LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

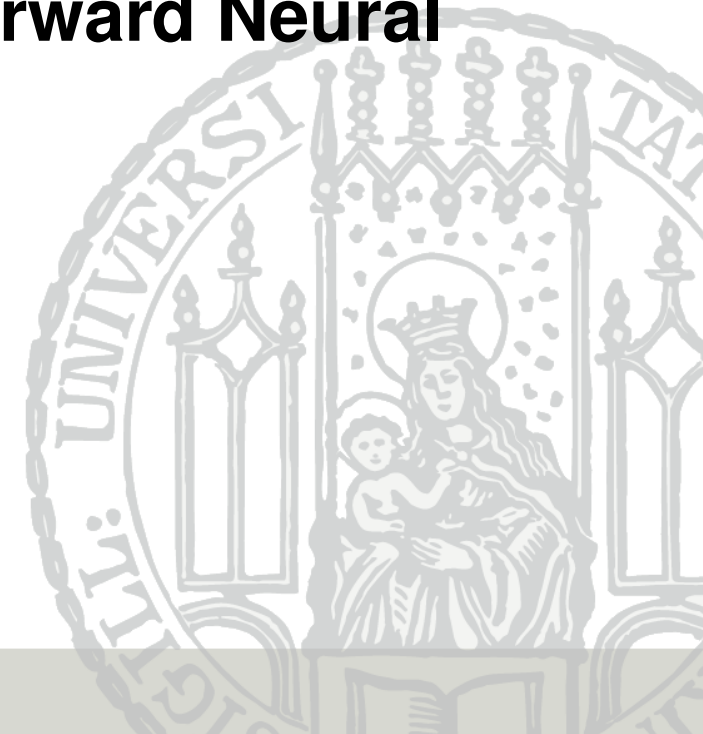
Introduction to Machine Learning

Chapter 3: Deep Learning- Feedforward Neural Networks

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2021



LECTURE OUTLINE

Single Hidden Layer Networks for Regression and Binary Classification

Multi-Layer Feedforward Neural Networks

Deep Learning

MOTIVATION

- We have a nice graphical way of representing simple functions/models like logistic regression. Why is that useful?

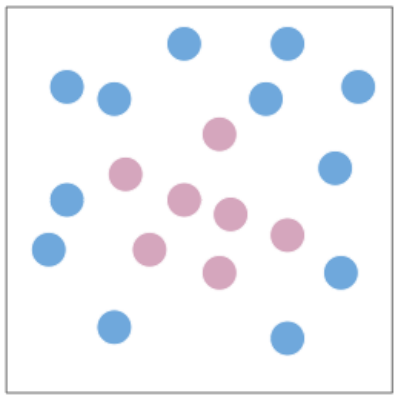
- It is useful because this visual metaphor allows us to use such individual neurons as building blocks of considerably more complicated functions.

- Therefore, networks of neurons can represent extremely complex hypothesis spaces.

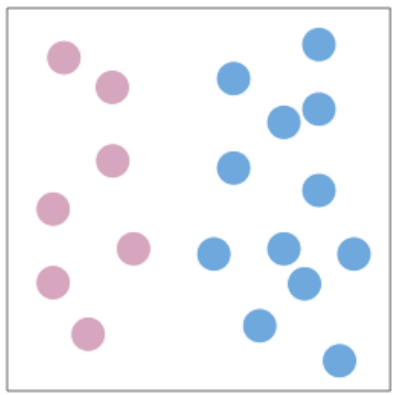
- Most importantly, it allows us to define the “right” kinds of hypothesis spaces to learn functions that are more common in our universe in a data-efficient way (see Lin, Tegmark et al. 2016).

MOTIVATION

- But why do we need more complicated functions? Is logistic regression not enough?
- Because a single neuron is restricted to learning only linear decision boundaries, its performance on the task below will be quite poor.

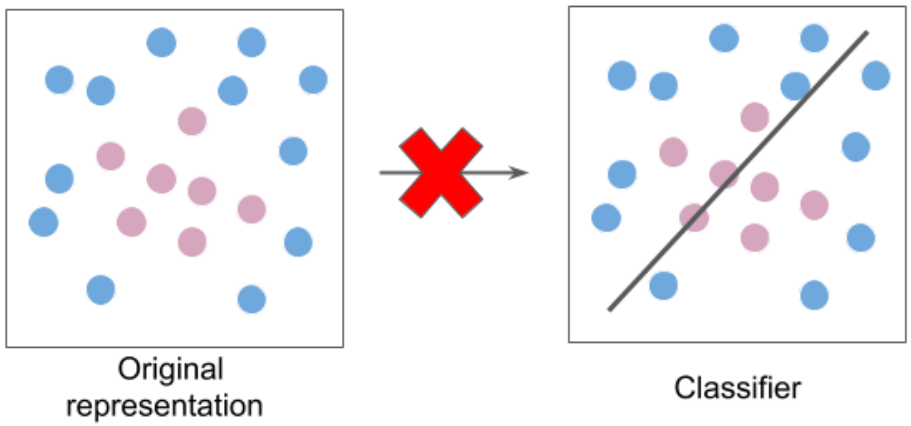


- However, if the original features are transformed (for example from Cartesian to polar coord.), the neuron can easily separate the classes.



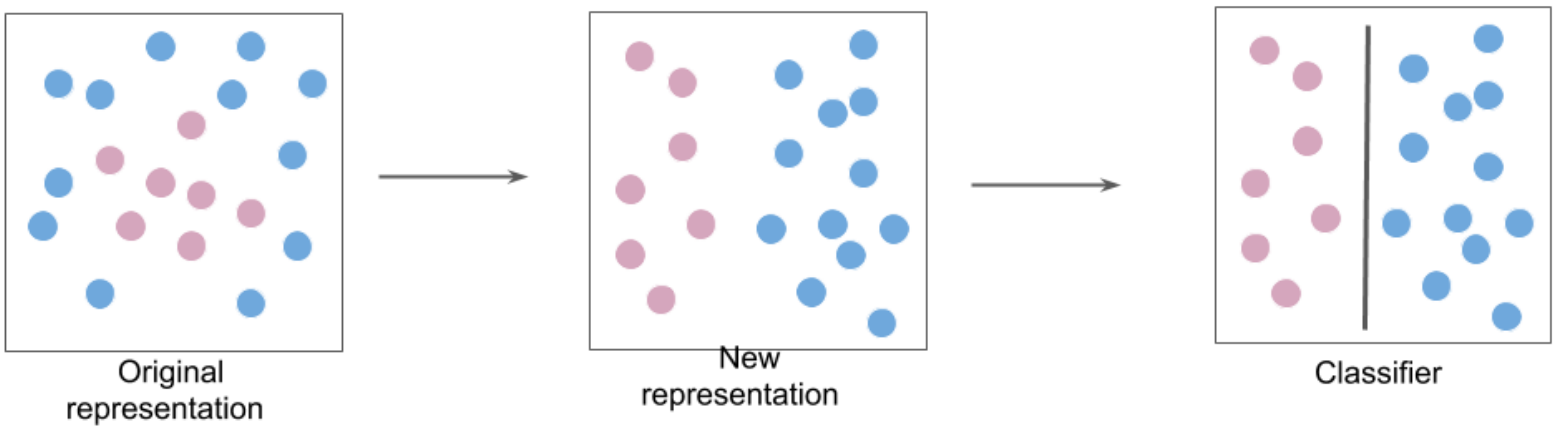
MOTIVATION

Instead of classifying the data in the original representation, ...



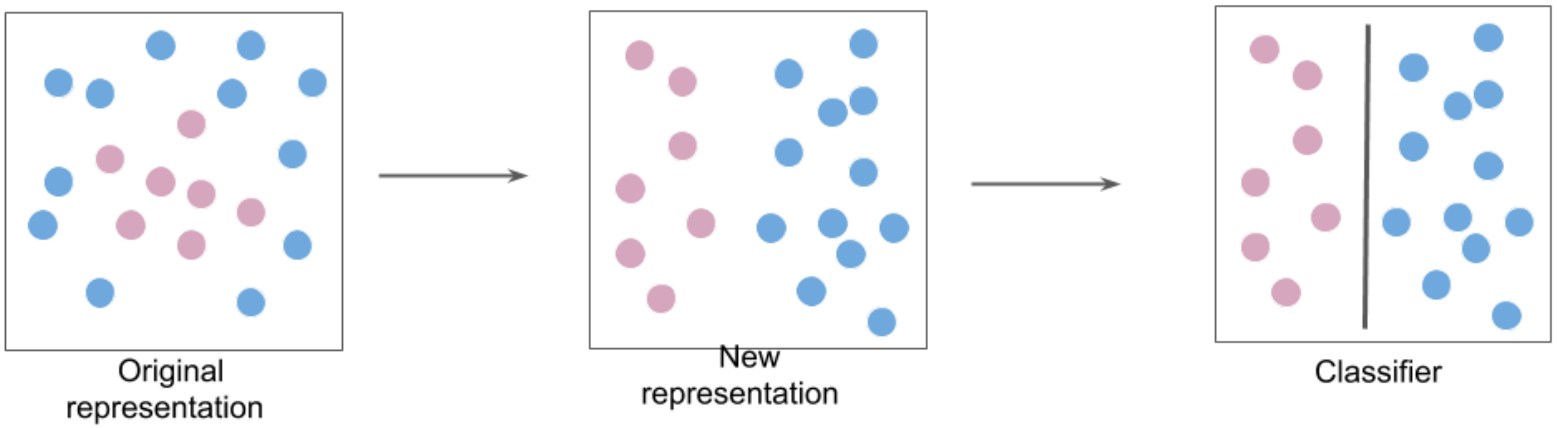
MOTIVATION

we classify it in the new feature space.

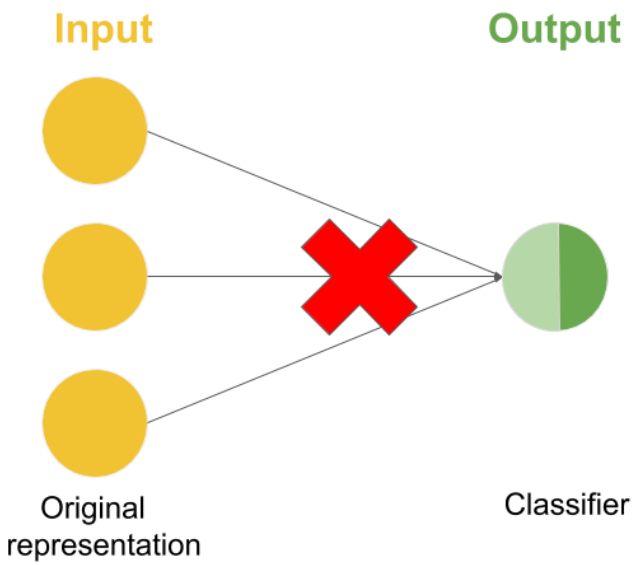


MOTIVATION

we classify it in the new feature space.

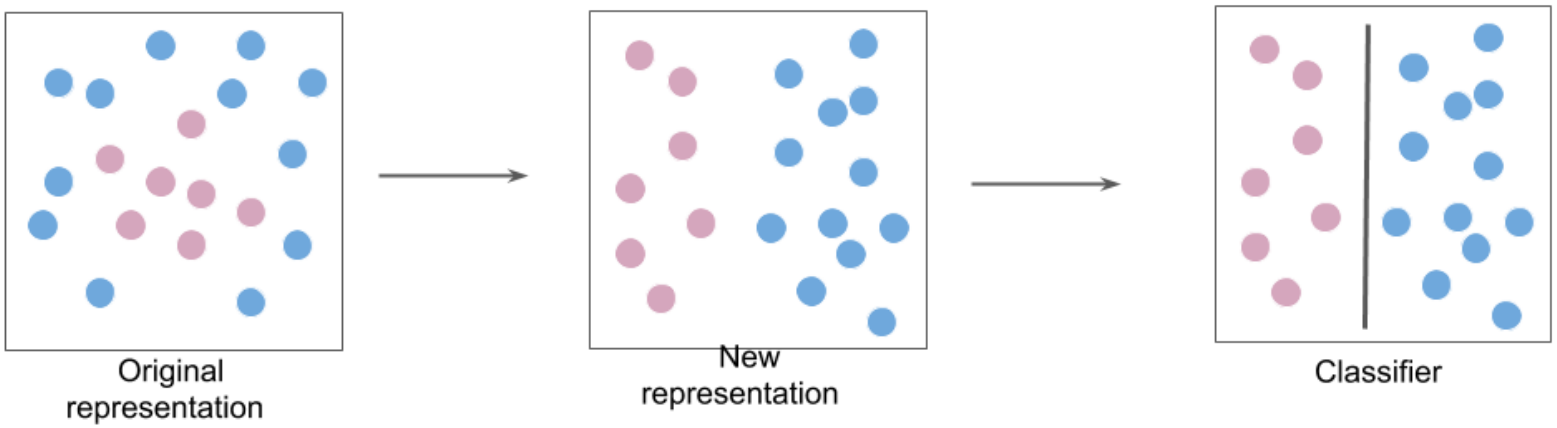


Analogously,

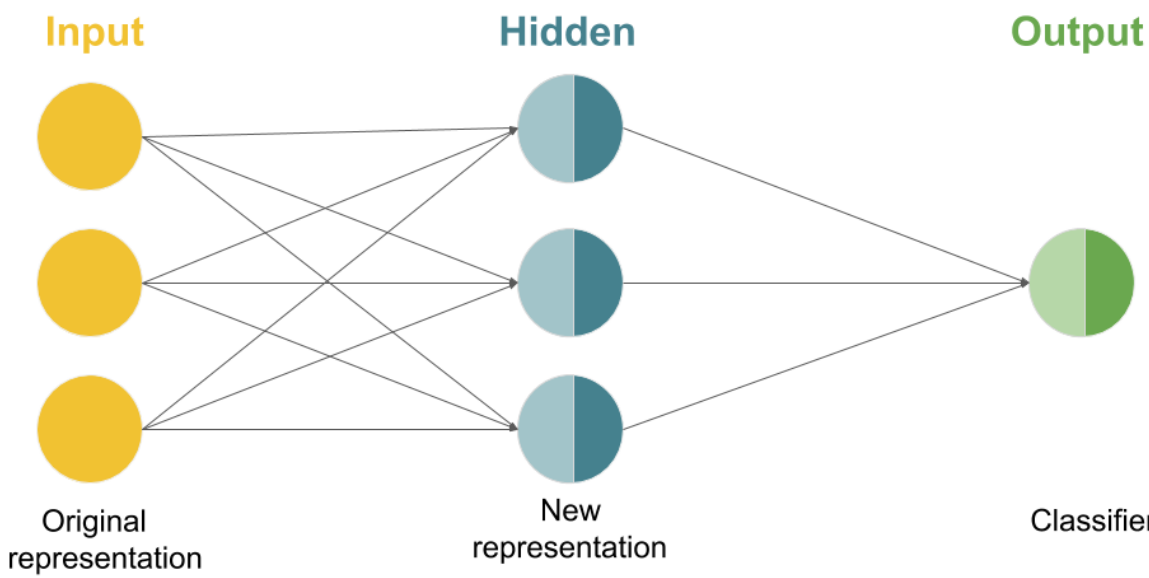


MOTIVATION

we classify it in the new feature space.



Analogously,



REPRESENTATION LEARNING

- Therefore, it is *very* critical to feed a classifier the “right” features in order for it to perform well.

- Before deep learning (DL) took off, features for tasks like machine vision and speech recognition were “hand-designed” by domain experts. This step of the machine learning pipeline is called **feature engineering**.

- The single biggest reason DL is so important is that it automates feature engineering. This is called **representation learning**.

Single Hidden Layer Networks for Regression and Binary Classification

SINGLE HIDDEN LAYER NETWORKS

- The input $\mathbf{x}$ is a column vector with dimensions $p \times 1$.
- $\mathbf{W}$ is a weight matrix with dimensions $p \times m$:

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p,1} & w_{p,2} & \cdots & w_{p,m} \end{pmatrix}$$

- For example, to obtain z_1, we pick the first column of W:

$$\mathbf{W}_1 = \begin{pmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{p,1} \end{pmatrix}$$

and compute $z_1 = \sigma(W_1^\top \mathbf{x} + b_1)$, where b_1 is the bias of the first hidden neuron and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation:

- The network has m hidden neurons $z_1, \ldots, z_m$ with

$$z_j = \sigma(\mathbf{W}_j^\top \mathbf{x} + b_j)$$

  - $z_{in,j} = \mathbf{W}_j^\top \mathbf{x} + b_j$
  - $z_{out,j} = \sigma(z_{in,j}) = \sigma(\mathbf{W}_j^\top \mathbf{x} + b_j)$

  for $j \in \{1, \ldots, m\}$.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

- Vectorized notation:
  - $\mathbf{z}_{in} = \left(z_{in,1}, \ldots, z_{in,m}\right)^\top = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$
    (Note: $\mathbf{W}^\top \mathbf{x} = \left(\mathbf{x}^\top \mathbf{W}\right)^\top$)
  - $\mathbf{z} = \mathbf{z}_{out} = \sigma\left(\mathbf{z}_{in}\right) = \sigma\left(\mathbf{W}^\top \mathbf{x} + \mathbf{b}\right)$, where the (hidden layer) activation function σ is applied element-wise to $\mathbf{z}_{in}$.
- Bias term:
  - We sometimes omit the bias term by adding a constant feature to the input $\tilde{\mathbf{x}} = (1, x_1, ..., x_p)$ and by adding the bias term to the weight matrix

$$\tilde{\mathbf{W}} = (\mathbf{b}, \mathbf{W}_1, ..., \mathbf{W}_p).$$

  - **Note**: For simplification purposes, we will not explicitly represent the bias term graphically in the following. However, the above "trick" makes it straightforward to represent it graphically.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation:

- For regression or binary classification: one output unit f where
  - $f_{in} = \mathbf{u}^\top \mathbf{z} + c$, i.e. a linear combination of derived features plus the bias term c of the output neuron, and
  - $f(\mathbf{x}) = f_{out} = \tau(f_{in}) = \tau(\mathbf{u}^\top \mathbf{z} + c)$, where τ is the output activation function.
- For regression τ is the identity function.
- For binary classification, τ is a sigmoid function.
- **Note**: The purpose of the hidden-layer activation function σ is to introduce non-linearities so that the network is able to learn complex functions whereas the purpose of τ is merely to get the final score on the same scale as the target.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

General notation: Multiple inputs

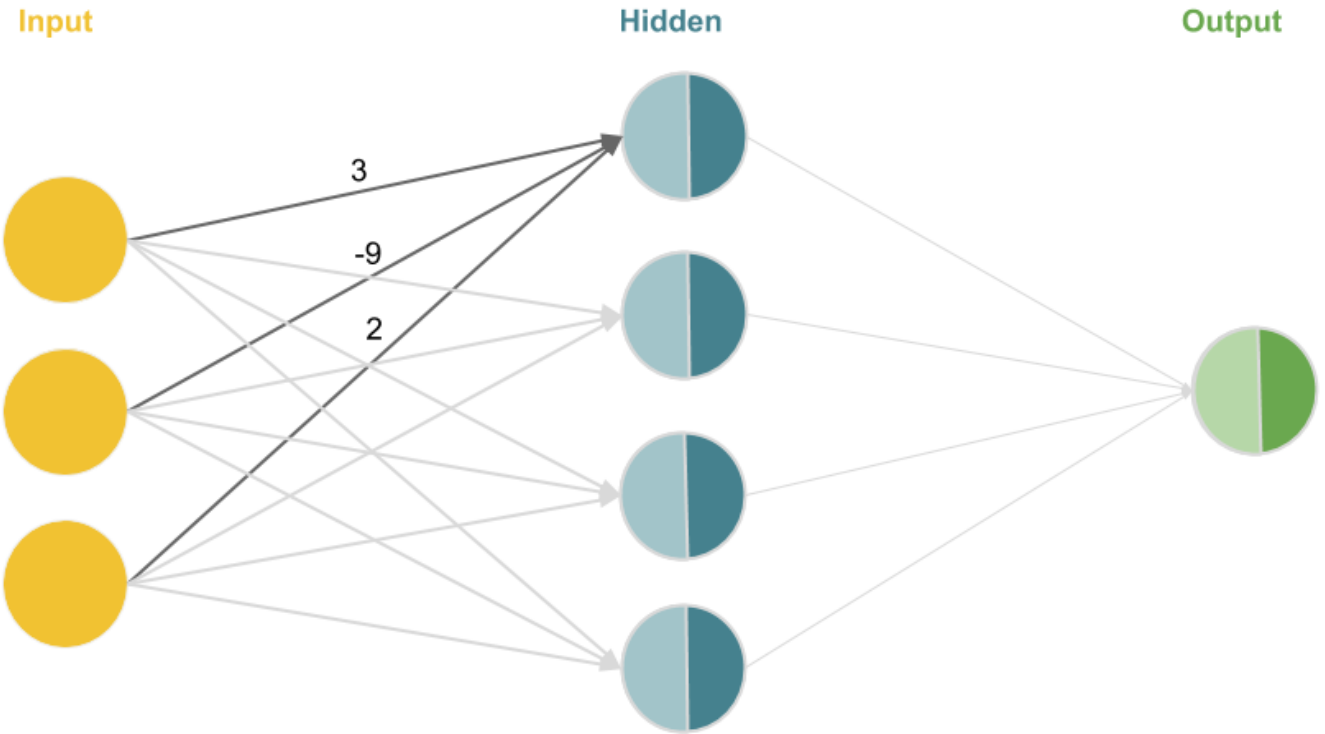
- It is possible to feed multiple inputs to a neural network simultaneously.
- The inputs $\mathbf{x}^{(i)}$, for $i \in \{1, \ldots, n\}$, are arranged as rows in the **design matrix X**.
  - **X** is a $(n \times p)$-matrix.
- The weighted sum in the hidden layer is now computed as $\mathbf{XW} + \boldsymbol{B}$, where,
  - **W**, as usual, is a $(p \times m)$ matrix, and,
  - $\boldsymbol{B}$ is a $(n \times m)$ matrix containing the bias vector **b** (duplicated) as the rows of the matrix.
- The *matrix* of hidden activations $\boldsymbol{Z} = \sigma(\mathbf{XW} + \boldsymbol{B})$
  - $\boldsymbol{Z}$ is a $(n \times m)$ matrix.

SINGLE HIDDEN LAYER NETWORKS: NOTATION

- The final output of the network, which contains a prediction for each input, is $\tau(\boldsymbol{Z}\mathbf{u} + \boldsymbol{C})$, where
  - $\mathbf{u}$ is the vector of weights of the output neuron, and,
  - $\boldsymbol{C}$ is a $(n \times 1)$ matrix whose elements are the (scalar) bias c of the output neuron.

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \end{pmatrix} \quad \begin{pmatrix} 5 \end{pmatrix}$$

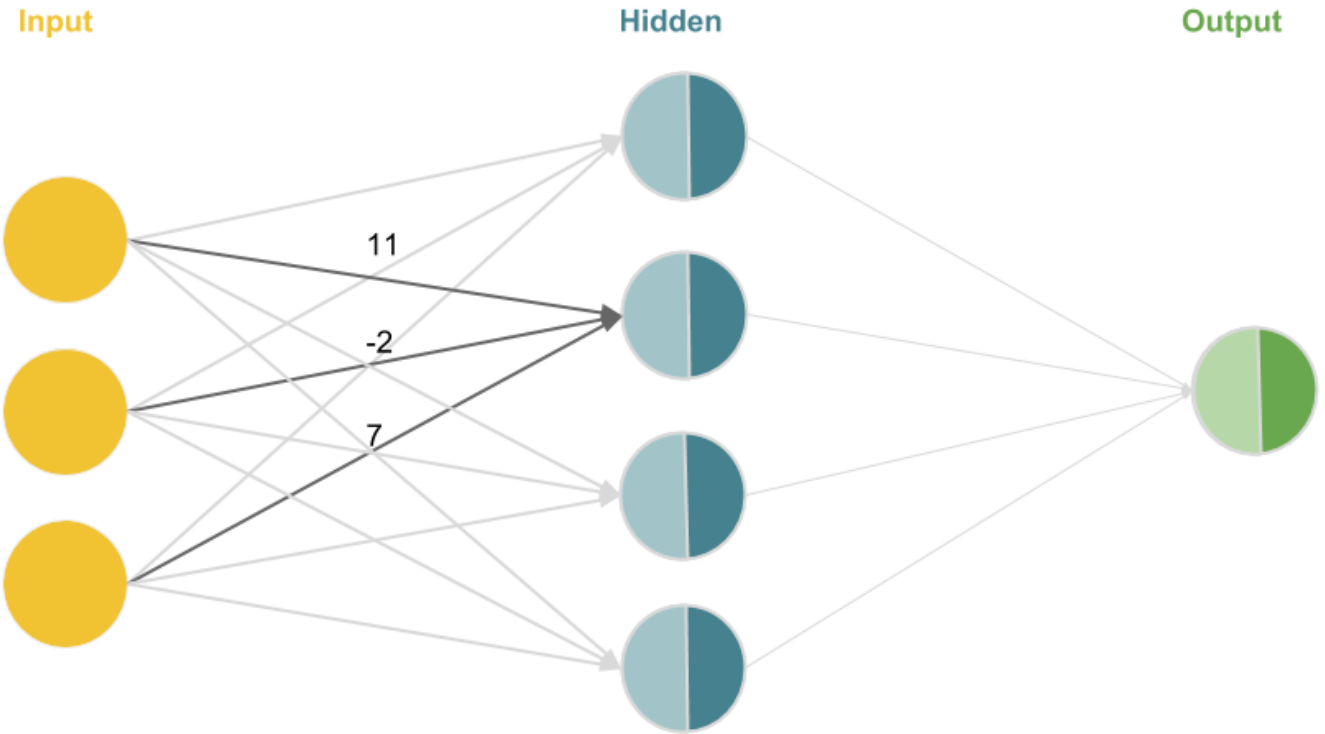
W^T $\mathbf{b}$

$$\begin{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \end{pmatrix}$$

$\mathbf{u}^T$ c

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \\ \mathbf{11} & \mathbf{-2} & \mathbf{7} \end{pmatrix} \begin{pmatrix} 5 \\ \mathbf{2} \end{pmatrix}$$

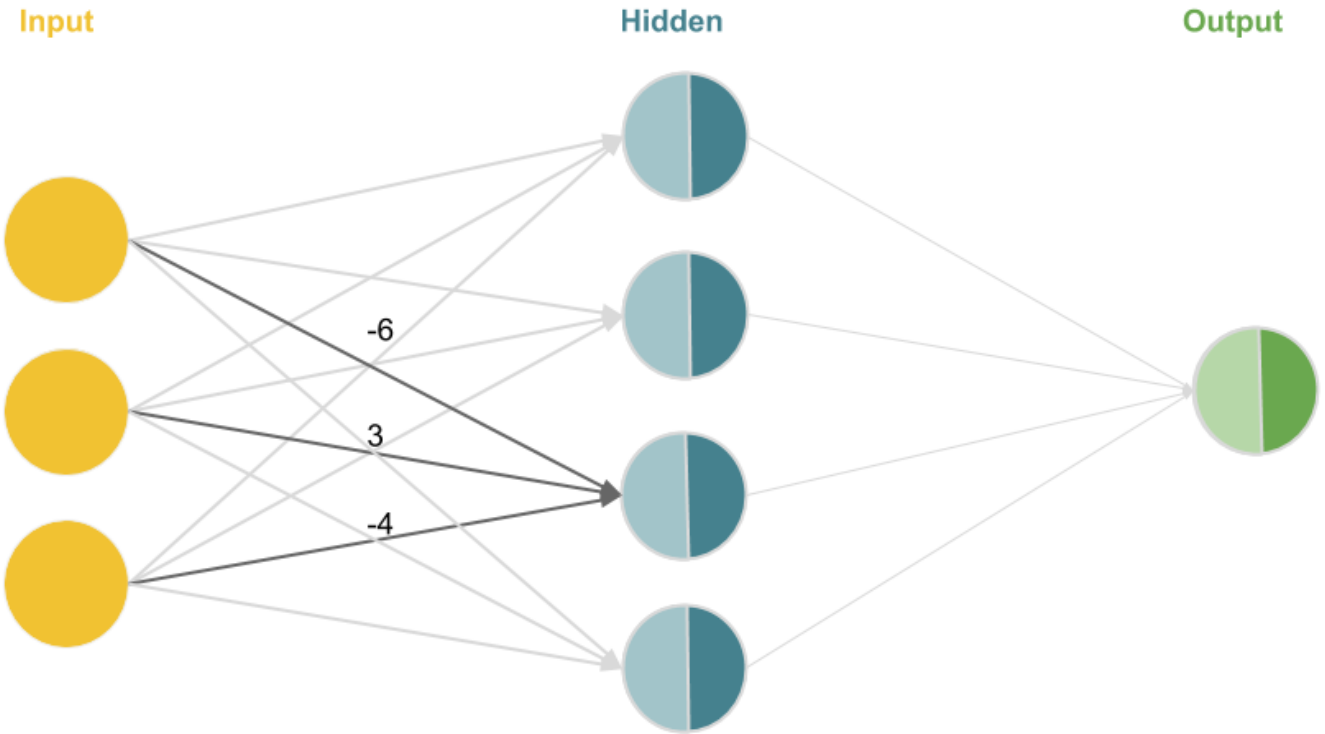
W^T $\mathbf{b}$

$$\begin{pmatrix} & & \end{pmatrix} \begin{pmatrix} \end{pmatrix}$$

$\mathbf{u}^T$ c

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ \mathbf{-6} & \mathbf{3} & \mathbf{-4} \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ \mathbf{-1} \end{pmatrix}$$

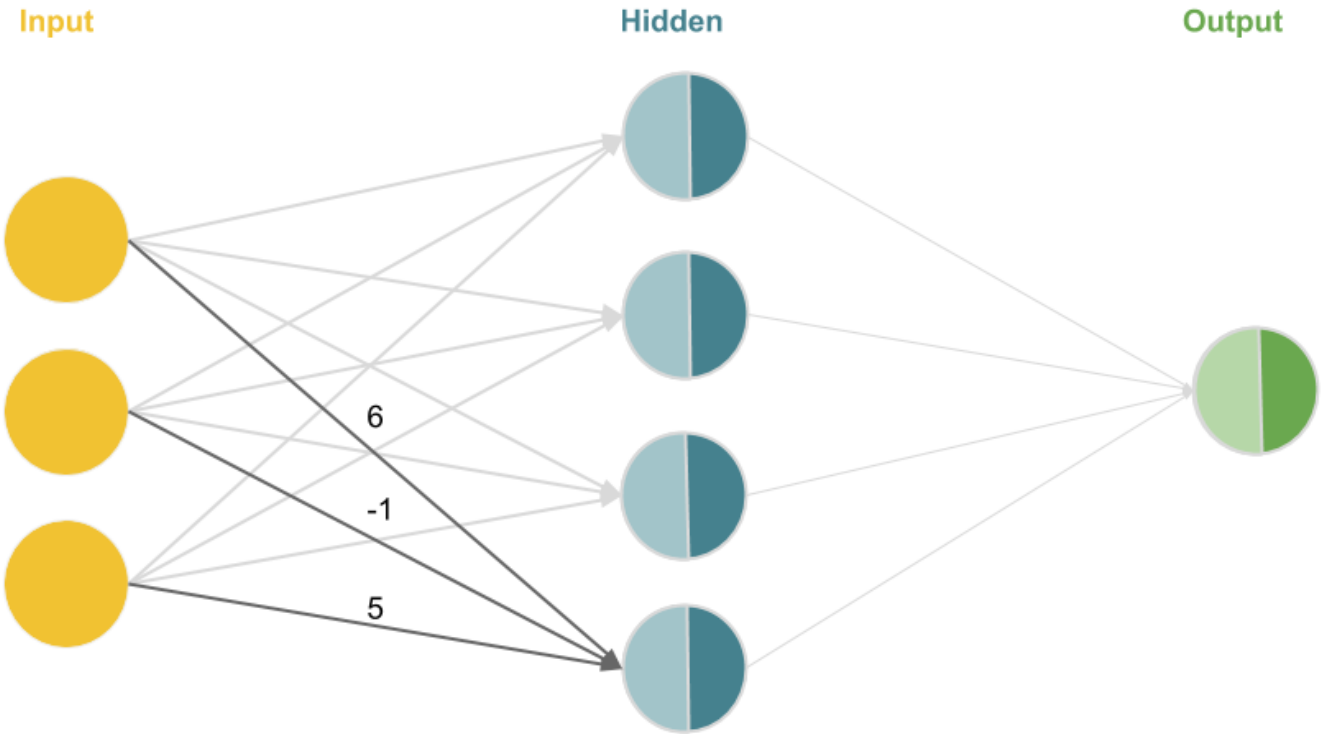
W^T $\mathbf{b}$

$$\begin{pmatrix} \quad \end{pmatrix} \begin{pmatrix} \ \end{pmatrix}$$

$\mathbf{u}^T$ c

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ \mathbf{6} & \mathbf{-1} & \mathbf{5} \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -1 \\ \mathbf{1} \end{pmatrix}$$

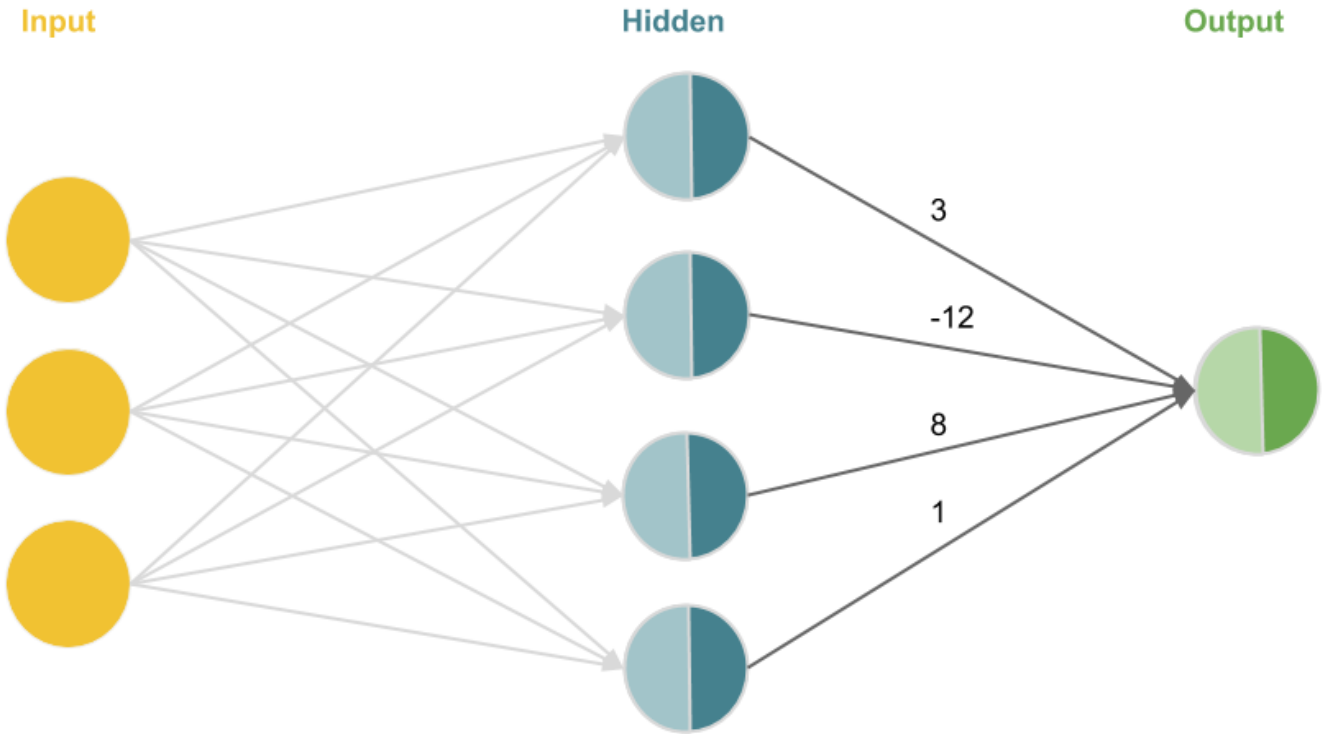
$W^T \quad \mathbf{b}$

$$\begin{pmatrix} \quad \end{pmatrix} \begin{pmatrix} \ \end{pmatrix}$$

$\mathbf{u}^T \quad c$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}$$

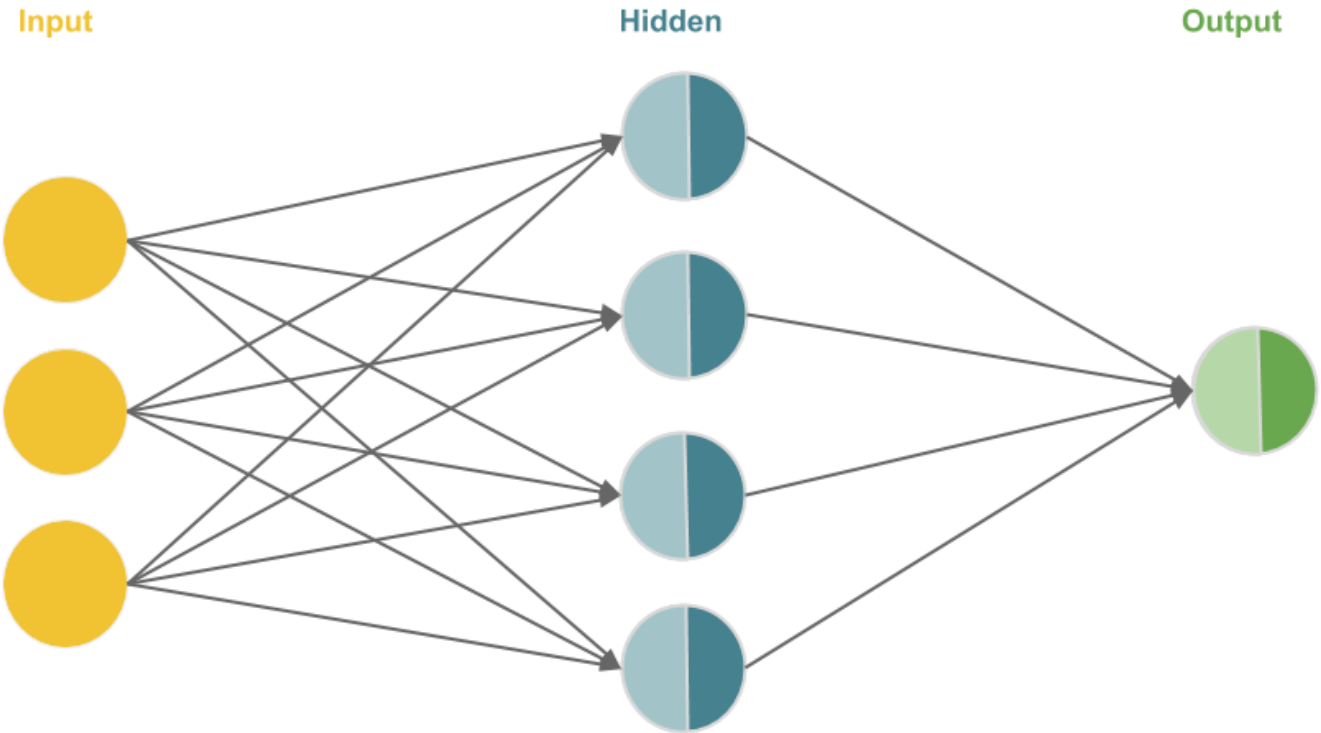
$W^T \quad \mathbf{b}$

$$\begin{pmatrix} 3 & -12 & 8 & 1 \end{pmatrix} \begin{pmatrix} 6 \end{pmatrix}$$

$\mathbf{u}^T \quad c$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

- Weights (and biases) of the network.



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}$$

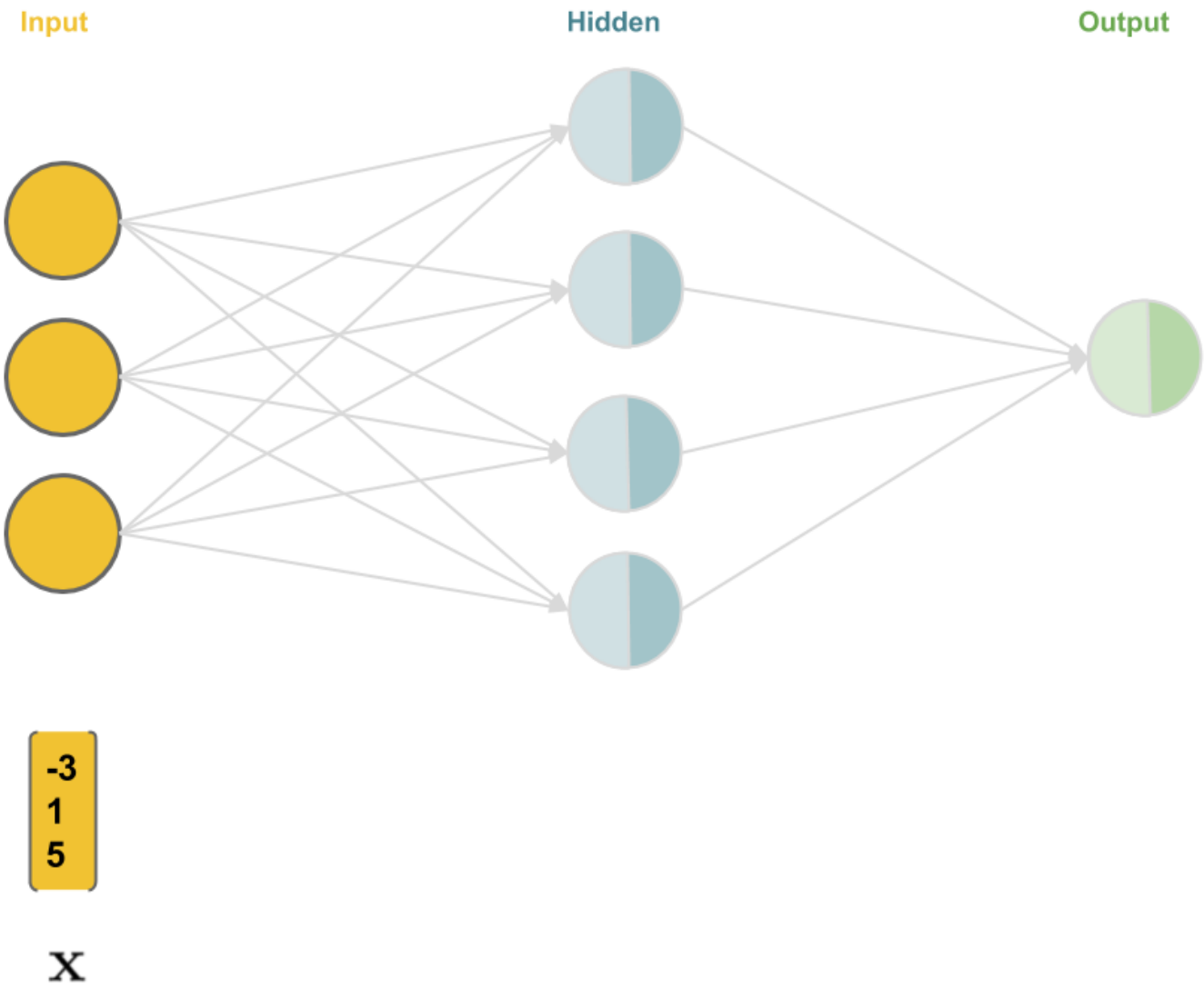
W^T $\mathbf{b}$

$$\begin{pmatrix} 3 & -12 & 8 & 1 \end{pmatrix} \begin{pmatrix} 6 \end{pmatrix}$$

$\mathbf{u}^T$ c

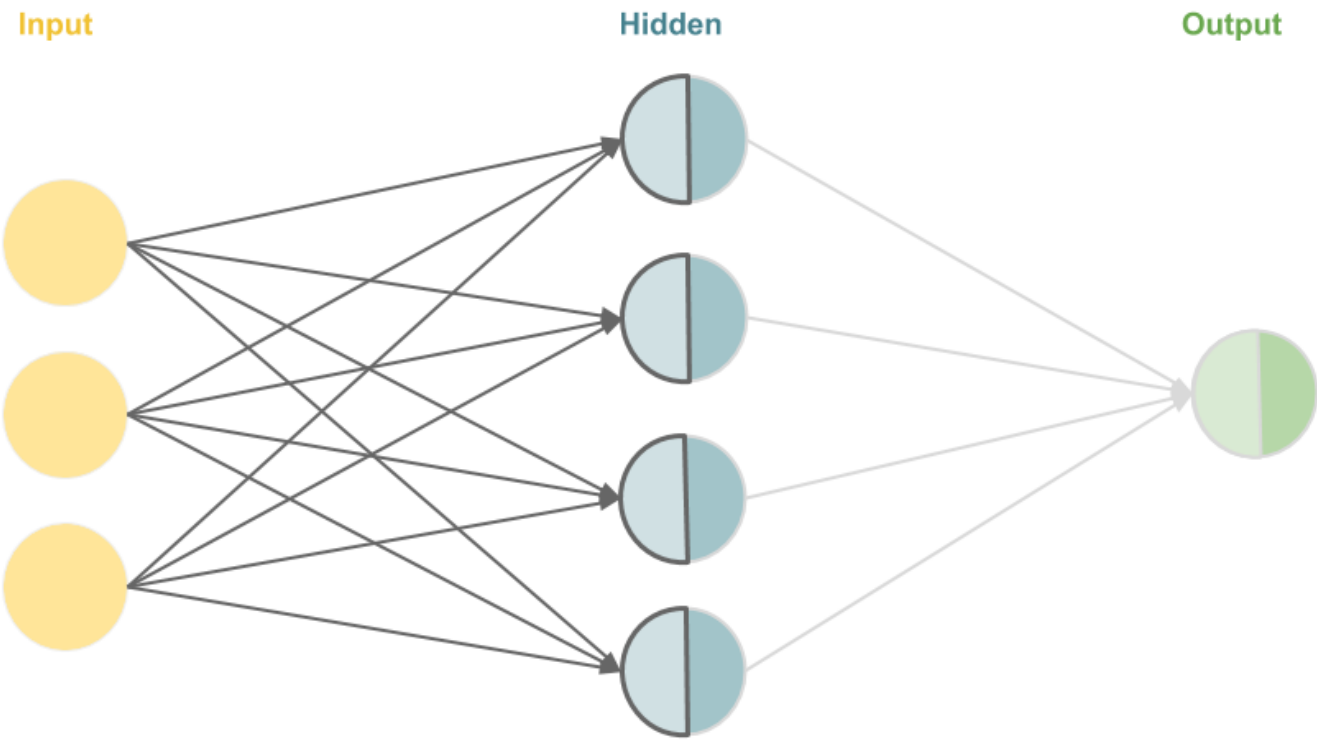
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



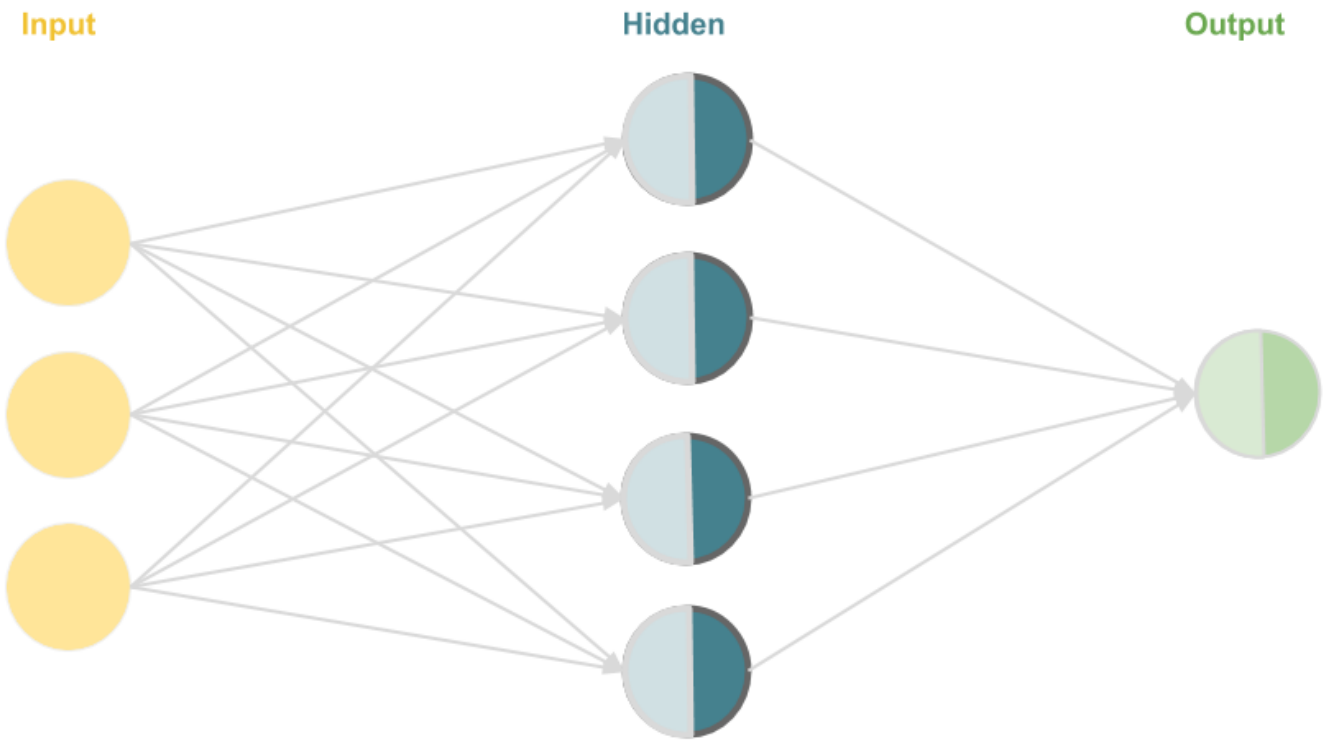
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



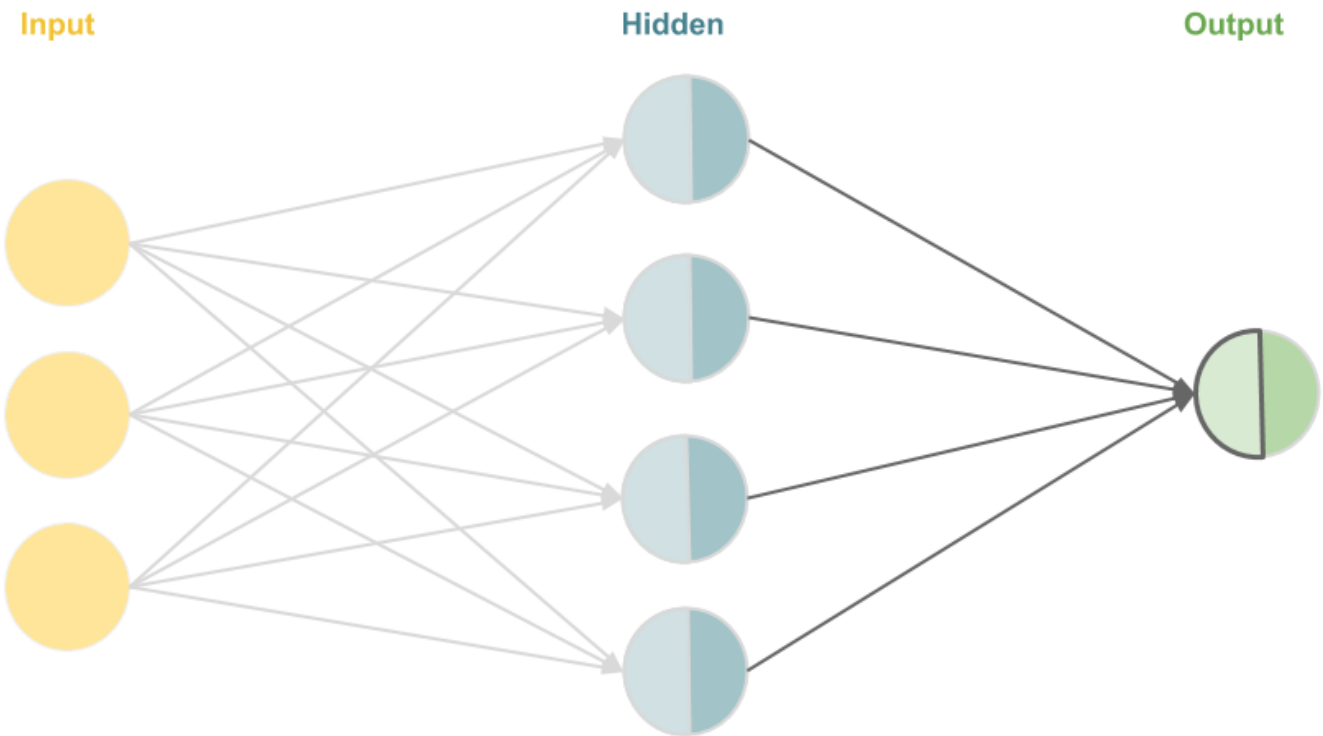
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



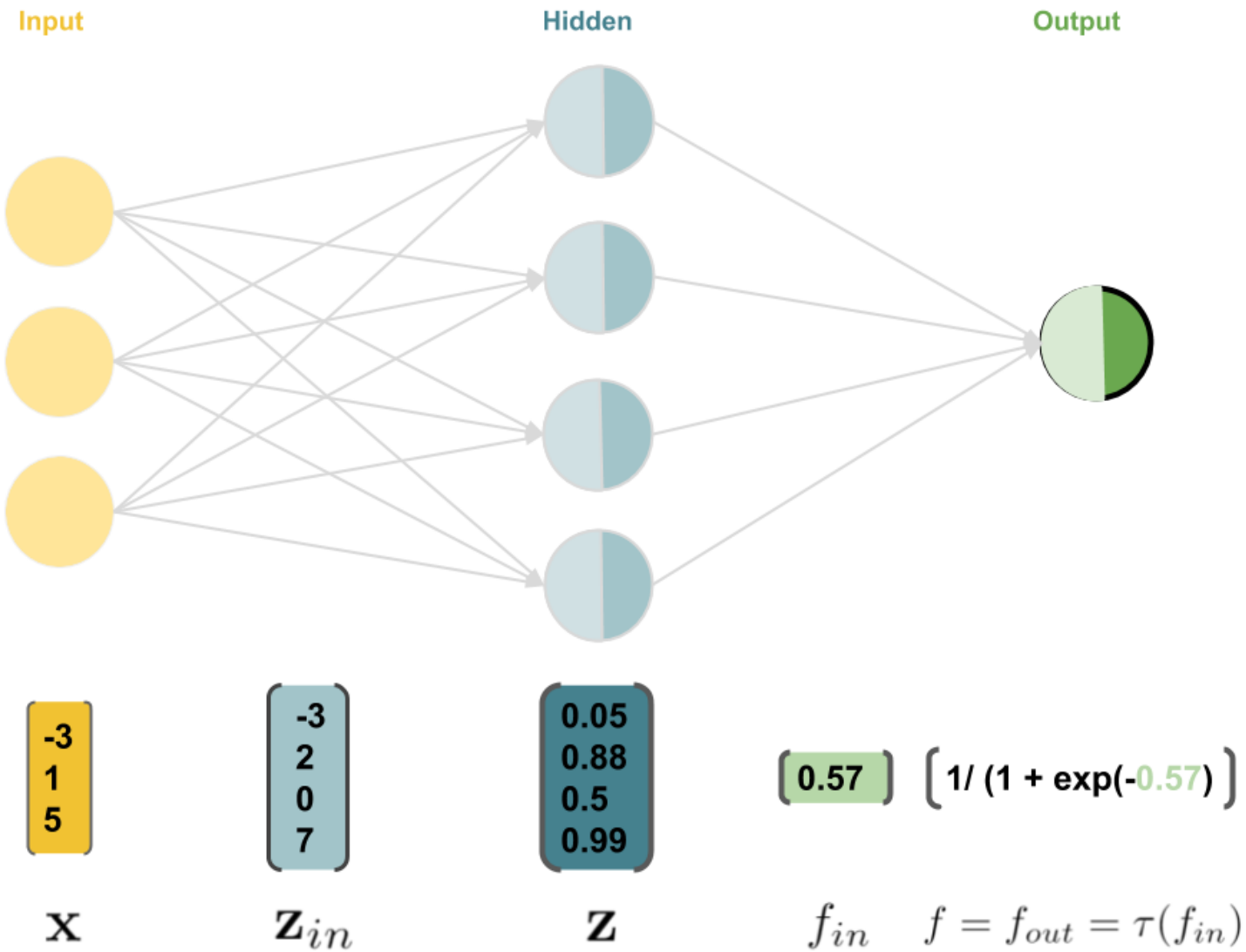
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



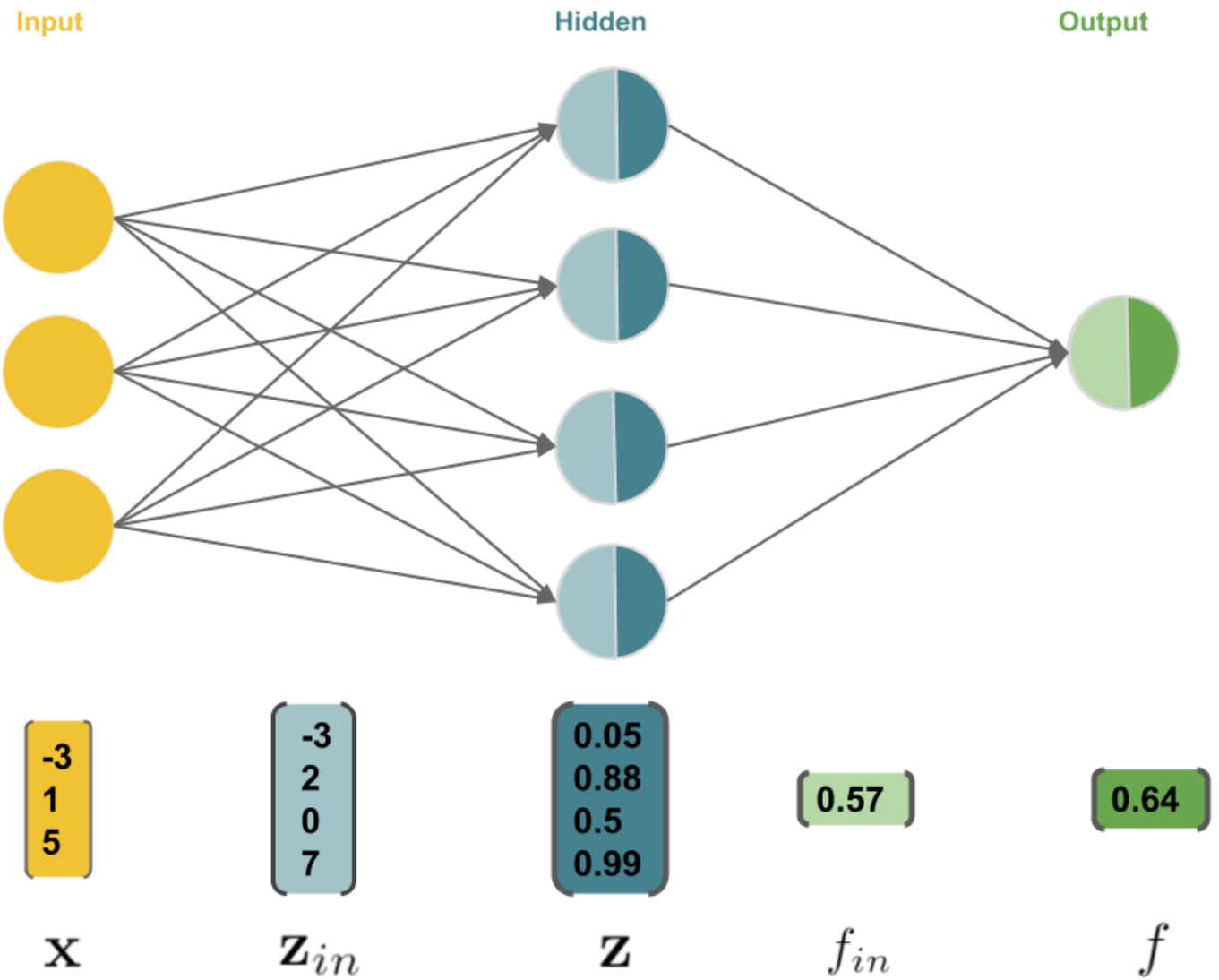
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Forward pass through the shallow neural network.



HIDDEN LAYER: ACTIVATION FUNCTION

- It is important to note that if the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.
- For simplification purposes, we drop the bias terms in notation and let $\sigma = \text{id}$. Then:

$$\begin{aligned} f(\mathbf{x}) &= \tau(\mathbf{u}^\top \mathbf{z}) = \tau(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x})) \\ &= \tau(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x})) \\ &= \tau(\mathbf{u}^\top \mathbf{W}^\top \mathbf{x}) = \tau(\mathbf{v}^\top \mathbf{x}) \end{aligned}$$

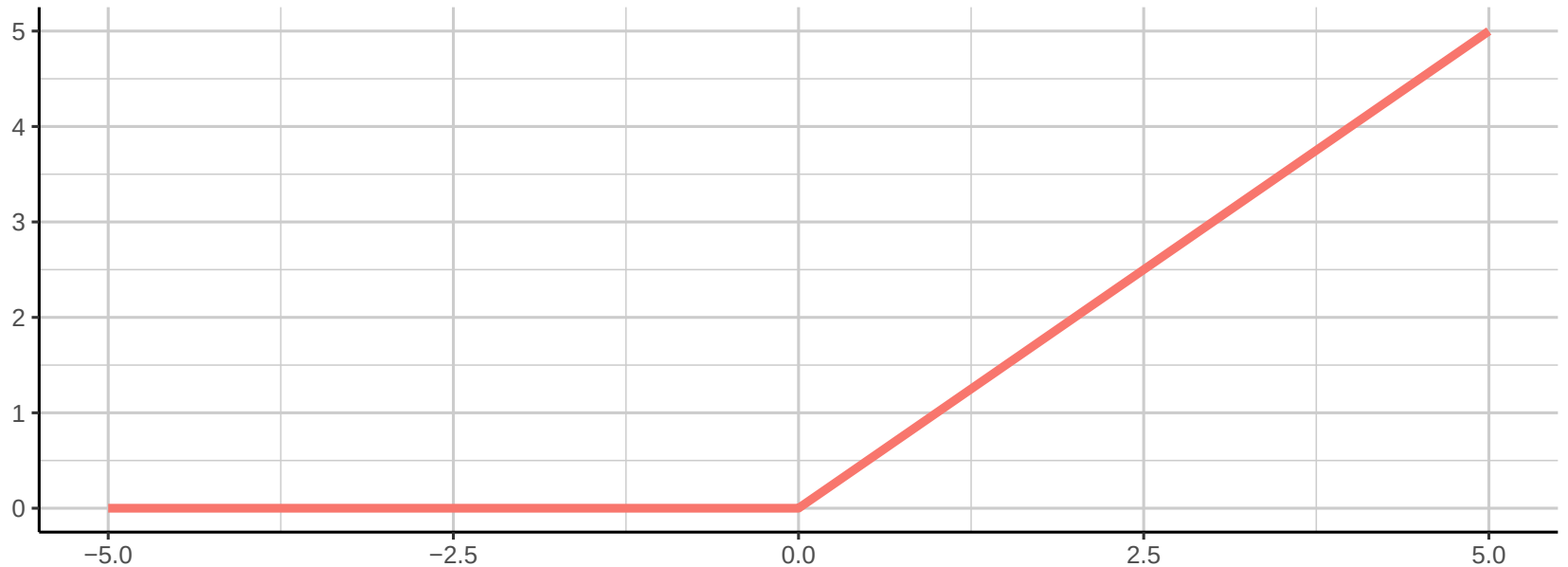
where $\mathbf{v} = \mathbf{W}\mathbf{u}$. It can be seen that $f(\mathbf{x})$ can only yield a linear decision boundary.

HIDDEN LAYER: ACTIVATION FUNCTION

ReLU activation:

- Currently the most popular choice is the ReLU (rectified linear unit):

$$\sigma(v) = \max(0, v)$$



HIDDEN LAYER: ACTIVATION FUNCTION

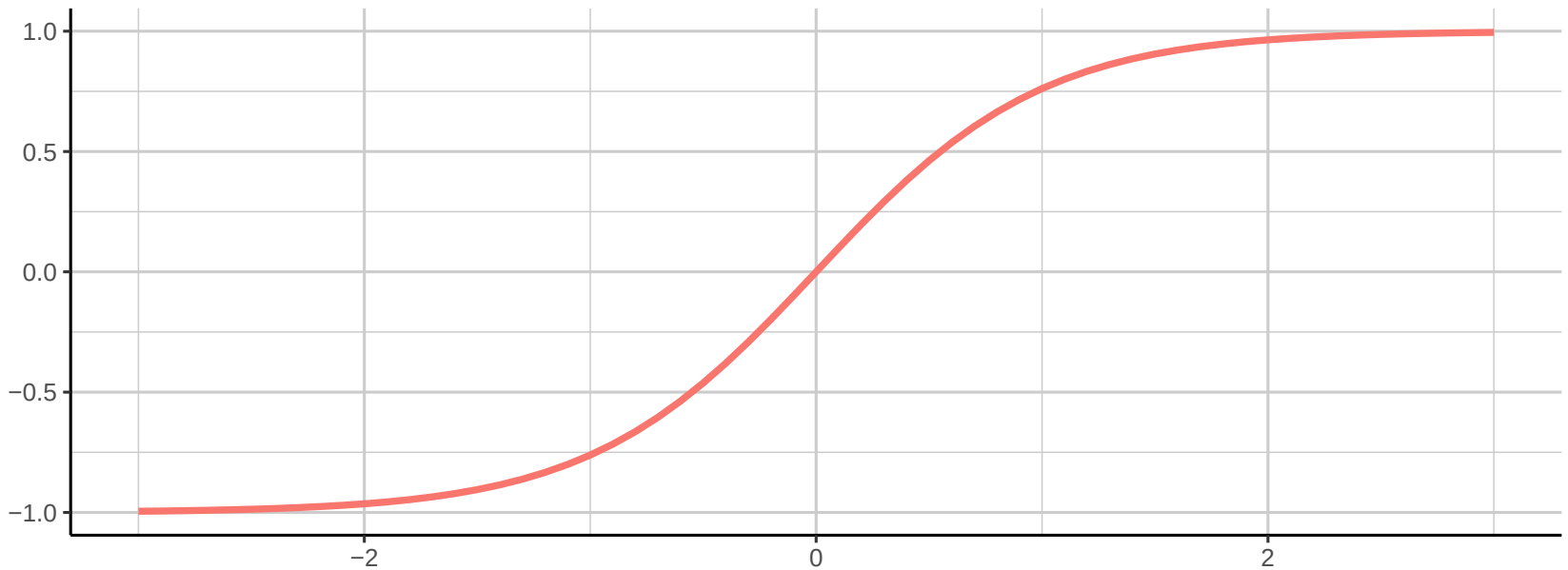
- Some important properties of the ReLU function include:
  - limits:
$$\lim_{v \to -\infty} \sigma(v) = 0 \text{ and } \lim_{v \to \infty} \sigma(v) = \infty$$
  - derivative:
$$\frac{\delta \sigma(v)}{\delta v} = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{else} \end{cases}$$

HIDDEN LAYER: ACTIVATION FUNCTION

Hyperbolic tangent activation:

- Another choice might be the hyperbolic tangent function:

$$\sigma(v) = \tanh(v) = \frac{\sinh(v)}{\cosh(v)} = 1 - \frac{2}{\exp(2v) + 1}$$



HIDDEN LAYER: ACTIVATION FUNCTION

- Some important properties of the hyperbolic tangent function include:
  - limits:
    $$\lim_{v \to -\infty} \sigma(v) = -1 \text{ and } \lim_{v \to \infty} \sigma(v) = 1$$
  - derivative:
    $$\frac{\delta \sigma(v)}{\delta v} = 1 - \tanh^2(v)$$
  - symmetry:

    $\sigma(v)$ is rotationally symmetric about $(0, 0)$

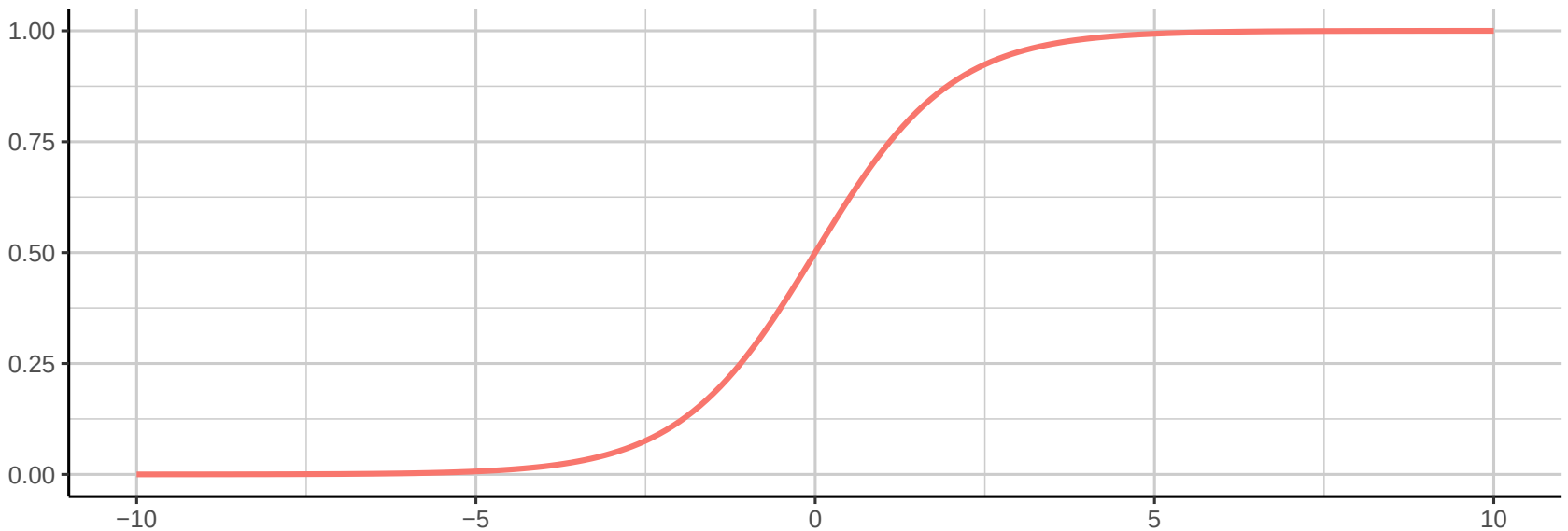
    (that is, a rotation of 180° does not change the graph of the function.)

HIDDEN LAYER: ACTIVATION FUNCTION

Sigmoid activation function:

- Of course, as seen in the previous example, the sigmoid function can be used even in the hidden layer:

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



HIDDEN LAYER: ACTIVATION FUNCTION

- Some important properties of the logistic sigmoid function include:
  - limits:
$$\lim_{v \to -\infty} \sigma(v) = 0 \text{ and } \lim_{v \to \infty} \sigma(v) = 1$$
  - the derivative:
$$\frac{\delta \sigma(v)}{\delta v} = \frac{\exp(v)}{(1 + \exp(v))^2} = \sigma(v)(1 - \sigma(v))$$
  - symmetry:

    $\sigma(v)$ is rotationally symmetric about $(0, 0.5)$

Multi-Layer Feedforward Neural Networks

FEEDFORWARD NEURAL NETWORKS

- We will now extend the model class once again, such that we allow an arbitrary amount of l (hidden) layers.

- The general term for this model class is (multi-layer) **feedforward networks** (inputs are passed through the network from left to right, no feedback-loops are allowed)

FEEDFORWARD NEURAL NETWORKS

- We can characterize those models by the following chain structure:

$$f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(l)} \circ \phi^{(l)} \circ \sigma^{(l-1)} \circ \phi^{(l-1)} \circ \ldots \circ \sigma^{(1)} \circ \phi^{(1)}$$

  where $\sigma^{(i)}$ and $\phi^{(i)}$ are the activation function and the weighted sum of hidden layer i, respectively. τ and ϕ are the corresponding components of the output layer.

- Each hidden layer has:
  - an associated weight matrix $\mathbf{W}^{(i)}$, bias $\mathbf{b}^{(i)}$ and activations $\mathbf{z}^{(i)}$ for $i \in \{1 \ldots l\}$
  - $\mathbf{z}^{(i)} = \sigma^{(i)}(\phi^{(i)}) = \sigma^{(i)}(\mathbf{W}^{(i)T}\mathbf{z}^{(i-1)} + \mathbf{b}^{(i)})$, where $\mathbf{z}^{(0)} = \mathbf{x}$.

- Again, without non-linear activations in the hidden layers, the network can only learn linear decision boundaries.

FEEDFORWARD NEURAL NETWORKS

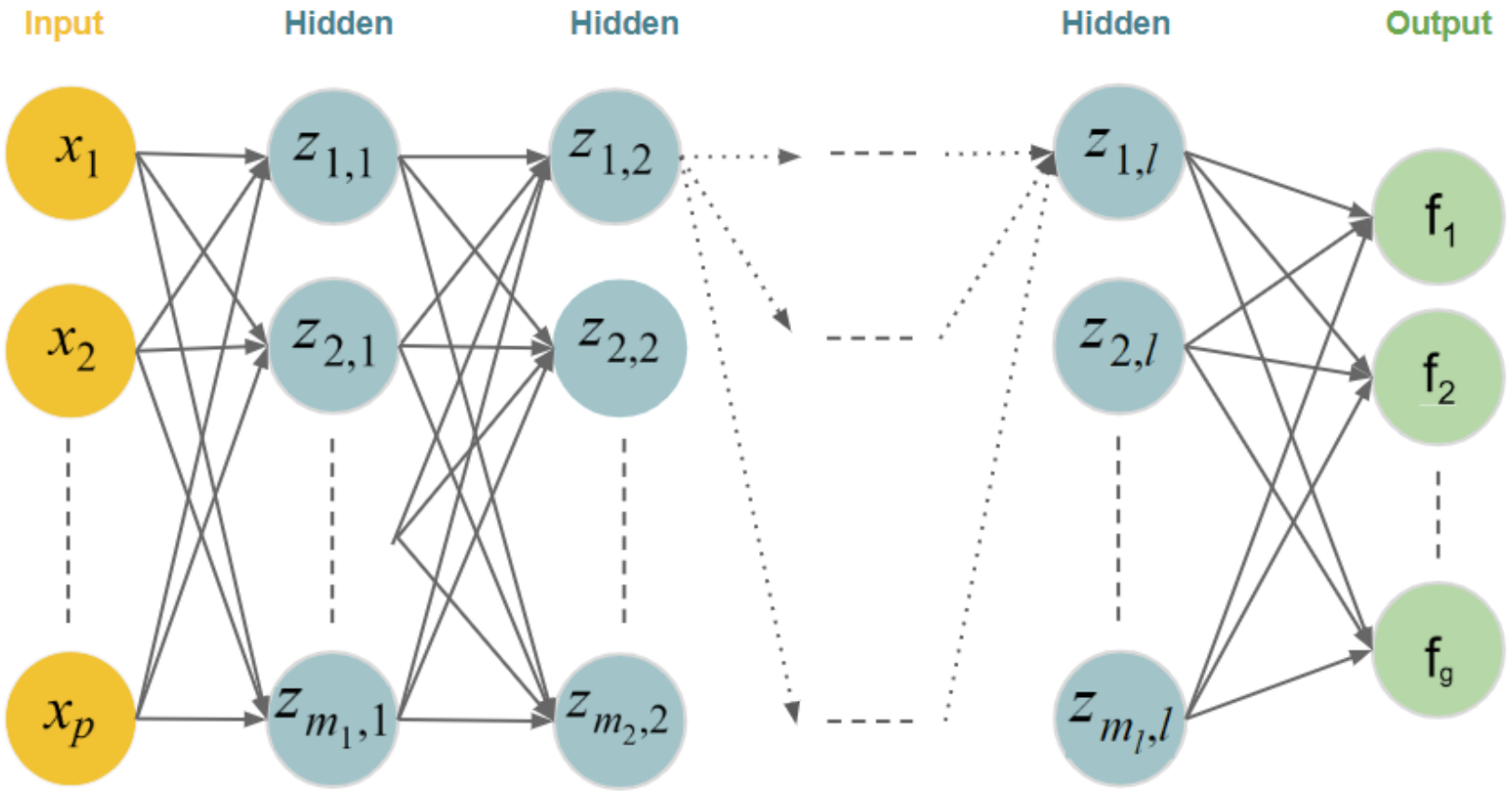
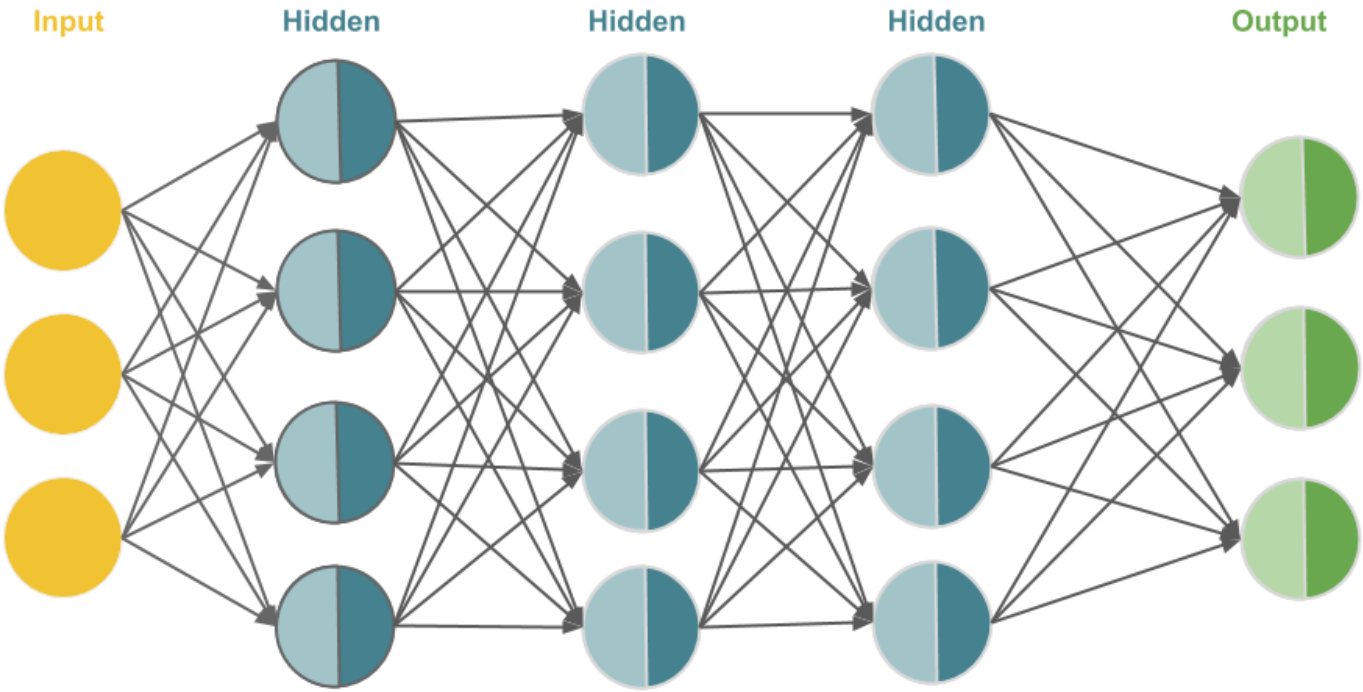


Figure: Structure of a deep neural network with l hidden layers (bias terms omitted).

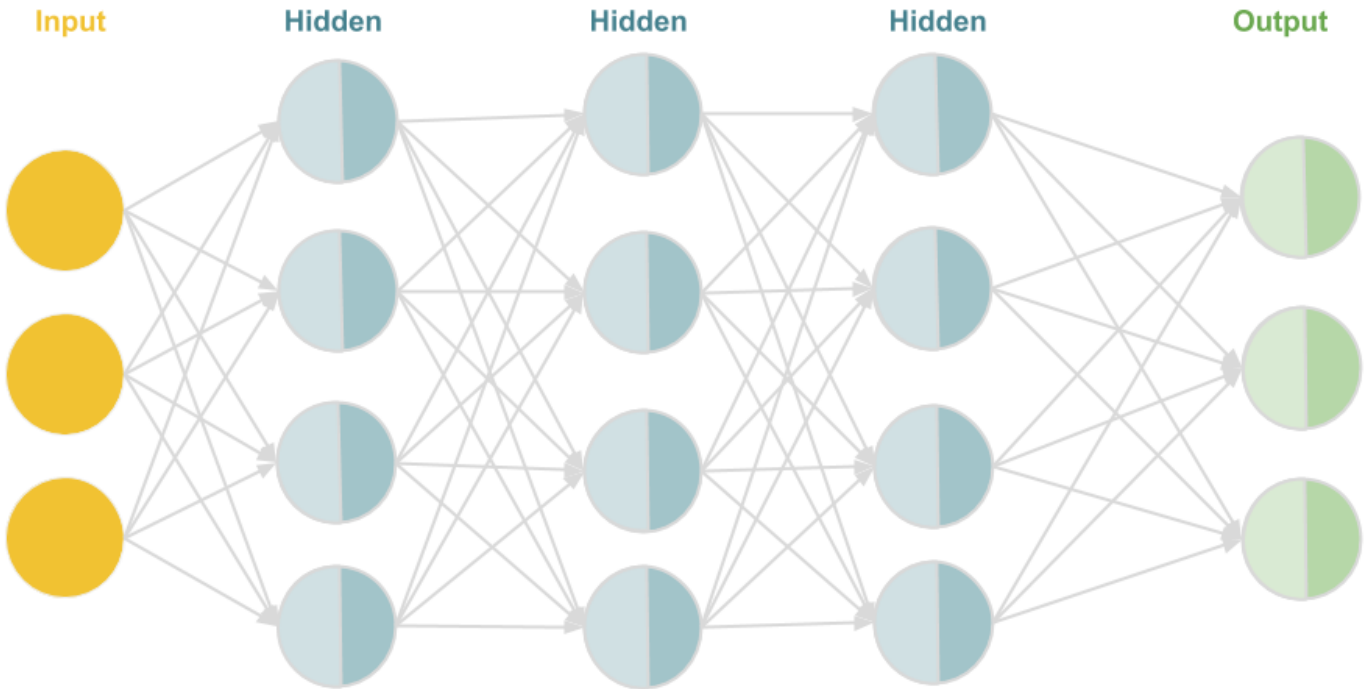
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{pmatrix} 13 & -9 & 2 \\ -8 & 0 & 3 \\ 4 & -1 & 5 \\ -3 & 12 & 7 \end{pmatrix} \begin{pmatrix} 5 \\ -2 \\ 2 \\ 11 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -4 & 1 \\ 0 & 11 & 2 & -14 \\ -1 & 5 & -2 & 16 \\ 0 & -9 & -3 & 4 \end{pmatrix} \begin{pmatrix} -5 \\ 3 \\ 1 \\ -8 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & -18 & -7 \\ 3 & -4 & 8 & 0 \\ -2 & 1 & 21 & 5 \\ 2 & -2 & 11 & -13 \end{pmatrix} \begin{pmatrix} 4 \\ -6 \\ 1 \\ -17 \end{pmatrix} \quad \begin{pmatrix} 9 & 3 & -1 & -4 \\ -8 & -2 & 14 & 3 \\ 13 & 2 & -9 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ -4 \\ -30 \end{pmatrix}$$

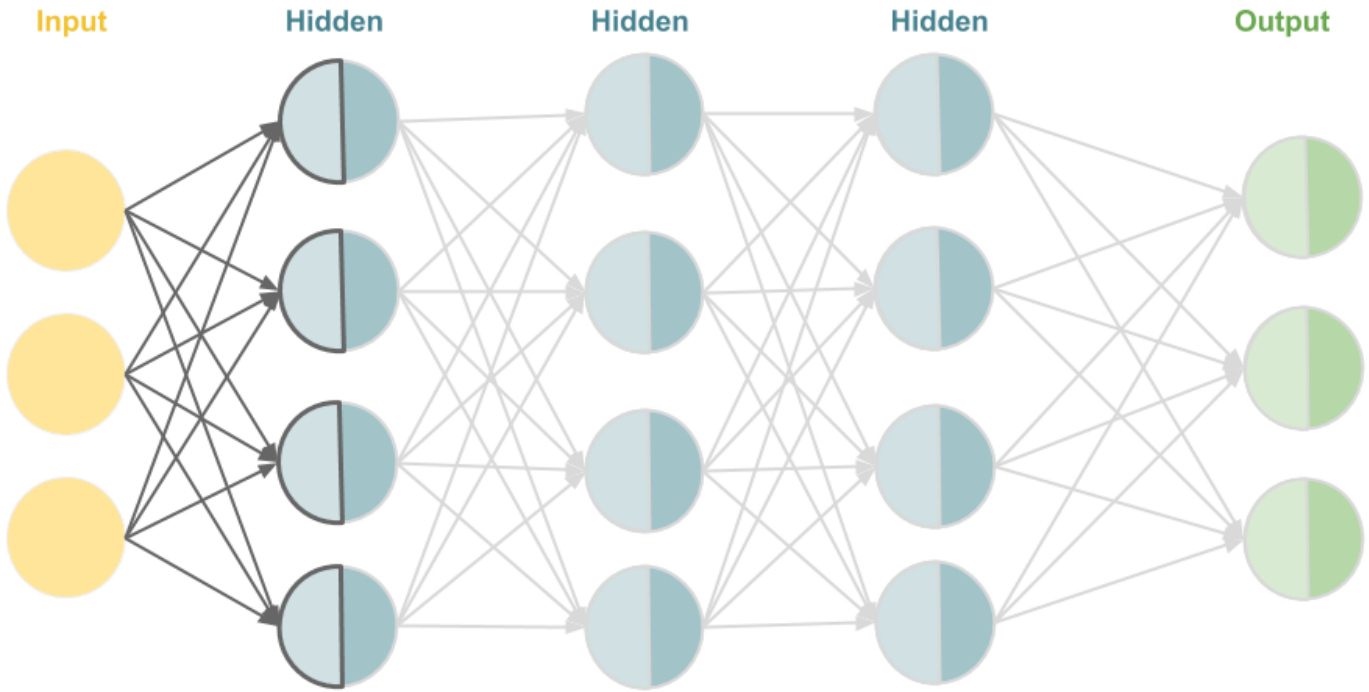
$$(W^{(1)})^T \quad \mathbf{b}^{(1)} \quad (W^{(2)})^T \quad \mathbf{b}^{(2)} \quad (W^{(3)})^T \quad \mathbf{b}^{(3)} \quad U^T \quad \mathbf{c}$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} = \begin{pmatrix} 7 \\ 1 \\ -4 \end{pmatrix}$$

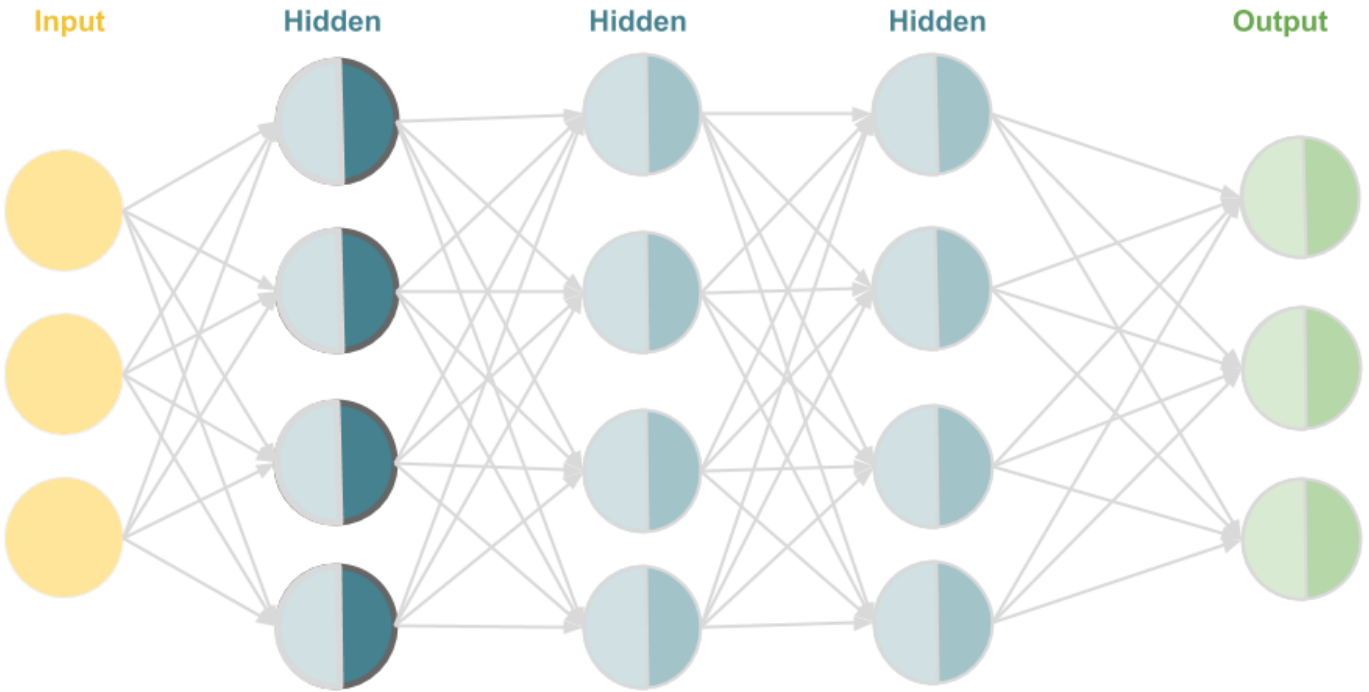
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{matrix} 7 \\ 1 \\ -4 \end{matrix} \begin{pmatrix} 7*13 + 1*(-9) + (-4)*2 + 5 \\ 7*(-8) + 1*0 + (-4)*3 + (-2) \\ 7*4 + 1*(-1) + (-4)*5 + 2 \\ 7*(-3) + 1*12 + (-4)*7 + 11 \end{pmatrix}$$

$\mathbf{x}$ $\quad \mathbf{z}_{in}^{(1)} = W^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}$

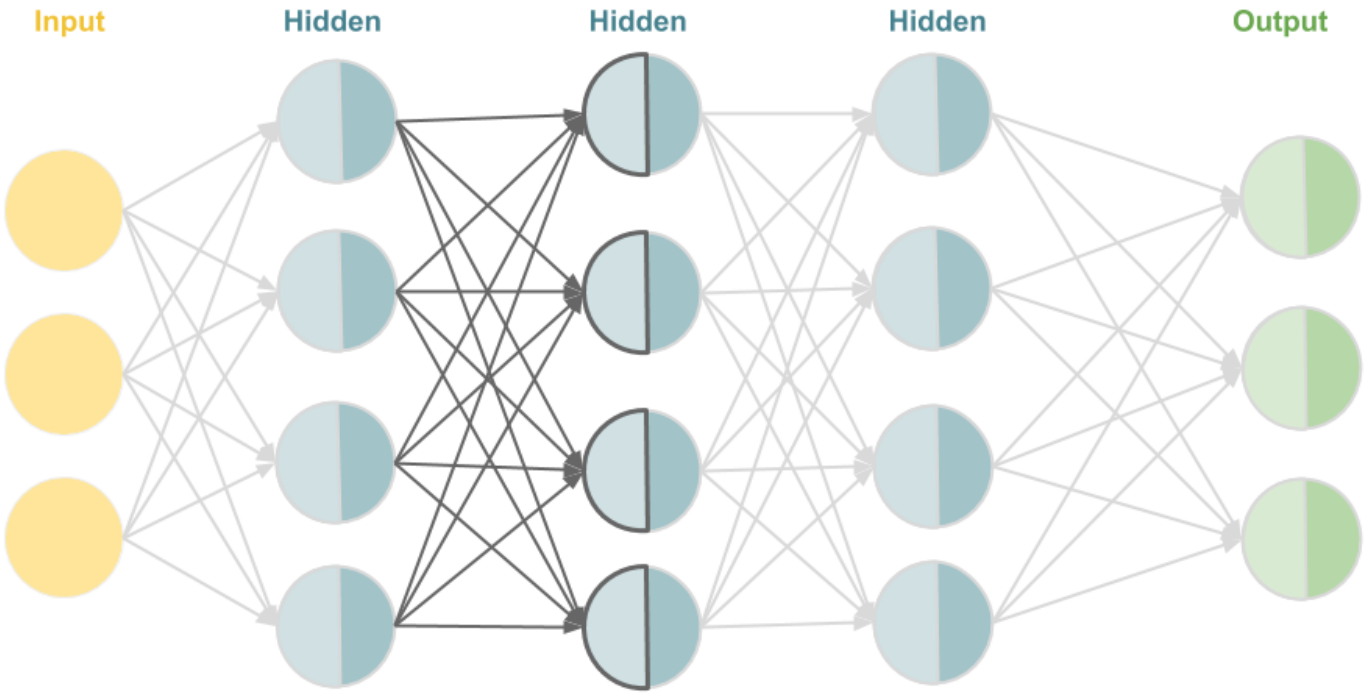
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



7 1 -4	79 -70 9 -26	max(0, 79) max(0, -70) max(0, 9) max(0, -26)

$\mathbf{x}$ $\quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} = \mathbf{z}_{out}^{(1)} = \sigma(\mathbf{z}_{in}^{(1)})$

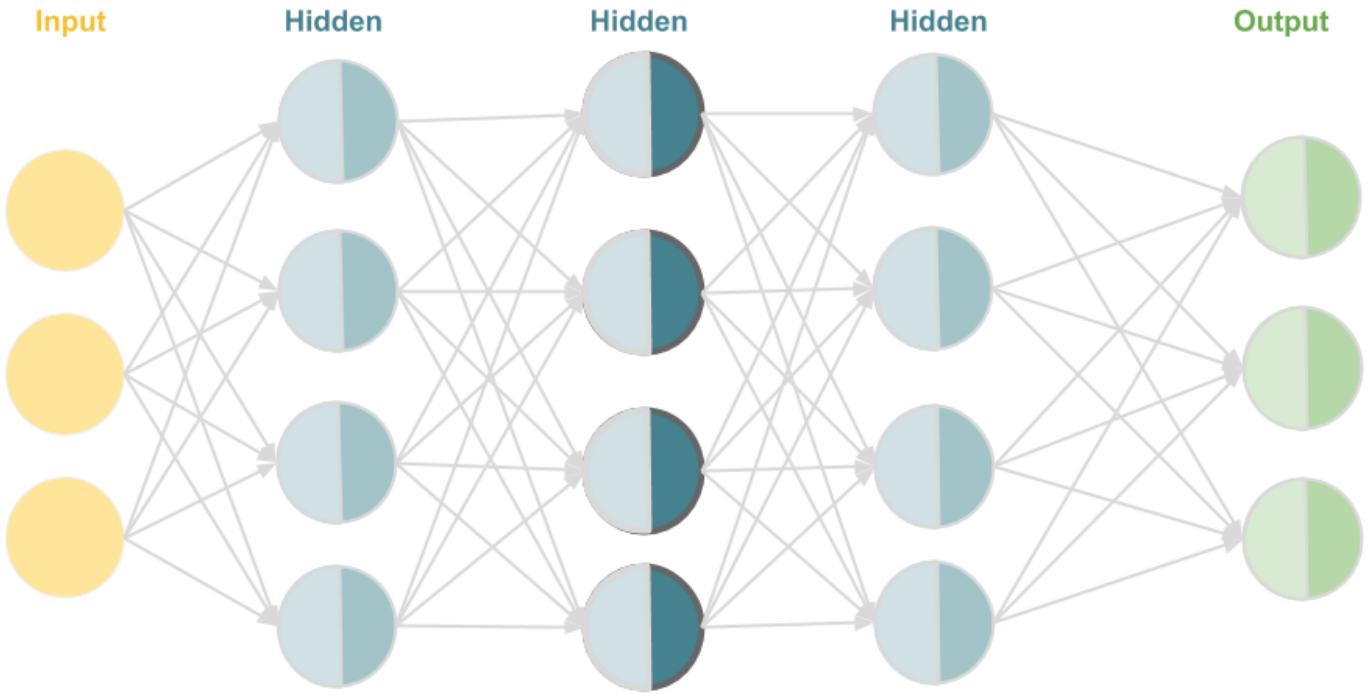
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



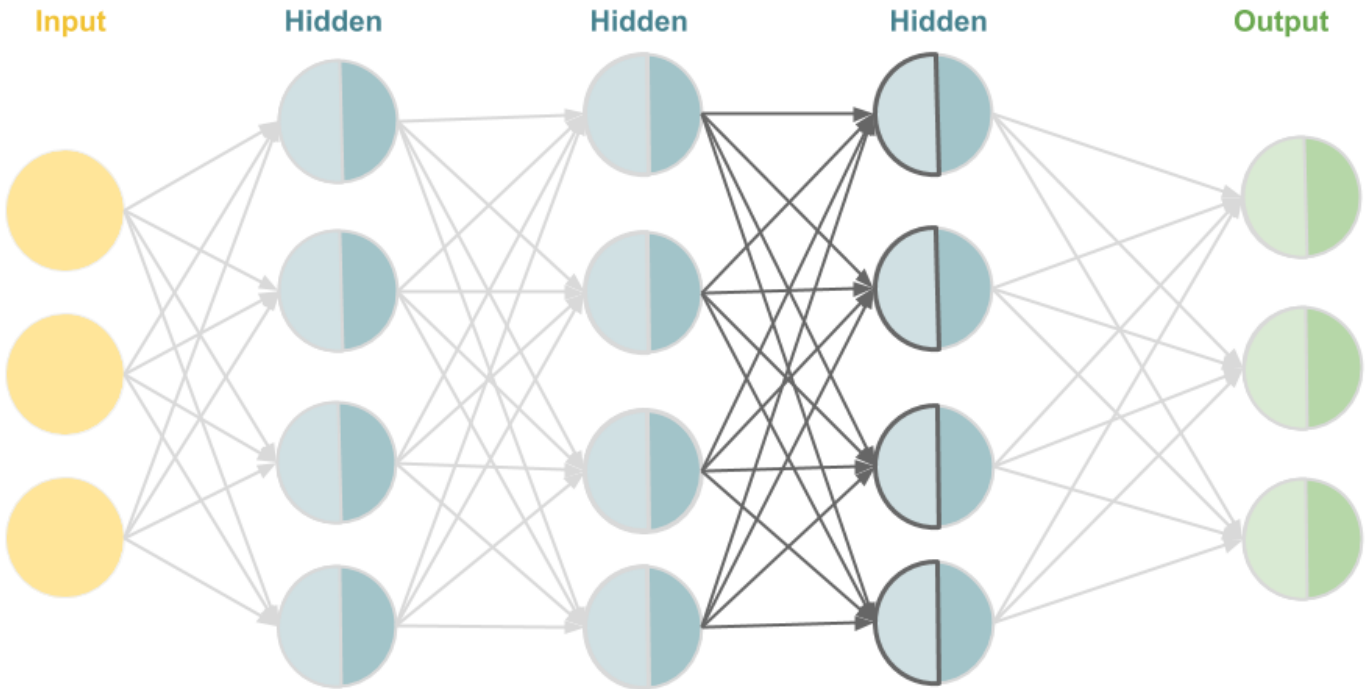
$$\mathbf{x} = \begin{pmatrix} 7 \\ 1 \\ -4 \end{pmatrix} \quad \mathbf{z}_{in}^{(1)} = \begin{pmatrix} 79 \\ -70 \\ 9 \\ -26 \end{pmatrix} \quad \mathbf{z}^{(1)} = \begin{pmatrix} 79 \\ 0 \\ 9 \\ 0 \end{pmatrix} \quad \mathbf{z}_{in}^{(2)} = W^{(2)T}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}$$

$$\begin{pmatrix} 79*1 + 0*0 + 9*(-4) + 0*1 + (-5) \\ 79*0 + 0*11 + 9*2 + 0*(-14) + 3 \\ 79*(-1) + 0*5 + 9*(-2) + 0*16 + 1 \\ 79*0 + 0*(-9) + 9*(-3) + 0*4 + (-8) \end{pmatrix}$$

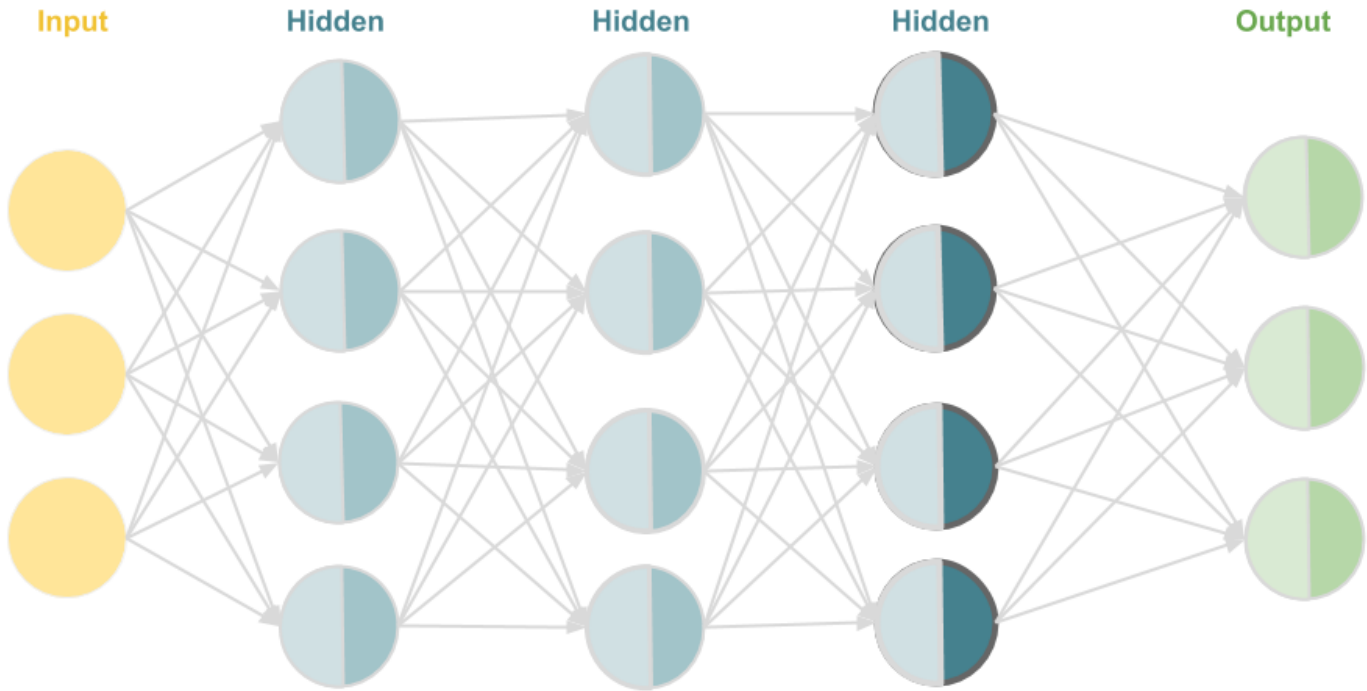
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



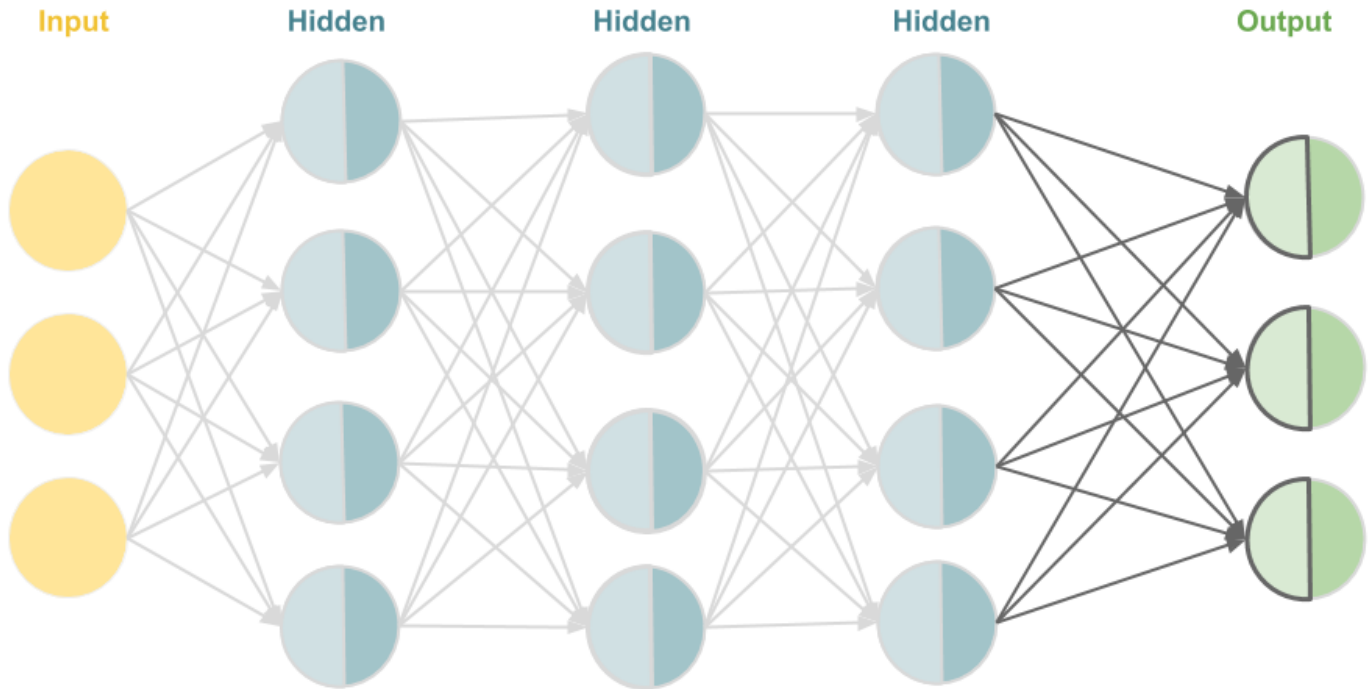
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



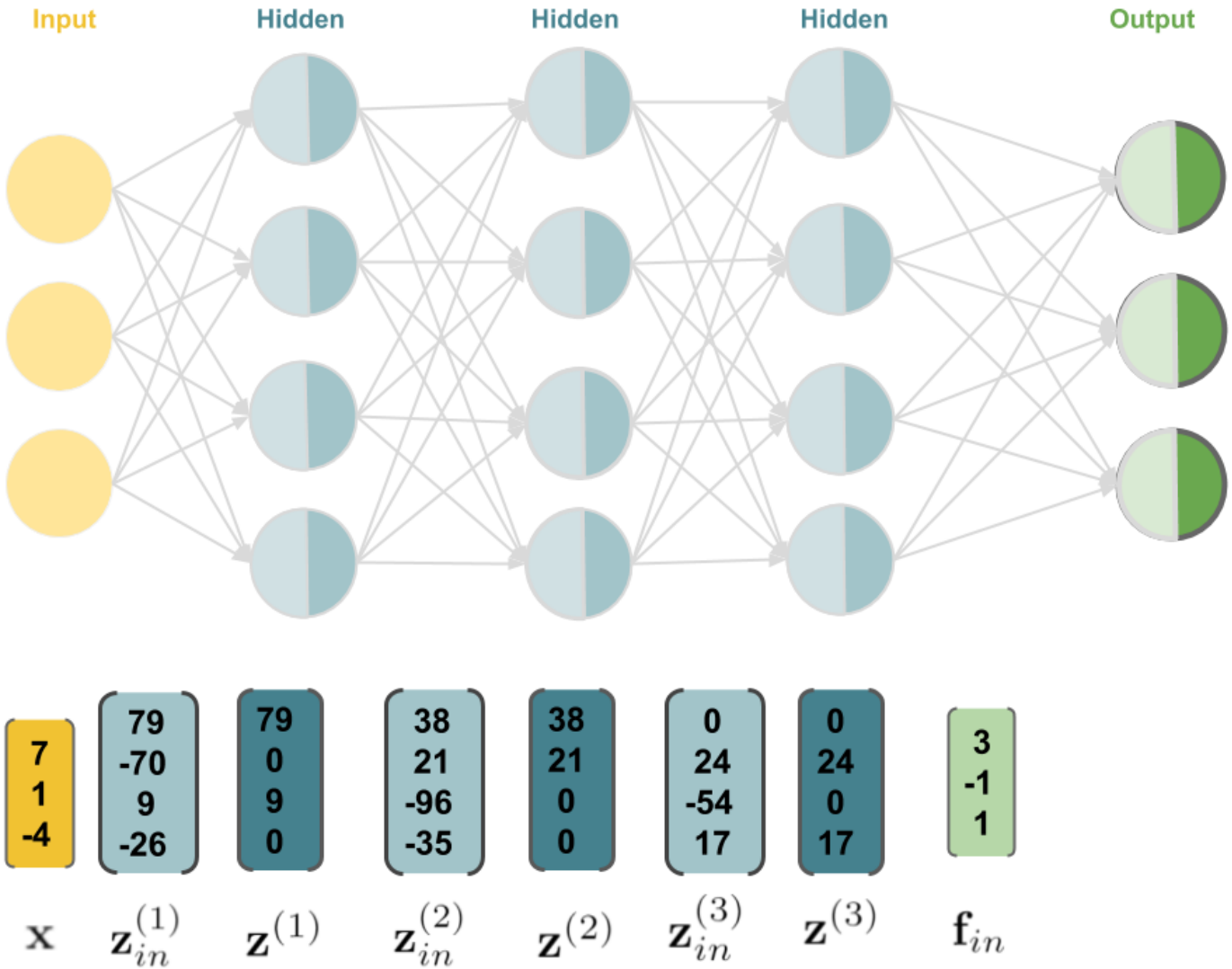
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



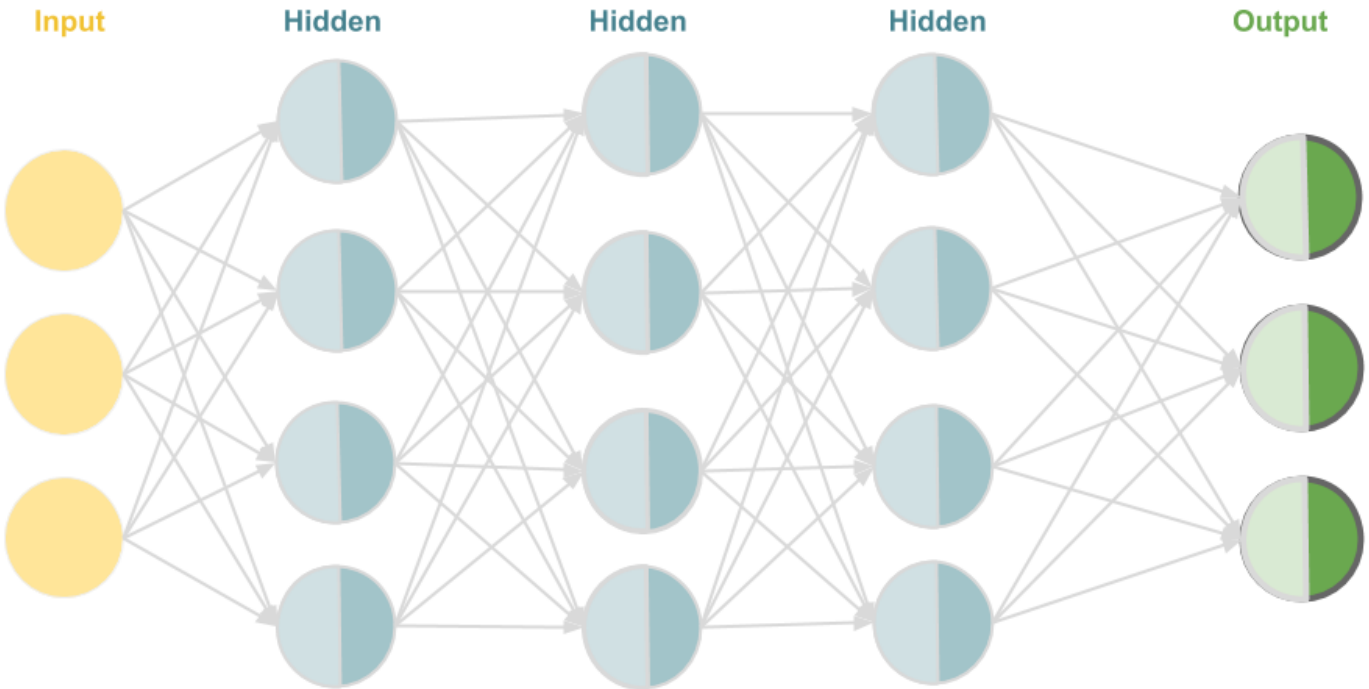
FEEDFORWARD NEURAL NETWORKS: EXAMPLE



FEEDFORWARD NEURAL NETWORKS: EXAMPLE



FEEDFORWARD NEURAL NETWORKS: EXAMPLE



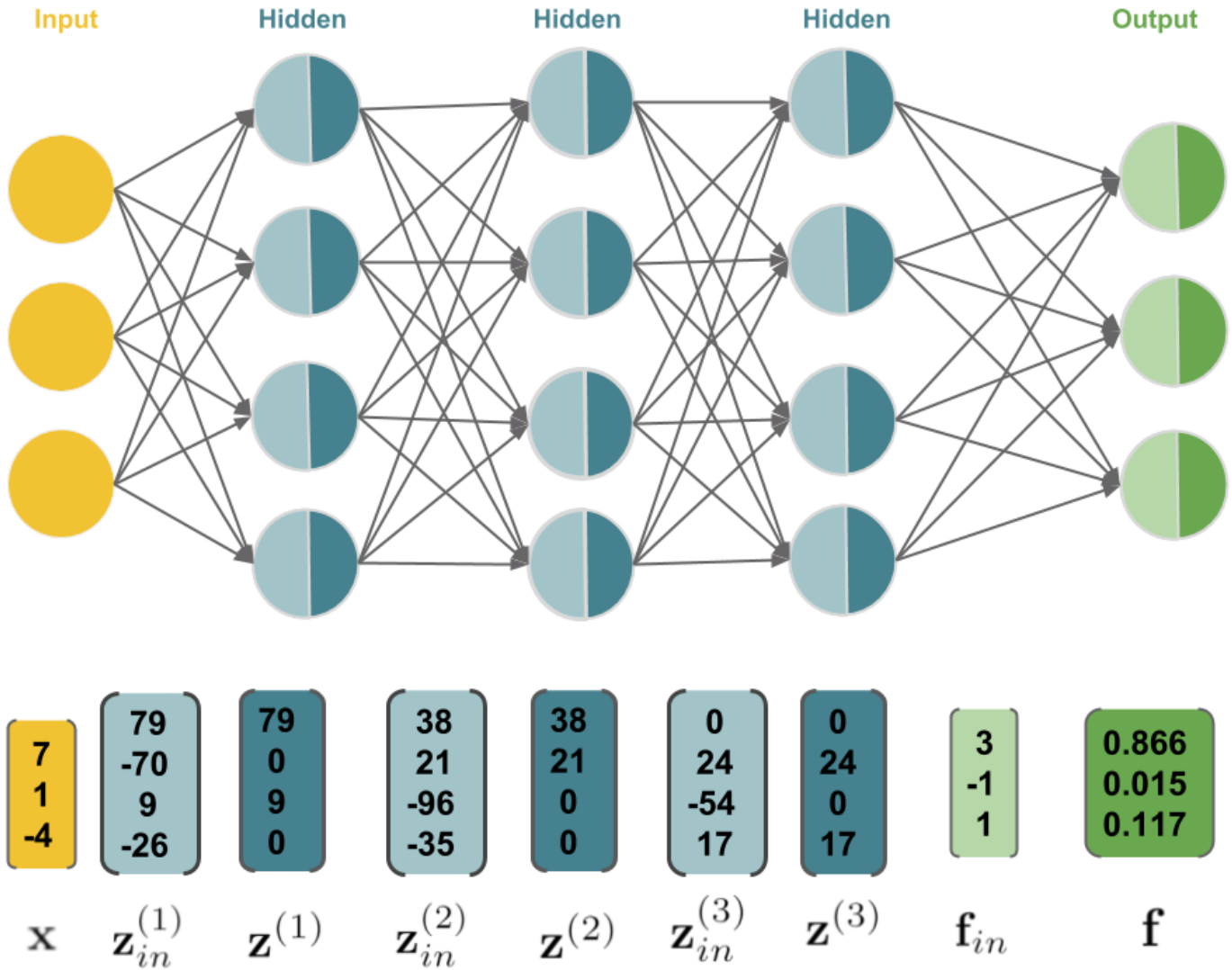
$$\mathbf{f}_{in}$$

3
-1
1

$$\frac{\exp(3)}{\exp 3+\exp(-1)+\exp(1)}$$
$$\frac{\exp(-1)}{\exp 3+\exp(-1)+\exp(1)}$$
$$\frac{\exp(1)}{\exp 3+\exp(-1)+\exp(1)}$$

$$\mathbf{f} = \mathbf{f}_{out} = \tau(\mathbf{f}_{in})$$

FEEDFORWARD NEURAL NETWORKS: EXAMPLE



WHY ADD MORE LAYERS?

- Multiple layers allow for the extraction of more and more abstract representations.

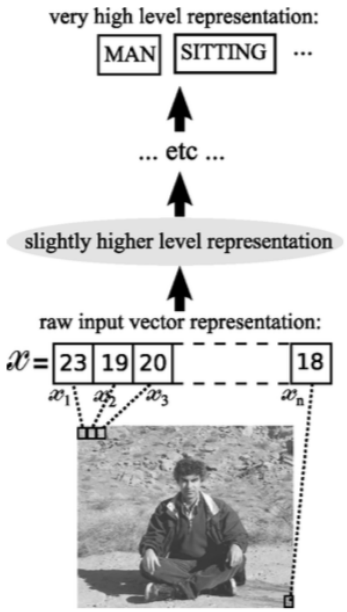


Figure: Y. Bengio, Learning Deep Architectures for AI, Foundations and trends® in Machine Learning, 2009

WHY ADD MORE LAYERS?

- Each layer in a feed-forward neural network adds its own degree of non-linearity to the model.

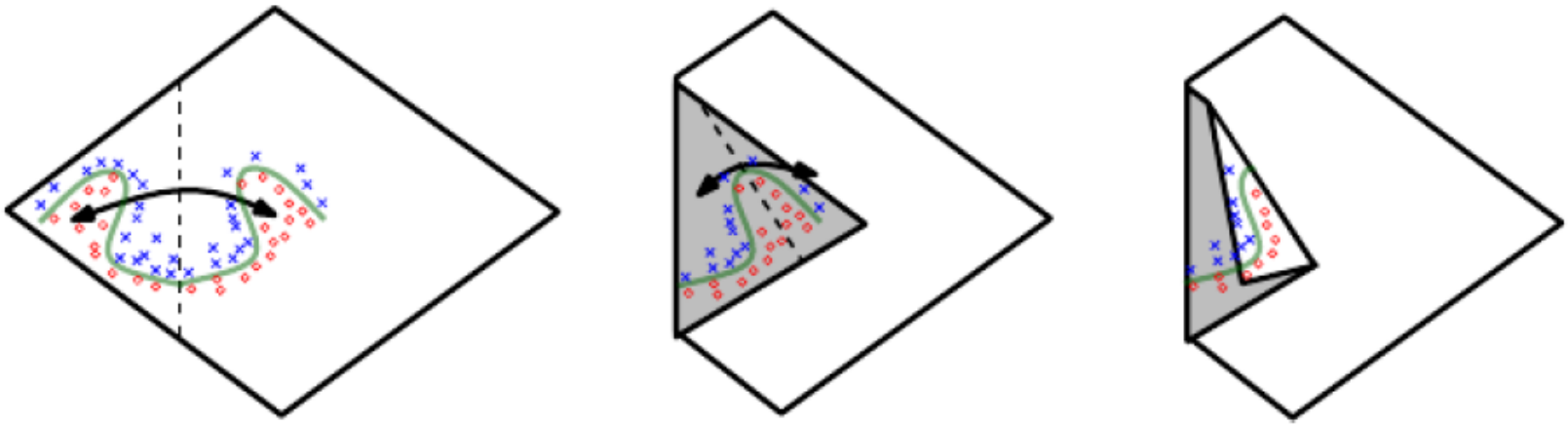


Figure: An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

Deep Learning

DEEP NEURAL NETWORKS

- Neural networks today can have dozens or even hundreds of hidden layers. The greater the number of layers, the "deeper" the network.
- Historically, deep neural networks were very challenging to train for several reasons.
- For one thing, the use of sigmoid activations (such as logistic sigmoid and tanh) significantly slowed down training due to a phenomenon known as “vanishing gradients”. The introduction of the ReLU activation largely solved this problem.
- Additionally, training deep neural networks on CPUs was too slow to be practical. Switching over to GPUs (Graphics Processing Units) cut down training time by more than an order of magnitude.
- Another reason neural networks were not popular until the late ’00s is that when dataset sizes are small, other models (such as SVMs) and techniques (such as feature engineering) outperform them.

DEEP NEURAL NETWORKS

- Therefore, the availability of large datasets (such as ImageNet) and novel architectures that are capable to handle even complex tensor-shaped data (e.g. CNNs for image data), significantly faster hardware, and equally better optimization and regularization methods made it feasible to successfully implement deep neural networks in the last decade.
- An increase in depth often translates to an increase in performance on a given task.
- State-of-the-art neural networks, however, are much more sophisticated than the simple architectures we have encountered so far. (Stay tuned!)
- The term "**deep learning**" encompasses all of these developments and refers to the field as a whole.