Fundamentals / Foundation

Data Science 9CFU

Computer Science 6CFU

# of Data Science

Indro Spinelli

Sapienza University of Rome

From Manual Features to Learned Representations

In traditional machine learning (e.g., SVMs, Logistic Regression), the model's performance depends entirely on the quality of the input features.

Human's Job: A domain expert must manually engineer features from the raw data.

Example (Spam Detection):

- **Raw Data:** "Buy Viagra now for cheap!!!"
- **Manual Features:**
  - contains_word_"viagra" = 1
  - number_of_exclamation_points = 3
  - uses_all_caps = 1
  - contains_word_"cheap" = 1

Problem: This is time-consuming, requires expert knowledge, and is brittle. What if we miss the most important features?

Representation Learning

This is the core paradigm shift.

- **Definition:** Representation Learning is a set of techniques that allows a model to **automatically learn the most useful features (or "representations")** directly from the raw data.
- **Goal:** Instead of a human hand-crafting features, the model *discovers* the optimal way to represent the data to best accomplish the task (e.g., classification or regression).

Deep Learning

Deep Learning is the most powerful and popular method for representation learning.

- **Definition:** Deep learning uses a **hierarchy of layers** to learn multiple levels of representation at increasing levels of complexity.
- **The Hierarchy:** The model doesn't learn everything at once. It builds complex concepts from simpler ones, layer by layer.

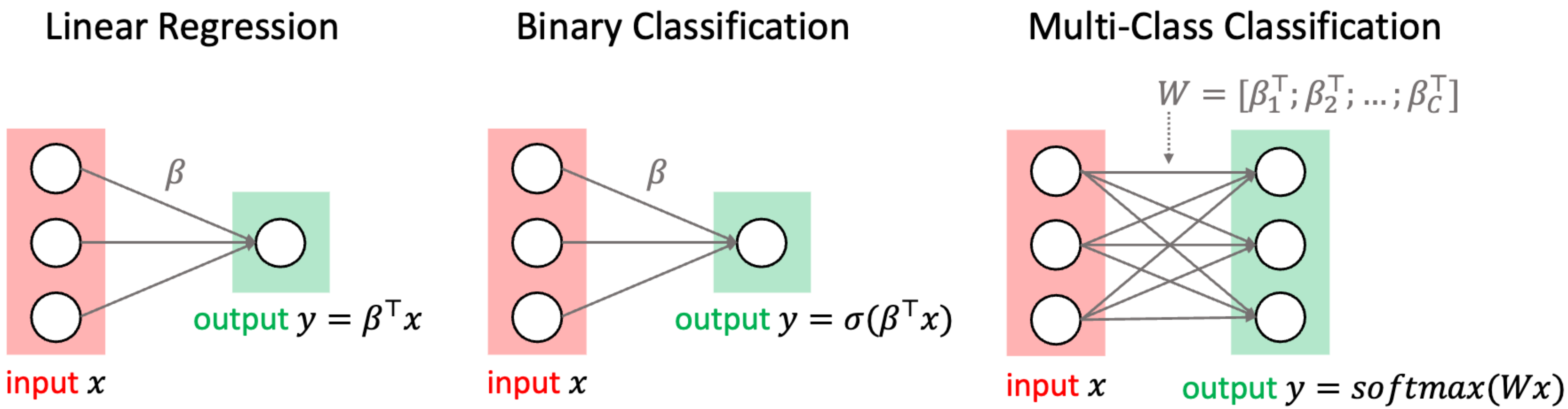
Image Recognition

Imagine a deep neural network learning to recognize a face:

- **Layer 1 (Low-Level):** Learns to "see" simple features like **edges**, **corners**, and **colors** from raw pixels.
- **Layer 2 (Mid-Level):** Learns to **combine edges** to form simple **shapes** and **textures** (e.g., an "oval," a "line," a "patch of skin").
- **Layer 3 (High-Level):** Learns to **combine shapes** to form **"object parts"** (e.g., an "eye," a "nose," a "mouth").
- **Final Layer (Abstract):** Learns to **combine the parts** into the final, abstract concept of a **"face."**

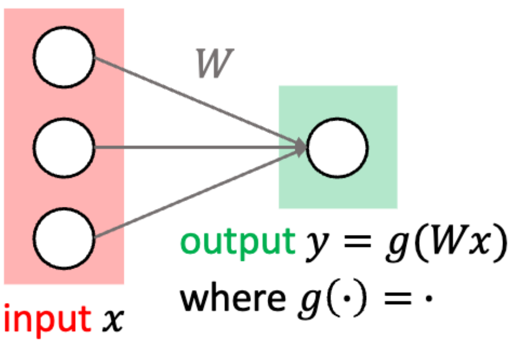
A Unifying View

Linear transformation of input features followed by an activation function

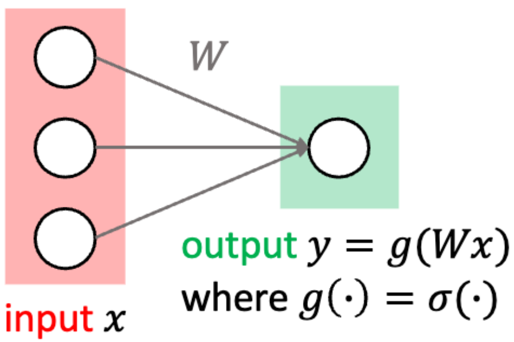


Linear transformation of input features (Wx) followed by an activation function $g(\cdot)$

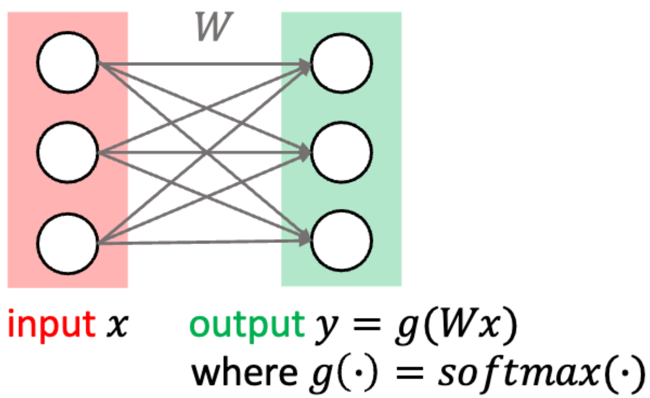
Linear Regression



Binary Classification



Multi-Class Classification



$W \in \mathbb{R}^{C \times D}$ where D is the input dimension and C is the output dimension.

History

1943: Perceptron model (McCulloch & Pitts)

- Intended as theoretical model of biological neurons

1969: Perceptrons cannot learn XOR (Minsky & Papert)

- Highly controversial (may have helped cause “AI winter”)

1997: Long Short-term Memory Networks (Hochreiter, Schmidhuber)

- This breakthrough enabled models to effectively learn from long sequences

1998: Convolutional neural networks for MNIST (Lecun)

- Human-level performance on handwritten digit recognition

Modern History

2012: ImageNet breakthrough (Krizhevsky, Sutskever, & Hinton)
Reduced error on image classification by 50%

2017: Transformer architecture (Vaswani et al.)

2018: Turing award (Bengio, Hinton, & Lecun)

2018:Improving Language Understanding by Generative Pre-Training (OpenAI)

2019: Language Models are Unsupervised Multitask Learners (OpenAI)

2020: Language Models are Few-Shot Learners (OpenAI)

Now and the Future

The Nobel Prize in Chemistry 2024 (Barker, Hassabis, Jumper)

The Nobel Prize in Physics 2024 (Hopfield, Hinton)

Generative AI & LLMs

Why Do We Need Non-Linear Activation Functions?

Deep learning is powerful because it builds **complex, non-linear models**. 👉 The **activation function** is what makes this possible.

Let's build a multi-layer network **using only linear operations**.

3-Layer Linear Network

$$y = W_3\big(W_2(W_1 x)\big) = (W_3 W_2 W_1)x = W_{stacked}x$$

"deep" stack of **linear layers** is **mathematically equivalent** to a single linear layer.

Loss Functions & Capacity

Same principles as in single-layer models

Regression:

- *Mean Squared Error (MSE)*

Classification:

- *Binary Classification:* **Binary Cross-Entropy (BCE)**
- *Multi-Class Classification:* **Cross-Entropy (CE)**

Model Capacity

Capacity of a feed-forward neural network is affected by both:

- Depth: number of hidden layers
- Width: number of neurons in each hidden layer

More neurons = more capacity

Optimization

Backpropagation

The core algorithm for training neural networks.

It's a direct application of the **chain rule** from calculus.

- Computes the error at the final layer.
- Propagates this error signal **backward**, one layer at a time.

- **Modern Practice:** You don't do this by hand!
  - Modern frameworks (PyTorch, Jax) handle it automatically using Automatic Differentiation (Autodiff).

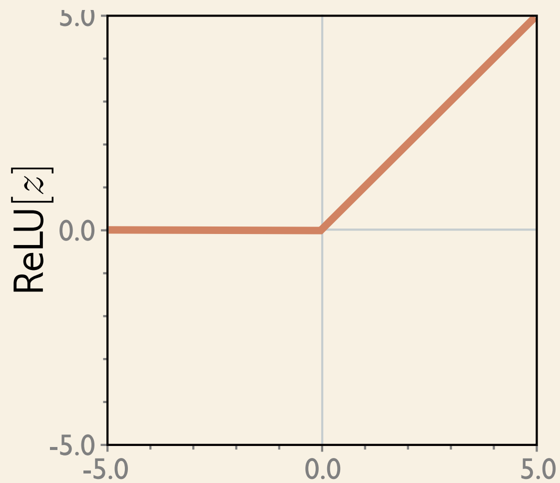
Example shallow network

Activation function

$$y = \mathrm{f}[x, \boldsymbol{\phi}]$$
$$= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}.$$

Rectified Linear Unit



Example shallow network

$$
\begin{aligned}
y &= \mathrm{f}[x, \boldsymbol{\phi}] \\
&= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x]
\end{aligned}
$$

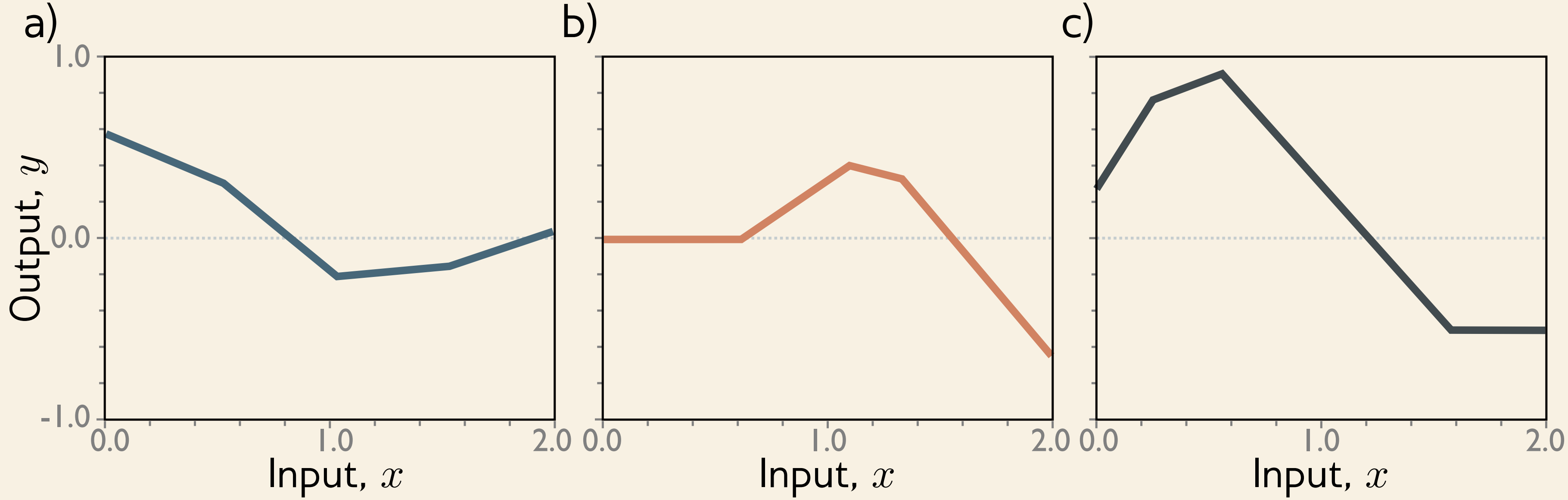
This model has 10 parameters:

$$\boldsymbol{\phi} = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$$

- Represents a family of functions
- Parameters determine particular function
- Given parameters can perform inference (run equation)
- Given training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$
- Define loss function $L[\boldsymbol{\phi}]$ (least squares)
- Change parameters to minimize loss function

Example shallow network

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



Piecewise linear functions with three joints

Hidden units

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$

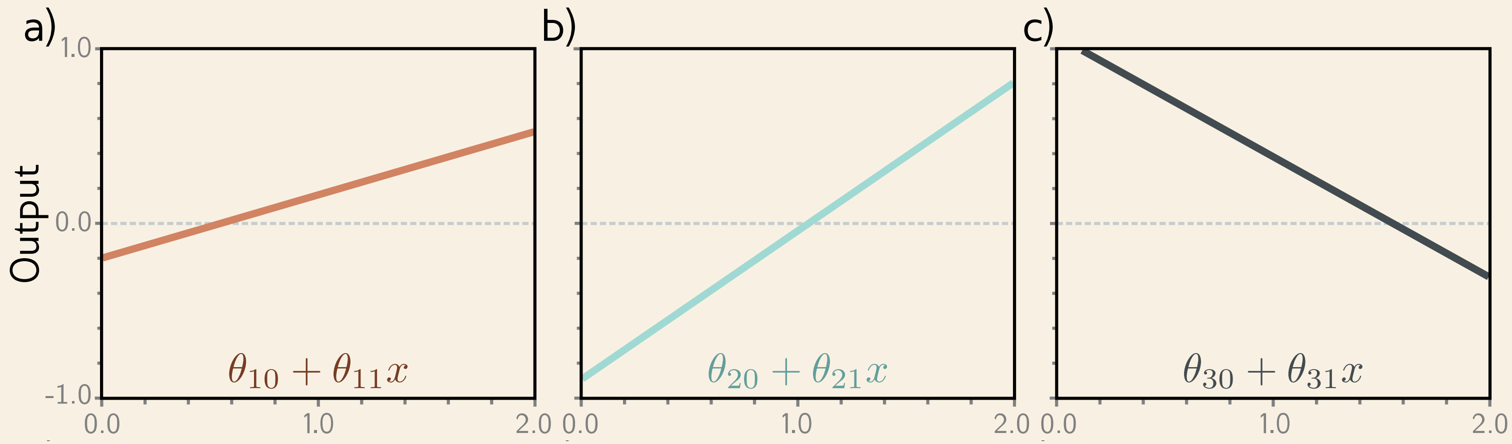
Break down into two parts:

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

where:

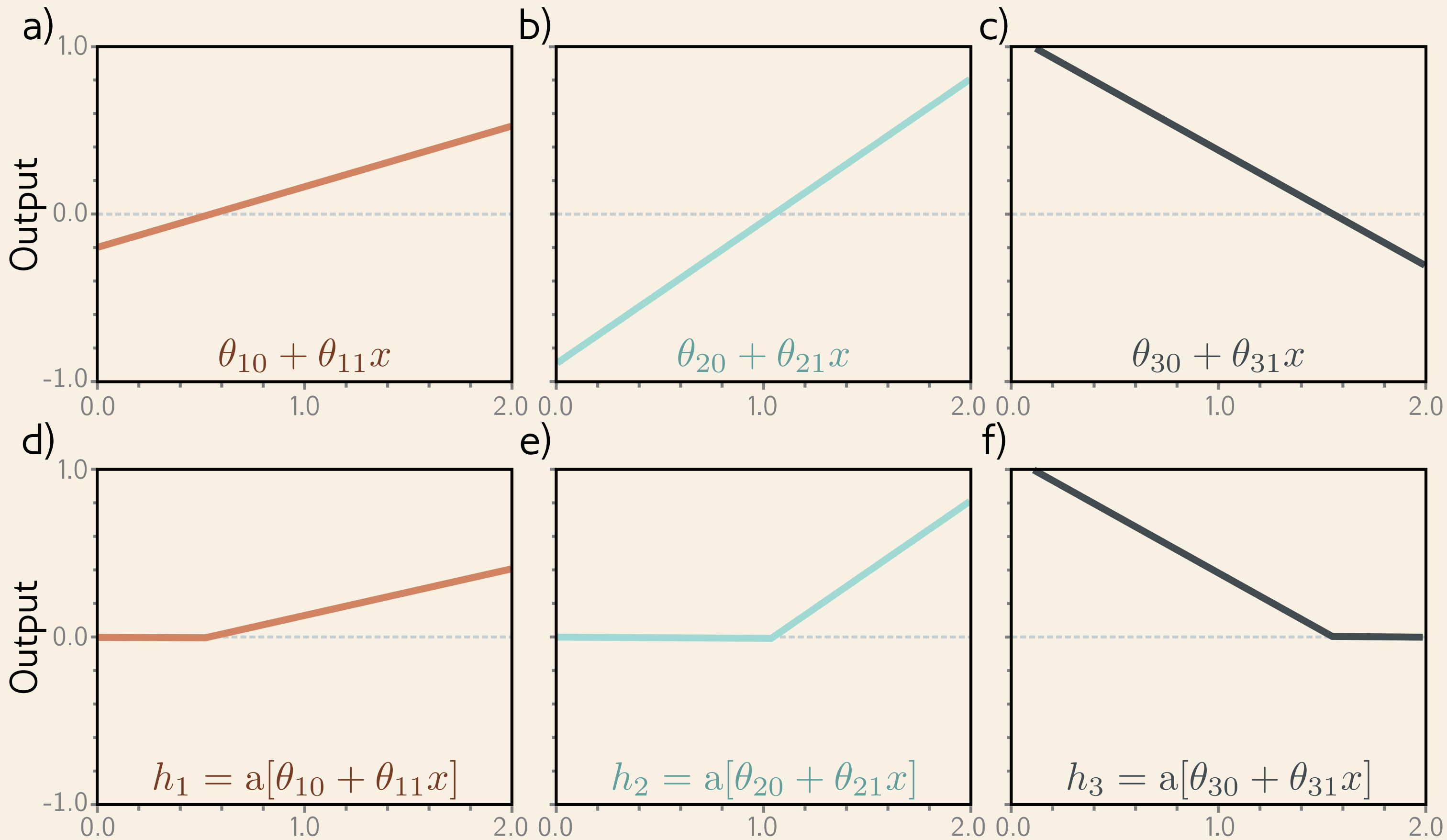
$$\text{Hidden units} \left\{ \begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \end{aligned} \right.$$

1. compute three
linear functions

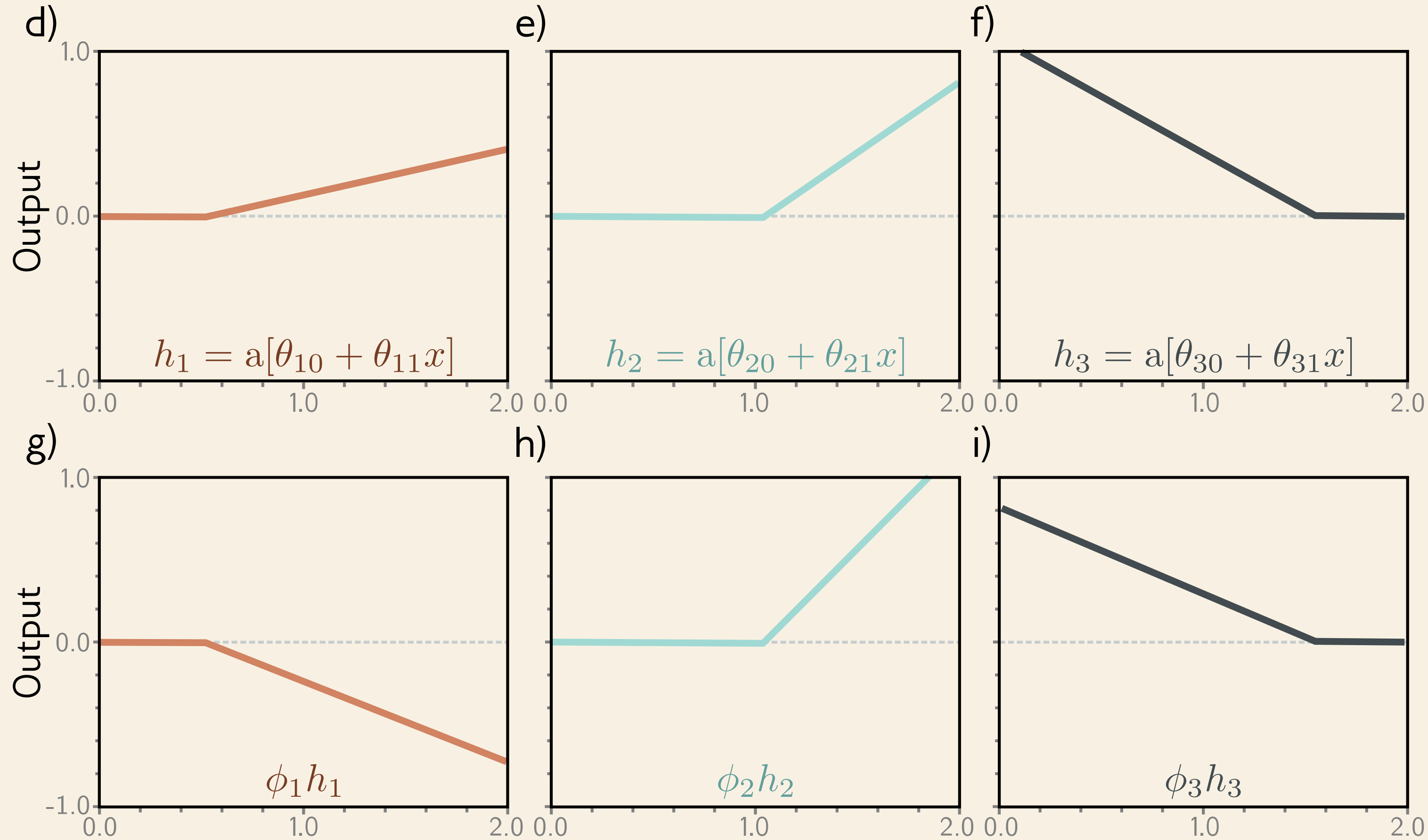


2. Pass through ReLU functions (creates hidden units)

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x],$$

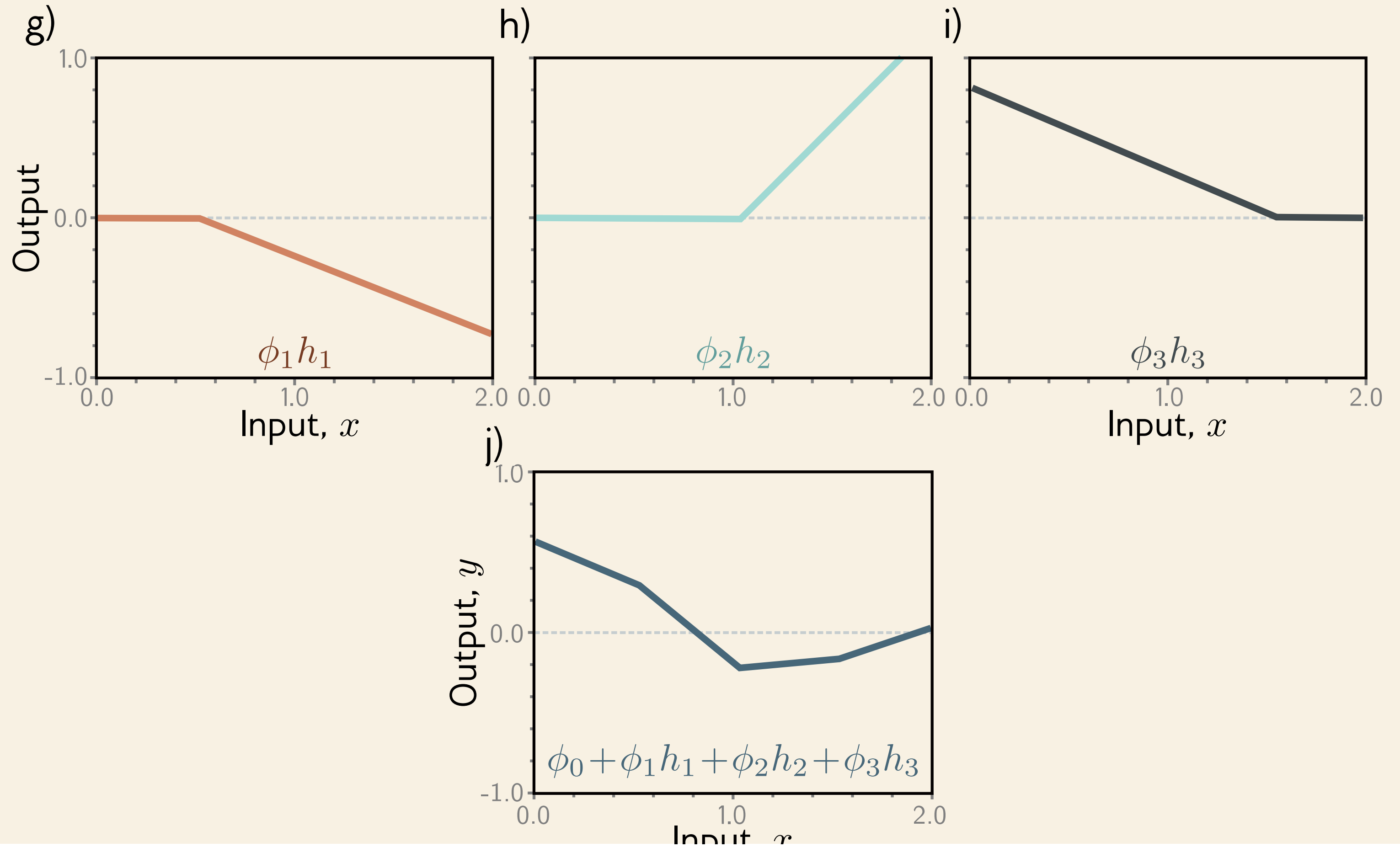


3. Weight the hidden units



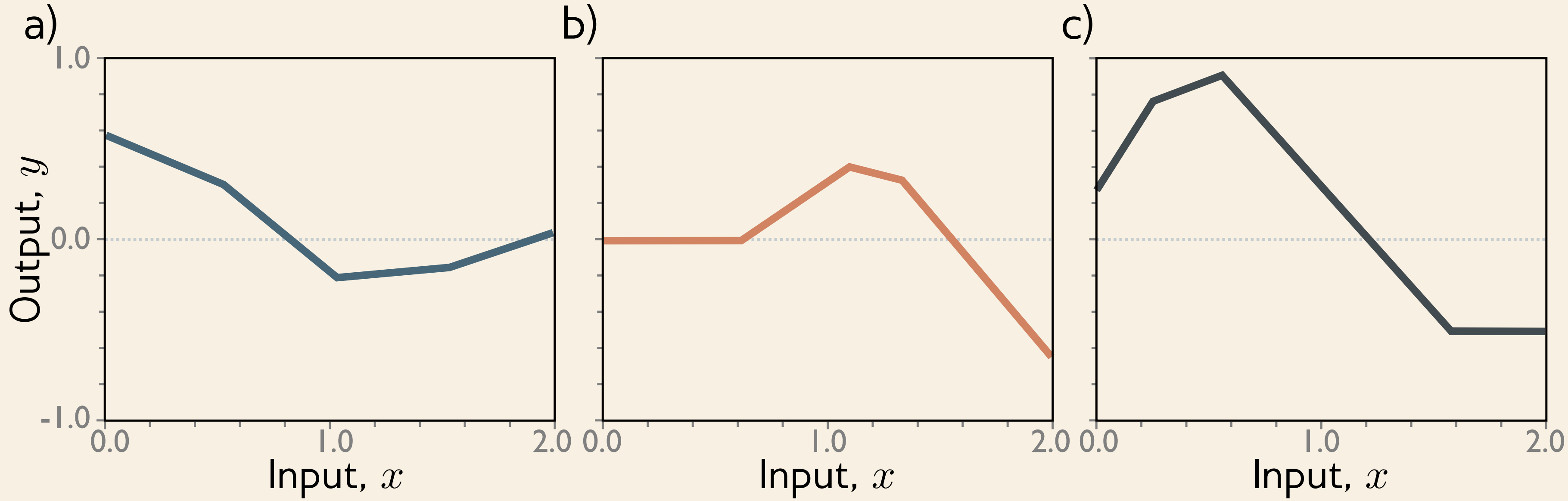
4. Sum the weighted hidden units to create output

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



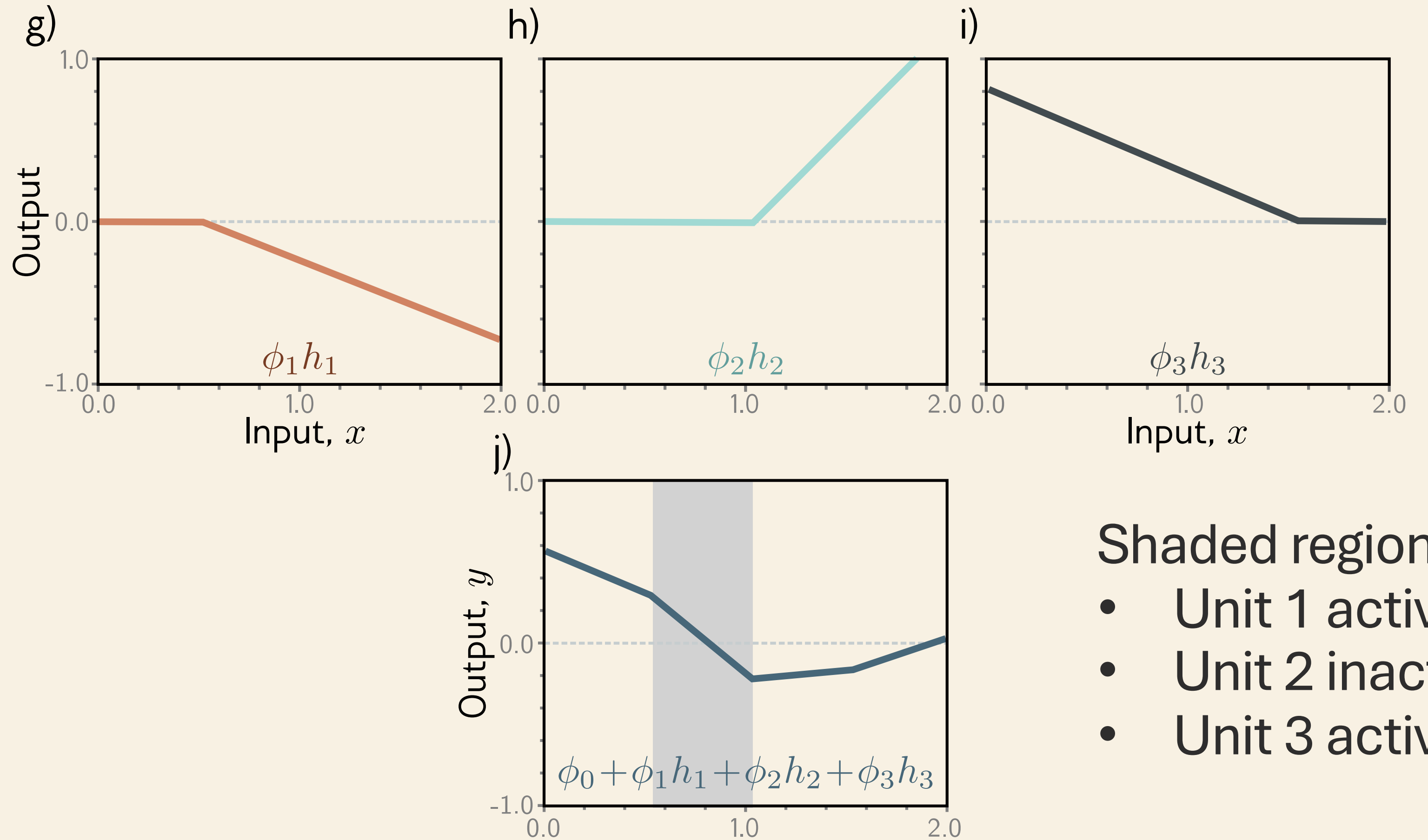
Example shallow network

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



Example shallow network = piecewise linear functions
1 “joint” per ReLU function

Activation pattern = which hidden units are activated



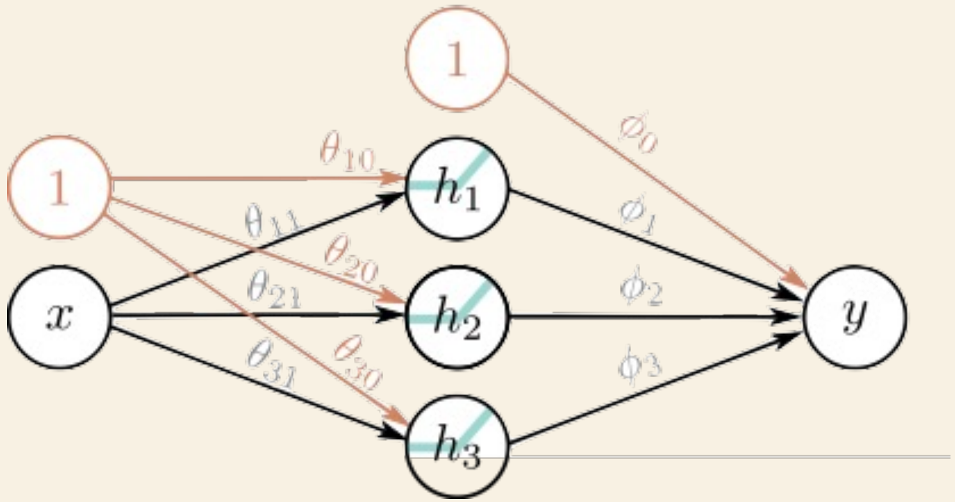
Shaded region:

- Unit 1 active
- Unit 2 inactive
- Unit 3 active

Depicting neural networks

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



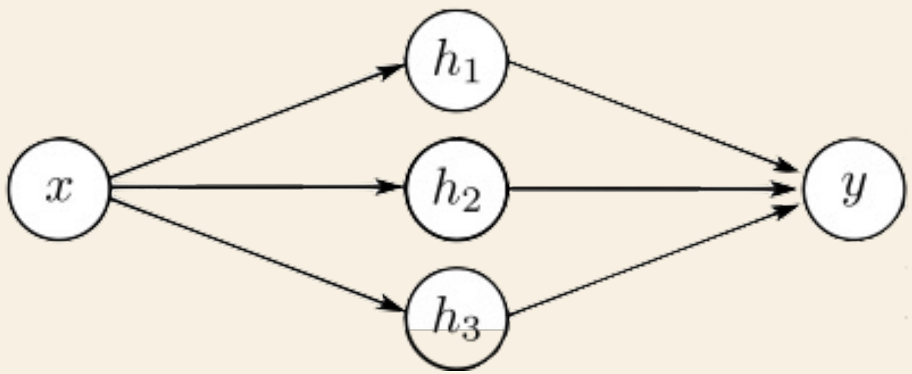
Each parameter multiplies its source and adds to its target

Depicting neural networks

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

b)



With 3 hidden units:

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x]
\end{aligned}
$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

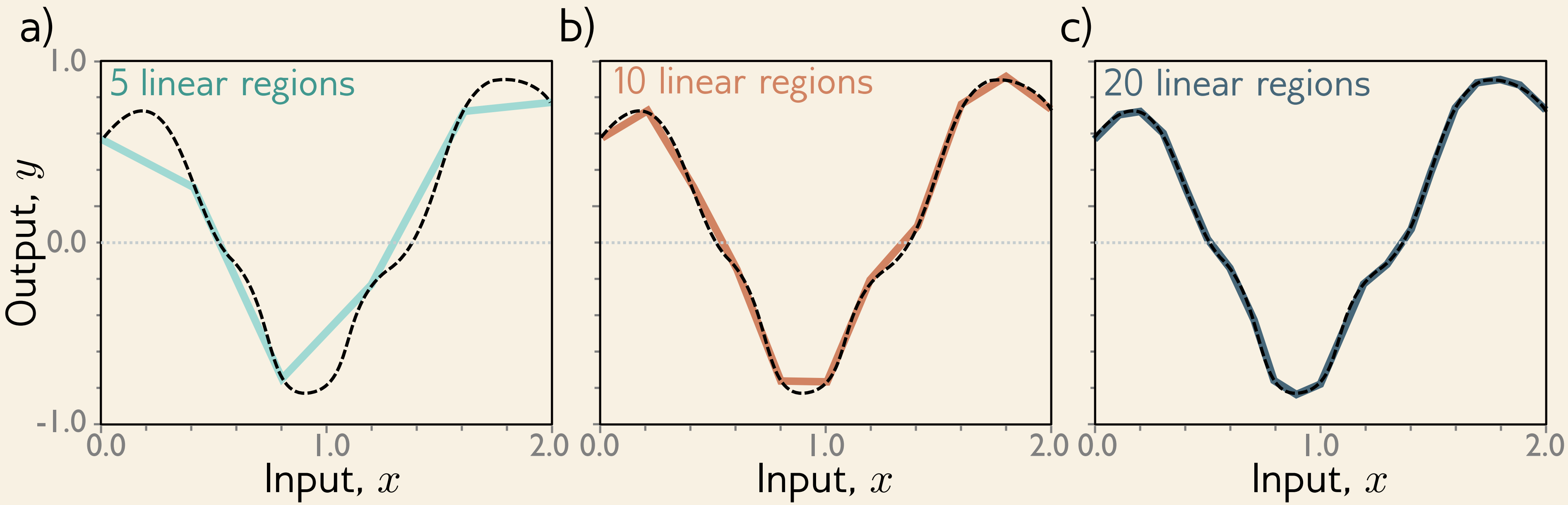
With D hidden units:

$$h_d = \mathrm{a}[\theta_{d0} + \theta_{d1}x]$$

$$y = \phi_0 + \sum_{d=1}^{D} \phi_d h_d$$

With enough hidden units…

… we can describe any 1D function to arbitrary accuracy



Universal approximation theorem

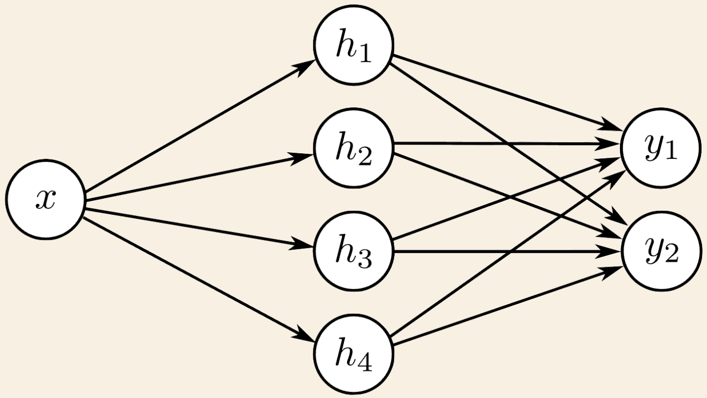
“a formal proof that, with enough hidden units, a shallow neural network can describe any continuous function on a compact subset of $\mathbb{R}^D$ to arbitrary precision”

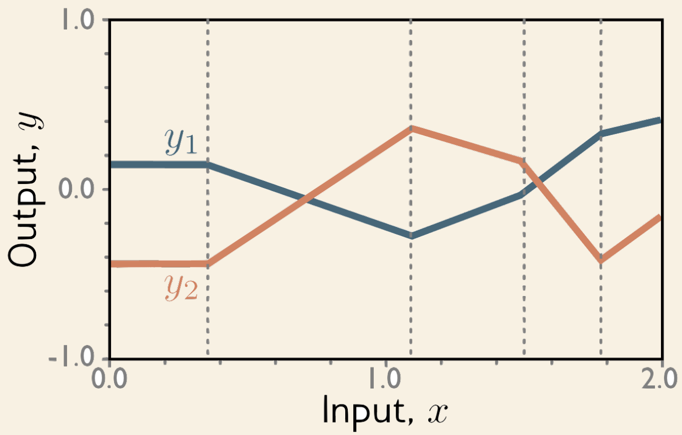
Two outputs

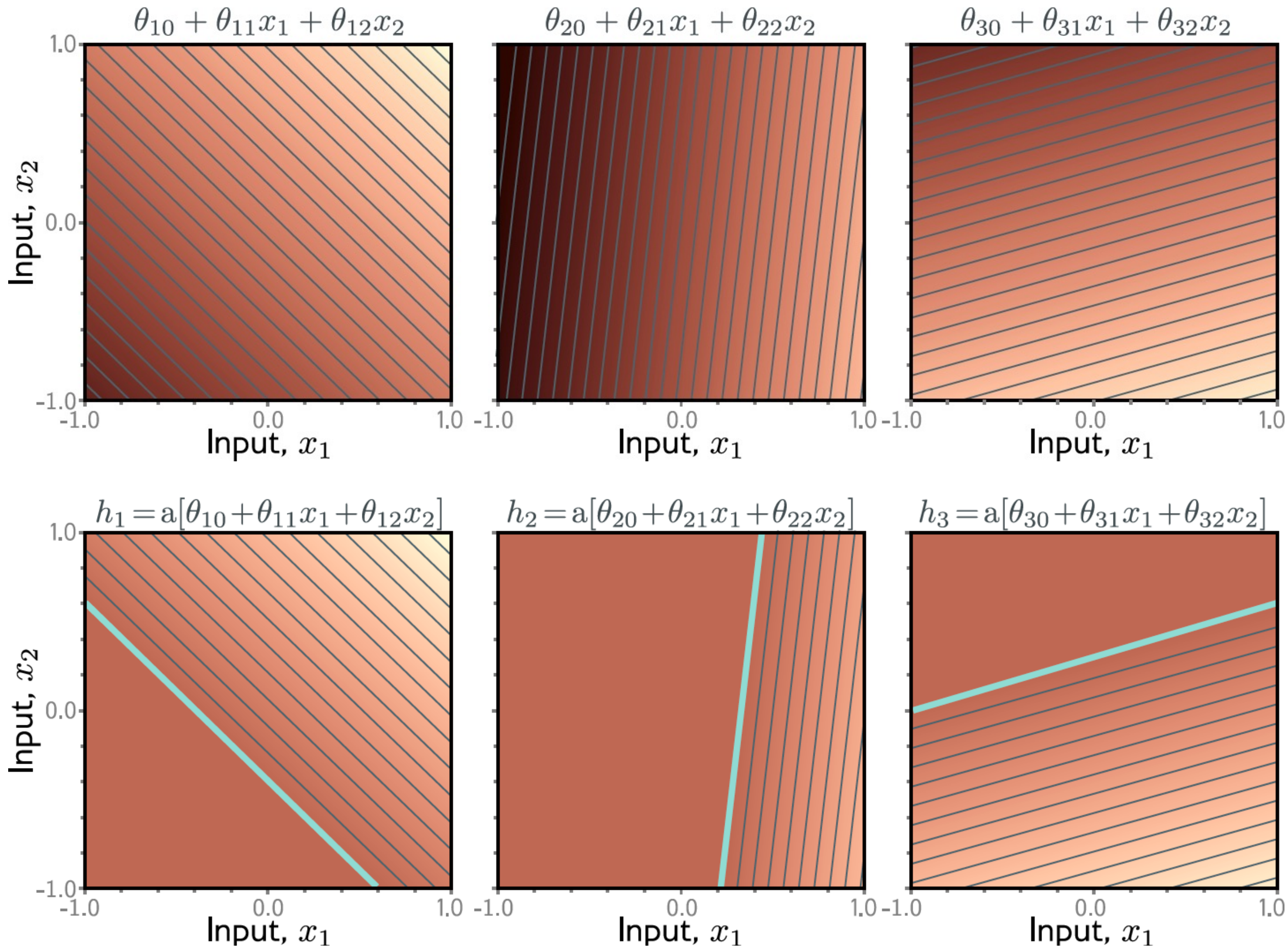
- 1 input, 4 hidden units, 2 outputs

$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \\
h_4 &= \mathrm{a}[\theta_{40} + \theta_{41}x]
\end{aligned}
$$

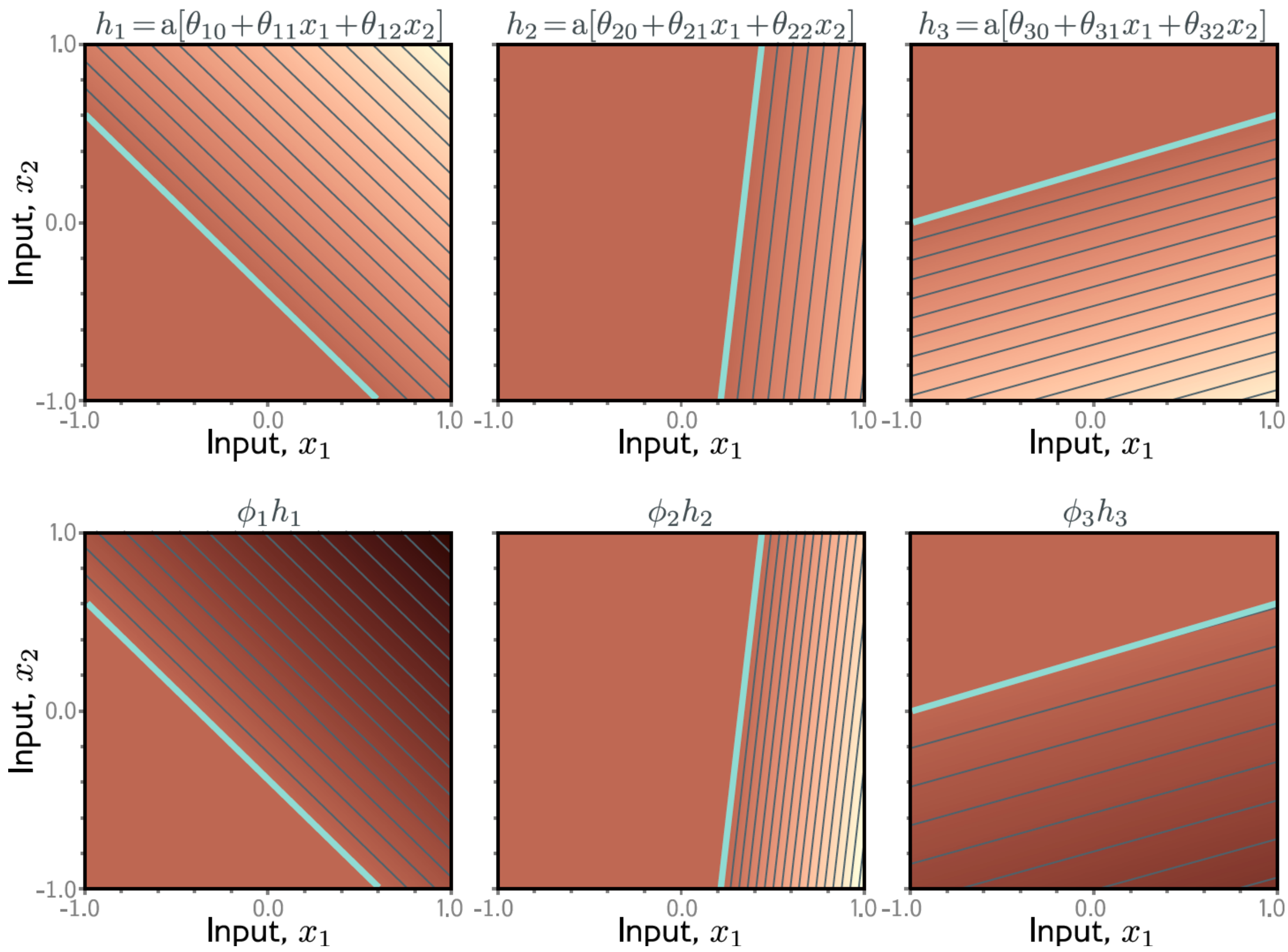
$$
\begin{aligned}
y_1 &= \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4 \\
y_2 &= \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4
\end{aligned}
$$



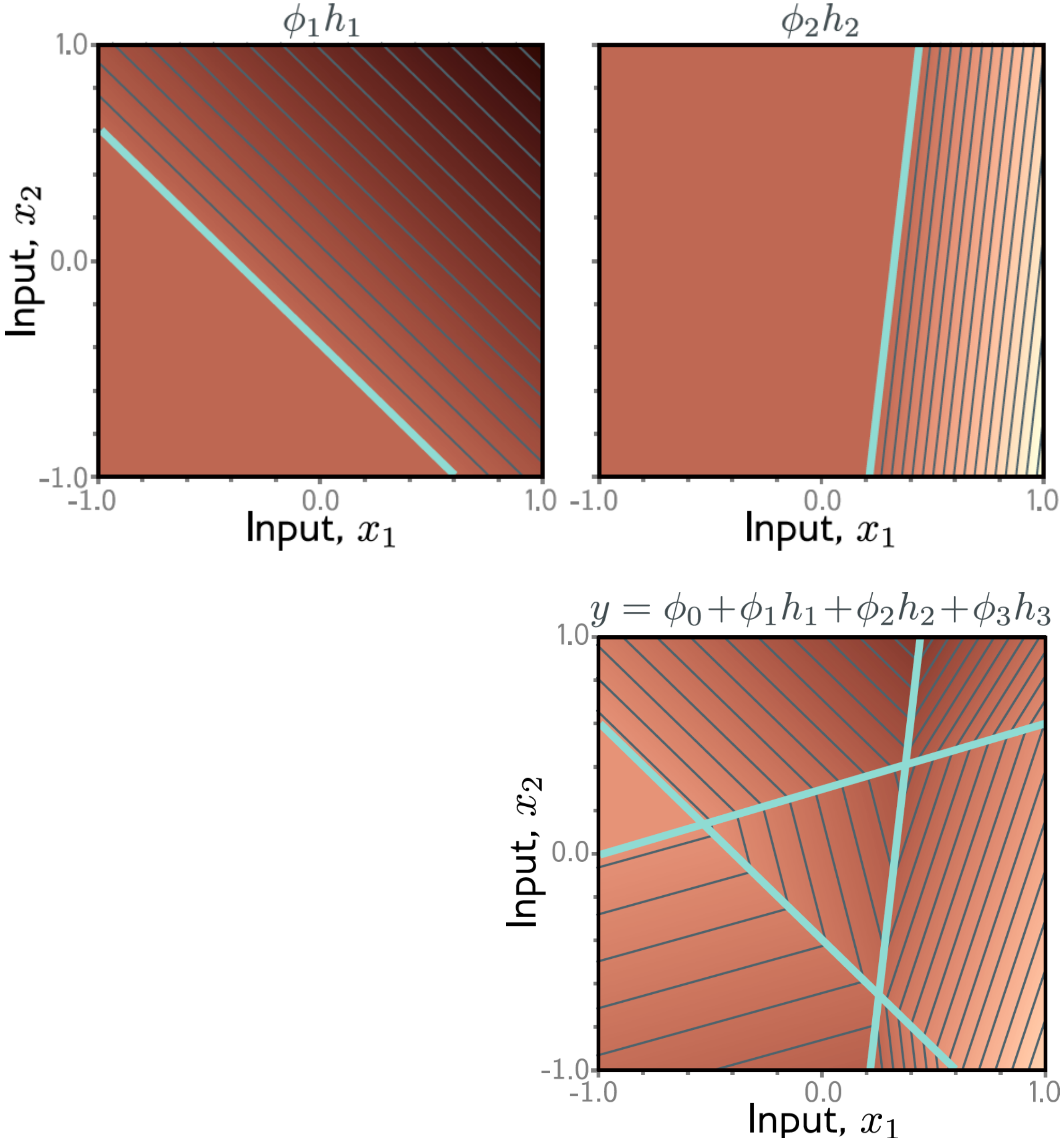




$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
$h_1 = \mathrm{a}[\theta_{10}+\theta_{11}x_1+\theta_{12}x_2]$
$h_2 = \mathrm{a}[\theta_{20}+\theta_{21}x_1+\theta_{22}x_2]$
$h_3 = \mathrm{a}[\theta_{30}+\theta_{31}x_1+\theta_{32}x_2]$
Input, x_1
Input, x_2
1.0
0.0
-1.0

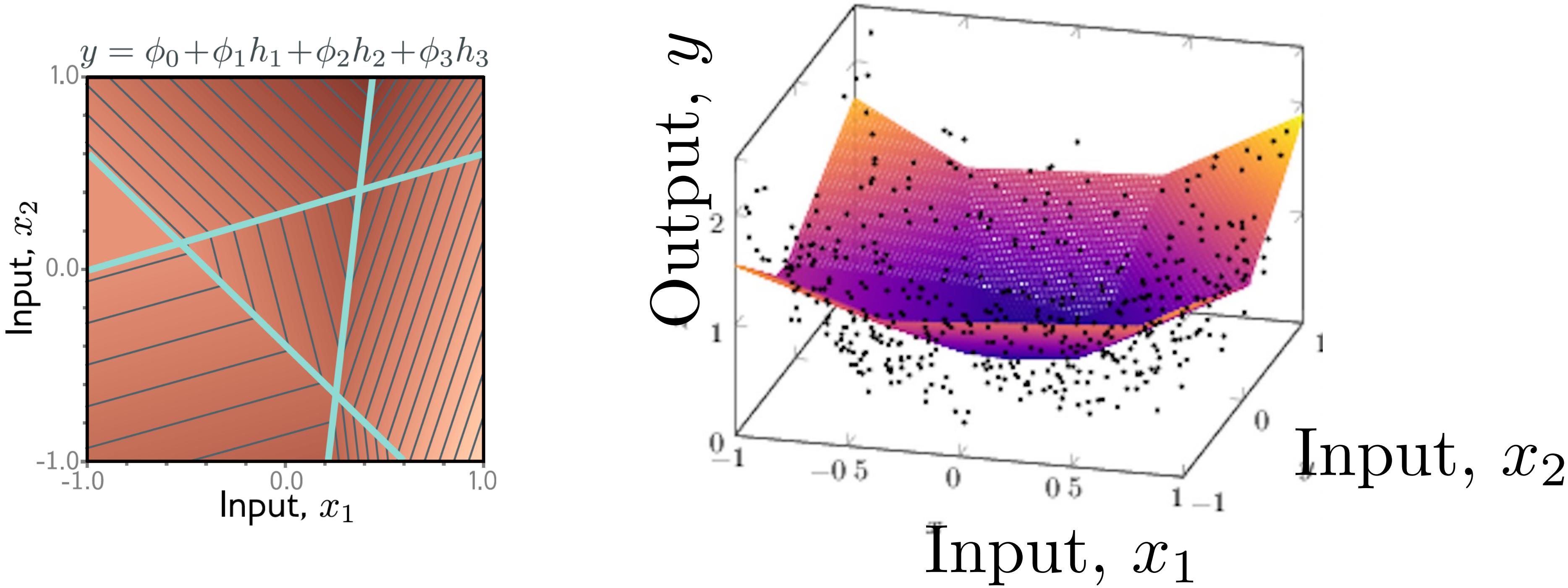


$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
$\phi_1 h_1$
$\phi_2 h_2$
$\phi_3 h_3$
Input, x_1
Input, x_2
-1.0
0.0
1.0



Convex polygons

Fitting



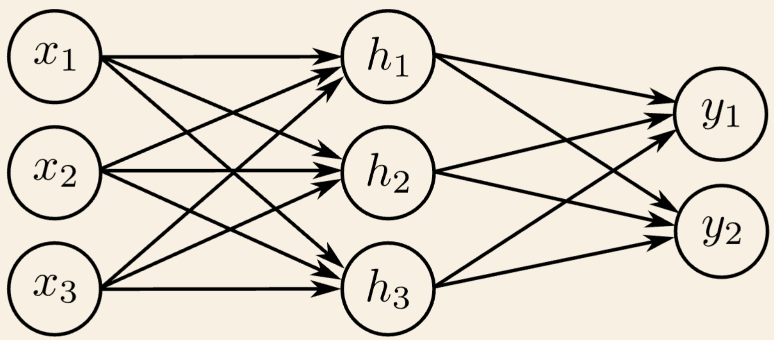
$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$
1.0
0.0
-1.0
Input, x_2
-1.0
0.0
1.0
Input, x_1
Output, y
0
1
2
−1
−0.5
0
0.5
1
−1
0
1
Input, x_2
Input, x_1

Arbitrary inputs, hidden units, outputs

- D_o Outputs, *D* hidden units, and D_i inputs

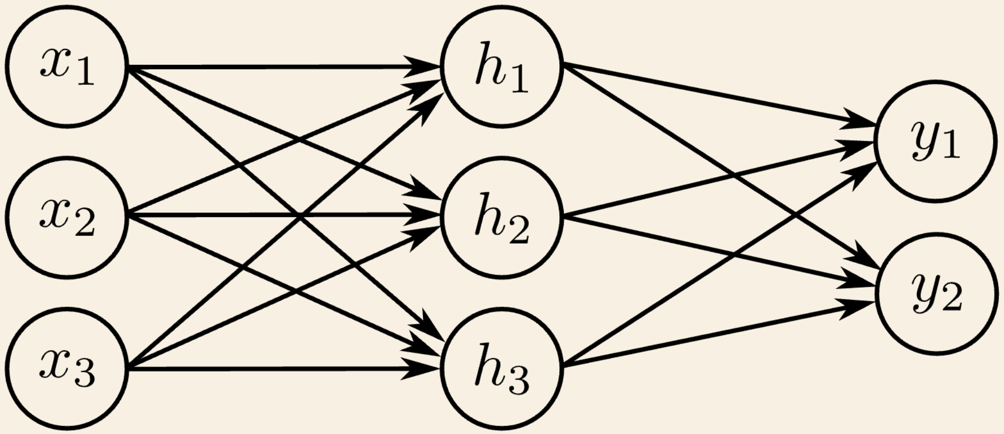
$$h_d = \mathrm{a}\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i\right] \qquad y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d$$

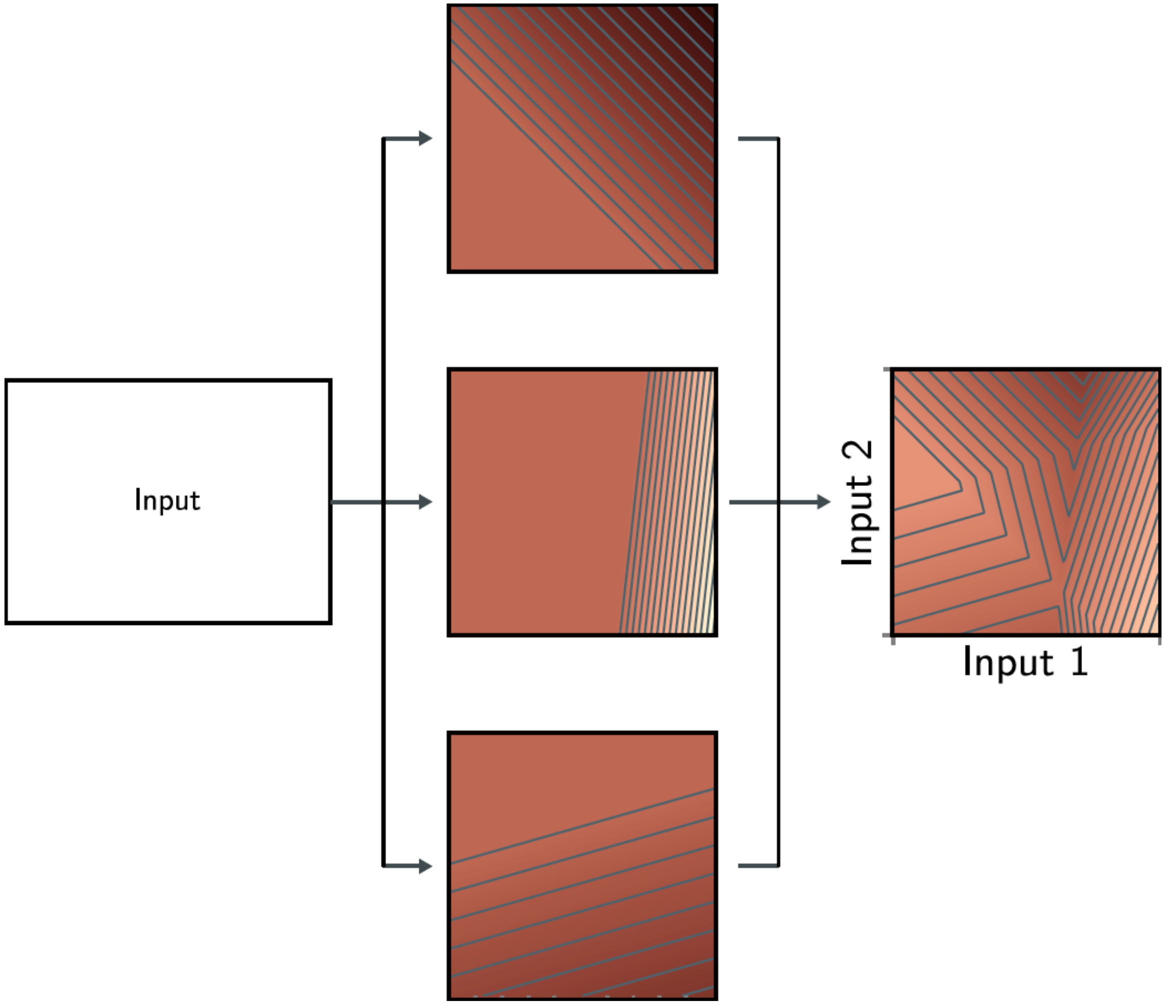
- e.g., Three inputs, three hidden units, two outputs

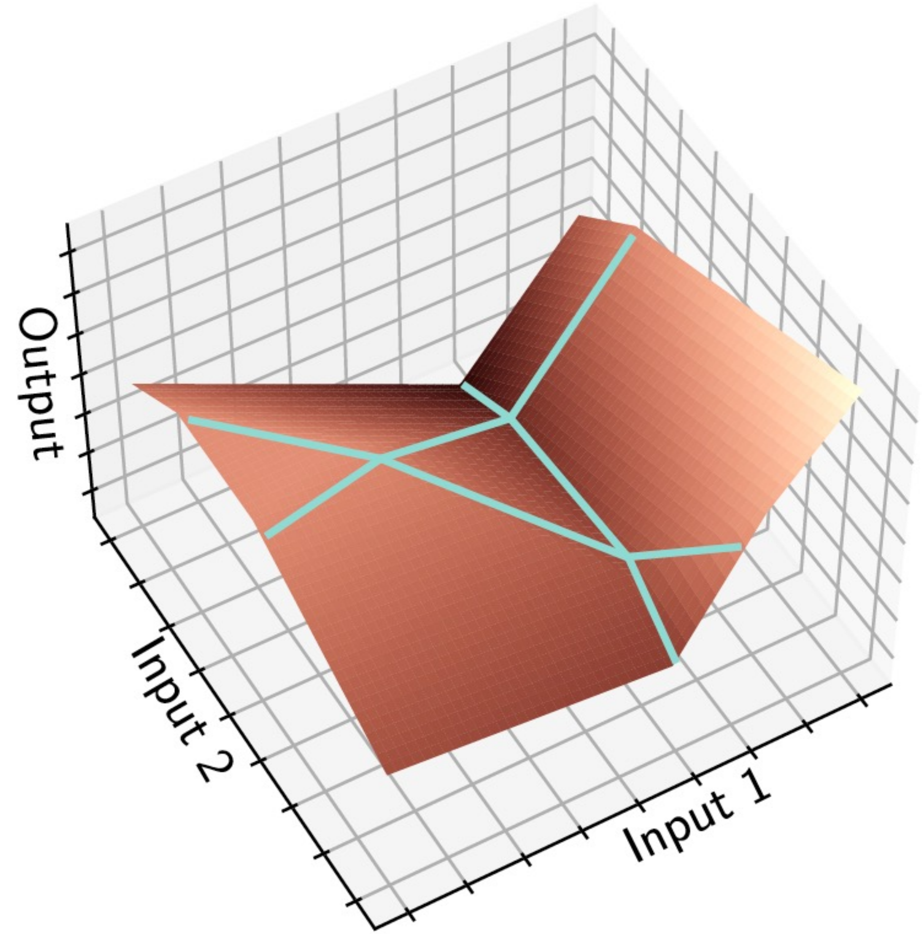


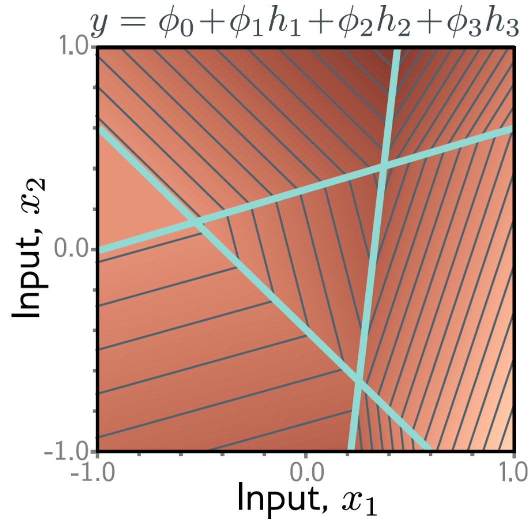
Question:

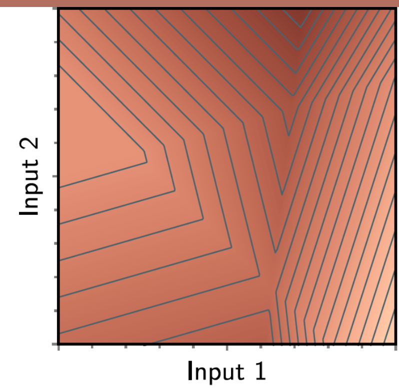
- How many parameters does this model have?

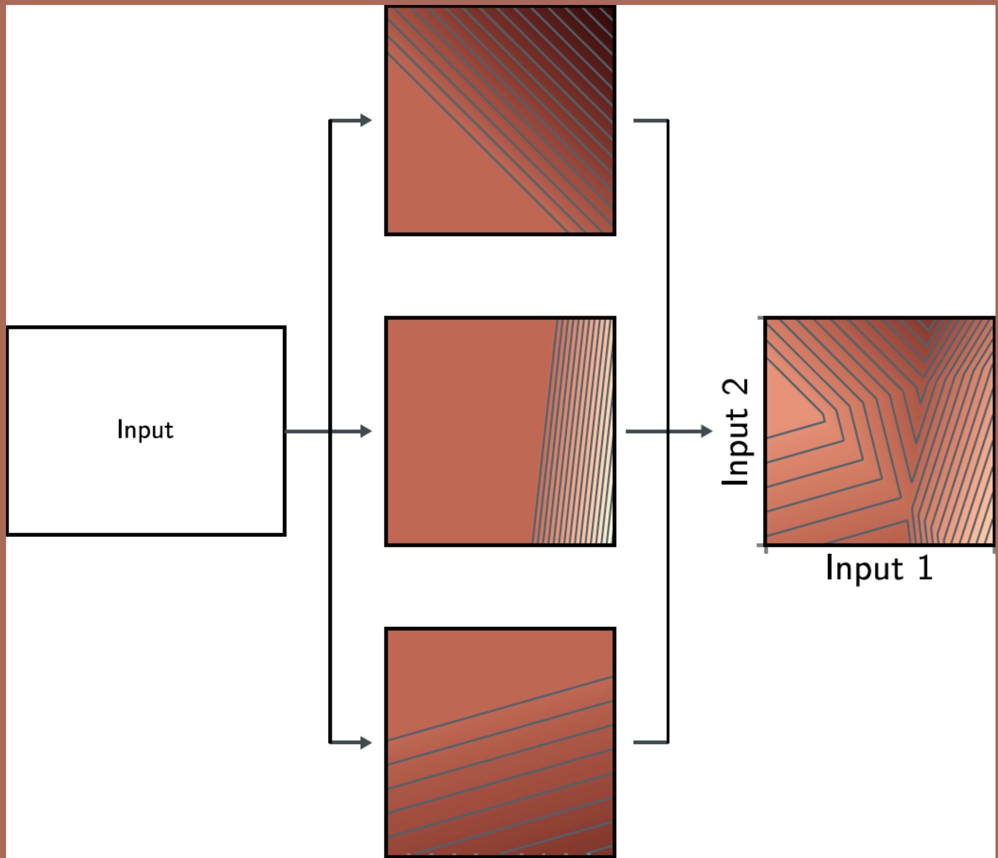


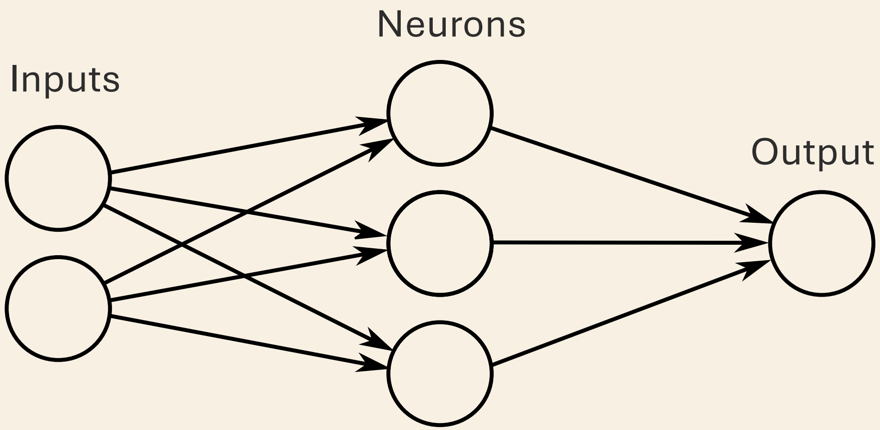

Input
Input 2
Input 1


Output
Input 2
Input 1


$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$
1.0
0.0
-1.0
-1.0
0.0
1.0
Input, x_2
Input, x_1


Input 2
Input 1

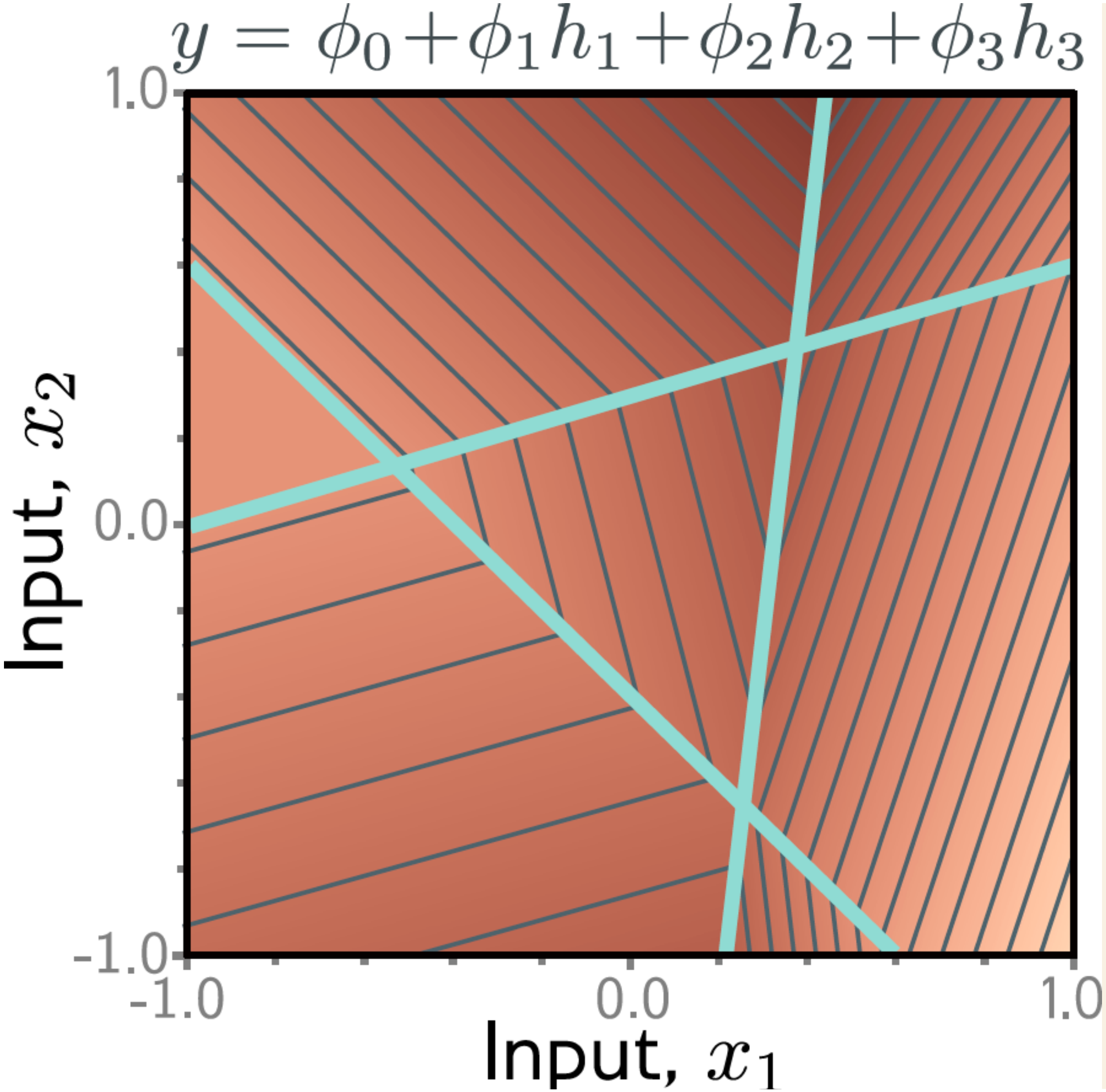




"neural network"

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11} x_1 + \theta_{12} x_2]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21} x_1 + \theta_{22} x_2]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31} x_1 + \theta_{32} x_2]$$

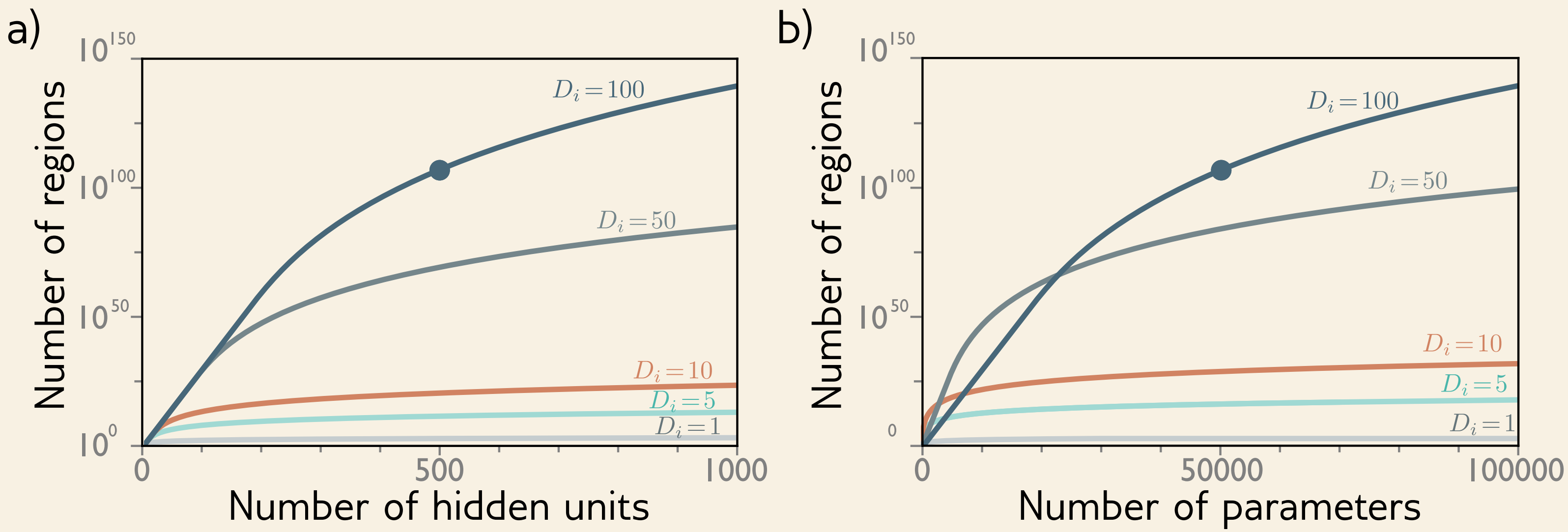


Number of output regions

- In general, each output consists of D dimensional convex polytopes
- With two inputs, and three outputs, we saw there were seven polygons.

Number of output regions

In general, each output consists of D dimensional convex polytopes



Highlighted point = 500 hidden units or 51,001 parameters

Number of regions:

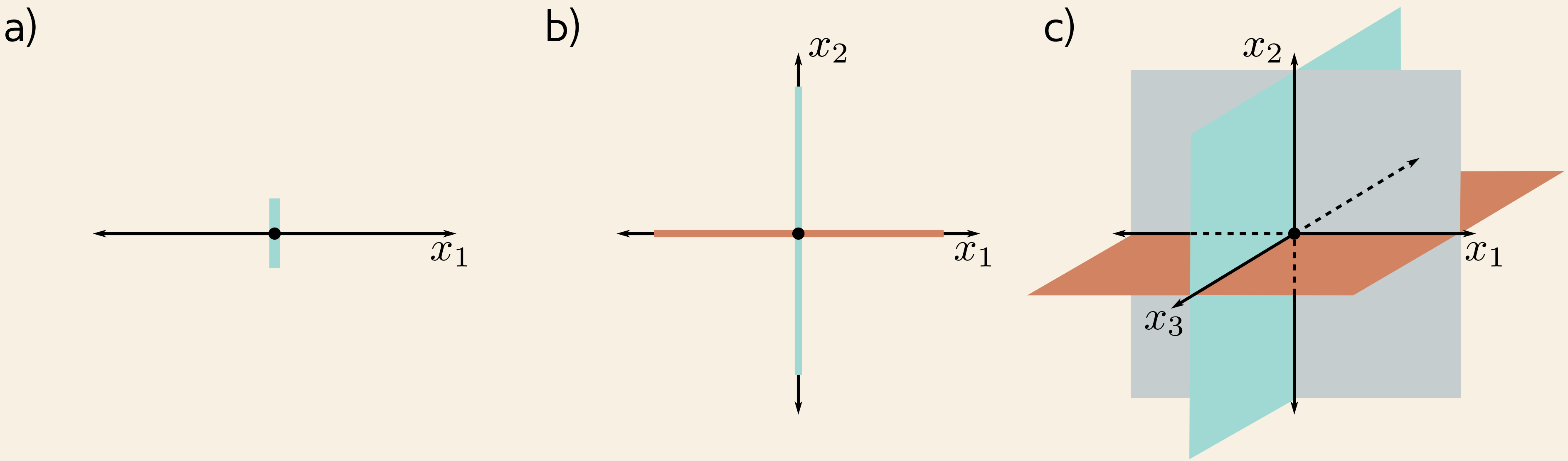
- Number of regions created by $D > D_i$ planes in D_i dimensions was proved by Zaslavsky (1975) to be:

$$\sum_{j=0}^{D_i} \binom{D}{j}$$

Binomial coefficients!

- How big is this? It's greater than 2^{Di} but less than 2^{D}.

Proof that more regions than 2^{Di}

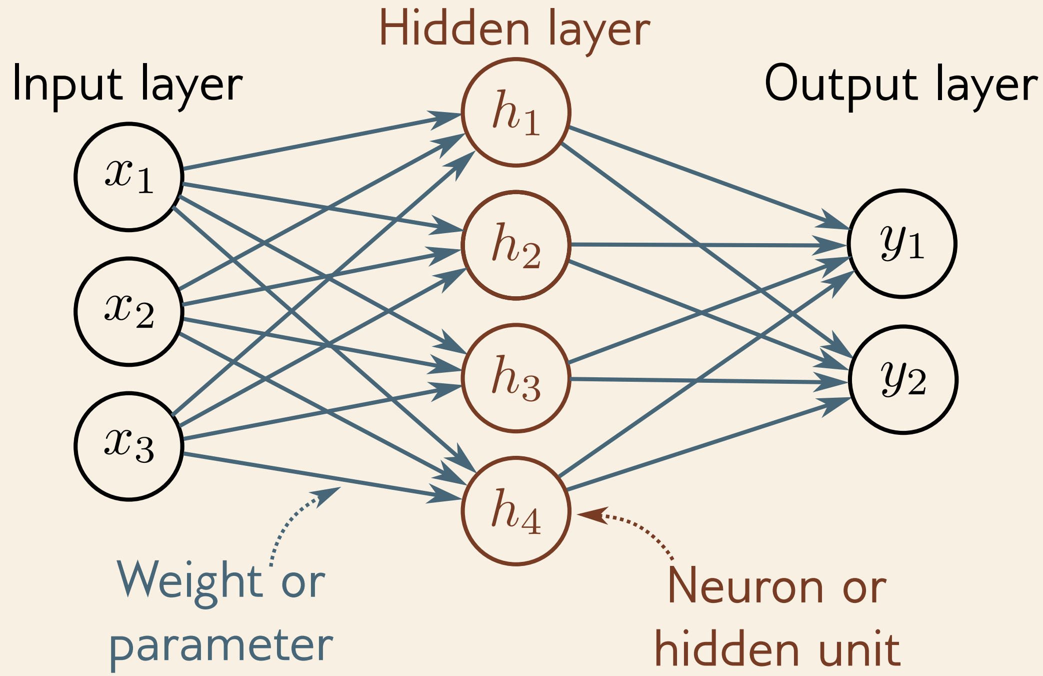


1D input with 1 hidden unit creates two regions (one joint)

2D input with 2 hidden units creates four regions (two lines)

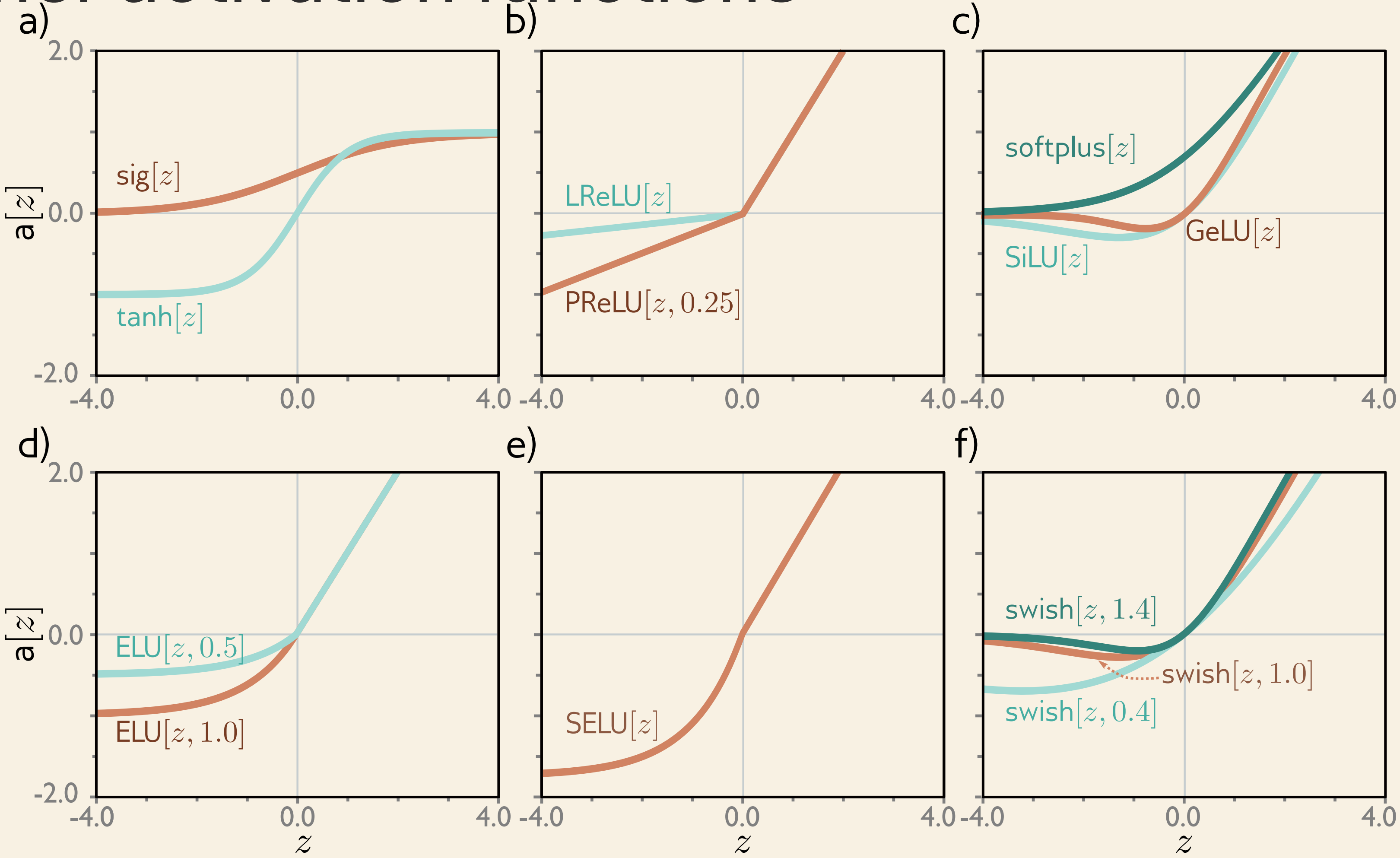
3D input with 3 hidden units creates eight regions (three planes)

Nomenclature



- Y-offsets = biases
- Slopes = weights
- Everything in one layer connected to everything in the next = fully connected network
- No loops = feedforward network
- Values after ReLU (activation functions) = activations
- Values before ReLU = pre-activations
- One hidden layer = shallow neural network
- More than one hidden layer = deep neural network
- Number of hidden units ≈ capacity

Other activation functions



A Simple Neural Network for Binary Classification

We’ll explore a minimal neural network that predicts whether an input belongs to **class 0 or 1**

Data

- (X, y): dataset and corresponding labels $X \in \mathbb{R}^{N\times 2}, \quad y \in \{01\}^{N},$

Model

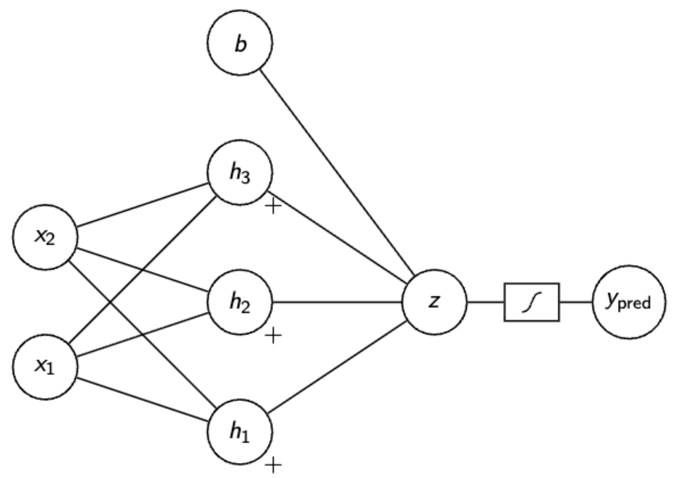
- $f_\theta(x): \mathbb{R}^2 \to [0, 1]$ outputs the **probability** that x belongs to class 1
- θ: all **learnable parameters** (weights and biases)

Activation Functions

- **ReLU:** +(x) = max(0, x). **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$

Binary Cross-Entropy (BCE) Loss

- Average over all samples: $\mathcal{L}(p, y) = \frac{1}{N}\sum_{i=1}^{N} \ell_i\,(p_i, y_i)$



$$f_\theta(\mathbf{x}) = \sigma\big(\underbrace{\max(0, \mathbf{x}\mathbf{w}_1)\mathbf{w}_2 + b}_{\mathbf{z}}\big)$$

Computing Gradients

We want to calculate the **gradient of the Binary Cross-Entropy (BCE) loss** with respect to the network parameters w_1, w_2,and b.

- **Binary Cross-Entropy Loss**
- $\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \ell_i \, \mathrm{f(x)}$
- Where our objective:
- $\nabla_{w_1,w_2,b} \, \mathcal{L}(f_{\theta}, (X) y)$

Chain Rule

Output layer

$$\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2}. \qquad \frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Hidden Layer

$$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial h} \cdot \frac{\partial h}{\partial (x w_1)} \cdot \frac{\partial (x w_1)}{\partial w_1}$$

Where $h=\max(0,xw_1)$

Gradient of the Loss w.r.t. Predictions

The gradient of the scalar loss with respect to the prediction vector is itself a **vector**:

$$\frac{\partial \ell}{\partial \hat{y}} = \begin{bmatrix} \frac{\partial \ell}{\partial \hat{y}_1} \\ \frac{\partial \ell}{\partial \hat{y}_2} \\ \vdots \\ \frac{\partial \ell}{\partial \hat{y}_N} \end{bmatrix}^{\mathrm{T}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

Each component measures how the loss changes when a single prediction $\hat{y}_i$ changes.

Derivative of Predictions w.r.t. Pre-Activations

Both $\hat{y}$ and z are vectors of size N. Since the output activation is applied **element-wise**: $\hat{y} = \sigma(z)$ then the derivative is also **element-wise**:

$$\frac{\partial \hat{y}}{\partial z} = \sigma(z)\big(1 - \sigma(z)\big) = \hat{y}(1 - \hat{y})$$

Computing $\frac{\partial z}{\partial h}$ and $\frac{\partial \mathrm{h}}{\partial \mathrm{w}_2}$

$z = hw_2 + b$, in matrix form $\begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ \vdots & \vdots & \vdots \\ h_{N1} & h_{N2} & h_{N3} \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \end{bmatrix} + b$

where $h \in \mathbb{R}^{N\times 3}$, $w_2 \in \mathbb{R}^{3\times 1}$, and $b \in \mathbb{R}$(broadcasted over all Nsamples).

The gradient of z with respect to a single weight $w_{2,j}$ depends on **all samples**:

$$\frac{\partial z}{\partial w_{2,j}} = \begin{bmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{Nj} \end{bmatrix}.$$

Gradient with respect to the Bias b

In the equation $z = hw_2 + b$, the bias term b is **broadcasted** across all N samples. We can make this explicit as:

$$z = hw_2 + \mathbf{1}^\top b,$$

where $\mathbf{1} \in \mathbb{R}^N$ is a vector of ones.

Derivative $\frac{\partial z}{\partial b} = \mathbf{1}^\top$

Computing $\frac{\partial z}{\partial h}$

$z = hw_2 + b$, in matrix form $\begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ \vdots & \vdots & \vdots \\ h_{N1} & h_{N2} & h_{N3} \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \end{bmatrix} + b$

where $h \in \mathbb{R}^{N\times 3}$, $w_2 \in \mathbb{R}^{3\times 1}$, and $b \in \mathbb{R}$ (broadcasted over all N samples).

For each sample i: $z_i = h_{i1}w_{2,1} + h_{i2}w_{2,2} + h_{i3}w_{2,3}$.

The gradient of z_i with respect to the row vector h_i is: $\frac{\partial z_i}{\partial h_i} = w_2^\top$.

Computing the Gradient with respect to w_1

To compute $\frac{\partial \mathcal{L}}{\partial w_1}$, we need two components: $\frac{\partial h}{\partial (xw_1)}$ *and* $\frac{\partial (xw_1)}{\partial w_1}$.

For the ReLU function $h = ReLU(xw_1)$, the derivative is given by:

$$\frac{\partial h_{ij}}{\partial \left(xw_1\right)_{ij}} = \begin{cases} 1, & if \left(x \cdot w_1\right)_{ij} > 0 \\ 0, & otherwise. \end{cases}$$ or equivalently $\frac{\partial h}{\partial (xw_1)} = \mathbf{1}_{xw_1>0}$,

a binary mask that passes gradients only where activations are positive.

Derivative of the linear transformation

The forward pass is: $h = xw_1$ where:

$$x = \begin{bmatrix} x_{11} & x_{12} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}, w_1 = \begin{bmatrix} w_{1,11} & w_{1,12} & w_{1,13} \\ w_{1,21} & w_{1,22} & w_{1,23} \end{bmatrix}.$$

Thus, the gradient of xw_1 with respect to w_1 depends directly on the input matrix x:

$$\frac{\partial(xw_1)}{\partial w_1} = x.$$

Each **column** of w_1 interacts with **all input features**, since every output dimension receives contributions from all columns of x.

Solution

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \big(\hat{y}(1 - \hat{y})\big) \cdot (h)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \big(\hat{y}(1 - \hat{y})\big) \cdot (1^T)$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial h} \cdot \frac{\partial h}{\partial (x w_1)} \cdot \frac{\partial (x w_1)}{\partial w_1}$$
$$= \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \big(\hat{y}(1 - \hat{y})\big) \cdot (w_2^{\mathsf{T}}) \cdot \big(1_{x w_1 > 0}\big) \cdot (x)$$

Slides from Understanding deep learning S. Prince Chapter 3