Neural Networks

COMP3314 — Lecture 5

Lingpeng Kong
Department of Computer Science, The University of Hong Kong

Based on: Probabilistic Machine Learning by Kevin Murphy
Slides from: Saw Shier Nee with special thanks!

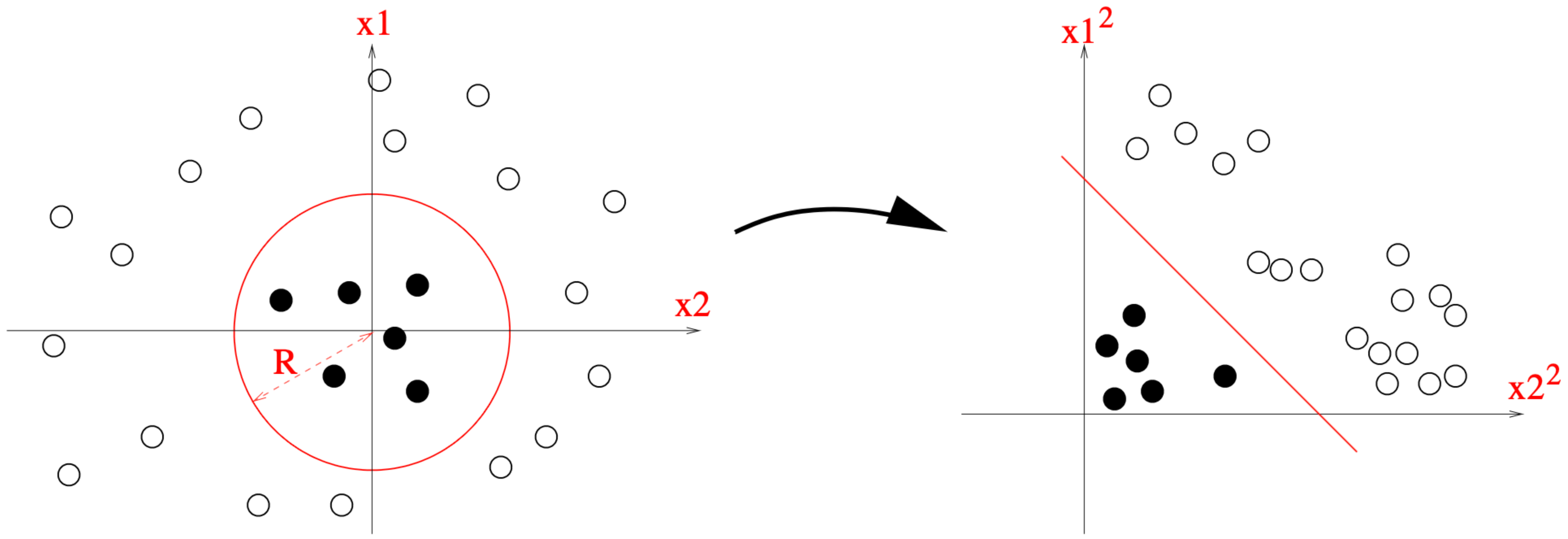
Basis Function Expansion

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathrm{Ber}(y|\boldsymbol{\sigma}(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b))$$

Logistic Regression

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}$$

$$\boldsymbol{\phi}(x) = [1, x, x^2, x^3, \ldots]$$



Basis Function Expansion

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \mathbf{W}\phi(\boldsymbol{x}) + \boldsymbol{b} \qquad \phi(x) = [1, x, x^2, x^3, \ldots]$$

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \mathbf{W}\phi(\boldsymbol{x};\boldsymbol{\theta}_2) + \boldsymbol{b}$$

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x}))\cdots)) \qquad f_\ell(\boldsymbol{x}) = f(\boldsymbol{x};\boldsymbol{\theta}_\ell)$$

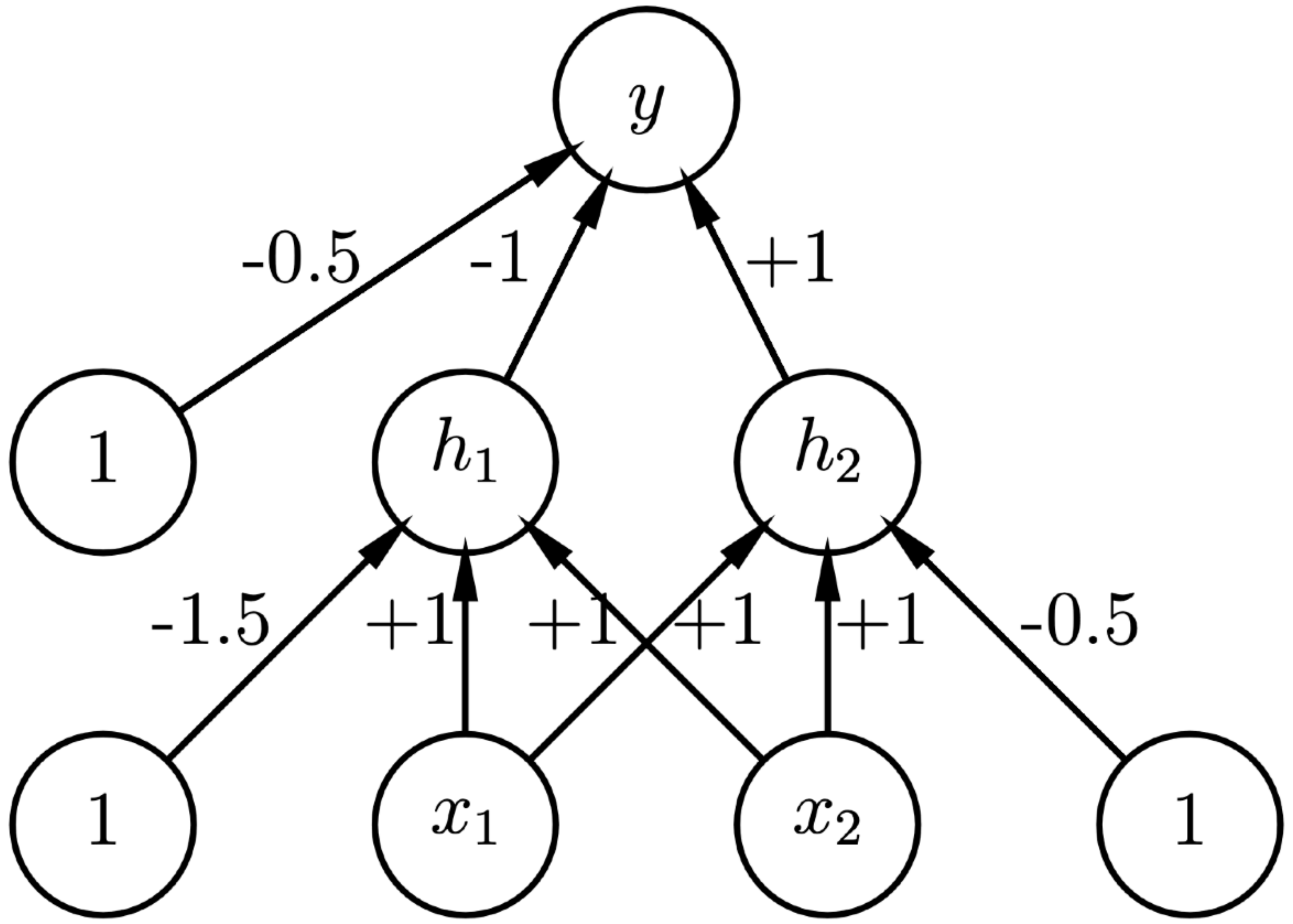
Deep Neural Networks (DNNs)

Multilayer Perceptrons (MLPs)

perceptron: a deterministic version of logistic regression

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \mathbb{I}\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b \geq 0\right) = H(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)$$

heaviside step function / linear threshold function



x1	x2	XOR(x1, x2)
0	0	0
0	1	1
1	0	1
1	1	0

Multilayer Perceptrons (MLPs)

Differentiable MLPs

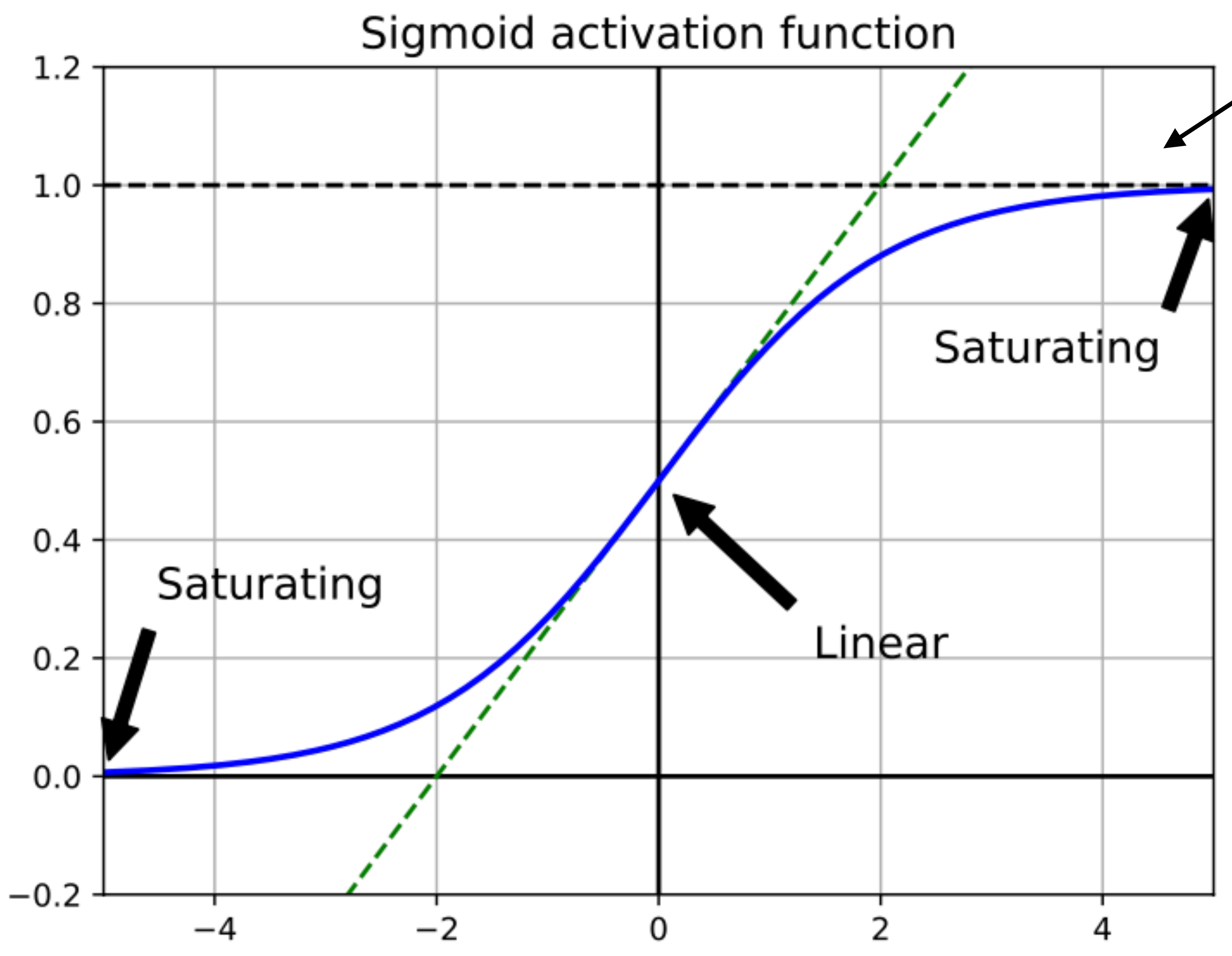
$$\boldsymbol{z}_l = f_l(\boldsymbol{z}_{l-1}) = \varphi_l\left(\boldsymbol{b}_l + \mathbf{W}_l \boldsymbol{z}_{l-1}\right)$$

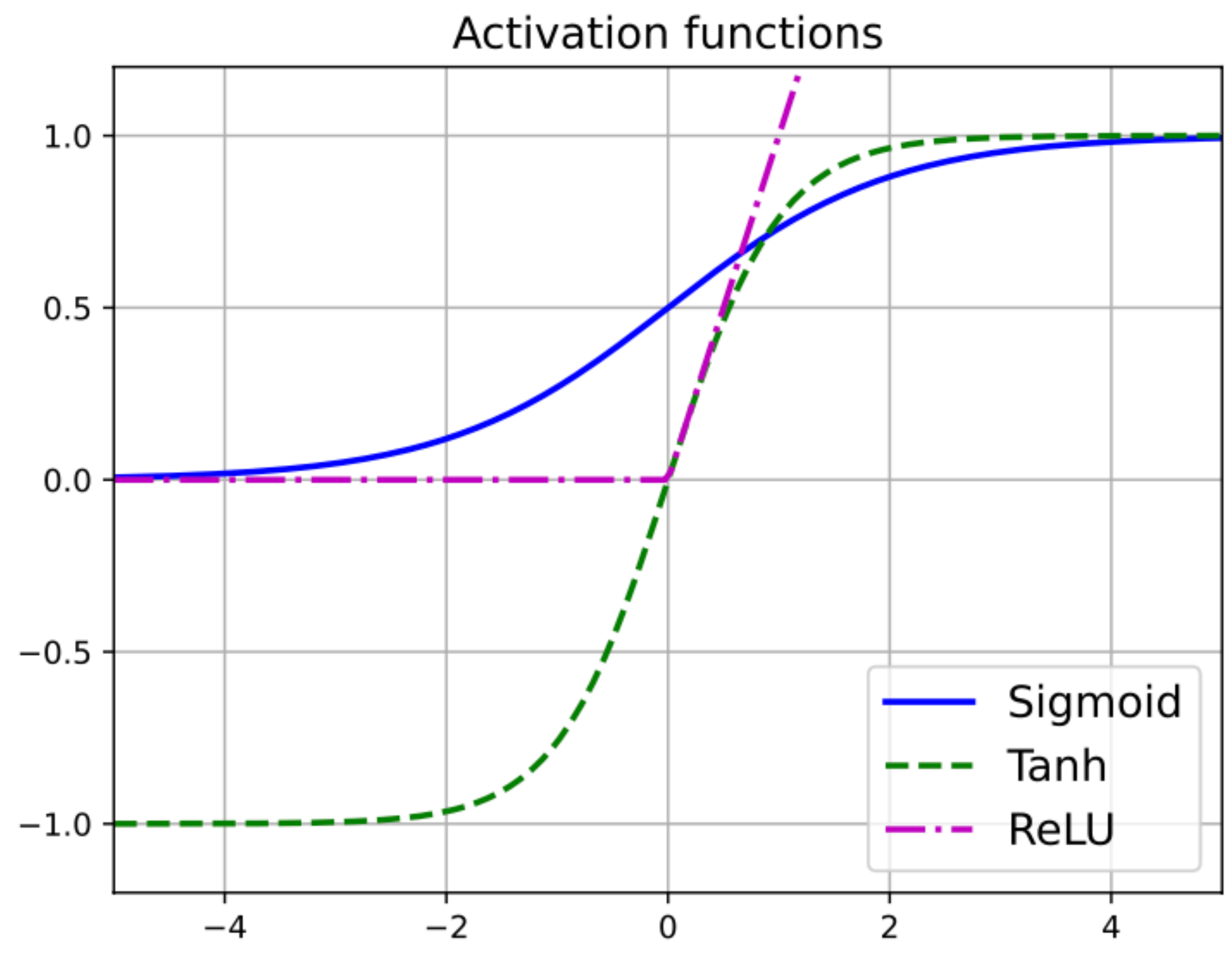
Activation function

What's bad here?

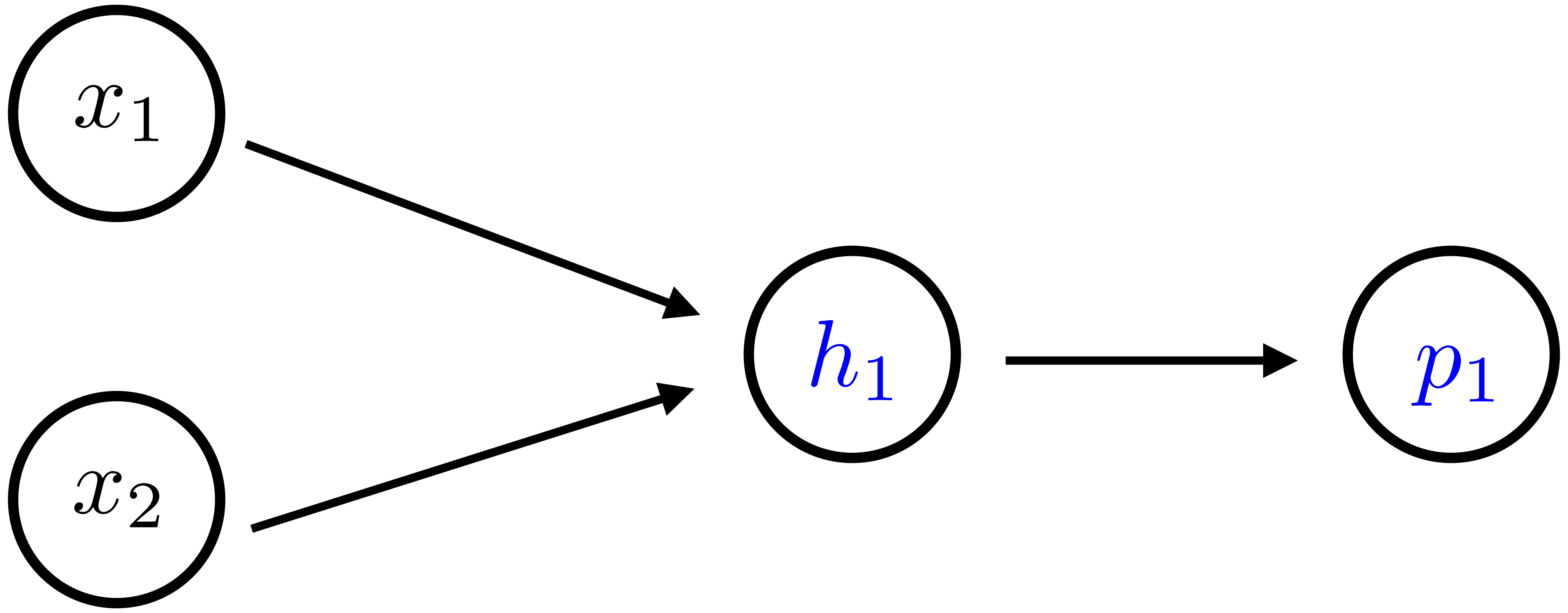
$$\mathrm{ReLU}(a) = \max(a, 0) = a\mathbb{I}\left(a > 0\right)$$

(ReLU: rectified linear unit)





Computational Graphs

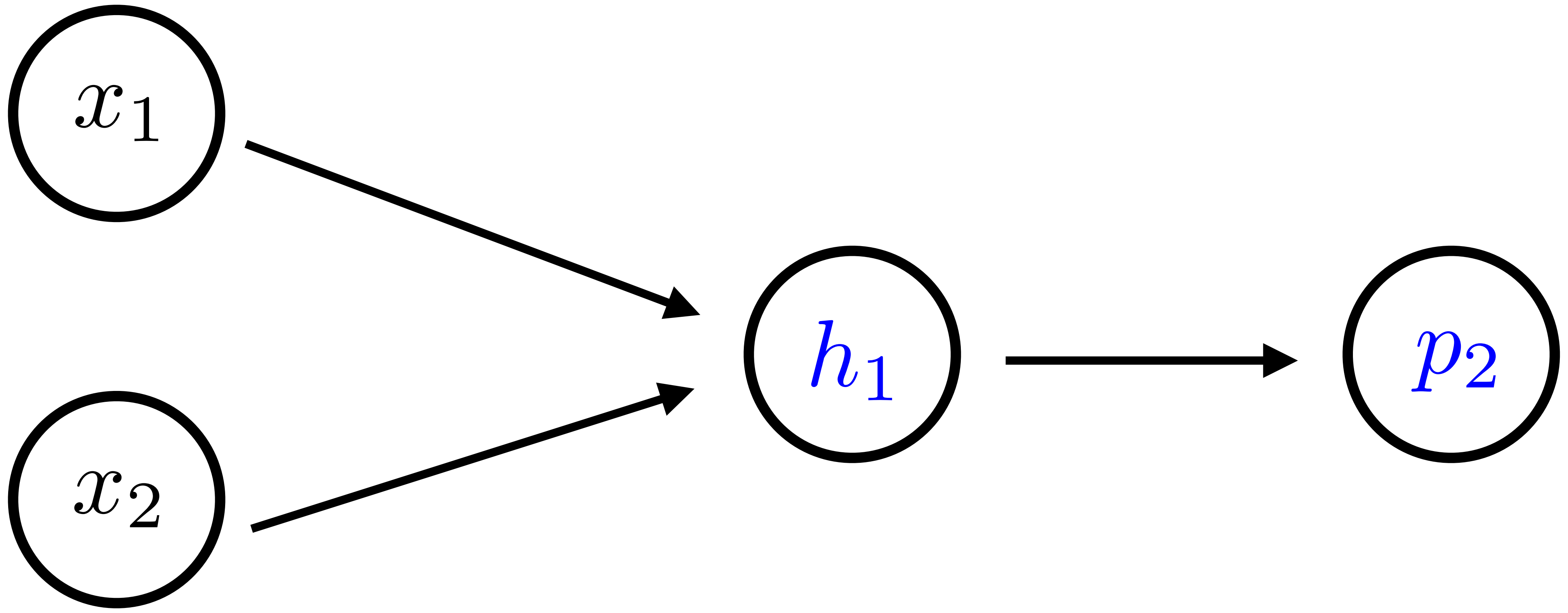


$$p_1 := p(y = 1 \mid x_1, x_2)$$

$$h_1 = w_1 x_1 + w_2 x_2 + b$$

$$p_1 = \frac{1}{1 + \exp(-h_1)}$$

Logistic Regression



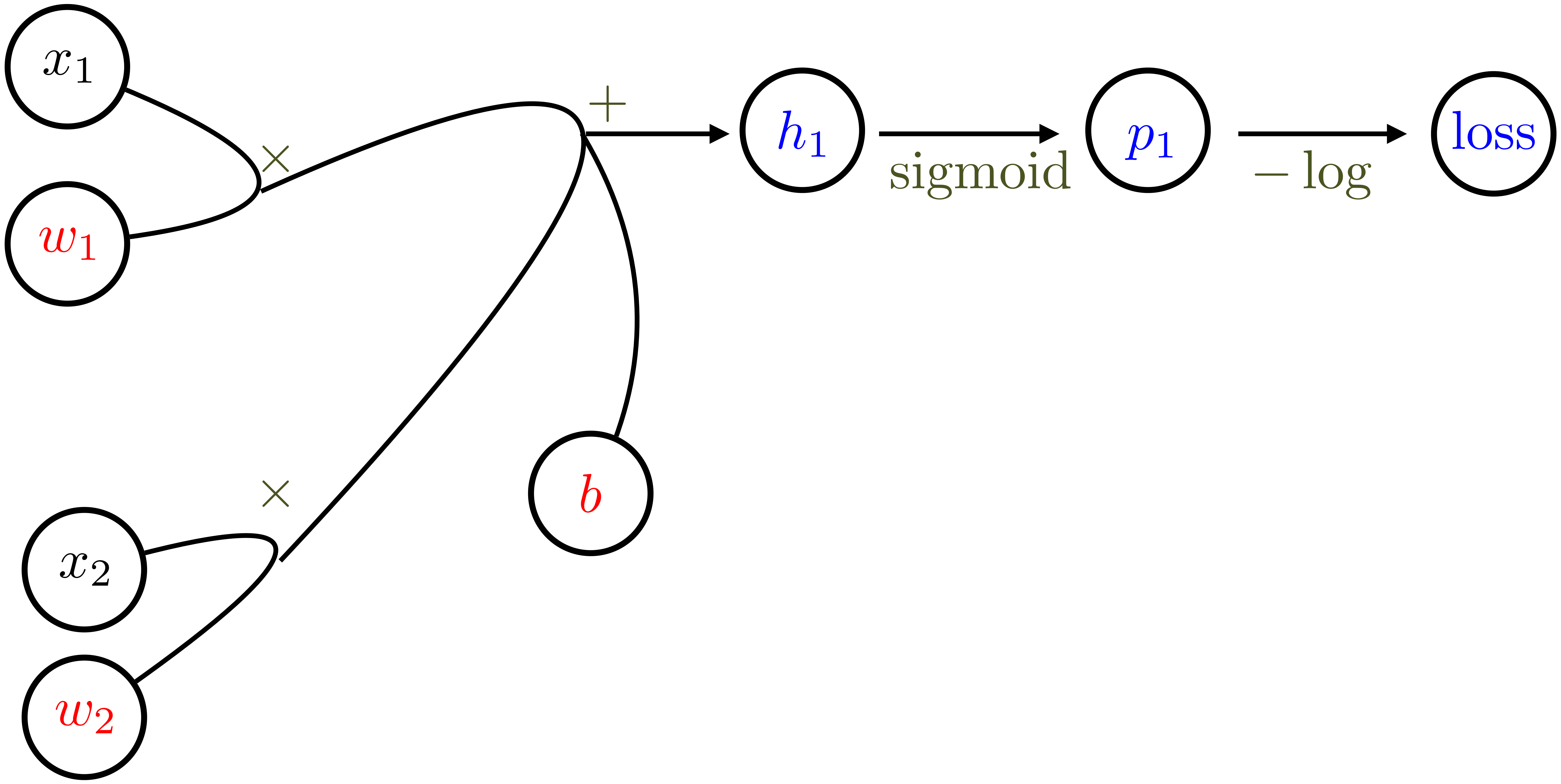
$$p_2 := p(y = 0 \mid x_1, x_2)$$

$$h_1 = w_1 x_1 + w_2 x_2 + b$$

$$p_2 = 1 - \frac{1}{1 + \exp(-h_1)} = \frac{\exp(-h_1)}{1 + \exp(-h_1)}$$

Loss Function

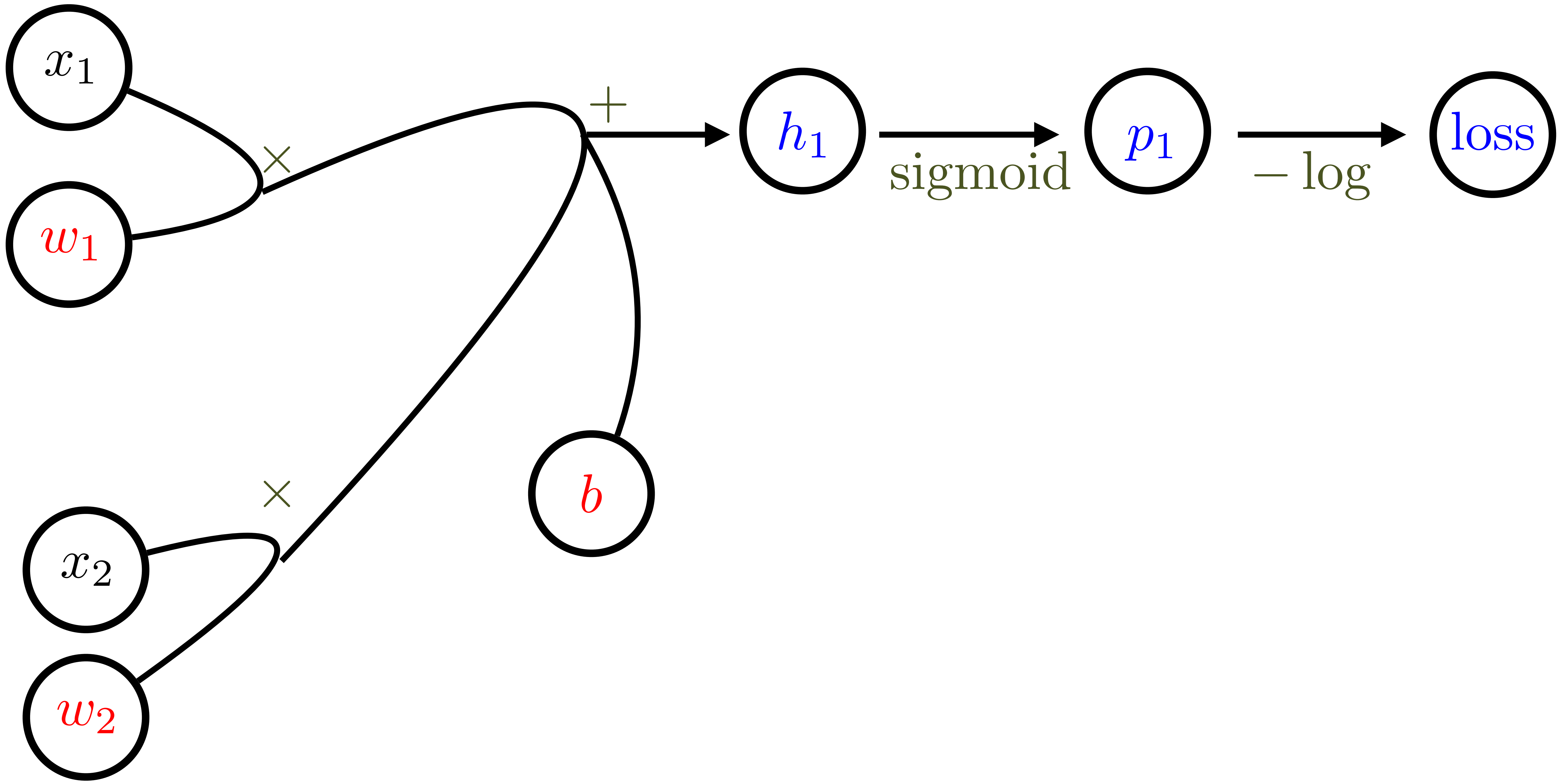
case $y = 1$:



Computational Graphs

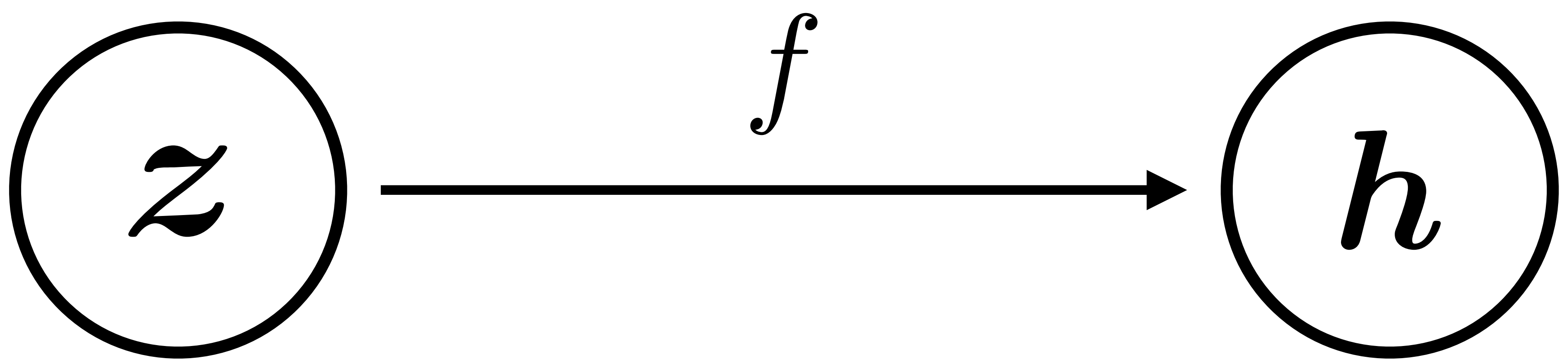
Input	x_1	x_2		
Parameter	w_1	w_2		
Expression	h_1	p_1	loss	
Operation	$\times$	$+$	sigmoid	$-\log$

Special



How to minimize? (Automatic Differentiation)

$$\boldsymbol{h} = f(\boldsymbol{z})$$



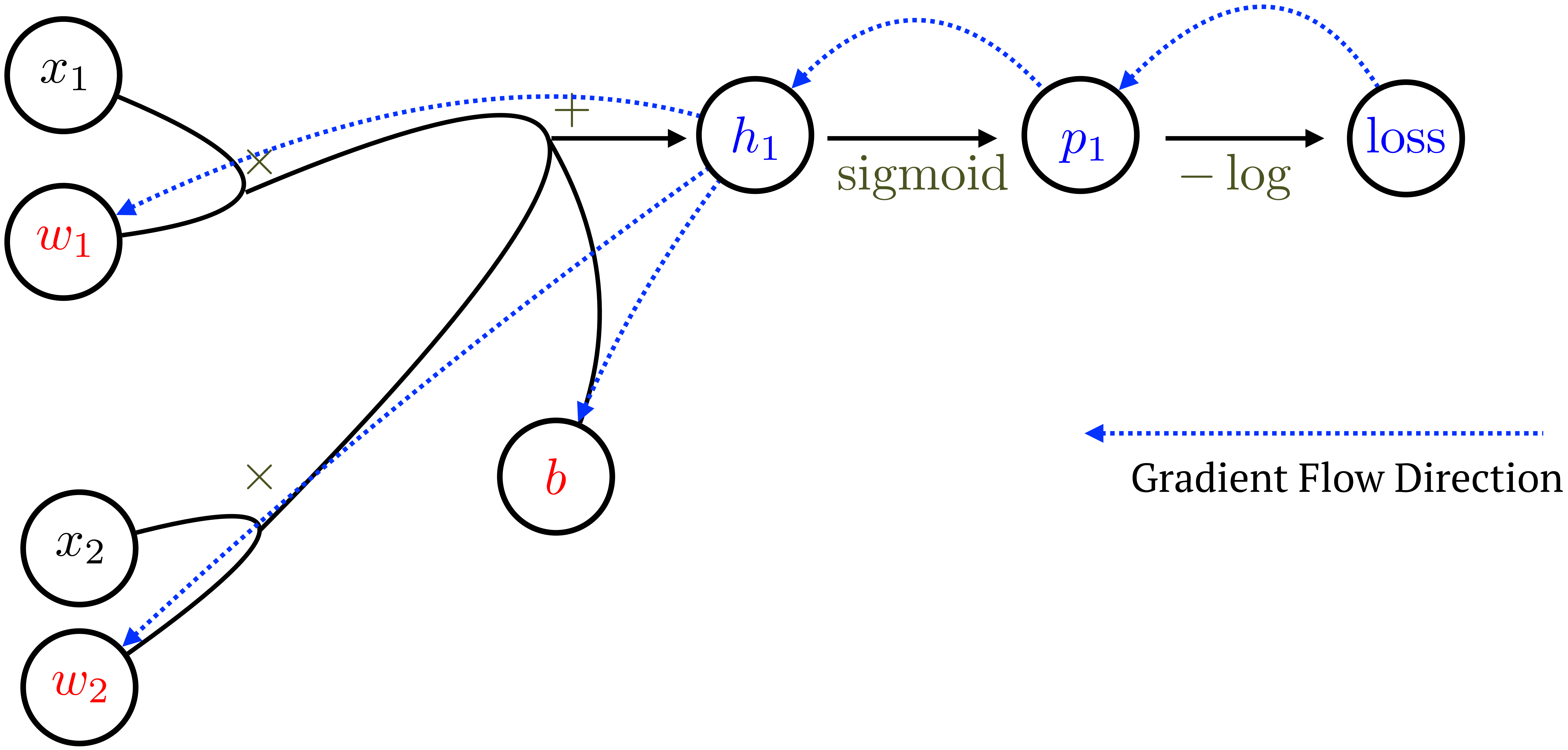
Chain rule $$\frac{\partial s}{\partial \boldsymbol{z}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$$ $$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$$ $$\frac{\partial s}{\partial \boldsymbol{h}}$$

Downstream Gradient

Local Gradient

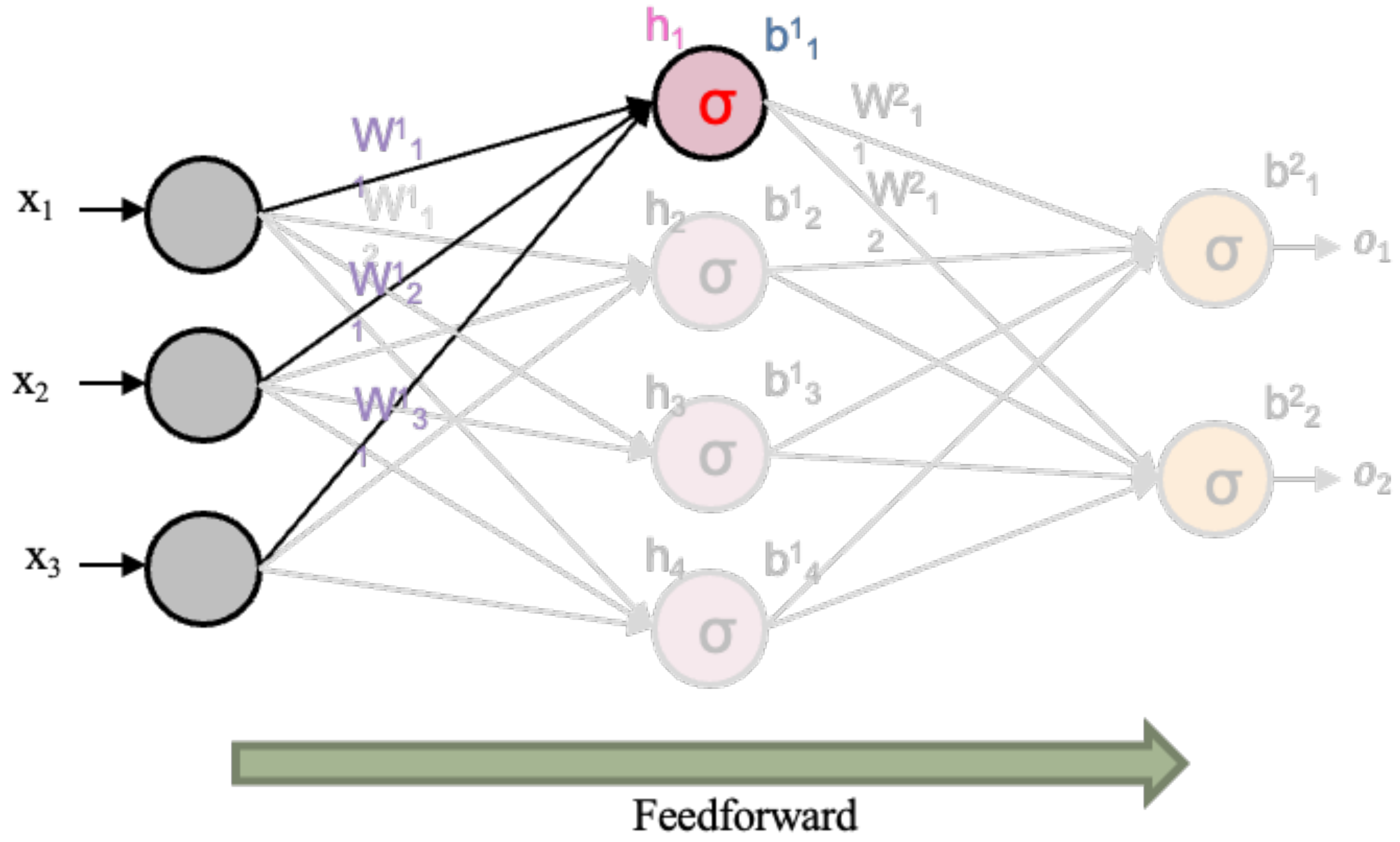
Upstream Gradient

How to minimize? (Automatic Differentiation)



Example: Feedforward Neural Network

Feedforward Neural Network

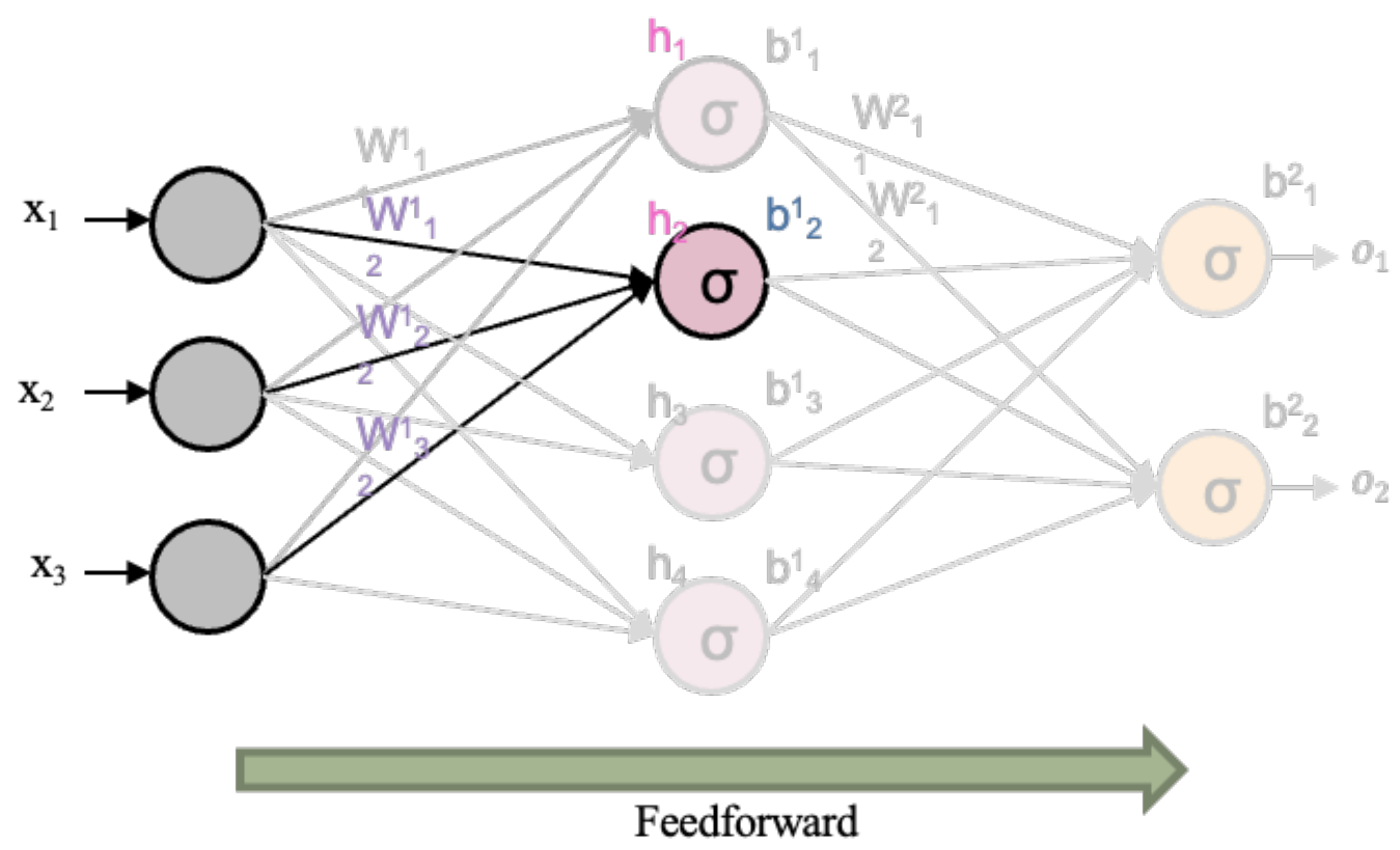


Input x Weight + Bias

$$h_1 = \sigma(x_1 w_{11}^1 + x_2 w_{21}^1 + x_3 w_{31}^1 + b_1^1)$$

Activation function

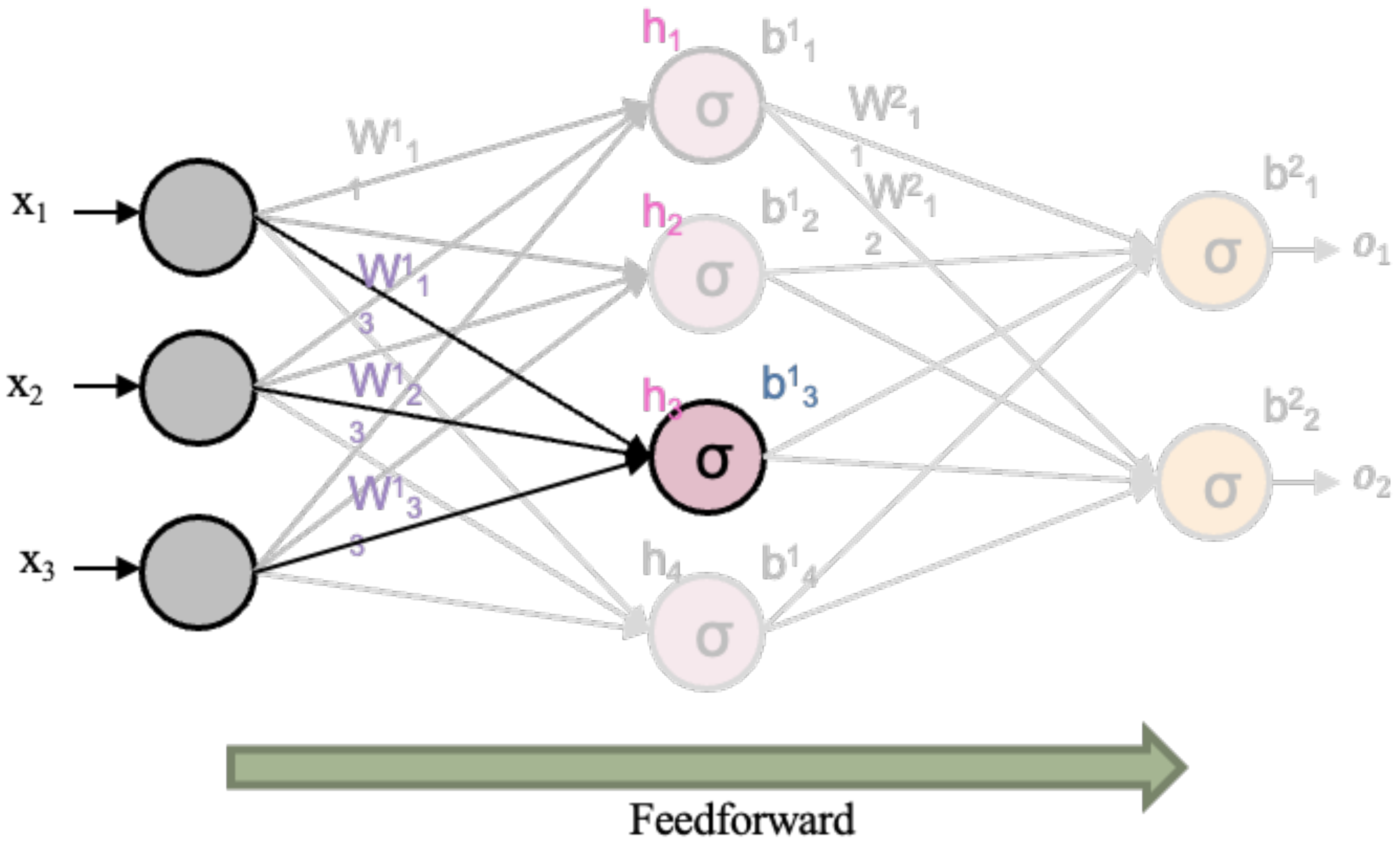
Example: Feedforward Neural Network



$$h_1 = \sigma(x_1 w_{11}^1 + x_2 w_{21}^1 + x_3 w_{31}^1 + b_1^1)$$

$$h_2 = \sigma(x_1 w_{12}^1 + x_2 w_{22}^1 + x_3 w_{32}^1 + b_2^1)$$

Example: Feedforward Neural Network

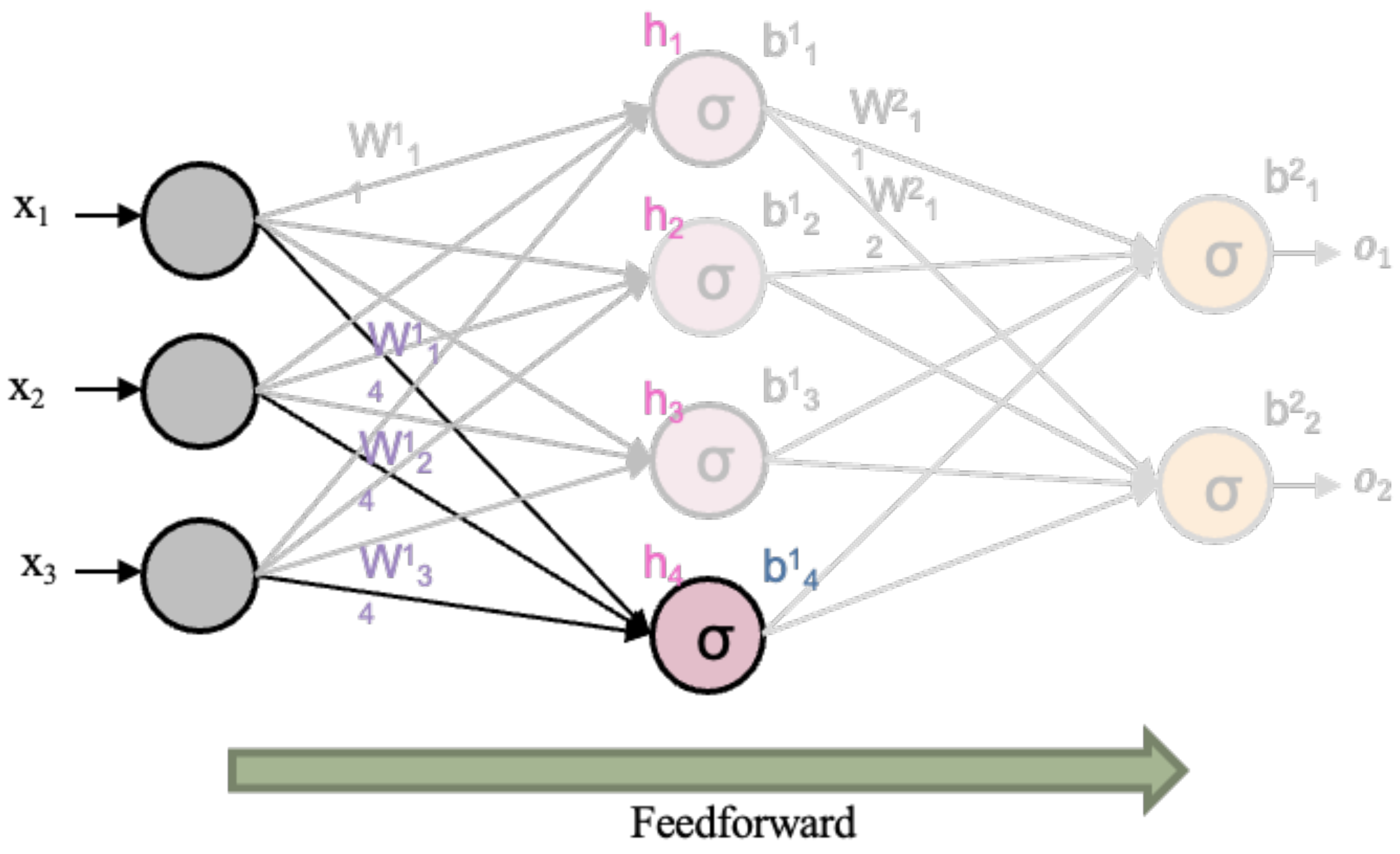


$$h_1 = \sigma(x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + b^1_1)$$

$$h_2 = \sigma(x_1 w^1_{12} + x_2 w^1_{22} + x_3 w^1_{32} + b^1_2)$$

$$h_3 = \sigma(x_1 w^1_{13} + x_2 w^1_{23} + x_3 w^1_{33} + b^1_3)$$

Example: Feedforward Neural Network



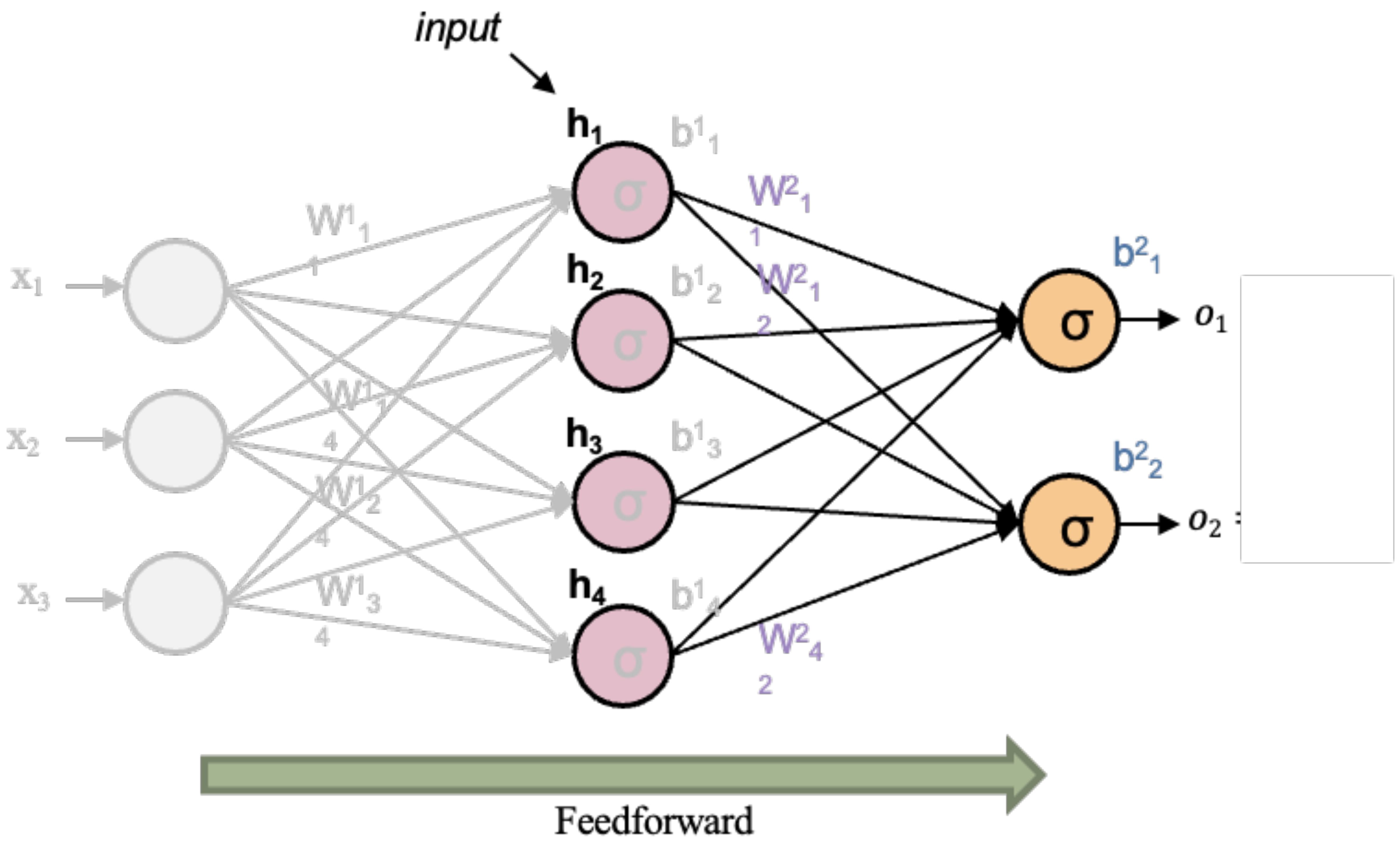
$$h_1 = \sigma(x_1 w_{11}^1 + x_2 w_{21}^1 + x_3 w_{31}^1 + b_1^1)$$

$$h_2 = \sigma(x_1 w_{12}^1 + x_2 w_{22}^1 + x_3 w_{32}^1 + b_2^1)$$

$$h_3 = \sigma(x_1 w_{13}^1 + x_2 w_{23}^1 + x_3 w_{33}^1 + b_3^1)$$

$$h_4 = \sigma(x_1 w_{14}^1 + x_2 w_{24}^1 + x_3 w_{34}^1 + b_4^1)$$

Example: Feedforward Neural Network

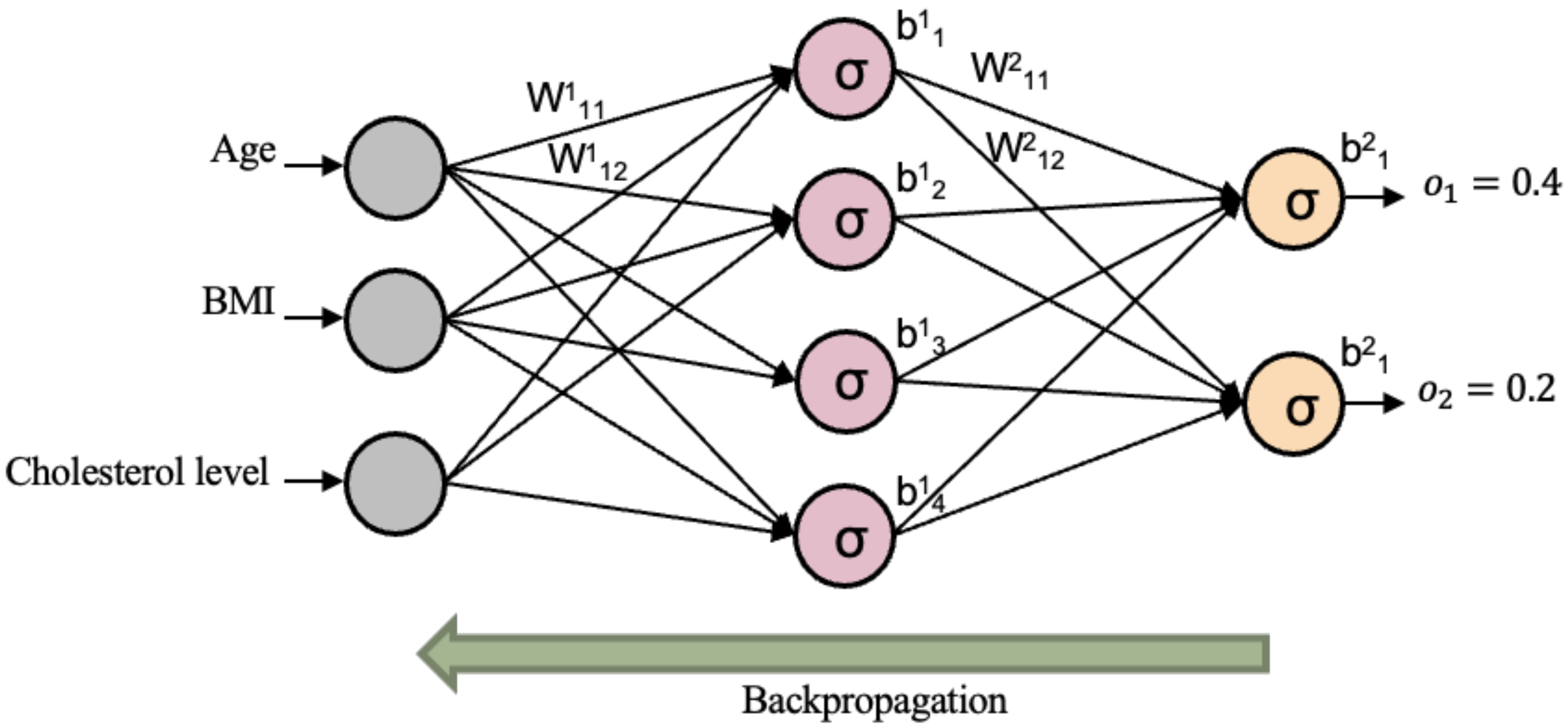


$$o_1 = \sigma(h_1 w_{11}^2 + h_2 w_{21}^2 + h_3 w_{31}^2 + h_4 w_{41}^2 + b_1^2)$$

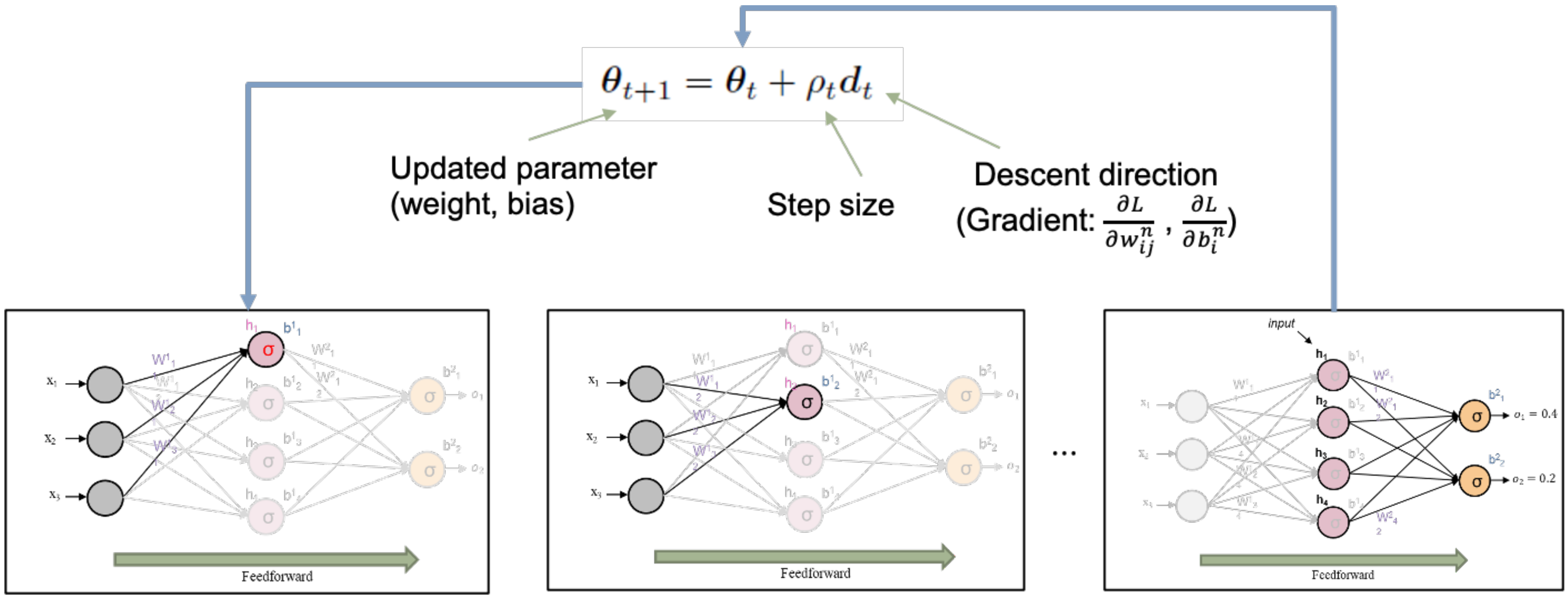
$$o_2 = \sigma(h_1 w_{12}^2 + h_2 w_{22}^2 + h_3 w_{32}^2 + h_4 w_{42}^2 + b_2^2)$$

Example: Feedforward Neural Network

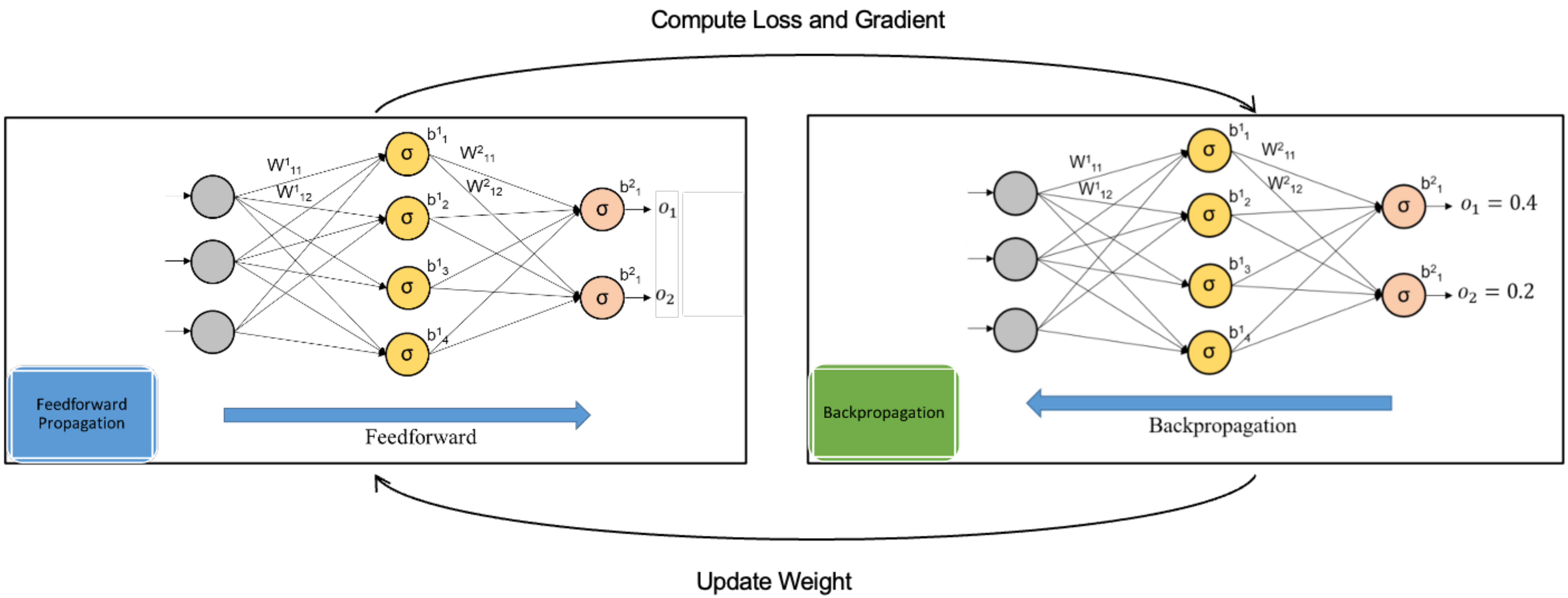
- Compute Gradients, i.e: $\frac{\partial L}{\partial w_{ij}^{n}}, \frac{\partial L}{\partial b_{i}^{n}}$
- Backpropagate the gradient to updates the weights



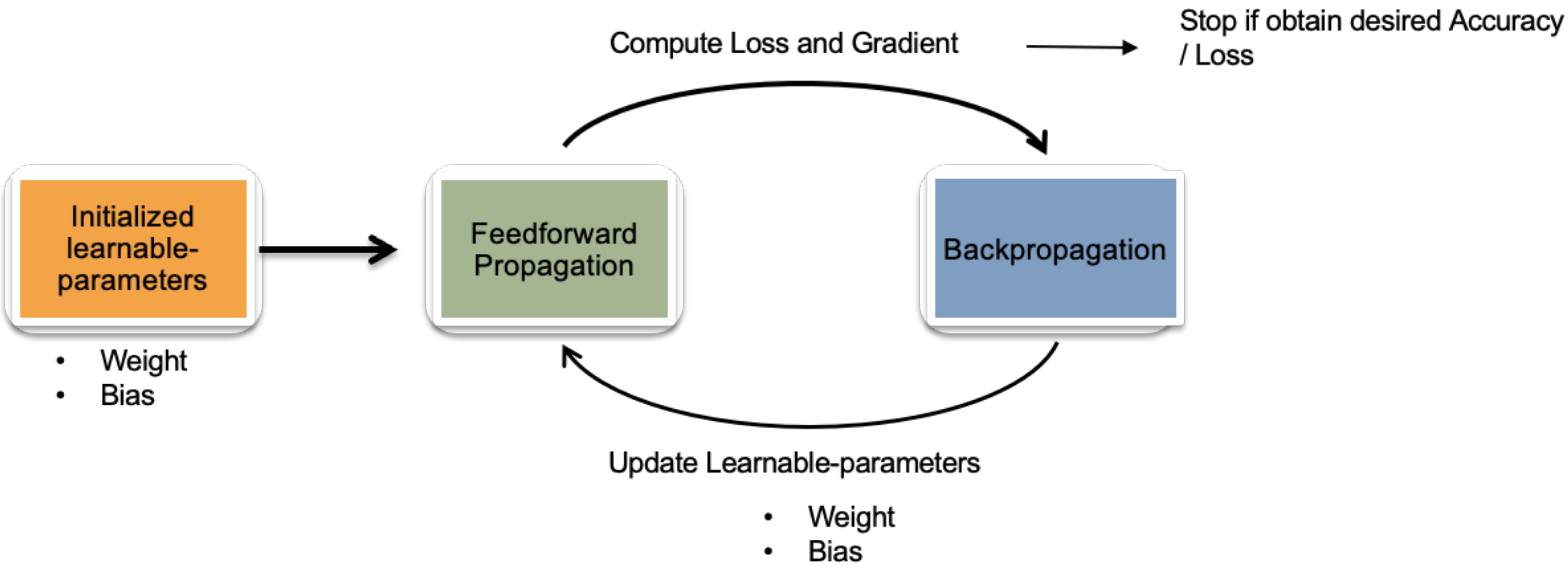
Example: Feedforward Neural Network



Example: Feedforward Neural Network



Summary

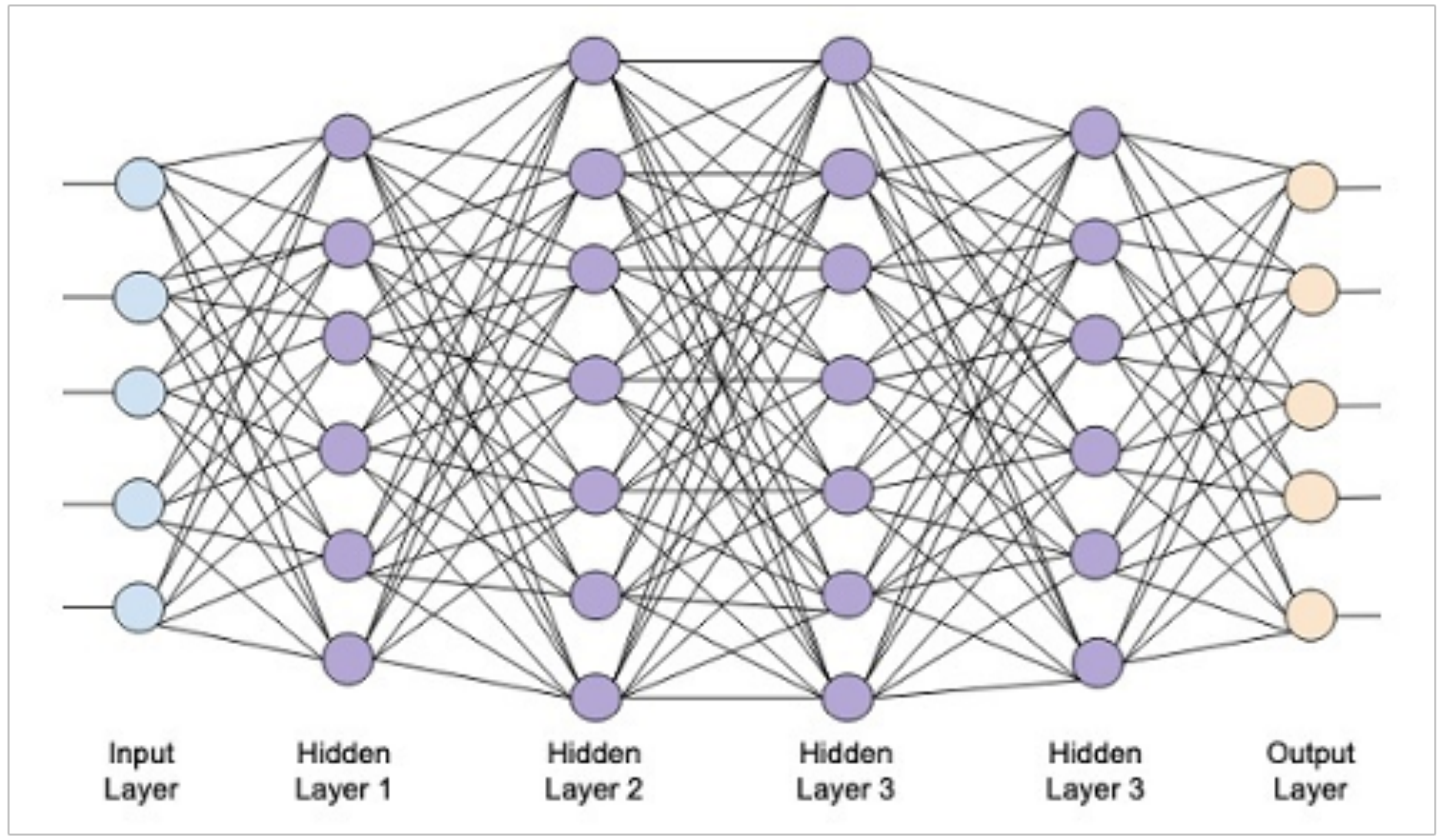


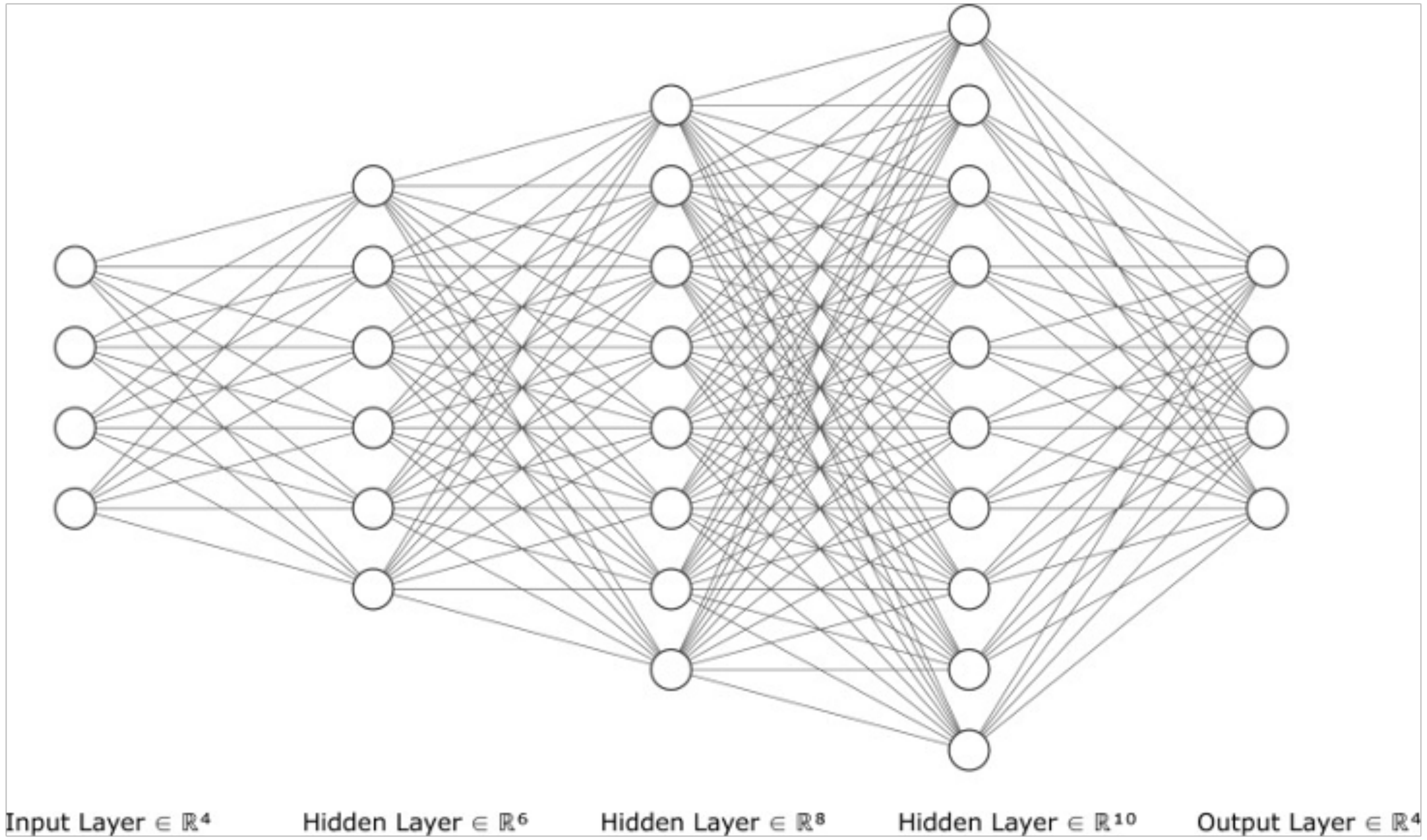
Compute Loss and Gradient
Stop if obtain desired Accuracy / Loss
Initialized learnable-parameters
Feedforward Propagation
Backpropagation
• Weight
• Bias
Update Learnable-parameters
• Weight
• Bias

Larger Networks

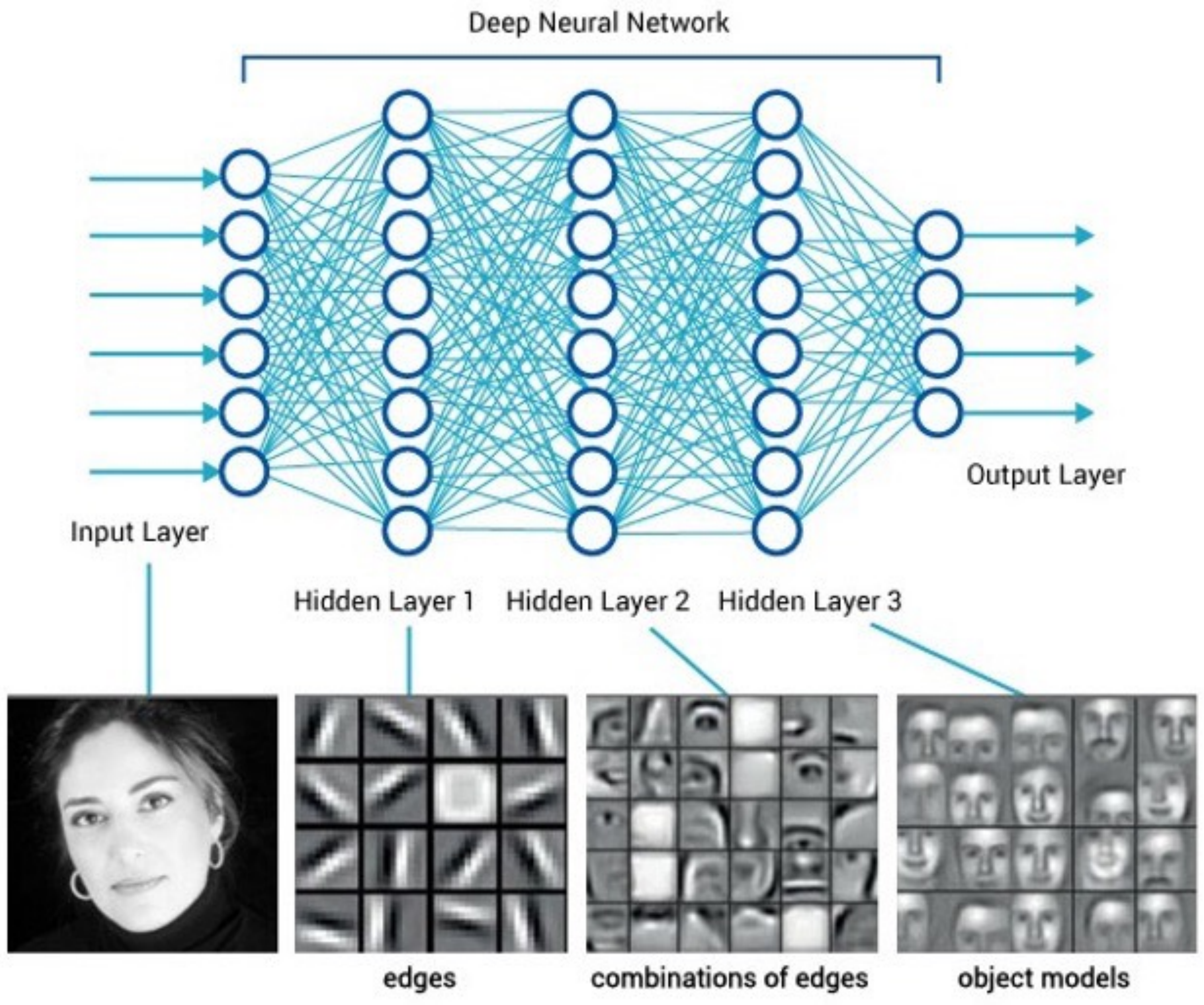
Increase number of nodes

Increase number of hidden layers

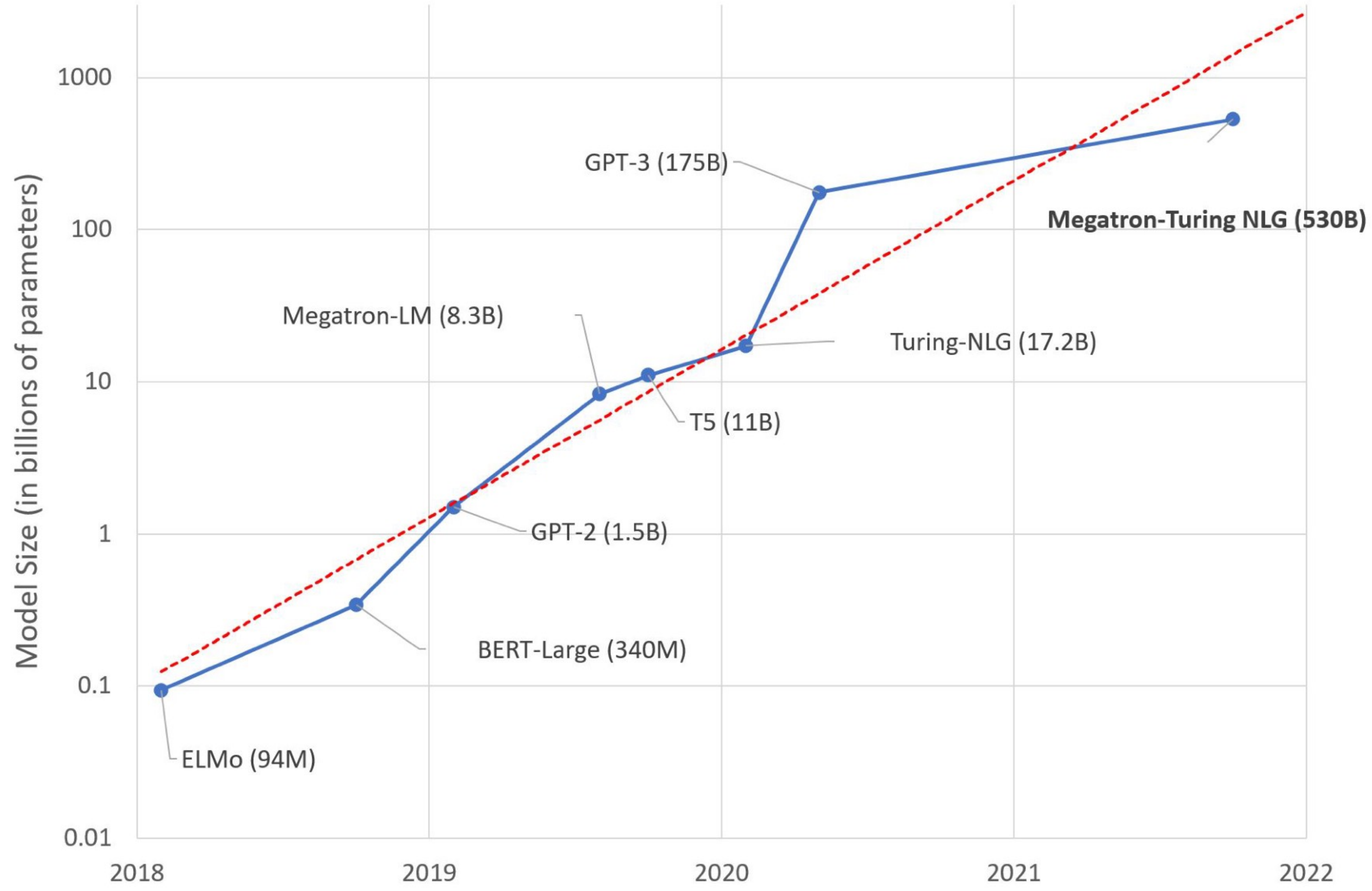




Larger Networks



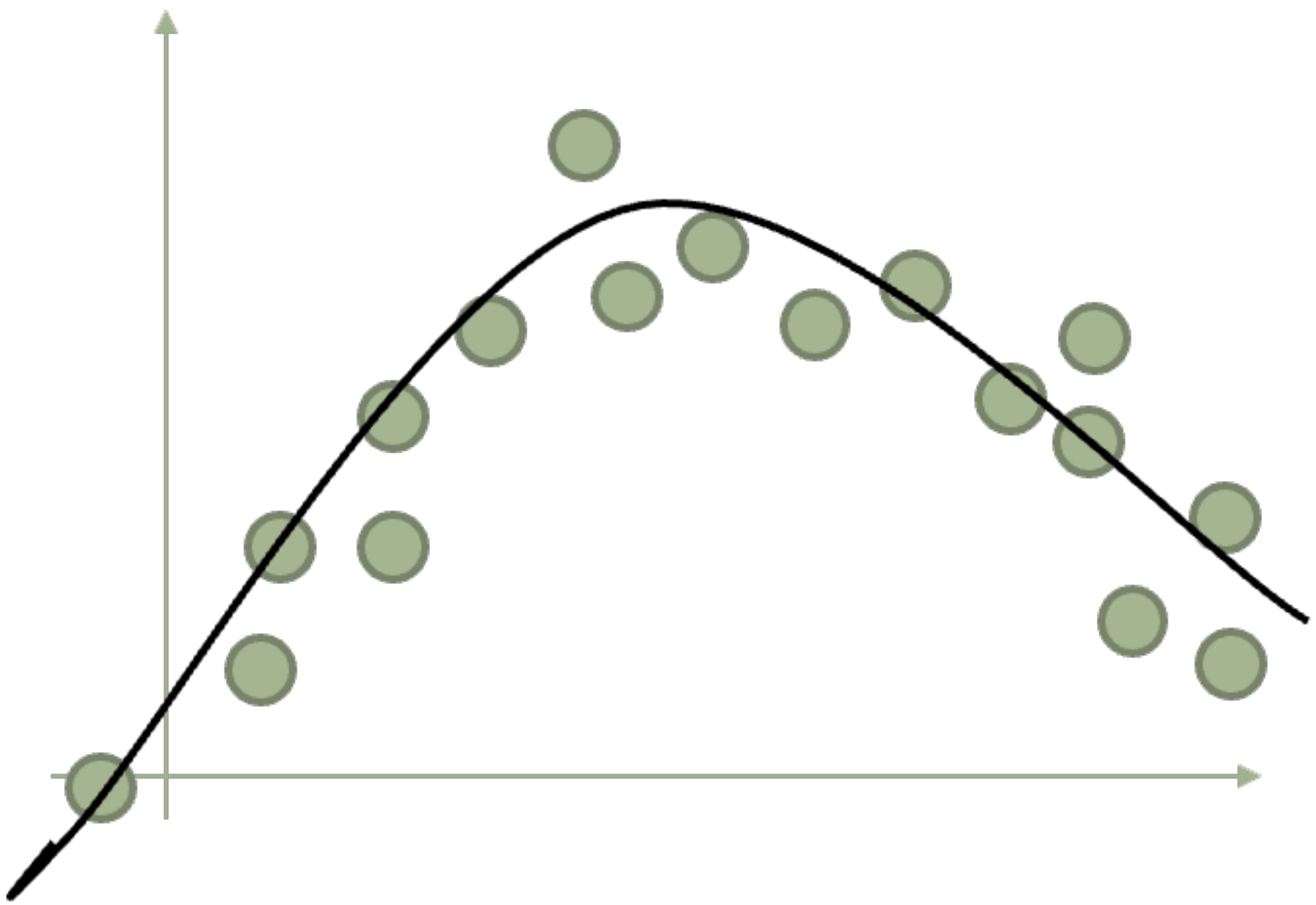
$12 million or ~300 V100 GPUs for 3 months



Microsoft Research Blog. Oct 6, 2021.

Overfitting

Say we have the following graph

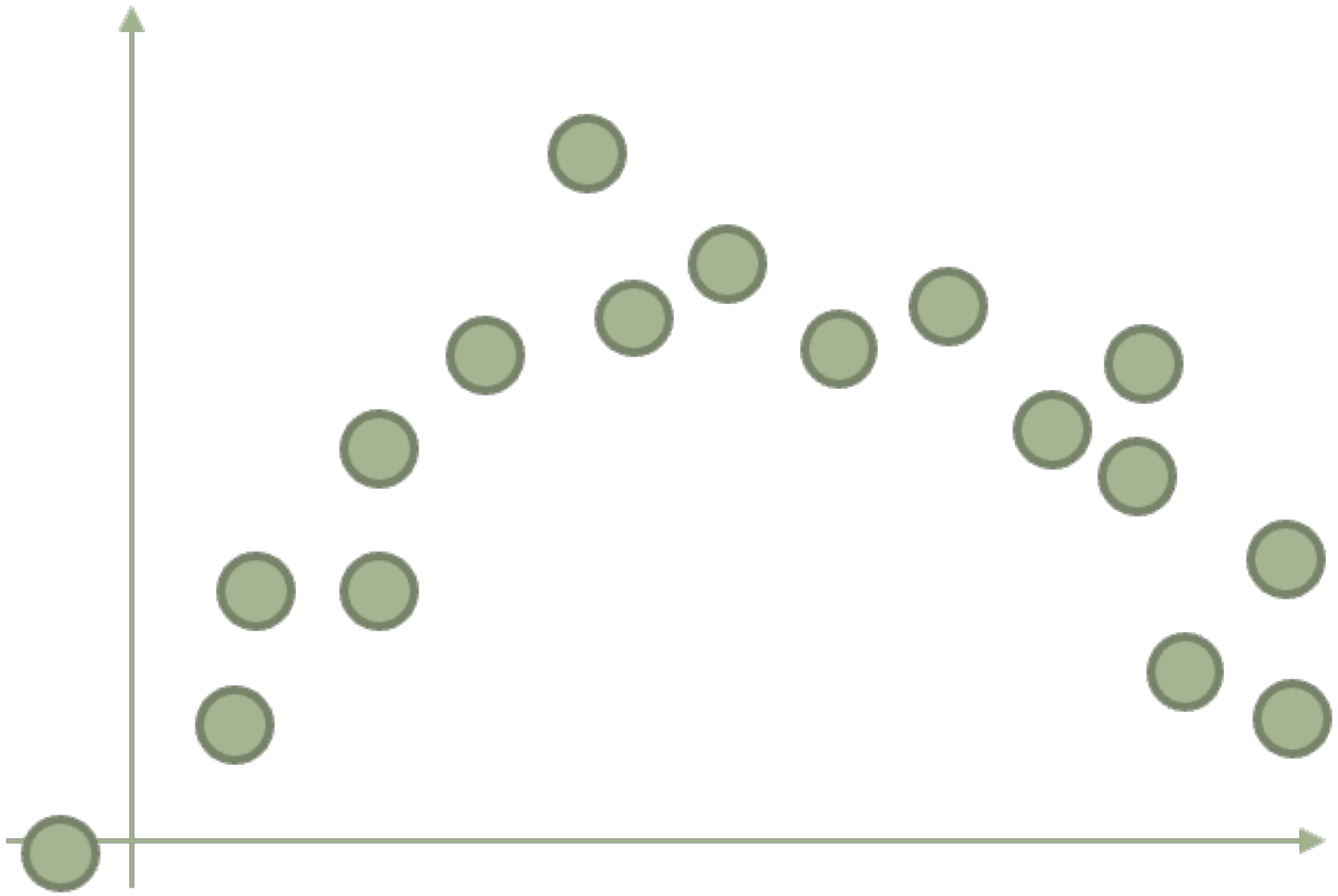


We know it should be a quadratic graph

Let's assume the best fit graph would be $y = -3x^2 + x + 0.5$

Overfitting

Say we have the following graph



We know it should be a quadratic graph

Let’s assume the best fit graph would be y =-3x^2+x+0.5

If we fit with a higher order arbitrary polynomial function, $y = 0.4x^8+1.9x^7-1.4x^6+\cdots+1.9$ -> overfit

Regularization

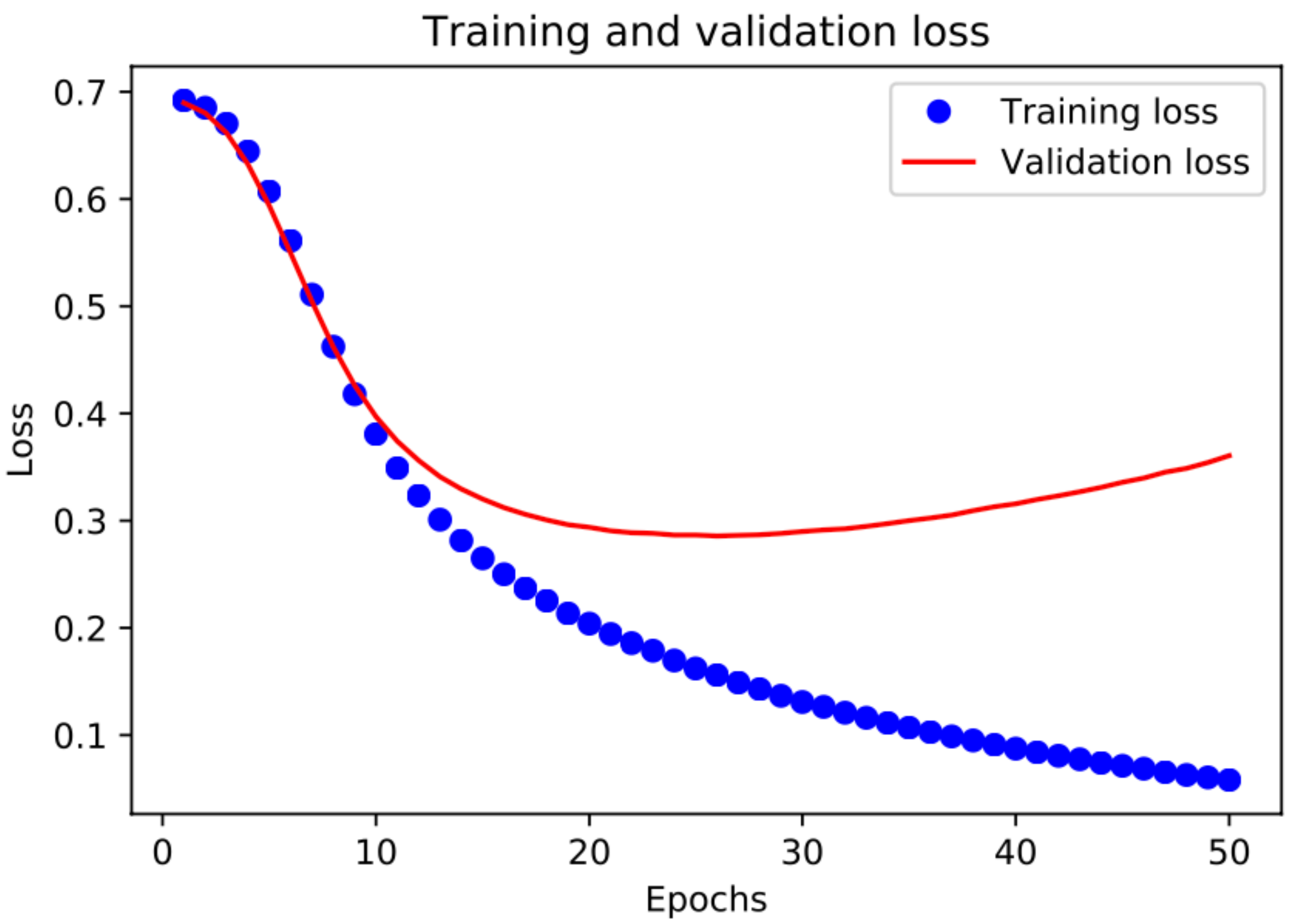
Weight Decay

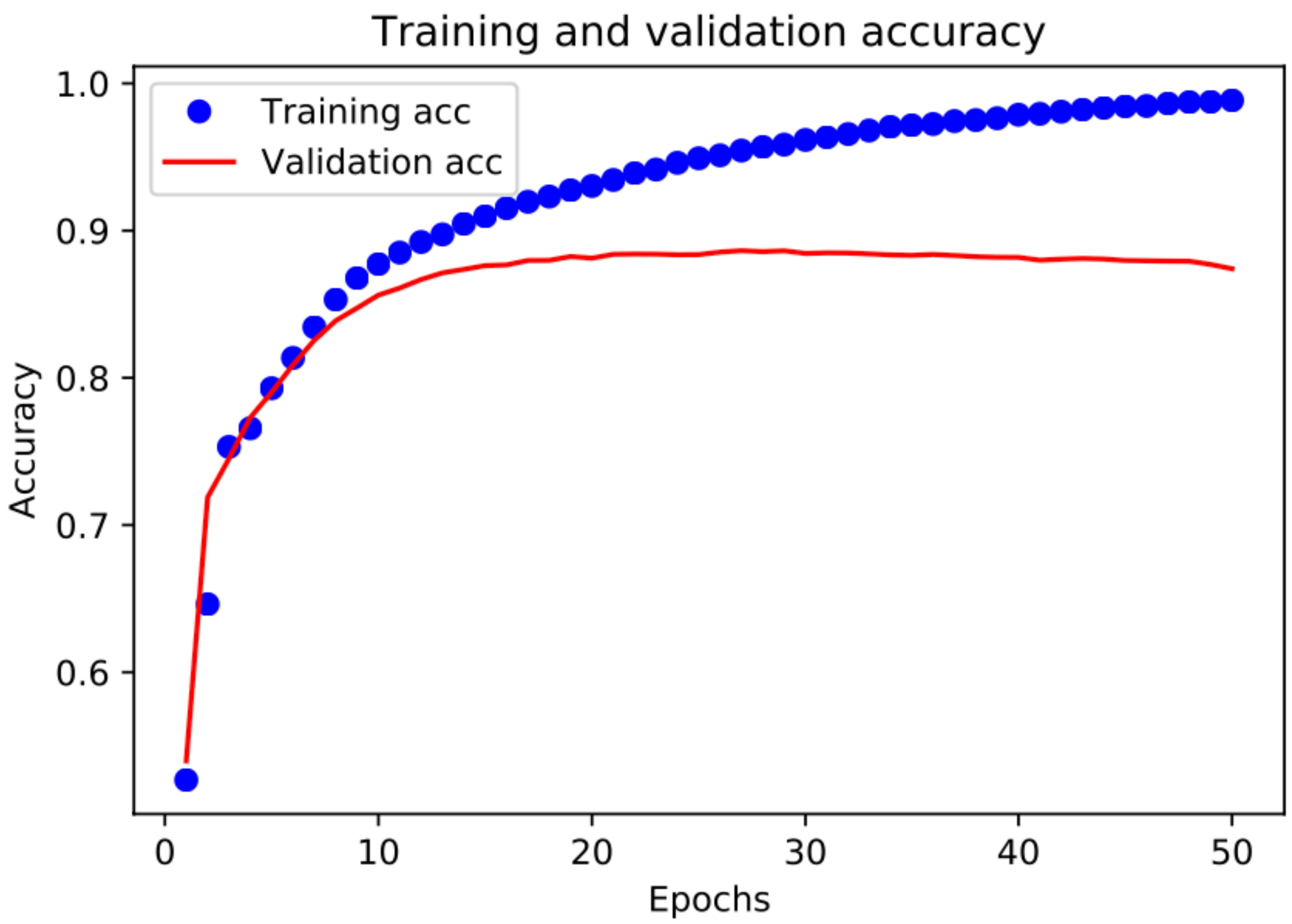
$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{n=1}^{N} L(\mathbf{w}; x_n, y_n)}_{\text{empirical loss}} + \underbrace{\Omega(\mathbf{w})}_{\text{regularizer}}$$

$$||\boldsymbol{w}||_2 = \sqrt{\sum_{d=1}^{D} |w_d|^2} = \sqrt{\boldsymbol{w}^T \boldsymbol{w}}$$

Regularization

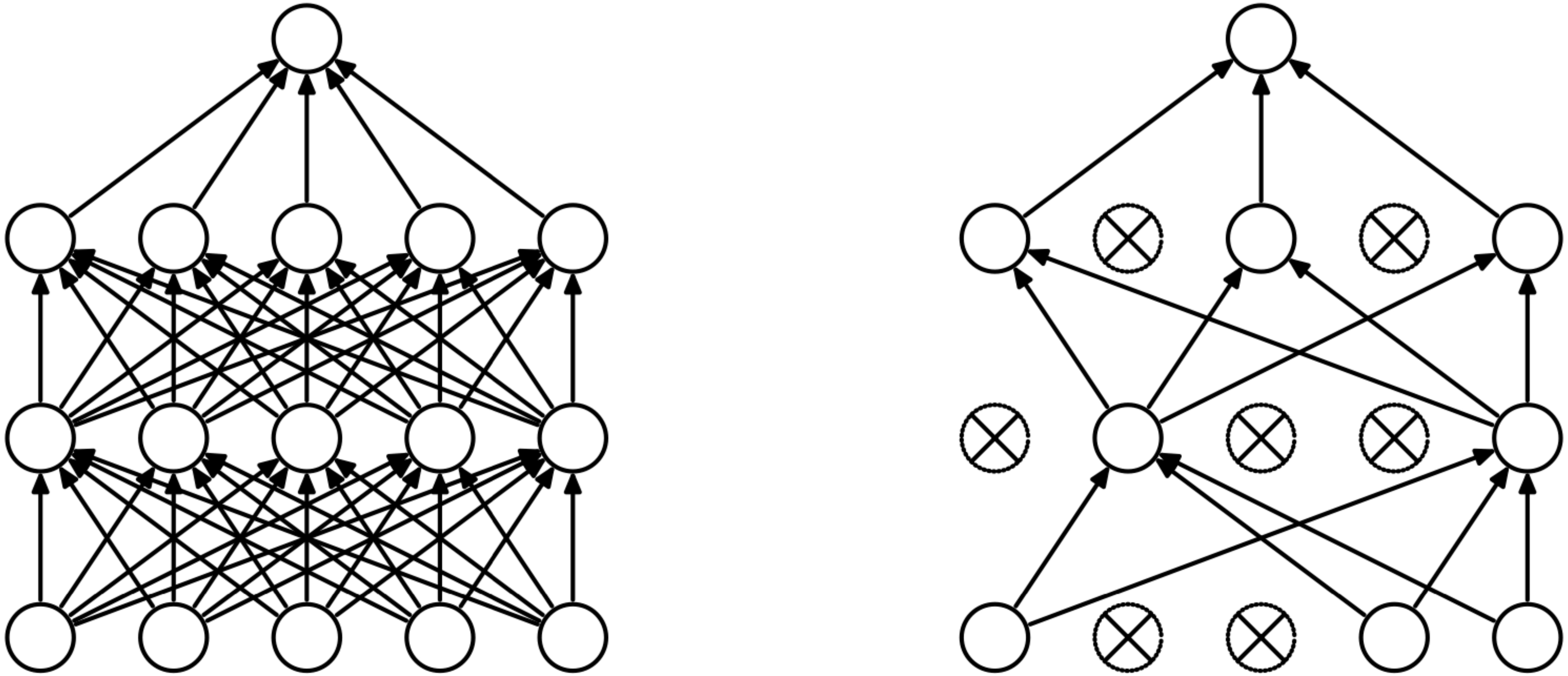
Early stopping





Regularization

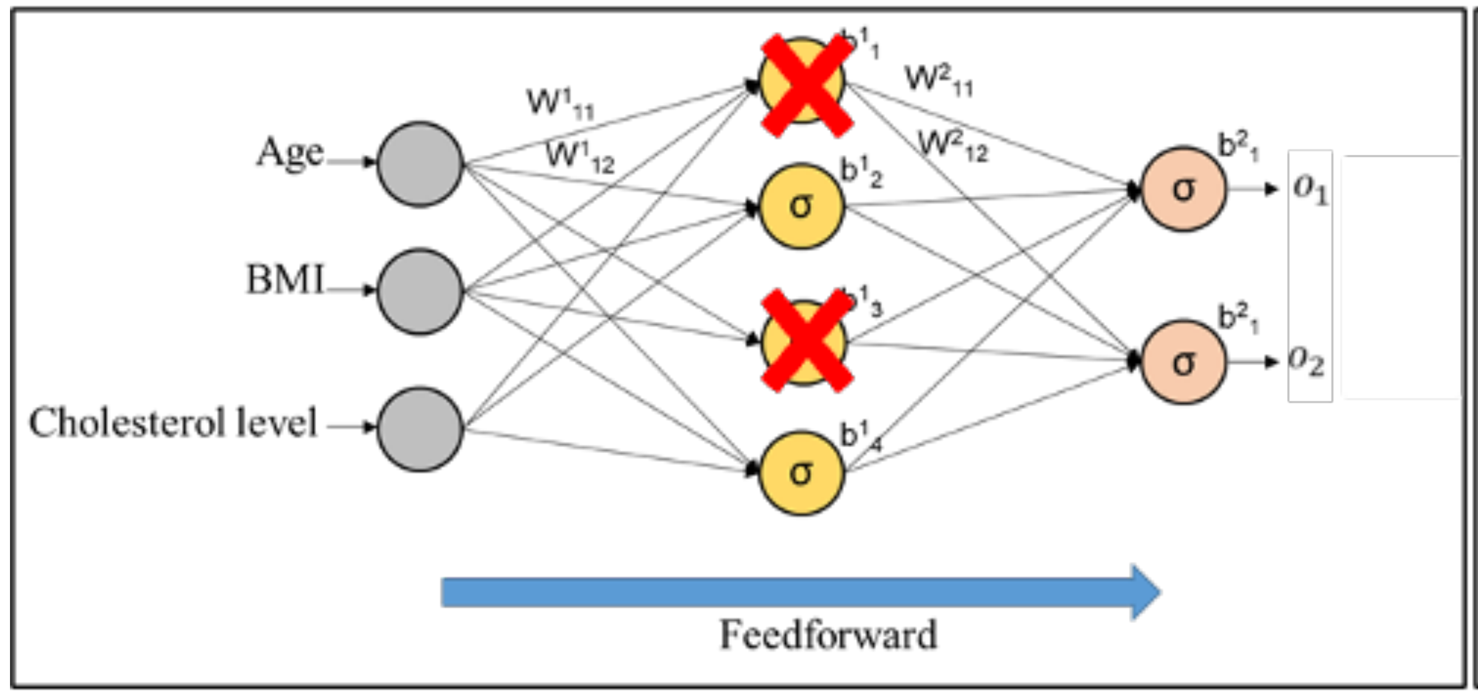
Dropout



Regularization

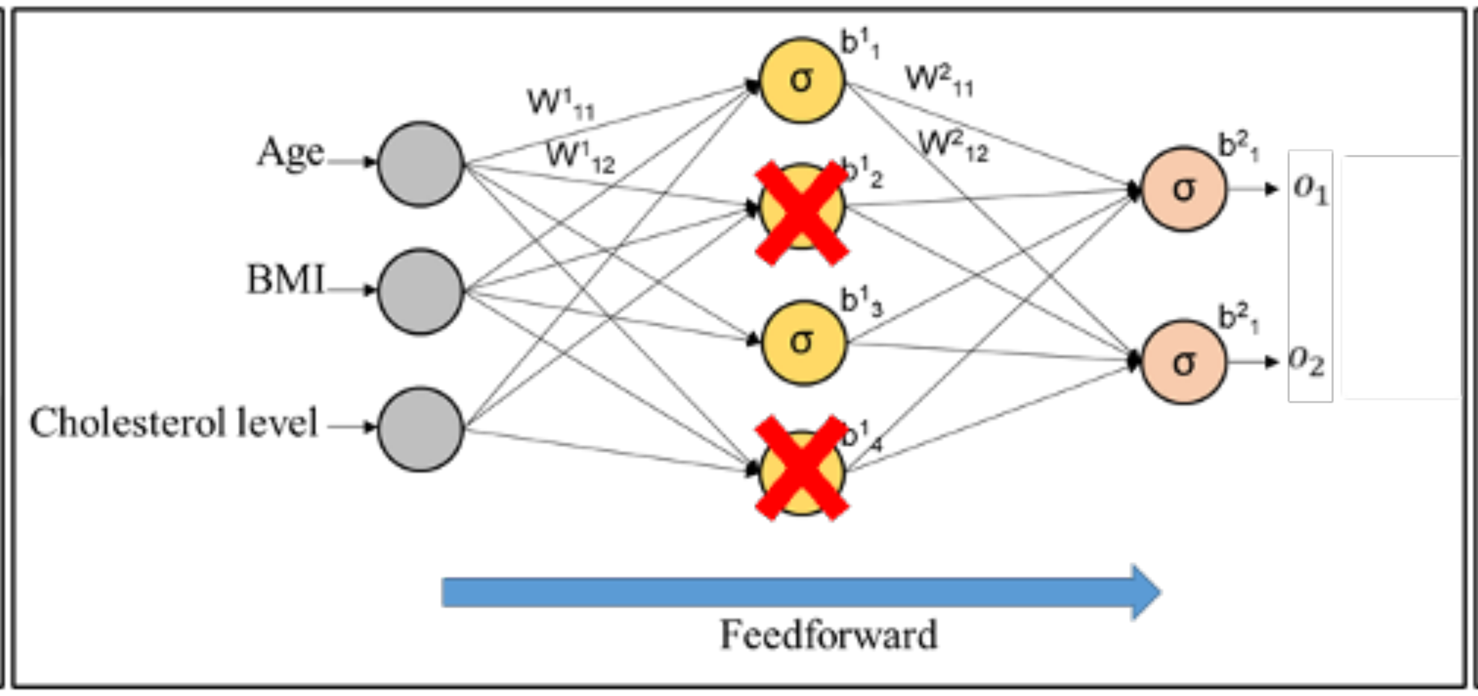
Dropout

1^{st} iteration (Model1)



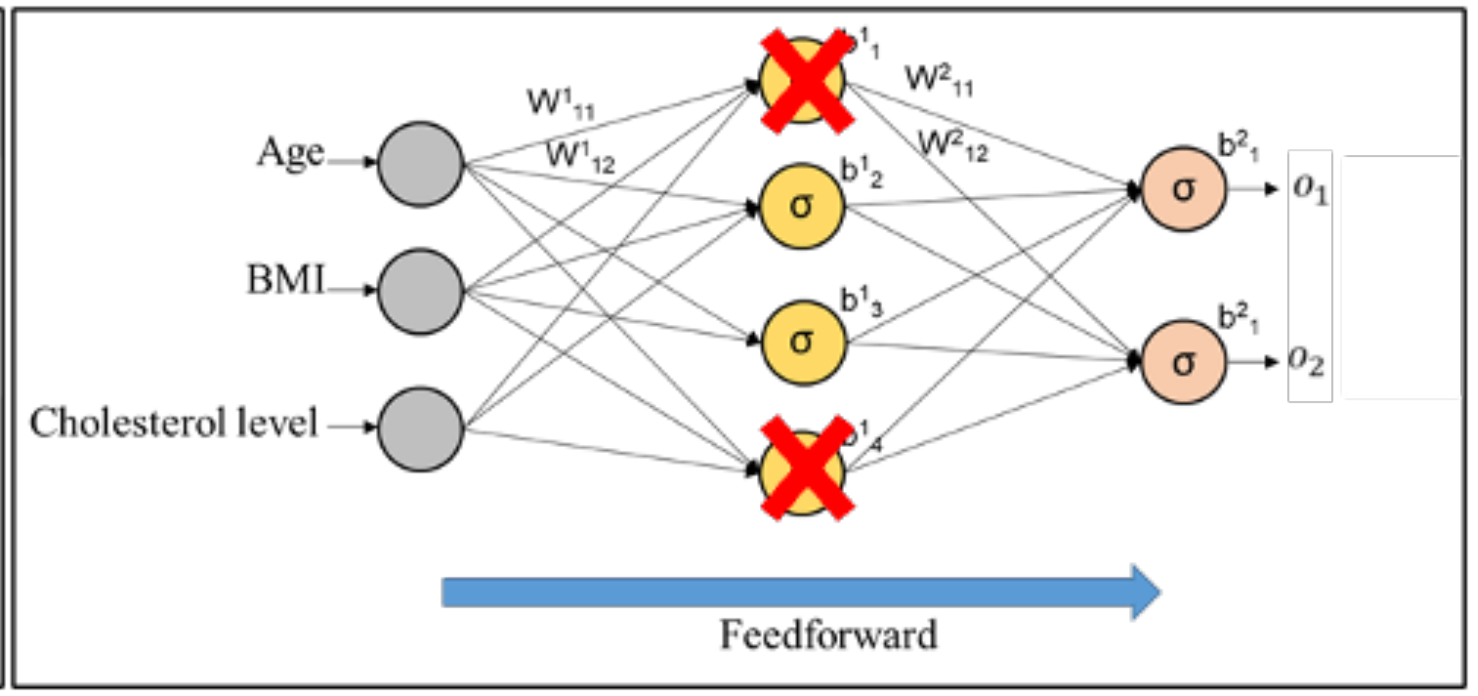
1. Forward
2. Compute gradient
3. Update weight

2^{nd} iteration (Model2)



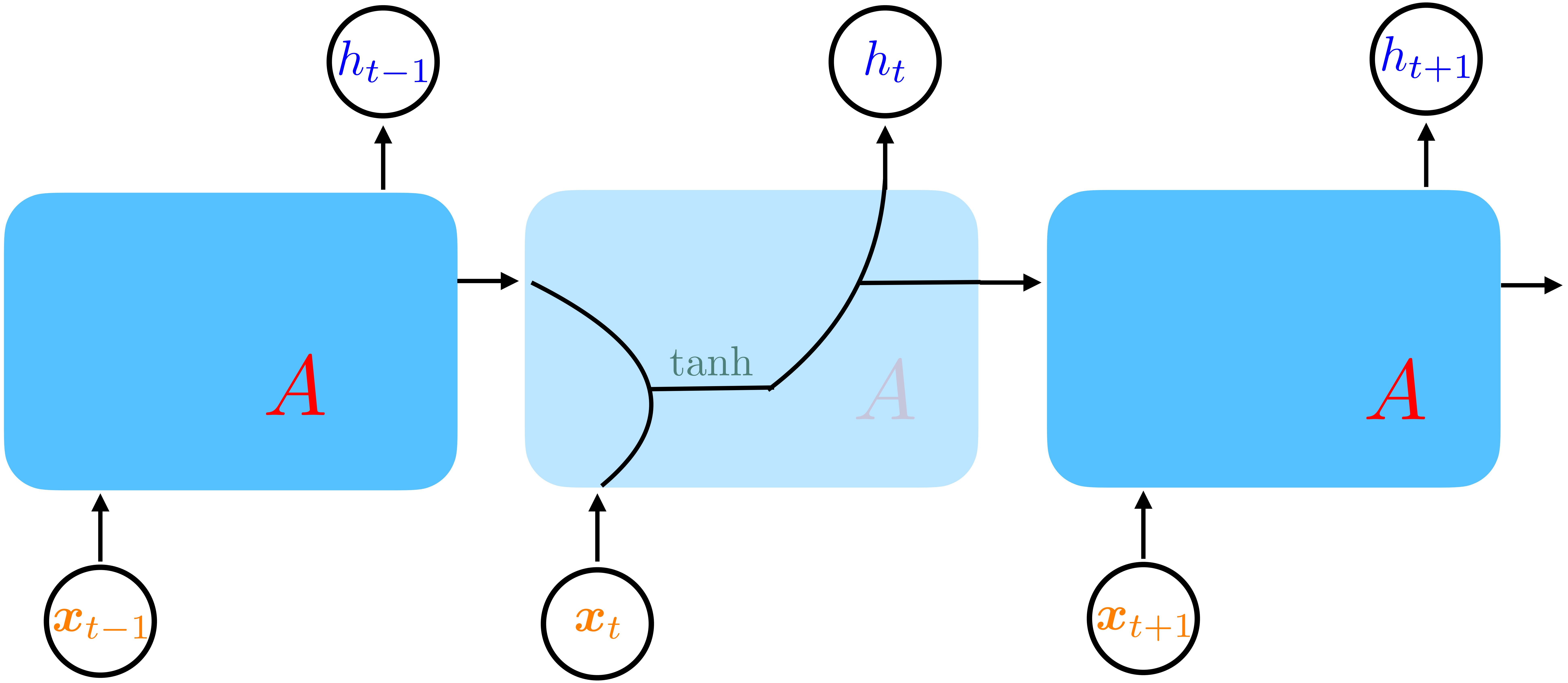
1. Forward
2. Compute gradient
3. Update weight

3^{rd} iteration (Model3)

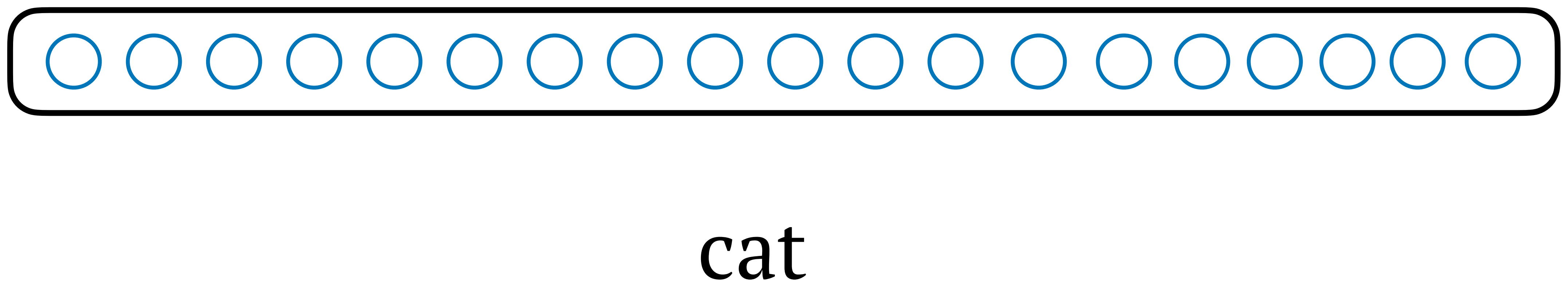


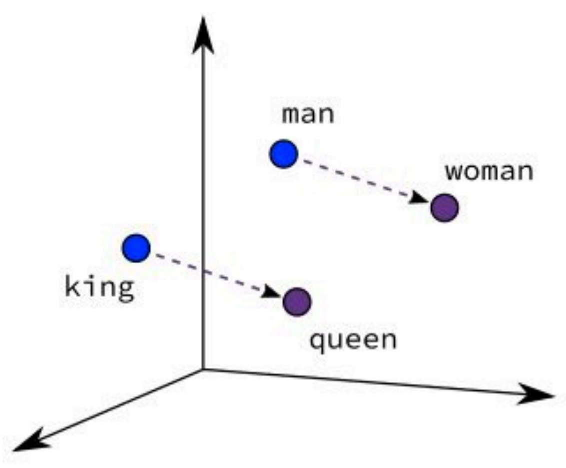
1. Forward
2. Compute gradient
3. Update weight

Recurrent Neural Networks (RNNs)

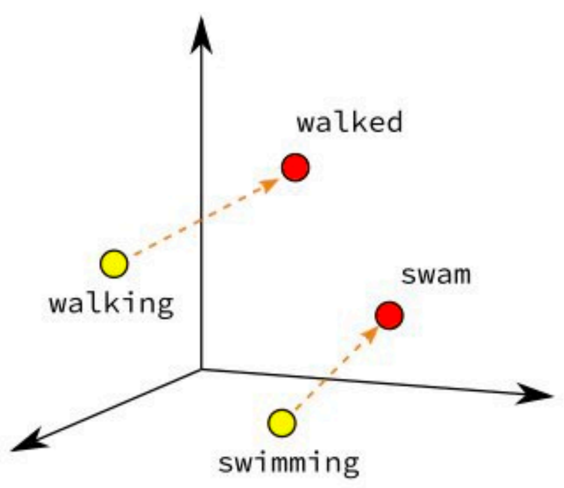


Word Embeddings

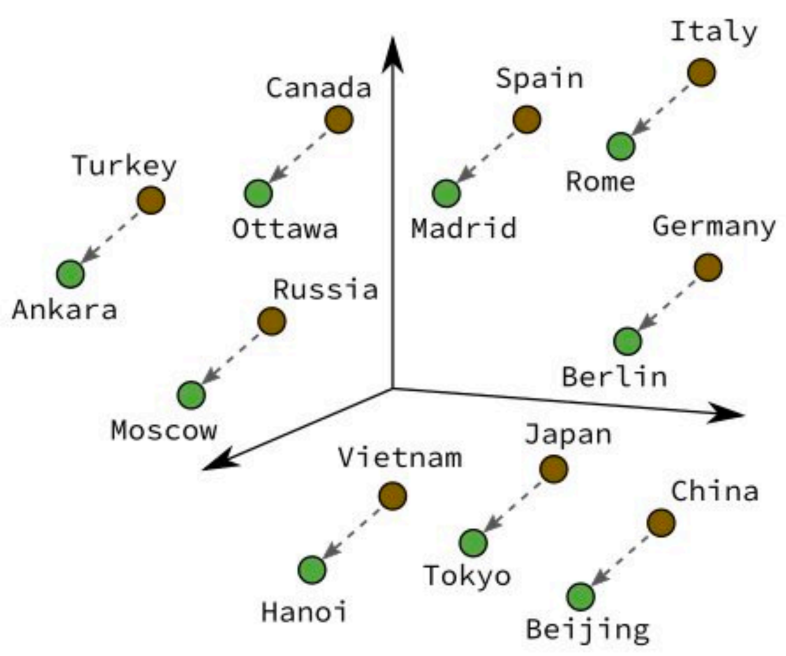




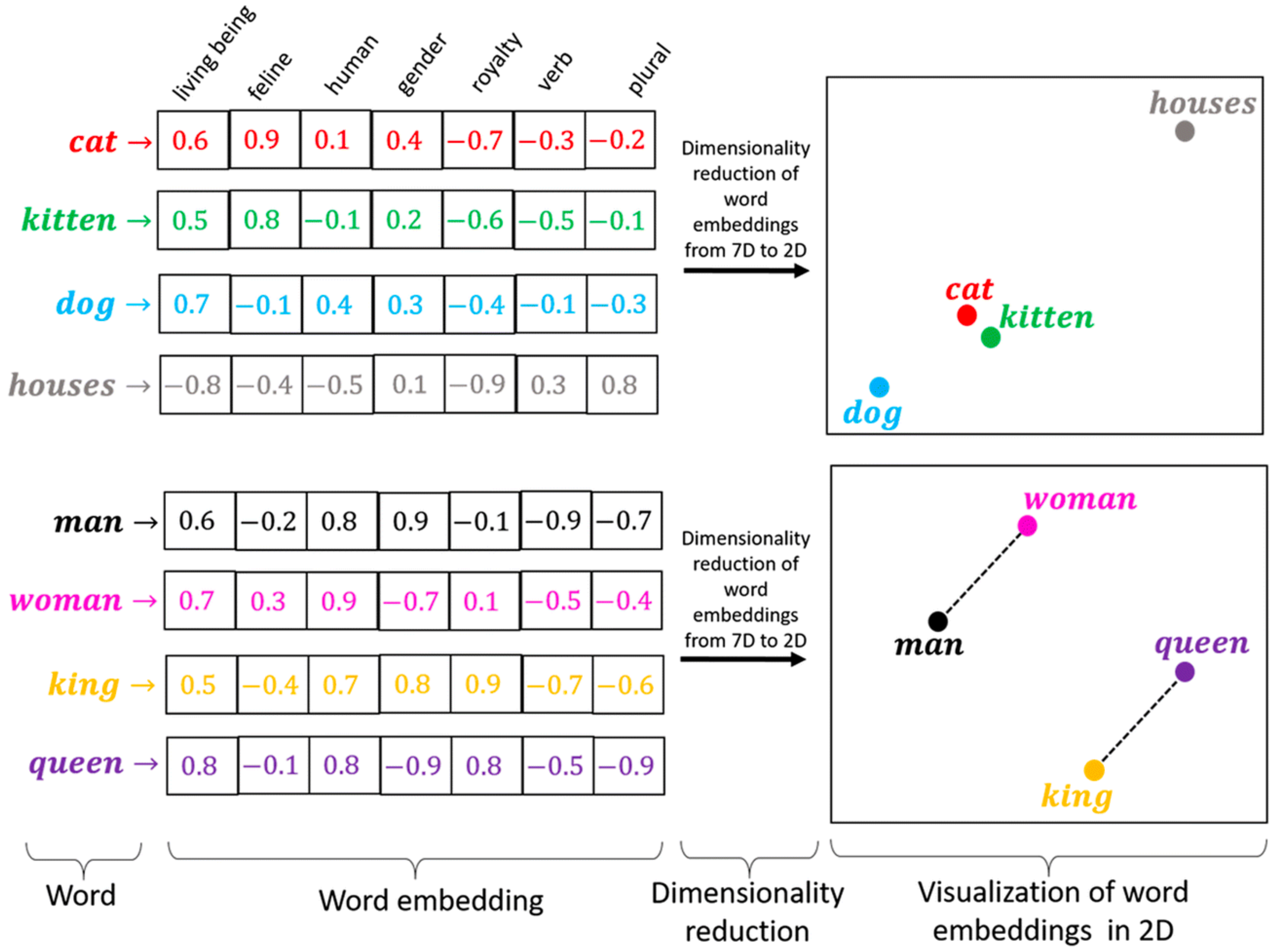
Male-Female



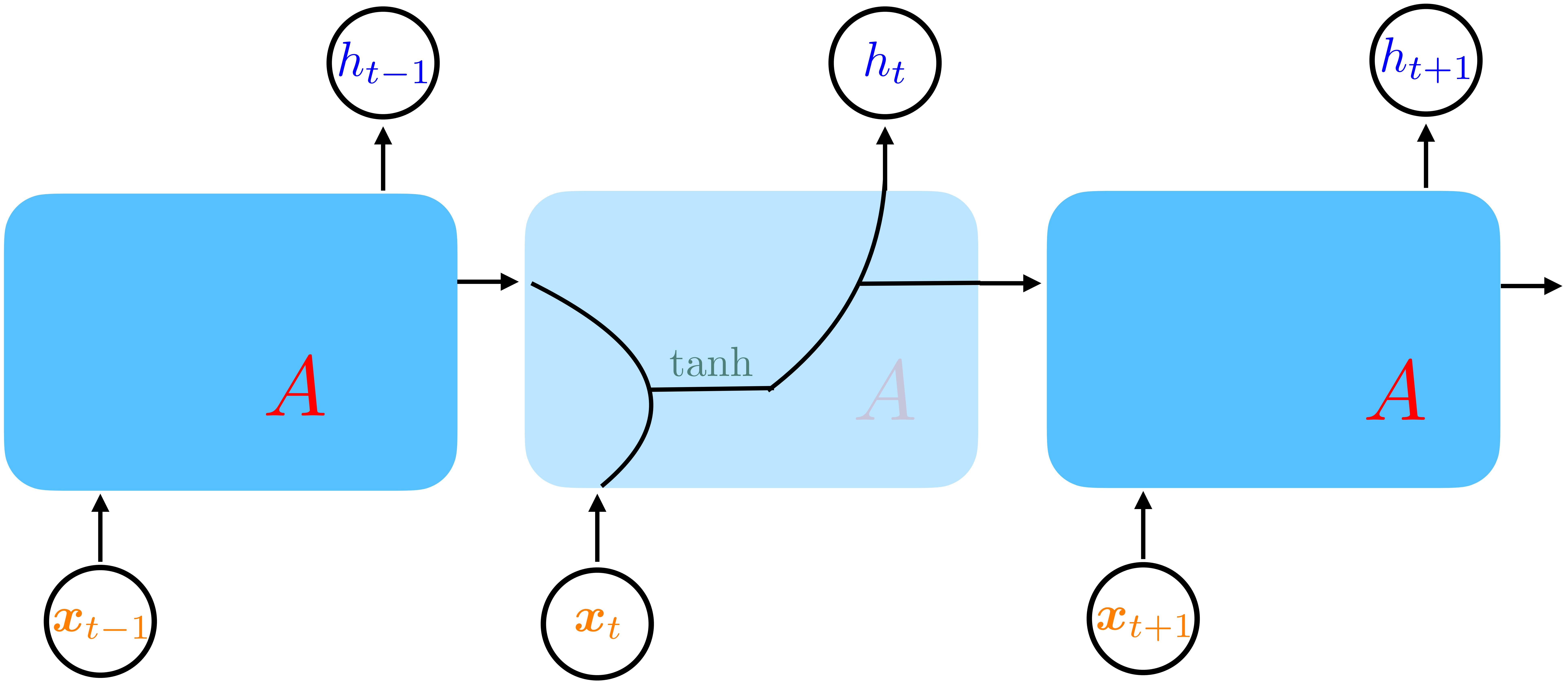
Verb Tense



Country-Capital



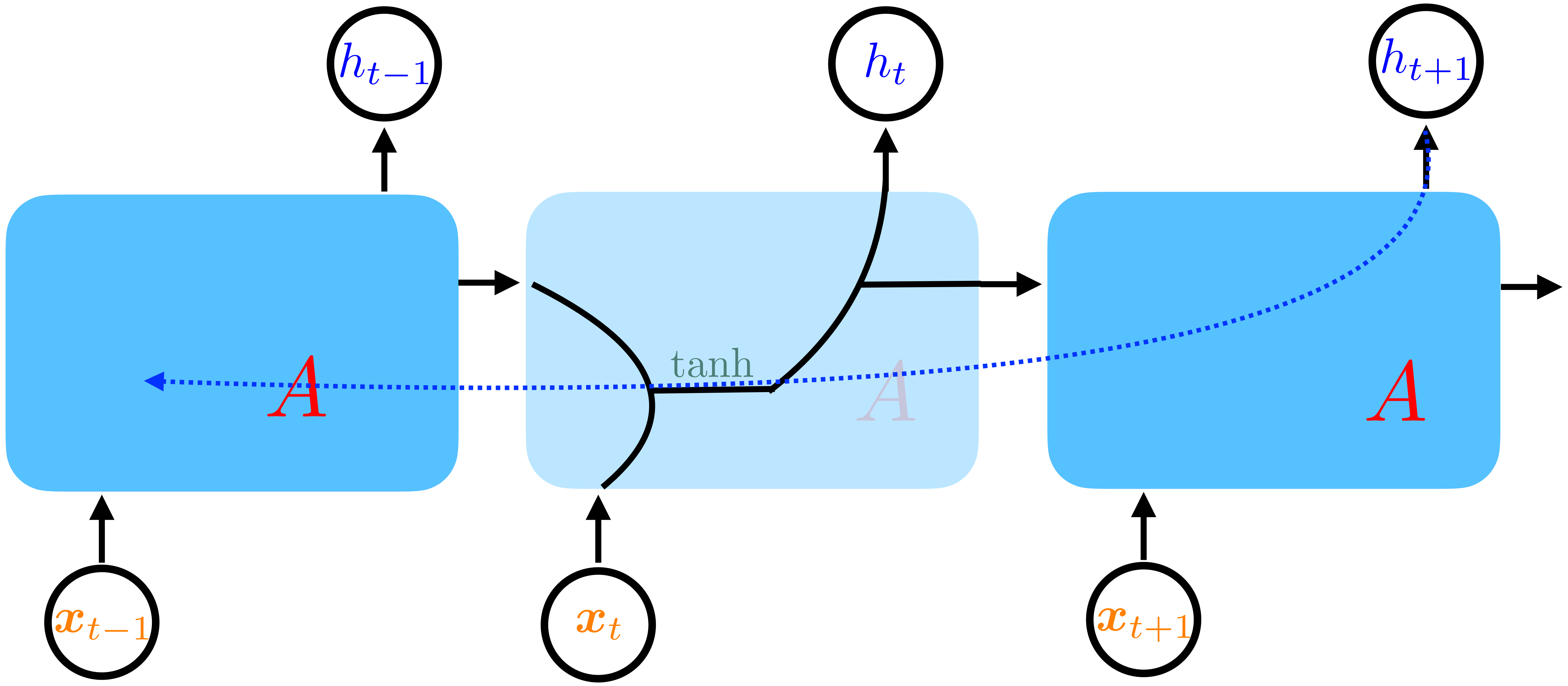
Recurrent Neural Networks (RNNs)

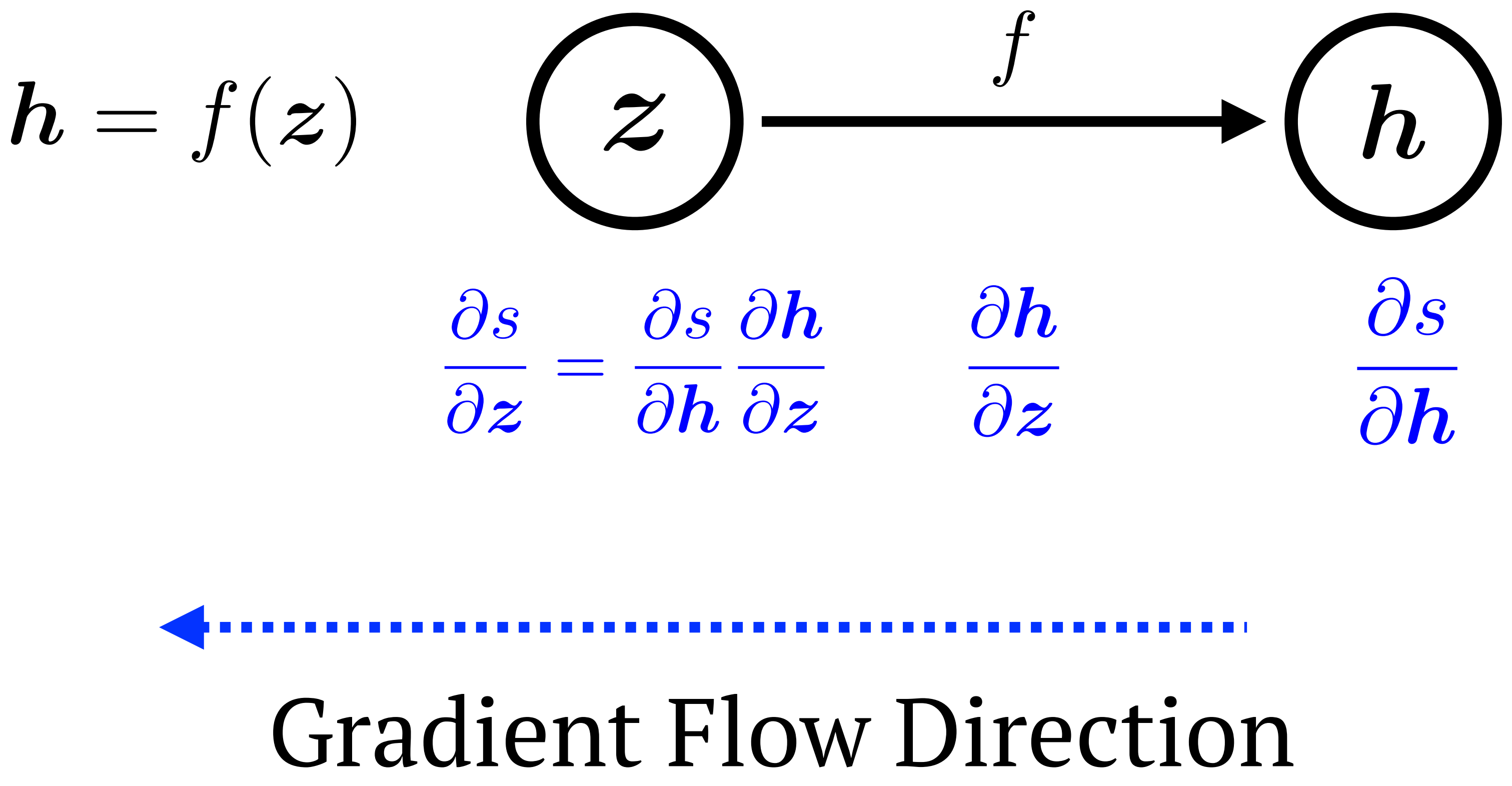


$$\mathbf{h}_t = \sigma(\mathbf{C}_x \mathbf{x}_t + \mathbf{C}_h \mathbf{h}_{t-1})$$

$$\mathbf{y}_t = f(\mathbf{W}\mathbf{h}_t + \mathbf{b})$$

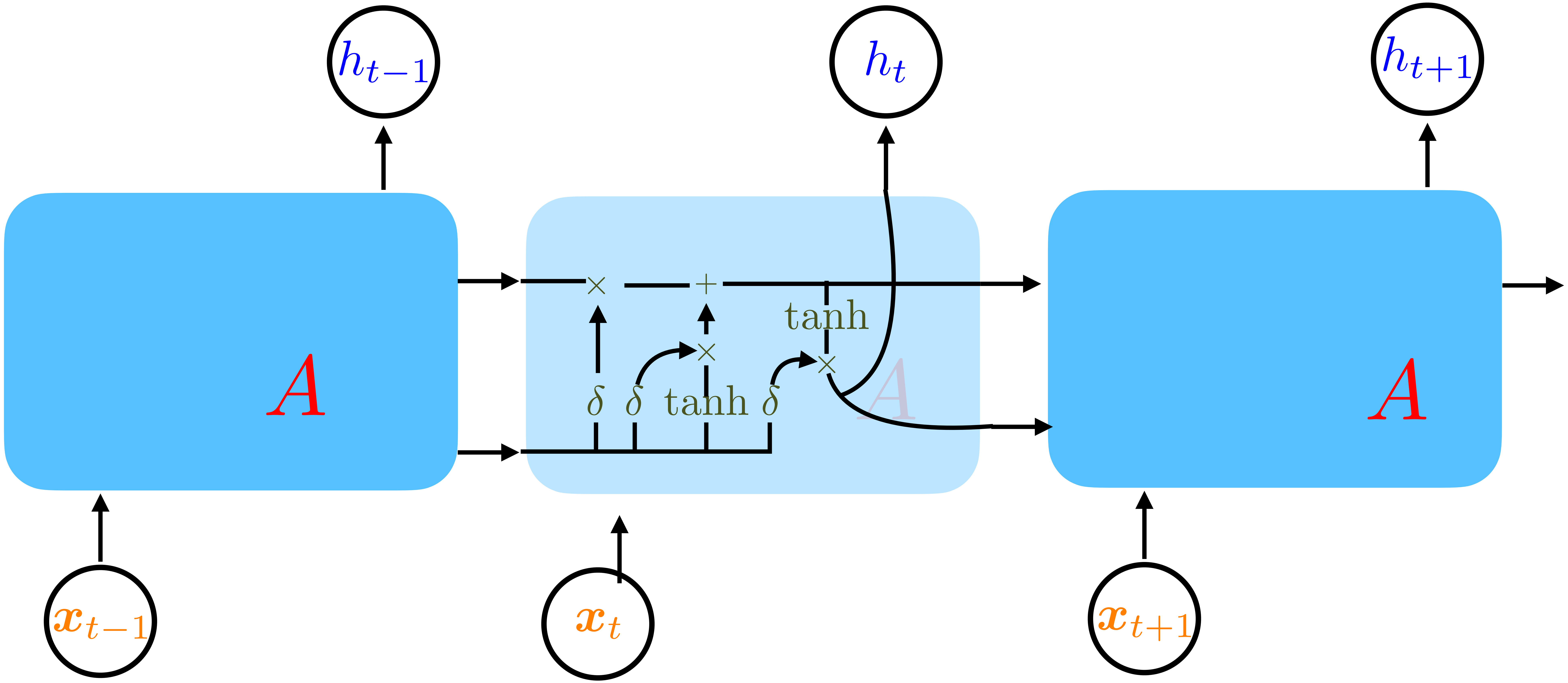
Vanishing Gradient in RNNs





In general, the longer the path, the smaller the gradient signal.

Long Short-term Memory (LSTM)



Long Short-term Memory (LSTM)

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{I}_x \mathbf{x}_t + \mathbf{I}_h \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{F}_x \mathbf{x}_t + \mathbf{F}_h \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{O}_x \mathbf{x}_t + \mathbf{O}_h \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot g(\mathbf{C}_x \mathbf{x}_t + \mathbf{C}_h \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{h}_t &= \mathbf{o}_t \odot g(\mathbf{c}_t)
\end{aligned}
$$

$$
\mathbf{y}_t = f(\mathbf{W}\mathbf{h}_t + \mathbf{b})
$$

Part-of-Speech Tagging

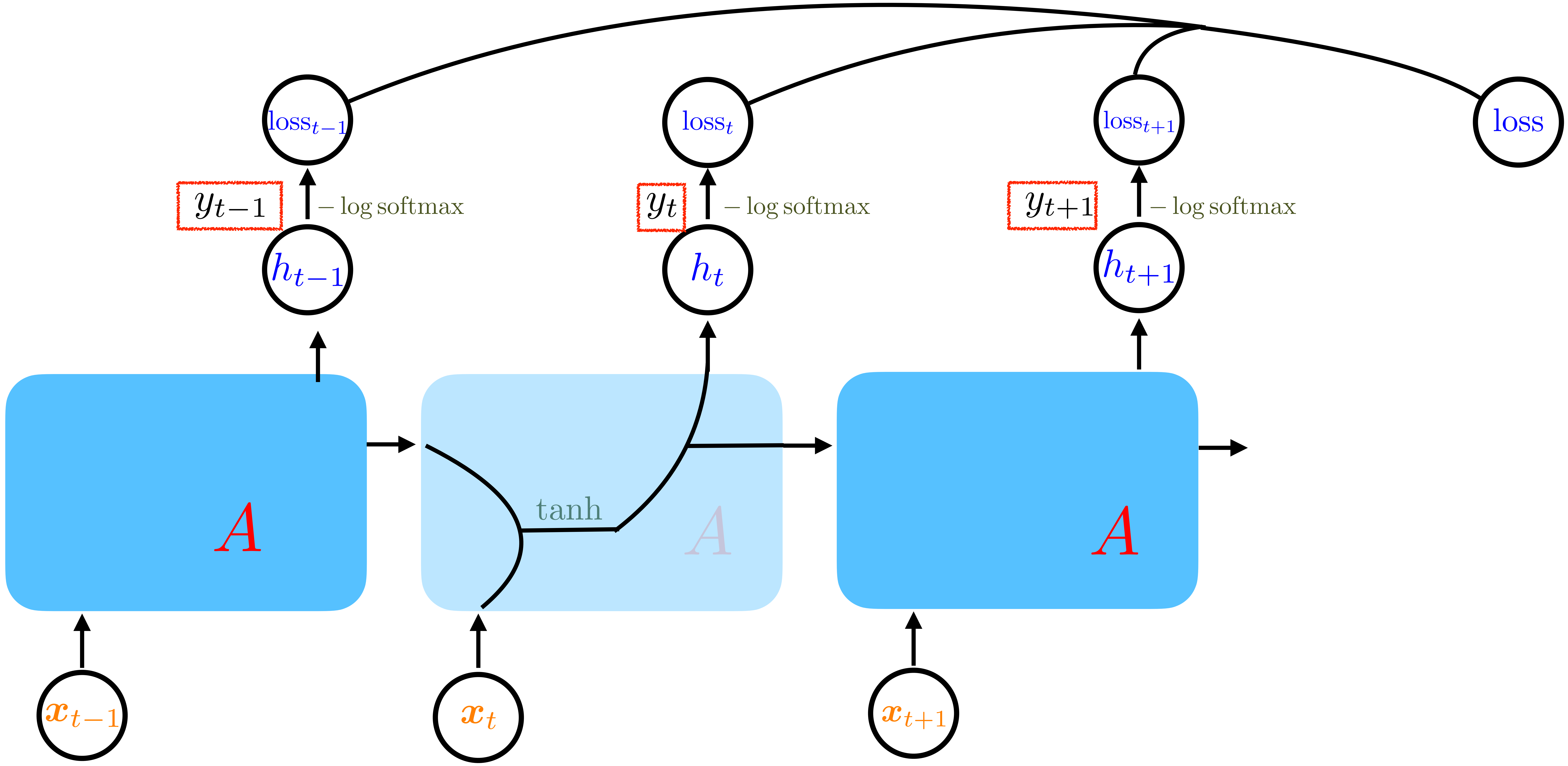
INPUT:
Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT:
Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

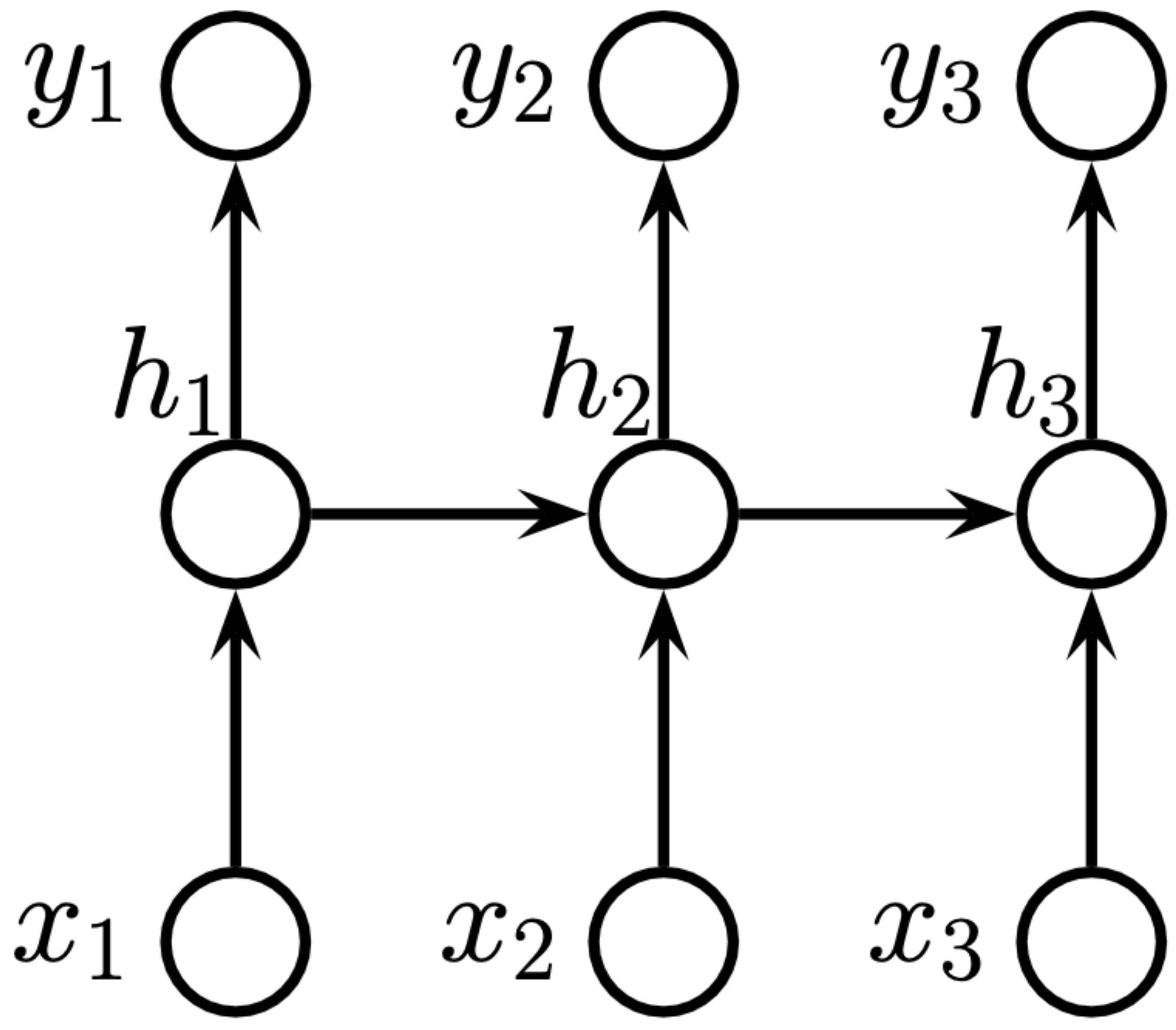
N	= Noun
V	= Verb
P	= Preposition
Adv	= Adverb
Adj	= Adjective

...

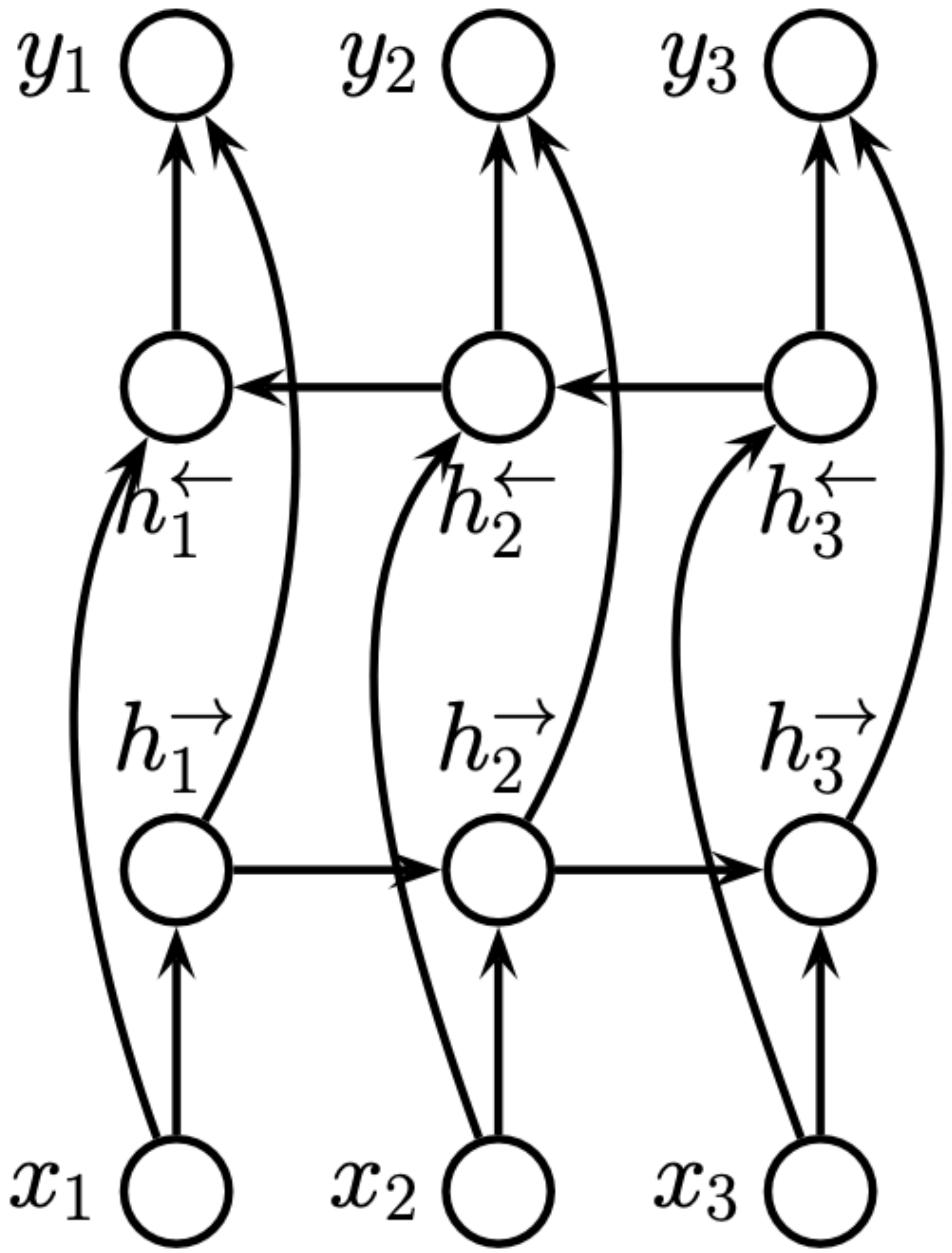
RNNs for Tagging



RNNs for Tagging



Left-to-right (Uni-directional)



Bidirectional