深度前馈神经网络

深度前馈网络（deep feedforward network），也叫作**前馈神经网络**（feedforward neural network）或者**多层感知机**（multilayer perceptron, MLP），是典型的深度学习模型。前馈网络的目标是近似某个函数 f^*。例如，对于分类器，$y = f^*(\boldsymbol{x})$ 将输入 $\boldsymbol{x}$ 映射到一个类别 y。前馈网络定义了一个映射 $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$，并且学习参数 $\boldsymbol{\theta}$ 的值，使它能够得到最佳的函数近似。

这种模型被称为**前向**（feedforward）的，是因为信息流过 $\boldsymbol{x}$ 的函数，流经用于定义 f 的中间计算过程，最终到达输出 $\boldsymbol{y}$。在模型的输出和模型本身之间没有**反馈**（feedback）连接。当前馈神经网络被扩展成包含反馈连接时，它们被称为**循环神经网络**（recurrent neural network），在第十章介绍。

- 感知机的限制
- 深度前馈神经网络
- 深度前馈神经网络的训练-BP算法
- 深度前馈神经网络设计

为了扩展线性模型来表示 $\boldsymbol{x}$ 的非线性函数，我们可以不把线性模型用于 $\boldsymbol{x}$ 本身，而是用在一个变换后的输入 $\phi(\boldsymbol{x})$ 上，这里 ϕ 是一个非线性变换。同样，我们可以使用第 5.7.2 节中描述的核技巧，来得到一个基于隐含地使用 ϕ 映射的非线性学习算法。我们可以认为 ϕ 提供了一组描述 $\boldsymbol{x}$ 的特征，或者认为它提供了 $\boldsymbol{x}$ 的一个新的表示。

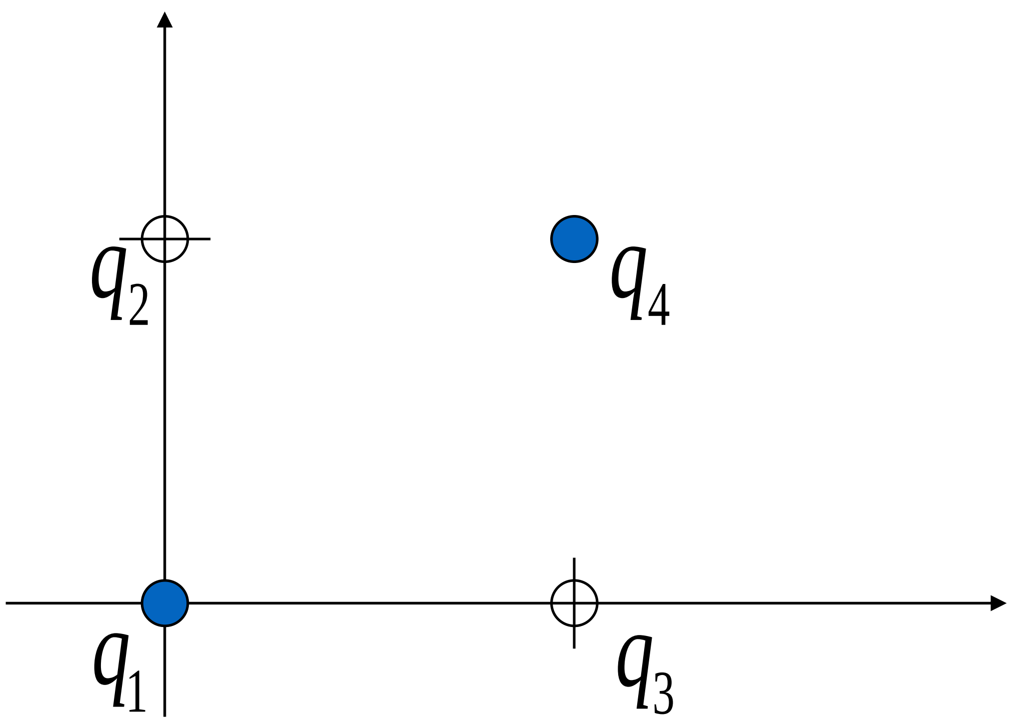
深度学习的策略是去学习 ϕ。在这种方法中，我们有一个模型 $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$。我们现在有两种参数：用于从一大类函数中学习 ϕ 的参数 $\boldsymbol{\theta}$，以及用于将 $\phi(\boldsymbol{x})$ 映射到所需的输出的参数 $\boldsymbol{w}$。这是深度前馈网络的一个例子，其中 ϕ 定义了一个隐藏层。这是三种方法中唯一一种放弃训练问题的凸性的方法，但是利大于弊。在这种方法中，我们将表示参数化为 $\phi(\boldsymbol{x}; \boldsymbol{\theta})$，并且使用优化算法来寻找 $\boldsymbol{\theta}$，使它能够得到一个好的表示。如果我们想要的话，这种方法也可以通过使它变得高度通用以获得第一种方法的优点——我们只需使用一个非常广泛的函数族 $\phi(\boldsymbol{x}; \boldsymbol{\theta})$。这种方法也可以获得第二种方法的优点。人类专家

感知机是线性分类器，当样本线性可分时，能收敛到一个线性分类器

当数据不是线性可分时，算法不收敛，无法找到线性分类器。

例子：XOR函数

XOR 问题



$$\left\{ q_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_1 = 0 \right\} \qquad \left\{ q_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_2 = 1 \right\}$$

$$\left\{ q_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_4 = 0 \right\} \qquad \left\{ q_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_3 = 1 \right\}$$

目标：拟合训练集。

$$\left\{ q_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_1 = 0 \right\} \quad \left\{ q_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_2 = 1 \right\}$$

模型：感知机

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b.$$

$$\left\{ q_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_4 = 0 \right\} \quad \left\{ q_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_3 = 1 \right\}$$

目标函数：MSE

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2.$$

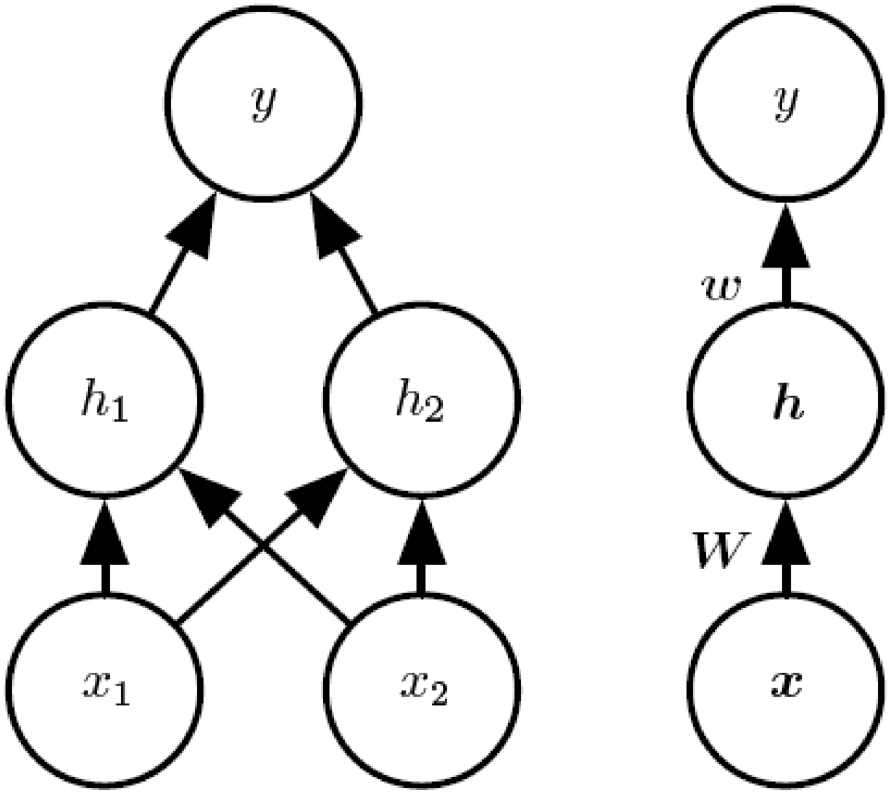
学习算法：LMS或直接最小化目标函数，得到闭式解：

W ＝（0, 0），b=0.5，无法拟合XOR函数

解决办法：设计两层的神经网络

$$\left\{q_1=\begin{bmatrix}0\\0\end{bmatrix},t_1=0\right\}\qquad\left\{q_2=\begin{bmatrix}0\\1\end{bmatrix},t_2=1\right\}$$

$$\left\{q_4=\begin{bmatrix}1\\1\end{bmatrix},t_4=0\right\}\qquad\left\{q_3=\begin{bmatrix}1\\0\end{bmatrix},t_3=1\right\}$$

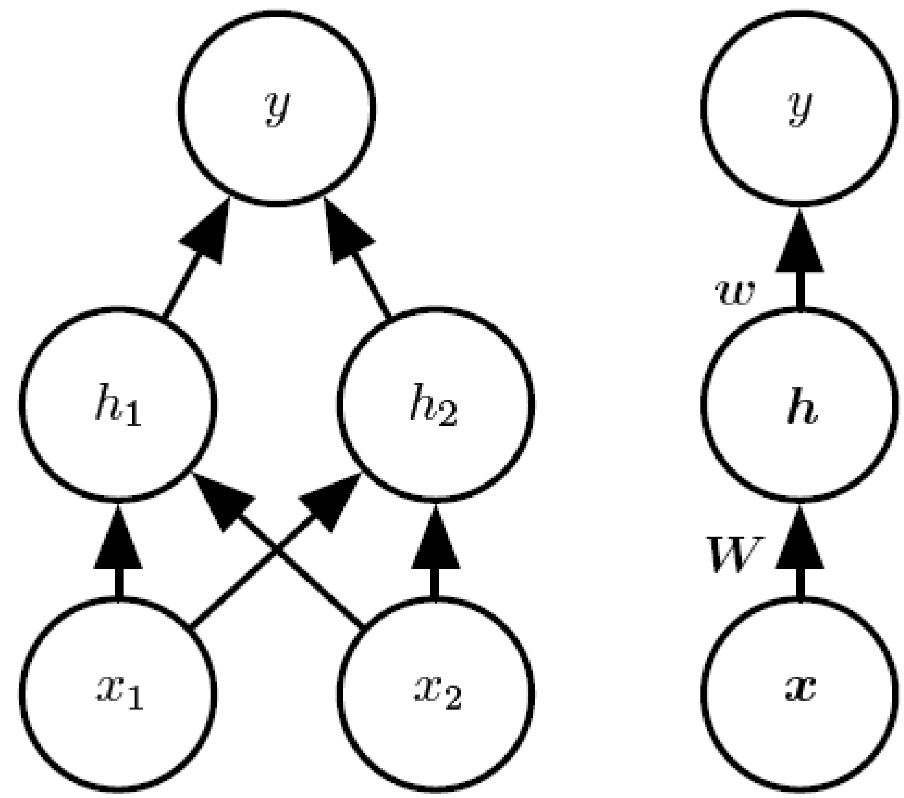


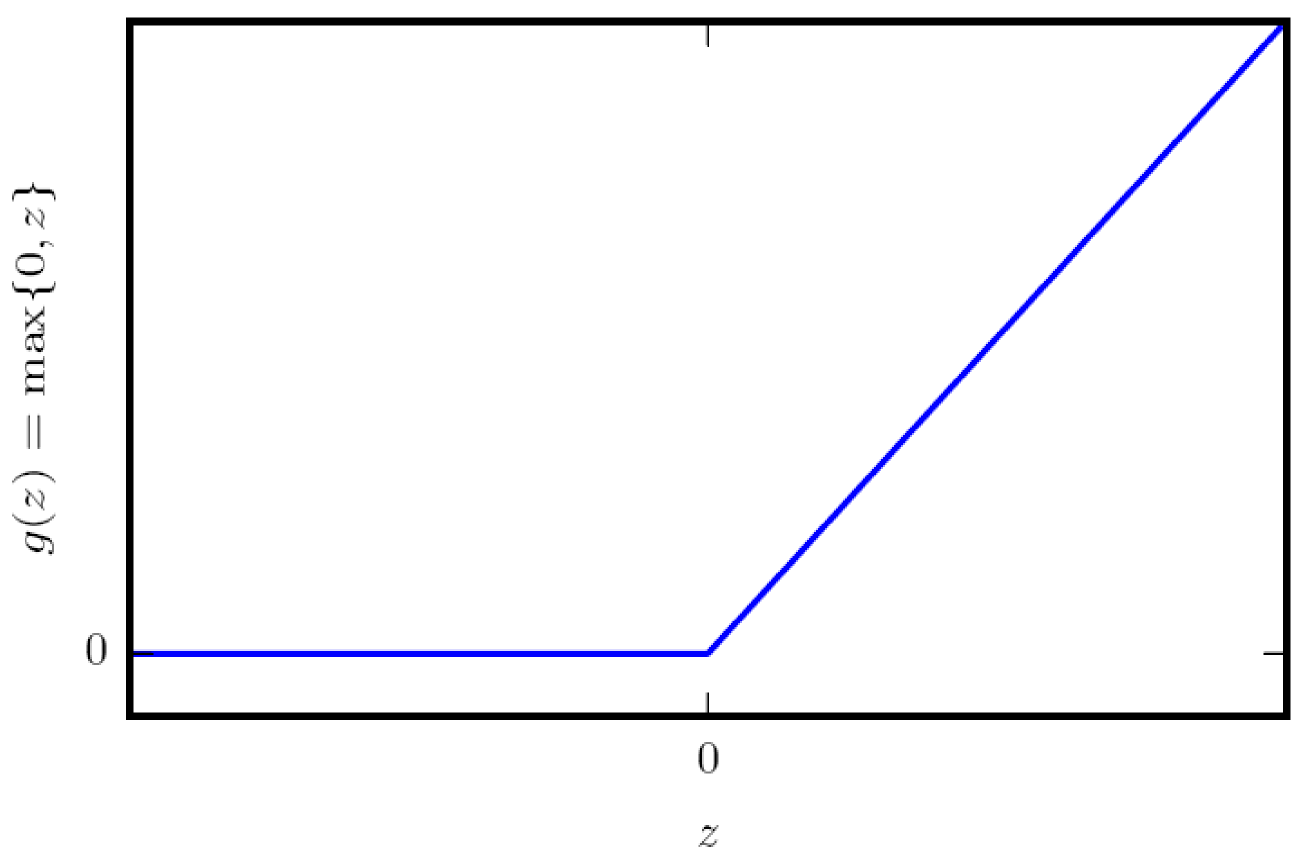
使用两种不同样式绘制的前馈网络的示例。它有单个隐藏层，包含两个单元。(左) 在这种样式中，每个单元绘制为图中的一个节点。(右)在这种样式中，将表示每一层激活的整个向量绘制为图中的一个节点。有时，图中的边使用参数名进行注释，用来描述两层之间的关系的。用矩阵**W 描述从x 到h 的映射，用向量w 描述从h 到y 的映射。当标记这种图时，我们通常省**略与每个层相关联的截距参数。

隐藏层的激活函数为：

ReLU（整流线性单元）

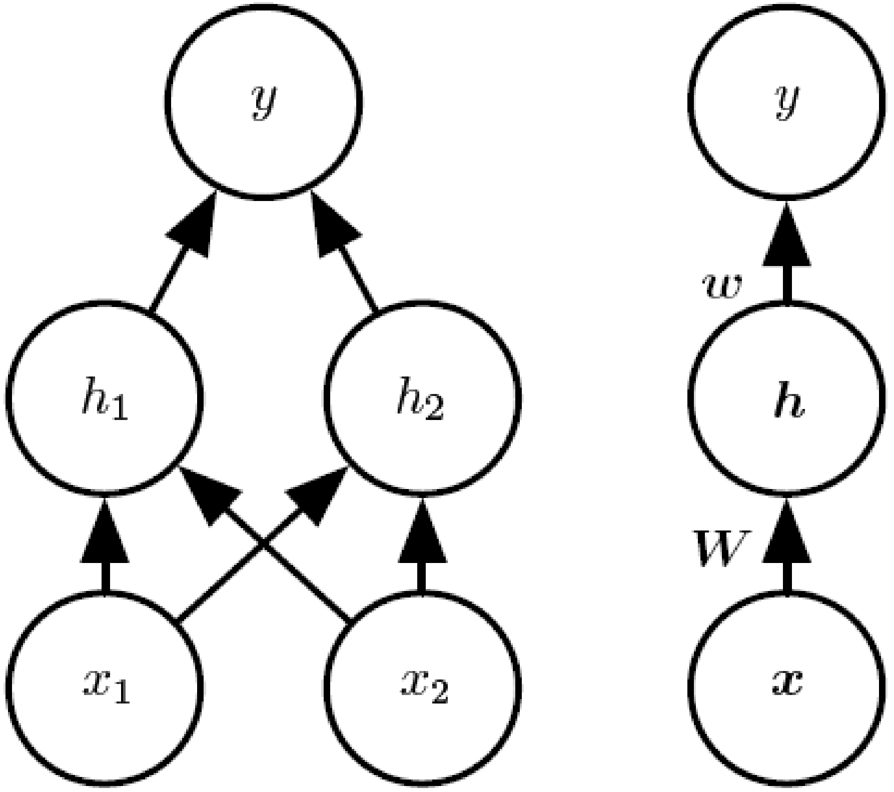
g（z）=max{0，z}





整个网络表示的函数为：

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

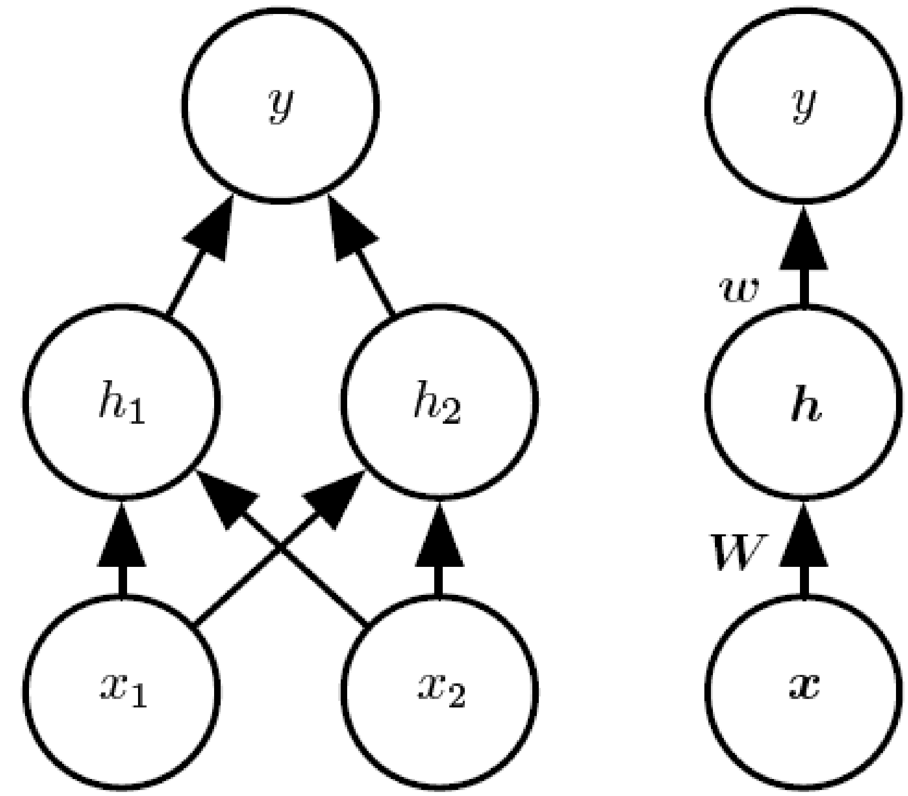


给定该网络一个解：

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$



其中b=0

该网络成功拟合XOR函数

分析：

设计矩阵为：

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

则输入与第一层的权值矩阵：

$$XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

加上偏置向量c:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

为隐藏层输入a，使用ReLU激活函数的输出h：

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

第二层：激活函数输出h乘以权重向量w后输出为：

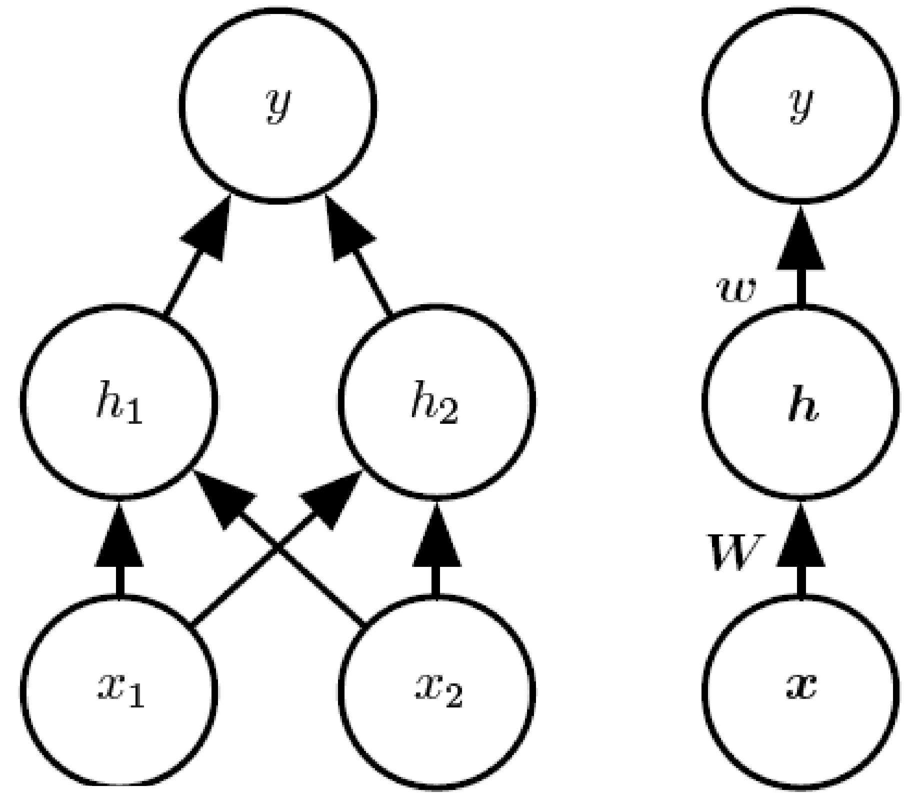
$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

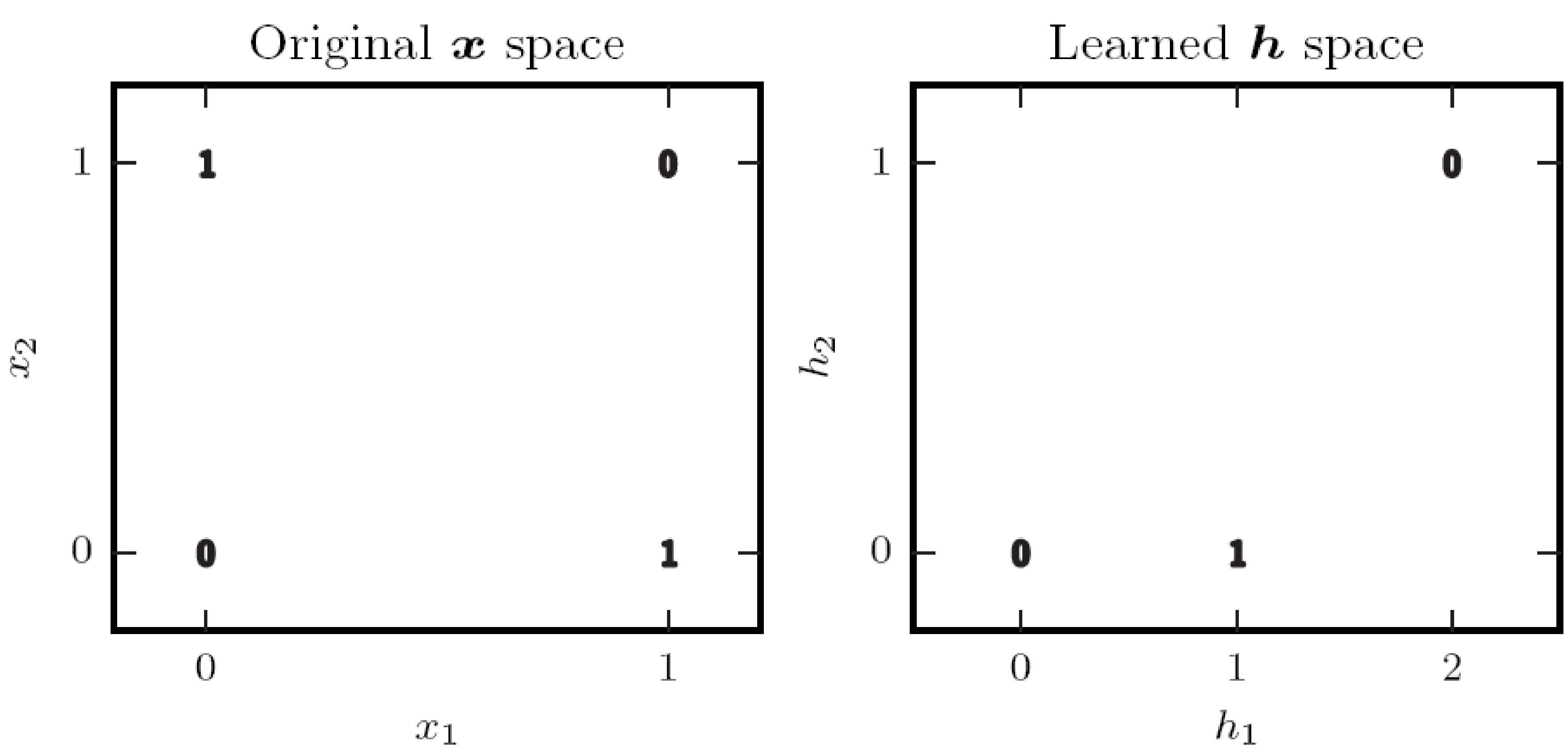
成功拟合XOR函数!

拟合过程：

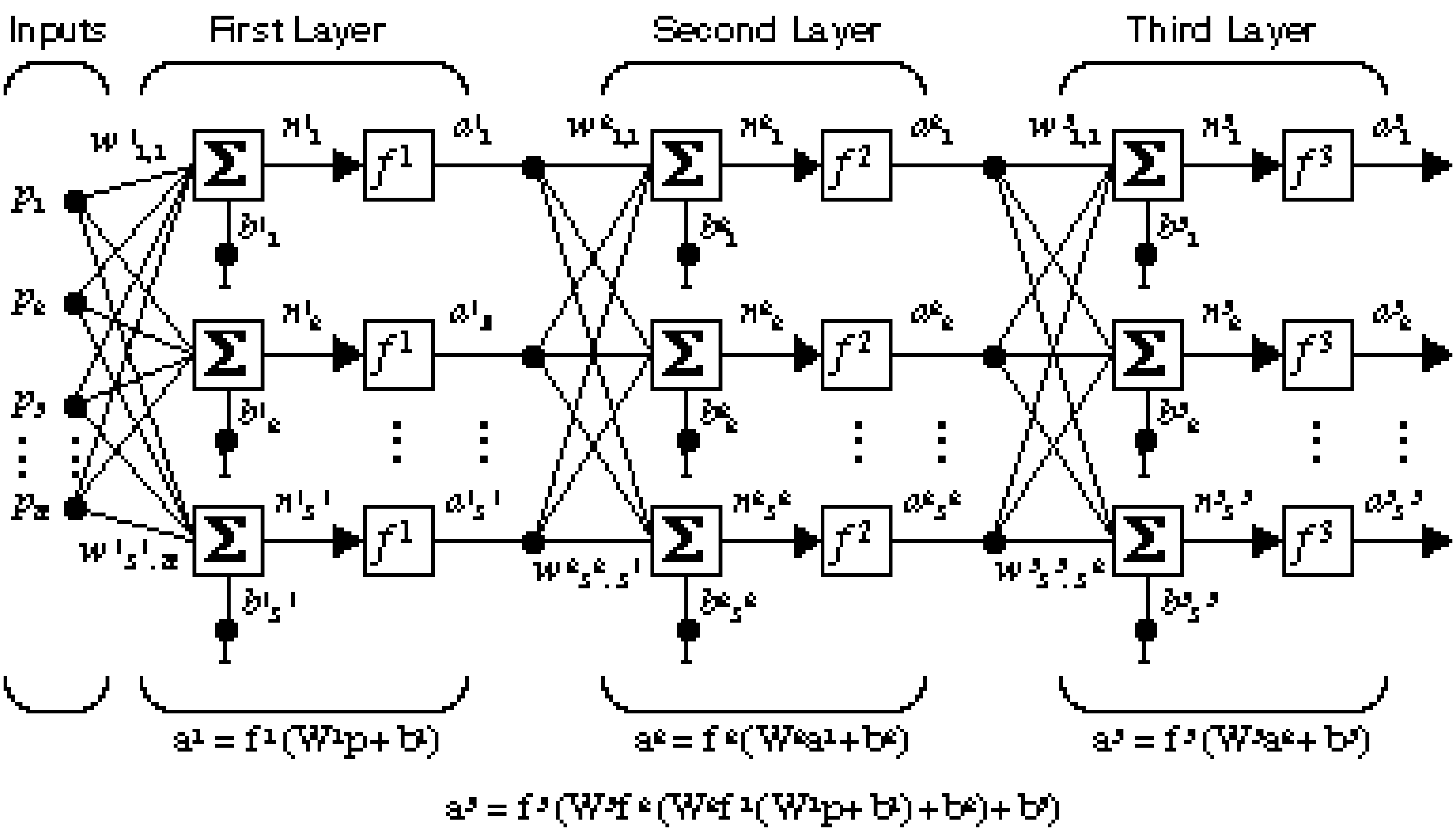
以第一层（隐藏层）输出为特征空间，

在特征空间中实现线性分类（输出层）



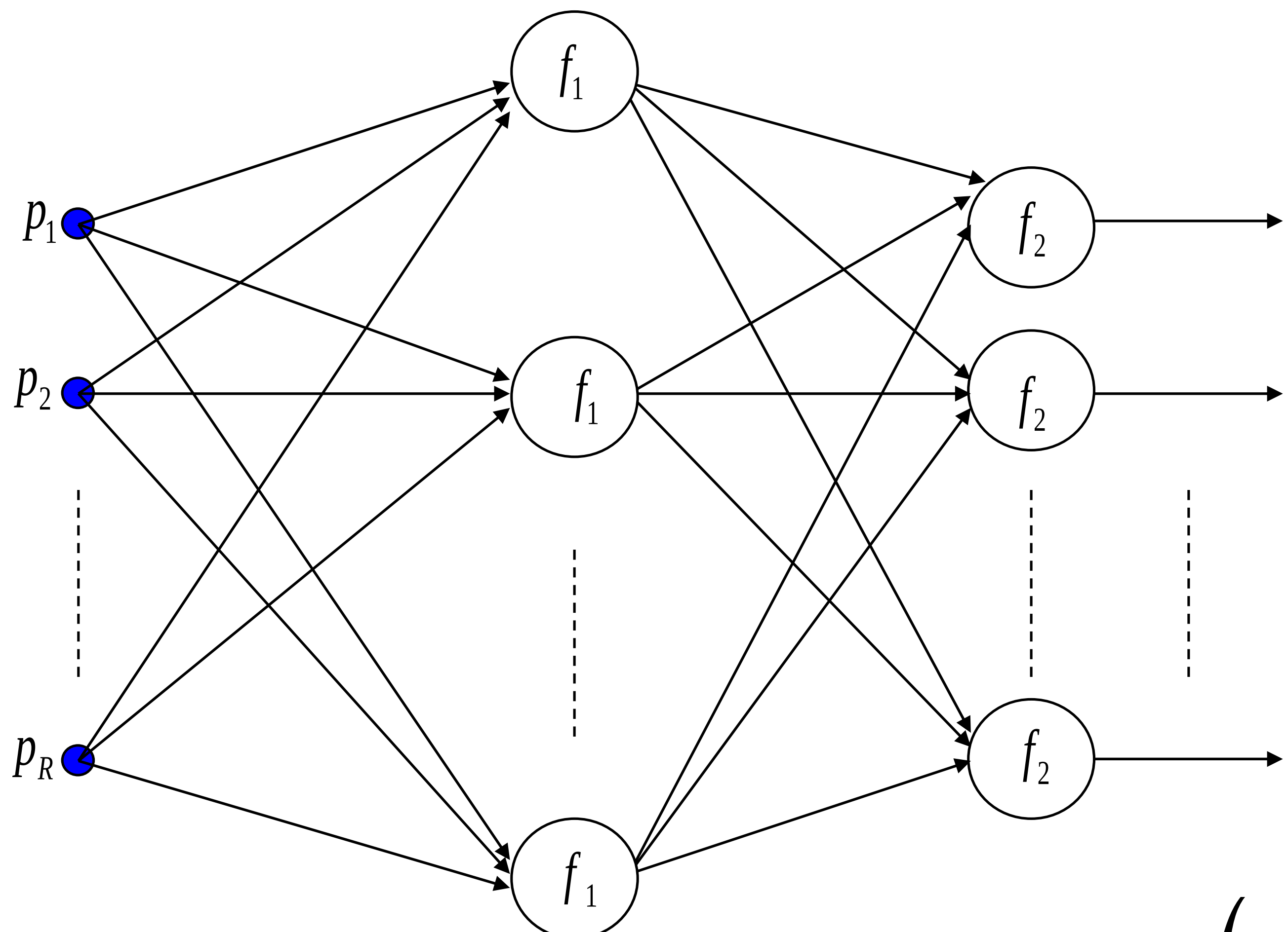


深度前馈神经网络，多层感知机（MLP）



$\mathbf{R - S^1 - S^2 - S^3}$ **Network**

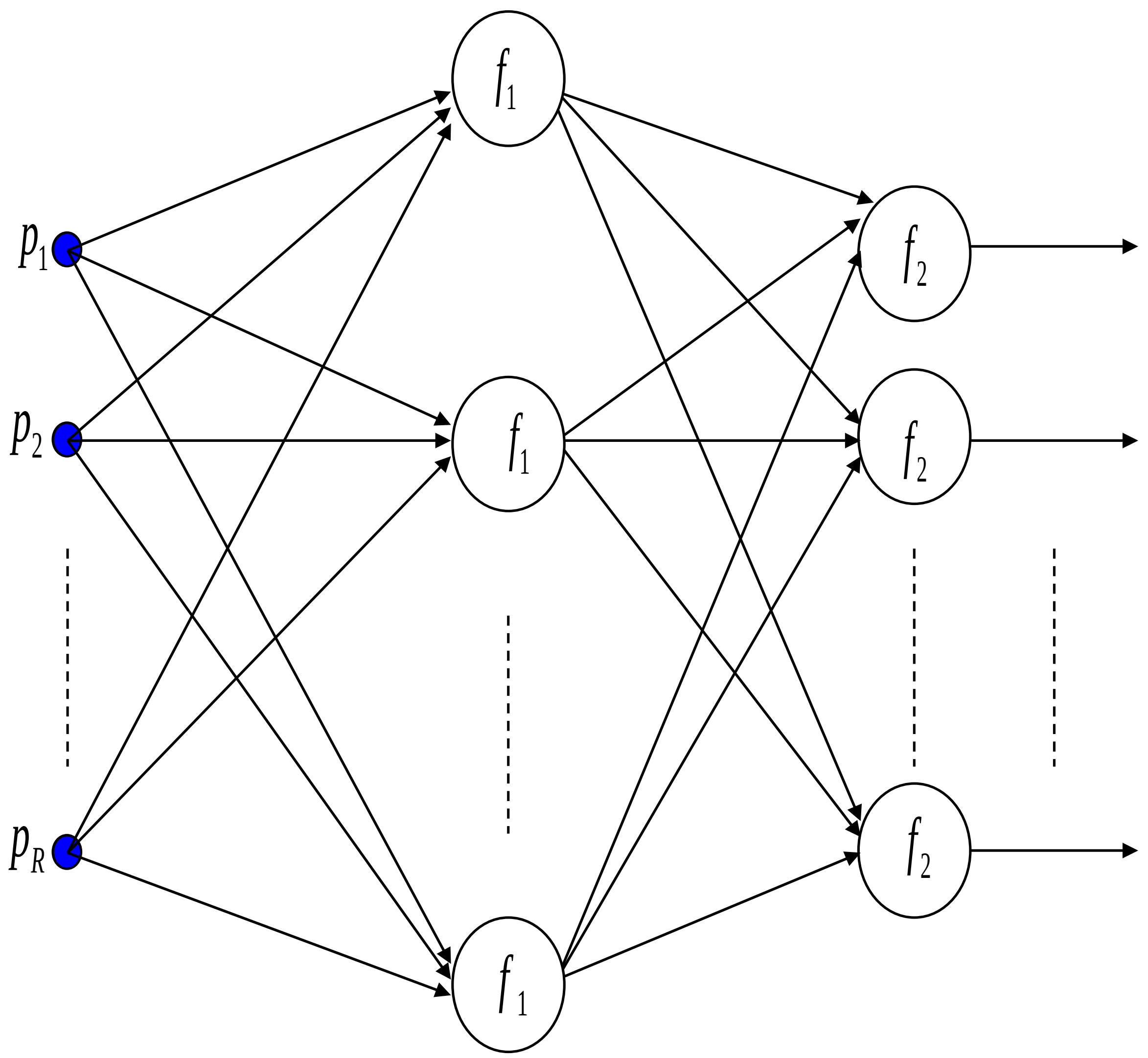
深度前馈神经网络，多层感知机（MLP）



$$a_2 = f_2\left(w_2{}^T a_1\right)$$

$$a_1 = f_1\left(w_1{}^T p\right)$$

深度前馈神经网络，多层感知机（MLP）



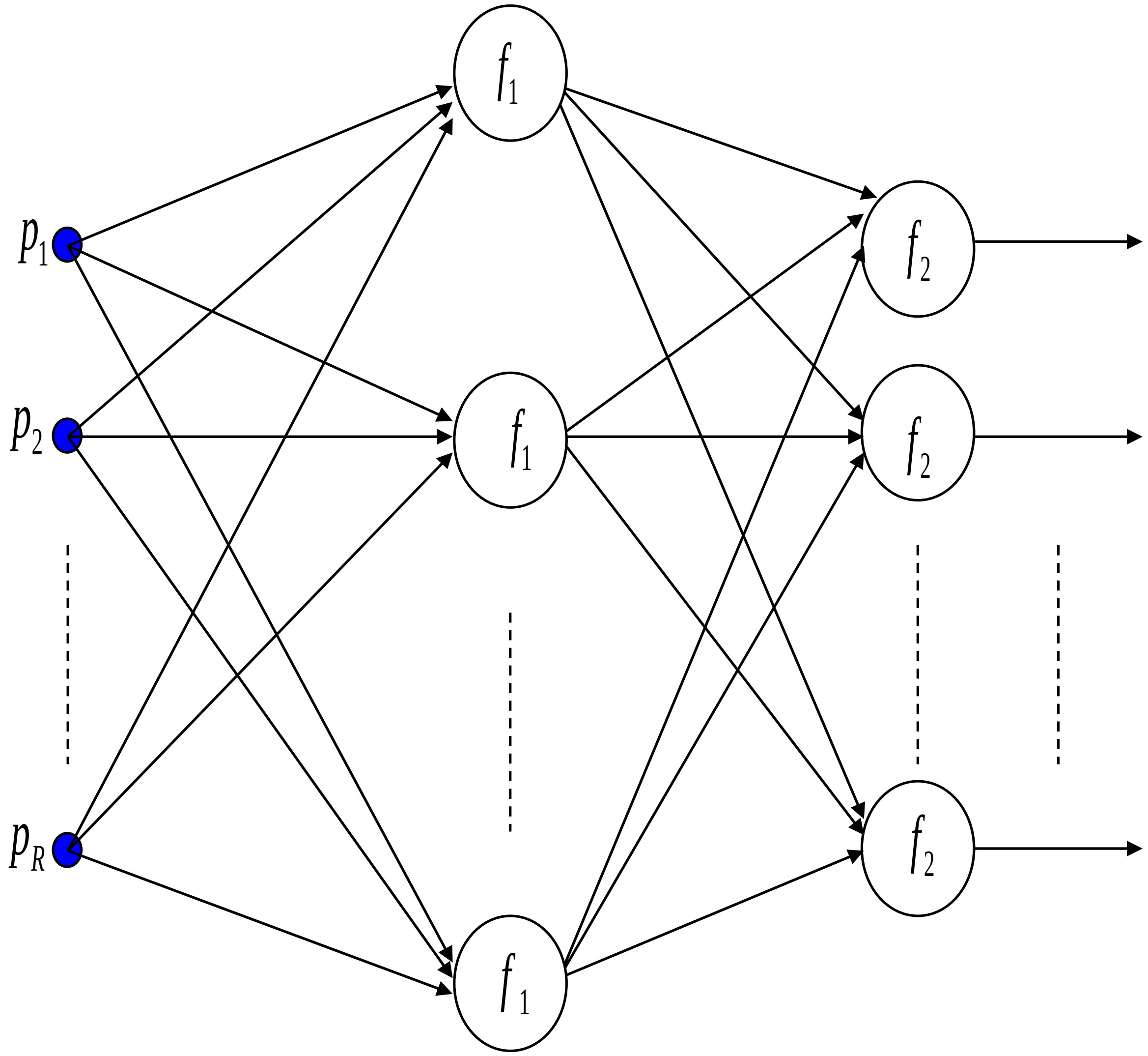
$$a_1 = f_1\left(w_1{}^T p\right)$$

$$a_2 = f_2\left(w_2{}^T a_1\right)$$

$$a_2 = f_2\left(w_2{}^T f_1\left(w_1{}^T p\right)\right)$$

$$a = f\left(WP\right)$$

如何训练M L P？



$$a_1 = f_1\left(w_1^T p\right)$$

$$a_2 = f_2\left(w_2^T a_1\right)$$

$$a_2 = f_2\left(w_2^T f_1\left(w_1^T p\right)\right)$$

$$a = f\left(WP\right)$$

Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\},\ \{\mathbf{p}_2, \mathbf{t}_2\},\ \ldots,\ \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$t_i = f\left(WP_i\right)$$

训练步骤：

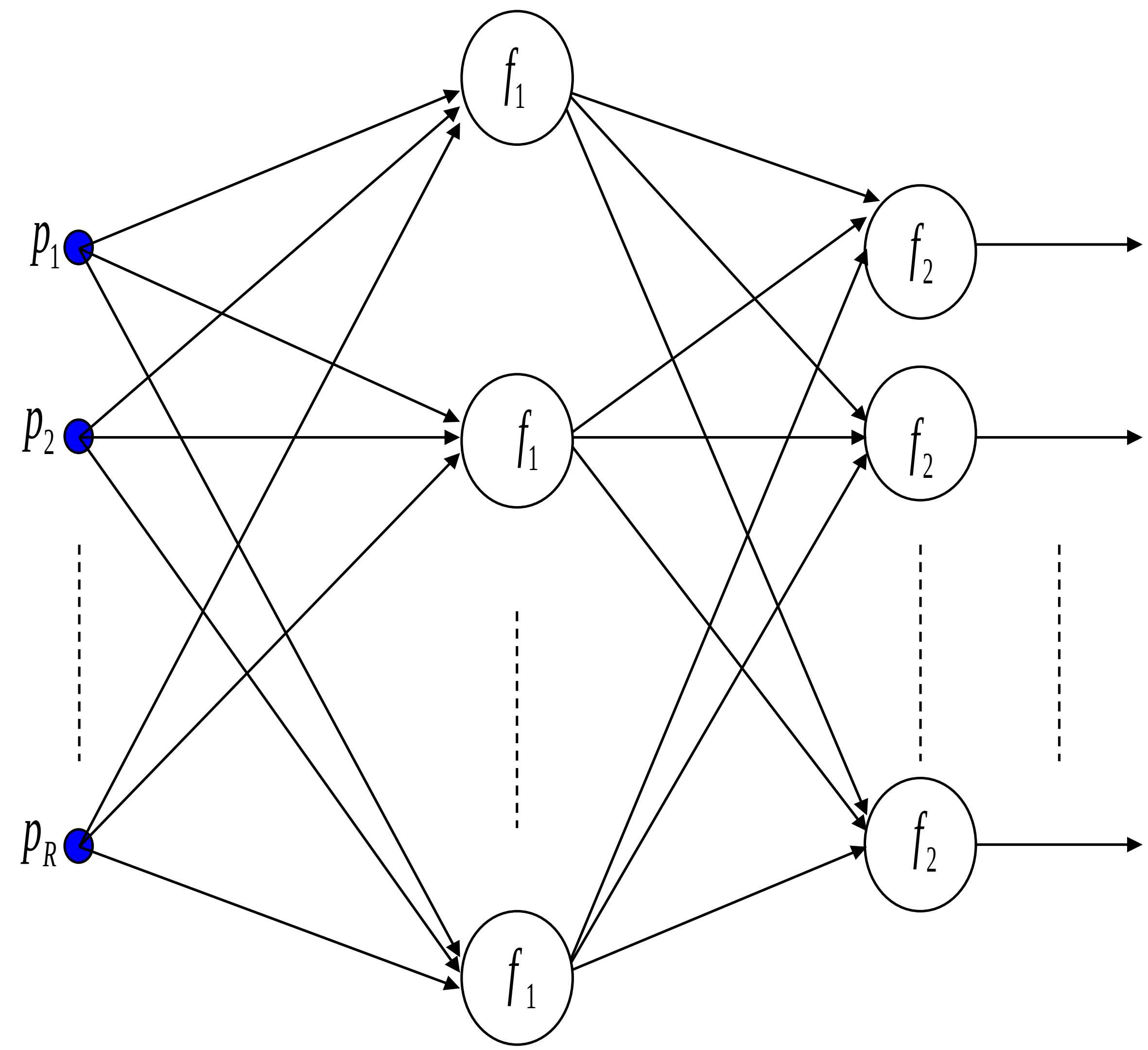
1.随机初始化：给网络参数W随机初始化权值，通常是接近0的随机初始化值；

2.前向计算：将样本本作为输入，利用：

$$a_2 = f_2\left(w_2{}^T f_1\left(w_1{}^T p\right)\right)$$

计算网络的输出值。

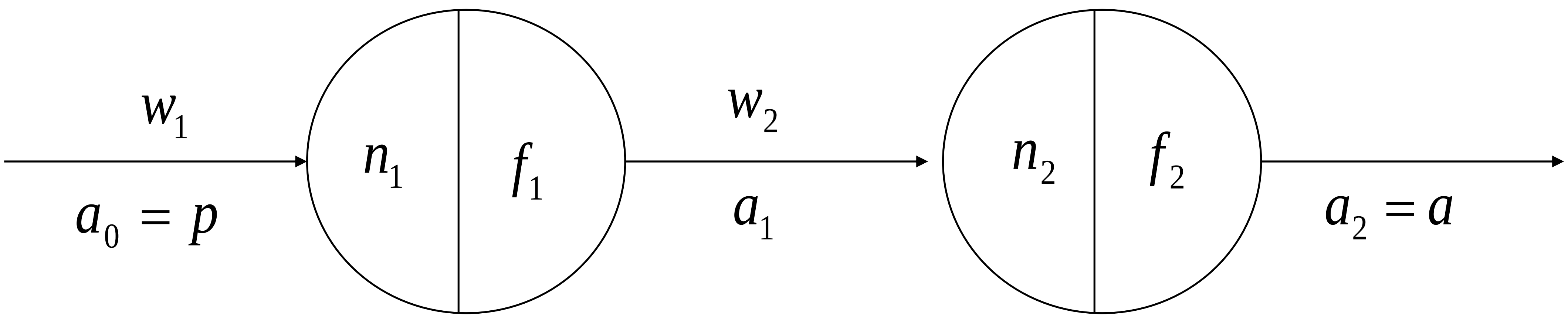
3.定义目标函数，采用BP算法更新权值。



Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

BP Principle

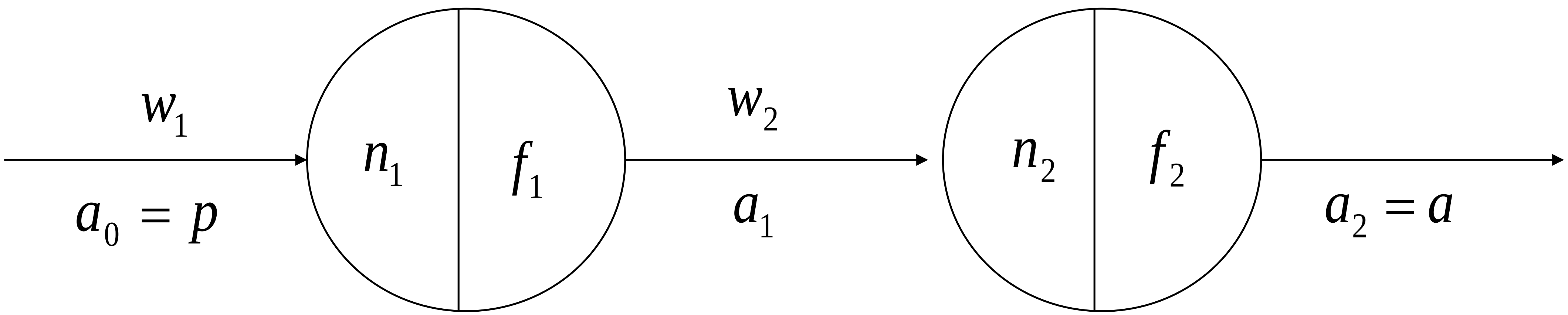


$$n_1 = w_1 a_0 \qquad n_2 = w_2 a_1$$

$$a_1 = f_1(n_1) \qquad a = a_2 = f_2(n_2)$$

19

BP Principle

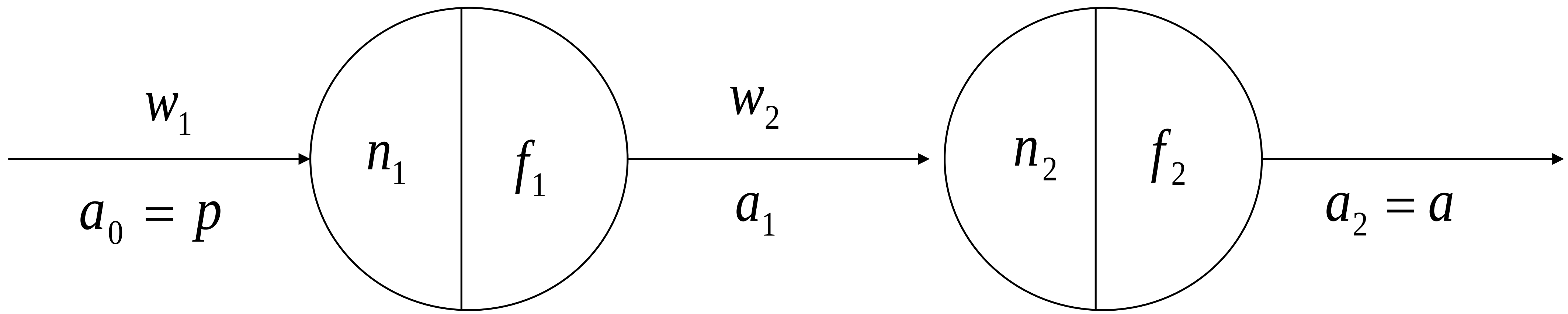


$$\{p_1, t_1\}, \{p_2, t_2\}, \cdots\cdots\cdots, \{p_q, t_q\}$$

$$E = E\left[(t-a)^2\right] = E(w_1, w_2) = E(n_1, n_2)$$

$$E \approx (t-a)^2$$

BP Principle



$$E \approx (t-a)^2$$

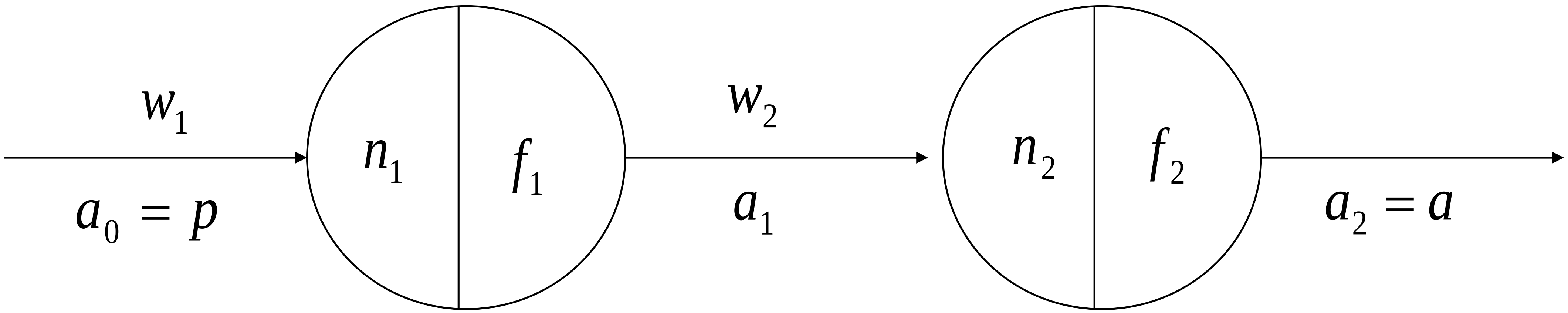
W?

$\Rightarrow$ **Steepest Descent**

$$w_1(k+1) = w_1(k) - \alpha \cdot \frac{\partial E}{\partial w_1} \qquad w_2(k+1) = w_2(k) - \alpha \cdot \frac{\partial E}{\partial w_2}$$

21

BP Principle



$$E = E(w_1, w_2) = E(n_1, n_2)$$

$$n_1 = w_1 a_0 \qquad n_2 = w_2 a_1$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial n_1} \cdot \frac{\partial n_1}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial n_2} \cdot \frac{\partial n_2}{\partial w_2}$$

Sensitivity

$$\frac{\partial E}{\partial n_1} = s_1$$

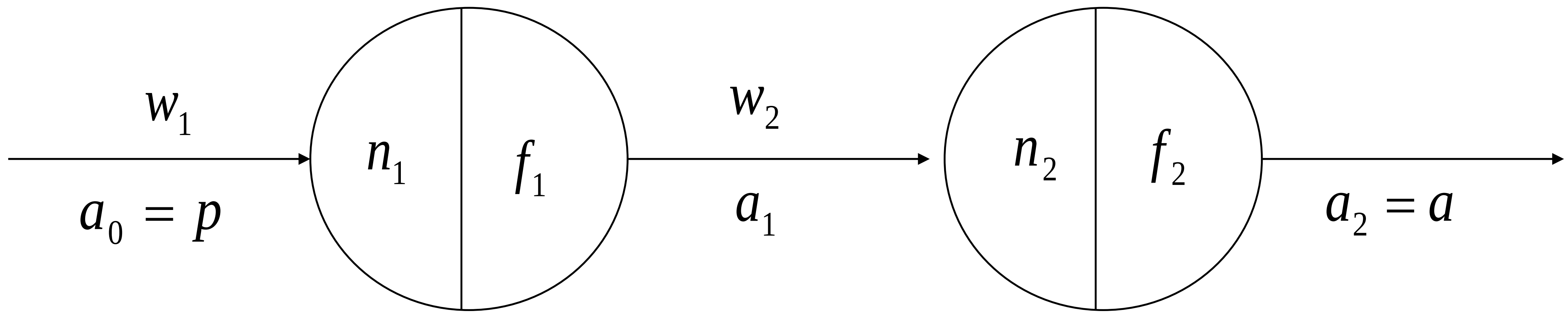
$$\frac{\partial E}{\partial n_2} = s_2$$

$$\frac{\partial n_1}{\partial w_1} = \frac{\partial (w_1 a_0)}{\partial w_1} = a_0$$

$$\frac{\partial n_2}{\partial w_2} = \frac{\partial (w_2 a_1)}{\partial w_2} = a_1$$

22

BP Principle



$$w_1(k+1) = w_1(k) - \alpha \cdot \frac{\partial E}{\partial w_1}$$

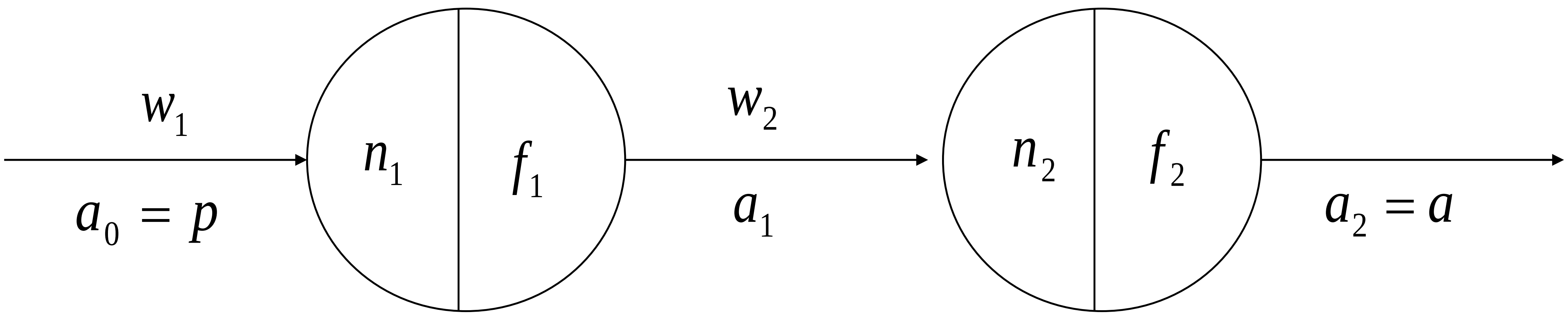
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial n_1} \cdot \frac{\partial n_1}{\partial w_1} = s_1 a_0$$

$$w_2(k+1) = w_2(k) - \alpha \cdot \frac{\partial E}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial n_2} \cdot \frac{\partial n_2}{\partial w_2} = s_2 a_1$$

23

BP Principle

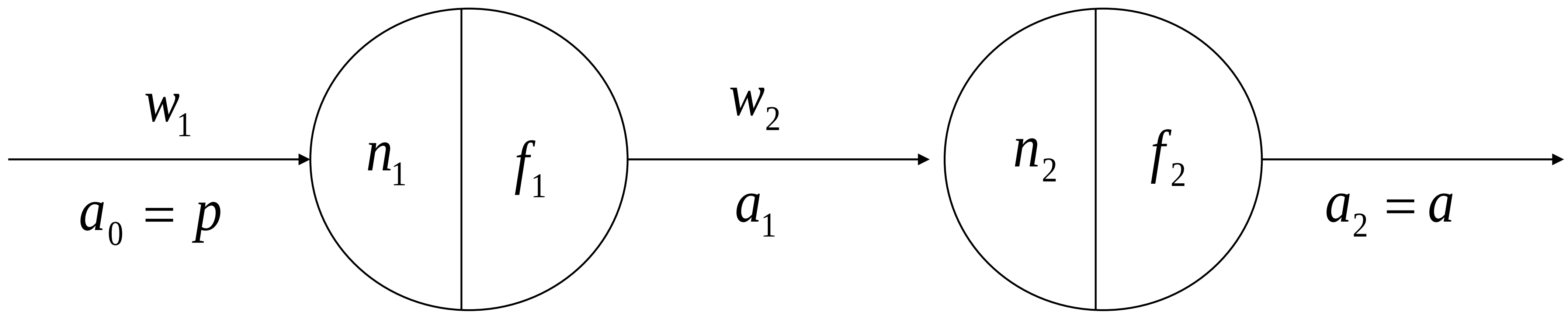


$$w_1(k+1) = w_1(k) - \alpha \cdot s_1 \cdot a_0$$

$$w_2(k+1) = w_2(k) - \alpha \cdot s_2 \cdot a_1$$

24

BP Principle



Sensitivity

$$\frac{\partial E}{\partial n_1} = s_1$$

$$\frac{\partial E}{\partial n_2} = s_2$$

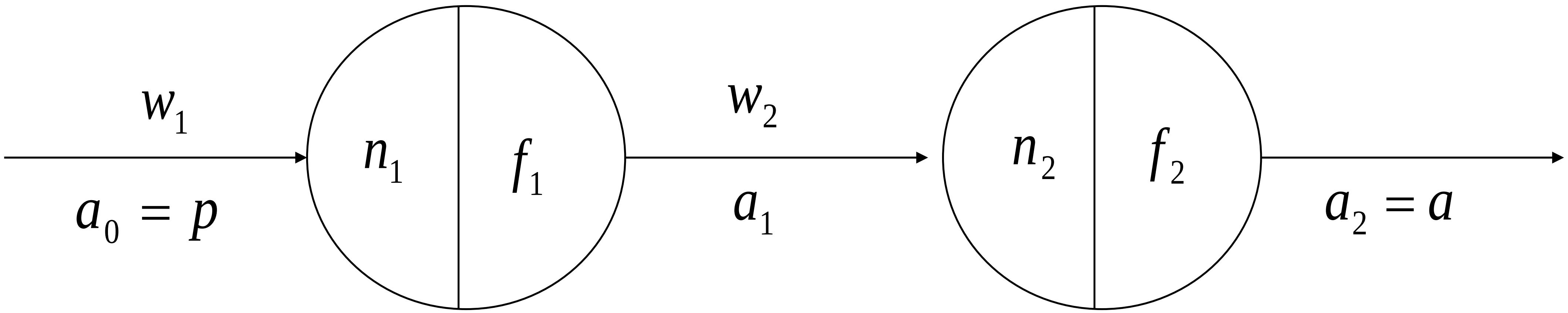
$$s_1 = \frac{\partial E}{\partial n_1} = \frac{\partial E}{\partial n_2} \cdot \frac{\partial n_2}{\partial n_1} = s_2 \cdot \frac{\partial n_2}{\partial n_1}$$

$$\frac{\partial n_2}{\partial n_1} = \frac{\partial (w_2 a_1)}{\partial n_1} = w_2 \cdot \frac{\partial a_1}{\partial n_1} = w_2 \cdot \frac{\partial f_1(n_1)}{\partial n_1} = w_2 \cdot \dot{f}_1(n_1)$$

$$\boxed{s_1 = \dot{f}_1(n_1) \cdot w_2 \cdot s_2}$$

25

BP Principle



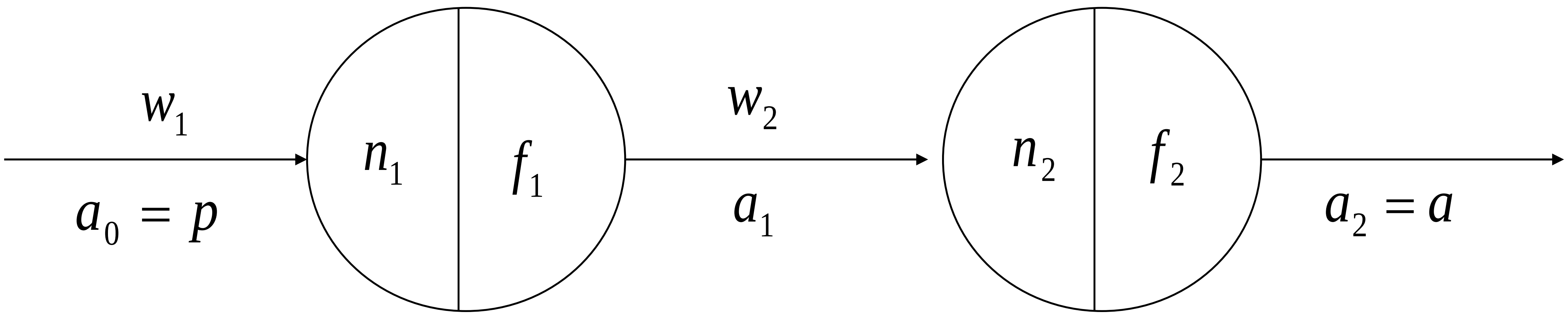
$$s_2 = \frac{\partial E}{\partial n_2} = \frac{\partial (t-a)^2}{\partial n_2} = -2(t-a)\cdot\frac{\partial a}{\partial n_2} = -2(t-a)\cdot\frac{\partial f_2(n_2)}{\partial n_2}$$

$$= -2(t-a)\cdot \dot{f}_2(n_2)$$

$$\boxed{s_2 = -2\cdot \dot{f}_2(n_2)\cdot(t-a)}$$

26

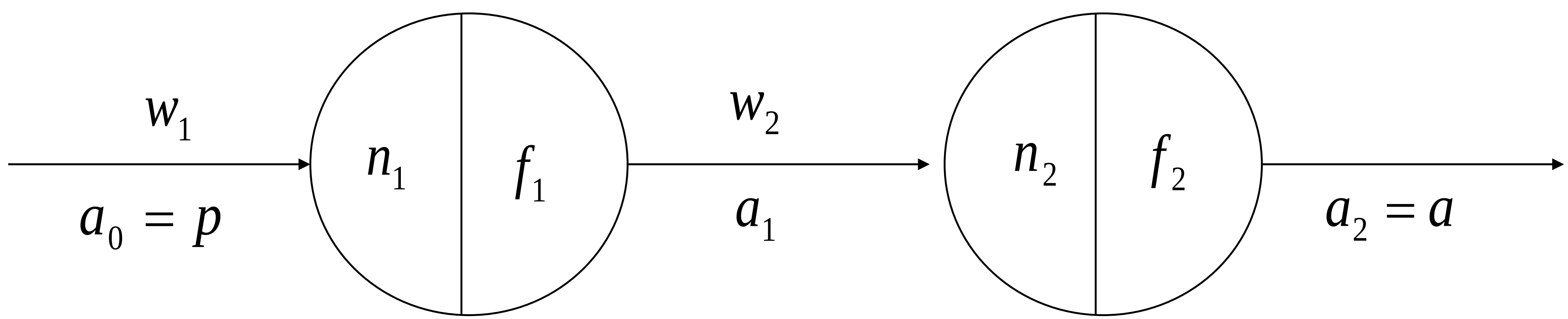
BP Principle



$$s_1 = \dot{f}_1(n_1) \cdot w_2 \cdot s_2 \Longleftarrow s_2 = -2 \cdot \dot{f}_2(n_2) \cdot (t - a)$$

27

BP Principle



$$a_0 = p(k) \Longrightarrow a_1 = f_1(w_1 a_0) \Longrightarrow a_2 = f_2(w_2 a_1)$$

$$\Downarrow$$

$$s_1 = \dot{f}_1(n_1) \cdot w_2 \cdot s_2 \Longleftarrow s_2 = -2 \cdot \dot{f}_2(n_2) \cdot (t(k) - a(k))$$

$$\Downarrow$$

$$w_1(k+1) = w_1(k) - \alpha \cdot s_1 \cdot a_0 \Longrightarrow w_2(k+1) = w_2(k) - \alpha \cdot s_2 \cdot a_1$$

BP Principle

Forward Propagation

$a_0 = p(k) \Longrightarrow a_1 = f_1(w_1 a_0) \Longrightarrow a_2 = f_2(w_2 a_1)$

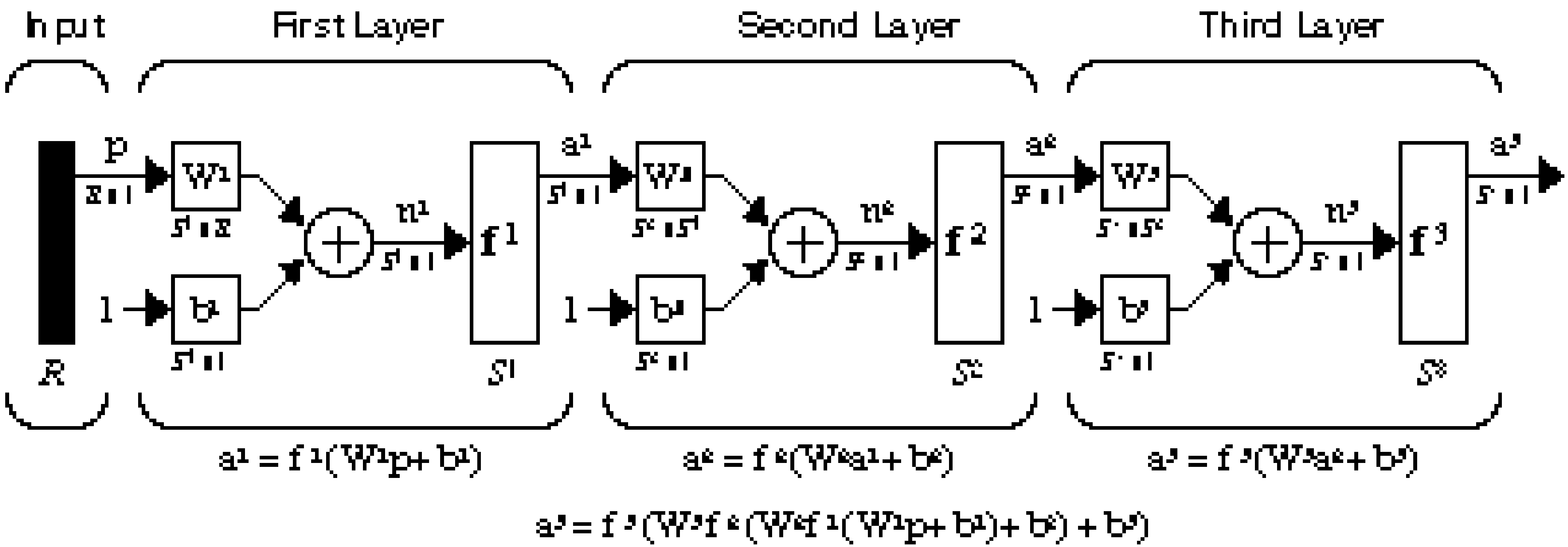
Backpropagation

$s_1 = \dot{f}_1(n_1) \cdot w_2 \cdot s_2 \Longleftarrow s_2 = -2 \cdot \dot{f}_2(n_2) \cdot (t(k) - a(k))$

Weight Update

$w_1(k+1) = w_1(k) - \alpha \cdot s_1 \cdot a_0 \Longrightarrow w_2(k+1) = w_2(k) - \alpha \cdot s_2 \cdot a_1$

Multilayer Network



$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^{0} = \mathbf{p} \qquad\qquad \mathbf{a} = \mathbf{a}^{M}$$

Multilayer Network

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

31

Performance Index

Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha\frac{\partial\hat{F}}{\partial w_{i,j}^m} \qquad b_i^m(k+1) = b_i^m(k) - \alpha\frac{\partial\hat{F}}{\partial b_i^m}$$

32

Chain Rule

Approximate Steepest Descent

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{m}} \qquad b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial b_{i}^{m}}$$

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_{i}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial b_{i}^{m}}$$

Gradient Calculation

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Steepest Descent

$$w_{i,j}^m(k+1) = w_{i,j}^m(\mathrm{k}) - \alpha s_i^m a_j^{m-1} \qquad b_i^m(k+1) = b_i^m(k) - \alpha s_i^m$$

$$\boxed{\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m}$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \dfrac{\partial \hat{F}}{\partial n_1^m} \\ \dfrac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \dfrac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m}$$

$$= w_{i,j}^{m+1} \cdot \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \cdot \dot{f}^m(n_j^m)$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

Jacobian Matrix

$$\frac{\partial n^{m+1}}{\partial n^m} = W^{m+1} \cdot \dot{F}^m(n^m) \qquad \dot{F}^m(n^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dot{f}^m(n_{s^m}^m) \end{bmatrix}$$

Backpropagation (Sensitivities)

$$s^m = \frac{\partial \hat{F}}{\partial n^m} = \left[\frac{\partial n^{m+1}}{\partial n^m}\right]^T \frac{\partial \hat{F}}{\partial n^{m+1}} = \dot{F}^m(n^m) \cdot \left(W^{m+1}\right)^T \cdot \frac{\partial \hat{F}}{\partial n^{m+1}}$$

$$\boxed{s^m = \dot{F}^m(n^m) \cdot \left(W^{m+1}\right)^T \cdot s^{m+1}}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Initialization (Last Layer)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

$$s_i^M = -2(t_i - a_i)\,\dot{f}^M(n_i^M)$$

$$\boxed{s^M = -2\dot{F}^M(n^M)\cdot(t-a)}$$

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

$$s^M = -2\dot{F}^M(n^M)(t-a)$$

$$s^m = \dot{F}^M(n^m)\left(W^{m+1}\right)^T s^{m+1} \qquad m = M-1, \ldots, 2, 1$$

Weight Update

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

ＢＰ算法总结：

设代价函数为Ｃ，则ＢＰ算法核心是下列四个公式：

总结：反向传播的四个方程式

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \qquad \text{(BP4)}$$

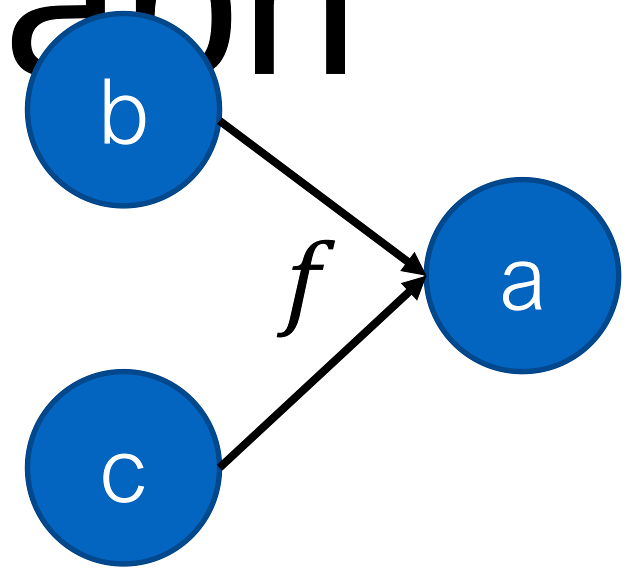
其中：

$\delta^L = s^l$， $\sigma(z^L) = f(a^L)$

Computational Graph

Computational Graph

$$a = f(b, c)$$



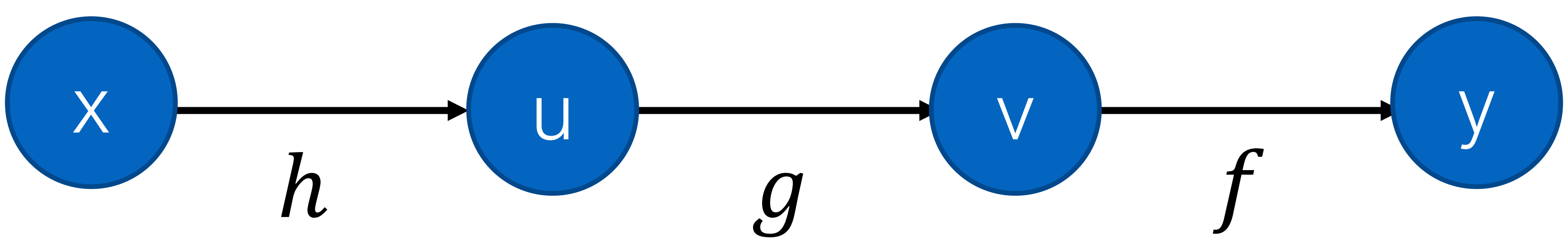
- A “language” describing a function

Example $y = f\left(g\left(h(x)\right)\right)$

- ***Node***: variable (scalar, vector, tensor ……)

$$u = h(x) \qquad v = g(u) \qquad y = f(v)$$

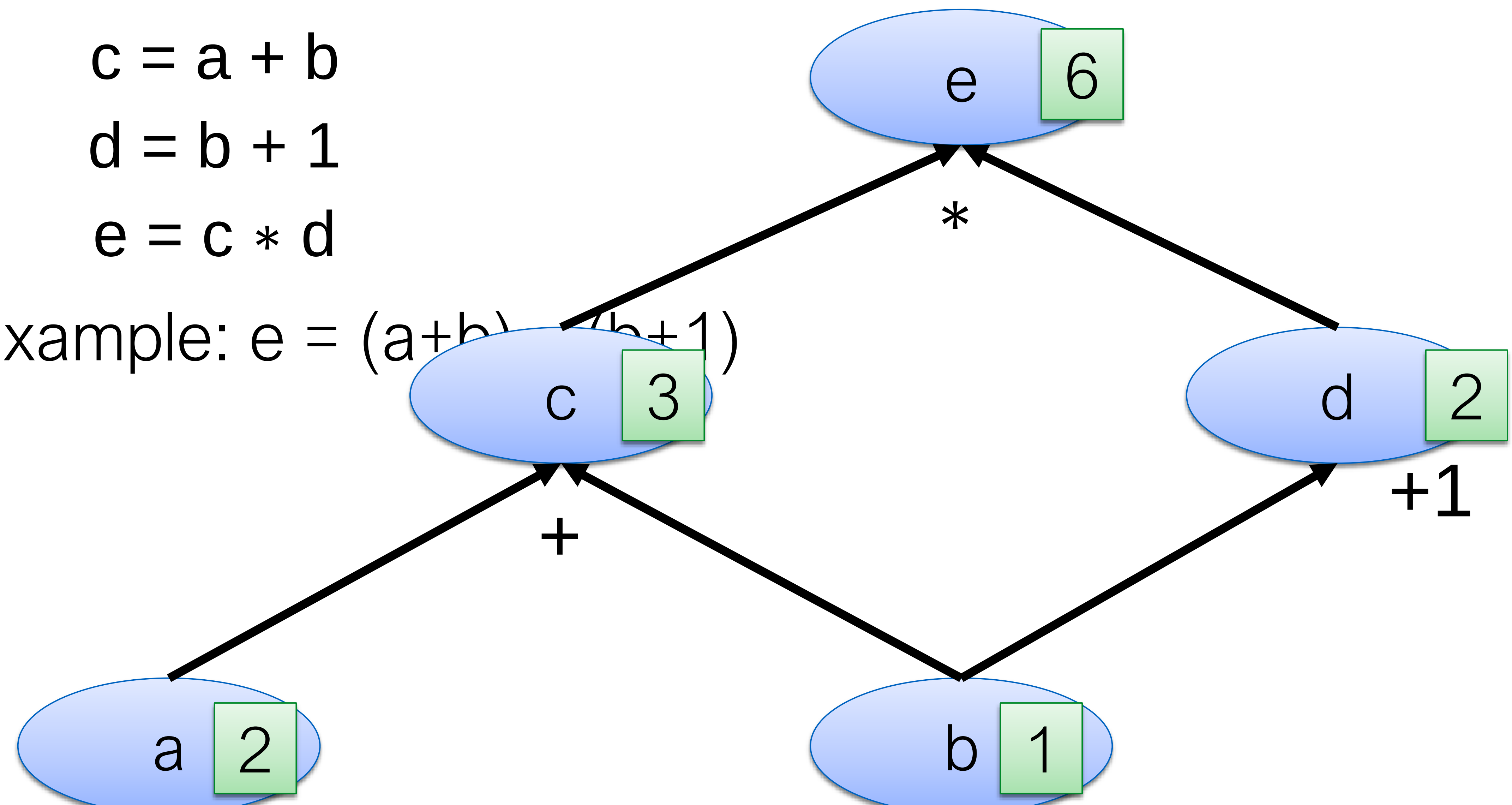
- ***Edge***: operation (simple function)



Computational Graph

$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$

- Example: e = (a+b) ∗ (b+1)



Review: Chain Rule

Case 1

$$z = f(x) \quad \Rightarrow \quad y = g(x) \quad z = h(y)$$

$$x \xrightarrow{g} y \xrightarrow{h} z$$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx}$$

Case 2

$$z = f(s) \quad \Rightarrow \quad x = g(s) \quad y = h(s) \quad z = k(x, y)$$

$$\begin{tikzcd} & x \arrow[dr] & \\ s \arrow[ur] \arrow[dr] & & z \\ & y \arrow[ur] & \end{tikzcd} \qquad \begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd} \qquad \frac{dz}{ds}$$

Computational Graph

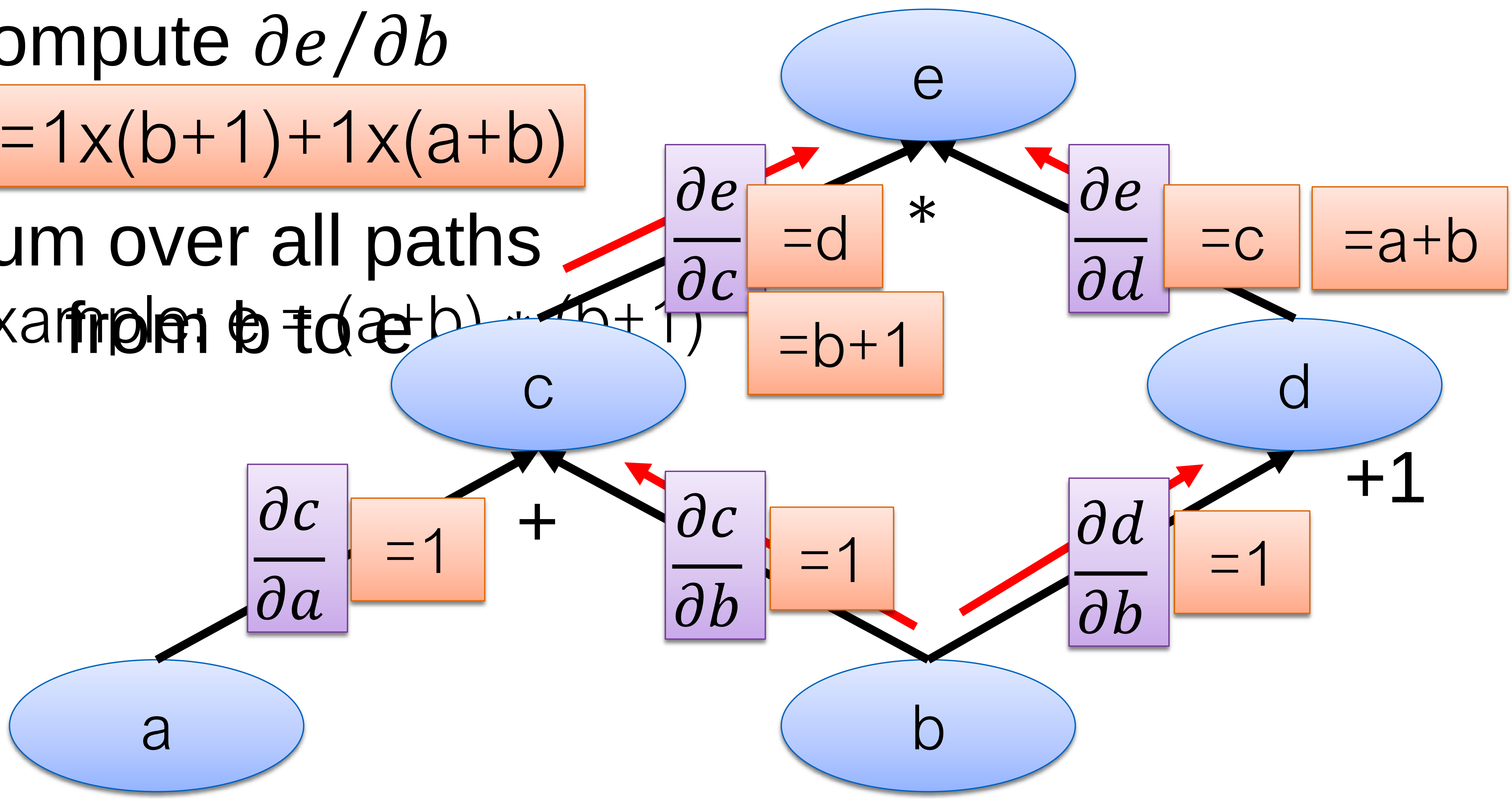
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

Compute $\partial e / \partial b$

=1x(b+1)+1x(a+b)

Sum over all paths from b to e

- Example: e = (a+b) ∗ (b+1)



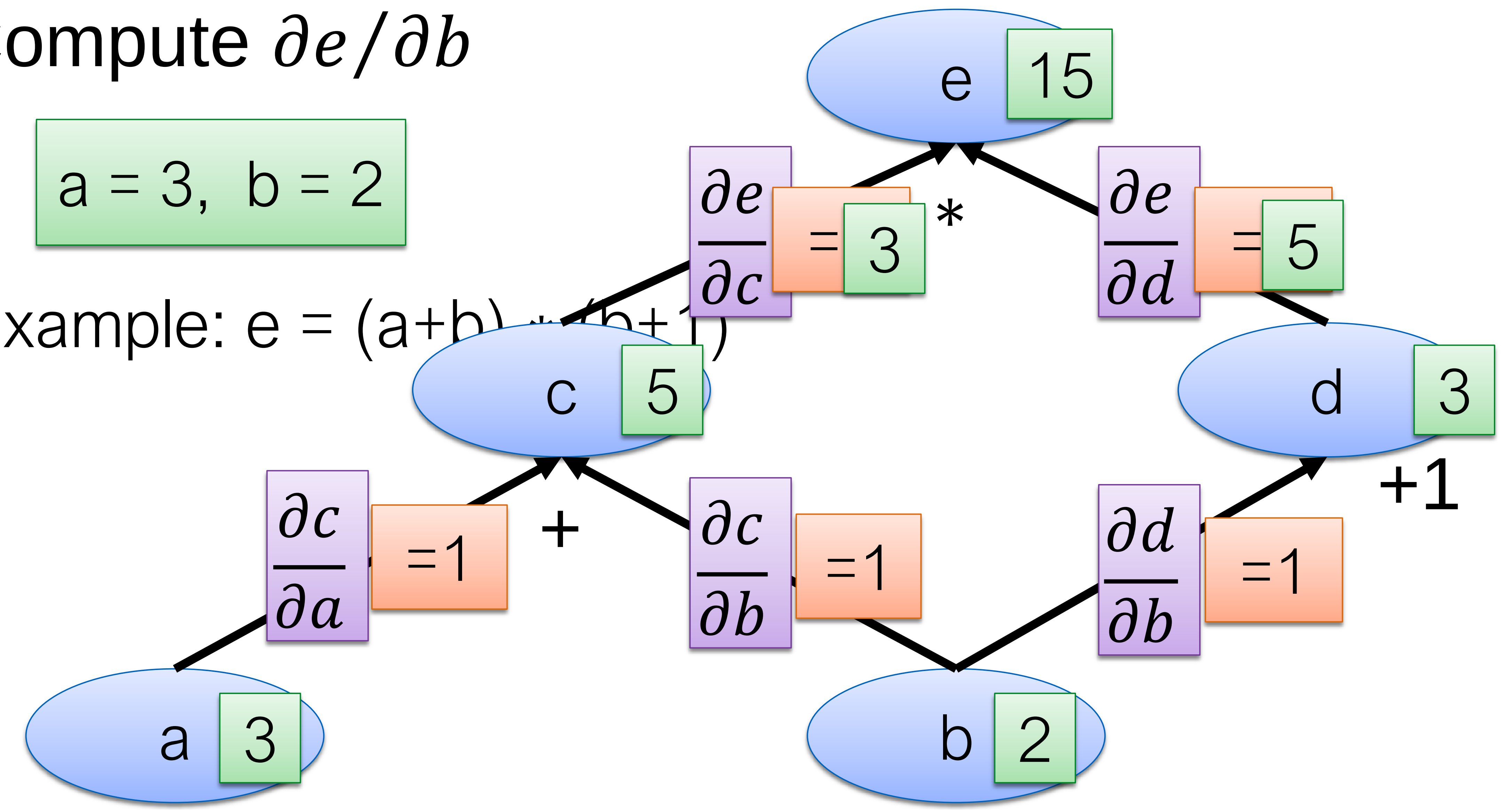
Computational Graph

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

Compute $\partial e / \partial b$

a = 3, b = 2

- Example: e = (a+b) * (b+1)

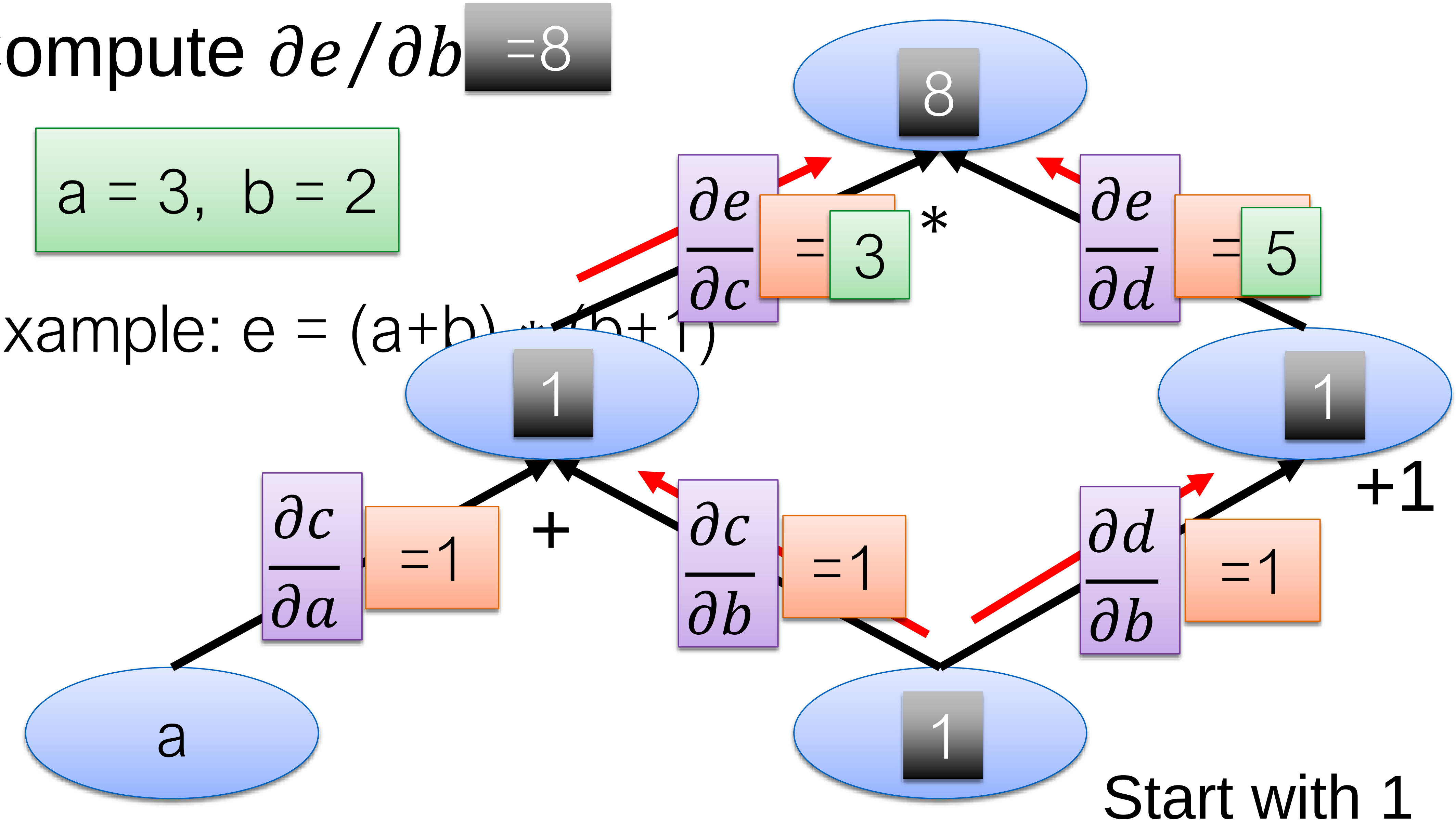


Computational Graph

Compute $\partial e/\partial b$ =8

a = 3, b = 2

- Example: e = (a+b) ∗ (b+1)

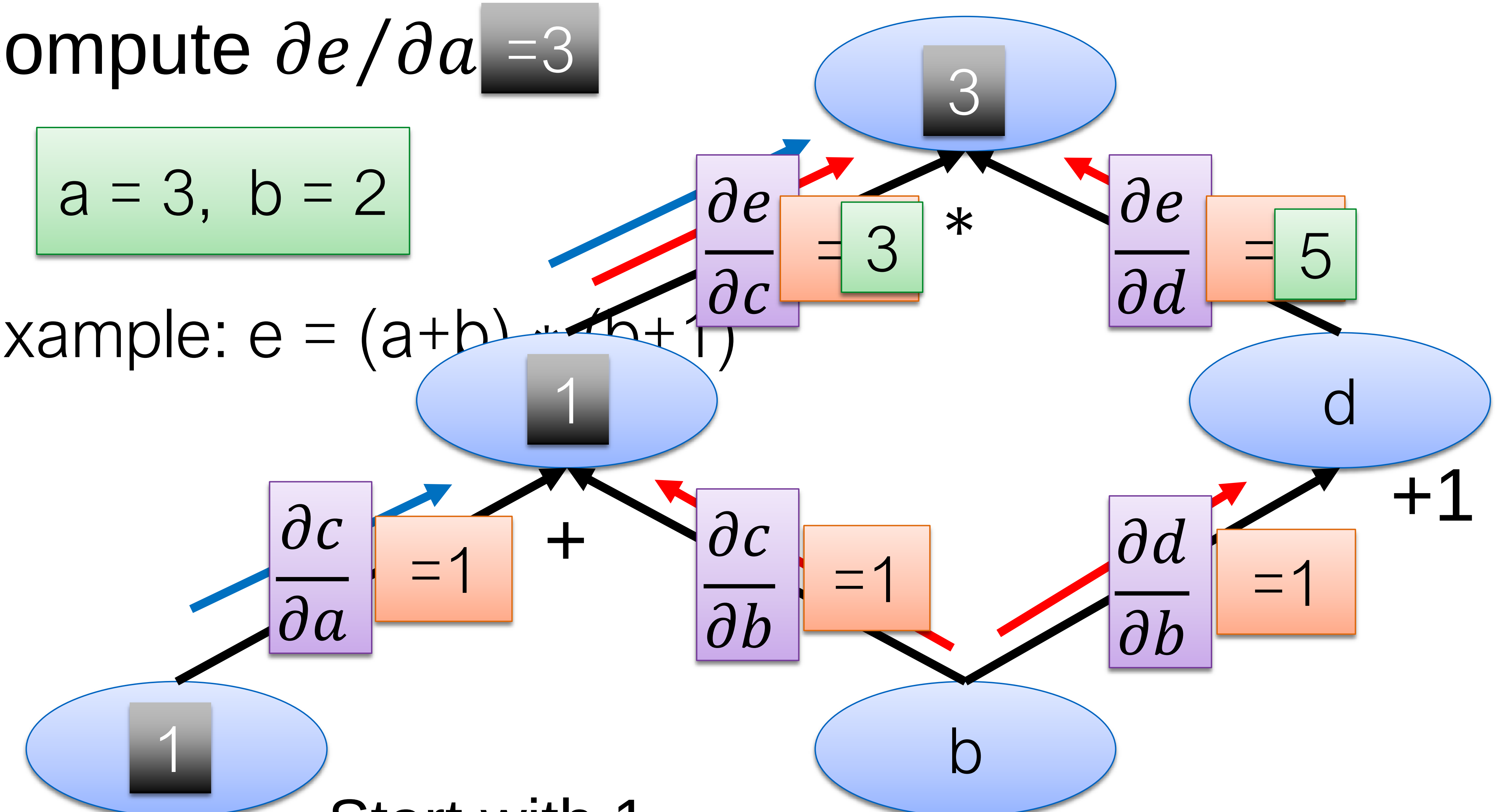


Computational Graph

Compute $\partial e/\partial a$ =3

a = 3, b = 2

- Example: e = (a+b) * (b+1)



Start with 1

Computational Graph

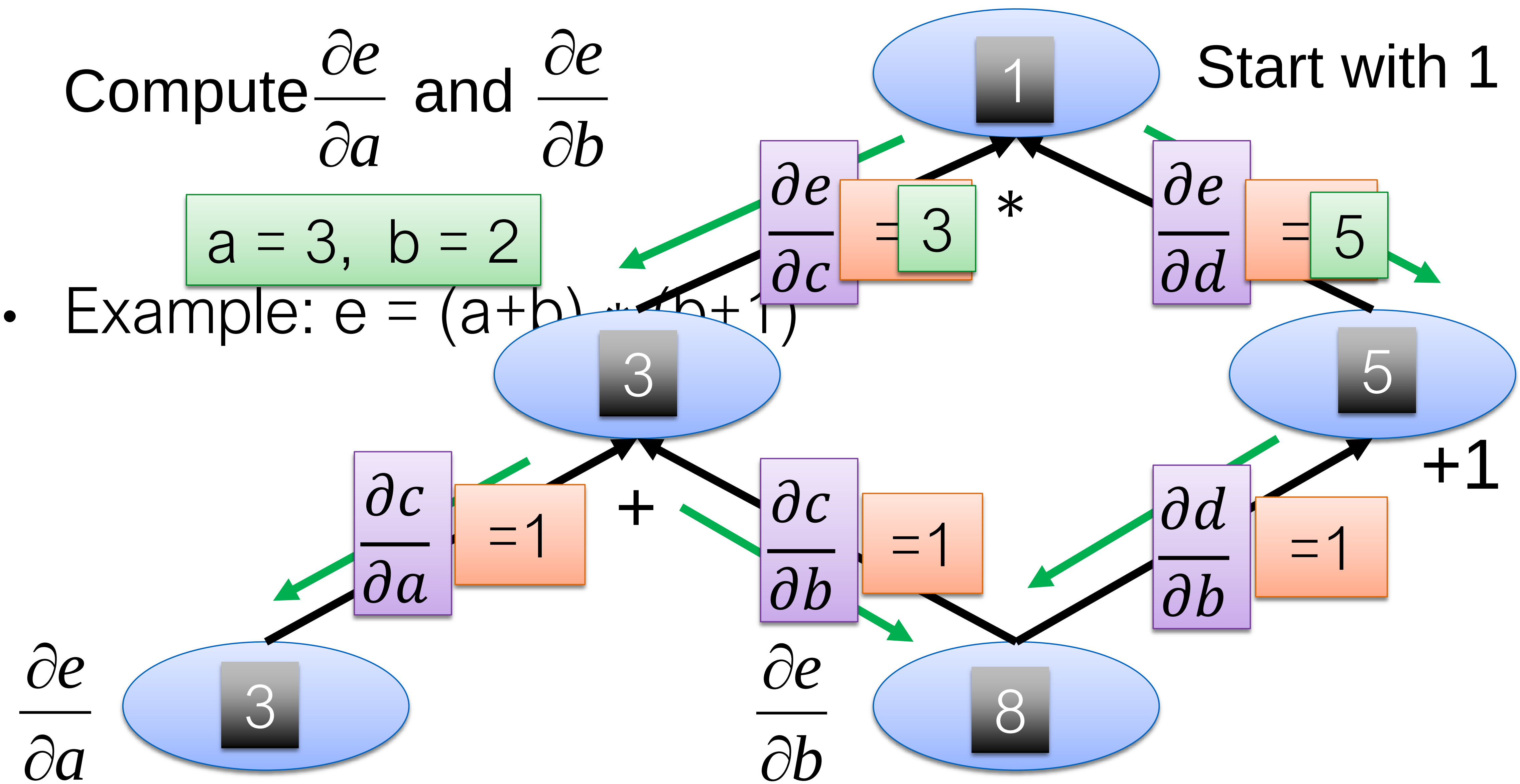
Reverse mode

What is the benefit?

Compute $\frac{\partial e}{\partial a}$ and $\frac{\partial e}{\partial b}$

a = 3, b = 2

- Example: e = (a+b) ∗ (b+1)

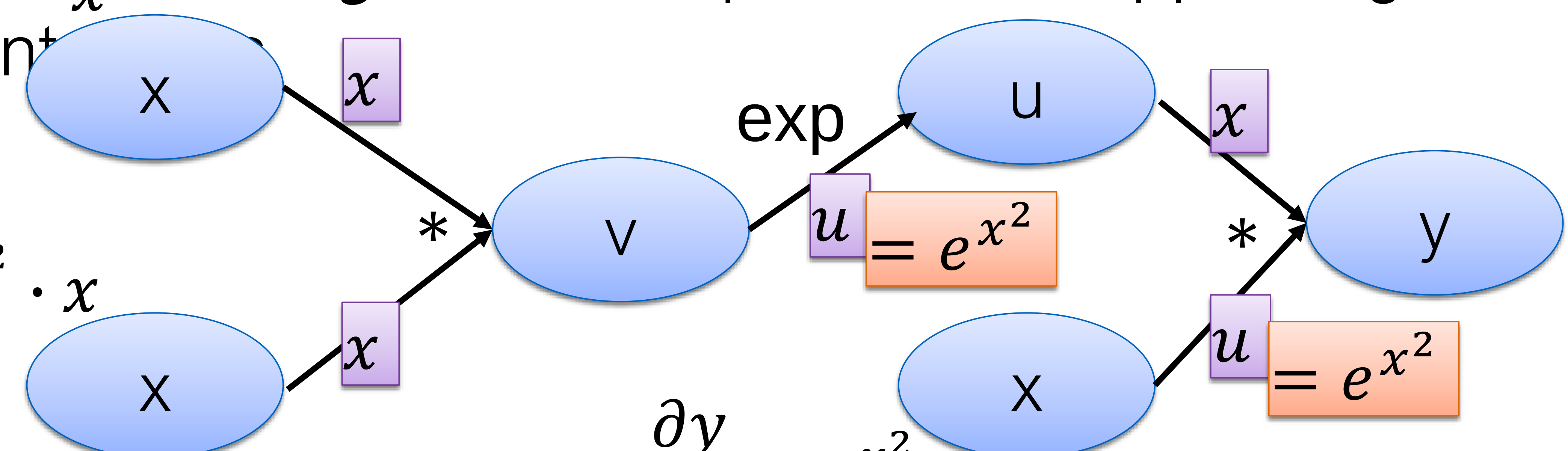


Computational Graph

$$y = xe^{x^2} \qquad \frac{\partial y}{\partial x} = ? \quad e^{x^2} + x \cdot e^{x^2} \cdot 2x$$

$$\frac{\partial y}{\partial x} = x \cdot e^{x^2} \cdot x$$

- *Parameter sharing*: the same parameters appearing in different



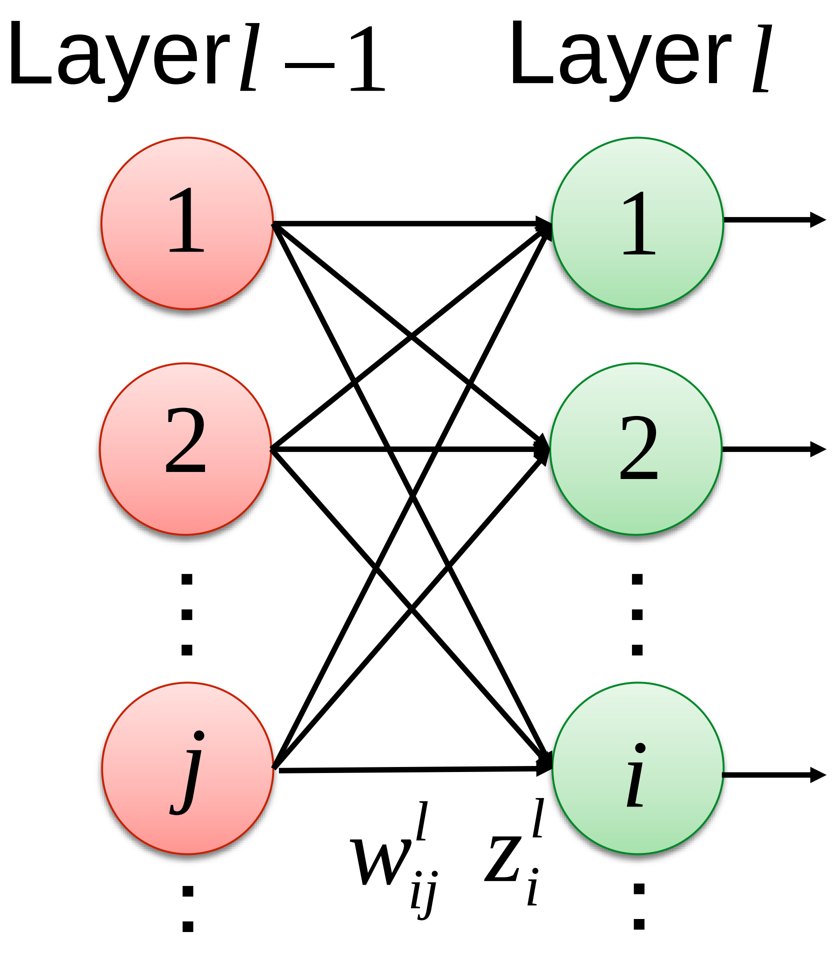
$$\frac{\partial y}{\partial x} = x \cdot e^{x^2} \cdot x$$

$$\frac{\partial y}{\partial x} = e^{x^2}$$

Computational Graph for Feedforward Net

Review: Backpropagation

$$\frac{\partial C}{\partial w_{ij}^l} = \boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}} \boxed{\frac{\partial C}{\partial z_i^l}}$$



$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$

Error signal

$$\delta_i^l$$

Forward Pass

$$z^1 = W^1 x + b^1$$

$$a^1 = \sigma\left(z^1\right)$$

$$\cdots\cdots$$

$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

$$a^{l-1} = \sigma\left(z^{l-1}\right)$$

Backward Pass

$$\delta^L = \sigma'\left(z^L\right) \bullet \nabla_y C$$

$$\delta^{L-1} = \sigma'\left(z^{L-1}\right) \bullet \left(W^L\right)^T \delta^L$$

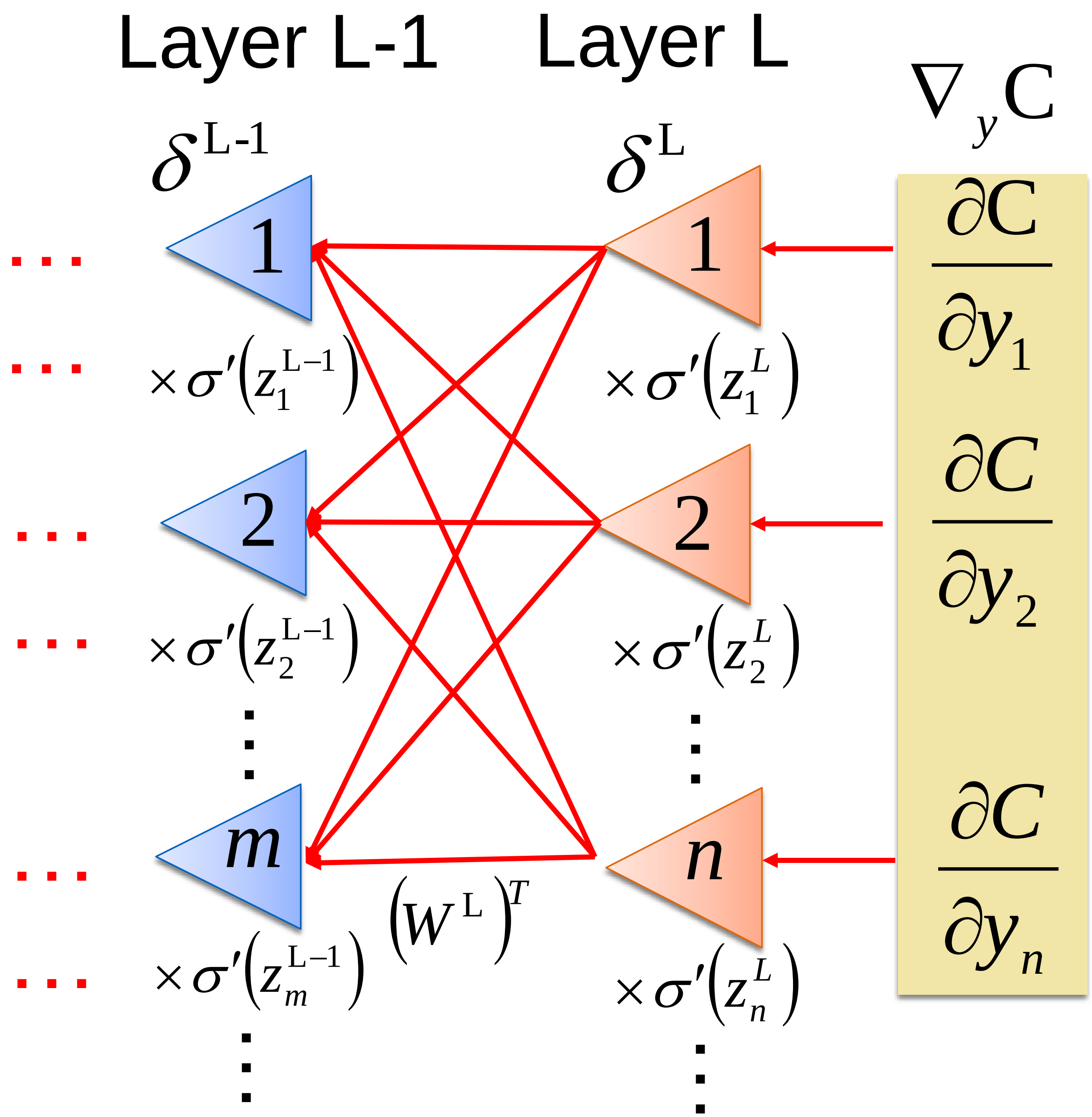
$$\cdots\cdots$$

$$\delta^l = \sigma'\left(z^l\right) \bullet \left(W^{l+1}\right)^T \delta^{l+1}$$

$$\cdots\cdots$$

Review: Backpropagation

$$\frac{\partial C}{\partial w_{ij}^l} = \boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}} \boxed{\frac{\partial C}{\partial z_i^l}}$$



Error signal

δ_i^l

Backward Pass

$$\delta^L = \sigma'(z^L) \bullet \nabla_y C$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

……

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

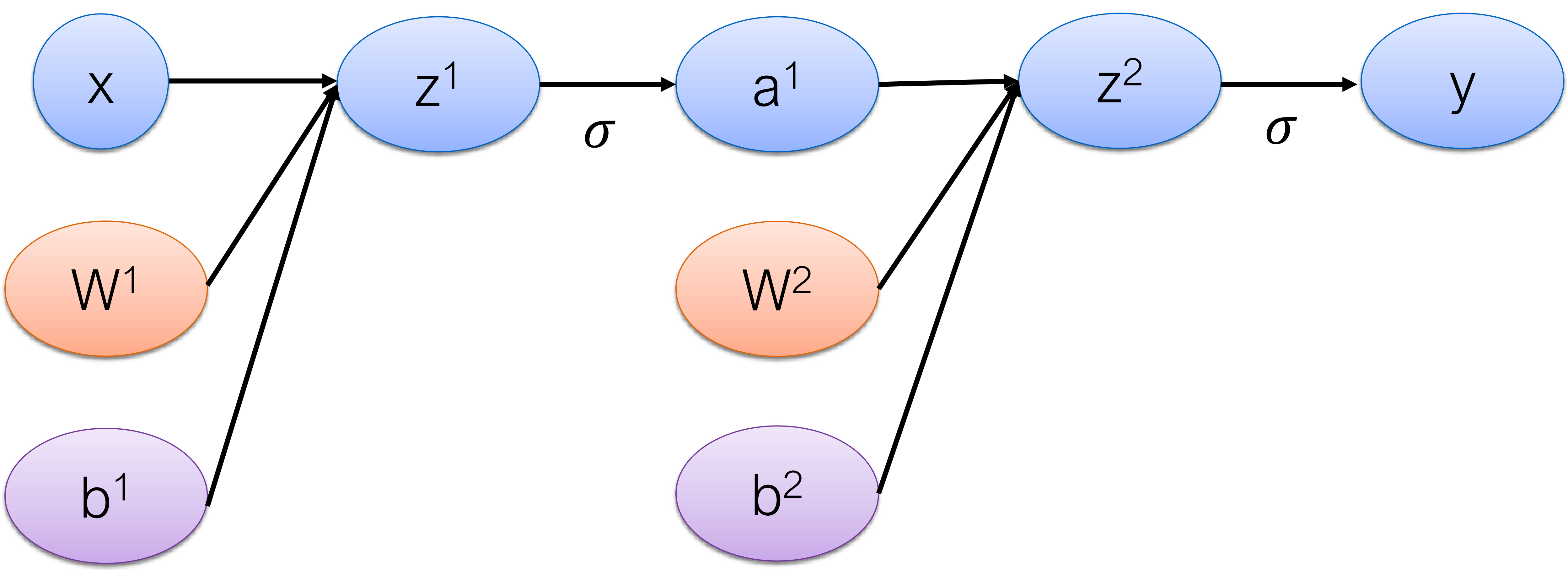
……

(we do not use

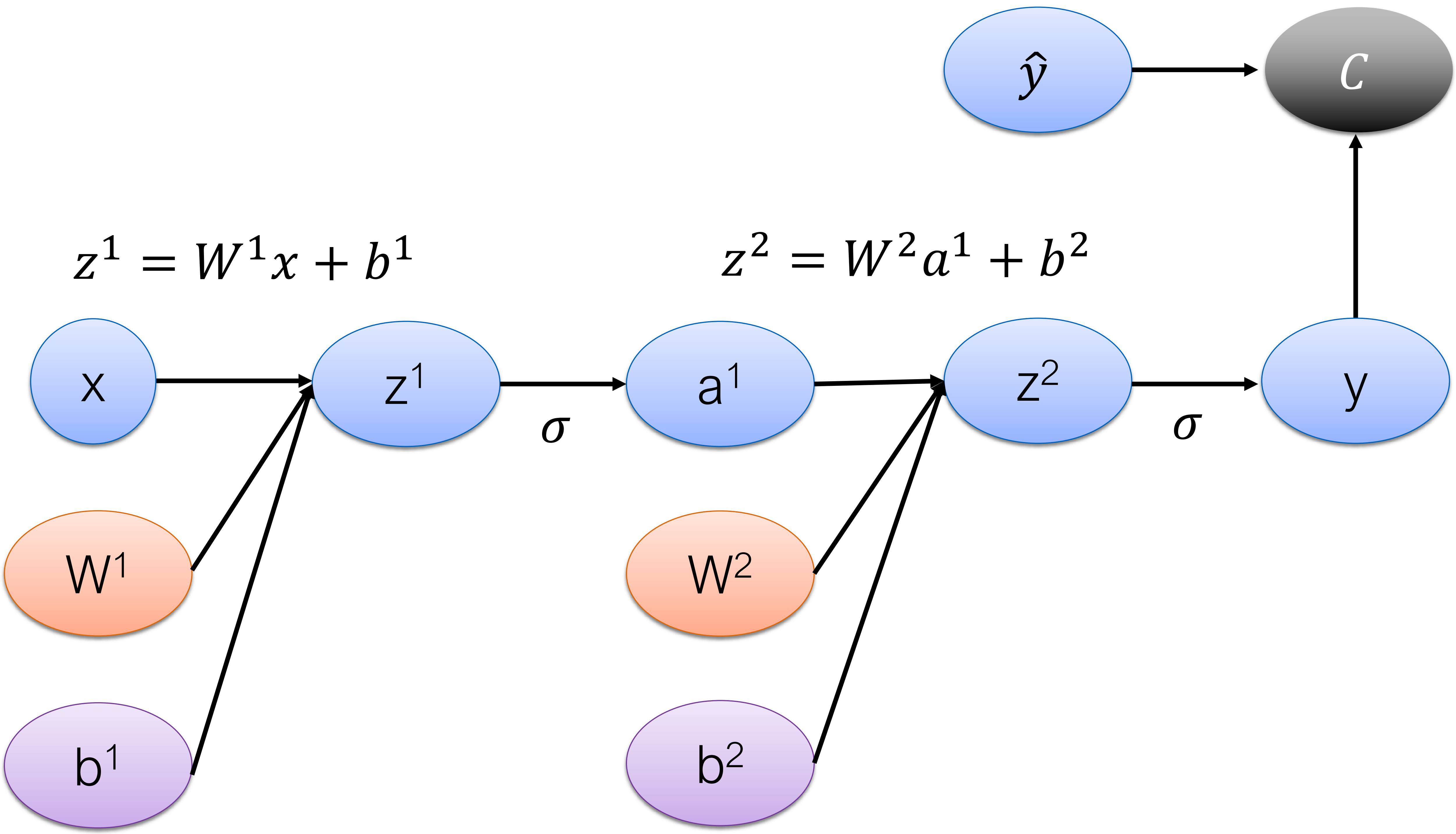
Feedforward Network

$$y = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

$z^1 = W^1 x + b^1$ $\quad\quad$ $z^2 = W^2 a^1 + b^2$



Loss Function of Feedforward Network



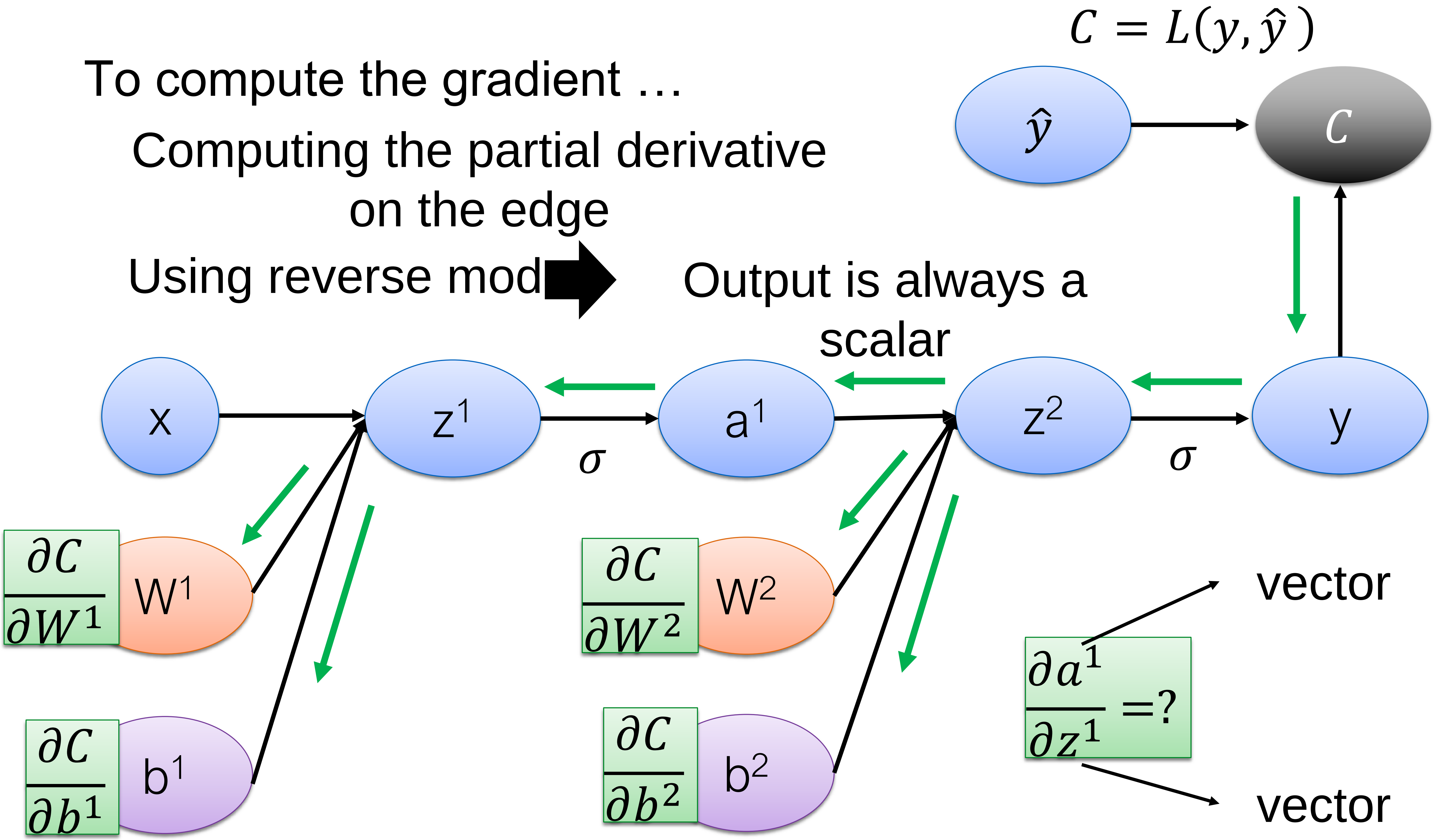
Gradient of Cost Function

To compute the gradient …

Computing the partial derivative on the edge

Using reverse mode

Output is always a scalar



Jacobian Matrix

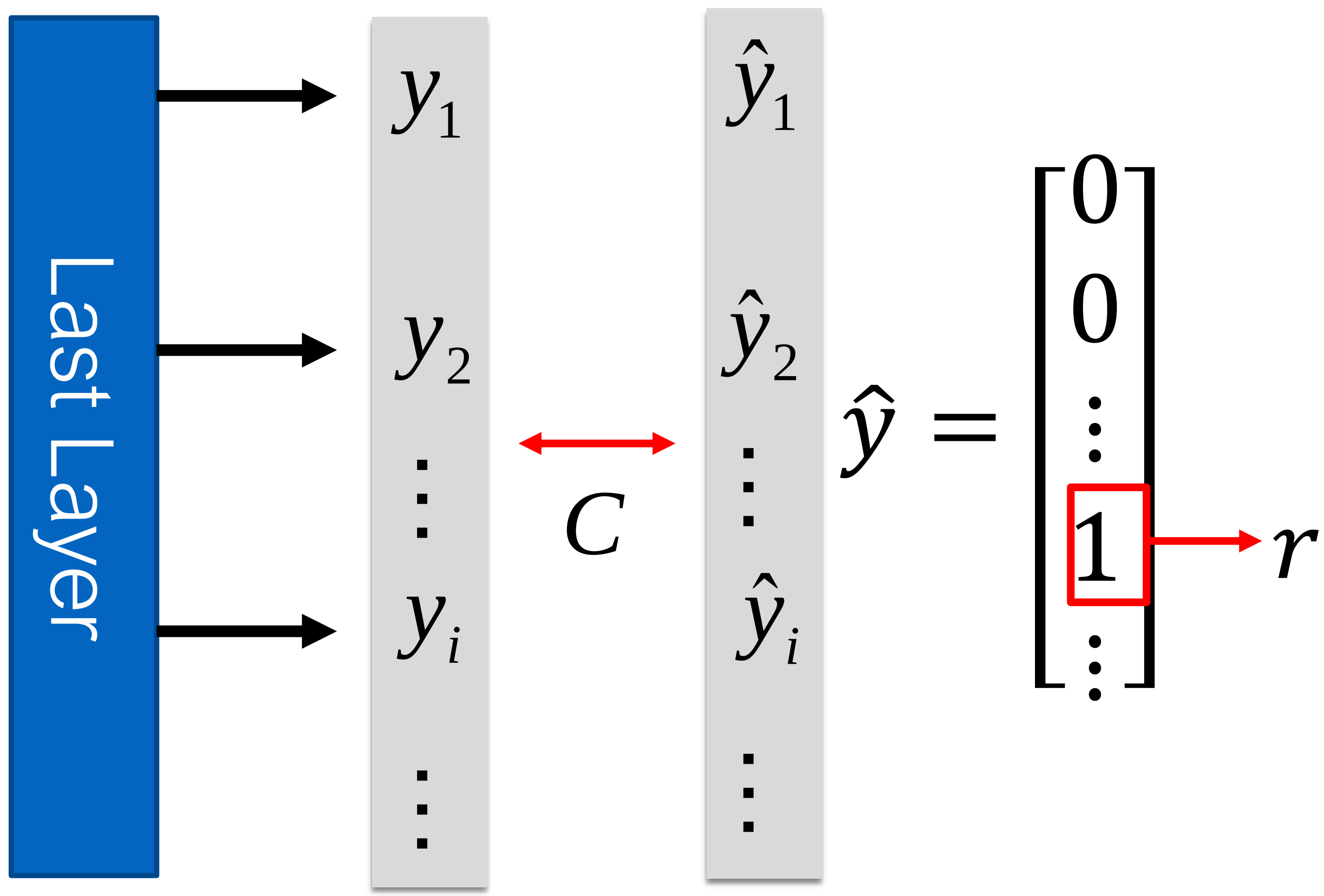
$$y = f(x) \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\frac{\partial y}{\partial x} = \underbrace{\left[\qquad\qquad\qquad\qquad\qquad\qquad \right]}_{\text{size of x}} \left.\vphantom{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}\right\} \text{size of y}$$

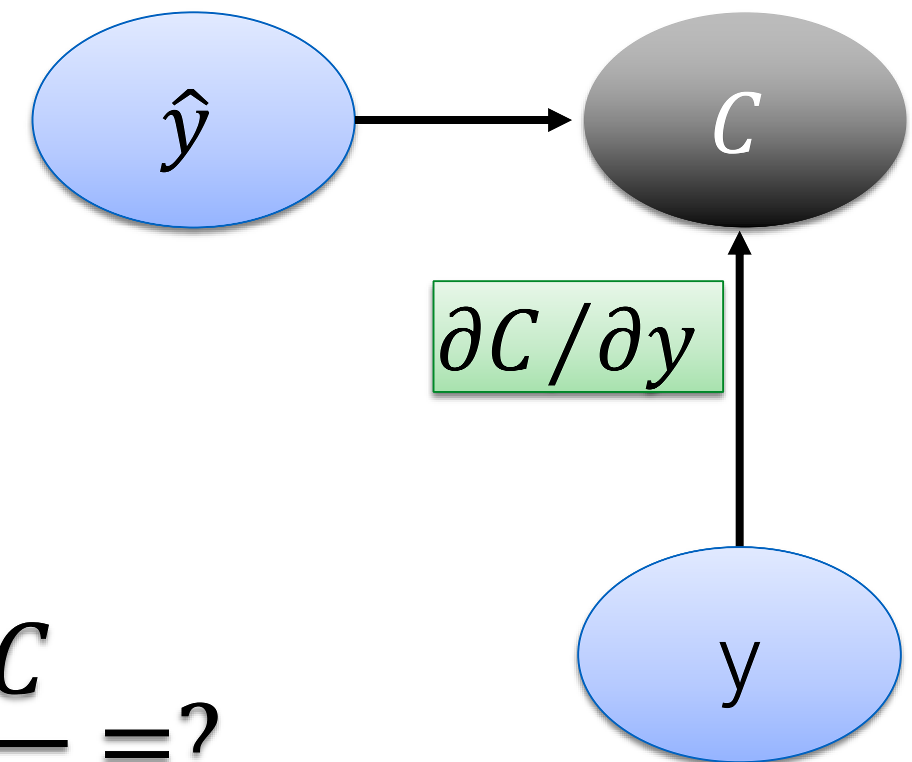
Example

$$\begin{bmatrix} x_1 + x_2 x_3 \\ 2x_3 \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) \qquad \frac{\partial y}{\partial x} = \left[\qquad\qquad\qquad\qquad \right]$$

Gradient of Cost Function



$C = L(y, \hat{y})$



$$\frac{\partial C}{\partial y_i} = ?$$

Cross Entropy: $C = -log\, y_r$

$$\frac{\partial C}{\partial y} = [\quad]$$

$i = r$:

$$\partial C / \partial y_r = -1/y_r$$

$i \neq r$: $\partial C / \partial y_i = 0$

Gradient of Cost Function

$\frac{\partial y}{\partial z^2}$ is a Jacobian mat square y

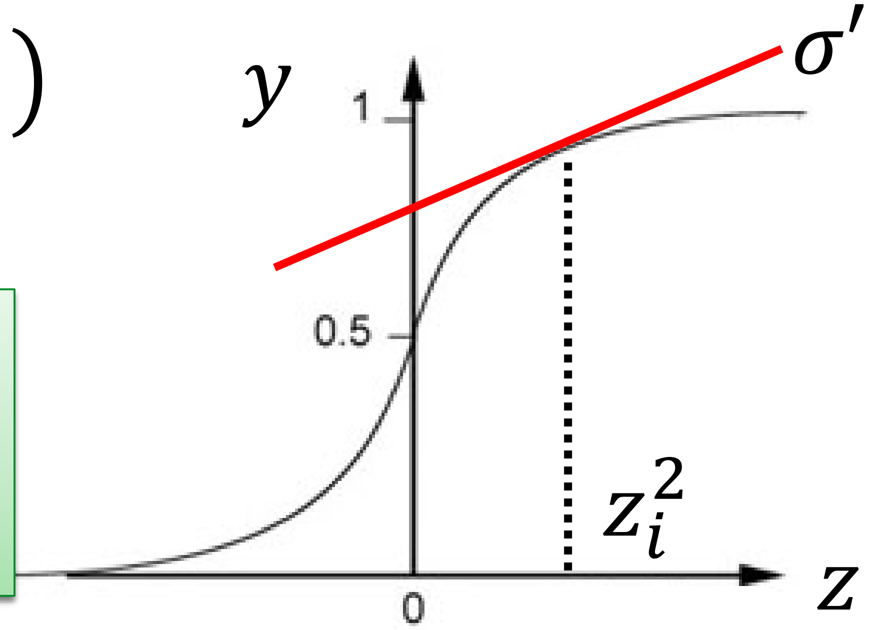
z^2

i-th row, j-th column: $\partial y_i / \partial z_j^2$

$i \neq j$: $\partial y_i / \partial z_j^2 = 0$

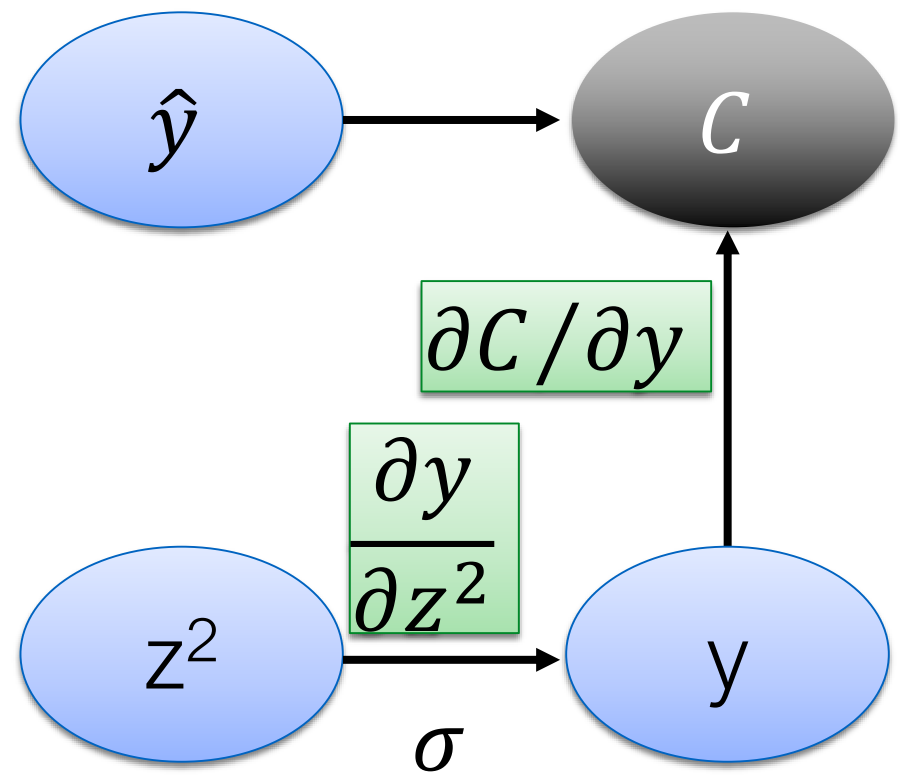
$i = j$: $\partial y_i / \partial z_i^2 = \sigma'(z_i^2)$

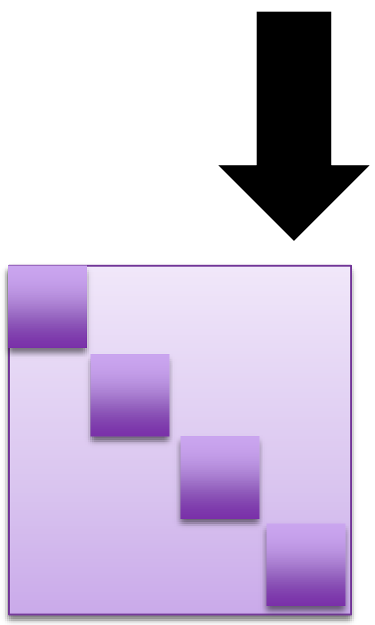
$y_i = \sigma(z_i^2)$



How about softmax? ☺

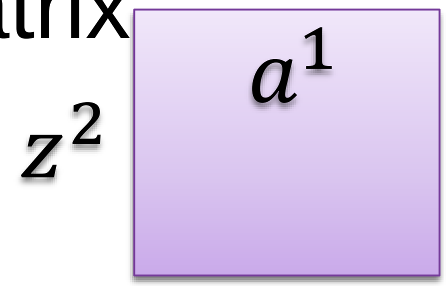
$C = L(y, \hat{y})$





Diagonal Matrix

$\dfrac{\partial z^2}{\partial a^1}$ is a Jacobian matrix

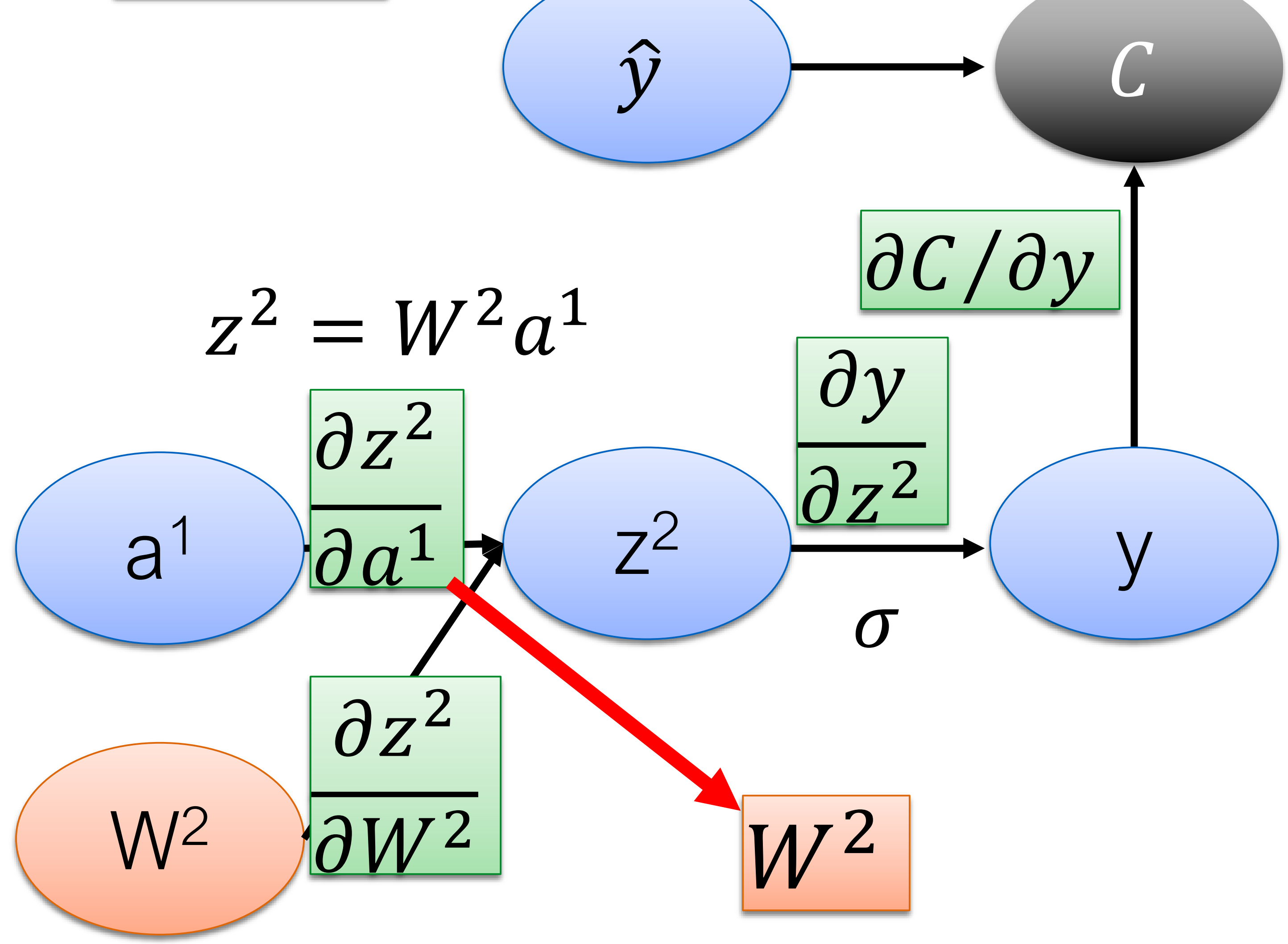


i-th row, j-th column:

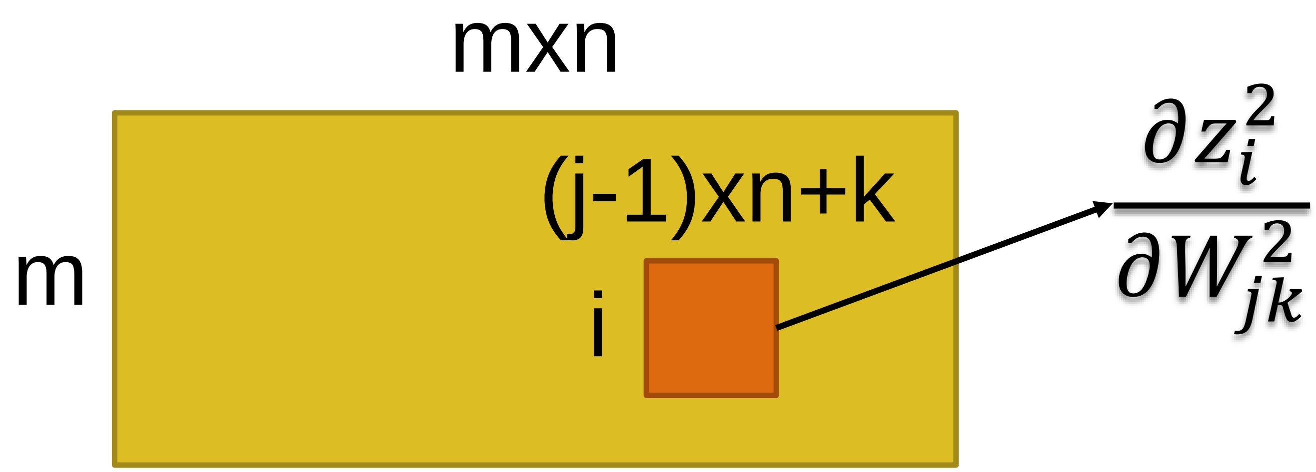
$$\frac{\partial z_i^2}{\partial a_j^1} =$$

$$z_i^2 = w_{i1}^2 a_1^1 + w_{i2}^2 a_2^1 + \cdots + w_{in}^2 a_n^1$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$



$$\frac{\partial z^2}{\partial W^2} =$$

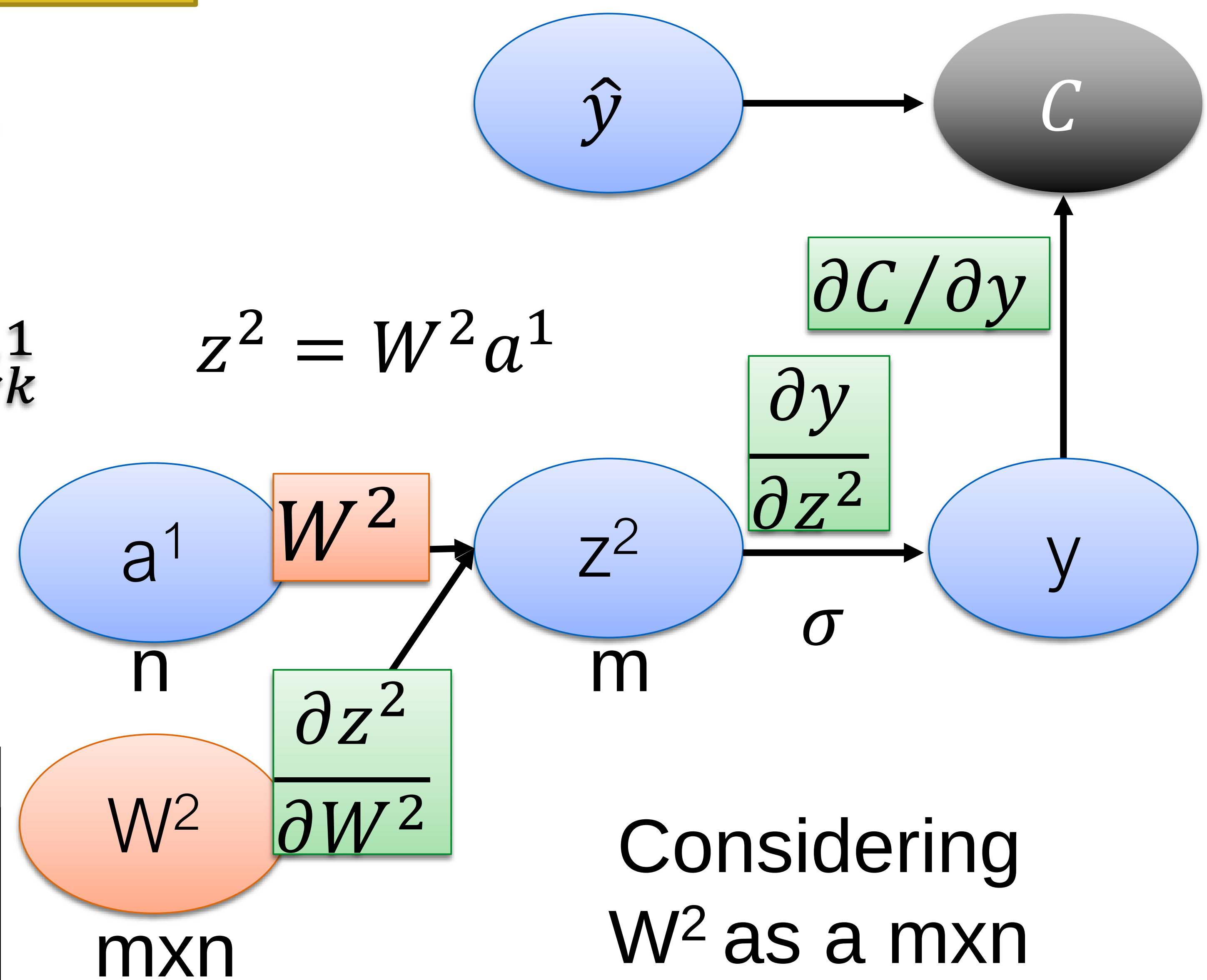


$$C = L(y, \hat{y})$$

$$\frac{\partial z_i^2}{\partial W_{jk}^2} = ? \qquad i \neq j: \frac{\partial z_i^2}{\partial W_{jk}^2} = 0$$

$$i = j: \frac{\partial z_i^2}{\partial W_{ik}^2} = a_k^1$$

$$z^2 = W^2 a^1$$



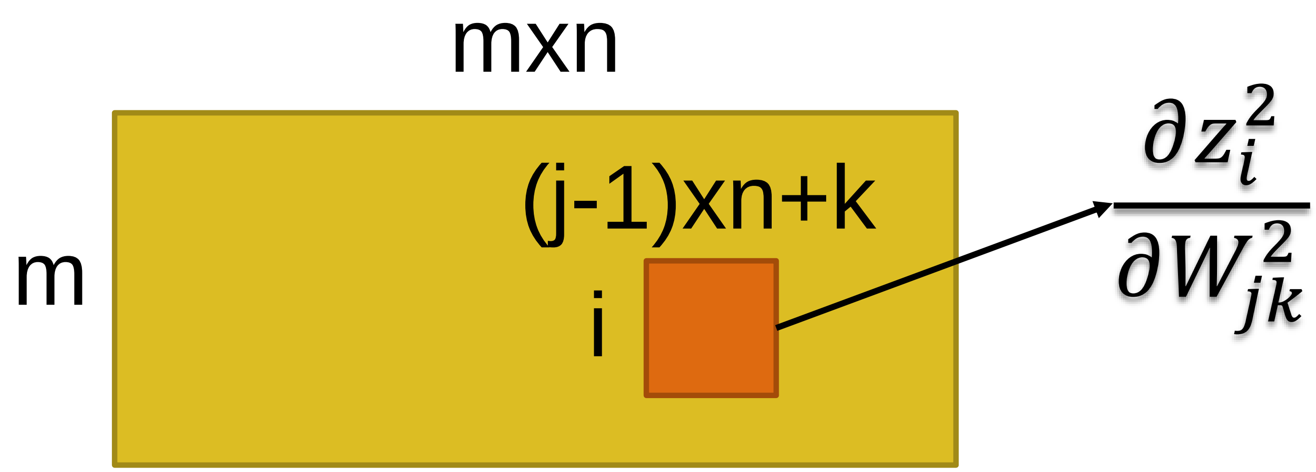
$$z_i^2 = w_{i1}^2 a_1^1 + w_{i2}^2 a_2^1 + \cdots + w_{in}^2 a_n^1$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

Considering W^2 as a mxn **vector**

(considering $\partial z^2 / \partial W^2$ as a tensor makes thing easier)

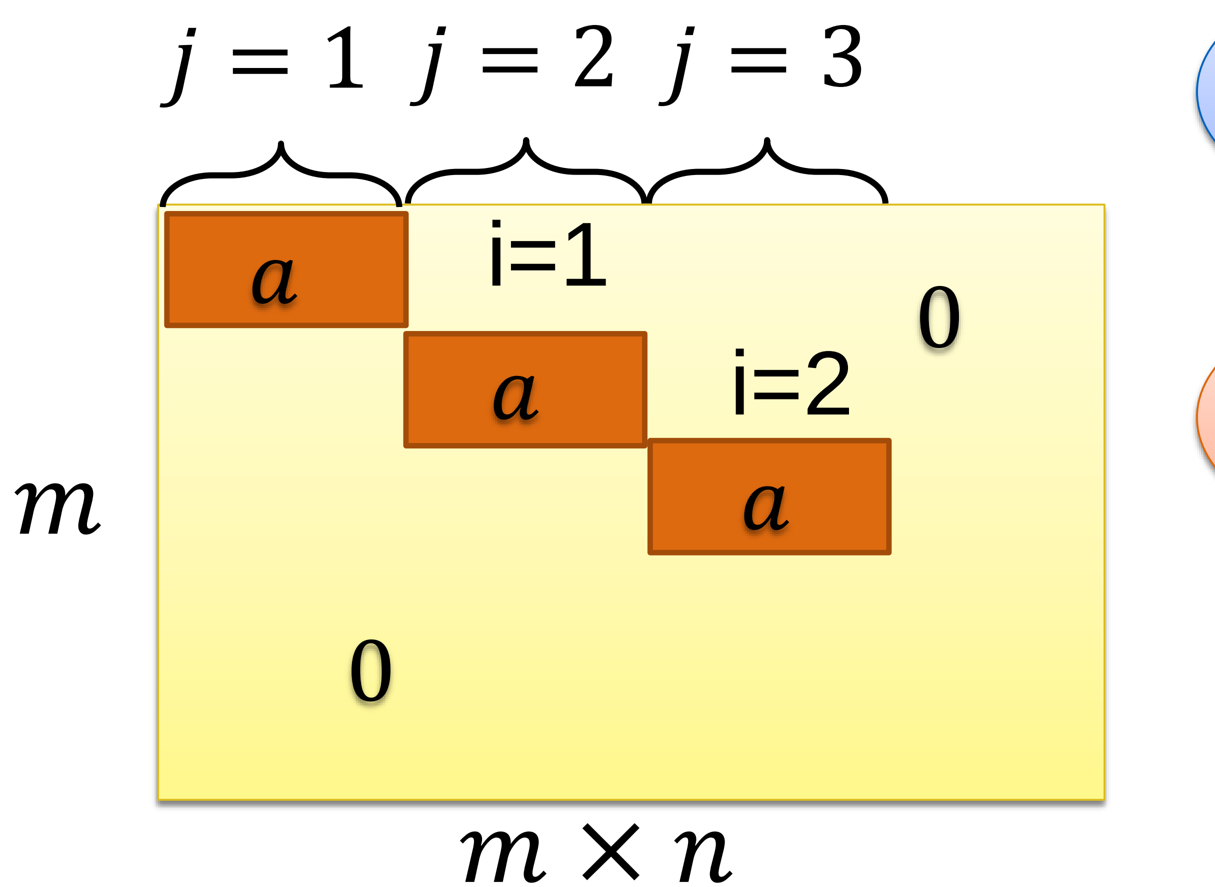
$$\frac{\partial z^2}{\partial W^2} =$$



$$\frac{\partial z_i^2}{\partial W_{jk}^2} = ?$$

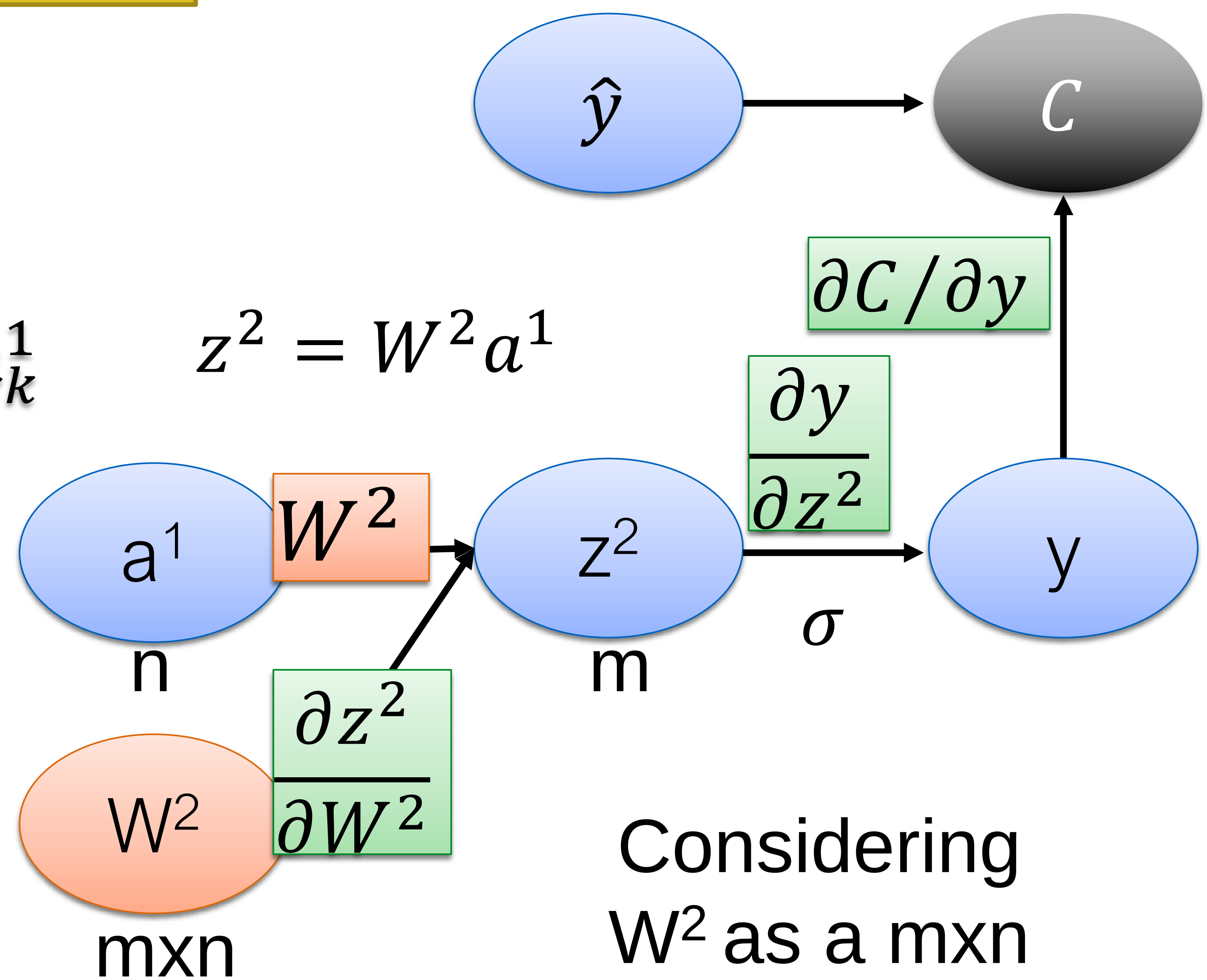
$$i \neq j: \frac{\partial z_i^2}{\partial W_{jk}^2} = 0$$

$$i = j: \frac{\partial z_i^2}{\partial W_{ik}^2} = a_k^1$$



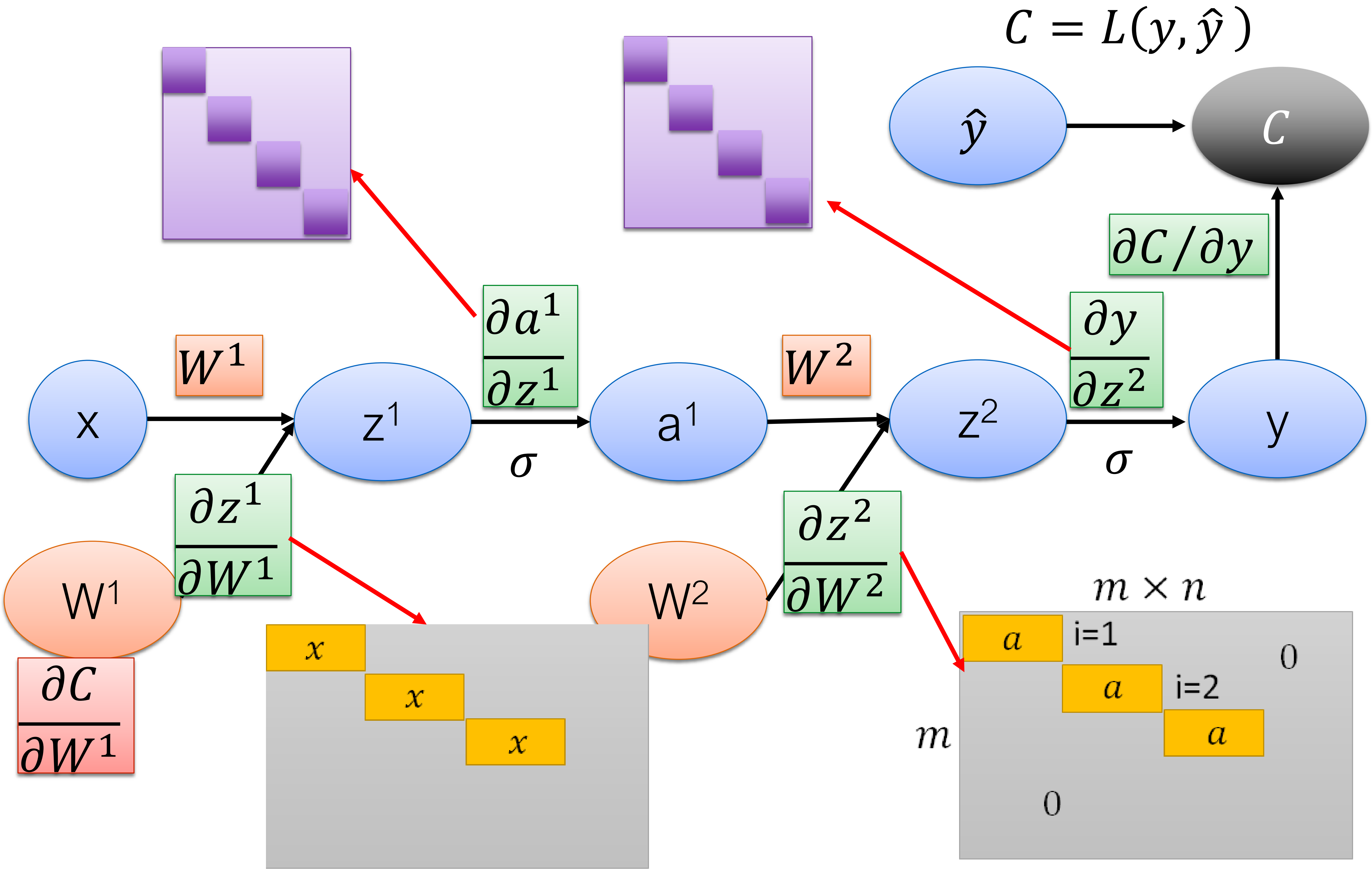
$$C = L(y, \hat{y})$$

$$z^2 = W^2 a^1$$



Considering W^2 as a mxn **vector**

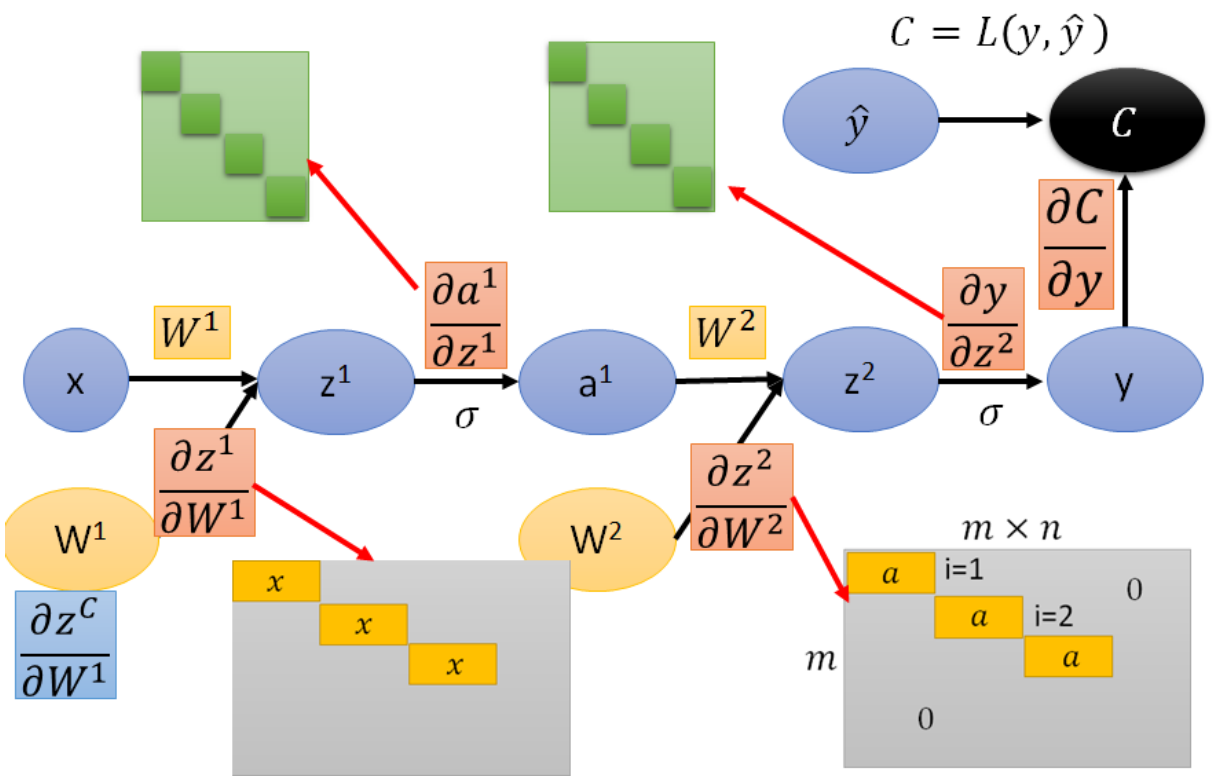
$$\frac{\partial C}{\partial W^1} = \frac{\partial C}{\partial y}\,\frac{\partial y}{\partial z^2}\,W^2\,\frac{\partial a^1}{\partial z^1}\,\frac{\partial z^1}{\partial W^1} = \left[\cdots \quad \frac{\partial C}{\partial W^1_{ij}} \quad \cdots\right]$$



Que

$$\frac{\partial C}{\partial w_{ij}^{l}} = \boxed{\frac{\partial z_i^{l}}{\partial w_{ij}^{l}}} \boxed{\frac{\partial C}{\partial z_i^{l}}}$$

Error signal



Forward Pass

$$z^1 = W^1 x + b^1$$

$$a^1 = \sigma(z^1)$$

$$\cdots\cdots$$

$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

$$a^{l-1} = \sigma(z^{l-1})$$

Backward Pass

$$\delta^{L} = \sigma'(z^{L}) \bullet \nabla_y C$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^{L})^{T} \delta^{L}$$

$$\cdots\cdots$$

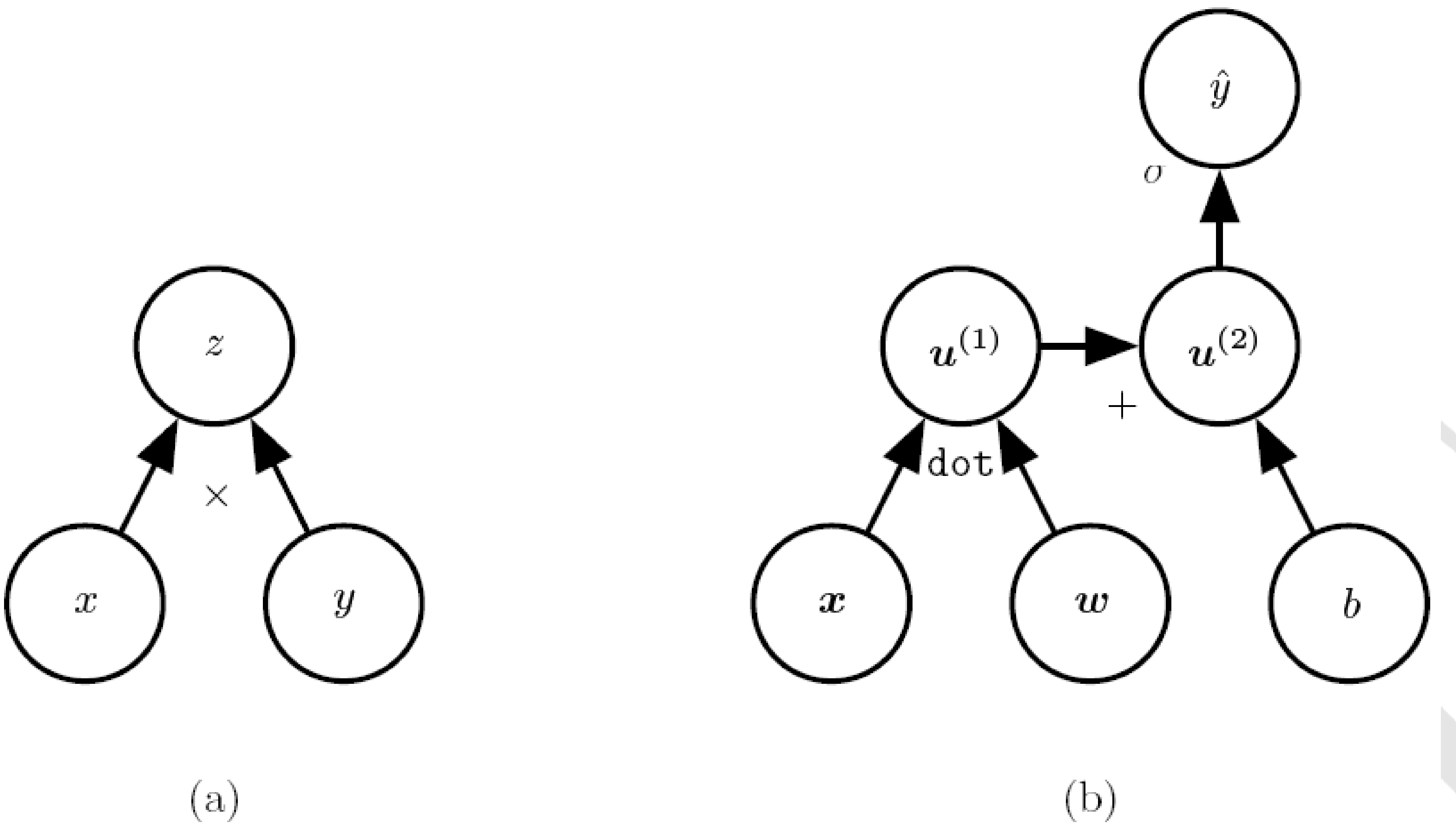
$$\delta^{l} = \sigma'(z^{l}) \bullet (W^{l+1})^{T} \delta^{l+1}$$

$$\cdots\cdots$$

Q: Only backward pass for computational graph?

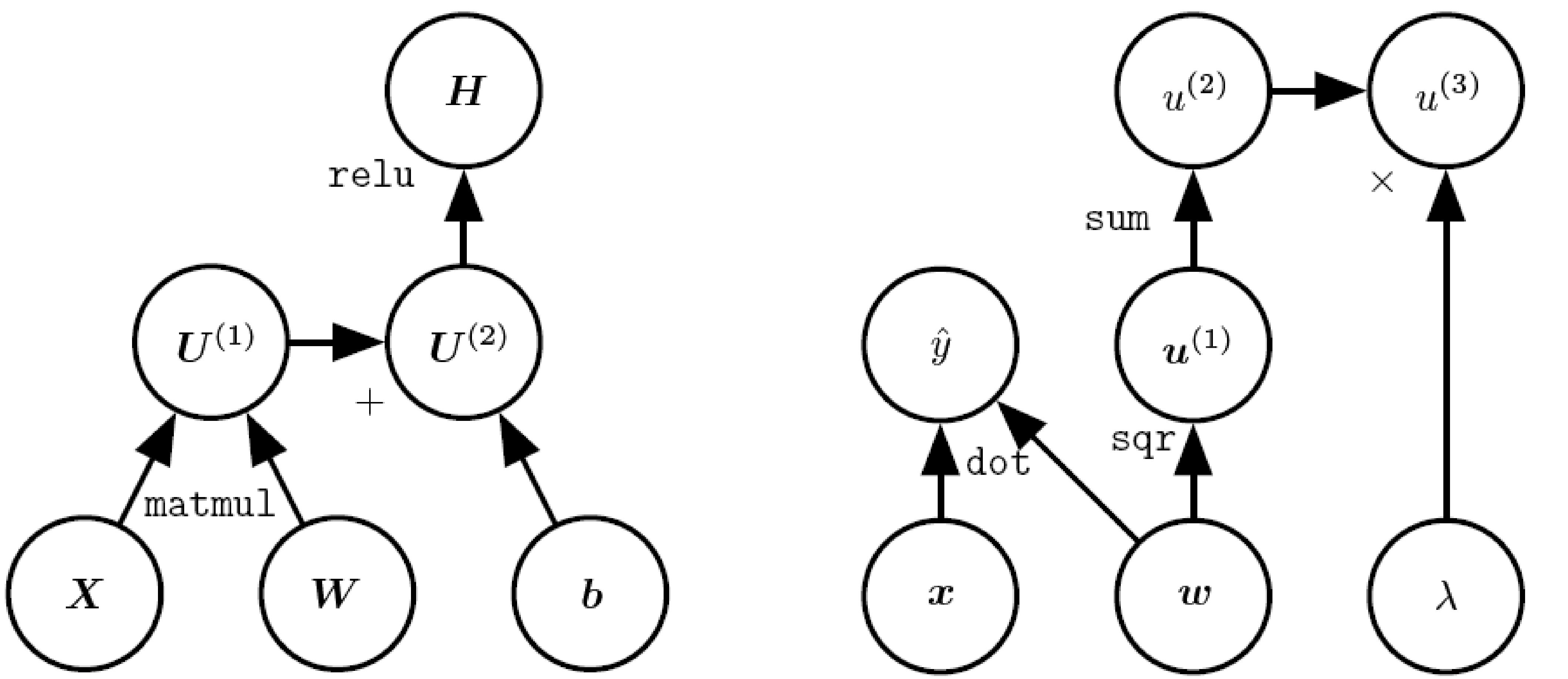
Q: Do we get the same results from the two different approaches?

计算图：节点表示变量，变量之间的计算称为操作。



（a）表示的z=xy的计算图；（b）逻辑回归的计算图

计算图：节点表示变量，变量之间的计算称为操作。



(c) (d)

（c） $\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\}$

（d）带权值衰减的线性回归（回归y和衰减u（3））

梯度计算：

设 $f(g(x)) = f(y)$。那么链式法则是说

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

假设 $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$，$g$ 是从 $\mathbb{R}^m$ 到 $\mathbb{R}^n$ 的映射，f 是从 $\mathbb{R}^n$ 到 $\mathbb{R}$ 的映射。如果 $\boldsymbol{y} = g(\boldsymbol{x})$ 并且 $z = f(\boldsymbol{y})$，那么

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

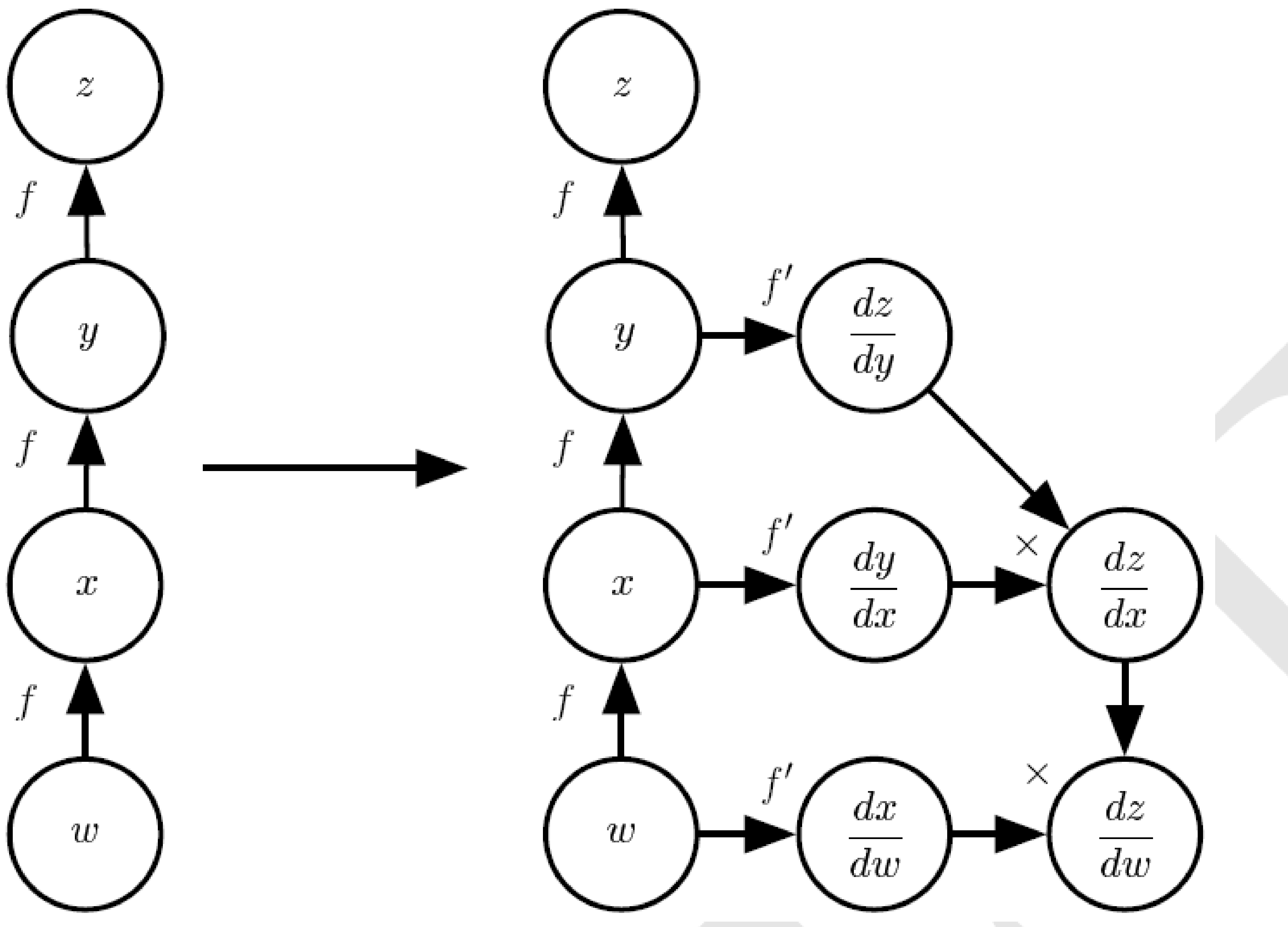
梯度计算：

用向量方式表示：

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z,$$

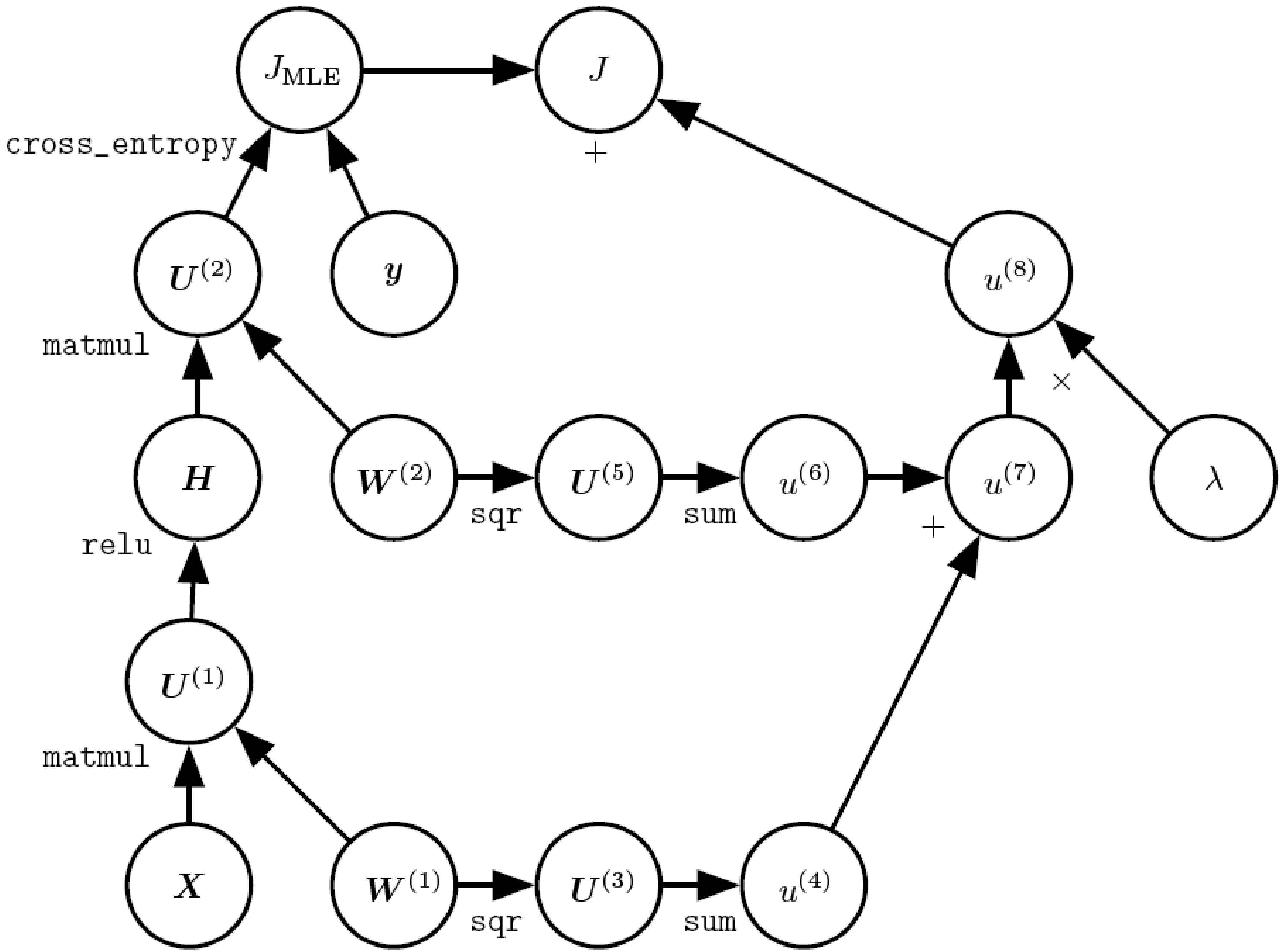
这里 $\frac{\partial y}{\partial x}$ 是 g 的 $n \times m$ 的 Jacobian 矩阵。

BP算法计算图例子：



计算图：例子（两层神经网络）

$$J = J_{\text{MLE}} + \lambda \left(\sum \left(W_{i,j}^{(1)} \right)^2 + \sum \left(W_{i,j}^{(2)} \right)^2 \right)$$



深度前馈神经网络设计：

1. 目标函数：最大似然准则

2. 输出单元设计：根据任务进行设计

3. 隐藏层单元设计：ReLu及其它

深度前馈神经网络设计：

1. 目标函数：最大似然准则

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

其具体的形式依赖于P_{model}（y|x）的形式，如假设

$$p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{I})$$

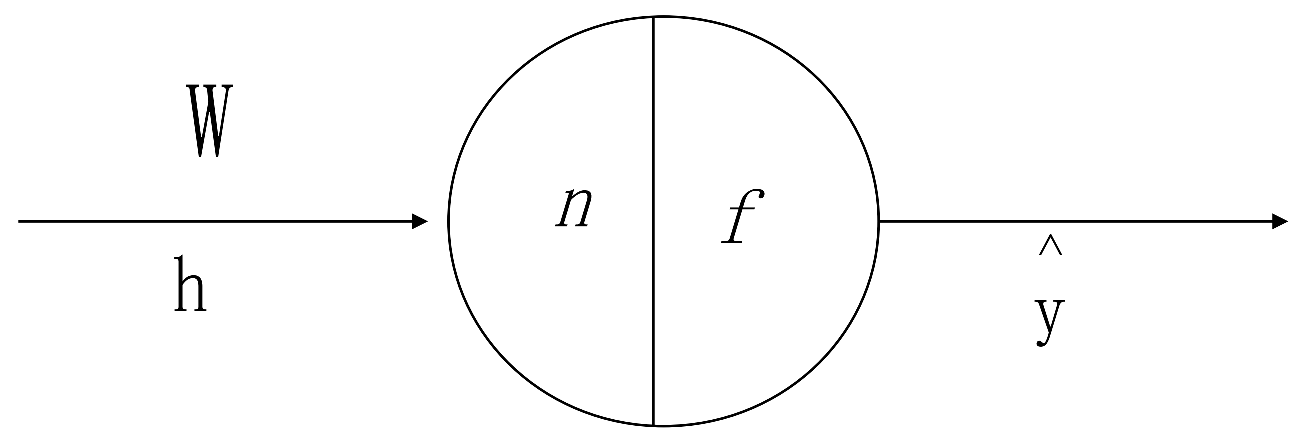
则目标函数为：

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x}; \boldsymbol{\theta})||^2 + \text{const}$$

深度前馈神经网络设计：

2. 输出单元设计：

输出单元设计与任务、代价函数相关。



（1）线性输出单元

$$\hat{y} = \boldsymbol{W}^{\top}\boldsymbol{h} + \boldsymbol{b}$$

应用于高斯分布均值：

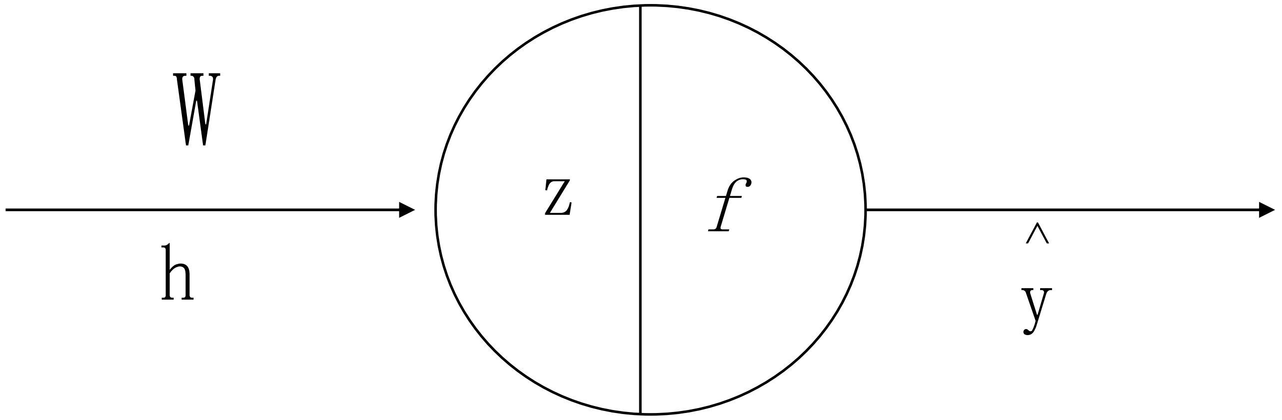
$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

深度前馈神经网络设计：

2. 输出单元设计：

（2）sigmoid输出单元

$$\hat{y} = \sigma\left(\boldsymbol{w}^{\top}\boldsymbol{h} + b\right)$$



应用于二分类问题的贝努力分布：

$$P(y) = \sigma((2y - 1)z)$$

目标函数：

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\log P(y \mid \boldsymbol{x}) \\ &= -\log \sigma((2y - 1)z) \\ &= \zeta((1 - 2y)z). \end{aligned}$$

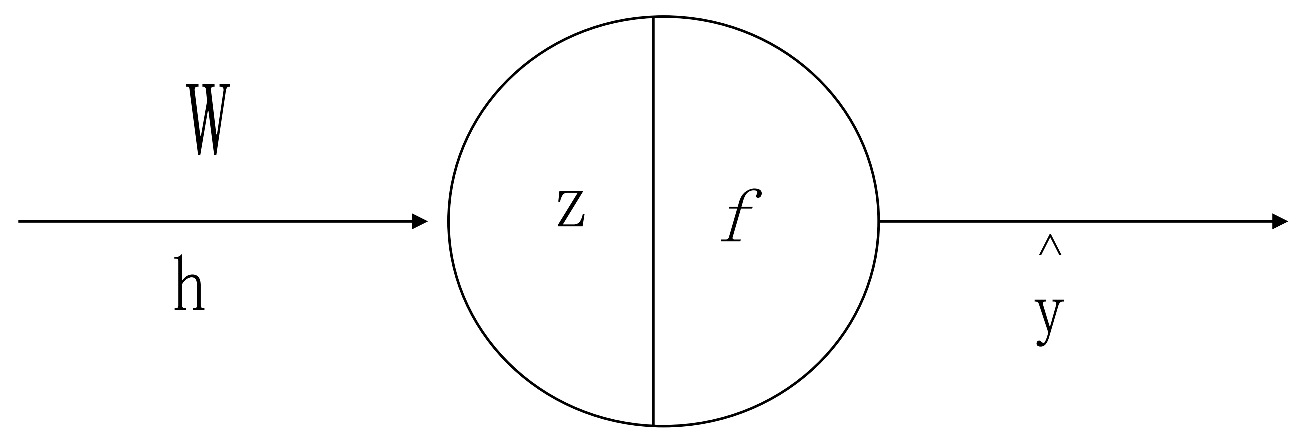
深度前馈神经网络设计：

2.输出单元设计：

（3）softmax输出单元

输入输出都是一个向量

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b},$$



Softmax函数：

$$\operatorname{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

表示：

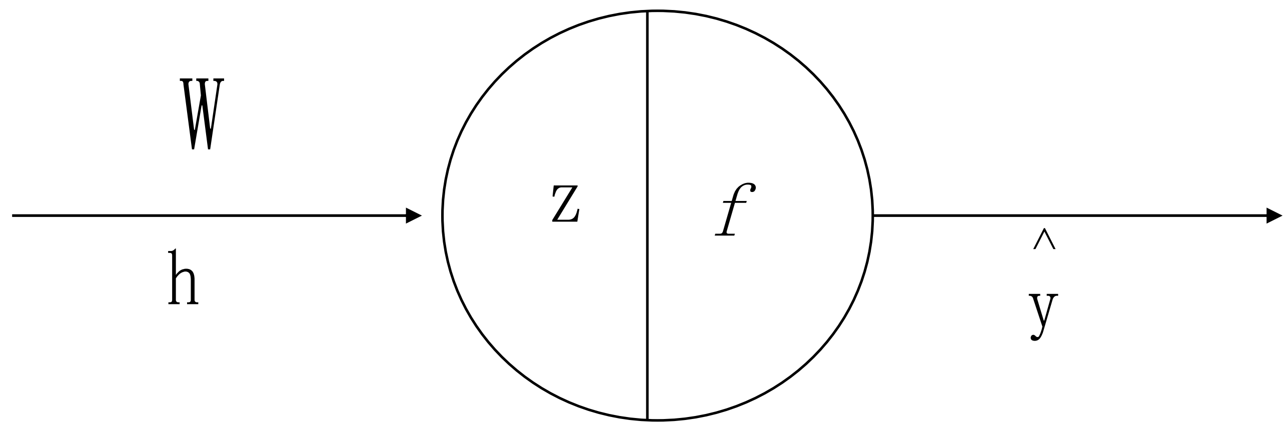
$$P(y=i \mid x) = \operatorname{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

深度前馈神经网络设计：

3. 隐藏单元设计：

（1）ReLU激活函数

$$z = \boldsymbol{W}^{\top}\boldsymbol{h} + \boldsymbol{b},$$



则激活函数为：

f（x）= max{0，z}

特点：神经元被激活时导数为1，所以不会梯度消失；导数为常数，计算稳定。

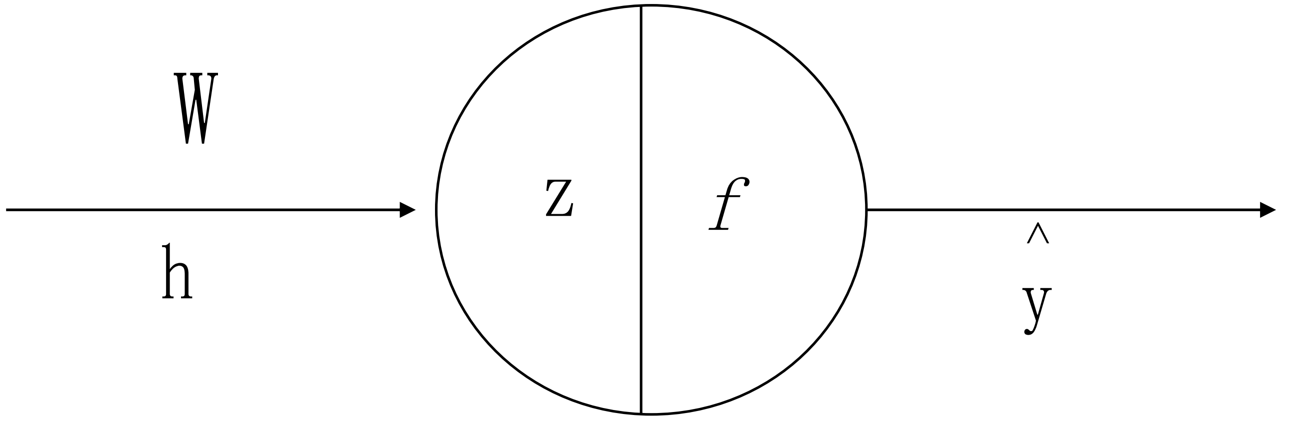
深度前馈神经网络设计：

3.隐藏单元设计：

$\tanh(z) = 2\sigma(2z) - 1$

（2）Sigmoid和双曲面正切函数

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



$$\tanh(z) = 2\sigma(2z) - 1$$

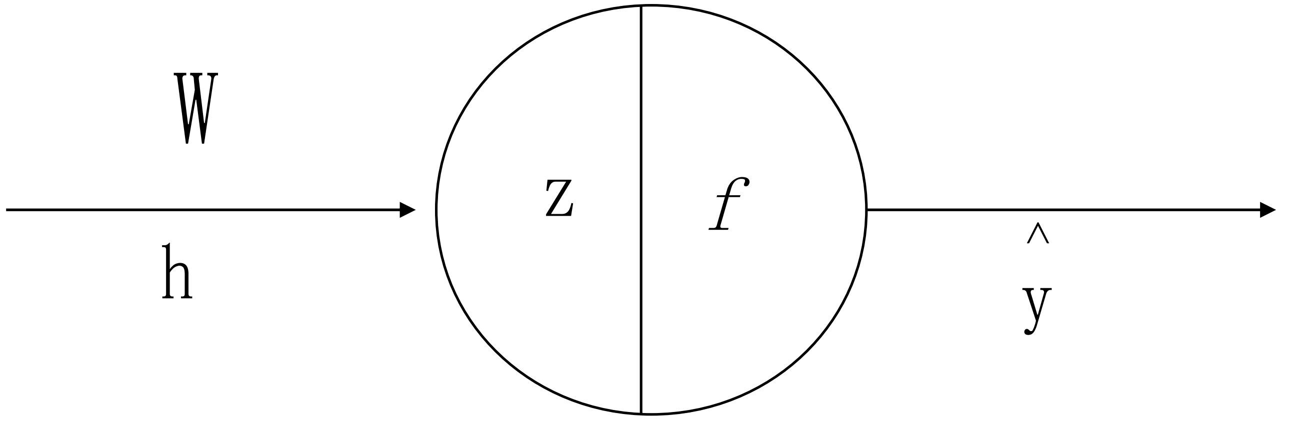
（3）其它激活函数

深度前馈神经网络设计：

3.隐藏单元设计：

（2）Sigmoid和双曲面正切函数

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



$$\tanh(z) = 2\sigma(2z) - 1$$

（3）其它激活函数（略）

深度前馈神经网络设计：

3. 结构设计：

考虑：使用多少神经元以及这些神经元如何连接。

方法：通过实验的方法验证和探测合理的神经网络结构。

正则化策略：

1. 参数范数惩罚：

$$\tilde{J}(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

(1)L2范数：

参数更新：

$$\boldsymbol{w} \leftarrow (1-\epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}).$$

（2）L1范数：

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}),$$

正则化策略：

2. 数据集增强：

训练数据集不足：创建假数据集并添加到训练集中。

3. 增加噪声

（1）输入数据增加噪声；

（2）参数增加噪声；

（3）输出数据增加噪声。

正则化策略：

4. 半监督学习

P（x）产生的无标记样本和P（x，y）中的标记样本用于估计条件概率

P（y|x）

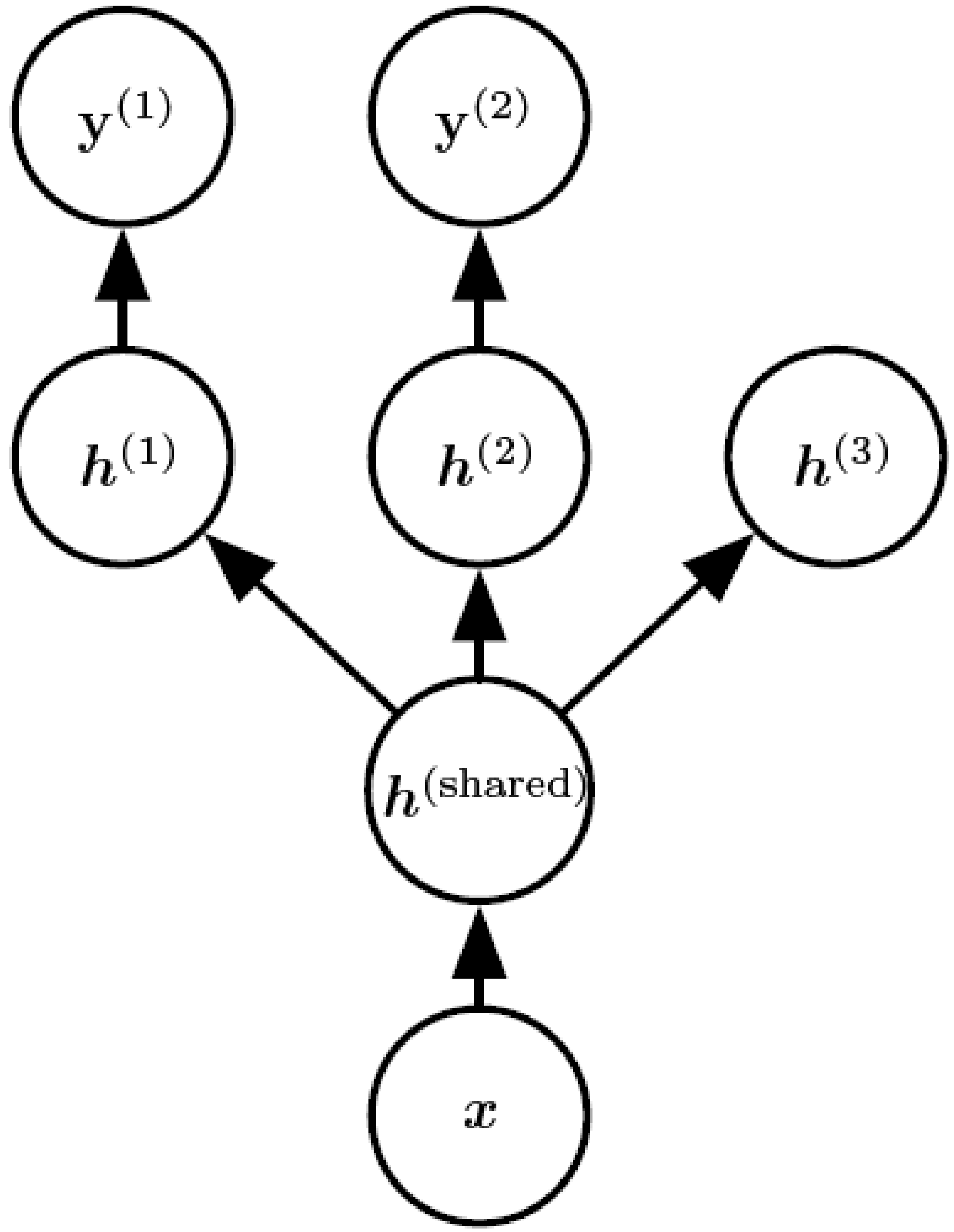
5. 多任务学习

在多个任务之间共享表示。

正则化策略：

5. 多任务学习

在多个任务之间共享表示。



正则化策略：

6. 提前终止

将数据集分为训练集和测试集，测试集用来探测最优的模型。当探测到最优模型，则训练终止。

提前终止是一种正则化方法，其效果好于范数惩罚。

正则化策略：

8. 参数绑定与参数共享

设置两个模型：

$$\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x}) \text{ 和 } \hat{y}^{(B)} = f(\boldsymbol{w}^{(B)}, \boldsymbol{x})$$

设置模型惩罚：

$$\left\| \boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)} \right\|_2^2$$

参数共享：要求模型A中参数与模型B中参数相等

正则化策略：

9. 稀疏表示

隐藏层被激活的神经元数据较少。

10. 集成方法

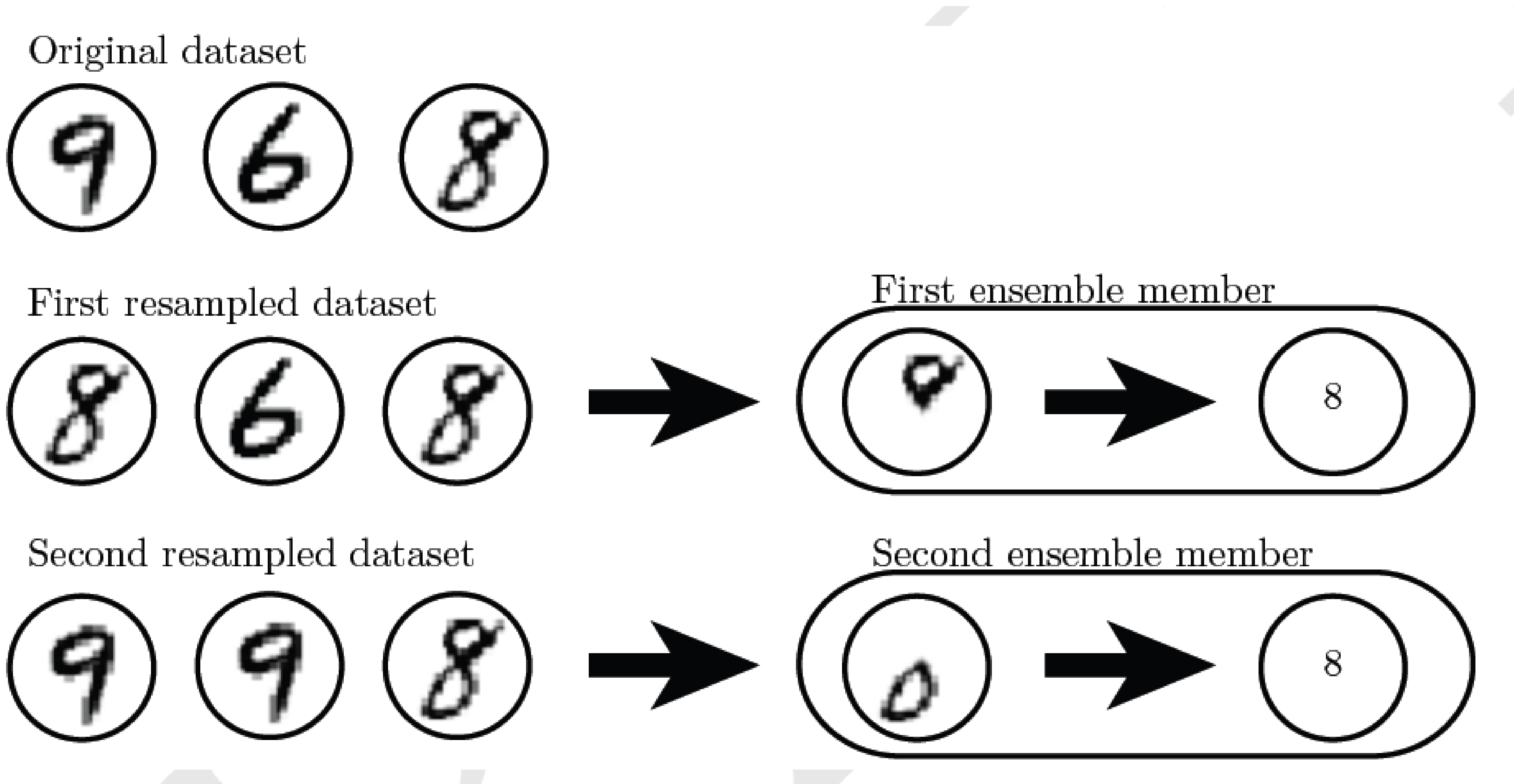
假设我们有 k 个回归模型。假设每个模型在每个例子上的误差是 ϵ_i，这个误差服从零均值方差为 $\mathbb{E}[\epsilon_i^2] = v$ 且协方差为 $\mathbb{E}[\epsilon_i \epsilon_j] = c$ 的多维正态分布。通过所有集成模型的平均预测所得误差是 $\frac{1}{k}\sum_i \epsilon_i$。集成预测器平方误差的期望是

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right], \tag{7.50}$$

$$= \frac{1}{k}v + \frac{k-1}{k}c. \tag{7.51}$$

10. 集成方法

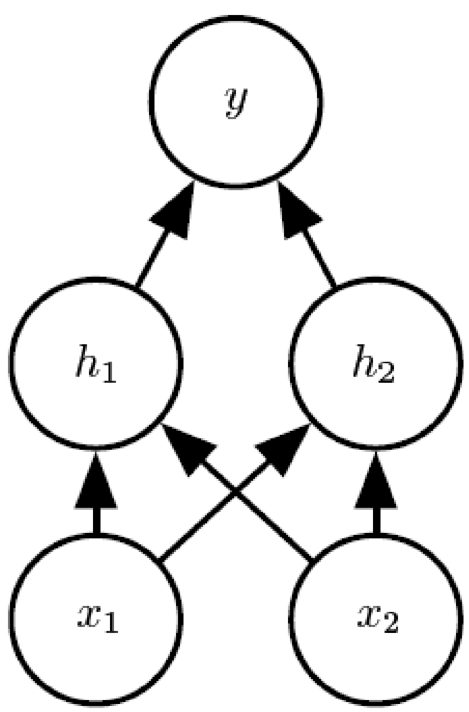
Bagging方法：从训练集有重复采样k个子集，每个子集独立训练一个分类器。

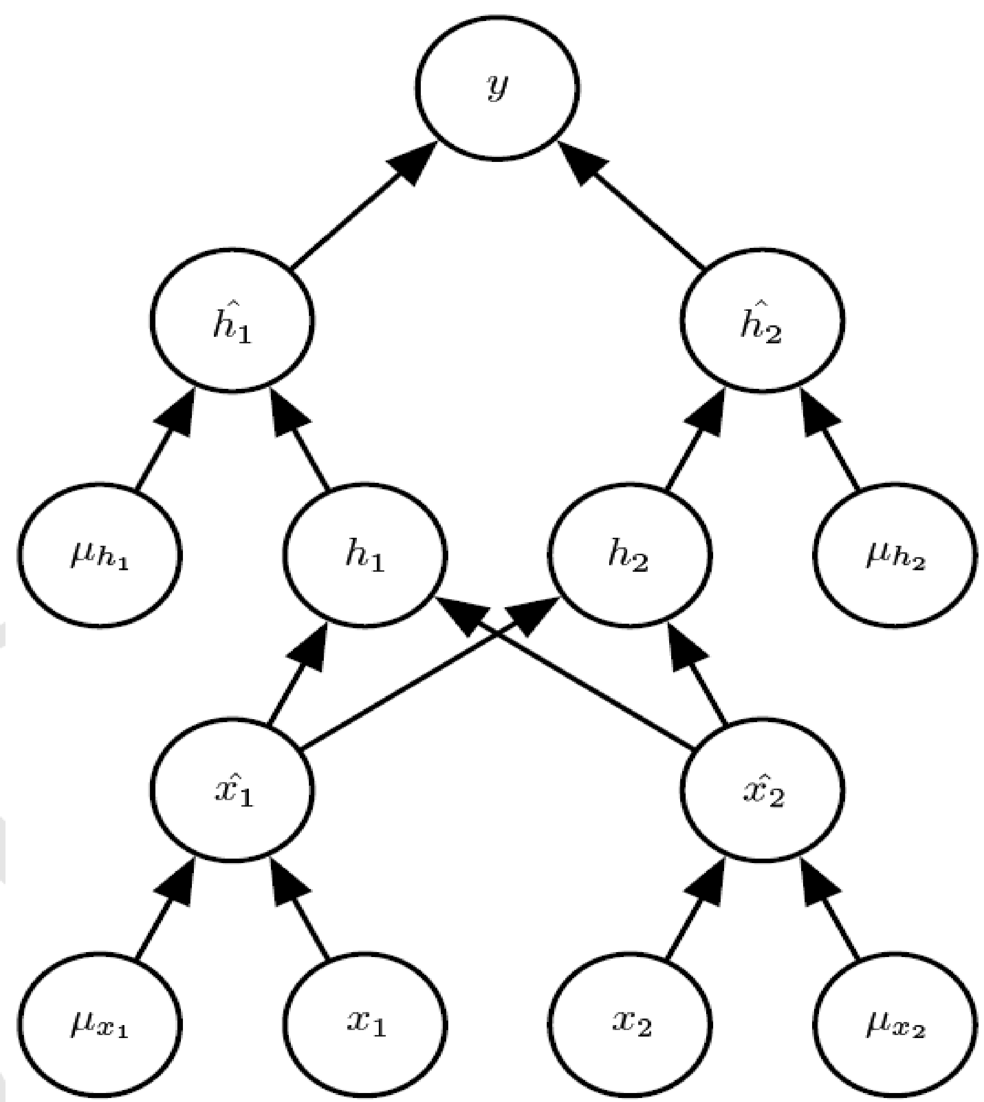


正则化策略：

11. dropout

思想：在模型上通过训练很多的子模型来实现近似Bagging方法。

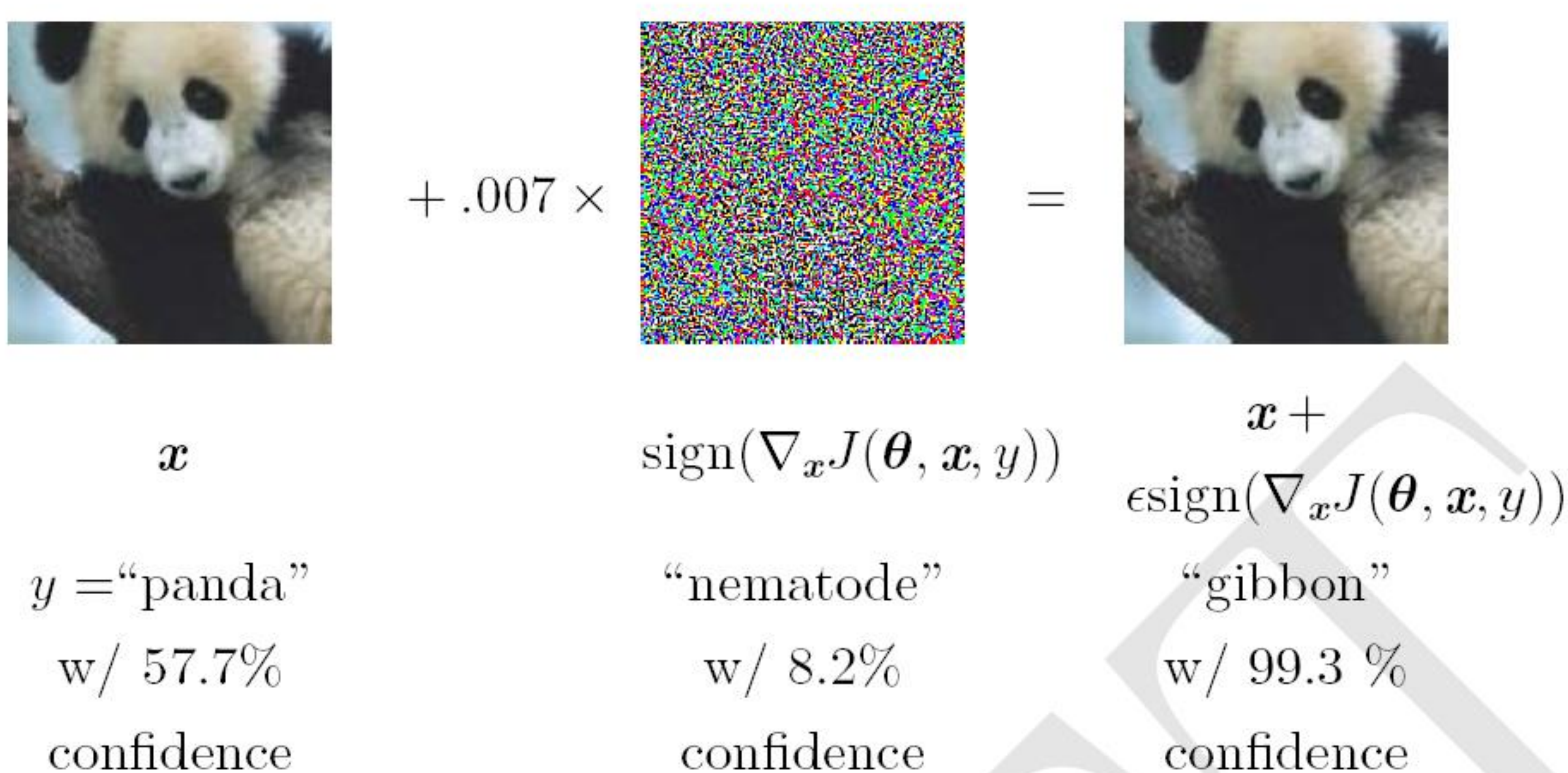




正则化策略：

12. 对抗训练

思想：生成对抗样本实现对抗训练。



深度前馈神经网络优化：

优化的目标：泛化误差最小

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathrm{y})\sim p_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

其中，L是代价函数，f(x,θ)输入x预测的输出，y是目标，Pdata是数据真实生成分布。

因Pdata不知道，所以优化为：

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathrm{y})\sim \hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

深度前馈神经网络优化：

通常我们只有数据样本，所以优化近似为：

$$\mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}}[L(f(\boldsymbol{x};\boldsymbol{\theta}),y)] = \frac{1}{m}\sum^{m}L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)})$$

上式称为经验风险最小化

优化策略：

1. 通常使用代理代价函数来代替真实的代价函数，如分类中用对数似然代替0-1代价函数，使得目标函数可导；

2. 训练算法：训练算法不会寻找一个局部最小点，而是在一个有较大导数地方提前终止，避免过度拟合。

优化策略：

3. 使用小批量的样本估计梯度（随机梯度下降）。

这是因为代价函数可以写完样本之和的形式，如对数似然

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}).$$

即经验分布上期望：

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

其梯度为：

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

优化策略：

3.使用小批量的样本估计梯度（随机梯度下降）。

优点：能够利用GPU进行计算，样本为2的幂次；样本具有高度相似性；

优化中存在的挑战：

1. 病态

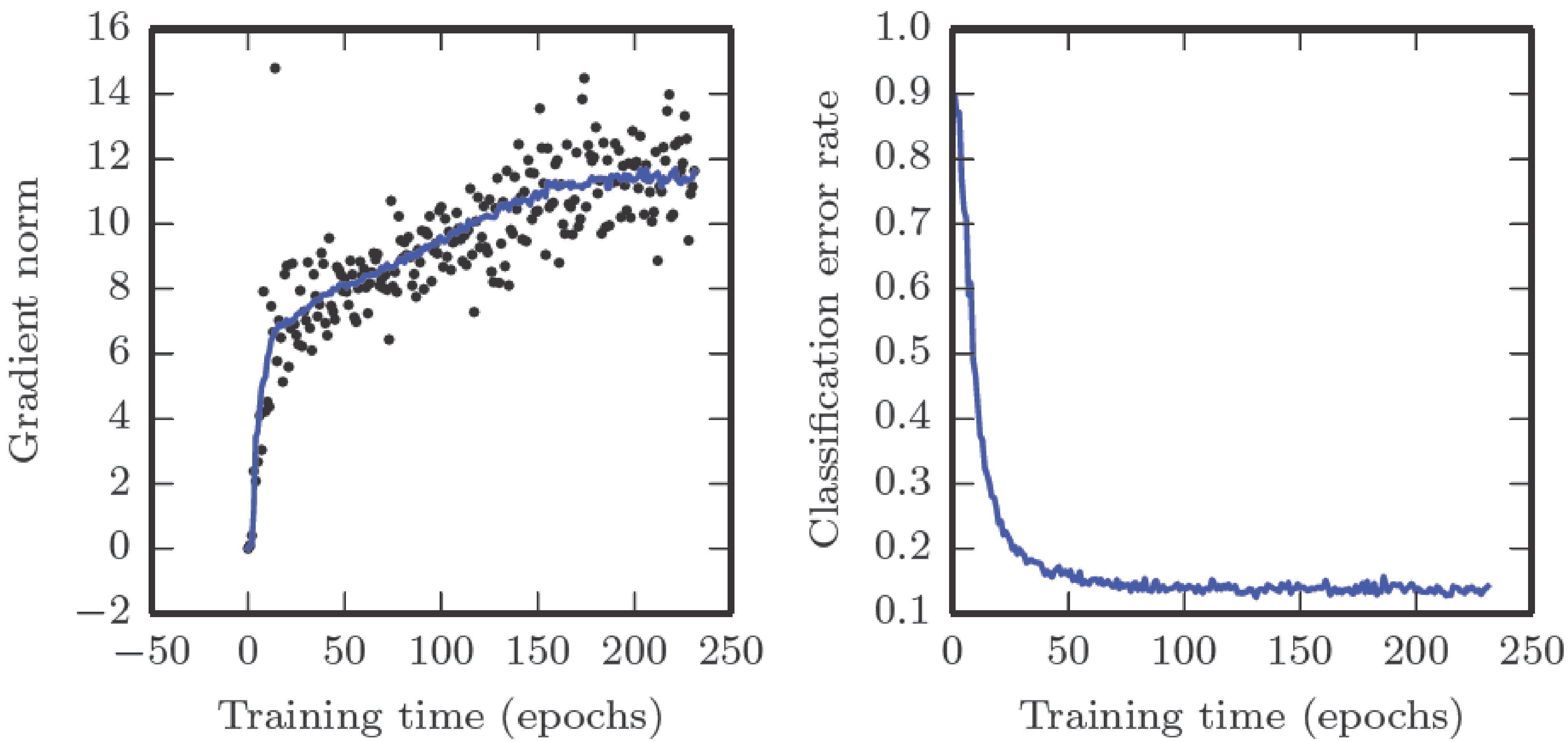
代价函数的二阶展式为：

$$\frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g} - \epsilon \boldsymbol{g}^\top \boldsymbol{g}$$

当第一项值超过第二项时，病态问题出现。此时尽管梯度很强，但学校缓慢。方法：**通过收缩学习率**。

优化中存在的挑战：

梯度在增强，误差减少，但目标函数没有到极值点



优化中存在的挑战：

2. 局部极小点问题

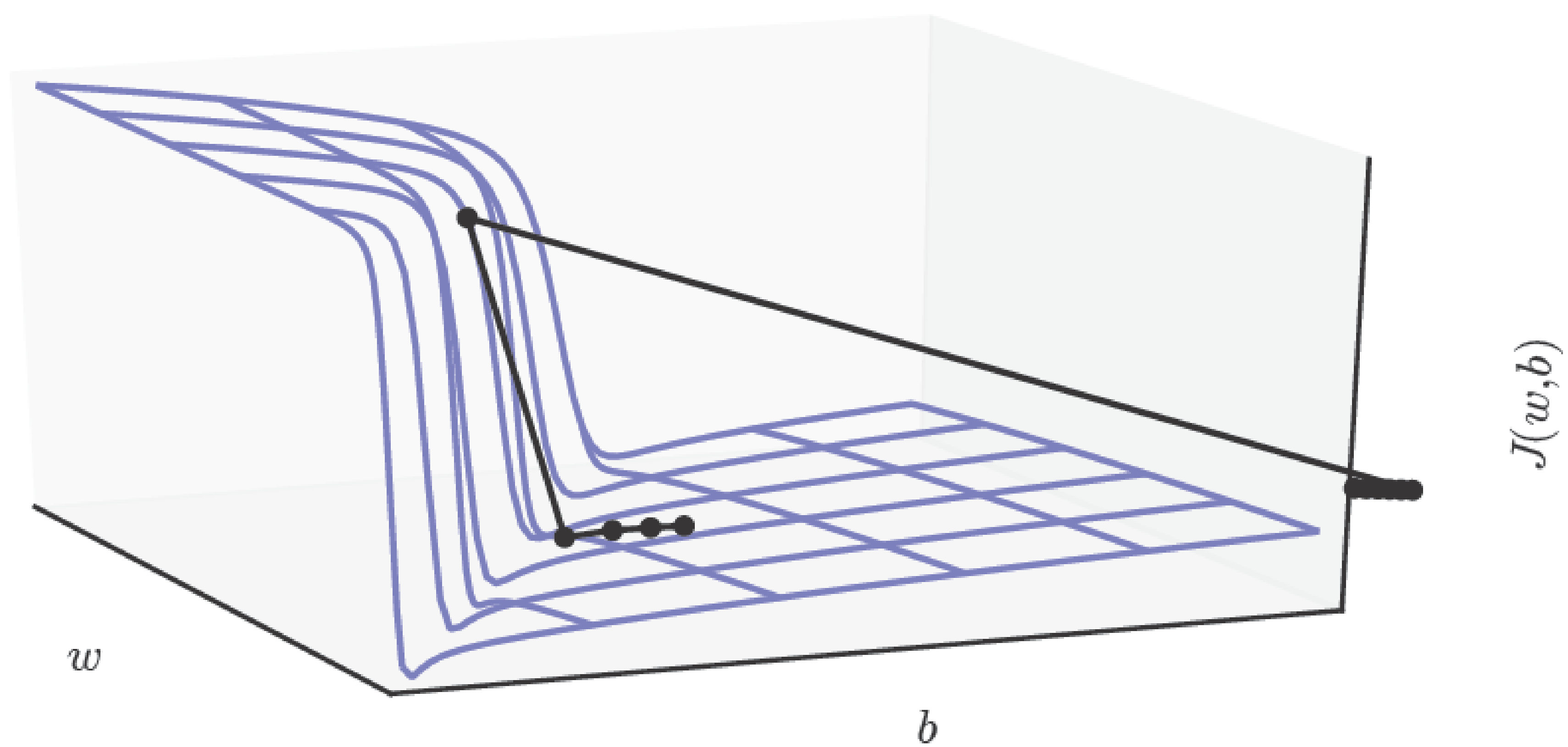
高维空间有很多局部极小点，学习会在局部极小点而无法逃逸该点。

3. 高原、鞍点和平坦区域

4. 悬崖和梯度爆炸

优化中存在的挑战：

4. 悬崖和梯度爆炸



优化中存在的挑战：

5. 长期依赖

在循环神经网络，如果存在很深的路径，则

$$\boldsymbol{W}^t = (\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1})^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

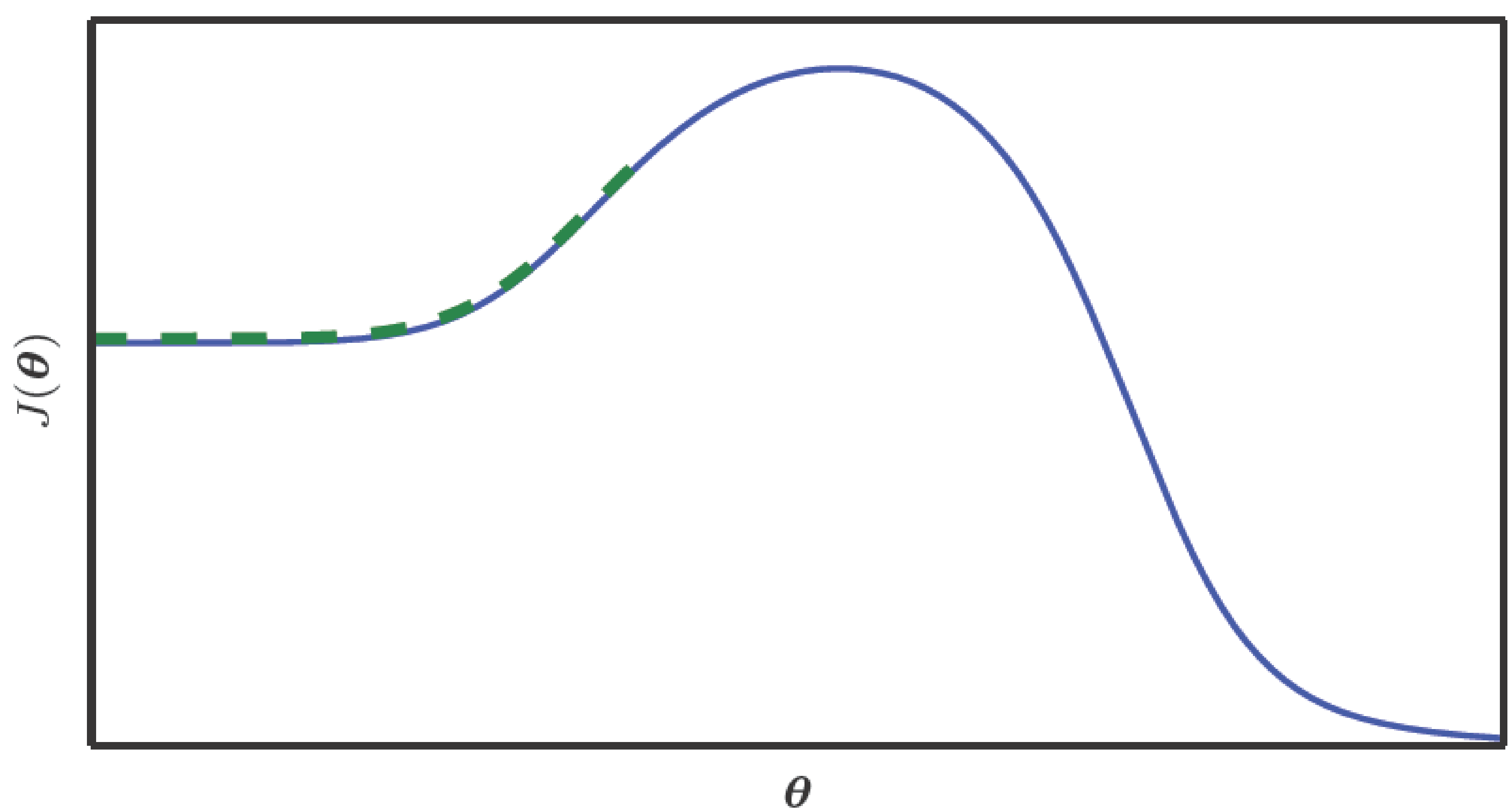
当特征值大于1时，梯度爆炸，当特征值小于1时，则梯度消失。

6. 非精确梯度

优化中存在的挑战：

5. 局部与全局弱对应问题

全局最优点离我们遥远，梯度下降又不能指向到达该点的路径。



基本算法：

1. 随机梯度下降算法

算法 8.1 随机梯度下降（SGD）在第 k 个训练迭代的更新

Require: 学习率 ϵ_k

Require: 初始参数 $\boldsymbol{\theta}$

while 停止准则未满足 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度估计：$\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$

end while

基本算法：

1. 随机梯度下降算法

实践中，一般会线性衰减学习率直到第 τ 次迭代：

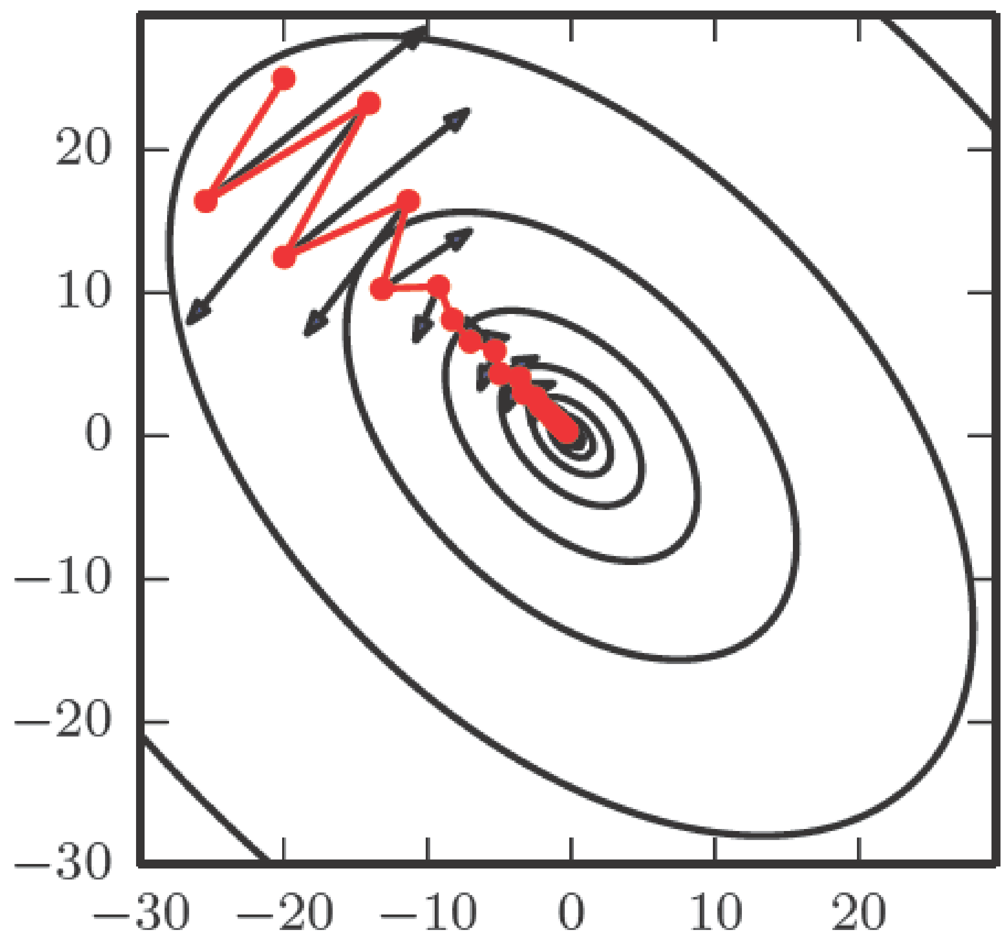
$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

其中 $\alpha = \frac{k}{\tau}$。在 τ 步迭代之后，一般使 ϵ 保持常数。

基本算法：

2. 动量

处理高曲率，平坦区域和带噪声的梯度



基本算法：

2. 动量

更新规则如下：

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

基本算法：

2. 动量

算法如下：

算法 8.2 使用动量的随机梯度下降（SGD）

Require: 学习率 ϵ，动量参数 α

Require: 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

while 没有达到停止准则 **do**

  从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

  计算梯度估计：$\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

  计算速度更新：$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$

  应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

基本算法：

3. 初始化方法

策略：随机初始化为非零的小权值；非监督学习的初始化。

4. 自适应学习率算法

AdaGrad，RMProp等

5. 二阶优化方法

牛顿法等