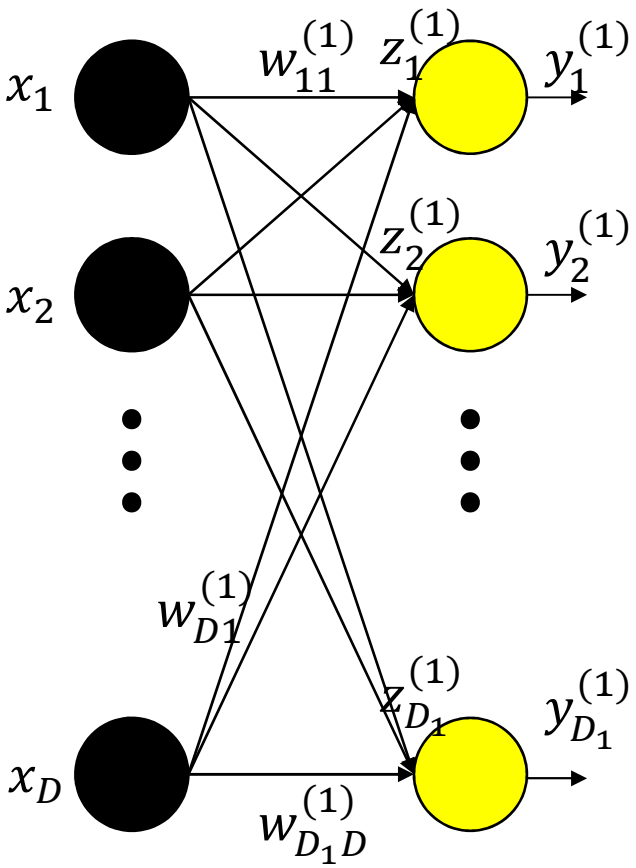
Vector formulation

- For layered networks it is generally simpler to think of the process in terms of vector operations
  - Simpler arithmetic
  - Fast matrix libraries make operations *much* faster

- We can restate the entire process in vector terms
  - On slides, please read
  - This is what is *actually* used in any real system
  - Will appear in quiz

Vector formulation

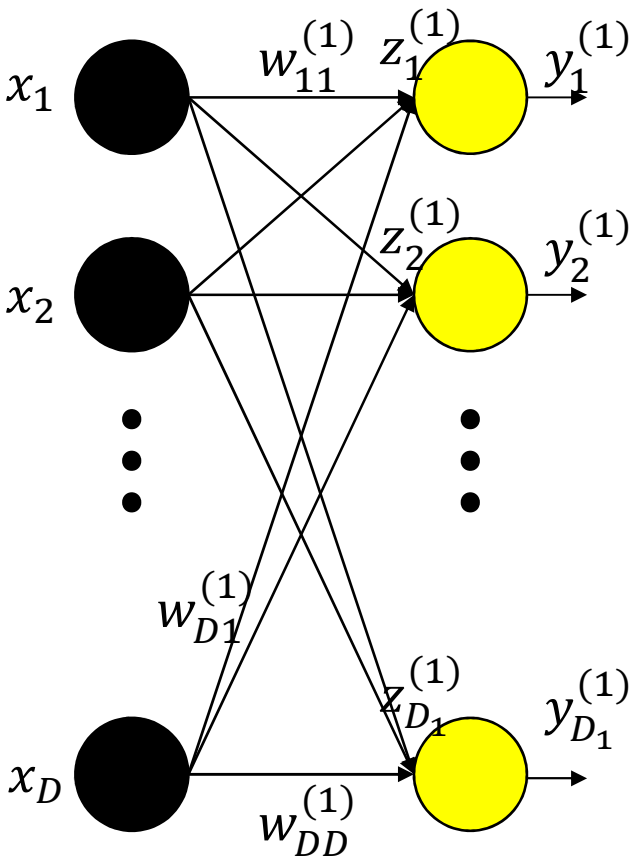


$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \qquad \mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

$$\mathbf{W}_k = \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \vdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \vdots & w_{D_{k-1}2}^{(k)} \\ \cdots & \cdots & \ddots & \vdots \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix} \qquad \mathbf{b_k} = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{D_{k+1}}^{(k)} \end{bmatrix}$$

- Arrange all inputs to the network in a vector $\mathbf{x}$
- Arrange the *inputs* to neurons of the kth layer as a vector $\mathbf{z_k}$
- Arrange the outputs of neurons in the kth layer as a vector $\mathbf{y_k}$
- Arrange the weights to any layer as a matrix $\mathbf{W}_k$
  - Similarly with biases

Vector formulation



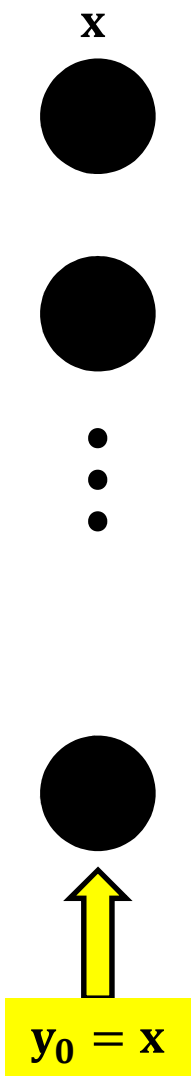
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \qquad \mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

$$\mathbf{W}_k = \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \vdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \vdots & w_{D_{k-1}2}^{(k)} \\ \cdots & \cdots & \ddots & \vdots \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix} \qquad \mathbf{b_k} = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{D_{k+1}}^{(k)} \end{bmatrix}$$

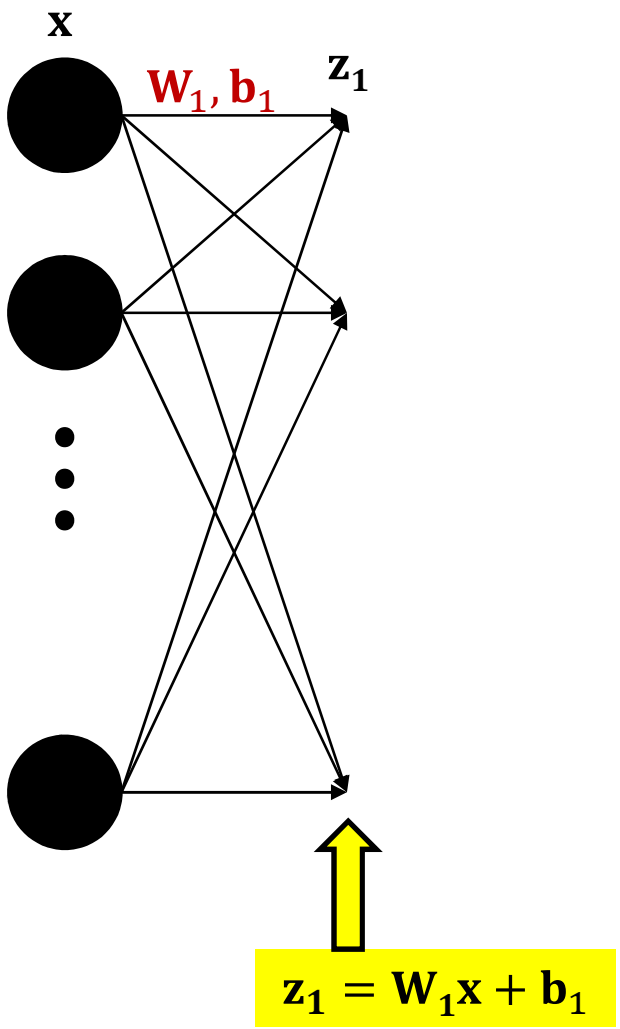
- The computation of a single layer is easily expressed in matrix notation as (setting $\mathbf{y_0} = \mathbf{x}$):

$$\mathbf{z_k} = \mathbf{W_k y_{k-1}} + \mathbf{b_k} \qquad \mathbf{y_k} = \boldsymbol{f}_k(\mathbf{z_k})$$

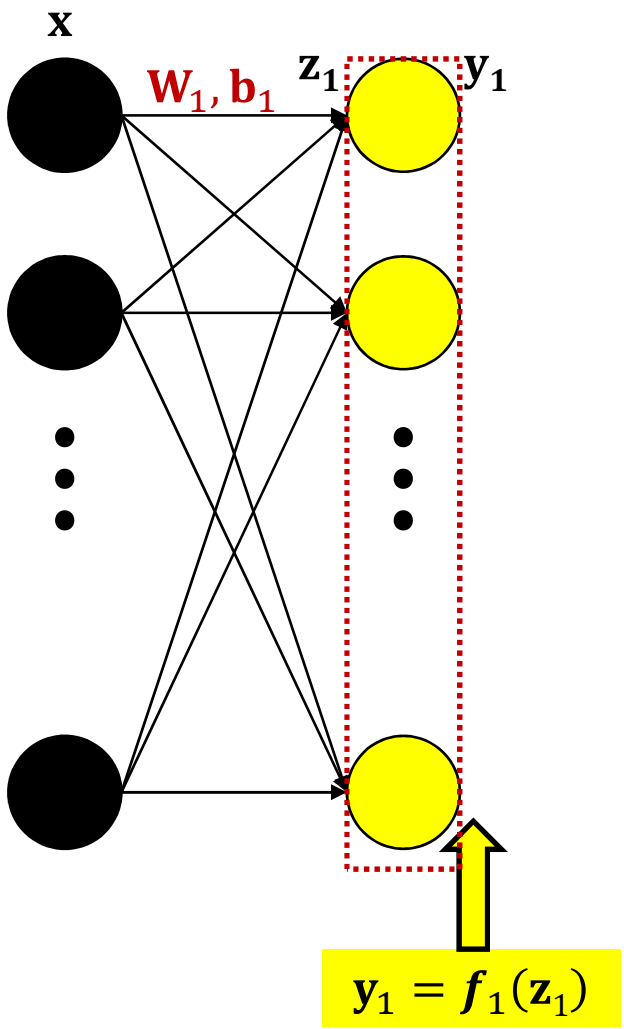
The forward pass: Evaluating the network



The forward pass



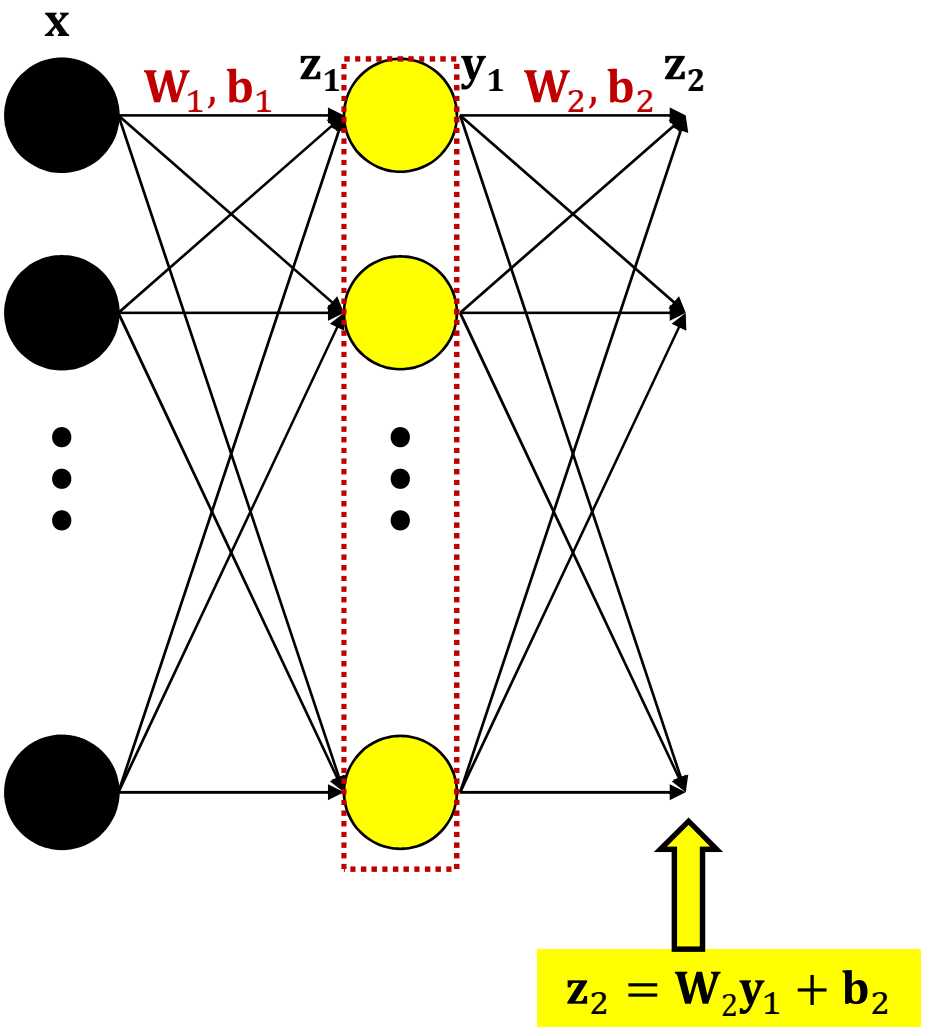
The forward pass



The Complete computation

$$\mathbf{y}_1 = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

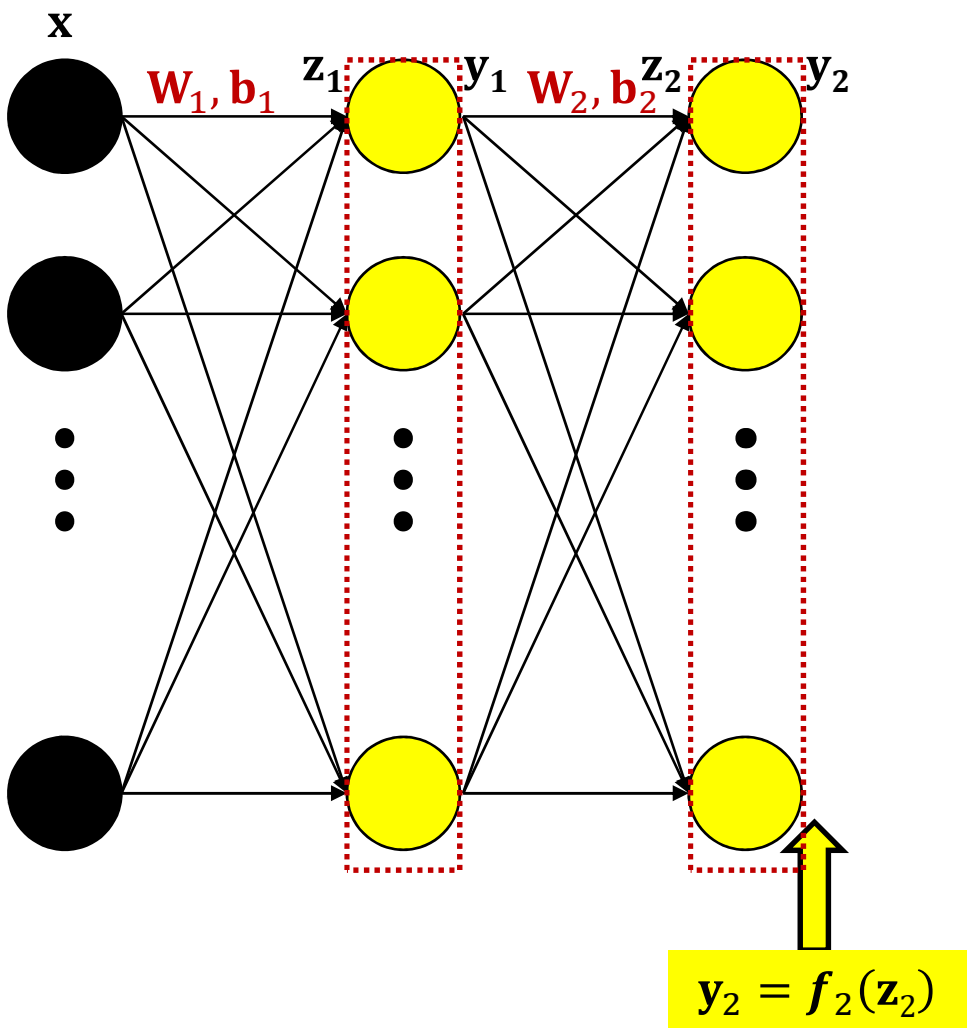
The forward pass



The Complete computation

$$\mathbf{y}_1 = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

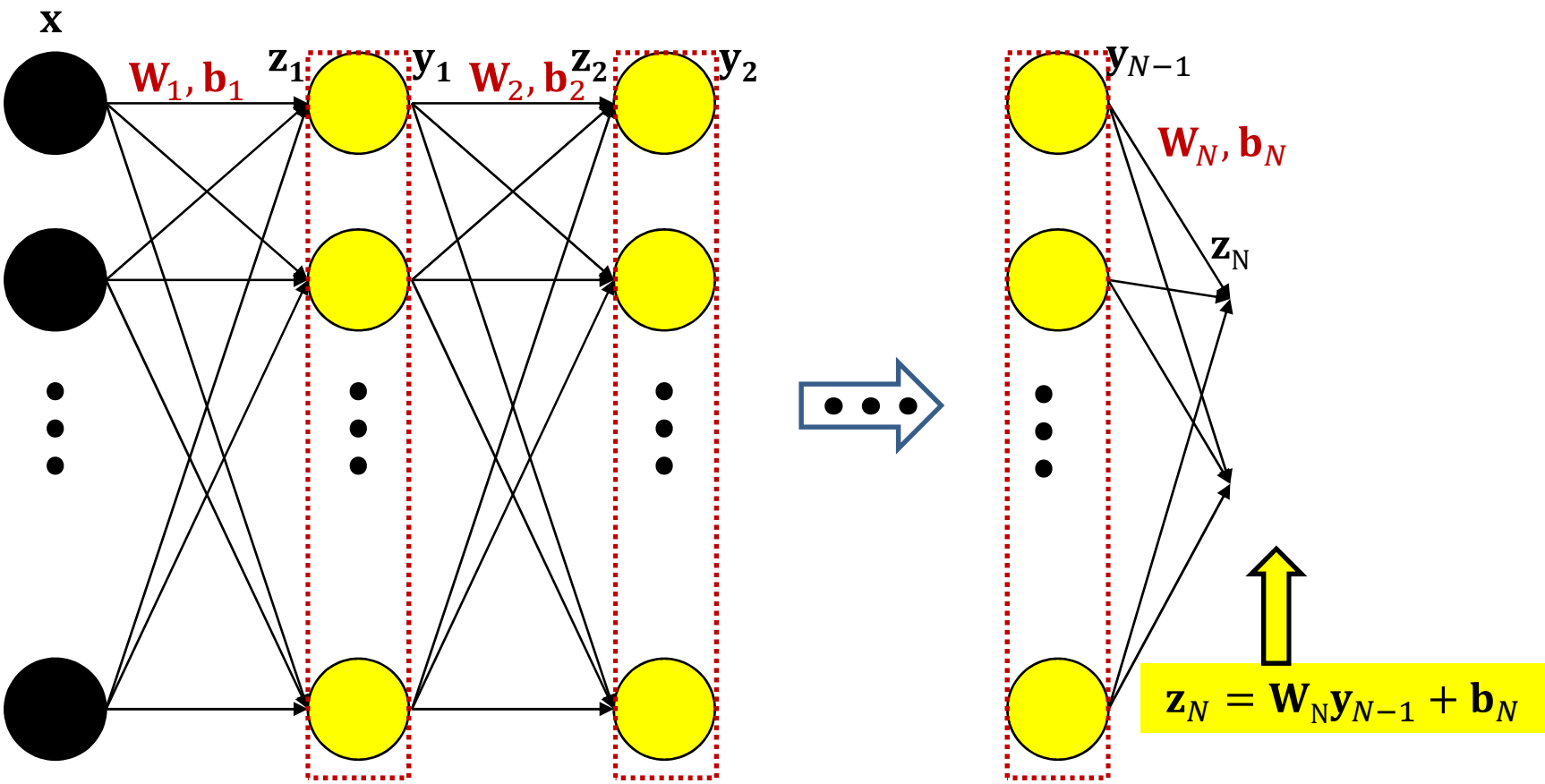
The forward pass



The Complete computation

$$\mathbf{y}_2 = f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

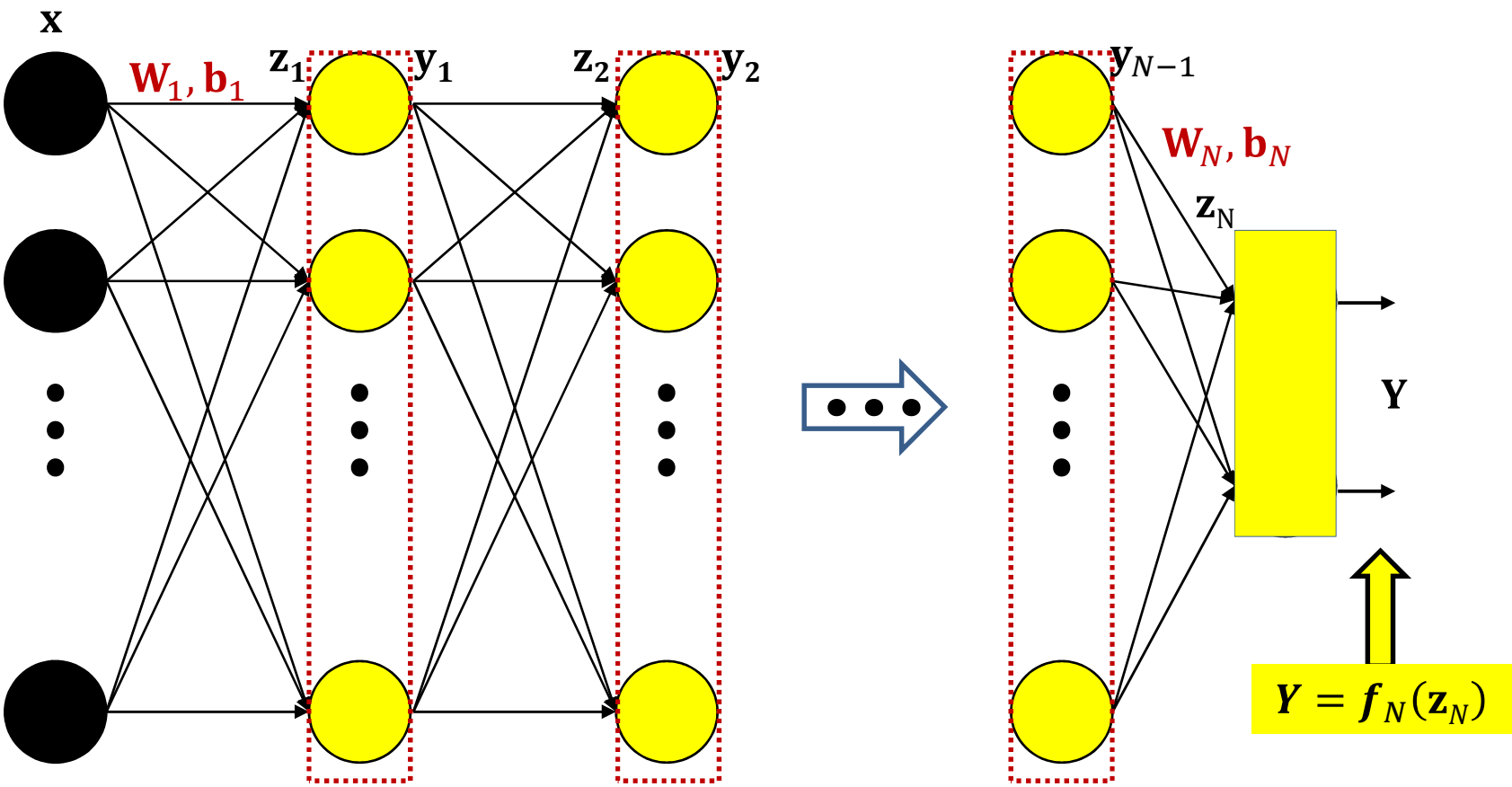
The forward pass



The Complete computation

$$\mathbf{y}_2 = f_2(\mathbf{W}_2 f_1(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

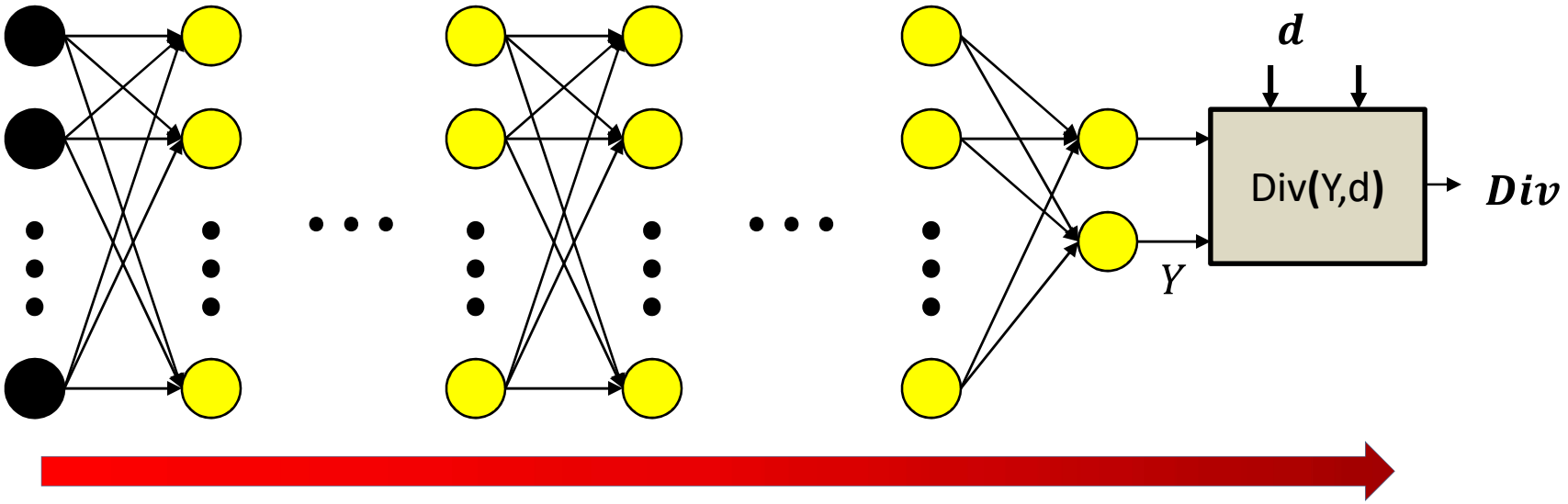
The forward pass



The Complete computation

$$\boldsymbol{Y} = f_N(\mathbf{W}_N f_{N-1}(\dots f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\dots) + \mathbf{b}_N)$$

Forward pass



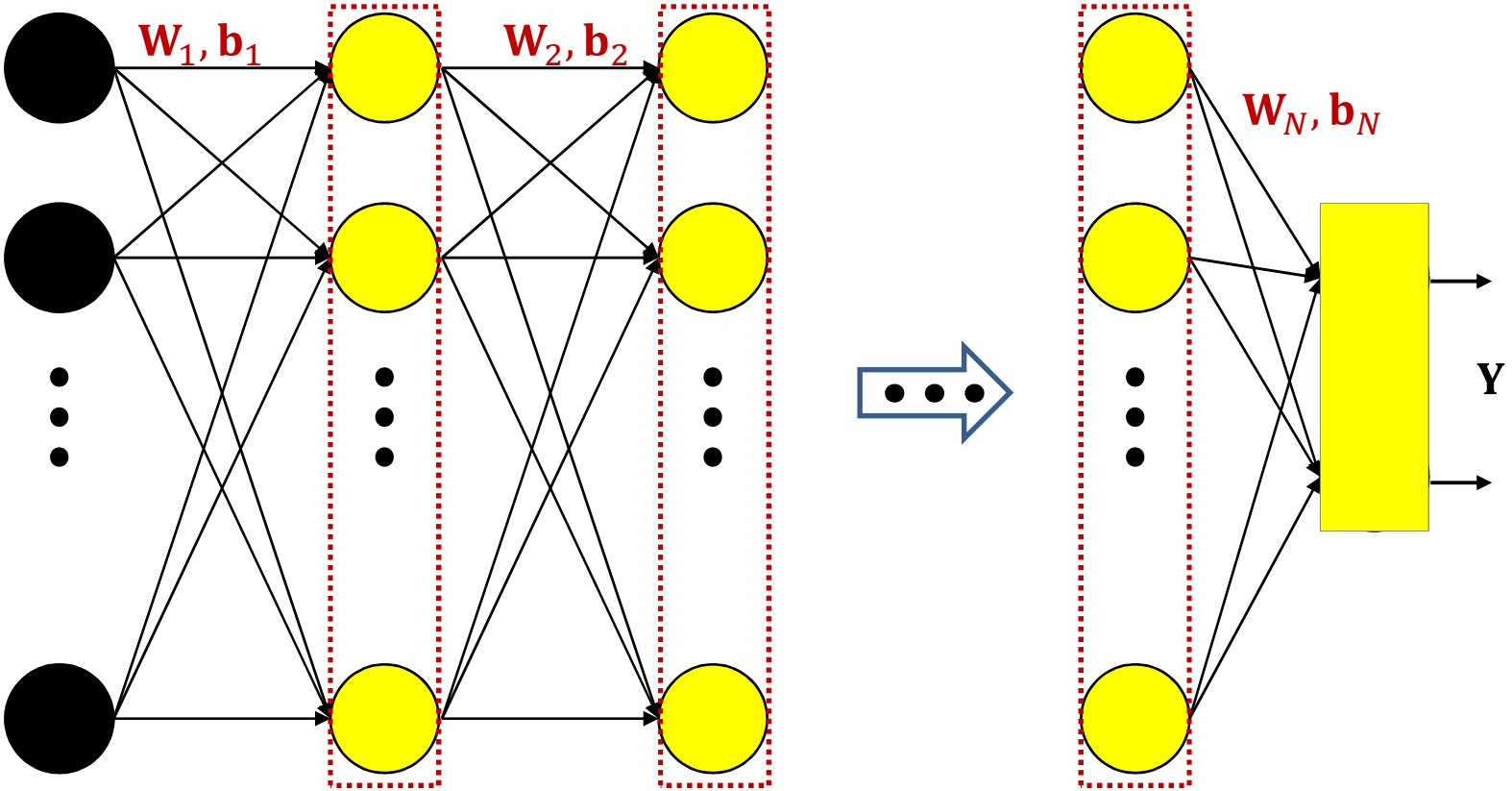
Forward pass:

Initialize $\mathbf{y}_0 = \mathbf{x}$

For k = 1 to N: $\mathbf{z}_k = \mathbf{W}_k \mathbf{y}_{k-1} + \mathbf{b}_k$ $\mathbf{y}_k = \boldsymbol{f}_k(\mathbf{z}_k)$

Output $\mathrm{Y} = \mathbf{y}_N$

The backward pass



- The network is a nested function

$$\boldsymbol{Y} = f_N(\mathbf{W}_N f_{N-1}(\dots f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\dots) + \mathbf{b}_N)$$

- The error for any $\mathbf{x}$ is also a nested function

$$Div(Y, d) = Div(f_N(\mathbf{W}_N f_{N-1}(\dots f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\dots) + \mathbf{b}_N), d)$$

Calculus recap 2: The Jacobian

- The derivative of a vector function w.r.t. vector input is called a *Jacobian*
- It is the matrix of partial derivatives given below

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = f\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_D \end{bmatrix}\right)$$

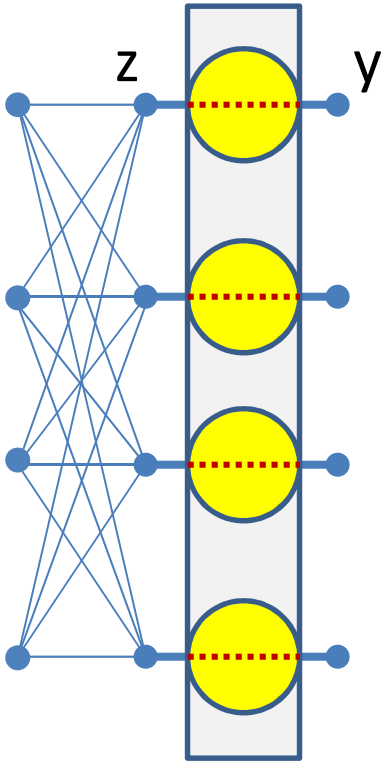
Using vector notation

$$\mathbf{y} = f(\mathbf{z})$$

$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_D} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_D} \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial y_M}{\partial z_1} & \frac{\partial y_M}{\partial z_2} & \cdots & \frac{\partial y_M}{\partial z_D} \end{bmatrix}$$

Check: $\Delta\mathbf{y} = J_{\mathbf{y}}(\mathbf{z})\Delta\mathbf{z}$

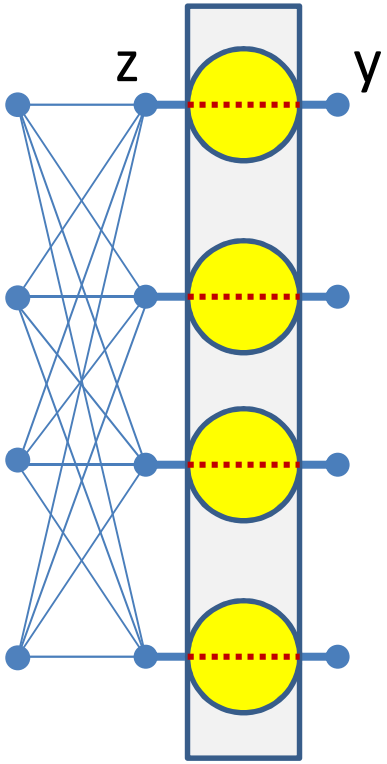
Jacobians can describe the derivatives of neural activations w.r.t their input



$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{dy_1}{dz_1} & 0 & \cdots & 0 \\ 0 & \frac{dy_2}{dz_2} & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \frac{dy_D}{dz_D} \end{bmatrix}$$

- **For Scalar activations**
  - Number of outputs is identical to the number of inputs
- Jacobian is a diagonal matrix
  - Diagonal entries are individual derivatives of outputs w.r.t inputs
  - Not showing the superscript "(k)" in equations for brevity

Jacobians can describe the derivatives of neural activations w.r.t their input

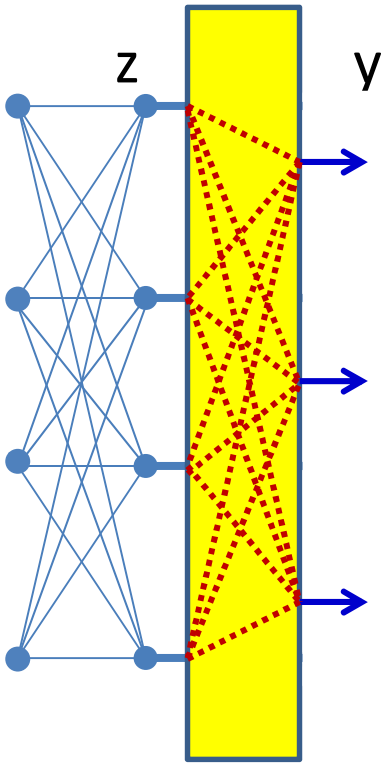


$$y_i = f(z_i)$$

$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} f'(y_1) & 0 & \cdots & 0 \\ 0 & f'(y_2) & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & f'(y_M) \end{bmatrix}$$

- **For scalar activations (shorthand notation):**
  - Jacobian is a diagonal matrix
  - Diagonal entries are individual derivatives of outputs w.r.t inputs

For *Vector* activations



$$J_{\boldsymbol{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_D} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_D} \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial y_M}{\partial z_1} & \frac{\partial y_M}{\partial z_2} & \cdots & \frac{\partial y_M}{\partial z_D} \end{bmatrix}$$

- Jacobian is a full matrix
  - Entries are partial derivatives of individual outputs w.r.t individual inputs

Special case: Affine functions

$$\mathbf{z} = \mathbf{W}\mathbf{y} + \mathbf{b}$$

$$J_{\mathbf{z}}(\mathbf{y}) = \mathbf{W}$$

- Matrix $\mathbf{W}$ and bias $\mathbf{b}$ operating on vector $\mathbf{y}$ to produce vector $\mathbf{z}$
- The Jacobian of $\mathbf{z}$ w.r.t $\mathbf{y}$ is simply the matrix $\mathbf{W}$

Vector derivatives: Chain rule

- We can define a chain rule for Jacobians
- **For vector functions of vector inputs:**

$$\mathbf{y} = \boldsymbol{f}(\boldsymbol{g}(\mathbf{x}))$$

$$\mathbf{z} = \boldsymbol{g}(\mathbf{x})$$

$$\mathbf{y} = \boldsymbol{f}(\mathbf{z})$$

$$J_{\mathbf{y}}(\mathbf{x}) = J_{\mathbf{y}}(\mathbf{z})J_{\boldsymbol{z}}(\mathbf{x})$$

Check

$$\Delta\mathbf{z} = J_{\mathbf{z}}(\mathbf{x})\Delta\mathbf{x}$$

$$\Delta\mathbf{y} = J_{\mathbf{y}}(\mathbf{z})\Delta\mathbf{z}$$

$$\Delta\mathbf{y} = J_{\mathbf{y}}(\mathbf{z})J_{\mathbf{z}}(\mathbf{x})\Delta\mathbf{x} = J_{\mathbf{y}}(\mathbf{x})\Delta\mathbf{x}$$

Note the order: The derivative of the outer function comes first

Vector derivatives: Chain rule

- *The chain rule can combine Jacobians and Gradients*
- **For *scalar* functions of vector inputs ($\boldsymbol{g}()$ is vector):**

$$D = f\big(\boldsymbol{g}(\mathbf{x})\big)$$

$$\mathbf{z} = \boldsymbol{g}(\mathbf{x})$$

$$D = f(\mathbf{z})$$

$$\nabla_{\mathbf{x}} D = \nabla_{\mathbf{z}}(D) J_{\mathbf{z}}(\mathbf{x})$$

Check

$$\Delta\mathbf{z} = J_{\mathbf{z}}(\mathbf{x})\Delta\mathbf{x}$$

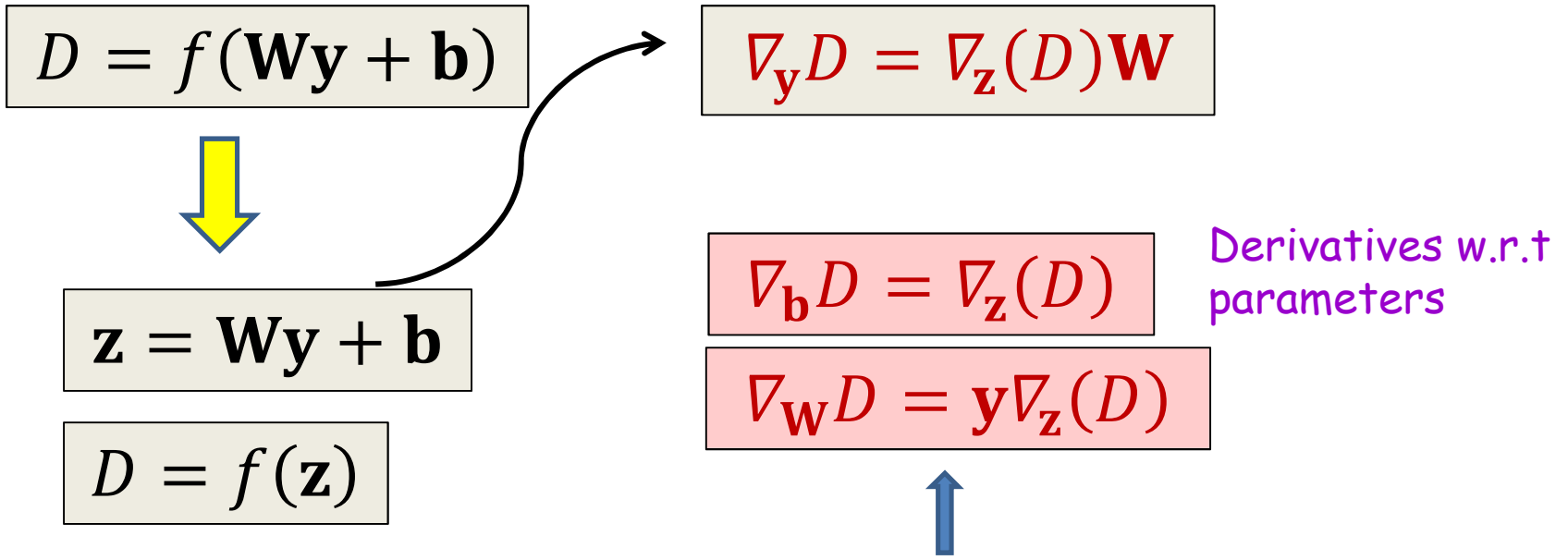
$$\Delta D = \nabla_{\mathbf{z}}(D)\Delta\mathbf{z}$$

$$\Delta D = \nabla_{\mathbf{z}}(D) J_{\mathbf{z}}(\mathbf{x})\Delta\mathbf{x} = \nabla_{\mathbf{x}} D\,\Delta\mathbf{x}$$

Note the order: The derivative of the outer function comes first

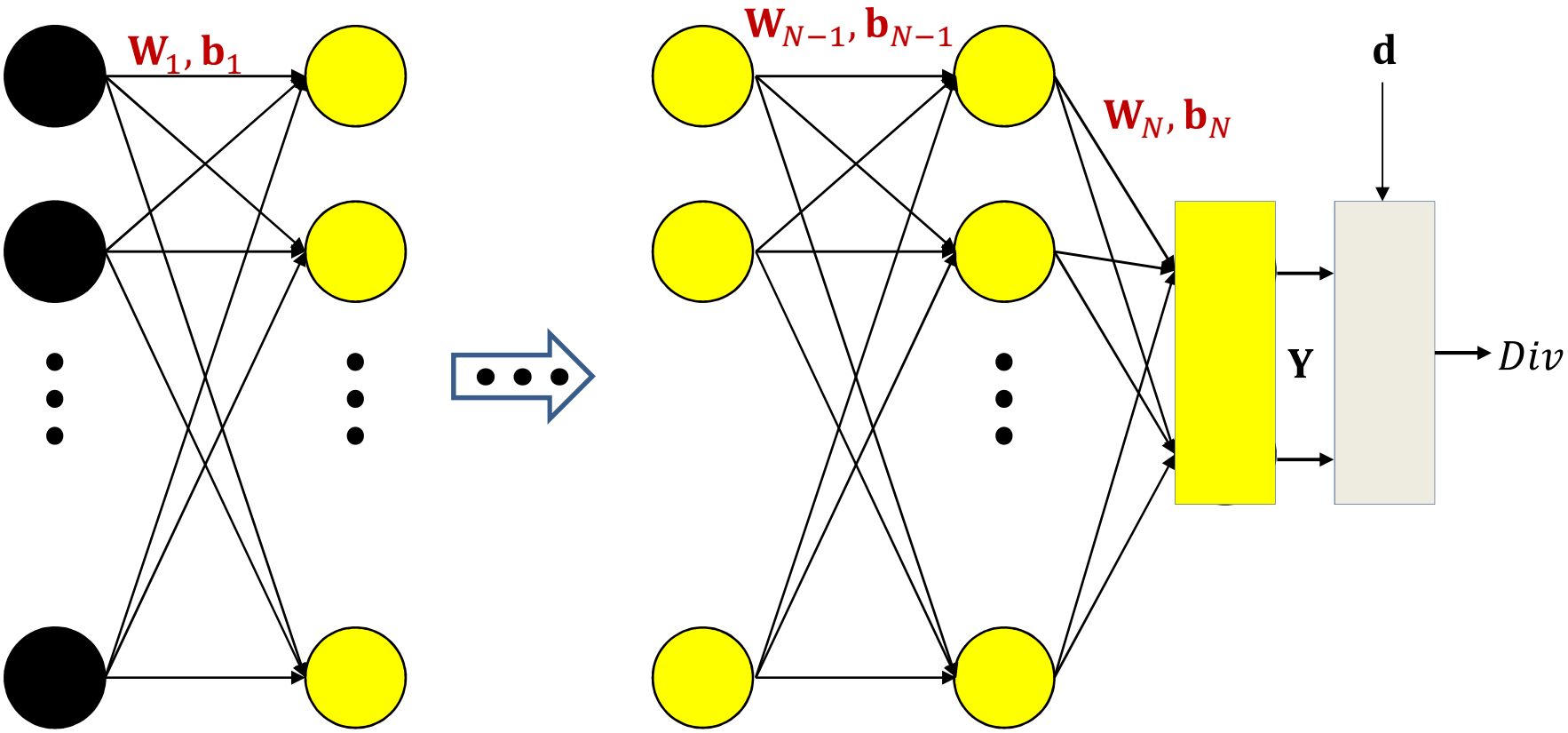
Special Case

- Scalar functions of Affine functions



Note reversal of order. This is in fact a simplification of a product of tensor terms that occur in the *right* order

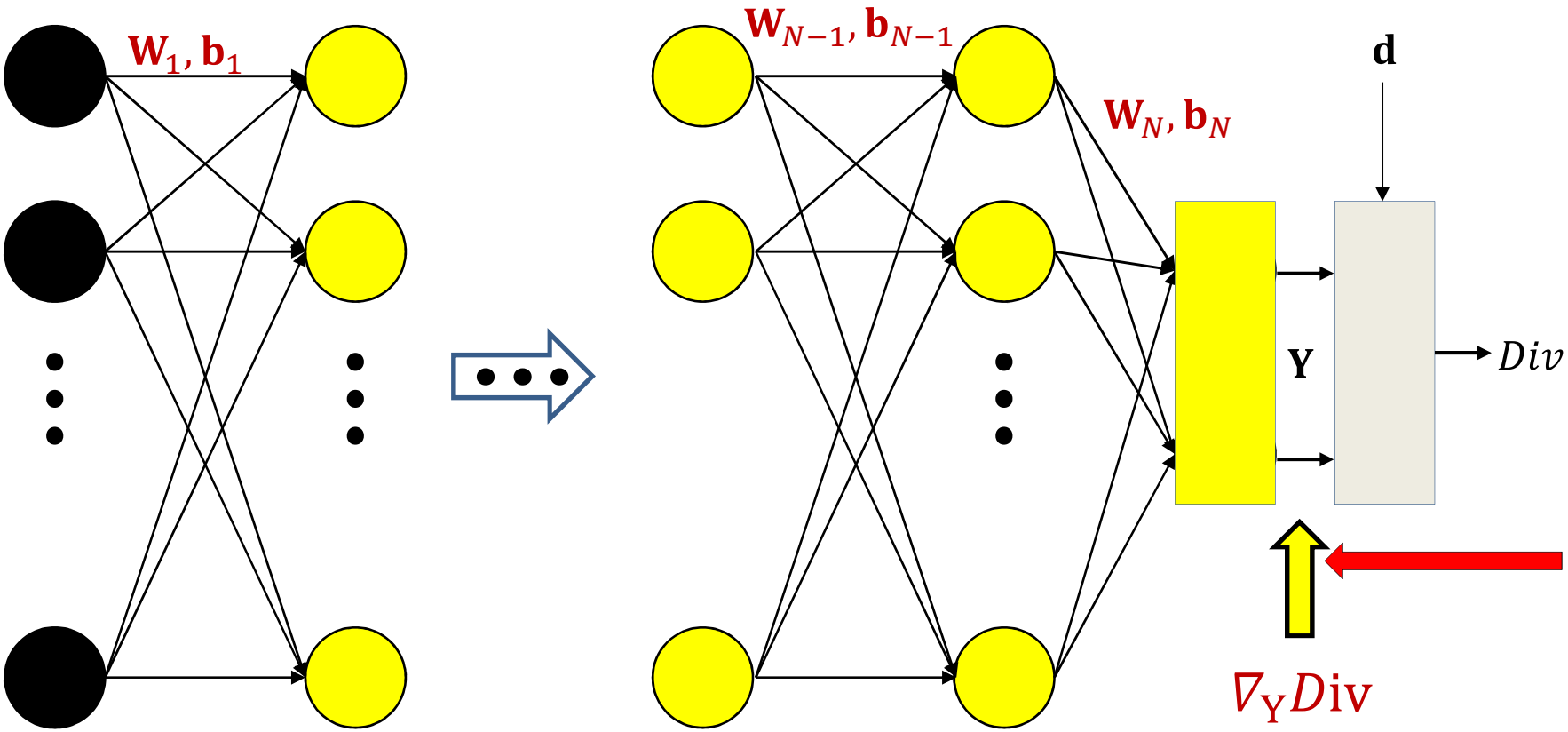
The backward pass



In the following slides we will also be using the notation $\nabla_{\mathbf{z}}\mathbf{Y}$ to represent the Jacobian $J_{\mathbf{Y}}(\boldsymbol{z})$ to explicitly illustrate the chain rule

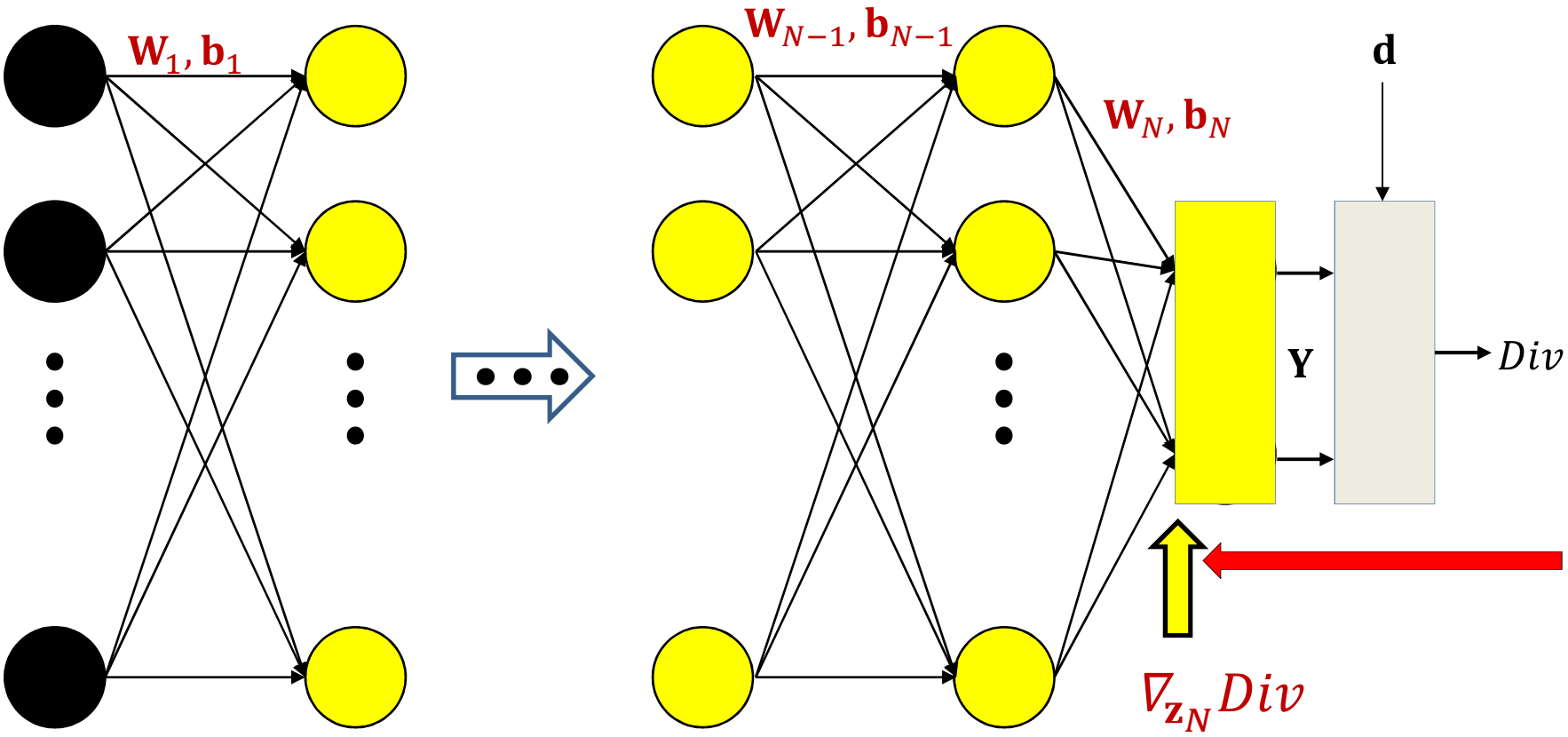
In general $\nabla_{\mathbf{a}}\mathbf{b}$ represents a derivative of $\mathbf{b}$ w.r.t. $\mathbf{a}$ and could be a gradient (for scalar $\mathbf{b}$) Or a Jacobian (for vector $\mathbf{b}$)

The backward pass



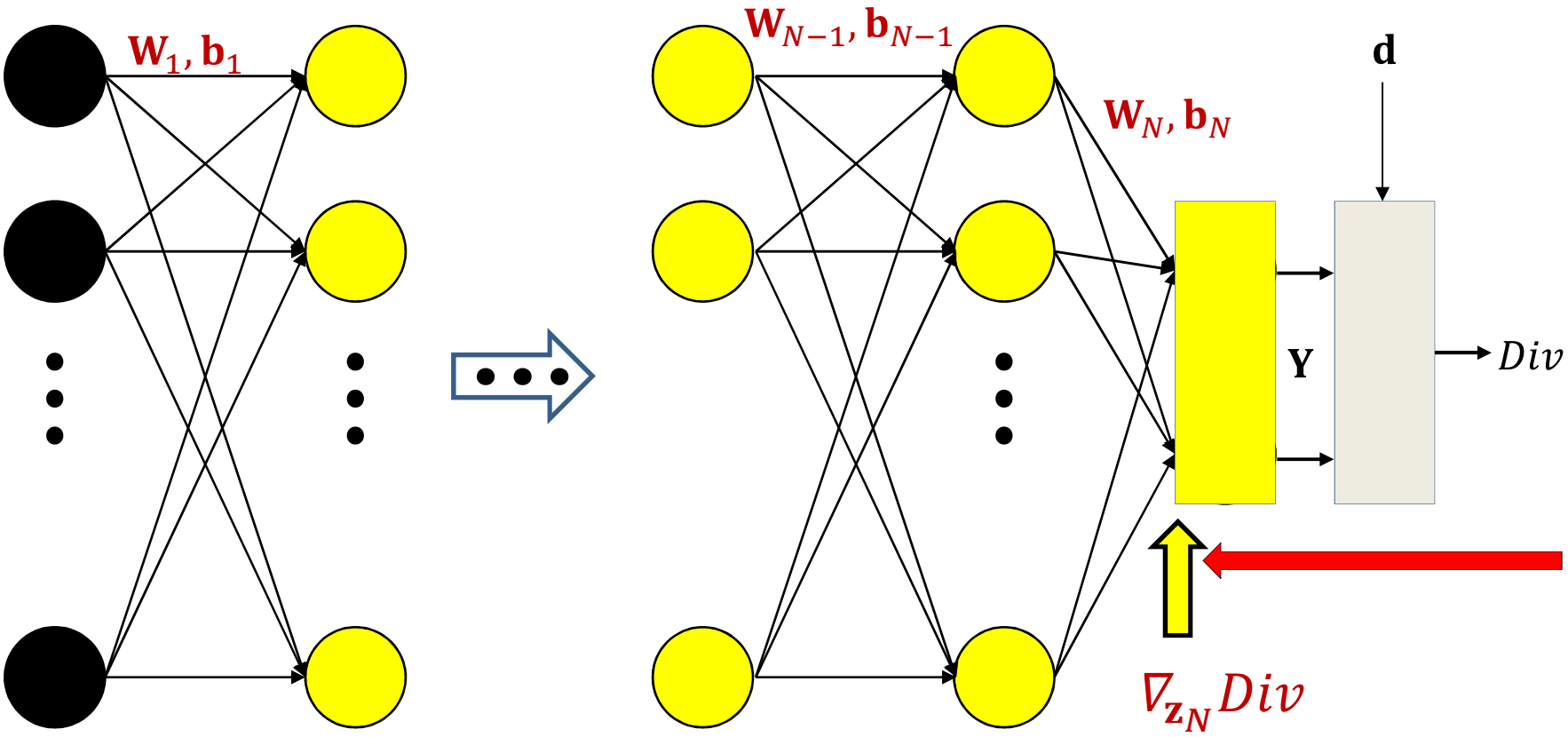
First compute the gradient of the divergence w.r.t. Y.
The actual gradient depends on the divergence function.

The backward pass



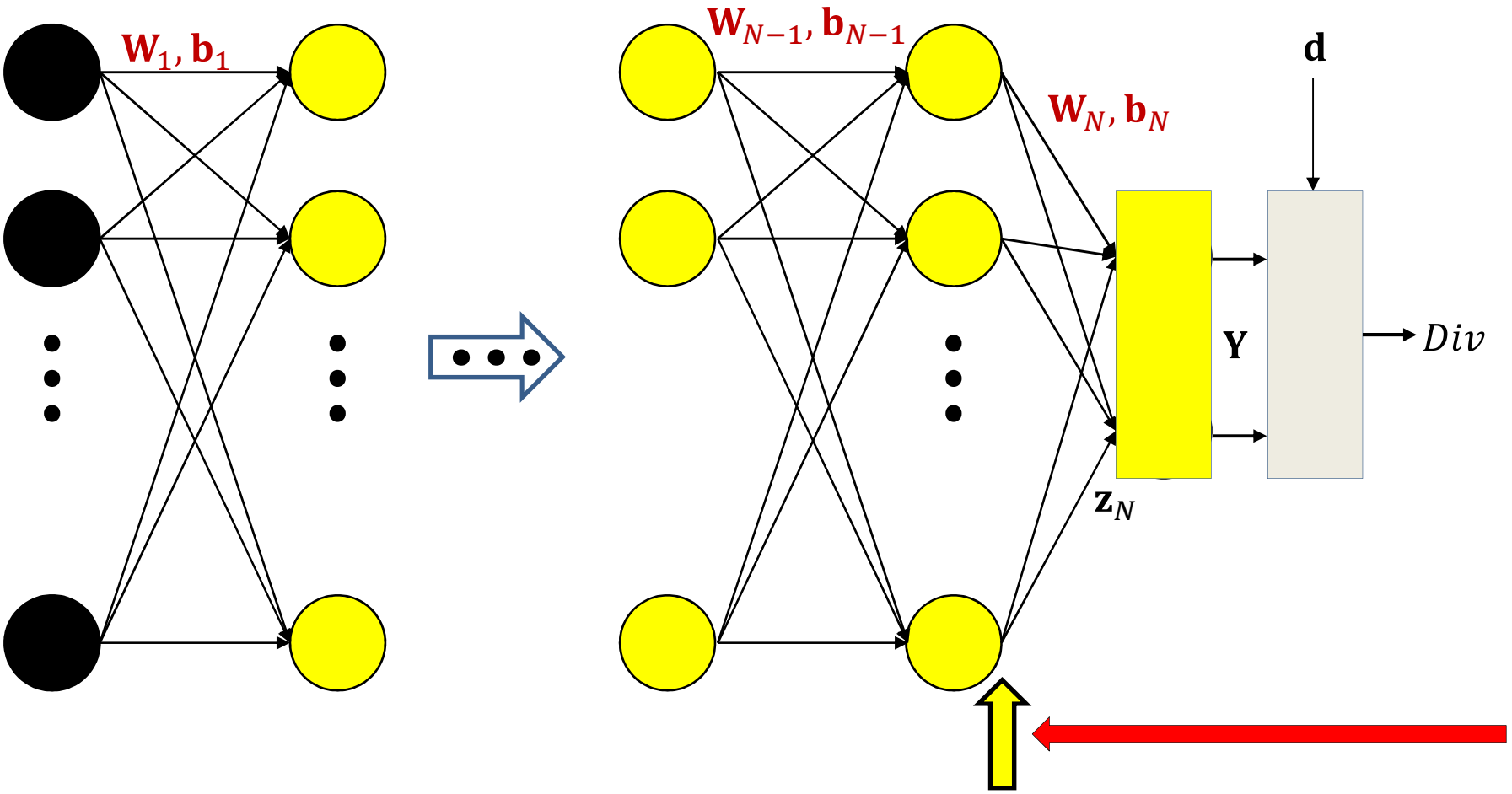
$$\nabla_{\mathbf{z}_N} Div = \nabla_{\mathbf{Y}} Div . \nabla_{\mathbf{z}_N} \mathbf{Y}$$

The backward pass



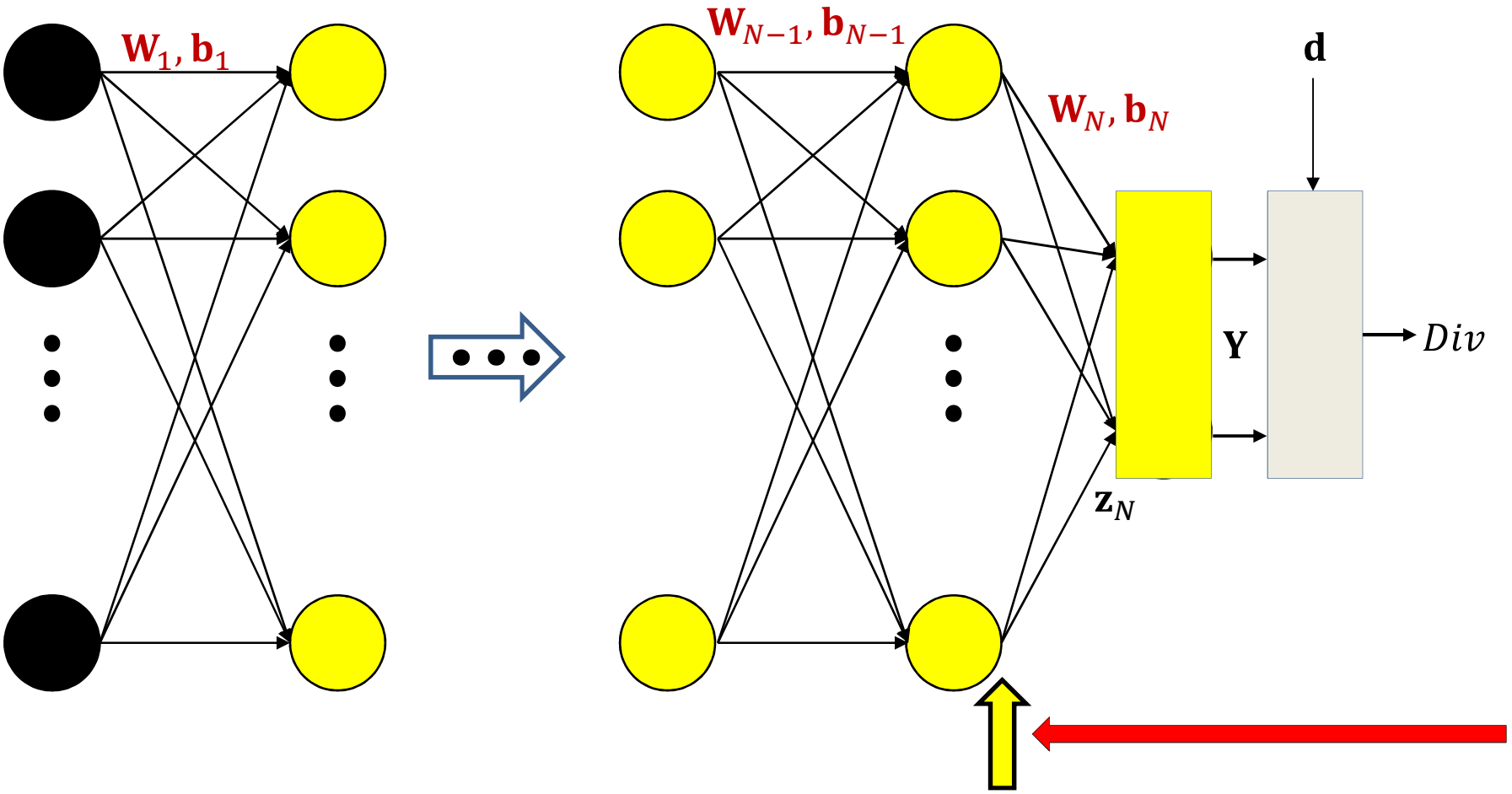
$$\nabla_{\mathbf{z}_N} Div = \nabla_{\mathbf{Y}} Div \, J_{\mathbf{Y}}(\mathbf{z}_N)$$

The backward pass



$$\nabla_{\mathbf{y}_{N-1}} Div = \nabla_{\mathbf{z}_N} Div . \nabla_{\mathbf{y}_{N-1}} \mathbf{z}_N$$

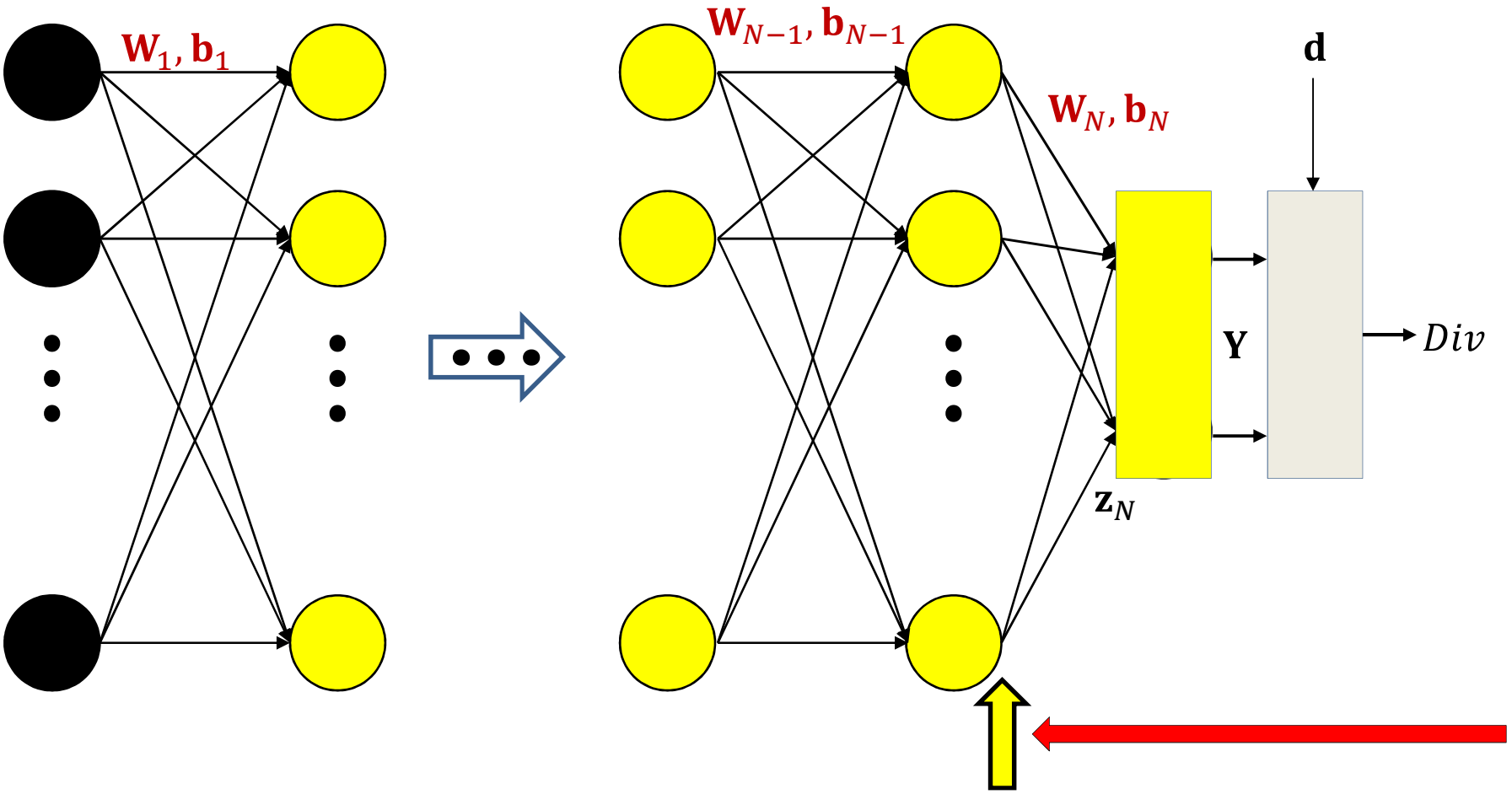
The backward pass



$$\nabla_{\mathbf{y}_{N-1}} Div = \nabla_{\mathbf{z}_N} Div \, \mathbf{W}_N$$

$$\nabla_{\mathbf{y}_{N-1}} Div$$

The backward pass

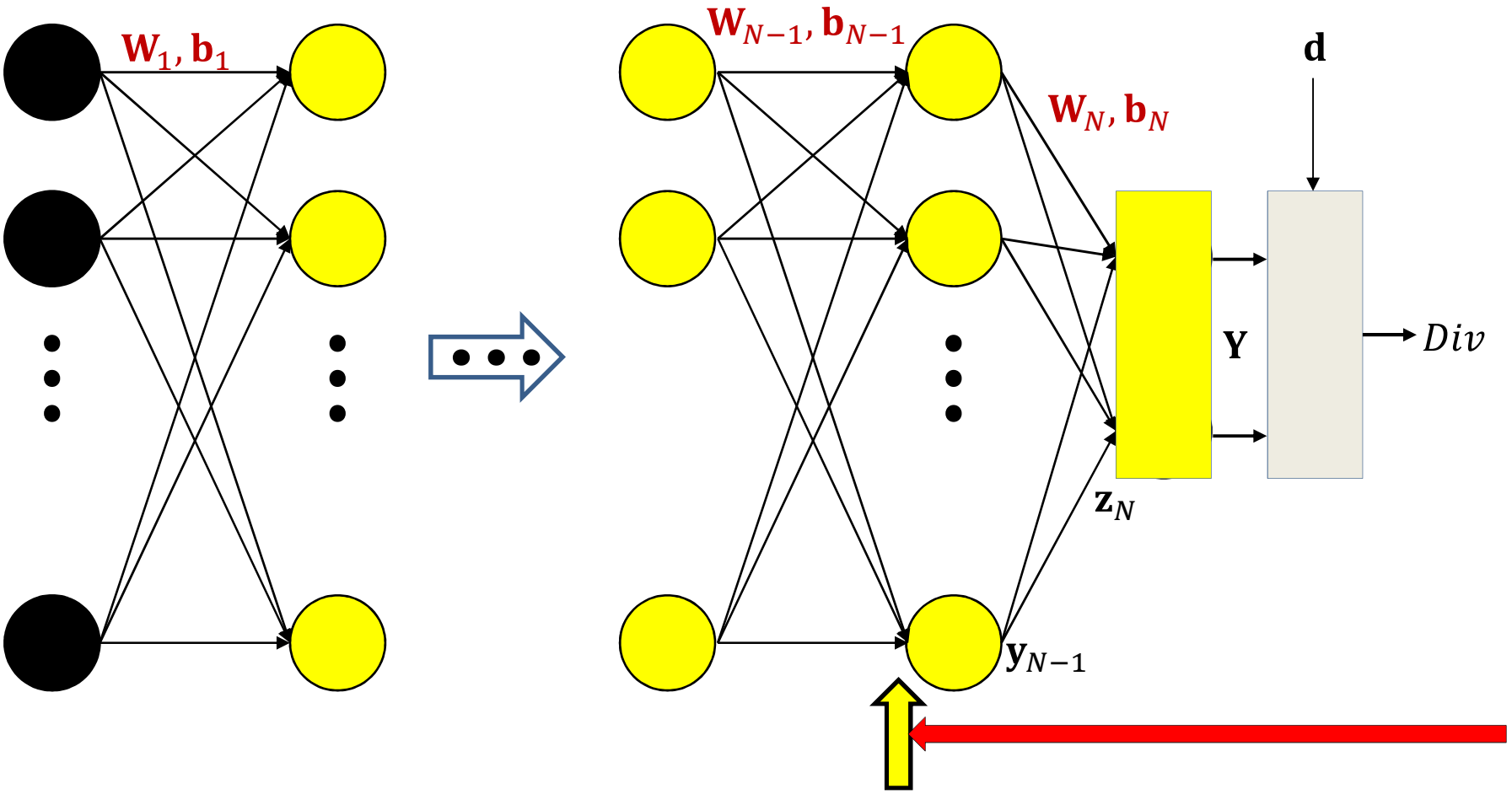


$$\nabla_{\mathbf{y}_{N-1}} Div = \nabla_{\mathbf{z}_N} Div \, \mathbf{W}_N$$

$$\nabla_{\mathbf{W}_N} Div = \mathbf{y}_{N-1} \nabla_{\mathbf{z}_N} Div$$

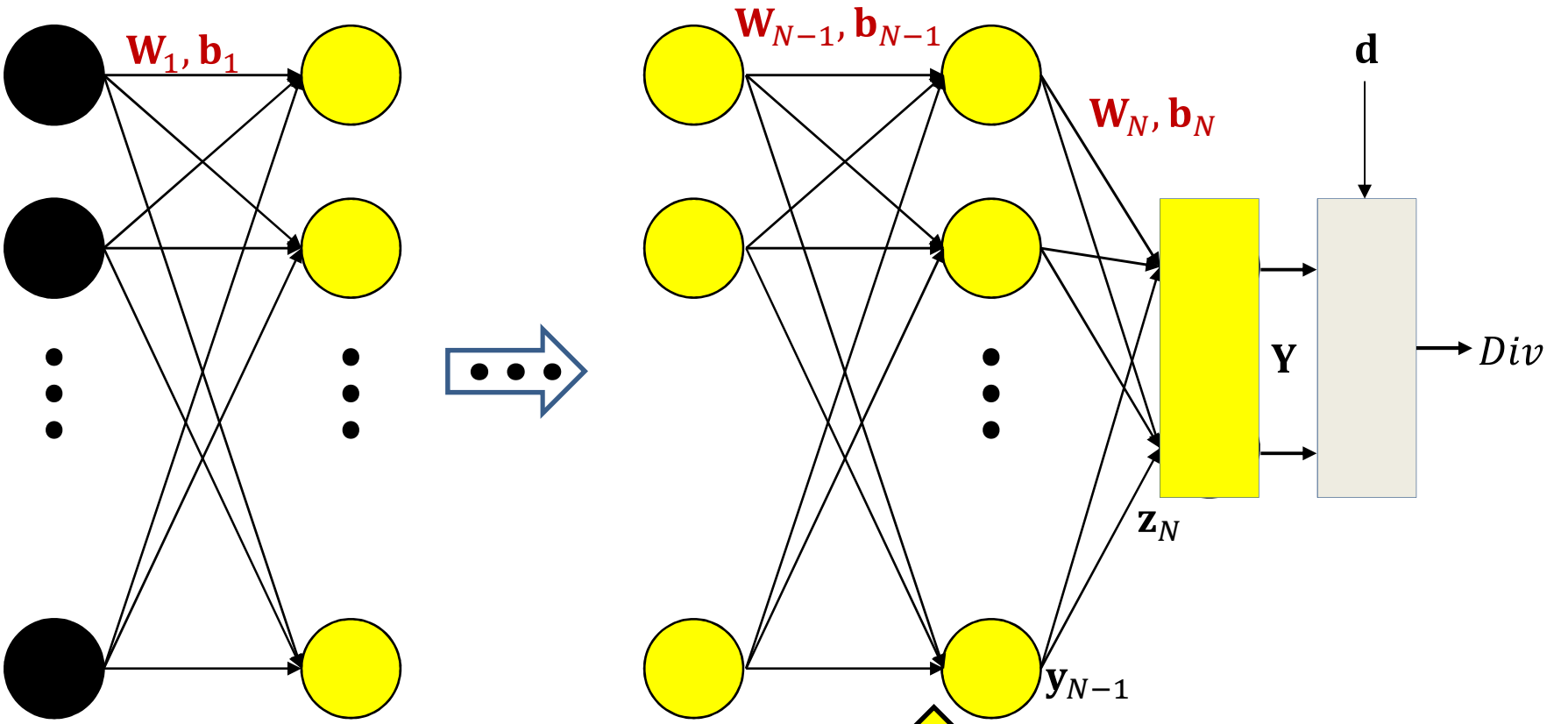
$$\nabla_{\mathbf{b}_N} Div = \nabla_{\mathbf{z}_N} Div$$

The backward pass



$$\nabla_{\mathbf{z}_{N-1}} Div = \nabla_{\mathbf{y}_{N-1}} Div . \nabla_{\mathbf{z}_{N-1}} \mathbf{y}_{N-1}$$

The backward pass

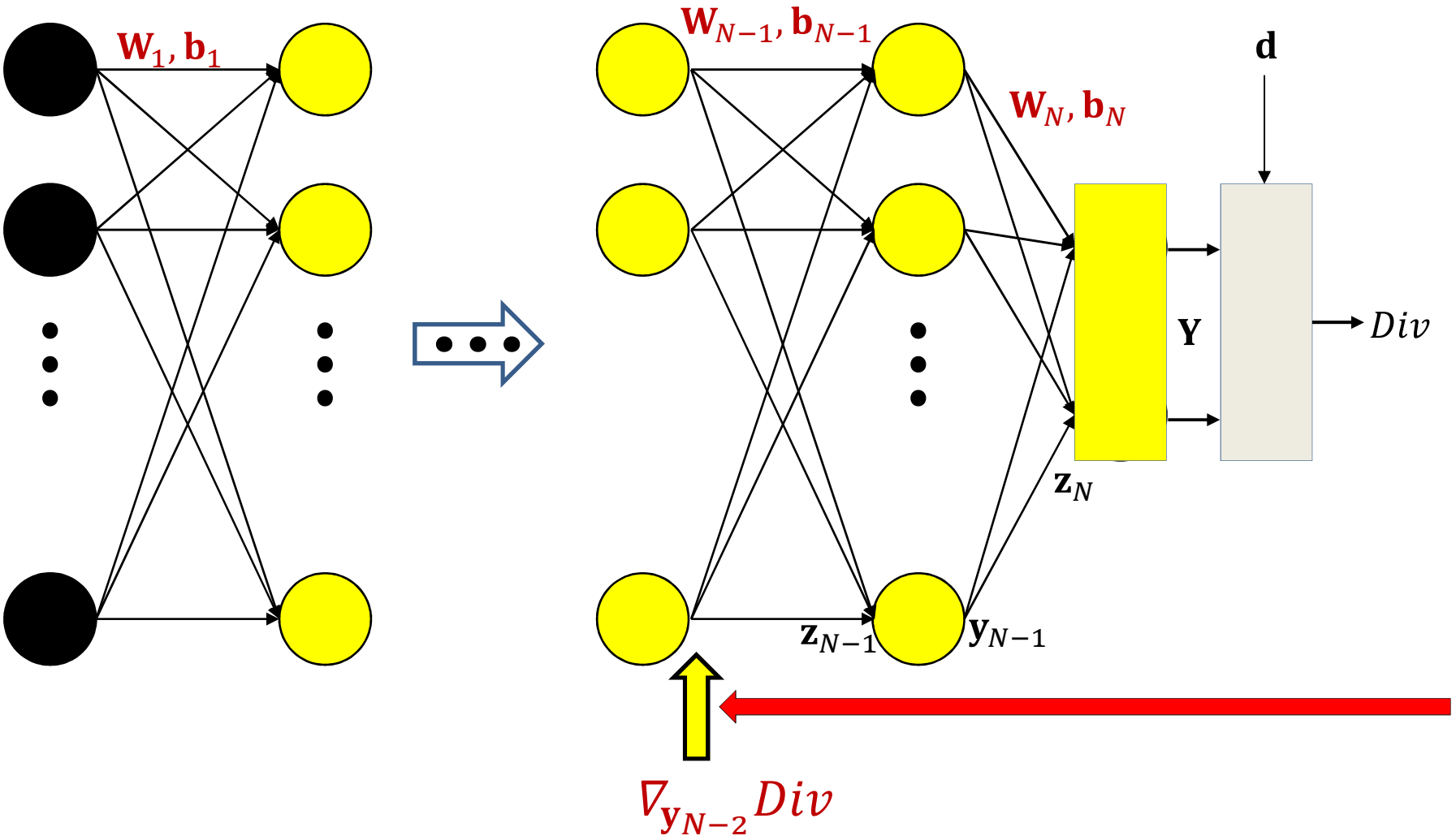


$$\nabla_{\mathbf{z}_{N-1}} Div = \nabla_{\mathbf{y}_{N-1}} Div \, J_{\mathbf{y}_{N-1}}(\mathbf{z}_{N-1})$$

$\nabla_{\mathbf{z}_{N-1}} Div$

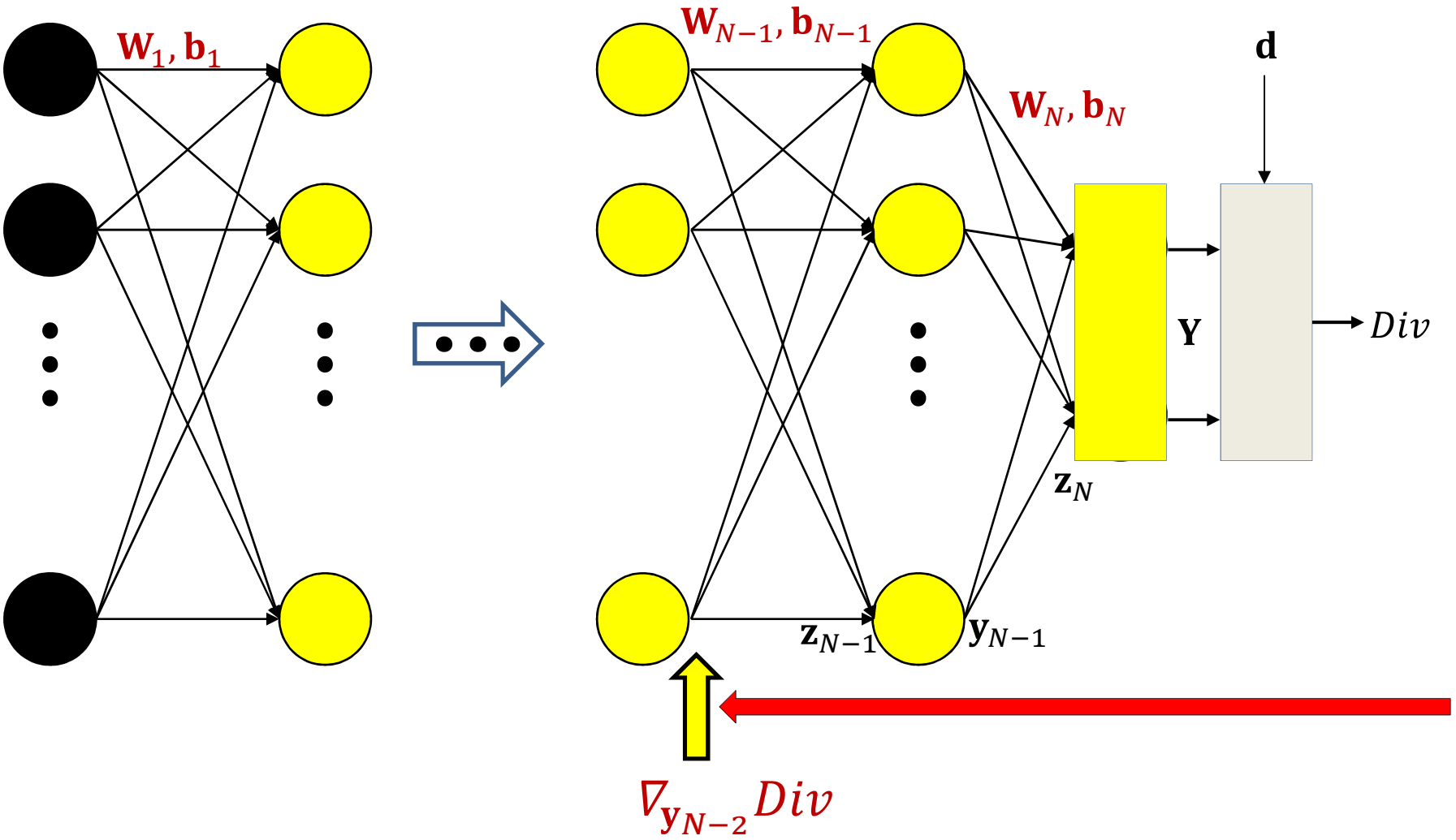
The Jacobian will be a diagonal matrix for scalar activations

The backward pass



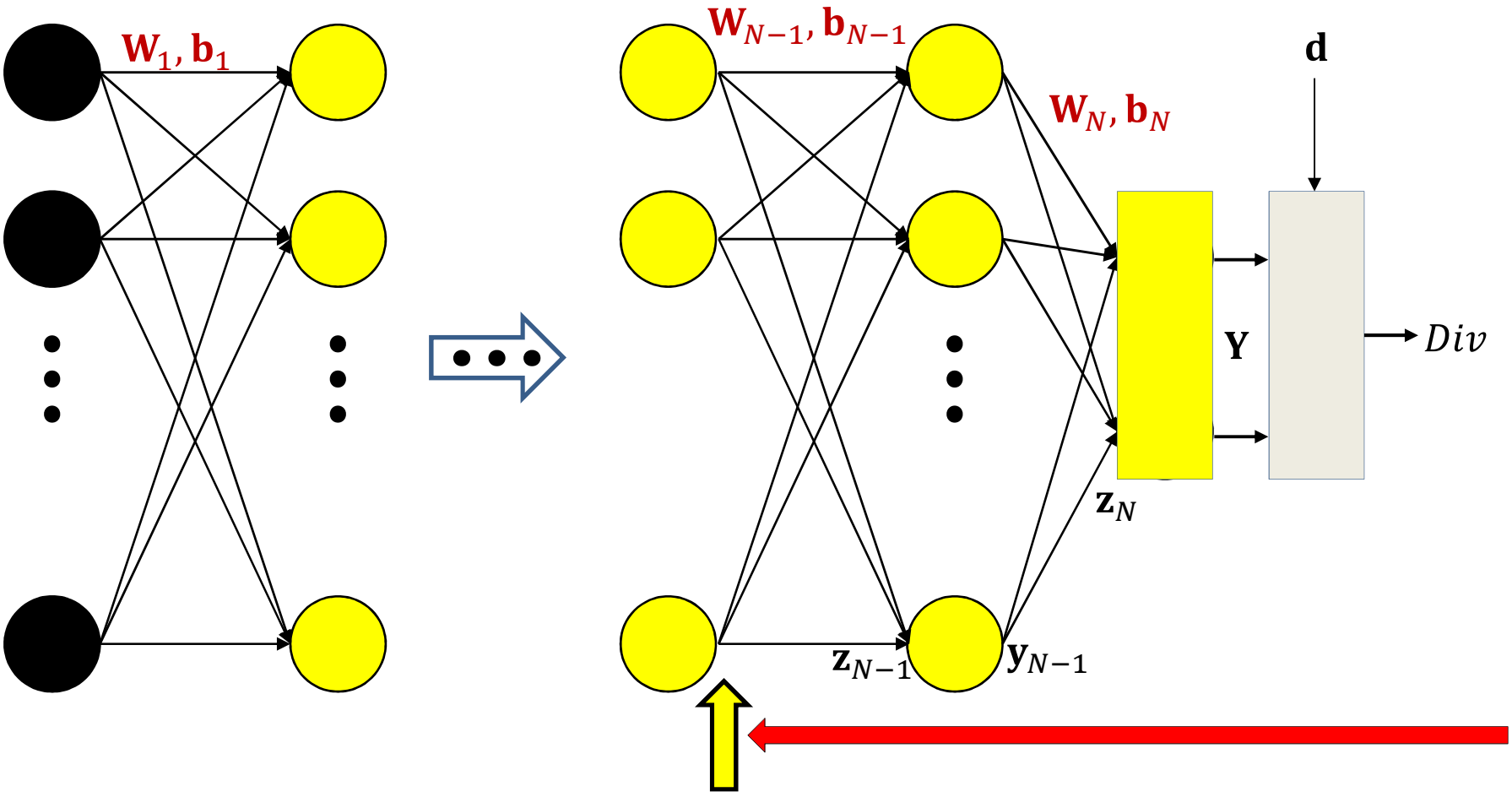
$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div . \nabla_{\mathbf{y}_{N-2}} \mathbf{z}_{N-1}$$

The backward pass



$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div \, \mathbf{W}_{N-1}$$

The backward pass

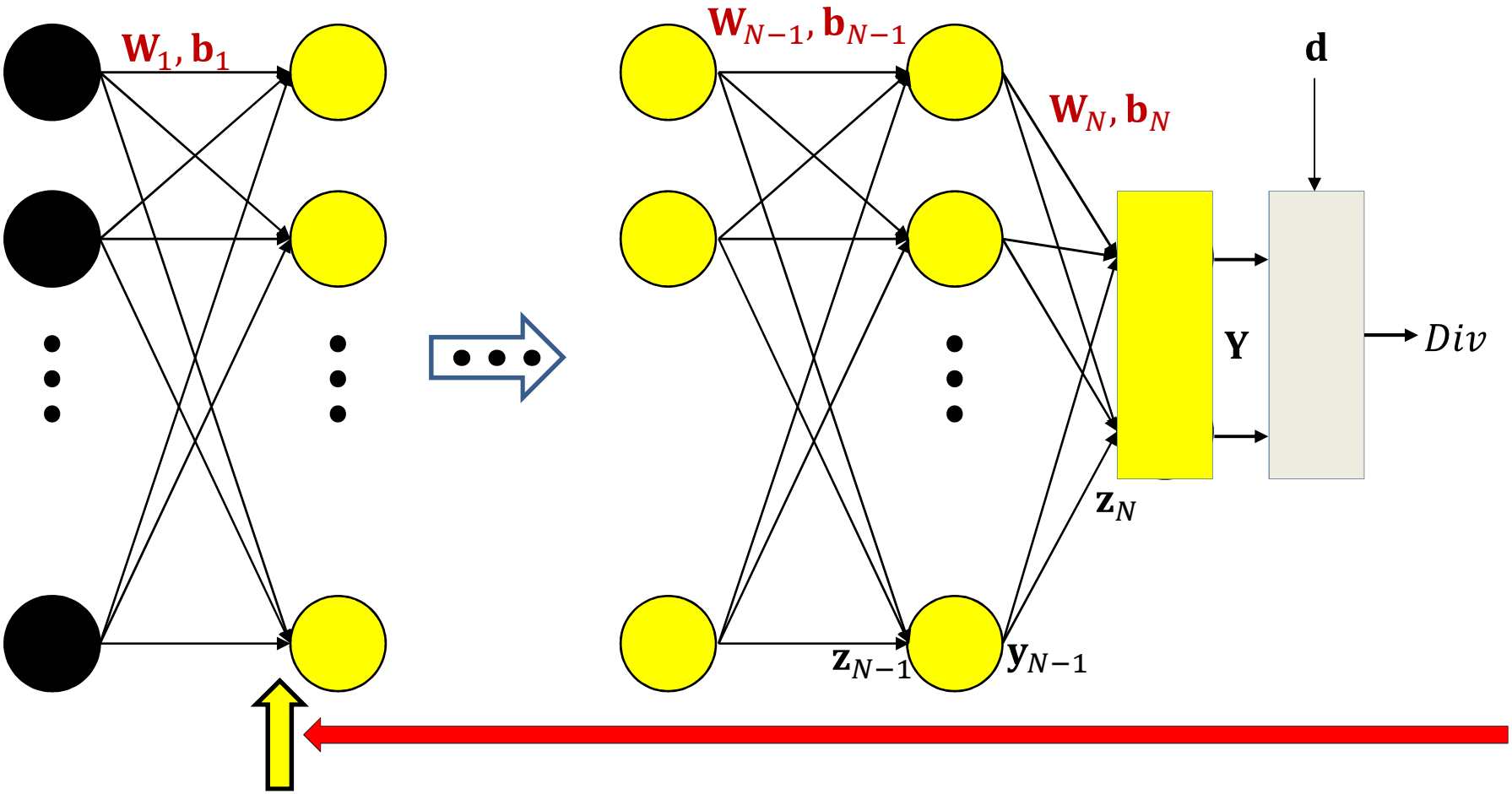


$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div \, \mathbf{W}_{N-1}$$

$$\nabla_{\mathbf{W}_{N-1}} Div = \mathbf{y}_{N-2} \nabla_{\mathbf{z}_{N-1}} Div$$

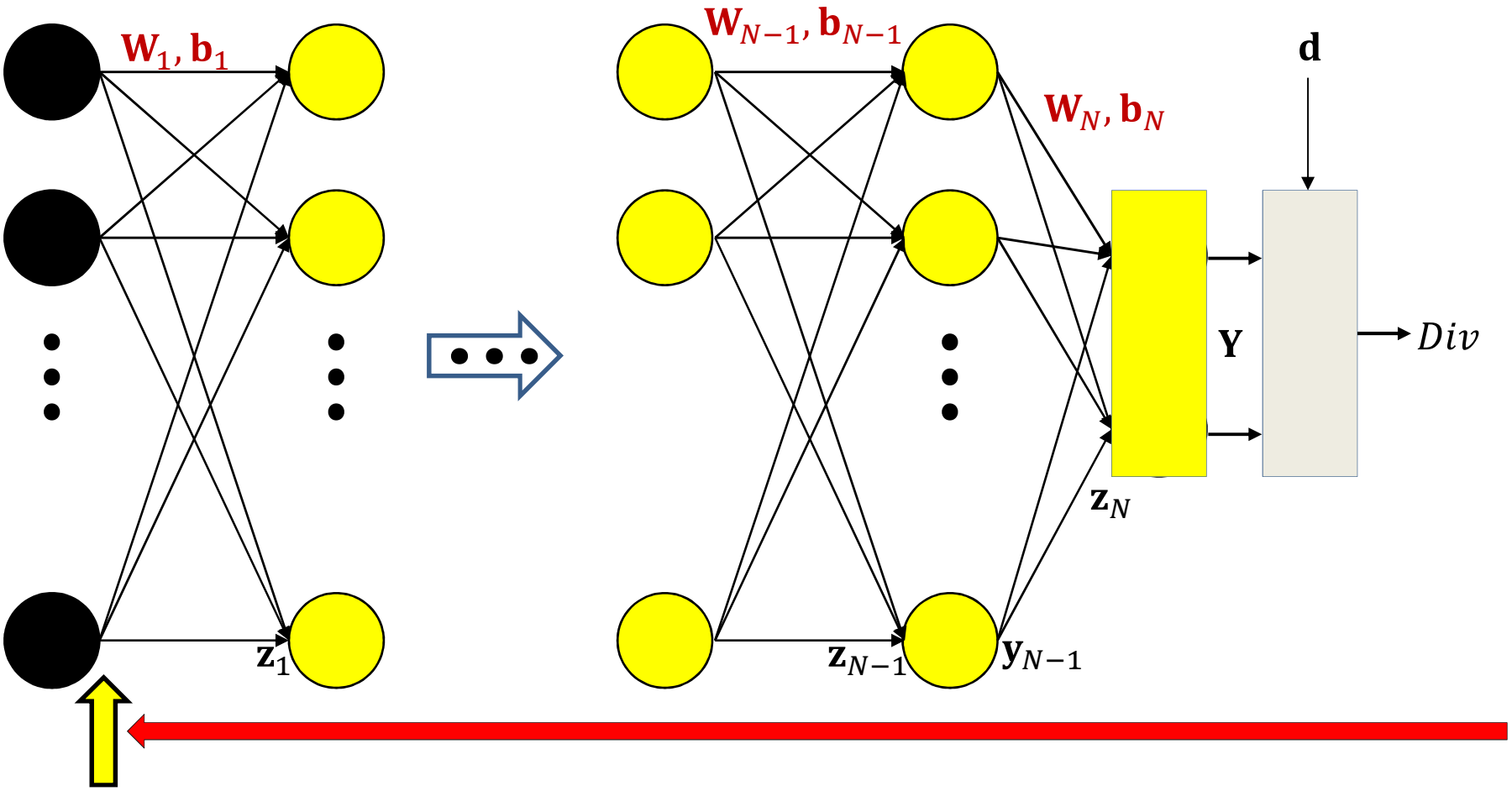
$$\nabla_{\mathbf{b}_{N-1}} Div = \nabla_{\mathbf{z}_{N-1}} Div$$

The backward pass



$$\nabla_{\mathbf{z}_1} Div = \nabla_{\mathbf{y}_1} Div \, J_{\mathbf{y}_1}(\mathbf{z}_1)$$

The backward pass



$$\nabla_{\mathbf{W}_1} Div = \mathbf{x} \nabla_{\mathbf{z}_1} Div$$

$$\nabla_{\mathbf{b}_1} Div = \nabla_{\mathbf{z}_1} Div$$

In some problems we will also want to compute the derivative w.r.t. the input

The Backward Pass

- Set $\mathbf{y}_N = Y$, $\mathbf{y}_0 = \mathbf{x}$
- Initialize: Compute $\nabla_{\mathbf{y}_N} Div = \nabla_Y Div$

- For layer k = N downto 1:
  - Compute $J_{\mathbf{y}_k}(\mathbf{z}_k)$
    - Will require intermediate values computed in the forward pass
  - Recursion:

$$\nabla_{\mathbf{z}_k} Div = \nabla_{\mathbf{y}_k} Div \, J_{\mathbf{y}_k}(\mathbf{z}_k)$$
$$\nabla_{\mathbf{y}_{k-1}} Div = \nabla_{\mathbf{z}_k} Div \, \mathbf{W}_k$$

  - Gradient computation:

$$\nabla_{\mathbf{W}_k} Div = \mathbf{y}_{k-1} \nabla_{\mathbf{z}_k} Div$$
$$\nabla_{\mathbf{b}_k} Div = \nabla_{\mathbf{z}_k} Div$$

The Backward Pass

- Set $\mathbf{y}_N = Y$, $\mathbf{y}_0 = \mathbf{x}$
- Initialize: Compute $\nabla_{\mathbf{y}_N} Div = \nabla_Y Div$

- For layer k = N downto 1:
  - Compute $J_{\mathbf{y}_k}(\mathbf{z}_k)$
    - Will require intermediate values computed in the forward pass
  - Recursion:

Note analogy to forward pass

$$\nabla_{\mathbf{z}_k} Div = \nabla_{\mathbf{y}_k} Div \, J_{\mathbf{y}_k}(\mathbf{z}_k)$$
$$\nabla_{\mathbf{y}_{k-1}} Div = \nabla_{\mathbf{z}_k} Div \, \mathbf{W}_k$$

  - Gradient computation:

$$\nabla_{\mathbf{W}_k} Div = \mathbf{y}_{k-1} \nabla_{\mathbf{z}_k} Div$$
$$\nabla_{\mathbf{b}_k} Div = \nabla_{\mathbf{z}_k} Div$$

For comparison: The Forward Pass

- Set $\mathbf{y}_0 = \mathbf{x}$

- For layer k = 1 to N:
  - Recursion:

$$\mathbf{z}_k = \mathbf{W}_k \mathbf{y}_{k-1} + \mathbf{b}_k$$
$$\mathbf{y}_k = \boldsymbol{f}_k(\mathbf{z}_k)$$

- Output:

$$\mathbf{Y} = \mathbf{y}_N$$

Neural network training algorithm

- Initialize all weights and biases $(\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \ldots, \mathbf{W}_N, \mathbf{b}_N)$
- Do:
  - $Err = 0$
  - For all k, initialize $\nabla_{\mathbf{W}_k} Err = 0$, $\nabla_{\mathbf{b}_k} Err = 0$
  - For all $t = 1:T$
    - Forward pass : Compute
      - Output $\boldsymbol{Y}(\boldsymbol{X}_t)$
      - Divergence $\boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$
      - $Err \mathrel{+}= \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$
    - Backward pass: For all k compute:
      - $\nabla_{\mathbf{y}_k} Div = \nabla_{\mathbf{z}_{k+1}} Div \, \mathbf{W}_k$
      - $\nabla_{\mathbf{z}_k} Div = \nabla_{\mathbf{y}_k} Div \, J_{\mathbf{y}_k}(\mathbf{z}_k)$
      - $\nabla_{\mathbf{W}_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$; $\nabla_{\mathbf{b}_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$
      - $\nabla_{\mathbf{W}_k} Err \mathrel{+}= \nabla_{\mathbf{W}_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$; $\nabla_{\mathbf{b}_k} Err \mathrel{+}= \nabla_{\mathbf{b}_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$
  - For all k, update:

$$\mathbf{W}_k = \mathbf{W}_k - \frac{\eta}{T}\left(\nabla_{\mathbf{W}_k} Err\right)^T; \qquad \mathbf{b}_k = \mathbf{b}_k - \frac{\eta}{T}\left(\nabla_{\mathbf{W}_k} Err\right)^T$$

- Until Err has converged