Fundamentals / Foundation of Data Science

Data Science 9CFU

Computer Science 6CFU

Indro Spinelli

Sapienza University of Rome

Composing two networks

Network 1:

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

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

Network 2:

$$h'_1 = \mathrm{a}[\theta'_{10} + \theta'_{11}y]$$
$$h'_2 = \mathrm{a}[\theta'_{20} + \theta'_{21}y]$$
$$h'_3 = \mathrm{a}[\theta'_{30} + \theta'_{31}y]$$

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

Composing two networks

Network 1:

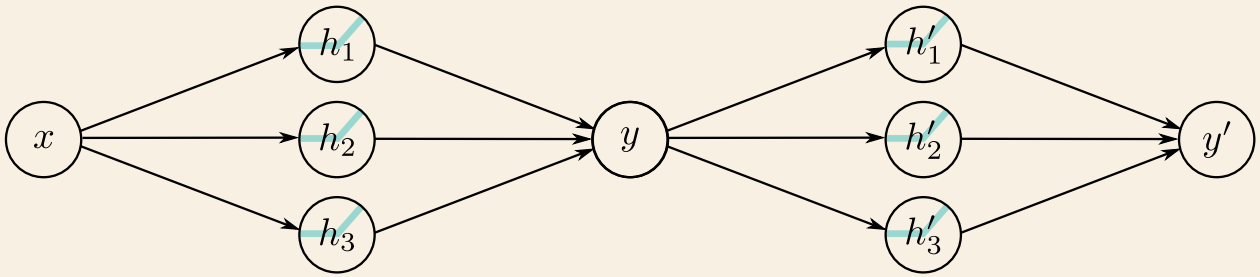
$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

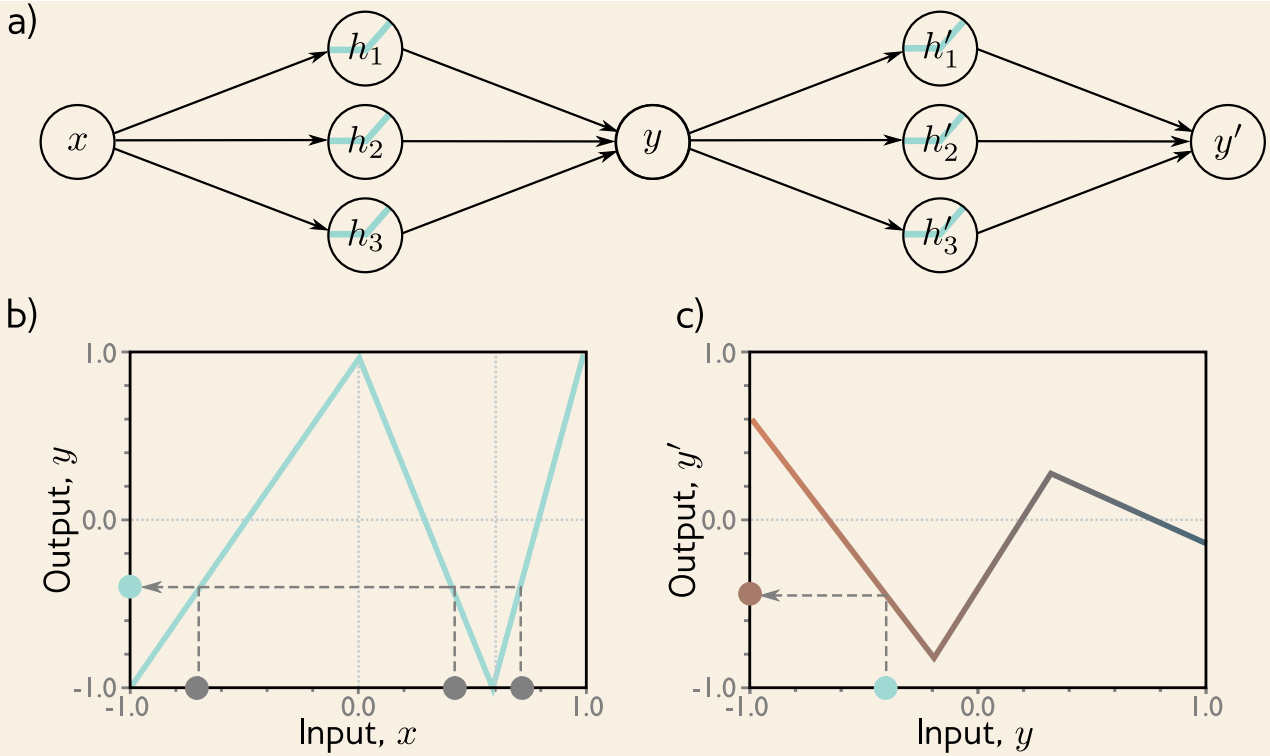
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

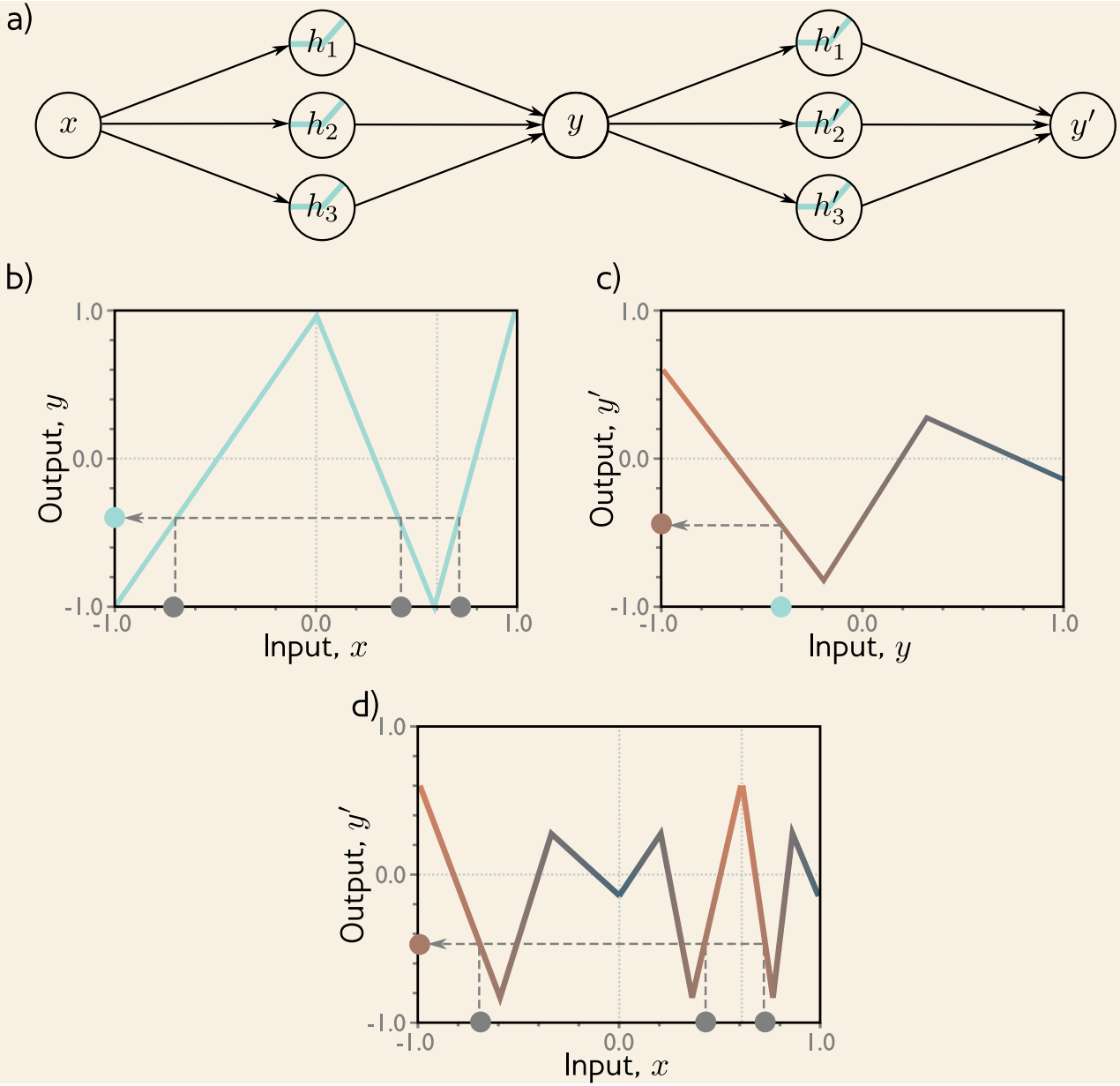
Network 2:

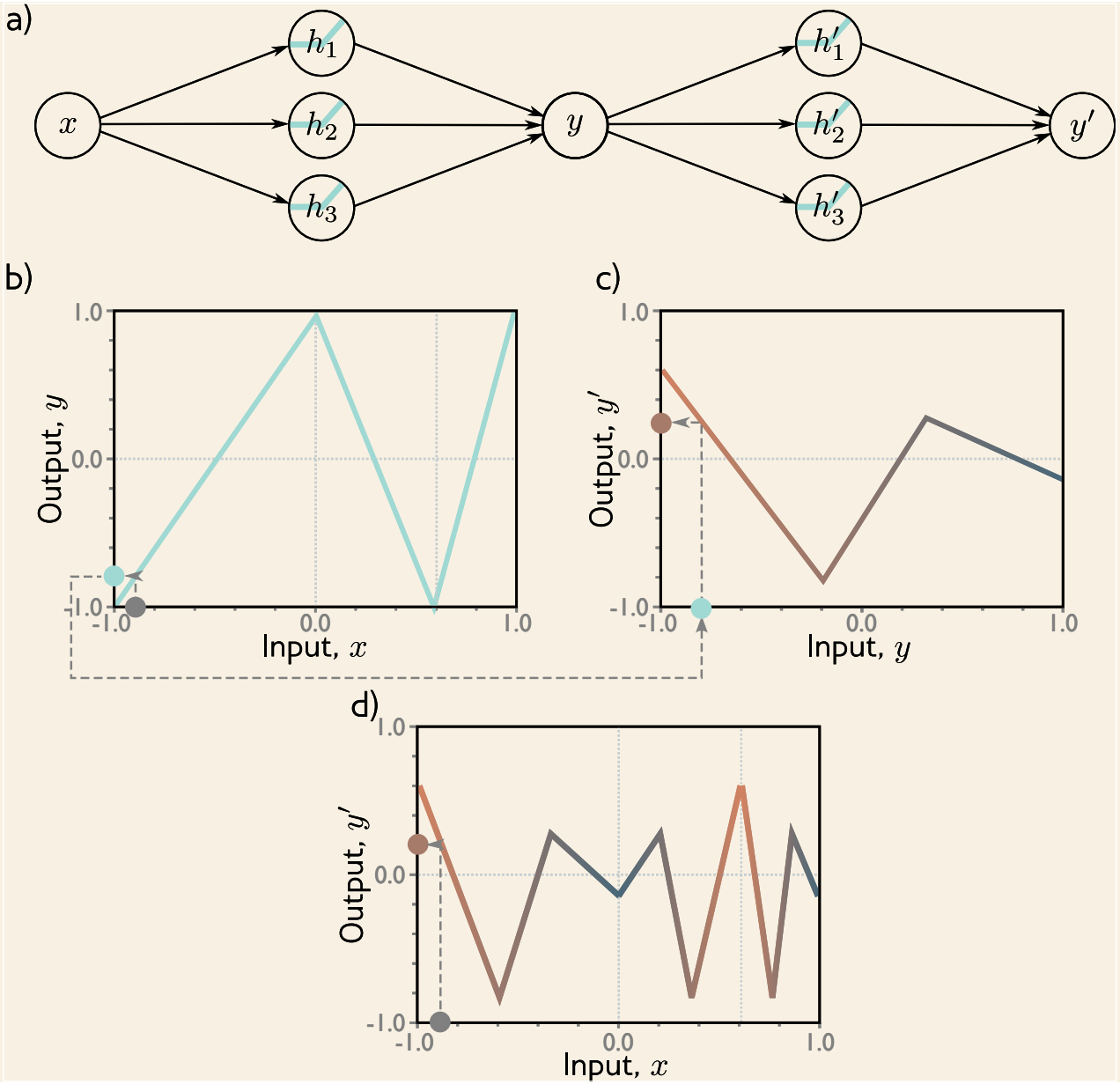
$$h'_1 = \mathrm{a}[\theta'_{10} + \theta'_{11}y]$$
$$h'_2 = \mathrm{a}[\theta'_{20} + \theta'_{21}y]$$
$$h'_3 = \mathrm{a}[\theta'_{30} + \theta'_{31}y]$$

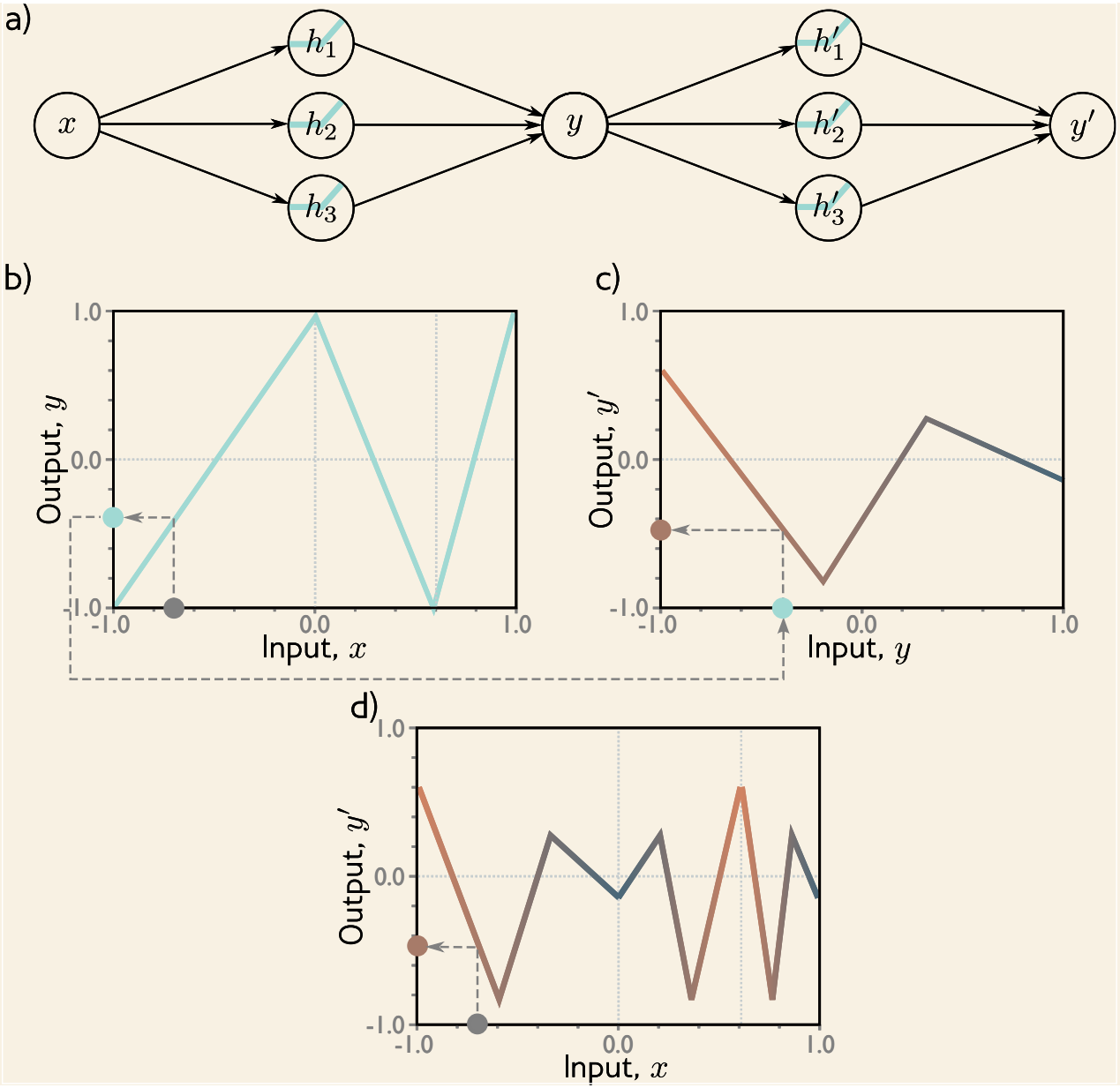
$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$

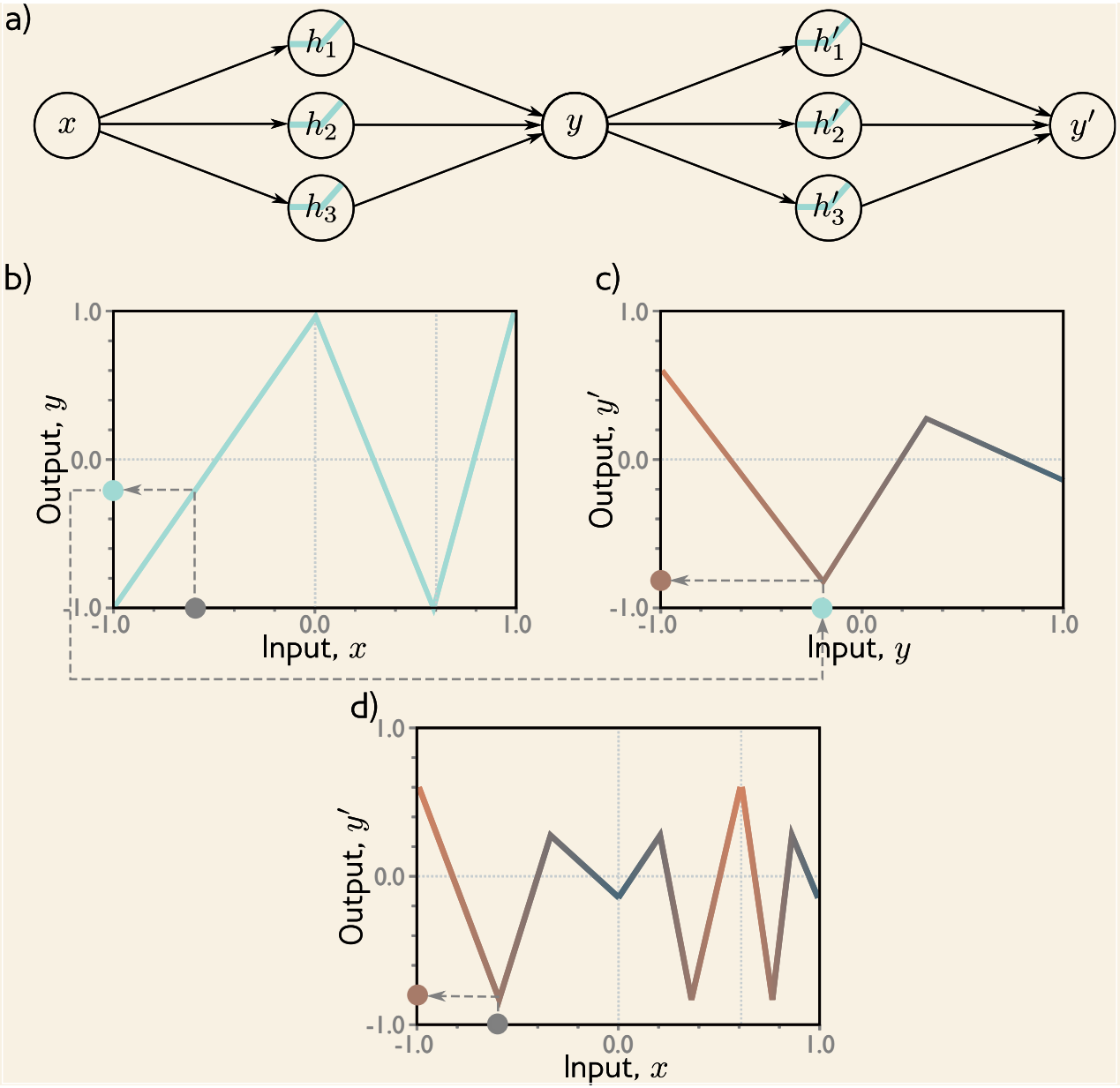


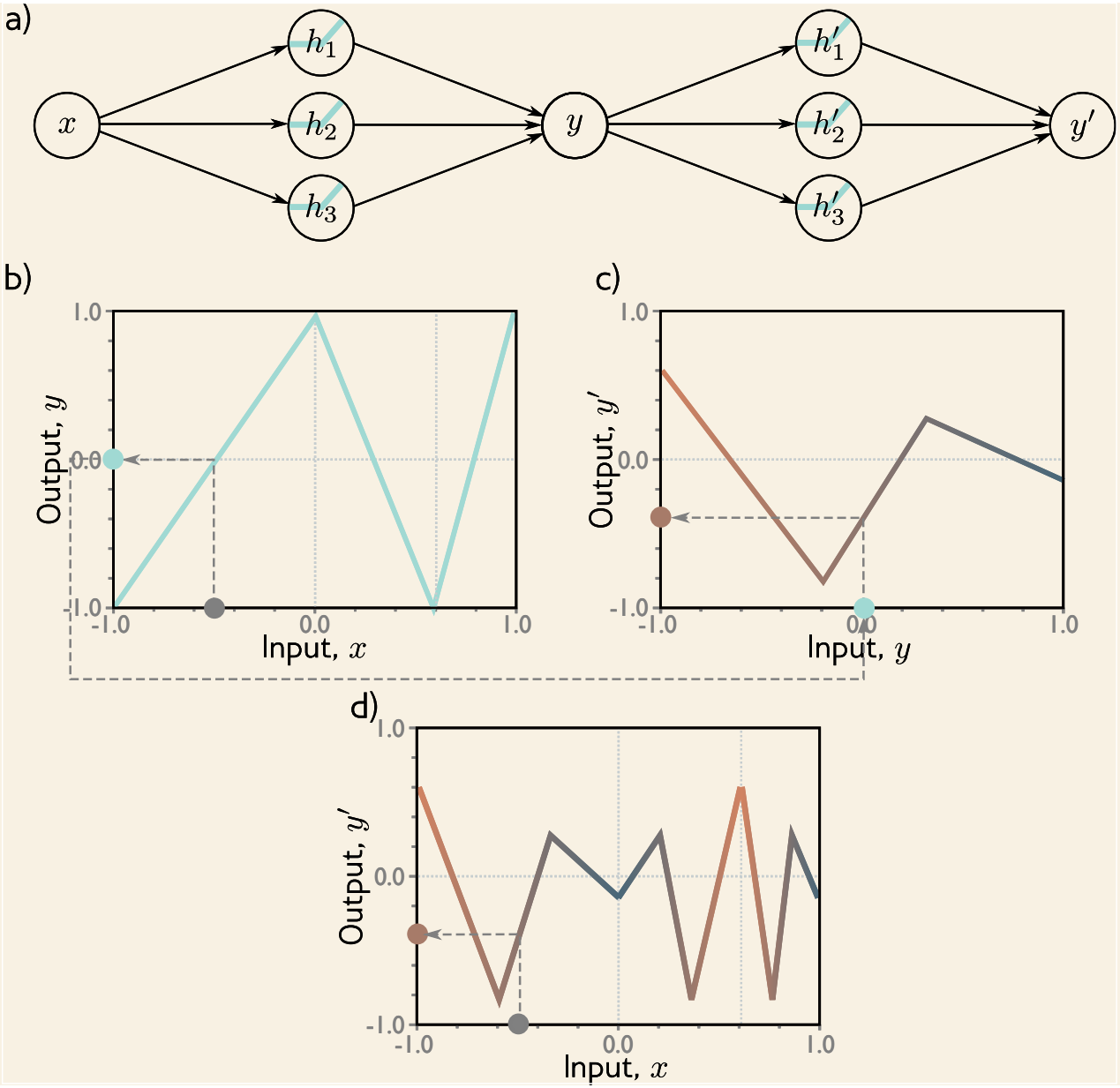

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
1.0
0.0
-1.0
Output, y
-1.0
0.0
1.0
Input, x
c)
1.0
0.0
-1.0
Output, y'
-1.0
0.0
1.0
Input, y

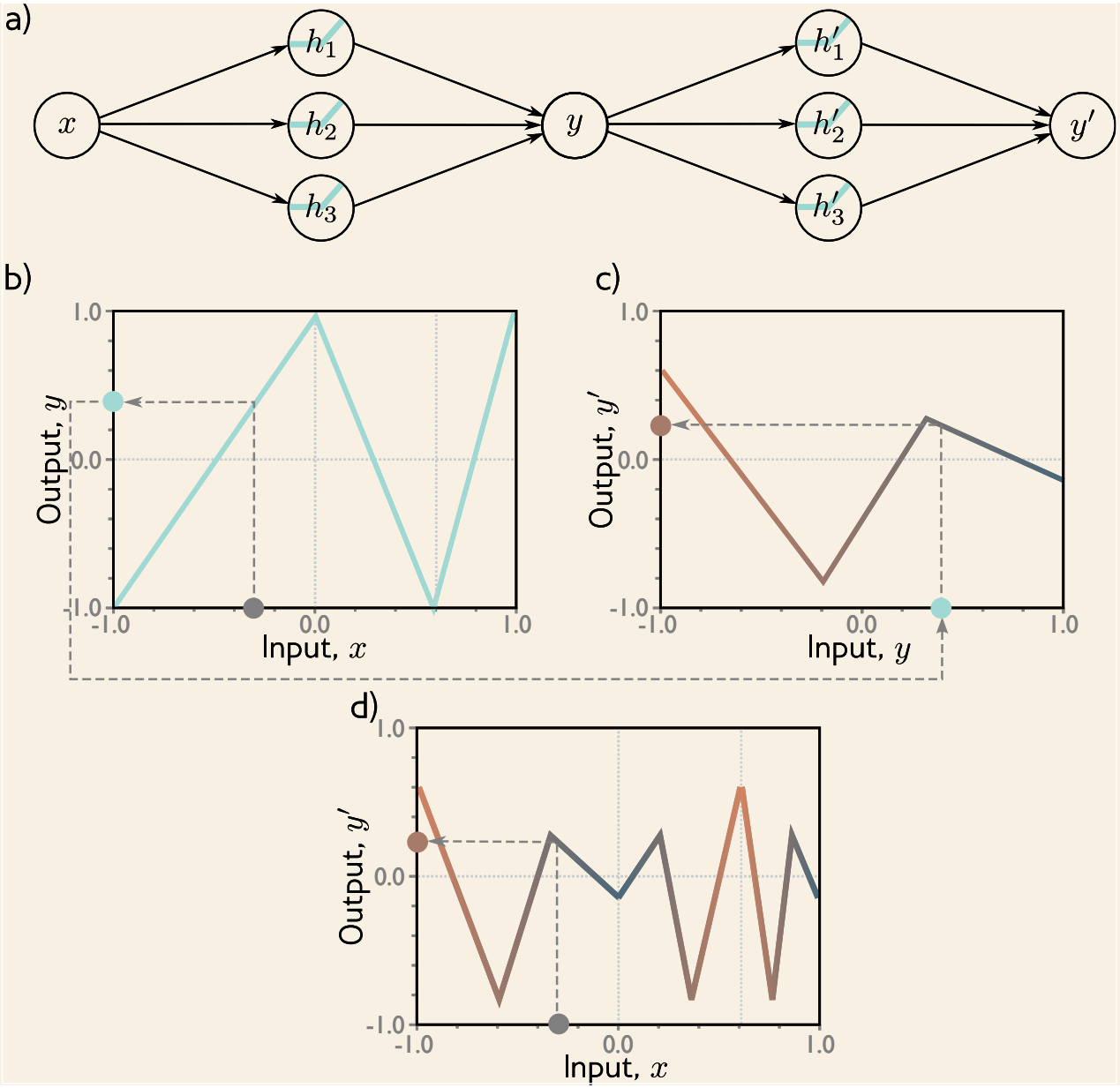

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

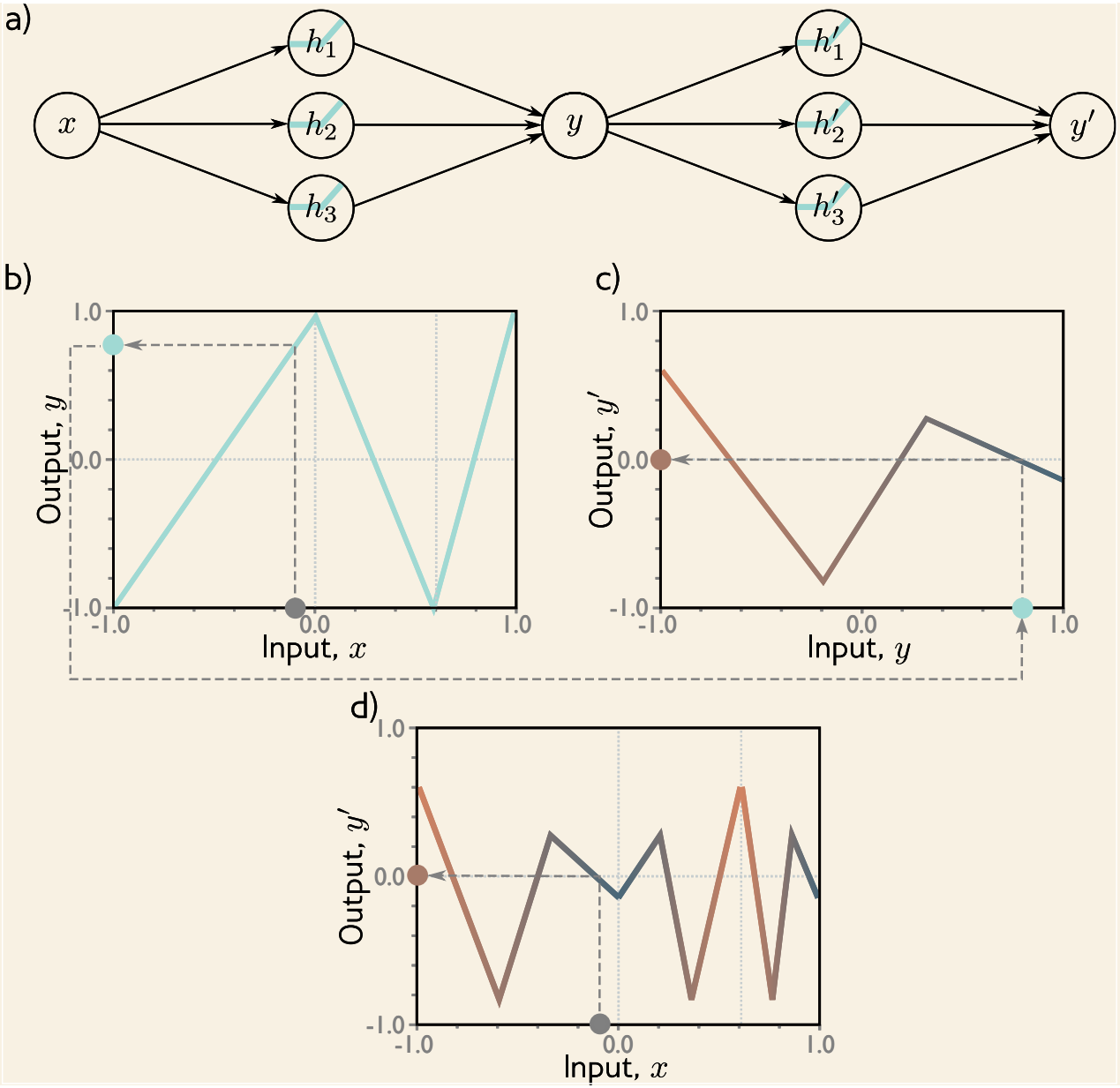

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

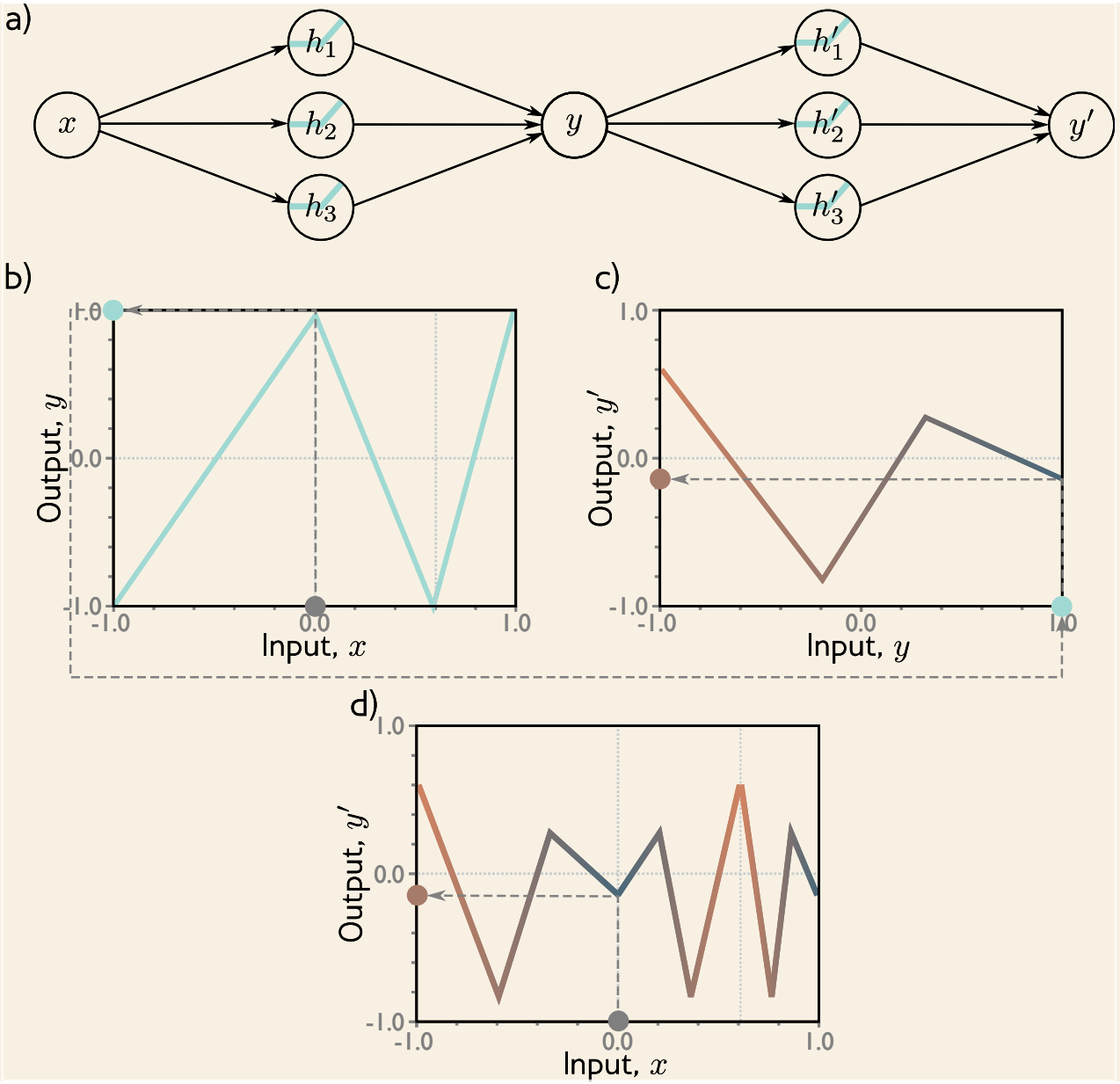

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

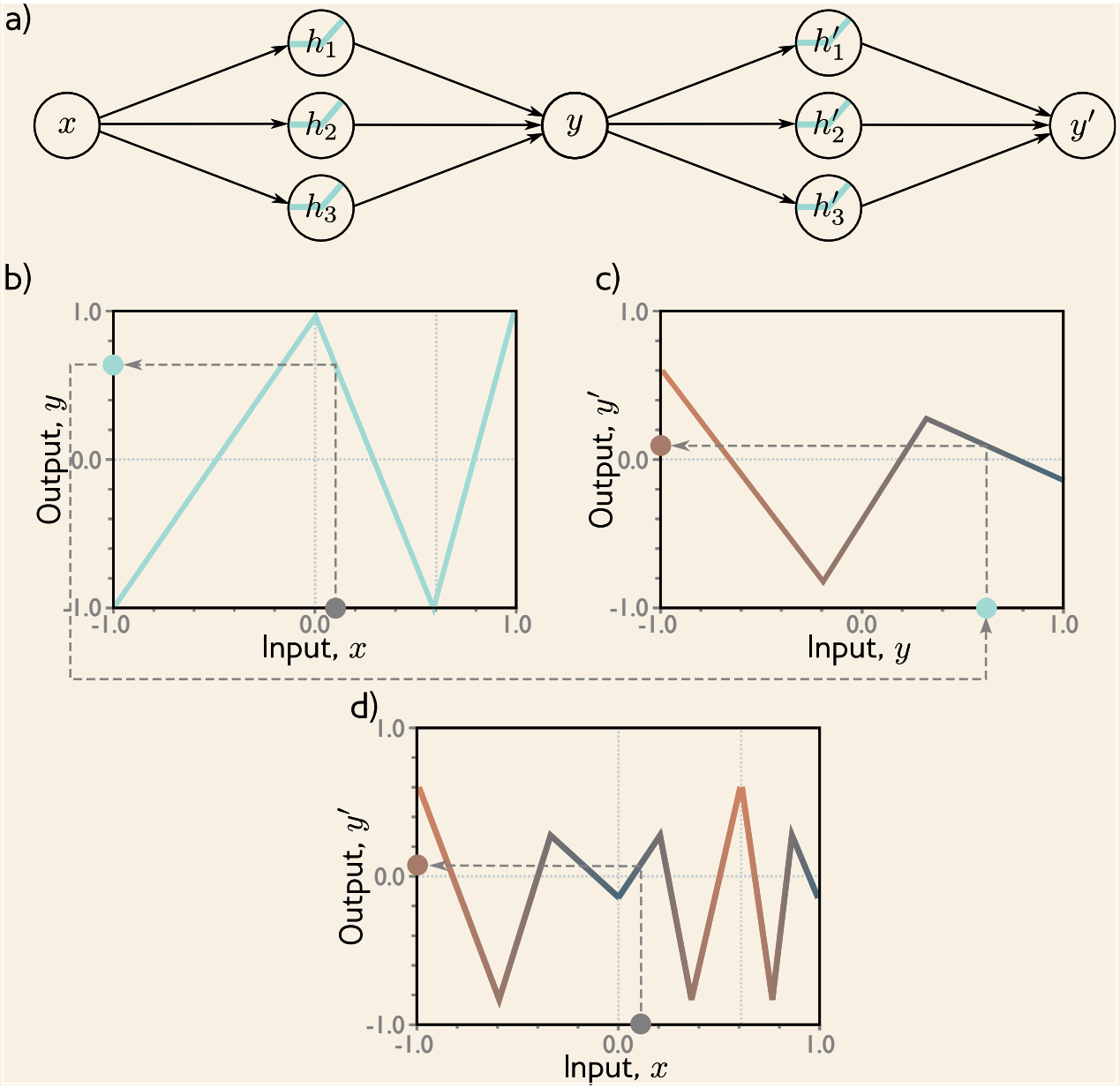

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

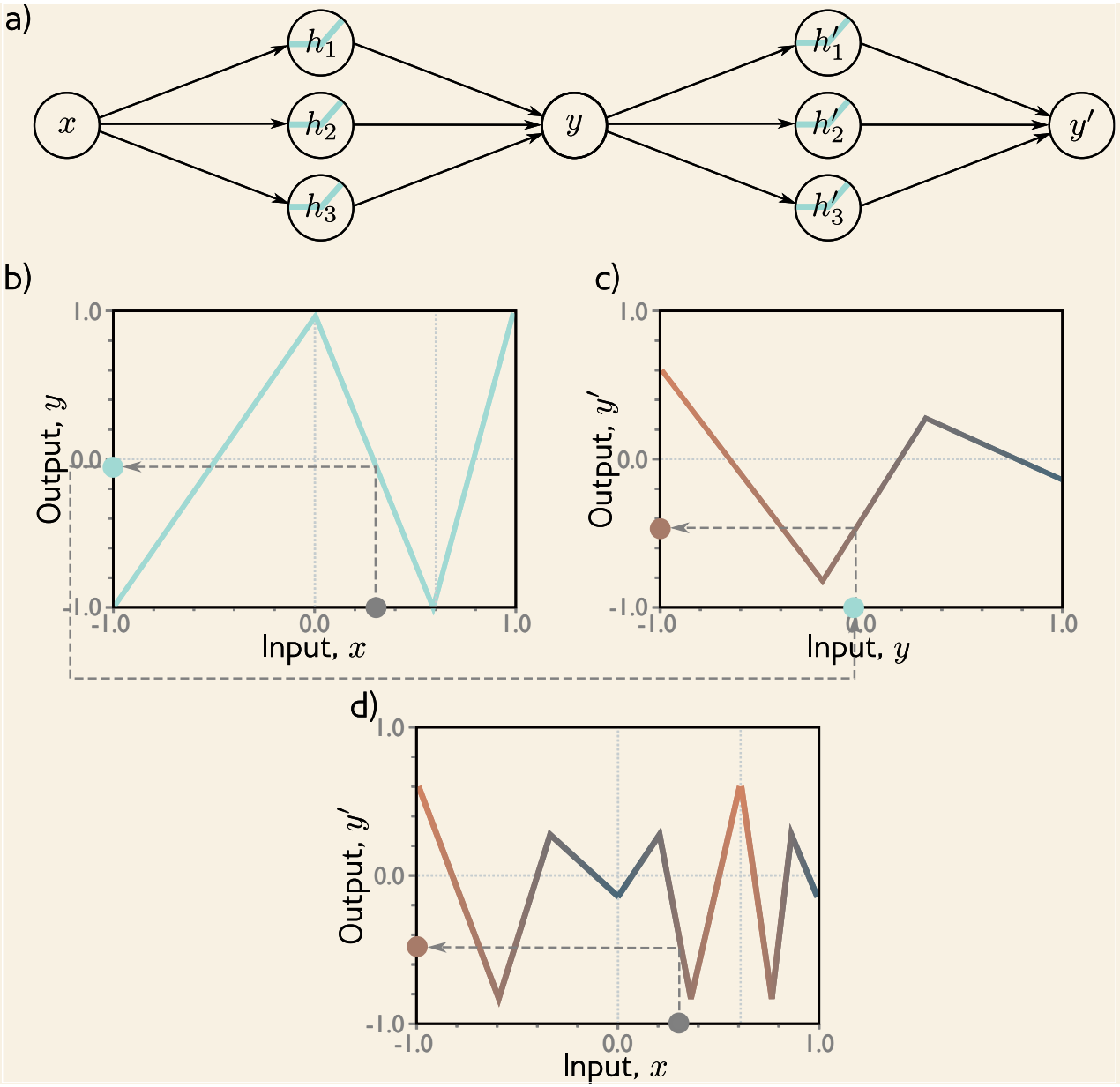

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

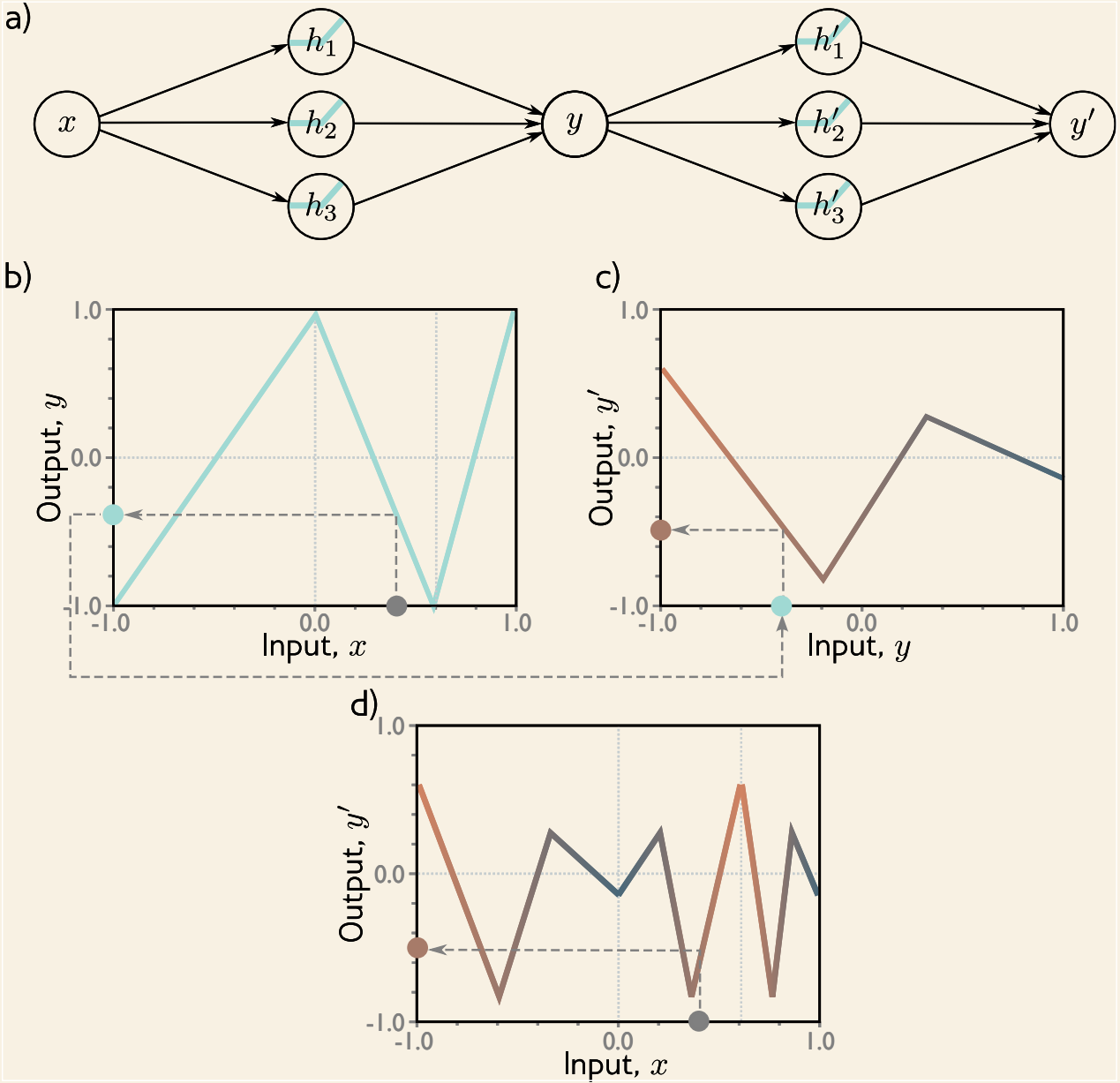

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

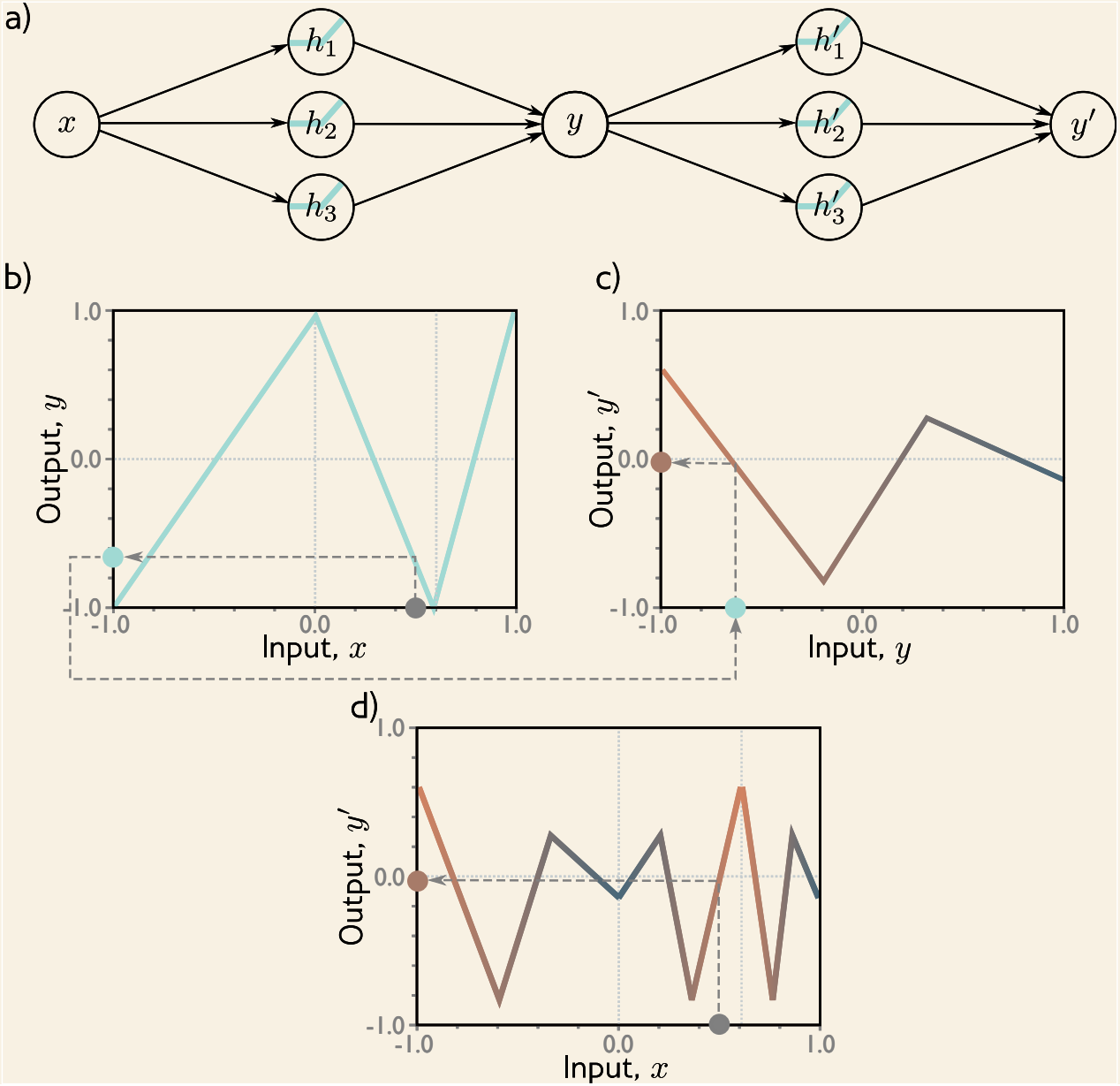

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

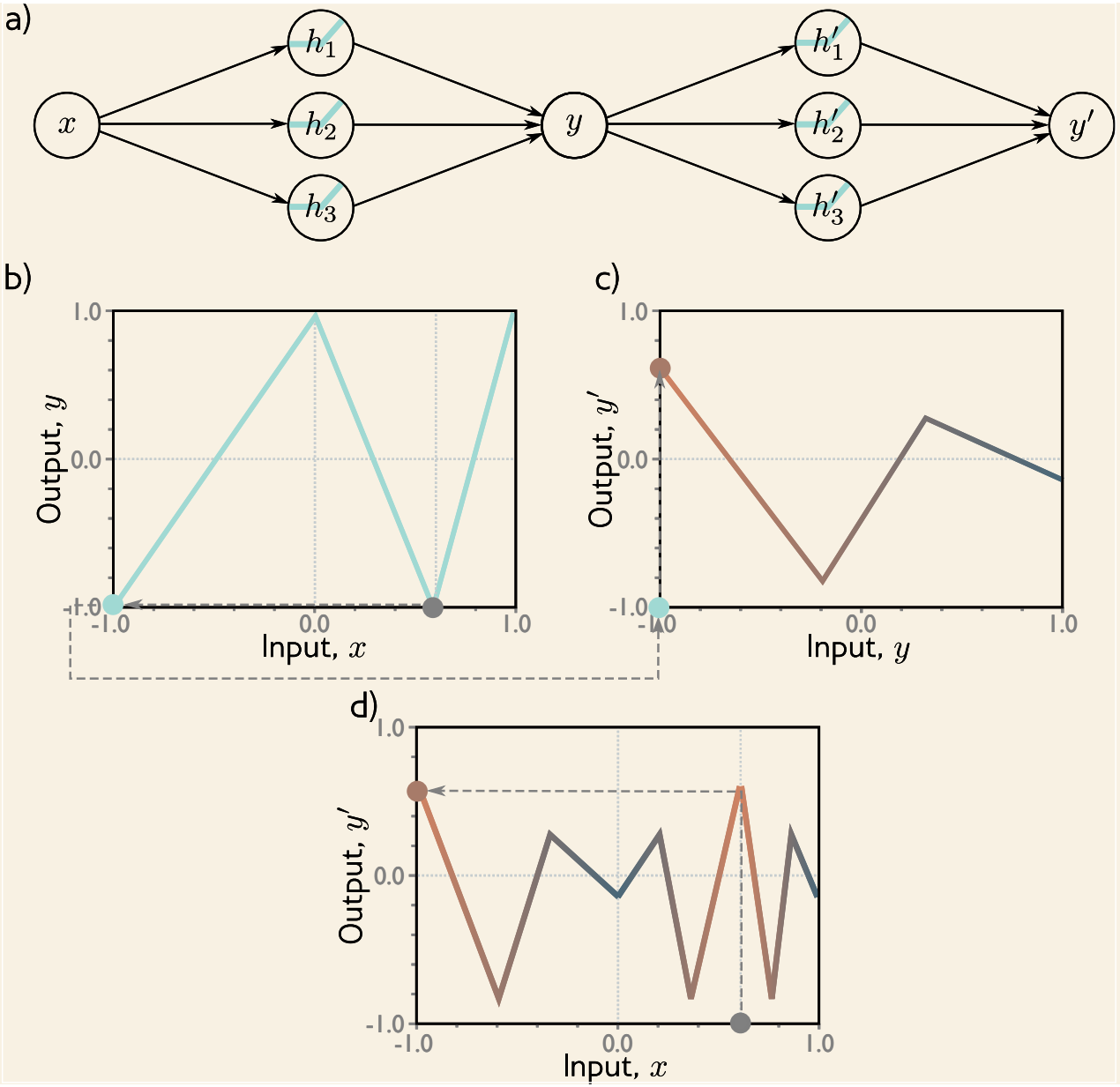

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x


a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x


a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x


a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

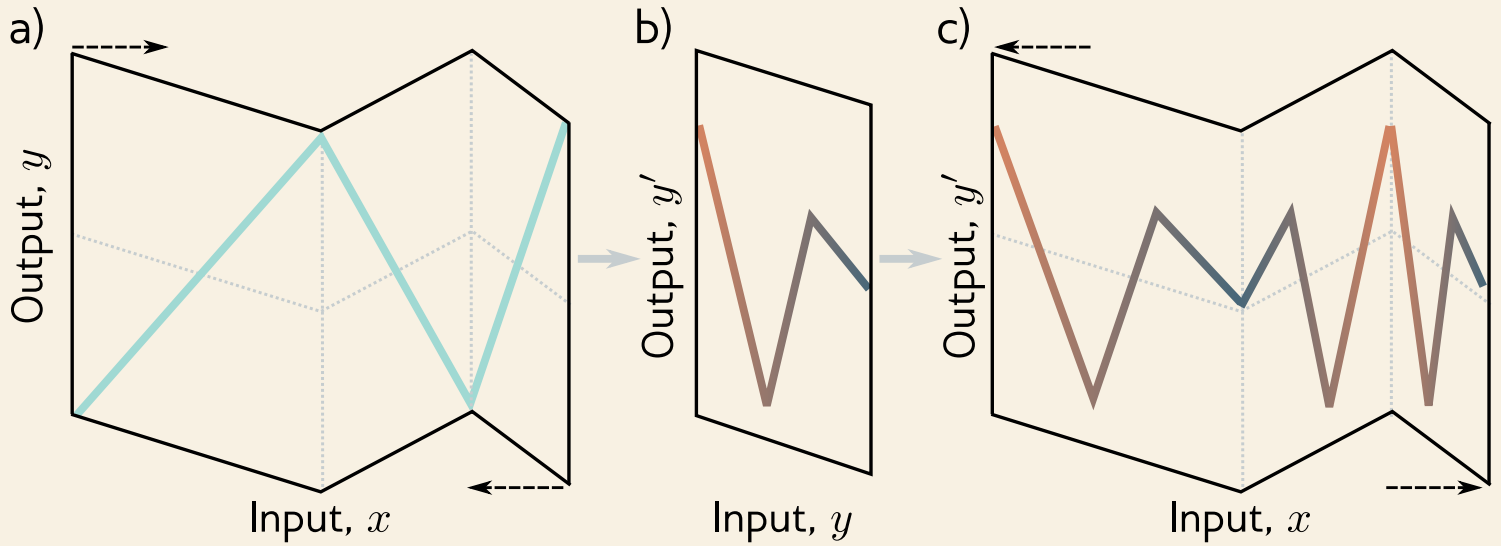

a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x


a)
x
h_1
h_2
h_3
y
h'_1
h'_2
h'_3
y'
b)
Output, y
Input, x
1.0
0.0
-1.0
c)
Output, y'
Input, y
d)
Output, y'
Input, x

“Folding analogy”

The first network folds the input space x back onto itself so that multiple inputs generate the same output.

The second network applies a function which is replicated at all points that were folded on top of one onother.



Combine two networks into one

Network 1:

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

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

Network 2:

$$h'_1 = \mathrm{a}[\theta'_{10} + \theta'_{11}y]$$
$$h'_2 = \mathrm{a}[\theta'_{20} + \theta'_{21}y]$$
$$h'_3 = \mathrm{a}[\theta'_{30} + \theta'_{31}y]$$

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

Hidden units of second network in terms of first:

$$\begin{aligned}
h'_1 &= \mathrm{a}[\theta'_{10} + \theta'_{11}y] &&= \mathrm{a}[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1 h_1 + \theta'_{11}\phi_2 h_2 + \theta'_{11}\phi_3 h_3] \\
h'_2 &= \mathrm{a}[\theta'_{20} + \theta'_{21}y] &&= \mathrm{a}[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1 h_1 + \theta'_{21}\phi_2 h_2 + \theta'_{21}\phi_3 h_3] \\
h'_3 &= \mathrm{a}[\theta'_{30} + \theta'_{31}y] &&= \mathrm{a}[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1 h_1 + \theta'_{31}\phi_2 h_2 + \theta'_{31}\phi_3 h_3]
\end{aligned}$$

Create new variables

$$
\begin{aligned}
h_1' = \quad & \mathrm{a}[\theta_{10}' + \theta_{11}' y] \quad = \quad \mathrm{a}[\theta_{10}' + \theta_{11}'\phi_0 + \theta_{11}'\phi_1 h_1 + \theta_{11}'\phi_2 h_2 + \theta_{11}'\phi_3 h_3] \\
h_2' = \quad & \mathrm{a}[\theta_{20}' + \theta_{21}' y] \quad = \quad \mathrm{a}[\theta_{20}' + \theta_{21}'\phi_0 + \theta_{21}'\phi_1 h_1 + \theta_{21}'\phi_2 h_2 + \theta_{21}'\phi_3 h_3] \\
h_3' = \quad & \mathrm{a}[\theta_{30}' + \theta_{31}' y] \quad = \quad \mathrm{a}[\theta_{30}' + \theta_{31}'\phi_0 + \theta_{31}'\phi_1 h_1 + \theta_{31}'\phi_2 h_2 + \theta_{31}'\phi_3 h_3]
\end{aligned}
$$

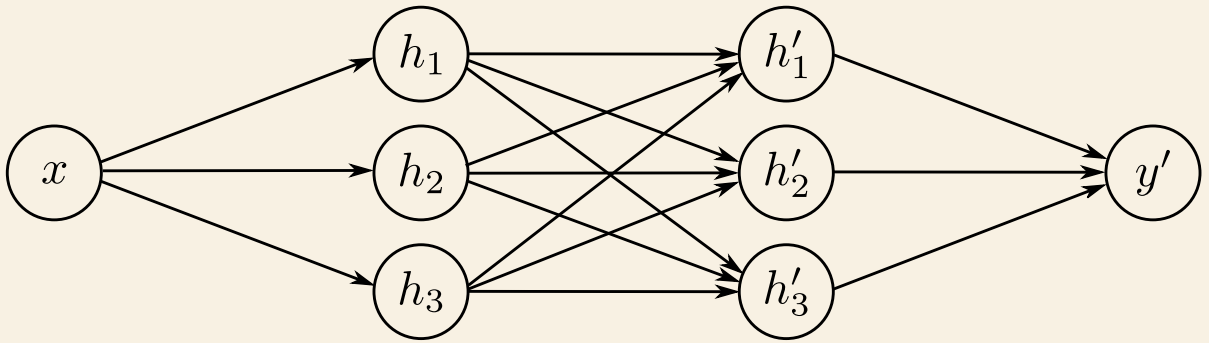
$$
\begin{aligned}
h_1' &= \mathrm{a}[\psi_{10} + \psi_{11} h_1 + \psi_{12} h_2 + \psi_{13} h_3] \\
h_2' &= \mathrm{a}[\psi_{20} + \psi_{21} h_1 + \psi_{22} h_2 + \psi_{23} h_3] \\
h_3' &= \mathrm{a}[\psi_{30} + \psi_{31} h_1 + \psi_{32} h_2 + \psi_{33} h_3]
\end{aligned}
$$

Two-layer network

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

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

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



Two-layer network as one equation

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

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

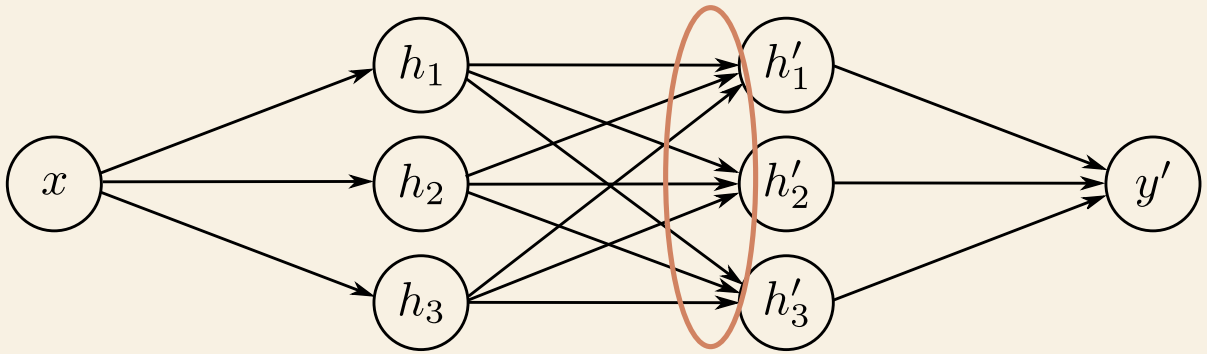
$$
\begin{aligned}
y' = \phi'_0 &+ \phi'_1 \mathrm{a}\left[\psi_{10} + \psi_{11}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{12}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{13}\mathrm{a}[\theta_{30} + \theta_{31}x]\right] \\
&+ \phi'_2 \mathrm{a}[\psi_{20} + \psi_{21}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{22}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{23}\mathrm{a}[\theta_{30} + \theta_{31}x]] \\
&+ \phi'_3 \mathrm{a}[\psi_{30} + \psi_{31}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{32}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{33}\mathrm{a}[\theta_{30} + \theta_{31}x]]
\end{aligned}
$$

Networks as composing functions

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

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

Consider the pre-activations at the second hidden units
At this point, it's a one--layer network with three outputs

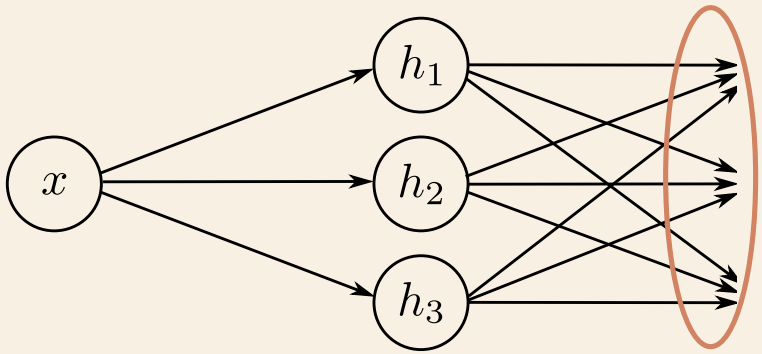


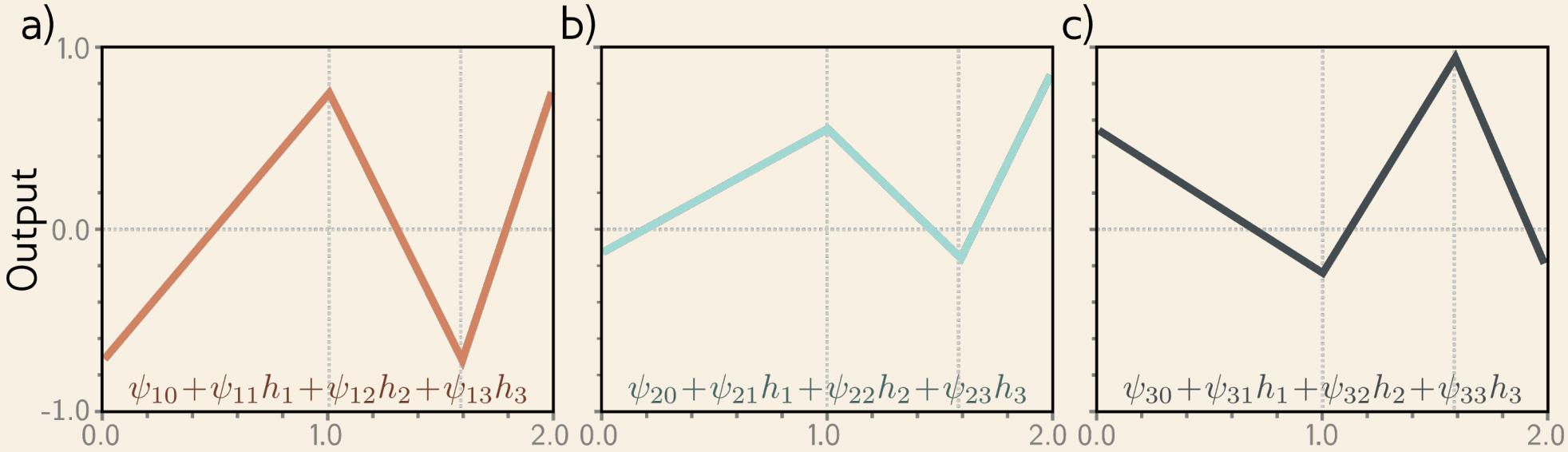
Networks as composing functions

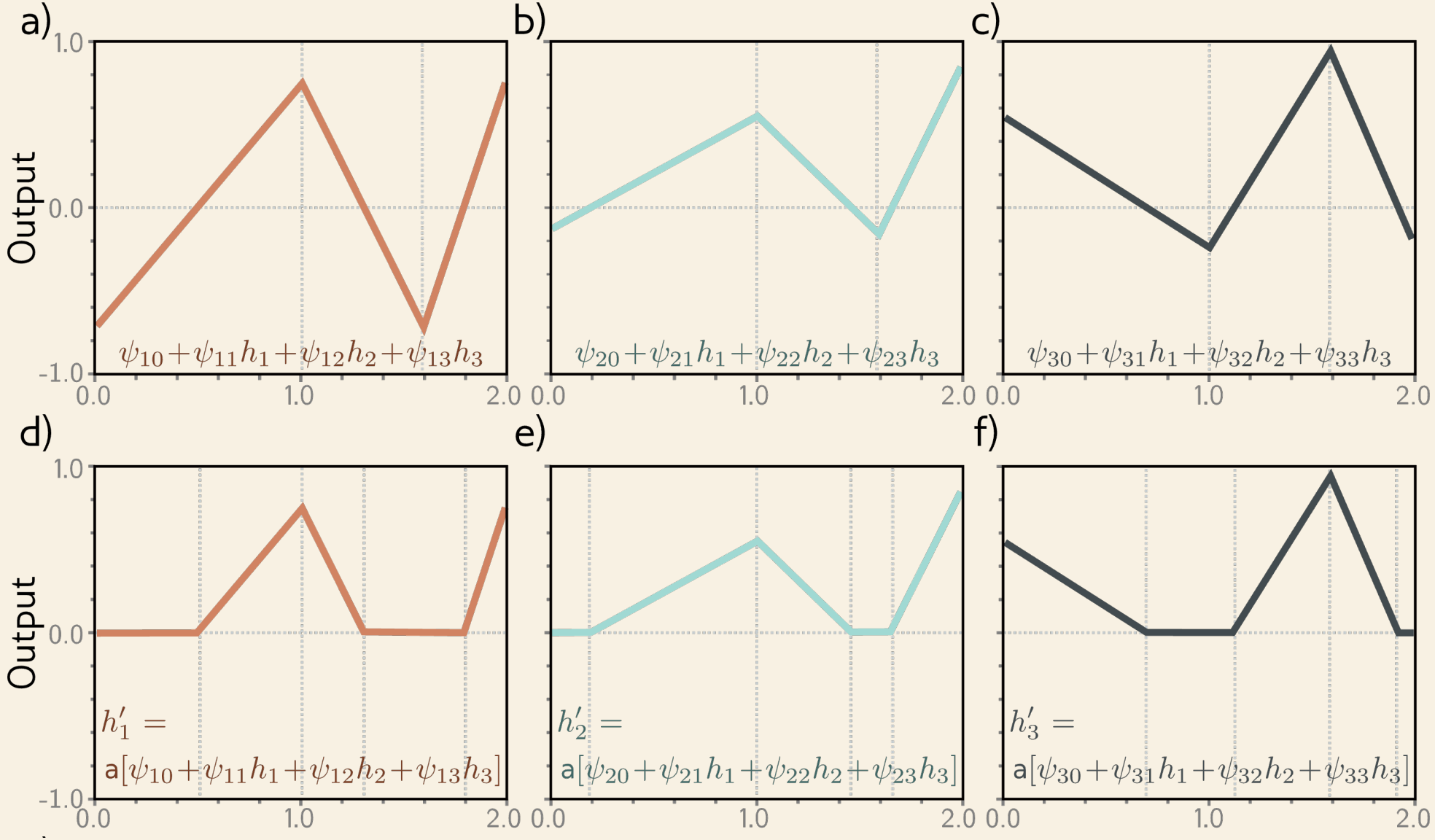
$$h_1 = a[\theta_{10} + \theta_{11}x]$$
$$h_2 = a[\theta_{20} + \theta_{21}x]$$
$$h_3 = a[\theta_{30} + \theta_{31}x]$$

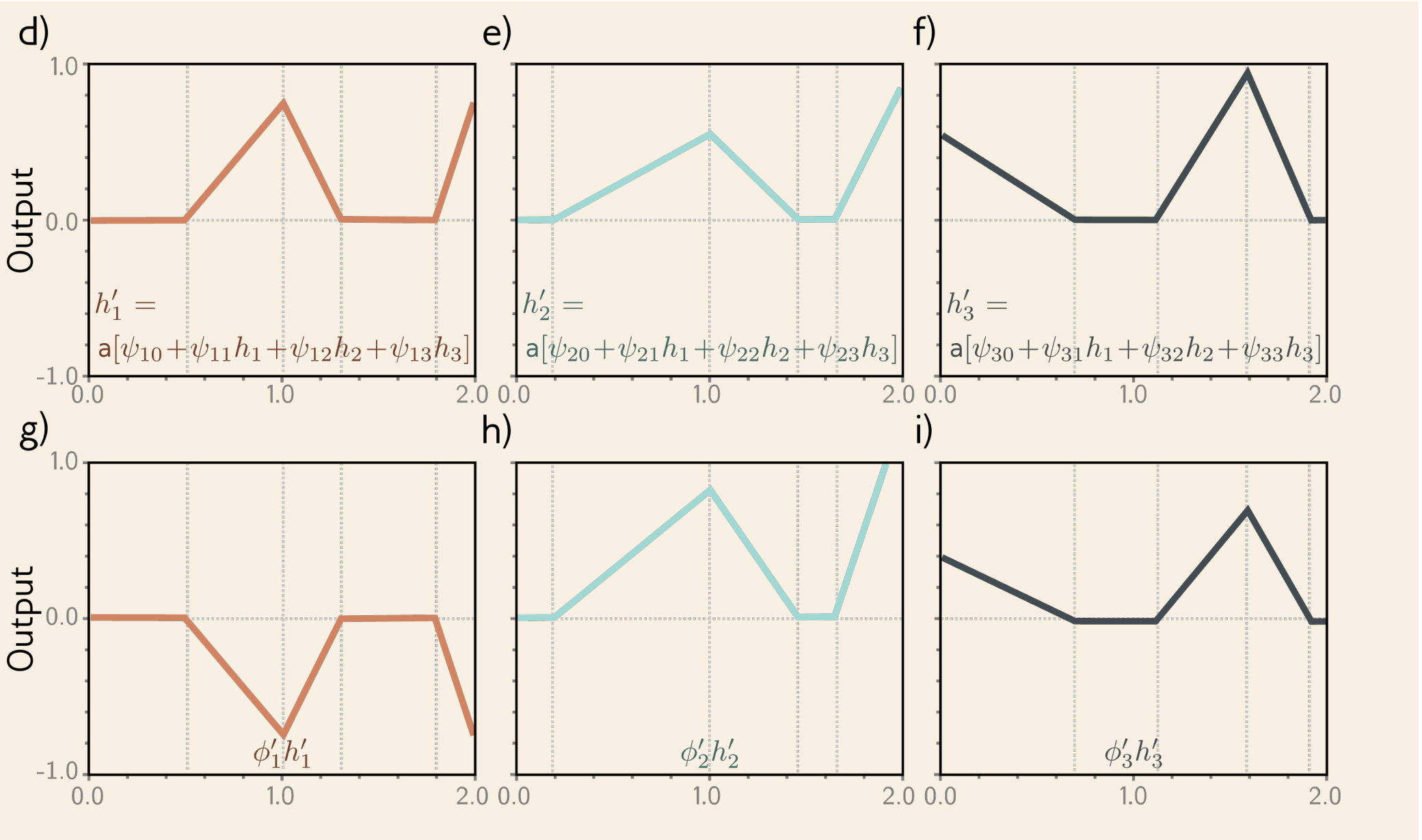
$$h'_1 = a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$
$$h'_2 = a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3]$$
$$h'_3 = a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]$$

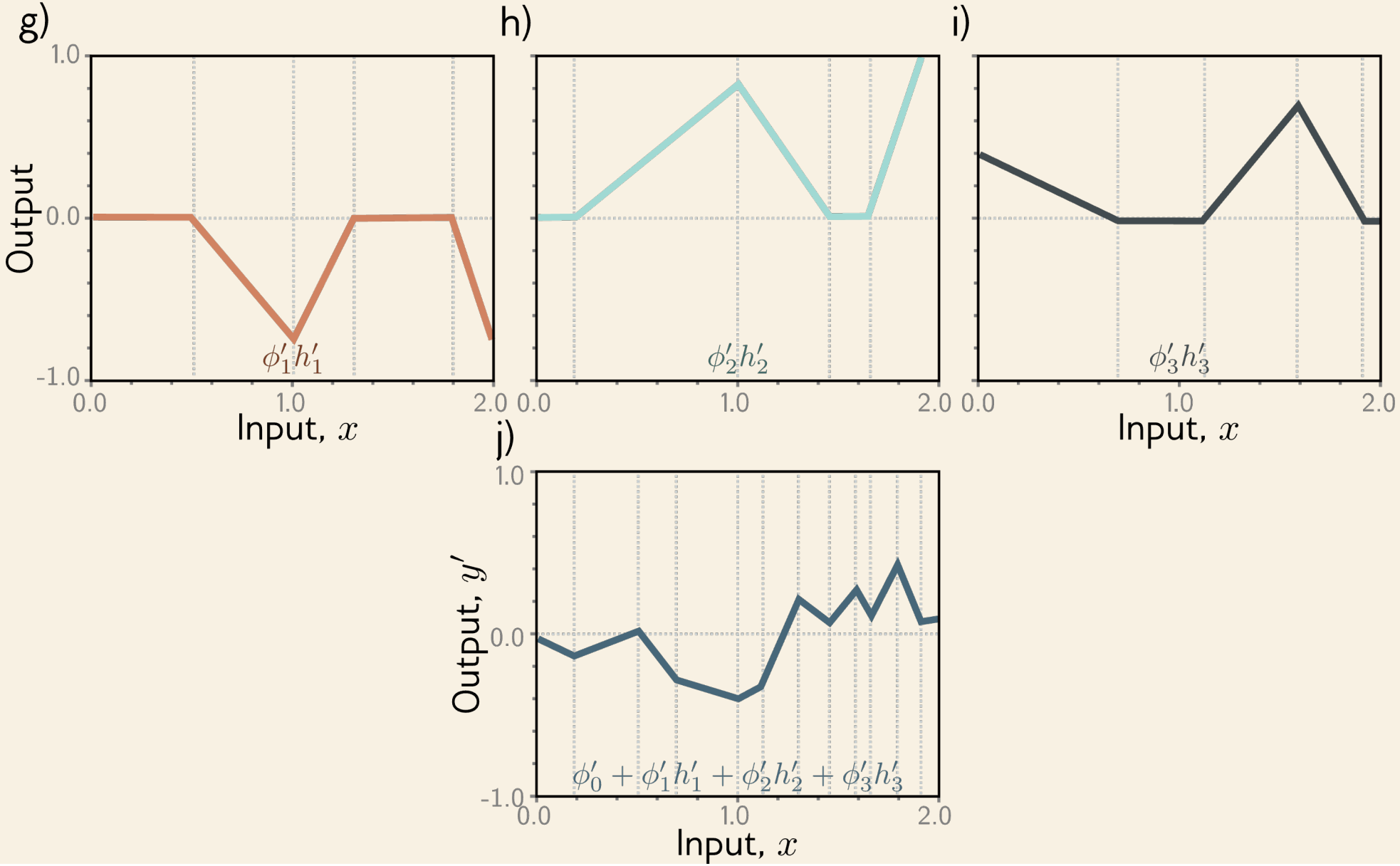
Consider the pre-activations at the second hidden units
At this point, it's a one--layer network with three outputs




a)
b)
c)
Output
1.0
0.0
-1.0
0.0
1.0
2.0
$\psi_{10}+\psi_{11}h_1+\psi_{12}h_2+\psi_{13}h_3$
$\psi_{20}+\psi_{21}h_1+\psi_{22}h_2+\psi_{23}h_3$
$\psi_{30}+\psi_{31}h_1+\psi_{32}h_2+\psi_{33}h_3$


a)
b)
c)
d)
e)
f)
Output
Output
1.0
0.0
-1.0
0.0
1.0
2.0
$\psi_{10}+\psi_{11}h_1+\psi_{12}h_2+\psi_{13}h_3$
$\psi_{20}+\psi_{21}h_1+\psi_{22}h_2+\psi_{23}h_3$
$\psi_{30}+\psi_{31}h_1+\psi_{32}h_2+\psi_{33}h_3$
$h_1' =$
$\mathsf{a}[\psi_{10}+\psi_{11}h_1+\psi_{12}h_2+\psi_{13}h_3]$
$h_2' =$
$\mathsf{a}[\psi_{20}+\psi_{21}h_1+\psi_{22}h_2+\psi_{23}h_3]$
$h_3' =$
$\mathsf{a}[\psi_{30}+\psi_{31}h_1+\psi_{32}h_2+\psi_{33}h_3]$


d)
e)
f)
g)
h)
i)
Output
Output
1.0
0.0
-1.0
0.0
1.0
2.0
$h_1' = \mathrm{a}[\psi_{10}+\psi_{11}h_1+\psi_{12}h_2+\psi_{13}h_3]$
$h_2' = \mathrm{a}[\psi_{20}+\psi_{21}h_1+\psi_{22}h_2+\psi_{23}h_3]$
$h_3' = \mathrm{a}[\psi_{30}+\psi_{31}h_1+\psi_{32}h_2+\psi_{33}h_3]$
$\phi_1' h_1'$
$\phi_2' h_2'$
$\phi_3' h_3'$


g)
h)
i)
j)
1.0
0.0
-1.0
Output
$\phi_1' h_1'$
$\phi_2' h_2'$
$\phi_3' h_3'$
0.0
1.0
2.0
Input, x
Output, y'
$\phi_0' + \phi_1' h_1' + \phi_2' h_2' + \phi_3' h_3'$

Notation change #1

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

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

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

Notation change #1

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

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

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

Notation change #1

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

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

Notation change #1

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

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

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

Notation change #2

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x\right] \longrightarrow \mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right]$$

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a}\left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{32} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}\right] \longrightarrow \mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right]$$

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

Notation change #3

$$\mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \longrightarrow \mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \longrightarrow \mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1]$$

$$y = \boldsymbol{\phi}_0' + \boldsymbol{\phi}'\mathbf{h}' \longrightarrow \mathbf{y} = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

Notation change #3

Bias vector

Weight matrix

$$\mathbf{h} = \mathbf{a}\left[\boldsymbol{\theta}_0 + \boldsymbol{\theta} x\right] \longrightarrow \mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a}\left[\boldsymbol{\psi}_0 + \boldsymbol{\Psi}\mathbf{h}\right] \longrightarrow \mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1]$$

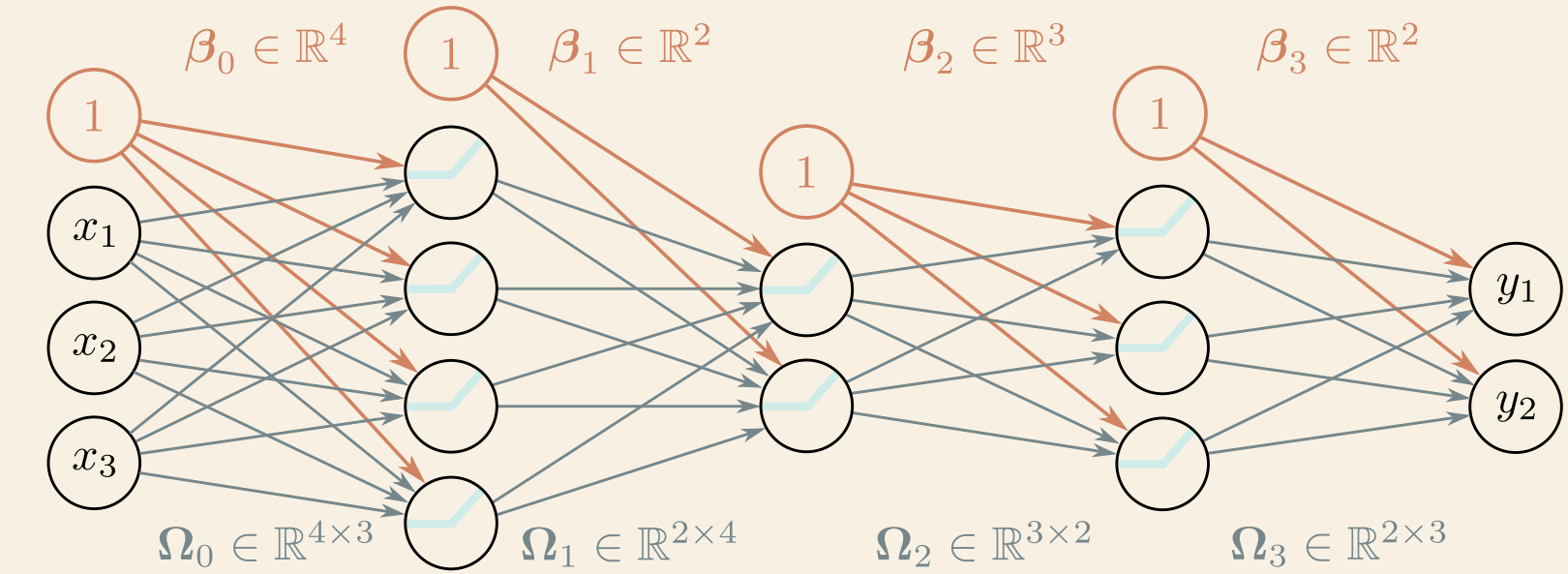
$$y = \boldsymbol{\phi}_0' + \boldsymbol{\phi}'\mathbf{h}' \longrightarrow \mathbf{y} = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

General equations for deep network

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

$$
\mathbf{y} = \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{a}\left[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{a}\left[\ldots\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{a}\left[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{a}\left[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}\right]\right]\ldots\right]\right]
$$

Example



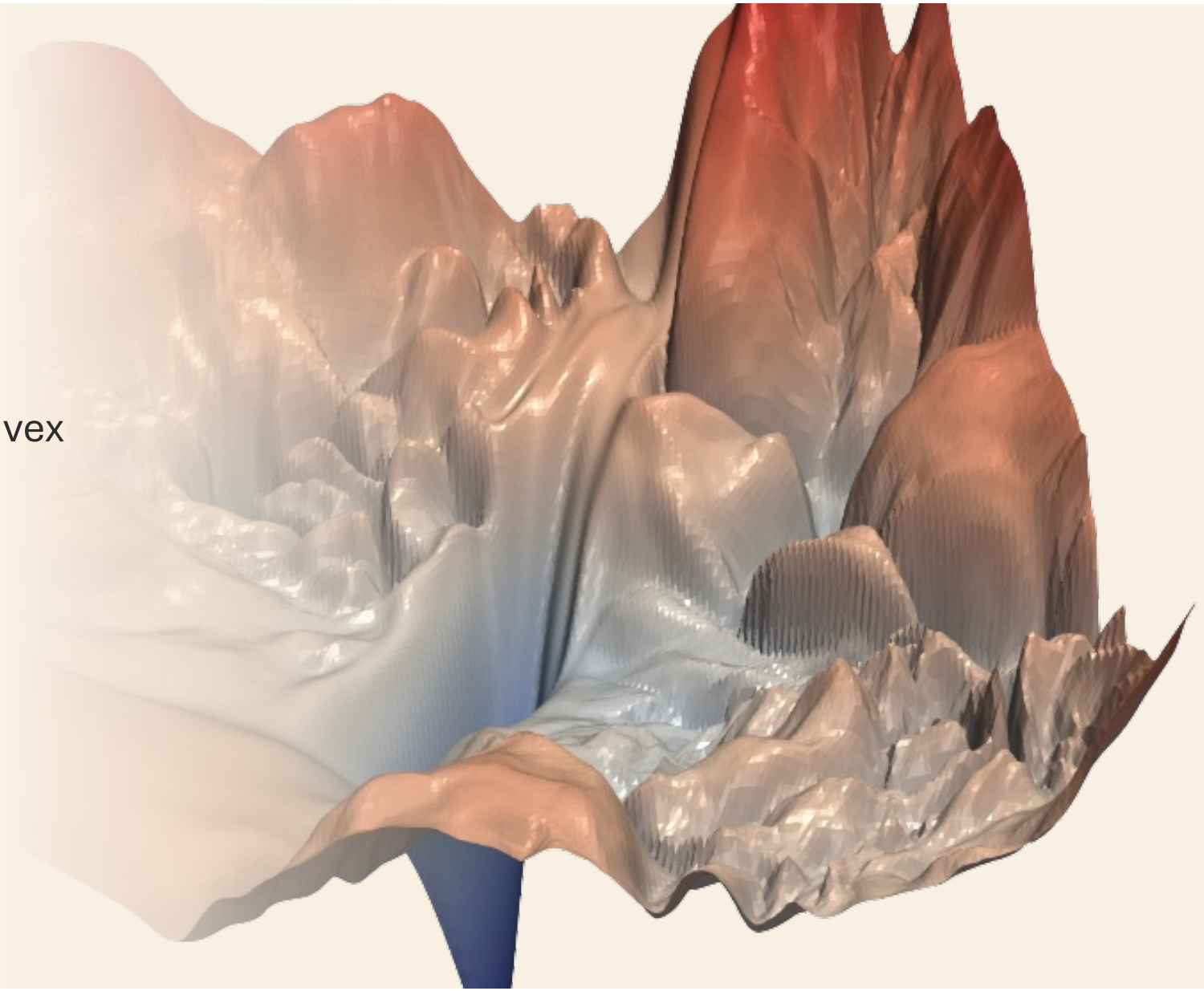
Input, $\mathbf{x}$	Hidden layer, $\mathbf{h}_1$	Hidden layer, $\mathbf{h}_2$	Hidden layer, $\mathbf{h}_3$	Output, $\mathbf{y}$
$D_i = 3$	$D_1 = 4$	$D_2 = 2$	$D_3 = 3$	$D_o = 2$

Optimization Challenges

Local minima, saddle points due to non-convex loss

Narrow valleys

Exploding/vanishing gradient



Minibatch Gradient descent

- While not converged, on dev, sample data in pieces (batches) updating model
- Training Set
- EPOCH

$$\theta = \theta - \mu \nabla Loss(\text{batch } 1)$$
$$\theta = \theta - \mu \nabla Loss(\text{batch } 2)$$
$$\theta = \theta - \mu \nabla Loss(\text{batch } 3)$$
$$\theta = \theta - \mu \nabla Loss(\text{batch } 4)$$

Momentum-Based Optimization

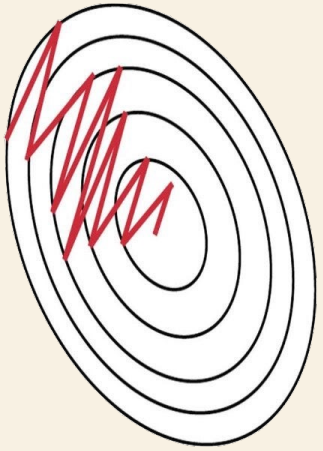
Gradient Descent:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(f_\theta(x), \mathrm{y})$$

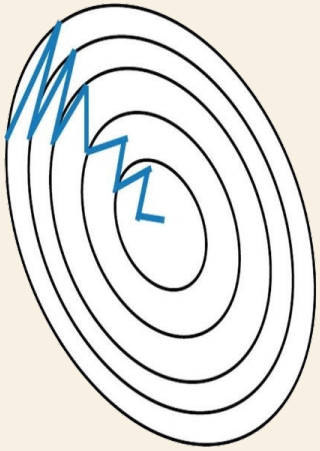
Momentum Update:

$$\rho \leftarrow \mu\rho - \alpha \nabla_\theta L$$
$$\theta \leftarrow \theta + \rho$$

- Momentum accumulates direction of past gradients
- Helps escape small local minima and narrow valleys
- Typical μ ≈ 0.9 – 0.99



Stochastic Gradient Descent **without** Momentum



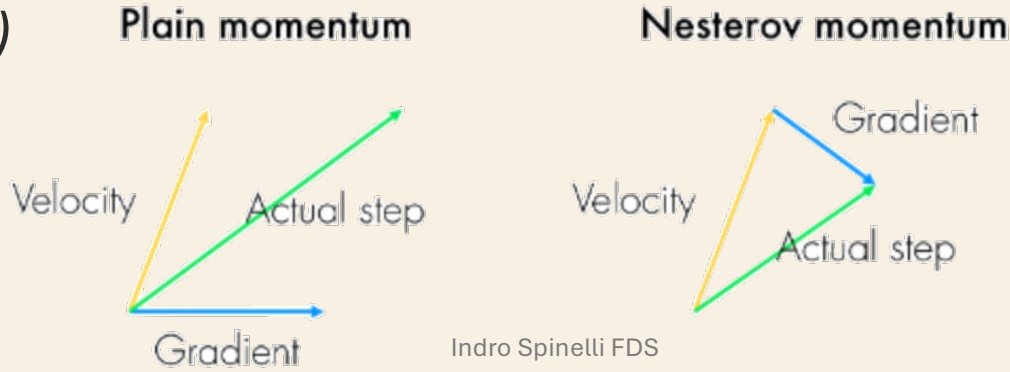
Stochastic Gradient Descent **with** Momentum

Nesterov Momentum

Lookahead gradient: compute gradient at future position

$$\rho \leftarrow \mu\rho - \alpha\nabla_\theta L(f_{\theta+\mu\rho}(x), \mathrm{y})$$

- Prevents overshooting when near optimum
- Often converges faster and smoother than vanilla momentum
- *(Visual: arrows showing "look-ahead" step vs standard momentum)*



Adaptive Learning Rates

Key idea: adjust the step size per parameter

AdaGrad: divides update by accumulated squared gradients → good for sparse features

$$
\begin{aligned}
g_t &= \nabla_\theta L(f_\theta(x_\mathrm{t}), \mathrm{y_t}) \\
G_t &= G_{t-1} + g_t \odot g_t \\
\theta_t &= \theta_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t
\end{aligned}
$$

Adaptive Learning Rates

Key idea: adjust the step size per parameter

AdaGrad: divides update by accumulated squared gradients → good for sparse features

$$g_t = \nabla_\theta L(f_\theta(x_t), y_t)$$
$$G_t = G_{t-1} + g_t^2$$
$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} g_t$$

g_t: gradient at step t

G_t: element-wise sum of squared gradients

ϵ: small constant for numerical stability

Adaptive Learning Rates

RMSProp: Adapts learning rate per parameter, avoiding AdaGrad's rapid decay. uses exponential moving average instead of full history

$$g_t = \nabla_\theta L(f_\theta(x_t), y_t)$$

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

γ: decay rate (typically 0.9)

$E[g^2]_t$: exponentially decaying average of past squared gradients

Adaptive Learning Rates

Adam: Combines momentum (*first moment*) and adaptive learning rate (*second moment*)

$$
\begin{aligned}
g_t &= \nabla_\theta L(f_\theta(x_t), y_t) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\end{aligned}
$$

Typical values: $\beta_1 = 0.9,\ \beta_2 = 0.999,\ \epsilon = 10^{-8}$

Default optimizer in many deep learning frameworks

Learning Rate Schedules

The *most critical hyperparameter!!!*

Use validation loss to guide tuning

- *Decay schedules*: reduce LR when progress stalls
  - Popular strategies:
    - Step decay
    - Exponential decay
    - Cosine annealing

Weight Initialization

Why initialization matters:

- Prevents gradients from exploding or vanishing
- Ensures all neurons start from diverse states

Bad practice: zero initialization → identical neurons

Long history of trick based on “fan in” aka input dimension d_{in}

Xavier (Glorot) Init: for tanh activations $W_j \sim \mathcal{N}\left(0, \frac{1}{d_{in}+d_{out}}\right)$

Kaiming (He) Init: for ReLU activations $W_j \sim \mathcal{N}\left(0, \frac{2}{d_{in}}\right)$

Activation Functions: Why They Matter

Purpose: introduce non-linearity to neural networks

- Enable modeling of complex input–output mappings
- Affect both *gradient flow* and *training stability*
- Choice of activation impacts network depth and performance

Issues with sigmoid

Sigmoid function:

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

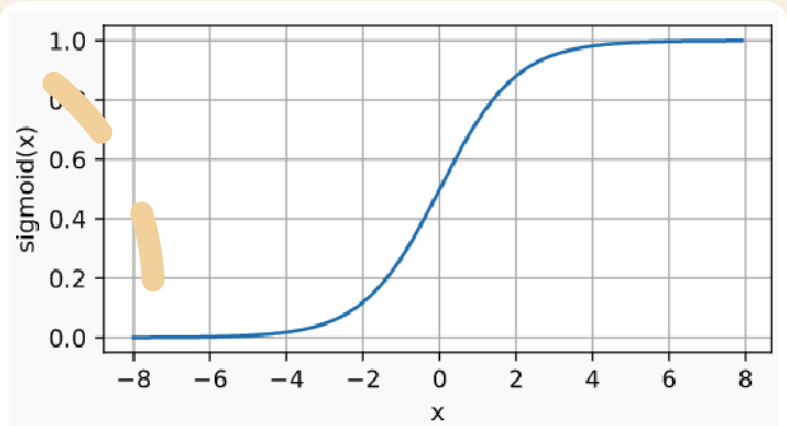
Derivative:

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

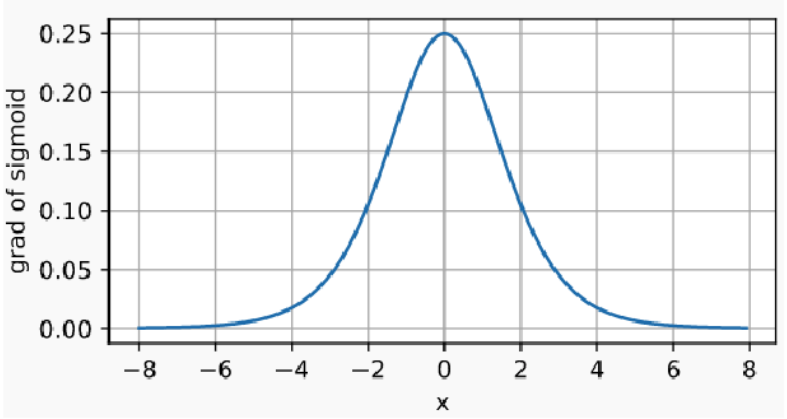
Problem: $\sigma'(x) \leq 0.25$, so gradient multiplies by ≤ 0.25 at each layer.

Consequence: In deep networks, gradients shrink exponentially → early layers learn very slowly.

Example: For 10 layers, max gradient ≈ 0.25^{10} ≈ 10^{-6}



sigmoid



sigmoid gradient

Relu as solution

ReLU activation: ReLU(x) = max(0,x)

Derivative: $f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$

Why it solves vanishing gradient:

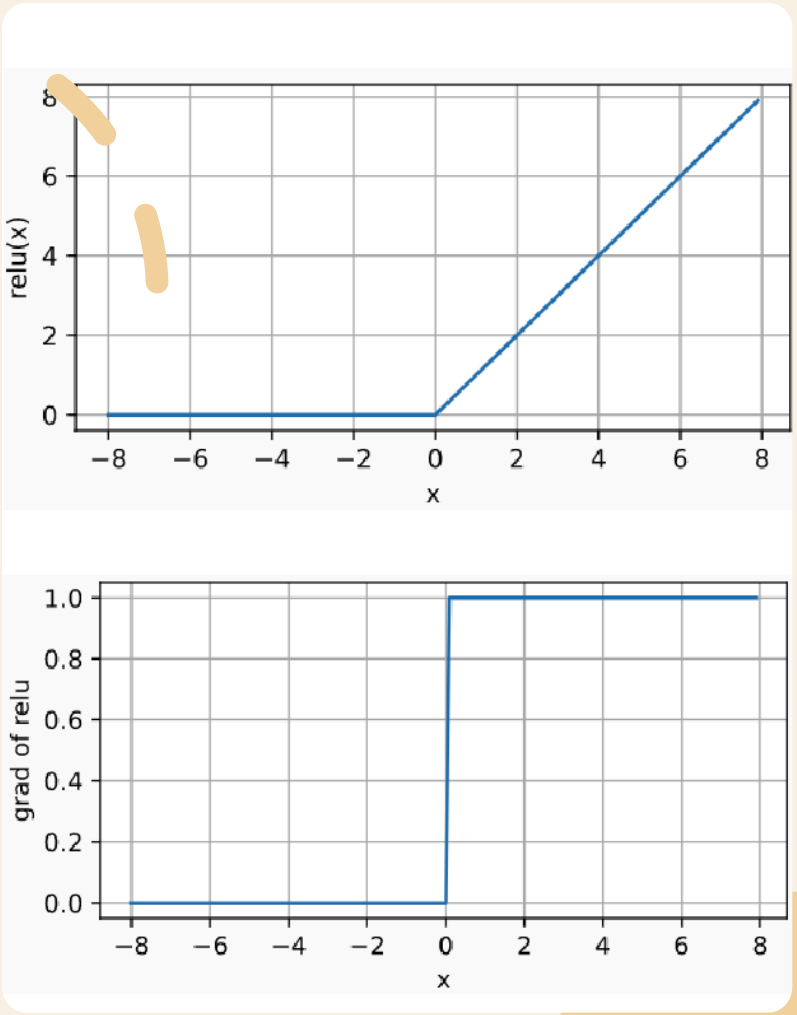
Derivative = 1 for positive activations → gradient does **not shrink** in backprop.

Enables deep networks to train efficiently.

Sparse activations → more efficient computation.

Simpler gradient propagation → faster convergence.

Caveat: “Dying ReLU” problem (neurons stuck at 0). Can be mitigated with **Leaky**



Residual Connections

- **Introduction:** Introduced in Residual Networks (ResNets) to solve the vanishing gradient problem.
- **Core Idea:** Instead of forcing a layer to learn a complete transformation $H(x)$,we have it learn a *residual* (or *change*) $F(x)$ to *add* to the input x.

$$h = F(x) + x$$

Intuition: It's easier for a layer to learn to "do nothing" (just output $F(x) = 0$than to learn the identity function ($H(x) = x$). This "skip connection" passes the original input x straight through.

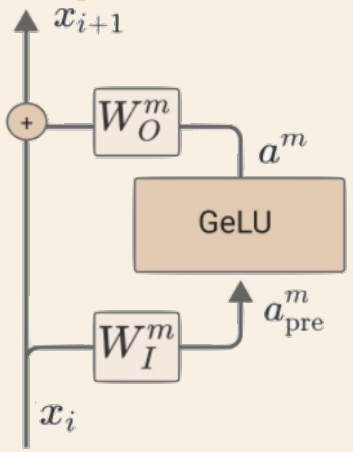
Residual Connections

During backpropagation:

$$\nabla_x(F(x) + x) = \nabla_x F(x) + 1$$

The **+1** creates a direct, uninterrupted "highway" for gradients to flow back through the network.

- Gradients can propagate through hundreds of layers without vanishing.
- The network can **easily learn the identity function** $(F(x) = 0)$, so adding more layers doesn't hurt performance.



$$x_{i+1} = x_i + W_O^m a^m$$

$$a^m = \text{GeLU}(a^m_{\text{pre}})$$

$$a^m_{\text{pre}} = W_I^m x_i$$

Deep Residual Networks (ResNet)

What they are: ResNets are built by stacking these residual units. is typically a block of 2-3 convolutional layers (next week).

The Results (ImageNet):

- A 152-layer ResNet achieved 3.57% top-5 error, winning ImageNet 2015.
- This was the first time a model surpassed human-level performance (which was ~5.1%).
- Researchers were able to train ResNets over 1000 layers deep.

Key Takeaway: Residual connections finally allowed for the successful training of *extremely* deep neural networks.

Refresher: Standard Scaling

What is it? A common pre-processing step for an *entire dataset* X.

Goal: Transform each feature (column) to have a zero mean and unit variance.

$$X' = (X - \mu)/\sigma$$

- μ is the mean of each feature.
- σ is the standard deviation of each feature.

Helps optimization algorithms converge faster.

Batch Normalization (BN): The Concept

The Idea: What if we applied this same "Standard Scaling" principle *inside* the network, for every layer's output?

The Problem:

- We don't have access to the *entire* dataset's statistics for each layer.
- These statistics would "shift" during training anyway as the network's parameters change (this is called "Internal Covariate Shift").

The Solution: Normalize the layer's output using the statistics (mean and variance) of the **current mini-batch.**

Batch Normalization (BN): During Training

Reduce **internal covariate shift** → stabilize and accelerate training.
Normalizes layer inputs across the **batch**:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- Then scales and shifts:

$$y_i = \alpha \hat{x}_i + \beta$$

  - Keeps mean ≈ 0 and variance ≈ 1 for each layer input
  - Allows **higher learning rates**
  - They allow the network to **adapt the mean and variance** for optimal performance.

Batch Normalization (BN): During Inference

Problem: We can't use mini-batch stats at test time. The output for one sample shouldn't depend on what other samples are in its "batch".

Standard Solution: Moving Averages

- During training, the network maintains a **moving average** of the mini-batch means (μ) and variances (σ^2).
- At inference time, these final, fixed averages are used to normalize the test input.

$$\hat{x}_i = \frac{x_i - \mu_{\text{avg}}}{\sqrt{\sigma^2_{\text{avg}} + \epsilon}} \qquad y_i = \alpha \hat{x}_i + \beta$$

Batch Normalization: Challenges

Mini-Batch Dependency:

- BN creates a dependency between samples in a mini-batch.
- This can be problematic for certain tasks (e.g., contrastive learning) or distributed training.

Fails on Small Batch Sizes:

- If the batch size is very small (e.g., 1 or 2), the mean and variance estimates are extremely noisy and unstable.
- This is a major problem for large models (like in NLP) that require small batches to fit in memory.

Layer Normalization

Normalizes **all features of a single sample** rather than across the batch:

$$\hat{x}_i = \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}}$$

- Then scales and shifts:

$$y_i = \gamma \hat{x}_i + \beta$$

  - Works well for **RNNs** or **small batch sizes**
  - Gradient flow is stabilized **per sample**, independent of batch size
  - Improves training stability in deep architectures

Dropout Regularization: The Core Idea

Goal: A regularization technique to prevent overfitting.

How it Works: During training, randomly "drop" (set to 0) a fraction of neurons at a given layer.

Hyperparameter p: The "dropout rate" (e.g., $p = 0.5$), or the probability a neuron is dropped. The "keep probability" is $(1 - p)$.

- **Intuition:**
  - Forces the network to learn redundant representations.
  - Prevents neurons from "co-adapting" or relying too much on specific other neurons.
  - It's like training many different "thinned" networks in parallel.

The Standard Implementation (Inverted Dropout)

The Problem: If we *only* drop neurons at train time, the layer's output will be much larger at test time (since all neurons are active).

Naïve Solution: At test time, scale all activations down by the keep probability $(1 - p)$.

$$y = (1 - p)x$$

This is inefficient! It requires different code for training and testing.

Standard Solution: Inverted Dropout

We scale the activations *during training* instead.

We divide the activations of the *kept* neurons by the keep probability.

Train: $h = \frac{(\text{mask} \odot x)}{(1-p)}$

Test: h = x

The test-time (inference) code is identical to a network *without* dropout. No scaling needed.

Slides from Understanding deep learning S. Prince Chapter 4, Prof. O. Bastani and Prof. M. Zhao