TUTORIAL

The tutorial material (including solutions) is available at the following website:

https://github.com/lucainnocenti/ICTP-winter-college-2020-labs

Machine learning for quantum technologies

Alessandro Ferraro & Luca Innocenti

(Queen's University Belfast)

Outline

Part 1 – Overview of ML methods and applications to QT

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

Part 2 – Introduction to Artificial Neural Networks

1. Perceptron
2. Sigmoid neurons
3. Example: handwritten digits classification
4. Stochastic gradient descend
5. A glimpse of the back-propagation algorithm
6. Overfitting
7. Convolutional neural networks

Useful resources

Books on (classical) machine learning

- M Nielsen, “Neural networks and deep learning”, (web – free)
- A Geron, “Hands-on machine learning with Scikit-Learn and TensorFlow” (basic)
- Russell & Norvig, “Artificial intelligence: a modern approach” (classic)

Video lectures on (classical) machine learning (all freely available)

- F Marquardt, video lectures “Machine learning for physicists”
- “Deep Learning and Reinforcement Learning”, Summer School, Toronto 2018
- A Ng, “Machine learning”, full course, YouTube (Stanford U)
- H Larochelle, “Réseaux neuronaux” (eng), course IF725, YouTube (Sherbrooke U)

Many other resources available on line… (e.g., colah.github.io)

Useful resources

Reviews on machine learning & quantum information & physics

(all available on the ArXiv)

Schuld et al., https://arxiv.org/abs/1409.3097

Biamonte et al., https://arxiv.org/abs/1611.09347

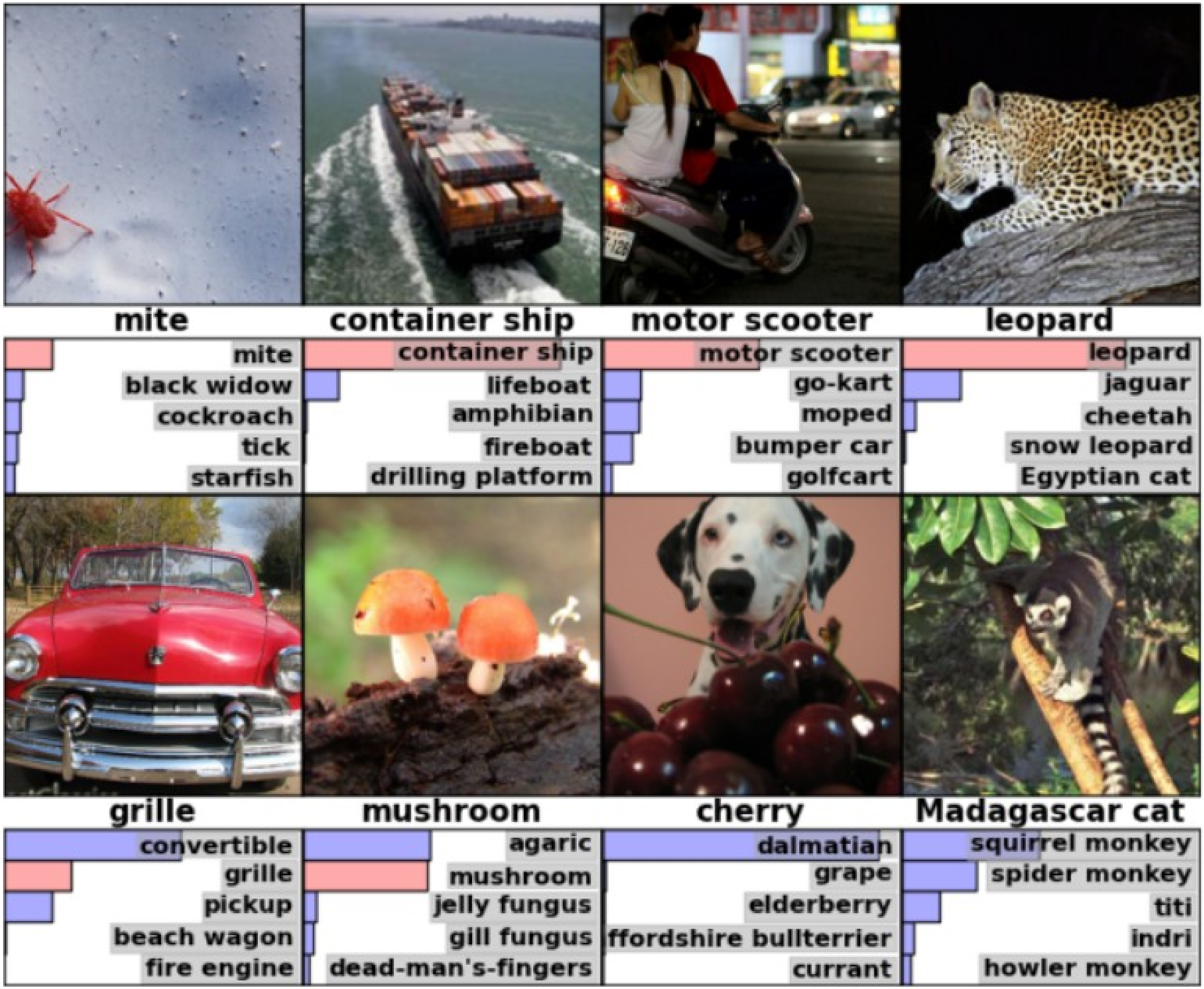
Ciliberto et al., https://arxiv.org/abs/1707.08561

Dunjko et al., https://arxiv.org/abs/1709.02779

Mehta et al., https://arxiv.org/abs/1803.08823

Carleo et al., https://arxiv.org/abs/1903.10563

1.1 Supervised learning



Top: 4 correctly classified examples. Bottom: 4 incorrectly classified examples. Each example has an image, followed by its label, followed by the top 5 guesses with probabilities. From Krizehvsky *et al.* (2012).

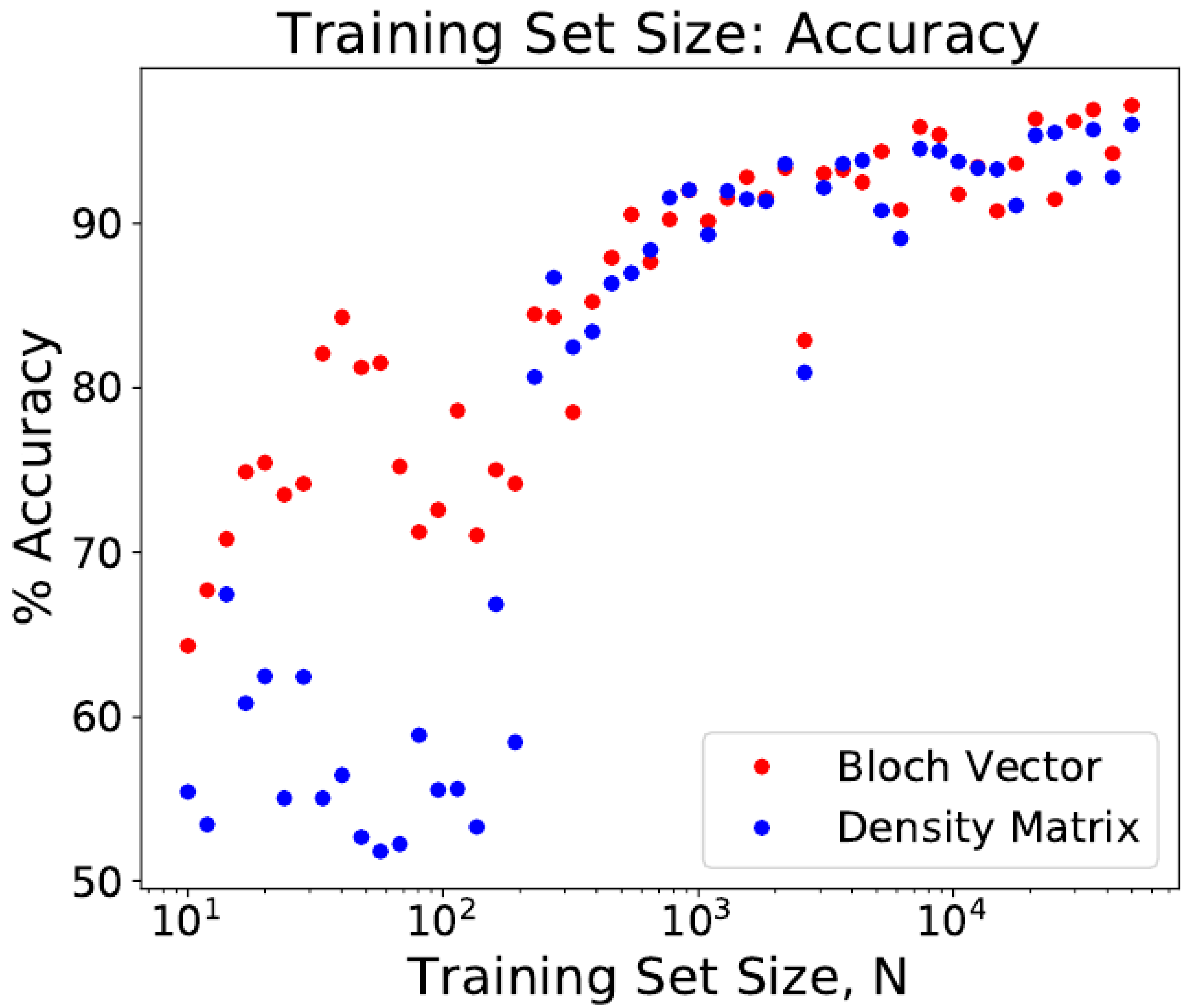
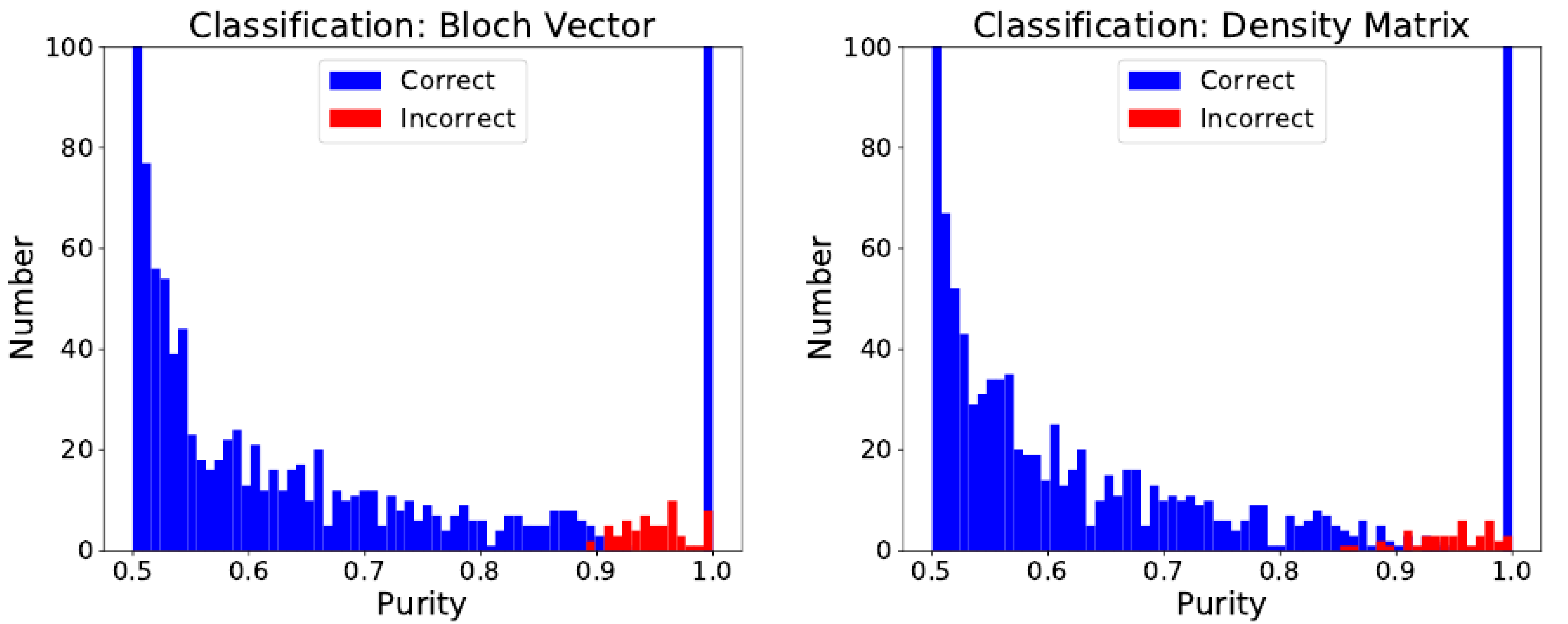


Figure 8: The effect of training set size on the classification accuracy of single qubit states as mixed or pure. A comparison between the vectorised density matrix and Bloch vector inputs is also shown. Each point is the average value of 10 iterations.

[D Craig, Master thesis]



(a) Bloch Vector Input

(b) Density Matrix Input

Figure 9: The distribution of correct and incorrect classification of states as mixed or pure using the Bloch vector and density matrix inputs. A training set of size $N = 10^4$ and test set of size $M = 2 \times 10^3$ was used in each case. In both distributions the classification accuracy was approximately 97%. The y-axis has been limited to 100 in each case as half of the states have purity $\mathcal{P} = 1$.

[D Craig, Master thesis]

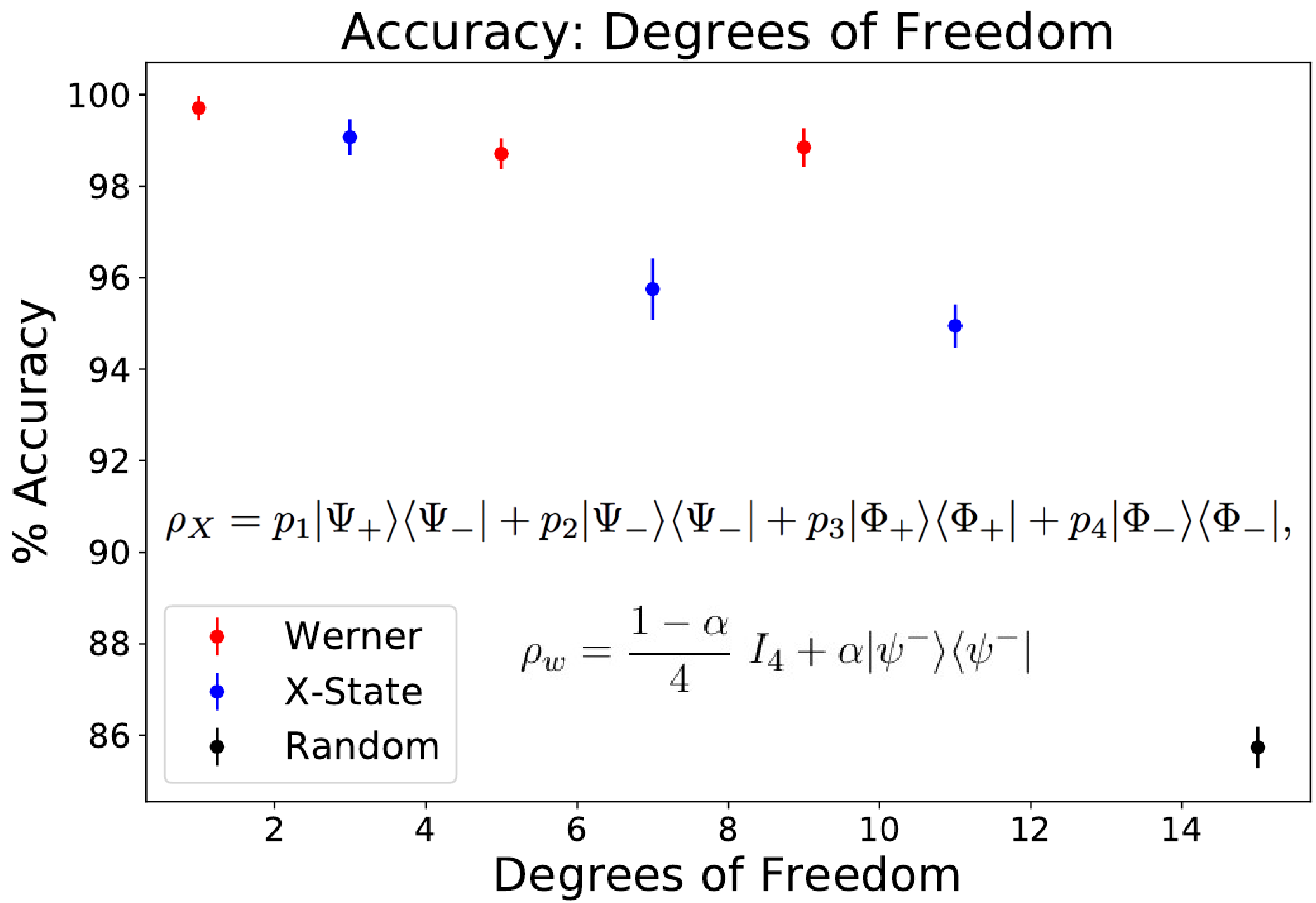


Figure 16: The average classification accuracy for each kind of state with a standard deviation error. Each point is an average over 100 independent training iterations. The states have been identified using the degrees of freedom (e.g. the 3 red points represent unrotated Werner states, single rotated Werner states, and double rotated Werner states).

[D Craig, Master thesis]

Deep Neural Network Probabilistic Decoder for Stabilizer Codes

SCIENTIFIC REPORTS | 7: 11003

Stefan Krastanov[1,2] & Liang Jiang[1,2]

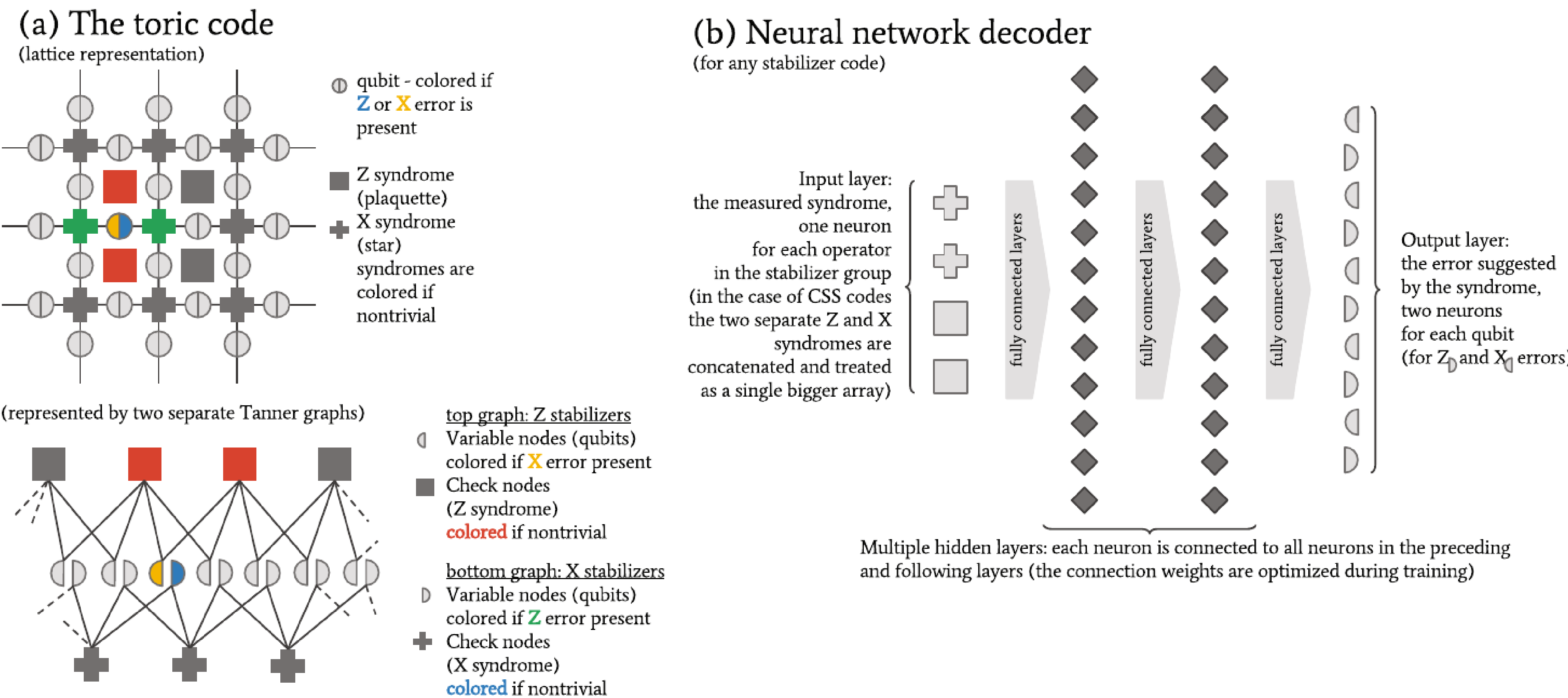


Figure 1. Quantum Error Correcting Codes. A very general class of QEC codes is the class stabilizer codes, defined by the stabilizer subgroup of the physical qubits that leaves the state of the logical qubits unperturbed. Our neural architecture can be readily applied to such codes, however many codes of practical interest (like the one we are testing against) have additional structure that would be interesting to consider. The example in (**a**) shows a small patch of a toric code, which is a CSS code (the stabilizer operators are products of only Z or only X operators, permitting us to talk of Z and X syndromes separately). Moreover, the toric code possesses a lattice structure that provides for a variety of decoders designed to exploit that structure. Our decoder, depicted in (**b**), does not have built-in knowledge of that structure, rather it learns it through training. Due to size constraints, the depictions present only a small subset of all qubit or syndrome nodes.

Deep Neural Network Probabilistic Decoder for Stabilizer Codes

SCIENTIFIC REPORTS | 7: 11003

Stefan Krastanov[1,2] & Liang Jiang[1,2]

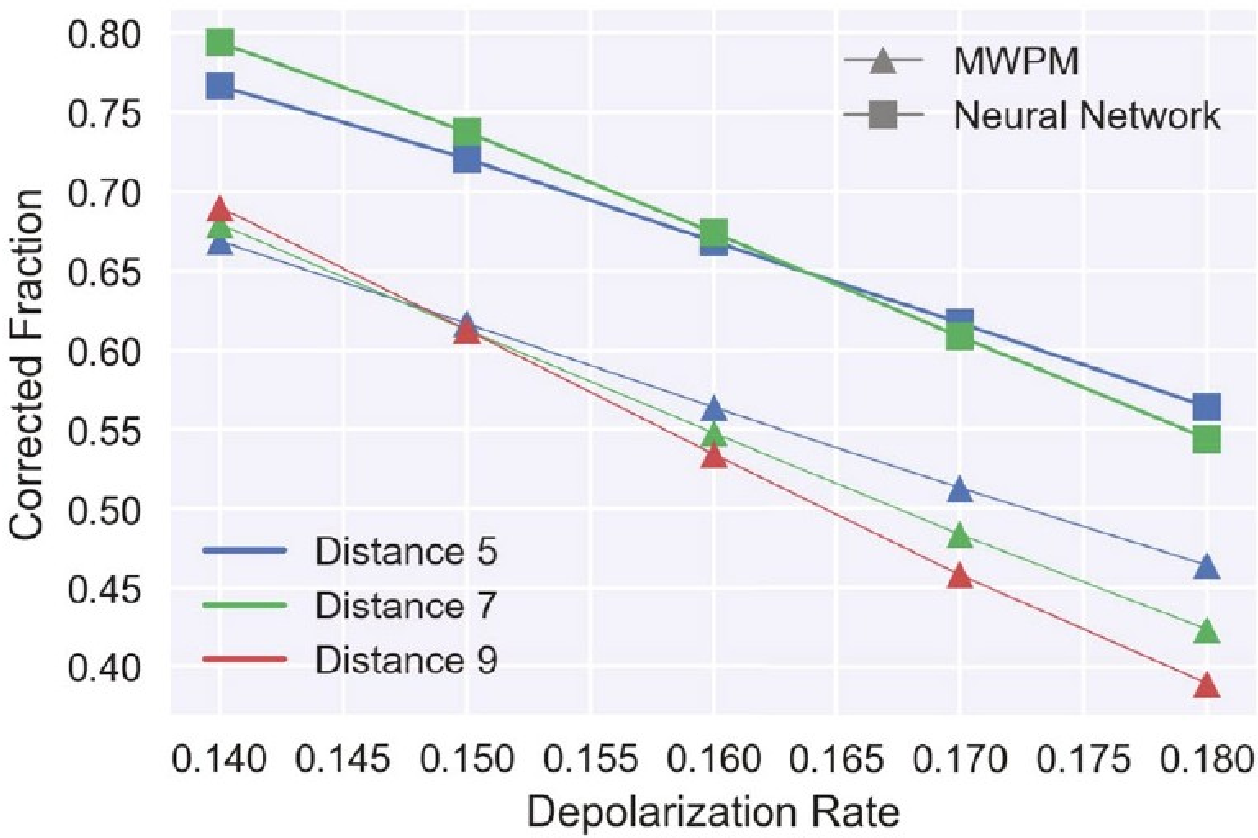


Figure 2. Decoder performance for toric codes of distances 5 and 7. The x axis is the depolarization rate of the physical qubits (the probability that an X, Y, or Z error has occurred), while the y axis is the fraction of properly decoded code iterations (the conjugate of the logical error rate). The neural network decoder (rectangular markers) significantly outperforms the minimal weight perfect matching decoder (triangular markers), both in terms of threshold and logical error rate. For the above plots, neural networks with 18 hidden layers were used.

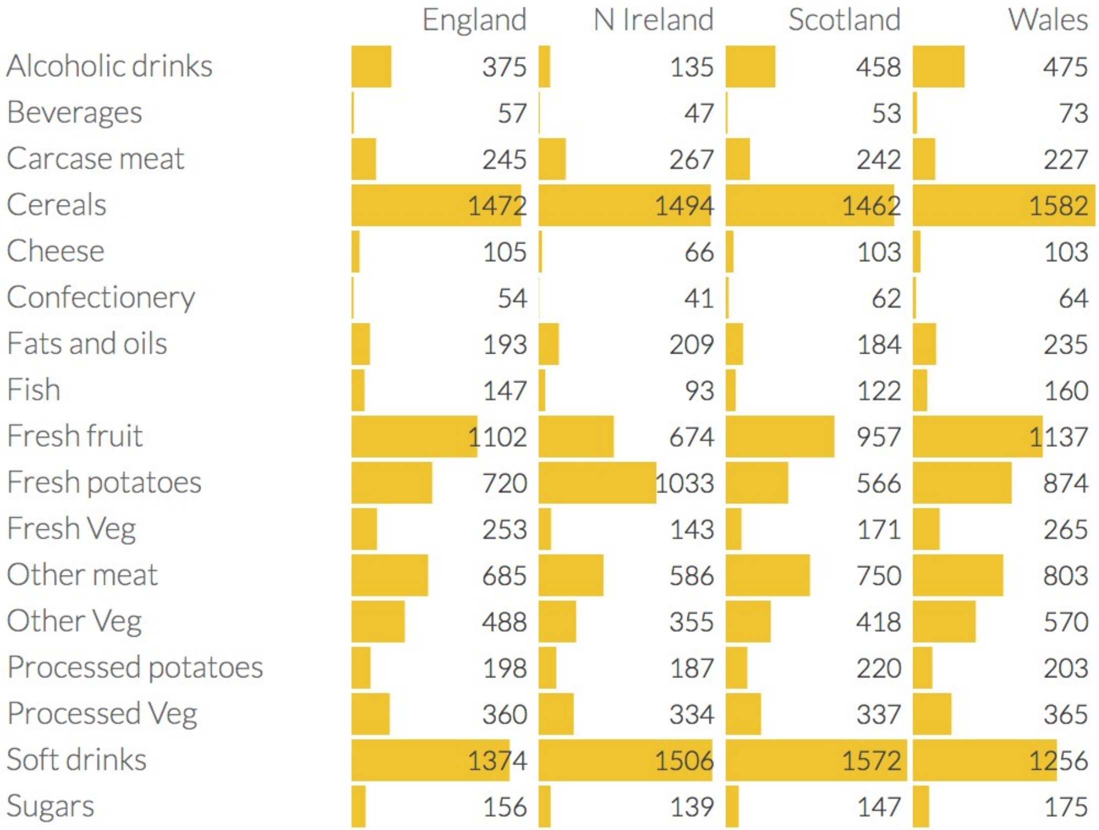
1.2 Unsupervised learning

Principal Component Analysis (PCA)

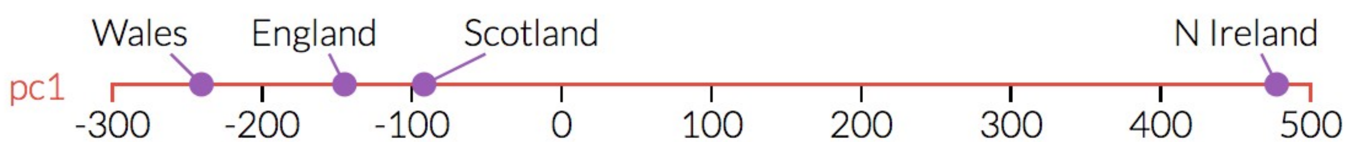
	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175

PCA for the average food consumed (in grams) per person per week for each country in the UK

Principal Component Analysis (PCA)

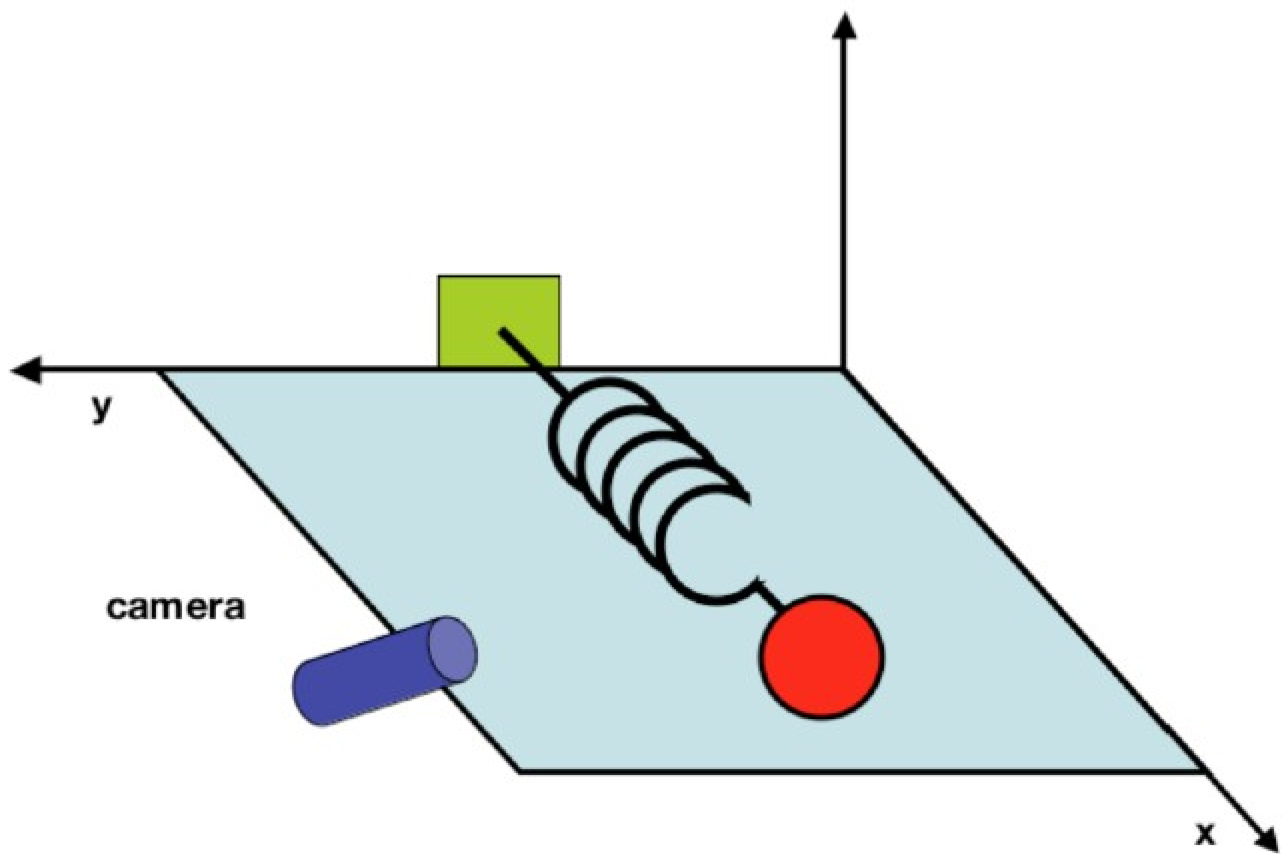


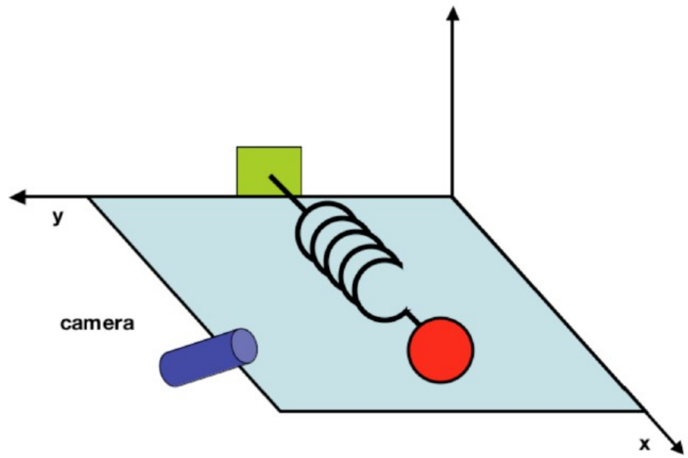
PCA for the average food consumed (in grams) per person per week for each country in the UK



1st principal component: in NI more potatoes, less fruit/cheese/alcohol

[http://setosa.io/ev/principal-component-analysis/]


y
camera
x



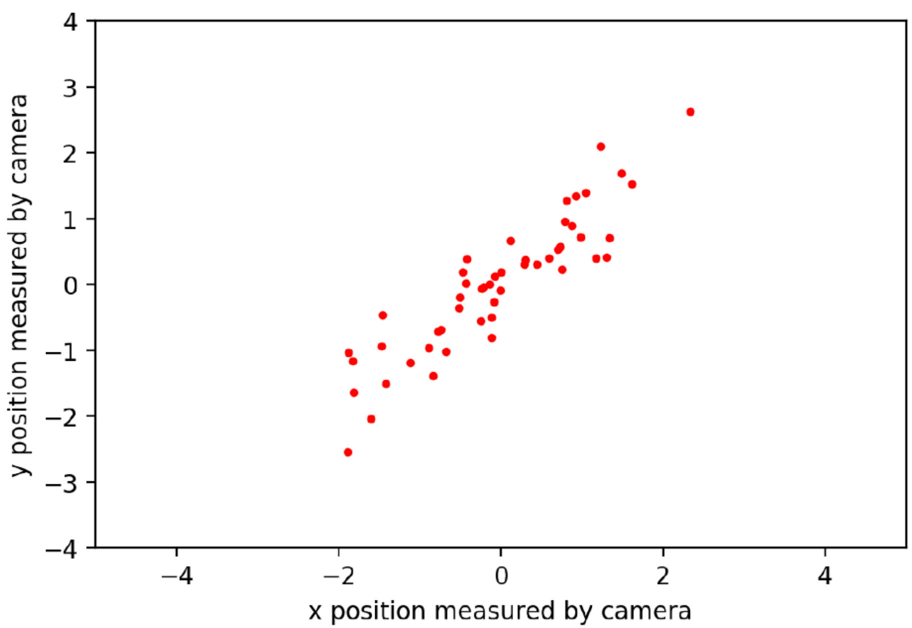


Figure 4: Position of ball measured by camera at random times

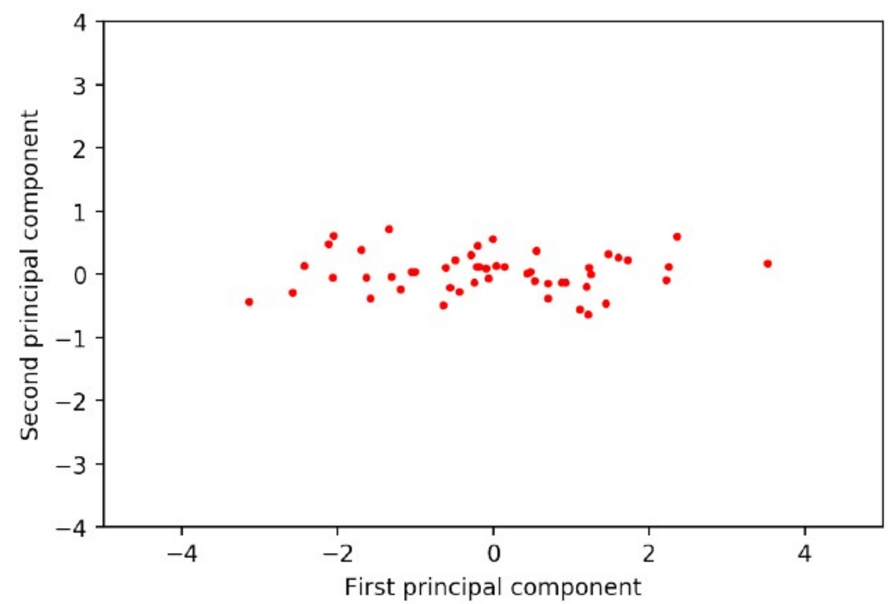
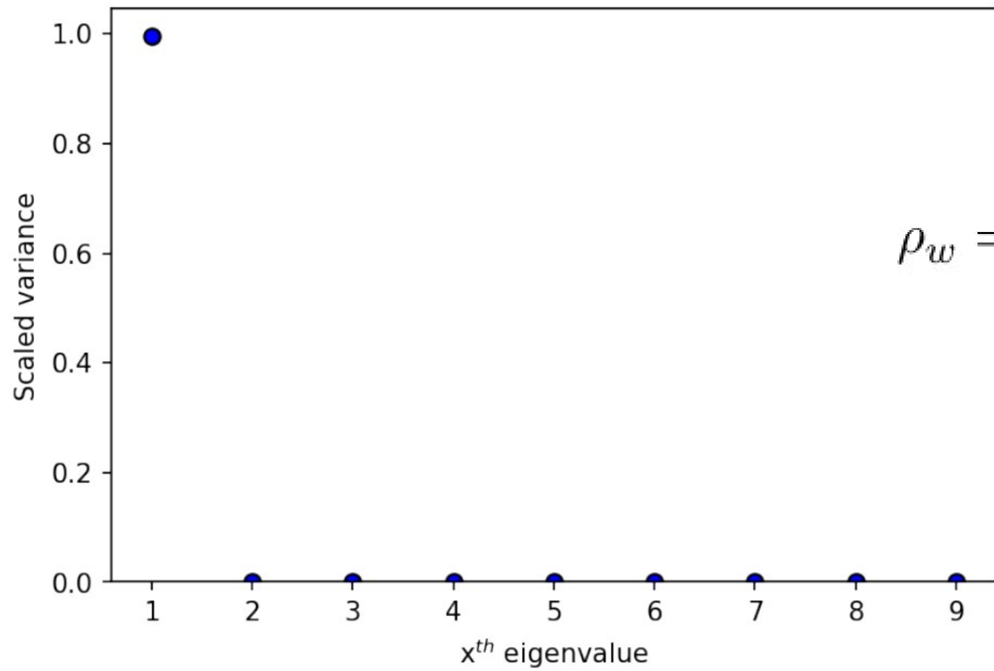


Figure 6: Data expressed using the principal components as basis vectors

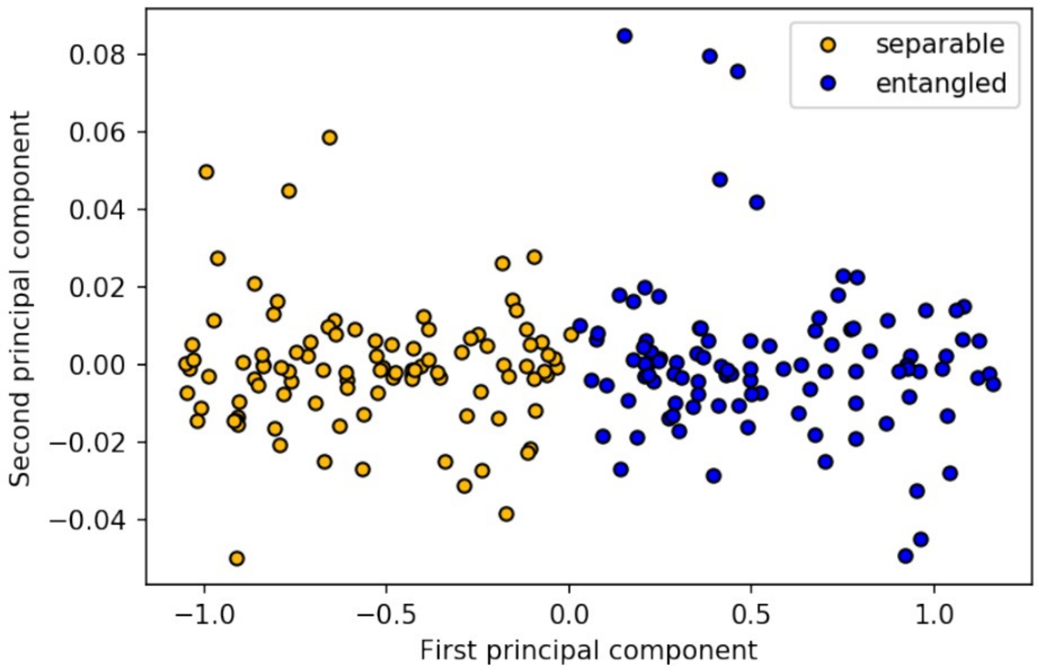
[J Shlens, arXiv:1404.1100]

[R McGibbon, Master thesis]



$$\rho_w = \frac{1-\alpha}{4}\ I_4 + \alpha|\psi^-\rangle\langle\psi^-|$$

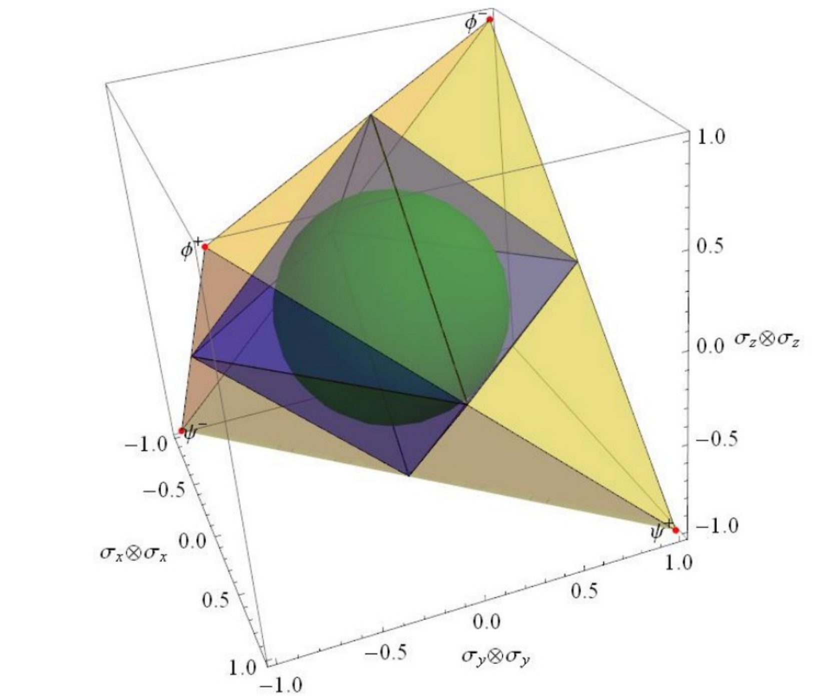
Eigenvalues from PCA carried out on data set of 200 rotated Werner states



This plot shows 200 Werner states. The axes are the first two principal components from the data set of reduced Fano vectors of the states [R McGibbon, Master thesis]

X states – convex combination of Bell states

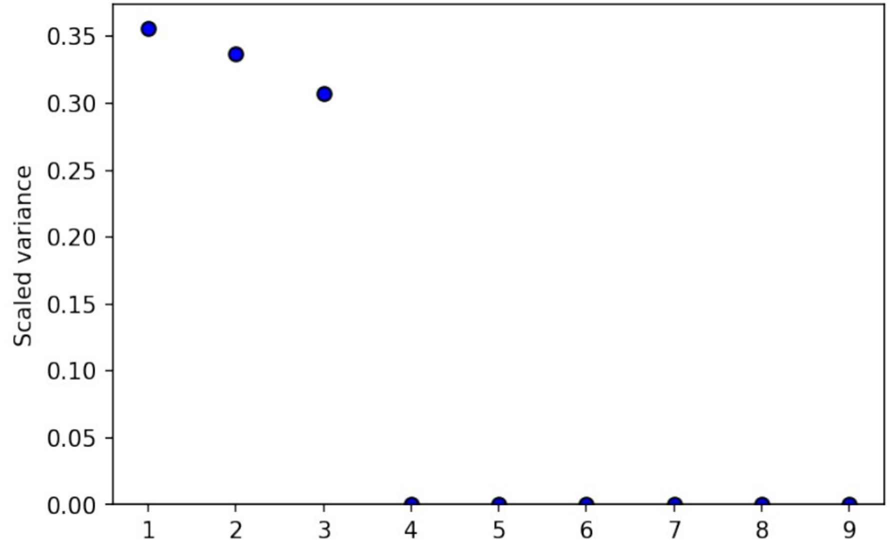
$$\rho_X = p_1|\Psi_+\rangle\langle\Psi_-| + p_2|\Psi_-\rangle\langle\Psi_-| + p_3|\Phi_+\rangle\langle\Phi_+| + p_4|\Phi_-\rangle\langle\Phi_-|, \qquad \sum_{i=1}^{4} p_i = 1$$



Fixed rotation on first and second qubit, hides structure

$$\rho = (U_A \otimes U_B)\rho_X(U_A \otimes U_B)^\dagger$$

[R McGibbon, Master thesis]



Eigenvalues from PCA carried out on data set of 1000 rotated X states

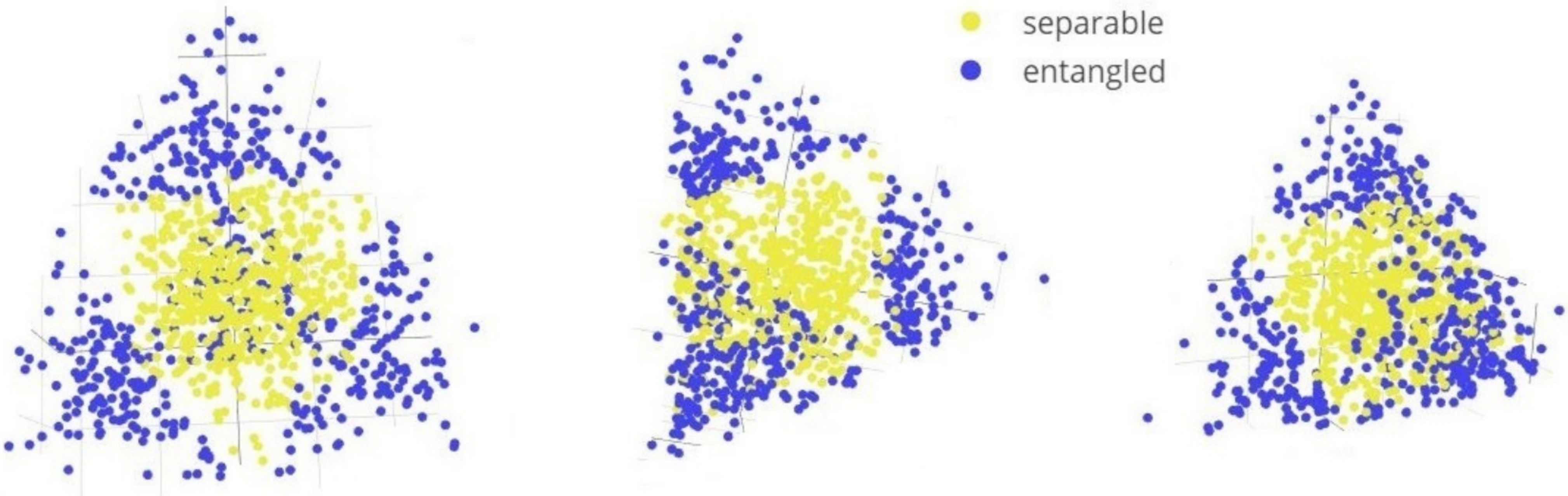


Figure 48: This plot shows 1000 X states from three different angles. The axes are the first three principal components from the data set of reduced Fano vectors of the states. Available to view at

https://plot.ly/~quantum_data_analysis/0/first-three-principal-components-of-x-states/#/

1.3 Reinforcement learning

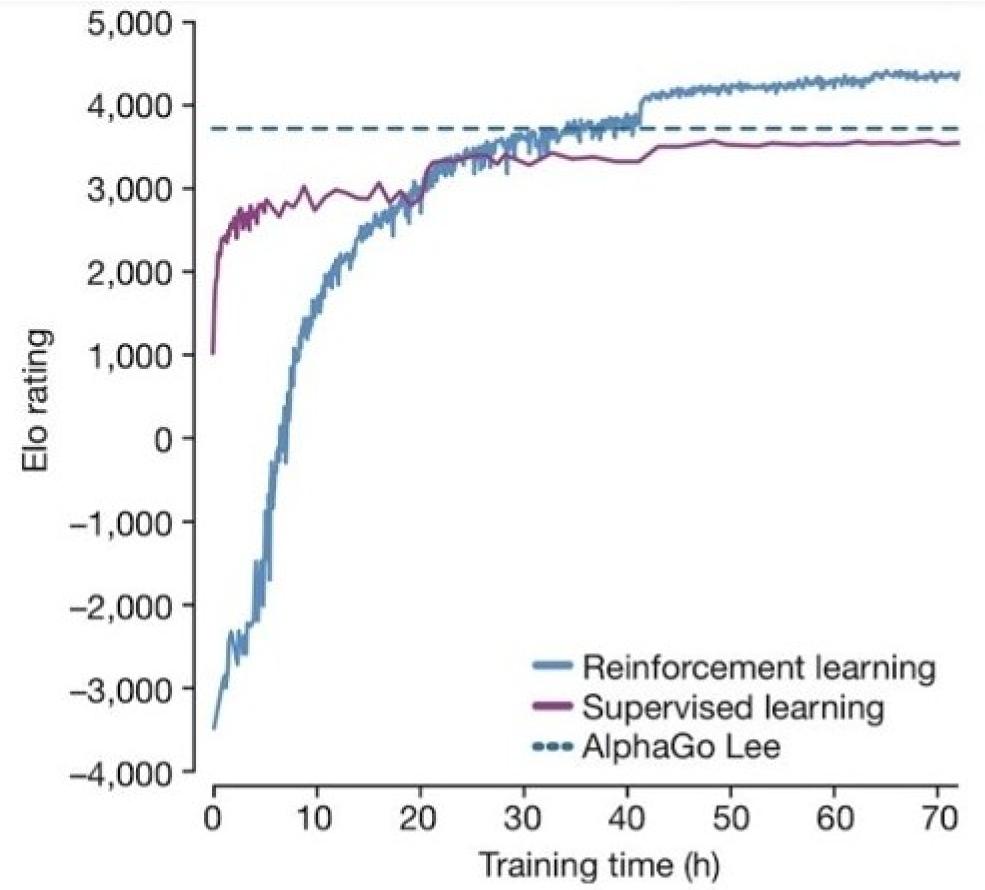
Alpha Go

- Played since 4th century BC
- 19 x 19 board
- The lower bound on the number of legal board positions in Go has been estimated to be 2×10^{170}

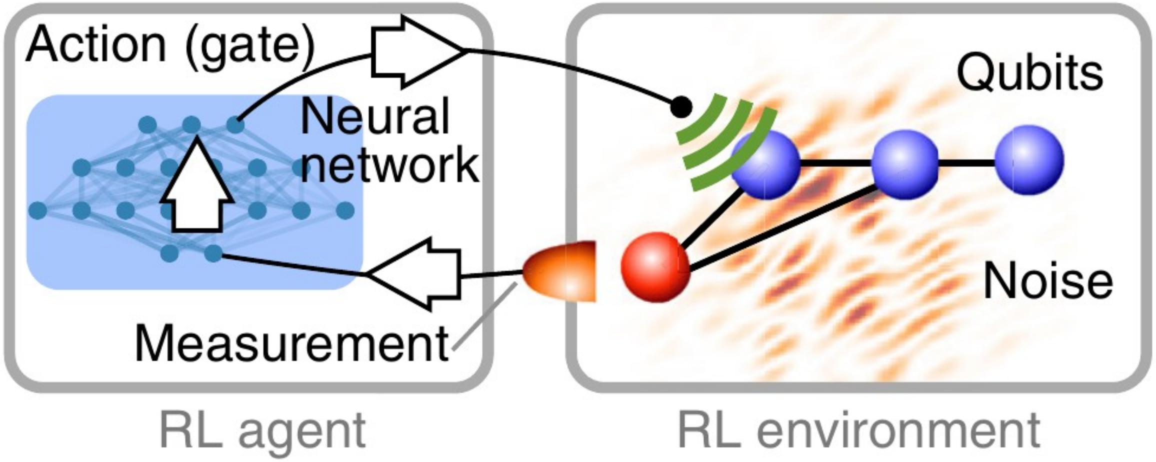
Expert beaten for the first time by a program in 2015.

Training: SL (learning from experts) + policy-gradient RL + ...

Alpha Go Zero

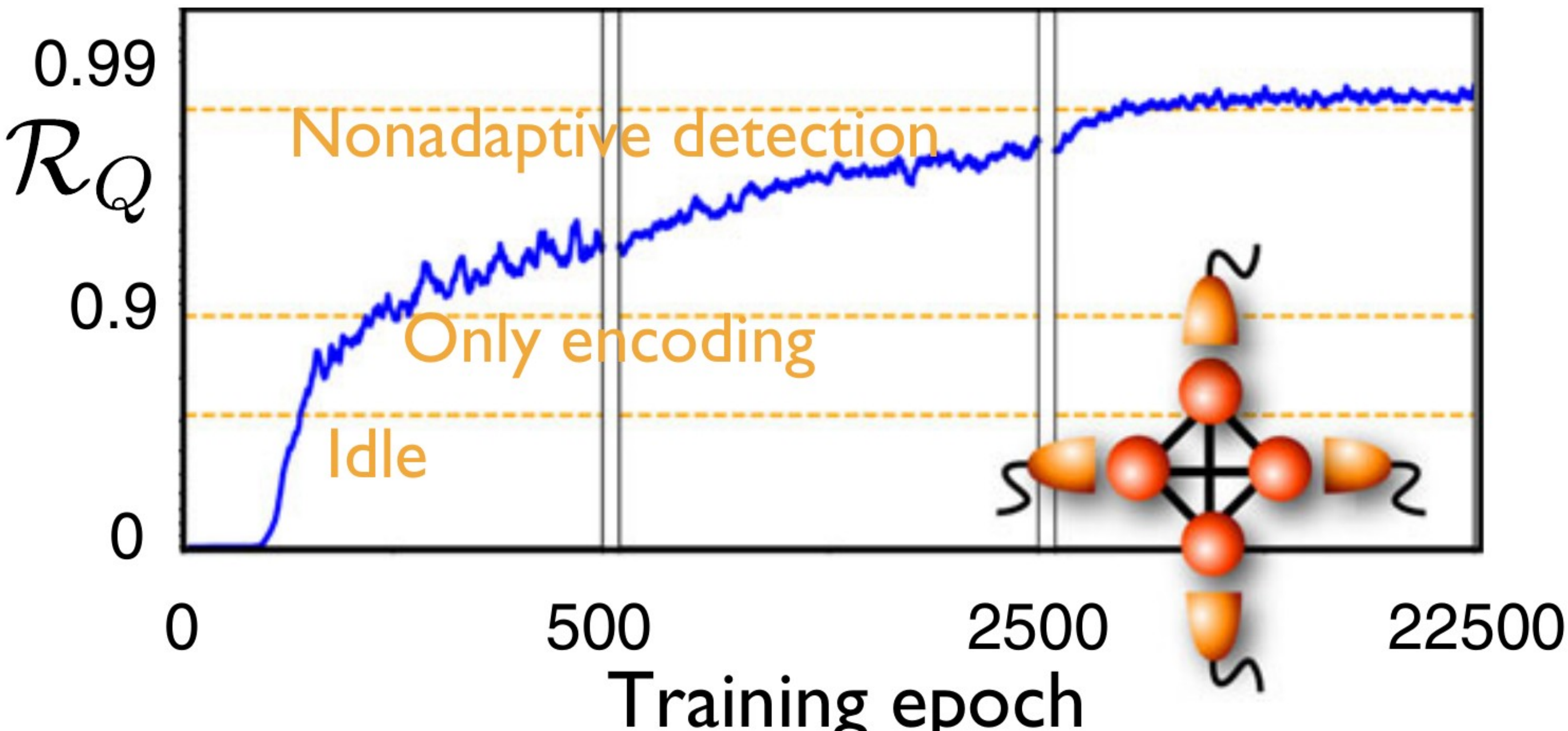


No initial supervised learning training using games played by experts
[Silver et al, Nature 2017]



Fosel et al, PRX 8, 031084 (2018)

(a) Recoverable quantum information



Outline

Part 1 – Overview of ML methods and applications to QT

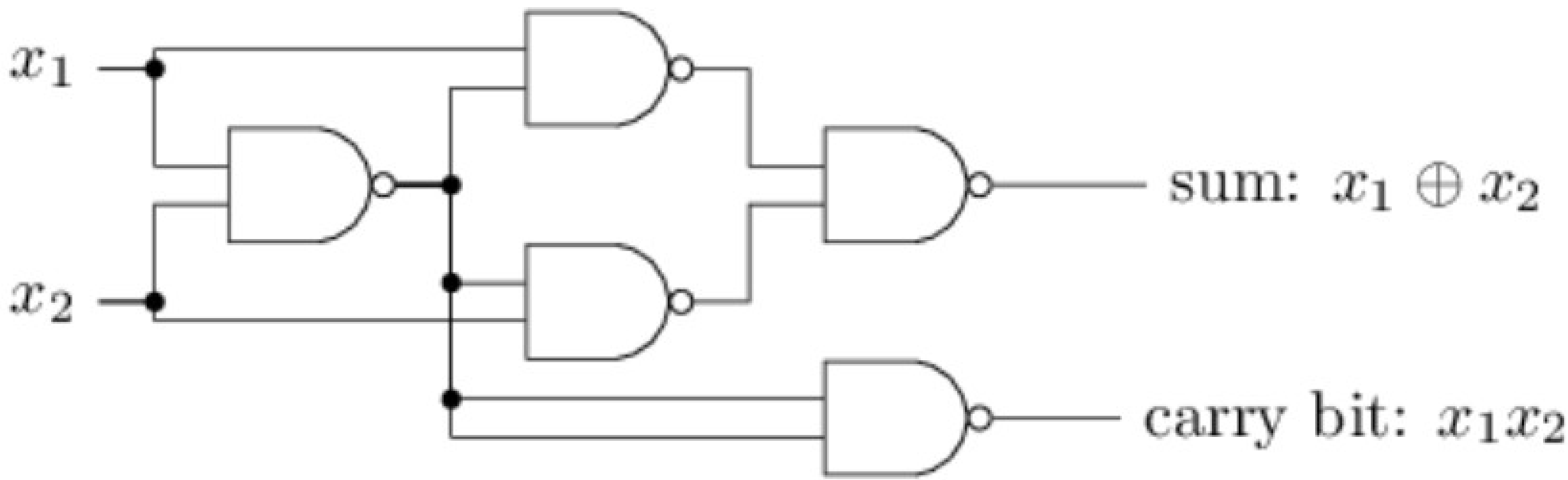
1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

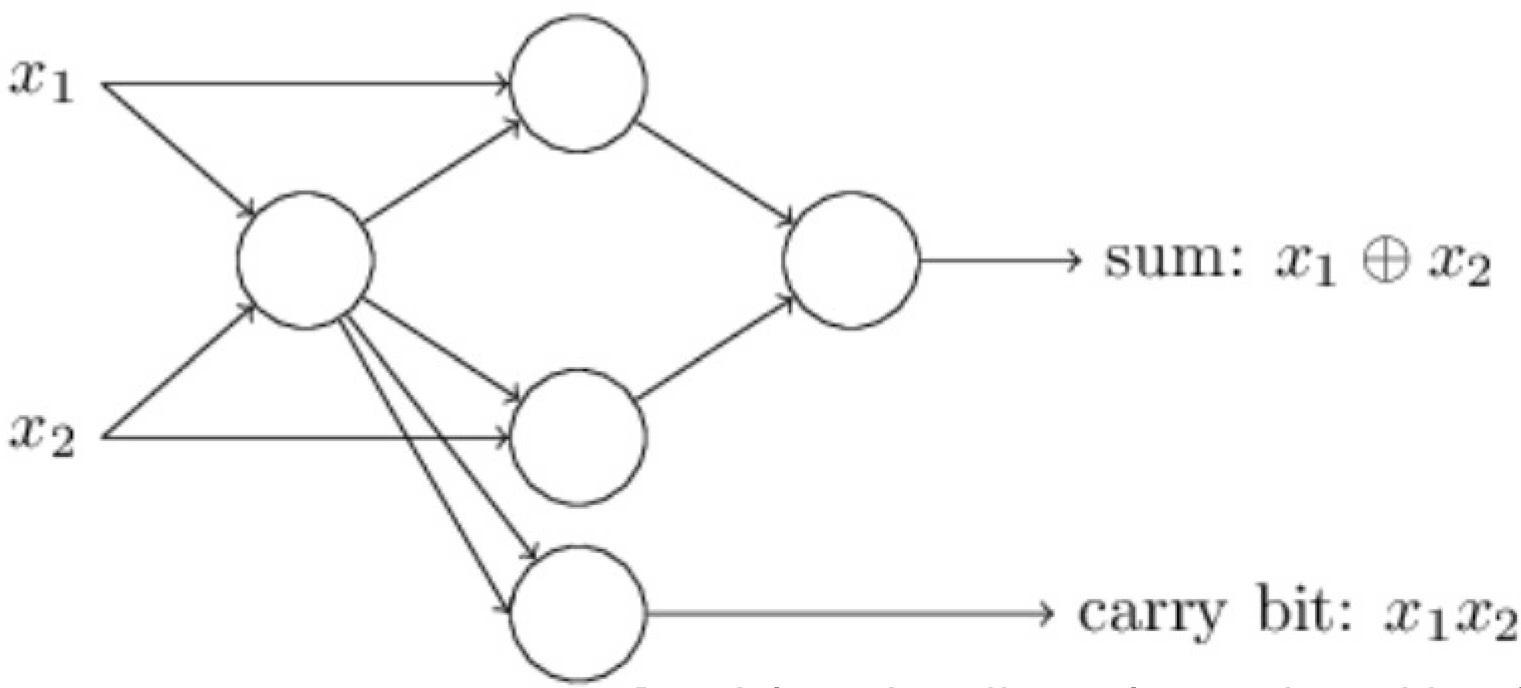
Part 2 – Introduction to Artificial Neural Networks

1. Perceptron
2. Sigmoid neurons
3. Example: handwritten digits classification
4. Stochastic gradient descend
5. A glimpse of the back-propagation algorithm
6. Overfitting
7. Convolutional neural networks

2. Introduction to artificial neural networks [M Nielsen, Neural networks and deep learning]

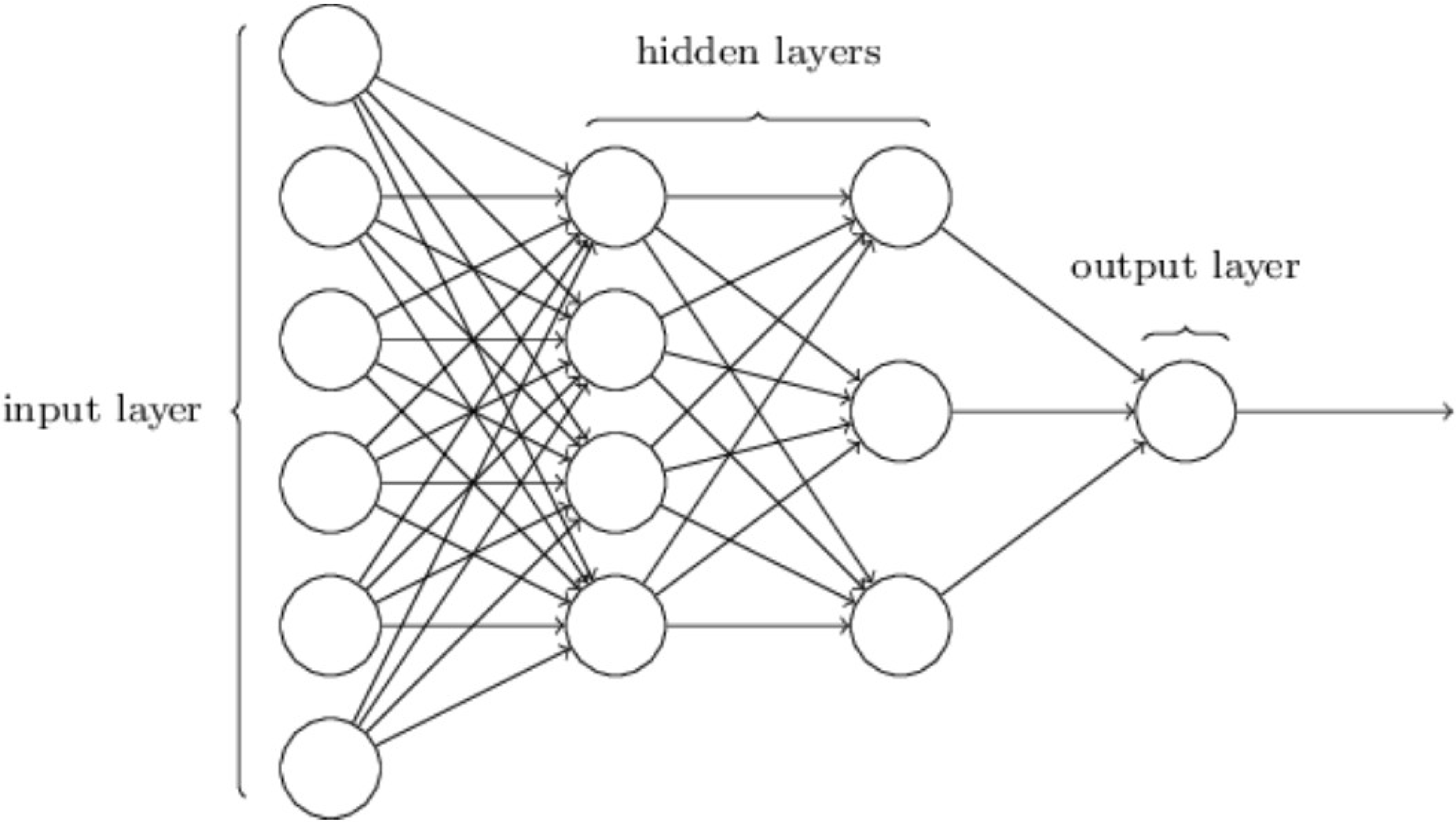
Example: Adder circuit – standard and perceptron





[M Nielsen, http://neuralnetworksanddeeplearning.com/]

General architecture of a feed-forward neural network



[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Stochastic Gradient descent – update rule

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits [Nielsen's code, 74 lines, GitHub]

Input neurons	784
Hidden neurons (1 layer only)	30
Output neurons	10
Training images (n)	50'000
Mini-batch size (m)	10
Learning rate	3
Epoch	30
Test images	10'000

```
Epoch 0: 9129 / 10000
Epoch 1: 9295 / 10000
Epoch 2: 9348 / 10000
...
Epoch 27: 9528 / 10000
Epoch 28: 9542 / 10000
Epoch 29: 9534 / 10000
```

Final accuracy: 95.34%

[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits: slower learning rate

Input neurons	784
Hidden neurons (1 layer only)	100
Output neurons	10
Training images (n)	50’000
Mini-batch size (m)	10
Learning rate	**0.001**
Epoch	30
Test images	10’000

```
Epoch 0: 1139 / 10000
Epoch 1: 1136 / 10000
Epoch 2: 1135 / 10000
...
Epoch 27: 2101 / 10000
Epoch 28: 2123 / 10000
Epoch 29: 2142 / 10000
```

Final accuracy: 21.41% (increasing)

[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits: faster learning rate

Input neurons	784
Hidden neurons (1 layer only)	100
Output neurons	10
Training images (n)	50'000
Mini-batch size (m)	10
Learning rate	**100**
Epoch	30
Test images	10'000

```
Epoch 0: 1009 / 10000
Epoch 1: 1009 / 10000
...
Epoch 27: 982 / 10000
Epoch 28: 982 / 10000
Epoch 29: 982 / 10000
```

Final accuracy: 9.82%
(random)

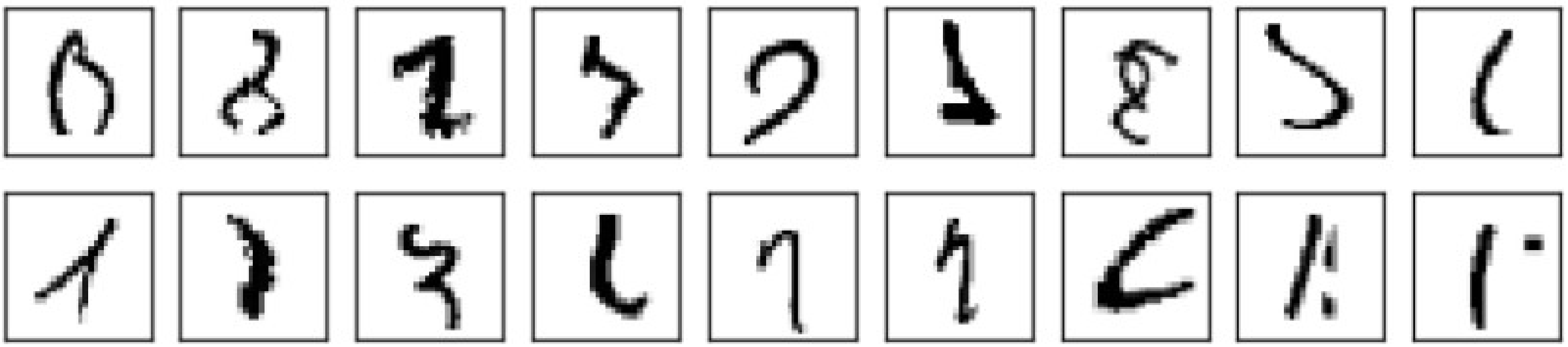
[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits: high performance codes

Accuracy: 99.979% [Wan et al. (2013)]

All images are correctly classified, except 21.

Examples of incorrectly classified images:



[M Nielsen, http://neuralnetworksanddeeplearning.com/]

2.5 – A glimpse of the back-propagation algorithm

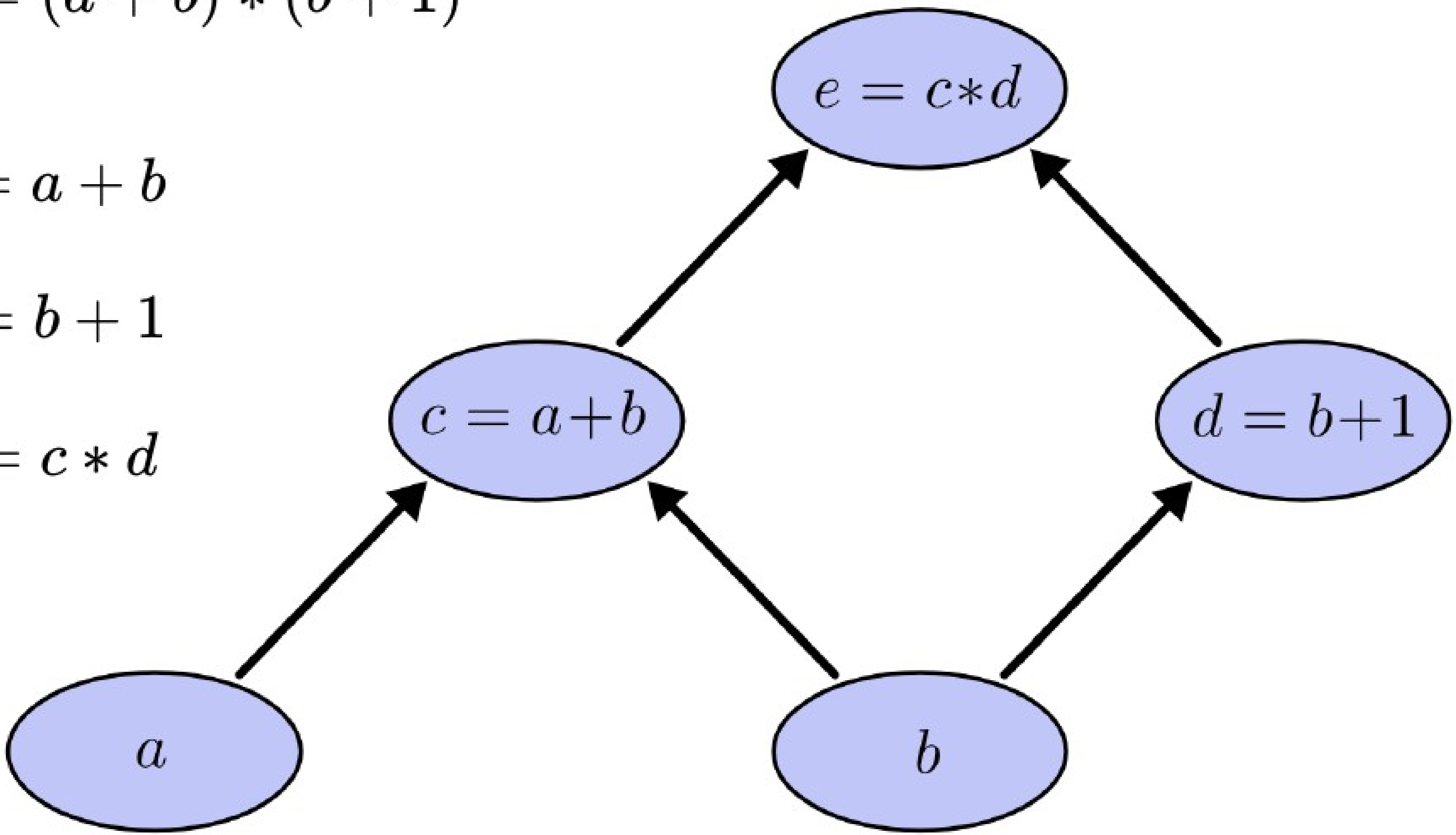
Computational graphs

$$e = (a + b) * (b + 1)$$

$$c = a + b$$

$$d = b + 1$$

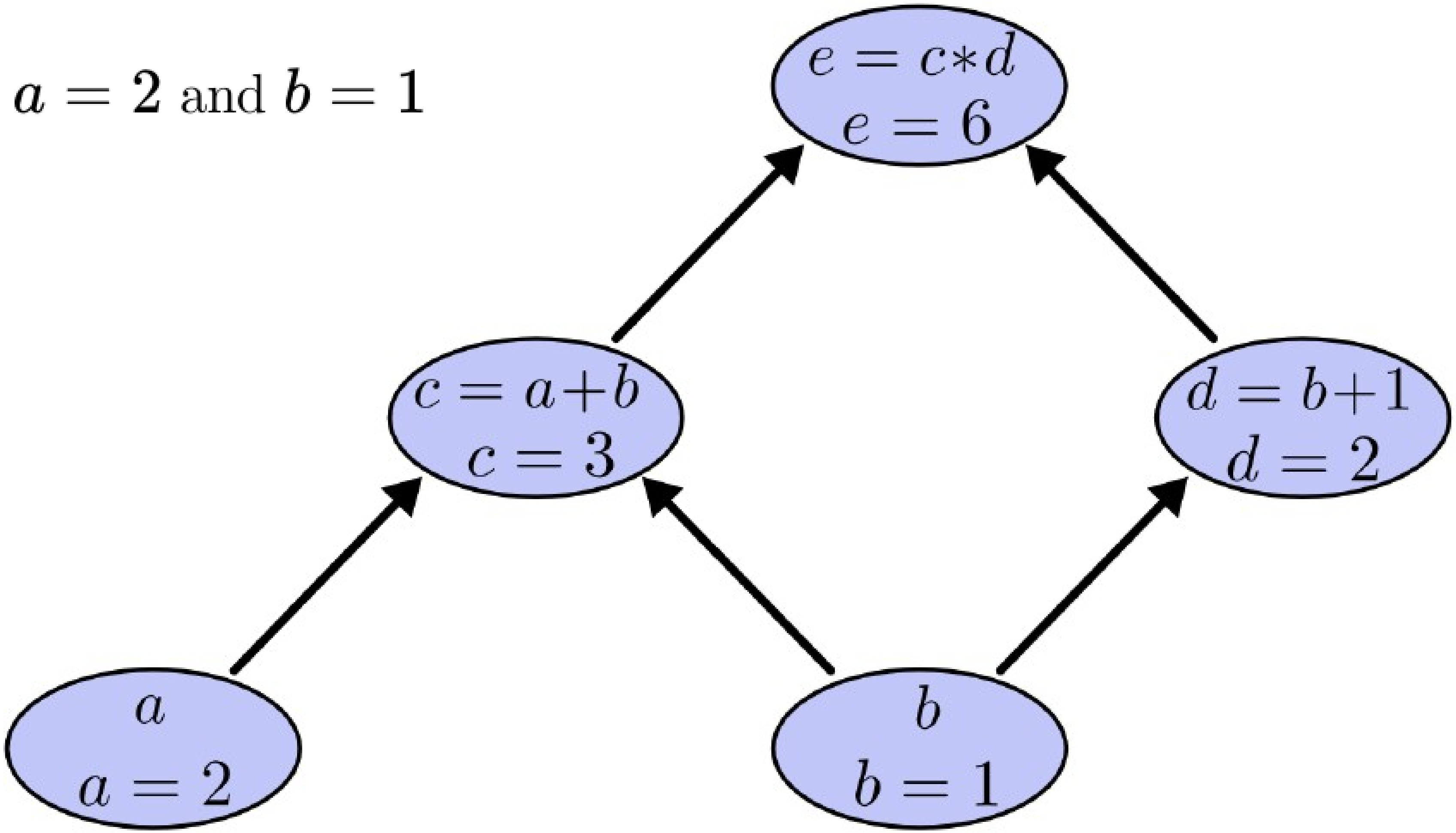
$$e = c * d$$



http://colah.github.io

Arithmetic on computational graphs

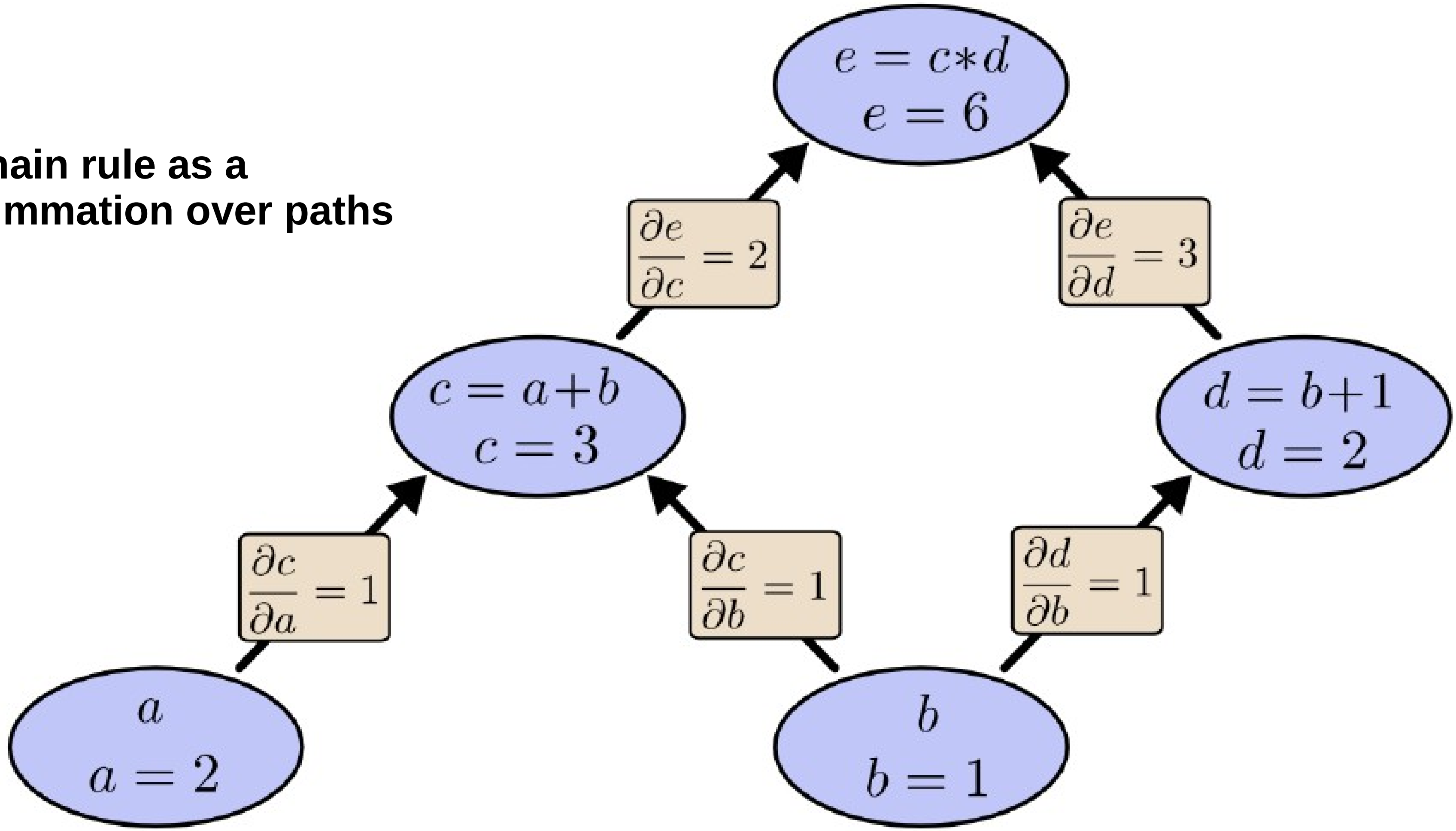
$a = 2$ and $b = 1$



http://colah.github.io

Derivatives on computational graphs

Chain rule as a summation over paths

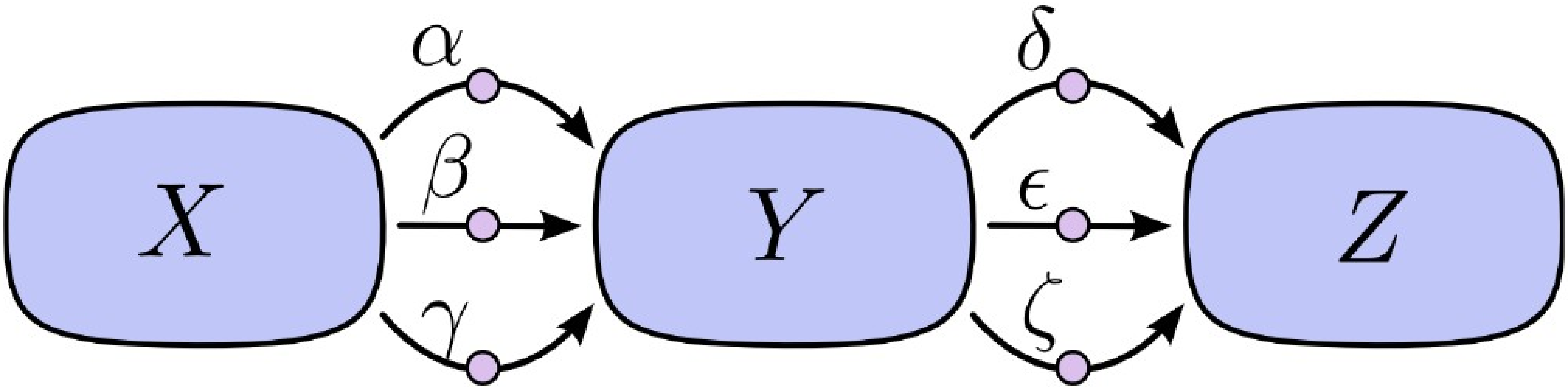


$$\frac{\partial e}{\partial a} = 1 * 2$$

$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3$$

http://colah.github.io

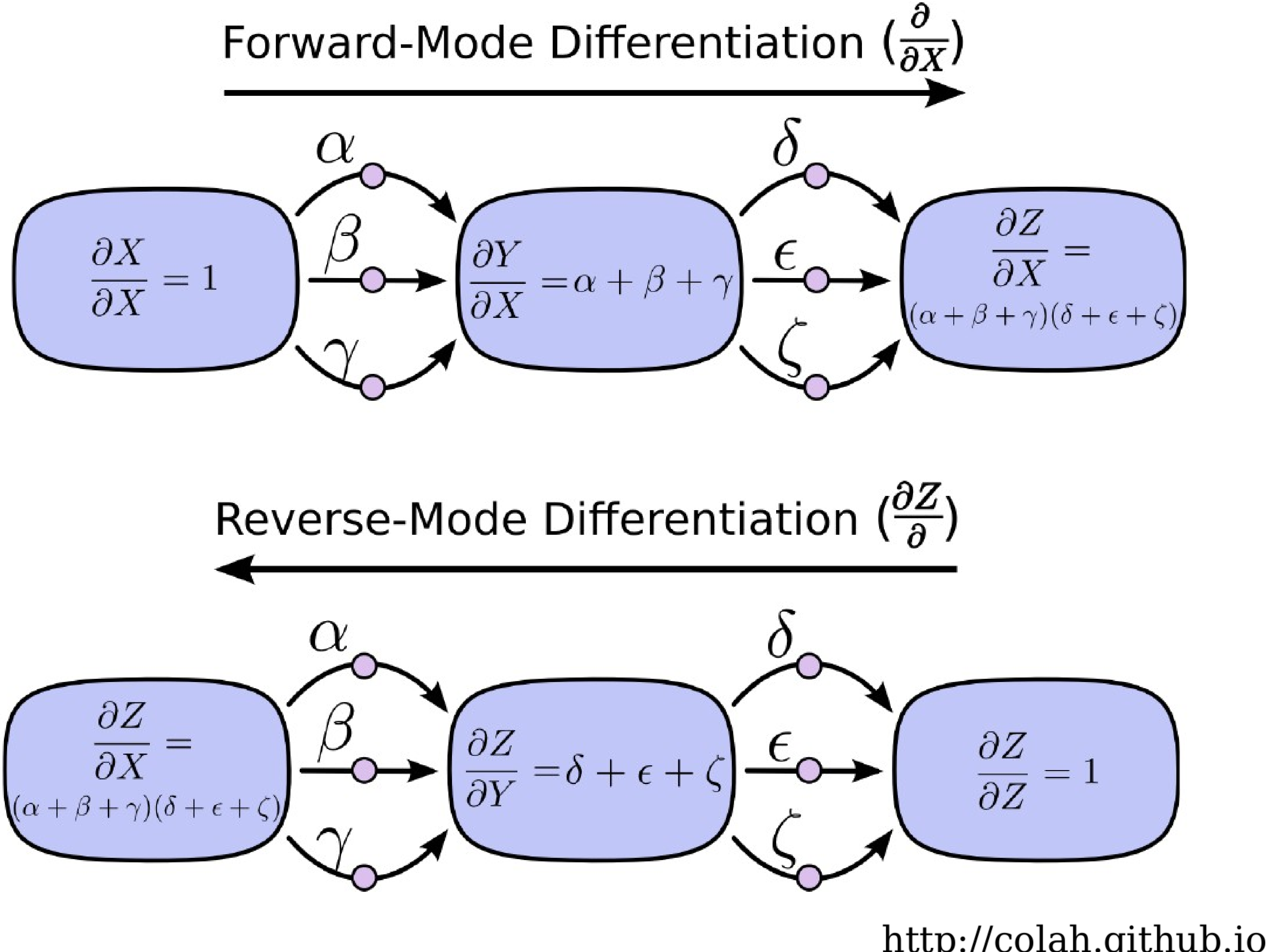
Factoring computational graph paths



$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$$

http://colah.github.io


Forward-Mode Differentiation ($\frac{\partial}{\partial X}$)
α
β
γ
δ
ϵ
ζ
$\frac{\partial X}{\partial X} = 1$
$\frac{\partial Y}{\partial X} = \alpha + \beta + \gamma$
$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$
Reverse-Mode Differentiation ($\frac{\partial Z}{\partial}$)
α
β
γ
δ
ϵ
ζ
$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$
$\frac{\partial Z}{\partial Y} = \delta + \epsilon + \zeta$
$\frac{\partial Z}{\partial Z} = 1$
http://colah.github.io

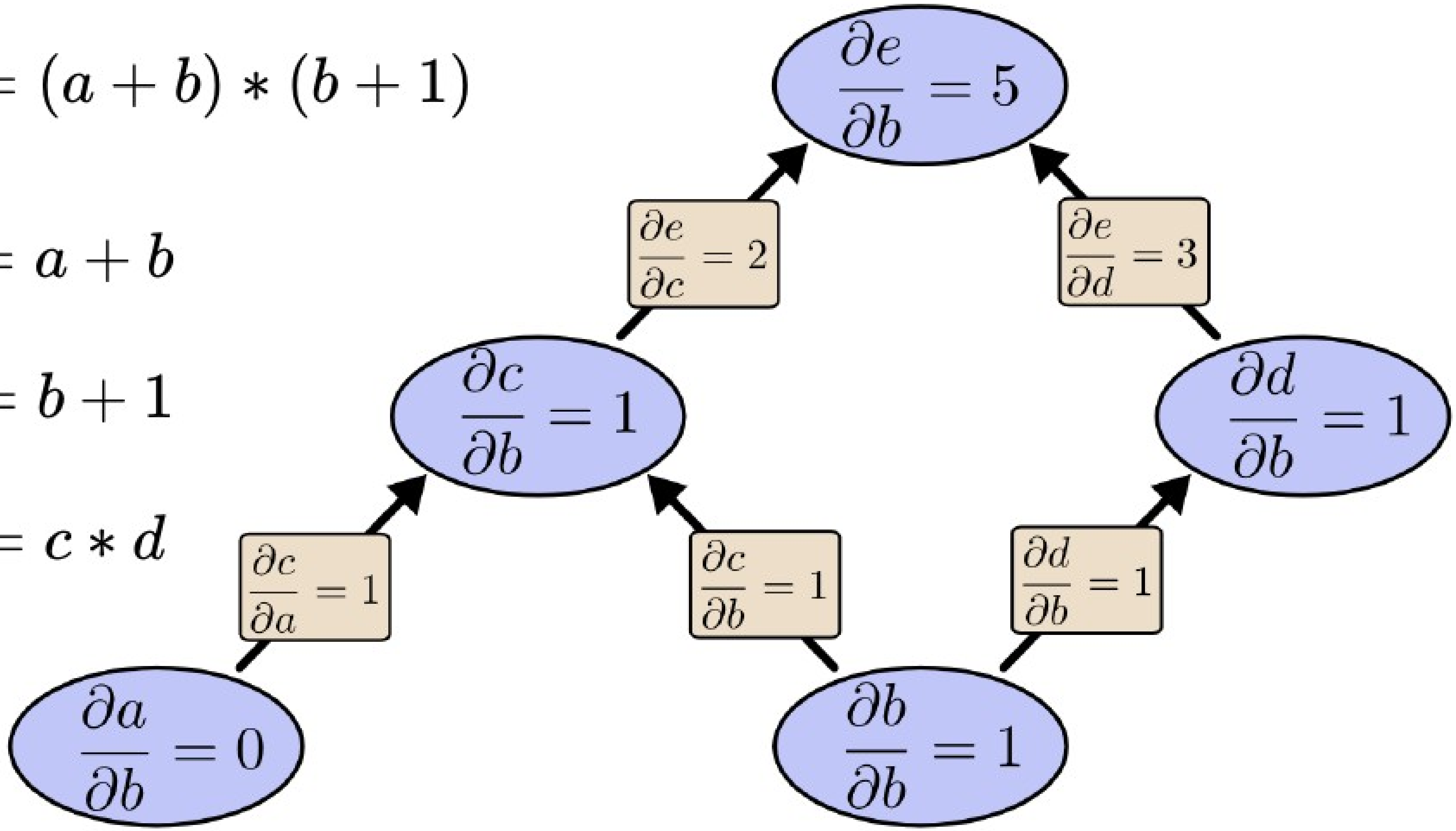
Example: forward-mode differentiation

$$e = (a+b)*(b+1)$$

$$c = a + b$$

$$d = b + 1$$

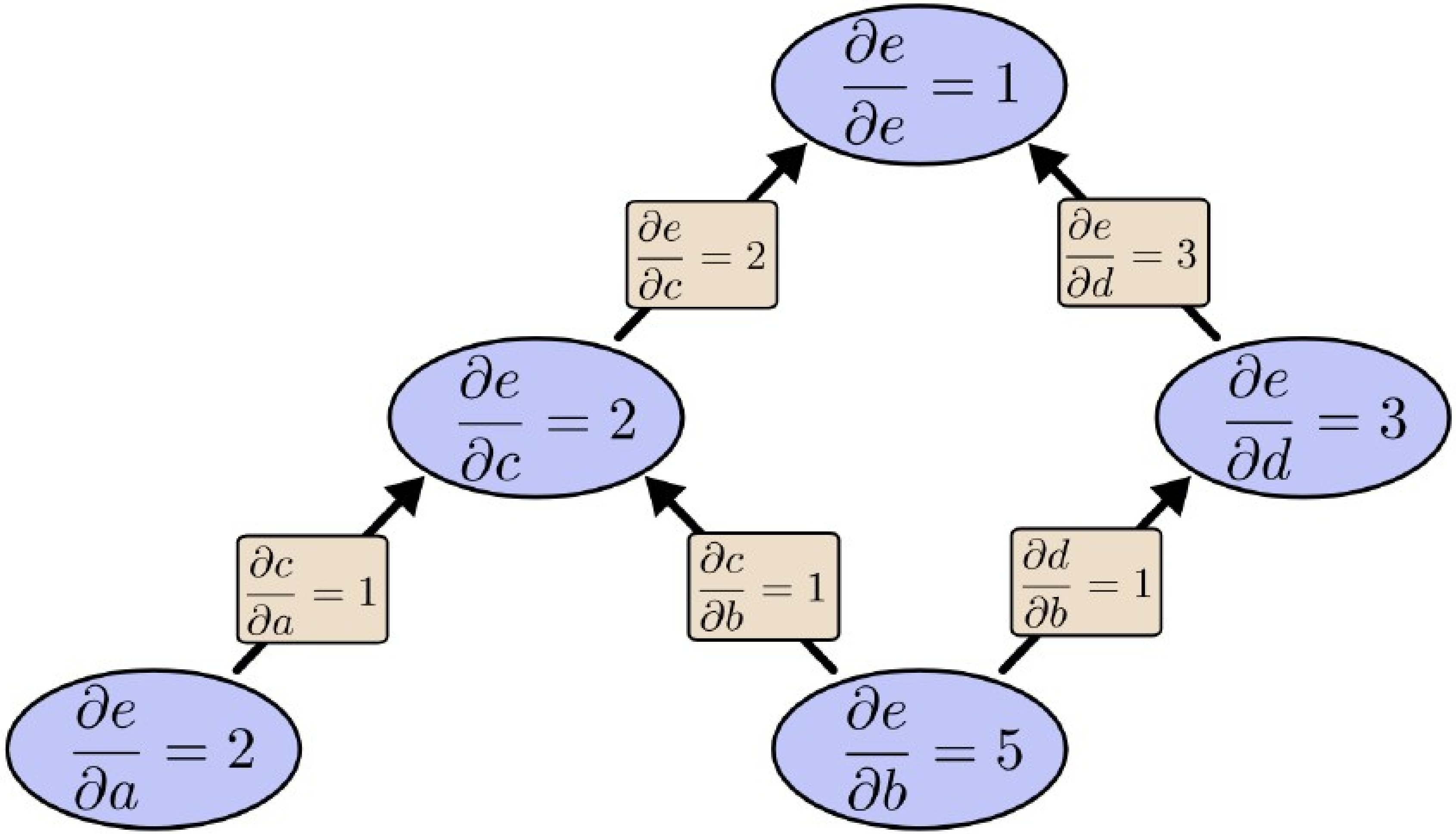
$$e = c * d$$



Derivative of the output with respect to a single input

http://colah.github.io

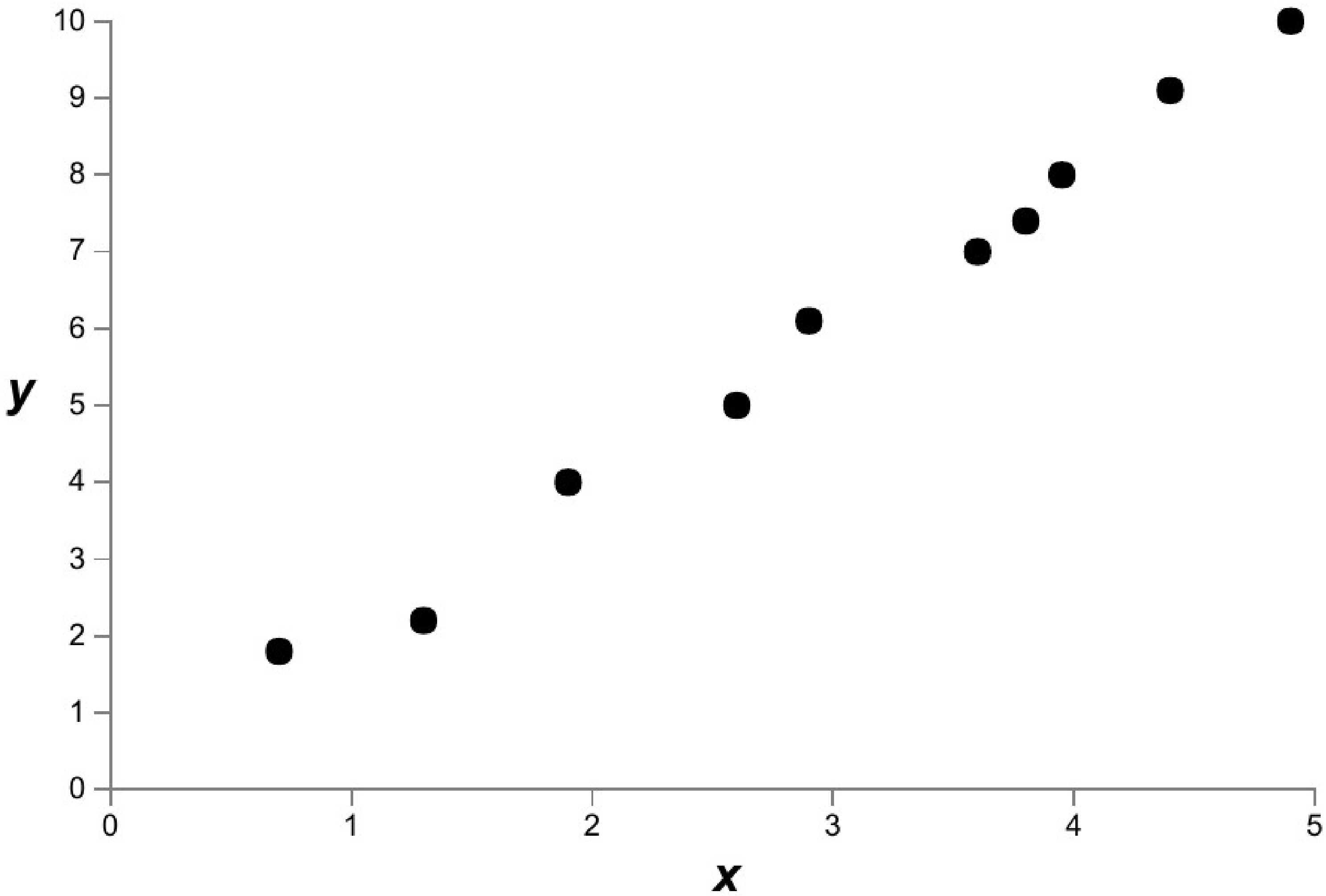
Example: backward-mode differentiation



Derivative of our output with respect to all input!
Very practical to calculate the partial derivatives
of the network cost function wrt all weights and biases.

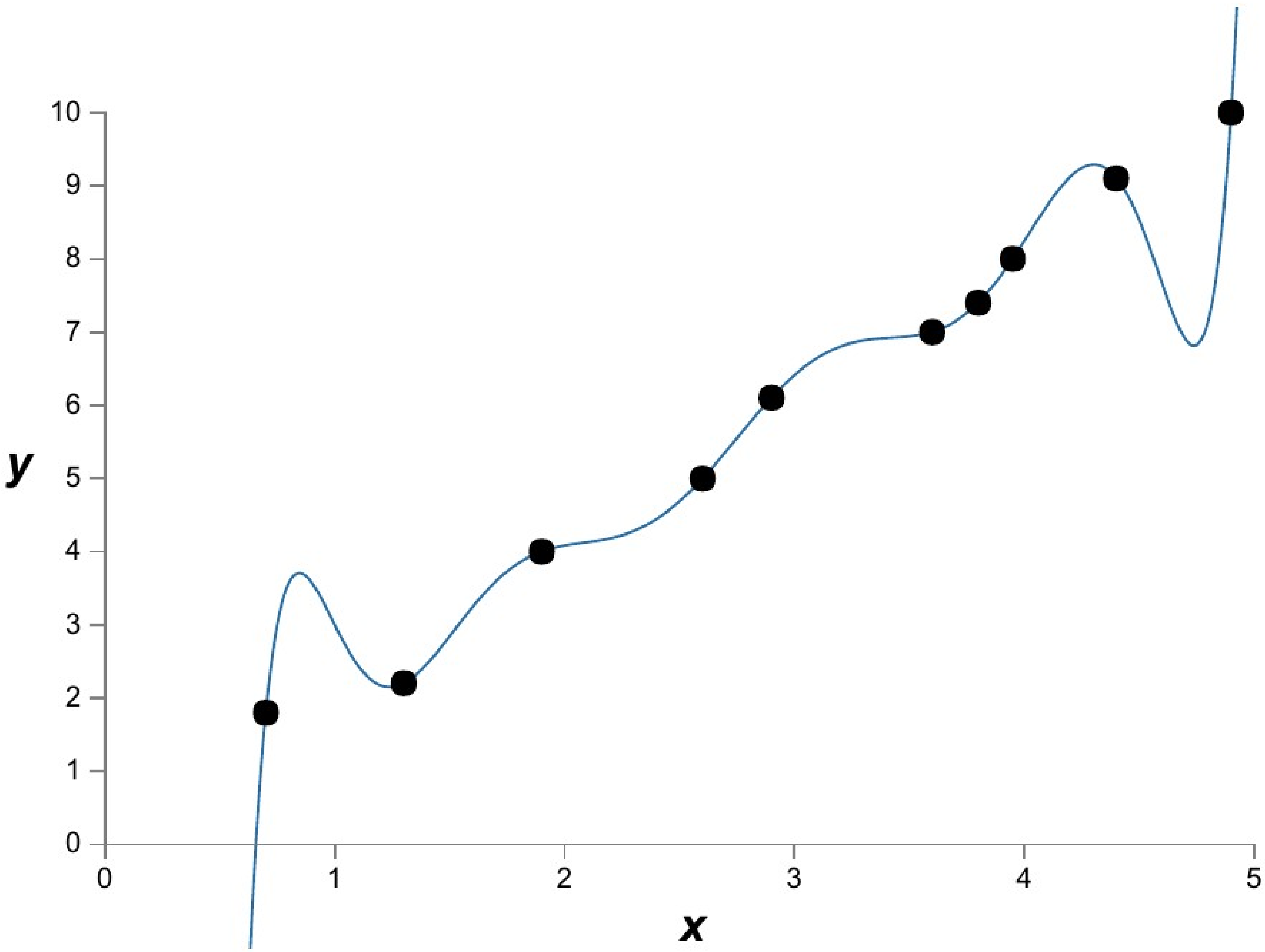
http://colah.github.io

2.6 – Overfitting



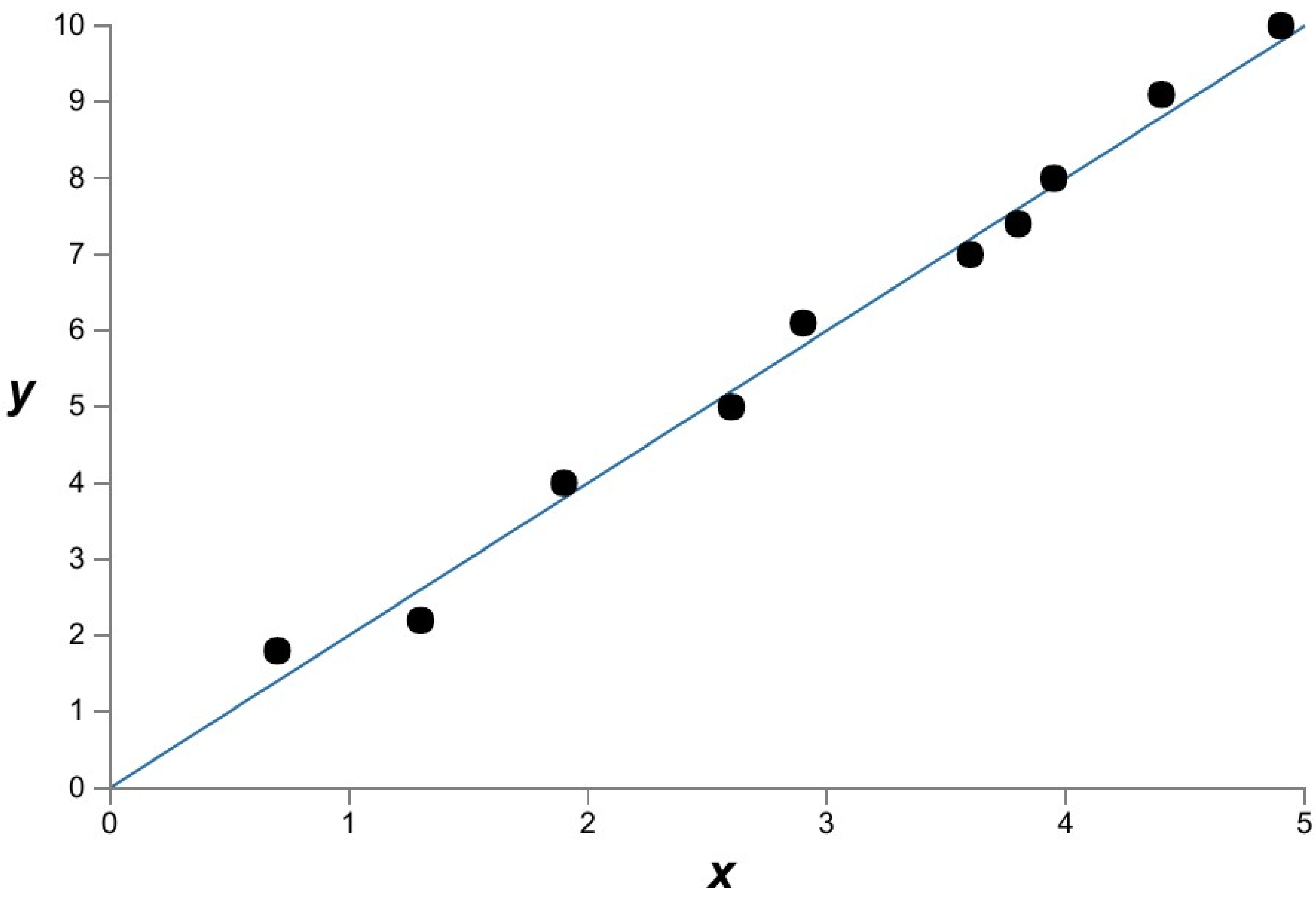
[M Nielsen, http://neuralnetworksanddeeplearning.com/]

2.6 – Overfitting



[M Nielsen, http://neuralnetworksanddeeplearning.com/]

2.6 – Overfitting



[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits

Input neurons	784
Hidden neurons (1 layer only)	30
Output neurons	10
Training images (n)	50'000
Mini-batch size (m)	10
Learning rate	3
Epoch	30
Test images	10'000

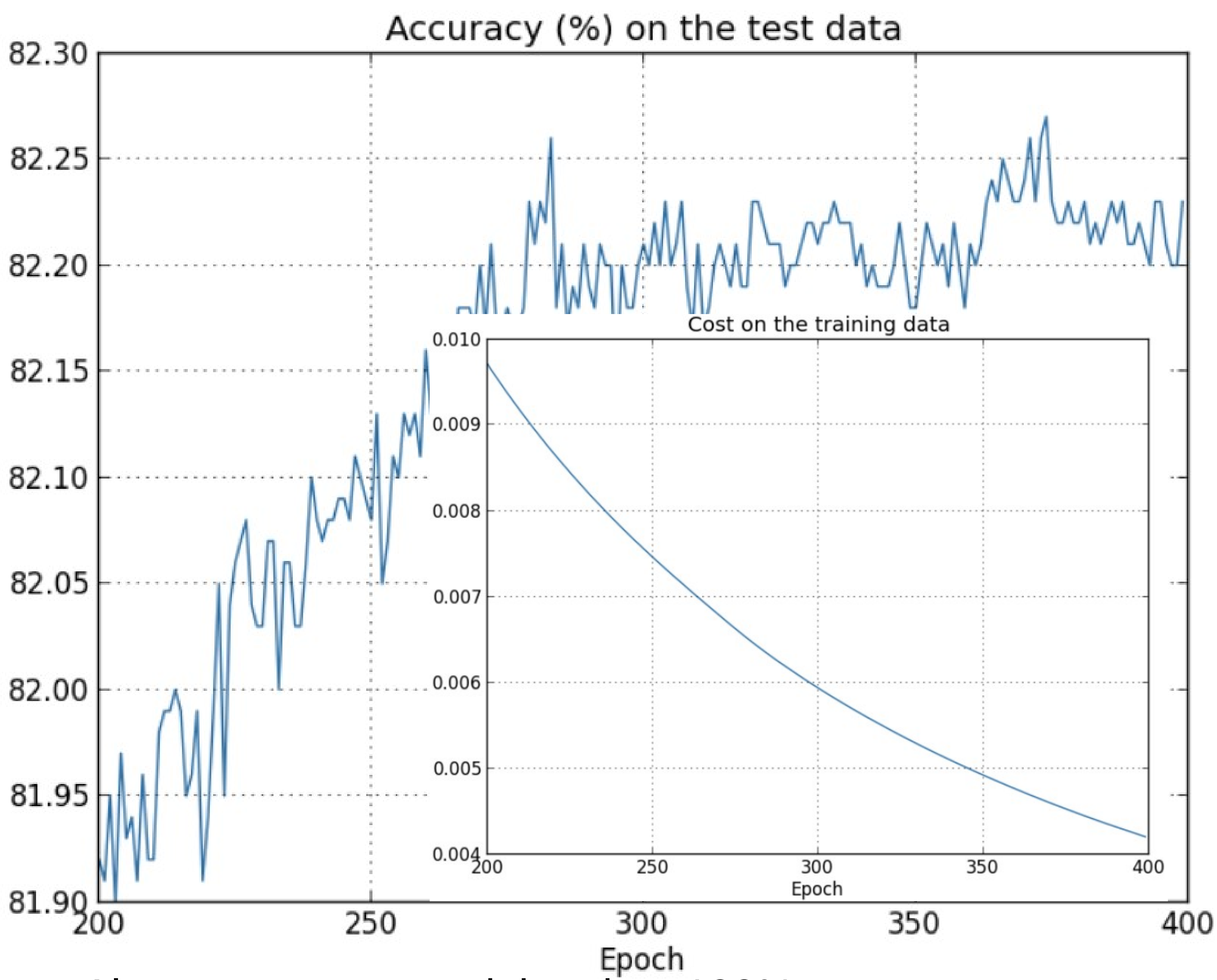
```
Epoch 0: 9129 / 10000
Epoch 1: 9295 / 10000
Epoch 2: 9348 / 10000
...
Epoch 27: 9528 / 10000
Epoch 28: 9542 / 10000
Epoch 29: 9534 / 10000
```

Final accuracy: 95.34%

[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits: small training set & many epochs

Input neurons	784
Hidden neurons (1 layer only)	30
Output neurons	10
Training images (n)	**1’000**
Mini-batch size (m)	10
Learning rate	0.5
Epoch	**400**
Test images	10’000



Also, accuracy on training data 100%

[M Nielsen, http://neuralnetworksanddeeplearning.com/]

Example of training a FFNN to classify hand-written digits: large training set

Input neurons	784
Hidden neurons (1 layer only)	30
Output neurons	10
Training images (n)	**50’000**
Mini-batch size (m)	10
Learning rate	0.5
Epoch	30
Test images	10’000

[M Nielsen, http://neuralnetworksanddeeplearning.com/]