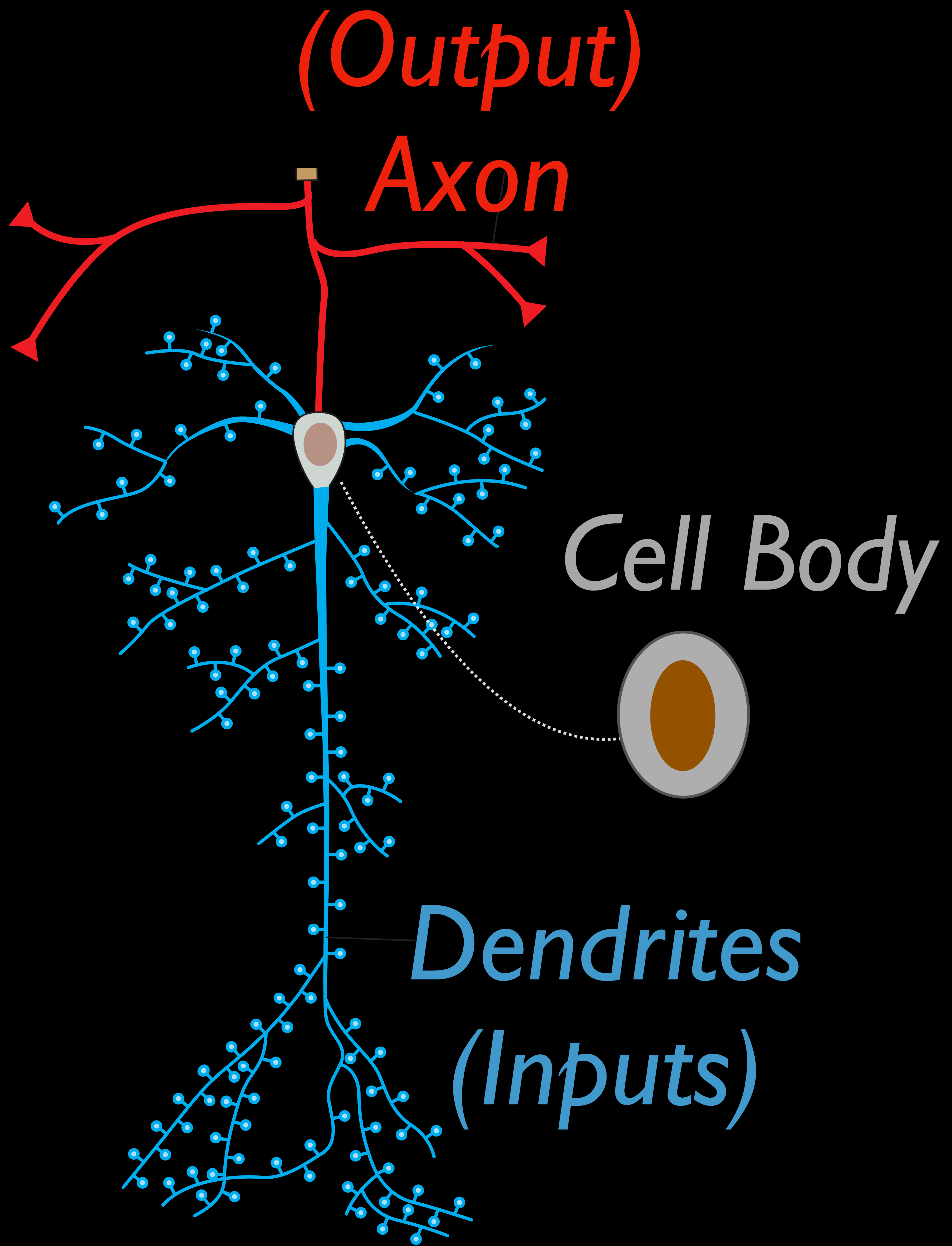
COS234:

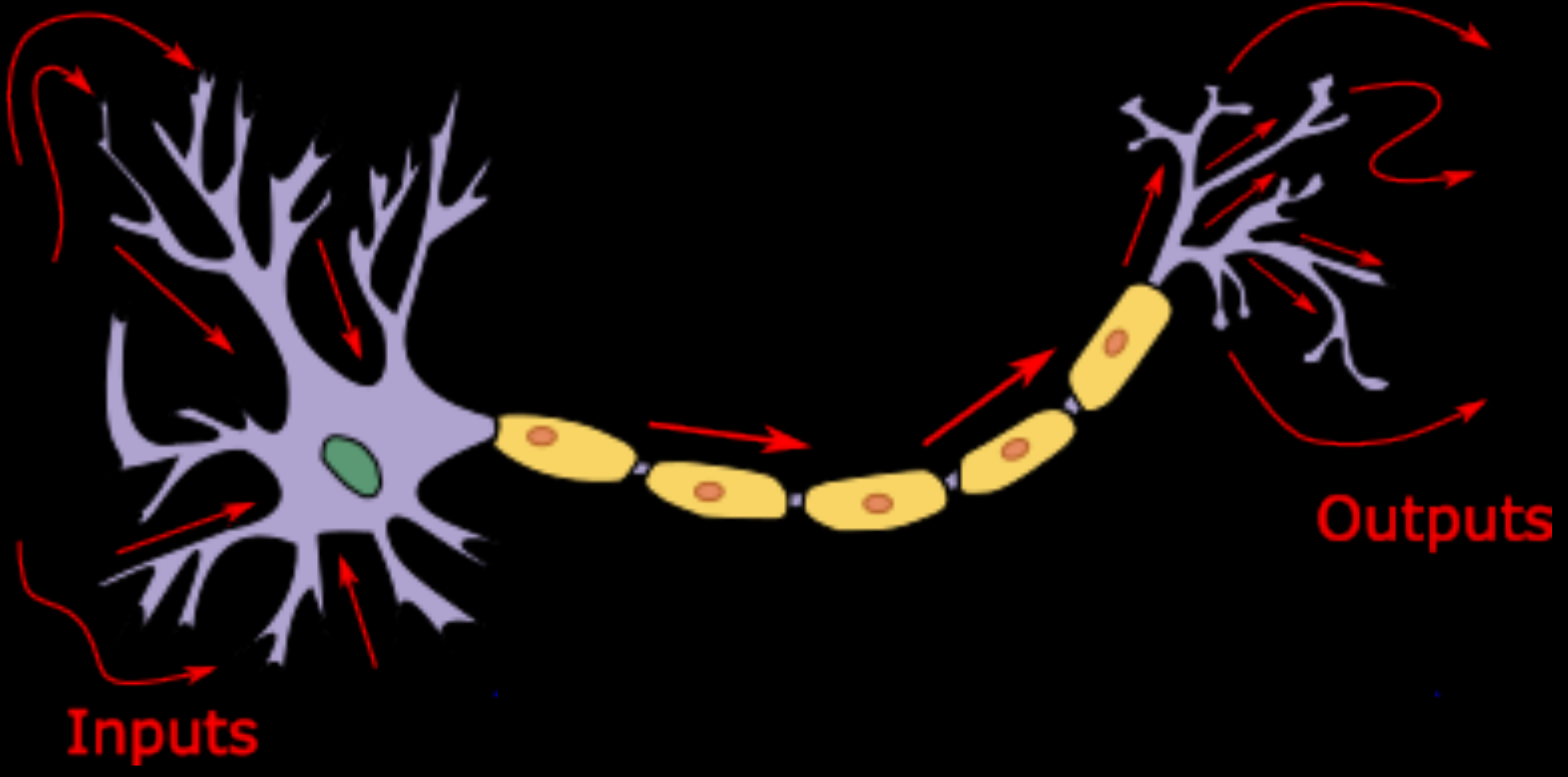
Introduction To Machine Learning

Prof. Yoram Singer

PRINCETON UNIVERSITY

Topic: Feed-Forward Neural Networks



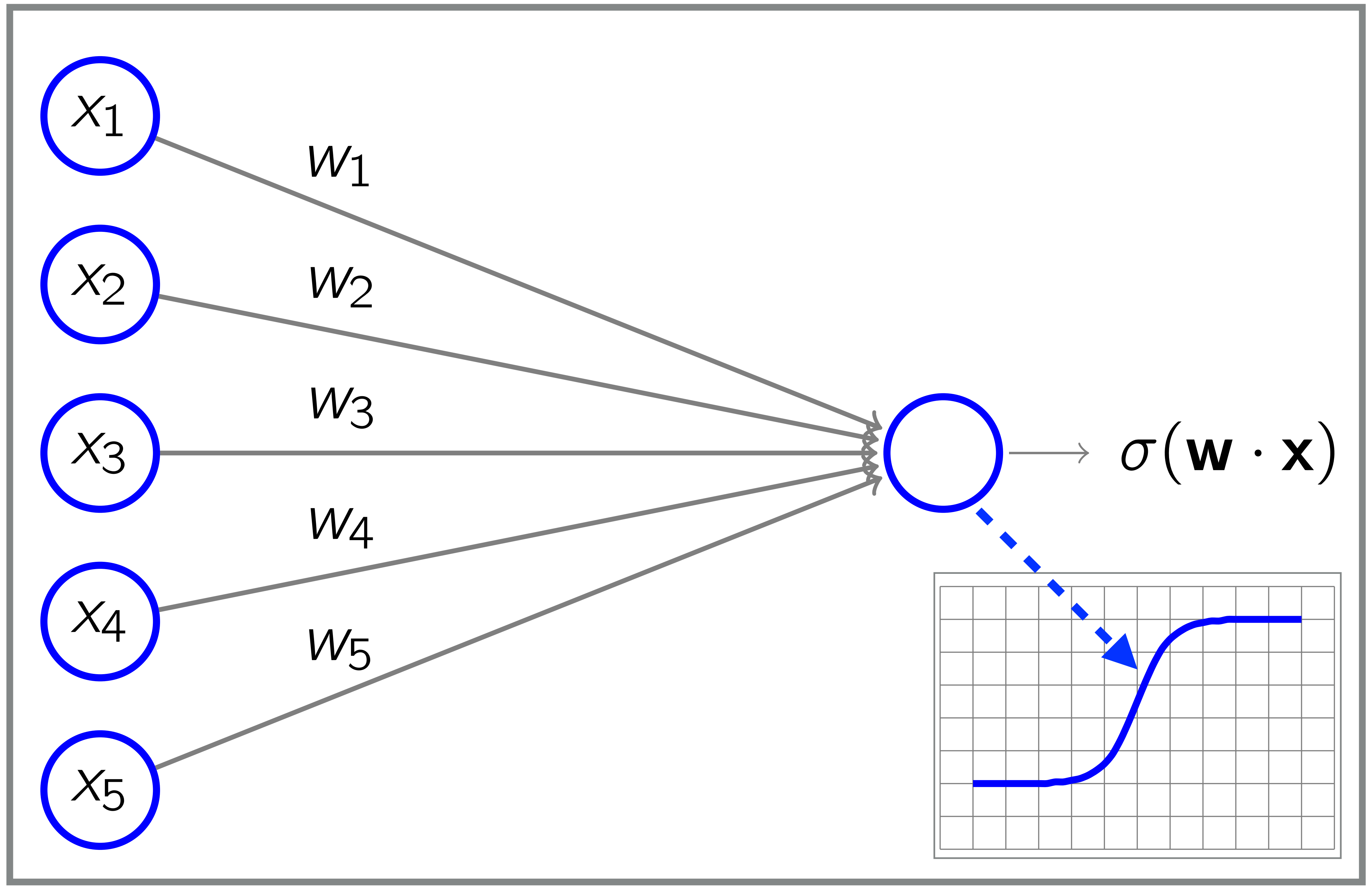


“Neuron”

GLM $\Rightarrow$ Neuron

Loss asside, GLM is input-output map:

$$\hat{y} = h(\mathbf{w} \cdot \mathbf{x})$$



Encapsulate as "black box" called Neuron

Reuse neurons with different parameters

Two Layer Neural Network

"Concatenate" multiple mappings:

$$\mathsf{h}_1(\mathbf{w}_1 \cdot \mathbf{x})\,, \mathsf{h}_2(\mathbf{w}_2 \cdot \mathbf{x})\,,\ \ldots\ , \mathsf{h}_m(\mathbf{w}_m \cdot \mathbf{x})$$

Activations need not be identical,

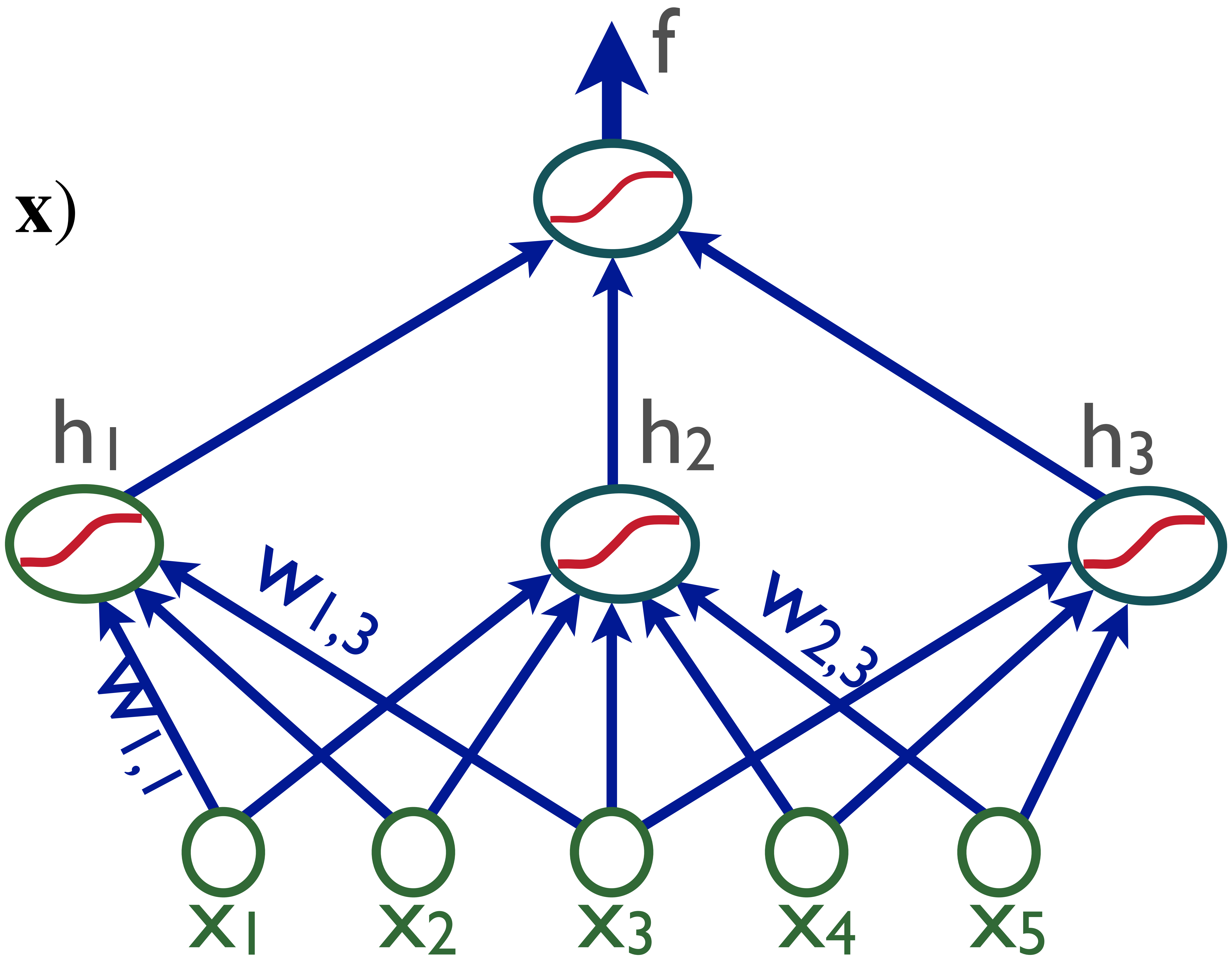
but typically: $\mathsf{h}_j = \mathsf{h}^{\text{generic}}$

Intermediate output is called a layer:

$$\mathbf{h}(\mathbf{x}) = \big(\mathsf{h}_1(\mathbf{x}), \mathsf{h}_2(\mathbf{x}), \ldots, \mathsf{h}_m(\mathbf{x})\big)$$

Output of network:

$$\mathsf{f}(\mathbf{x}) = \mathbf{u} \cdot \mathbf{h}(\mathbf{x})$$



Multilayer Net

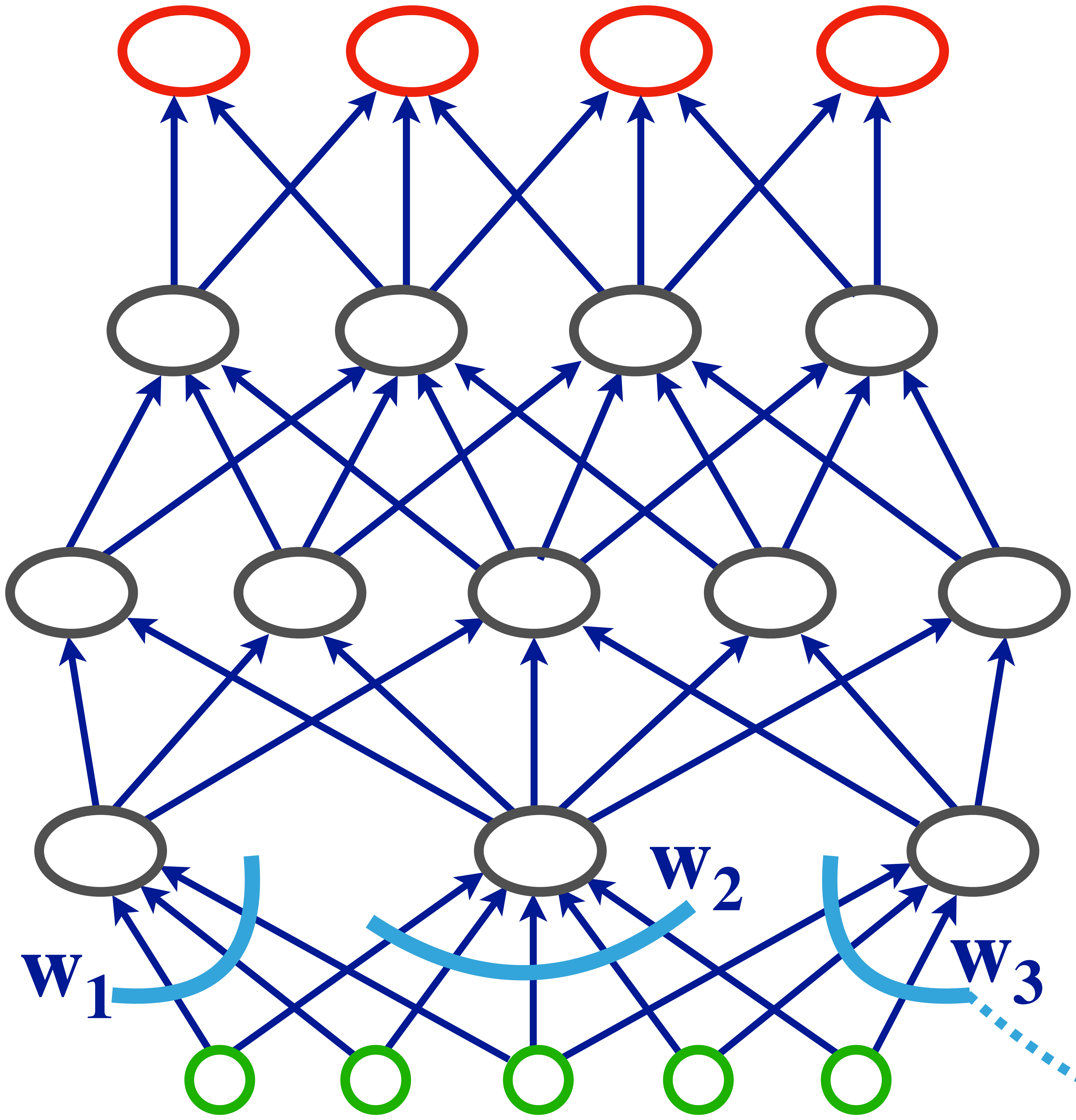
Output Layer

Hidden Layer III

Hidden Layer II

Hidden Layer I

Input Layer



$$A^4 = \begin{bmatrix} -\mathbf{r}_1- \\ -\mathbf{r}_2- \\ -\mathbf{r}_3- \\ -\mathbf{r}_4- \end{bmatrix} \in \mathbf{R}^{4\times 4}$$

$$A^3 = \begin{bmatrix} -\mathbf{v}_1- \\ -\mathbf{v}_2- \\ -\mathbf{v}_3- \\ -\mathbf{v}_4- \end{bmatrix} \in \mathbf{R}^{4\times 5}$$

$$A^2 = \begin{bmatrix} -\mathbf{u}_1- \\ -\mathbf{u}_2- \\ \cdots \\ -\mathbf{u}_5- \end{bmatrix} \in \mathbf{R}^{5\times 3}$$

$$A^1 = \begin{bmatrix} -\mathbf{w}_1- \\ -\mathbf{w}_2- \\ -\mathbf{w}_3- \end{bmatrix} \in \mathbf{R}^{3\times 5}$$

Multilayer Net

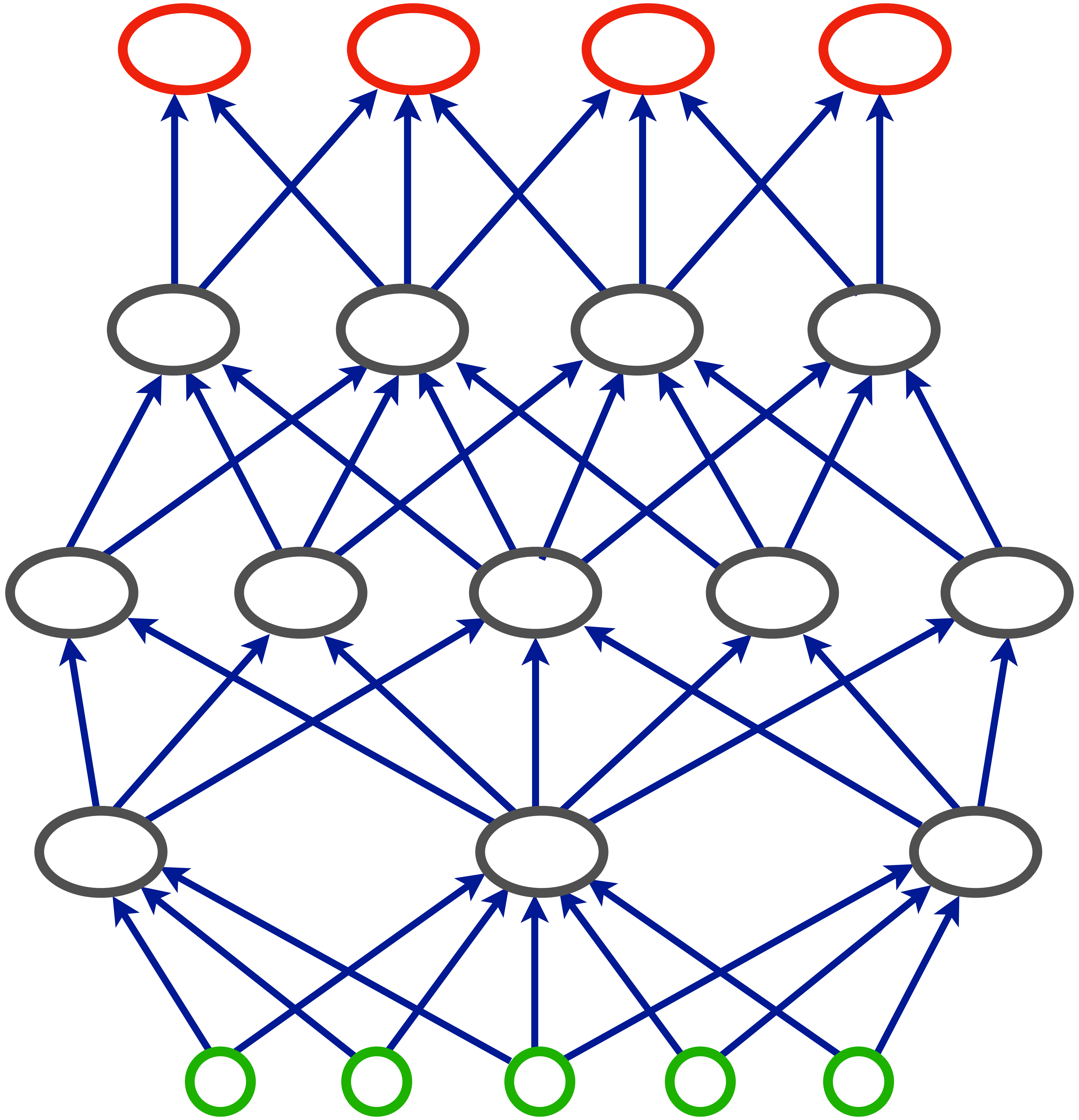
$\mathbf{h}^4 \in \mathbf{R}^4$

$\mathbf{h}^3 \in \mathbf{R}^4$

$\mathbf{h}^2 \in \mathbf{R}^5$

$\mathbf{h}^1 \in \mathbf{R}^3$

$\mathbf{h}^0 \in \mathbf{R}^5$



$$A^4 = \begin{bmatrix} -\mathbf{r}_1- \\ -\mathbf{r}_2- \\ -\mathbf{r}_3- \\ -\mathbf{r}_4- \end{bmatrix} \in \mathbf{R}^{4\times 4}$$

$$A^3 = \begin{bmatrix} -\mathbf{v}_1- \\ -\mathbf{v}_2- \\ -\mathbf{v}_3- \\ -\mathbf{v}_4- \end{bmatrix} \in \mathbf{R}^{4\times 5}$$

$$A^2 = \begin{bmatrix} -\mathbf{u}_1- \\ -\mathbf{u}_2- \\ \cdots \\ -\mathbf{u}_5- \end{bmatrix} \in \mathbf{R}^{5\times 3}$$

$$A^1 = \begin{bmatrix} -\mathbf{w}_1- \\ -\mathbf{w}_2- \\ -\mathbf{w}_3- \end{bmatrix} \in \mathbf{R}^{3\times 5}$$

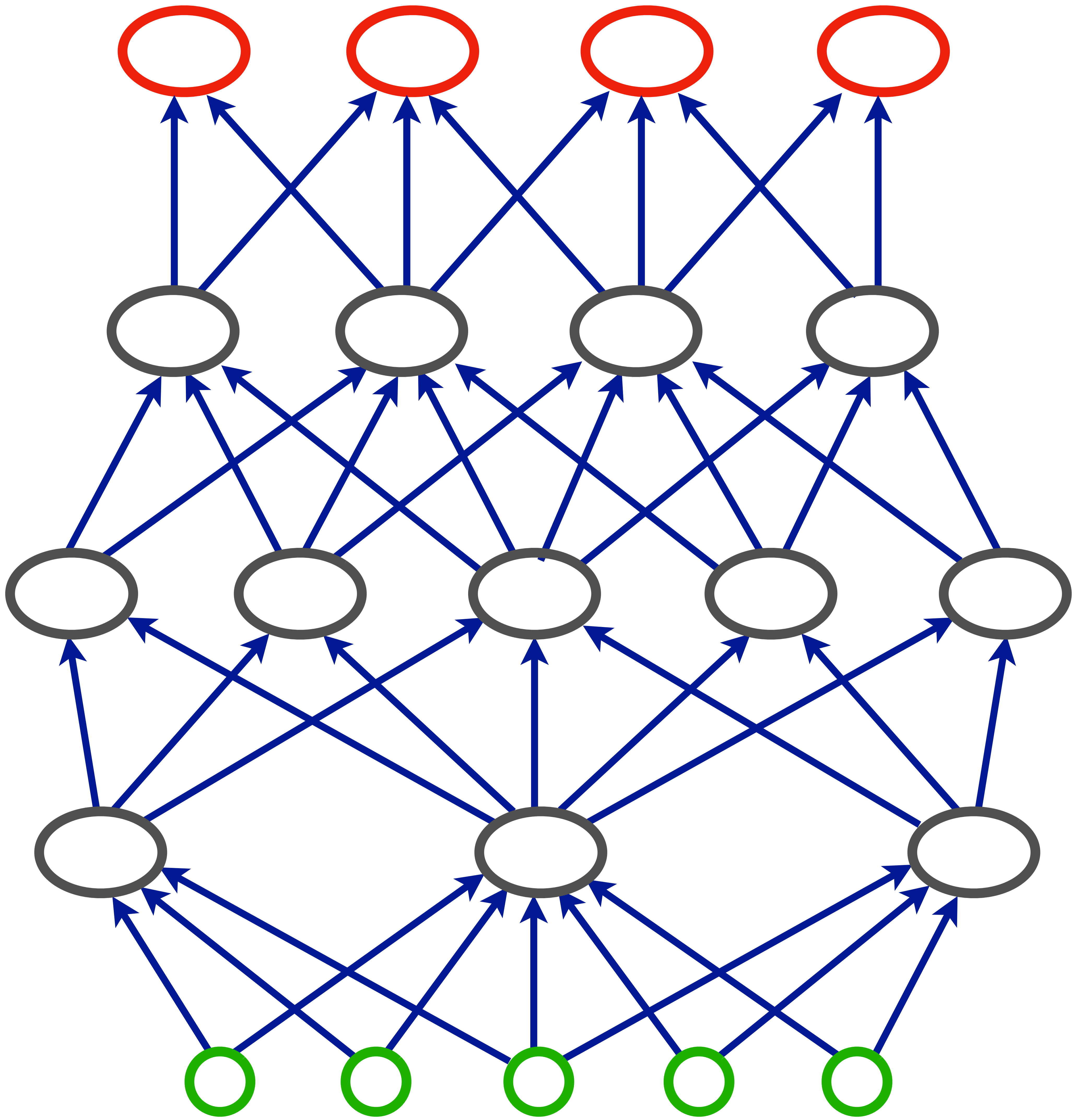
Multilayer Net

$$\mathbf{h}^4 = \sigma\left(\mathrm{A}^4\,\mathbf{h}^3\right)$$

$$\mathbf{h}^3 = \sigma\left(\mathrm{A}^3\,\mathbf{h}^2\right)$$

$$\mathbf{h}^2 = \sigma\left(\mathrm{A}^2\,\mathbf{h}^1\right)$$

$$\mathbf{h}^1 = \sigma\left(\mathrm{A}^1\,\mathbf{h}^0\right)$$

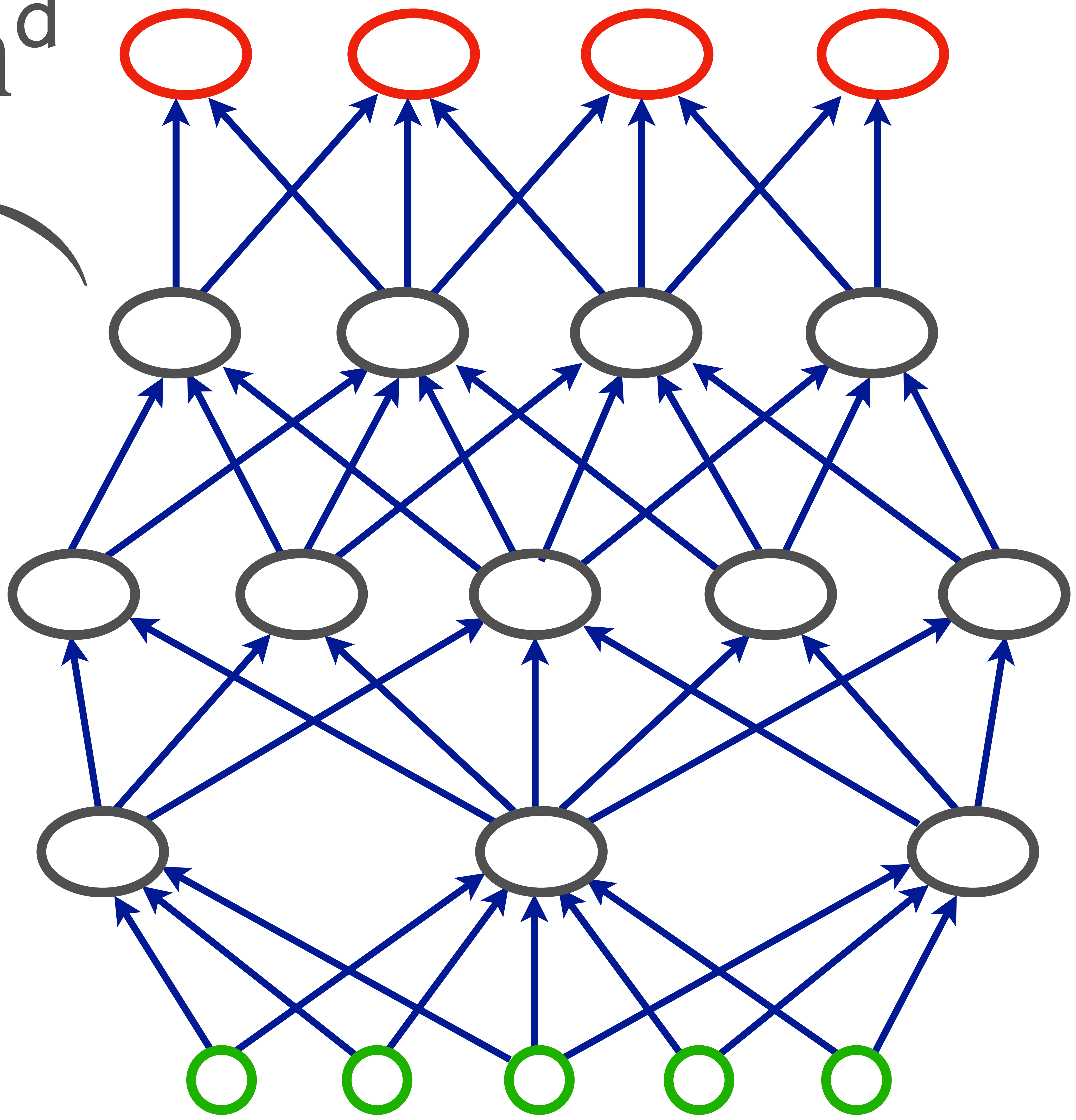


Inference aka Forward Pass

$$\mathbf{f}(\mathbf{x}) = \mathbf{h}^d$$

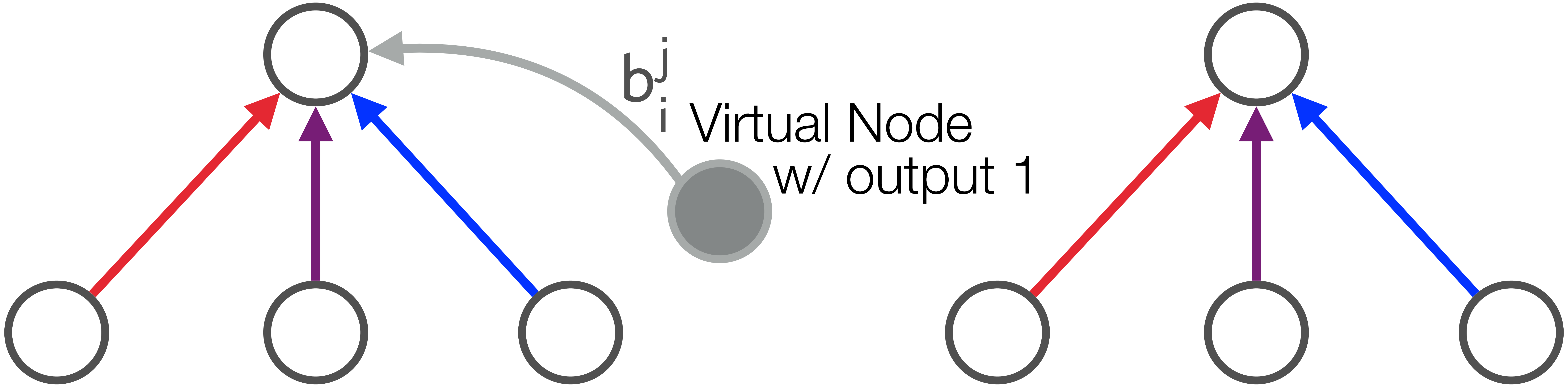
$$\mathbf{h}^i = \sigma\left(A^i \mathbf{h}^{i-1}\right)$$

$$\mathbf{h}^0 = \mathbf{x}$$



Adding Bias Vectors

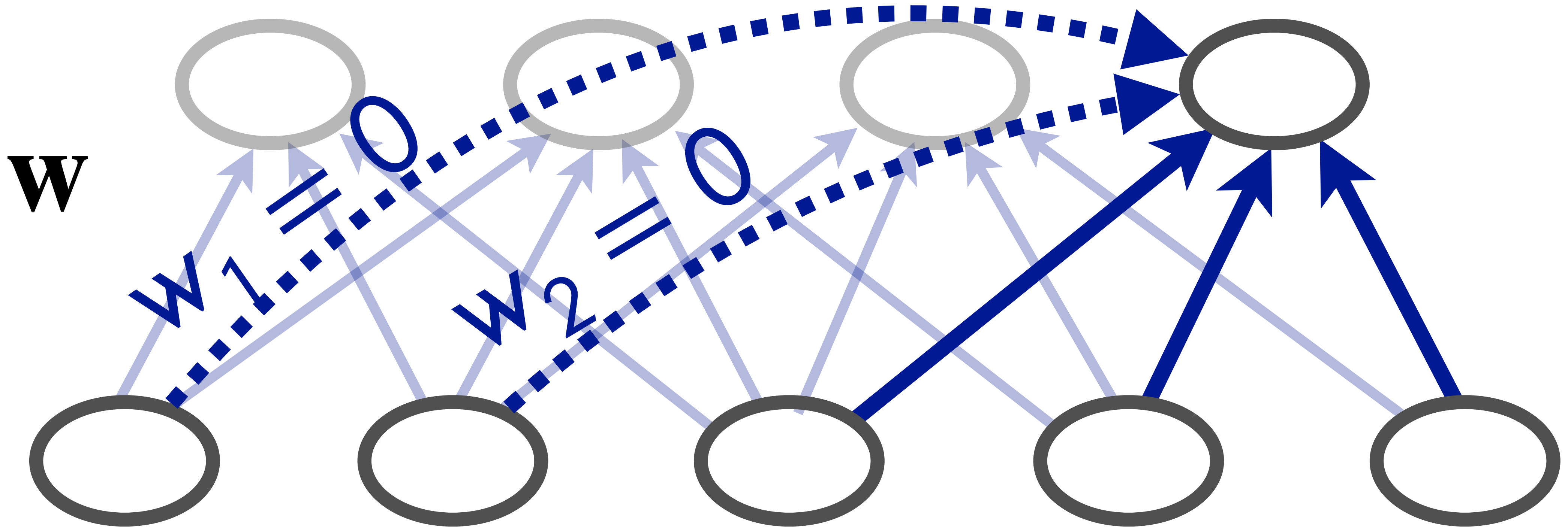
$h^j_i = \sigma\left(A^j_{\star i} \cdot \mathbf{h}^{j-1} + b^j_i\right)$ $\qquad$ $h^j_i = \sigma\left(A^j_{\star i} \cdot \mathbf{h}^{j-1}\right)$



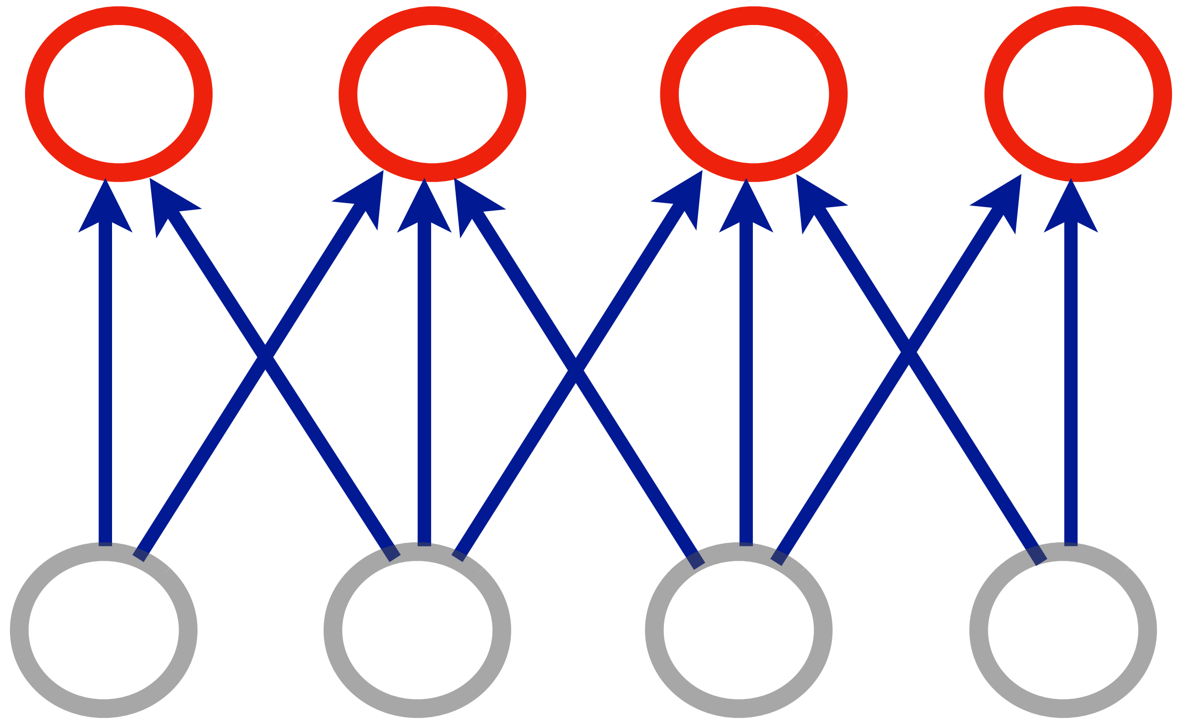
$$\mathbf{h}^j = \sigma\left(A^j \mathbf{h}^{j-1} + \mathbf{b}^j\right)$$

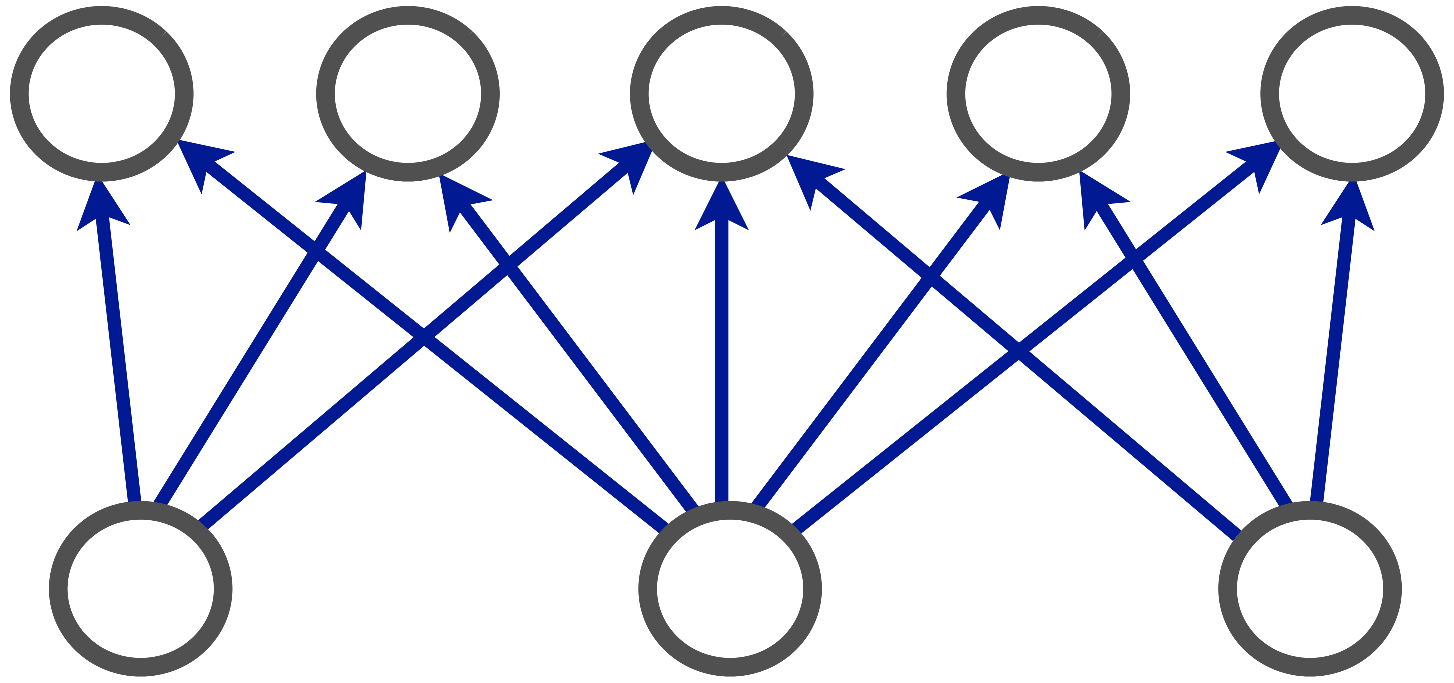
NN Architecture

Vector of weights of a single neuron $\mathbf{w}$
(incoming edges) can be sparse



Multiple Outputs $\mathbf{f}(\mathbf{x}) = \mathbf{h}^d$





#Neurons in each layer can be different

Feed-forward NN: Formal Definitions

- Obtained by connecting several neurons together
- **Feed-forward** networks: directed (acyclic) graph $G = (V, E)$ & denoting $m = |V|$
- Input nodes: nodes w/o incoming edges $v_1, \ldots, v_n$
- Output nodes: nodes w/o outgoing edges $v_{m-k+1}, \ldots, v_m$
- Weights associated with each edge $w : E \to R$
- Computation defined by NN [for (input) node $j \in [n]$ we define $h[v_j] = x_j$]

$$h[v] = \sigma\Big(\sum_{u \to v \in E} \mathbf{w}[u \to v]\, h[u]) \Big)$$

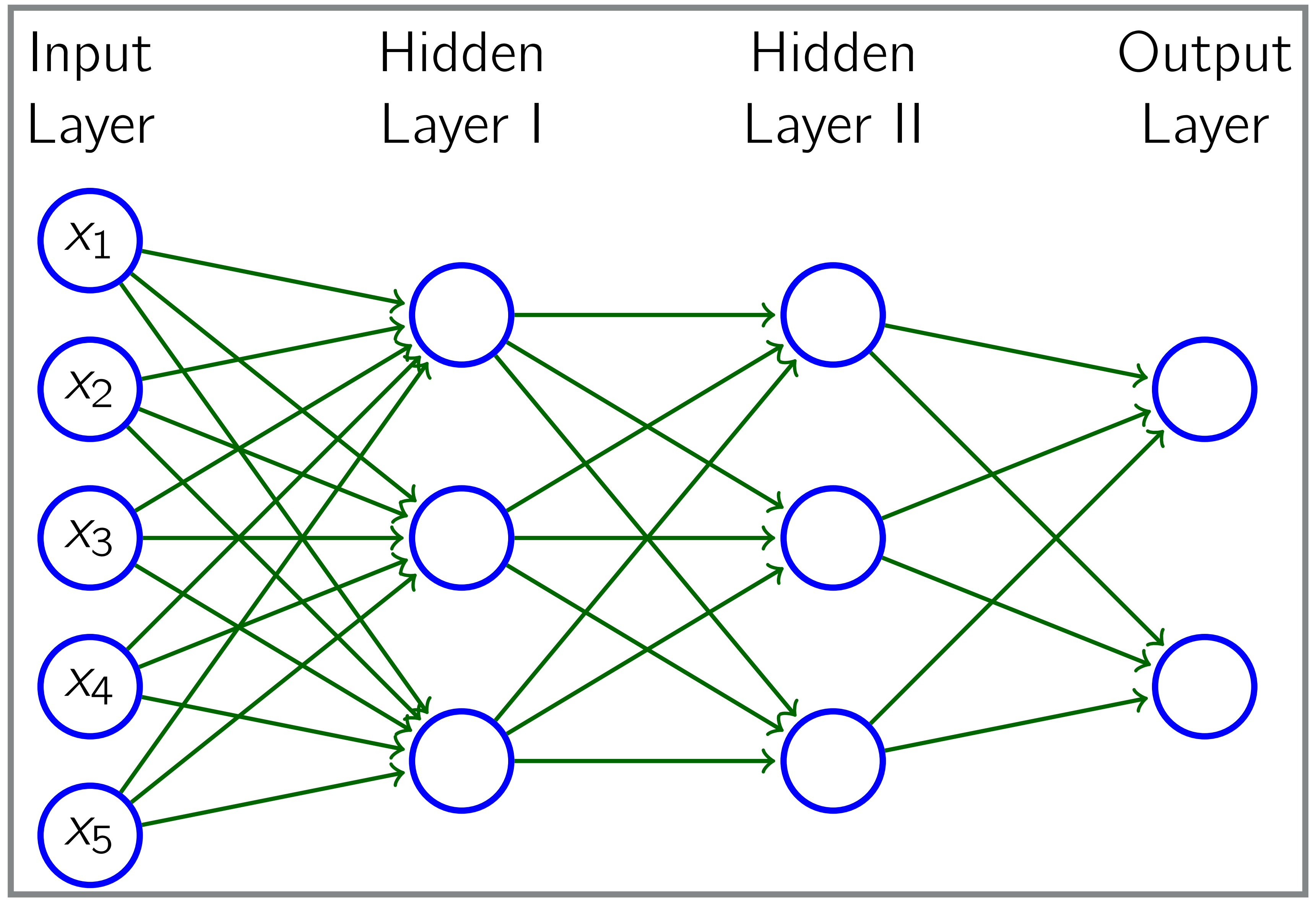
- Feed-forward NN defines a nonlinear mapping function $h : R^n \to R^k$

Multilayer Perceptron (MLP)

Nodes divided into layers: $V = \left\{ V_j \right\}_{j=0}^{d}$

Edges solely between adjacent layers:

$$u \to v \in E \;\; \Rightarrow \;\; \text{for } j \in [d] : u \in V_{j-1} \wedge v \in V_j$$



#Inputs	n=5
Depth	d = 3
#Neurons	m = 5 + 3 + 3 + 2 = 13
#Weights	|E| = 3x5 + 3x3 + 2x3 = 30

Multilayer Perceptron

Nodes divided into layers: $V = \left\{V_j\right\}_{j=1}^{d}$

Edges solely between adjacent layers: $u \to v \in E \;\Rightarrow\; u \in V_j \wedge v \in V_{j+1}$

Define $A^j \in \mathbf{R}^{|V_{j-1}| \times |V_j|}$ where

$$A^j[\mathscr{I}(v_a), \mathscr{I}(v_b)] = w[v_a \to v_b] \text{ and } \mathscr{I}(v_a) = a - \sum_{i<j} |V_i|$$

Then for $j \in [d]$: $\mathbf{h}^j = \sigma\left(A^j \mathbf{h}^{j-1}\right)$

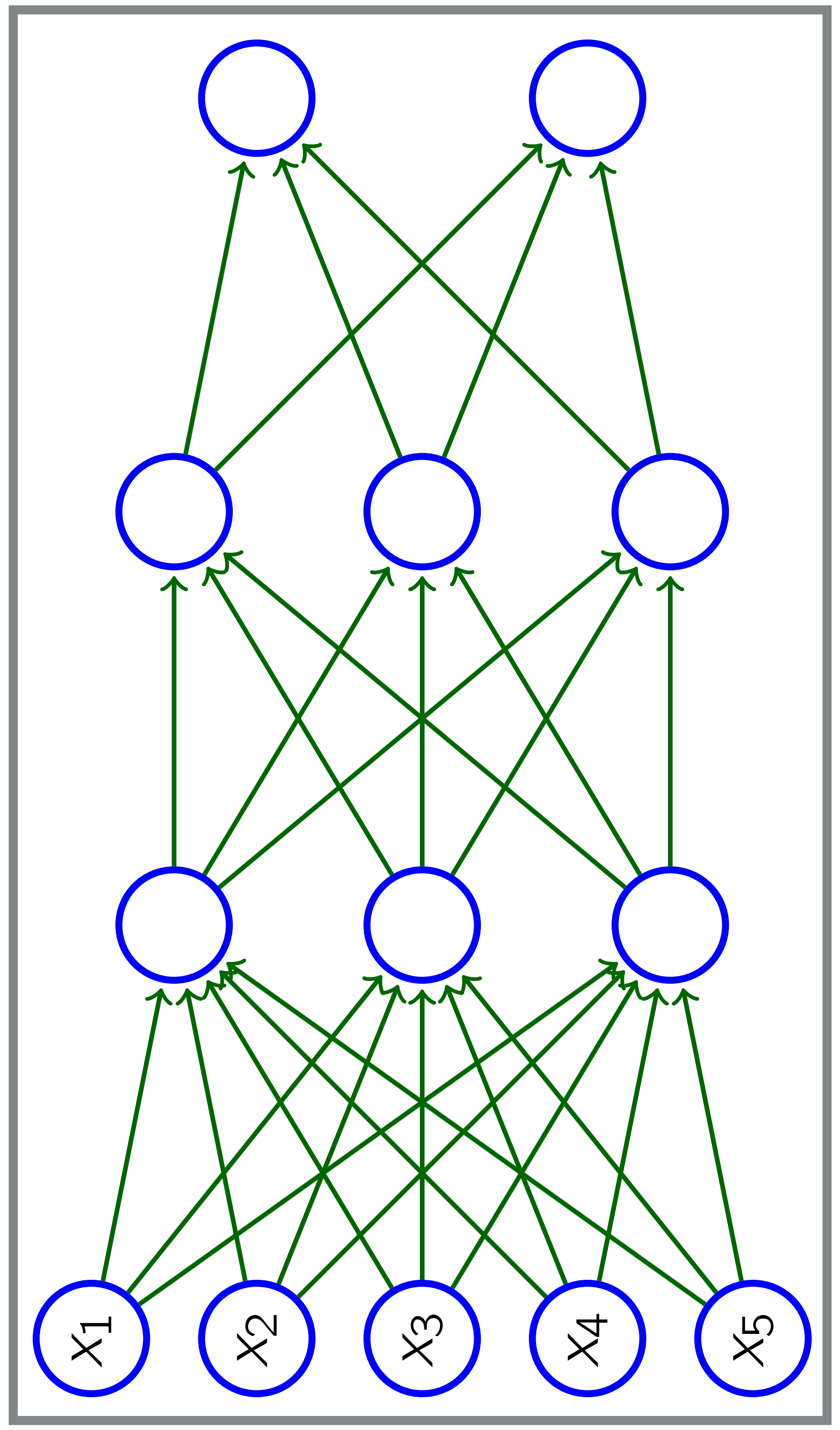
```python
import numpy as np

def rand_matrix(m, n):
    return np.random.randn(m, n) / np.sqrt(m)

def MLP(dims):
  As, Bs = [], []
  for i in range(len(dims)-1):
    As.append(rand_matrix(dims[i], dims[i+1]))
    Bs.append(rand_matrix(1, dims[i+1]))
  return (As, Bs)

def predict(net, X, act):
  As, Bs = net
  H = X
  for cd in range(len(As)):
    H = act(H @ As[cd] + Bs[cd])
  return H

dims = [5, 3, 3, 2]
net = MLP(dims)
X = rand_matrix(dims[0], 10).transpose()
O = predict(net, X, np.tanh)
```



Training Neural Networks (Deep Learning)

- MLP defines a non-convex function f($\mathbf{x}$)
- Even if loss $\ell(\mathrm{f}(\mathbf{x}), \mathrm{y})$ is convex E.R.M is non-convex
- Initialization is important: symmetry breaking, scale sensitive [more later]
- Gradient-based training takes into account NN's architecture
- Back-propagation: efficient way to calculate gradients using chain rule

$$\nabla_{A_j} \ell(\mathrm{f}(\mathbf{x}), \mathrm{y})$$

- Inference & gradient as computations on a graph
- Despite long time to train, often yields good results, many tricks-of-the-trade

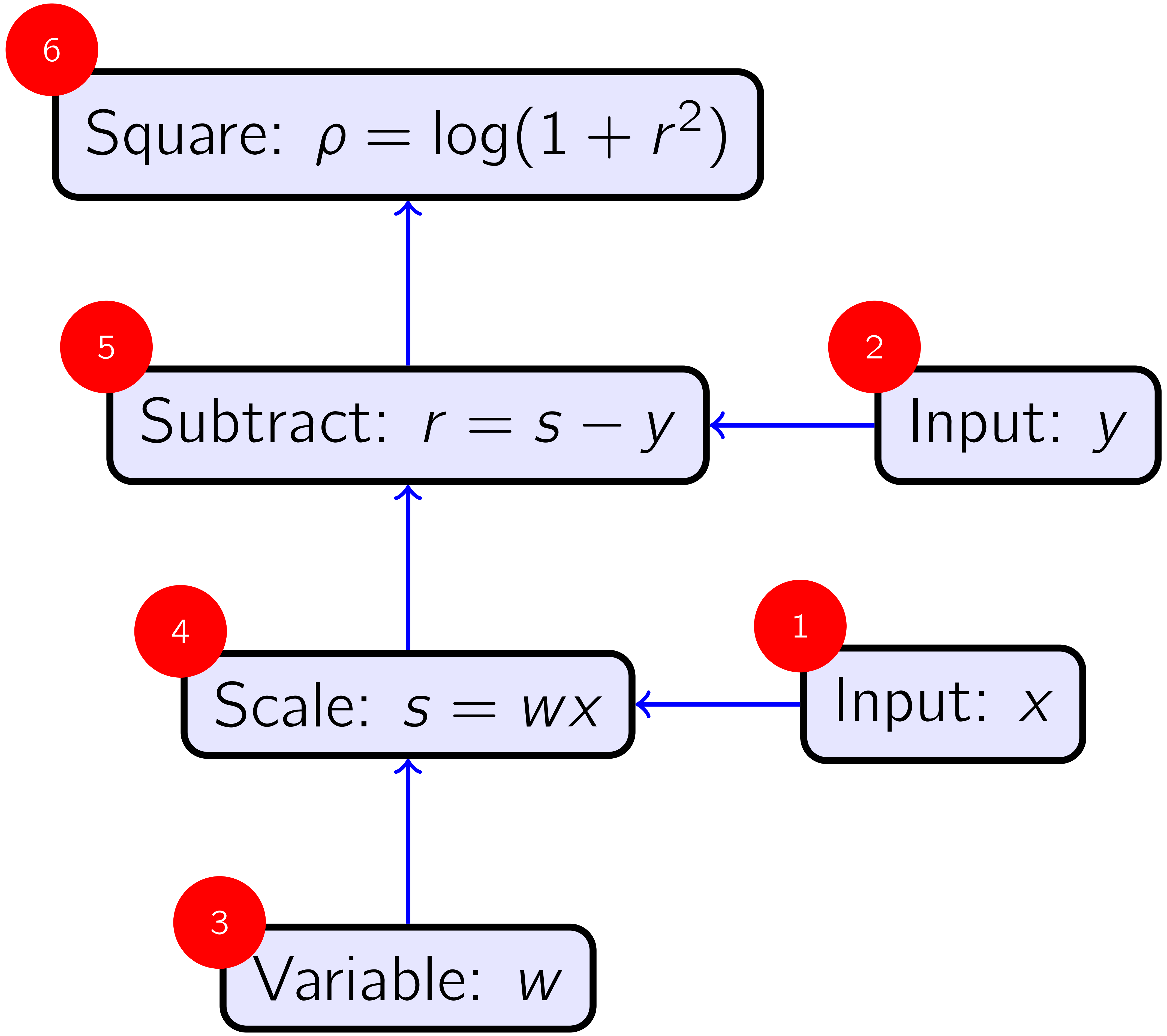
Computation Graph

One dimensional problem

Nodes sorted topologically

Loss: $\ell(\mathrm{y}, \hat{\mathrm{y}}) = \log(1 + (\hat{\mathrm{y}} - \mathrm{y})^2)$

with $\hat{\mathrm{y}} = \mathrm{wx}$ and $\mathrm{x}, \mathrm{y}, \mathrm{w} \in \mathbf{R}$



Derivative Using Chain Rule

Fix x, y and define functions:

$$\rho(z) = \log(1 + z^2)$$
$$r_y(z) = z - y$$
$$s_x(z) = xz$$

Write ℓ as a function of w:

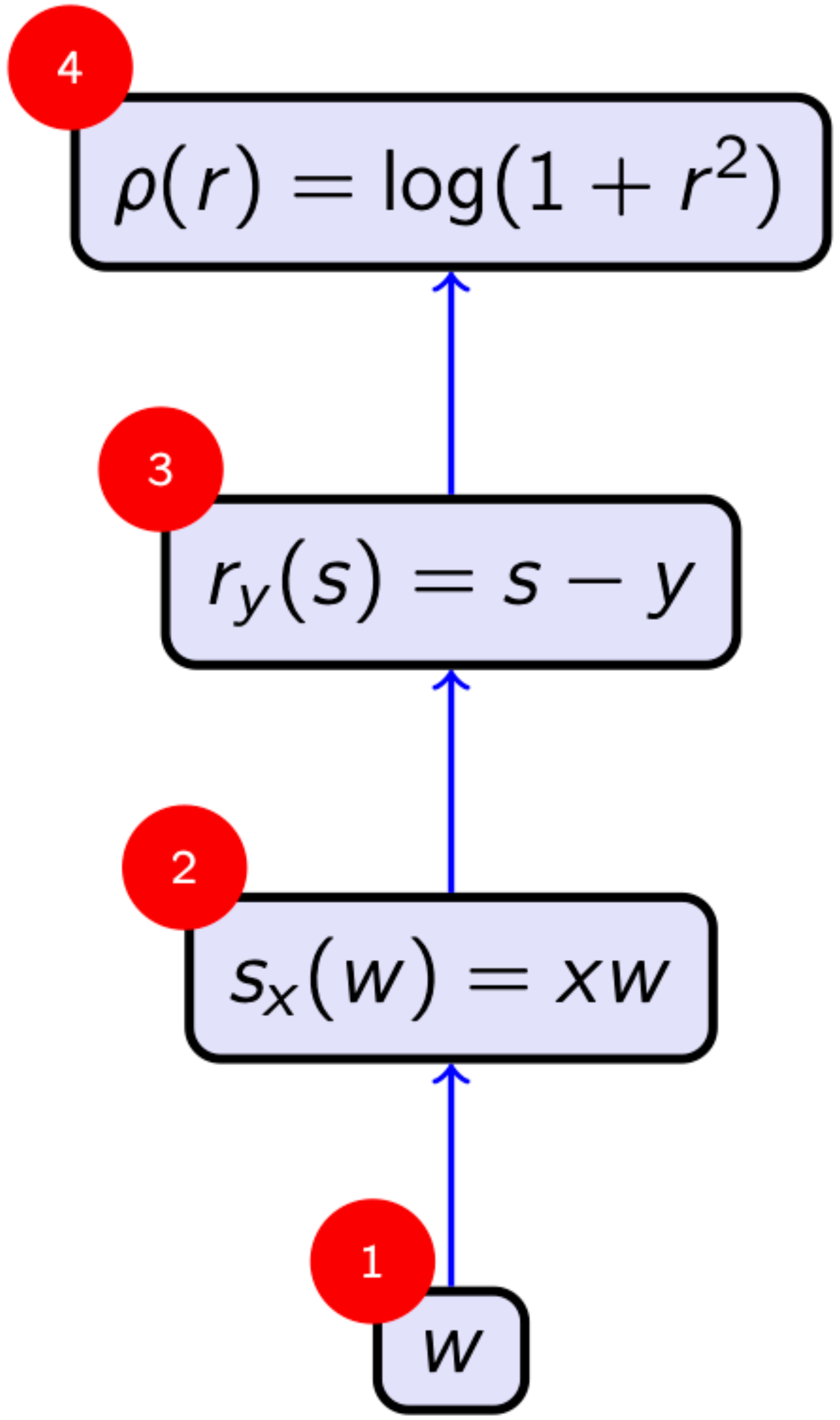
$$\ell(w) = \rho(r_y(s_x(w))) = (\rho \circ r_y \circ s_x)(w)$$

Chain rule:

$$\begin{aligned} \ell'(w) &= (\rho \circ r_y \circ s_x)'(w) \\ &= \rho'(r_y(s_x(w))) \cdot (r_y \circ s_x)'(w) \\ &= \rho'(r_y(s_x(w))) \cdot r_y'(s_x(w)) \cdot s_x'(w) \end{aligned}$$

Inference: Forward Pass

For v = 1, . . . , m:
    v.output = v.op(v.inputs())

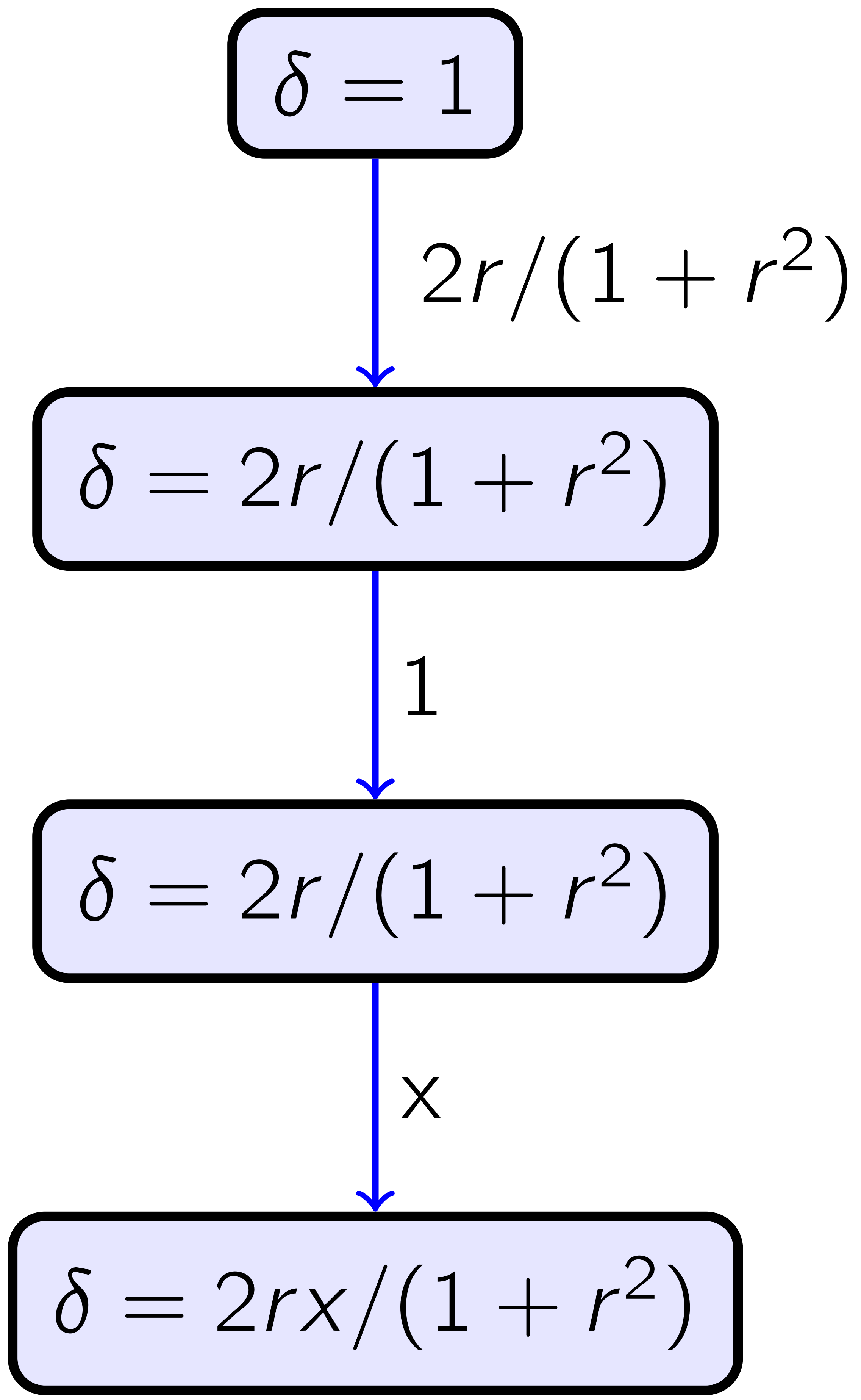


Gradient: Backward Pass (BackProp)

m->delta = 1

Foreach u s.t. u→v ∈ E:

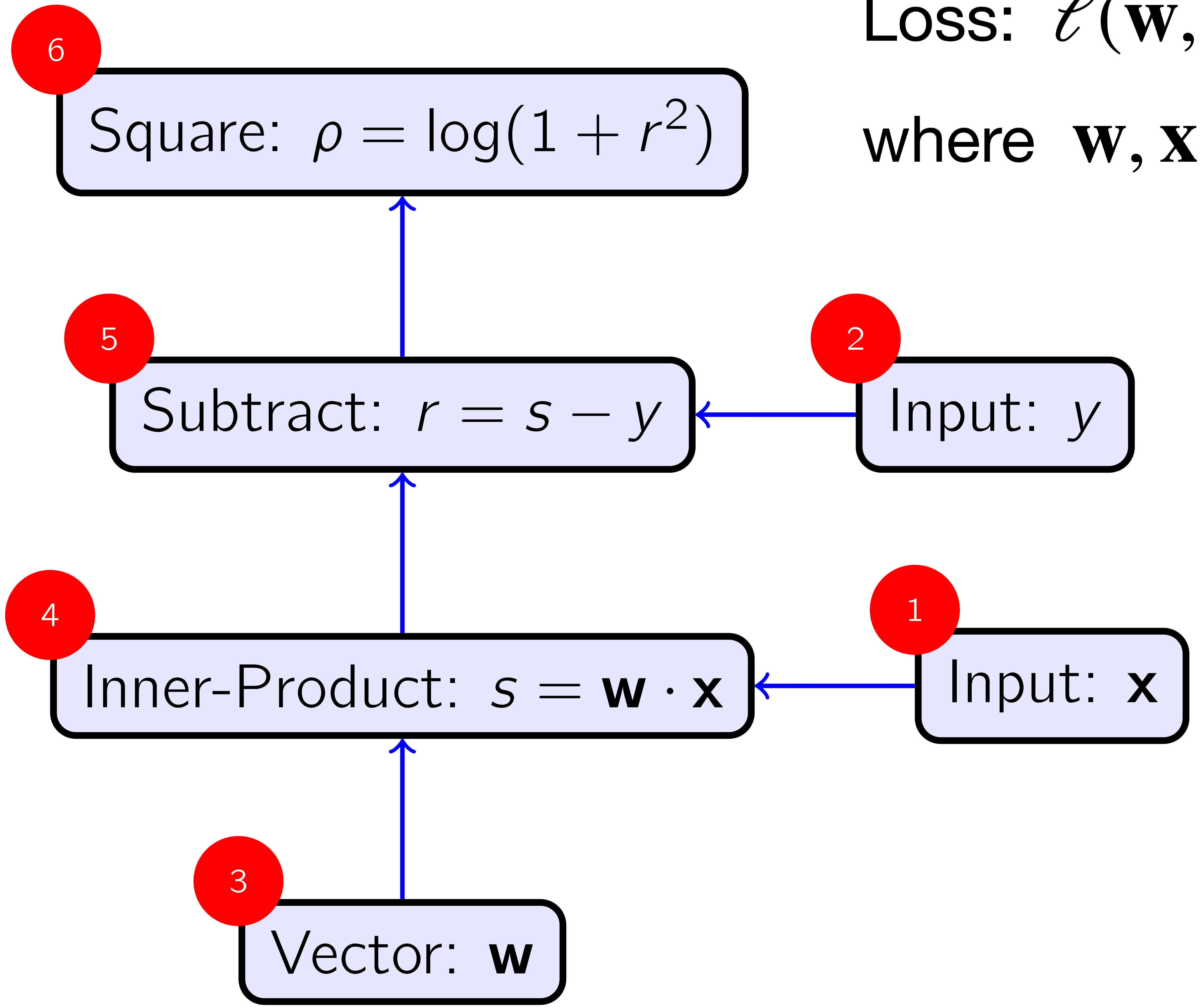
    u->delta = v->delta * v->deriv(u)



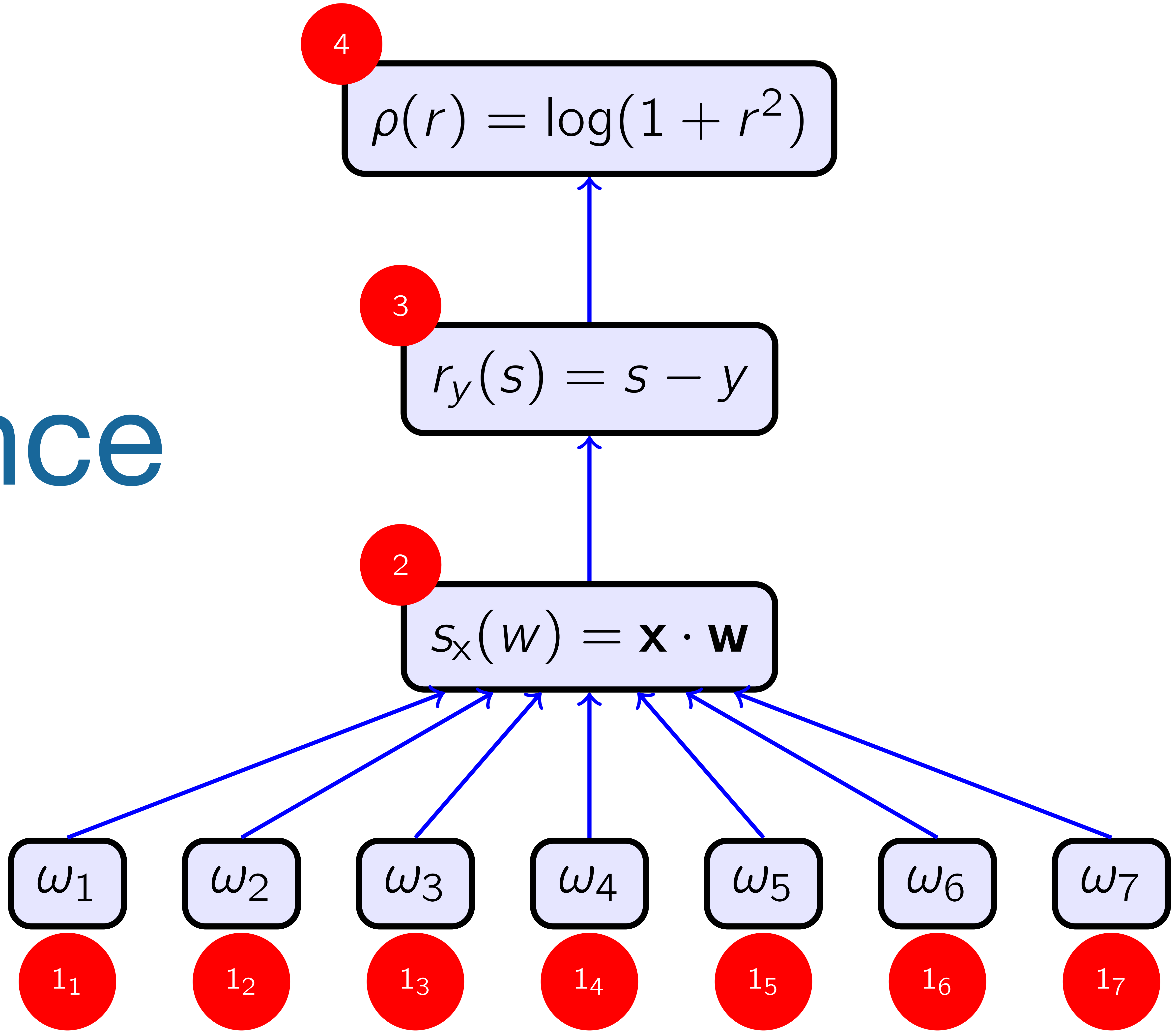
From Scalars to Neurons

Loss: $\ell(\mathbf{w},(\mathbf{x},y)) = \log\left(1+(\mathbf{w}\cdot\mathbf{x}-y)^2\right)$

where $\mathbf{w},\mathbf{x}\in\mathbf{R}^d \quad y\in\mathbf{R}$

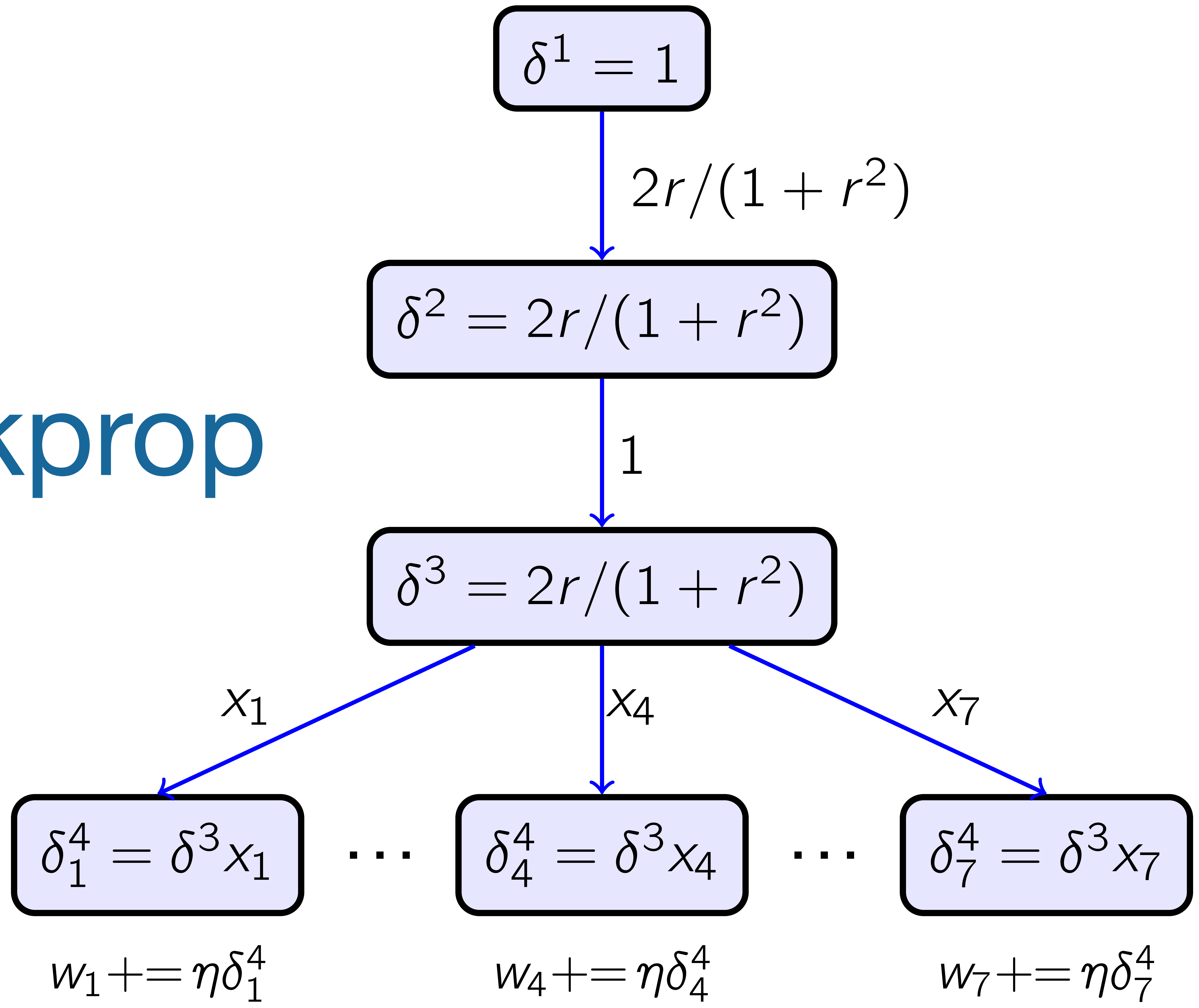


Inference



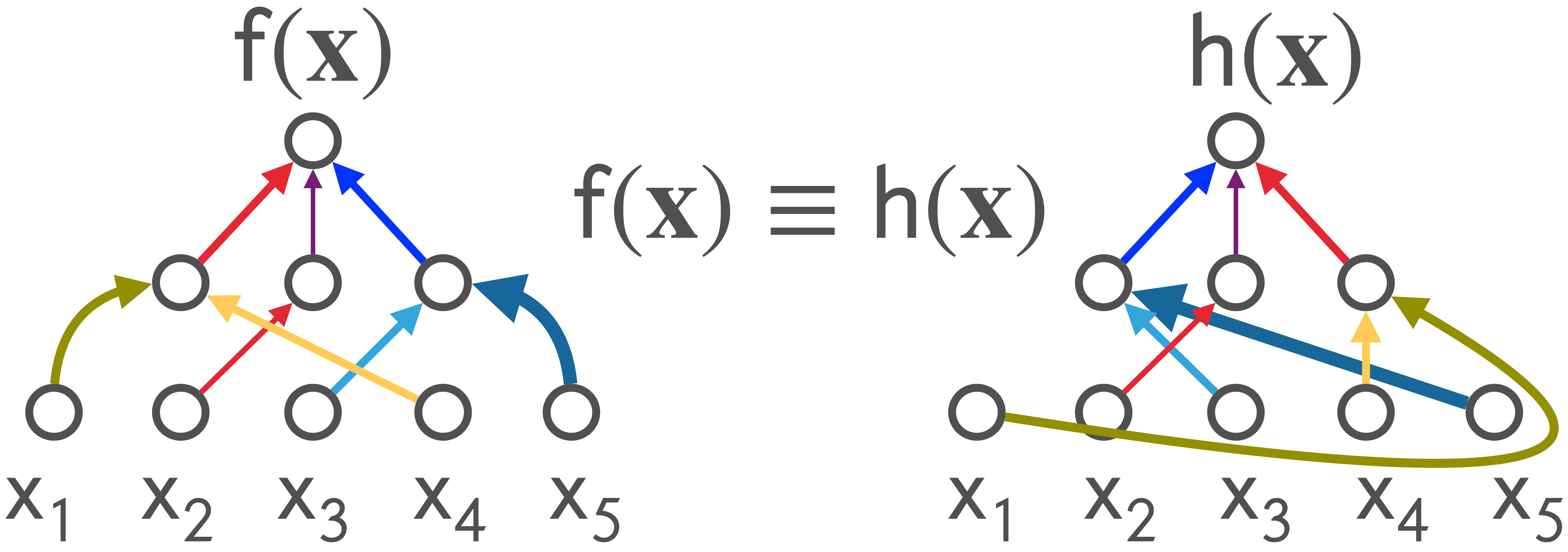
4
$\rho(r) = \log(1 + r^2)$
3
$r_y(s) = s - y$
2
$s_x(w) = \mathbf{x} \cdot \mathbf{w}$
ω_1
ω_2
ω_3
ω_4
ω_5
ω_6
ω_7
1_1
1_2
1_3
1_4
1_5
1_6
1_7

Backprop



$\delta^1 = 1$
$2r/(1+r^2)$
$\delta^2 = 2r/(1+r^2)$
1
$\delta^3 = 2r/(1+r^2)$
x_1
x_4
x_7
$\delta^4_1 = \delta^3 x_1$
$\cdots$
$\delta^4_4 = \delta^3 x_4$
$\cdots$
$\delta^4_7 = \delta^3 x_7$
$w_1 {+}{=} \eta \delta^4_1$
$w_4 {+}{=} \eta \delta^4_4$
$w_7 {+}{=} \eta \delta^4_7$

Symmetries

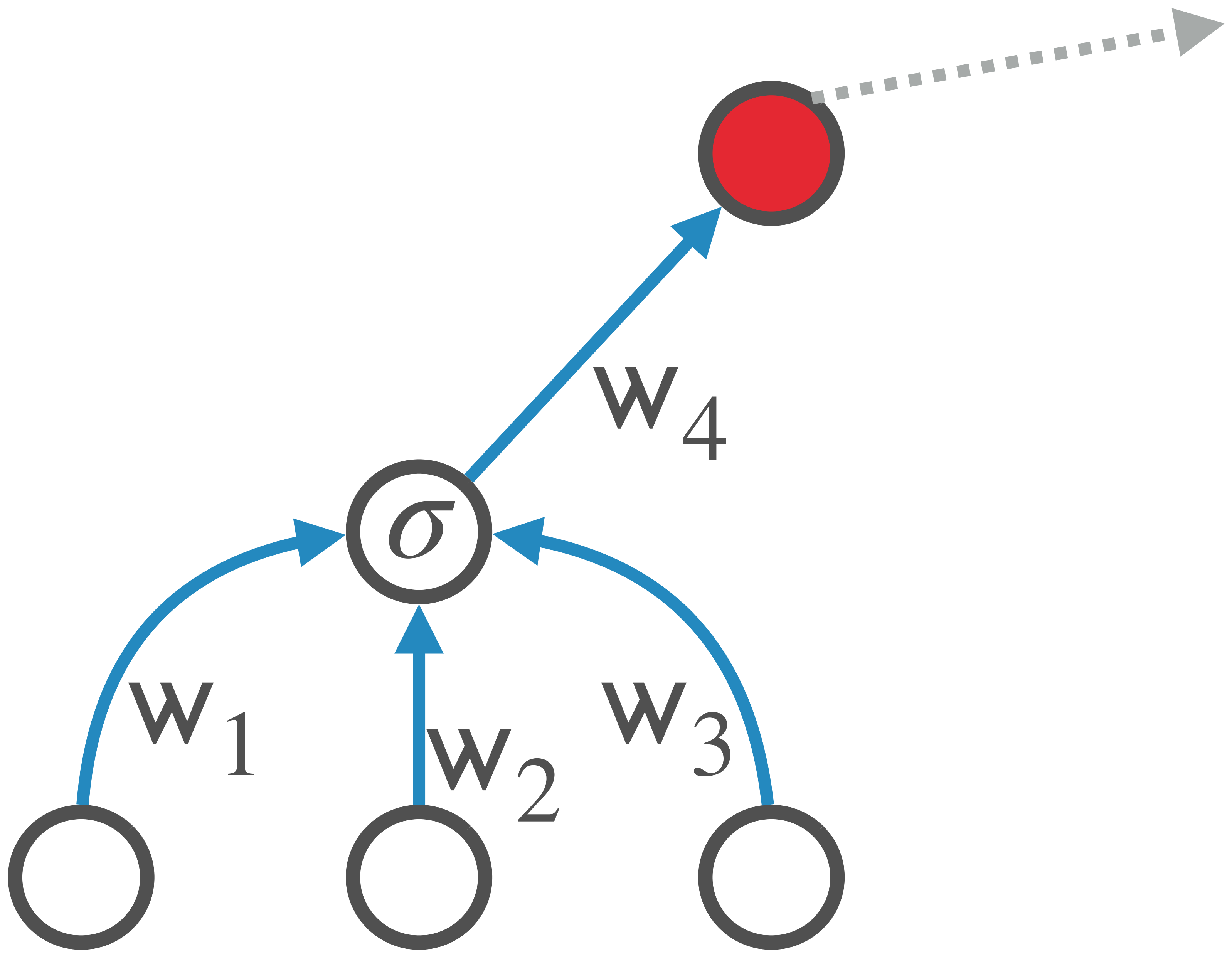


- Input presented in the same order
- Permuting edges and nodes creates seemingly different graphs which are functionally the same

Degrees of Freedom

- Assume $\sigma(z) = [z]_+$
- Input to **red-neuron:**

$$w_4\ \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3)$$



- Scale : $w_4 \mapsto \dfrac{w_4}{\alpha}$ and also

$$w_1 \mapsto \alpha w_1 \quad w_2 \mapsto \alpha w_2 \quad w_3 \mapsto \alpha w_3$$

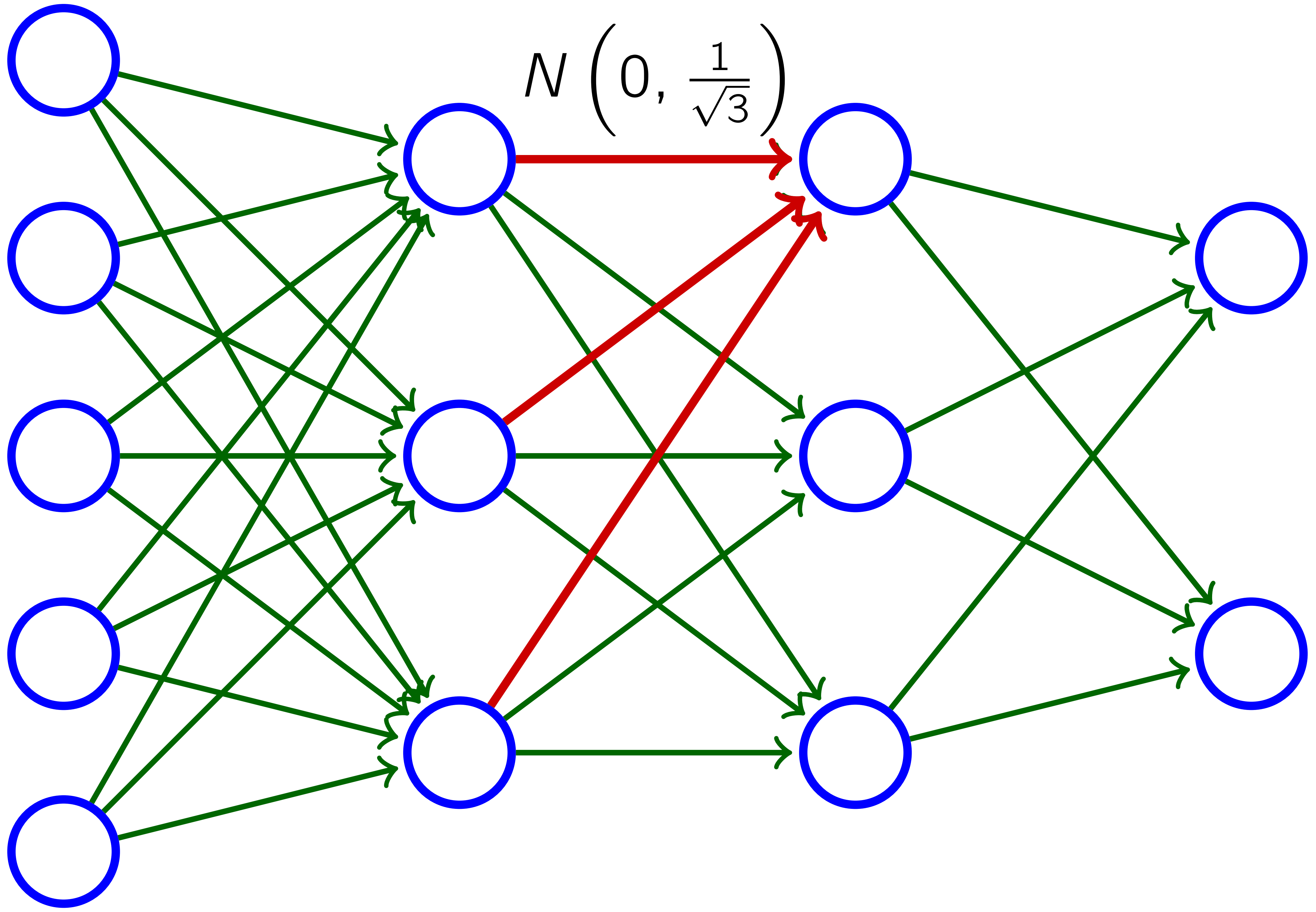
- Input to **red-neuron** remains the same

Weight Initialization

To break symmetry and d.o.f.

$$w[u \rightarrow v] \sim \mathcal{N}\left(0\,, \mathrm{in}(v)^{-\frac{1}{2}}\right)$$

where $\mathrm{in}(v) = \left|\{u : u \rightarrow v \in E\}\right|$



SGD for NN

Design an MLP architecture suitable for problem:

- Number of layers
- Hidden neurons in each layer
- Connectivity between layers
- Activation function

Define loss function between $\mathbf{y}$ and $\hat{\mathbf{y}} = \mathrm{MLP}(\mathbf{x})$

Initialize weights of MLP

Loop:

Obtain a mini-batch of examples

Perform **Inference** for each example

Perform **Backprop** for each example

Calculate average (over mini-batch) gradient

Update weights

Back-propagation