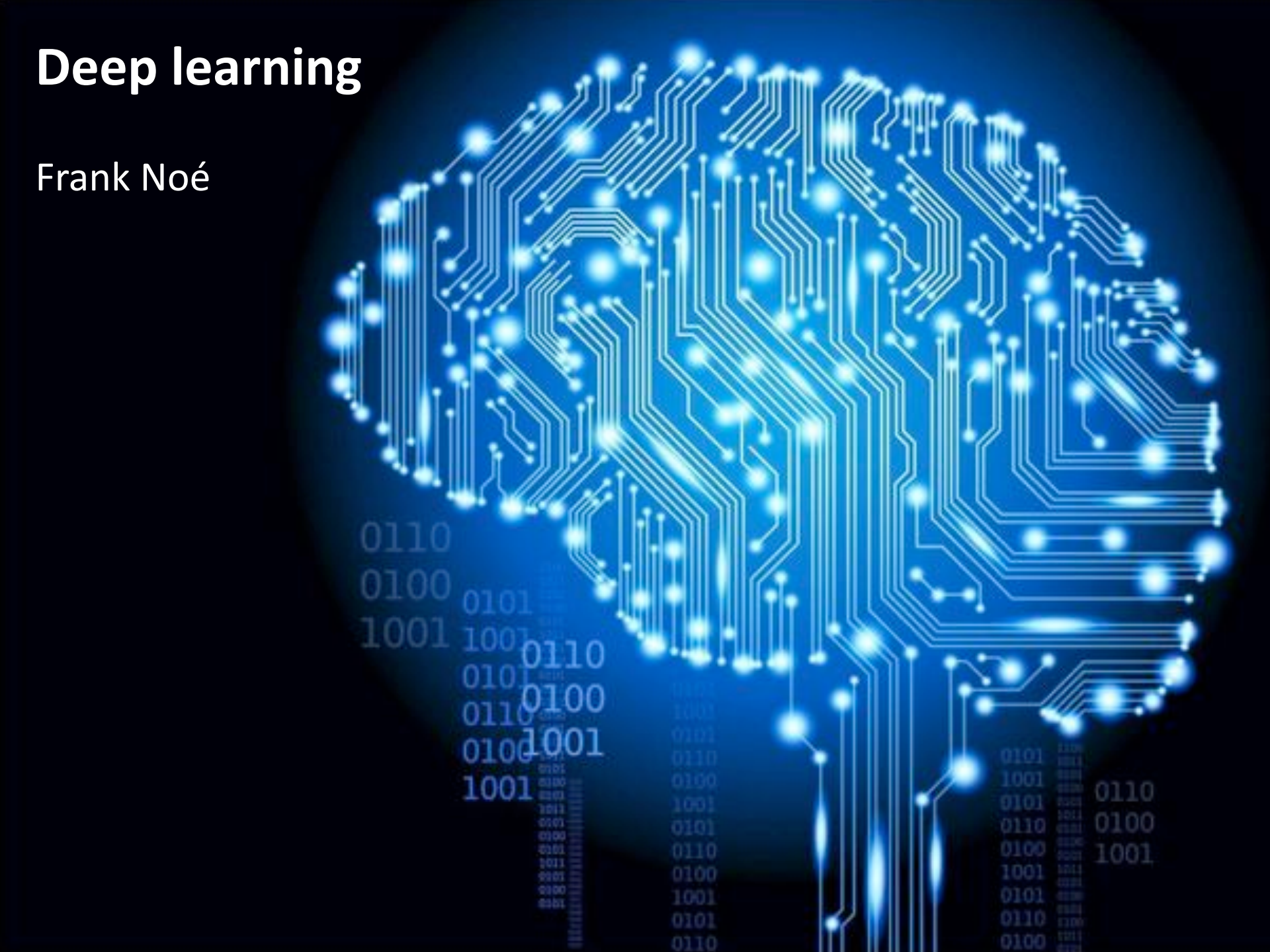
Deep learning

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

Deep learning

WaveNet

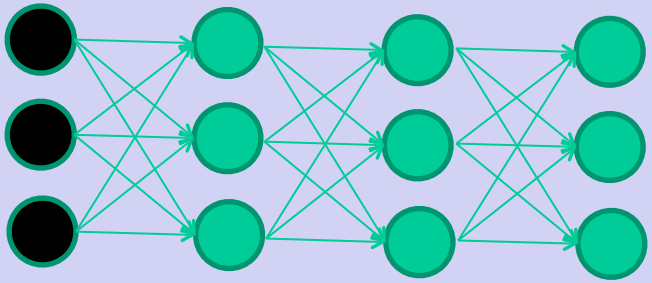
Deep learning

WaveNet

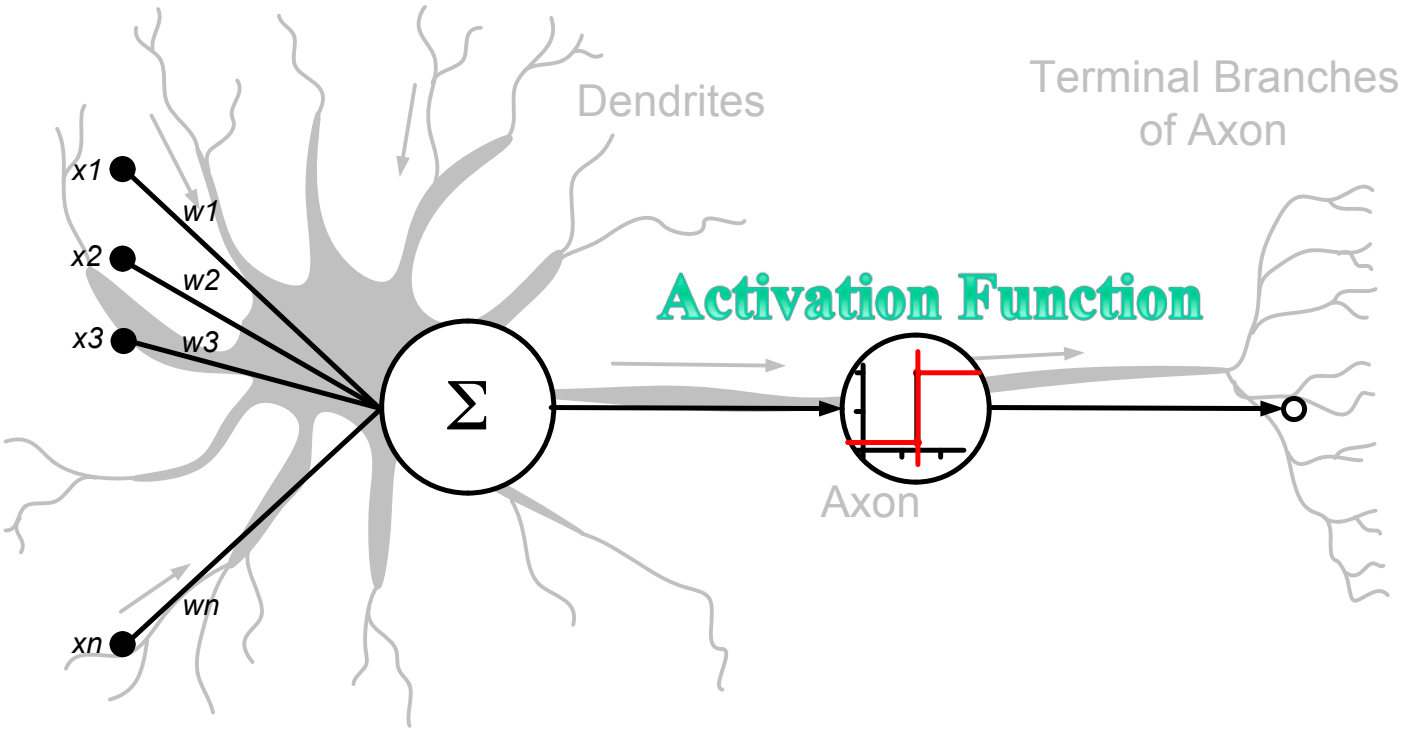
Deep learning

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.

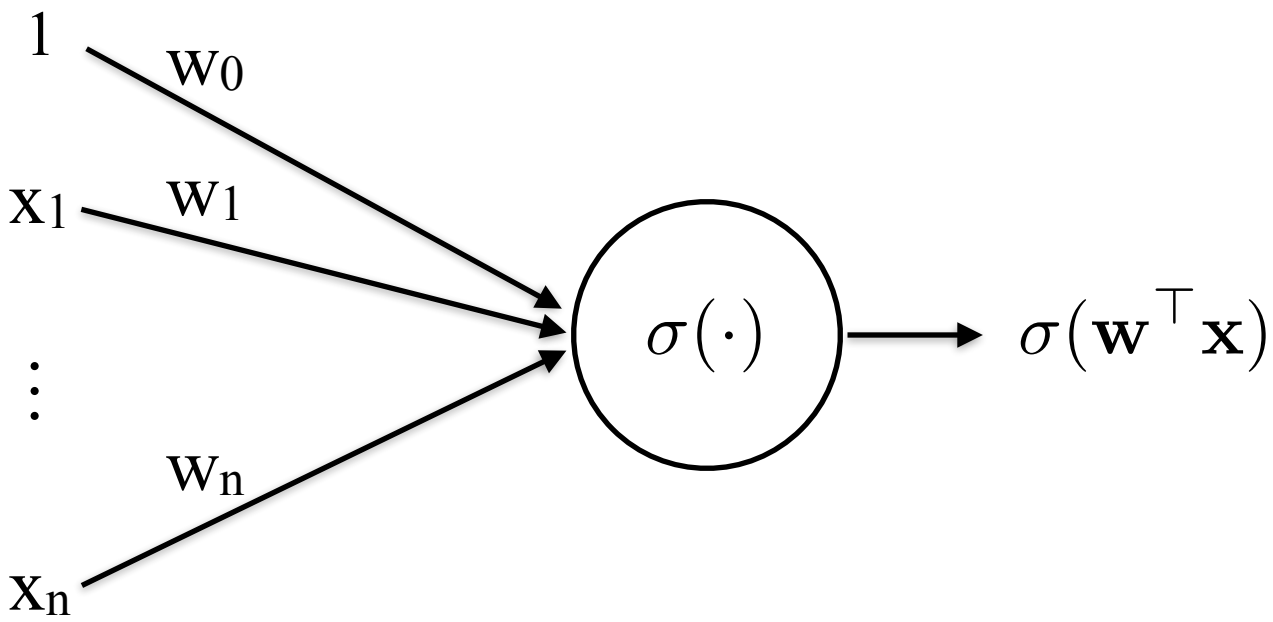
Supervised learning with feedforward neural networks



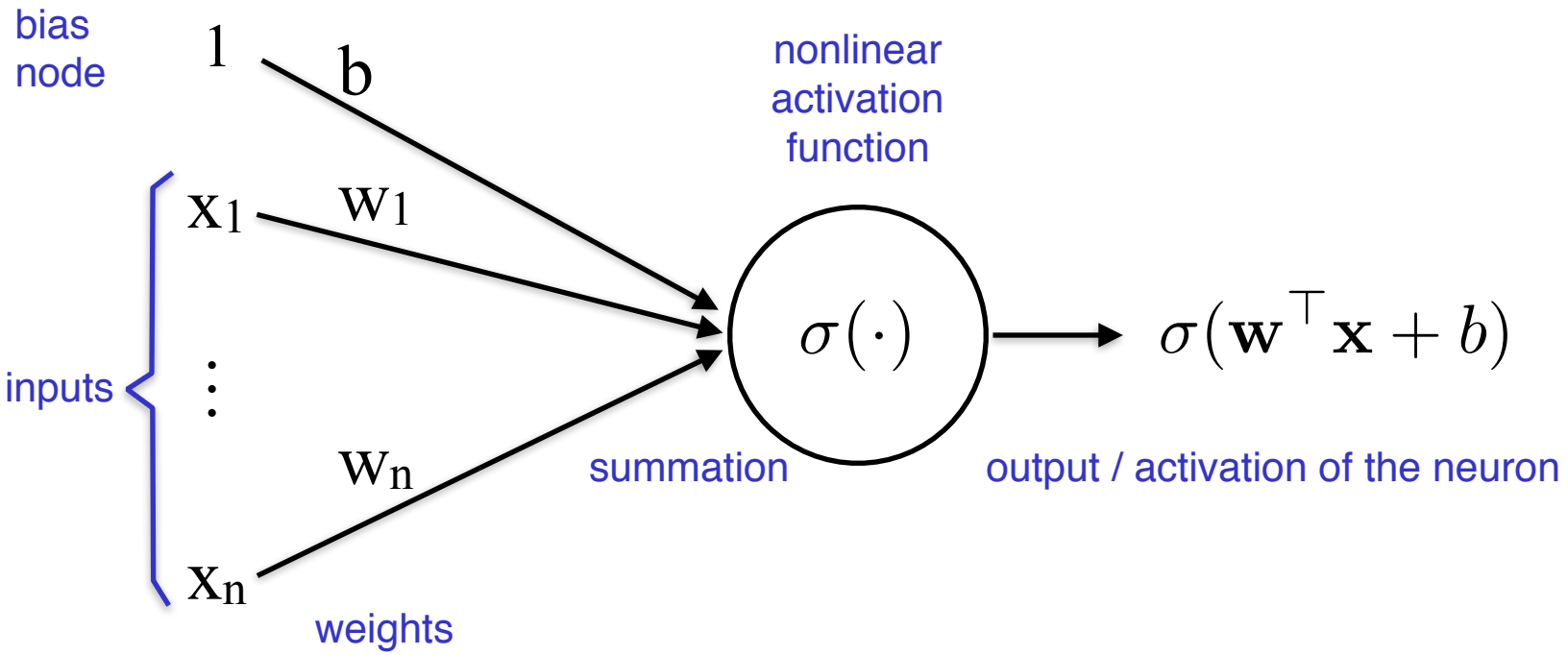
Neuron


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

Artificial neuron

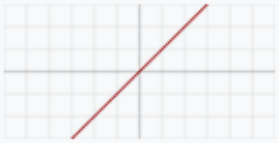
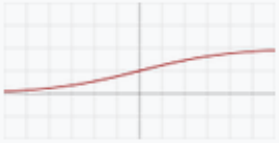
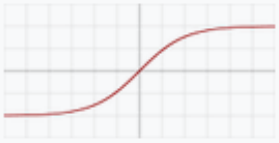
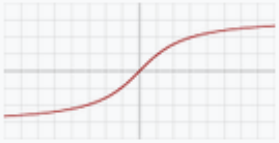
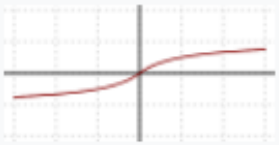


Artificial neuron



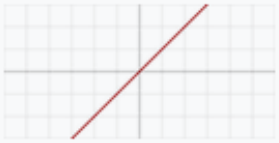
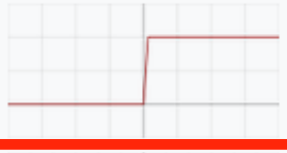
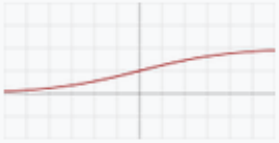
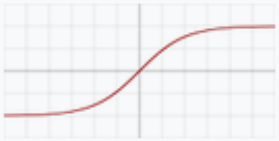
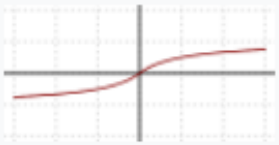
Activation functions

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

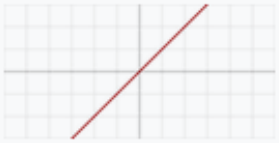
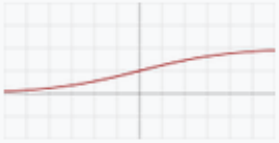
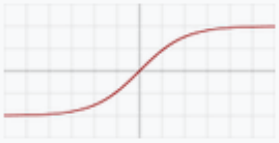
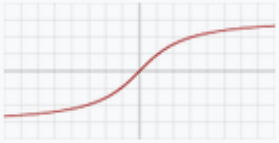
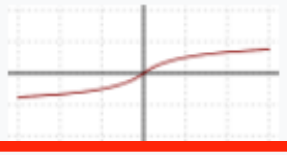
Activation functions

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

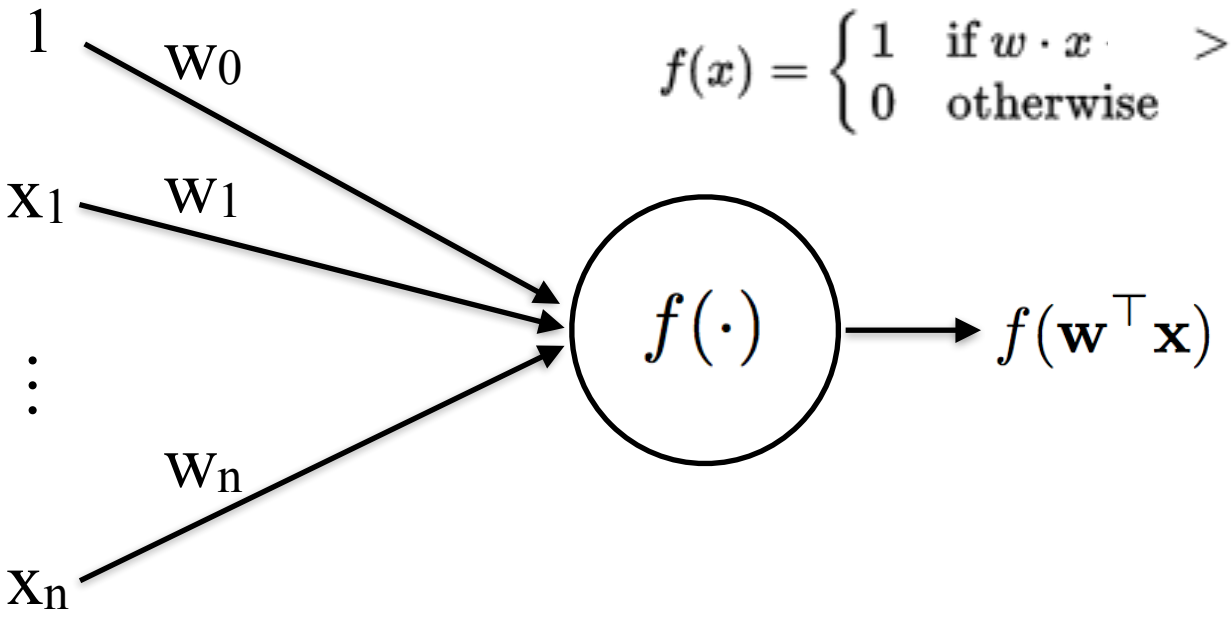
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1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

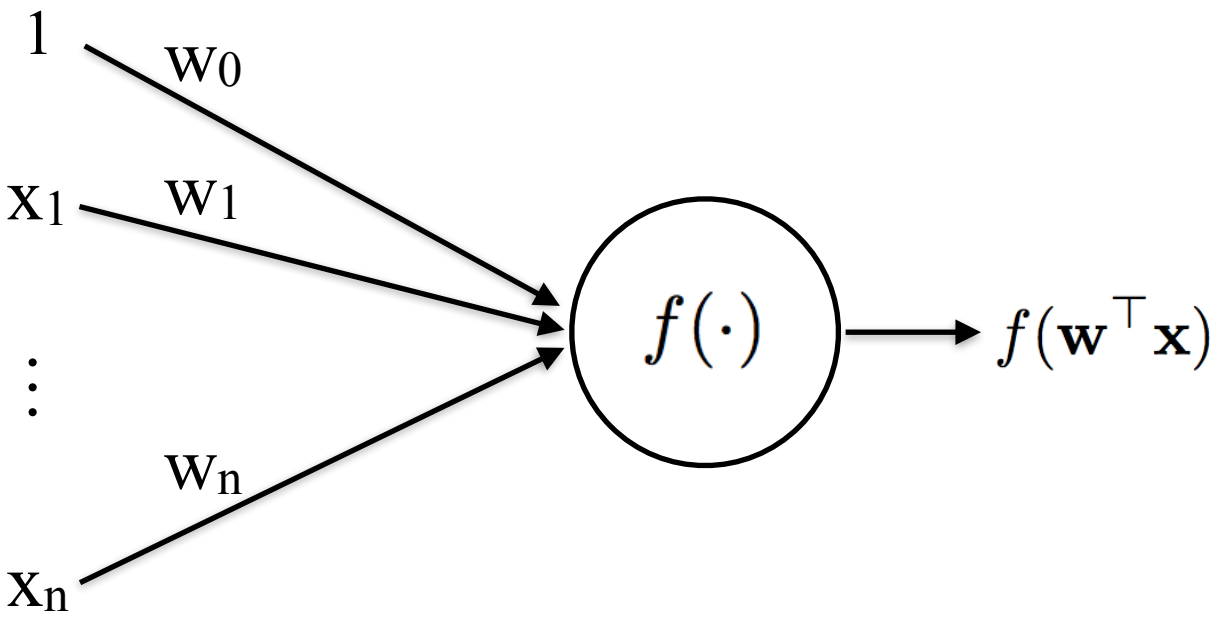
Perceptron (Rosenblatt 1957)

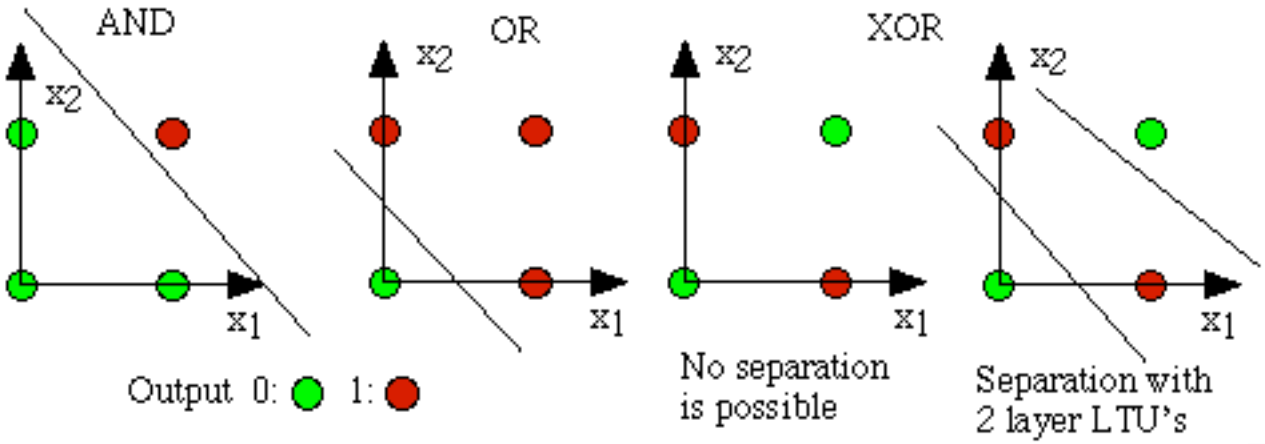
Used as a binary classifier - equivalent to support vector machine (SVM)



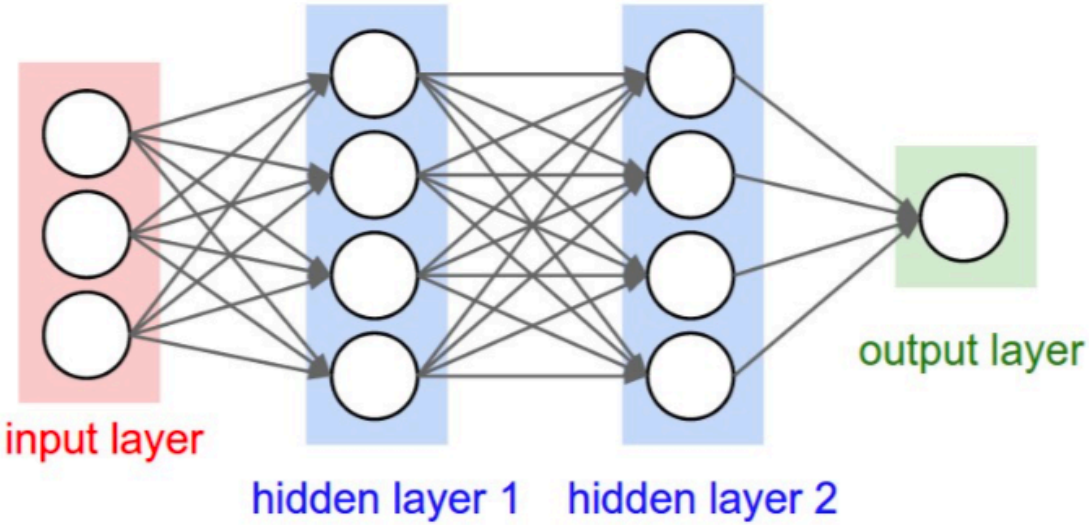
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x \quad > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron (Rosenblatt 1957)





Multilayer perceptrons (~1985)



- **Network architecture** is typically the most complex model decision:
  - Number of hidden layers $L-1$ and number of neurons $(n_1, \ldots n_{L-1})$.
  - Type of activation functions σ.
  - Layer types (dense, convolutional, etc. – more details later)
  - Additional regularization terms, e.g. promoting sparsity.
- **Choosing the network architecture**:
  - In principle: Hyperparameter selection problem.
  - In practice: Engineering problem, problem specific architecture.
  - Depends on amount and type of data and available computational resources.

Universal Representation Theorem

Hornik et al., 1989; Cybenko, 1989

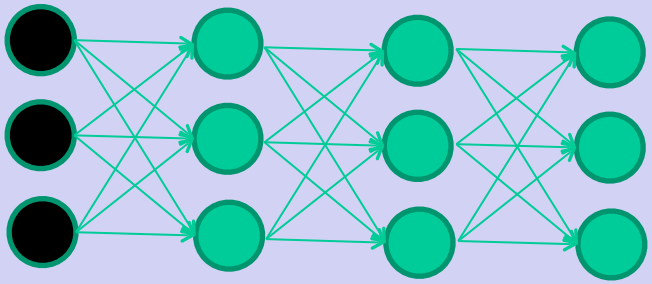
Sloppy formulation: *A neural network with **a single** hidden layer and a monotonically increasing nonlinearity σ can approximate any continuous function $F : \mathbb{R}^{n_0} \mapsto \mathbb{R}^{n_2}$ from inputs $\mathbb{R}^{n_0}$ to outputs $\mathbb{R}^{n_2}$ with arbitrary accuracy given that sufficiently many hidden neurons n_1 are provided.*

- *Proof for sigmoid activation functions*: G. Cybenko: "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems 2, 303-314 (1989)
- *Generalization*: K. Hornik: "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks 4, 251-257 (1991)

Caveats:

- Existence of network parameters that approximate F does not mean these parameters can be efficiently found.
- Approximation of a function does not mean exact representation, error might be too large for practical purposes.
- For many complex functions, the number of hidden neurons required to achieve acceptable accuracy is unpractical. $\rightarrow$ deep neural networks.

Parameter optimization (weight training)



Gradient descent of loss function

Cost or **loss function** C quantifies performance of network with parameters θ to predict observations $\mathbf{X}$.

Learning network weights

$$\hat{\theta} = \arg\min_\theta C(\mathbf{X}, \mathbf{Y}, \theta)$$

Minimimizing *cost* $C \equiv$ maximizing *score* $-C$. Often we just write $C(\theta)$.

Neural networks are usually trained with some form of gradient descent:

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- Networks are functions of functions, so we will need to use the chain rule of differentiation.
- Key is to make differentiation fully automatic so that we can just define the network architecture without worrying about functions and gradients.
  $\rightarrow$ **Backpropagation**.

Backprop

- Activation of a single neuron:

$$x_i^l = \sigma\left(z_i^l\right)$$
$$z_i^l = \sum_j w_{ij}^l x_j^{l-1} + b_i^l$$

$$C(\mathbf{X}, \theta) = \frac{1}{N}\sum_{k=1}^{N} c(\mathbf{x}_k, \mathbf{y}_k, \theta)$$
$$\frac{\partial C(\mathbf{X}, \theta)}{\partial \hat{y}_{k,i}} = \frac{1}{N}\sum_{k=1}^{N} \frac{\partial c(\mathbf{x}_k, \mathbf{y}_k, \theta)}{\partial \hat{y}_{k,i}}$$

- **Error**: Loss gradient with respect to the ith weighted input

$$\mathbf{e}_i^L = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^L} = \underbrace{\frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial \hat{y}_i}}_{\text{loss derivative}} \frac{\partial \hat{y}_i}{\partial z_i^L} = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial \sigma(z_i^L)} \underbrace{\sigma'(z_i^L)}_{\text{activation derivative}}$$

- **Backpropagate error** to earlier layers

$$\mathbf{e}_i^l = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^l} = \sigma'(z_i^l)\sum_j \mathbf{e}_j^{l+1} w_{ji}^{l+1}$$

- **Weight gradients** at each layer

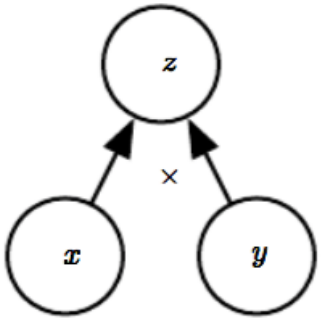
$$\frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial b_i^l} = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial b_i^l} = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^l} = \mathbf{e}_i^l$$
$$\frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial w_{ij}^l} = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l} = \frac{\partial c(\mathbf{x}, \mathbf{y}, \theta)}{\partial z_i^l} x_j^{l-1} = \mathbf{e}_i^l x_j^{l-1}$$

- **Weight update**:

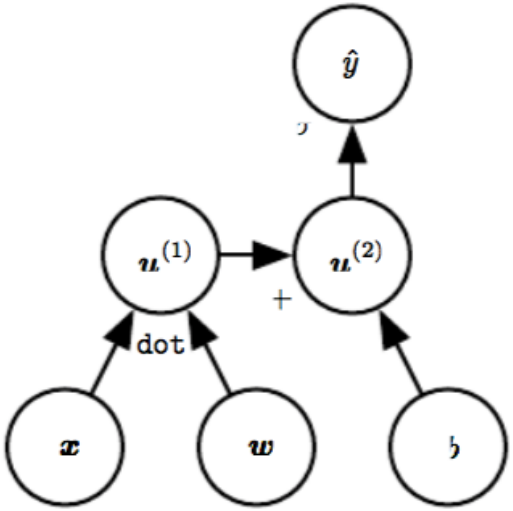
$$b_i^l \leftarrow b_i^l - \eta \mathbf{e}_i^l$$
$$w_{ij}^l \leftarrow w_{ij}^l - \eta \mathbf{e}_i^l x_j^{l-1}$$

with learning rate η.

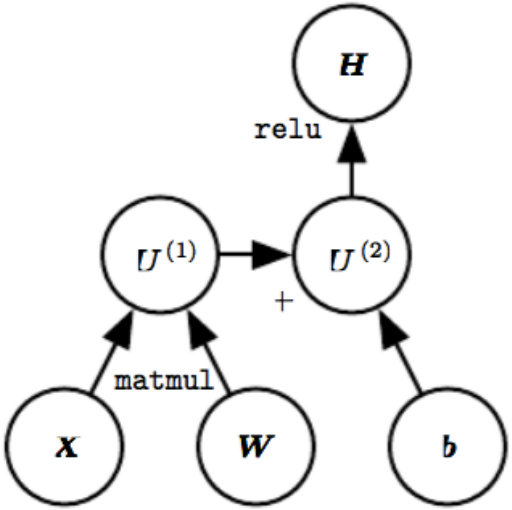
Computing graphs

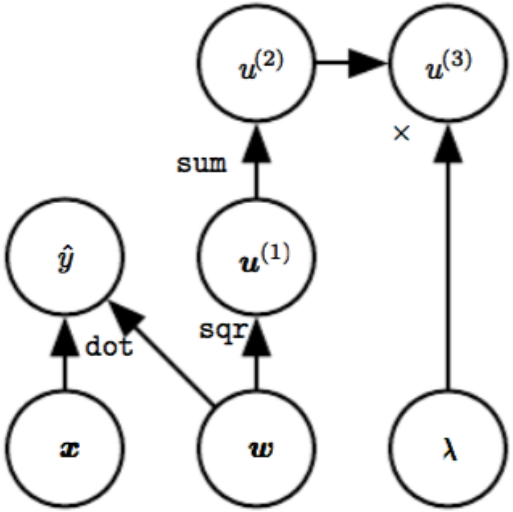


(a)



(b)





Stochastic gradient descent

Many cost functions and their gradients are of the form

$$C(\mathbf{X}, \theta) = \frac{1}{N}\sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta) \qquad \nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta)$$

with data samples $\mathbf{x}_i$. Computing such gradients is expensive, involving $\mathcal{O}(N)$ operations, where N may be millions.

Idea: rewrite gradient as an expectation $\mathbb{E}_\mathbf{x}$:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_\mathbf{x}[\nabla_\theta c(\mathbf{x}, \theta)]$$

Subdivide the N-sample average into M averages over n samples each. Call index sets B_m:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{n}\sum_{i\in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_\mathbf{x}[\nabla_\theta c(\mathbf{x}, \theta)]$$

Stochastic gradient descent

1. Initialize θ_0
2. For *epoch* $e = 1, \ldots, E$ or until converged:
   1. For *minibatch* $m = 1, \ldots, M$:
      1. Compute minibatch gradient $\mathbf{g}_m(\theta) = \frac{1}{n}\sum_{i\in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta)$
      2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_m(\theta_t)$

Stochastic gradient descent

SGD update with Momentum

Using the momentum parameter $0 \leq \gamma \leq 1$, we can implement SGD as:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t - \mathbf{v}_t$$

or, equivalently, written with parameter updates $\Delta\theta_t = \theta_t - \theta_{t-1}$:

$$\Delta\theta_{t+1} = \gamma \Delta\theta_t - \eta_t \mathbf{g}(\theta_t)$$

SGD with momentum is equivalent to numerically solving a dynamical equation of a particle with mass m and position θ moving in a viscous medium under the dimensionless potential $C(\theta)$:

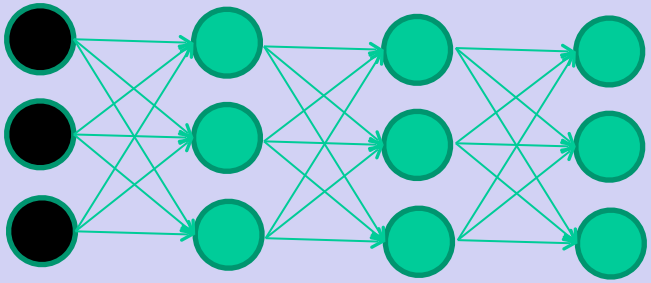
$$\underbrace{m\frac{d^2\theta}{dt^2}}_{\text{total force}} = \underbrace{-\nabla_\theta C(\theta)}_{\text{force from potential}} \underbrace{-\mu\frac{d\theta}{dt}}_{\text{drag force}}.$$

Langevin form: When approximating the gradient $\nabla_\theta C(\theta)$ with minibatches, it can be written as a deterministic part (gradient in the limit $M \to N$), plus a noise term depending on minibatch size. Then, Eq. (1) is a Langevin equation.

Second-order methods

- Desirable to adapt the learning rate by taking large steps in shallow, flat directions
- Second-order methods do this by computing the Hessian matrix, but this is computationally expensive.
- Alternative: methods that track not only the gradient but also its second moment.
- Examples:
  - AdaGrad (Duchi et al., 2011)
  - AdaDelta (Zeiler, 2012)
  - RMS-Prop (Tieleman and Hinton, 2012)
  - ADAM (Kingma and Ba, 2014).

What does a neural network learn?



Feature detectors

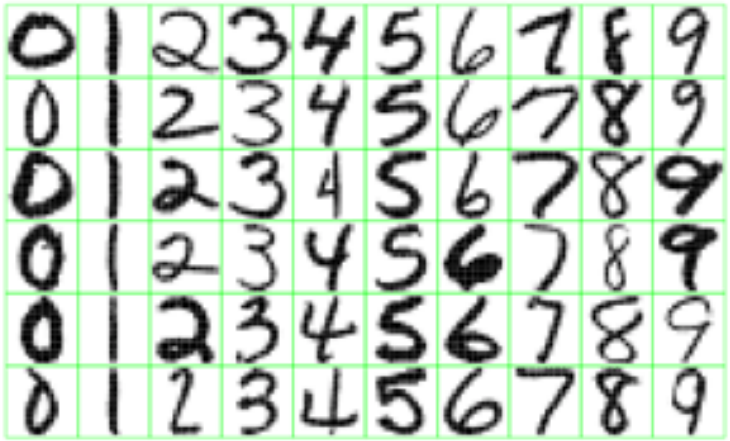
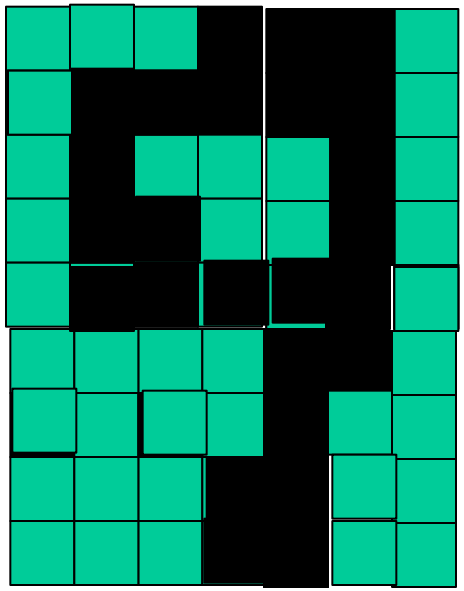
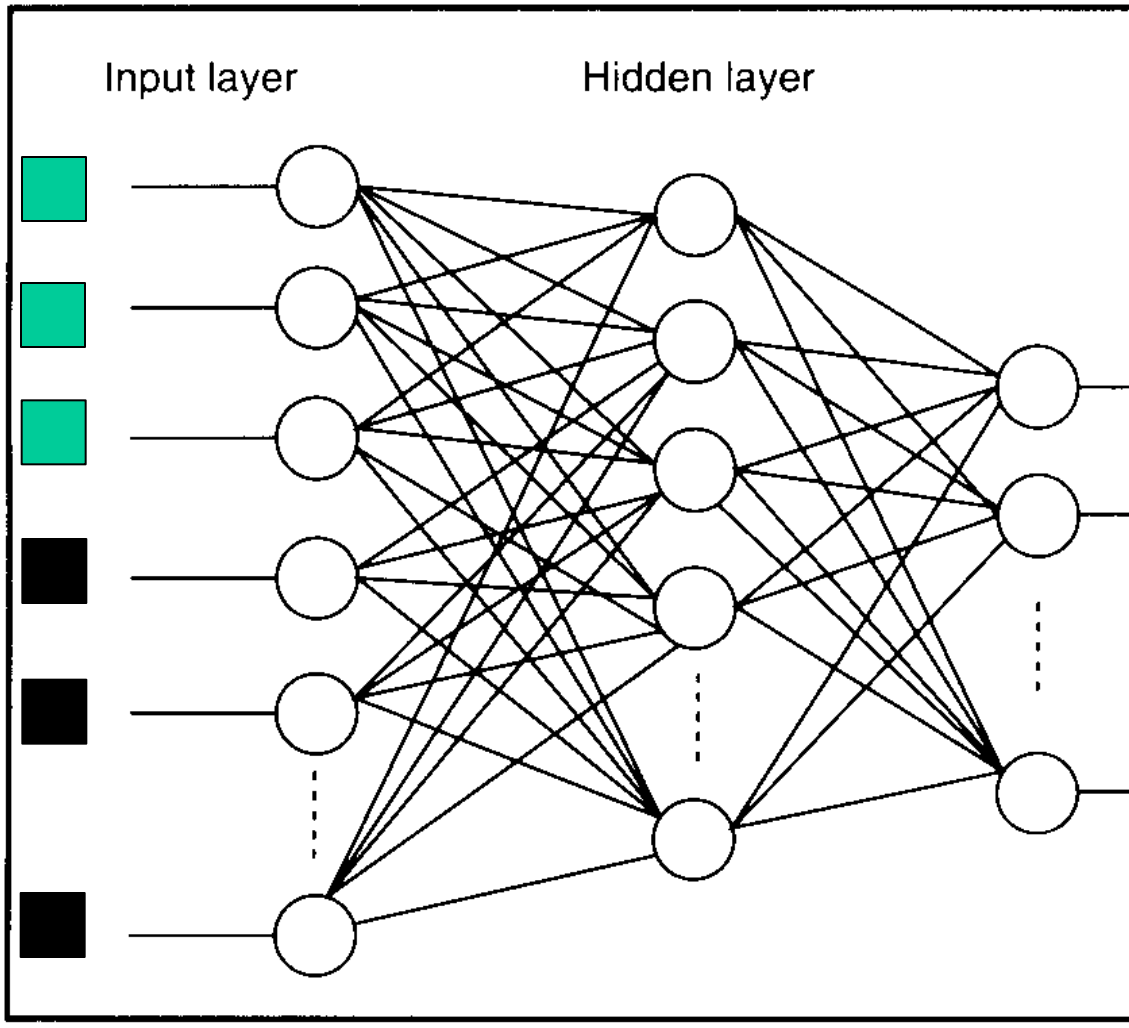


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





What is this unit doing?

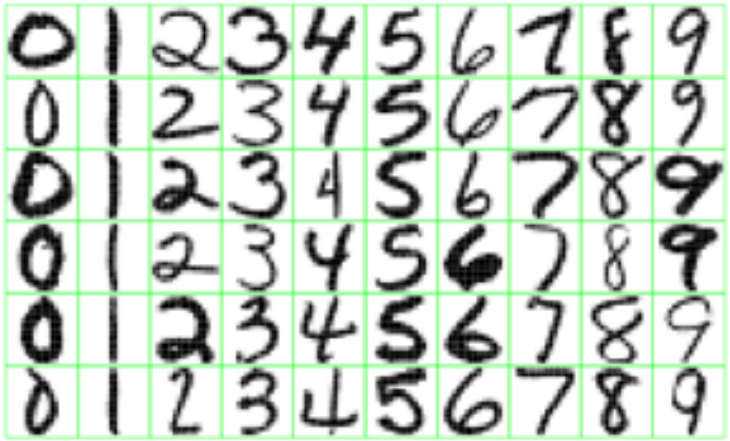
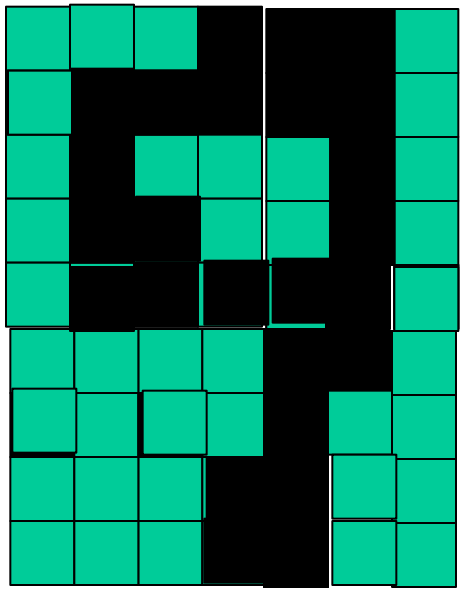
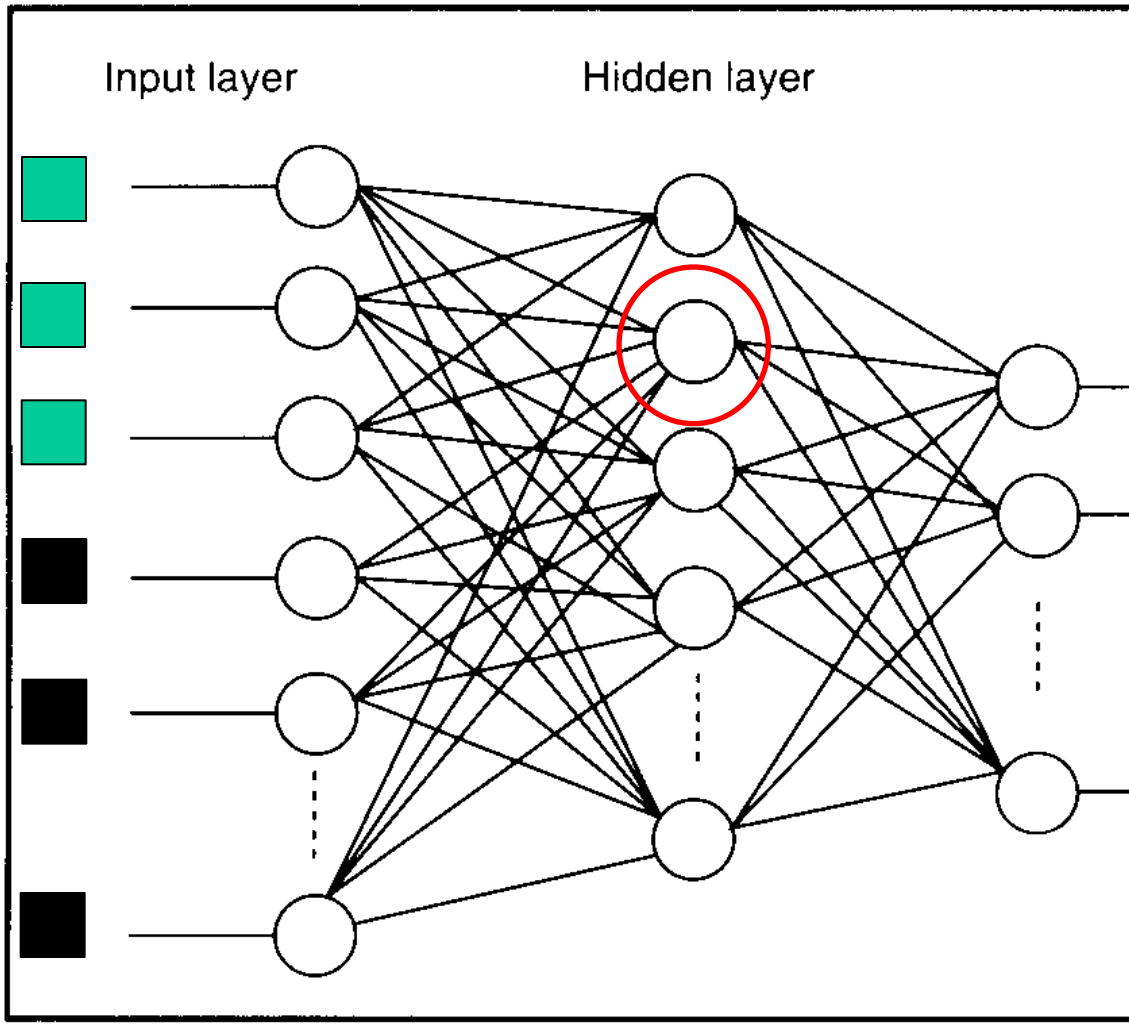
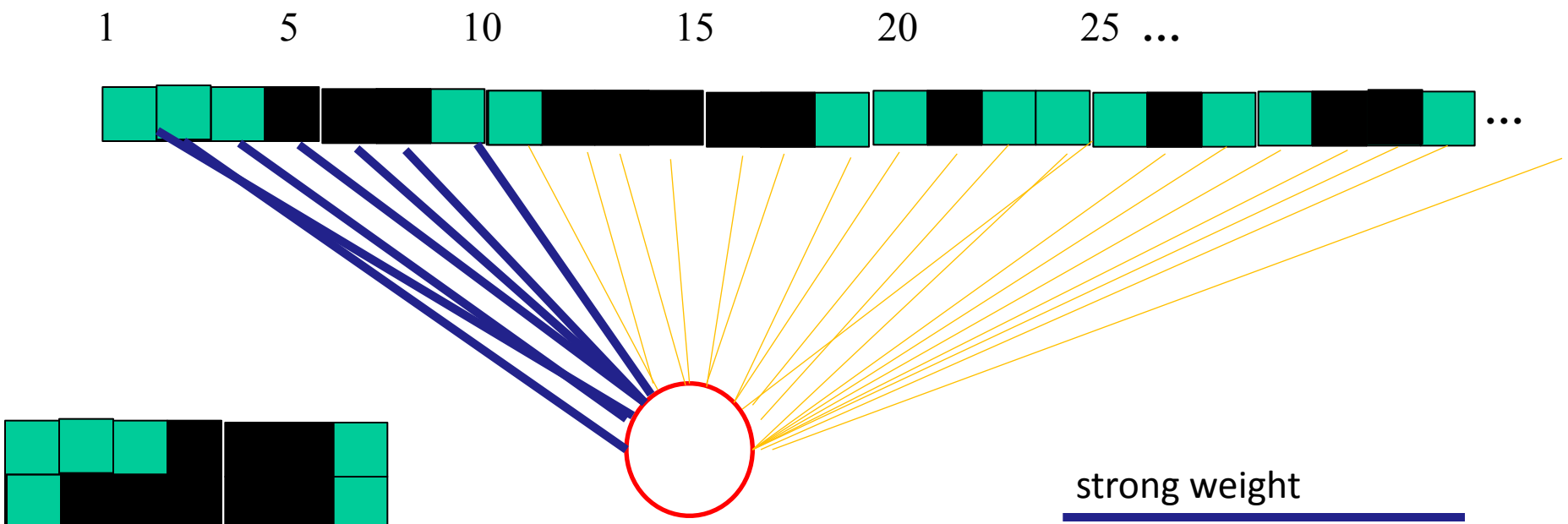


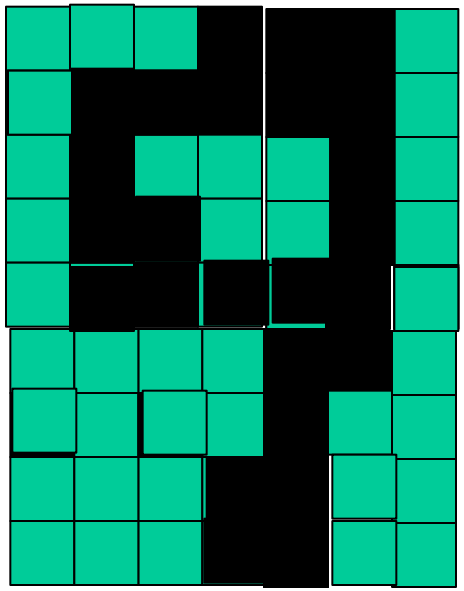
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





Hidden layers become self-organized feature detectors

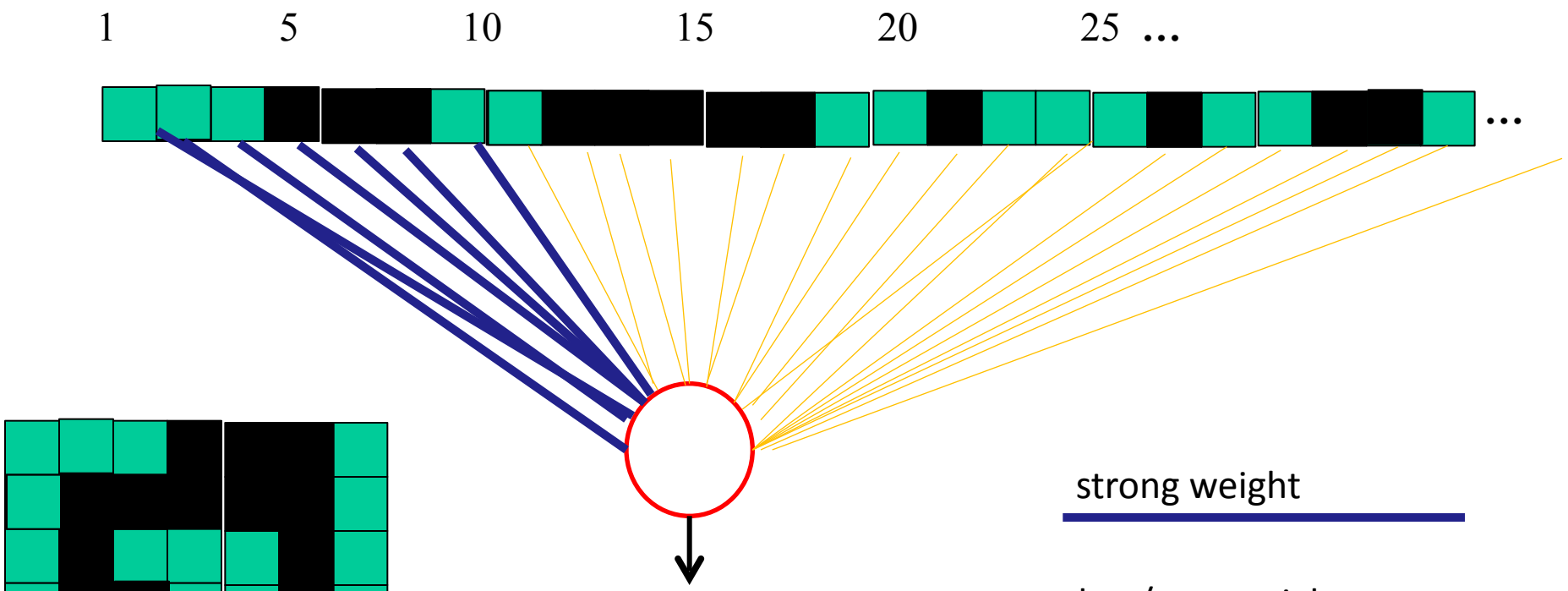


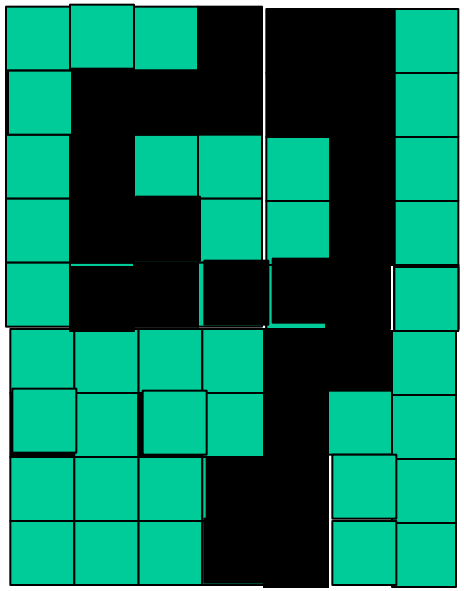


strong weight

low/zero weight

What does this unit detect?

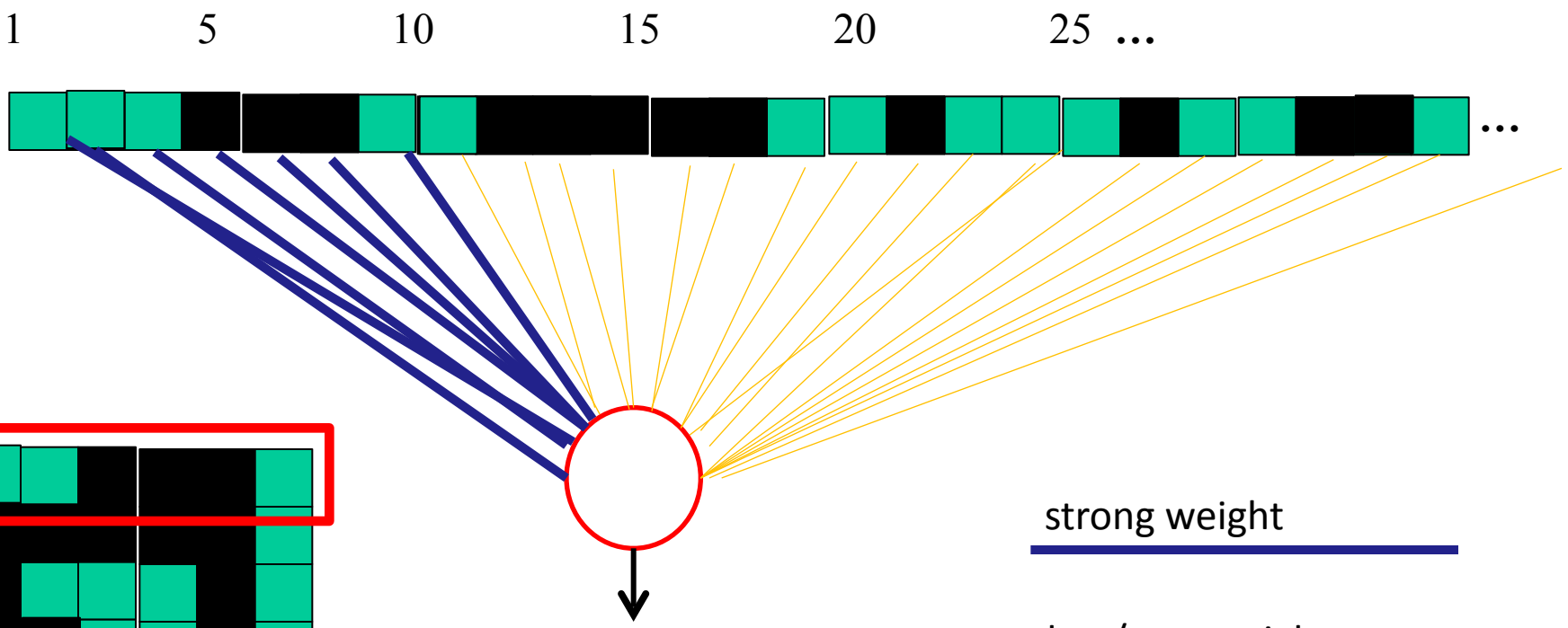


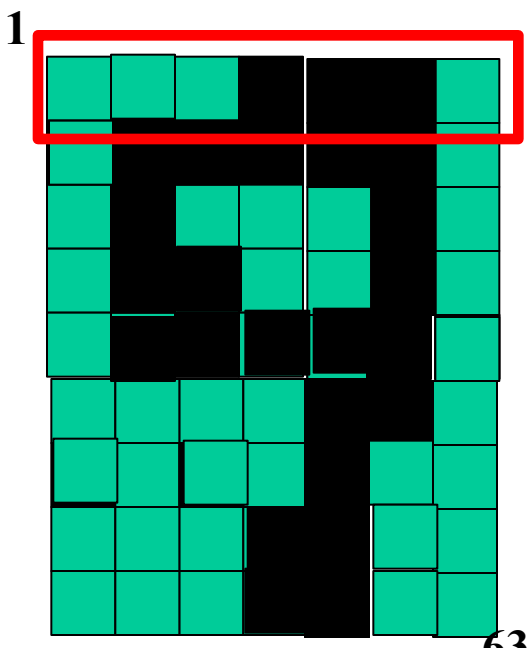


strong weight

low/zero weight

What does this unit detect?



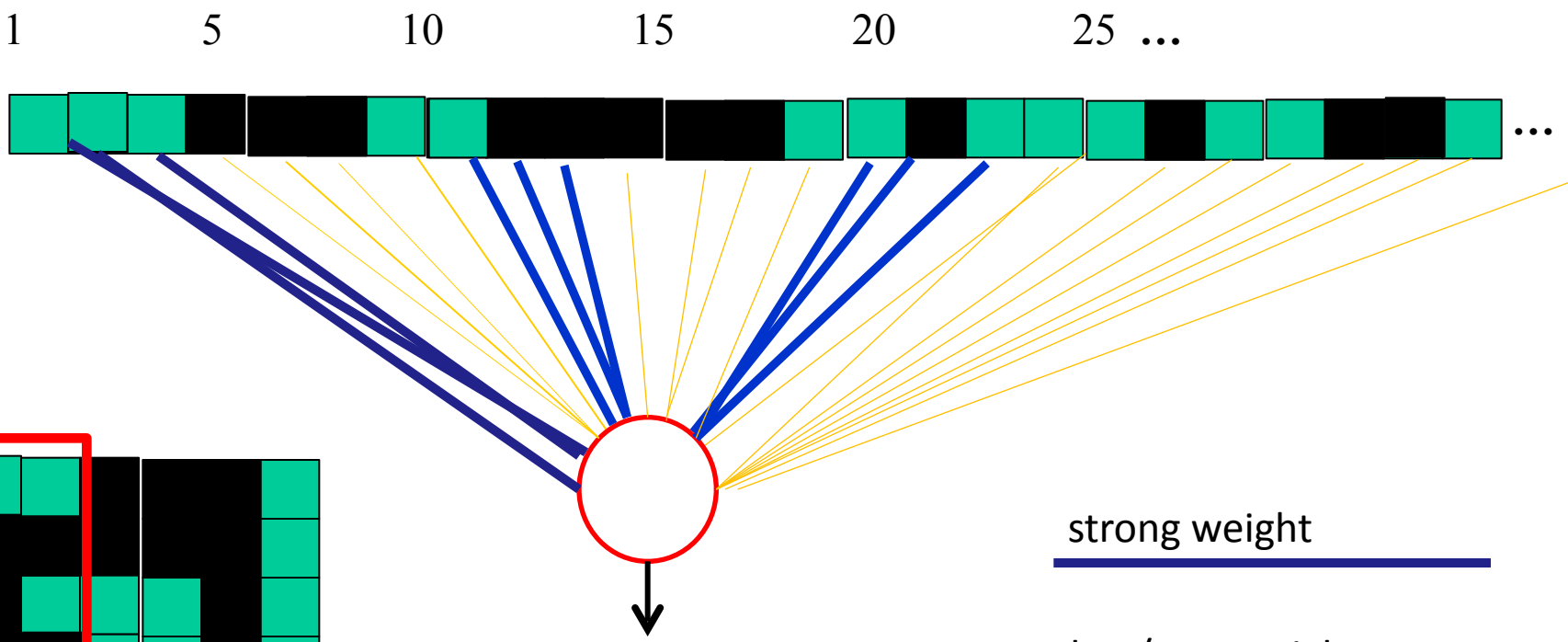


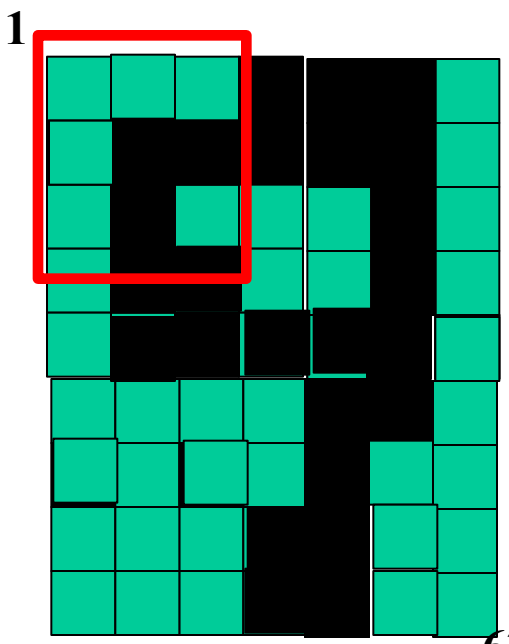
strong weight

low/zero weight

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

What does this unit detect?

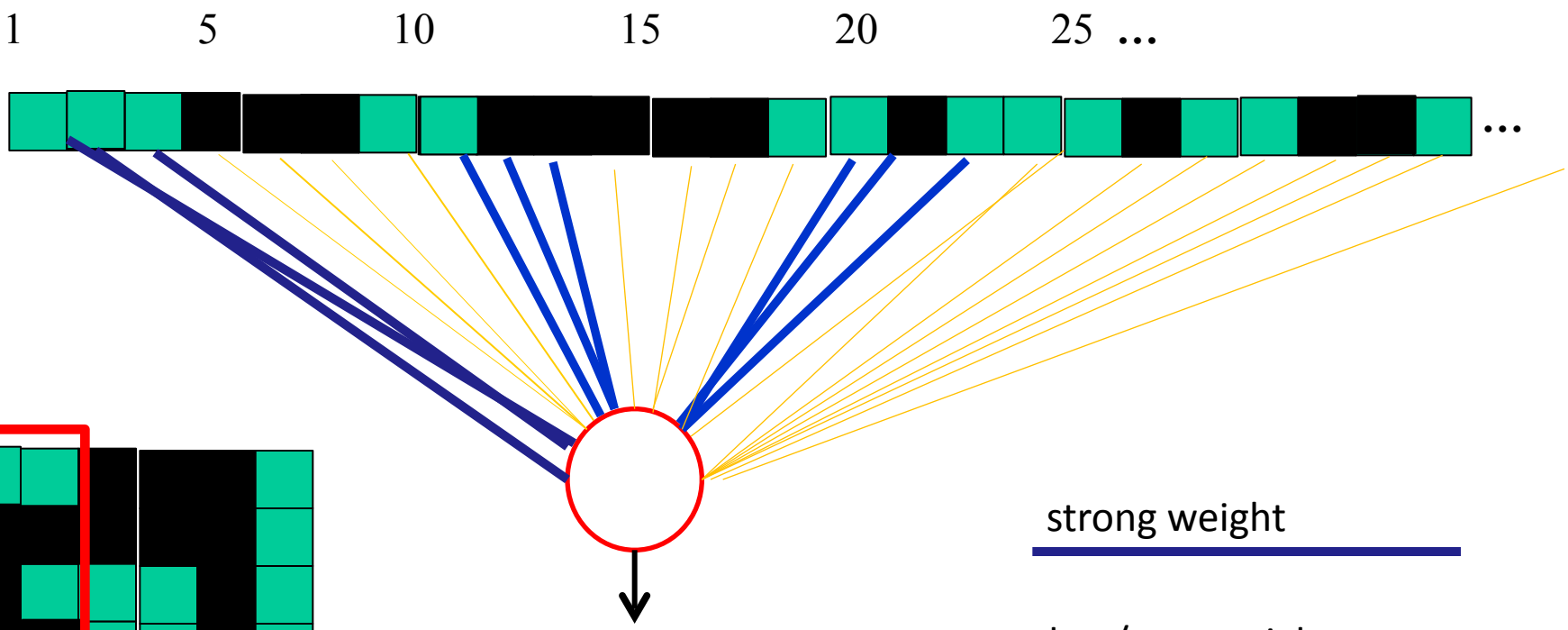


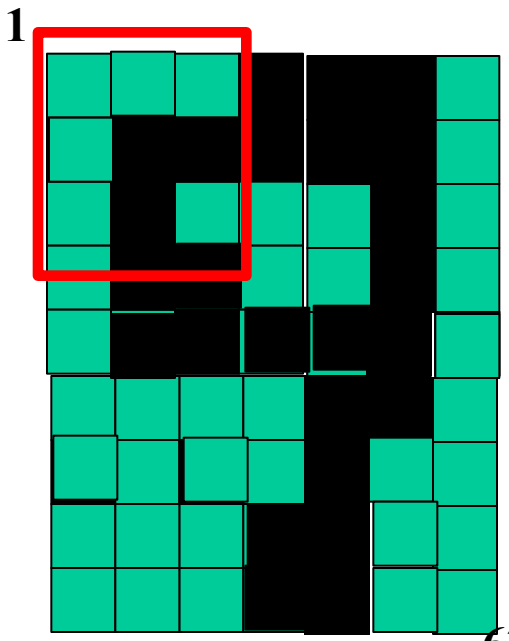


strong weight

low/zero weight

What does this unit detect?





strong weight

low/zero weight

Strong signal for a dark area in the top left corner

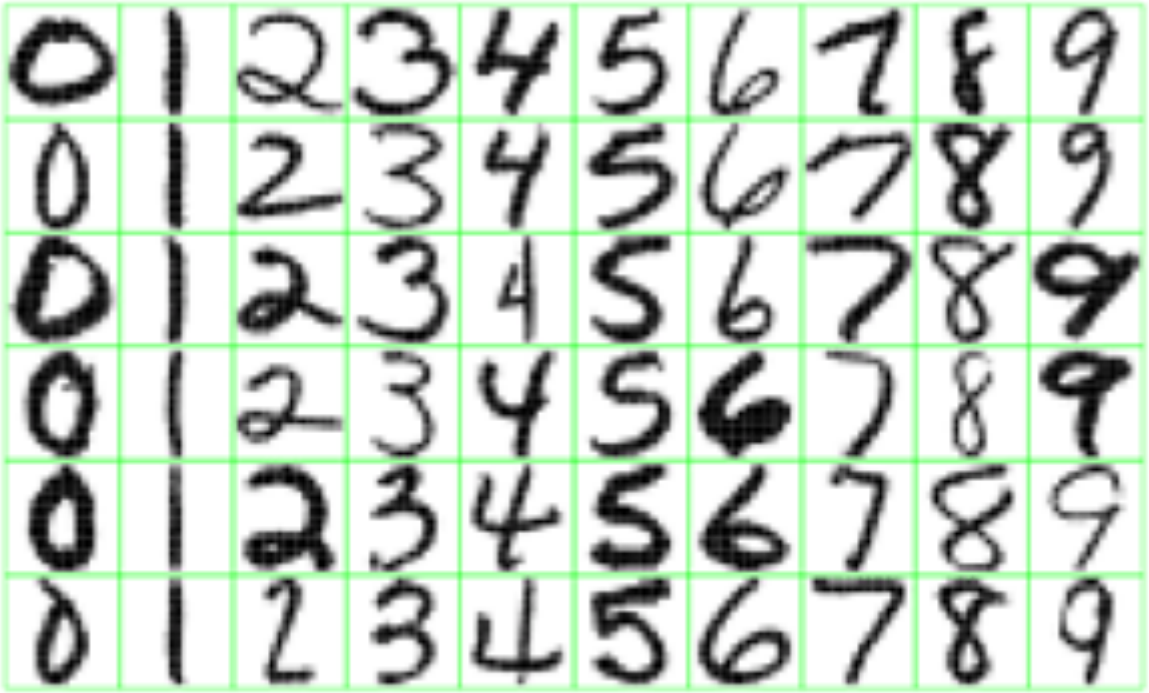


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

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

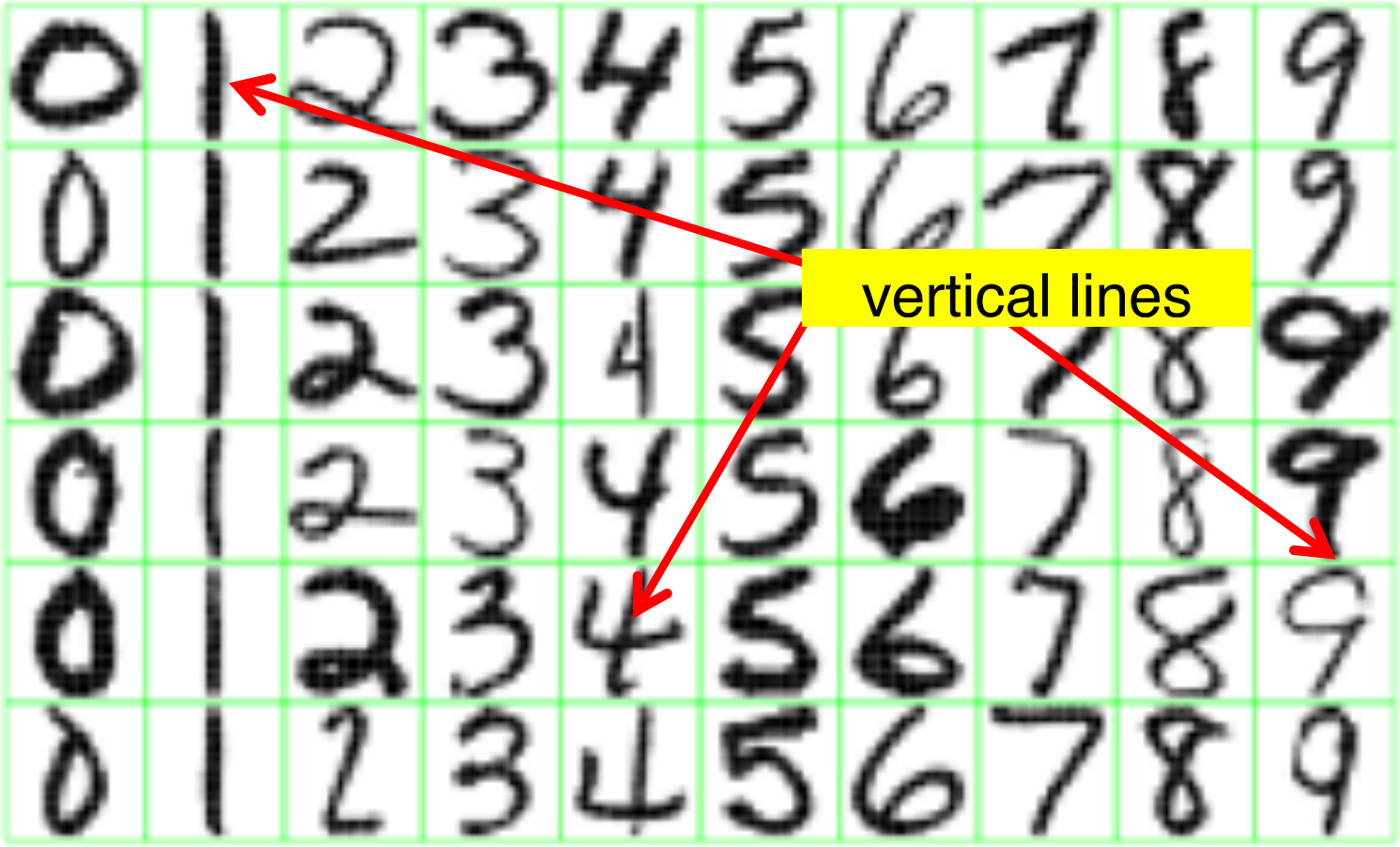


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

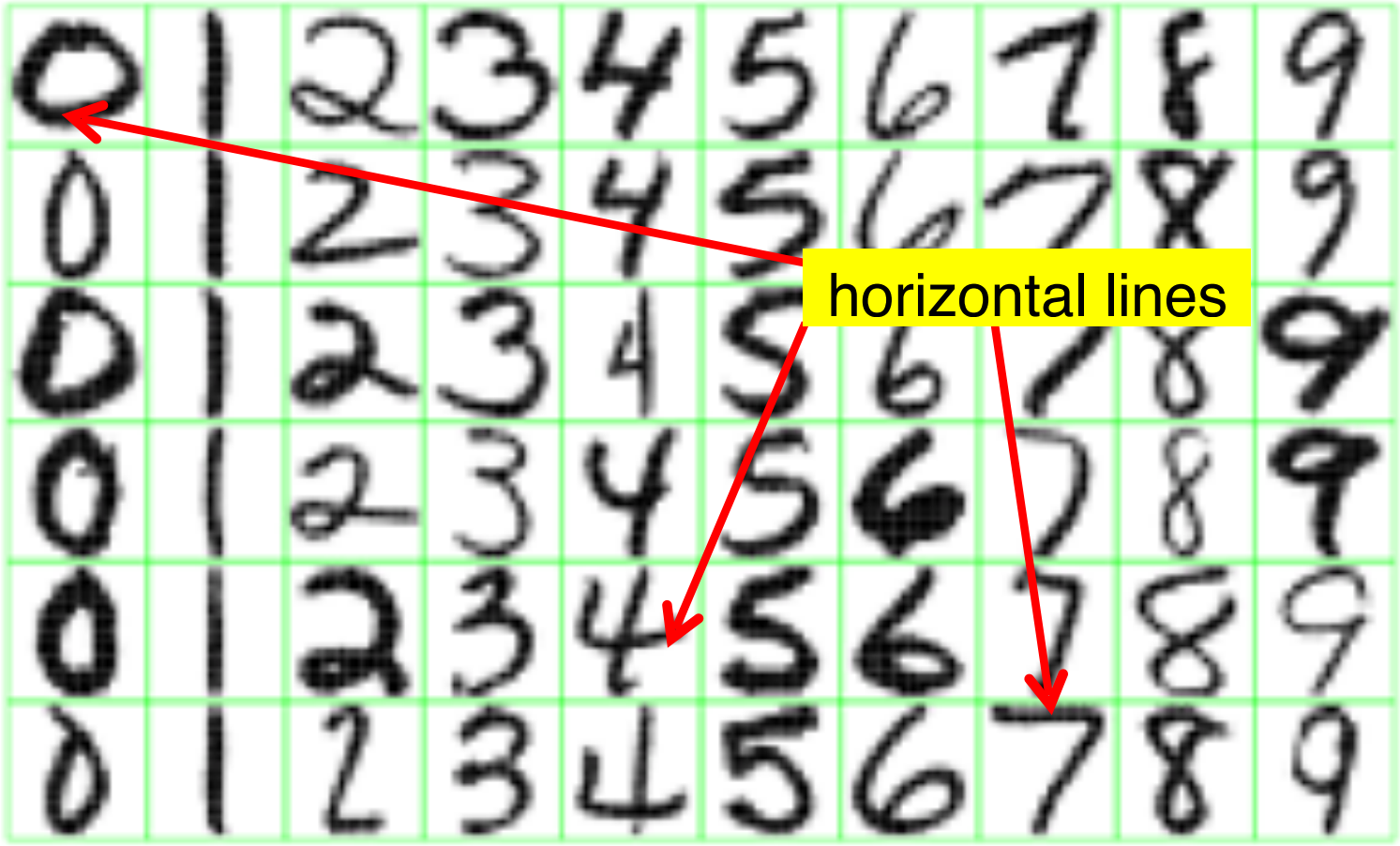
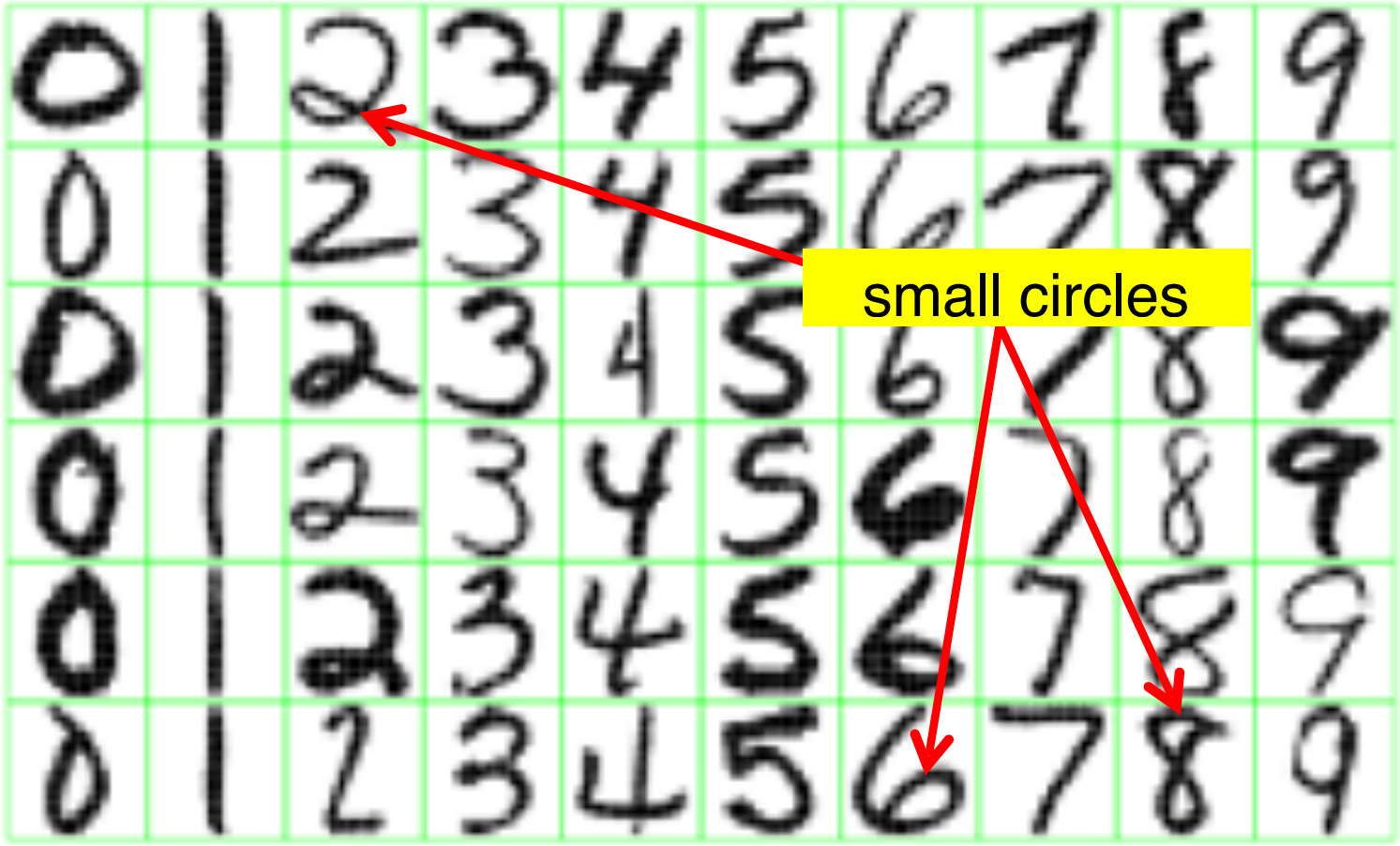
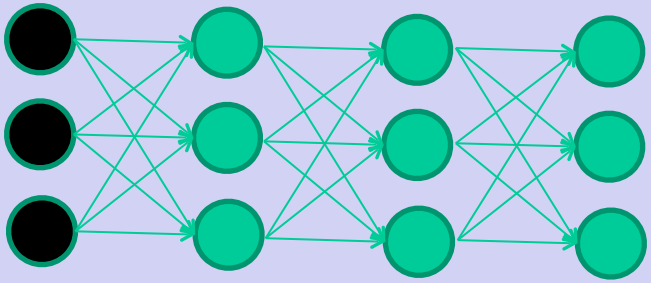


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

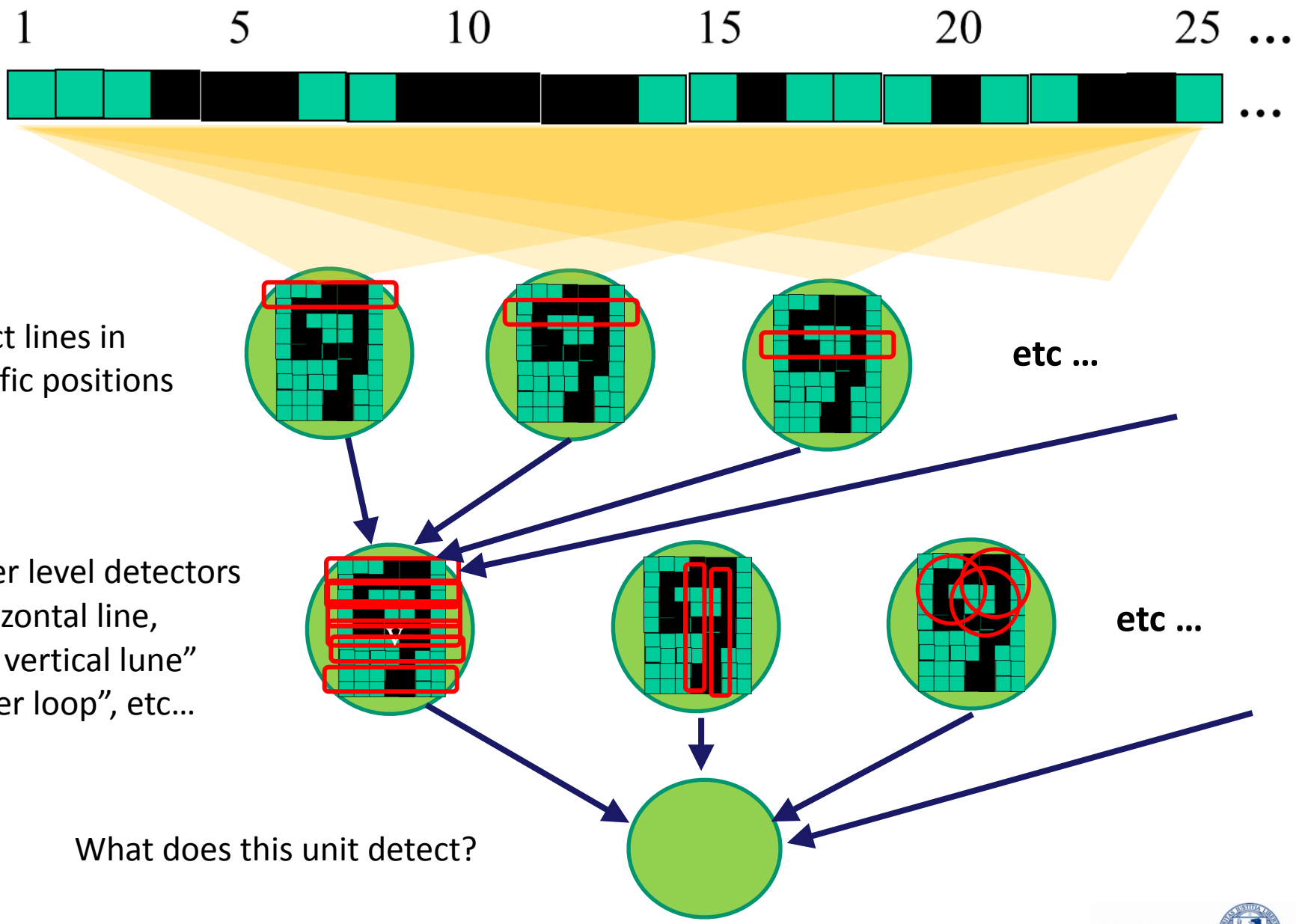


But what about position invariance?
Our example unit detectors were tied to specific parts of the image

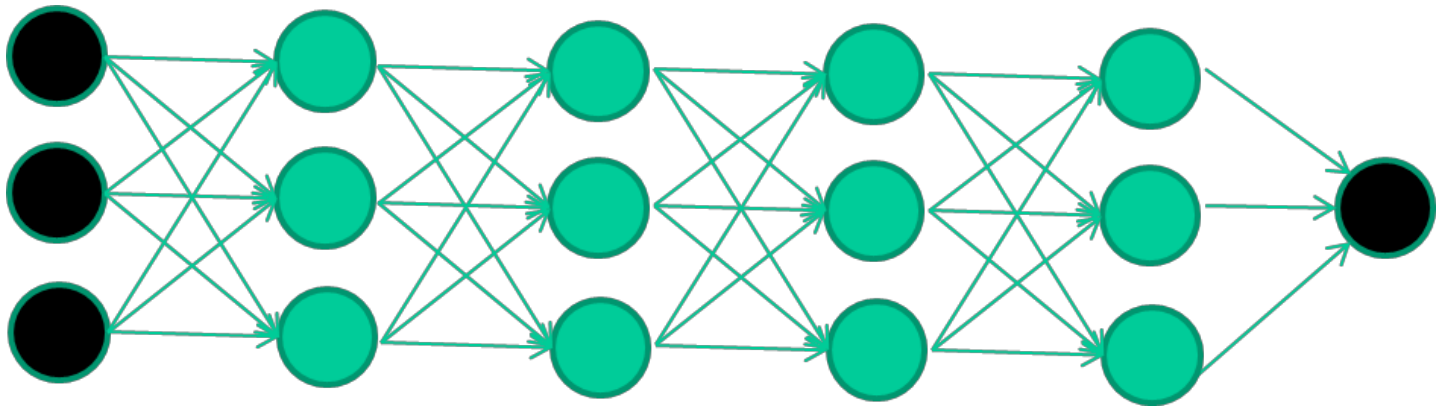
Deep Networks



Successive layers can detect higher-level features



But: until few years ago, 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*

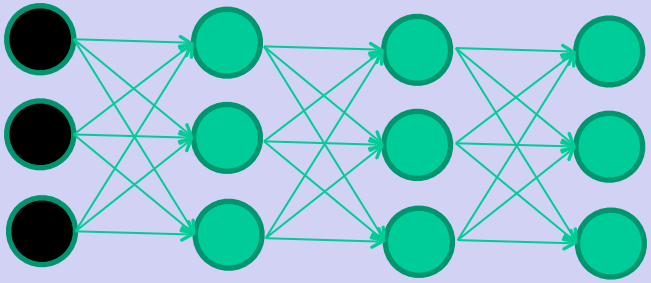
2019 Turing award

Key advances in deep learning

Until 2009, deep networks were rarely used as training them seemed too difficult. Key advances that enabled the training of deep networks include:

1. **Rectifiers**: Changing from activation functions such as logistic, tanh or arctan to rectifiers (ReLu, ELU, SoftPlus) avoids the vanishing gradient problem, makes the loss surface less "frustrated" and thus improves convergence.
2. **Stochastic Gradient Descent**: Changing from full gradient descent to stochastic gradient descent resulted in significant reduction of the computational cost to convergence (cheaper iterations and ability to escape flat local minima and saddle points)
3. **GPU implementations** of neural networks (Theano, TensorFlow), the resulting computational speedup, and the flexibility of automatic differentiation.

Applications to molecular systems



Physical Systems — example: O_2 molecule

1. **Invariance to reference frame**: $U(\mathbf{x})$ is invariant when translating or rotating the molecule. We can thus choose an arbitrary position and orientation of the molecule, e.g.:

$$\mathbf{x}_1 = (0,0,0) \qquad \mathbf{x}_2 = (d,0,0),$$

$\rightarrow$ energy becomes a function of the interatomic distance, $E(d)$.

2. **Energy conservation**: $U(\mathbf{x})$ and $\mathbf{f}(\mathbf{x})$ are related by

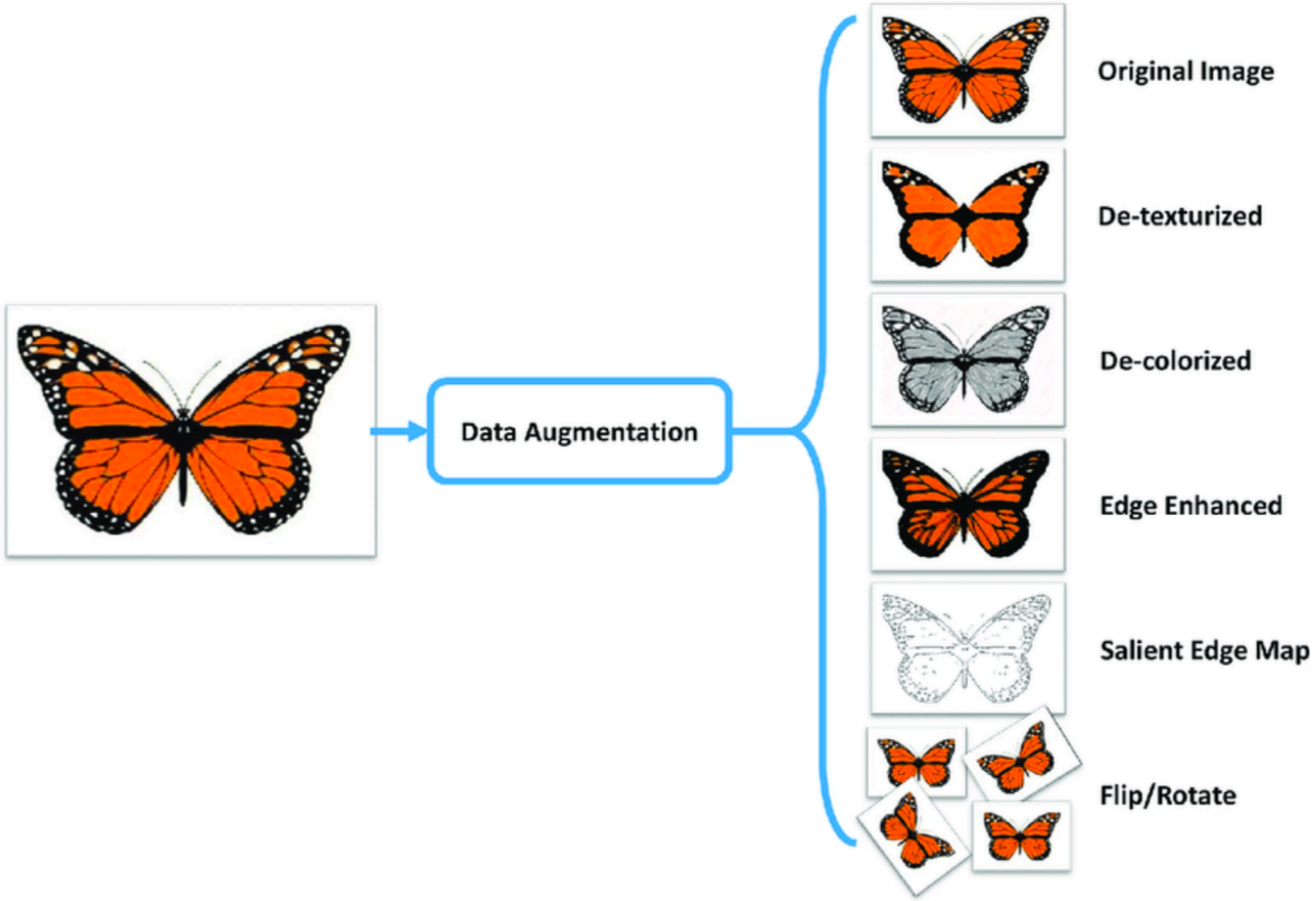
$$\mathbf{f}(\mathbf{x}) = -\nabla U(\mathbf{x}).$$

With the choice, we can compute the components of $\mathbf{f}(\mathbf{x})$ as:

$$\mathbf{f}_1 = \left(-\frac{\partial U(d)}{\partial d},0,0\right) \qquad \mathbf{f}_2 = \left(\frac{\partial U(d)}{\partial d},0,0\right)$$

3. **Universality of physical laws**: Same physical laws apply everywhere in the universe. Energy is unchanged if exchange the labels "1" and "2" (permutation invariance).

Data augmentation



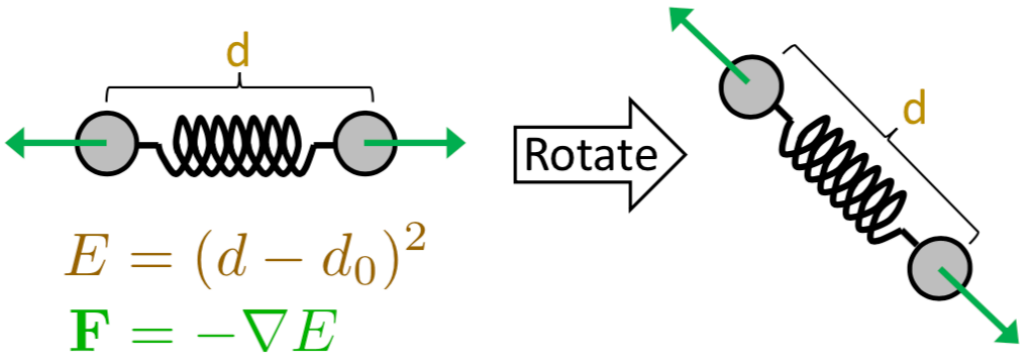
Physical Systems — example: O_2 molecule

- O_2 **example**: Energy is invariant wrt rotation, translation and permutation.
- **Equivariance**: A function is equivariant if it transforms the same way as its argument. For example, the force is defined by

$$\mathbf{f}(\mathbf{x}) = -\nabla U(\mathbf{x}).$$

If we rotate the molecule with $\mathbf{R}\cdot$, the force will rotate in the same way as the gradient is equivariant wrt rotation

$$\begin{array}{ccc} \mathbf{x} & \xrightarrow{\mathbf{R}\cdot} & \mathbf{Rx} \\ \downarrow \nabla U(\cdot) & & \downarrow \nabla U(\cdot) \\ \nabla U(\mathbf{x}) & \xrightarrow{\mathbf{R}\cdot} & \mathbf{R}\nabla U(\mathbf{x}) = \nabla U(\mathbf{Rx}) \end{array}$$



$$E = (d - d_0)^2$$
$$\mathbf{F} = -\nabla E$$

Building Physics into the ML model

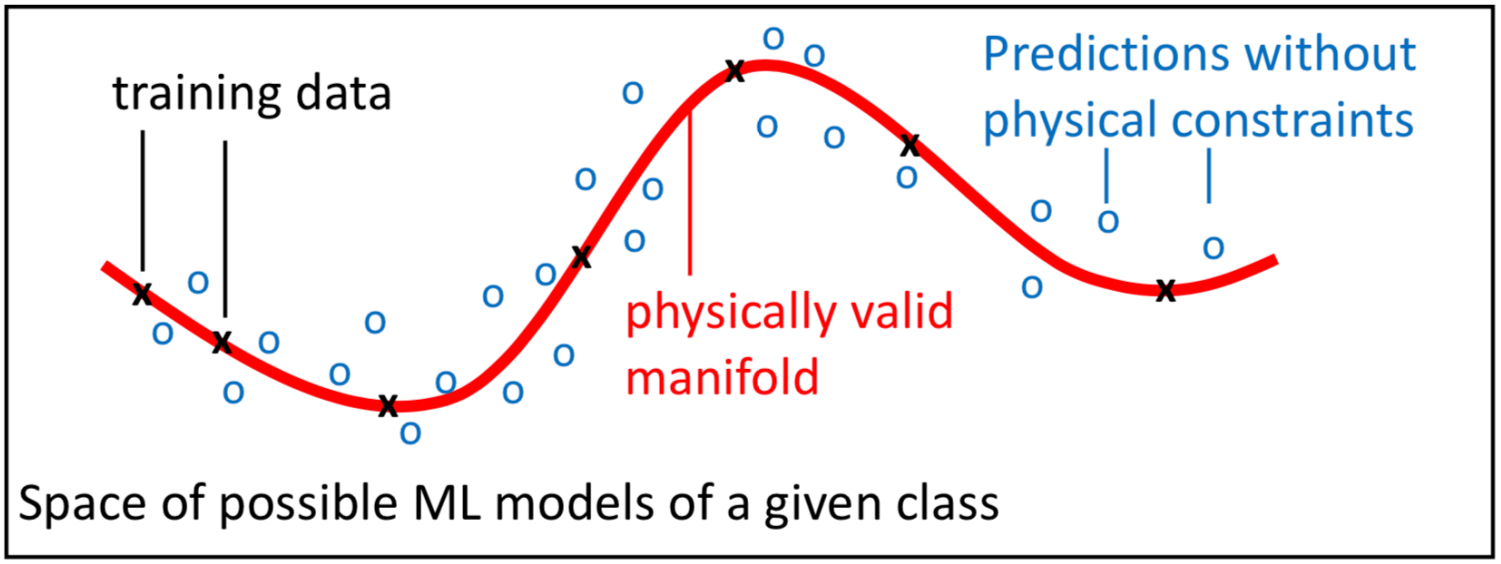
- Incorporating physical symmetries into ML model avoid learning them "by heart" via data augmentation.
- Reduces the dimensionality of the problem. In the O_2 example above, we have reduced the learning of:

$$U(\mathbf{x}) : \mathbb{R}^6 \rightarrow \mathbb{R}, \mathbf{f}(\mathbf{x}) : \mathbb{R}^6 \rightarrow \mathbb{R}^6$$

to

$$U(d) : \mathbb{R} \rightarrow \mathbb{R}, \mathbf{f}(d) = -\nabla_{\mathbf{x}} U(d).$$

- We only operate on the manifold of physically meaningful solutions.



Translation, rotation, energy conservation

- $\mathbf{R}\cdot$ rotates each atom with random 3D rotation matrix, $\mathbf{T}$ translates each atom with a translation vector.
- Potential or free **energy** of a molecule without external field is **invariant** to roto-translation:

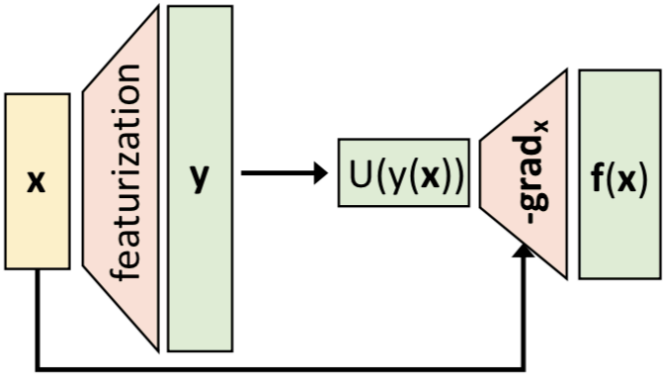
$$U(\mathbf{Rx}+\mathbf{T}) = U(\mathbf{x}).$$

$\rightarrow$ easily achieved by transforming the $\mathbf{x}$ into rototranslationally invariant features $\mathbf{y}$ (e.g., distances, angles).

- **Force** is **equivariant** to rotation, but invariant to translation

$$-\nabla U(\mathbf{Rx}+\mathbf{T}) = -\mathbf{R}\nabla U(\mathbf{x})$$

achieved by computing force explicitly in network (SchNet, CGnet)



Permutation invariance

- **Energy/force** a molecule are **invariant/equivariant** with permutation of equivalent particles:

$$U(\mathbf{P}\mathbf{x}) = U(\mathbf{x})$$
$$-\nabla U(\mathbf{P}\mathbf{x}) = -\mathbf{P}\nabla U(\mathbf{x})$$

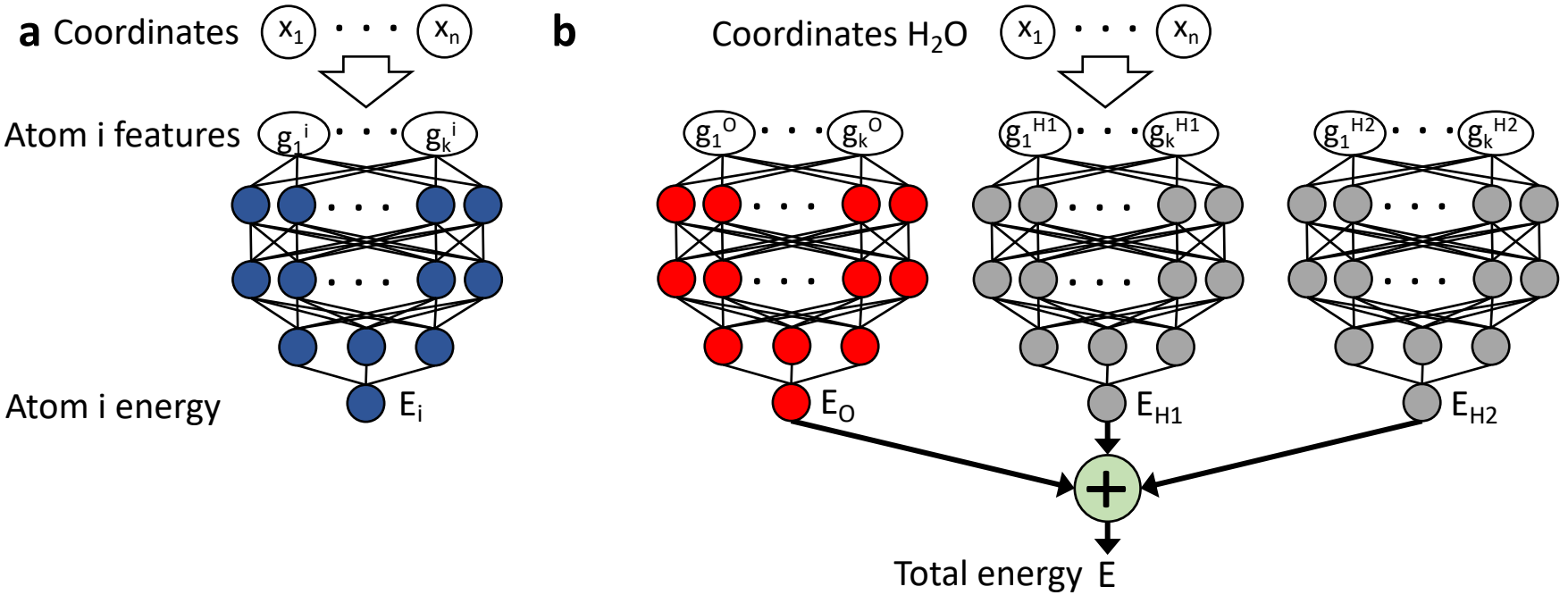
- Number of permutations increases **exponentially** with the number of identical particles $\rightarrow$ learning permutation invariance by **data augmentation is hopeless**.
- **DeepSets**[1], Theorem 2: any permutation symmetric function $f(r)$ can be represented in the form $\rho(\sum_i \phi(\mathbf{r}_i))$, with functions ρ, ϕ.
- **Common choice**: compute potential energy as a sum

$$E(\mathbf{x}) = \sum_i E_i(\mathbf{x}), \tag{4}$$

E_i is the contribution of the energy by the ith atom in its chemical environment.

[1] Zaheer et al, *NIPS* 2017

Learning to represent energy functions

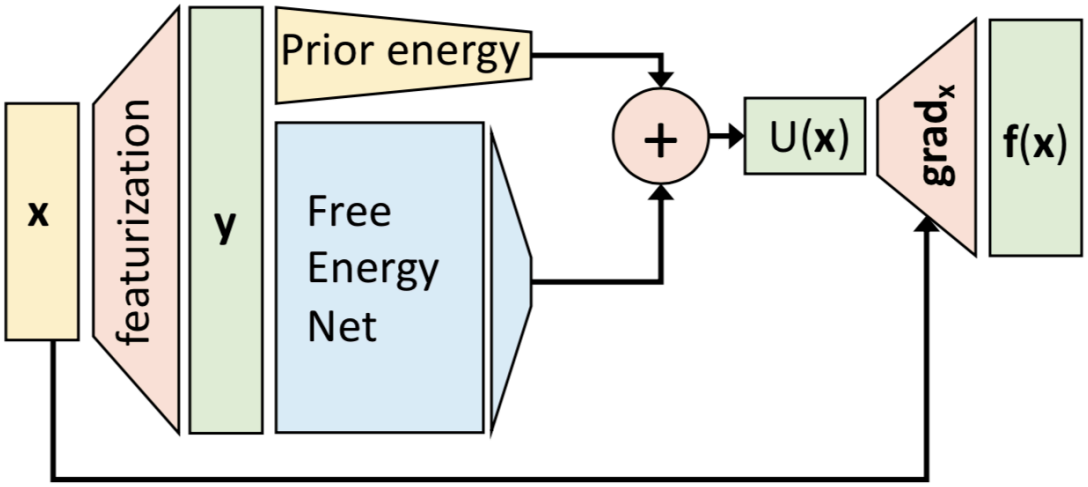


- Rototranslational and permutation invariances; parameter sharing.
- **a**) Network for computing the atomic energy of a single atom i. The system coordinates are mapped to rototranslationally-invariant features describing the chemical environment of atom i.
- **b**) Molecular system, e.g., H_2O, is composed by employing a network copy for each atom. Parameters are shared between networks for same elements.
- Total energy equals sum of atomic energies $\rightarrow$ permutation

Behler and Parrinello, PRL 98, 146401 (2007)

Learning coarse-grained free energy functions

CGnet[9]

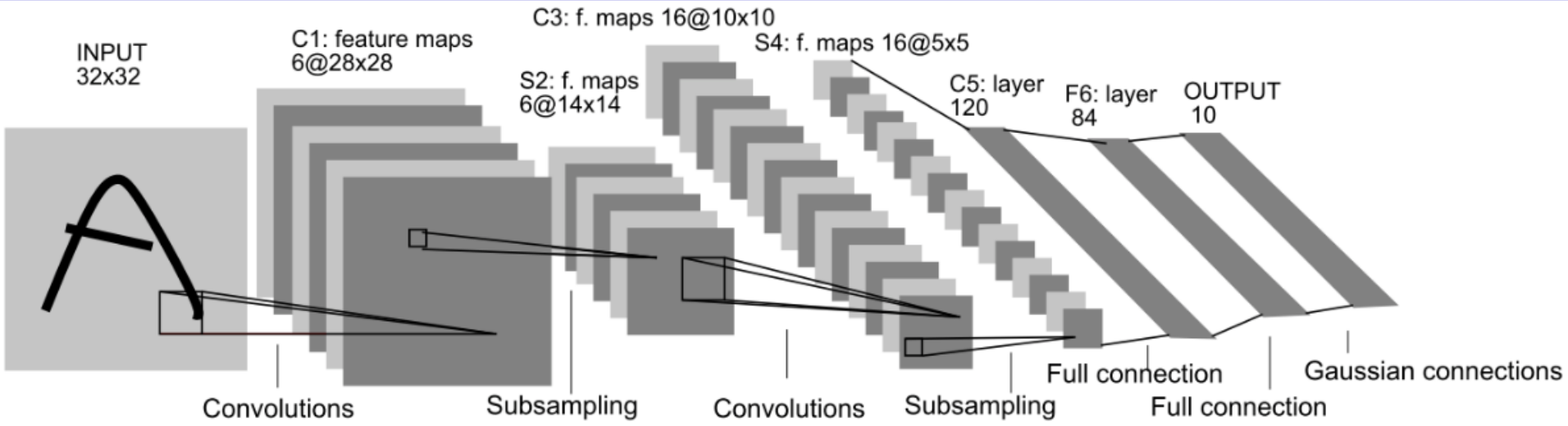


- Rototranslationally invariant features $\mathbf{y} \rightarrow$ invariant energy $U(\mathbf{x})$.
- Energy conserving: $\mathbf{f}(\mathbf{x}) = -\nabla_{\mathbf{x}} U(\mathbf{x}) \rightarrow$ equivariant force $\mathbf{f}(\mathbf{x})$.
- Prior energy: defines correct asympotic behavior where $U(\mathbf{x}) \rightarrow \infty$.
- **No** permutation invariance.
- **No** parameter sharing $\rightarrow$ not scalable, not transferable.

---

[9] Wang, Olsson, Wehmeyer, Pérez, Charron, De Fabritiis, Noé, Clementi: *ACS Cent. Sci.* 5, 755-767 (2019)

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

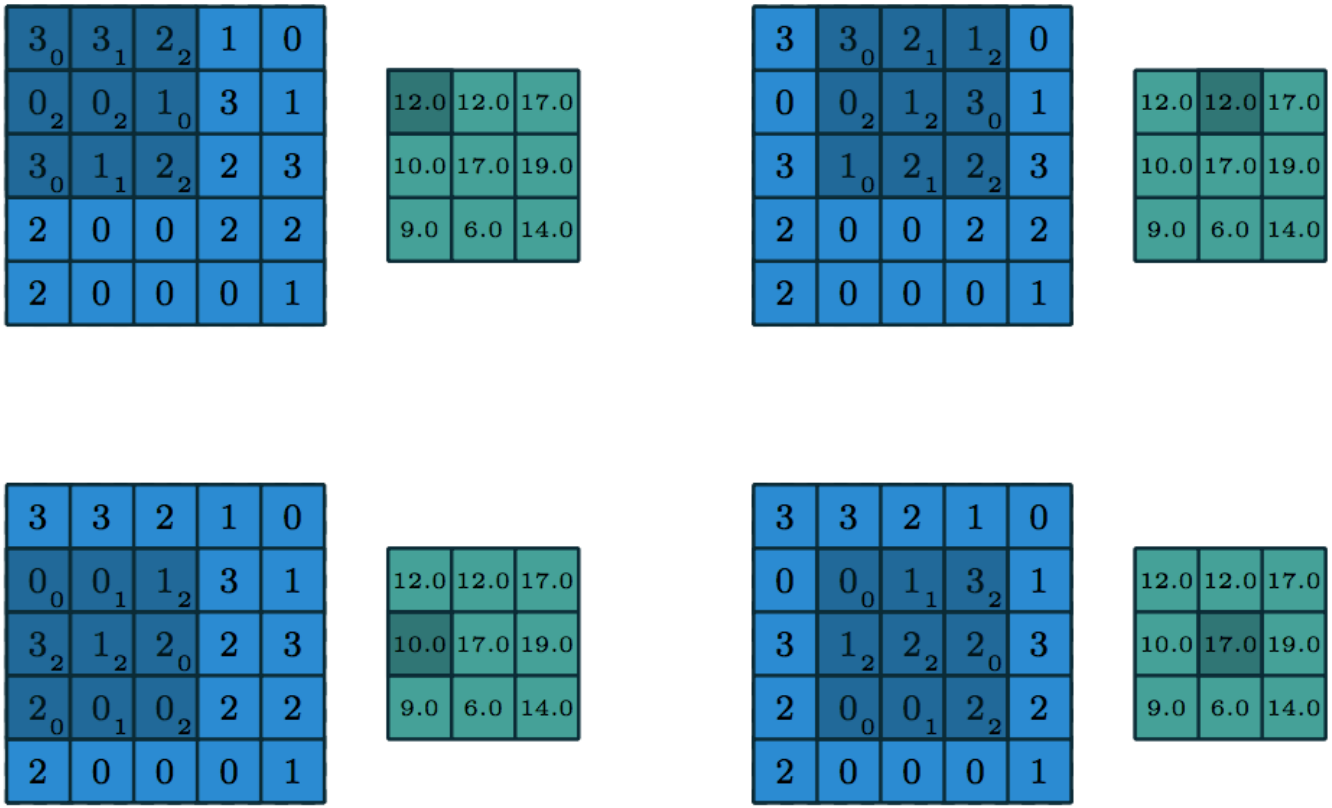
Discrete convolution:

$$y_i = (\mathbf{x} * \mathbf{w})_i = \sum_j x_j w_{i-j} = \sum_j w_j x_{i-j}$$

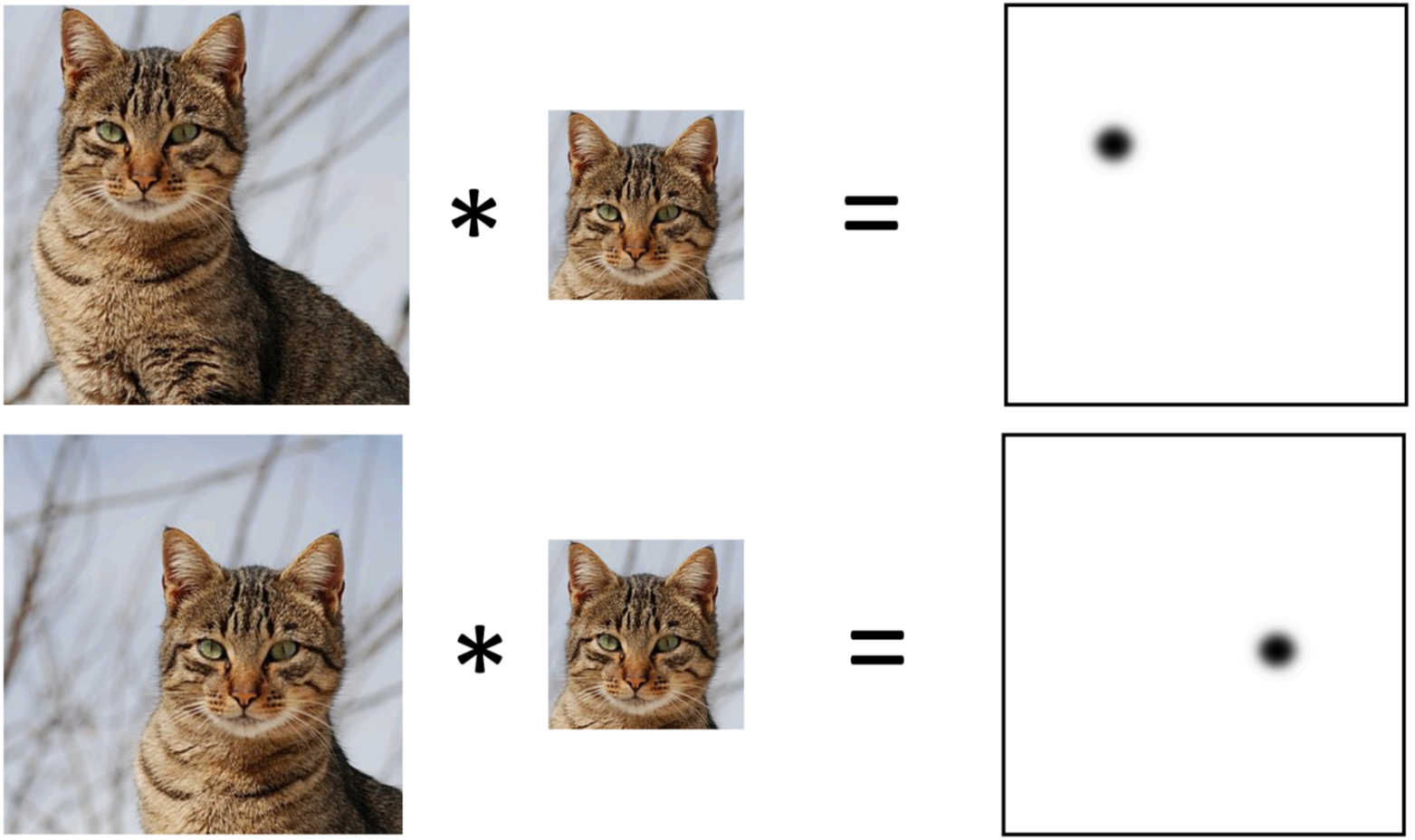
Convolutional Kernel / Filter

0	1	2
2	2	0
0	1	2

Apply convolutions



Convolutions are translation-equivariant

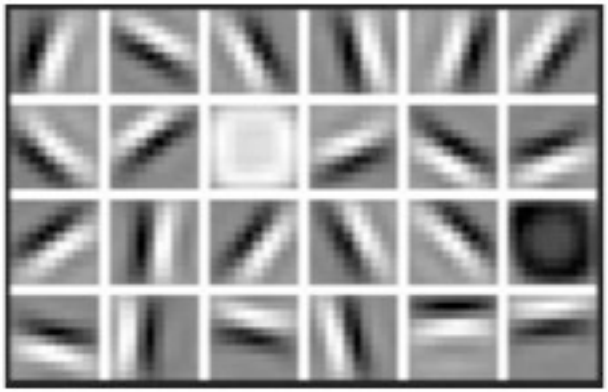


- Convolutions combine **parameter sharing and equivariance**.
- Standard convolutions are **translation-equivariant**.
- **Rotation-equivariant** convolutions exist (spherical CNNs, SchNet).

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

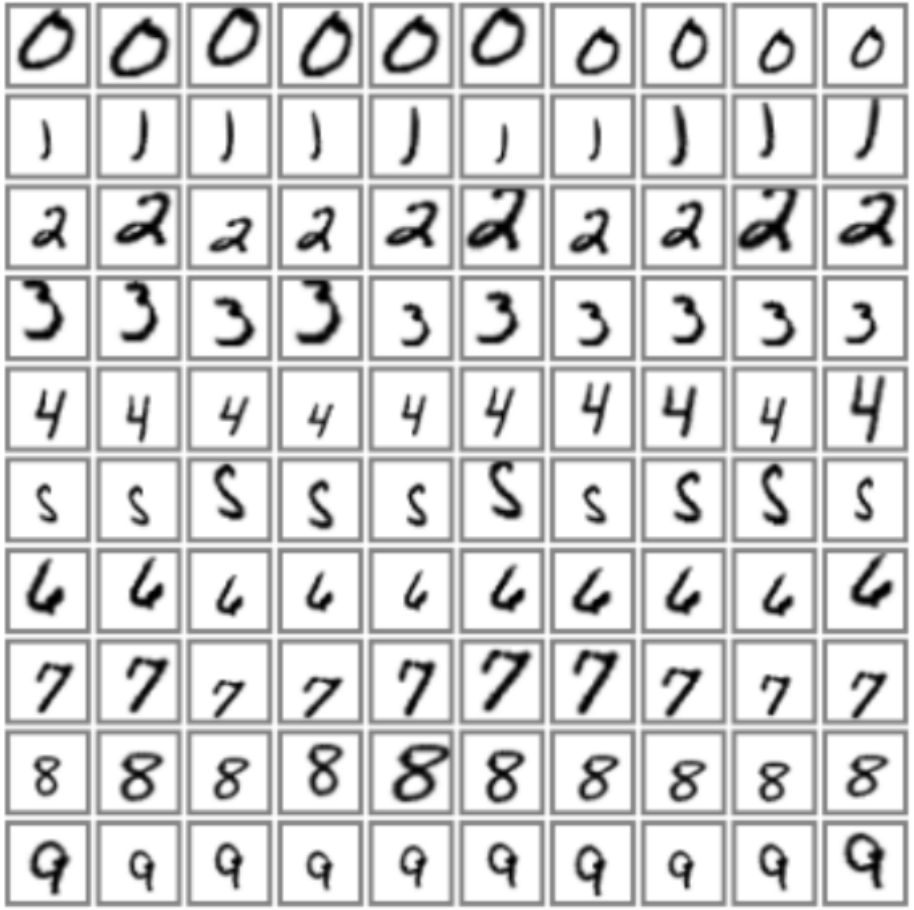
Second Layer Representation

Third Layer Representation

MNIST dataset

60,000 original datasets

Test error: 0.95%



540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

MNIST misclassified examples

Convolution is a linear operation

Convolving $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{w} \in \mathbb{R}^m$, $m \leq n$ can be written as linear operation

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

with $\mathbf{W} \in \mathbb{R}^{n-m+1 \times n}$ being a **Toeplitz matrix**:

$$\mathbf{W} = \begin{pmatrix} w_1 & \cdots & w_m & & \\ & \ddots & & \ddots & \\ & & w_1 & \cdots & w_m \end{pmatrix}$$

Convolutional Networks:

↑

Pooling Layer — Compresses image

↑

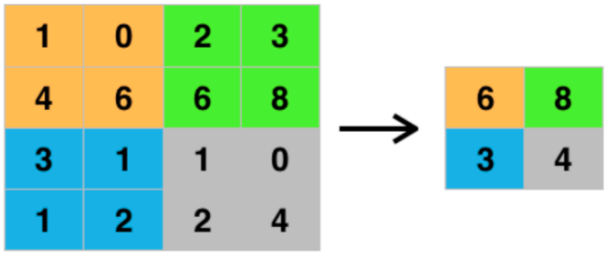
Detector Layer — Nonlinear transform

↑

Convolution Layer — Affine transform

↑

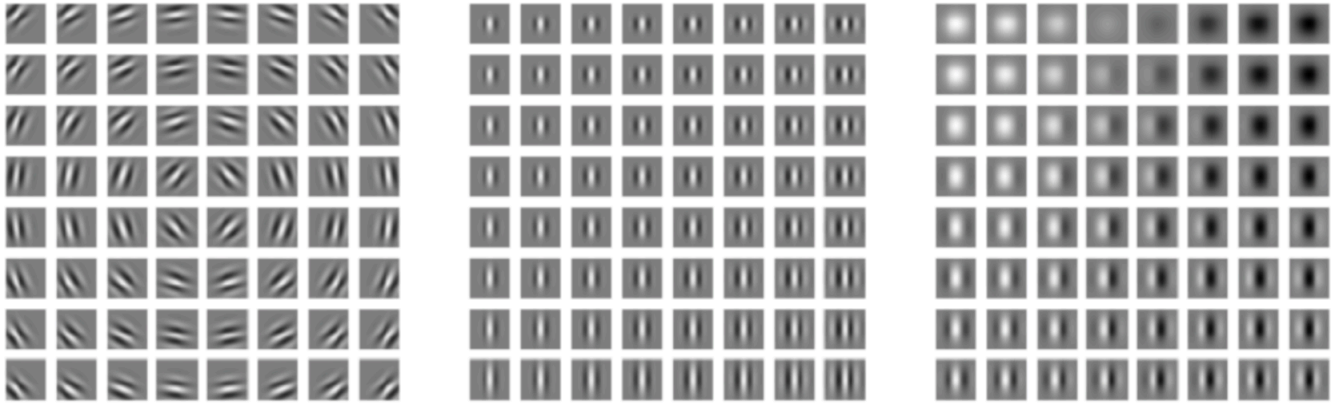
max pooling



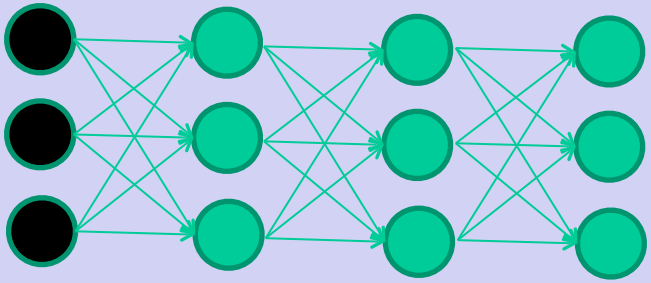
Motivation from neuroscience

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.

- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina.
    $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field.
    $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.



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 $\mathbf{X}^\top\mathbf{X}$

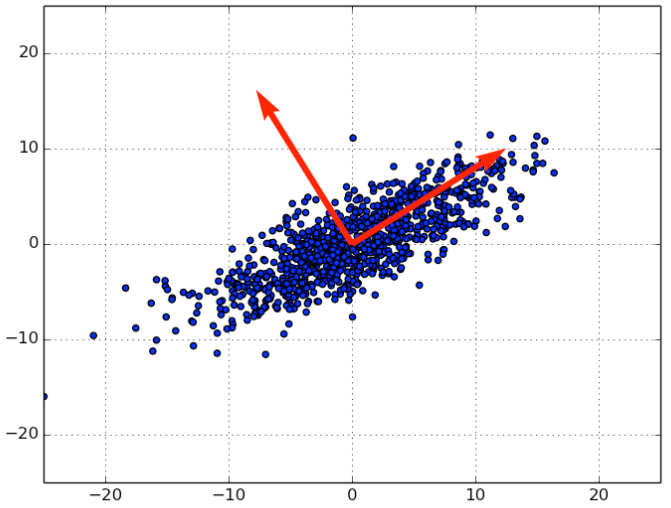
2. Solve the Eigenvalue problem:

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

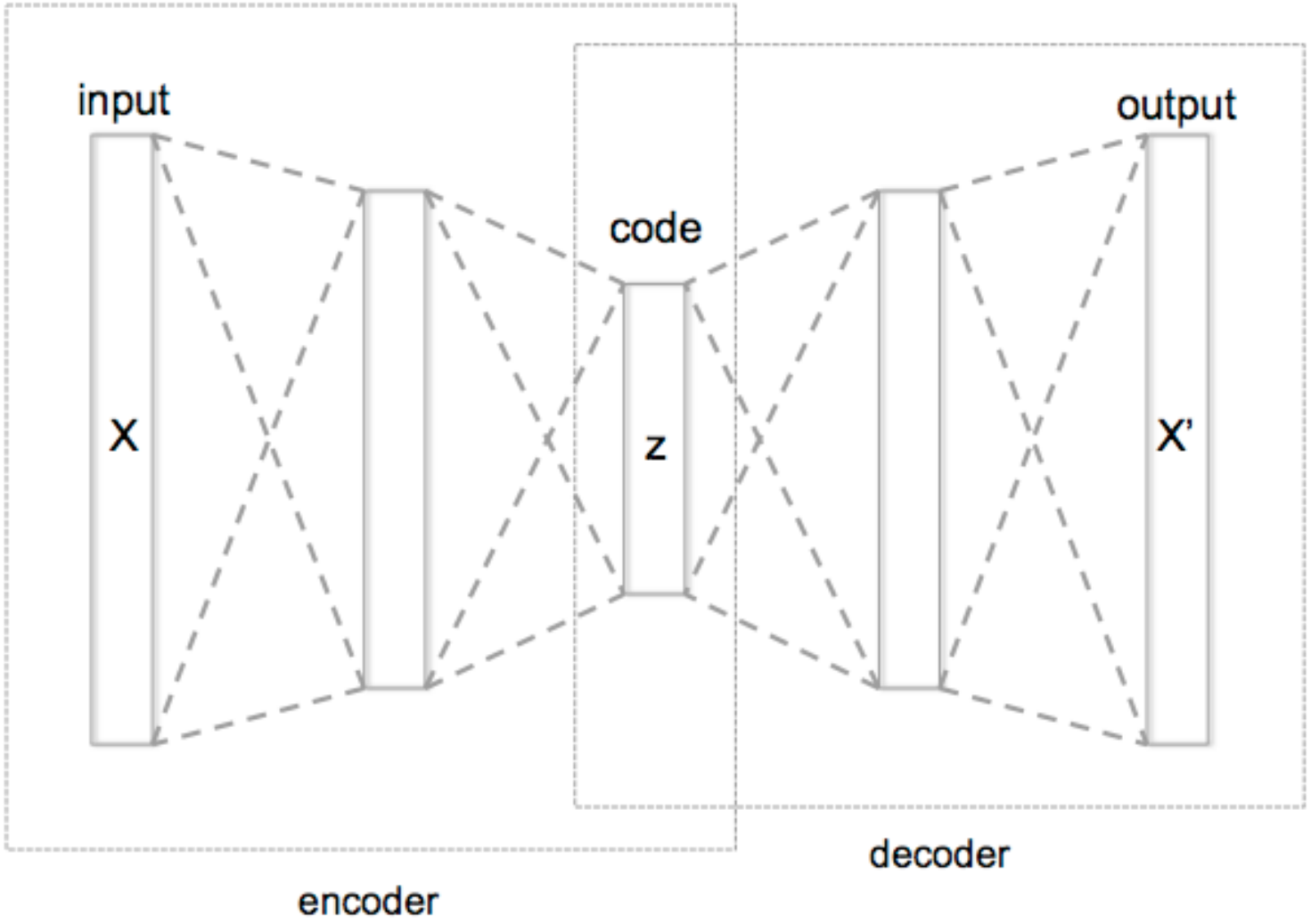
with normalization $\mathbf{w}_i^\top\mathbf{w}_i = 1$.

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

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

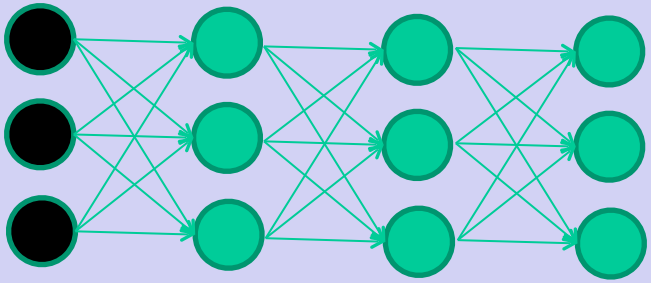


Autoencoder

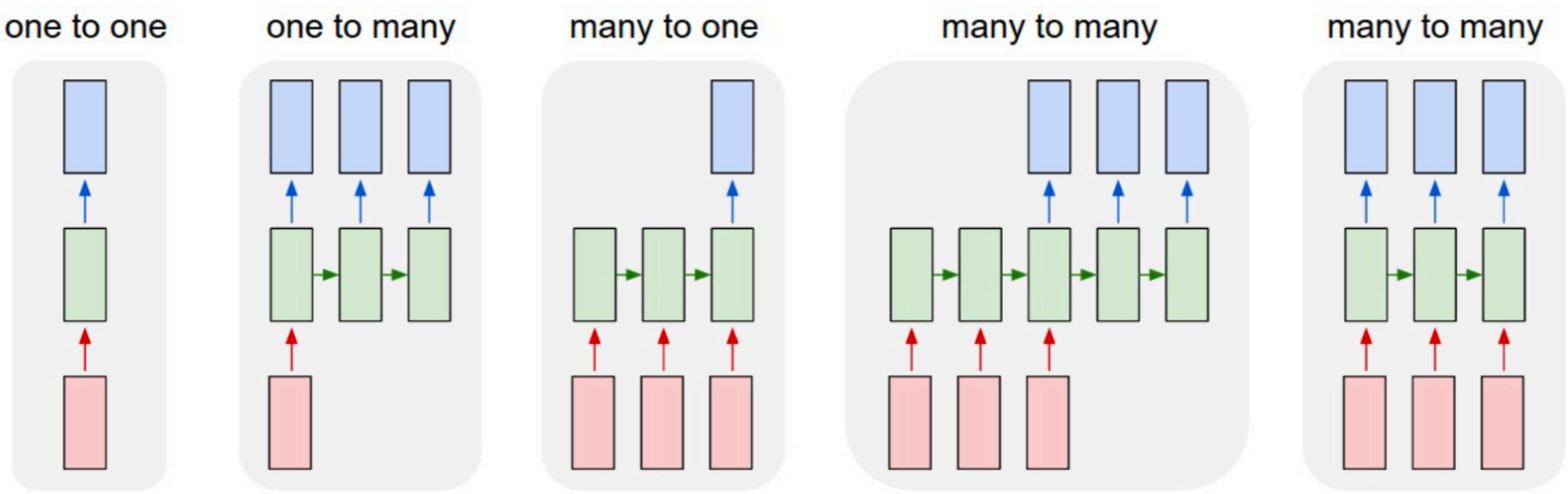


$$\text{Loss}(\mathbf{x};\theta) = \sum_i [\hat{y}_i(\mathbf{x};\theta) - x_i]^2$$

Recurrent neural networks



RNNs



- **One-to-one**: Classical (dense or convolutional) feedforward nets
- **One-to-many**: Given Image, generate sequence of words.
- **Many-to-one**: Given sequence of words, make classification (e.g., sentiment)
- **Many-to-many**: Text translation, on-line video segmentation

RNNs

- **Specialized** for:
  - processing a sequence of values $[\mathbf{x}(1), ..., \mathbf{x}(t), ..., \mathbf{x}(T)]$. ($t$ not necessarily physical time)
  - Sequences of variable length.

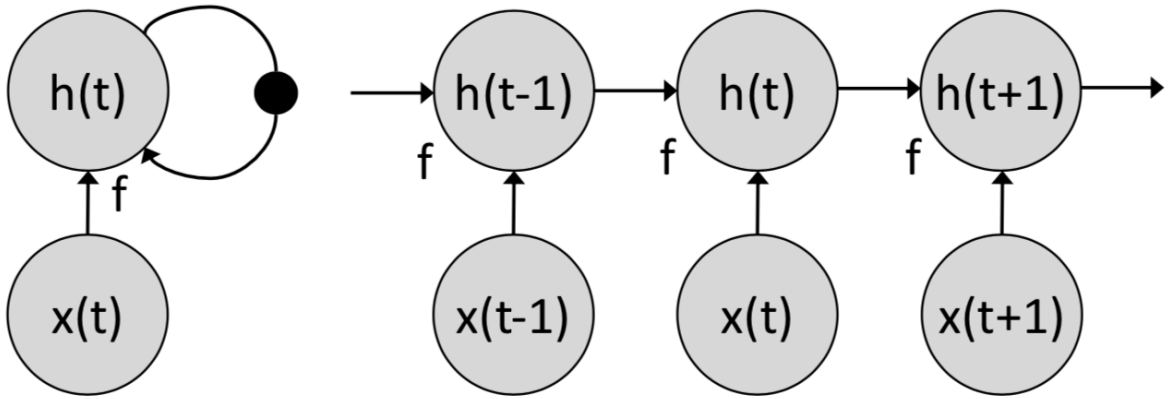
- **Example**: "*I went to Chile in 2013*" and "*In 2013, I went to Chile.*"
  - Extract the year the narrator went to Chile.
  - Traditional fully connected net would have separate parameters for each input feature, i.e. learn all of the rules of the language separately at each position in the sentence.

- **Key idea**: parameter sharing.
  - related idea: Convolution across a 1-D temporal sequence (time-delay neural networks – see Lang and Hinton, 1988; Waibel et al., 1989; Lang et al., 1990).

- **Example**: dynamical system driven by external signal $\mathbf{x}(t)$:

$$\mathbf{h}(t) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

  - Train network to predict future from the past.
  - $\mathbf{h}(t)$ is summary of present and past. It is lossy as it represents an arbitrary-length sequence $(\mathbf{x}_1, ..., \mathbf{x}_t)$ with fixed length.

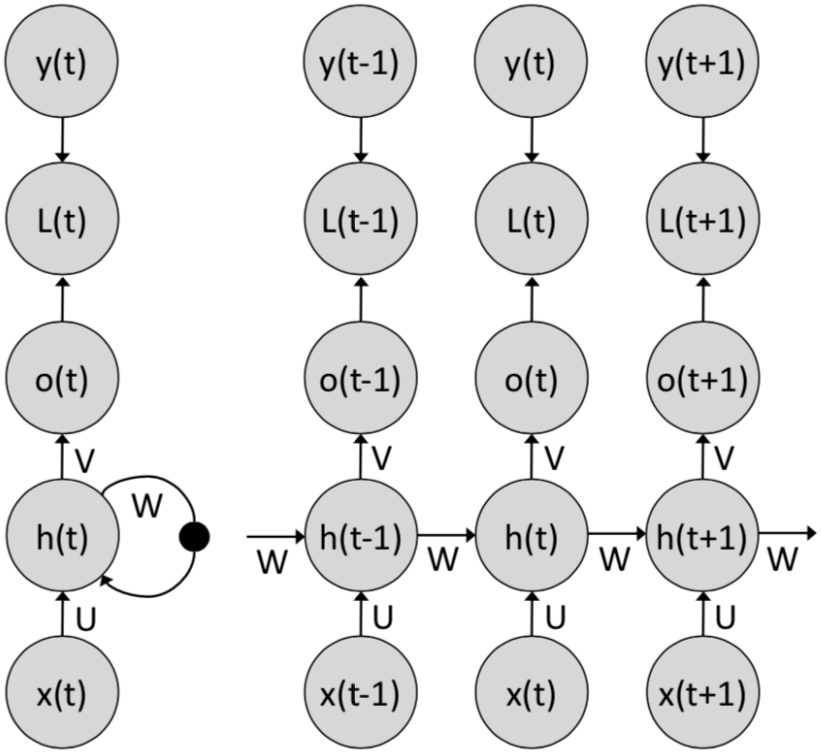


- **Unfolding**: represent recurrence after t steps with a function g_t:

$$\mathbf{h}(t) = g_t\left(\mathbf{x}(t), ..., \mathbf{x}(1)\right) = f\left(\mathbf{h}(t-1), \mathbf{x}(t); \theta\right)$$

  - Folding factorizes g_t into repeated application of a function f.
  - f has same input size regardless of sequence length.
  - Parameter sharing: use same transition function f every time step.

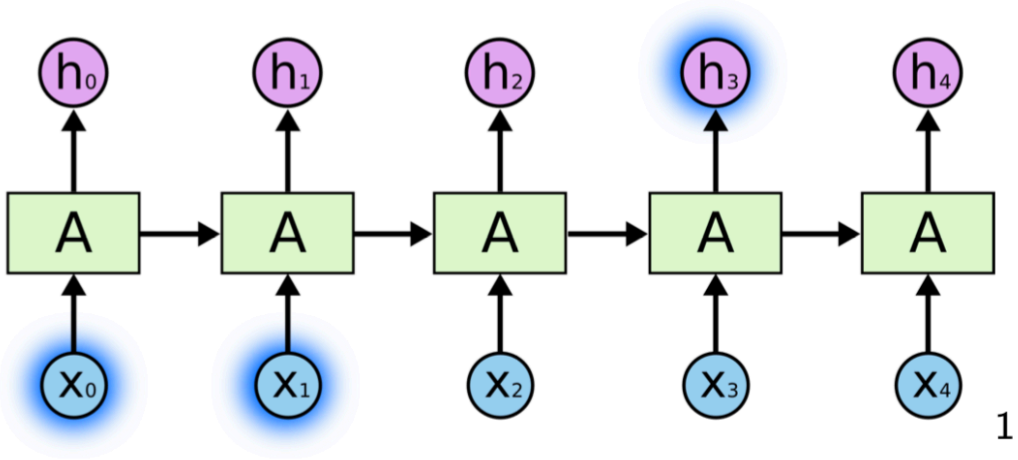
Basic universal RNN



- Can with finite size compute any function computable by a Turing machine (Siegelmann and Sontag, 1991; Siegelmann, 1995; Siegelmann and Sontag, 1995; Hyotyniemi, 1996).
- Loss L measures distance of predictions $\mathbf{o}_t$ from training targets $\mathbf{y}_t$.

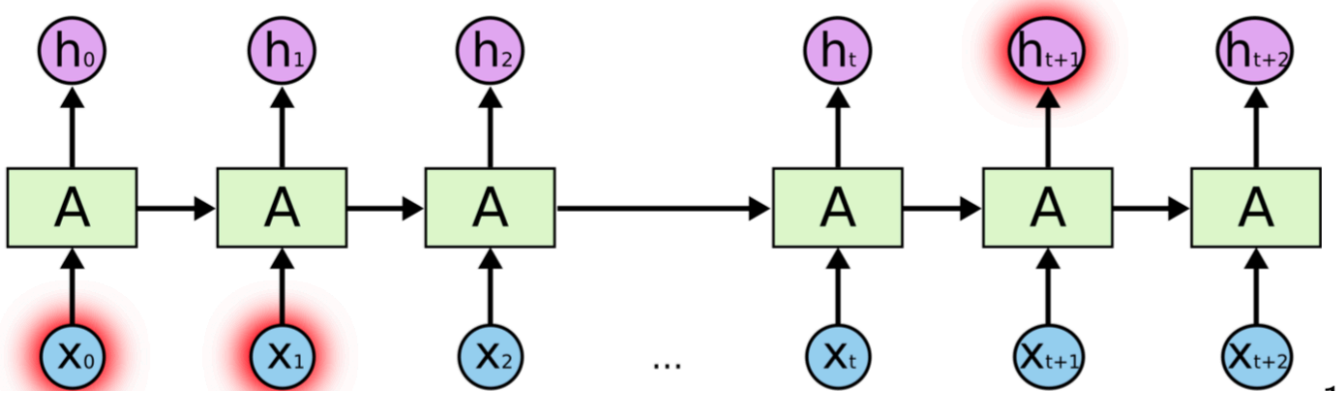
RNNs are deep — gradient annihilation

short "time" dependency: *The clouds are in the sky.*



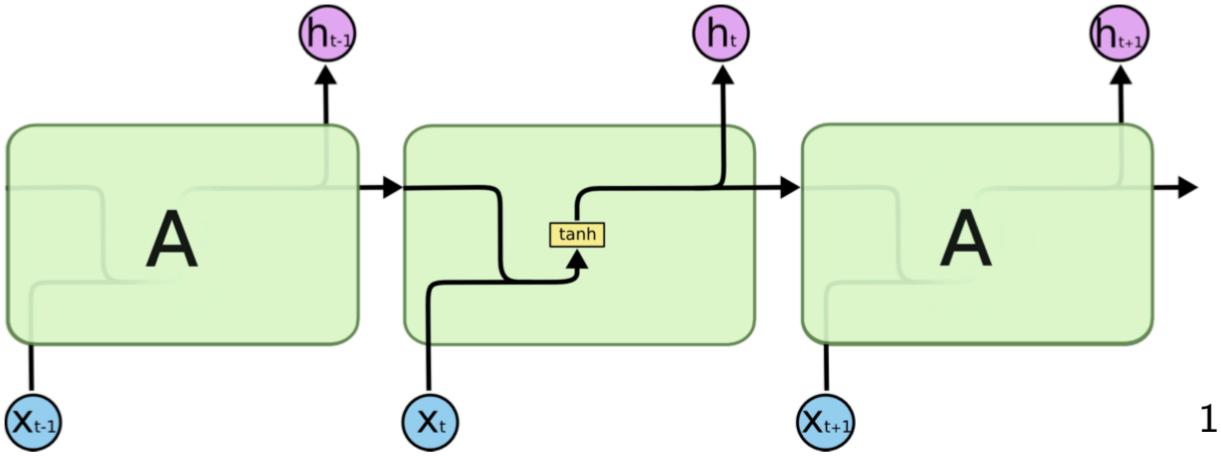
1

long "time" dependency: *I was born in France. Over the years, I have learned many foreign languages, but my native language is french.*



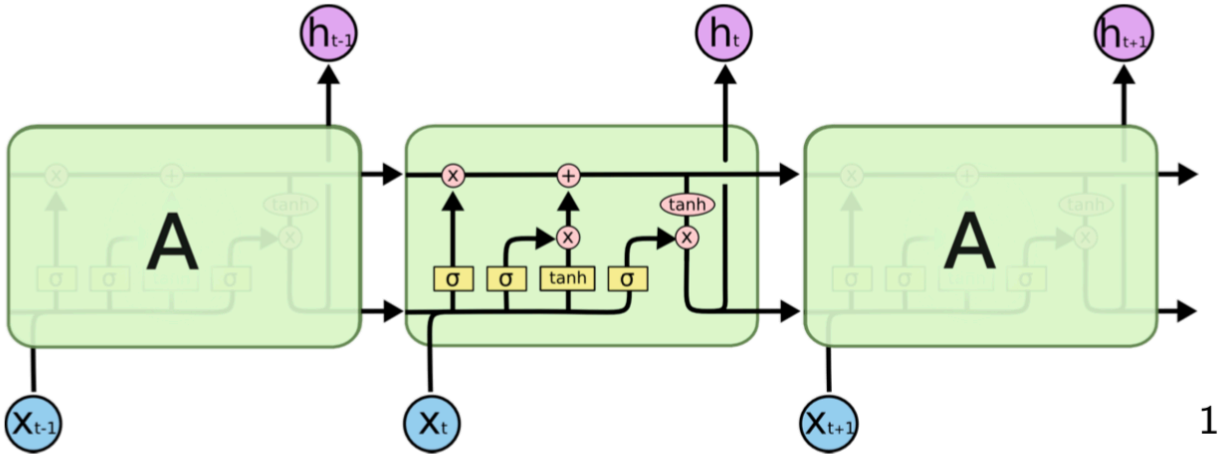
Long-short time memory

Simple RNN

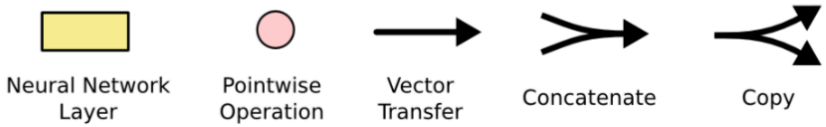


1

Long Short Time Memory (LSTM, Hochreiter & Schmidhuber 1997)



1



Generate Shakespeare

- 4.4 MB text input,
- 3-layer RNN, 512 hidden nodes per layer.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour."

Generate Math

- Source: 16 MB LaTeX code *http://stacks.math.columbia.edu/*
- Sample: almost compiles, minor corrections needed.

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. $\square$

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

Generate Math

Pretending Math[1]

- Source: 16 MB LaTeX code *http://stacks.math.columbia.edu/*
- Sample: almost compiles, minor corrections needed.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

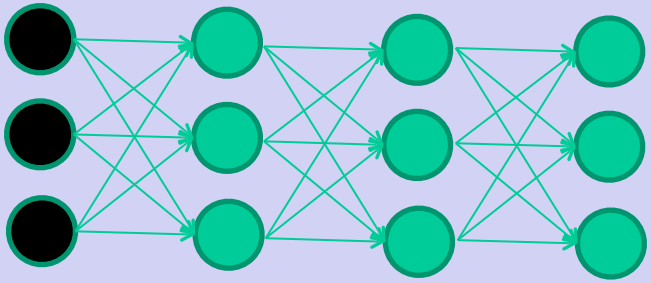
Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Commonly-used Tricks



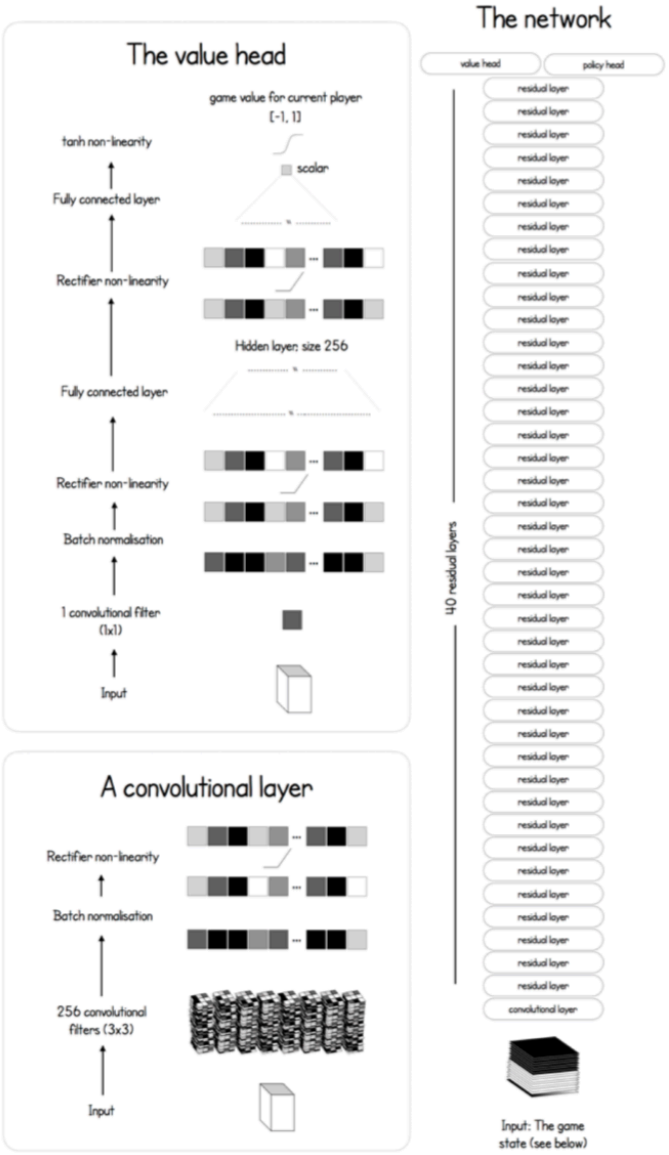
Multi-task learning

$$\mathbf{x} \rightarrow \mathbf{h}^{\text{shared}} \nearrow \mathbf{h}^{(1)} \rightarrow \mathbf{y}^{(1)}$$
$$\mathbf{x} \rightarrow \mathbf{h}^{\text{shared}} \searrow \mathbf{h}^{(1)} \rightarrow \mathbf{y}^{(2)}$$

Alpha Go Zero

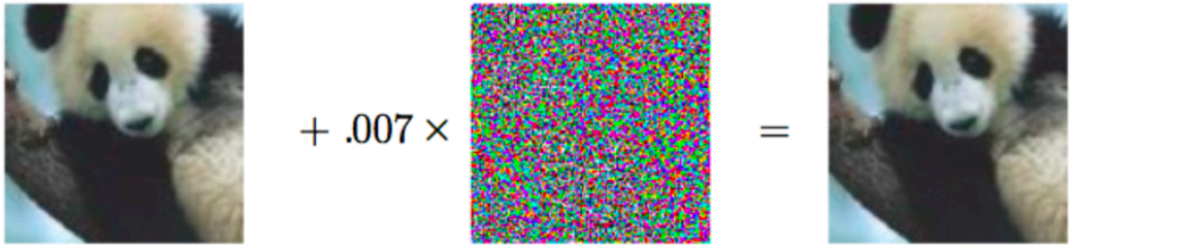
The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves



Adversarial attacks and training

- **Adversarial attacks** (trying to fake the network):
  - **Intended perturbation**: Accuracy reduced to random with test examples with small $\left\|\mathbf{x}_i^{\text{test}} - \mathbf{x}_j^{\text{train}}\right\|$ but large $\left\|y_i^{\text{test}} - y_j^{\text{train}}\right\|$ for given i, j.
  - **Random perturbation**: Add Gaussian noise with a very small amplitude. Picture indistinguishable for humans, but breaks neural network performance (Szegedy et al. 2014).

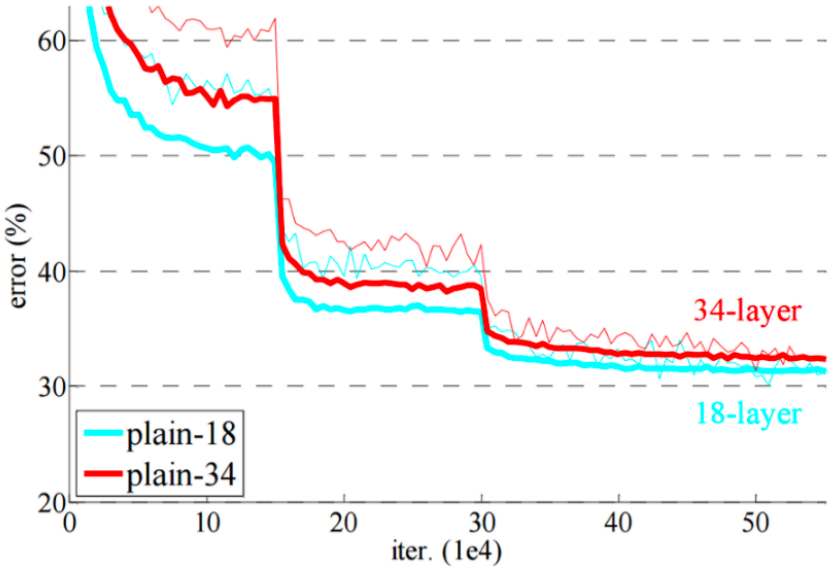


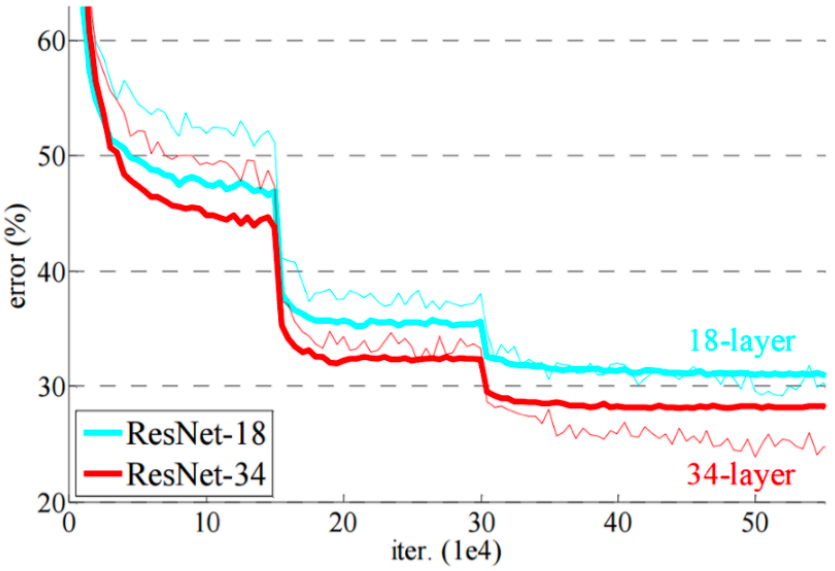
1

- **Adversarial training**: include adversarial attacks in the training data to force predictions to be locally constant near training data.

ResNets

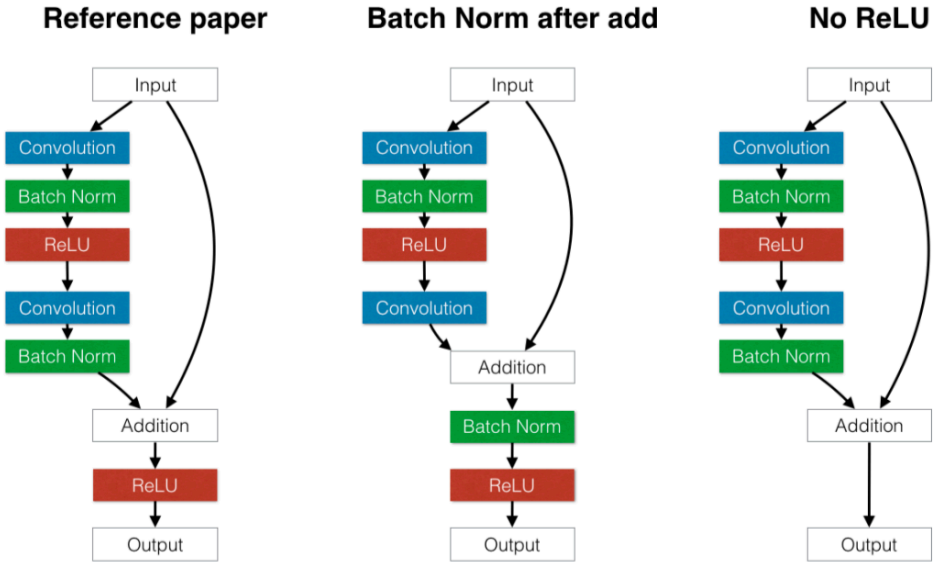
- With most network architectures, when adding layers (increasing depth), the training loss first reduces but then increases.
- Indicates training problem – adding layers make the network more expressive, so training loss should be non-increasing.
  $\rightarrow$ also affects the test loss.
- Residual Networks (He, Zhang, Ren, Sun, 2015) enable training of very deep networks.

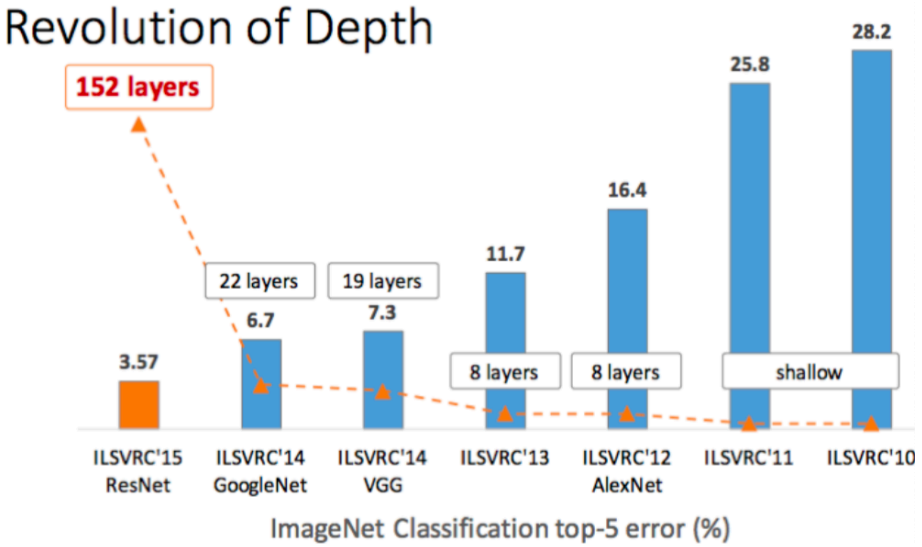




ResNets

- As in LSTMs, ResNets introduces a short-cut path that can carry gradients deep.
- ResNets are state-of-the-art in many problems (CIFAR, ImageNet, AlphaGo Zero).





Generative networks ==> Monday