Neural Network Decoders

April 2017, Stefan Krastanov, Liang Jiang

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

Error Correcting Codes (ECC) 101

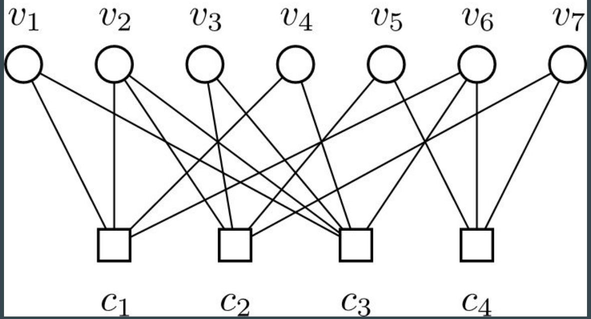
Classical Binary ECC

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.

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



$$s = He$$

Disclaimer: erasure errors are detectable (less severe) - we will not worry about them.

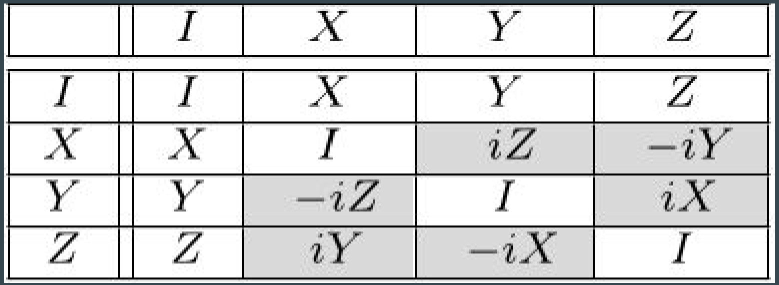
Error Model for a Qubit

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.



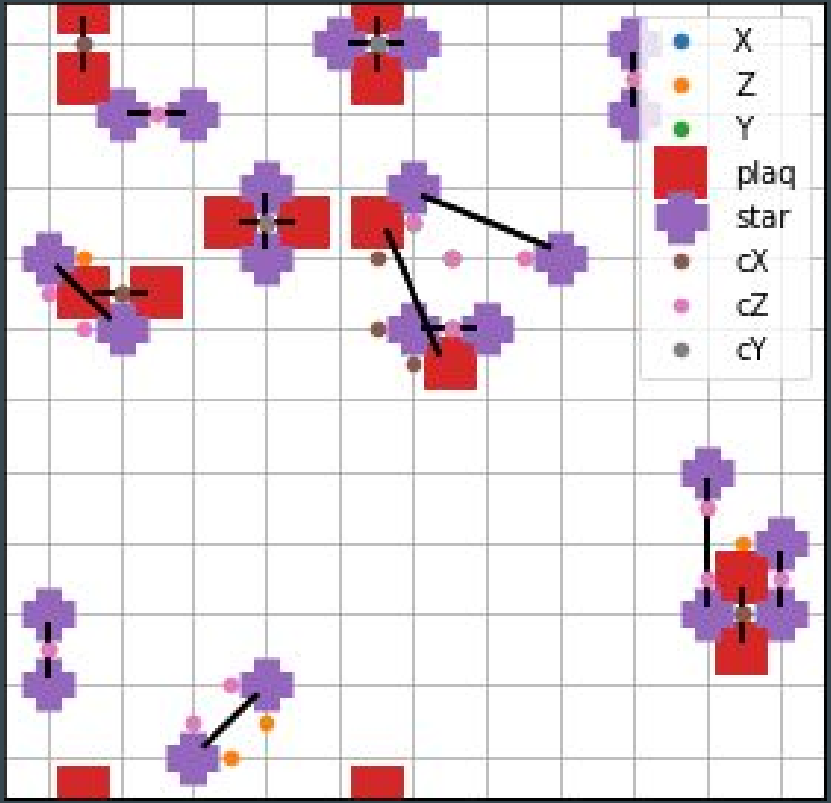
	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-iZ$	I	iX
Z	Z	iY	$-iX$	I

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

Decoding (Correcting) the Code

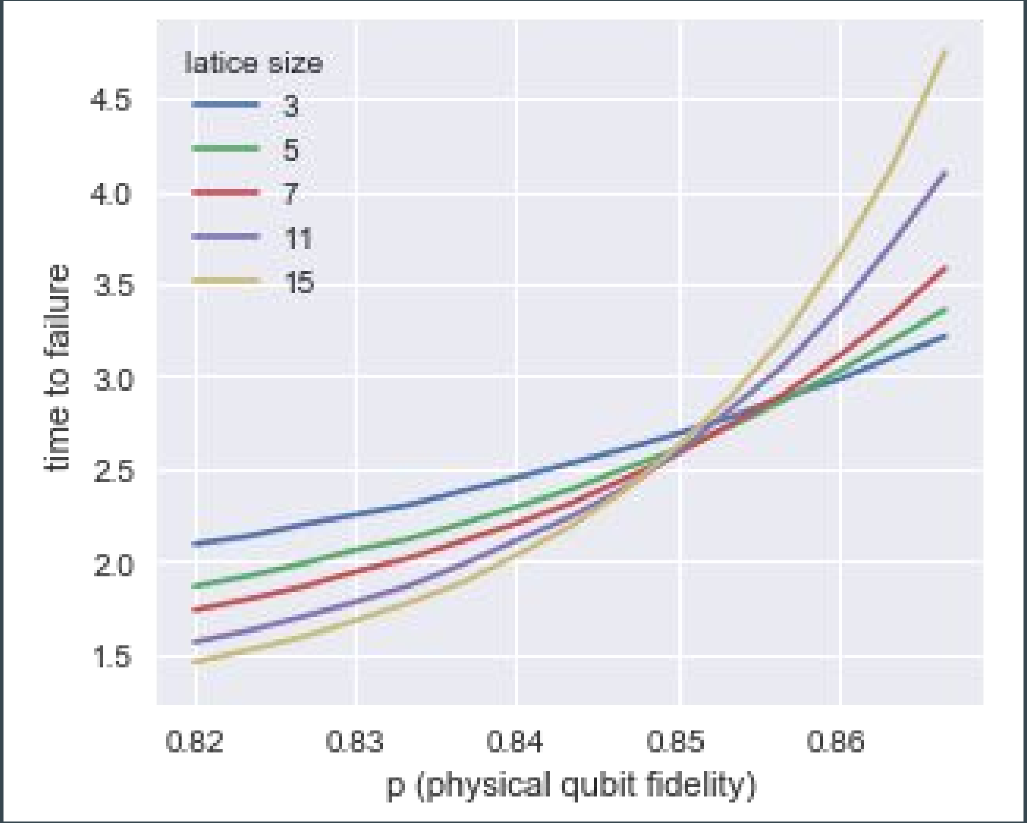
- **e** to **s** is easy
- **s** to **e** is impossible (it is one to many relation)
- **s** to *most probable* **e** is difficult (infeasible, i.e. NP complete)
- **s** to *fairly probable* **e** if there is *additional structure* can be "easy"

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



Example of Additional Structure - Toric Code

Toric Code cont. - memory lifetime vs physical qubit quality



Neural Networks 101

Valiant’s Theorem: Learning “might” be “feasible”

A baby is learning a language by listening:

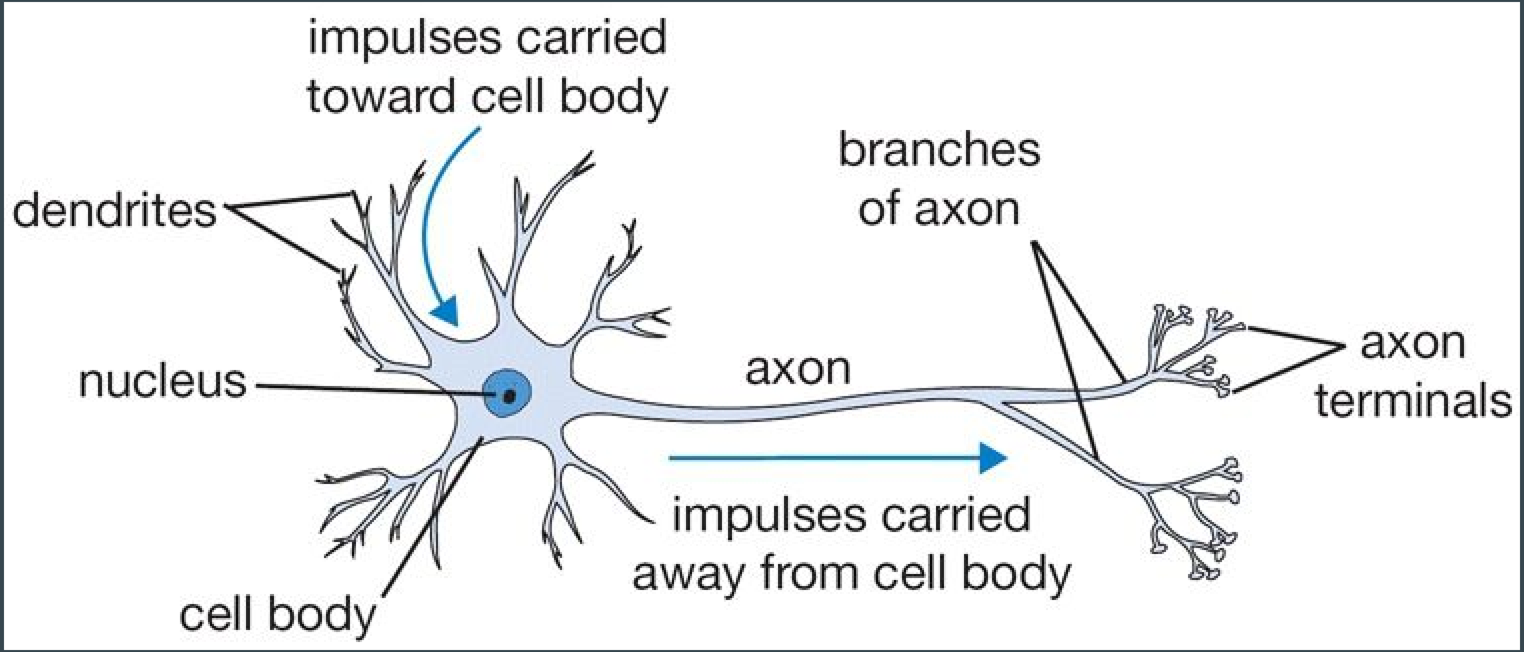
x - a sentence
D - the set of all possible (correct or not) sentences
f - a boolean function saying whether a sentence is grammatically correct
h - an approximation of f that the baby is trying to learn
ε - the fraction of misclassified sentences

$$Pr_{x \in D}[h(x) = f(x)] \geq 1 - \varepsilon$$

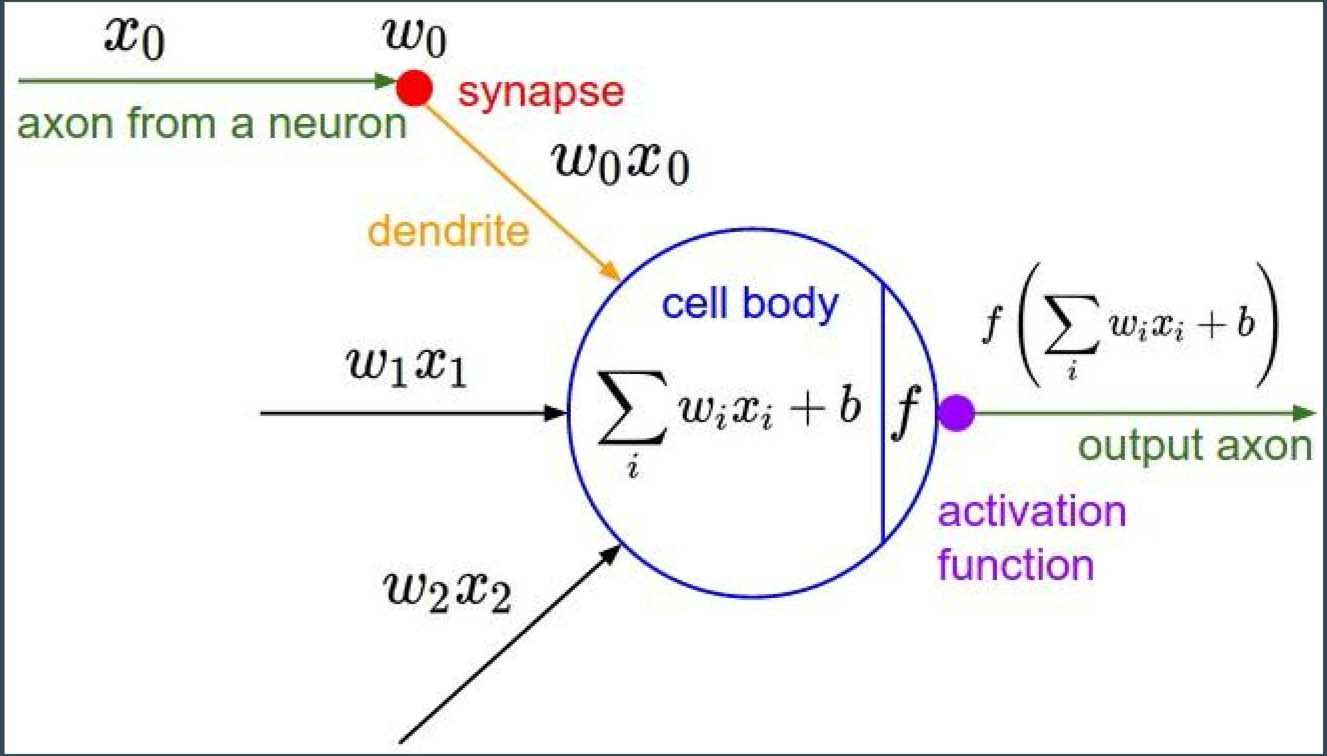
$$m \geq \frac{1}{\varepsilon} \log\left(\frac{|C|}{\delta}\right)$$

m (defined above) samples are sufficient:

C - the set of all possible h
δ - (the probability of learning an h with misclassification rate higher than ε)

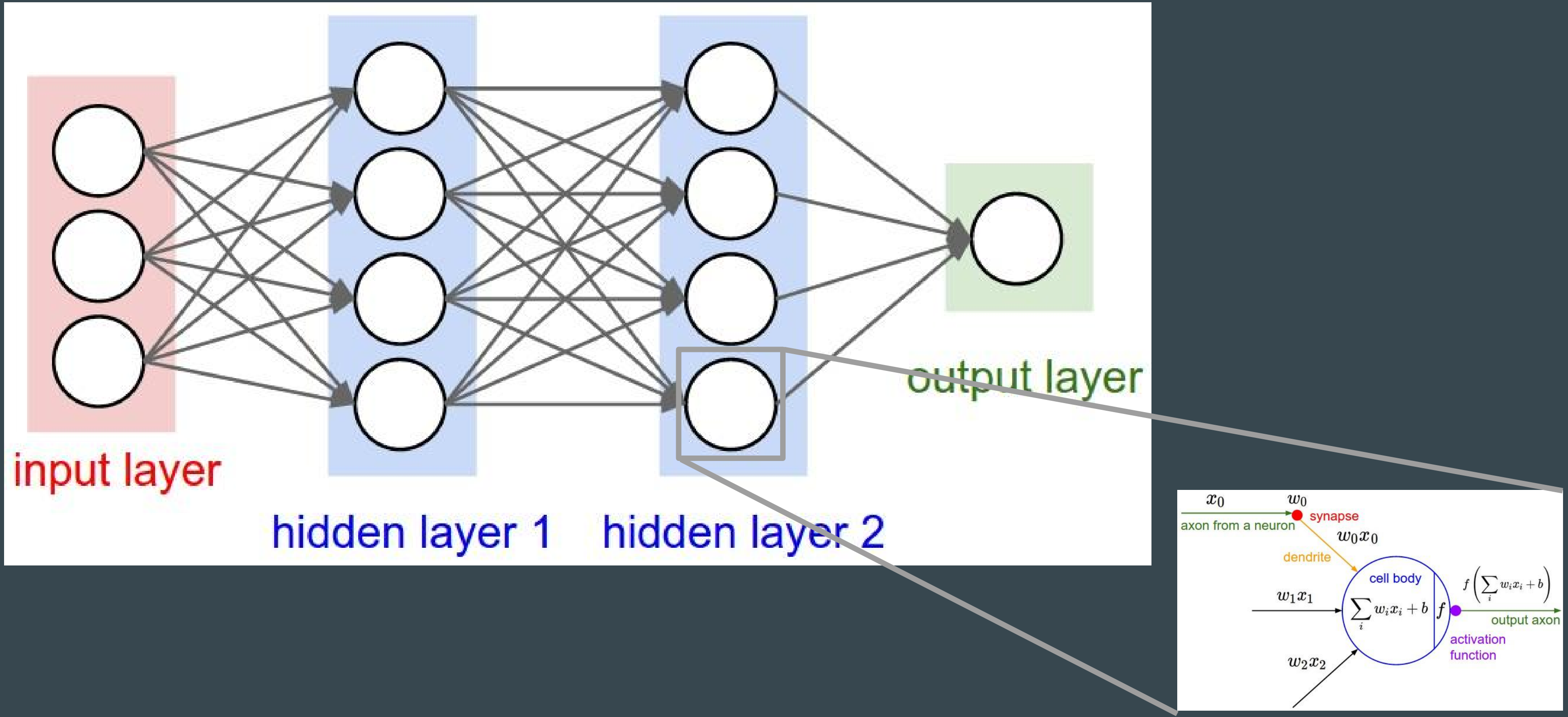

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

A Neuron

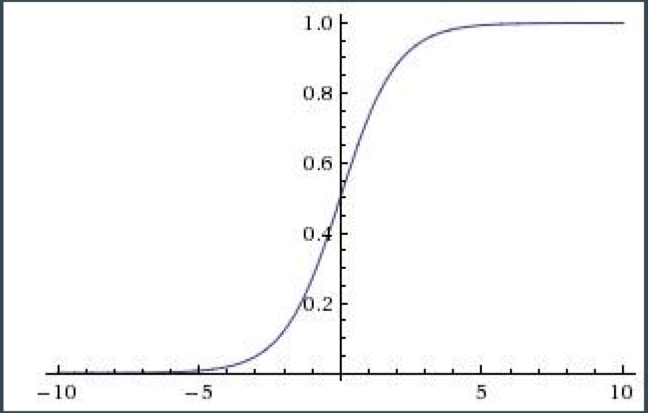


A Nicer Neuron

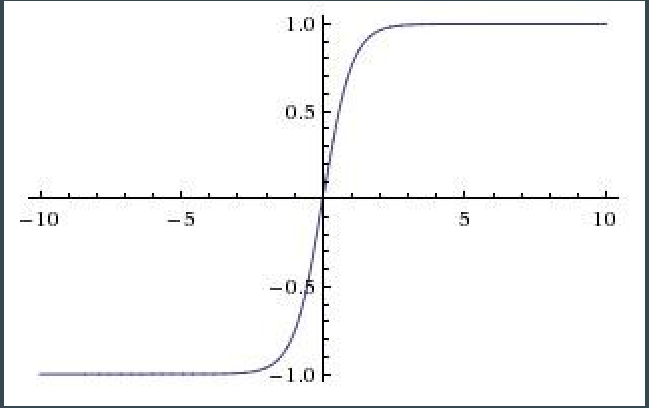
Layered Neural Network



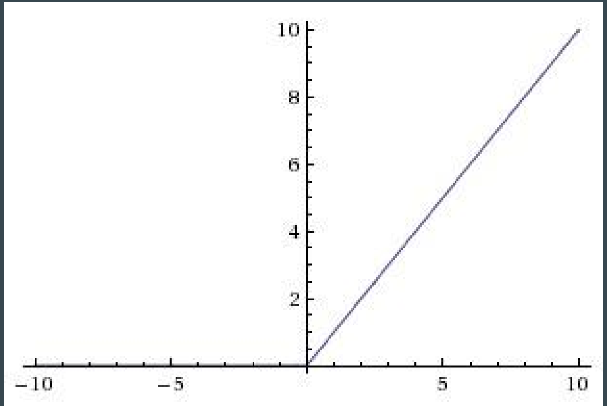
Nonlinearity (a.k.a. Activation Function)



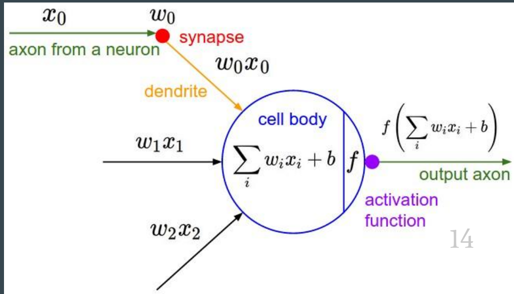
Sigmoid (Fermi distribution)



tanh



ReLU



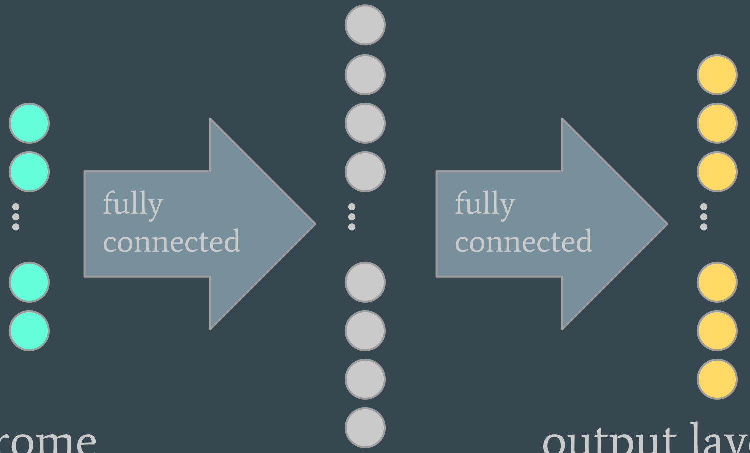
Training

- Gradient descent with automated analytical gradients.
- Training data sets need at least millions of data points.
- Too slow to evaluate the gradient on the entire training set:
  - Use batches (evaluate on small subset of the training set)
  - Use epochs (iterate through the training set multiple times)

Applying Neural Nets to Decoding

Reversal of a one-way function

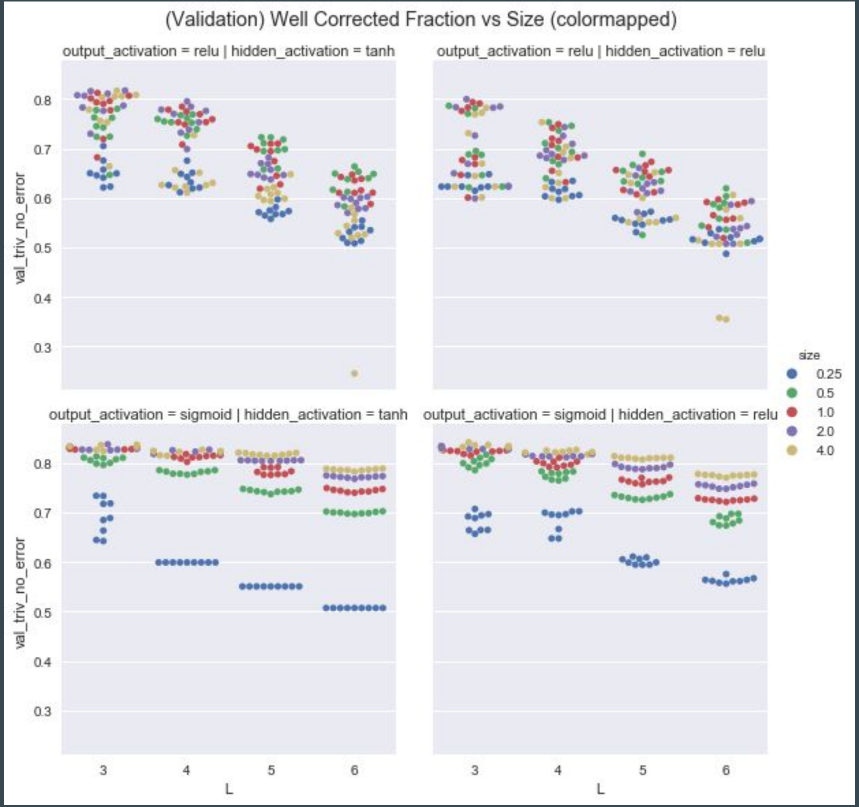
- Generate millions of errors **e** (for a toric code of distance L).
- Compute corresponding syndromes **s**.
- Train the network to learn the reverse **s** → **e** mapping.



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

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

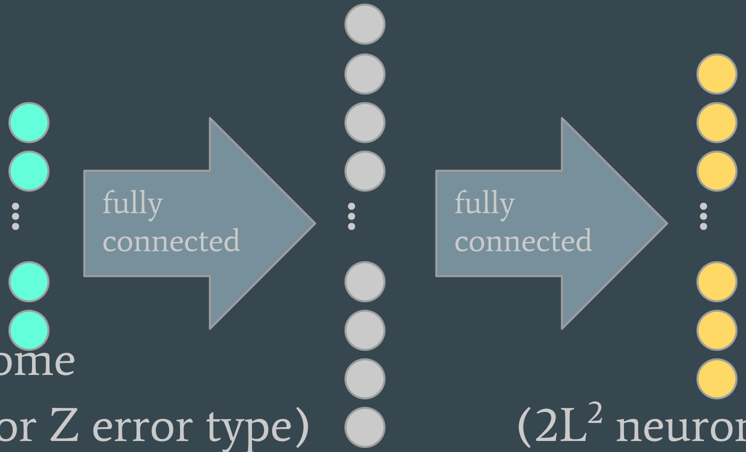
Hyperparameter Search



The details of the plot do not matter. We are just showing off how much work we did.

Hyperparameter Search Results

- One hidden layer of size 4x the input layer (after that we reach diminishing returns)
- Hidden layer activation: tanh
- Output layer activation: sigmoid
- Loss: binary crossentropy

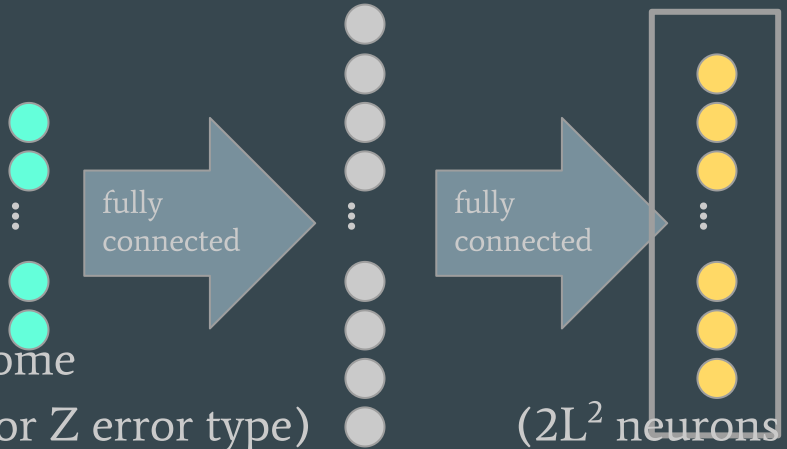


input layer - syndrome
(L^2 neurons per X or Z error type)

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 use 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.

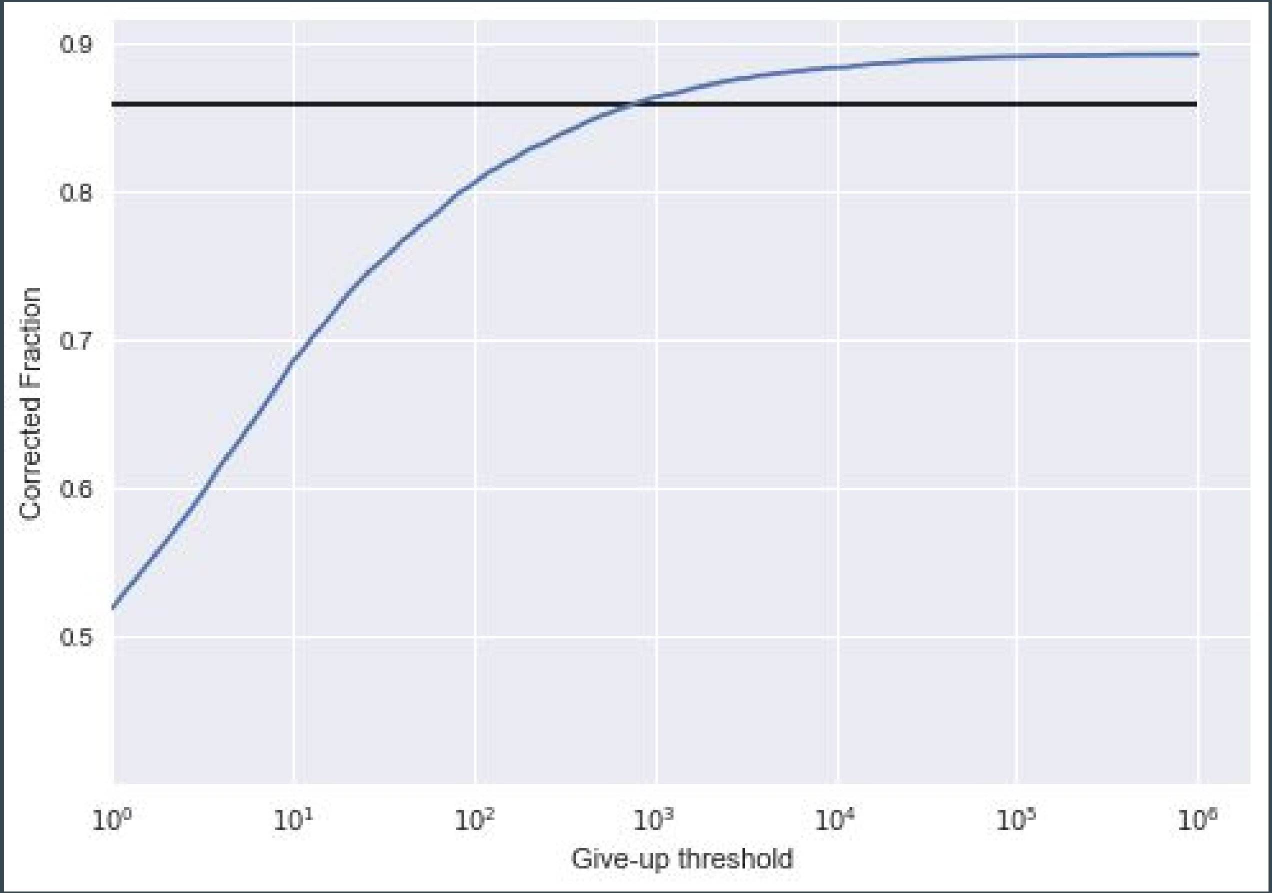
Using buzzwords is fun, but for the rest of the presentation you can think of neural networks as the easiest-to-implement black box to deduce a conditional probability distribution given some data.

Sampling the Neural Network

- Measure syndrome $\mathbf{s}$
- From the neural network get
  probability distribution over possible errors
  $\mathbf{P_s(e)}$ *(we are not using the neural network after this point)*
- Sample an $\mathbf{e}$ from $\mathbf{P}$
- 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"



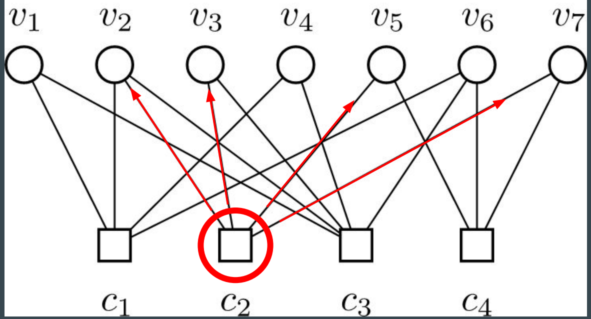
Application: Distance 5 Toric Code at 10% physical error rate

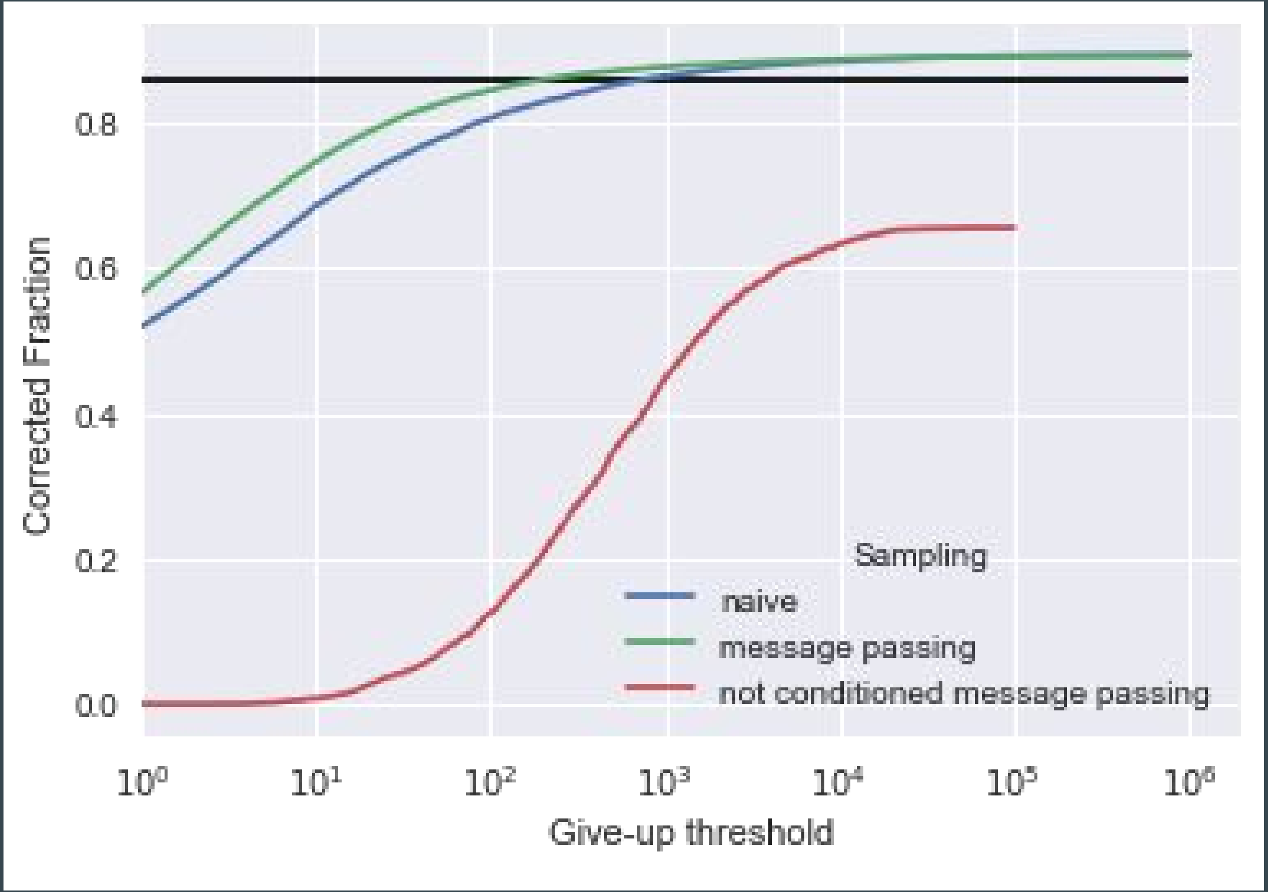
Slightly Smarter Sampling

If $\mathbf{s}_{\mathbf{new}}$=$\mathbf{s}$+$\mathbf{He}$ is not trivial, resample only the “broken” nodes.

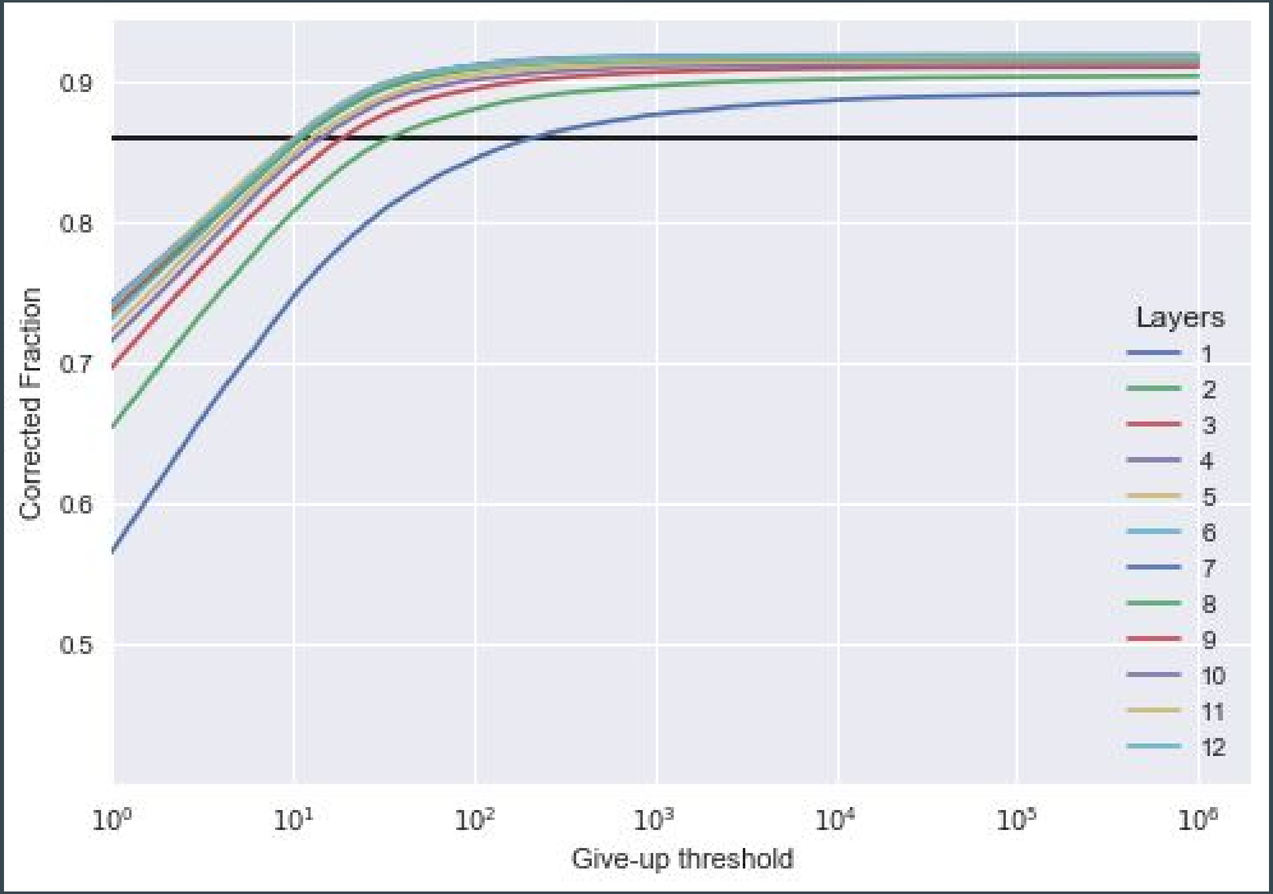
Poorman’s version of
“hard decision”
belief propagation.

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



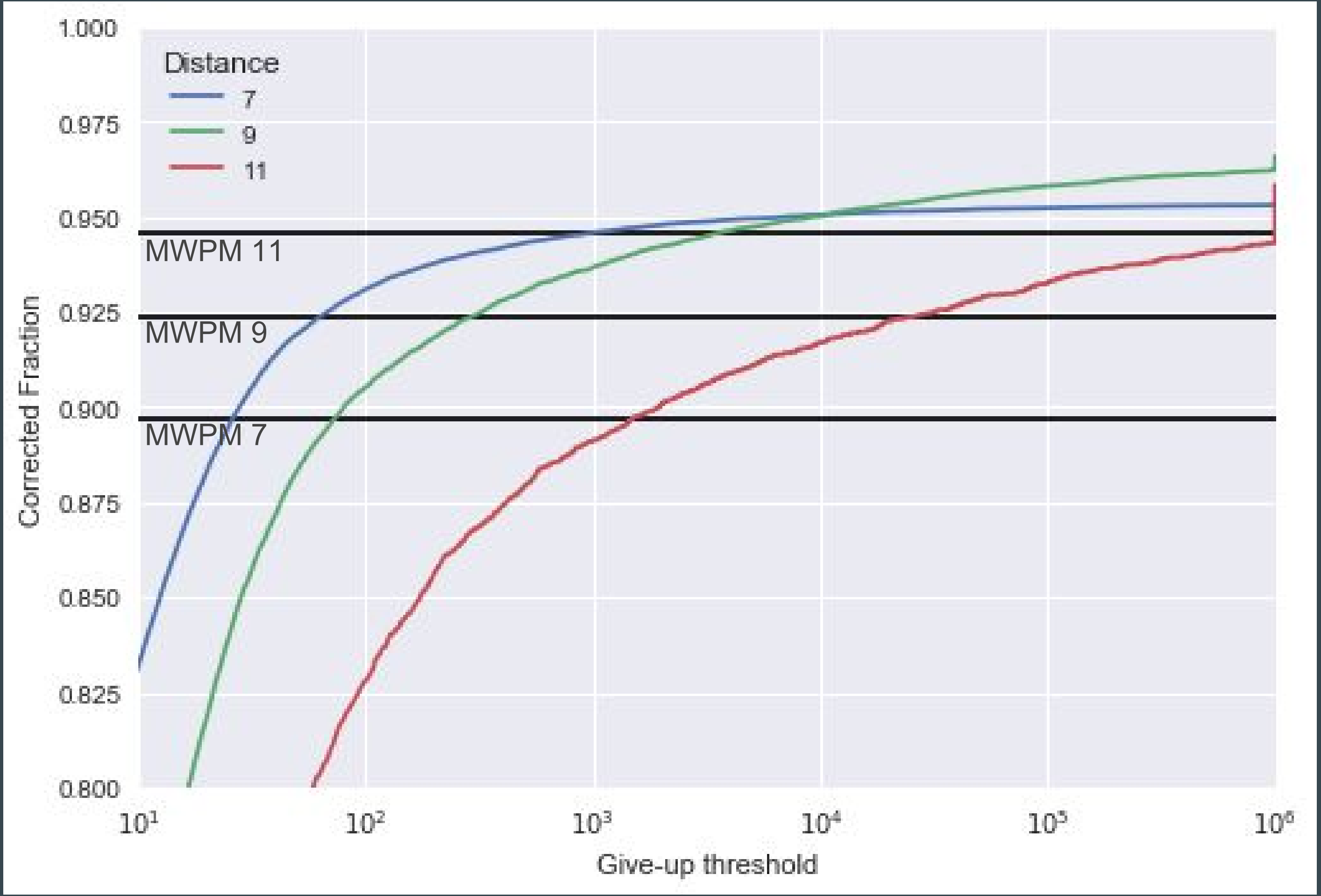


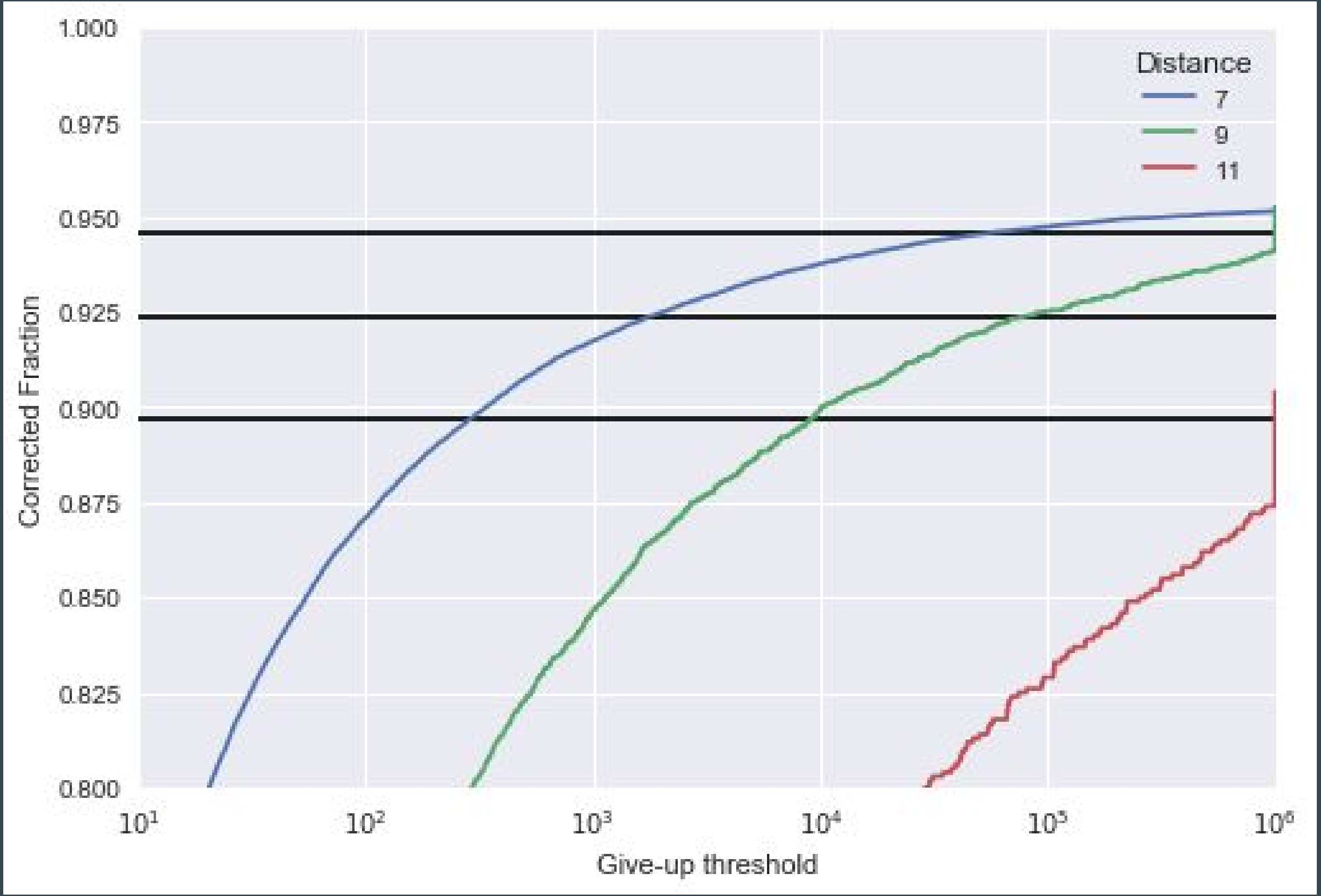
Smarter Sampling (Message Passing / Belief Propagation)


Corrected Fraction
0.9
0.8
0.7
0.6
0.5
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





Bigger Codes (naive sampling)

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.

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

TODO

For Toric Code

- Use the lattice structure: RG or ConvNet inspired decoders (but is it worth it?)
- Get a threshold

For LDPC Codes (and Toric Codes)

- See how well the current approach works on them
- Look into the statistics of the NN output (are correlations preserved)
- See whether belief propagation can be seeded with the learnt distribution
- Use a recurrent or generative NN (anything that remembers correlations)

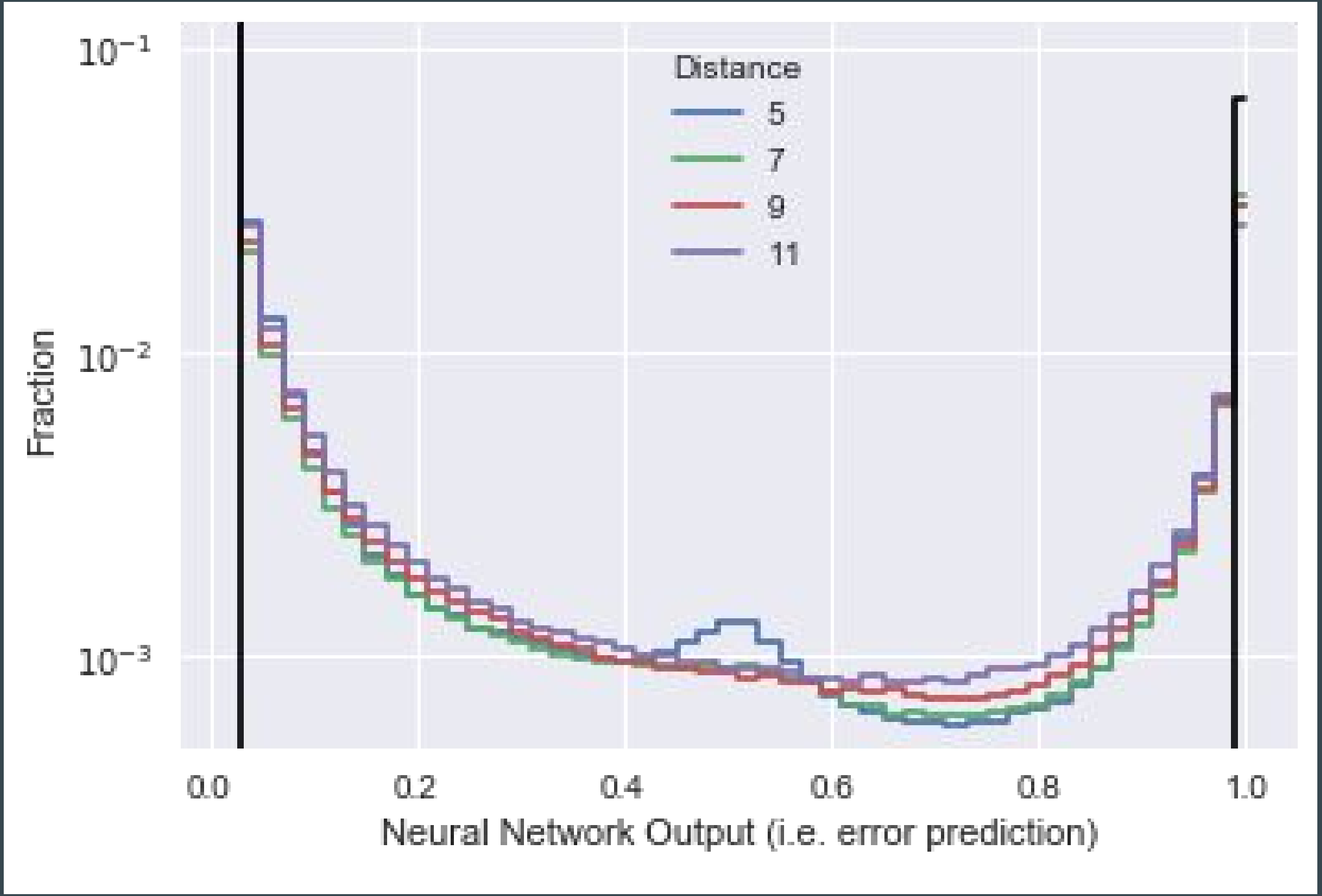
The Loss: Binary Crossentropy

Comparing probability distribution p to probability distribution q:

$$H(p, q) = -\sum_x p(x)\ \log q(x)$$

Using this as the loss (i.e. training goal to be minimized) reinforces the interpretation of the result as a “learned” conditional probability distribution.

Output Stats


Distance
Fraction
10⁻¹
10⁻²
10⁻³
Neural Network Output (i.e. error prediction)