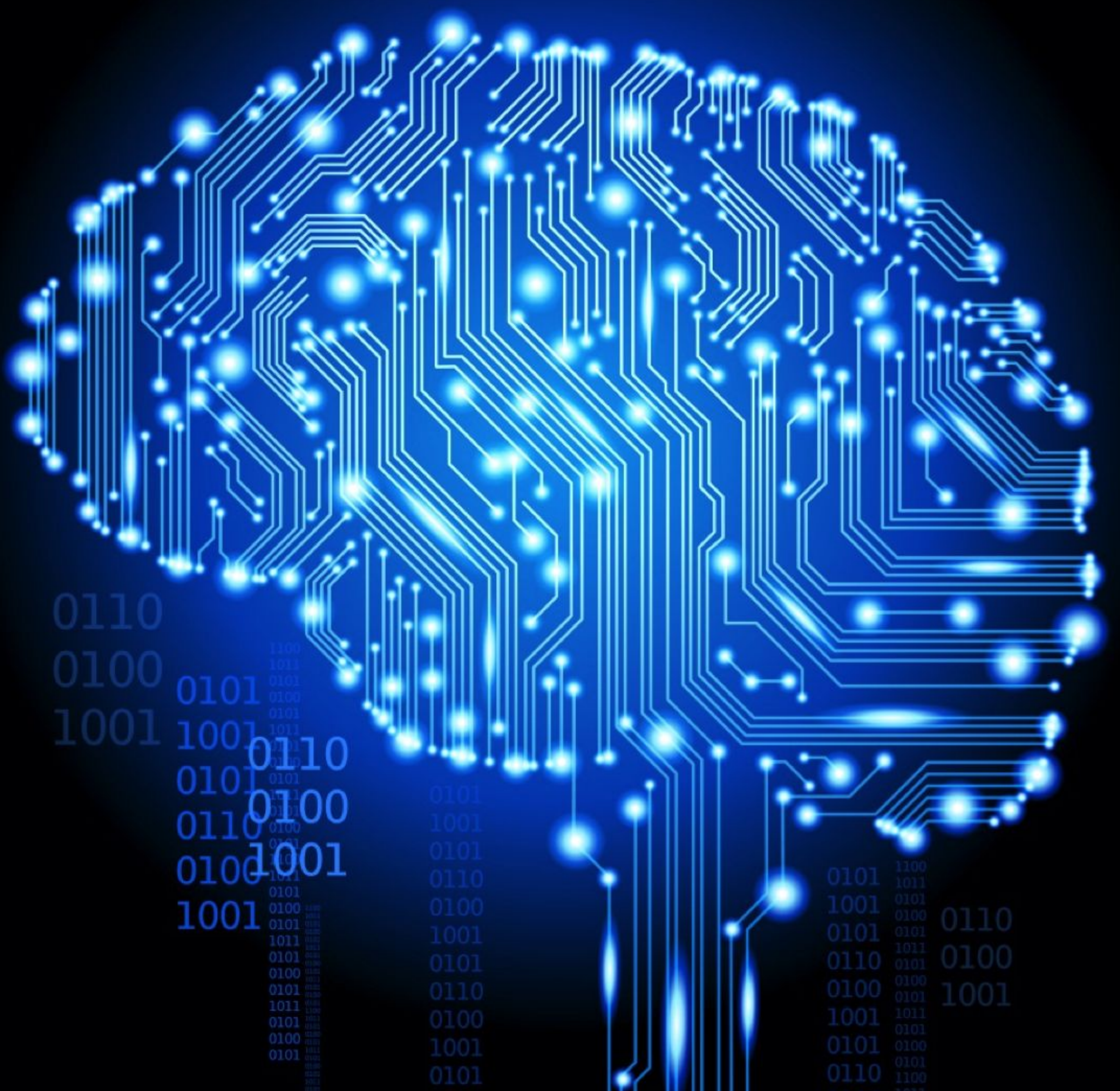
Deep Learning

ESADE - MIBA (FALL 2017)



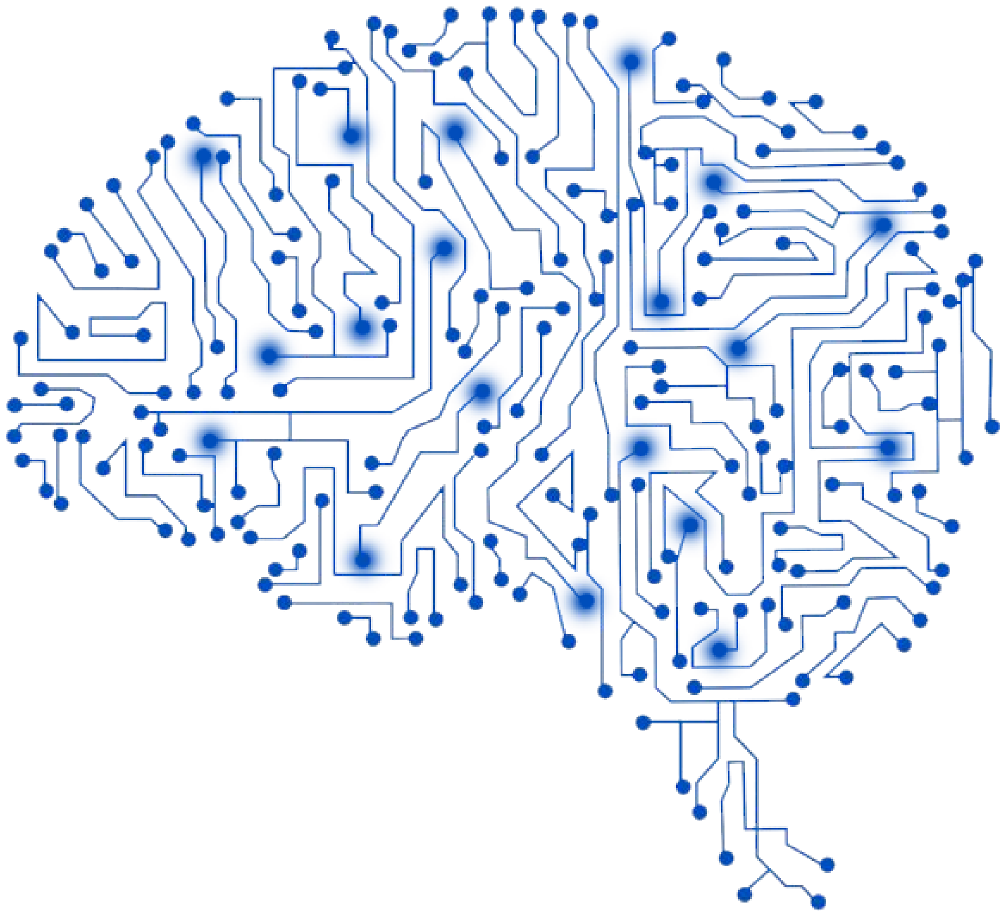
JORDI **TORRES** | FRANCESC **SASTRE**

Summary

1. What is Deep Learning?
2. Neural Networks
3. Loss and optimization functions
4. ConvNets
5. Train

Deep Learning

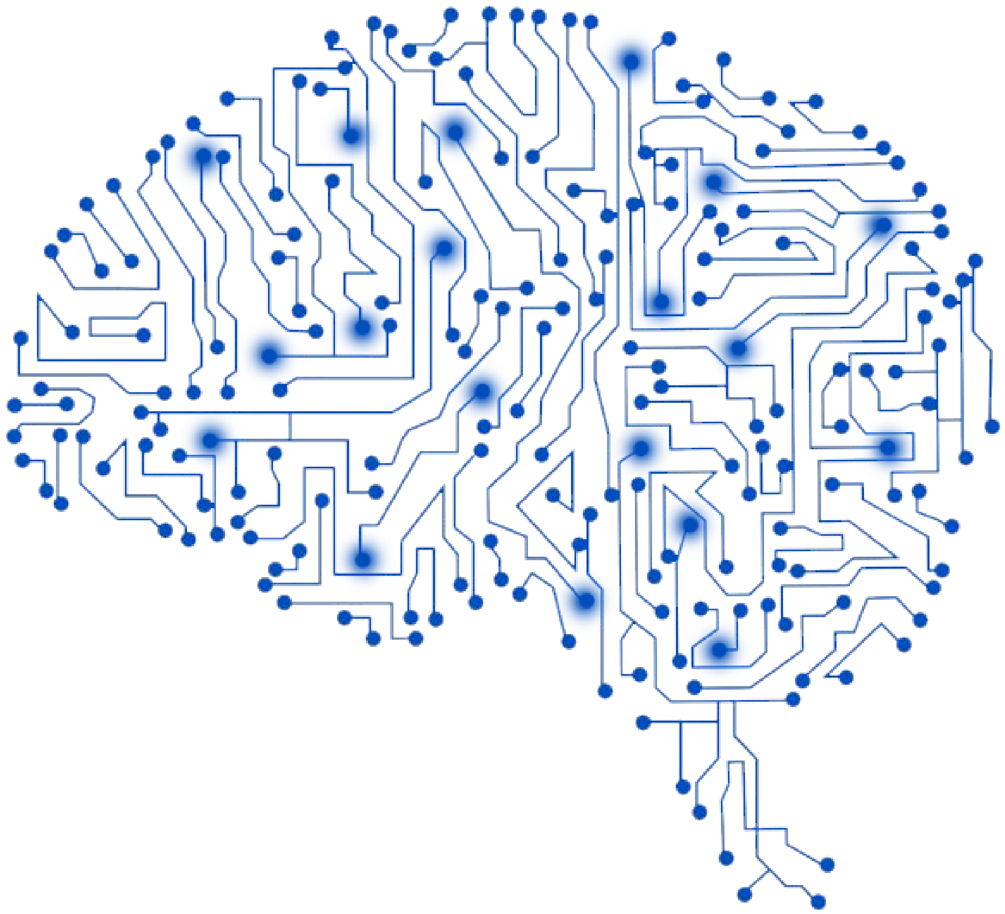
- Allows models to learn representations of data with multiple levels of abstraction
- Discovers intricate structure in large data sets (Patterns)
- Dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, …



Deep learning. In: Nature 521.7553 (May 2015). Yann LeCun et al.

Deep Learning

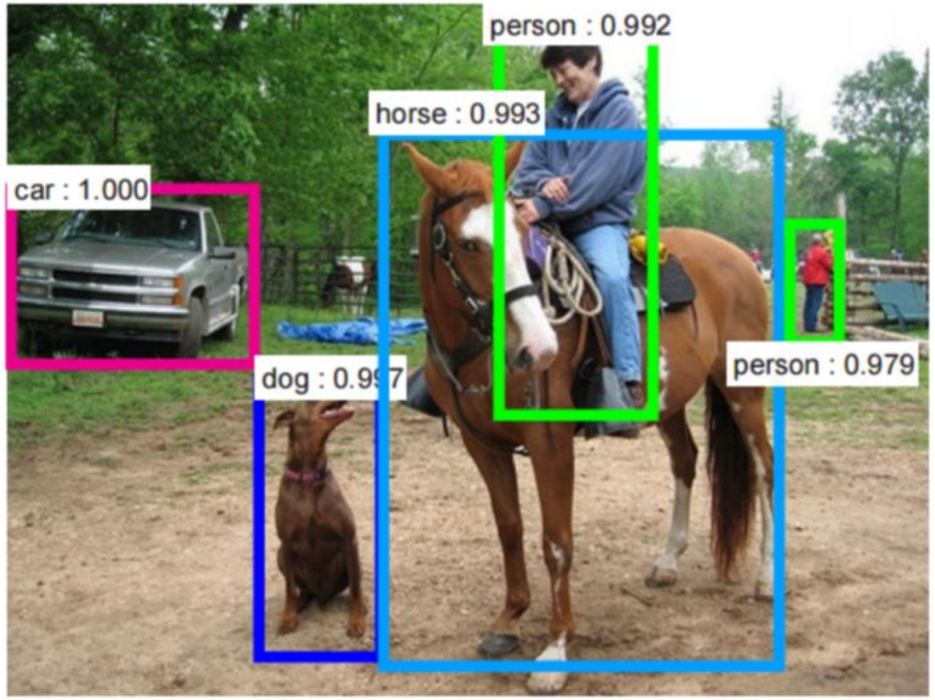
- **Supervised Learning**
  - **Training data is labeled**
  - **Goal is correctly label new data**
- Reinforcement Learning
  - Training data is unlabeled
  - System receives feedback for its actions
- Goal is to perform better actions
  - Unsupervised Learning
  - Training data is unlabeled
  - Goal is to categorize the observations



Source: Nvidia Research

Deep Learning

ImageNet classification. (Source: Alex Krizhevsky et al.)



Object detection and classification. (Source: Shaoqing Ren et al.)

Deep Learning

Image Colorization. (Source: Richard Zhang et al.)

Images captioning. (Source: Andrej Karpathy et al.)

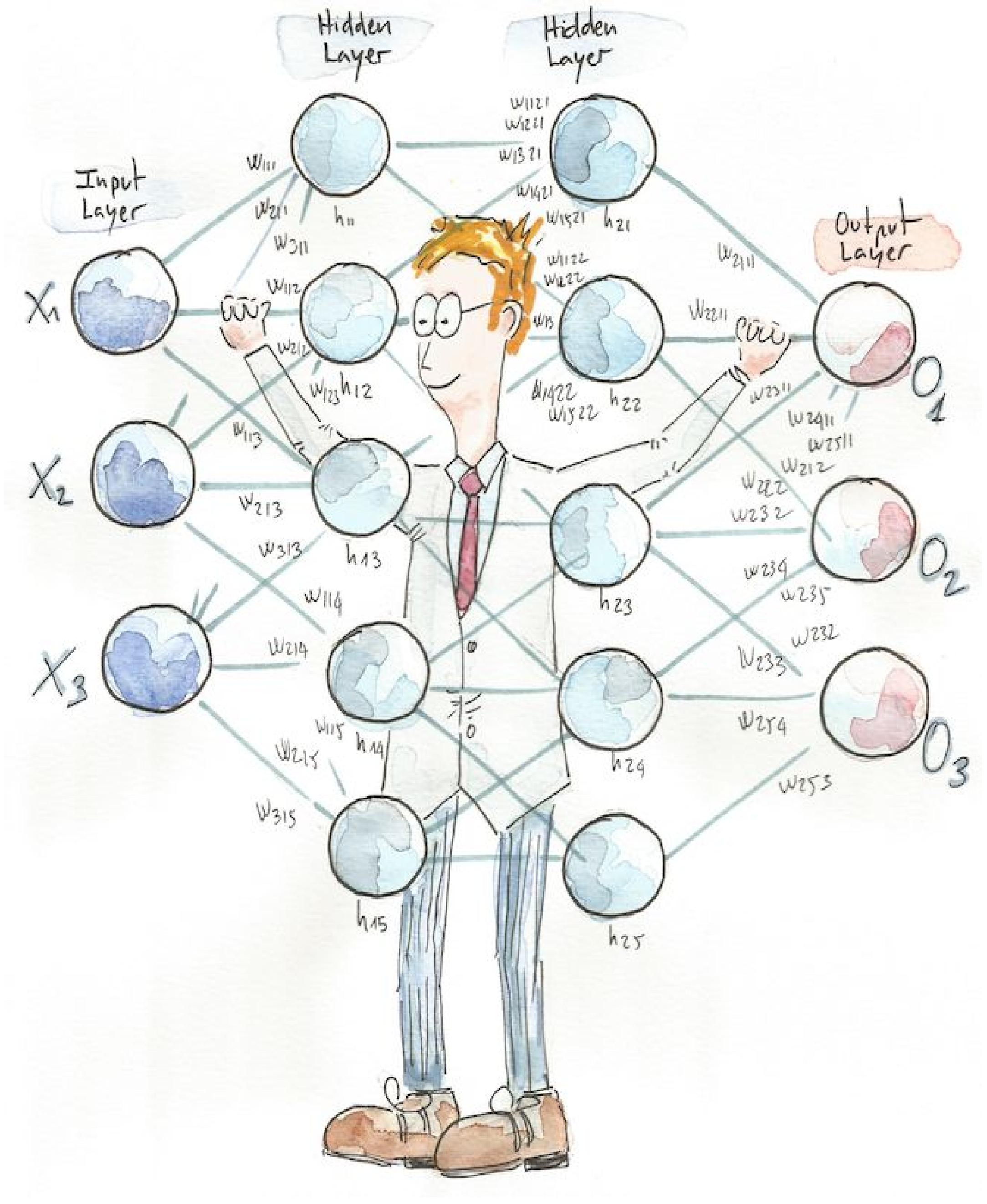
"baseball player is throwing ball in game."

"woman is holding bunch of bananas."

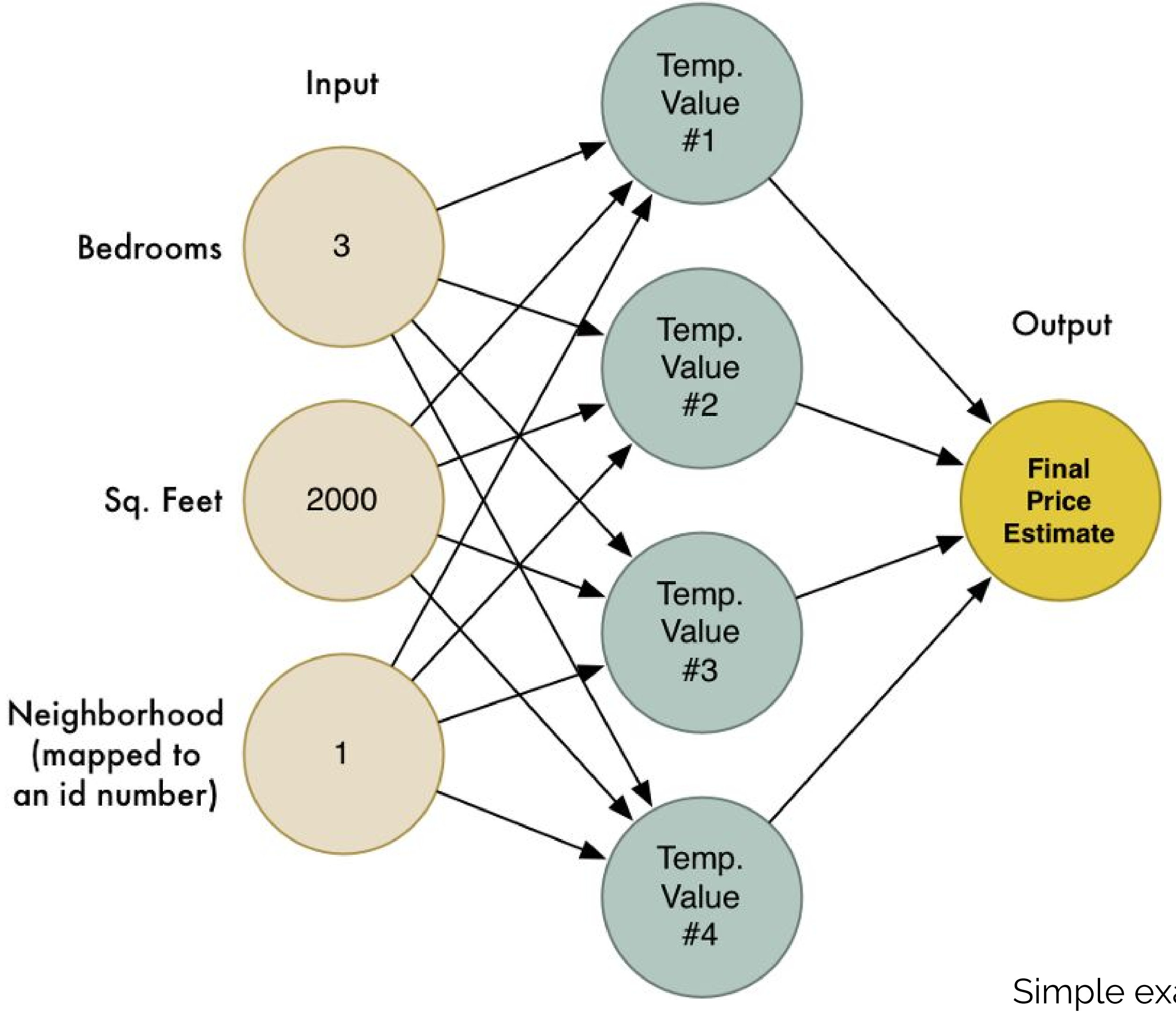
"black cat is sitting on top of suitcase."

Neural Networks

- Set of neurons
- Each neuron contains an activation function
- Different topologies
- The connections are the inputs and outputs of the functions
- Each connection has a weight and bias

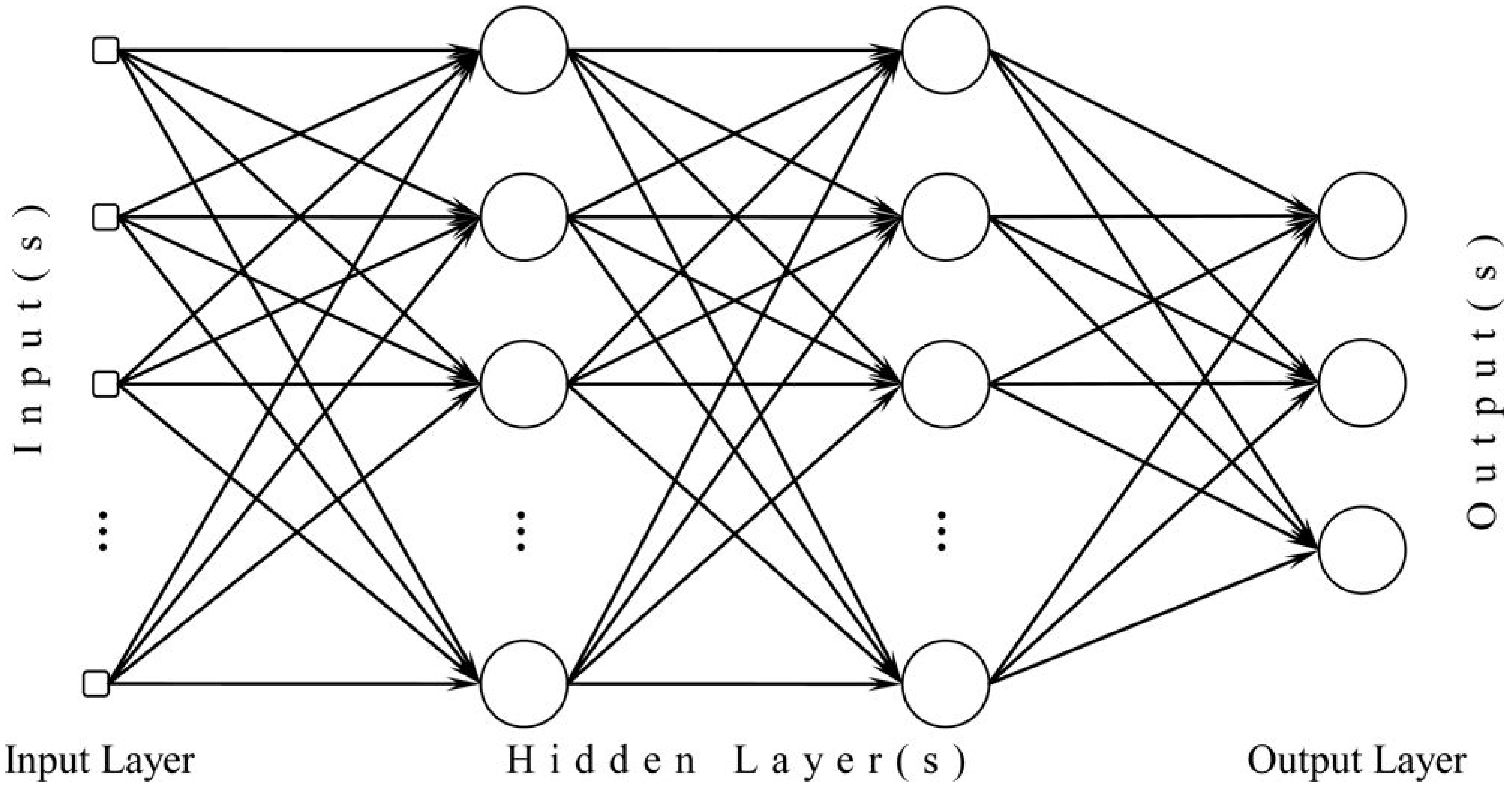


Neural Networks

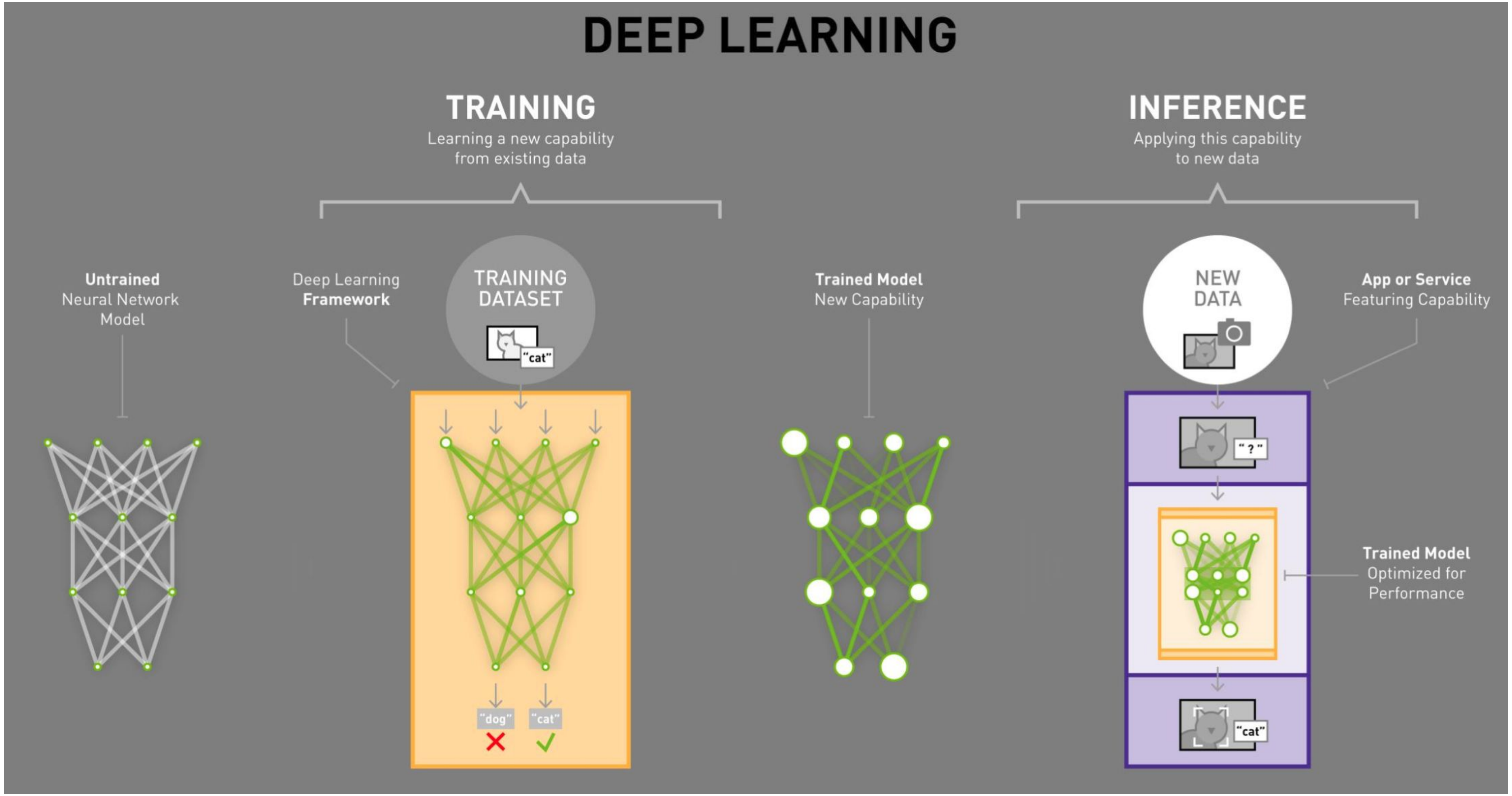


Simple example. (Source: Adam Geitgey)

Neural Networks



Neural Networks



Source: Nvidia Research

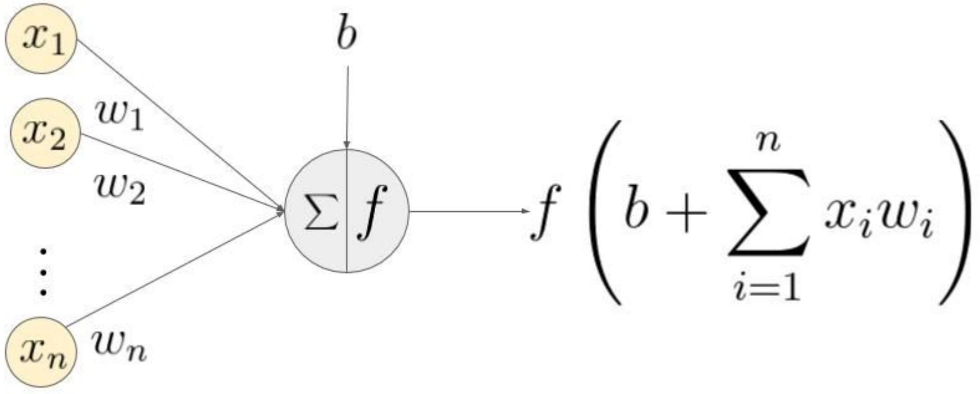
Neurons

- Inputs:
  - Outputs from other neurons
  - Input data
- Each input has a different weight
- One output
- Different activation functions

$$b + \sum_{i=1}^{n} x_i w_i$$

Neurons

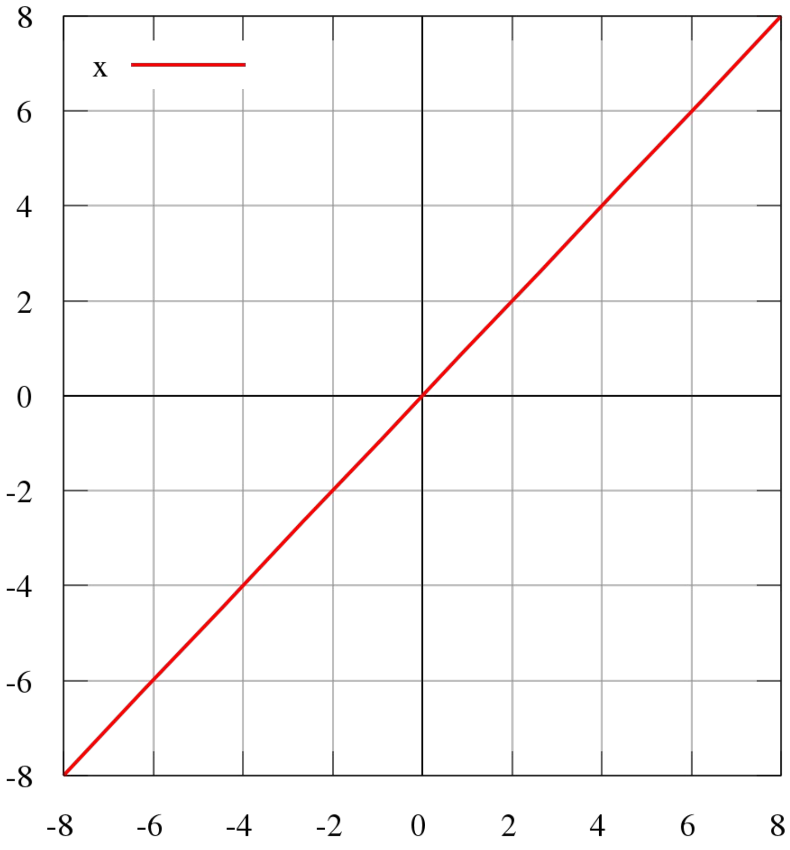
- Inputs:
  - Outputs from other neurons
  - Input data
- Each input has a different weight
- One output
- Different activation functions



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Activation functions

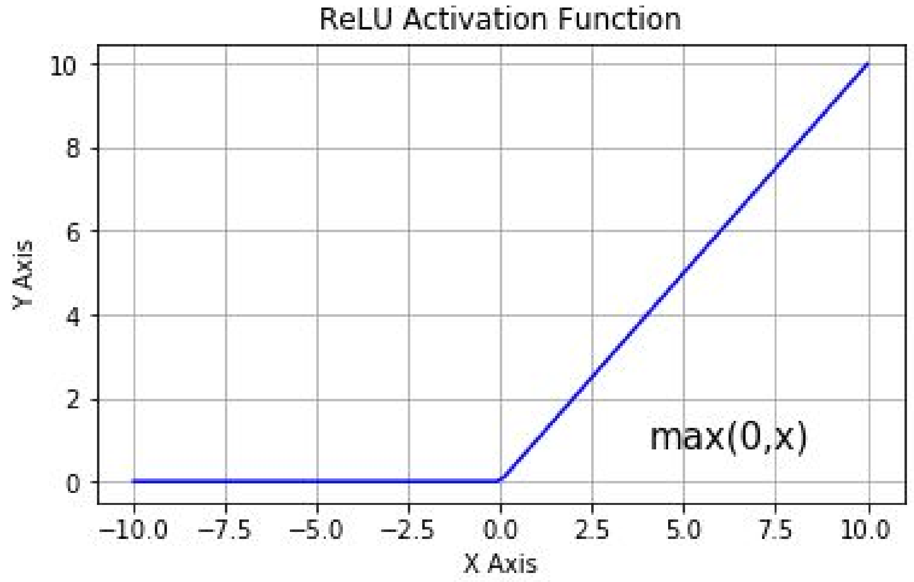
- **Linear functions**
  - Identity
- Non-linear functions

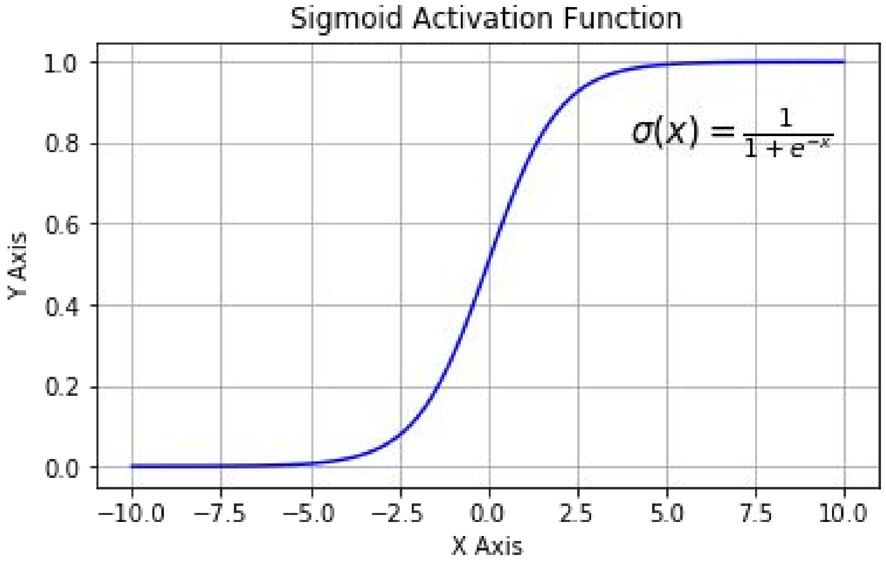


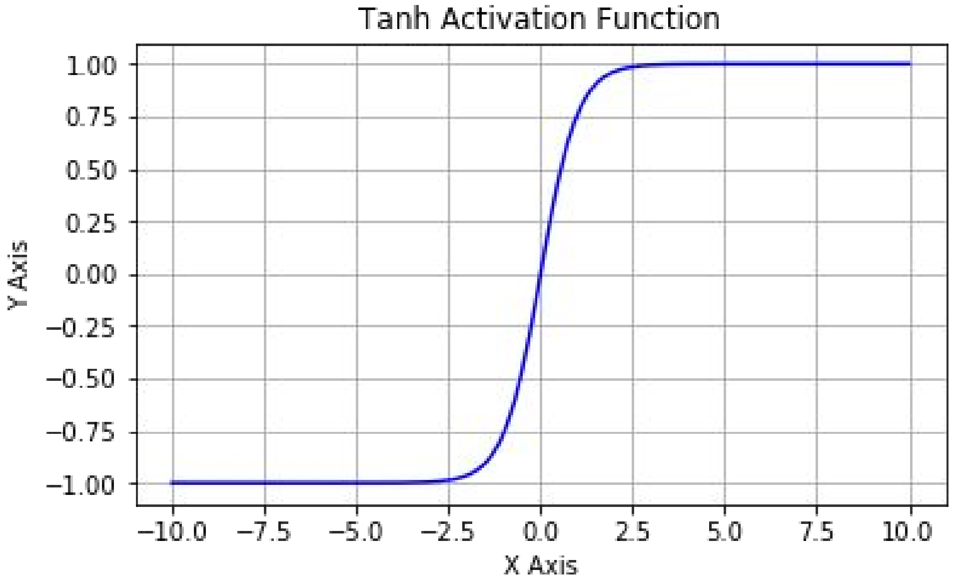
$$f(x) = x$$

Activation functions

- Linear functions
- **Non-linear functions**



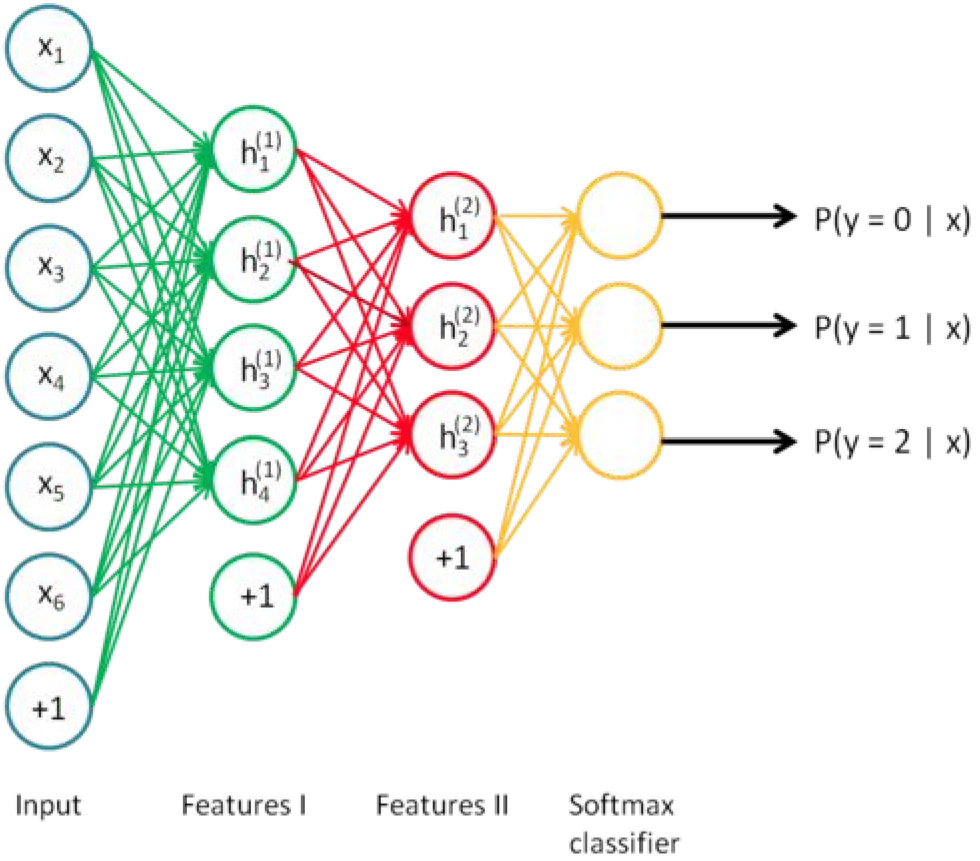




Source: Learn OpenCV

Activation functions

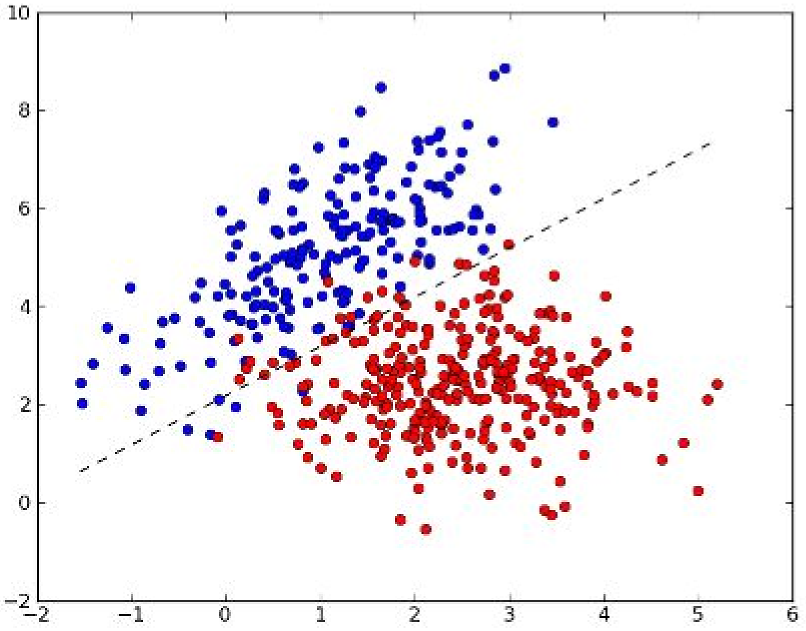
Softmax



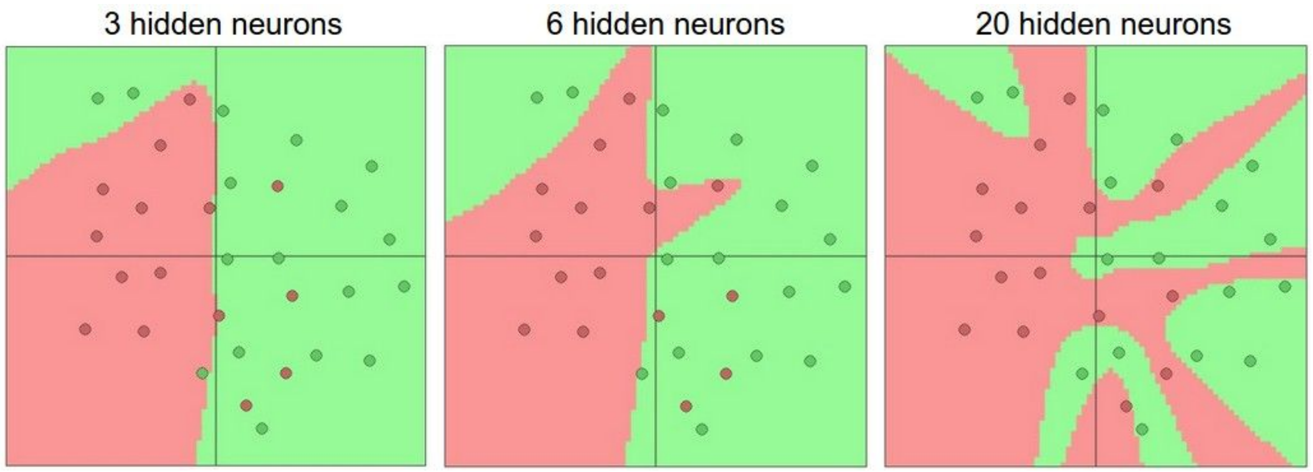
Source: Stanford Deep Learning

Activation functions

Linear

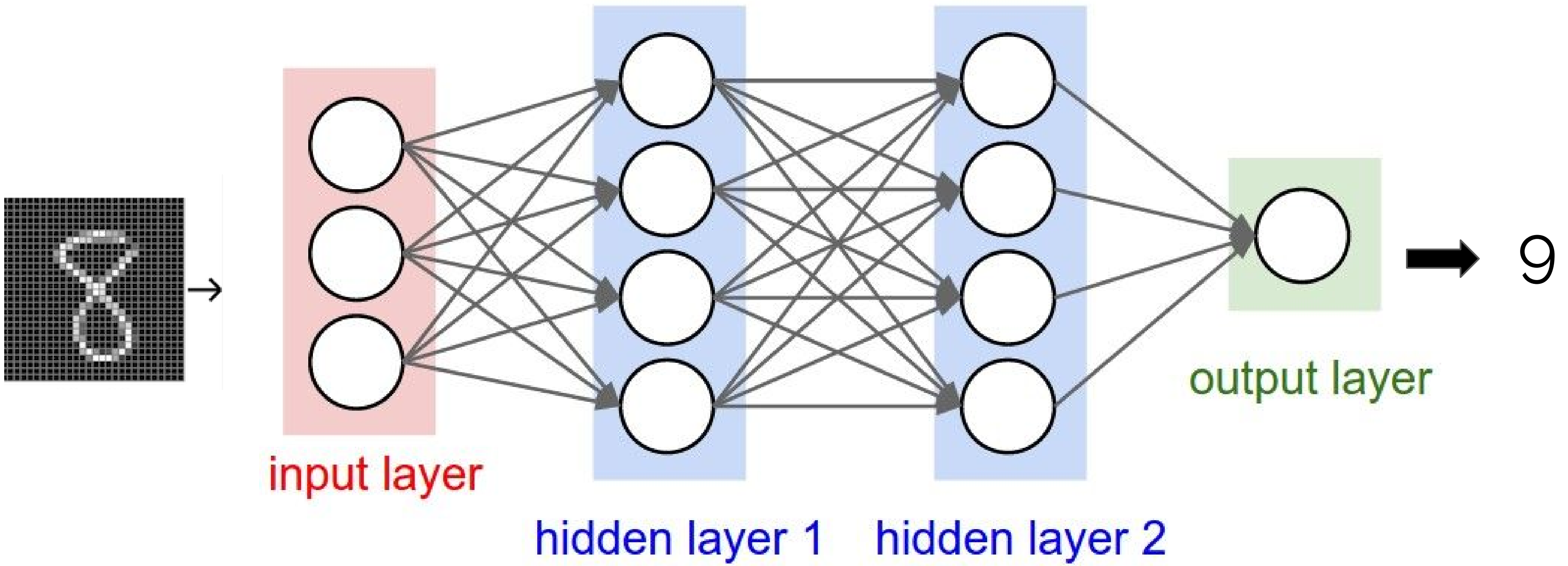


Non - Linear



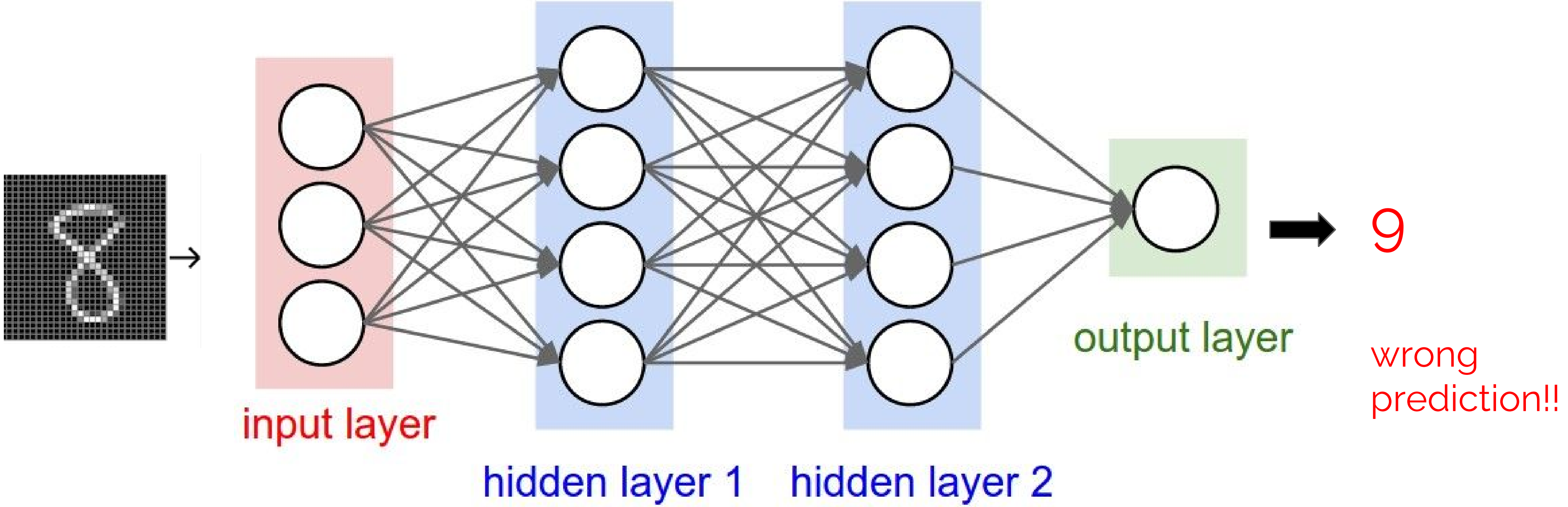
Source: Stanford cs231n

Loss function



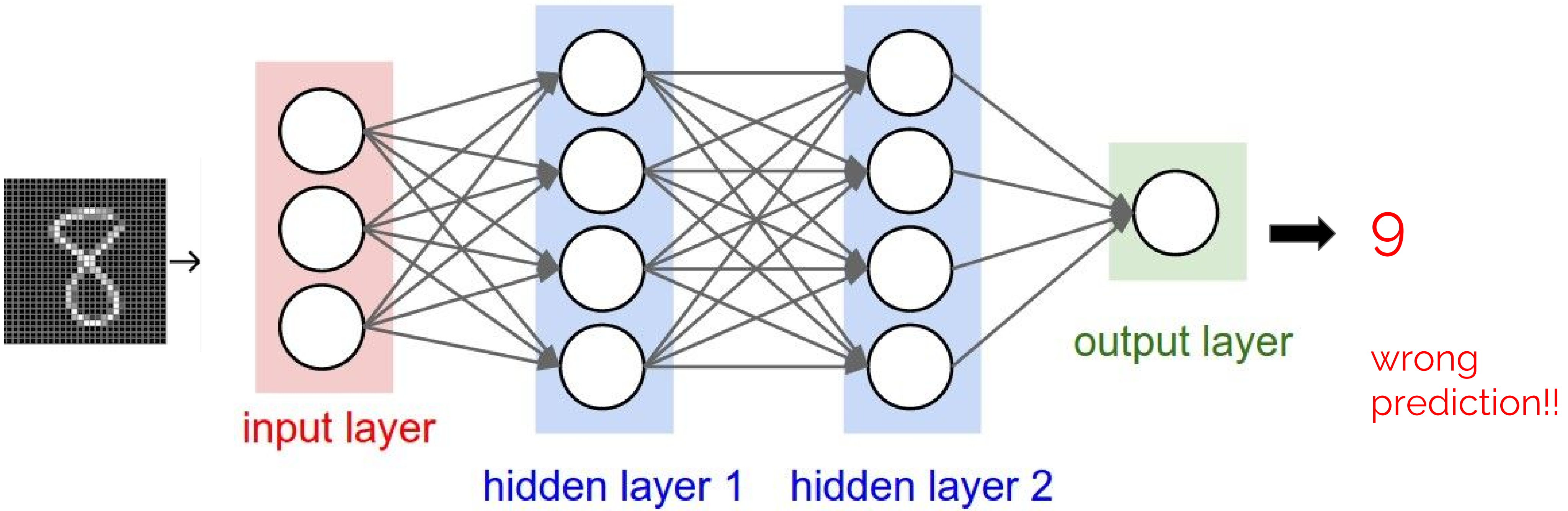
Source: Stanford cs231n

Loss function



Source: Stanford cs231n

Loss function

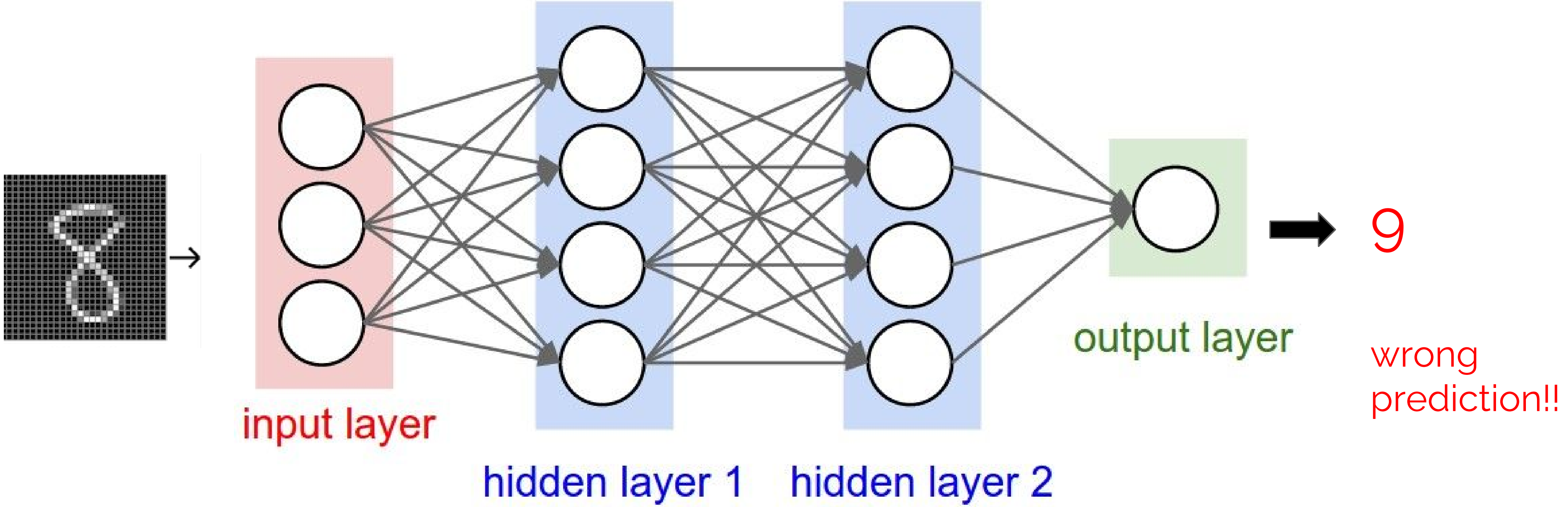


loss_function(target, output) → loss_function(8, 9)

Source: Stanford cs231n

Loss function

- Cross - entropy
- Mean squared error
- Binary Cross - entropy

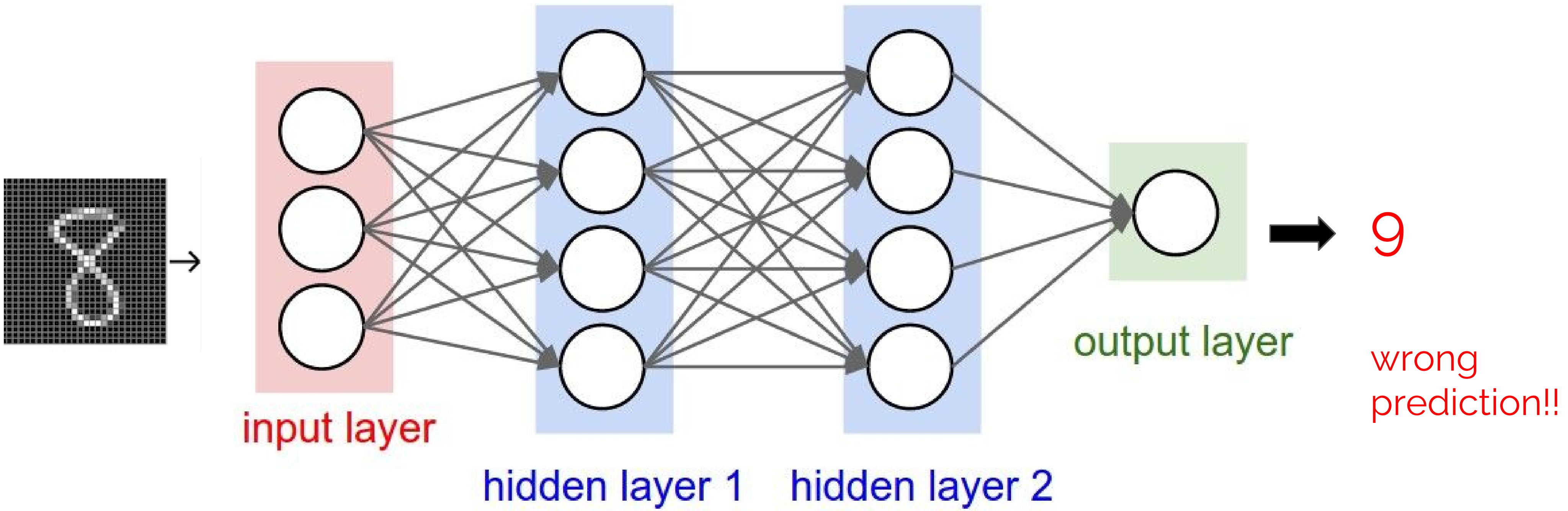


loss_function(target, output) → loss_function(8, 9)

Source: Stanford cs231n

Loss function

- Cross - entropy
- Mean squared error
- Binary Cross - entropy



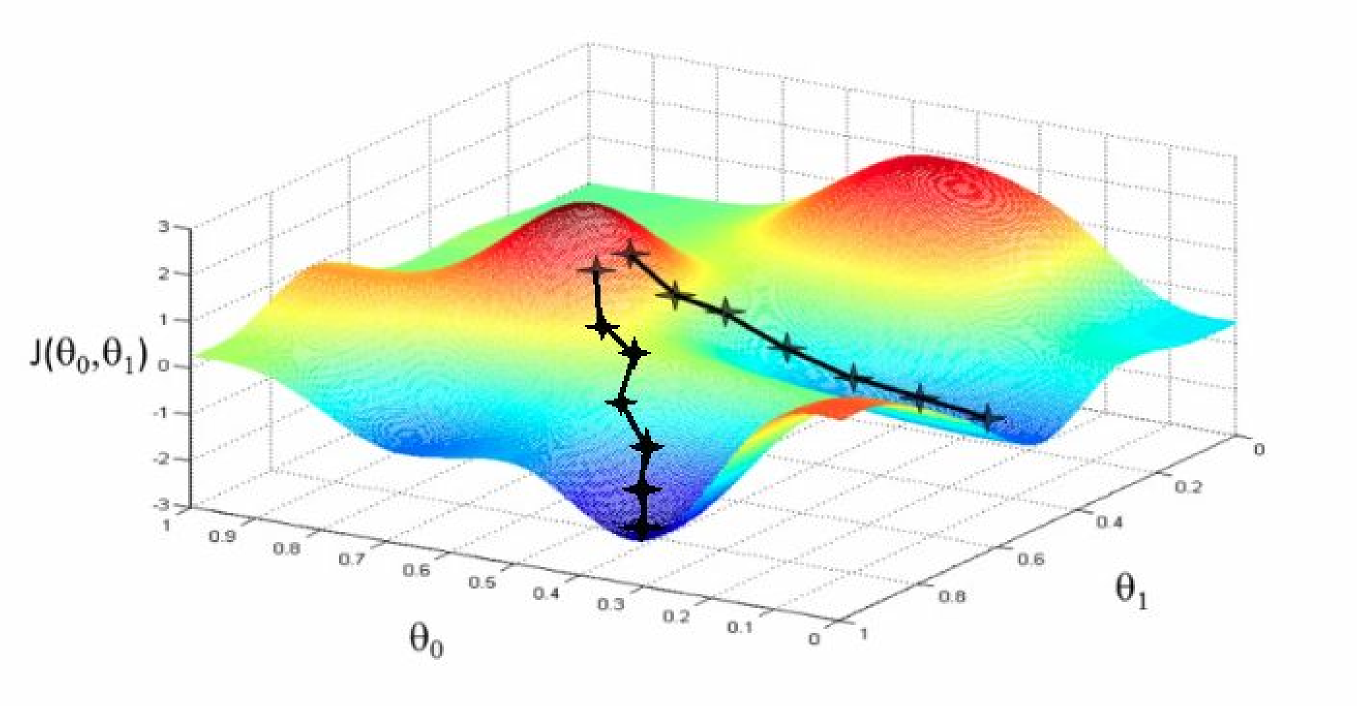
loss_function(target, output) → loss_function(8, 9)

Minimize it!

Source: Stanford cs231n

Optimization functions

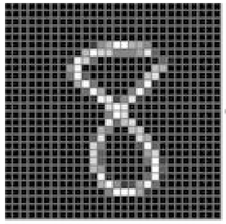
- Adagrad
- Adadelta
- Gradient descent



Source: Vasilis Vryniotis

Inputs

- **Input layer 2D**

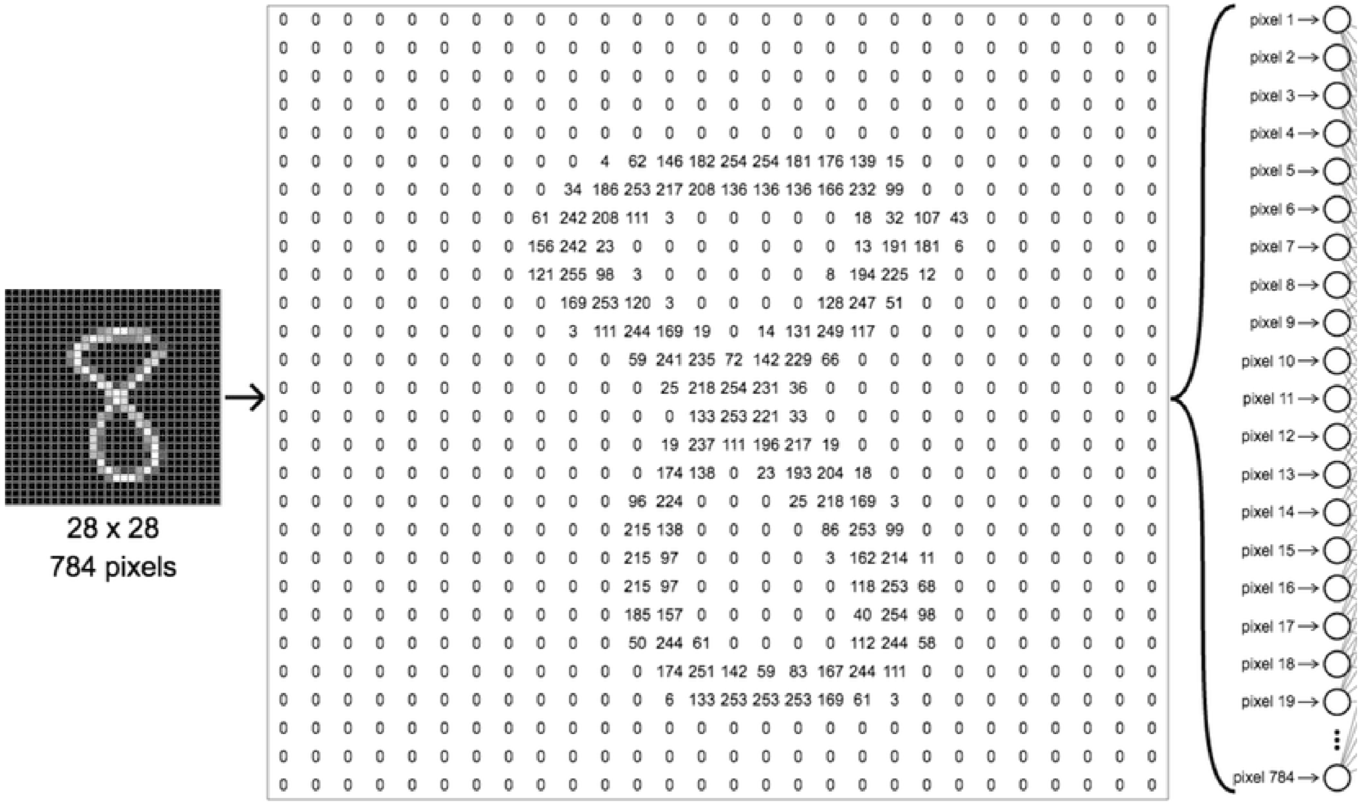


28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

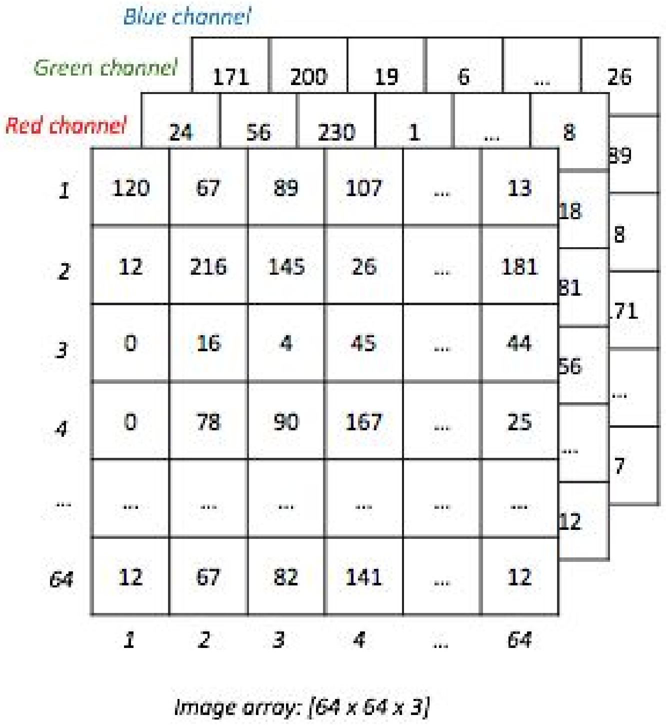
Inputs

- **Input layer 1D**



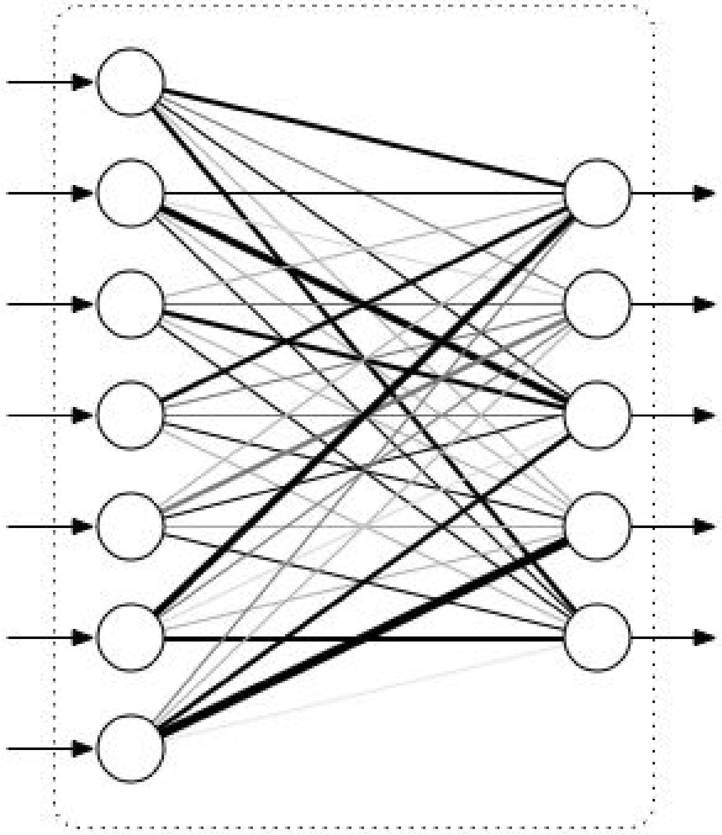
Inputs

- **Input layer 3D**



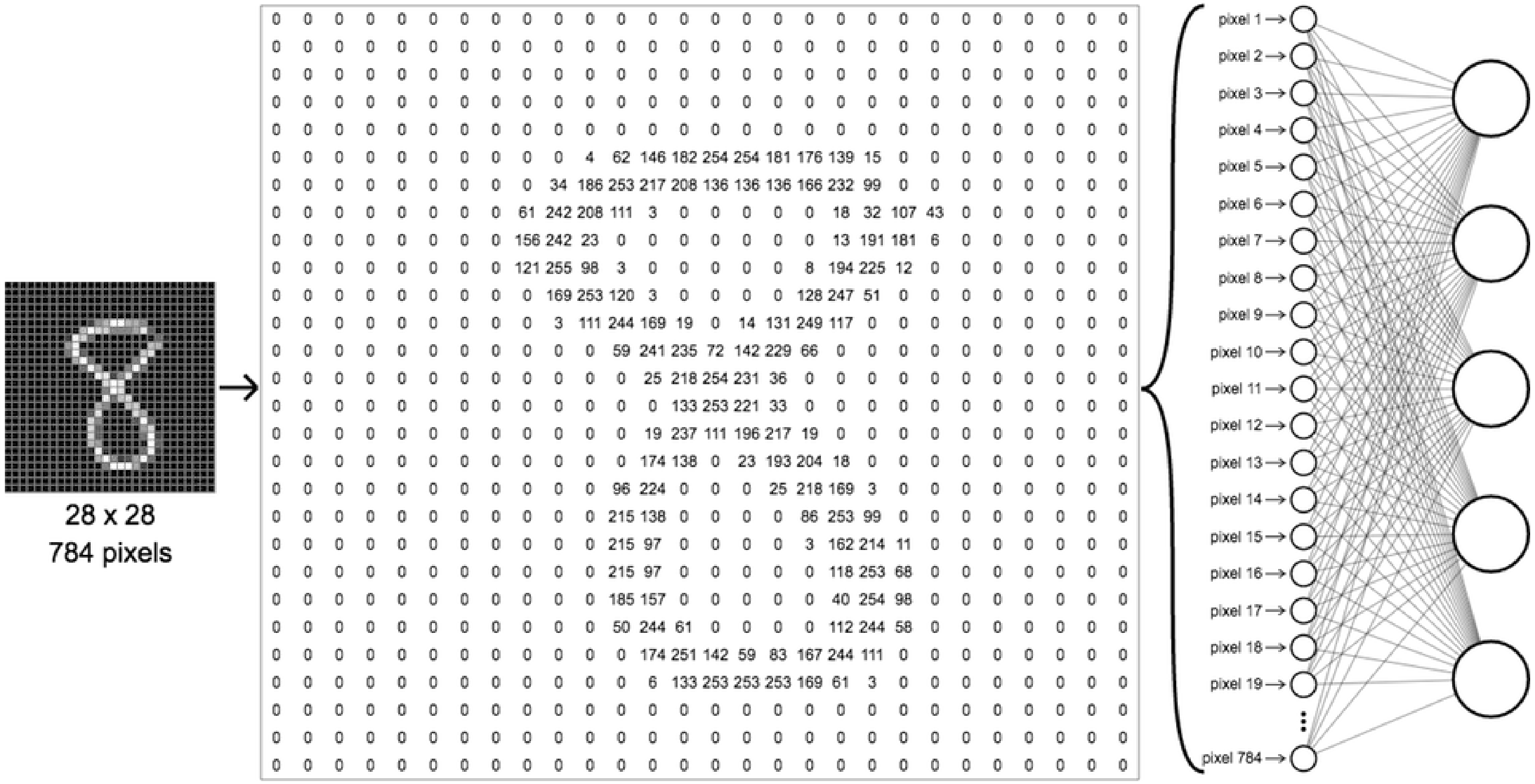
Layers

- **Fully connected | Dense**



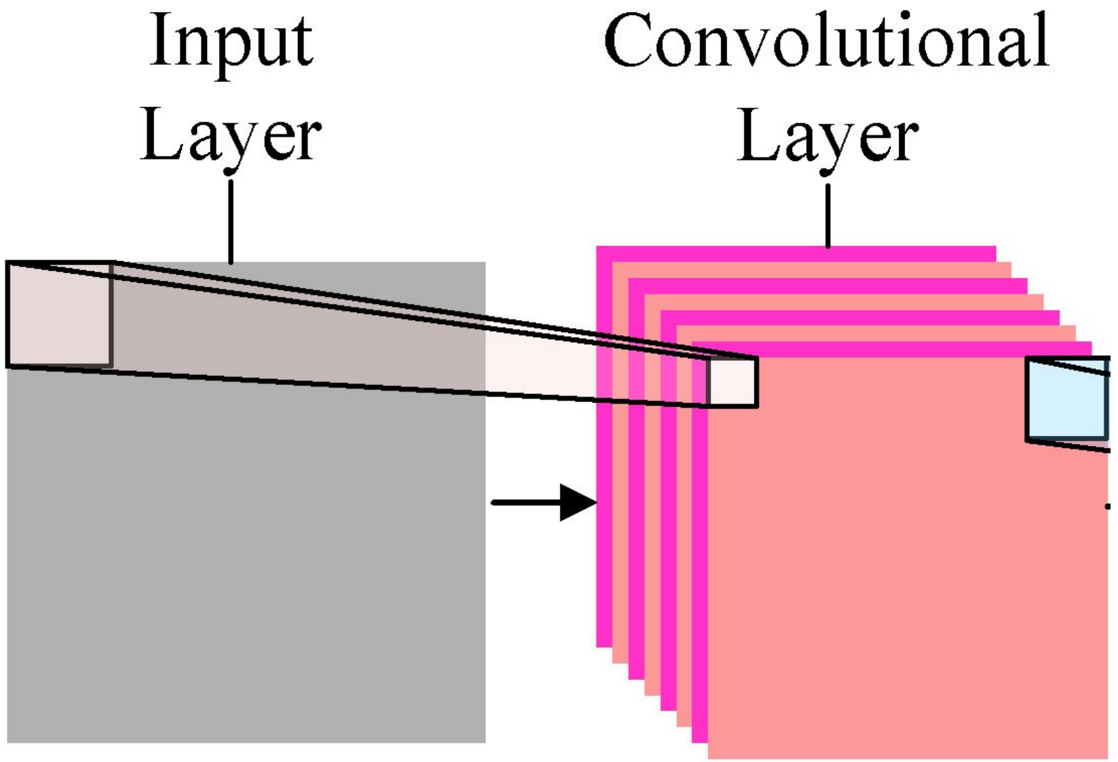
Layers

- **Flatten**



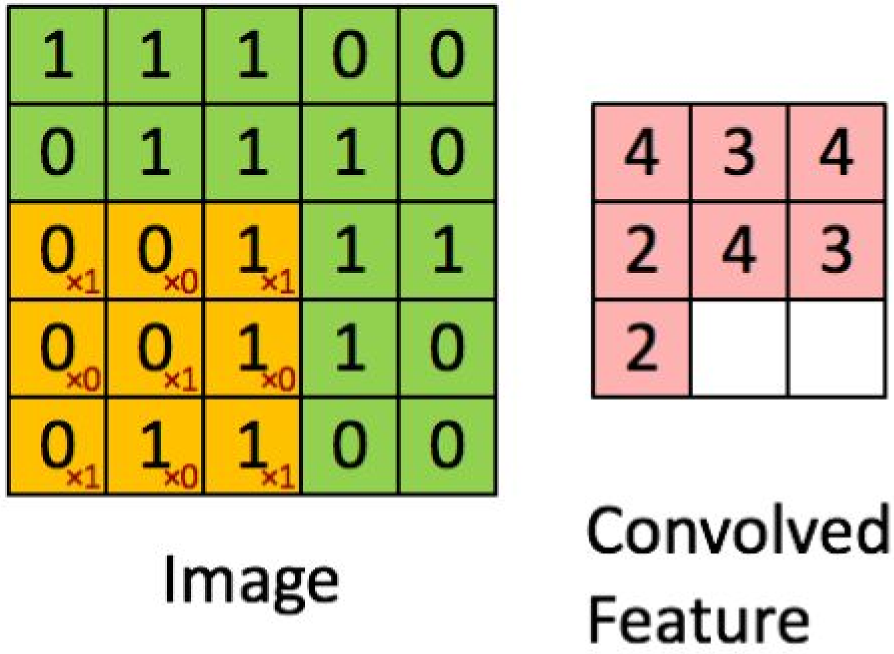
Layers

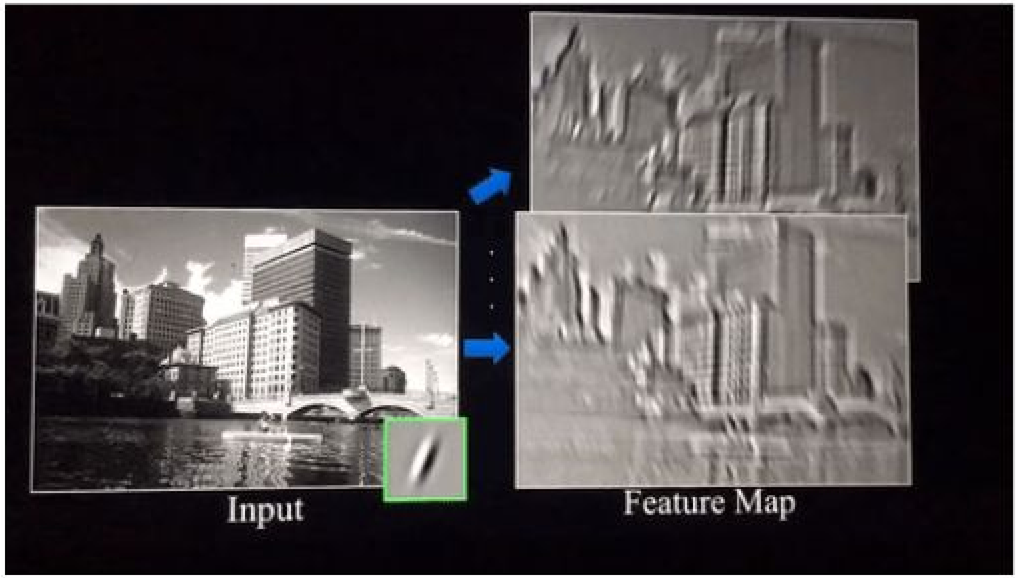
- **Convolution (3D)**



Layers

- **Convolution (3D)**

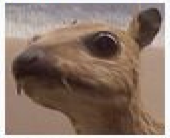
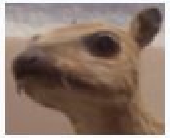
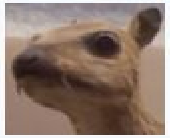
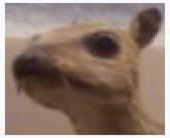




Source: S. Lazebnik

Layers

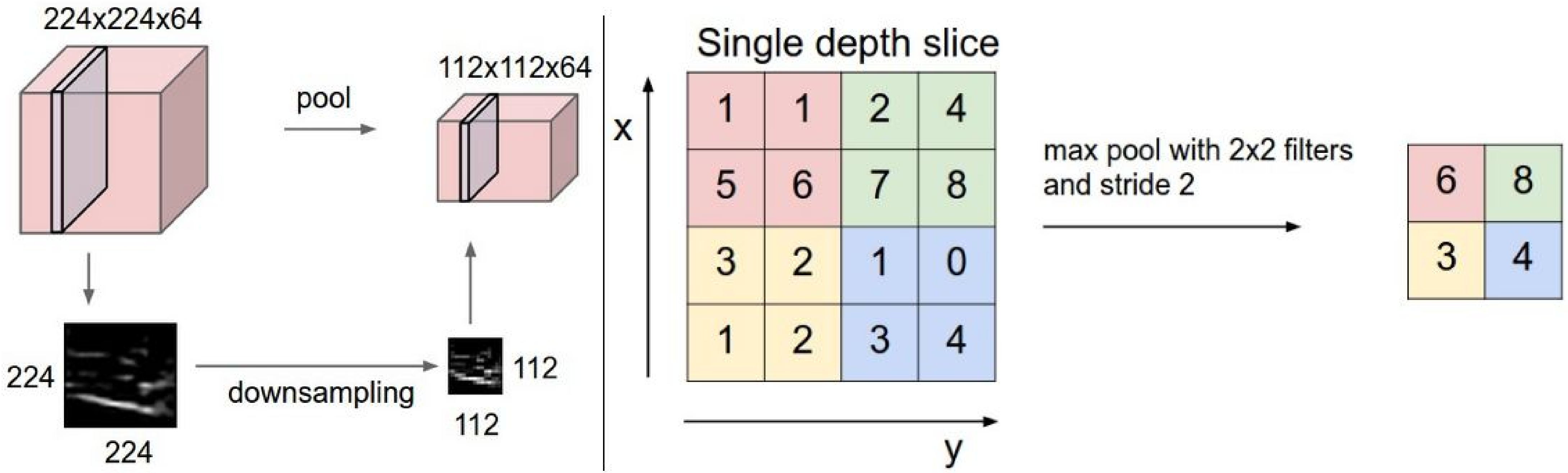
- **Filters**

Operation	Kernel	Image result
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 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Source: Wikipedia

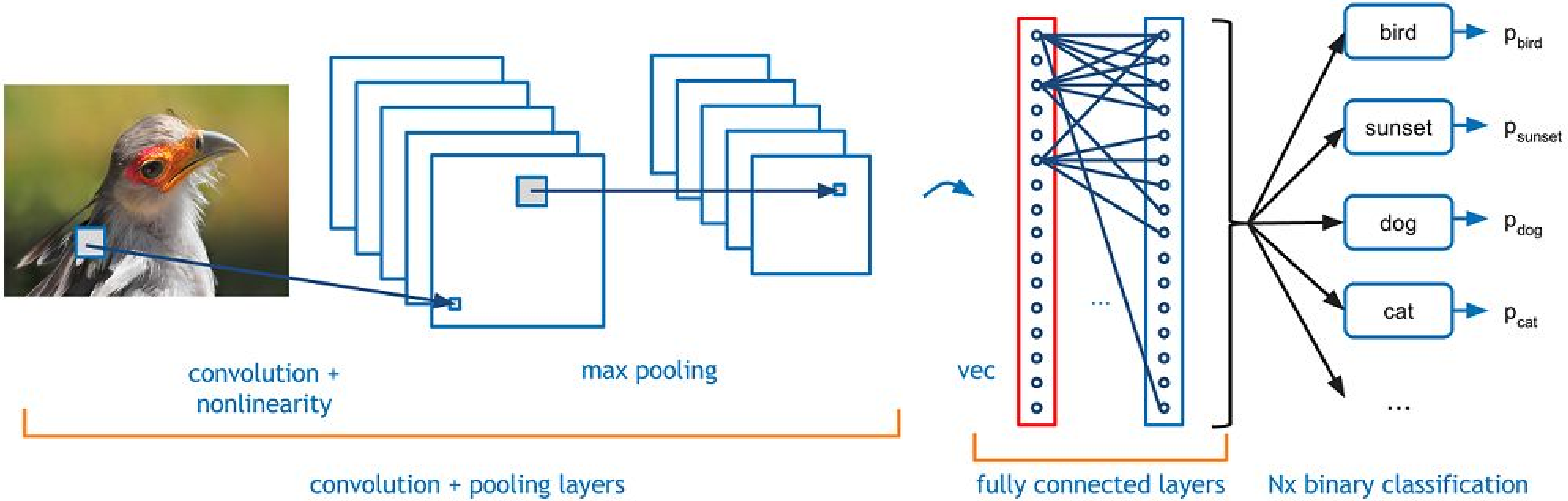
Layers

- **Downsampling | Max pooling**



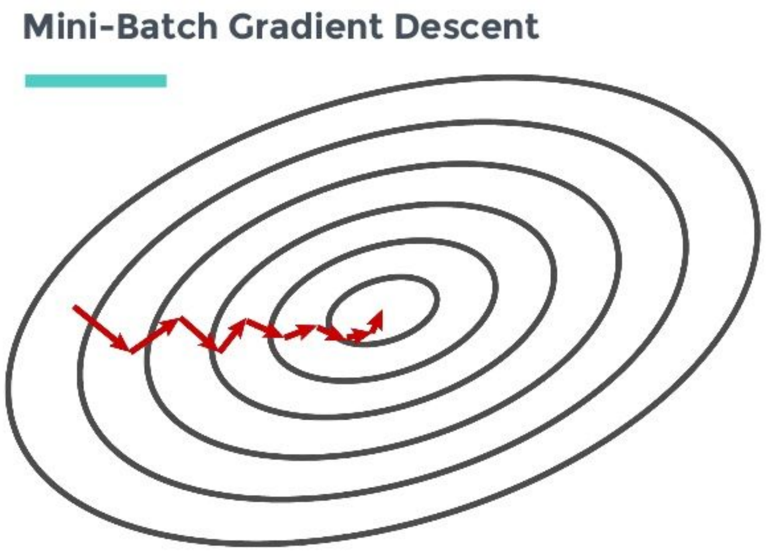
Source: Stanford cs231n

Layers


bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution +
nonlinearity
max pooling
vec
convolution + pooling layers
fully connected layers
Nx binary classification

Train

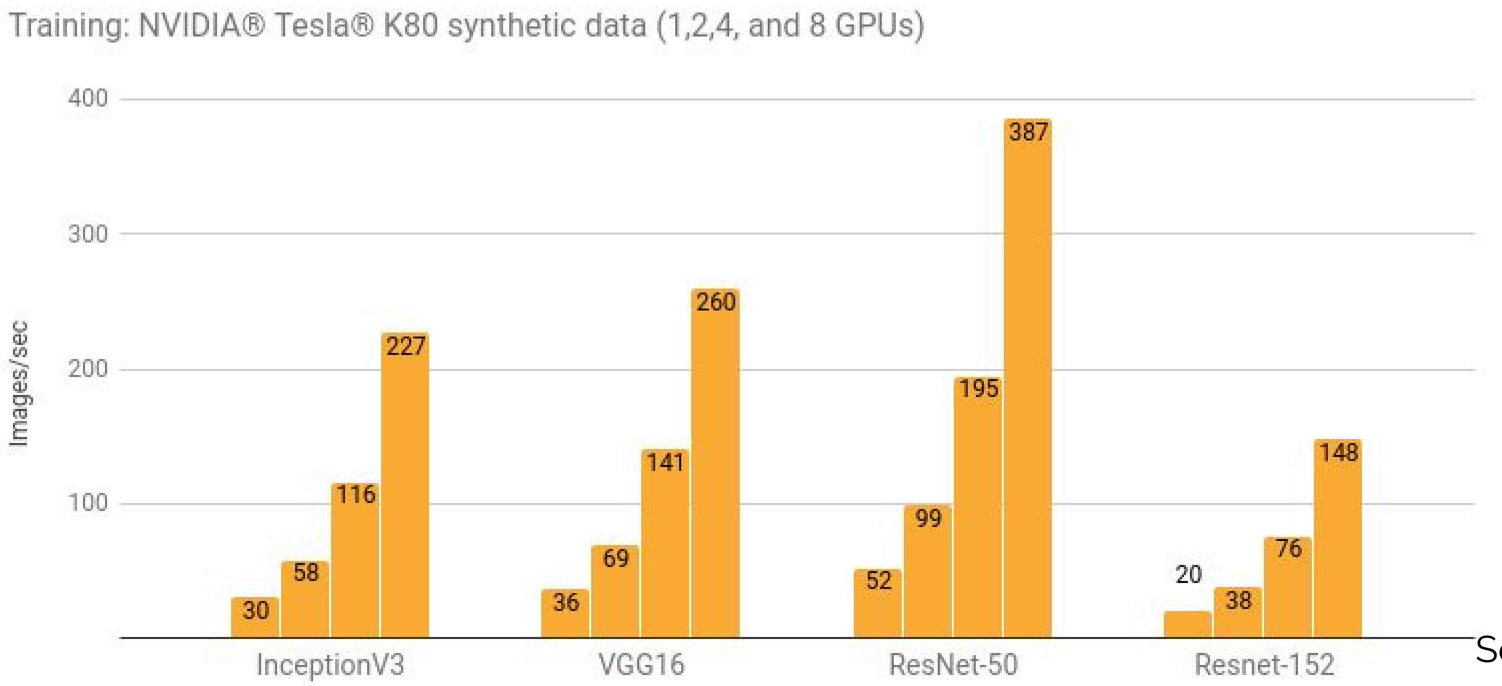
- **Batching**
  - The network can not be trained with all data at once
  - The data is divided in batches
  - Update the loss and the accuracy for each step/batch



Train

Also allows distributed training!

- **Batching**
  - The network can not be trained with all data at once
  - The data is divided in batches
  - Update the loss and the accuracy for each step/batch



Source: TensorFlow

Train

- **Hyperparameters**
    - Batch size
    - Epochs
    - Learning rate
    - Loss function
    - Regularization
    - ...

Train

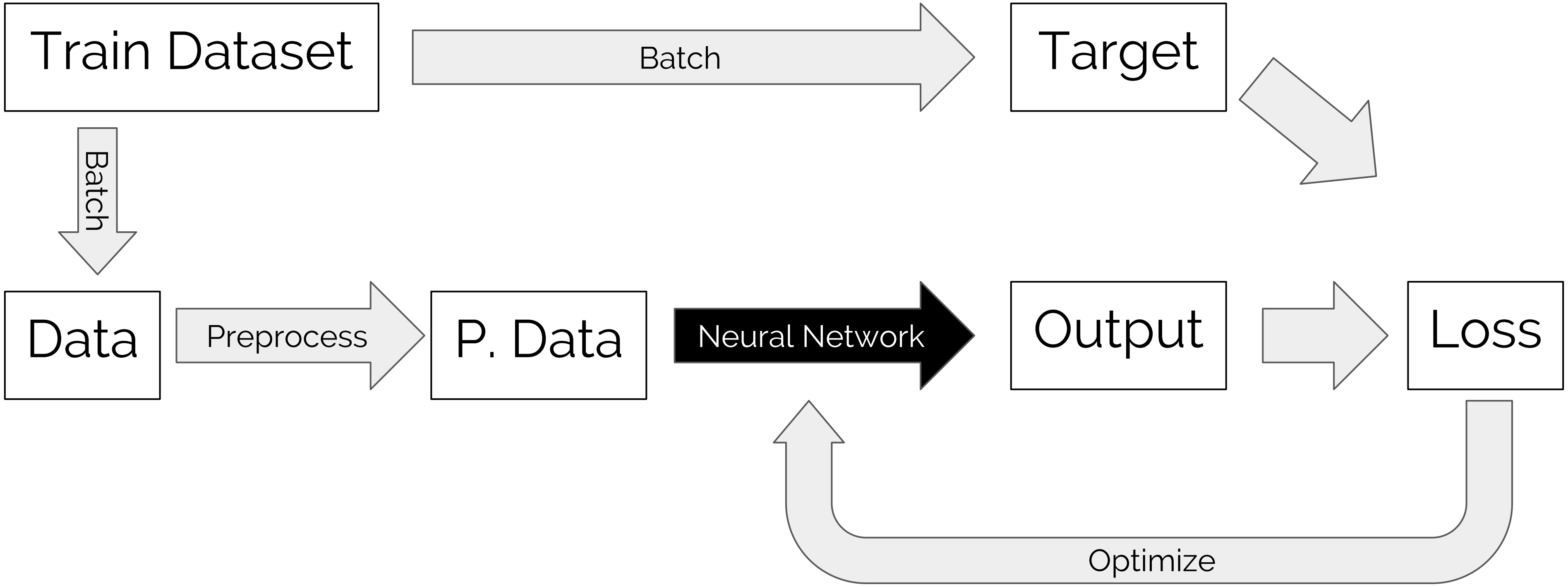
- **Data**
  - Training set
  - Validation set
  - Test set

Train

- **Data**
  - Training set
  - Validation set → **For hyperparameter tuning**
  - Test set → **Test model performance**

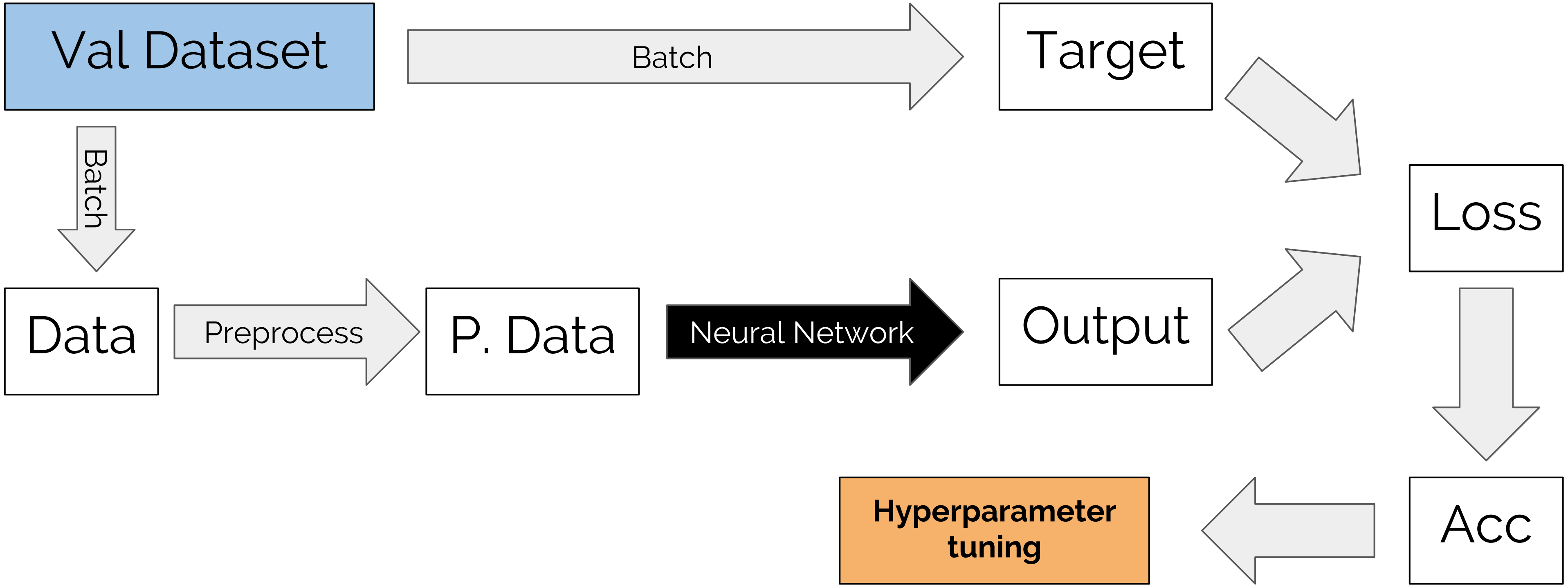
Train

- **Pipeline**

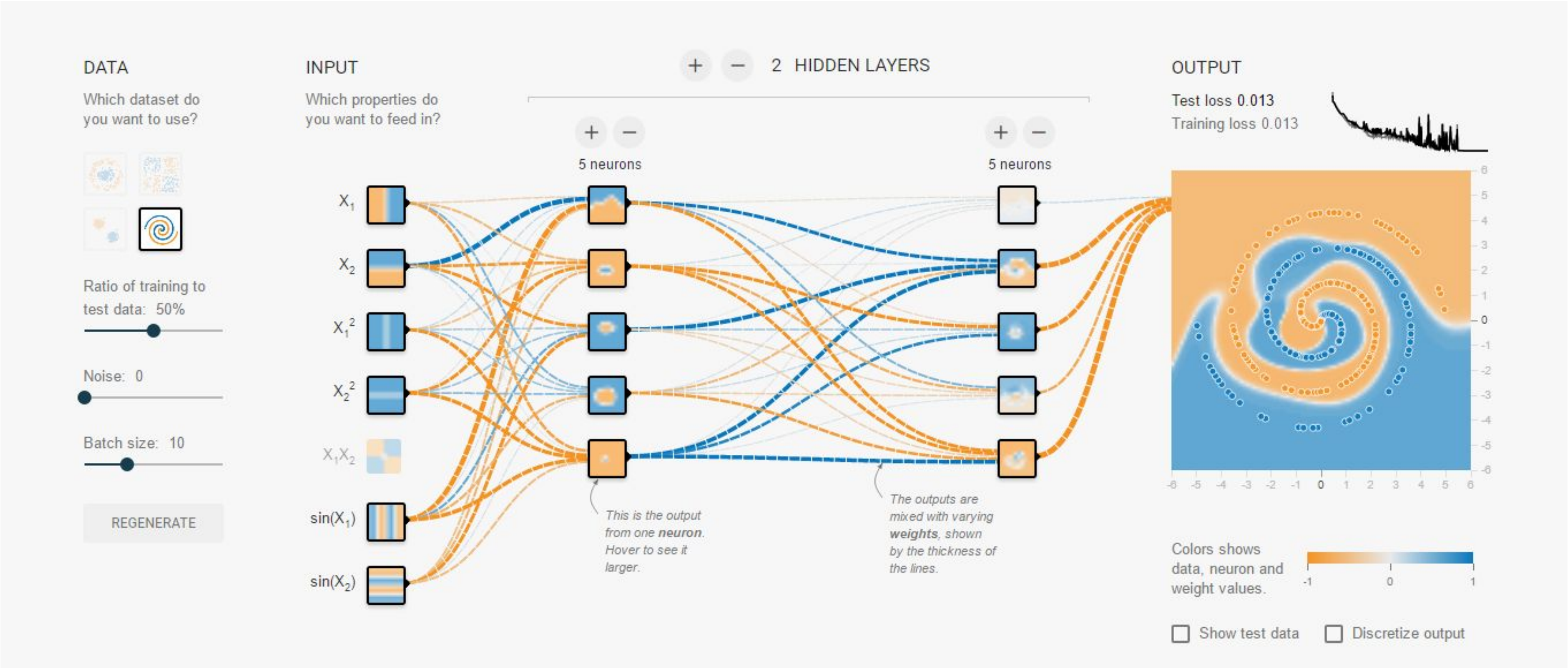


Train

- **Pipeline**
  - After epoch

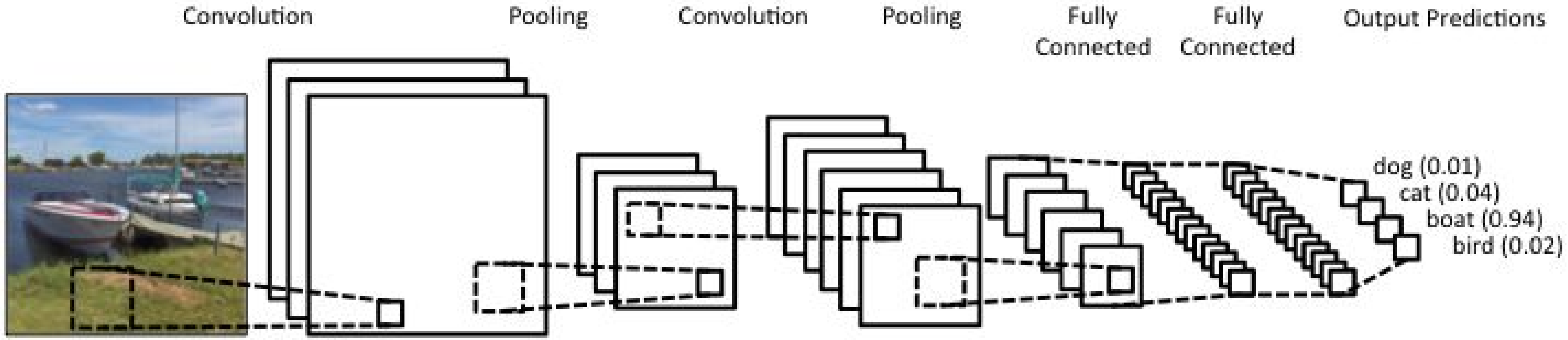


Let's play



http://playground.tensorflow.org

Network example



Source: clarifai

Network example

```
model.add(Conv2D(20, (5, 5), input_shape=(28, 28, 3),
      activation="relu", padding="same"))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(50, (5, 5), activation="relu", padding="same"))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Flatten())

model.add(Dense(500, activation="relu"))

model.add(Dense(10, activation="softmax"))
```

Inference example

Using docker, run:

```
docker pull jorditorresbcn/dl

docker run -it -p 8888:8888 jorditorresbcn/dl

wget https://raw.githubusercontent.com/jorditorresBCN/dlaimet/master/keras/Inference.ipynb

jupyter notebook --allow-root --ip=0.0.0.0
```

Inference example

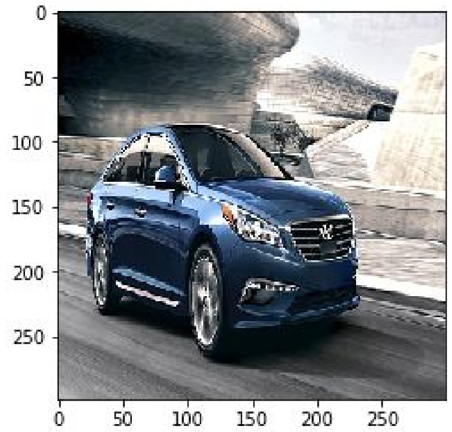
In [8]:

```
url_image = "https://media.brstatic.com/2017/03/17170632/2016-hyundai-sonata-mst.jpg"
```

In [9]:

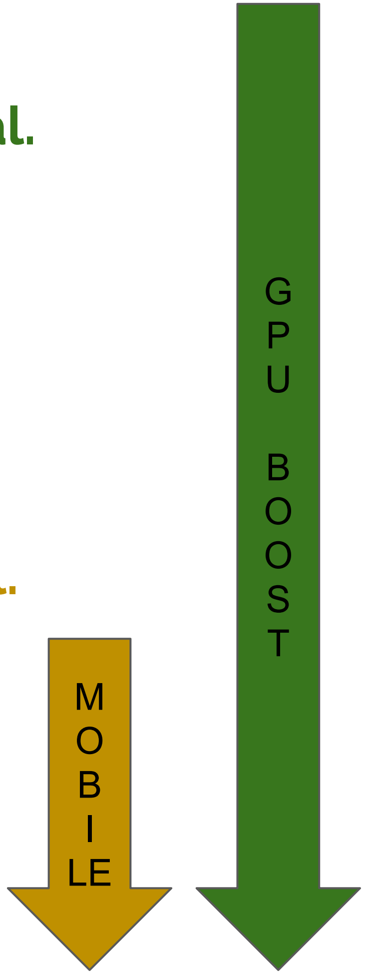
```
img_path = '/tmp/image'
urllib.request.urlretrieve(url_image, img_path)
img = image.load_img(img_path, target_size=(299, 299))
plt.imshow(img)
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n04285008', 'sports_car', 0.69676977), ('n04037443', 'racer', 0.08645054), ('n02974003', 'car_wheel', 0.046
464231)]
```



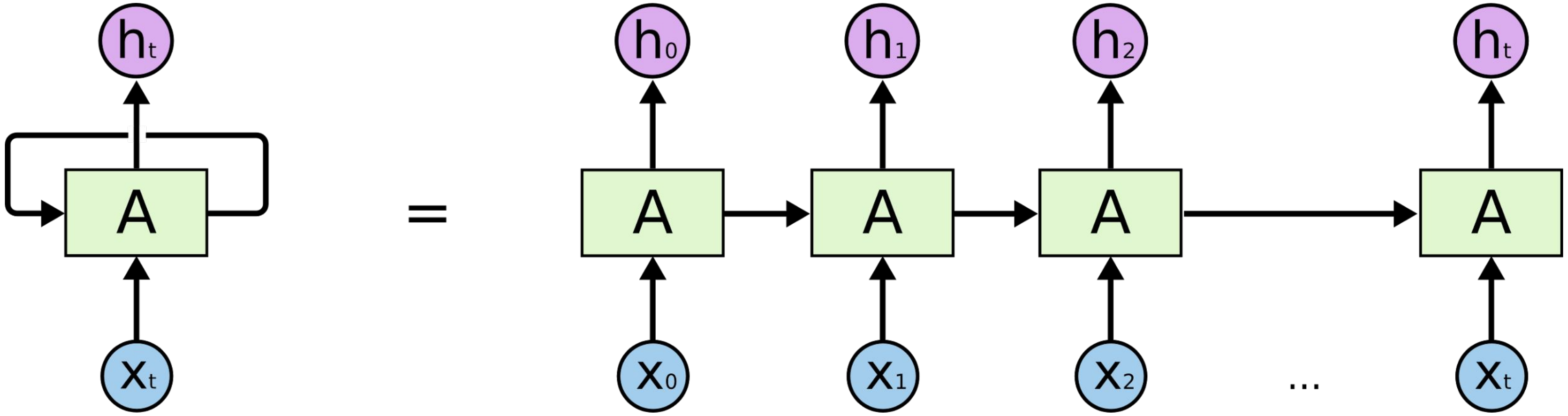
Networks

- LeNet (1990) - Yann Lecun
- AlexNet (2012) - Alex Krizhevsky et al.
- GoogLeNet (2014) - Christian Szegedy et al.
  - Inception V2 (2015)
  - Inception V3 (2015)
- VGG (2014) - Karen Simonyan et al.
- ResNet (2015) - Kaiming He et al.
  - Inception (v4)-ResNet (2016)
- MobileNet (2017) - Andrew G. Howard et al.



Other Networks

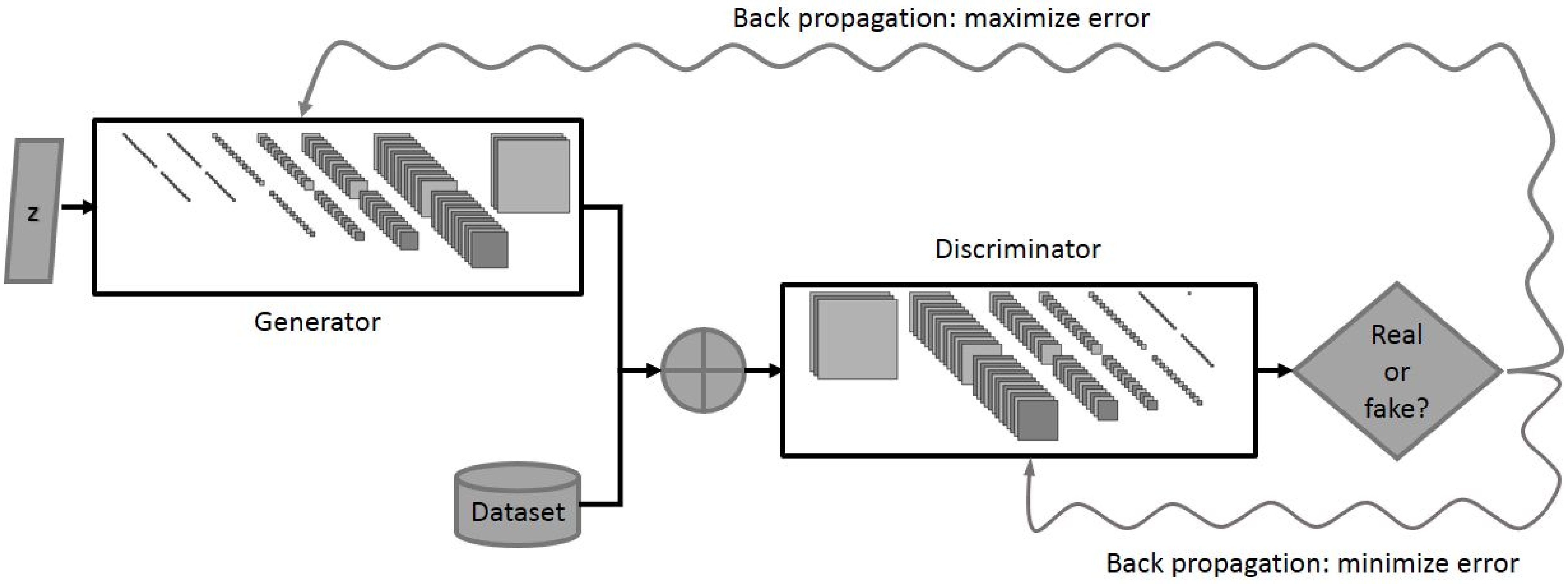
- **Recurrent Neural Networks (RNN)**



Source: Christopher Olah

Other Networks

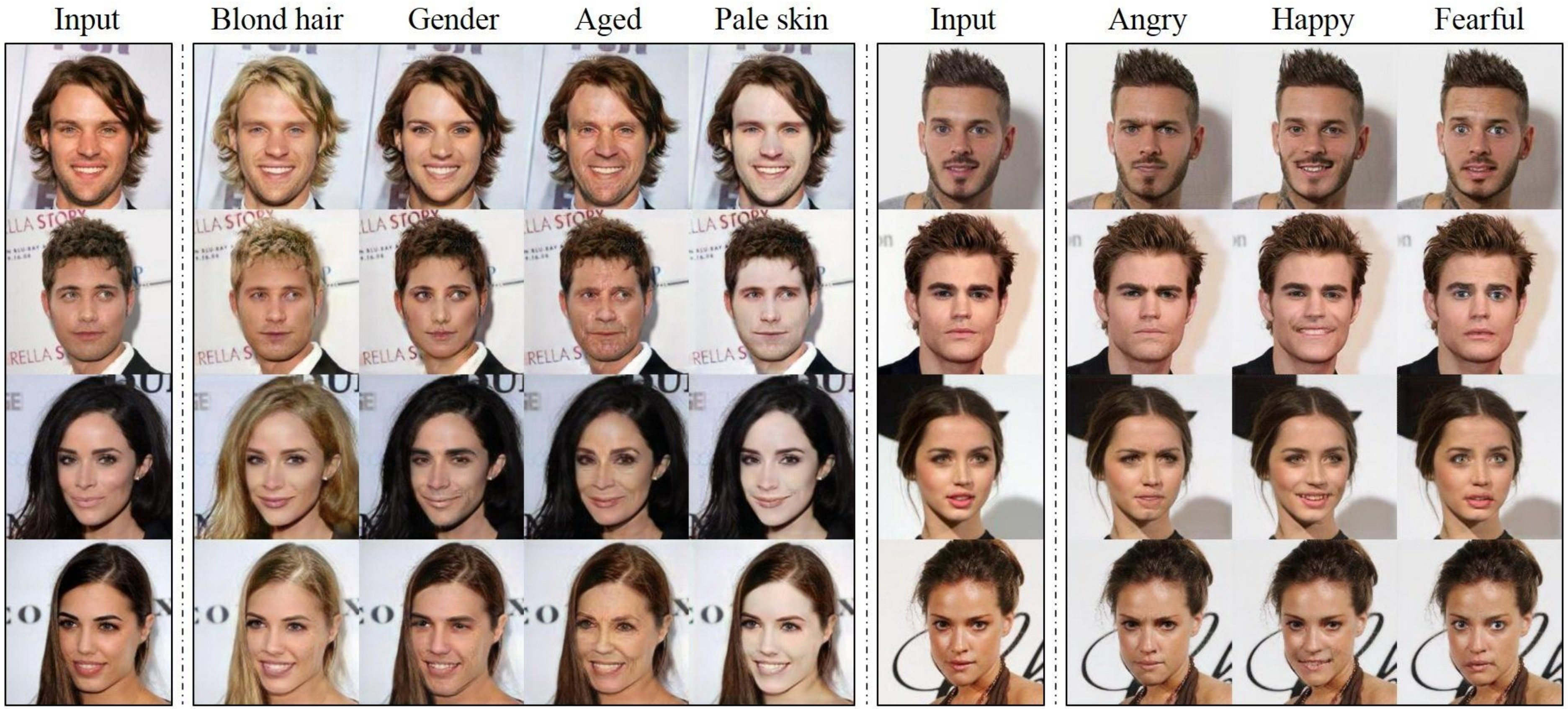
- **Generative Adversarial Networks (GAN)**



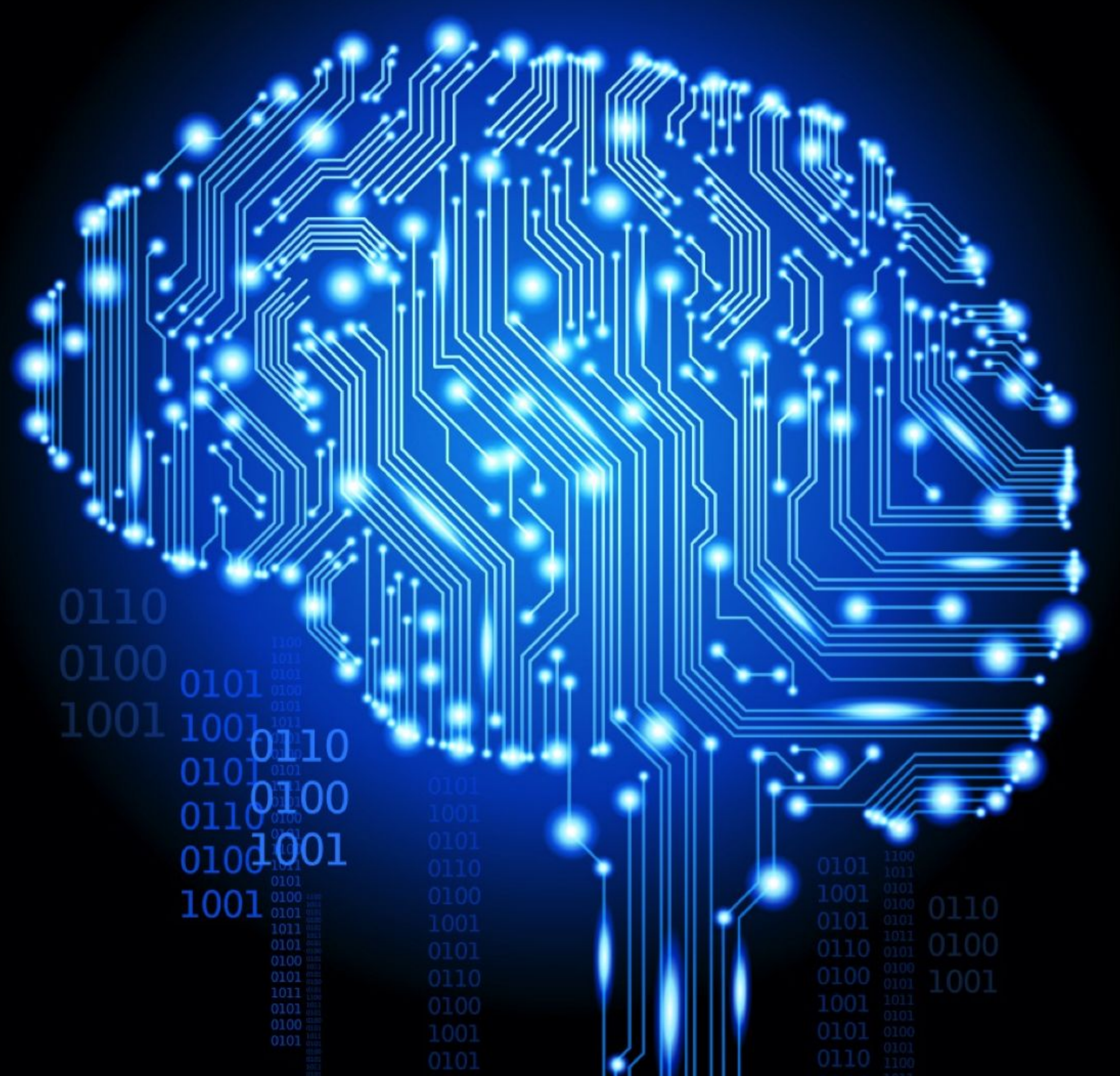
Source: Nvidia research

Other Networks

- **Generative Adversarial Networks (GAN)**



Source: Yunjey Choi



JORDI **TORRES** | FRANCESC **SASTRE**