When neural networks meet error correcting codes: new perspectives for resilient associative memories

Vincent Gripon
Joint work with Claude Berrou

Télécom Bretagne

January 22, 2013

Motivation

Associative memory

- Principle: retrieve a previously stored message given part of its content,
- Used in CPU caches, databases, intrusion detection systems...

Framework

Error correcting codes

Robustness

Associative memories

Architecture

Neural network

Motivation

Associative memory

- Principle: retrieve a previously stored message given part of its content,
- Used in CPU caches, databases, intrusion detection systems...

Framework

Error correcting codes

Robustness

Associative memories

Architecture

Neural network

Motivation

Associative memory

- Principle: retrieve a previously stored message given part of its content,
- Used in CPU caches, databases, intrusion detection systems...

Framework

Error correcting codes

Robustness

Associative memories

Architecture

Neural network

Motivation

Associative memory

- Principle: retrieve a previously stored message given part of its content,
- Used in CPU caches, databases, intrusion detection systems...

Framework

Error correcting codes

Robustness

Associative memories

Architecture

Neural network

Motivation

Associative memory

- Principle: retrieve a previously stored message given part of its content,
- Used in CPU caches, databases, intrusion detection systems...

Framework

Error correcting codes

Robustness

Associative memories

Architecture

Neural network

Existing techniques

Parameters

Diversity (# of messages), memory efficiency, speed convergence, error rate...

Electronics

CAMs

Pros:

- Speed,
- Error rate.

Cons:

- Power consumption,
- Flexibility.

Neural networks

Hopfield Networks

Pros:

- Flexibility
- Redundancy.

Cons:

- Error rate,
- Diversity and memory efficiency.

Inspiration from neural network

Architecture

- Partition into macrocolumns, microcolumns,
- Population coding,
- Neural clique (set of neurons fully interconnected).

Laws

- Hebb's rule (creation of connections between simultaneously activated neurons),
- Aggregation of inputs in neurons (McCulloch and Pitts' model),
- Local winner-take-all.

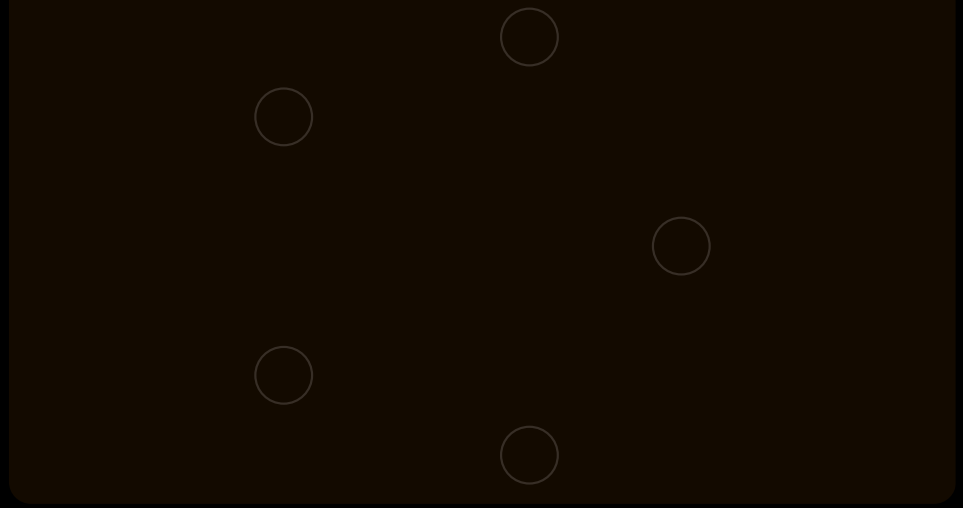
Bio-inspired error correcting codes

Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code

Clique code



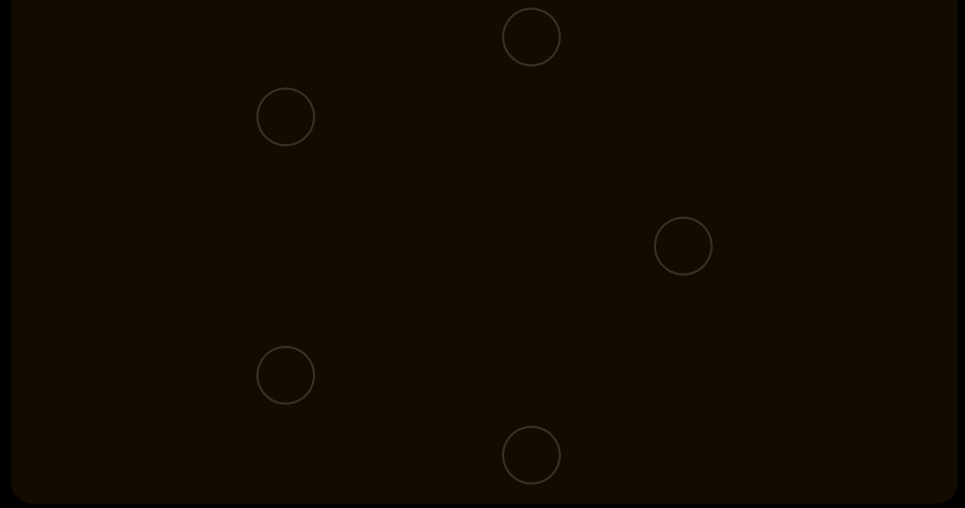
Bio-inspired error correcting codes

Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code

Clique code

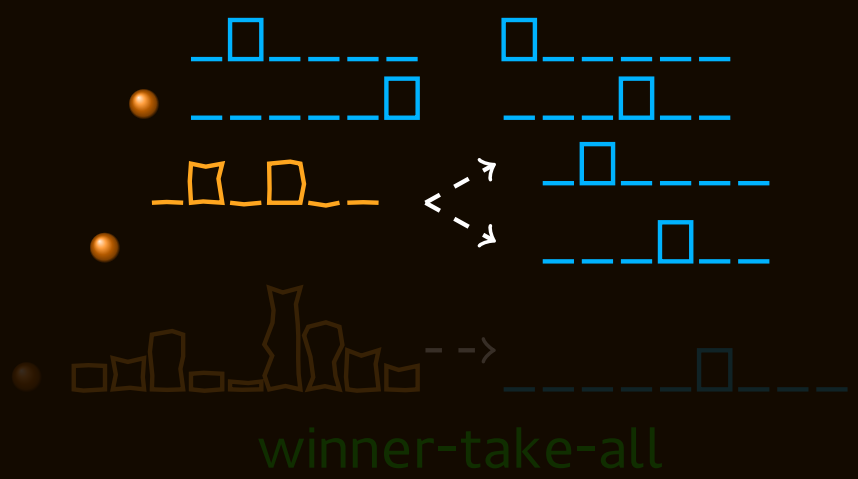


Bio-inspired error correcting codes

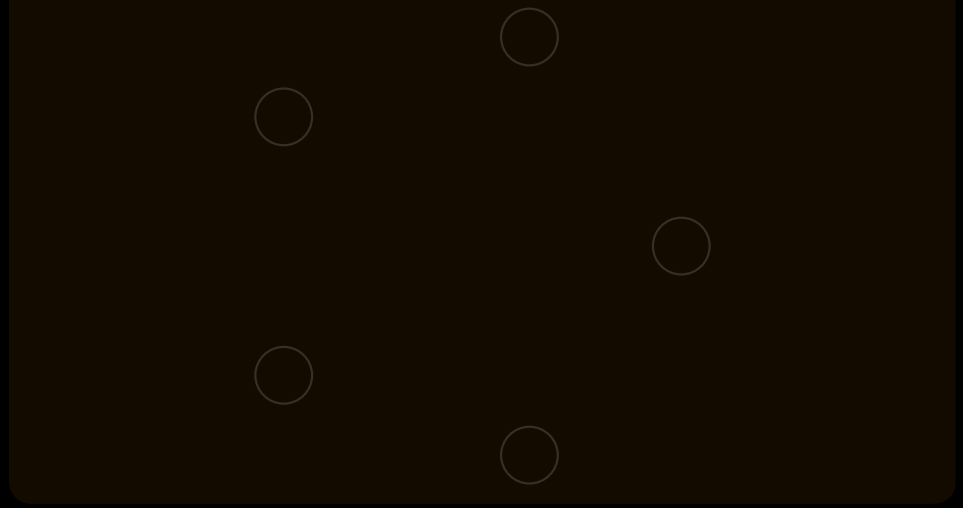
Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code



Clique code

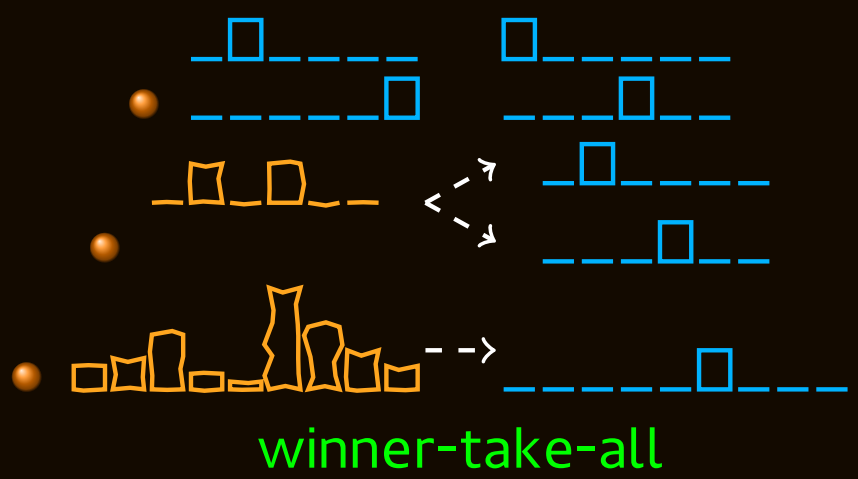


Bio-inspired error correcting codes

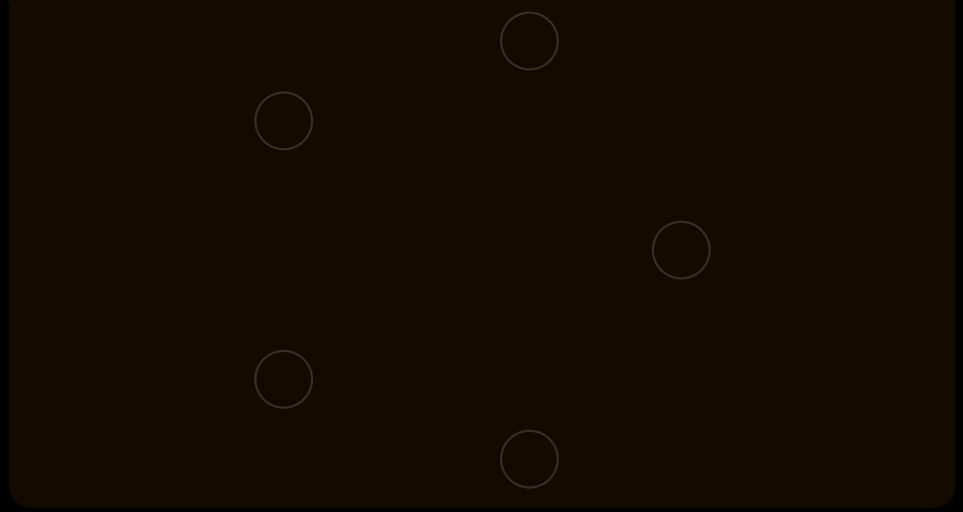
Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code



Clique code

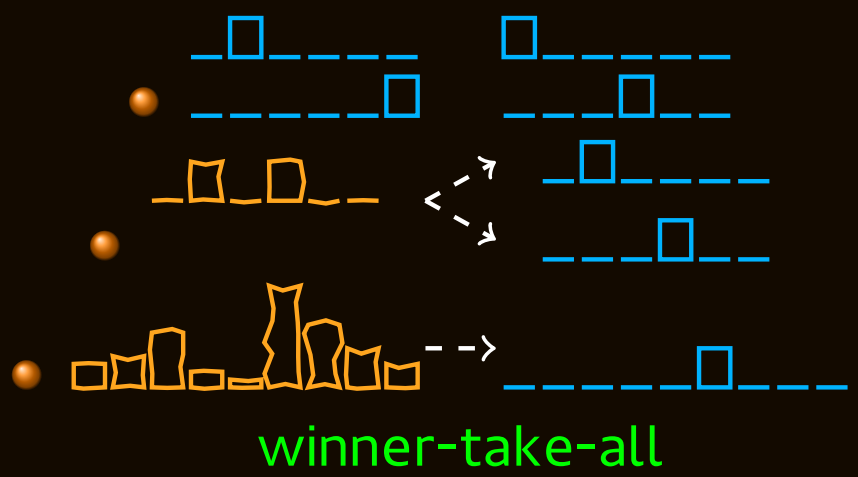


Bio-inspired error correcting codes

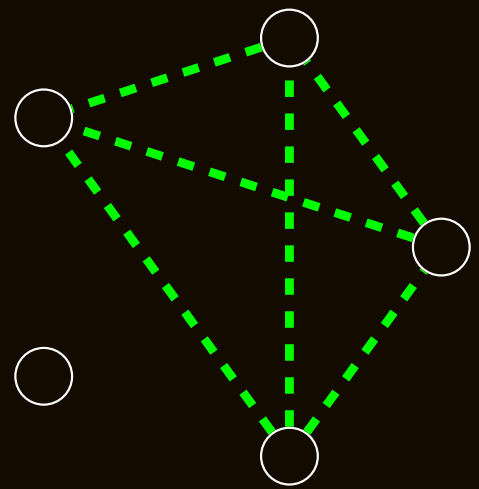
Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code



Clique code

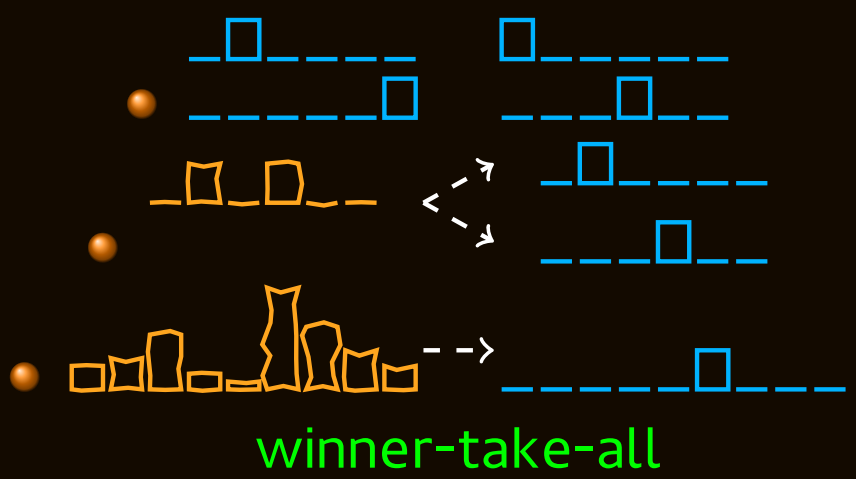


Bio-inspired error correcting codes

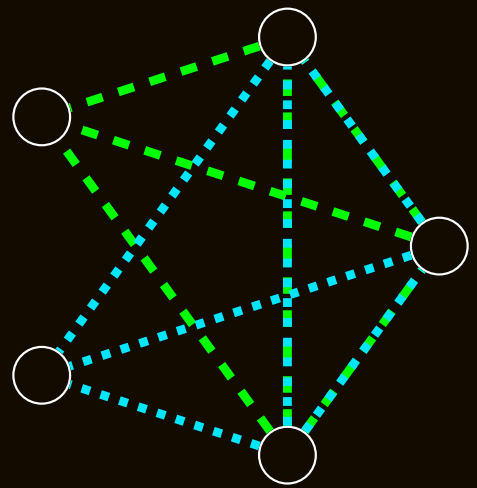
Definition

Encoding of pieces of information into distant representations in order to protect them against noise.

Thrifty code

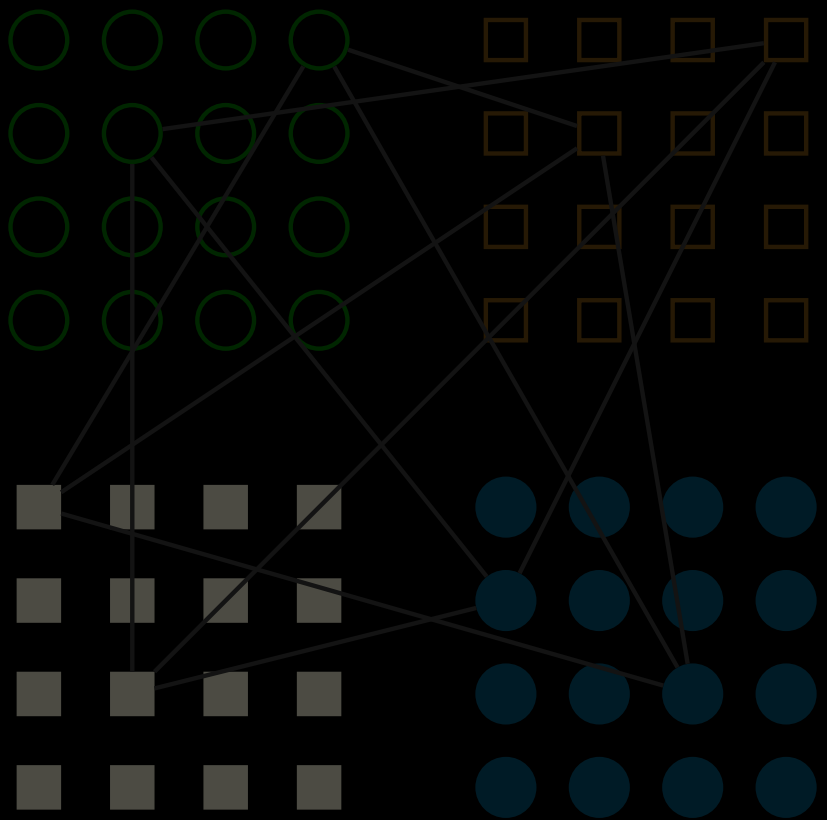


Clique code



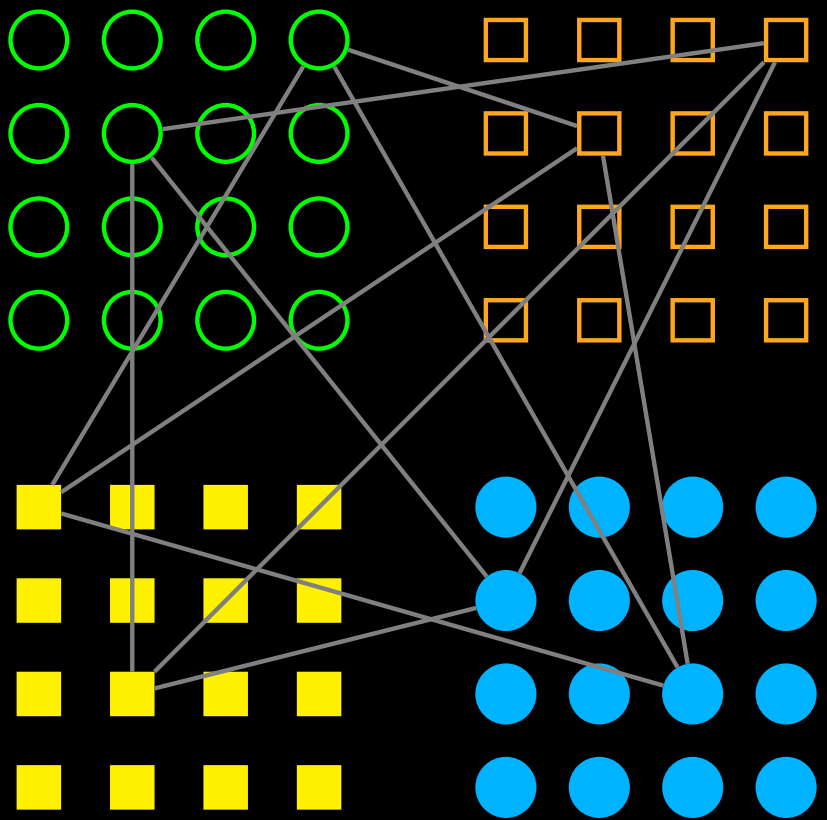
How to store?

- Message to store: 1000001100101001
- Let us use a network of $c = 4$ clusters of $l = 16$ neurons each,
- $\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4}$,



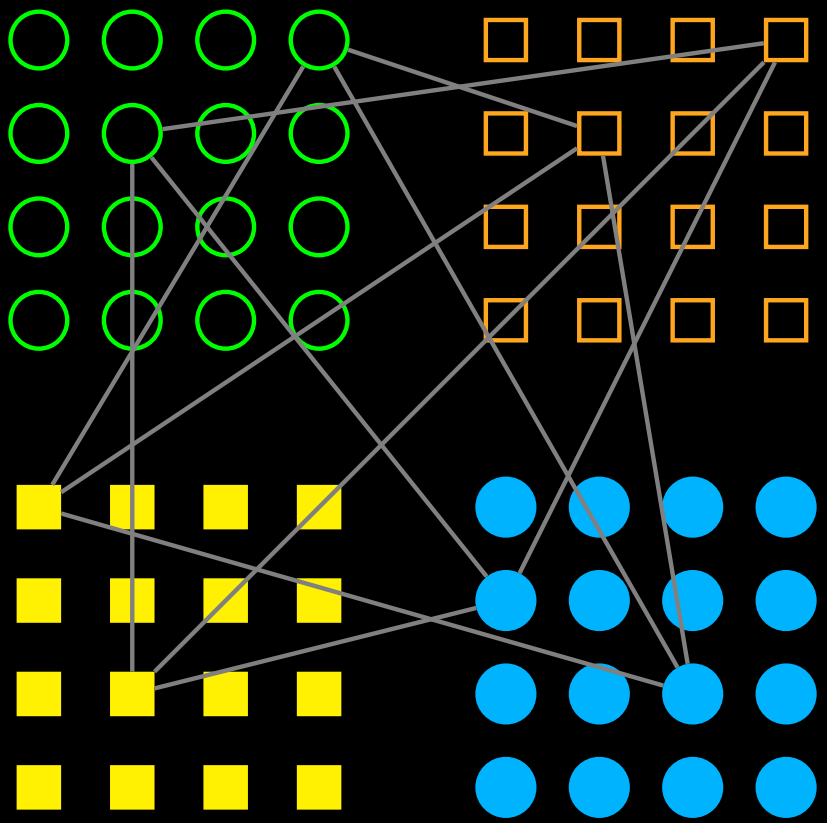
How to store?

- Message to store: 1000001100101001
- Let us use a network of $c = 4$ clusters of $l = 16$ neurons each,
- $\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4}$,



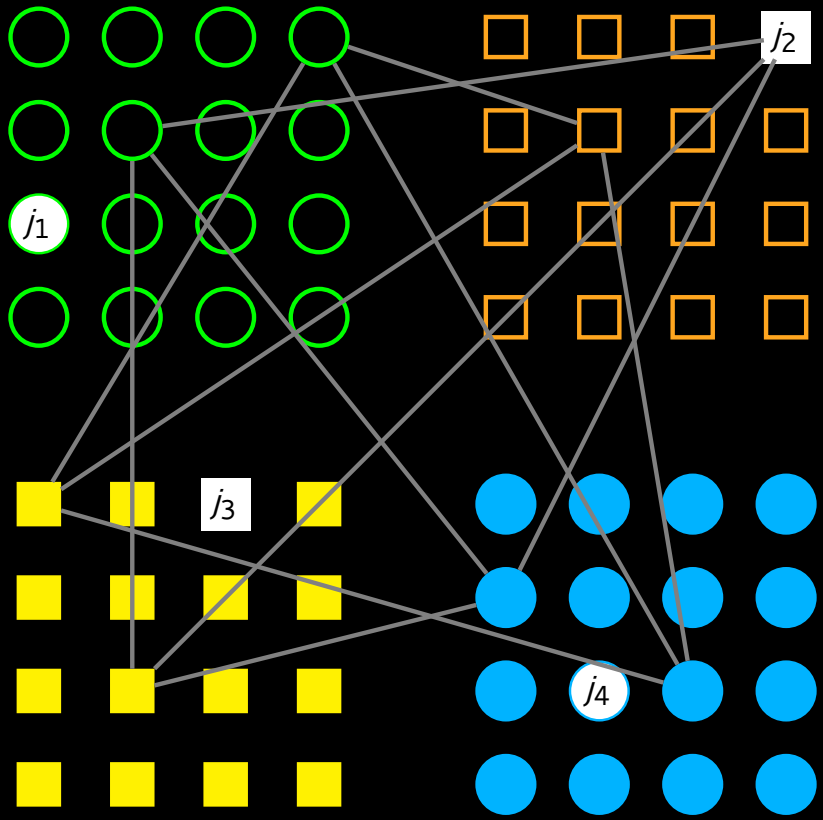
How to store?

- Message to store: 1000001100101001
- Let us use a network of $c = 4$ clusters of $l = 16$ neurons each,
- $\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4}$,



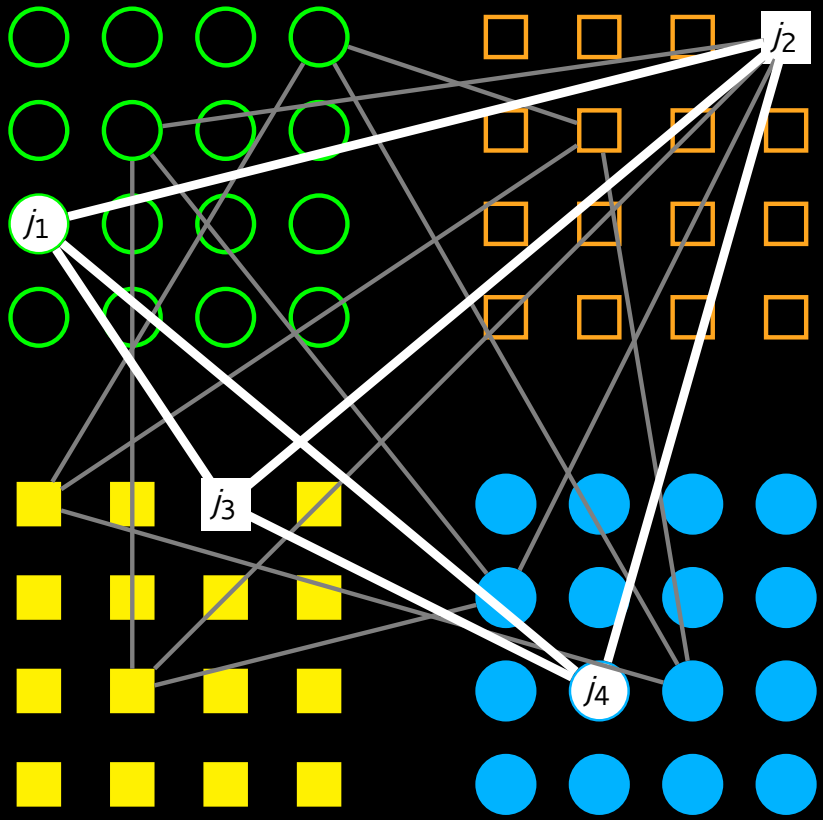
How to store?

- Message to store: 1000001100101001
- Let us use a network of $c = 4$ clusters of $l = 16$ neurons each,
- $\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4}$, (Thrifty code



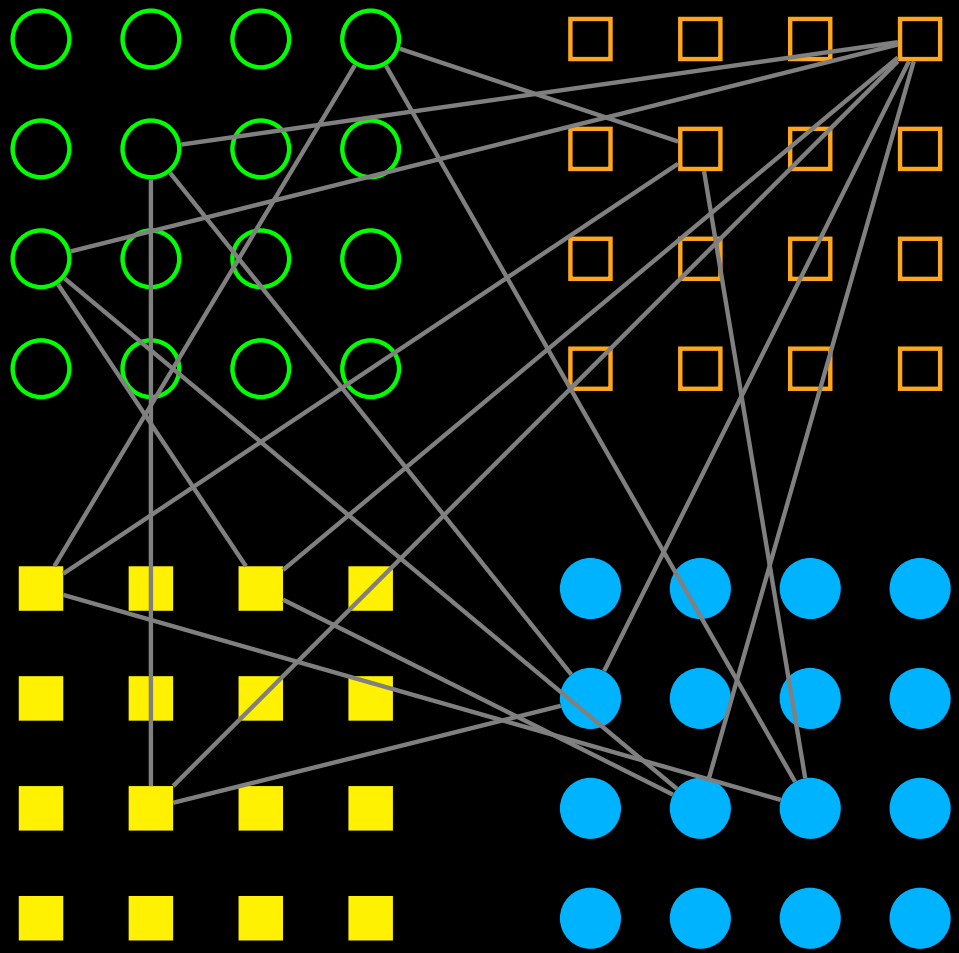
How to store?

- Message to store: 1000001100101001
- Let us use a network of $c = 4$ clusters of $l = 16$ neurons each,
- $\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4}$, (Thrifty code then clique code)



How to retrieve?

$\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}$????,

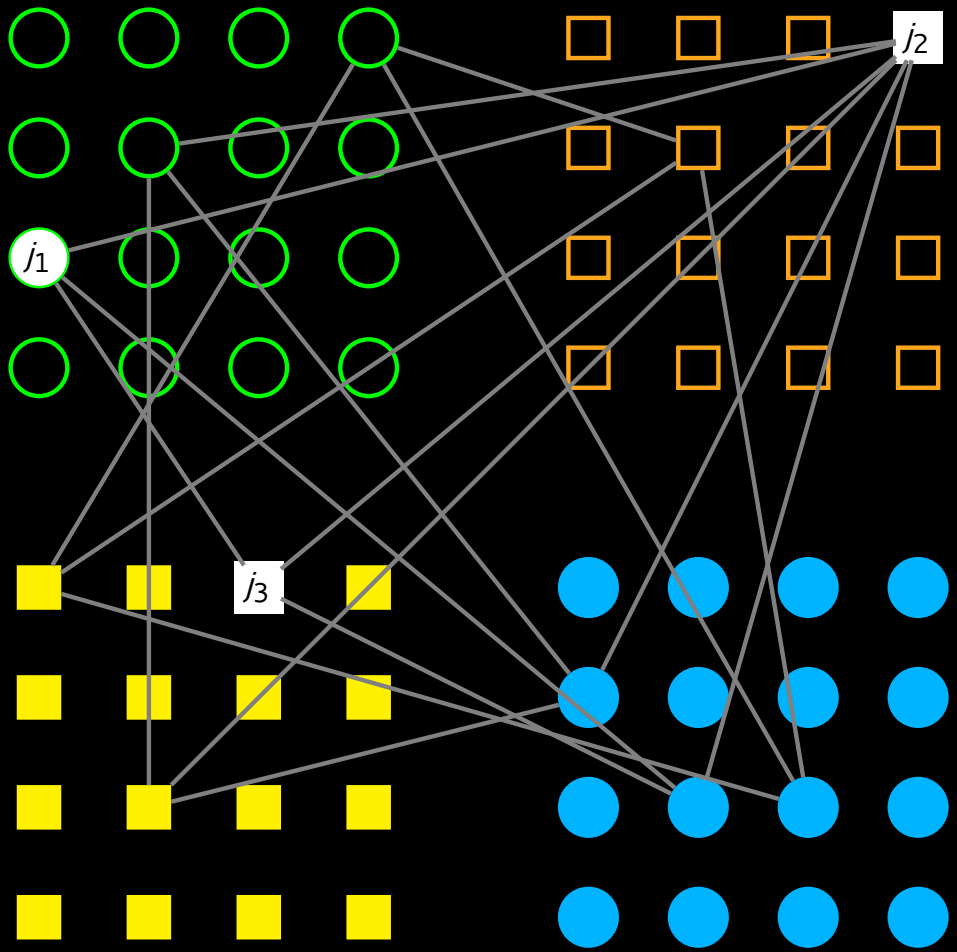


- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

How to retrieve?

1000 0011 0010 ????,

j_1 in c_1 j_2 in c_2 j_3 in c_3

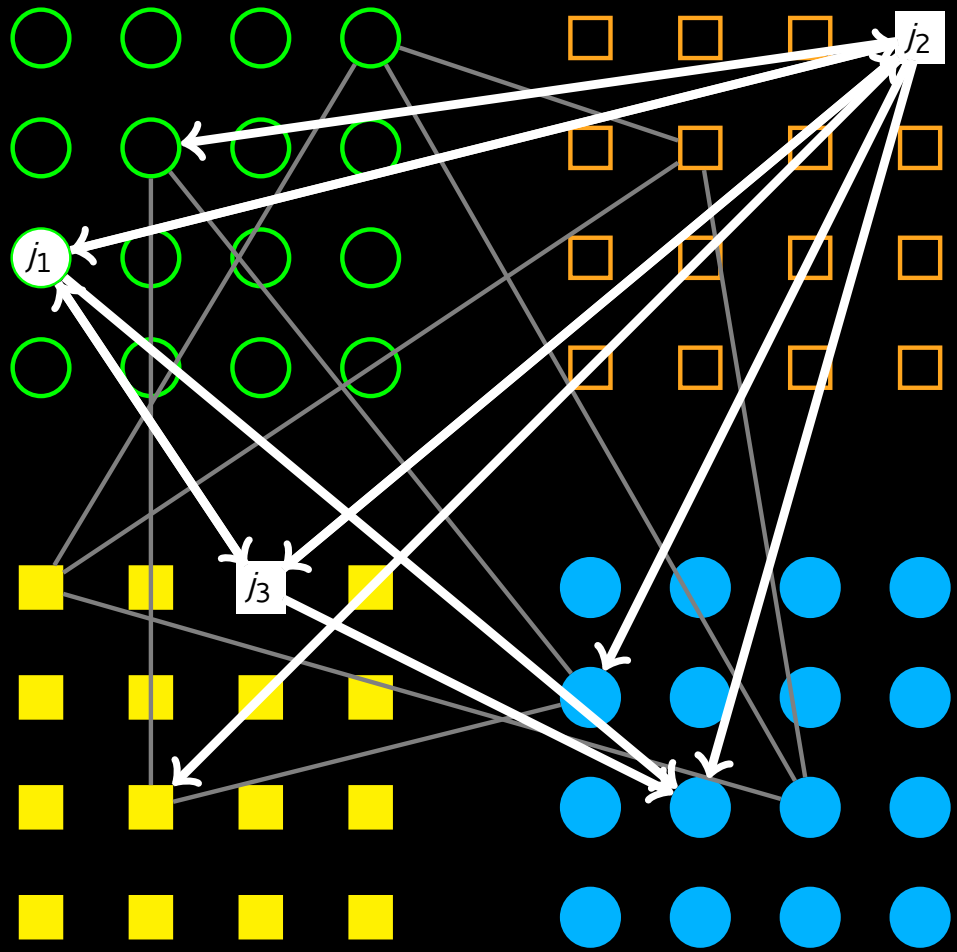


- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

How to retrieve?

1000 0011 0010 ????,

j_1 in c_1 j_2 in c_2 j_3 in c_3

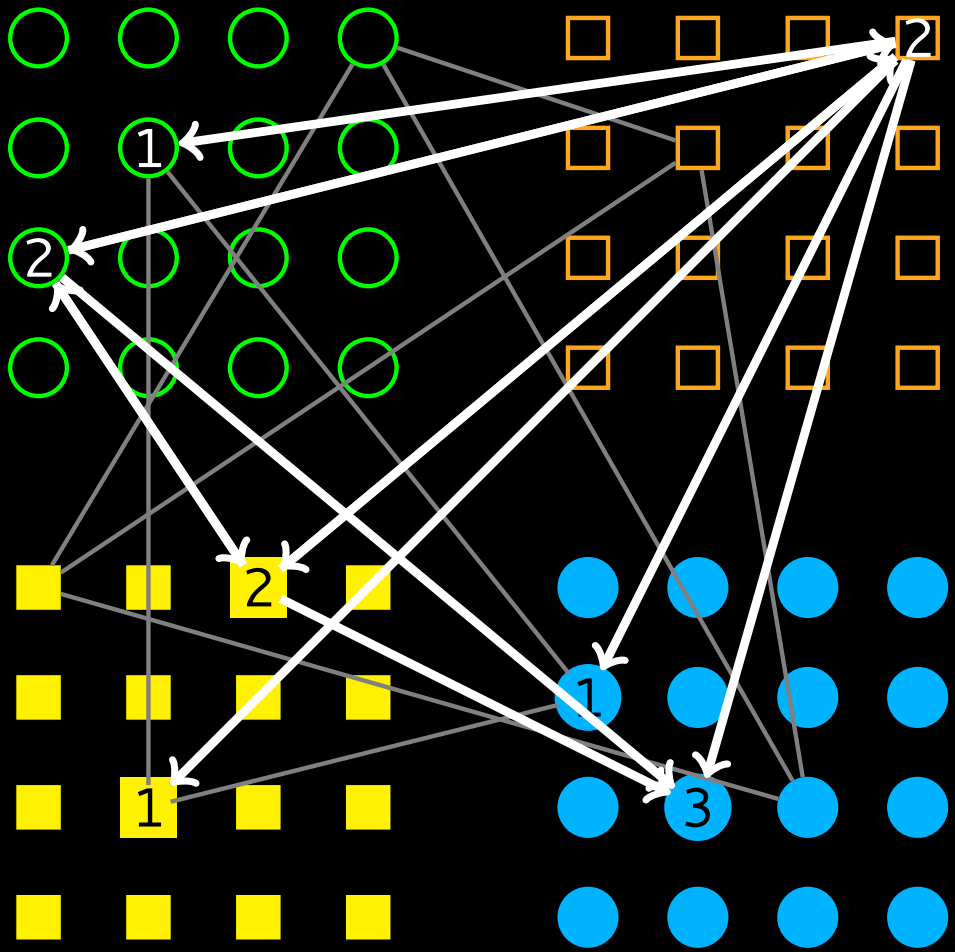


- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

How to retrieve?

1000 0011 0010 ????,

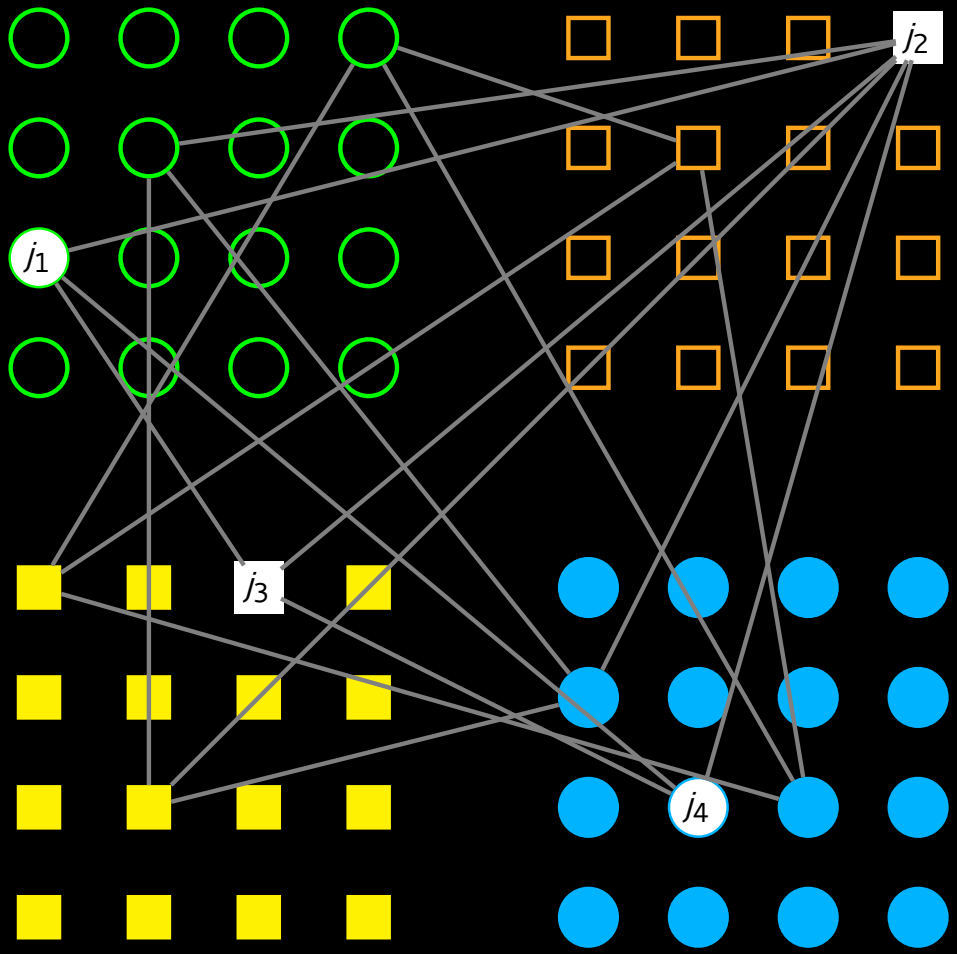
j_1 in c_1 j_2 in c_2 j_3 in c_3



- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

How to retrieve?

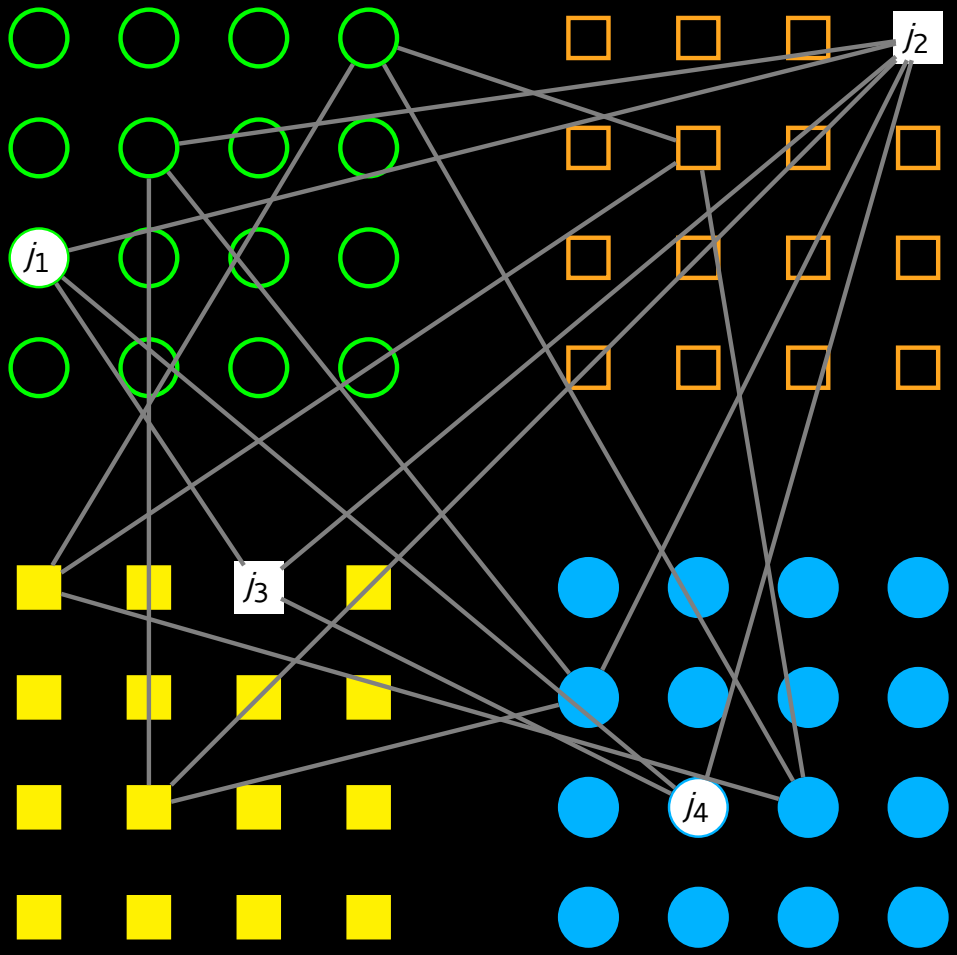
$$\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4},$$



- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

How to retrieve?

$$\underbrace{1000}_{j_1 \text{ in } c_1}\ \underbrace{0011}_{j_2 \text{ in } c_2}\ \underbrace{0010}_{j_3 \text{ in } c_3}\ \underbrace{1001}_{j_4 \text{ in } c_4},$$



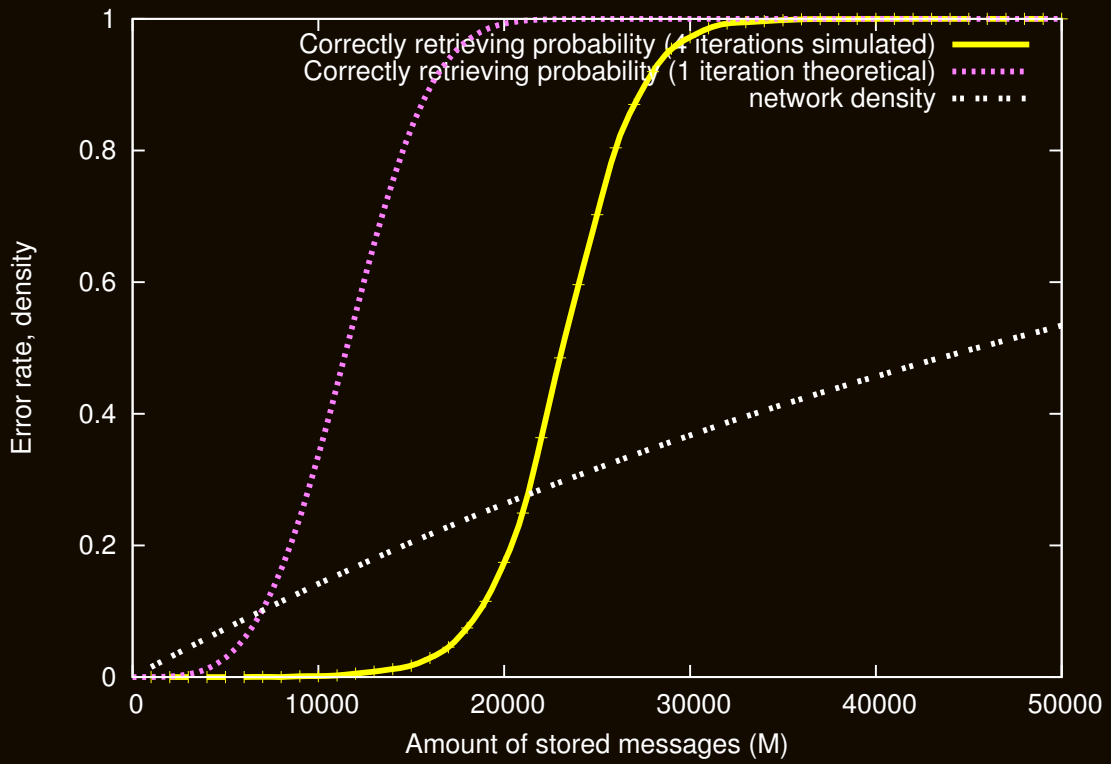
- Projection to the network,
- Global decoding: sum,
- Local decoding: winner-take-all,
- Possibly iterate the two decodings.

Performance as an associative memory

Context

Retrieve a previously stored message given half its bits.

Error probability



$c = 8$ clusters of $l = 256$ neurons each ($\sim$ messages of 64 bits).

Hopfield network equivalent: 60 messages stored.

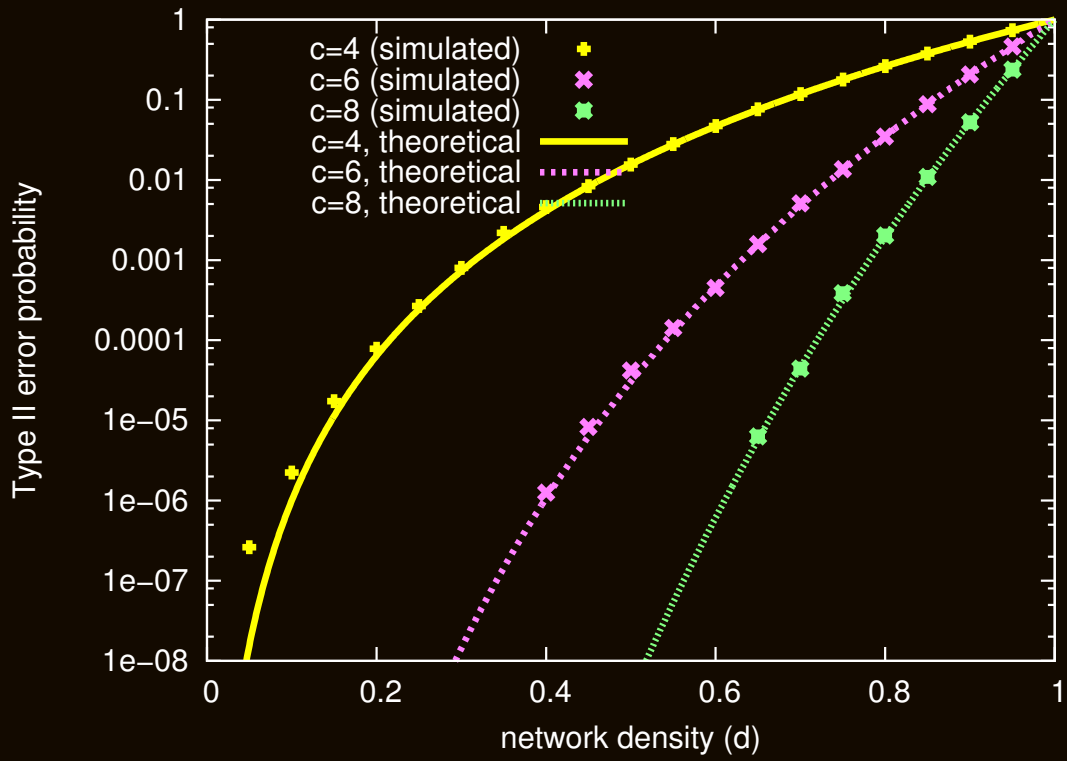
Performance as a classifier

Type I error

No Type I error: a stored message is always recognized.

Type II error

Type II error is not zero: a non stored message may be recognized.



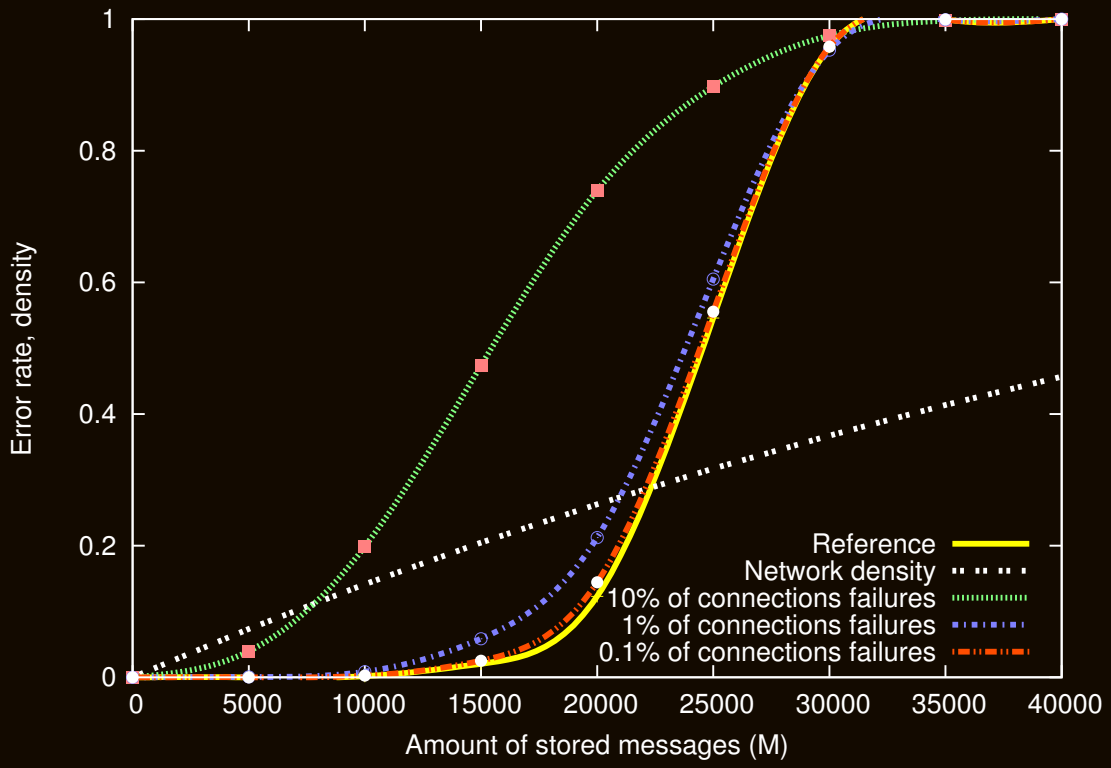
Type II error rate for various number of clusters c and for $l = 512$ neurons per cluster.

Performance against connections erasures

Context

Connections are erased at random.

Error probability



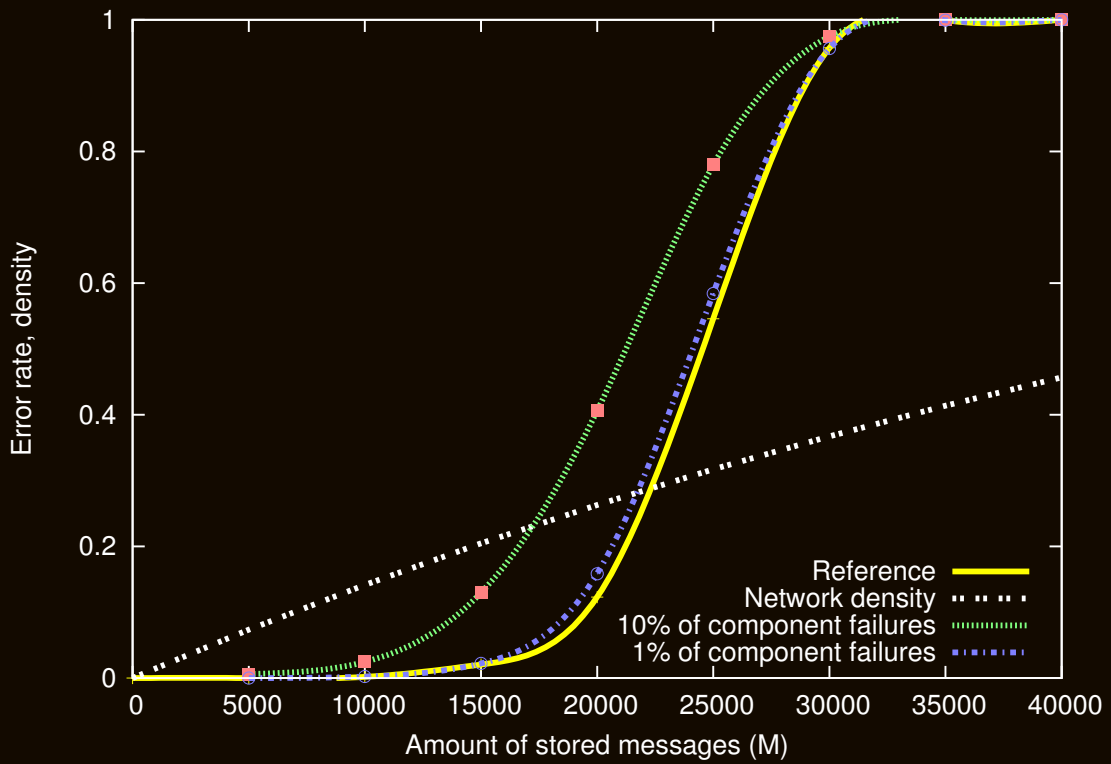
$c = 8$ clusters of $l = 256$ neurons each ($\sim$ messages of 64 bits). 4 symbols missing.

Performance against components faults

Context

Neurons occasionally miss fire.

Error probability



$c = 8$ clusters of $l = 256$ neurons each ($\sim$ messages of 64 bits). 4 symbols missing.

Conclusion

Performance

- Nearly optimal associative memories,
- Simple and concurrent functionning,
- Parameters to balance performance and complexity.

Robustness and applications

- Reliable on unreliable hardware,
- Can be directly used as a cache, a set implementation or a database engine.

Conclusion

Performance

- Nearly optimal associative memories,
- Simple and concurrent functionning,
- Parameters to balance performance and complexity.

Robustness and applications

- Reliable on unreliable hardware,
- Can be directly used as a cache, a set implementation or a database engine.

Conclusion

Performance

- Nearly optimal associative memories,
- Simple and concurrent functionning,
- Parameters to balance performance and complexity.

Robustness and applications

- Reliable on unreliable hardware,
- Can be directly used as a cache, a set implementation or a database engine.

Conclusion

Performance

- Nearly optimal associative memories,
- Simple and concurrent functionning,
- Parameters to balance performance and complexity.

Robustness and applications

- Reliable on unreliable hardware,
- Can be directly used as a cache, a set implementation or a database engine.

Conclusion

Performance

- Nearly optimal associative memories,
- Simple and concurrent functionning,
- Parameters to balance performance and complexity.

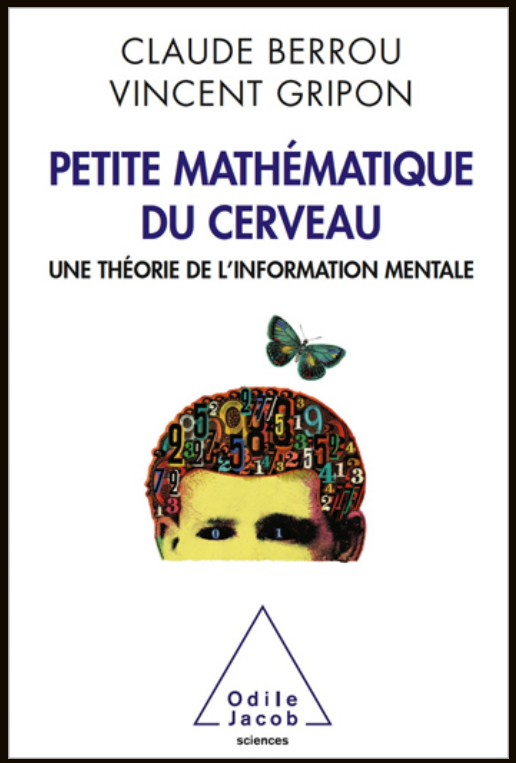
Robustness and applications

- Reliable on unreliable hardware,
- Can be directly used as a cache, a set implementation or a database engine.

Questions

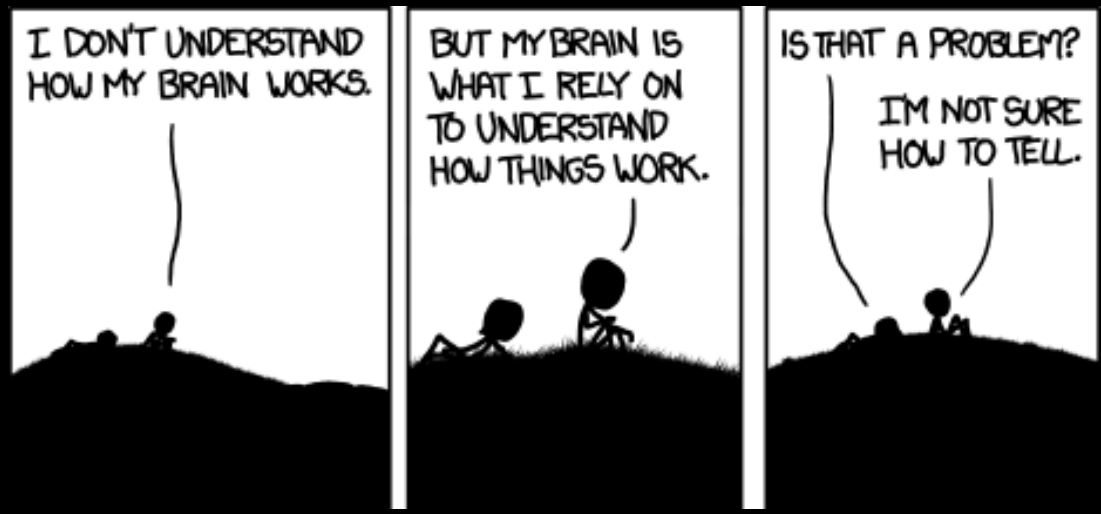
Thank you for your attention, I am at your disposal if you have any questions.

If you read French



To learn more

Visit:
http://www.vincent-gripon.com/?p1=100



XKCD of January 21st, 2013.