EE5904/ME5404: Neural Networks
Lecture 03

Xingyu Liu
xyl@nus.edu.sg

Assistant Professor
Department of Electrical & Computer Engineering
National University of Singapore

Reminder: Assignment 1

Due 23:59 (SGT), Sunday, 15 February 2026.

Late submission will not be accepted unless it is well justified!

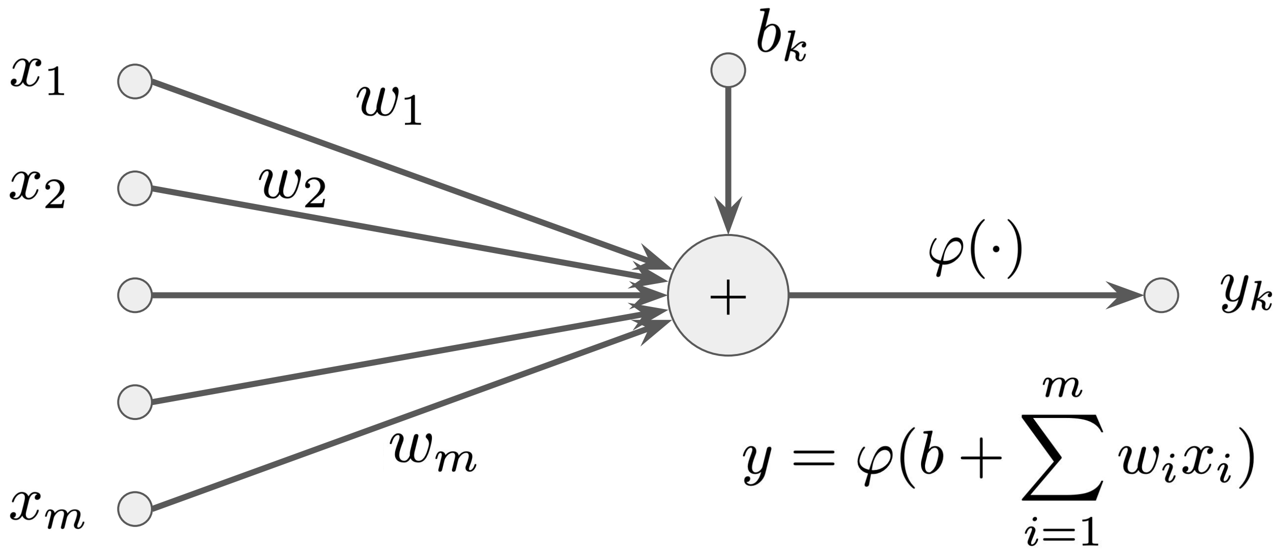
Submission instructions

- Submit the assignment via Canvas.
- Any **Python code** and **Python code generated results** should be included as an attachment.

Handwritten submissions are encouraged!

- If all questions (except the **Python code** and **code generated results such as figures**) are handwritten, you will receive a **10% bonus** on the assignment score.
- For handwritten work, **take clear photos of the pages** and upload them to Canvas.
- Ensure that your handwriting and photos are **clear and legible**. Illegible submissions may lose marks.

Recap of last lecture: perceptron



$$y = \varphi(b + \sum_{i=1}^{m} w_i x_i)$$

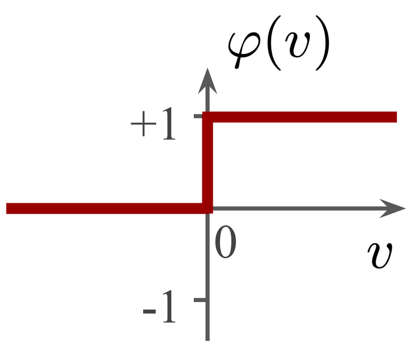
A single artificial neuron with threshold activation function

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

$$\boldsymbol{x} = [1, x_1, x_2, \ldots, x_m]^T$$

$$\boldsymbol{w} = [b, w_1, w_2, \ldots, w_m]^T$$

$$\boldsymbol{y} = \varphi(\boldsymbol{w}^T \boldsymbol{x})$$



Recap of last lecture: perceptron learning algorithm

Notation: use (n) to denote variables in n-th update

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
while there exist data sample that are misclassified **do**
draw the next misclassified sample $(\boldsymbol{x}^{(n)}, y^{*(n)})$
compute prediction and prediction error:

$$y^{(n)} \leftarrow \varphi({\boldsymbol{w}^{(n)}}^T \boldsymbol{x}^{(n)})$$

$$e^{(n)} \leftarrow y^{*(n)} - y^{(n)}$$

update the weight vector

$$\boldsymbol{w}^{(n+1)} \leftarrow \boldsymbol{w}^{(n)} + \eta \cdot e^{(n)} \boldsymbol{x}^{(n)}$$

increment n
end

Recap of last lecture: perceptron learning algorithm

Definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$ $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

Recap of last lecture: perceptron learning algorithm

Definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\}$ $\beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

This means $n \leq \dfrac{\|\boldsymbol{w}_0\|^2\beta}{\alpha^2} = \dfrac{\|\boldsymbol{w}_0\|^2 \cdot \max_i \|\boldsymbol{x}^{(i)}\|^2}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|^2} = \Big(\dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\Big)^2$

What is $\dfrac{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$?

Recap of last lecture: perceptron learning algorithm

Definitions: $\alpha = \min\{\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}\} \quad \beta = \max\{\|\boldsymbol{x}^{(i)}\|^2\}$

If the algorithm continues at update n, it must satisfy

$$\frac{n^2\alpha^2}{\|\boldsymbol{w}_0\|^2} \leq \|\boldsymbol{w}^{(n+1)}\|^2 \leq n\beta$$

This means $n \leq \dfrac{\|\boldsymbol{w}_0\|^2\beta}{\alpha^2} = \dfrac{\|\boldsymbol{w}_0\|^2 \cdot \max_i \|\boldsymbol{x}^{(i)}\|^2}{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|^2} = \left(\dfrac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2$

What is $\dfrac{\min_i |\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \min\limits_i \dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|} = \gamma$?

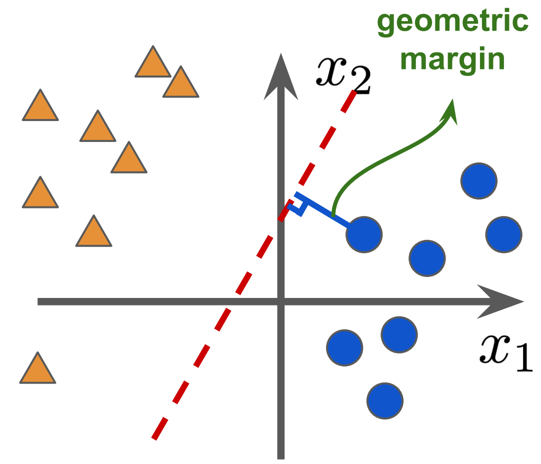
What is $\dfrac{|\boldsymbol{w}_0^T \boldsymbol{x}^{(i)}|}{\|\boldsymbol{w}_0\|}$?

Recall 2D case:
The perpendicular distance from point (x_0, y_0) *to line* $ax + by = 0$ *is*

$$d = \frac{|ax_0 + by_0|}{\sqrt{a^2 + b^2}}$$

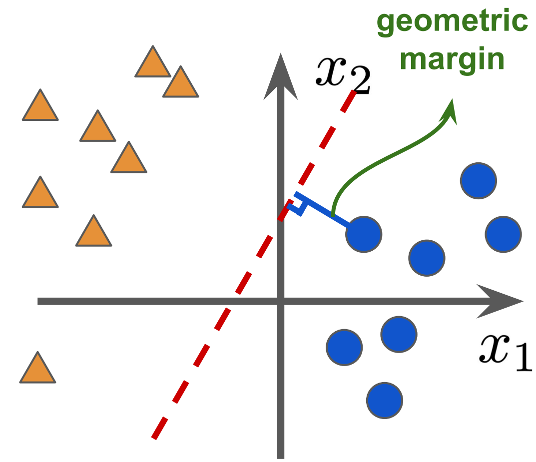
Recap of last lecture: perceptron learning convergence

γ is the geometric margin of the dataset with respect to the separating hyperplane $\boldsymbol{w}_0^T\boldsymbol{x} = 0$
(Data samples are not only correctly separated by the hyperplane $\boldsymbol{w}_0^T\boldsymbol{x} = 0$, but with a margin of at least γ to the decision boundary)



Recap of last lecture: perceptron learning convergence

γ **is the** **geometric margin** **of the dataset with respect to the separating hyperplane** $\boldsymbol{w}_0^T \boldsymbol{x} = 0$ (Data samples are not only correctly separated by the hyperplane $\boldsymbol{w}_0^T \boldsymbol{x} = 0$, but with a margin of at least γ to the decision boundary)



If the data samples are linearly separable by margin of at least γ, the number of updates that perceptron learning algorithm has is at most

$$\left(\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\gamma}\right)^2$$

In other words, if the perceptron learning algorithm does not end after n updates, it means the data samples are either 1) not linearly separable, or 2) they are linearly separable but the separation margin is less than

$$\frac{\max_i \|\boldsymbol{x}^{(i)}\|}{\sqrt{n}}$$

Recap of last lecture: Some basics of matrix calculus

Product rule:

$$\nabla_{\boldsymbol{x}}(A(\boldsymbol{x})^T \cdot B(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} A(\boldsymbol{x}))^T \cdot B(\boldsymbol{x}) + (\nabla_{\boldsymbol{x}} B(\boldsymbol{x}))^T \cdot A(\boldsymbol{x})$$

Example:

$$\nabla_{\boldsymbol{x}}(\boldsymbol{x}^T \boldsymbol{x}) = (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} + (\nabla_{\boldsymbol{x}} \boldsymbol{x})^T \cdot \boldsymbol{x} = I^T \boldsymbol{x} + I^T \boldsymbol{x} = 2\boldsymbol{x}$$

Chain rule:

$$\nabla_{\boldsymbol{x}} f(g(\boldsymbol{x})) = (\nabla_{\boldsymbol{x}} g(\boldsymbol{x}))^T \nabla_g f(g)$$

Recap of last lecture: Least-Mean-Squares algorithm (1960)

Notation: use (n) to denote variables in n-th iteration

Algorithm:
Randomly initialize weight vector $\boldsymbol{w}^{(1)}$
for $i = 1, 2, \ldots, n$ **do**

$$e^{(i)} \leftarrow y^{(i)} - \boldsymbol{w}^{(i)T} \boldsymbol{x}^{(i)}$$

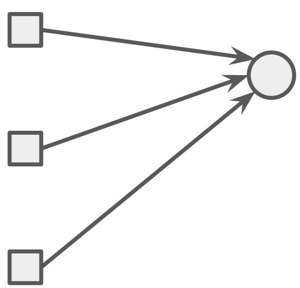
$$w^{(i+1)} \leftarrow w^{(i)} + \eta e^{(i)} \boldsymbol{x}^{(i)}$$

end

Multilayer Perceptron (MLP)

Connections of perceptrons

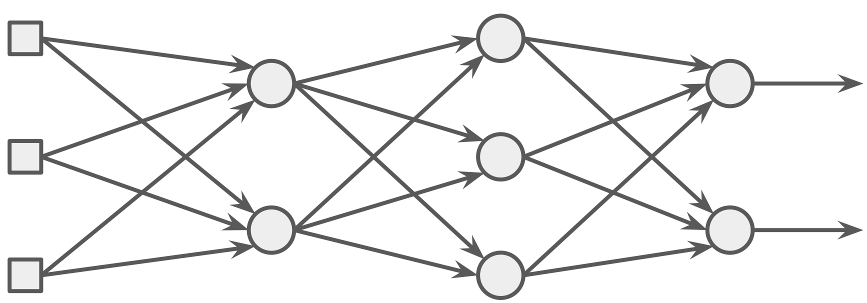
A single neuron / perceptron is insufficient for more complex problems



Connections of perceptrons

A single neuron / perceptron is insufficient for more complex problems; networks consisting of a large number of interconnected neurons are often used.

Network architecture defines the number of neurons and how they are connected!



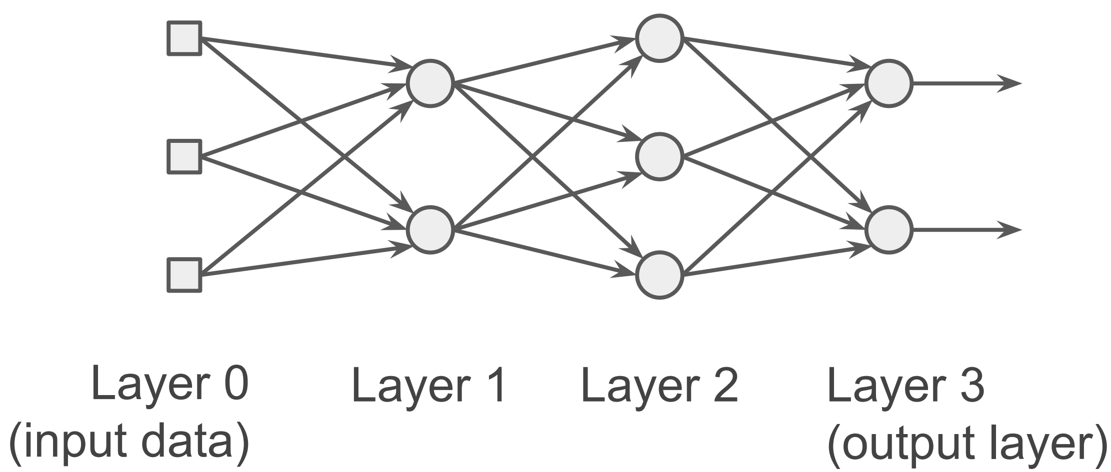
Connections of perceptrons

A single neuron / perceptron is insufficient for more complex problems; networks consisting of a large number of interconnected neurons are often used.

Network architecture defines the number of neurons and how they are connected!

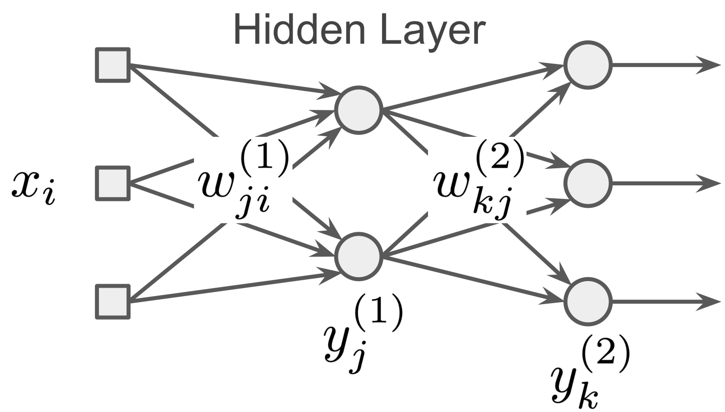
Layered Feedforward Networks:

- Nodes are partitioned into subsets called **layers**;
- No connections from layer j to layer i if $j > i$;
- Connections from a directed graph, no cyclic connections;



Multilayer Perceptron (MLP)

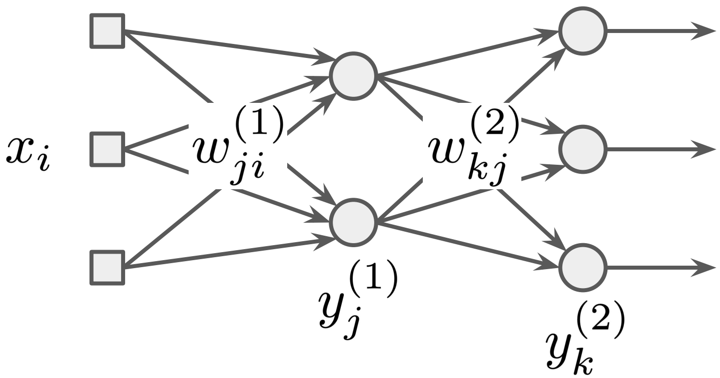
Multilayer perceptron (MLP) is a type of layered feedforward networks with non-linear activation function and with at least one hidden layer



$$y_j^{(1)} = \varphi(\sum_i w_{ji}^{(1)} x_i) \quad \Rightarrow \quad y_k^{(2)} = \sum_j w_{kj}^{(2)} y_j^{(1)} = \sum_j w_{kj}^{(2)} \varphi\left(\sum_i w_{ji}^{(1)} x_i\right)$$

$$y_k^{(2)} = \sum_j w_{kj}^{(2)} y_j^{(1)}$$

MLP: vectorizing computation



$$\boldsymbol{x} = [x_1, x_2, \ldots, x_i, \ldots]^T$$

$$\boldsymbol{y}^{(1)} = [y_1^{(1)}, y_2^{(1)}, \ldots, y_j^{(1)}, \ldots]^T$$

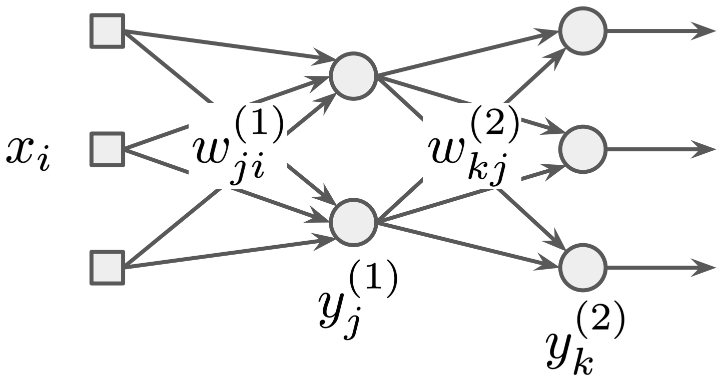
$$\boldsymbol{y}^{(2)} = [y_1^{(2)}, y_2^{(2)}, \ldots, y_k^{(2)}, \ldots]^T$$

$$\boldsymbol{W}^{(1)} = [w_{ji}^{(1)}] \qquad \boldsymbol{W}^{(2)} = [w_{kj}^{(2)}]$$

$$\boldsymbol{y}^{(1)} = \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x}) \qquad \Rightarrow \qquad \boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$$

$$\boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}$$

MLP: vectorizing computation



$$\boldsymbol{x} = [x_1, x_2, \dots, x_i, \dots]^T$$

$$\boldsymbol{y}^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_j^{(1)}, \dots]^T$$

$$\boldsymbol{y}^{(2)} = [y_1^{(2)}, y_2^{(2)}, \dots, y_k^{(2)}, \dots]^T$$

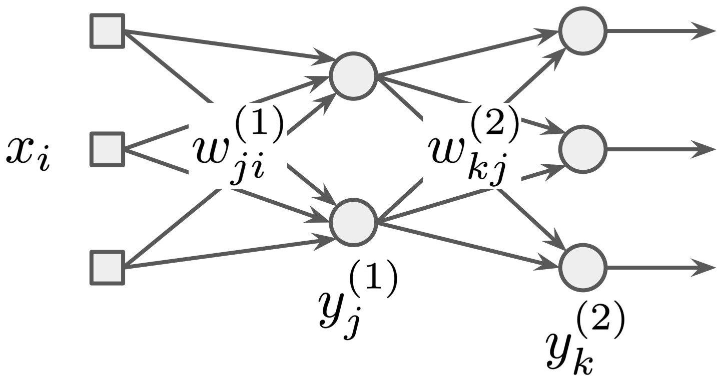
$$\boldsymbol{W}^{(1)} = [w_{ji}^{(1)}] \qquad \boldsymbol{W}^{(2)} = [w_{kj}^{(2)}]$$

$$\boldsymbol{y}^{(1)} = \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$$
$$\boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}$$
$$\Rightarrow \qquad \boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$$

Activation function φ

- Introduces non-linearity to the network.
- Applied to hidden layers.
- Usually **not** applied to the output layer.

MLP: vectorizing computation



$$\boldsymbol{x} = [x_1, x_2, \ldots, x_i, \ldots]^T$$
$$\boldsymbol{y}^{(1)} = [y_1^{(1)}, y_2^{(1)}, \ldots, y_j^{(1)}, \ldots]^T$$
$$\boldsymbol{y}^{(2)} = [y_1^{(2)}, y_2^{(2)}, \ldots, y_k^{(2)}, \ldots]^T$$
$$\boldsymbol{W}^{(1)} = [w_{ji}^{(1)}] \qquad \boldsymbol{W}^{(2)} = [w_{kj}^{(2)}]$$

$$\boldsymbol{y}^{(1)} = \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x}) \qquad \Rightarrow \qquad \boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\varphi(\boldsymbol{W}^{(1)}\boldsymbol{x})$$
$$\boldsymbol{y}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{y}^{(1)}$$

Activation function φ

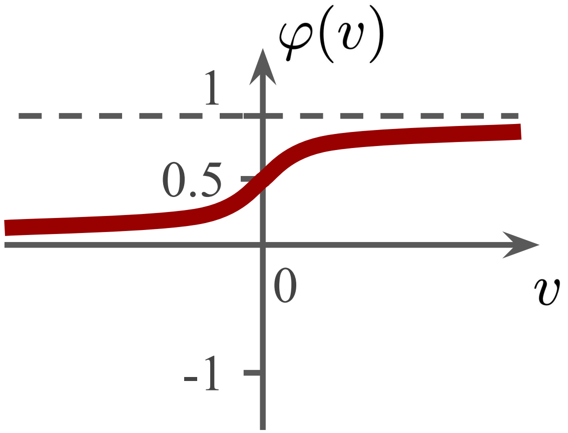
- Introduces non-linearity to the network.
- Applied to hidden layers.
- Usually **not** applied to the output layer.

Reason of layered architecture:

In the programming of neural networks, vectorized operations are pretty much always used for speed reasons!

Activation function choices

Sigmoid function:



$$\sigma(v) = \frac{1}{1+e^{-av}}$$

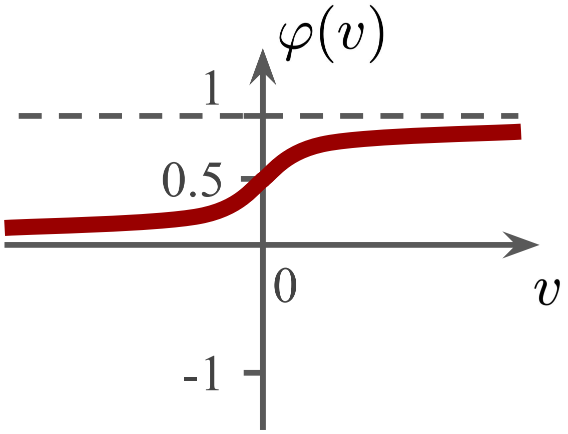
$$\sigma'(v) = \frac{ae^{-av}}{(1+e^{-av})^2} = a\frac{e^{-av}}{1+e^{-av}}\frac{1}{1+e^{-av}} = a\sigma(v)(1-\sigma(v))$$

$$\sigma'(0) = a(1-\frac{1}{2})\frac{1}{2} = \frac{a}{4}$$

Recall: $(\frac{u}{v})' = \frac{u'v - uv'}{v^2}$

When $a \to +\infty$, Sigmoid function is reduced to a step function at 0.

Activation function choices



Sigmoid function:

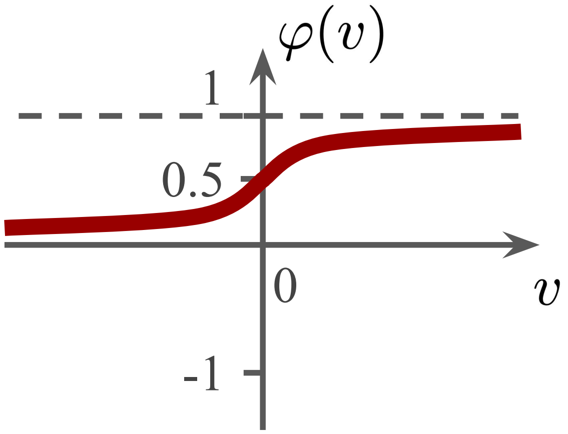
$$\sigma(v) = \frac{1}{1+e^{-av}}$$

One of the most commonly used activation functions

<u>Nice properties of Sigmoid function as activation</u>:

- Value range is limited between (0,1)
- Monotonically increasing
- Continuous & differentiable everywhere
- Gradient non-zero everywhere
- Slope can be adjusted by adjusting parameter a

Activation function choices



Sigmoid function:

$$\sigma(v) = \frac{1}{1 + e^{-av}}$$

One of the most commonly used activation functions

<u>Nice properties of Sigmoid function as activation</u>:

- Value range is limited between (0,1)
- Monotonically increasing
- Continuous & differentiable everywhere
- Gradient non-zero everywhere
- Slope can be adjusted by adjusting parameter a

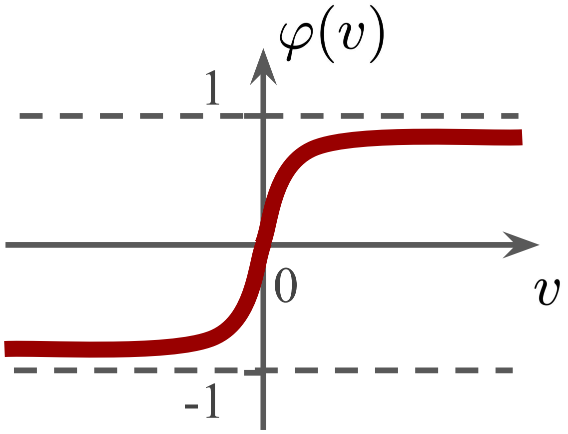
<u>Limitations of Sigmoid function as activation</u>:

- Value range is limited between (0,1)
- Vanishing gradient when $v \to +\infty$ or $v \to -\infty$

Activation function choices

Hyperbolic tangent function:

$$\tanh(v) = \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}}$$

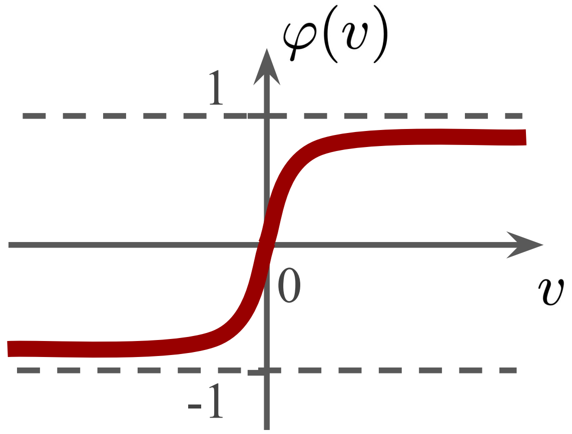


Activation function choices

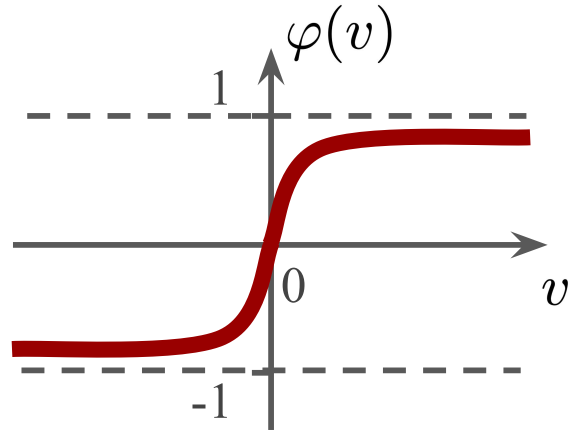
Hyperbolic tangent function:

$$\tanh(v) = \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}}$$

$$\tanh(v) = 2\sigma(2v) - 1$$



Activation function choices



Hyperbolic tangent function:

$$\tanh(v) = \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}}$$

$$\tanh(v) = 2\sigma(2v) - 1$$

<u>Similar nice properties as activation</u>:

- Value range is limited between (-1,1)
- Monotonically increasing
- Continuous & differentiable everywhere
- Gradient non-zero everywhere
- Slope can be adjusted by adjusting parameter a

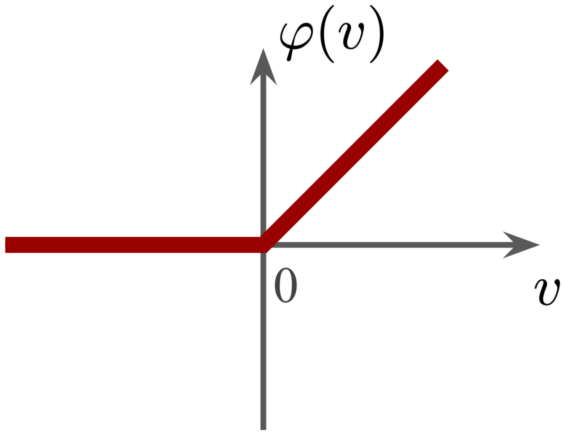
<u>Similar limitations as activation</u>:

- Value range is limited between (-1,1)
- Vanishing gradient when $v \rightarrow +\infty$ or $v \rightarrow -\infty$

Activation function choices

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v)$$
$$= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

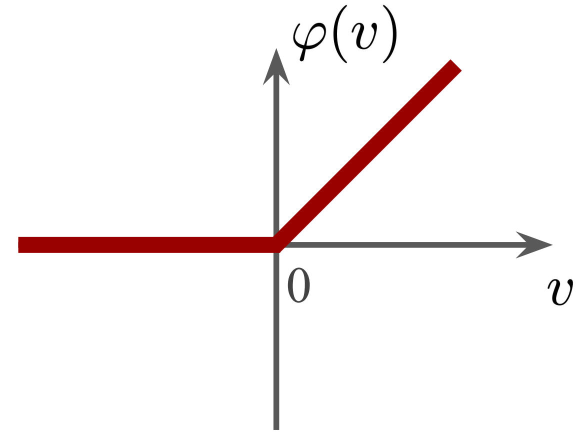


Activation function choices

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v)$$
$$= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$

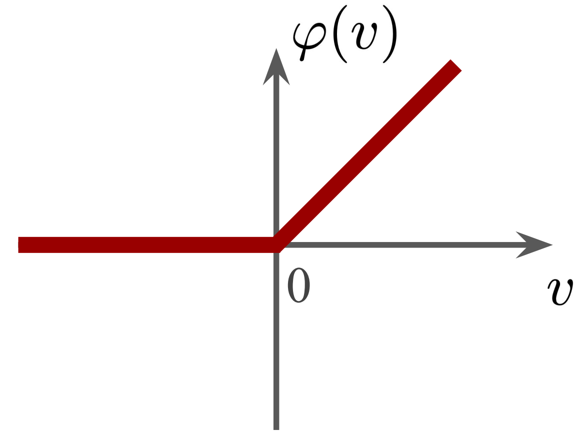


Activation function choices

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v)$$
$$= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$



<u>Nice properties of ReLU as activation</u>:

- Non-saturating for positive inputs
  - avoids vanishing gradients when $v \to +\infty$
- Unbounded positive output
  - allows large activations when needed
- Simple gradient
- Sparse activations

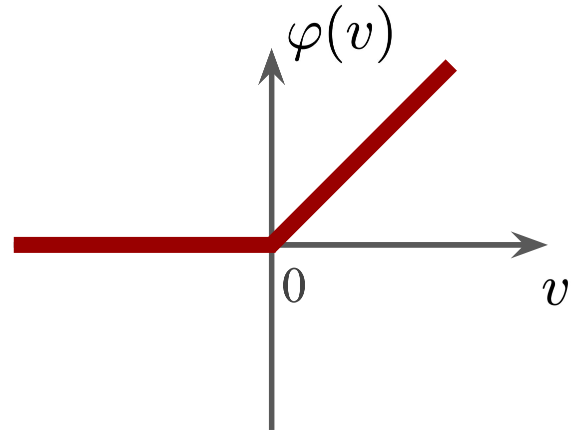
Activation function choices

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v)$$

$$= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$



<u>**Nice properties of ReLU as activation**</u>:

- Non-saturating for positive inputs
  - avoids vanishing gradients when $v \to +\infty$
- Unbounded positive output
  - allows large activations when needed
- Simple gradient
- Sparse activations

<u>**Limitations of ReLU as activation**</u>:

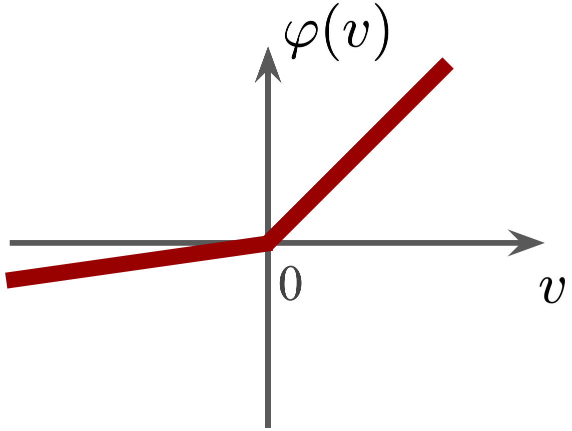
- Zero gradient for negative inputs
- Not differentiable at 0

Activation function choices

Leaky ReLU:

$$\mathrm{LeakyReLU}(x) = \max(x, \alpha x) \qquad (0 < \alpha \ll 1)$$

$$\mathrm{LeakyReLU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x < 0 \end{cases}$$

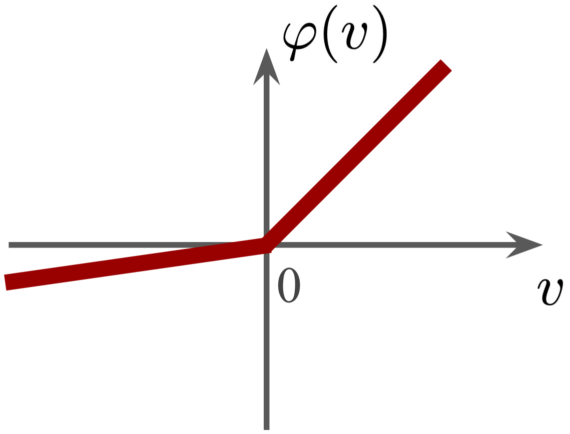


Activation function choices

Leaky ReLU:

$$\text{LeakyReLU}(x) = \max(x, \alpha x) \qquad (0 < \alpha \ll 1)$$

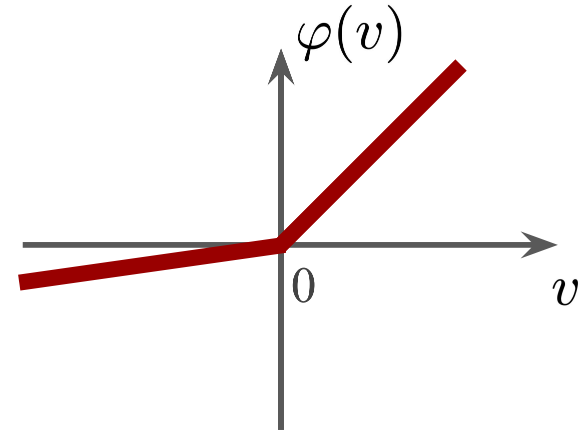
$$\text{LeakyReLU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x < 0 \end{cases}$$



<u>**Nice properties of Leaky ReLU as activation**</u>:

- Non-saturating for positive inputs
- Non-zero gradient for negative inputs
- Unbounded positive output
- Allows negative outputs
- Simple gradient

Activation function choices



Leaky ReLU:

$$\text{LeakyReLU}(x) = \max(x, \alpha x) \qquad (0 < \alpha \ll 1)$$

$$\text{LeakyReLU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x < 0 \end{cases}$$

<u>Nice properties of Leaky ReLU as activation</u>:
- Non-saturating for positive inputs
- Non-zero gradient for negative inputs
- Unbounded positive output
- Allows negative outputs
- Simple gradient

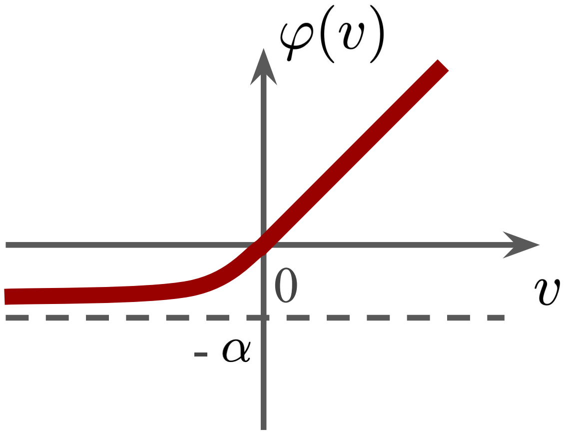
<u>Limitations of Leaky ReLU as activation</u>:
- No sparse activations
- Not differentiable at 0

Activation function choices

Exponential Linear Unit (ELU):

$$\mathrm{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha\left(e^{x} - 1\right), & x \leq 0 \end{cases}$$

$$\mathrm{ELU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^{x}, & x \leq 0 \end{cases}$$

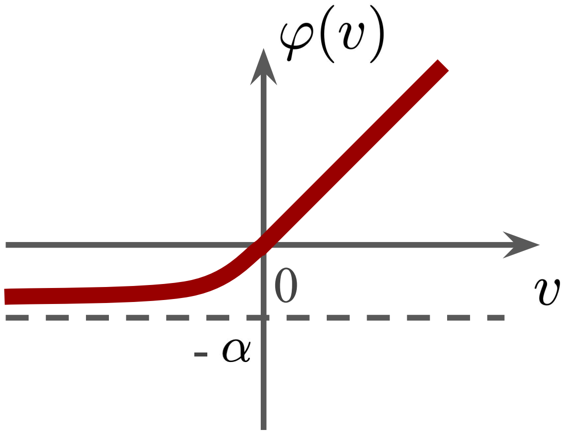


Activation function choices

Exponential Linear Unit (ELU):

$$\mathrm{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha\left(e^{x} - 1\right), & x \leq 0 \end{cases}$$

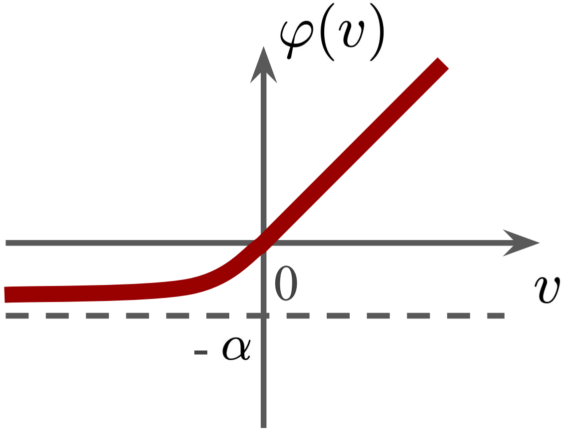
$$\mathrm{ELU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^{x}, & x \leq 0 \end{cases}$$



<u>Nice properties of ELU as activation</u>:

- Non-saturating for positive inputs
- Non-zero gradient for negative inputs
- Unbounded positive output
- Allows negative outputs

Activation function choices



Exponential Linear Unit (ELU):

$$\mathrm{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha\left(e^x - 1\right), & x \leq 0 \end{cases}$$

$$\mathrm{ELU}'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x \leq 0 \end{cases}$$

Nice properties of ELU as activation:

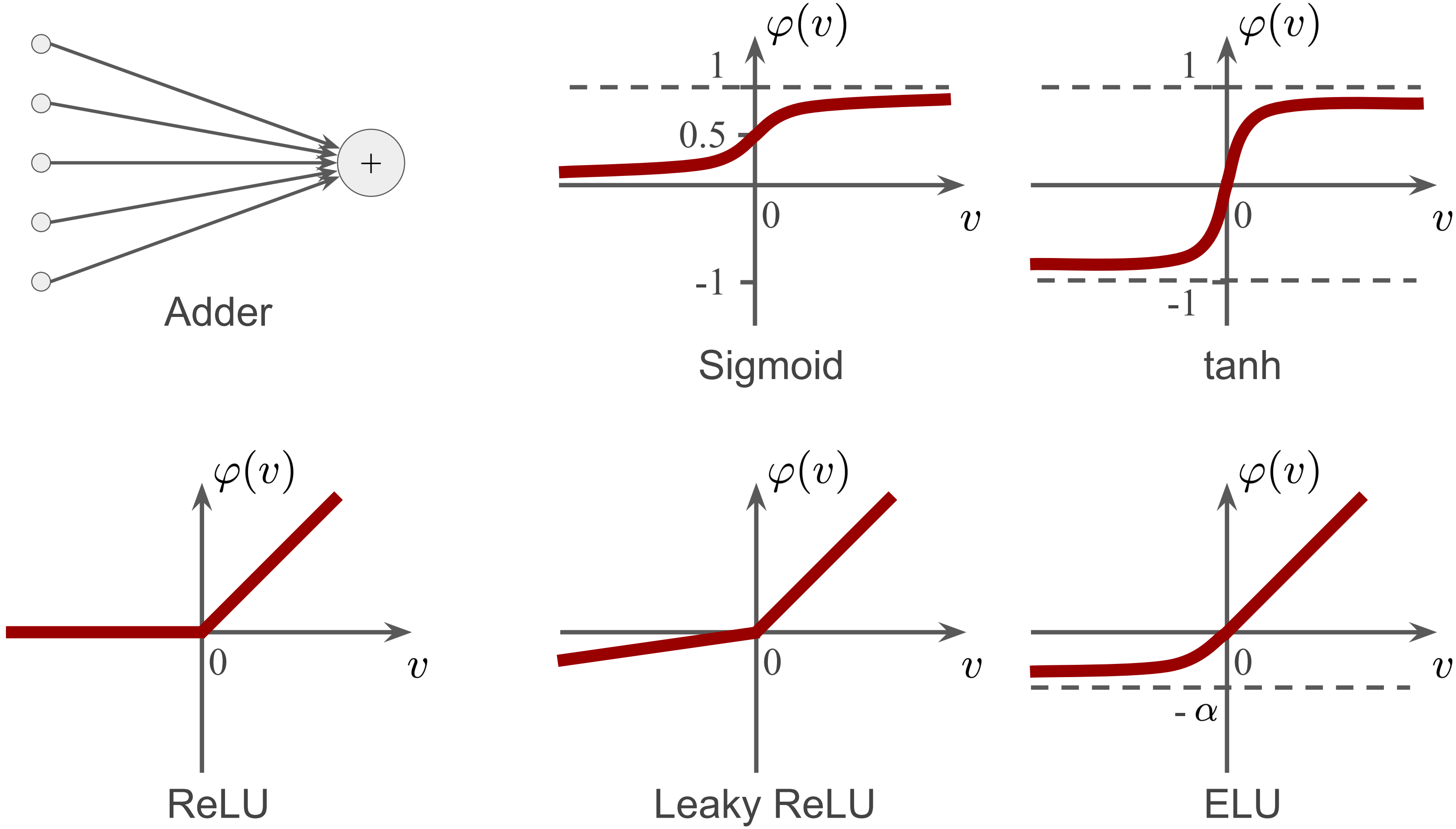
- Non-saturating for positive inputs
- Non-zero gradient for negative inputs
- Unbounded positive output
- Allows negative outputs

Limitations of ELU as activation:

- More expensive to compute than ReLU and Leaky ReLU (due to exp)
- Saturates for large negative inputs
- No sparse activations

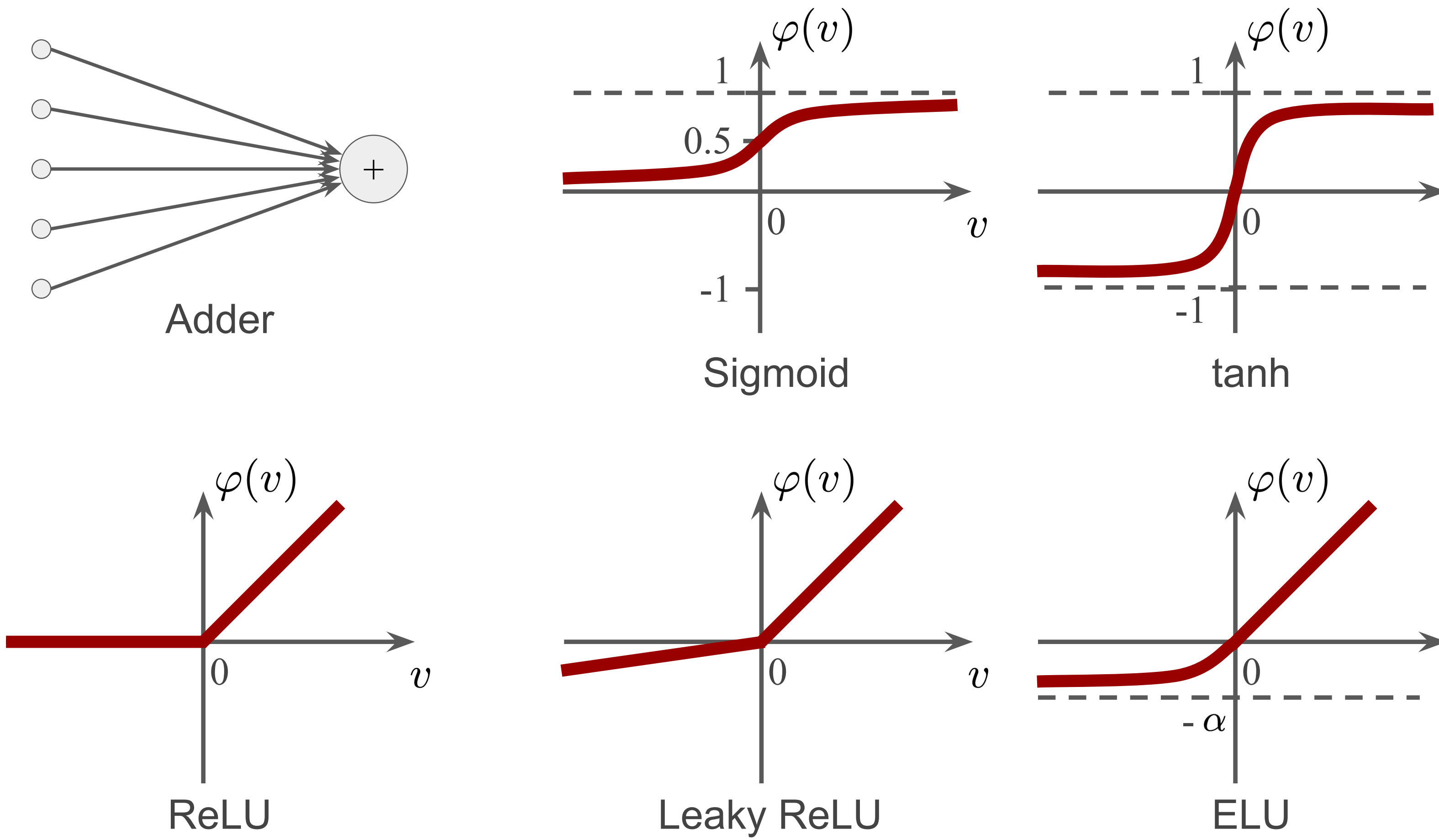
Activation function choices

What happens when we stack many layers of “Adder +activations”?



Activation function choices

What happens when we stack many layers of “Adder +activations”?



What kinds of functions emerge?

Universal Approximation Theorem (UAT)

Universal Approximation Theorem (UAT)

Why should we care?

- Neural networks work shockingly well in complex real-world problems
- But *why* do they work so well?
- Can they really represent any function we care about?

Universal Approximation Theorem (UAT)

Why should we care?

- Neural networks work shockingly well in complex real-world problems
- But *why* do they work so well?
- Can they really represent any function we care about?

The Big Question

Given a function $f(x)$, can a neural network approximate it arbitrarily well?

Universal Approximation Theorem (UAT)

Why should we care?

- Neural networks work shockingly well in complex real-world problems
- But *why* do they work so well?
- Can they really represent any function we care about?

The Big Question

Given a function $f(x)$, can a neural network approximate it arbitrarily well?

Yes!

This guaranteed by the Universal Approximation Theorem of neural networks

Let's first look at the case of neural networks with Sigmoid activation

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

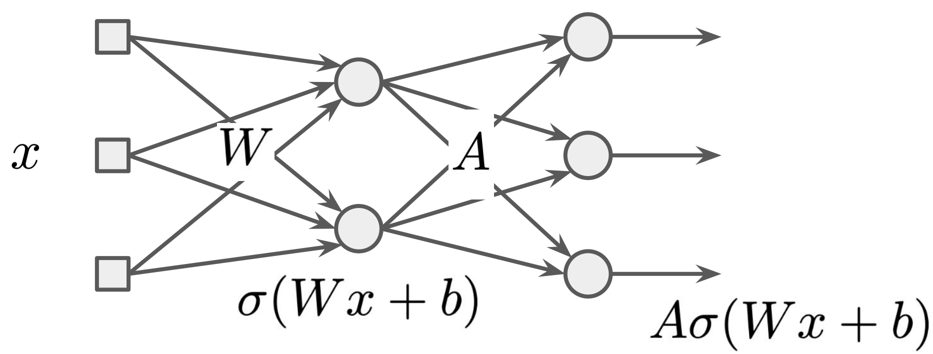
$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

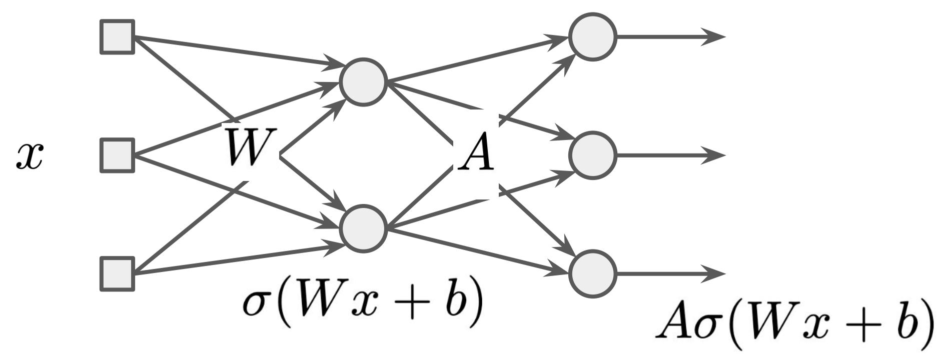


Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$



It means ***a <u>two-layer</u> neural network with <u>sufficient number of hidden layer neurons</u> can approximate any continuous vector-valued function defined on a <u>bounded</u> high-dimensional region***.

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$
  - There is no guarantee for values outside the compact set

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \| f(x) - A\sigma(Wx + b) \| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$
  - There is no guarantee for values outside the compact set
- $f : K \to \mathbb{R}^n$ is required to be a **continuous** function

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$
  - There is no guarantee for values outside the compact set
- $f : K \to \mathbb{R}^n$ is required to be a **continuous** function
  - There is no guarantee for functions with discontinuity

Universal Approximation Theorem (UAT): Sigmoid activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$
  - There is no guarantee for values outside the compact set
- $f : K \to \mathbb{R}^n$ is required to be a **continuous** function
  - There is no guarantee for functions with discontinuity
- For a specific $\varepsilon > 0$, we can only know N, W, b, A **exist**

Universal Approximation Theorem (UAT): Sigmoid activation

<u>**Theorem**</u>

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$

Key caveat about UAT

- **"Compact set"** means a region that is finite in size and includes its boundary. For example $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$
    - <u>There is no guarantee for values outside the compact set</u>
- $f : K \to \mathbb{R}^n$ is required to be a **continuous** function
    - <u>There is no guarantee for functions with discontinuity</u>
- For a specific $\varepsilon > 0$, we can only know N, W, b, A **exist**
    - <u>There is no guarantee that they can be found through a learning or optimization algorithm</u>

UAT: Sigmoid activation, 1D input, 1D output

Theorem

For any continuous function f on $[a, b]$ and any $\varepsilon > 0$,
there exists N, a_i, w_i, b_i such that:

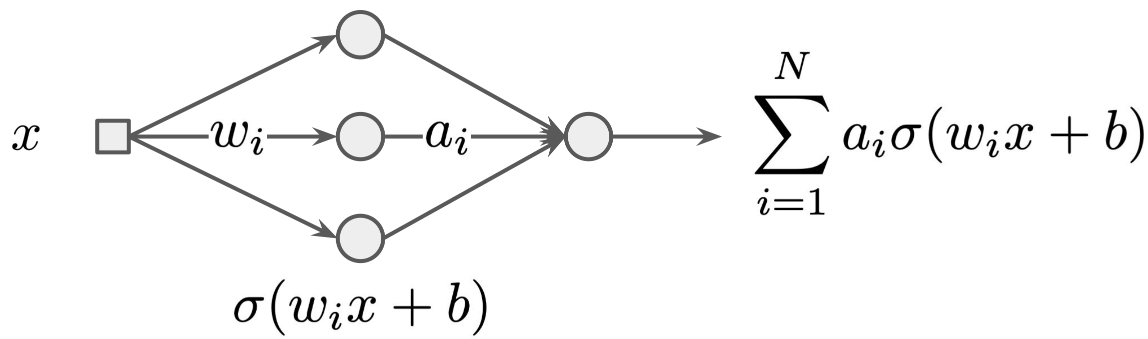
$$\left| f(x) - \sum_{i=1}^{N} a_i \sigma(w_i x + b) \right| < \varepsilon$$

UAT: Sigmoid activation, 1D input, 1D output

Theorem

For any continuous function f on $[a, b]$ and any $\varepsilon > 0$,
there exists N, a_i, w_i, b_i such that:

$$\left| f(x) - \sum_{i=1}^{N} a_i \sigma(w_i x + b) \right| < \varepsilon$$



Let’s prove this simplest case first!

UAT Proof: Sigmoid activation, 1D input, 1D output

What is $\sigma(w_i x + b)$?

UAT Proof: Sigmoid activation, 1D input, 1D output

What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

UAT Proof: Sigmoid activation, 1D input, 1D output

What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

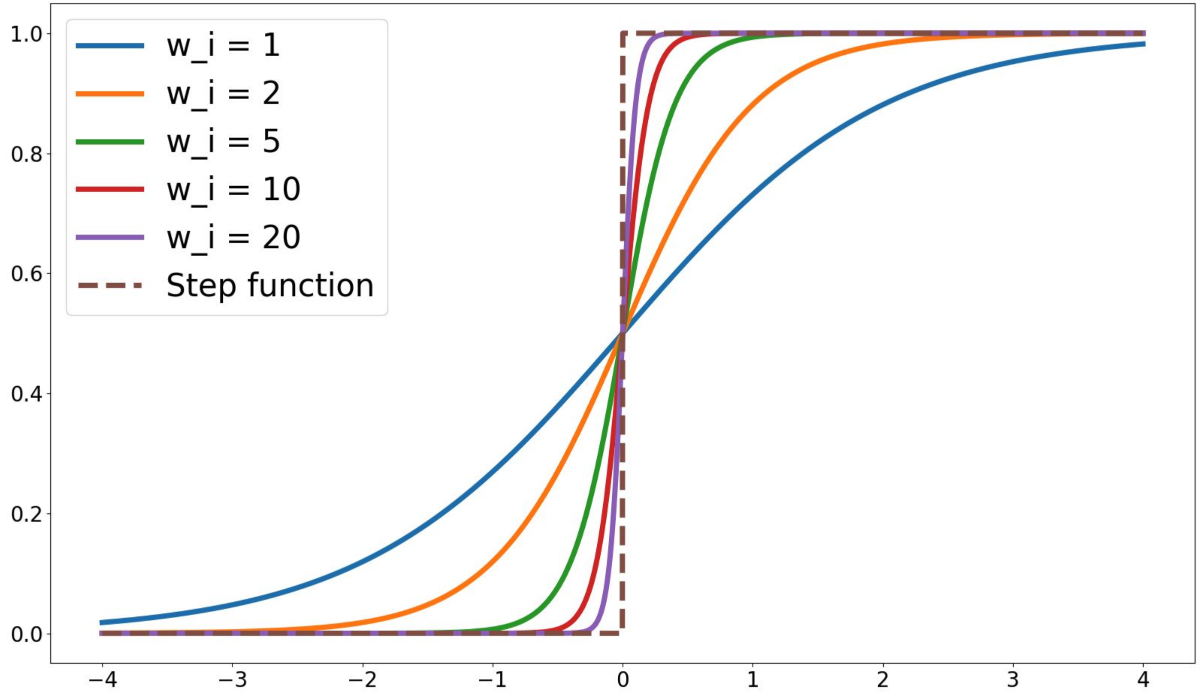
1. **Horizontal scaling**: squeezing or stretching along the x-axis:
   Larger $|w_i|$ means more squeezed; negative w_i means horizontal flip

UAT Proof: Sigmoid activation, 1D input, 1D output

<u>What is $\sigma(w_i x + b)$?</u>

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**: squeezing or stretching along the x-axis:
   Larger $|w_i|$ means more squeezed; negative w_i means horizontal flip



UAT Proof: Sigmoid activation, 1D input, 1D output

<u>What is $\sigma(w_i x + b)$?</u>

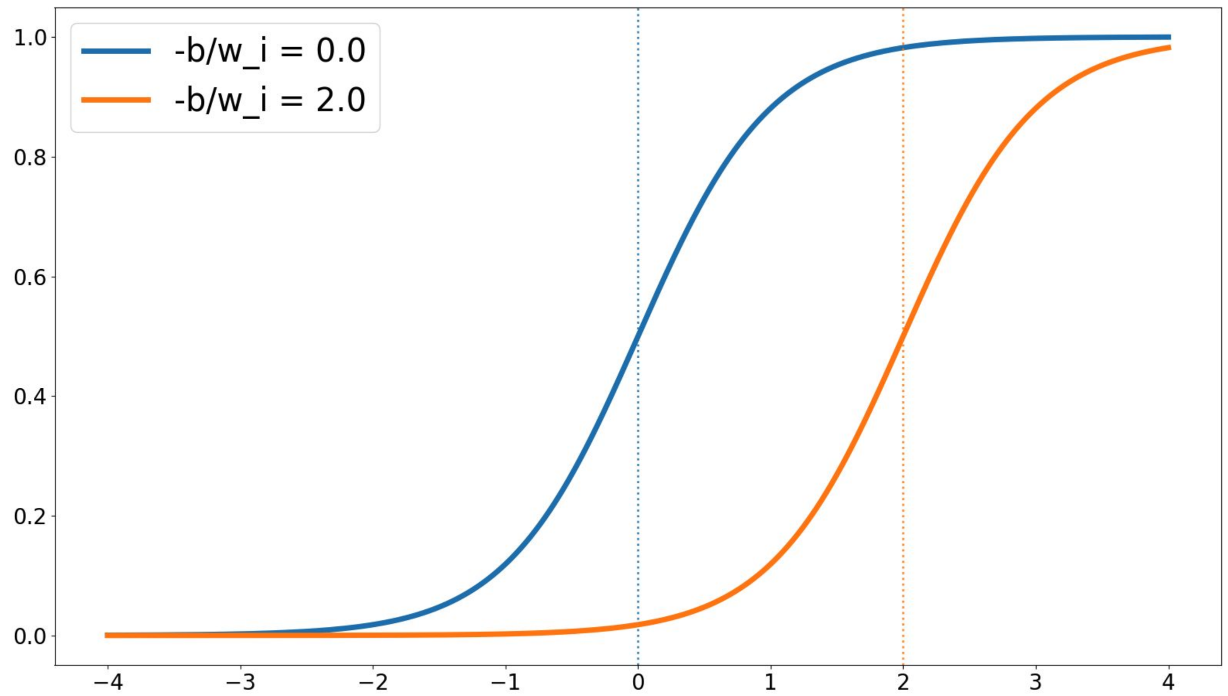
It is a sigmoid function that has undergone two transformations:

2. **Horizontal shift**: shifts the sigmoid left or right along the x-axis: The center of the sigmoid will be at $-b/w_i$

UAT Proof: Sigmoid activation, 1D input, 1D output

What is $\sigma(w_i x + b)$?
It is a sigmoid function that has undergone two transformations:

2. **Horizontal shift**: shifts the sigmoid left or right along the x-axis: The center of the sigmoid will be at $-b/w_i$



UAT Proof: Sigmoid activation, 1D input, 1D output

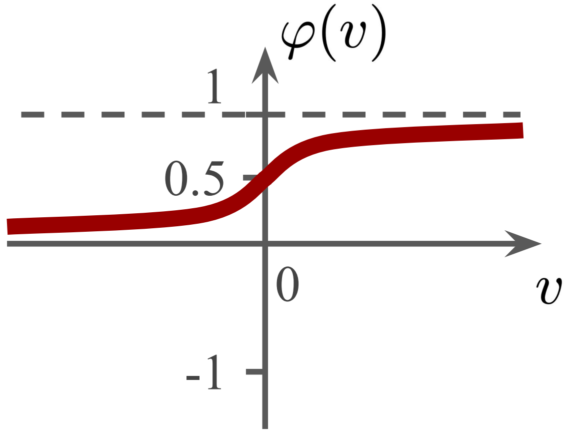
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

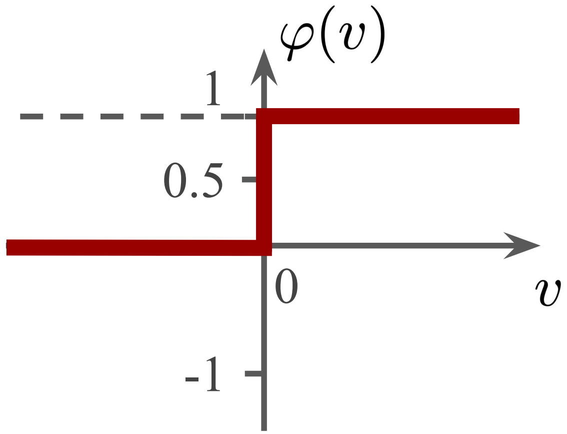
1. **Horizontal scaling**
2. **Horizontal shift**

We can let $w_i \to +\infty$, the Sigmoid function will approach a step function

$$\sigma(w_i(x-c)) \to \varphi(x-c)$$



$\rightarrow$



UAT Proof: Sigmoid activation, 1D input, 1D output

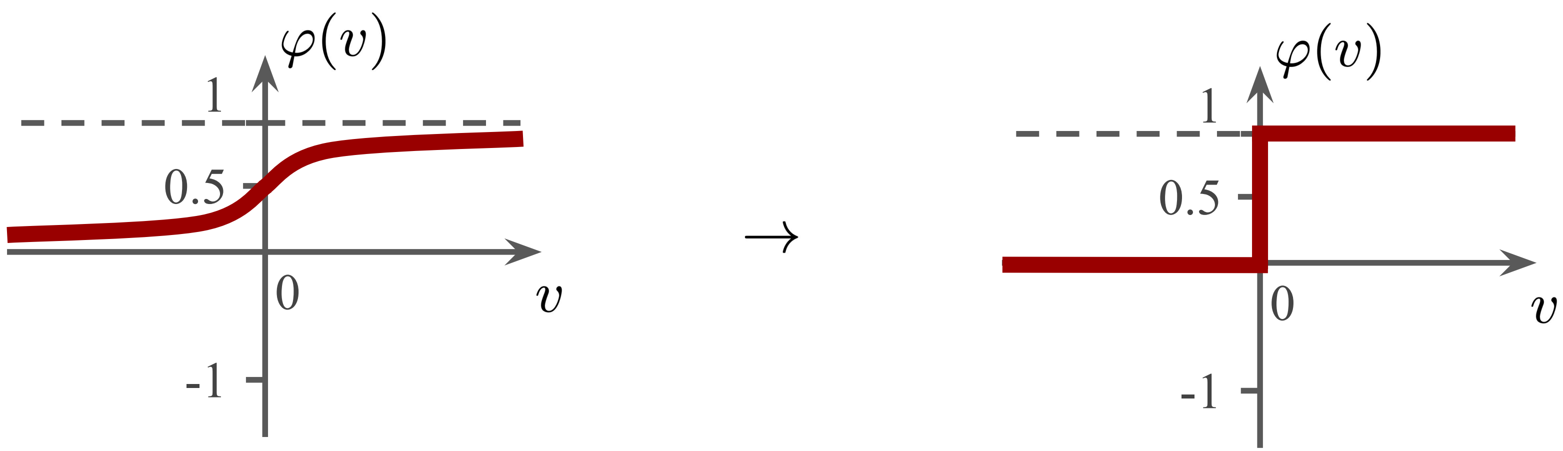
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

We can let $w_i \to +\infty$, the Sigmoid function will approach a step function

$$\sigma(w_i(x-c)) \to \varphi(x-c)$$



That is, for any $\varepsilon_1 > 0$, there exists sufficiently large w', such that

$$|\sigma(w'(x-c)) - \varphi(x-c)| < \varepsilon_1$$

UAT Proof: Sigmoid activation, 1D input, 1D output

<u>What is $\sigma(w_i x + b)$?</u>

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$

UAT Proof: Sigmoid activation, 1D input, 1D output

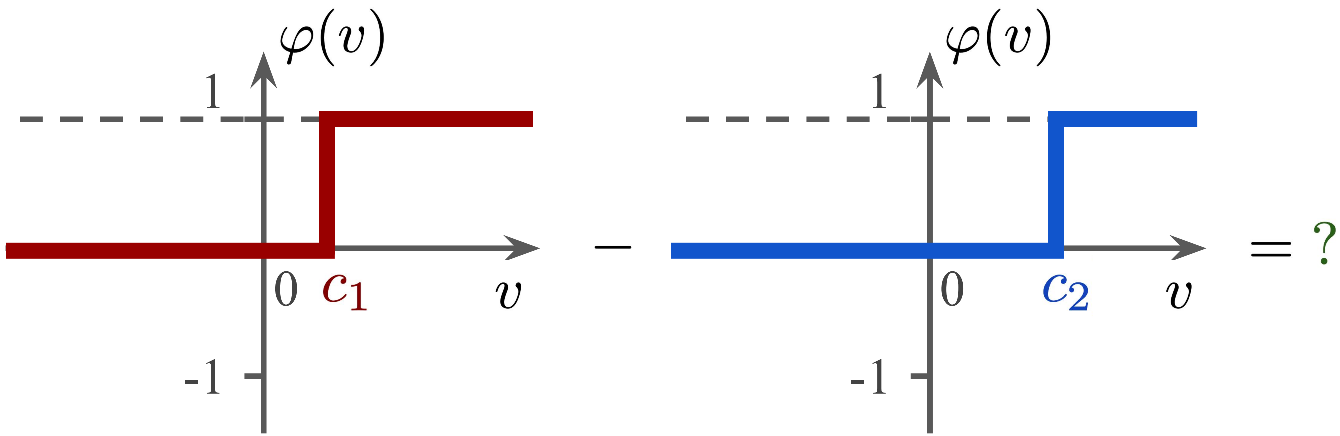
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$



UAT Proof: Sigmoid activation, 1D input, 1D output

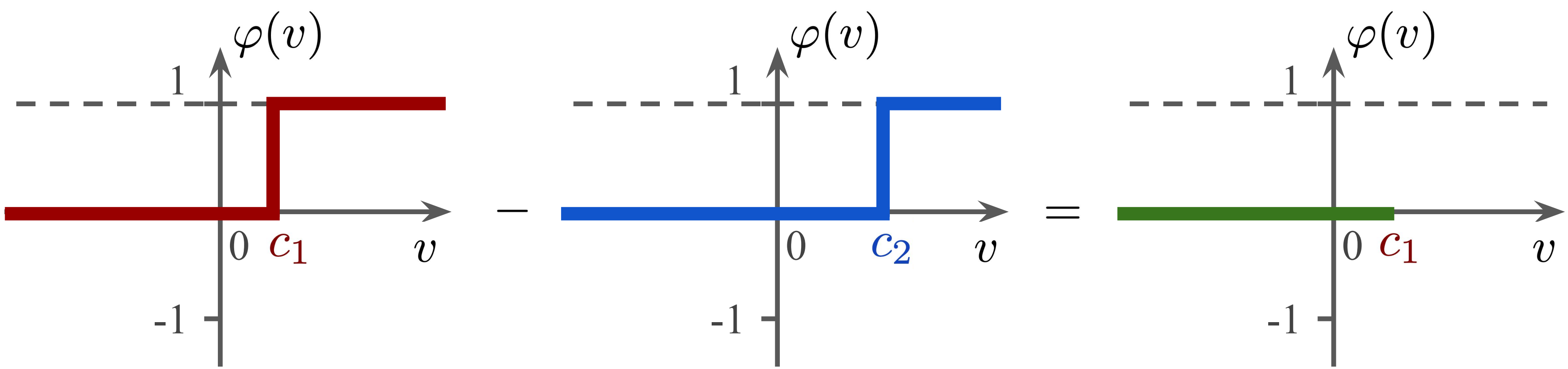
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$



UAT Proof: Sigmoid activation, 1D input, 1D output

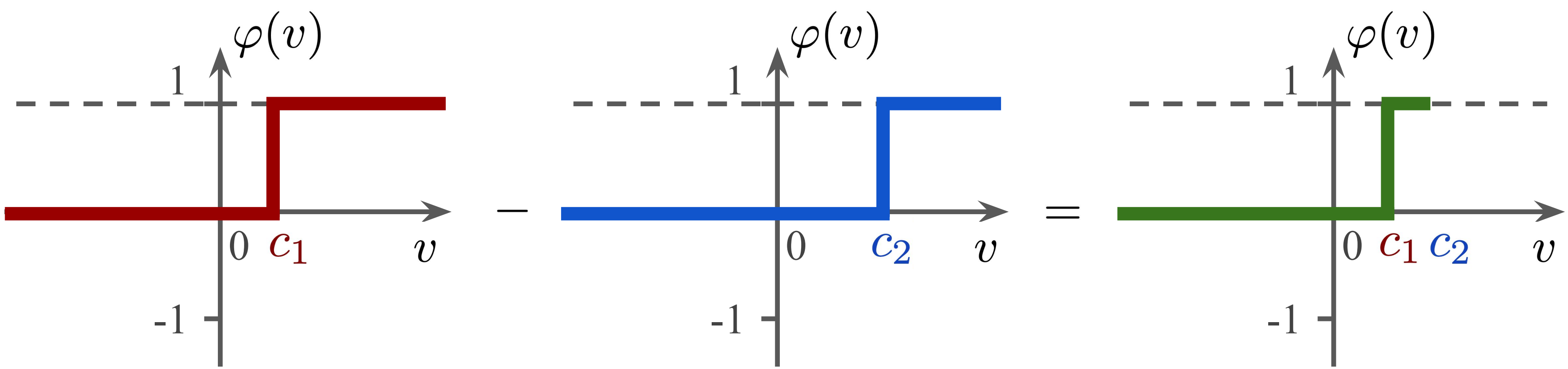
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$



UAT Proof: Sigmoid activation, 1D input, 1D output

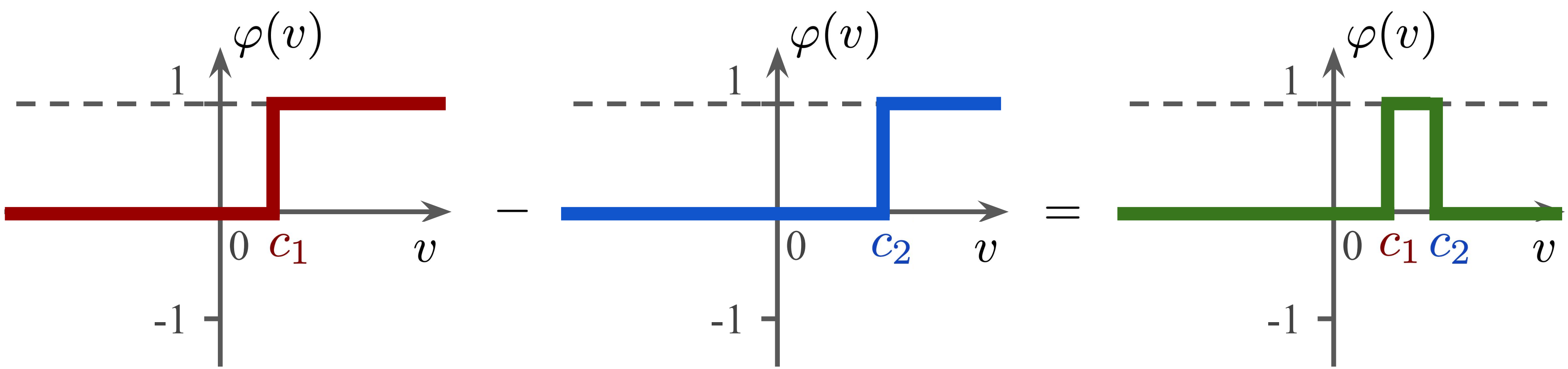
<u>**What is $\sigma(w_i x + b)$?**</u>

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$



UAT Proof: Sigmoid activation, 1D input, 1D output

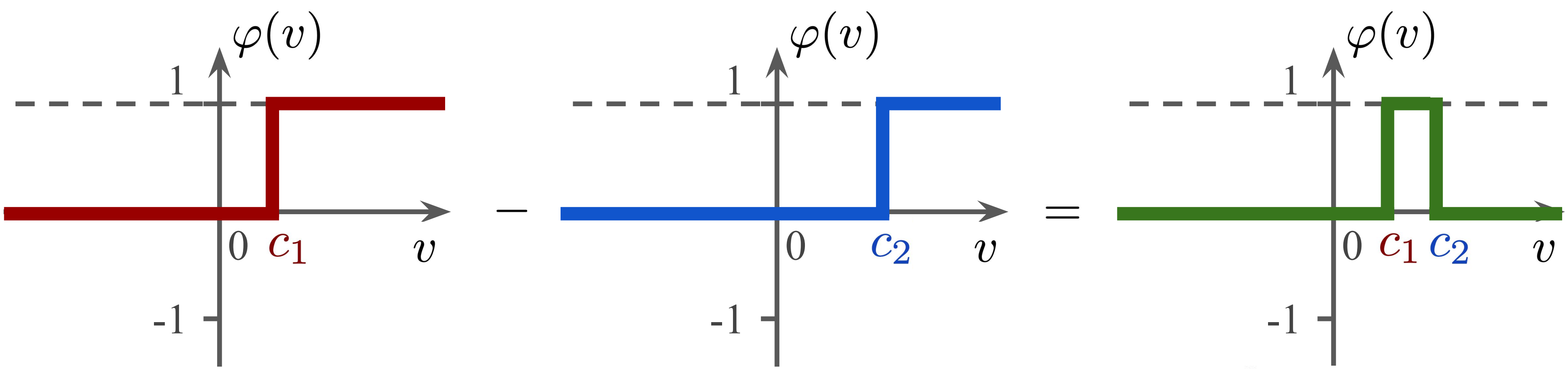
What is $\sigma(w_i x + b)$?

It is a sigmoid function that has undergone two transformations:

1. **Horizontal scaling**
2. **Horizontal shift**

Subtract a step function horizontally shifted by c_1 with another step function horizontally shifted by c_2, what is the result? Suppose $c_1 < c_2$

$$\varphi(x - c_1) - \varphi(x - c_2) = ?$$



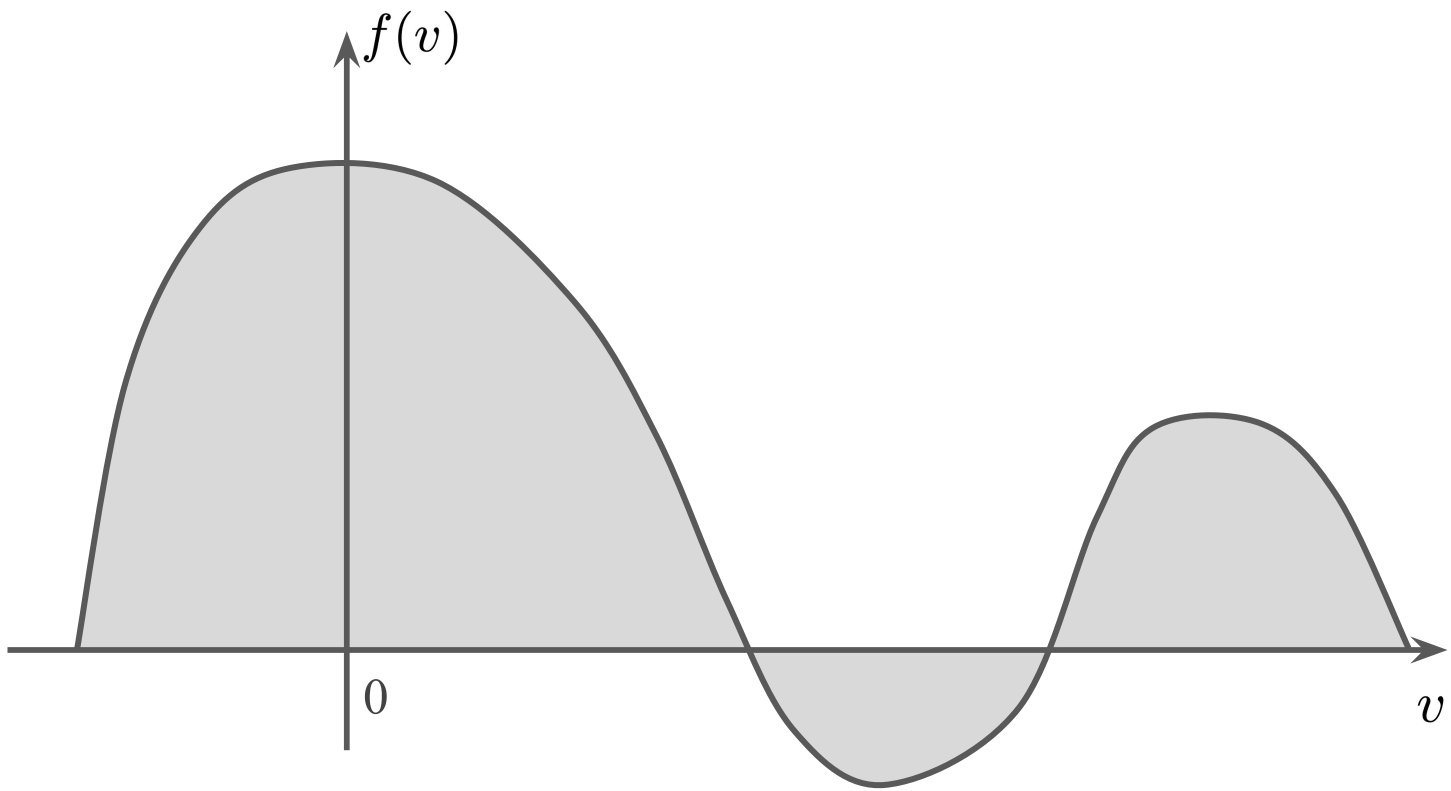
The result is an indicator function of interval $\mathbf{1}_{[c_1, c_2]}(x) = \begin{cases} 1, & x \in [c_1, c_2] \\ 0, & x \notin [c_1, c_2] \end{cases}$

UAT Proof: Sigmoid activation, 1D input, 1D output

Can a continuous function be approximated by a linear combination of multiple interval indicator functions?

UAT Proof: Sigmoid activation, 1D input, 1D output

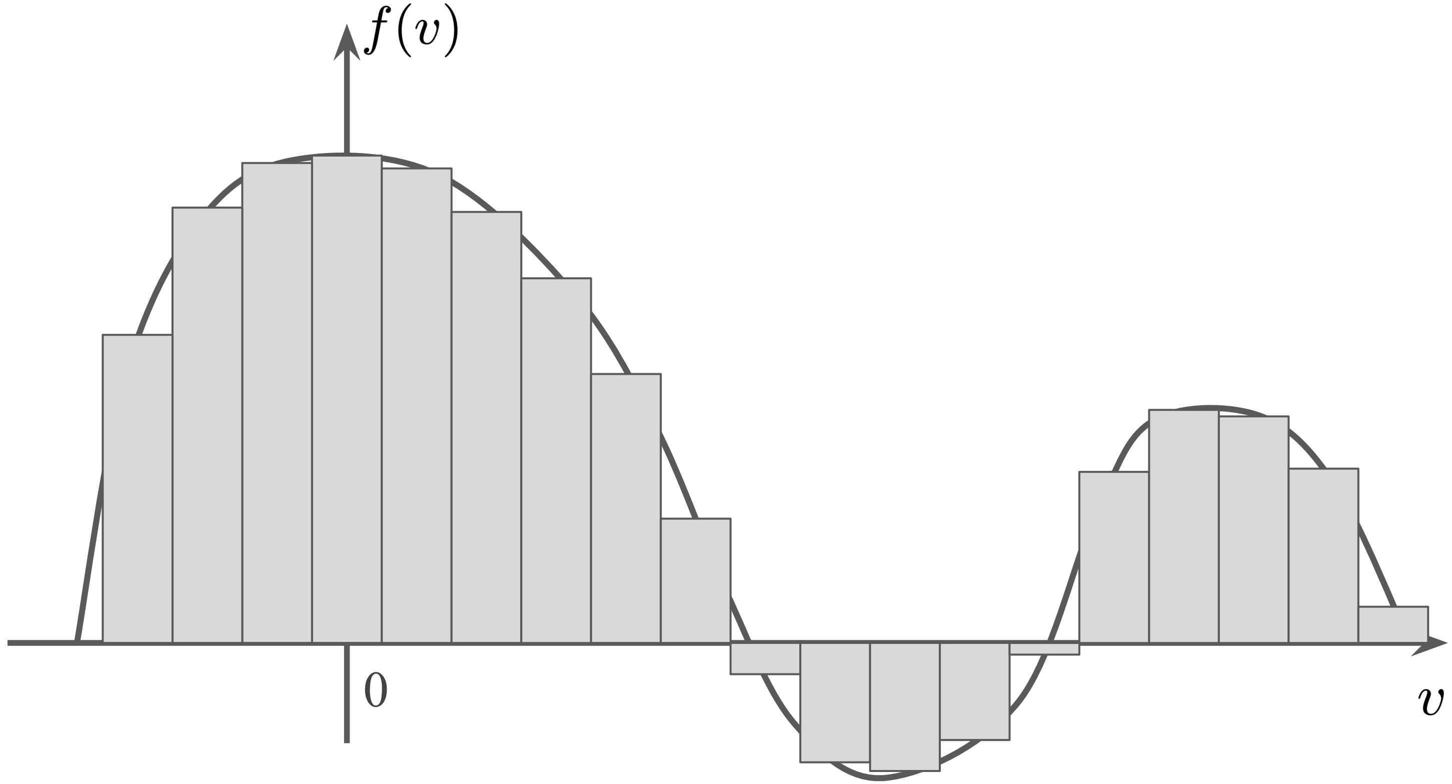
Can a continuous function be approximated by a linear combination of multiple interval indicator functions?
Yes.



UAT Proof: Sigmoid activation, 1D input, 1D output

<u>Can a continuous function be approximated by a linear combination of multiple interval indicator functions?</u>

Yes.



Because the function is continuous!

UAT Proof: Sigmoid activation, 1D input, 1D output

Can a continuous function be approximated by a linear combination of multiple interval indicator functions?
Yes.

Since $[a, b]$ is compact, we can divide it into a finite number of intervals:

$$a = x_0 < x_1 < x_2 < \ldots < x_N = b$$

such that each interval length satisfies: $x_i - x_{i-1} < \delta$

UAT Proof: Sigmoid activation, 1D input, 1D output

Can a continuous function be approximated by a linear combination of multiple interval indicator functions?

Yes.

Since $[a, b]$ is compact, we can divide it into a finite number of intervals:

$$a = x_0 < x_1 < x_2 < \ldots < x_N = b$$

such that each interval length satisfies: $x_i - x_{i-1} < \delta$

By continuity assumption, for any $\varepsilon_2 > 0$, as long as δ is sufficiently small to ensure that

$$\forall x \in [x_{i-1}, x_i], |f(x) - f(\frac{x_{i-1} + x_i}{2})| < \varepsilon_2$$

UAT Proof: Sigmoid activation, 1D input, 1D output

Can a continuous function be approximated by a linear combination of multiple interval indicator functions?
Yes.

Since $[a, b]$ is compact, we can divide it into a finite number of intervals:

$$a = x_0 < x_1 < x_2 < \ldots < x_N = b$$

such that each interval length satisfies: $x_i - x_{i-1} < \delta$

By continuity assumption, for any $\varepsilon_2 > 0$, as long as δ is sufficiently small to ensure that

$$\forall x \in [x_{i-1}, x_i], |f(x) - f(\frac{x_{i-1} + x_i}{2})| < \varepsilon_2$$

Function $g(x)$ will be able to approximate $f(x)$ sufficiently close:

$$g(x) = \sum_{i=1}^{N} f(\frac{x_{i-1} + x_i}{2}) \cdot [\sigma(w'(x - x_{i-1})) - \sigma(w'(x - x_i))]$$

ε_1 and ε_2 can be selected according to ε

$\square$

UAT Proof: Sigmoid activation, 1D input, n-D output

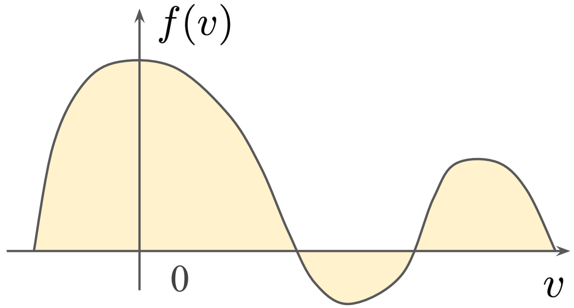
<u>**What would be different if the output is n dimensional $(n > 1)$?**</u>

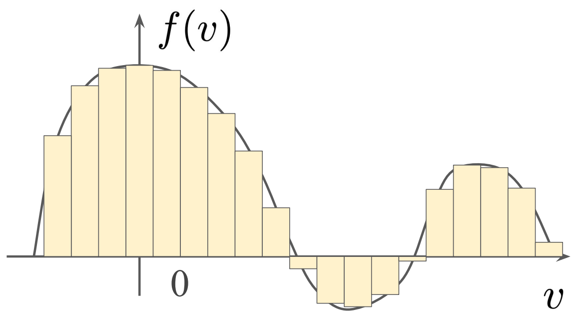
UAT Proof: Sigmoid activation, 1D input, n-D output

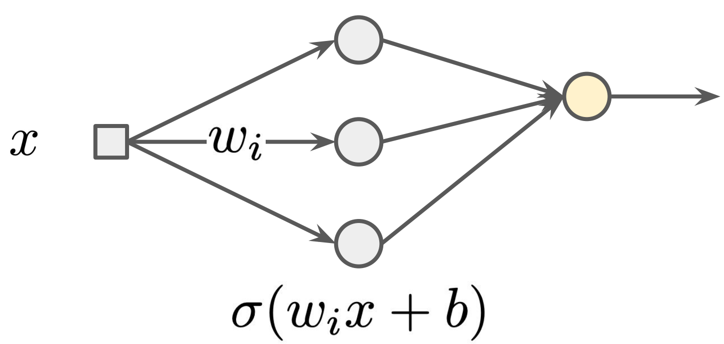
What would be different if the output is n dimensional $(n > 1)$?
This is simple, instead of using one linear combination, we can use multiple linear combinations of the same set of 1D indicator functions.

UAT Proof: Sigmoid activation, 1D input, n-D output

<u>What would be different if the output is n dimensional ($n > 1$)?</u>
This is simple, instead of using one linear combination, we can use multiple linear combinations of the same set of 1D indicator functions.

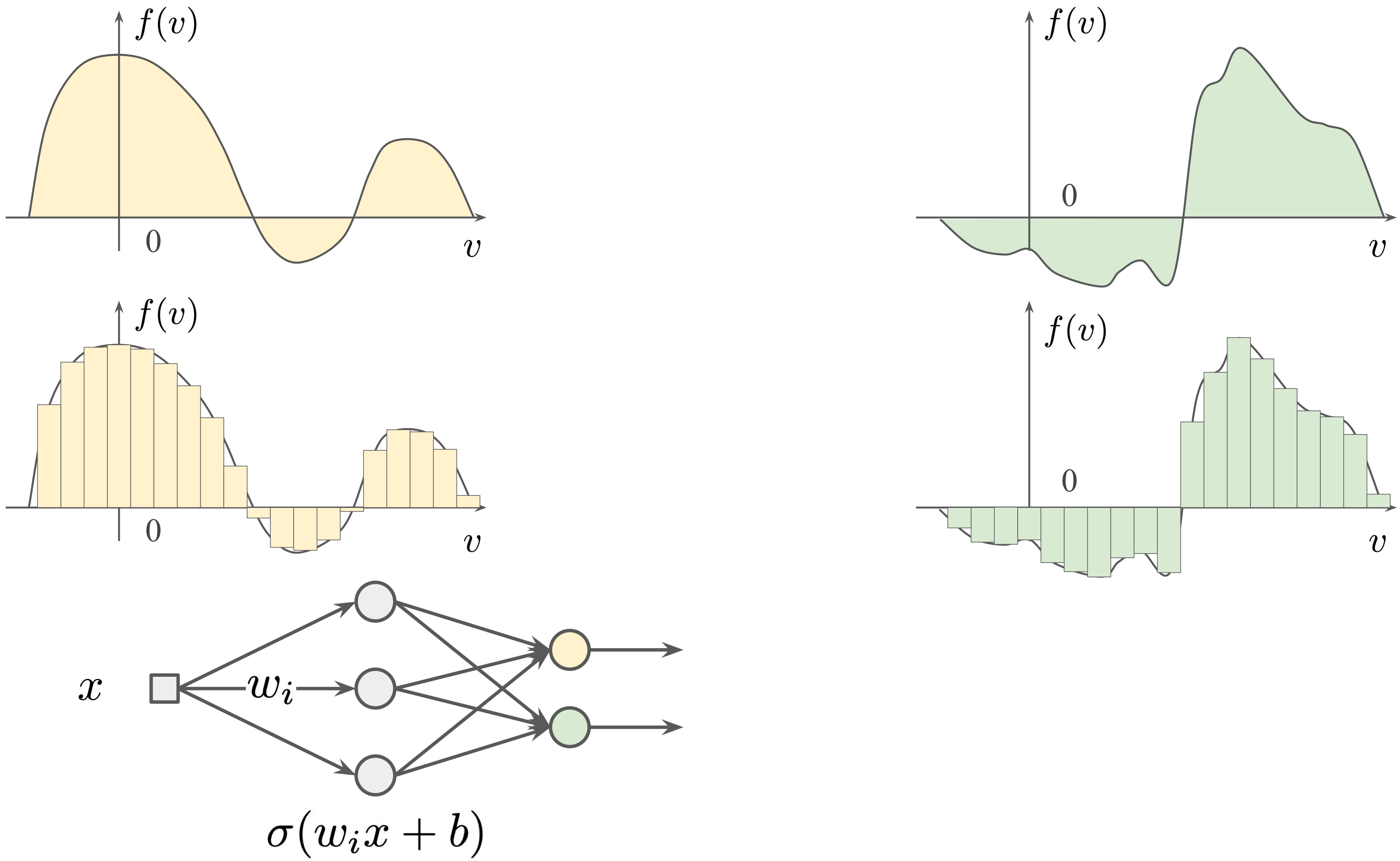






UAT Proof: Sigmoid activation, 1D input, n-D output

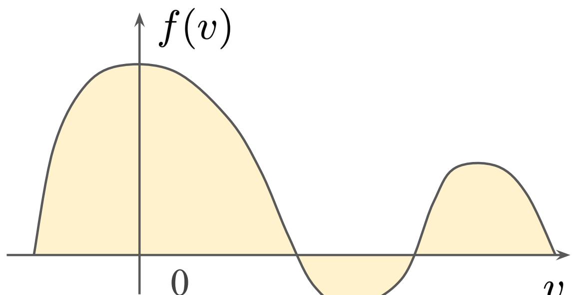
<u>What would be different if the output is n dimensional ($n > 1$)?</u>
This is simple, instead of using one linear combination, we can use multiple linear combinations of the same set of 1D indicator functions.

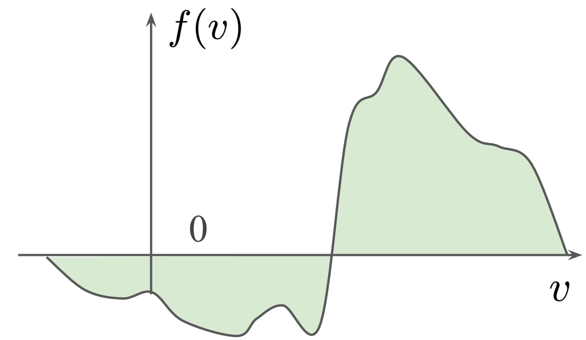


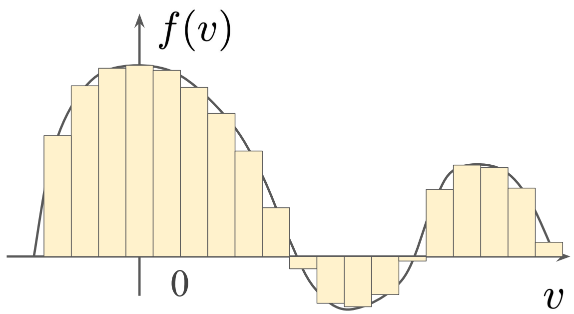
UAT Proof: Sigmoid activation, 1D input, n-D output

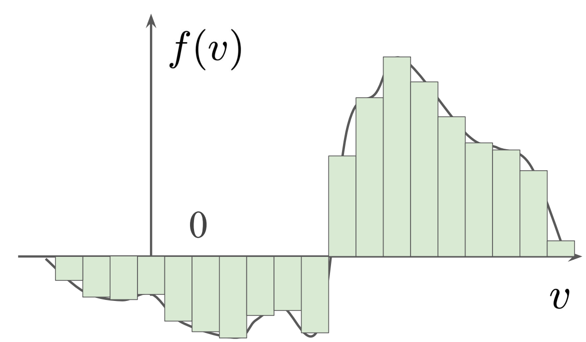
<u>What would be different if the output is n dimensional ($n > 1$)?</u>

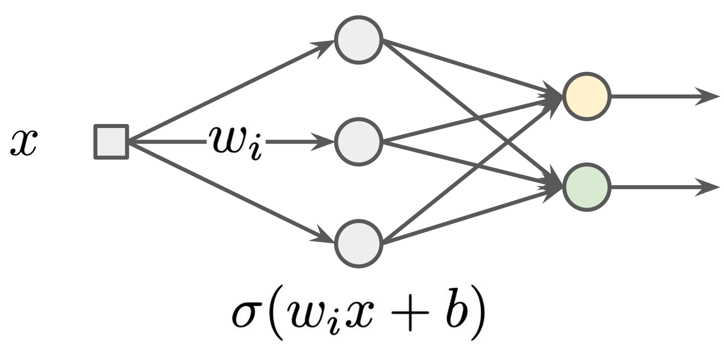
This is simple, instead of using one linear combination, we can use multiple linear combinations of the same set of 1D indicator functions.











Similar conclusions can be extended to the case where $n > 2$

UAT Proof: Sigmoid activation, m-D input, 1D output

What would be different if the input is m dimensional $(m > 1)$?

UAT Proof: Sigmoid activation, m-D input, 1D output

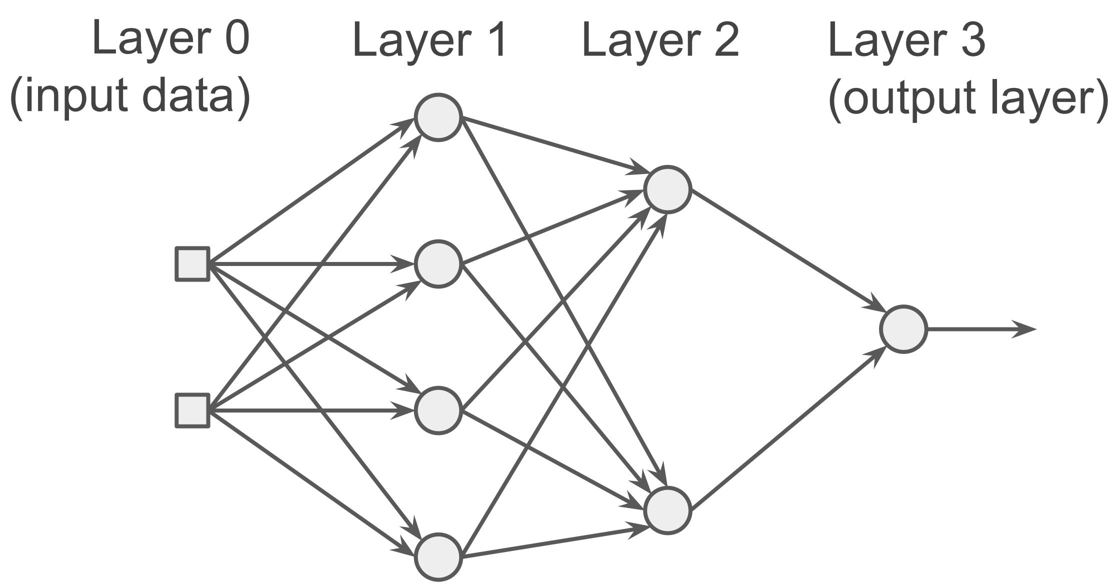
What would be different if the input is m dimensional ($m > 1$)?

It can still be approximated with a two-layer neural network with one hidden layer, but the proof is much more complex (it needs to use conclusions from advanced topics of measure / functional analysis).

UAT Proof: Sigmoid activation, m-D input, 1D output

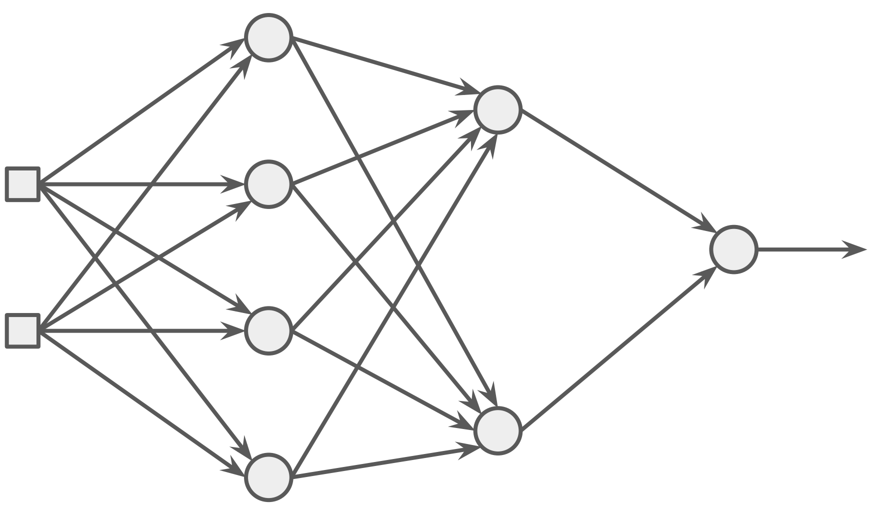
<u>What would be different if the input is m dimensional $(m > 1)$?</u>
It can still be approximated with a two-layer neural network with one hidden layer, but the proof is much more complex (it needs to use conclusions from advanced topics of measure / functional analysis).

So instead, let's prove that it can be approximated with a **three-layer neural network** with **two hidden layers**.



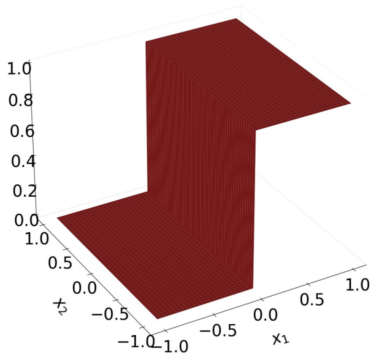
UAT Proof: Sigmoid activation, m-D input, 1D output

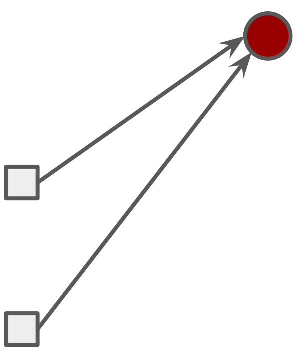
With 2D input and sufficiently large w', Sigmoid approximates a 2D step



UAT Proof: Sigmoid activation, m-D input, 1D output

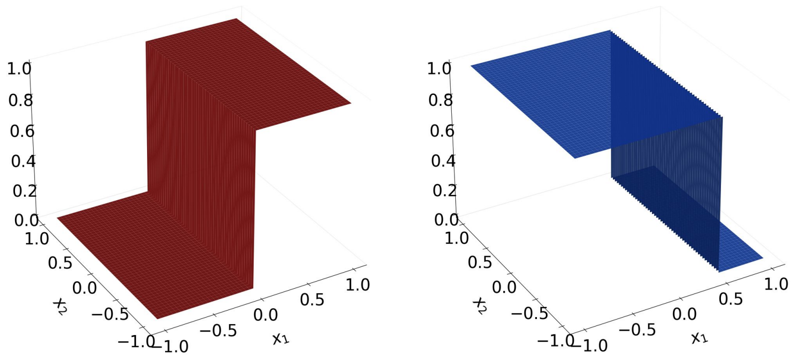
With 2D input and sufficiently large w', Sigmoid approximates a 2D step

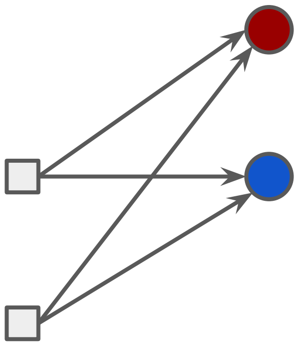




UAT Proof: Sigmoid activation, m-D input, 1D output

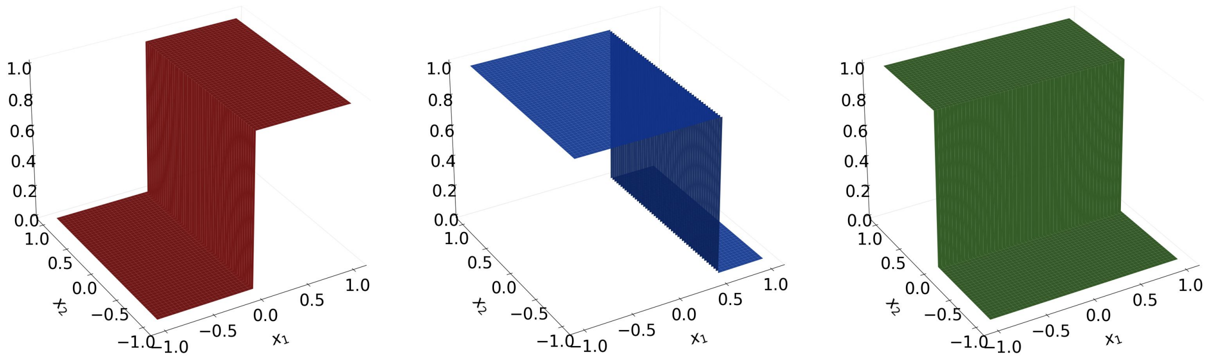
With 2D input and sufficiently large w', Sigmoid approximates a 2D step

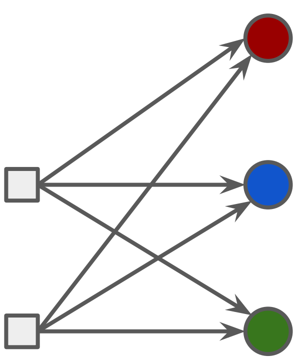




UAT Proof: Sigmoid activation, m-D input, 1D output

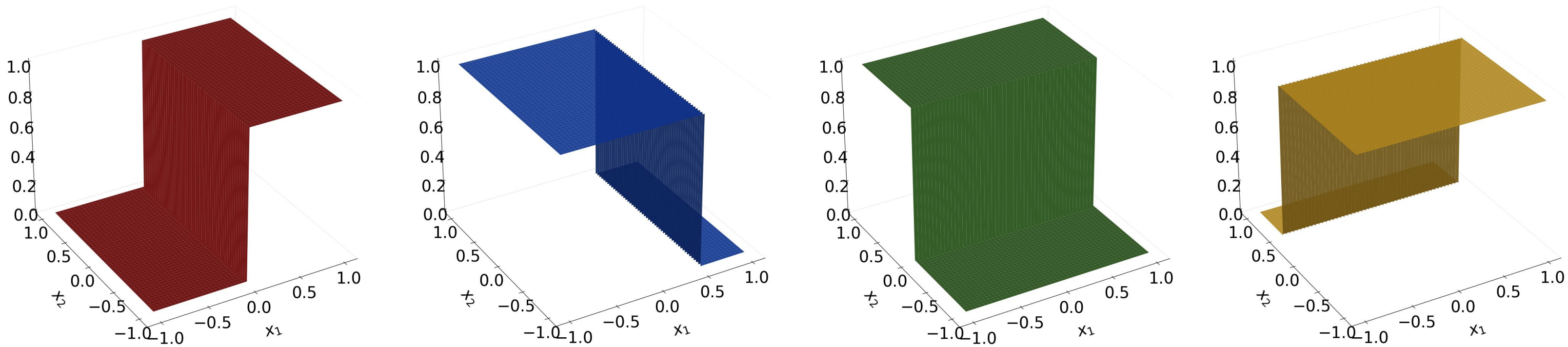
With 2D input and sufficiently large w', Sigmoid approximates a 2D step

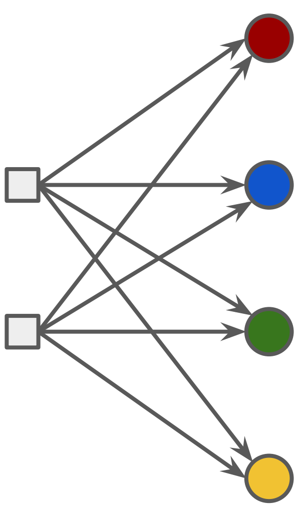




UAT Proof: Sigmoid activation, m-D input, 1D output

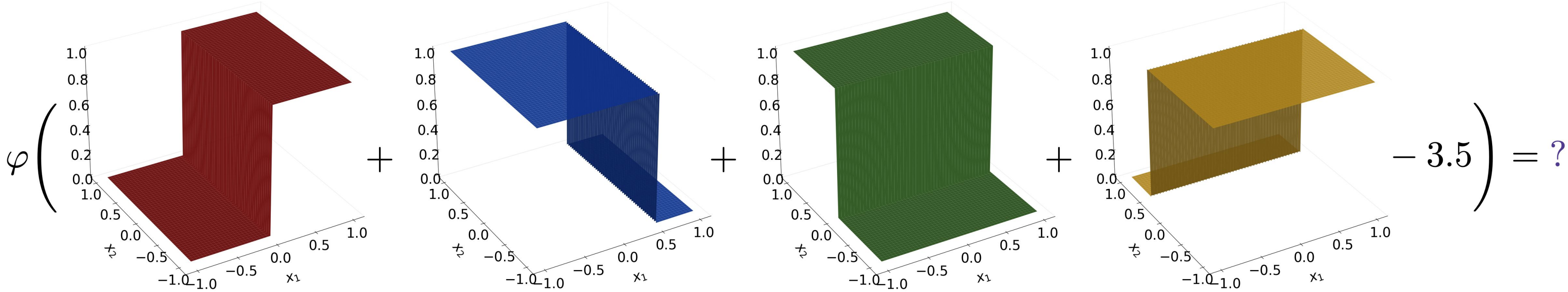
With 2D input and sufficiently large w', Sigmoid approximates a 2D step

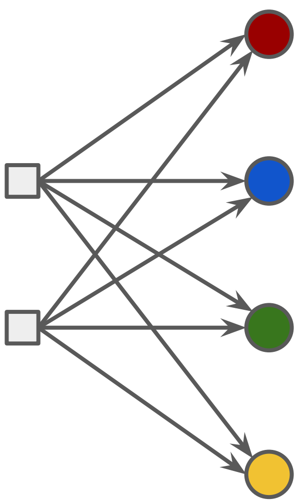




UAT Proof: Sigmoid activation, m-D input, 1D output

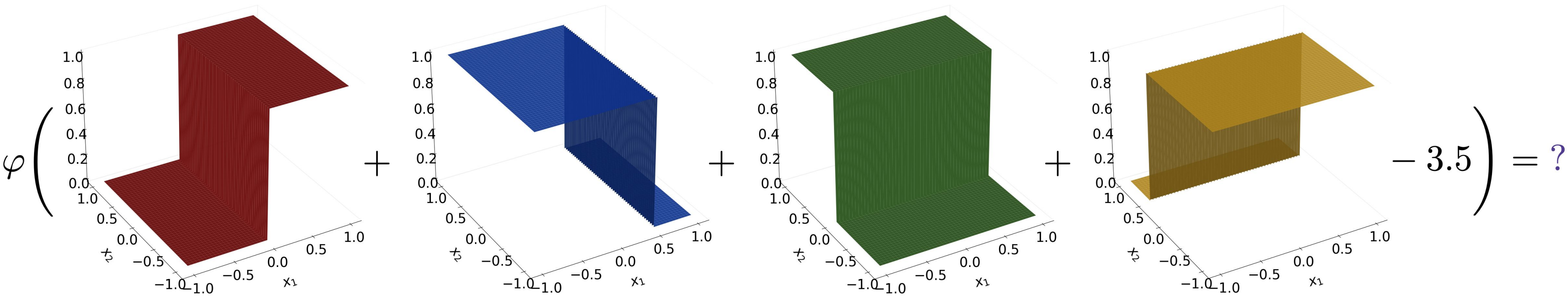
<u>What happens if we sum these four 2D step functions, subtract 3.5, and then apply a threshold function?</u> (Why do we choose 3.5?)

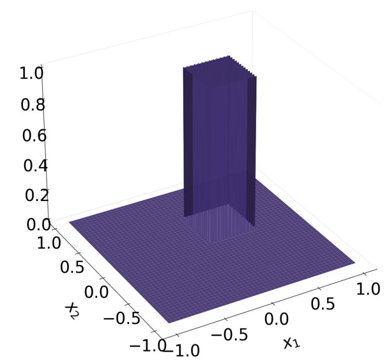


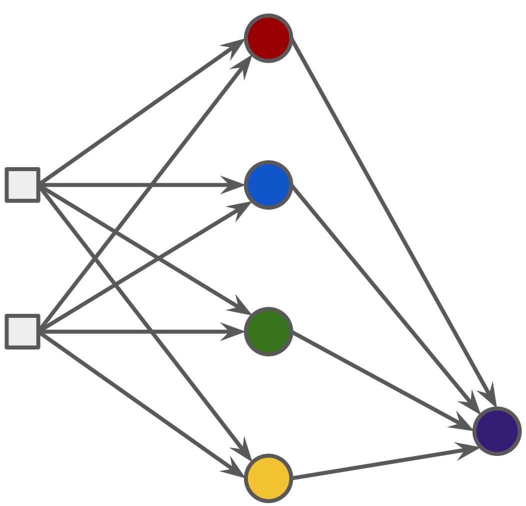


UAT Proof: Sigmoid activation, m-D input, 1D output

<u>What happens if we sum these four 2D step functions, subtract 3.5, and then apply a threshold function?</u> (Why do we choose 3.5?)





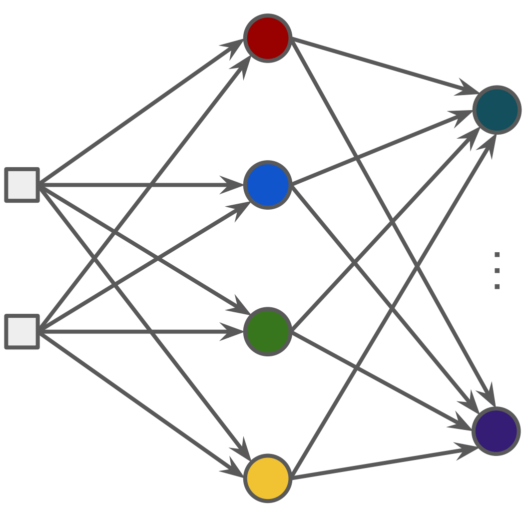


The result is an indicator function of a 2D interval:

$$\mathbf{1}_{[c_1,c_2]\times[d_1,d_2]}(x) = \begin{cases} 1, & x \in [c_1, c_2] \times [d_1, d_2] \\ 0, & x \notin [c_1, c_2] \times [d_1, d_2] \end{cases}$$

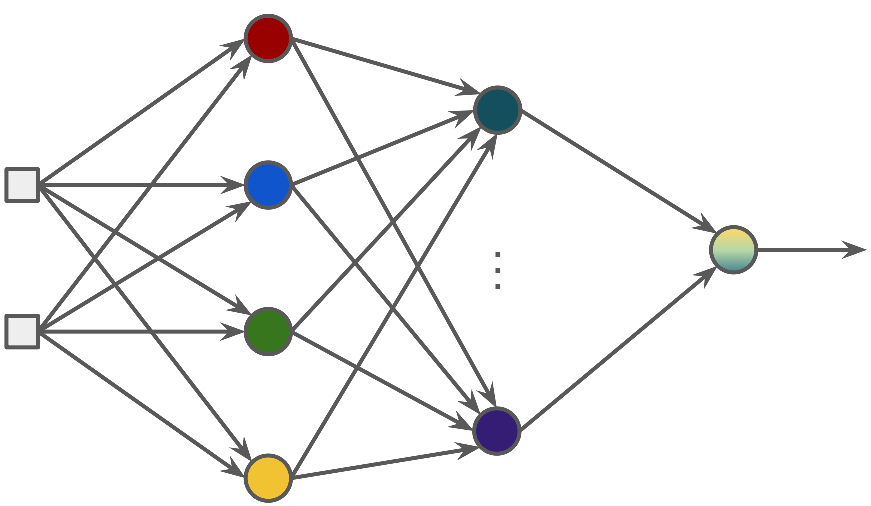
UAT Proof: Sigmoid activation, m-D input, 1D output

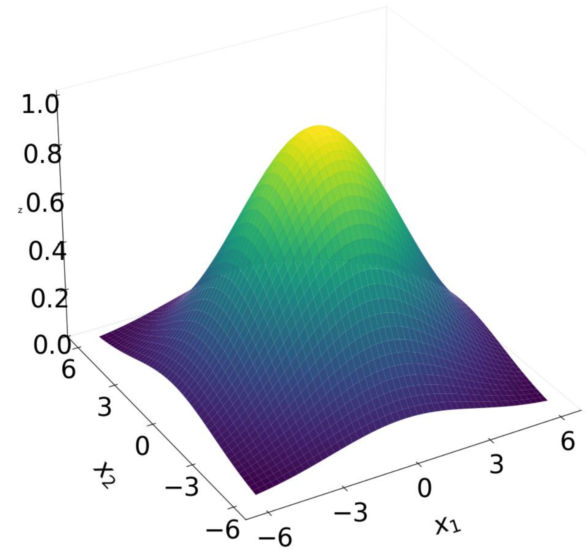
Can a linear combination of indicator function of 2D intervals approximate any 2D-input continuous function?

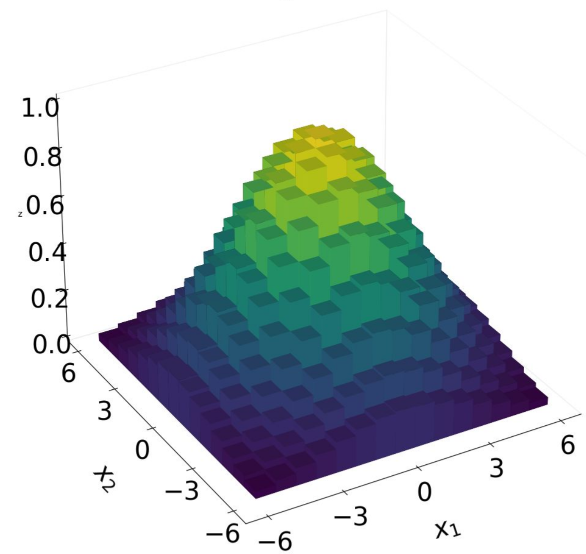


UAT Proof: Sigmoid activation, m-D input, 1D output

Can a linear combination of indicator function of 2D intervals approximate any 2D-input continuous function?
Yes.

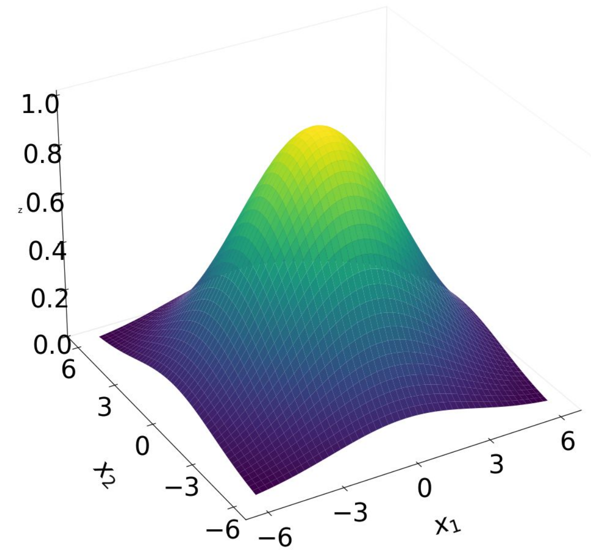




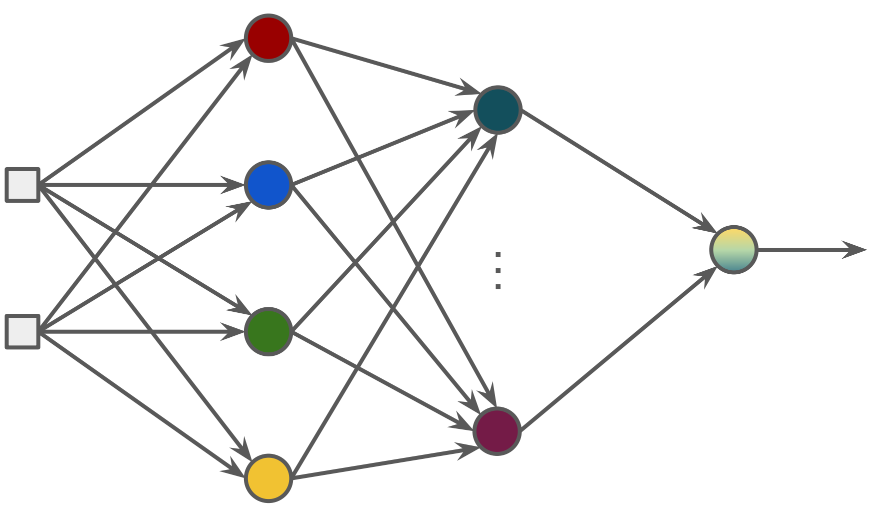


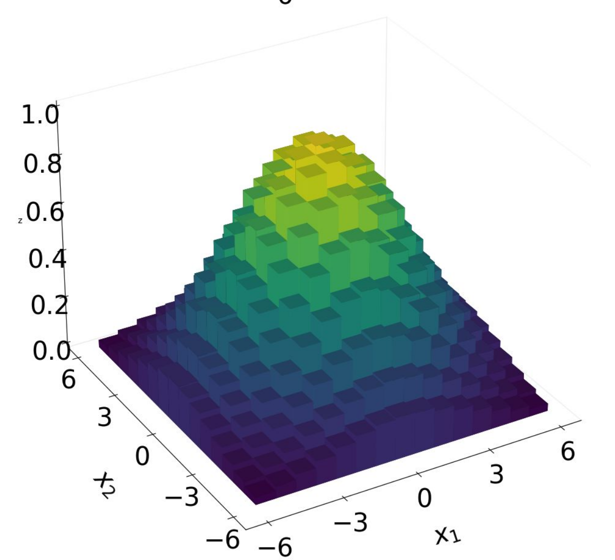
UAT Proof: Sigmoid activation, m-D input, 1D output

Can a linear combination of indicator function of 2D intervals approximate any 2D-input continuous function?
Yes.



Similar conclusions can be extended to the case where $m > 2$





A short break

We will be back in 5 mins

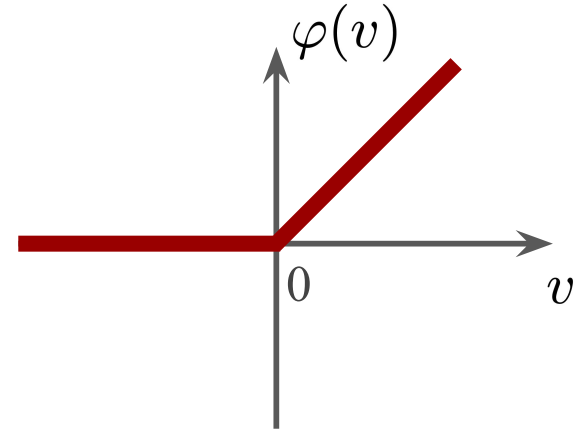
Universal Approximation Theorem (UAT): ReLU activation

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v)$$

$$= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$



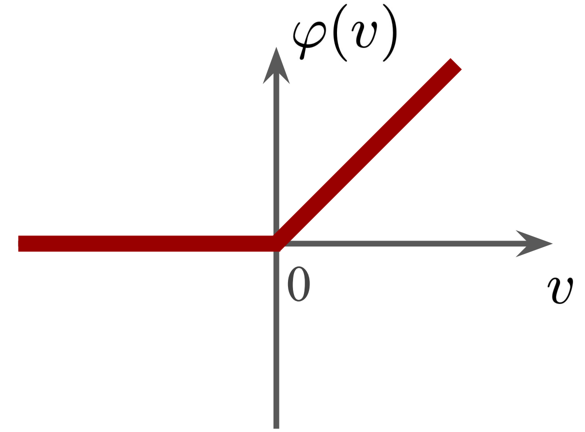
ReLU is much more widely used as activation function than Sigmoid in modern network network architectures (CNN, transformer, …)

Universal Approximation Theorem (UAT): ReLU activation

Rectified linear unit (ReLU):

$$\mathrm{ReLU}(v) = \max(0, v) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$



ReLU is much more widely used as activation function than Sigmoid in modern network network architectures (CNN, transformer, …)

<u>Can neural networks with ReLU as activation also approximate any function?</u>

Universal Approximation Theorem (UAT): ReLU activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \| f(x) - A \cdot \mathrm{ReLU}(Wx + b) \| < \varepsilon$$

Universal Approximation Theorem (UAT): ReLU activation

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \| f(x) - A \cdot \mathrm{ReLU}(Wx + b) \| < \varepsilon$$

Still, we first prove the simplest case of $n = 1$, $m = 1$

Theorem

For any continuous function f on $[a, b]$ and any $\varepsilon > 0$, there exists N, a_i, w_i, b_i such that:

$$\left| f(x) - \sum_{i=1}^{N} a_i \mathrm{ReLU}(w_i x + b) \right| < \varepsilon$$

Universal Approximation Theorem (UAT): ReLU activation

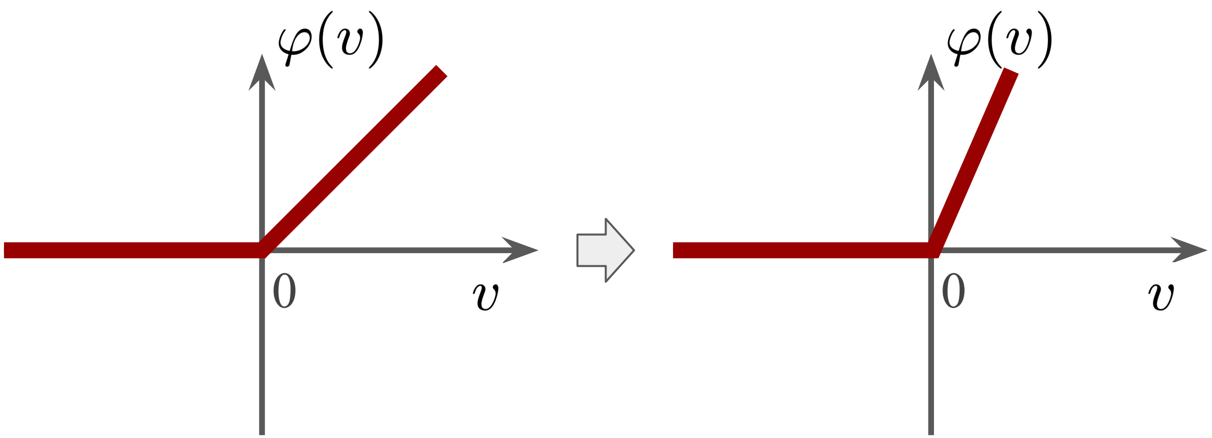
What is $a_i \mathrm{ReLU}(w_i x + b)$?

Universal Approximation Theorem (UAT): ReLU activation

What is $a_i\mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$



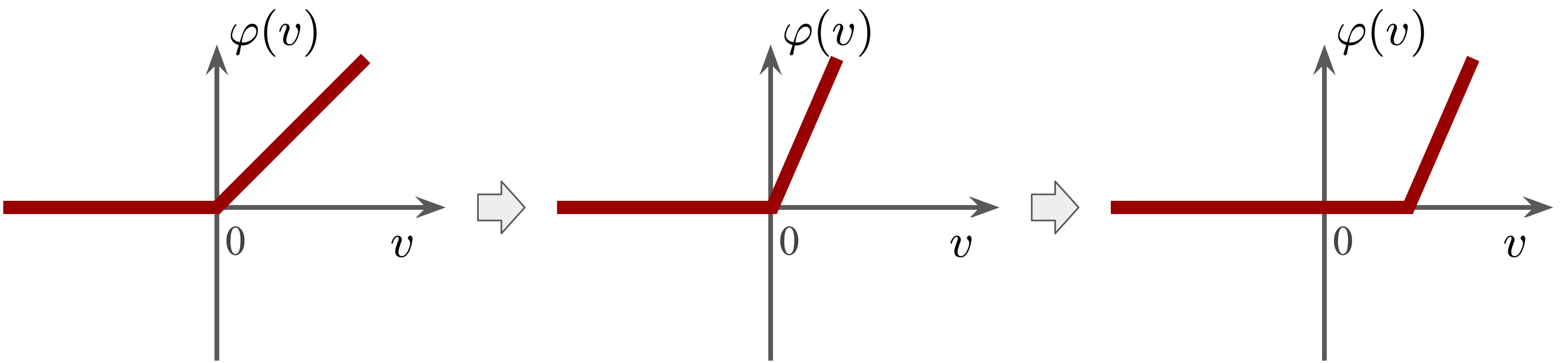
Horizontal scaling

Universal Approximation Theorem (UAT): ReLU activation

What is $a_i \mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$



Horizontal scaling

Horizontal shift

Universal Approximation Theorem (UAT): ReLU activation

What is $a_i\text{ReLU}(w_ix + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to a_iw_i
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i\text{ReLU}(w_ix + b)$?

Let's look at the summation of two $a_i\text{ReLU}(w_ix + b)$

Universal Approximation Theorem (UAT): ReLU activation

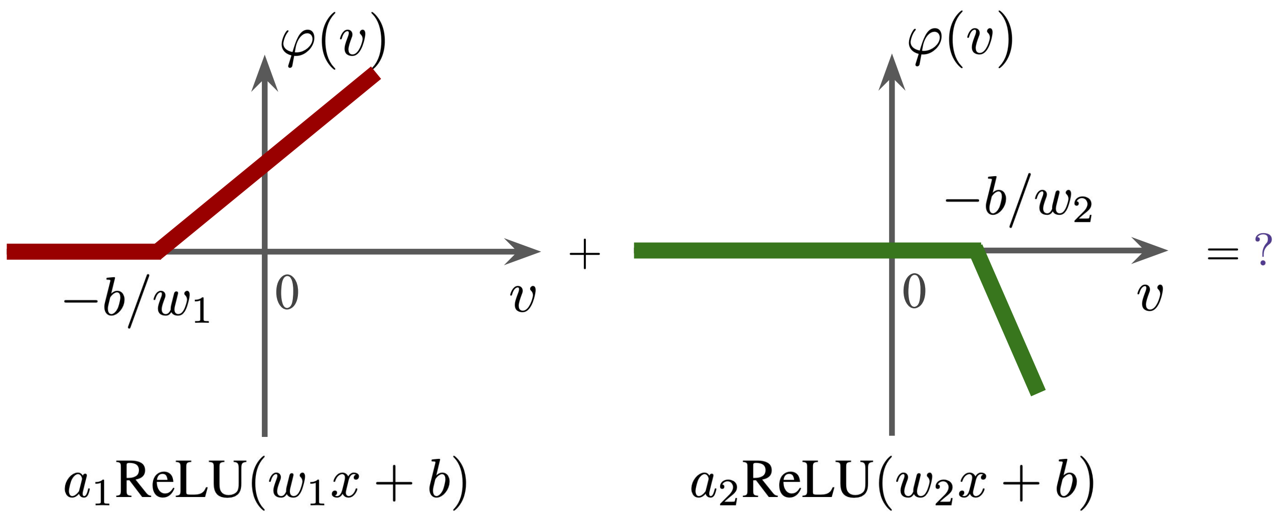
What is $a_i \mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i \mathrm{ReLU}(w_i x + b)$?

Let's look at the summation of two $a_i \mathrm{ReLU}(w_i x + b)$



Universal Approximation Theorem (UAT): ReLU activation

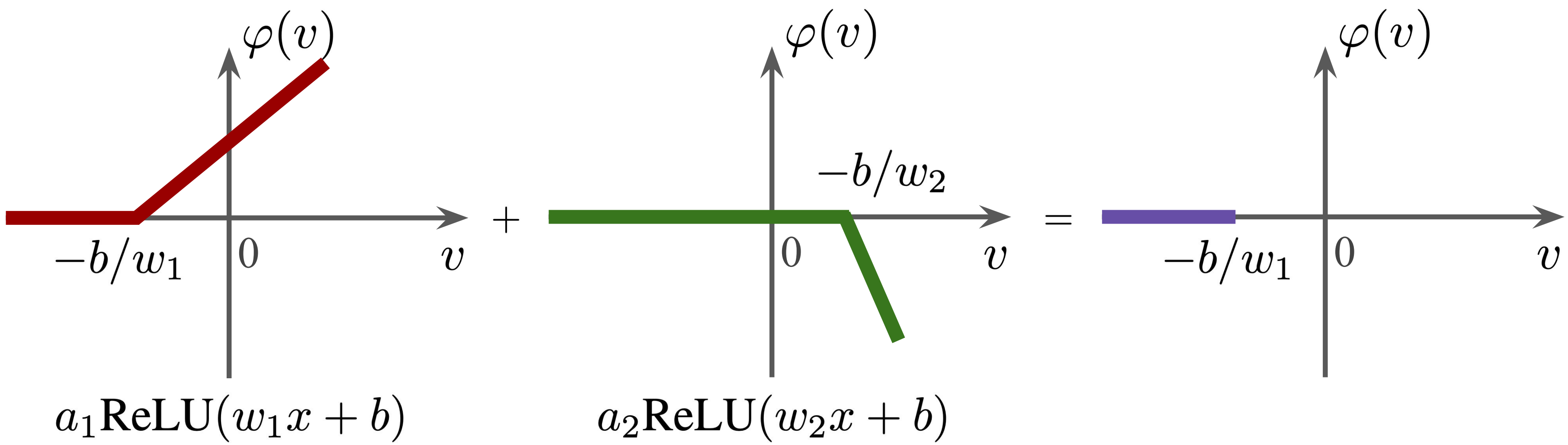
What is $a_i\mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i\mathrm{ReLU}(w_i x + b)$?

Let's look at the summation of two $a_i\mathrm{ReLU}(w_i x + b)$



Universal Approximation Theorem (UAT): ReLU activation

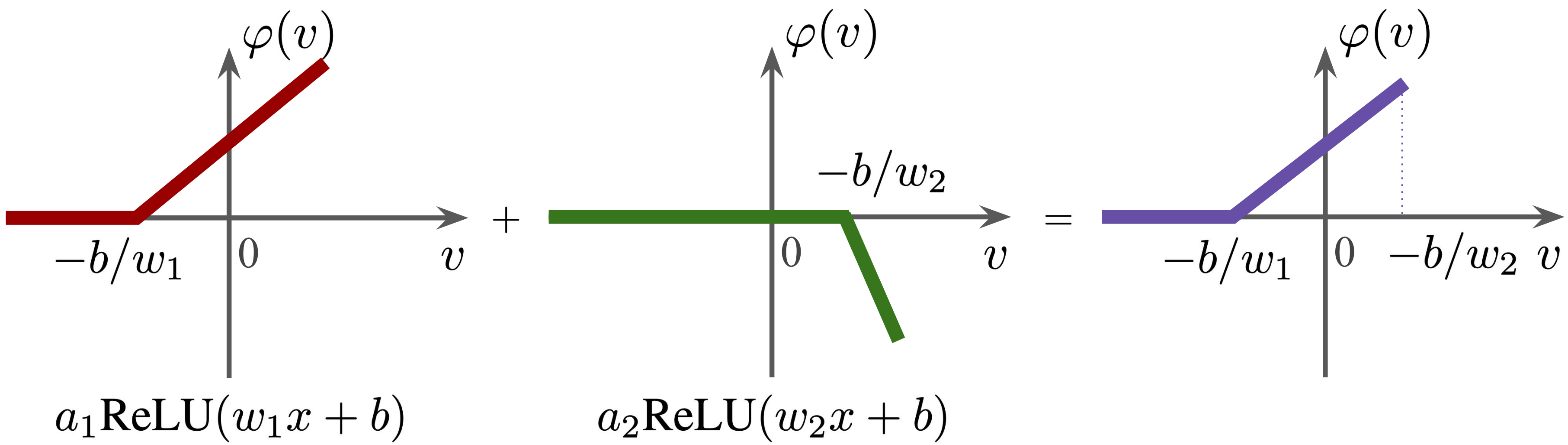
What is $a_i \mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i \mathrm{ReLU}(w_i x + b)$?

Let's look at the summation of two $a_i \mathrm{ReLU}(w_i x + b)$



Universal Approximation Theorem (UAT): ReLU activation

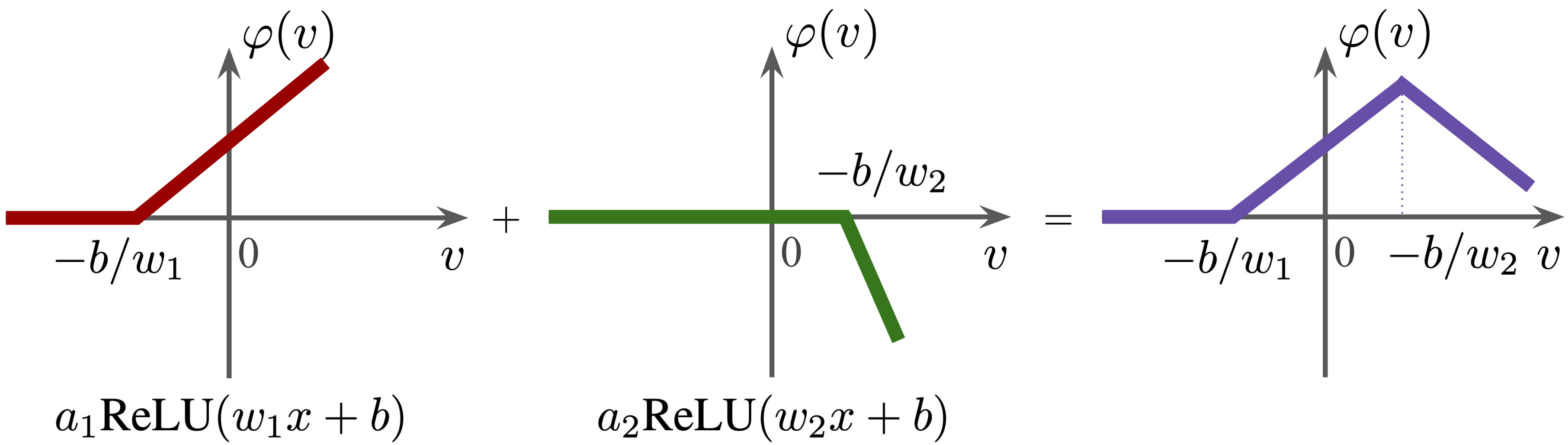
What is $a_i \text{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i \text{ReLU}(w_i x + b)$?

Let's look at the summation of two $a_i \text{ReLU}(w_i x + b)$



Universal Approximation Theorem (UAT): ReLU activation

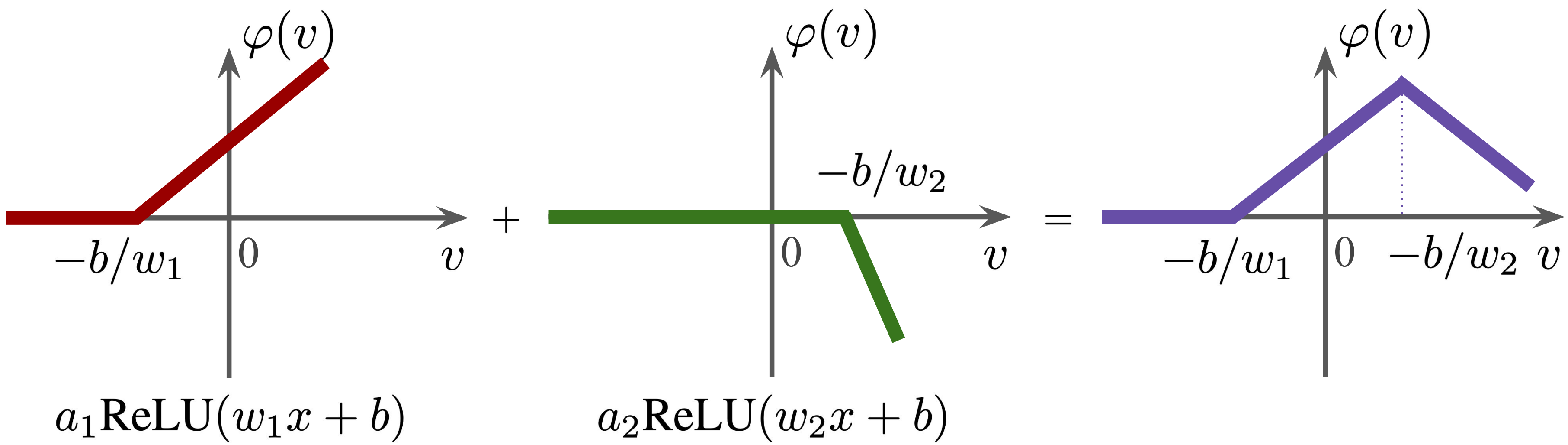
What is $a_i \mathrm{ReLU}(w_i x + b)$?

It is a ReLU function that has undergone two transformations:

- **Horizontal scaling**: changes the slope to $a_i w_i$
- **Horizontal shift**: shift the location of the "turning point" to $-b/w_i$

What is the linear combination of $a_i \mathrm{ReLU}(w_i x + b)$?

Let's look at the summation of two $a_i \mathrm{ReLU}(w_i x + b)$



The linear combination of $a_i \mathrm{ReLU}(w_i x + b)$ is a piecewise linear function!

UAT Proof: ReLU activation, 1D input, 1D output

Can a continuous function be approximated by a piecewise linear function?

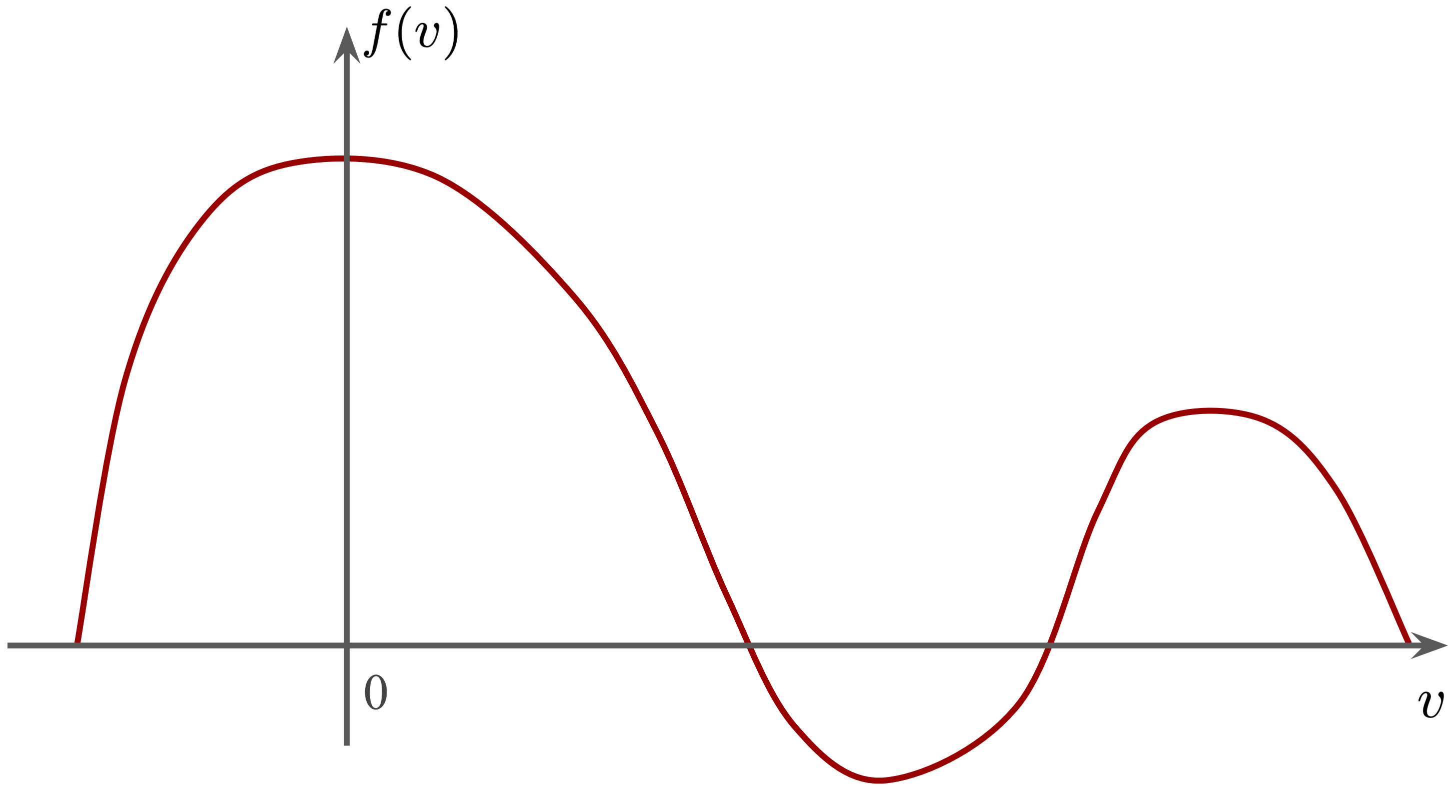
UAT Proof: ReLU activation, 1D input, 1D output

Can a continuous function be approximated by a piecewise linear function?
Yes.

UAT Proof: ReLU activation, 1D input, 1D output

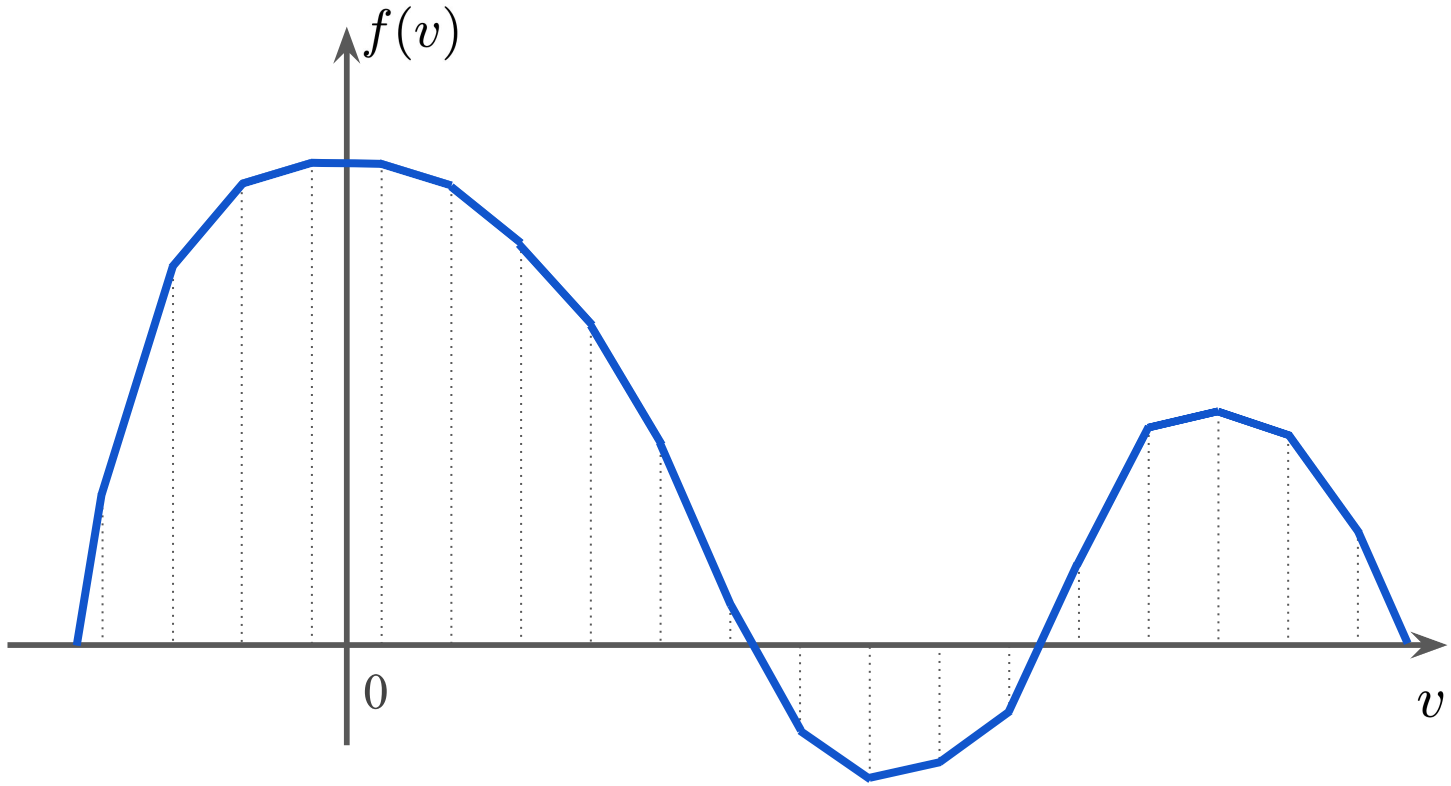
Can a continuous function be approximated by a piecewise linear function?

Yes.



UAT Proof: ReLU activation, 1D input, 1D output

Can a continuous function be approximated by a piecewise linear function?
Yes.



Because the function is continuous!

UAT Proof: ReLU activation, m-D input, n-D output

<u>**How to extend the proof to the general m-D input and n-D output case? ($m > 1, n > 1$)**</u>

UAT Proof: ReLU activation, m-D input, n-D output

How to extend the proof to the general m-D input and n-D output case? ($m > 1, n > 1$)

The 2D input and 2D output case will be left as an exercise in the assignment

UAT Proof: ReLU activation, m-D input, n-D output

How to extend the proof to the general m-D input and n-D output case? ($m > 1, n > 1$)

The 2D input and 2D output case will be left as an exercise in the assignment

Reminder 1*: Recall how neural networks with sigmoid activation handle multi-dimensional outputs.*

UAT Proof: ReLU activation, m-D input, n-D output

How to extend the proof to the general m-D input and n-D output case? ($m > 1, n > 1$)

The 2D input and 2D output case will be left as an exercise in the assignment

Reminder 1*: Recall how neural networks with sigmoid activation handle multi-dimensional outputs.*

Reminder 2*: Does a piecewise linear function exist for 2D inputs?*
If so, how can it be used for function approximation?

How to measure approximation to target?

We proved that neural networks can approximate continuous functions

But how do we measure “**good approximation**”?

How to measure approximation to target?

We proved that neural networks can approximate continuous functions

But how do we measure "**good approximation**"?

Loss functions quantifies the output discrepancy compared to the desired prediction outputs

How to measure approximation to target?

We proved that neural networks can approximate continuous functions

But how do we measure “**good approximation**”?

Loss functions quantifies the output discrepancy compared to the desired prediction outputs

During the training of neural networks, we want to find the neural network parameters (weights and biases) that can minimize the <u>total loss</u>, i.e. the <u>sum</u> of loss over training data.

How to measure approximation to target?

We proved that neural networks can approximate continuous functions

But how do we measure “**good approximation**”?

Loss functions quantifies the output discrepancy compared to the desired prediction outputs

During the training of neural networks, we want to find the neural network parameters (weights and biases) that can minimize the total loss, i.e. the sum of loss over training data. *(why not the product of loss over training data?)*

Frequently used loss functions

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

Frequently used loss functions

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

Frequently used loss functions

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1 - y) \log(1 - \sigma(f_\theta(x)))$$

Frequently used loss functions

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1 - y) \log(1 - \sigma(f_\theta(x)))$$

4. Softmax cross-entropy loss for multi-class classification:

$$\mathcal{L}(f_\theta(x), y) = -\sum_{k=1}^{K} y_k \log(\frac{e^{f_k(x)}}{\sum_{j=1}^{K} e^{f_j(x)}})$$

where $y_k = \begin{cases} 1, & k = c \\ 0, & k \neq c \end{cases}$, and $c \in \{1, 2, \ldots, K\}$ is the true class

What is exactly being quantified by the loss functions?

<u>Supervised Learning:</u>

Suppose $y_i = f_\theta(x_i), (x_i, y_i) \sim p_{\text{data}}(x, y)$

Goal is the find θ such that $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim p_{\text{data}}} [\mathcal{L}(f_\theta(x), y)]$

What is exactly being quantified by the loss functions?

Supervised Learning:
Suppose $y_i = f_\theta(x_i), (x_i, y_i) \sim p_{\text{data}}(x, y)$
Goal is the find θ such that $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim p_{\text{data}}} [\mathcal{L}(f_\theta(x), y)]$

From a statistical perspective:
Training a neural network can be viewed as learning a conditional probabilistic model that maps inputs x_i to a distribution over outputs y_i, i.e. learning to model $p_\theta(y \mid x)$

What is exactly being quantified by the loss functions?

Supervised Learning:
Suppose $y_i = f_\theta(x_i), (x_i, y_i) \sim p_{\text{data}}(x, y)$
Goal is the find θ such that $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim p_{\text{data}}} [\mathcal{L}(f_\theta(x), y)]$

From a statistical perspective:
Training a neural network can be viewed as learning a conditional probabilistic model that maps inputs x_i to a distribution over outputs y_i, i.e. learning to model $p_\theta(y \mid x)$

Loss corresponds to the negative log-likelihood of θ

What is exactly being quantified by the loss functions?

Supervised Learning:
Suppose $y_i = f_\theta(x_i), (x_i, y_i) \sim p_{\text{data}}(x, y)$
Goal is the find θ such that $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y)\sim p_{\text{data}}} [\mathcal{L}(f_\theta(x), y)]$

From a statistical perspective:
Training a neural network can be viewed as learning a conditional probabilistic model that maps inputs x_i to a distribution over outputs y_i, i.e. learning to model $p_\theta(y \mid x)$

Loss corresponds to the negative log-likelihood of θ

- *Probability: given parameter θ, the probability of output y_i?*
  - *It is a function of y_i*
- *Likelihood: given output y_i, the likelihood of parameter θ?*
  - *It is a function of θ*

Why does loss correspond to negative log-likelihood?

The loss quantifies the discrepancy between prediction and observation, minimizing loss should maximize the likelihood of θ

Why does loss correspond to negative log-likelihood?

The loss quantifies the discrepancy between prediction and observation, minimizing loss should maximize the likelihood of θ

Given a dataset $\{(x_i, y_i)\}_{i=1}^N$, the **likelihood** of parameters θ is:

$$p_\theta(y \mid x) = \prod_{i=1}^{N} p_\theta(y_i \mid x_i) = \exp\left(\sum_{i=1}^{N} \log p_\theta(y_i \mid x_i)\right)$$

Why does loss correspond to negative log-likelihood?

The loss quantifies the discrepancy between prediction and observation, minimizing loss should maximize the likelihood of θ

Given a dataset $\{(x_i, y_i)\}_{i=1}^{N}$, the **likelihood** of parameters θ is:

$$p_\theta(y \mid x) = \prod_{i=1}^{N} p_\theta(y_i \mid x_i) = \exp\left(\sum_{i=1}^{N} \log p_\theta(y_i \mid x_i)\right)$$

With likelihood-based loss, the training of neural network is essentially <u>Maximum Likelihood Estimation (MLE)</u>

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} -\log p_\theta(y_i \mid x_i)$$

Why does loss correspond to negative log-likelihood?

The loss quantifies the discrepancy between prediction and observation, minimizing loss should maximize the likelihood of θ

Given a dataset $\{(x_i, y_i)\}_{i=1}^N$, the **likelihood** of parameters θ is:

$$p_\theta(y \mid x) = \prod_{i=1}^{N} p_\theta(y_i \mid x_i) = \exp\left(\sum_{i=1}^{N} \log p_\theta(y_i \mid x_i)\right)$$

With likelihood-based loss, the training of neural network is essentially <u>Maximum Likelihood Estimation (MLE)</u>

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} -\log p_\theta(y_i \mid x_i)$$

Therefore, the loss of a data sample corresponds to the negative log-likelihood:

$$\ell_i(\theta) = -\log p_\theta(y_i \mid x_i)$$

Connecting loss function to probabilistic distribution

Which probabilistic distribution $p_\theta(y \mid x)$ does each loss function $\mathcal{L}$ correspond to (via negative log-likelihood)?

Connecting loss function to probabilistic distribution

Which probabilistic distribution $p_\theta(y \mid x)$ does each loss function $\mathcal{L}$ correspond to (via negative log-likelihood)?

When we decide to use a specific loss function, what underlying distribution of the data are we assuming?

Connecting loss function to probabilistic distribution

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

Connecting loss function to probabilistic distribution

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

We have:

$$-\log p_\theta(y \mid x) = c_1 \cdot (y - f_\theta(x))^2 + c_2$$

Connecting loss function to probabilistic distribution

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

We have:

$$-\log p_\theta(y \mid x) = c_1 \cdot (y - f_\theta(x))^2 + c_2$$

$$p_\theta(y \mid x) = e^{-c_2} \cdot \exp(-c_1 \cdot (y - f_\theta(x))^2)$$

Connecting loss function to probabilistic distribution

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

We have:

$$-\log p_\theta(y \mid x) = c_1 \cdot (y - f_\theta(x))^2 + c_2$$

$$\begin{aligned} p_\theta(y \mid x) &= e^{-c_2} \cdot \exp(-c_1 \cdot (y - f_\theta(x))^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\Big(- \frac{(y - f_\theta(x))^2}{2\sigma^2} \Big) \end{aligned}$$

Connecting loss function to probabilistic distribution

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

We have:

$$-\log p_\theta(y \mid x) = c_1 \cdot (y - f_\theta(x))^2 + c_2$$

$$\begin{aligned} p_\theta(y \mid x) &= e^{-c_2} \cdot \exp(-c_1 \cdot (y - f_\theta(x))^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\Big(-\frac{(y - f_\theta(x))^2}{2\sigma^2}\Big) \end{aligned}$$

This means when using L2 loss for regression, we assume the distribution of y given x is a **Gaussian distribution**!

And the **mean** of the Gaussian distribution is $f_\theta(x)$!

Connecting loss function to probabilistic distribution

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

Connecting loss function to probabilistic distribution

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

We have

$$-\log p_\theta(y \mid x) = c_1 \cdot \|y - f_\theta(x)\| + c_2$$

Connecting loss function to probabilistic distribution

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

We have

$$-\log p_\theta(y \mid x) = c_1 \cdot \|y - f_\theta(x)\| + c_2$$

$$p_\theta(y \mid x) = e^{-c_2} \cdot \exp(-c_1 \cdot \|y - f_\theta(x)\|_1)$$

Connecting loss function to probabilistic distribution

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

We have

$$-\log p_\theta(y \mid x) = c_1 \cdot \|y - f_\theta(x)\| + c_2$$

$$\begin{aligned} p_\theta(y \mid x) &= e^{-c_2} \cdot \exp(-c_1 \cdot \|y - f_\theta(x)\|_1) \\ &= \frac{1}{2b} \cdot \exp\Big(- \frac{\|y - f_\theta(x)\|_1}{b} \Big) \end{aligned}$$

Connecting loss function to probabilistic distribution

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

We have

$$-\log p_\theta(y \mid x) = c_1 \cdot \|y - f_\theta(x)\| + c_2$$

$$\begin{aligned} p_\theta(y \mid x) &= e^{-c_2} \cdot \exp(-c_1 \cdot \|y - f_\theta(x)\|_1) \\ &= \frac{1}{2b} \cdot \exp\left(-\frac{\|y - f_\theta(x)\|_1}{b}\right) \end{aligned}$$

This means when using L1 loss for regression, we assume the distribution of y given x is a **Laplace distribution**!

And the **mean** of the Laplace distribution is $f_\theta(x)$!

Connecting loss function to probabilistic distribution

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1-y)\log(1-\sigma(f_\theta(x)))$$

Connecting loss function to probabilistic distribution

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1-y)\log(1-\sigma(f_\theta(x)))$$

By setting $p = \sigma(f_\theta(x))$, we have

$$-\log p_\theta(y \mid x) = -y \log p - (1-y)\log(1-p)$$

$$p_\theta(y \mid x) = p^y (1-p)^{1-y}$$

Connecting loss function to probabilistic distribution

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1-y)\log(1-\sigma(f_\theta(x)))$$

By setting $p = \sigma(f_\theta(x))$, we have

$$-\log p_\theta(y \mid x) = -y \log p - (1-y)\log(1-p)$$

$$p_\theta(y \mid x) = p^y (1-p)^{1-y}$$

This means when using logistic loss for binary classification, we assume the distribution of y given x is a **Bernoulli distribution**!

And the **probability of** $y = 1$ is $\sigma(f_\theta(x))$!

Connecting loss function to probabilistic distribution

4. Softmax cross-entropy loss for multi-class classification:

$$\mathcal{L}(f_\theta(x), y) = -\sum_{k=1}^{K} y_k \log(\frac{e^{f_k(x)}}{\sum_{j=1}^{K} e^{f_j(x)}})$$

where $y_k = \begin{cases} 1, & k = c \\ 0, & k \neq c \end{cases}$, and $c \in \{1, 2, \ldots, K\}$ is the true class

Connecting loss function to probabilistic distribution

4. Softmax cross-entropy loss for multi-class classification:

$$\mathcal{L}(f_\theta(x), y) = -\sum_{k=1}^{K} y_k \log\left(\frac{e^{f_k(x)}}{\sum_{j=1}^{K} e^{f_j(x)}}\right)$$

where $y_k = \begin{cases} 1, & k = c \\ 0, & k \neq c \end{cases}$, and $c \in \{1, 2, \ldots, K\}$ is the true class

What is the assumption of the distribution of y when using cross-entropy loss for multi-class classification?

Will be left as an exercise in the assignment!

Thanks