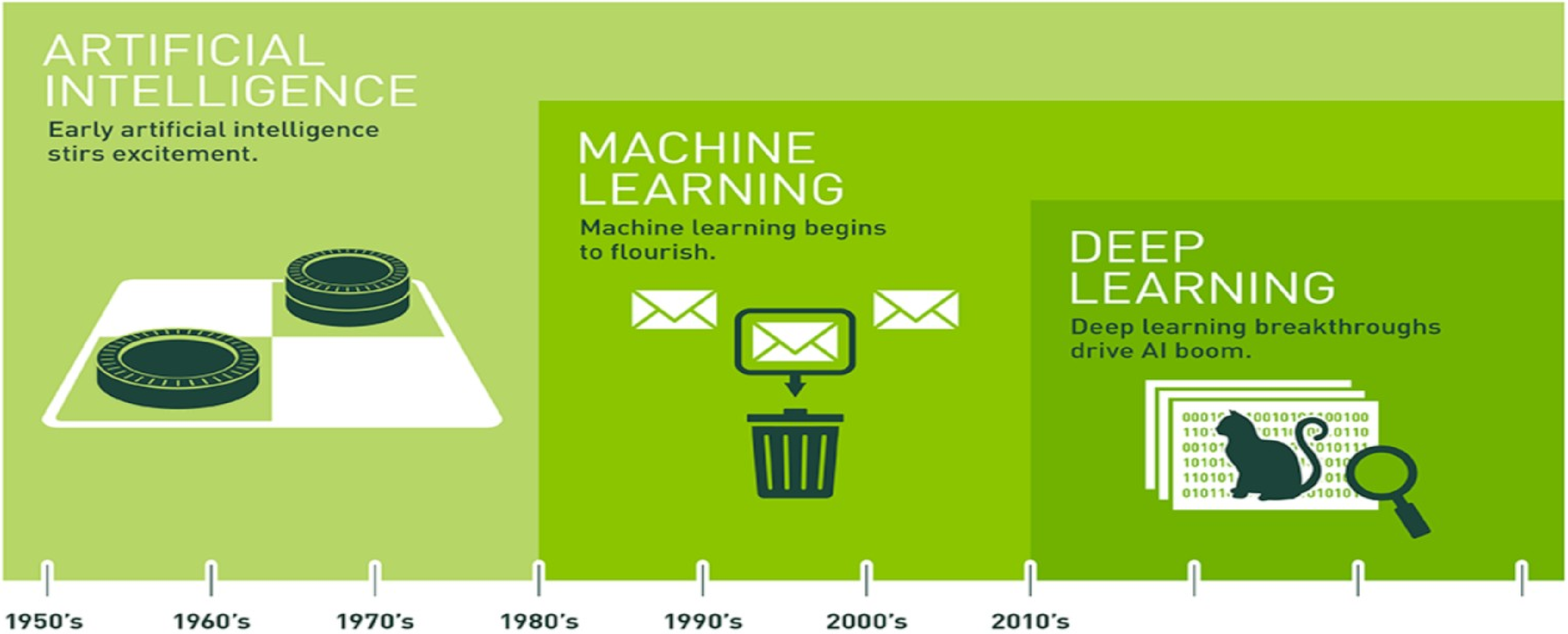
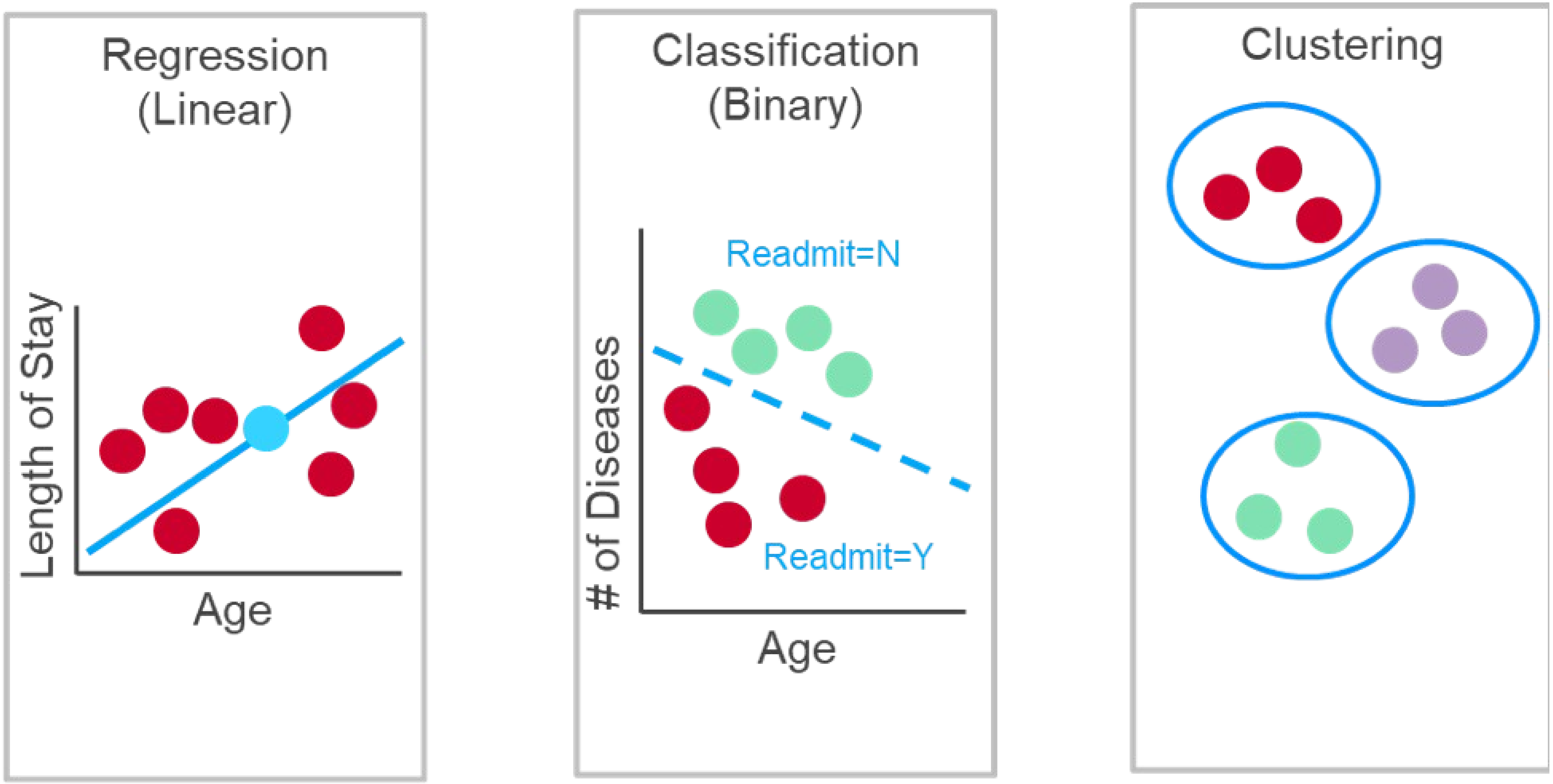
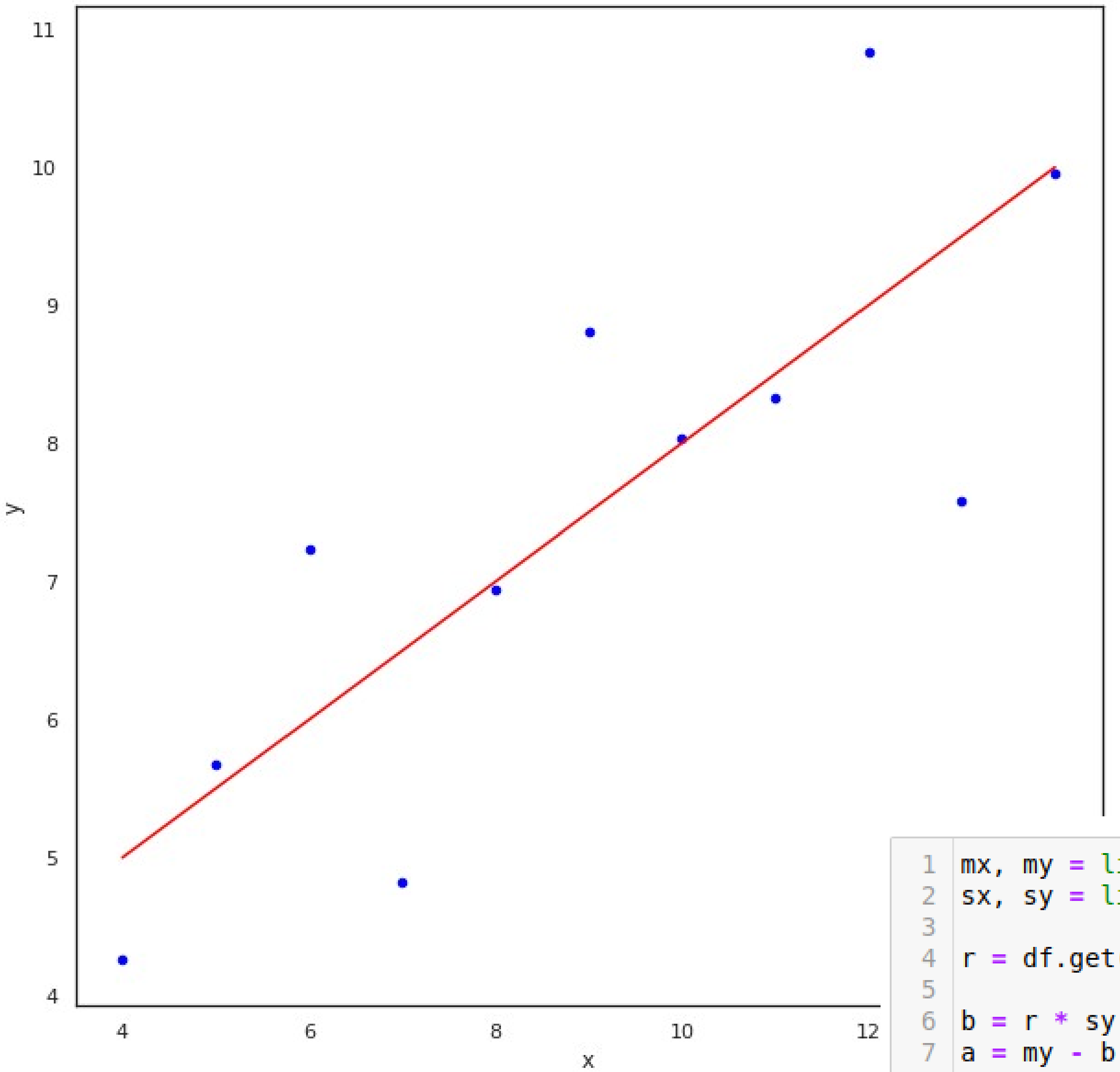
Aprendizaje automático


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Data analytics



Length of Stay | Readmit Yes/No | Diabetes Subgroups

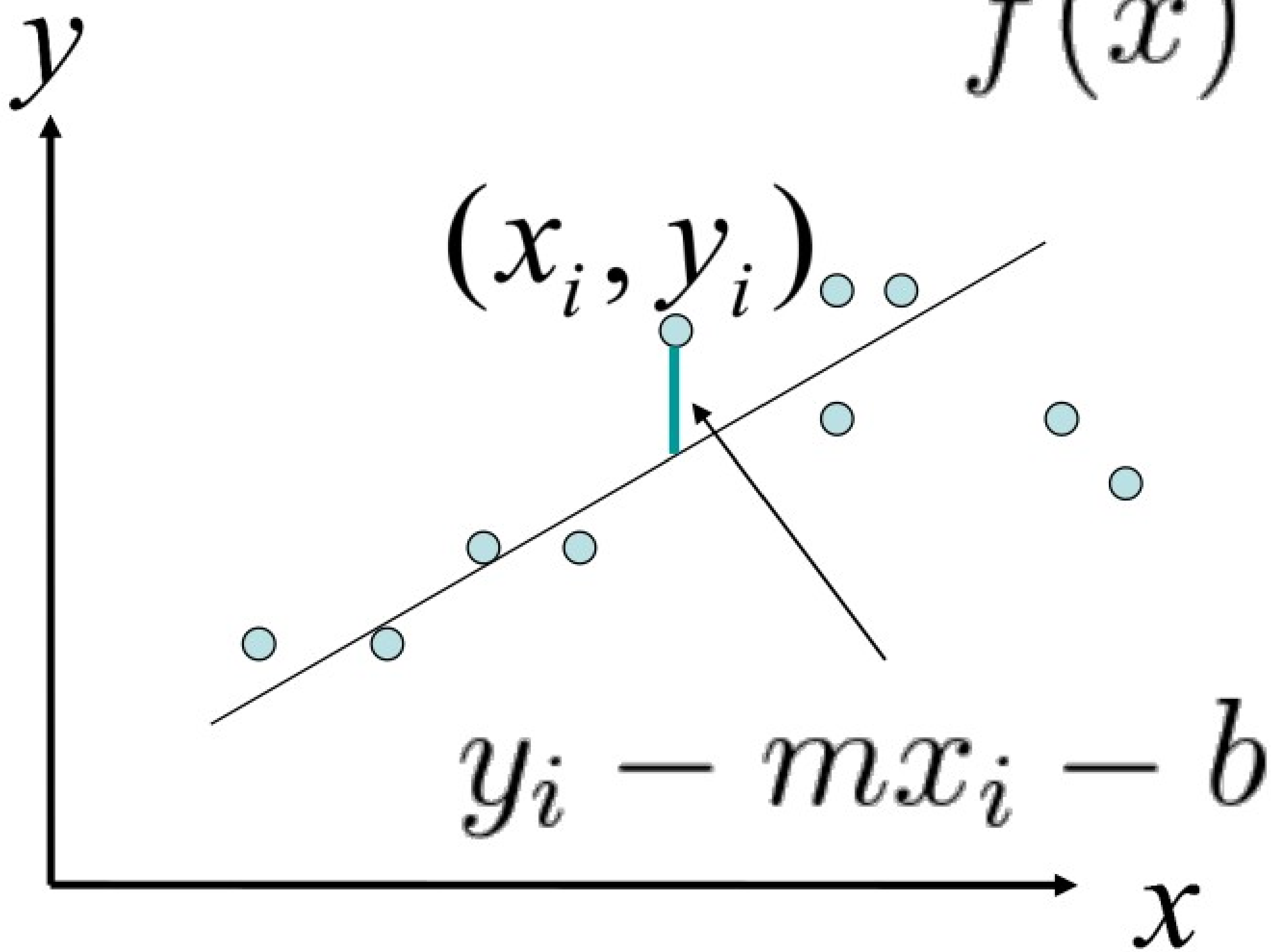



```
mx, my = list(df.get(df.dataset == 'I').mean())
sx, sy = list(df.get(df.dataset == 'I').std())

r = df.get(df.dataset == 'I')[['x','y']].corr().iloc[0,1]

b = r * sy / sx
a = my - b * mx
```

```
df.get(df.dataset == 'I').plot.scatter('x','y', color='blue')
plt.plot([4,14],[a+b*4, a+b*14], color='red')

```

Residuals

$$f(x) = mx + b$$



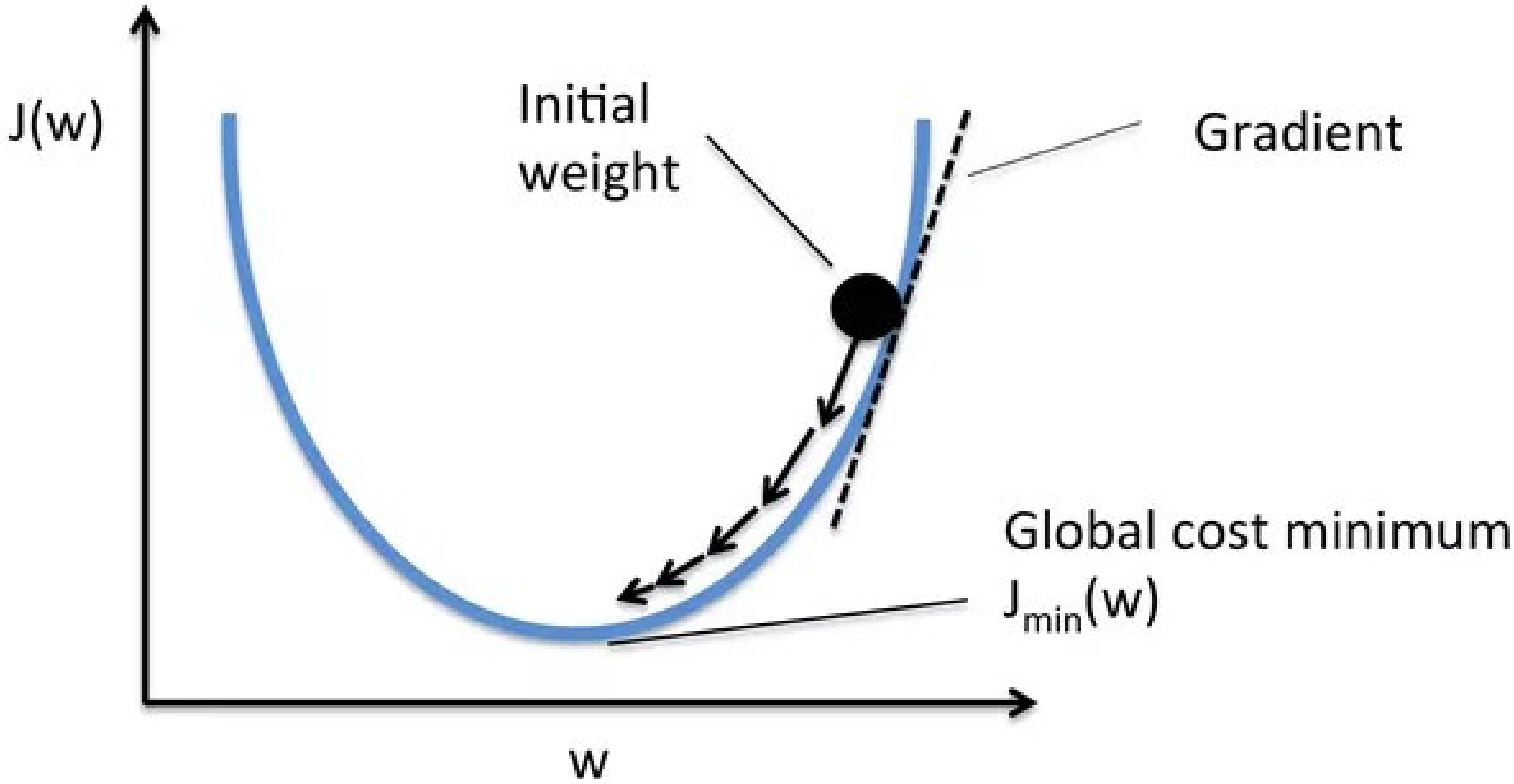
MSE

Error (also known as loss) is a method of evaluating how well a specific algorithm models the given data.

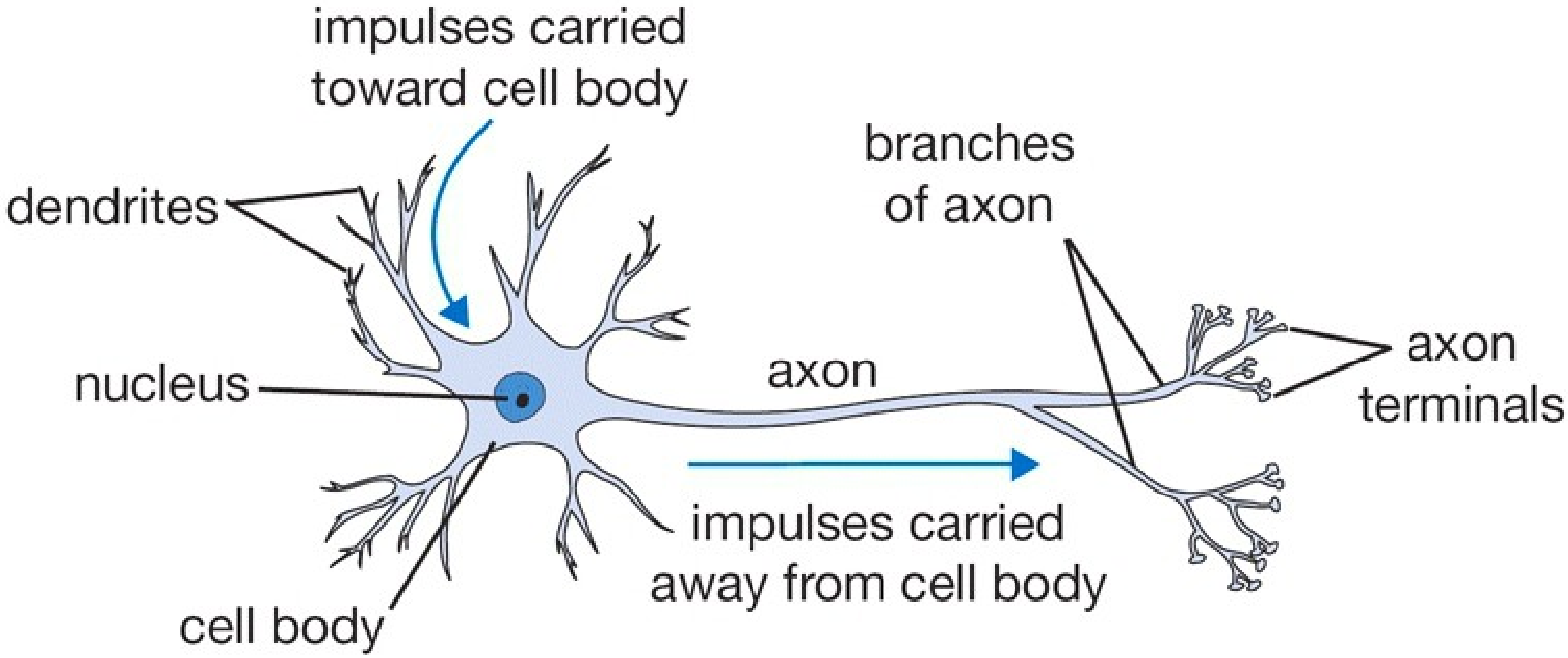
- If predictions deviates too much from actual results, error would be high.
- A popular error used a lot in CV and statistics is called MSE (mean square error, also known as L2 loss or quadratic loss).

$$e_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

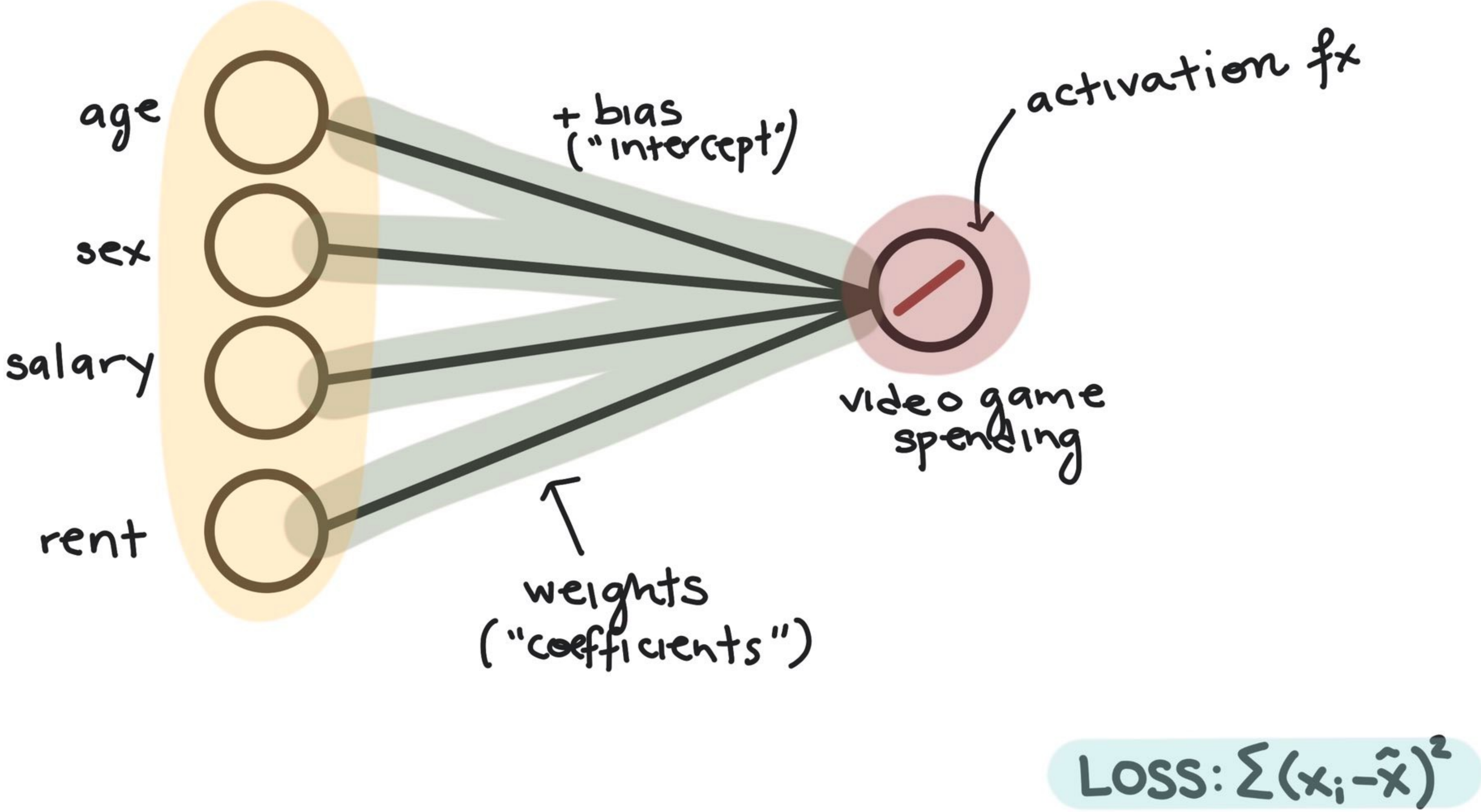
Cost optimization



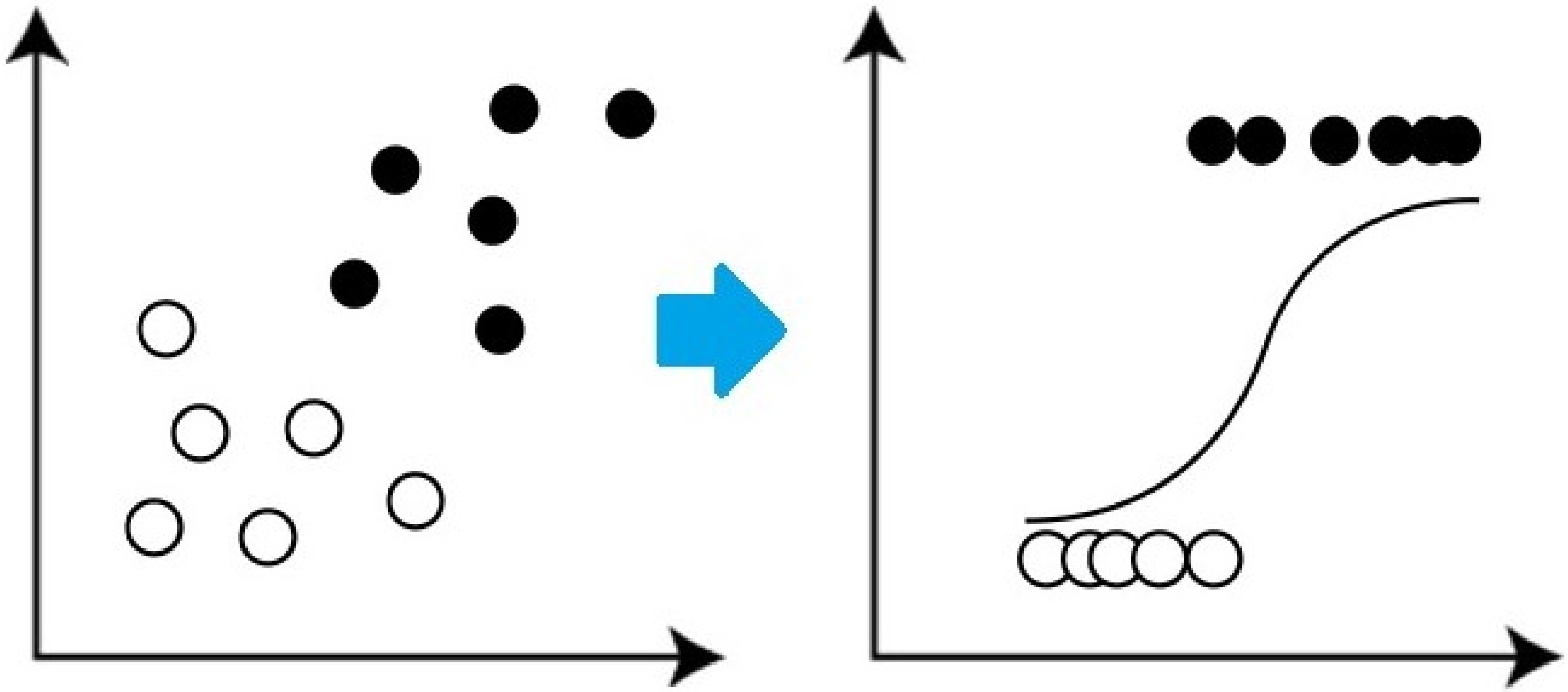
Neural networks



LINEAR REGRESSION
(as a neural network)

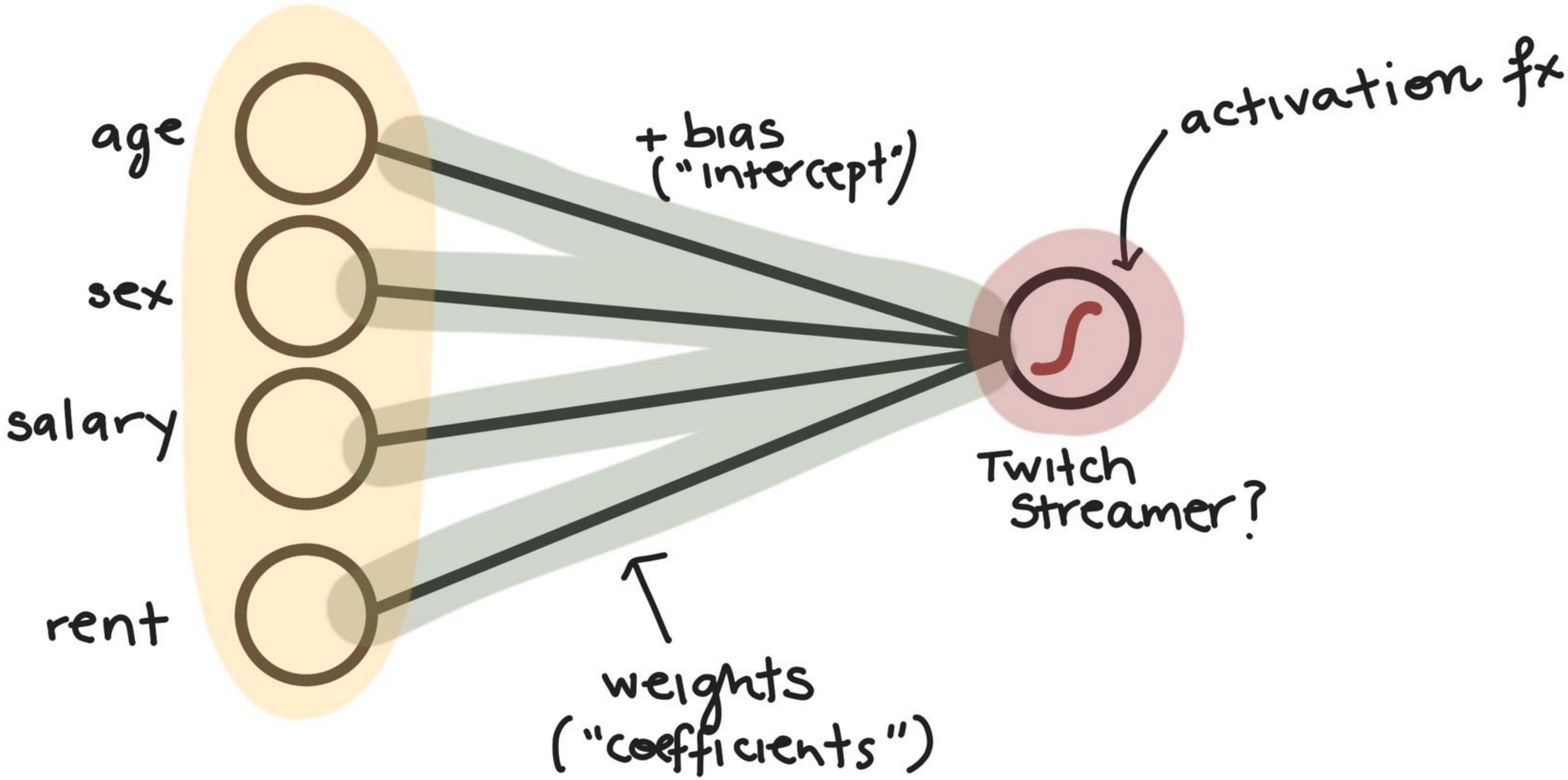


Classification as regression



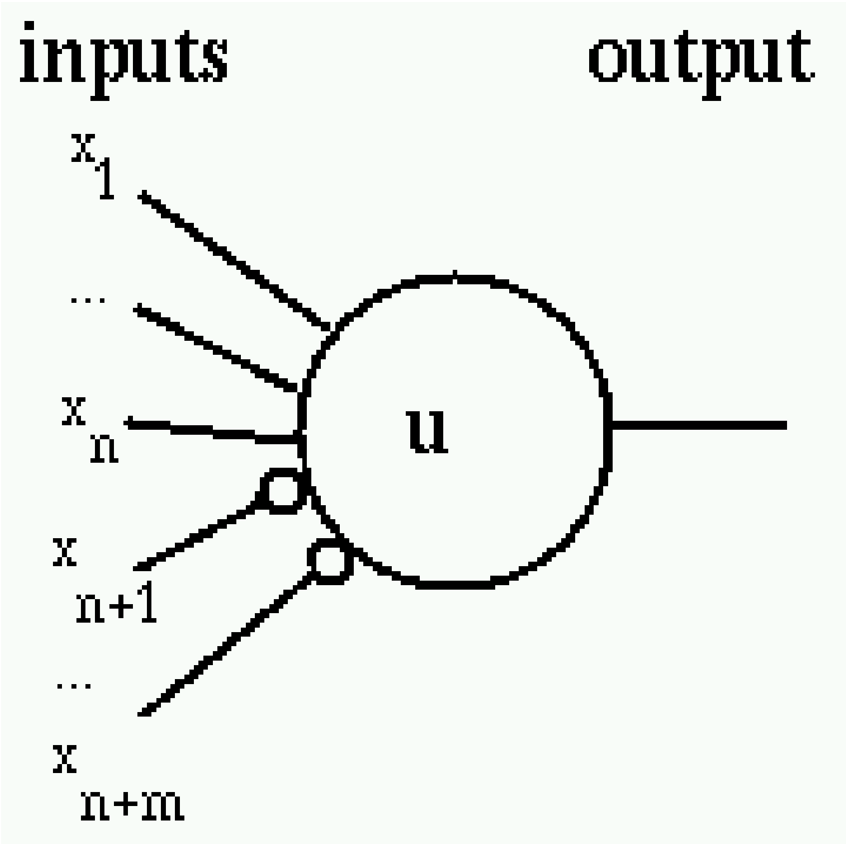
LOGISTIC REGRESSION

(as a neural network)



LOSS: $\sum -y_i \log(\hat{p}_i) - (1-y_i)\log(1-\hat{p}_i)$

Neural networks



Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092–8240/90$3.00+0.00
Pergamon Press plc
Society for Mathematical Biology

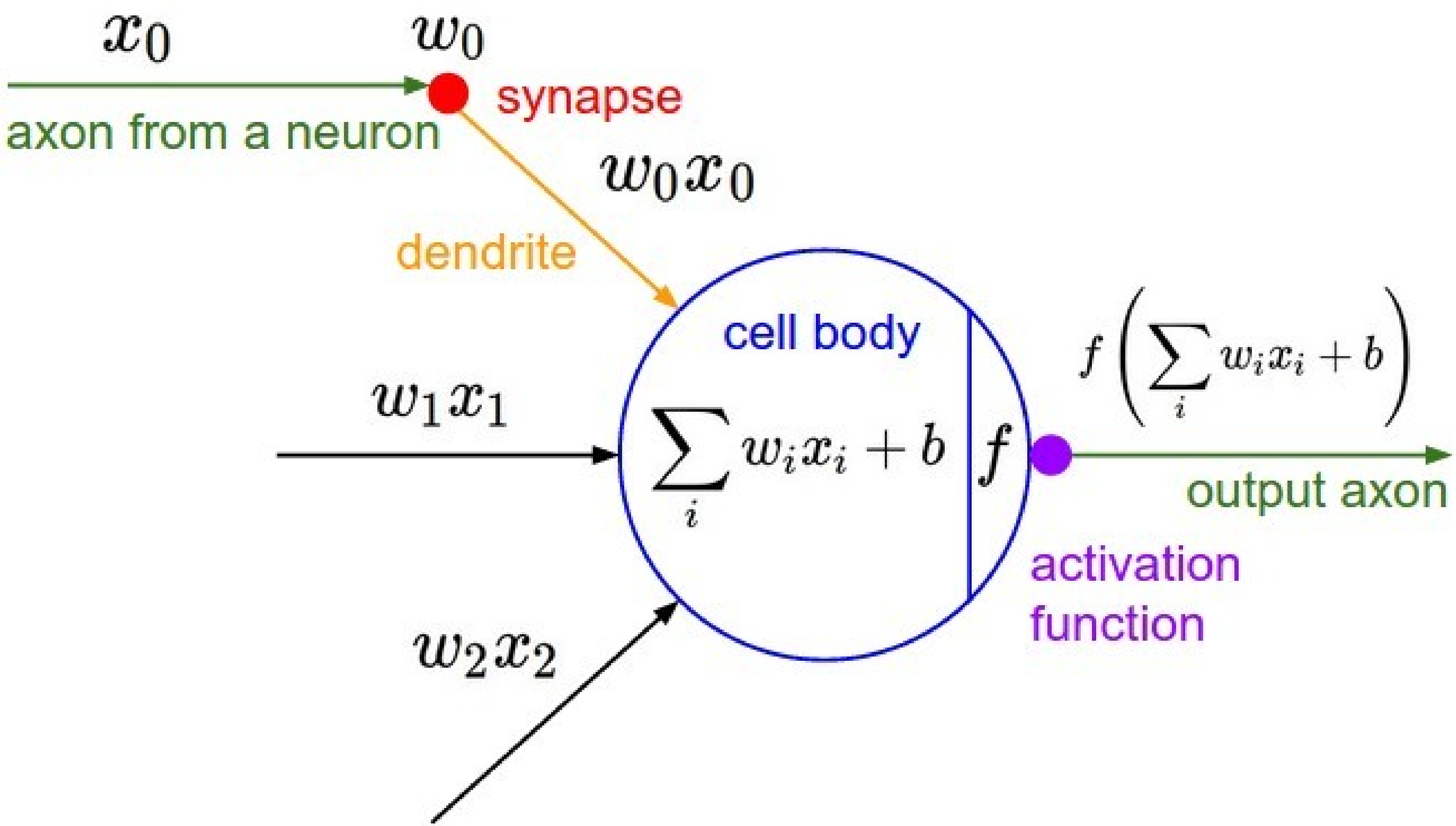
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

1. Introduction. Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from $<1\ \mathrm{ms}^{-1}$ in thin axons, which are usually short, to $>150\ \mathrm{ms}^{-1}$ in thick axons, which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequally remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this depends upon irreciprocity of individual synapses or merely upon prevalent anatomical configurations. To suppose the latter requires no hypothesis *ad hoc* and explains known exceptions, but any assumption as to cause is compatible

Neural networks

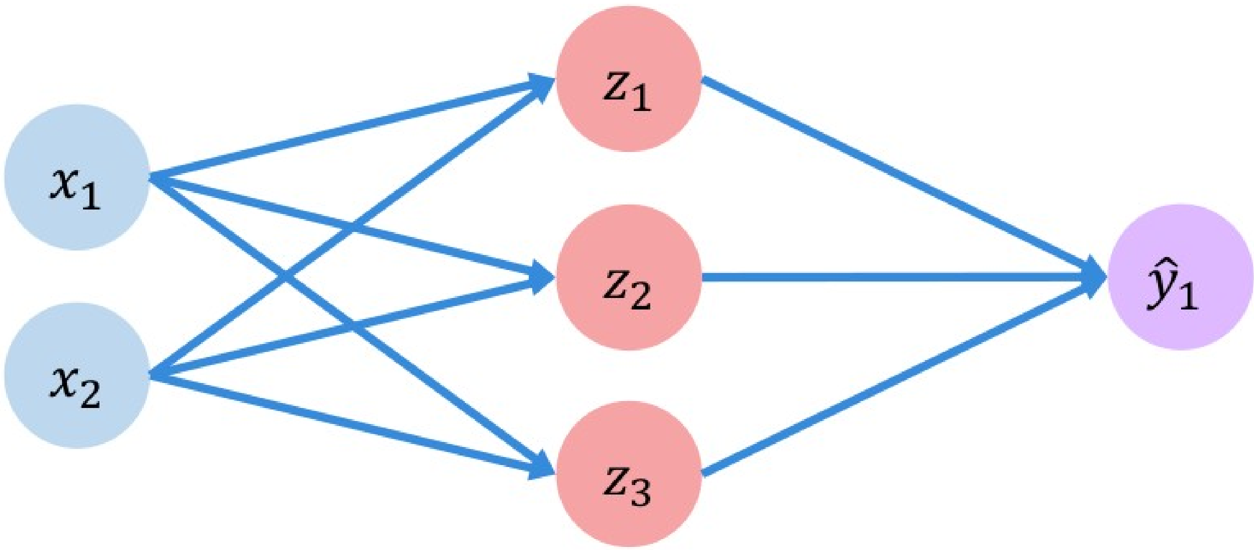


Neural networks

```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation functior
    return firing_rate
```

Neural networks

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, \boldsymbol{J}(\boldsymbol{W})$$

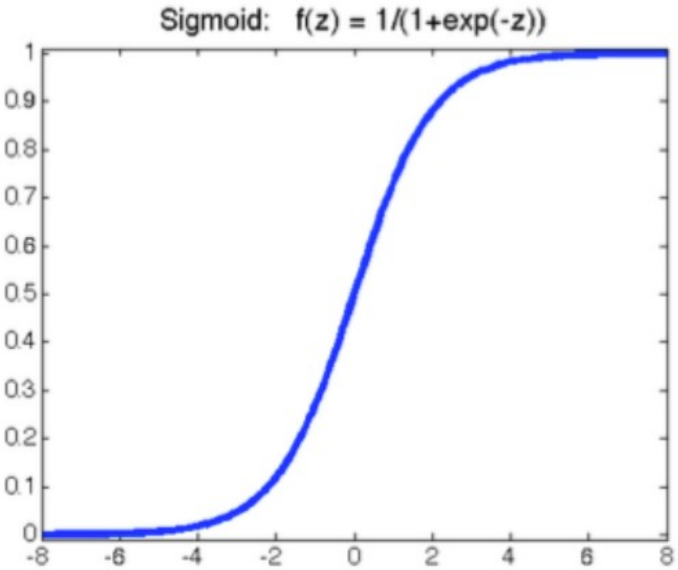
Neural networks

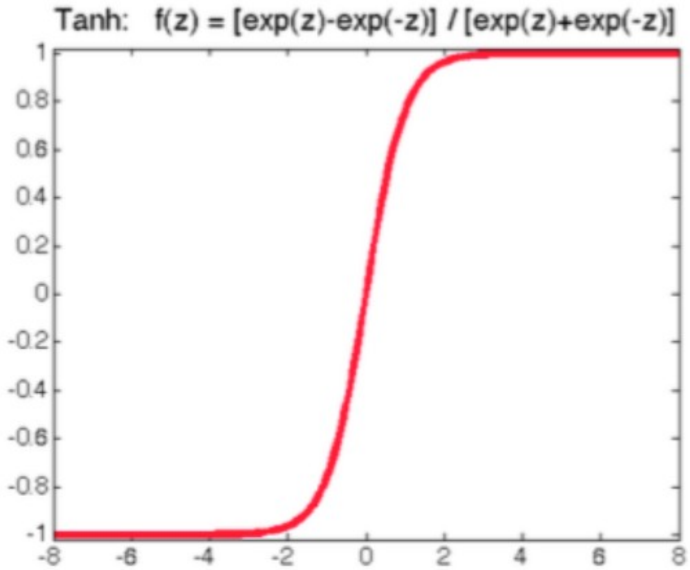
```python
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation functior
    return firing_rate
```

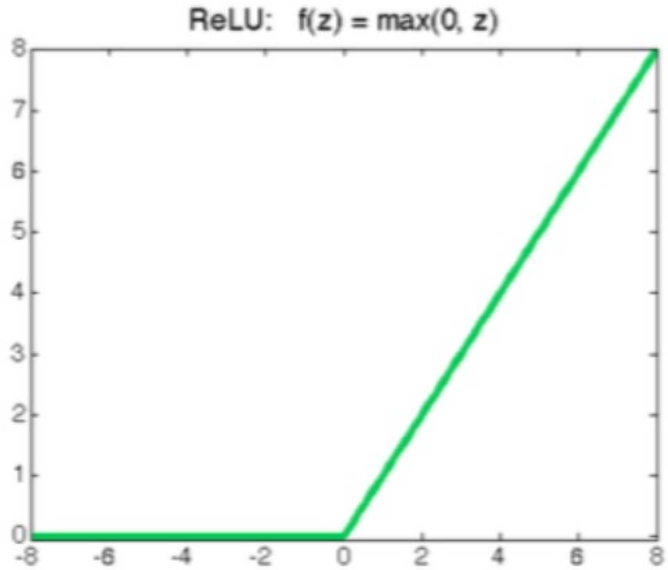
```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Activation function

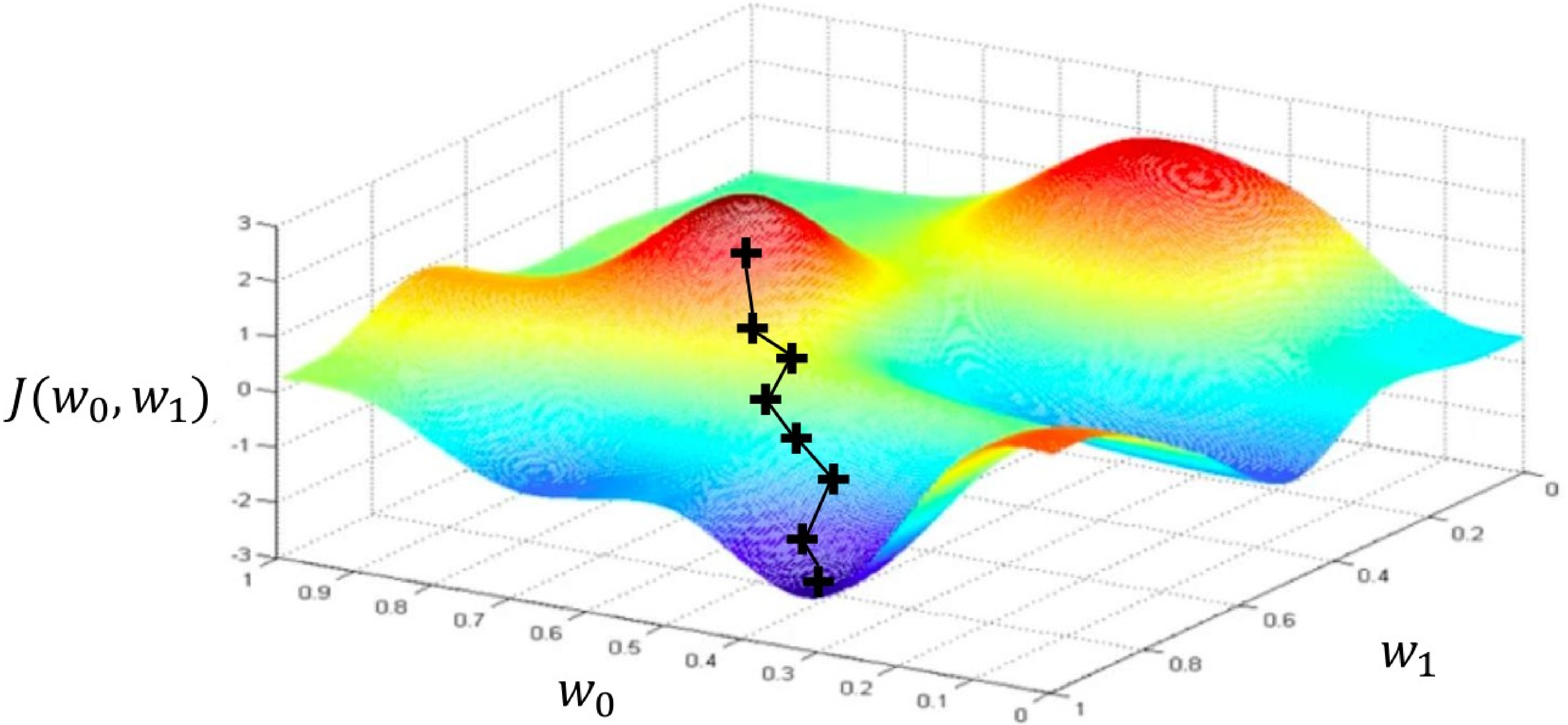
- **Sigmoid:** $f(x) = 1 / (1+e^{-x})$
- **Tanh:** $f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$
- **ReLU (Rectified Linear Unit):** $f(x) = max(0, x)$







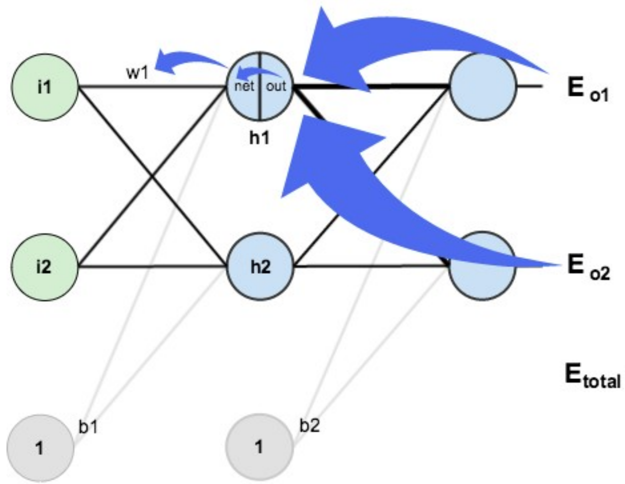
Gradient descent



Backpropagation

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

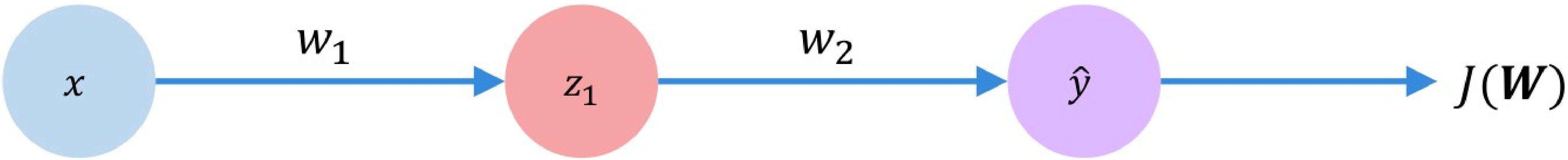
The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

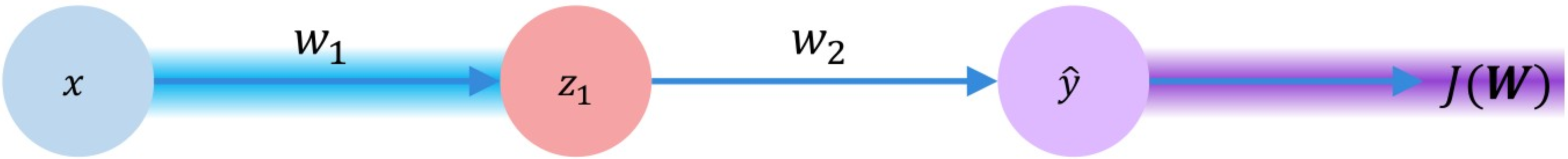
Units can be given biases by introducing an extra input to each

Backpropagation



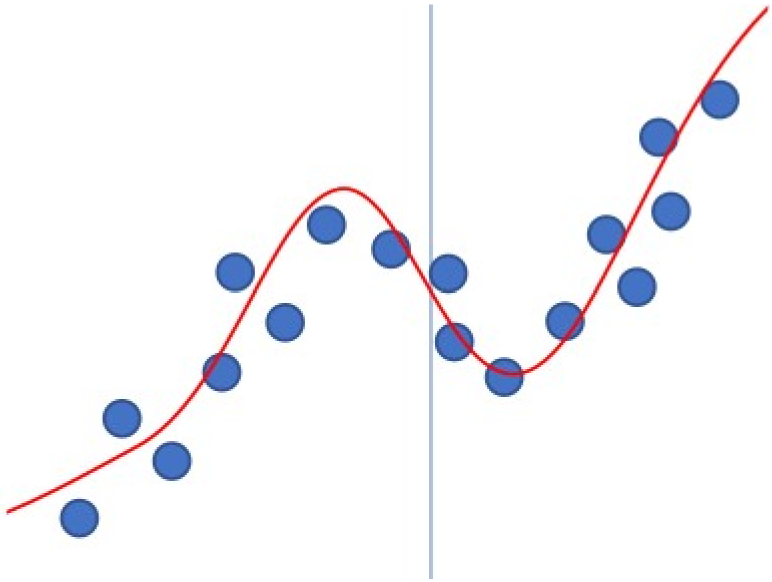
$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Backpropagation



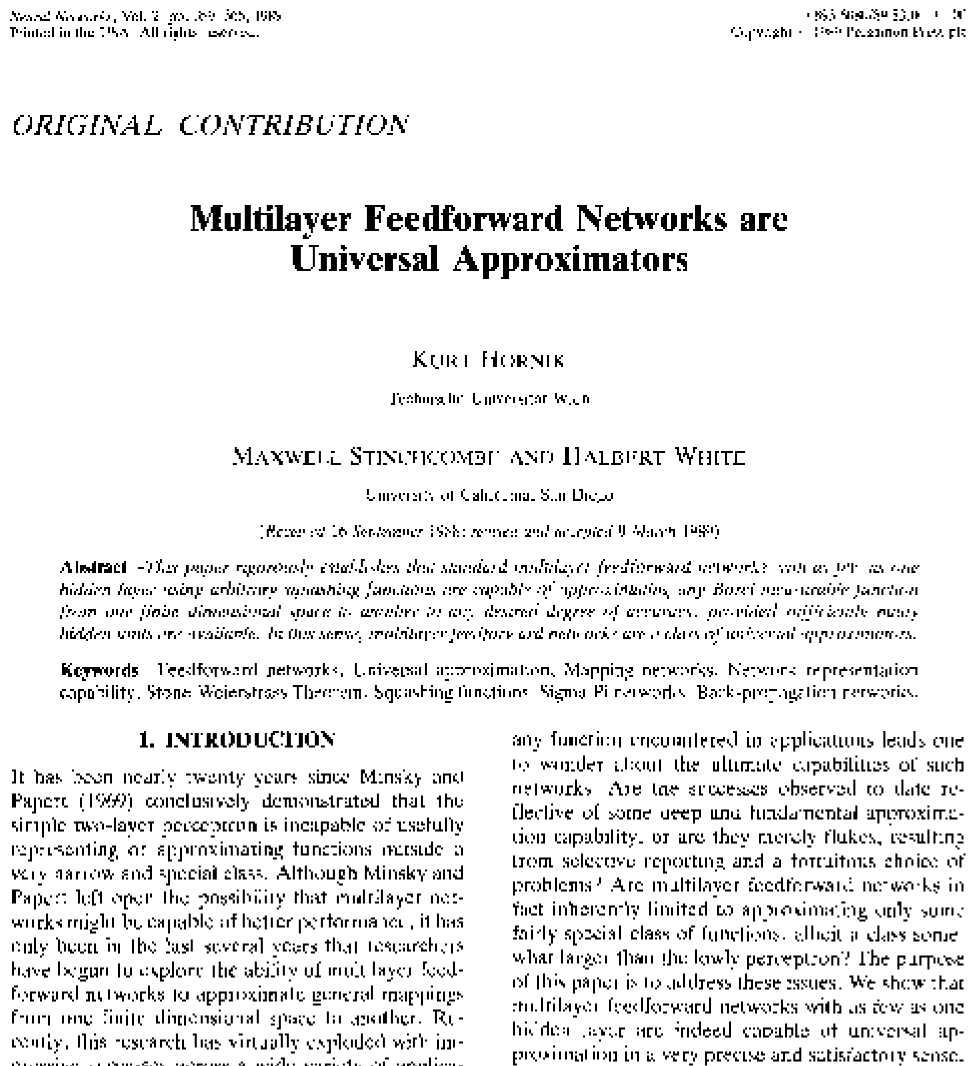
$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Universal Approximation Theorem



A feedforward network with a single layer is sufficien
approximate, to an arbitrary precision, any continuou
function.
- The resulting model may not generalize
- The number of hidden units may be infeasibly large

Hornik et al. Neural Networks. (1989)



Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK
Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE
University of California, San Diego

(Received 16 September 1988; revised and accepted 9 March 1989)

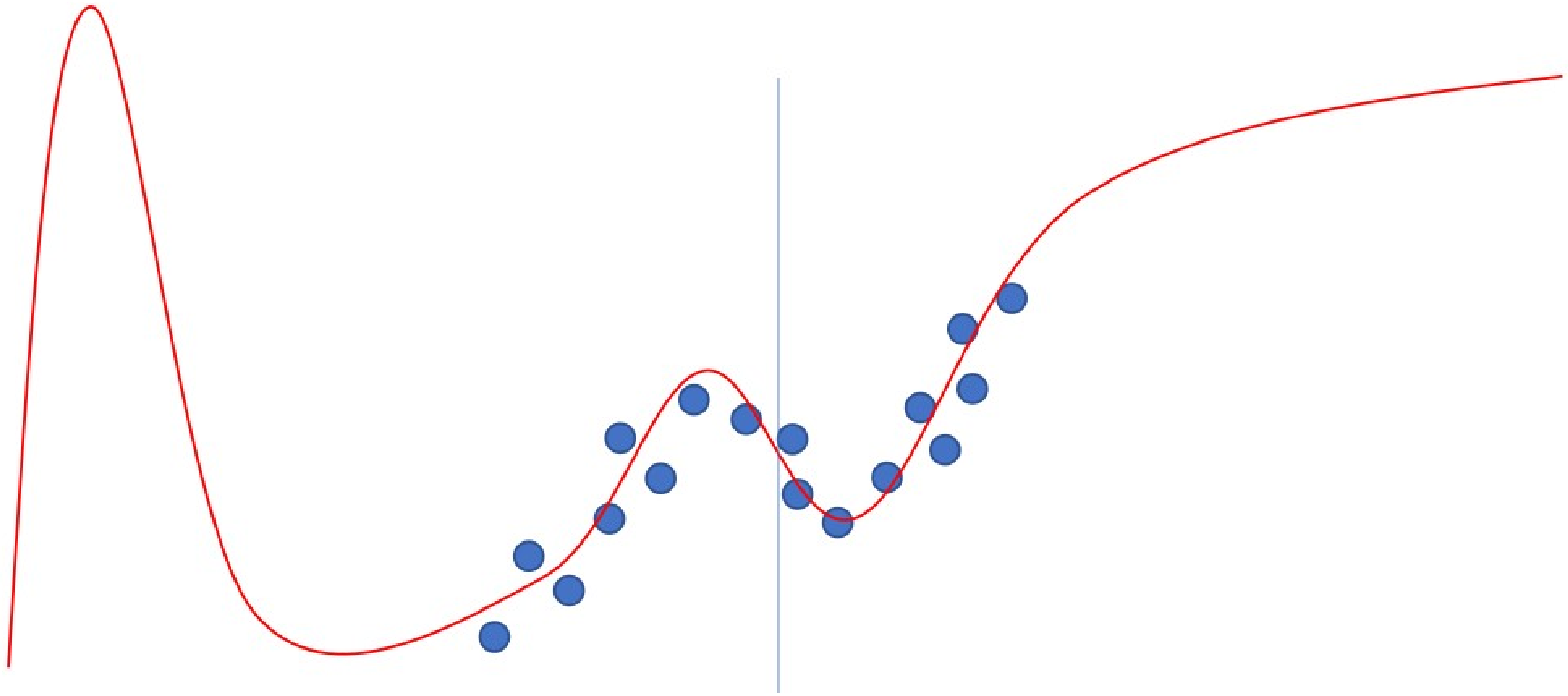
Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

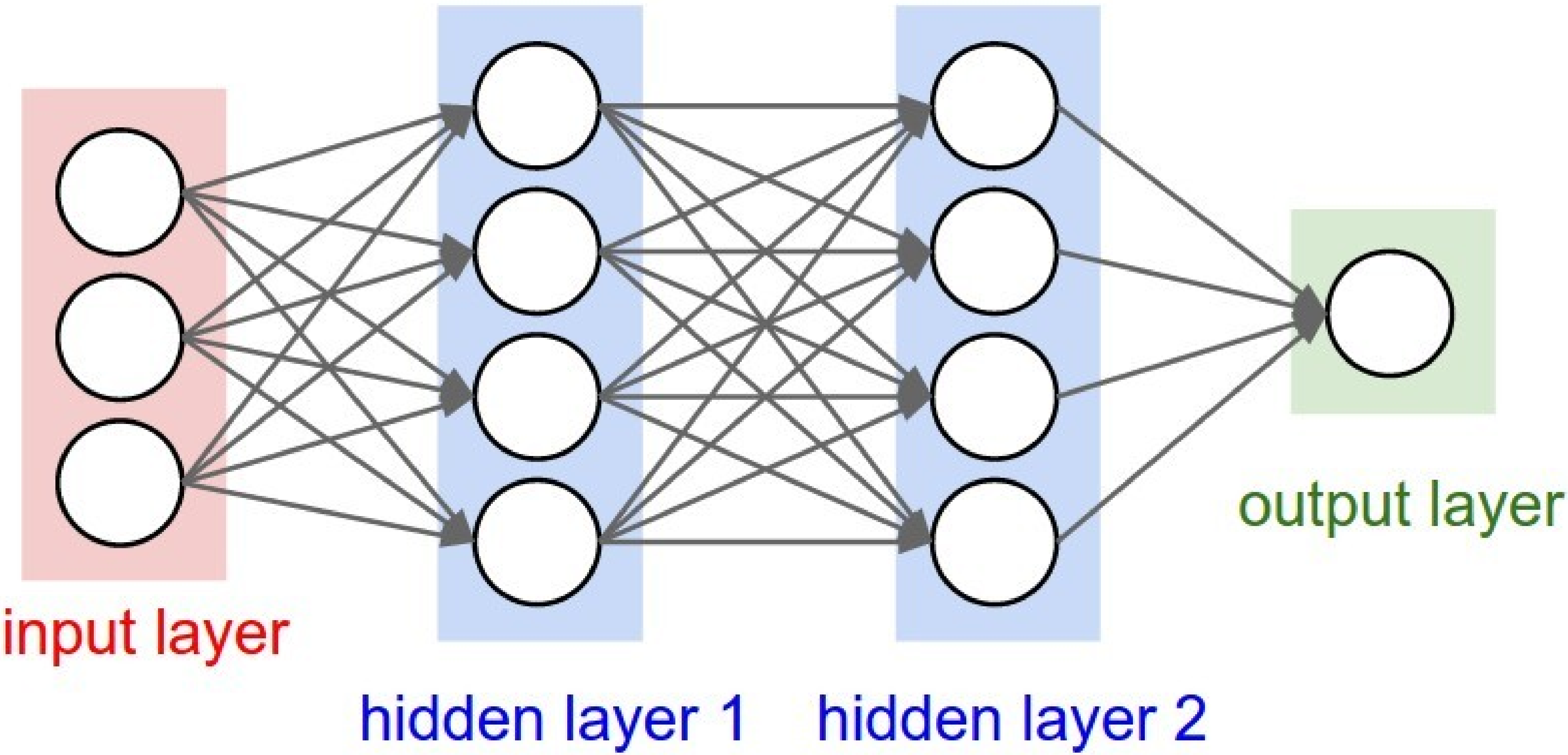
1. INTRODUCTION

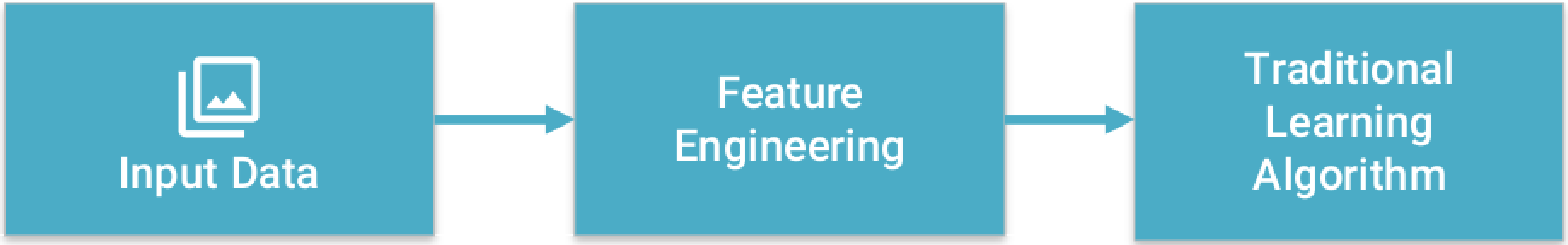
It has been nearly twenty years since Minsky and Papert (1969) conclusively demonstrated that the simple two-layer perceptron is incapable of usefully representing or approximating functions outside a very narrow and special class. Although Minsky and Papert left open the possibility that multilayer networks might be capable of better performance, it has only been in the last several years that researchers have begun to explore the ability of multilayer feedforward networks to approximate general mappings from one finite dimensional space to another. Recently, this research has virtually exploded with im-

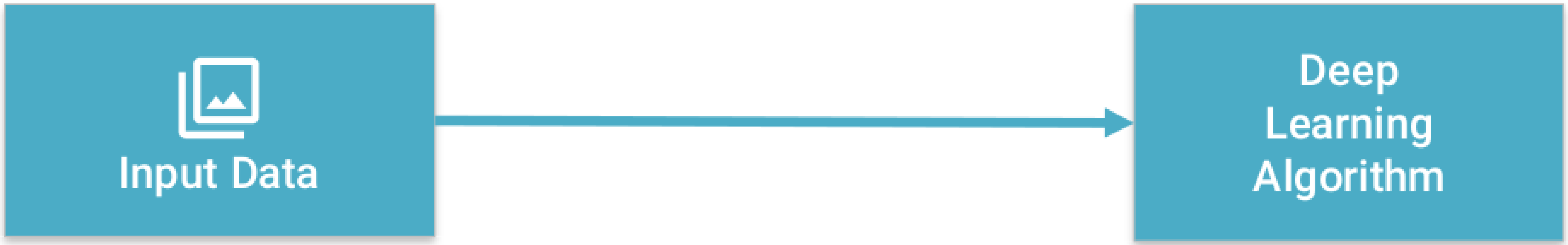
any function encountered in applications leads one to wonder about the ultimate capabilities of such networks. Are the successes observed to date reflective of some deep and fundamental approximation capability, or are they merely flukes, resulting from selective reporting and a fortuitous choice of problems? Are multilayer feedforward networks in fact inherently limited to approximating only some fairly special class of functions, albeit a class somewhat larger than the lowly perceptron? The purpose of this paper is to address these issues. We show that multilayer feedforward networks with as few as one hidden layer are indeed capable of universal approximation in a very precise and satisfactory sense.

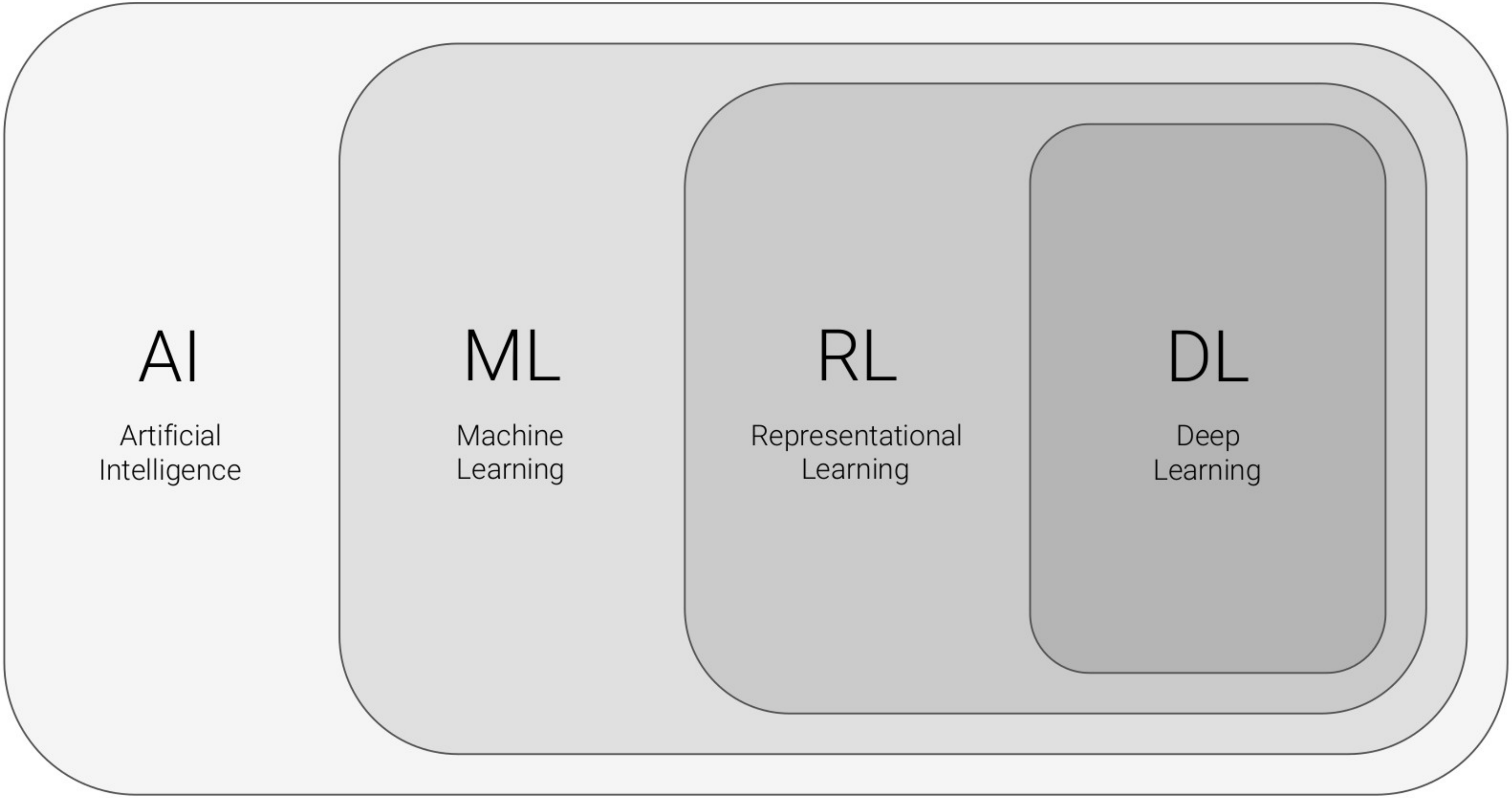


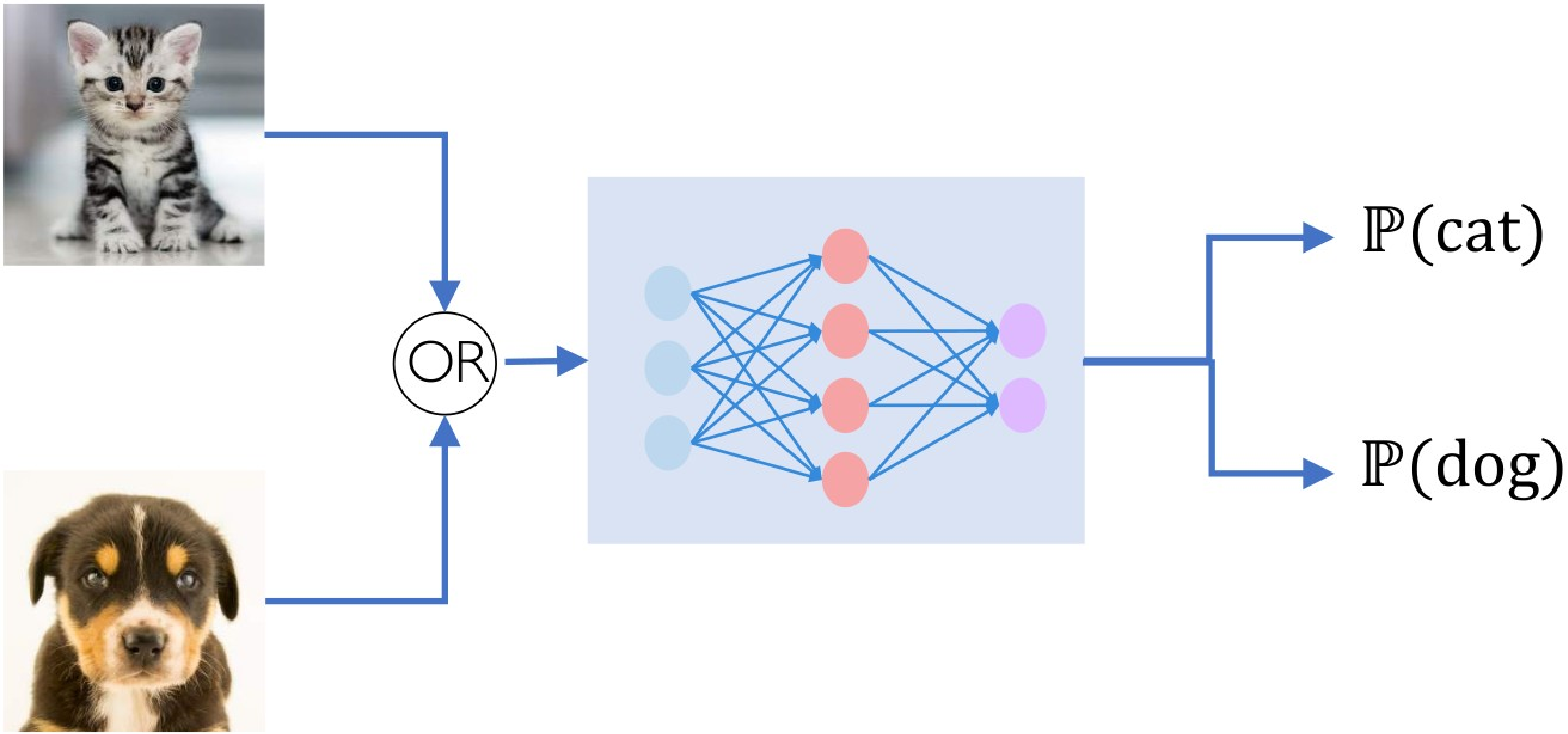
Neural networks

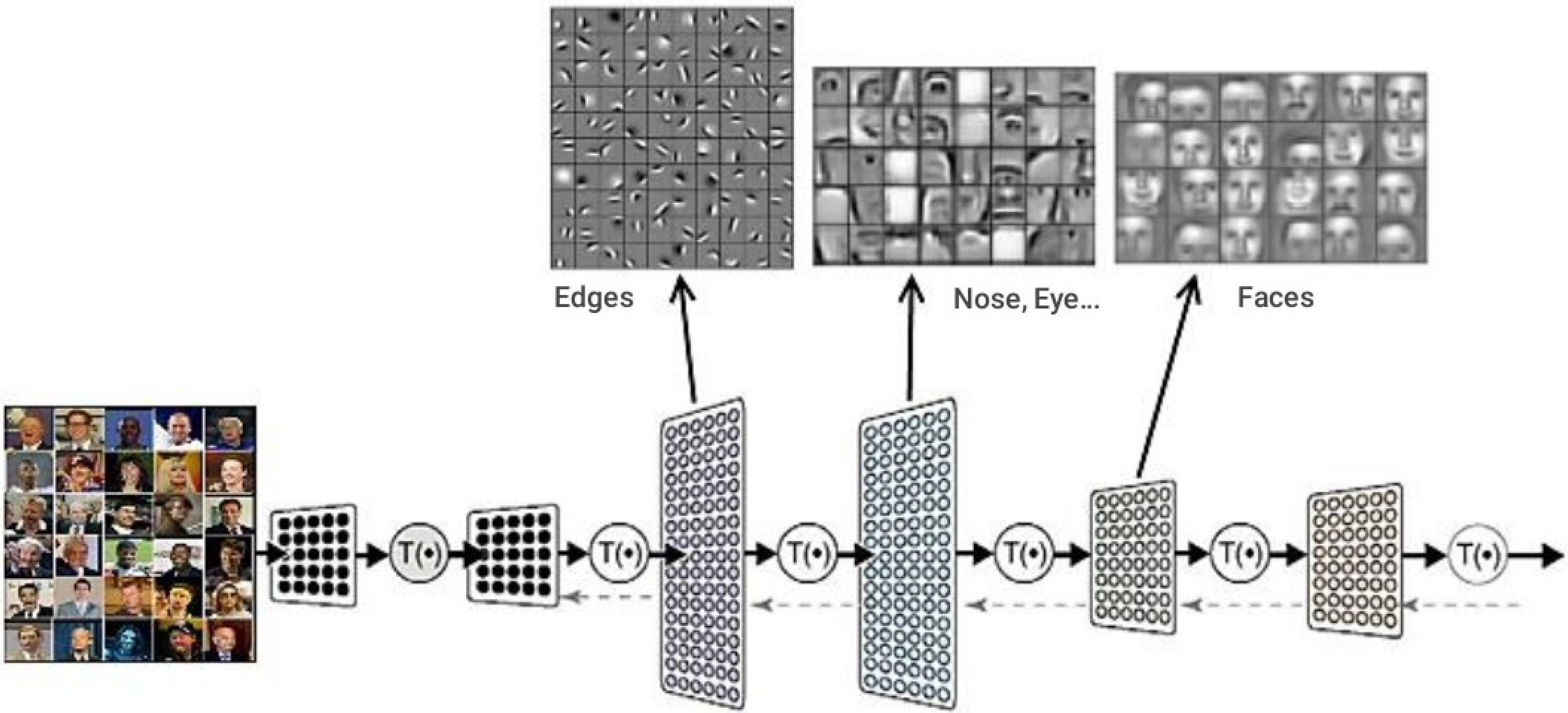



Input Data
Feature Engineering
Traditional Learning Algorithm


Input Data
Deep Learning Algorithm

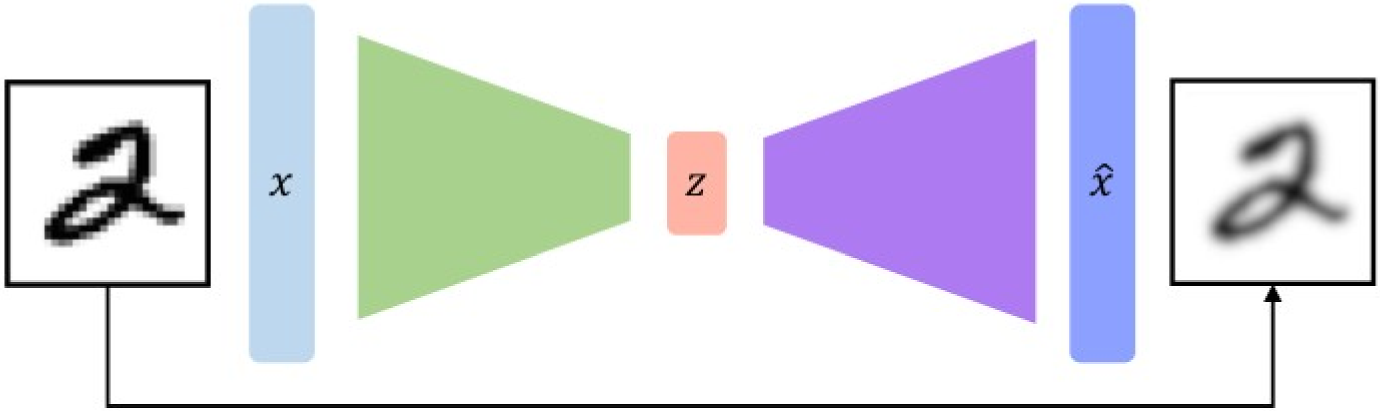

AI
Artificial Intelligence
ML
Machine Learning
RL
Representational Learning
DL
Deep Learning


OR
ℙ(cat)
ℙ(dog)



Edges
Nose, Eye...
Faces
T(•)
T(•)
T(•)
T(•)
T(•)
T(•)

Autoencoders

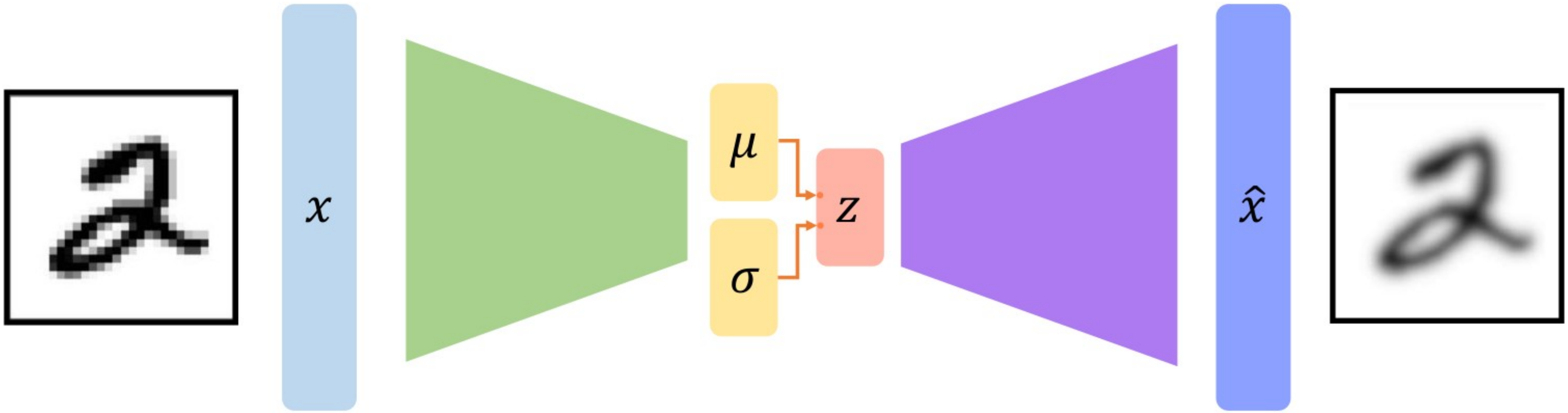


$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function doesn't use any labels!!

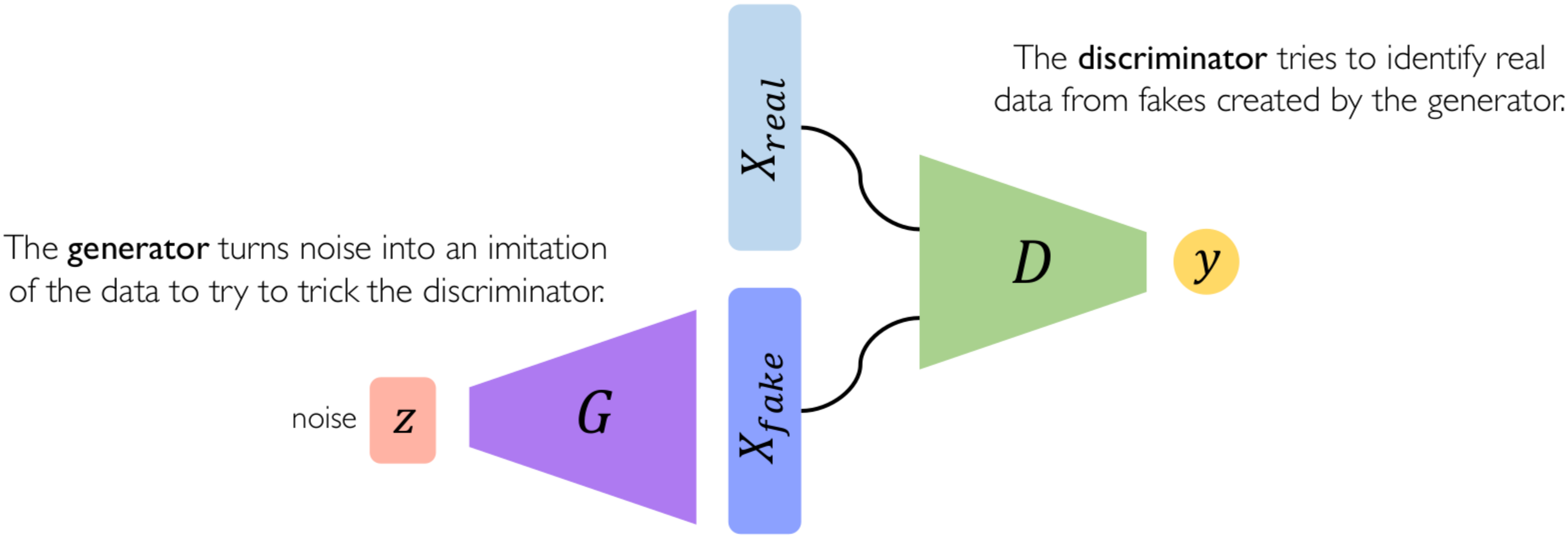
Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data

Variational autoencoder

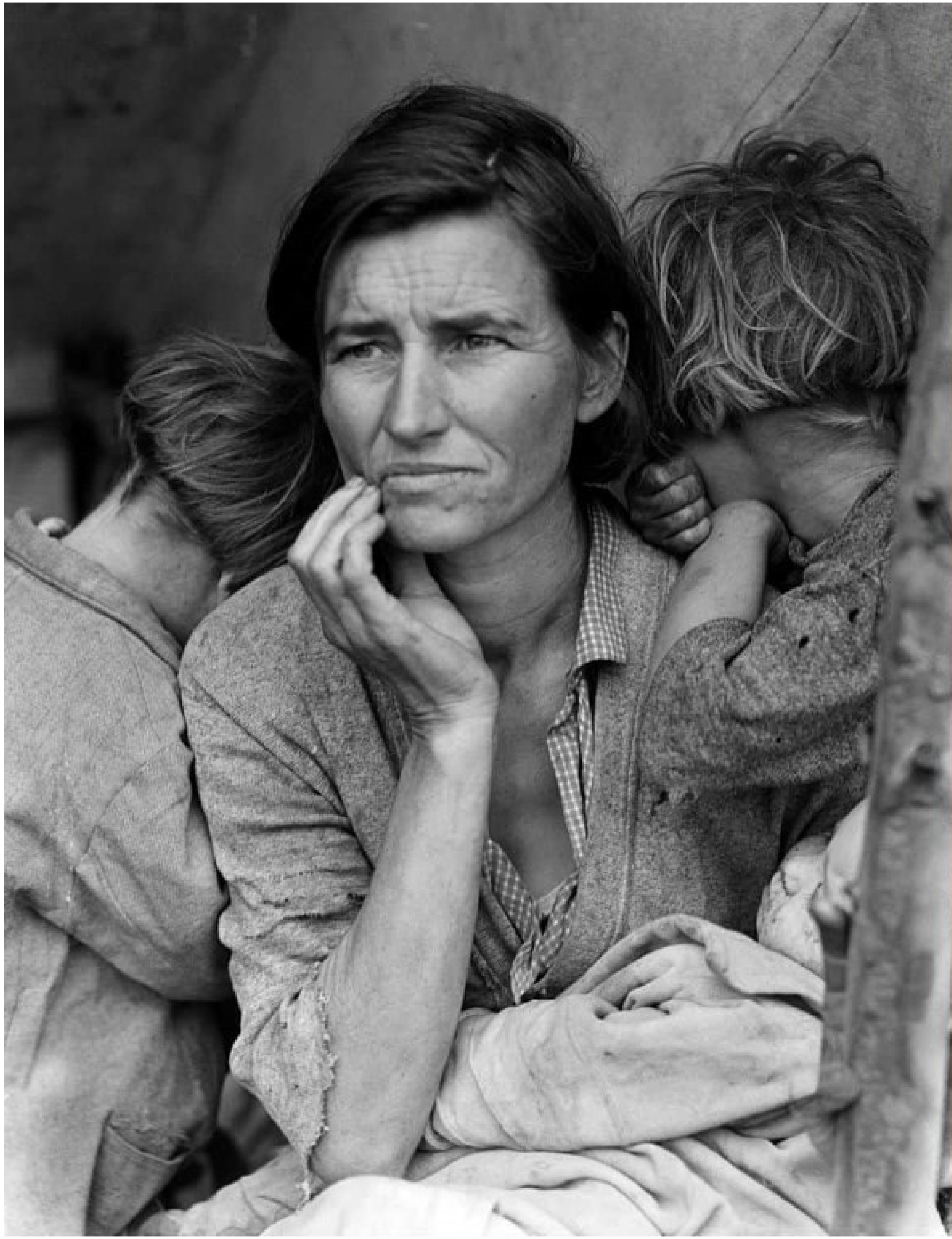
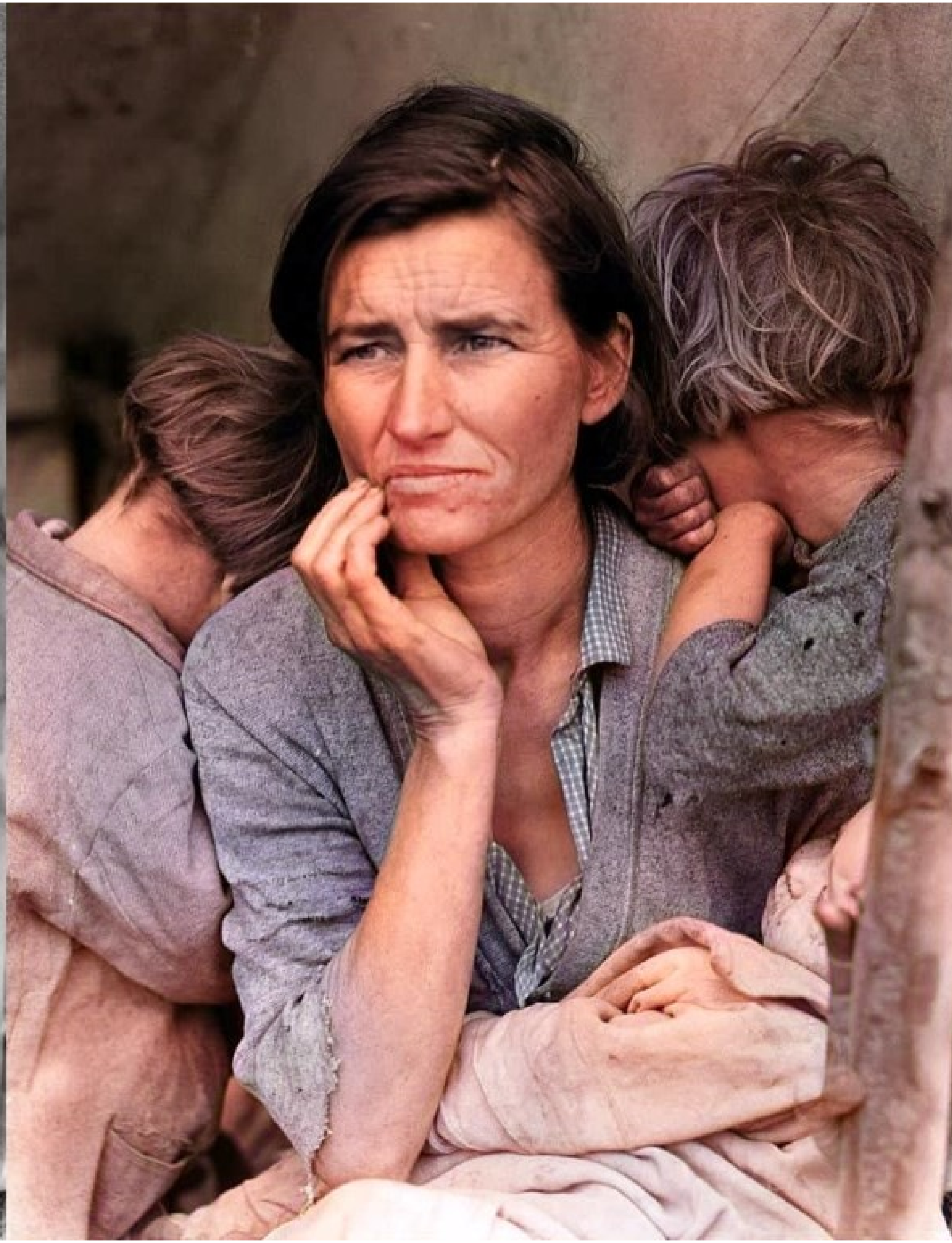


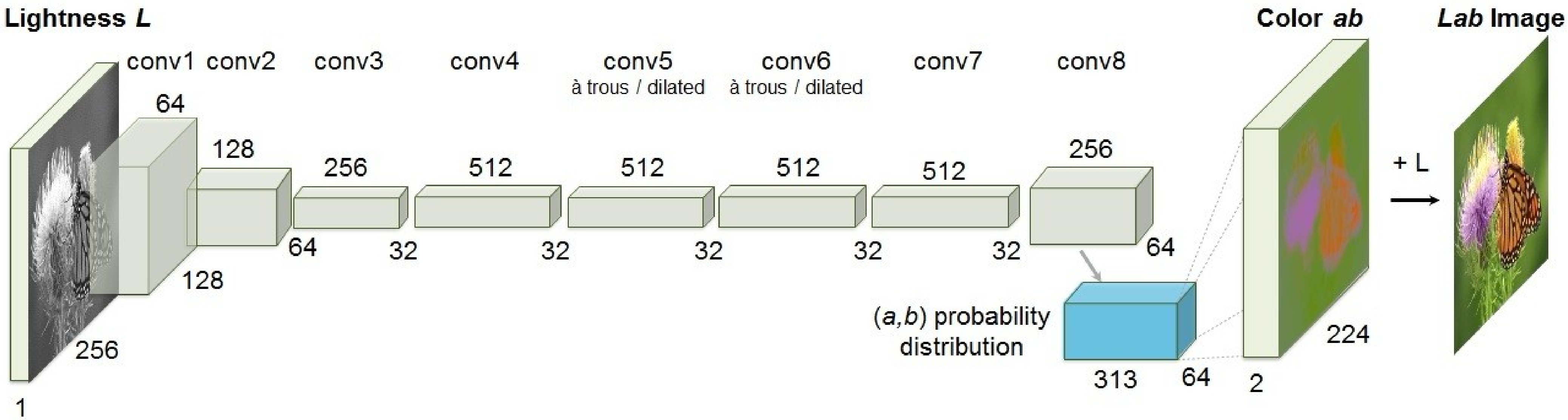
GANs

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

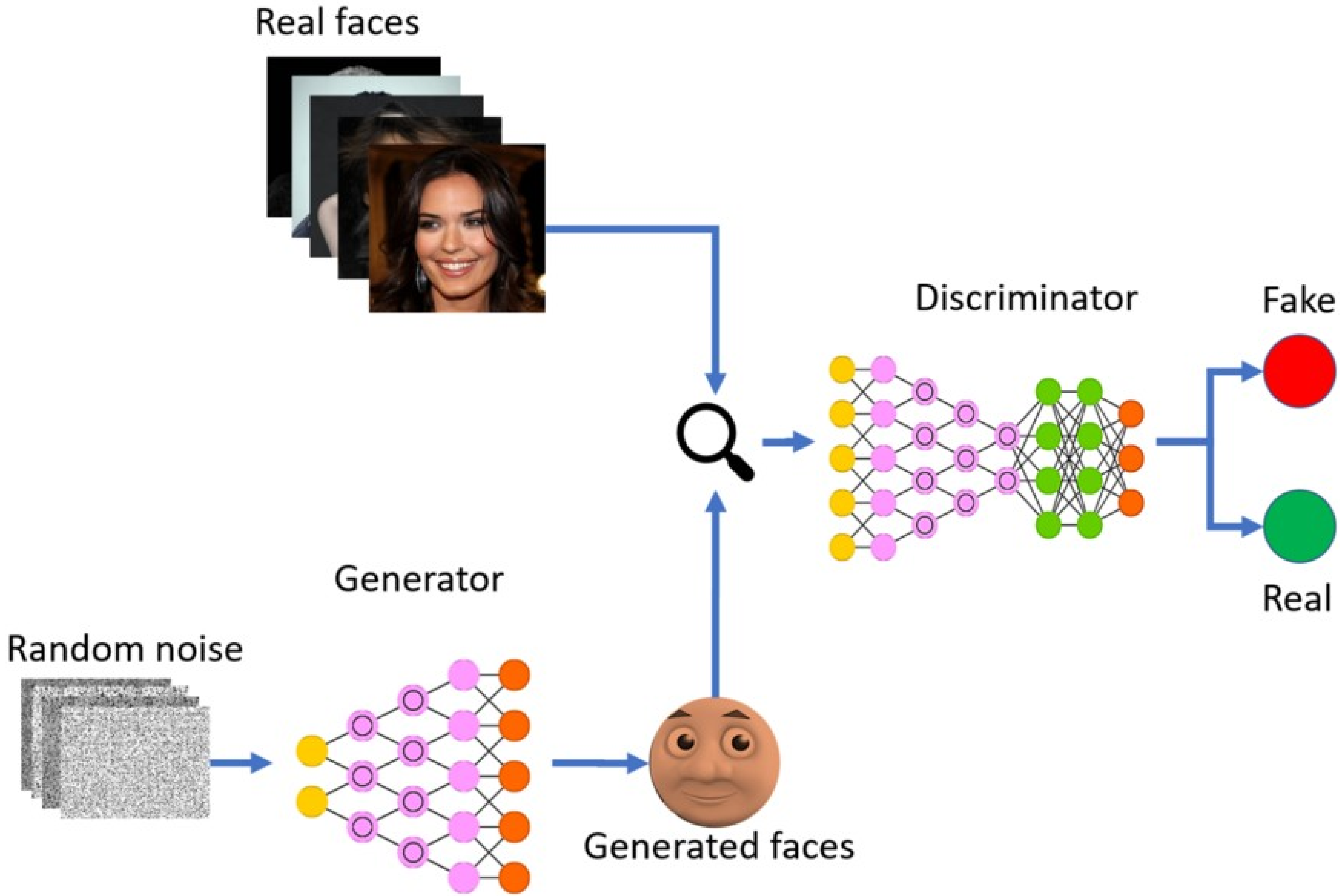


Examples



Lightness *L*
conv1
conv2
conv3
conv4
conv5
à trous / dilated
conv6
à trous / dilated
conv7
conv8
Color *ab*
Lab Image
64
128
256
512
512
512
512
256
+ L
1
256
128
64
32
32
32
32
32
64
(*a*,*b*) probability distribution
313
64
2
224


Real faces
Discriminator
Fake
Real
Generator
Random noise
Generated faces

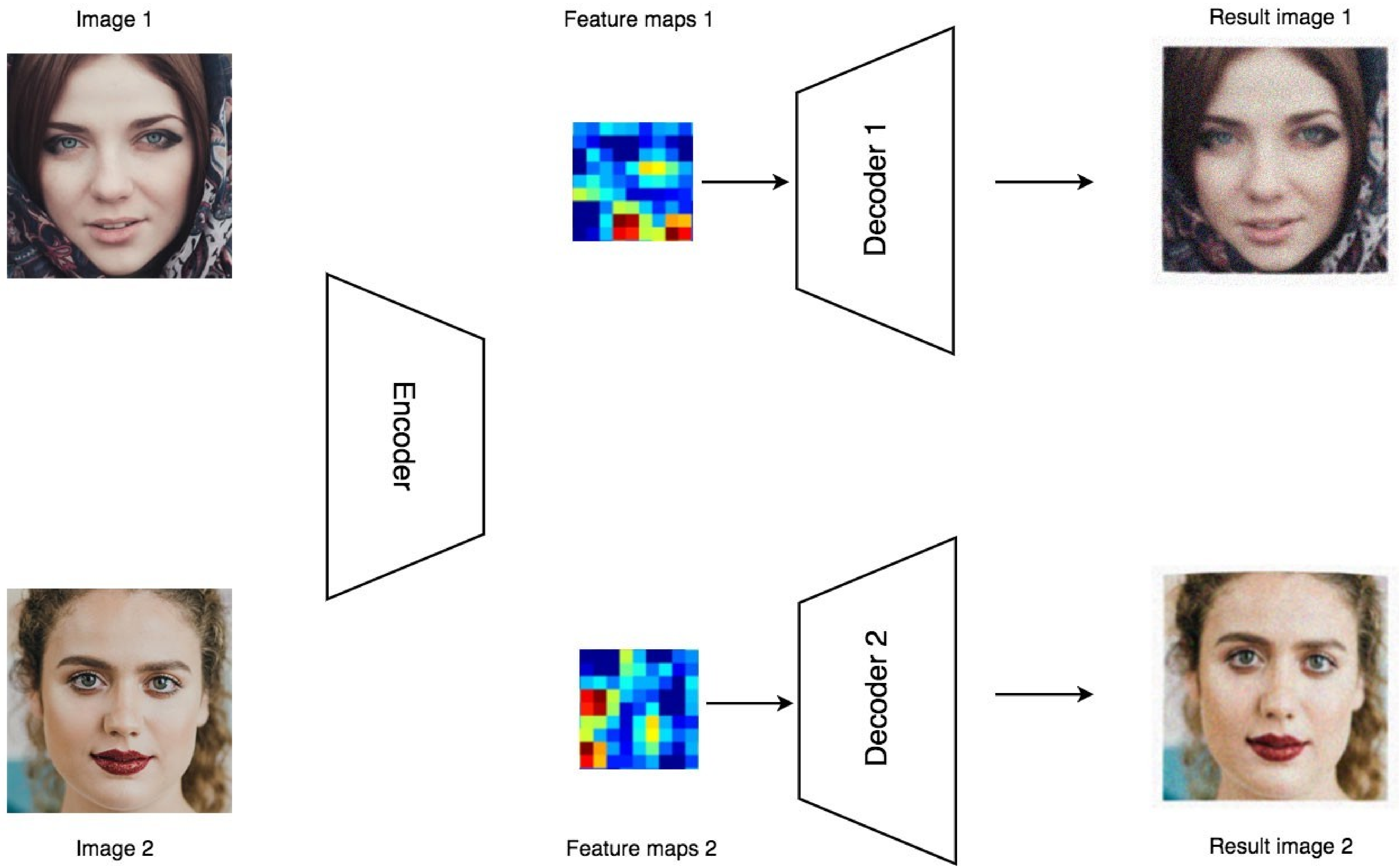
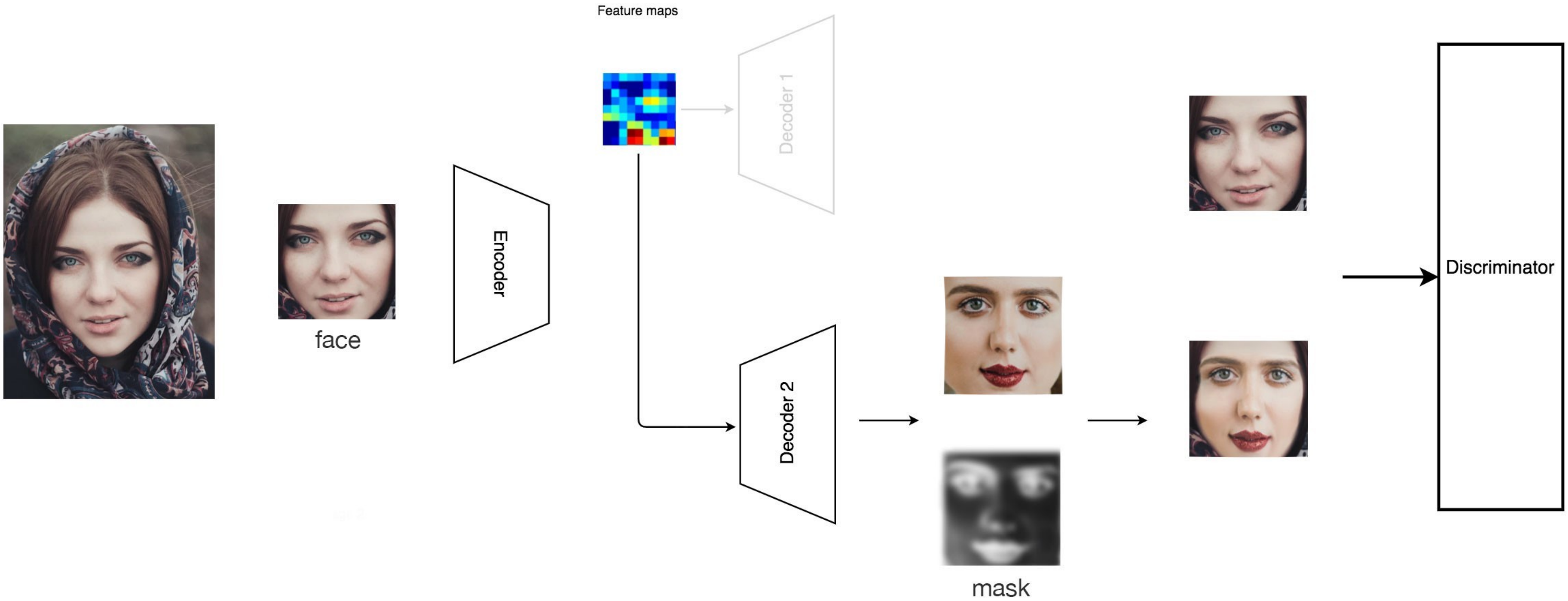
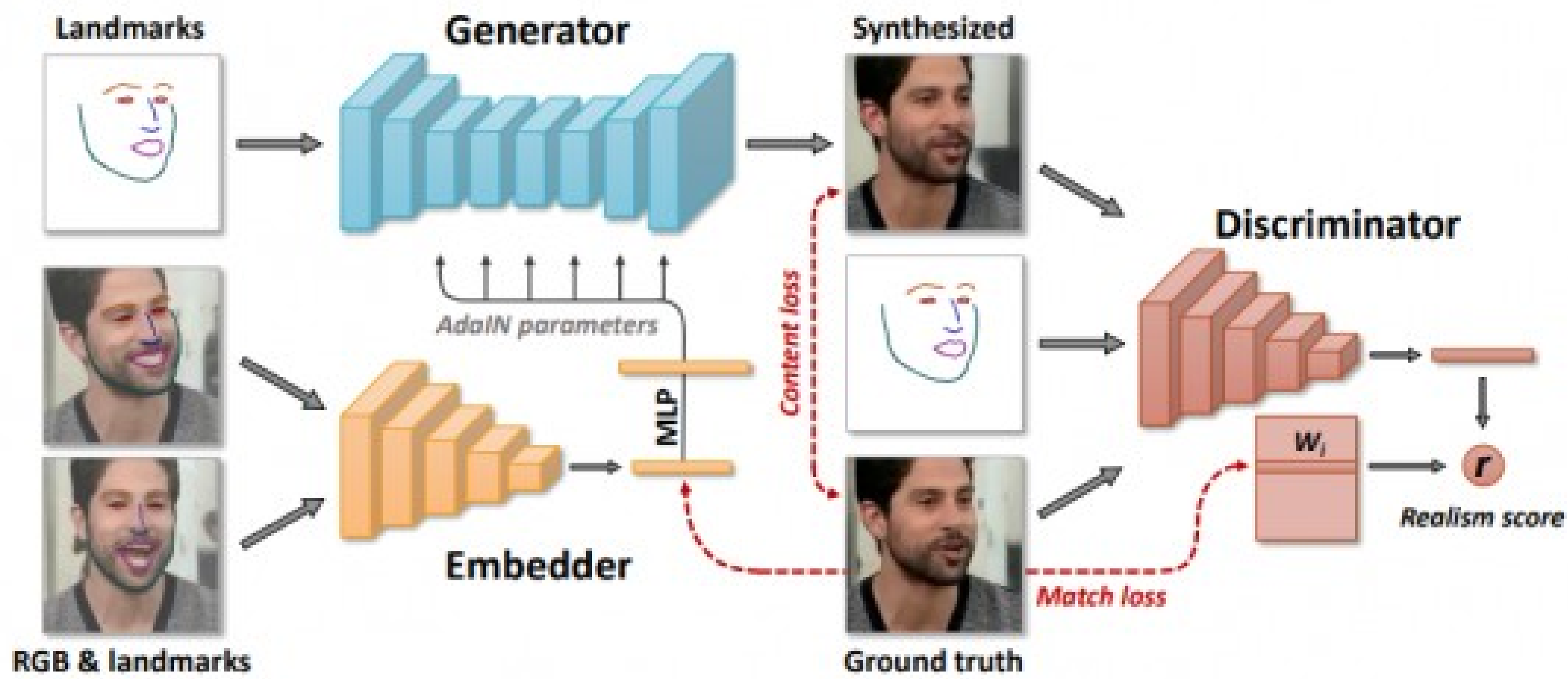
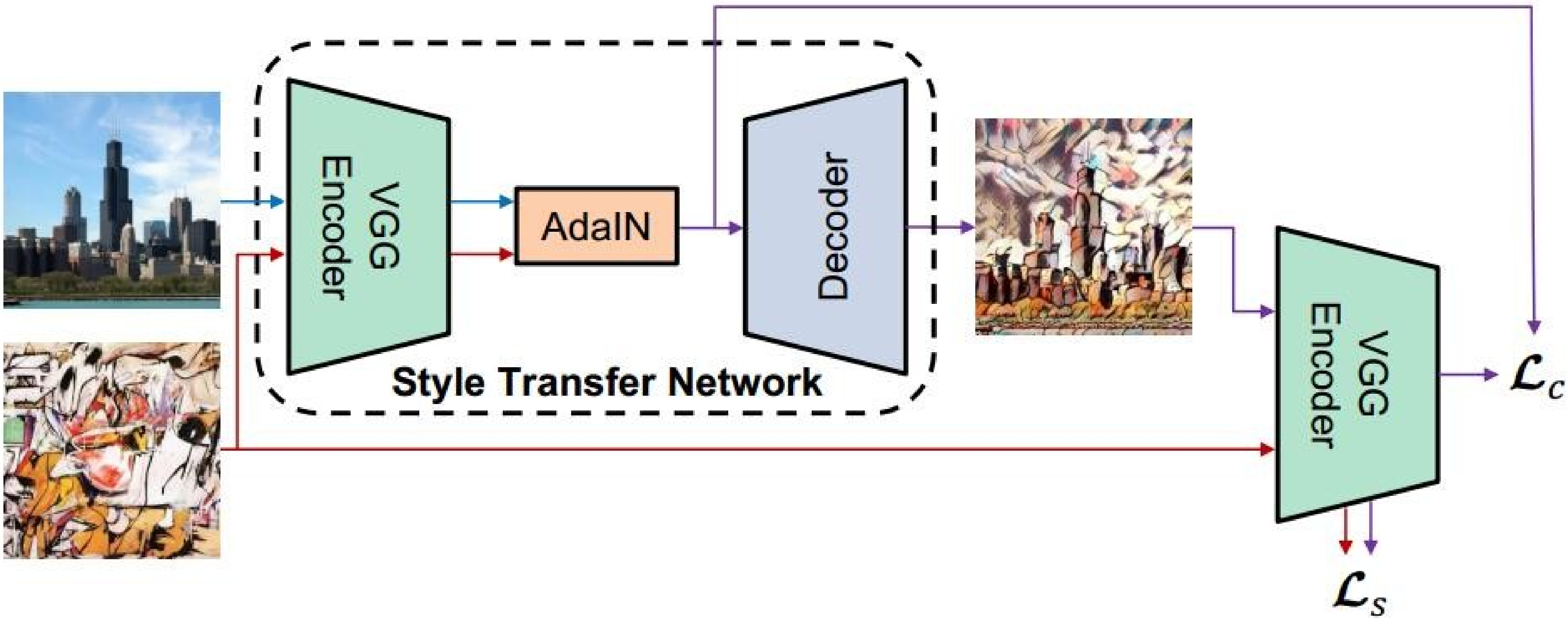


Image 1
Feature maps 1
Result image 1
Decoder 1
Encoder
Decoder 2
Image 2
Feature maps 2
Result image 2

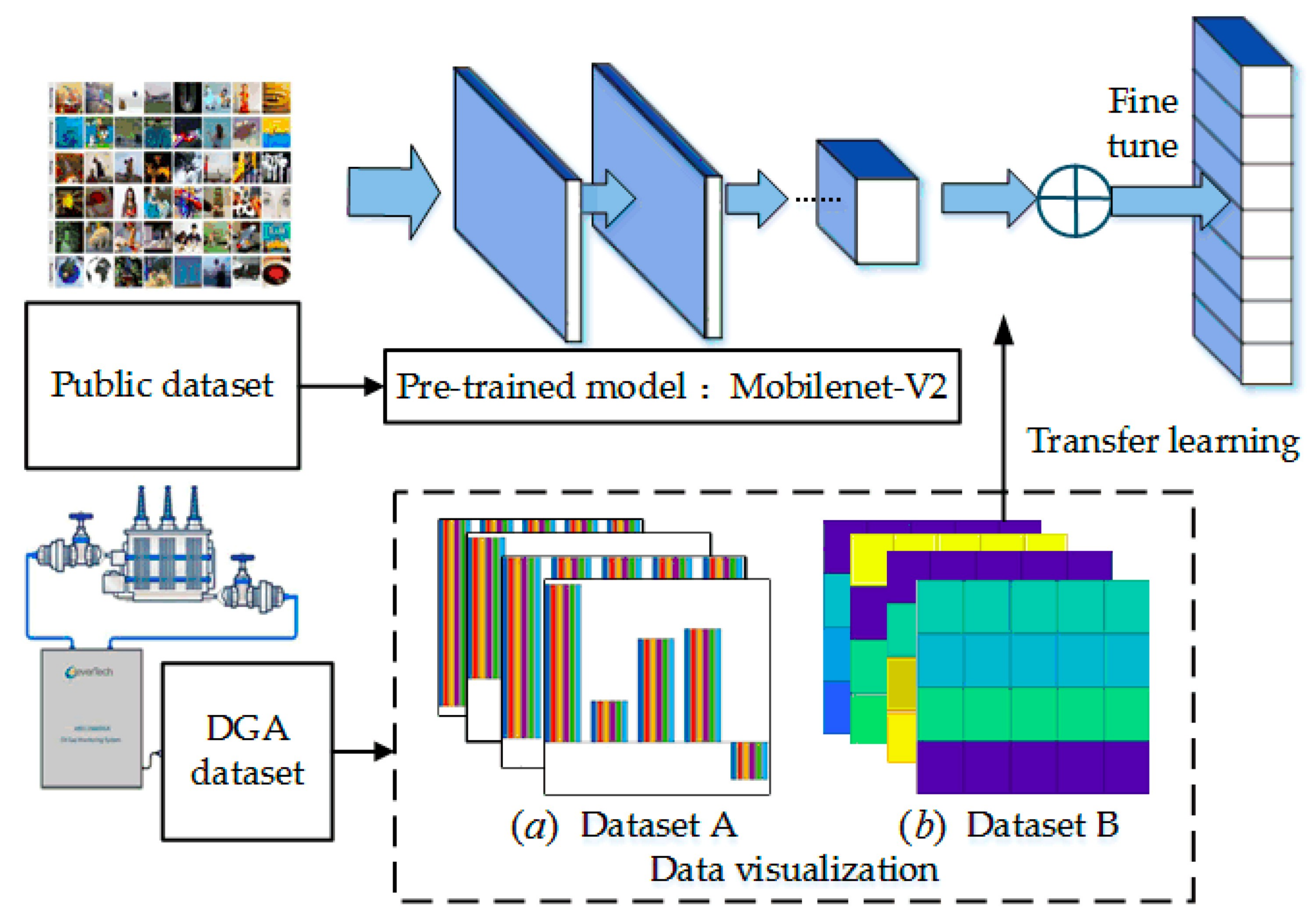

Feature maps
face
Encoder
Decoder 1
Decoder 2
mask
Discriminator

Adversarial Networks (GAN)





VGG Encoder
AdaIN
Decoder
Style Transfer Network
VGG Encoder
$\mathcal{L}_c$
$\mathcal{L}_s$



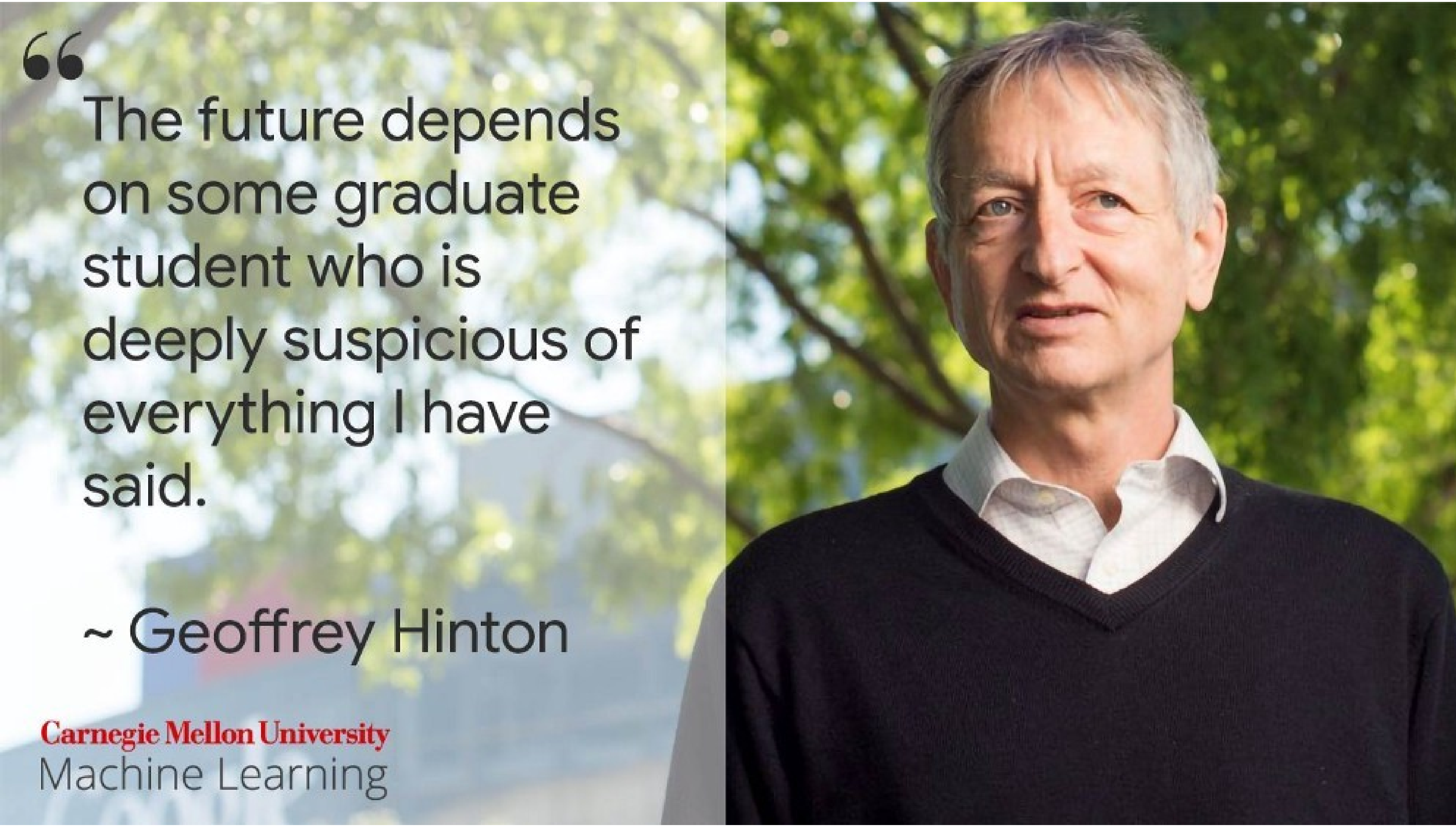
Fine tune
Public dataset
Pre-trained model：Mobilenet-V2
Transfer learning
DGA dataset
(a) Dataset A
(b) Dataset B
Data visualization

Aplicaciones

- Clasificación de imágenes
- Reconocimiento de actividades en imágenes
- OCR y escritura a mano
- Detección de caras y emociones. Agrupación.
- Detección de caras y objetos en vídeos
- Reconocimiento personalizado de imágenes

Aplicaciones

- Voz a texto
- Traducción en tiempo real
- Identificación y verificación por voz
- Análisis de texto (opiniones, extracción frases clave)
- Corrección ortográfica
- Detección de idioma y traducción de textos
- Reconocimiento del lenguaje natural (LUIS)


"The future depends on some graduate student who is deeply suspicious of everything I have said.
~ Geoffrey Hinton
Carnegie Mellon University
Machine Learning