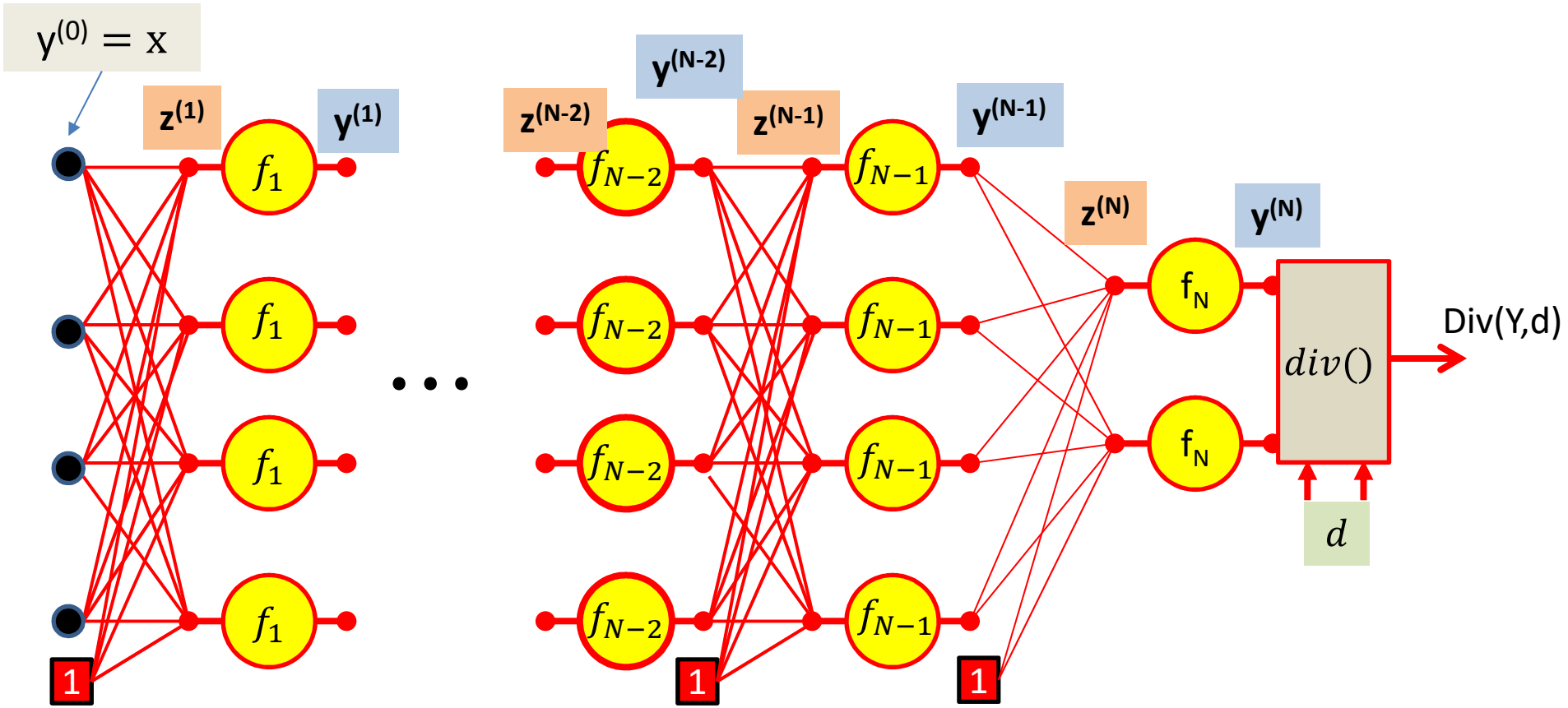
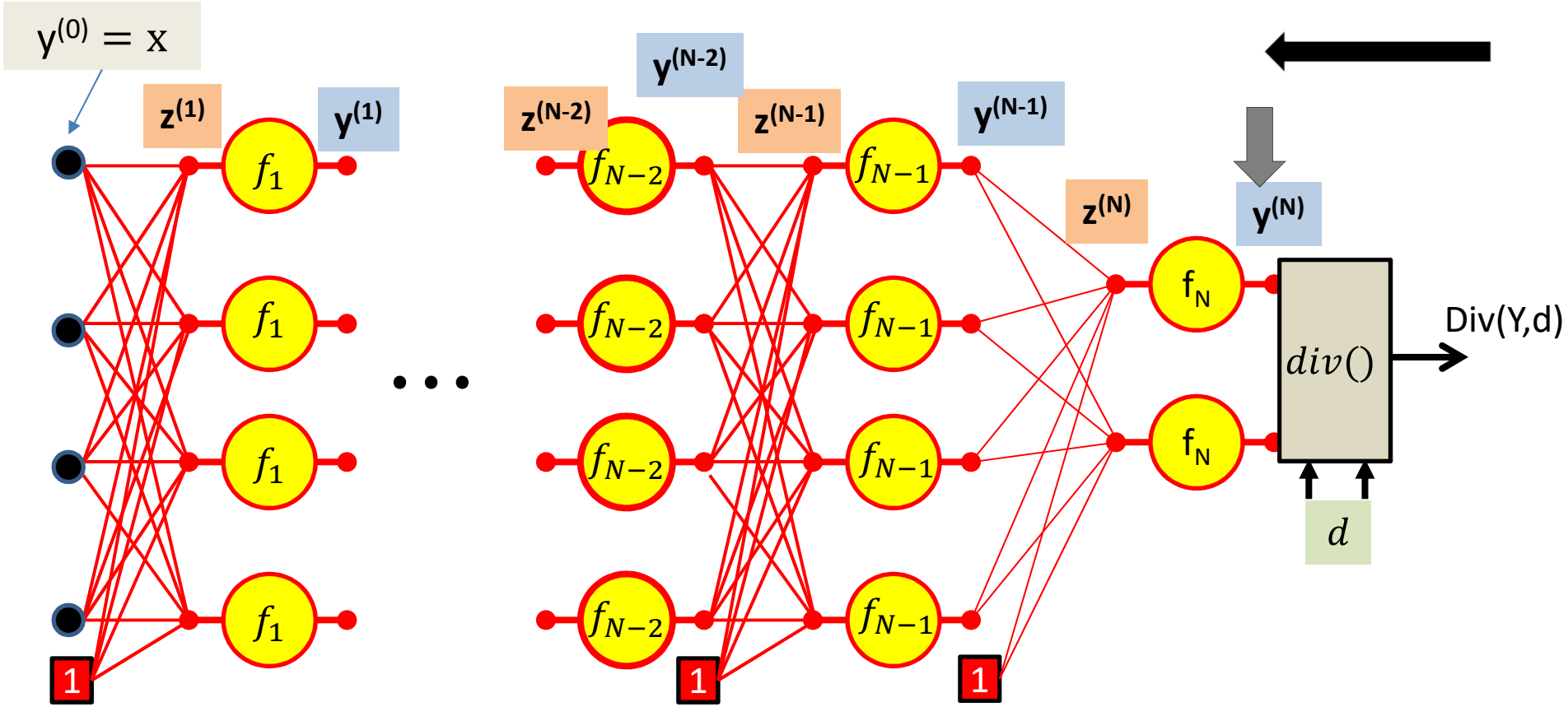
Forward "Pass"

- Input: D dimensional vector $\mathbf{x} = [x_j,\ j = 1 \ldots D]$
- Set:
  - $D_0 = D,$ is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j,\ j = 1 \ldots D; \quad y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$ D_k is the size of the kth layer
    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k\left(z_j^{(k)}\right)$
- Output:
  - $Y = y_j^{(N)}, j = 1..D_N$

Computing derivatives



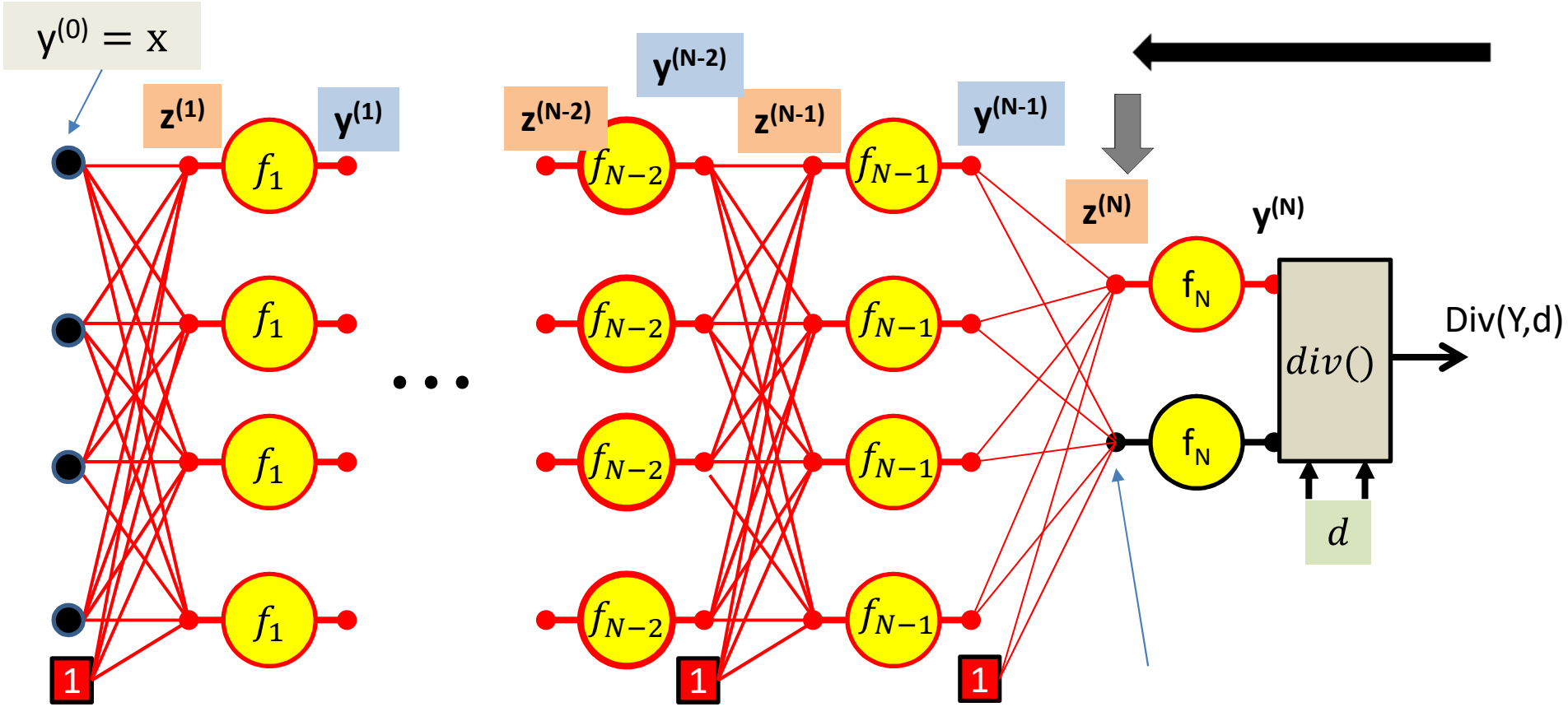
Computing derivatives



The derivative w.r.t the actual output of the network is simply the derivative w.r.t to the output of the final layer of the network

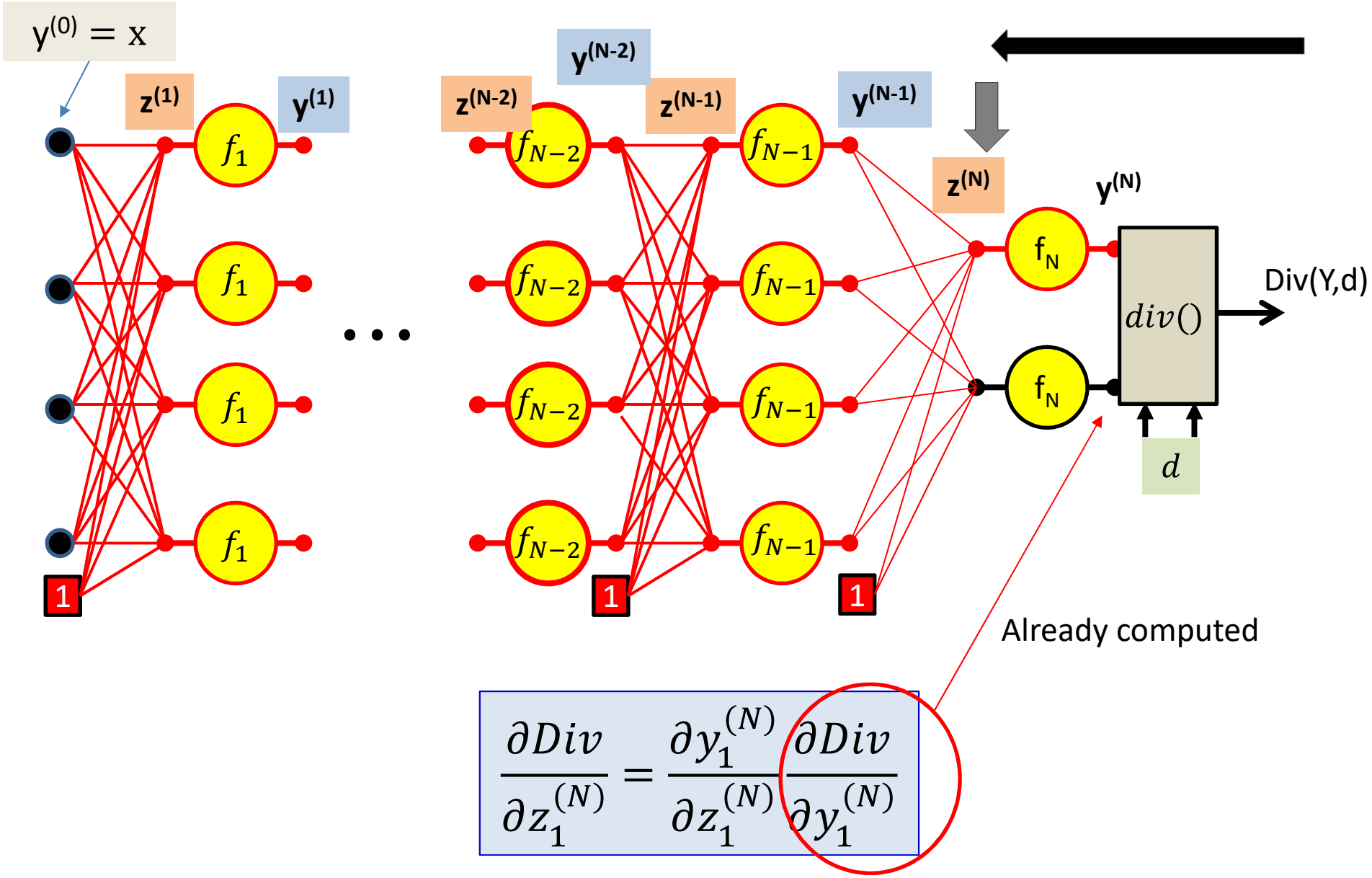
$$\frac{\partial Div(Y,d)}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$$

Computing derivatives

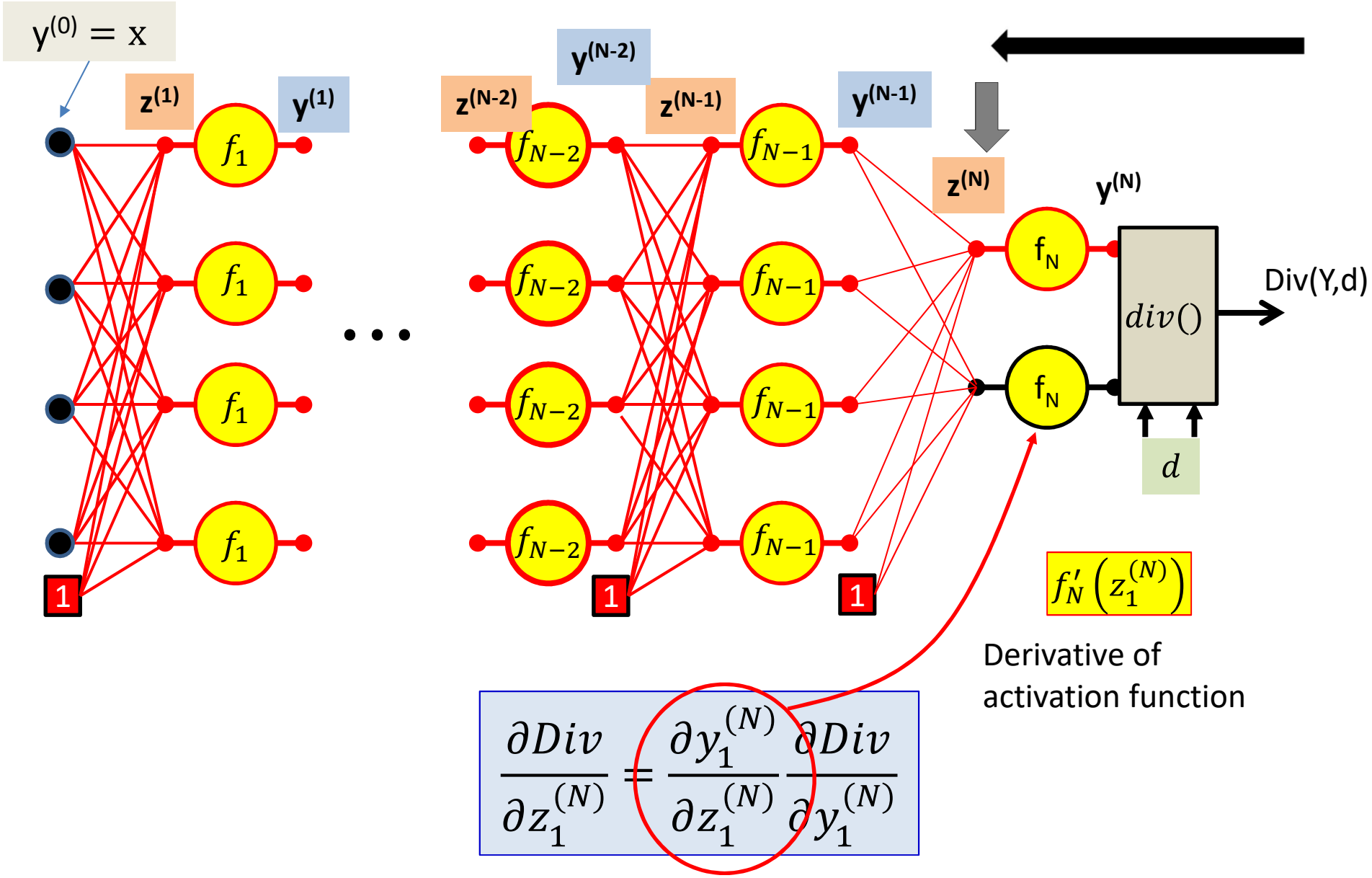


$$\frac{\partial Div}{\partial z_1^{(N)}} = \frac{\partial y_1^{(N)}}{\partial z_1^{(N)}} \frac{\partial Div}{\partial y_1^{(N)}}$$

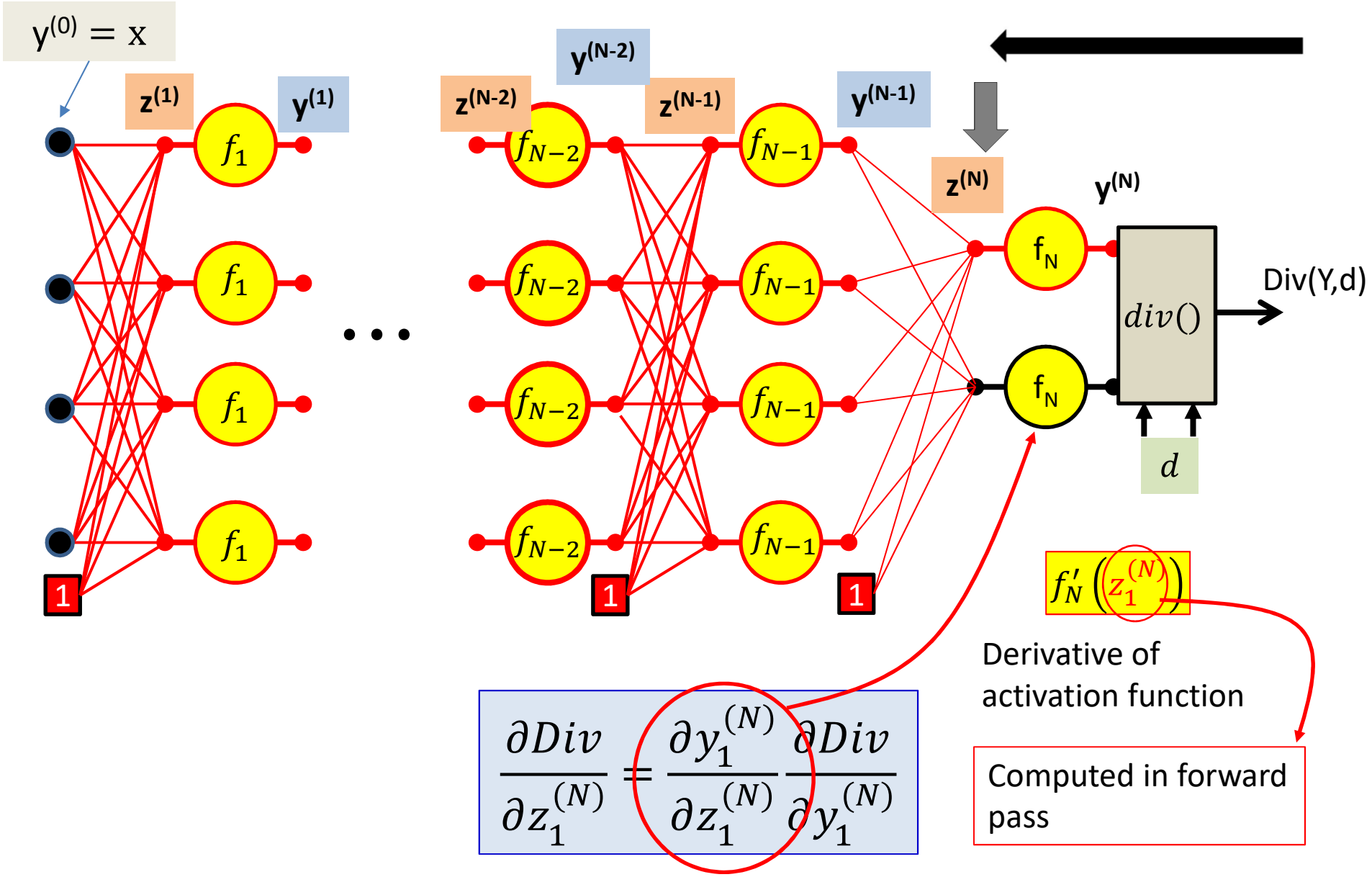
Computing derivatives



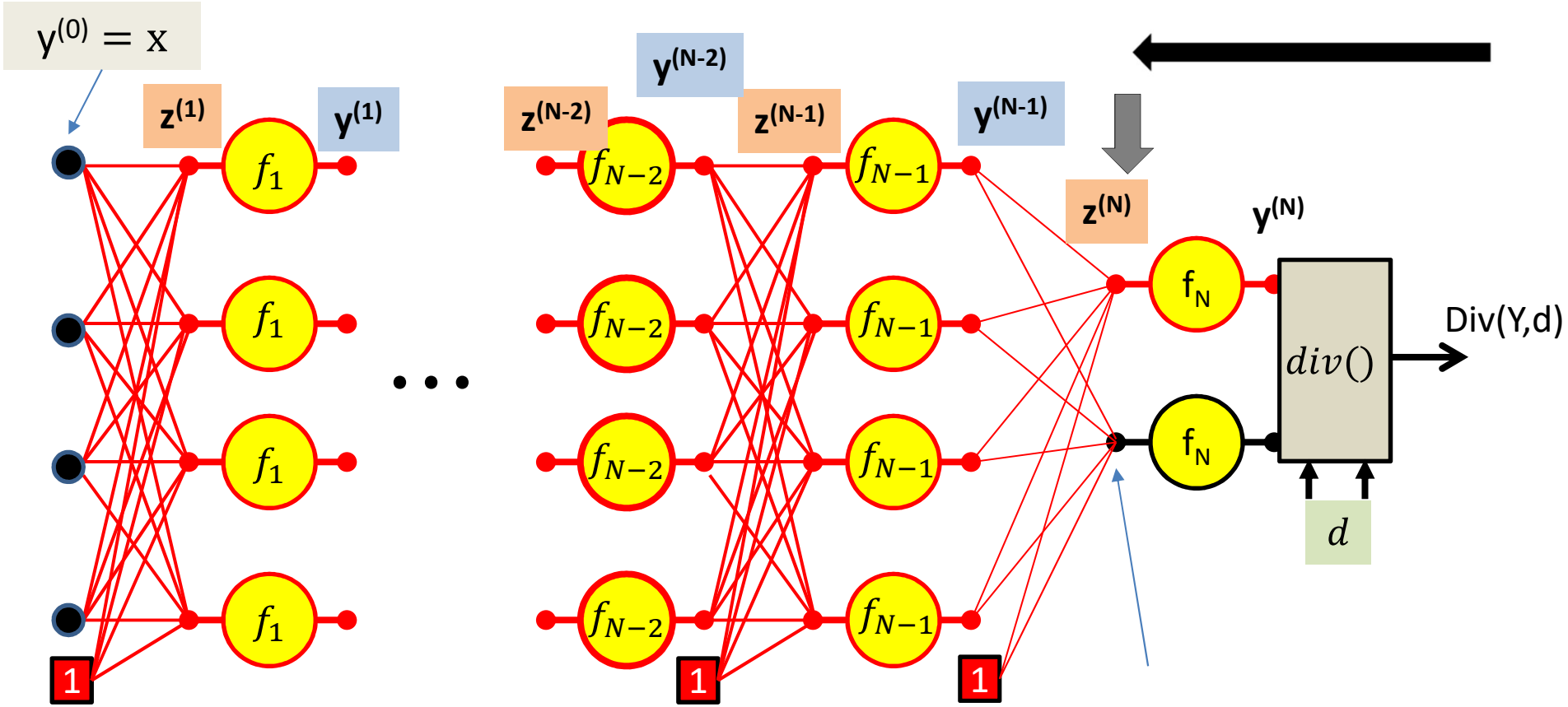
Computing derivatives



Computing derivatives

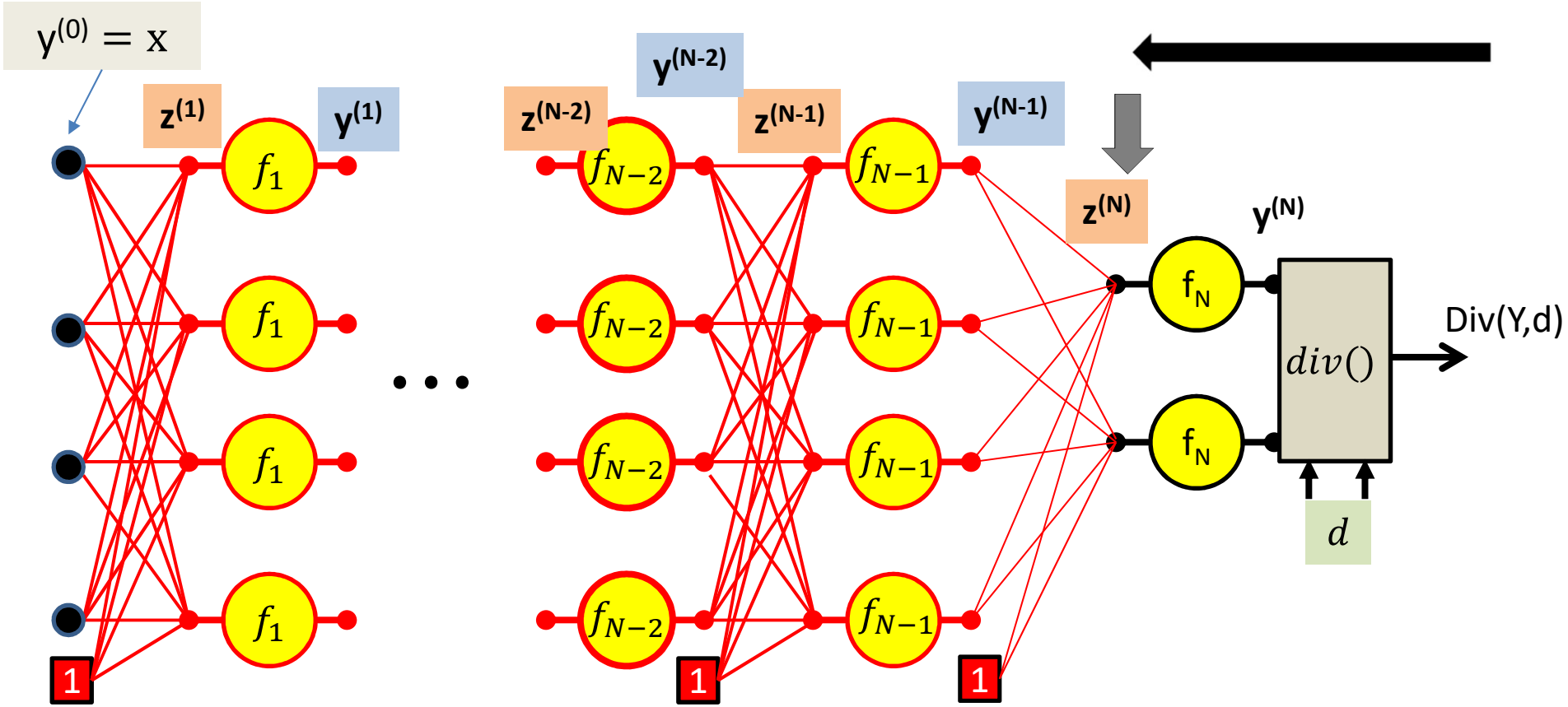


Computing derivatives



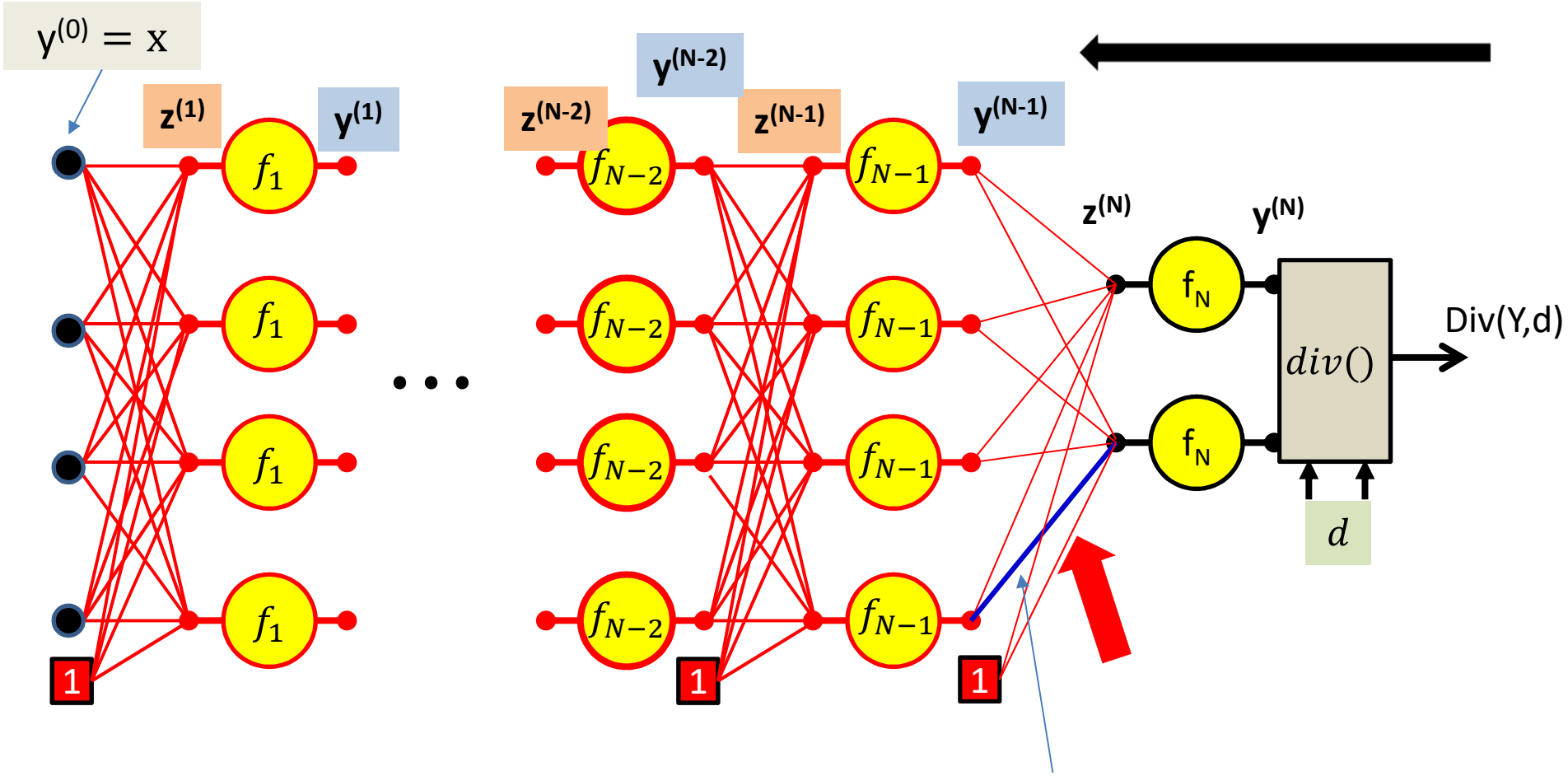
$$\frac{\partial Div}{\partial z_1^{(N)}} = f_N'\left(z_1^{(N)}\right)\frac{\partial Div}{\partial y_1^{(N)}}$$

Computing derivatives



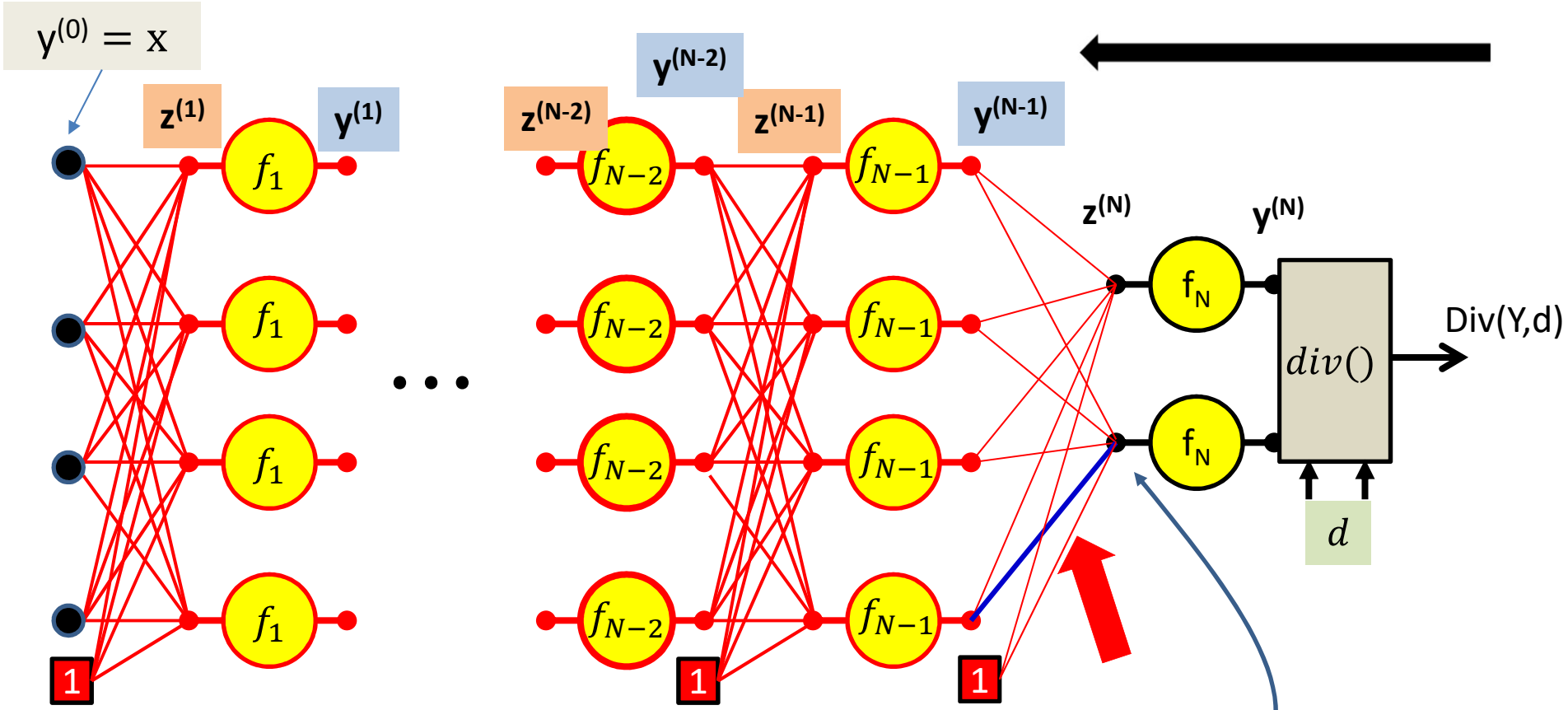
$$\frac{\partial Div}{\partial z_i^{(N)}} = f_N'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

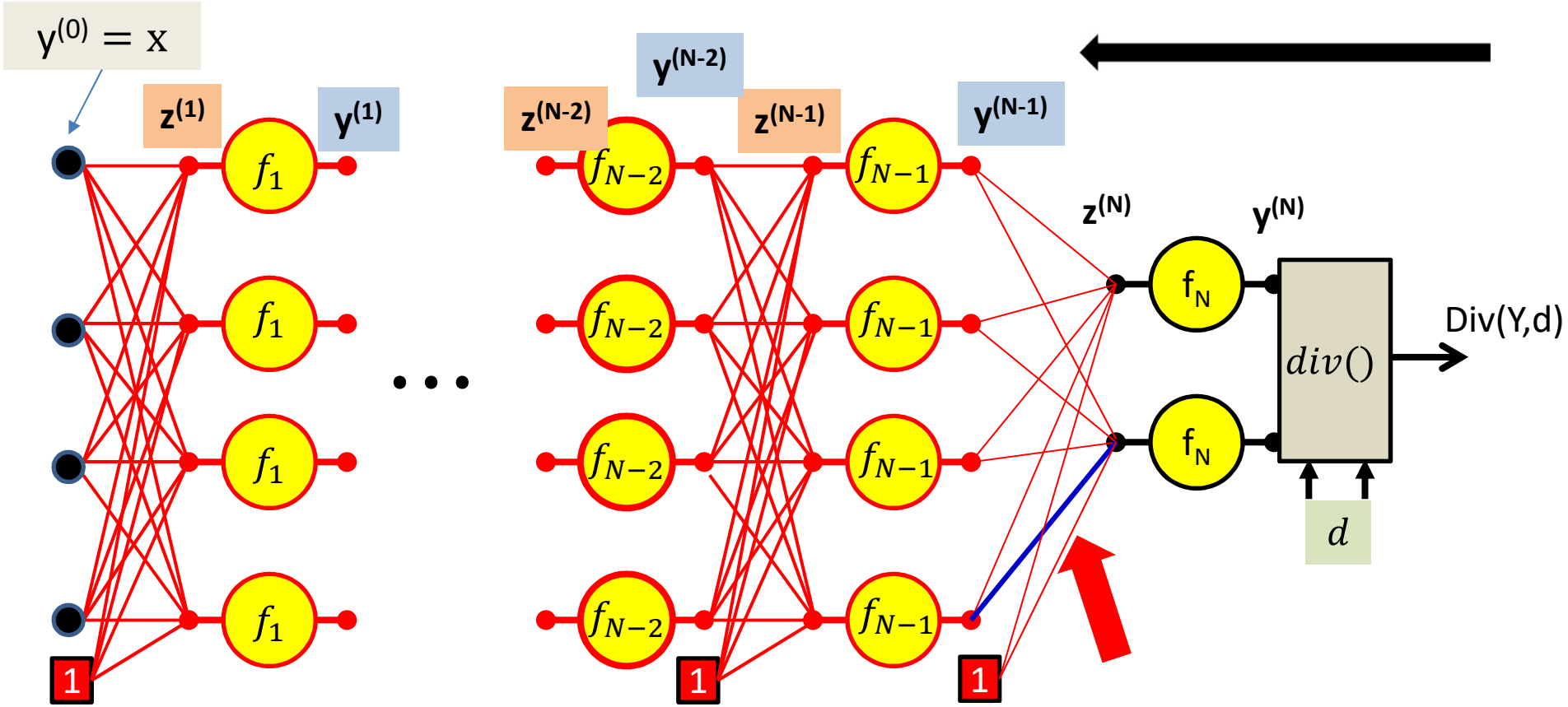
Computing derivatives



$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

Just computed

Computing derivatives



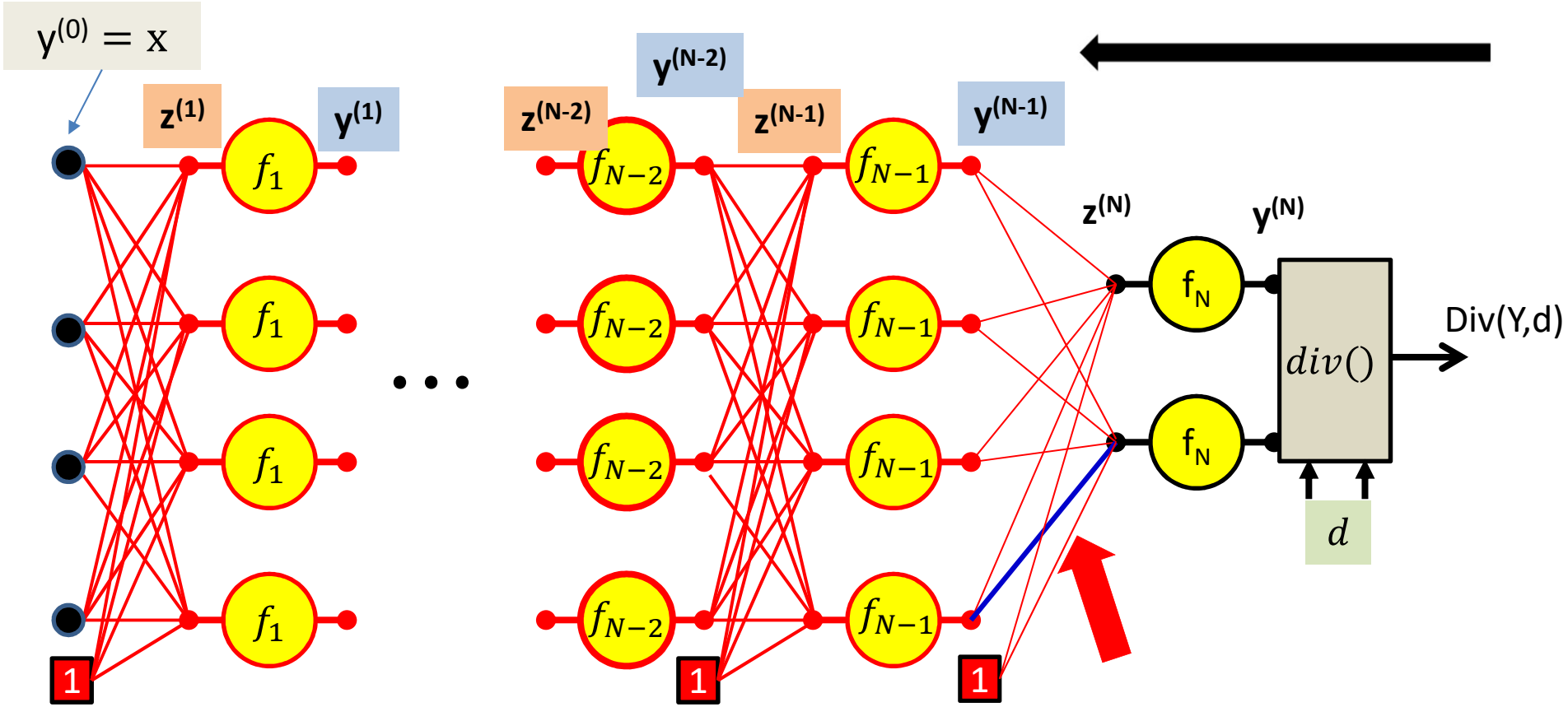
$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

$y_1^{(N-1)}$

Because

$$z_1^{(N)} = w_{11}^{(N)} y_1^{(N-1)} + \text{other terms}$$

Computing derivatives



$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

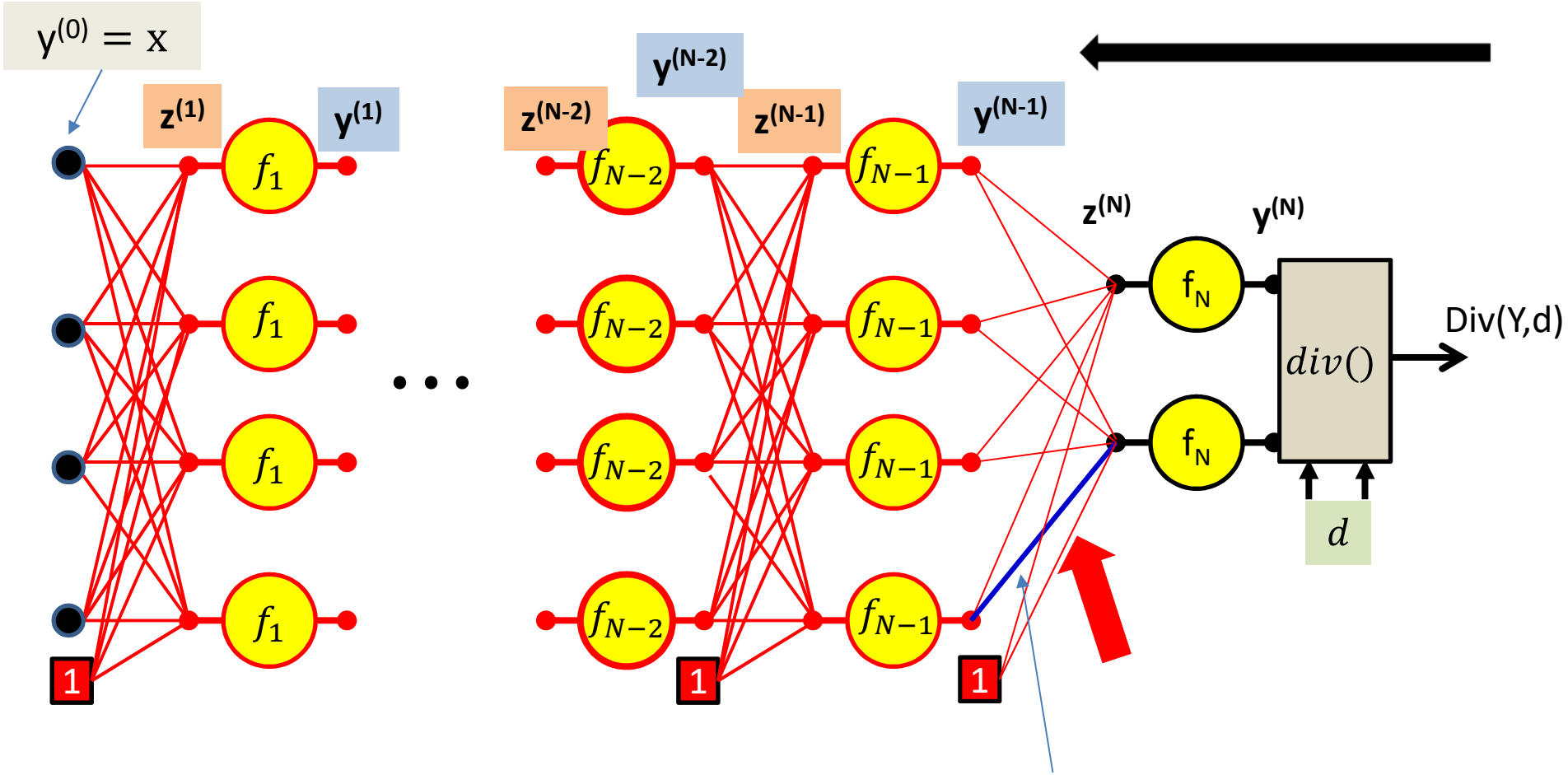
$y_1^{(N-1)}$

Because

$$z_1^{(N)} = w_{11}^{(N)} y_1^{(N-1)} + \text{other terms}$$

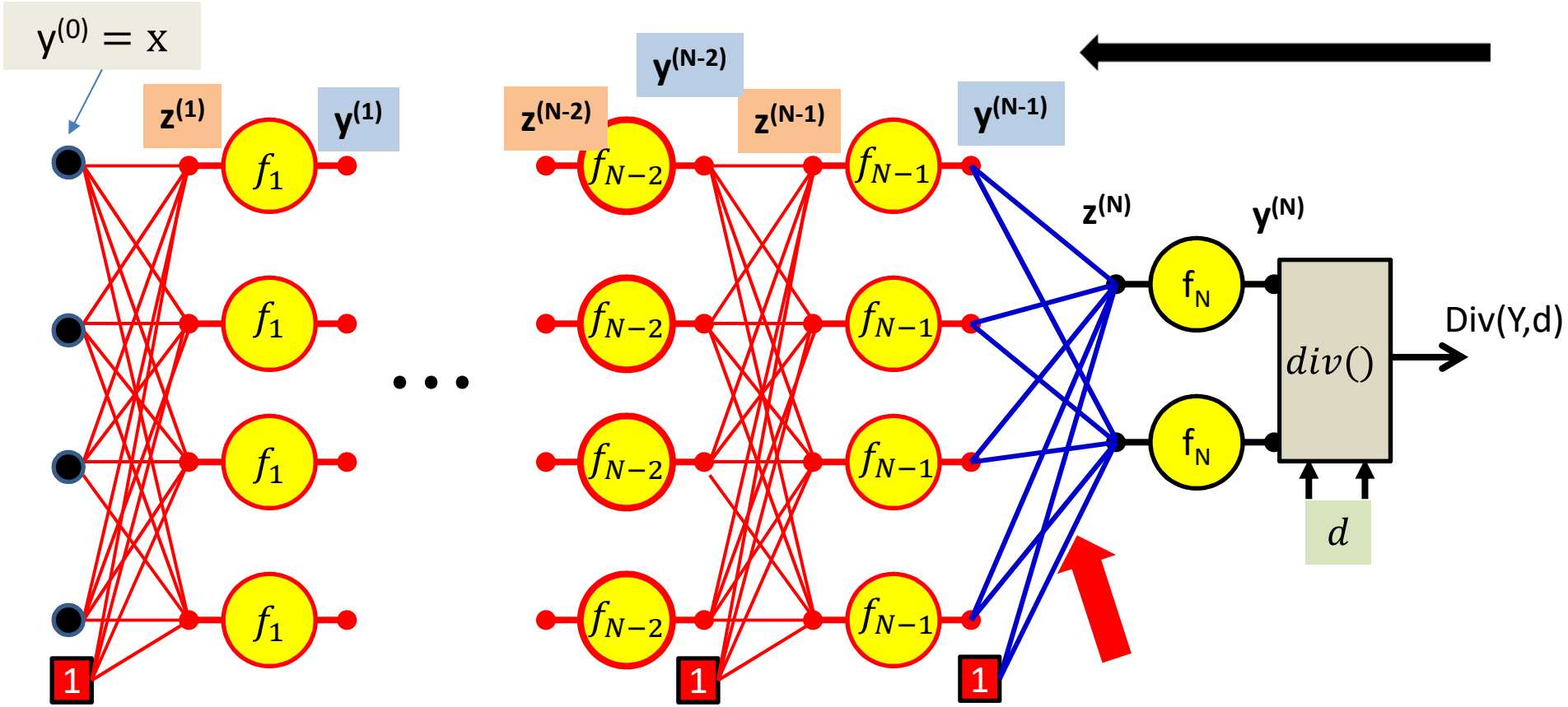
Computed in forward pass

Computing derivatives



$$\frac{\partial Div}{\partial w_{11}^{(N)}} = y_1^{(N-1)} \frac{\partial Div}{\partial z_1^{(N)}}$$

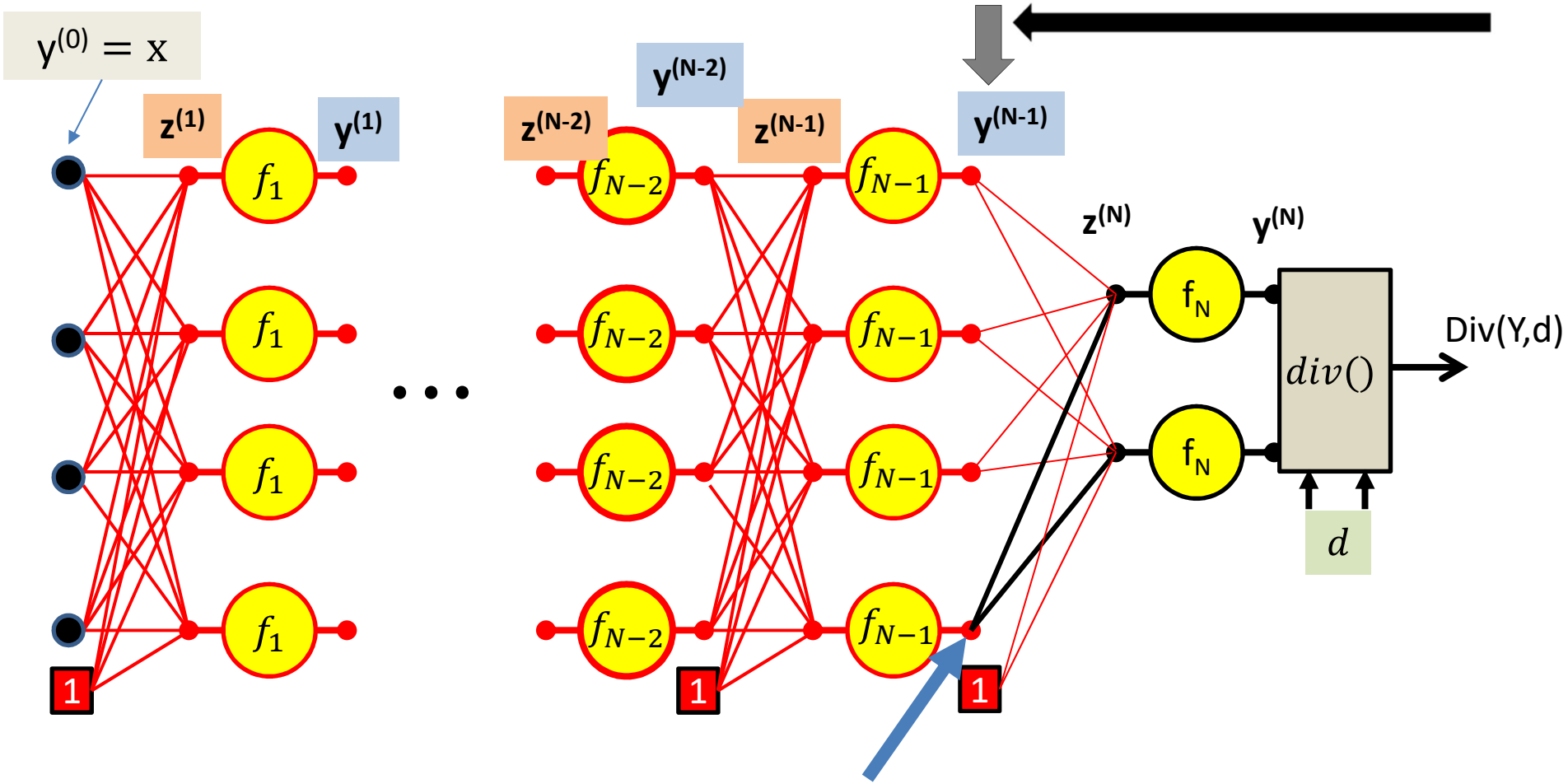
Computing derivatives



$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$$

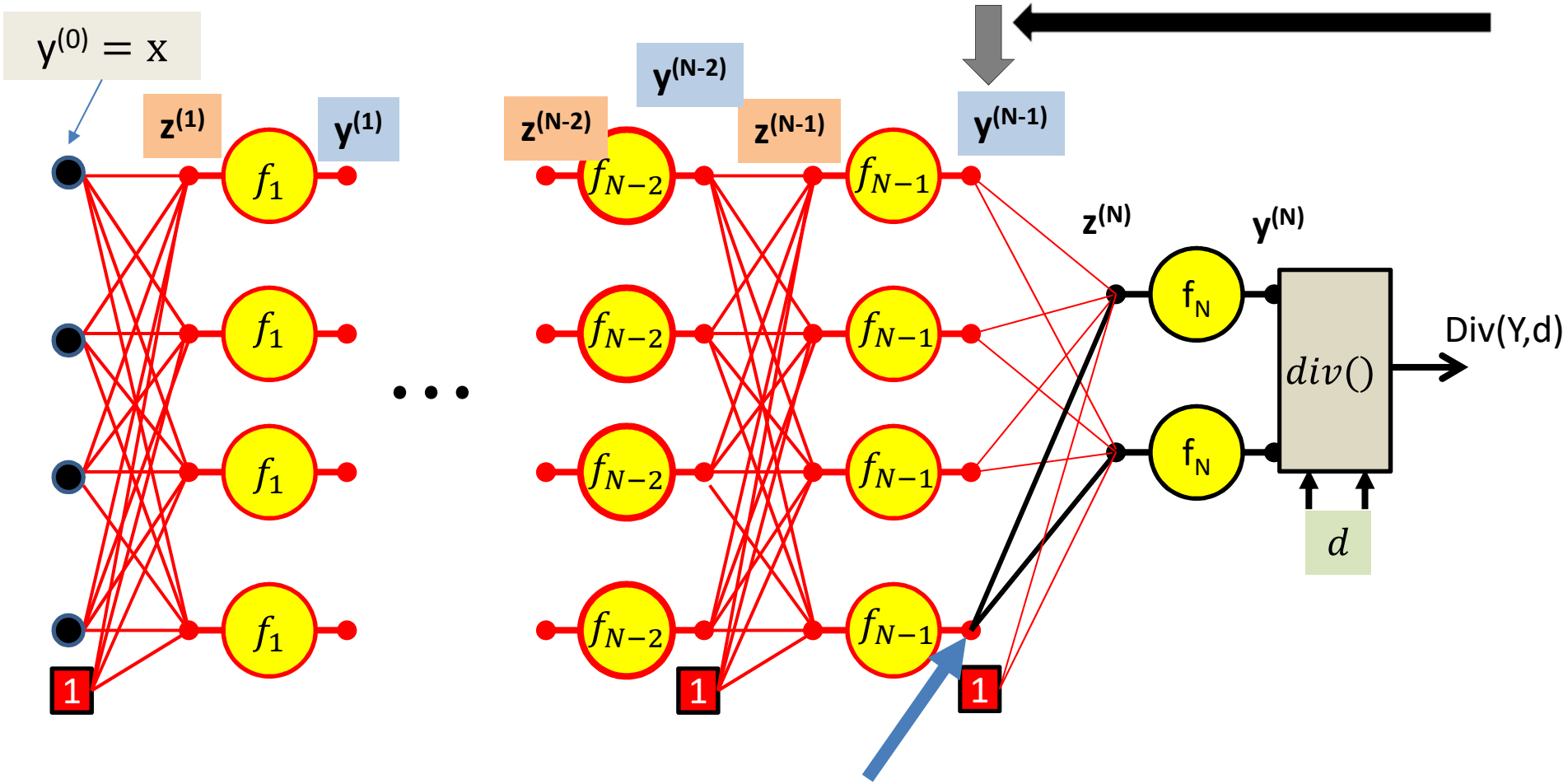
For the bias term $y_0^{(N-1)} = 1$

Computing derivatives



$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

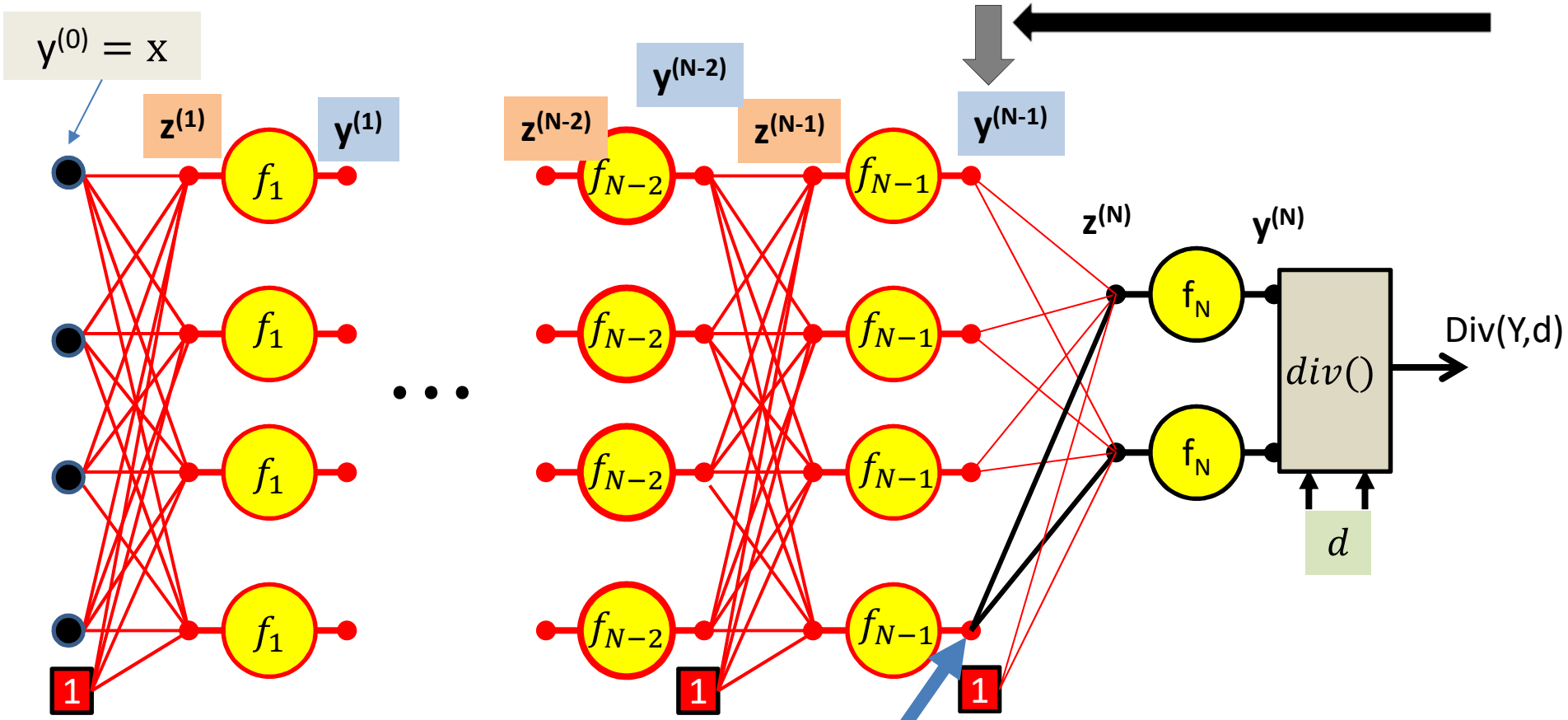
Computing derivatives



$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

Already computed

Computing derivatives



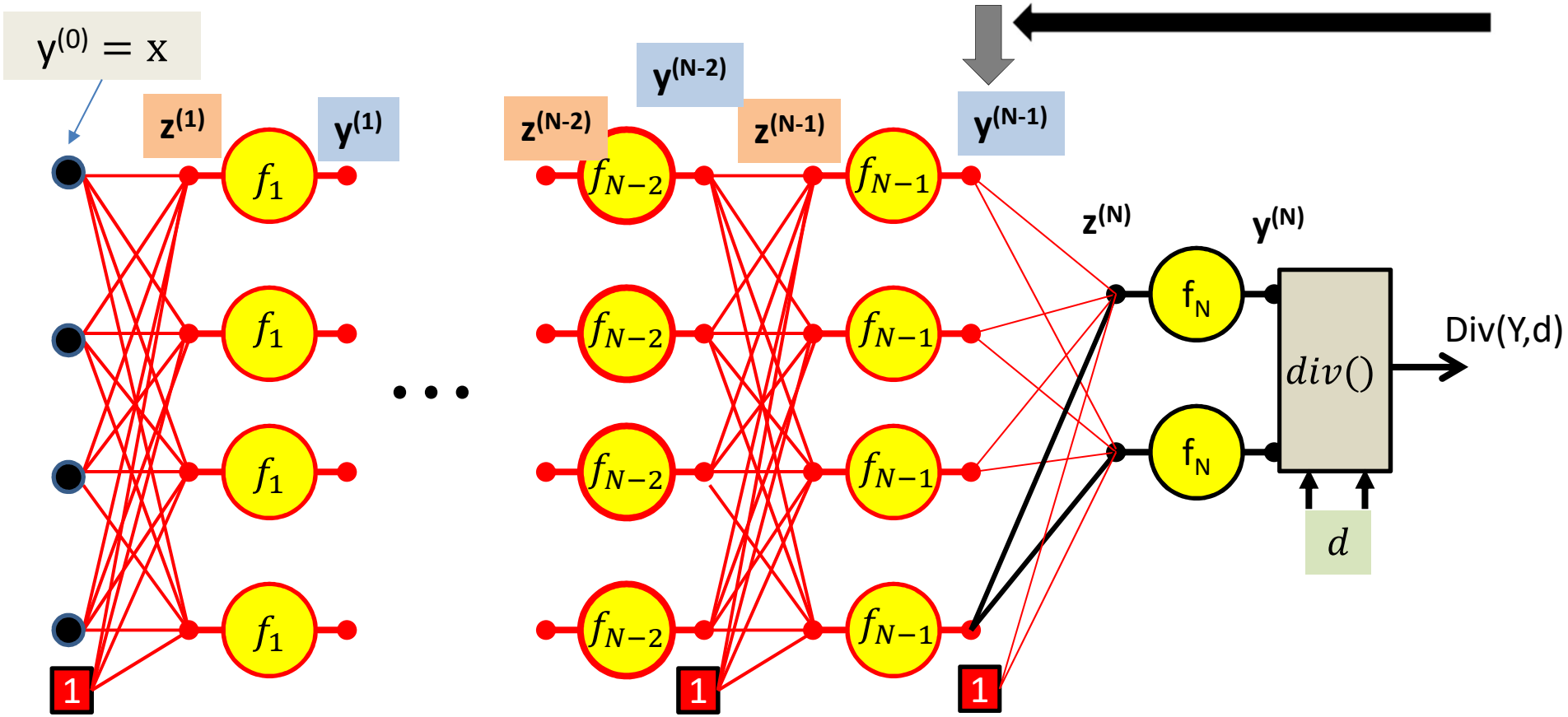
$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$w_{1j}^{(N-1)}$

Because

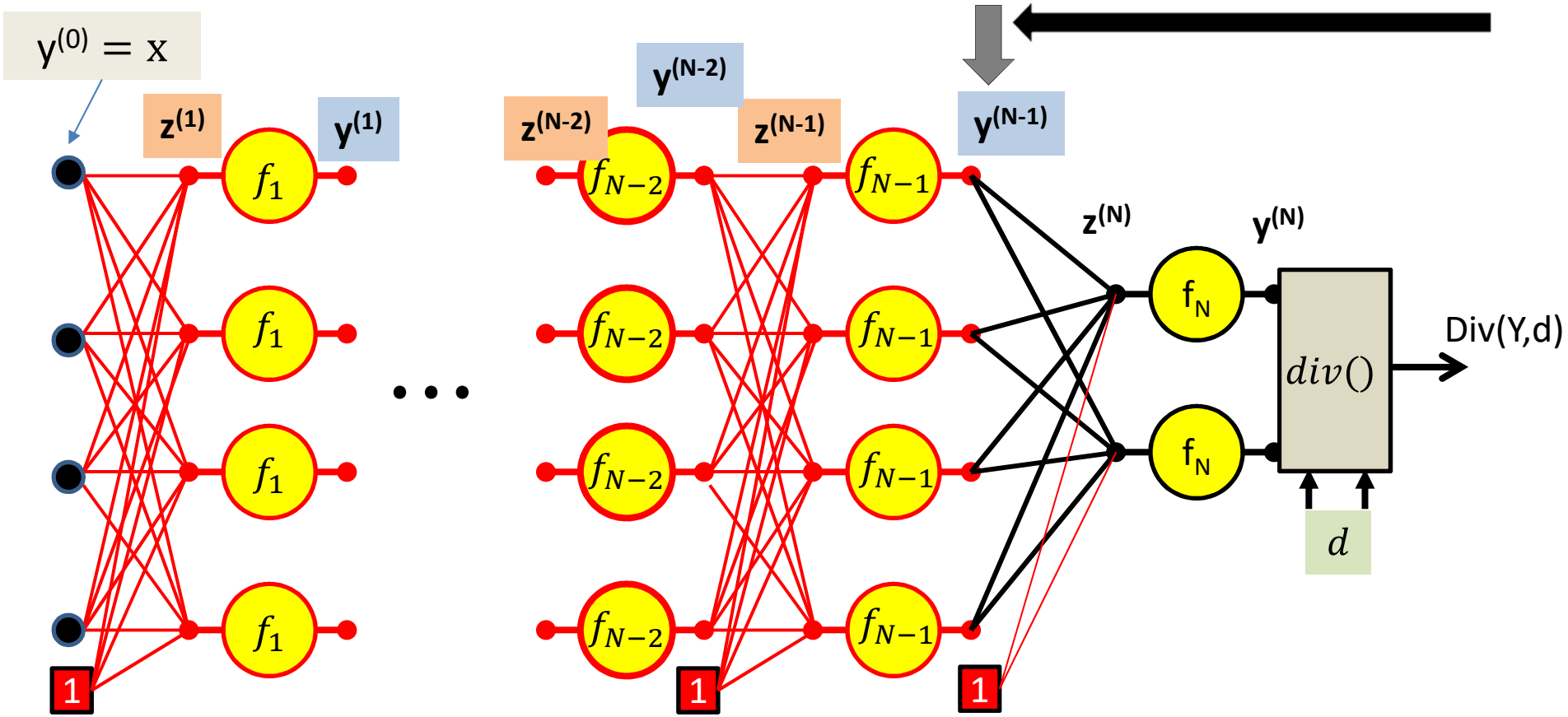
$$z_j^{(N)} = w_{1j}^{(N)} y_1^{(N-1)} + \text{other terms}$$

Computing derivatives



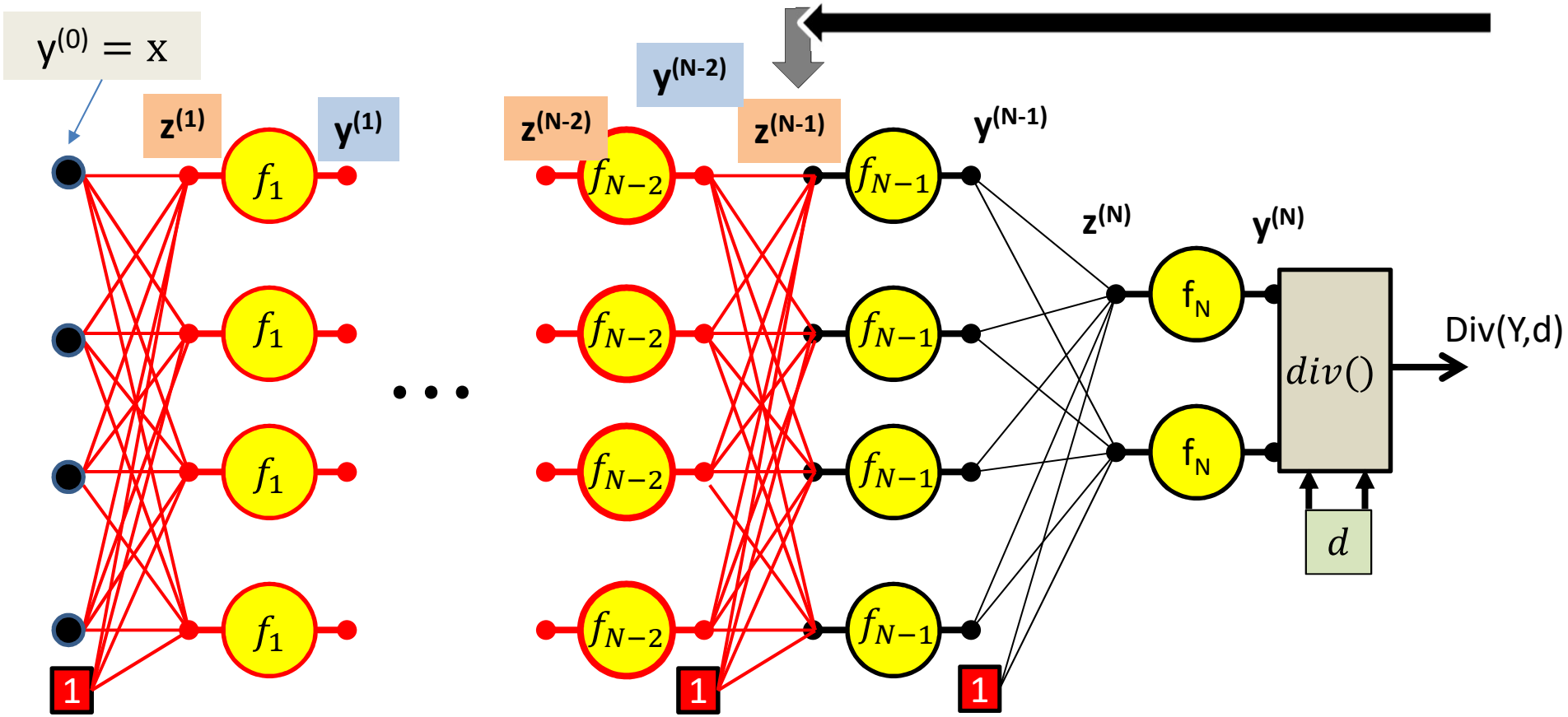
$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j w_{1j}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



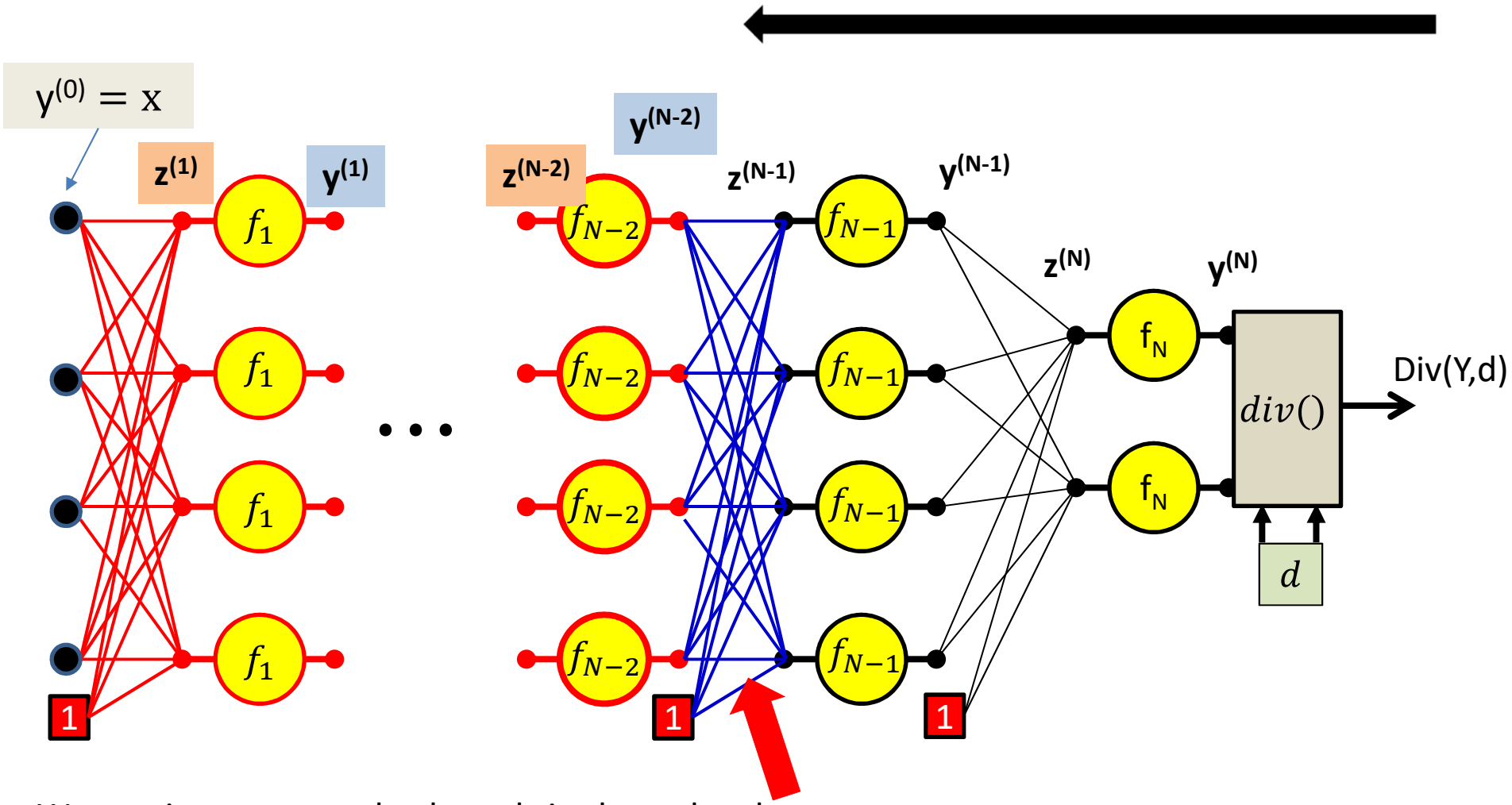
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



We continue our way backwards in the order shown

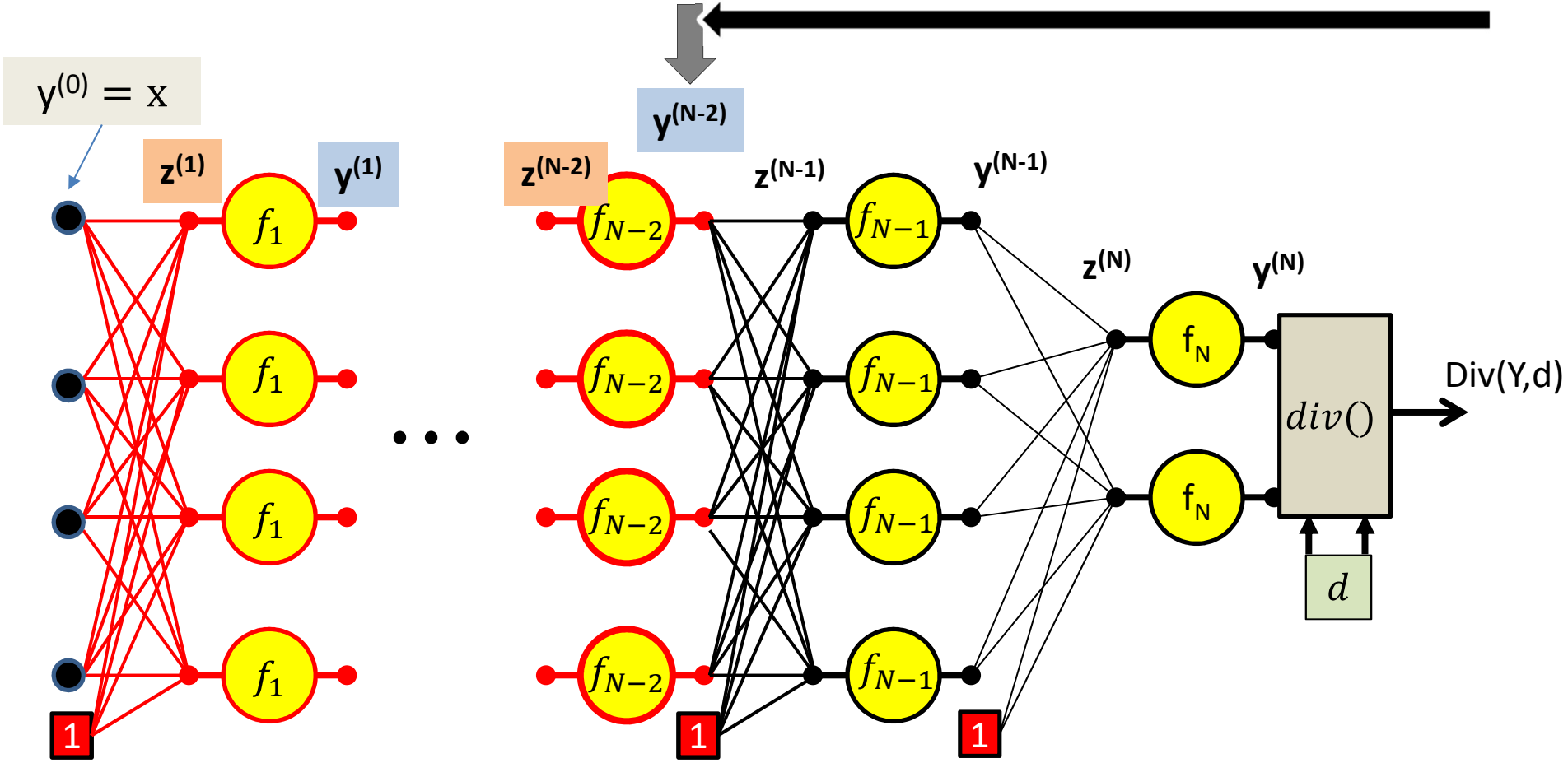
$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f'_{N-1}\left(z_i^{(N-1)}\right)\frac{\partial Div}{\partial y_i^{(N-1)}}$$



We continue our way backwards in the order shown

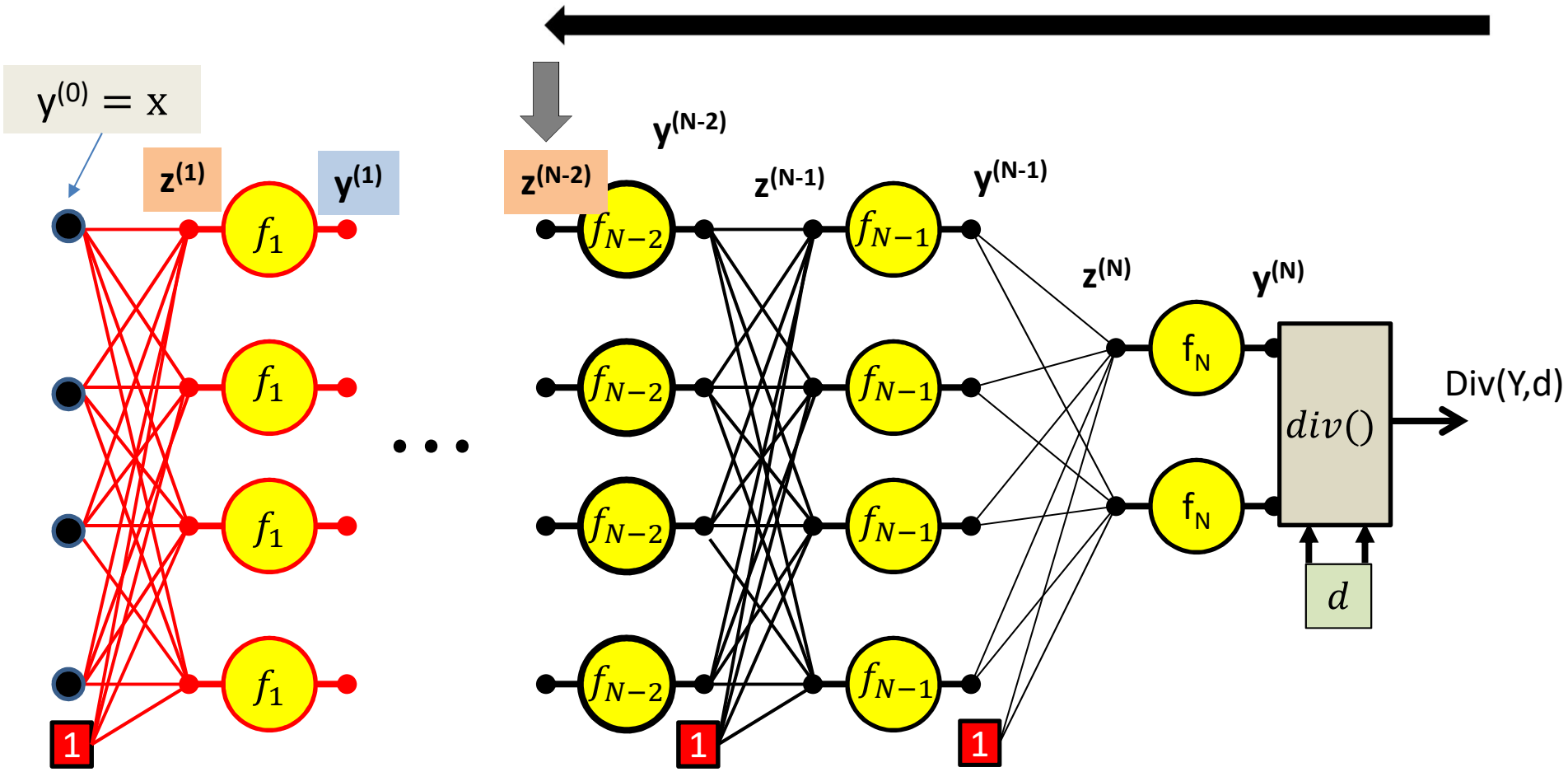
$$\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial Div}{\partial z_j^{(N-1)}}$$

For the bias term $y_0^{(N-2)} = 1$



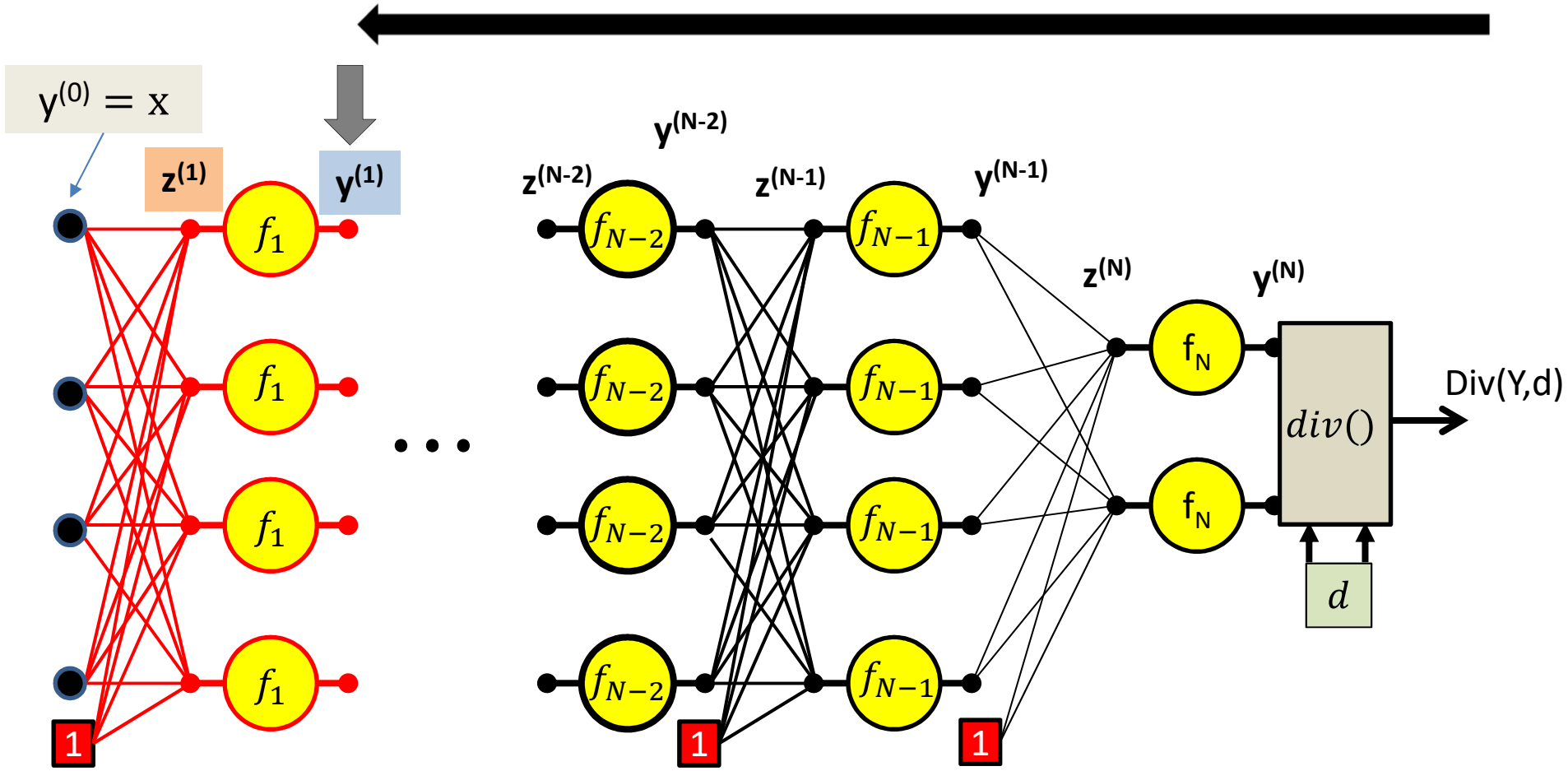
We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial y_i^{(N-2)}} = \sum_j w_{ij}^{(N-1)} \frac{\partial Div}{\partial z_j^{(N-1)}}$$



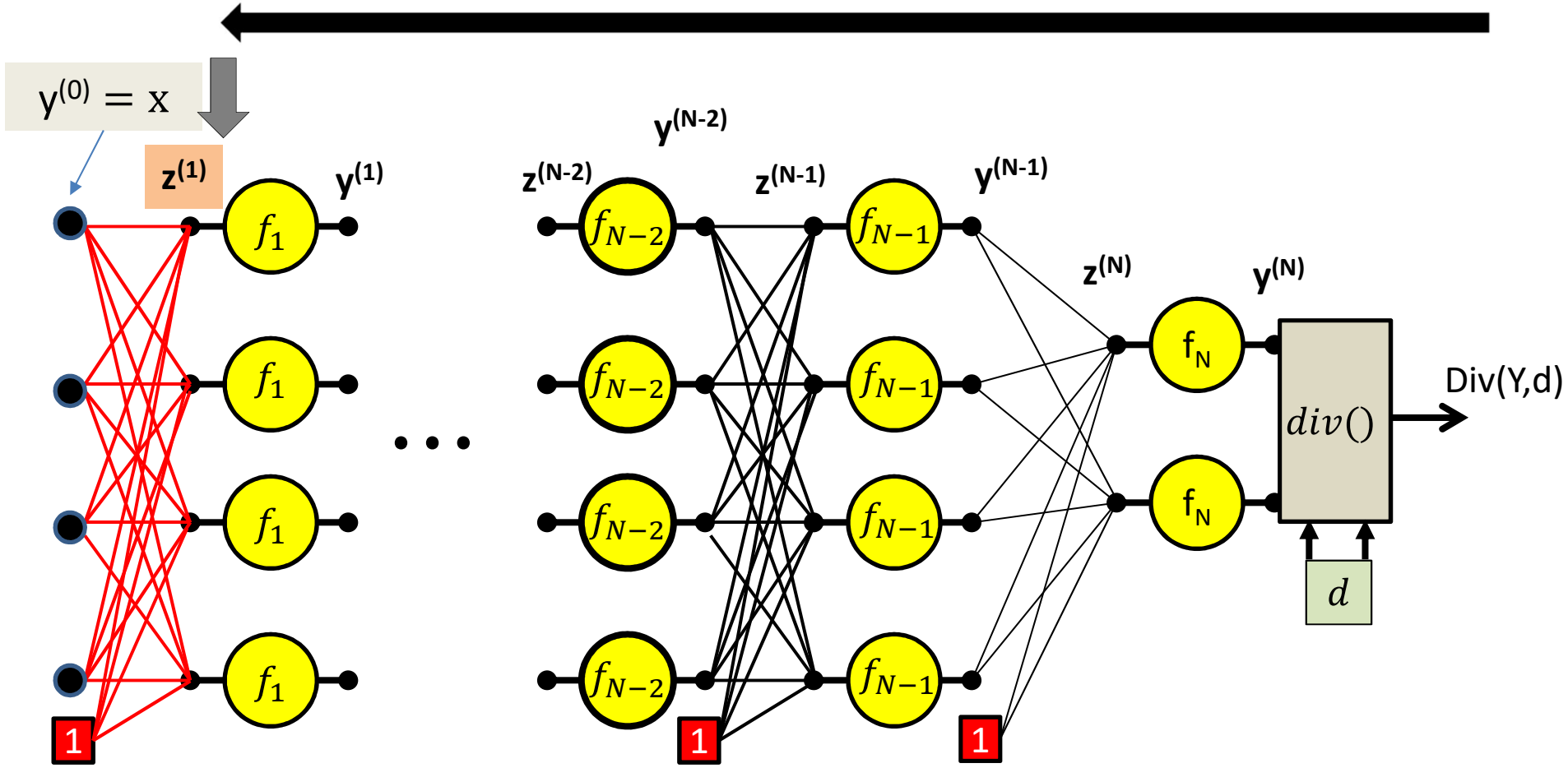
We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial z_i^{(N-2)}} = f'_{N-2}\left(z_i^{(N-2)}\right)\frac{\partial Div}{\partial y_i^{(N-2)}}$$



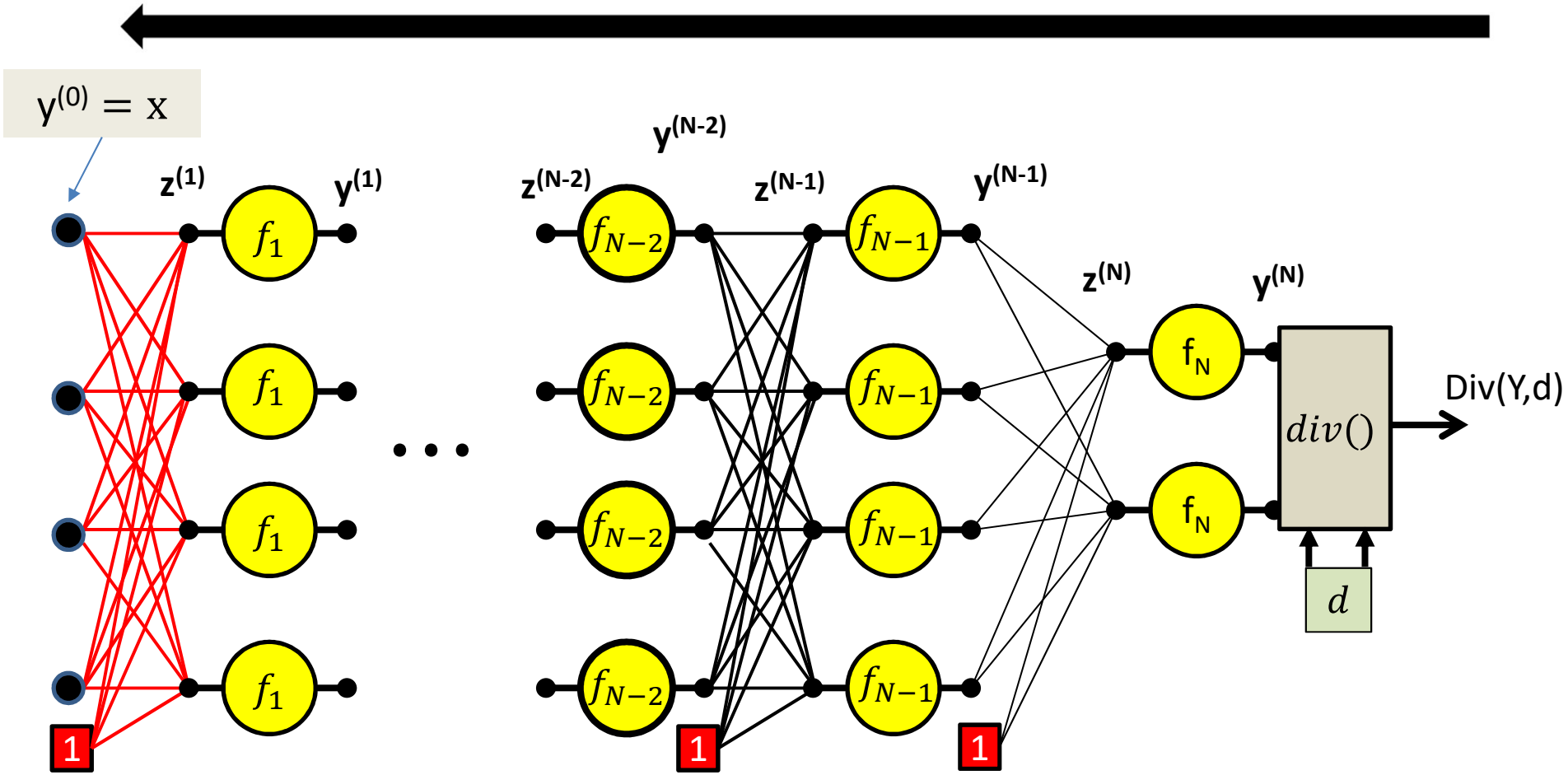
We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial y_1^{(1)}} = \sum_j w_{ij}^{(2)} \frac{\partial Div}{\partial z_j^{(2)}}$$



We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial z_i^{(1)}} = f_1'\left(z_i^{(1)}\right)\frac{\partial Div}{\partial y_i^{(1)}}$$



We continue our way backwards in the order shown

$$\frac{\partial Div}{\partial w_{ij}^{(1)}} = y_i^{(1)} \frac{\partial Div}{\partial z_j^{(1)}}$$

Gradients: Backward Computation

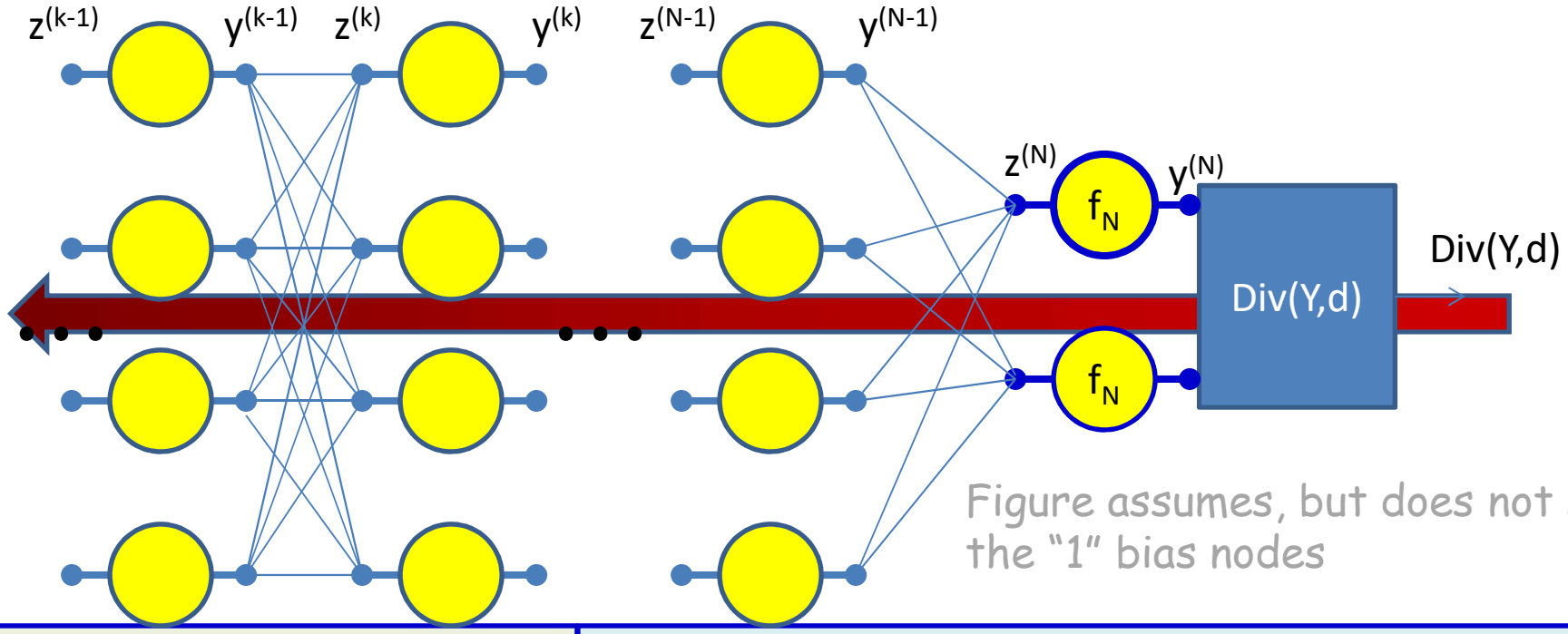


Figure assumes, but does not show the "1" bias nodes

Initialize: Gradient w.r.t network output

$$\frac{\partial Div}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$$

$$\frac{\partial Div}{\partial z_i^{(N)}} = f_k'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

$For\ k = N-1..0$

$For\ i = 1: layer\ width$

$$\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)}\frac{\partial Div}{\partial z_j^{(k+1)}}$$

$$\frac{\partial Div}{\partial z_i^{(k)}} = f_k'\left(z_i^{(k)}\right)\frac{\partial Div}{\partial y_i^{(k)}}$$

$$\forall j\ \frac{\partial Div}{\partial w_{ij}^{(k+1)}} = y_i^{(k)}\frac{\partial Div}{\partial z_j^{(k+1)}}$$

Backward Pass

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Di}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$
- For layer $k = N - 1$ $downto$ 0
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ji}^{(k+1)}} = y_j^{(k)} \frac{\partial Di}{\partial z_i^{(k+1)}}$ for $j = 1 \ldots D_{k+1}$

Backward Pass

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$
- For layer $k = N-1$ *downto* 0
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Di}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ji}^{(k+1)}} = y_j^{(k)} \frac{\partial Div}{\partial z_i^{(k+1)}}$ for $j = 1 \ldots D_{k+1}$

Called "Backpropagation" because the derivative of the error is propagated "backwards" through the network

Very analogous to the forward pass:

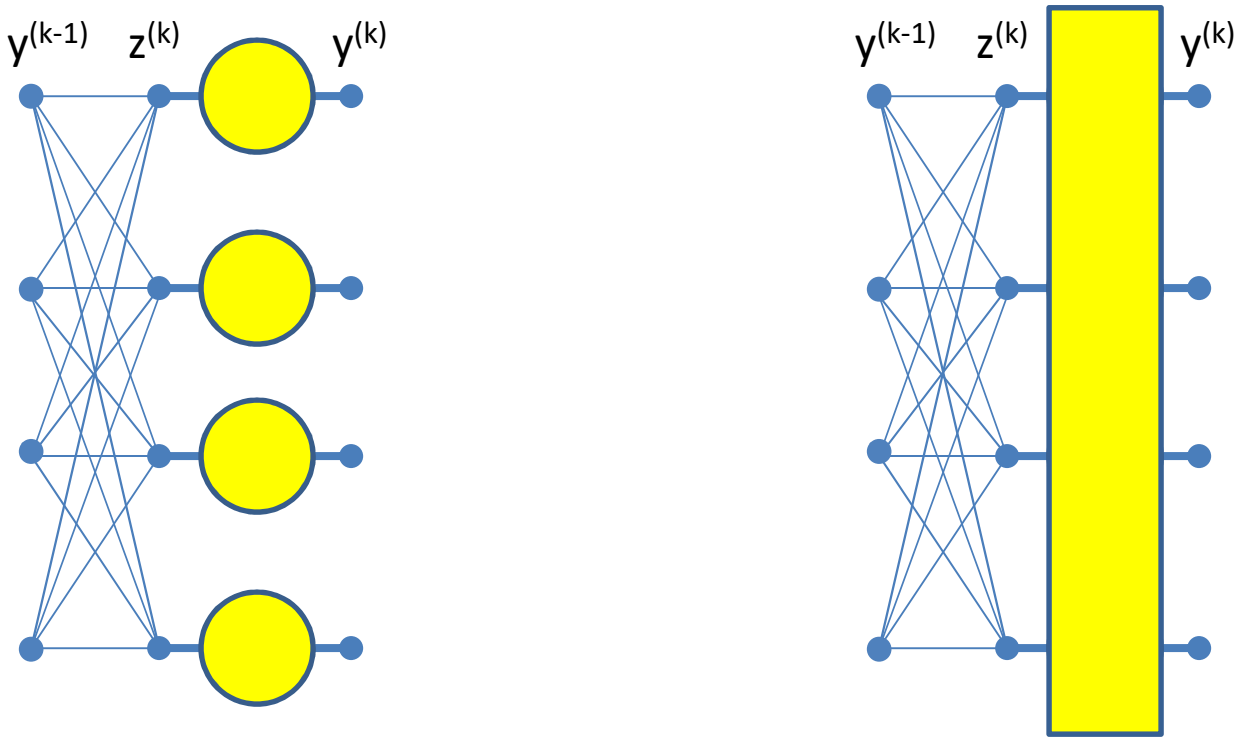
Backward weighted combination of next layer

Backward equivalent of activation

For comparison: the forward pass again

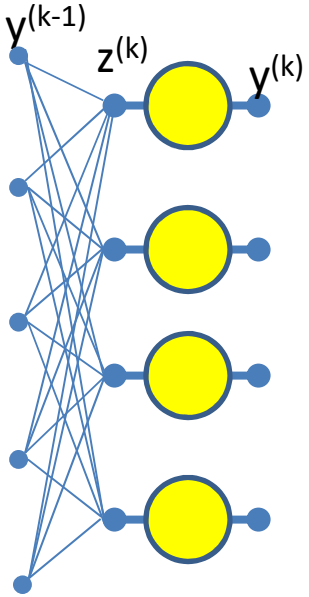
- Input: D dimensional vector $\mathbf{x} = [x_j,\ j = 1 \ldots D]$
- Set:
  - $D_0 = D,$ is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j,\ j = 1 \ldots D; \quad y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$
    - $z_j^{(k)} = \sum_{i=0}^{N_k} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k\left(z_j^{(k)}\right)$
- Output:
  - $Y = y_j^{(N)}, j = 1..D_N$

Special Case 1. Vector activations



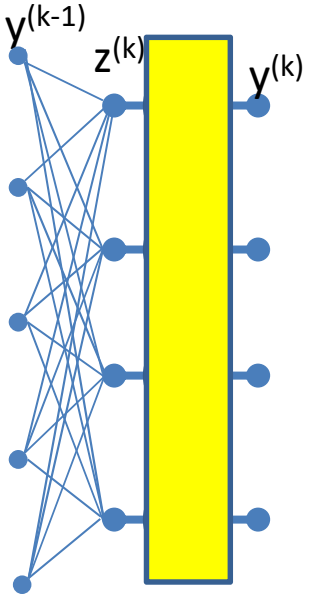
- Vector activations: all outputs are functions of all inputs

Special Case 1. Vector activations



Scalar activation: Modifying a z_i only changes corresponding y_i

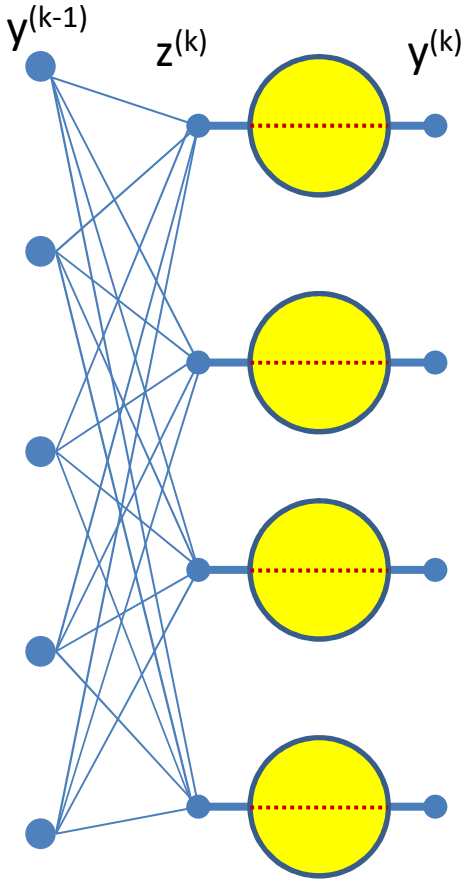
$$y_i^{(k)} = f\left(z_i^{(k)}\right)$$



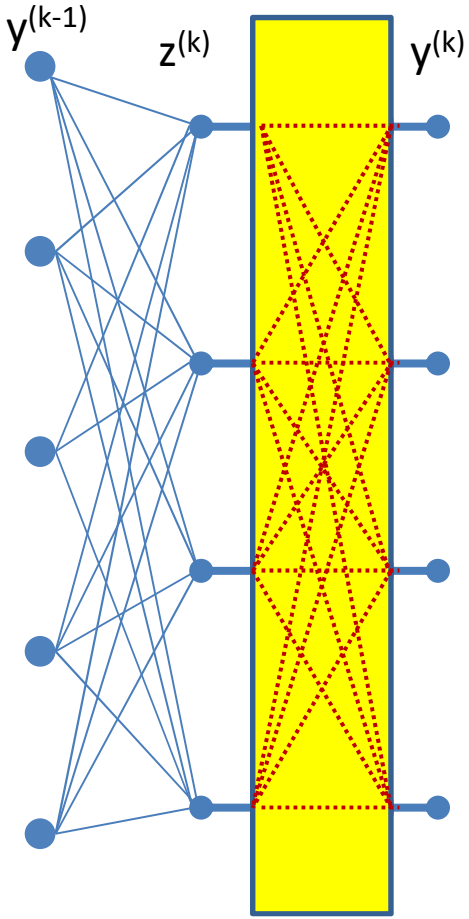
Vector activation: Modifying a z_i potentially changes all, $y_1 \ldots y_M$

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

“Influence” diagram

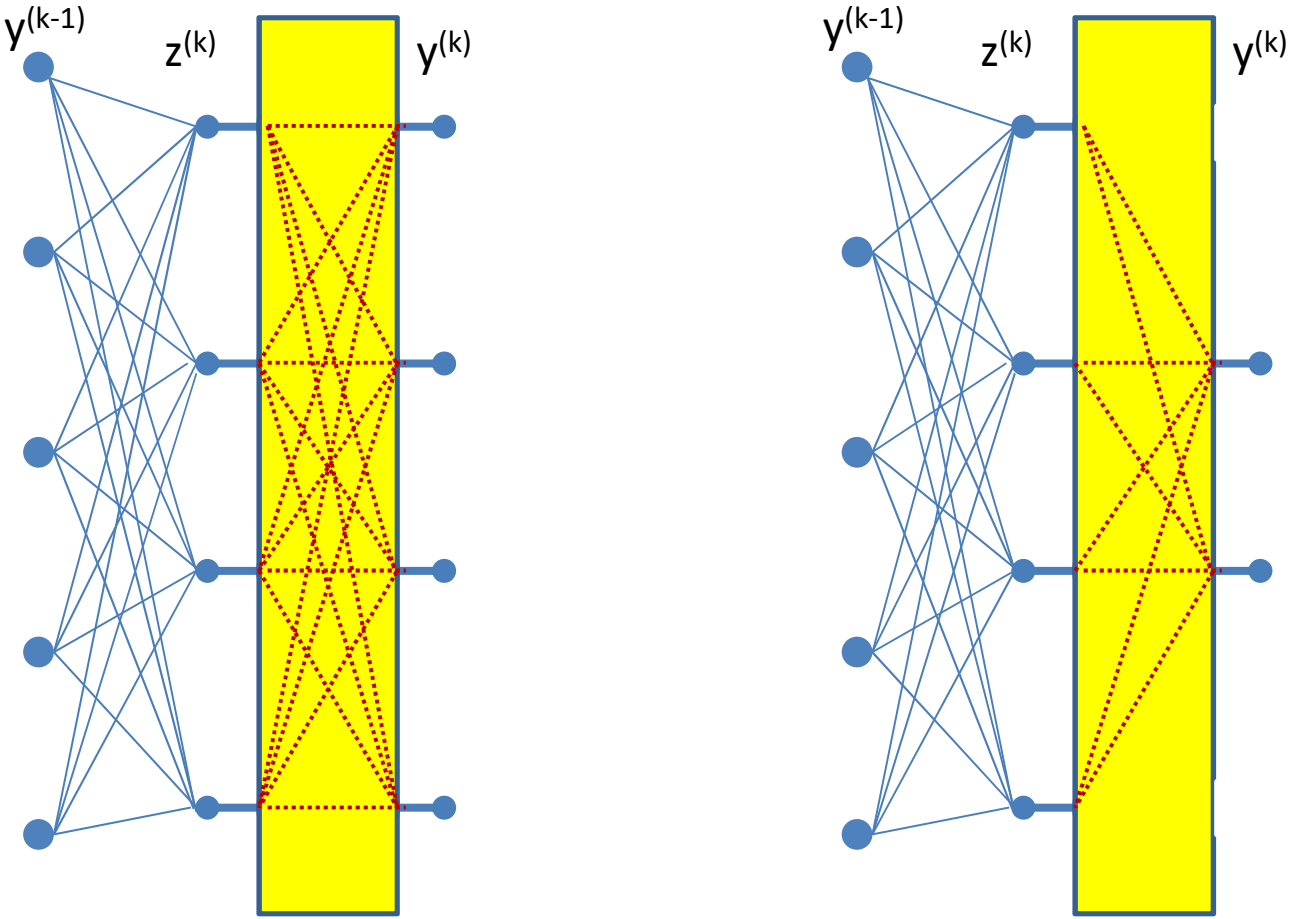


Scalar activation: Each z_i influences *one* y_i



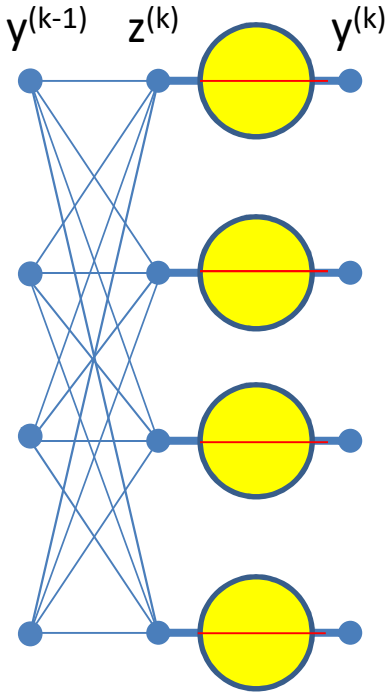
Vector activation: Each z_i influences all, $y_1 \ldots y_M$

The number of outputs



- Note: The number of outputs ($y^{(k)}$) need not be the same as the number of inputs ($z^{(k)}$)
  - May be more or fewer

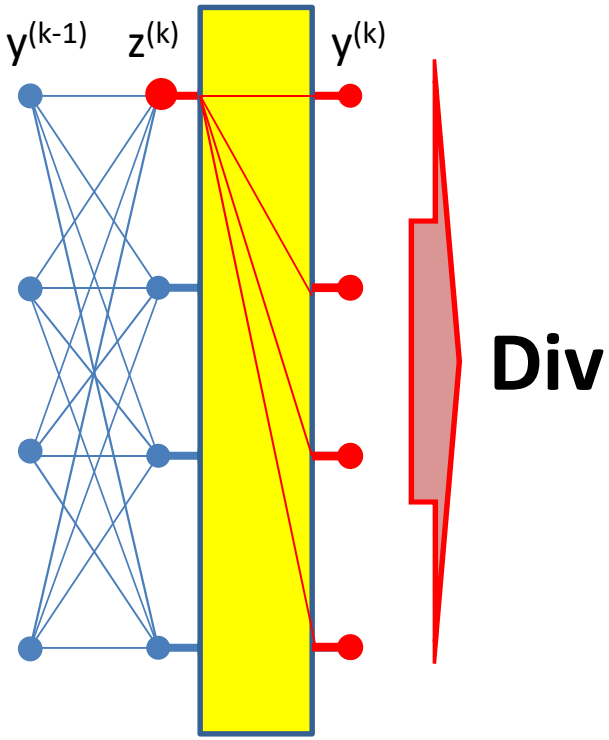
Scalar Activation: Derivative rule



$$\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{dy_i^{(k)}}{dz_i^{(k)}}$$

- In the case of *scalar* activation functions, the derivative of the error w.r.t to the input to the unit is a simple product of derivatives

Derivatives of vector activation



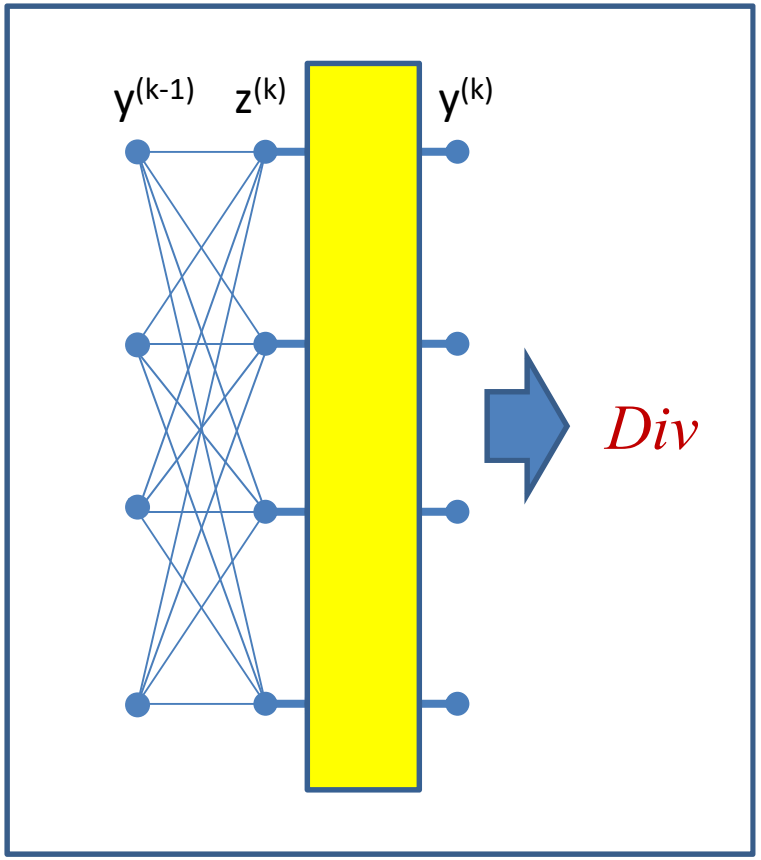
$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

Note: derivatives of scalar activations are just a special case of vector activations:

$$\frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} = 0 \;\; for \;\; i \neq j$$

- For *vector* activations the derivative of the error w.r.t. to any input is a sum of partial derivatives
  - Regardless of the number of outputs $y_j^{(k)}$

Example Vector Activation: Softmax



$$y_i^{(k)} = \frac{exp\left(z_i^{(k)}\right)}{\sum_j exp\left(z_j^{(k)}\right)}$$

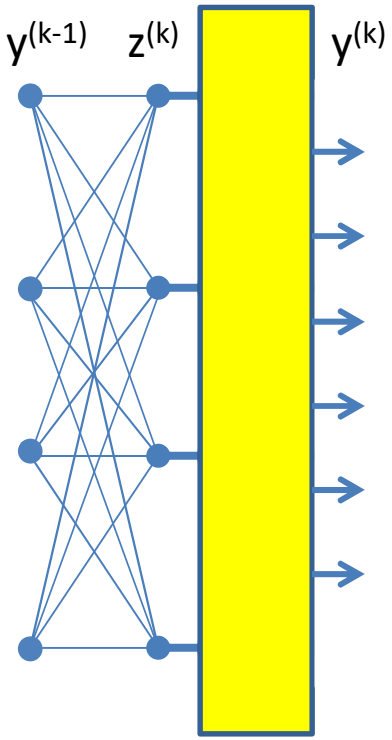
$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

$$\frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} = \begin{cases} y_i^{(k)}\left(1 - y_i^{(k)}\right) & \text{if } i = j \\ -y_i^{(k)} y_j^{(k)} & \text{if } i \neq j \end{cases}$$

$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} y_i^{(k)}\left(\delta_{ij} - y_j^{(k)}\right)$$

- For future reference
- δ_{ij} is the Kronecker delta: $\delta_{ij} = 1 \text{ if } i = j, \quad 0 \text{ if } i \neq j$

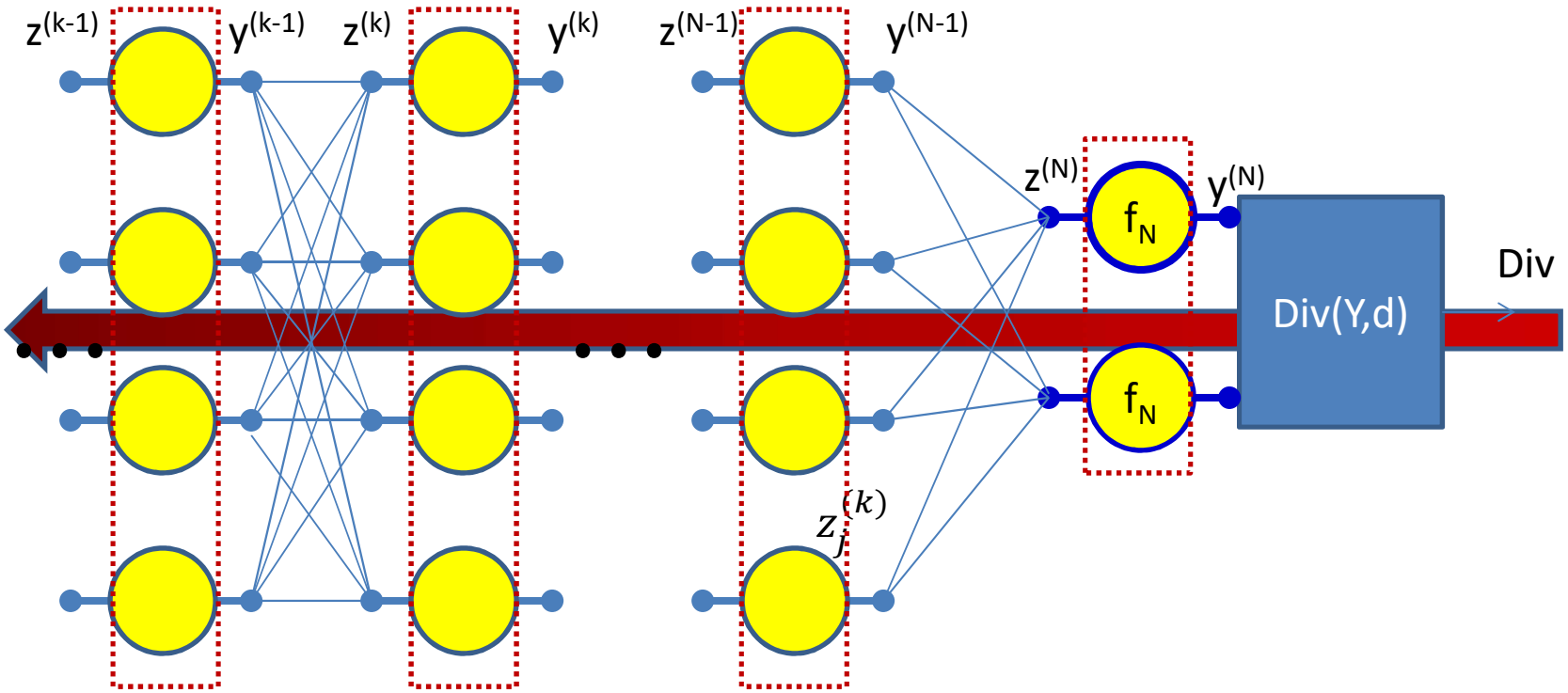
Vector Activations



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

- In reality the vector combinations can be anything
  - E.g. linear combinations, polynomials, logistic (softmax), etc.

Gradients: Backward Computation



For $k = N...1$

For $i = 1$:layer width

If layer has vector activation

$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

$\Leftrightarrow$ Else if activation is scalar

$$\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$$

$$\frac{\partial Div}{\partial y_i^{(k-1)}} = \sum_j w_{ij}^{(k)} \frac{\partial Div}{\partial z_j^{(k)}}$$

$$\frac{\partial Div}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial Div}{\partial z_j^{(k)}}$$

Backward Pass: Recap

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial Y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Di}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$ $\quad OR \quad$ $\sum_j \frac{\partial Div}{\partial y_j^{(N)}} \frac{\partial y_j^{(N)}}{\partial z_i^{(N)}}$ (vector activation)
- For layer $k = N - 1$ $downto$ 0
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$ $\quad OR \quad$ $\sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$ (vector activation)
    - $\frac{\partial Div}{\partial w_{ji}^{(k+1)}} = y_j^{(k)} \frac{\partial Div}{\partial z_i^{(k+1)}}$ for $j = 1 \ldots D_{k+1}$

Overall Approach

- For each data instance
  - **Forward pass**: Pass instance forward through the net. Store all intermediate outputs of all computation
  - **Backward pass**: Sweep backward through the net, iteratively compute all derivatives w.r.t weights
- Actual Error is the sum of the error over all training instances

$$\mathbf{Err} = \frac{1}{|\{X\}|}\sum_{X} Div(Y(X), d(X))$$

- Actual gradient is the sum or average of the derivatives computed for each training instance

$$\nabla_W \mathbf{Err} = \frac{1}{|\{X\}|}\sum_{X} \nabla_W Div(Y(X), d(X)) \qquad W \leftarrow W - \eta \nabla_W \mathbf{Err}$$

Training by BackProp

- Initialize all weights $(\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots, \boldsymbol{W}^{(K)})$
- Do:
  - **Initialize** $Err = 0$; For all i, j, k, initialize $\frac{dErr}{dw_{i,j}^{(k)}} = 0$
  - For all $t = 1:T$ (Loop over training instances)
    - **Forward pass:** Compute
      - Output $\boldsymbol{Y_t}$
      - $Err \mathrel{+}= \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$
    - **Backward pass:** For all i, j, k:
      - Compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
      - Compute $\frac{dErr}{dw_{i,j}^{(k)}} \mathrel{+}= \frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
  - For all i, j, k, update:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} - \frac{\eta}{T} \frac{dErr}{dw_{i,j}^{(k)}}$$

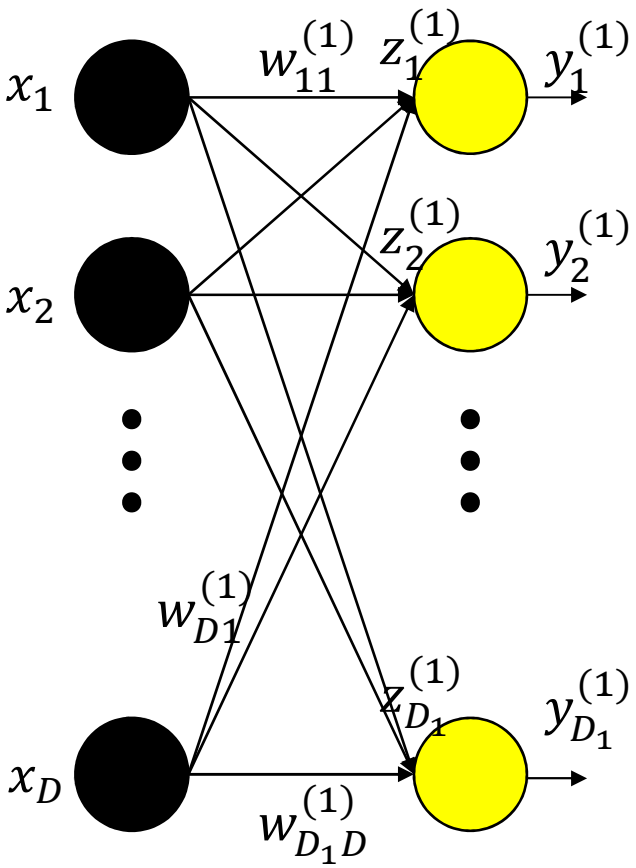
- Until Err has converged

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

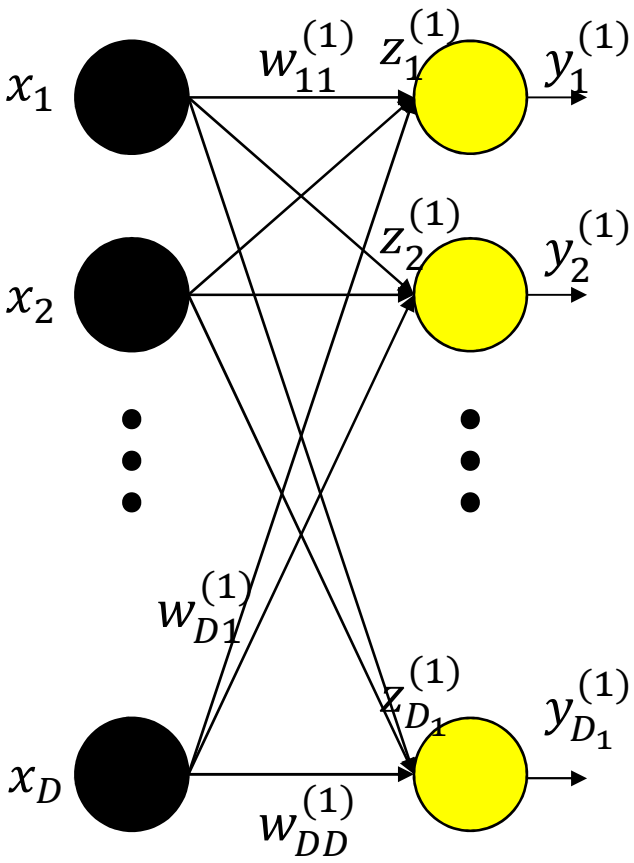


$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \qquad \mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

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

- 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



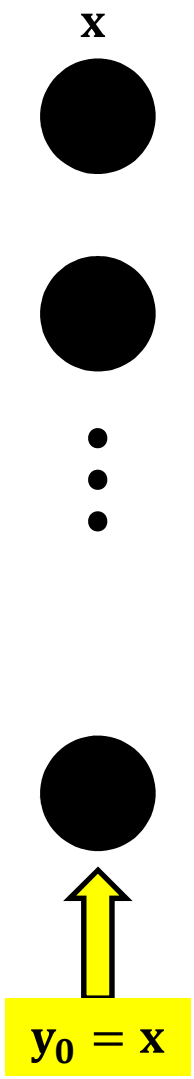
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \qquad \mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

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

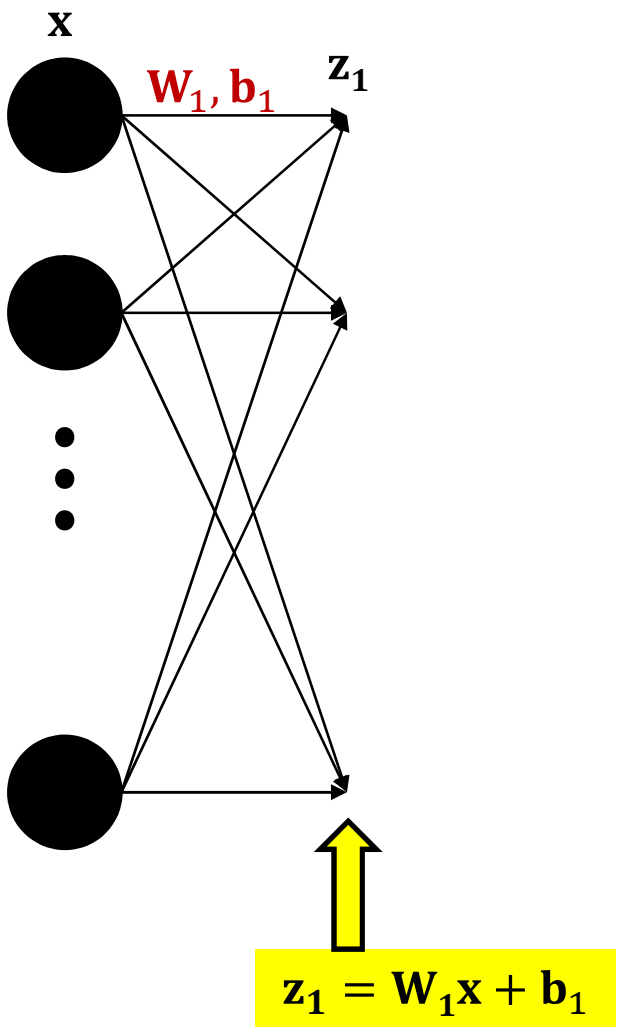
- The computation of a single layer is easily expressed in matrix notation as (setting $\mathbf{y_0} = \mathbf{x}$):

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

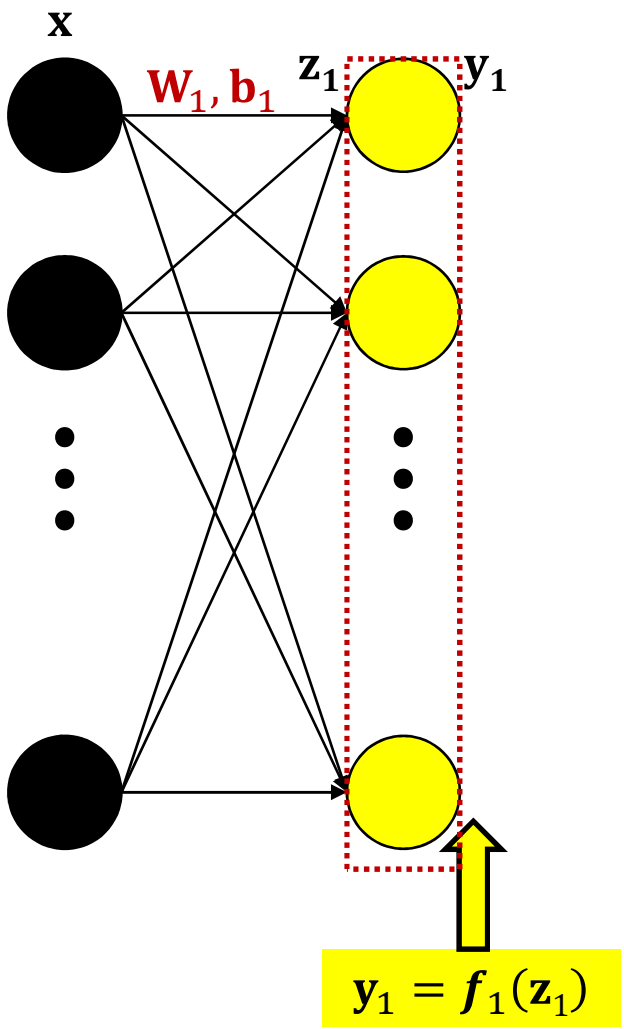
The forward pass: Evaluating the network



The forward pass



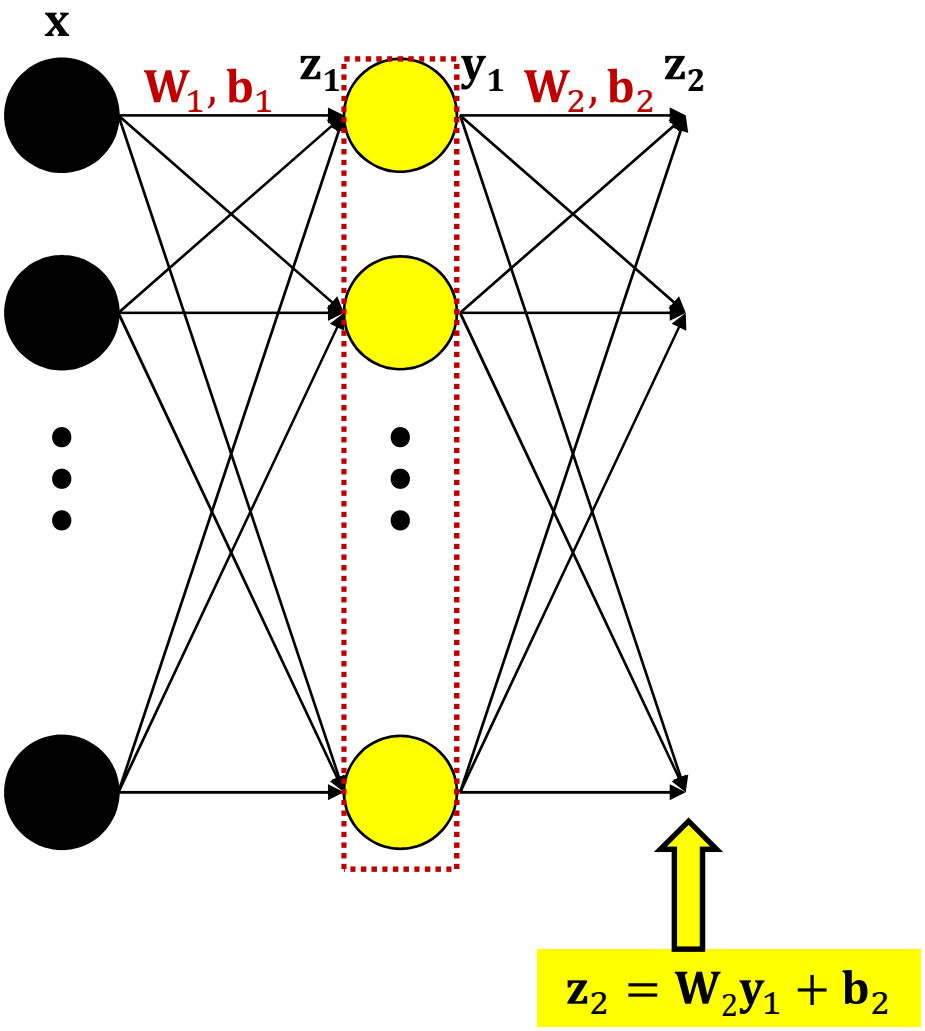
The forward pass



The Complete computation

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

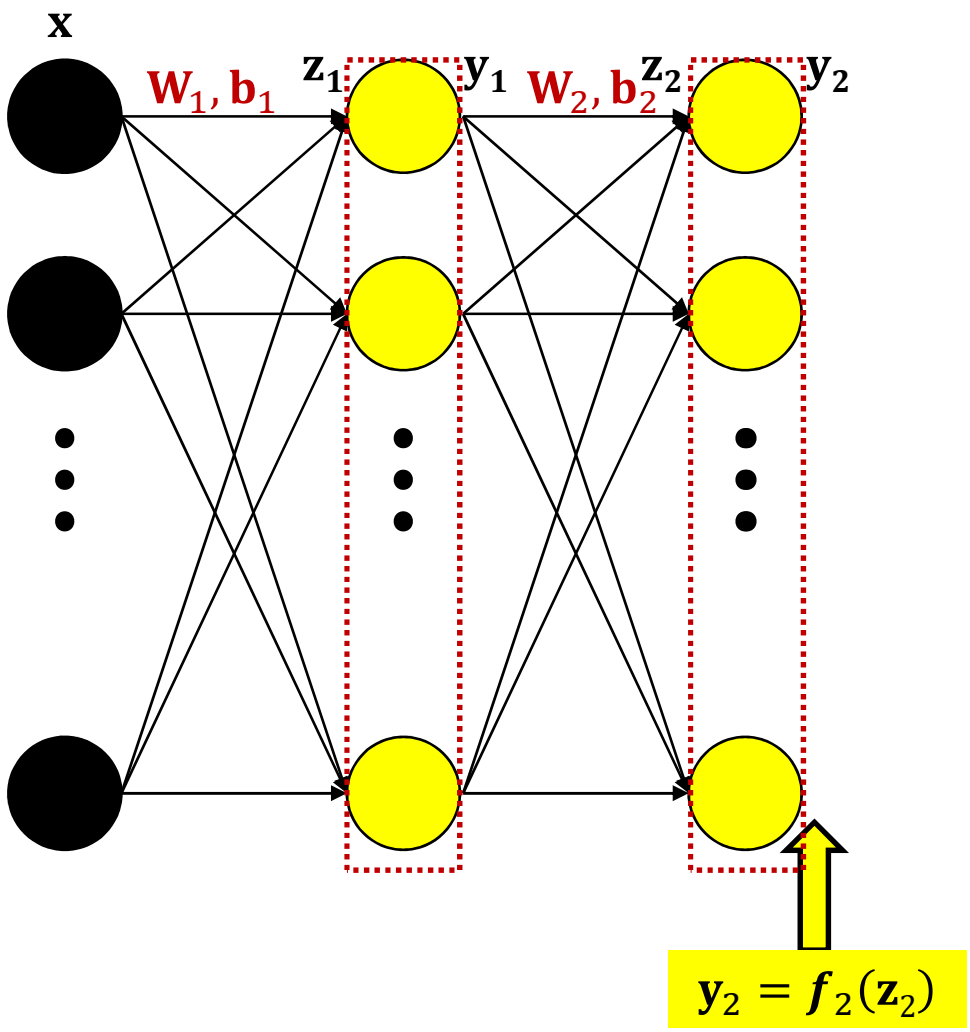
The forward pass



The Complete computation

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

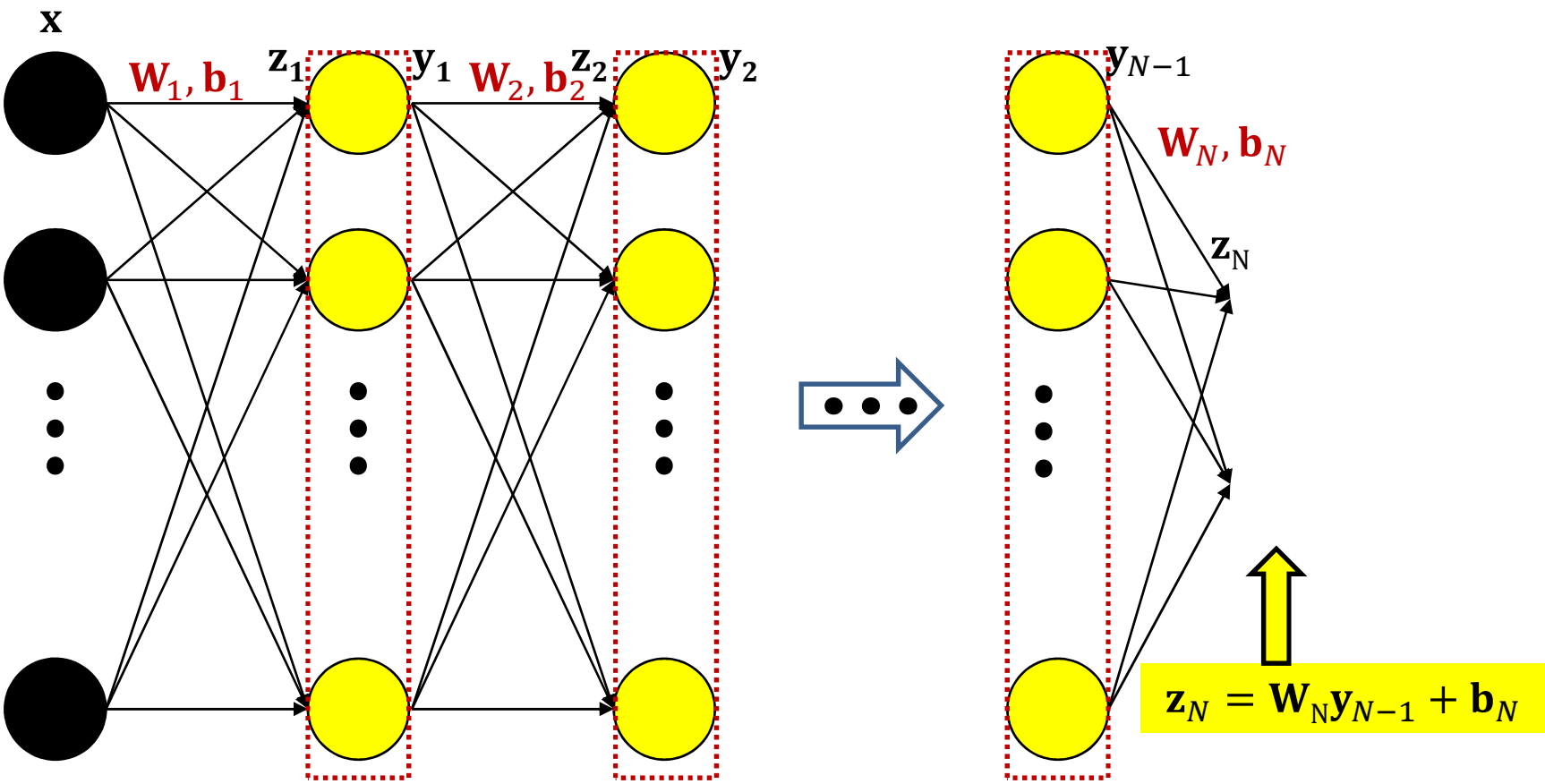
The forward pass



The Complete computation

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

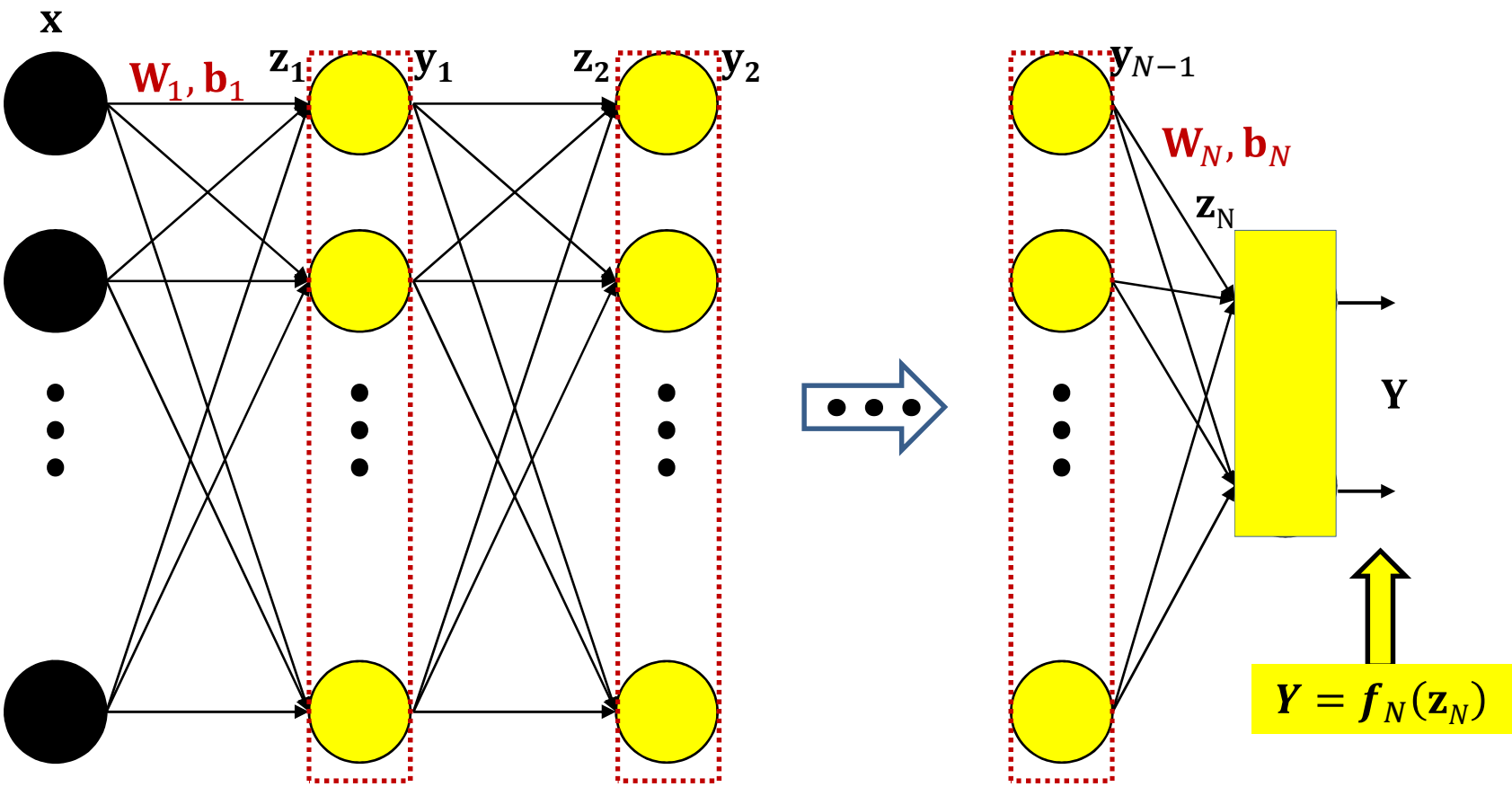
The forward pass



The Complete computation

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

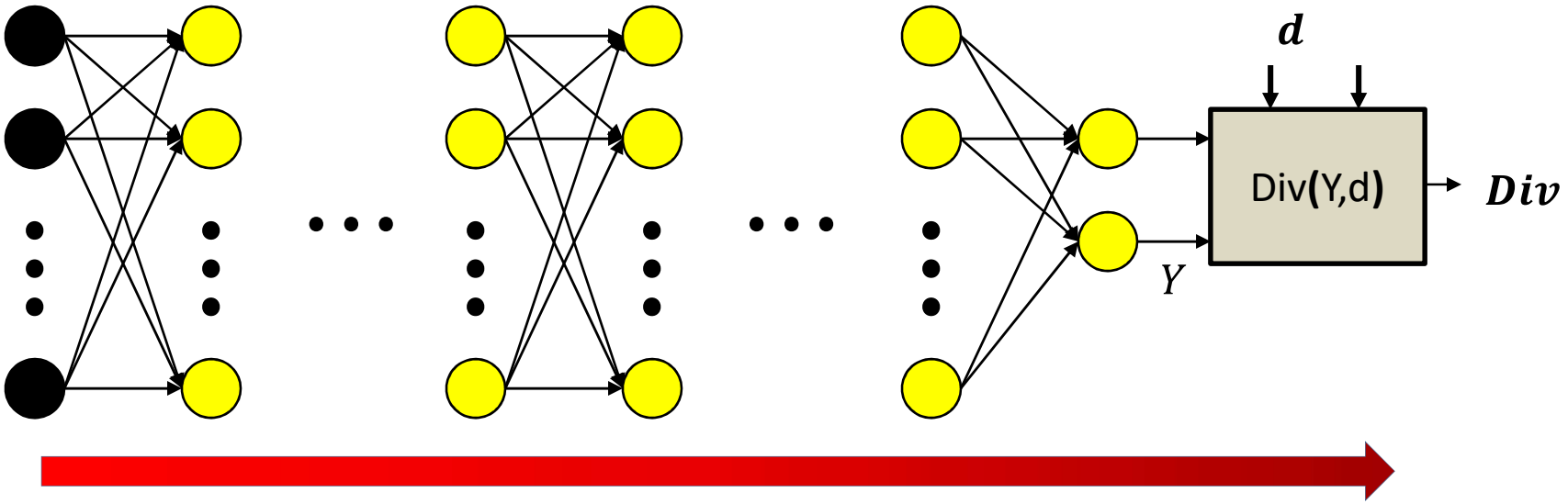
The forward pass



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



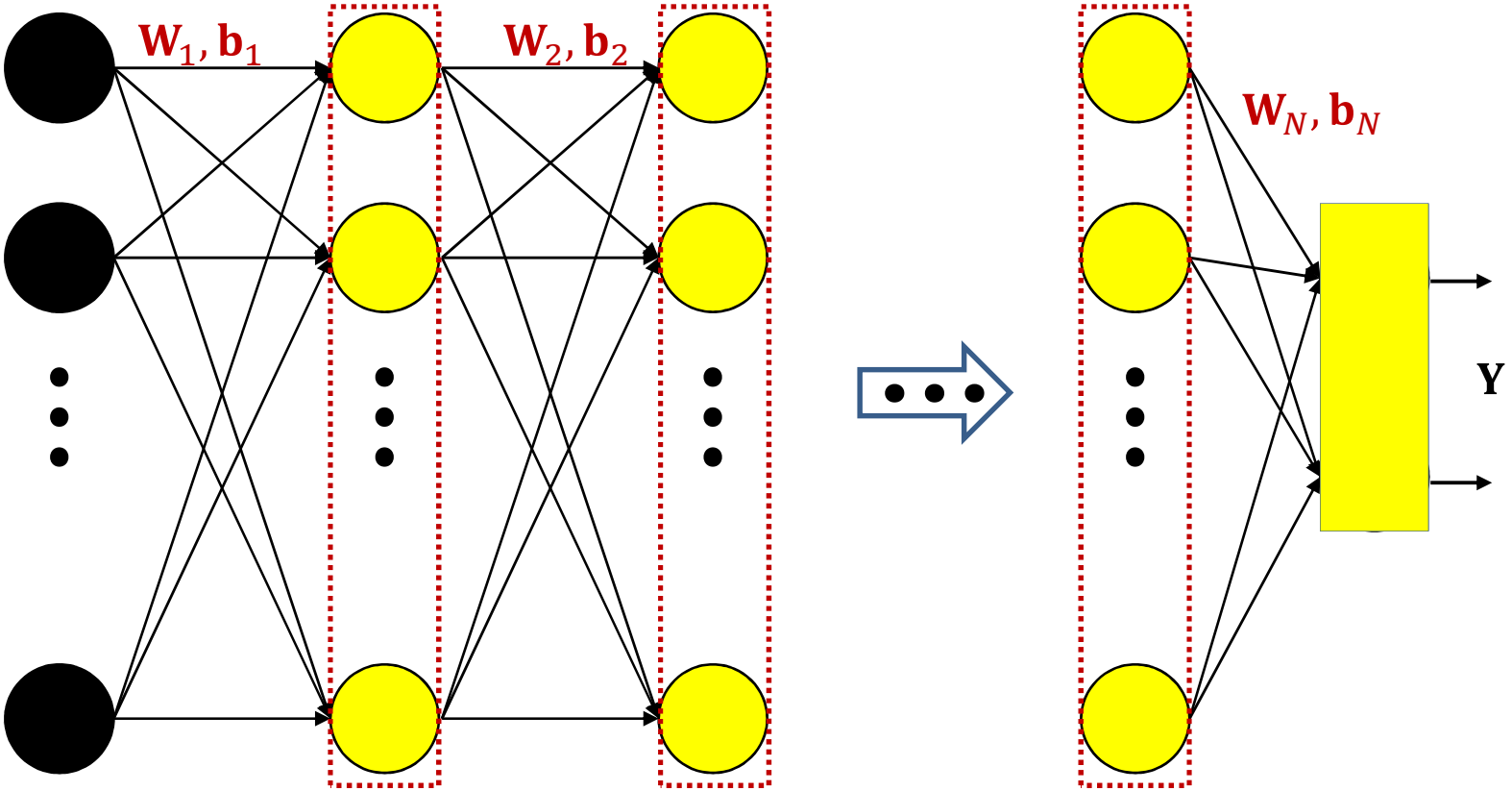
Forward pass:

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

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

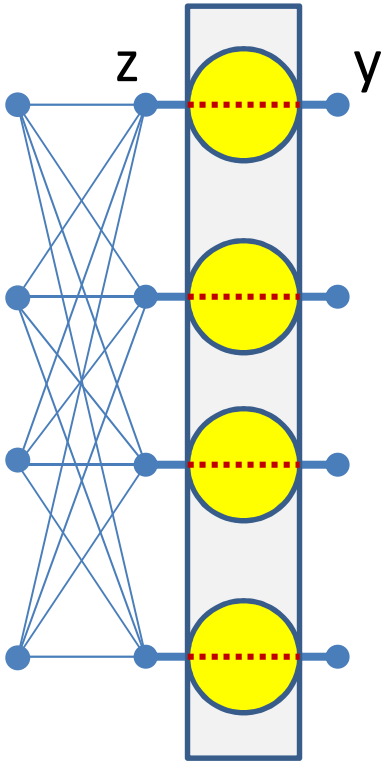
Using vector notation

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

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

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

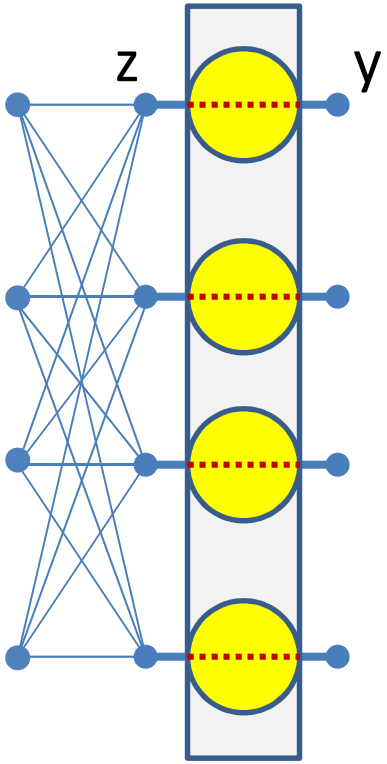
Jacobians can describe the derivatives of neural activations w.r.t their input



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

- **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

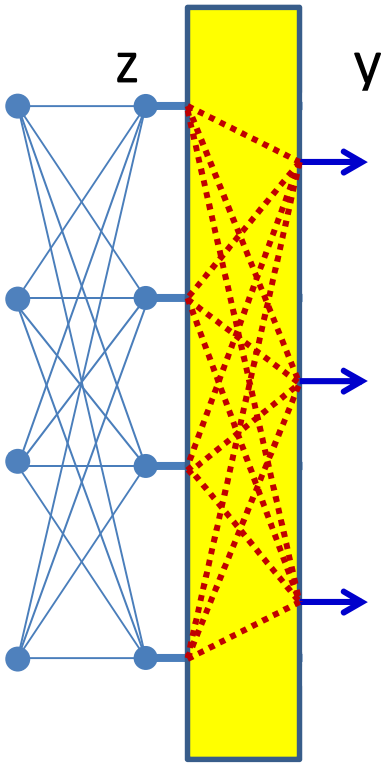


$$y_i = f(z_i)$$

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

- **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

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

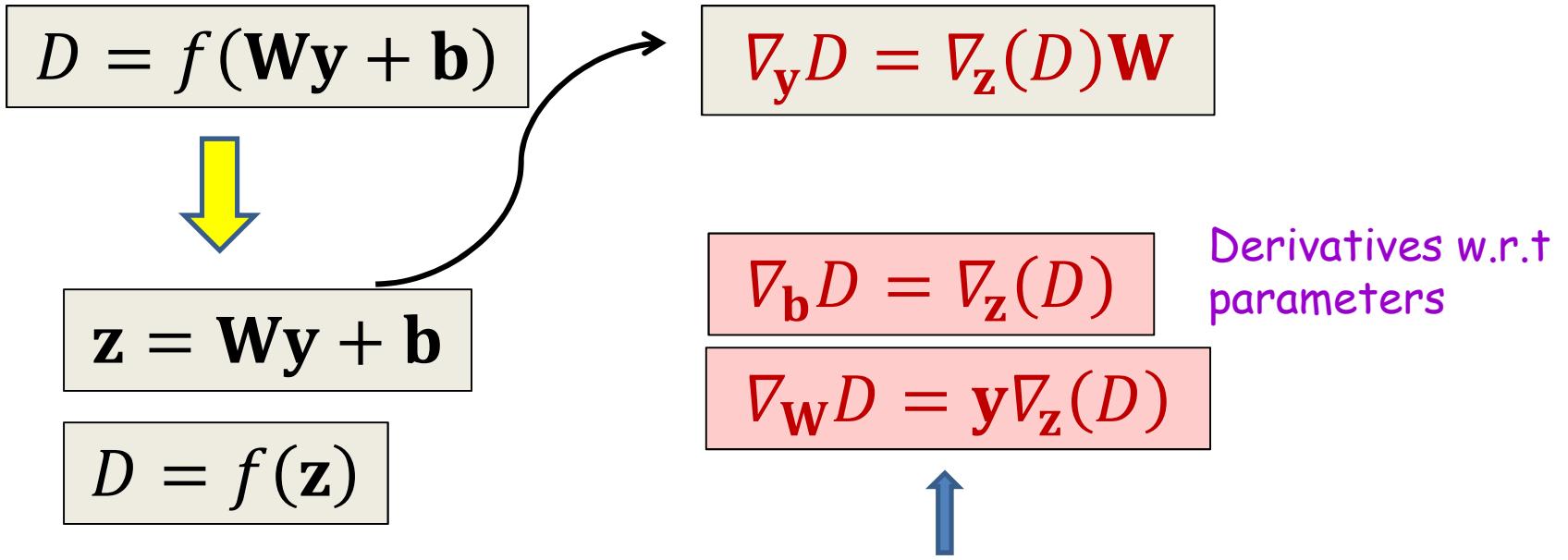
$$\Delta D = \nabla_{\mathbf{z}}(D)\Delta\mathbf{z}$$

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

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

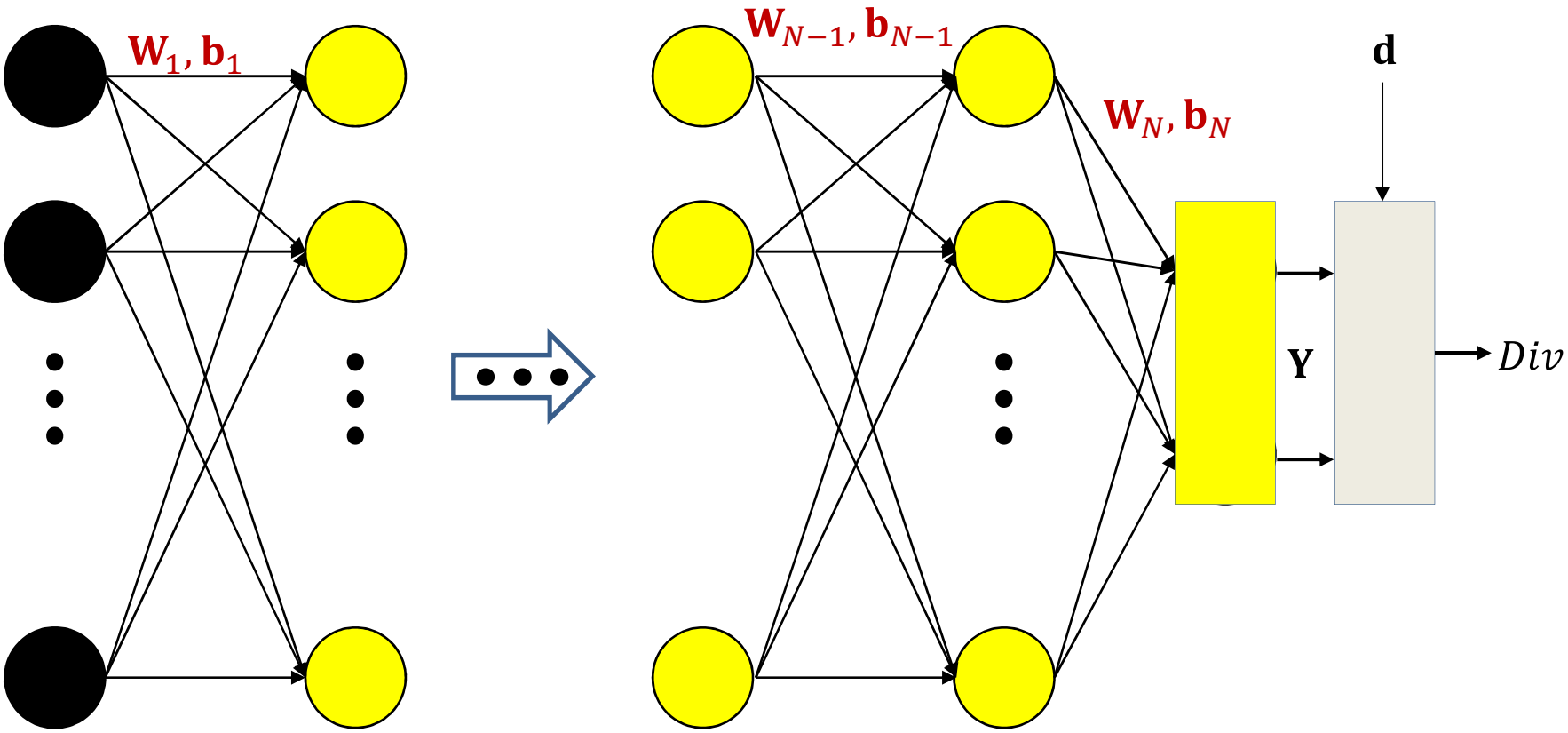
Special Case

- 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

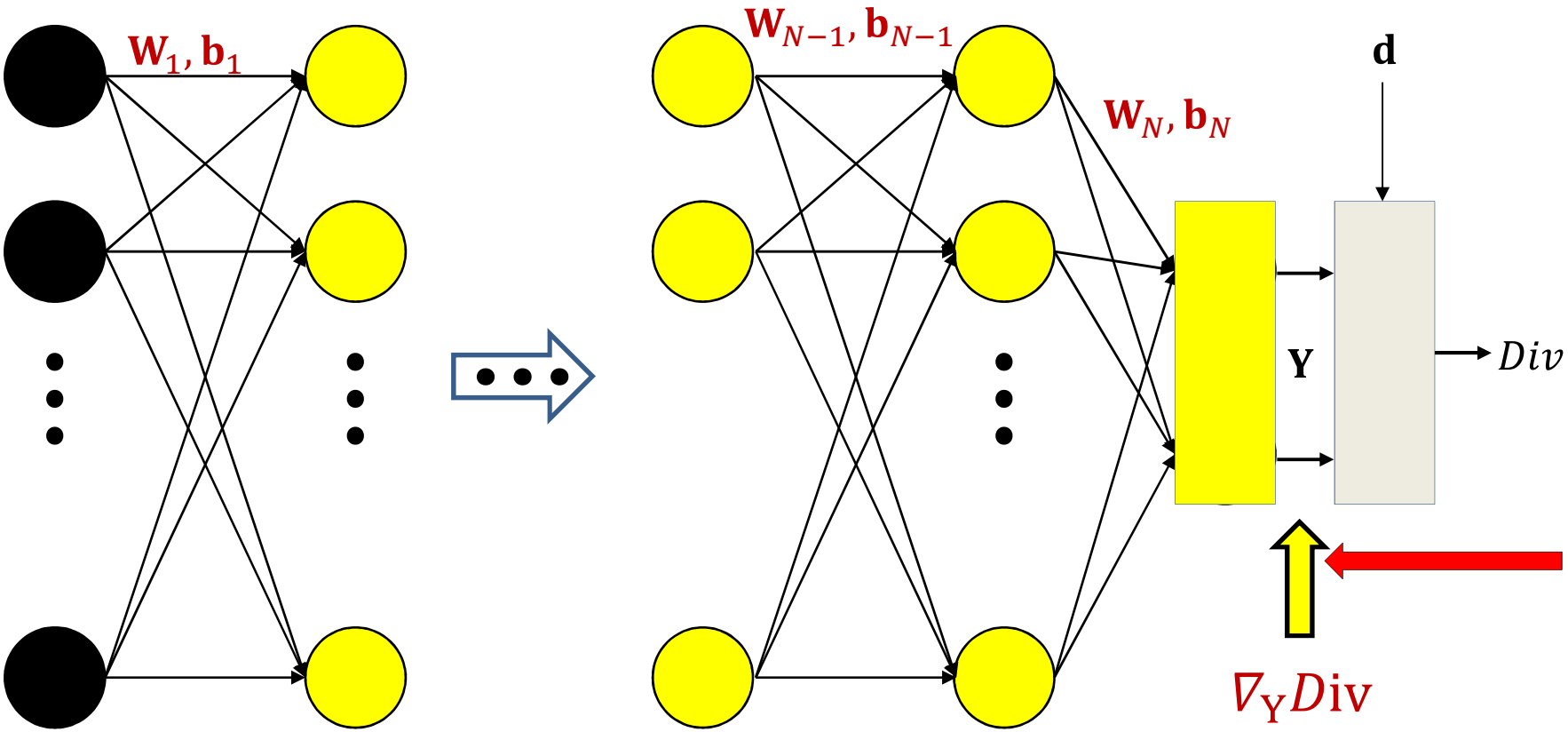
The backward pass



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

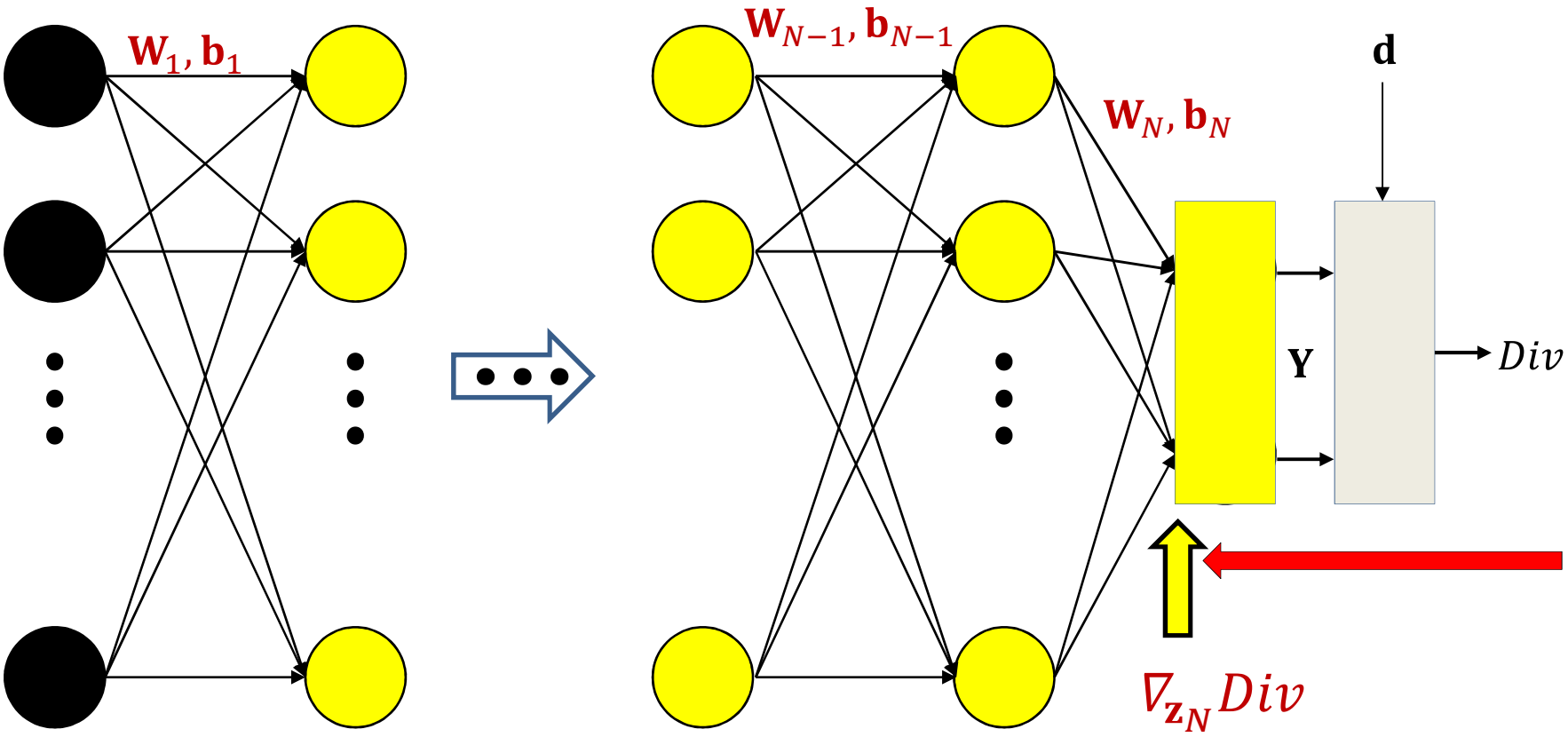
In general $\nabla_{\mathbf{a}}\mathbf{b}$ represents a derivative of $\mathbf{b}$ w.r.t. $\mathbf{a}$ and could be a gradient (for scalar $\mathbf{b}$) Or a Jacobian (for vector $\mathbf{b}$)

The backward pass



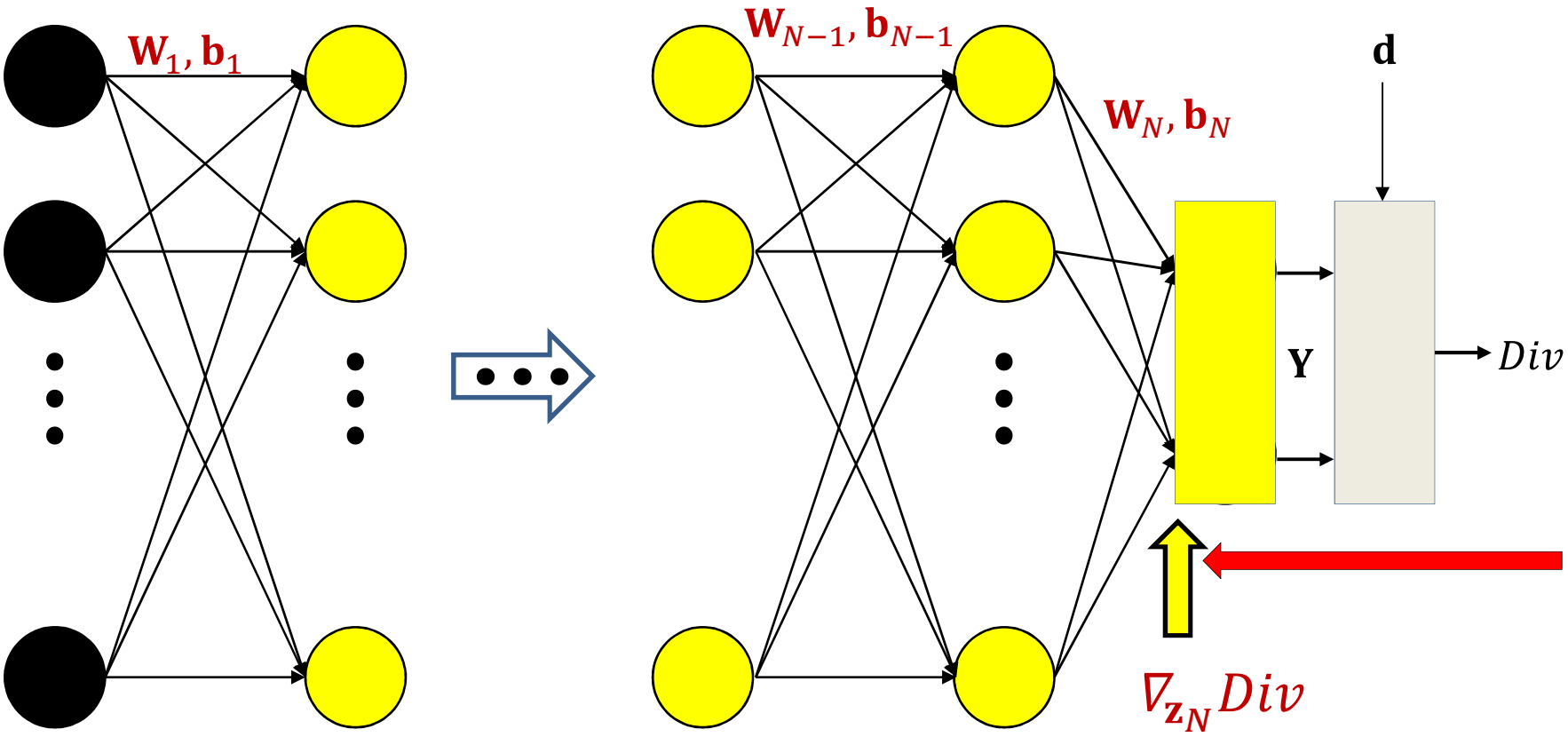
First compute the gradient of the divergence w.r.t. Y.
The actual gradient depends on the divergence function.

The backward pass



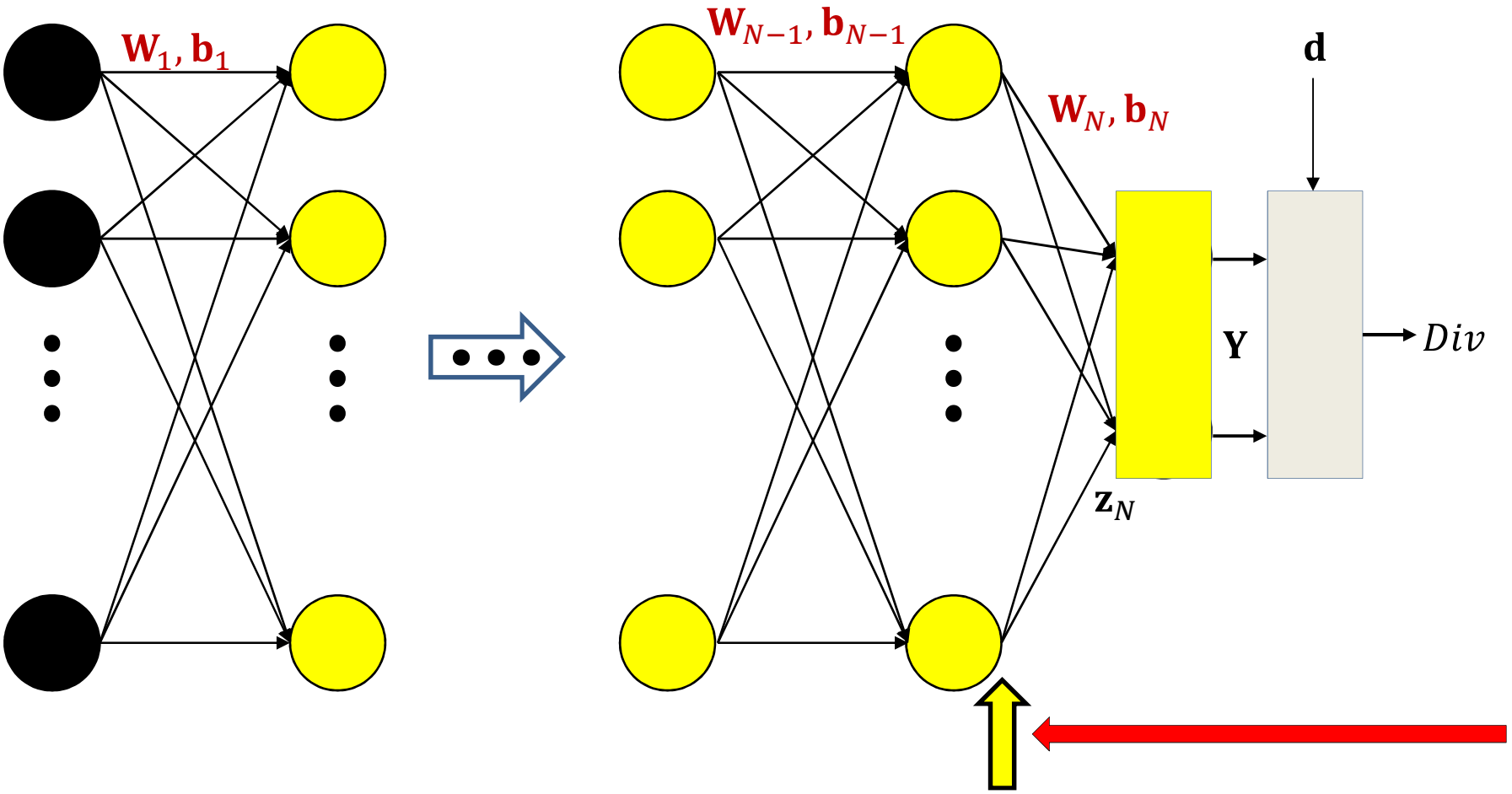
$$\nabla_{\mathbf{z}_N} Div = \nabla_{\mathbf{Y}} Div . \nabla_{\mathbf{z}_N} \mathbf{Y}$$

The backward pass



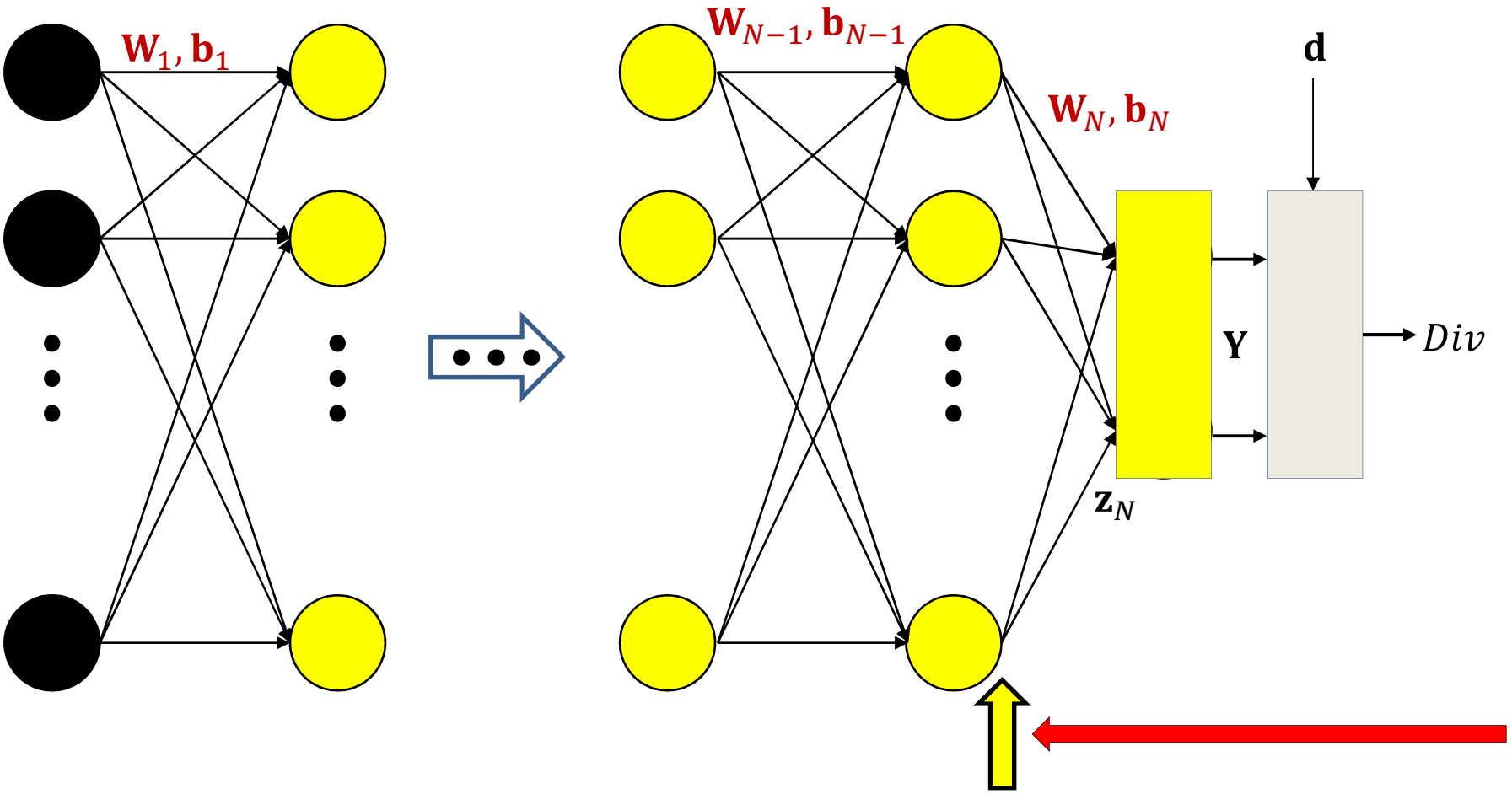
$$\nabla_{\mathbf{z}_N} Div = \nabla_{\mathbf{Y}} Div \, J_{\mathbf{Y}}(\mathbf{z}_N)$$

The backward pass



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

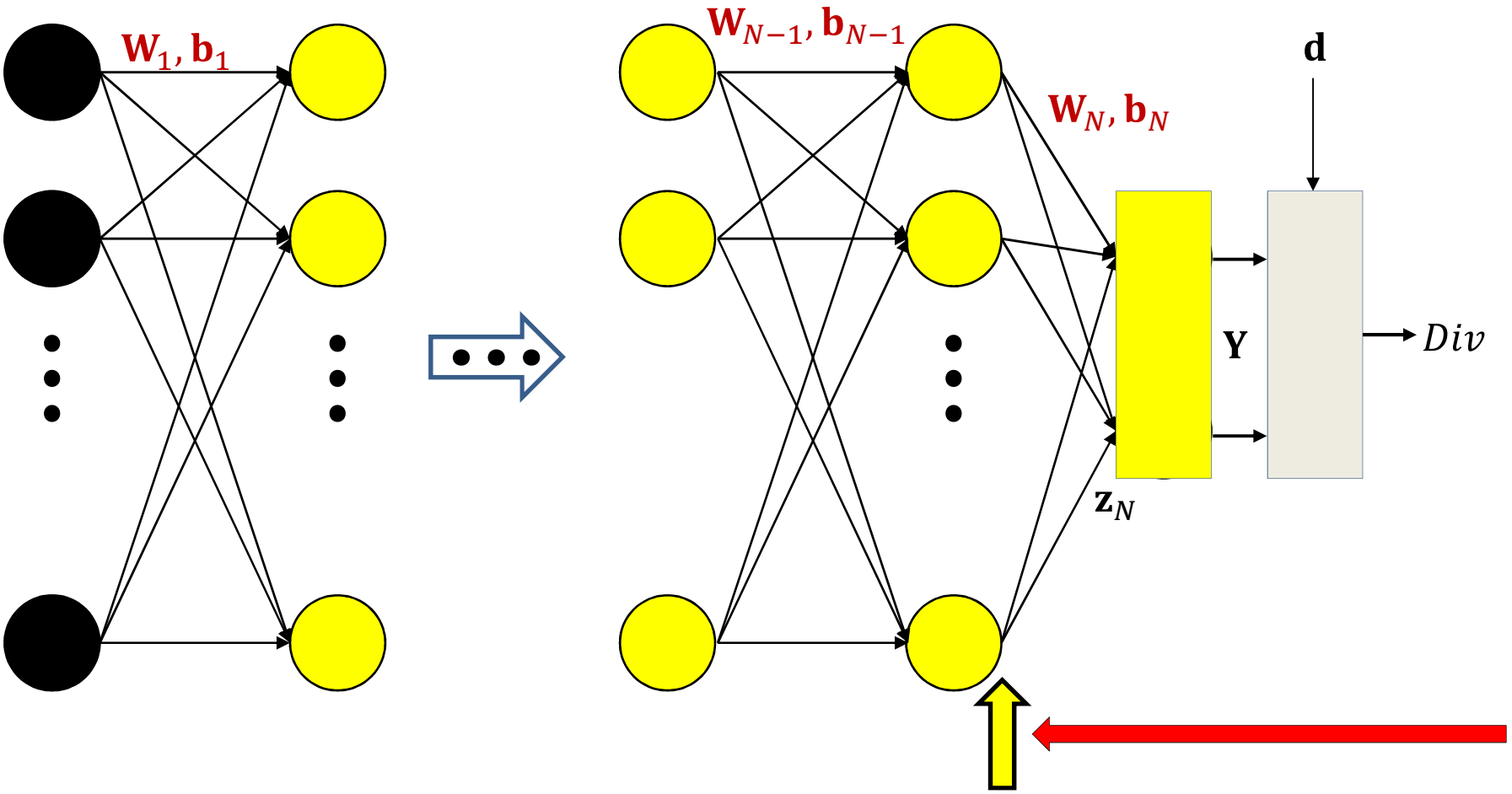
The backward pass



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

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

The backward pass

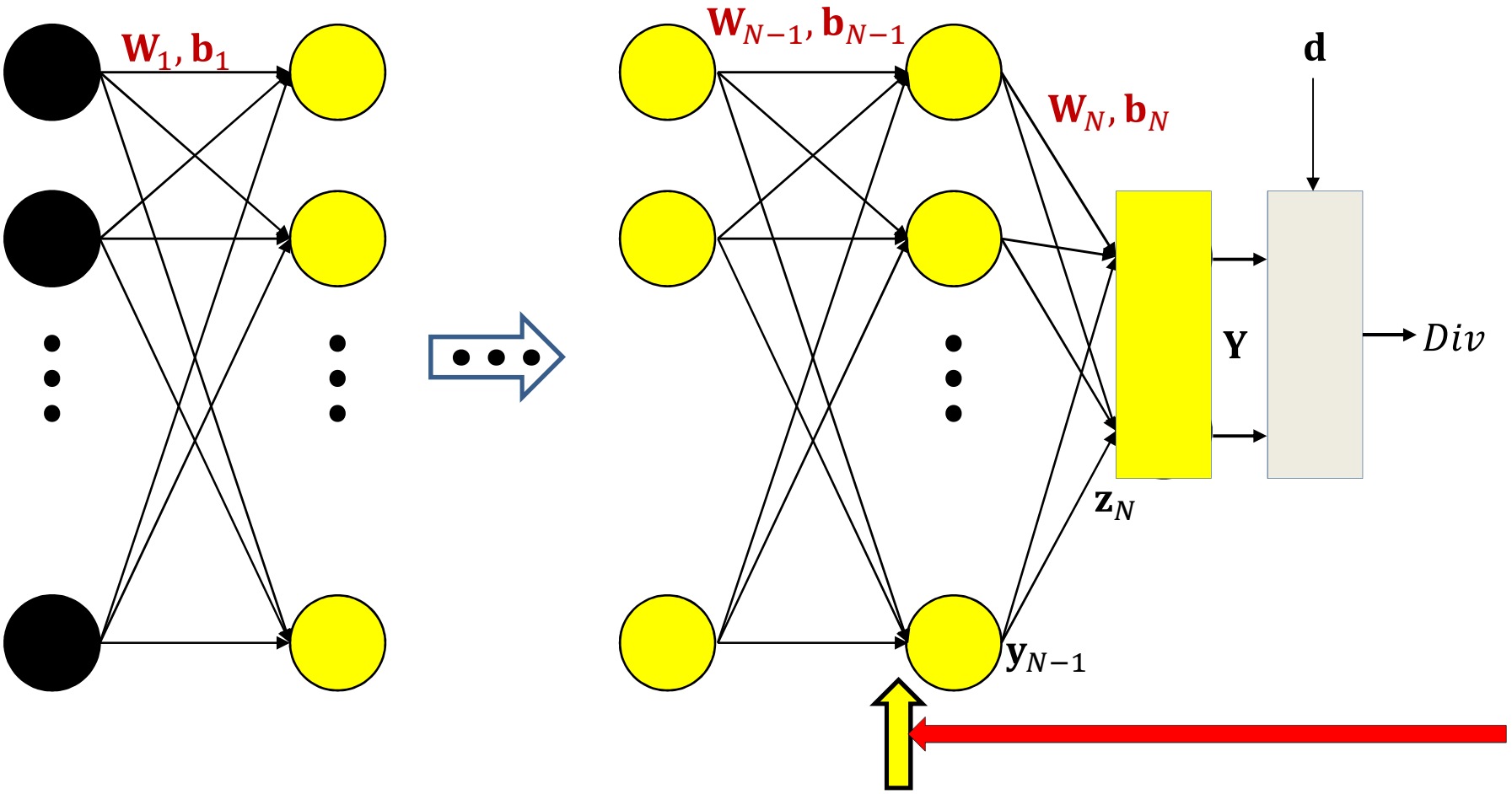


$$\nabla_{\mathbf{y}_{N-1}} Div = \nabla_{\mathbf{z}_N} Div \, \mathbf{W}_N$$

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

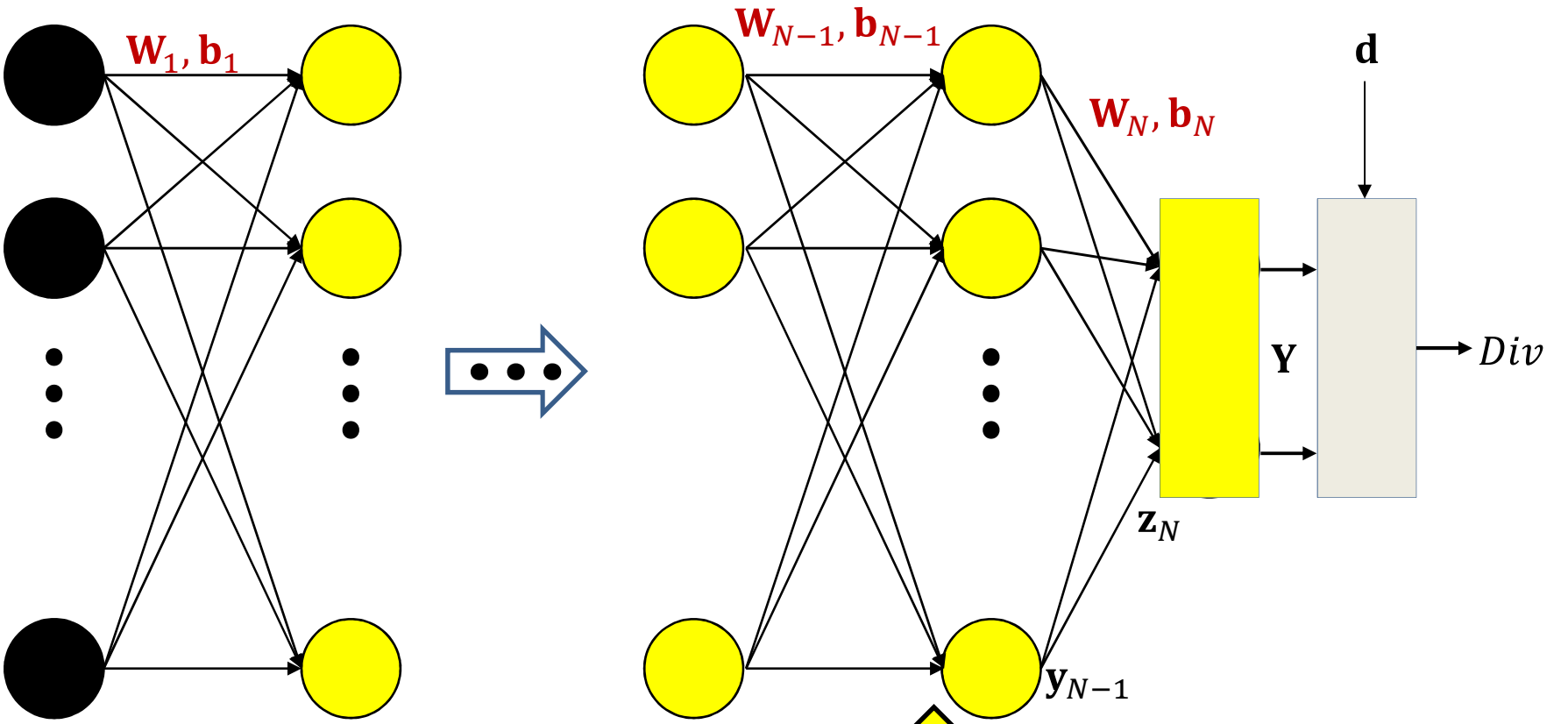
$$\nabla_{\mathbf{b}_N} Div = \nabla_{\mathbf{z}_N} Div$$

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

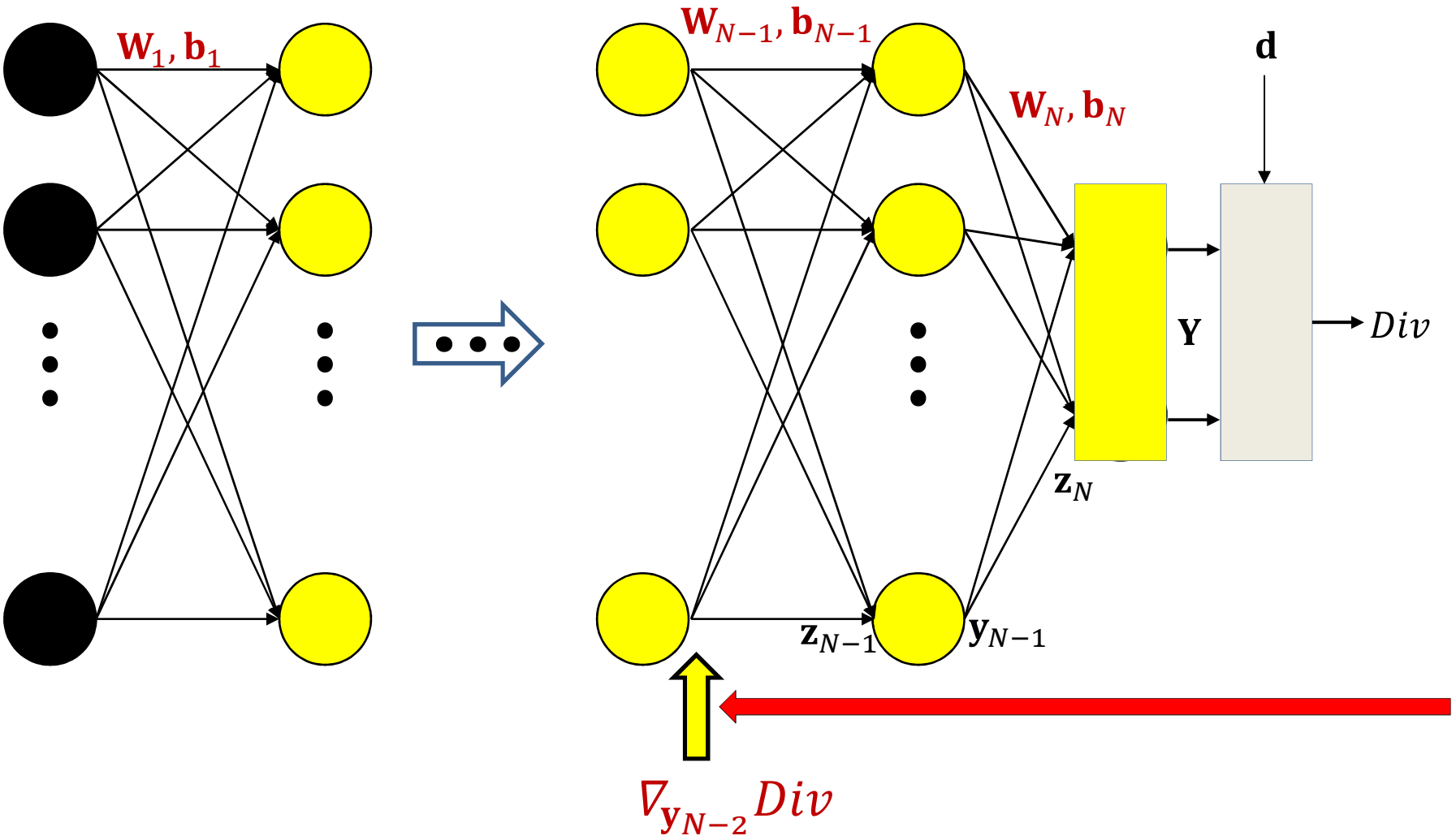


$$\nabla_{\mathbf{z}_{N-1}} Div = \nabla_{\mathbf{y}_{N-1}} Div \, J_{\mathbf{y}_{N-1}}(\mathbf{z}_{N-1})$$

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

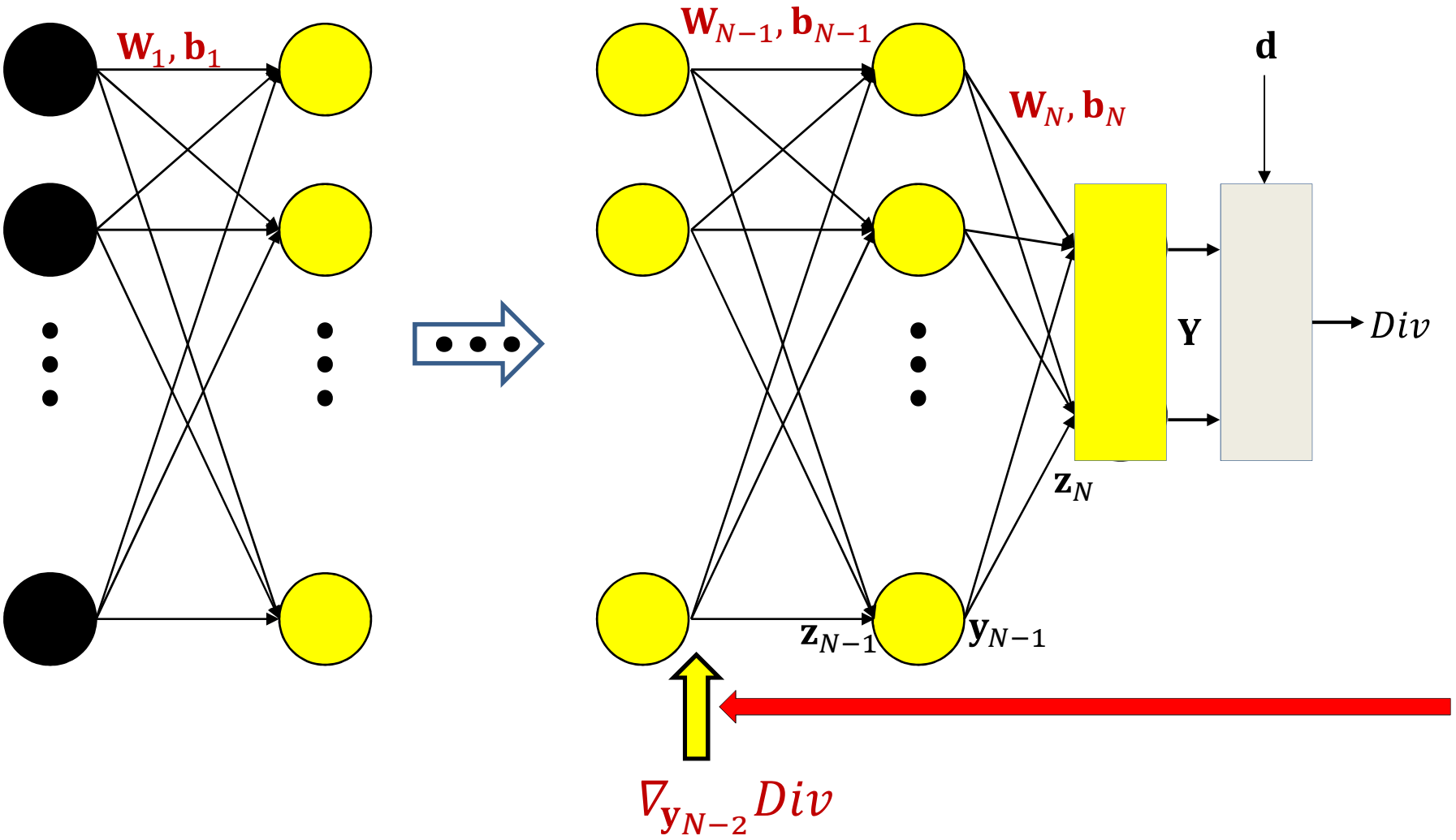
The Jacobian will be a diagonal matrix for scalar activations

The backward pass



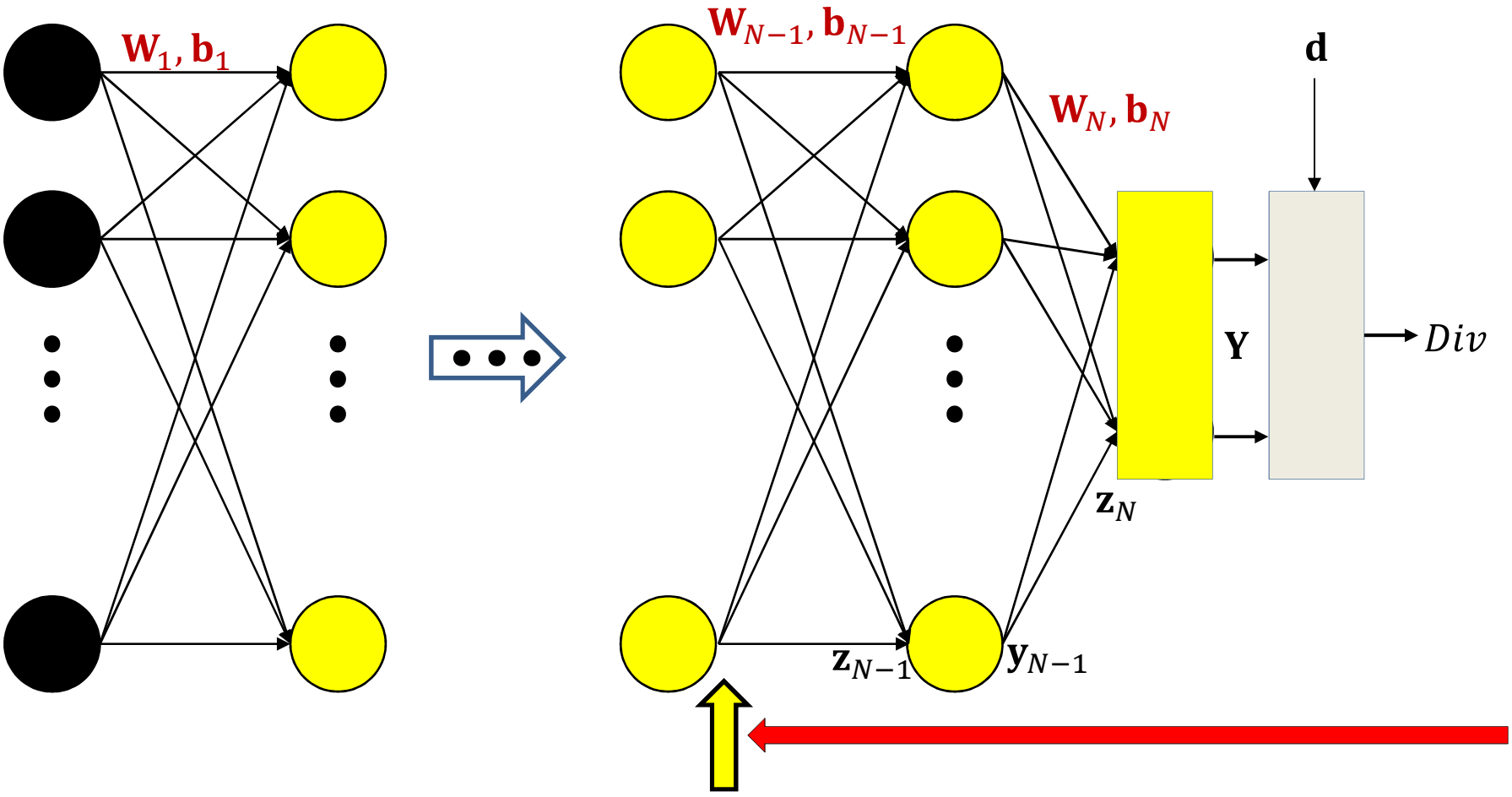
$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div . \nabla_{\mathbf{y}_{N-2}} \mathbf{z}_{N-1}$$

The backward pass



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

The backward pass

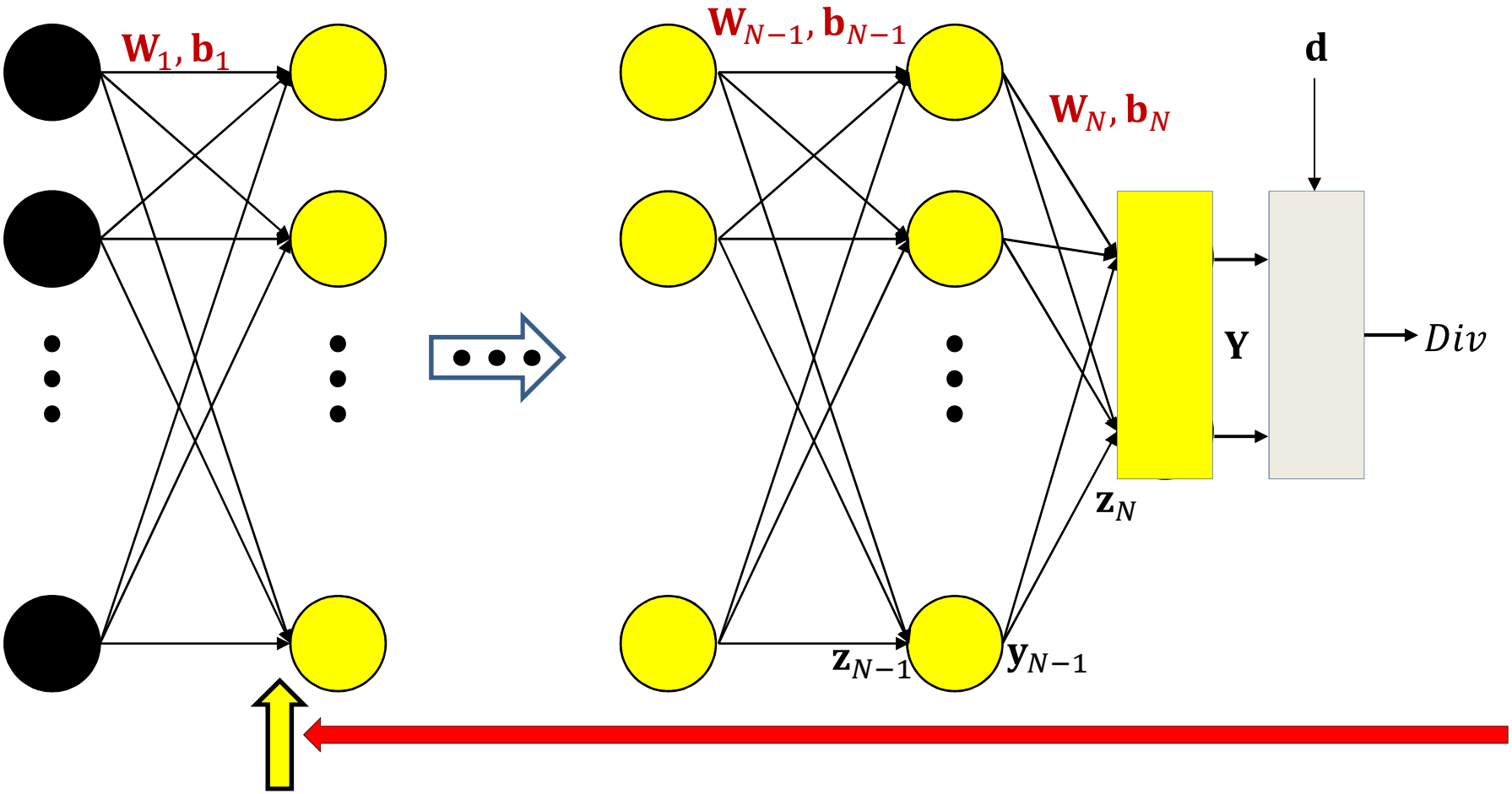


$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div \, \mathbf{W}_{N-1}$$

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

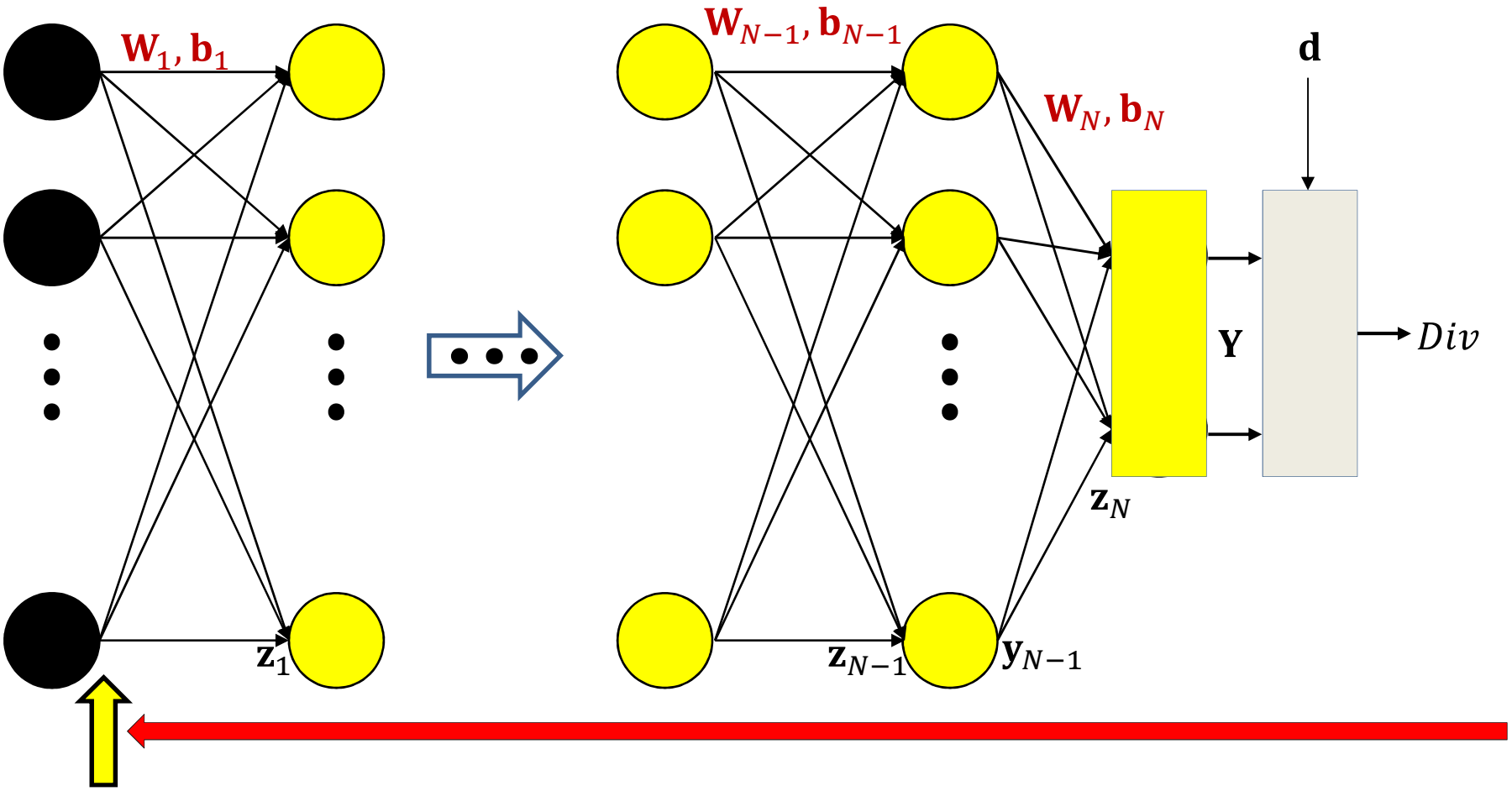
$$\nabla_{\mathbf{b}_{N-1}} Div = \nabla_{\mathbf{z}_{N-1}} Div$$

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