



THE UNIVERSITY *of* EDINBURGH
informatics

Machine Learning Practical (MLP)

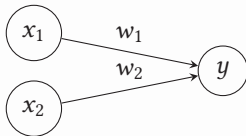
Deep Neural Networks (1)

Changjian Li | Pavlos Andreadis

Limitations of Single Layer Networks

Single layer network

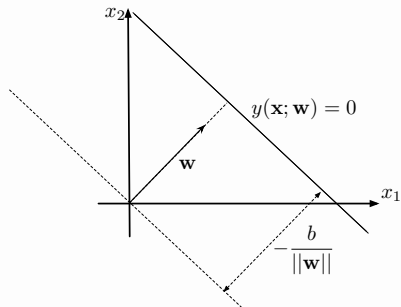
Single-layer network, 1 output, 2 inputs



$$y = x_1 w_1 + b_1 + x_2 w_2 + b_2$$

Geometric interpretation

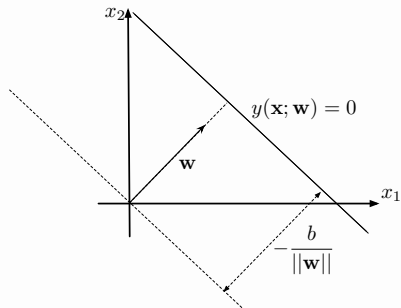
Single-layer network, 1 output, 2 inputs (Bishop, sec 6.1)



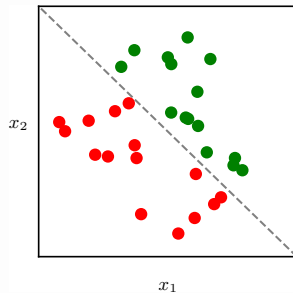
$$y(x; w) = wx + b$$

Geometric interpretation

Single-layer network, 1 output, 2 inputs (Bishop, sec 6.1)

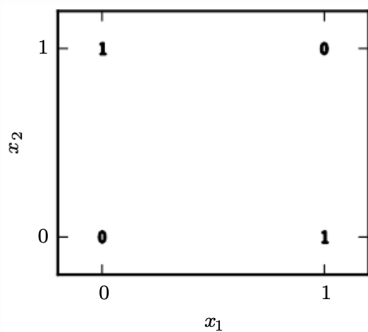


$$y(\mathbf{x}; \mathbf{w}) = \mathbf{w} \mathbf{x} + b$$



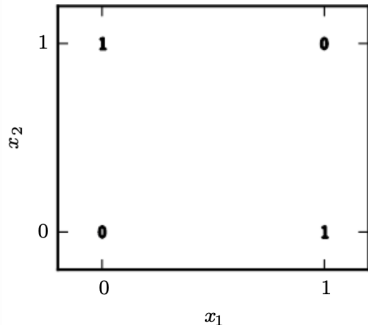
decision boundary

Learning XOR



(Goodfellow DL, fig 6.1)

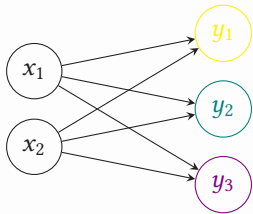
Learning XOR



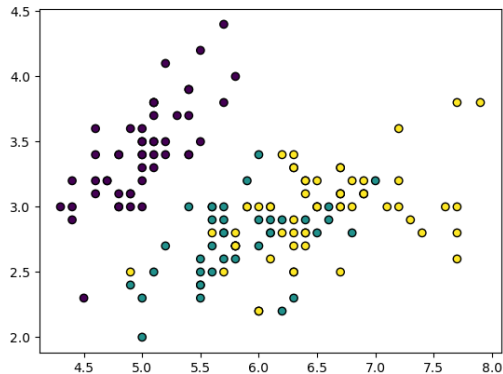
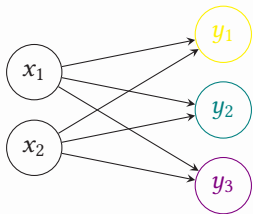
(Goodfellow DL, fig 6.1)

Linear models cannot learn XOR!

Decision boundaries for more classes

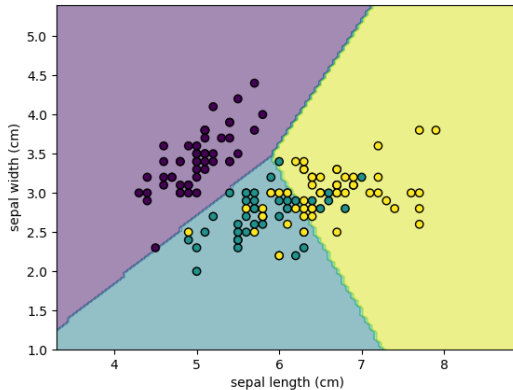
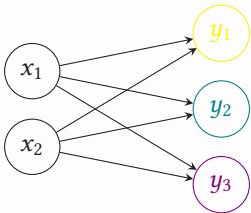


Decision boundaries for more classes



Iris dataset: samples from 3 species of iris flowers

Decision boundaries for more classes



Multi-Layer Networks

Adding layers

A simple example: a linear model with 1 input / output node:

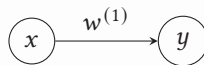
$$y = wx + b$$

Adding layers

A simple example: a linear model with 1 input / output node:

$$y = w^{(1)}x + b^{(1)}$$

$^{(1)}$ means layer 1



Adding layers

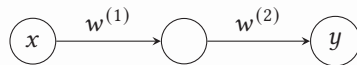
A simple example: a linear model with 1 input / output node:

$$y = w^{(1)}x + b^{(1)}$$

⁽¹⁾ means layer 1

Let's add a hidden (intermediate) layer:

$$y = w^{(2)} \left(w^{(1)}x + b^{(1)} \right) + b^{(2)}$$



Adding layers

A simple example: a linear model with 1 input / output node:

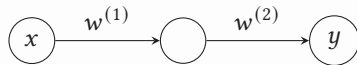
$$y = w^{(1)}x + b^{(1)}$$

⁽¹⁾ means layer 1

Let's add a hidden (intermediate) layer:

$$y = w^{(2)} \left(w^{(1)}x + b^{(1)} \right) + b^{(2)}$$

Yay! Now we have a multi-layer network!



Can we now learn XOR?

Adding layers

A simple example: a linear model with 1 input / output node:

$$y = w^{(1)}x + b^{(1)}$$

⁽¹⁾ means layer 1

Let's add a hidden (intermediate) layer:

$$y = w^{(2)} \left(w^{(1)}x + b^{(1)} \right) + b^{(2)}$$

Magic ingredient: non-linear activation functions **a**

$$y = w^{(2)} \underbrace{a \left(w^{(1)}x + b^{(1)} \right)}_h + b^{(2)}$$



Adding layers

A simple example: a linear model with 1 input / output node:

$$y = w^{(1)}x + b^{(1)}$$

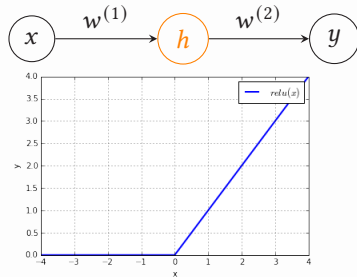
⁽¹⁾ means layer 1

Let's add a hidden (intermediate) layer:

$$y = w^{(2)} \left(w^{(1)}x + b^{(1)} \right) + b^{(2)}$$

Magic ingredient: non-linear activation functions **a**

$$y = w^{(2)} \underbrace{a \left(w^{(1)}x + b^{(1)} \right)}_h + b^{(2)}$$



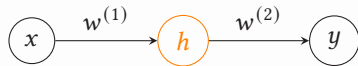
E.g. Rectified Linear Units (ReLU): $a(x) = \max(0, x)$

Hidden layers/units

Let's rewrite $y = w^{(2)} a(w^{(1)}x + b^{(1)}) + b^{(2)}$ as:

$$y = w^{(2)} h + b^{(2)}$$

where $h = a(w^{(1)}x + b^{(1)})$ is the hidden unit



Hidden layers/units

Let's rewrite $y = w^{(2)} a(w^{(1)}x + b^{(1)}) + b^{(2)}$ as:

$$y = w^{(2)} h + b^{(2)}$$

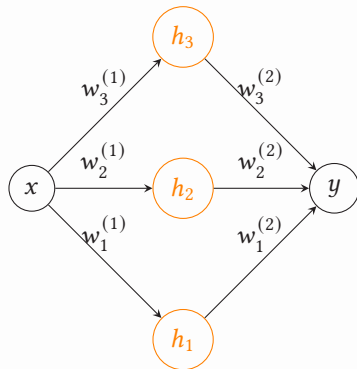
where $h = a(w^{(1)}x + b^{(1)})$ is the hidden unit

Let's add more hidden units:

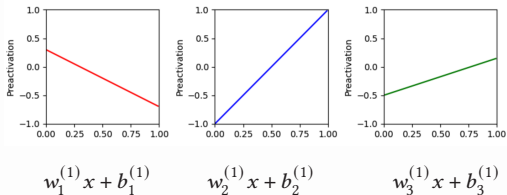
$$y = w_1^{(2)} h_1 + w_2^{(2)} h_2 + w_3^{(2)} h_3 + b^{(2)}$$

where $h_j = a(w_j^{(1)}x + b_j^{(1)})$.

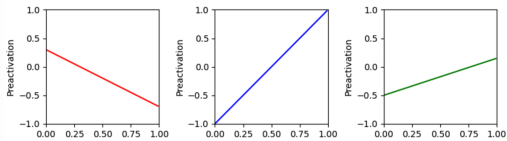
j means unit j



Hidden layers/units



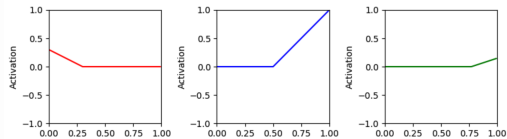
Hidden layers/units



$$w_1^{(1)}x + b_1^{(1)}$$

$$w_2^{(1)}x + b_2^{(1)}$$

$$w_3^{(1)}x + b_3^{(1)}$$

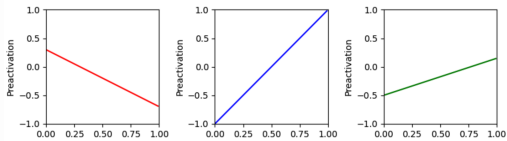


$$a(w_1^{(1)}x + b_1^{(1)})$$
$$h_1$$

$$a(w_2^{(1)}x + b_2^{(1)})$$
$$h_2$$

$$a(w_3^{(1)}x + b_3^{(1)})$$
$$h_3$$

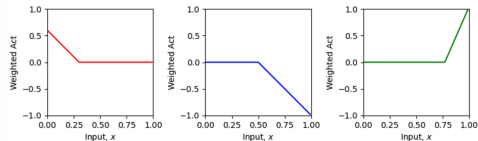
Hidden layers/units



$$w_1^{(1)}x + b_1^{(1)}$$

$$w_2^{(1)}x + b_2^{(1)}$$

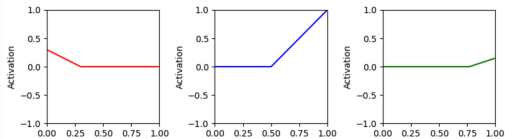
$$w_3^{(1)}x + b_3^{(1)}$$



$$w_1^{(2)}h_1$$

$$w_2^{(2)}h_2$$

$$w_3^{(2)}h_3$$



$$a(w_1^{(1)}x + b_1^{(1)})$$

$$h_1$$

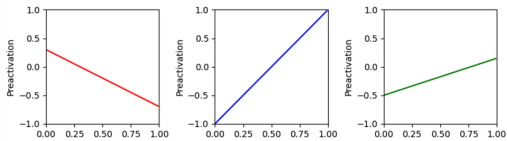
$$a(w_2^{(1)}x + b_2^{(1)})$$

$$h_2$$

$$a(w_3^{(1)}x + b_3^{(1)})$$

$$h_3$$

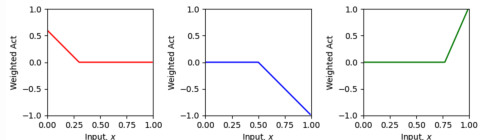
Hidden layers/units



$$w_1^{(1)}x + b_1^{(1)}$$

$$w_2^{(1)}x + b_2^{(1)}$$

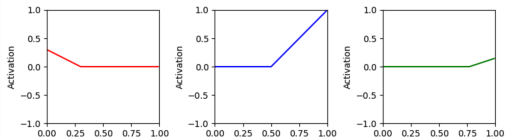
$$w_3^{(1)}x + b_3^{(1)}$$



$$w_1^{(2)}h_1$$

$$w_2^{(2)}h_2$$

$$w_3^{(2)}h_3$$



$$a(w_1^{(1)}x + b_1^{(1)})$$

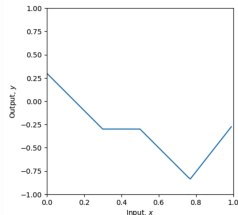
$$h_1$$

$$a(w_2^{(1)}x + b_2^{(1)})$$

$$h_2$$

$$a(w_3^{(1)}x + b_3^{(1)})$$

$$h_3$$



$$y = w_1^{(2)}h_1 + w_2^{(2)}h_2 + w_3^{(2)}h_3 + b^{(2)}$$

Universal Approximation Theory

Let's assume that there are H hidden units $\{h_1, \dots, h_H\}$.

$$y = b^{(2)} + \sum_{j=1}^H w_j^{(2)} h_j$$

H can be seen as a measure of the *capacity* of the network.

The theorem proves that a two layer network with enough capacity (hidden units) can describe any continuous function defined on a compact subset to arbitrary precision.

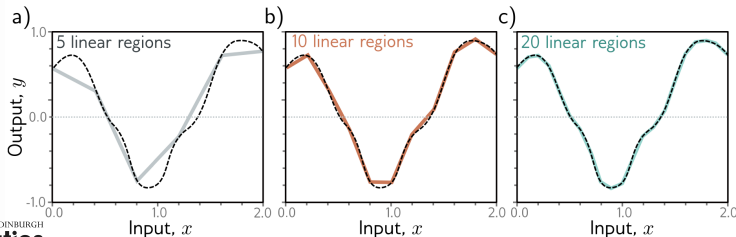
Universal Approximation Theory

Let's assume that there are H hidden units $\{h_1, \dots, h_H\}$.

$$y = b^{(2)} + \sum_{j=1}^H w_j^{(2)} h_j$$

H can be seen as a measure of the *capacity* of the network.

The theorem proves that a two layer network with enough capacity (hidden units) can describe any continuous function defined on a compact subset to arbitrary precision.



Prince UDL Fig 3.5

Universal Approximation Theory

Let's assume that there are H hidden units $\{h_1, \dots, h_H\}$.

$$y = b^{(2)} + \sum_{j=1}^H w_j^{(2)} h_j$$

H can be seen as a measure of the *capacity* of the network.

The theorem proves that a two layer network with enough capacity (hidden units) can describe any continuous function defined on a compact subset to arbitrary precision.

The theorem does not say anything about

- required number of hidden units
- how to learn the weights

Network Depth

Universal approximation theory tells us that we can approximate any function with a single hidden layer. Why do we need more layers?

Network Depth

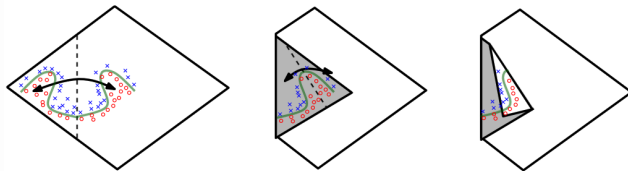
Universal approximation theory tells us that we can approximate any function with a single hidden layer. Why do we need more layers?

- Composition of several simpler functions can represent more complex functions more efficiently
- Greater depth has empirically been shown to lead to better generalization

Network Depth

Universal approximation theory tells us that we can approximate any function with a single hidden layer. Why do we need more layers?

- Composition of several simpler functions can represent more complex functions more efficiently
- Greater depth has empirically been shown to lead to better generalization



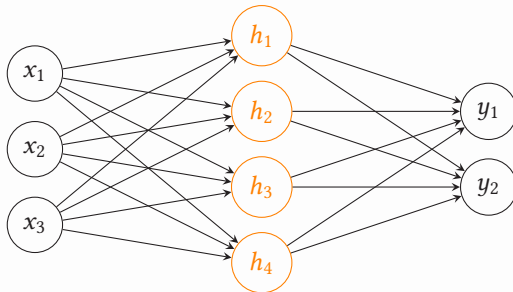
Montufar et al. 2014

Terminology

Input units

Hidden units

Output units



Two layer network with 3 input units, 4 hidden units and 2 output units.

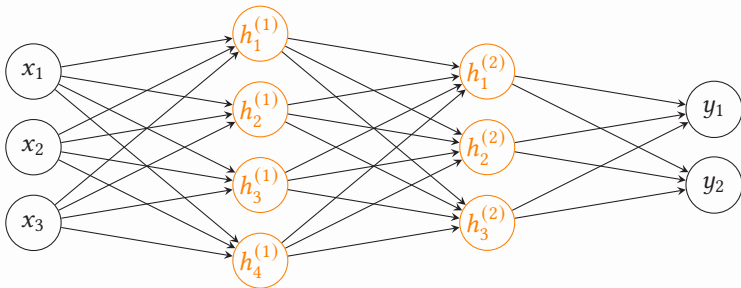
A deeper network

Input units

Hidden units

Hidden units

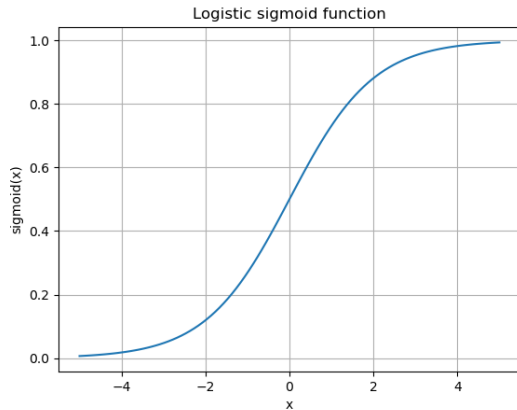
Output units



Three layer network with 3 input units, 4 + 3 hidden units and 2 output units.

Activation Functions

Sigmoid function



Sigmoid Hidden Units (SigmoidLayer)

- ☺ Compress unbounded inputs to $(0,1)$, saturating high magnitudes to 1
- ☺ Interpretable as the probability of a feature defined by their weight vector

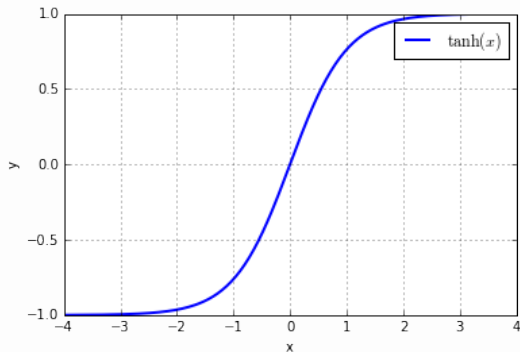
Sigmoid Hidden Units (SigmoidLayer)

- ☺ Compress unbounded inputs to $(0,1)$, saturating high magnitudes to 1
- ☺ Interpretable as the probability of a feature defined by their weight vector

However...

- ☹ Saturation causes gradients to approach 0
 - If the output of a sigmoid unit is h , then the gradient is $h(1 - h)$ which approaches 0 as h saturates to 0 or 1 – hence the gradients it multiplies into approach 0.
 - Small gradients result in small parameter changes, so learning becomes slow
- ☹ Outputs are not centred at 0
 - The output of a sigmoid layer will have $\text{mean} > 0$
 - Positive mean value of the next layer input, affecting the network convergence (e.g. activations and gradients becoming too big).

tanh



$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh(x) = 2 \operatorname{sigmoid}(2x) - 1$$

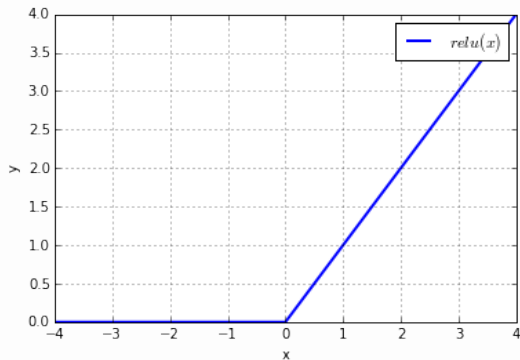
Derivative:

$$\frac{d}{dx} \tanh(x) = 1 - \tanh^2(x)$$

tanh hidden units (TanhLayer)

- ☺ tanh has same shape as sigmoid but has output range ± 1
- ☺ Results about approximation capability using sigmoid layers also apply to tanh layers
- ☹ Saturation is still a problem

Rectified Linear Unit – ReLU



$$relu(x) = \max(0, x)$$

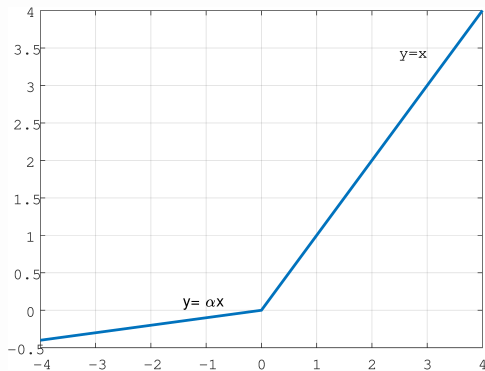
Derivative:

$$\frac{d}{dx} relu(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

ReLU hidden units (ReluLayer)

- 😊 Similar approximation results to tanh and sigmoid hidden units
- 😊 Empirical results for speech and vision show consistent improvements using ReLu over sigmoid or tanh
- 😊 Unlike tanh or sigmoid there is no positive saturation – saturation results in very small derivatives (and hence slower learning)
- 😊 ReLu is computationally efficient: $\max(0, x)$
- 😞 Negative input to ReLu results in zero gradient (and hence no learning)
- 😞 ReLu units can “die” (i.e. respond with 0 to everything)

ReLU Variants: Leaky and Parametric ReLU



$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

Derivative:

$$\frac{d}{dx} \text{LeakyReLU}(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

ReLU Variants: Leaky and Parametric ReLU

- Leaky ReLU has a small predetermined slope for negative values, $y = \alpha x$ (e.g. $\alpha = 0.01$)
- Parametric ReLU (PReLU) is a type of Leaky ReLU but α is learned during training
- It fixes the “dying ReLU” problem, no saturation for $x < 0$
- Gradient is not sparse
- Not always superior to ReLU

Summary

- Understanding what single-layer networks compute
- How multi-layer networks allow feature computation
- The importance of non-linear activation functions
- Universal approximation theorem
- **Reading:**
 - Recommended: Goodfellow 6.1, 6.3, 6.4
 - Keen readers: Montufar et al. 2014



<https://forms.cloud.microsoft/e/WdeYNv4A5c>