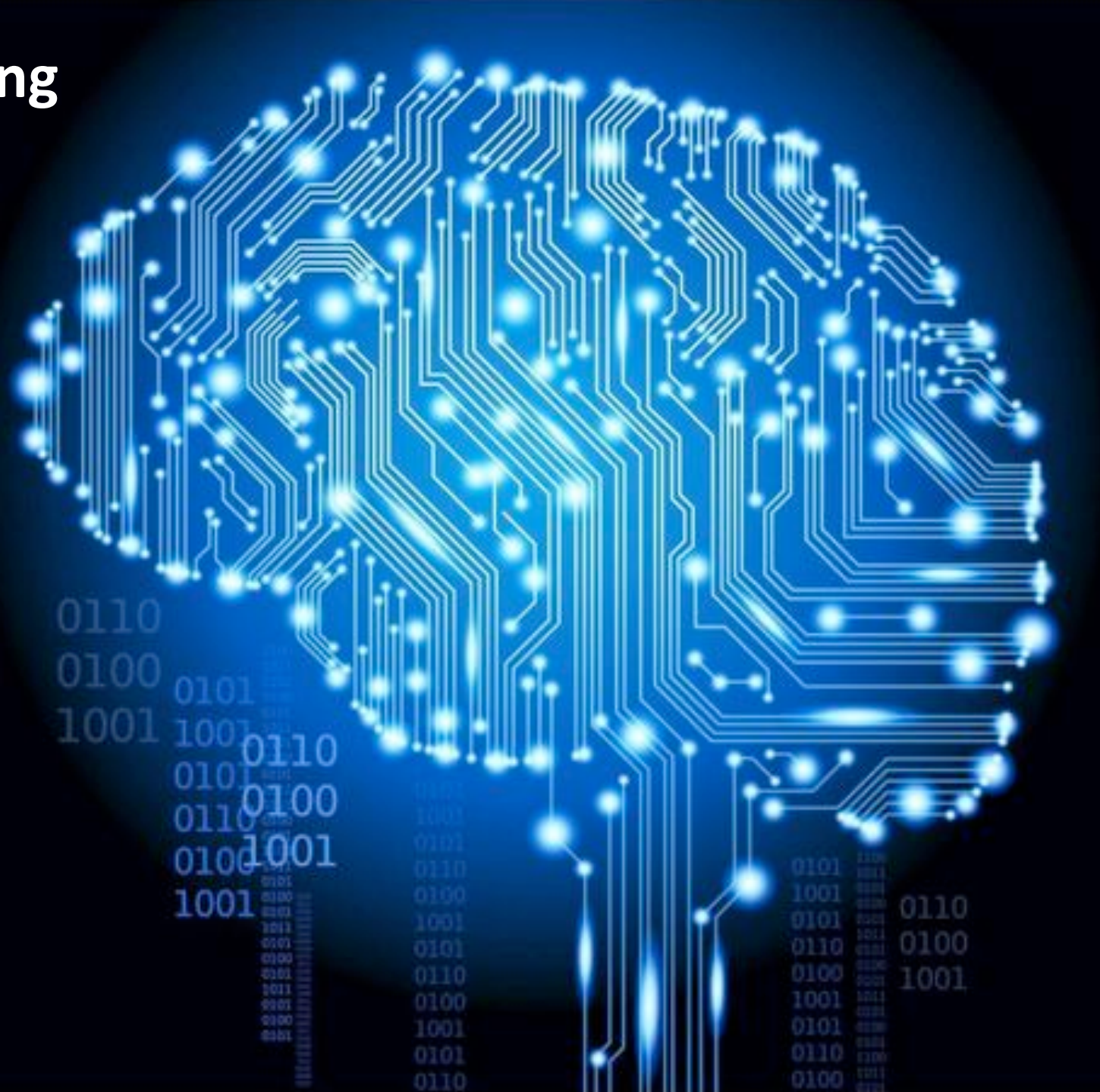
Deep learning

FU Berlin
Summer ’18

Frank Noé



Deep learning: Resources

1. Goodfellow, Courville and Bengio: *Deep Learning*, MIT Press 2016
   http://www.deeplearningbook.org/
   https://github.com/janishar/mit-deep-learning-book-pdf

2. P. Mehta, M. Bukov, C.-H. Wang, A.G.R. Day, C. Richardson, C.K. Fisher, D.J. Schwab: *A high-bias, low-variance introduction to Machine Learning for physicists*
   https://arxiv.org/abs/1803.08823

Deep learning: Registration and Questions

1. **Register** in the KVV / Eventline System:
   https://kvv.imp.fu-berlin.de
   Will be used for all communication
   (Emails, Lecture materials, homework, these slides)

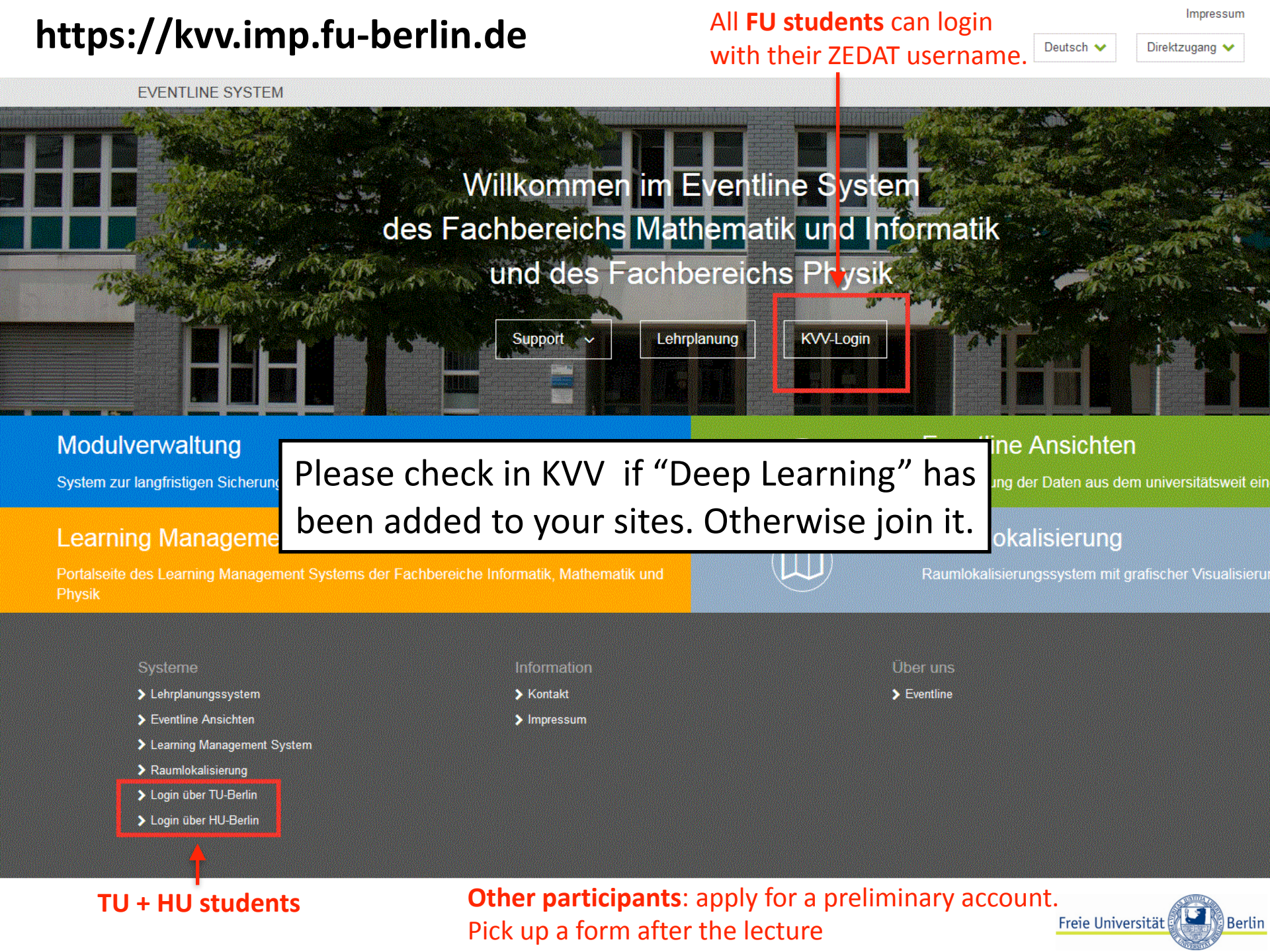
2. **Tutorials Registration**: on KVV, probably after Monday 12:00
   (Email will be sent to KVV-registered people)
   Do not register for tutorials if you do not need credit.
   Tutorials are limited to 35 (space).
   Wed 8-10, Wed 12-14, Thu 14-16. Room: Taku 9 / 049
   First come first serve.

3. **Questions**:
   On tutorials or organization:
   Christoph Wehmeyer - christoph.wehmeyer@fu-berlin.de

   On lecture content: to me after the lecture (not by Email!)

https://kvv.imp.fu-berlin.de

All **FU students** can login with their ZEDAT username.



Please check in KVV if "Deep Learning" has been added to your sites. Otherwise join it.

TU + HU students

Other participants: apply for a preliminary account.
Pick up a form after the lecture

Freie Universität Berlin

Deep learning: Tutorials

1. **Homework tutorials**
   - You can work together in groups of up to 3.
   - Submit results individually in KVV (initially, mostly multiple choice)
   - To secure the points, you should be able to show details in the tutorials.

2. **Computer tutorials**:
   - Homeworks will involve programming tasks later in the semester
   - Feasibility of live computer tutorials not yet clear.
   - If possible: small projects in larger groups.
     submission of code, automatic testing framework.

3. First worksheets: Fri, Apr 27.
   First submission deadline: Fri, May 4
   First tutorials: Week of May 7-11

4. Everyone is welcome to attend the lecture (space permitting)
   to use the lecture materials and do the homework for yourself -
   Our limitation is number and size of tutorials

Deep learning: Exam and Grade

1. **Grade**: Exam in the last semester week (No date set yet).
   Will be announced in KVV and course catalog.

2. FU Students who wish to continue and take exam have to sign up in Campus Management until Friday, May 4th.

3. After that students can manually register or un-register via the registrar's office / Prüfungsbüro.
   For forms see Prüfungsbüro website.

Requirements: in order to pass this class you will need…

1. Basics in linear algebra:
   - Have *routine* in doing **matrix-vector operations** and their properties
   - **Matrix decompositions** and properties (Eigenvalue d., Singular value d.)
   - see: Linear Algebra 1+2, Numerics 1
2. Basics in calculus:
   - **Multivariate calculus**, integration and differentiation, partial derivatives
   - Basics of **optimization**: Properties of minimum, maximum, saddle point
   - see: Calculus 1+2
3. Basics in statistics:
   - **Random variables**, **PDF**, CDF, **moments** and their properties.
   - **Transformations** between random variables, **Jacobians**.
   - see: Stochastics 1 or Statistical Physics 1
4. Basics in functional transforms:
   - **Fourier transform / DFT / FFT**.
5. Programming:

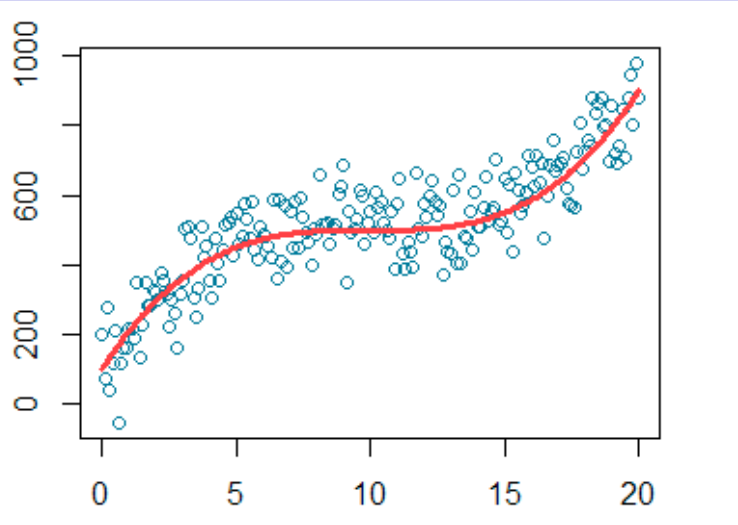
   Python. **Numpy**. **Scipy**. **Jupyter notebooks**. **Git**. **Github**

   - Check this to see if you are ready: https://github.com/cwehmeyer/scipro

Content

1. Introduction / Overview
2. Estimator Theory: Training/Test-set Bias/Variance.
3. Multilayer neural network, universal representation theorem, backprop.
4. Minimization methods.
5. Convolutional Neural Networks
6. Autoencoder versus principal component analysis
7. Time-autoencoder versus time-lagged independent component analysis
8. Generative learning: Variational Autoencoders and Generative Adversarial Networks
9. Probability measures and learning distributions.
10. Recurrent neural networks
11. Active and reinforcement learning

Linear least squares regression



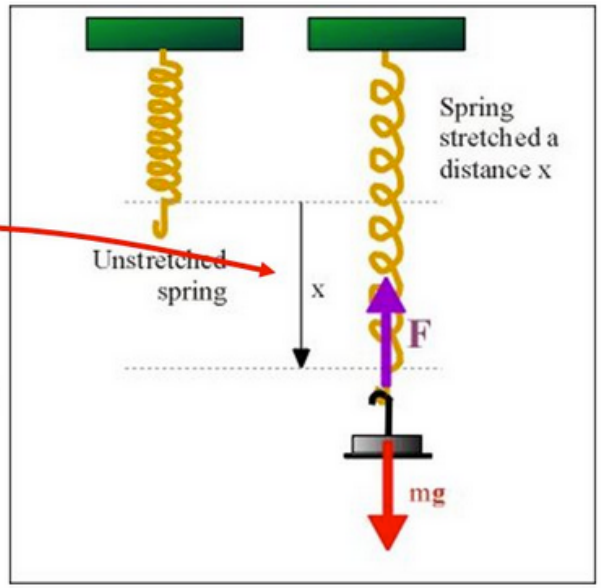
Linear example

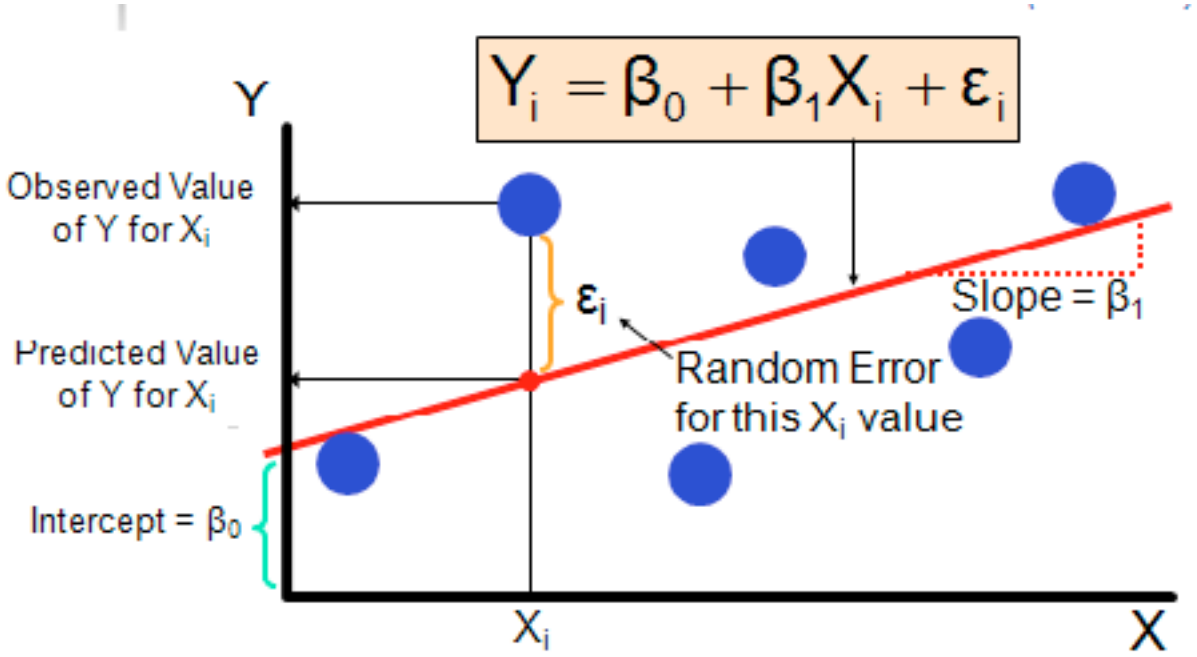
The spring force is given by

$$F_{spring} = kx$$

Where x is the amount that the spring stretched and k is the "spring constant"

which describes how stiff the spring is





Nonlinear functions with linear regression

Nonlinear function

$$E(l) = \frac{1}{2}kl^2$$

Linearize by mapping into **feature space x**

$$x = l^2$$

$$E(x) = \frac{1}{2}kx$$

—> still linear regression

Example: linear least squares

We are given a **training set** of m points (x_i, y_i). In vector notation: $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$.
We seek model equations that describe the data *approximately*:

$$y_i \approx f(x_i;\ \mathbf{w})$$

where $\mathbf{w} \in \mathbb{R}^n$ are parameters we need to **learn**.
Minimize the residuals $\boldsymbol{\Delta}$:

$$\Delta_i = y_i - f(x_i;\ \mathbf{w}).$$

In a linear regression problem, the model equation has the form:

$$f(x_i;\ \mathbf{w}) = w_1\phi_1(x_i) + \ldots + w_n\phi_n(x_i)$$

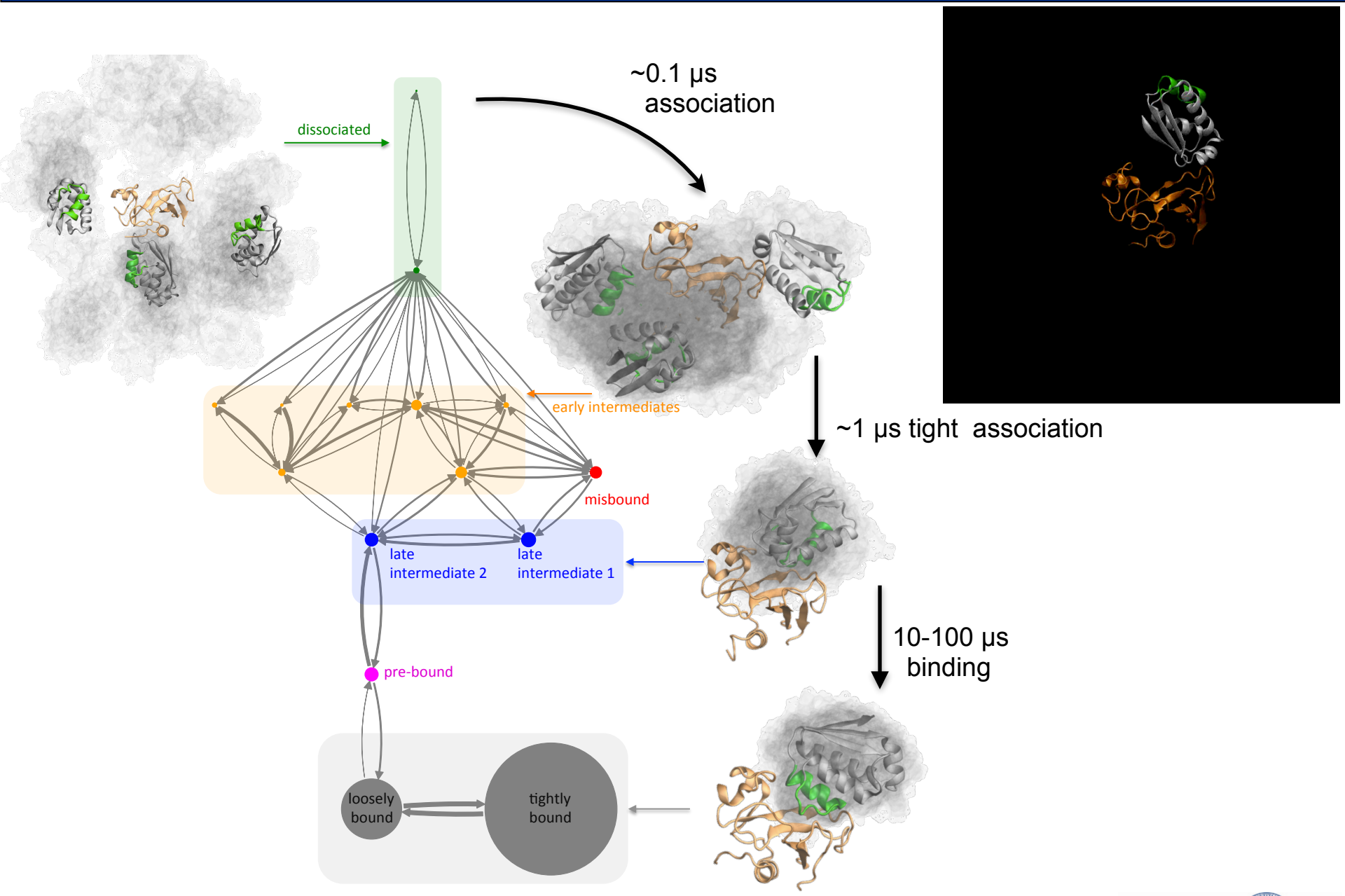
where $\phi_j : \mathbb{R} \to \mathbb{R}$ are arbitrary **basis functions** or **feature functions** that can be nonlinear in x.
Define the matrix $\mathbf{X}$

$$X_{ij} = \phi_j(x_i),$$

to rewrite the linear least squares regression problem as:

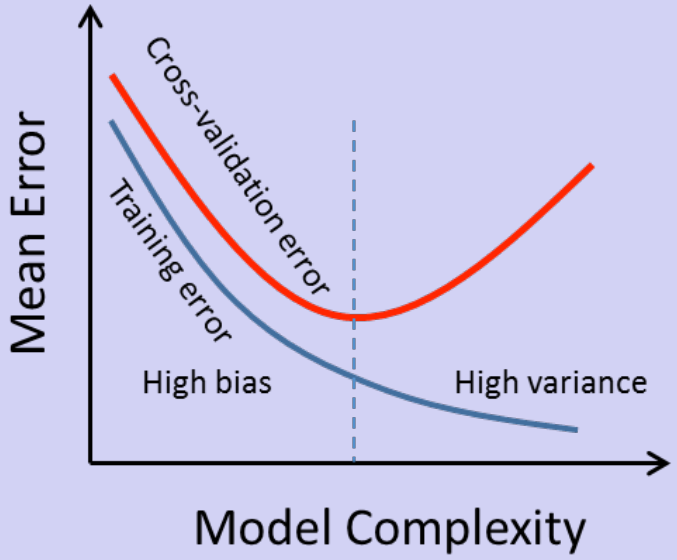
$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2$$

Non-obvious Example: Markov state model



Plattner, Doerr, De Fabritiis, Noé **Nature Chemistry** 2017

Validation, cross-validation and hyperparameter selection



Validation, cross-validation and hyperparameter selection

- *Validation*: LLS solution fives us the **training error** $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2$, but we would like to validate how good the learnt model **predicts** independent **test data**.

- *Hyperparameter selection*: Hyperparameters are parameters that cannot not be obtained from the learning algorithm (here LLS).
  Example: The type of function ϕ used for training cannot be determined by minimizing the training error. For example, the function

$$f(x) = \sum_{i=1}^{N} w_i \delta(|x - x_i|)$$

  has a training error of zero, but predicts nothing ($f(x) = 0$ for every point x not in the training set).

Validation

Divide dataset into

- **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
- **test set** $(\mathbf{X}^{\text{test}}, \mathbf{y}^{\text{test}})$.

Validation

Divide dataset into

- **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
- **test set** $(\mathbf{X}^{\text{test}}, \mathbf{y}^{\text{test}})$.

Learn parameters using the training set:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

The resulting residual $\epsilon^{\text{train}} = \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$ is the **training error** or **training loss**.

Validation

Divide dataset into

- **training set** $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$
- **test set** $(\mathbf{X}^{\text{test}}, \mathbf{y}^{\text{test}})$.

Learn parameters using the training set:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$$

The resulting residual $\epsilon^{\text{train}} = \left\| \mathbf{y}^{\text{train}} - \mathbf{X}^{\text{train}} \mathbf{w} \right\|_2$ is the **training error** or **training loss**.

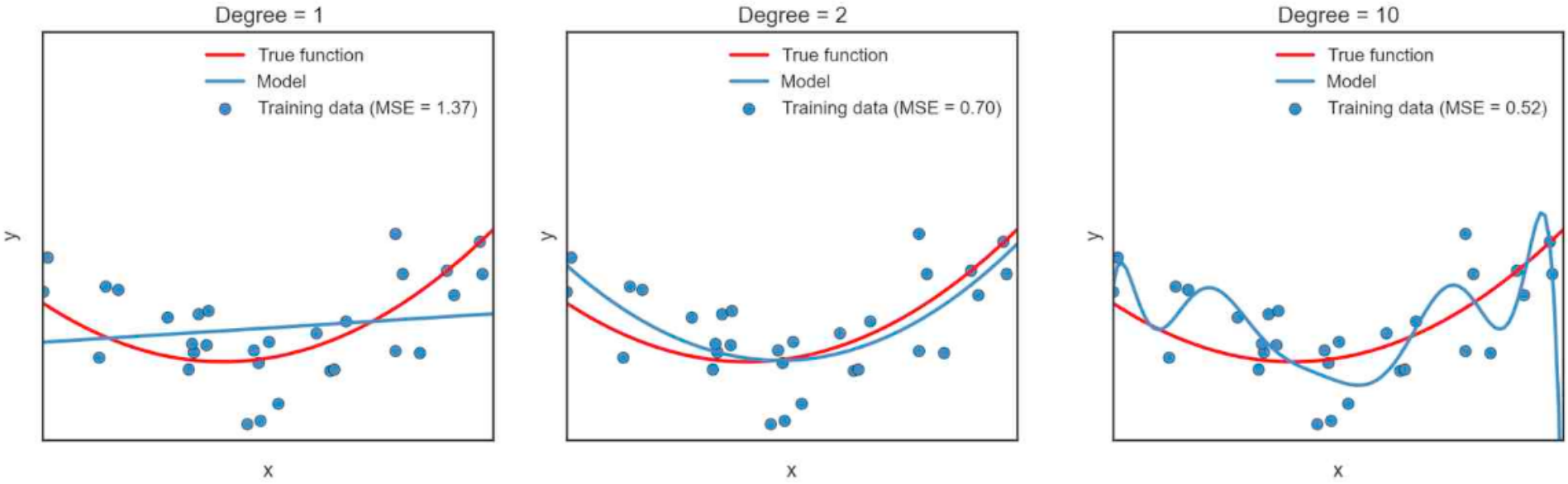
The error of the learnt model in predicting data not used for the training,

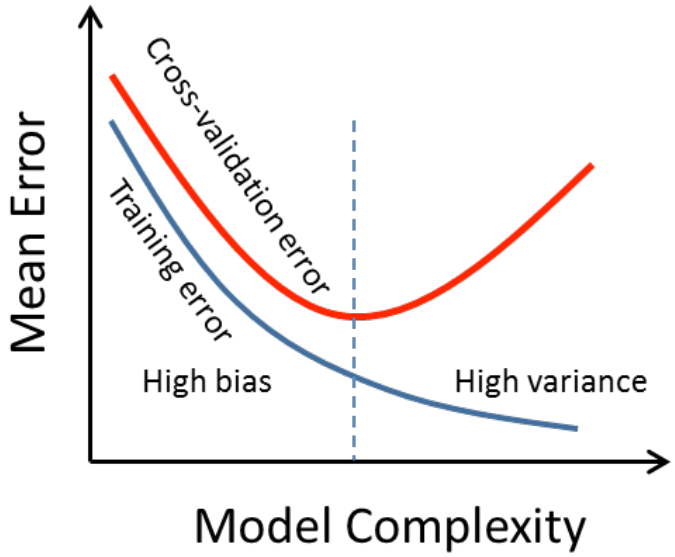
$$\epsilon^{\text{test}} = \left\| \mathbf{y}^{\text{test}} - \mathbf{X}^{\text{test}} \mathbf{w} \right\|_2$$

is called the **validation** or **test error/loss**. It provides a metric to validate how well the model generalized to new data, and this error can be used for hyperparameter optimization.

Cross-validation

Hyperparameter selection





Deep learning

Deep learning

WaveNet

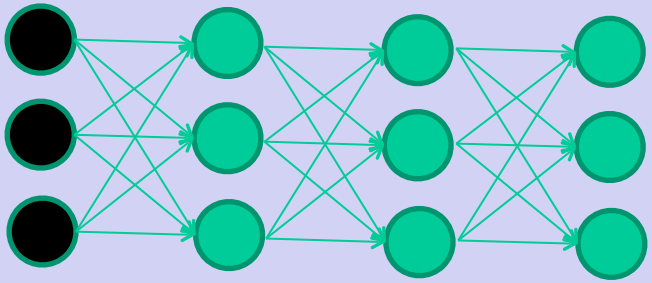
Deep learning

WaveNet

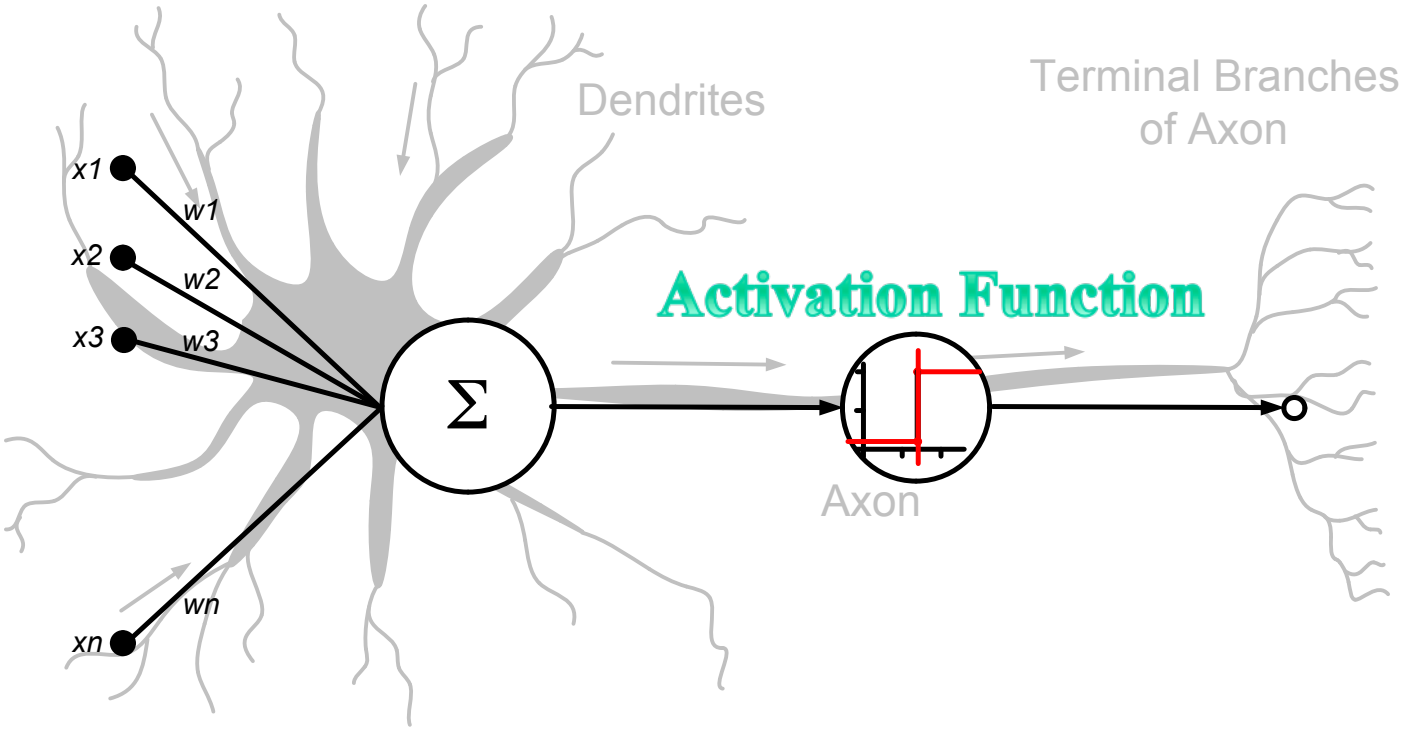
Deep learning

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 A person riding a motorcycle on a dirt road.	 Two dogs play in the grass.	 A skateboarder does a trick on a ramp.	 A dog is jumping to catch a frisbee.
 A group of young people playing a game of frisbee.	 Two hockey players are fighting over the puck.	 A little girl in a pink hat is blowing bubbles.	 A refrigerator filled with lots of food and drinks.
 A herd of elephants walking across a dry grass field.	 A close up of a cat laying on a couch.	 A red motorcycle parked on the side of the road.	 A yellow school bus parked in a parking lot.

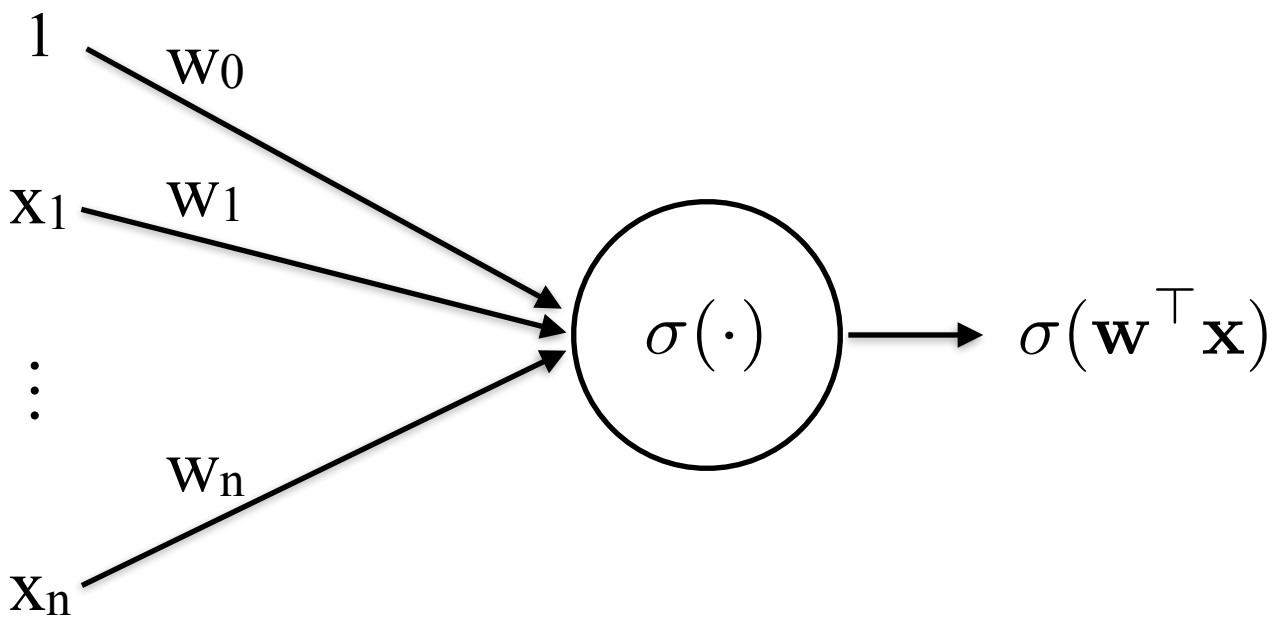
Supervised learning with feedforward neural networks



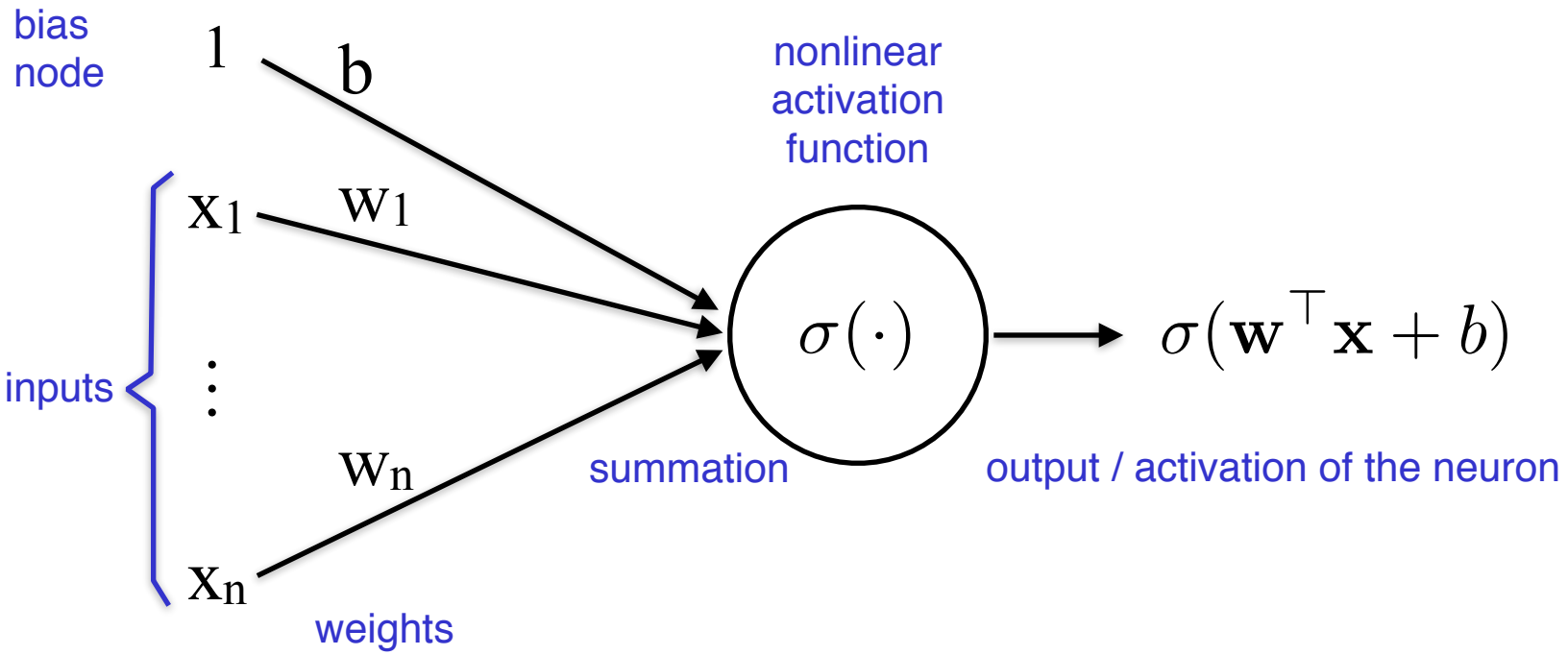
Neuron


Dendrites
Terminal Branches
of Axon
x1
x2
x3
xn
w1
w2
w3
wn
Σ
Activation Function
Axon

Artificial neuron

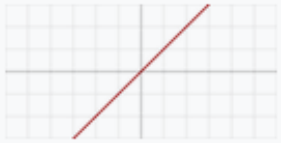
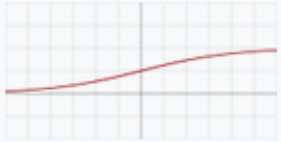
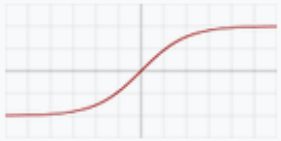
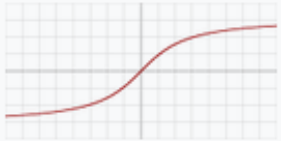
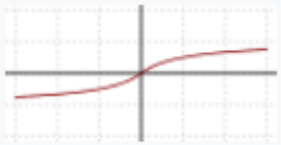


Artificial neuron



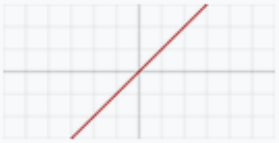
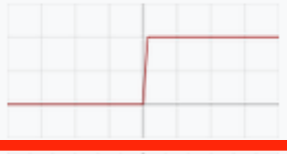
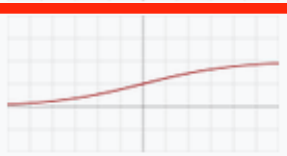
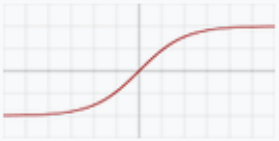
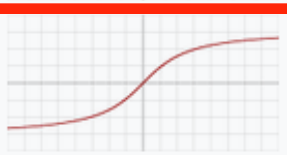
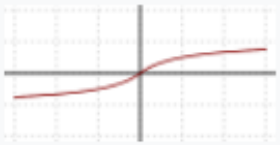
Activation functions

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]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

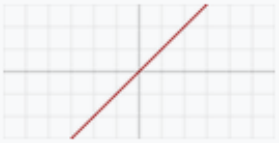
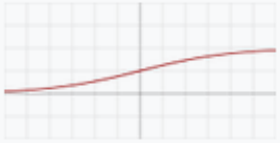
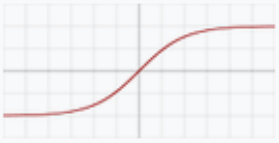
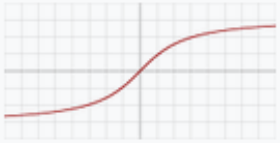
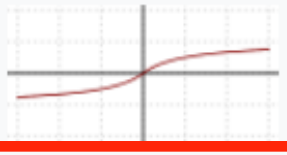
Activation functions

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

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$ traditionally used
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

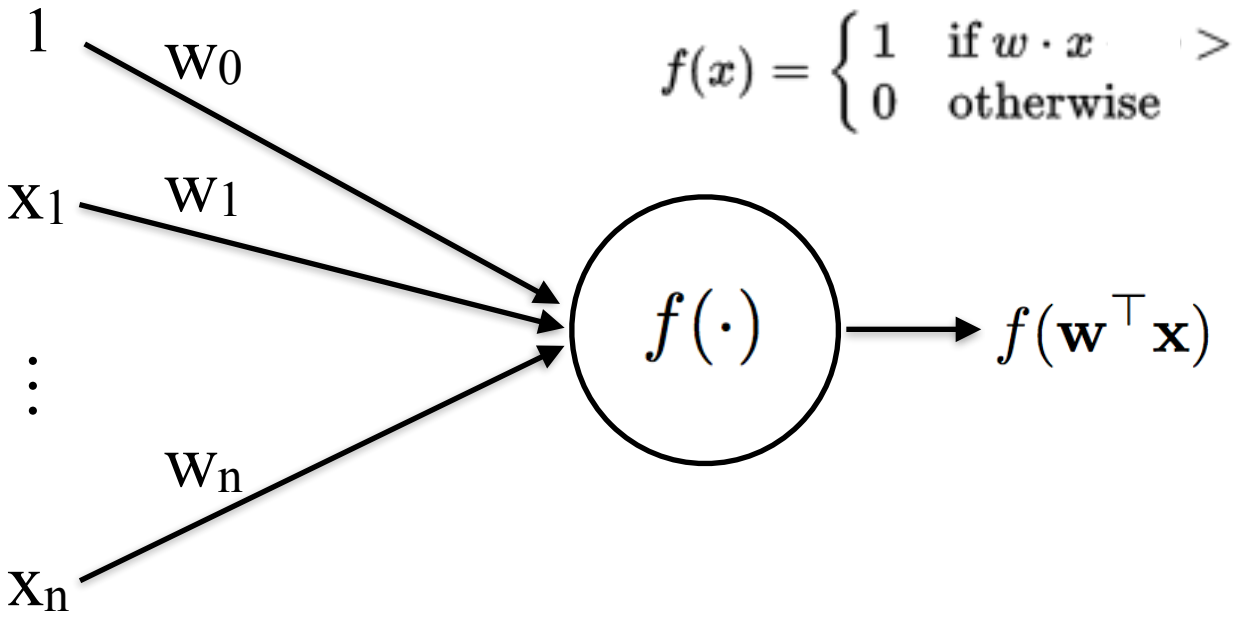
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1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

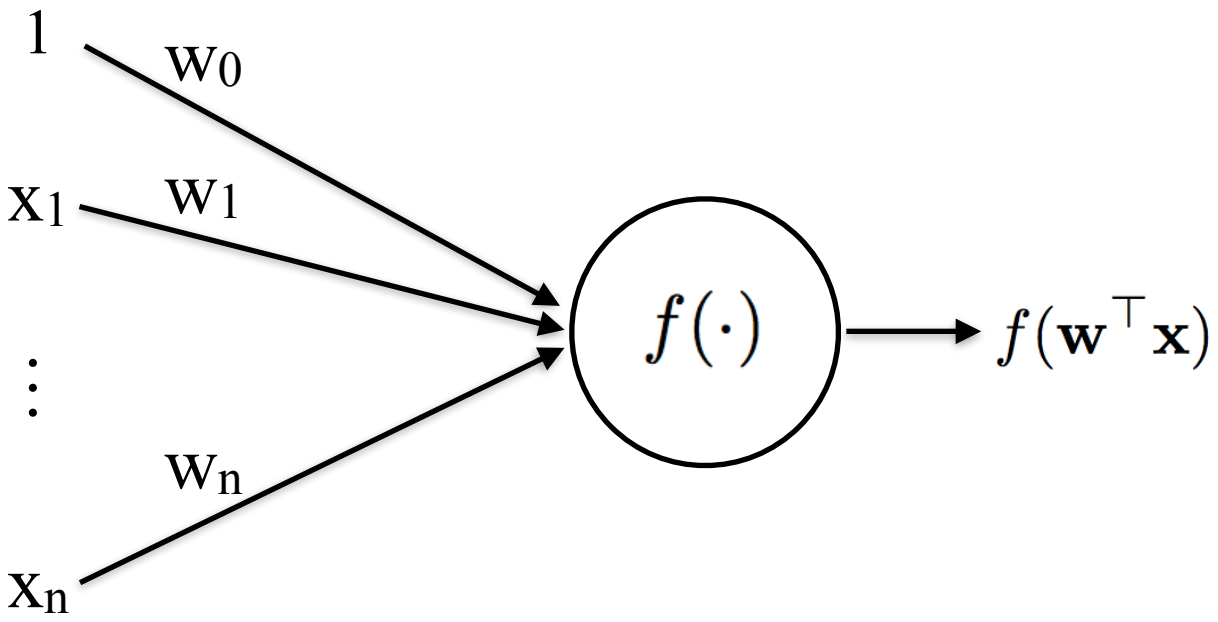
Perceptron (Rosenblatt 1957)

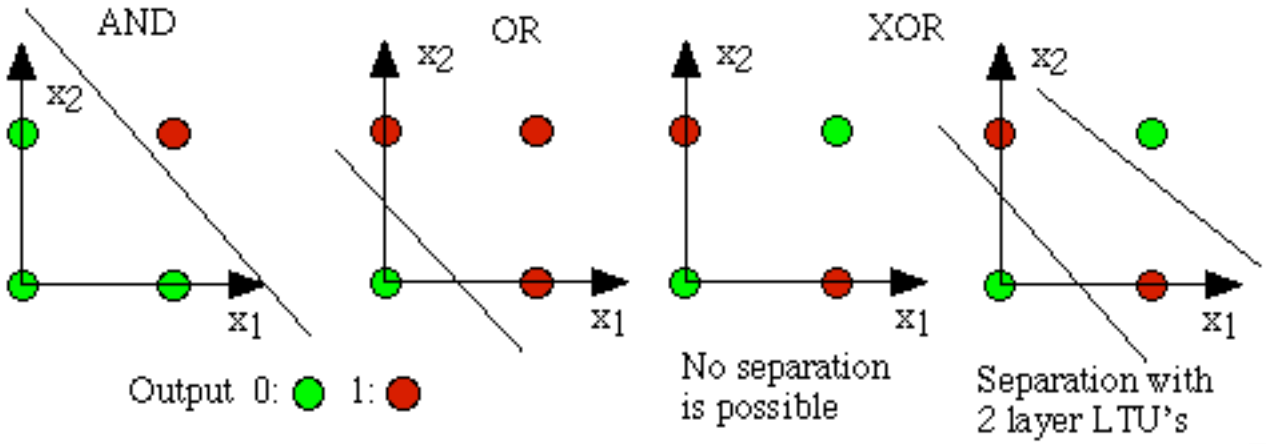
Used as a binary classifier - equivalent to support vector machine (SVM)



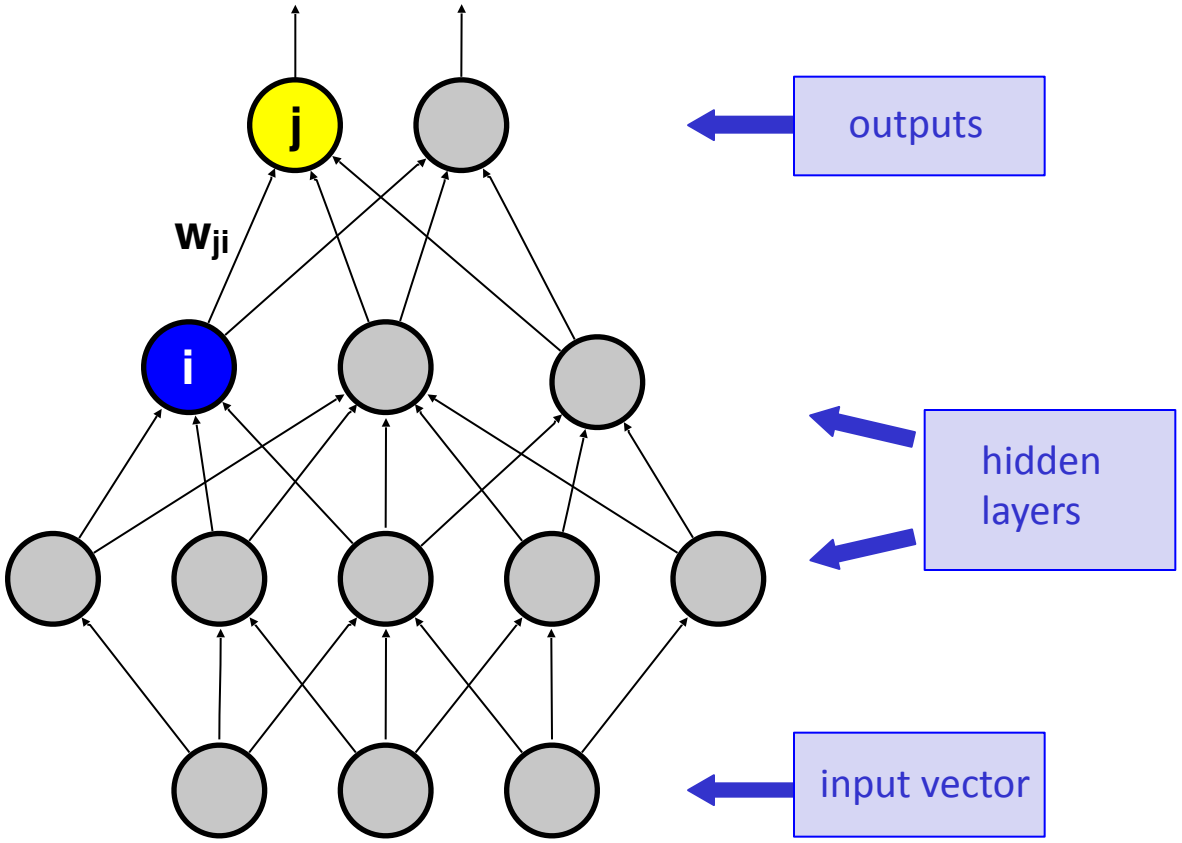
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron (Rosenblatt 1957)





Multilayer perceptrons (~1985)



Universal Representation Theorem

Hornik et al., 1989; Cybenko, 1989)

- Networks with a **single** hidden layer can represent **any** function F(x) with arbitrary accuracy in the large hidden layer size limit

- However:
  - A given learning algorithm may be unable to (efficiently) find an optimum with this accuracy.

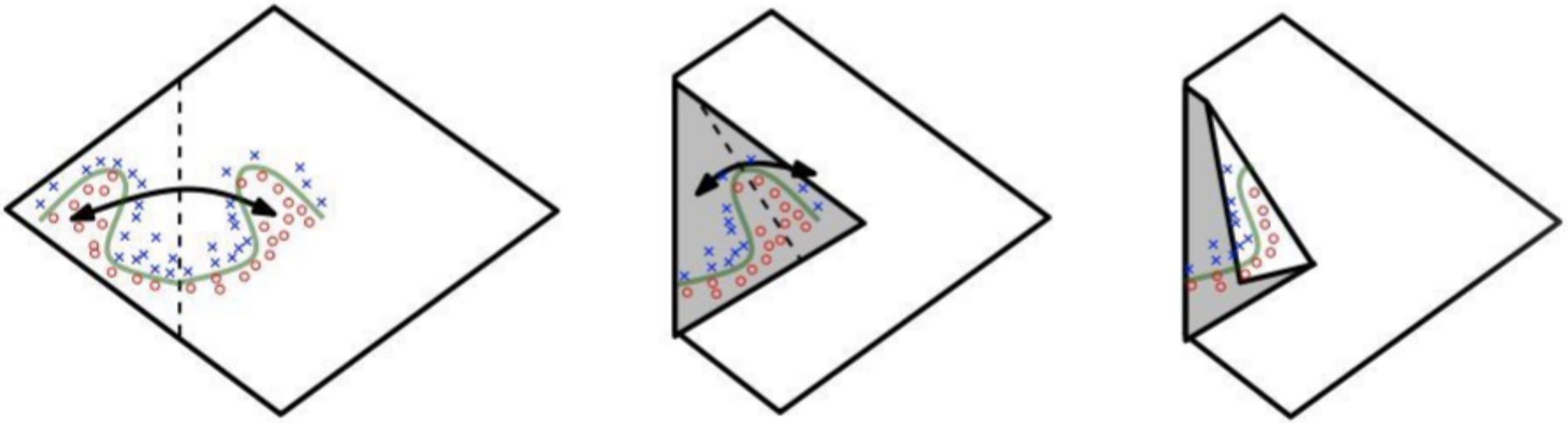
  - Networks with single hidden layers can be inefficient in representing nonlinear functions. Some examples require a number of hidden neurons that is exponential in the input size

  - For many datasets, deep networks can represent the function F(x) even with narrow layers.

Representation efficiency of deep networks

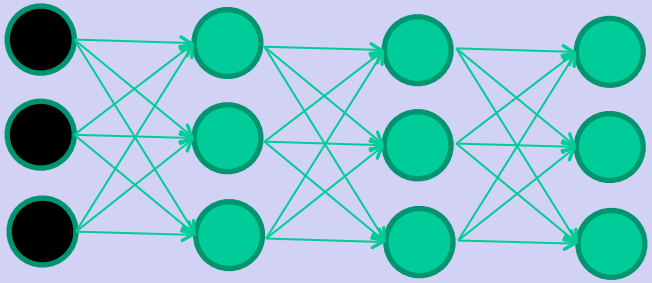
Some families of functions can be approximated efficiently by a network with depth > d, but require an number of hidden units that is exponential in input size when using a shallow network.

Håstad (1986), Håstad and Goldmann (1991), Hajnal et al., (1993), Maass (1992), Maass et al., (1994), Montufar et al. 2014



Montufar et al. 2014

Parameter optimization (weight training)



Multilayer perceptrons (~1985)

Compare outputs with correct answer to get error signal

Error at output for a given example:

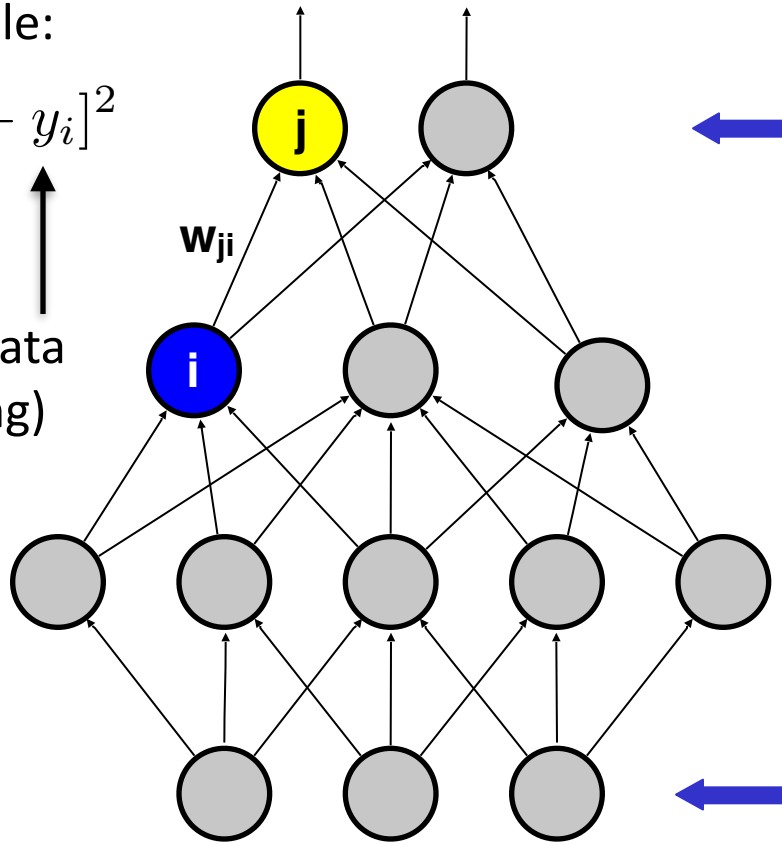
$$\text{Loss}(\mathbf{X};\theta) = \frac{1}{2}\sum_i [\hat{y}_i(\mathbf{x}_i;\theta) - y_i]^2$$

Network prediction

Labels in training data (supervised learning)

Learn network parameters

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$$



outputs

hidden layers

input vector

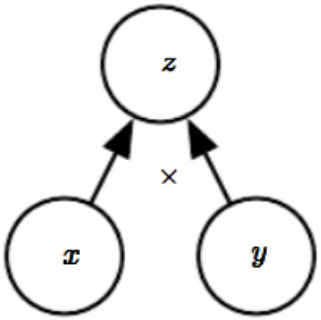
Idea:

- Compute derivatives of Loss wrt neuron activation at output layer
- **Backpropagate** derivatives through network using the chain rule of differentiation
- Update weights

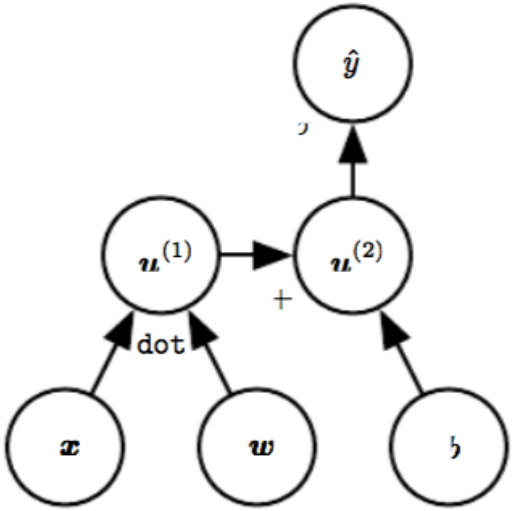
$$\Delta w_{ij}^l = -\epsilon \frac{\partial \text{Loss}(\mathbf{X};\theta)}{\partial w_{ij}^l}$$

Learning rate

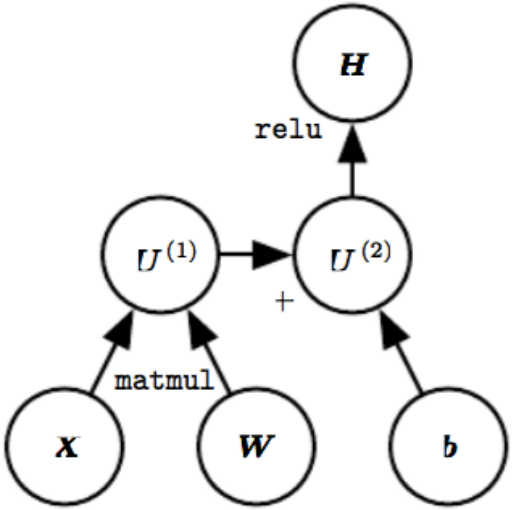
Computing graphs

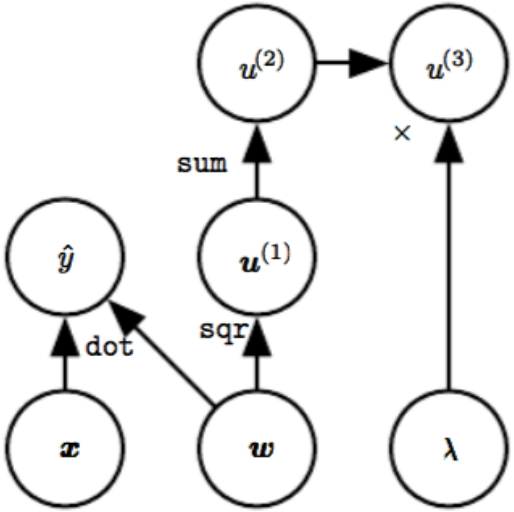


(a)



(b)





Forward and backward propagation

Forward propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

Backprop

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for

Stochastic gradient descent

Loss function of type $$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Gradients $$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}).$$

Sample **Minibatch** $$\mathbb{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$$

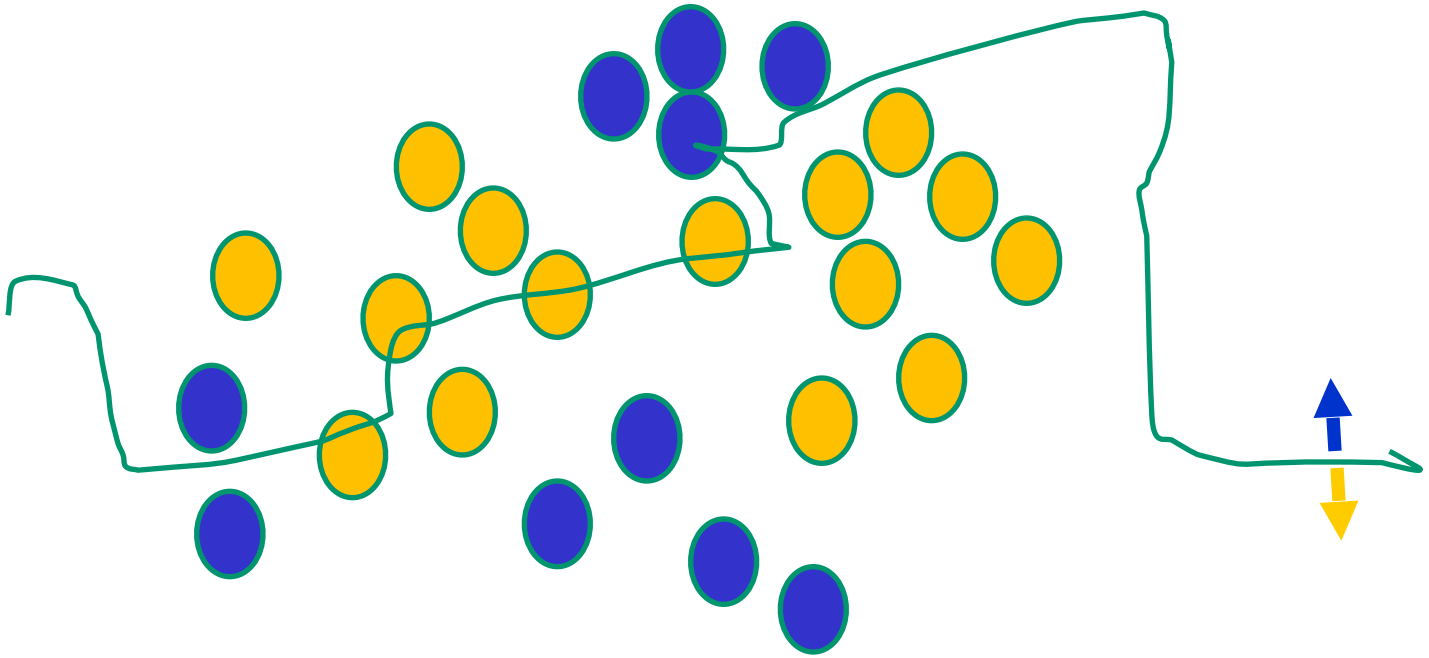
Estimate gradient from Minibatch: $$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}).$$

Update parameters $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

Learning rate

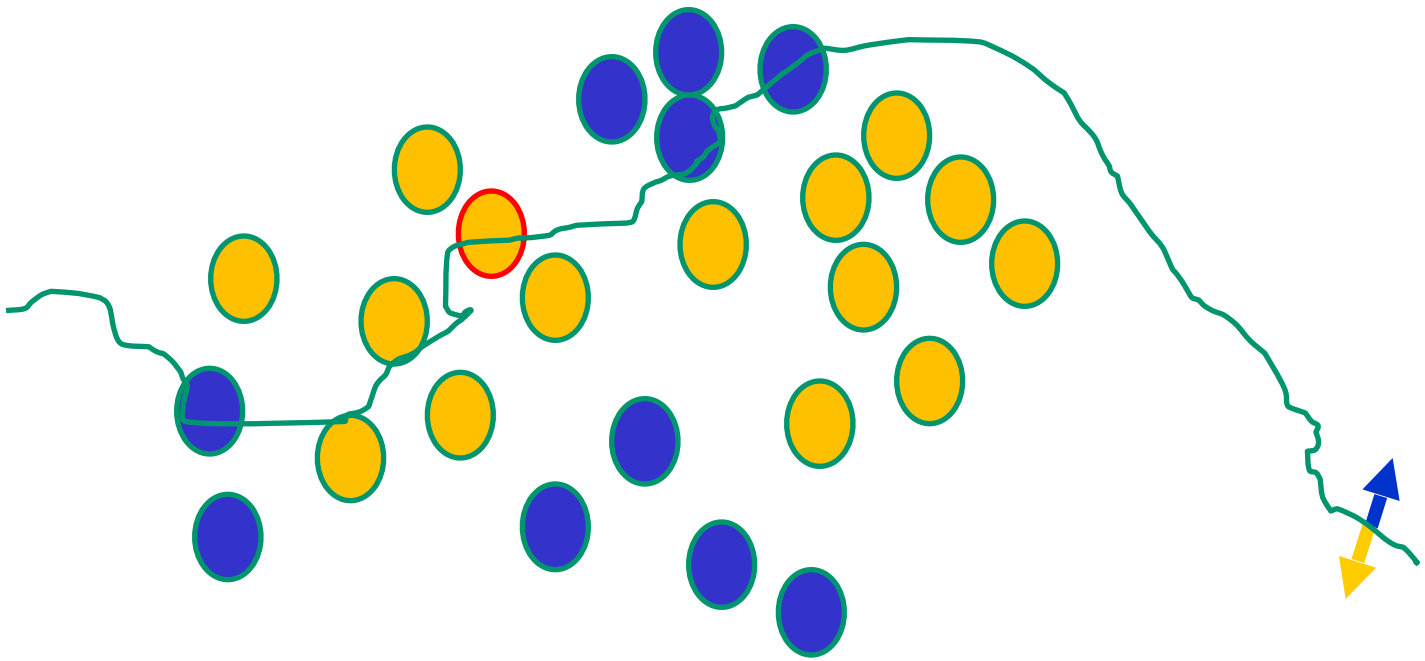
Decision boundary perspective on learning

Initial random weights



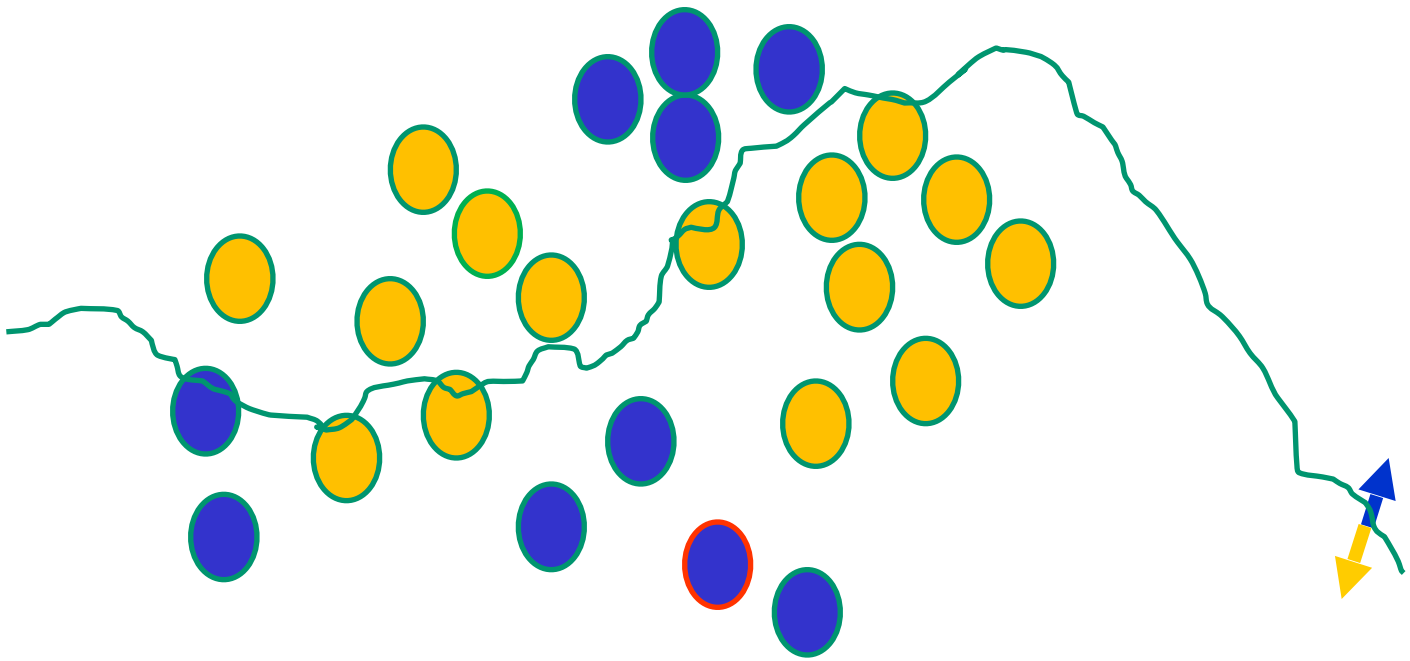
Decision boundary perspective on learning

Present a training batch / adjust the weights



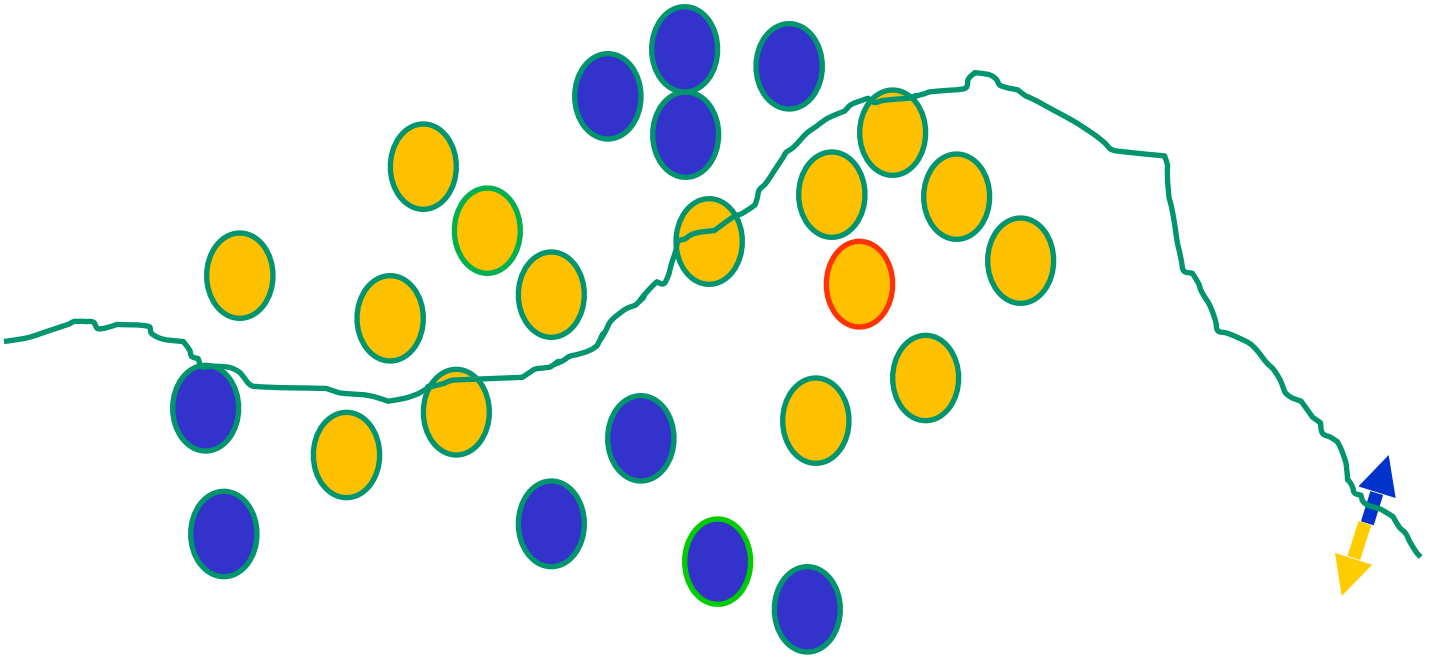
Decision boundary perspective on learning

Present a training batch / adjust the weights



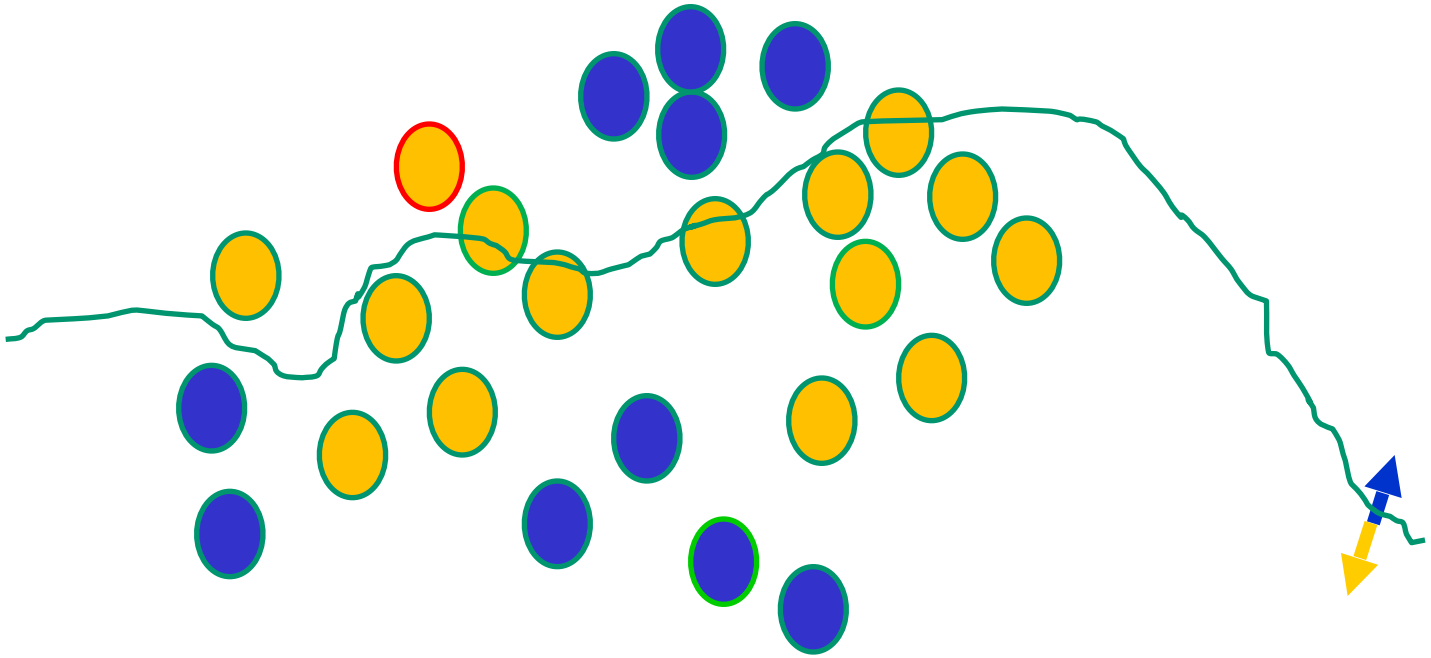
Decision boundary perspective on learning

Present a training batch / adjust the weights



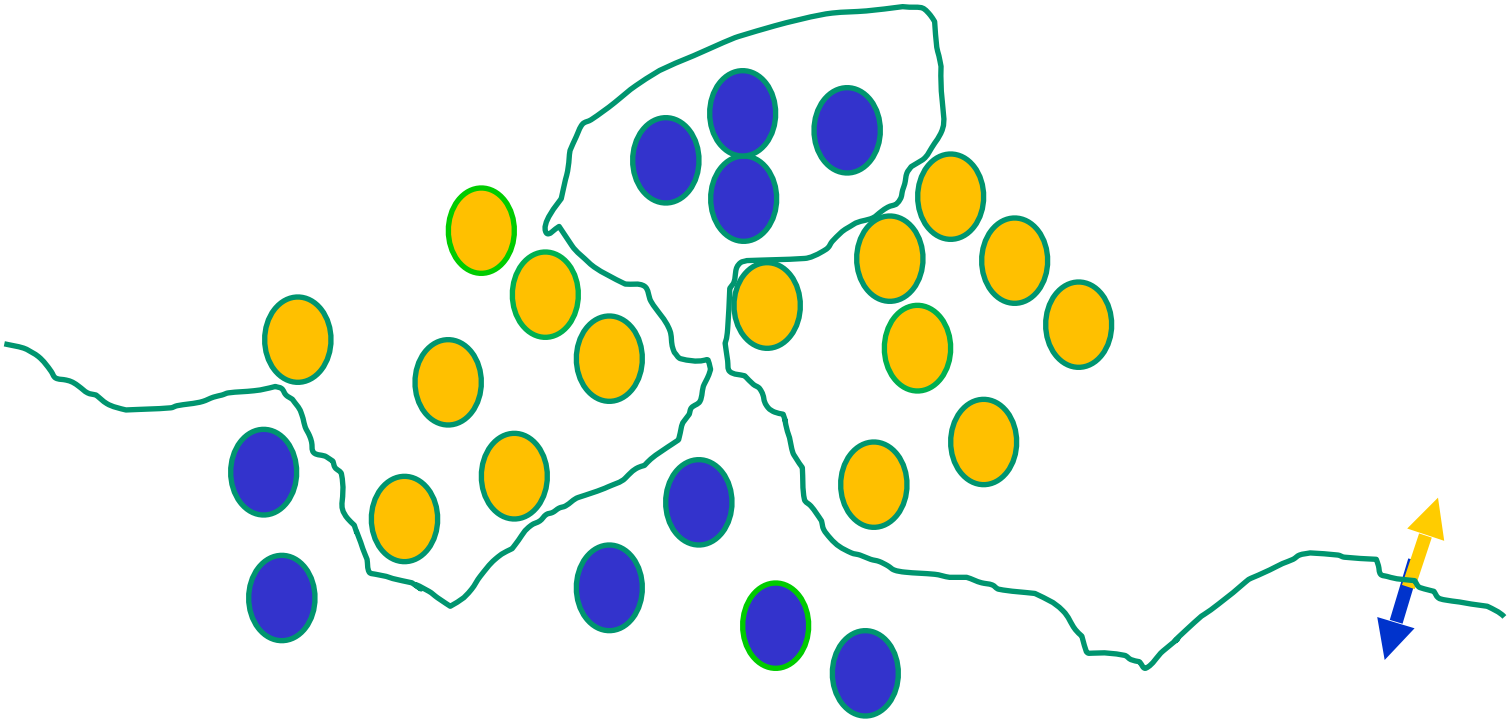
Decision boundary perspective on learning

Present a training batch / adjust the weights



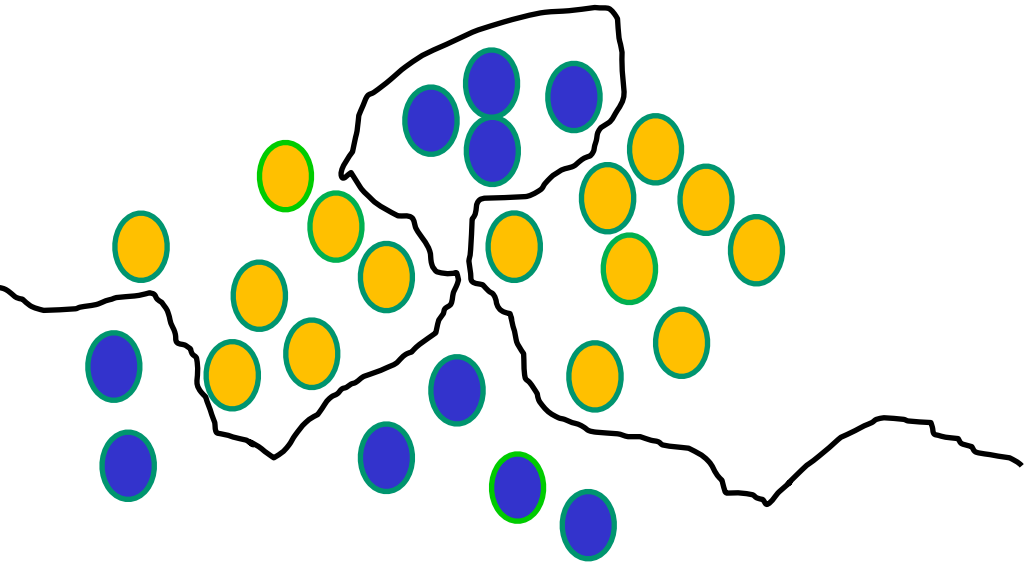
Decision boundary perspective on learning

Eventually…

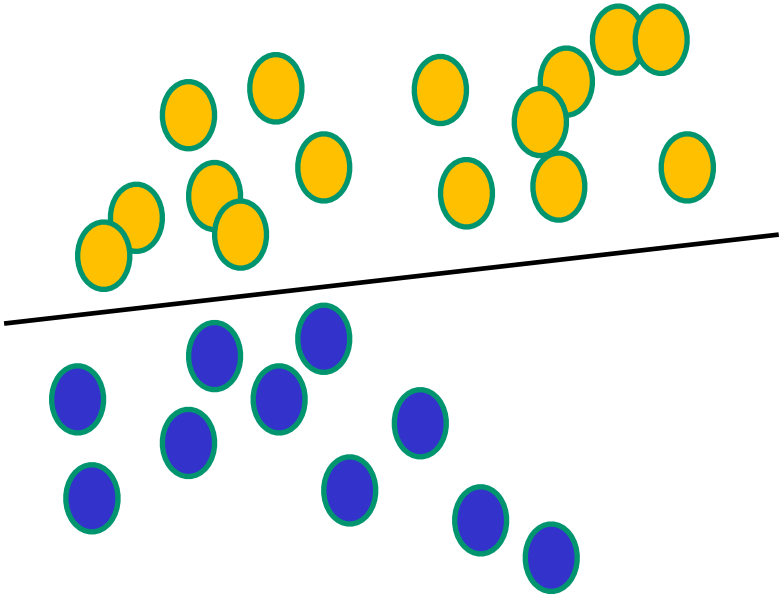


Decision boundary perspective on learning

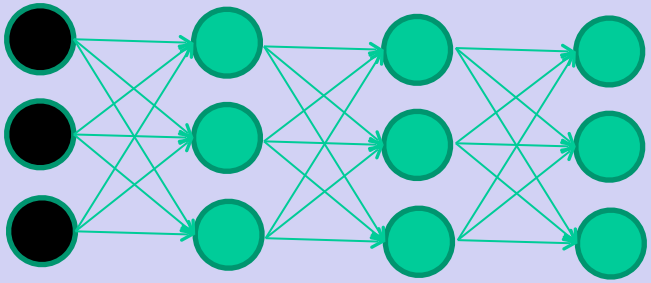
Deep Neural network



Kernel method



What does a neural network learn?



Feature detectors

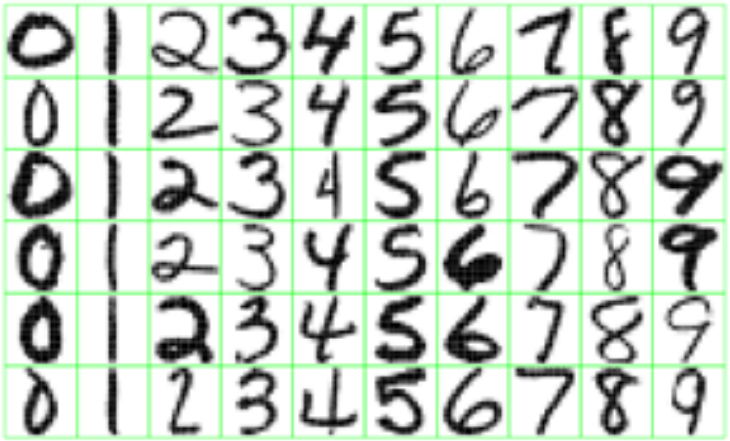
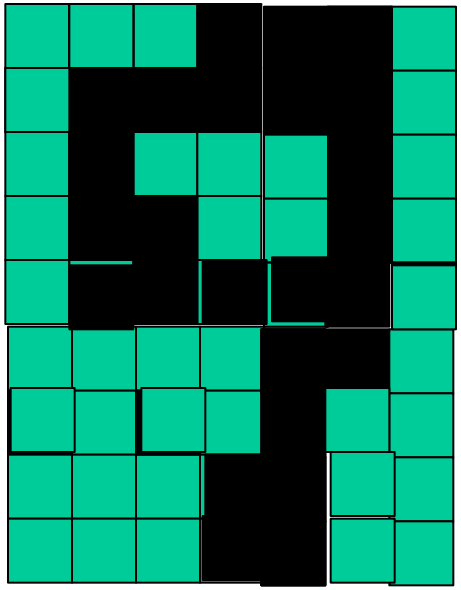
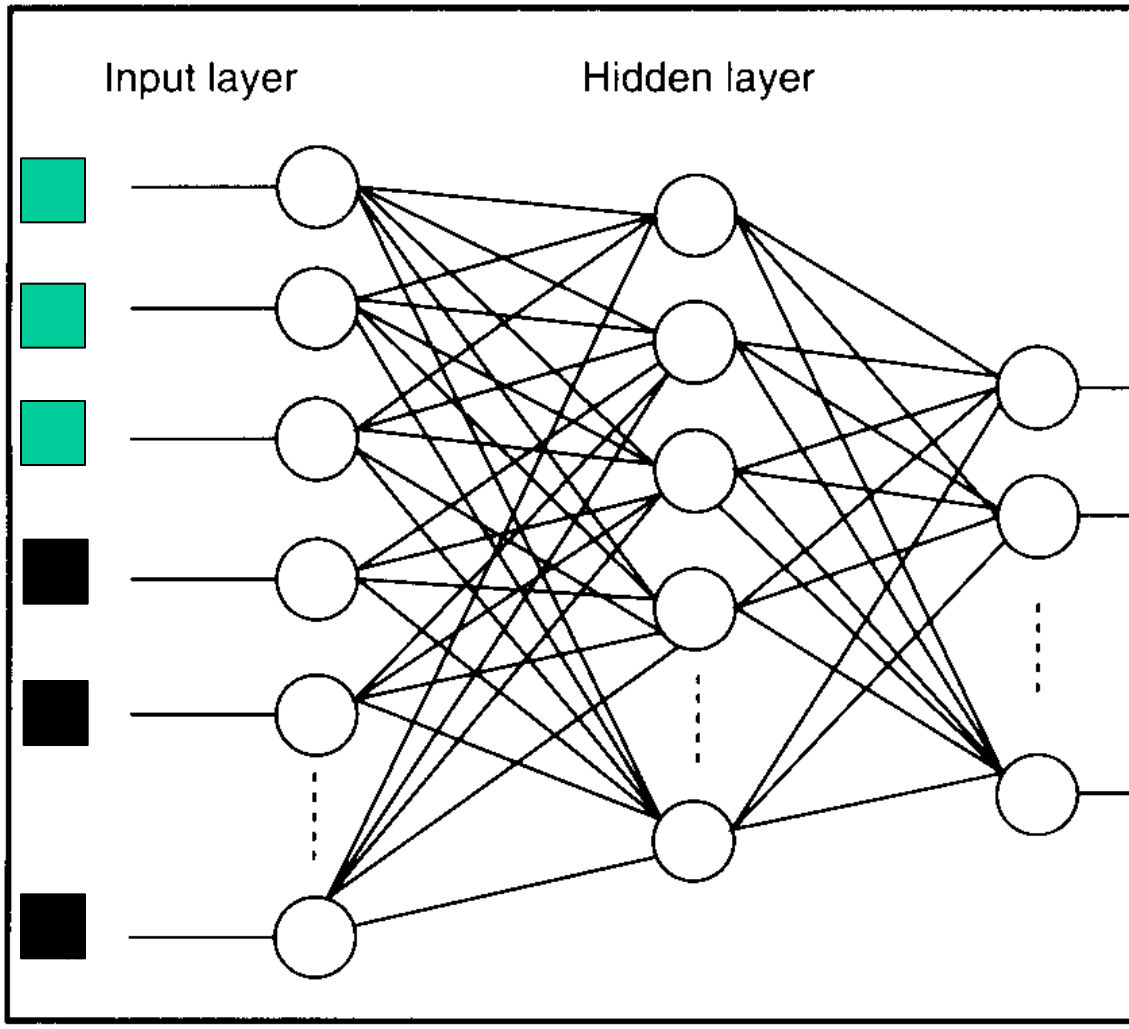


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





What is this unit doing?

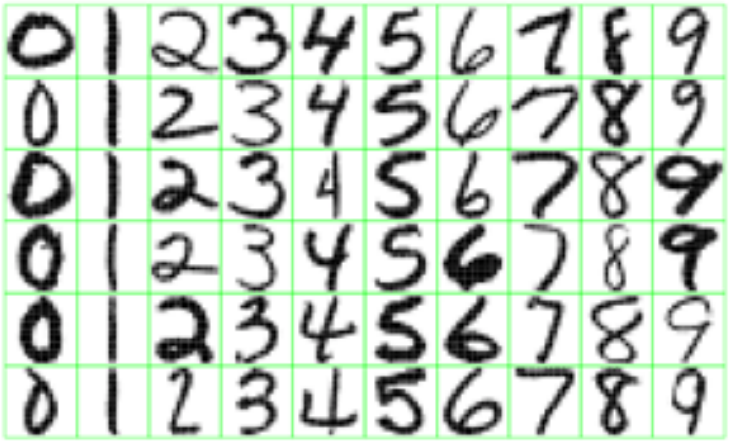
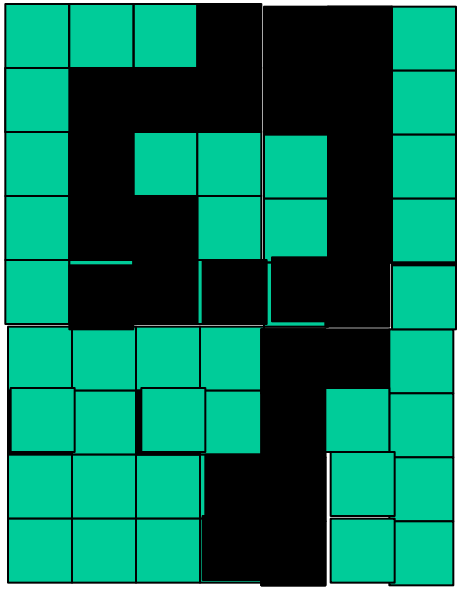
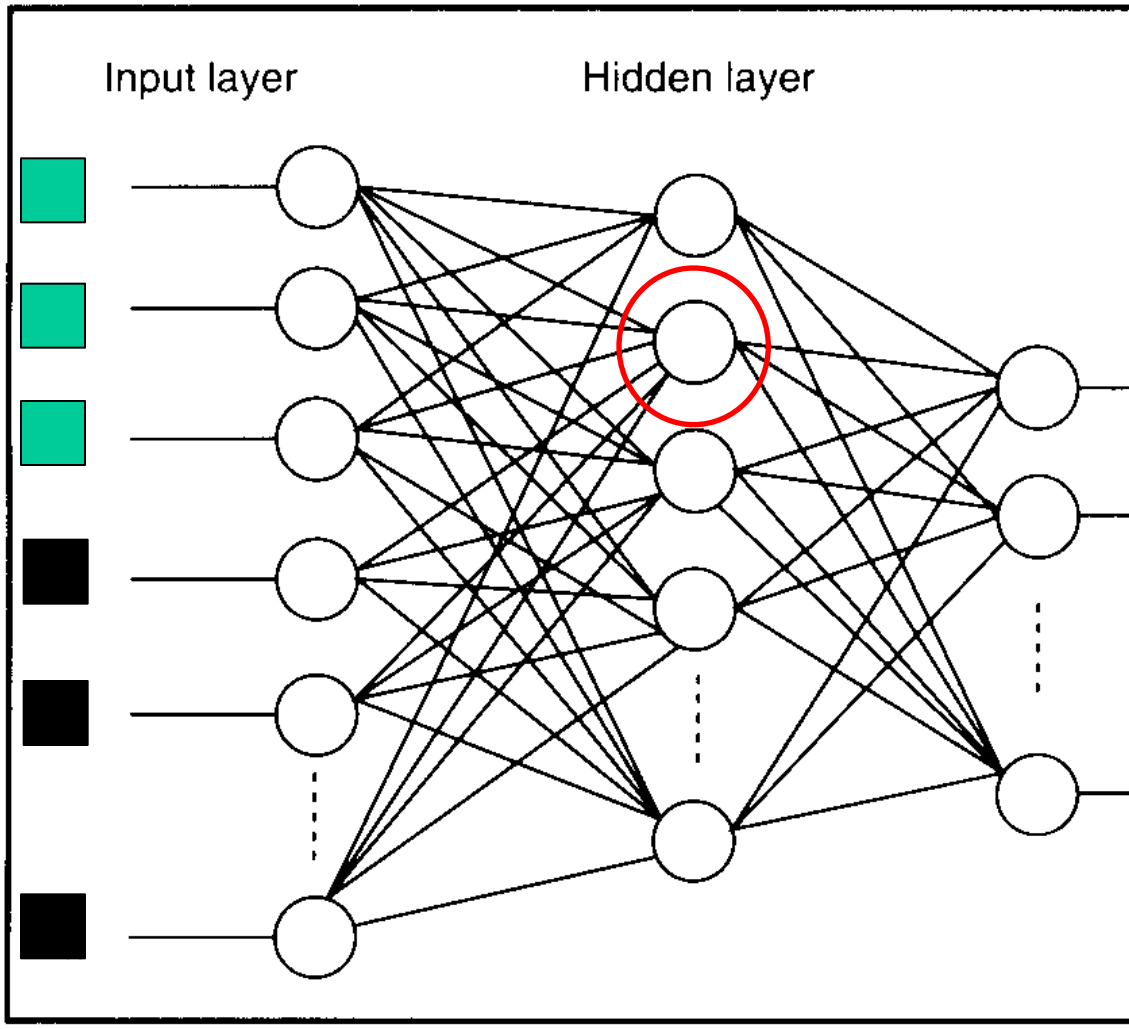
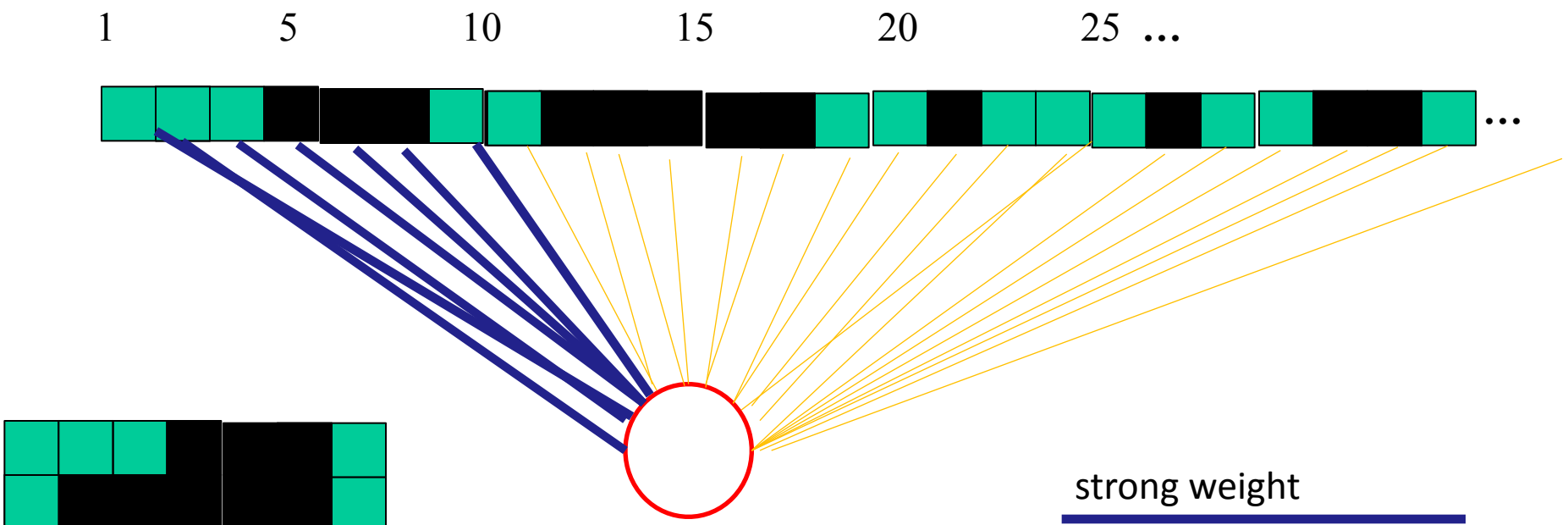


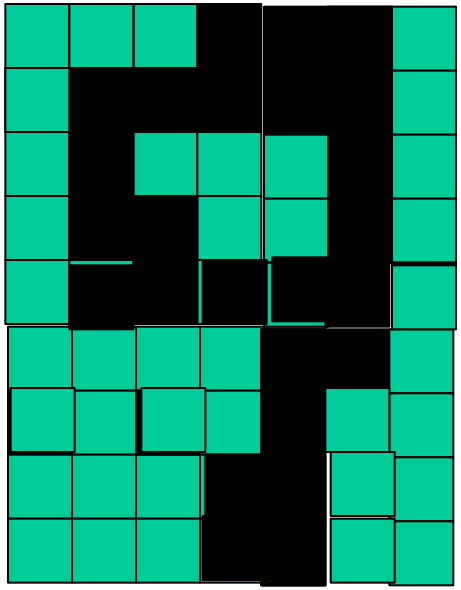
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





Hidden layers become self-organized feature detectors

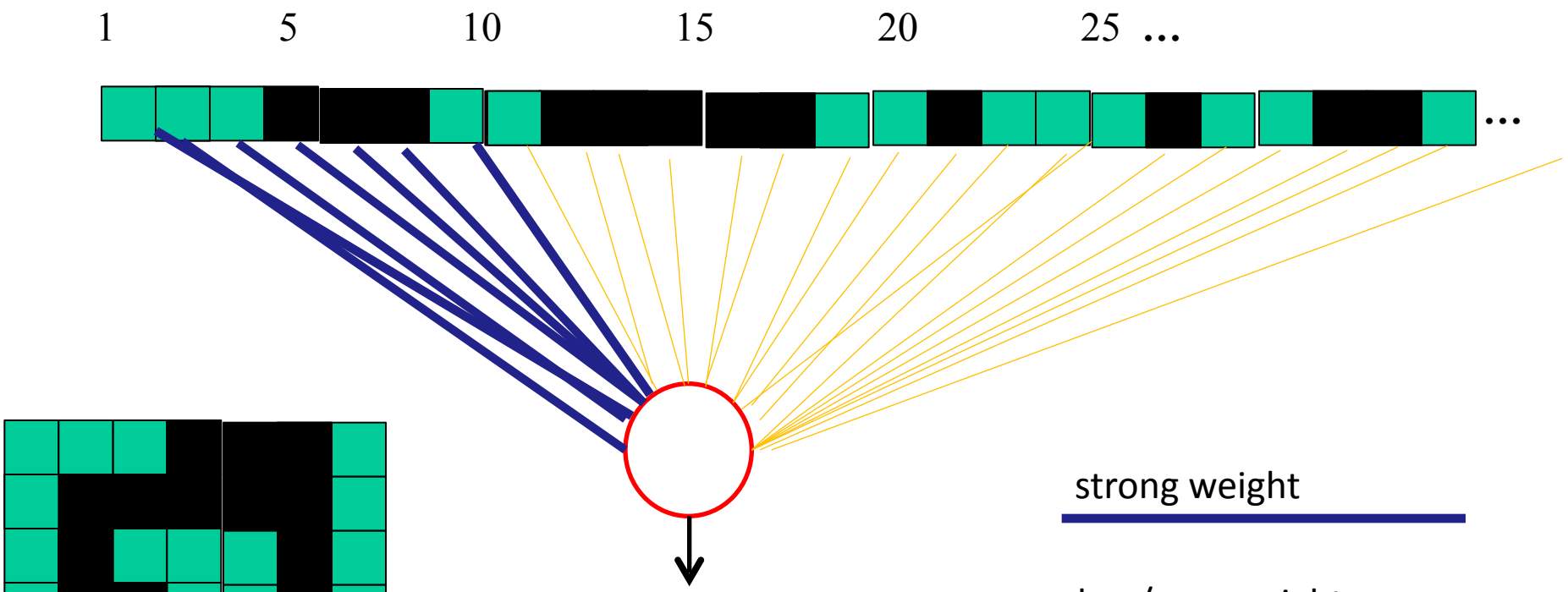


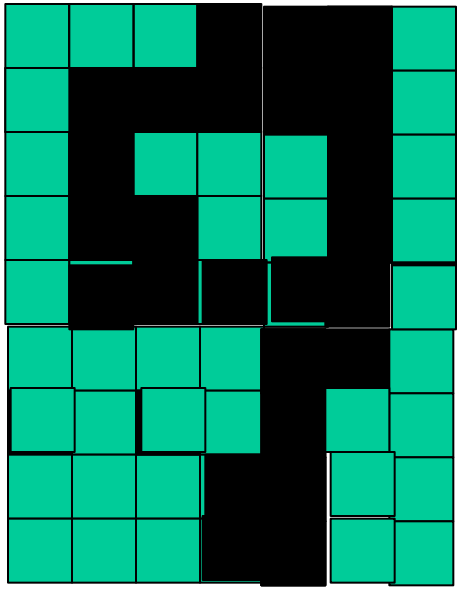


strong weight

low/zero weight

What does this unit detect?

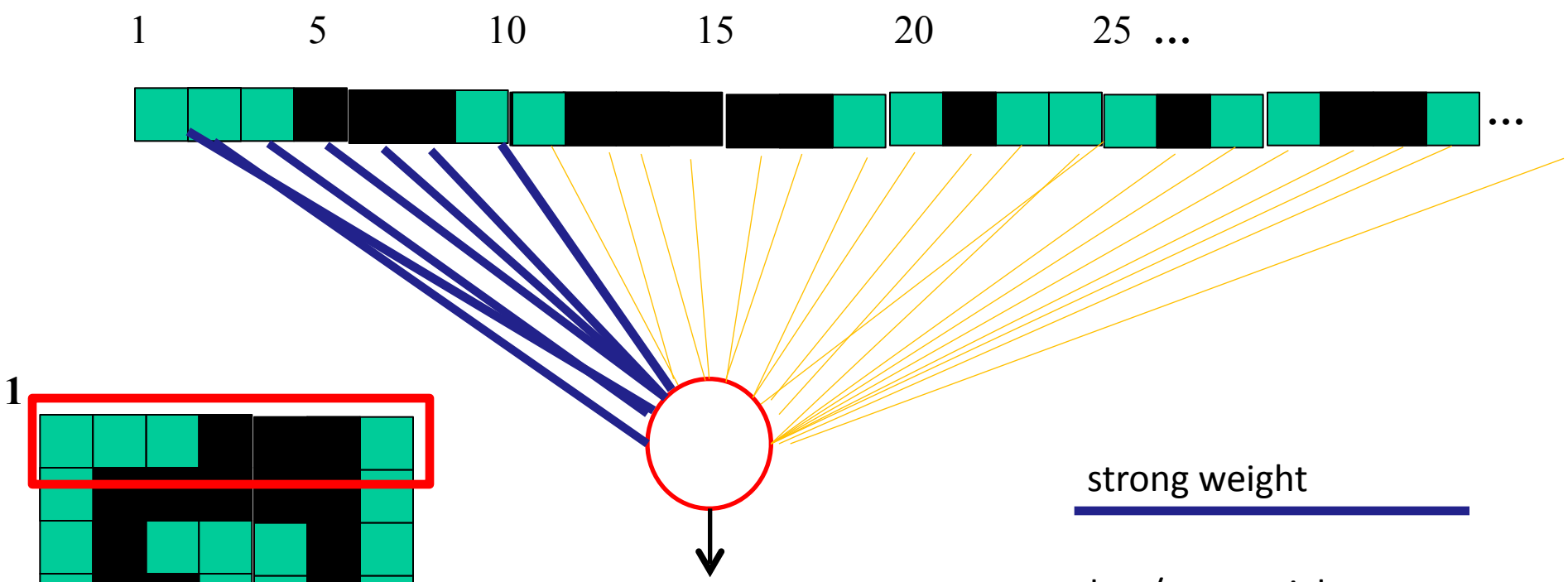


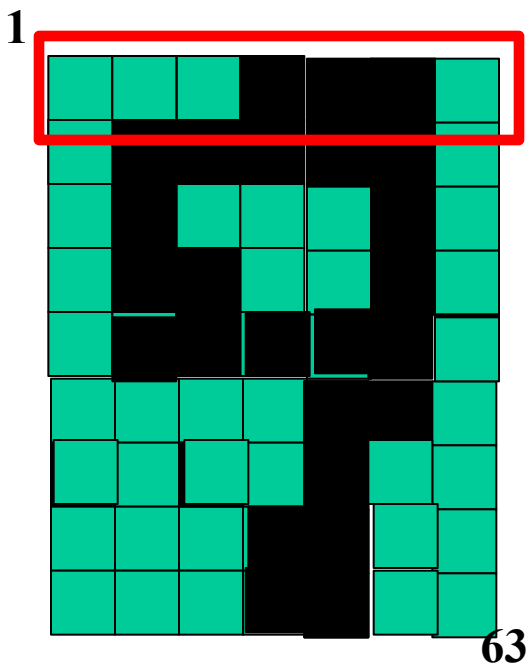


strong weight

low/zero weight

What does this unit detect?



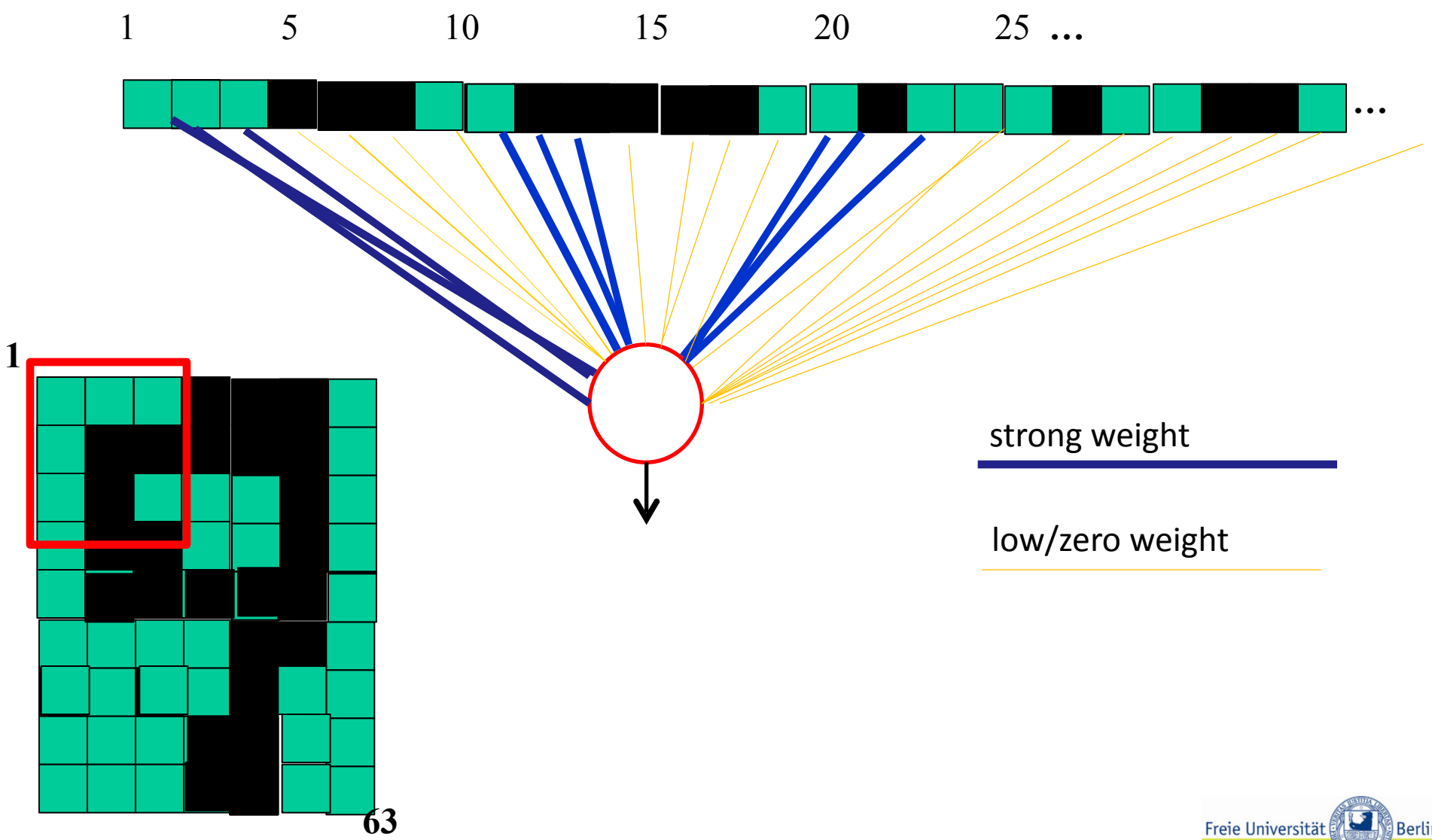


strong weight

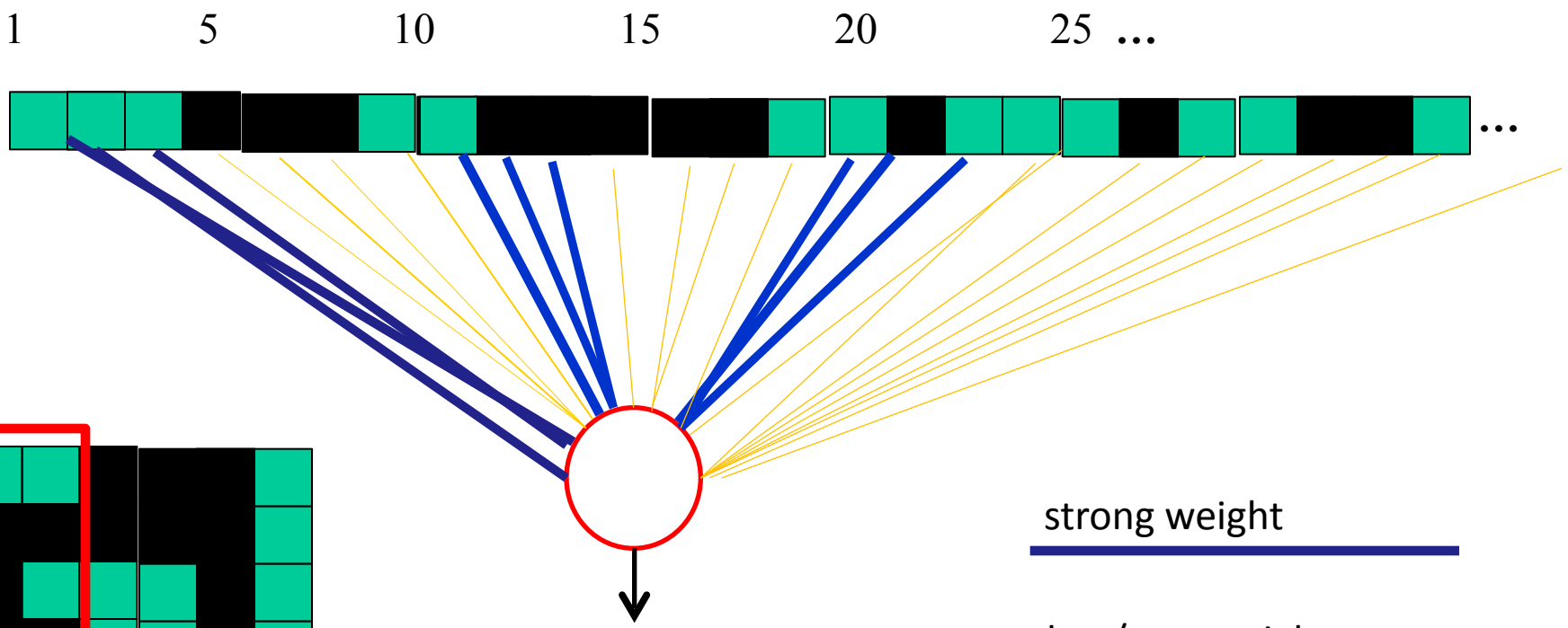
low/zero weight

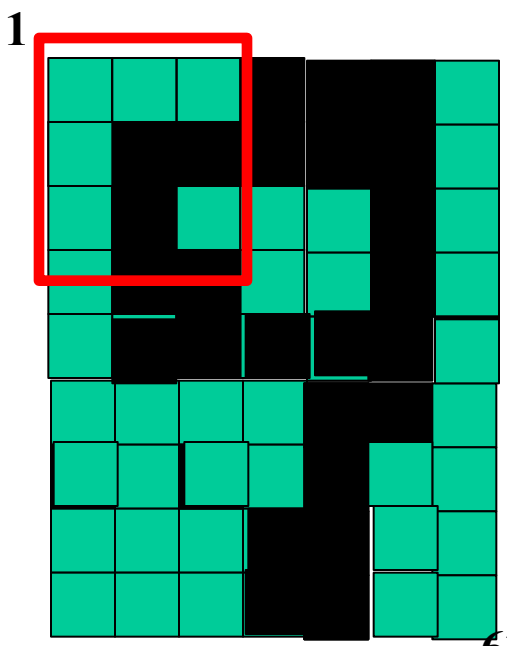
it will send strong signal for a horizontal line in the top row, ignoring everywhere else

What does this unit detect?



What does this unit detect?





strong weight

low/zero weight

Strong signal for a dark area in the top left corner

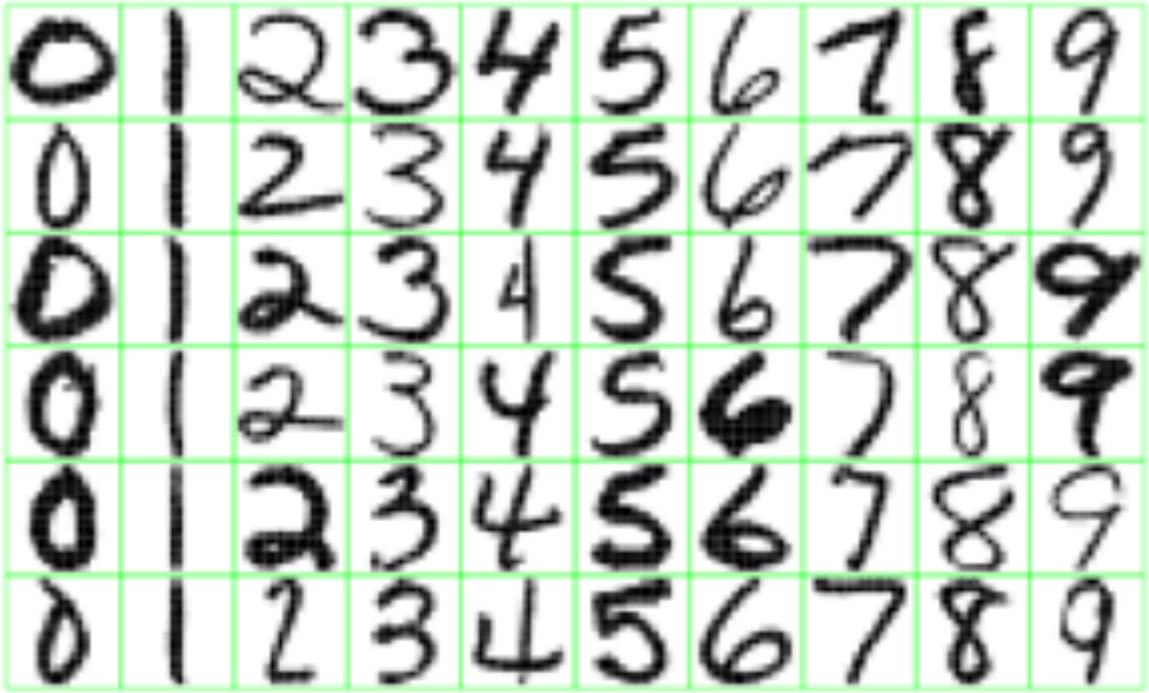


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

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

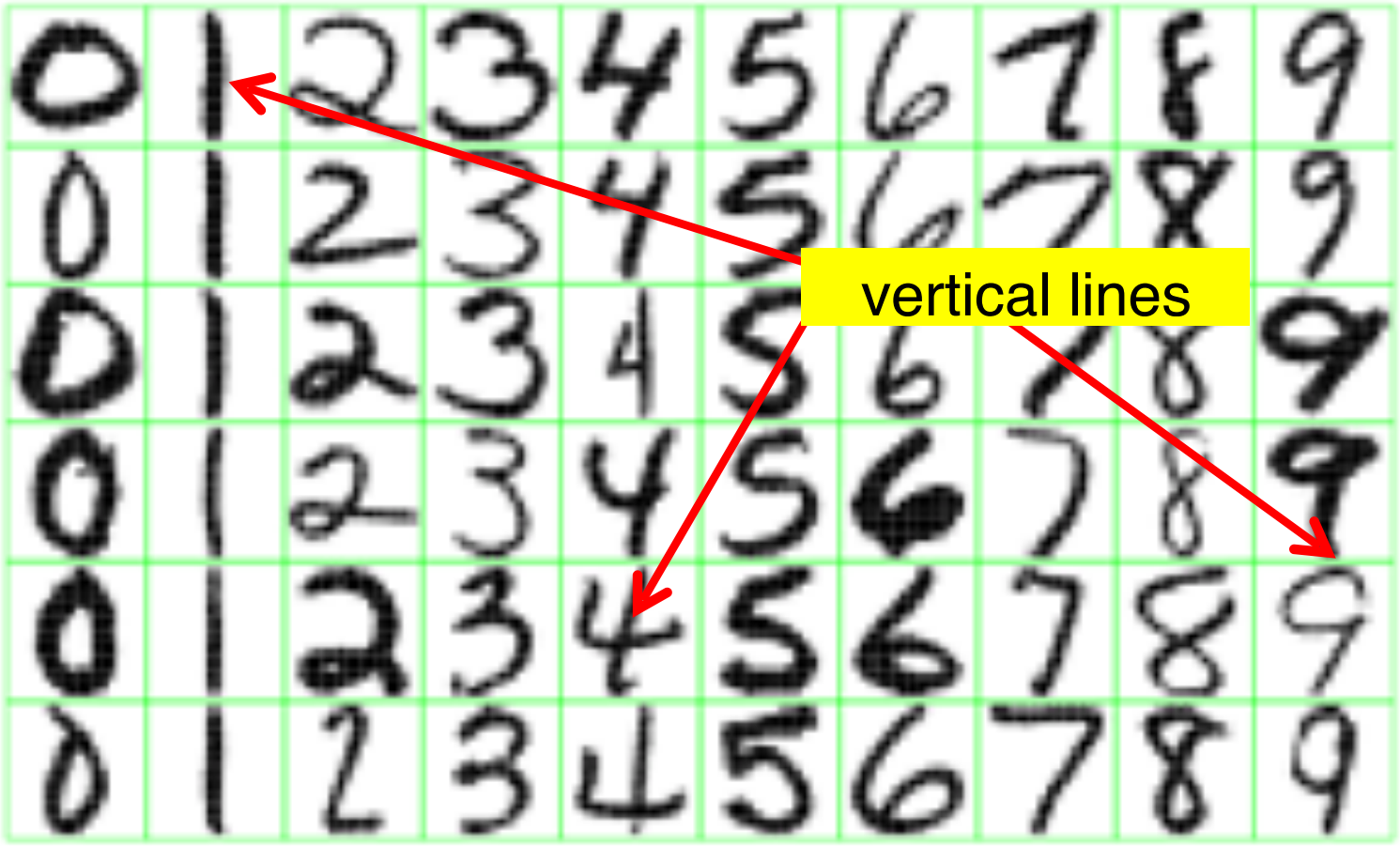


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

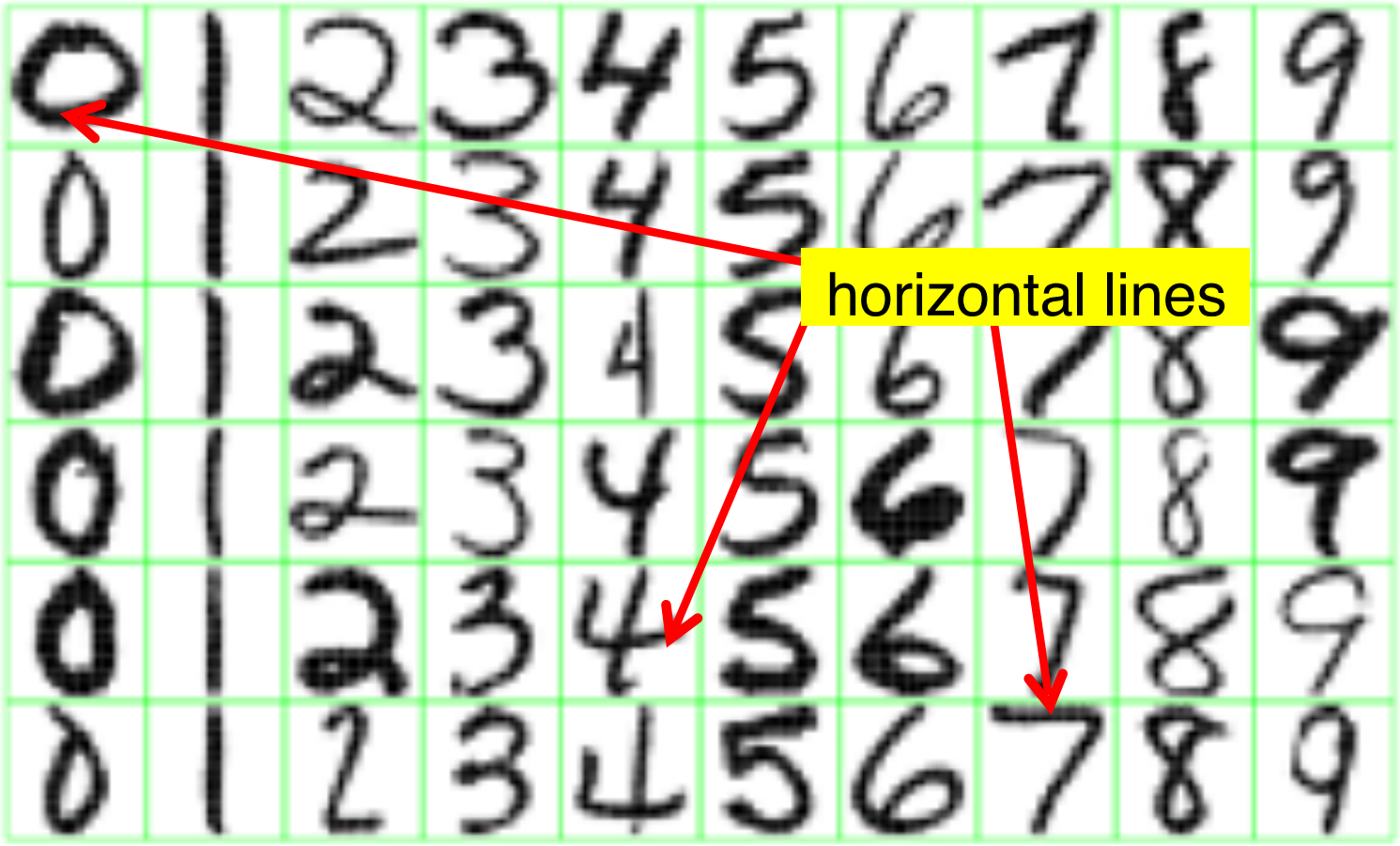
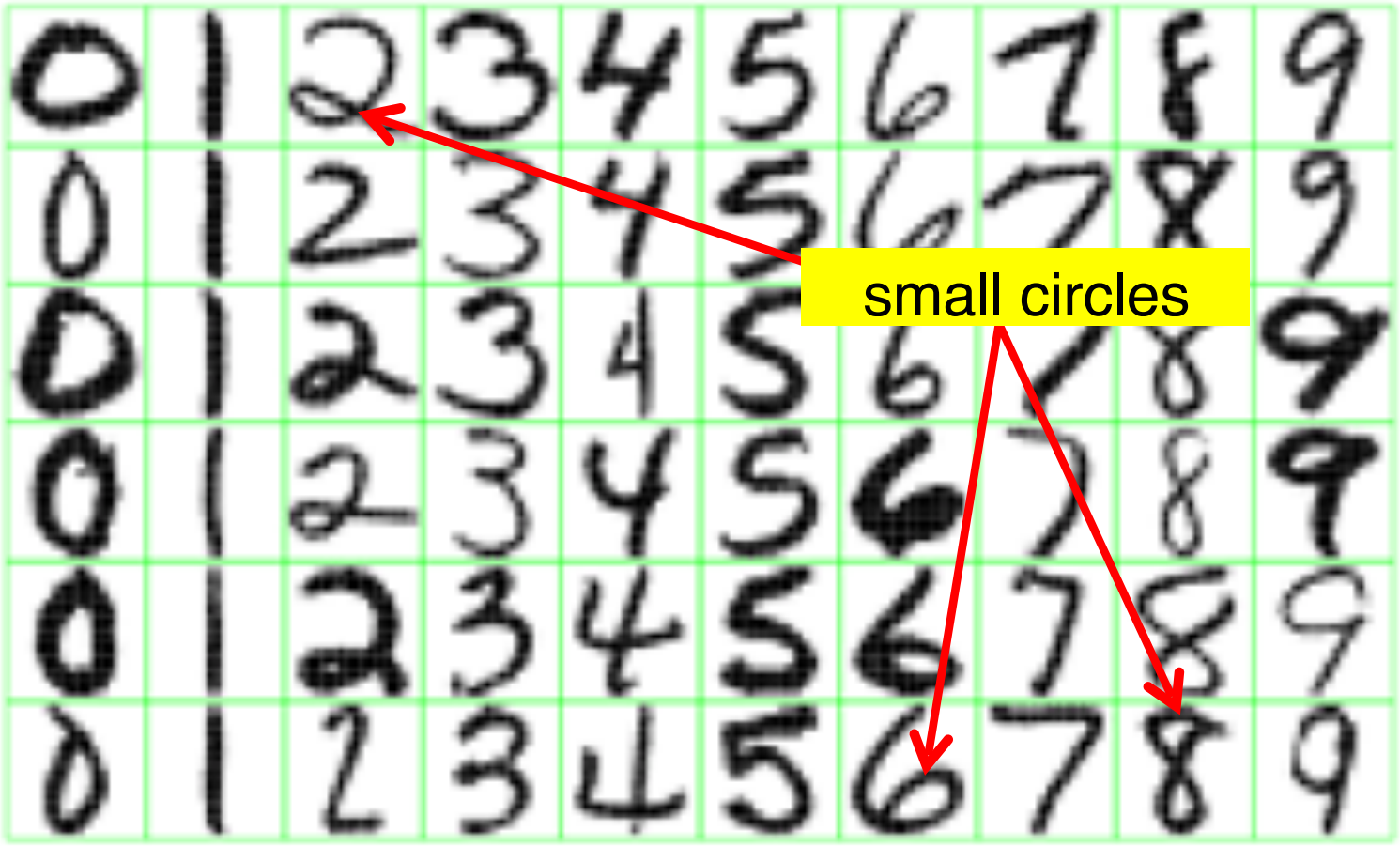
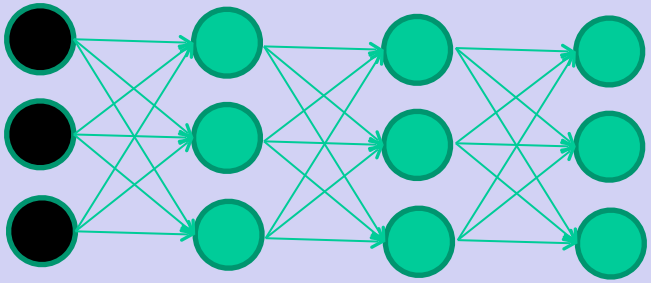


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

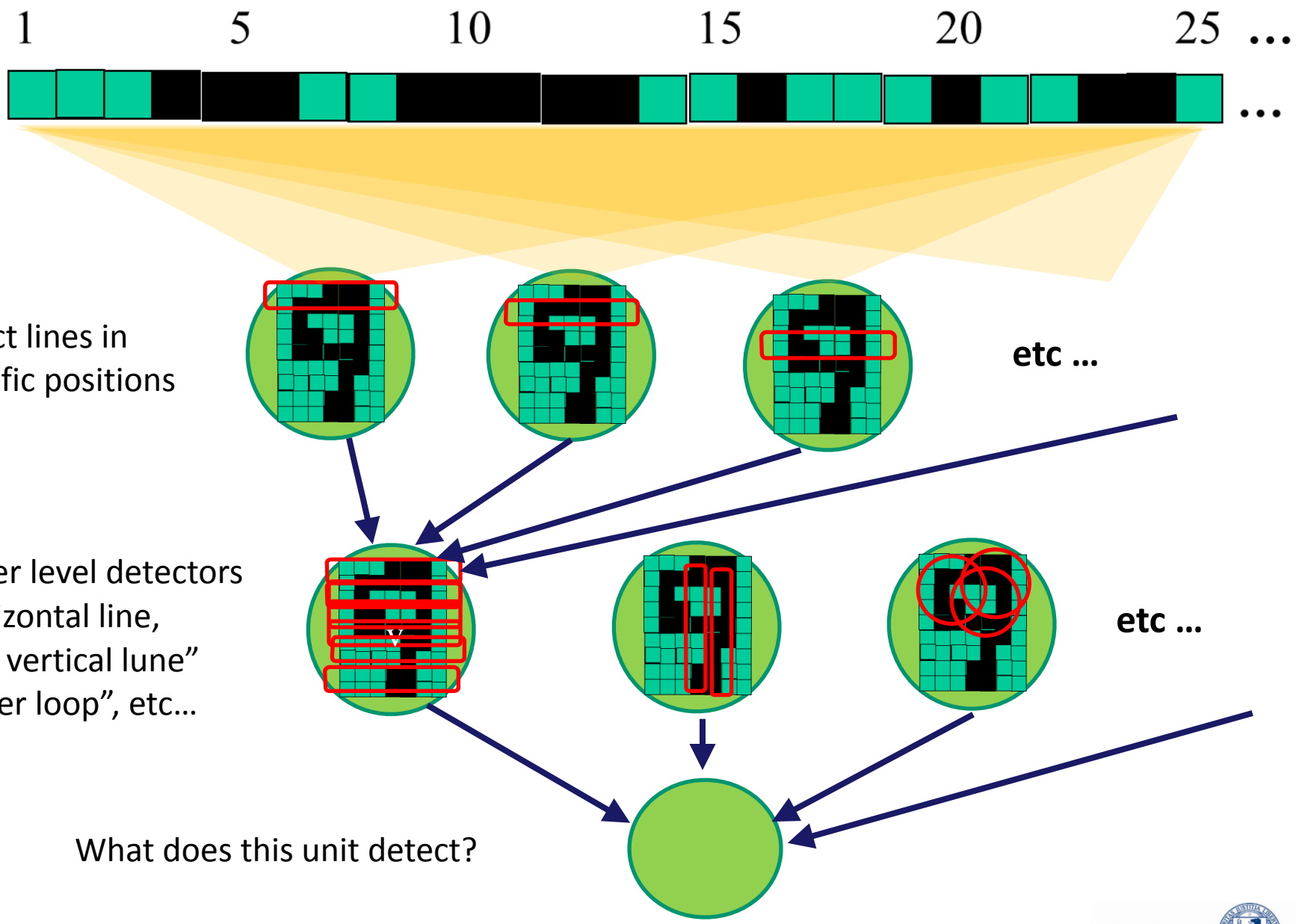


But what about position invariance?
Our example unit detectors were tied to specific parts of the image

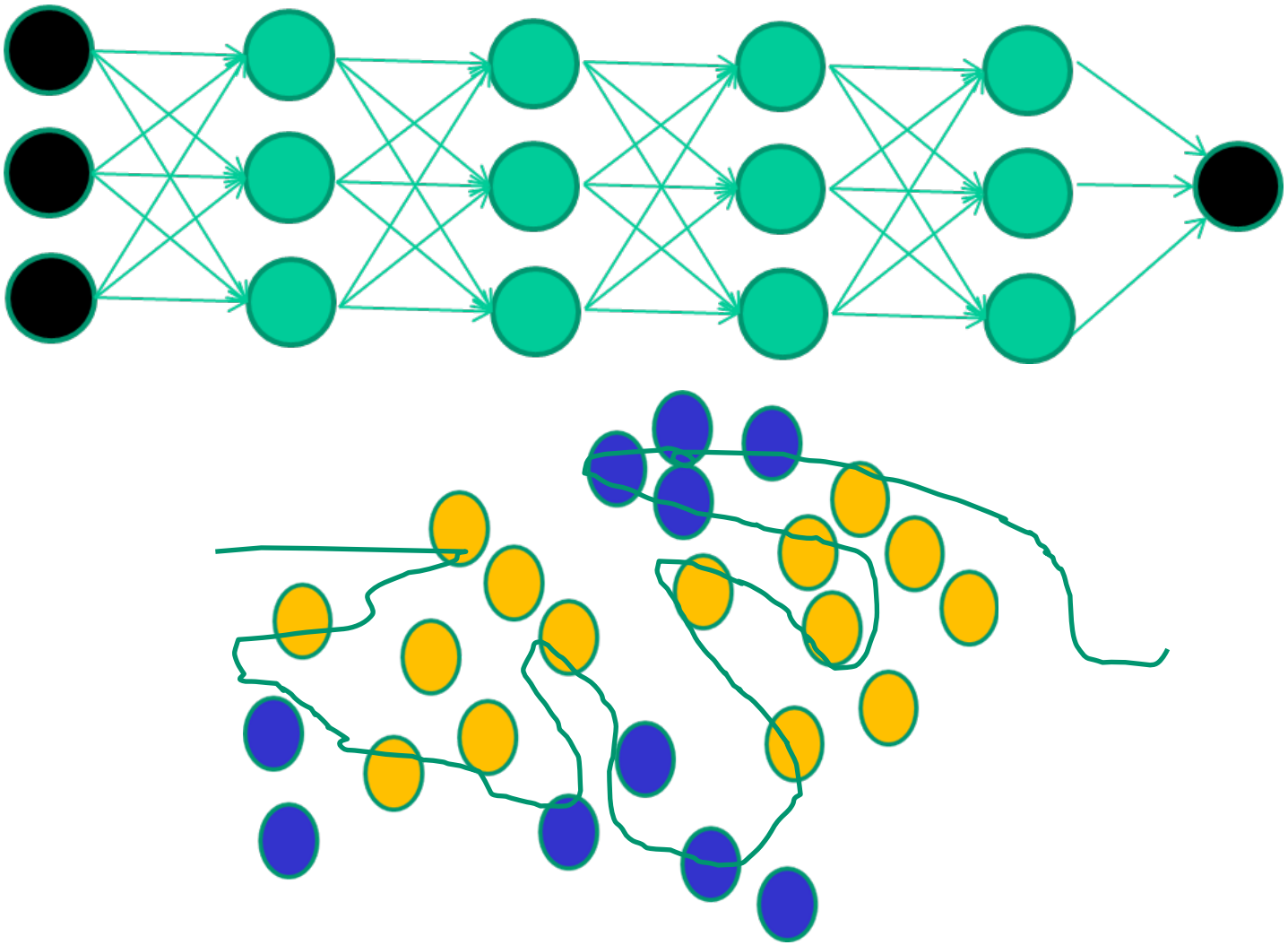
Deep Networks



Successive layers can detect higher-level features



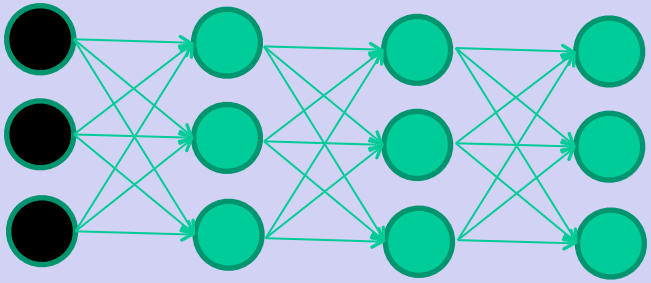
But: until recently deep networks could not be efficiently trained



2006: the deep learning breakthrough

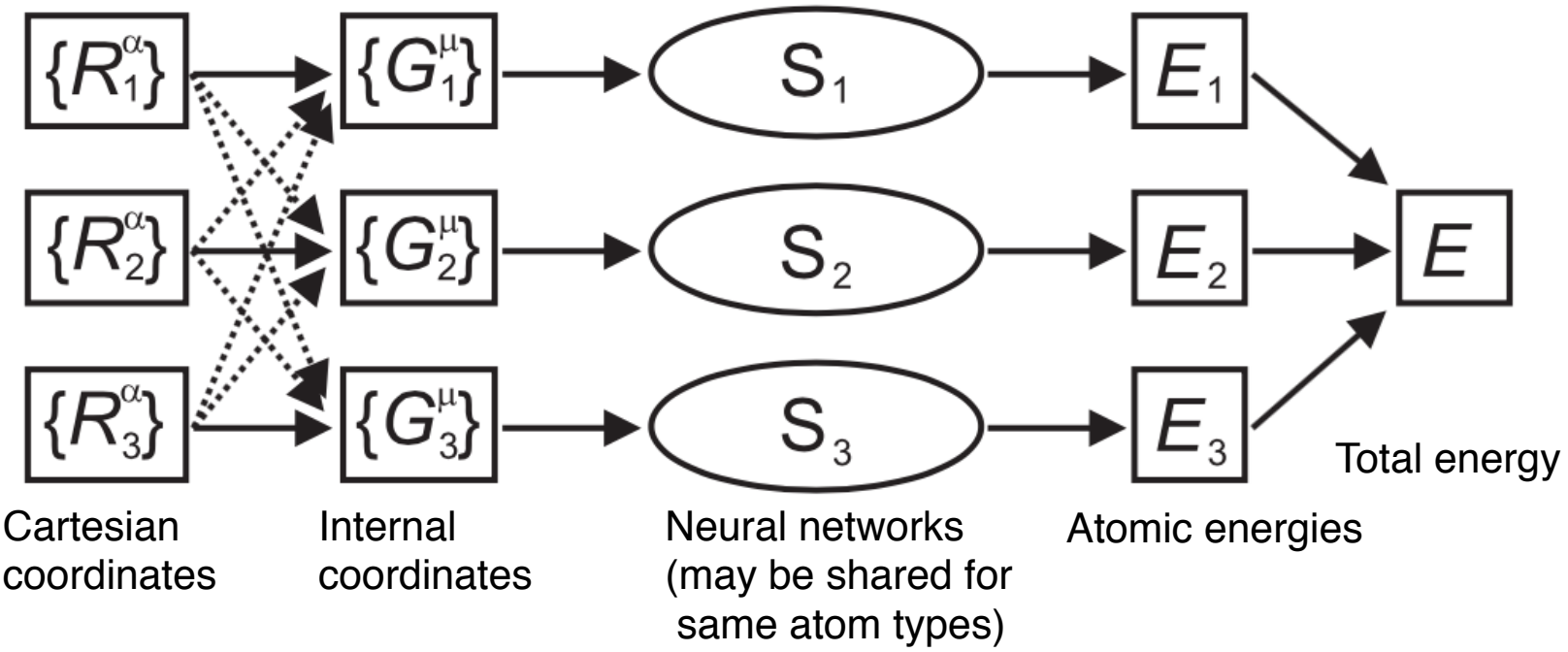
- Hinton, Osindero & Teh « A Fast Learning Algorithm for Deep Belief Nets », *Neural Computation*, 2006
- Bengio, Lamblin, Popovici, Larochelle « Greedy Layer-Wise Training of Deep Networks », *NIPS'2006*
- Ranzato, Poultney, Chopra, LeCun « Efficient Learning of Sparse Representations with an Energy-Based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

Applications to molecular systems



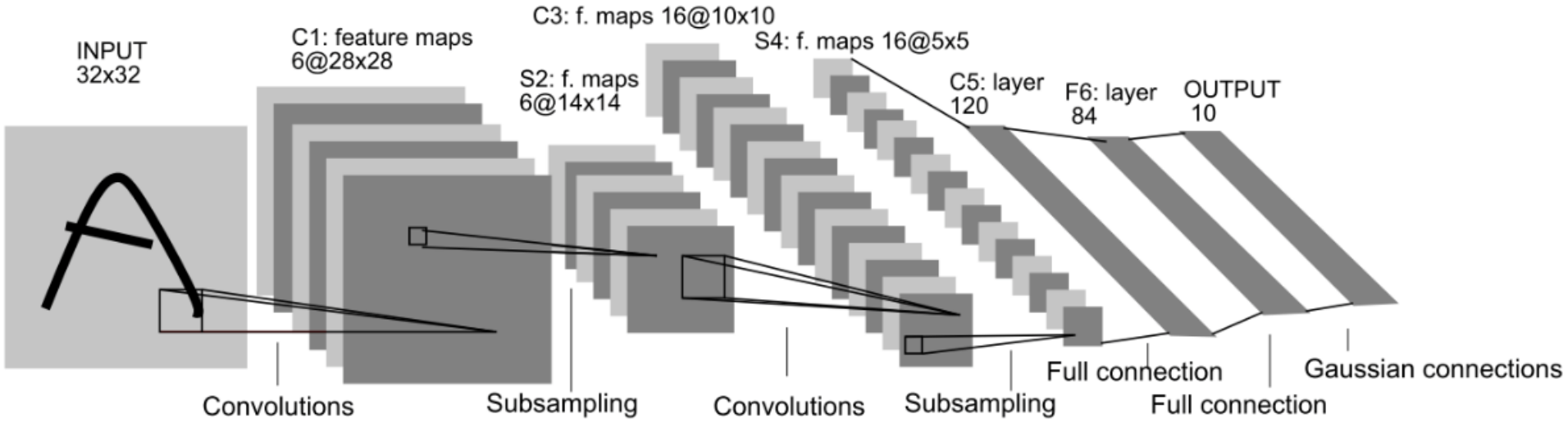
Learning to represent energy functions

Behler-Parrinello network



Behler and Parrinello, PRL 98, 146401 (2007)

Convolutional Neural Networks



LeNet 5

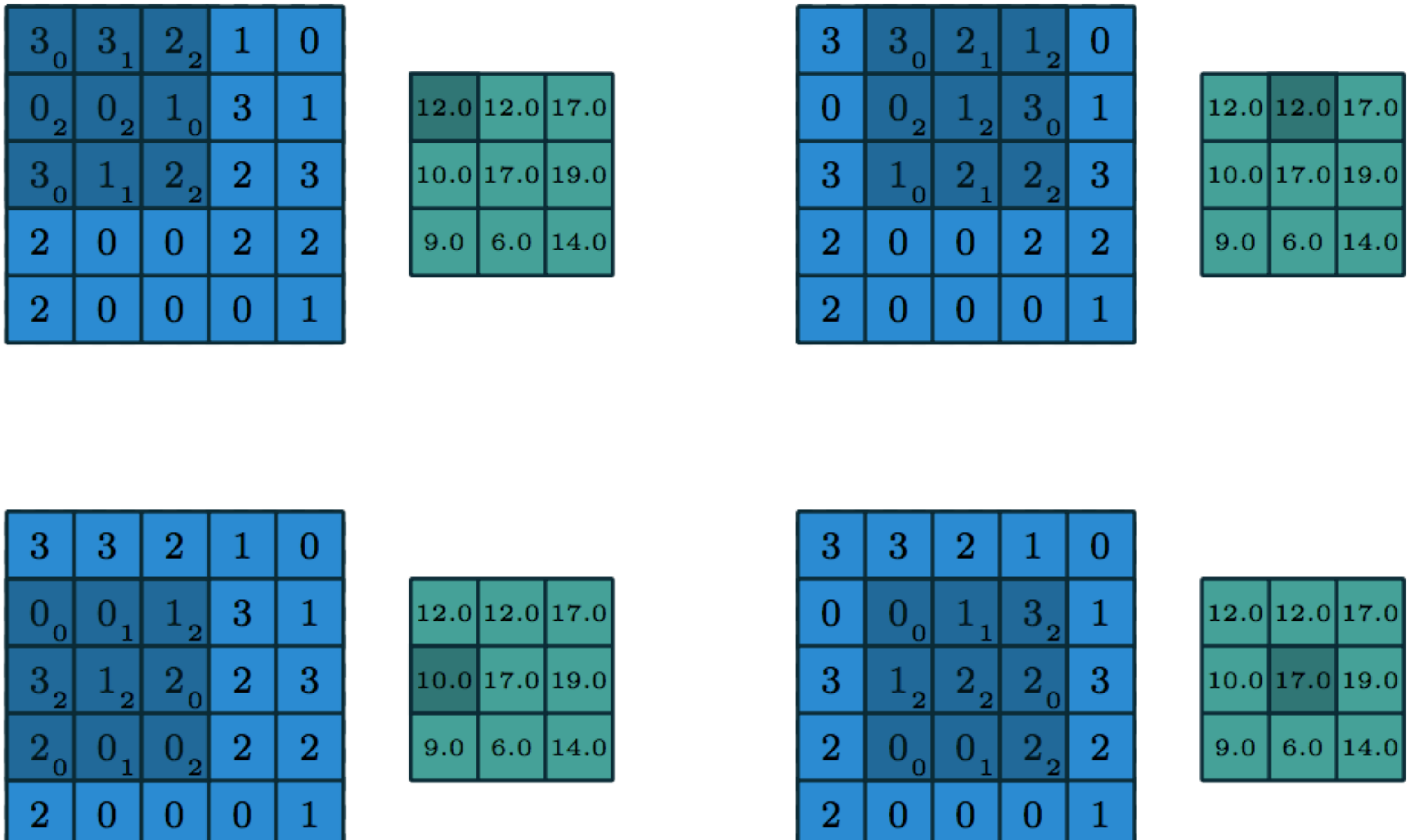
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE, 86(11):2278-2324, November* ***1998***

Convolutional filters

Convolutional Kernel / Filter

0	1	2
2	2	0
0	1	2

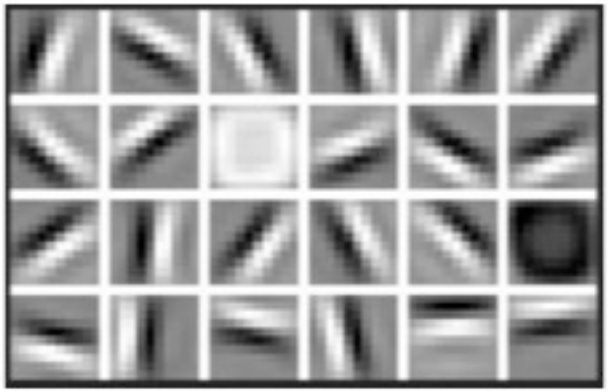
Apply convolutions



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: face recognition



First Layer Representation

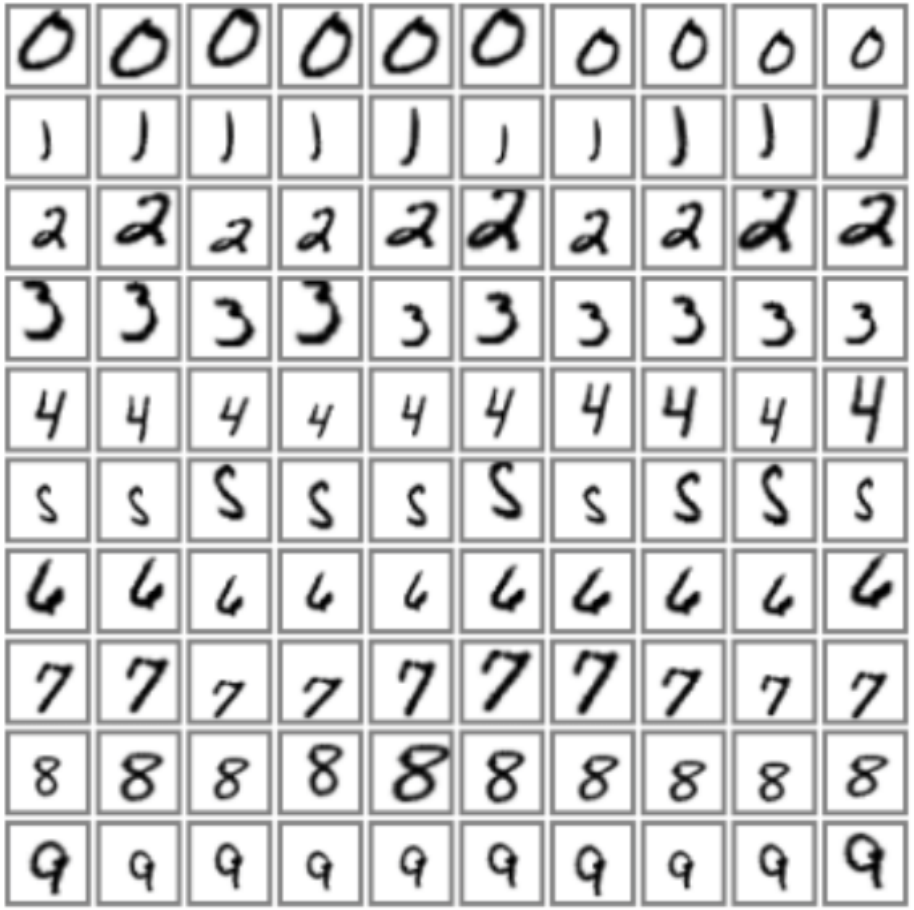
Second Layer Representation

Third Layer Representation

MNIST dataset

60,000 original datasets

Test error: 0.95%



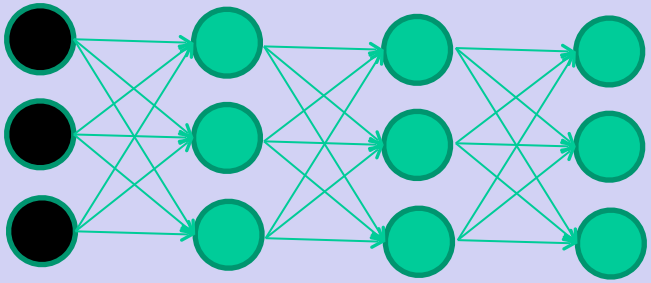
540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

Misclassified examples

Unsupervised learning



Principal component analysis

1. Compute the covariance matrix

$$\mathbf{C}_0 = \frac{1}{T-1}\mathbf{X}^\top\mathbf{X},$$

which is just a scaled version of $\mathbf{X}^\top\mathbf{X}$

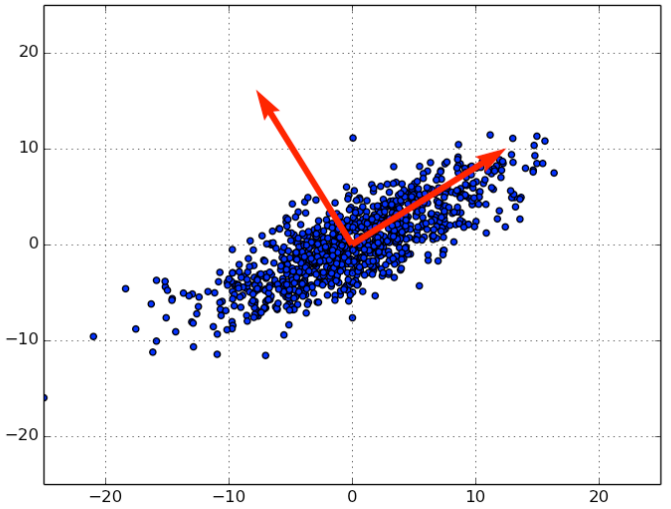
2. Solve the Eigenvalue problem:

$$\mathbf{C}_0\mathbf{w}_i = \sigma_i^2\mathbf{w}_i$$

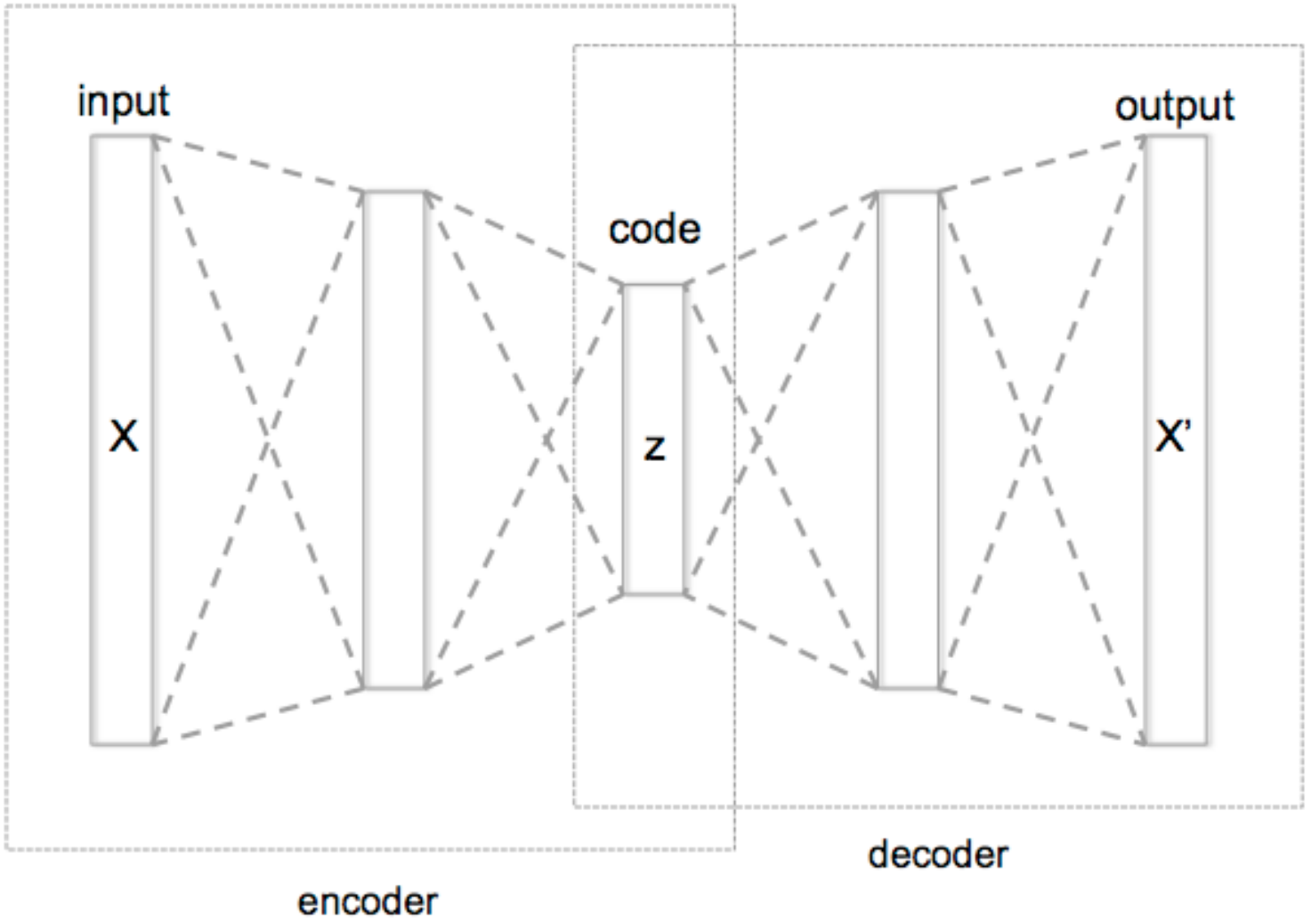
with normalization $\mathbf{w}_i^\top\mathbf{w}_i = 1$.

3. Select m eigenvectors with largest eigenvalues.
4. Reduce dimension from n to m with $\mathbf{W}_m = [\mathbf{w}_1, \ldots \mathbf{w}_m]$:

$$\mathbf{Y}_m = \mathbf{X}\mathbf{W}_m.$$

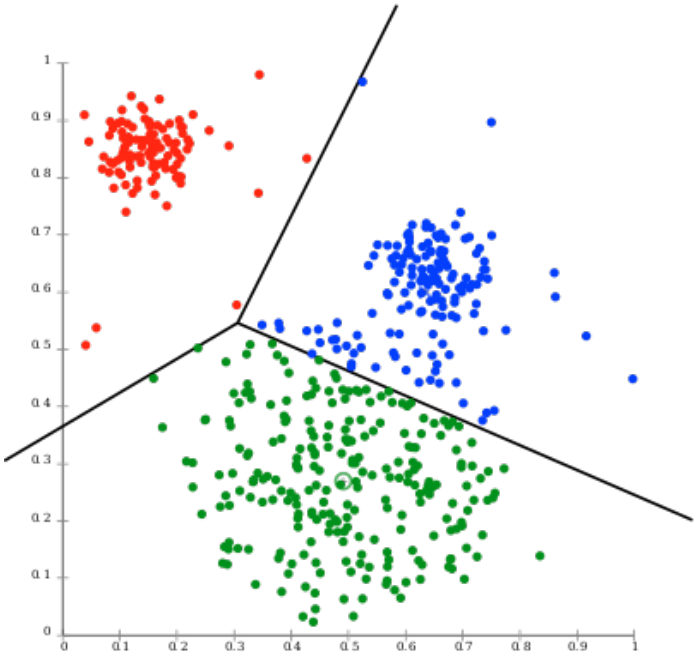


Autoencoder

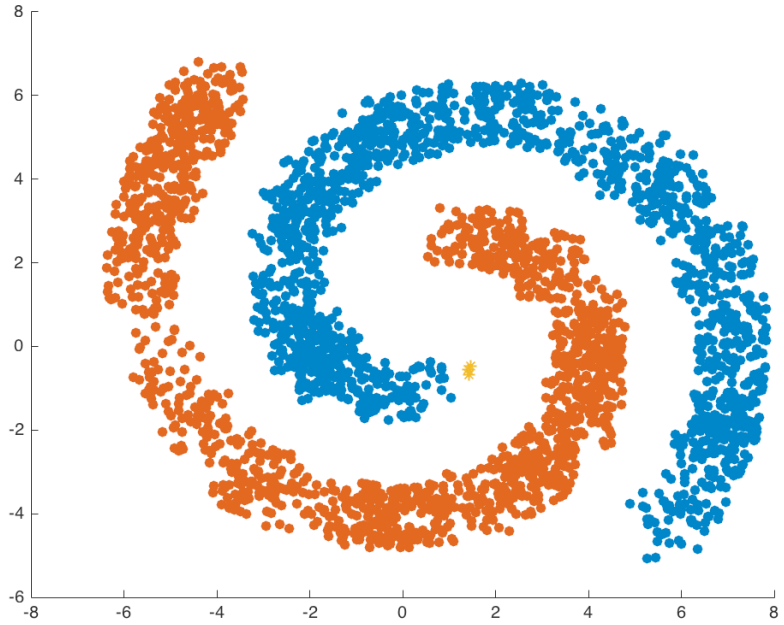


$$\text{Loss}(\mathbf{x};\theta) = \sum_i [\hat{y}_i(\mathbf{x};\theta) - x_i]^2$$

Clustering



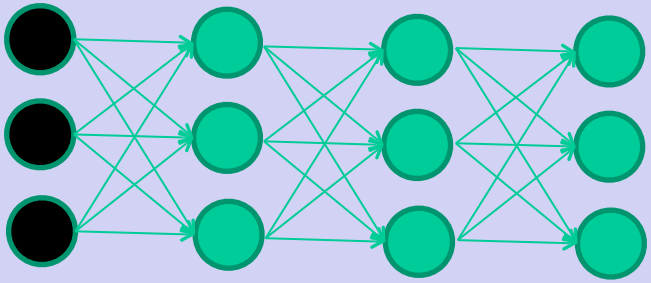
k-means



dbscan

Neural networks: e.g. Gaussian mixture variational autoencoders

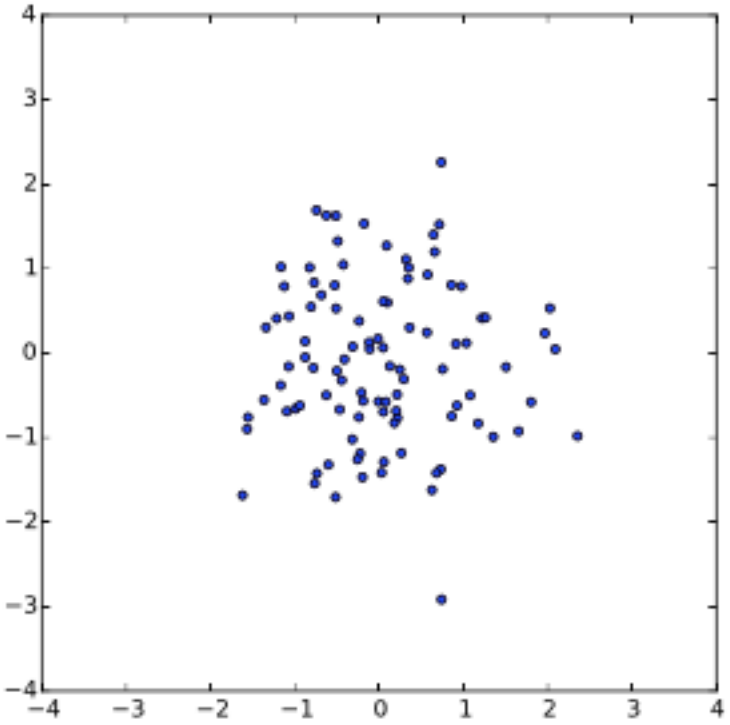
Generative networks

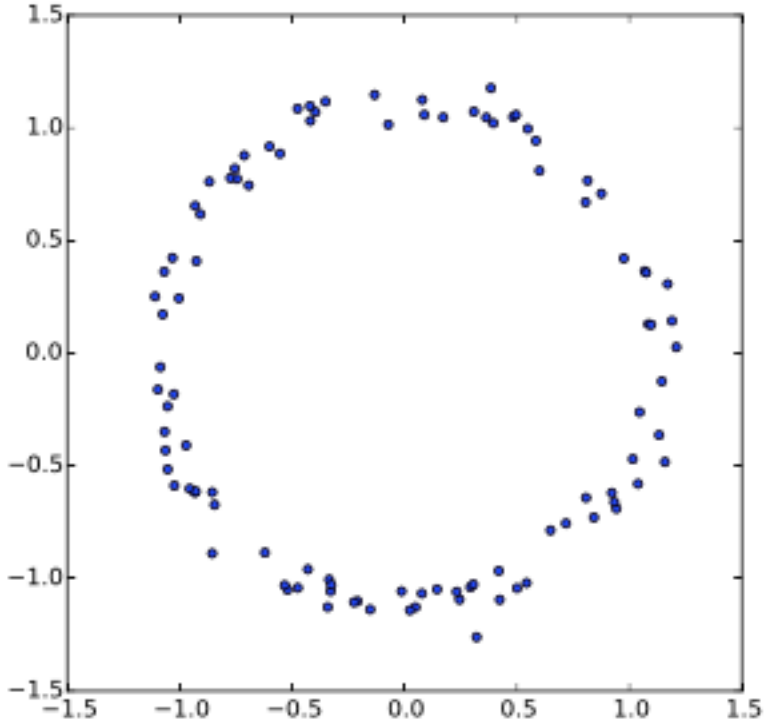


Latent Variables

- “Hidden” or unknown variables
- Encode the essential state of the system
- Determine probabilities of the visible variables and dependency structure between them.
- For every data point X in the dataset, there is one (or many) settings of the latent variables which causes the model to generate something very similar to X.
- Formally, say we have a vector of latent variables z in a high-dimensional space Z which we can easily sample according to some probability density function (PDF) $P(z)$ defined over Z

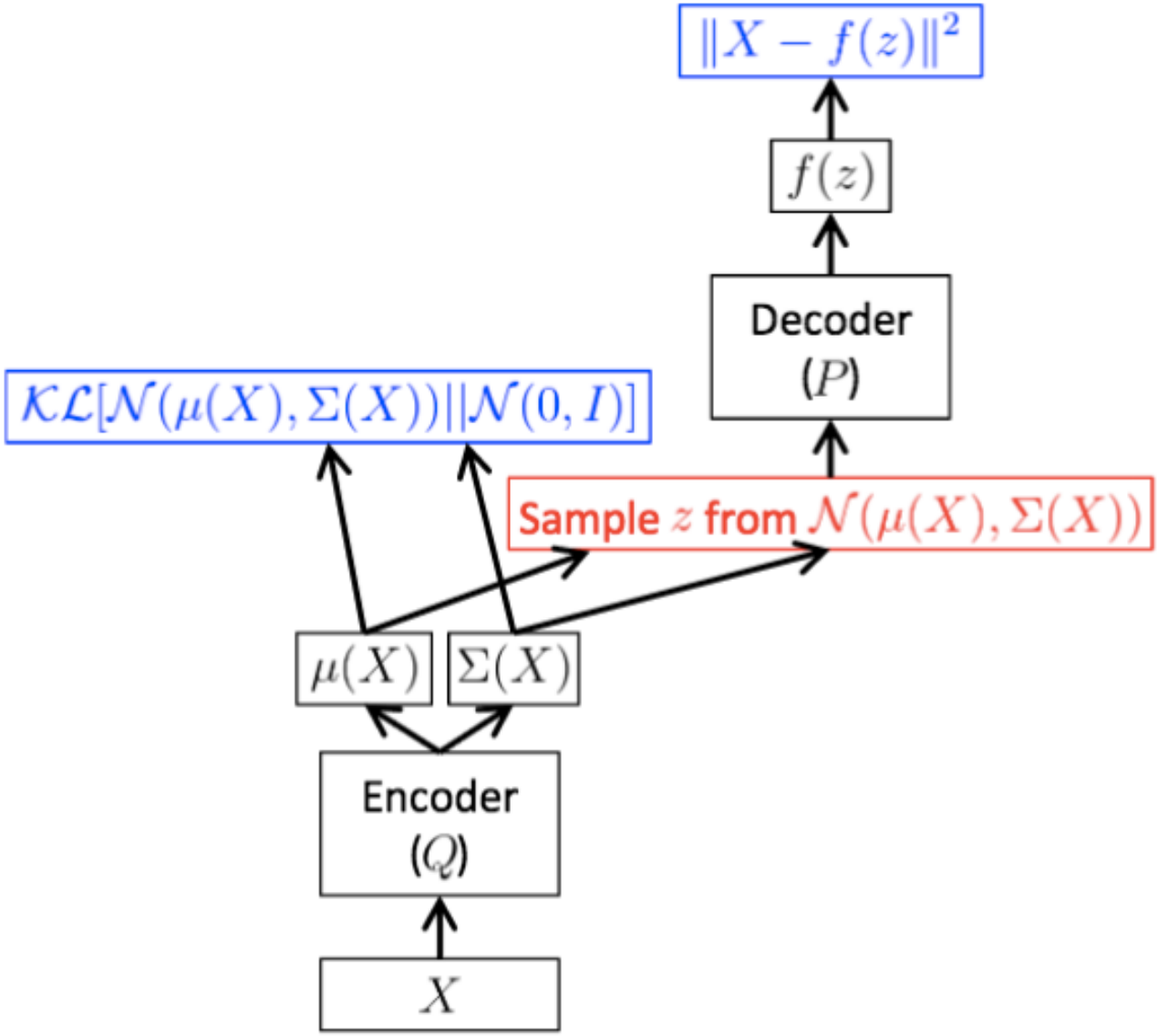
Variational Autoencoder





Kingma and Welling: Auto-Encoding Variational Bayes arXiv:1312.6114 (2013)

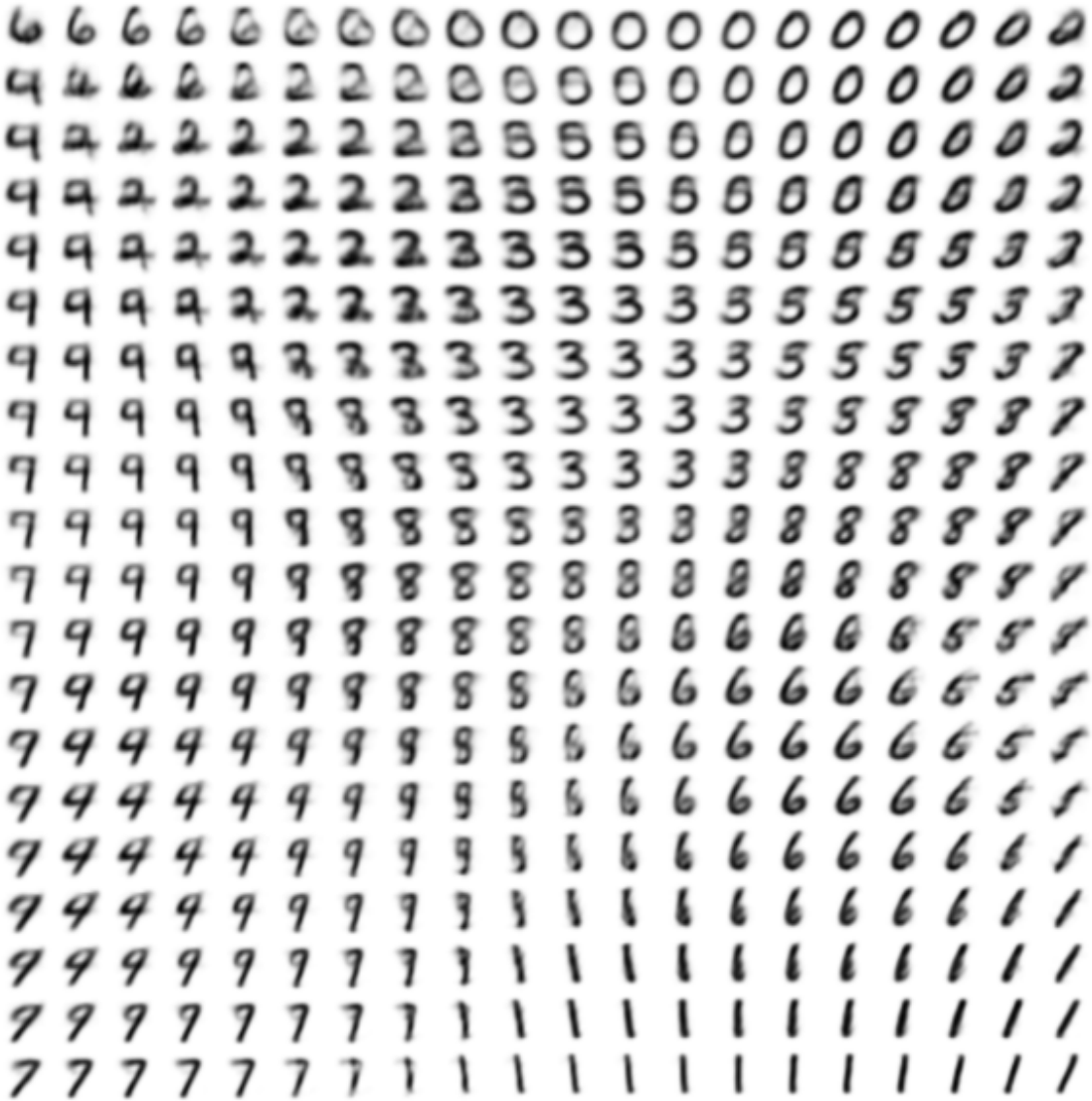
Variational Autoencoder



Kingma and Welling: Auto-Encoding Variational Bayes arXiv:1312.6114 (2013)

Variational Autoencoder

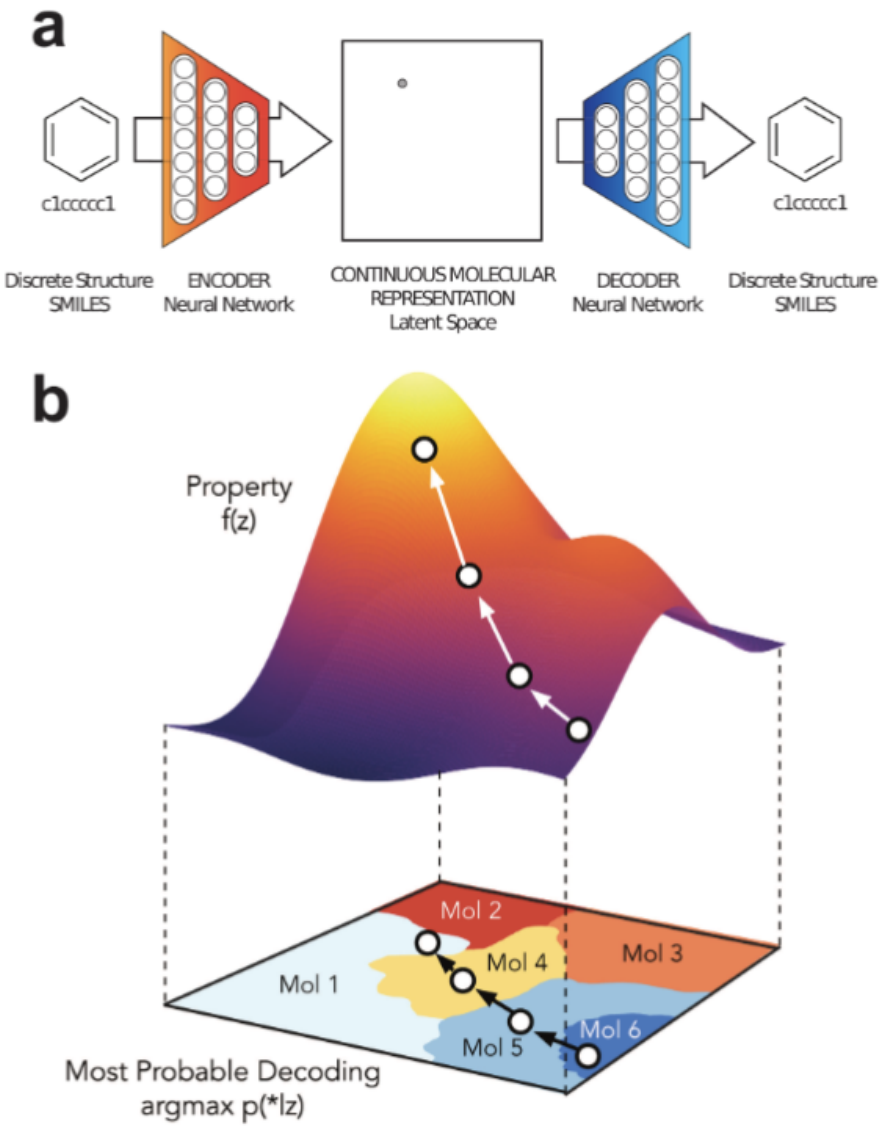
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

Kingma and Welling: Auto-Encoding Variational Bayes arXiv:1312.6114 (2013)

Variational Autoencoder



Rafael Gómez-Bombarelli et al: Automatic chemical design using a data-driven continuous representation of molecules

Variational Autoencoder

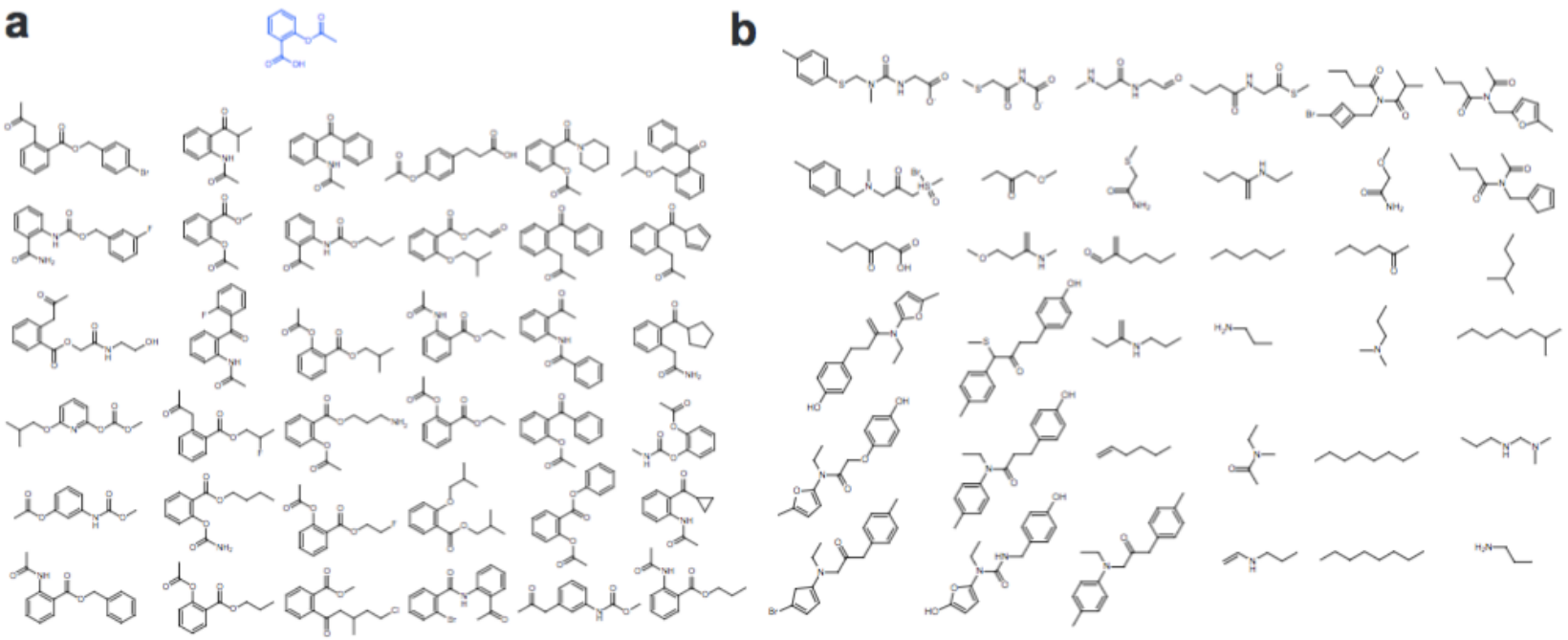
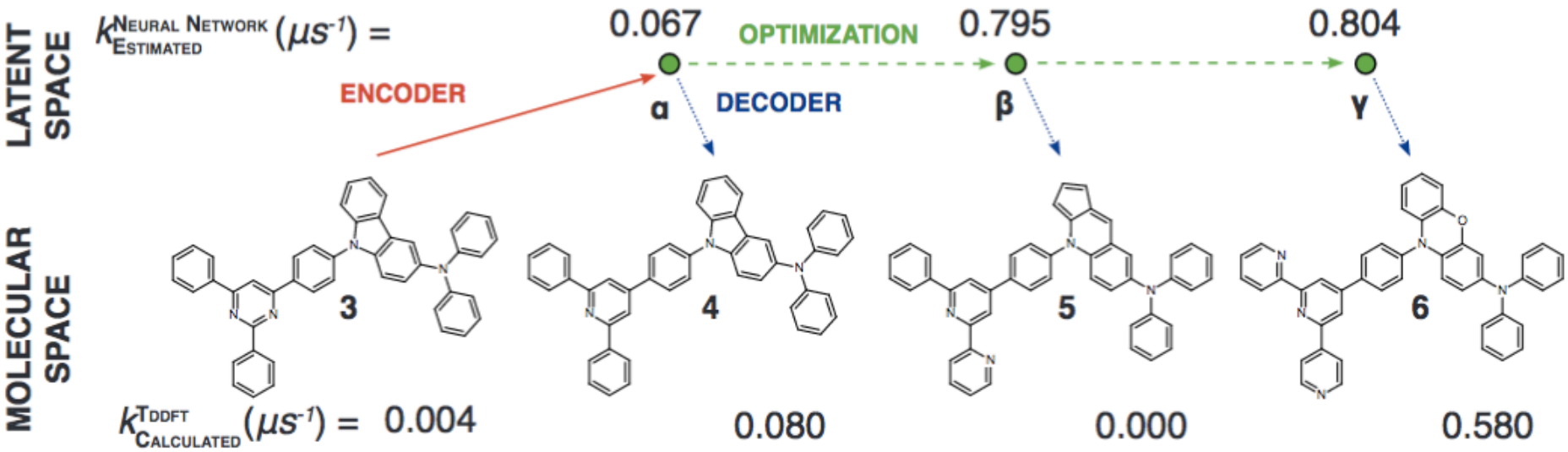
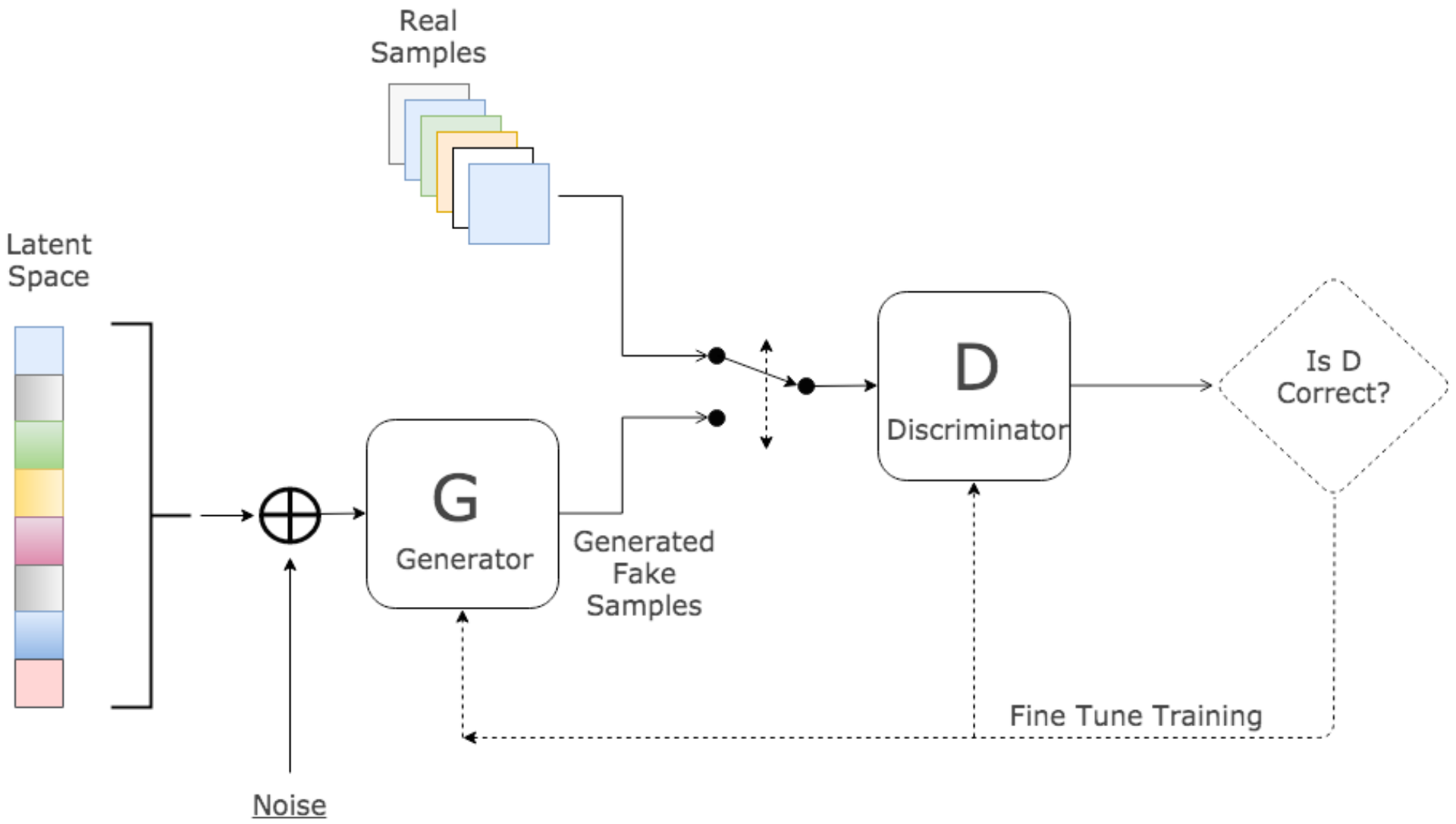


Figure 3: **a)**. Random sampling. Molecules decoded from randomly-sampled points in the latent space of a variational autoencoder, near to a given molecule (aspirin [2-(acetyloxy)benzoic acid], highlighted in blue). **b)**. Interpolation. Two-dimensional interpolation between four random points in in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

Variational Autoencoder



Generative Adversarial Network

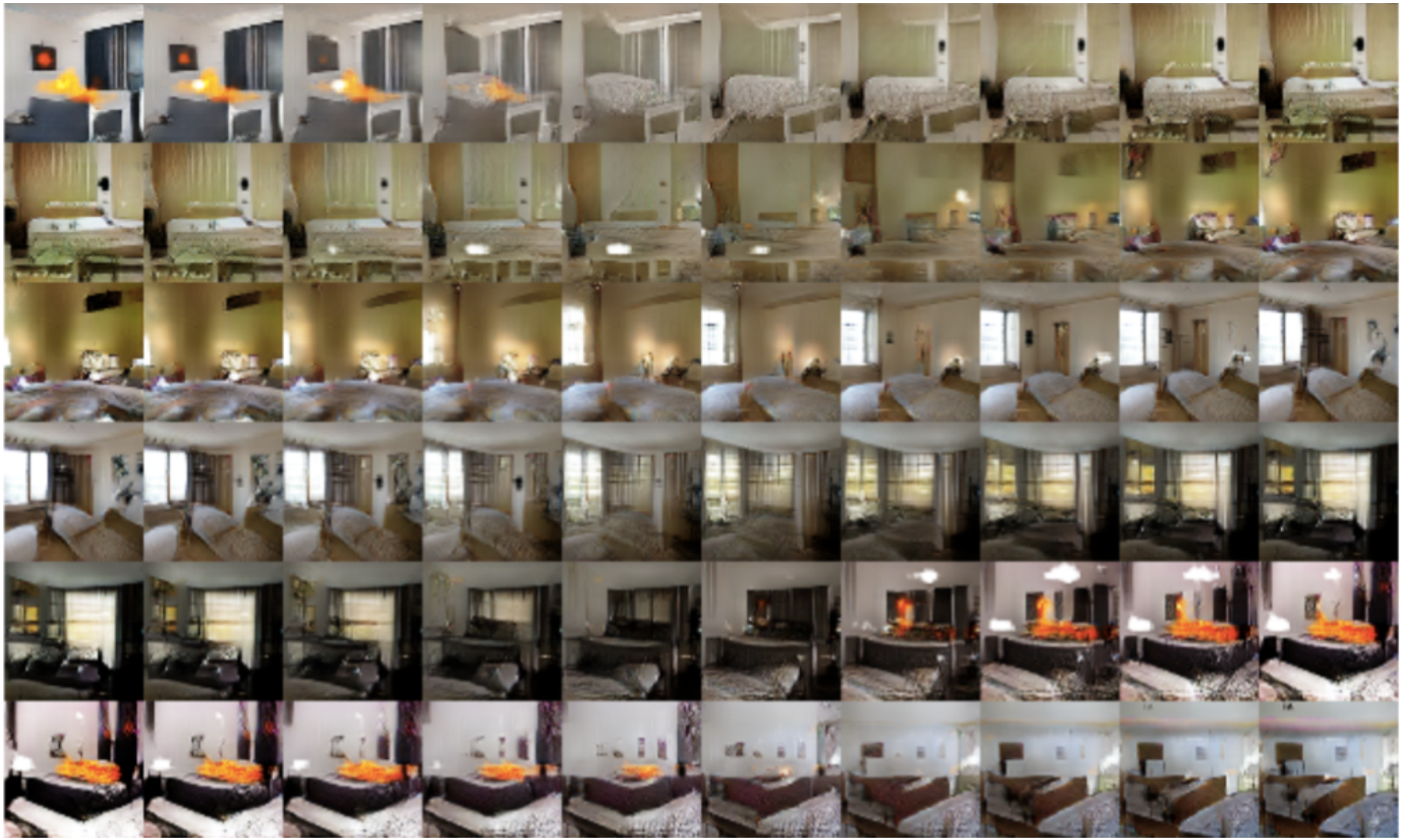


Goodfellow et al. "Generative Adversarial Networks". arXiv:1406.2661 (2014)

Generative Adversarial Network

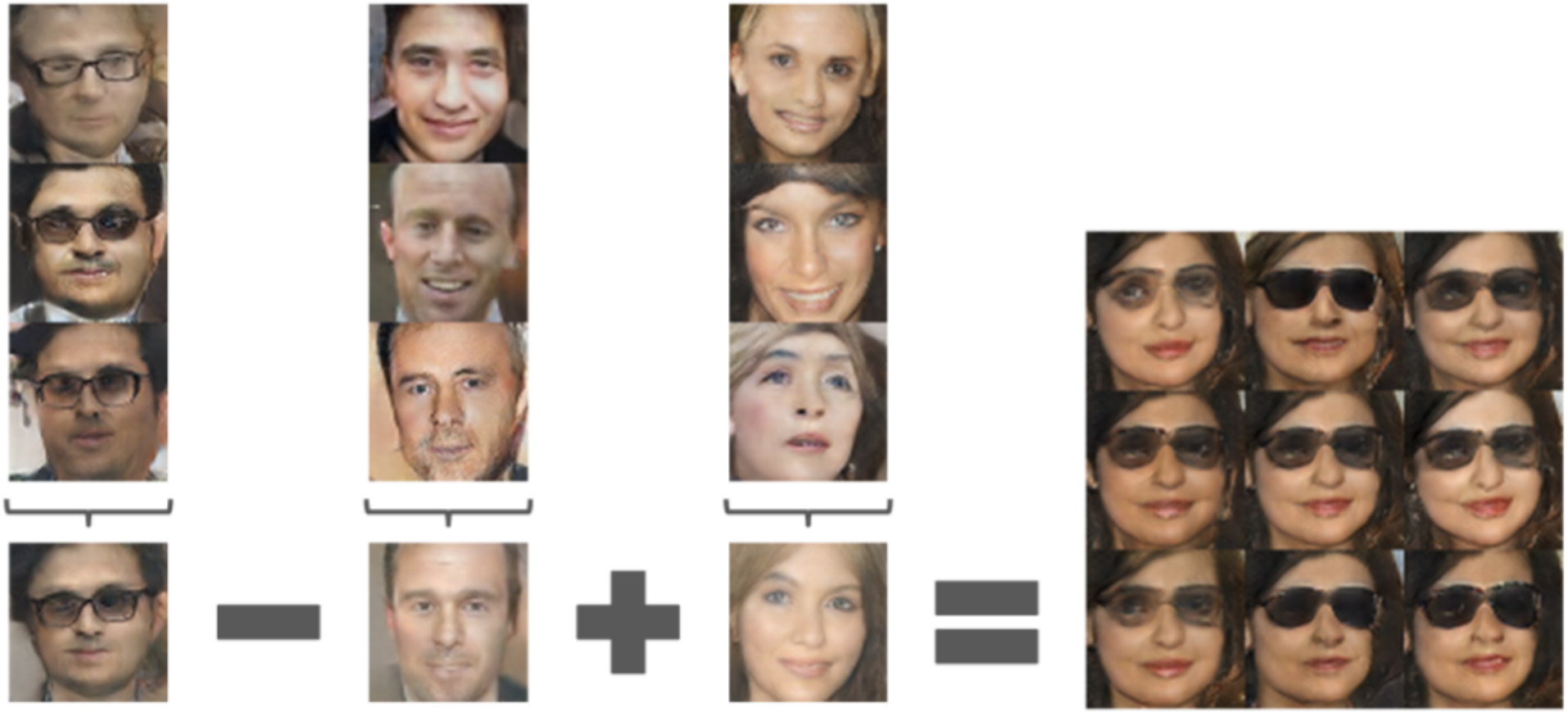
Radford et al: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks arXiv:1511.06434 (2015)

Generative Adversarial Network



Radford et al: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks arXiv:1511.06434 (2015)

Generative Adversarial Network



Radford et al: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks arXiv:1511.06434 (2015)

DEEP
LEARNING

MACHINE
LEARNING

ARTIFICIAL
INTELLIGENCE