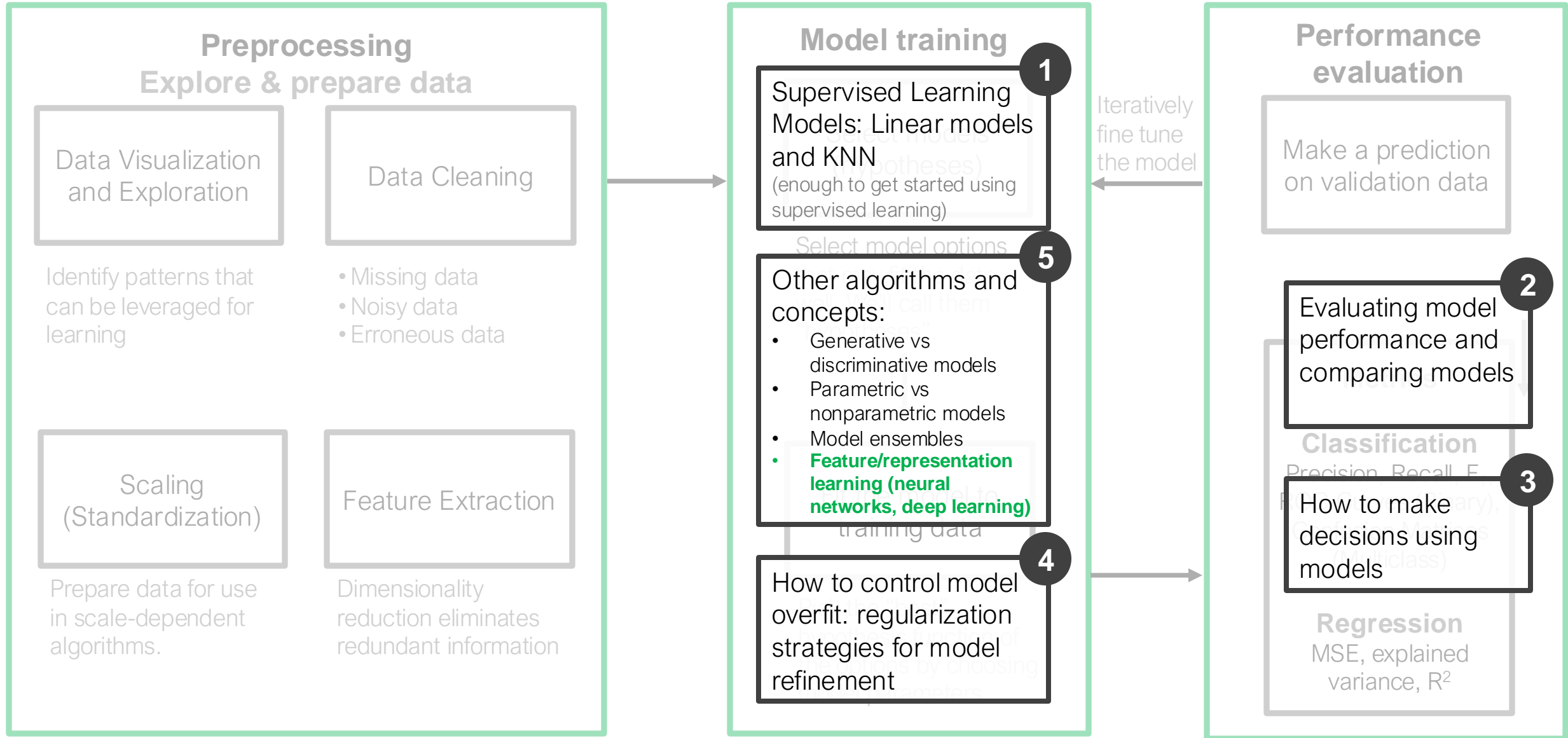


Neural Networks I

Supervised learning in practice



What's the hype around neural networks?

Character/handwriting recognition

Self-driving cars

Natural language processing and translation

Speech recognition

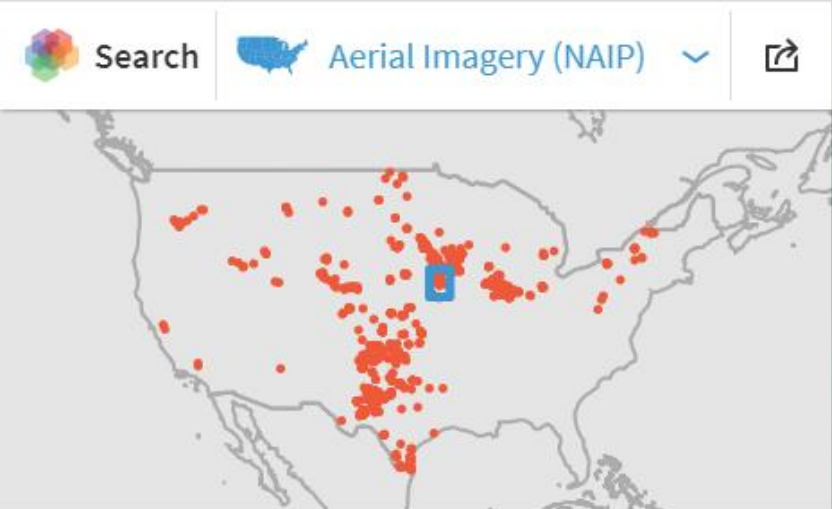
Medical devices, diagnosis, and treatment

Materials development

Automated financial trading systems

Industrial automation

Computer vision applications...



Top 1000 Closest Matches: 1 [Clear Results](#)



Geovisual Search

<https://search.descarteslabs.com/>

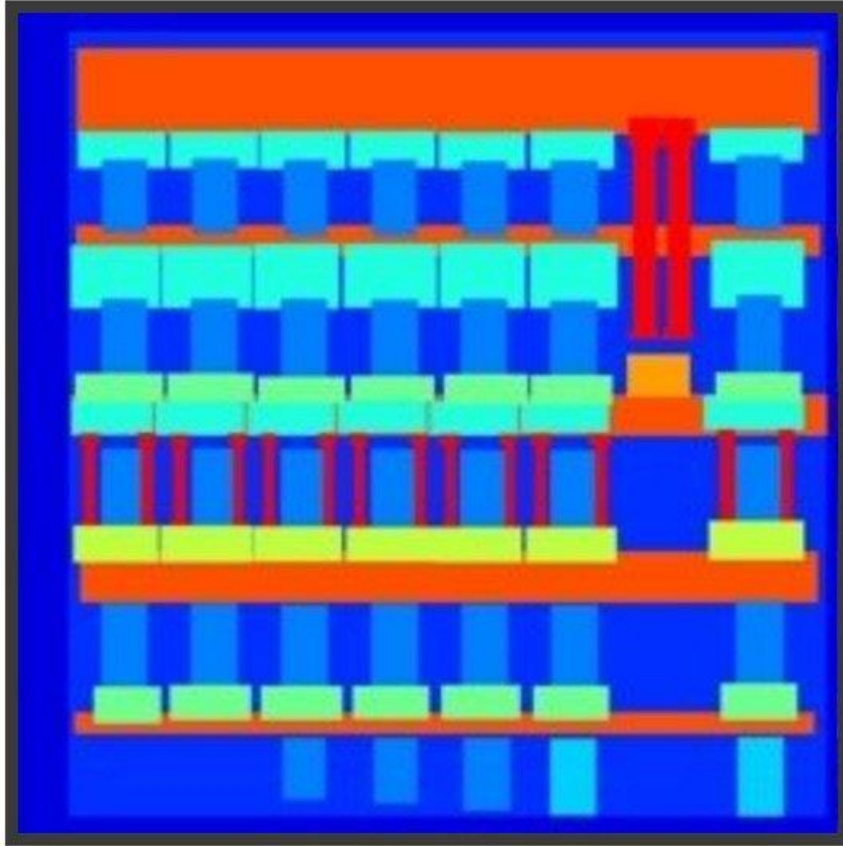


Image-to-image translation

TOOL

- background
- wall
- door
- window**
- window sill
- window head
- shutter
- balcony
- trim
- cornice
- column
- entrance

INPUT



pix2pix
process

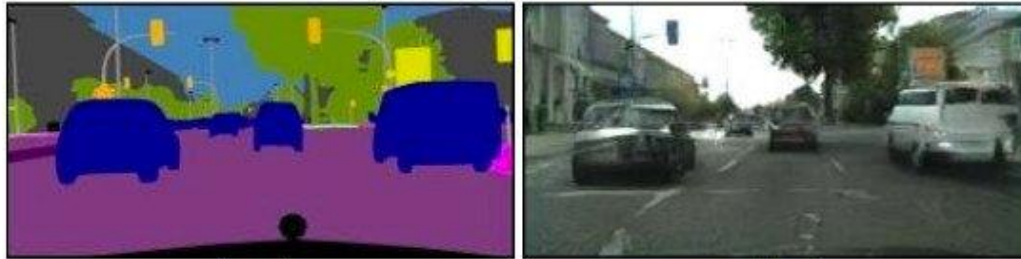
OUTPUT



Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint (2017).

Image-to-image translation

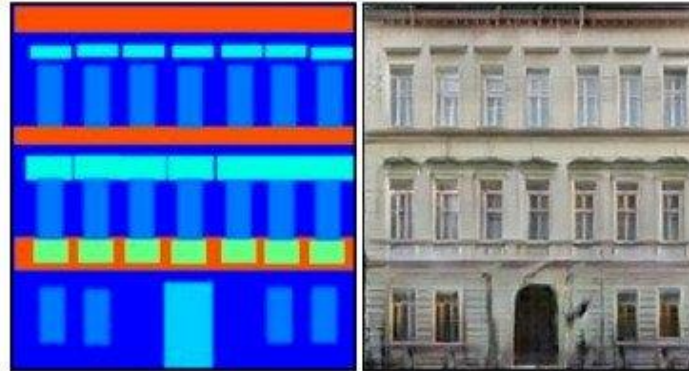
Labels to Street Scene



input

output

Labels to Facade



input

output

BW to Color



input

output

Aerial to Map



input

output

Day to Night



input

output

Edges to Photo



input

output

Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint (2017).

Image-to-image translation



Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint (2017).

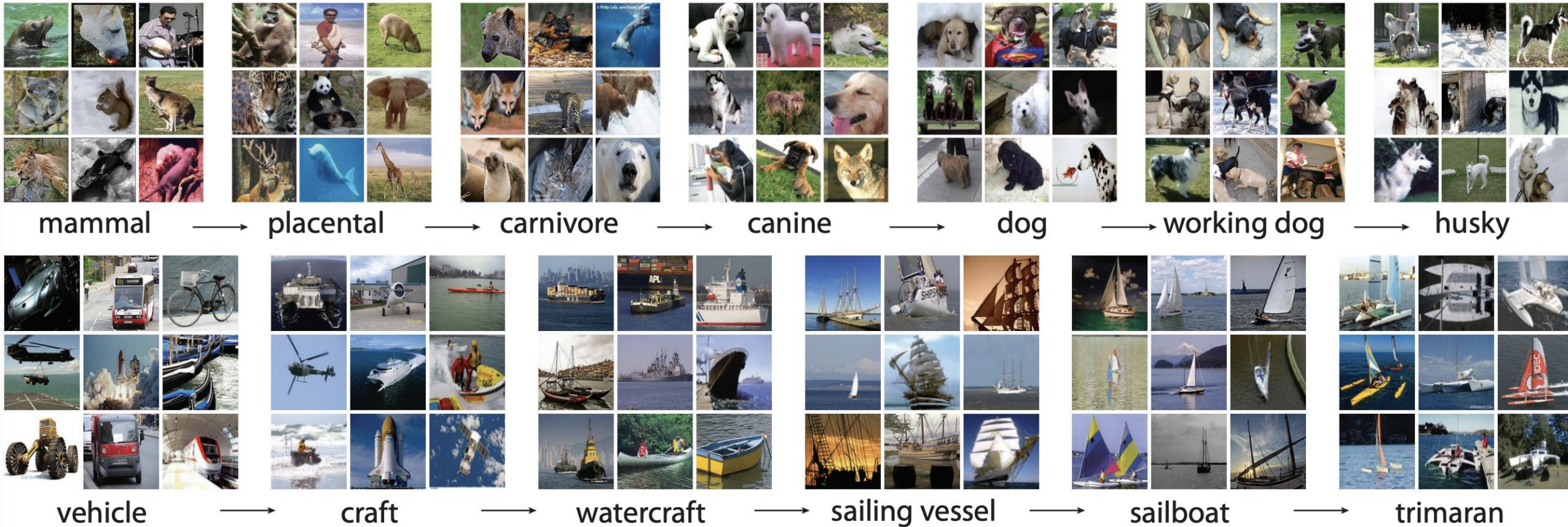
Image Style Transfer



Dumoulin, Vincent, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style." CoRR, abs/1610.07629 2.4 (2016): 5.

ImageNet Competition

- Image classification challenge
- 14,197,122 annotated images
- 1,000 classes

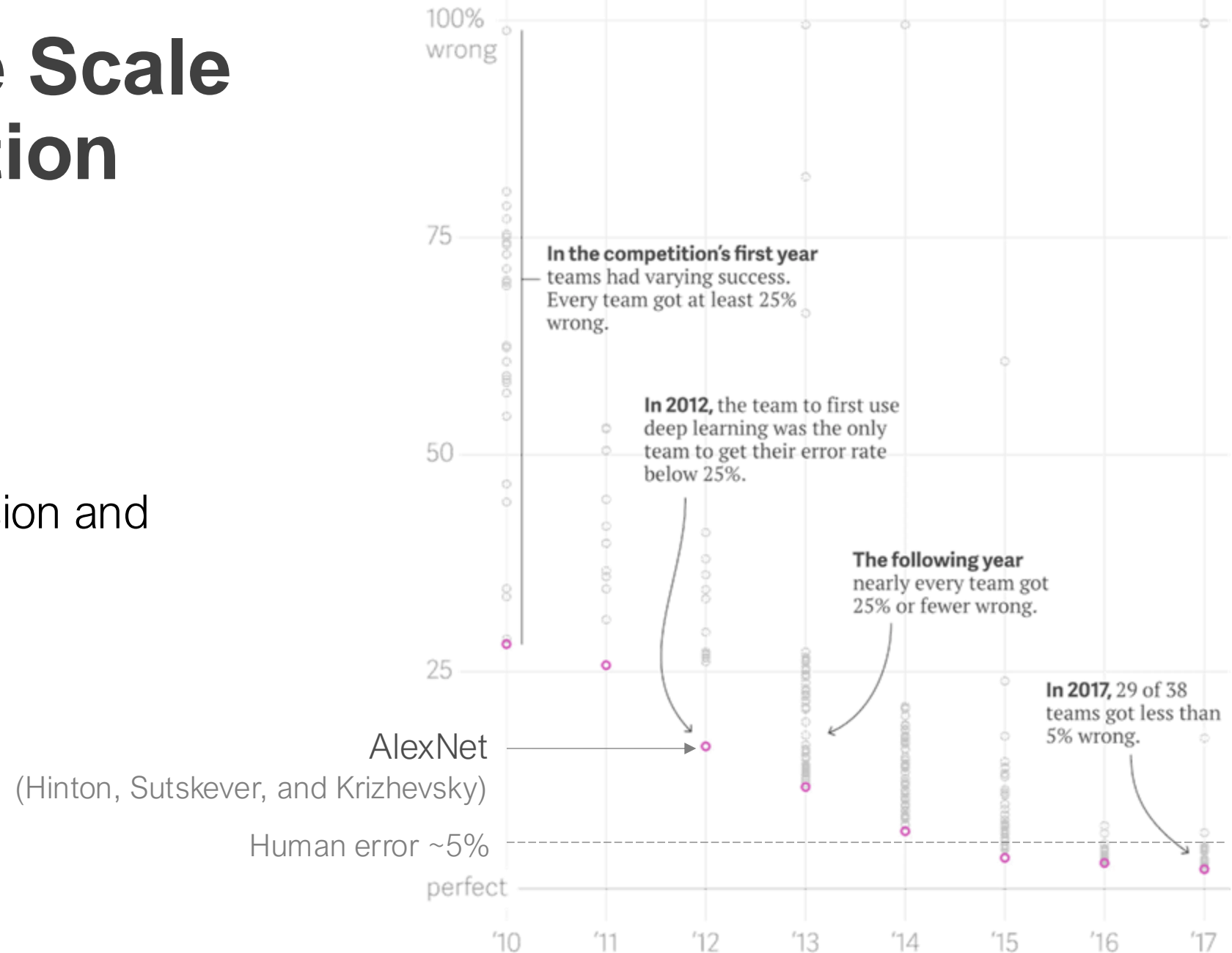


Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

ImageNet Large Scale Visual Recognition Challenge

Fei-Fei Li et al. 2010 ([link](#))

Competition at:
Conference on Computer Vision and
Pattern Recognition (CVPR)



Source: Quartz, [link](#)

David Yanofsky | Quartz

Data: ImageNet

ChatGPT: How do I teach how neural networks work?

Teaching how neural networks work can be a complex task, but here are some general steps to help guide you:

1. Start with the basics: Before diving into the details of neural networks, it's essential to ensure that your students have a basic understanding of concepts like linear algebra, calculus, and probability.
2. Introduce the concept of neurons: Start by explaining how neurons work in the human brain and how this concept has been translated into artificial neural networks.
3. Explain the different layers of a neural network: Discuss the input layer, hidden layers, and output layer, explaining the role each layer plays in processing information.
4. Discuss activation functions: Activation functions help to determine the output of a neuron. Explain how activation functions like ReLU, sigmoid, and tanh work and why they are important.
5. Teach backpropagation: Backpropagation is the algorithm used to update the weights and biases of a neural network. Explain how it works and why it's essential.
6. Discuss types of neural networks: There are many types of neural networks, including feedforward, convolutional, and recurrent neural networks. Teach the differences between them and when to use each one.
7. Provide hands-on experience: Provide examples and hands-on experience with coding neural networks. This can help your students to better understand how neural networks work in practice.
8. Discuss current research and applications: Finally, discuss current research and applications of neural networks, including computer vision, natural language processing, and robotics. This can help to motivate your students and demonstrate the real-world impact of neural networks.

Remember, teaching neural networks is a process that requires time and patience. Break down the concepts into smaller, digestible parts and provide lots of examples and opportunities for practice.



Neural networks are not appropriate for every problem

- Small datasets
- Heterogeneous, tabular data
- Cases when model interpretability is paramount

What makes neural networks special?

Neural network learning is **representation learning**

Previous ML algorithms we discussed required us to manually determine feature transformations

Neural networks **learn** feature transformations

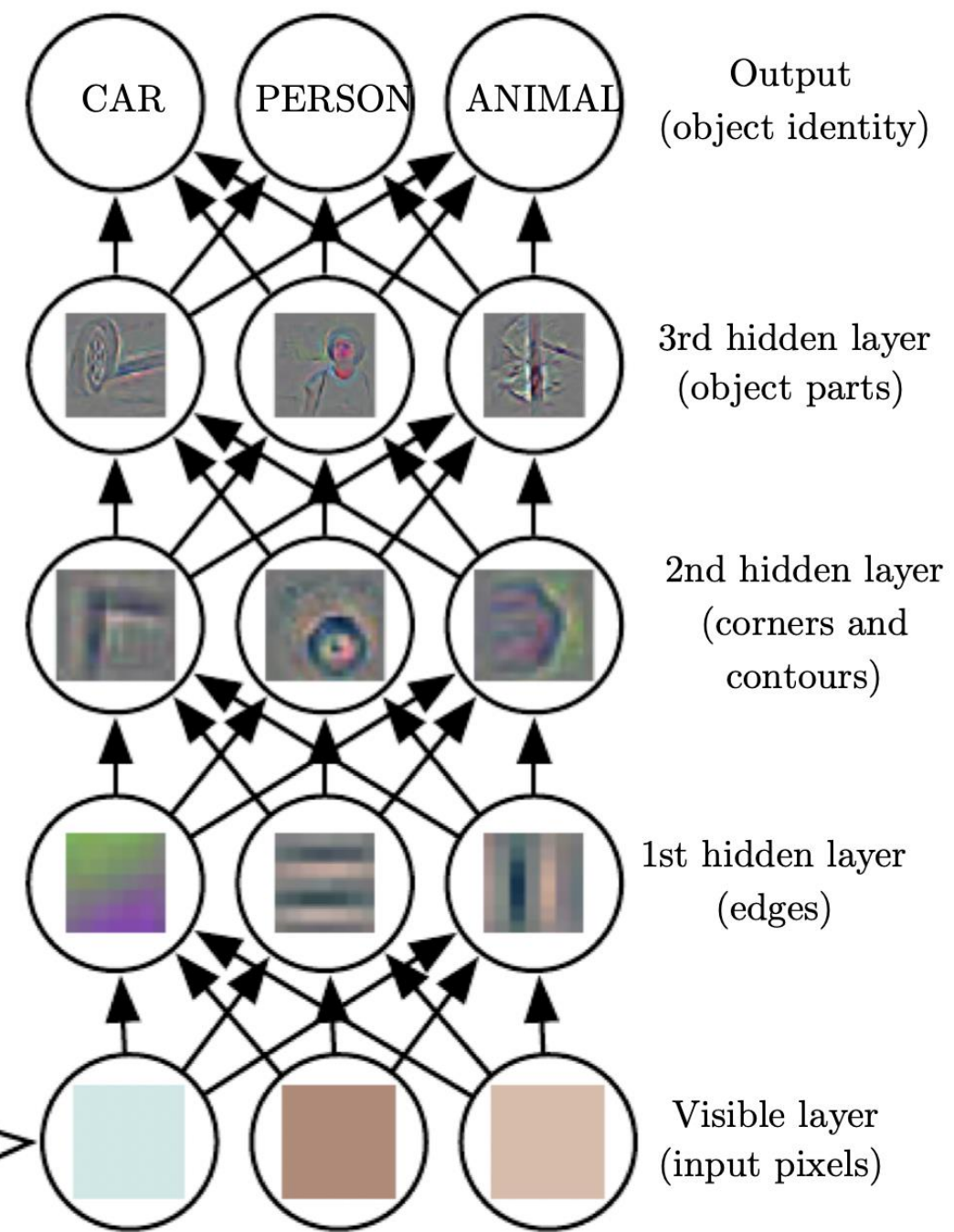
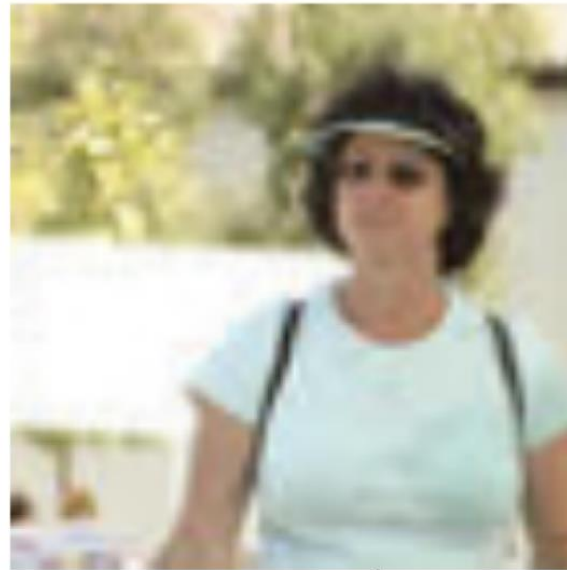


Image from Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. Deep learning (Vol. 1, No. 2). Cambridge: MIT press.

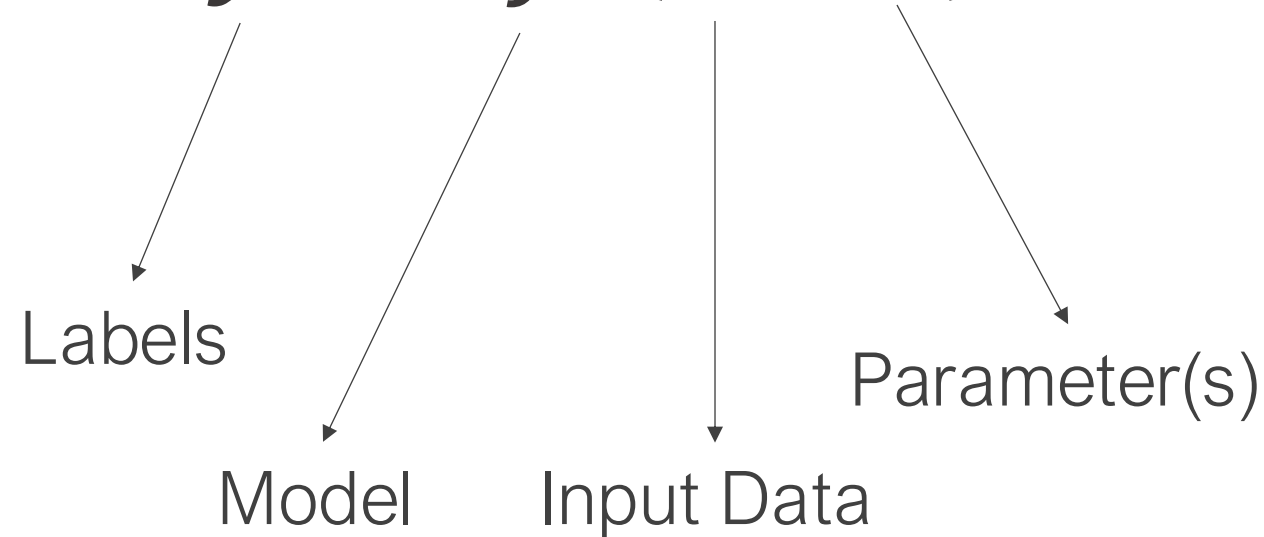
What is a neural network and **how does it work?**

How do we **optimize model weights?**
(i.e. how do we fit our model to data)

What are the challenges of using neural networks?

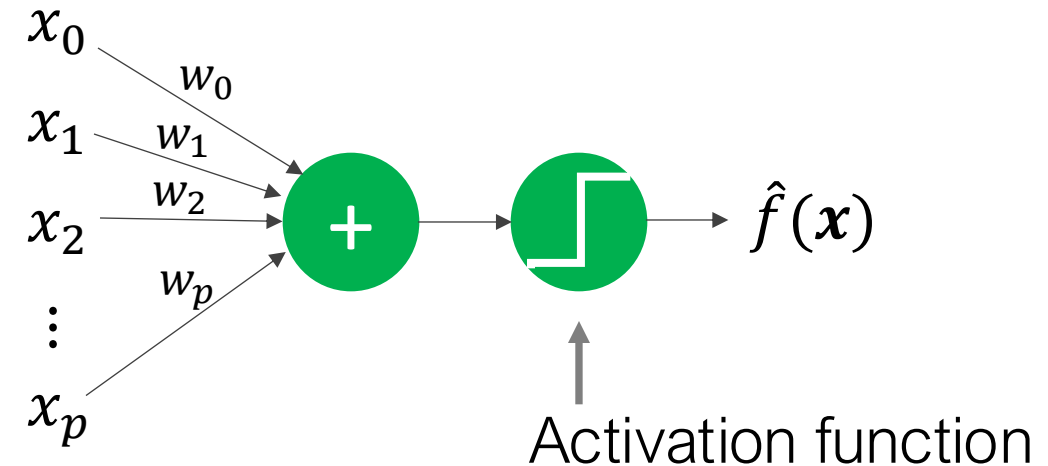
Recall our goal in supervised learning

$$y = f(x, w)$$



Perceptron

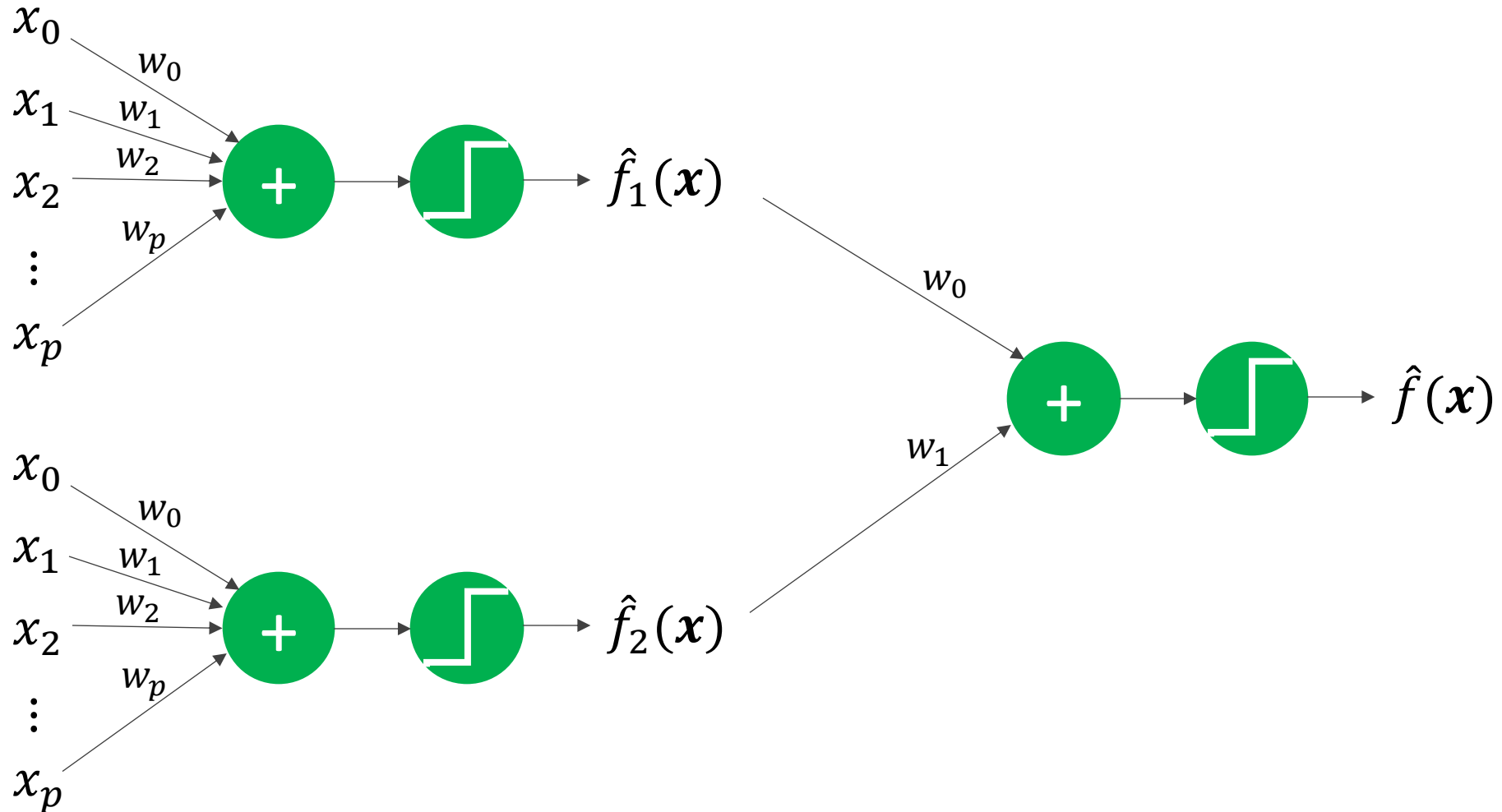
$$\hat{f}(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^p w_i x_i\right)$$



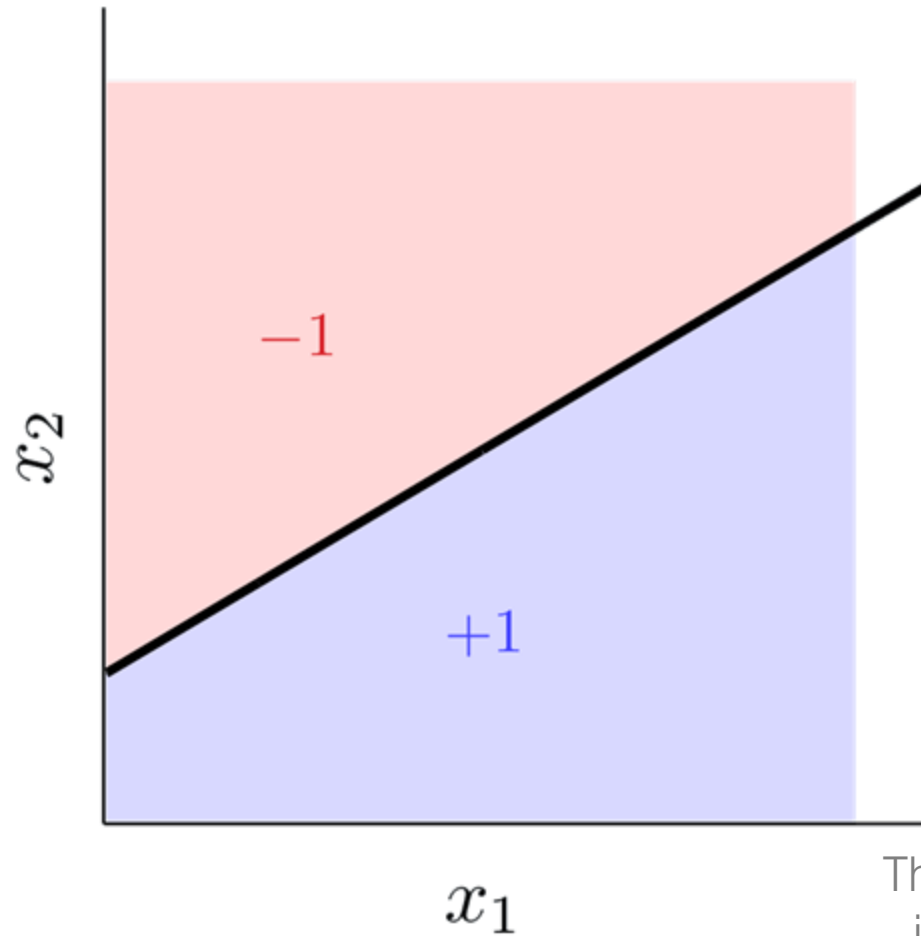
Source: Abu-Mostafa, Learning from Data, Caltech

Multilayer Perceptron

What if we stuck multiple perceptrons together?



Perceptron #1

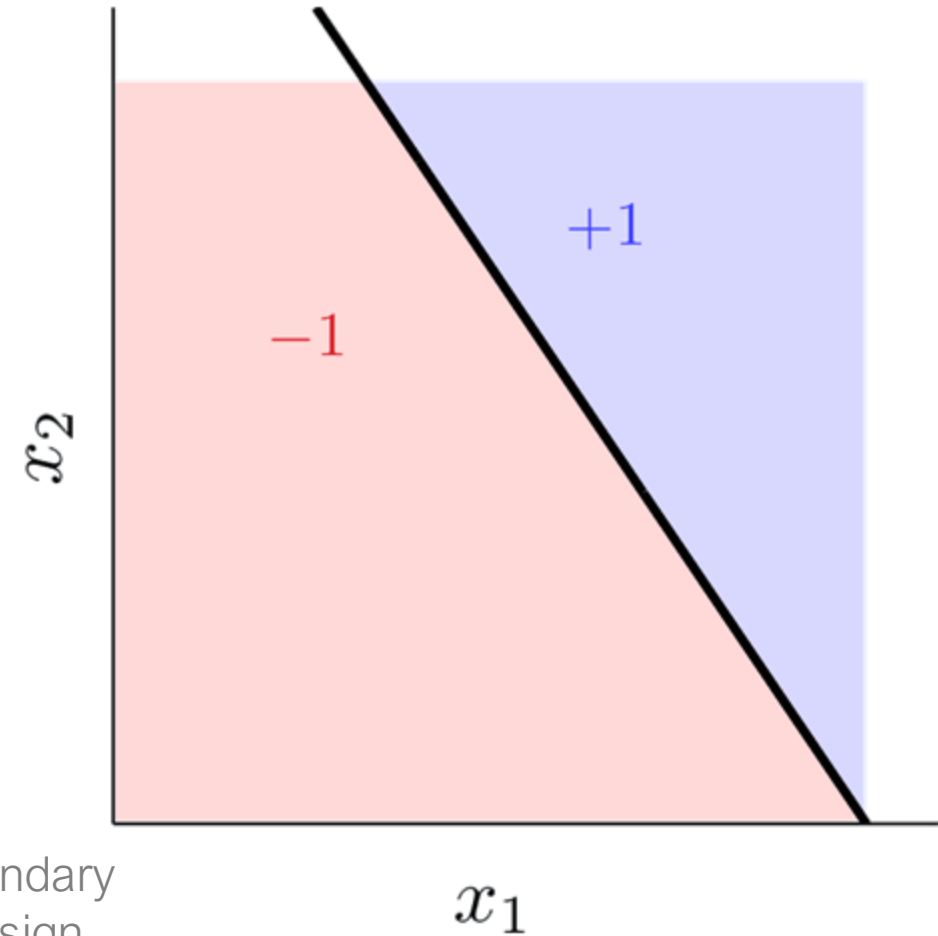


$$\hat{f}_1(\mathbf{x}) = \text{sign}(\mathbf{w}_1^T \mathbf{x})$$

The sharp boundary
is due to our sign
function



Perceptron #2



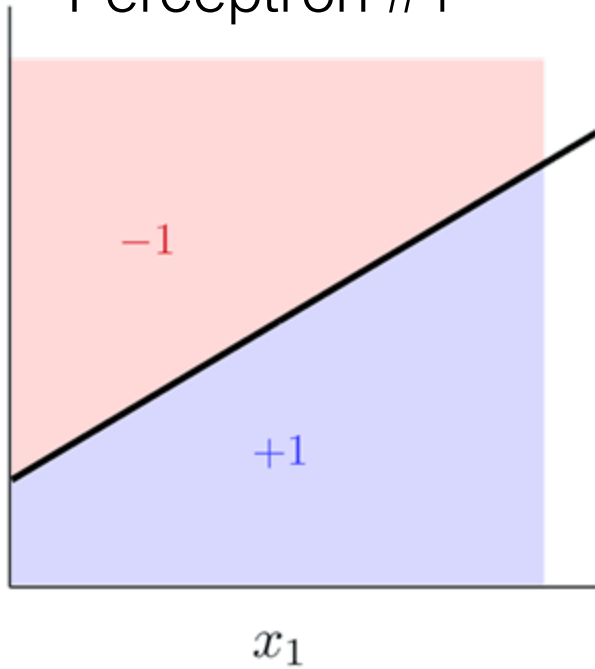
$$\hat{f}_2(\mathbf{x}) = \text{sign}(\mathbf{w}_2^T \mathbf{x})$$

Source: Abu-Mostafa, Learning from Data, Caltech

Multilayer perceptron:

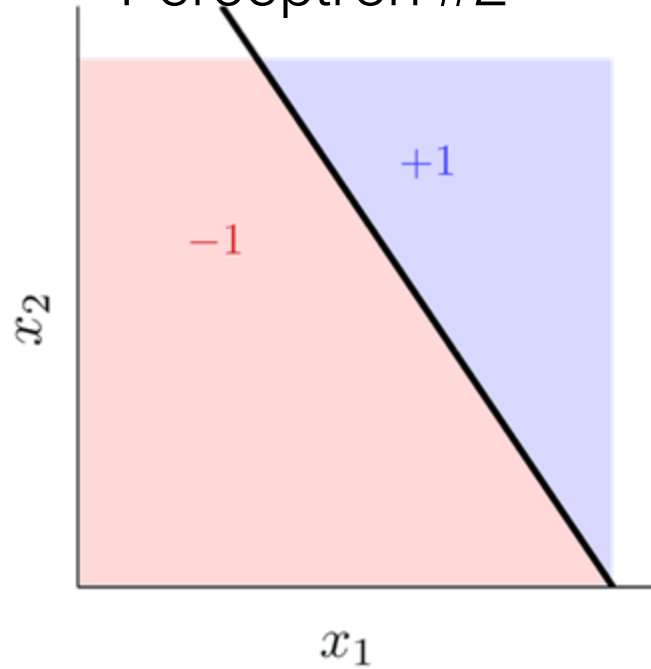
$$\hat{f}(\mathbf{x}) = \begin{cases} +1, & \hat{f}_1(\mathbf{x}) \neq \hat{f}_2(\mathbf{x}) \\ -1, & \hat{f}_1(\mathbf{x}) = \hat{f}_2(\mathbf{x}) \end{cases}$$

Perceptron #1

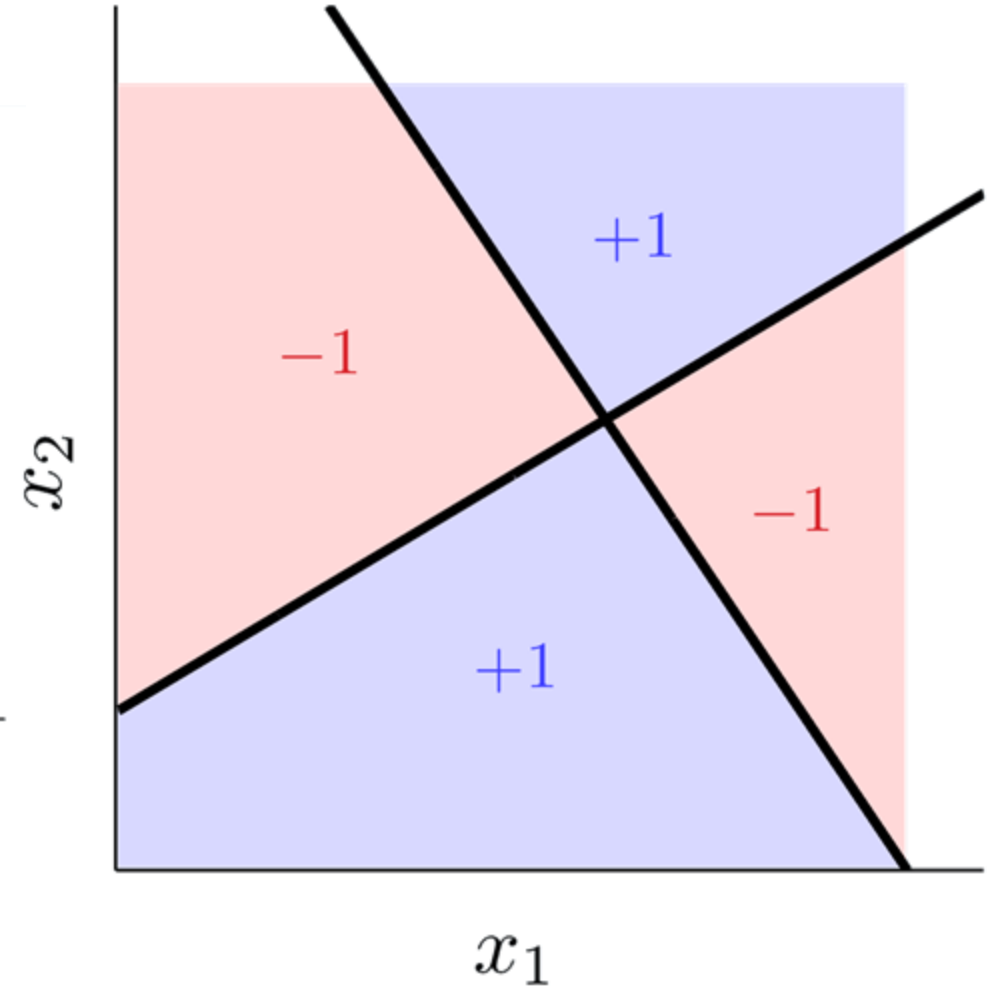


$$\hat{f}_1(\mathbf{x}) = \text{sign}(\mathbf{w}_1^T \mathbf{x})$$

Perceptron #2

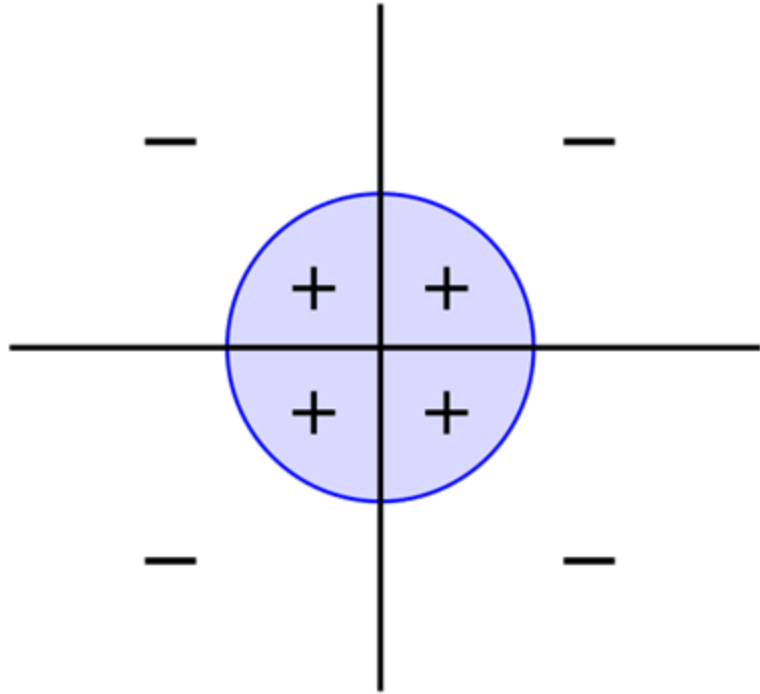


$$\hat{f}_2(\mathbf{x}) = \text{sign}(\mathbf{w}_2^T \mathbf{x})$$

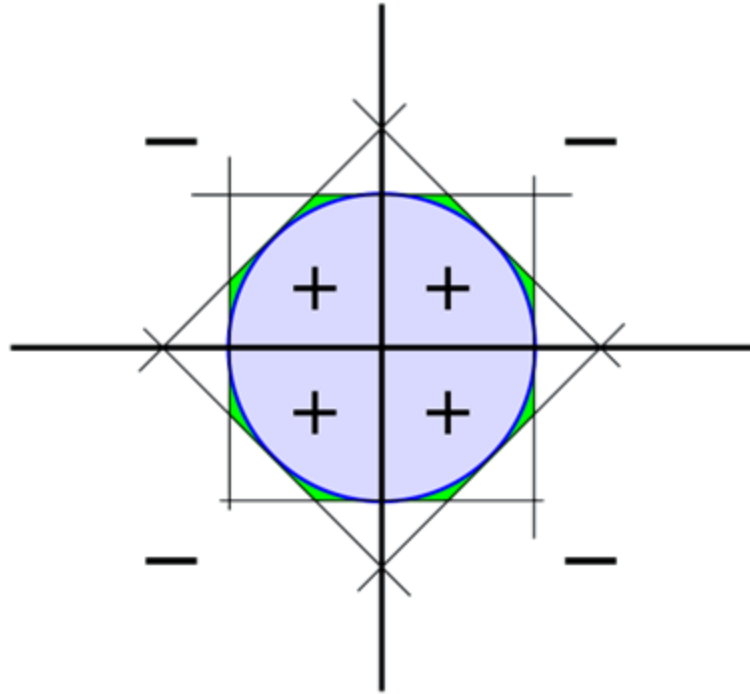


Source: Abu-Mostafa, Learning from Data, Caltech

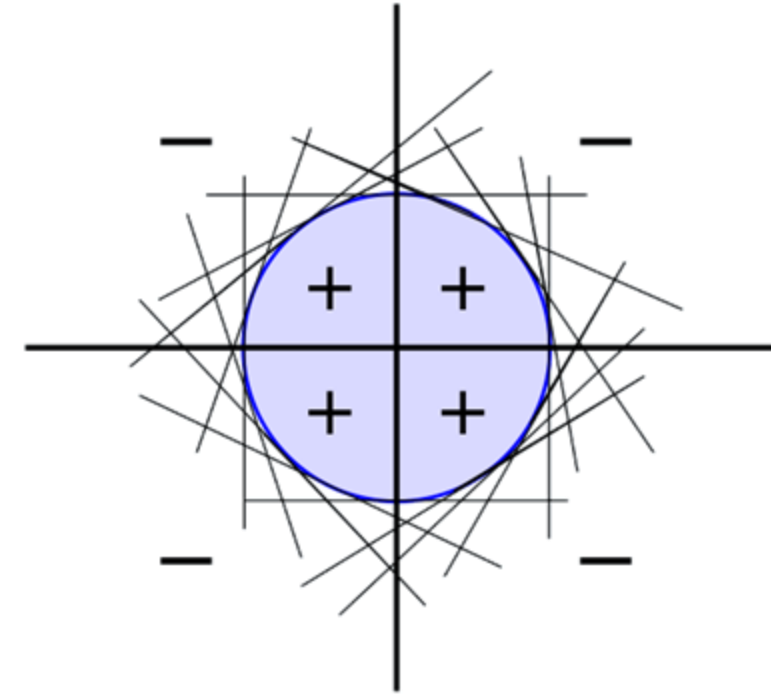
Multilayer Perceptron



Target



8 perceptrons



16 perceptrons

The more nodes/neurons, the more flexible is the model

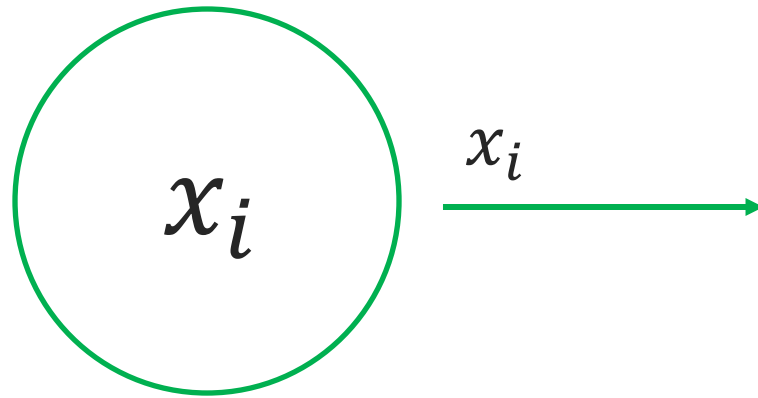
Source: Abu-Mostafa, Learning from Data, Caltech

Universal function approximation

“A **feedforward network** with a single layer is sufficient to represent **any function**, but the layer may be infeasibly large and may fail to learn and generalize correctly.”

Ian Goodfellow, Deep Learning
Creator of generative adversarial networks

Input nodes / neurons



Simply passes the input value to the next layer

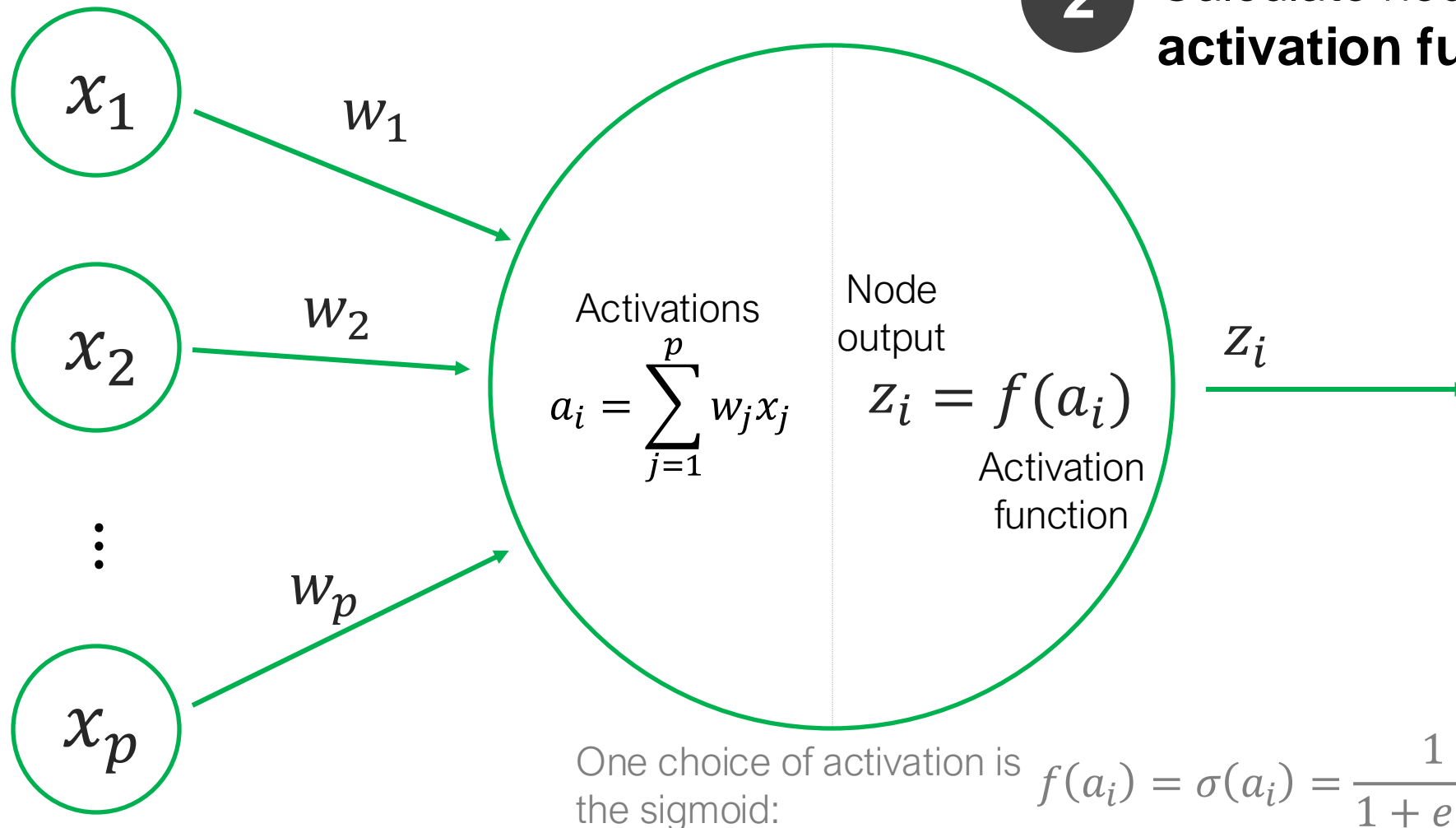
Hidden & output nodes

1

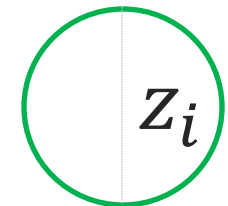
Calculate the **activations**: linear combinations of weights and the last layer's output

2

Calculate node output: apply the **activation function** to the activations

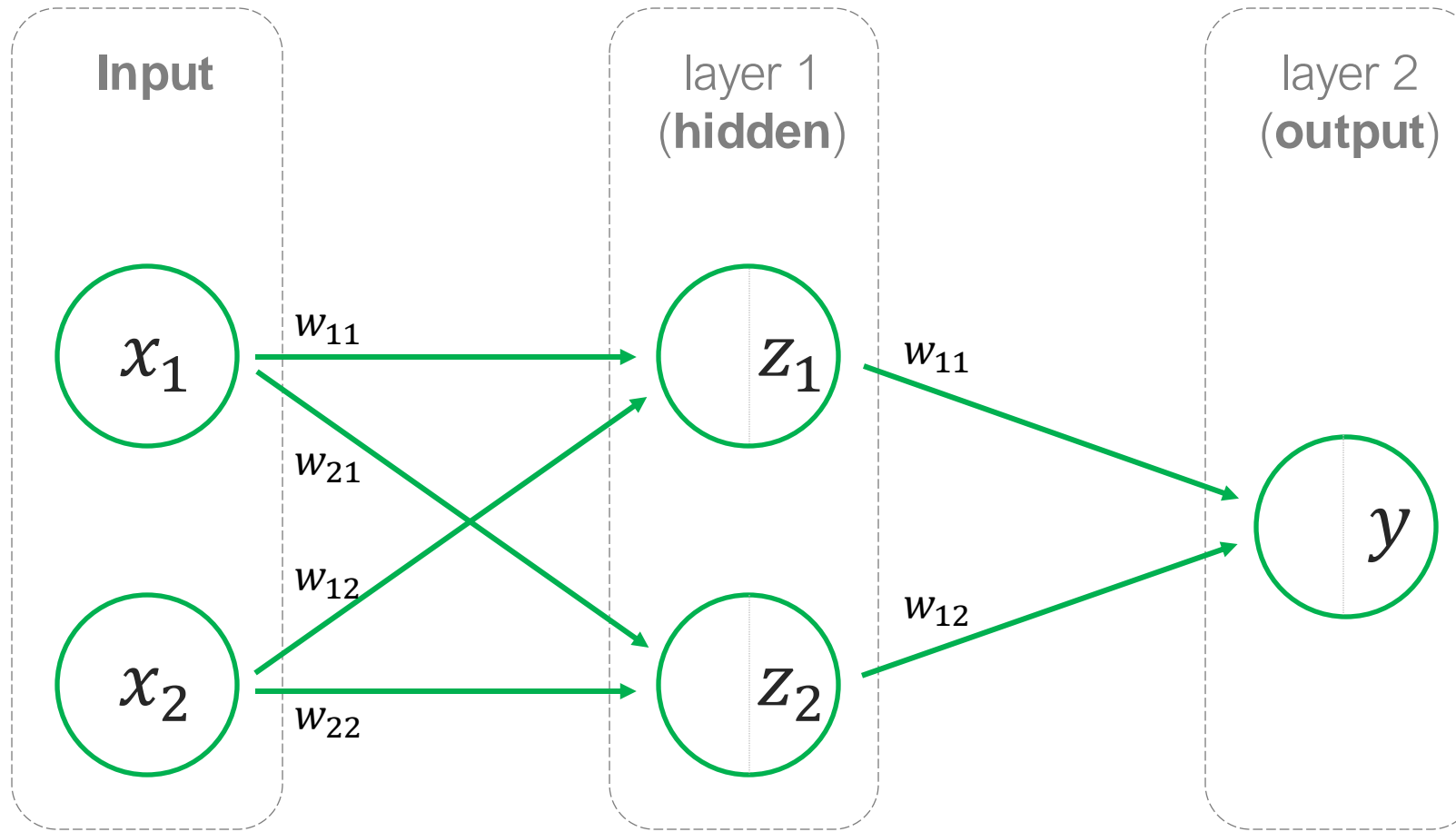


Represented as:



One choice of activation is the sigmoid: $f(a_i) = \sigma(a_i) = \frac{1}{1 + e^{-a_i}}$

Simple Neural Network



Notational shorthand:
(a more precise
alternative notation)

$$w_{ij} = w_{ij}^{(1)}$$

$$z_i = z_i^{(1)}$$

$$w_{ij} = w_{ij}^{(2)}$$

$$y = z_1^{(2)}$$

$w_{ij}^{(k)}$

- Layer k
- From node j
(in the last layer)
- to node i
(in the next layer)

Forward Propagation

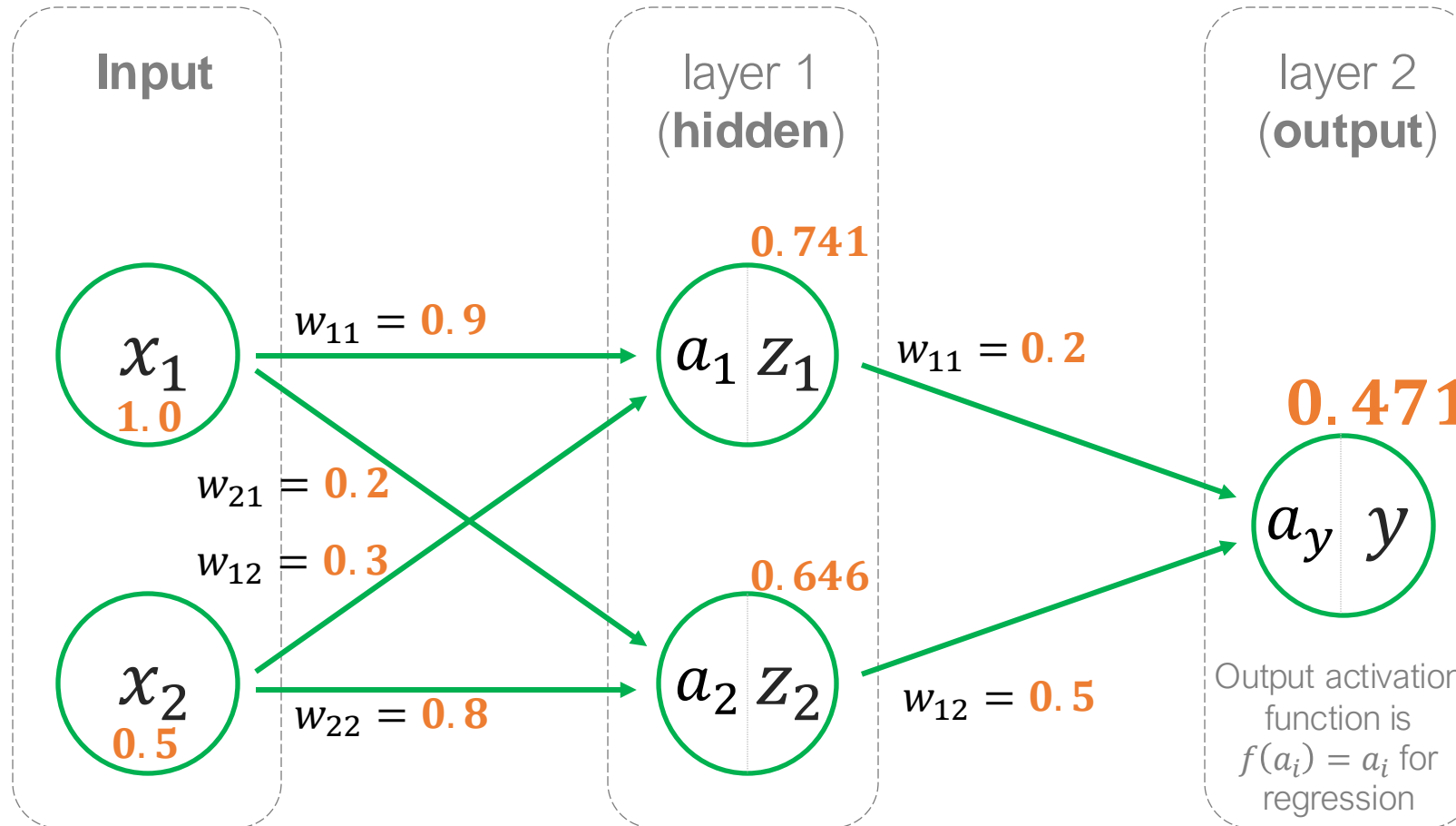
Calculating the output from input

$$a_1 = (0.9)(1.0) + (0.3)(0.5) = 1.05 \quad \text{Hidden layer calculations}$$

$$a_2 = (0.2)(1.0) + (0.8)(0.5) = 0.6$$

$$z_1 = \sigma(a_1) = \sigma(1.05) = 0.741$$

$$z_2 = \sigma(a_2) = \sigma(0.6) = 0.646$$



Output layer calculations

$$a_y = (0.2)(0.741) + (0.5)(0.646) = 0.471$$

$$y = a_y = 0.471 \quad \text{Regression}$$

Alternatively...

$$y = \sigma(a_y) = \sigma(0.471) = 0.616$$

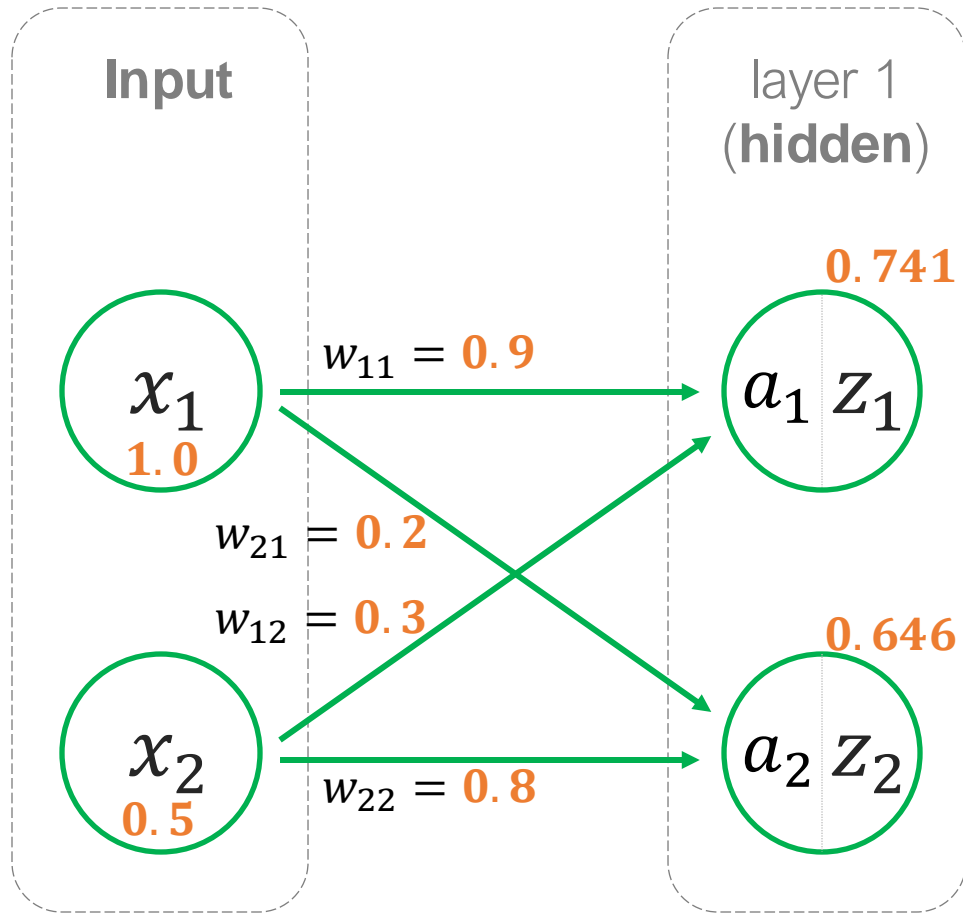
Classification

Output activation function is $f(a_i) = a_i$ for regression

$$\sigma(a_i) = \frac{1}{1 + e^{-a_i}}$$

Forward Propagation

Calculating the output from input



Hidden layer matrix calculations

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{array}{l} \longrightarrow \text{The weights INTO node } z_1 \\ \longrightarrow \text{The weights INTO node } z_2 \end{array}$$

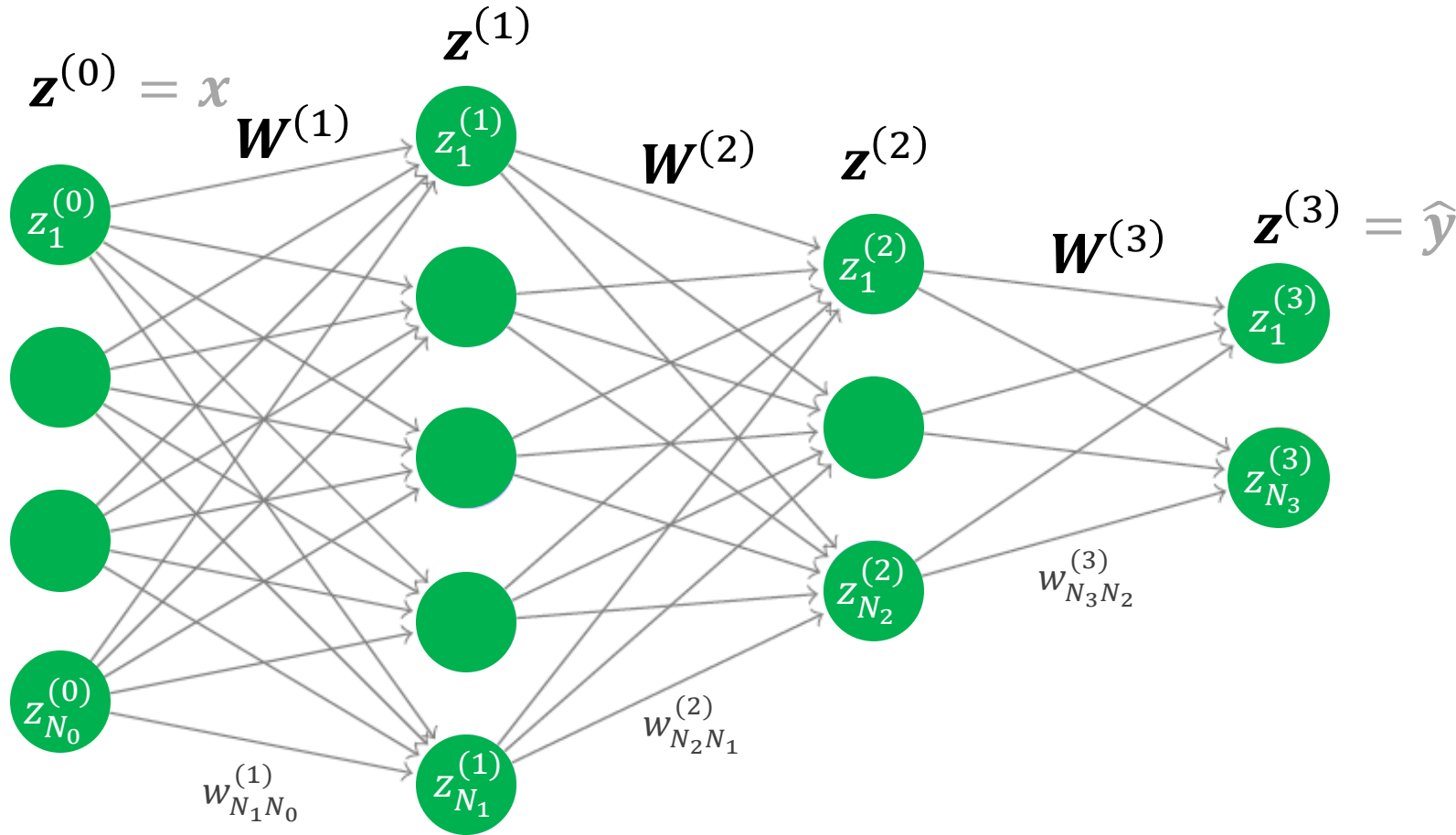
$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{bmatrix}$$

$$\mathbf{z} = \sigma(\mathbf{a}) = \begin{bmatrix} \sigma(w_{11}x_1 + w_{12}x_2) \\ \sigma(w_{21}x_1 + w_{22}x_2) \end{bmatrix}$$

Forward Propagation

Example neural network with $L = 3$ layers and the i th layer has N_i nodes



Simple steps for forward propagation:

For $i = 1$ to L :

$$\mathbf{z}^{(i)} = \sigma(\mathbf{W}^{(i)} \mathbf{z}^{(i-1)})$$

Where:

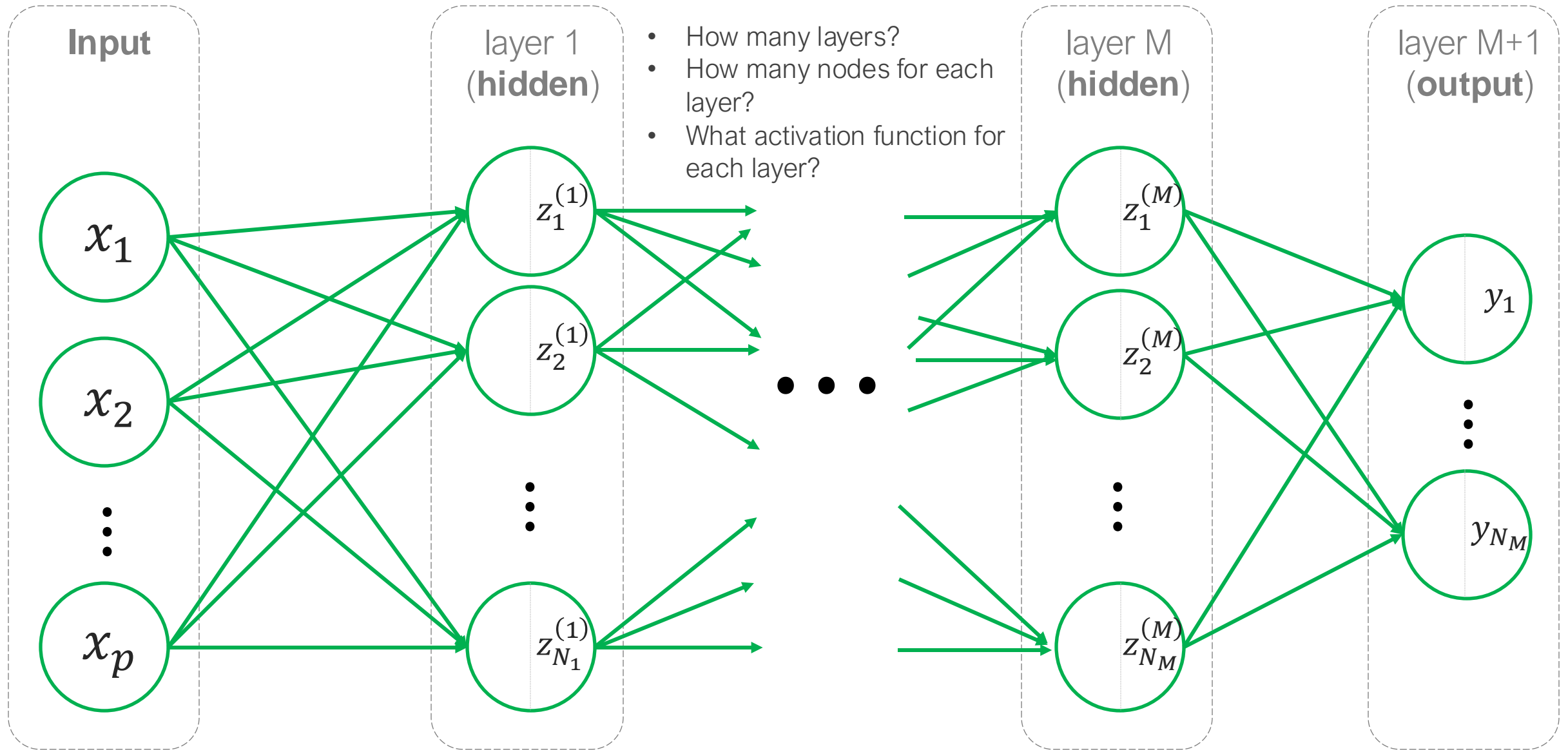
$$\mathbf{z}^{(0)} = \mathbf{x}$$

$$\hat{\mathbf{y}} = \mathbf{z}^{(L)}$$

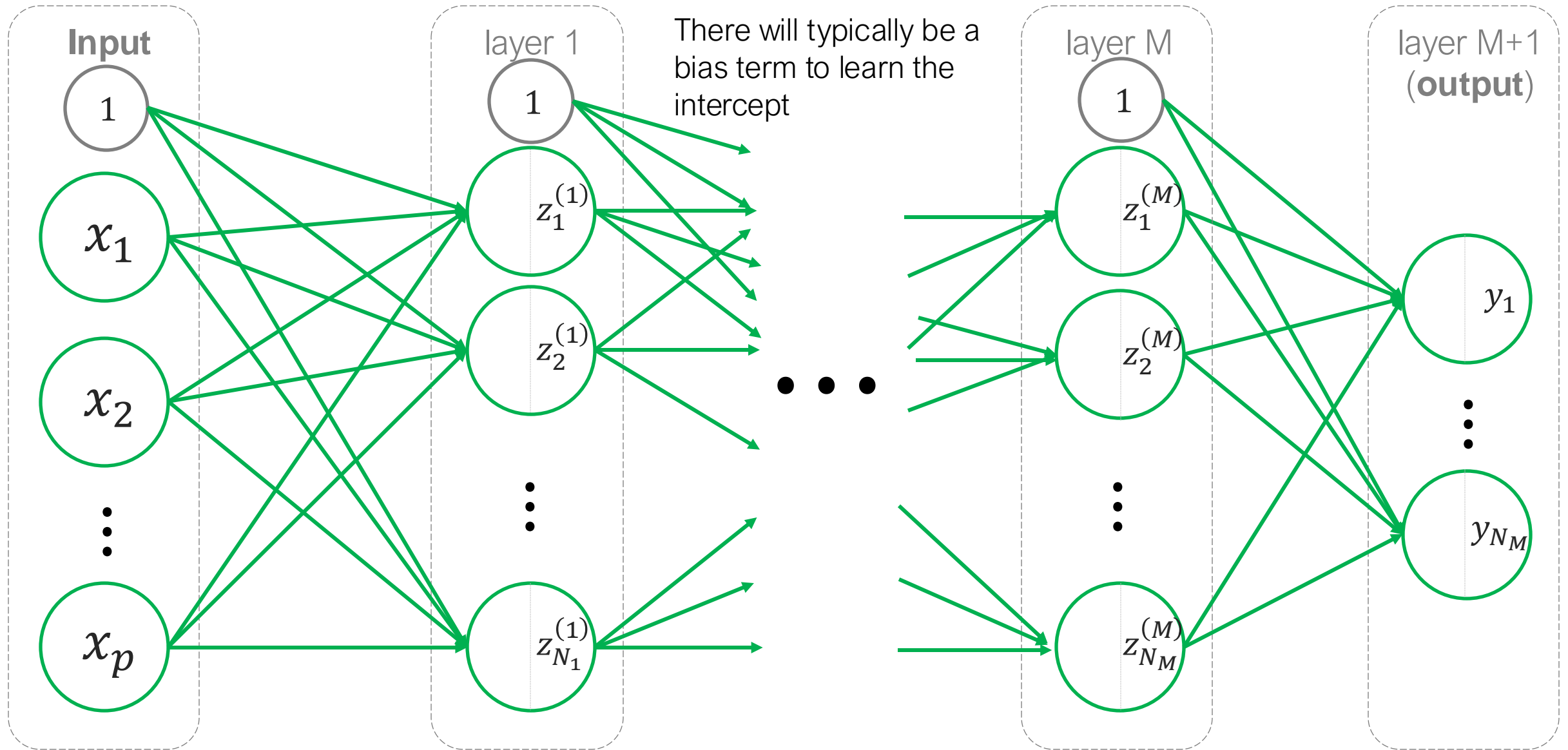
Prediction error is measured:

$$E_n = \frac{1}{2} (\hat{y}_n - y_n)^2$$

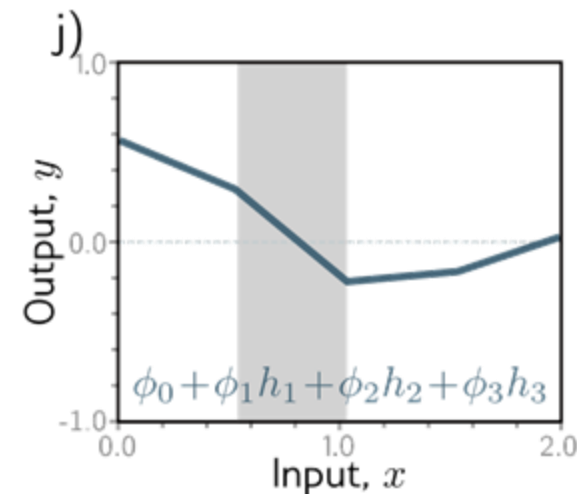
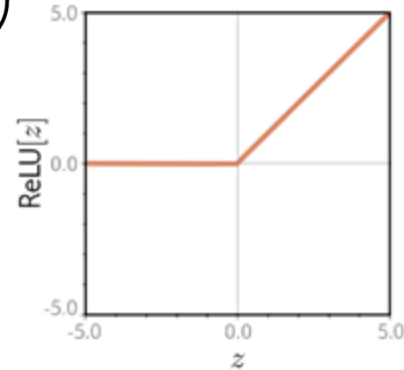
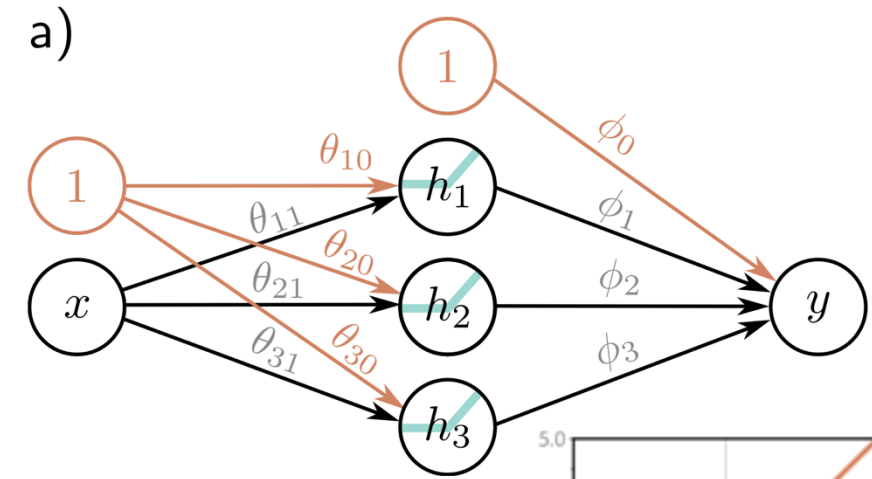
Neural networks can be customized



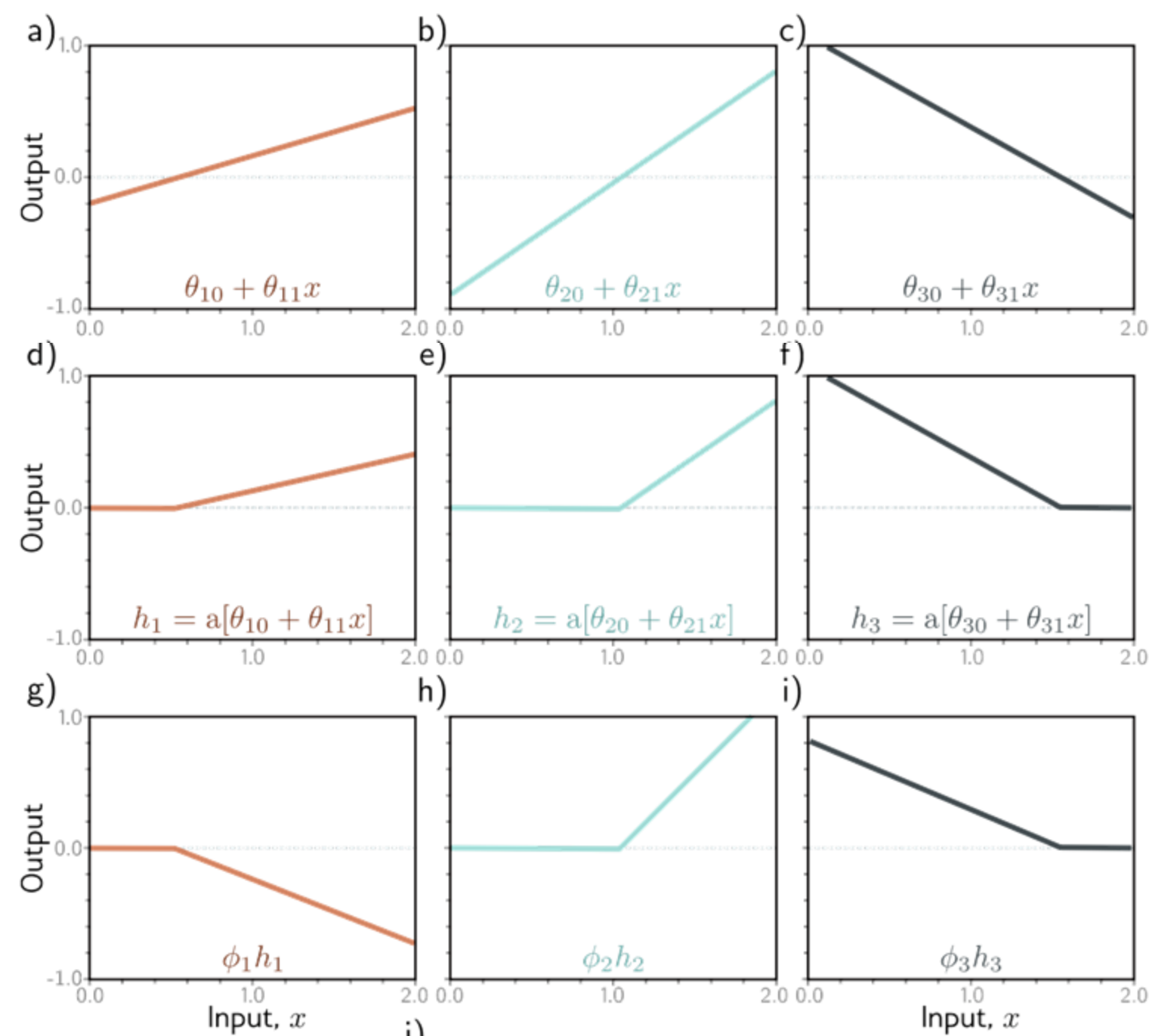
Neural networks can be customized



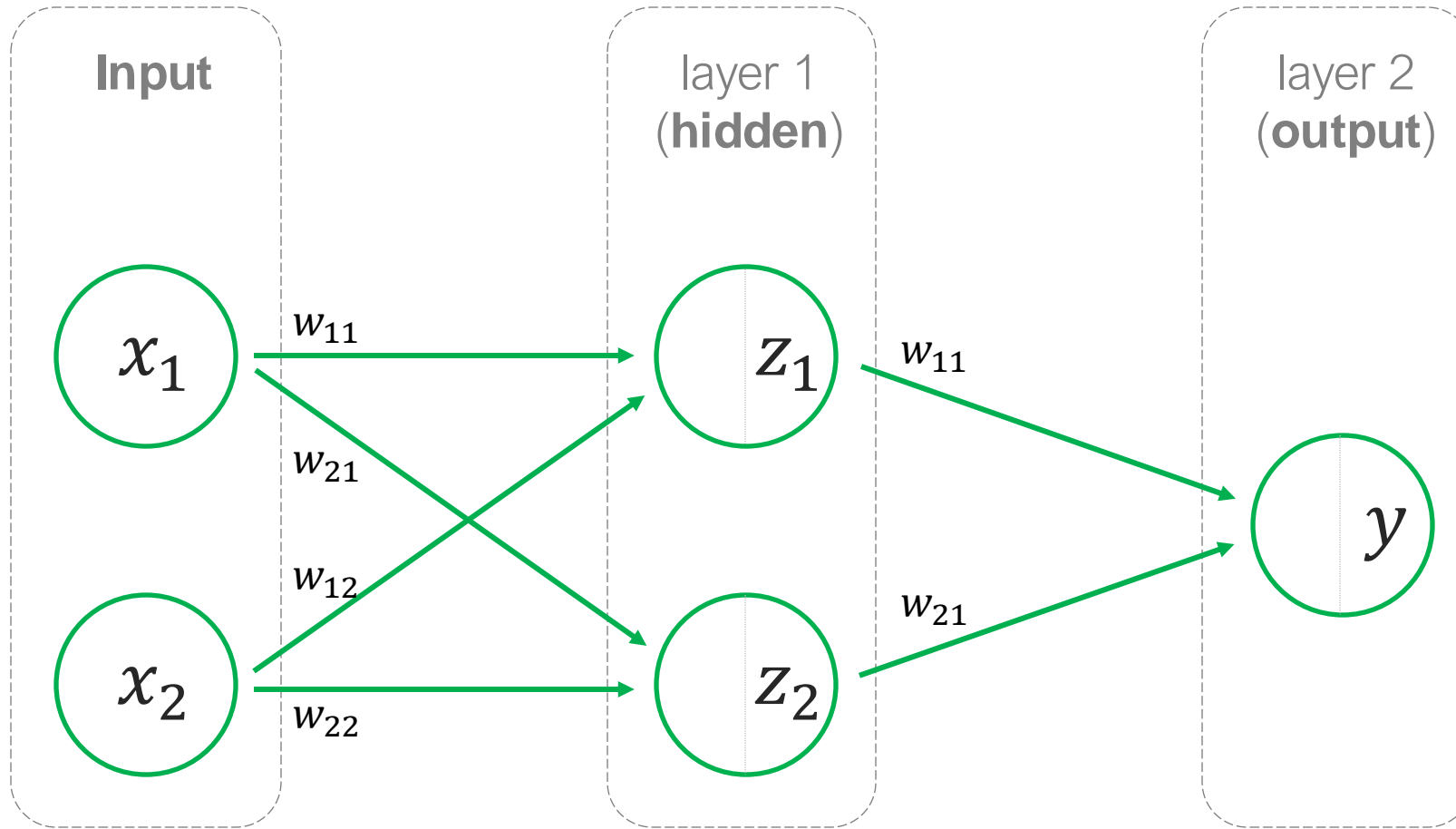
Example of simple neural network for regression



Images from Prince, Understanding Deep Learning, 2023

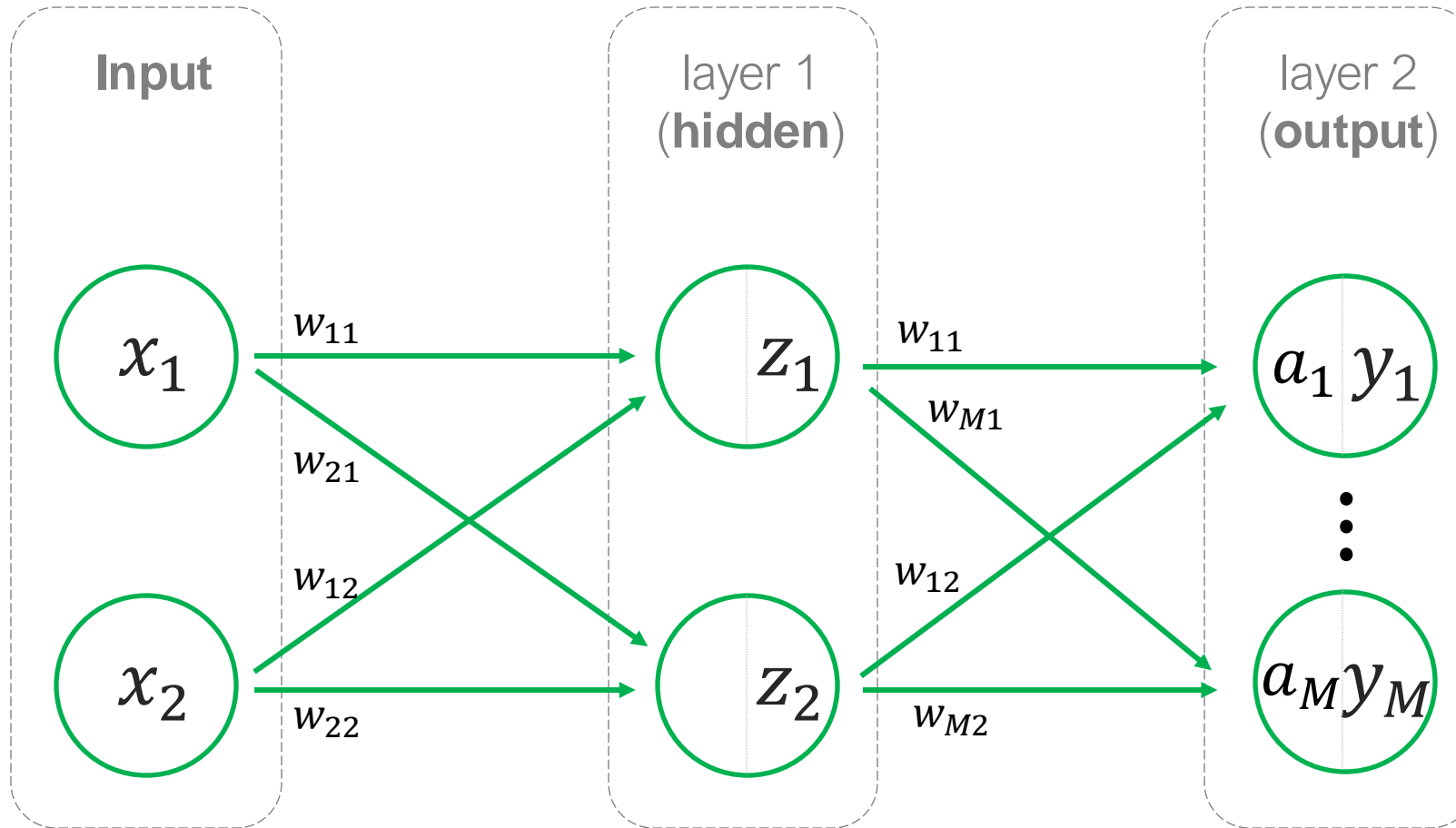


From **binary** to multiclass classification



For **binary classification** with a sigmoid activation function, the output is between zero and one, so threshold this value to assign the class

From binary to **multiclass** classification



For **multiclass problems**, we can have multiple outputs and use a softmax function:
(a generalization of the sigmoid / logistic function)

$$y_i = g(a_i) = \frac{e^{a_i}}{\sum_{n=1}^M e^{a_n}}$$

Choose the largest y value as the predicted class

Softmax

Generalization of the logistic function to multiple dimensions

To use softmax as a cost function, we use a similar expression to cross entropy loss:

$$C = - \sum_{i=1}^M y_i \log(\text{softmax}(a_i))$$

Output activations

a_i

$$\begin{bmatrix} 5.1 \\ 4.2 \\ -3.1 \\ 0.7 \end{bmatrix}$$

Activation Function

$\text{softmax}(a_i)$

$$\text{Softmax} \quad \frac{e^{a_i}}{\sum_{n=1}^M e^{a_n}}$$

Output

\hat{y}_i

$$\begin{bmatrix} 0.7046 \\ 0.2865 \\ 0.0002 \\ 0.0087 \end{bmatrix}$$

Always sums to 1
(normalizes to be a
probability distribution)

Next time...

What is a neural network and **how does it work?**

How do we **optimize model weights?**
(i.e. how do we fit our model to data)

What are the challenges of using neural networks?