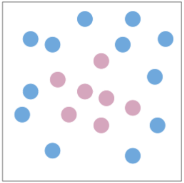
Single hidden layer neural networks

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

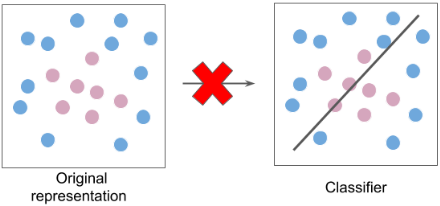
- As a single neuron is restricted to learning only linear decision boundaries, its performance on the following task is quite poor:



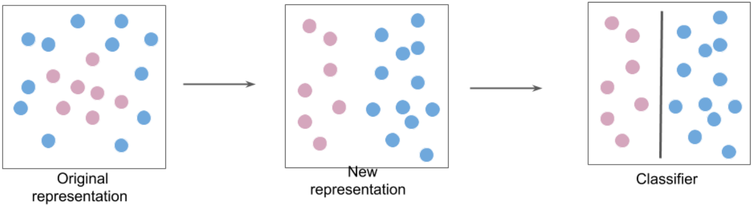
- However, the neuron can easily separate the classes if the original features are transformed (e.g., from Cartesian to polar coordinates):

MOTIVATION

- Instead of classifying the data in the original representation,

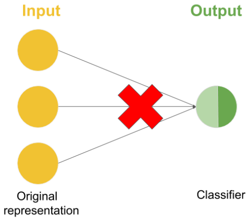


- we classify it in a new feature space.

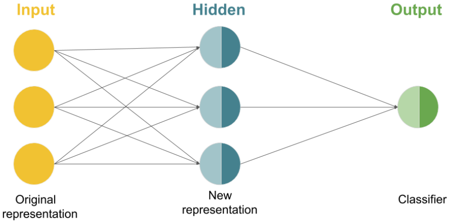


MOTIVATION

- Analogously, instead of a single neuron,



- we use more complex networks.



REPRESENTATION LEARNING

- It is *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**.

- DL 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.

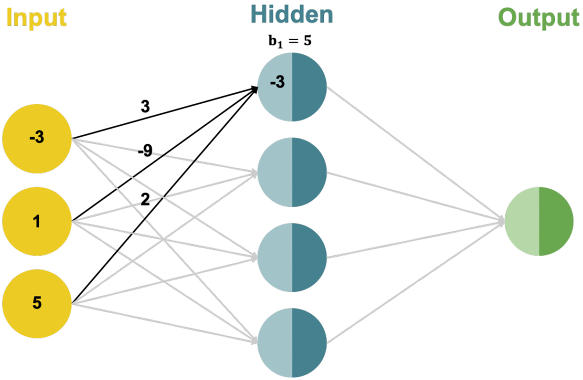
Single hidden layer networks consist of two layers (without input layer):

1. **Hidden Layer:** having a set of neurons.
2. **Output Layer:** having one or more output neurons.

- 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

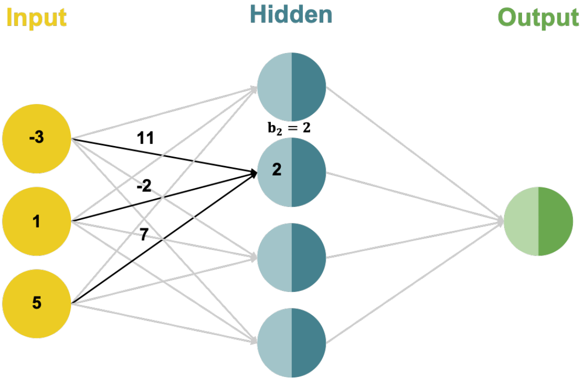
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{in}^{(1)} = w_{11}x^{(1)} + w_{21}x^{(2)} + w_{31}x^{(3)} + b_1$$
$$z_{in}^{(1)} = 3 * (-3) + (-9) * 1 + 2 * 5 + 5 = -3$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

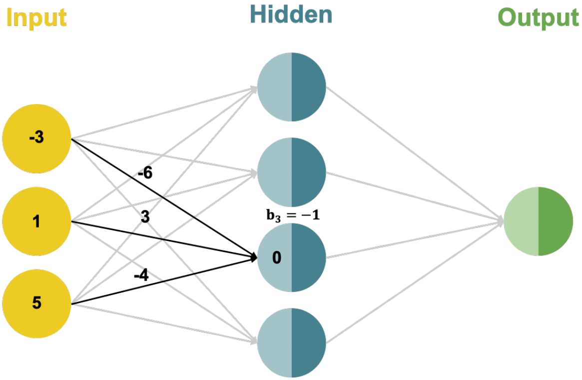
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(2)} = w_{12}x^{(1)} + w_{22}x^{(2)} + w_{32}x^{(3)} + b_2$$
$$z_{\text{in}}^{(2)} = 11 * (-3) + (-2) * 1 + 7 * 5 + 2 = 2$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

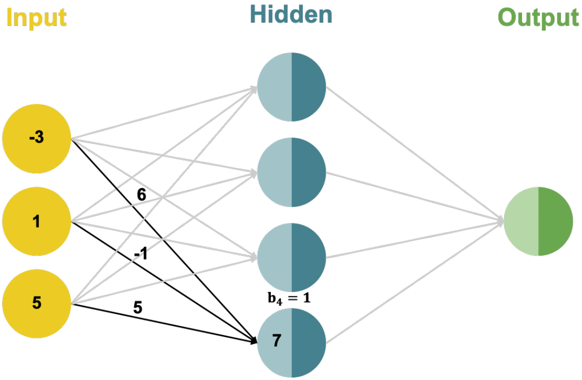
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(3)} = w_{13}x^{(1)} + w_{23}x^{(2)} + w_{33}x^{(3)} + b_3$$
$$z_{\text{in}}^{(3)} = (-6) * (-3) + 3 * 1 + (-4) * 5 - 1 = 0$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

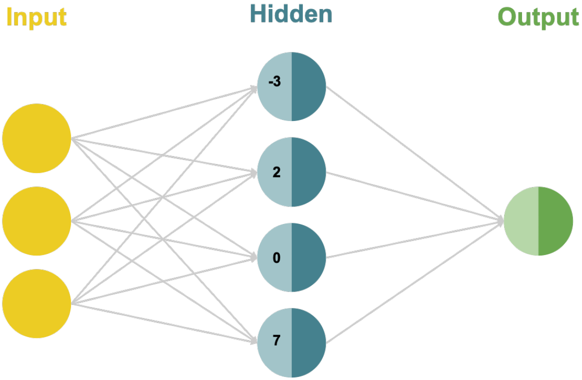
Each neuron in the hidden layer performs an **affine transformation** on the inputs:



$$z_{\text{in}}^{(4)} = w_{14}x^{(1)} + w_{24}x^{(2)} + w_{34}x^{(3)} + b_4$$
$$z_{\text{in}}^{(4)} = 6 * (-3) + (-1) * 1 + 5 * 5 + 1 = 7$$

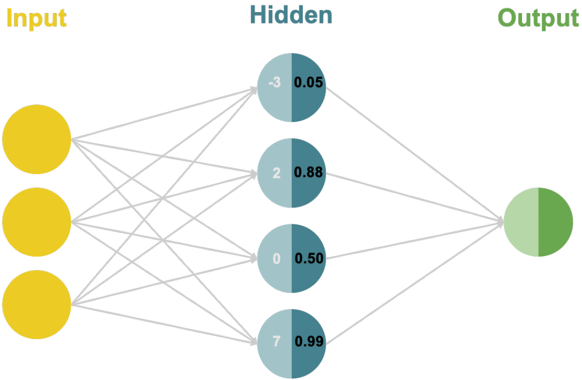
SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

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



SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

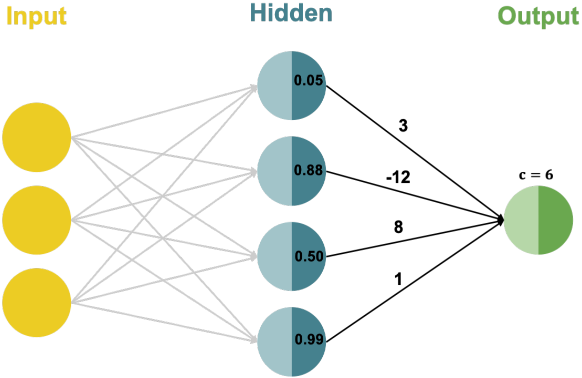
Each hidden neuron performs a non-linear **activation** transformation on the weight sum:



$$z_{\text{out}}^{(i)} = \sigma\left(z_{\text{in}}^{(i)}\right) = \frac{1}{1+e^{z_{\text{in}}^{(i)}}}$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

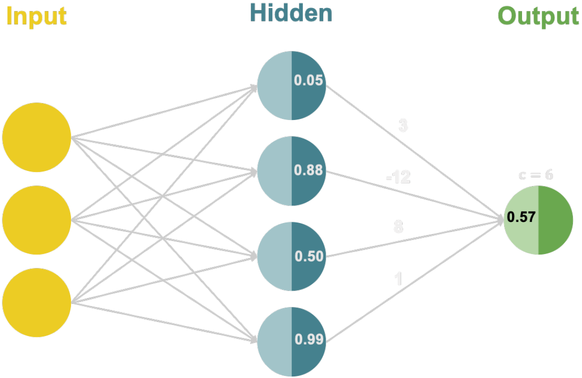
The output neuron performs an **affine transformation** on its inputs:



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

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

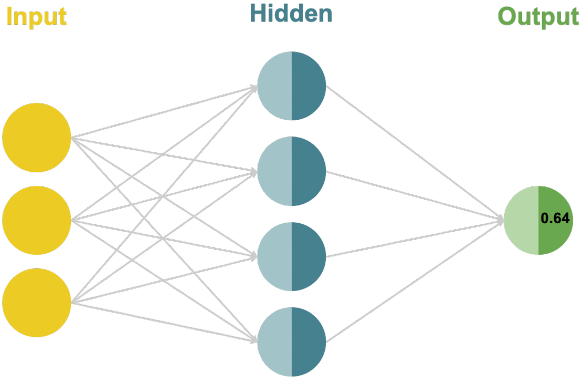
The output neuron performs an **affine transformation** on its inputs:



$$f_{\text{in}} = u_1 z_{\text{out}}^{(1)} + u_2 z_{\text{out}}^{(2)} + u_3 z_{\text{out}}^{(3)} + u_4 z_{\text{out}}^{(4)} + c$$
$$f_{\text{in}} = 3 * 0.05 + (-12) * 0.88 + 8 * 0.50 + 1 * 0.99 + 6 = 0.57$$

SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

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



$$f_{\text{out}} = \sigma(f_{\text{in}}) = \frac{1}{1+e^{f_{\text{in}}}}$$
$$f_{\text{out}} = \frac{1}{1+e^{0.57}} = 0.64$$

HIDDEN LAYER: ACTIVATION FUNCTION

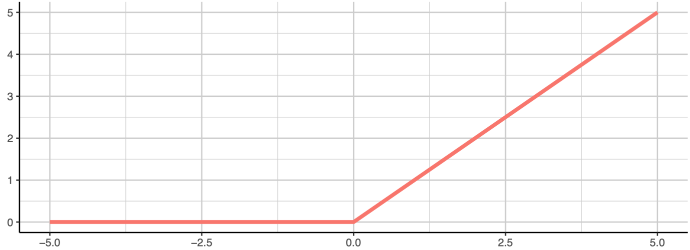
- If the hidden layer does not have a non-linear activation, the network can only learn linear decision boundaries.
- A lot of different activation functions exist.

HIDDEN LAYER: ACTIVATION FUNCTION

ReLU Activation:

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

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

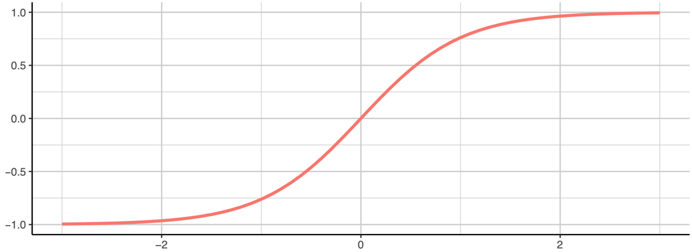


HIDDEN LAYER: ACTIVATION FUNCTION

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:

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