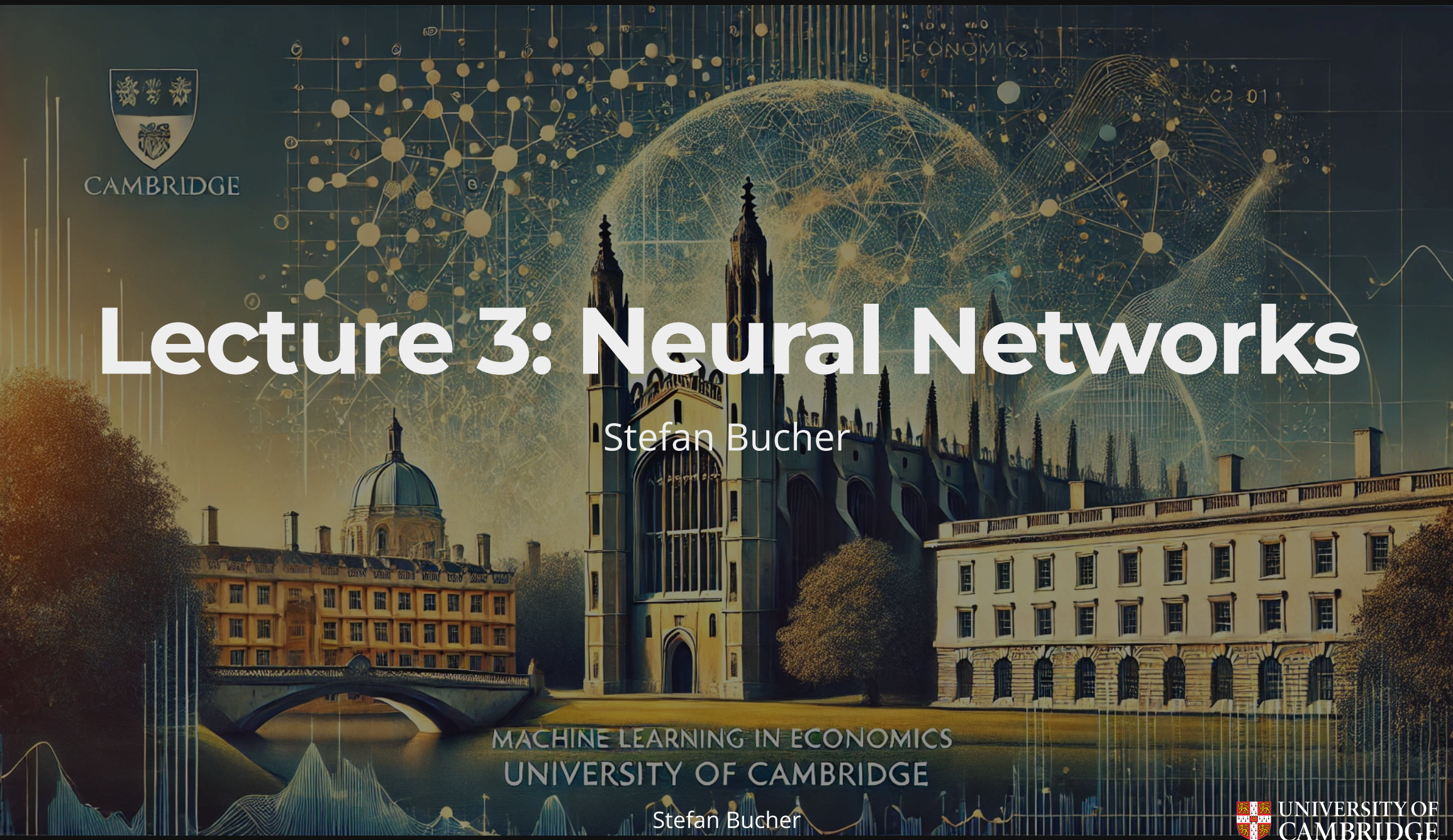


Lecture 3: Neural Networks

Stefan Bucher

Open in Colab

Prince (2023, chaps. 3, 4, 7, 10).[1]

1. Figures taken or adapted from Prince (2023). All rights belong to the original author and publisher. These materials are intended solely for educational purposes.

Shallow Neural Networks

Prince (2023, chap. 3)

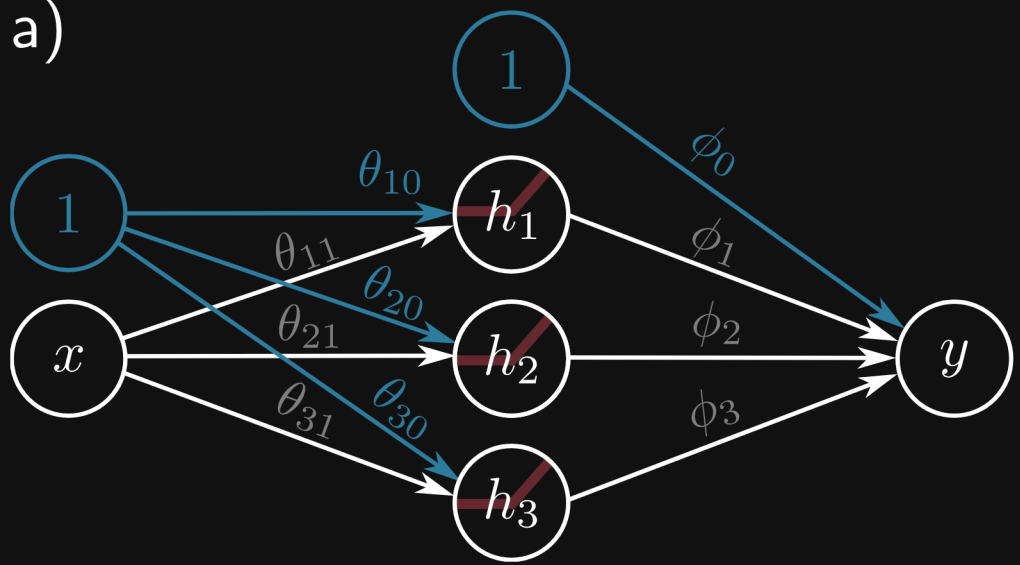
Example

$$f[x,\phi] = \phi_0 + \phi_1 \underbrace{a(\theta_{10} + \theta_{11}x)}_{h_1} + \phi_2 \underbrace{a(\theta_{20} + \theta_{21}x)}_{h_2} + \phi_3 \underbrace{a(\theta_{30} + \theta_{31}x)}_{h_3}$$

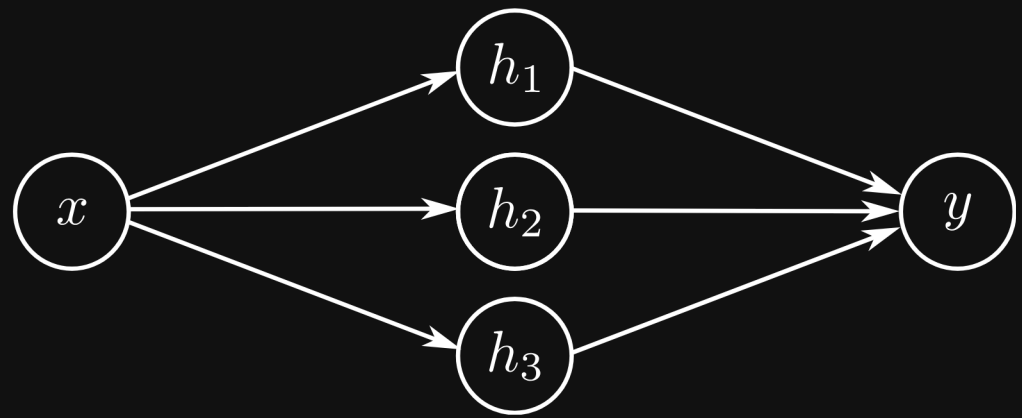
$$a(z) = ReLU(z) = \max\{z, 0\}$$

Neural networks (shallow & deep!) with ReLU represent continuous piecewise linear functions.

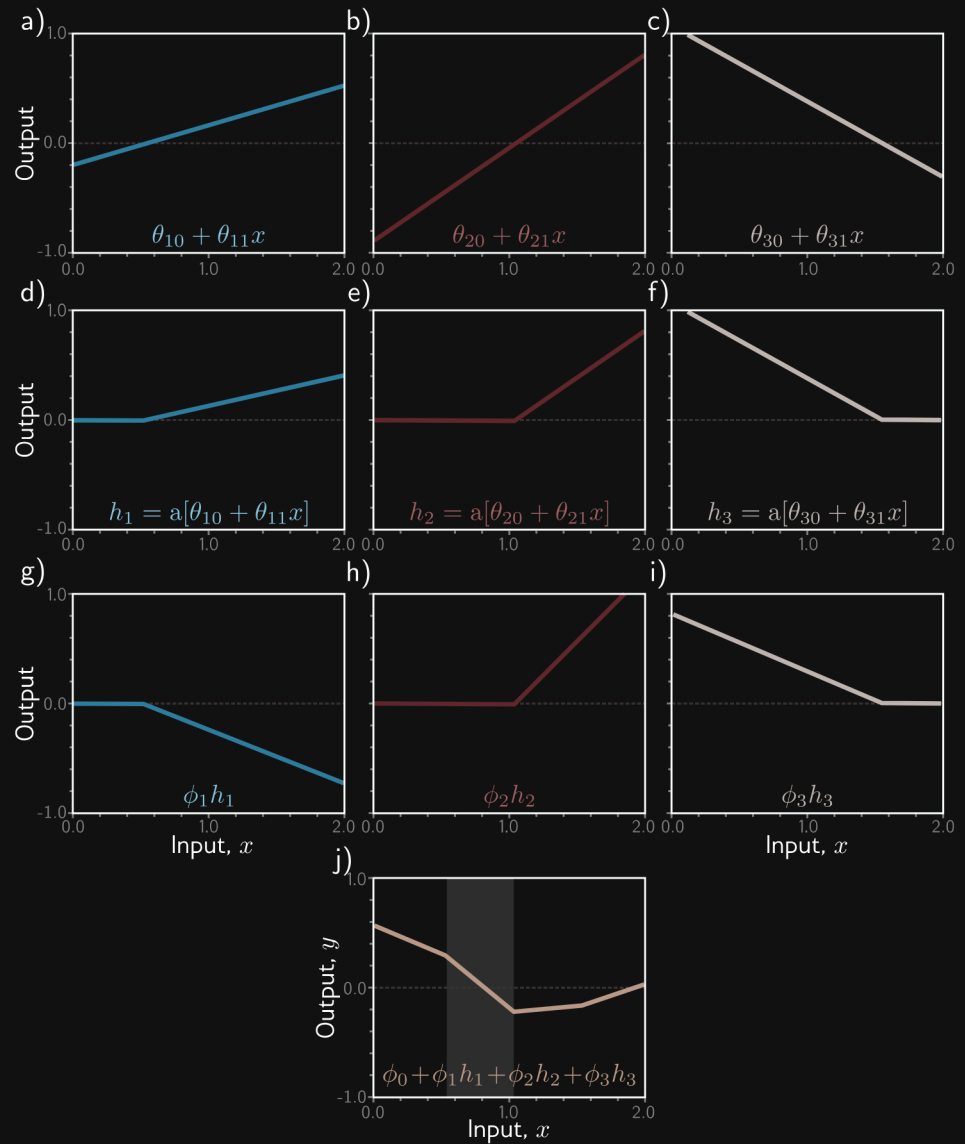
a)



b)

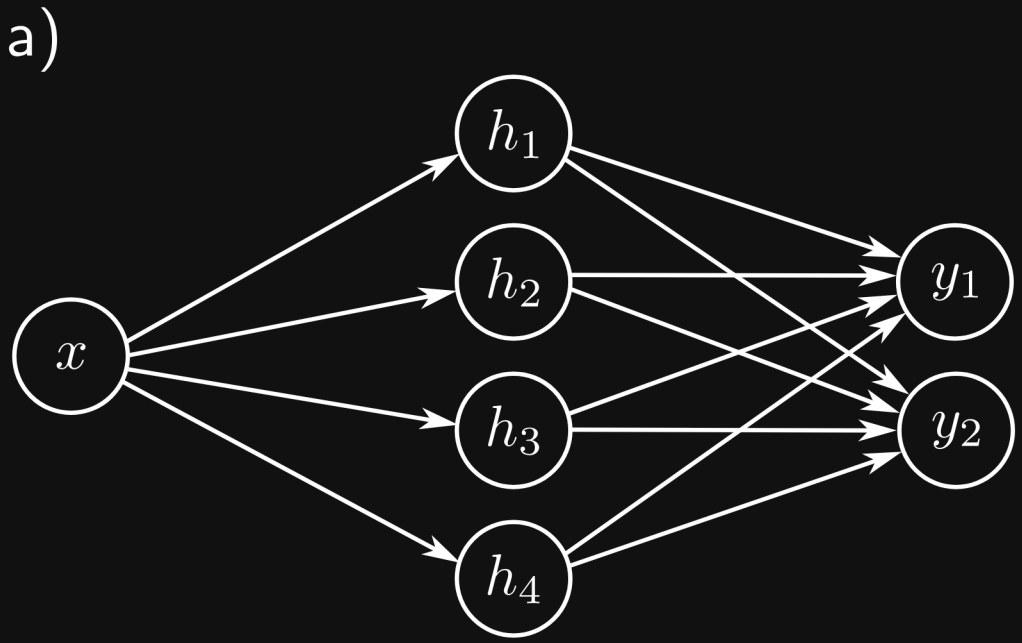


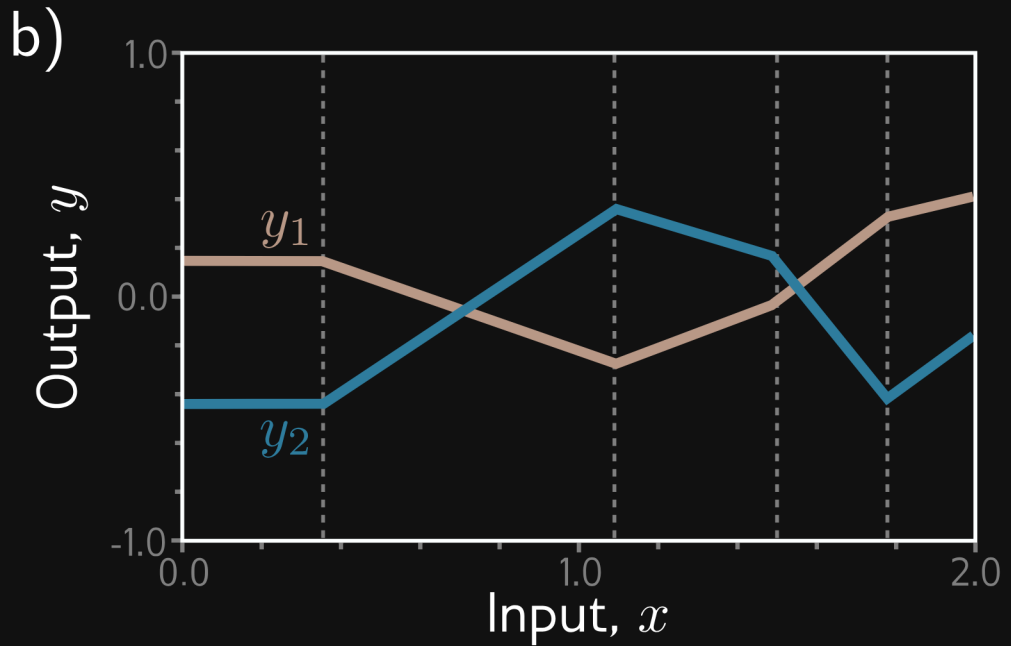
Activation Patterns



Joints where one hidden unit becomes (in)active.

Multivariate Output

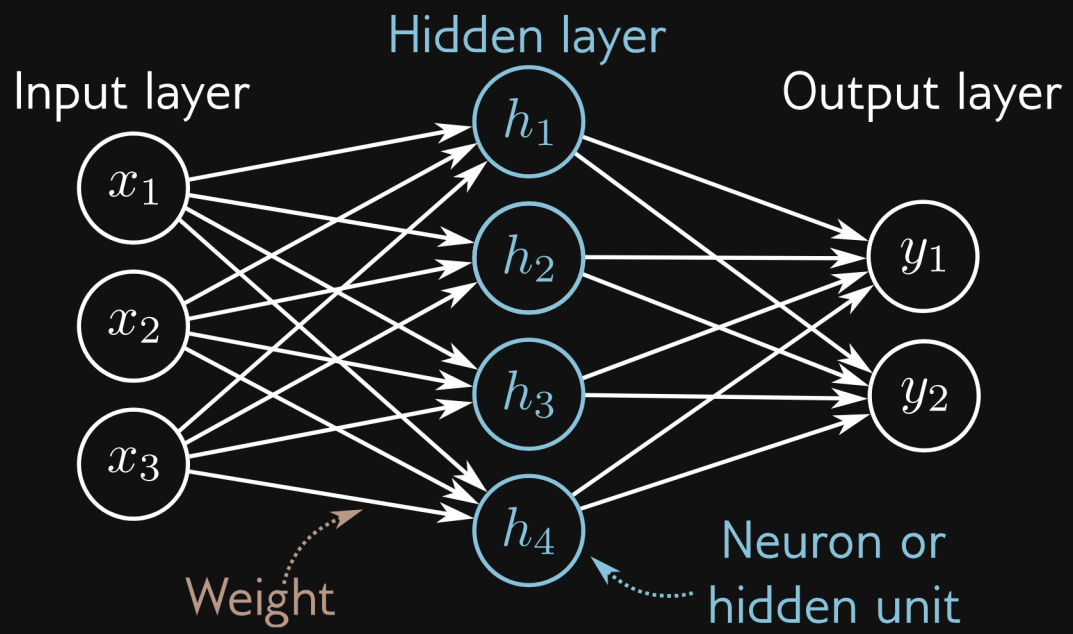




Shallow Neural Networks

Map n_{in}-dimensional input to n_{out}-dimensional output.

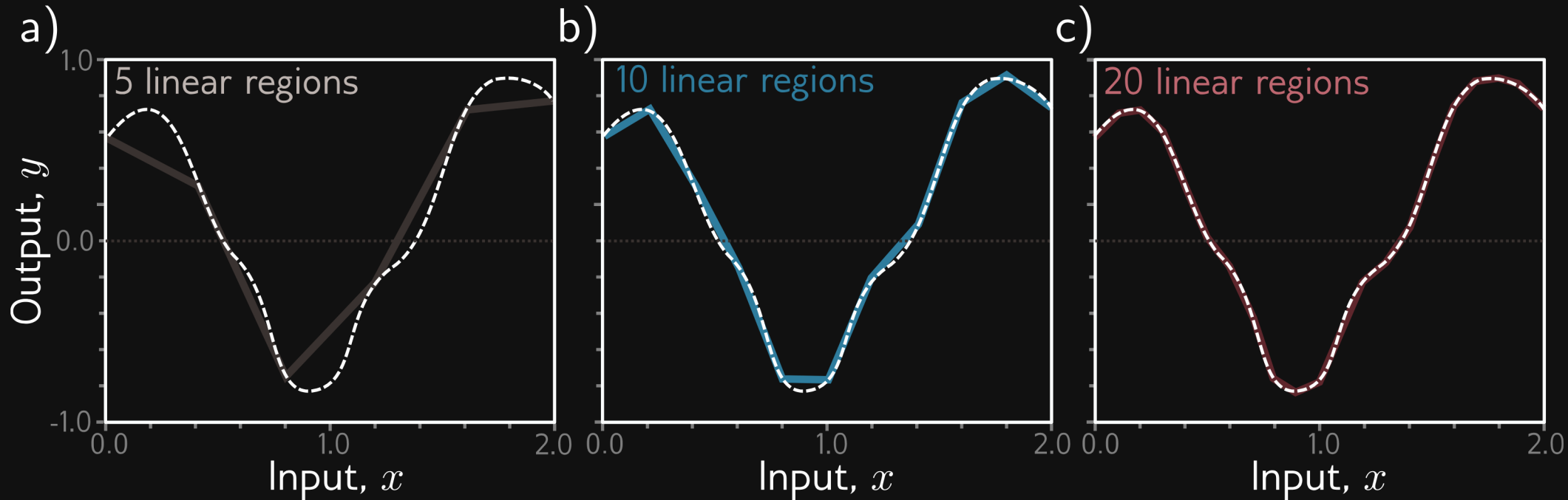
$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} \underbrace{a(\theta_{d0} + \sum_{i=1}^{n_{in}} \theta_{di} x_i)}_{h_d}, \; j = 1, \ldots, n_{out}$$



Shallow Networks (3.3 & 3.8)

Universal Appproximation Thm

$\exists$ shallow neural network with a single layer of sufficiently many hidden units that can approximate, with arbitrary precision, any continuous function from a compact subset of $R^{n_{in}}$ to $R^{n_{out}}$.

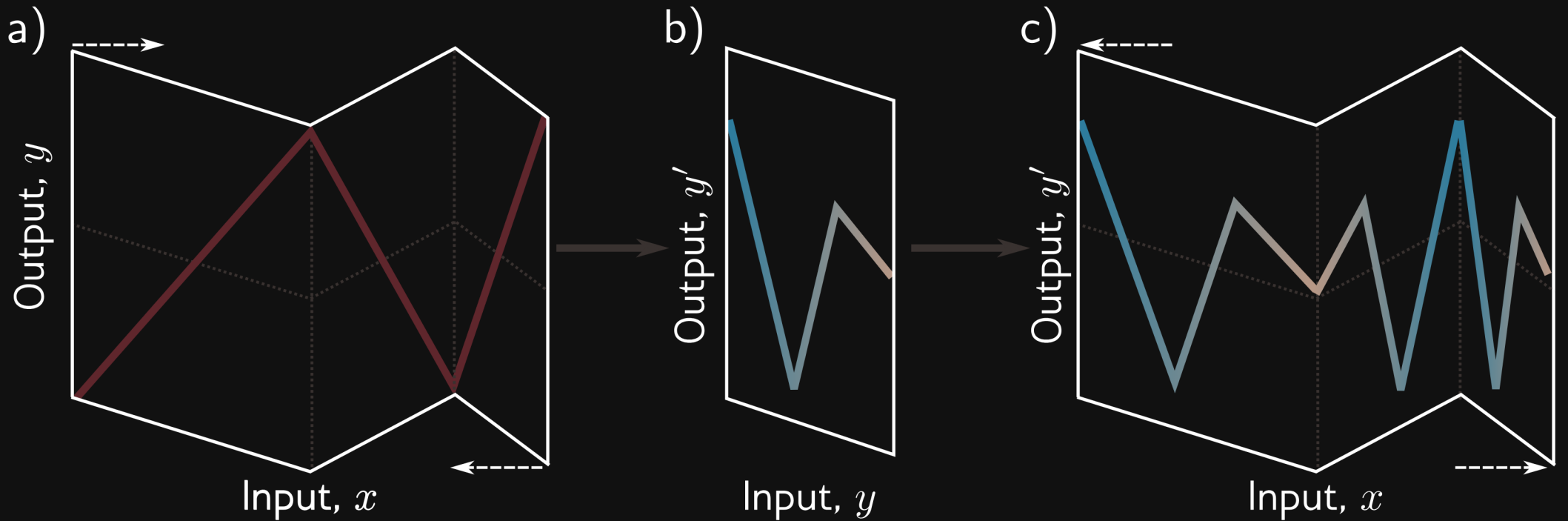


Deep Neural Networks

Prince (2023, chap. 4)

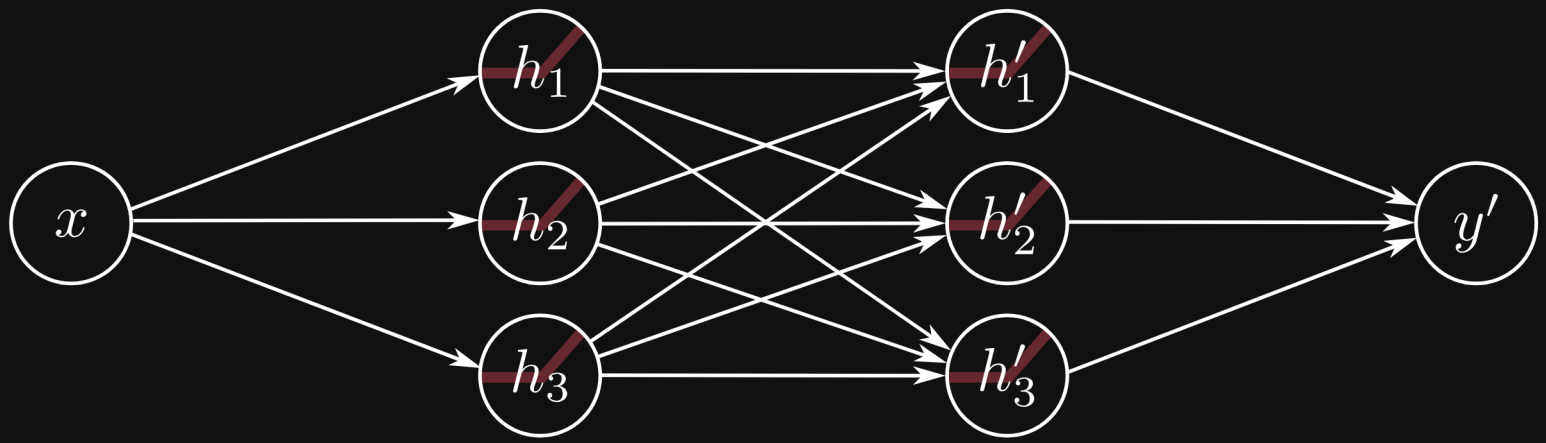
Why Depth? (4.1)

Extra Layer is Folding

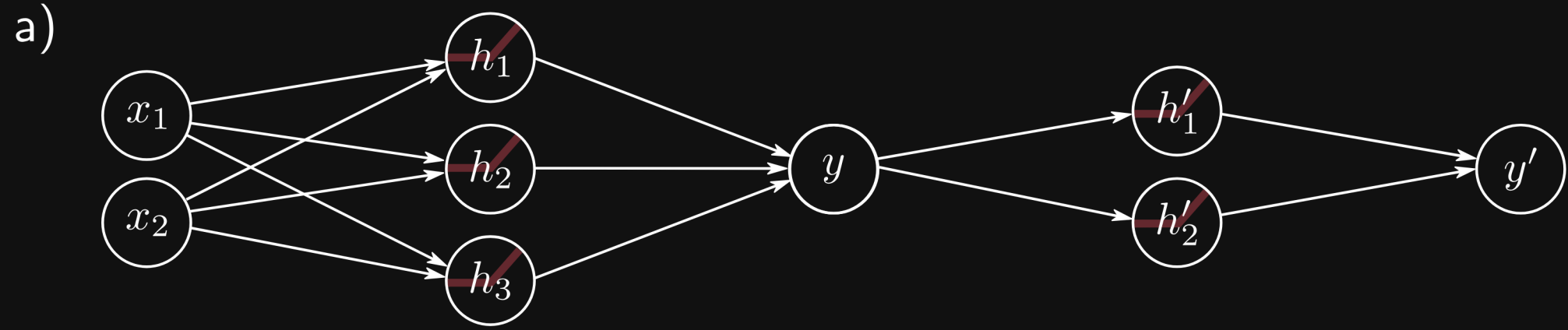


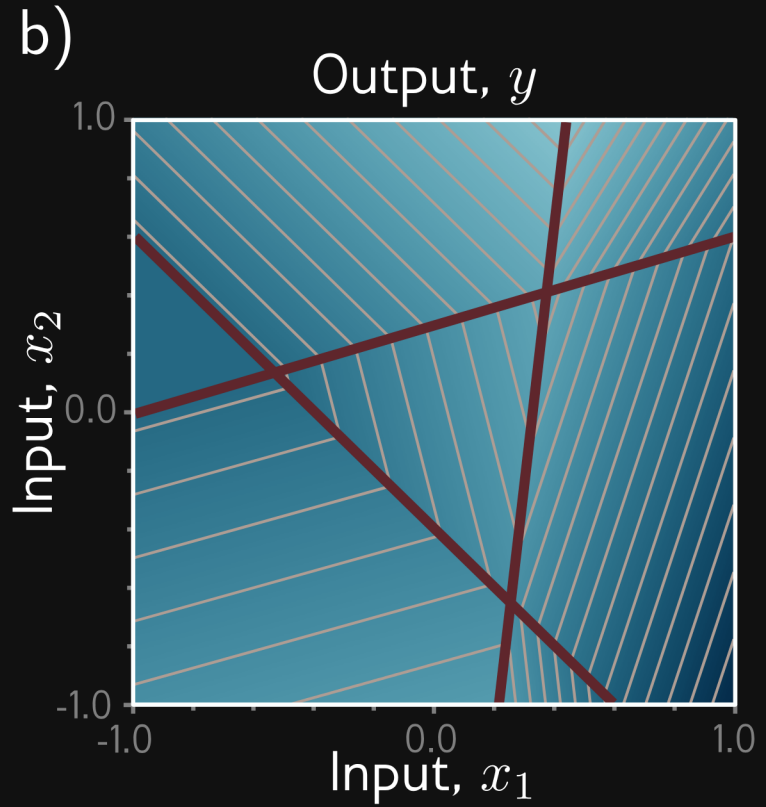
Extra Layer is Clipping (4.5)

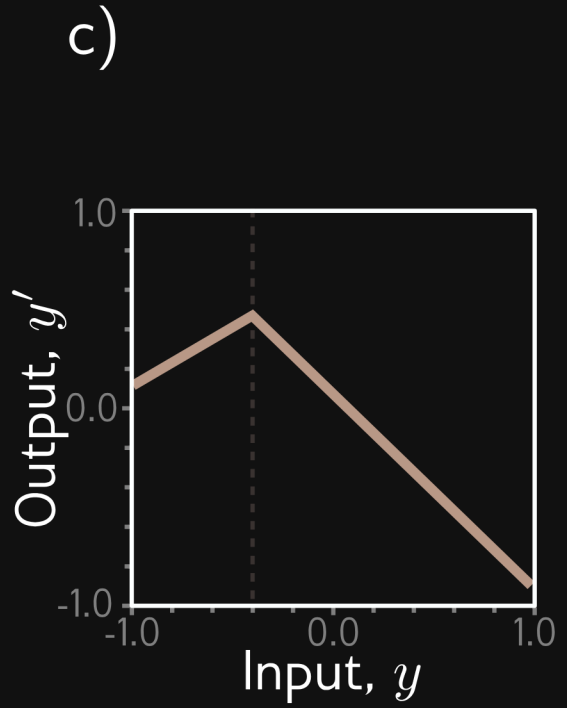
Composing as Special Case

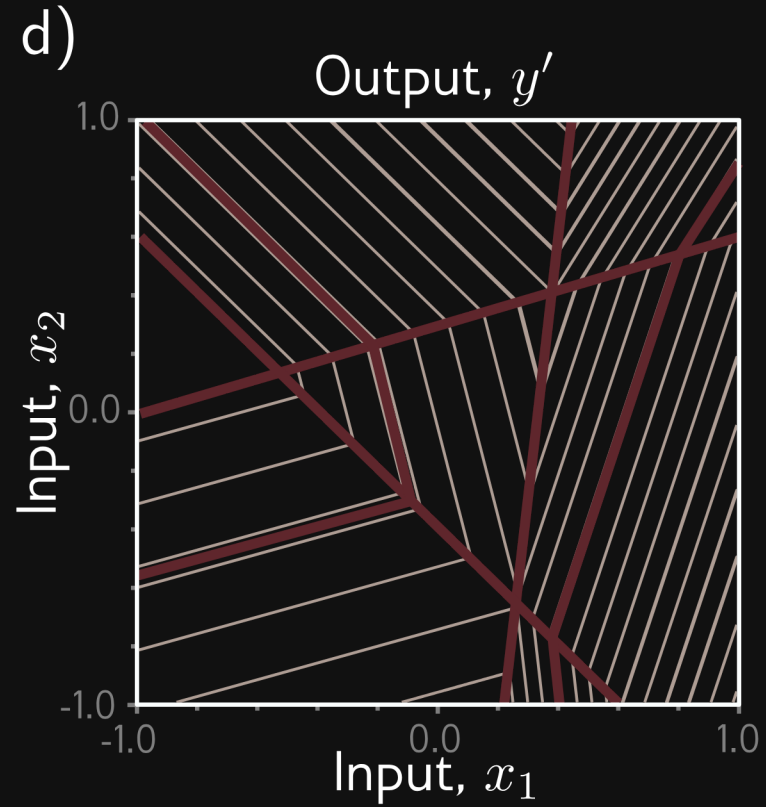


Composing in 2D



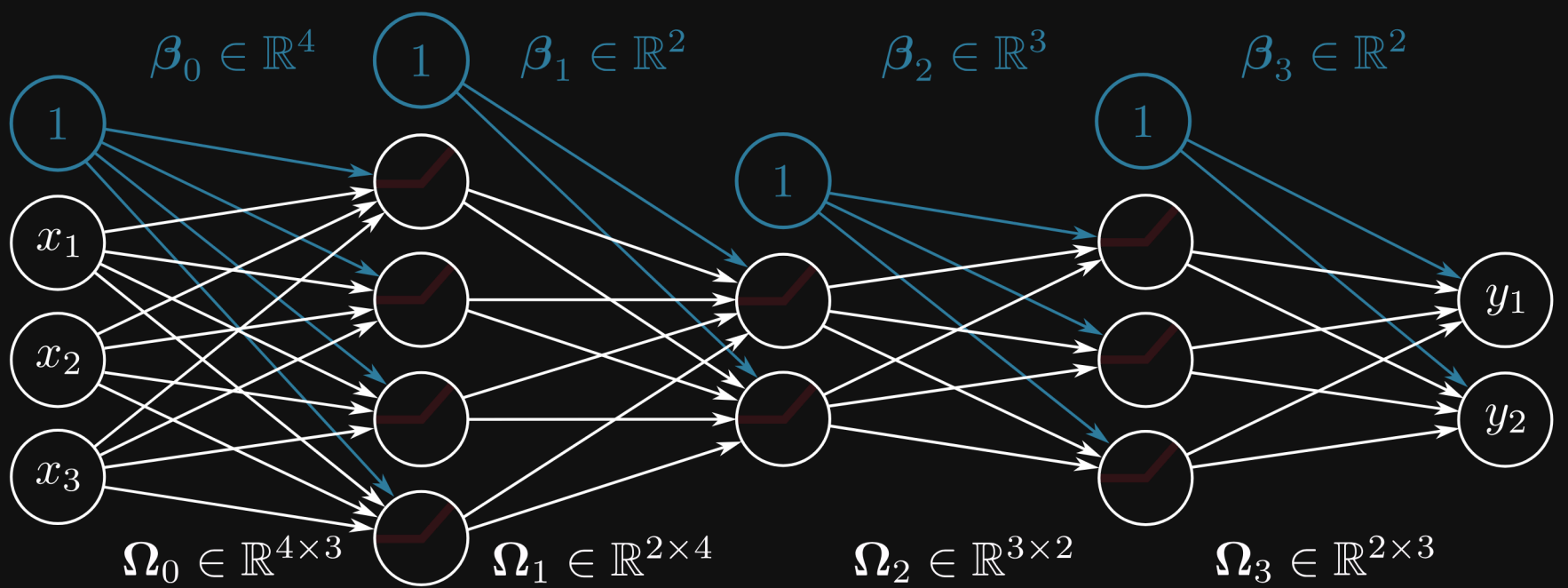






Deep Neural Network

$$\mathbf{h}_1 = \mathbf{a}(\beta_0 + \mathbf{\Omega}_0 \mathbf{x})$$
$$\mathbf{h}_k = \mathbf{a}(\beta_{k-1} + \mathbf{\Omega}_{k-1} \mathbf{h}_{k-1}), \quad k = 2, \ldots, K$$
$$\mathbf{y} = \beta_K + \mathbf{\Omega}_K \mathbf{h}_K$$



Input, $\mathbf{x}$ — $D_i = 3$

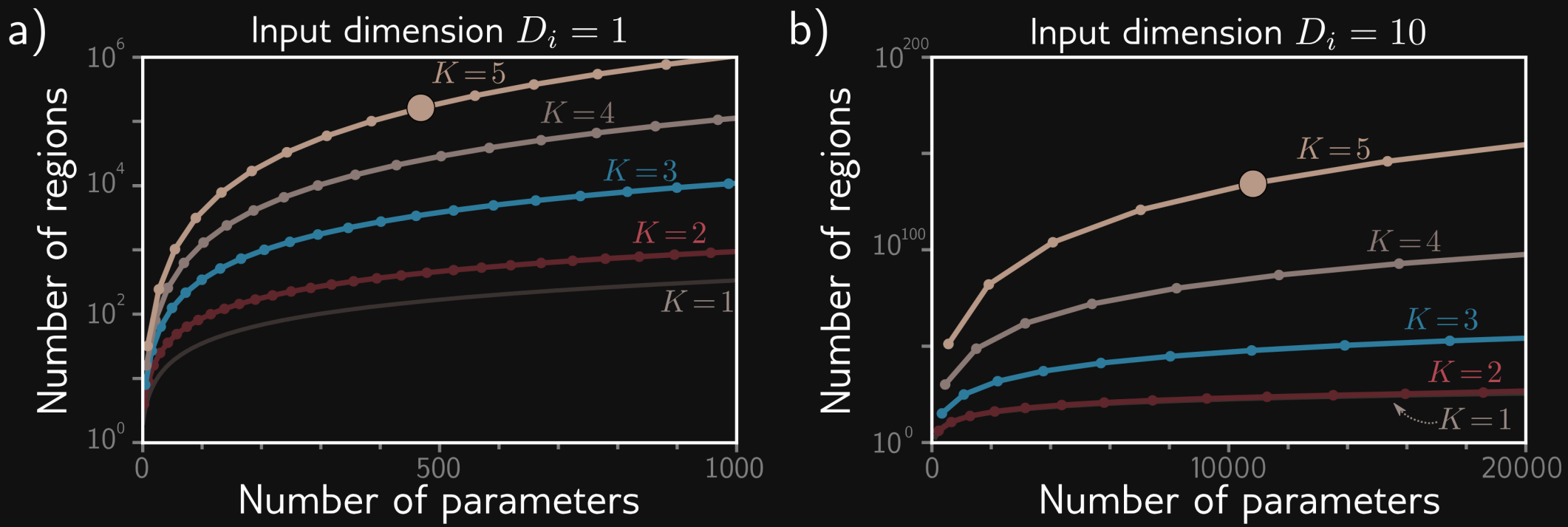
Hidden layer, $\mathbf{h}_1$ — $D_1 = 4$

Hidden layer, $\mathbf{h}_2$ — $D_2 = 2$

Hidden layer, $\mathbf{h}_3$ — $D_3 = 3$

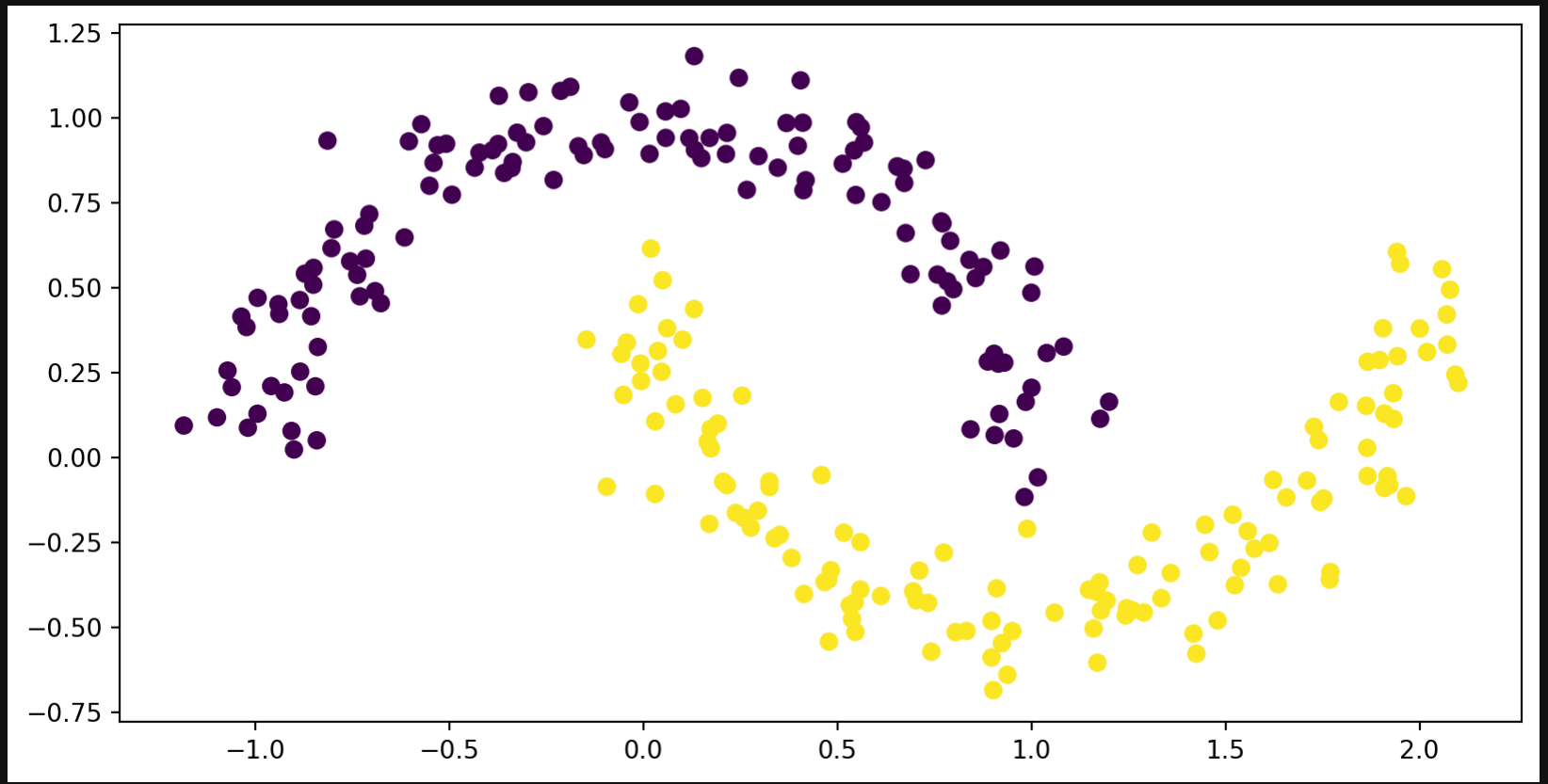
Output, $\mathbf{y}$ — $D_o = 2$

Depth Efficiency



PyTorch Implementation

Moons Dataset



Neural Network Implementation

```python
import torch
import torch.nn as nn

class MyFirstNet(nn.Module):
  def __init__(self):
    super(MyFirstNet, self).__init__()
    self.layers = nn.Sequential(
        nn.Linear(2, 16),
        nn.ReLU(),
        nn.Linear(16, 2),
    )

  def forward(self, x):
     return self.layers(x)

  def predict(self, x):
    output = self.forward(x)
    return torch.argmax(output, 1)

  def train(self, X, y):
    loss_function = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)

    epochs = 15000
```

```
MyFirstNet(
  (layers): Sequential(
    (0): Linear(in_features=2, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=2, bias=True)
  )
```

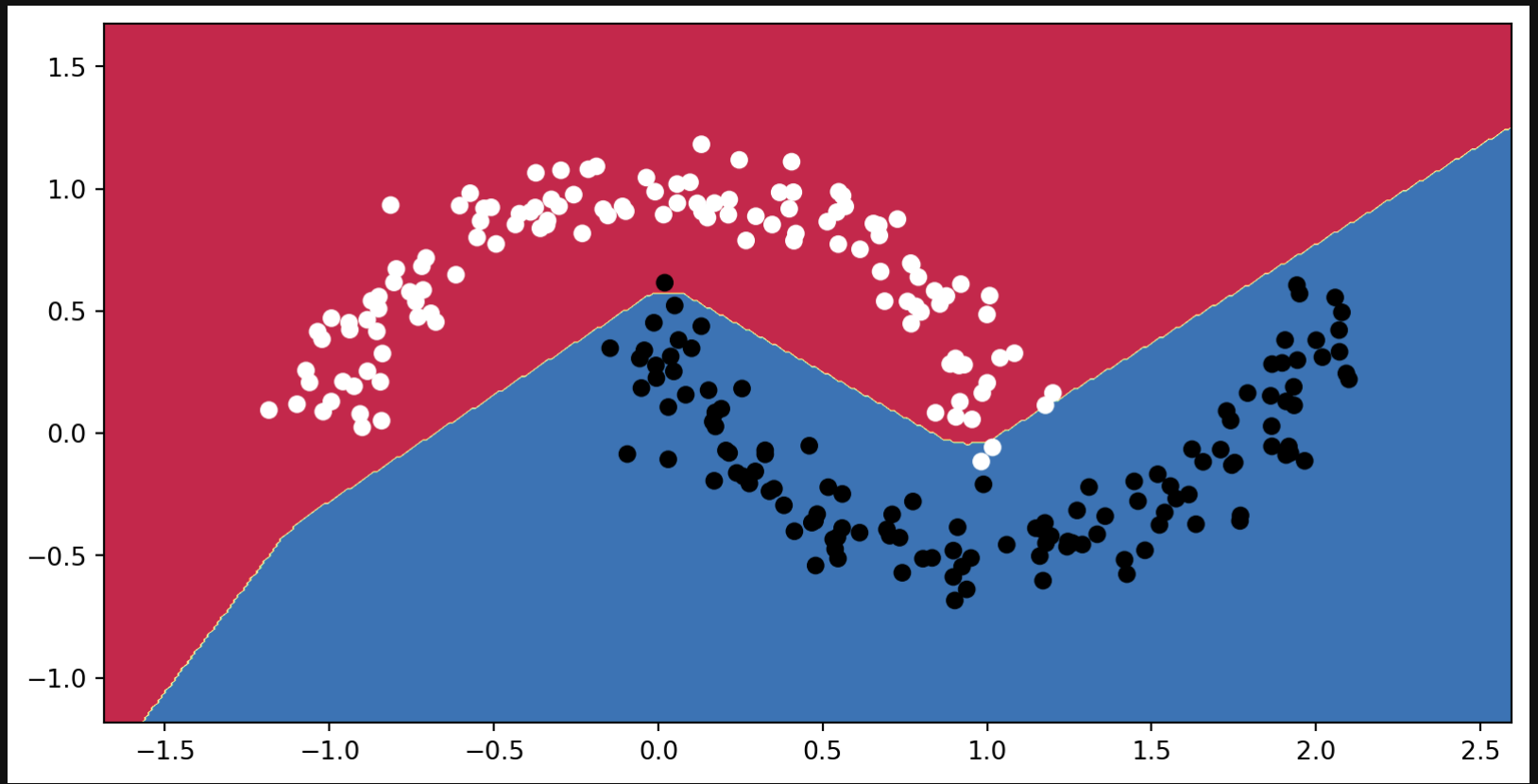
```
)
Sample input:
 tensor([[-0.8137,  0.9335],
        [ 0.5321, -0.4338],
        [ 0.2155, -0.0803],
        [ 1.9061,  0.3816],
        [ 0.6543,  0.8580]], device='mps:0')

Network output:
 tensor([[-0.1091, -0.4467],
        [-0.2461,  0.0059],
        [-0.2210, -0.0897],
```

Training

```
1  losses = model.train(X, y)
```

```
Epoch 0 loss is 0.6180233955383301
Epoch 1000 loss is 0.2884750962257385
Epoch 2000 loss is 0.2551516890525818
Epoch 3000 loss is 0.2375335544347763
Epoch 4000 loss is 0.22057485580444336
Epoch 5000 loss is 0.20335149765014648
Epoch 6000 loss is 0.1836058646440506
Epoch 7000 loss is 0.1620502769947052
Epoch 8000 loss is 0.1405417025089264
Epoch 9000 loss is 0.12077677249908447
Epoch 10000 loss is 0.103335440158844
Epoch 11000 loss is 0.08853869885206223
Epoch 12000 loss is 0.07637479156255722
Epoch 13000 loss is 0.06647869944572449
Epoch 14000 loss is 0.058444686233997345
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Backpropagation

Prince (2023, chap. 7)

Convolutional Neural Networks (CNN)

Prince (2023, chap. 10)