LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Deep Learning

Chapter 4: MLP as Predictor

Bernd Bischl

Department of Statistics – LMU Munich

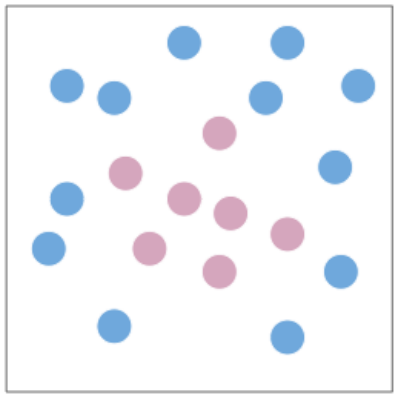
Winter term 2021

MOTIVATION

- The graphical way of representing simple functions/models, like logistic regression. Why is that useful?
- Because individual neurons can be used as building blocks of more complicated functions.
- 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

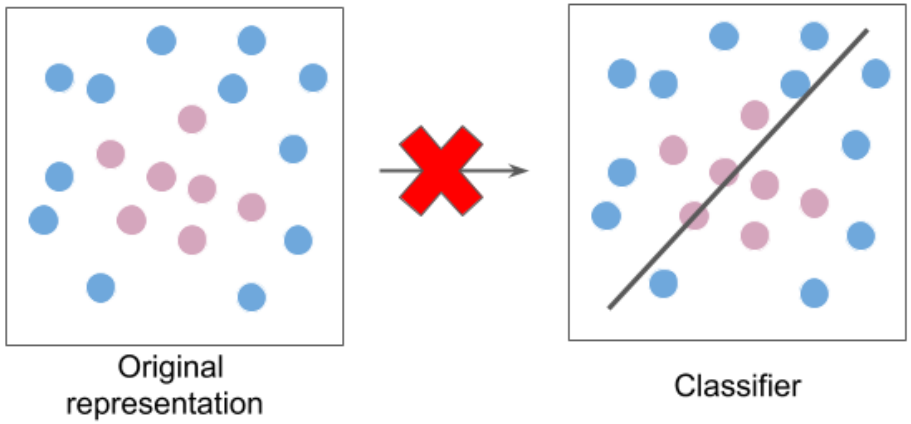
- As a single neuron is restricted to learning only linear decision boundaries, its performance on the following task is quite poor:



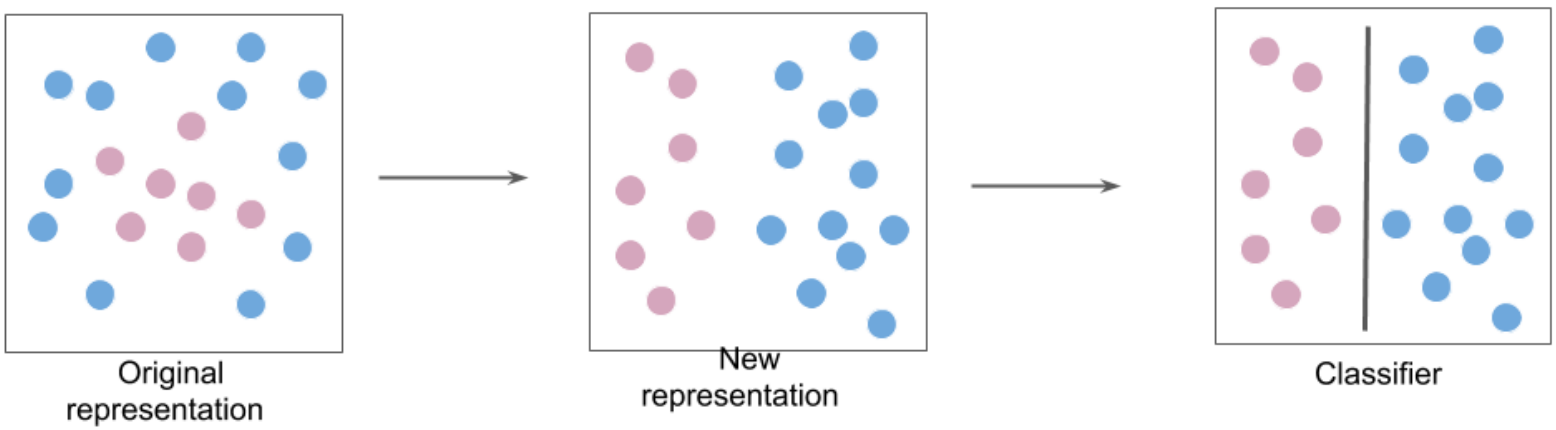
- However, the neuron can easily separate the classes if the original features are transformed (e.g., from Cartesian to polar coordinate):

MOTIVATION

- Instead of classifying the data in the original representation,

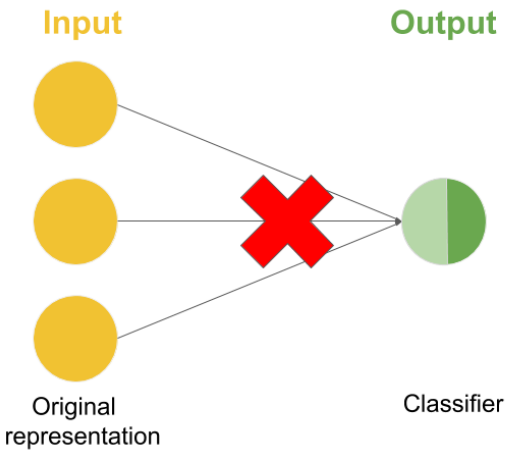


- we classify it in a new feature space.

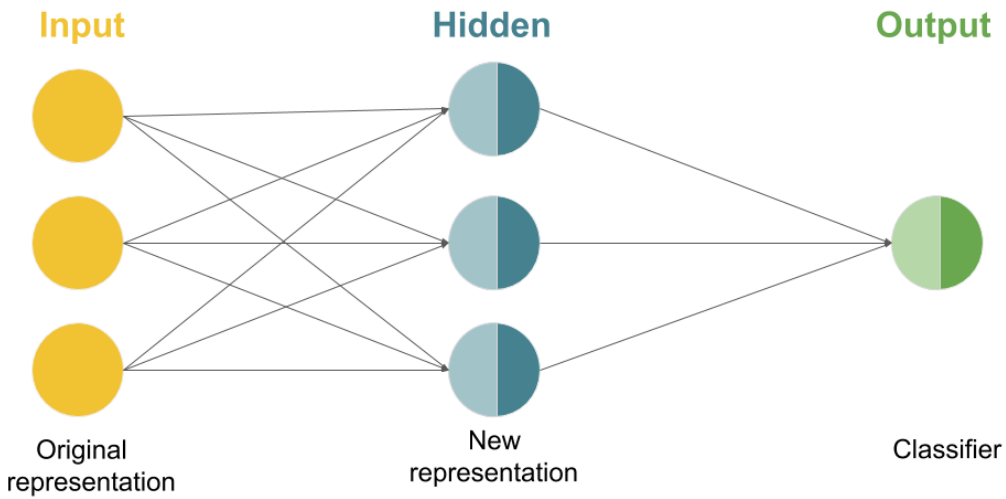


MOTIVATION

- Analogously, instead of a single neuron,



- we use more complex networks.



REPRESENTATION LEARNING

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

- Before deep learning 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

Single neurons perform a 2-step computation:

1. **Affine Transformation:** a weighted sum of inputs plus bias.
2. **Activation:** a non-linear transformation on the weighted sum.

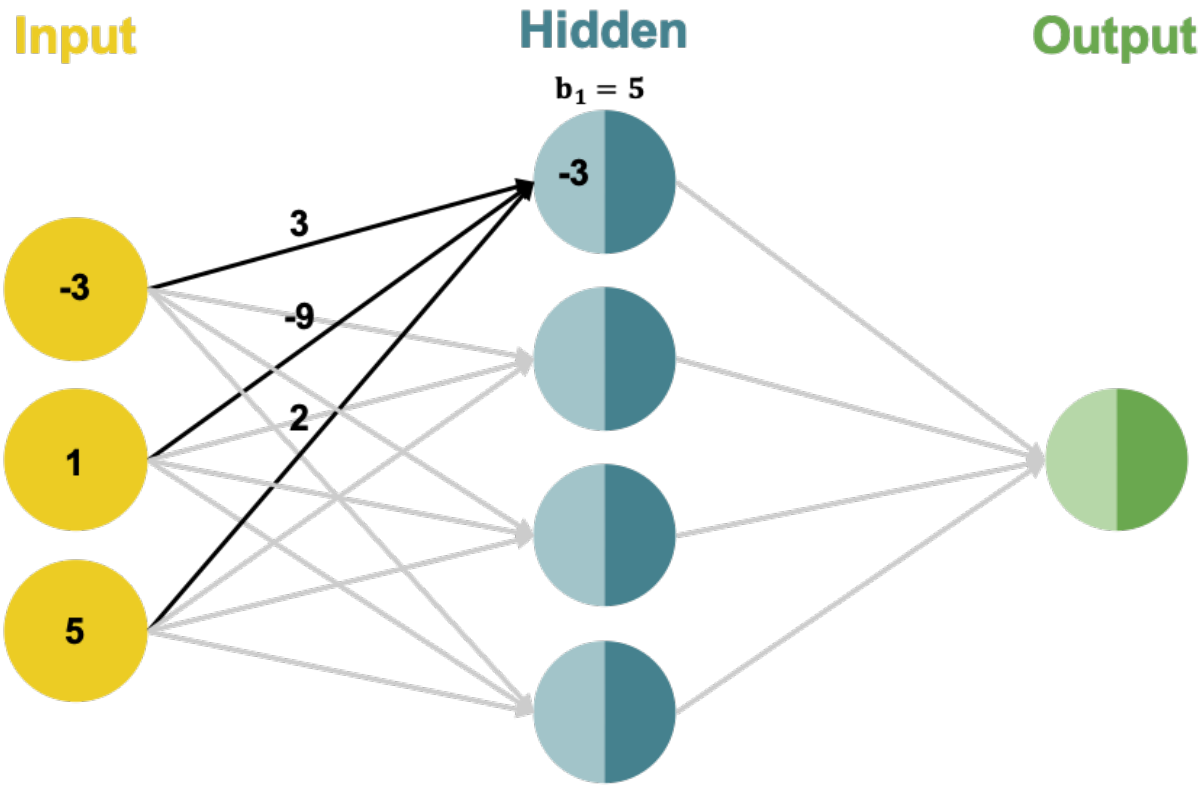
Single hidden layer networks consist of two layers:

1. **Hidden Layer:** having a set of neurons.
2. **Output Layer:** having one (potentially more) output neuron.

- Multiple inputs are simultaneously fed to the network.
- Each neuron in the hidden layer performs a 2-step computation.
- The final output of the network is then calculated by another 2-step computation performed by the neuron in the output layer.

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

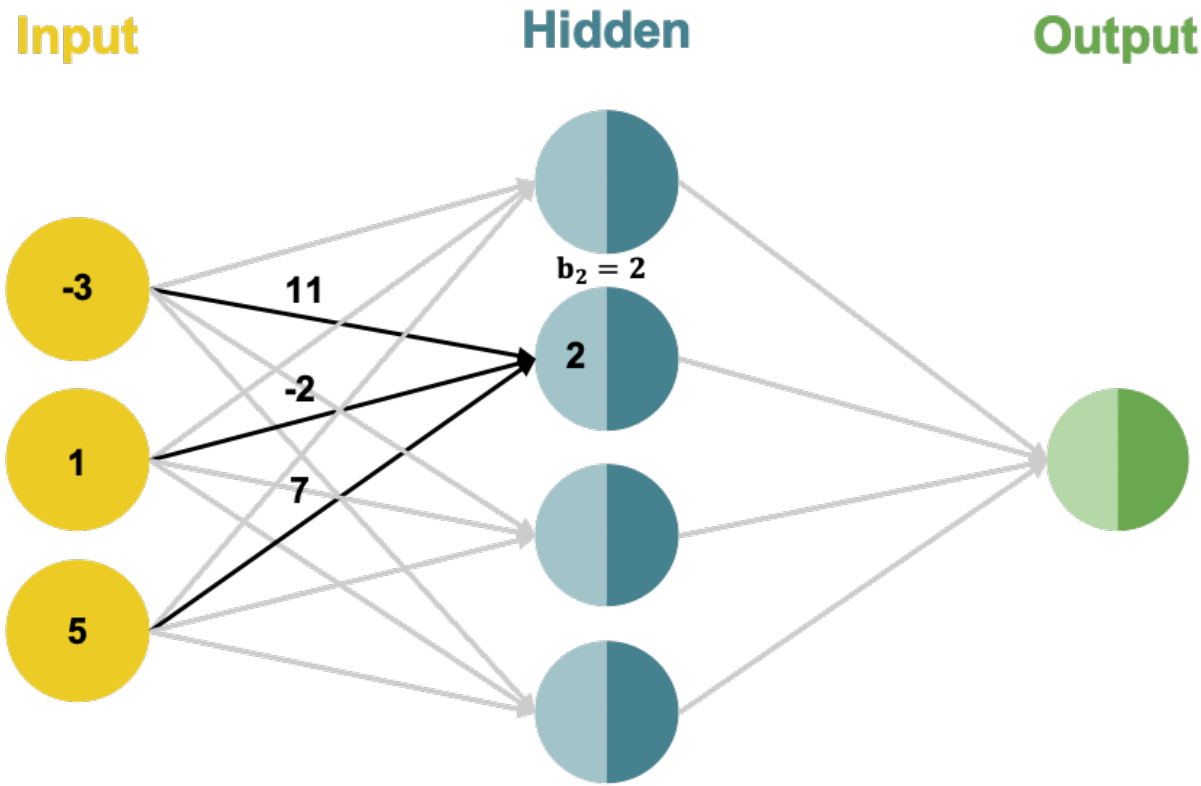


$$z_{in}^{(1)} = w_{11}x^{(1)} + w_{21}x^{(2)} + w_{31}x^{(3)} + b_1$$

$$z_{in}^{(1)} = 3 * (-3) + (-9) * 1 + 2 * 5 + 5 = -3$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

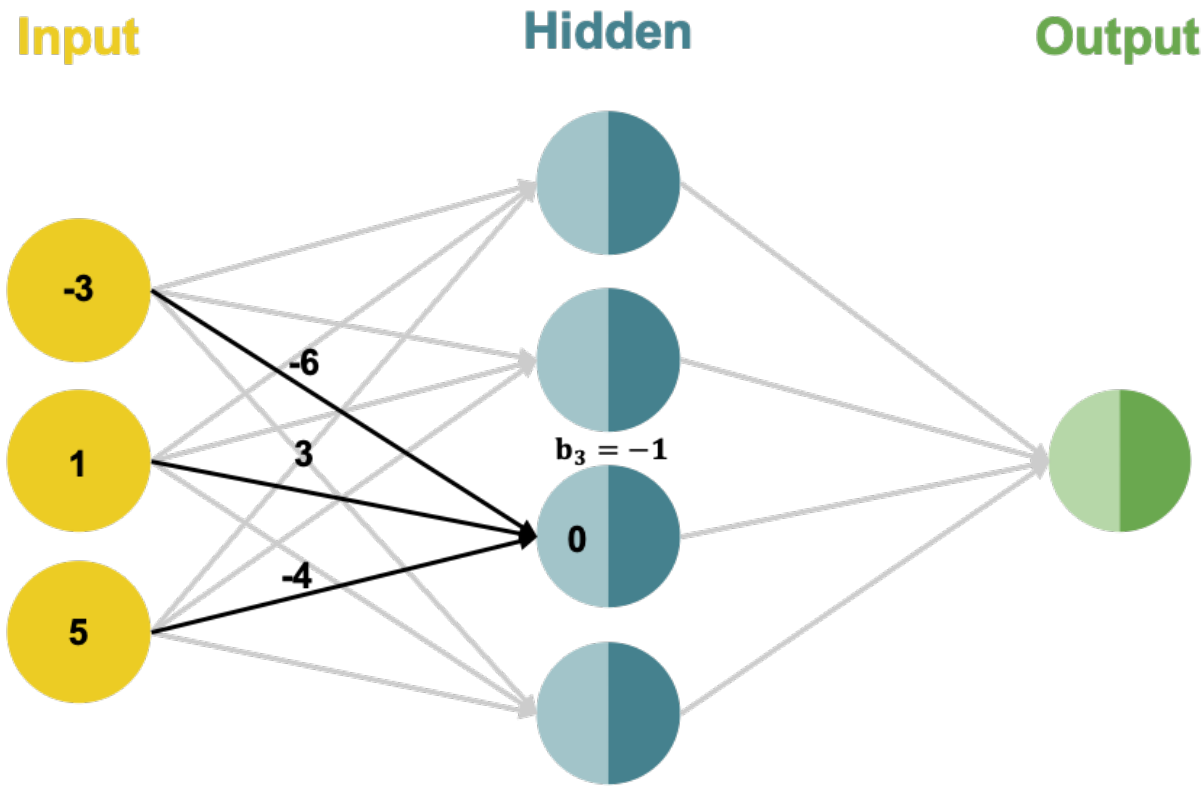


$$z_{in}^{(2)} = w_{12}x^{(1)} + w_{22}x^{(2)} + w_{32}x^{(3)} + b_2$$

$$z_{in}^{(2)} = 11 * (-3) + (-2) * 1 + 7 * 5 + 2 = 2$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

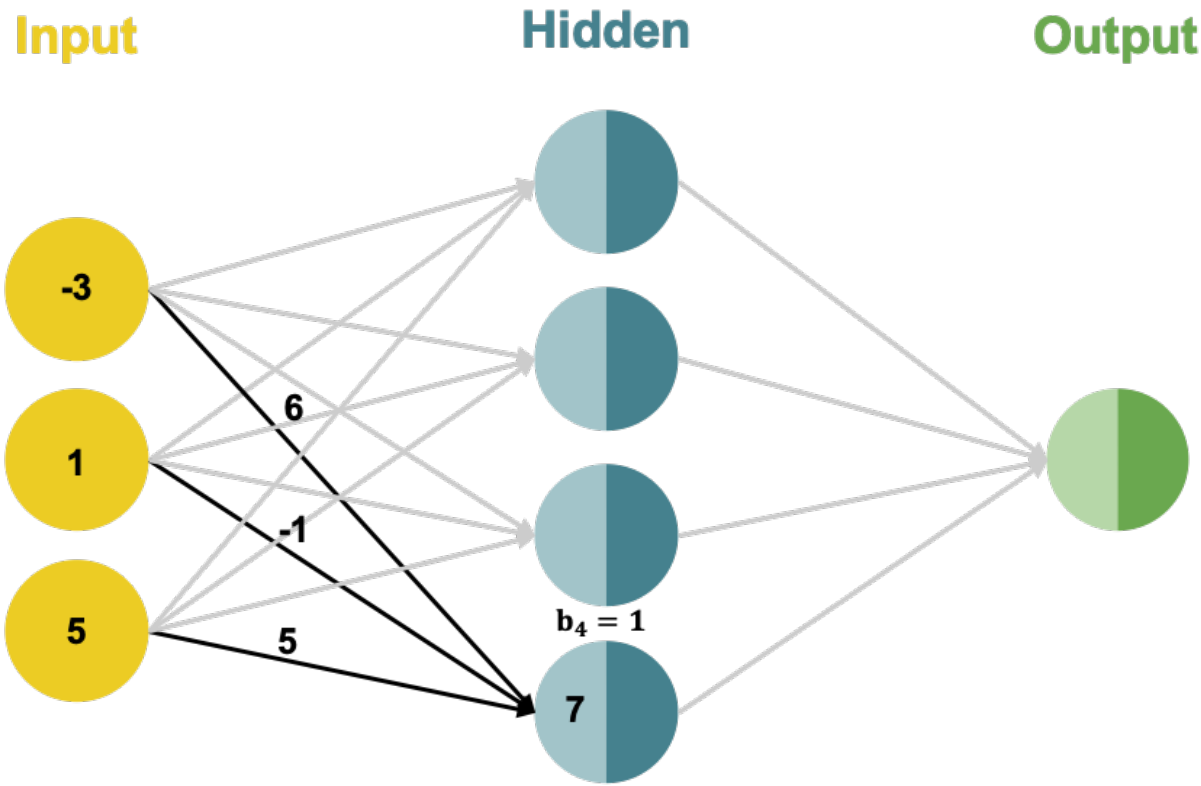


$$z_{in}^{(3)} = w_{13}x^{(1)} + w_{23}x^{(2)} + w_{33}x^{(3)} + b_3$$

$$z_{in}^{(3)} = (-6) * (-3) + 3 * 1 + (-4) * 5 - 1 = 0$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

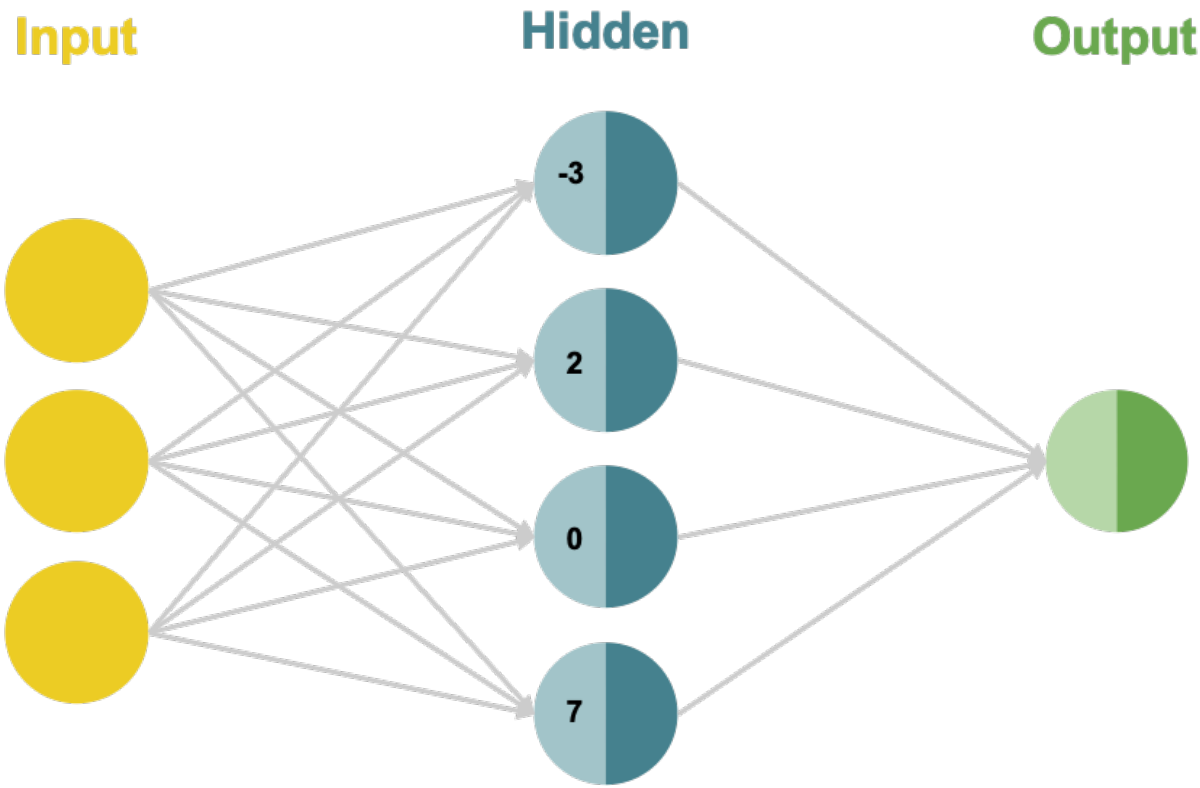


$$z_{in}^{(4)} = w_{14}x^{(1)} + w_{24}x^{(2)} + w_{34}x^{(3)} + b_4$$

$$z_{in}^{(4)} = 6 * (-3) + (-1) * 1 + 5 * 5 + 1 = 7$$

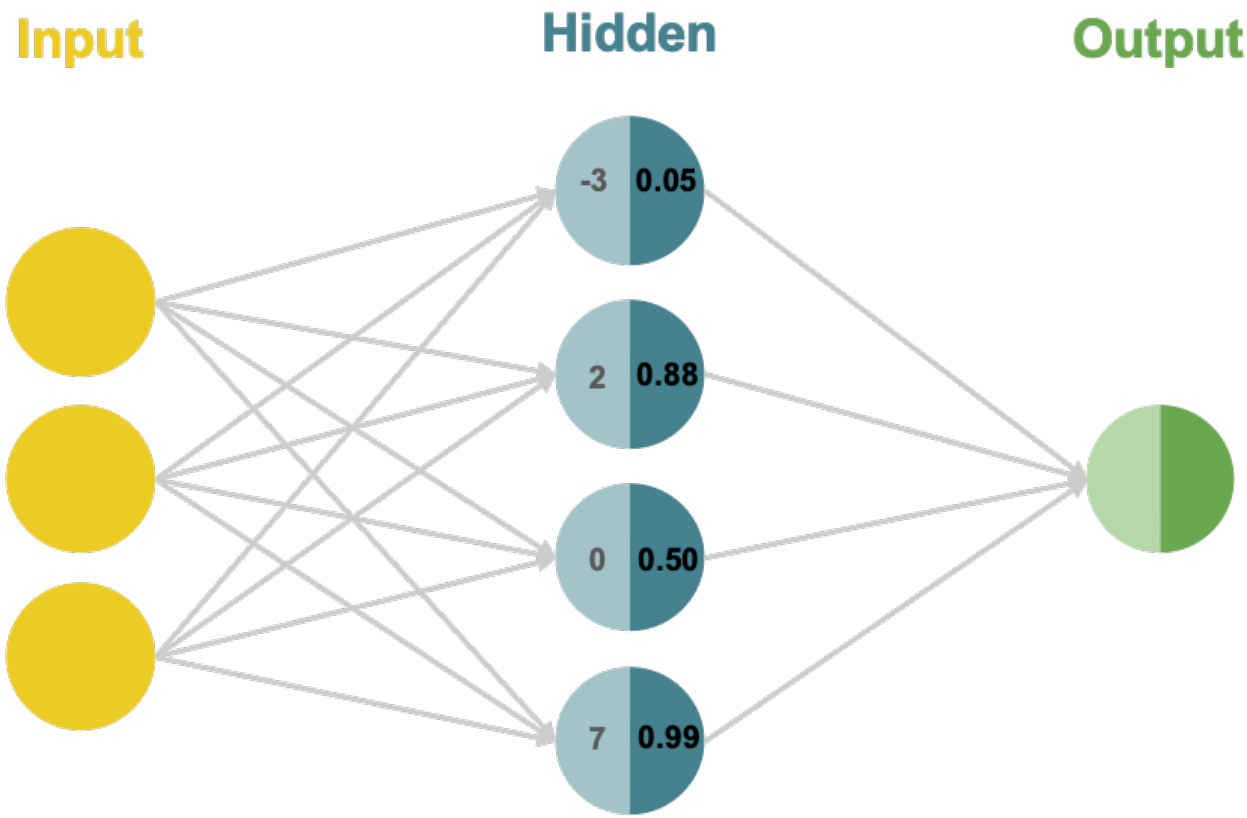
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:



SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

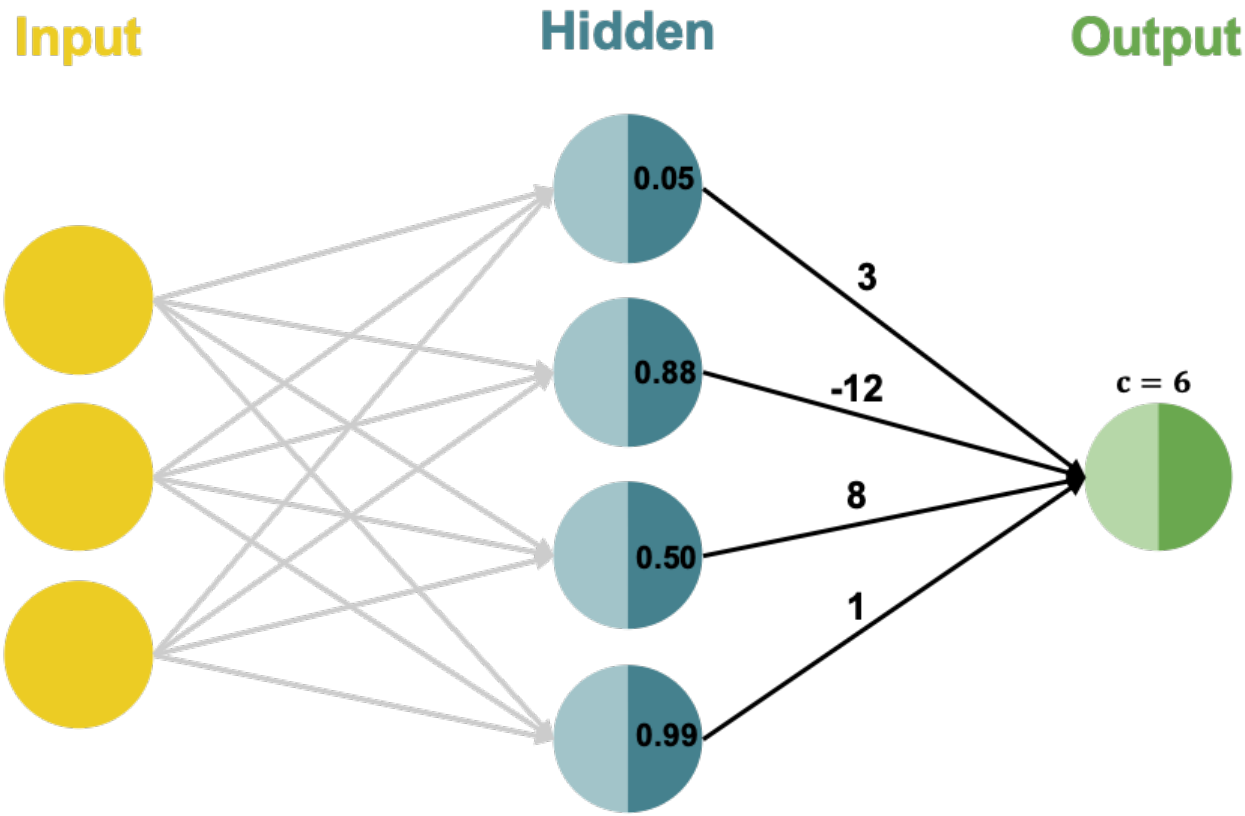
Each hidden neurons perform a non-linear **activation** transformation on the weight sum:



$$\mathbf{z}_{\mathbf{out}}^{(\mathbf{i})} = \boldsymbol{\sigma}\left(\mathbf{z}_{\mathbf{in}}^{(\mathbf{i})}\right) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{\mathbf{z}_{\mathbf{in}}^{(\mathbf{i})}}}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

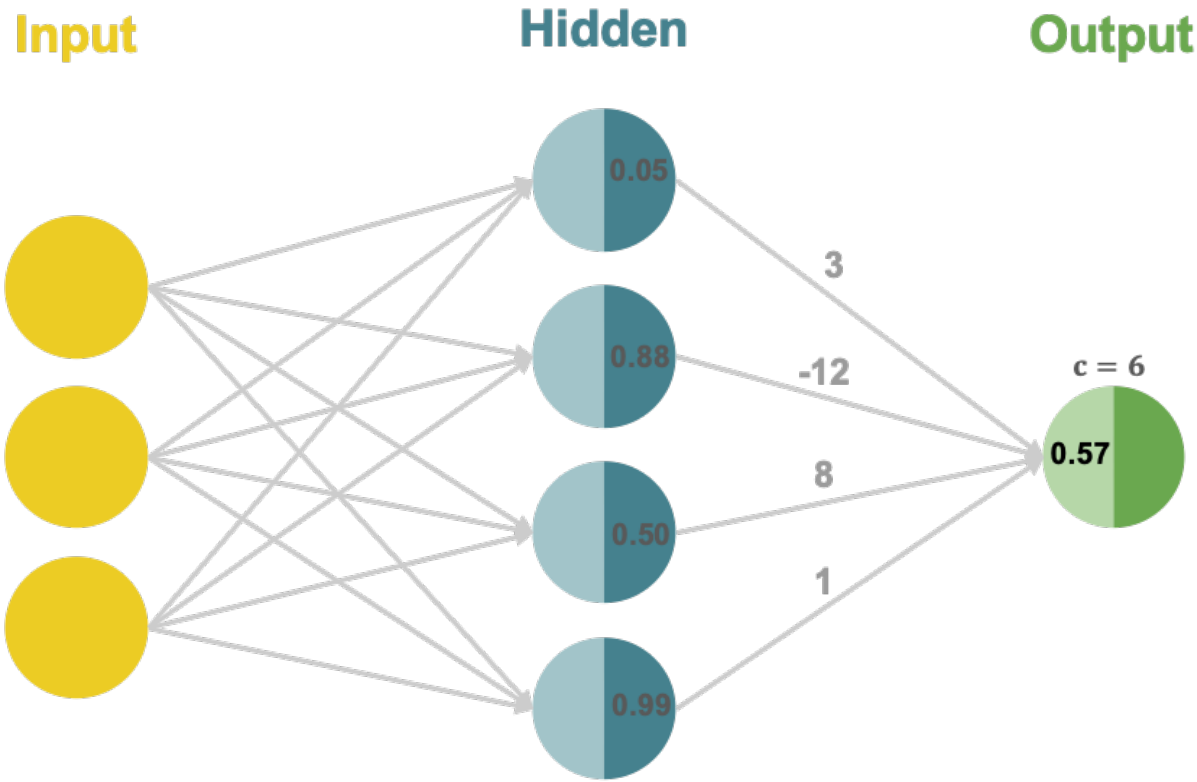
The output neuron performs an **affine transformation** on its inputs:



$$\mathbf{f_{in}} = \mathbf{u_1 z_{out}^{(1)}} + \mathbf{u_2 z_{out}^{(2)}} + \mathbf{u_3 z_{out}^{(3)}} + \mathbf{u_4 z_{out}^{(4)}} + \mathbf{c}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs an **affine transformation** on its inputs:

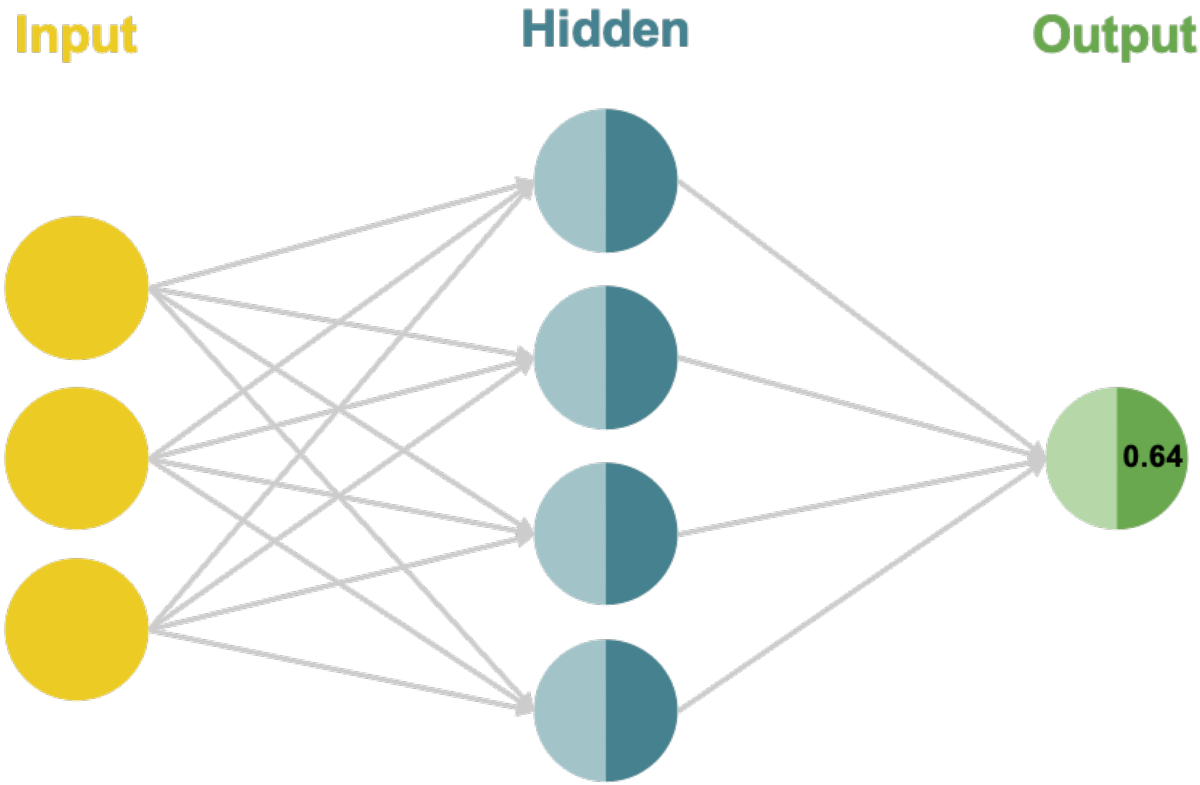


$$\mathbf{f_{in}} = \mathbf{u_1 z_{out}^{(1)}} + \mathbf{u_2 z_{out}^{(2)}} + \mathbf{u_3 z_{out}^{(3)}} + \mathbf{u_4 z_{out}^{(4)}} + \mathbf{c}$$

$$\mathbf{f_{in}} = \mathbf{3 * 0.05 + (-12) * 0.88 + 8 * 0.50 + 1 * 0.99 + 6} = \mathbf{0.57}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs a non-linear **activation** transformation on the weight sum:



$$\mathbf{f_{out}} = \boldsymbol{\sigma}\,(\mathbf{f_{in}}) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{\,\mathbf{f_{in}}}}$$

$$\mathbf{f_{out}} = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{\mathbf{0.57}}} = \mathbf{0.64}$$

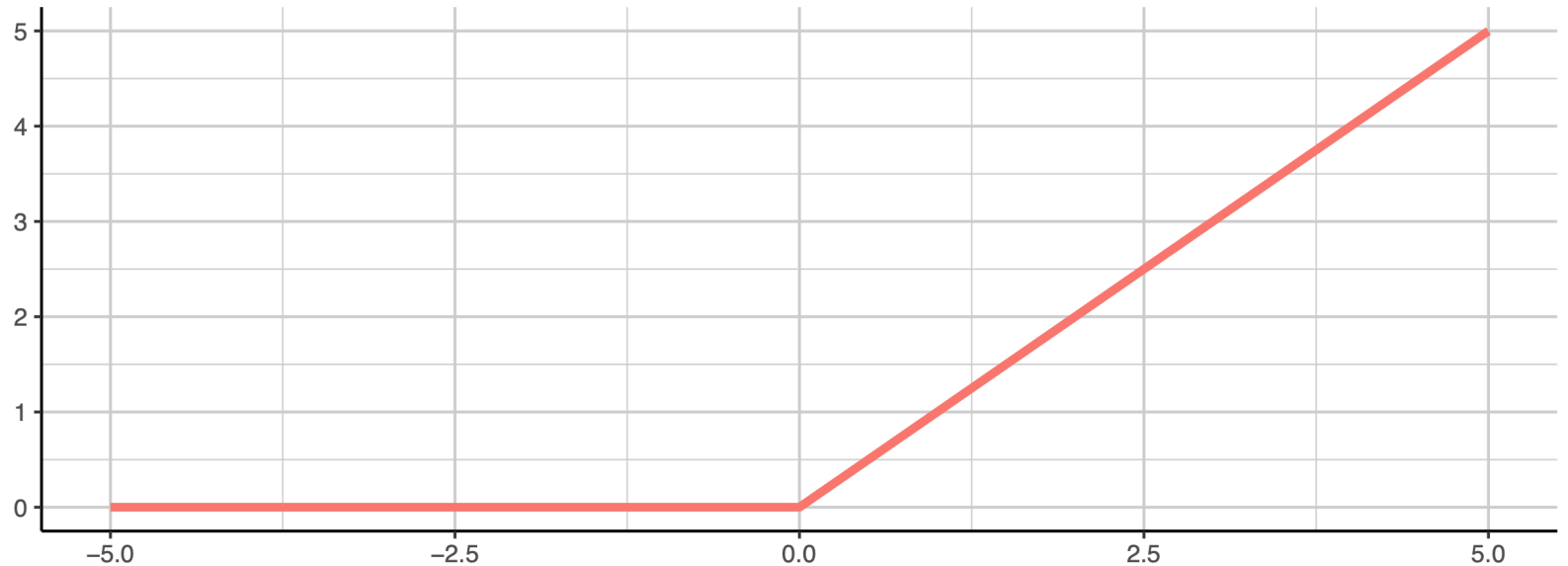
HIDDEN LAYER: ACTIVATION FUNCTION

Note: if the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.

ReLU Activation:

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

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

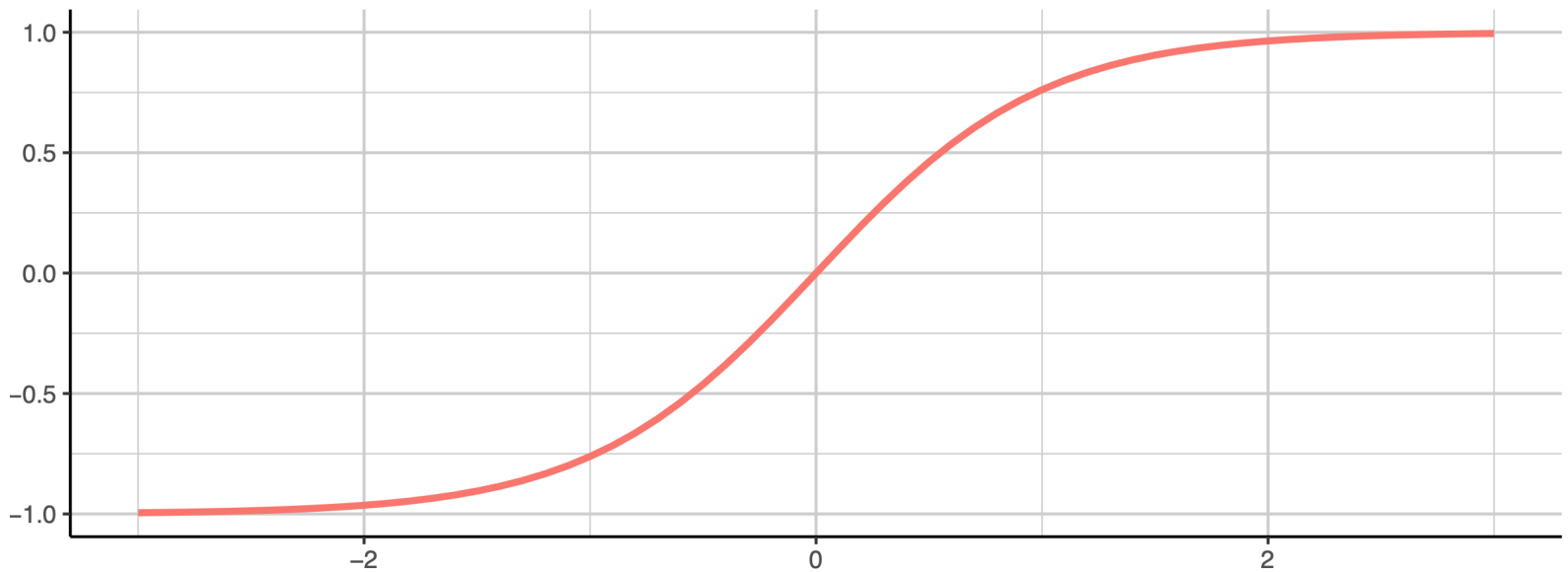


HIDDEN LAYER: ACTIVATION FUNCTION

Hyperbolic Tangent Activation:

- Another choice might be the hyperbolic tangent function:

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

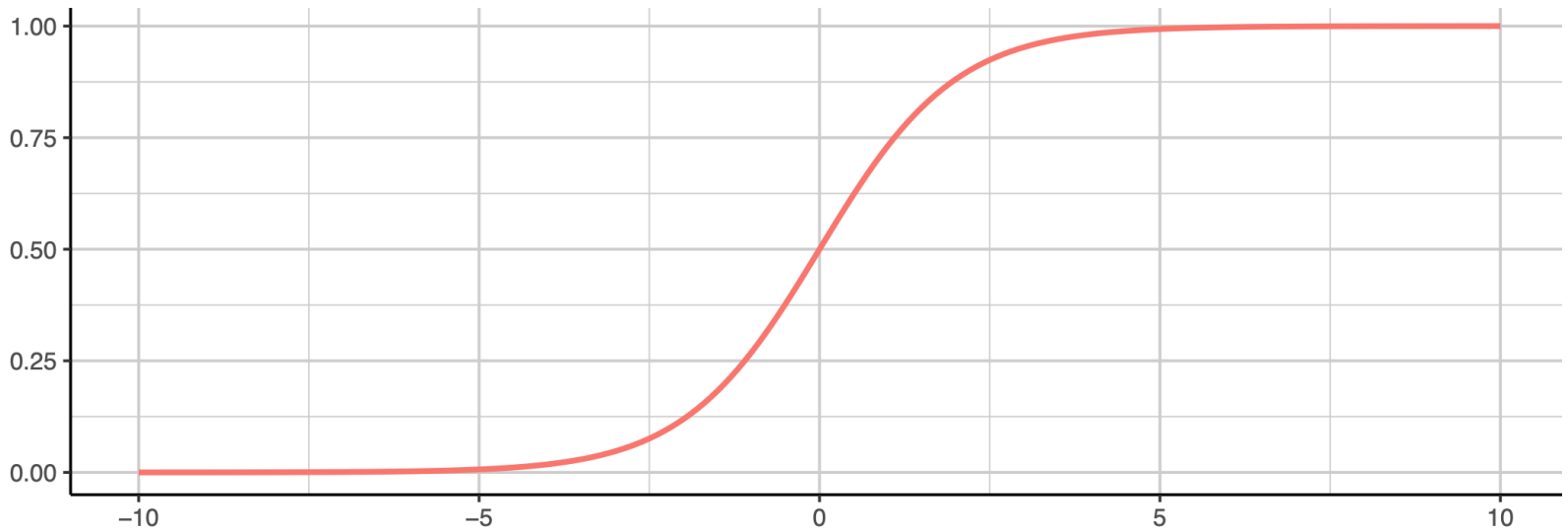


HIDDEN LAYER: ACTIVATION FUNCTION

Sigmoid Activation Function:

- The sigmoid function can be used even in the hidden layer:

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



DEEP FEEDFORWARD NETWORK

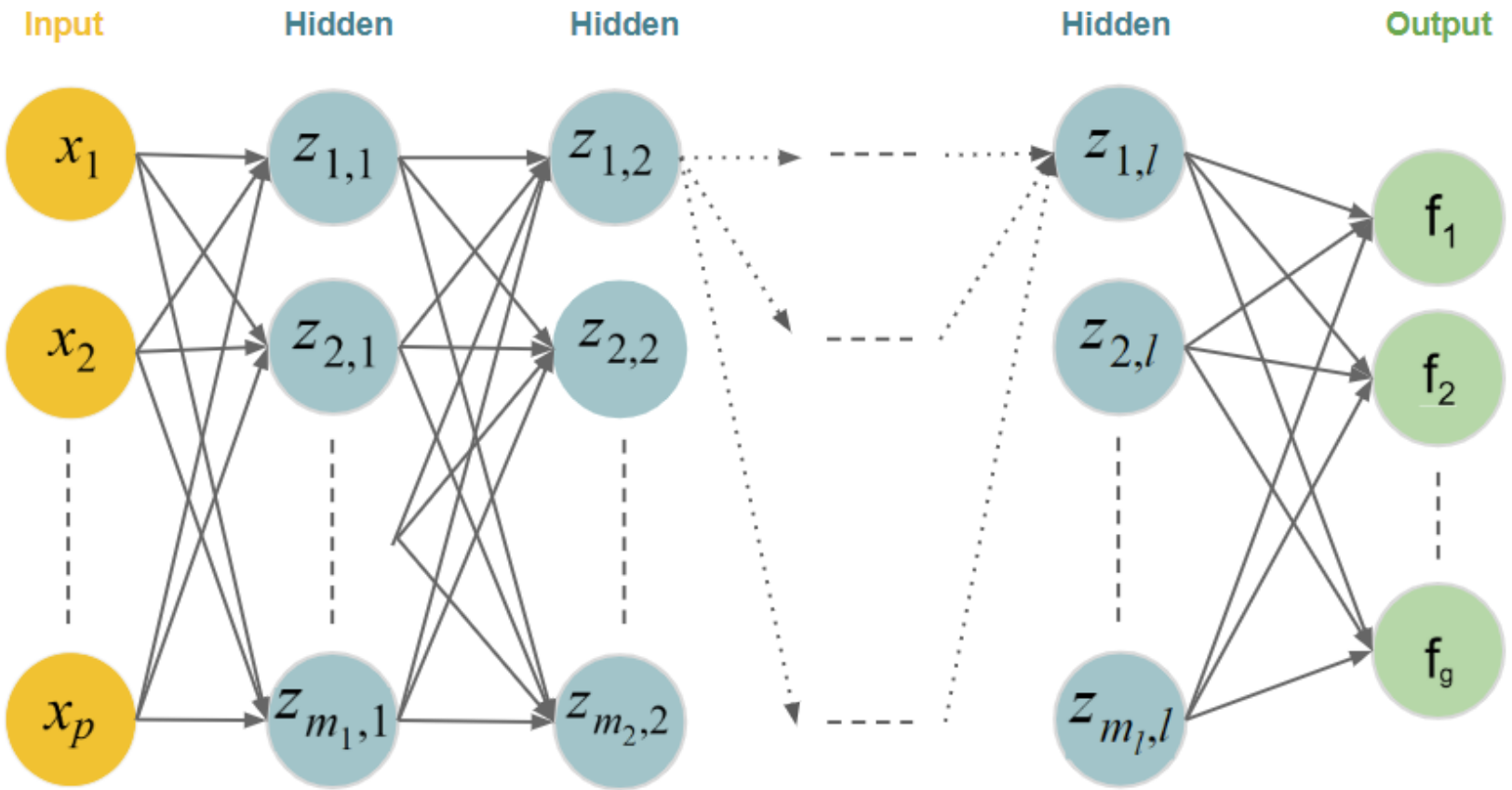


Figure: Structure of a deep neural network with l hidden layers.

WHY ADD MORE LAYERS?

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

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

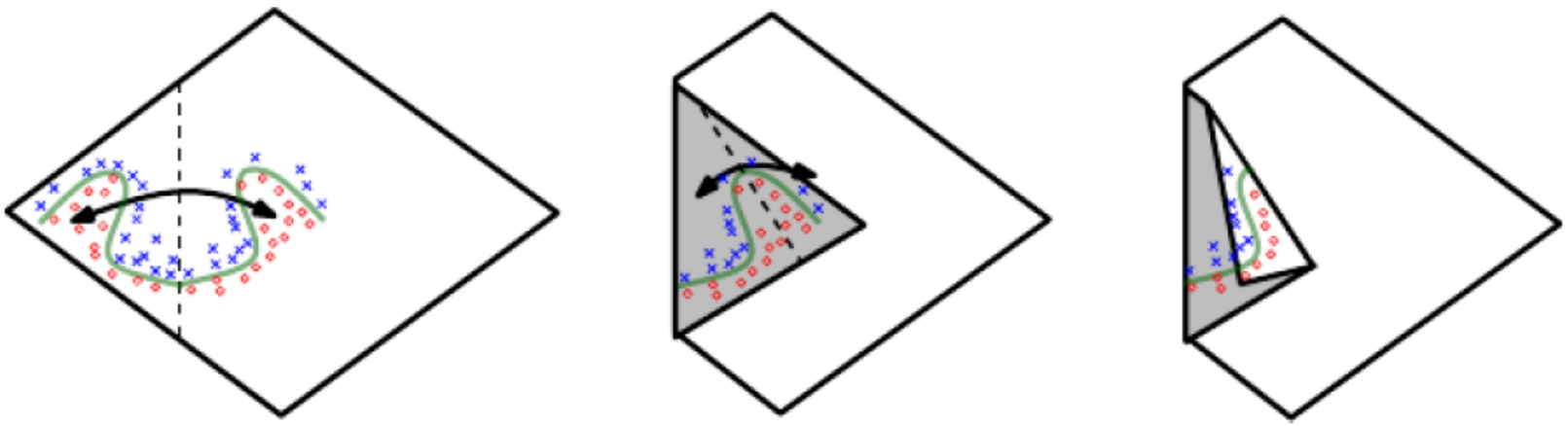


Figure: An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

DEEP 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, however, deep neural networks were very challenging to train for several reasons:

1. For one thing, the use of sigmoid activations (e.g., 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.
2. Training deep neural networks on CPUs was too slow to be practical. Switching over to GPUs cut down training time by more than an order of magnitude.
3. 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

- The availability of large datasets and novel architectures that are capable to handle even complex tensor-shaped data (e.g. CNNs for image data), faster hardware, and 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.

- The term "**deep learning**" encompasses all of these developments and refers to the field as a whole.