Introduction to Neural Networks

Radek Poleski

rpoleski@astrouw.edu.pl

23.10.2024

Motivation

- Chat GPT (Nov 2022),
- End of semester poll of SJC (Jun 2024),
- OpenAI o1preview (Sep 2024),
- Nobel prizes in physics and chemistry (Oct 2024).

Motivation

- Chat GPT (Nov 2022),
- End of semester poll of SJC (Jun 2024),
- OpenAI o1preview (Sep 2024),
- Nobel prizes in physics and chemistry (Oct 2024).

Motivation

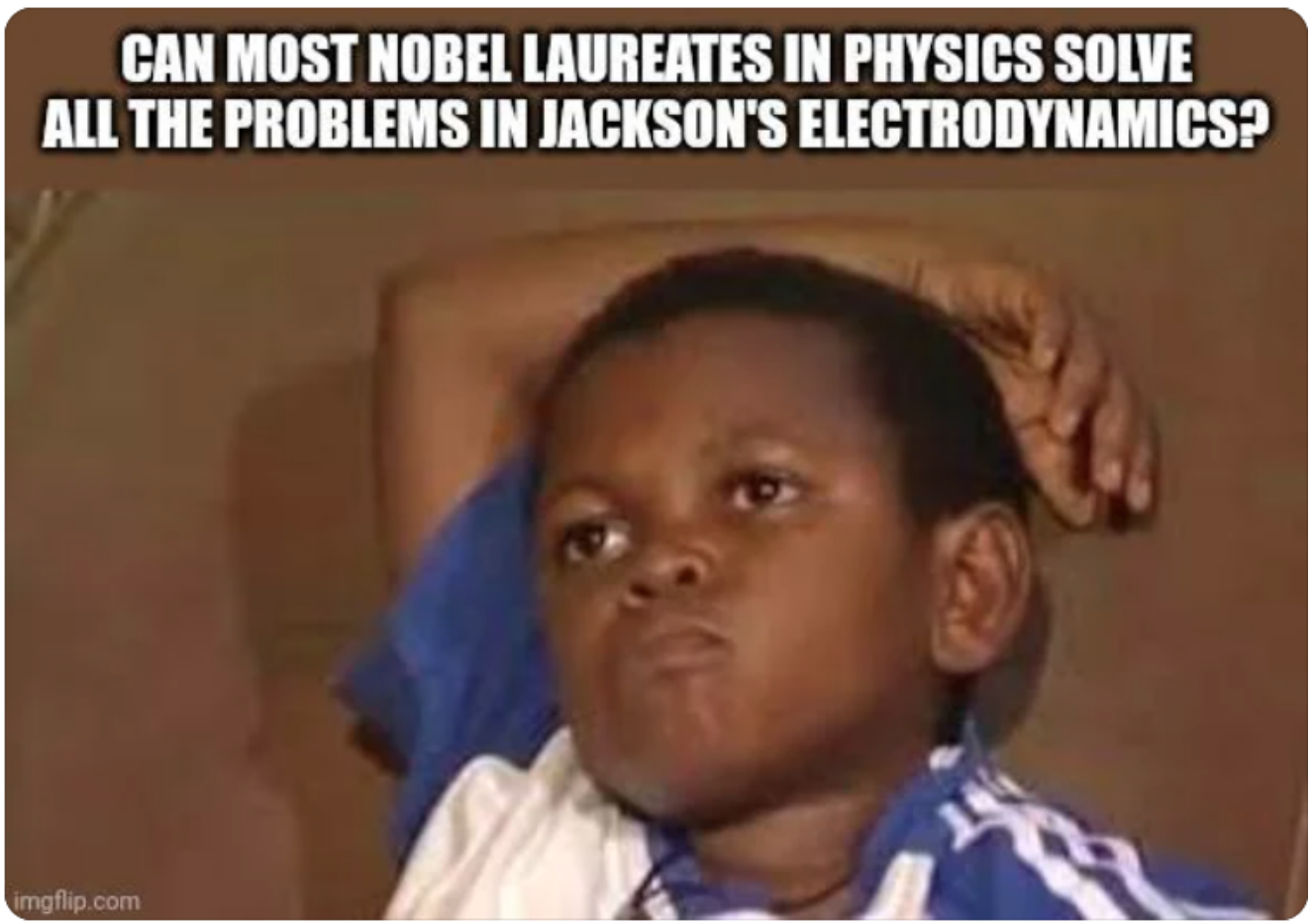
- Chat GPT (Nov 2022),
- End of semester poll of SJC (Jun 2024),
- OpenAI o1preview (Sep 2024),
- Nobel prizes in physics and chemistry (Oct 2024).

Motivation

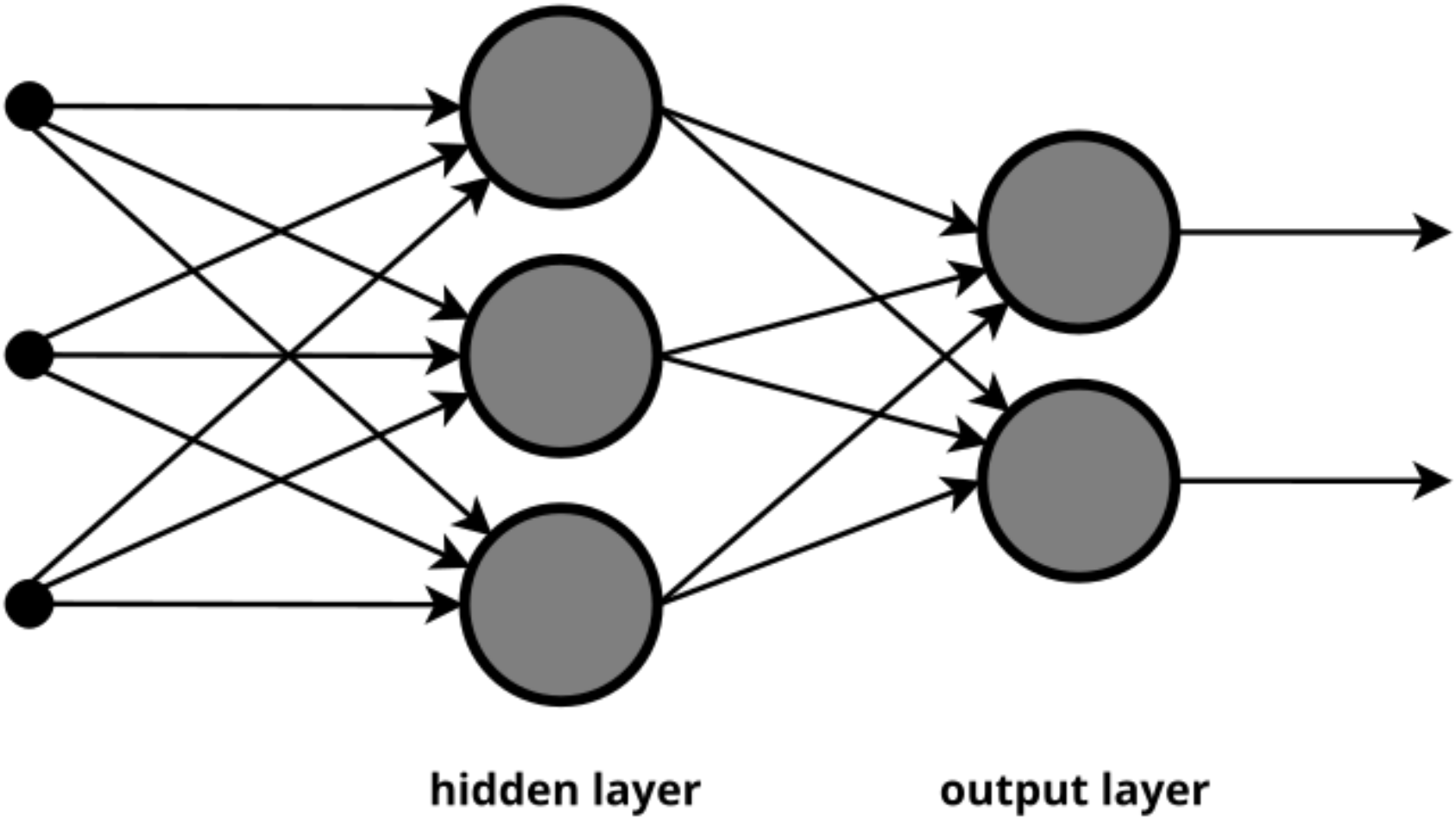
- Chat GPT (Nov 2022),
- End of semester poll of SJC (Jun 2024),
- OpenAI o1preview (Sep 2024),
- Nobel prizes in physics and chemistry (Oct 2024).

Motivation

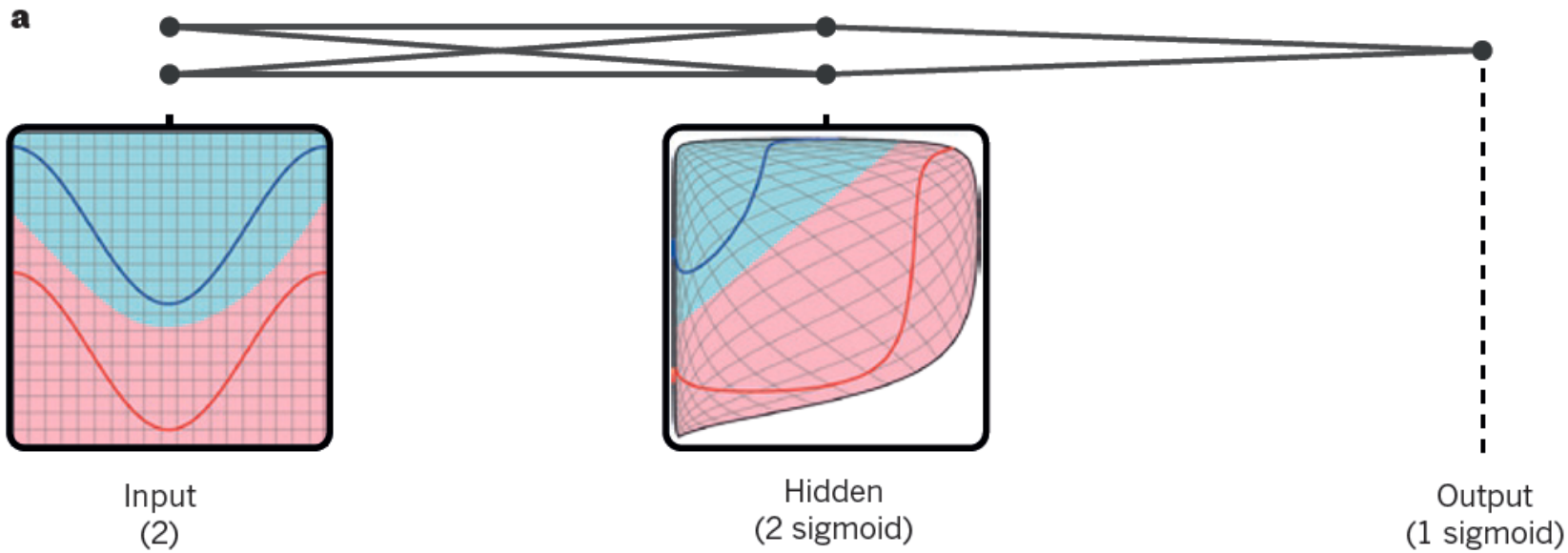
- Chat GPT (Nov 2022),
- End of semester poll of SJC (Jun 2024),
- OpenAI o1preview (Sep 2024),
- Nobel prizes in physics and chemistry (Oct 2024).



Feedforward Neural Network

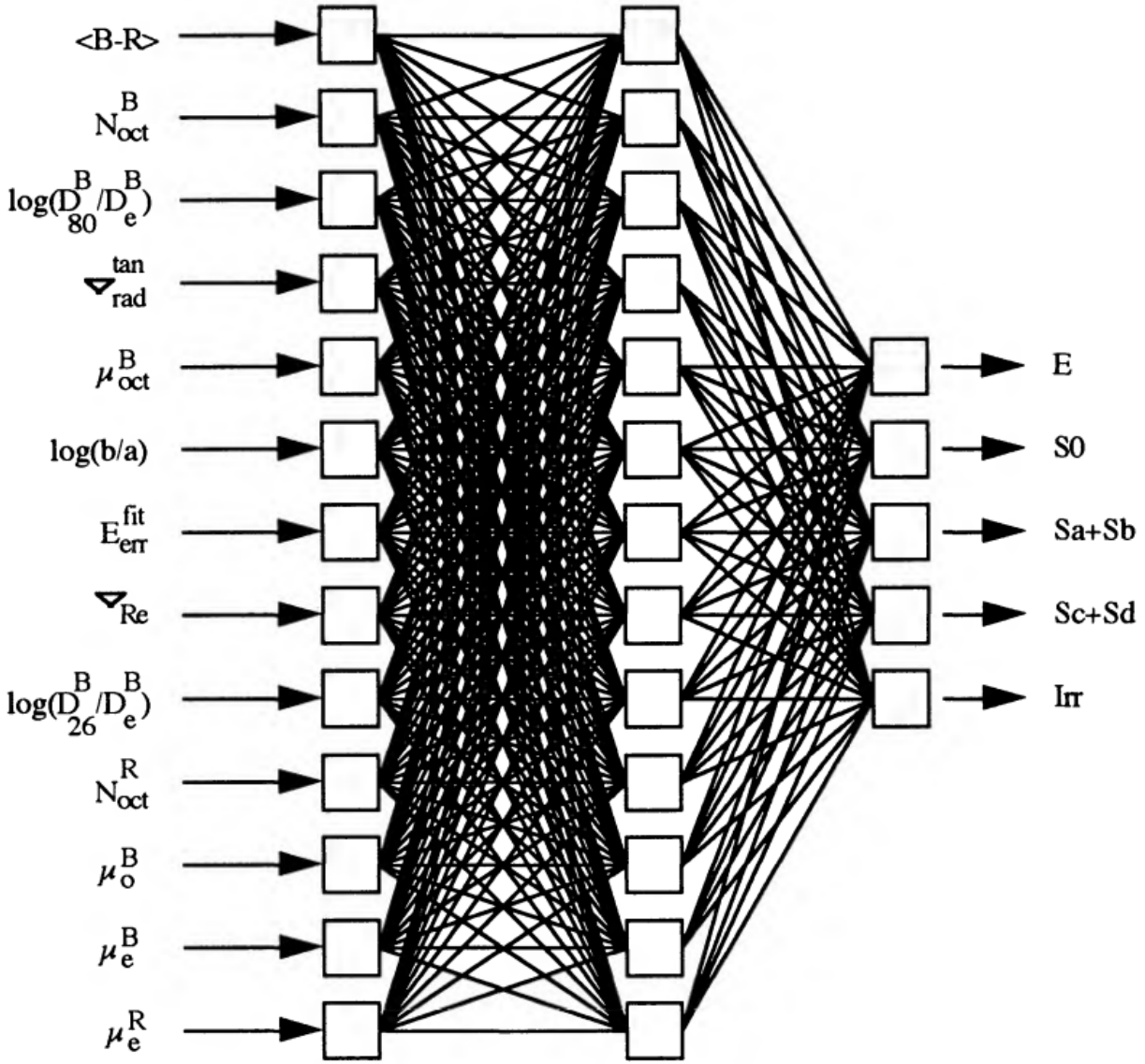


Very Simple Neural Network



LeCun et al. 2015; `https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`

First Neural Network in Astronomy



Storrie-Lombardi et al. 1992

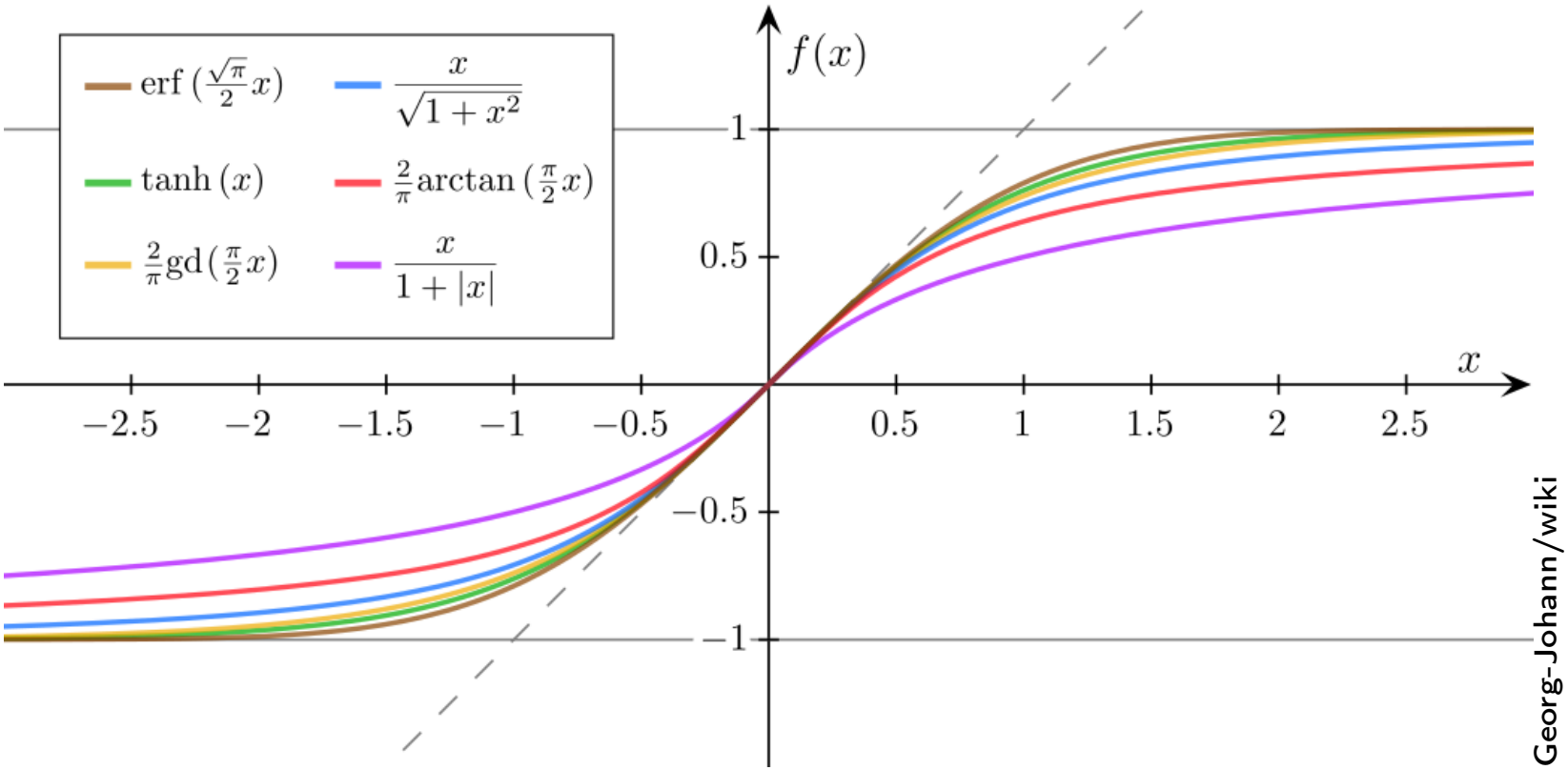
First Neural Network in Astronomy

Table 2. Galaxy classification.

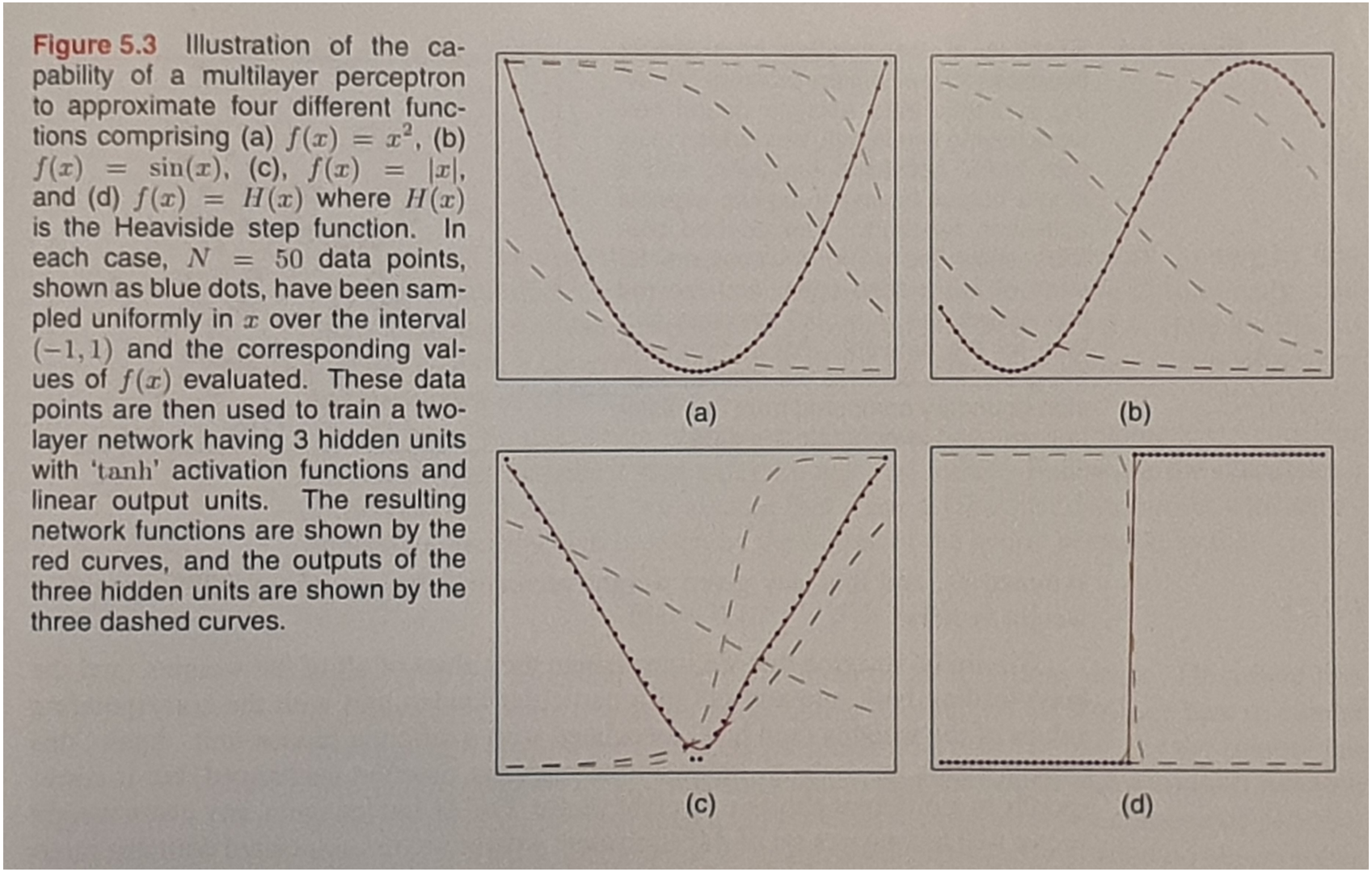
	(a) ANN					(b) ESO AUTO				
Class	E	S0	Sa+Sb	Sc+Sd	Irr	E	S0	Sa+Sb	Sc+Sd	Irr
E	203	77	25	1	5	197	87	17	5	5
S0	109	229	240	7	2	184	218	155	28	2
Sa+Sb	12	85	1281	218	15	106	12	791	664	38
Sc+Sd	1	4	304	415	36	22	11	24	631	72
Irr	0	0	53	69	126	22	9	31	42	144

Storrie-Lombardi et al. 1992

Smooth Activation Functions

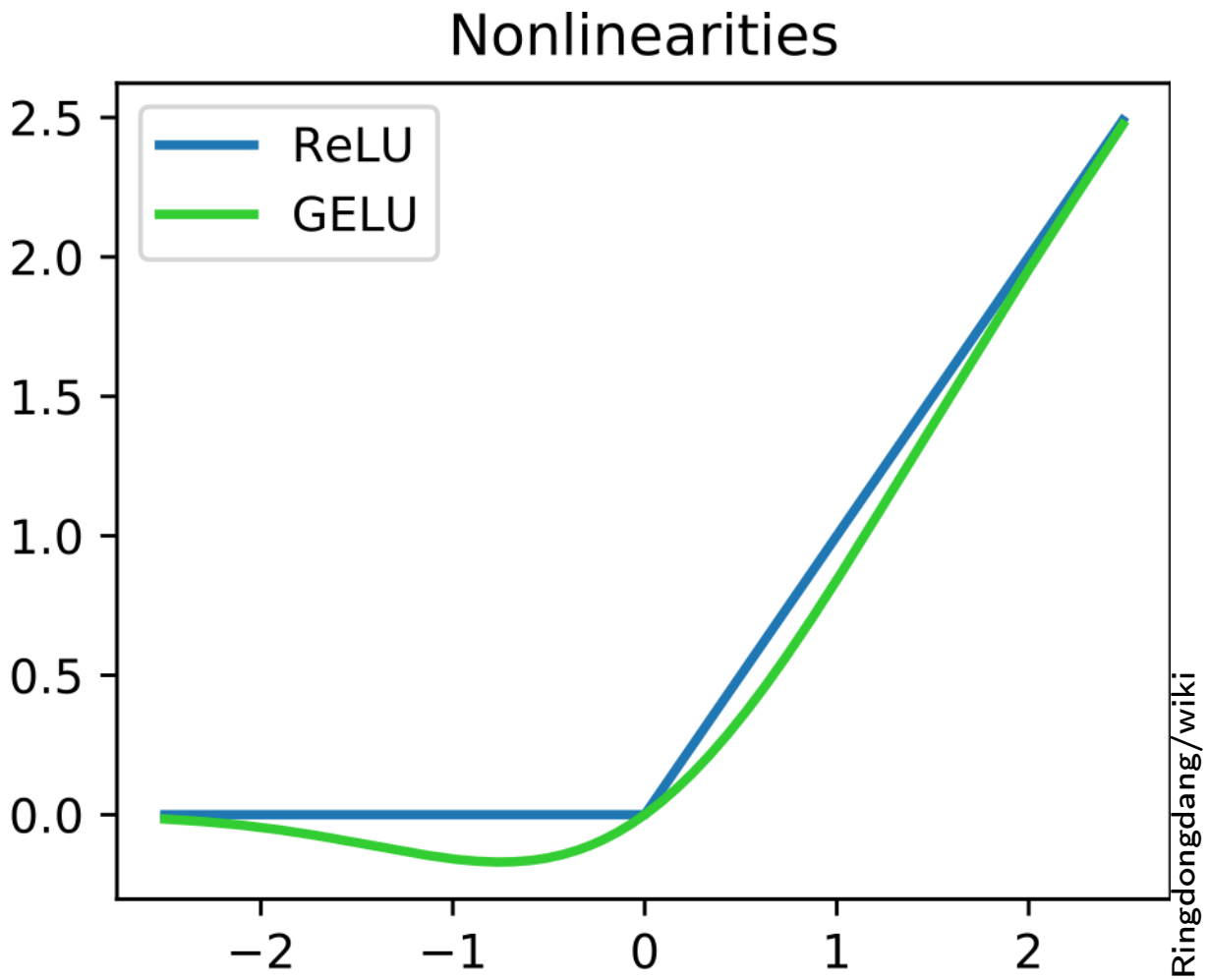


Ability to Approximate



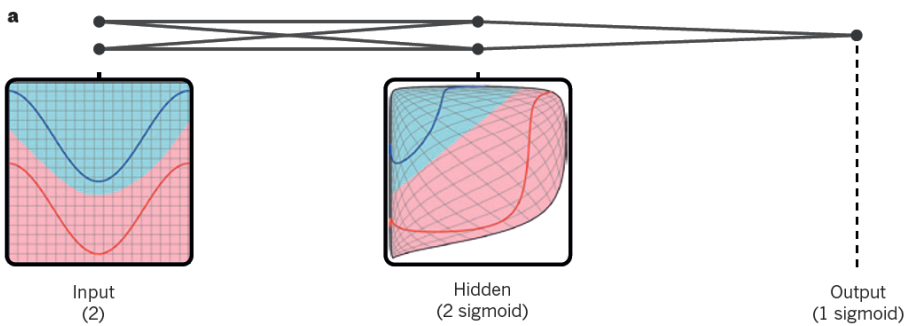
Bishop 2009 "Pattern Recognition and Machine Learning"

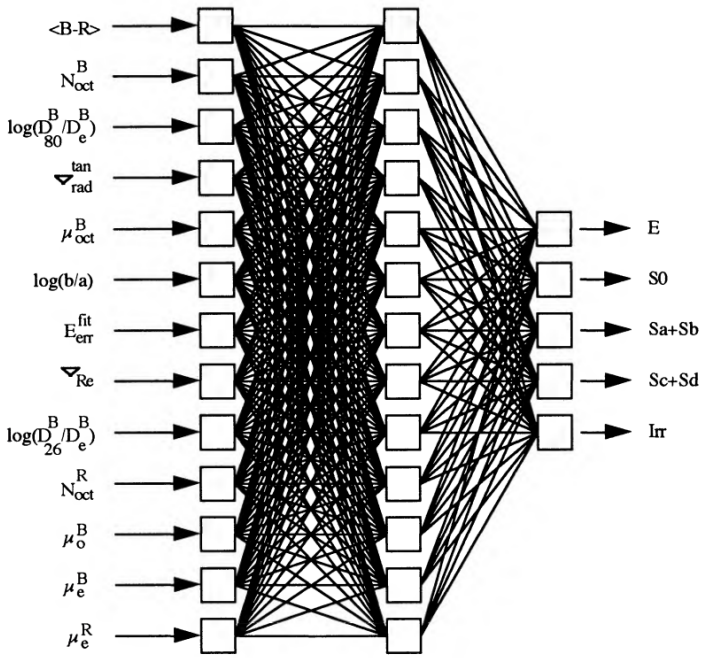
Rectified Linear Unit (ReLU)



LeCun et al. (2015): ReLU typically learns much faster in networks with many layers than tanh(z) or 1/(1+exp(-z)).

What it means to learn or teach the neural network?





Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.

$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.

Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.

To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.

From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 1

$z_j = h(a_j)$, where $h()$ is the activation function.
$a_j = \sum_i w_{j,i} z_i$, where we sum over all units that send connections to unit j in forward propagation.
Loss function E, e.g., $E = \sum_k (y_k - t_k)^2$, where y_k are network outputs and t_k are corresponding target outputs.
To train the network we want to know the gradient $\partial E / \partial w_{j,i}$, which is calculated using chain rule:

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}}.$$

We introduce notation: $\delta_j \equiv \partial E / \partial a_j$.
From definition of a_j we have:

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i.$$

After substituting:

$$\frac{\partial E}{\partial w_{j,i}} = \delta_j z_i.$$

Backpropagation 2

We have to find δ_j – use chain rule once more:

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j}.$$

From the definition of $a_k = \sum_j w_{k,j} z_j$ and $z_j = h(a_j)$:

$$\frac{\partial a_k}{\partial a_j} = \sum_j w_{k,j} \frac{\partial h(a_j)}{\partial a_j}.$$

Last two equations combined:

$$\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}.$$

Compare it with

$$a_j = \sum_k w_{j,k} z_k.$$

Backpropagation 2

We have to find δ_j – use chain rule once more:

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j}.$$

From the definition of $a_k = \sum_j w_{k,j} z_j$ and $z_j = h(a_j)$:

$$\frac{\partial a_k}{\partial a_j} = \sum_j w_{k,j} \frac{\partial h(a_j)}{\partial a_j}.$$

Last two equations combined:

$$\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}.$$

Compare it with

$$a_j = \sum_k w_{j,k} z_k.$$

Backpropagation 2

We have to find δ_j – use chain rule once more:

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j}.$$

From the definition of $a_k = \sum_j w_{k,j} z_j$ and $z_j = h(a_j)$:

$$\frac{\partial a_k}{\partial a_j} = \sum_j w_{k,j} \frac{\partial h(a_j)}{\partial a_j}.$$

Last two equations combined:

$$\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}.$$

Compare it with

$$a_j = \sum_k w_{j,k} z_k.$$

Backpropagation 2

We have to find δ_j – use chain rule once more:

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j}.$$

From the definition of $a_k = \sum_j w_{k,j} z_j$ and $z_j = h(a_j)$:

$$\frac{\partial a_k}{\partial a_j} = \sum_j w_{k,j} \frac{\partial h(a_j)}{\partial a_j}.$$

Last two equations combined:

$$\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}.$$

Compare it with

$$a_j = \sum_k w_{j,k} z_k.$$

Backpropagation 3 – algorithm

1. Feed forward network using $a_j = \sum_i w_{j,i} z_i$ and $z_j = h(a_j)$.
2. Evaluate δ_k for output units: $\delta_k = 2(y_k - t_k)h'(a_k)$.
3. Evaluate δ_k for all other units using backpropagation: $\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}$.
4. Evaluate derivatives: $\partial E / \partial w_{j,i} = \delta_j z_i$.
5. Use derivatives to update weights $w_{j,i}$.

Backpropagation 3 – algorithm

1. Feed forward network using $a_j = \sum_i w_{j,i} z_i$ and $z_j = h(a_j)$.
2. Evaluate δ_k for output units: $\delta_k = 2(y_k - t_k)h'(a_k)$.
3. Evaluate δ_k for all other units using backpropagation: $\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}$.
4. Evaluate derivatives: $\partial E / \partial w_{j,i} = \delta_j z_i$.
5. Use derivatives to update weights $w_{j,i}$.

Backpropagation 3 – algorithm

1. Feed forward network using $a_j = \sum_i w_{j,i} z_i$ and $z_j = h(a_j)$.
2. Evaluate δ_k for output units: $\delta_k = 2(y_k - t_k)h'(a_k)$.
3. Evaluate δ_k for all other units using backpropagation: $\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}$.
4. Evaluate derivatives: $\partial E / \partial w_{j,i} = \delta_j z_i$.
5. Use derivatives to update weights $w_{j,i}$.

Backpropagation 3 – algorithm

1. Feed forward network using $a_j = \sum_i w_{j,i} z_i$ and $z_j = h(a_j)$.
2. Evaluate δ_k for output units: $\delta_k = 2(y_k - t_k)h'(a_k)$.
3. Evaluate δ_k for all other units using backpropagation: $\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}$.
4. Evaluate derivatives: $\partial E / \partial w_{j,i} = \delta_j z_i$.
5. Use derivatives to update weights $w_{j,i}$.

Backpropagation 3 – algorithm

1. Feed forward network using $a_j = \sum_i w_{j,i} z_i$ and $z_j = h(a_j)$.
2. Evaluate δ_k for output units: $\delta_k = 2(y_k - t_k)h'(a_k)$.
3. Evaluate δ_k for all other units using backpropagation: $\delta_j = h'(a_j) \sum_k \delta_k w_{k,j}$.
4. Evaluate derivatives: $\partial E / \partial w_{j,i} = \delta_j z_i$.
5. Use derivatives to update weights $w_{j,i}$.

Deep learning – basic notes

Representation learning – automatic detection or classification based on raw data, i.e., no domain expertise needed.

Deep learning – representation learning with multiple levels of representation. Each level is composed of modules that transform representations from one level down to the more abstract level up. Each module increases selectivity and the invariance of the representation. For images typically:

1. pixel values,
2. presence of edges at particular locations and orientations,
3. detection of motifs by particular arrangements of edges,
4. detection of objects composed of motifs,
5. ...

Typically there are between 5 and 20 layers in deep networks.

based on LeCun et al. (2015)

Deep learning – basic notes

Representation learning – automatic detection or classification based on raw data, i.e., no domain expertise needed.

Deep learning – representation learning with multiple levels of representation. Each level is composed of modules that transform representations from one level down to the more abstract level up.

Each module increases selectivity and the invariance of the representation. For images typically:

1. pixel values,
2. presence of edges at particular locations and orientations,
3. detection of motifs by particular arrangements of edges,
4. detection of objects composed of motifs,
5. ...

Typically there are between 5 and 20 layers in deep networks.

based on LeCun et al. (2015)

Deep learning – basic notes

Representation learning – automatic detection or classification based on raw data, i.e., no domain expertise needed.

Deep learning – representation learning with multiple levels of representation. Each level is composed of modules that transform representations from one level down to the more abstract level up. Each module increases selectivity and the invariance of the representation. For images typically:

1. pixel values,
2. presence of edges at particular locations and orientations,
3. detection of motifs by particular arrangements of edges,
4. detection of objects composed of motifs,
5. ...

Typically there are between 5 and 20 layers in deep networks.

based on LeCun et al. (2015)

Deep learning – basic notes

Representation learning – automatic detection or classification based on raw data, i.e., no domain expertise needed.

Deep learning – representation learning with multiple levels of representation. Each level is composed of modules that transform representations from one level down to the more abstract level up. Each module increases selectivity and the invariance of the representation. For images typically:

1. pixel values,
2. presence of edges at particular locations and orientations,
3. detection of motifs by particular arrangements of edges,
4. detection of objects composed of motifs,
5. ...

Typically there are between 5 and 20 layers in deep networks.

based on LeCun et al. (2015)

Deep learning – basic notes

Representation learning – automatic detection or classification based on raw data, i.e., no domain expertise needed.

Deep learning – representation learning with multiple levels of representation. Each level is composed of modules that transform representations from one level down to the more abstract level up. Each module increases selectivity and the invariance of the representation. For images typically:

1. pixel values,
2. presence of edges at particular locations and orientations,
3. detection of motifs by particular arrangements of edges,
4. detection of objects composed of motifs,
5. ...

Typically there are between 5 and 20 layers in deep networks.

based on LeCun et al. (2015)

Deep Learning input/output data

Convolutional Neural Networks (CNN) – finite data, i.e., images or RGB images.

Recurrent Neural Networks (RNN) – possibly infinite data, i.e., sound, text, lightcurves(?).

Many natural signals have structure of hierarchies: higher-level features are composed of lower-level ones. Deep networks exploit this by using many layers.

Deep Learning input/output data

Convolutional Neural Networks (CNN) – finite data, i.e., images or RGB images.

Recurrent Neural Networks (RNN) – possibly infinite data, i.e., sound, text, lightcurves(?).

Many natural signals have structure of hierarchies: higher-level features are composed of lower-level ones. Deep networks exploit this by using many layers.

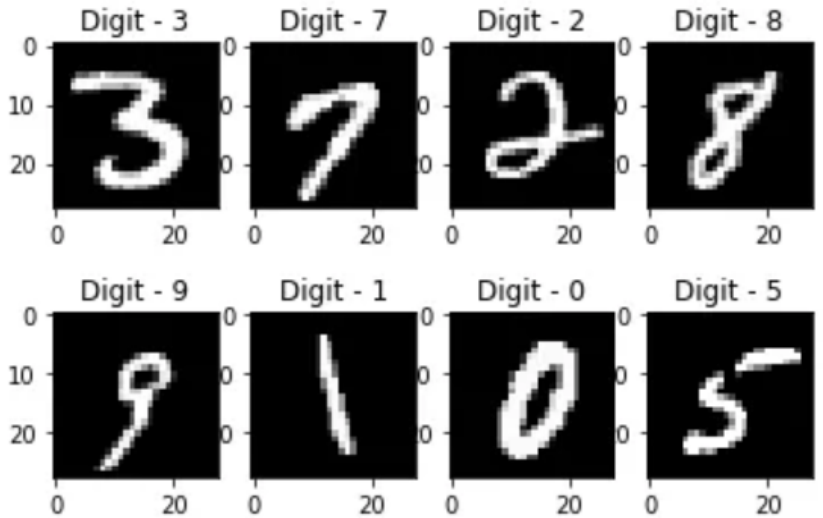
Deep Learning input/output data

Convolutional Neural Networks (CNN) – finite data, i.e., images or RGB images.

Recurrent Neural Networks (RNN) – possibly infinite data, i.e., sound, text, lightcurves(?).

Many natural signals have structure of hierarchies: higher-level features are composed of lower-level ones. Deep networks exploit this by using many layers.

Digit recognition



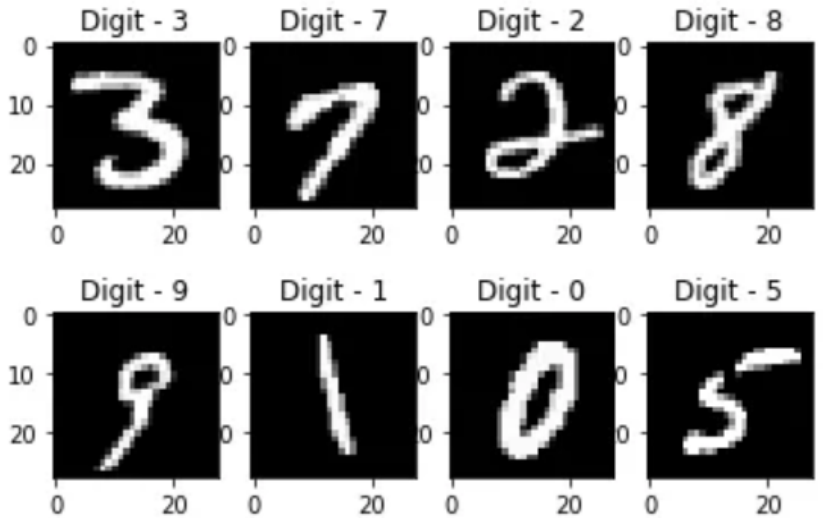
Sample Digits from MNIST dataset

Digits are invariant under translation, scaling, and small rotations.

Nearby pixels are more strongly correlated than distant pixels.

We have to start from local image features but it's hard to keep that information in fully-connected feed-forward network.

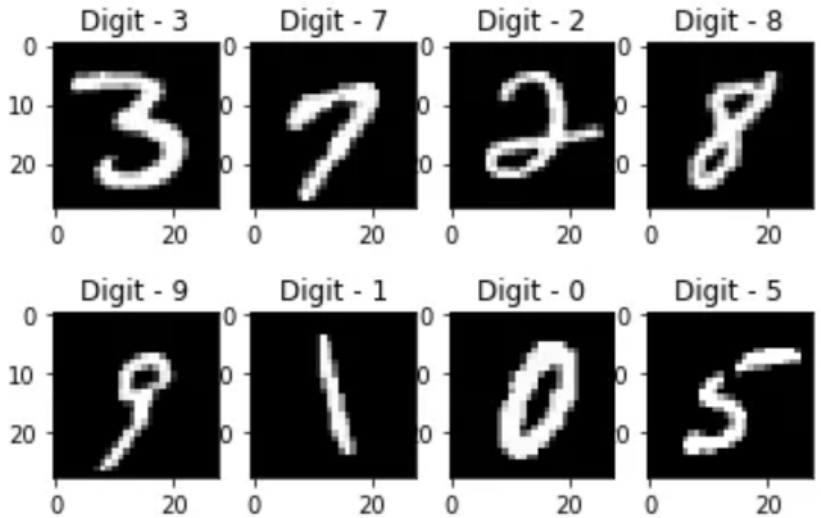
Digit recognition



Sample Digits from MNIST dataset

Digits are invariant under translation, scaling, and small rotations.
Nearby pixels are more strongly correlated than distant pixels.
We have to start from local image features but it's hard to keep that information in fully-connected feed-forward network.

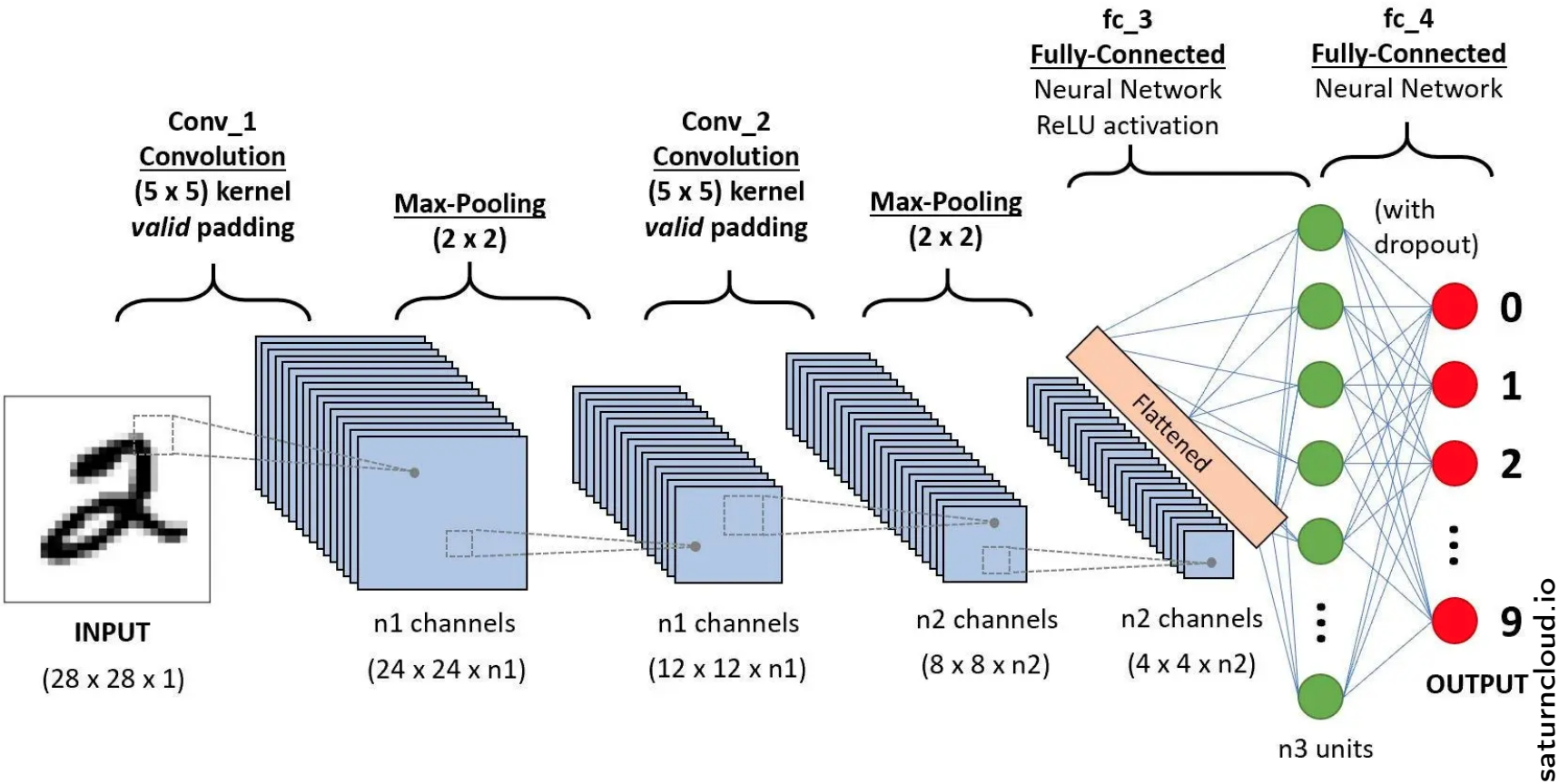
Digit recognition



Sample Digits from MNIST dataset

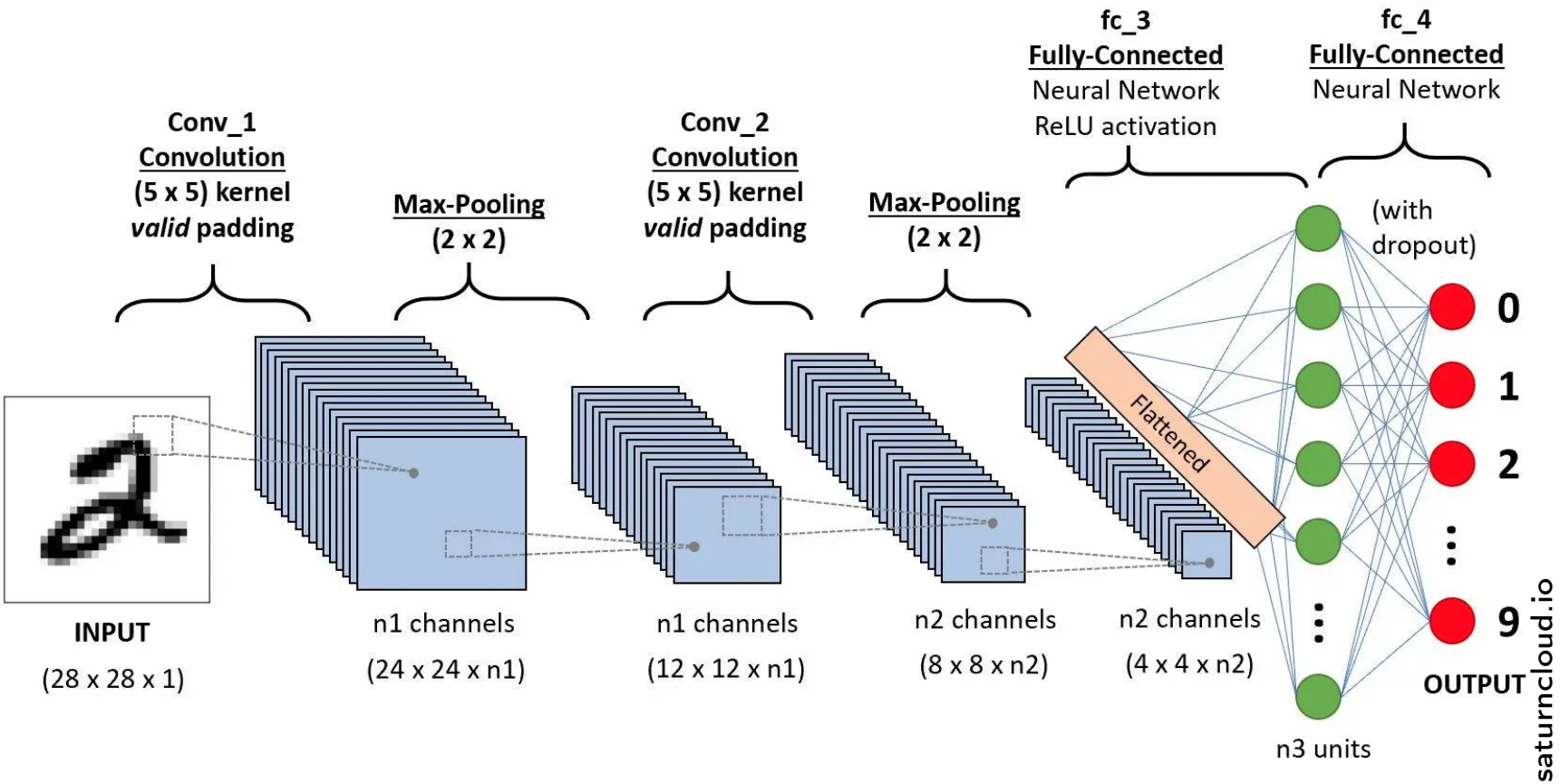
Digits are invariant under translation, scaling, and small rotations.
Nearby pixels are more strongly correlated than distant pixels.
We have to start from local image features but it's hard to keep that information in fully-connected feed-forward network.

CNN structure



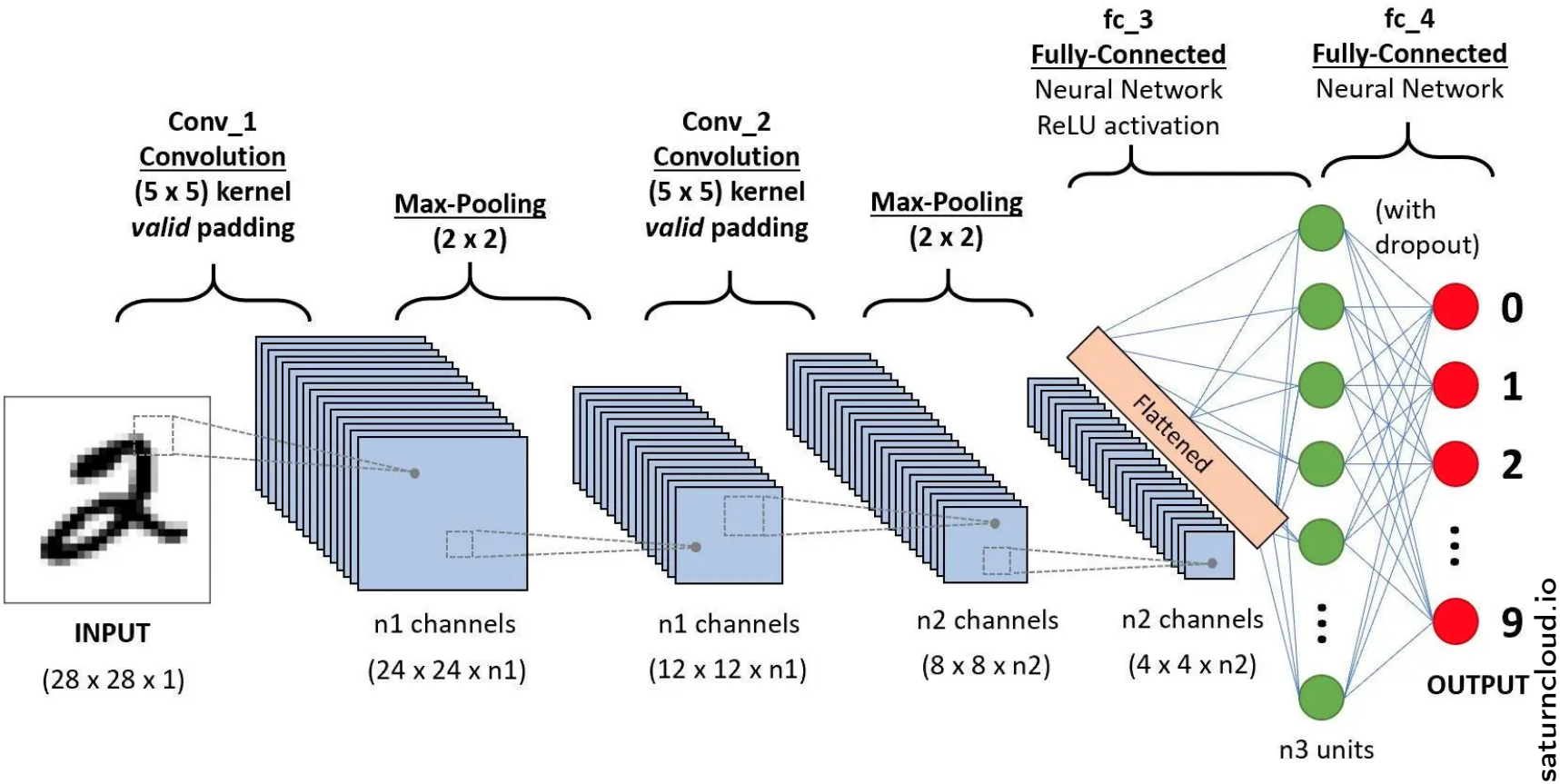
CNN: convolution + sub-sampling/pooling layers.

CNN structure



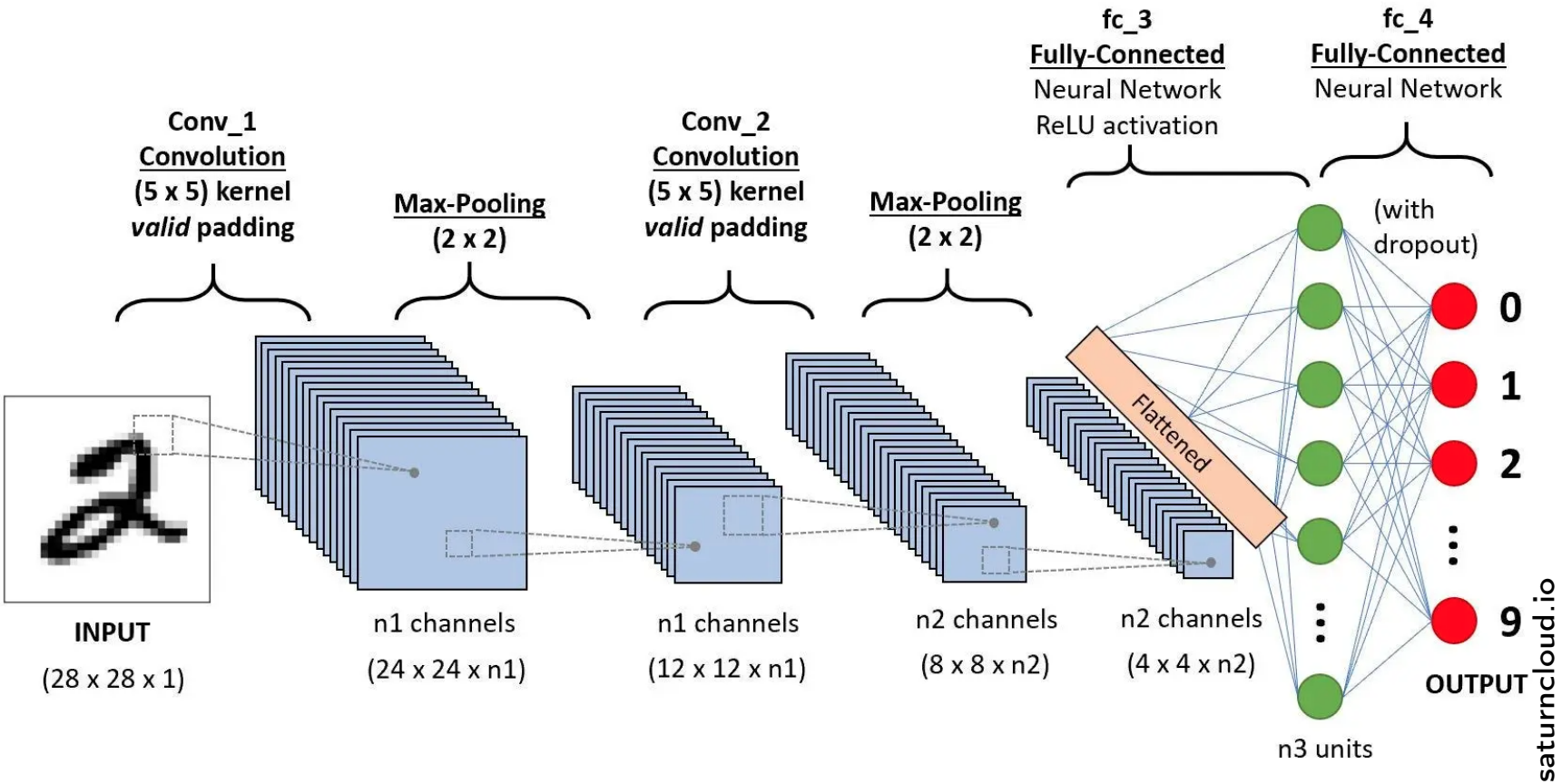
In convolution layer there are planes called feature maps.

CNN structure



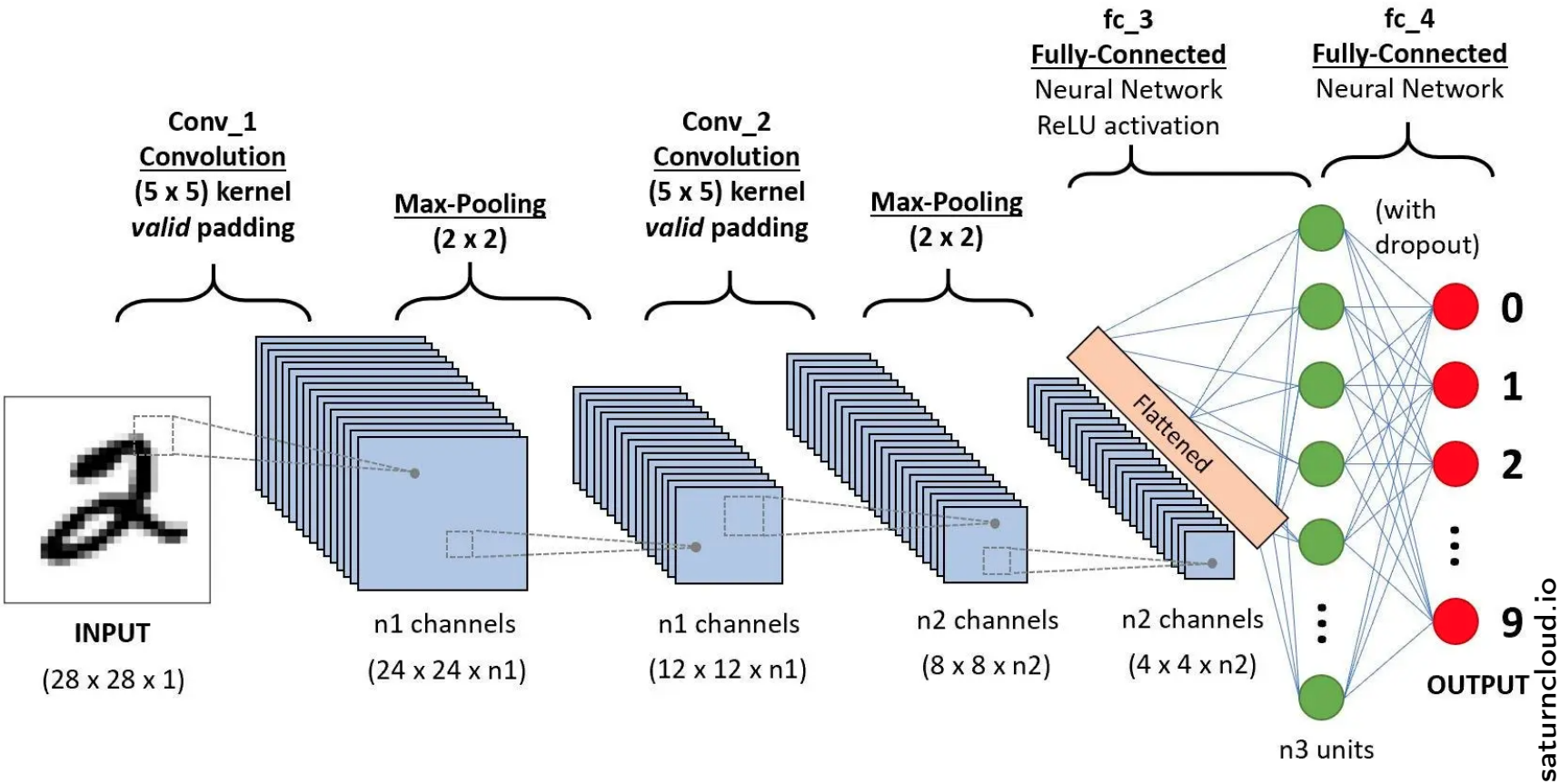
In convolution layer there are planes called feature maps. Units of feature map take small parts of the input data and apply weights.

CNN structure



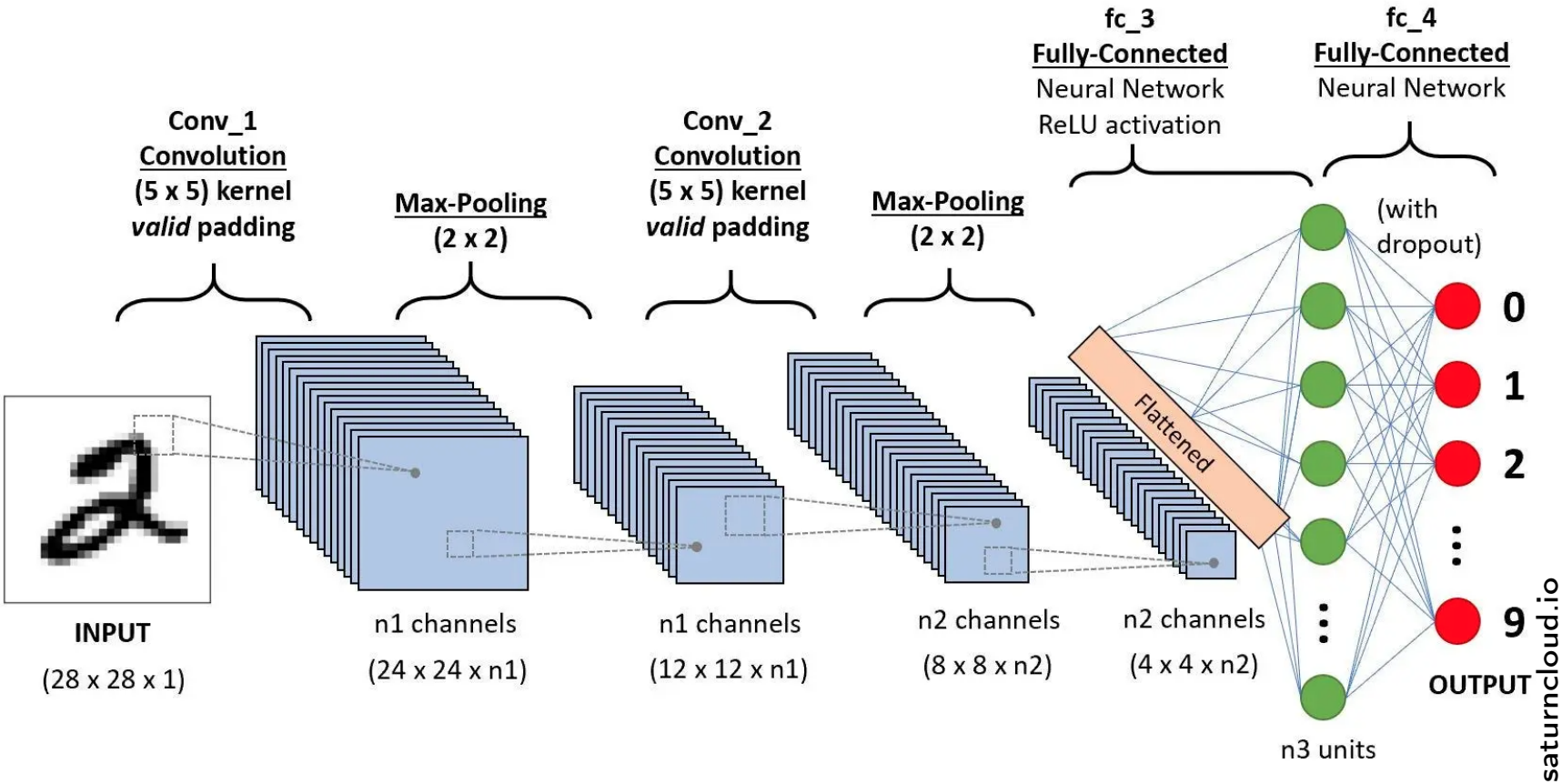
In convolution layer there are planes called feature maps. Units of feature map take small parts of the input data and apply weights. Weights are the same for all units in a map (convolution!; number of weights) but different maps have different weights.

CNN structure



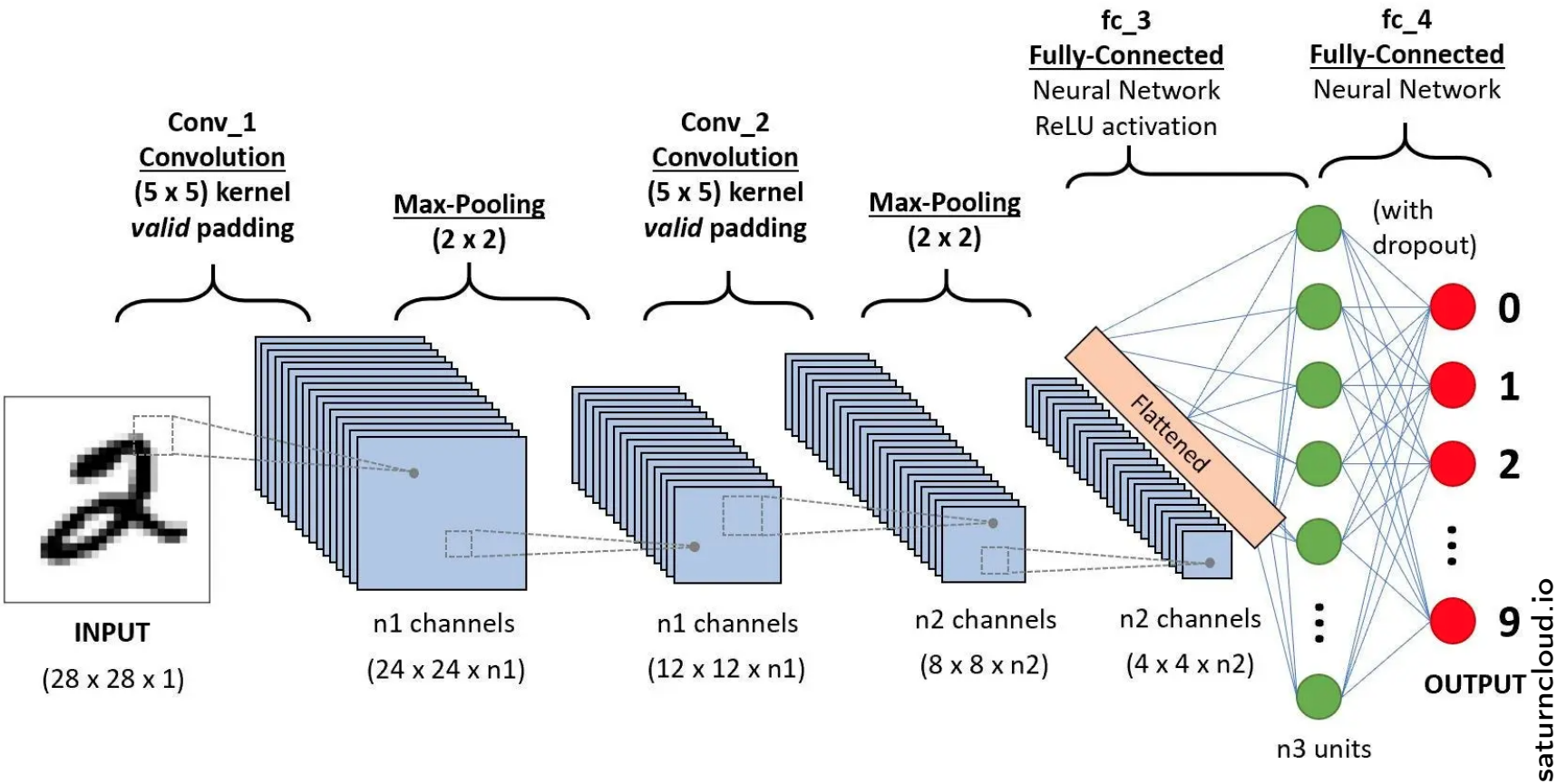
Each feature map detects a single pattern but at different locations on the image.

CNN structure



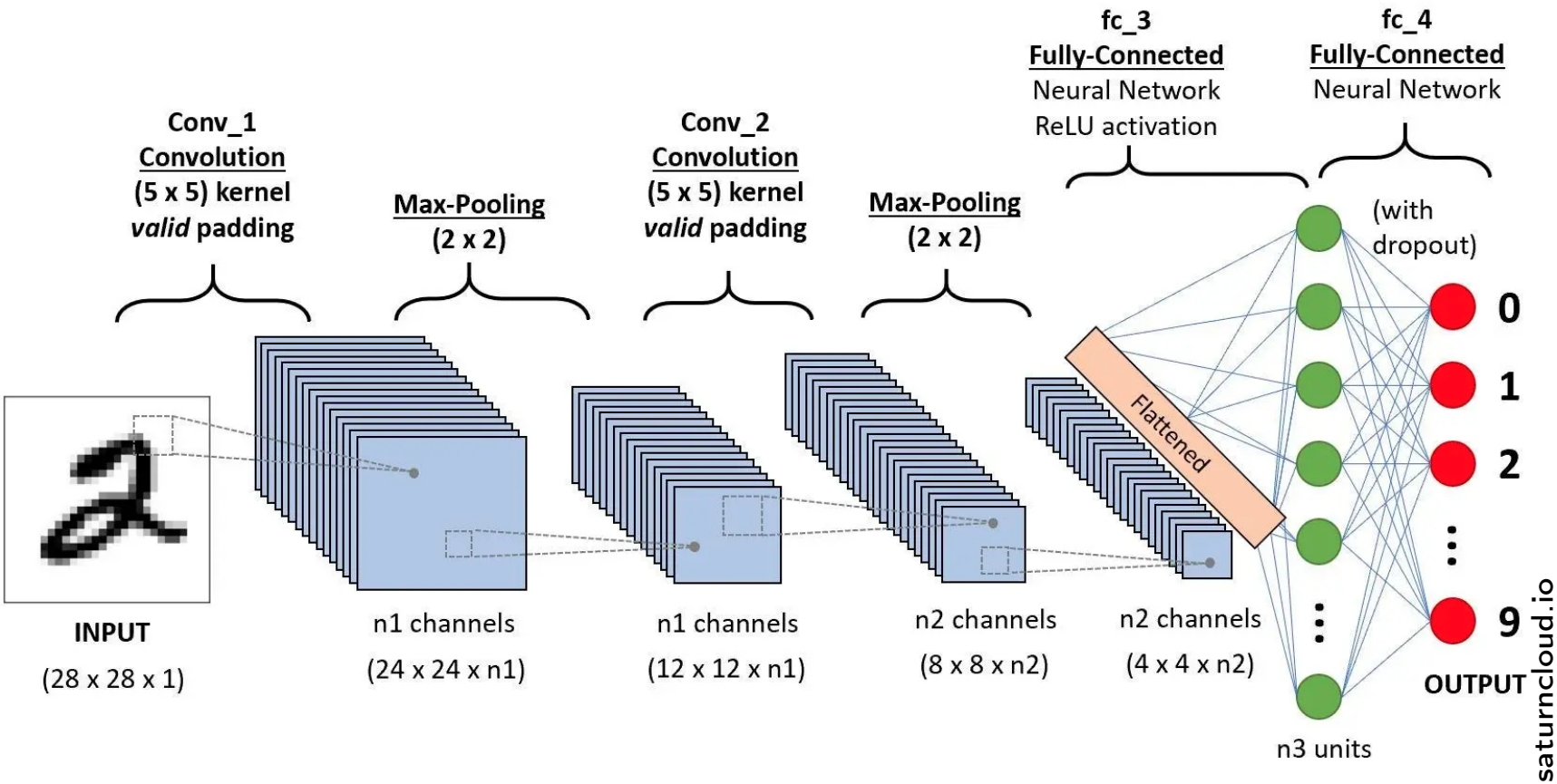
Each feature map detects a single pattern but at different locations on the image. Two units that are close on feature map take nearby parts of the input data.

CNN structure



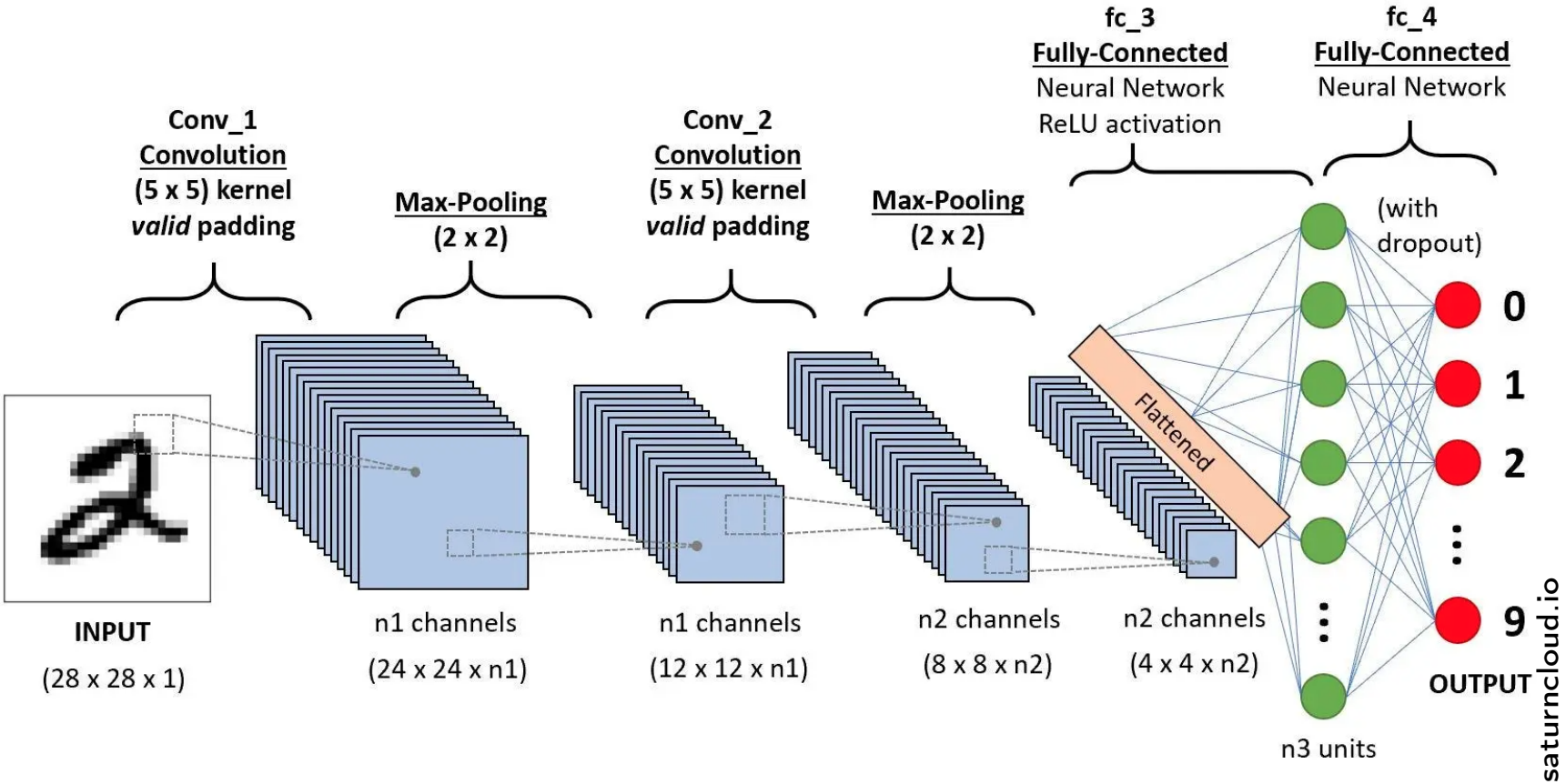
Pooling layer takes inputs from nearby parts of its input data.

CNN structure



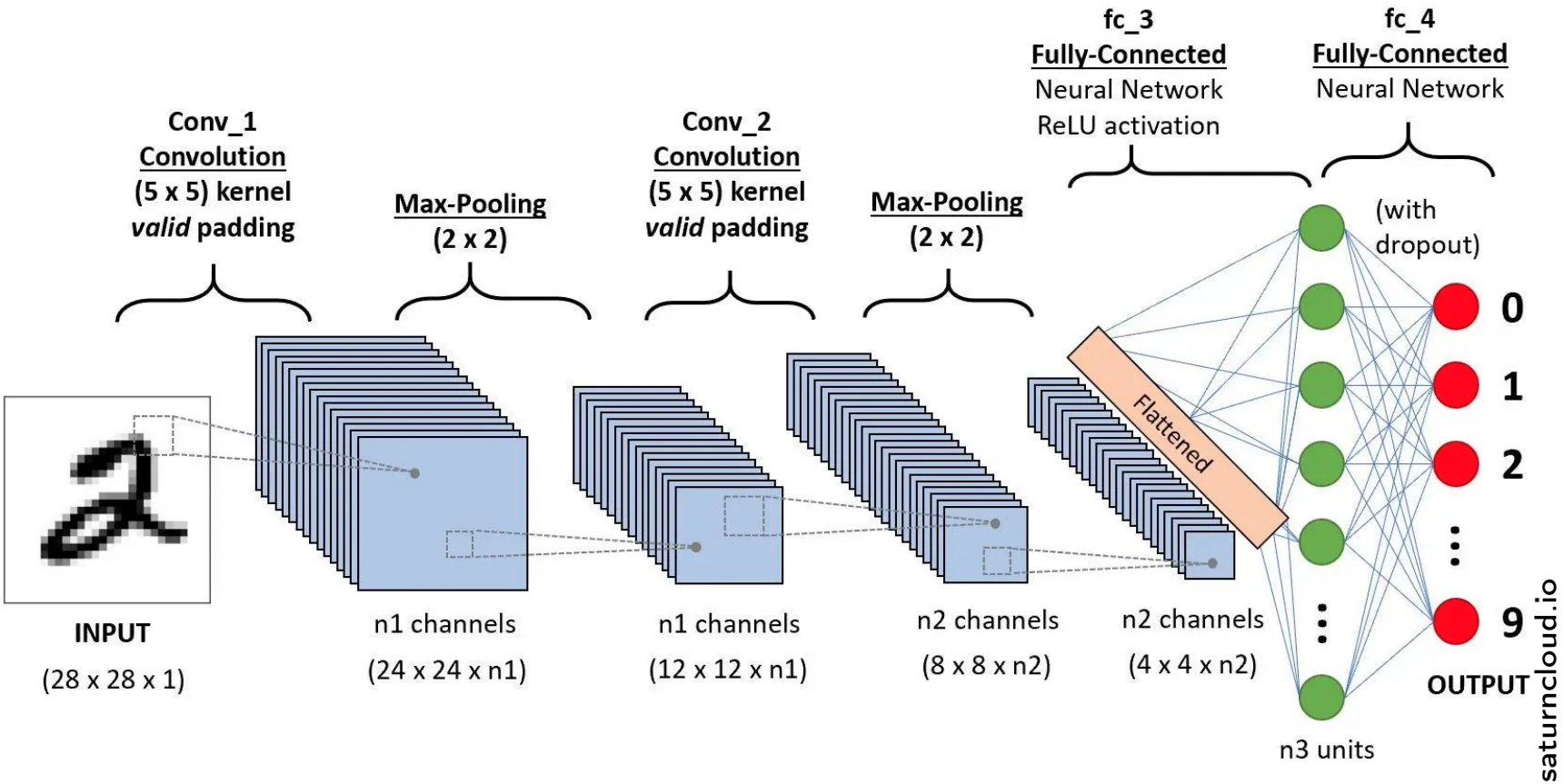
Pooling layer takes inputs from nearby parts of its input data. The inputs are small (e.g., 2x2) and non-overlapping.

CNN structure



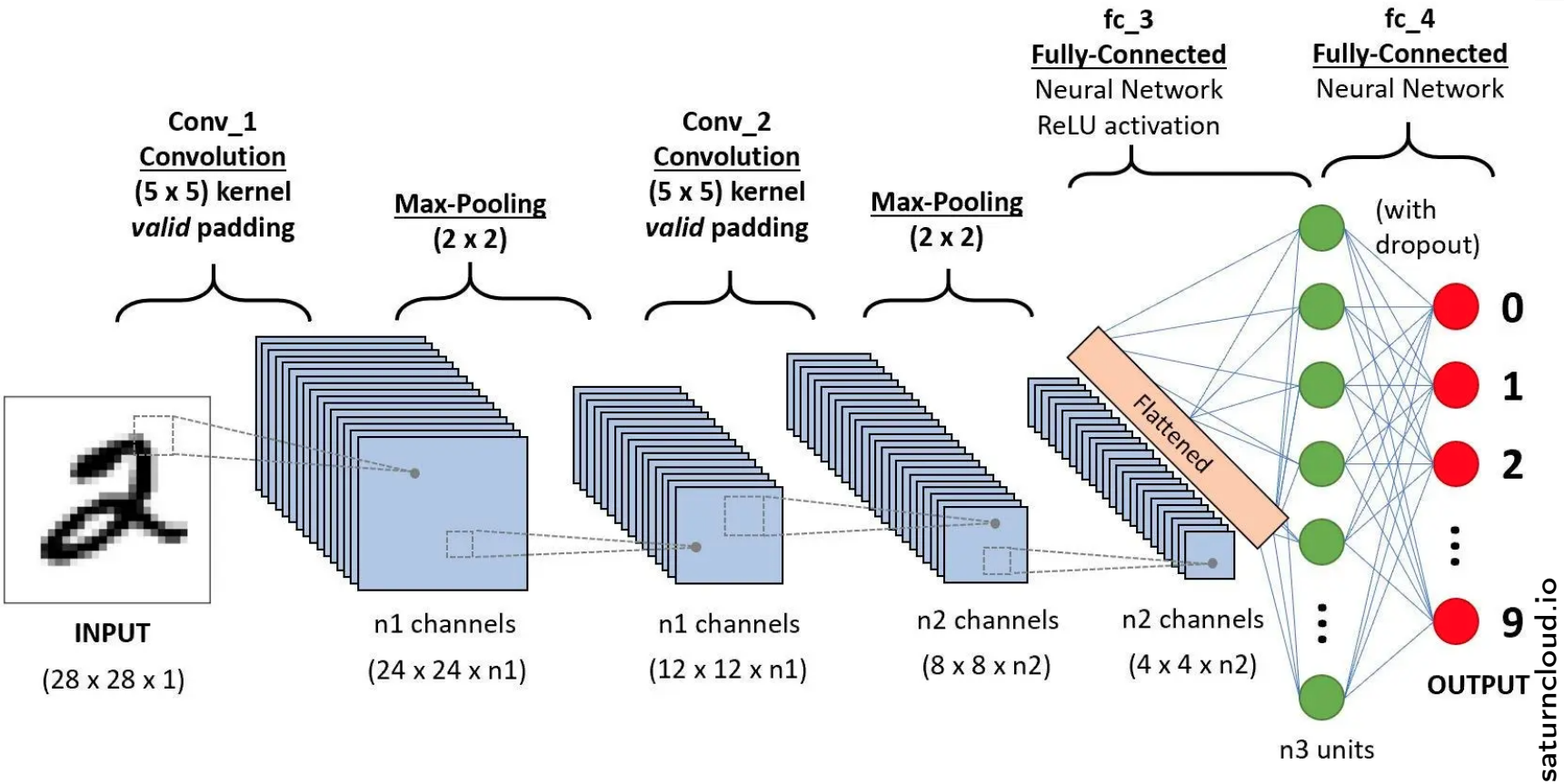
Pooling layer takes inputs from nearby parts of its input data. The inputs are small (e.g., 2x2) and non-overlapping. It does max() or average() operations.

CNN structure



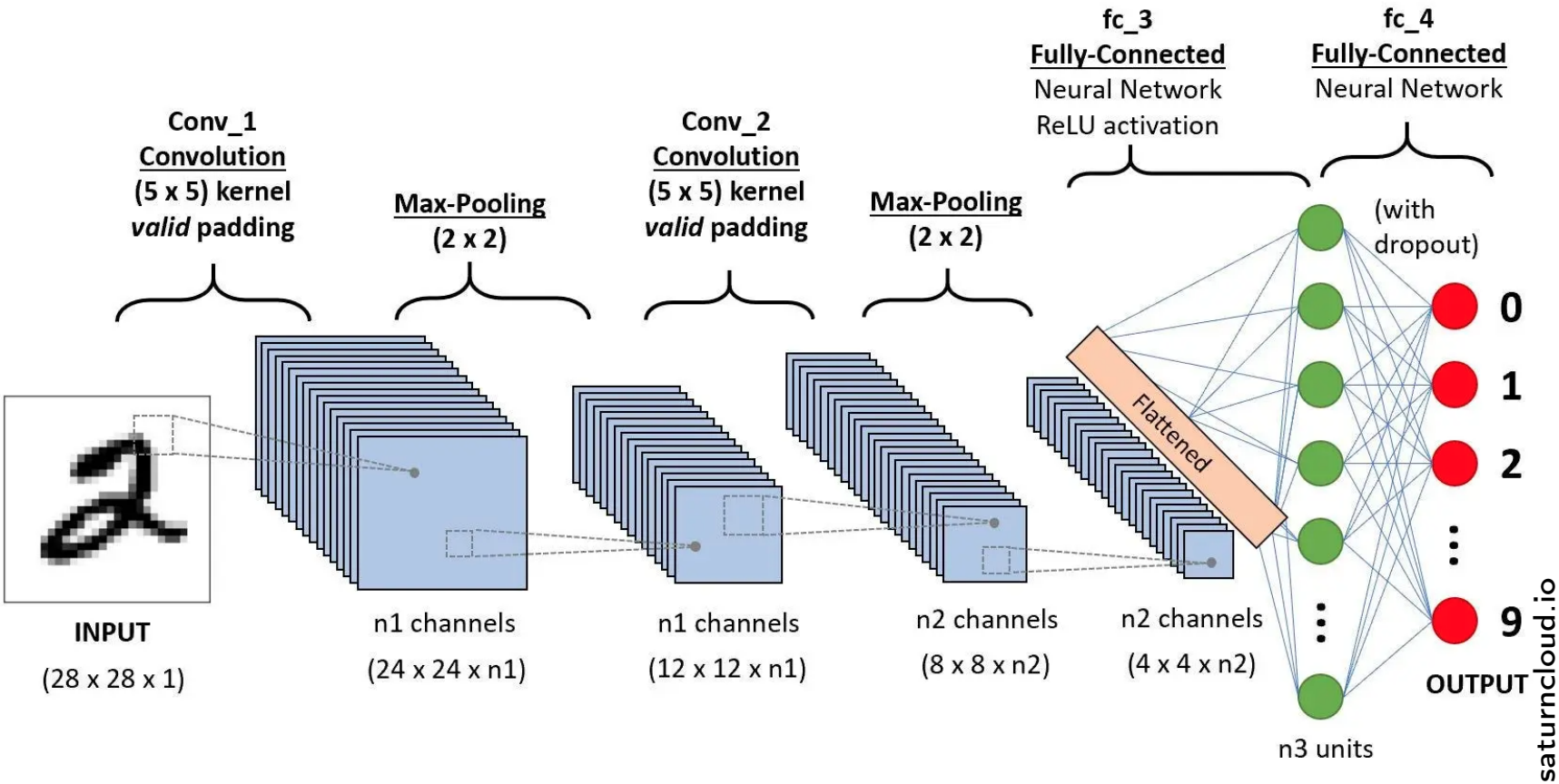
Pooling layer takes inputs from nearby parts of its input data. The inputs are small (e.g., 2x2) and non-overlapping. It does max() or average() operations. This makes invariance to small shifts of network input data.

CNN structure



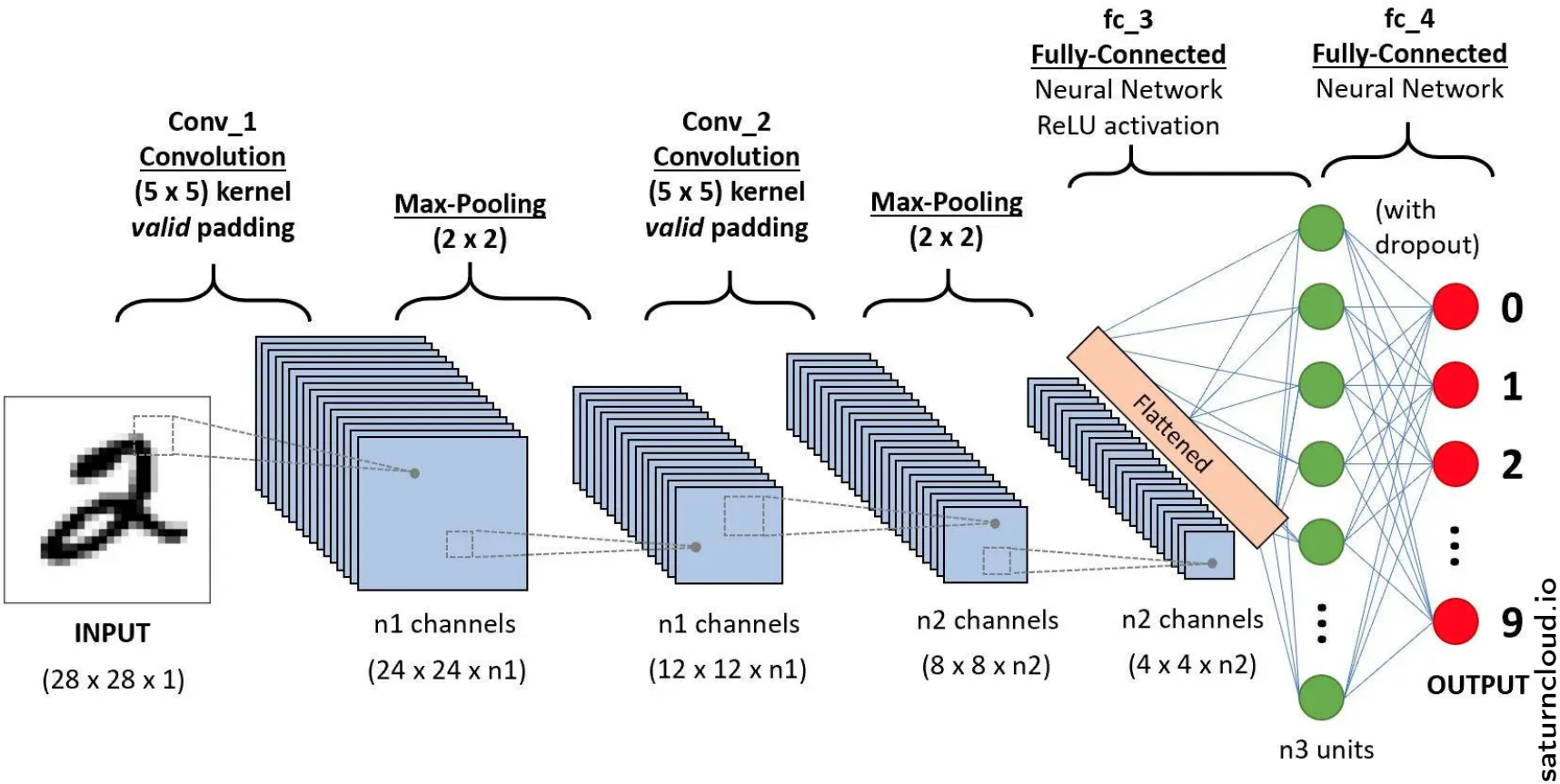
Pooling layer takes inputs from nearby parts of its input data. The inputs are small (e.g., 2x2) and non-overlapping. It does max() or average() operations. This makes invariance to small shifts of network input data. In successive steps, the spatial resolution is reduced but number of features extracted increases.

CNN structure



Pooling can be done from different feature maps, but from similar locations.

CNN structure



At the end typically there is a fully-connected feed-forward network that makes flattening on its input.

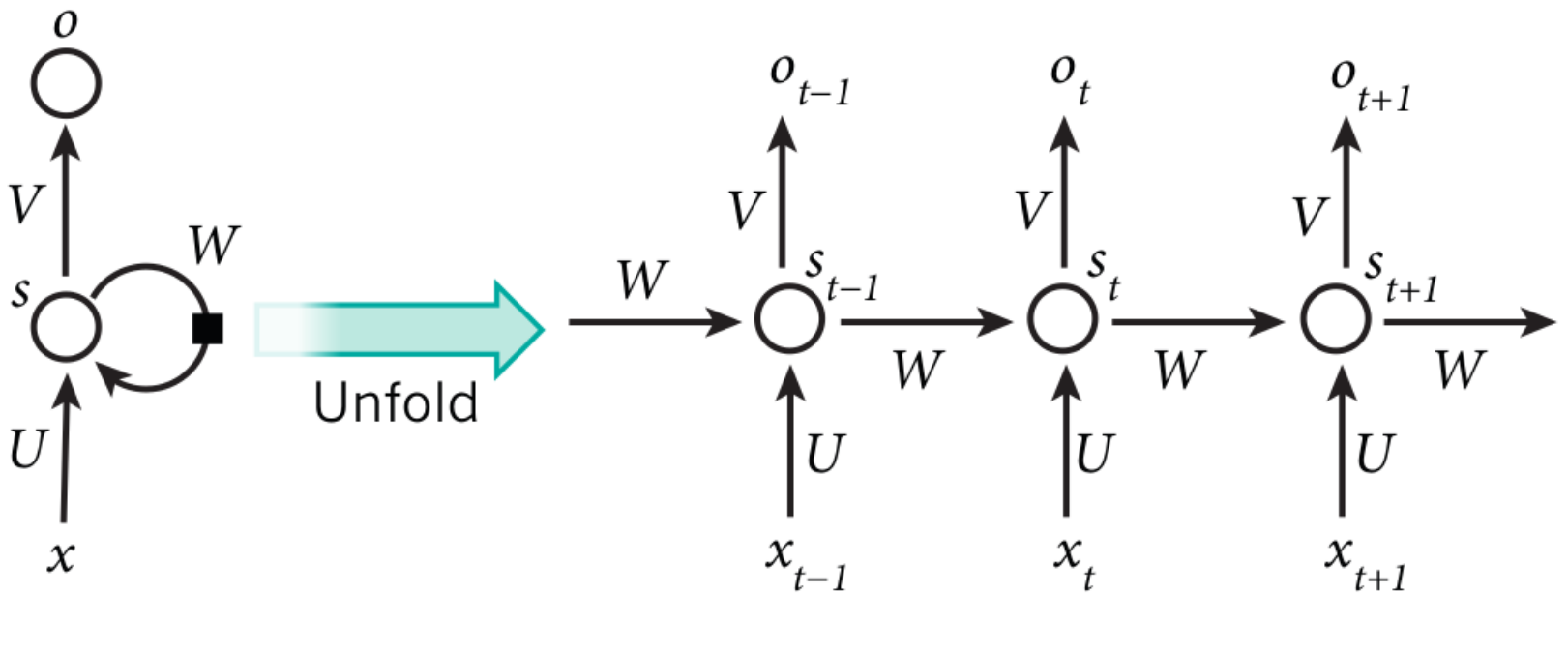
CNN – final notes

Key ideas behind CNNs:

- local connections,
- shared weights,
- pooling,
- use of many layers.

LeCun+15

Recurrency



LeCun+15

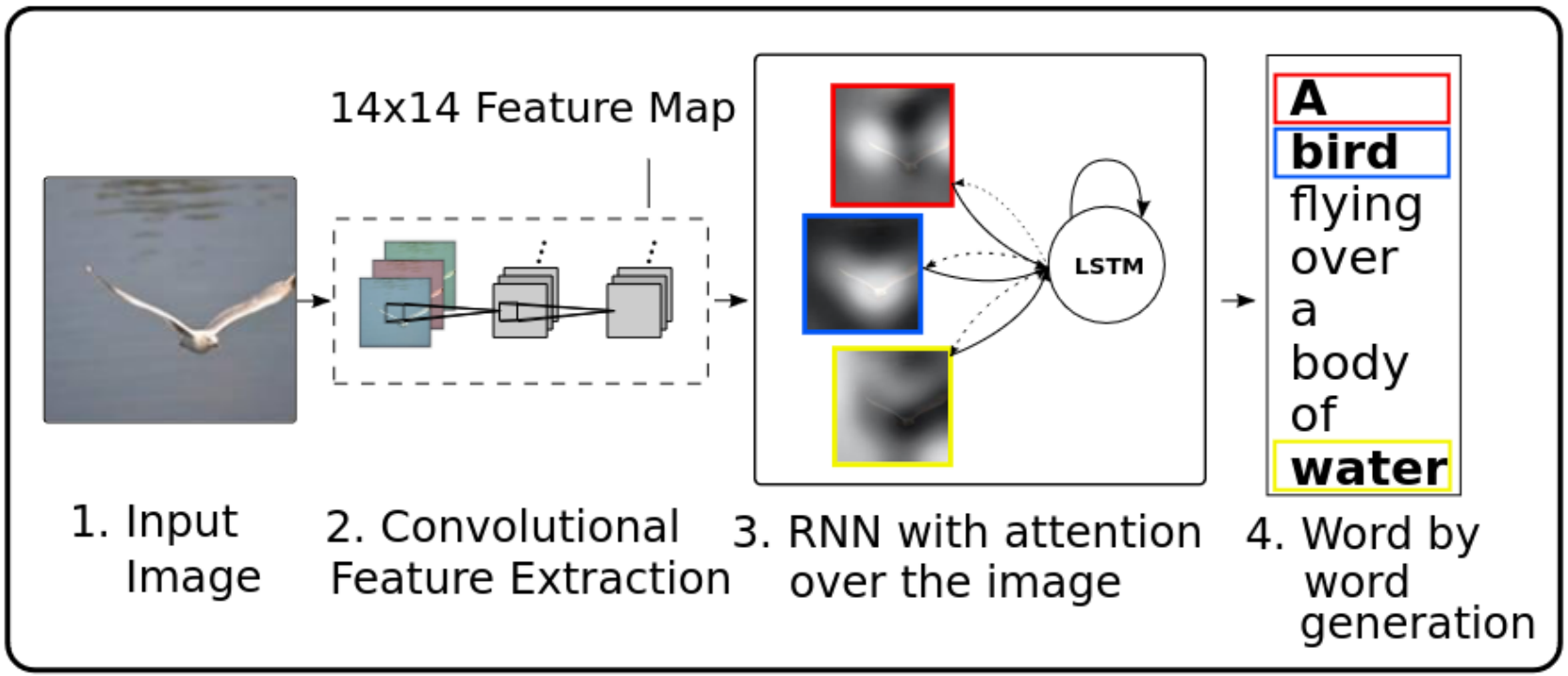
U, V, and W are matrixes.

Hard to train using backpropagation.

Memory can be implemented.

Long Short-Term Memory (LSTM)
Gated Recurrent Unit (GRU)

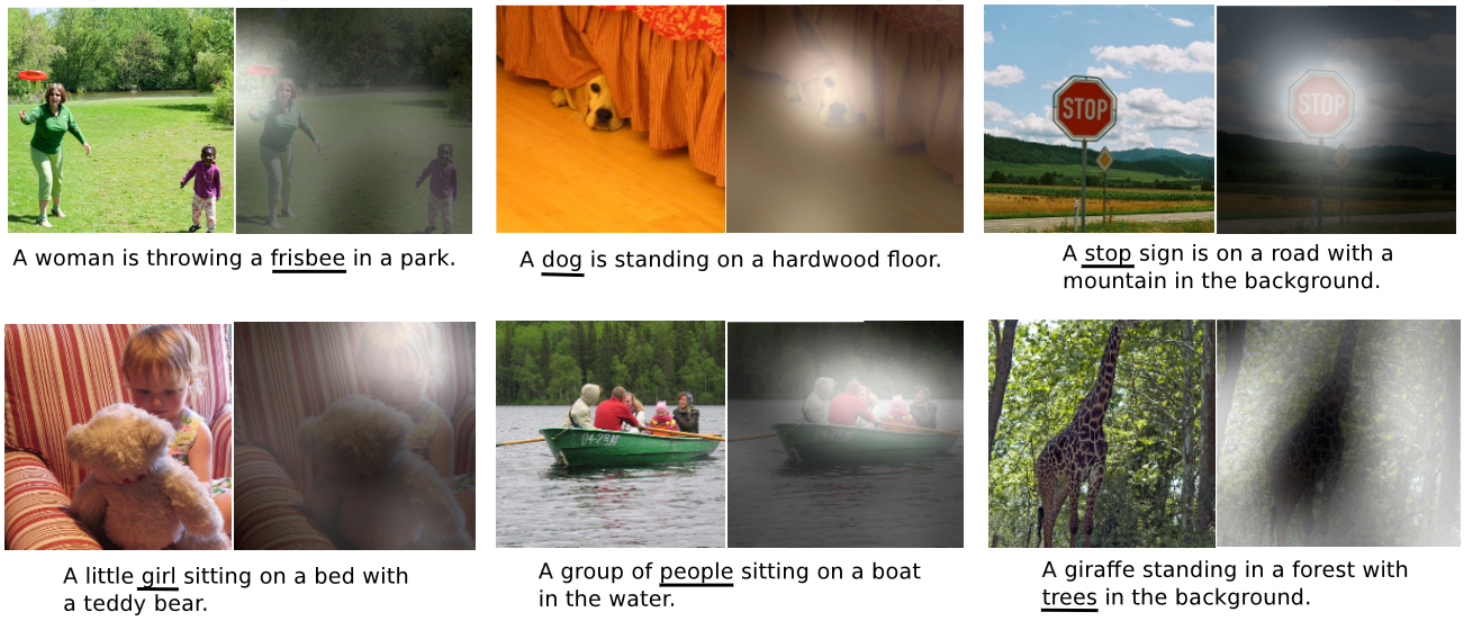
Network Structure



Xu+15 1502.03044

Successes

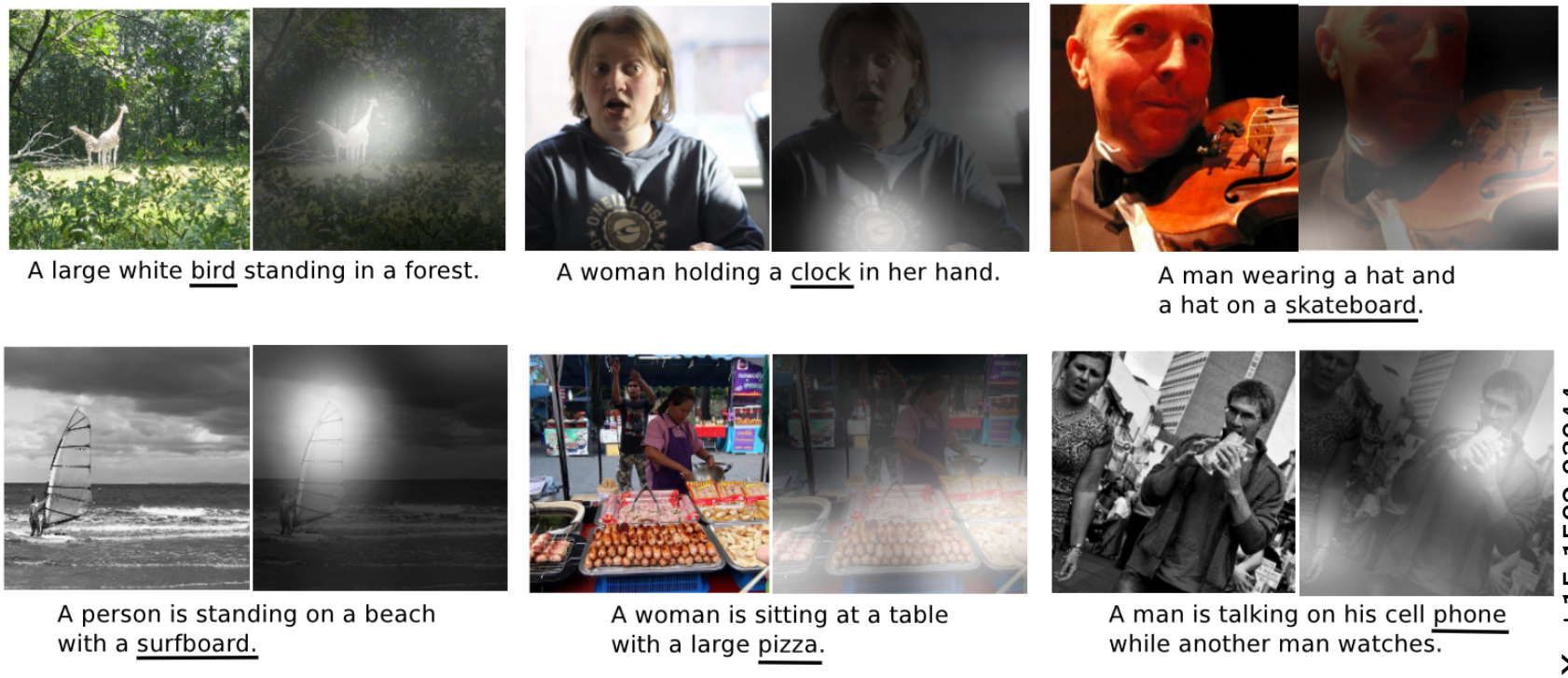
Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



Xu+15 1502.03044

Failures

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.



Xu+15 1502.03044

Last slide

- Feed-forward
- Fully-connected
- Features
- Activation functions
- Backpropagation
- Deep learning
- CNN
- RNN