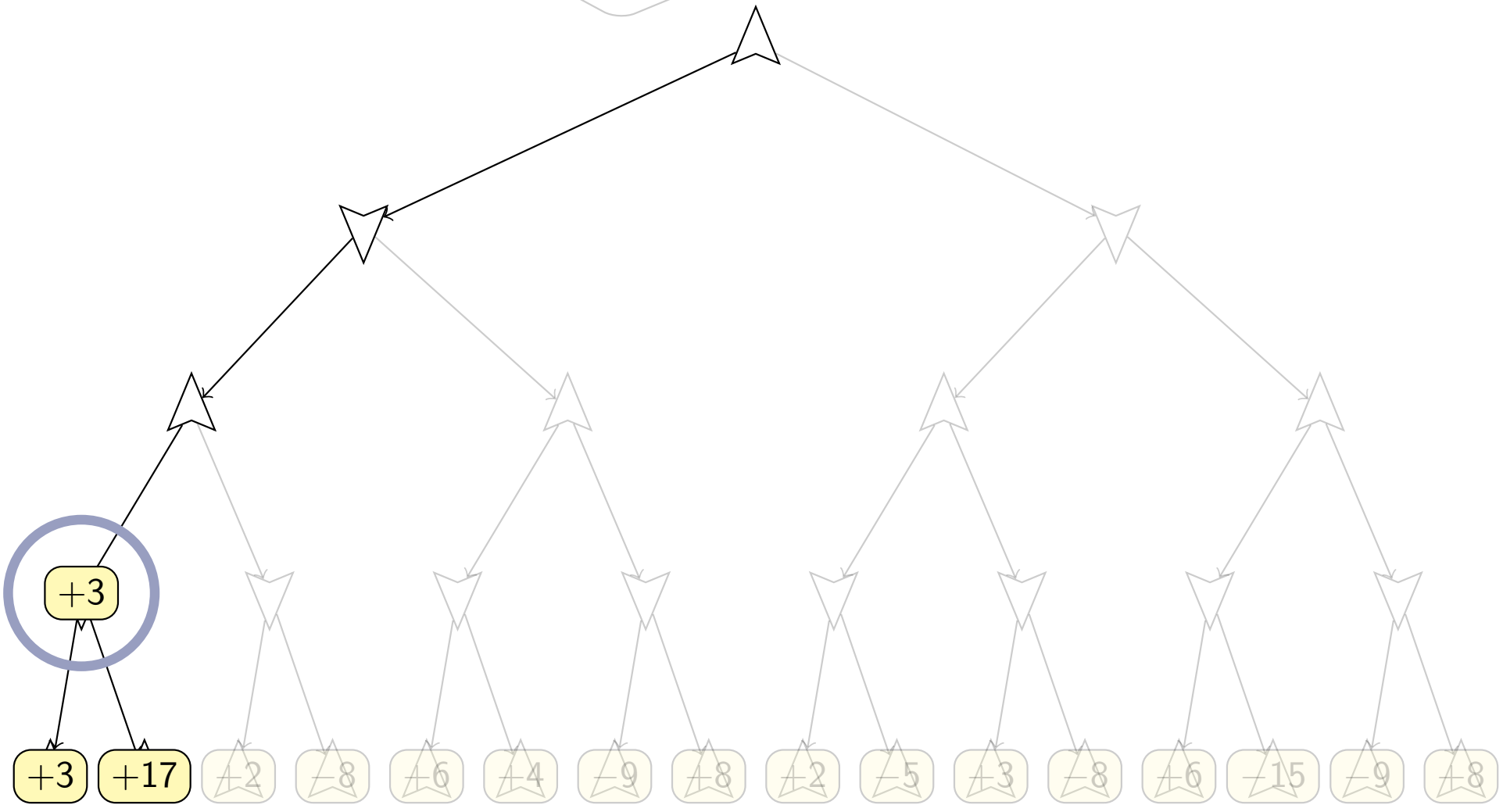


Alpha-Beta pruning

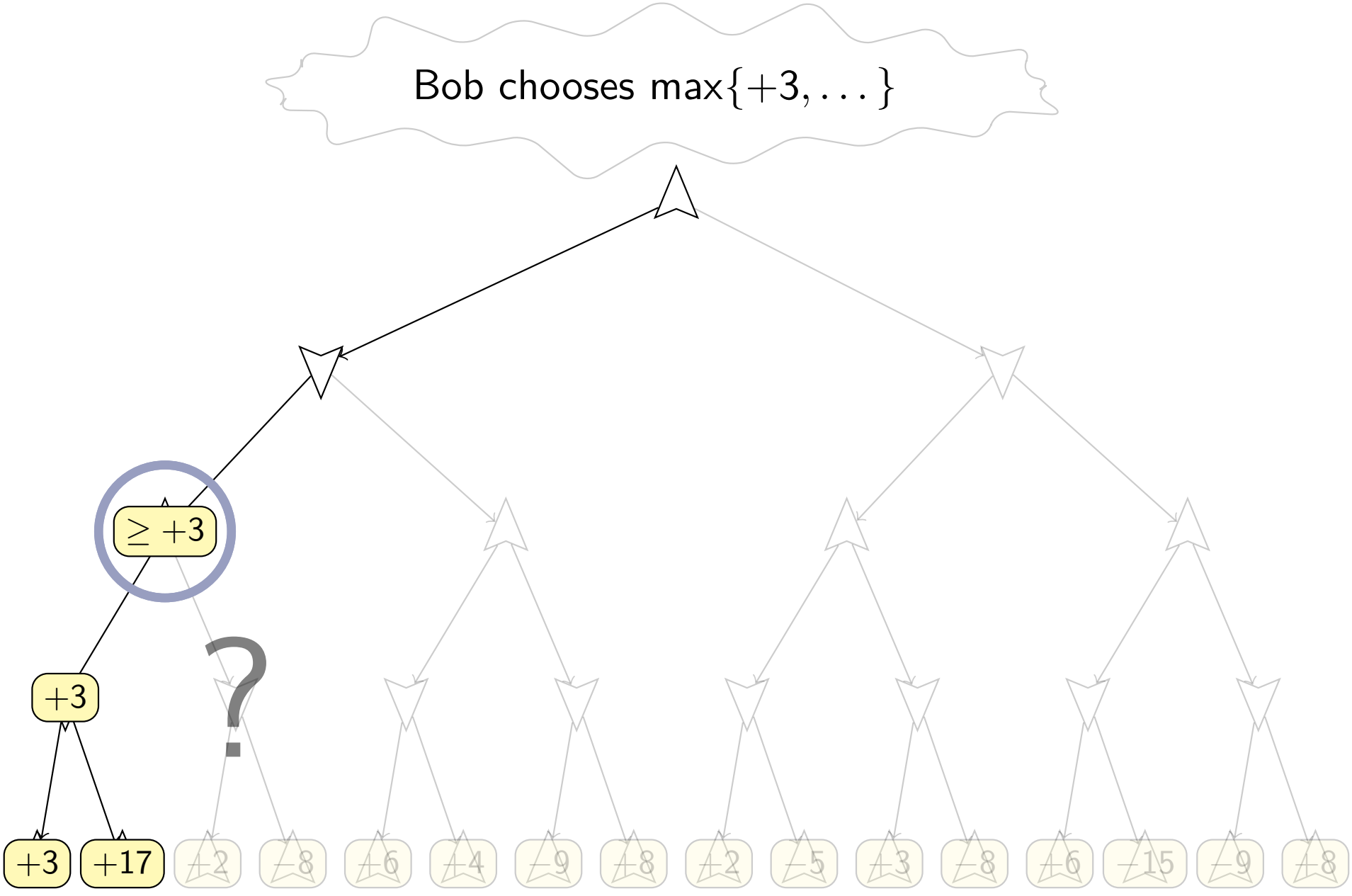


Alpha-Beta pruning

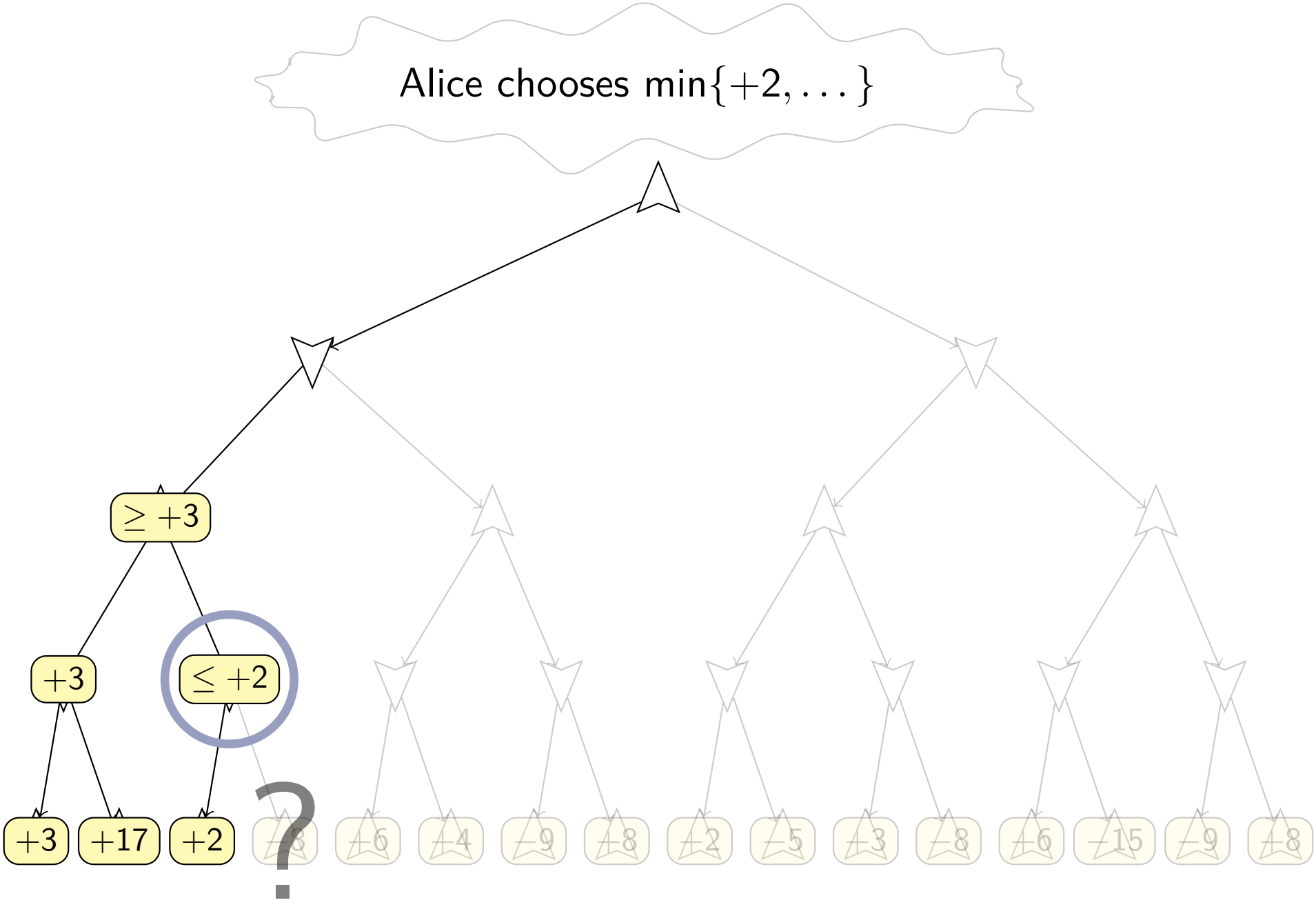
Next child is $+17$, thus Alice chooses $+3$



Alpha-Beta pruning

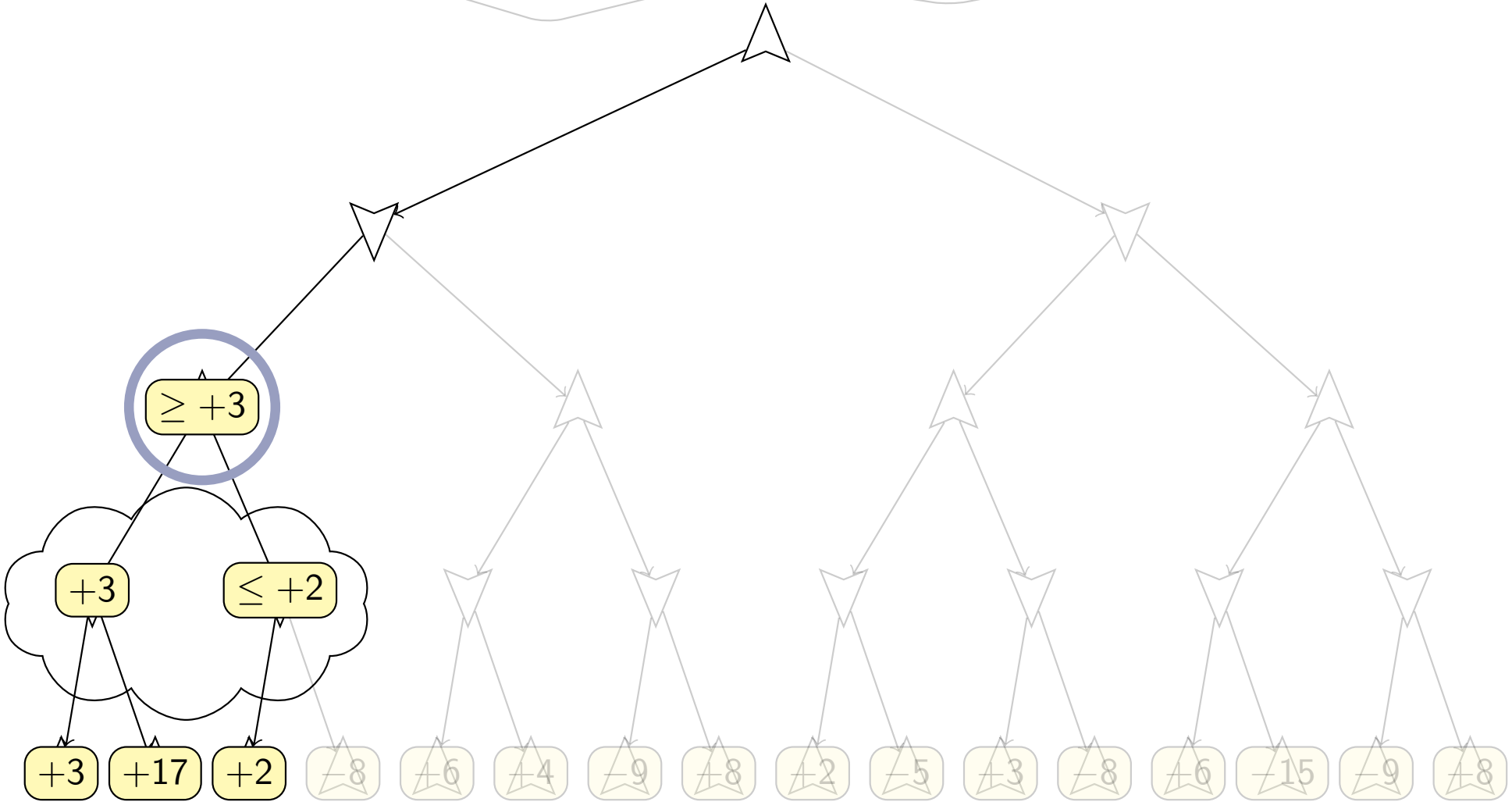


Alpha-Beta pruning

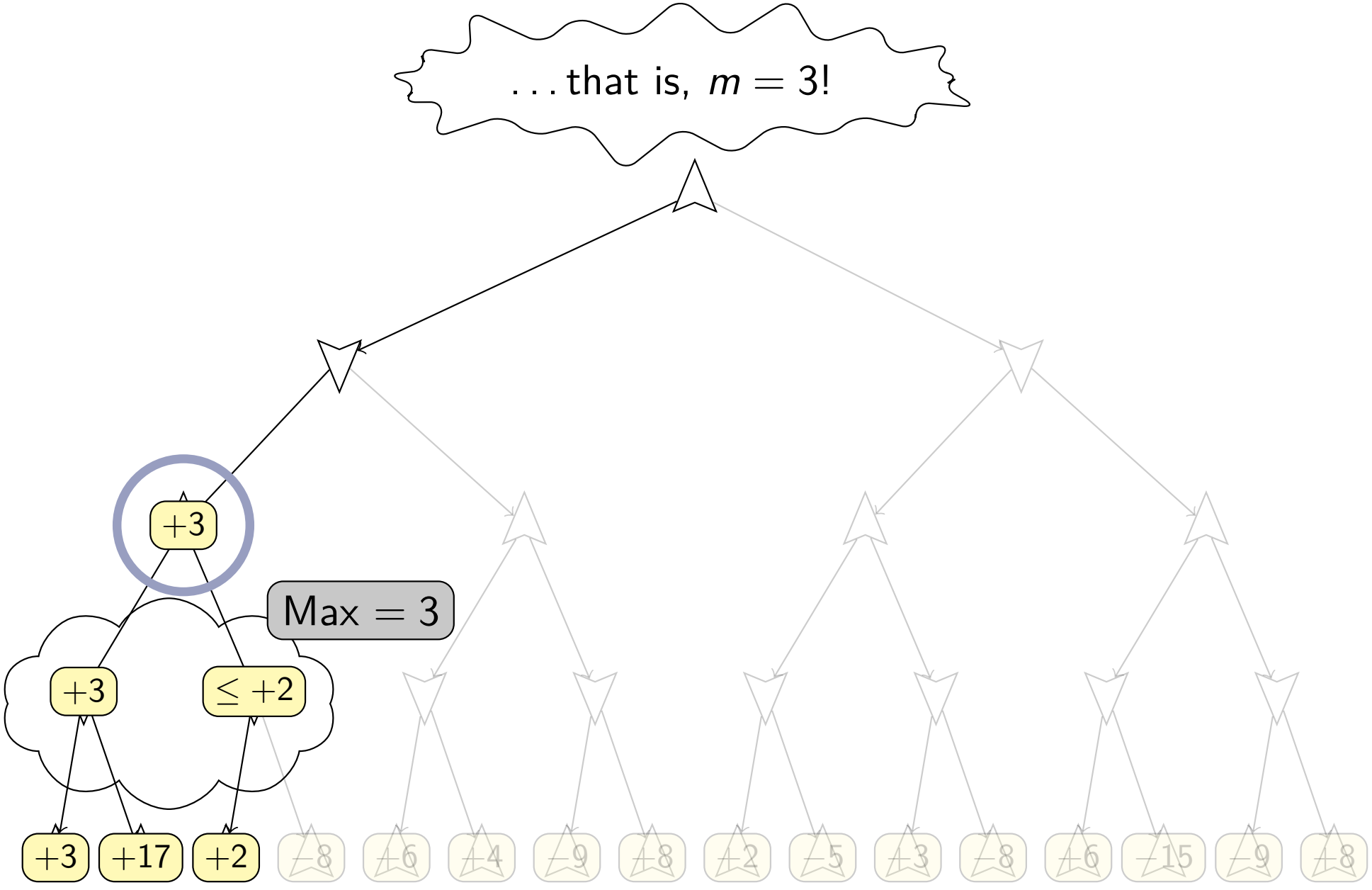


Alpha-Beta pruning

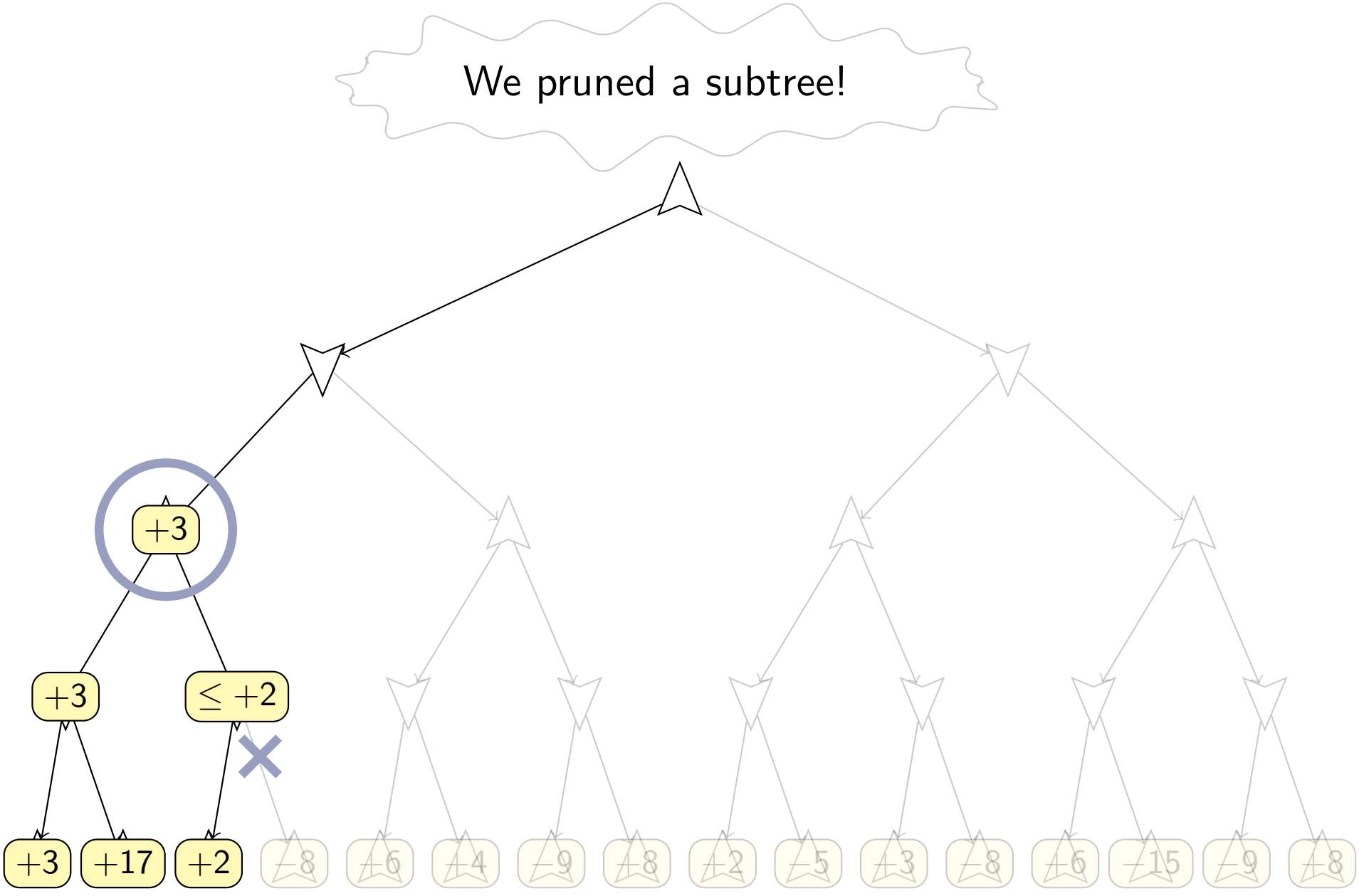
The smallest integer m satisfying 1) $m \geq 3$ and 2) $m \geq n$, where $n \leq 2 \ldots$



Alpha-Beta pruning

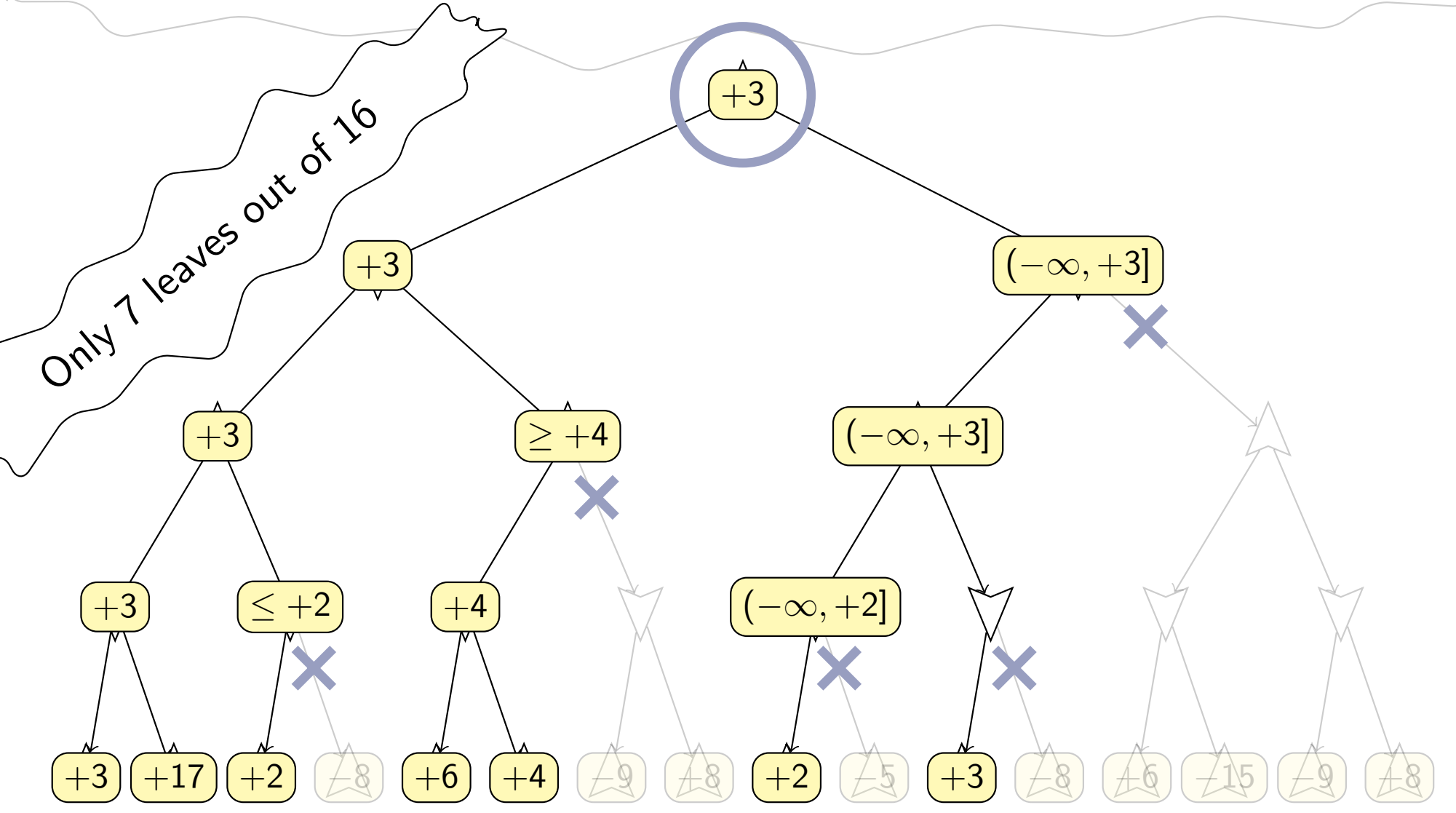


Alpha-Beta pruning



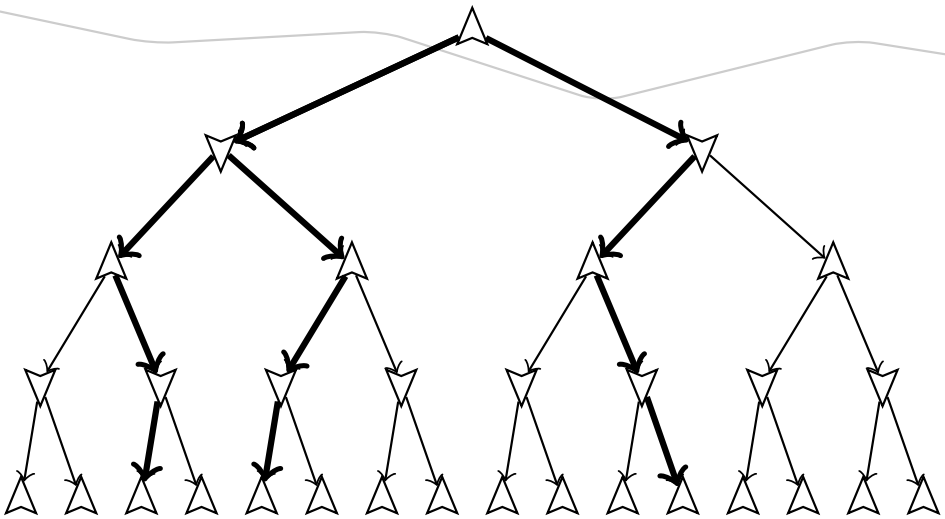
Alpha-Beta pruning

At the end, we obtain the same root value as before!



Game-tree complexity

Game-tree complexity b^d depends on the game's average branching factor (breadth b) and on the game's average number of plies (depth d)



2 nodes

$2 \cdot 2 = 4$ nodes

$2 \cdot 2 \cdot 2 = 8$ nodes

$2 \cdot 2 \cdot 2 \cdot 2 = 16$ leaves (=paths)

Real games have a high *branching factor*

Draughts: $d = 20, b = 10$. 10^{20} leaves!

Chess: $d = 40, b = 20$. $20^{40} \cong 10^{52}$ leaves!

Game-tree complexity

Reversi: 10^{58} leaves. Not solved, but the machine is stronger than humans since 1980.

Draughts: 10^{20} leaves. *Chinook* starts its computation in 1989: in 2007, after 18 years of computation, the game is solved.

- Tic-tac-toe. Solved browsing the game tree, draw.
- Awari. Solved browsing the game tree, draw.
- Connect Four. Solved browsing the game tree, $\mathcal{N}$-game (who starts the game wins).
- Hex. Solved by strategy-stealing, $\mathcal{N}$-game.
- Nim. Solved browsing the game tree, the winner depends on heaps and items.

Chess: 10^{52} leaves. *Deep Blue* wins against the world champion in 1997. 200 millions of positions per second, going as deep as 40 plies. Not solved.

Game-tree complexity in Go

Go: on average, $b = 200$ and $d = 150$: $200^{150} \cong 10^{345}$

Number of atoms in the universe: 10^{80}

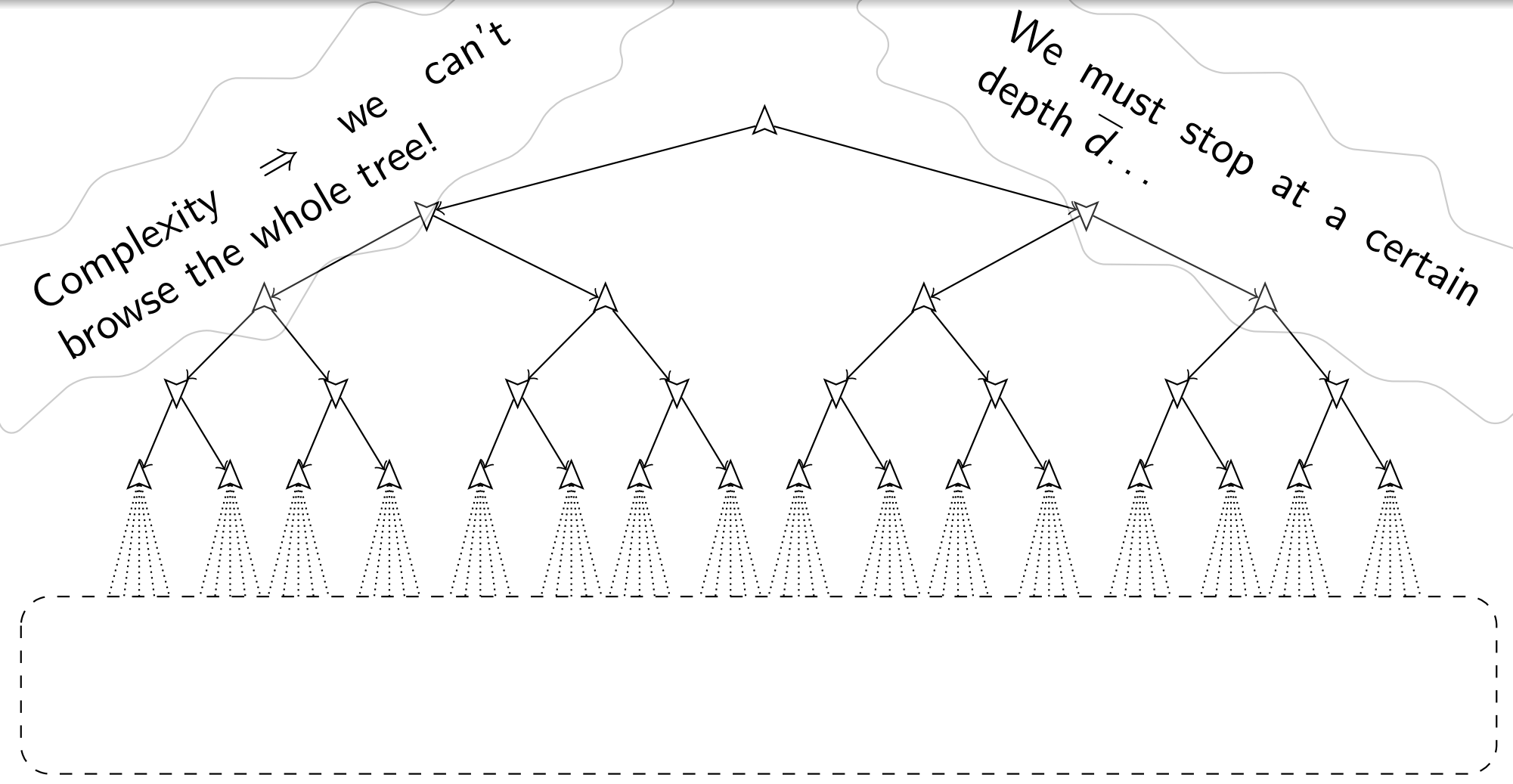
Even with Alpha-Beta pruning, in the best case we have $b = 100$

Number of particles in the universe: 10^{87}

No way to browse the whole tree, only few nodes per move.

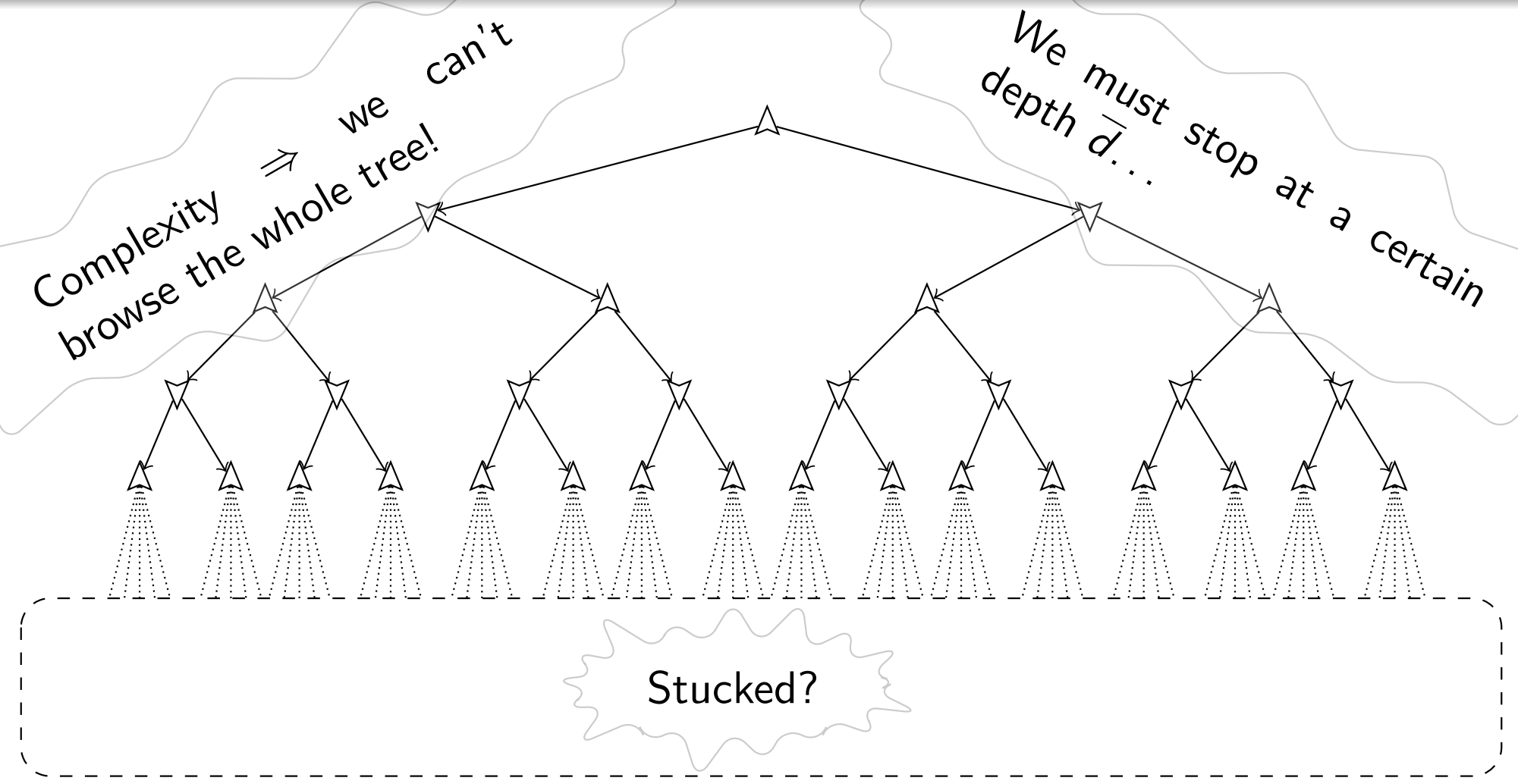
General methos to bound d and b: *value function* and *policy*.

Value function



. . . but labelling starts from leaves!

Value function

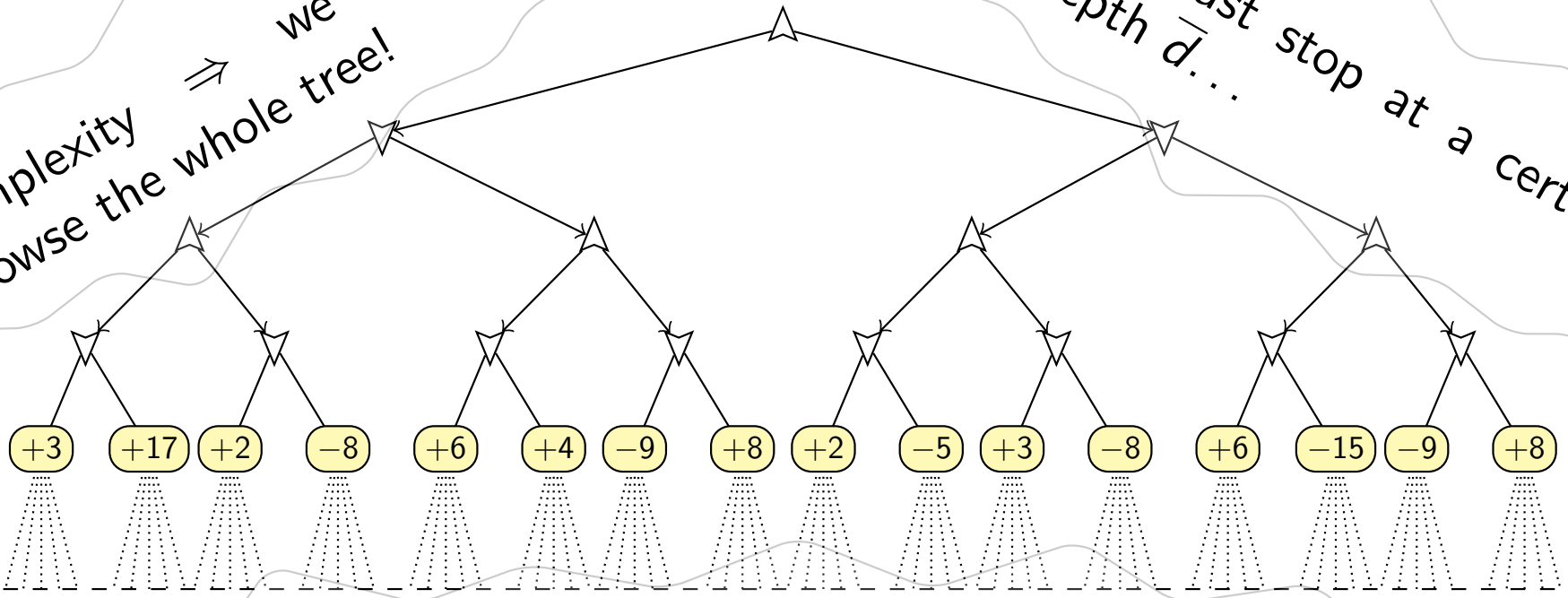


...but labelling starts from leaves!

Value function

Complexity $\Rightarrow$ we can't browse the whole tree!

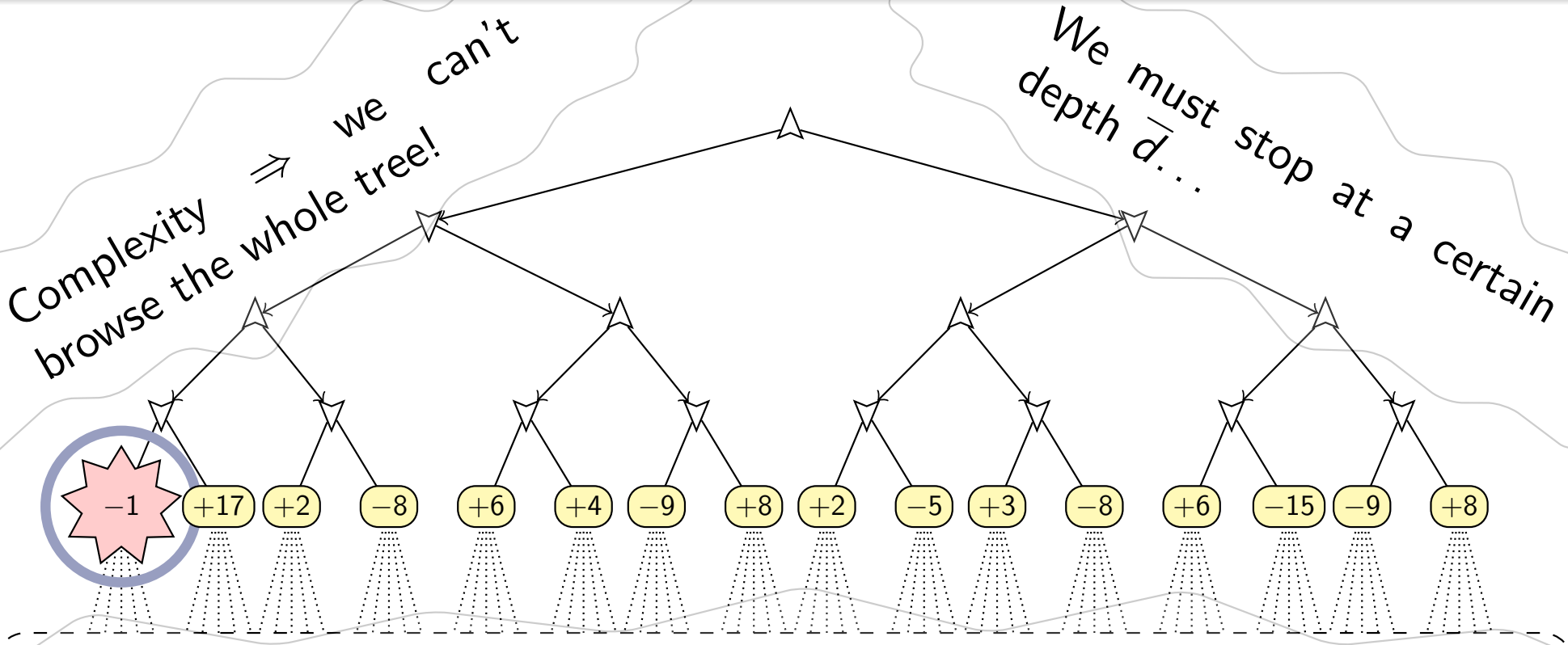
We must stop at a certain depth $\overline{d}$...



Solution: a genius of the lamp tells us the right values for the "leaves" at depth $\overline{d}$!

This genius has a name: *value function*

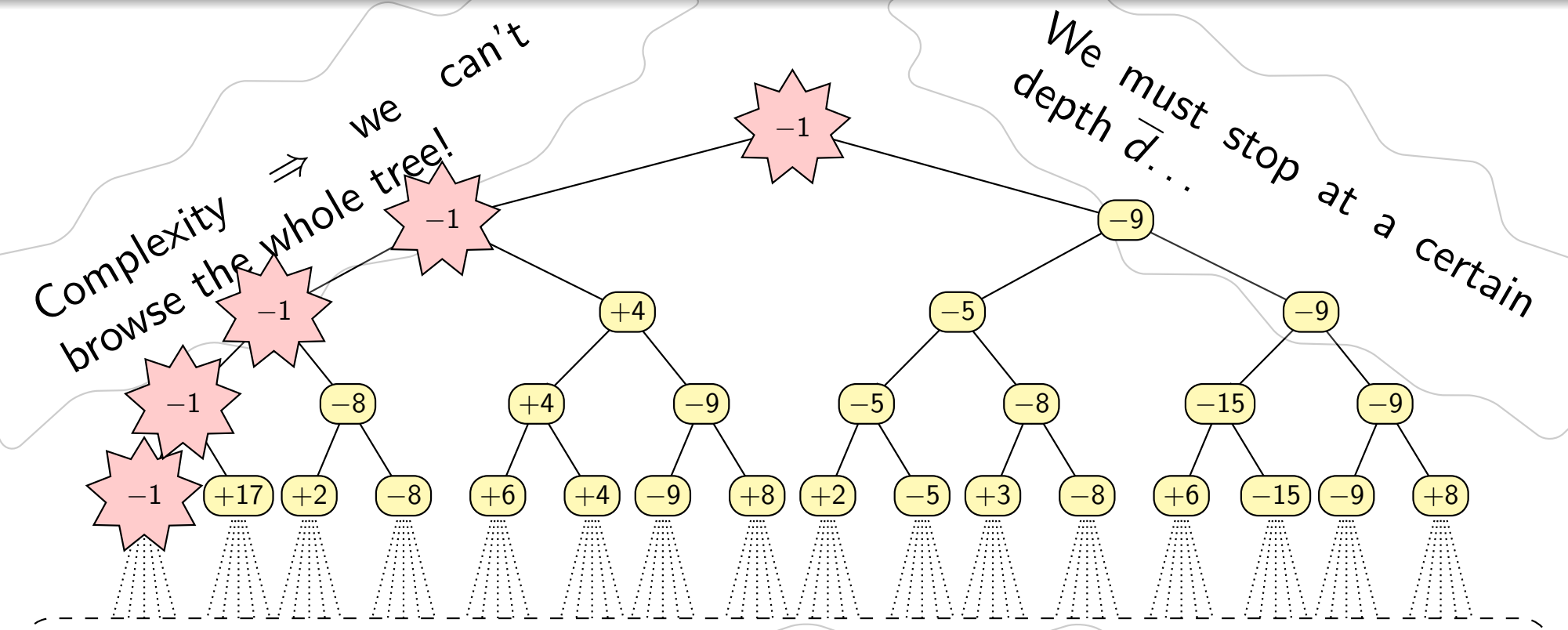
Value function



Be careful: if the value function is wrong, the final result can be completely wrong...

This genius has a name: *value function*

Value function



. . . and the winner can turn into a loser!

This genius has a name: *value function*

The value function

- Every time we think “If I reach this position, I win” or “If I’m forced to reach that position, I lose”, we are *evaluating* a position with some personal value function, to prune the game-tree!
- Draughts, Chess and Go players evaluate in every position if they are winning or losing. Their evaluation is personal!
- For a machine, evaluating a position in Go was impossible, up to October 2015!

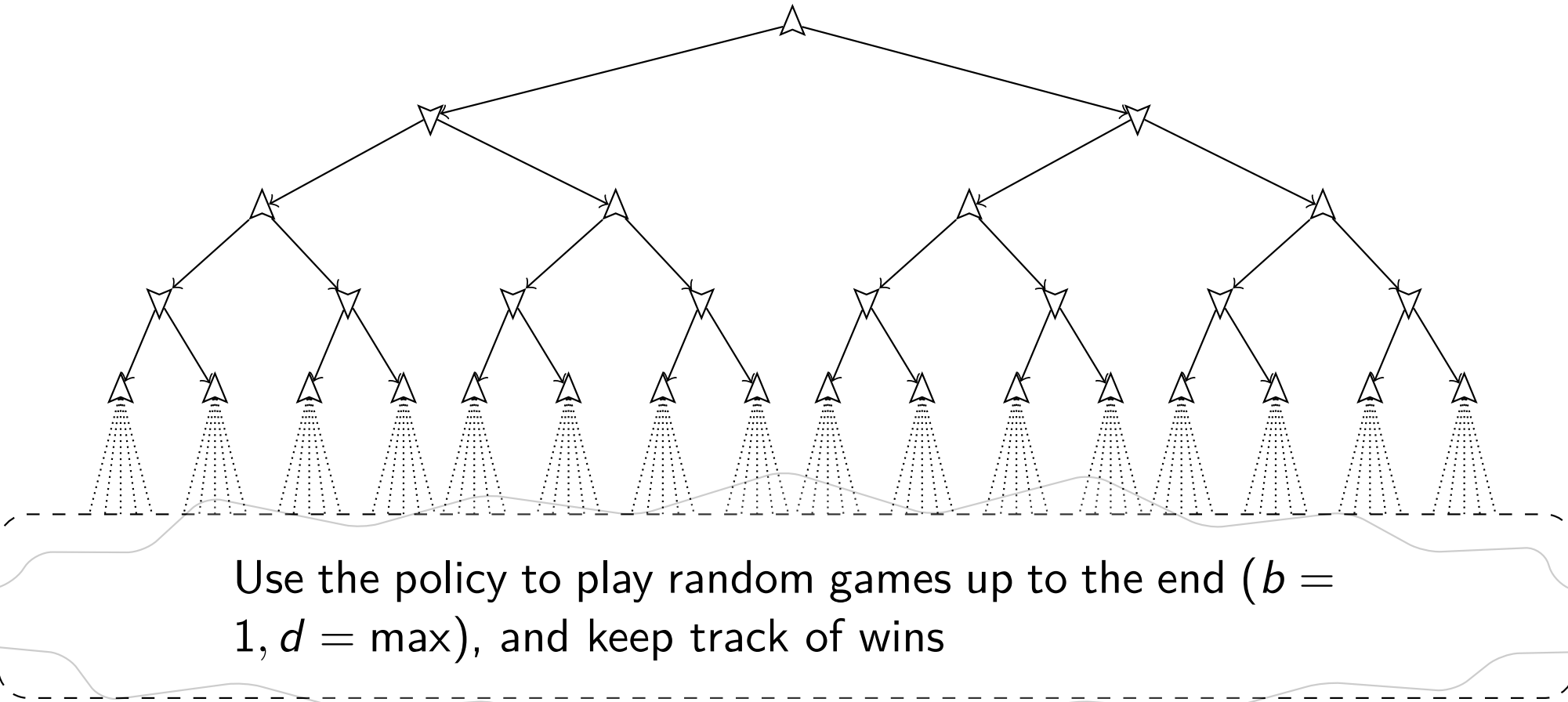
Examples of value function

- Draughts. Men value is 1, kings value is 2, and this values are weighted by the distance from the first line.
- Chess. The value of a position uses the pieces still in the game: pawn 1, bishop and knight 3, rook 5, queen 10.
- Deep Blue uses a weighted sum of several features: $w_1 \cdot C_1 + \cdots + w_n \cdot C_n$, where $n = 8000$.
- Nothing similar in Go!

The policy

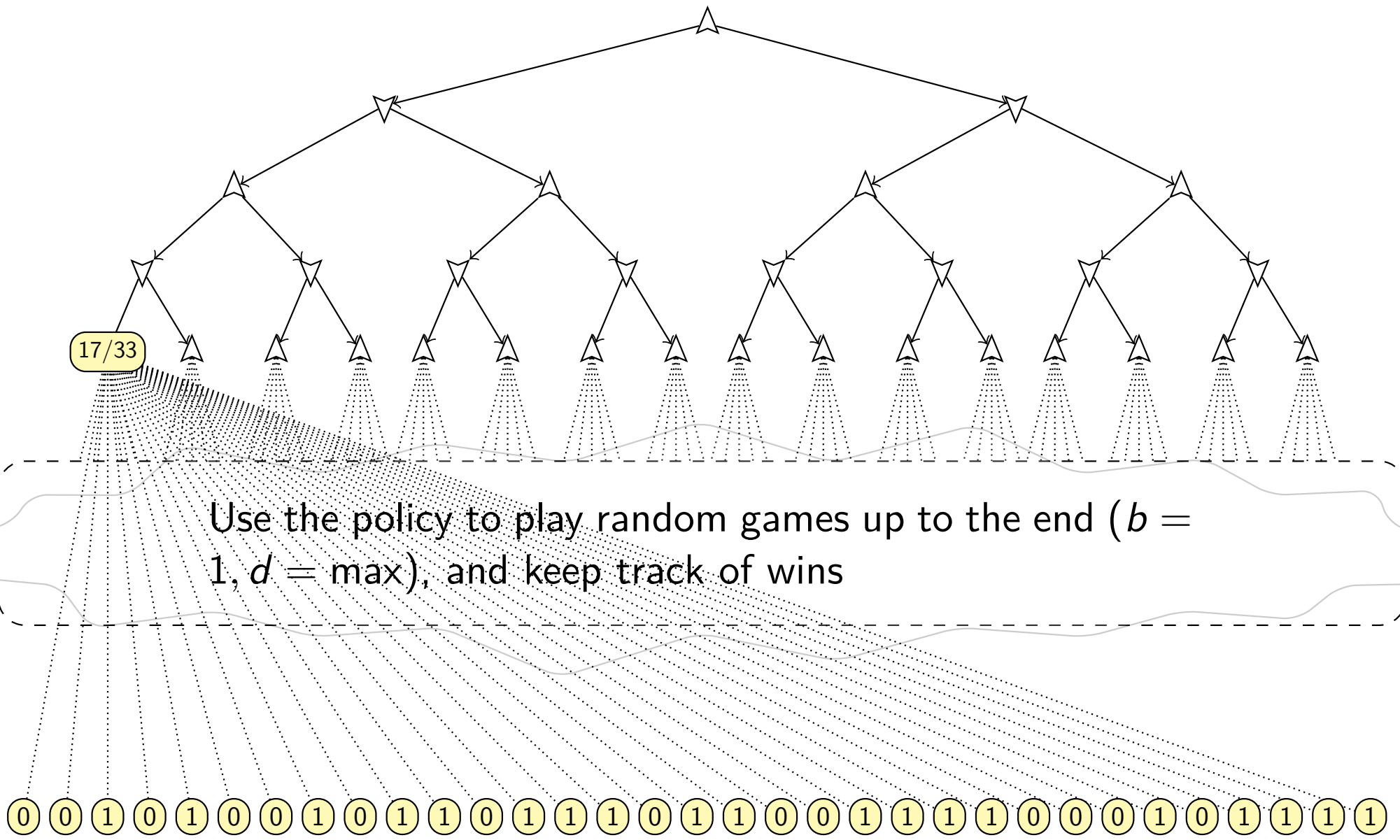
- "If I play here, he will probably play there. . . ". Keyword: *probably*.
- Assume we hire an expert player that for every game state s tells us, for every possible position a reachable from s: "I would play a with probability $p(a|s)$", we could use this information to bound the breadth b in our game tree search.
- The probability distribution $p(a|s)$ is called *selection policy*. It is a vector $\mathbf{p}$ with $19^2 + 1 = 362$ components $P(a, s)$ (every board point, and the pass move).

Pure Monte Carlo Tree Search: from selection policy to value function

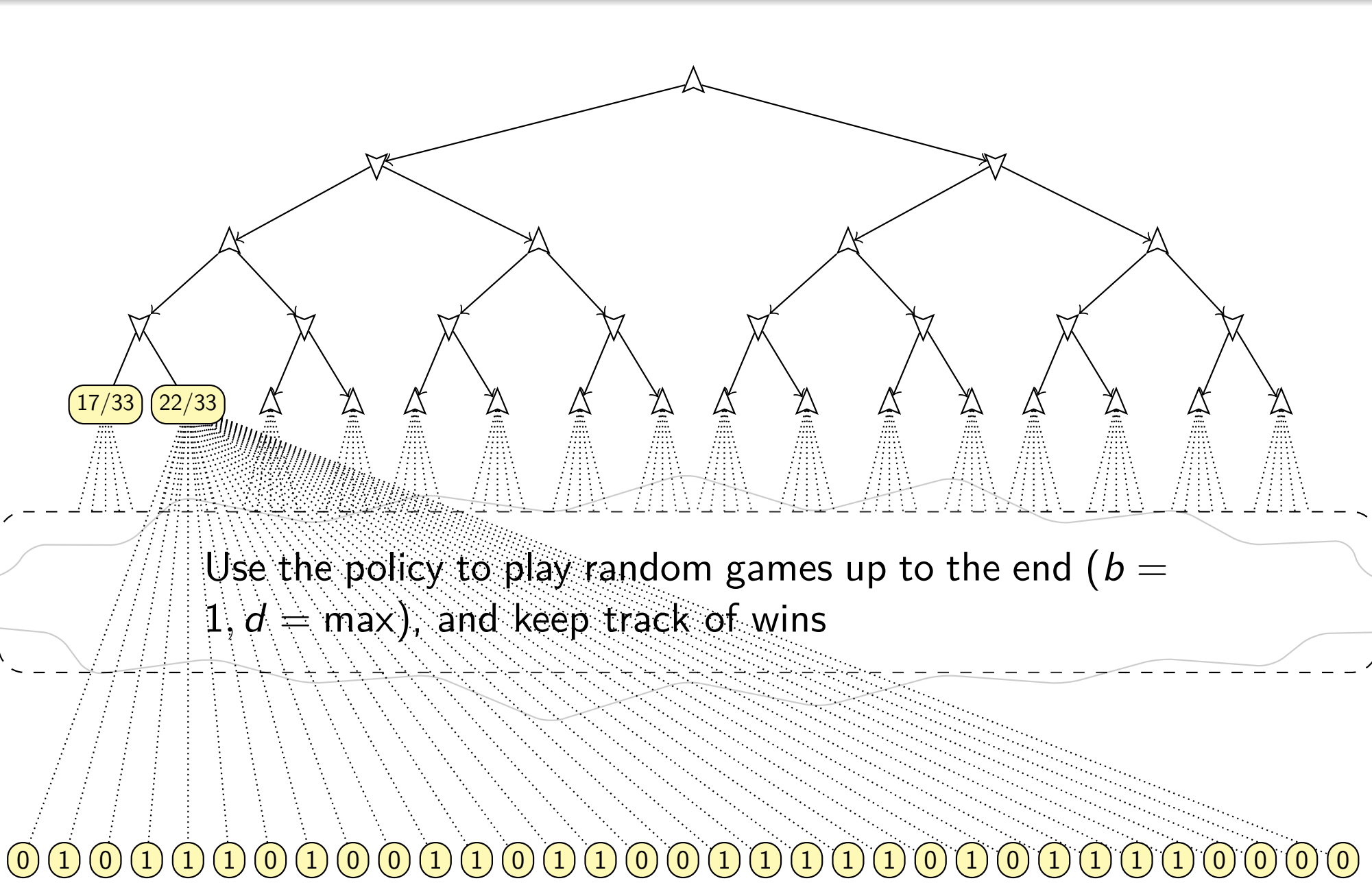


Use the policy to play random games up to the end ($b = 1, d = \max$), and keep track of wins

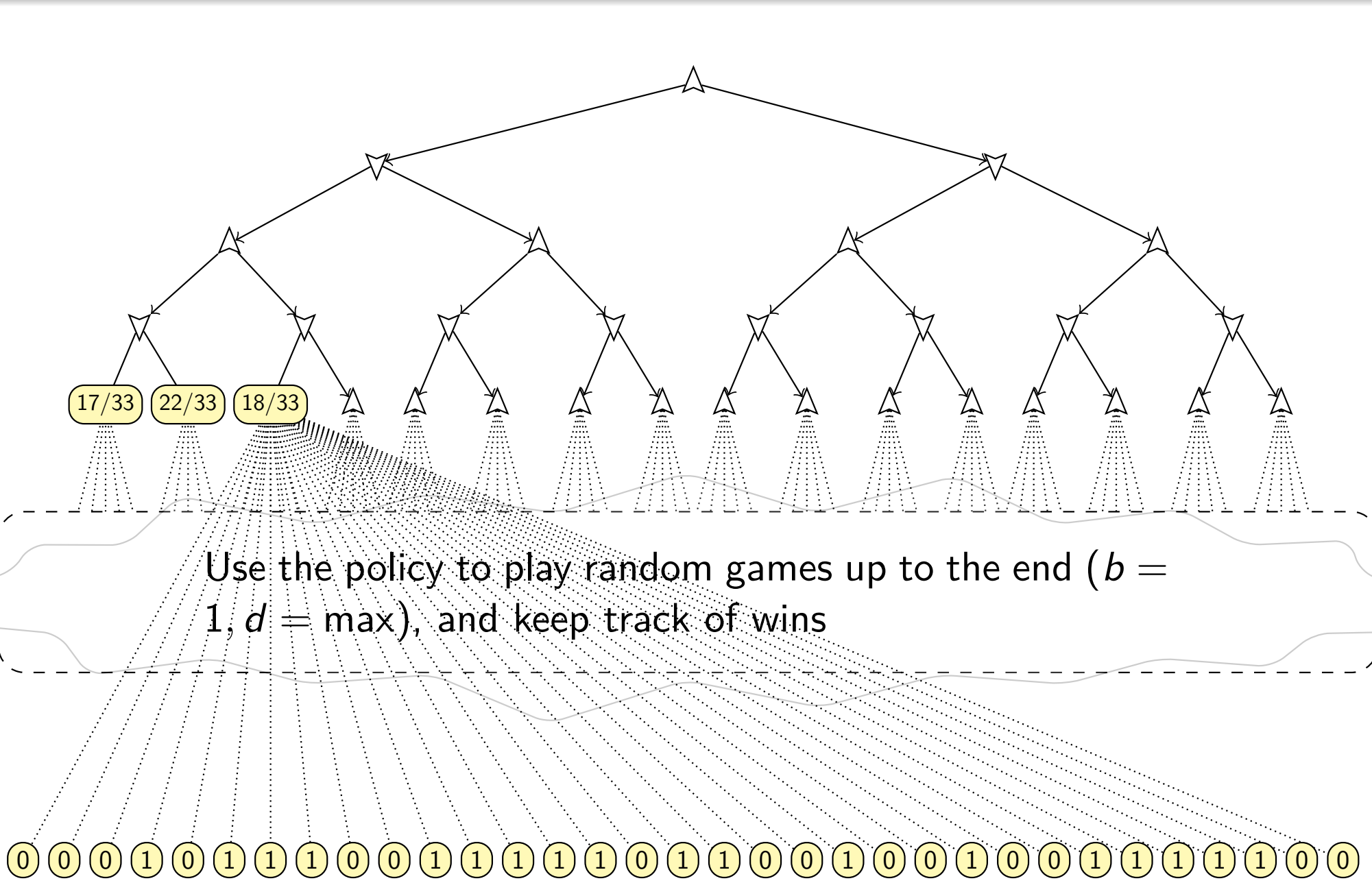
Pure Monte Carlo Tree Search: from selection policy to value function



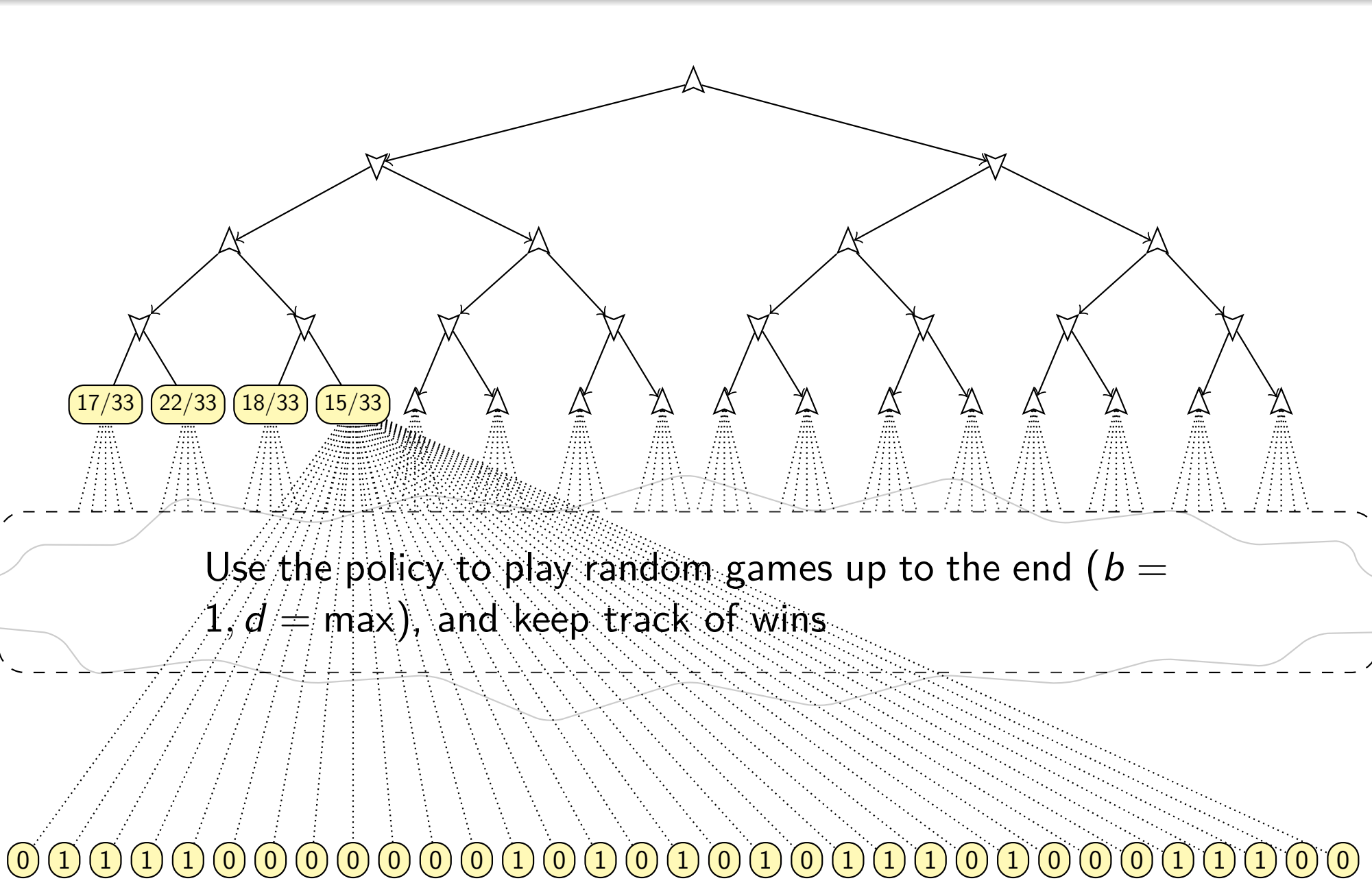
Pure Monte Carlo Tree Search: from selection policy to value function



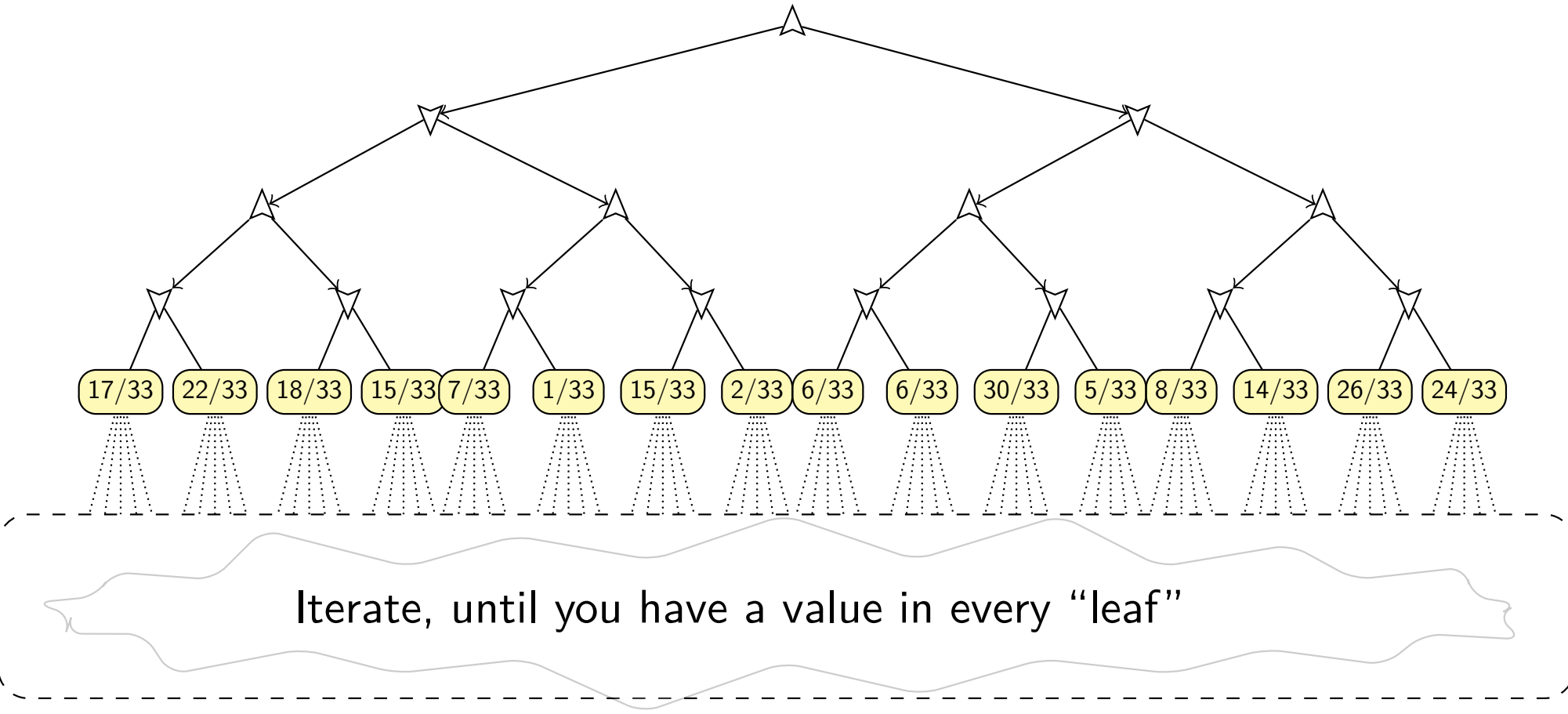
Pure Monte Carlo Tree Search: from selection policy to value function



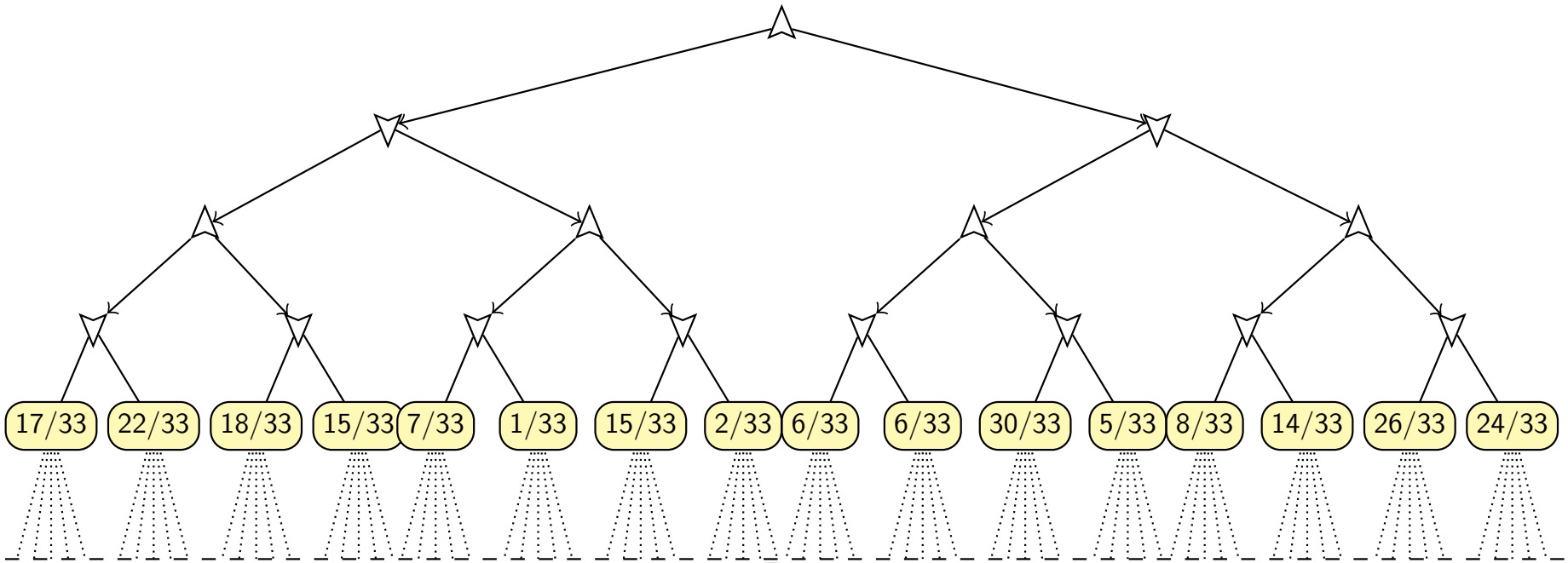
Pure Monte Carlo Tree Search: from selection policy to value function



Pure Monte Carlo Tree Search: from selection policy to value function



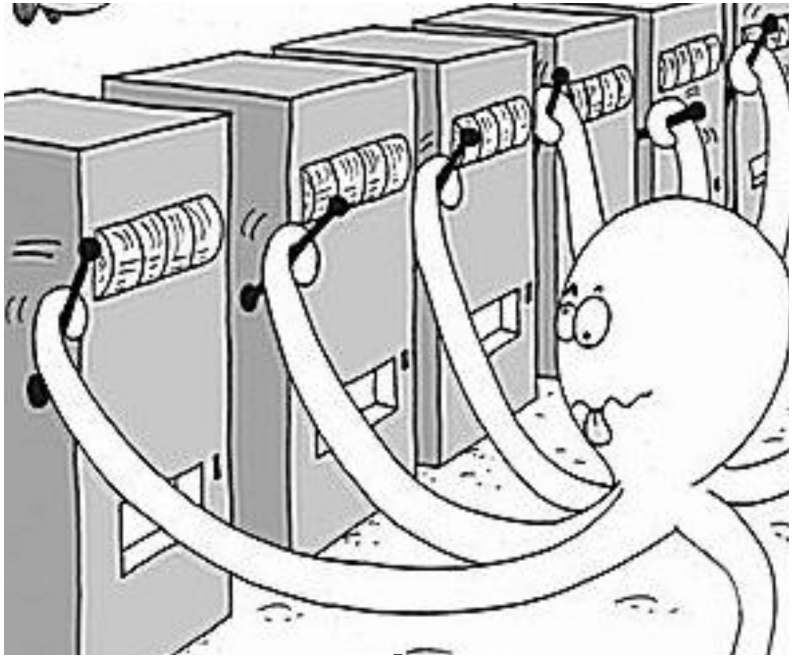
Pure Monte Carlo Tree Search: from selection policy to value function



Iterate, until you have a value in every "leaf"

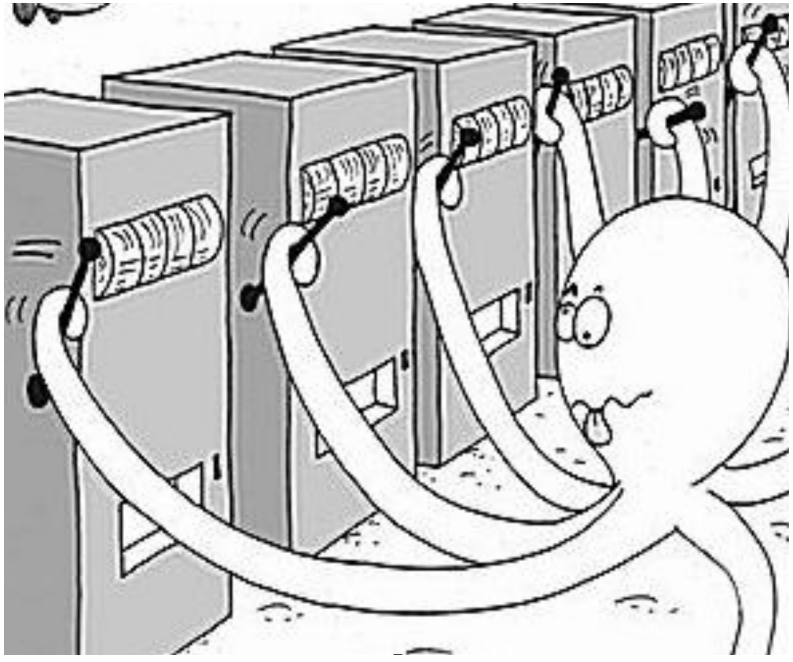
Problems: 1) this value function is strongly dependant on the policy, and 2) this policy is fixed, doesn't learn!

Multi-armed bandit and MCTS



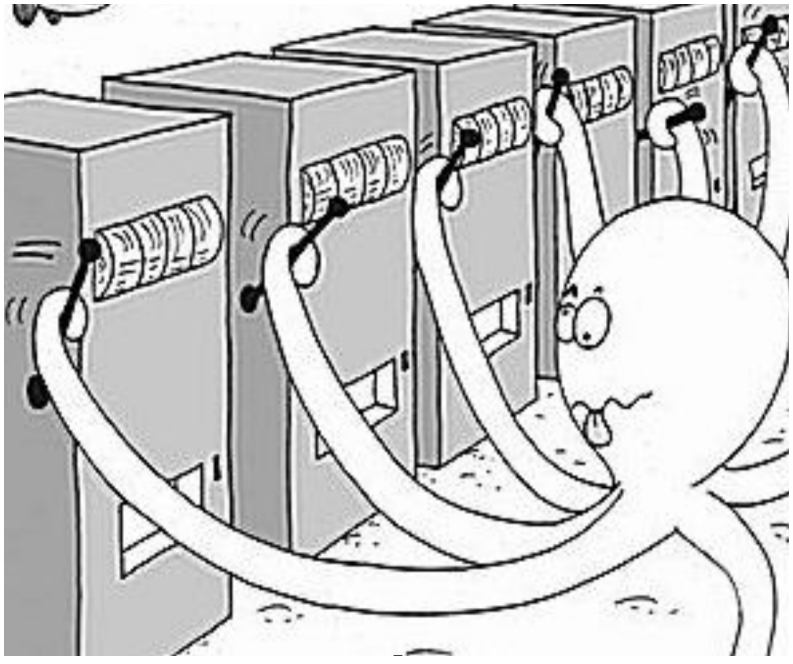
- Slot machines, with random reward from different *unknown* probability distributions.
- Task: maximize the win.
- Strategy: wait, look at other people playing, and guess the best machine by the law of large numbers.

Multi-armed bandit and MCTS



- Nobody there? Tired of waiting? Trade-off between *exploration* and *exploitation*.
- During playing, take note of the average win $\bar{E}$, and estimate the error ϵ.
- Strategy: choose the machine maximizing $\bar{E} + \epsilon$.

Multi-armed bandit and MCTS

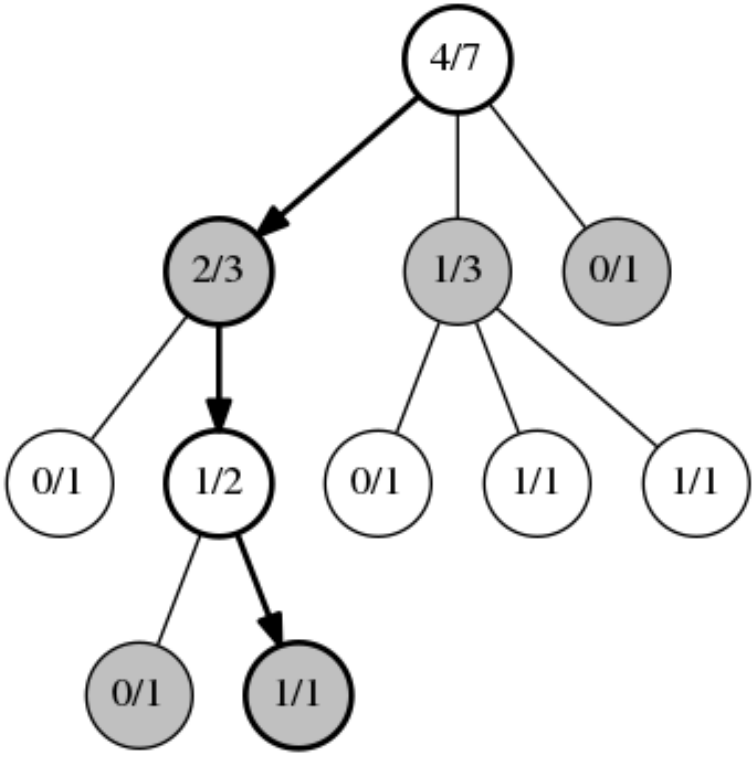


- Average win: W/n, where $n =$ how many times I used this particular machine.
- Error wrt the true, unknown average after N plays: $\sim \sqrt{\ln(N)/n}$

Multi-armed bandit and MCTS

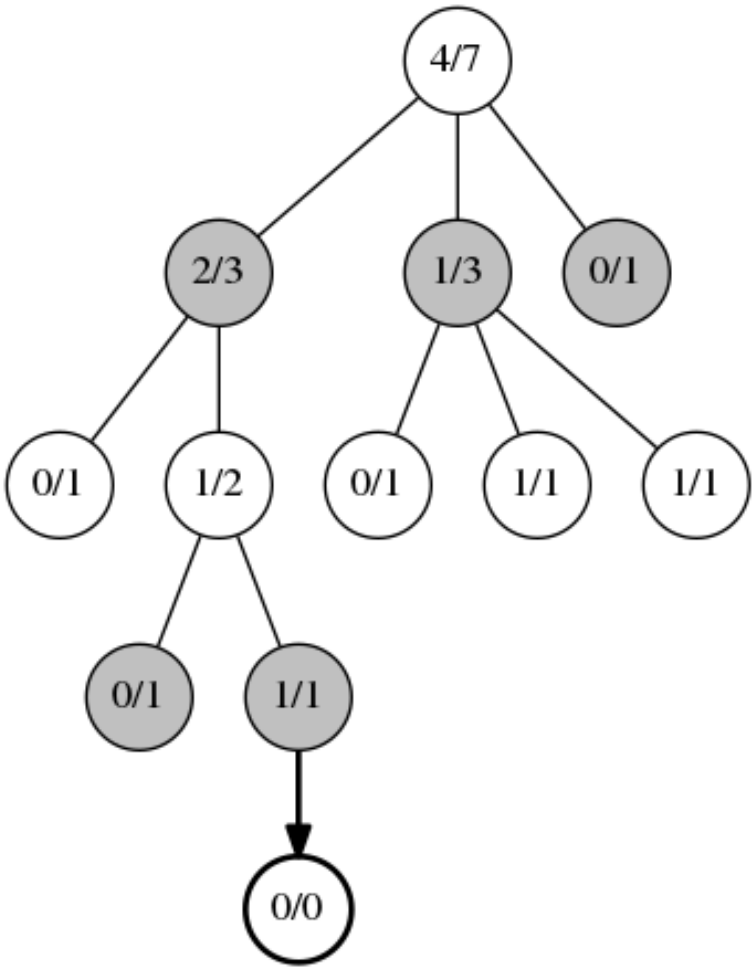
- Idea: best move = best slot machine.
- *Select* the move using multi-armed bandit.
- When a leaf s_L (a node without already expanded children) is reached, *expand* it: create one or more child nodes and select one of them C.
- *Simulate* a game starting from C with pure Monte Carlo or other methods (this is the first play at the slot machines).
- *Propagate* back through all the parents the result.

Multi-armed bandit and MCTS



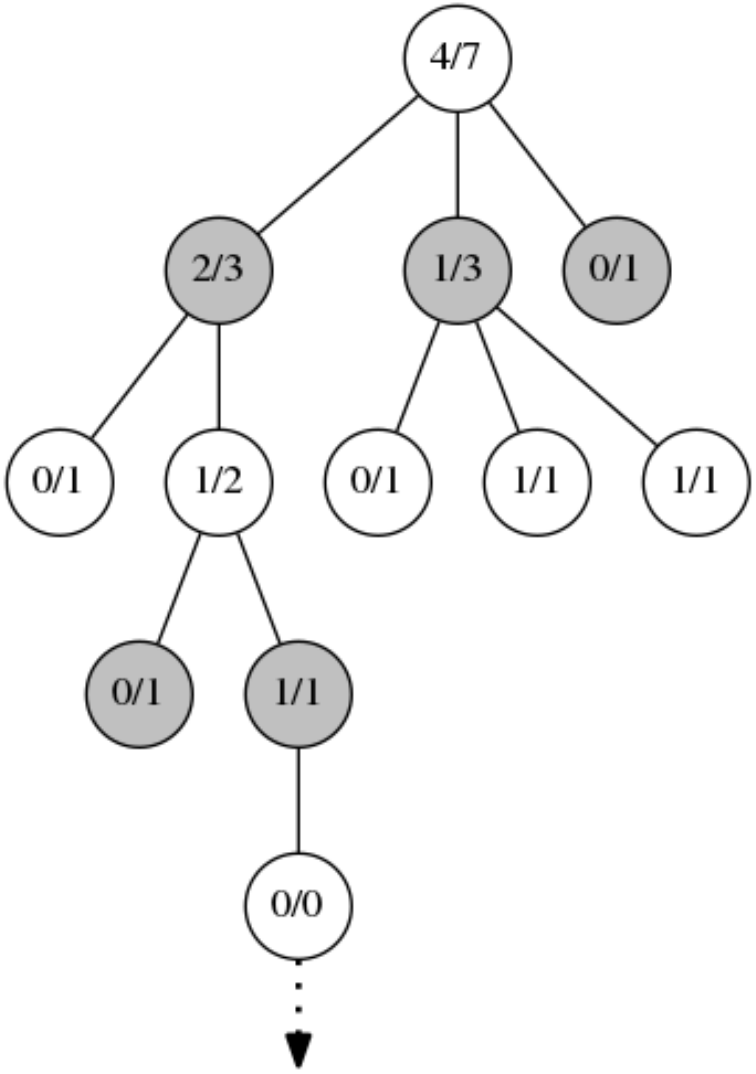
Selection. The *selected* moves are in boldface. The statistics inside the nodes come from previous playouts. In the nodes, number of wins / visits.

Multi-armed bandit and MCTS



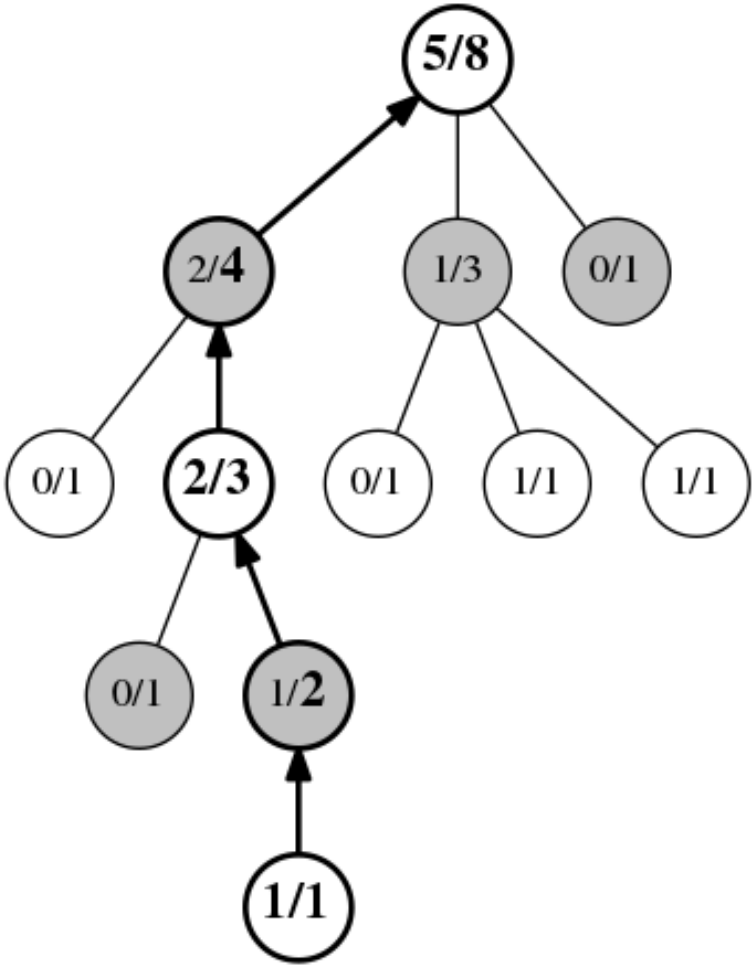
Expansion. The value $0/0$ denotes we selected a leaf. If it is not a terminal leaf, we *expand* it, see next step.

Multi-armed bandit and MCTS



Simulation. We *simulate* a game up to the end, using Monte Carlo. This step requires a selection policy.

Multi-armed bandit and MCTS



Propagate back the result of the playout (*update* the nodes).
Every node that contributed is updated.

Summing up

- Learn these terms: Nim, tree, root, nodes, leaves, MiniMax algorithm, labeling a tree, coloring a tree, perfect game, solved game, alpha-beta pruning, space complexity, time complexity, depth, breadth, 10^{80}, value function, policy, Monte Carlo Tree Search MCTS, selection, expansion, simulation, propagation, UCT.
- Every combinatorial finite game is a tree.
- The number of atoms in the universe is 10^{80}. Every problem involving numbers close to 10^{80} cannot be solved by brute force.
- A Monte Carlo method is any method using a probability distribution to sample solutions to a problem. In several cases, the law of large numbers can be used to approximate the solution to the problem.
- Upper confidence bound method applied to tree search = UCT is a clever way of browsing a tree. Clever = good compromise between exploration and exploitation.

Summing up

- This section was not about neural networks, but MCTS is a wide spread technique worth to know. And it is used in AlphaGo.

Almost the end

1. What are AI, ML, NN? ✓
2. (Very quick) historical introduction to AI ✓
3. Functions and neural networks as (computational) graphs ✓
4. Representational power of neural networks ✓
5. Looking for weights aka Learning ✓
6. Deep neural networks ✓
7. Examples of NN ✓
8. Monte Carlo Tree Search ✓
9. AlphaGo and his family

AlphaGo, finally!

What do you need to play Go?

- a value *function* (unknown);
- a policy, that is, a vector-valued *function* (unknown).

What is a NN?

A NN is a (way of approximating) a function (usually unknown).

AlphaGo = *value NN* + *policy NN* + MCTS.

AlphaGo, details

Which kind of NN?

Idea: CNN architecture works very well in image recognition, maybe it can also "recognize" Go positions.

Fact: Go is a *geometric* game: we talk about *shapes*.

Architecture

AlphaGo: CNN for policy (input $48 \cdot 19^2$, 13 layers, 192 filters), CNN for value function (input layer $49 \cdot 19^2$, 14 layers, 192 filters), fast CNN for MCTS simulations.

AlphaGo, training

Policy NN 1

1. Training set = 30 millions of positions extracted from *expert human games*. Expert, but *not professional players*. Accuracy = 57%, high, was 44%) before. SL.
2. RL. AlphaGo is trying to improve *without human assistance*, for this reason no professional games have been used.

Value NN

Training set = 30 millions of positions extracted from games played by the policy network. RL.

Policy NN 2, fast

Like policy NN 1, but smaller. 1500 times faster than policy NN 1 (2 μs versus $3ms$), used for MCTS playouts. Trained over 8 millions of human positions. SL.

AlphaGo, MCTS

AlphaGo uses a MCTS variant

- *Asynchronous policy and value MCTS*: APV-MCTS.
- Selection. Choice made maximizing

$$c \cdot \text{policy} \cdot \frac{\sqrt{N}}{1+n}$$

- Expansion. Choice made by policy NN 2, SL.
- Simulation. The value of the leaf s_L is given by simulations $\rightarrow z_L$ using the fast policy and value NN $\rightarrow v(s_L)$: $(1-\lambda)v(s_L) + \lambda z_L$.

AlphaGo, domain dependance

A careful knowledge of Go

- The policy $\mathbf{p}(a|s)$ is not pure Monte Carlo. It is a function of several Go patterns: atari? eye? nakade (8192)? inside a 3×3 square with center in a, how many liberties a has? how many stones in the square? and several other Go peculiarities. AlphaGo knows 109747 characteristics Go-specific.
- State s: several planes 19×19 indicates the state of the corresponding intersection. Color, liberties, legal or not legal, when was that move played, can be captured in a ladder. In total, 48 planes, 17328 input neurons.

AlphaGo, hyperparameters

Why these?

- Temperature: $\beta = 0.67$.
- Used in MCTS to assign a value to leaves: $\lambda = 0.5$.
- Virtual loss: $n_{\text{vl}} = 3$.
- Expansion threshold: $n_{\text{thr}} = 40$.
- Exploration: $c = 5$.

AlphaGo, hardware

AlphaGo-Fan

Distributed: 1202 CPU, 176 GPU. AlphaGo-Fan vs Fan Hui 5-0.

AlphaGo-Lee

Distributed: 48 first generation TPU. AlphaGo-Lee vs Lee Sedol 4-1.

AlphaGo-Master, January-May 2017

Single machine: 4 second generation TPU. AlphaGo-Master vs Various professional players 60-0, AlphaGo-Master vs Ke Jie 3-0.

AlphaGo

May 2017+5 months: AlphaGo Zero is born

AlphaGo Zero

AlphaGo = *value NN* + *policy NN* + MCTS

1. The policy NN **p** is trained with SL $\rightarrow$ **p** human level.
2. The policy NN **p** is trained with RL $\rightarrow$ **p** superhuman level.
3. The superhuman policy NN is udes to train the value NN $v \rightarrow$ v superhuman level.
4. End of training.
5. To play, AlphaGo uses MCTS, reducing breadth and depth of the game tree by **p** e v.

AlphaGo Zero, idea

MCTS = a way of improving the policy. If I use MCTS in step (2), at the I obtain an improved policy $\mathbf{p}^*$.

AlphaGo Zero

For mathematicians

MCTS is an operator $\boldsymbol{\pi}$ acting on policies $\mathbf{p}$, and AlphaGo Zero uses RL to approximate a fixed point $\mathbf{p}^*$:

$$\boldsymbol{\pi}(\mathbf{p}^*) = \mathbf{p}^*$$

that is, a policy that can't be improved by using MCTS.

AlphaGo Zero, details

- One NN f_θ.
- Input layer: s represents the board last 8 positions, and one additional layer for who is to play.
- Output layer: probability distribution $P(s, a)$ for every move $a \leftarrow s$ and winning probability $v(s)$.

$$f_\theta(s) = (\mathbf{p}(s), v(s)) = (P(s, a_\bullet), v(s)) \in \mathbb{R}^{362} \times \mathbb{R}$$

AlphaGo Zero: what does he know about the game?

AlphaGo Zero: *domain knowledge*

AlphaGo Zero is born to prove that learning is possible *without any human intervention*. He knows only:

1. Rules. When a game ends *is not part of the rules*! For AlphaGo Zero, a game ends when both players pass, or after $2 \cdot 19^2 = 722$ moves.
2. Tromp-Taylor scoring.
3. The state s is an image 19×19.
4. The isometry invariance is used for data augmenting and during MCTS.
5. Black-White invariance is used in the state representation: s is the board for the current player.

AlphaGo Zero, more details

MCTS with PUCT

- During the policy-improving phase 2, MCTS chooses from s the position a that maximizes

$$\frac{W(s,a)}{N(s,a)} + c \cdot P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

- Like before: trade-off between exploration $c >> 0$ and exploitation $c \simeq 0$.
- Different from before: exploration goes towards moves that are considered good from the policy $\mathbf{p}$.
- Like before, but different: when a leaf s_L is reached, playouts use the policy NN.
- Like before: W e N are backed up, and a new iteration starts.

AlphaGo Zero, more and more details

Learning: towards superhuman P and v

- The best NN plays against himself 25K games.
- Where does he moves starting from s? 1600 iterations of MCTS, then samples the move from

$$\pi(a|s) = N(s,a)^{1/\tau} / \sum_b N(s,b)^{1/\tau}$$

AlphaGo Zero, more and more details

Learning: towards superhuman P and v

- The best NN plays against himself 25K games.
- Where does he moves starting from s? 1600 iterations of MCTS, then samples the move from

$$\pi(a|s) \sim N(s,a)^{1/\tau}$$

AlphaGo Zero, more and more details

Learning: towards superhuman P and v

- The best NN plays against himself 25K games.
- Where does he moves starting from s? 1600 iterations of MCTS, then samples the move from
$$\pi(a|s) \sim N(s,a)^{1/\tau}$$
- For every state s and move a: store the improved policy $\pi(a|s)$ and the winner $z(s)$.
- Parallel computation. 2048 states s from the last 500K games. Train $f_\theta(s) = (\mathbf{p}(s), v(s))$ such that $P(s,a) \to \pi(a|s)$ and $v(s) \to z(s)$. Repeat 1000 times ($\sim$ 4 positions per game).
- The error function consider both errors:
$$l = (z-v)^2 - \boldsymbol{\pi}^T \cdot \ln(\mathbf{p}) + c|\theta|_2^2$$
- Parallel computation. Every 1000 training iterations, the new NN plays 400 games against the strongest NN. If he wins, the new NN is promoted to the strongest.

ALPHAGO ZERO CHEAT SHEET

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel

SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored

The game state
(see 'What is a Game State section')

π

The search probabilities
(from the MCTS)

The winner
(+1 if this player won, -1 if this player lost - added once the game has finished)

RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions
– The game states are the input (see 'Deep Neural Network Architecture')

Loss Function
Compares predictions from the neural network with the search probabilities and actual winner

PREDICTIONS **p** Cross-entropy + **v** Mean-squared error + Regularisation π ACTUAL

After every 1,000 training loops, evaluate the network

EVALUATE NETWORK

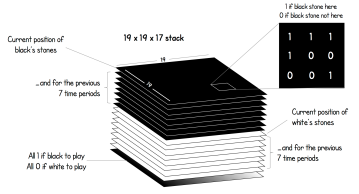
Test to see if the new network is stronger

Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player

WHAT IS A 'GAME STATE'



This stack is the input to the deep neural network

THE DEEP NEURAL NETWORK ARCHITECTURE

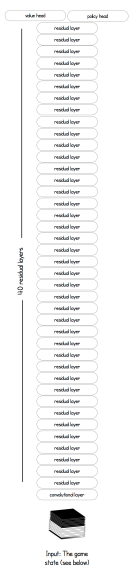
How AlphaGo Zero assesses new positions

The network learns 'tabula rasa' (from a blank slate)

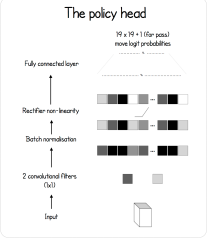
At no point is the network trained using human knowledge or expert moves

The value head

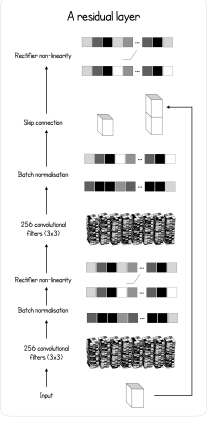
The network



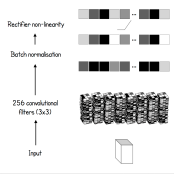
The policy head



A residual layer

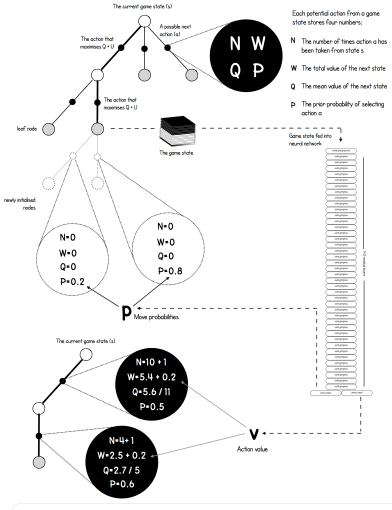


A convolutional layer



MONTE CARLO TREE SEARCH (MCTS)

How AlphaGo Zero chooses its next move



Each potential action from a game state stores four numbers:

N The number of times action a has been taken from state s

W The total value of the next state

Q The mean value of the next state

P The prior probability of selecting action a

First, run the following simulation 1,600 times...

Start at the root node of the tree (the current game state)

1. Choose the action that maximises...

$$Q + U$$

The mean value of the next state

A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later, Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

p Move probabilities

v Value of the state (for the current player)

The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$N \rightarrow N + 1$

$W \rightarrow W + v$

$Q = W / N$

...then select a move

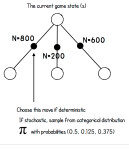
After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$$\pi \sim N^{\frac{1}{\tau}}$$

where τ is a temperature parameter controlling exploration



Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

AlphaZero: conquering the world

5 December 2017, arXiv paper on AlphaZero

AlphaGo Zero version playing Go, Chess and Shogi. 2 hours of training: stronger than Elmo (Shogi); 4 hours of training: stronger than Stockfish (Chess); 8 hours of training: stronger than AlphaGo Zero!

Stockfish

- AlphaZero vs Stockfish, 100 games, 1' per move: 25 wins of AlphaZero white, 3 wins of AlphaZero black (!!!), 72 draws.
- AlphaZero: 9 hours of training, 4 TPU, single machine.
- Stockfish: 64 search threads and 1 GB hash.

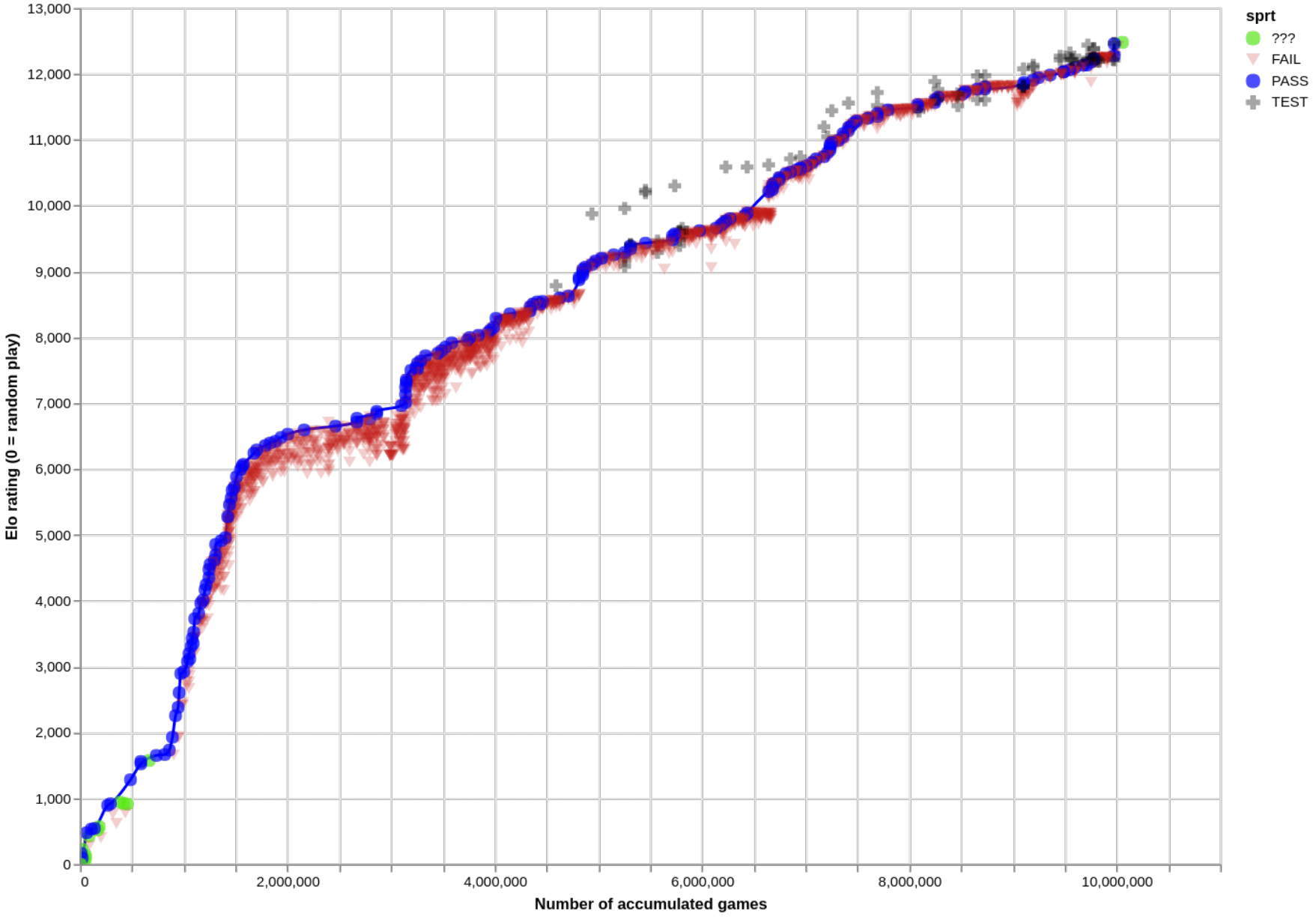
MCTS more accurate

AlphaZero analyzes 80K positions per second in Chess, Stockfish 70 millions. The NN selects the more promising variations, without wasting time analyzing other positions. This is a very human approach!

Leela Zero: a distributed, open source version of AlphaGo Zero

- Open source version of AlphaGo Zero? Problem: on a very powerful computer, AlphaGo Zero training requires 1700 years (yes, one-thousand-seven-hundred years :-)!
- Solution: distributed software.
- Architecture: player, client, server.
- GitHub. Leela Zero, author Giancarlo Pascutto. Well-known for his previous work on Chess and Go software.
- 15 March 2018: Leela Zero wins against Hajin Lee, Korean professional player, after having played "only" 5.6 millions training games.
- 600 persons contribute with 70K games per day.

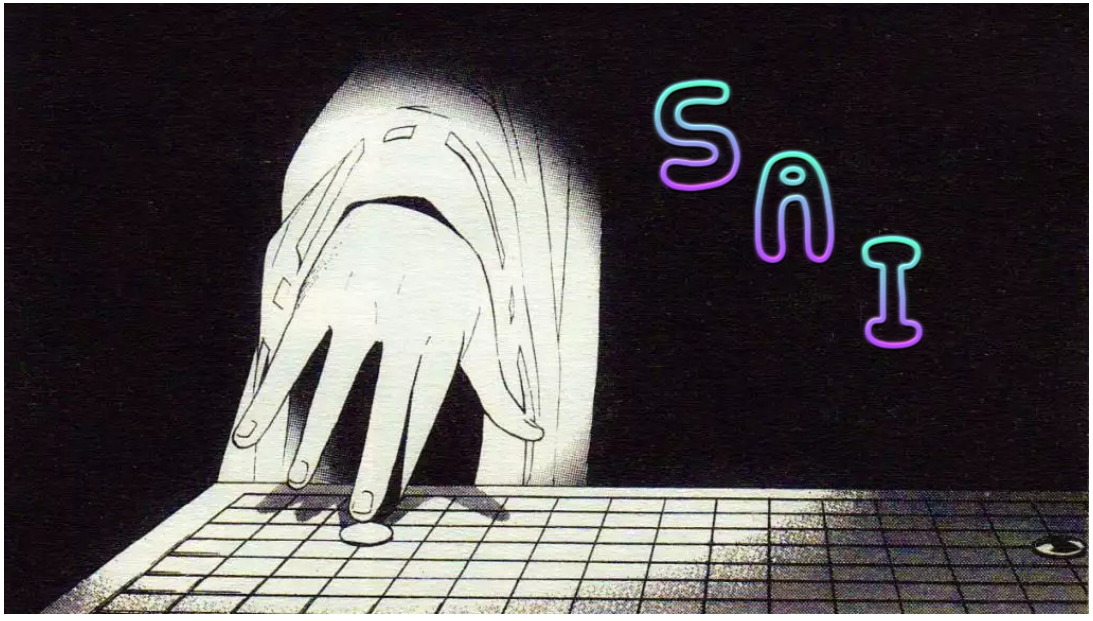
Leela Zero, strength progression as of 5 September 2018



SAI: Sensible Artificial Intelligence

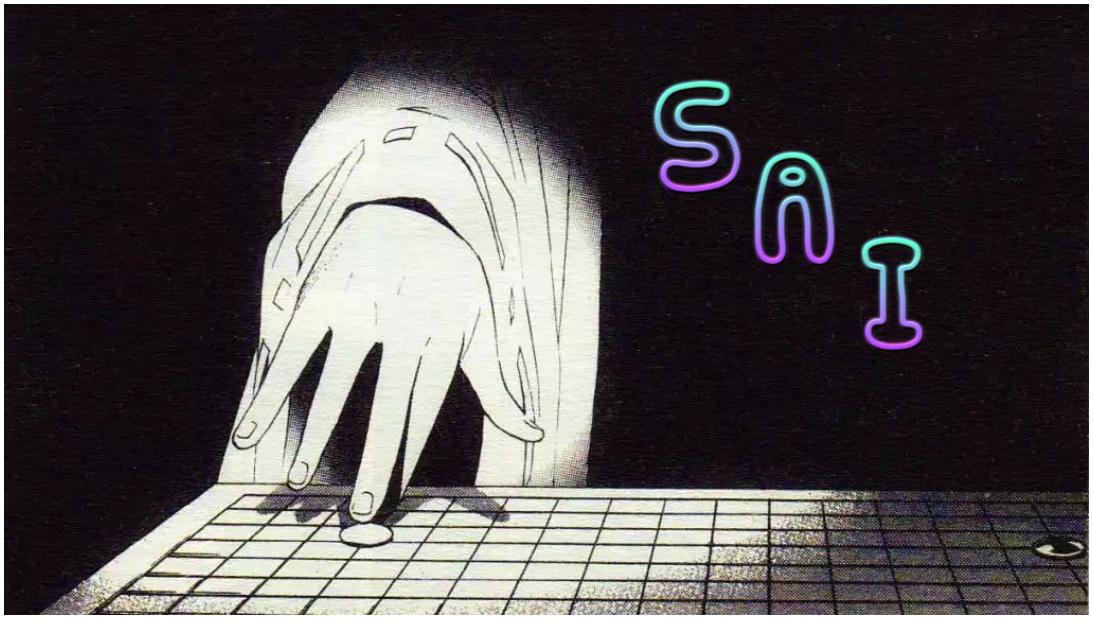
- Alpha[Go][ ][Zero] and Leela Zero: exceptional, but not perfect.
- Problem: in the final part of the game, when they could win by a big amount, the moves winning by only a small amount of points are selected more often in the UCT tree search.
- Probably due to the higher winning probability given to these moves by the value network.
- Implication: when they are winning, they play suboptimal moves.
- Several reasons for willing to know the best move also in the case when it is not required for winning (learn how to play for humans, finding the perfect game, and so on).

SAI: Sensible Artificial Intelligence



1. *Vectorial* value function: different winning probabilities according to different handicaps.
2. Policy: using 1), can choose a compromise between winning probability and final score.
3. Self-play and training of NN of different strength: both stones and score handicap can be given in a game.

SAI: Sensible Artificial Intelligence



Who are the authors of this beautiful software?

Five Italians: Gianluca Amato, Rosa Gini, Carlo Metta, Francesco Morandin, Maurizio Parton.

Where can I find SAI?

SAI is still in development. Will be presented at a conference in China, at the end of October 2018. Everybody is welcome!

Exercises

Exercise

Prove or disprove the following conjecture: every continuous function $f(a, b, c)$ is a superposition of finitely many functions (no continuity required) of at most 2 variables.

Exercise

Consider the map $\phi : I^n \to \mathbb{R}^{2n+1}$ given by

$$\phi(x_1, \ldots, x_n) = (\sum_{j=1}^{n} \lambda_j \psi_1(x_j), \ldots, \sum_{j=1}^{n} \lambda_j \psi_{2n+1}(x_j))$$

Prove that $\phi : I^n \to \phi(I^n)$ is a homeomorphism.

Exercise

Write a neural network approximating e^x in a neighbourhood of $x = 3$. Give an estimate of the number of nodes depending on the approximation required.

Exercises

Exercise

Give an example of a NN with multiplication used instead of addition as integration function. Explain what is the meaning of "locality of BP" and show that multiplication as integration function doesn't preserve locality of BP.

Exercise *

Prove that addition is the only integration function preserving locality of BP.

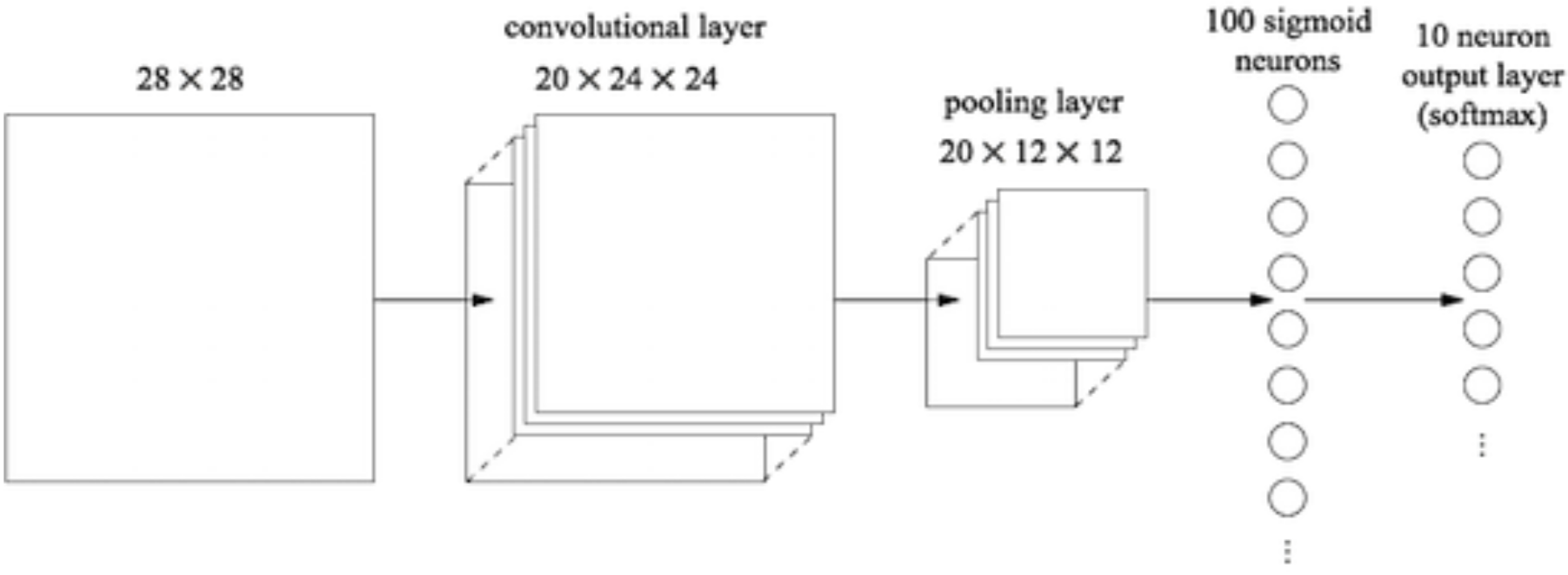
Exercise

Compute the number of parameters in a convolutional layer filtering a 28×28 image with 20 kernels of size 5×5 and stride 1, and compare with the fully connected case.

Exercises

Exercise

Compute how many parameters we have in this CNN.



Exercises

Let s be the softmax activation function.

$$s(x_1, \ldots, x_K) = (e^{x_1}, \ldots, e^{x_K}) / \sum_{i=1}^{K} e^{x_i}$$

Exercise

Show that s outputs a probability distribution. Show that increasing x_j will increase $s_j(x_1, \ldots, x_K)$ and decrease all other components.

Exercise

Compute $\lim_{c \to +\infty} e^{cx_j} / \sum_{i=1}^{K} e^{cx_i}$, and give a reason why softmax is called this way.

Exercises

Exercise *

Learn the basics of TensorFlow and use it to program your first NN.

Exercise **

Follow this Google crash course:

`https://developers.google.com/machine-learning/crash-course/`

and make any of its exercises.

Bibliography, before 2006

- Perceptrons: An Introduction to Computational Geometry, `https://mitpress.mit.edu/books/perceptrons`. Minsky-Papert, 1969. The book that started the first AI winter. Read it if you like to have solid foundations.
- Introduction to Machine Learning, `https://ai.stanford.edu/~nilsson/mlbook.html`. Nilsson, 1998. Read it if you like to frame neural networks in the machine learning environment.
- Neural Networks - A Systematic Introduction, `https://page.mi.fu-berlin.de/rojas/neural/`. Rojas, 1996. Read it.

Bibliography, after 2006

- Google crash course, `https://developers.google.com/machine-learning/crash-course/`. Online machine learning crash course by Google. Do it.
- Neural Networks and Deep Learning, `https://www.coursera.org/learn/neural-networks-deep-learning`. Ng-Katanforoosh-Mourri. Online, not free. Deep learning course, 1 out of 5. Do it if you expect to work in deep learning.
- Neural Networks and Deep Learning, `http://neuralnetworksanddeeplearning.com/`. Nielsen, 2017. The best online book ever. Read it carefully. Absolutely.
- Deep Learning, `http://neuralnetworksanddeeplearning.com/`. Goodfellow-Bengio-Courville, 2016. The most authoritative source on deep learning. Read it. Not easy.


That's all Folks!

Licenza