Building a network without backprop

Bernhard Gstrein

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a “network” without backprop

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a “network” without backprop
  - Using local search, SAT solving, etc.

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a "network" without backprop
  - Using local search, SAT solving, etc.
  - Logic synthesis; input: goals and constraints, output: network structure

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a "network" without backprop
  - Using local search, SAT solving, etc.
  - Logic synthesis; input: goals and constraints, output: network structure
- [Chatterjee 2018] describes a scheme to build something similar to a neural network without backprop

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a "network" without backprop
  - Using local search, SAT solving, etc.
  - Logic synthesis; input: goals and constraints, output: network structure
- [Chatterjee 2018] describes a scheme to build something similar to a neural network without backprop
  - Basic idea: lookup tables ("luts")

Background/Motivation

- For learning neural networks, backpropagation is almost exclusively used
- Our goal: building a “network” without backprop
  - Using local search, SAT solving, etc.
  - Logic synthesis; input: goals and constraints, output: network structure
- [Chatterjee 2018] describes a scheme to build something similar to a neural network without backprop
  - Basic idea: lookup tables (“luts”)
  - Some properties of neural networks are promiment in lut networks too

Table of Contents

Table of Contents

What is a lookup table (lut)?

	$\boldsymbol{X}$			y
	Lives in water	Has eyes	Has limbs	Vertebrate
$\boldsymbol{x}_1$	0	1	1	1
$\boldsymbol{x}_2$	1	1	0	1
$\boldsymbol{x}_3$	1	0	0	0

Model for classification of animals into vertebrates/invertebrates

What is a lookup table (lut)?

	$\boldsymbol{X}$			y
	Lives in water	Has eyes	Has limbs	Vertebrate
$\boldsymbol{x}_1$	0	1	1	1
$\boldsymbol{x}_2$	1	1	0	1
$\boldsymbol{x}_3$	1	0	0	0

Model for classification of animals into vertebrates/invertebrates

- We must binarize the features and labels

What is a lookup table (lut)?

	$\boldsymbol{X}$			y
	Lives in water	Has eyes	Has limbs	Vertebrate
$\boldsymbol{x}_1$	0	1	1	1
$\boldsymbol{x}_2$	1	1	0	1
$\boldsymbol{x}_3$	1	0	0	0

Model for classification of animals into vertebrates/invertebrates

- We must binarize the features and labels
- We must limit the complexity

What is a lookup table (lut)?

	$\boldsymbol{X}$ Lives in water	Has eyes	Has limbs	y Vertebrate
$\boldsymbol{x}_1$	0	1	1	1
$\boldsymbol{x}_2$	1	1	0	1
$\boldsymbol{x}_3$	1	0	0	0

Model for classification of animals into vertebrates/invertebrates

- We must binarize the features and labels
- We must limit the complexity
  - Suppose we have 784 columns $\rightarrow$ $2^{784} \propto 10^{236}$

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \rightarrow [0, 1]$

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \to [0, 1]$
- We binarize the data using the operator > 0.5

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \to [0, 1]$
- We binarize the data using the operator > 0.5
- Labels: $y = 0$ (numbers 0-4), $y = 1$ (numbers 5-9)

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \rightarrow [0, 1]$
- We binarize the data using the operator > 0.5
- Labels: $y = 0$ (numbers 0-4), $y = 1$ (numbers 5-9)
- We end up with

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \to [0, 1]$
- We binarize the data using the operator > 0.5
- Labels: $y = 0$ (numbers 0-4), $y = 1$ (numbers 5-9)
- We end up with
  - Features $\boldsymbol{X}$: matrix of shape $(N, 784)$, boolean entries

Preprocessing data

- MNIST dataset: 28x28 images of handwritten digits (0-9)
- We unroll the images: $28 \cdot 28 = 784$
- We map the values: $[0, 255] \to [0, 1]$
- We binarize the data using the operator > 0.5

- Labels: $y = 0$ (numbers 0-4), $y = 1$ (numbers 5-9)

- We end up with
  - Features $\boldsymbol{X}$: matrix of shape $(N, 784)$, boolean entries
  - Labels y: vector of shape $(N,)$, boolean entries

A single lut

- Every example is an instance of a “bit pattern” (e.g. $\boldsymbol{x} = 10$) and has a label (e.g. $y = 1$)

A single lut

- Every example is an instance of a "bit pattern" (e.g. $\boldsymbol{x} = 10$) and has a label (e.g. $y = 1$)
- We need a classification for each bit pattern: $f(00) = ?$, $f(01) = ?$, $f(10) = ?$, $f(11) = ?$

A single lut

- Every example is an instance of a “bit pattern” (e.g. $\boldsymbol{x} = 10$) and has a label (e.g. $y = 1$)
- We need a classification for each bit pattern: $f(00) = ?$, $f(01) = ?$, $f(10) = ?$, $f(11) = ?$
- For each bit pattern in $\boldsymbol{X}$, we cound how many times $y = 0$ and $y = 1$

A single lut

- Every example is an instance of a "bit pattern" (e.g. $\boldsymbol{x} = 10$) and has a label (e.g. $y = 1$)
- We need a classification for each bit pattern: $f(00) = ?$, $f(01) = ?$, $f(10) = ?$, $f(11) = ?$
- For each bit pattern in $\boldsymbol{X}$, we cound how many times $y = 0$ and $y = 1$

$$f(\text{bit pattern}) = \begin{cases} 0 & \text{if} \quad \sum\limits_{y=0} > \sum\limits_{y=1} \\ 1 & \text{if} \quad \sum\limits_{y=0} < \sum\limits_{y=1} \\ \text{rand}(0,1) & \text{if} \quad \sum\limits_{y=0} = \sum\limits_{y=1} \end{cases}$$

A single lut: example

Training set

$\boldsymbol{X}$	y
000	0
000	1
000	1
001	1
100	0
110	0
110	1

A single lut: example

Training set

$\boldsymbol{X}$	y
000	0
000	1
000	1
001	1
100	0
110	0
110	1

bit pattern	$\sum_{y=0}$	$\sum_{y=1}$
000	1	2
001	0	1
010	0	0
011	0	0
100	1	0
101	0	0
110	1	1
111	0	0

A single lut: example

Training set

$\boldsymbol{X}$	y
000	0
000	1
000	1
001	1
100	0
110	0
110	1

bit pattern	$\sum_{y=0}$	$\sum_{y=1}$
000	1	2
001	0	1
010	0	0
011	0	0
100	1	0
101	0	0
110	1	1
111	0	0

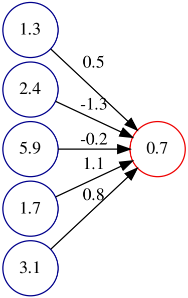
bit pattern	f
000	1
001	1
010	0*
011	1*
100	0
101	1*
110	1*
111	0*

*: randomly chosen entries

Table of Contents

A single neuron

Neural network

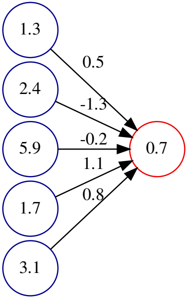


$z = 1.3 \cdot 0.5 - 2.4 \cdot 1.3 - 5.9 \cdot 0.2 + 1.7 \cdot 1.1 + 3.1 \cdot 0.8 = 0.7$

$f(z) = \text{ReLU}(z) = \max(0, z) = 0.7$

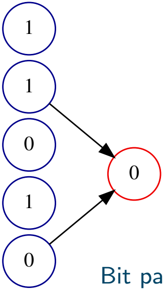
A single neuron

Neural network



$$z = 1.3 \cdot 0.5 - 2.4 \cdot 1.3 - 5.9 \cdot 0.2 + 1.7 \cdot 1.1 + 3.1 \cdot 0.8 = 0.7$$
$$f(z) = \mathsf{ReLU}(z) = \mathsf{max}(0, z) = 0.7$$

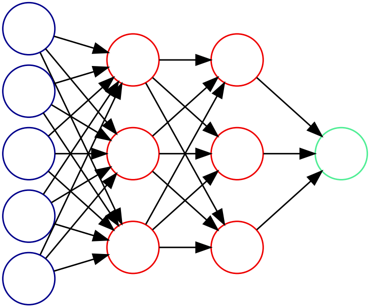
Lut network



Bit pattern	f
00	0
01	1
10	0
11	1

Neural network - lut network: comparison

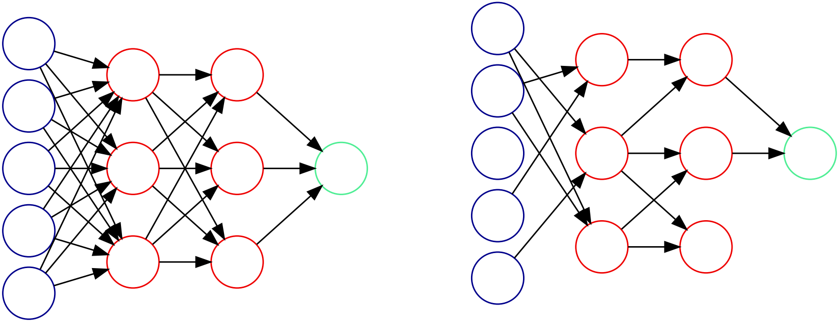
Neural network



Neural network - lut network: comparison

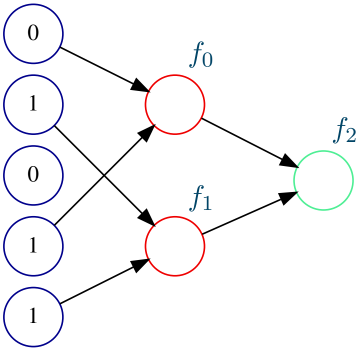
Neural network

Lut network



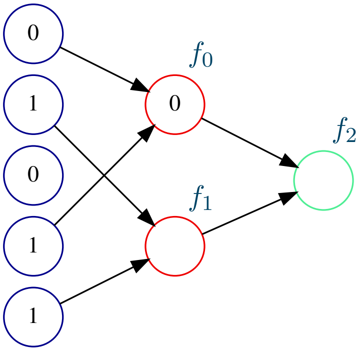
Lut network: example

bit pattern	f_0	f_1	f_2
00	1	1	1
01	0	0	1
10	1	0	0
11	0	1	0



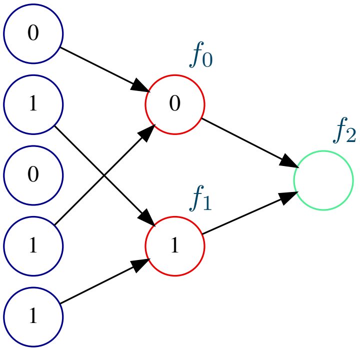
Lut network: example

bit pattern	f_0	f_1	f_2
00	1	1	1
01	0	0	1
10	1	0	0
11	0	1	0



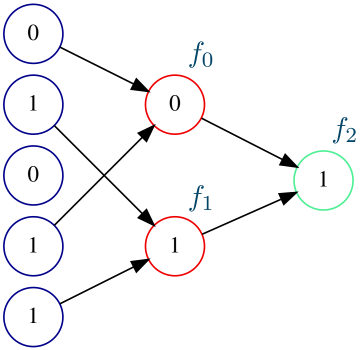
Lut network: example

bit pattern	f_0	f_1	f_2
00	1	1	1
01	0	0	1
10	1	0	0
11	0	1	0



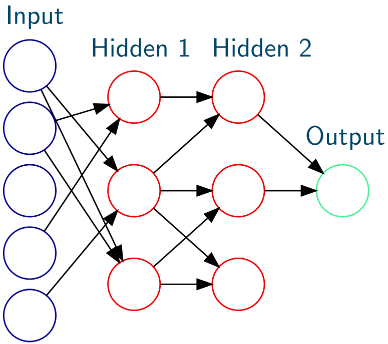
Lut network: example

bit pattern	f_0	f_1	f_2
00	1	1	1
01	0	0	1
10	1	0	0
11	0	1	0



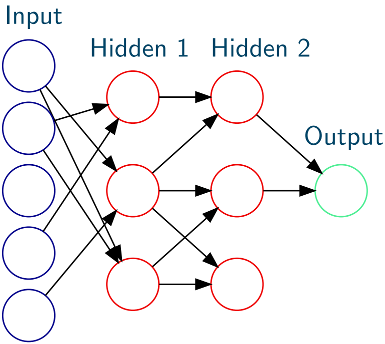
Training a network of luts

- Lut network: layers are trained successively



Training a network of luts

- Lut network: layers are trained successively
- Random choice of k columns for each lut



Training a network of luts

- Lut network: layers are trained successively
- Random choice of k columns for each lut
- Label vector is **always** y

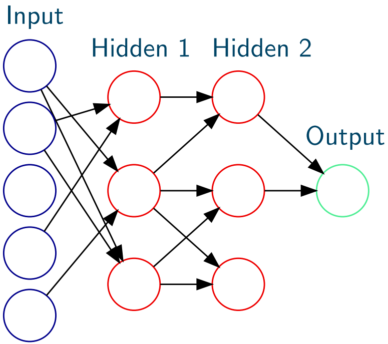


Table of Contents

First experiment

- Network with 5 hidden layers of 1024 luts and 1 lut in the output layer

First experiment

- Network with 5 hidden layers of 1024 luts and 1 lut in the output layer
- Each lut takes 8 inputs

First experiment

- Network with 5 hidden layers of 1024 luts and 1 lut in the output layer
- Each lut takes 8 inputs
- Training accuracy: 0.89

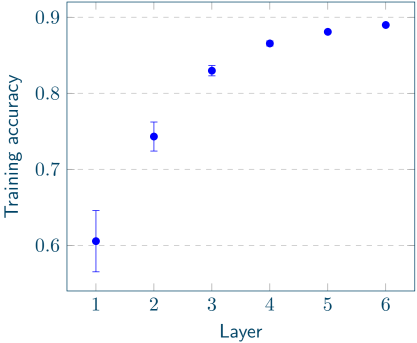
First experiment

- Network with 5 hidden layers of 1024 luts and 1 lut in the output layer
- Each lut takes 8 inputs
- Training accuracy: 0.89
- Accuracy on held-out set: 0.87

First experiment

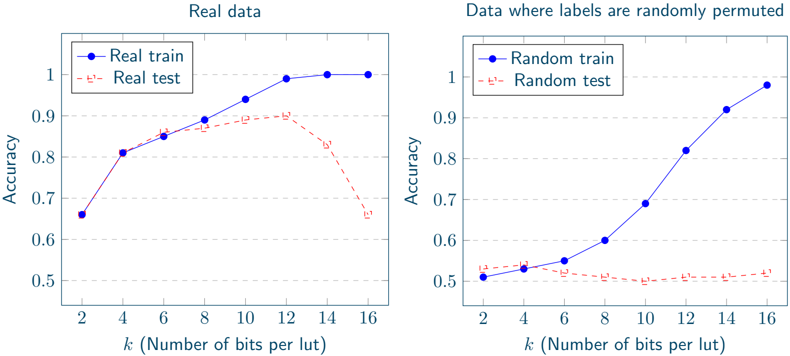
- Network with 5 hidden layers of 1024 luts and 1 lut in the output layer
- Each lut takes 8 inputs
- Training accuracy: 0.89
- Accuracy on held-out set: 0.87
- Results significantly above 0.5

Network of luts: depth improves performance



Training accuracy by layer for a network of 8-input lookup tables on Binary-MNIST. Each layer has 1024 luts except the last one which has only 1. Total height of error bars are two standard deviations.

Network of luts



Effect of varying lookup table size on Binary-MNIST. There are 5 hidden layers with 1024 luts per layer.

Table of Contents

Conclusion

- In the paper, there is also:

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models

- Neural networks and lut networks share some properties:

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models
- Neural networks and lut networks share some properties:
  - Depth improves performance

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models
- Neural networks and lut networks share some properties:
  - Depth improves performance
  - There is generalization on real data

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models
- Neural networks and lut networks share some properties:
  - Depth improves performance
  - There is generalization on real data
  - Random data can be memorized

Conclusion

- In the paper, there is also:
  - More experiments with other datasets and other tasks
  - Comparison of luts to other predictive models

- Neural networks and lut networks share some properties:
  - Depth improves performance
  - There is generalization on real data
  - Random data can be memorized

- **The lut network is built without backprop and is able to perform non-trivial tasks**

Outlook

- We already wrote code that implements lut networks

Outlook

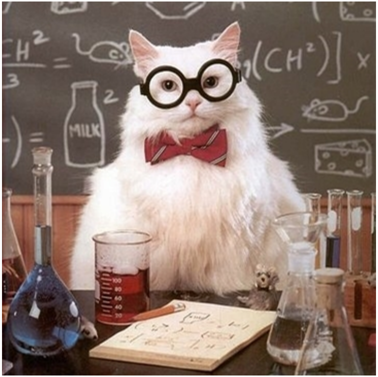
- We already wrote code that implements lut networks
  - Recreate paper results for practical work?

Outlook

- We already wrote code that implements lut networks
  - Recreate paper results for practical work?
- Should we continue with luts?

Outlook

- We already wrote code that implements lut networks
  - Recreate paper results for practical work?
- Should we continue with luts?
- Building an AIG using local search/SAT solving sounds more interesting



Thank you for your attention!

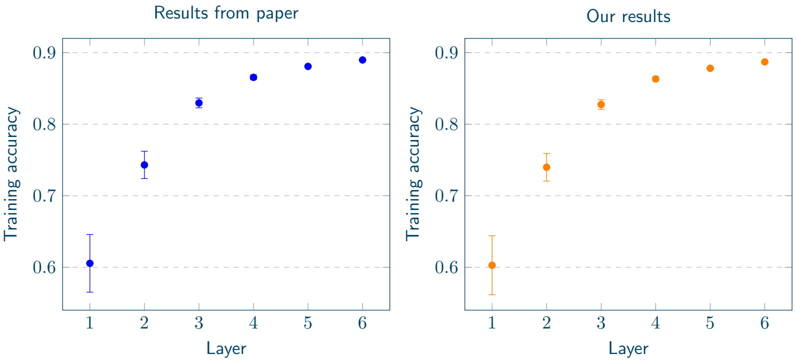
Table of Contents

References

Chatterjee, Satrajit (2018). "Learning and memorization". In: *International Conference on Machine Learning*. PMLR, pp. 755–763.

Table of Contents

Depth improves performance: Our results



Training accuracy by layer for a network of 8-input lookup tables on Binary-MNIST. Each layer has 1024 luts except the last one which has only 1. Total height of error bars are two standard deviations. We can see that our results closely match the results from the paper.