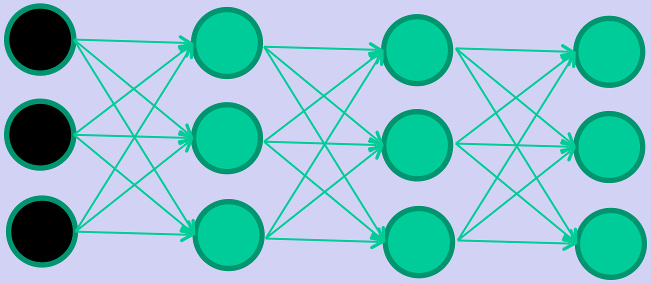
Introduction to Deep Learning

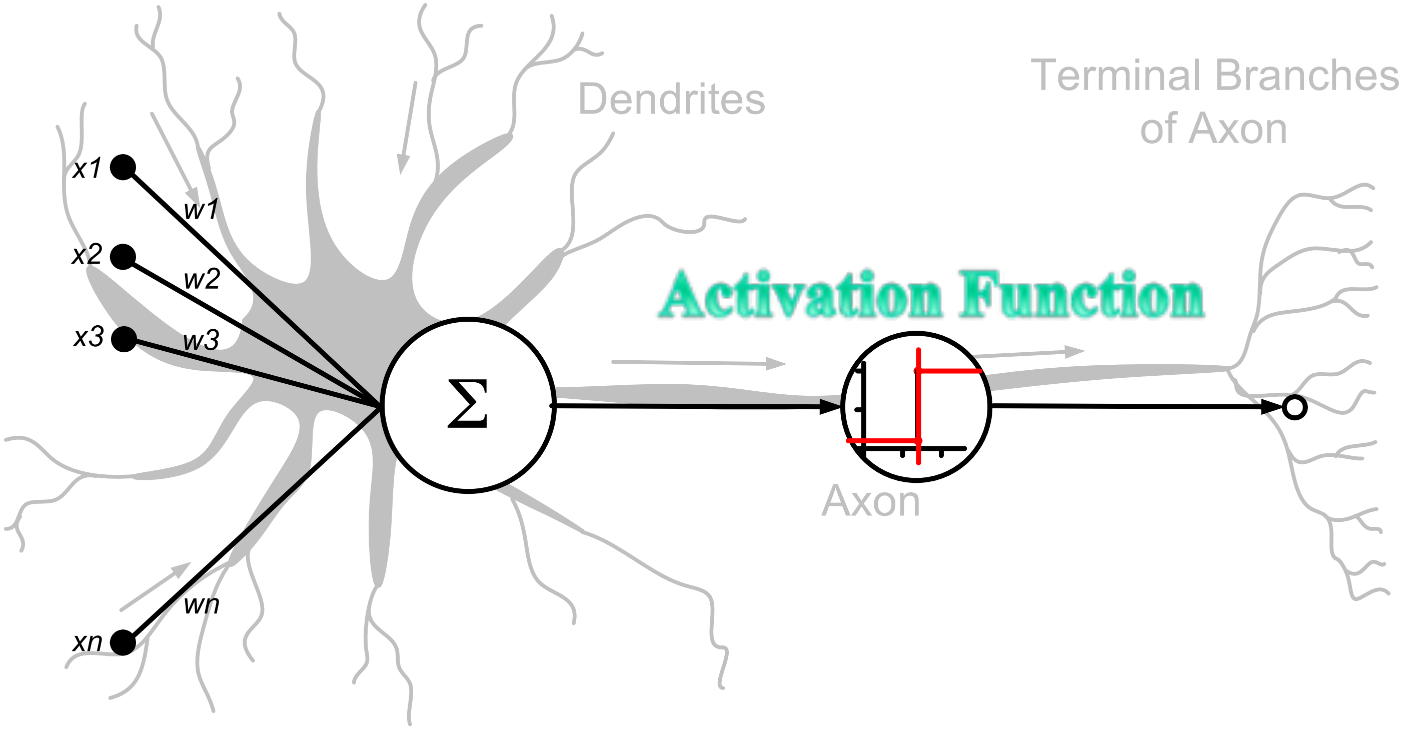
Some slides and images are taken from:

David Wolfe Corne	https://www.macs.hw.ac.uk/~dwcorne/Teaching/introdl.ppt
Wikipedia	
Geoffrey A. Hinton	

Feedforward networks for function approximation and classification

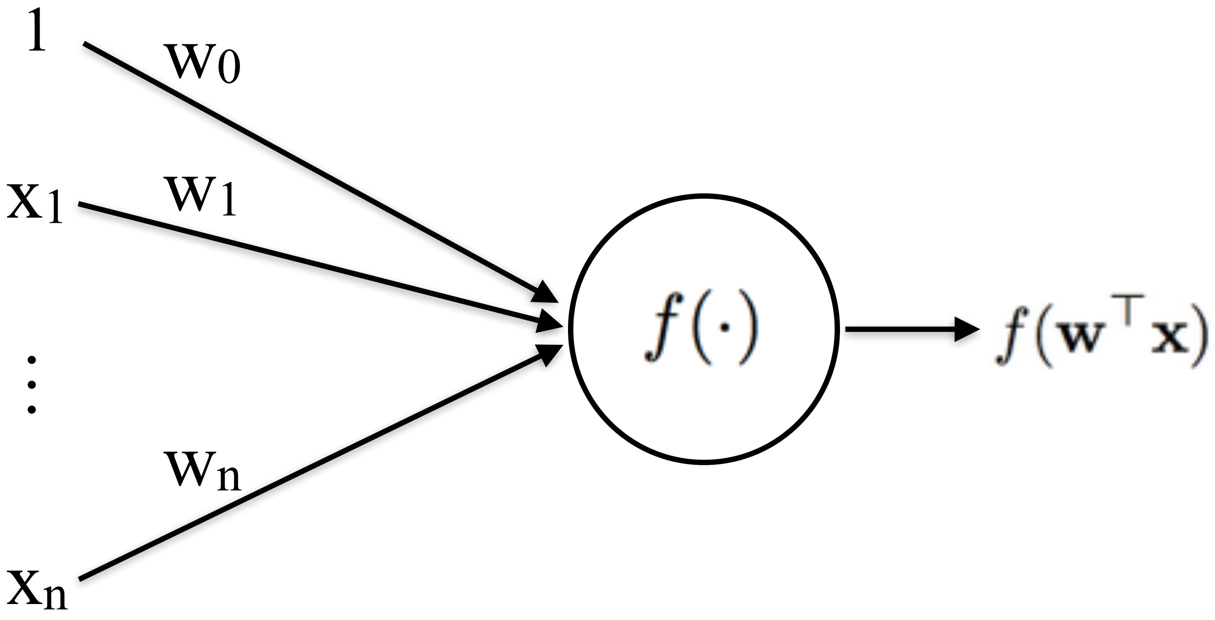


- Feedforward networks
- Deep tensor networks
- ConvNets

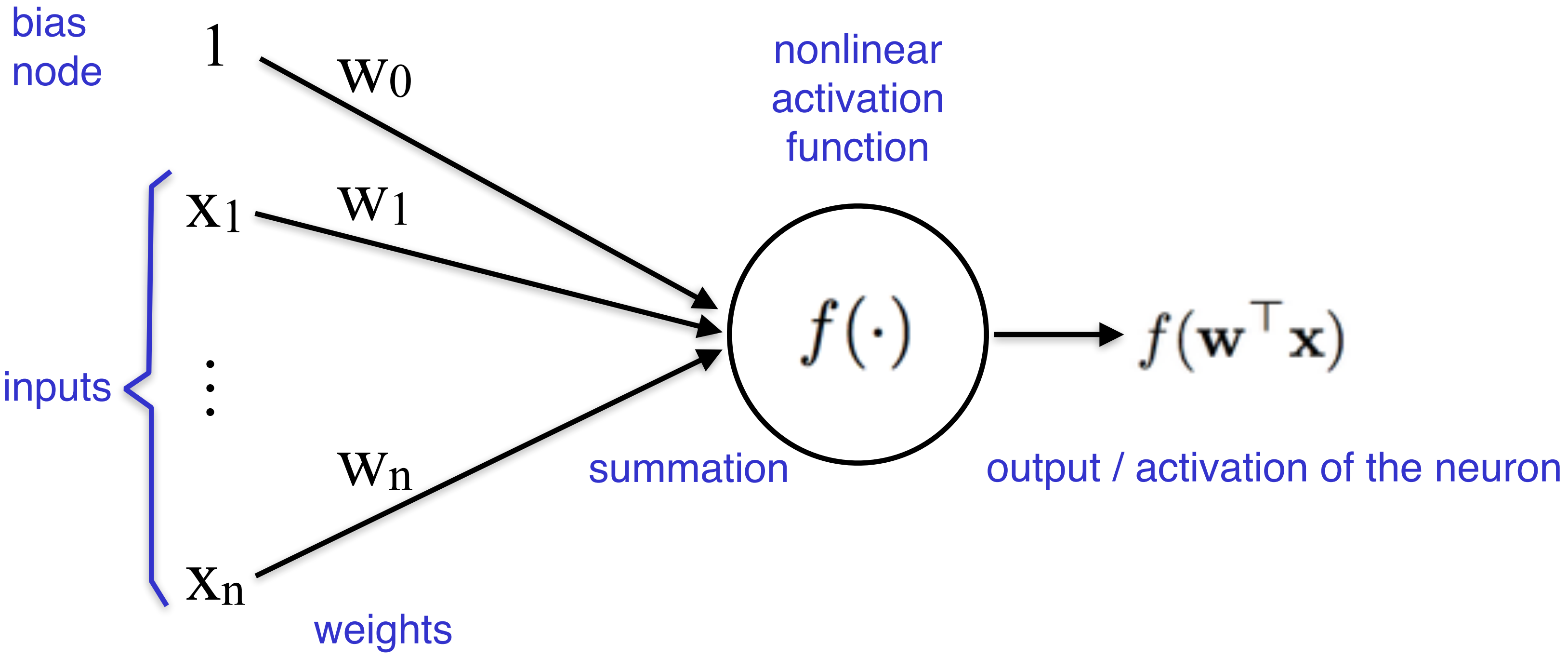


Activation Function

A single artificial neuron



A single artificial neuron



bias node

inputs

weights

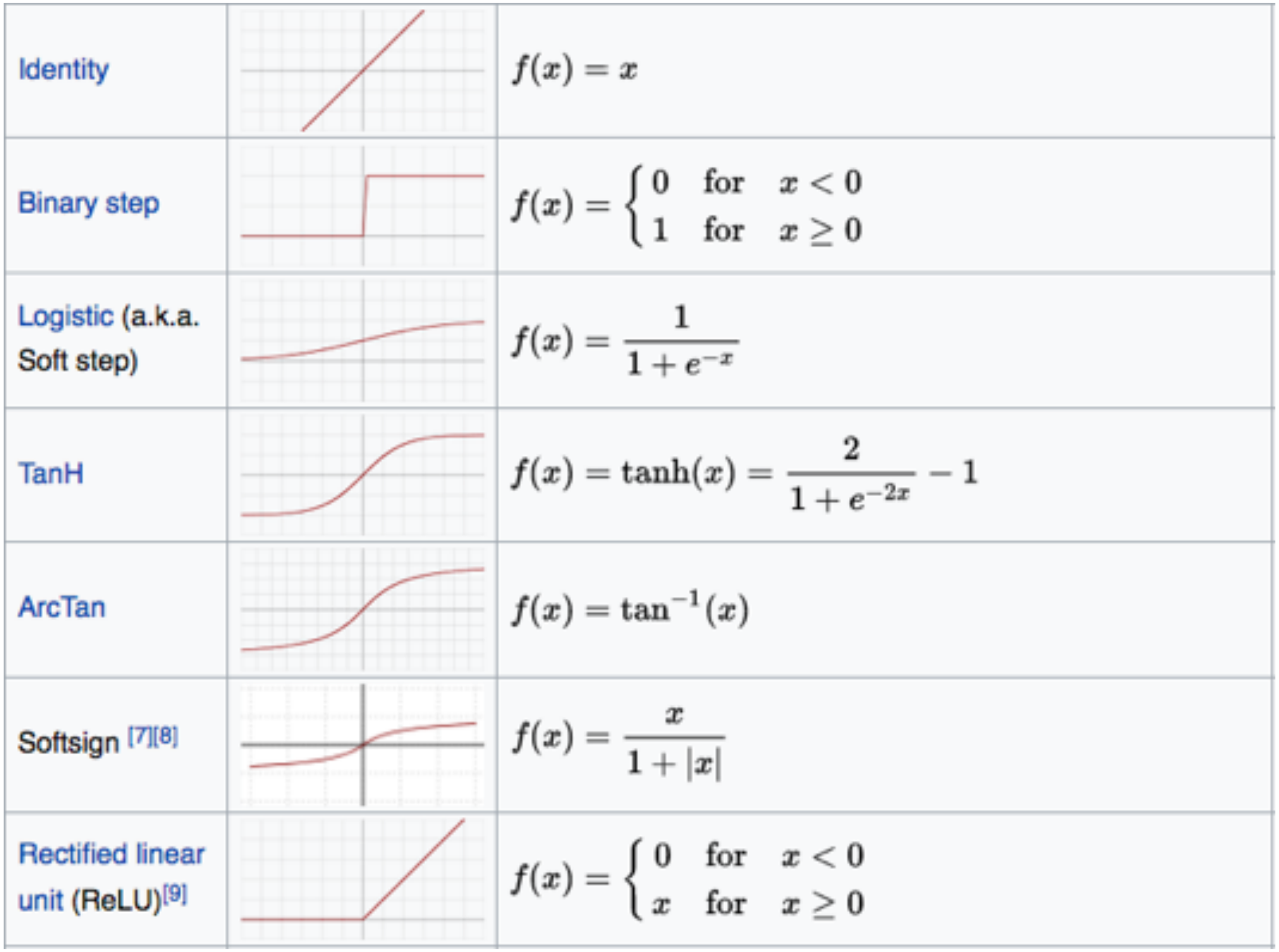
nonlinear activation function

summation

output / activation of the neuron

Activation functions

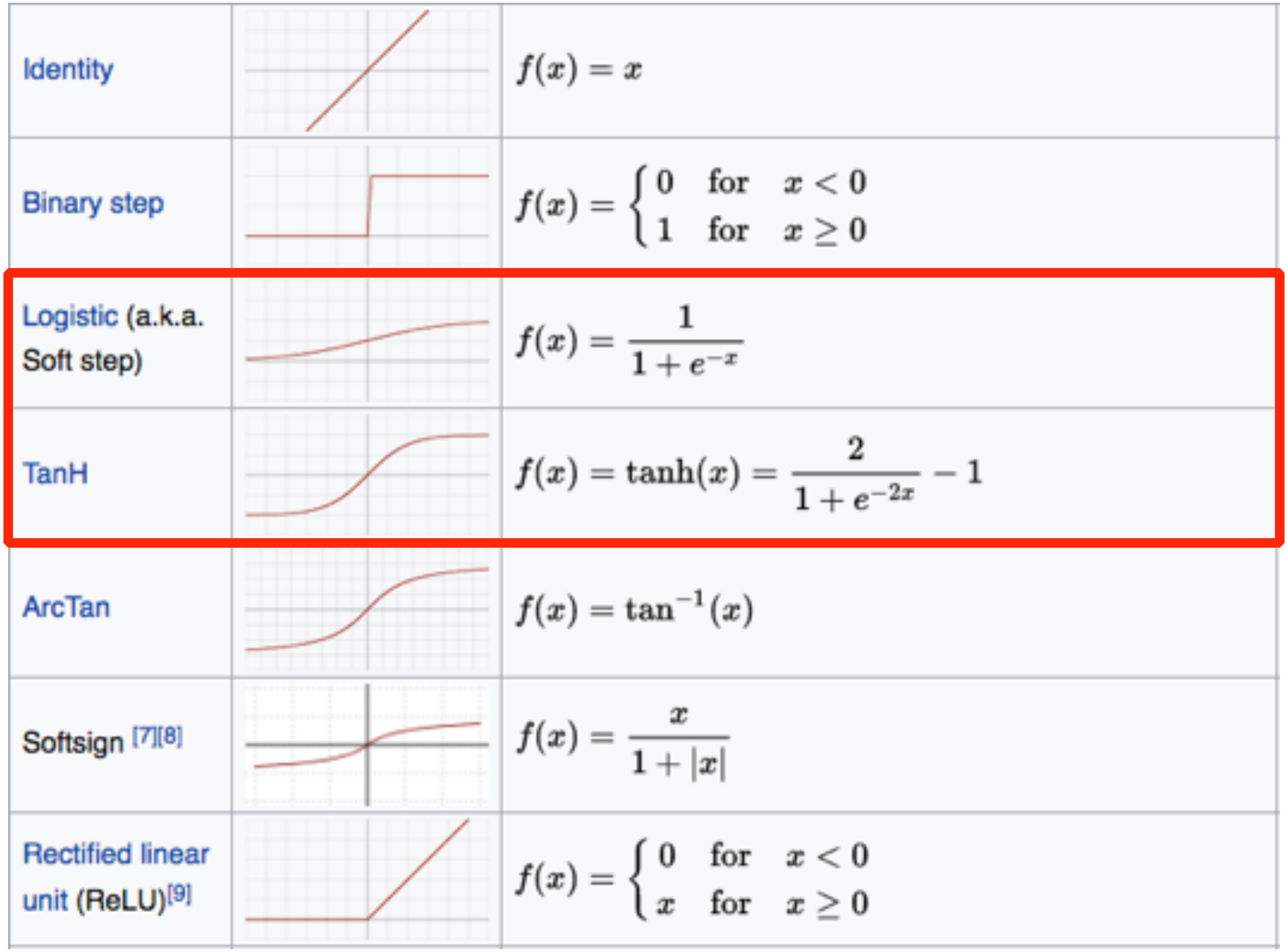
https://en.wikipedia.org/wiki/Activation_function



Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1 + \lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

Activation functions

https://en.wikipedia.org/wiki/Activation_function

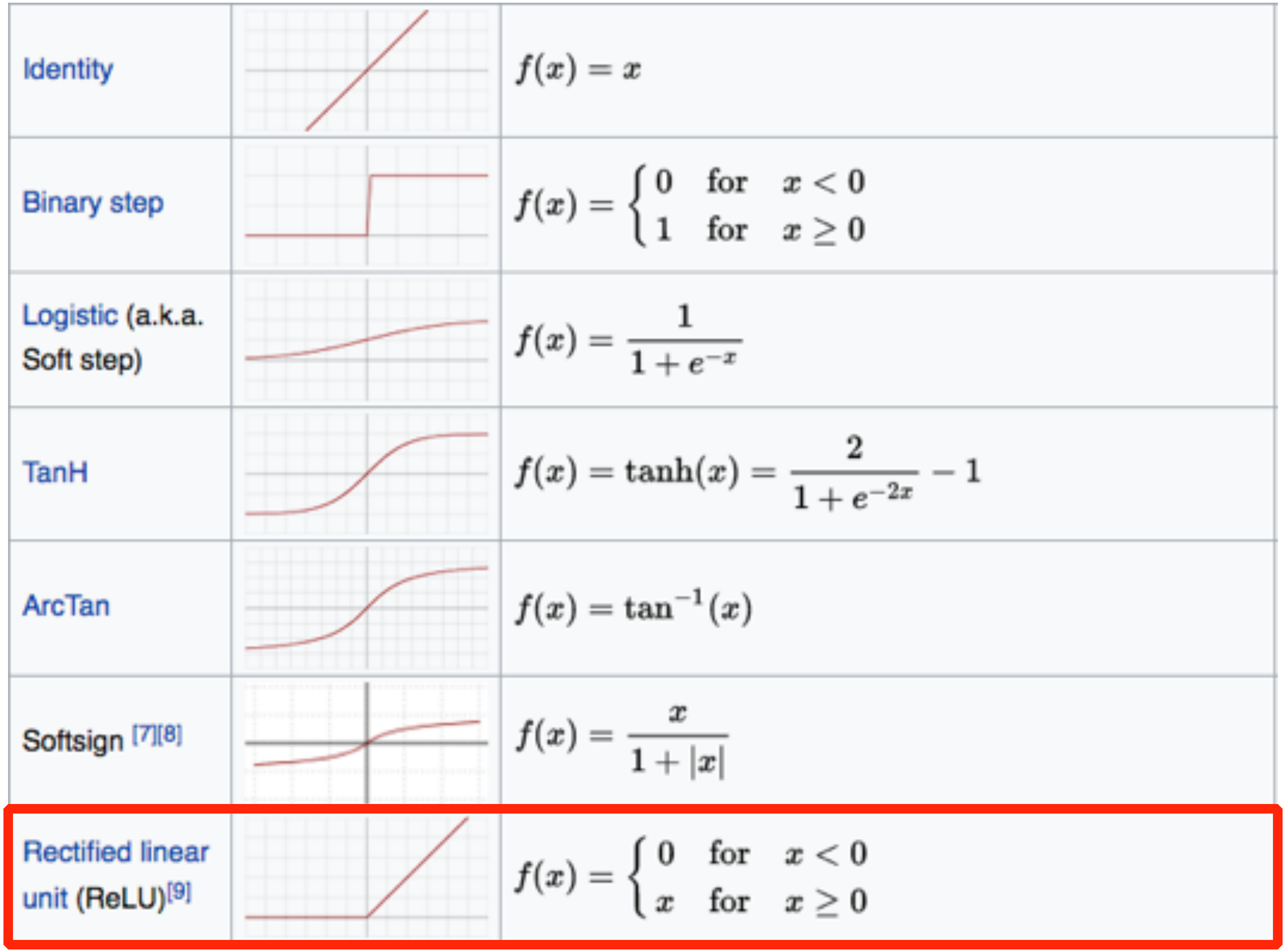


Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x\rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$

Traditionally used

Activation functions

https://en.wikipedia.org/wiki/Activation_function



Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1 + \lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

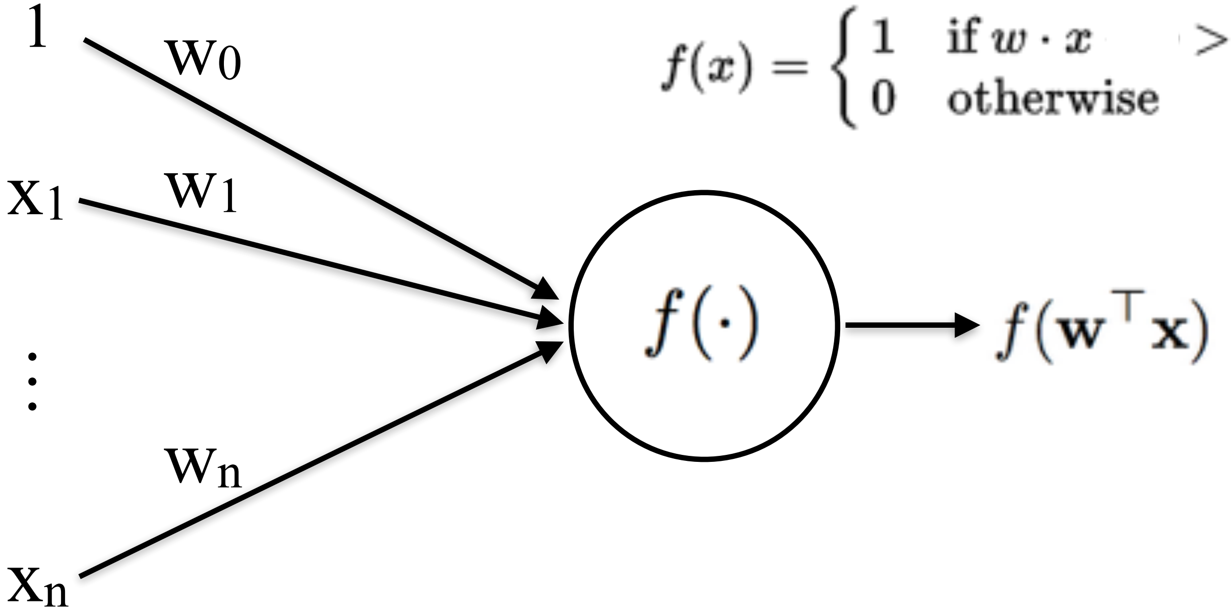
Currently most widely used. Empirically easier to train and results in sparse networks.

Nair and Hinton: Rectified linear units improve restricted Boltzmann machines ICLM'10, 807-814 (2010)
Glorot, Bordes and Bengio: Deep sparse rectifier neural networks. PMLR 15:315-323 (2011)

Perceptron (Rosenblatt 1957)

Used as a binary classifier - equivalent to support vector machine (SVM)

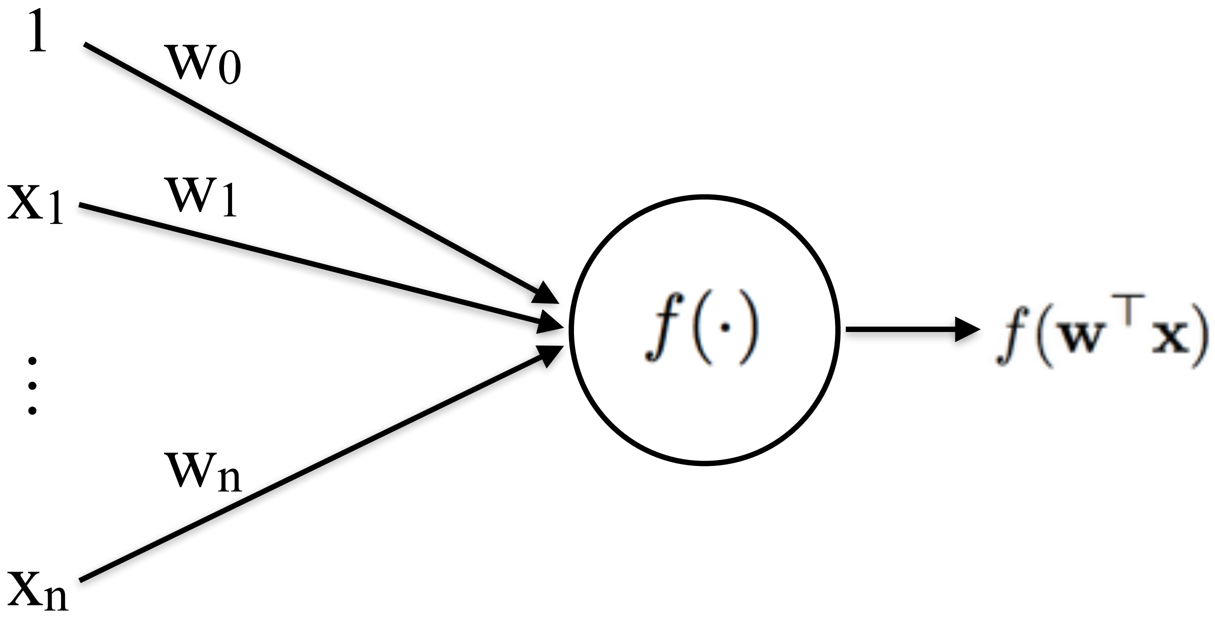
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x > 0 \\ 0 & \text{otherwise} \end{cases}$$

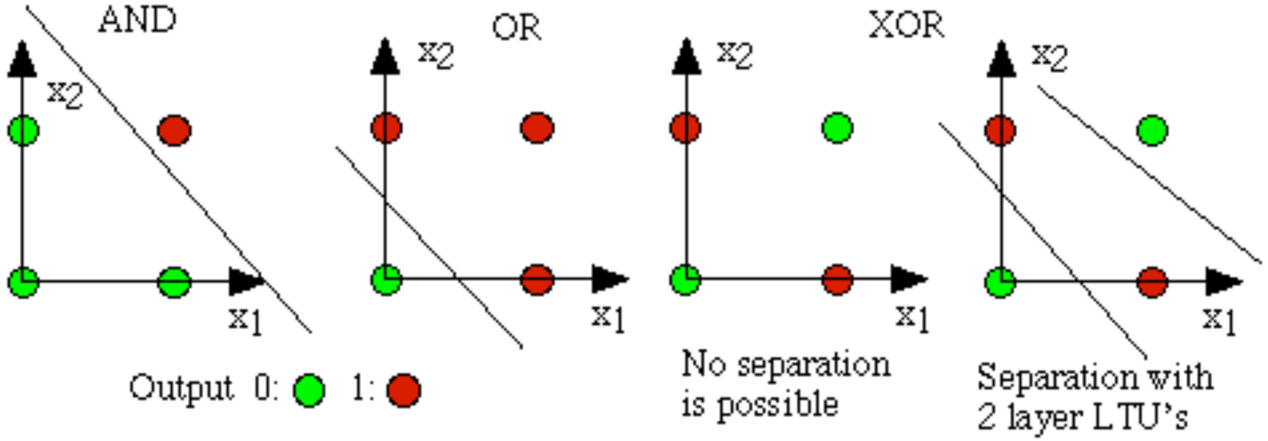


Train weights using examples $D = \{(\mathbf{x}_1, d_1), ..., (\mathbf{x}_S, d_S)\}$:

1. Initialize $\mathbf{w}$ with random values
2. For each example $(\mathbf{x}_j, d_j)$:
    (a) Calculate output: $y_j = f(\mathbf{w}^\top \mathbf{x}_j)$
    (b) Update weights: $\mathbf{w} \leftarrow \mathbf{w} + (d_j - y_j)\mathbf{x}_j$
3. Repeat 2 until convergence.

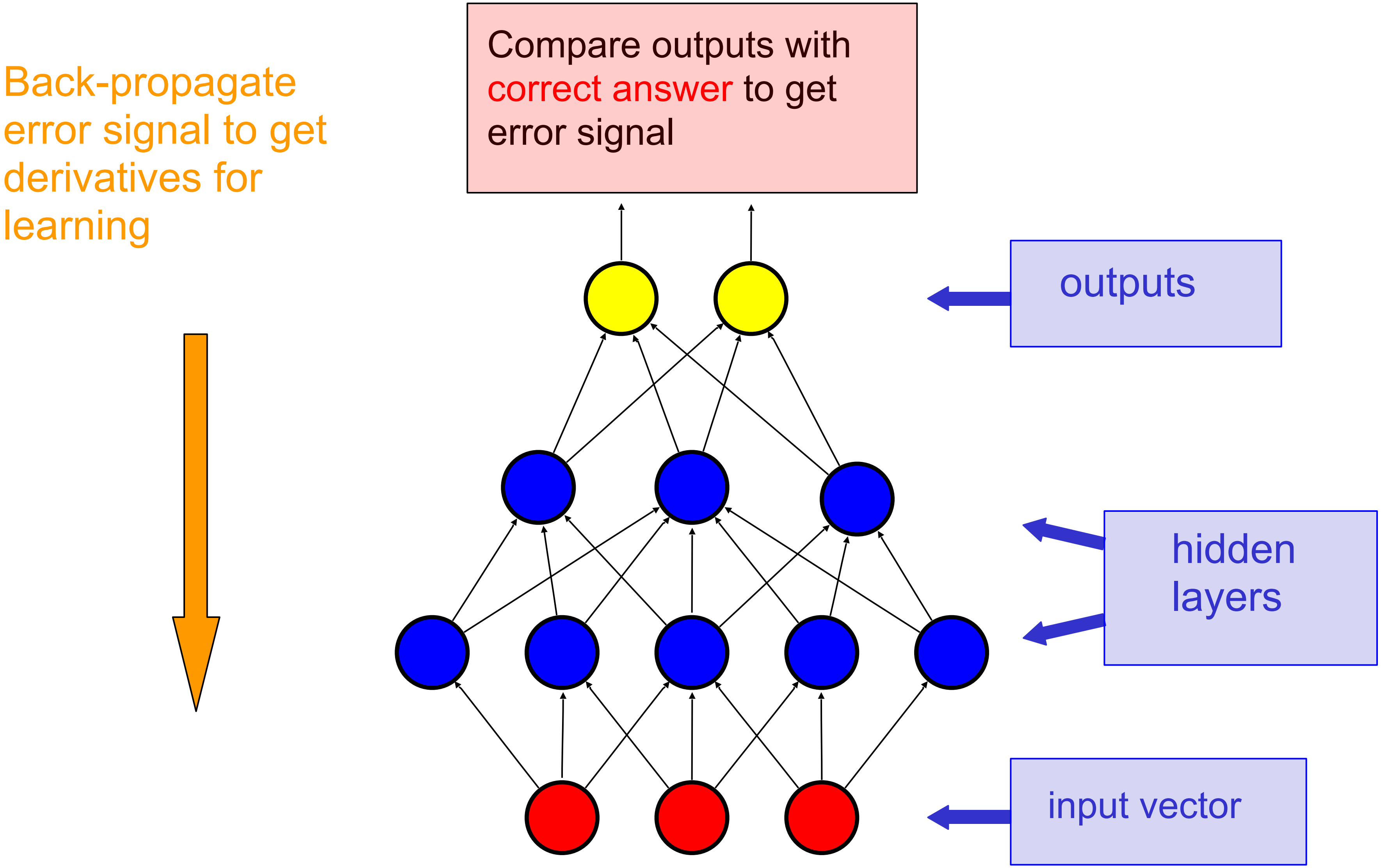
Perceptron (Rosenblatt 1957)





Second-generation neural networks (~1985)

Slide credit : Geoffrey Hinton



Back-propagate error signal to get derivatives for learning

Compare outputs with correct answer to get error signal

outputs

hidden layers

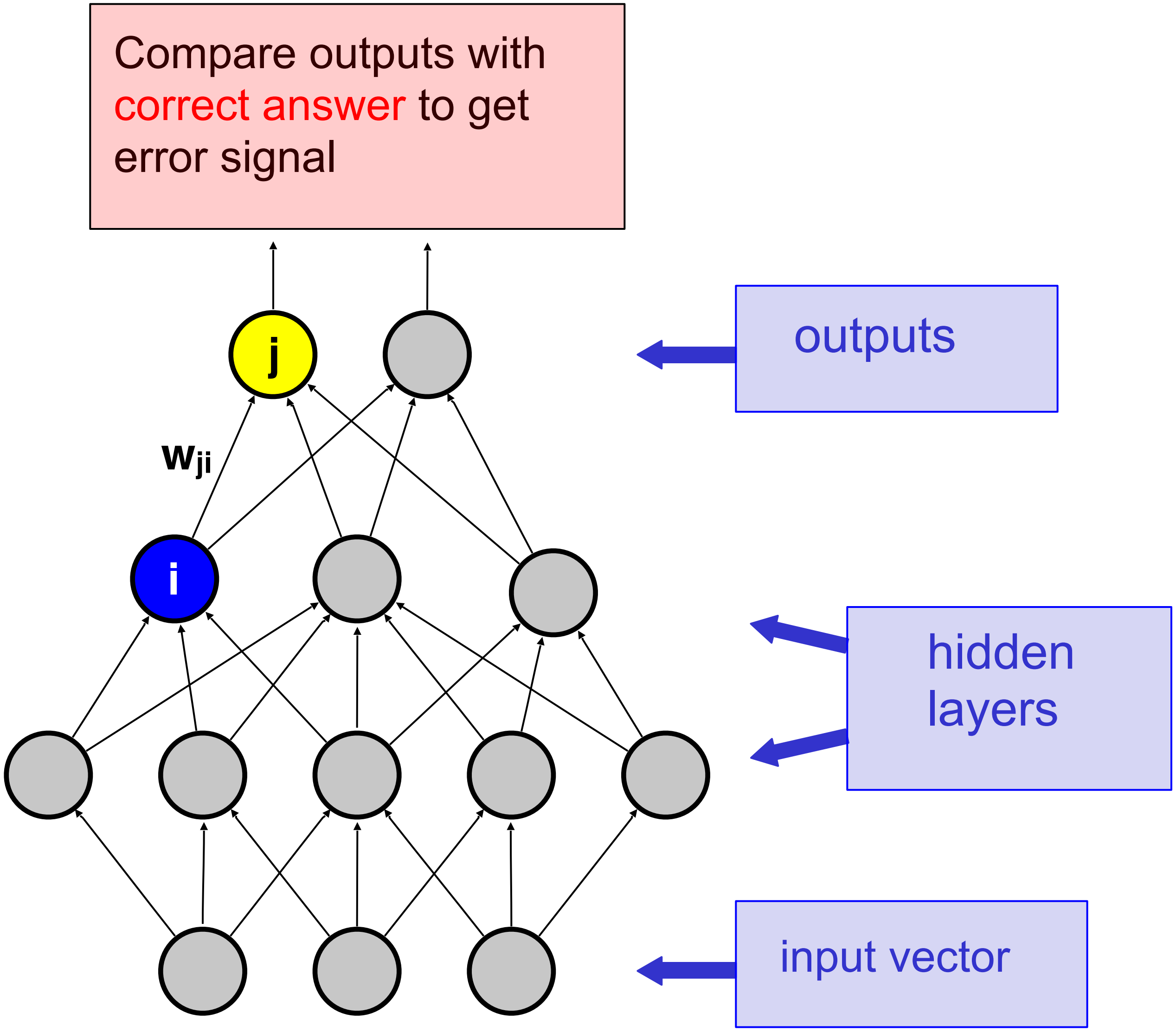
input vector

Second-generation neural networks (~1985)

Slide credit : Geoffrey Hinton

Error at output for a given example:

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2$$



Compare outputs with correct answer to get error signal

See more: https://hmkcode.com/ai/backpropagation-step-by-step/

Second-generation neural networks (~1985)

Slide credit :
Geoffrey Hinton

Error at output for a given example:

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2$$

Error sensitivity at output neuron j:

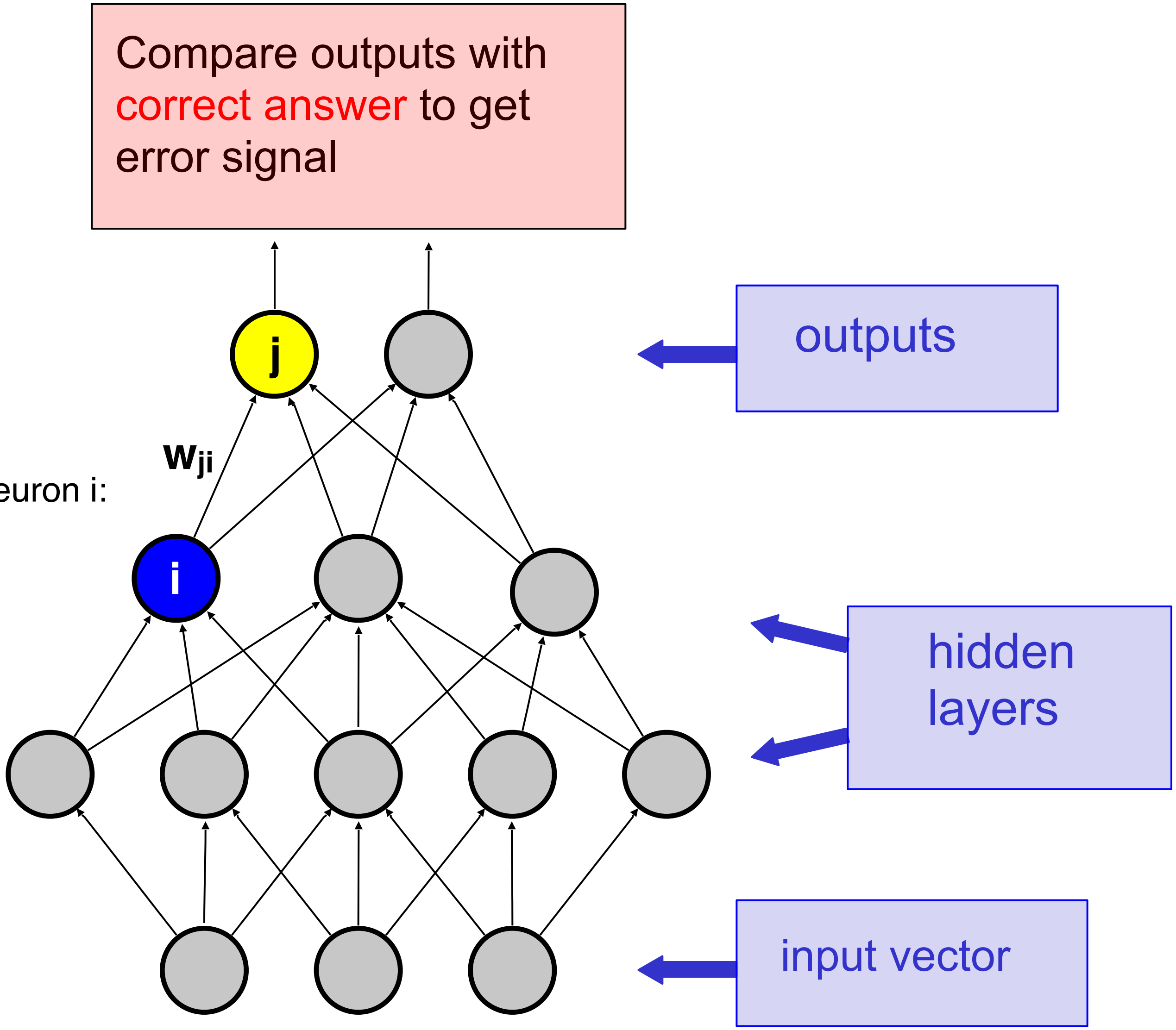
$$\frac{\partial E}{\partial y_j} = y_j - d_j$$

Backpropagate error sensitivity to neuron i:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial y_j} f'(x_j) w_{ji}$$

Sensitivity on weight w_{ji}:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} y_i$$



Compare outputs with correct answer to get error signal

outputs

hidden layers

input vector

Second-generation neural networks (~1985)

Slide credit : Geoffrey Hinton

Error at output for a given example:

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2$$

Error sensitivity at output neuron j:

$$\frac{\partial E}{\partial y_j} = y_j - d_j$$

Backpropagate error sensitivity to neuron i:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial y_j} f'(x_j) w_{ji}$$

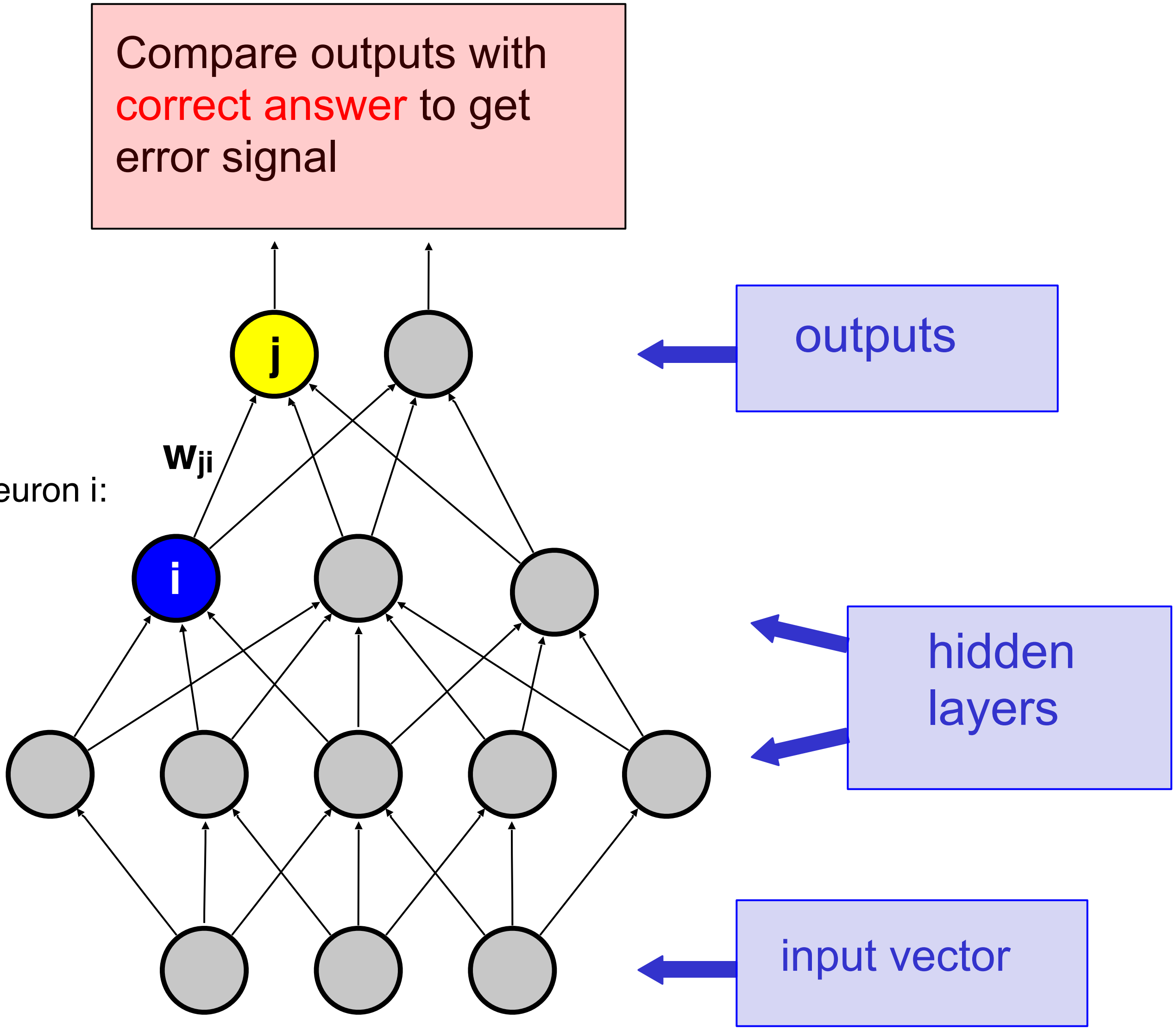
Sensitivity on weight w_{ji}:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} y_i$$

Update weight w_{ji}:

$$\Delta w_{ji} = -\overset{\uparrow}{\epsilon} \frac{\partial E}{\partial w_{ji}}$$

learning rate



Compare outputs with correct answer to get error signal

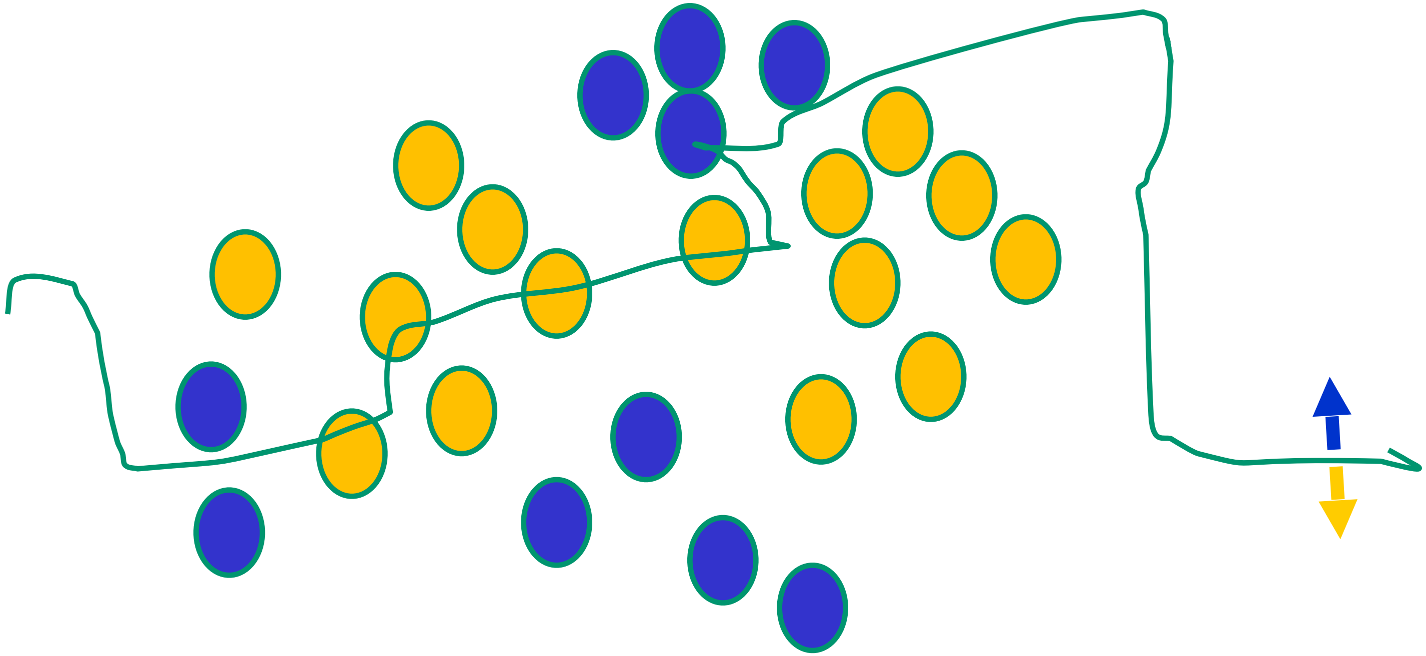
outputs

hidden layers

input vector

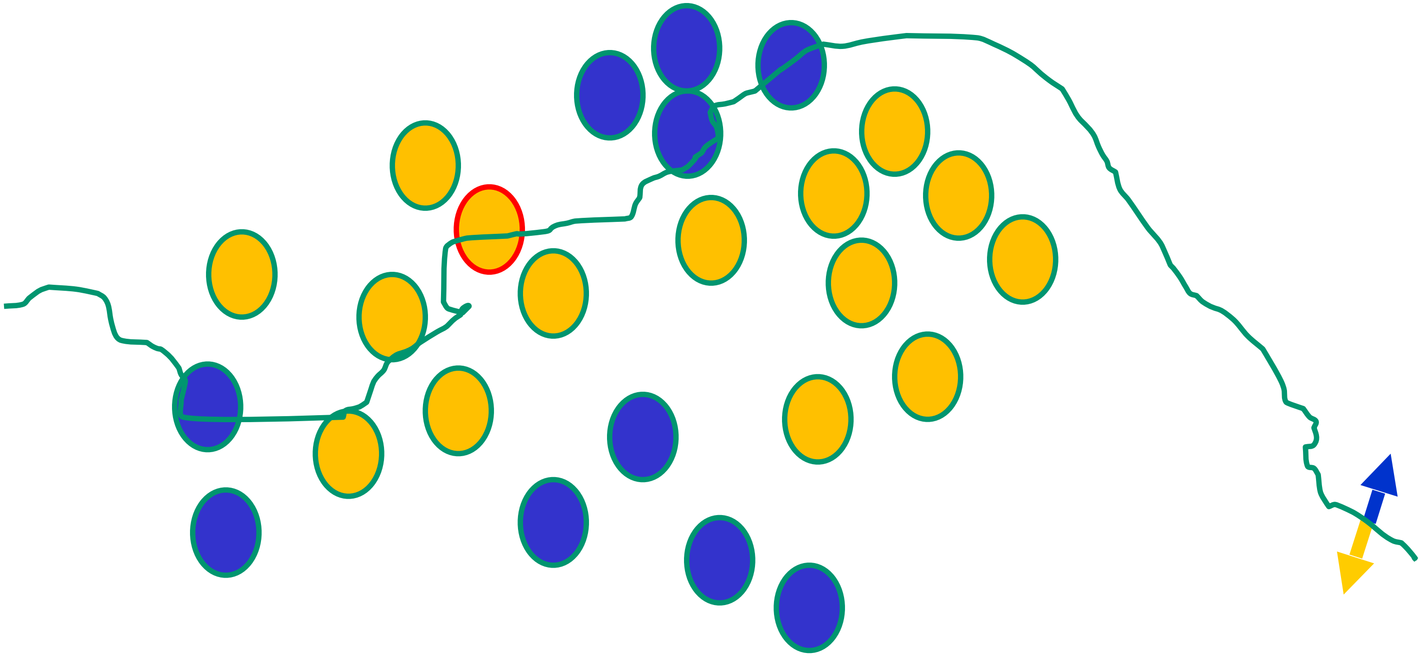
A decision boundary perspective on learning

Initial random weights



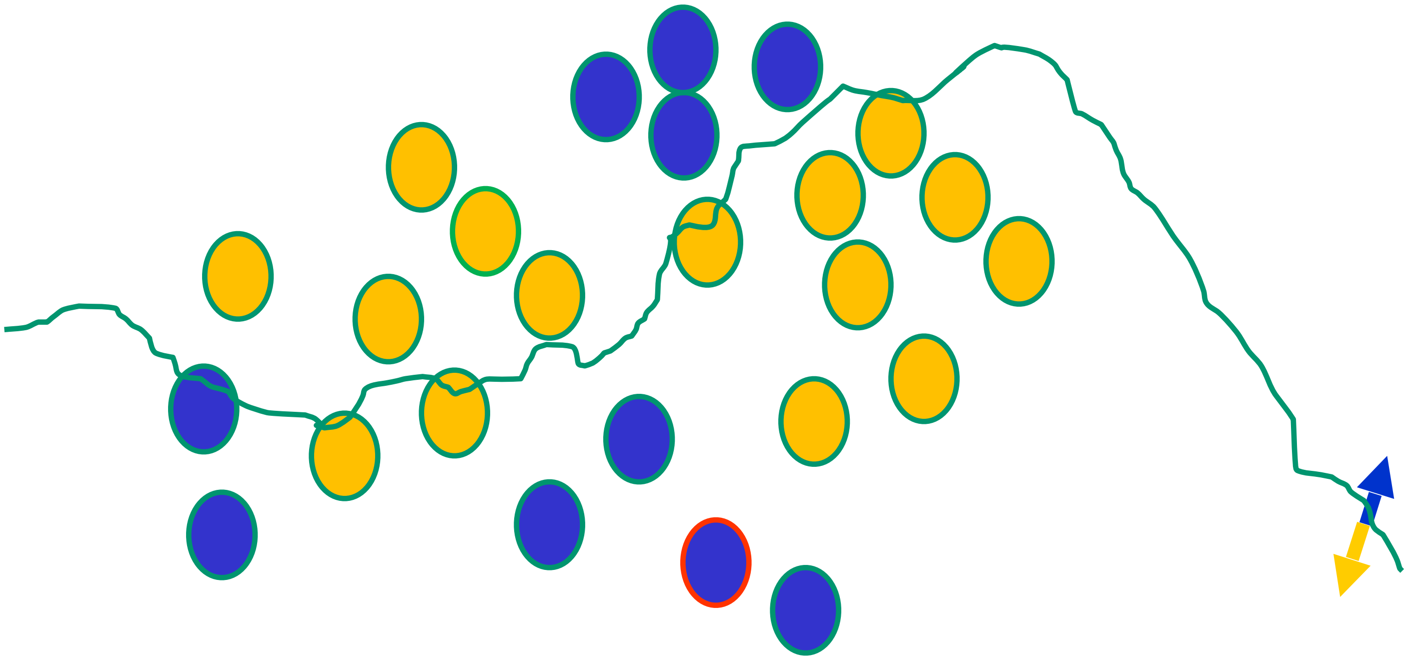
A decision boundary perspective on learning

Present a training instance / adjust the weights



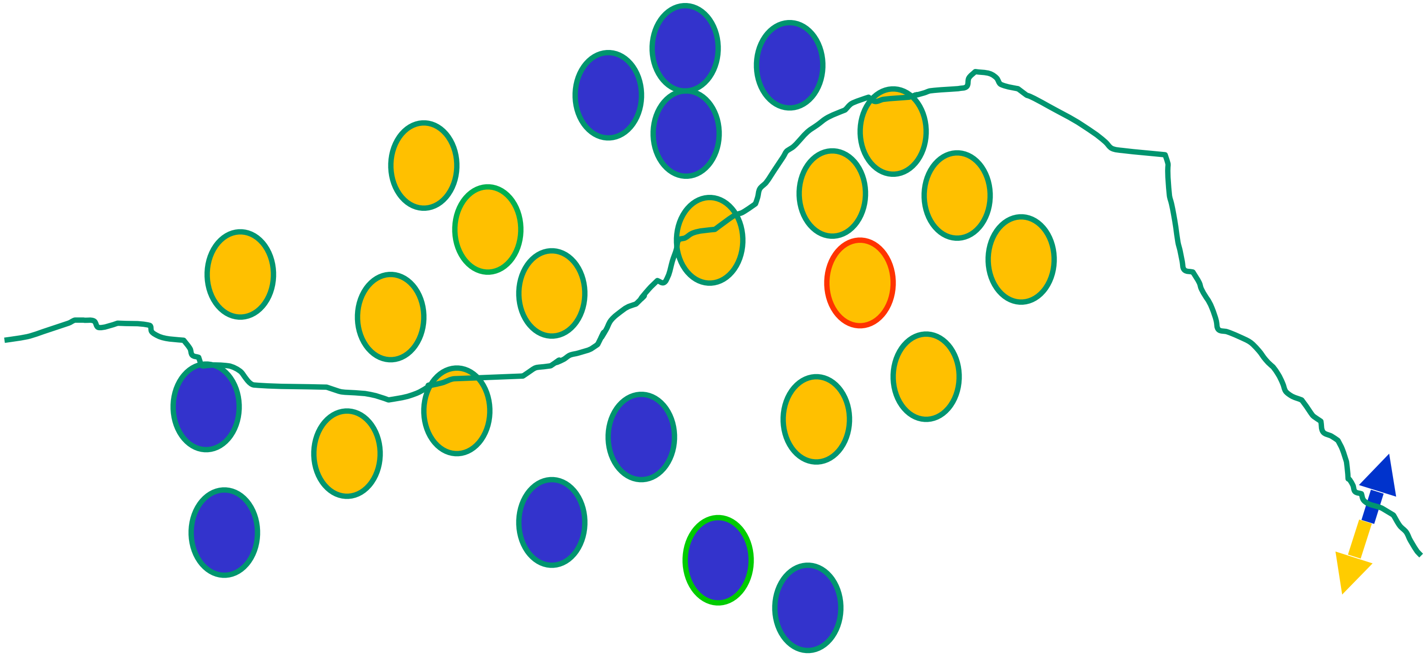
A decision boundary perspective on learning

Present a training instance / adjust the weights



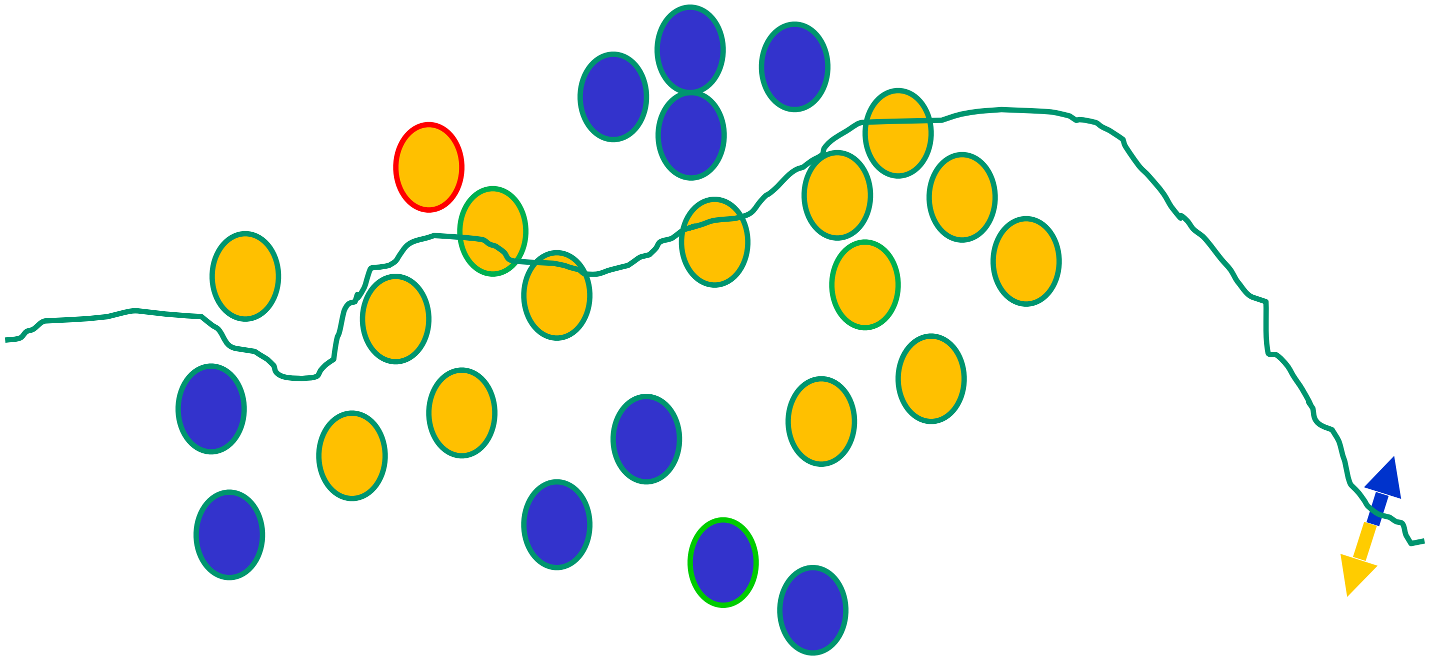
A decision boundary perspective on learning

Present a training instance / adjust the weights



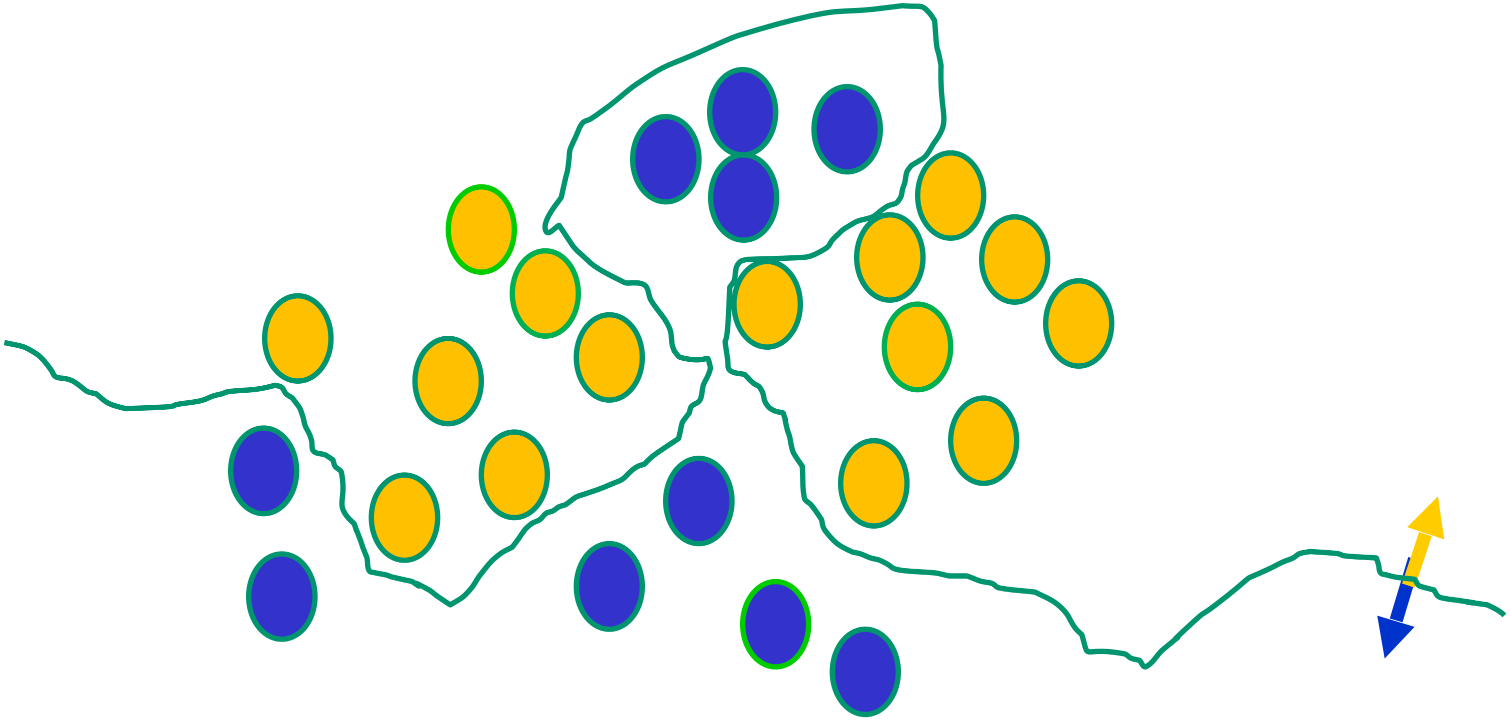
A decision boundary perspective on learning

Present a training instance / adjust the weights



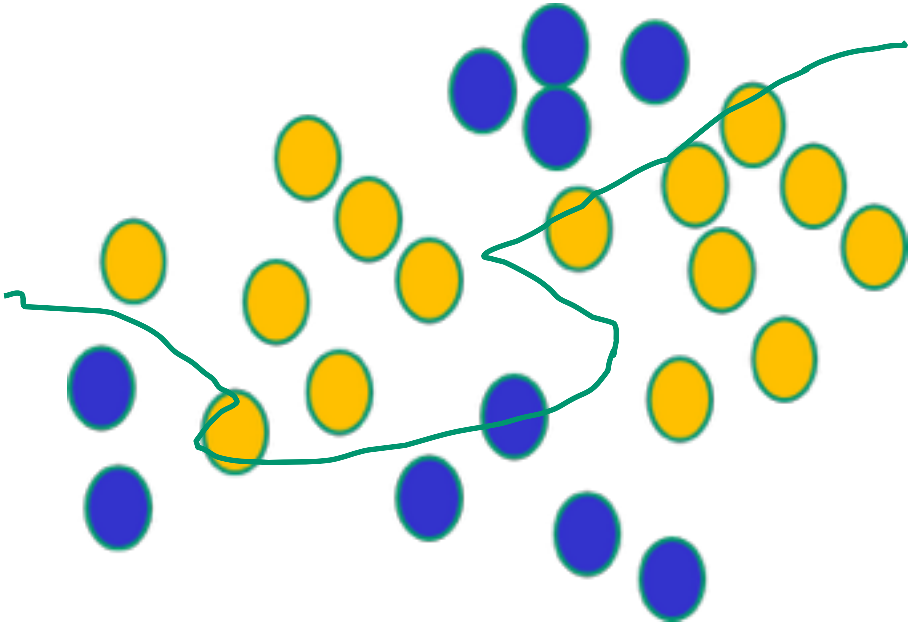
A decision boundary perspective on learning

Eventually ….

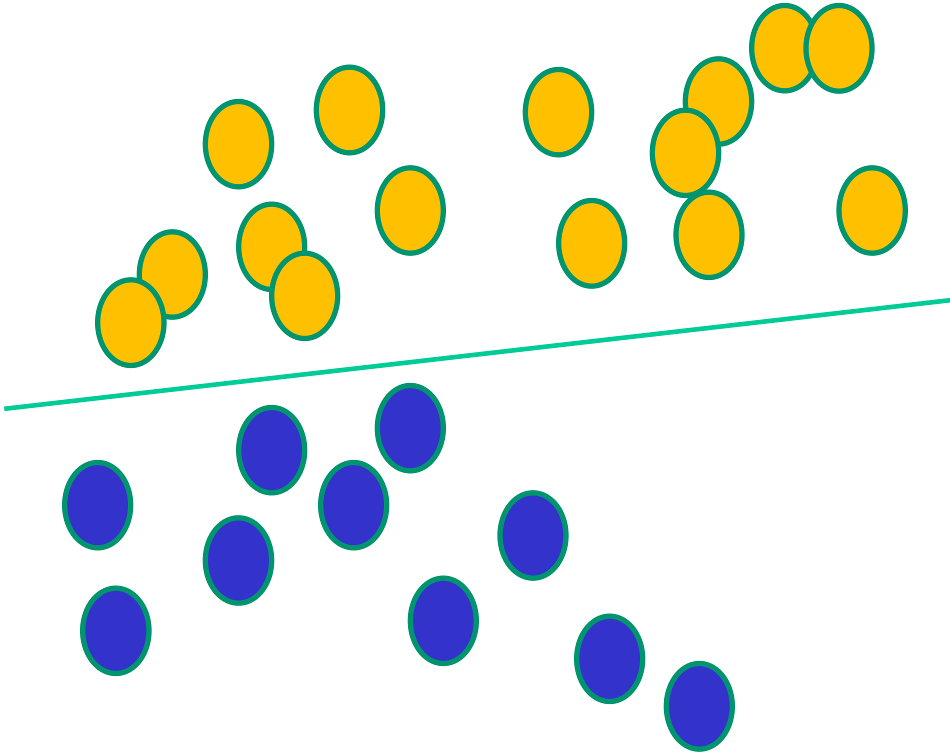


Nonlinear versus linear models

NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged



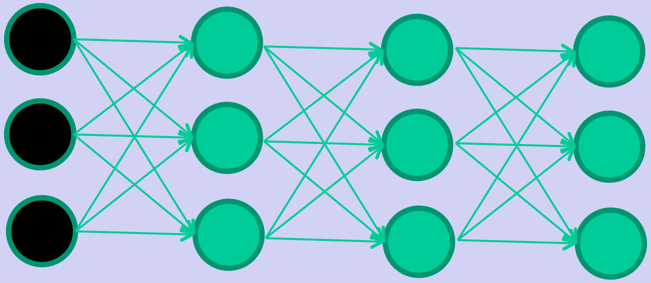
Kernel methods only draw straight lines, but transform the data first in a way that makes it linearly separable



Universal Representation Theorem

- Networks with a **single** hidden layer can represent **any** function F(x) with arbitrary precision in the large hidden layer size limit
- However, that doesn‘t mean, networks with single hidden layers are efficient in representing arbitrary functions. For many datasets, deep networks can represent the function F(x) even with narrow layers.

What does a neural network learn?



Feature detectors

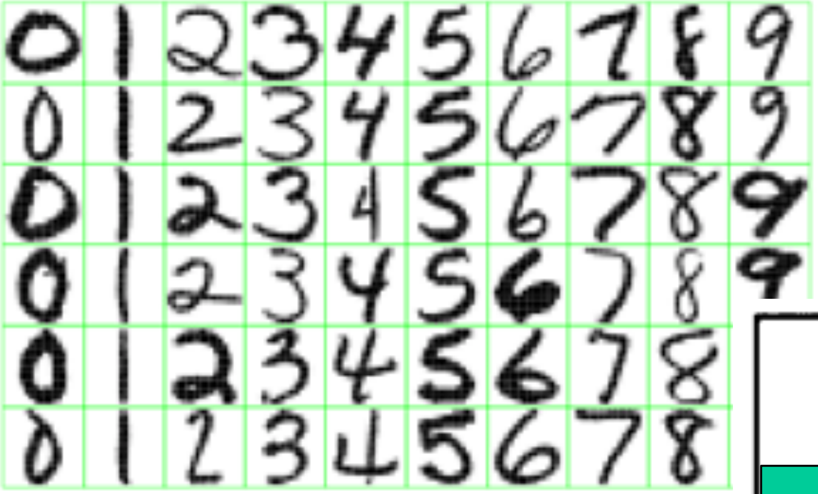
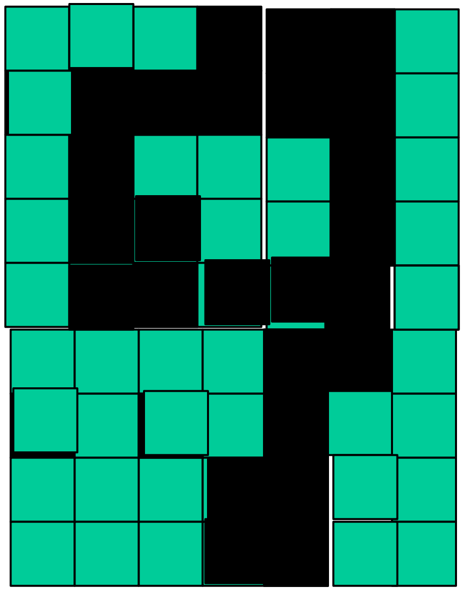
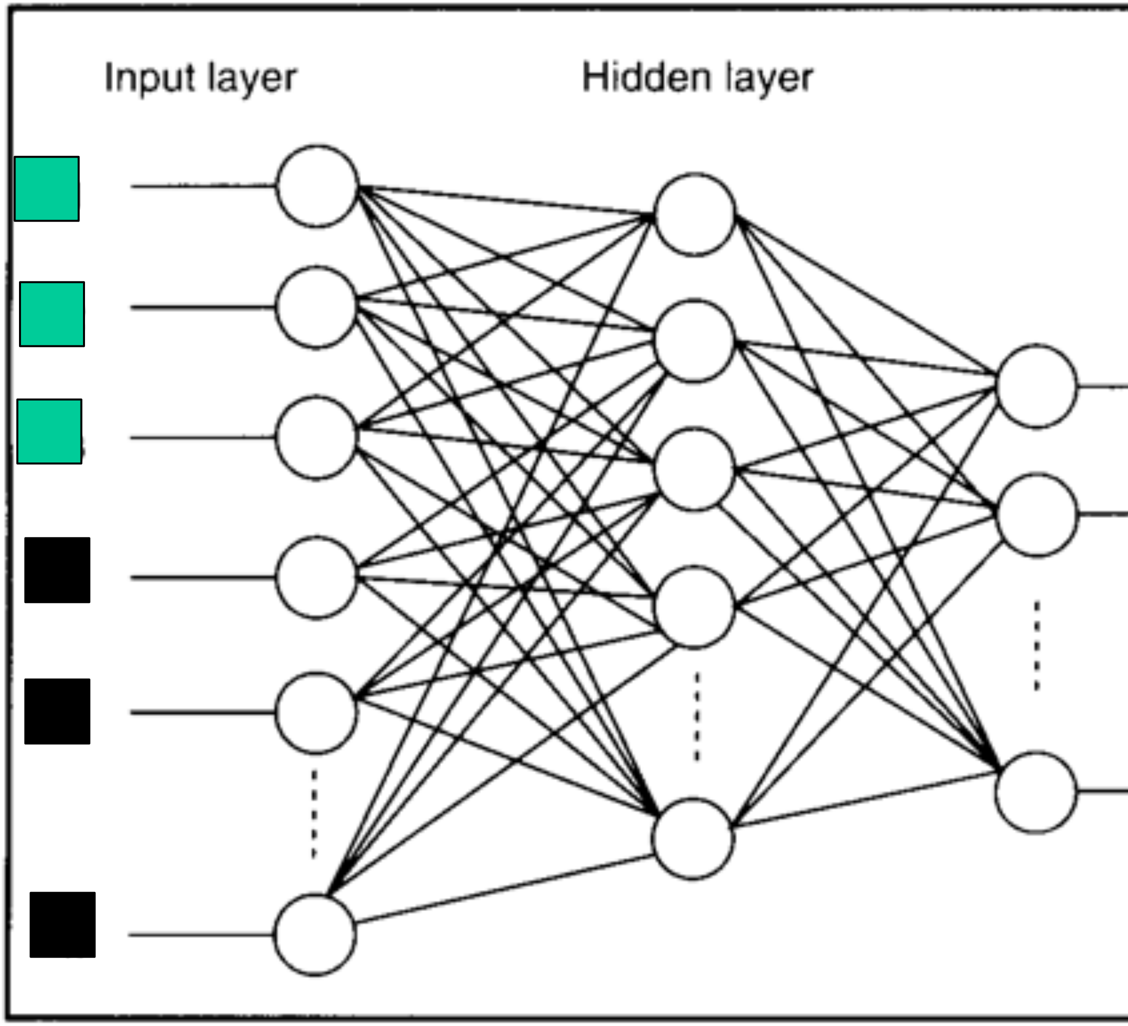


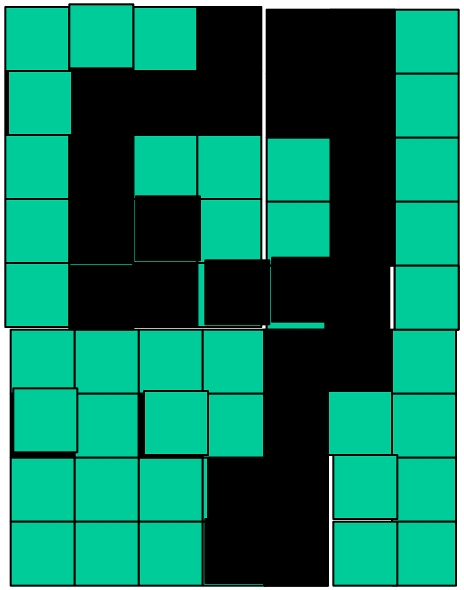
Figure 1.2: *Examples of handwritten digits from postal envelopes.*

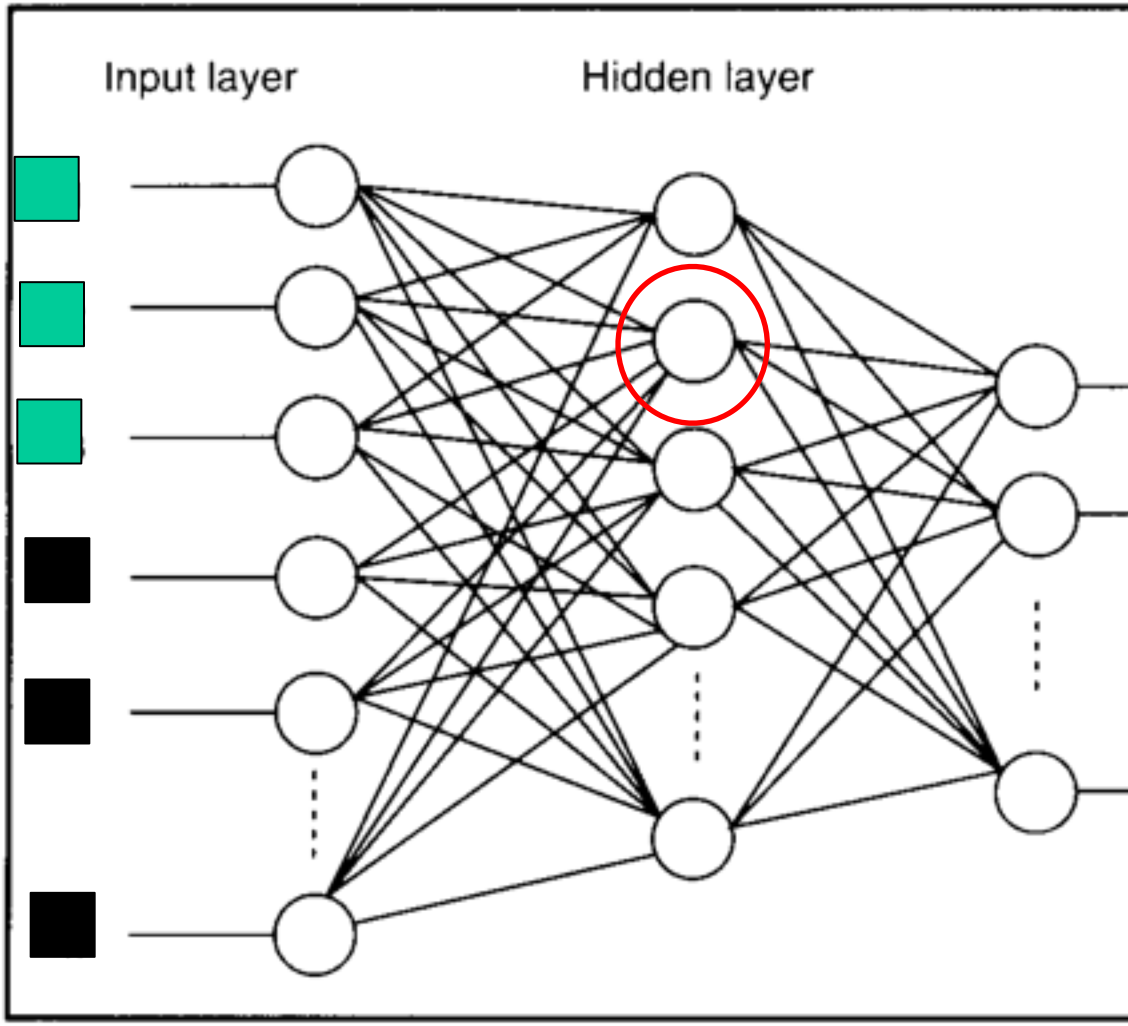




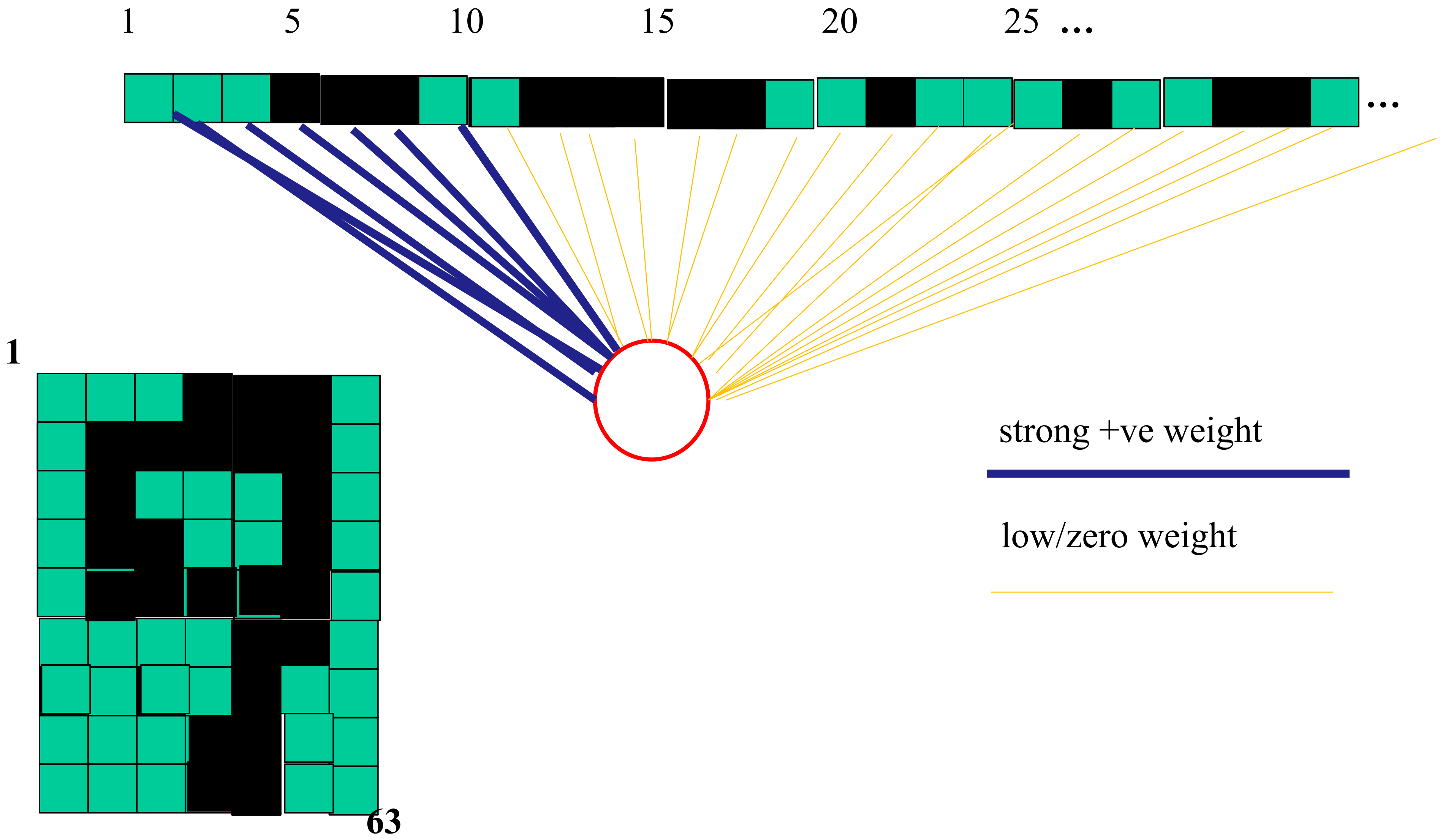
What is this unit doing?

Figure 1.2: *Examples of handwritten digits from postal envelopes.*

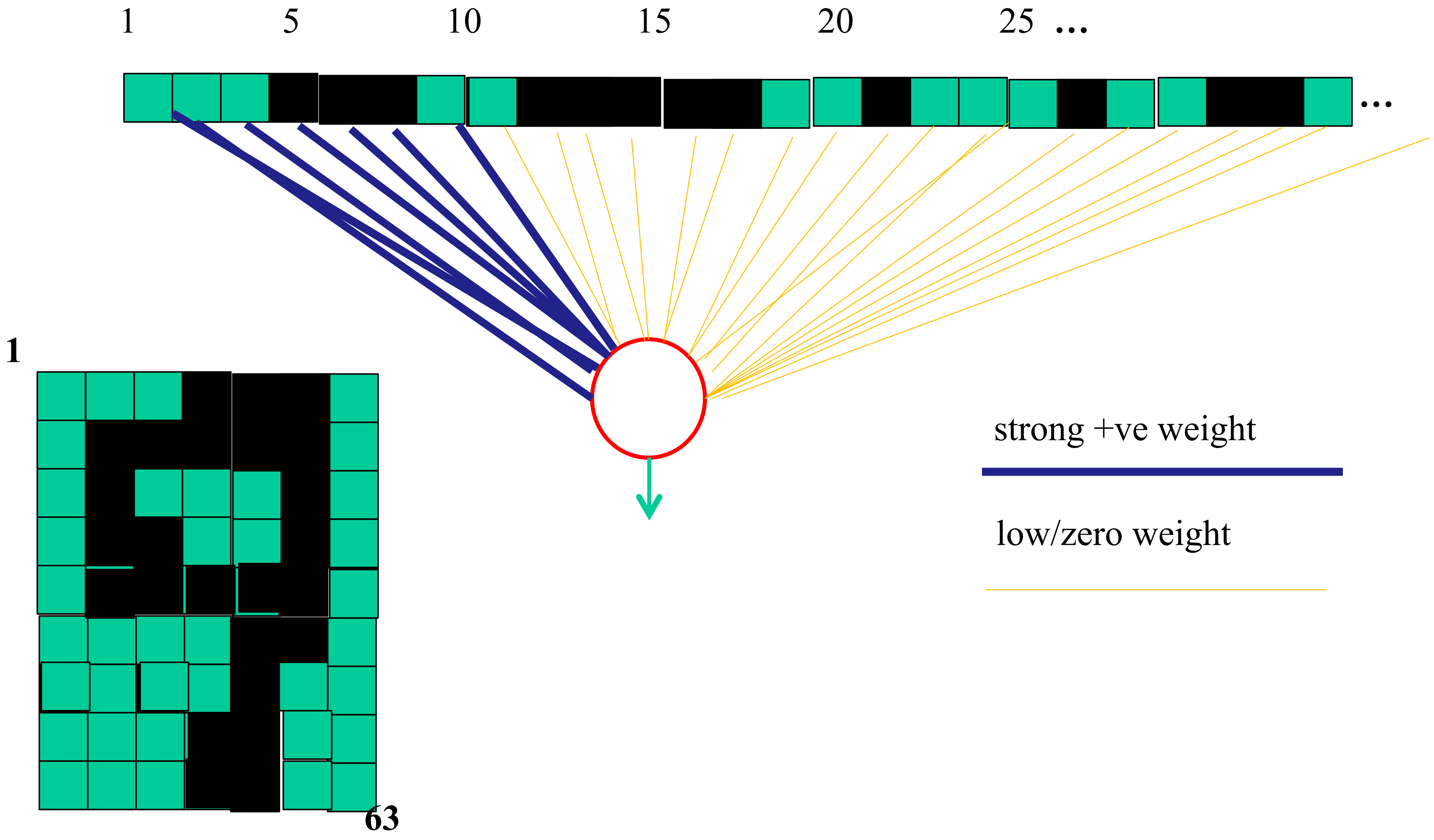




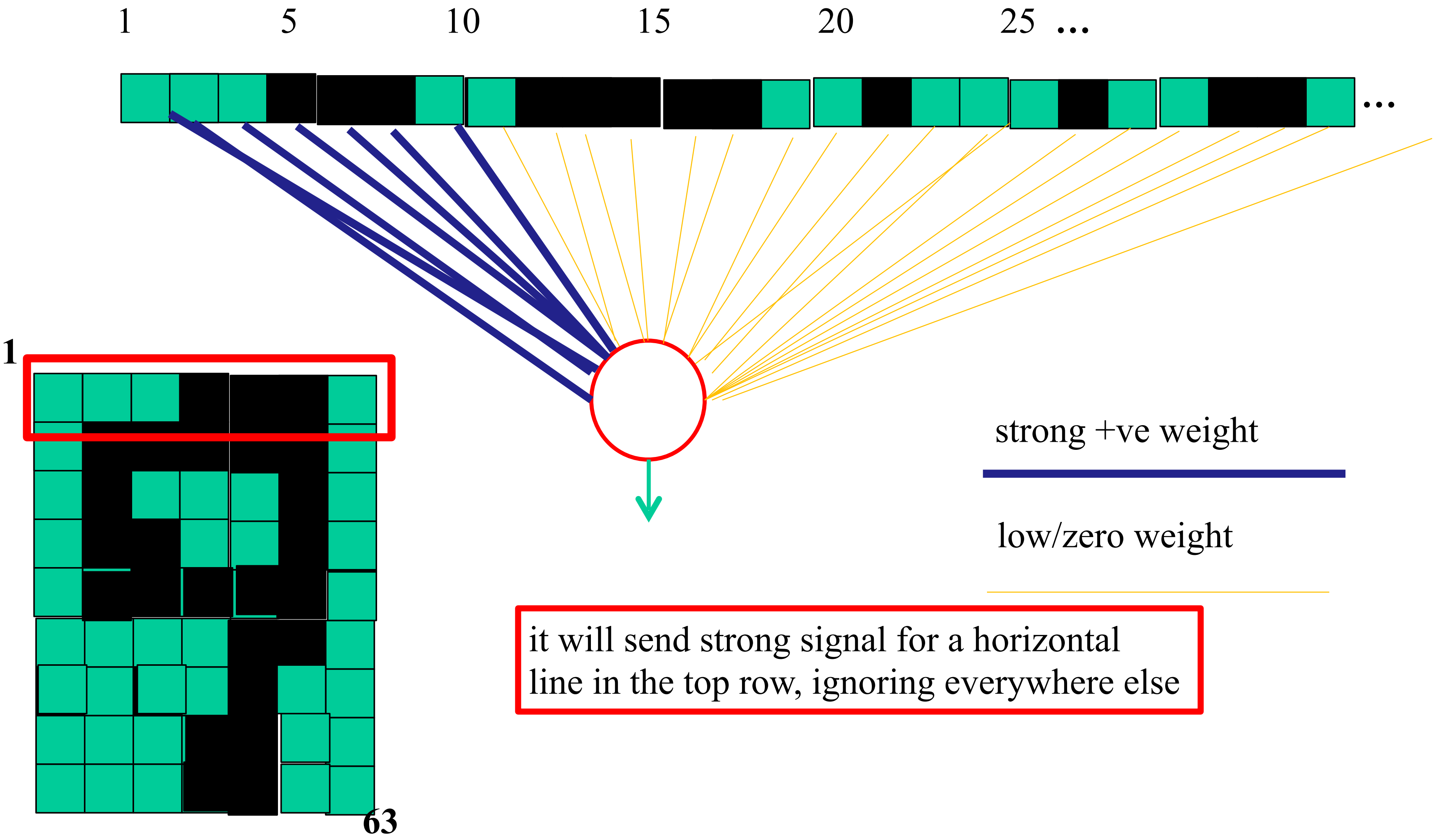
Hidden layer units become self-organized feature detectors



what does this unit detect?

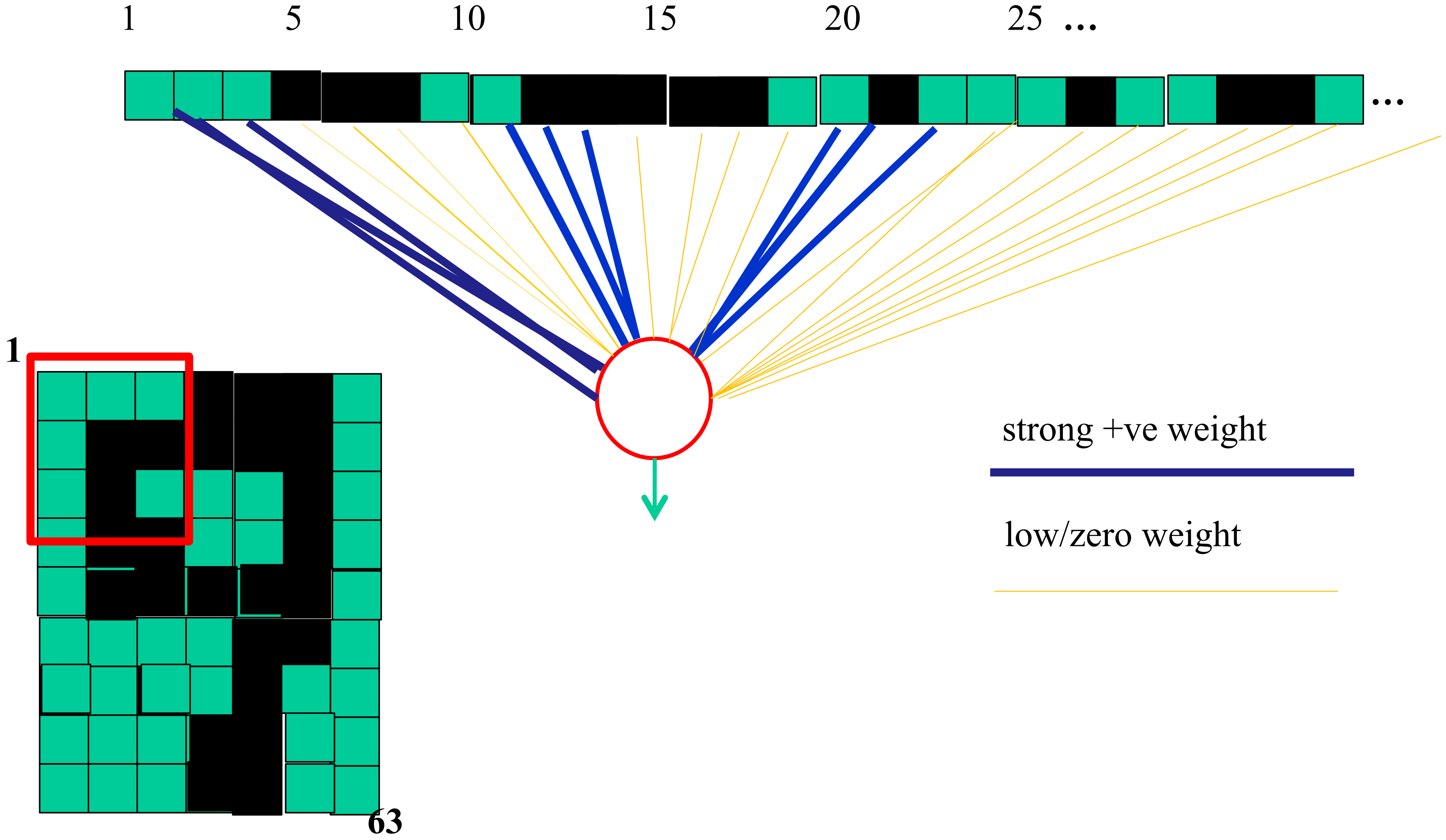


what does this unit detect?

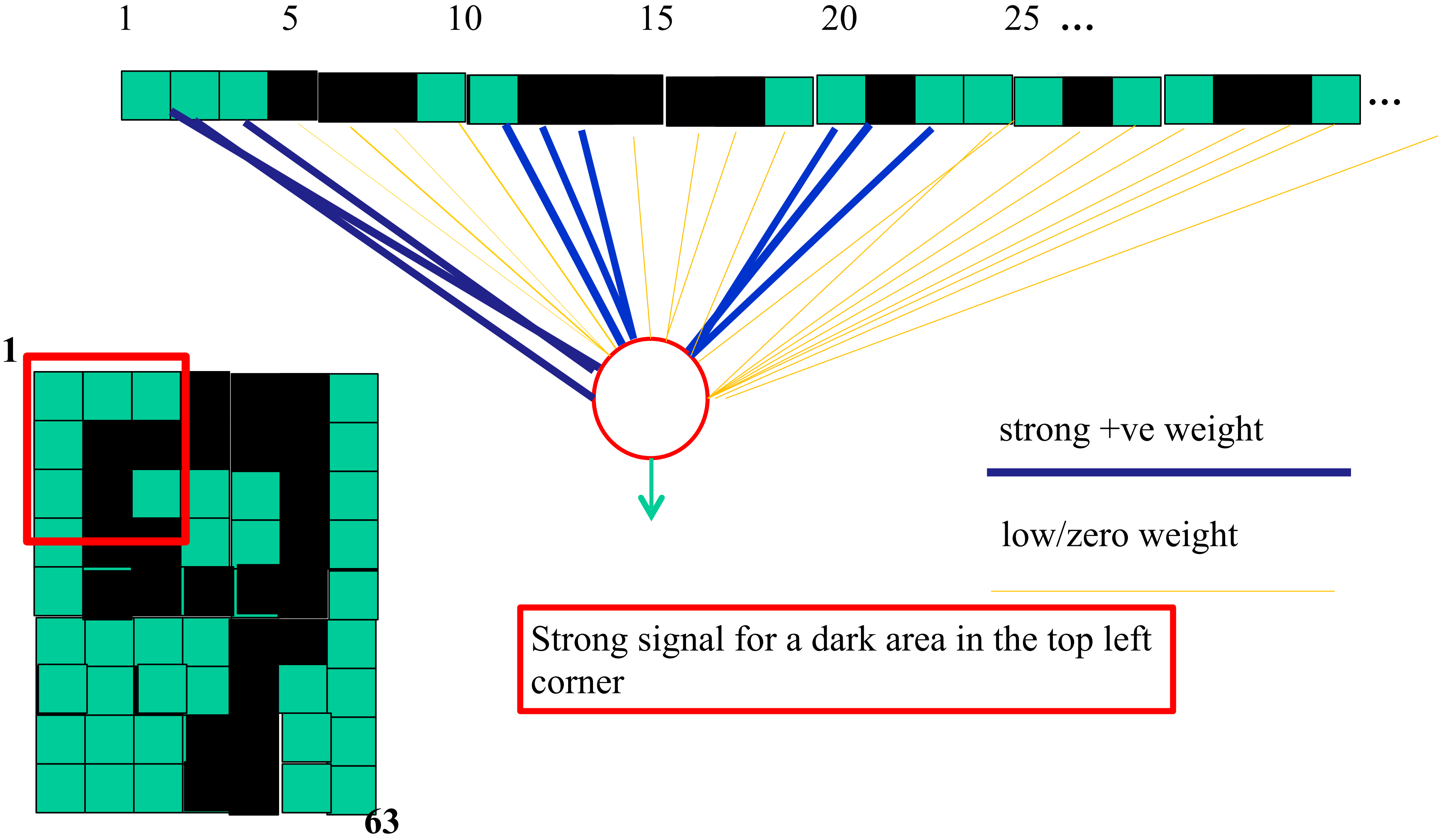


it will send strong signal for a horizontal line in the top row, ignoring everywhere else

what does this unit detect?



what does this unit detect?



Strong signal for a dark area in the top left corner

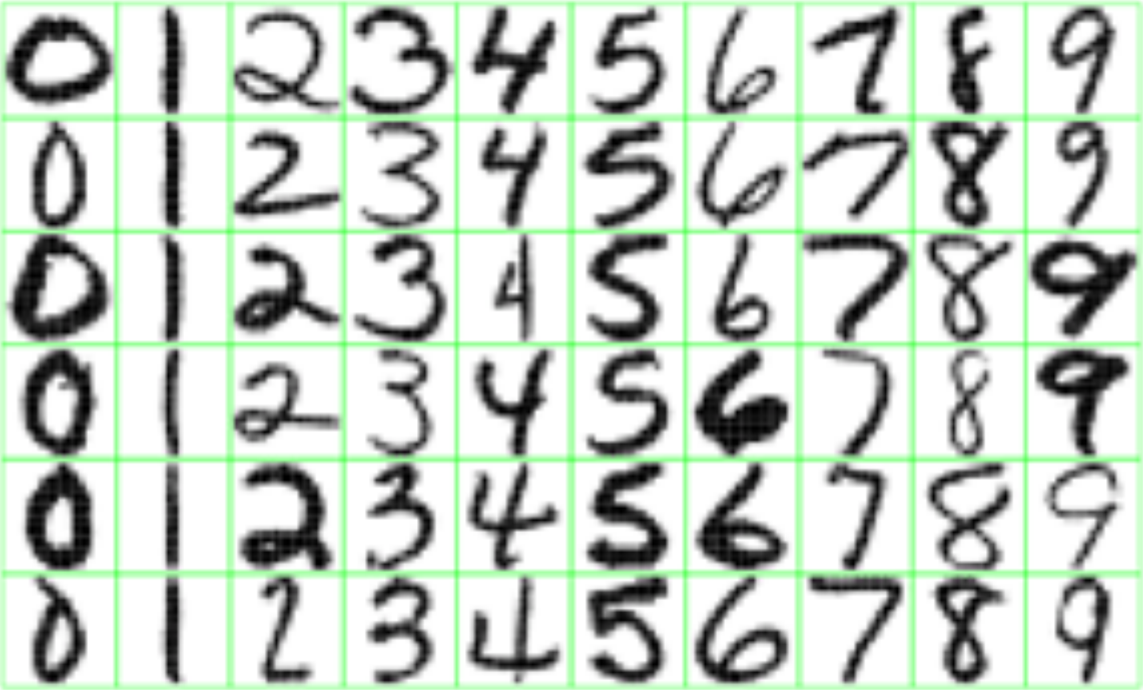


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

1

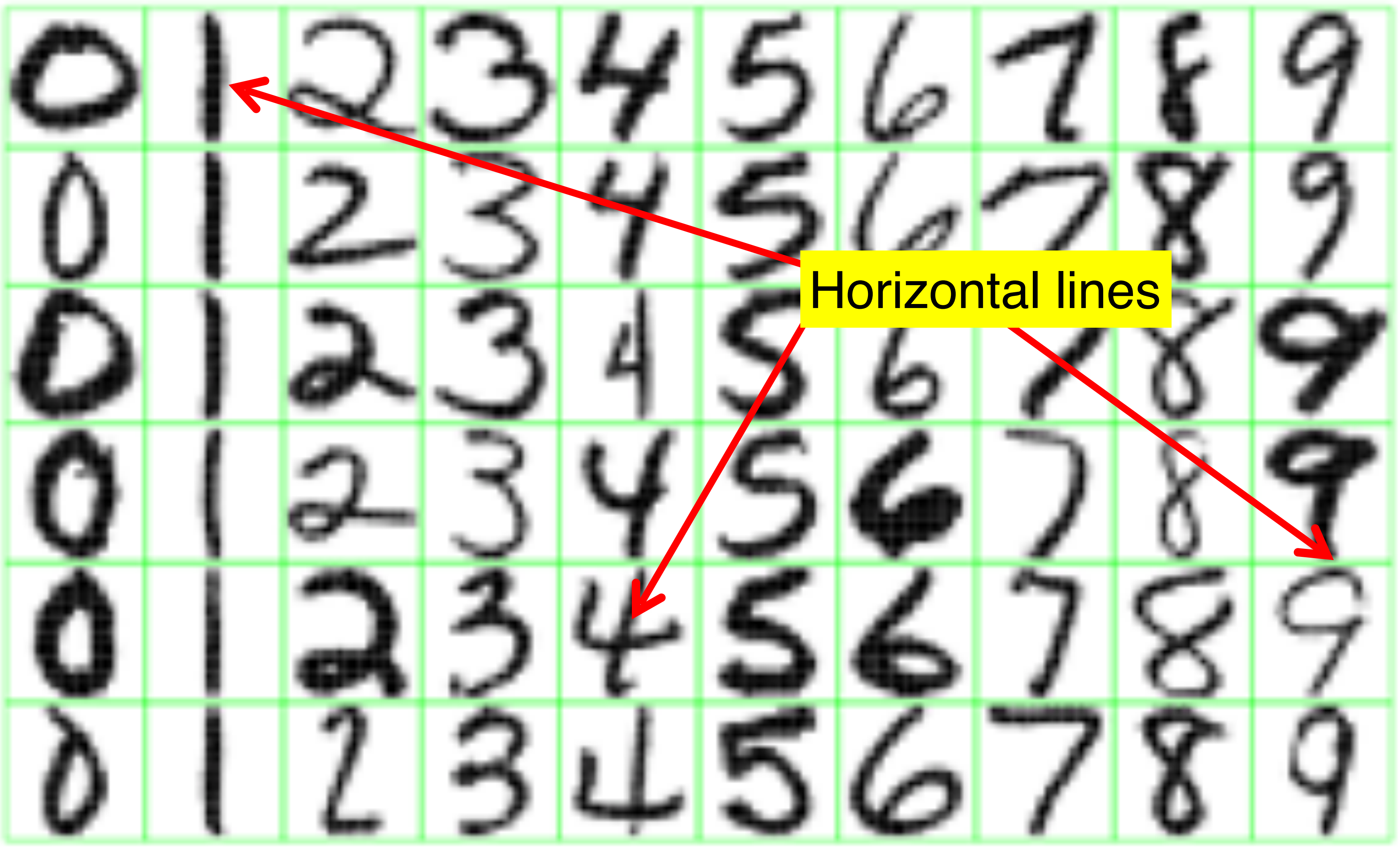


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

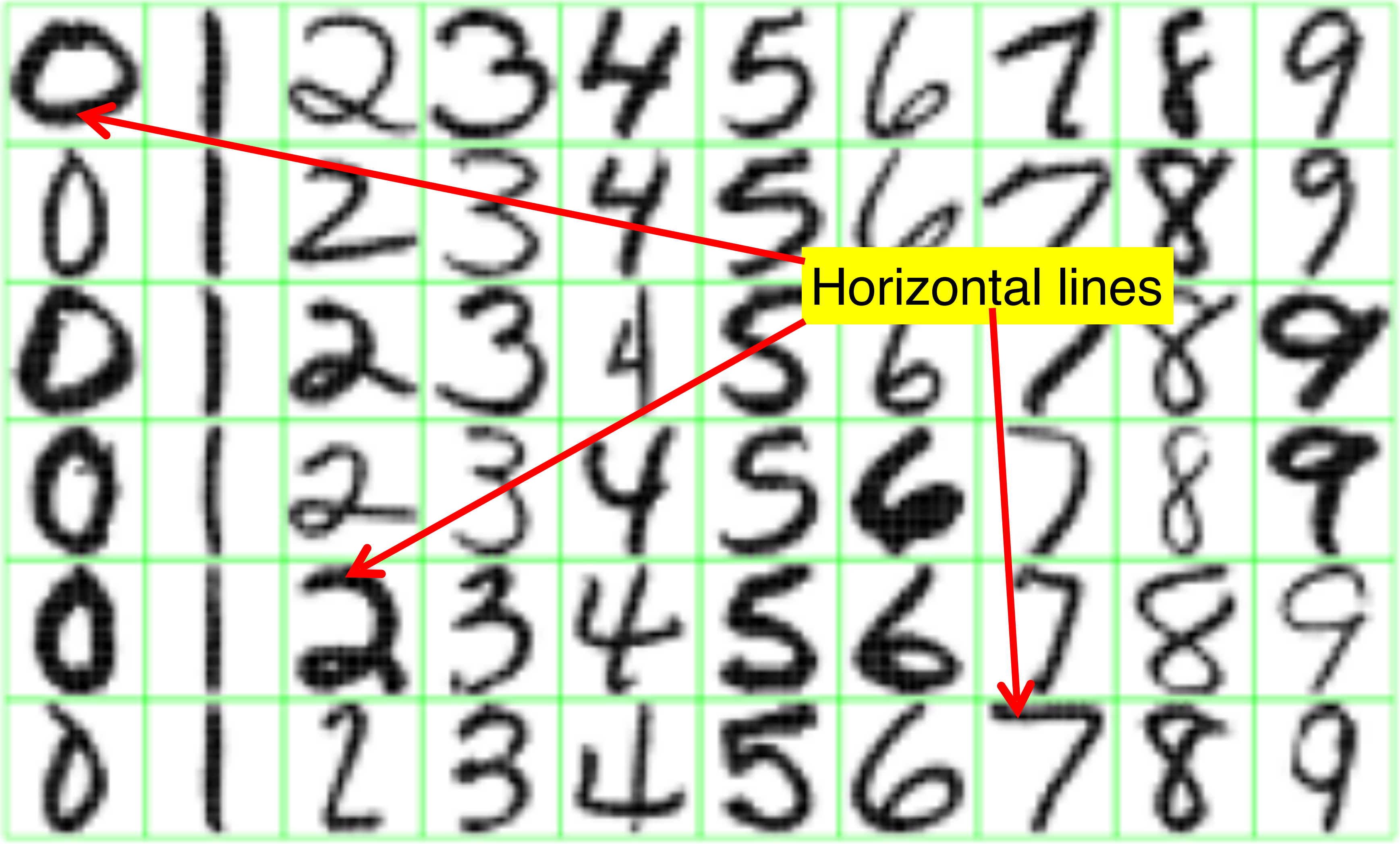


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

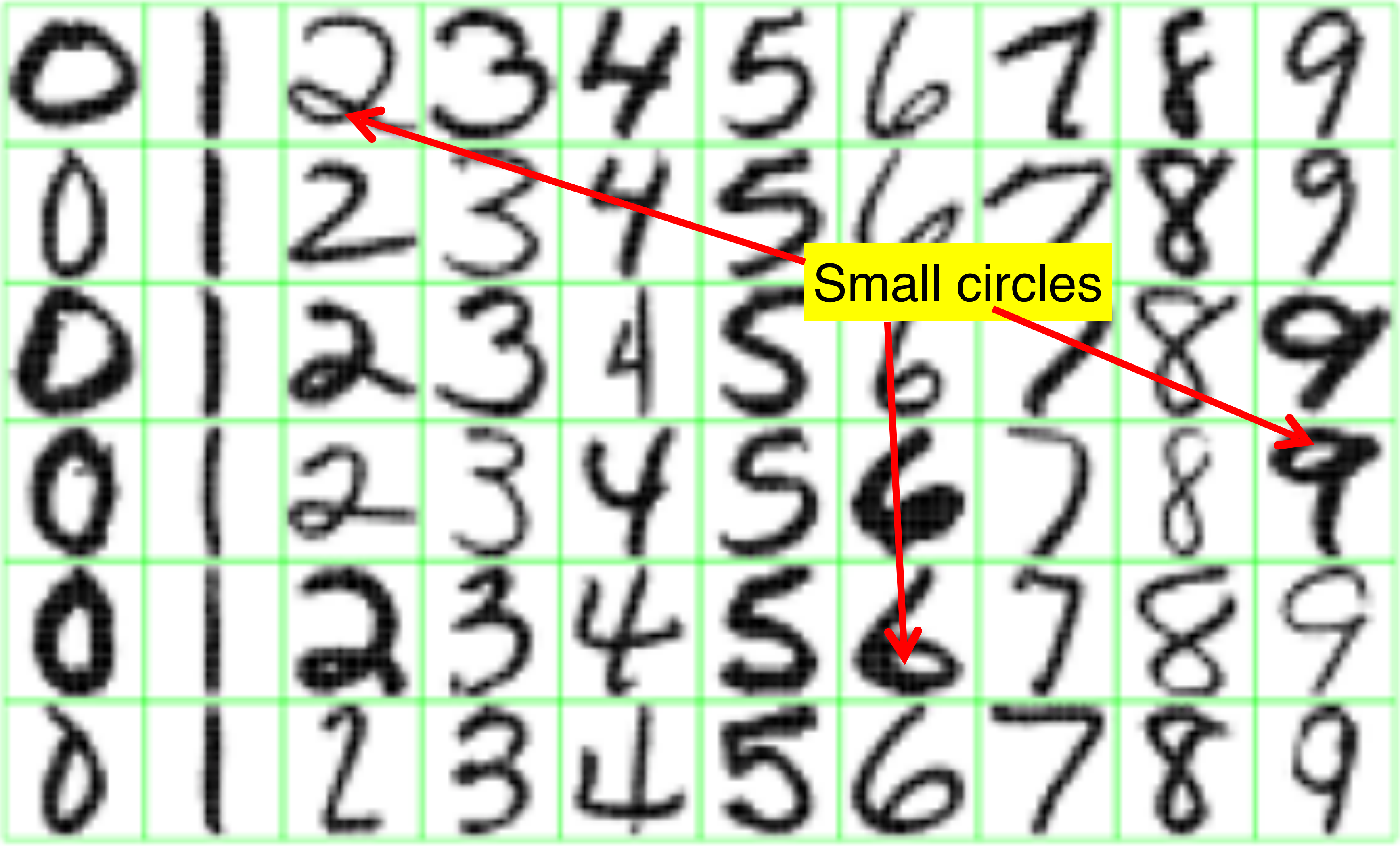
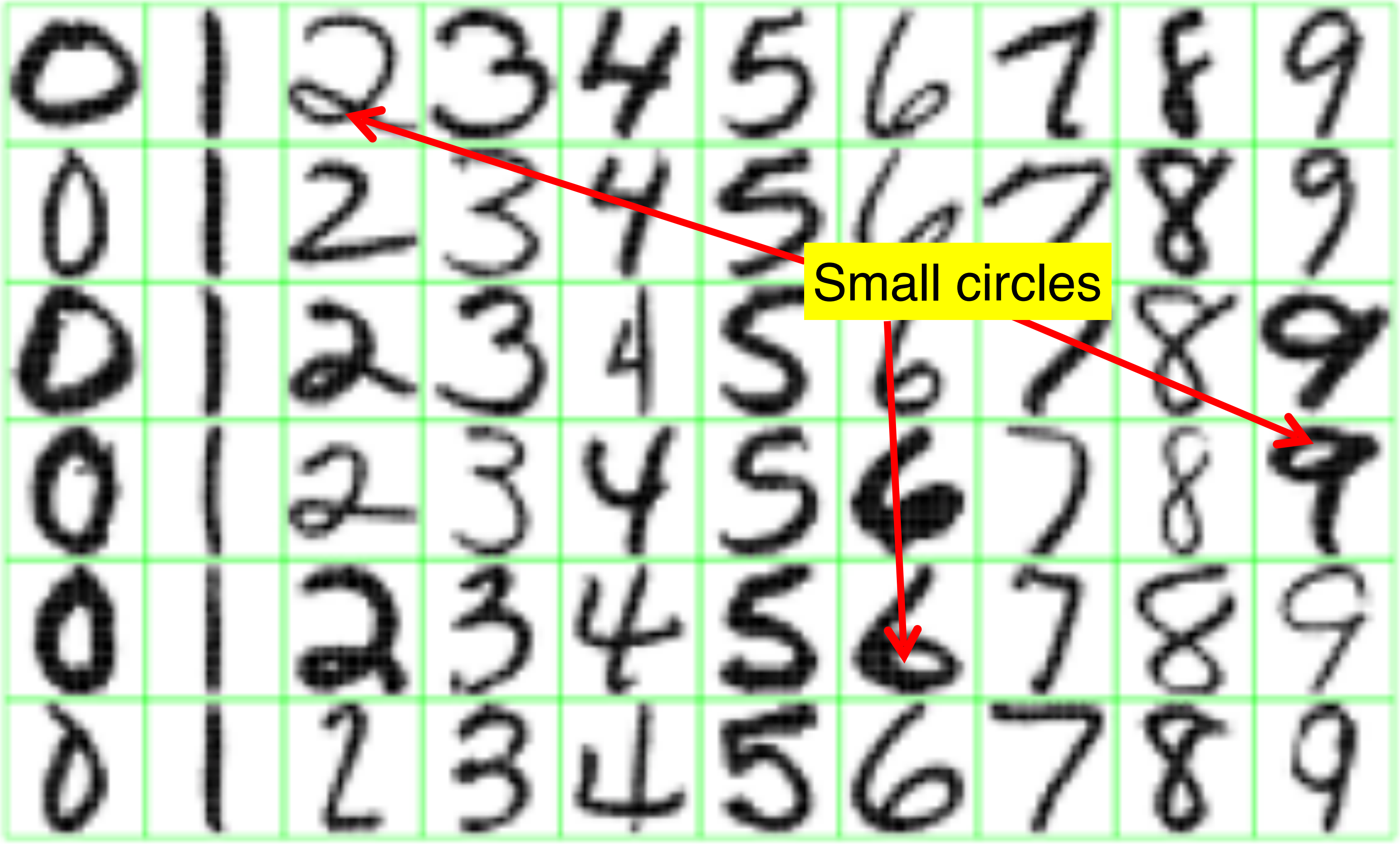

Small circles

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

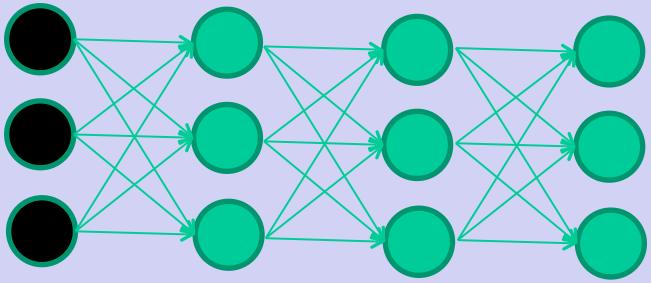
1



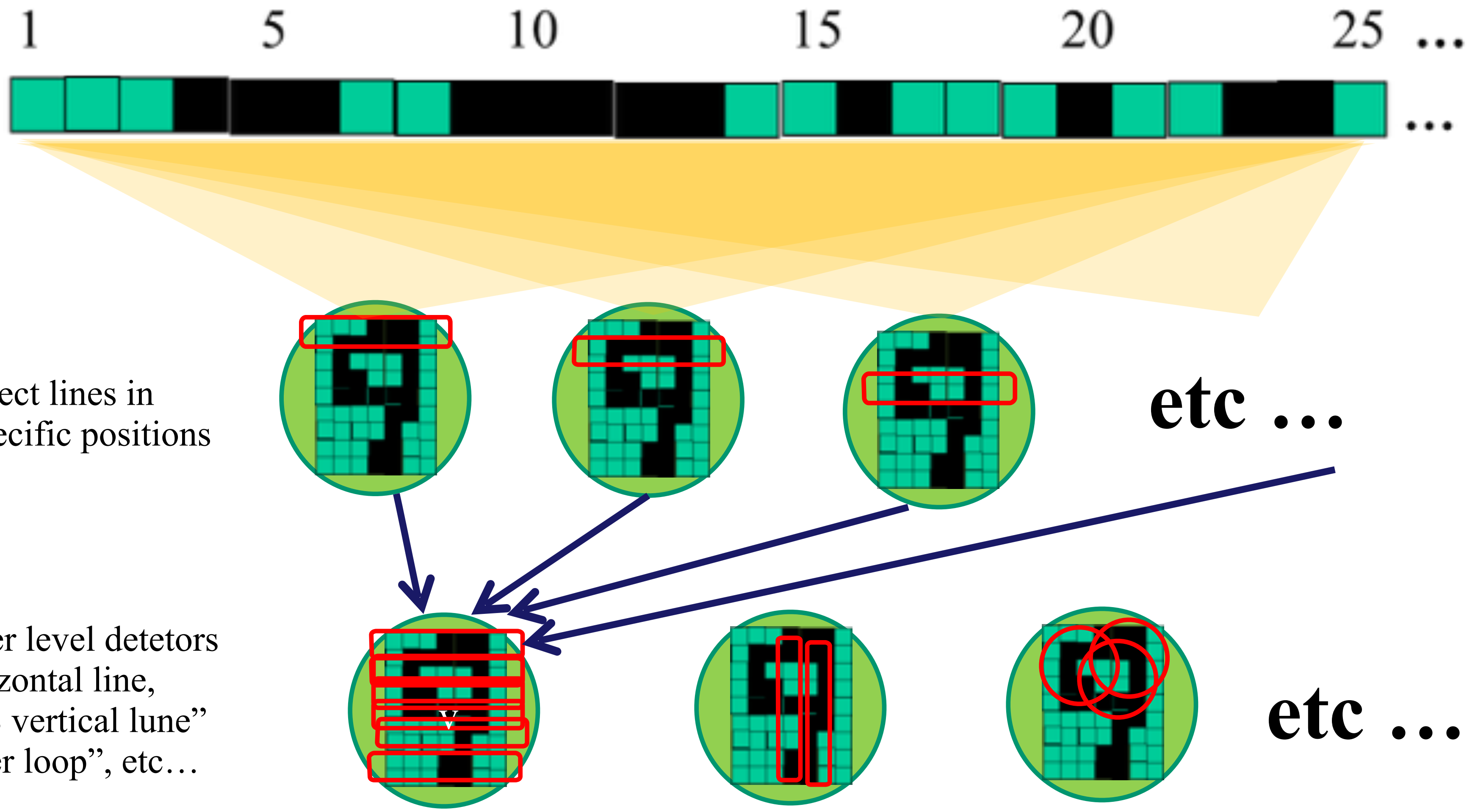
Small circles

But what about position invariance?
Our example unit detectors were tied to specific parts of the image

Deep Networks



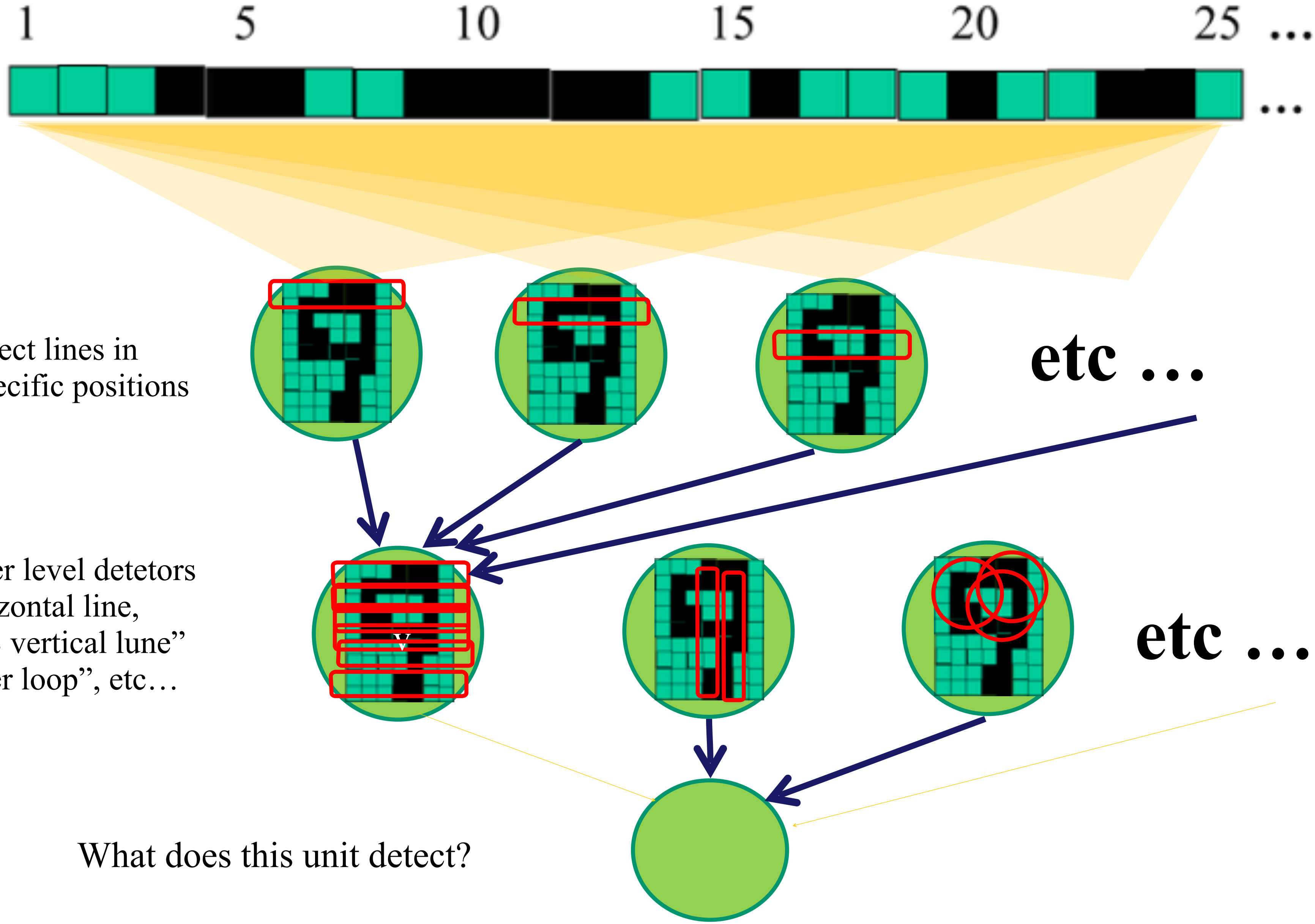
successive layers can detect higher-level features



detect lines in
Specific positions

Higher level detetors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc…

successive layers can detect higher-level features

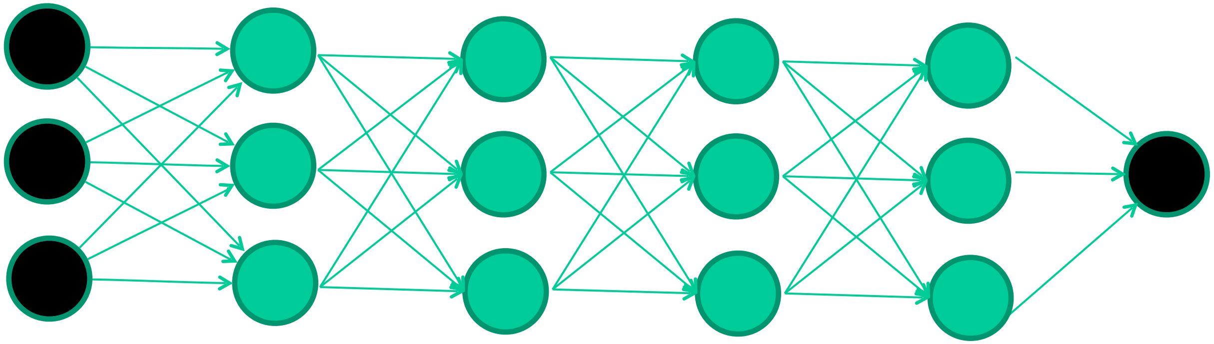


detect lines in
Specific positions

Higher level detetors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc…

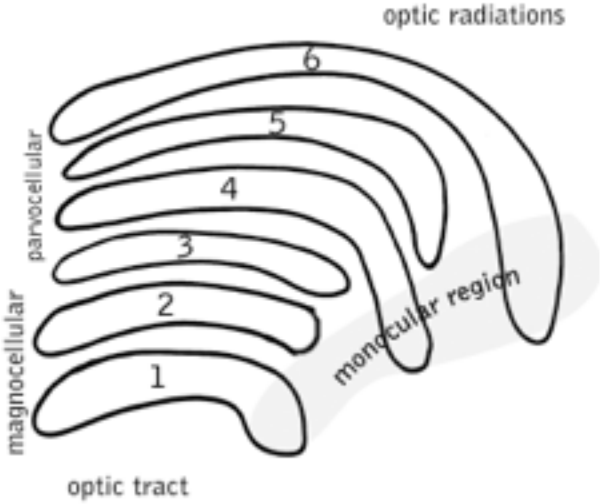
What does this unit detect?

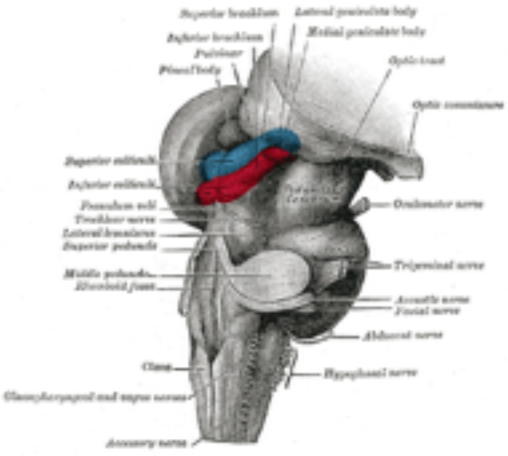
So: multiple layers make sense



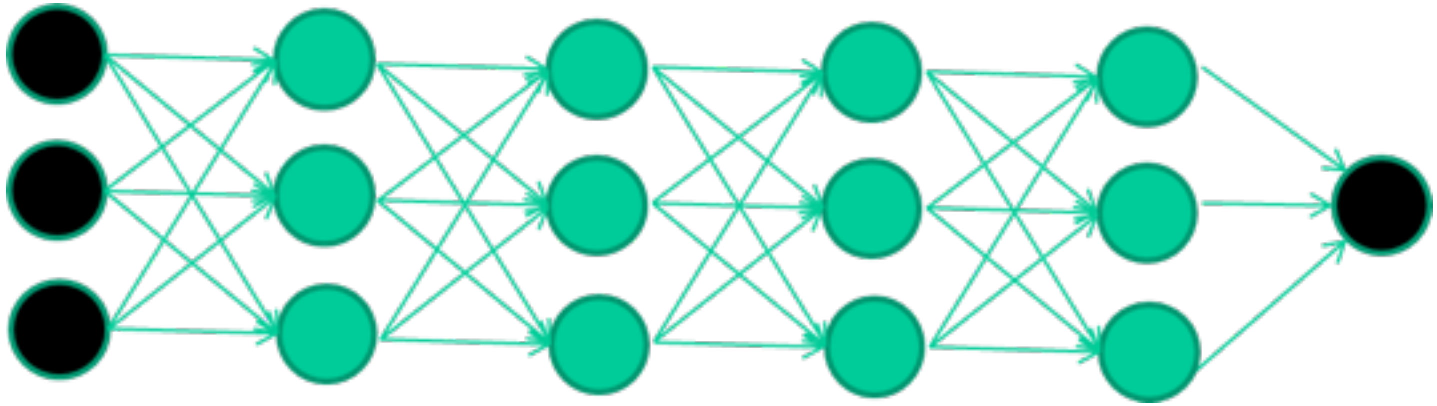
So: multiple layers make sense

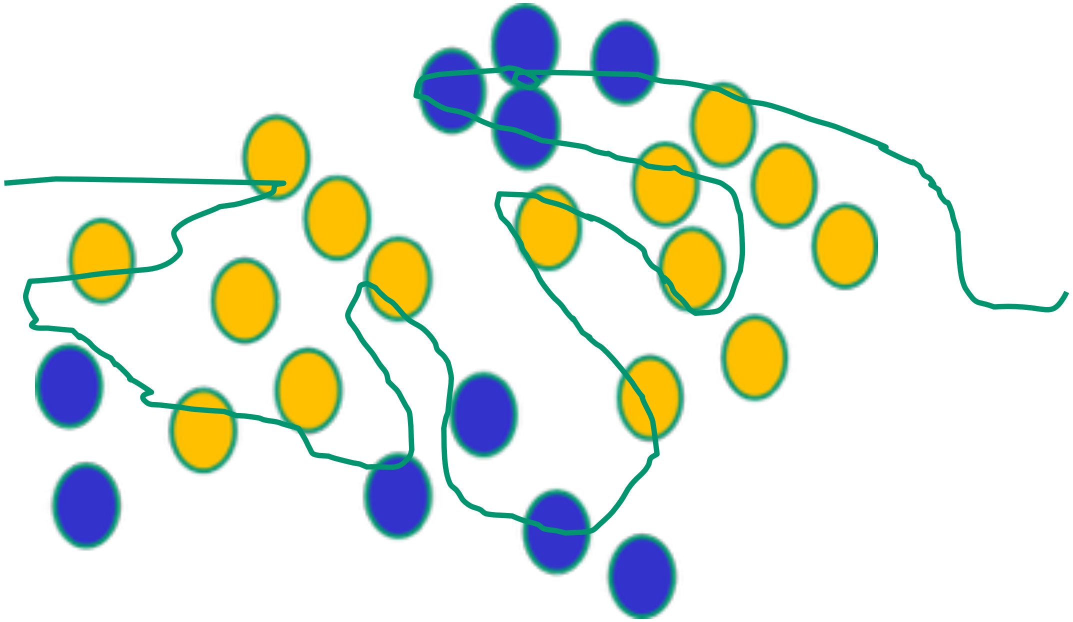
Multiple layers are also found in the brain, e.g. visual cortex





But: until recently deep networks could not be efficiently trained

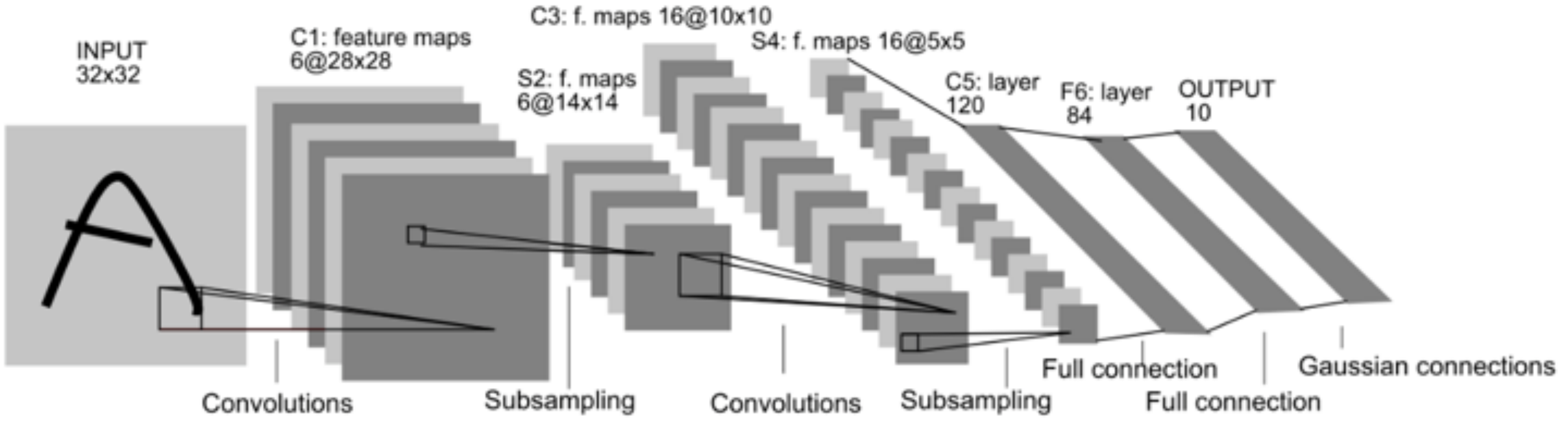




2006: The Deep Breakthrough

- Hinton, Osindero & Teh « A Fast Learning Algorithm for Deep Belief Nets », *Neural Computation*, 2006
- Bengio, Lamblin, Popovici, Larochelle « Greedy Layer-Wise Training of Deep Networks », *NIPS'2006*
- Ranzato, Poultney, Chopra, LeCun « Efficient Learning of Sparse Representations with an Energy-Based Model », *NIPS'2006*

Convolutional Neural Networks



Compared to standard feedforward neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train,
- while their theoretically-best performance is likely to be only slightly worse.

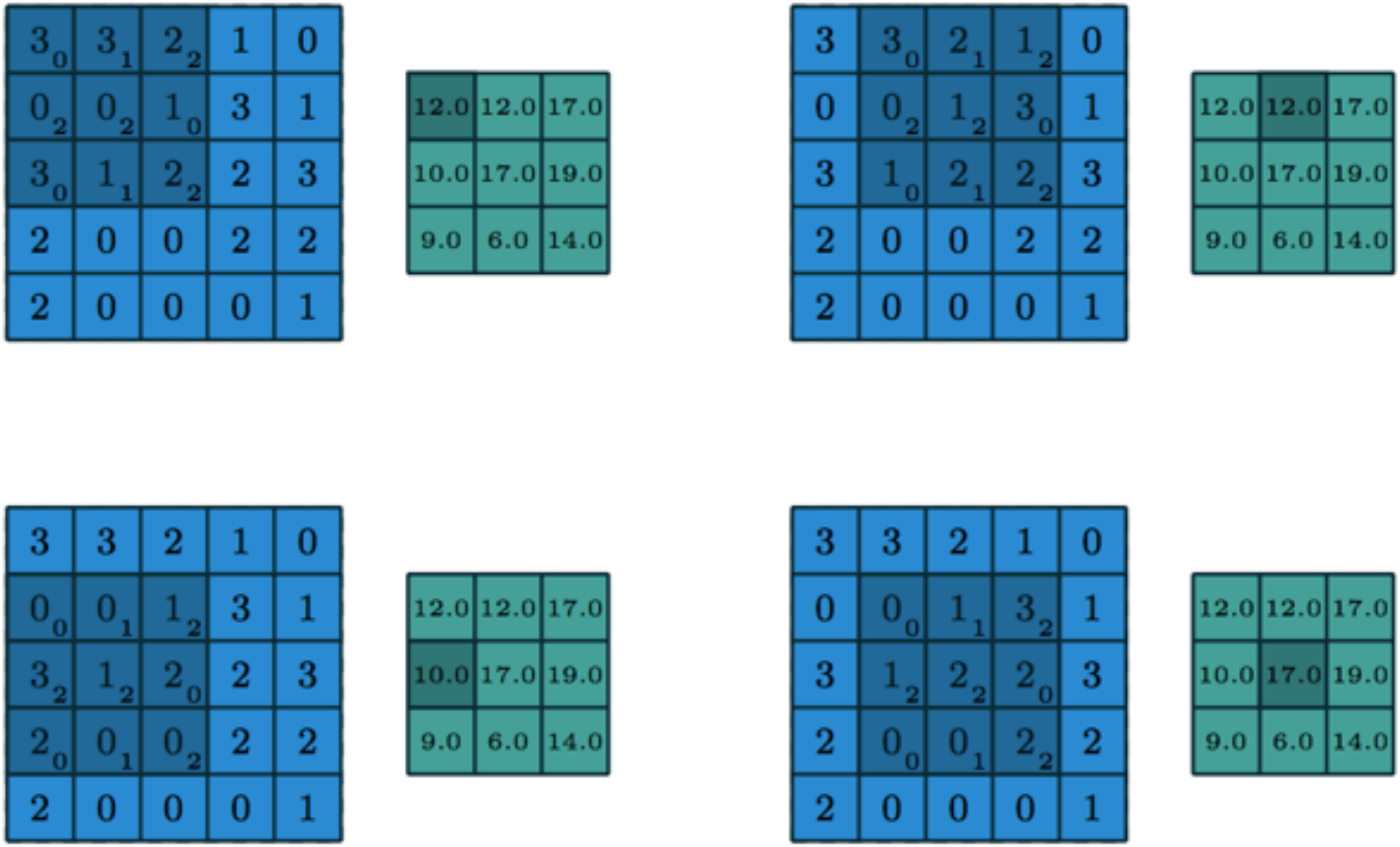
LeNet 5

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE, 86(11):2278-2324, November* ***1998***

Convolutional Kernel / Filter

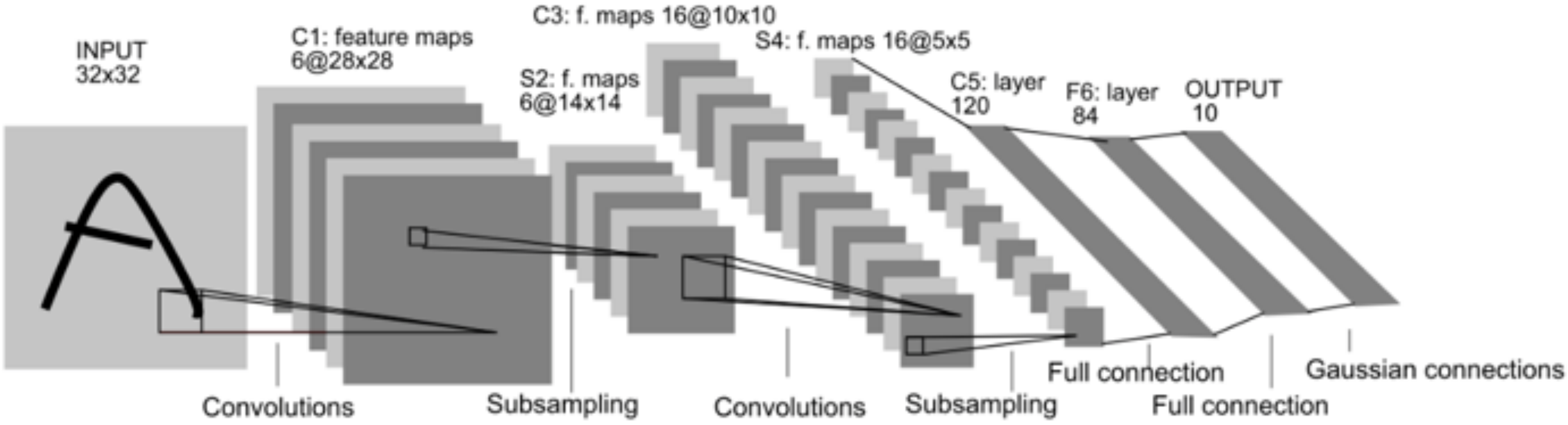
0	1	2
2	2	0
0	1	2

Apply convolutions



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

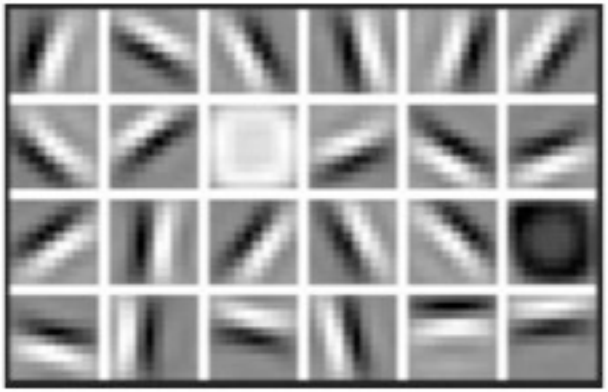


- Input: 32x32 pixel image. Largest character is 20x20
(All important info should be in the center of the receptive field of the highest level feature detectors)
- Cx: Convolutional layer
- Sx: Subsample layer
- Fx: Fully connected layer
- Black and White pixel values are normalized:
E.g. White = -0.1, Black =1.175 (Mean of pixels = 0, Std of pixels =1)

Convolutional filters perform image processing

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Example: filters in face recognition



First Layer Representation

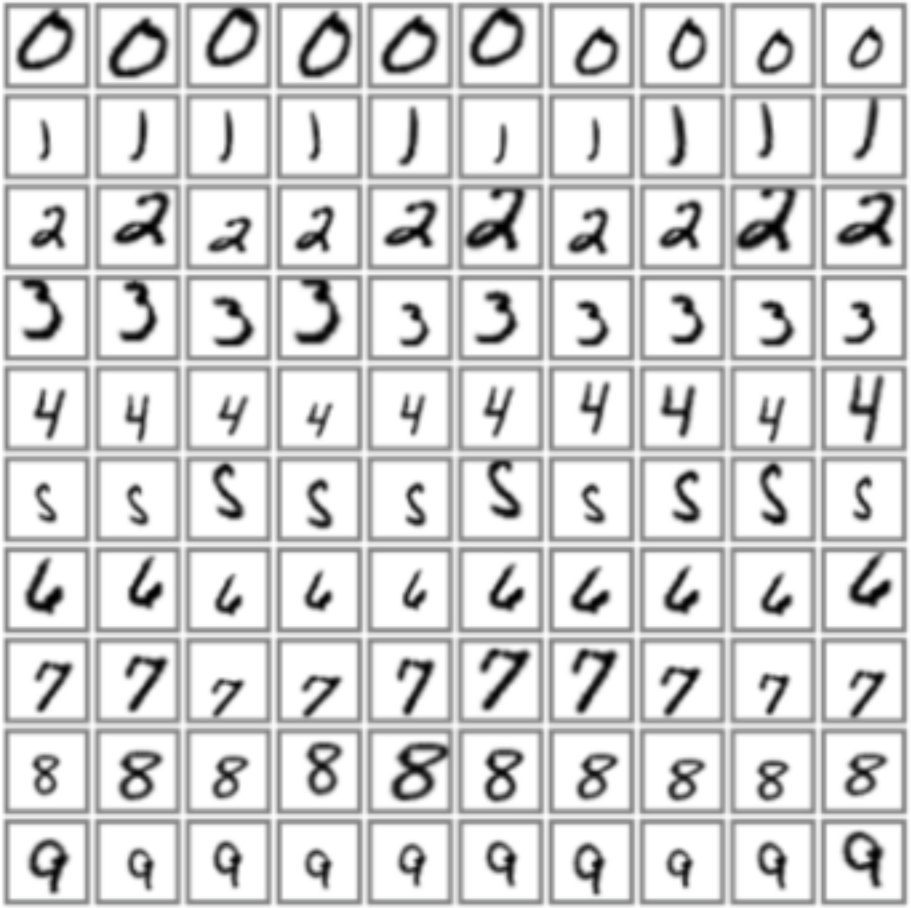
Second Layer Representation

Third Layer Representation

MNIST dataset

60,000 original datasets

Test error: 0.95%



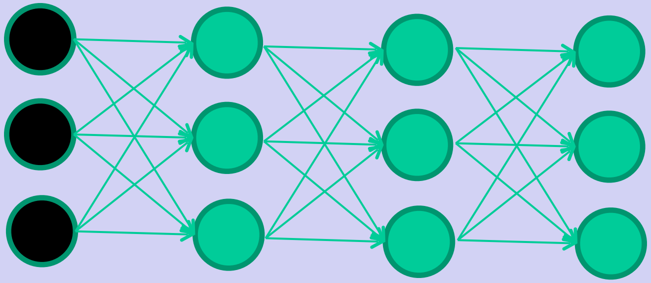
540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

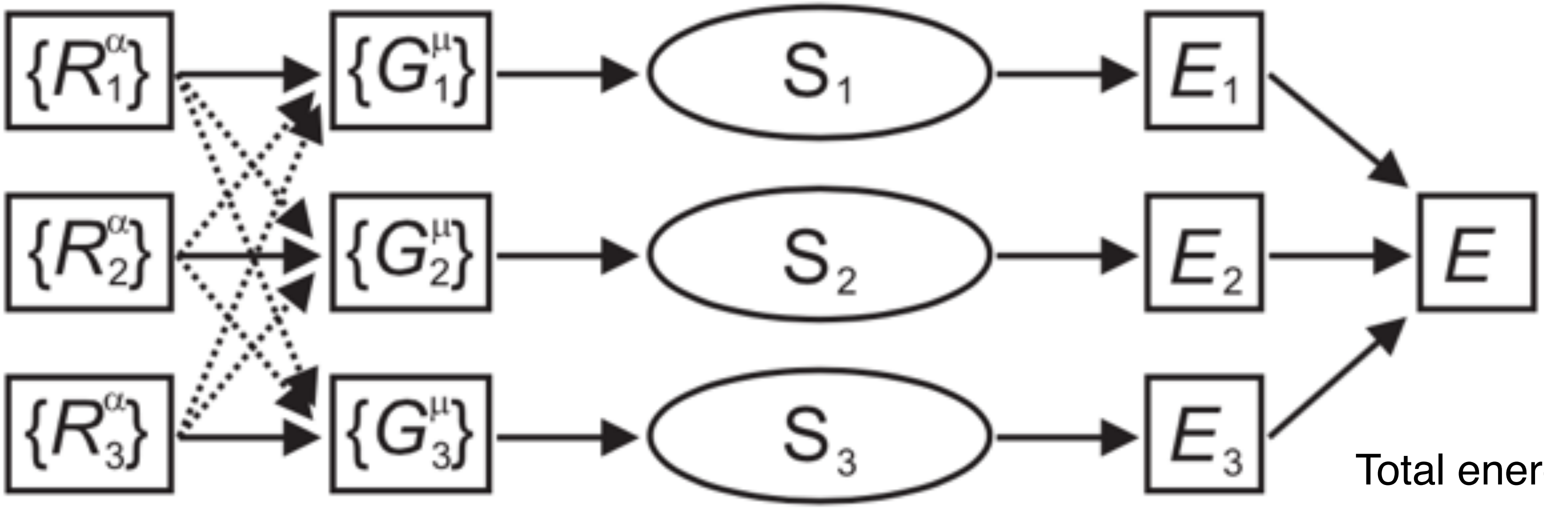
Misclassified examples

Applications to molecular systems



1) Learning to represent (effective) energy function

Behler-Parrinello network



Total energy

Cartesian coordinates

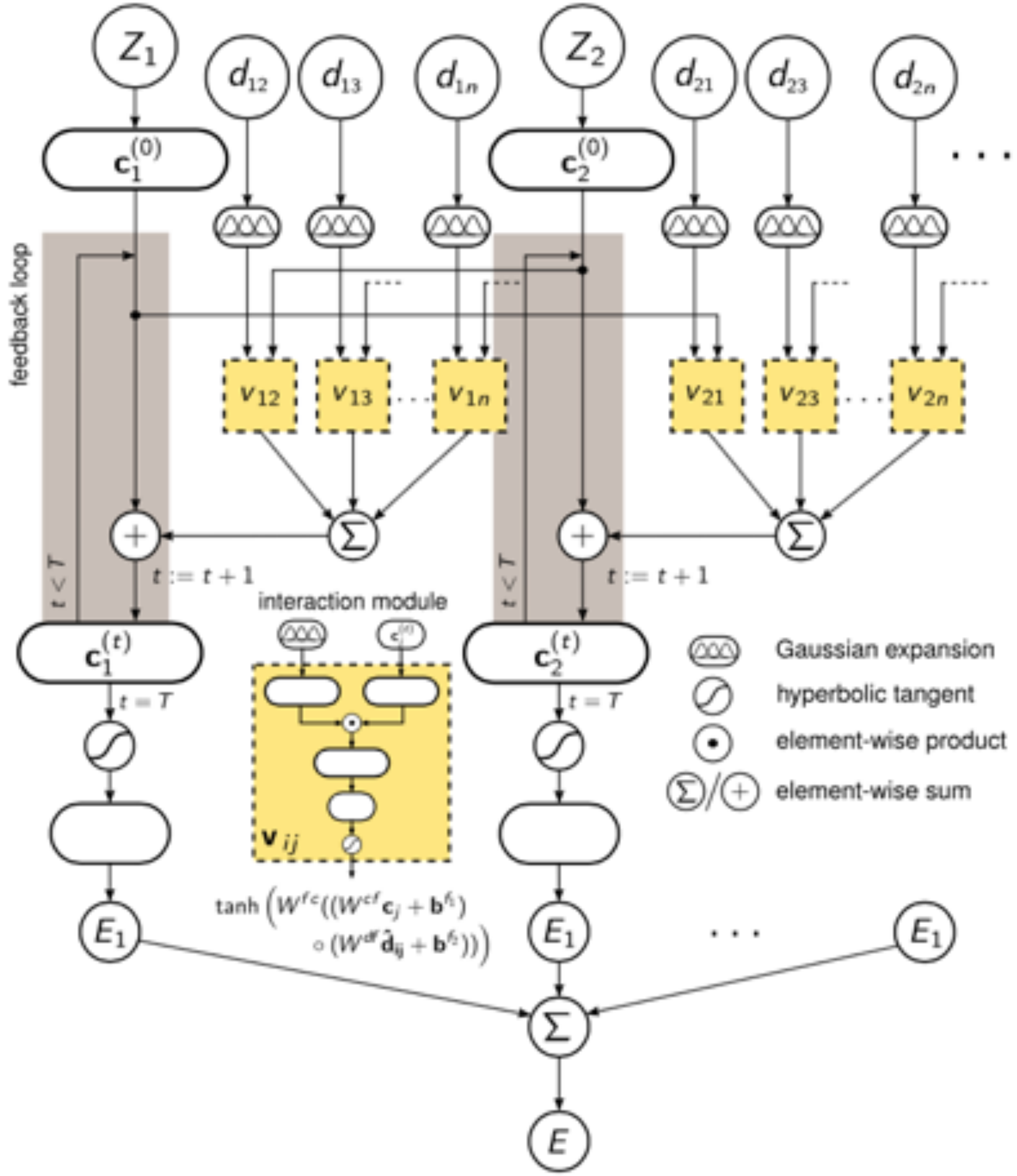
Internal coordinates

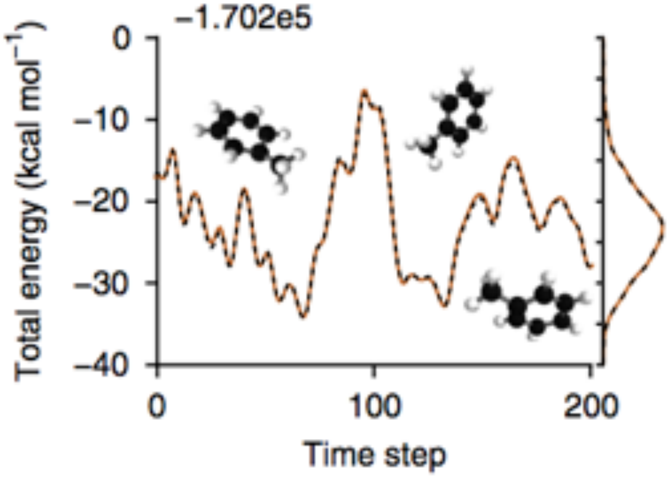
Neural networks (may be shared for same atom types)

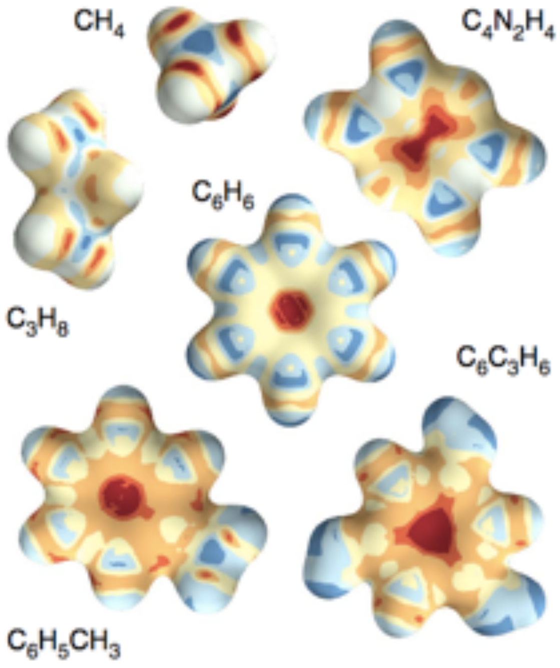
Atomic energies

Behler and Parrinello, PRL 98, 146401 (2007)

1) Learning to represent (effective) energy function

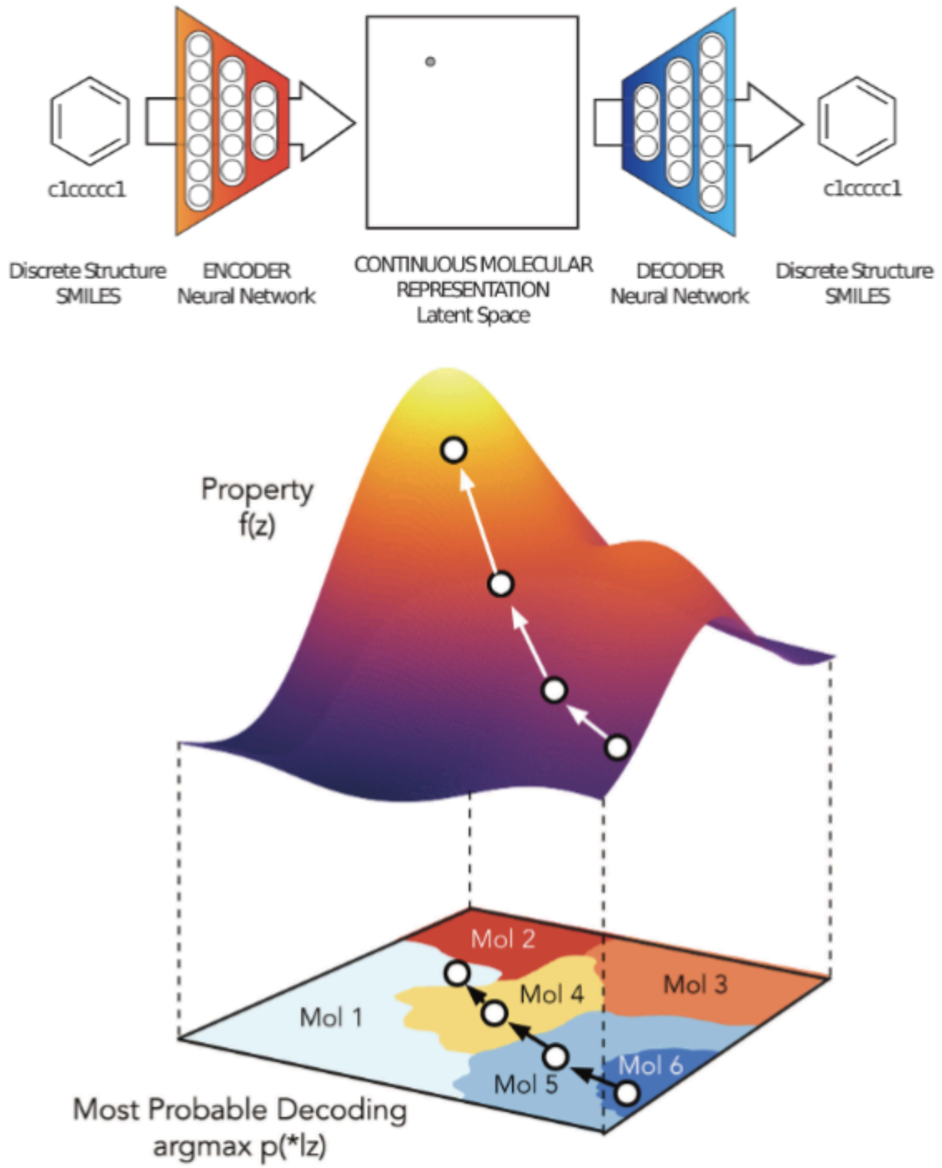






Schuett, Arbabzadah, Chmiela, Müller & Tkatchenko, Nature Communications 8, 13890 (2017)

2) Generator networks



Gómez-Bombarelli, …, Aspuru-Guzik: Automatic Chemical Design using Variational Autoencoders (2016)

2) Generator networks

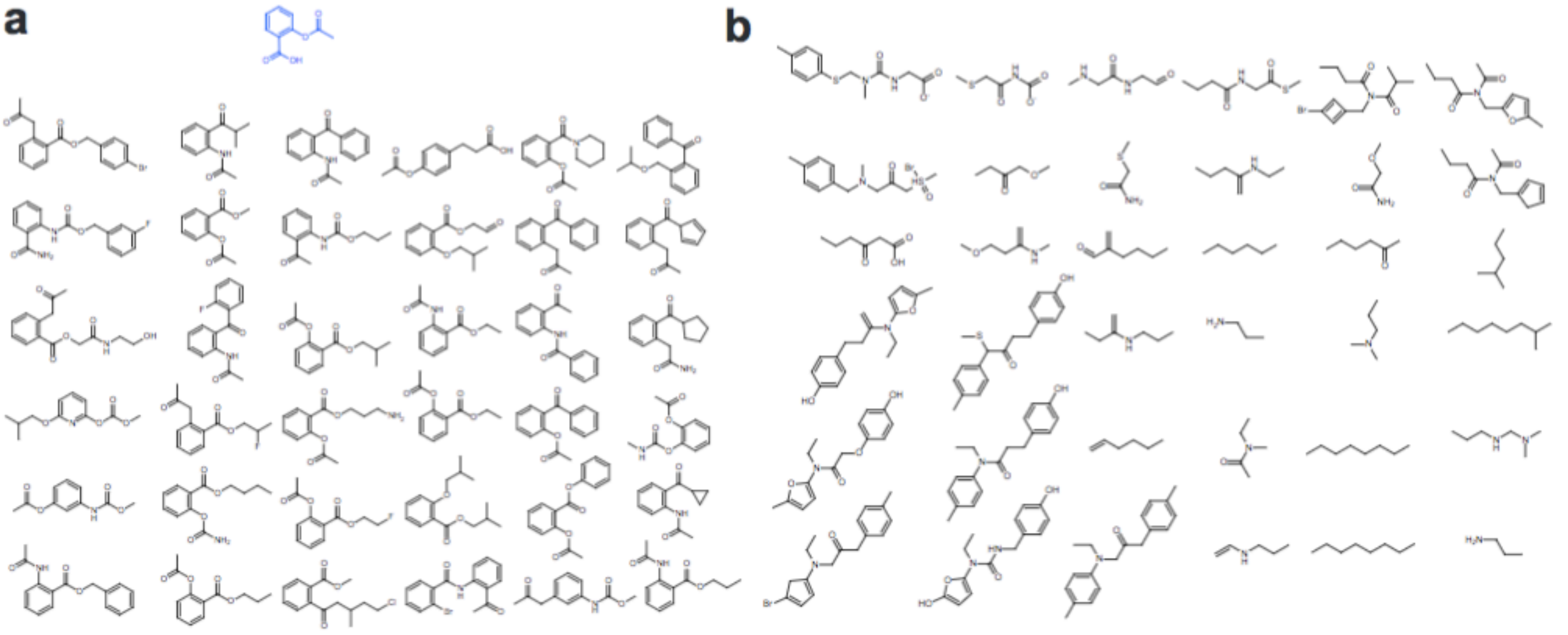
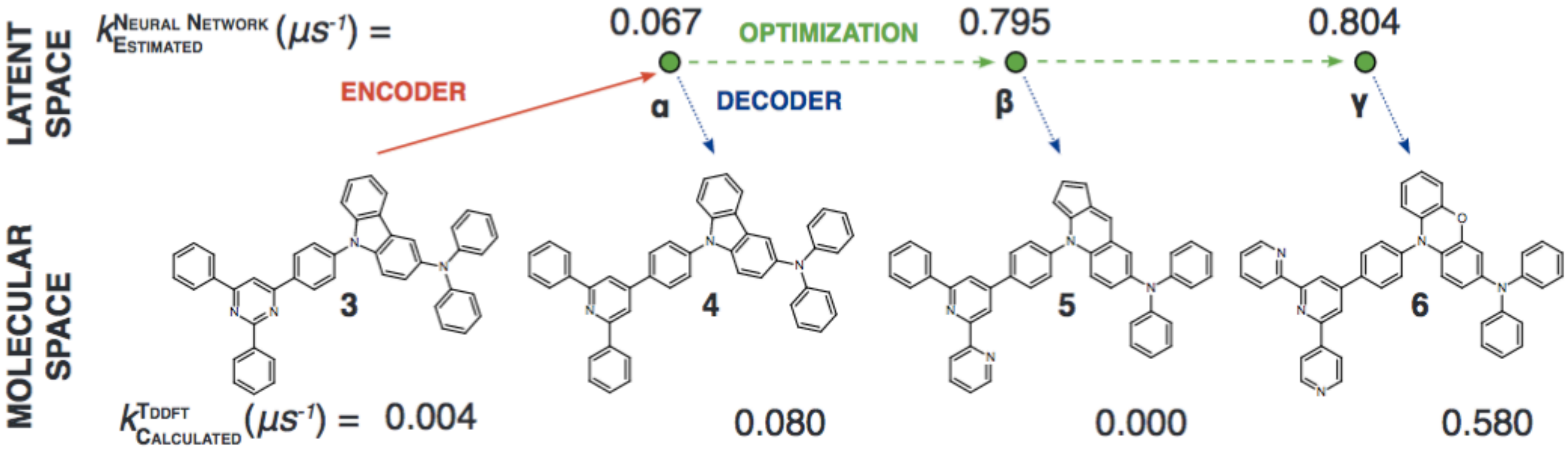


Figure 3: **a)**. Random sampling. Molecules decoded from randomly-sampled points in the latent space of a variational autoencoder, near to a given molecule (aspirin [2-(acetyloxy)benzoic acid], highlighted in blue). **b)**. Interpolation. Two-dimensional interpolation between four random points in in drug-like VAE. Decodings of interpolating linearly between the latent representations of the four molecules in the corners.

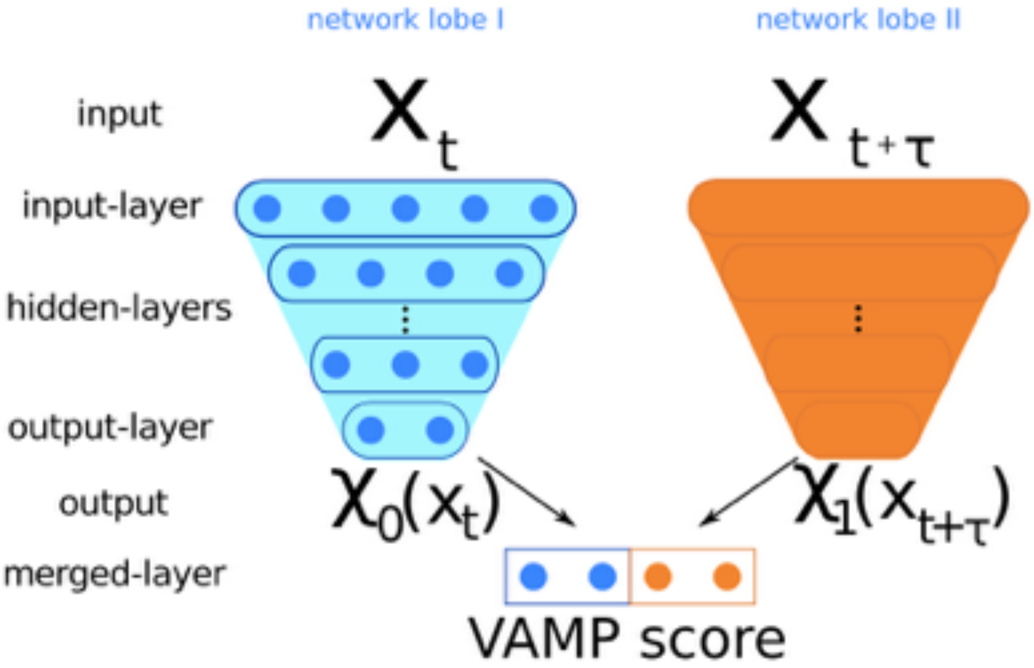
Gómez-Bombarelli, …, Aspuru-Guzik: Automatic Chemical Design using Variational Autoencoders (2016)

2) Generator networks



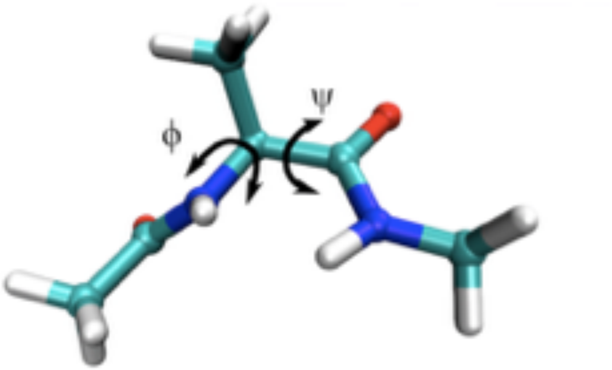
Gómez-Bombarelli, …, Aspuru-Guzik: Automatic Chemical Design using Variational Autoencoders (2016)

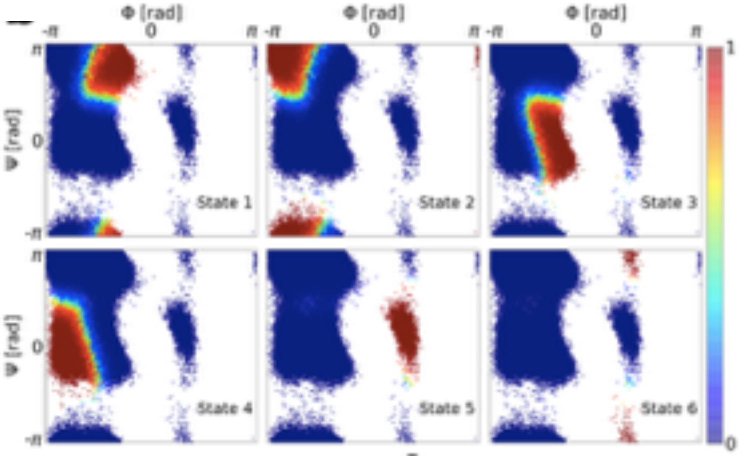
3) VAMPnets

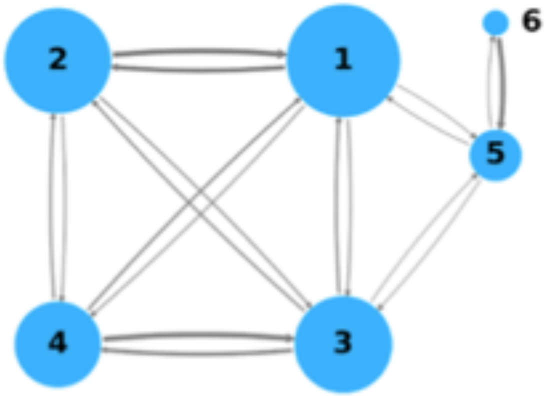


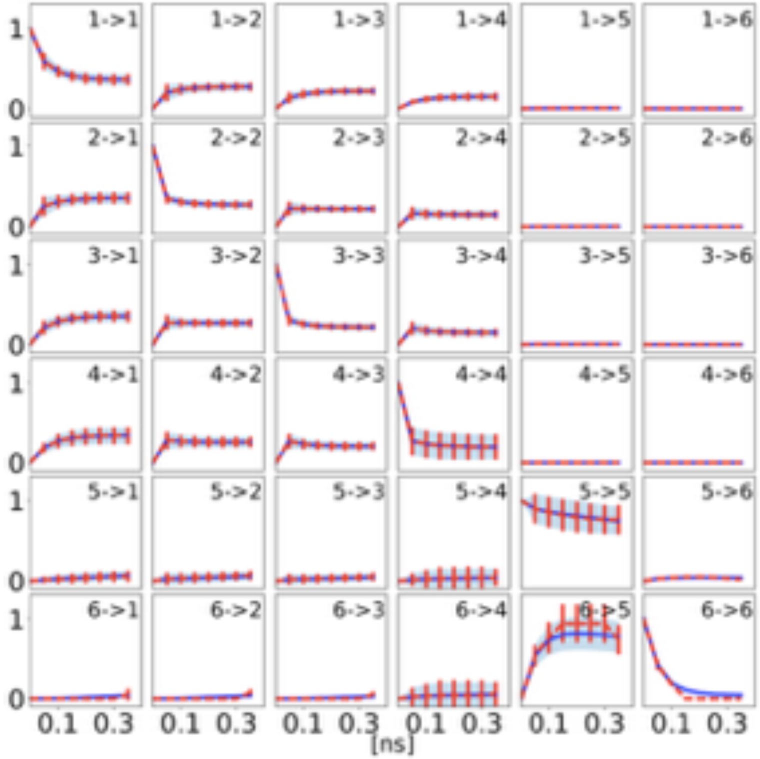
Mardt, Pasquali, Wu & Noé: VAMPnets - deep learning of molecular kinetics (2017)

3) VAMPnets









Mardt, Pasquali, Wu & Noé: VAMPnets - deep learning of molecular kinetics (2017)