MARCH MEETING2017

Yale University

Neural Network Decoders for Quantum Error Correcting Codes

...

Stefan Krastanov, Liang Jiang

YQ Yale Quantum Institute

ARL

AFOSR
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

Alfred P. Sloan FOUNDATION

the David & Lucile Packard FOUNDATION

Goal:
Given a syndrome deduce the error (efficiently).

Error Correcting Codes (ECC) 101

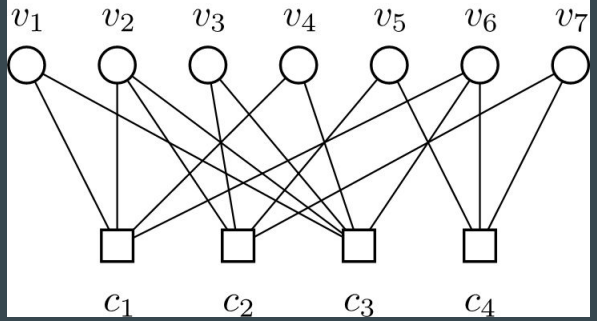
Classical Binary ECC

Use N unreliable physical bits
(susceptible to flips)
to represent K reliable logical bits
by imposing N-K constraints.

Use the syndrome (i.e. check which constraints are broken) to deduce what error happened (and reverse it).

Caveat: multiple errors can lead to the same syndrome - finding the most probable is the best you can do.

$$\mathbf{H} = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$



$$s = He$$

Error Model for a Qubit

Slightly more complicated than a bit...

X, Z, and Y errors can happen.

Y is equivalent to both X and Z happening.

We need two bits per qubit to track errors.

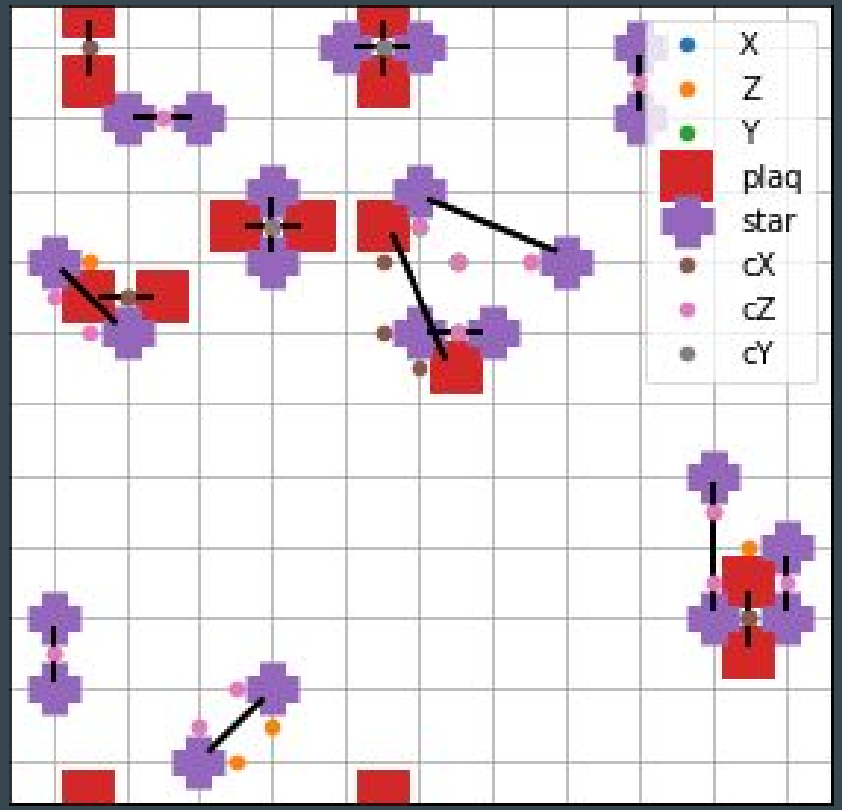
	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-iZ$	I	iX
Z	Z	iY	$-iX$	I

$$s_Z = H_Z.e_X$$
$$s_X = H_X.e_Z$$

Decoding (Correcting) the Code

- **e** to **s** is easy
- **s** to **e** is impossible (it is one to many relation)
- **s** to *most probable* **e** is difficult (infeasible / NP-hard)
- **s** to *fairly probable* **e** if there is *additional structure* can be “easy”

$$s_Z = H_Z.e_X$$
$$s_X = H_X.e_Z$$



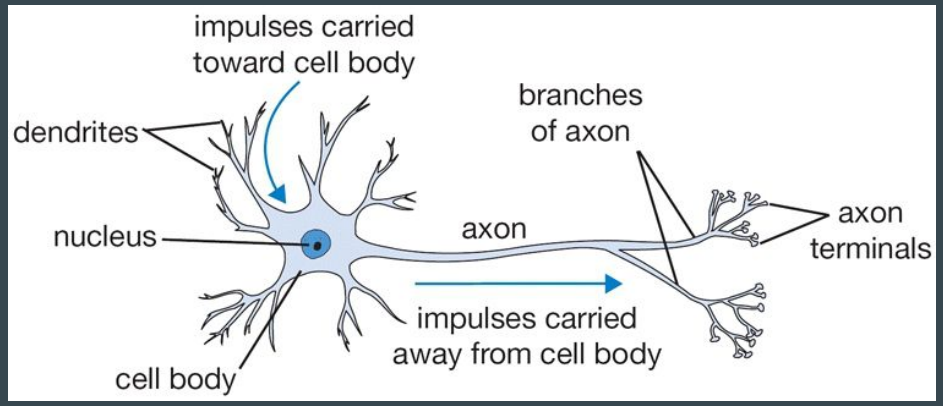
Example of Additional Structure - Toric Code

Neural Networks 101

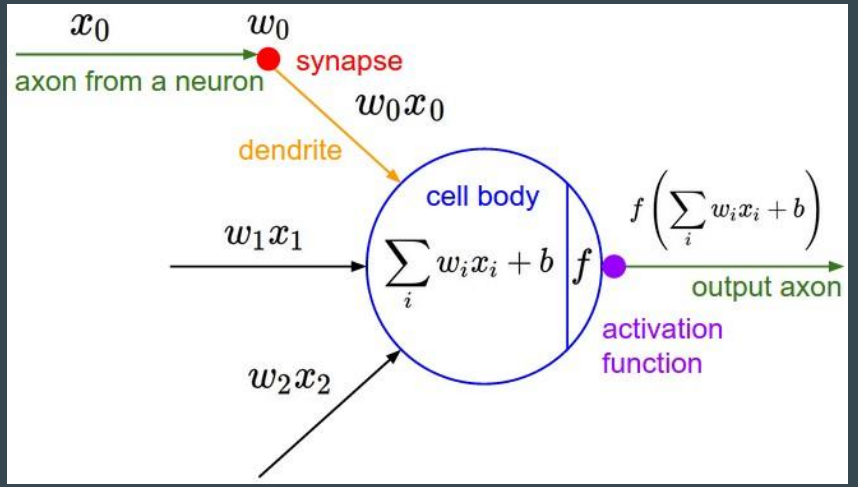
Different approach to ECC with Neural Networks suggested by:
R35.00002 : A neural decoder for topological codes - Torlai, Melko

Other interesting uses of Neural Networks for quantum many-body theory:
Solving the Quantum Many-Body Problem with Artificial Neural Networks - Carleo, Troyer (arXiv:1606.02318)

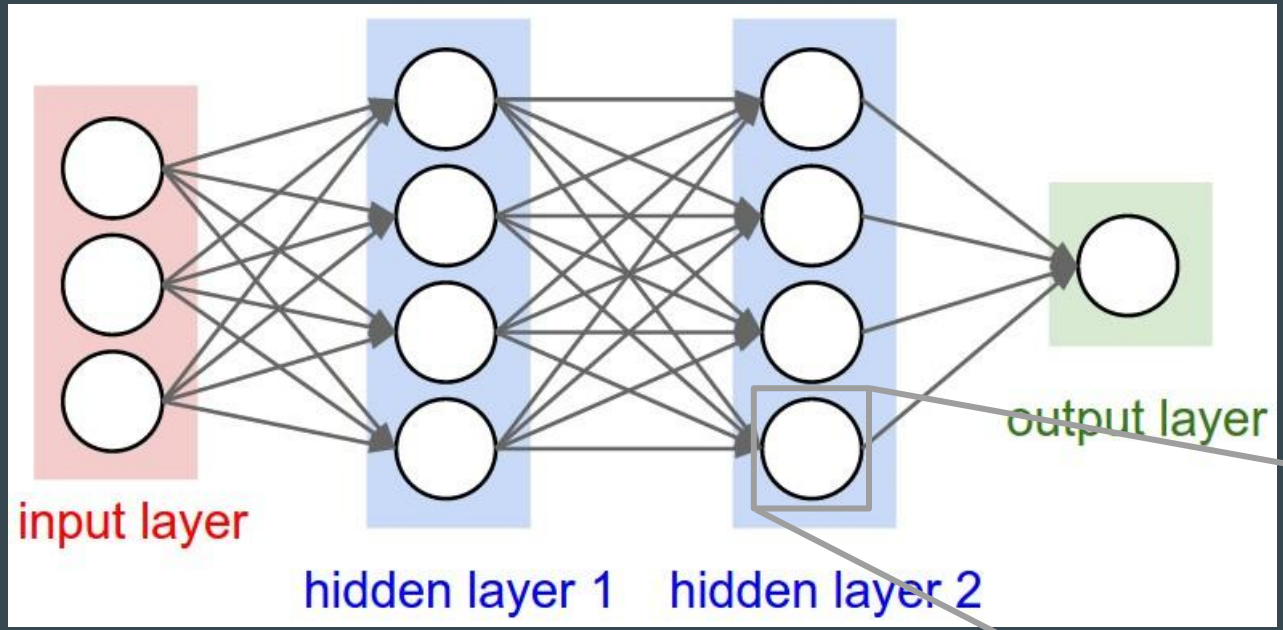
Neuron

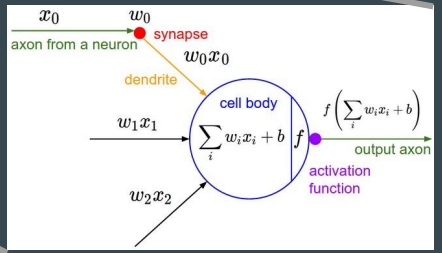


A Nicer Neuron



Layered Neural Network



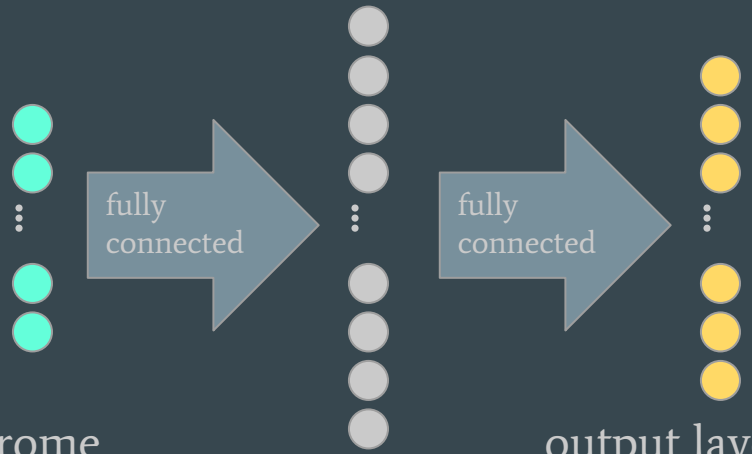


Neural Networks are “trained”: You provide training input/output pairs, and train the network to “learn” the input/output mapping.

Applying Neural Nets to Decoding

Reversal of a one-way function

- Generate millions of errors **e** (for a toric code of distance L).
- Compute corresponding syndromes **s**.
- Train the network to learn the reverse **s** → **e** mapping.

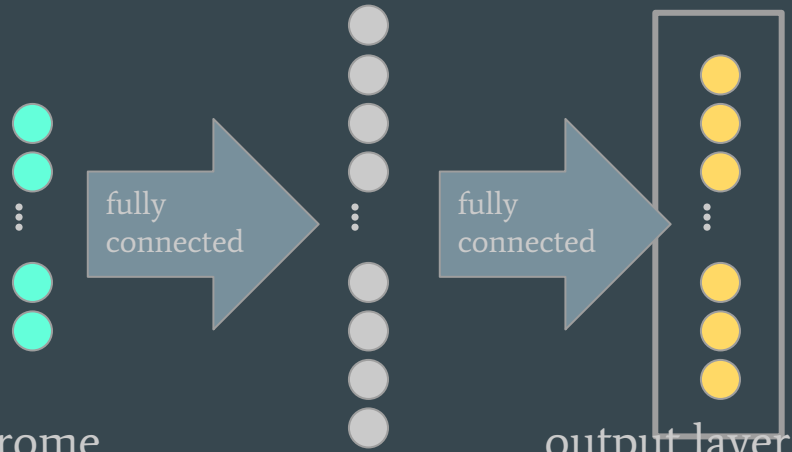


input layer - syndrome
(L^2 neurons per X or Z error type)

output layer - error
($2L^2$ neurons per X or Z error type)

The Output of the Neural Network

- Neural Networks are continuous (even differentiable) maps.
- The output is a real number between 0 and 1.
- The training data is binary (error either happened or not).
- How do we decide whether the Neural Network is predicting an error or not?



input layer - syndrome
(L^2 neurons per X or Z error type)

output layer - error
($2L^2$ neurons per X or Z error type)

Neural Networks do not provide answers, rather probability distributions over possible answers!

At least in the fairly common architecture we are using.

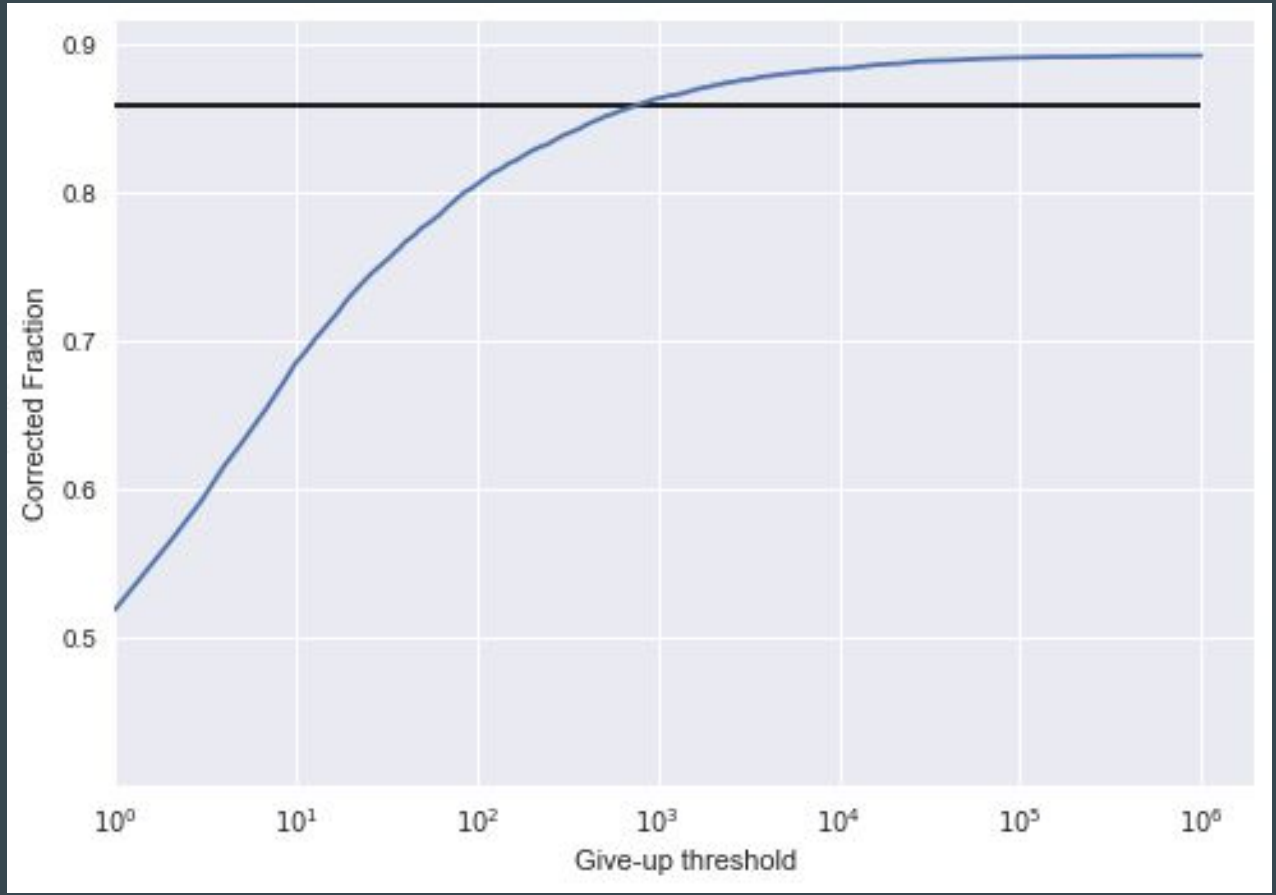
Using buzzwords is fun, but for the rest of the presentation you can think of neural networks as the easiest-to-implement black box to deduce a conditional probability distribution given some data.

Sampling the Neural Network

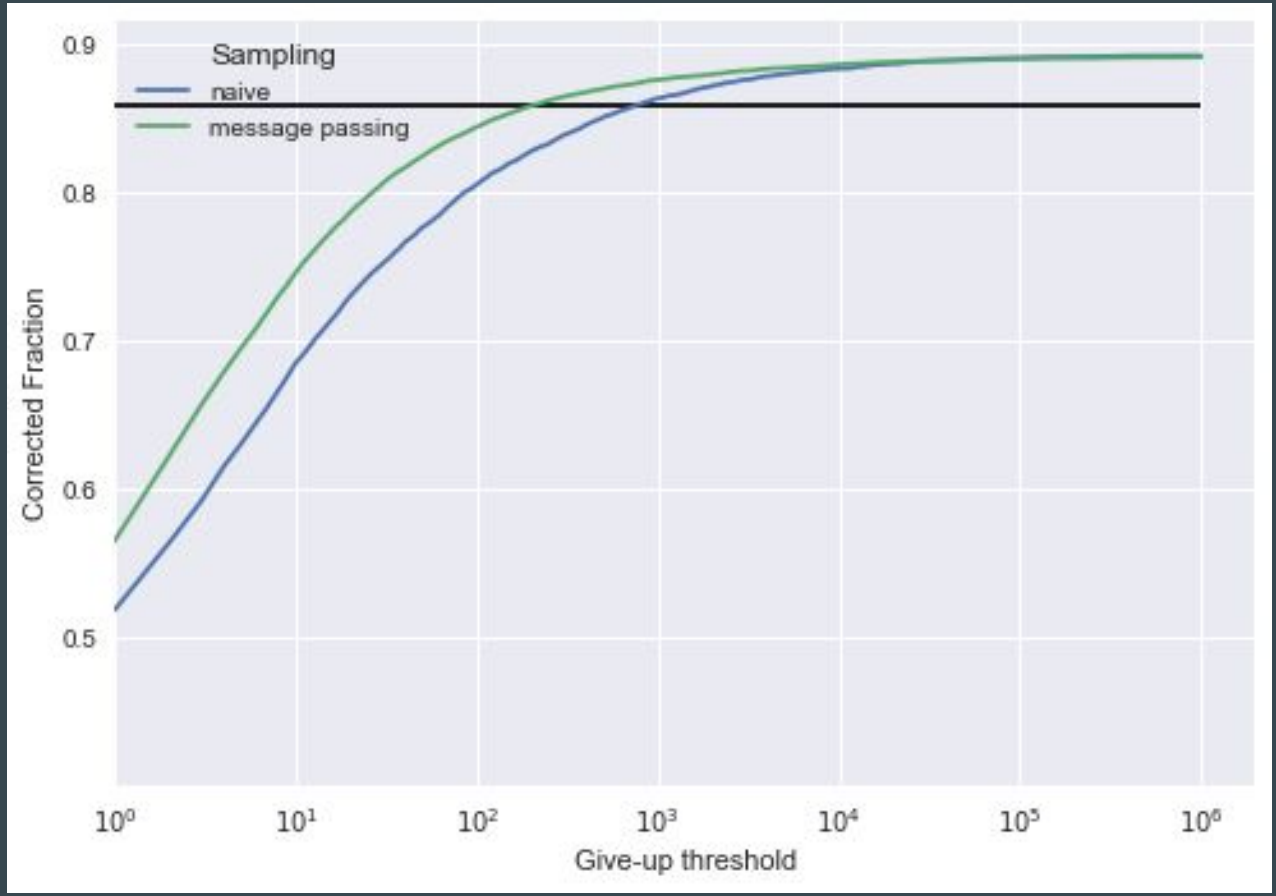
- Measure syndrome **s**
- From the neural network get
  probability distribution over possible errors
  $\mathbf{P_s(e)}$
- Sample an **e** from **P**
- You can do even better - you can check whether
  $\mathbf{s_{new}=s+He}$ is trivial
- If not, then continue sampling until it is (or you give up)

It beats Minimal Weight Perfect Matching (the black line) in two ways:

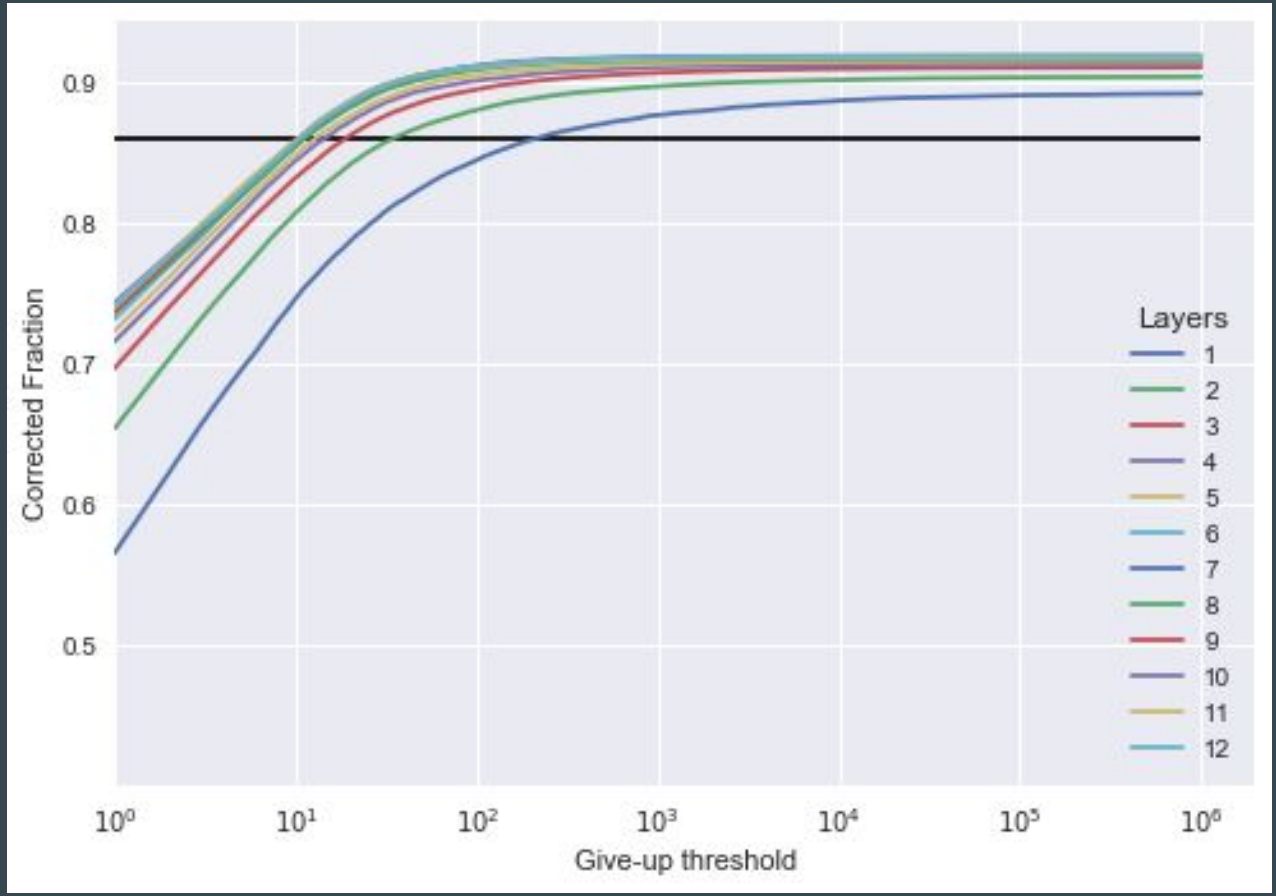
- It knows about correlations between X and Z
- It gives the most probable error, not the one with “minimal free energy”



Application: Distance 5 Toric Code at 10% physical error rate

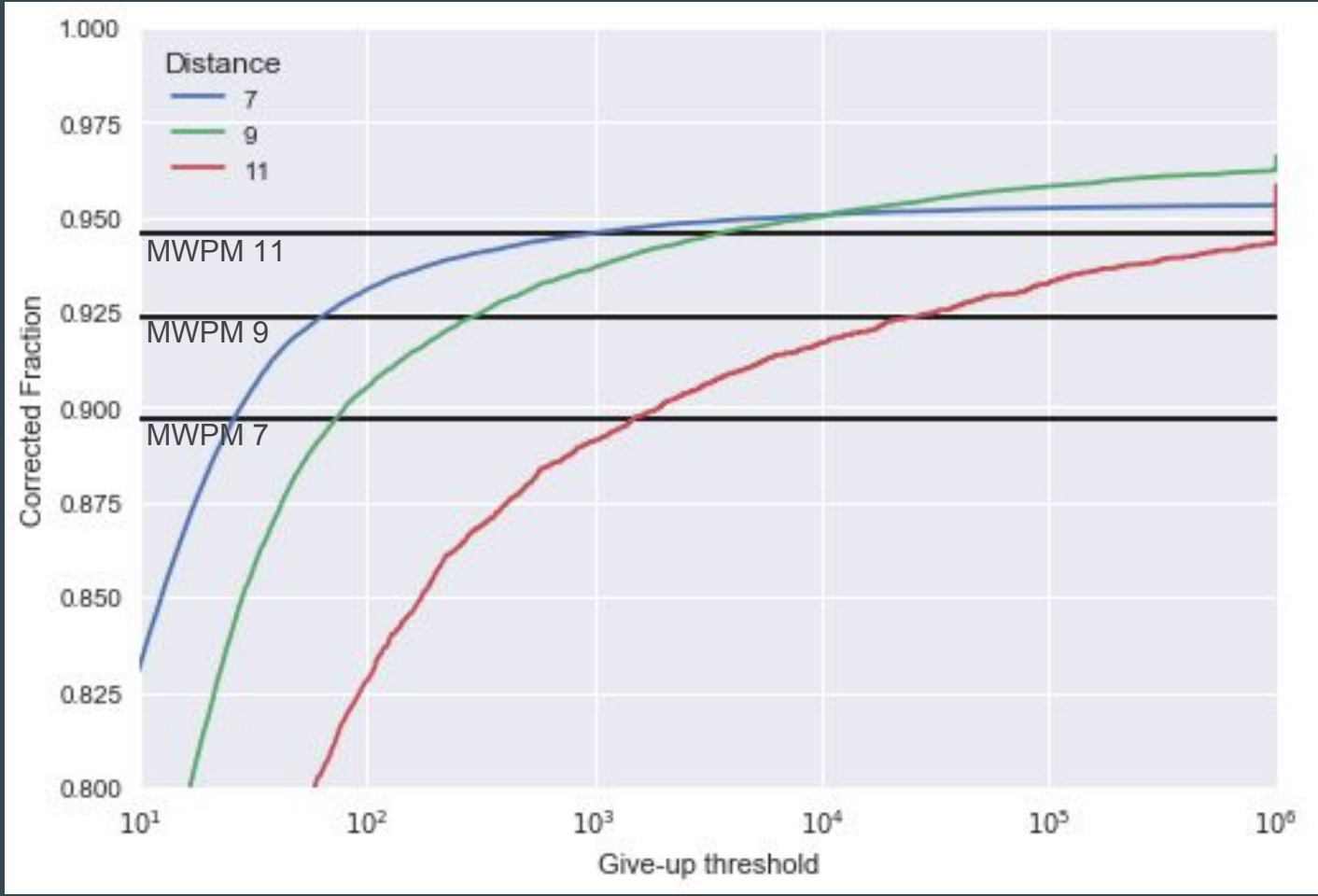


Smarter Sampling (Message Passing / Belief Propagation)


0.9
0.8
0.7
0.6
0.5
Corrected Fraction
Layers
1
2
3
4
5
6
7
8
9
10
11
12
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Give-up threshold

More Layers

Bigger Codes



Conclusions and Work-in-progress

- Works amazingly well on small instances of the Toric Code
- With ConvNets or RG algorithms it should work on large Toric Codes
- Coupled to intelligent sampling and graph algorithms it should work great on big LDPC codes.
- The network can be recurrent (have a feedback loop) in order to be used beyond single shot decoding.

stefan.krastanov@yale.edu
Work not published yet but I would be happy to discuss and share resources.

Further Reading (and sources for some intro graphics)

- Stanford CS231 by Andrej Karpathy cs231n.github.io
- Keras's software package documentation keras.io
- "The Unreasonable Effectiveness of Recurrent Neural Networks", A. Karpathy
- "Understanding LSTM Networks", Chris Olah
- "An Intuitive Explanation of Convolutional Neural Networks", Ujjwal Karn
- "A theory of the learnable", Valiant

Hyperparameter Search Results

- One or more hidden layers of size 4x the input layer (after that we reach diminishing returns)
- Hidden layer activation: tanh
- Output layer activation: sigmoid
- Loss: binary crossentropy (to be discussed in more detail)