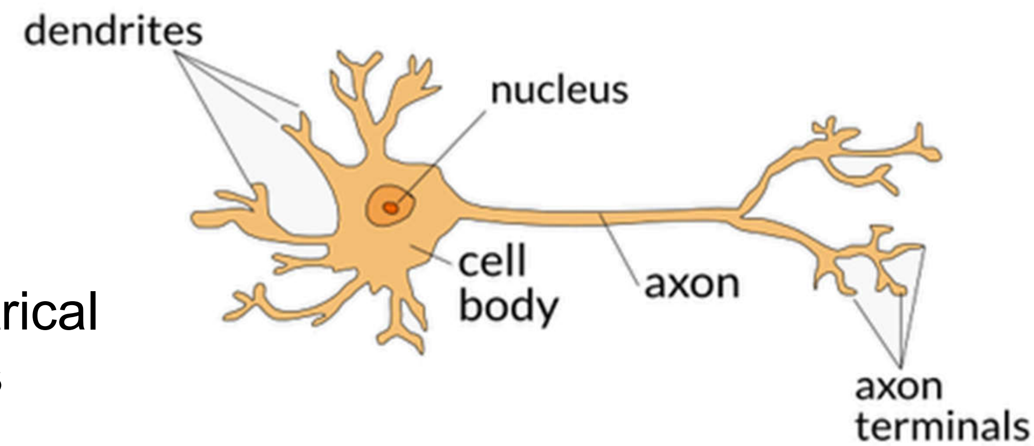
Machine Learning and Material Science
2. Artificial Neural Network

Steffen Brinckmann, Claas Hüter
IEK-2, Forschungszentrum Jülich GmbH

Artificial Neural Network (ANN)

Biologically inspired: represents information processing in brain



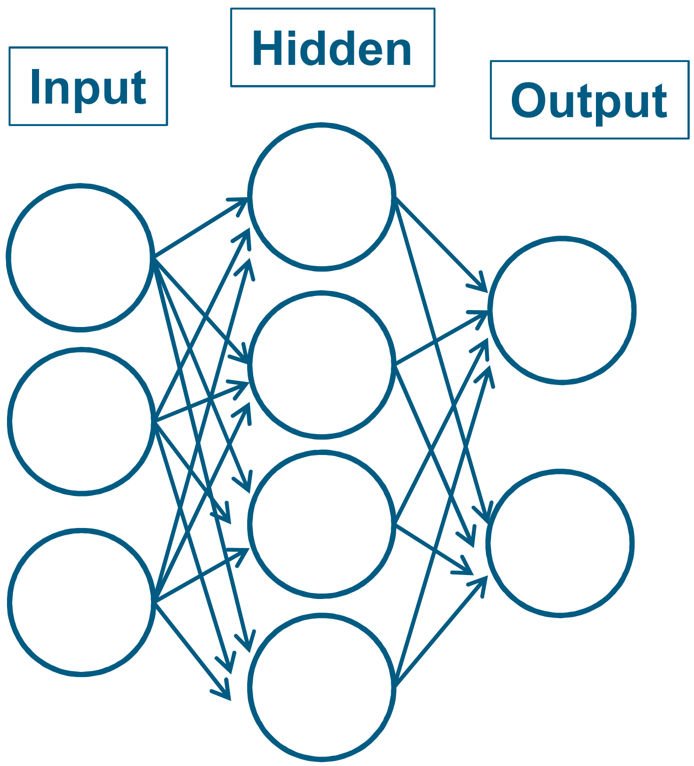
Dendrite: taking input (electrical impulse) from other neurons

Cell body: generating inferences from those inputs and produce output

Axon->Dendrite: Sending signals out to other neurons

https://appliedgo.net/perceptron/

Artificial Neural Network (ANN): classification and regression problems

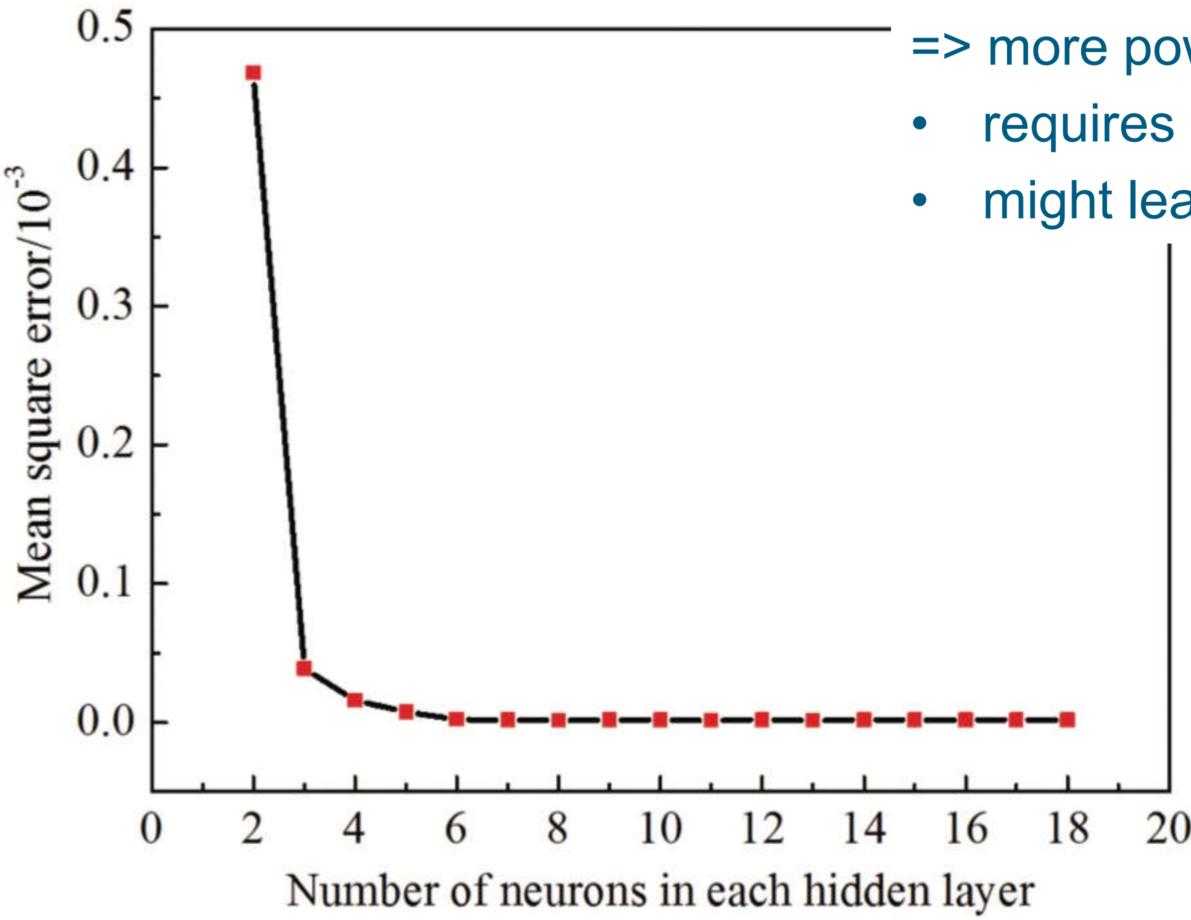


Input layer: some features

Output layer: target / response / prediction

Hidden layer(s): intermediate layers
to learn complicated relationships

Model complexity: number of neurons in hidden layer

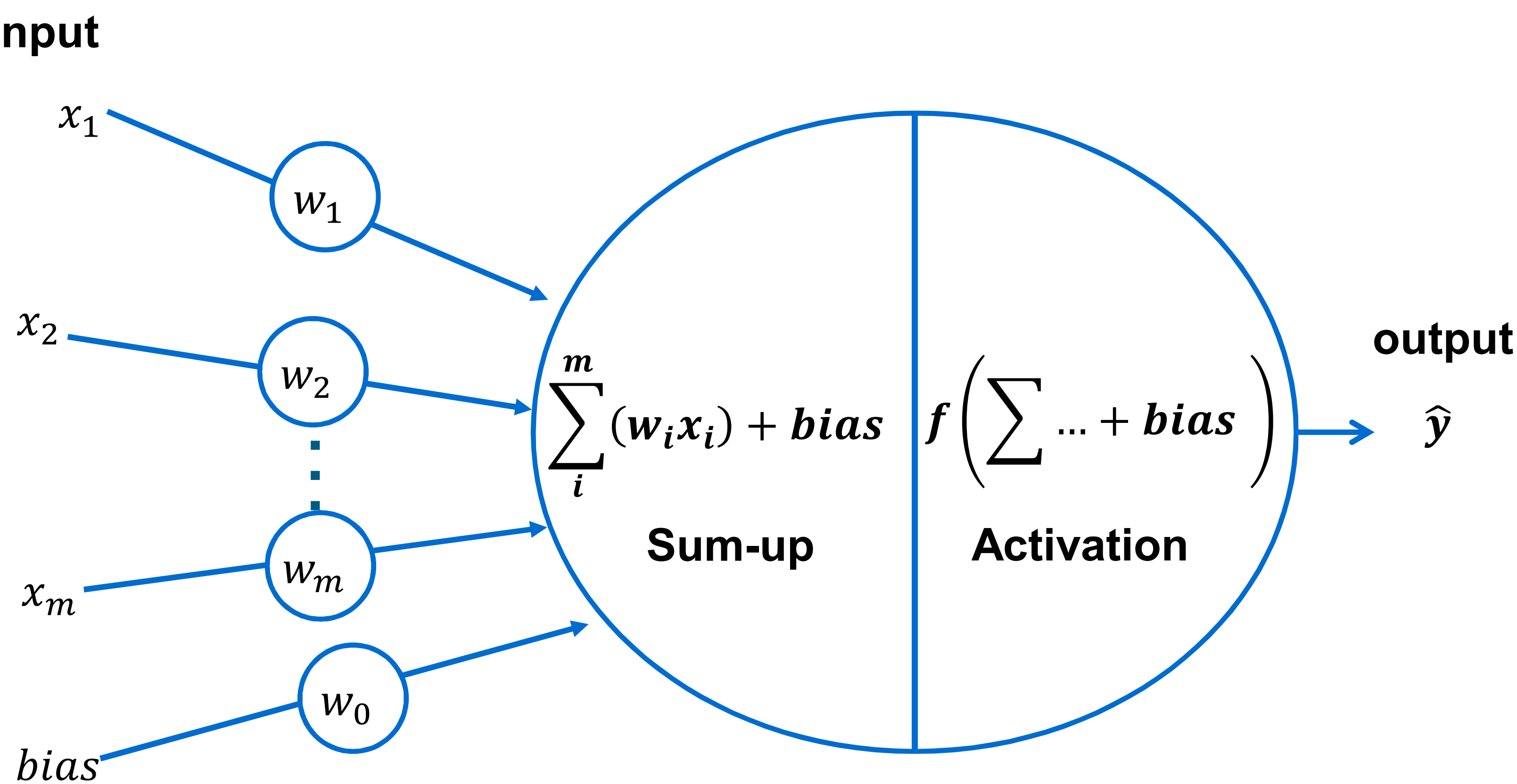


More hidden layers and neurons
=> more powerful model

- requires more data to train
- might lead to overfitting

G. Z. Quan et al., Materials Research **17**[5]:1102-1114, 2014.

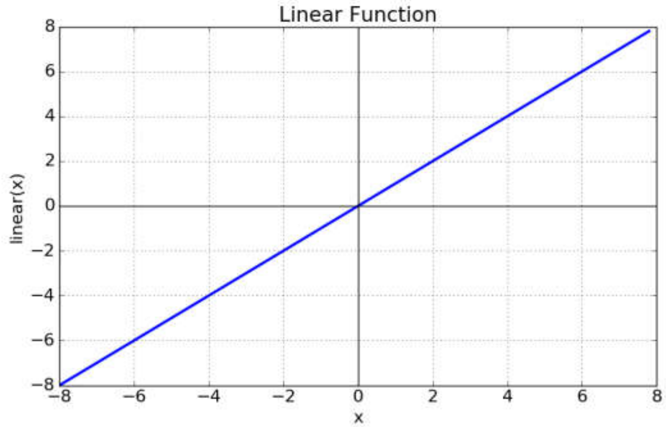
Single artificial neuron



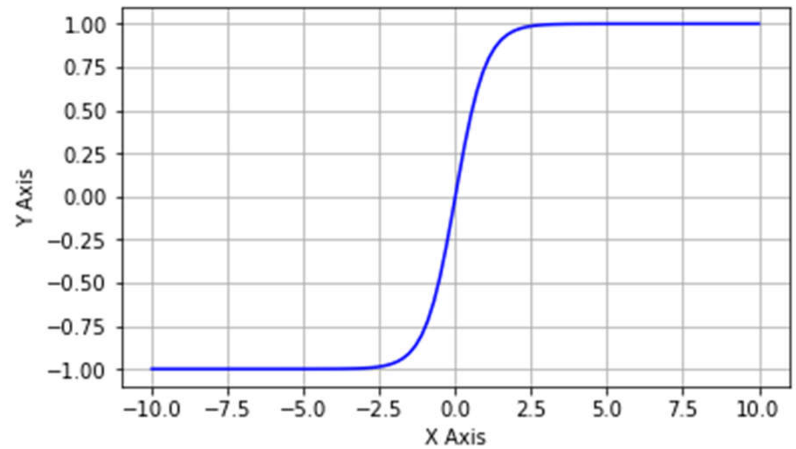
Simple ON-OFF activation function: $f(x) = \begin{cases} 1, if \sum_{i}^{m}(w_i x_i) + bias \geq 0 \\ 0, if \sum_{i}^{m}(w_i x_i) + bias < 0 \end{cases}$

Activation functions

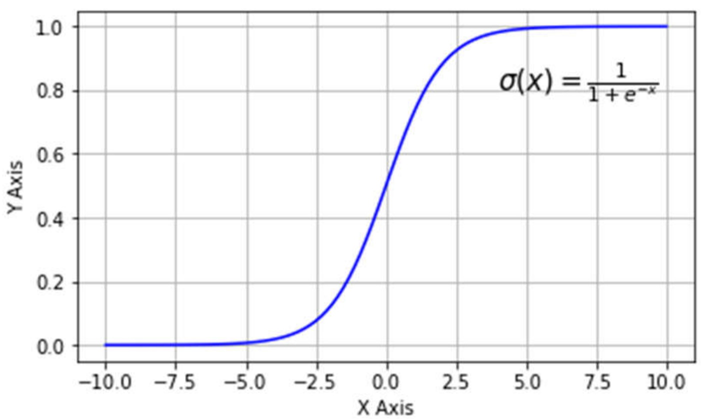
Linear function: DO NOT USE



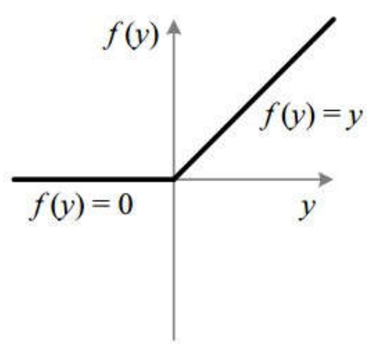
Tanh function: $A = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$



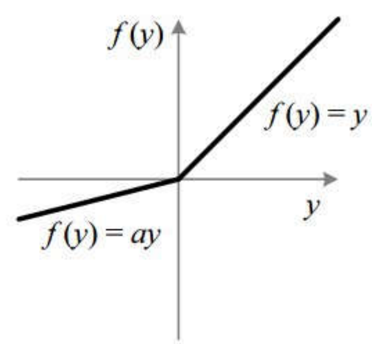
Sigmoid function: $A = \frac{1}{1+e^{-x}}$



ReLu function



Leaky ReLu function



Bias nodes in hidden and input layers

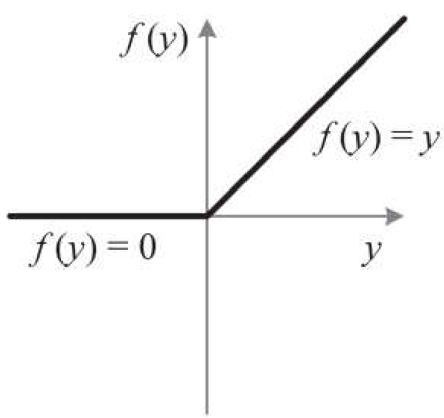
A single bias node per layer

Not connected to the previous layer

It can be arbitrary value, usually it is set to be 1

Why we need bias? Increase flexibility of model

Example: activation function ReLu $y = \max(0, w^T x + b)$

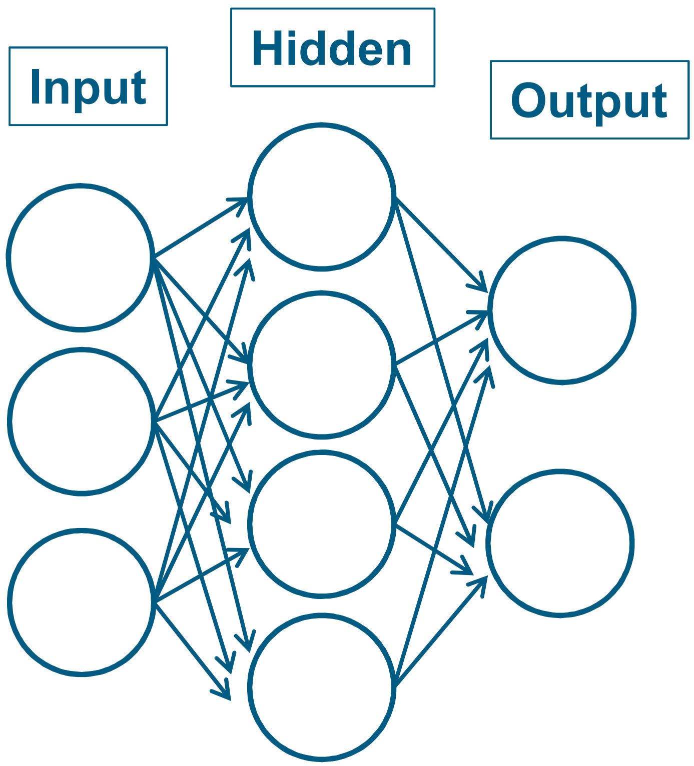


Neuron will activate, if

- $w^T x + b > 0$
- $w^T x > -b$

Bias term is an activation threshold

Artificial Neural Network (ANN):

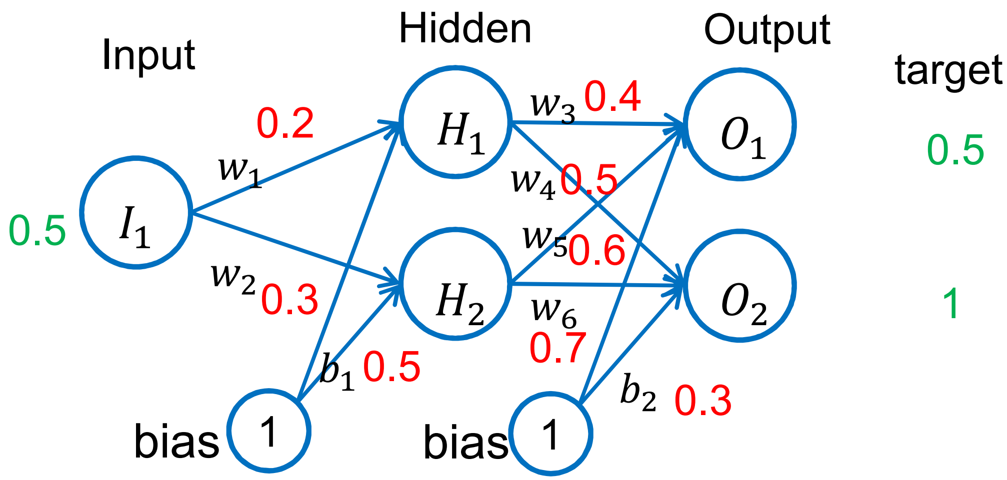


Input layer: some features

Output layer: target / response / prediction

Hidden layer(s): intermediate layers
to learn complicated relationships

Forward propagation



Activation function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Forward propagation: calculate outputs

$\boldsymbol{H_1}$: $Input = I_1 w_1 + 1 b_1 = 0.6$

$$Output = \frac{1}{1 + e^{-0.6}} = 0.65$$

$\boldsymbol{H_2}$: $Input = I_1 w_2 + 1 b_1 = 0.65$

$$Output = \frac{1}{1 + e^{-0.65}} = 0.66$$

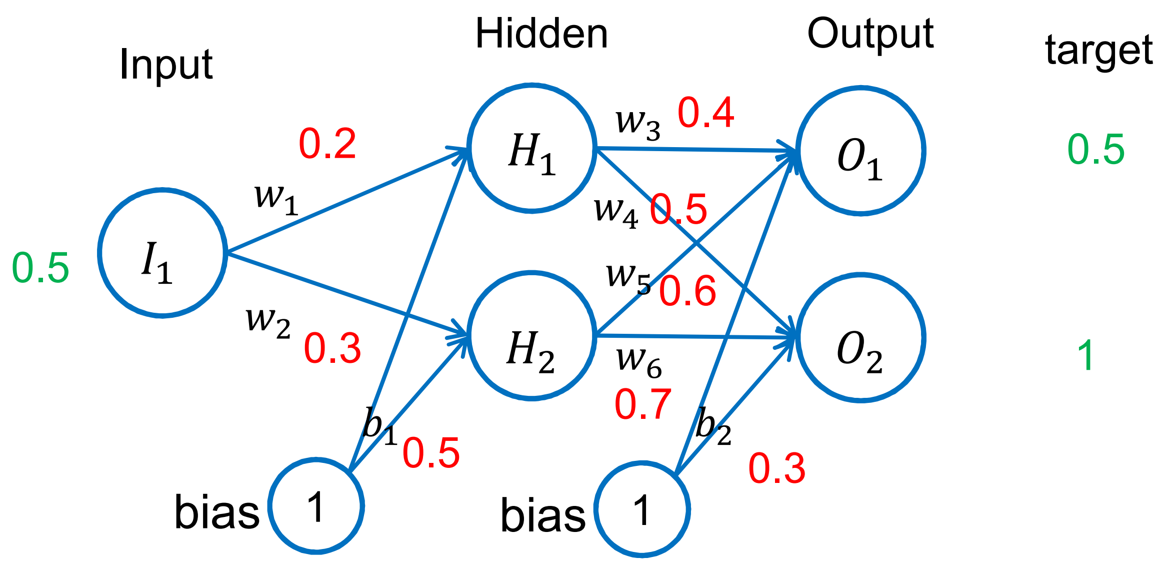
$\boldsymbol{O_1}$: $Input = H_1 w_3 + H_2 w_5 + 1 b_2 = 0.95$

$\boldsymbol{Output = 0.72}$

$\boldsymbol{O_2}$: $Input = H_1 w_4 + H_2 w_6 + 1 b_2 = 1.08$

$\boldsymbol{Output = 0.74}$

Error evaluation



Activation function:

$$f(x) = \frac{1}{1+e^{-x}}$$

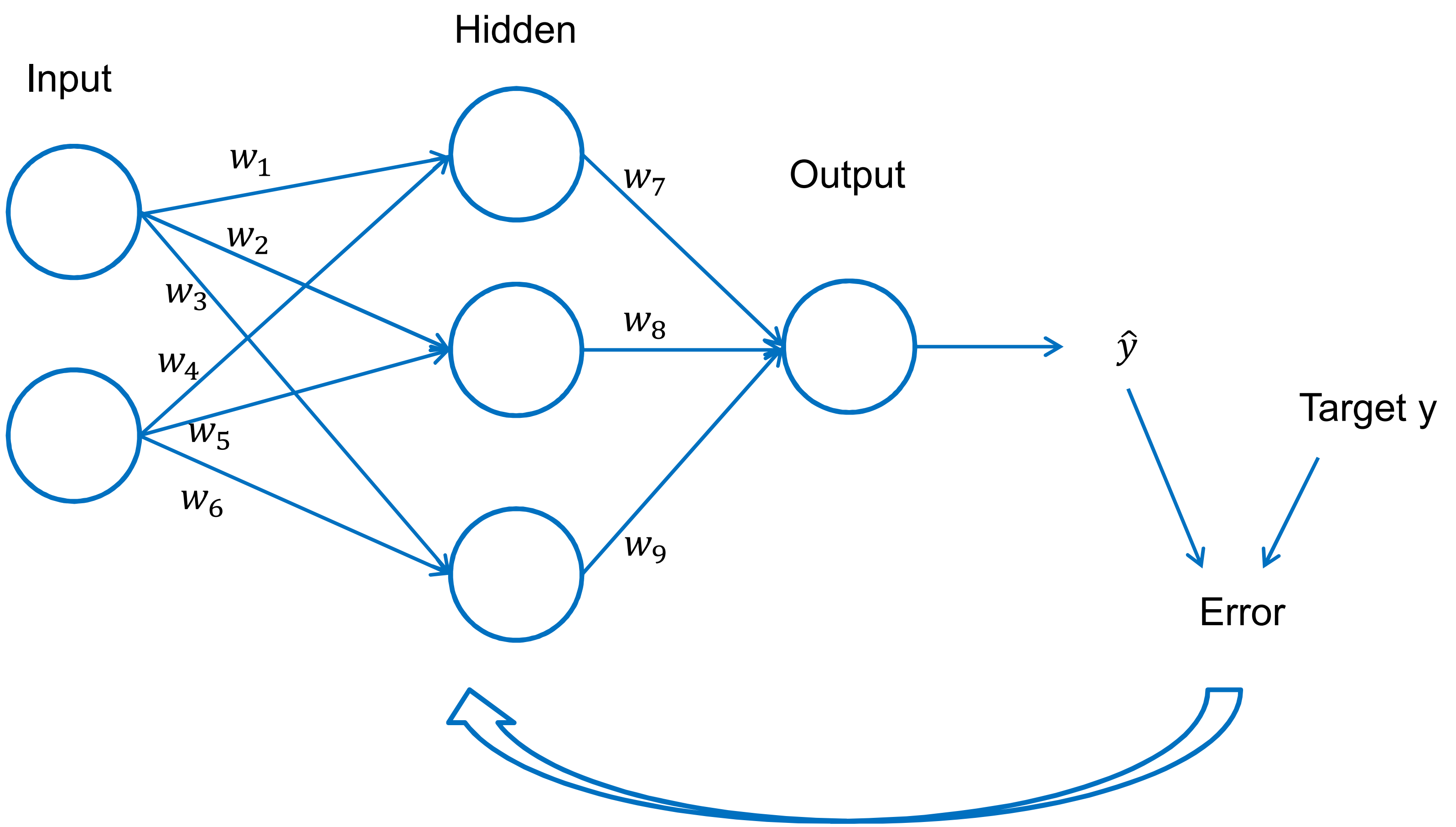
	predicted	target
O_1	0.72	0.5
O_2	0.75	1

Mean Squared Error:

$$E = \frac{1}{2}\sum(target - predicted)^2$$

=> Optimize red numbers

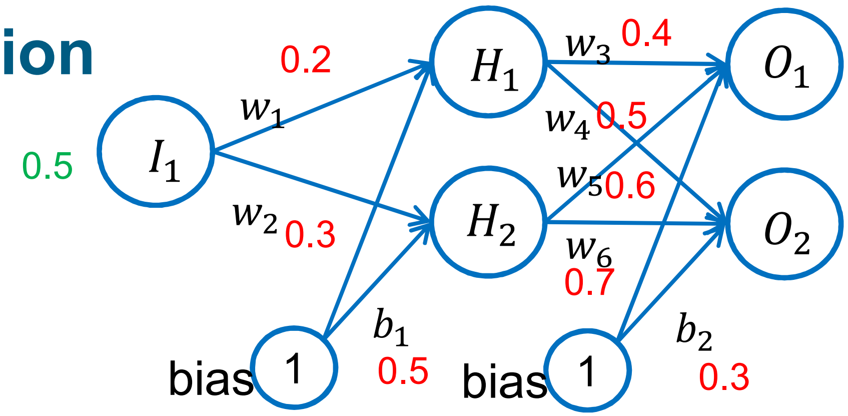
Learning by Back Propagation of error



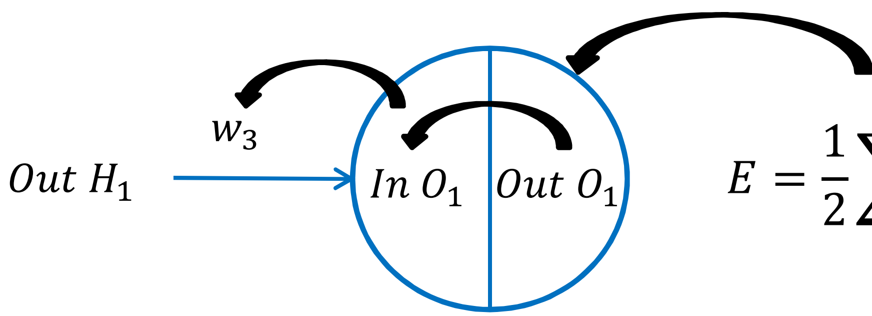
Back propagate error into the network and update weights

Gradient descent $w_i' = w_i - \alpha \frac{\partial Error}{\partial w_i}$ (same as in regression)

Backward propagation



How much does change of w_3 affect the error? $\frac{\partial E_{total}}{\partial w_3}$



$$E = \frac{1}{2}\sum(target - predicted)^2$$

Apply chain rule $\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_{total}}{\partial Out\ O_1}\frac{\partial Out\ O_1}{\partial In\ O_1}\frac{\partial In\ O_1}{\partial w_3}$

Derivatives of activation functions are programmed

Backward propagation

How much does change of w_3 affect the error? $\frac{\partial E_{total}}{\partial w_3}$

Apply chain rule: $$\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_{total}}{\partial Out\ O_1} \frac{\partial Out\ O_1}{\partial In\ O_1} \frac{\partial In\ O_1}{\partial w_3}$$

$$\frac{\partial E_{total}}{\partial Out\ O_1} = \frac{1}{2} 2(target\ O_1 - Out\ O_1)(-1) = 0.22$$

	predicted	target
O_1	0.72	0.5
O_2	0.75	1

$$\frac{\partial Out\ O_1}{\partial In\ O_1} = \frac{e^{-In O_1}}{(1 + e^{-In\ O_1})^2} = Out\ O_1(1 - Out\ O_1) = 0.20$$

Activation function: $\frac{1}{1+e^{-In\ O_1}}$

$$\frac{\partial In\ O_1}{\partial w_3} = Out\ H_1 = 0.65$$

$In\ O_1 = Out\ H_1 w_3 + Out\ H_2 w_5 + b_2$

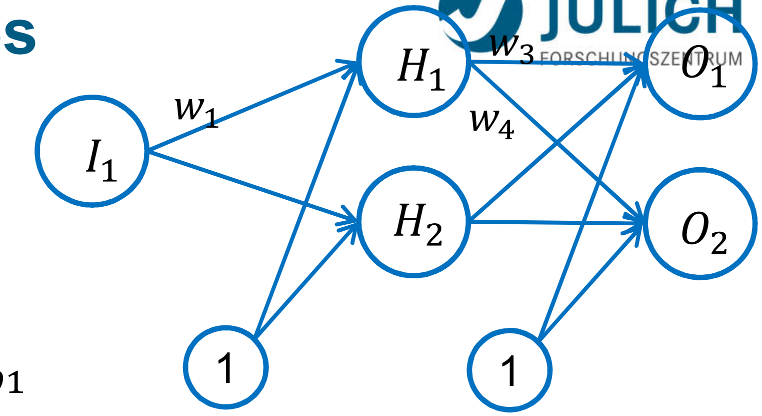
$\Rightarrow$ $\frac{\partial E_{total}}{\partial w_3} = 0.029$ $\Rightarrow$ $w'_3 = w_3 - \alpha \frac{\partial E_{total}}{\partial w_3}$

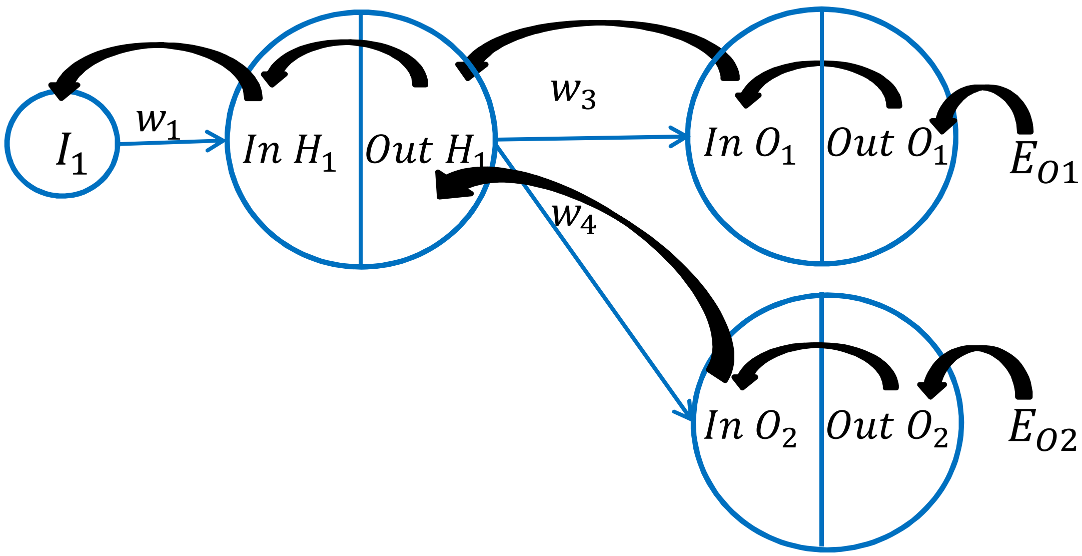
$w'_3 = 0.4 - 0.3 * 0.029 = 0.39$

α: learning rate $\alpha = 0.3$

Backward propagation in branches

How much a change in w_1 affect total error?





$$\frac{\partial E_{total}}{\partial w_1} = \frac{\boldsymbol{\partial E_{total}}}{\boldsymbol{\partial Out\ H_1}} \frac{\partial Out\ H_1}{\partial In\ H_1} \frac{\partial In\ H_1}{\partial w_1}$$

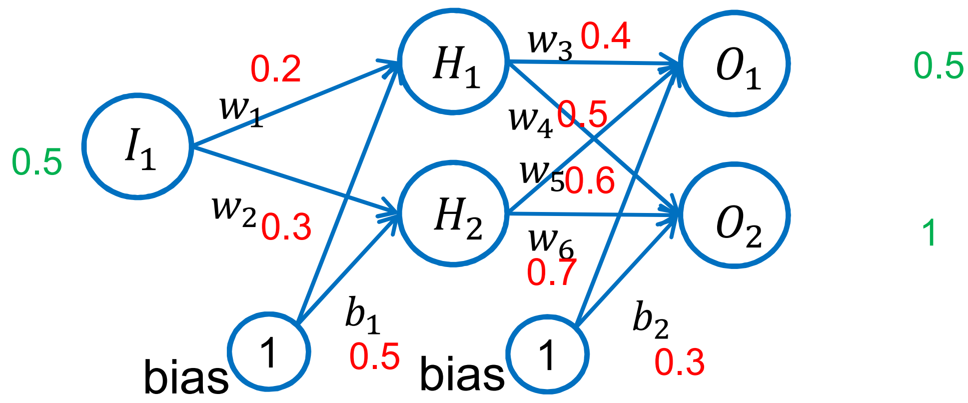
$$\frac{\partial E_{total}}{\partial Out\ H_1} = \frac{\partial Error\ O_1}{\partial Out\ H_1} + \frac{\partial Error\ O_2}{\partial Out\ H_1}$$

Sum of derivatives

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial Error\ O_1}{\partial Out\ O_1} \frac{\partial Out\ O_1}{\partial In\ O_1} \frac{\partial In\ O_1}{\partial Out\ H_1} + \frac{\partial Error\ O_2}{\partial Out\ O_2} \frac{\partial Out\ O_2}{\partial In\ O_2} \frac{\partial In\ O_2}{\partial Out\ H_1}$$

Simple Artificial Neural Network

Initial



1st prediction

$$w'_1 = 0.200209266$$
$$w'_2 = 0.300228378$$
$$b'_1 = 0.500875290$$
$$w'_3 = 0.391376778$$
$$w'_4 = 0.509260876$$
$$w'_5 = 0.591225134$$
$$w'_6 = 0.709423733$$
$$b'_2 = 0.300987606$$

	1st prediction	**2nd prediction**	**target**
O_1	0.721611403	0.719572715	0.5
O_2	0.747011298	0.749541084	1
Error	0.056557449	0.055470923	

updating weights => decreases error

many iterations ("epochs") => error below threshold

Learning rate

determines how fast weights / model change

may be different for different layers

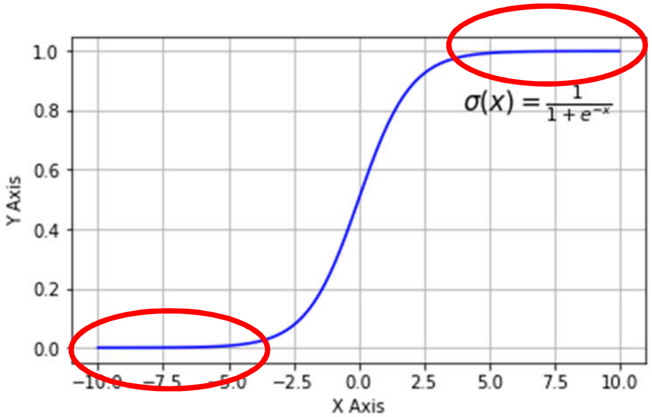
high learning rate, then training may not converge

Start: large learning rate because random weights are non-optimal

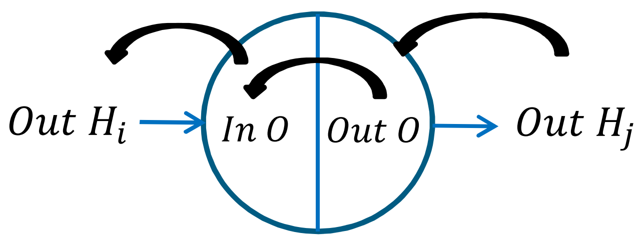
Then: learning rate should decrease for find-grained update of weights

Vanishing gradients during backpropagation

Sigmoid function: $A = \frac{1}{1+e^{-x}}$



$$\frac{dy}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} \qquad x \to \pm\infty, \quad \frac{dy}{dx} \to 0$$



$$\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_{total}}{\partial Out\ O_1} \underbrace{\frac{\partial Out\ O_1}{\partial In\ O_1}}_{\approx 0} \frac{\partial In\ O_1}{\partial w_3}$$

Activation functions (sigmoid, tanh):

- if gradients are almost 0
- weights of neurons change very slowly
- weights of up-stream neurons also very slowly change
- called “saturated neurons”

Training

1 epoch: one forward pass and backward pass of ALL samples

Batch size: number of samples in one forward/backward pass.

Example

800 training samples

16 batches & batch size 50

computing $\frac{\partial Error}{\partial w_i}$ of 50 samples

calculate average and update weights

do that for 16 batches

incl. updating the weights

finished 1 epoch and

start next ……

Example: Lattice constant of $GdFeO_3$-type perovskite

Linear Regression 4 features: $X = (r_A, r_B, t, r_A/t)$

Artificial Neural Network 5 features: $X = (r_A, r_B, x_A, x_B, z_A)$

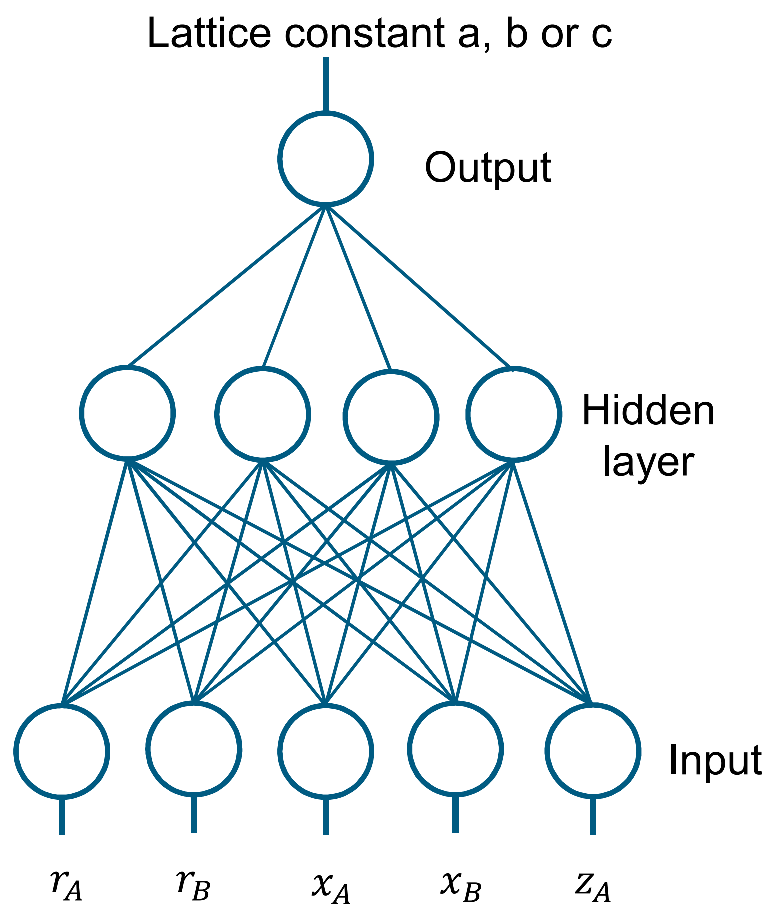
main advantage of Artificial Neural Networks

- many variables
- complex relationships

C. Li et al., J. Physics and Chemistry of Solids **64**:2147-2156, 2003.

Example: Lattice constant of $GdFeO_3$-type perovskite

3 Artificial Neural Networks



Activation function in output layer
pure linear

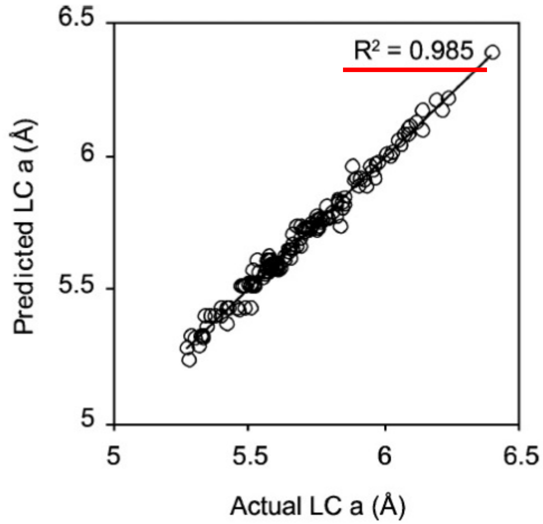
Activation function in hidden layer
Hyperbolic tangent sigmoid

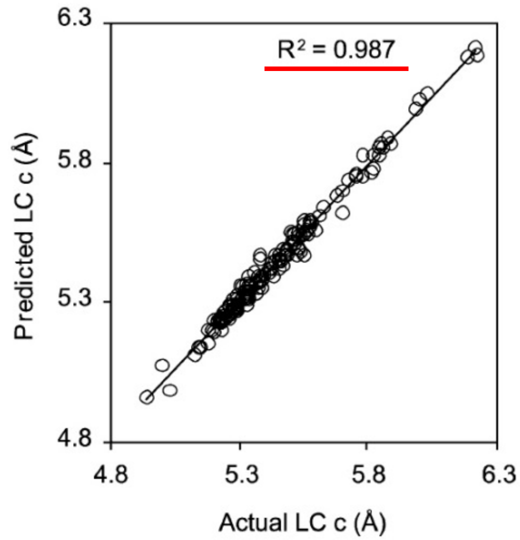
161 $GdFeO_3$-type compounds
Training data: random 90%
Testing data: remainder

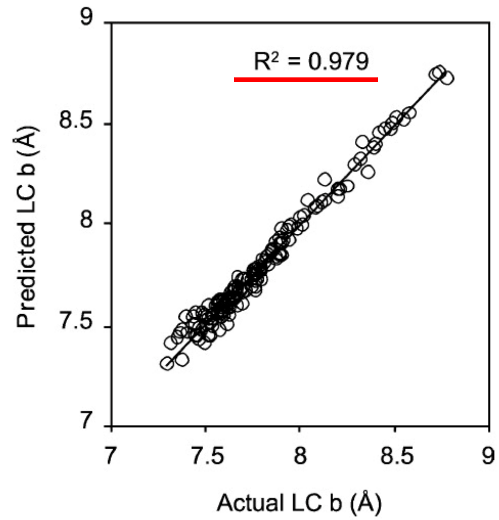
C. Li et al., J. Physics and Chemistry of Solids **64**:2147-2156, 2003.

Example: Lattice constant of $GdFeO_3$-type perovskite

Comparison plots







Percentage of Absolute Difference

$$PAD(\%) = \frac{|experimental \; - predicted|}{experimental} \times 100$$

	ANN and Linear Regression		
PAD (%)	a	b	c
Average	0.35 and 0.93	0.44 and 0.82	0.34 and 0.77

C. Li et al., J. Physics and Chemistry of Solids **64**:2147-2156, 2003.

Other examples of Artificial Neural Network in material science

Melting point of AB-type intermetallic compounds
C. H. Li et al., J. Phys. Chem. Solid **57**[12]:1797-1802, 1996

Young’s modules and yield stress of steel
K. K. Tho et al., Modelling Simul. Mater. Sci. Eng. **12**:1055-1062, 2004

Flow stress of 42CrMo high strength steel at different strain rate and temperature
G. Z. Quan et al., Materials Research **17**[5]:1102-1114, 2014

JÜLICH FORSCHUNGSZENTRUM

Convolutional

Neural

Networks

Image is matrix of pixel values

Black/white image: a single 2d matrix

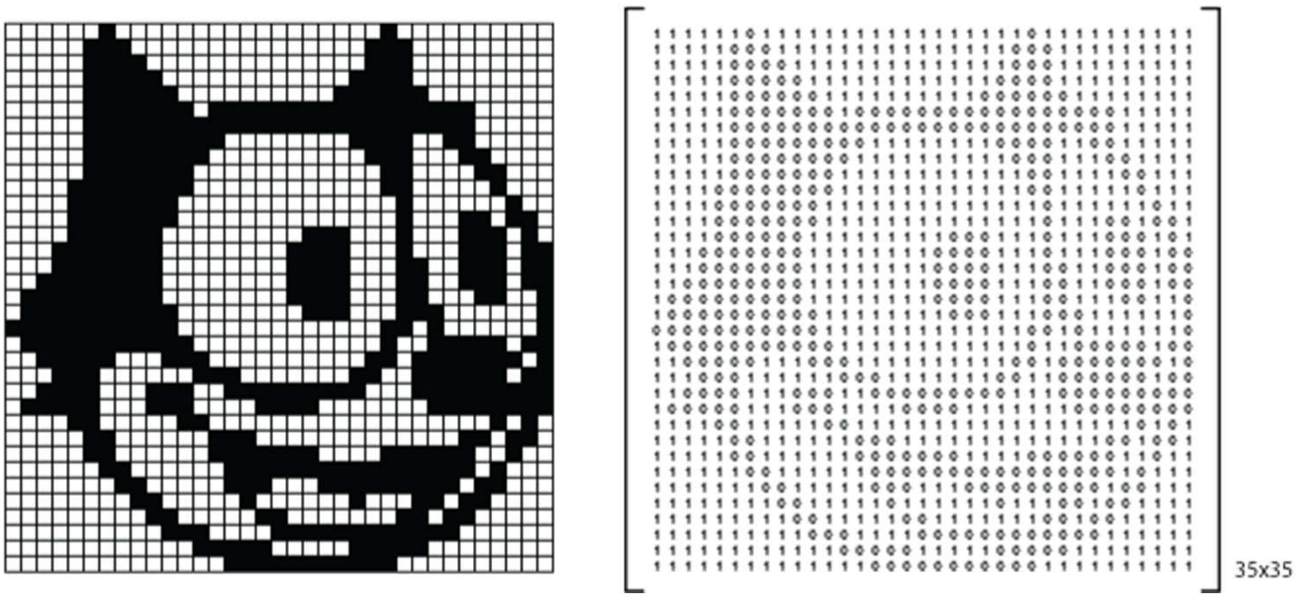
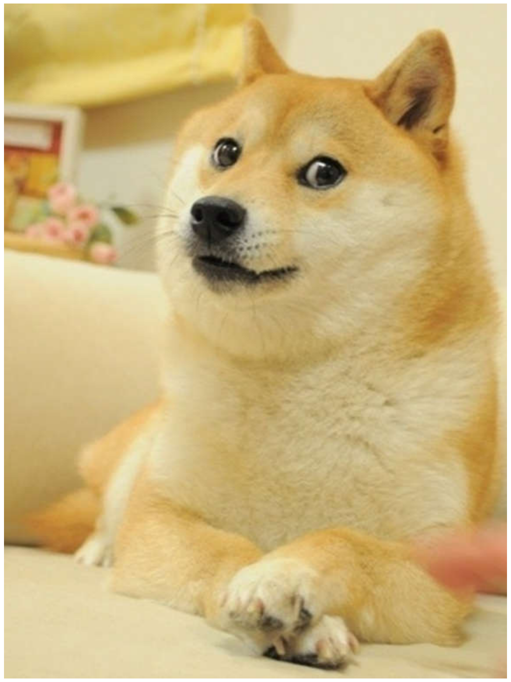


Image is matrix of pixel values



What we see

23	34	101	189	78	251	190
135	147	98	11	103	60	67
9	69	103	15	157	64	74
175	163	93	35	91	88	36
12	224	64	67	27	89	49
81	8	12	94	32	22	60
36	26	17	119	54	10	11
68	11	22	115	90	134	52
2	20	9	46	89	122	97

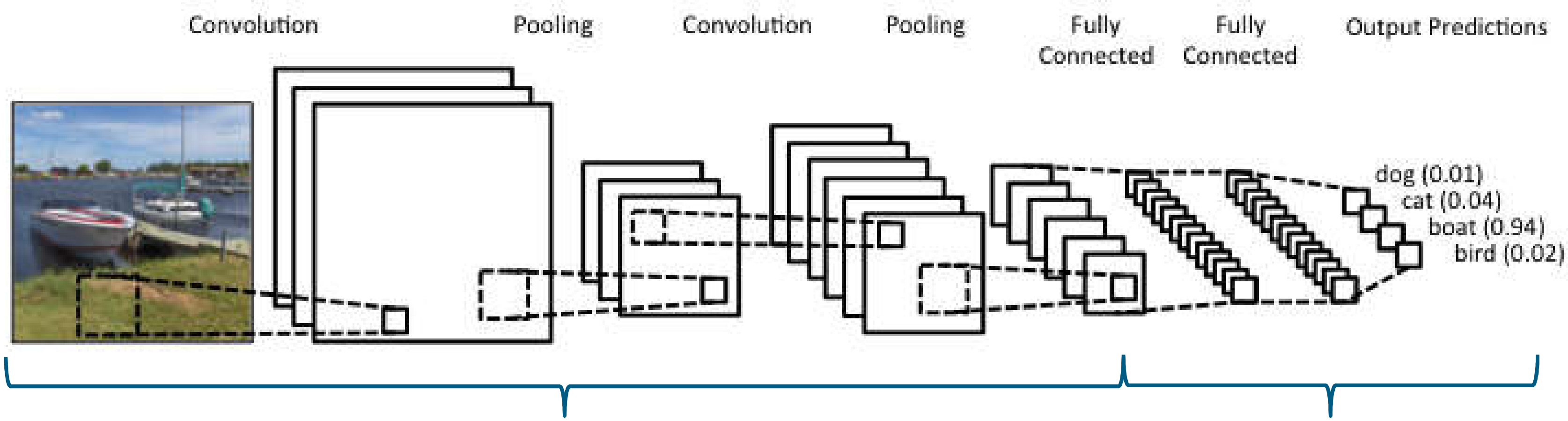
What computers see

Values from 0 to 255= 2^8 in matrix

One matrix/layer for red, green, blue

Standard image (i,j,3)

Convolutional Neural Network



Features extraction

- **Convolution layer**
- **Pooling layer**

Classification

Image source: https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780

Convolution layer

Convolves data (matrix) with linear filter

$$(h_k)_{ij} = (W_k * x)_{ij} + b_k$$

k: *k*-th feature map in convolution layer (feature: horizontal line)

(*i*, *j*) are location

x: input data (matrix of pixel)

W_k, b_k: trainable parameters (weights of linear filters and bias) for *k*-th feature map

$(h_k)_{ij}$: output of neuron; *k*-th feature map with position (*i*,*j*)

'$*$' : 2D convolution operation of input and feature map

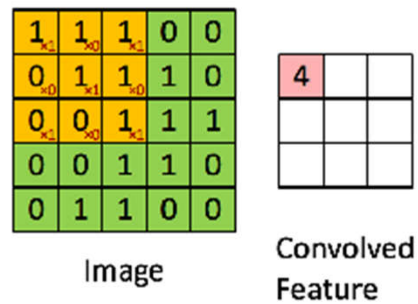
Convolution layer

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Matrix of
pixel values

1	0	1
0	1	0
1	0	1

Filter
or feature detector



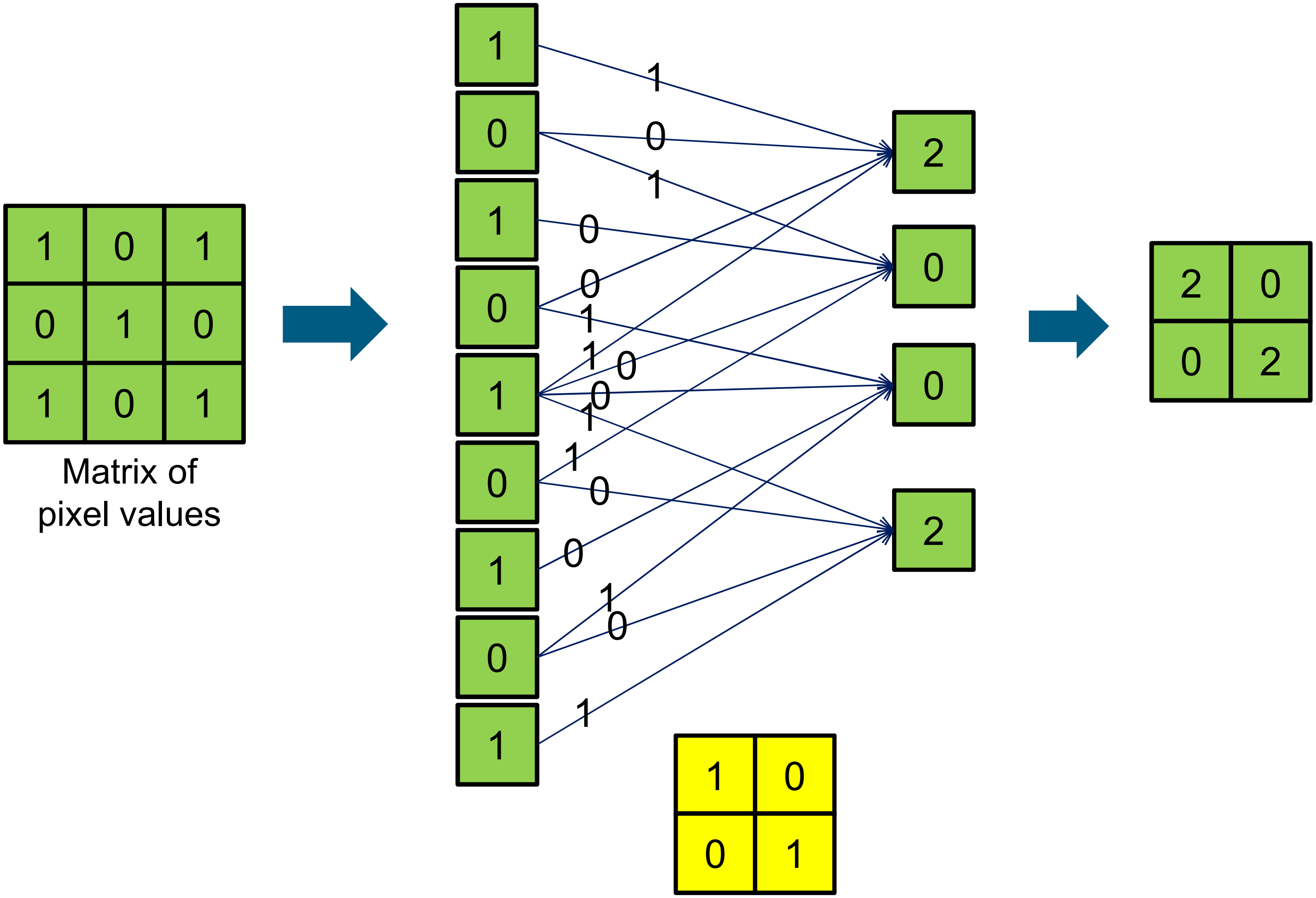
4	3	4
2	4	3
2	3	4

convolved feature
or feature map

Different filters: detect different patterns
Number of filters = number of feature map

Image Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolution Neural Network is Artificial Neural network



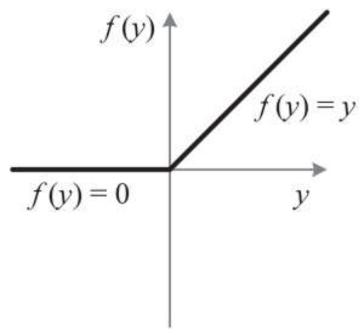
Filter or feature detector

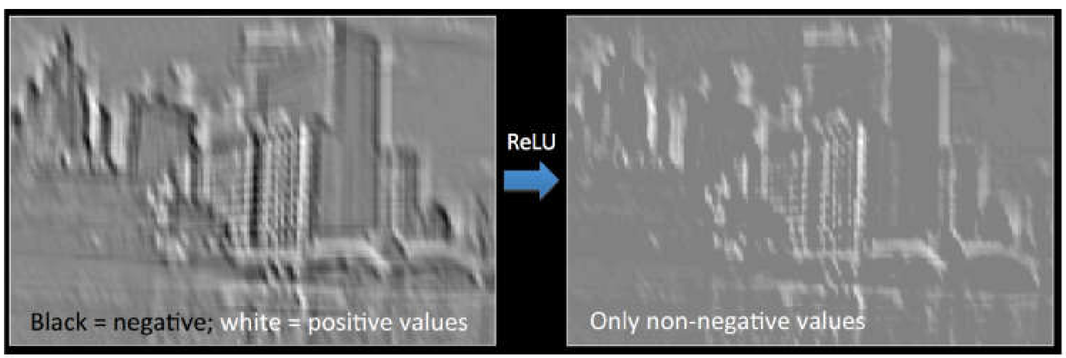
Convolution Neural Network has activation functions

4	3	4
2	4	3
2	3	4

convolved feature
or feature map

ReLU





- replace negative values by zero
- ignore features
- Linear → Nonlinear
- convolutions are linear
- same problem as in artificial neural networks
  (if linear, Neural Network is one layer deep)

Image Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Pooling

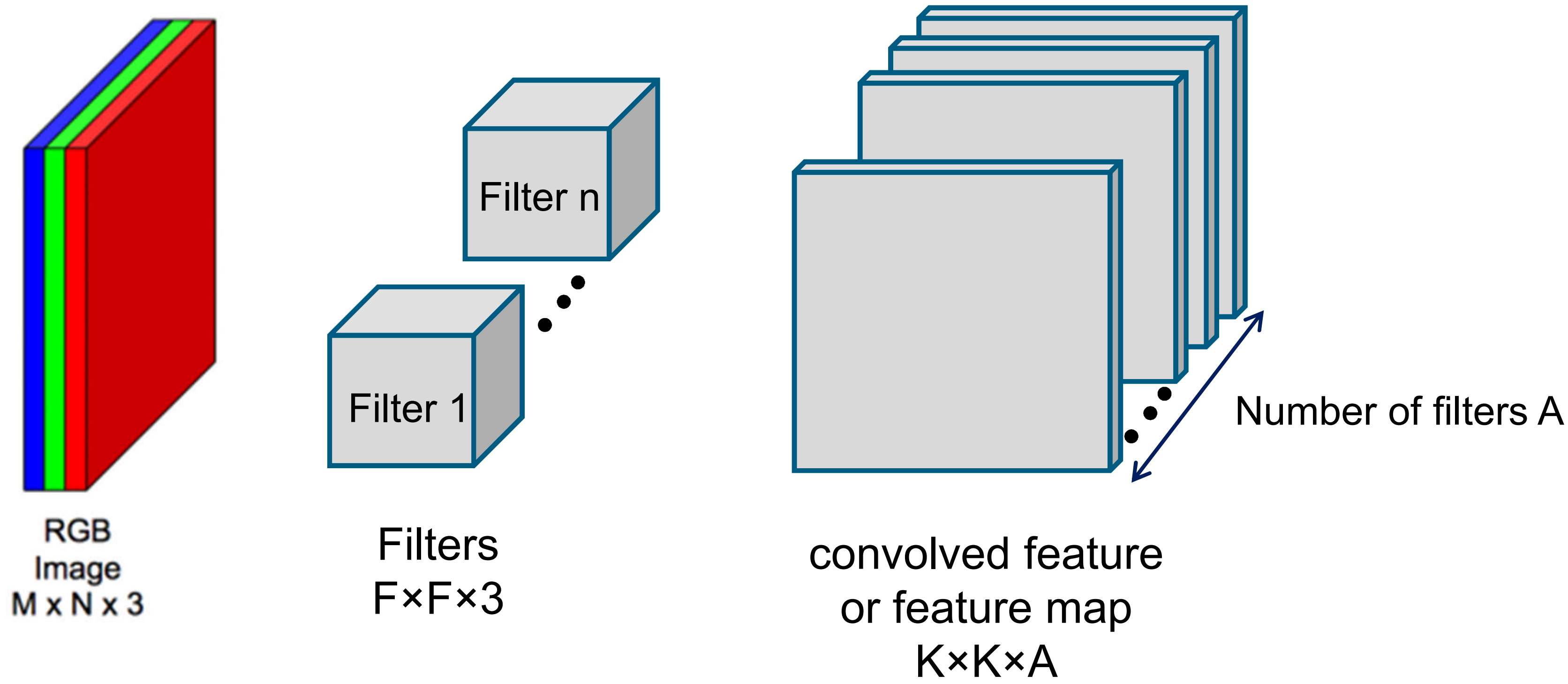
Reduce size of feature maps

Retain most important information

Types: Max, Average, Sum…

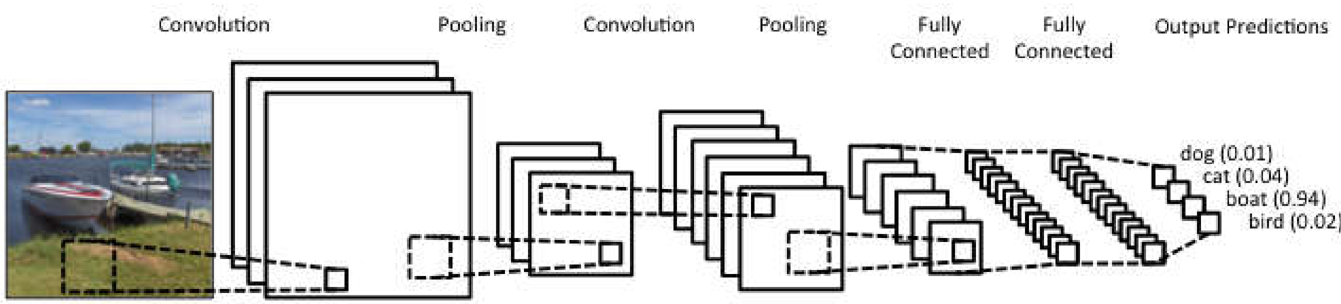
Feature map

Filters for three colors result into many feature maps



Summary

Input Image (pixel matrix) → Convolutional & Pooling layers → high-level features → Fully connected layer → predictions



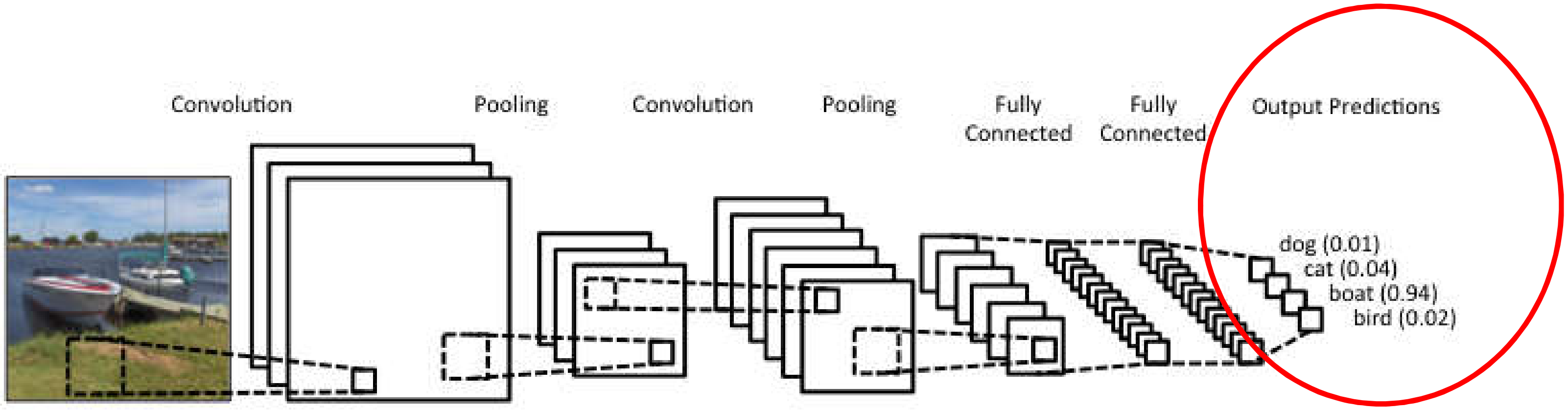
Training task: It is a boat

Input: boat image

Target output: [0, 0, 1, 0]

1. Initialization filter values and weights
2. Forward propagation & predicted value
3. $\mathrm{Error}(t, O)$
4. Back propagation: $\frac{\partial \mathrm{Error}}{\partial \mathrm{weights}}$ and $\frac{\partial \mathrm{Error}}{\partial \mathrm{filter\ values}}$
5. Updating continue to minimizing error

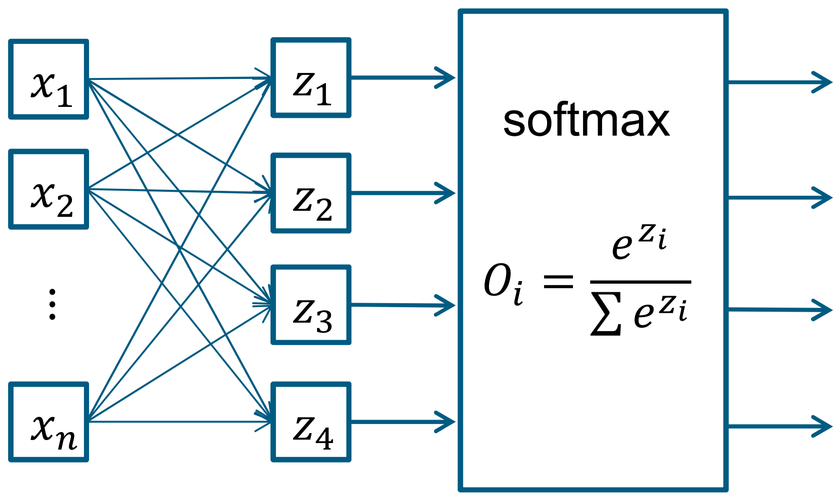
Probability that image includes object



Softmax function

$$F(x_i) = \frac{e^{z_i}}{\sum e^{z_i}}$$

probability prediction



Probability: O_i

Examples

2012: Alexnet

- Image size: 224x224x3
- 8 layers with 60 million parameters
  - 5 convolutional layers, 3 max-pooling, 3 fully-connected layers
- Activation function ReLU for all
- Output: 1000-classes softmax layer

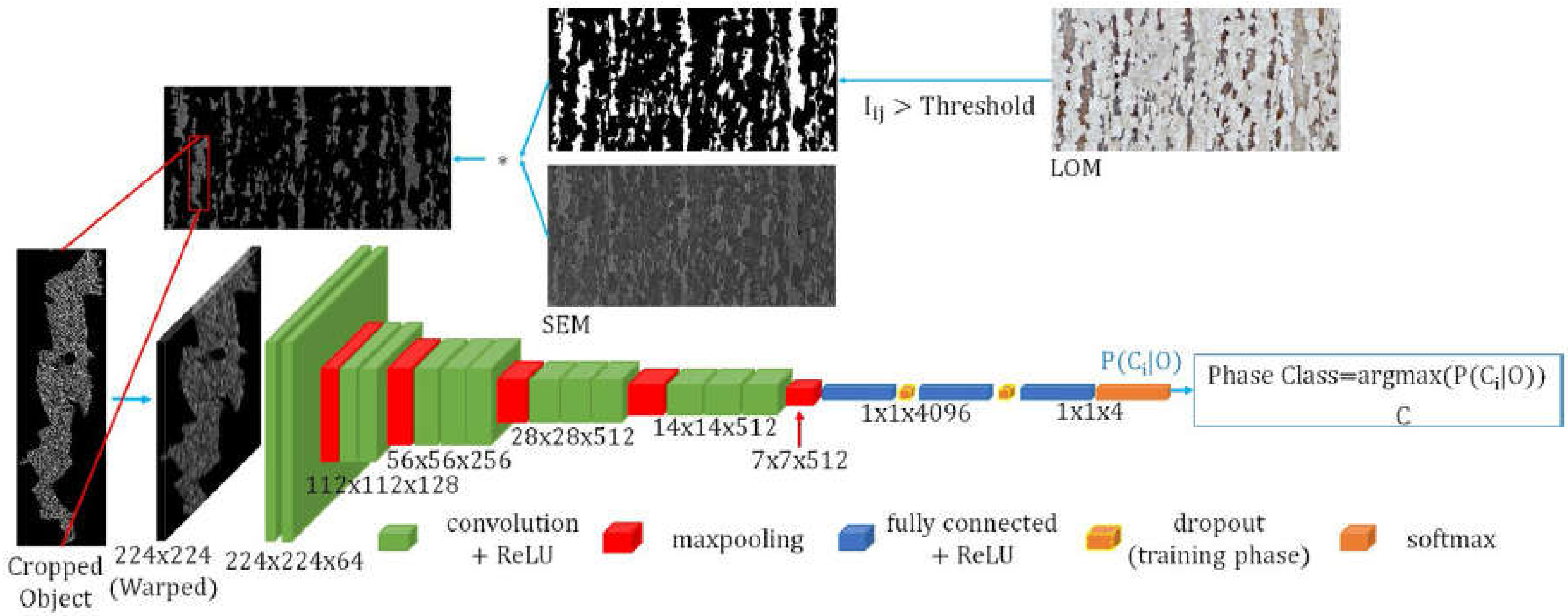
2014: *VGGNet*

- 19 layers with 138 million parameters
  - 13 convolutional layers, 5 max-pooling, 3 fully-connected layers
- 3x3 convolutional filter

VGGnet: Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-scale Image Recognition. In International Conference on Learning Representations (ICLR) (2015).

Alexnet: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

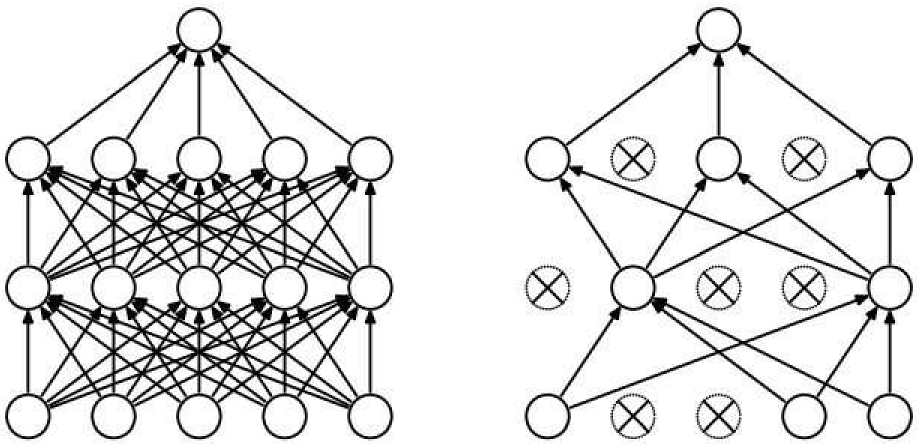
Example: Steel microstructure classification (Martensite, Bainite or Pearlite?)



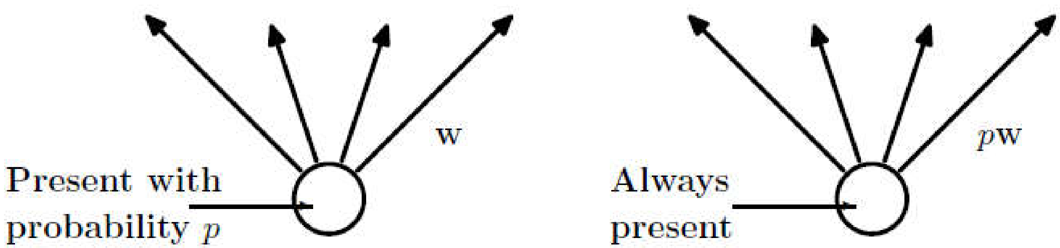
VGG16Net: 13 convolutional layers, 5 pooling layers and 3 fully connected layers
138 million parameters

[10] S. M. Azimi et al., Scientific Reports, **8**:2128, 2018.

Dropout layer



(a) Standard Neural Net (b) After applying dropout.



(a) At training time (b) At test time

Dropout layer has probability **p**

temporarily removing it along with connections

Prevent overfitting

N. Srivastava et al., Dropout: A simple Way to prevent neural network from overfitting, Journal of Machine Learning Research 15: 1929-1958, 2014.

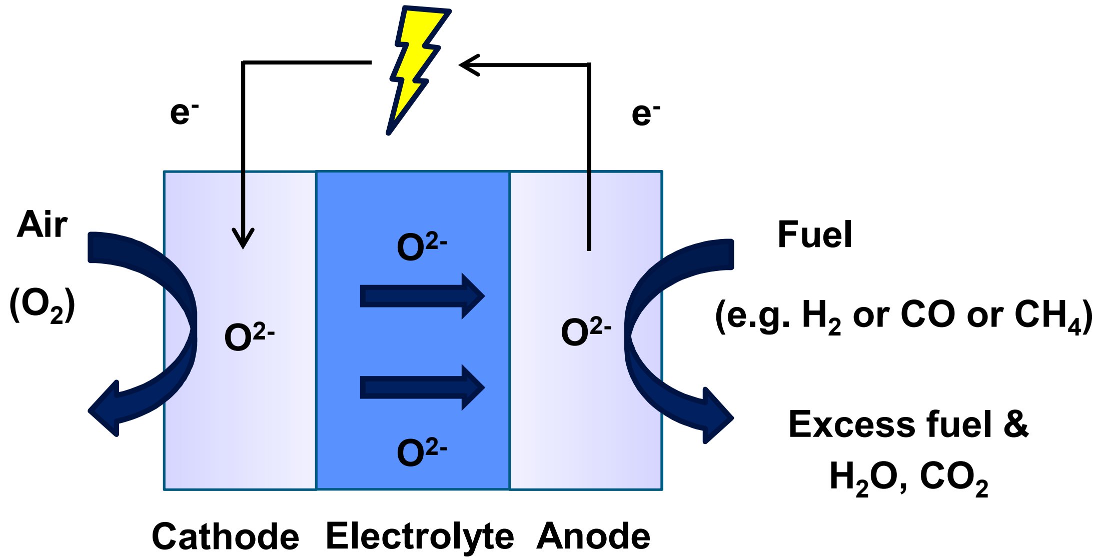
Convolutional Neural Networks

- Convolution & ReLu layers (stacked to recognize complex features)
- Pooling layers (size reduction, contrasting=max pooling)
- Softmax layer: categorization

Pros & Cons:

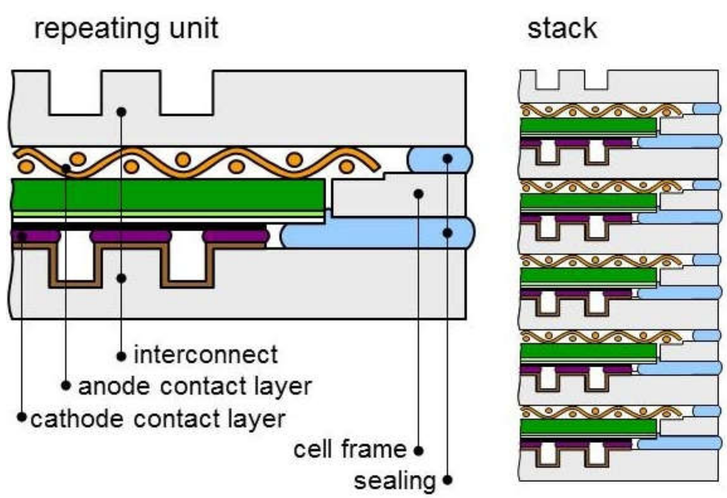
- No or Low-level preconditioning required
- Reach potentially high accuracies >99.9%
- Extremely memory and time consuming: e.g. 10^8 parameters

Our exercise: Solid oxide fuel cell (SOFC) and their stacks



Clean "green" energy:

produce energy from hydrogen or methanol



Use stacks of SOFC

increase voltage

use steel to connect

Our exercise: Solid oxide fuel cell (SOFC)

During operation:

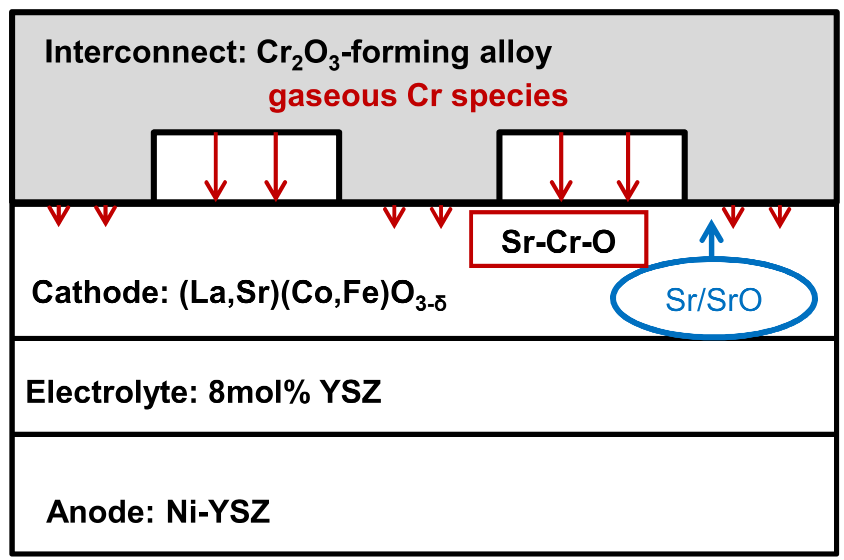
- Cr_2O_3 containing oxide scale forms on steel

  evaporation of gaseous Cr (CrO_3 or $CrO_2(OH)_2$)
- Sr is reactive element in the cathode

  Sr→SrO which segregates out of cathode
- Reaction of SrO & Cr species

  secondary phases that block oxygen reduction / functionality

$$SrO + CrO_3 \leftrightharpoons SrCrO_4$$
$$SrO + CrO_3 \leftrightharpoons SrCrO_3 + \frac{1}{2}O_2$$
$$SrO + \frac{2}{3}CrO_3 \leftrightharpoons \frac{1}{3}Sr_3Cr_2O_8 + \frac{1}{6}O_2$$
$$SrO + \frac{1}{2}CrO_3 \leftrightharpoons \frac{1}{2}Sr_2CrO_4 + \frac{1}{4}O_2$$



Our exercise: Solid oxide fuel cell (SOFC)

CSV file with data

Response/Target: chemical component

last column

Parameters first columns:

- Partial pressure of CrO_3 ($pCrO_3$)
- local oxygen partial pressure (pO_2)