



Introduction to Neural Networks

What is a neural network?

Artificial neural networks – algorithm inspired by neuronal brain activity in which input signals are processed and further transmitted amongst artificial neurons/nodes

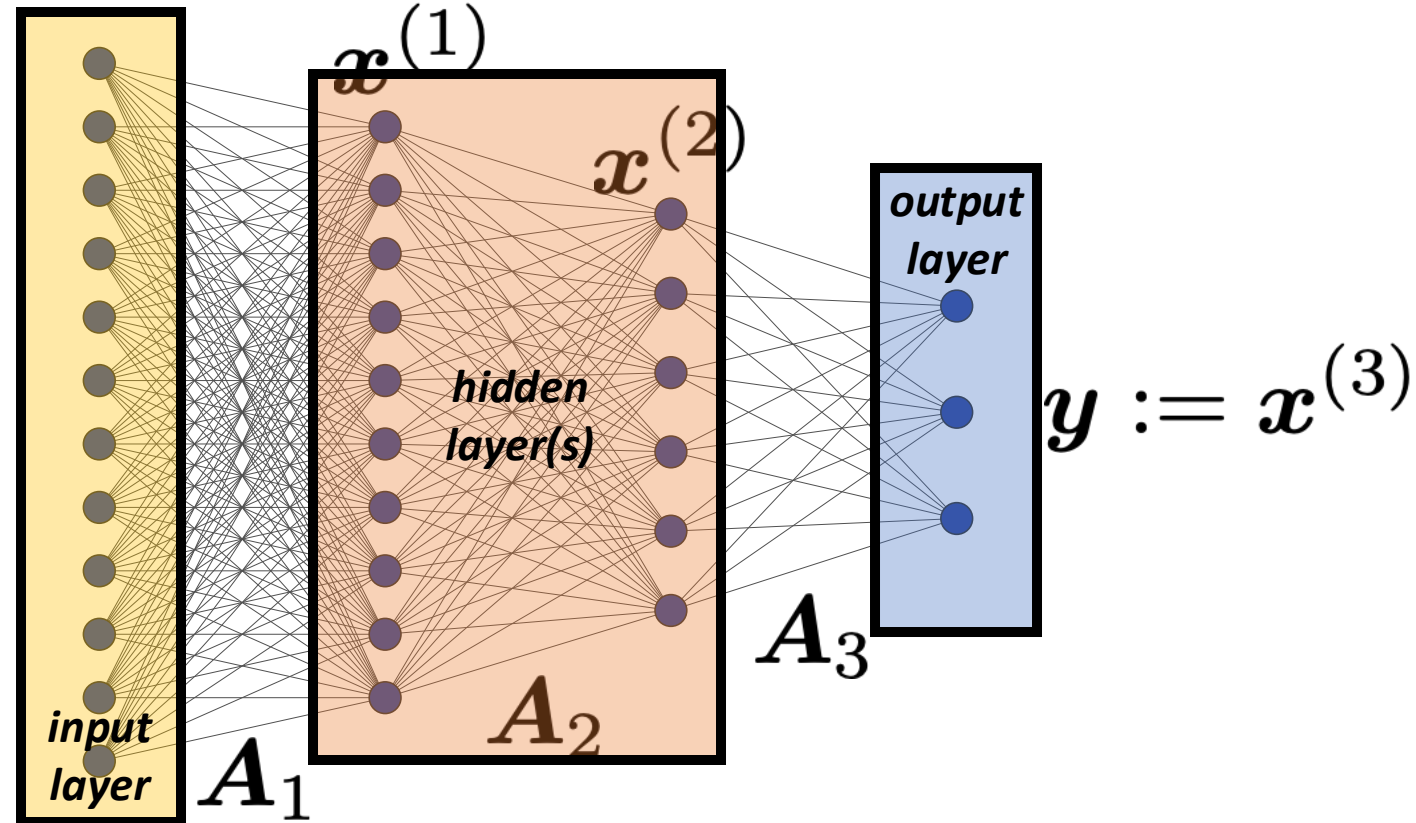
e.g., 2-layer “dense” network

*conceptually, the ANN is a
(potentially complex) function
that converts features into labels*

$$\mathbf{x} := \mathbf{x}^{(0)}$$

$$\mathbf{A}_k[i, j] = a^{(k)}(\mathbf{x}_i^{(k-1)} \rightarrow \mathbf{x}_j^{(k)})$$

*weights connecting neurons in
adjacent layers*



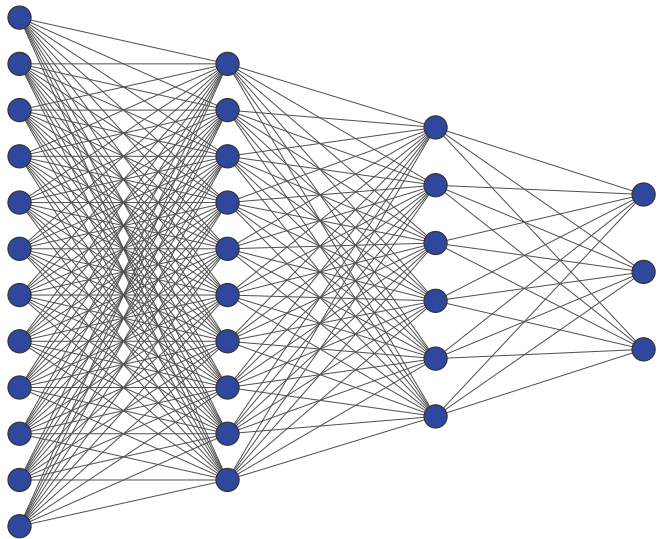
What is a neural network?

Artificial neural networks – algorithm inspired by neuronal brain activity in which input signals are processed and further transmitted amongst artificial neurons/nodes

Training of a neural network takes the general form of optimizing some composite function with regularization

$$\operatorname{argmin}_{\{\mathbf{A}_j\}} [\mathcal{E}(f_M(\mathbf{A}_m, f_{m-1}(\mathbf{A}_{m-1}, \dots, f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x})) \dots))) + \lambda g(\{\mathbf{A}_j\})]$$

allows for data representation/description *prevents overfitting*



- highly underdetermined system with large(!) number of parameters
- model is set by the number of layers
- connections can be dense or sparse, and they can have linear/non-linear “activations”

composite nature of function plus many parameters → optimization by ***stochastic gradient descent*** and ***backpropagation***

My, how times have changed...

Artificial neural networks (and variants) are currently the workhorse machine learning algorithm in science and engineering, but that's not always been the case

Top 10 Algorithms in Data Mining 2008

IEEE International Conference on Data Mining

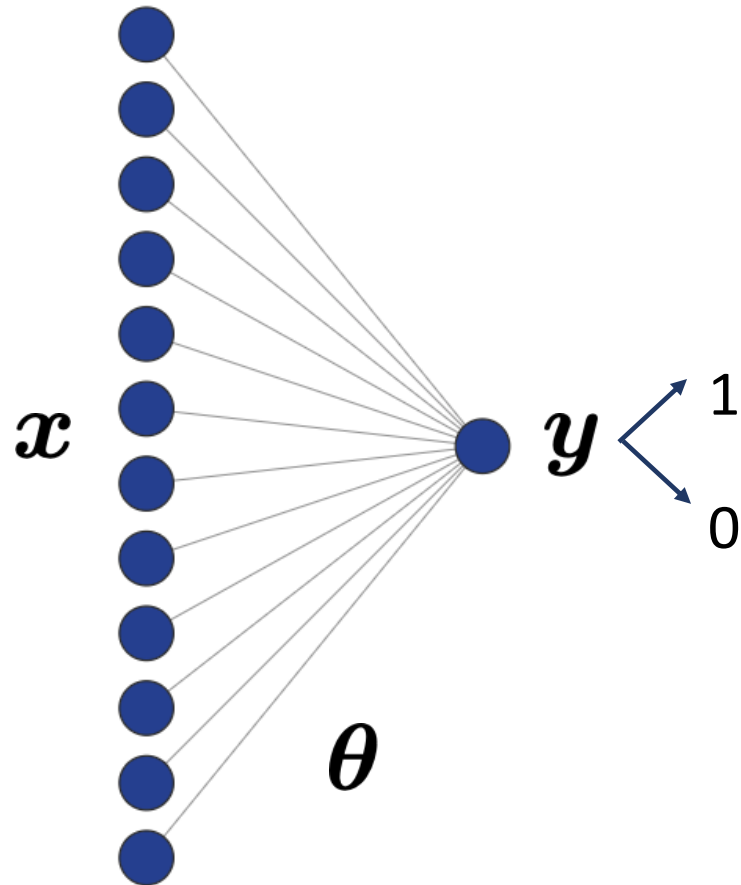
1. ***k-means*** (unsupervised method for clustering data into k groups)
2. **Expectation-Maximization, mixture models** (unsupervised learning to describe data from a mixture of probability distributions)
3. ***Support Vector Machines*** (supervised learning algorithm involving high-dimensional data cutting)
4. ***Classification and Regression Tree*** (split data along different variables to produce branches of a tree)
5. ***k-nearest Neighbors*** (supervised learning to determine labels based on k nearest neighbors)
6. ***Naïve Bayes*** (supervised learning based on conditional probabilities and Bayes theorem without complication parameter estimation)
7. ***AdaBoost*** (example of ensemble learning like random forest where importance of hard data is boosted in importance)
8. ***C4.5*** (another ensemble learning approach with decision trees)
9. ***Apriori algorithm*** (algorithm for finding frequent itemsets in data)
10. ***PageRank*** (produces static ranking of variables based on graph theory)

No Neural Networks!

Intuition from a Single Layer with Linear Activation

Let's start by considering a network with no hidden layers and purely linear activation for binary classification

Single-Layer Perceptron



$$X\theta = Y$$

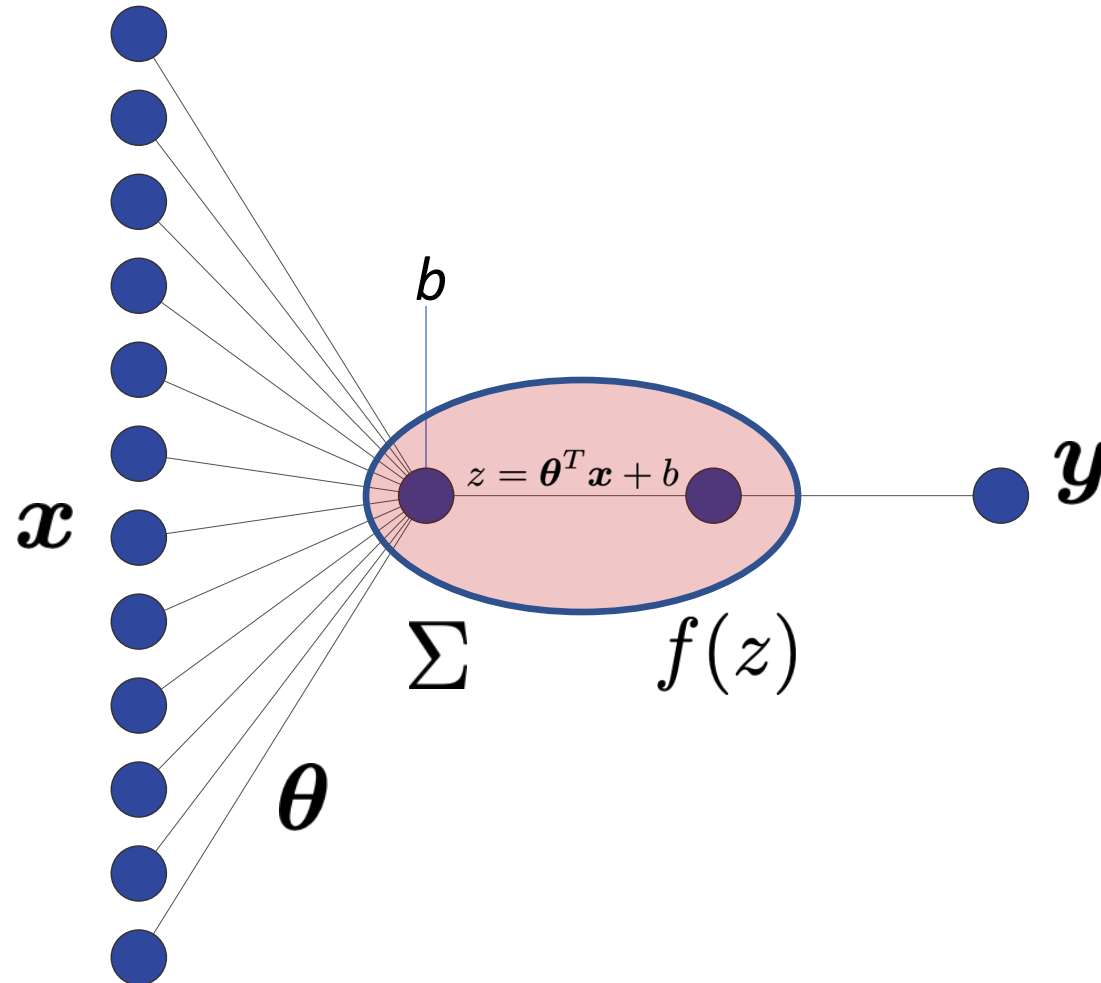
$$\begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}_{n \times m} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{pmatrix}_{m \times 1} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}_{n \times 1}$$

So, this is just linear regression. In the absence of any regularization, we can solve using the pseudoinverse, but otherwise, we can use gradient descent

Introducing other forms of activation

AANs can represent arbitrarily complex (non-linear) functions by using different activation functions

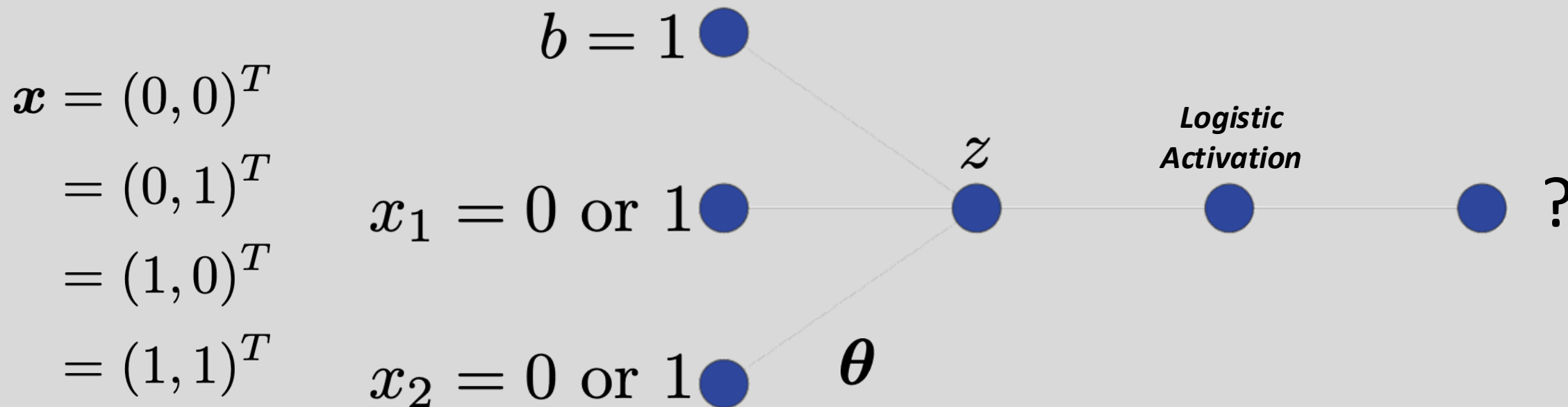
Single-Layer with explicit activation



Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$ ^[1]
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU) ^[9]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x \mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU) ^[4]		$\frac{1}{2} x \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$ $= x \Phi(x)$
Softplus ^[10]		$\ln(1 + e^x)$
Exponential linear unit (ELU) ^[11]		$\begin{cases} \alpha (e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α

Implementation of Logic

Let's try a couple examples to gain some intuition on how these architectures enable logic



$$\theta_a = \begin{pmatrix} -30 \\ 20 \\ 20 \end{pmatrix}$$

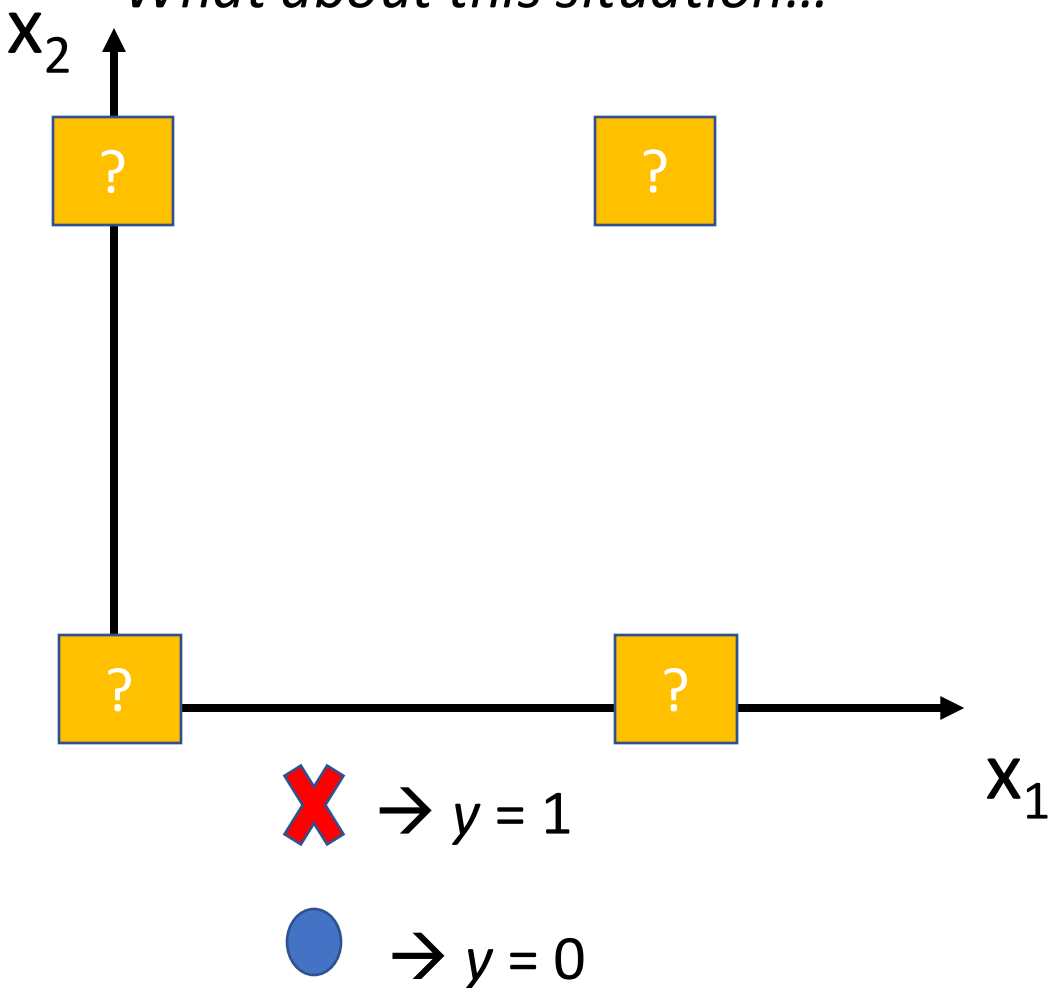
$$\theta_b = \begin{pmatrix} -10 \\ 20 \\ 20 \end{pmatrix}$$

$$\theta_c = \begin{pmatrix} 10 \\ -20 \\ -20 \end{pmatrix}$$

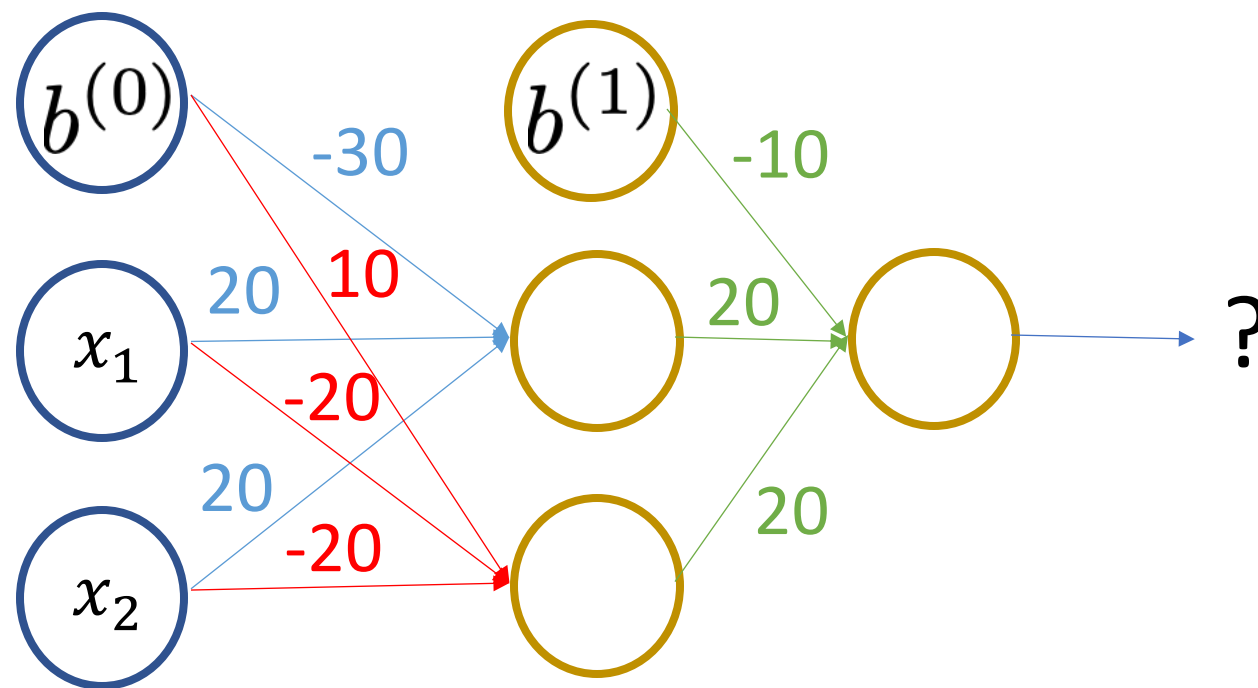
Implementation of Logic

Let's try a couple examples to gain some intuition on how these architectures enable logic

What about this situation...

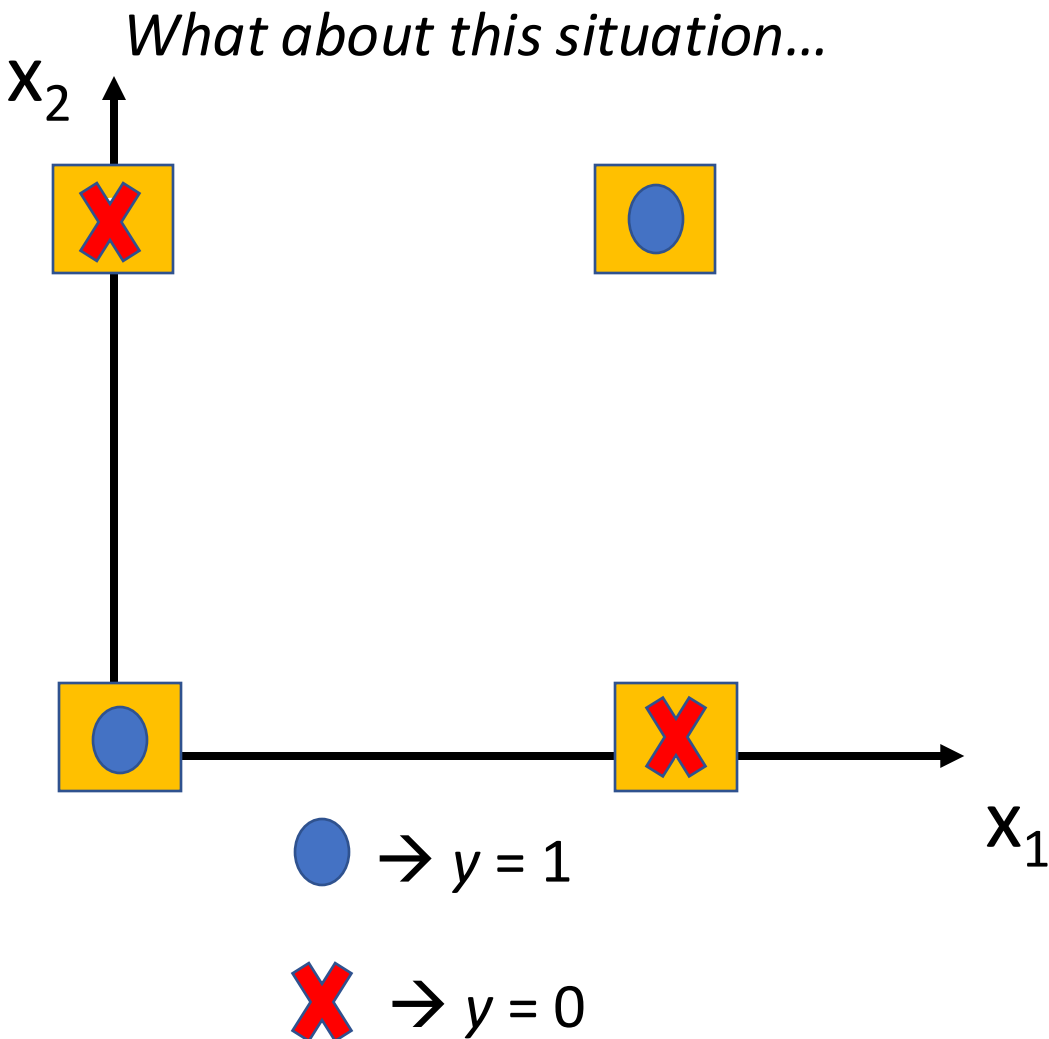


(using logistic activation in every neuron)



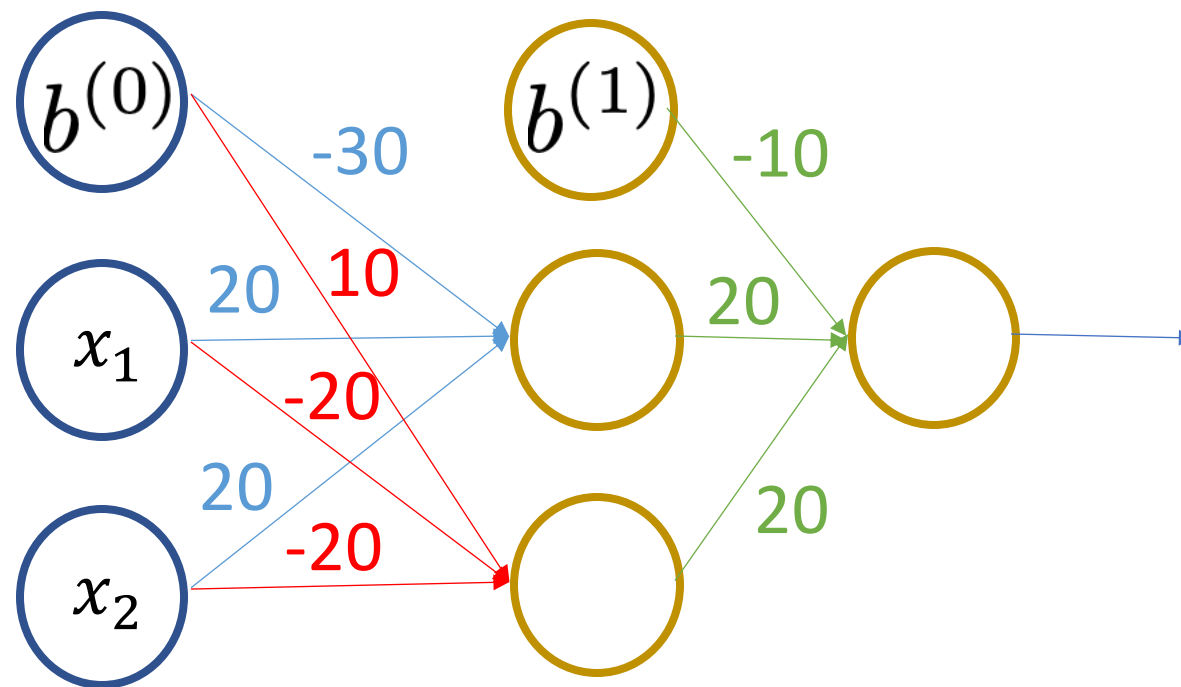
Implementation of Logic

Let's try a couple examples to gain some intuition on how these architectures enable logic



Logic Biconditional Exclusive Nor (XNOR)

(using logistic activation in every neuron)



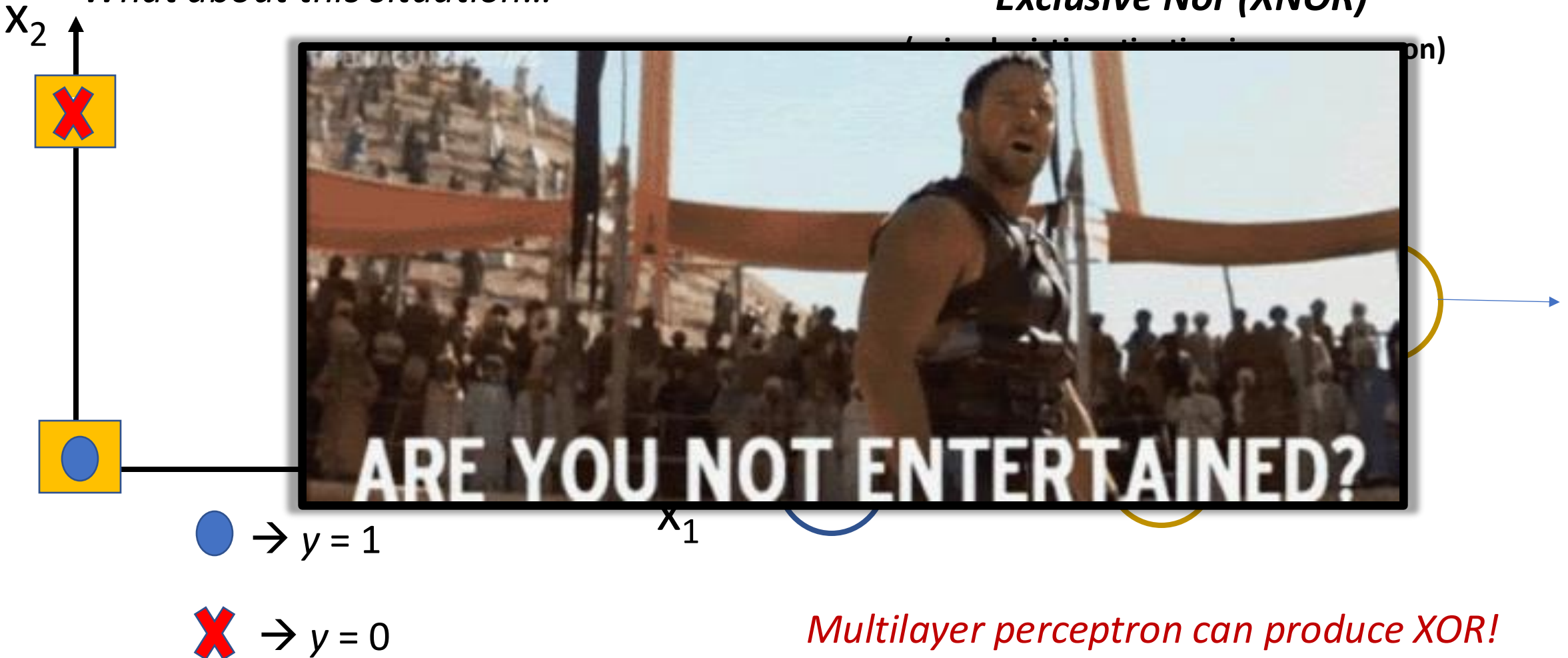
Multilayer perceptron can produce XOR!

Implementation of Logic

Let's try a couple examples to gain some intuition on how these architectures enable logic

*Logic Biconditional
Exclusive Nor (XNOR)*

What about this situation...



Universal Approximation Theorem

Artificial neural networks – algorithm inspired by neuronal brain activity in which input signals are processed and further transmitted amongst artificial neurons/nodes

Much of the power of neural networks stems from their ability to act as “**universal approximators**”

Universal Approximation Theorem (George Cybenko, Kurt Hornik):

From wikipedia:

Universal approximation theorem — Let $C(X, \mathbb{R}^m)$ denote the set of **continuous functions** from a subset X of a Euclidean \mathbb{R}^n space to a Euclidean space \mathbb{R}^m . Let $\sigma \in C(\mathbb{R}, \mathbb{R})$. Note that $(\sigma \circ x)_i = \sigma(x_i)$, so $\sigma \circ x$ denotes σ applied to each component of x .

Then σ is not **polynomial if and only if** for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, **compact** $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m)$, $\varepsilon > 0$ there exist $k \in \mathbb{N}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$, $C \in \mathbb{R}^{m \times k}$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

where $g(x) = C \cdot (\sigma \circ (A \cdot x + b))$

and other similarly gobbledy-gook type statements

Any continuous function $f: [0,1]^n \rightarrow [0,1]$ can be approximated arbitrarily well by a neural network with at least 1 hidden layer

Universal Approximation Theorem



Explain the universal approximation theorem like I am five.



Okay, imagine you have a magic box that can do all sorts of cool tricks. Now, you want this box to learn how to do any trick you can think of, like drawing pictures, playing music, or even telling jokes.

The universal approximation theorem is like a guarantee that says, "No matter how tricky the trick is, if you have a big enough magic box with lots of switches, you can teach it to do the trick really well." It's like saying you can use this magic box to make any shape you want with playdough by adjusting its switches.

So, the theorem tells us that with a big enough magic box (or a really powerful computer), we can teach it to do almost any task we can imagine, as long as we set its switches the right way. It's like having a super-smart friend who can learn to do anything with enough practice!

Universal Approximation Theorem

M

Explain the universal approximation theorem like I am a professor at Princeton university.



Certainly, as a professor at Princeton University, you'll appreciate a more detailed explanation:

The Universal Approximation Theorem is a fundamental concept in the field of machine learning and neural networks. It essentially states that a neural network with just one hidden layer, given a sufficient number of neurons (or units) in that layer, can approximate any continuous function to arbitrary accuracy within a bounded domain.

To break it down further:

1. **Neural Network:** Think of a neural network as a mathematical model inspired by the human brain. It's made up of layers of interconnected nodes (neurons) that process information.
2. **Hidden Layer:** This is a layer in the neural network that sits between the input and output layers. The theorem focuses on networks with a single hidden layer.
3. **Approximate Any Function:** The theorem tells us that this single hidden layer can be configured, or "tuned," in such a way that the neural network can closely mimic (or approximate) any continuous function. This means it can learn to map inputs to outputs in a highly flexible manner.
4. **Arbitrary Accuracy:** The neural network can approximate the function as accurately as you want, given enough neurons in the hidden layer. In other words, you can make the approximation as precise as needed for your specific problem.
5. **Bounded Domain:** The theorem assumes that this approximation works within a specific range or domain of inputs. It doesn't promise universal approximation outside of that domain.

In essence, the Universal Approximation Theorem is a powerful concept that highlights the expressive capabilities of neural networks. It's like saying that with the right architecture and a sufficient number of neurons in a single hidden layer, you can train a neural network to model and approximate nearly any mathematical function, making it a versatile tool in solving complex problems in fields like data analysis, pattern recognition, and artificial intelligence.

Universal Approximation Theorem

M

Simplify your response to one sentence.



The Universal Approximation Theorem states that a neural network, with a big enough hidden layer, can closely mimic any kind of function, making it a powerful tool in solving many different types of problems.

M

Simplify your response to three words.



Neural networks approximate.

M

One word.



Approximate.

M

No words.



Understood.

M

That was a word.

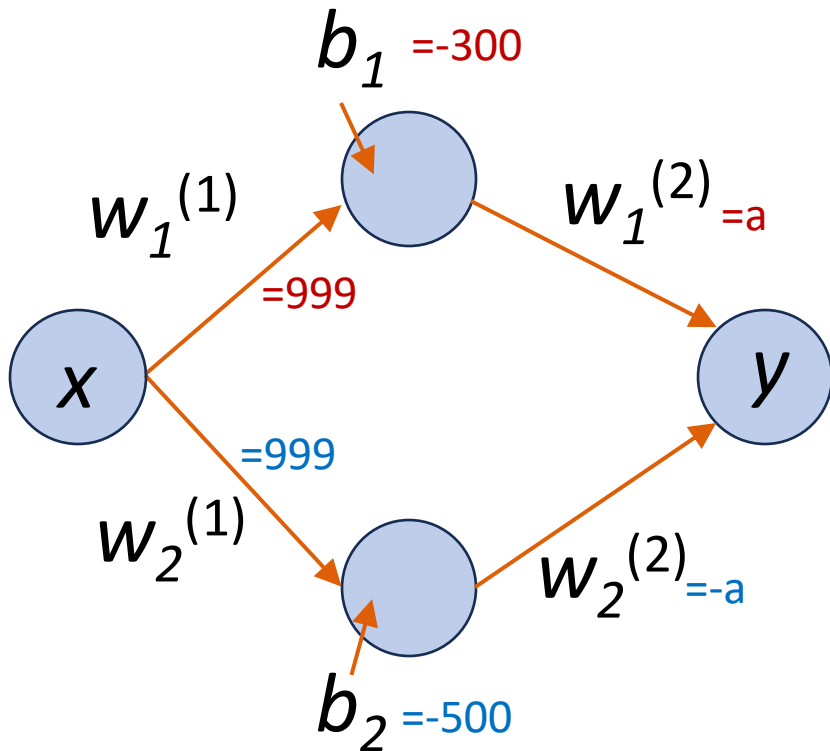


Apologies for the oversight.

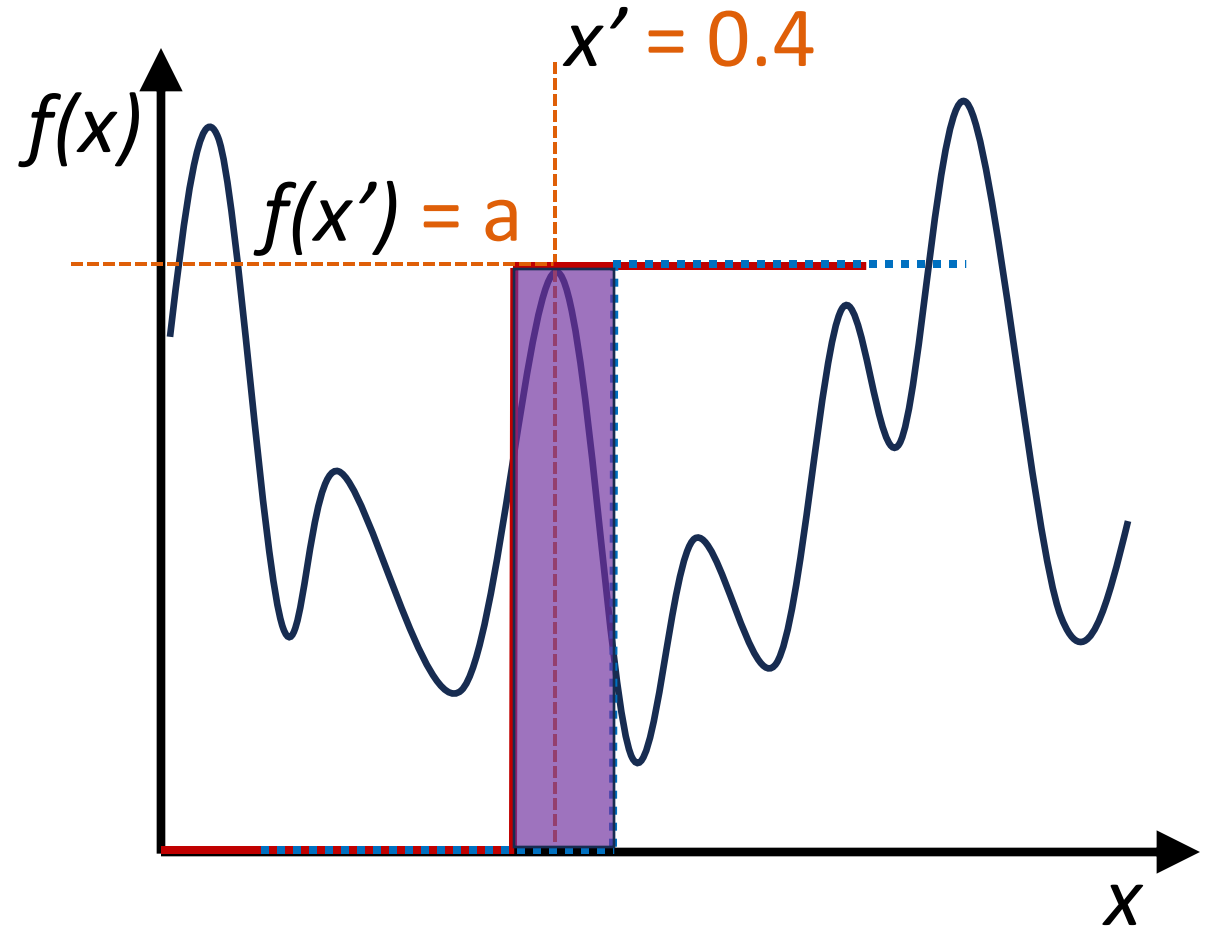


Universal Approximation Theorem

We can gain some intuition for the universal approximation theorem by considering a simple neural network with 2 hidden neurons



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



we can create a local function approximation via an effective delta function (bin with tunable width and height)