Deep Learning

02 - Shallow And Deep Networks

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Shallow Neural Networks

(Chapter 3)

SAPIENZA
UNIVERSITÀ DI ROMA

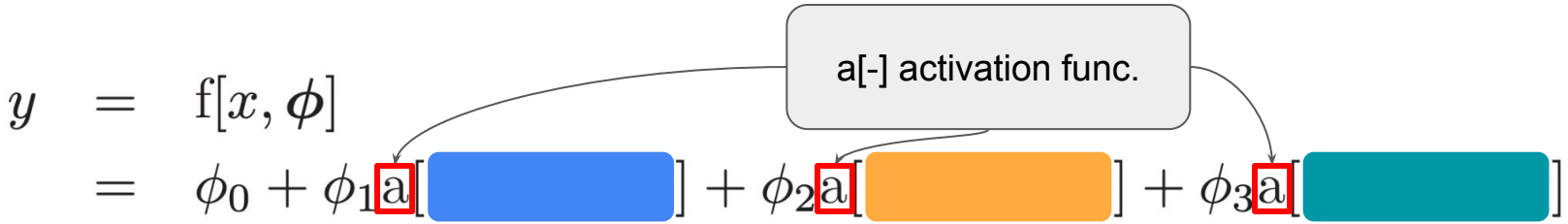
Introduction to Shallow Neural Networks

- Shallow neural networks are functions **y** = f[**x**, **ϕ**] with parameters **ϕ** that map multivariate inputs **x** to multivariate outputs **y**.
- We introduce the main ideas using an example network f[**x**, **ϕ**] that maps a scalar input **x** to a scalar output **y** and has ten parameters **ϕ** = {ϕ_0, ϕ_1, ϕ_2, ϕ_3, θ_{10}, θ_{11}, θ_{20}, θ_{21}, θ_{30}, θ_{31}}:

$$
\begin{aligned}
y &= f[x, \boldsymbol{\phi}] \\
&= \phi_0 + \phi_1 \quad + \phi_2 \quad + \phi_3
\end{aligned}
$$

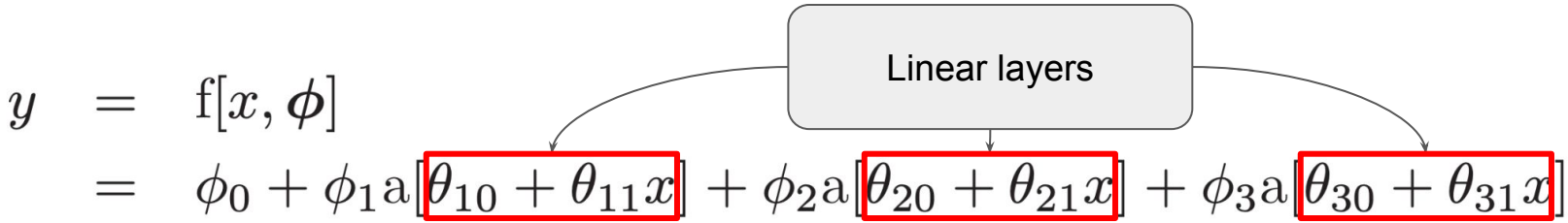
Introduction to Shallow Neural Networks

- Shallow neural networks are functions **y** = f[**x**, **ϕ**] with parameters **ϕ** that map multivariate inputs **x** to multivariate outputs **y**.
- We introduce the main ideas using an example network f[**x**, **ϕ**] that maps a scalar input **x** to a scalar output **y** and has ten parameters **ϕ** = {ϕ_0, ϕ_1, ϕ_2, ϕ_3, θ_{10}, θ_{11}, θ_{20}, θ_{21}, θ_{30}, θ_{31}}:



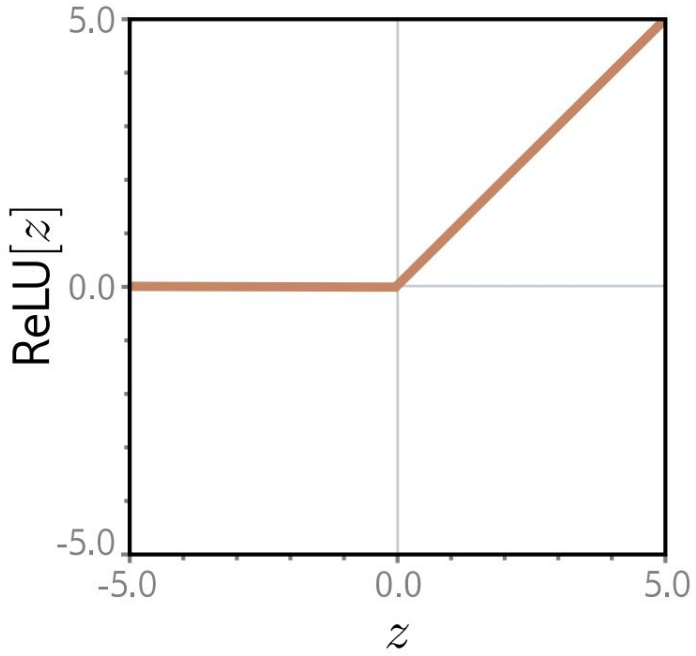
Introduction to Shallow Neural Networks

- Shallow neural networks are functions **y** = f[**x**, **ϕ**] with parameters **ϕ** that map multivariate inputs **x** to multivariate outputs **y**.
- We introduce the main ideas using an example network f[**x**, **ϕ**] that maps a scalar input **x** to a scalar output **y** and has ten parameters **ϕ** = {ϕ_0, ϕ_1, ϕ_2, ϕ_3, θ_{10}, θ_{11}, θ_{20}, θ_{21}, θ_{30}, θ_{31}}:



ReLU Activation

- It's the most common choice $\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$

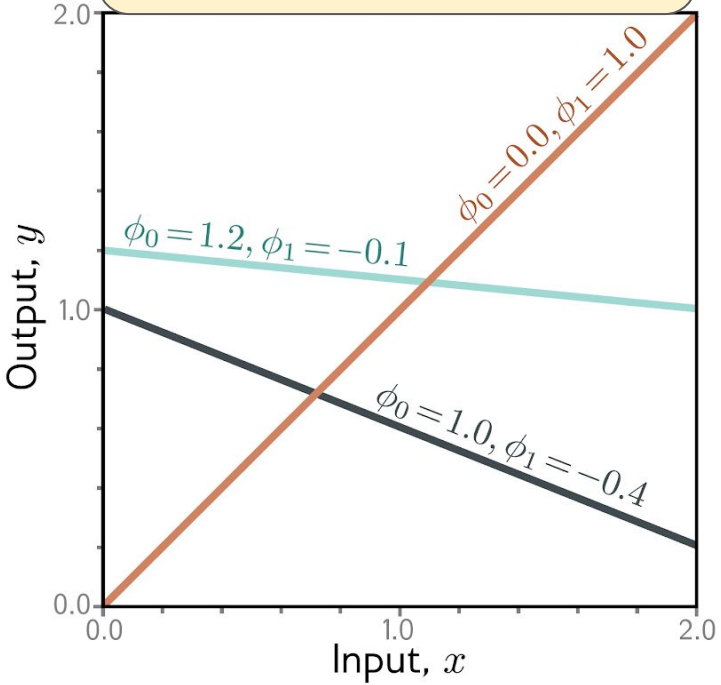


Family of functions defined by the

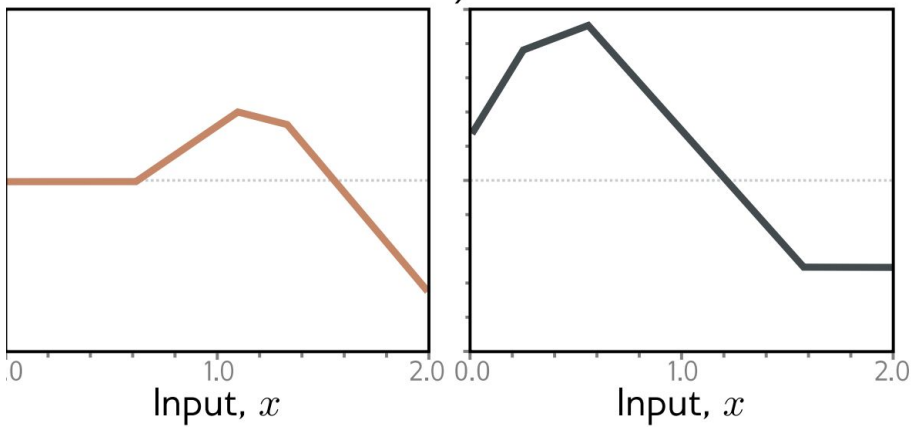
Compare with…

What's the main observation we can make about the "shape" of the three functions?

$$\ldots{}_1 x] + \phi_2 \mathrm{a}[\theta_{20} + \theta$$



c)



Recapping…

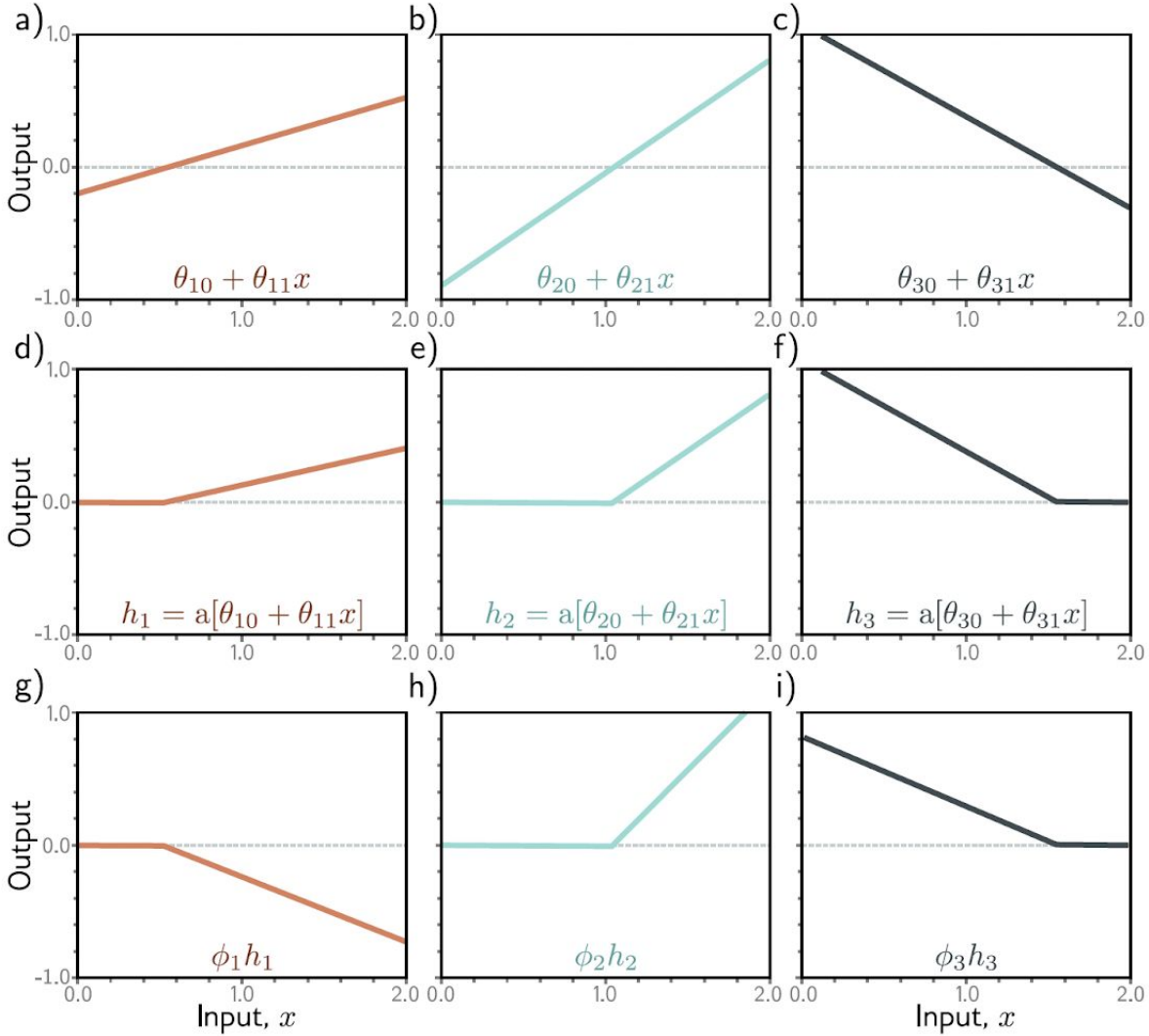
- We introduce three temp variables

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

  we refer to h_1, h_2, and h_3 to as **Hidden Units**.
- By combining these hidden units with a linear function we get:

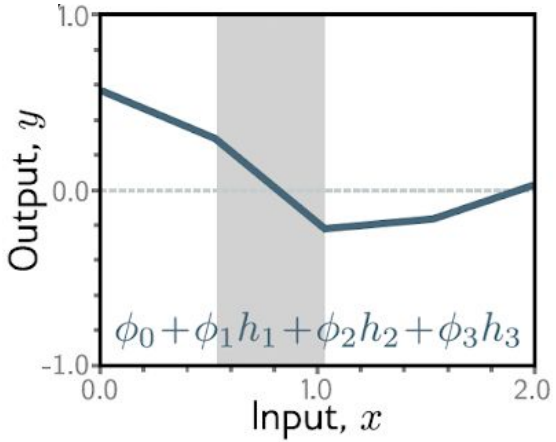
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

- Each hidden unit contains a linear function $\theta_{\bullet 0} + \theta_{\bullet 1}x$ of the input, and that line is clipped by the ReLU function $a[\bullet]$ below zero. The positions where the three lines cross zero become the three “joints” in the final output. The three clipped lines are then weighted by ϕ_1, ϕ_2, and ϕ_3, respectively. Finally, the offset ϕ_0 is added, which controls the overall height of the final function.



Summing up (pun intended 👀)…

- The clipped and weighted functions are summed, and an offset ϕ_0 that controls the height is added. Each of the four linear regions corresponds to a different activation pattern in the hidden units. In the shaded region, h_2 is inactive (clipped), but h_1 and h_3 are both active.



Summing up (pun intended 👀)…

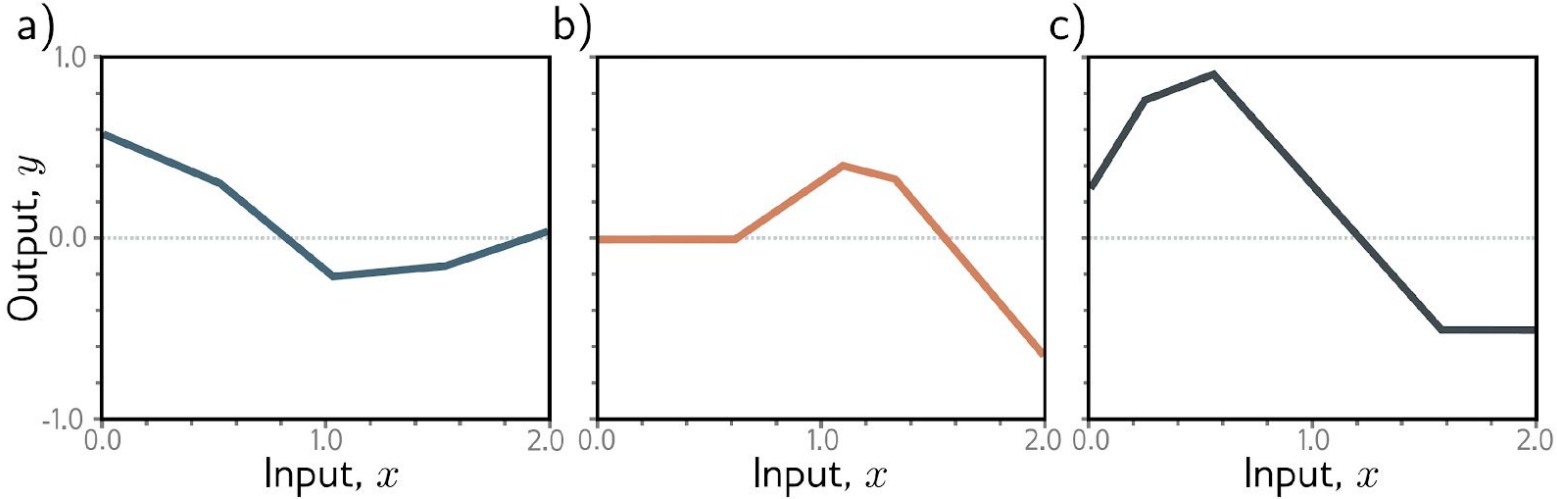
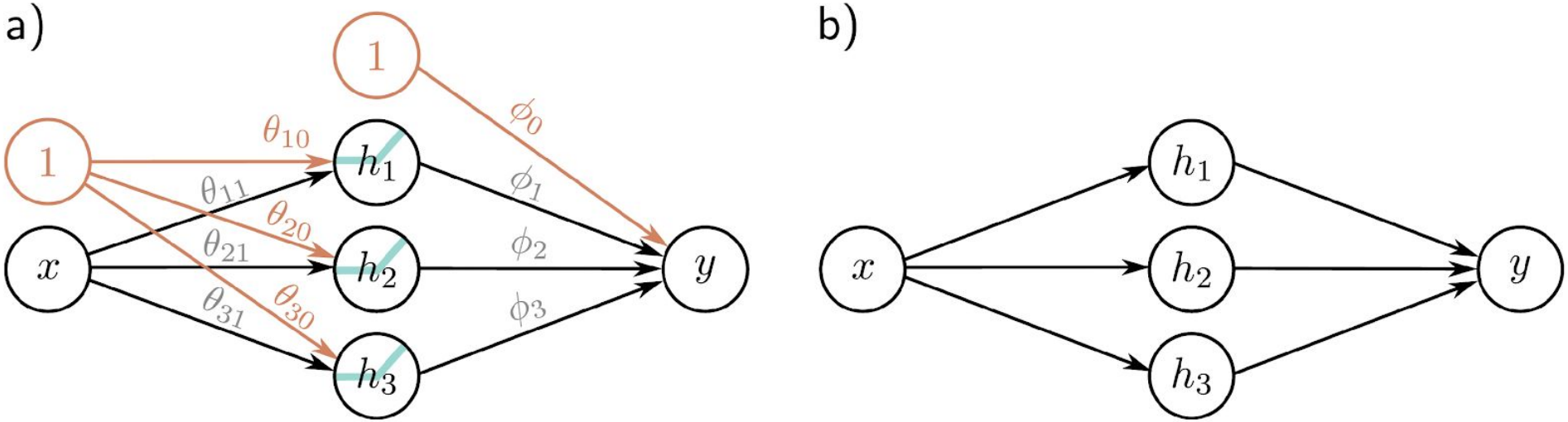
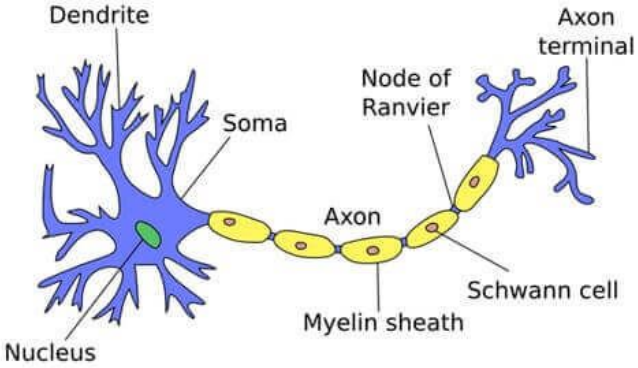


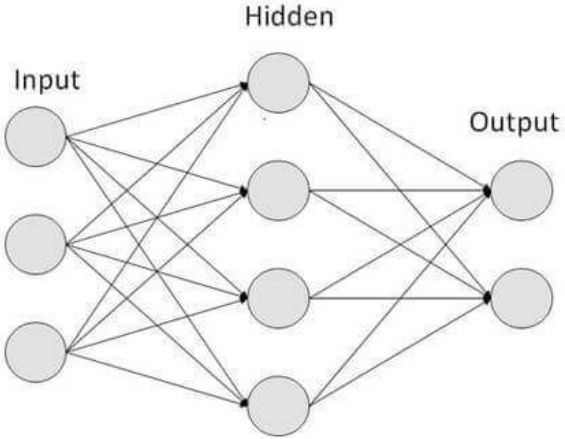
Figure 3.2 Family of functions defined by equation 3.1. a–c) Functions for three different choices of the ten parameters ϕ. In each case, the input/output relation is piecewise linear. However, the positions of the joints, the slopes of the linear regions between them, and the overall height vary.

Maybe more usually depicted like this…



Let’s stop using this…





Colab time: Shallow Networks I

https://colab.research.google.com/drive/1kxlRuMPhOy_I_y5XyMbFBXIINaB65_Id

Colab time: Shallow Networks II

https://colab.research.google.com/drive/16G7zO9XE6d7Q99CCjISIz0CSnRmqjRvB

Universal Approximation Theorem

- Consider the case with D hidden units where the d^{th} hidden unit is:

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

  and these are combined linearly to create the output:

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

- The number of hidden units in a shallow network is a measure of the network capacity. With ReLU activation functions, the output of a network with D hidden units has at most D joints and so is a piecewise linear function with at most D + 1 linear regions.
- As we add more hidden units, the model can approximate more complex functions.

Universal Approximation Theorem

- With enough capacity (hidden units), a shallow network can describe any continuous 1D function defined on a compact subset of the real line to arbitrary precision.
- To see this, consider that every time we add a hidden unit, we add another linear region to the function. As these regions become more numerous, they represent smaller sections of the function, which are increasingly well approximated by a line.
- The **universal approximation theorem** proves that for any continuous function, there exists a *shallow* network that can approximate this function to any specified precision.

Universal Approximation Theorem

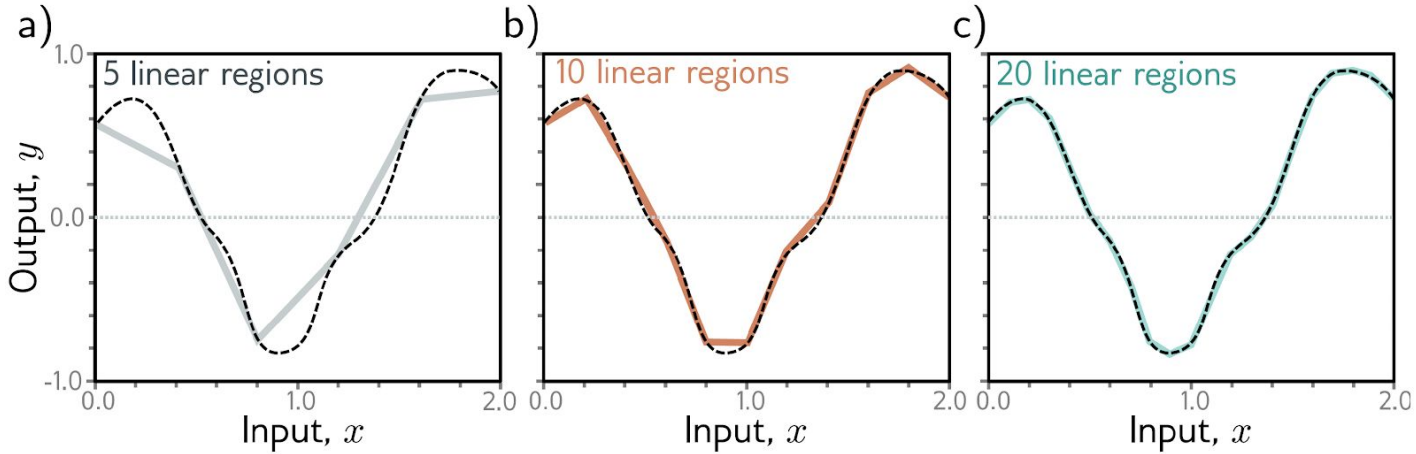
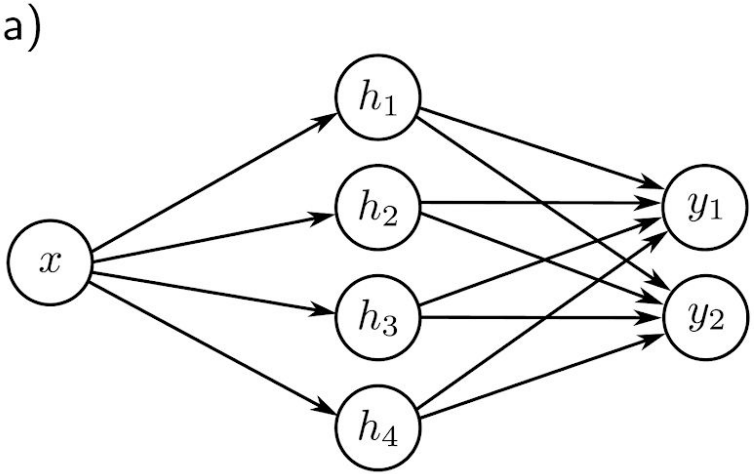
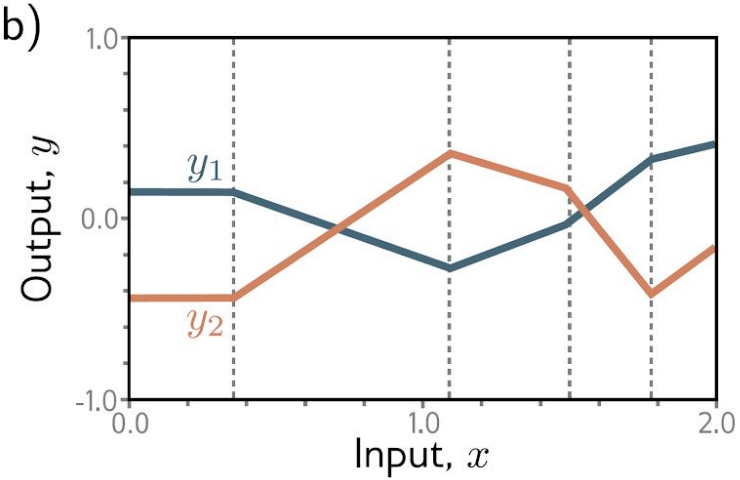


Figure 3.5 Approximation of a 1D function (dashed line) by a piecewise linear model. a–c) As the number of regions increases, the model becomes closer and closer to the continuous function. A neural network with a scalar input creates one extra linear region per hidden unit. The universal approximation theorem proves that, with enough hidden units, there exists a shallow neural network can describe any given continuous function defined on a compact subset of $\mathbb{R}^D$ to arbitrary precision.

Multivariate Outputs





$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$$

$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4.$$

units, but the slopes and overall height may differ.

Multivariate Inputs

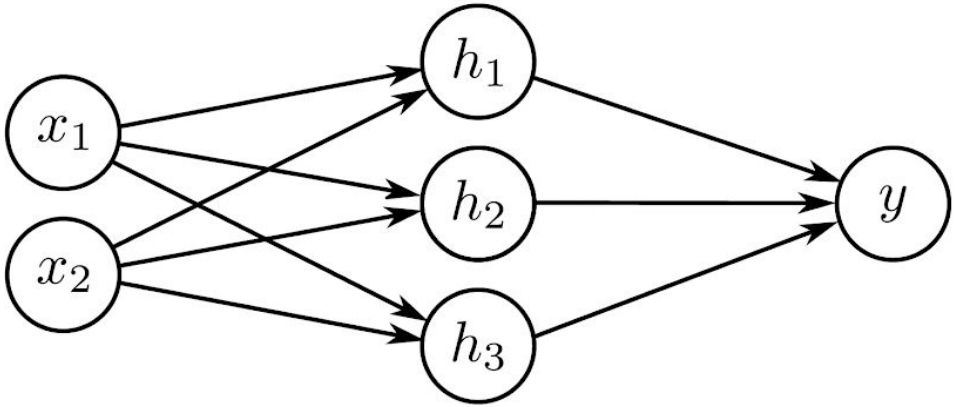
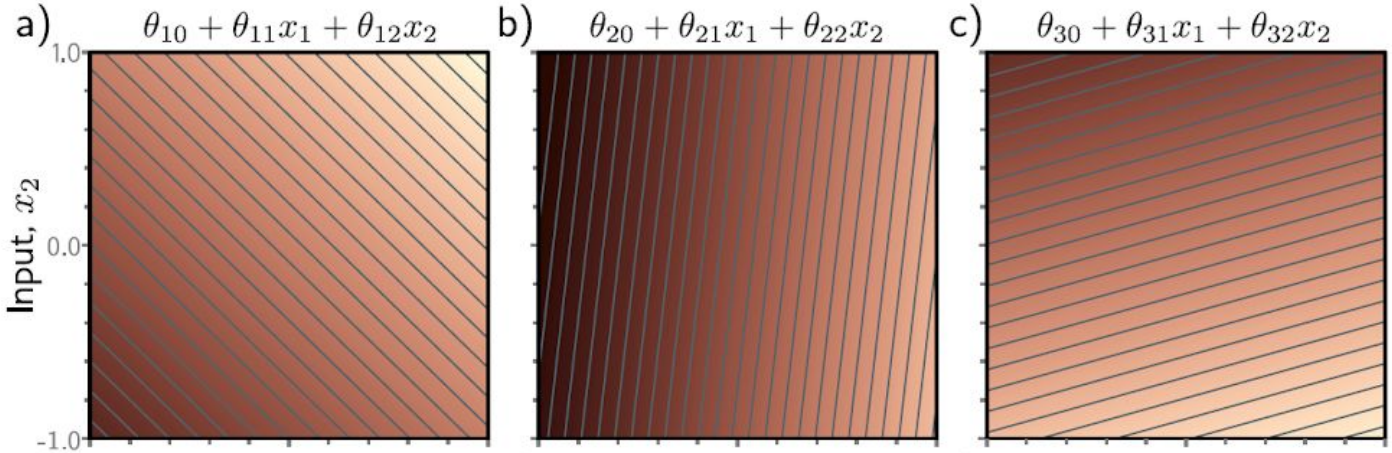


Figure 3.7 Visualization of neural network with 2D multivariate input $\mathbf{x} = [x_1, x_2]^T$ and scalar output y.

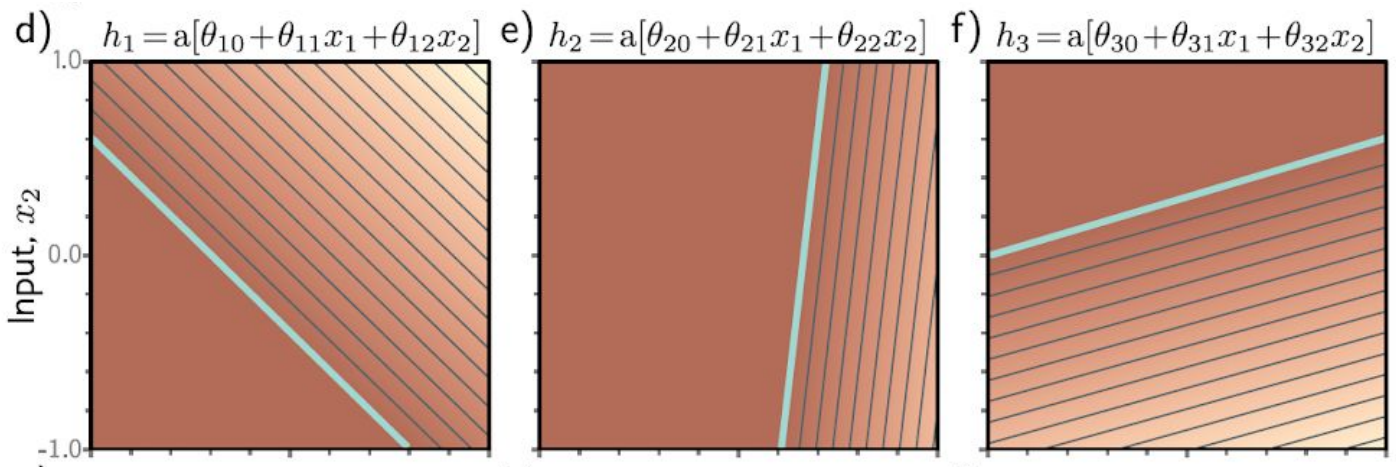
What happens when we add inputs and outputs?

- Processing in network with two inputs $\mathbf{x} = [x_1, x_2]^T$, three hidden units h_1, h_2, h_3, and one output y.
- The input to each hidden unit is a linear function of the two inputs, which corresponds to an oriented plane. Brightness indicates function output.
- For example, in panel (a), the brightness represents $\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$. Thin lines are contours.



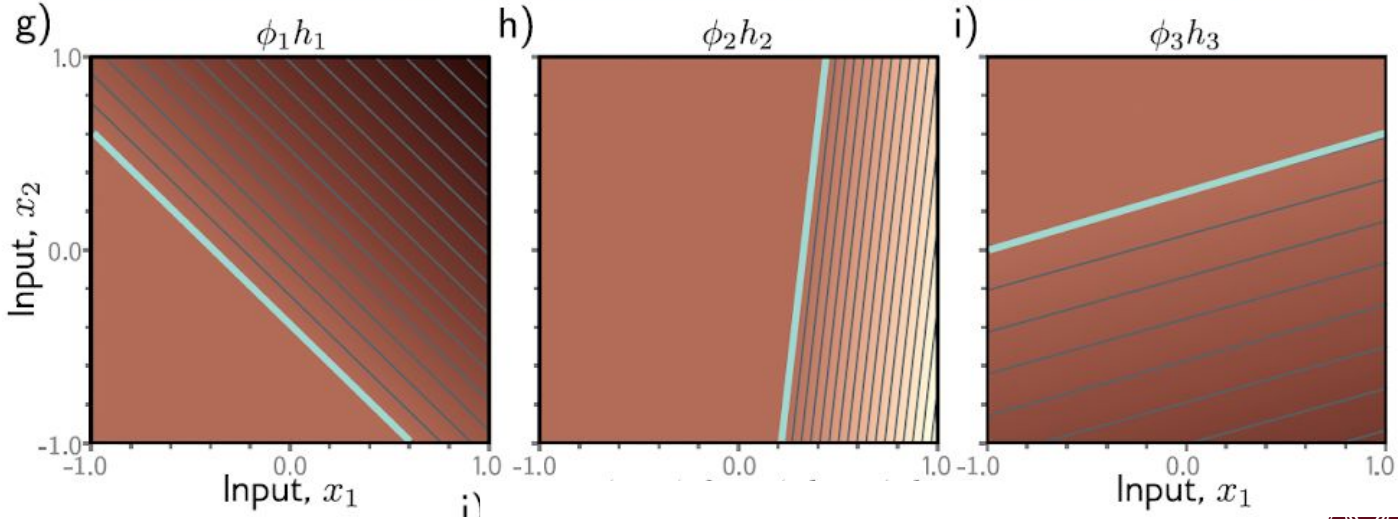
What happens when we add inputs and outputs?

- Each plane is clipped by the ReLU activation function
- Cyan lines are equivalent to “joints” in the 1-D case



What happens when we add inputs and outputs?

- The clipped planes are then weighted



Summing up (pun again intended 👀)…

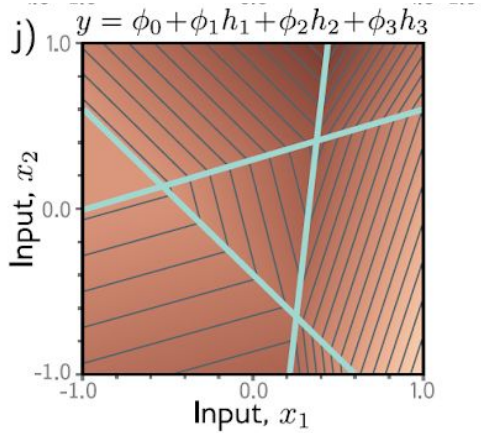


Figure 3.8 Processing in network with two inputs $\mathbf{x} = [x_1, x_2]^T$, three hidden units h_1, h_2, h_3, and one output y. a–c) The input to each hidden unit is a linear function of the two inputs, which corresponds to an oriented plane. Brightness indicates function output. For example, in panel (a), the brightness represents $\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$. Thin lines are contours. d–f) Each plane is clipped by the ReLU activation function (cyan lines are equivalent to "joints" in figures 3.3d–f). g-i) The clipped planes are then weighted, and j) summed together with an offset that determines the overall height of the surface. The result is a continuous surface made up of convex piecewise linear polygonal regions.

Hidden Units vs. Convex Piece-wise Linear Regions

- Each hidden unit defines a hyperplane that delineates the part of space where this unit is active from the Shallow network regions where it is not (cyan lines in d–f above).
- If we had the same number of hidden units as input dimensions D_i, we could align each hyperplane with one of the coordinate axes.
  - For two input dimensions, this would divide the space into four quadrants.
  - For three dimensions, this would create eight octants, and
  - for D_i dimensions, this would create 2^{D_i} orthants.
- Shallow neural networks usually have more hidden units than input dimensions, so they typically create more than 2^{D_i} linear regions.

Hidden Units vs. Convex Piece-wise Linear Regions

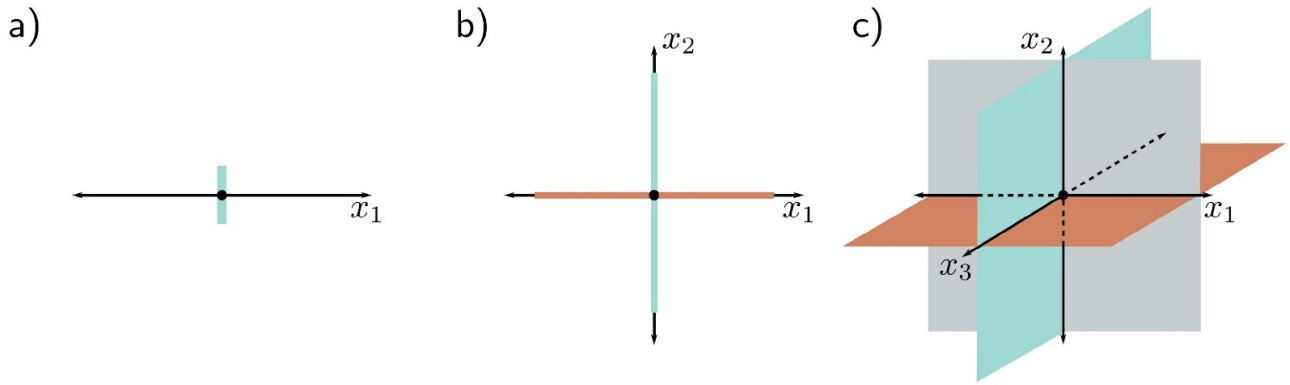


Figure 3.10 Number of linear regions vs. input dimensions. a) With a single input dimension, a model with one hidden unit creates one joint, which divides the axis into two linear regions. b) With two input dimensions, a model with two hidden units can divide the input space using two lines (here aligned with axes) to create four regions. c) With three input dimensions, a model with three hidden units can divide the input space using three planes (again aligned with axes) to create eight regions. Continuing this argument, it follows that a model with D_i input dimensions and D_i hidden units can divide the input space with D_i hyperplanes to create 2^{D_i} linear regions.

Hidden Units vs. Convex Piece-wise Linear Regions

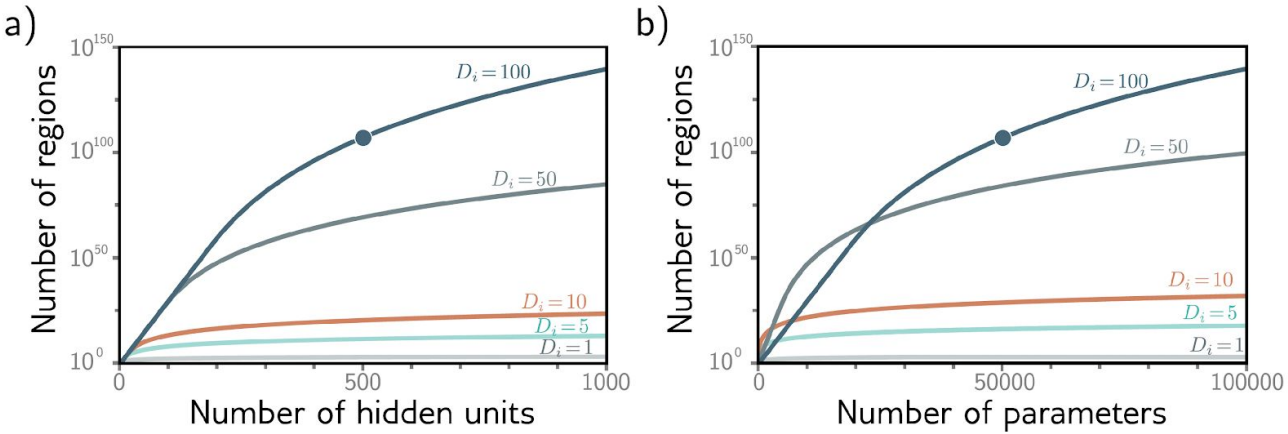


Figure 3.9 Linear regions vs. hidden units. a) Maximum possible regions as a function of the number of hidden units for five different input dimensions $D_i = \{1, 5, 10, 50, 100\}$. The number of regions increases rapidly in high dimensions; with $D = 500$ units and input size $D_i = 100$, there can be greater than 10^{107} regions (solid circle). b) The same data are plotted as a function of the number of parameters. The solid circle represents the same model as in panel (a) with $D = 500$ hidden units. This network has $51,001$ parameters and would be considered very small by modern standards.

Colab time: Number of Hidden Layers

https://colab.research.google.com/drive/1Y9_tCB73A4h5O77QQhPSG0AlJU-I3hDQ

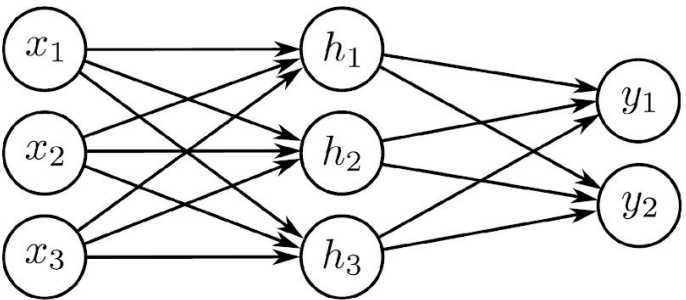
Shallow Networks: The Most General Case

- We now define a general equation for a shallow neural network **y** = f[**x**, **ϕ**]
  - **x** ∈ $\mathbb{R}^{D_i}$ to a multi-dimensional output **y** ∈ $\mathbb{R}^{D_o}$, **h** ∈ $\mathbb{R}^{D}$

$$h_d = a\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i\right]$$

- Combined linearly to create the output:

$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d$$



Terminology

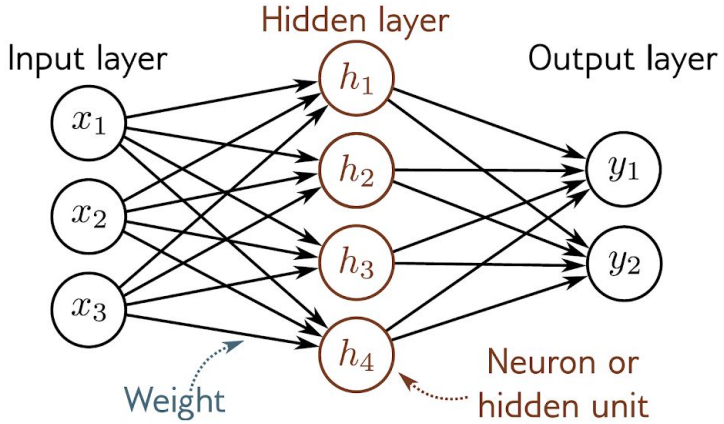


Figure 3.12 Terminology. A shallow network consists of an input layer, a hidden layer, and an output layer. Each layer is connected to the next by forward connections (arrows). For this reason, these models are referred to as feed-forward networks. When every variable in one layer connects to every variable in the next, we call this a fully connected network. Each connection represents a slope parameter in the underlying equation, and these parameters are termed weights. The variables in the hidden layer are termed neurons or hidden units. The values feeding into the hidden units are termed pre-activations, and the values at the hidden units (i.e., after the ReLU function is applied) are termed activations.

Activation Function (or Nonlinearities)

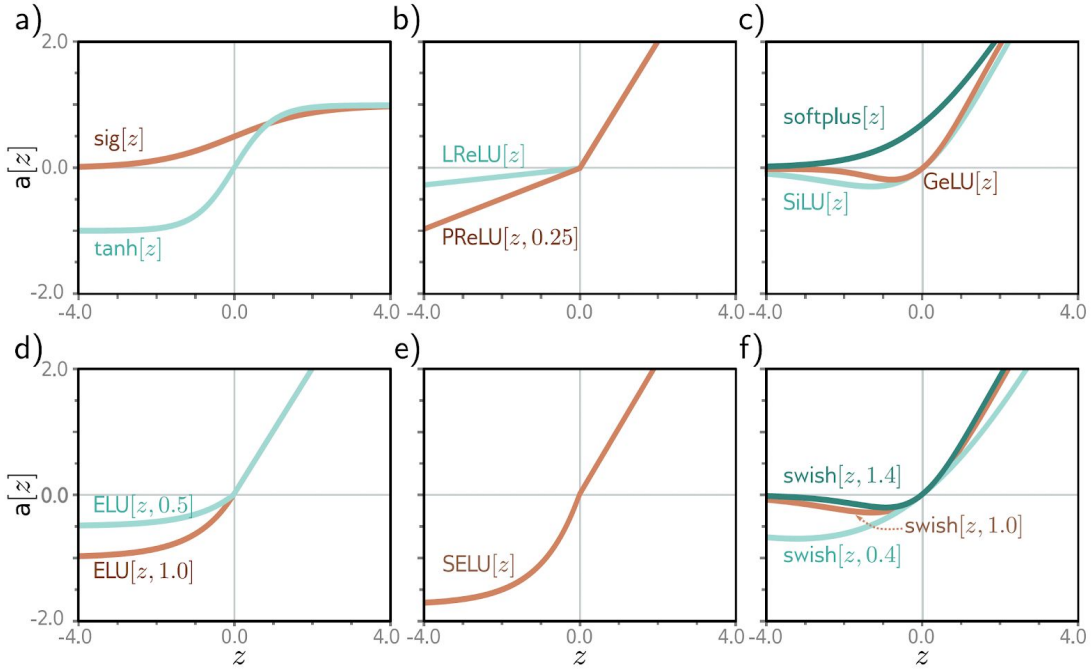


Figure 3.13 Activation functions. a) Logistic sigmoid and tanh functions. b) Leaky ReLU and parametric ReLU with parameter 0.25. c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit. d) Exponential linear unit with parameters 0.5 and 1.0, e) Scaled exponential linear unit. f) Swish with parameters 0.4, 1.0, and 1.4.

Colab time: Activation Functions (or nonlinearities)

https://colab.research.google.com/drive/1QPn1q5AKXnw3eKDDK1xLBS2rl9zodgs5

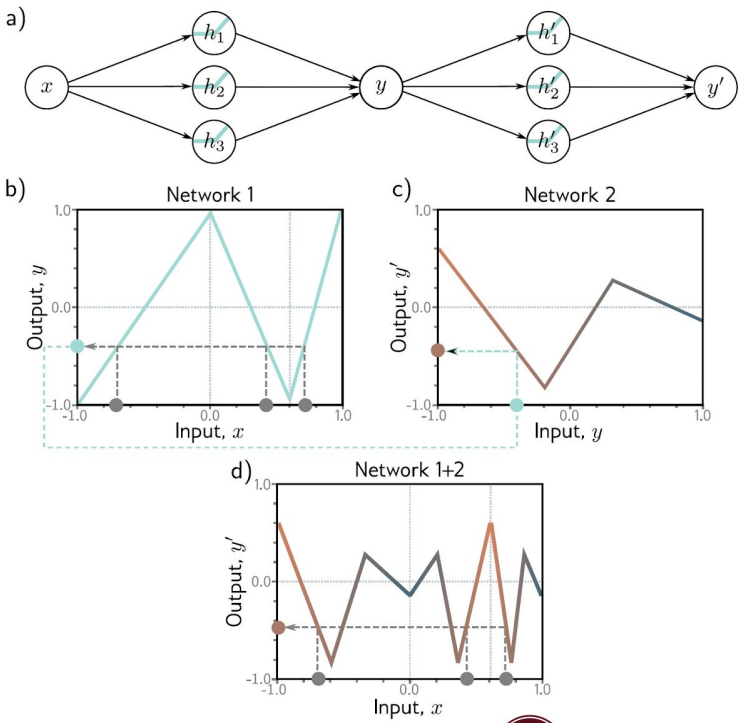
Deep Neural Networks

(Chapter 4)

SAPIENZA
UNIVERSITÀ DI ROMA

Deep Neural Networks as composition of 2 SNNs

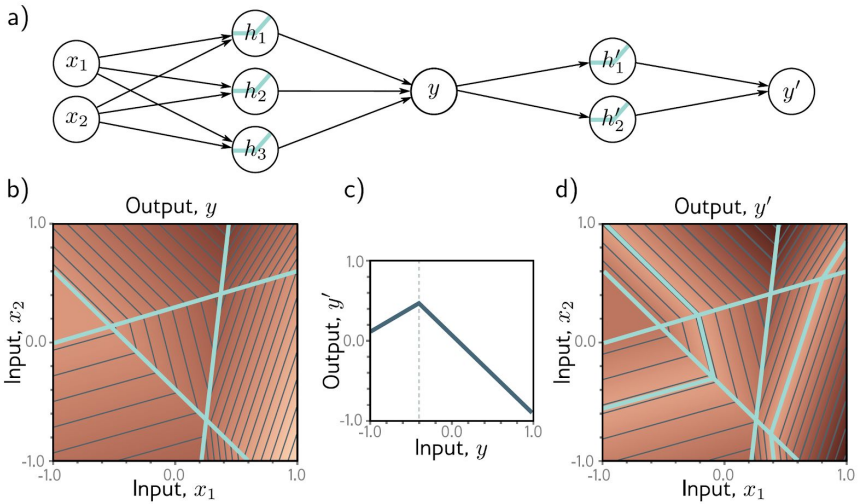
- The output y of the first network constitutes the input to the second network.
- The first network maps inputs x ∈ [−1, 1] to outputs y ∈ [−1, 1] using a function comprised of three linear regions that are chosen so that they alternate the sign of their slope. Multiple inputs x (gray circles) now map to the same output y (cyan circle)
- The second network defines a function comprising three linear regions that takes y and returns y′ (i.e., the cyan circle is mapped to the brown circle).
- The combined effect of these two functions when composed is that (i) three different inputs x are mapped to any given value of y by the first network and (ii) are processed in the same way by the second network; the result is that the function defined by the second network in panel (c) is duplicated three times, variously flipped and rescaled according to the slope of the regions of panel (b).



SAPIENZA
UNIVERSITÀ DI ROMA

Effect on convex regions

- The first network has three hidden units and takes two inputs x_1 and x_2 and returns a scalar output y. This is passed into a second network with two hidden units to produce y′.
- The first network produces a function consisting of seven linear regions, one of which is flat.
- The second network defines a function comprising two linear regions in y ∈ [−1, 1].
- When these networks are composed, each of the six non-flat regions from the first network is divided into two new regions by the second network to create a total of 13 linear regions.



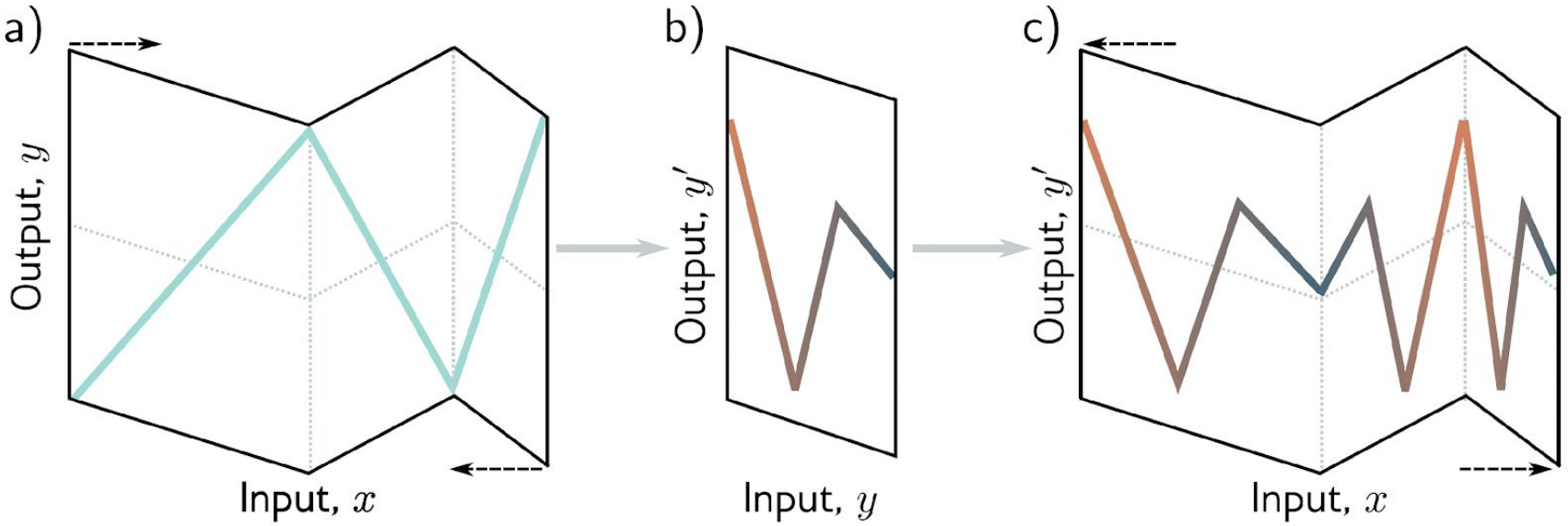


Figure 4.3 Deep networks as folding input space. a) One way to think about the first network from figure 4.1 is that it "folds" the input space back on top of itself. b) The second network applies its function to the folded space. c) The final output is revealed by "unfolding" again.

Colab time: Composing networks

https://colab.research.google.com/drive/1wj-dM9bemlAGRJMViAOBWK0XrUSfW8va

Multi Layer Perceptron (MLP)

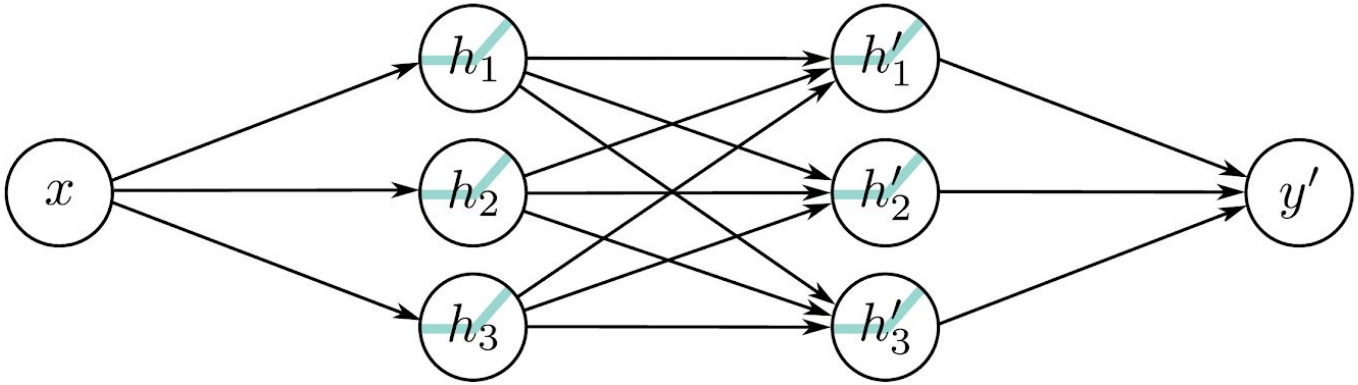
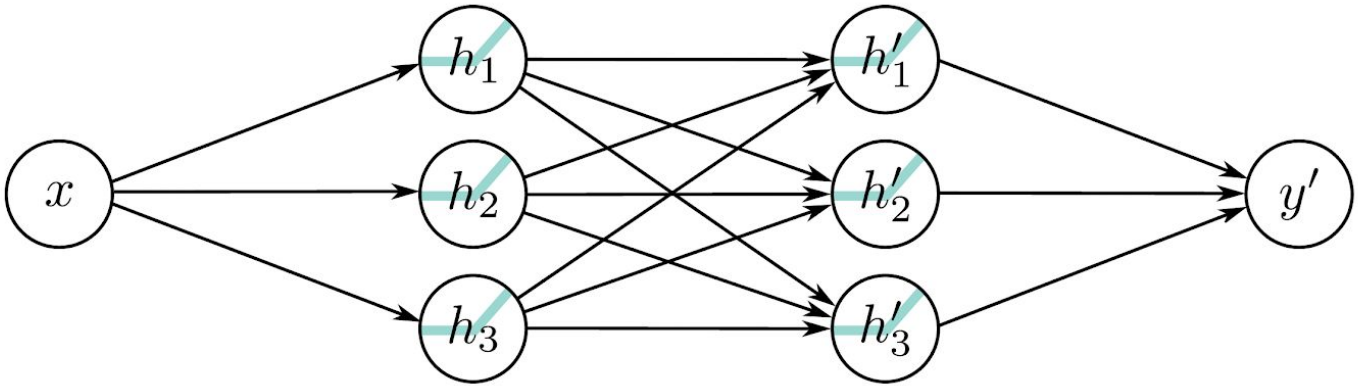


Figure 4.4 Neural network with one input, one output, and two hidden layers, each containing three hidden units.

Exercise

- Write the equation for (assume the nonlinearity/activation is ReLU)



Solution

- First Layer

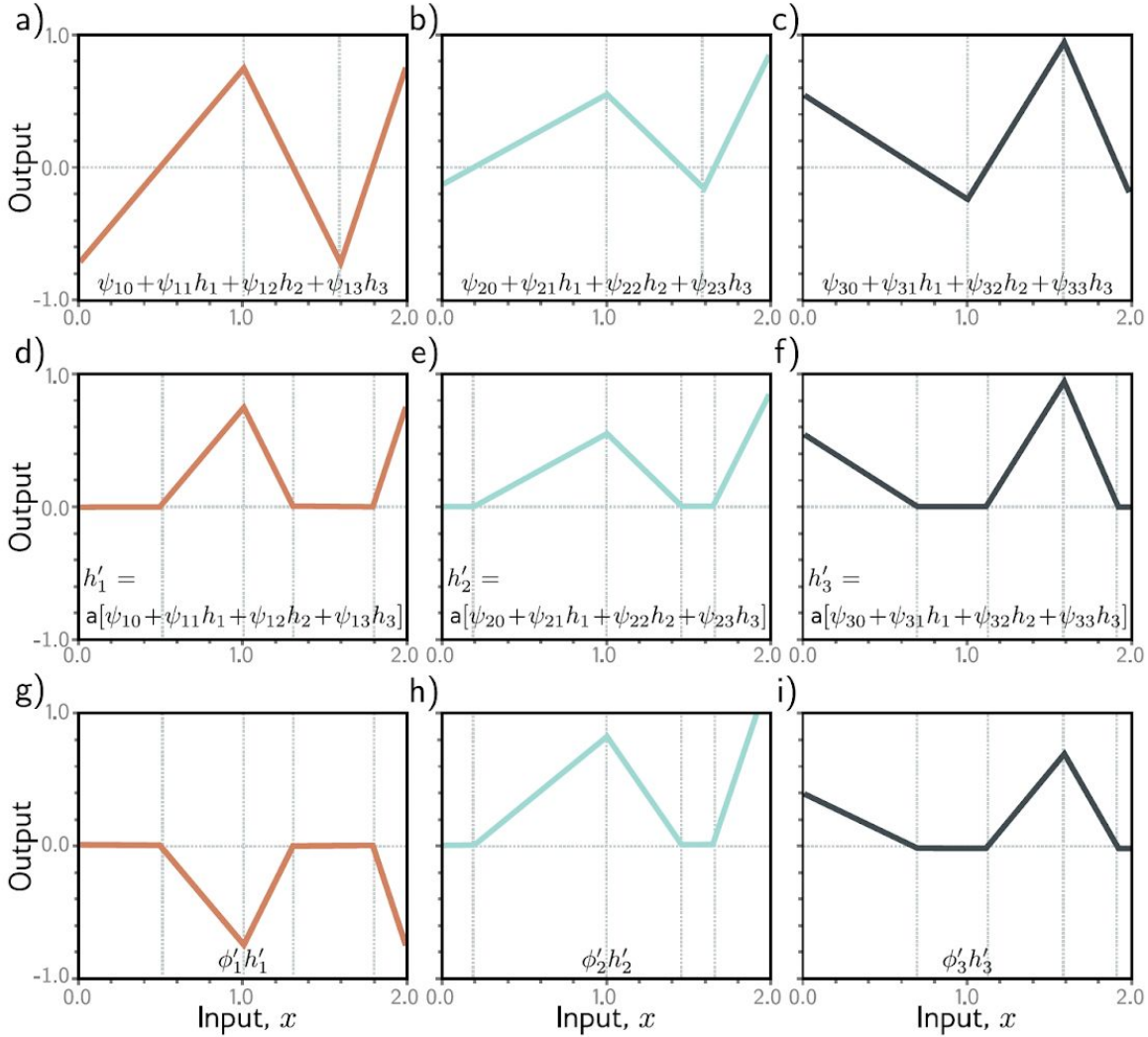
$$\begin{aligned} h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x] \\ h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x] \\ h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x] \end{aligned}$$

- Second Layer

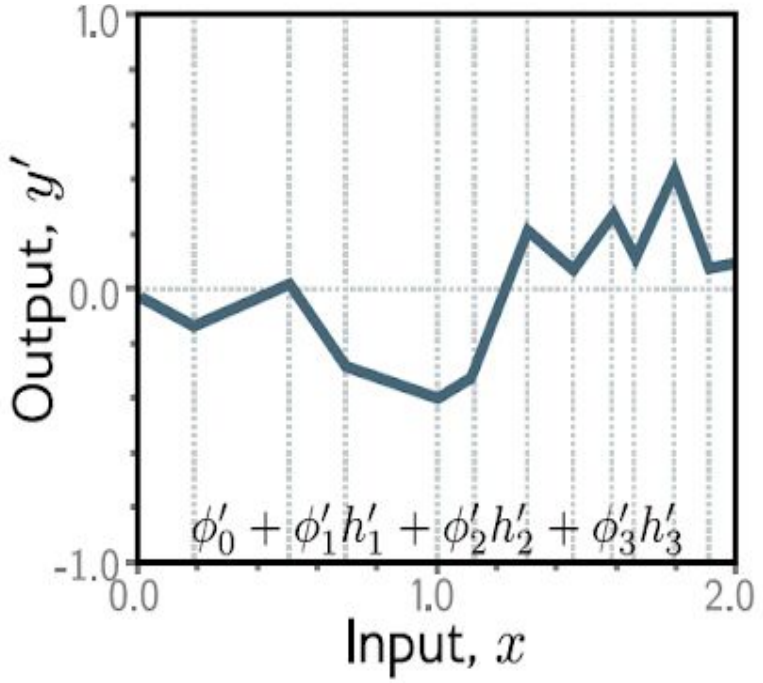
$$\begin{aligned} h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\ h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\ h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]. \end{aligned}$$

- Output

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$



Summing up (ok, it's a pun! 👀)…



Vector Notation

- First Layer

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right]$$

- Second Layer

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right]$$

- Output

$$y' = \phi'_0 + \begin{bmatrix} \phi'_1 & \phi'_2 & \phi'_3 \end{bmatrix} \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix}$$

Matrix Notation

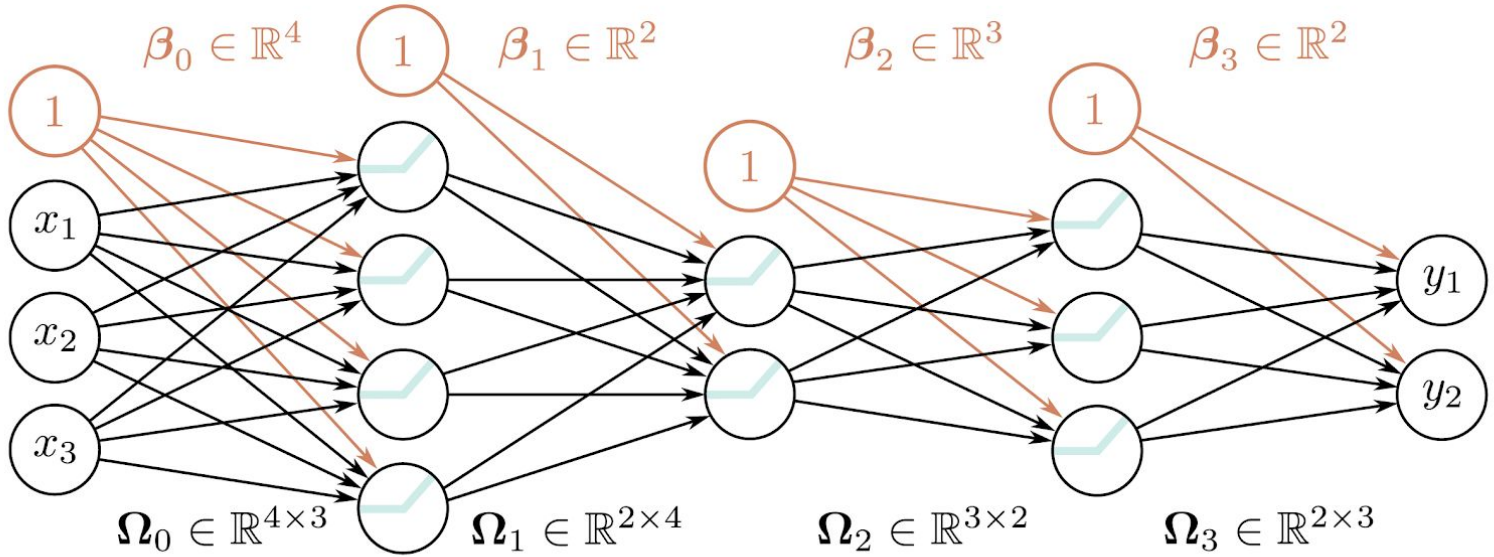
$$
\begin{aligned}
\mathbf{h} &= \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \\
\mathbf{h}' &= \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \\
y &= \boldsymbol{\phi}_0' + \boldsymbol{\phi}'\mathbf{h}'
\end{aligned}
$$

General Formulation and Notation

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}] \\
\mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{h}_1] \\
\mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{h}_2] \\
&\vdots \\
\mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{h}_{K-1}] \\
\mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{h}_K.
\end{aligned}
$$

- The parameters $\boldsymbol{\phi}$ of this model comprise all of these weight matrices and bias vectors $\boldsymbol{\phi} = \{\boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}_{k=0}^{K}$
- If the k-th layer has D_k hidden units, then the bias vector $\boldsymbol{\beta}_{k-1}$ will be of size D_k.
- The last bias vector $\boldsymbol{\beta}_K$ has the size D_o of the output.
- The first weight matrix $\boldsymbol{\Omega}_0$ has size $D_1 \times D_i$ where D_i is the size of the input.
- The last weight matrix $\boldsymbol{\Omega}_K$ is $D_o \times D_K$, and the remaining matrices $\boldsymbol{\Omega}_k$ are $D_{k+1} \times D_k$

3 Hidden Layers and Matrix Notation



Input, $\mathbf{x}$	Hidden layer, $\mathbf{h}_1$	Hidden layer, $\mathbf{h}_2$	Hidden layer, $\mathbf{h}_3$	Output, $\mathbf{y}$
$D_i = 3$	$D_1 = 4$	$D_2 = 2$	$D_3 = 3$	$D_o = 2$

Hyperparameters

- We can extend the deep network construction to more than two hidden layers
  - *modern networks might have more than a hundred layers with thousands of hidden units at each layer* (!!!)
- The number of hidden units in each layer is referred to as the width of the network, and the number of hidden layers as the depth. The total number of hidden units is a measure of the network's capacity.
- We denote the number of layers as K and the number of hidden units in each layer as $D_1, D_2, \ldots, D_K$.
  - These are examples of hyperparameters.
  - They are quantities chosen before we learn the model parameters (i.e., the slope and intercept terms). For fixed hyperparameters (e.g., K = 2 layers with $D_k = 3$ hidden units in each), the model describes a family of functions, and the parameters determine the particular function.

Depth Efficiency

- Both deep and shallow networks can model arbitrary functions, but some functions can be approximated much more efficiently with deep networks.
- Functions have been identified that require a shallow network with **exponentially more hidden units** to achieve an equivalent approximation to that of a deep network.
- This phenomenon is referred to as the depth efficiency of neural networks.
- This property is also attractive, but it's not clear that the real-world functions that we want to approximate fall into this category.

Exercise

- Imagine we have an input of size D_i = 6,220,800
- Imagine we want to process this with a 3-layer MLP with each hidden layer of sizes respectively D_1= 512, D_2= 1,024, D_3= 512
- Imagine we have a single output, D_o=1
- Assume we use bias
- How many parameters, i.e. $|\boldsymbol{\phi}|$, does our 3-layer MLP use?

Exercise

- Imagine we have an input of size D_i = 6,220,800
- Imagine we want to process this with a 3-layer MLP with each hidden layer of sizes respectively D_1= 512, D_2= 1,024, D_3= 512
- Imagine we have a single output, D_o=1
- Assume we use bias
- How many parameters, i.e. $|\phi|$, does our 3-layer MLP use?
- Solution: 3,186,100,736 parameters
- If each parameter is stored in a floating point we will need just to store the model:

  12,744,402,944 bytes, i.e., ~13GB!!!!!!!

Why D_i=6,220,800?

1920x1080

Colab time: Deep Neural Network

https://colab.research.google.com/drive/1m2iVH3tUerXr1bMH9Yi1u7cxzKUju1AK

Exercise

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

- Prove that ReLU[α·z] = α·ReLU[z]
- Use this to prove that
  - $\text{ReLU}[\beta_1+(\lambda_1\cdot\Omega_1)\text{ReLU}[\beta_0+(\lambda_0\cdot\Omega_0)x]]=\lambda_0\lambda_1\cdot\text{ReLU}[(1/\lambda_0\lambda_1)\beta_1+\Omega_1\text{ReLU}[(1/\lambda_0)\beta_0+\Omega_0 x]]$

  where λ_0 and λ_1 are non-negative scalars.
- From this, we see that the weight matrices can be rescaled by any magnitude as long as the biases are also adjusted, and the scale factors can be re-applied at the end of the network.

Large, Structured Inputs

- We have discussed fully connected networks where every element of each layer contributes to every element of the subsequent one.
- However, these are not practical for large, structured inputs like images, where the input might comprise lots of pixels.
  - The number of parameters would be prohibitive, and moreover, we want different parts of the image to be processed similarly
    - E.g., there is no point in independently learning to recognize the same object at every possible position in the image.
  - The solution is to process local image regions in parallel and then gradually integrate information from increasingly large regions.

Training, and Generalization

- A further possible advantage of deep networks over shallow networks is their ease of
- Fitting:
  - It is usually easier to train moderately deep networks than to train shallow ones
  - It may be that over-parameterized deep models have a large family of roughly equivalent solutions that are easy to find.
  - However, as we add more hidden layers, training becomes more difficult again, although many methods have been developed to mitigate this problem.
- Deep neural networks also seem to generalize to new data better than shallow ones.
- In practice, the best results for most tasks have been achieved using networks with tens or hundreds of layers.
- Anyway, neither of these phenomena are well understood.

Universal Approximation Theorem: Depth Version

- There exists a network that can approximate any given continuous function on a compact subset of $\mathbb{R}^{D_i}$ to arbitrary accuracy.
- There exists a network with ReLU activation functions and at least $D_i + 4$ hidden units in each layer can approximate any specified D_i-dimensional Lebesgue integrable function to arbitrary accuracy given enough layers.
- This is known as the depth version of the universal approximation theorem.
    - Lu, Zhou; Pu, Hongming; Wang, Feicheng; Hu, Zhiqiang; Wang, Liwei (2017). "The Expressive Power of Neural Networks: A View from the Width". Advances in Neural Information Processing Systems. Curran Associates. 30: 6231–6239. arXiv:1709.02540

Deep Learning

End of Lecture

02 - Shallow And Deep Networks

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri