CSCE 633: Machine Learning

Lecture 27: Neural Networks

Texas A&M University

Outline

- Perceptron
- Approximating linear functions
- Activation Function
- Backpropagation
- Optimization
- Neural Network Training and Design

Neural networks: Original motivation

- Inspiration from the brain
  - Brain is a powerful information processing device
  - Composed of a large number of processing units (neurons)
  - Neurons operating in parallel → large connectivity
  - Neural networks as a paradigm for parallel processing

$$f^{(1)}(\mathbf{x}) = a\left(w_0^{(1)} + \sum_{n=1}^{N} w_n^{(1)} x_n\right)$$

Single Layer NN

Recursive rescipe for single layer perceptron units

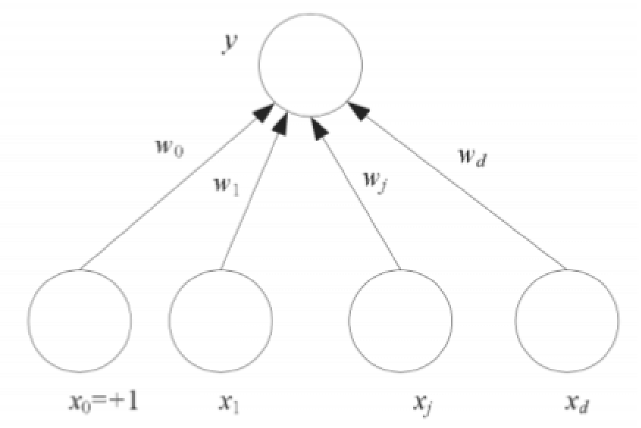
1: **input:** Activation function $a(\cdot)$

2: Compute linear combination: $v = w_0^{(1)} + \sum_{n=1}^{N} w_n^{(1)} x_n$

3: Pass result through activation: $a(v)$

4: **output:** Single layer unit $a(v)$

Perceptron



- Each input has an associated weight (synaptic weight)
- $y = a(\sum_{j=1}^{D} w_d x_d + w_0)$ where w_0 intercept makes the model more general - modeled as the weight coming from an extra bias unit (x_0) which is always +1.
- So, what is the learning procedure here?
- $y = a(\sum_{j=1}^{D} w_d x_d + w_0)$ defines a hyperplane - so we can create a linear discriminant function to make decisions on classes.
- Unlike SVM - we can also get posterior probability using sigmoid as the output (like logistic regression).

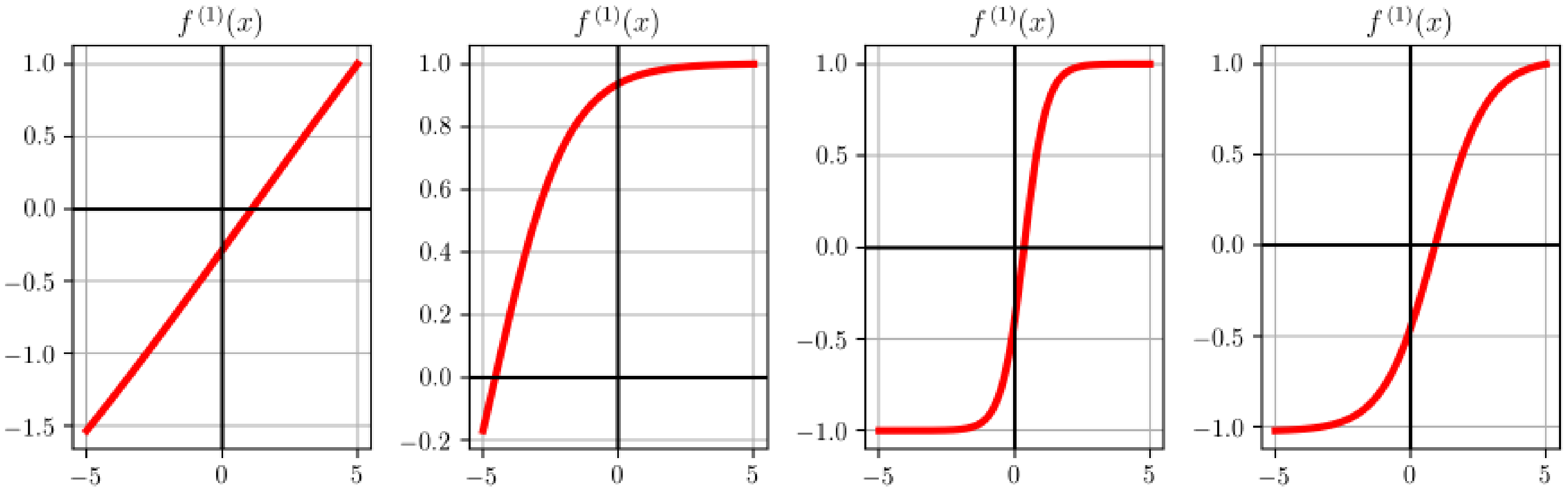
Activation Functions

Example 1. Illustrating the capacity of single layer units

Below we plot four instances of a single-layer unit using $\tanh$ as nonlinear activation function. These take the form

$$f^{(1)}(x) = \tanh\left(w_0^{(1)} + w_1^{(1)} x\right) \tag{2}$$

In each instance the internal parameters have been set randomly, giving each basis function a distinct shape.

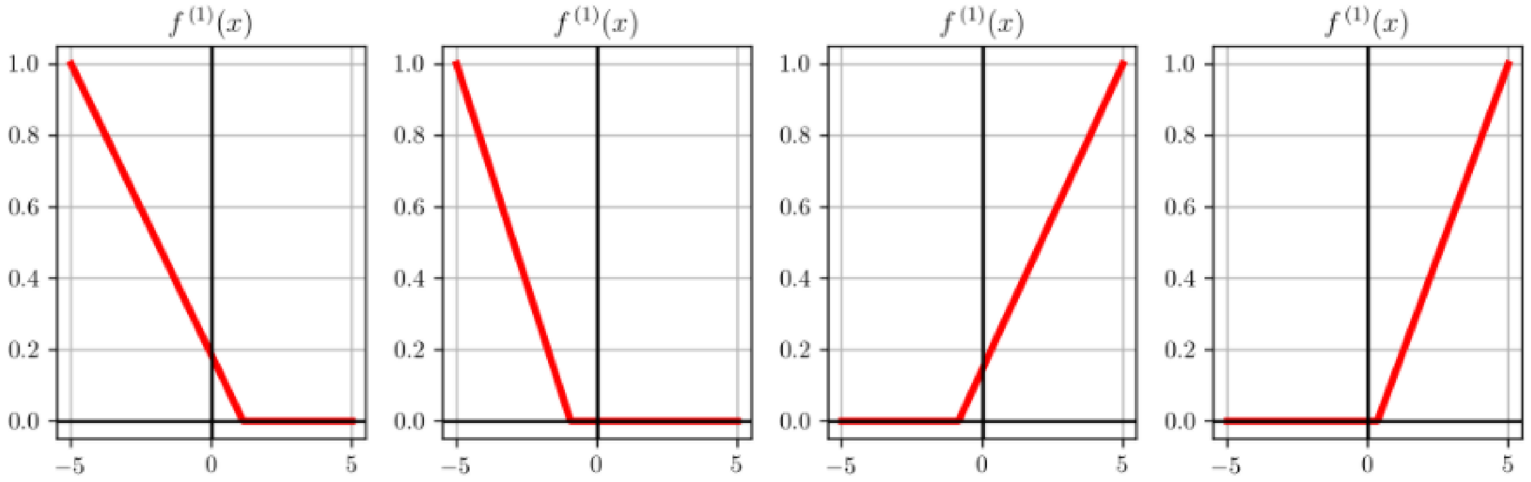


More Activation Functions

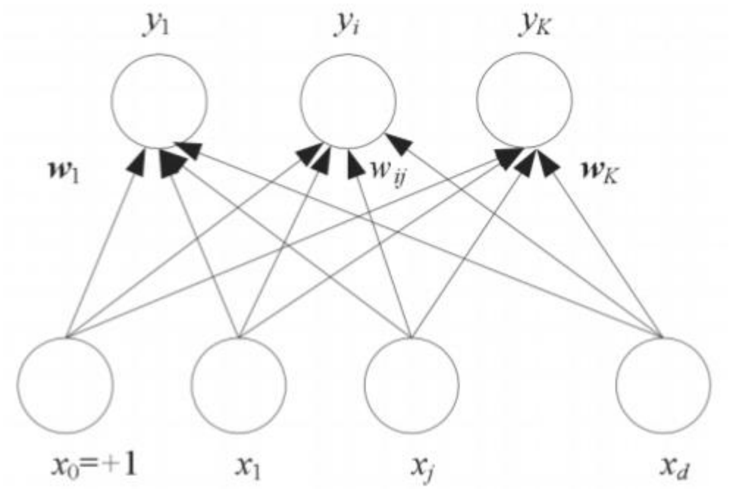
We now repeat this experiment swapping out $\tanh$ for the ReLU function, forming a single hidden layer unit with ReLU activation of the form

$$f^{(1)}(x) = \max\left(0, w_0^{(1)} + w_1^{(1)} x\right) \tag{3}$$

Once again the internal parameters of this unit allow it to take on a variety of shapes. Below have show four instances of this ReLU single layer unit, where in each instance the unit's internal parameters are set at random.

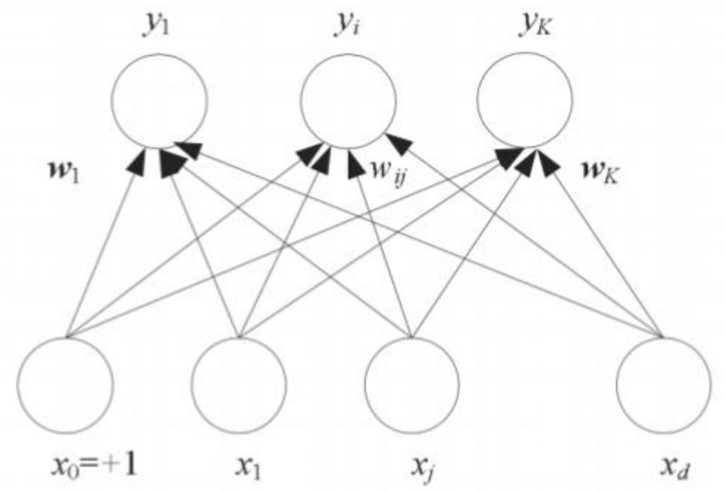


Perceptron: Multiple Classes



- For K classes, create K perceptrons.
- Choose class C_i if $y_i = \max_k y_k$
- If we need probabilities, $o_i = w_i^T x$ which yields $y_i = \frac{\exp o_i}{\sum_k \exp o_k}$ called the softmax values.

Perceptron: Multiple Classes



- Multiclass: $K > 2$ outputs
  - $y_k = a(\sum_{j=1}^{D} w_{kd}x_d + w_{k0}) = a(\boldsymbol{w_k^T x})$

    where w_{kd} is the weight from input x_j to output y_k

    e.g. $a(\mathbf{x}, \mathbf{w_1}, \dots, \mathbf{w_K}) = \frac{\exp(\boldsymbol{w_k^T x})}{\sum_{k=1} \exp(\boldsymbol{w_k^T x})}$
  - 0/1 encoding for output vector
    - e.g. in a 4-class problem: if class=3, then y = [0, 0, 1, 0]

Perceptron: Training

Online training

- Cost-efficient (computationally and memory-wise)
- Nature of data can change over time
- Error function expressed in terms of individual samples
- Weight update performed after each instance is seen

Perceptron: Training

Online training

- Evaluation: cross-entropy function for 1 instance $(\mathbf{x_n}, y_n)$
  $\mathcal{E}(\mathbf{w}) = -y_n \log\left[\sigma(\mathbf{w}^T\mathbf{x_n})\right] - (1 - y_n)\log\left[1 - \sigma(\mathbf{w}^T\mathbf{x_n})\right]$
  $\mathcal{E}(\mathbf{w_1}, \ldots, \mathbf{w_K}) = -\sum_{k=1}^{K} y_{nk} \log p(y_{nk} = 1|\mathbf{w_1}, \ldots \mathbf{w_K})$
- Optimization: gradient descent
  $\frac{\vartheta\mathcal{E}(\mathbf{w})}{\vartheta w_d} = \left(\sigma(\mathbf{w}^T\mathbf{x_n}) - y_n\right) x_{nd}$
  $\frac{\vartheta\mathcal{E}(\mathbf{w})}{\vartheta w_{kd}} = \left(\sigma(\mathbf{w}^T\mathbf{x_n}) - y_{nk}\right) x_{nd}$

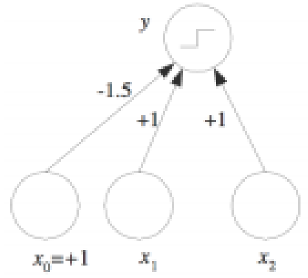
Outline

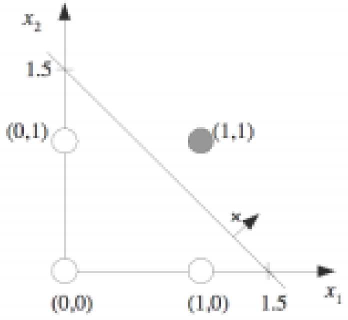
- Perceptron
- Approximating linear functions
- Activation Function
- Backpropagation
- Optimization
- Neural Network Training and Design

Approximating linear functions

Example: Boolean AND

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





Example of a perceptron implementing AND

$y = s(x_1 + x_2 - 1.5)$

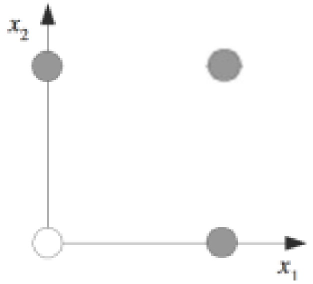
$\mathbf{w} = [-1.5\ 1\ 1]^T$

$\mathbf{x} = [1\ x_1\ x_2]^T$

Approximating linear functions

Example: Boolean OR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	1



Example of a perceptron implementing OR

$y = s(x_1 + x_2 - 0.5)$

$\mathbf{w} = [-0.5\ 1\ 1]^T$

$\mathbf{x} = [1\ x_1\ x_2]^T$

Approximating linear functions

Example: Boolean NOT

x_1	r
0	1
1	0

Example of a perceptron implementing NOT

$y = ?$

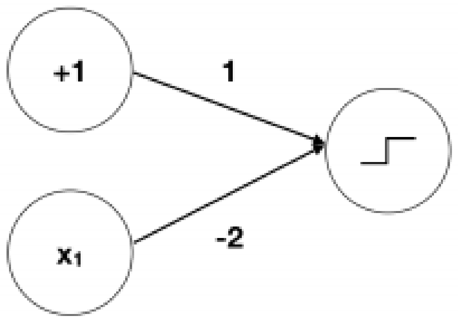
$\mathbf{w} = ?$

$\mathbf{x} = [1\ x_1]^T$

Approximating linear functions

Example: Boolean NOT

x_1	r
0	1
1	0



Example of a perceptron implementing OR

$y = s(x_1 - 2)$

$\mathbf{w} = [1 \;\; -2]^T$

$\mathbf{x} = [1 \; x_1]^T$

Approximating linear functions

Example: Boolean (NOT x_1) AND (NOT x_2)

x_1	x_2	r
0	0	1
0	1	0
1	0	0
1	1	0

Example of a perceptron implementing OR

$y = ?$

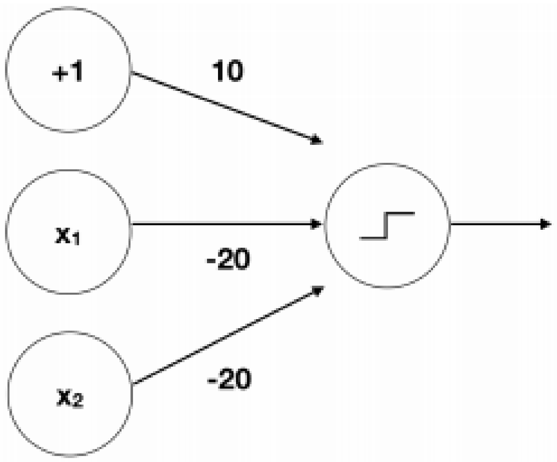
$\mathbf{w} = ?$

$\mathbf{x} = [1\ x_1\ x_2]^T$

Approximating linear functions

Example: Boolean (NOT x_1) AND (NOT x_2)

x_1	x_2	r
0	0	1
0	1	0
1	0	0
1	1	0



Example of a perceptron implementing OR

$y = s(-20x_1 - 20x_2 + 10)$

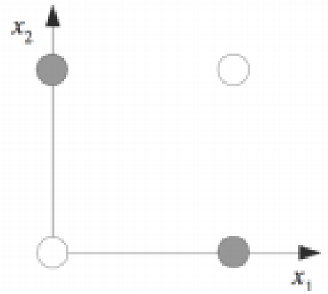
$\mathbf{w} = [10 \; -20 \; -20]^T$

$\mathbf{x} = [1 \; x_1 \; x_2]^T$

Approximating linear functions

Example: Boolean XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



Example of a perceptron implementing OR

$y = ?$

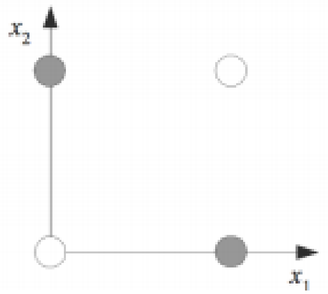
$\mathbf{w} = ?$

$\mathbf{x} = [1\ x_1\ x_2]^T$

Approximating linear functions

Example: Boolean XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



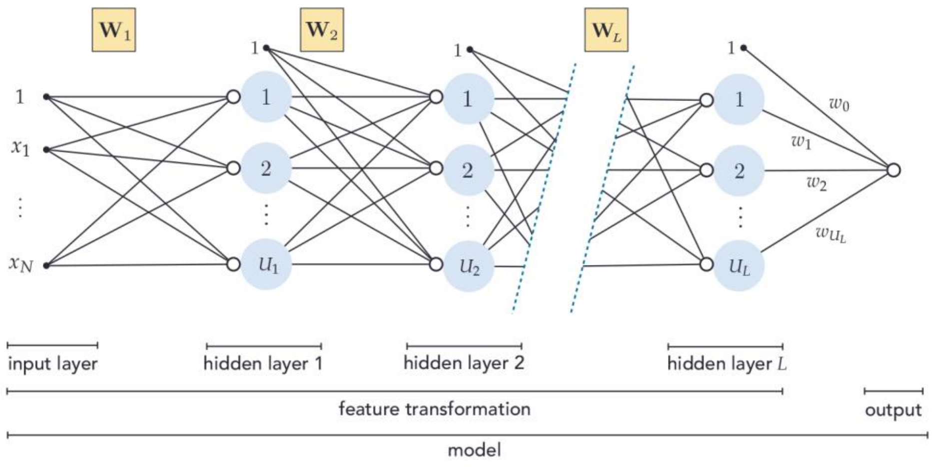
Not linearly separable

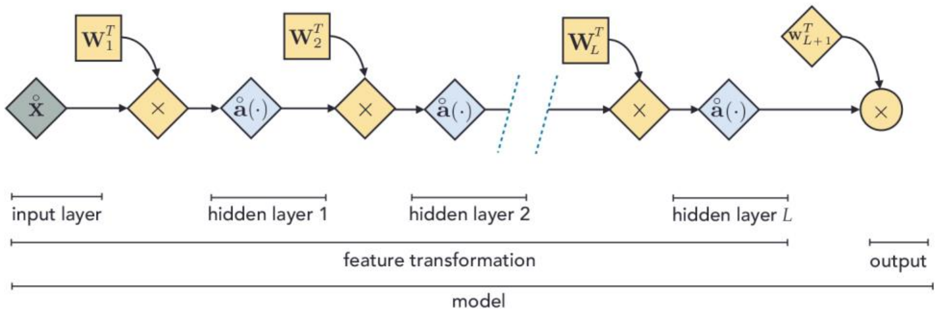
Need combination of more than one perceptrons $\rightarrow$ multilayer perceptrons

Outline

- Perceptron
- Approximating linear functions
- Activation Function
- Backpropagation
- Optimization
- Neural Network Training and Design

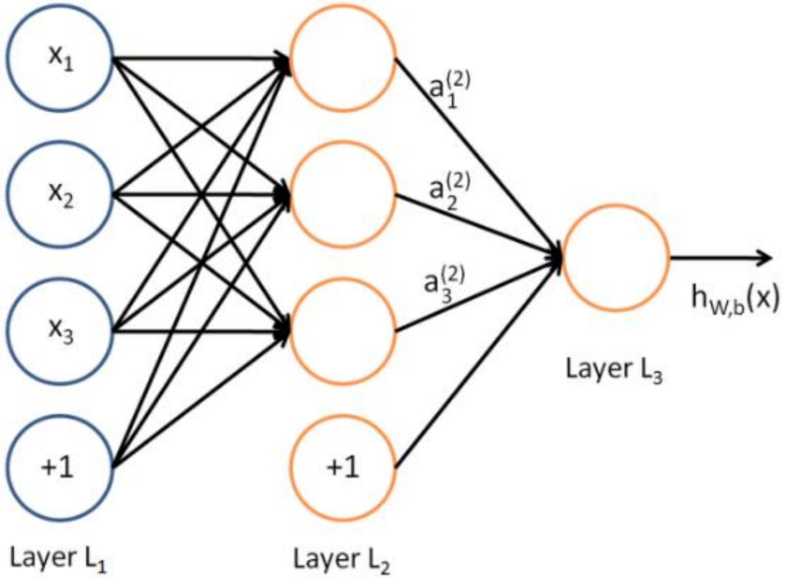
Can create complex functions with Deep Neural Networks



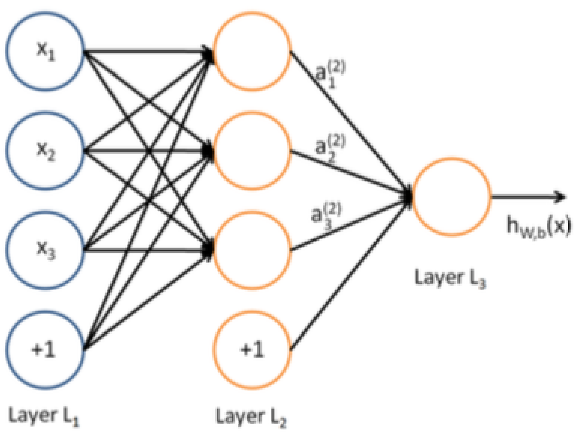


Multilayer Perceptron

- Type of feedforward neural network
- Can model non-linear associations
- "Multi-level combination" of many perceptrons



Multilayer Perceptron: Representation



$$\alpha_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$\alpha_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$\alpha_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \alpha_1^{(3)} = f(W_{11}^{(2)}\alpha_1^{(2)} + W_{12}^{(2)}\alpha_2^{(2)} + W_{13}^{(2)}\alpha_3^{(2)} + b_1^{(2)})$$

Terminology

$W_{ij}^{(l)}$: connection between unit j in layer l to unit i in layer $l+1$

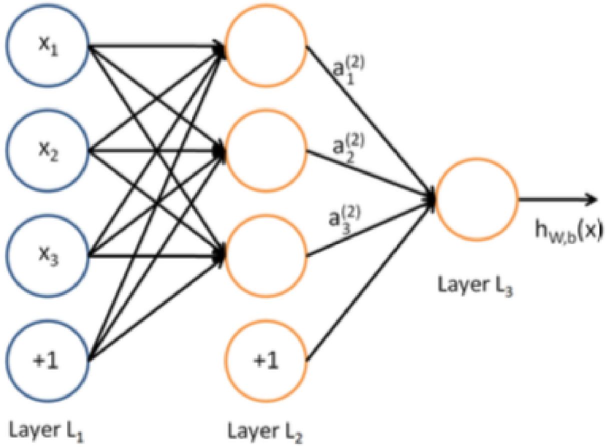
$\alpha_i^{(l)}$: activation of unit i in layer l

$b_i^{(l)}$: bias connected with unit i in layer $l+1$

Forward propagation: The process of propagating the input to the output through the activation of inputs and hidden units to each node

Multilayer Perceptron: Representation

Matrix notation



$$\alpha^{(2)} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \alpha^{(3)} = f(\mathbf{W}^{(2)}\alpha^{(2)} + \mathbf{b}^{(2)})$$

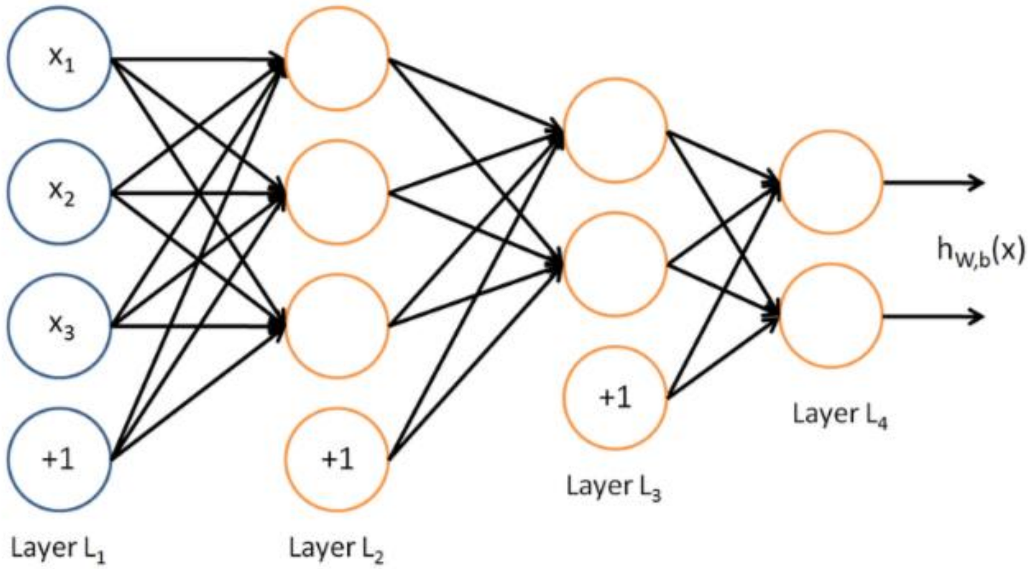
$$\mathbf{W}^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} & W_{33}^{(1)} \end{bmatrix}, \ \mathbf{b}^{(1)} = [b_1^{(1)} \ b_2^{(1)} \ b_3^{(1)}], \text{ etc.}$$

Multilayer Perceptron: Representation

Alternative architectures

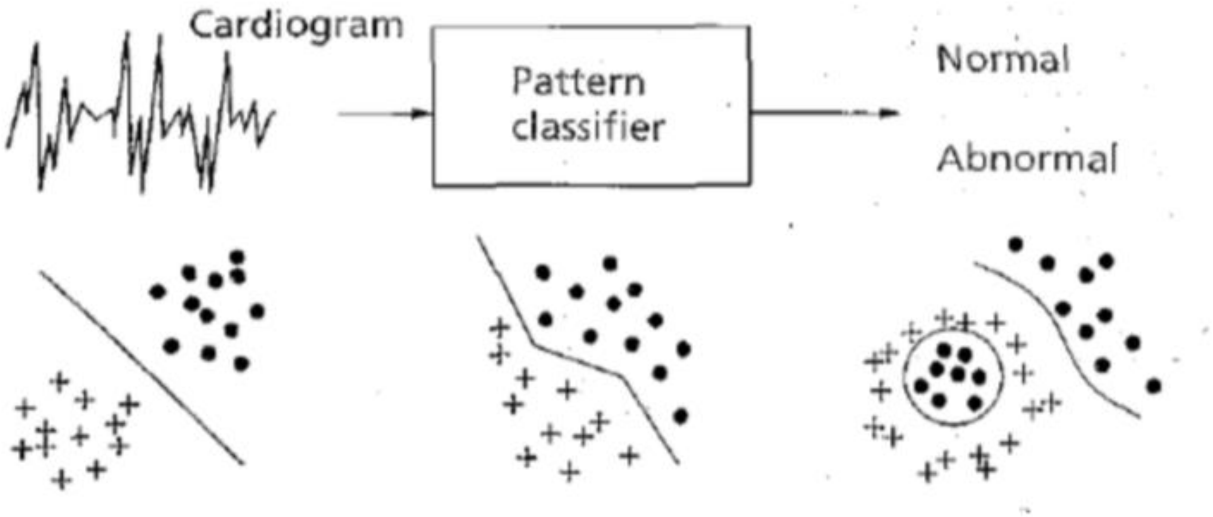
2 hidden layers, multiple output units

e.g. medical diagnosis: different outputs might indicate presence or absence of different diseases



Multilayer Perceptron

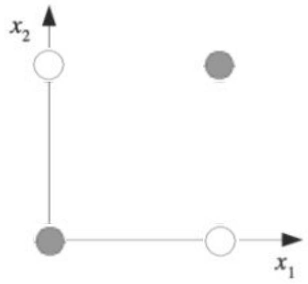
Non-linear feature learning

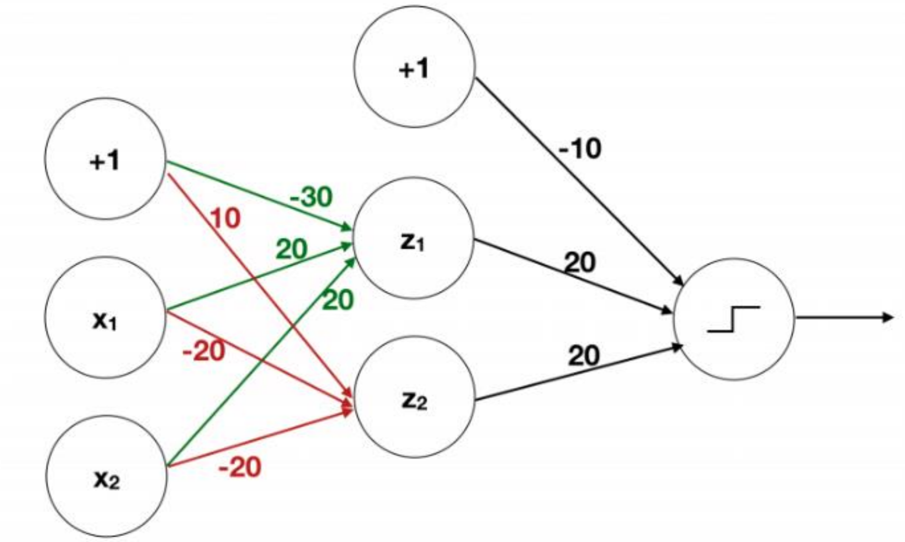


Multilayer Perceptron: Approximating non-linear functions

Example: Boolean XNOR multilayer perceptrons

x_1	x_2	z_1	z_2	r
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1





Outline

- Perceptron
- Approximating linear functions
- Activation Function
- Backpropagation
- Optimization
- Neural Network Training and Design

Activation Function

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.

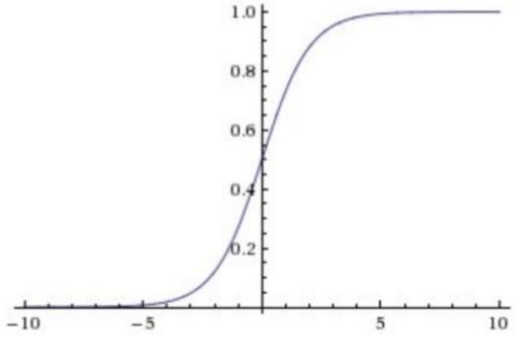
Purpose: introduce non-linearity into the output of a neuron.

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: $s(x) = \tanh(x) = 2\sigma(2x) - 1$
- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$
- Leaky ReLU: $f(x) = (ax) \cdot \mathbb{I}(x < 0) + (x) \cdot \mathbb{I}(x \geq 0)$ (e.g. $a = 0.01$)

Activation Function

Sigmoid: $s(x) = \frac{1}{1+e^{-x}}$

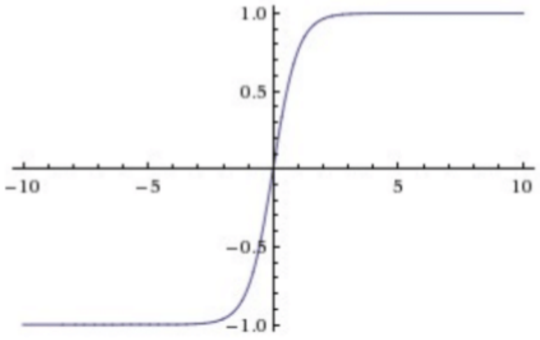
- Transforms a real-valued number between 0 and 1
- Large negative numbers become 0 (not firing at all)
- Large positive numbers become 1 (fully-saturated firing)
- Used historically because of its nice interpretation
- Saturates gradients: The gradient at either extremes (0 or 1) is almost zero, "killing" the signal will flow
- Non-zero centered output: Can be problematic during training, since it can bias outputs toward being always positive or always negative, causing unnecessary oscillations during the optimization



Activation Function

Hyperbolic tangent: $s(x) = \tanh(x) = 2\sigma(2x) - 1$

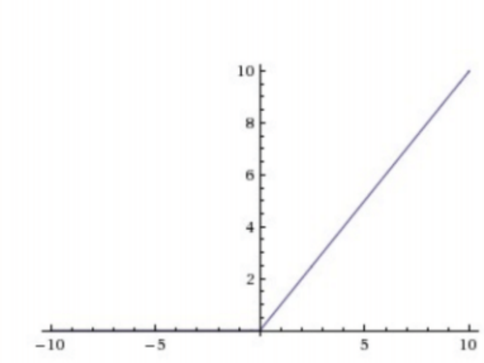
- Scaled version of sigmoid
- Transforms a real-valued number between -1 and 1
- Saturates gradients: Similar to sigmoid
- Output is zero-centered, avoiding some oscillation issues

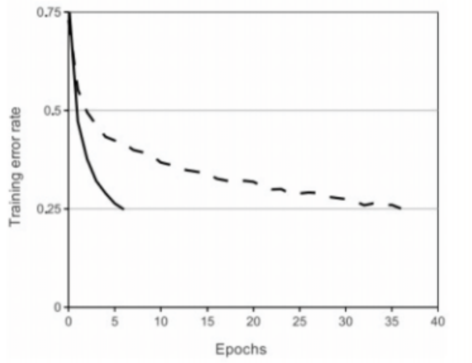


Activation Function

Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$

- Activation simply thresholded at zero
- Very popular during the last years
- Accelerates convergence (e.g. a factor of 6, see bellow) compared to the sigmoid/tanh (due to its linear, non-saturating form)
- Cheap implementation by simply thresholding at zero
- Activation can "die": a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again, proper adjustment of learning rate can mitigate that





Activation Function

Leaky ReLU: $f(x) = (ax) \cdot \mathbb{I}(x < 0) + (x) \cdot \mathbb{I}(x \geq 0)$

- Instead of the function being zero when $x < 0$, leaky ReLU will have a small negative slope (e.g. $a = 0.01$)
- Some successful results, but not always consistent