Photogrammetry & Robotics Lab

Intro to Neural Networks
Part 1: Network Basics

Cyrill Stachniss

The slides have been created by Cyrill Stachniss.

5 Minute Preparation for Today

https://www.ipb.uni-bonn.de/5min/

Image Classification

input | classifier | output

→ "cat"

→ "5"

Semantic Segmentation

"a label for each pixel"

Neural Networks

- Machine learning technique
- Often used for classification, semantic segmentation, and related tasks
- First ideas discussed in the 1950/60ies
- Theory work on NNs in the 1990ies
- Increase in attention from 2000 on
- Deep learning took off around 2010
- CNNs for image tasks from 2012 on

Part 1
Neural Networks Basics

Neural Network

What is a **neuron**?

fundamental unit
(of the brain)

What is a **network**?

connected elements

neural networks are connected elementary (computing) units

Biological Neurons

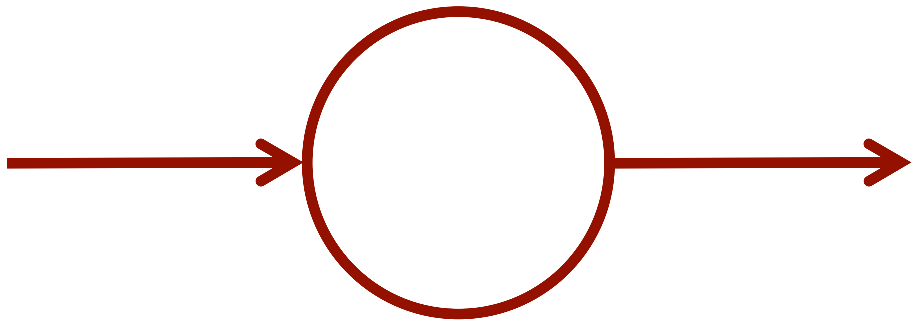
Biological neurons are the **fundamental units** of the brain that

- Receive sensory input from the external world or from other neurons
- Transform and relay signals
- Send signals to other neurons and also motor commands to the muscles

Artificial Neurons

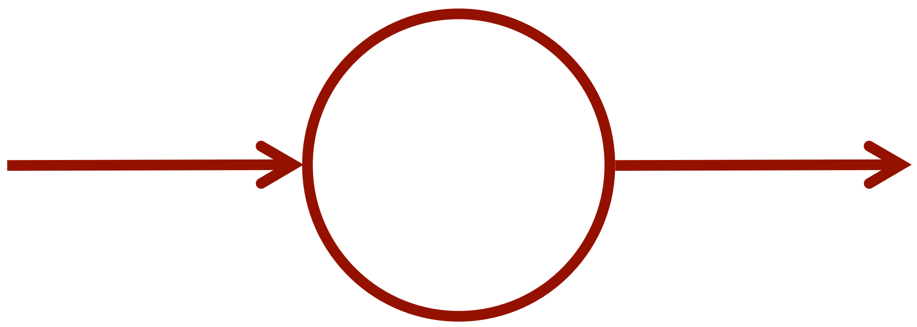
Artificial neurons are the fundamental units of artificial neural networks that

- Receive **inputs**
- **Transform** information
- Create an **output**



Neurons

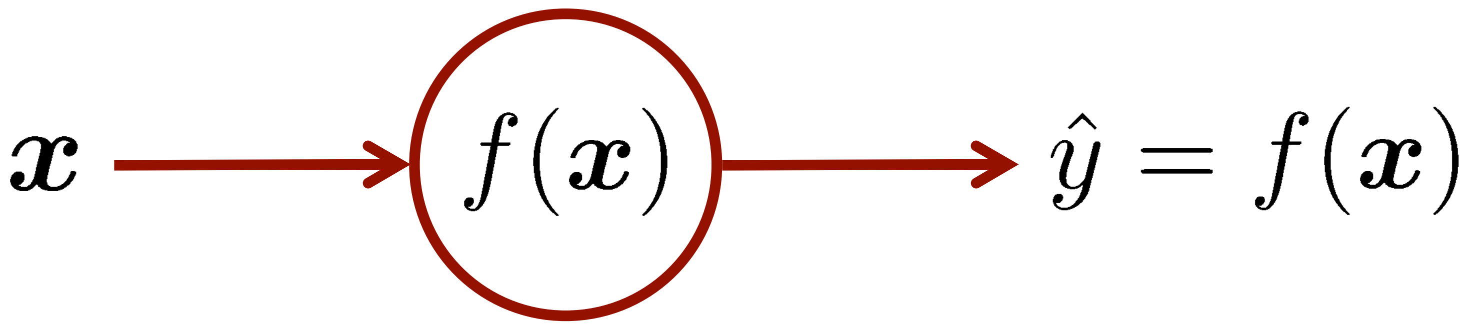
- Receive **inputs / activations** from sensors or other neurons
- **Combine / transform** information
- Create an **output / activation**



Neurons as Functions

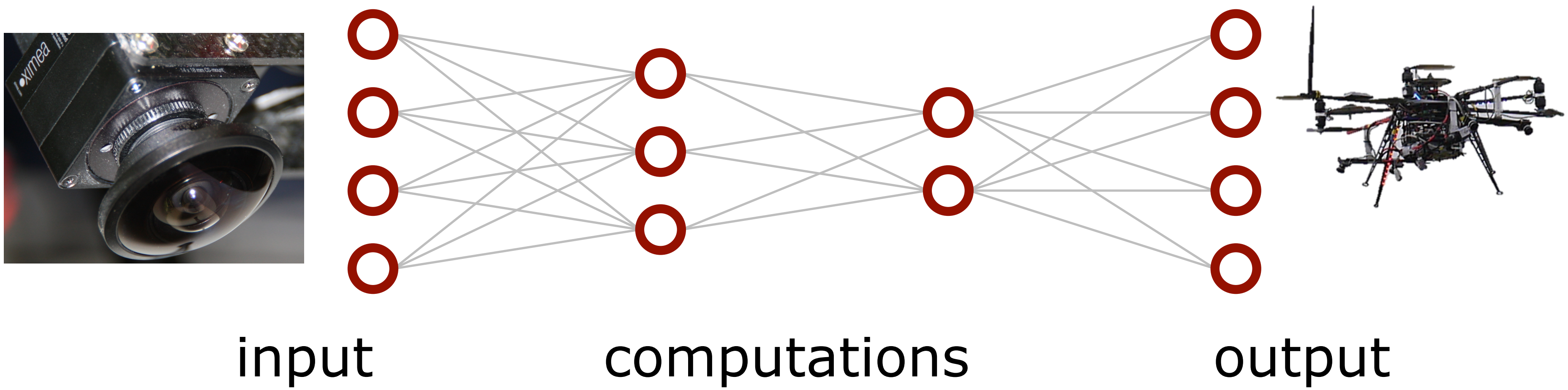
We can see a neuron as a function

- Input given by $\boldsymbol{x} \in \mathbb{R}^N$
- Transformation of the input data can be described by a function f
- Output $f(\boldsymbol{x}) = \hat{y} \in \mathbb{R}$



Neural Network

- NN is a network/graph of neurons
- Nodes are neurons
- Edges represent input-output connections of the data flow

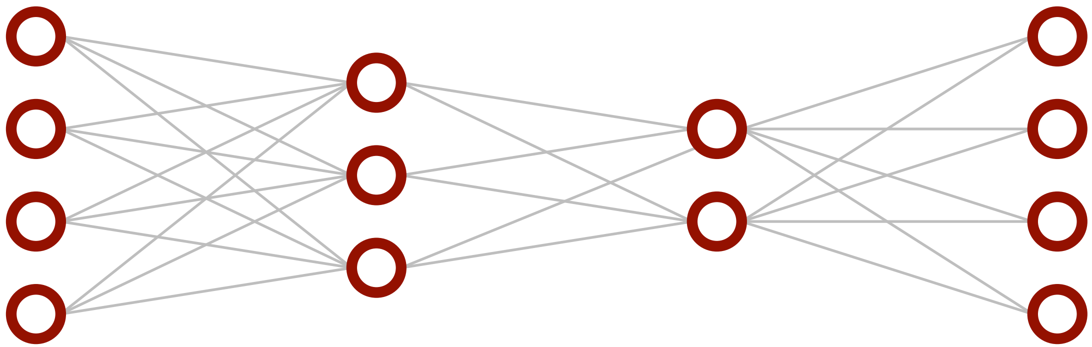


Neural Network as a Function

- The whole network is again a function
- Multi-layer perceptron or MLP is often seen as the “vanilla” neural network

input layer hidden layers output layer

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$



$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$\boldsymbol{x} \in \mathbb{R}^4$ $\hat{\boldsymbol{y}} = f_{NN}(\boldsymbol{x})$ $\hat{\boldsymbol{y}} \in \mathbb{R}^4$

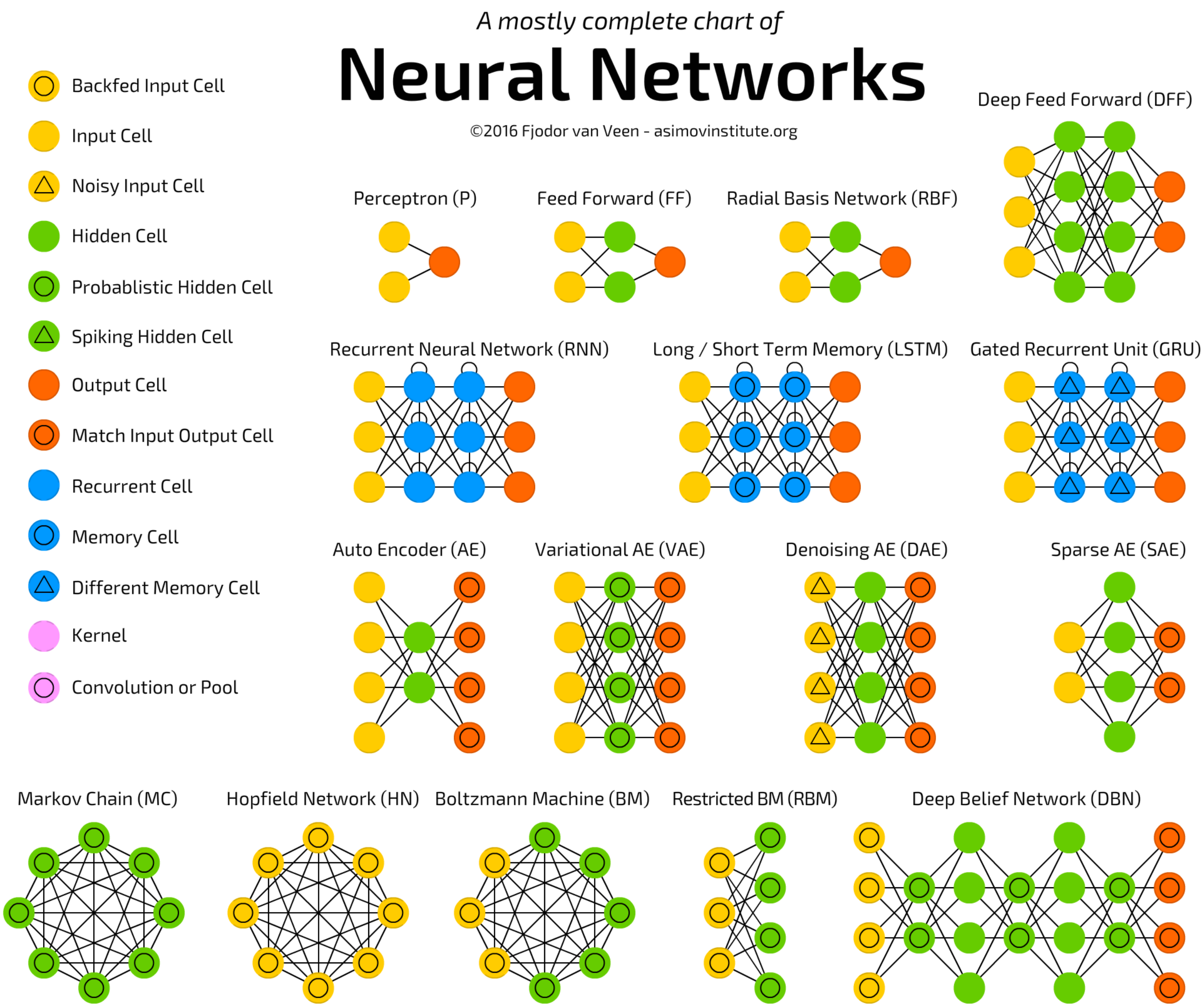
Neural Networks are Functions

- Neural networks are functions
- Consist of connected artificial neurons
- Input layer takes (sensor) data
- Output layer provides the function result (information or command)
- Hidden layers do some computations

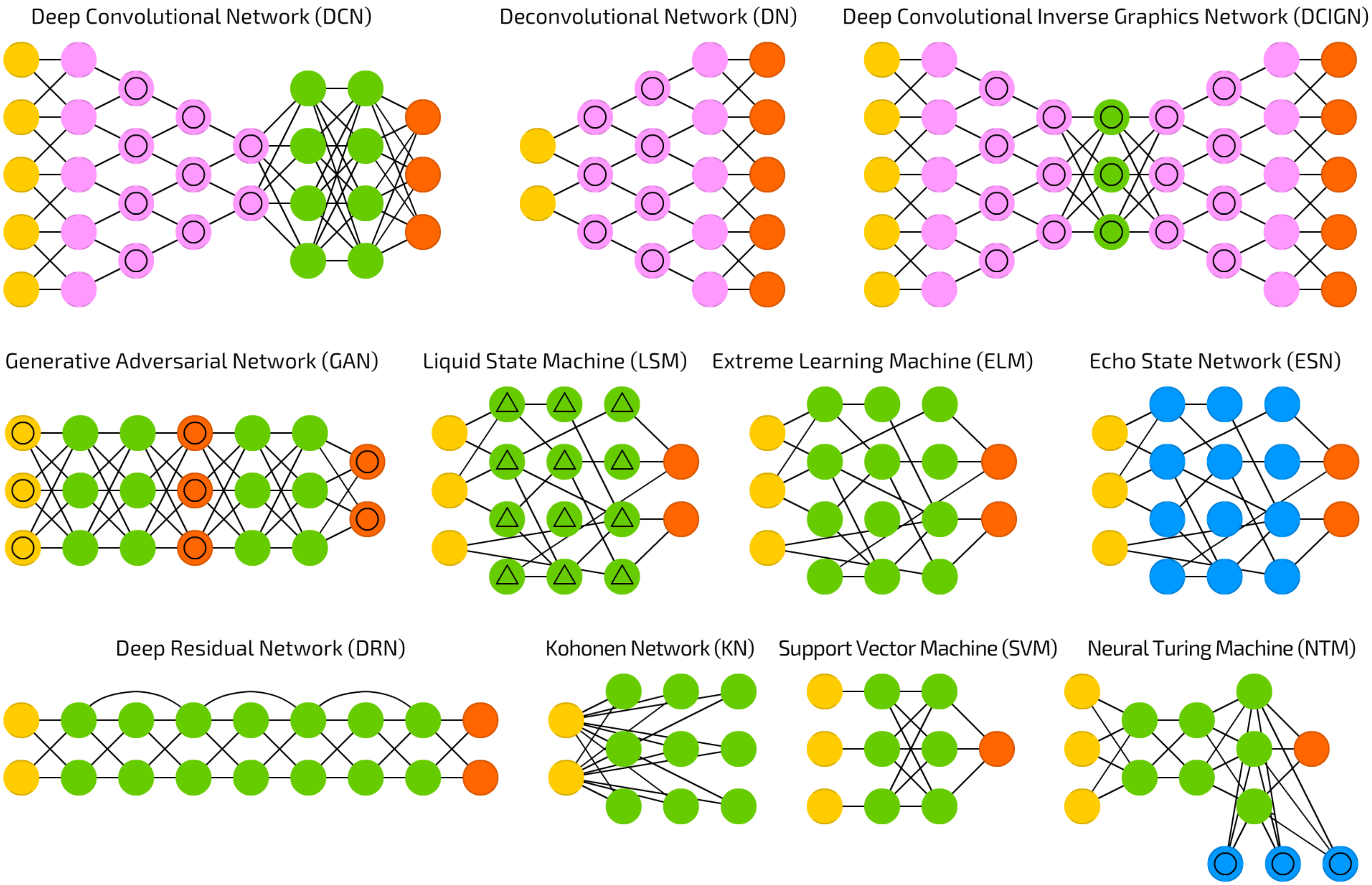
$\boldsymbol{x} \in \mathbb{R}^N$	$f_{NN}(\boldsymbol{x}) = \hat{\boldsymbol{y}}$	$\hat{\boldsymbol{y}} \in \mathbb{R}^M$
input layer	hidden layers	output layer

Different Types of NNs

- Perceptron
- MLP – Multilayer perceptron
- Autoencoder
- CNN – Convolutional NN
- RNN – Recurrent NN
- LSTM – Long/short term memory NN
- GANs – Generative adversarial network
- Graph NN
- Transformer
- ...

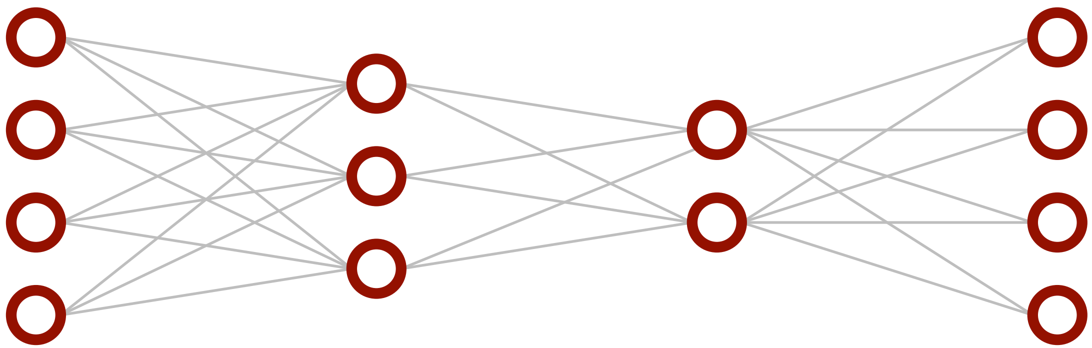


[Image courtesy: van Veen]



[Image courtesy: van Veen]

Multi-layer Perceptron (MLP)



Multi-layer Perceptron Seen as a Function

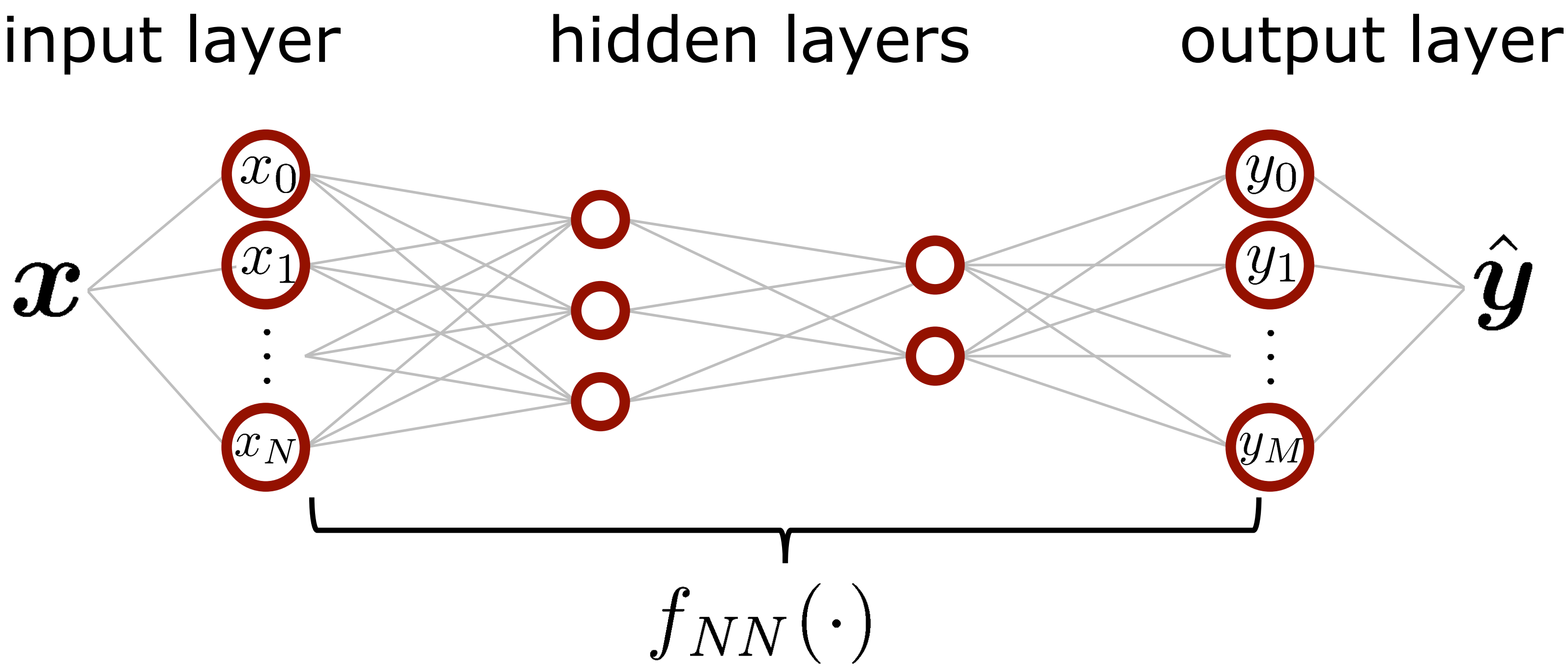
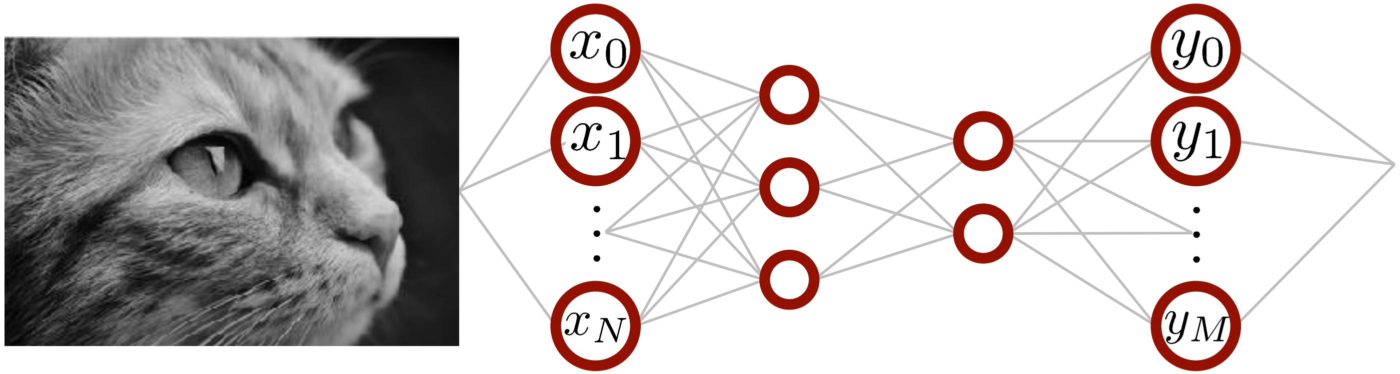


Image Classification Example



"cat"

$\boldsymbol{x}$ — input image

$f_{NN}(\cdot)$ — function that maps images to labels

$\hat{\boldsymbol{y}}$ — label

What is the Network's Input?

An image consists of individual pixels.

image

What is the Network's Input?

pixel intensities

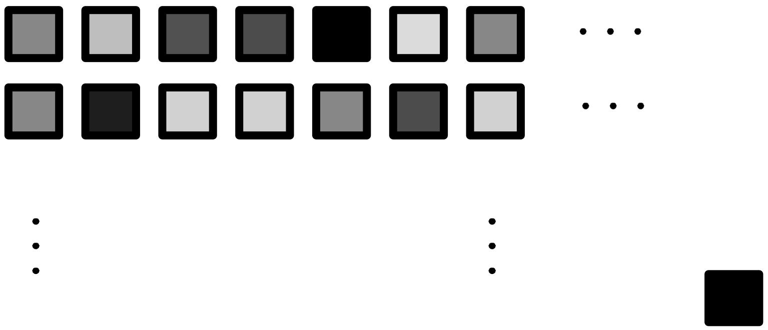
image

An image consists of individual pixels.

Each pixel stores an intensity value.

What is the Network’s Input?

pixel intensities

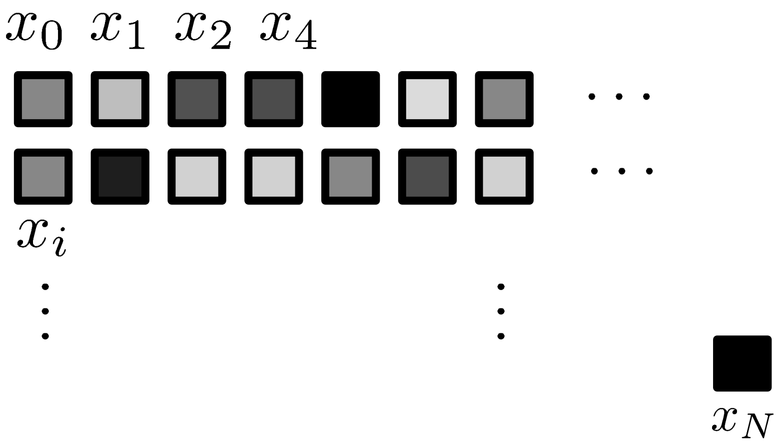


image

An image consists of individual pixels.

Each pixel stores an intensity value.

What is the Network’s Input?

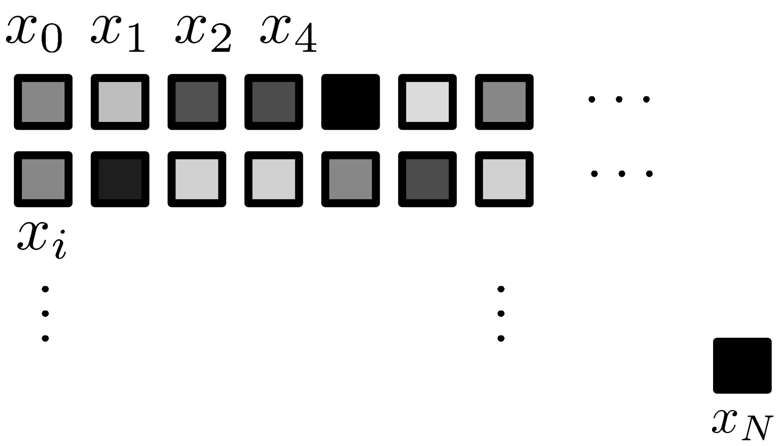


An image consists of individual pixels.

Each pixel stores an intensity value.

We have N+1 such intensity values.

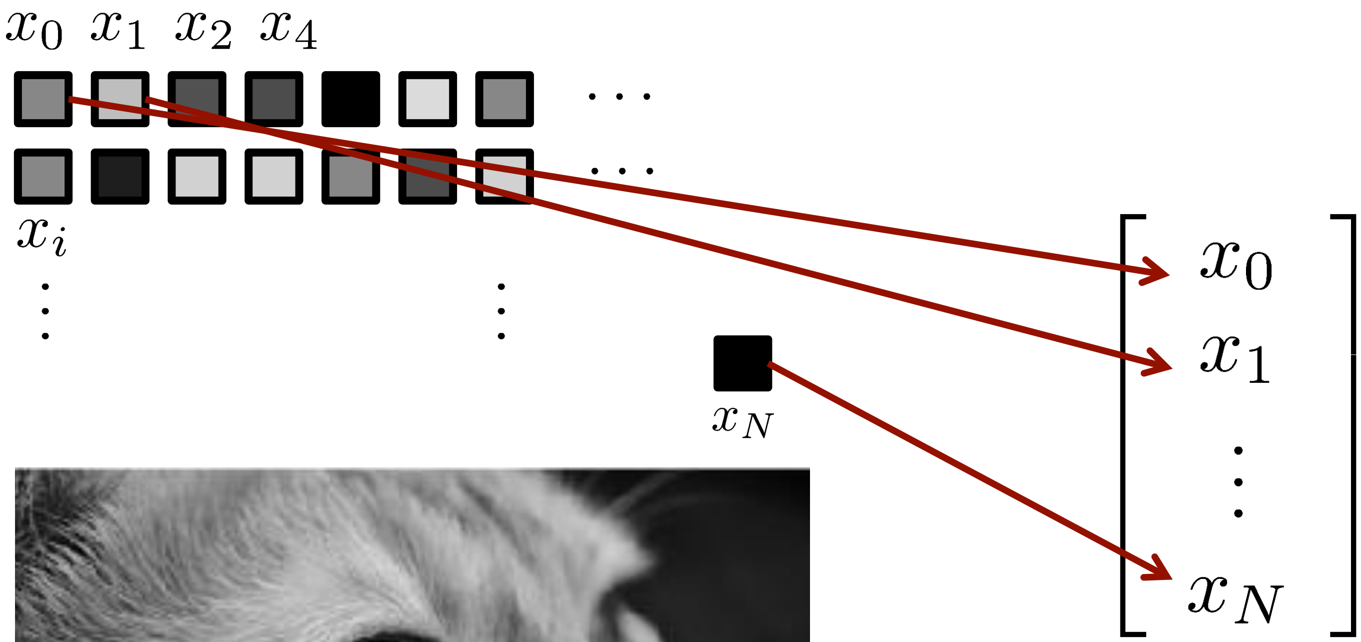
What is the Network's Input?



$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

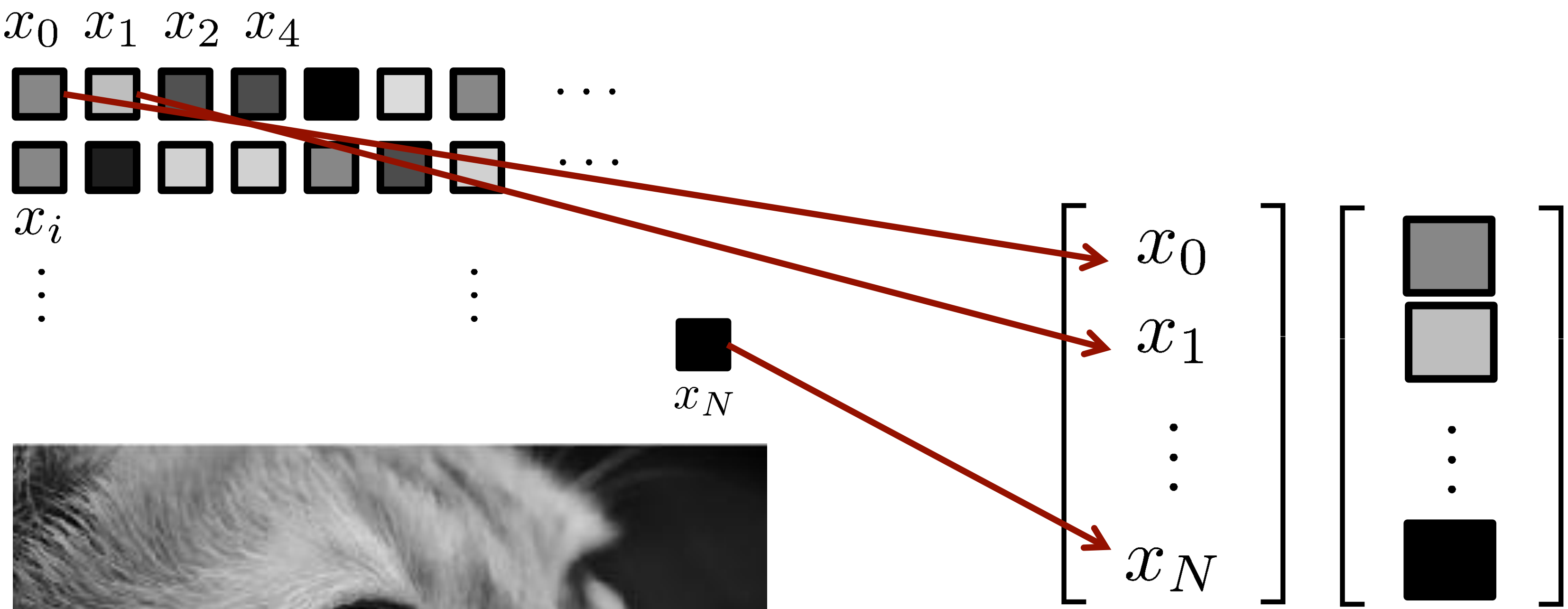
Arrange all the intensity values in a N+1 dim vector.

What is the Network’s Input?



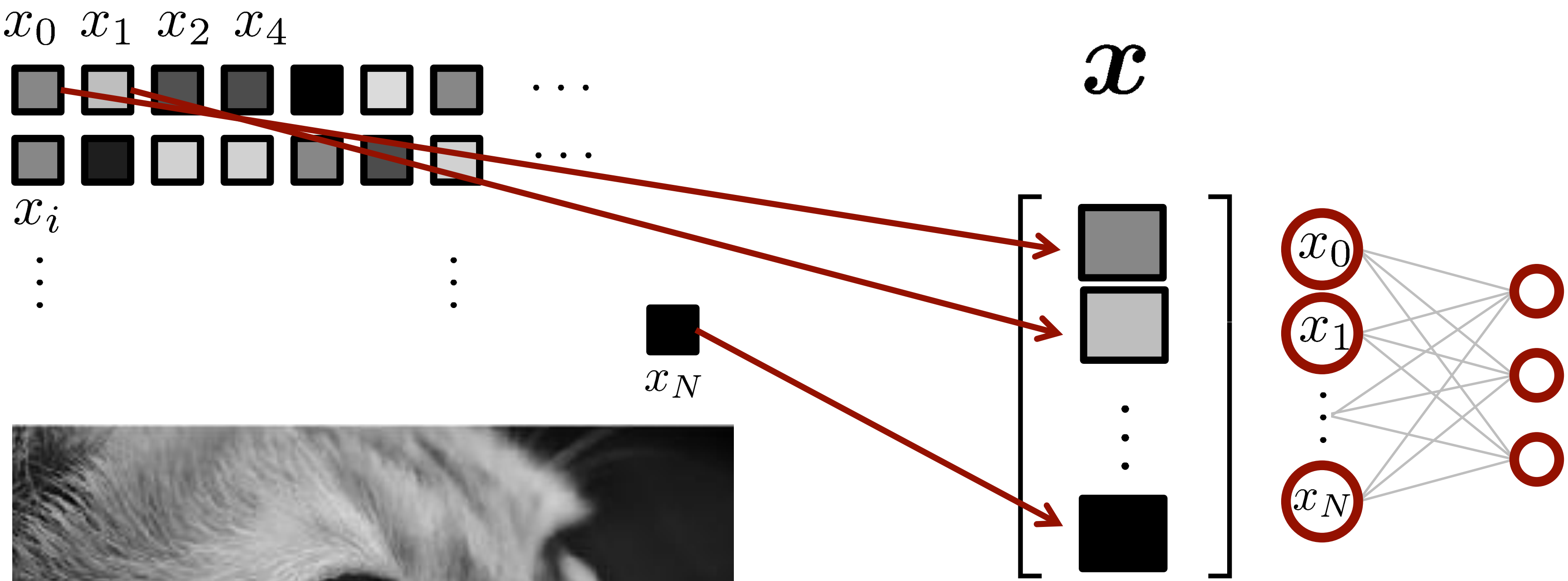
Arrange all the intensity values in a N+1 dim vector.

What is the Network's Input?



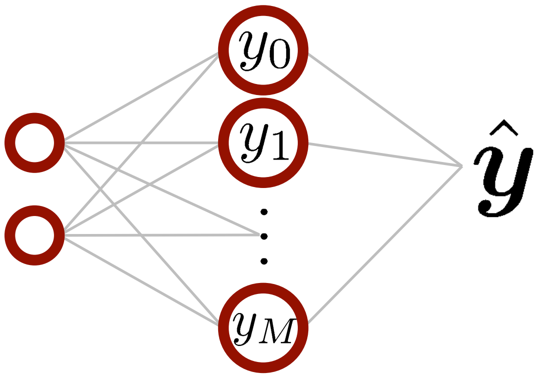
Arrange all the intensity values in a N+1 dim vector.

Input Layer of the Network



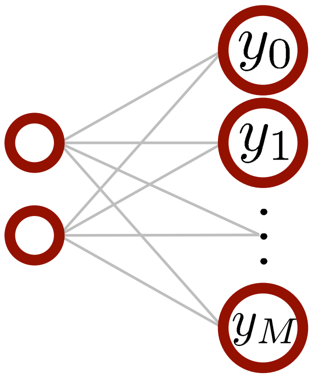
This vector is the input layer of our network!

What is the Network's Output?



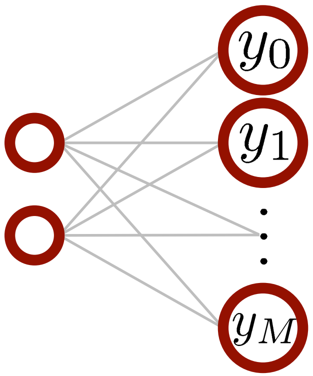
"cat"

What is the Network's Output?



Is it a…
cat or a
dog or a
human or a
...?

What is the Network's Output?

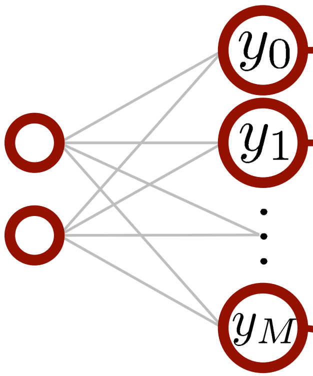


Is it a…
cat or a
dog or a
human or a
…?

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

indicator vector

What is the Network's Output?

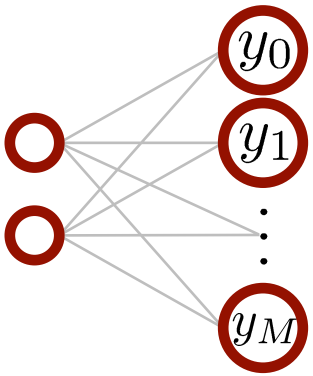


Is it a…
cat or a
dog or a
human or a
…?

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

indicator vector

What is the Network's Output?

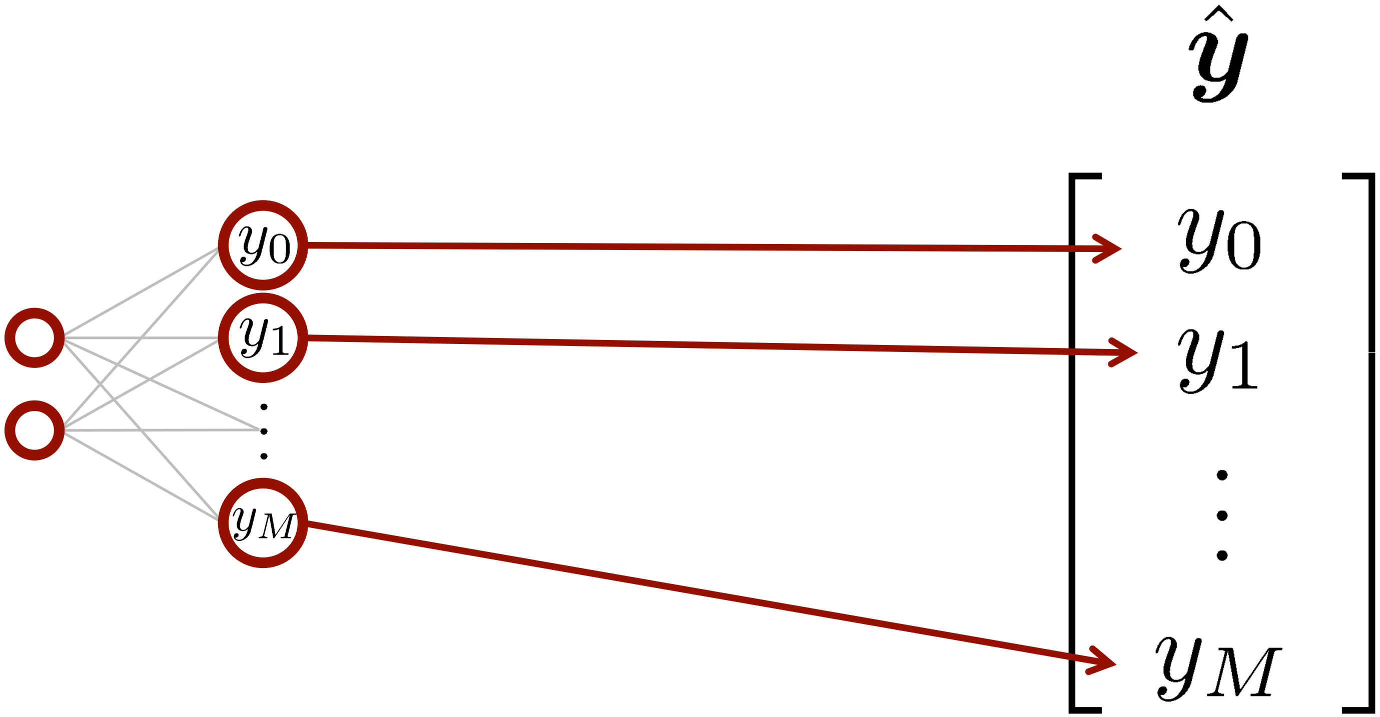


Is it a…
cat or a
dog or a
human or a
…?

$$\begin{bmatrix} 98\% \\ 1\% \\ 0.1\% \\ \vdots \end{bmatrix}$$

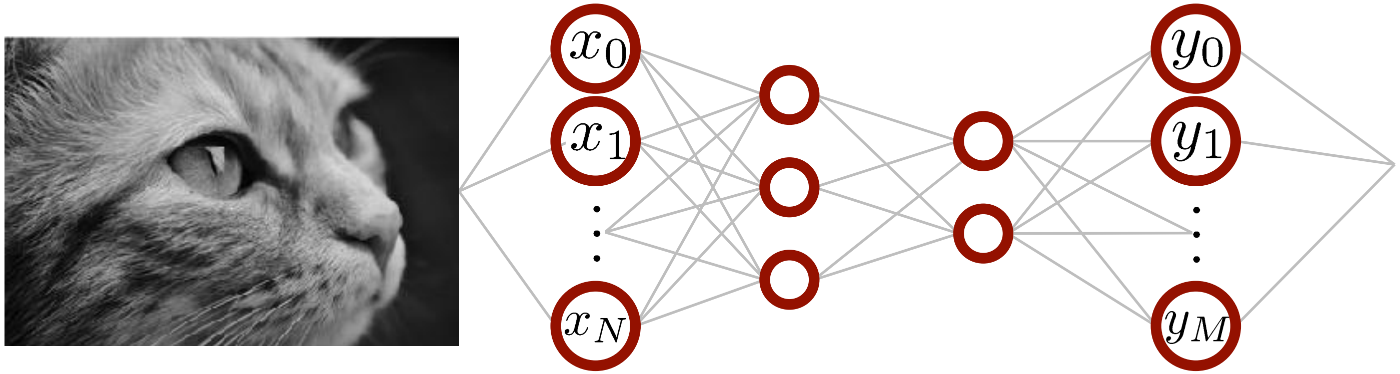
we are never certain…

Output of the Network



the output layer is vector indicating an activation/ likelihood for each label

Image Classification



"cat"

largest value

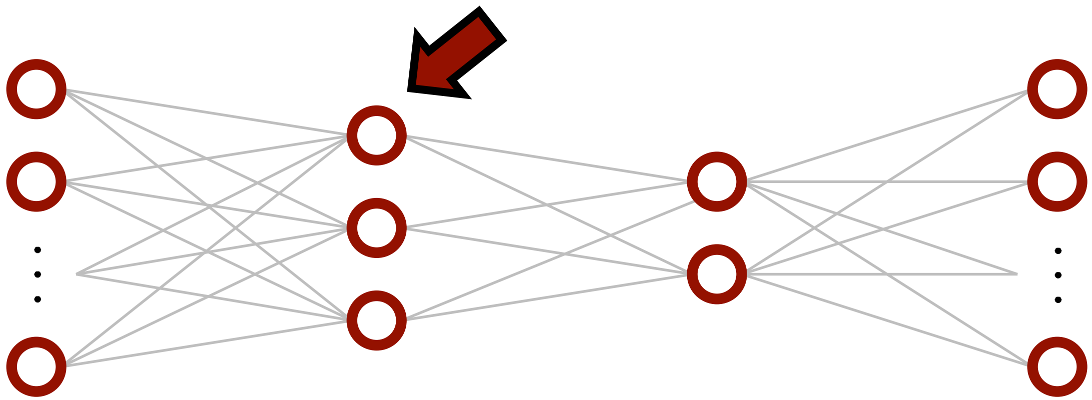
$\boldsymbol{x}$

$\hat{\boldsymbol{y}}$

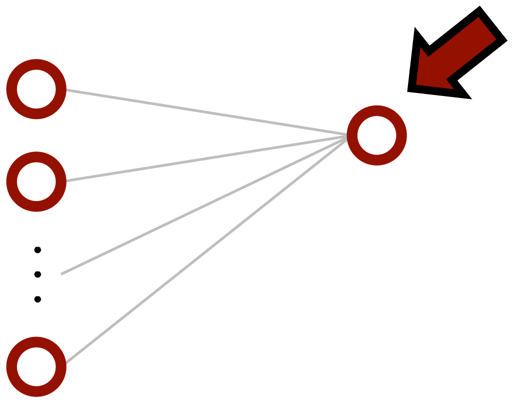
pixels intensities are the values of the input layer

output layer is a vector of likelihoods for the possible labels

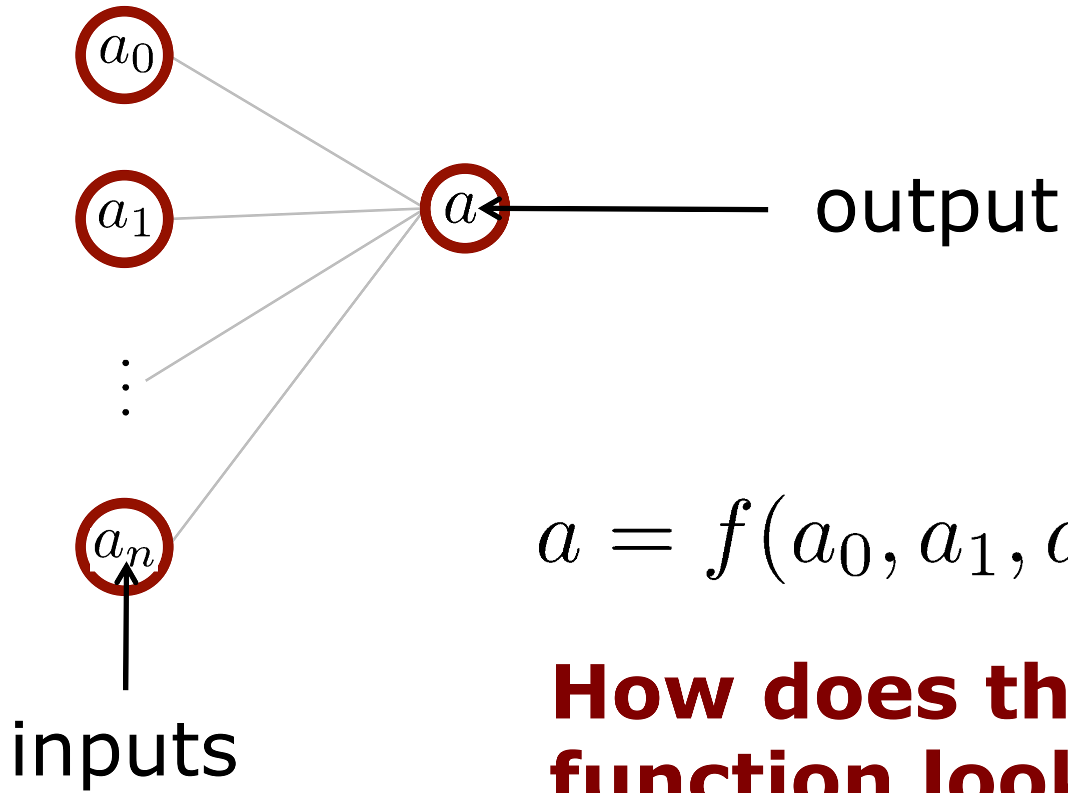
Multi-layer Perceptron
Let’s Look at a Single Neuron



Multi-layer Perceptron
Let’s Look at a Single Neuron



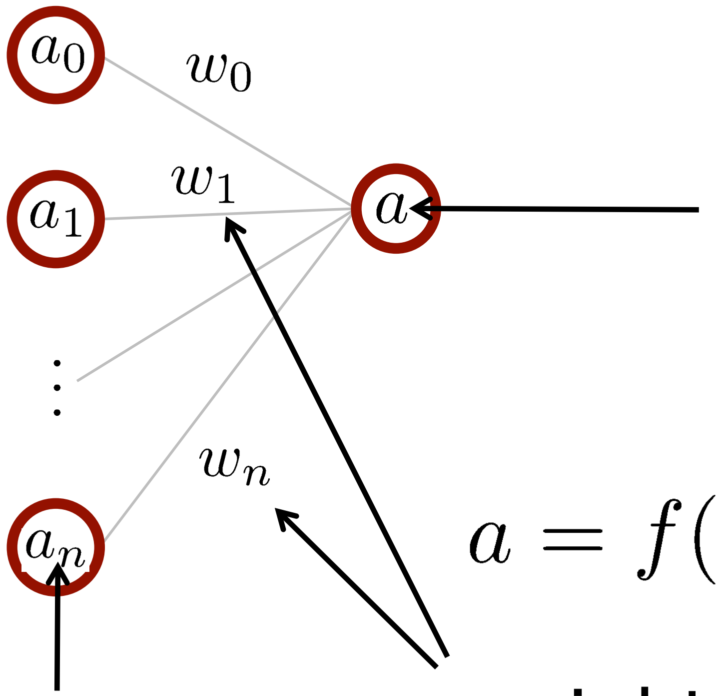
Perceptron (Single Neuron)



$$a = f(a_0, a_1, a_2, \ldots, a_n)$$

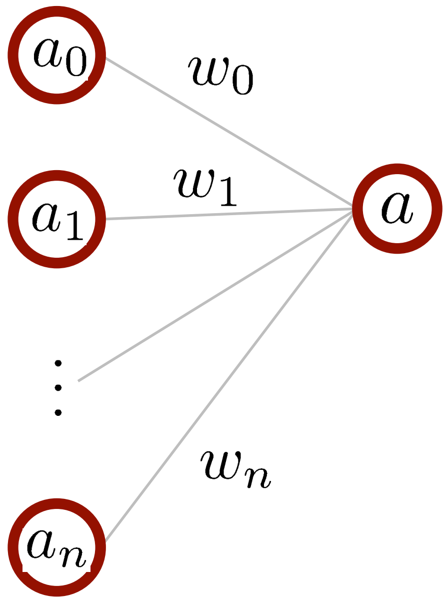
How does this function look like?

Perceptron (Single Neuron)



$$a = f(a_0, a_1, a_2, \ldots, a_n)$$

Function Behind a Neuron



(input) activations a_i

weights w_i

bias b

activation function $\sigma(\cdot)$

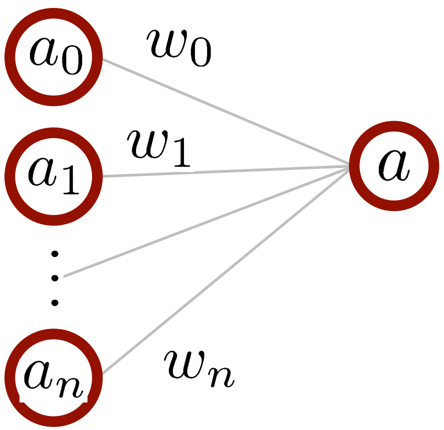
output activation a

Function Behind a Neuron

A neuron gets activated (a) through

- A weighted sum of input activations w_i, a_i
- A bias activation b
- An activation function $\sigma(\cdot)$

$$a = \sigma(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b)$$



Similarity to Convolutions?

- A neuron is similar to a convolution
- Remember linear shift-invariant kernels used as local operators

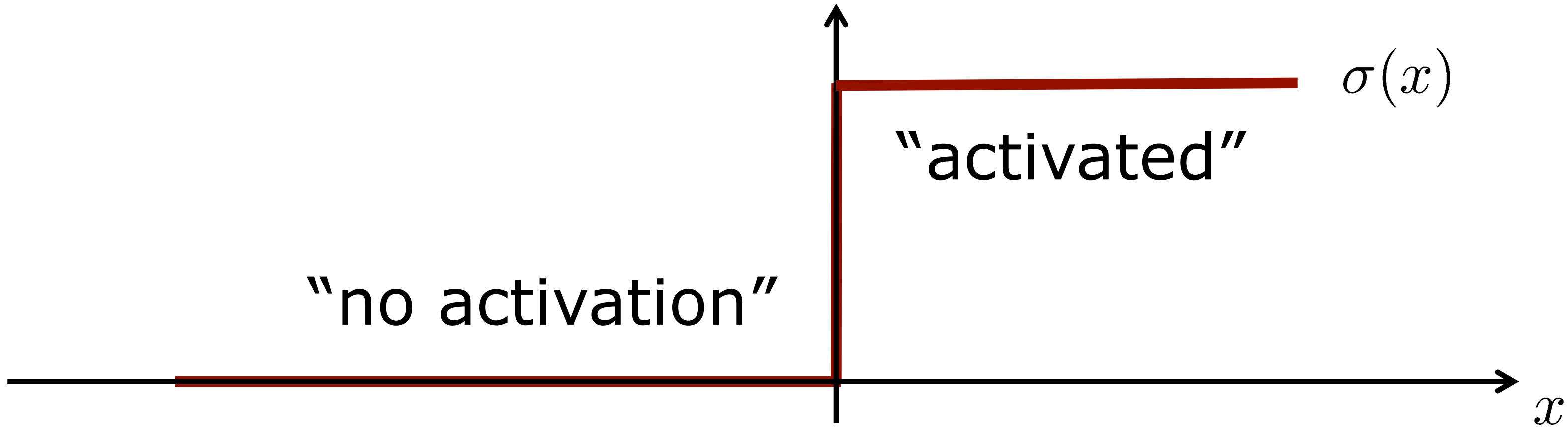
$$a = \sigma(\boxed{w_0 a_0 + w_1 a_1 + \ldots + w_n a_n} + b)$$

This part looks like the convolutions used for defining local operators

Additionally: activation function and bias

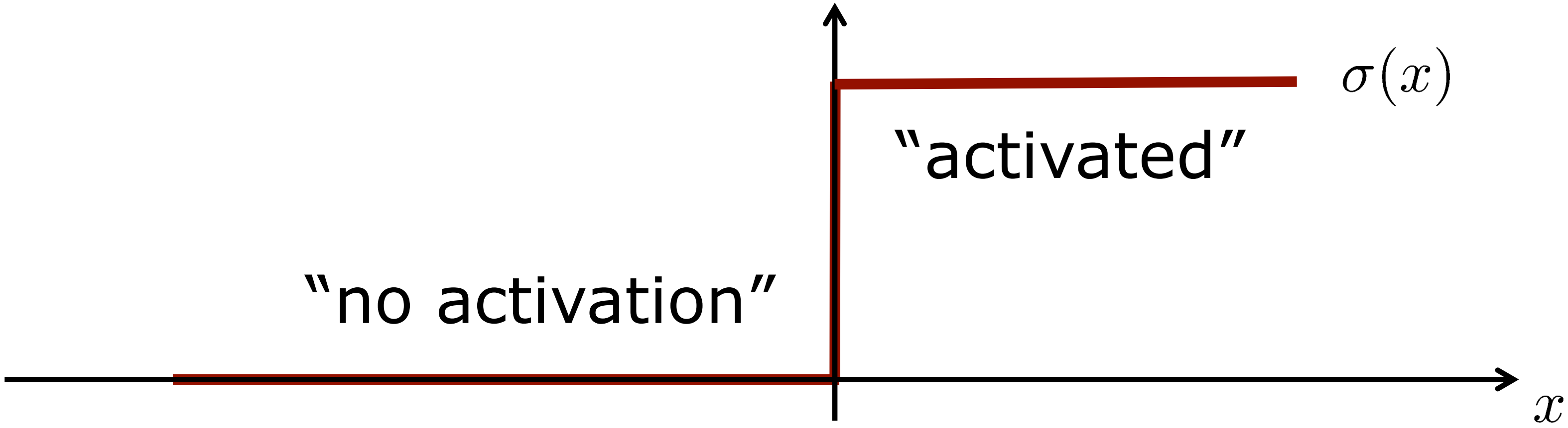
Activation Function

- Biological neurons are either active or not active
- We can see this as a step function:



- Bias tells us where the activation happens

Activation Function



- We can model this behavior through

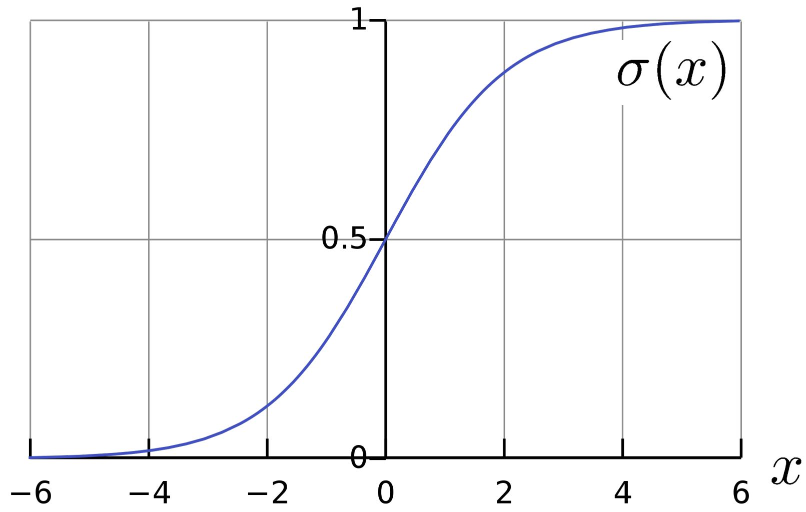
$$a = \begin{cases} 0 & \sum_i w_i a_i \leq -b \\ 1 & \text{otherwise} \end{cases}$$

- Non-smooth functions (eg, steps) have disadvantages later down the line...

Sigmoid Activation Function

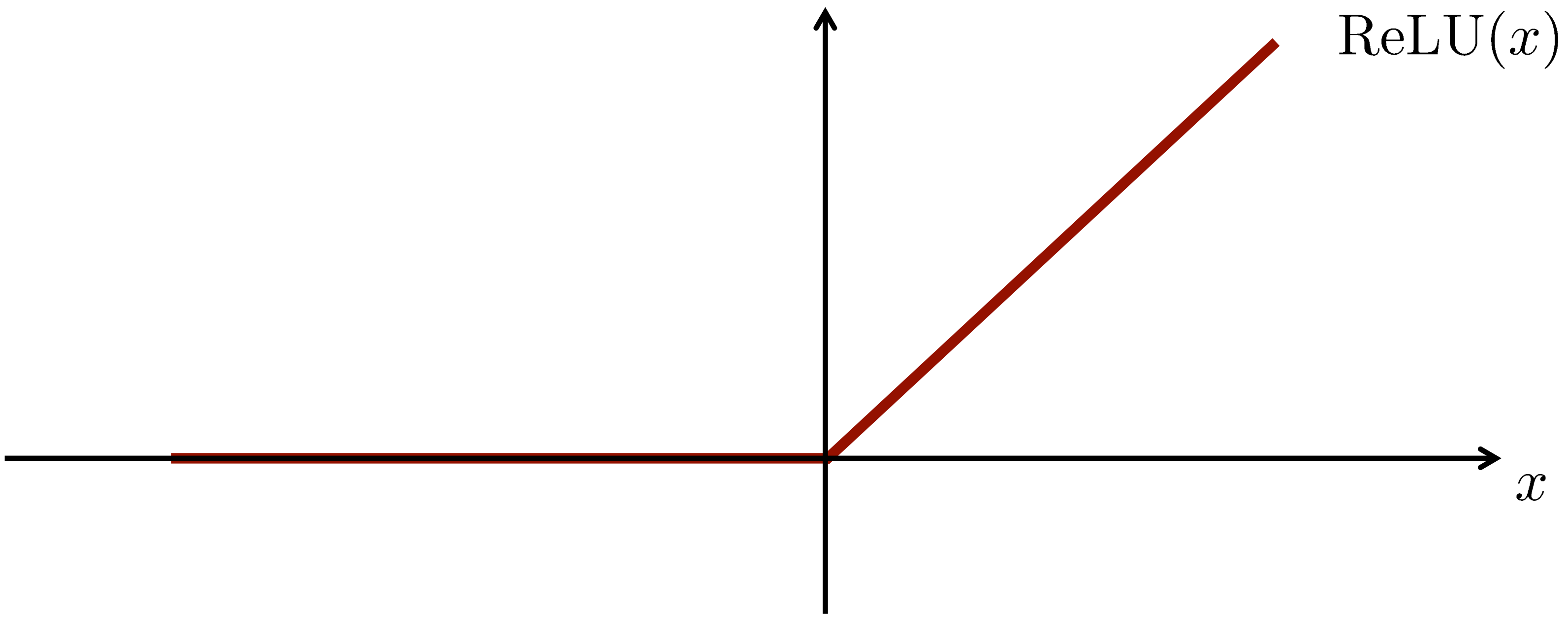
- Common activation function is a sigmoid (also called logistic function)
- Smooth function
- Squeezes values to [0,1]

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



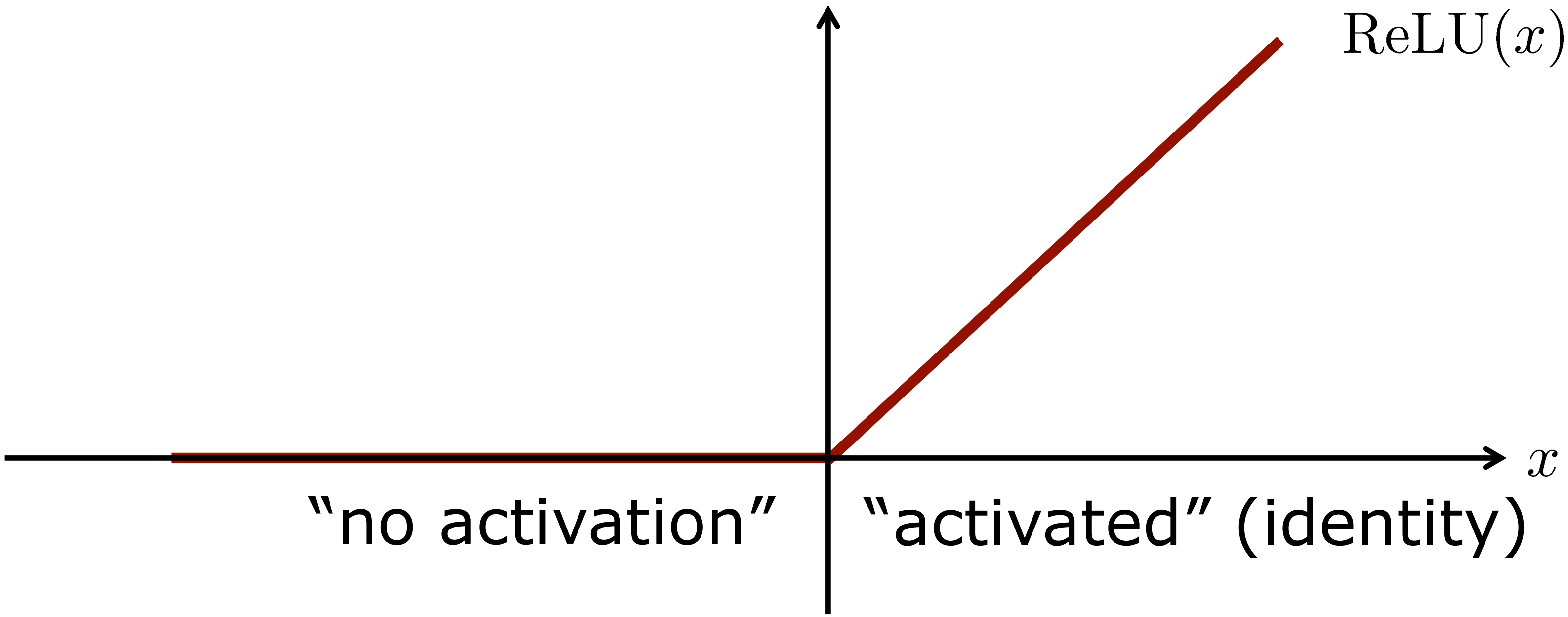
ReLU Activation Function

- Most commonly used one is the so-called "rectified linear unit" or ReLU
- $\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$
- Often advantages for deep networks



Neuron Activation

- A neuron is only activated if $x > 0$



- If $a = \mathrm{ReLU}(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b) > 0$
- the weighted activations are larger than the negative bias $-b$

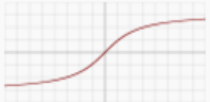
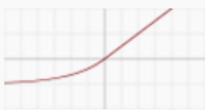
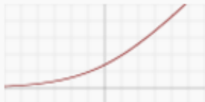
Common Activation Functions

There are different activation functions

- sigmoid()
- ReLU()
- tanh()
- atan()
- softplus()
- identity()
- step-function()
- …

ReLU is often used

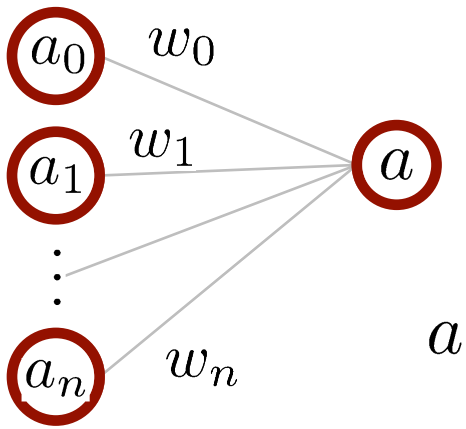
Illustration

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

[Courtesy of S. Sharma]

Function Behind a Neuron

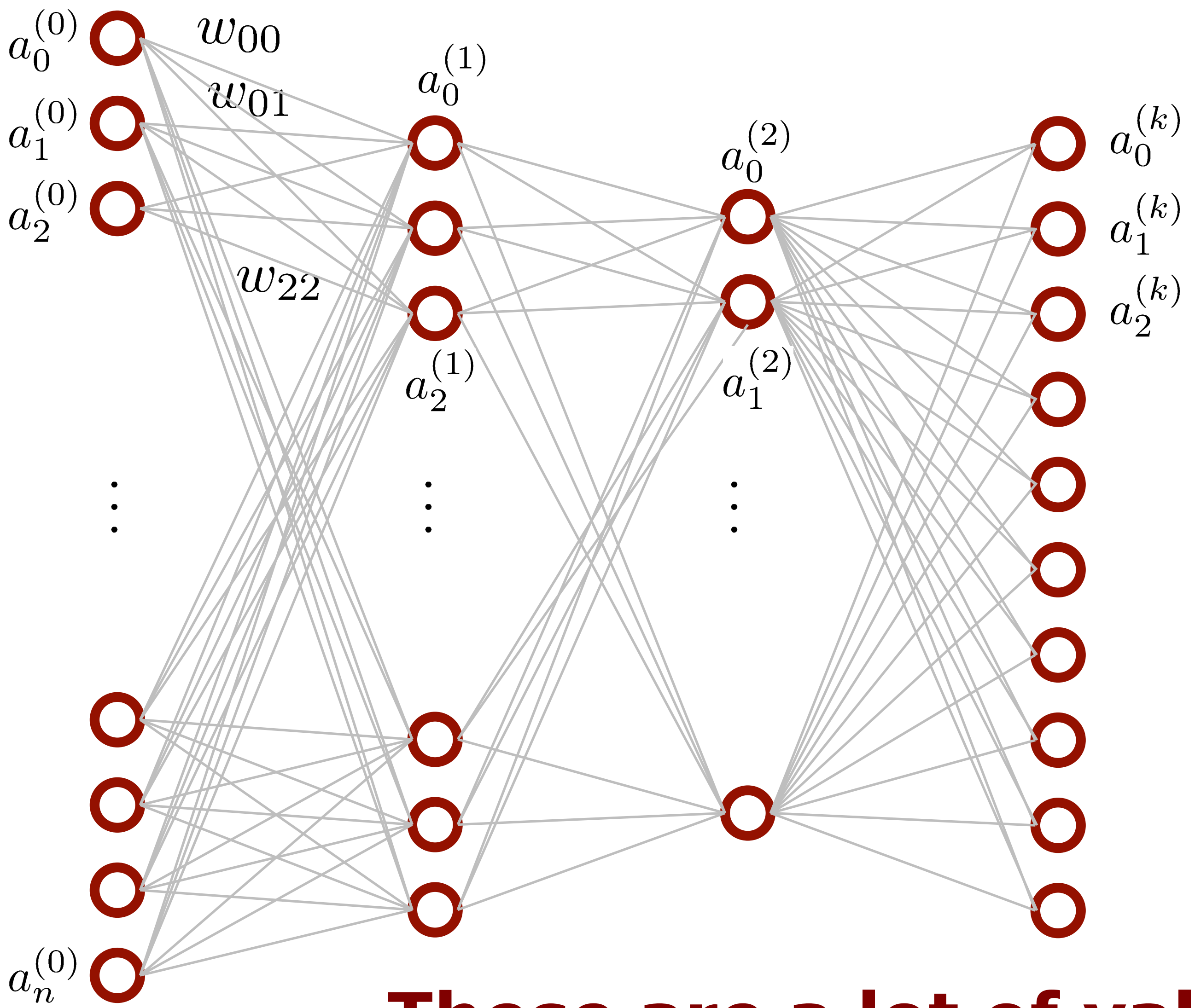
- Neuron gets activated if the weighted sum of input activations is large enough (larger than the negative bias)



$$a = \sigma\left(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b\right)$$

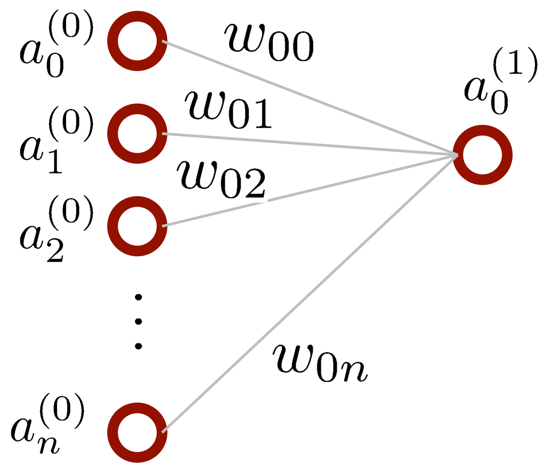
- This is the case for **all neurons** in the neural network

For All Neurons...



These are a lot of values!

Let’s Use a Matrix Notation



$$a_0^{(1)} = \sigma\left(w_{00}a_0^{(0)} + w_{01}a_1^{(0)} + \ldots + w_{0n}a_n^{(0)} + b_0^{(1)}\right)$$

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{00} & w_{01} & \ldots & w_{0n} \\ w_{10} & w_{11} & \ldots & w_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k0} & w_{k1} & \ldots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}\right)$$

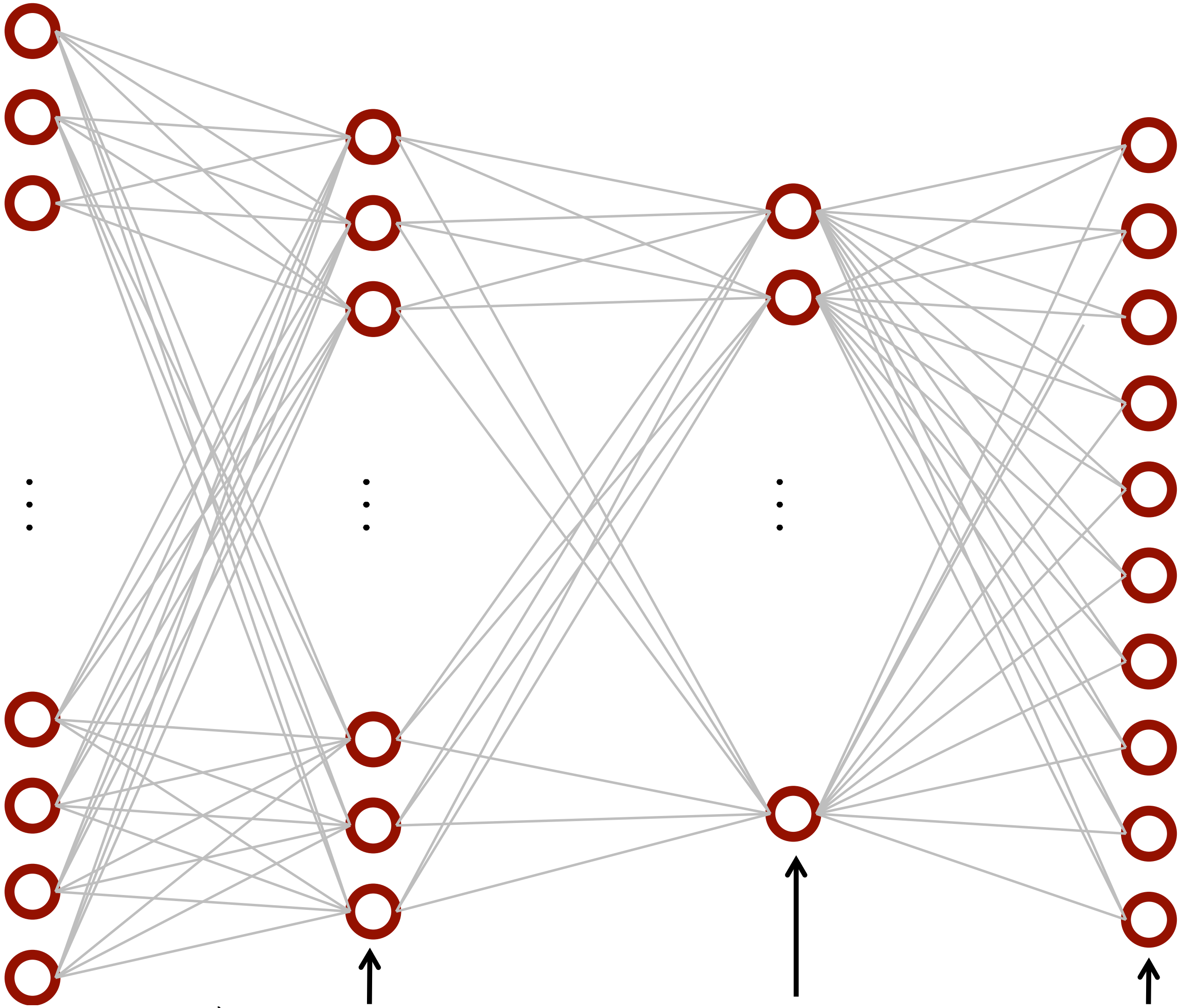
Each Layer Can Be Expressed Through Matrix Multiplications

layer 1 **layer 0**

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ w_{10} & w_{11} & \dots & w_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k0} & w_{k1} & \dots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \right)$$

$$\boldsymbol{a}^{(1)} = \sigma \left(W \boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)} \right)$$

Do It Layer by Layer...



$$\sigma\left(W^{(1)}\,\boldsymbol{a}^{(0)}+\boldsymbol{b}^{(1)}\right)=\boldsymbol{a}^{(1)} \qquad \cdots\,\sigma\left(W^{(k)}\,\boldsymbol{a}^{(k-1)}+\boldsymbol{b}^{(k)}\right)=\boldsymbol{a}^{(k)}$$

Do It Layer by Layer...

input = layer 0 $\boxed{\boldsymbol{x} = \boldsymbol{a}^{(0)}}$

layer 1 $\sigma\left(W^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}\right) = \boldsymbol{a}^{(1)}$

layer 2 $\sigma\left(W^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}\right) = \boldsymbol{a}^{(2)}$

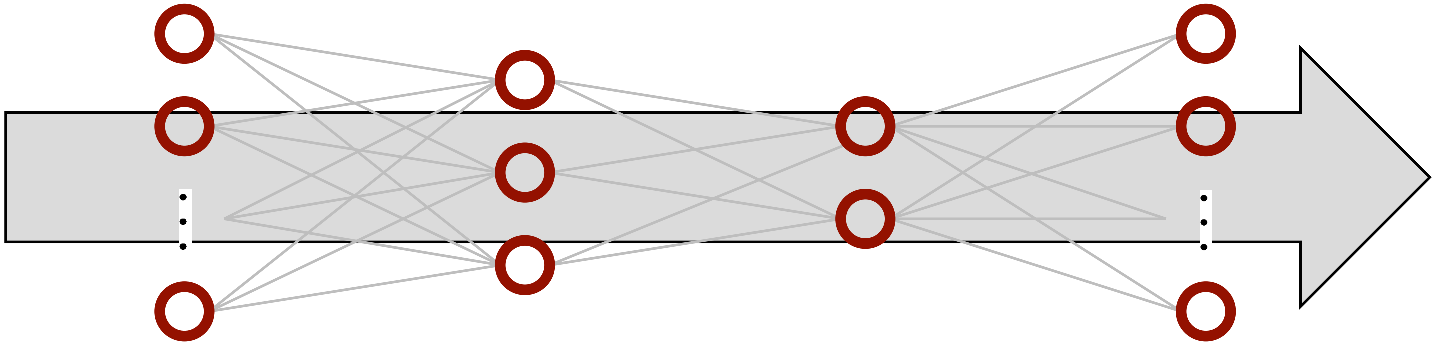
$\vdots$

layer k = **output** $\sigma\left(W^{(k)}\boldsymbol{a}^{(k-1)} + \boldsymbol{b}^{(k)}\right) = \boxed{\boldsymbol{a}^{(k)} = \hat{\boldsymbol{y}}}$

That not much more than linear algebra...

Feedforward Networks

- MLPs are feedforward networks
- The information flows form left to right
- There are no loops



- Such networks are called feedforward networks
- There exist other variants (eg, RNNs)

Example:
Handwritten Digit Recognition

Handwritten Digit Recognition

$=5$

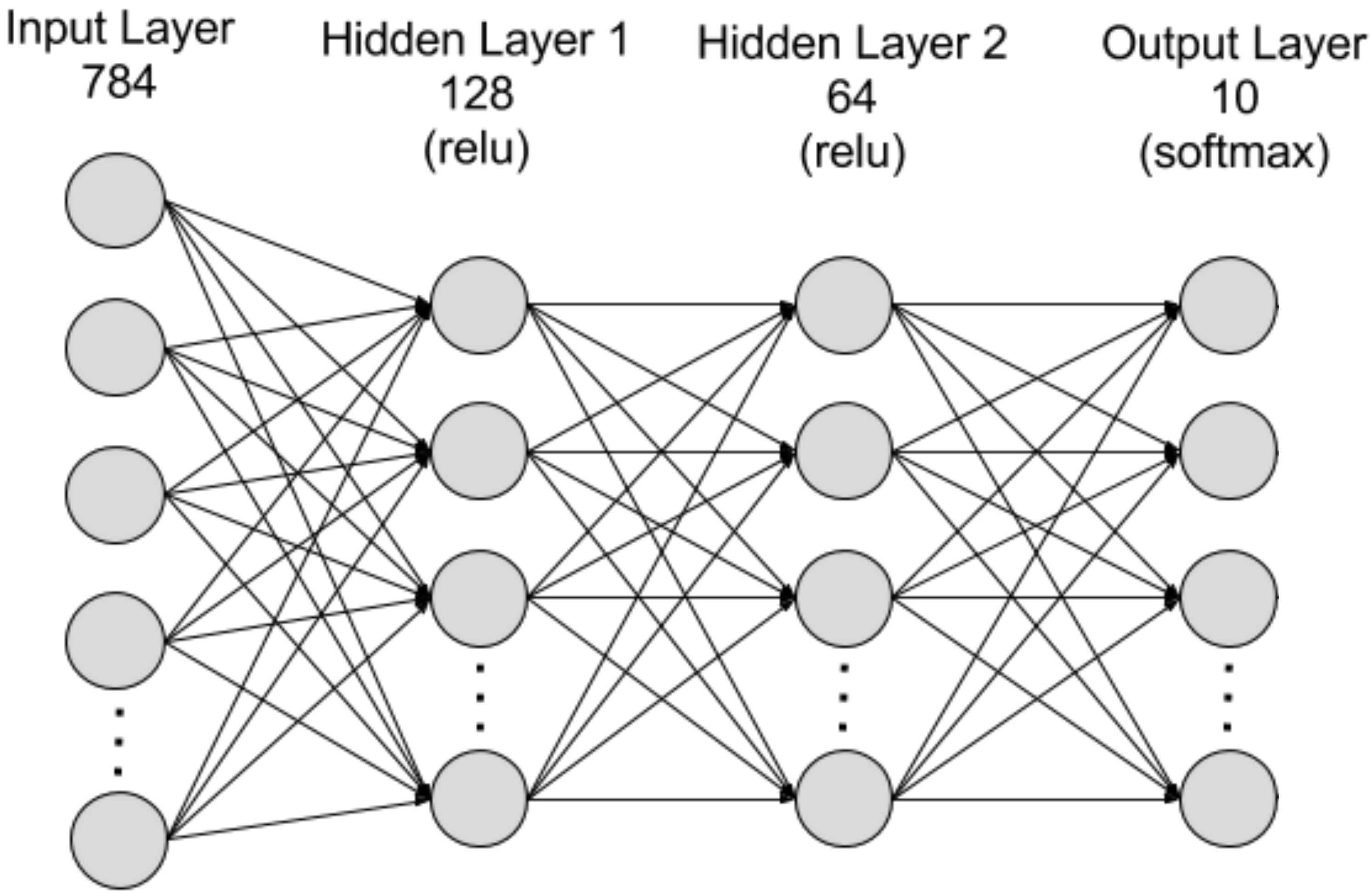
28x28 pixel image

[Image courtesy: Nielsen]

Handwritten Digit Recognition

[Image courtesy: Nielsen/Lecun]

A Basic MLP Recognizing Digits

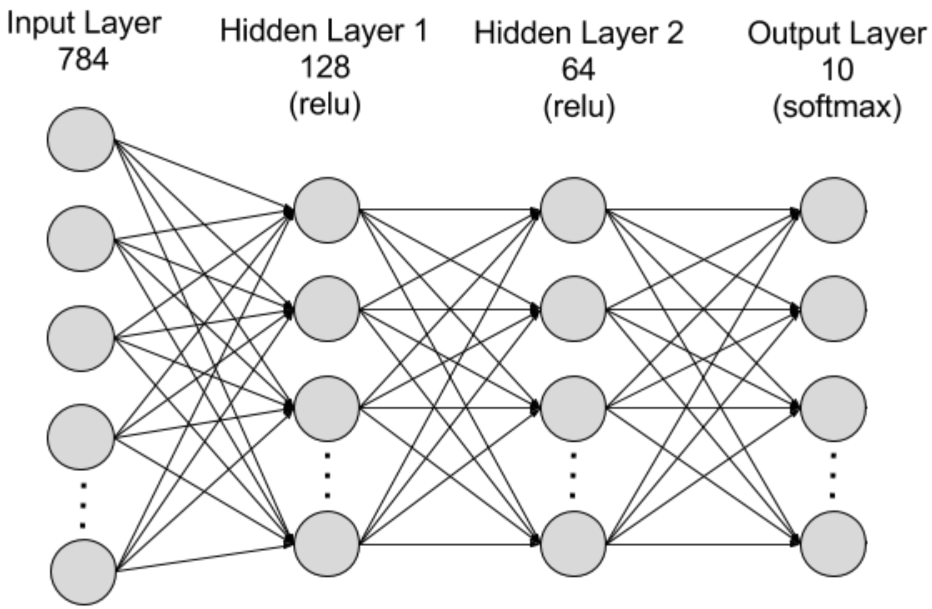


[Image courtesy: Nielsen]

Images to Digits - A Mapping from 784 to 10 Dimensions

28x28 pixel input images

(784 dim)



$$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\\vdots\end{bmatrix}$$

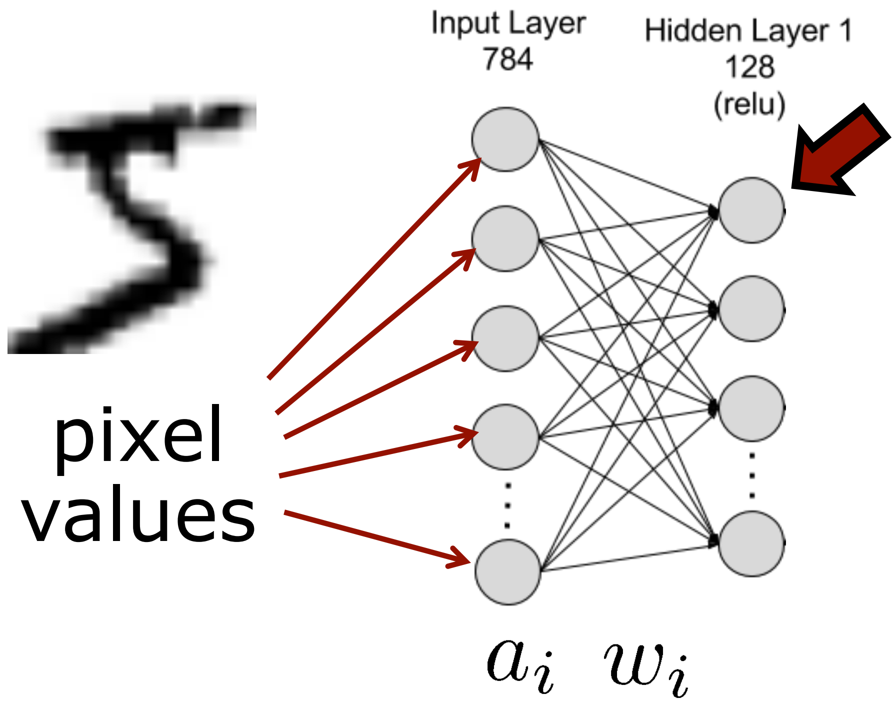
output vector

(10 dim)

[Partial image courtesy: Nielsen]

What Happens in the Layers?

What Happens in the 1st Layer?



784 input activations = pixel intensities

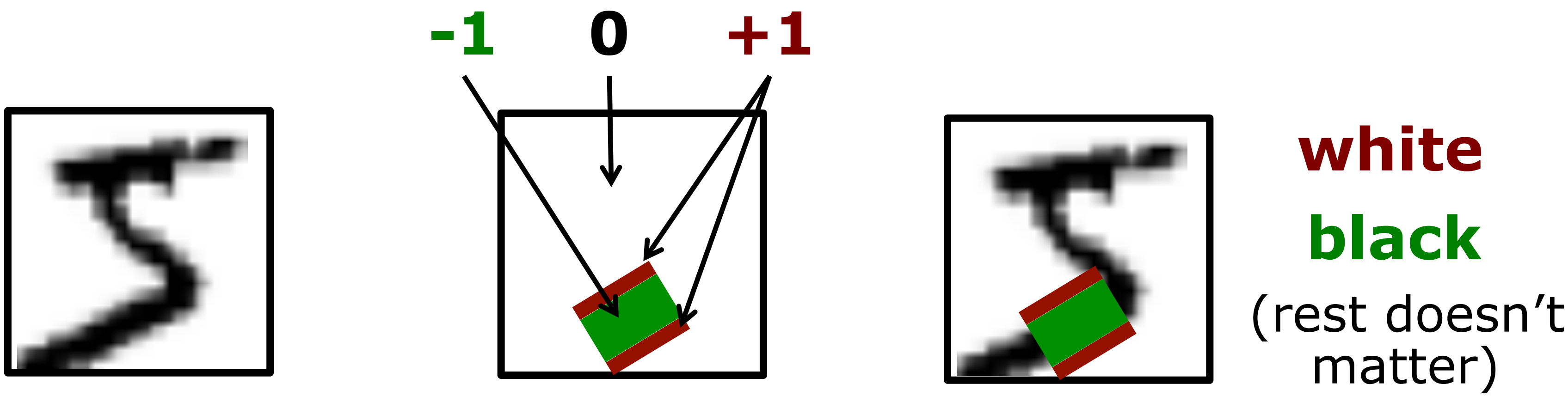
784 weights = weights for pixel intensities

What Happens in the 1st Layer?

784 input activations = pixel intensities

784 weights = weights for pixel intensities

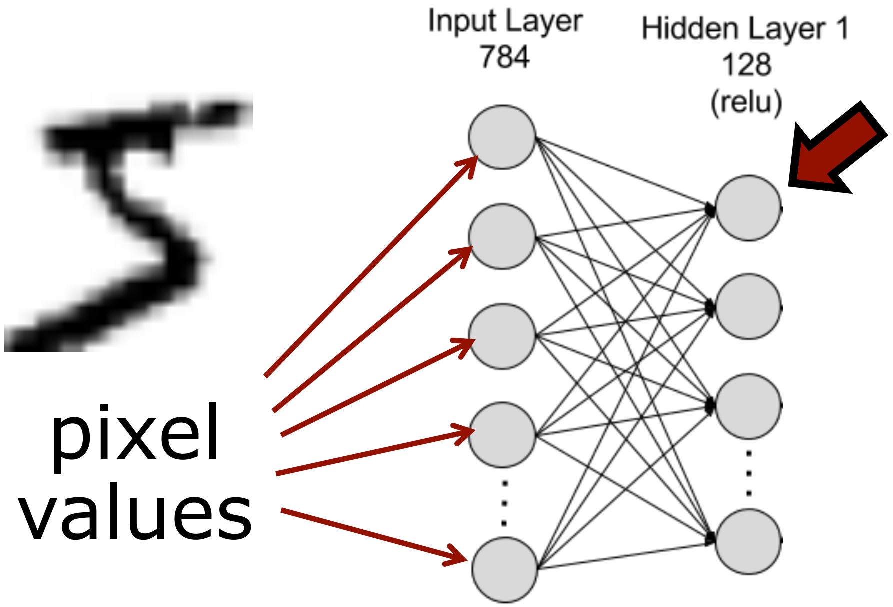
treat activations and weights as images



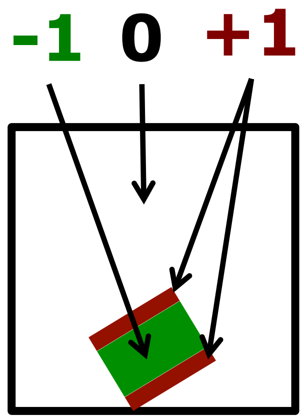
pixel values a_i weights w_i effect on the weighted sum

What Happens in the 1st Layer?

a_i w_i



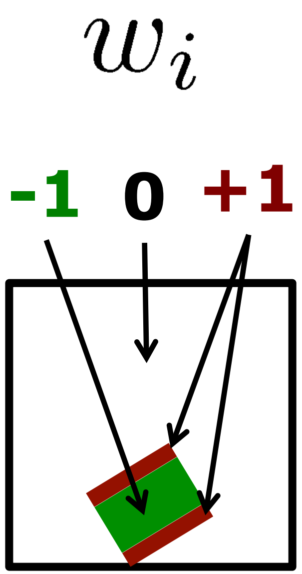
pixel values



weights tell us what matters for activating the neuron!

individual "weight images" for a neuron support individual patterns in the image

Link to Local Operators Defines Through Convolutions



weights tell us what matters

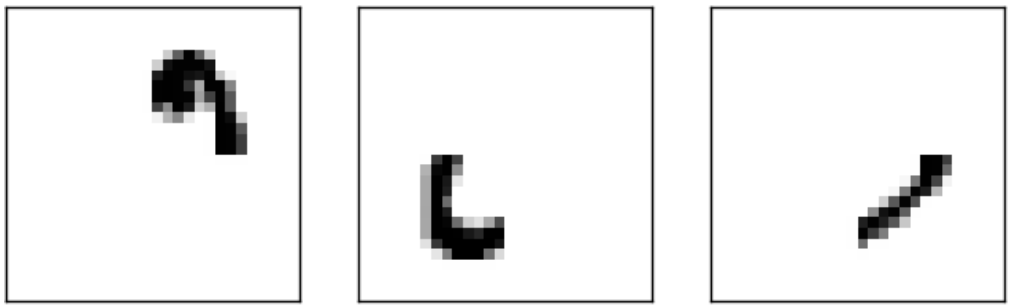
- Direct link to defining image operators through convolutions

Here:

- Global (not local) operators
- Weight matrix does not (yet) "slide over image"

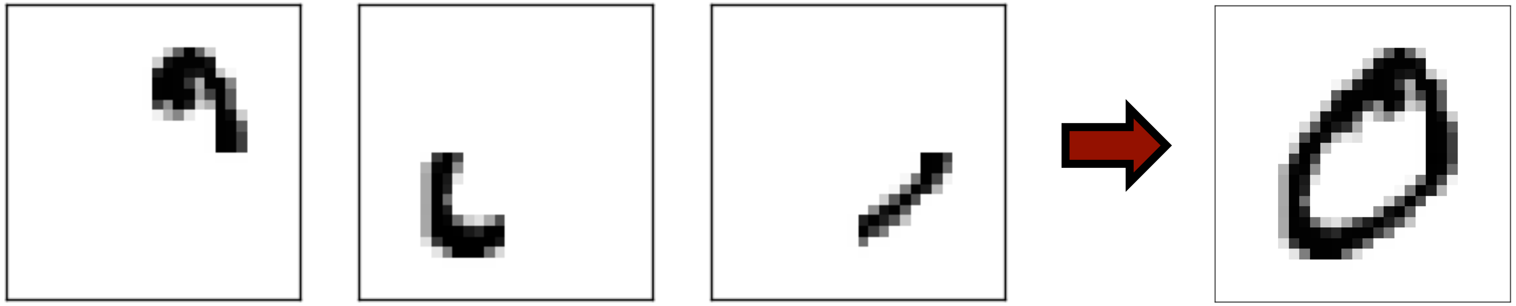
Weights & Bias = Patterns

- **Weights define** the **patterns** to look for in the image
- **Bias** tells us how well the image must **match** the pattern
- Activation functions **"switches the neuron on"** if it matches the pattern



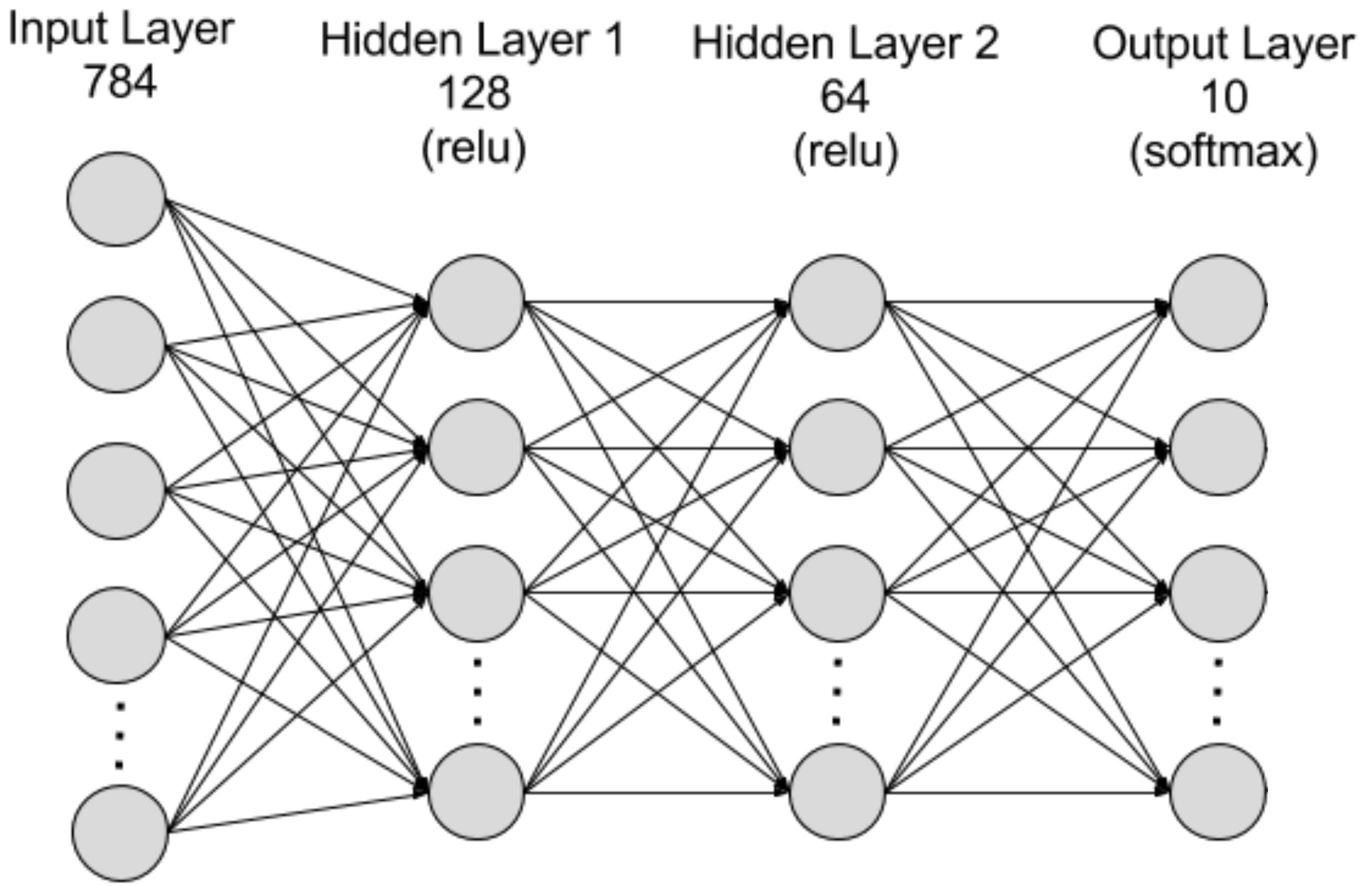
What Happens in the 2nd Layer?

- The weights in layer 2 tell us which 1st layer patterns should be combined
- The deeper we go, the more patterns get arranged and combined



- The last layer decides, which final patterns make up a digit

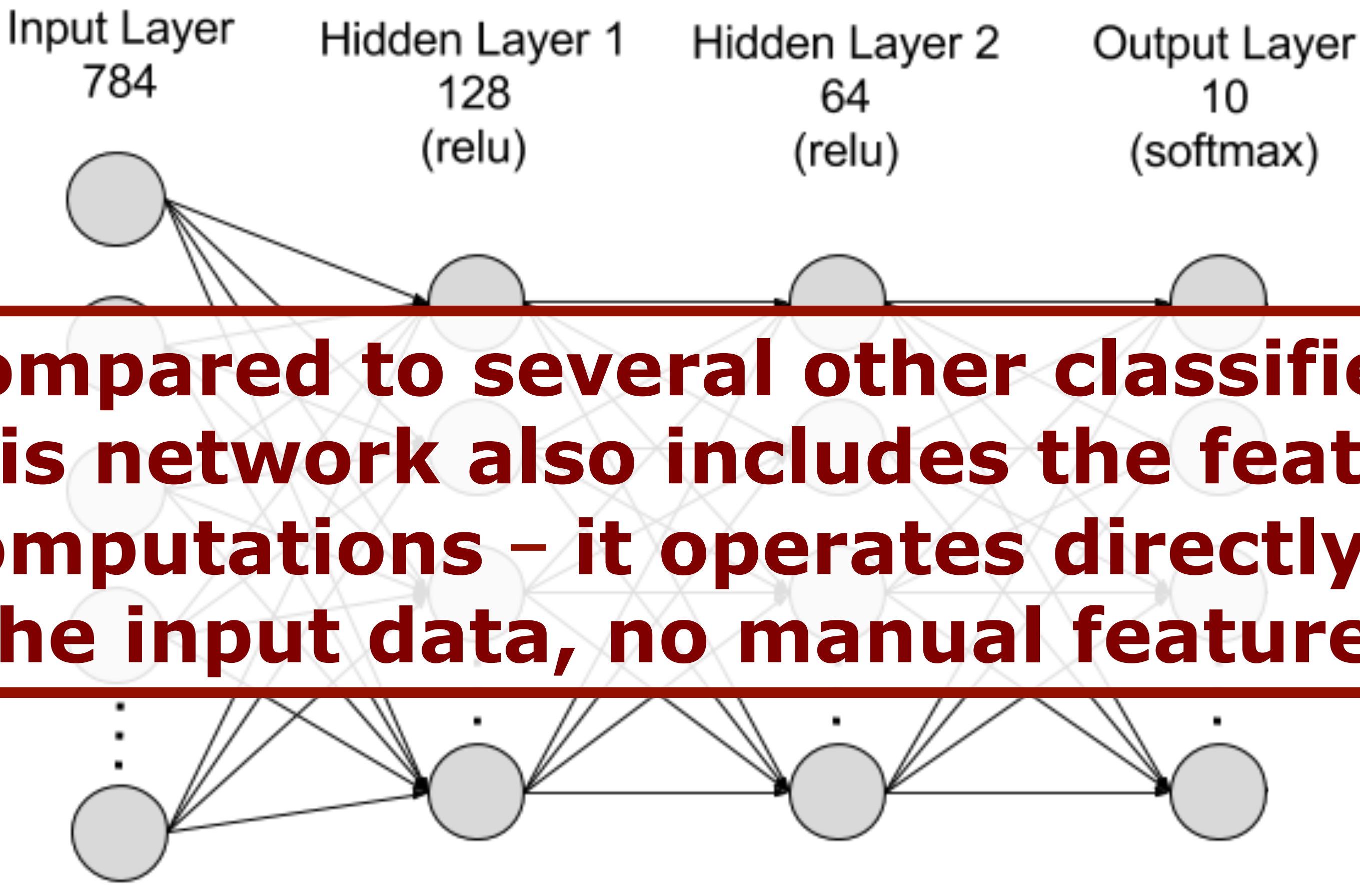
What Happens in the Layers?



raw pixels | **simple patterns** | **combined patterns** | **patterns to digits**

[Image courtesy: Nielsen]

No Manual Features

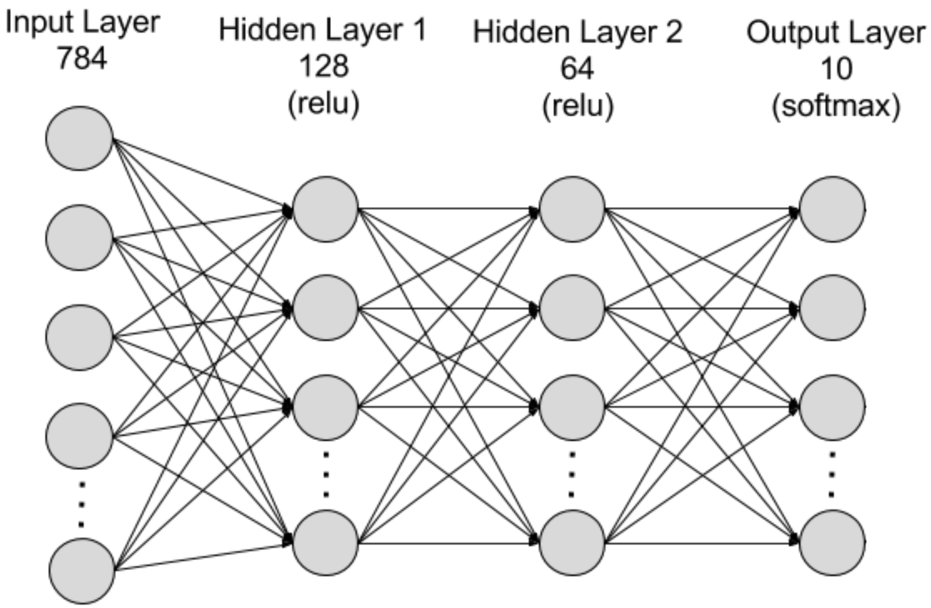


Compared to several other classifiers, this network also includes the feature computations – it operates directly on the input data, no manual features!

raw pixels | **simple patterns** | **combined patterns** | **patterns to digits**

[Image courtesy: Nielsen]

Classification Performance



$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

Such a simple MLP achieves a correct classification for ~96% of the examples

[Partial image courtesy: Nielsen]

Classification Performance

- A simple MLP achieves a classification accuracy of ~96%
- Note that there are tricky cases

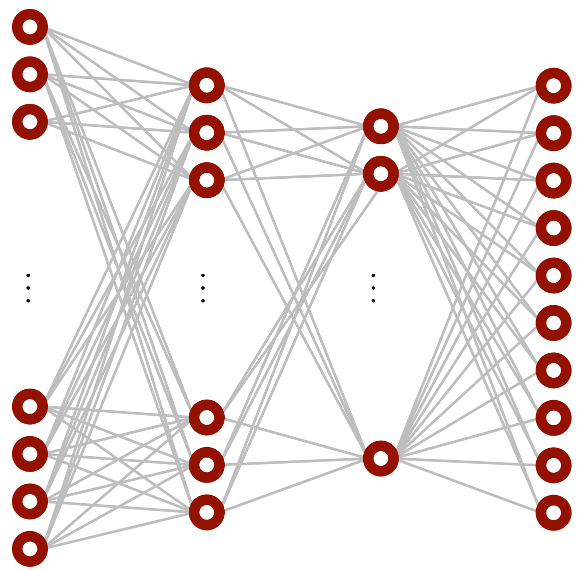
- That is a good performance for a simple model!
- Improved networks achieve ~99%

How to Design a Neural Network?

How to Make the Network Compute What We Want?

- So far, the network is a recipe for sequentially performing computations
- **Structure** and **parameters** are the **design choices**
- How to set them?

Learning!



Summary – Part 1

- What are neurons and neural networks
- Lots of different networks exists
- Focus: multi-layer perceptrons (MLP)
- Activations, weights, bias
- Networks have many parameters
- “It’s just a bunch of matrices and vectors”
- MLP for simple image classification
- Part 2: Learning the parameters

Literature & Resources

- Online Book by Michael Nielsen, Chapter 1:
  http://neuralnetworksanddeeplearning.com/chap1.html
- Nielsen, Chapter 1, Python3 code:
  https://github.com/MichalDanielDobrzanski/DeepLearningPython
- MNIST database:
- http://yann.lecun.com/exdb/mnist/
- Grant Sanderson, Neural Networks
  https://www.3blue1brown.com/
- Alpaydin, Introduction to Machine Learning

Slide Information

- The slides have been created by Cyrill Stachniss as part of the photogrammetry and robotics courses.
- **I tried to acknowledge all people from whom I used images or videos. In case I made a mistake or missed someone, please let me know.**
- Huge thank you to Grant Sanderson (3blue1brown) for his great educational videos that influenced this lecture.
- Thanks to Michael Nielsen for his free online book & code
- If you are a university lecturer, feel free to use the course material. If you adapt the course material, please make sure that you keep the acknowledgements to others and please acknowledge me as well. To satisfy my own curiosity, please send me email notice if you use my slides.

Cyrill Stachniss, cyrill.stachniss@igg.uni-bonn.de