# Neural Networks
# Learning the network: Part 3

11-785, Fall 2025
Lecture 5

Attendance: @382

# Training neural nets through Empirical Risk Minimization: Problem Setup

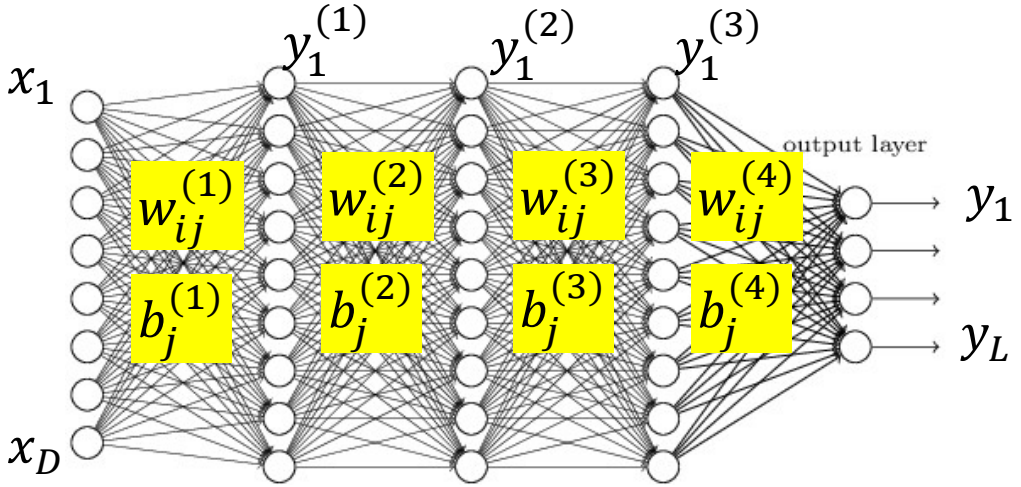
- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$

- The divergence on the i[th] instance is $div(Y_i, d_i)$
  - $Y_i = f(X_i; W)$
- The loss (empirical risk)

$$Loss(W) = \frac{1}{T}\sum_i div(Y_i, d_i)$$

- Minimize $Loss$ w.r.t $\left\{w_{ij}^{(k)}, b_j^{(k)}\right\}$ using gradient descent

# Notation




- The input layer is the 0th layer
- We will represent the output of the i-th perceptron of the kth layer as $y_i^{(k)}$
  - **Input to network:** $y_i^{(0)} = x_i$
  - **Output of network:** $y_i = y_i^{(N)}$
- We will represent the weight of the connection between the i-th unit of the k-1th layer and the jth unit of the k-th layer as $w_{ij}^{(k)}$
  - The bias to the jth unit of the k-th layer is $b_j^{(k)}$

# Recap: Gradient Descent Algorithm

To minimize any function Loss(W) w.r.t W

- Initialize:
  - $W^0$
  - $k = 0$
- do
  - $W^{k+1} = W^k - \eta^k \nabla Loss(W^k)^T$
  - $k = k + 1$
- while $\left|Loss\left(W^k\right) - Loss\left(W^{k-1}\right)\right| > \varepsilon$

# Recap: Gradient Descent Algorithm

- In order to minimize $L(W)$ w.r.t. $W$
- Initialize:
  - $W^0$
  - $k = 0$
- do
  - For every component $i$
    - $W_i^{k+1} = W_i^k - \eta^k \frac{\partial L}{\partial W_i}$

    Explicitly stating it by component
  - $k = k + 1$
- while $\left| L\left(W^k\right) - L\left(W^{k-1}\right) \right| > \varepsilon$

# Training Neural Nets through Gradient Descent

**Total training Loss:**

$$Loss = \frac{1}{T}\sum_{t} Div(\boldsymbol{Y_t}, \boldsymbol{d_t})$$

- Gradient descent algorithm:
- Initialize all weights and biases $\left\{w_{ij}^{(k)}\right\}$
  - Using the extended notation: the bias is also a weight
- Do:
  - For every layer $k$ for all $i, j$, update:
    - $w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{dLoss}{dw_{i,j}^{(k)}}$
- Until $Loss$ has converged

Assuming the bias is also represented as a weight

# Training Neural Nets through Gradient Descent

**Total training Loss:**

$$Loss = \frac{1}{T}\sum_{t} Div(\boldsymbol{Y_t}, \boldsymbol{d_t})$$

- Gradient descent algorithm:
- Initialize all weights and biases $\left\{w_{ij}^{(k)}\right\}$
  - Using the extended notation: the bias is also a weight
- Do:
  - For every layer $k$ for all $i, j$, update:
    - $w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{dLos}{dw_{i,j}^{(k)}}$
- Until $Loss$ has converged

Assuming the bias is also represented as a weight

# The derivative

**Total training Loss:**

$$Loss = \frac{1}{T}\sum_{t} Div(\boldsymbol{Y_t}, \boldsymbol{d_t})$$

- Computing the derivative

**Total derivative:**

$$\frac{dLoss}{dw_{i,j}^{(k)}} = \frac{1}{T}\sum_{t} \frac{dDiv(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$$

# The derivative

**Total training Loss:**

$$Loss = \frac{1}{T}\sum_{t} Div(\boldsymbol{Y_t}, \boldsymbol{d_t})$$

- Computing the derivative

**Total derivative:**

$$\frac{dLoss}{dw_{i,j}^{(k)}} = \frac{1}{T}\sum_{t} \frac{dDiv(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$$

- So we must first figure out how to compute the derivative of divergences of individual training inputs

# Calculus Refresher: Basic rules of calculus

For any differentiable function

$$y = f(x)$$

with derivative

$$\frac{dy}{dx}$$

the following must hold for sufficiently small $\Delta x$ ⇒ $\Delta y \approx \frac{dy}{dx}\Delta x$

# Calculus Refresher: Basic rules of calculus

For any differentiable function

$$y = f(x)$$

with derivative

$$\frac{dy}{dx}$$

the following must hold for sufficiently small $\Delta x$ ⇒ $\Delta y \approx \frac{dy}{dx}\Delta x$


Introducing the "influence" diagram: x influences y

# Calculus Refresher: Basic rules of calculus

For any differentiable function

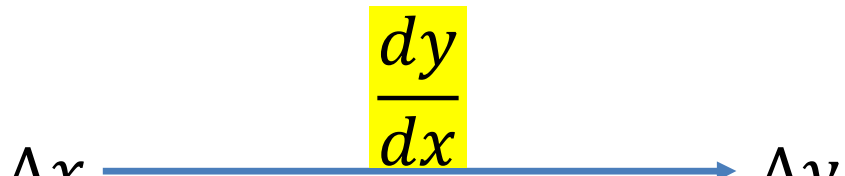
$$y = f(x)$$

with derivative

$$\frac{dy}{dx}$$

the following must hold for sufficiently small $\Delta x$ ⇒ $\Delta y \approx \frac{dy}{dx}\Delta x$


Introducing the "influence" diagram: x influences y




$$\Delta y = \frac{dy}{dx}\Delta x$$

The derivative graph: The edge carries the derivative.

Node and edge weights multiply

# Calculus Refresher: Basic rules of calculus

For any differentiable function

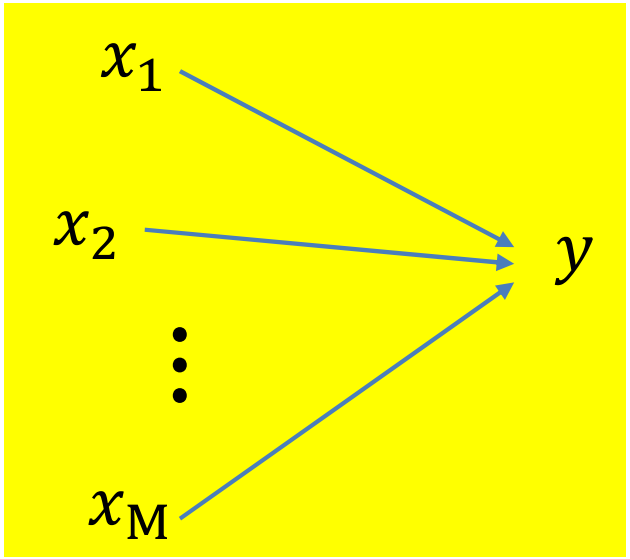
$$y = f(x_1, x_2, \ldots, x_M)$$

What is the influence diagram relating $x_1, x_2, \ldots, x_M$ and $y$?

# Calculus Refresher: Basic rules of calculus

For any differentiable function

$$y = f(x_1, x_2, \ldots, x_M)$$




The derivative diagram?

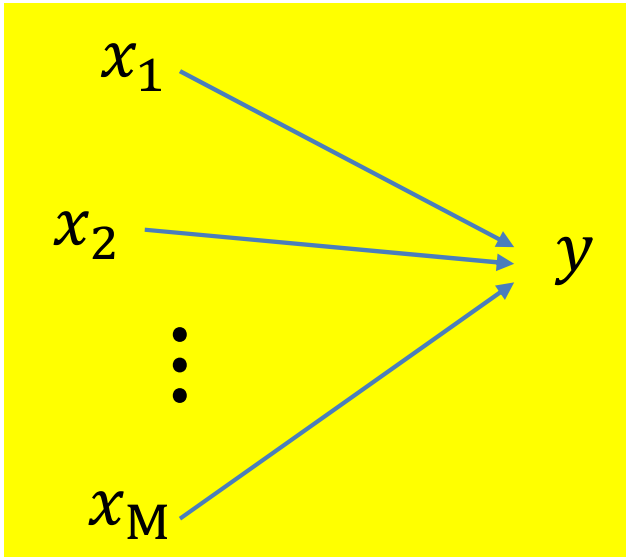
# Calculus Refresher: Basic rules of calculus

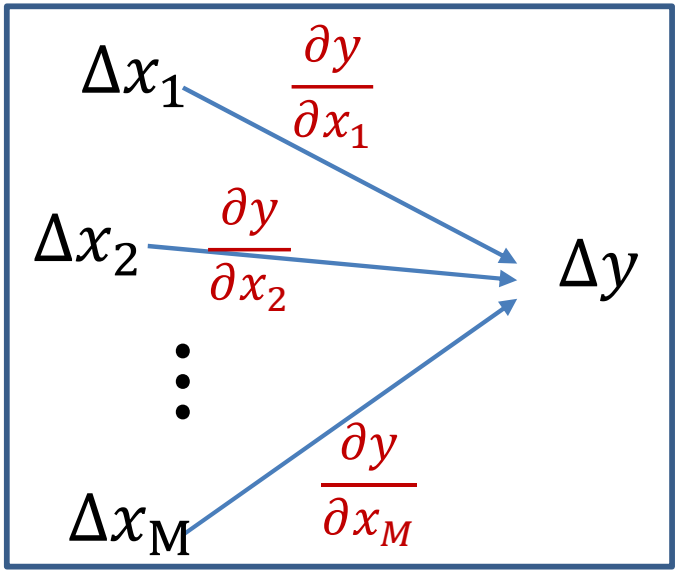
For any differentiable function

$$y = f(x_1, x_2, \ldots, x_M)$$

with partial derivatives

$$\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_M}$$



# Calculus Refresher: Basic rules of calculus

For any differentiable function

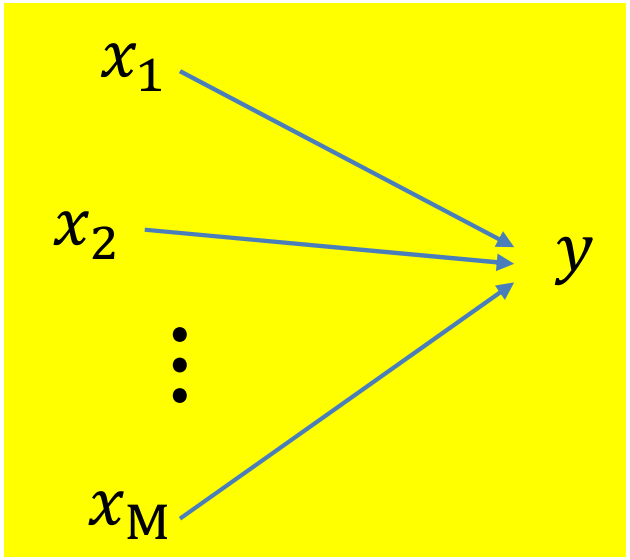
$$y = f(x_1, x_2, \ldots, x_M)$$

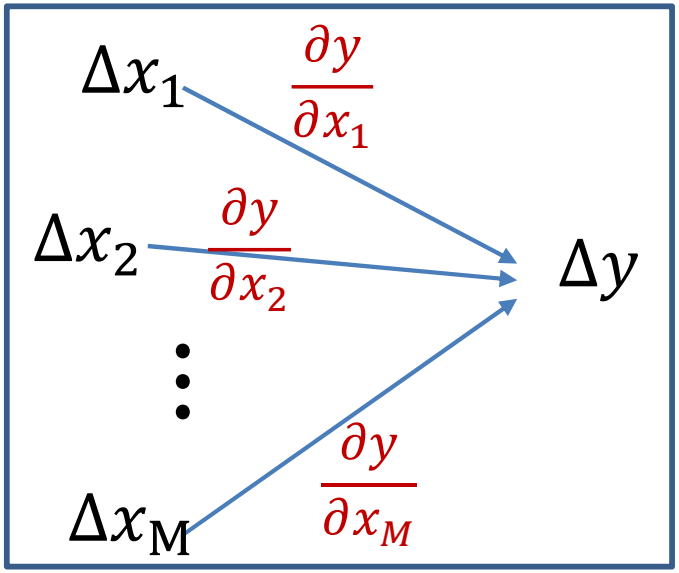
with partial derivatives

$$\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_M}$$

the following must hold for sufficiently small $\Delta x_1, \Delta x_2, \ldots, \Delta x_M$

$$\Delta y \approx \frac{\partial y}{\partial x_1}\Delta x_1 + \frac{\partial y}{\partial x_2}\Delta x_2 + \cdots + \frac{\partial y}{\partial x_M}\Delta x_M$$



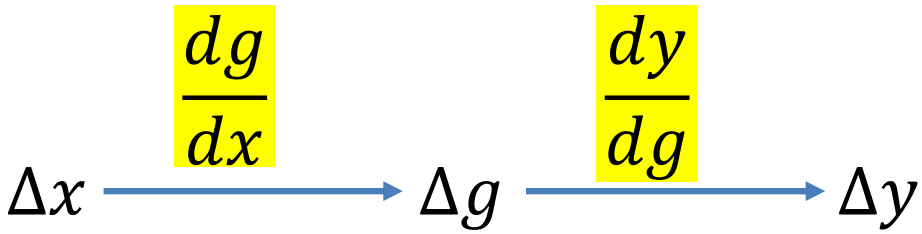
# Calculus Refresher: Chain rule

For any nested function $y = f(g(x))$

# Calculus Refresher: Chain rule

For any nested function $y = f(g(x))$

$$\frac{dy}{dx} = \frac{dy}{dg(x)} \frac{dg(x)}{dx}$$


$$\Delta y = \frac{dy}{dg(x)} \frac{dg(x)}{dx} \Delta x$$

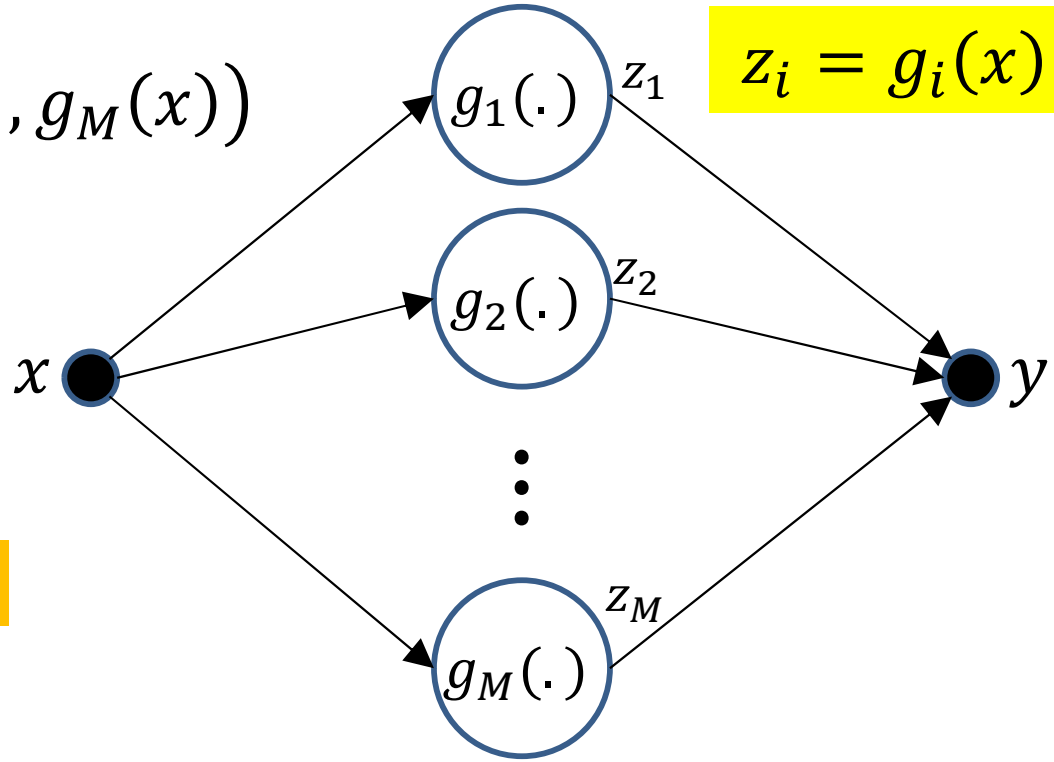
# Distributed Chain Rule: Influence Diagram

$$y = f\big(g_1(x), g_1(x), \dots, g_M(x)\big)$$

Shorthand: $z_i = g_i(x)$

# Distributed Chain Rule: Influence Diagram

$$y = f\big(g_1(x), g_1(x), \dots, g_M(x)\big)$$

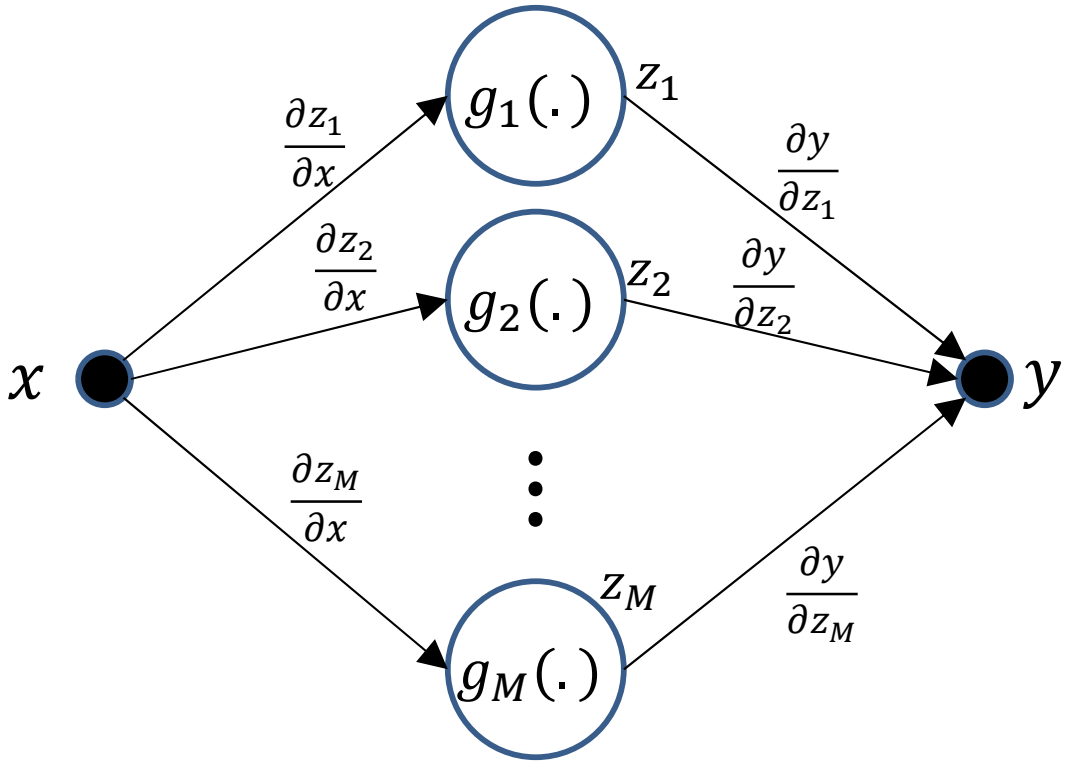



Derivative rule?

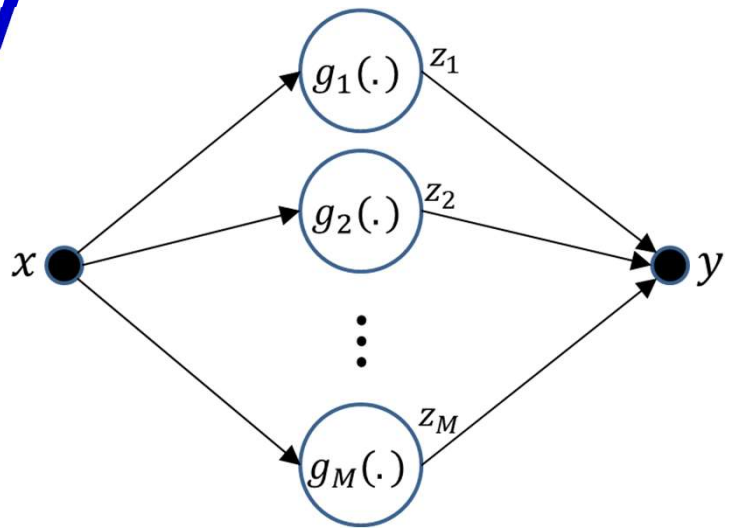
- $x$ affects $y$ through each of $g_1 \dots g_M$

# Distributed Chain Rule: Influence Diagram

$$y = f\big(g_1(x), g_1(x), \ldots, g_M(x)\big)$$



# Calculus Refresher: Chain rule summary

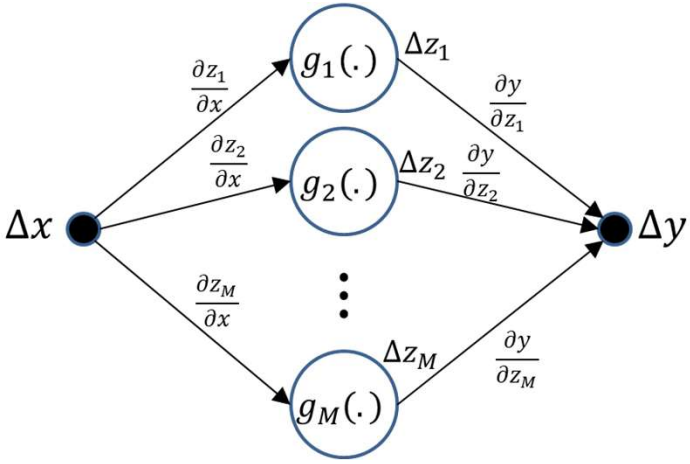



For $y = f(z_1, z_2, \ldots, z_M)$

where $z_i = g_i(x)$

$$\Delta y = \sum_i \frac{\partial y}{\partial z_i} \Delta z_i \qquad \Delta z_i = \frac{dz_i}{dx} \Delta x$$

$$\frac{dy}{dx} = \frac{\partial y}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial y}{\partial z_2}\frac{dz_2}{dx} + \cdots + \frac{\partial y}{\partial z_M}\frac{dz_M}{dx}$$



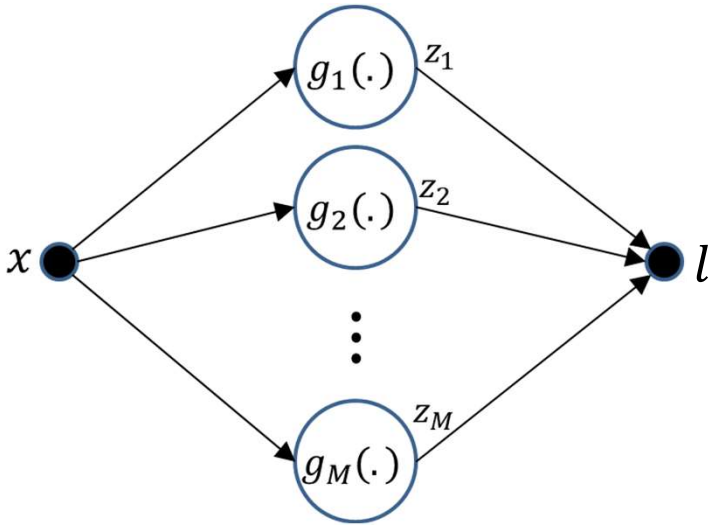
# Calculus Refresher: Chain rule summary

For any nested function $l = f(y)$ where $y = g(z)$

$$\frac{dl}{dz} = \frac{dl}{dy}\frac{dy}{dz}$$

For $l = f(z_1, z_2, \ldots, z_M)$

where $z_i = g_i(x)$




$$\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx} + \cdots + \frac{\partial l}{\partial z_M}\frac{dz_M}{dx}$$

# Our problem for today

- How to compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y},\boldsymbol{d})}{dw_{i,j}^{(k)}}$ for a single data instance

# Poll 1 : @383, @385

1. The chain rule of derivatives can be derived from the basic definition of derivatives, dy = derivative * dx, true or false
   - True
   - False

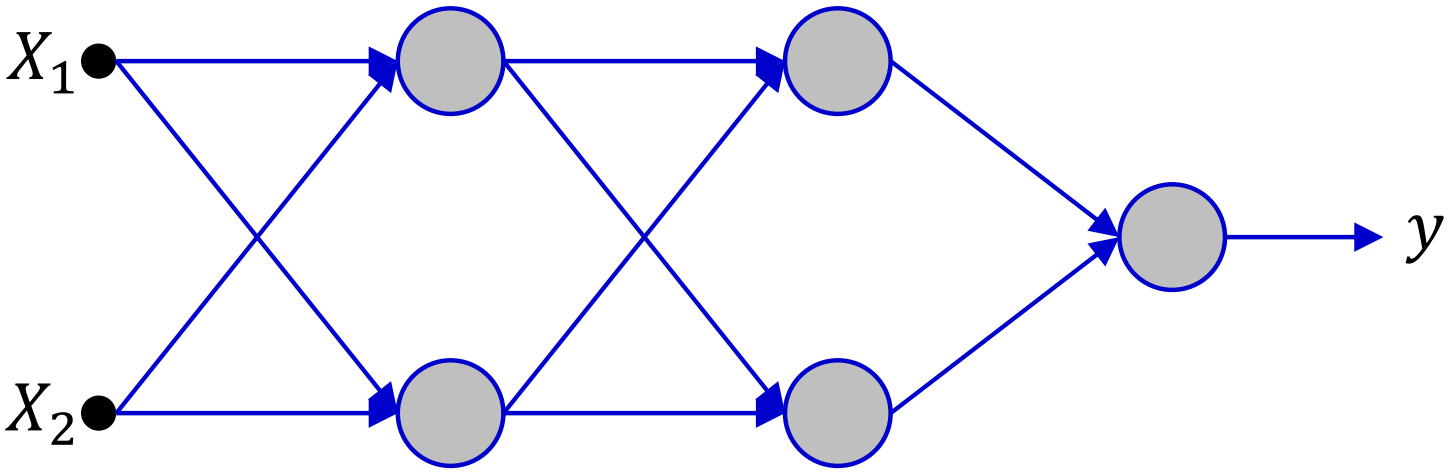
2. Which of the following is true of the “influence diagram”
   - It graphically shows all paths (and variables) through which one variable influences the other
   - The derivative of the influenced (outcome) variable with respect to the influencer (input) variable must be summed over all outgoing paths from the influencer variable

# Poll 1

1. The chain rule of derivatives can be derived from the basic definition of derivatives, dy = derivative * dx, true or false
   - True **(correct)**
   - False

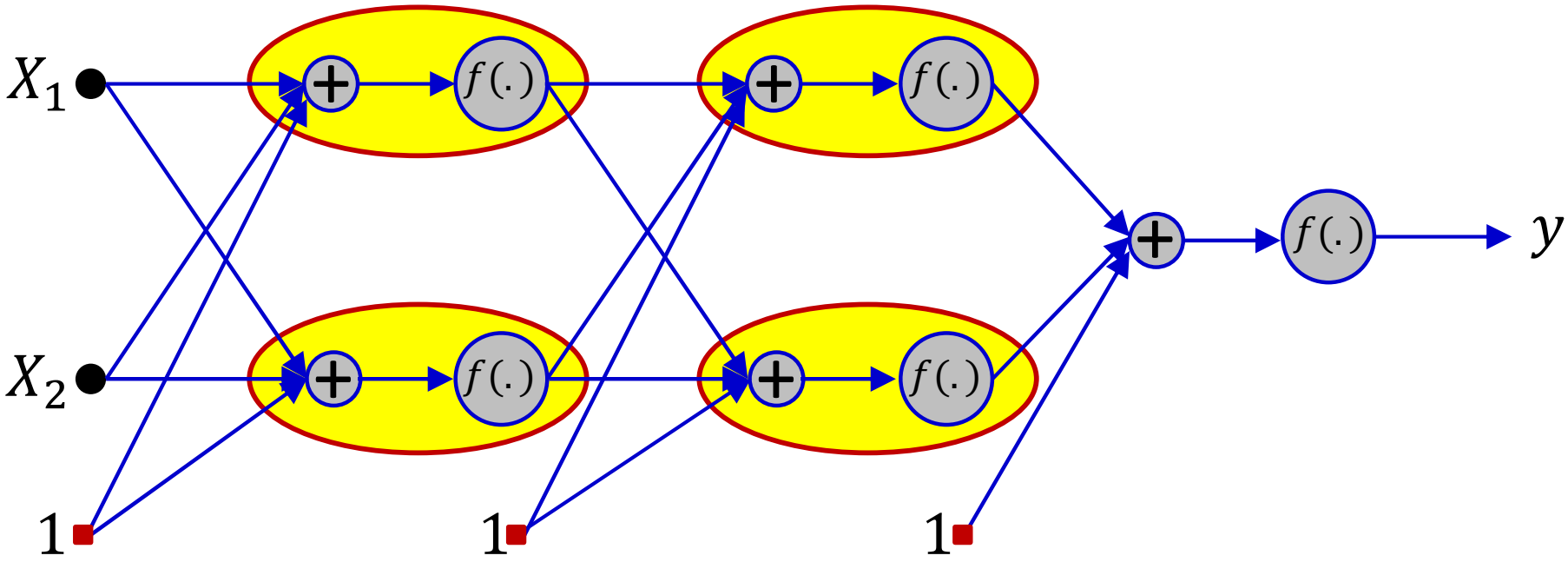
2. Which of the following is true of the "influence diagram"
   - It graphically shows all paths (and variables) through which one variable influences the other **(true)**
   - The derivative of the influenced (outcome) variable with respect to the influencer (input) variable must be summed over all outgoing paths from the influencer variable **(true)**

# A first closer look at the network



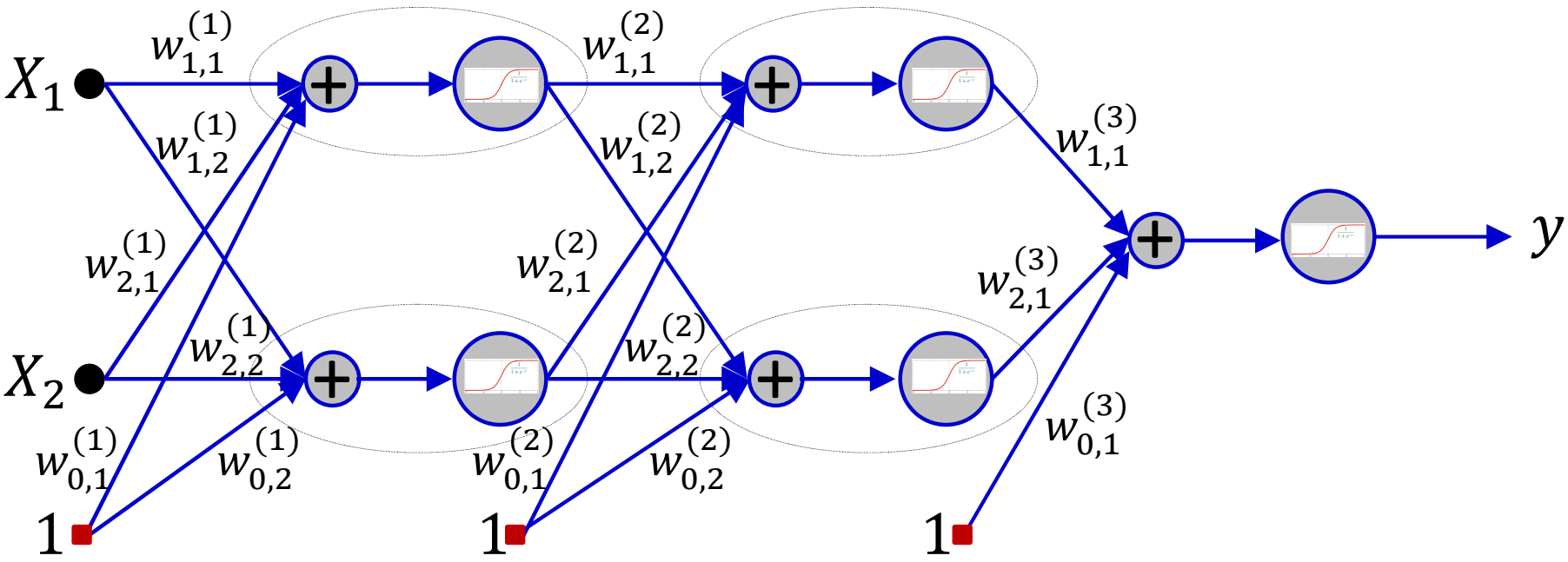

- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs

# A first closer look at the network



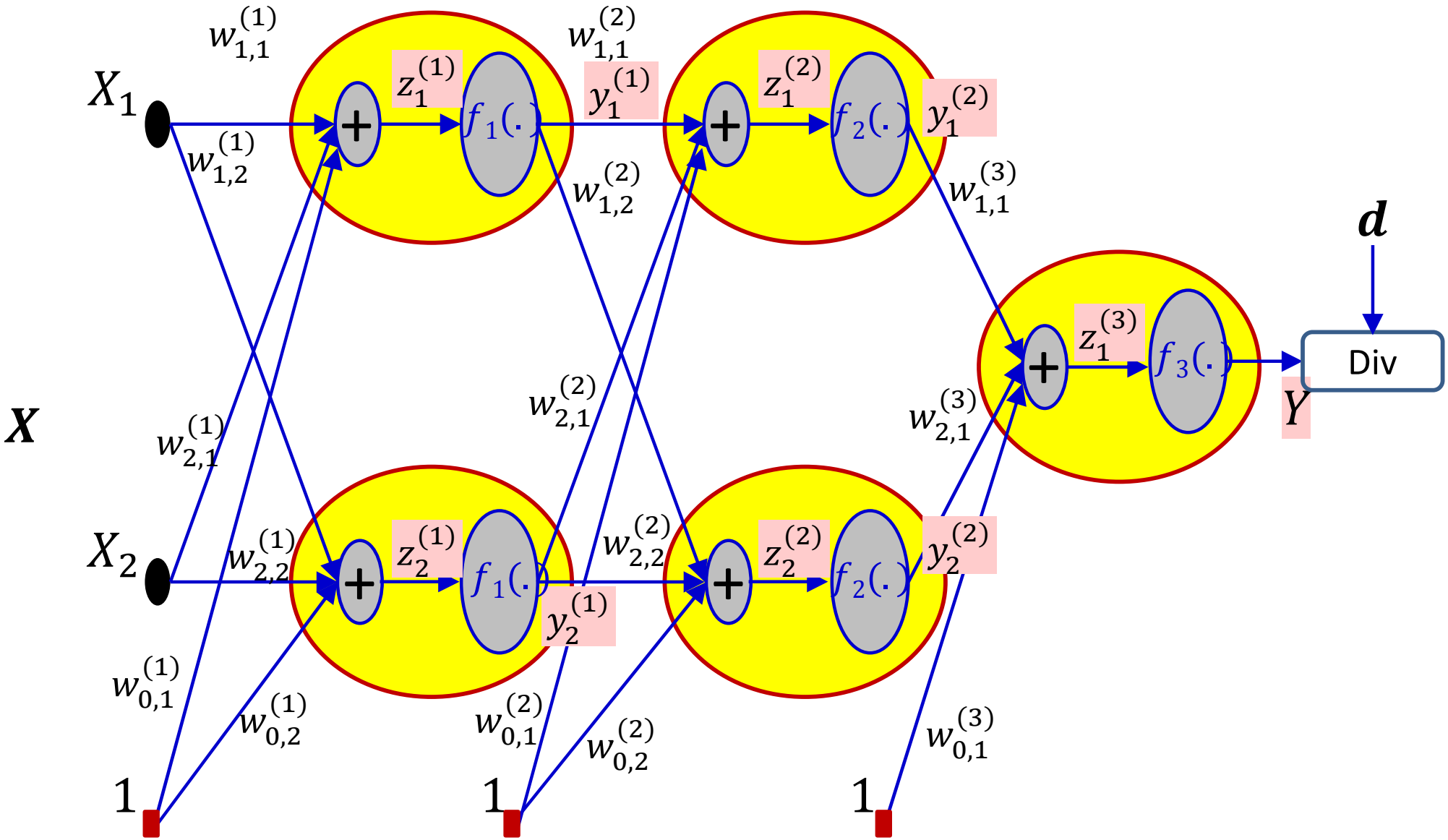

- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Explicitly separating the affine function of inputs from the activation

# A first closer look at the network

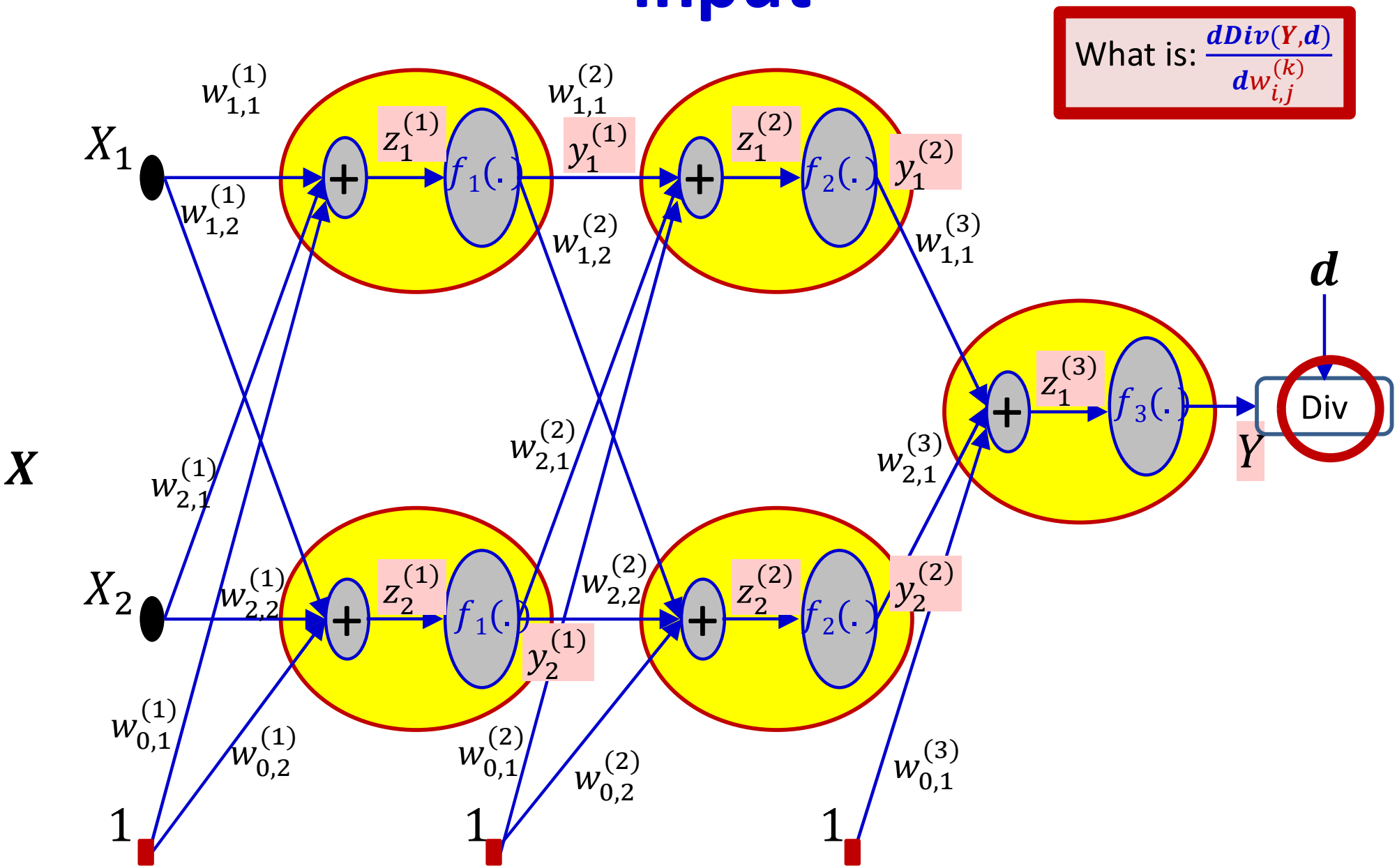



- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Expanded **with all weights shown**
- Let's label the other variables too…

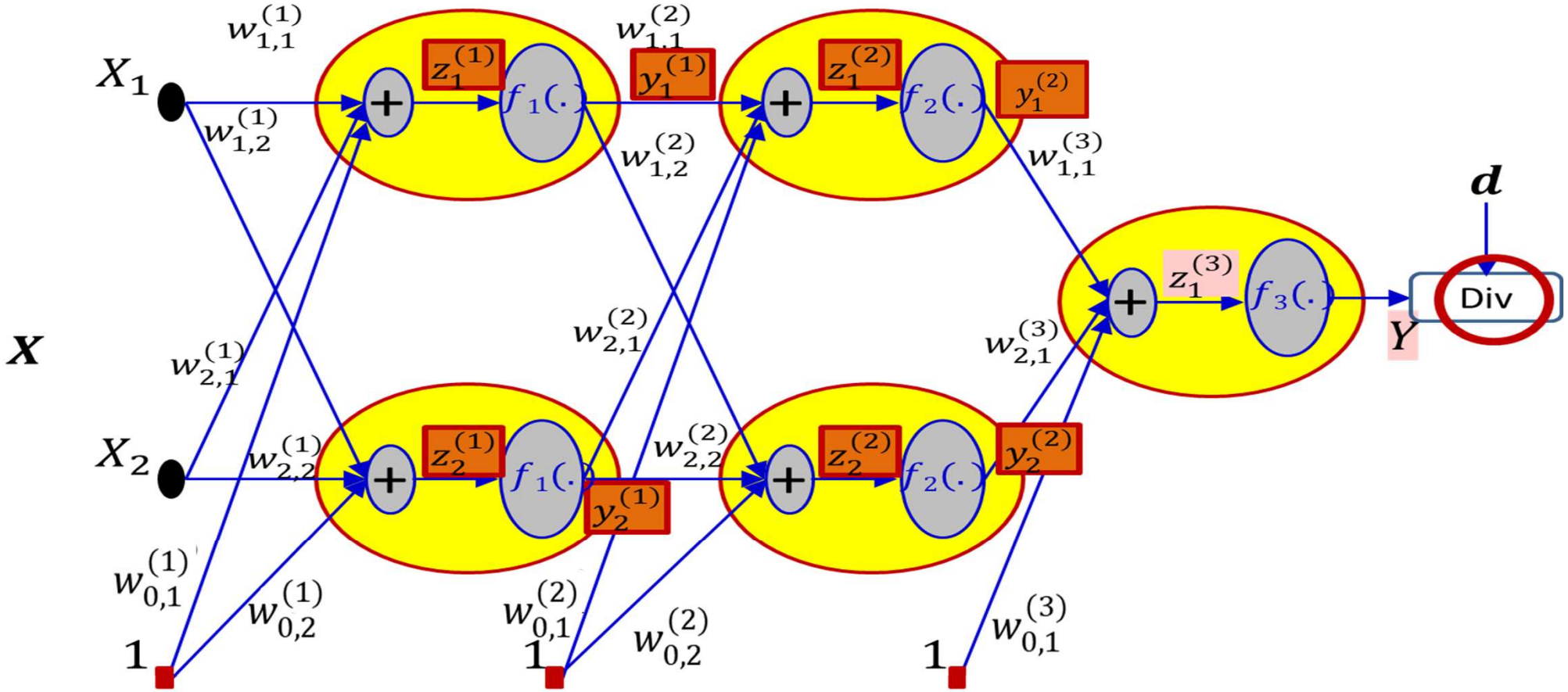
# Computing the derivative for a *single* input



# Computing the derivative for a *single* input



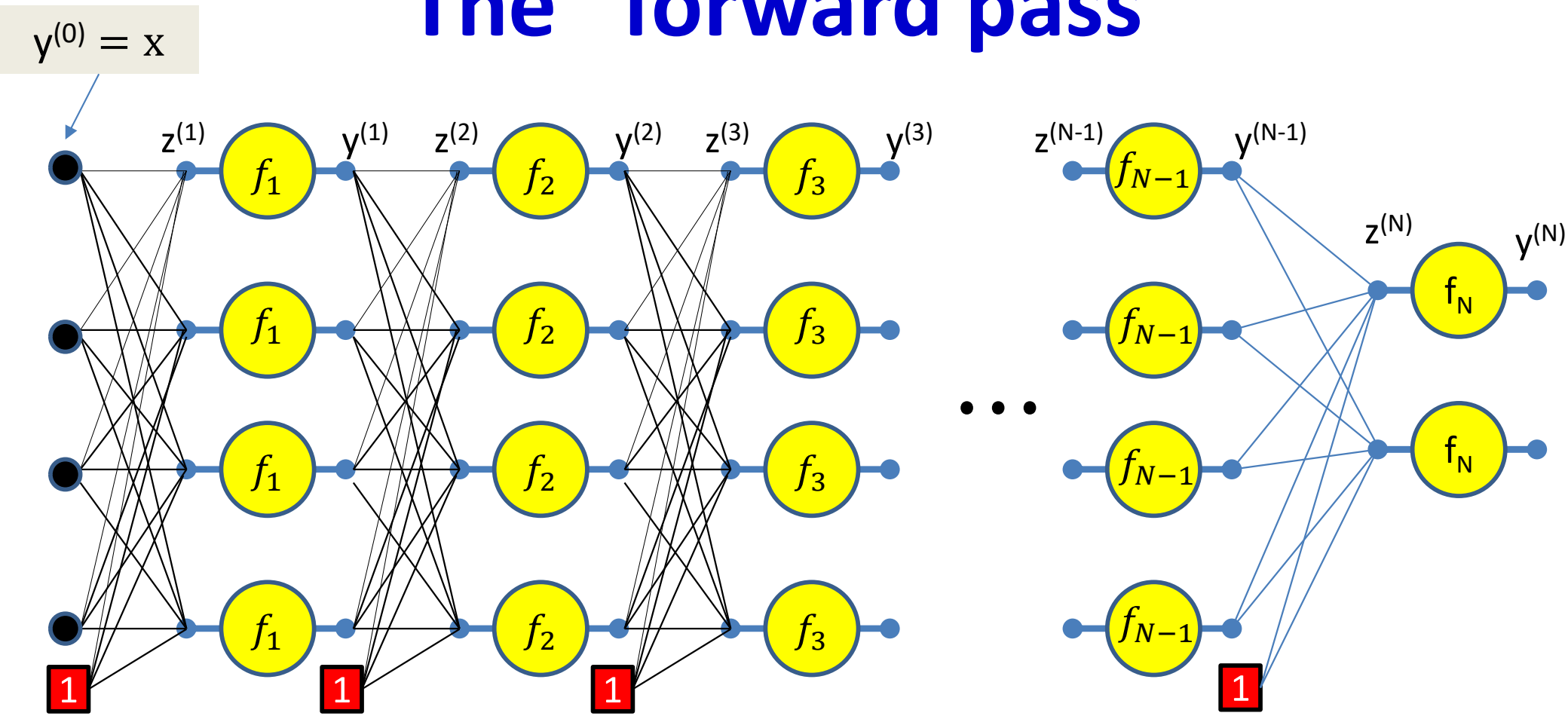
# Computing the gradient




- Note: computation of the derivative $\frac{\boldsymbol{dDiv(Y,d)}}{\boldsymbol{dw_{i,j}^{(k)}}}$ requires

  intermediate and final output values of the network in response to the input

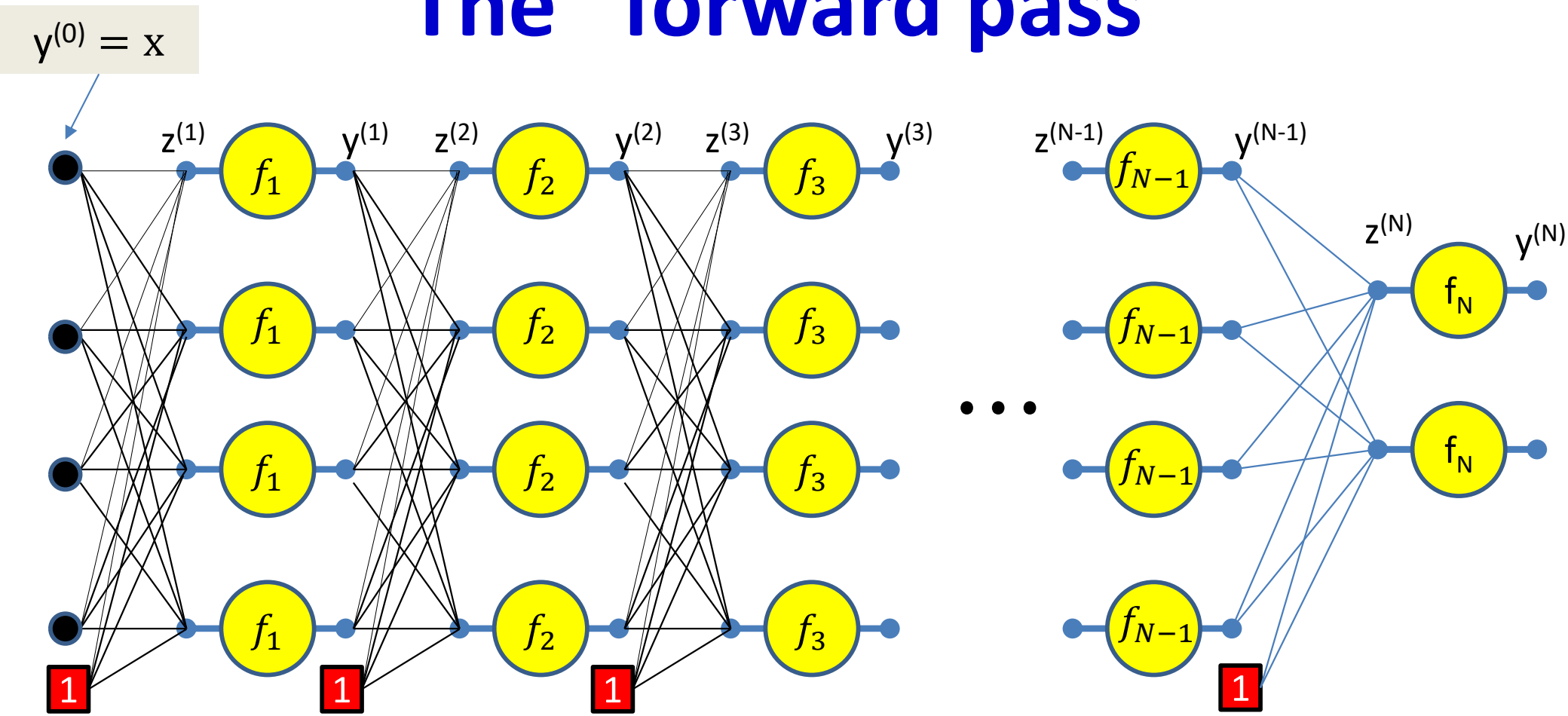
# The "forward pass"




We will refer to the process of computing the output from an input as the *forward pass*

We will illustrate the forward pass in the following slides

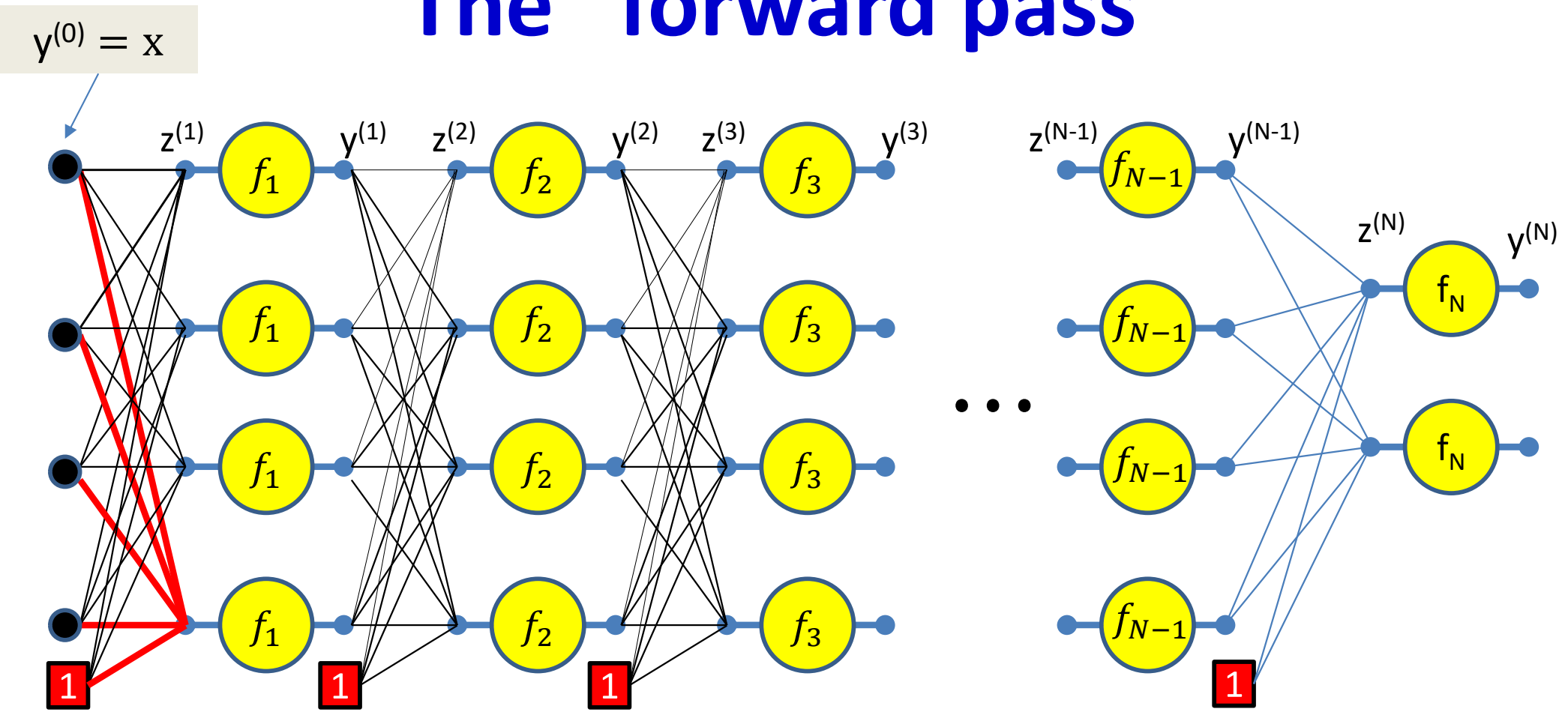
# The "forward pass"




Setting $y_i^{(0)} = x_i$ for notational convenience

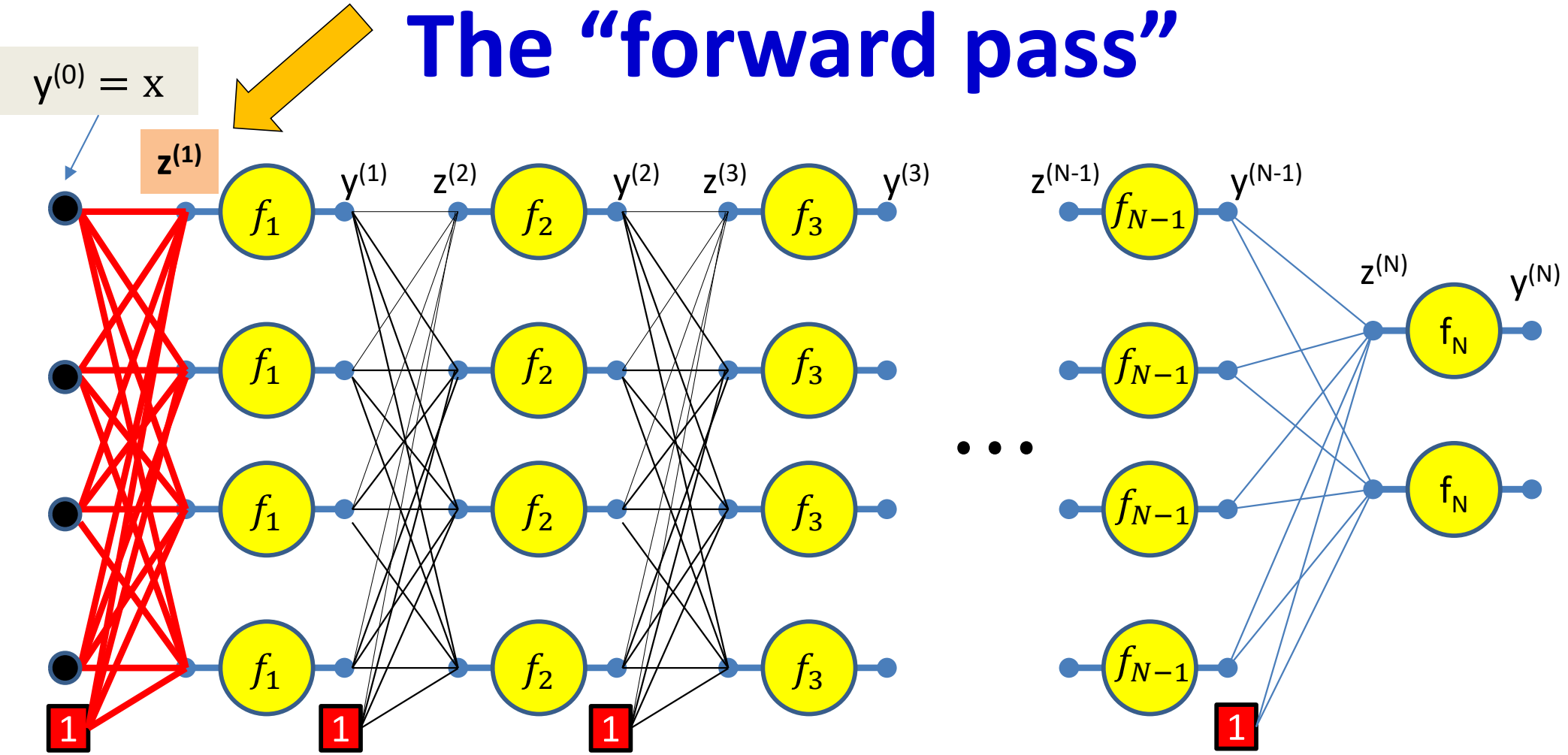
Assuming $w_{0j}^{(k)} = b_j^{(k)}$ and $y_0^{(k)} = 1$ -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases

# The "forward pass"

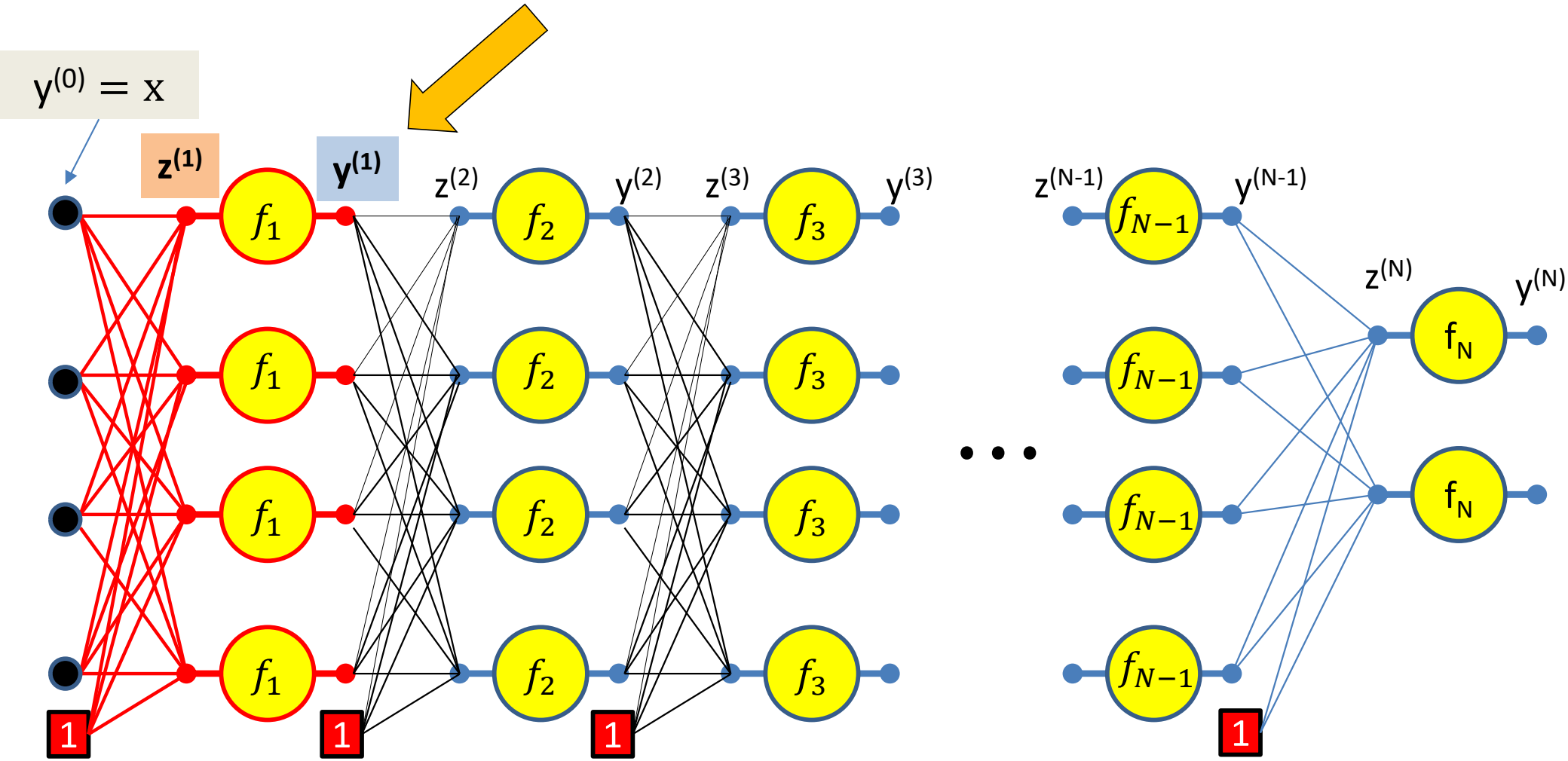



$$z_1^{(1)} = \sum_i w_{i1}^{(1)} y_i^{(0)}$$

# The "forward pass"

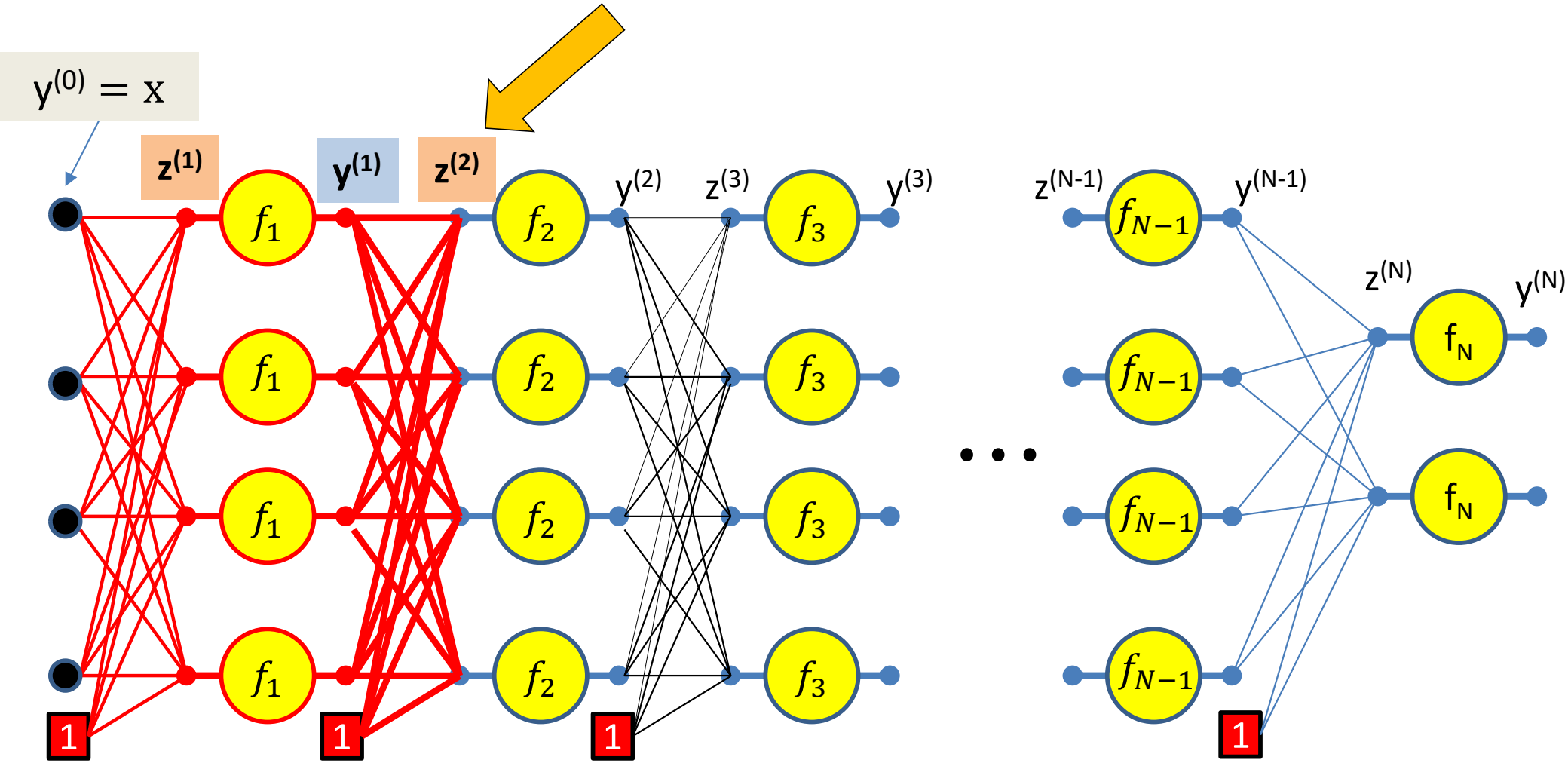



$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$




$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

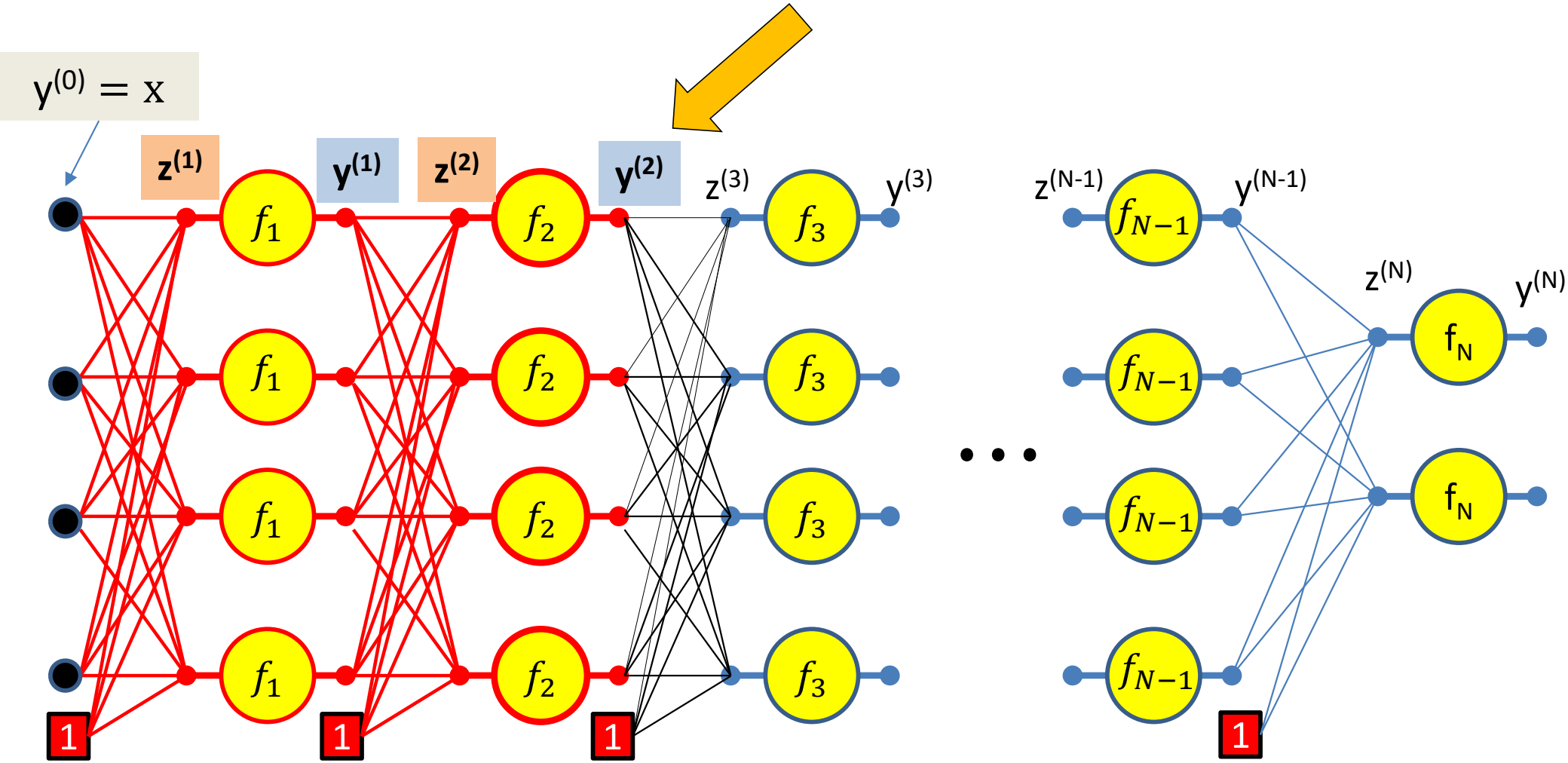
$$y_j^{(1)} = f_1\left(z_j^{(1)}\right)$$




$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

$$y_j^{(1)} = f_1\left(z_j^{(1)}\right)$$

$$z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)}$$

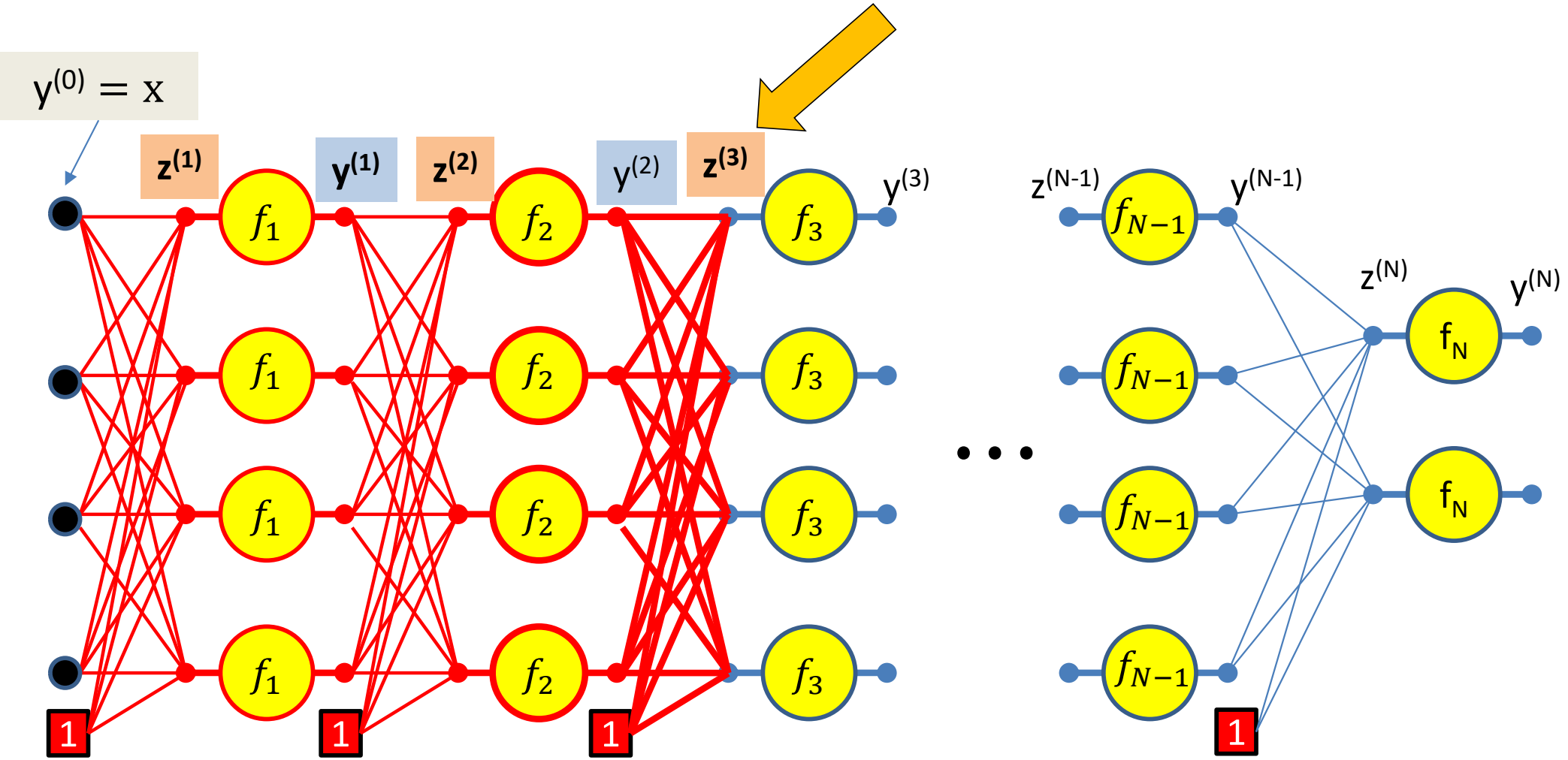



$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

$$y_j^{(1)} = f_1\left(z_j^{(1)}\right)$$

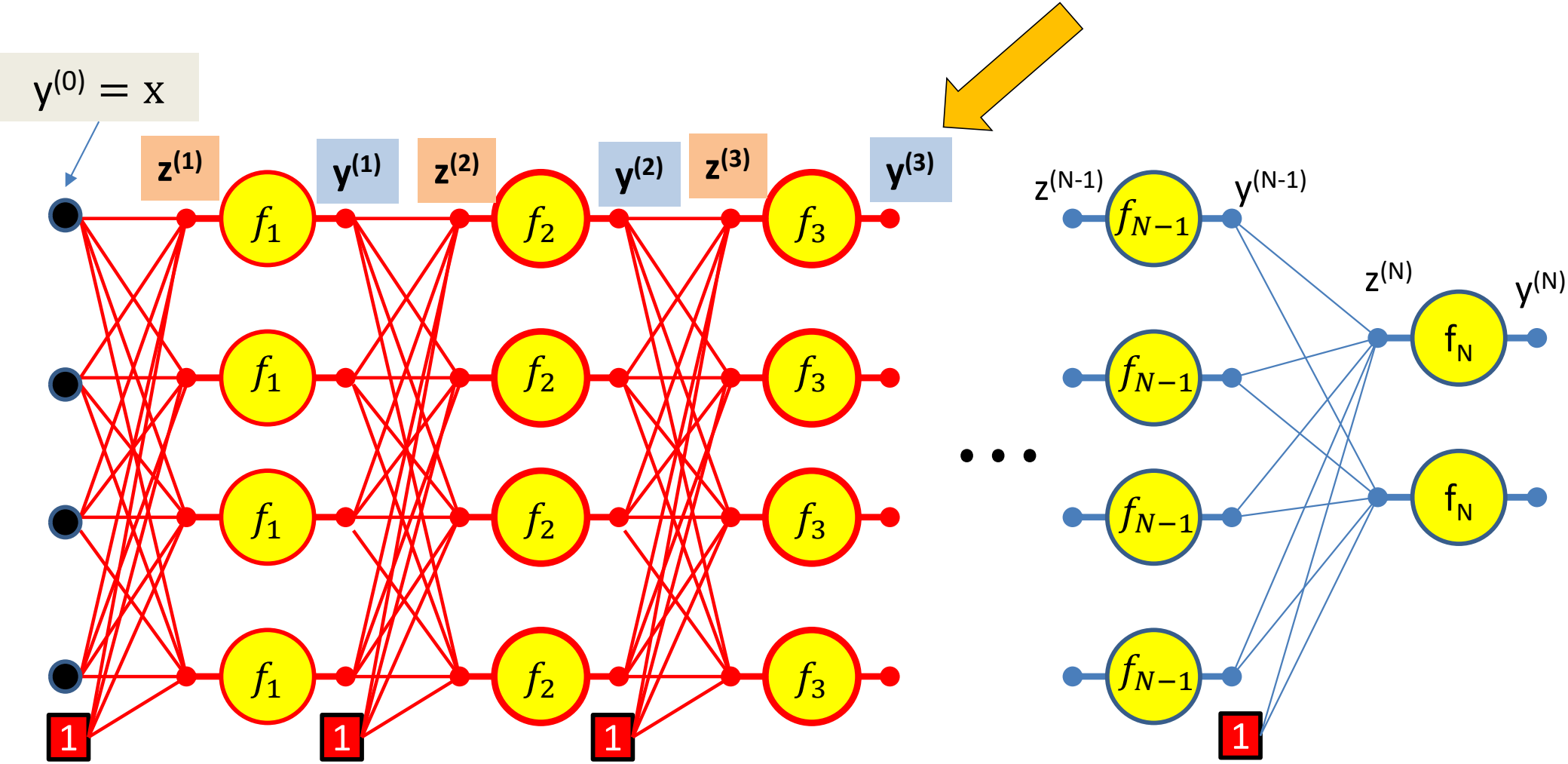
$$z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)}$$

$$y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$



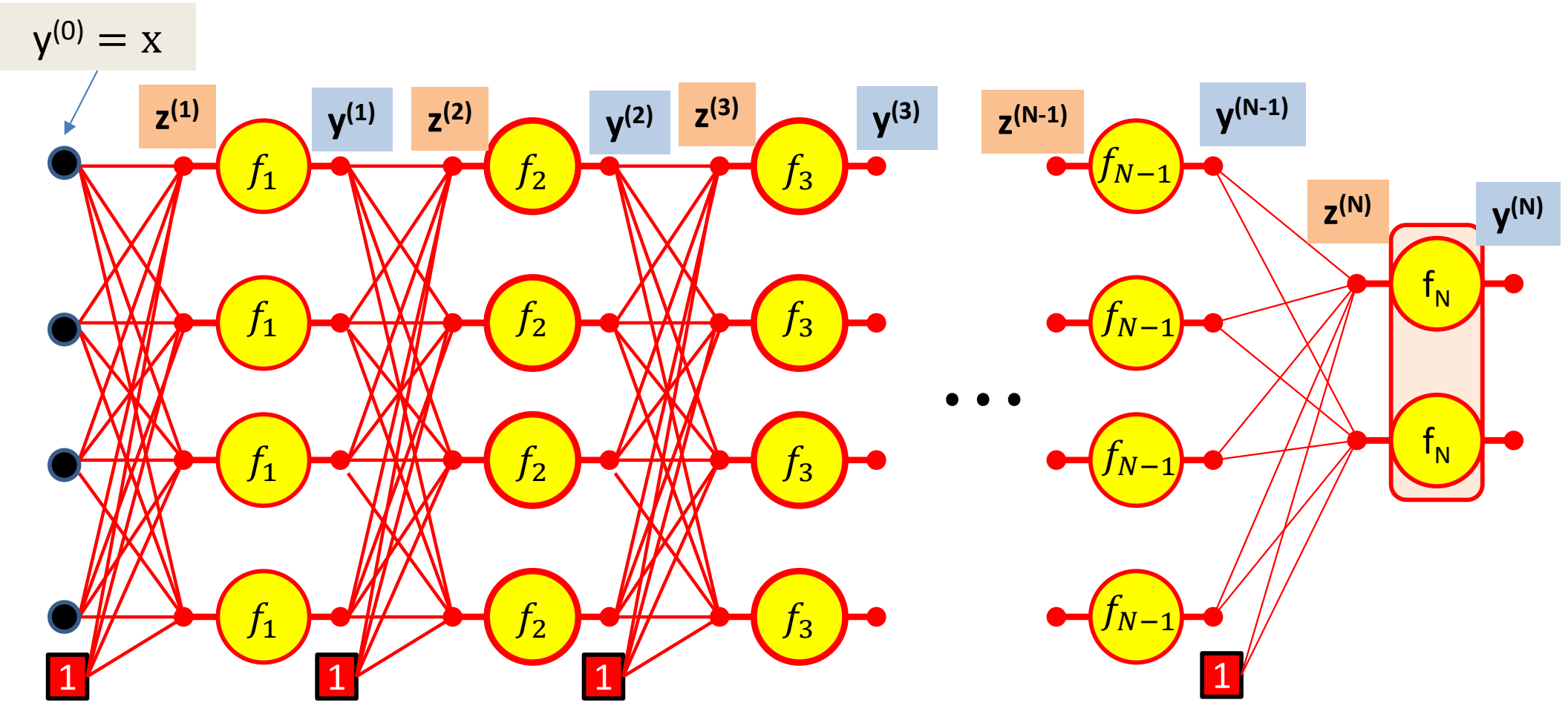

$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \qquad y_j^{(1)} = f_1\left(z_j^{(1)}\right) \qquad z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \qquad y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$

$$z_j^{(3)} = \sum_i w_{ij}^{(3)} y_i^{(2)}$$




$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \qquad y_j^{(1)} = f_1\left(z_j^{(1)}\right) \qquad z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \qquad y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$

$$z_j^{(3)} = \sum_i w_{ij}^{(3)} y_i^{(2)} \qquad y_j^{(3)} = f_3\left(z_j^{(3)}\right) \qquad \cdots$$



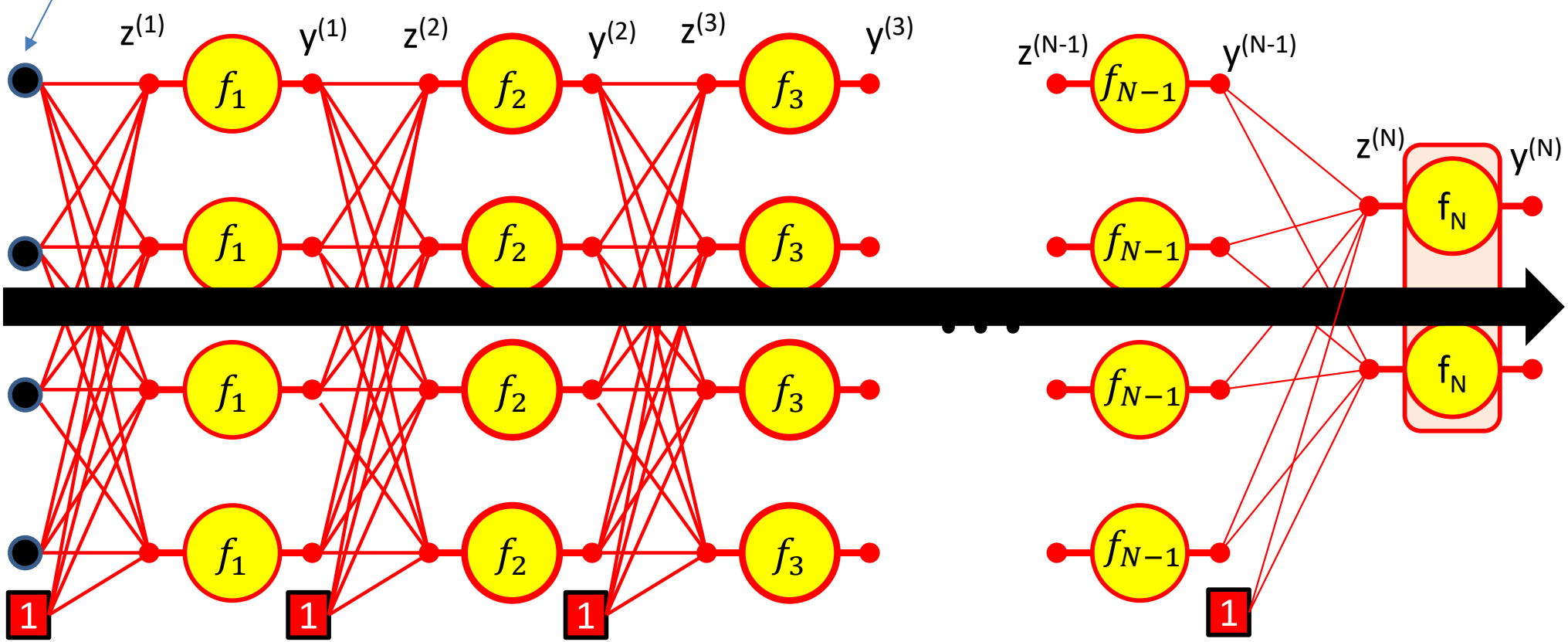

$$y_j^{(N-1)} = f_{N-1}\left(z_j^{(N-1)}\right)$$

$$z_j^{(N)} = \sum_i w_{ij}^{(N)} y_i^{(N-1)}$$

$$\boldsymbol{y}^{(N)} = f_N\left(\boldsymbol{z}^{(N)}\right)$$

# Forward Computation

$y^{(0)} = x$




ITERATE FOR k = 1:N

for j = 1:layer-width

$$y_i^{(0)} = x_i$$

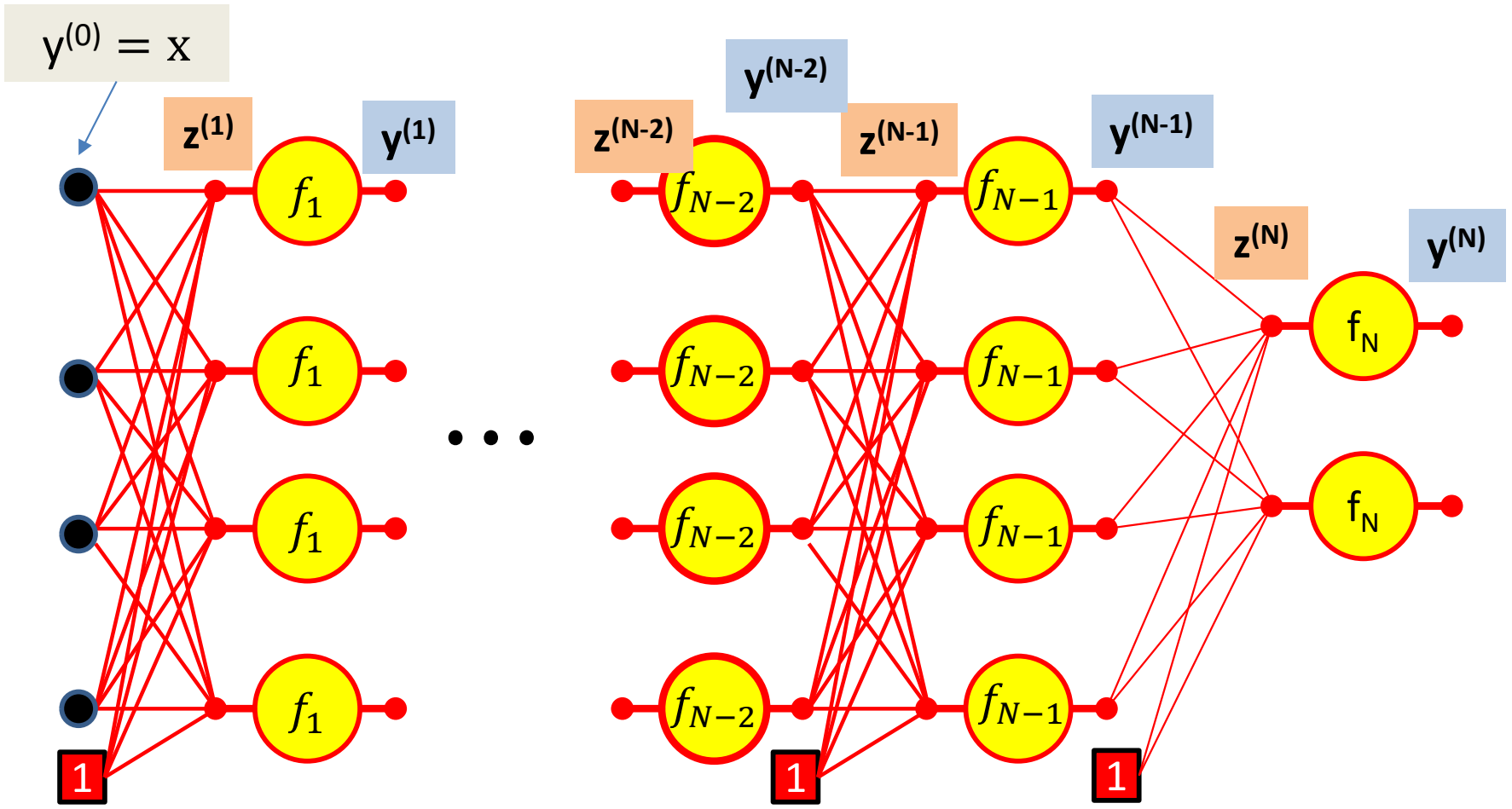
$$z_j^{(k)} = \sum_i w_{ij}^{(k)} y_i^{(k-1)}$$

$$y_j^{(k)} = f_k\left(z_j^{(k)}\right)$$

# 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:
  - $Y = y_j^{(N)}, j = 1..D_N$

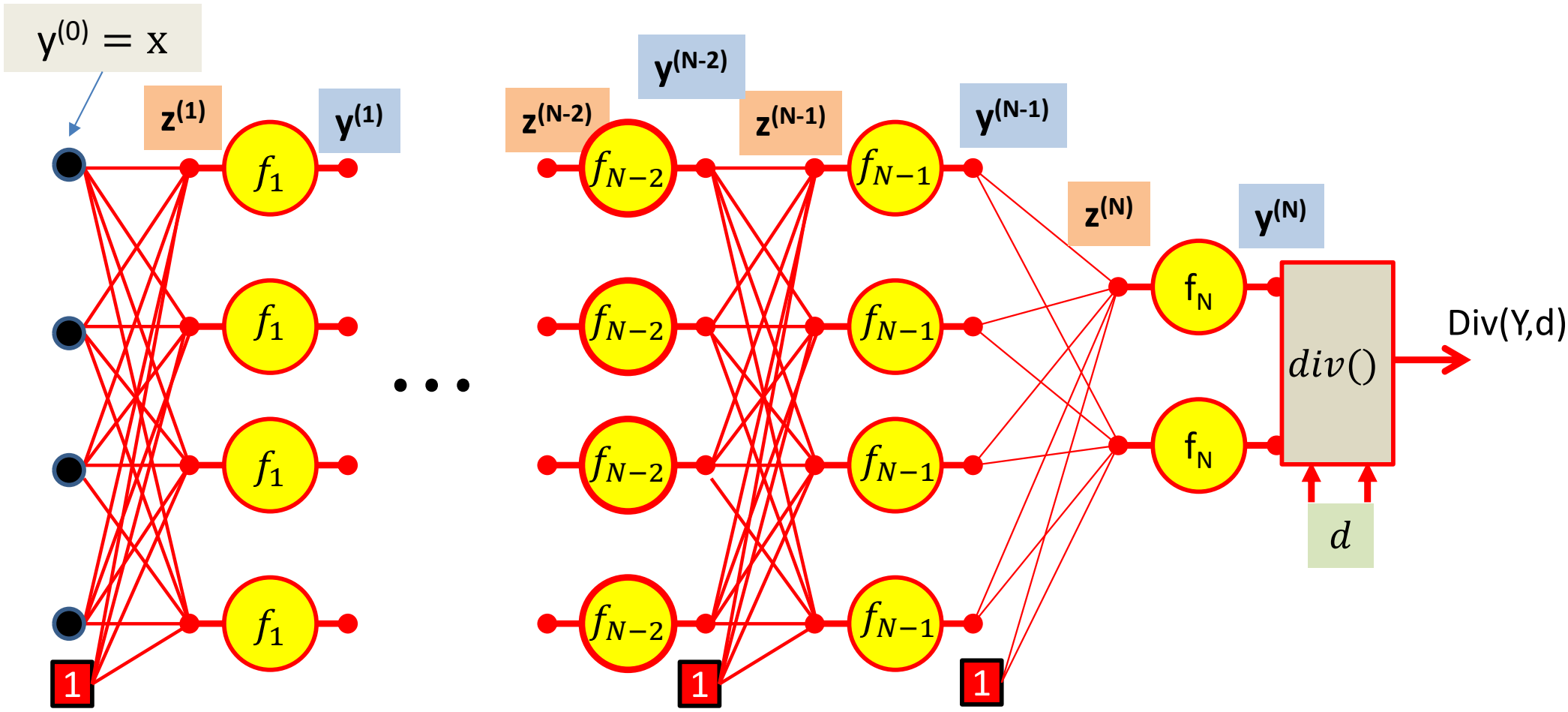
# Computing derivatives




We have computed all these intermediate values in the forward computation

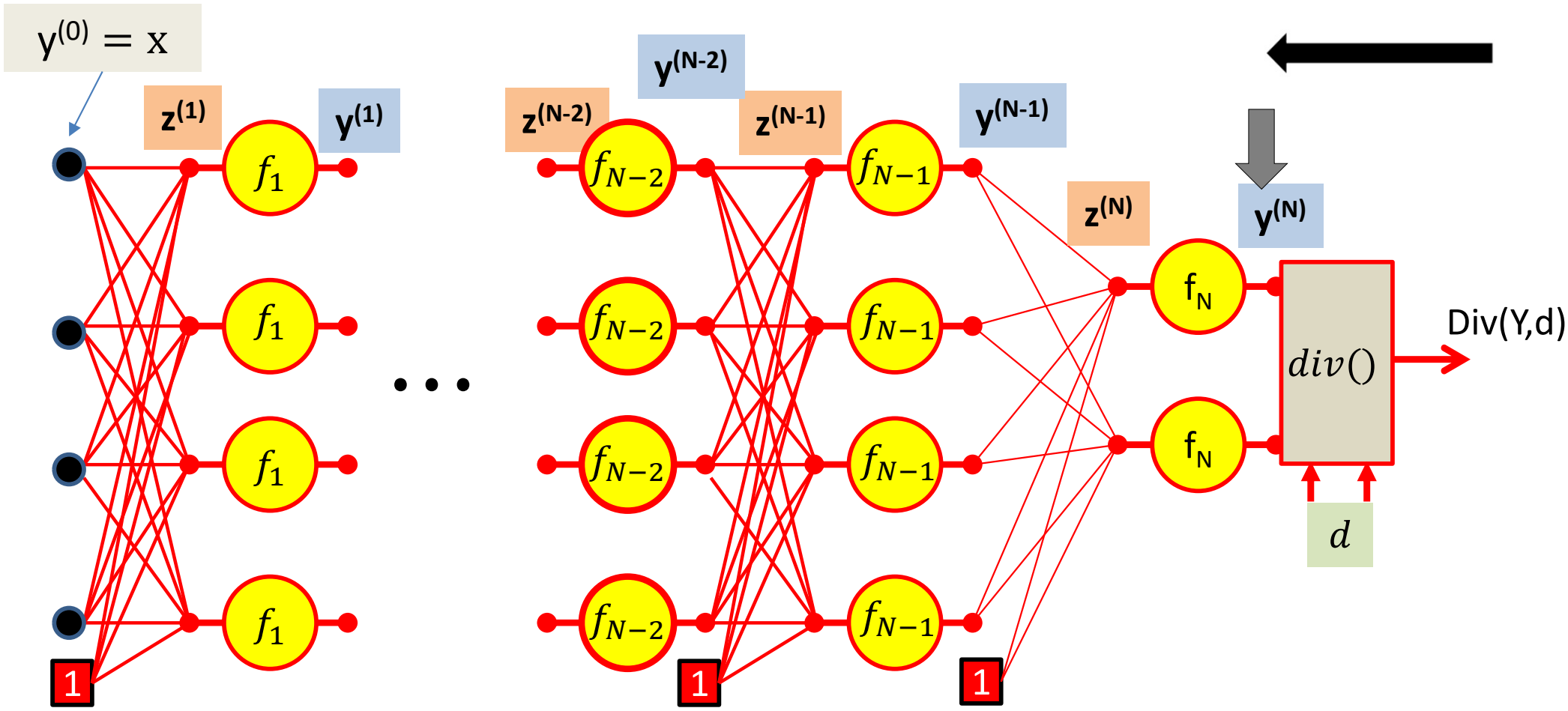
We must remember them – we will need them to compute the derivatives

# Computing derivatives



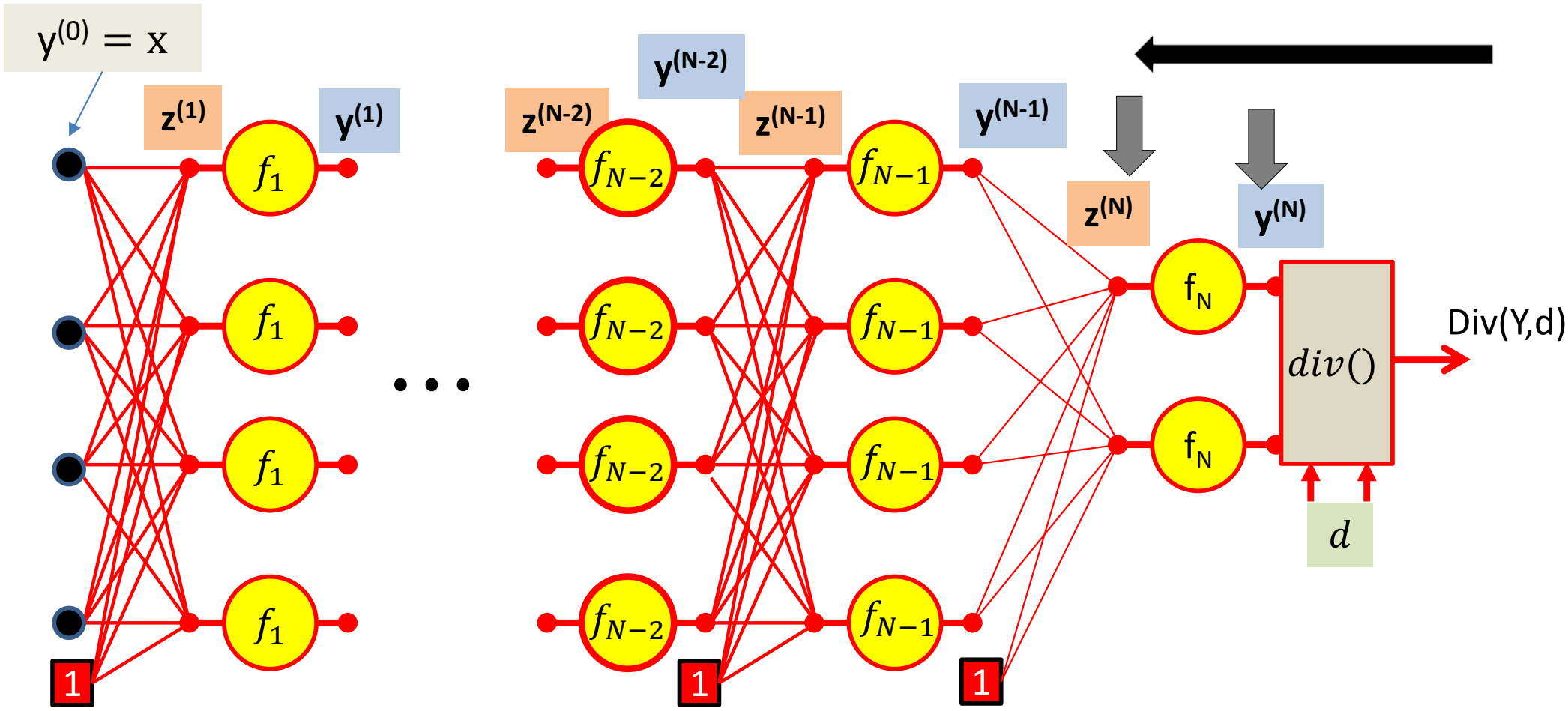

First, we compute the divergence between the output of the net $y = y^{(N)}$ and the desired output $d$

# Computing derivatives




We then compute $\nabla_{Y^{(N)}} div(.)$ the derivative of the divergence w.r.t. the final output of the network $y^{(N)}$

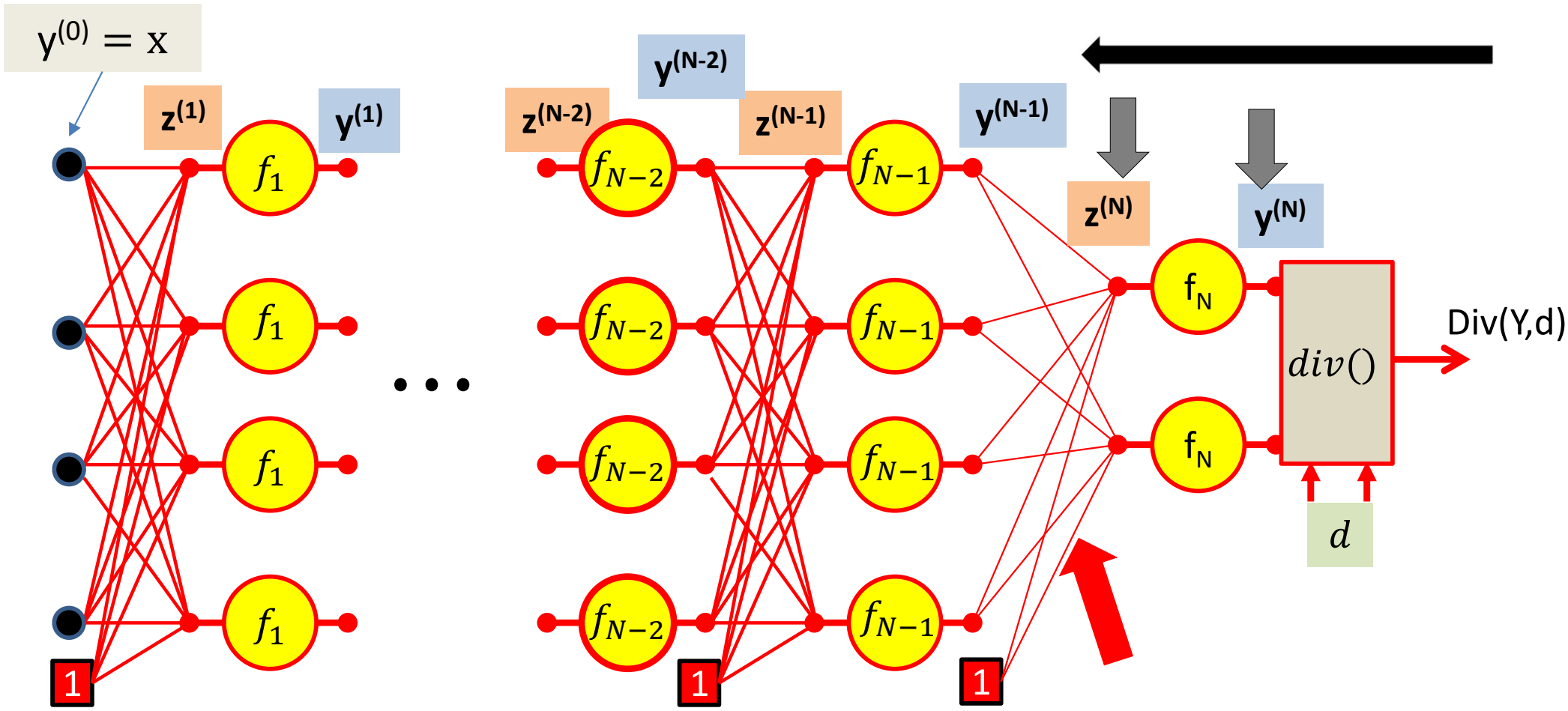
# Computing derivatives




We then compute $\nabla_{Y^{(N)}} div(.)$ the derivative of the divergence w.r.t. the final output of the network $y^{(N)}$

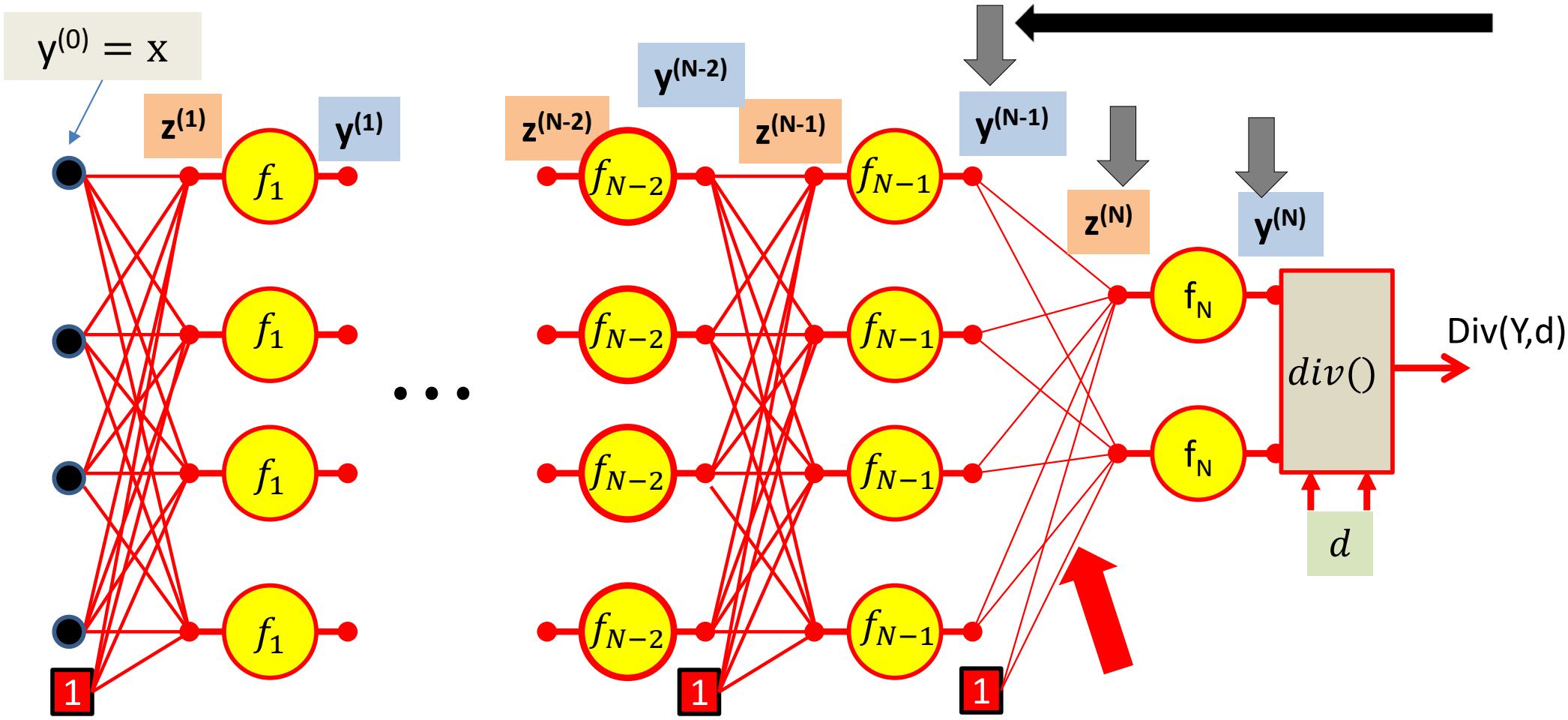
We then compute $\nabla_{Z^{(N)}} div(.)$ the derivative of the divergence w.r.t. the *pre-activation* affine combination $z^{(N)}$ using the chain rule

# Computing derivatives




Continuing on, we will compute $\nabla_{W^{(N)}} div(.)$ the derivative of the divergence with respect to the weights of the connections to the output layer

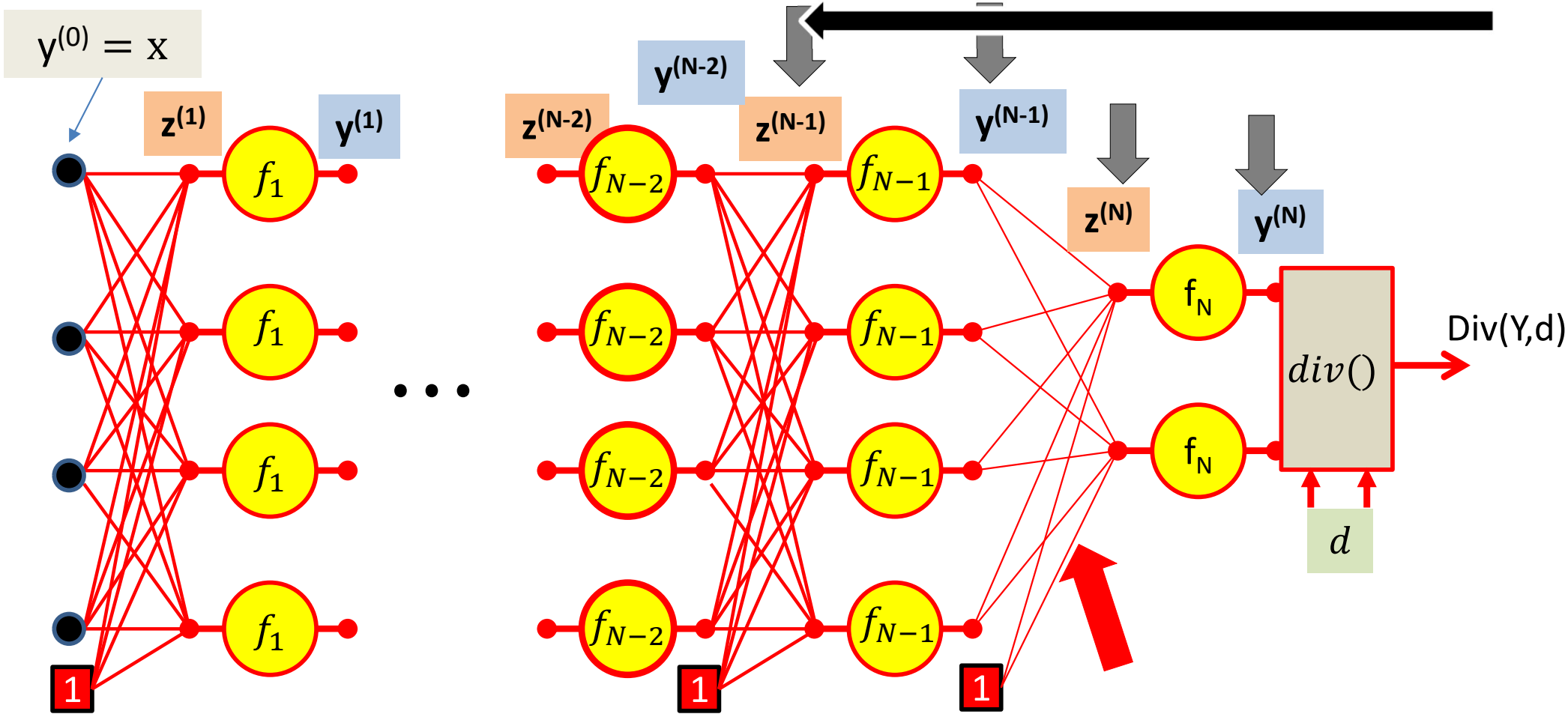
# Computing derivatives




Continuing on, we will compute $\nabla_{W^{(N)}} div(.)$ the derivative of the divergence with respect to the weights of the connections to the output layer

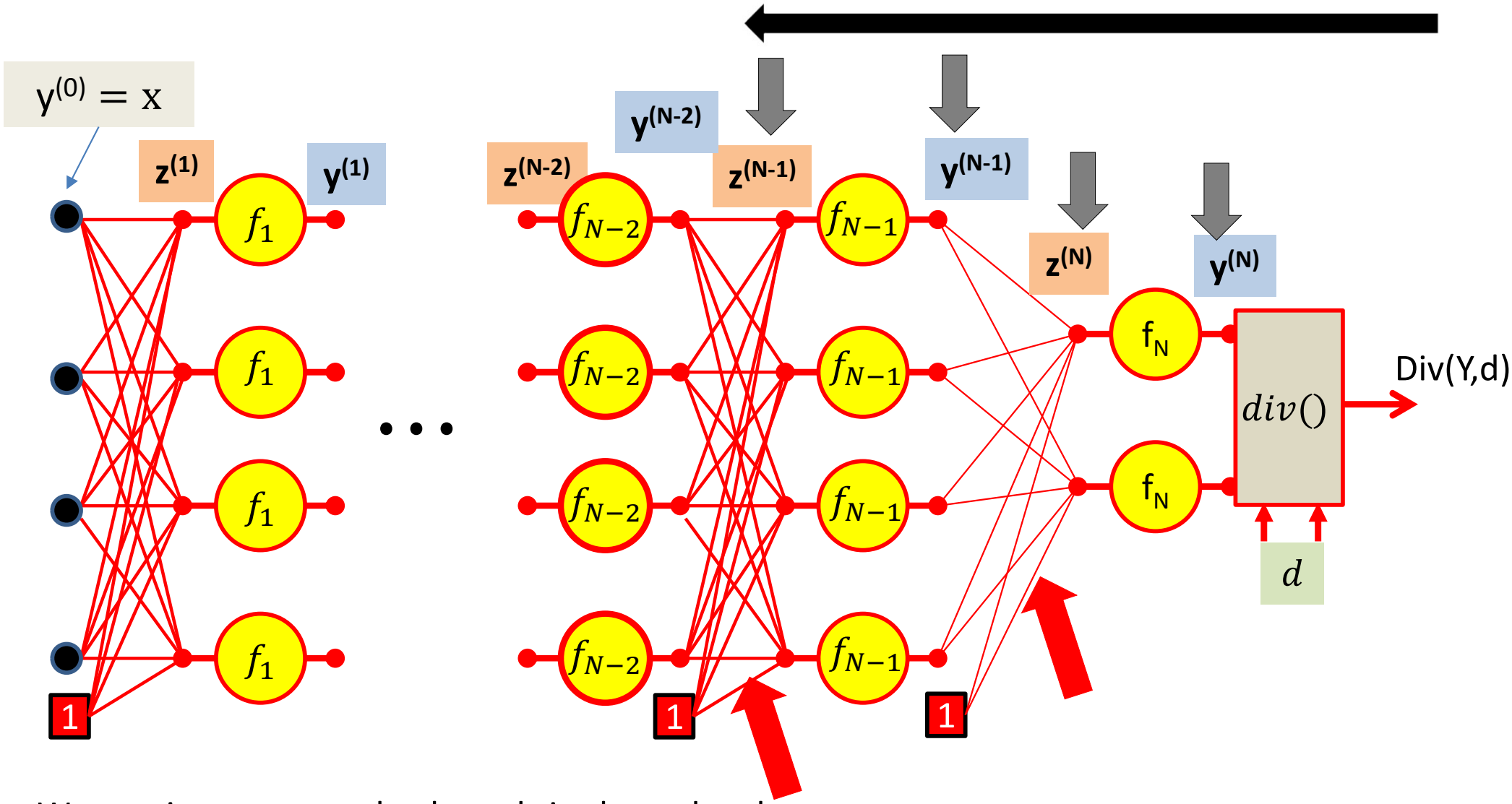
Then continue with the chain rule to compute $\nabla_{Y^{(N-1)}} div(.)$ the derivative of the divergence w.r.t. the output of the N-1th layer

# Computing derivatives



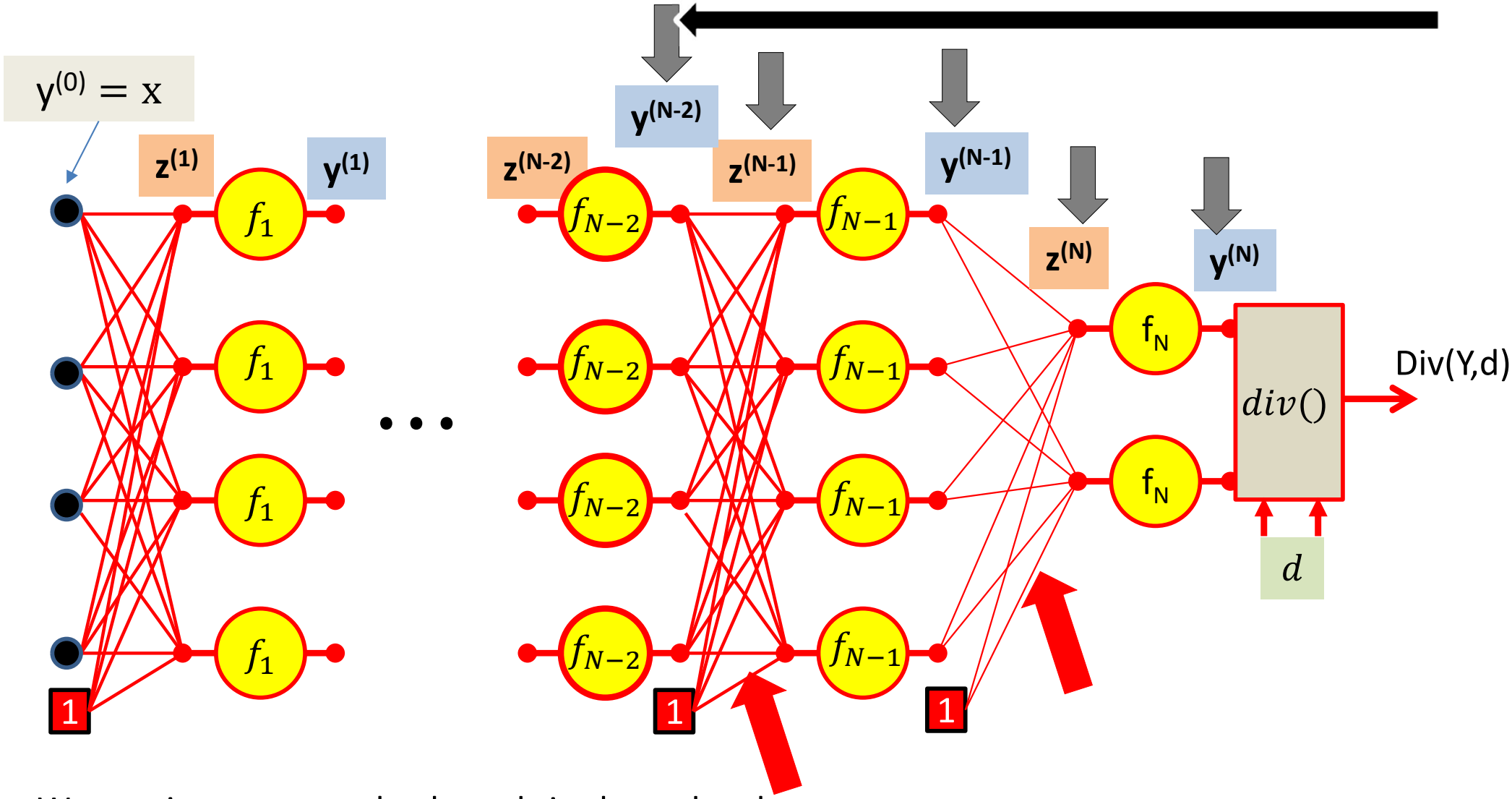

We continue our way backwards in the order shown

$$\nabla_{z^{(N-1)}} div(.)$$



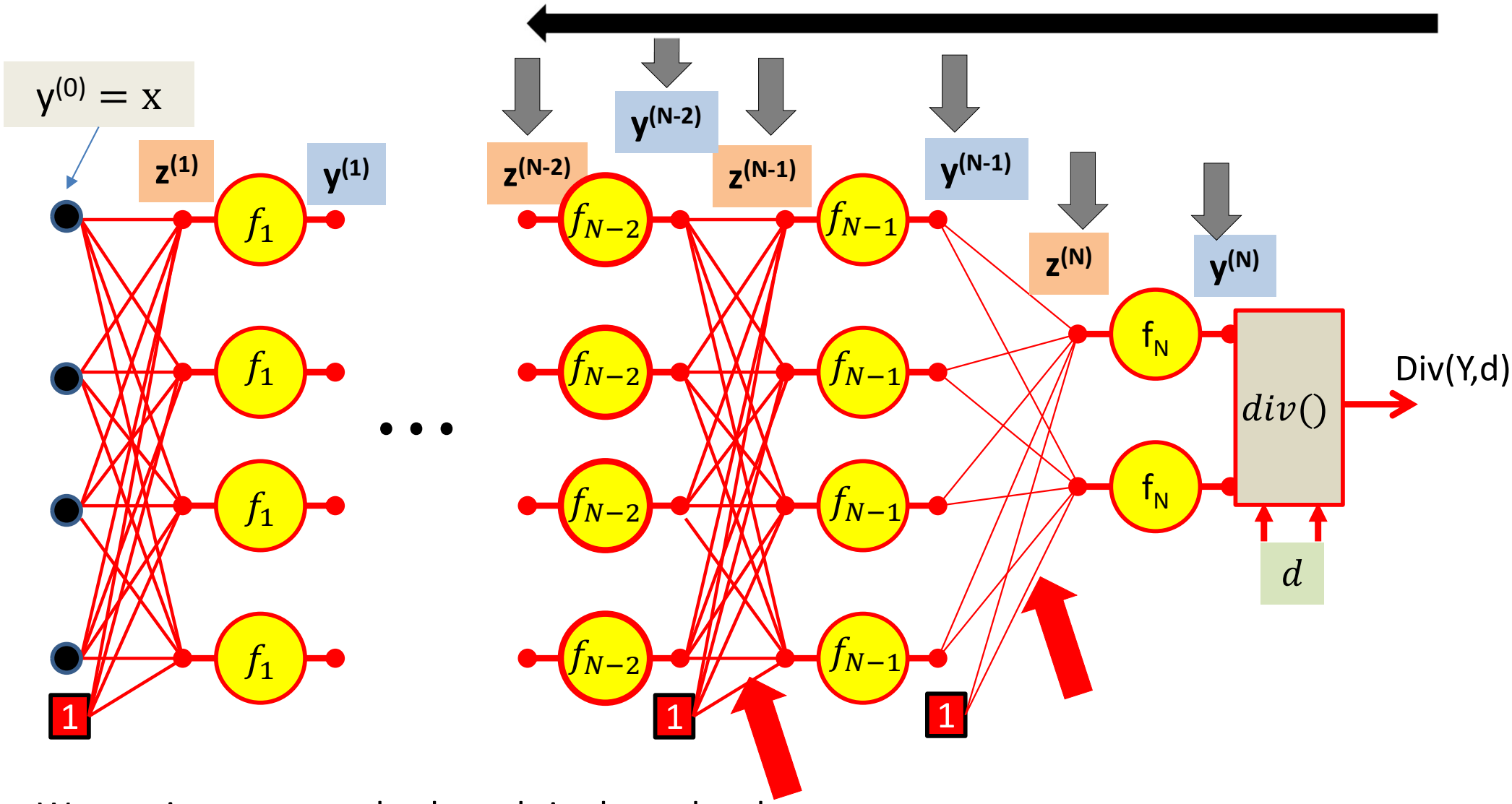

We continue our way backwards in the order shown

$$\nabla_{W^{(N-1)}} div(.)$$



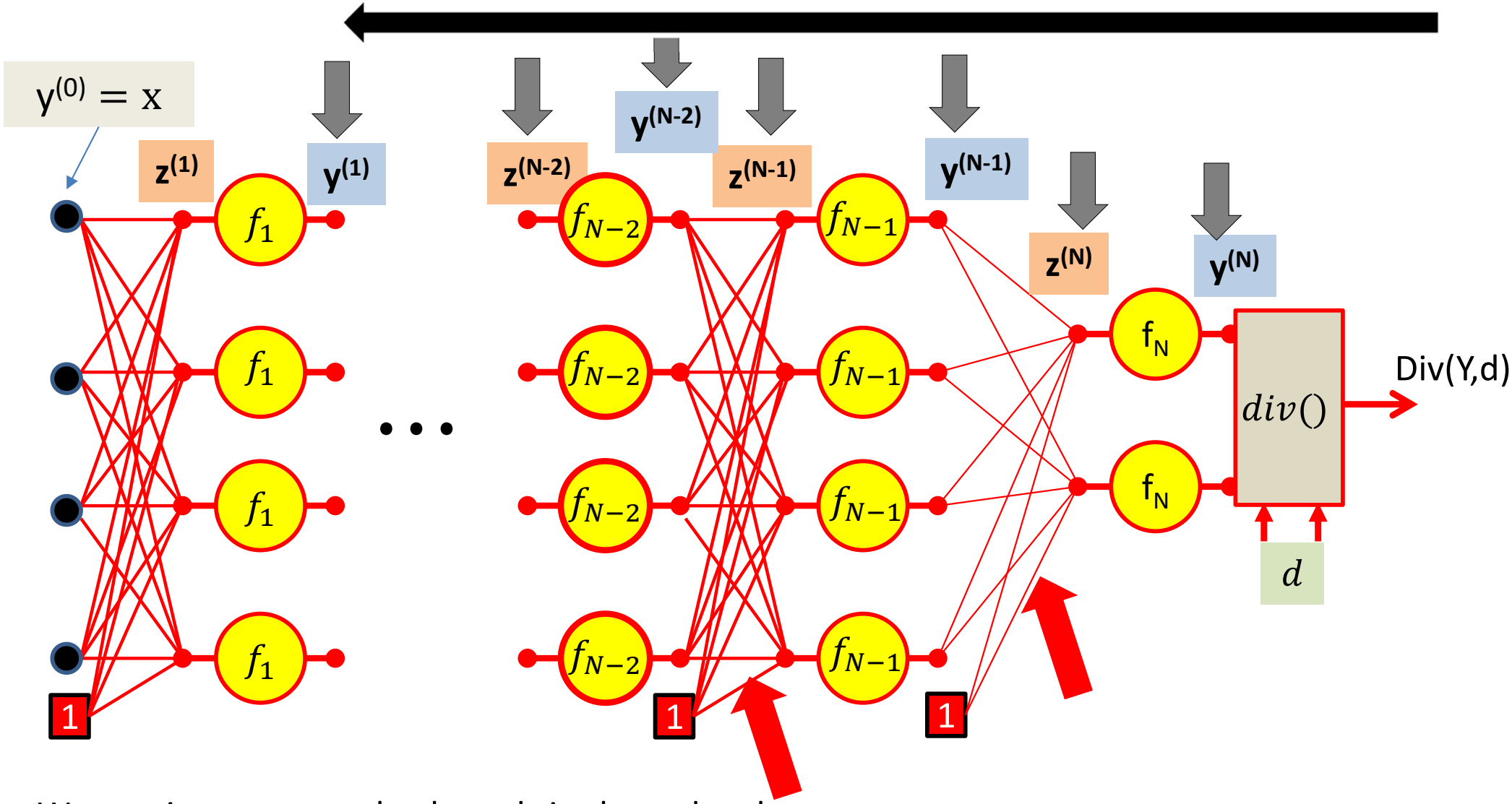

We continue our way backwards in the order shown

$$\nabla_{Y^{(N-2)}} div(.)$$



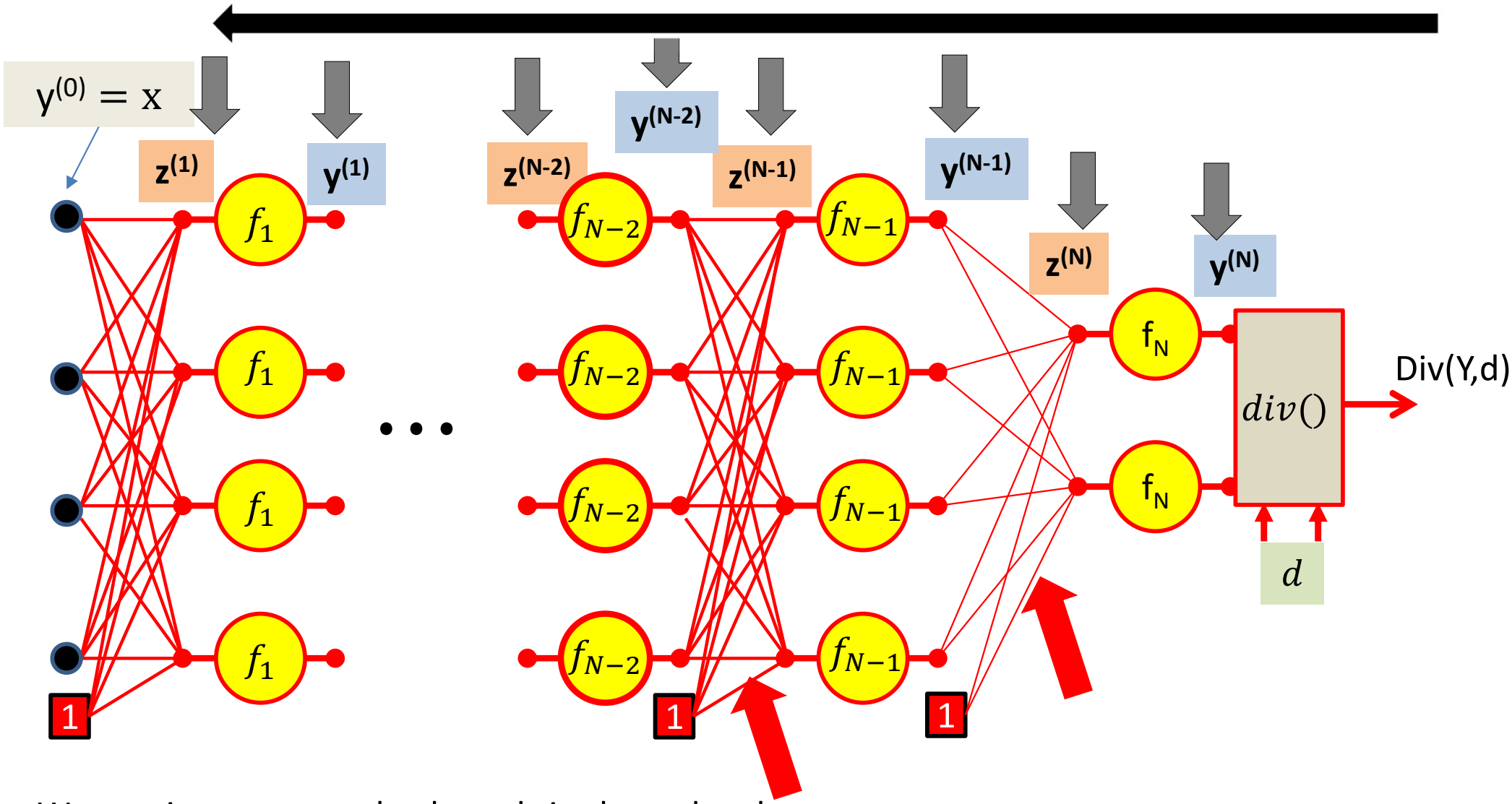

We continue our way backwards in the order shown

$$\nabla_{z^{(N-2)}} div(.)$$



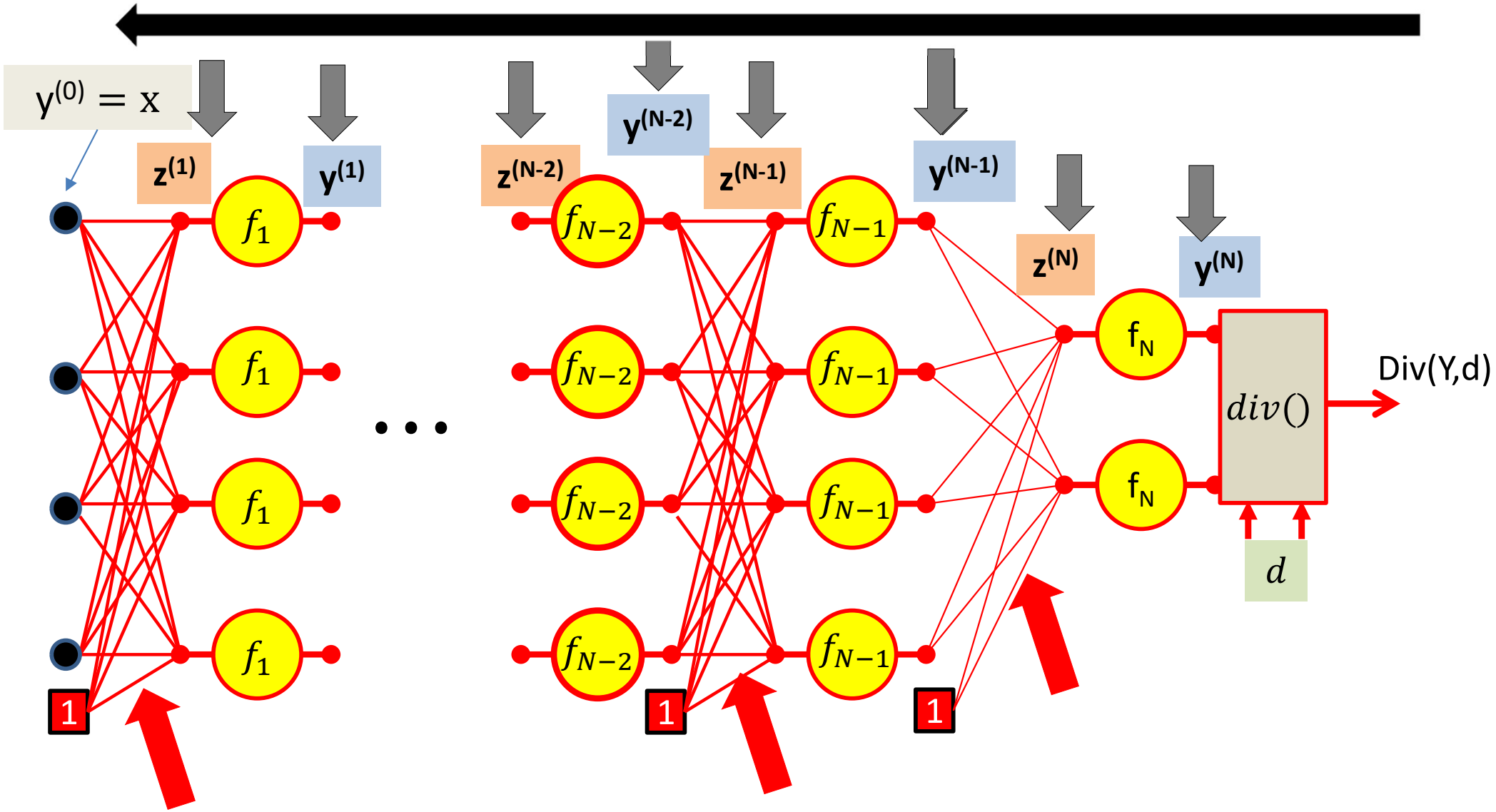

We continue our way backwards in the order shown

$$\nabla_{Y(1)} div(.)$$




We continue our way backwards in the order shown

$$\nabla_{z^{(1)}} div(.)$$



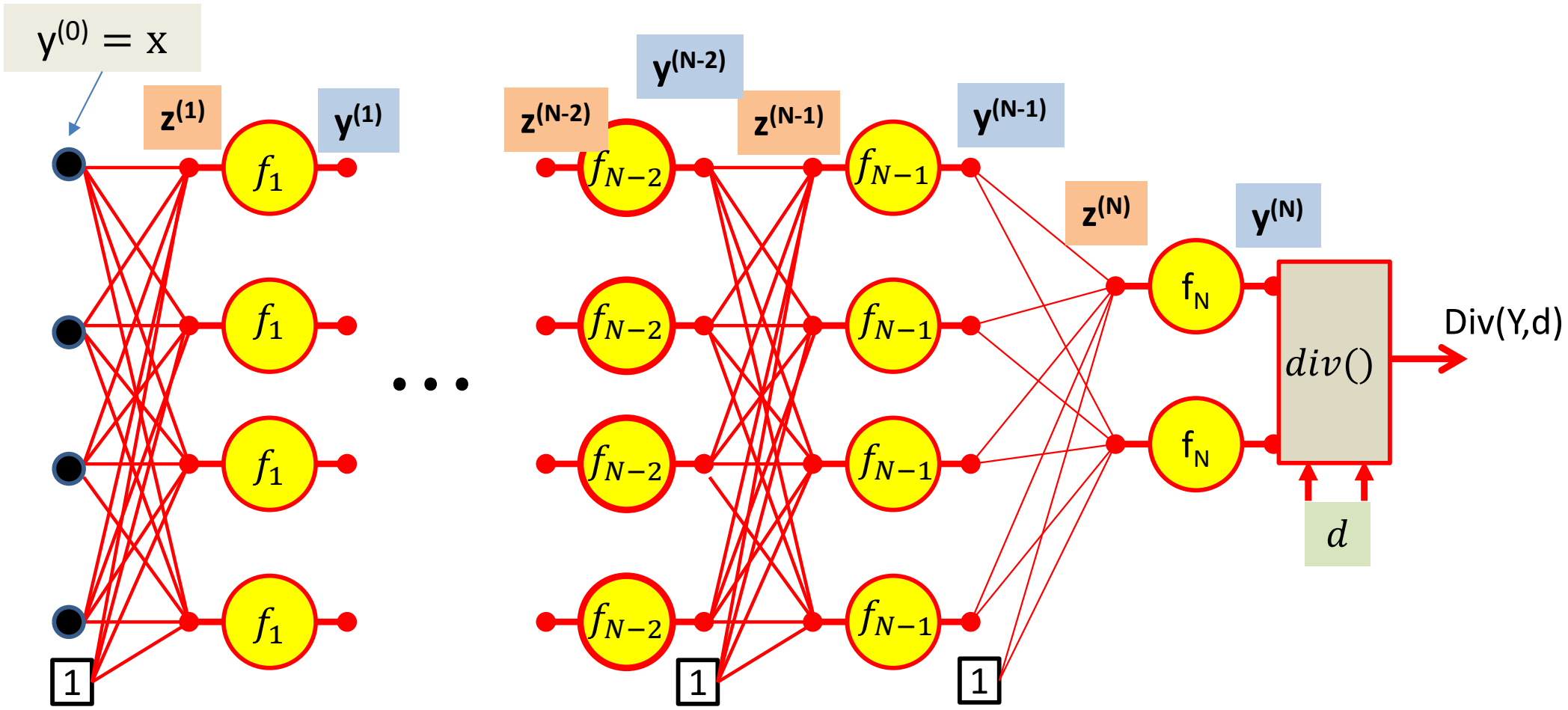

We continue our way backwards in the order shown

$$\nabla_{W^{(1)}} div(.)$$

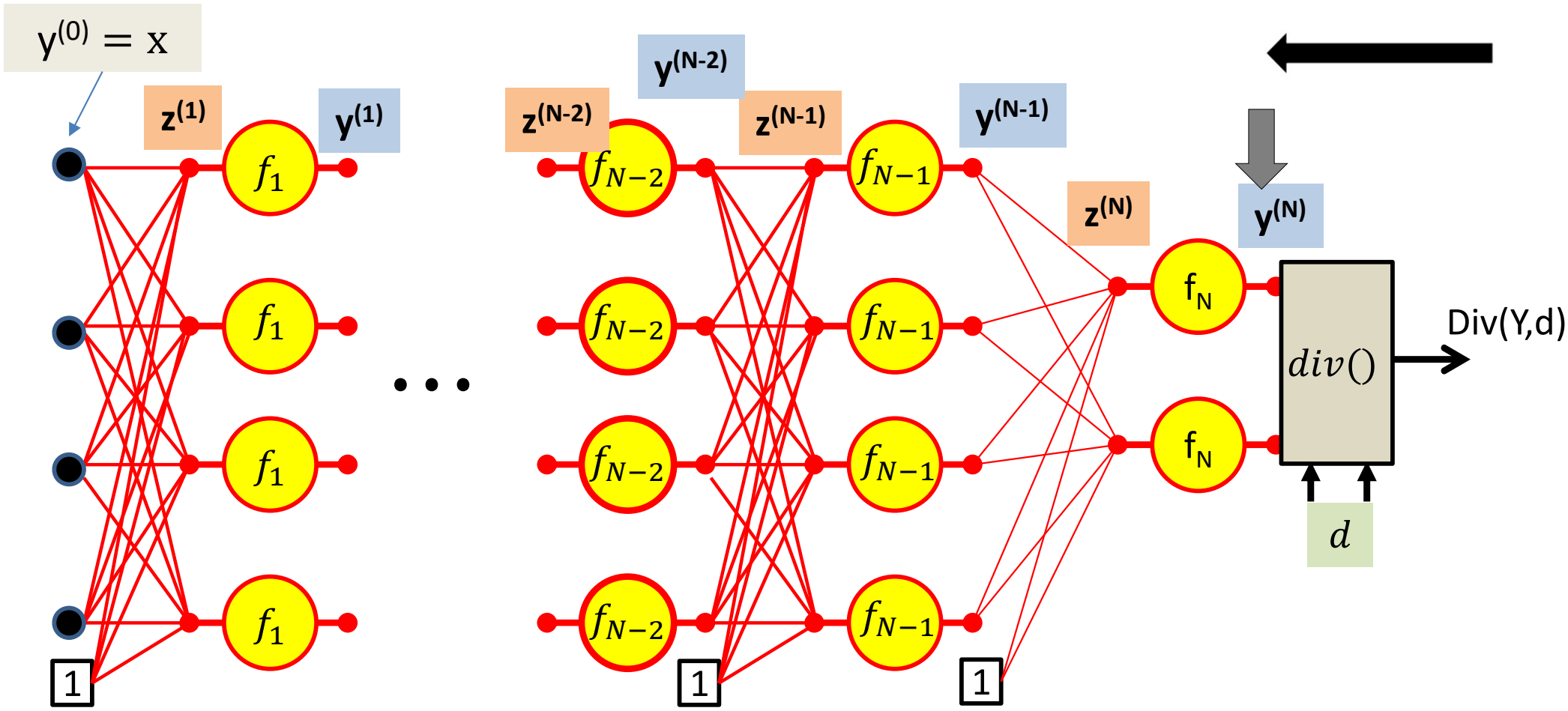
# Backward Gradient Computation

- Let’s actually see the math..

# Computing derivatives



# Computing derivatives




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

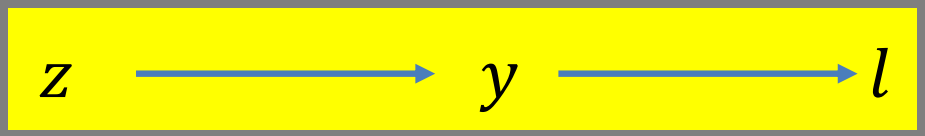
$$\frac{\partial Div(Y,d)}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$$

# Calculus Refresher: Chain rule

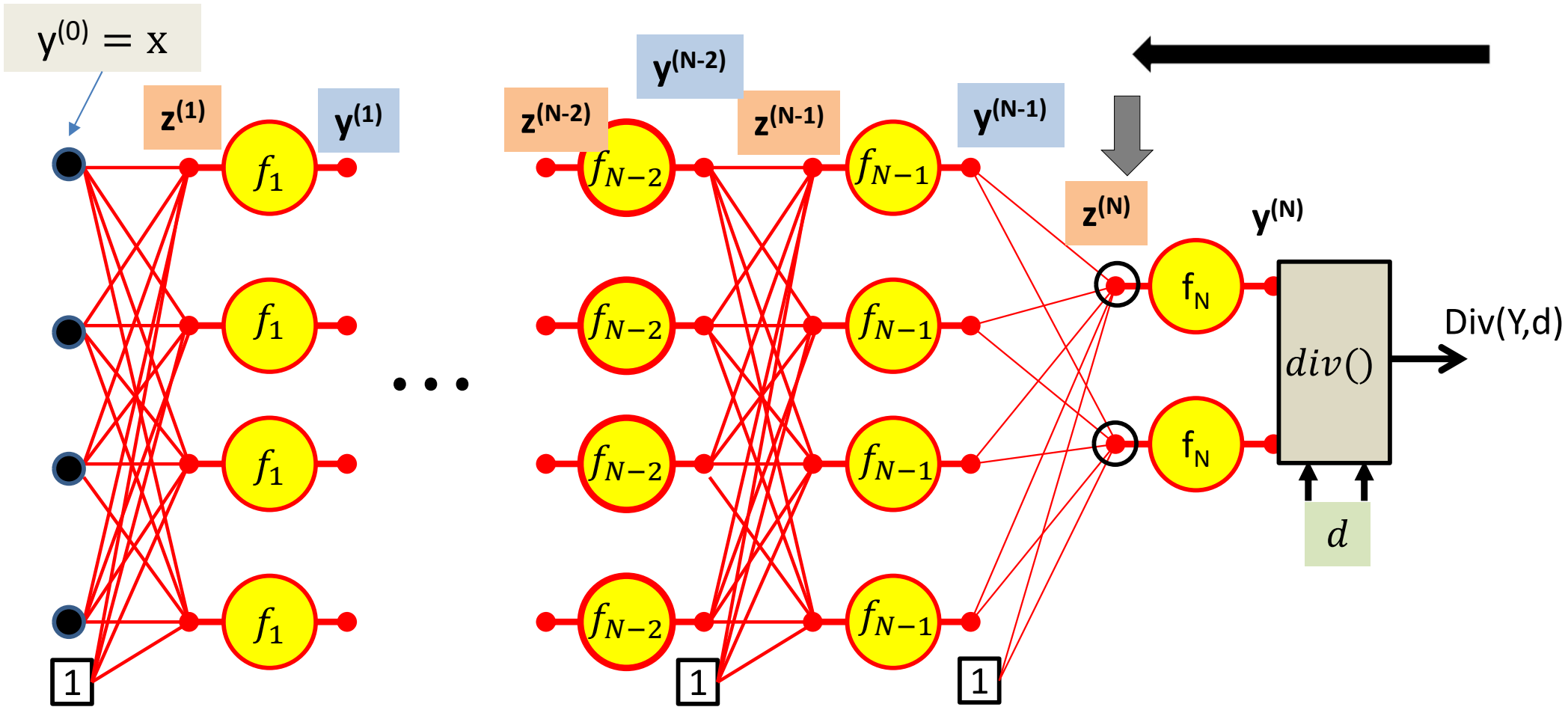
For any nested function $l = f(y)$ where $y = g(z)$

$$\frac{dl}{dz} = \frac{dl}{dy}\frac{dy}{dz}$$

$$\frac{dl}{dz} = \frac{dl}{dy}\frac{dy}{dz}$$

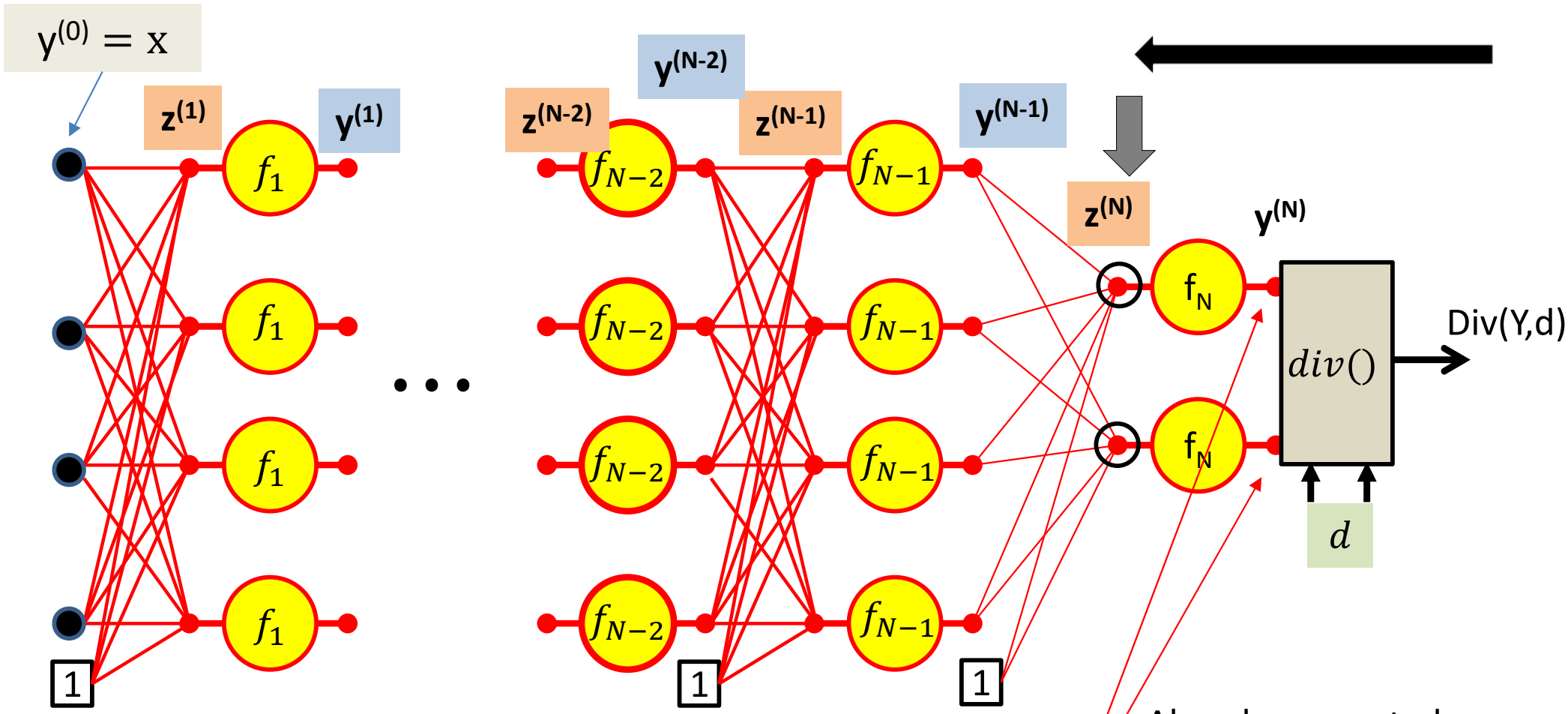


# Computing derivatives



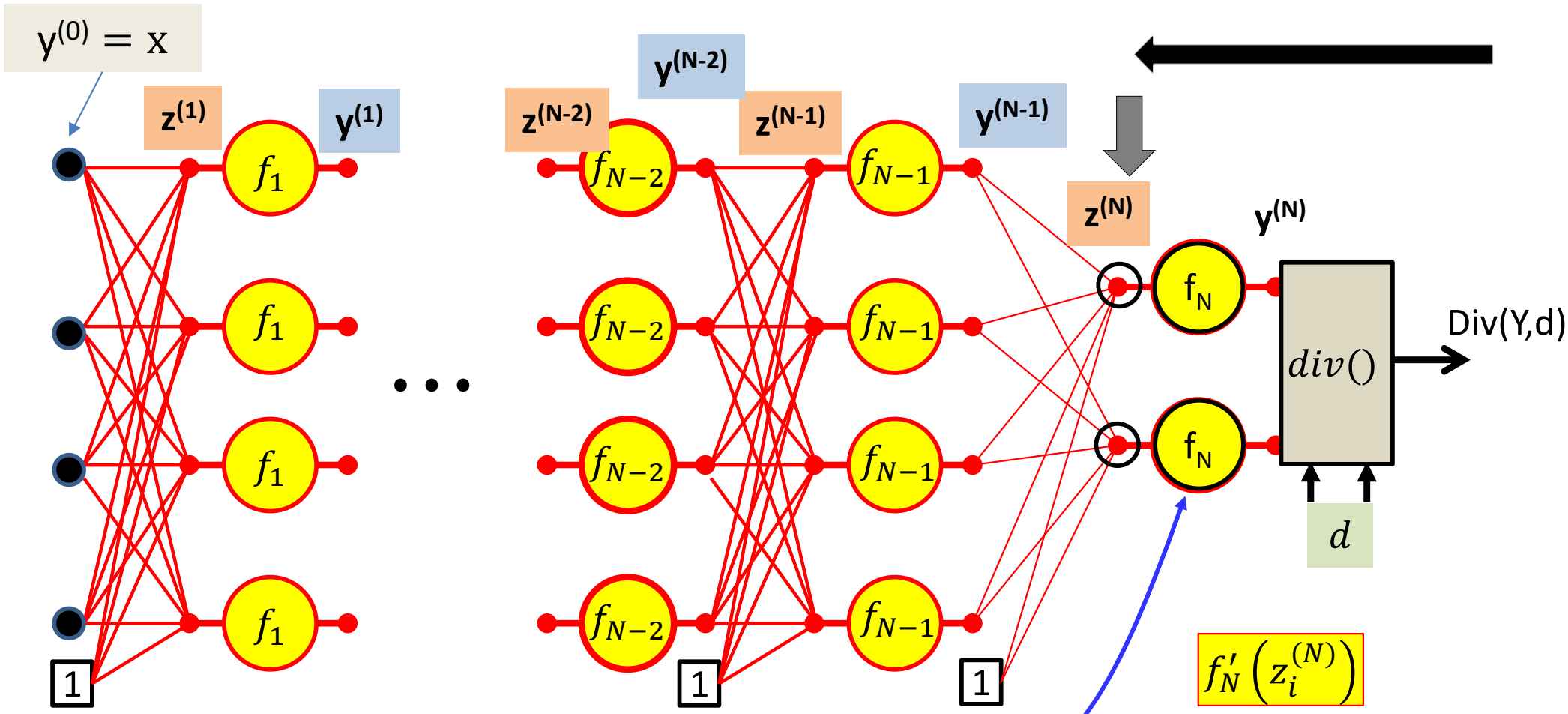

$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

# Computing derivatives




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

# Computing derivatives

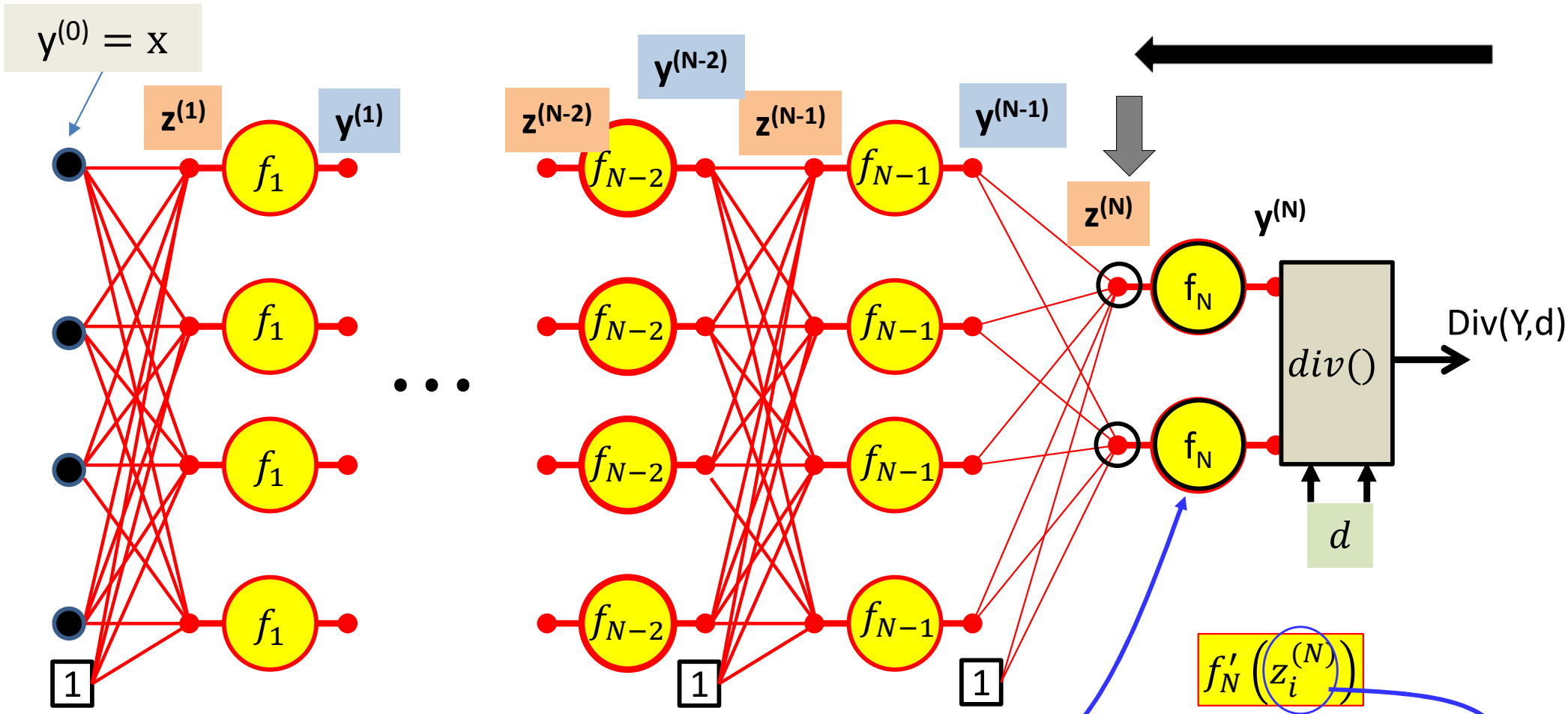



$$f_N'\left(z_i^{(N)}\right)$$

Derivative of activation function

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

# Computing derivatives



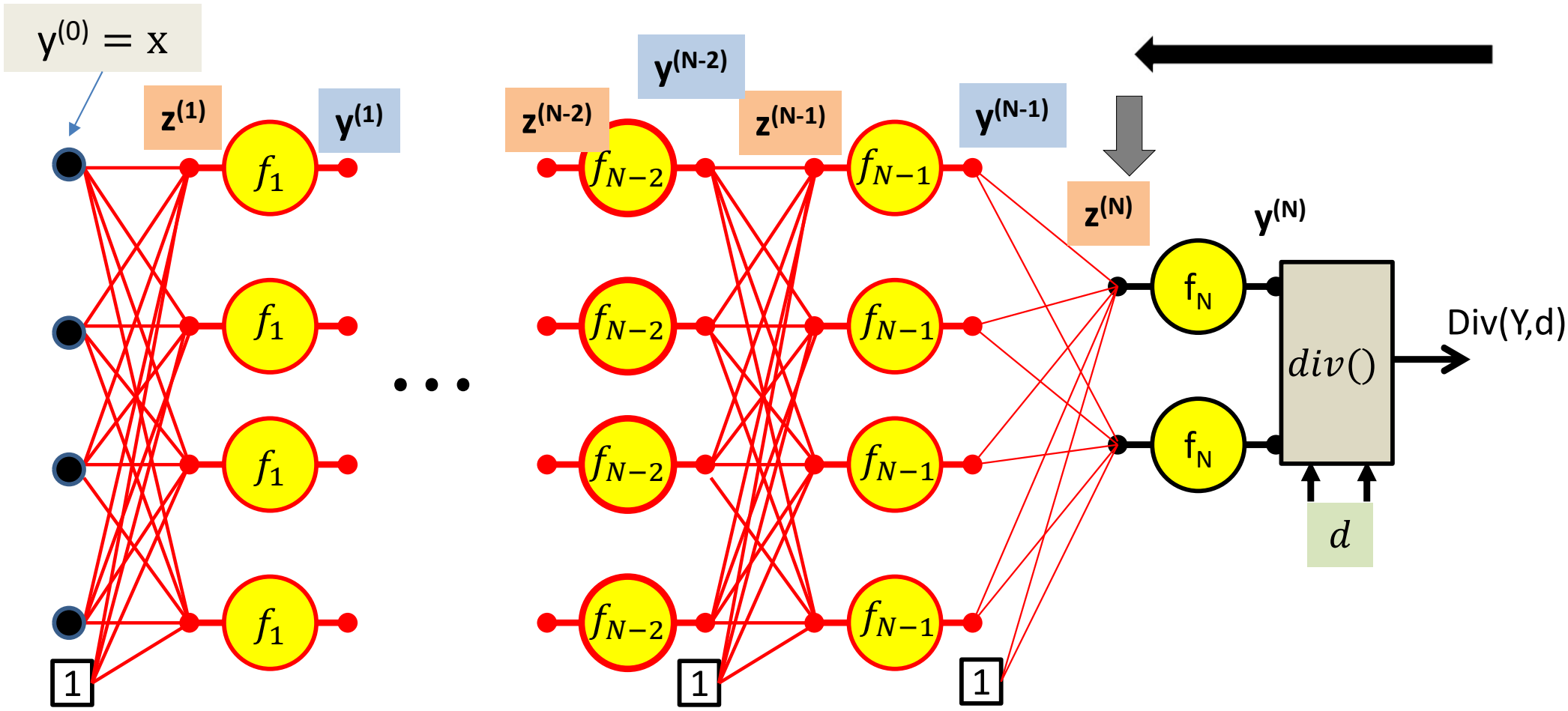

$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

$f_N'\left(z_i^{(N)}\right)$

Derivative of activation function

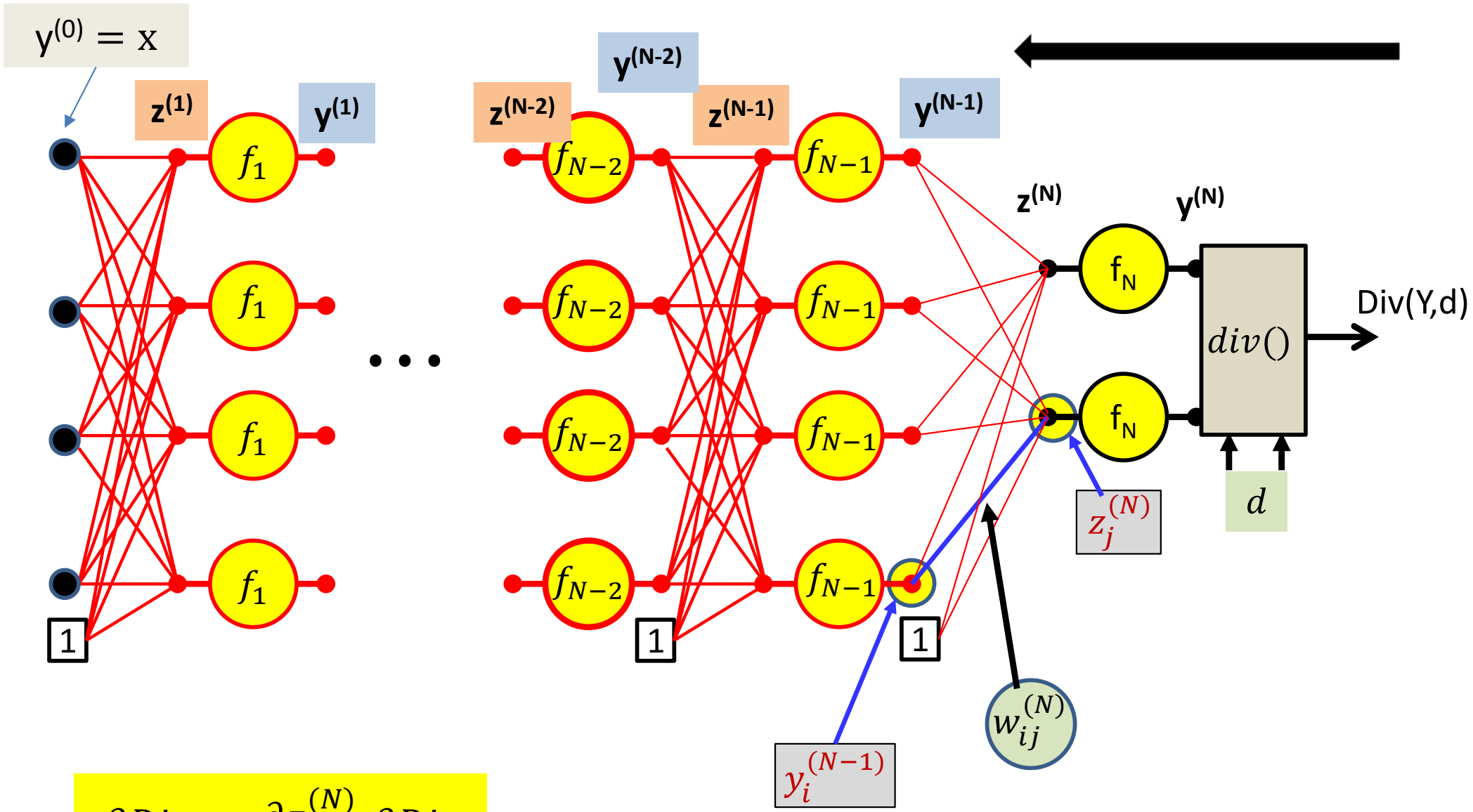
Computed in forward pass

# Computing derivatives



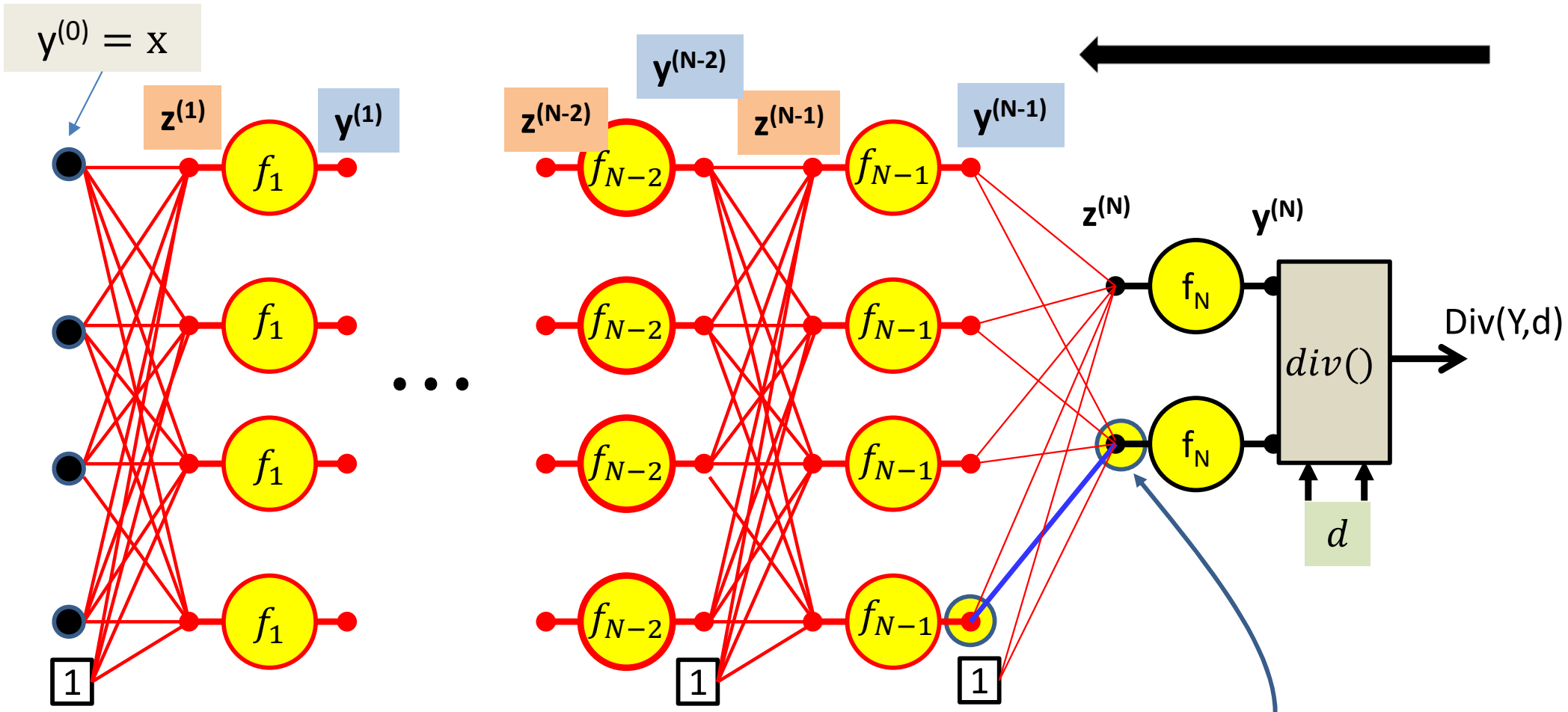

$$\frac{\partial Div}{\partial z_i^{(N)}} = f_N'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

# Computing derivatives




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

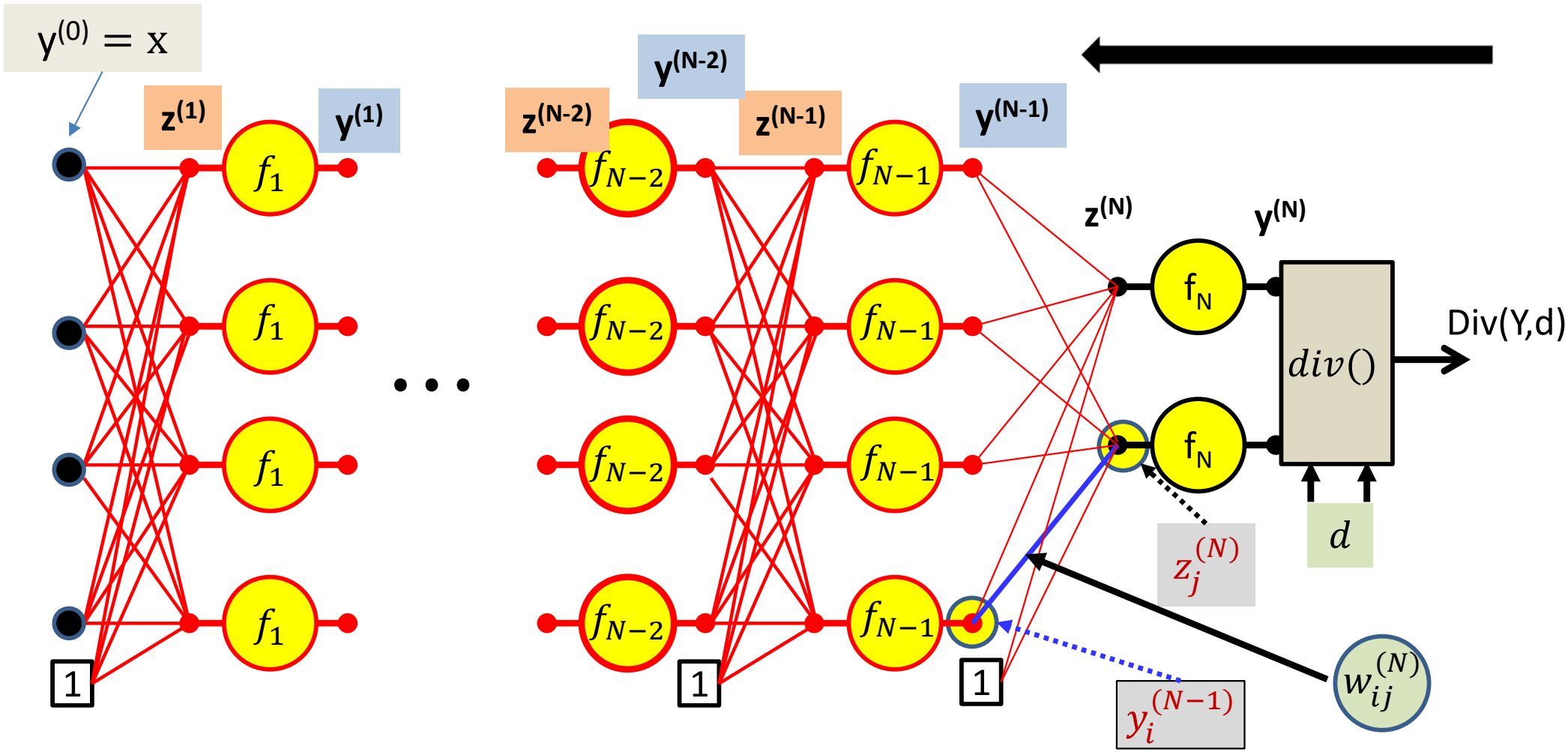
# Computing derivatives




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

Just computed

# Computing derivatives



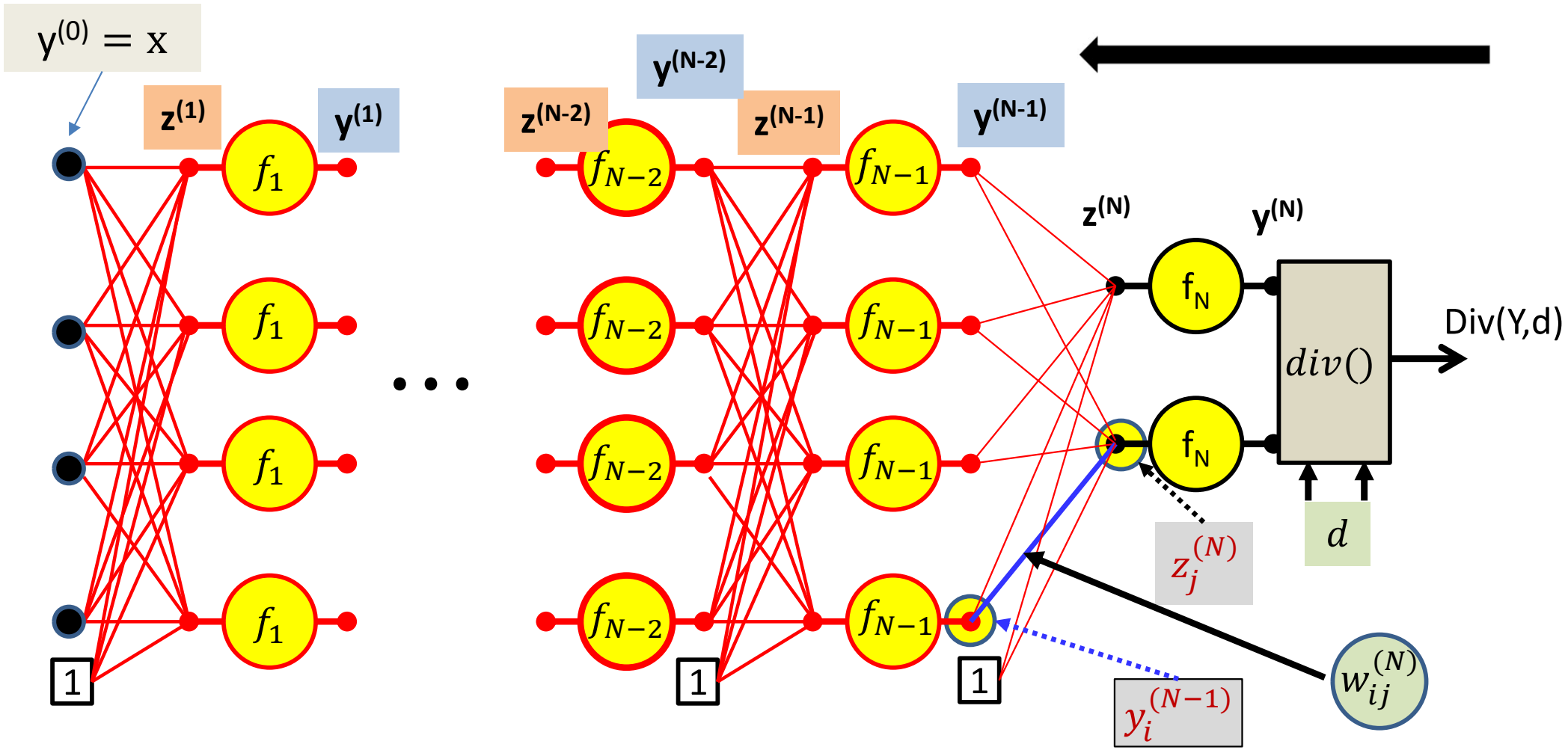

$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$

Because

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

# Computing derivatives




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

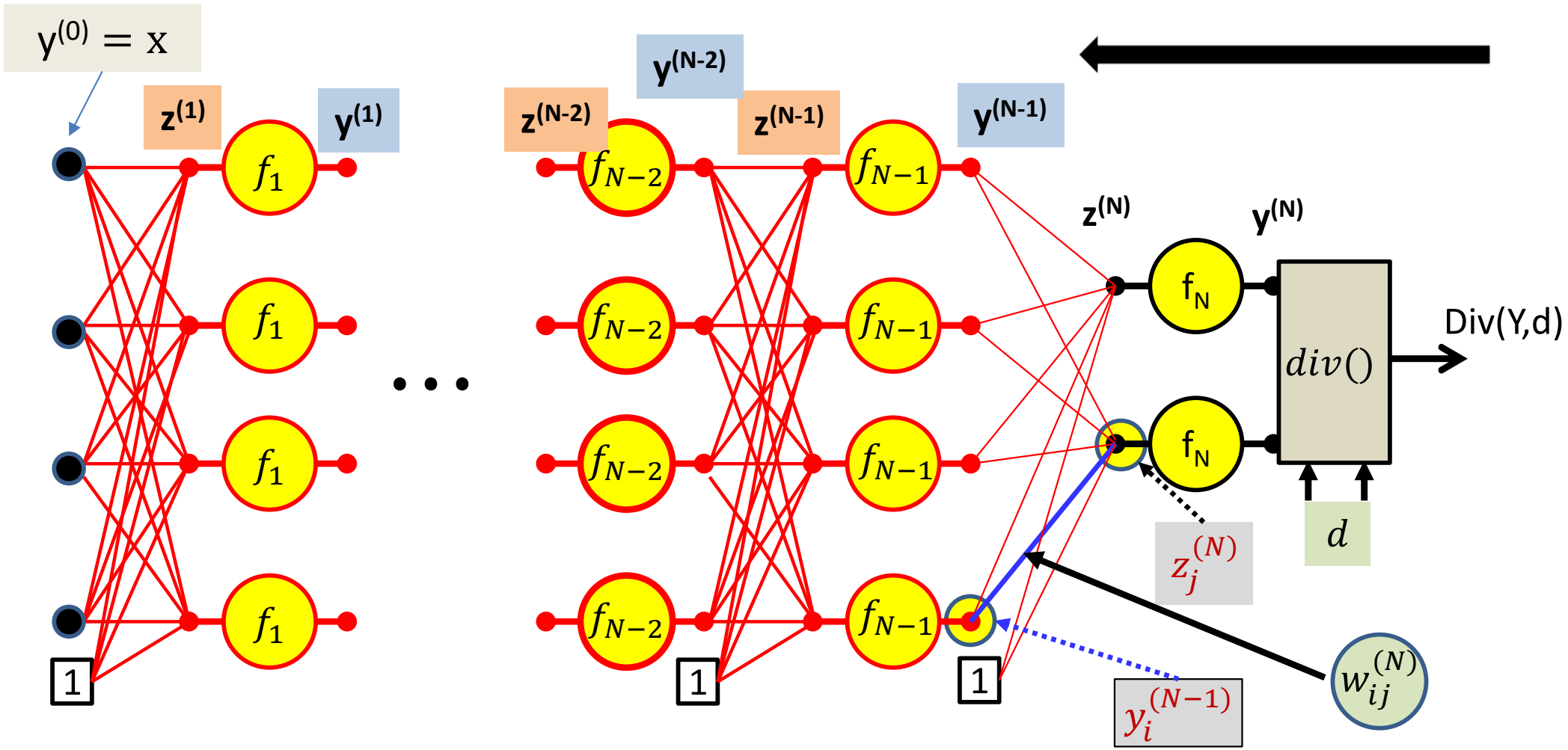
$y_i^{(N-1)}$

Because

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

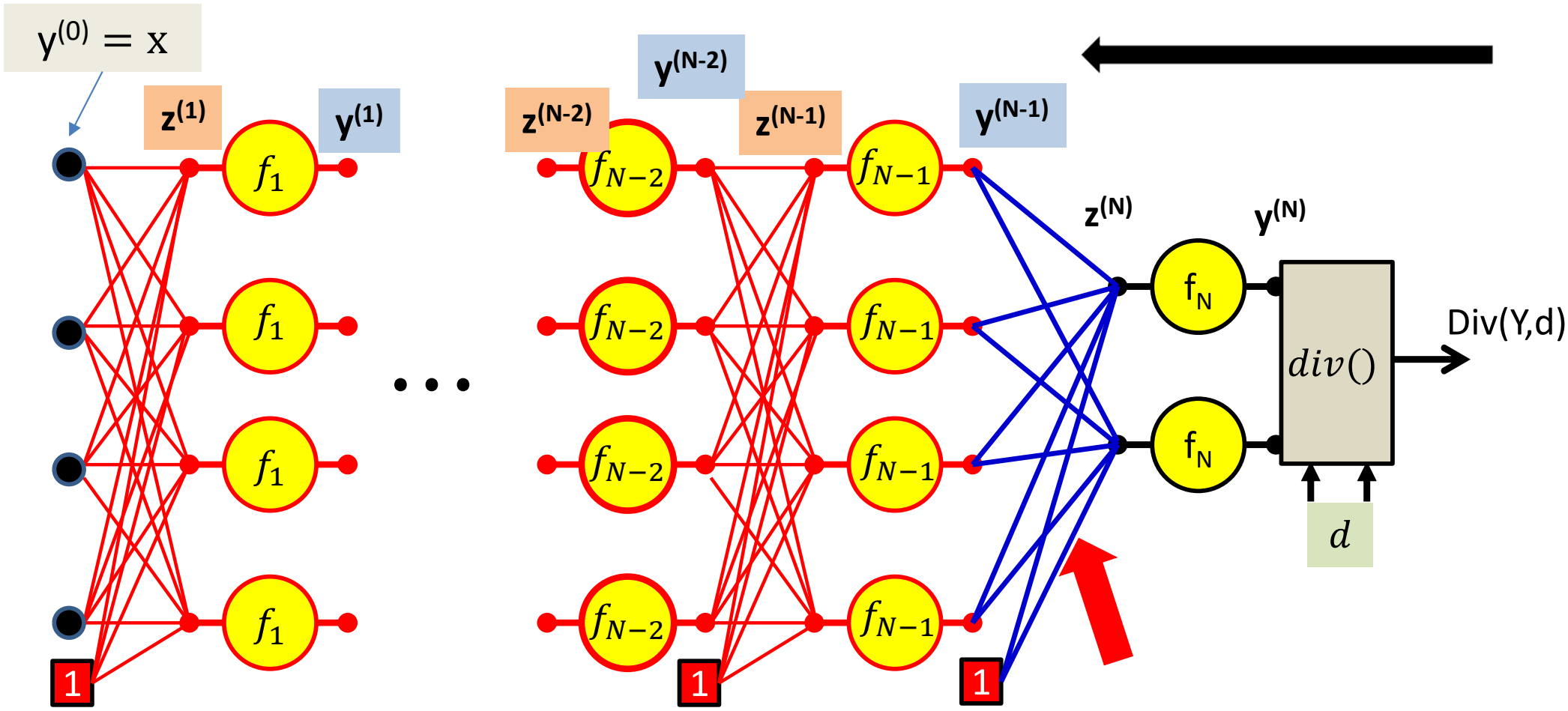
Computed in forward pass

# Computing derivatives




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

# Computing derivatives



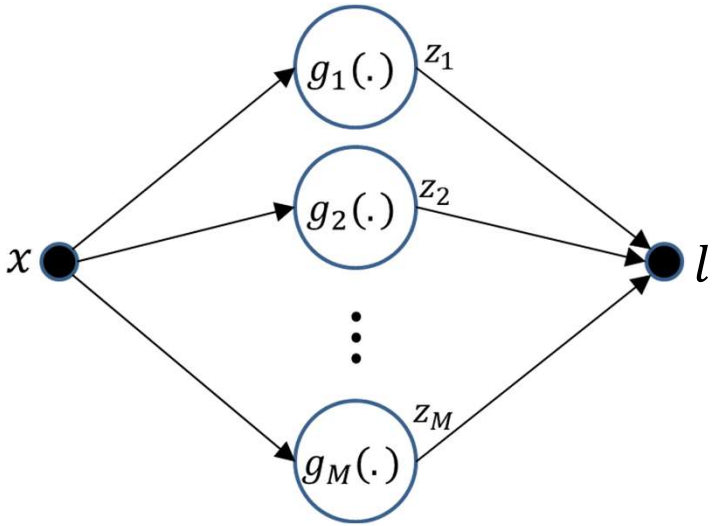

$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$$

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

# Calculus Refresher: Chain rule

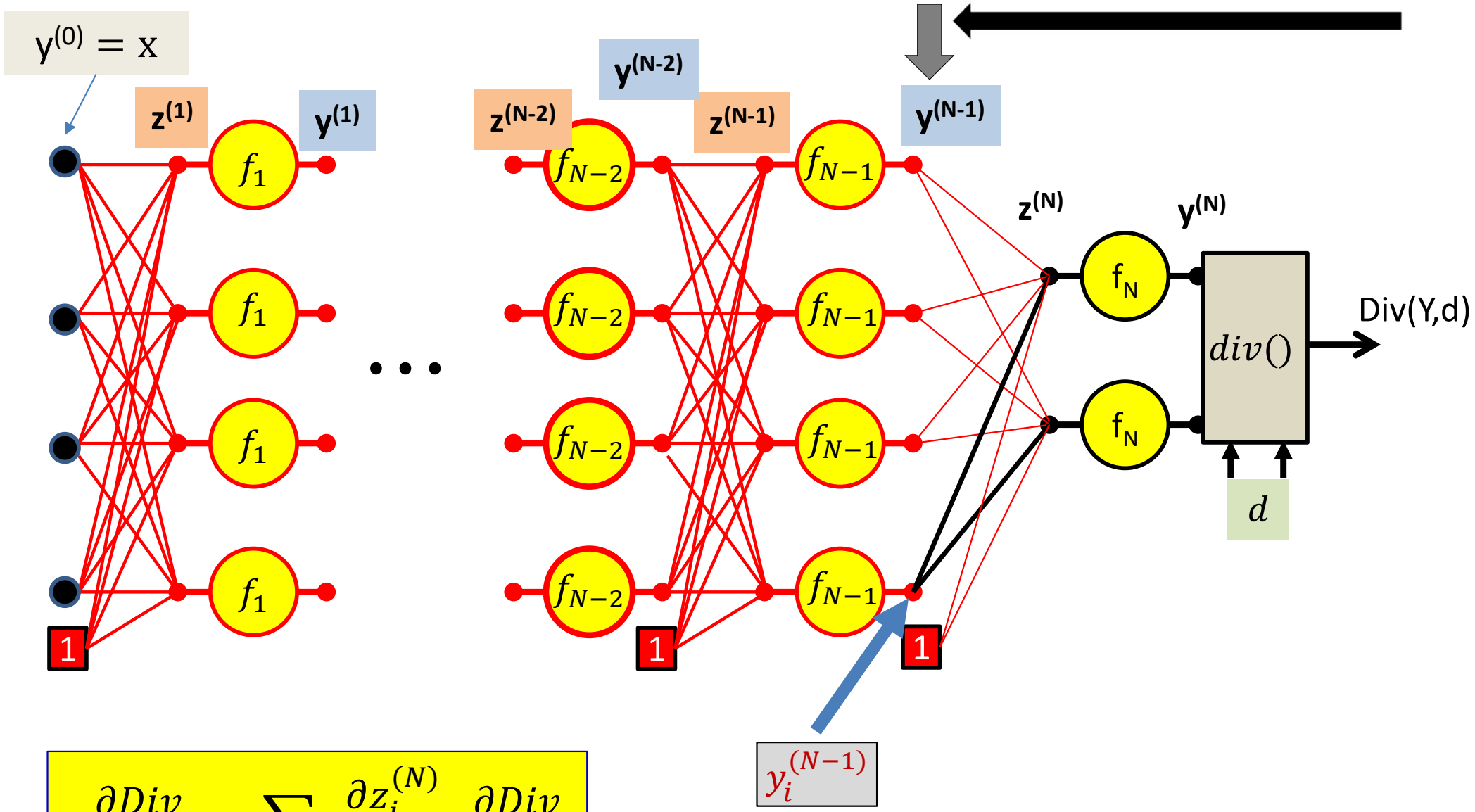
For $l = f(z_1, z_2, \ldots, z_M)$

where $z_i = g_i(x)$



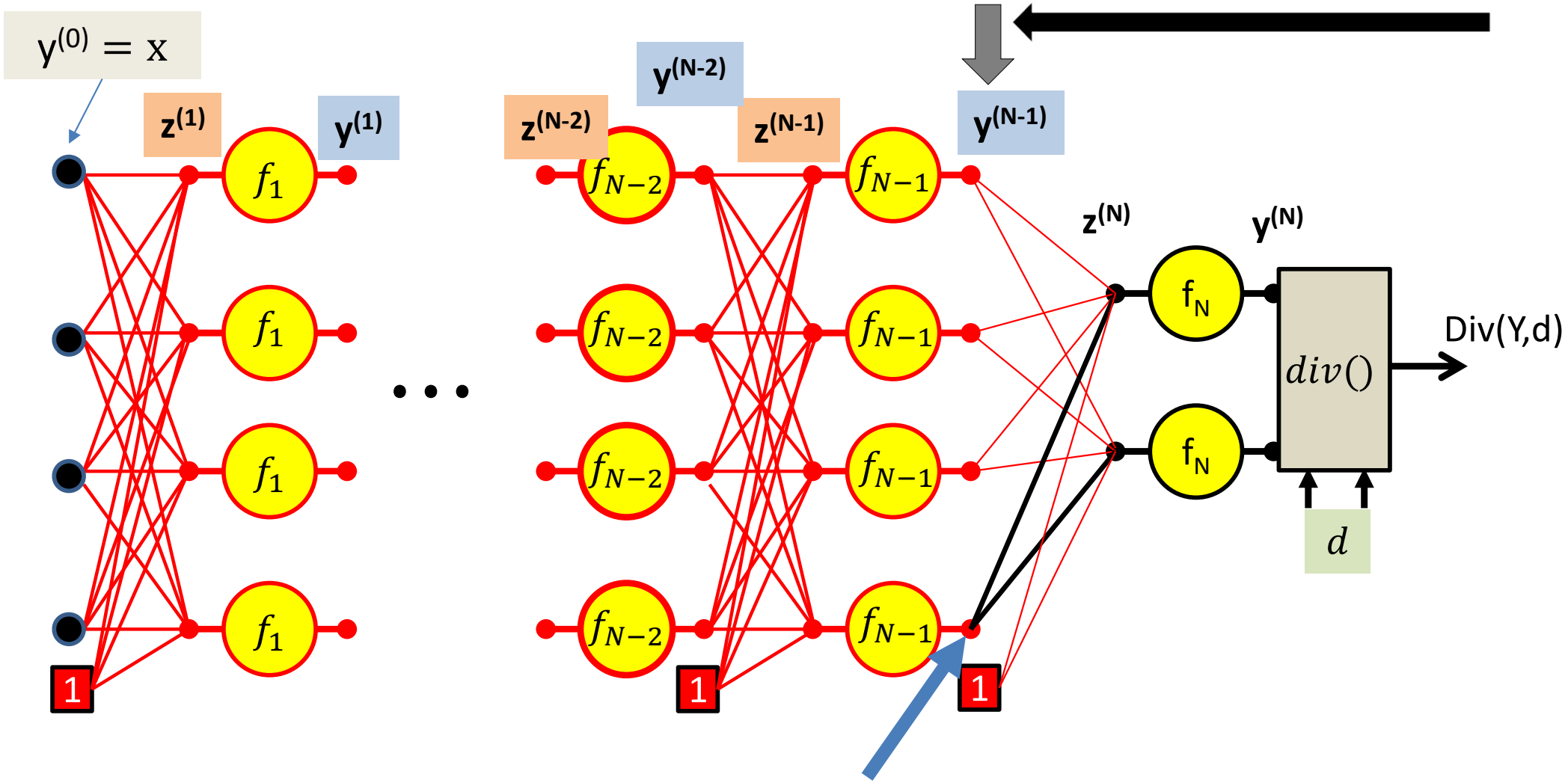

$$\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx} + \cdots + \frac{\partial l}{\partial z_M}\frac{dz_M}{dx}$$

# Computing derivatives




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

# Computing derivatives

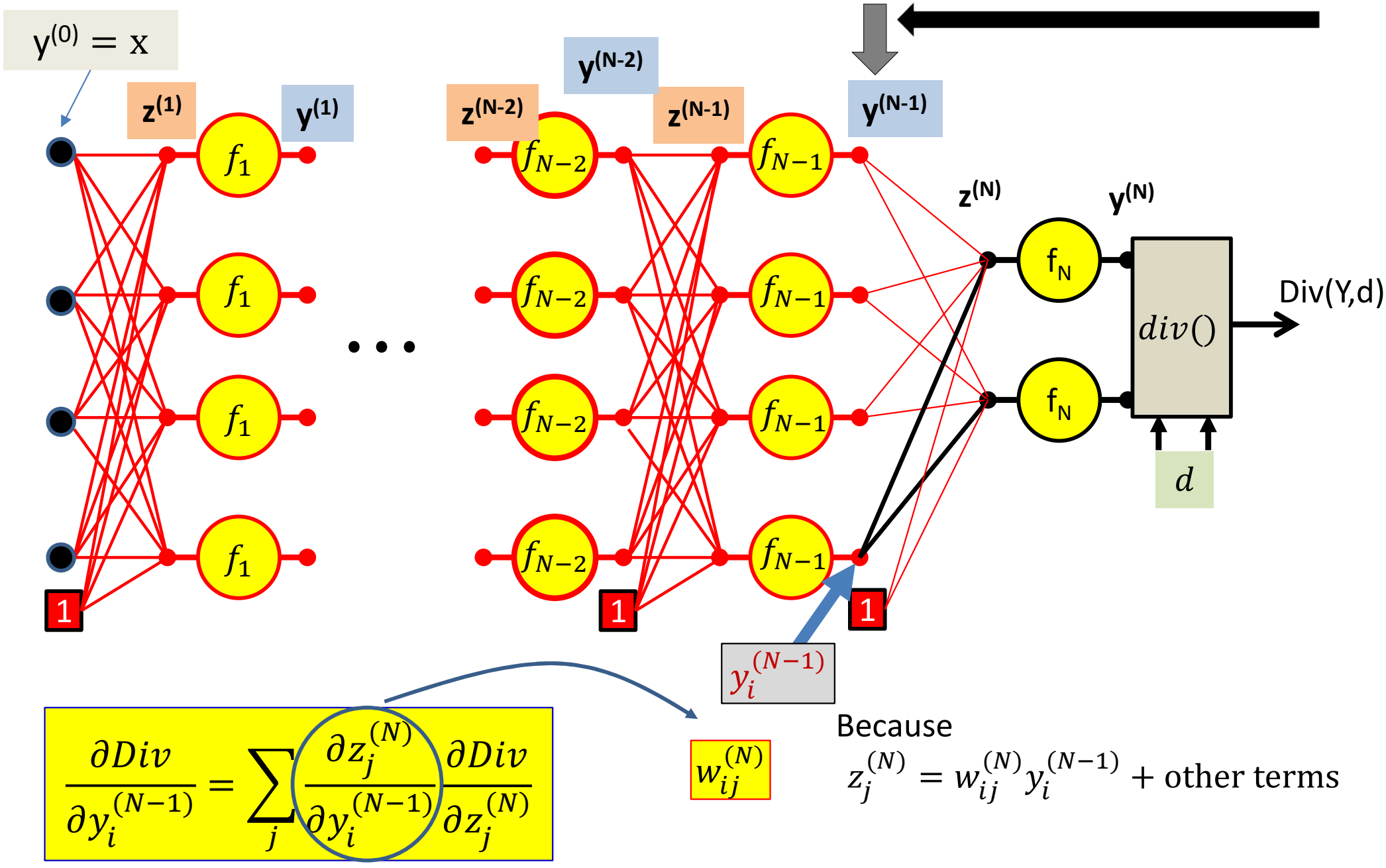



$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$

Already computed

# Computing derivatives

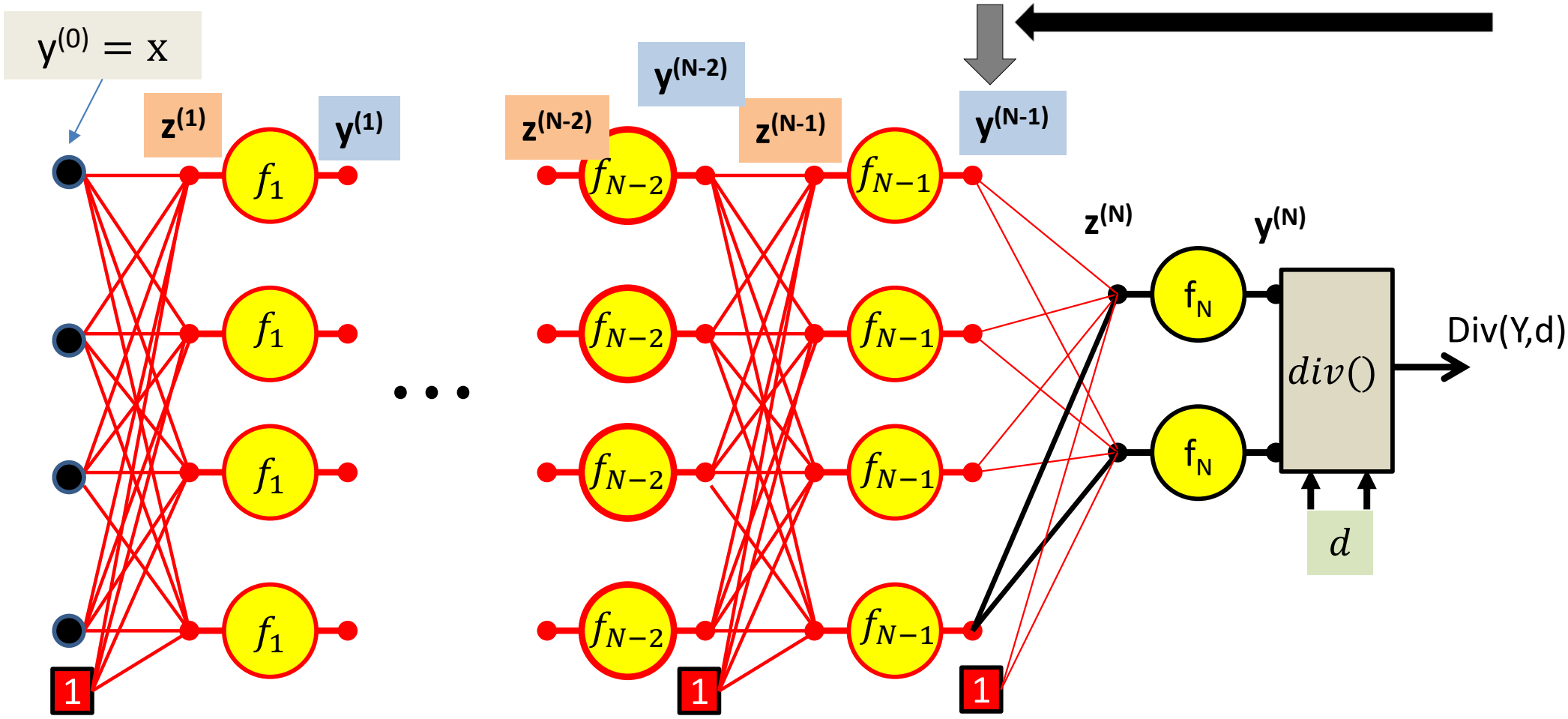



$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

Because

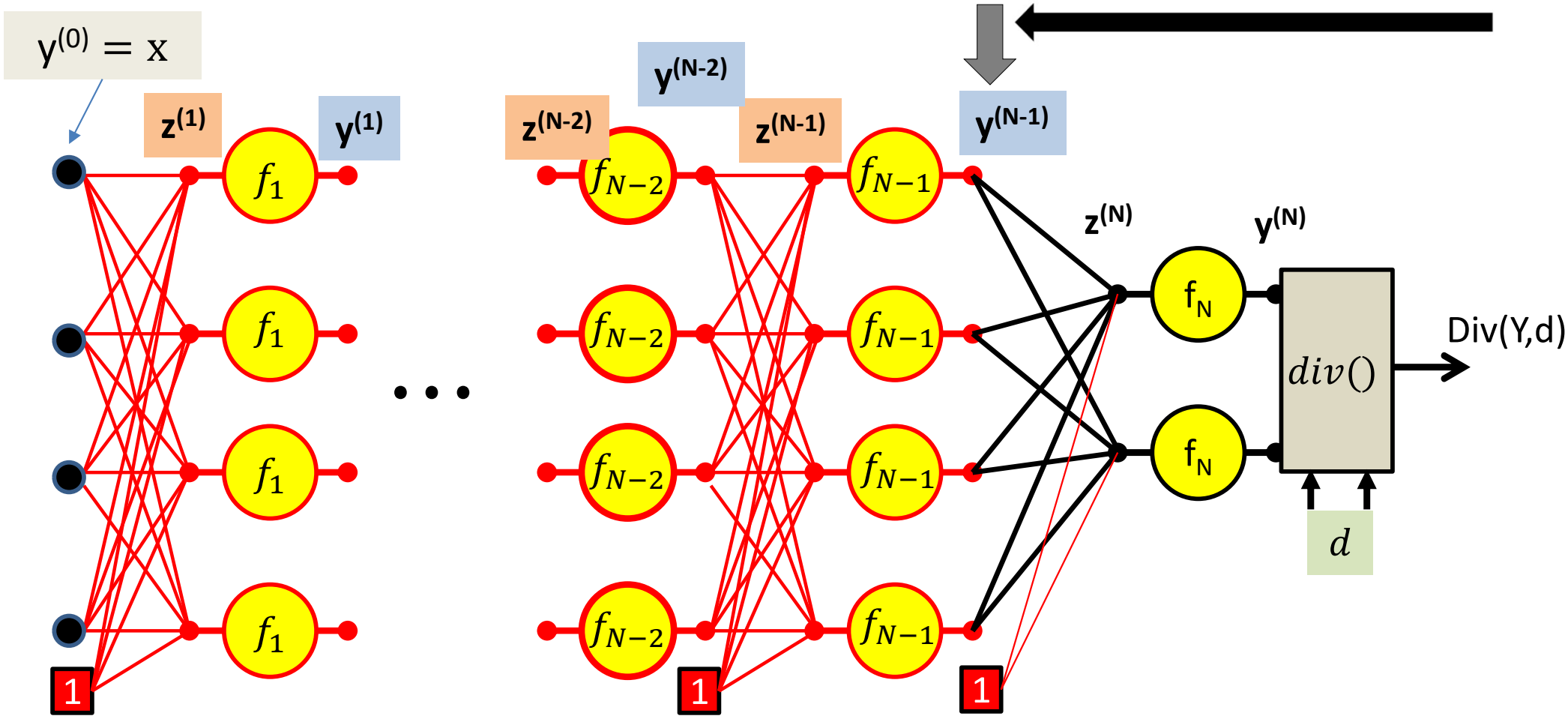
$$z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$$

# Computing derivatives



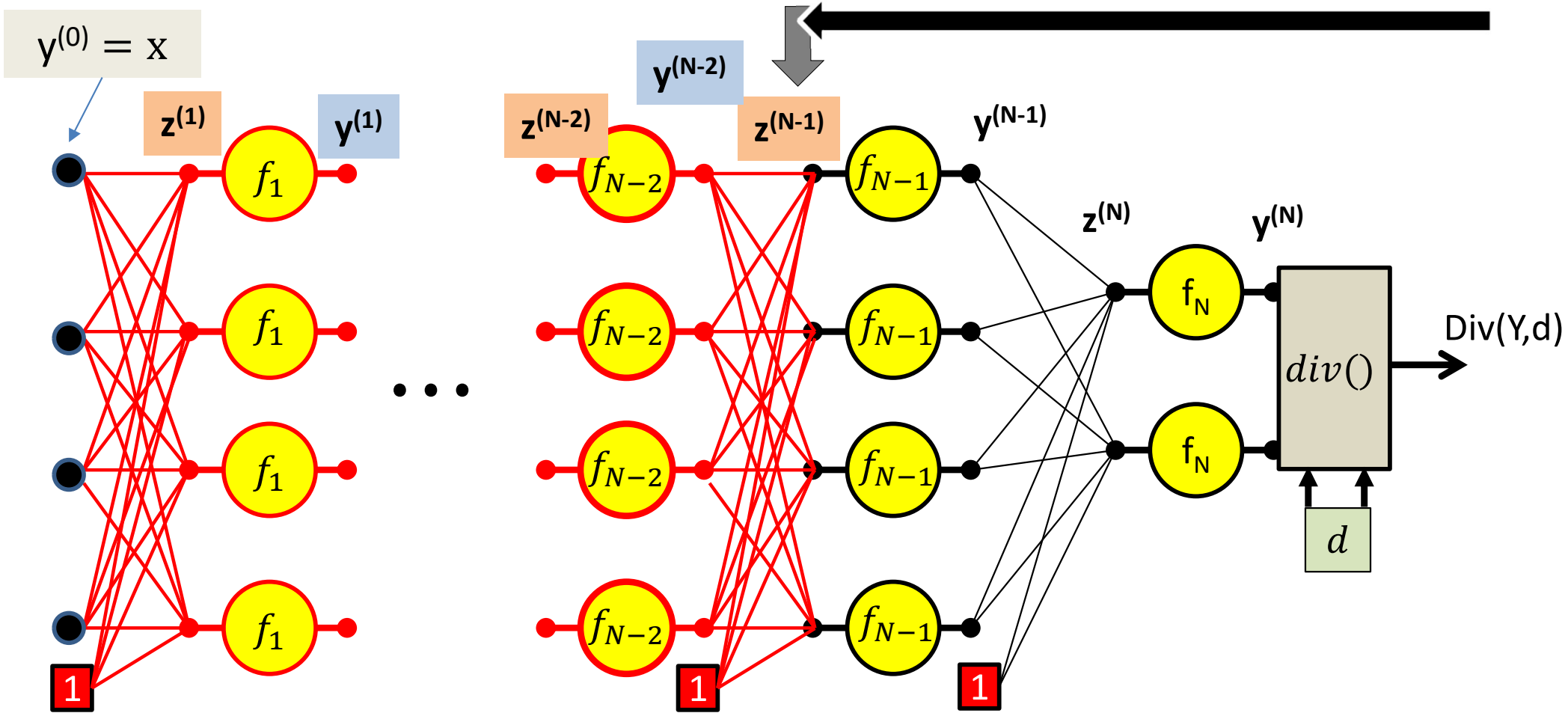

$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

# Computing derivatives



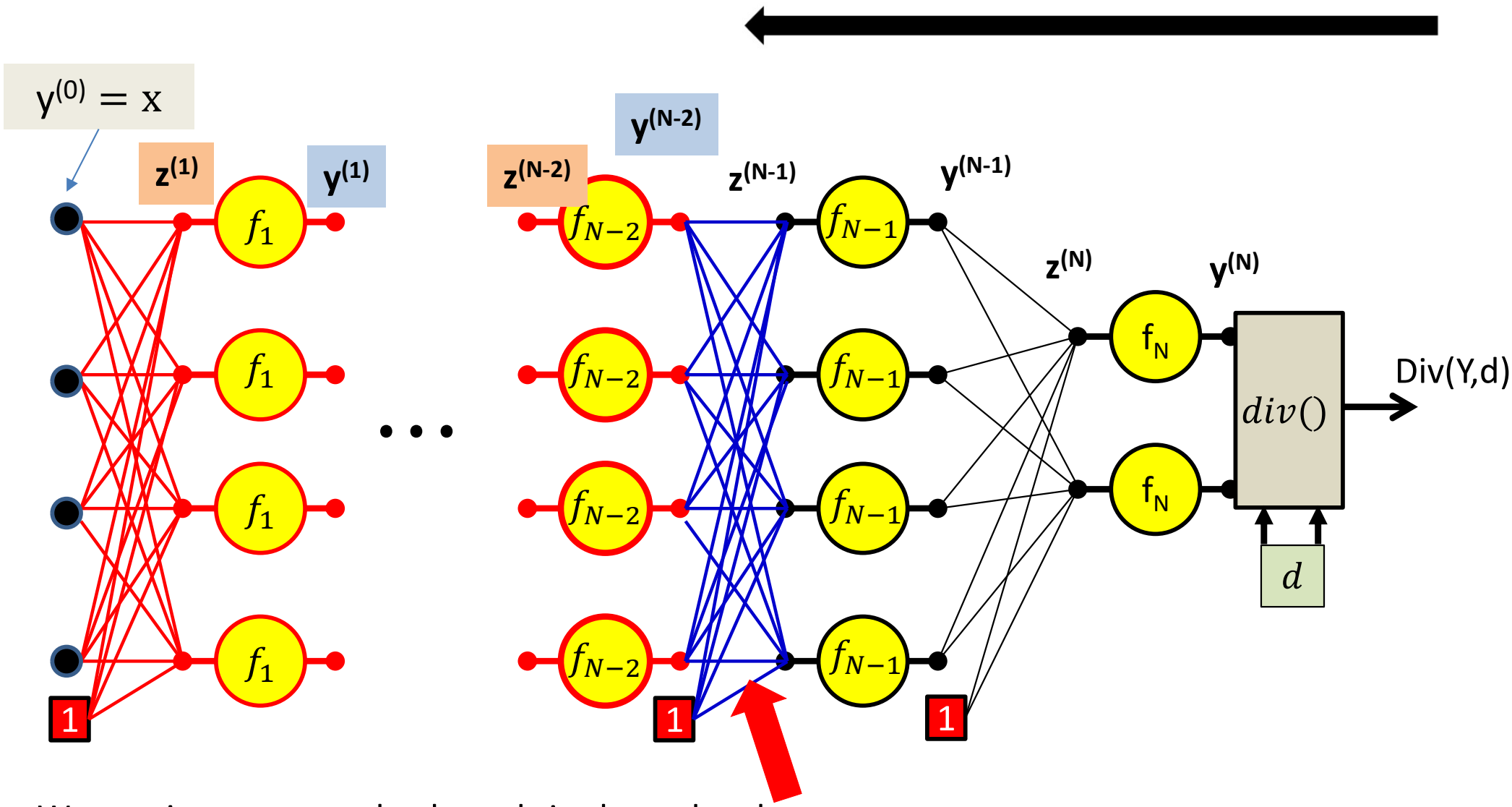

$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

# Computing derivatives




We continue our way backwards in the order shown

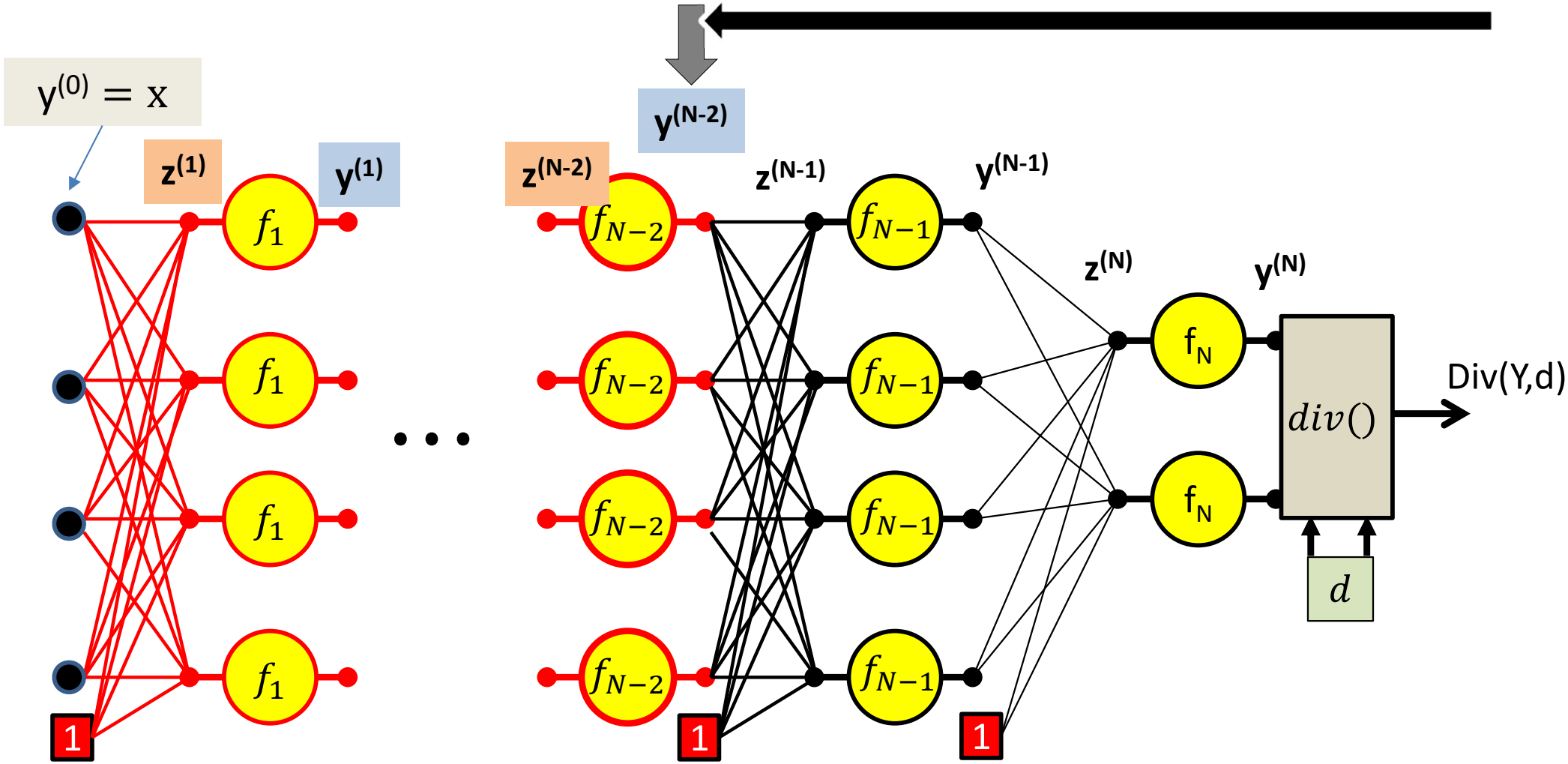
$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f'_{N-1}\left(z_i^{(N-1)}\right)\frac{\partial Div}{\partial y_i^{(N-1)}}$$




We continue our way backwards in the order shown

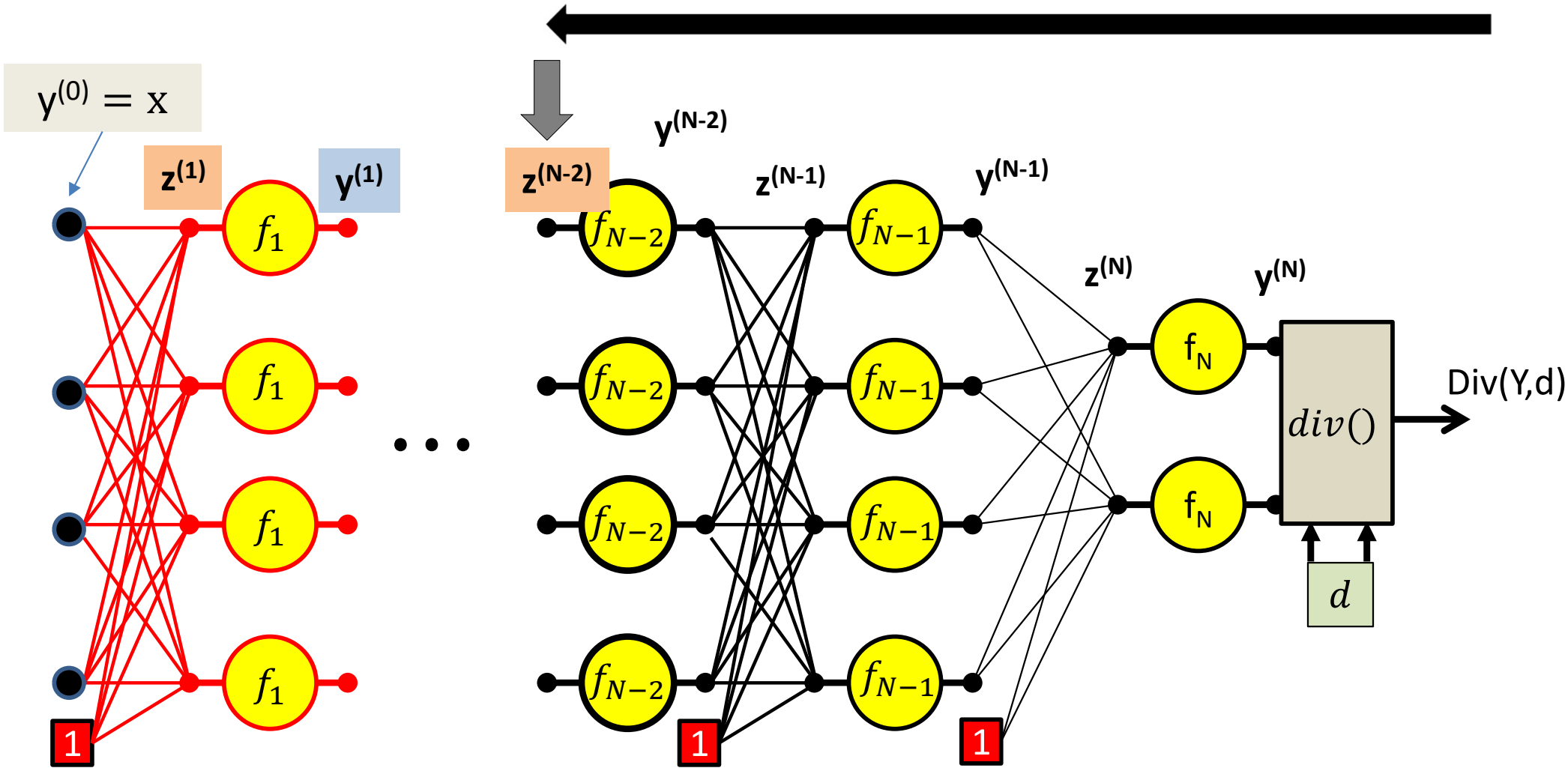
$$\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial Div}{\partial z_j^{(N-1)}}$$

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



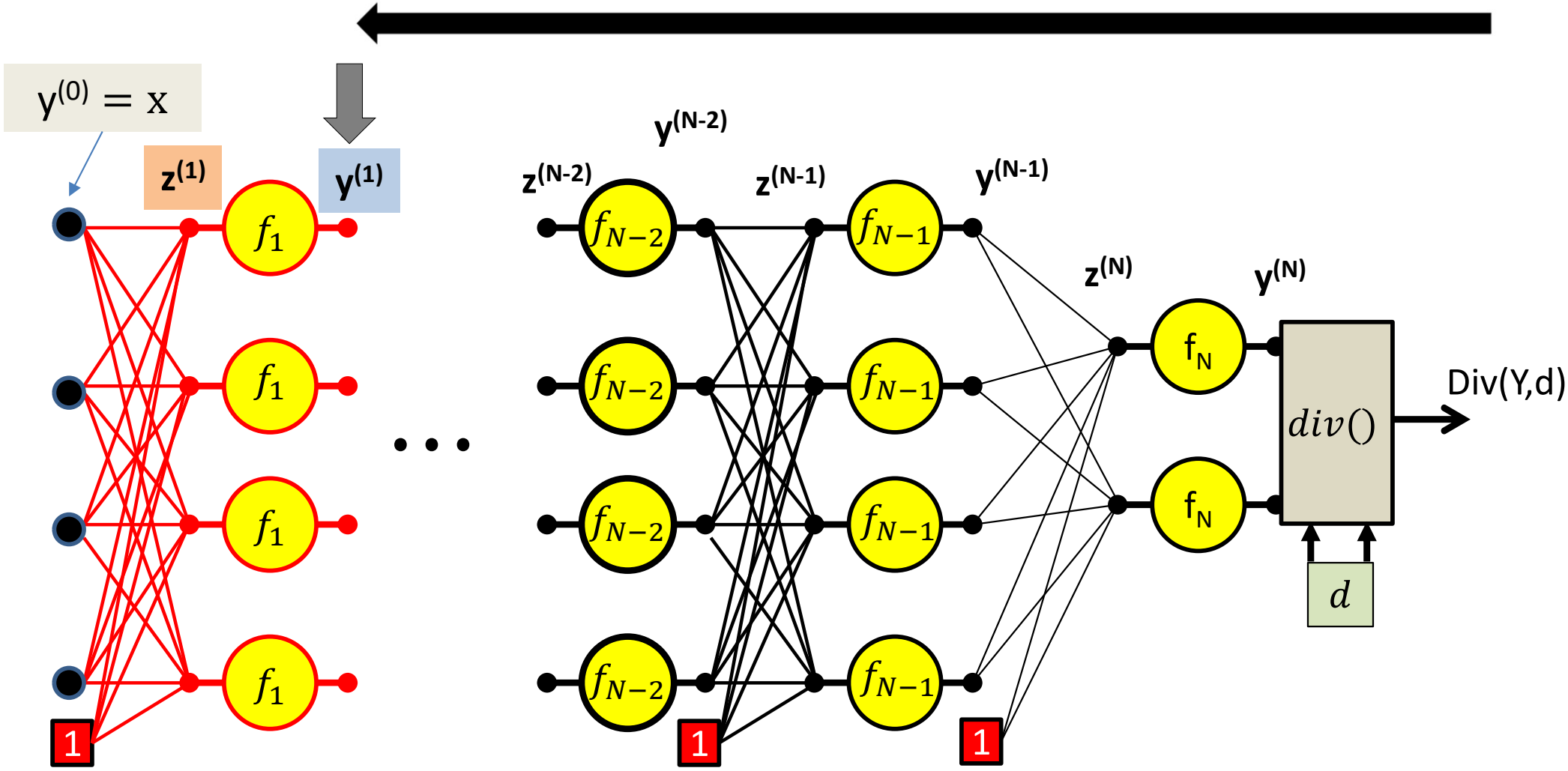

We continue our way backwards in the order shown

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



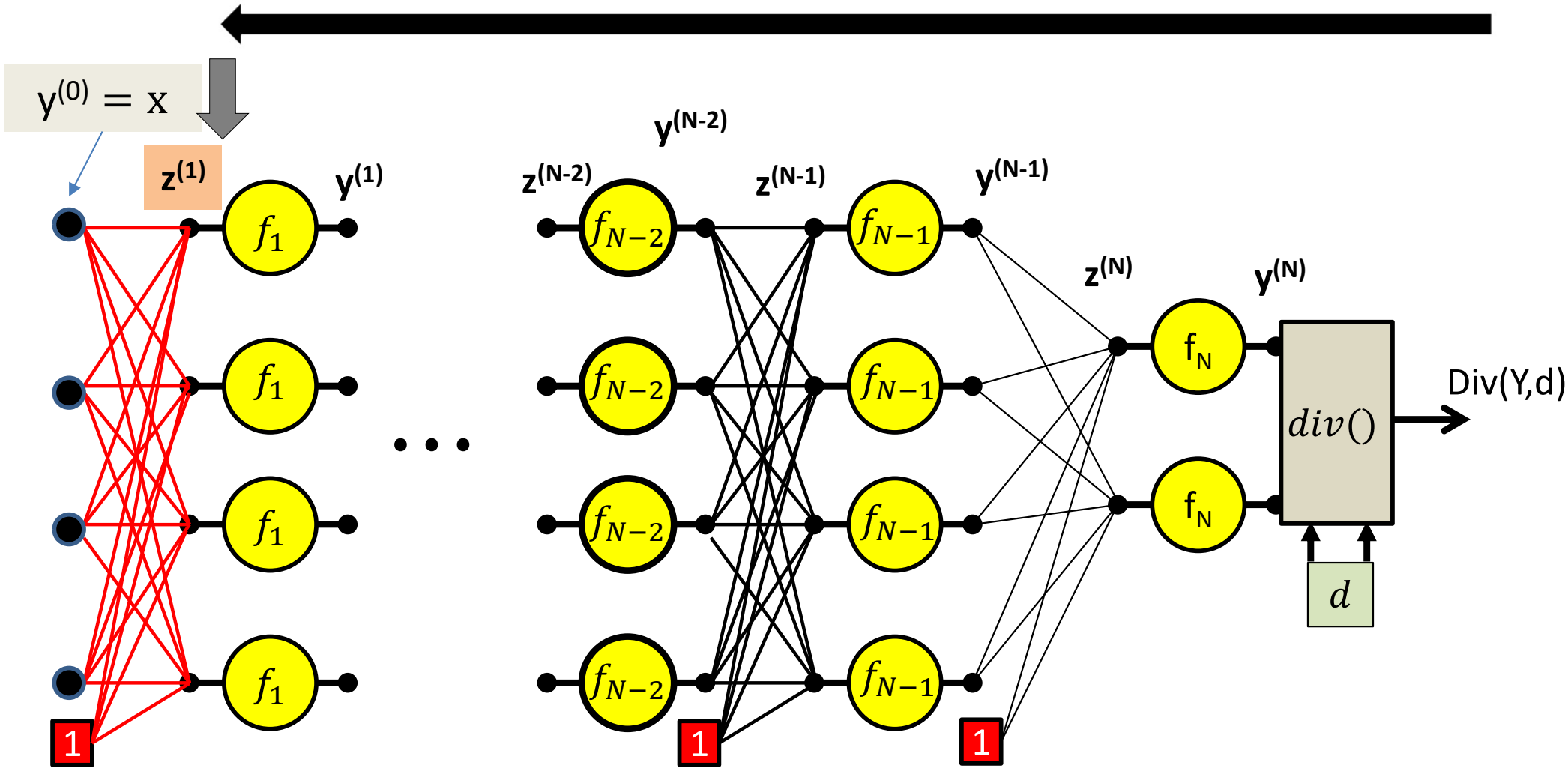

We continue our way backwards in the order shown

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



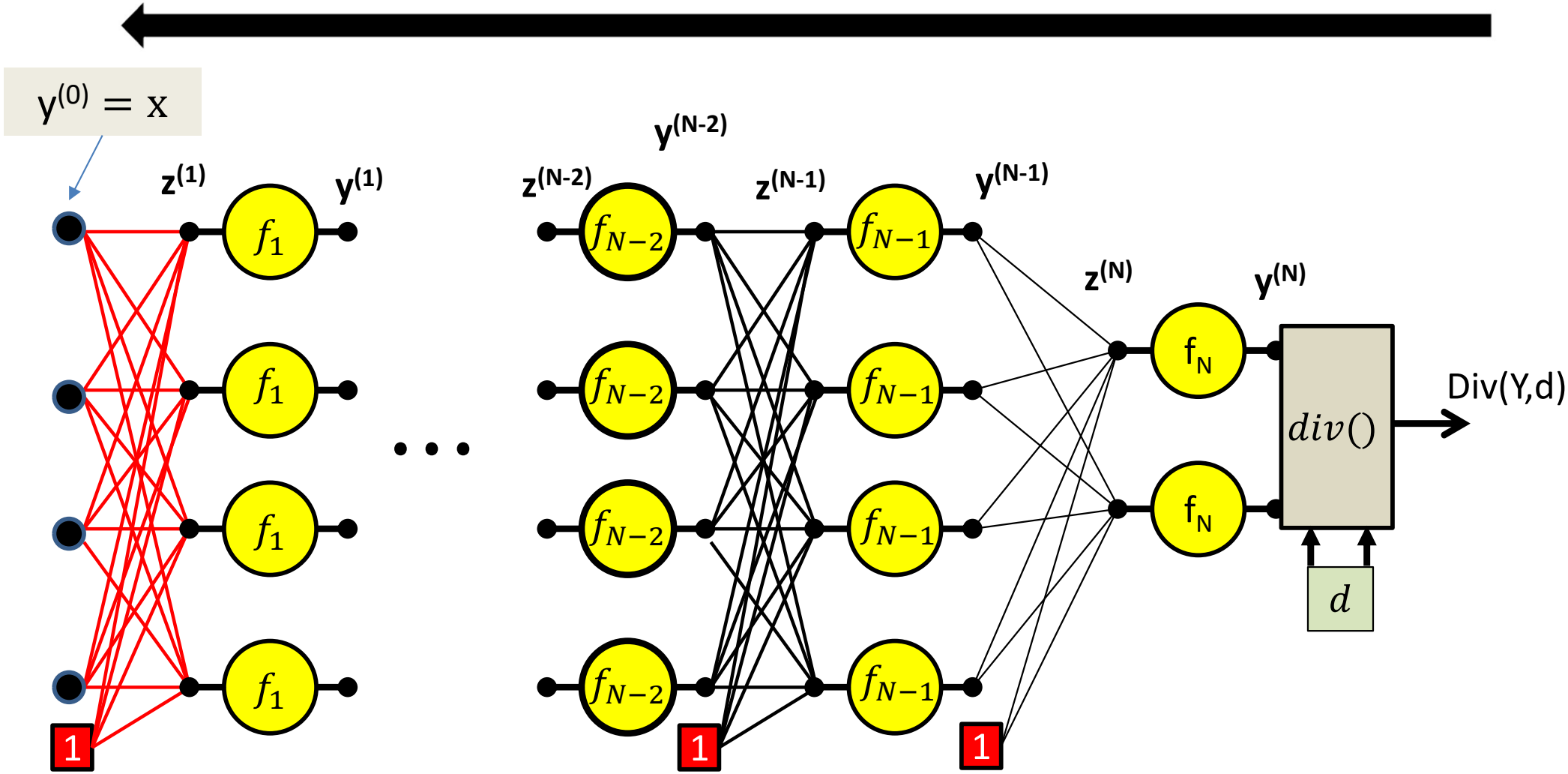

We continue our way backwards in the order shown

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




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

$$\frac{\partial Div}{\partial w_{ij}^{(1)}} = y_i^{(0)} \frac{\partial Div}{\partial z_j^{(1)}}$$

# Gradients: Backward Computation

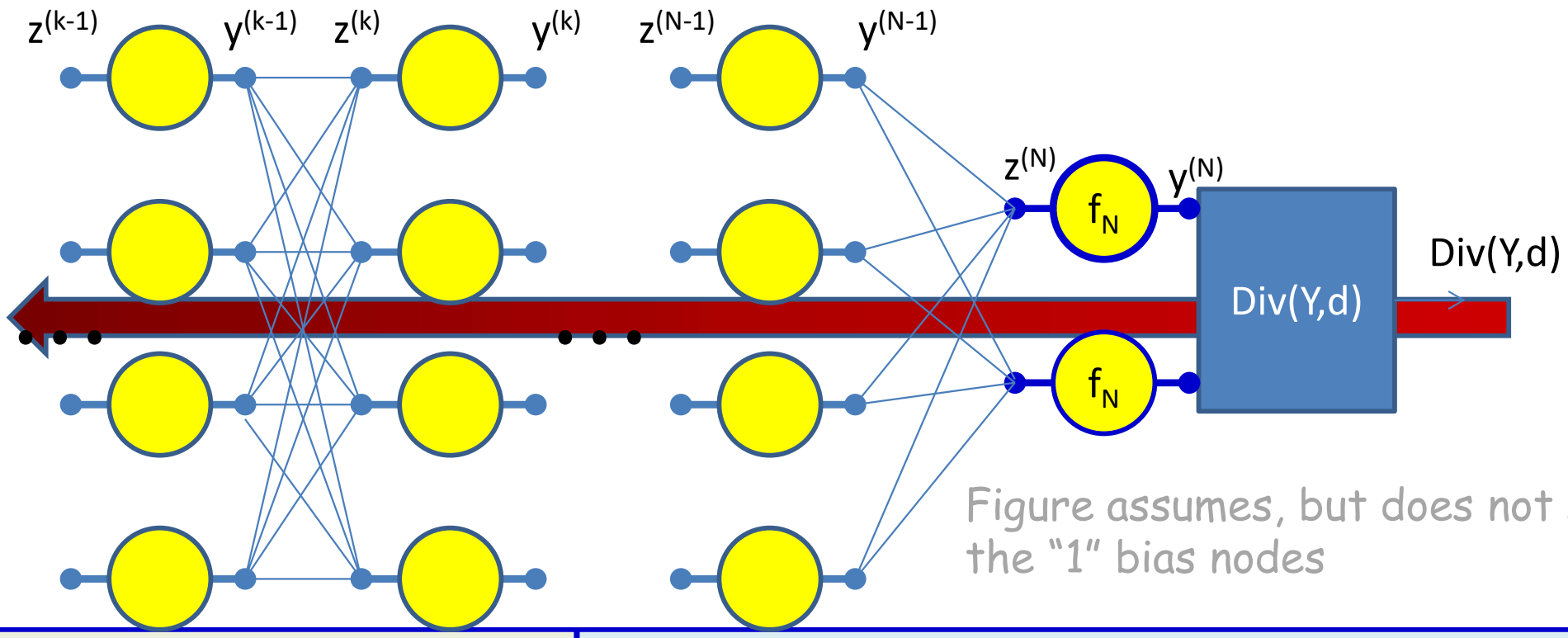



Figure assumes, but does not show the "1" bias nodes

Initialize: Gradient w.r.t network output

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

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

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

$For\ i = 1: layer\ width$

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

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

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

# Backward Pass

- Output layer $(N)$ :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$ [This is the derivative of the divergence]
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$ for $j = 0 \ldots D_{N-1}$
- For layer $k = N - 1 \; downto \; 1$
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial Div}{\partial z_j^{(k)}}$ for $j = 0 \ldots D_{k-1}$

# Backward Pass

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

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

Very analogous to the forward pass:

Backward weighted combination of next layer

Backward equivalent of activation

Using notation $\dot{y} = \frac{\partial Div(Y,d)}{\partial y}$ etc (overdot represents derivative of $Div$ w.r.t variable)

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\dot{y}_i^{(N)} = \frac{\partial Div}{\partial y_i}$
    - $\dot{z}_i^{(N)} = \dot{y}_i^{(N)} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ji}^{(N)}} = y_j^{(N-1)} \dot{z}_i^{(N)}$ for $j = 0 \ldots D_{N-1}$

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

- For layer $k = N-1$ *downto* $1$
  - For $i = 1 \ldots D_k$
    - $\dot{y}_i^{(k)} = \sum_j w_{ij}^{(k+1)} \dot{z}_j^{(k+1)}$
    - $\dot{z}_i^{(k)} = \dot{y}_i^{(k)} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Di}{\partial w_{ji}^{(k)}} = y_j^{(k-1)} \dot{z}_i^{(k)}$ for $j = 0 \ldots D_{k-1}$

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
  - $y_j^{(0)} = x_j,\ j = 1 \ldots D; \qquad y_0^{(k=1\ldots N)} = x_0 = 1$
- 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:
  - $Y = y_j^{(N)}, j = 1..D_N$

# Poll 2 : @384

How does backpropagation relate to training the network (pick one)

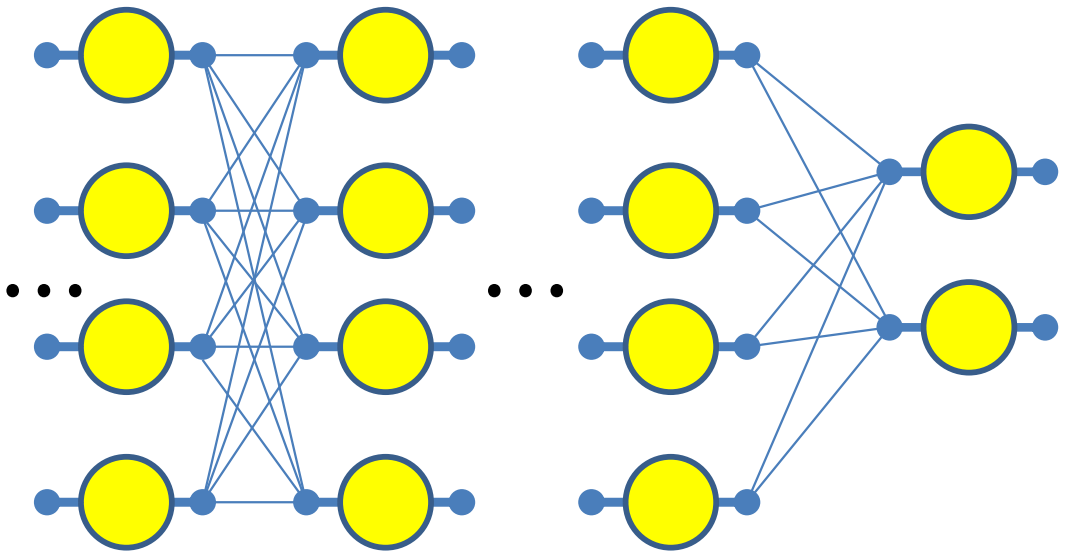
- Backpropagation is the process of training the network
- Backpropagation is used to update the model parameters during training
- Backpropagation is used to compute the derivatives of the divergence with respect to model parameters, to be used in gradient descent.

# Poll 2

How does backpropagation relate to training the network (pick one)

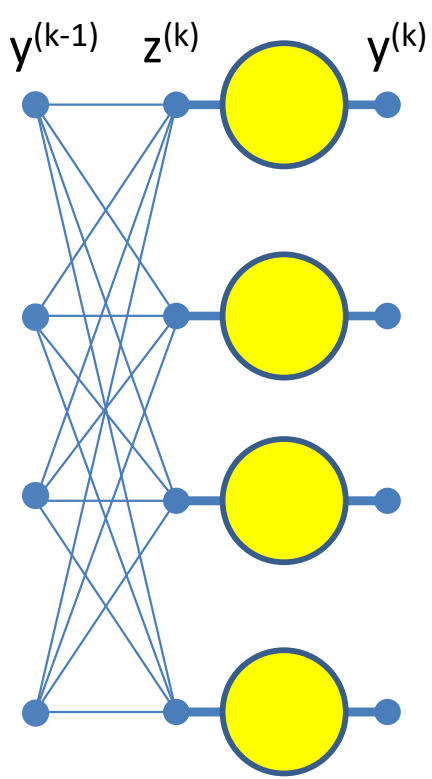
- Backpropagation is the process of training the network
- Backpropagation is used to update the model parameters during training
- Backpropagation is used to compute the derivatives of the divergence with respect to model parameters, to be used in gradient descent. **(correct)**

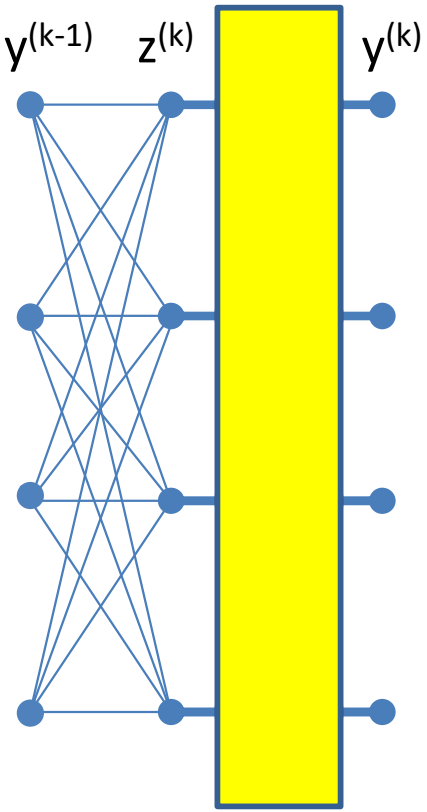
# Special cases



- Have assumed so far that
  1. The computation of the output of one neuron does not directly affect computation of other neurons in the same (or previous) layers
  2. Inputs to neurons only combine through weighted addition
  3. Activations are actually differentiable
  - All of these conditions are frequently not applicable
- Will not discuss all of these in class, but explained in slides
  - Will appear in quiz. Please read the slides

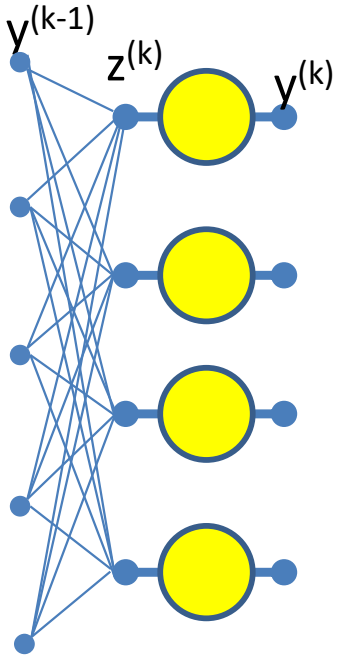
# Special Case 1. Vector activations

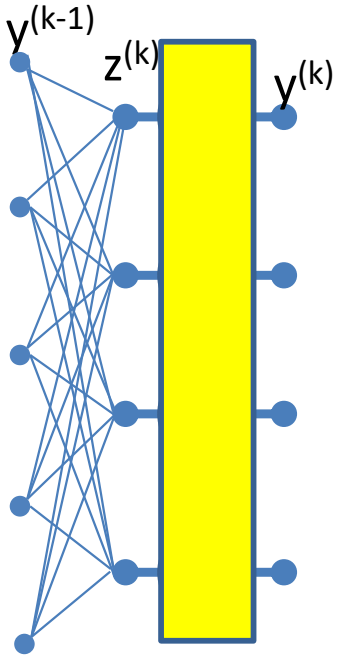



- Vector activations: all outputs are functions of all inputs

# Special Case 1. Vector activations



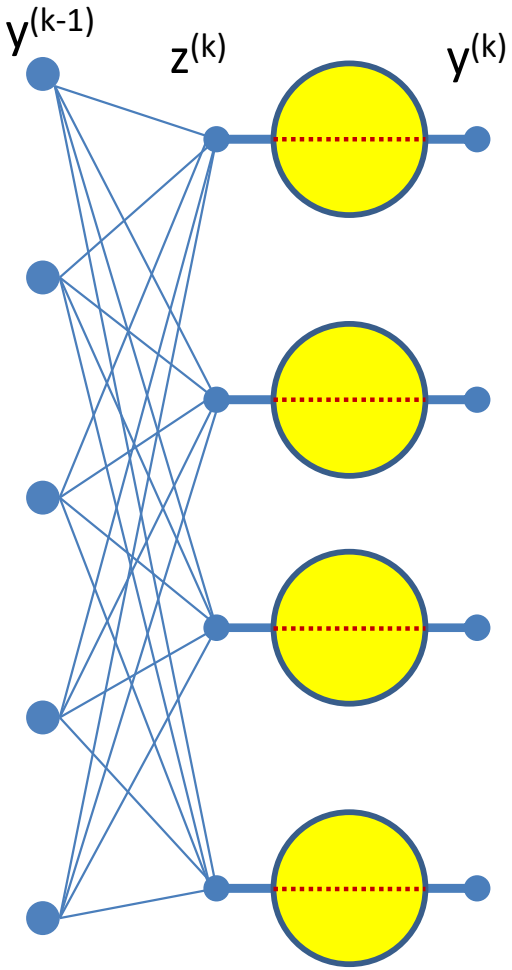

Scalar activation: Modifying a $z_i$ only changes corresponding $y_i$

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

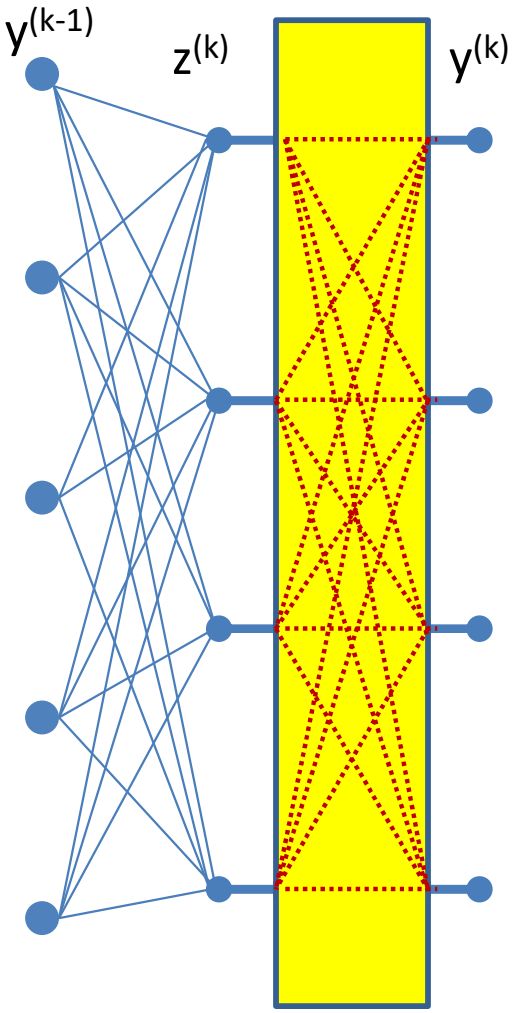
Vector activation: Modifying a $z_i$ potentially changes all, $y_1 \ldots y_M$

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

# “Influence” diagram

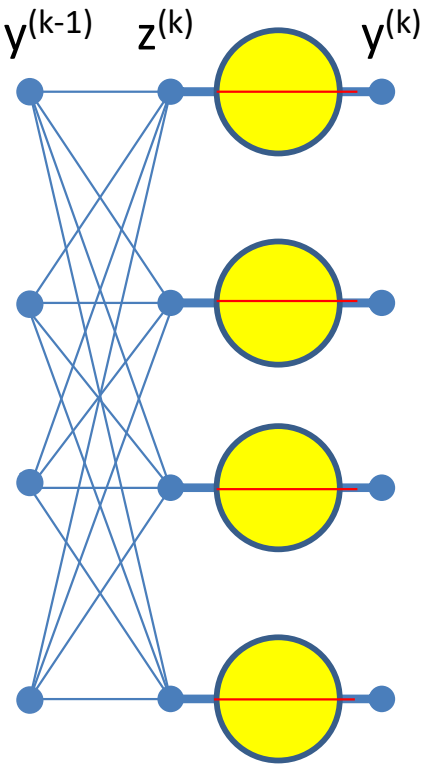



Scalar activation: Each $z_i$ influences *one* $y_i$




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

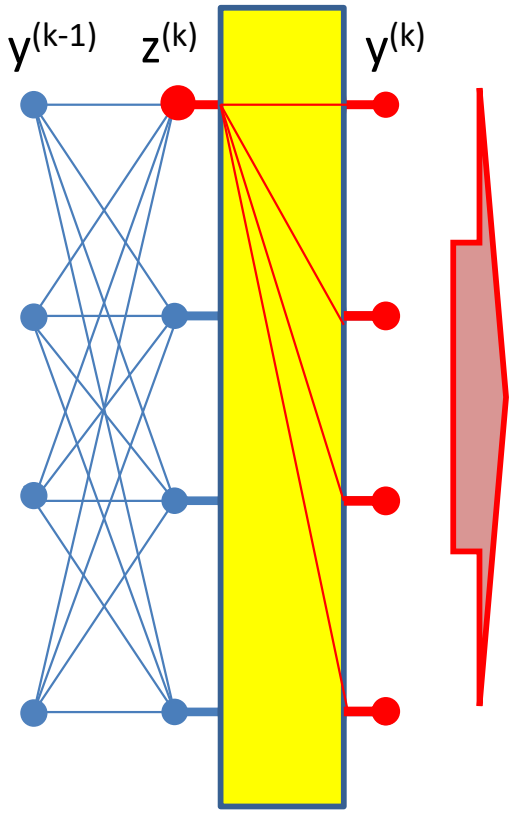
# Scalar Activation: Derivative rule




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

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

# Derivatives of vector activation




**Div**

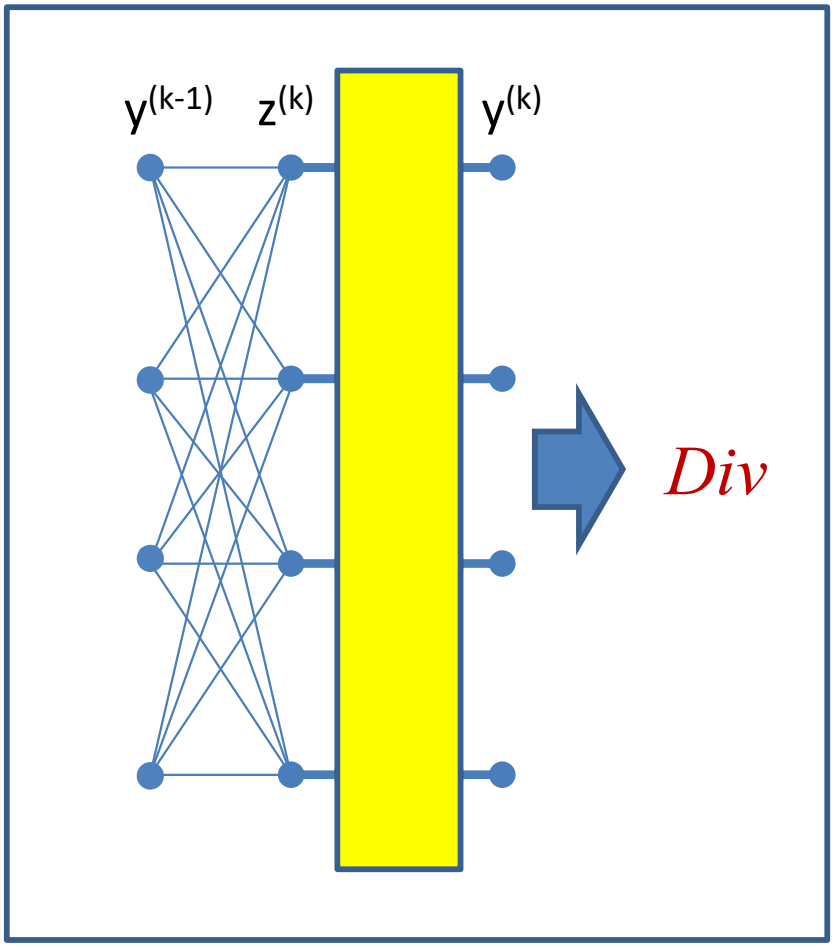
$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

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

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

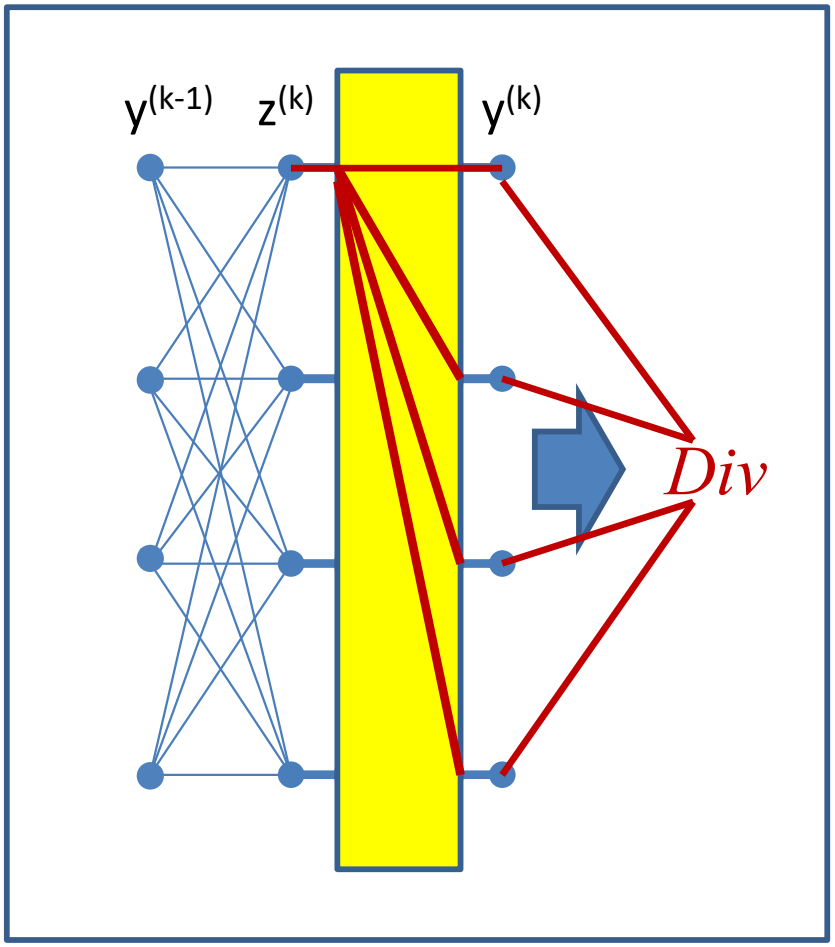
- For *vector* activations the derivative of the loss w.r.t. to any input is a sum of partial derivatives
  - Regardless of the number of outputs $y_j^{(k)}$

# Example Vector Activation: Softmax




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

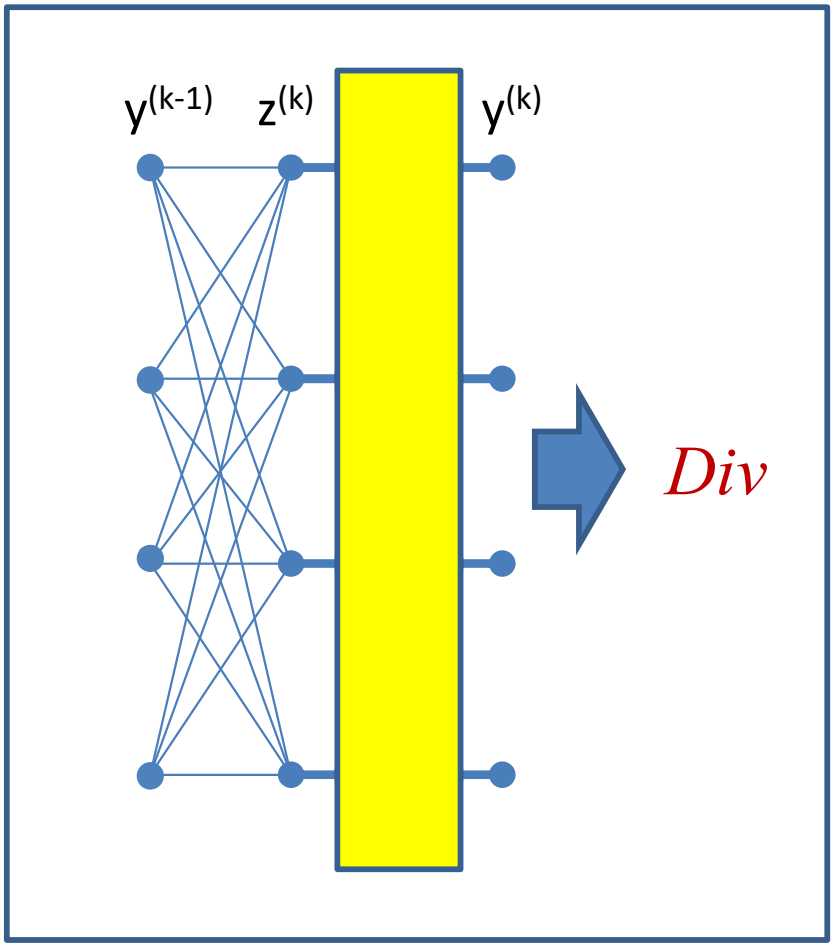
# Example Vector Activation: Softmax




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

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

# Example Vector Activation: Softmax

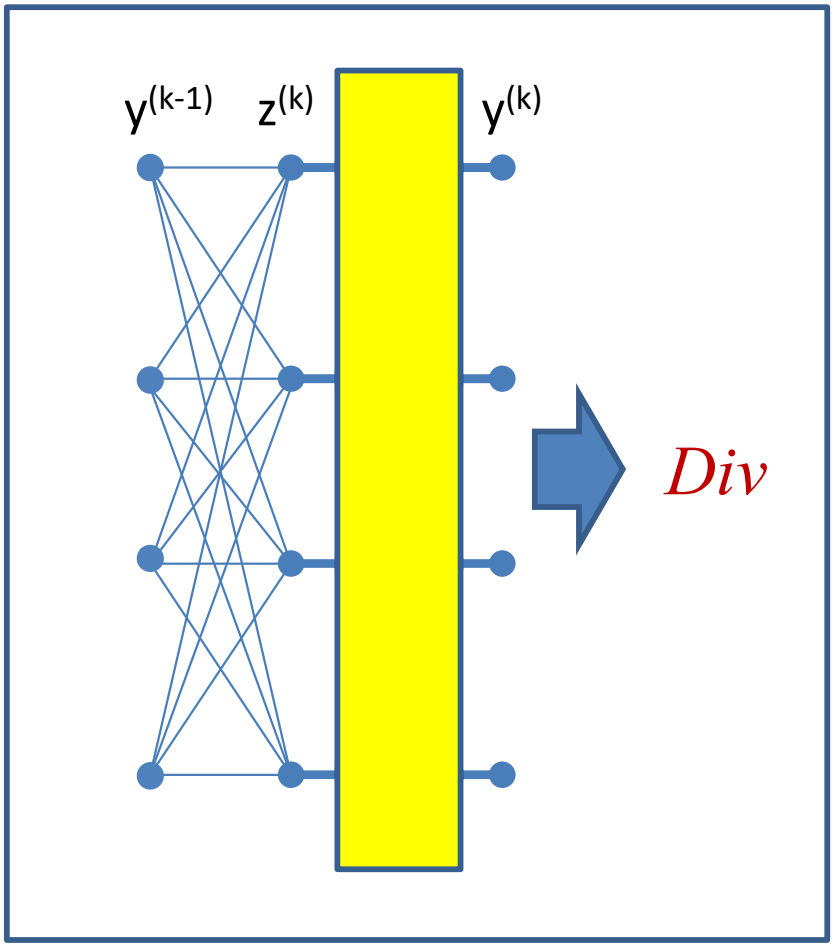



$$y_i^{(k)} = \frac{exp\left(z_i^{(k)}\right)}{\sum_j exp\left(z_j^{(k)}\right)}$$

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

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

# Example Vector Activation: Softmax




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

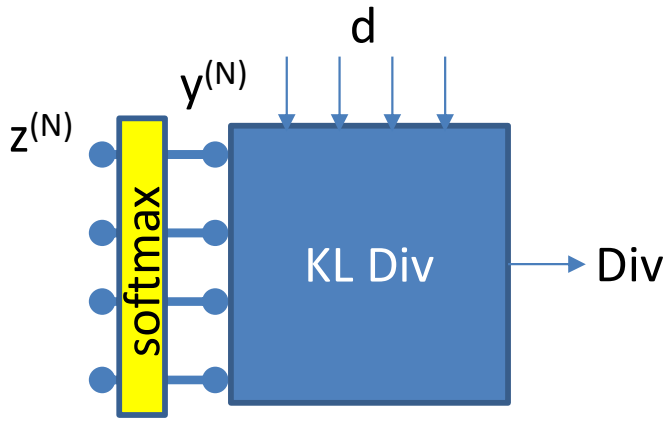
$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

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

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

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

# Backward Pass for *softmax output layer*

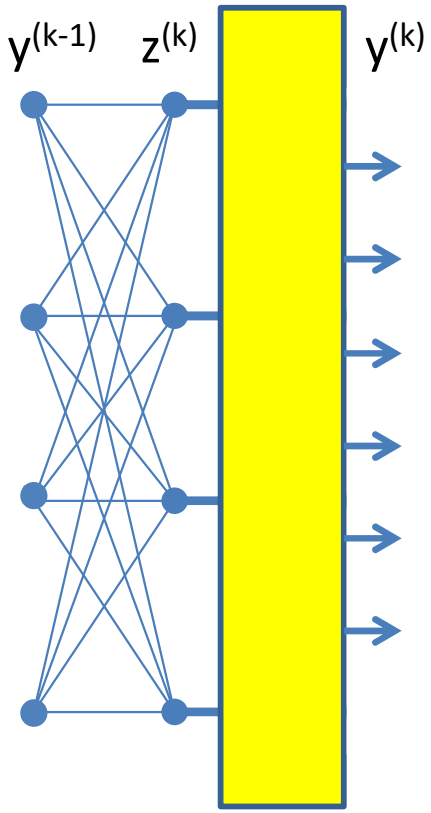



- Output layer $(N)$ :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Di\ (Y,d)}{\partial y_i}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \sum_j \frac{\partial Div(Y,d)}{\partial y_j^{(N)}} y_i^{(N)} \left(\delta_{ij} - y_j^{(N)}\right)$
    - $\frac{\partial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

# Special cases

- Examples of vector activations and other special cases on slides
  - Please look up
  - Will appear in quiz!

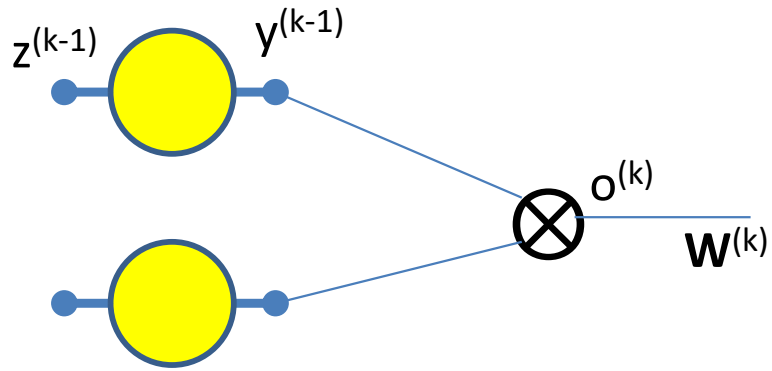
# Vector Activations




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

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

# Special Case 2: Multiplicative networks

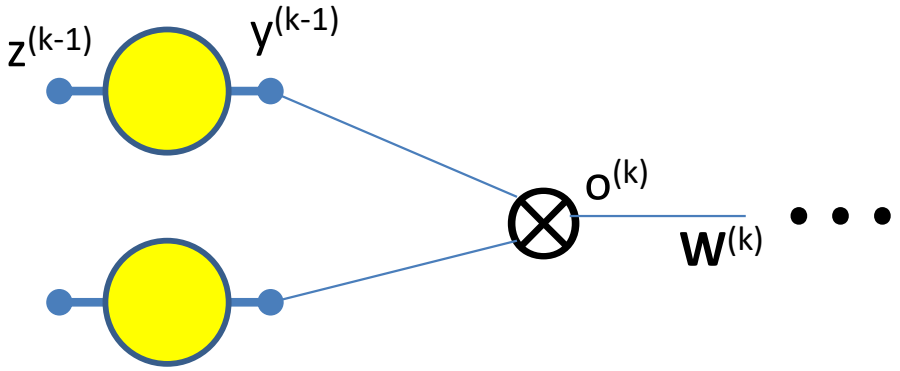



• • •

Forward: $o_i^{(k)} = y_j^{(k-1)} y_l^{(k-1)}$

- Some types of networks have *multiplicative* combination
  - In contrast to the *additive* combination we have seen so far
- Seen in networks such as LSTMs, GRUs, attention models, etc.

# Backpropagation: Multiplicative Networks




Forward:

$$o_i^{(k)} = y_j^{(k-1)} y_l^{(k-1)}$$

Backward:

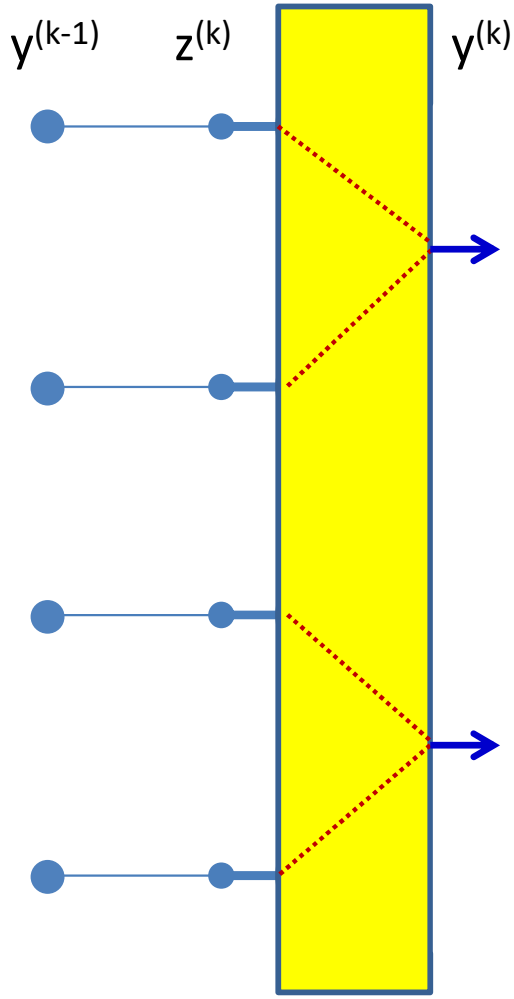
$$\frac{\partial Div}{\partial o_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$$

$$\frac{\partial Div}{\partial y_j^{(k-1)}} = \frac{\partial o_i^{(k)}}{\partial y_j^{(k-1)}} \frac{\partial Div}{\partial o_i^{(k)}} = y_l^{(k-1)} \frac{\partial Div}{\partial o_i^{(k)}}$$

$$\frac{\partial Div}{\partial y_l^{(k-1)}} = y_j^{(k-1)} \frac{\partial Div}{\partial o_i^{(k)}}$$

- Some types of networks have *multiplicative* combination

# Multiplicative combination as a case of vector activations

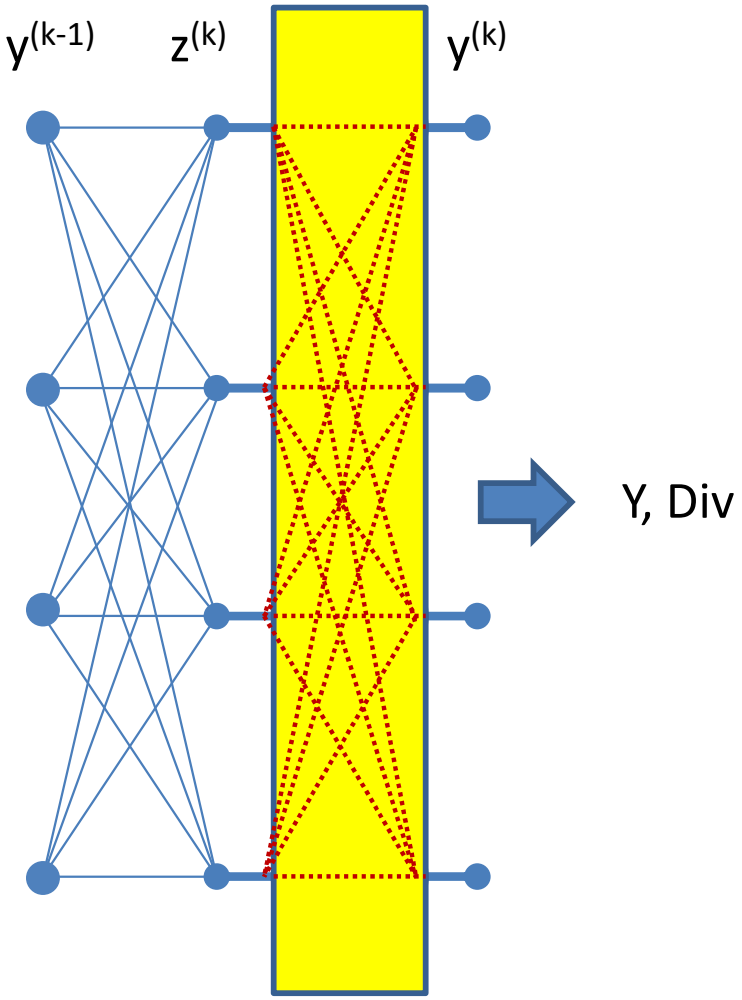



$$z_i^{(k)} = y_i^{(k-1)}$$

$$y_i^{(k)} = z_{2i-1}^{(k)} z_{2i}^{(k)}$$

- A layer of multiplicative combination is a special case of vector activation

# Multiplicative combination: Can be viewed as a case of vector activations




$$z_i^{(k)} = \sum_j w_{ji}^{(k)} y_j^{(k-1)}$$

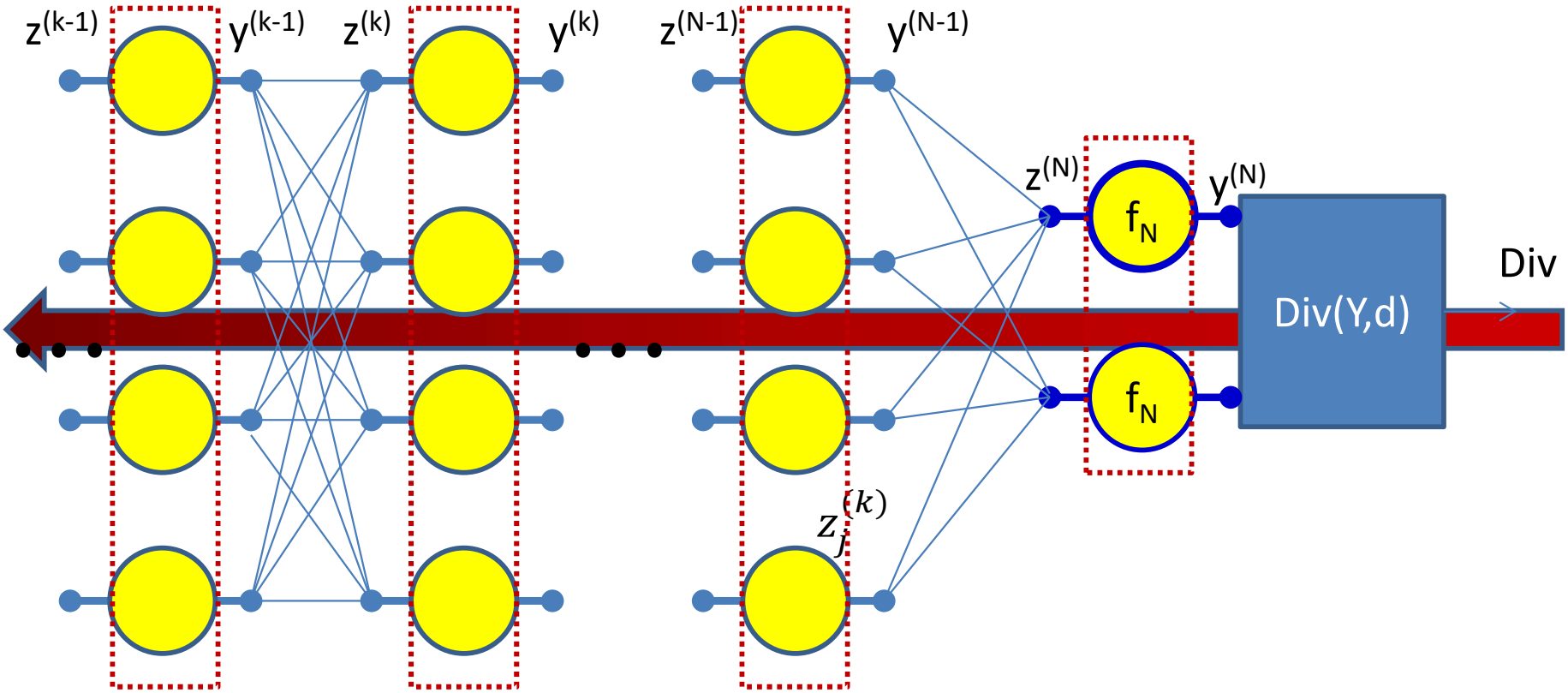
$$y_i^{(k)} = \prod_l \left(z_l^{(k)}\right)^{\alpha_{li}^{(k)}}$$

$$\frac{\partial y_i^{(k)}}{\partial z_j^{(k)}} = \alpha_{ji}^{(k)} \left(z_j^{(k)}\right)^{\alpha_{ji}^{(k)}-1} \prod_{l \neq j} \left(z_l^{(k)}\right)^{\alpha_{li}^{(k)}}$$

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

- A layer of multiplicative combination is a special case of vector activation

# Gradients: Backward Computation




For $k = N...1$

For $i = 1:layer\ width$

If layer has vector activation

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

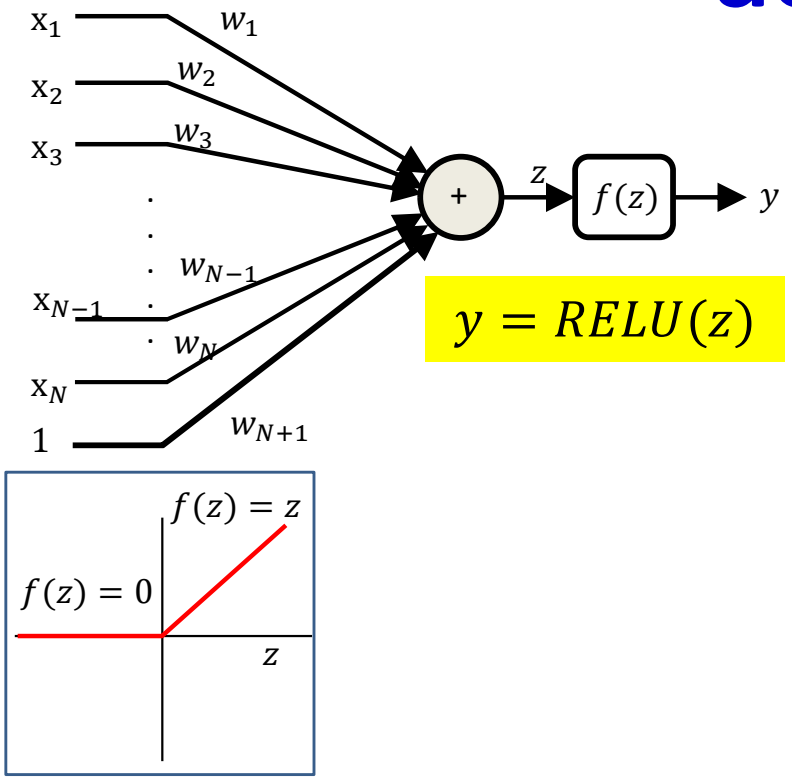
$\Leftrightarrow$ Else if activation is scalar

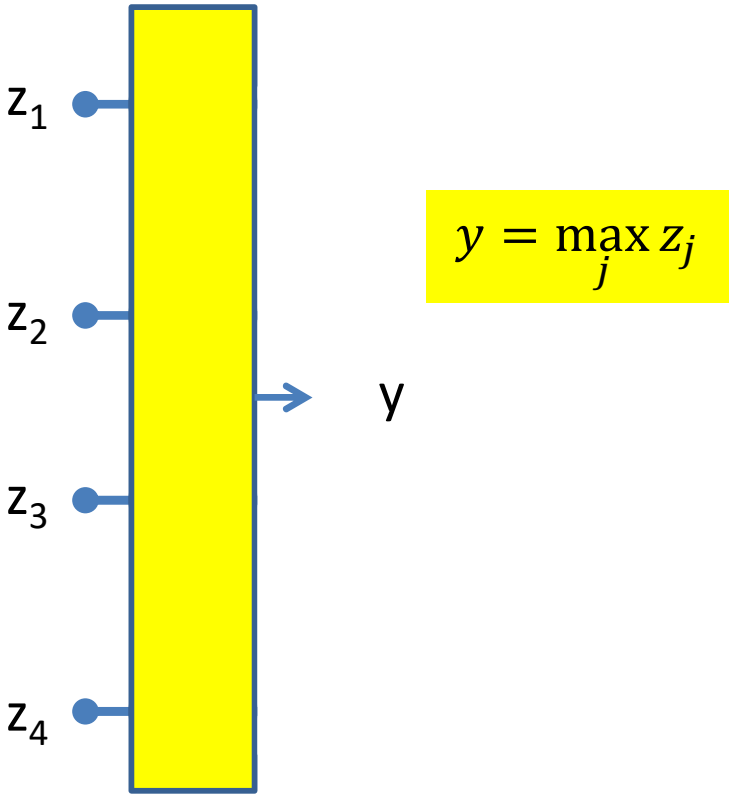
$$\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$$

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

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

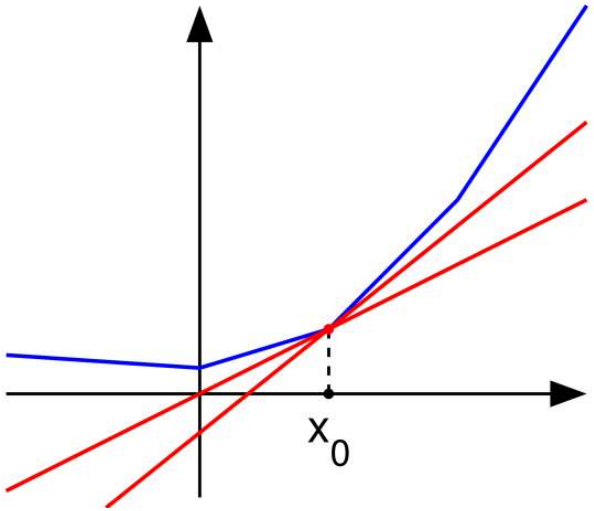
# Special Case : Non-differentiable activations




- Activation functions are sometimes not actually differentiable
  - E.g. The RELU (Rectified Linear Unit)
    - And its variants: leaky RELU, randomized leaky RELU
  - E.g. The "max" function
- Must use "subgradients" where available
  - Or "secants"

# The subgradient



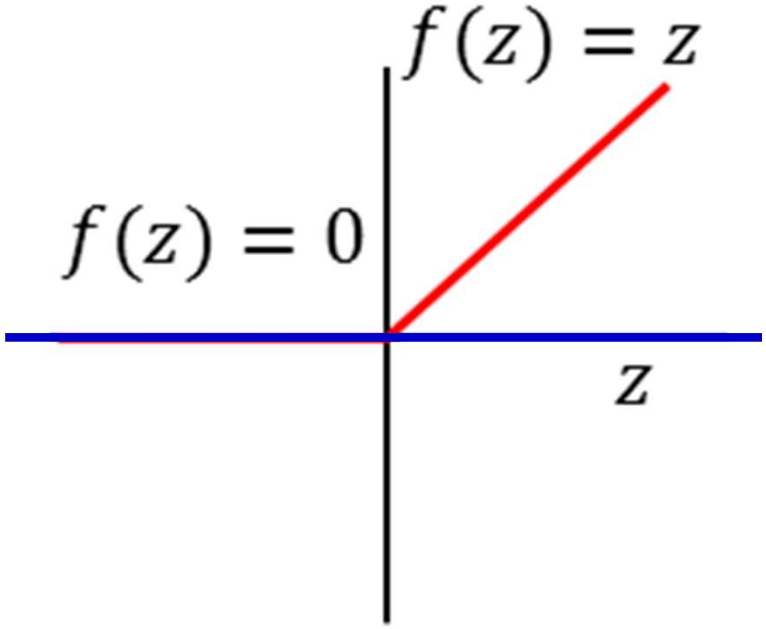

- A subgradient of a function $f(x)$ at a point $x_0$ is any vector $v$ such that

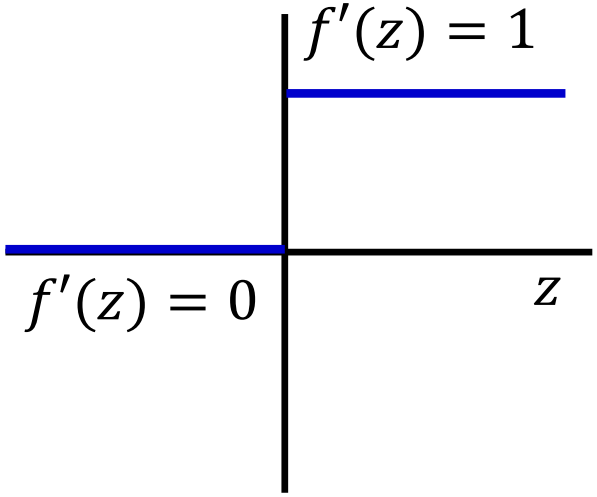
$$(f(x) - f(x_0)) \geq v^T(x - x_0)$$

  - Any direction such that moving in that direction increases the function
- Guaranteed to exist only for convex functions
  - "bowl" shaped functions
  - For non-convex functions, the equivalent concept is a "quasi-secant"

- The subgradient is a direction in which the function is guaranteed to increase
- If the function is differentiable at $x_0$, the subgradient is the gradient
  - The gradient is not always the subgradient though

# Non-differentiability: RELU



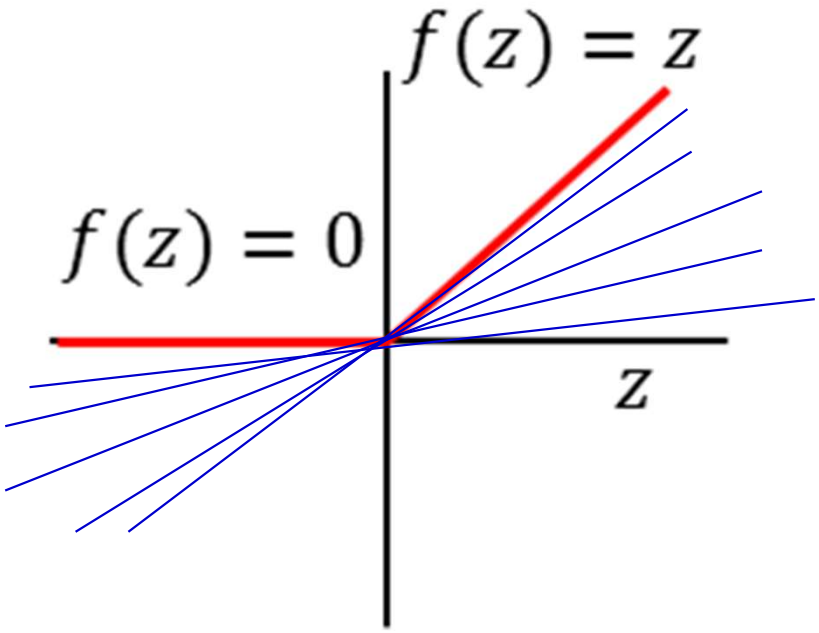



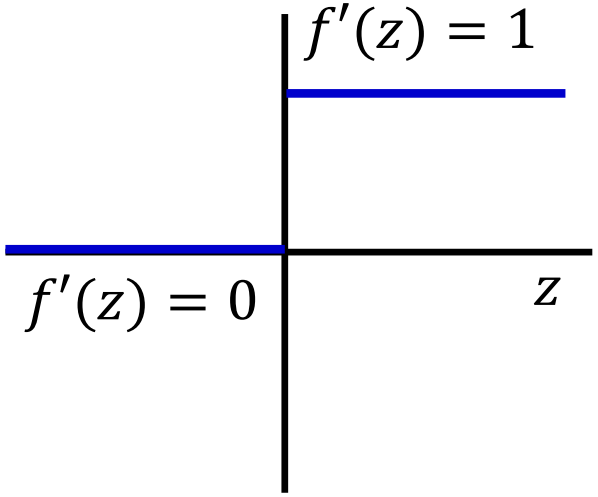

$$f'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

$$\Delta f(z) = \alpha \Delta z$$

- At 0 a *negative* perturbation $\Delta z < 0$ results in no change of $f(z)$
  - $\alpha = 0$
- A *positive* perturbation $\Delta z > 0$ results in $\Delta f(z) = \Delta z$
  - $\alpha = 1$
- Peering very closely, we can imagine that the curve is rotating continuously from slope = 0 to slope = 1 at $z = 0$
  - So any slope between 0 and 1 is valid

# Subgradients and the RELU

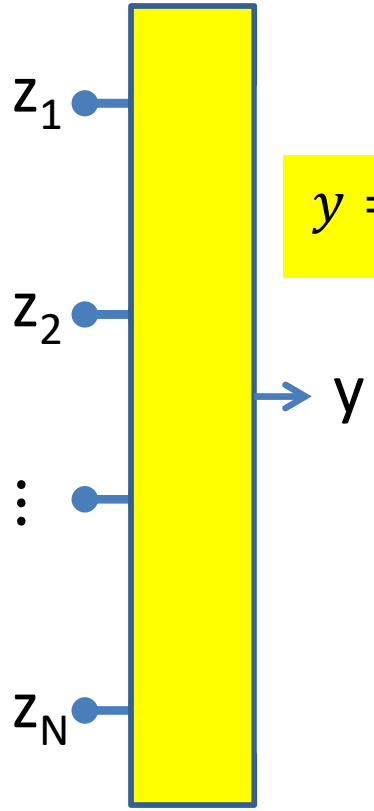



$$f'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

- The *subderivative* of a RELU is the slope of any line that lies entirely under it
  - The subgradient is a generalization of the subderivative
  - At the differentiable points on the curve, this is the same as the gradient
- Can use any subgradient at 0
  - Typically, will use the equation given

# Subgradients and the Max




$$y = \max_j z_j$$

$$\frac{\partial y}{\partial z_i} = \begin{cases} 1, & i = \underset{j}{\operatorname{argmax}}\, z_j \\ 0, & otherwise \end{cases}$$

- Vector equivalent of subgradient
  - 1 w.r.t. the largest incoming input
    - Incremental changes in this input will change the output
  - 0 for the rest
    - Incremental changes to these inputs will not change the output

# Poll 3 : @386

We have y = max(z1, z2, z3), computed at z1 = 1, z2 = 2, z3 = 3. Select all that are true

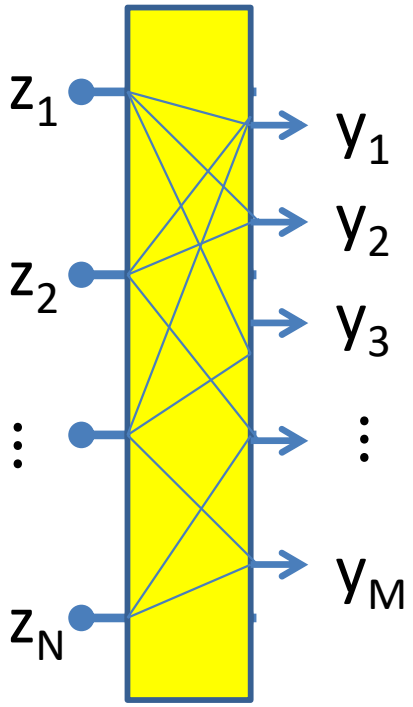
- dy/dz1 = 1
- dy/dz1 = 0
- dy/dz2 = 1
- dy/dz2 = 0
- dy/dz3 = 1
- dy/dz3 = 0

# Poll 3

We have y = max(z1, z2, z3), computed at z1 = 1, z2 = 2, z3 = 3. Select all that are true

- dy/dz1 = 1
- dy/dz1 = 0 **(correct)**
- dy/dz2 = 1
- dy/dz2 = 0 **(correct)**
- dy/dz3 = 1 **(correct)**
- dy/dz3 = 0

# Subgradients and the Max




$$y_i = \max_{l \in S_j} z_l$$

$$\frac{\partial y_j}{\partial z_i} = \begin{cases} 1, & i = \operatorname{argmax}_{l \in S_j} z_l \\ 0, & otherwise \end{cases}$$

- Multiple outputs, each selecting the max of a different subset of inputs
  - Will be seen in convolutional networks
- Gradient for any output:
  - 1 for the specific component that is maximum in corresponding input subset
  - 0 otherwise

# Backward Pass: Recap

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

These may be subgradients

# 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 loss is the sum of the divergence over all training instances

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

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

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

$$W \leftarrow W - \eta \nabla_W \mathbf{Loss}^{\mathrm{T}}$$

# Training by BackProp

- Initialize weights $\boldsymbol{W}^{(k)}$ for all layers $k = 1 \ldots K$
- Do: *(Gradient descent iterations)*
  - **Initialize** $Loss = 0$; For all $i, j, k$, initialize $\frac{dLoss}{dw_{i,j}^{(k)}} = 0$
  - For all $t = 1{:}T$ *(Iterate over training instances)*
    - **Forward pass:** Compute
      - Output $\boldsymbol{Y_t}$
      - $Loss \mathrel{+}= \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$
    - **Backward pass:** For all $i, j, k$:
      - Compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
      - $\frac{dLoss}{dw_{i,j}^{(k)}} \mathrel{+}= \frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
  - For all $i, j, k$, update:

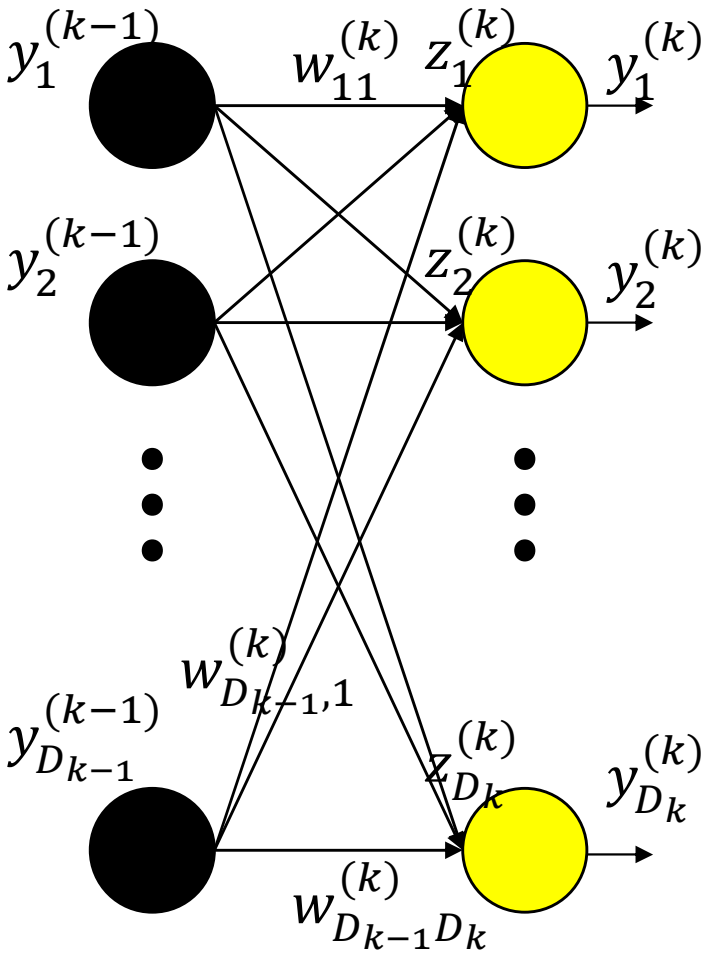
$$w_{i,j}^{(k)} = w_{i,j}^{(k)} - \frac{\eta}{T} \frac{dLoss}{dw_{i,j}^{(k)}}$$

- Until $Loss$ 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
  - This is what is *actually* used in any real system

# Vector formulation

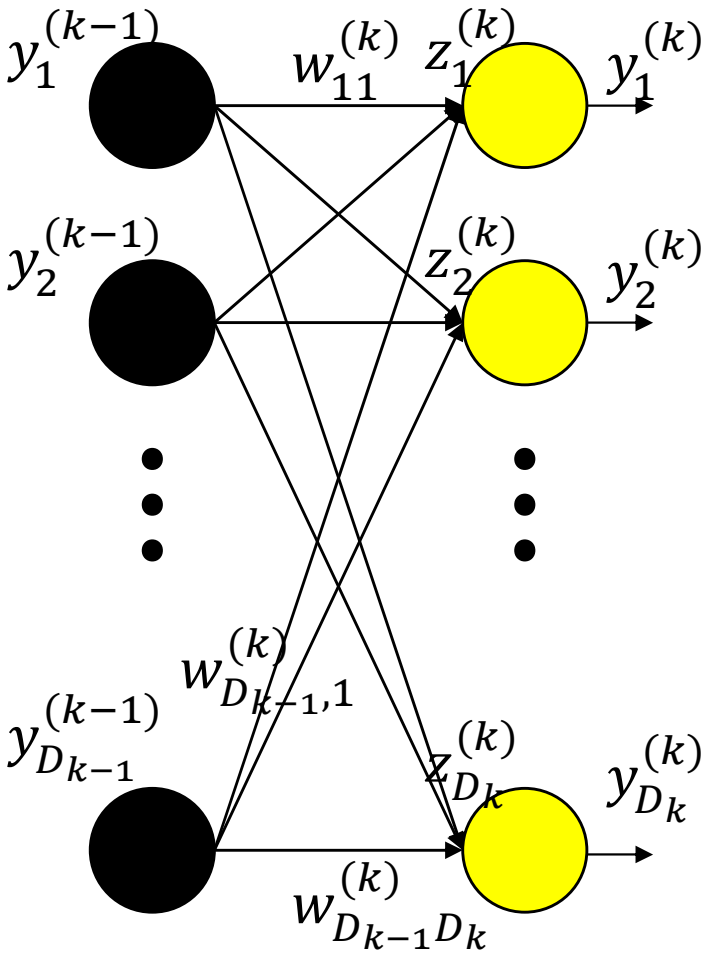



$$\mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

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

- 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



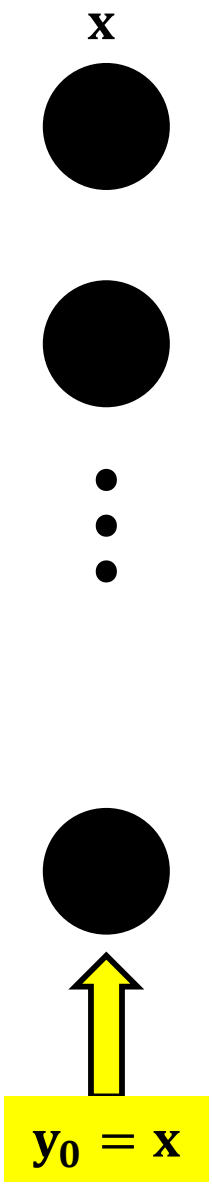

$$\mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

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

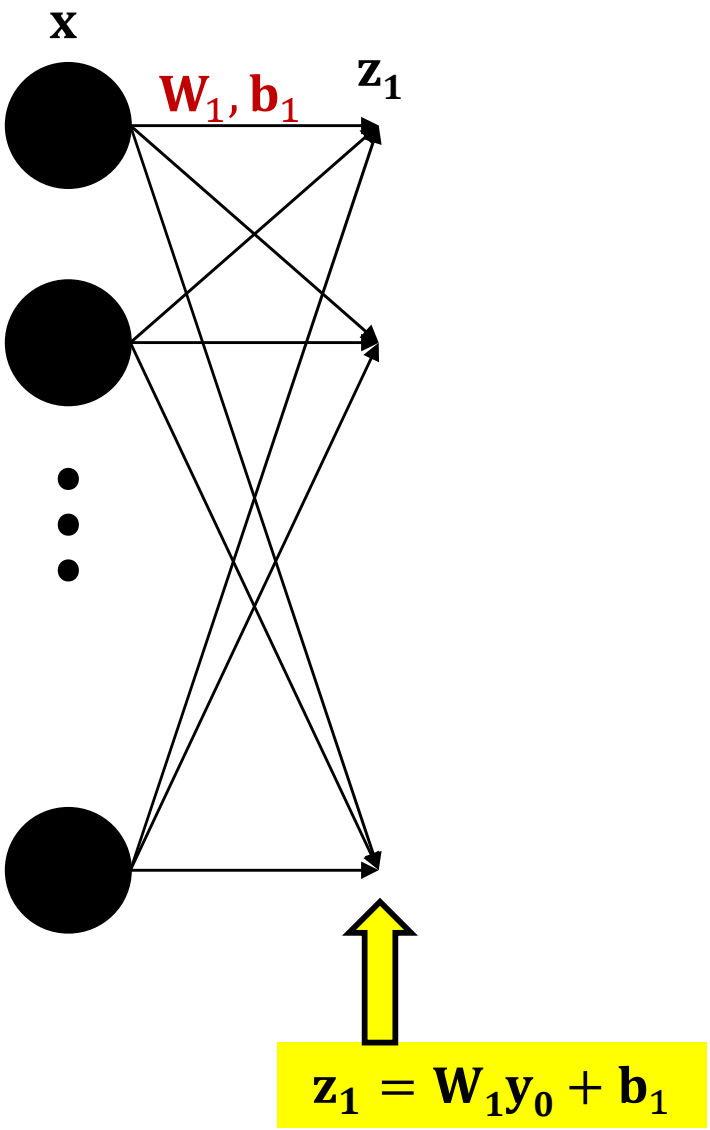
- The computation of a single layer is easily expressed in matrix notation as (setting $\mathbf{y_0} = \mathbf{x}$):

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

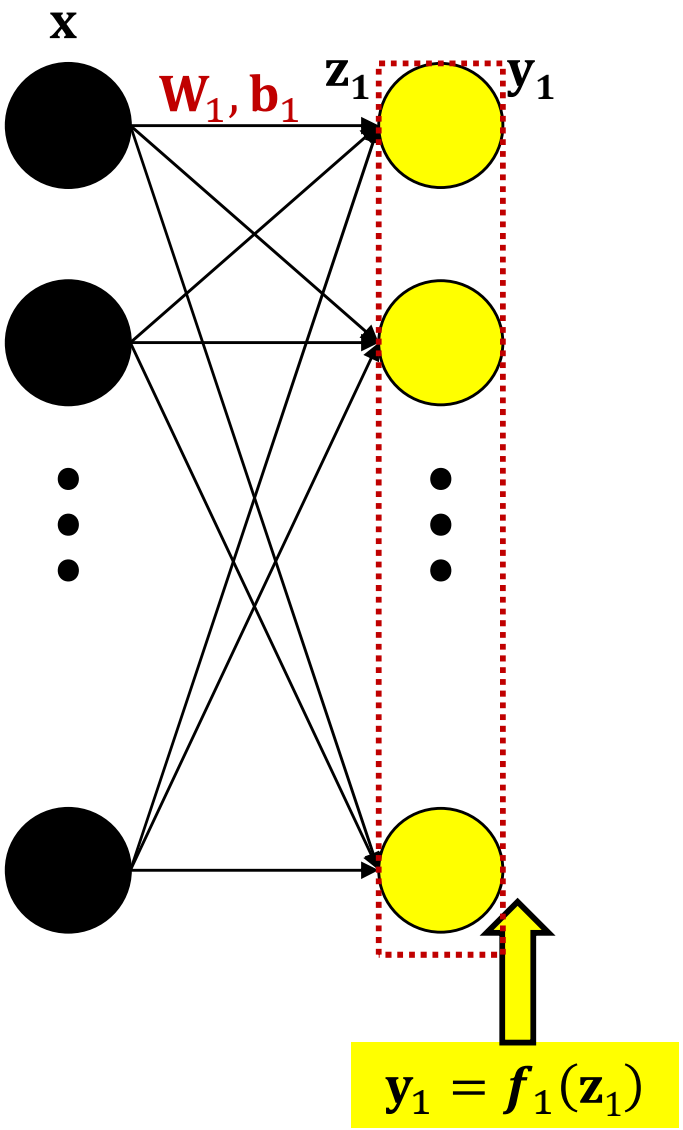
# The forward pass: Evaluating the network



# The forward pass



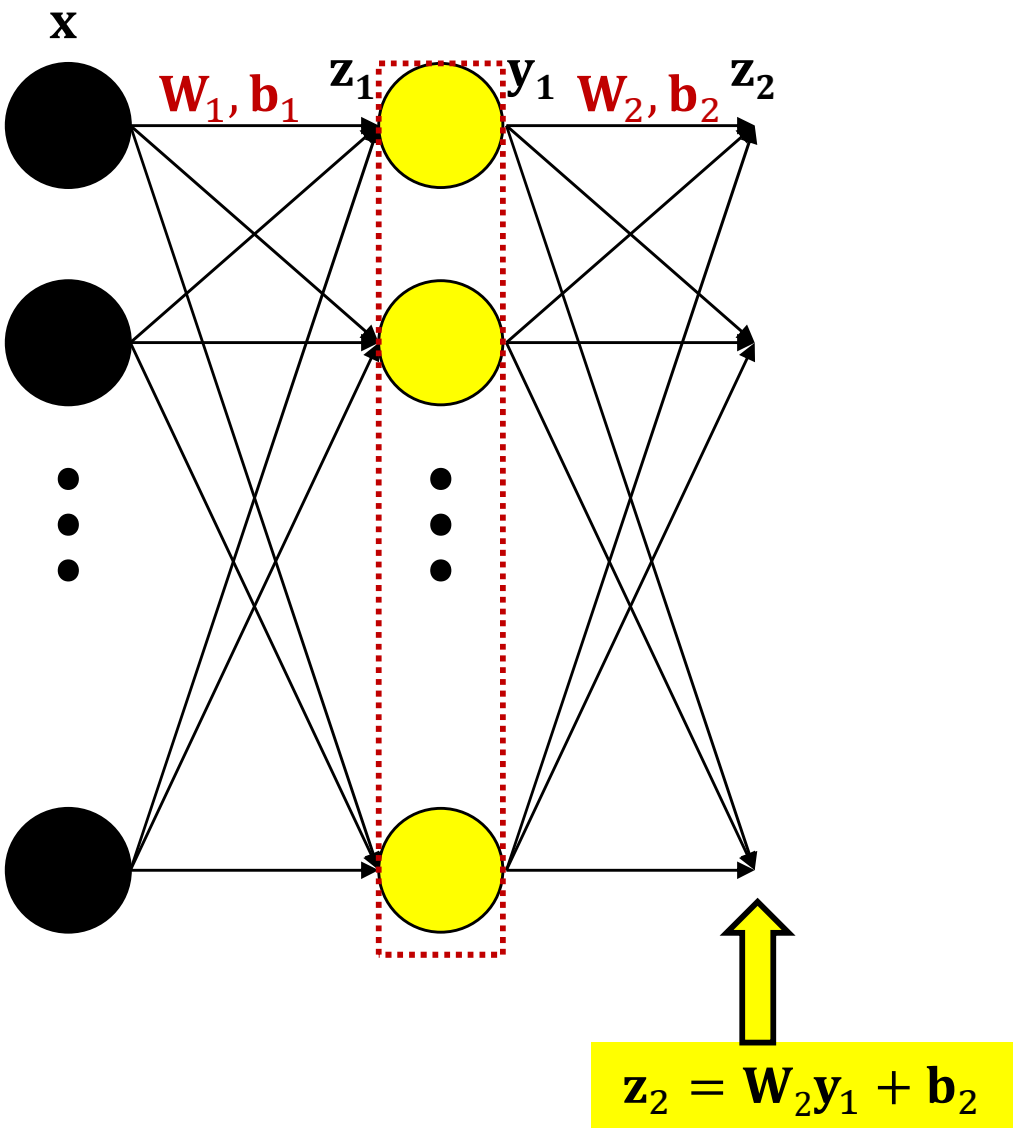
# The forward pass




The Complete computation

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

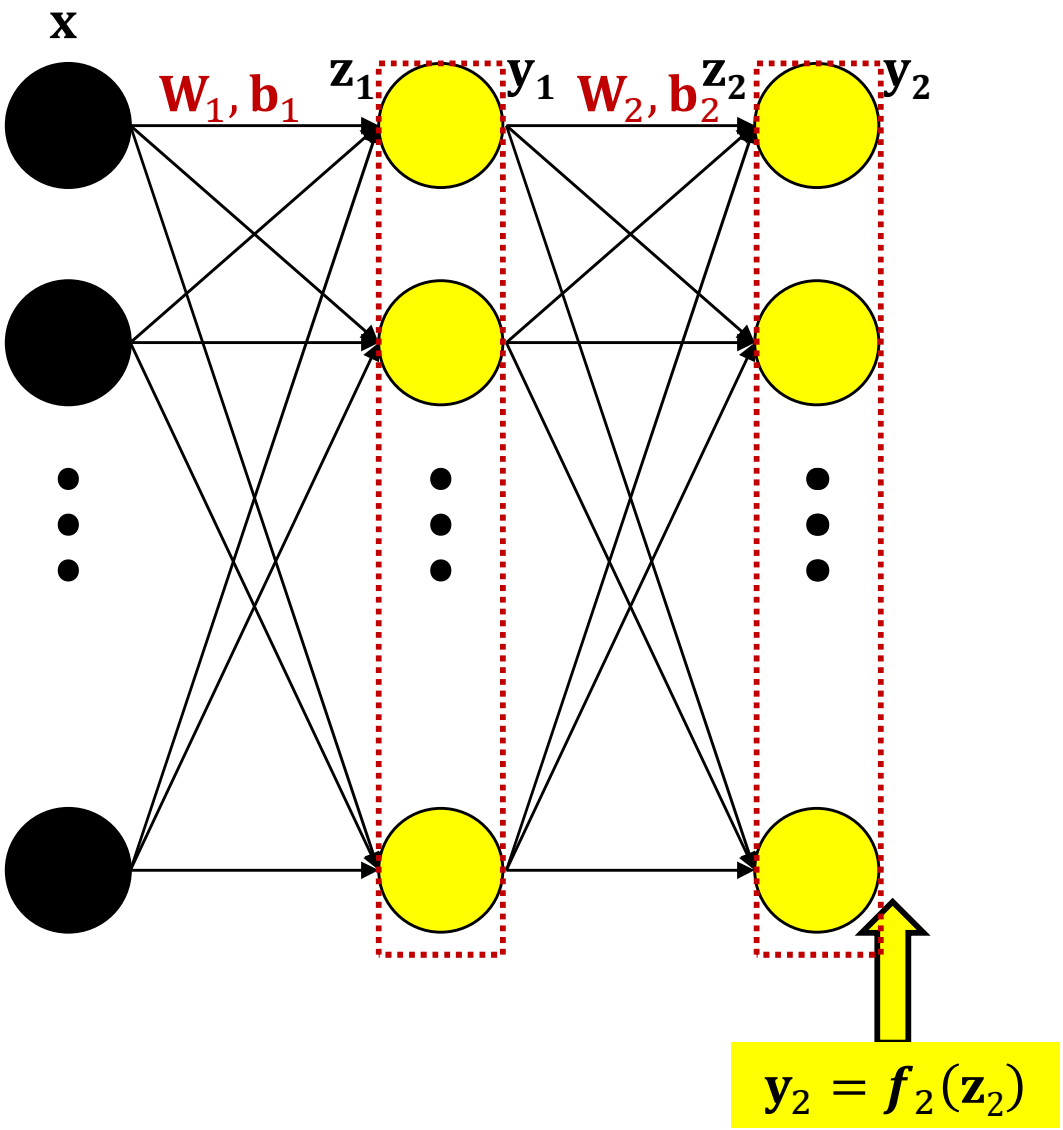
# The forward pass




The Complete computation

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

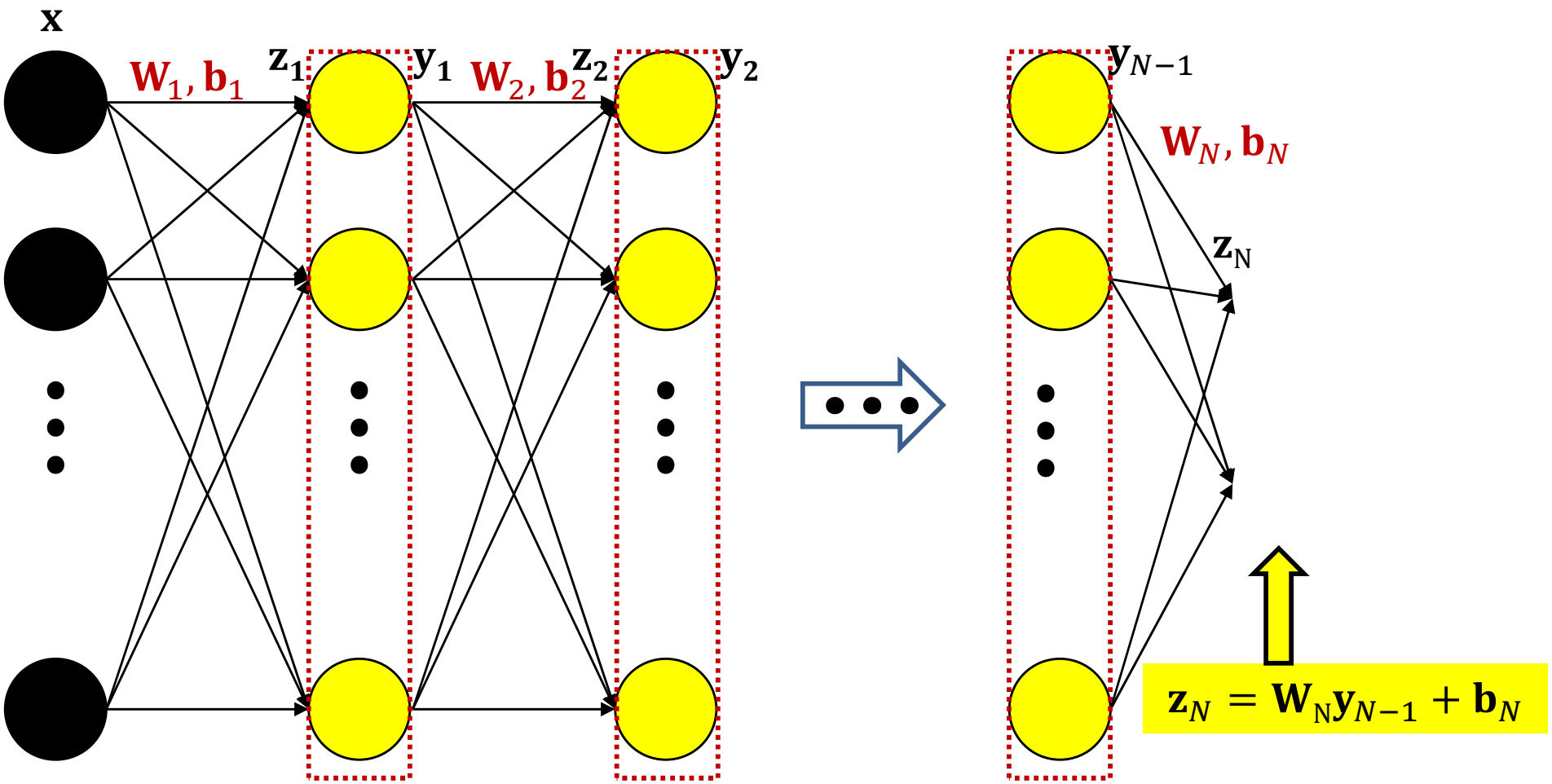
# The forward pass




The Complete computation

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

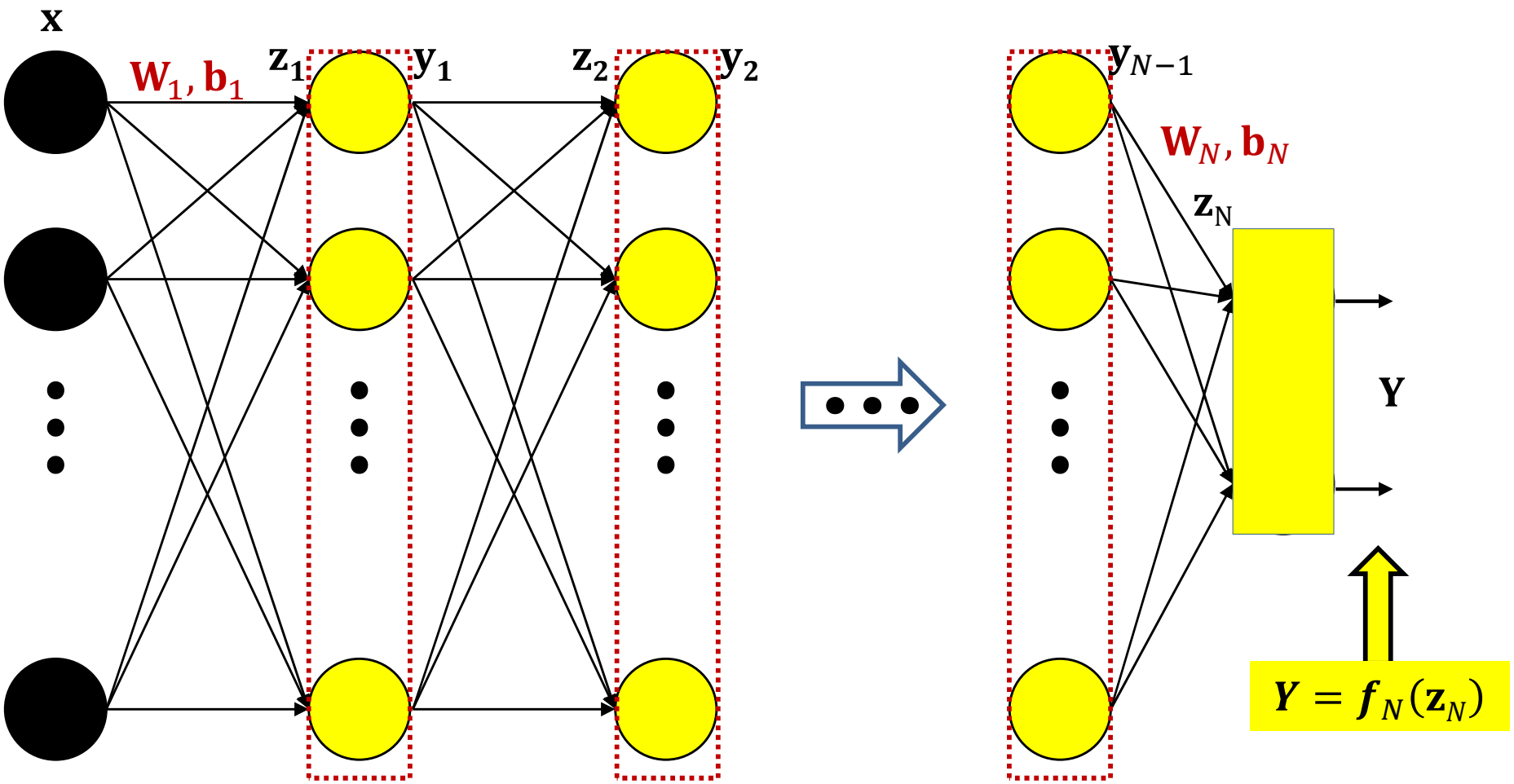
# The forward pass




The Complete computation

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

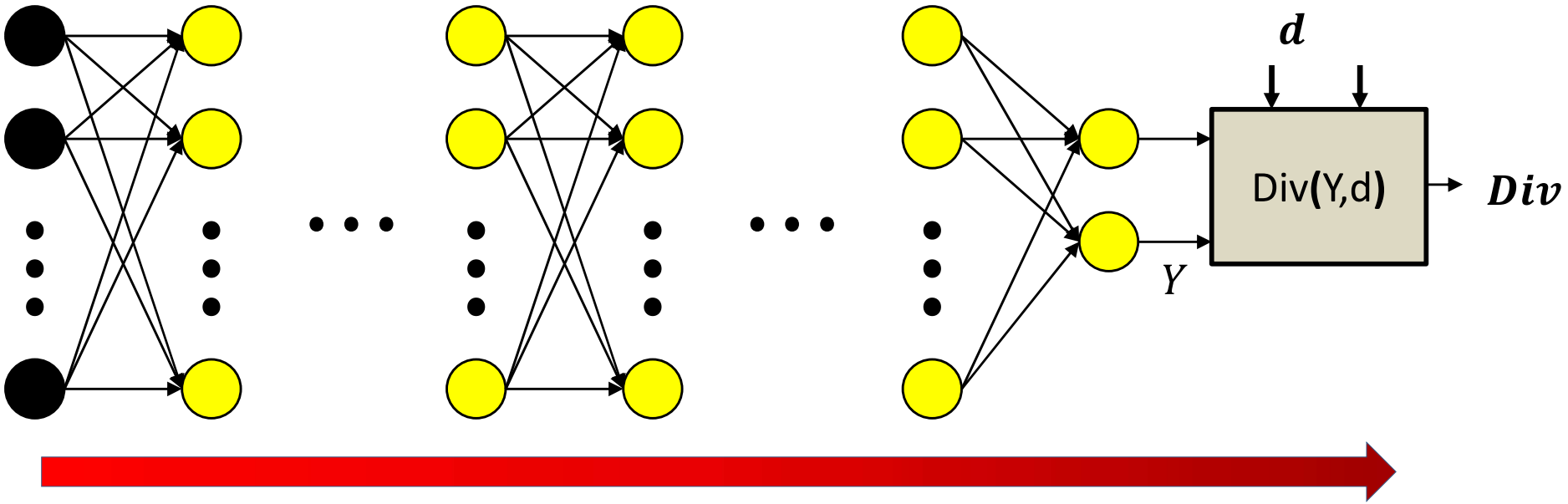
# The forward pass




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 Forward Pass

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

- Iterate through layers:
  - For layer 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:

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

# The Backward Pass

- Have completed the forward pass
- Before presenting the backward pass, some more calculus…
  - *Vector* calculus this time

# Vector Calculus Notes 1: Definitions

- *A derivative is a multiplicative factor that multiplies a perturbation in the input to compute the corresponding perturbation of the output*

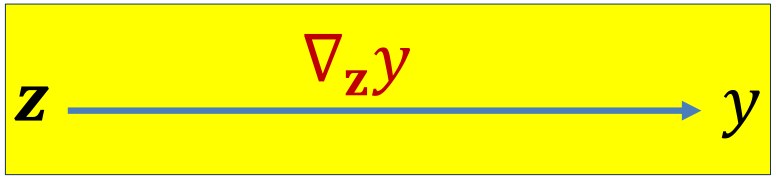
- **For a scalar function of a vector argument**

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

$$\Delta y = \nabla_{\mathbf{z}} y \, \Delta \mathbf{z}$$

- If $\mathbf{z}$ is an $R \times 1$ vector, $\nabla_{\mathbf{z}} y$ is a $1 \times R$ vector
  - The shape of the derivative is the *transpose* of the shape of $\mathbf{z}$

- $\nabla_{\mathbf{z}} y^{\top}$ is called the *gradient* of $y$ w.r.t $\mathbf{z}$




(influence diagram)

# Vector Calculus Notes 1: Definitions

- **For a *vector* function of a vector argument**

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

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

$$\Delta \boldsymbol{y} = \nabla_{\mathbf{z}} \boldsymbol{y}\, \Delta \mathbf{z}$$

- If $\mathbf{z}$ is an $R \times 1$ vector, and $\mathbf{y}$ is an $L \times 1$ $\nabla_{\mathbf{z}}\mathbf{y}$ is an $L \times R$ matrix
  - Or the dimensions won't match
- $\nabla_{\mathbf{z}}\mathbf{y}$ is called the *Jacobian* of $\mathbf{y}$ w.r.t $\mathbf{z}$

# Calculus Notes: 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

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

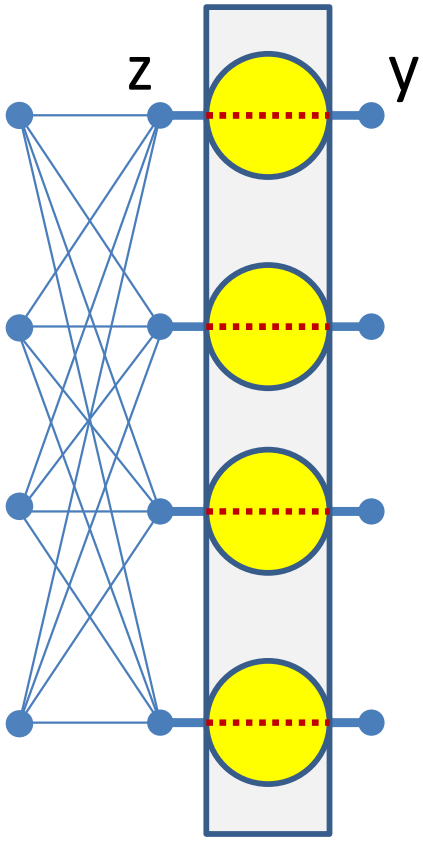
Using vector notation

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

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

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

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

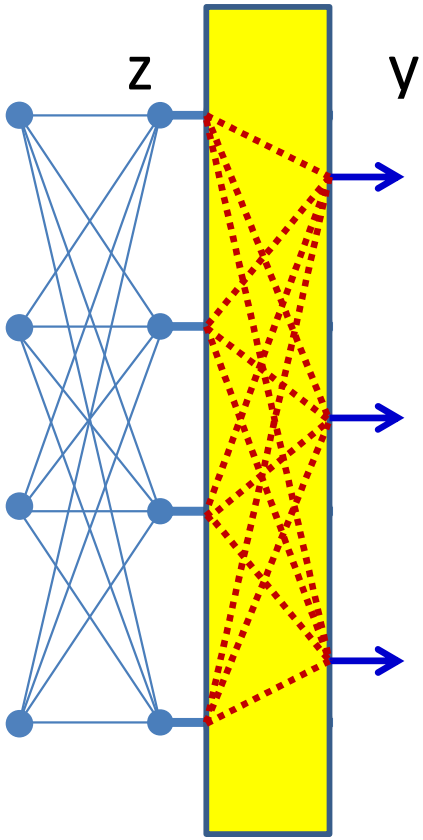



$$y_i = f(z_i)$$

$$J_{\boldsymbol{y}}(\mathbf{z}) = \begin{bmatrix} f'(z_1) & 0 & \cdots & 0 \\ 0 & f'(z_2) & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & f'(z_M) \end{bmatrix}$$

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

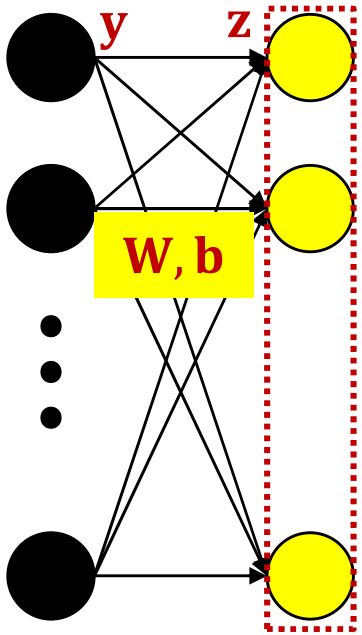
# For *Vector* activations




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

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

# Special case: Affine functions




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

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

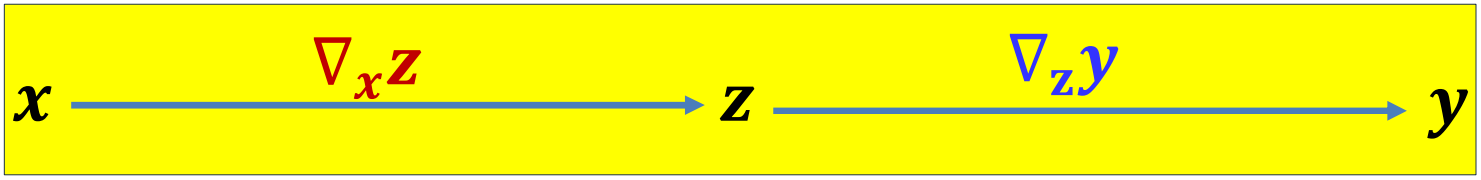
$$\nabla_{\mathbf{b}}\mathbf{z} = J_{\mathbf{z}}(\mathbf{b}) = \boldsymbol{I}$$

- 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 Calculus Notes 2: Chain rule

- For nested functions we have the following chain rule

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




$$\nabla_{\boldsymbol{x}}\boldsymbol{y} = \nabla_{\mathbf{z}}\boldsymbol{y}\nabla_{\boldsymbol{x}}\boldsymbol{z}$$

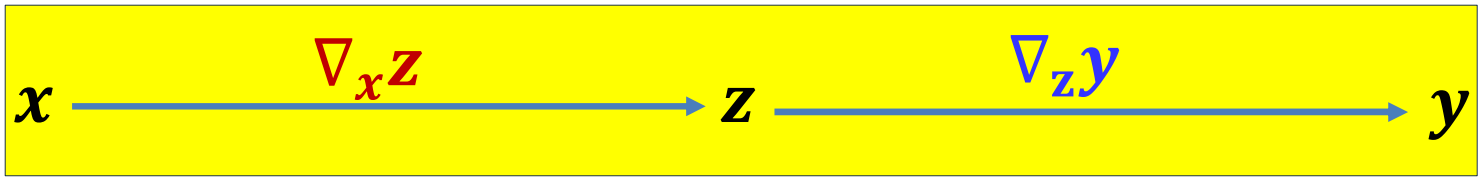
This holds regardless of whether $\boldsymbol{y}$ is scalar or vector

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

# Vector Calculus Notes 2: Chain rule

- For nested functions we have the following chain rule

$$\mathbf{y} = y(z(\mathbf{x})) \quad \Rightarrow \quad \nabla_{\mathbf{x}}\mathbf{y} = \nabla_{\mathbf{z}}y\nabla_{\mathbf{x}}\mathbf{z}$$




Check

$$\Delta\mathbf{y} = \nabla_{\mathbf{z}}y\Delta z$$

$$\Delta z = \nabla_{x}\mathbf{z}\Delta x$$

$$\Delta\mathbf{y} = \nabla_{\mathbf{z}}y\nabla_{x}\mathbf{z}\,\Delta x$$

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

# Vector Calculus Notes 2: Chain rule

- Chain rule for Jacobians:
- **For vector functions of vector inputs:**

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

Check

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

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

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

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

# Vector Calculus Notes 2: Chain rule

- *Combining Jacobians and Gradients*
- **For *scalar* functions of vector inputs ($\mathbf{z}()$ is vector):**

$$D = D\big(\mathbf{y}(\mathbf{z})\big) \quad \Rightarrow \quad \nabla_{\mathbf{z}} D = \nabla_{\mathbf{y}}(D) J_{\mathbf{y}}(\mathbf{z})$$

Check

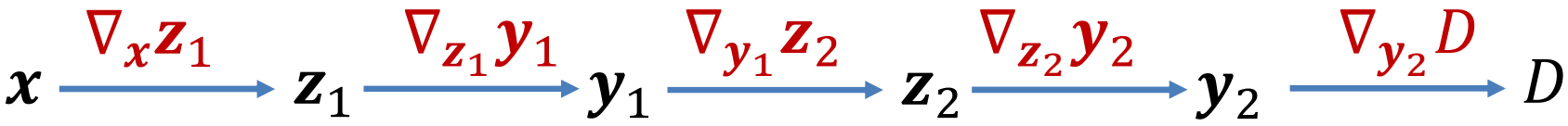
$$\Delta D = \nabla_{\mathbf{y}}(D)\Delta \mathbf{y}$$

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

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

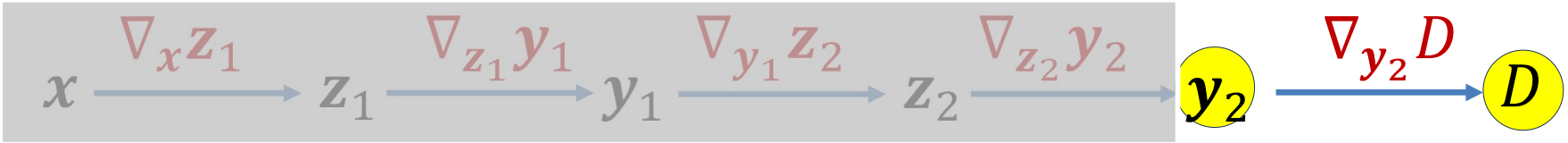
Note the order: The derivative of the outer function comes first

# Extended Chain rule

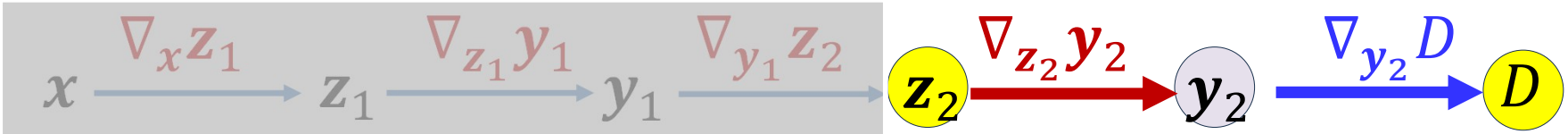



How do we compute the derivative of $D$ w.r.t. $x$, $z_1$, $y_1$, $z_2$ and $y_2$, from the local derivatives shown on the edges?

# Extended Chain rule



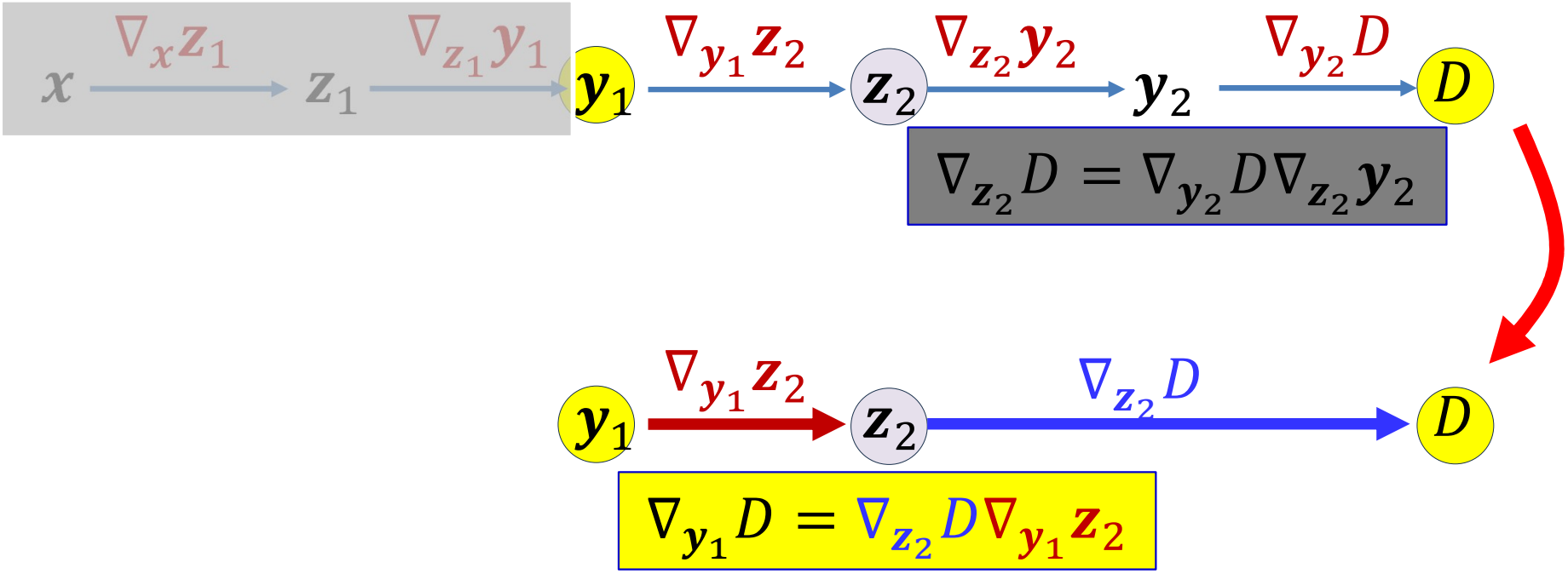
# Extended Chain rule




$$\nabla_{z_2} D = \nabla_{y_2} D \nabla_{z_2} y_2$$

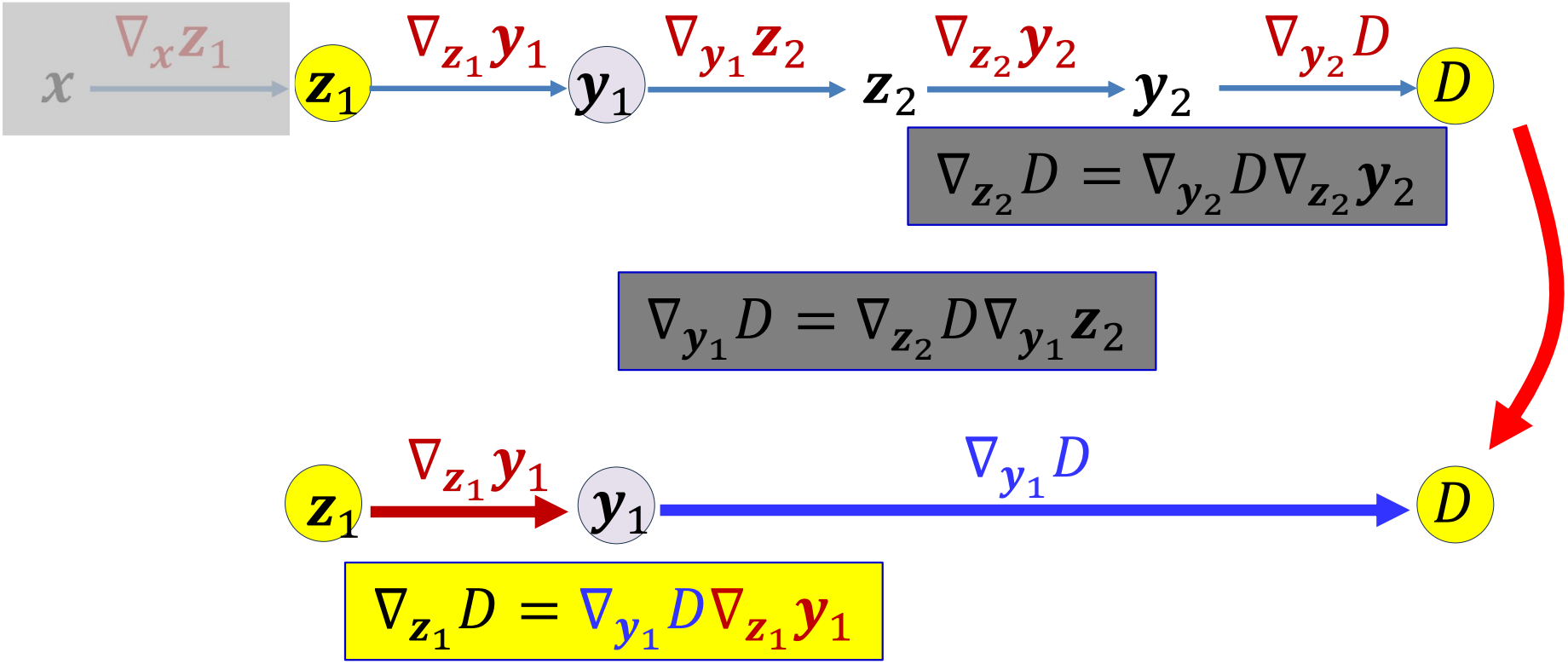
Note the order: The derivative of the outer function comes first

# Extended Chain rule



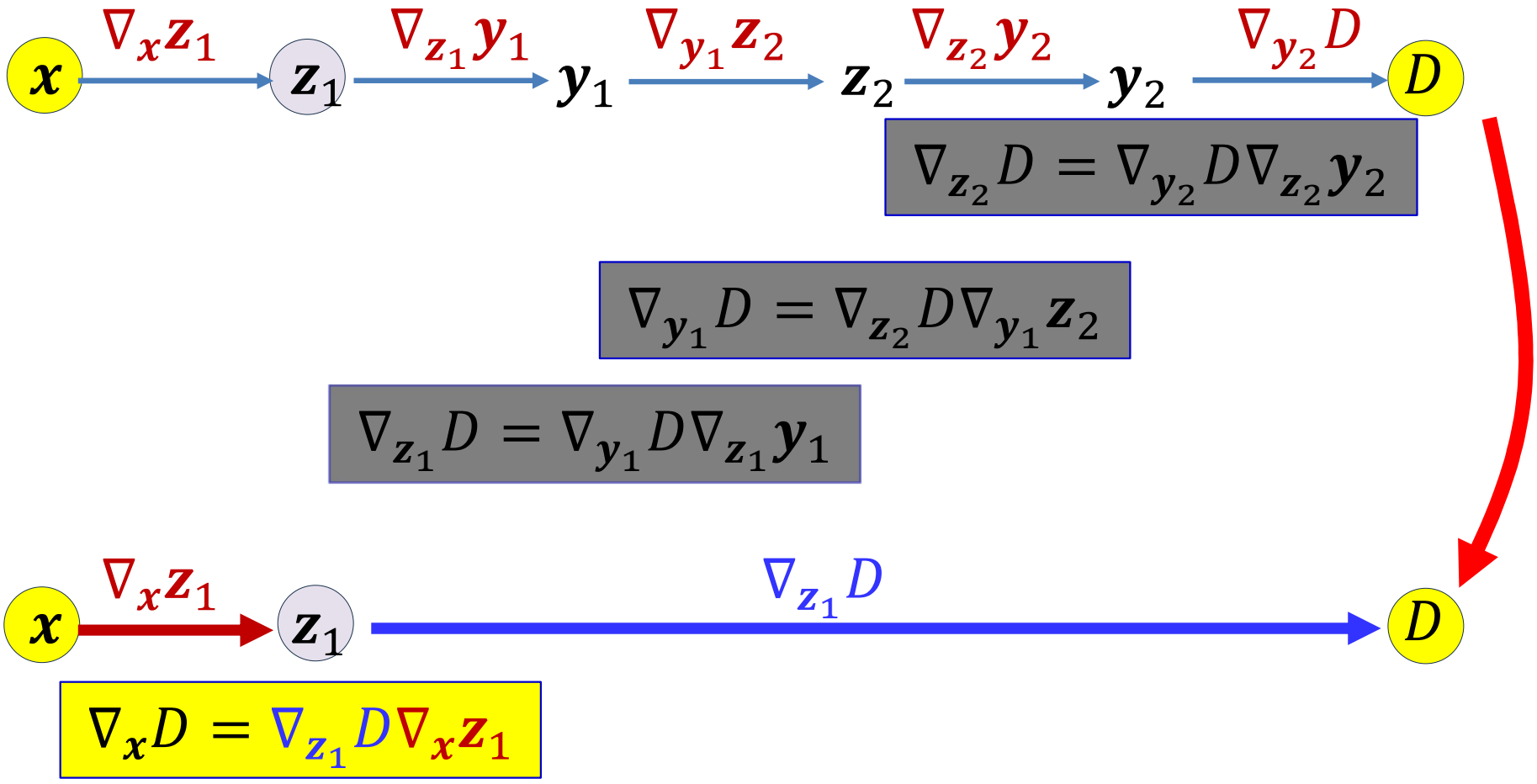

Note the order: The derivative of the outer function comes first

# Extended Chain rule



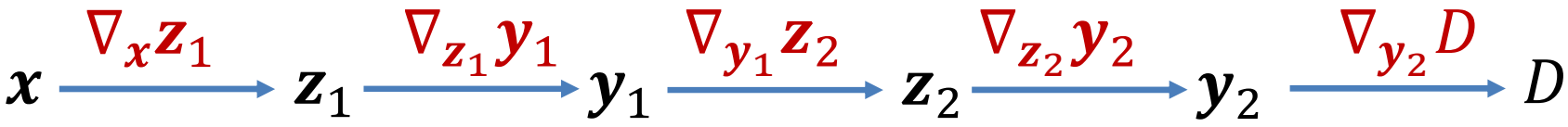

Note the order: The derivative of the outer function comes first

# Extended Chain rule




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

# Extended Chain rule




$$\nabla_{z_2} D = \nabla_{y_2} D \nabla_{z_2} y_2$$

$$\nabla_{y_1} D = \nabla_{z_2} D \nabla_{y_1} z_2$$

$$\nabla_{z_1} D = \nabla_{y_1} D \nabla_{z_1} y_1$$

$$\nabla_x D = \nabla_{z_1} D \nabla_x z_1$$

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

# Vector Calculus Notes 2: Chain rule

- For nested functions we have the following chain rule

$$D = D\left(\boldsymbol{y}_N\left(\boldsymbol{z}_N\left(\boldsymbol{y}_{N-1}\left(\boldsymbol{z}_{N-1}\left(\cdots \boldsymbol{y}_1\left(\boldsymbol{z}_1(\mathbf{x})\right)\right)\right)\right)\right)\right)$$

$$\nabla_{\mathbf{x}} D = \nabla_{\mathbf{y}_N} D \nabla_{\mathbf{z}_N} \boldsymbol{y}_N \nabla_{\boldsymbol{y}_{N-1}} \mathbf{z}_N \nabla_{\mathbf{z}_{N-1}} \boldsymbol{y}_{N-1} \cdots \nabla_{\mathbf{z}_1} \boldsymbol{y}_1 \nabla_{\mathbf{x}} \mathbf{z}_1$$

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

# Vector Calculus Notes 2: Chain rule

- For nested functions we have the following chain rule

$$D = D\left(\boldsymbol{y}_N\left(\boldsymbol{z}_N\left(\boldsymbol{y}_{N-1}\left(\boldsymbol{z}_{N-1}\left(\cdots \boldsymbol{y}_1\left(\boldsymbol{z}_1(\mathbf{x})\right)\right)\right)\right)\right)\right)$$

$$\nabla_{\mathbf{x}} D = \nabla_{\mathbf{y}_N} D \nabla_{\mathbf{z}_N} \boldsymbol{y}_N \nabla_{\boldsymbol{y}_{N-1}} \mathbf{z}_N \nabla_{\mathbf{z}_{N-1}} \boldsymbol{y}_{N-1} \cdots \nabla_{\mathbf{z}_1} \boldsymbol{y}_1 \nabla_{\mathbf{x}} \mathbf{z}_1$$

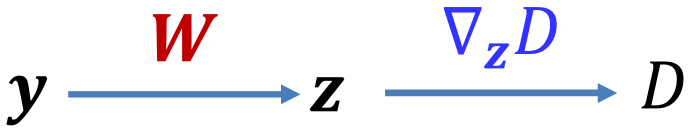
Note the order: The derivative of the outer function comes first

# More calculus: Special Case

- Scalar functions of Affine functions

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

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



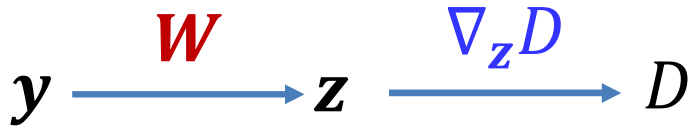

$$\nabla_{\mathbf{y}} D = \nabla_{\mathbf{z}} D \mathbf{W}$$

# More calculus: Special Case

- Scalar functions of Affine functions

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

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




$$\nabla_{\mathbf{y}} D = \nabla_{\mathbf{z}} D \mathbf{W}$$

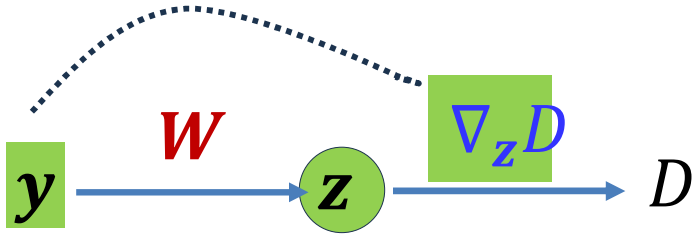
$$\nabla_{\mathbf{b}} D = \nabla_{\mathbf{z}} D \nabla_{\mathbf{b}} \mathbf{z} = \nabla_{\mathbf{z}} D$$

# More calculus: Special Case

- Scalar functions of Affine functions

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

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




$$\nabla_{\mathbf{y}} D = \nabla_{\mathbf{z}} D \mathbf{W}$$

$$\nabla_{\mathbf{b}} D = \nabla_{\mathbf{z}} D$$

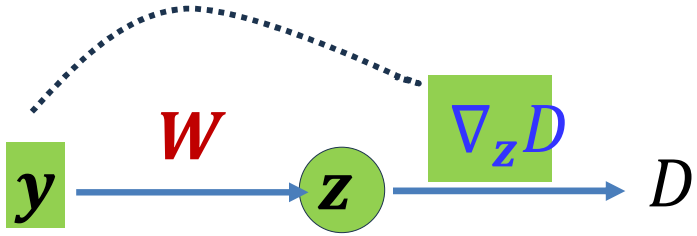
$$\nabla_{\boldsymbol{W}} D = \mathbf{y} \nabla_{\mathbf{z}} D$$

# More calculus: Special Case

- Scalar functions of Affine functions

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

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




$$\nabla_{\mathbf{y}} D = \nabla_{\mathbf{z}} D \mathbf{W}$$

- The derivative of a scalar divergence w.r.t. an Nx1 vector is 1xN
- The derivative of a scalar divergence w.r.t. an NxM matrix is MxN

$$\nabla_{\mathbf{b}} D = \nabla_{\mathbf{z}} D$$

$$\nabla_{\boldsymbol{W}} D = \mathbf{y} \nabla_{\mathbf{z}} D$$

- If z is Nx1 and y is Mx1…
  - What is the size of W
  - Verify that $\nabla_{\boldsymbol{W}} D$ is the correct size

# More calculus: Special Case

- Scalar functions of Affine functions

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

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

$$\nabla_{\mathbf{y}} D = \nabla_{\mathbf{z}}(D)\mathbf{W}$$

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

$$\nabla_{\mathbf{W}} D = \mathbf{y}\nabla_{\mathbf{z}}(D)$$

Derivatives w.r.t parameters

- Note: the derivative shapes are the *transpose* of the shapes of $\mathbf{W}$ and $\mathbf{b}$

# More calculus: Special Case

- Scalar functions of Affine functions

$$\mathbf{z} = \mathbf{W}\mathbf{y} + \mathbf{b} \qquad D = f(\mathbf{z})$$

- Writing the transpose

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

$$\nabla_{\boldsymbol{W}^\top} \boldsymbol{z}^\top = \mathbf{y}^\top$$

$$\nabla_{\boldsymbol{W}^\top} D = \nabla_{\boldsymbol{z}^\top} D \, \nabla_{\boldsymbol{W}^\top} \boldsymbol{z}^\top = \nabla_{\boldsymbol{z}^\top} D \;\; \mathbf{y}^\top$$

$$\nabla_{\boldsymbol{W}} D = (\nabla_{\boldsymbol{W}^\top} D)^\top = \mathbf{y} \nabla_{\boldsymbol{z}} D$$

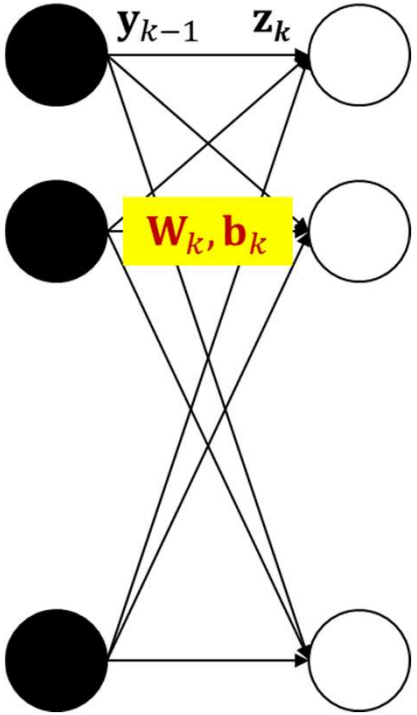
$$\nabla_{\mathbf{W}} D = \mathbf{y} \nabla_{\mathbf{z}} (D)$$

# Special Case: Application to a network

- Scalar functions of Affine functions

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

$$Div = Div(\mathbf{z}) \quad \Rightarrow \quad \nabla_{\mathbf{y}} Div = \nabla_{\mathbf{z}} Div \mathbf{W}$$




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

The divergence is a scalar function of $\mathbf{z}_k$

Applying the above rule

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

# Special Case: Application to a network

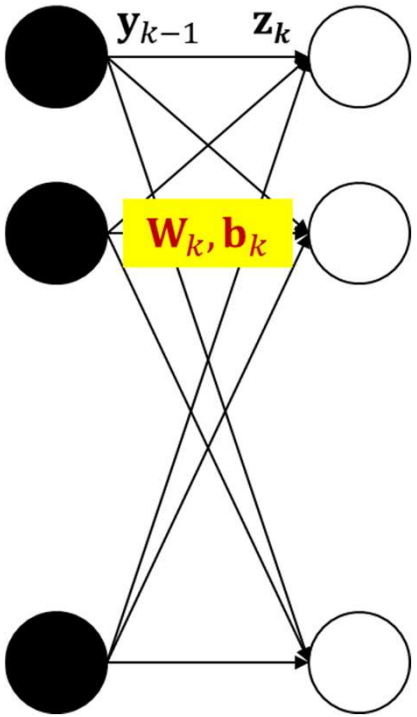
- Scalar functions of Affine functions

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

$$Div = Div(\mathbf{z})$$

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

$$\nabla_{\mathbf{W}} Div = \mathbf{y} \nabla_{\mathbf{z}} Div$$




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

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

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

# Poll 4 : @387

We are given the function $Y = F(G(H(X)))$, where $Y$ and $X$ are vectors, and $G$ and $H$ also compute vector outputs.

Select the correct formula for the derivative of $Y$ w.r.t. $X$. We use the notation $\nabla_X(Y)$ to represent the derivative of $Y$ w.r.t $X$.

- $\nabla_X(H)\,\nabla_H(G)\,\nabla_G(F)$
- $\nabla_G(F)\nabla_H(G)\,\nabla_X(H)$
- Both are correct

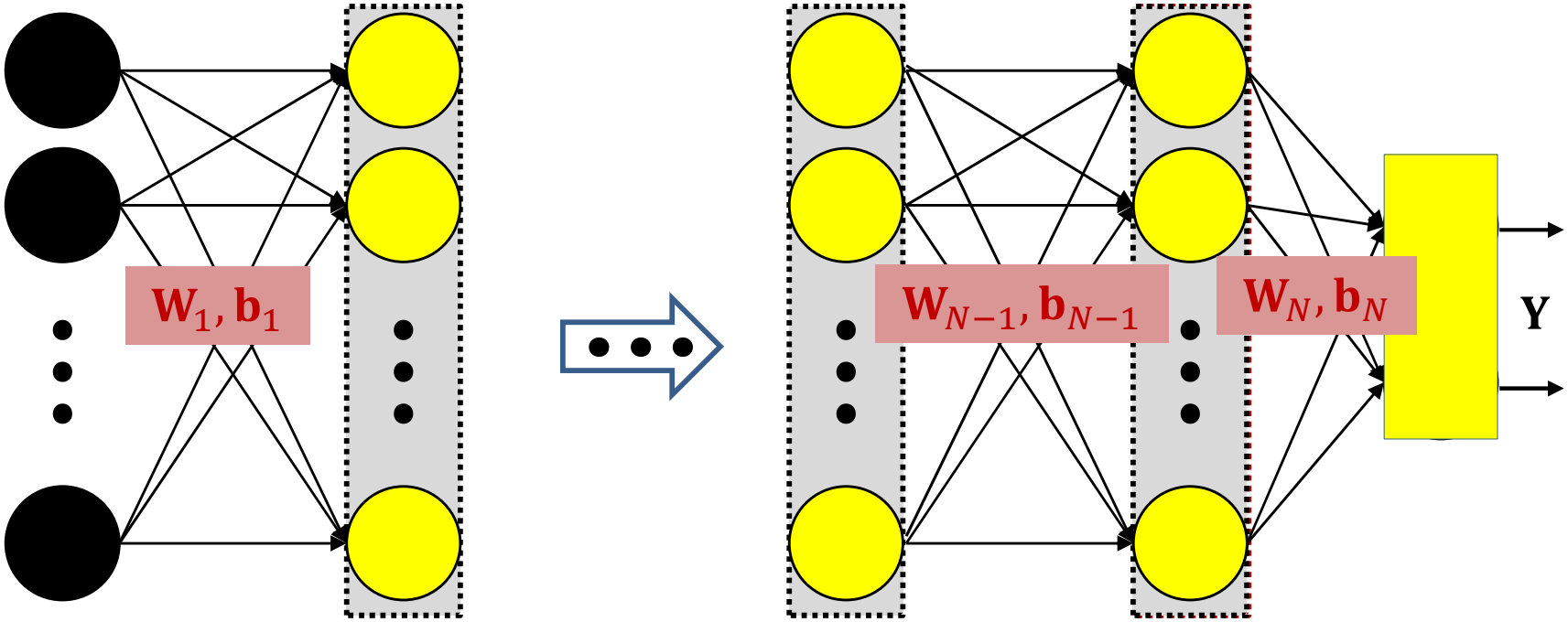
# Poll 4

We are given the function $Y = F(G(H(X)))$, where $Y$ and $X$ are vectors, and $G$ and $H$ also compute vector outputs.

Select the correct formula for the derivative of $Y$ w.r.t. $X$. We use the notation $\nabla_X(Y)$ to represent the derivative of $Y$ w.r.t $X$.

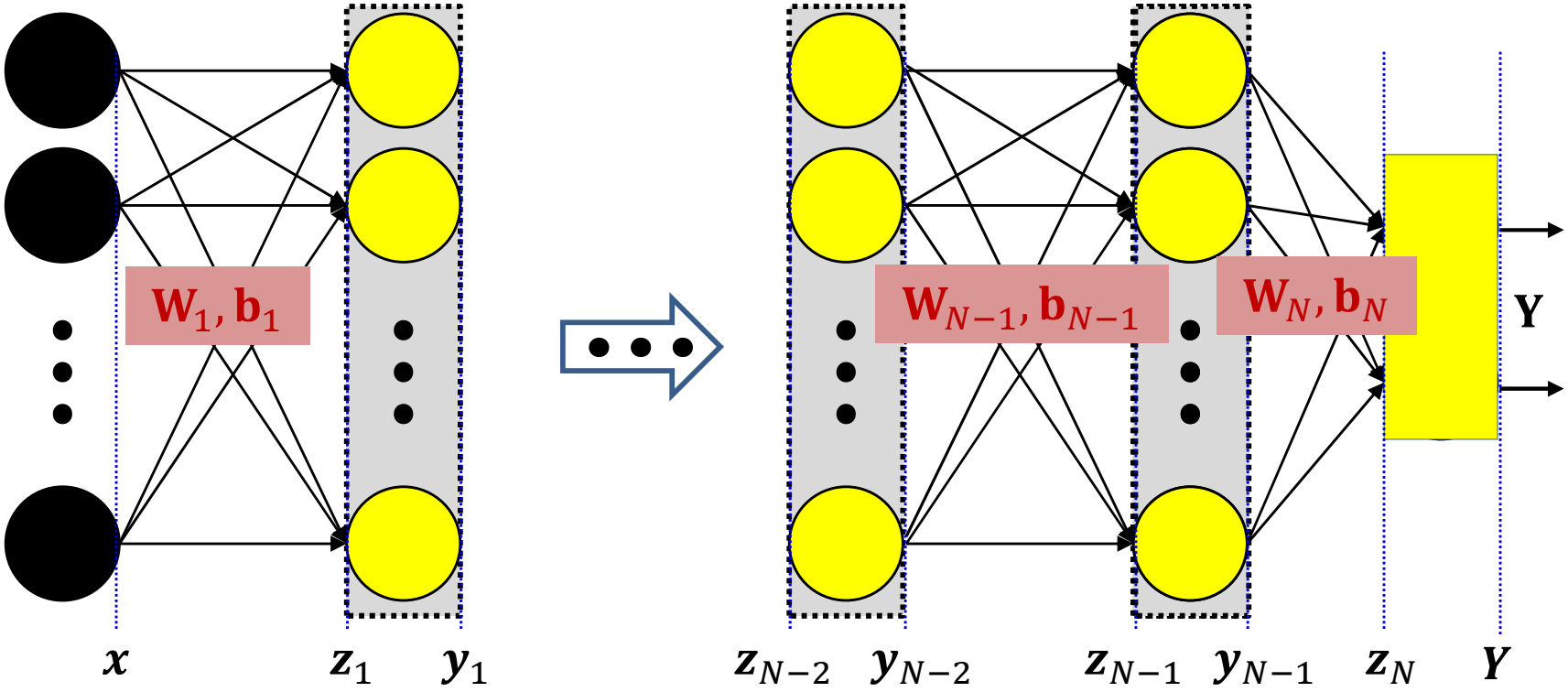
- $\nabla_X(H)\,\nabla_H(G)\,\nabla_G(F)$
- $\nabla_G(F)\nabla_H(G)\,\nabla_X(H)$ **(correct)**
- Both are correct

# The backward pass



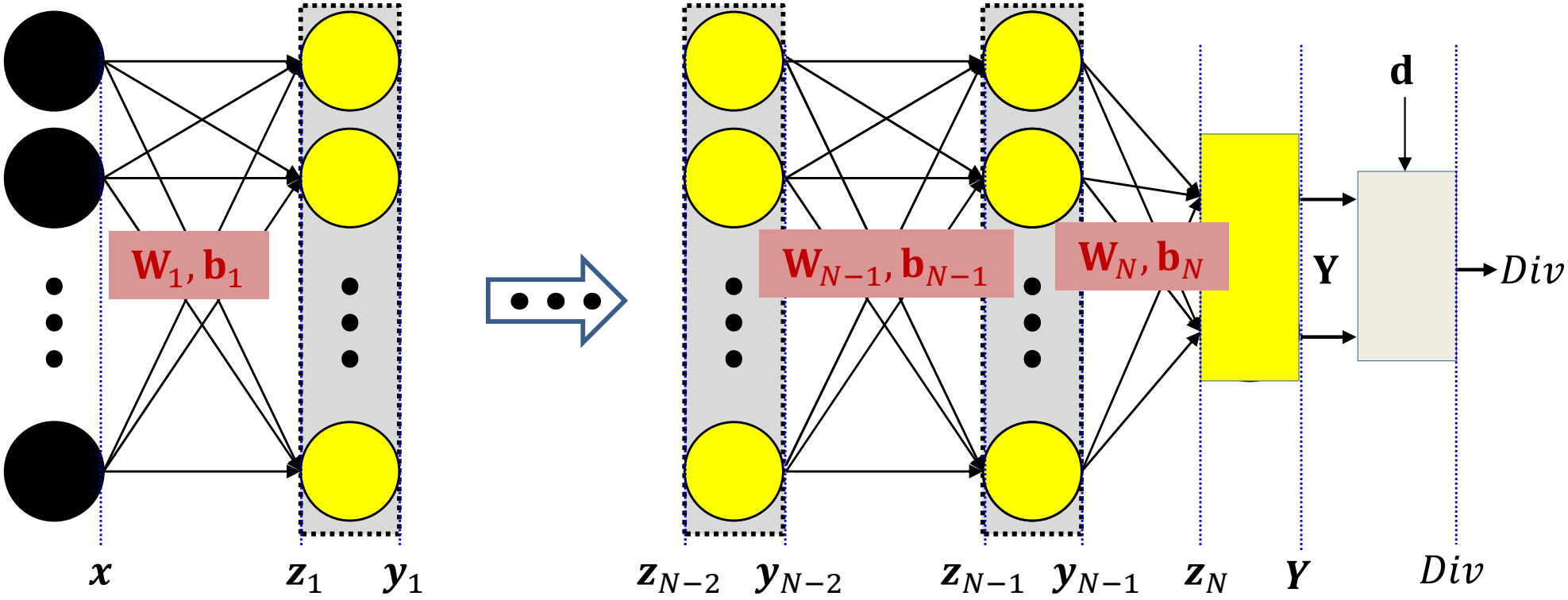

- The network again

# The backward pass



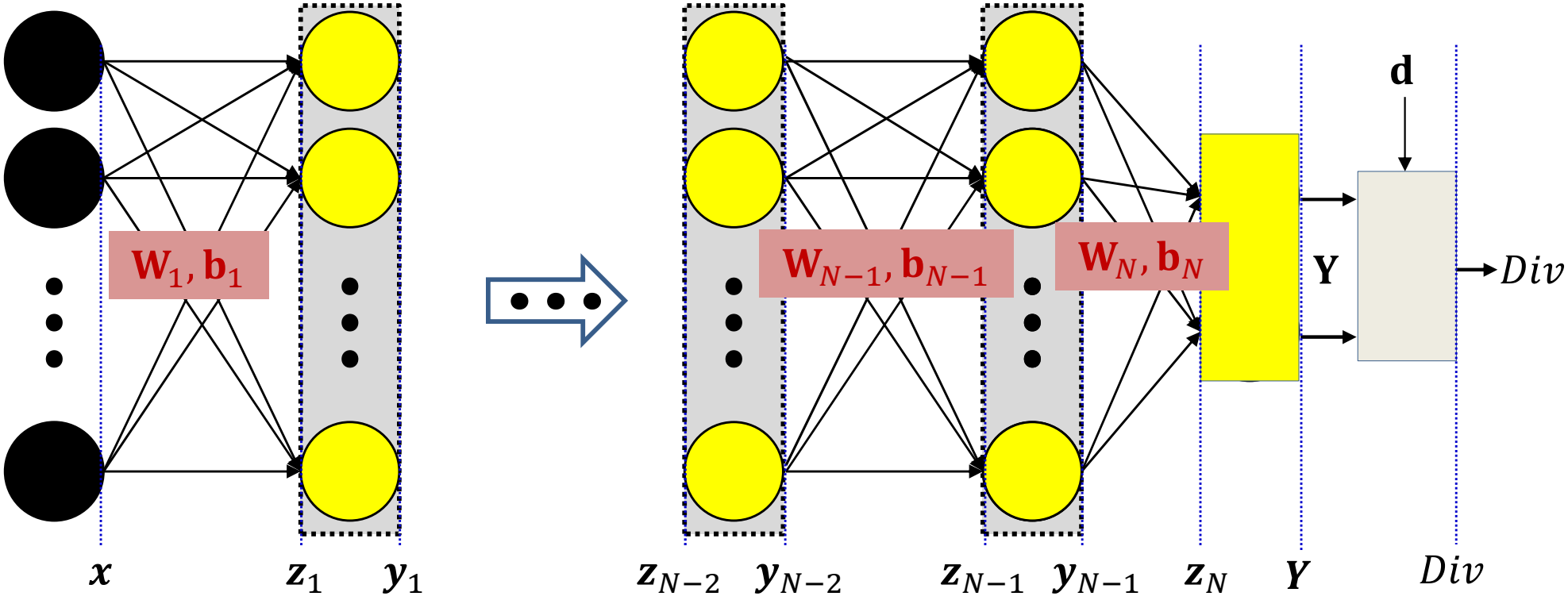

- The network again (with variables shown)…

# The backward pass




- The network again (with variables shown)…
- With the divergence we will minimize…

# The backward pass



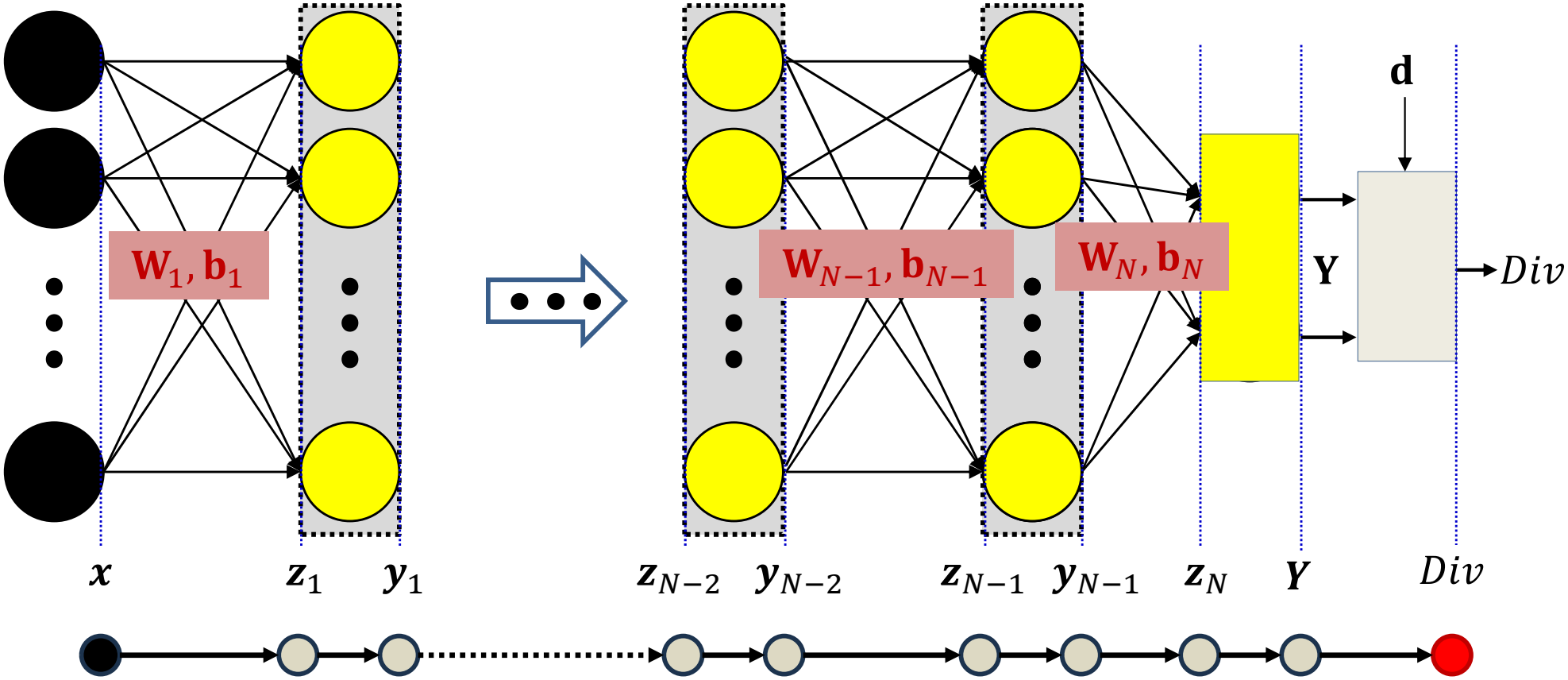

- The network is a nested function

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

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

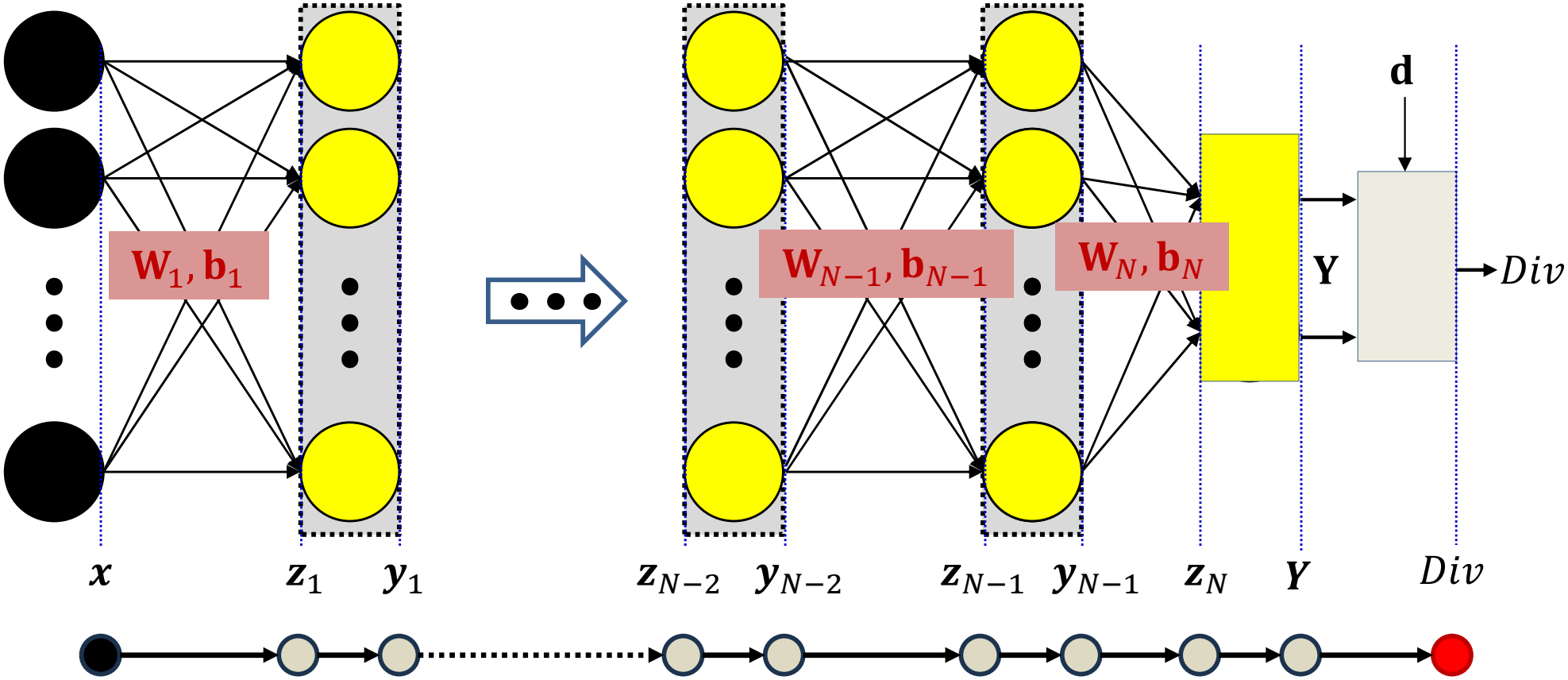
$$Div(Y, d) = Div(f_N(\mathbf{W}_N f_{N-1}(\dots f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\dots) + \mathbf{b}_N), d)$$

# The backward pass




- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram

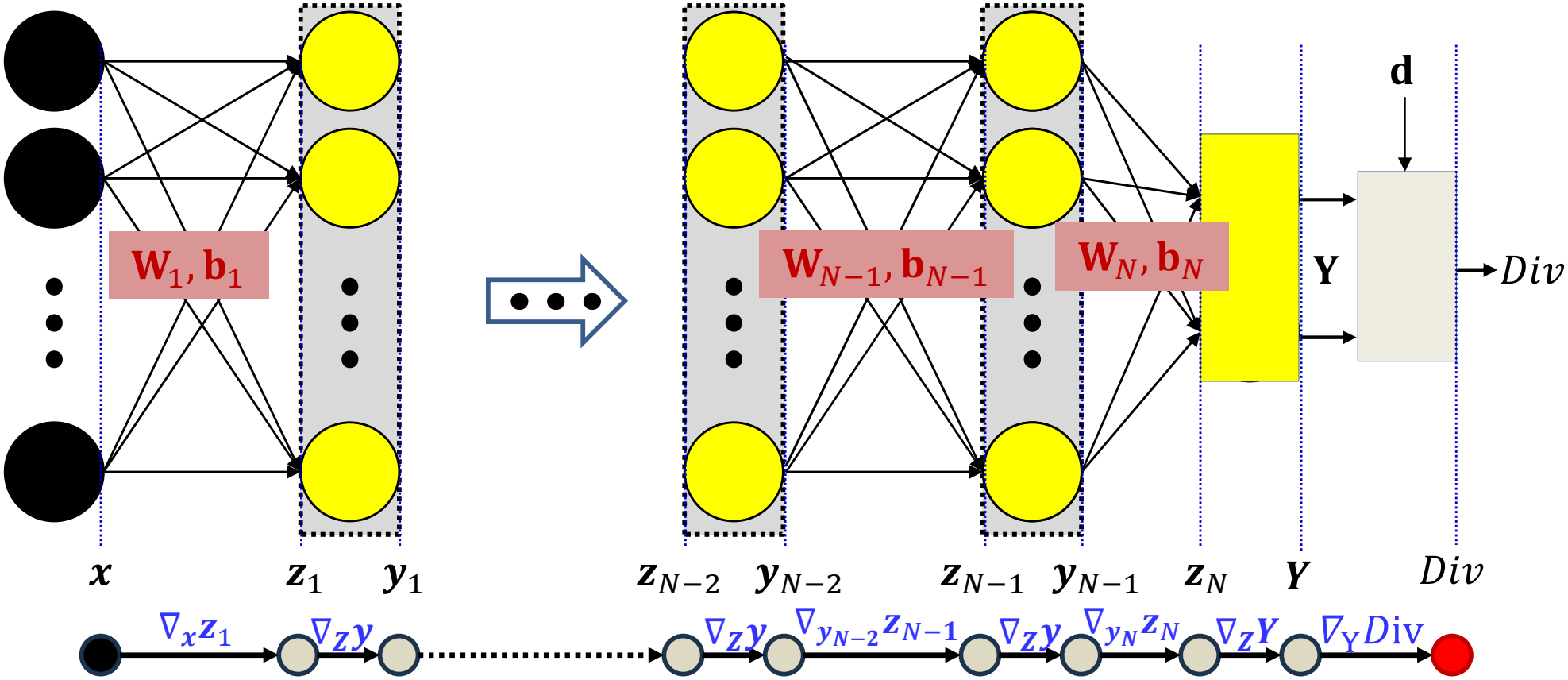
# The backward pass




In the following slides we will also be using the notation $\nabla_{\mathbf{z}}\mathbf{y}$ to represent the derivative of any $\mathbf{y}$ w.r.t any $\mathbf{z}$

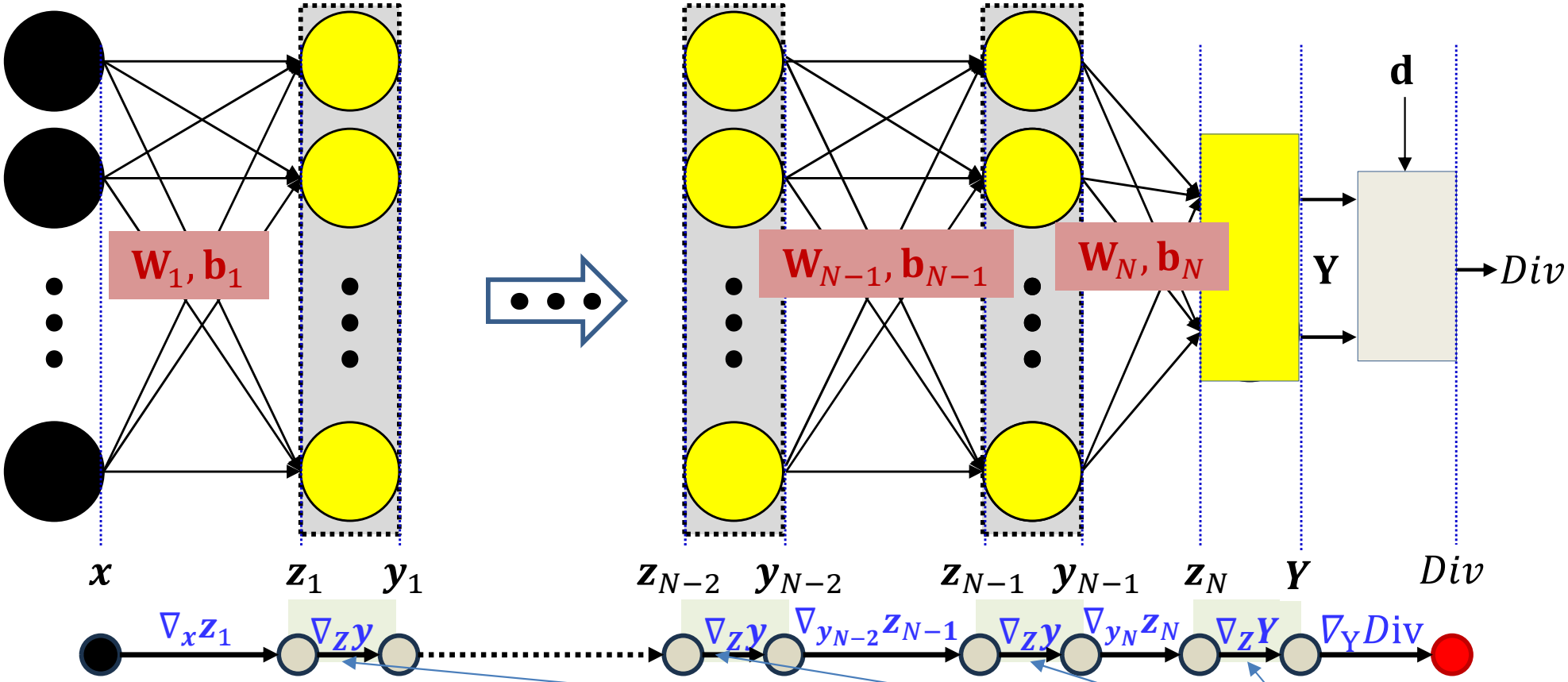
Note that for activation functions, these are actually Jacobians

# The backward pass



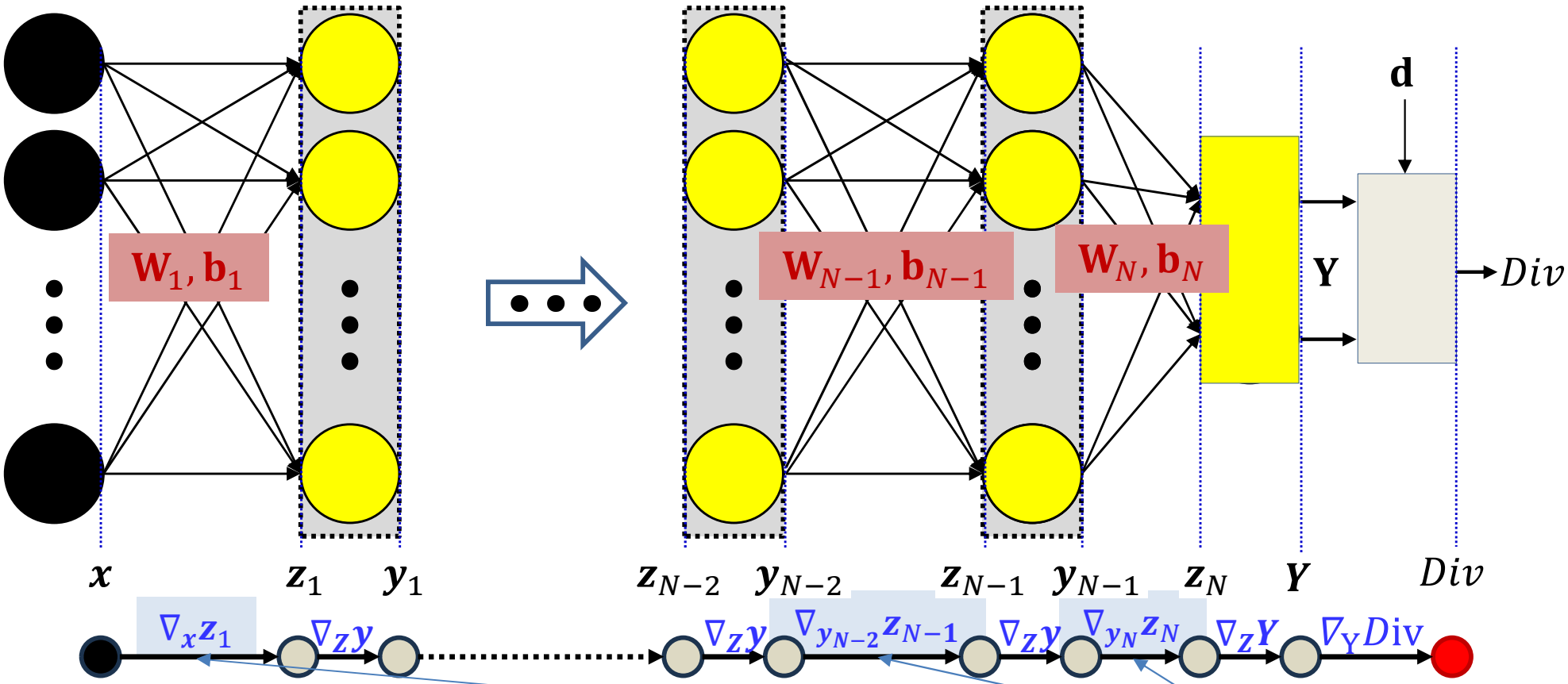

- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

# The backward pass



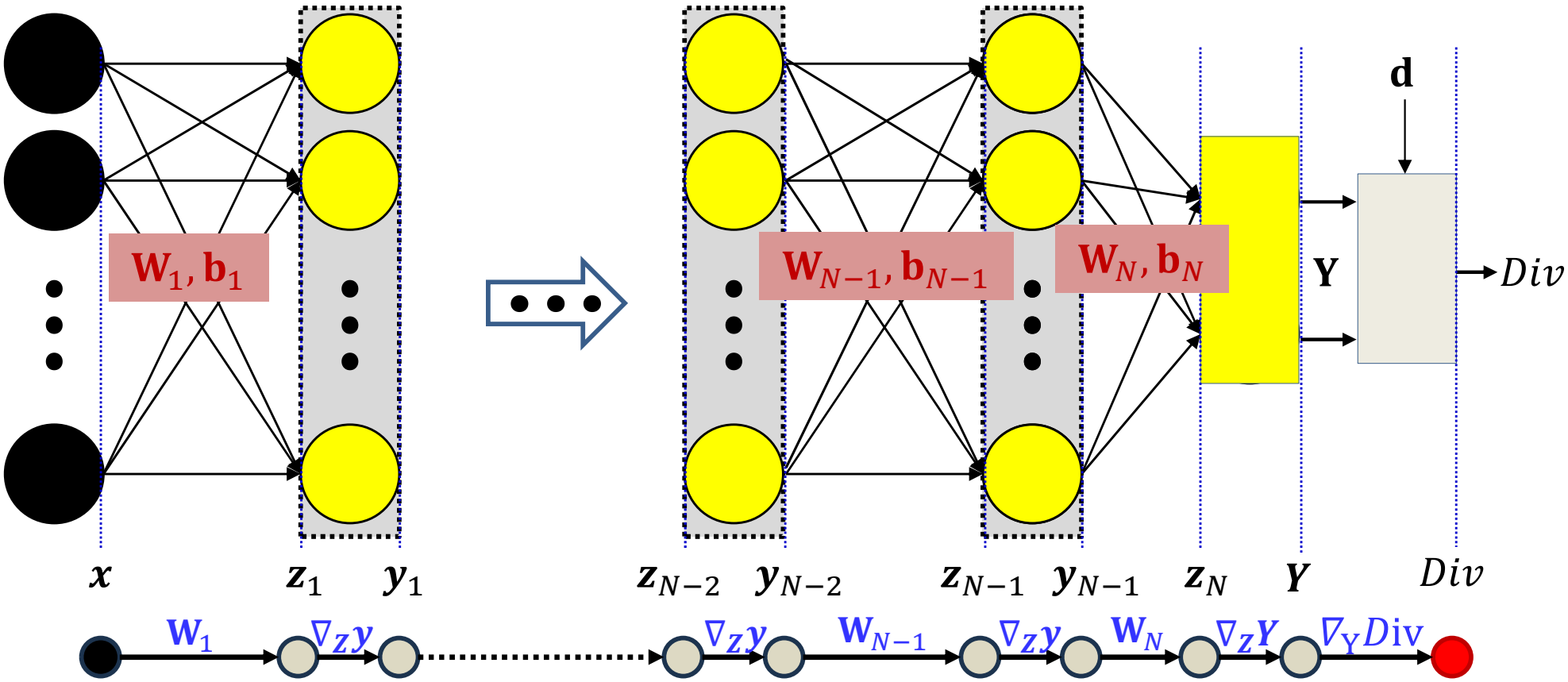

- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

# The backward pass



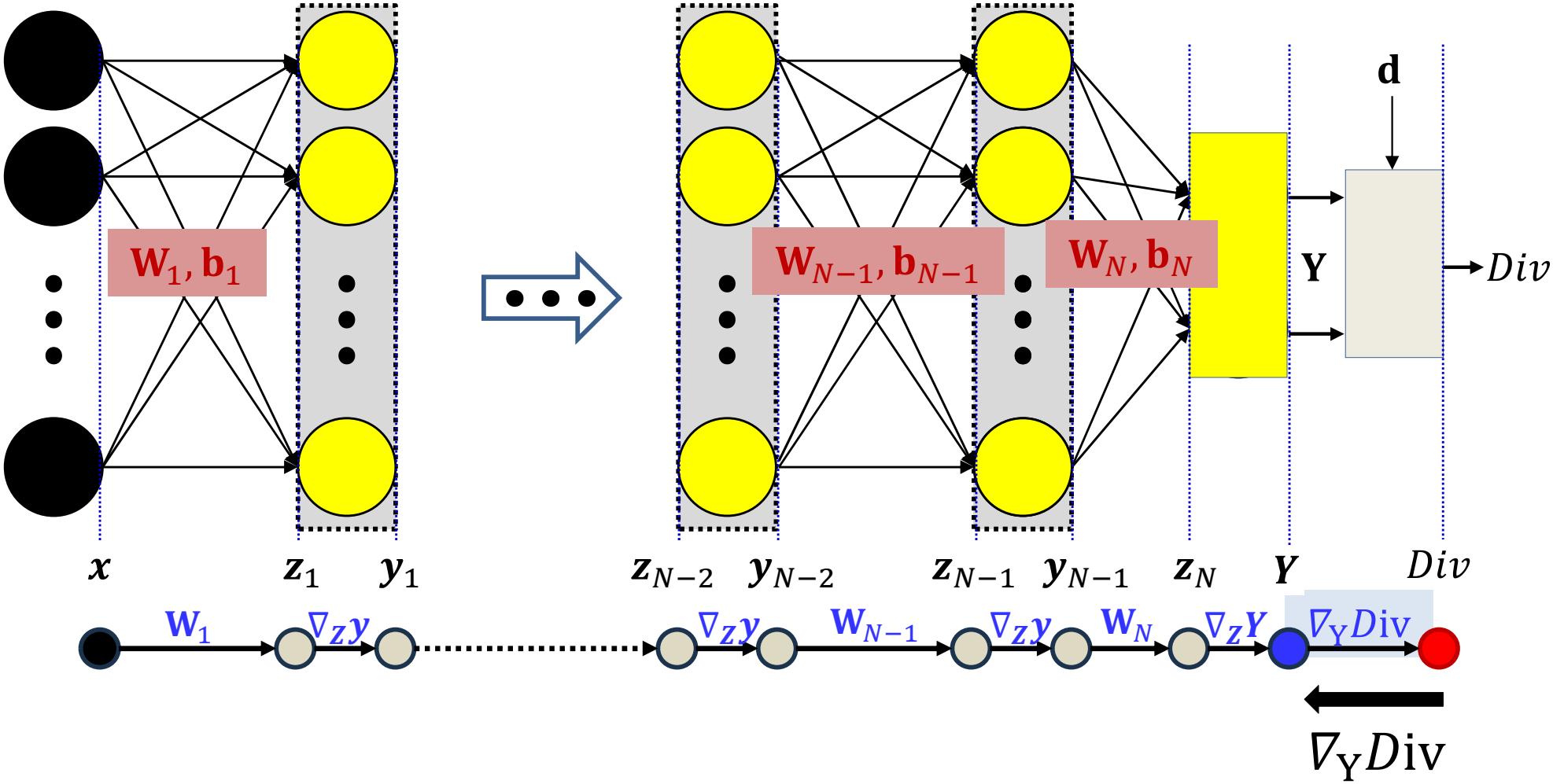

- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

# The backward pass




- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

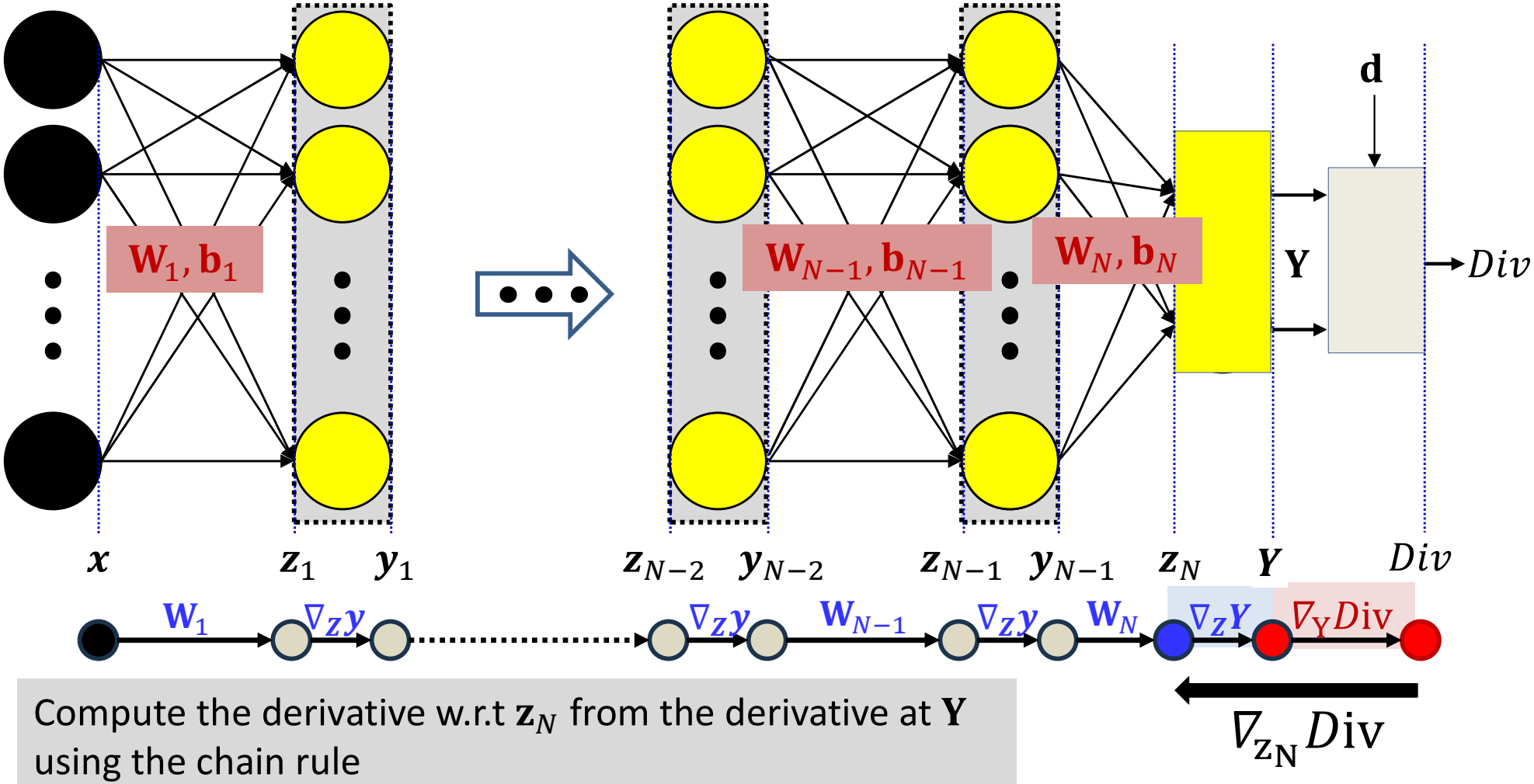
# The backward pass




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

N.B: The gradient is the transpose of the derivative

# The backward pass

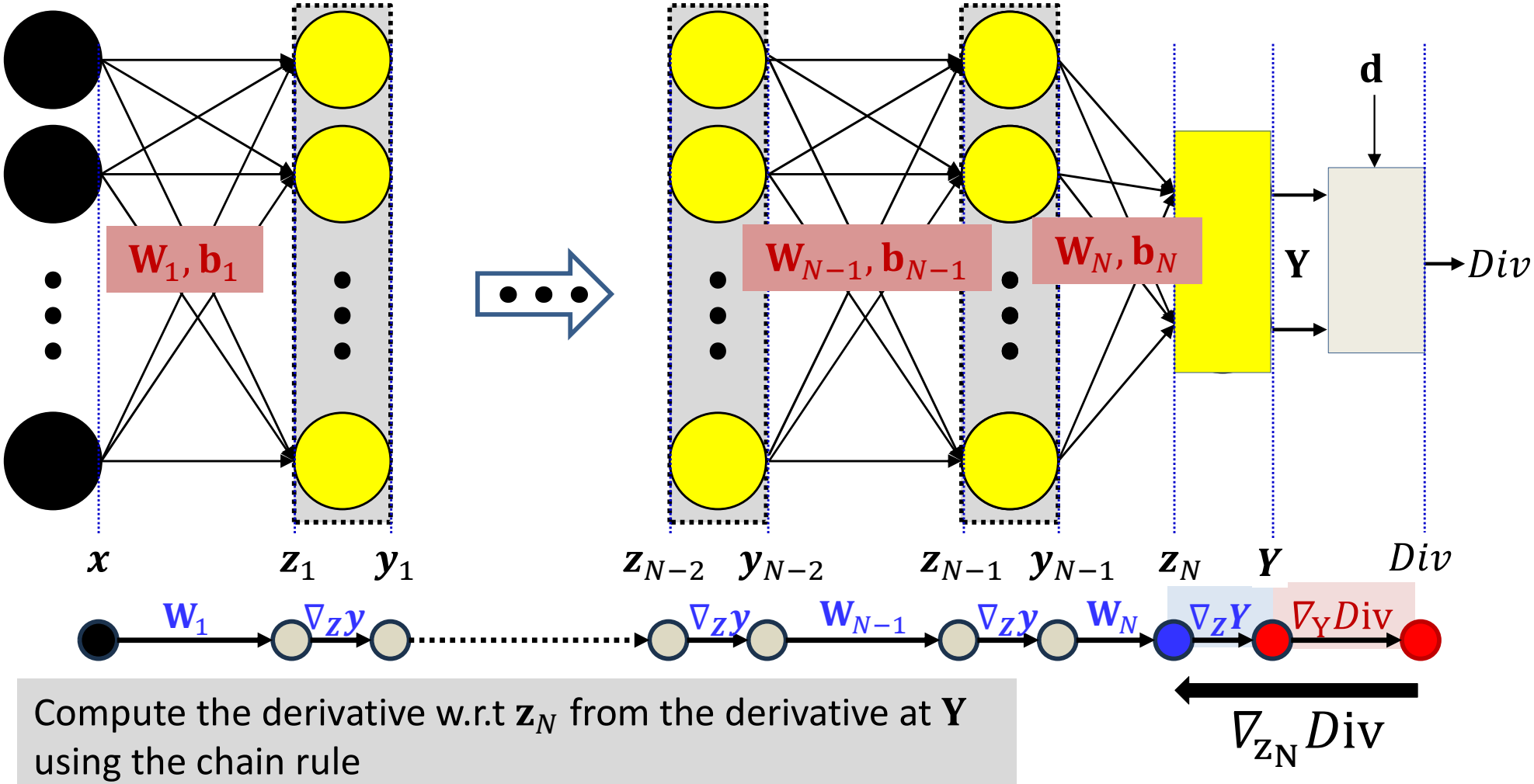



Compute the derivative w.r.t $\mathbf{z}_N$ from the derivative at $\mathbf{Y}$ using the chain rule

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

Already computed  New term

# The backward pass



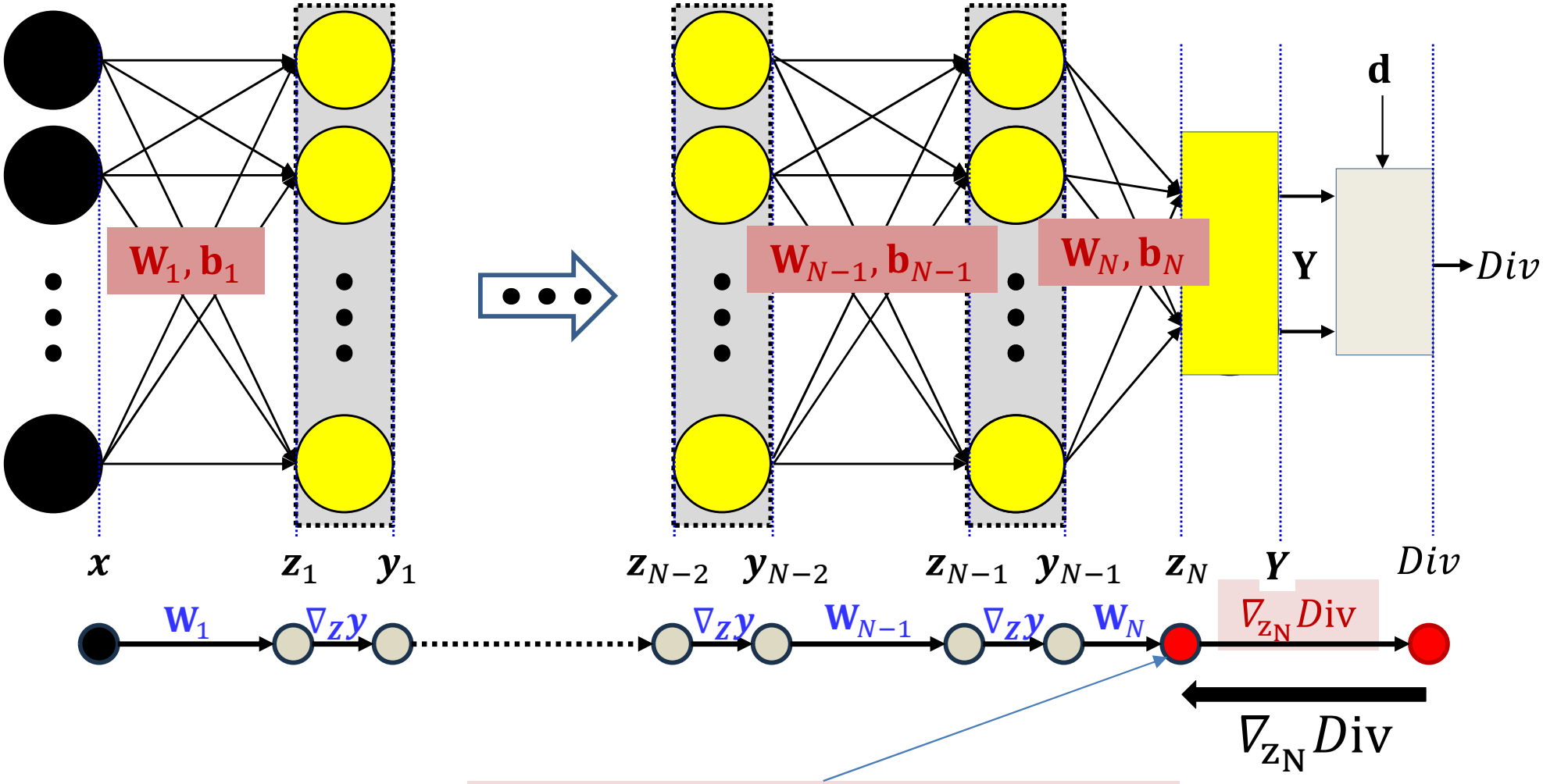

Compute the derivative w.r.t $\mathbf{z}_N$ from the derivative at $\mathbf{Y}$ using the chain rule

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

Already computed — Jacobian

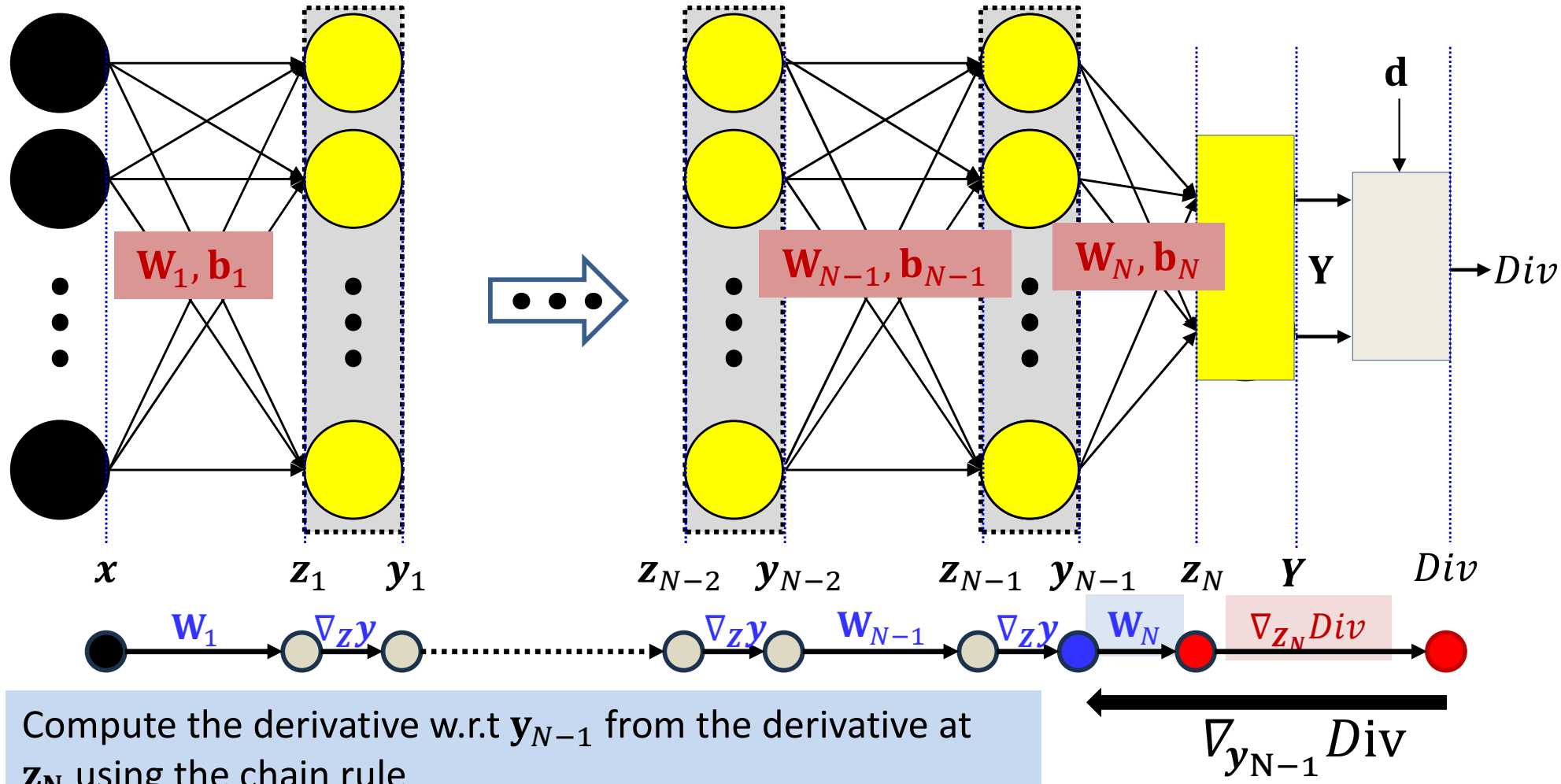
$\nabla_{\mathbf{z}_N} \mathbf{Y}$ is just the Jacobian of the activation function

# The backward pass




We now have the derivative for $z_N$

# The backward pass

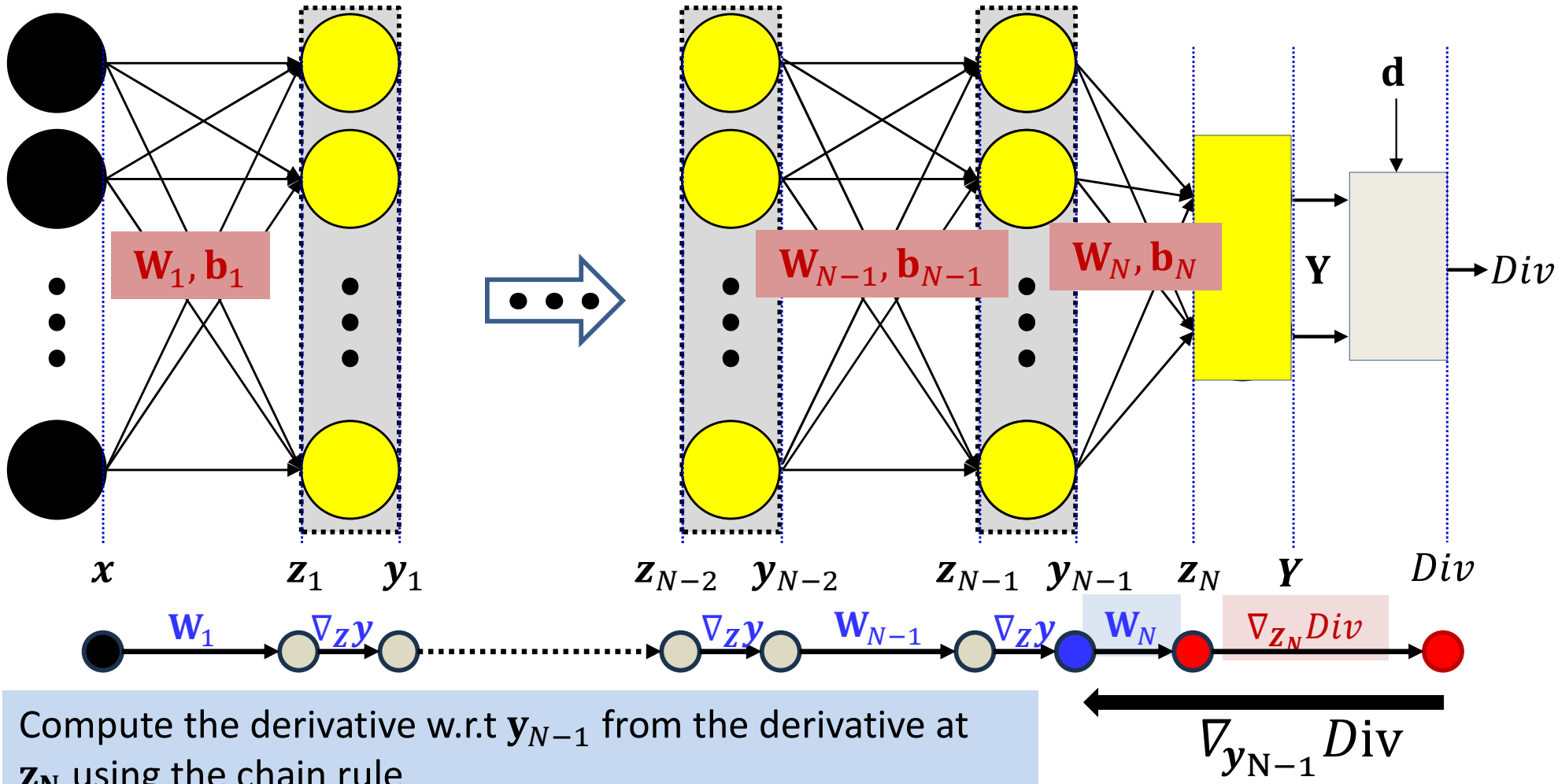



Compute the derivative w.r.t $\mathbf{y}_{N-1}$ from the derivative at $\mathbf{z}_N$ using the chain rule

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

$$\mathbf{z}_N = \mathbf{W}_N \mathbf{y}_{N-1} + \mathbf{b}_N \quad \Rightarrow \quad \nabla_{\mathbf{y}_{N-1}} \mathbf{z}_N = \mathbf{W}_N$$

# The backward pass



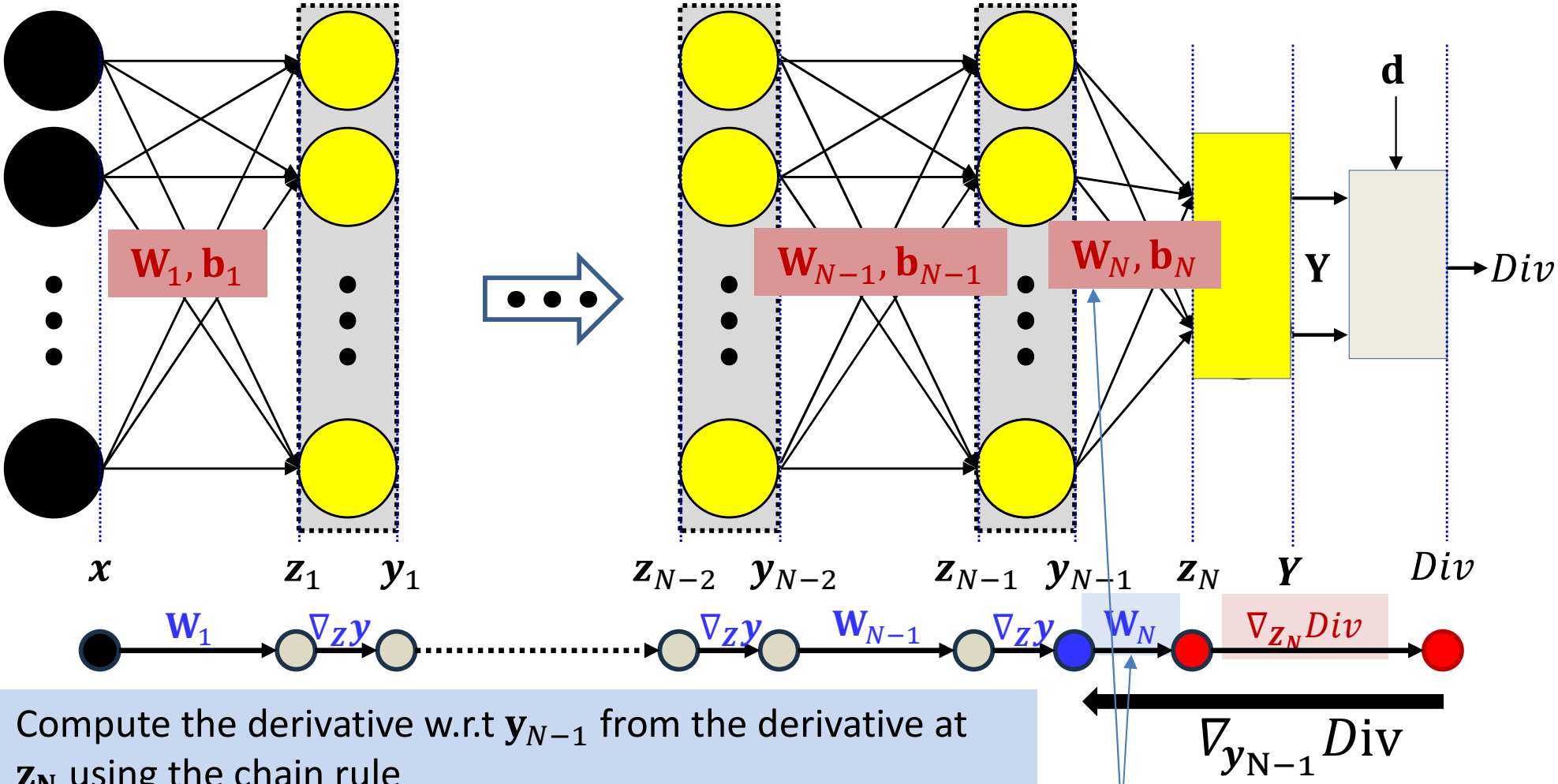

Compute the derivative w.r.t $\mathbf{y}_{N-1}$ from the derivative at $\mathbf{z}_N$ using the chain rule

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

Already computed  New term

$$\mathbf{z}_N = \mathbf{W}_N \mathbf{y}_{N-1} + \mathbf{b}_N \quad \Rightarrow \quad \nabla_{\mathbf{y}_{N-1}} \mathbf{z}_N = \mathbf{W}_N$$

# The backward pass




Compute the derivative w.r.t $\mathbf{y}_{N-1}$ from the derivative at $\mathbf{z}_N$ using the chain rule

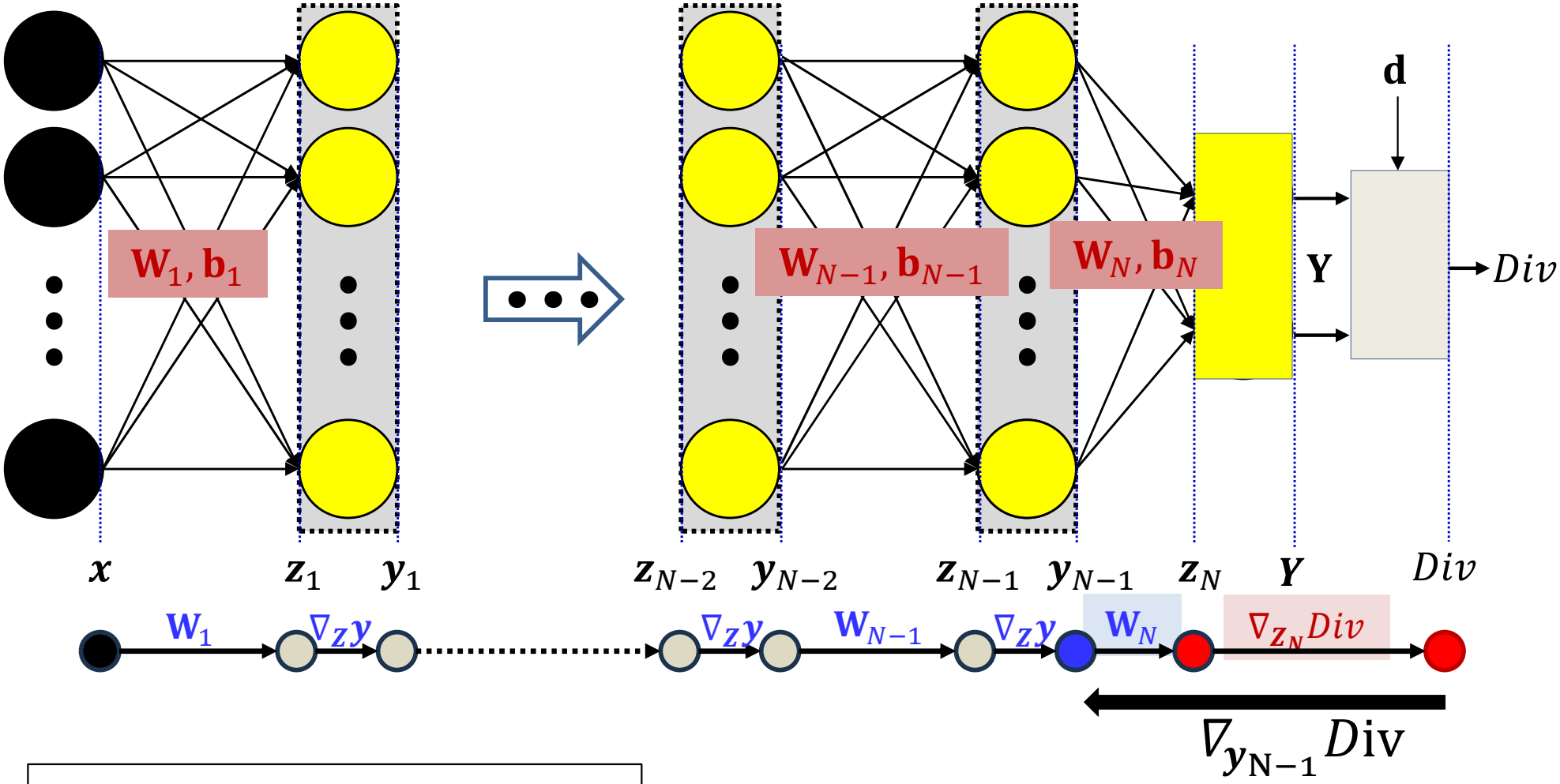
Must also compute the derivative w.r.t the $\mathbf{W}_N$ and $\mathbf{b}_N$ using the rule for affine transforms

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

Already computed  New term

$$\mathbf{z}_N = \mathbf{W}_N \mathbf{y}_{N-1} + \mathbf{b}_N \quad \Rightarrow \quad \nabla_{\mathbf{y}_{N-1}} \mathbf{z}_N = \mathbf{W}_N$$

# The backward pass




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

Affine parameter rules

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

$Div = Div(\mathbf{z})$

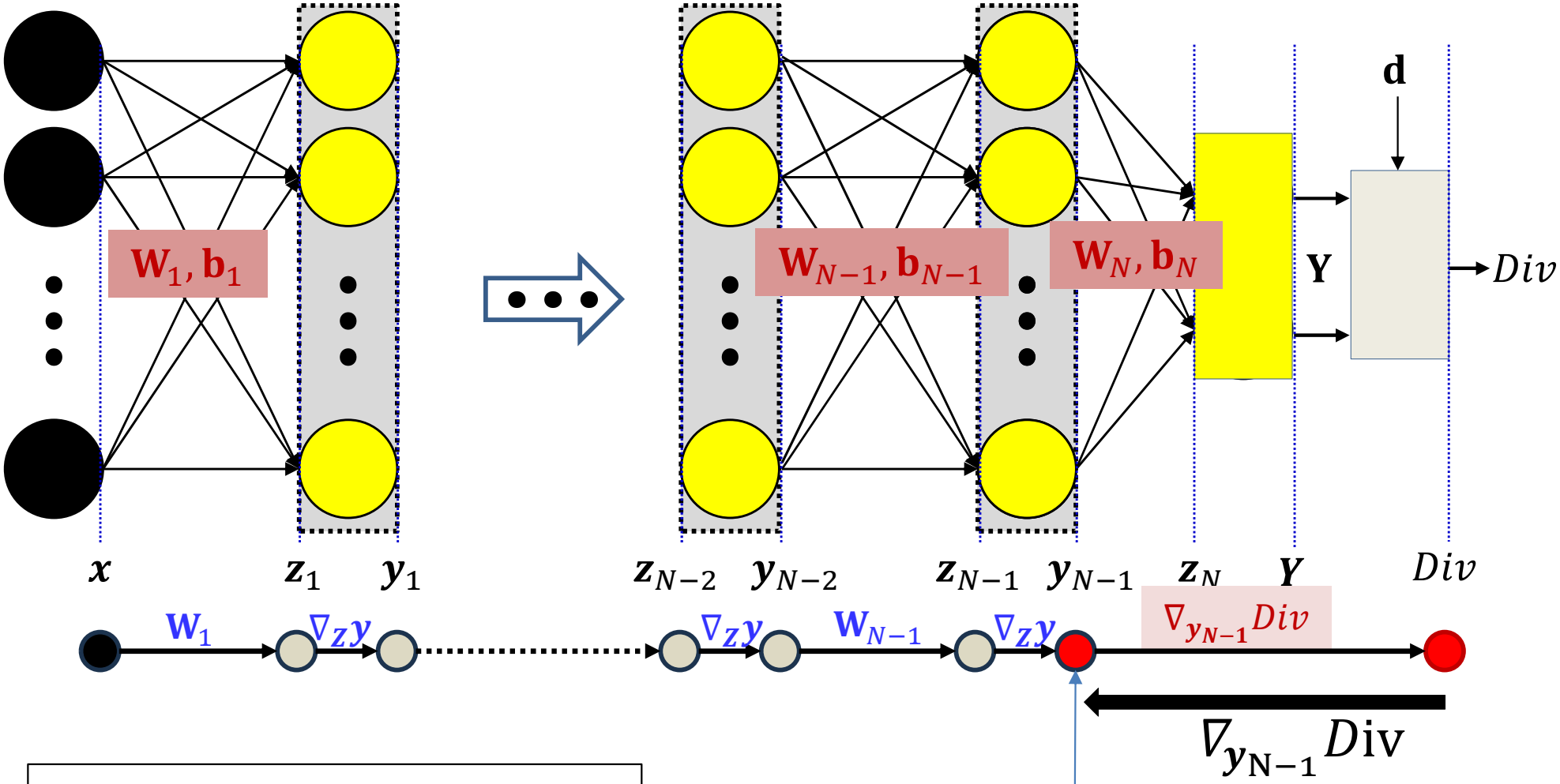
$$\nabla_{\mathbf{b}} Div = \nabla_{\mathbf{z}} Div$$

$$\nabla_{\mathbf{W}} Div = \mathbf{y} \nabla_{\mathbf{z}} Div$$

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

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

# The backward pass



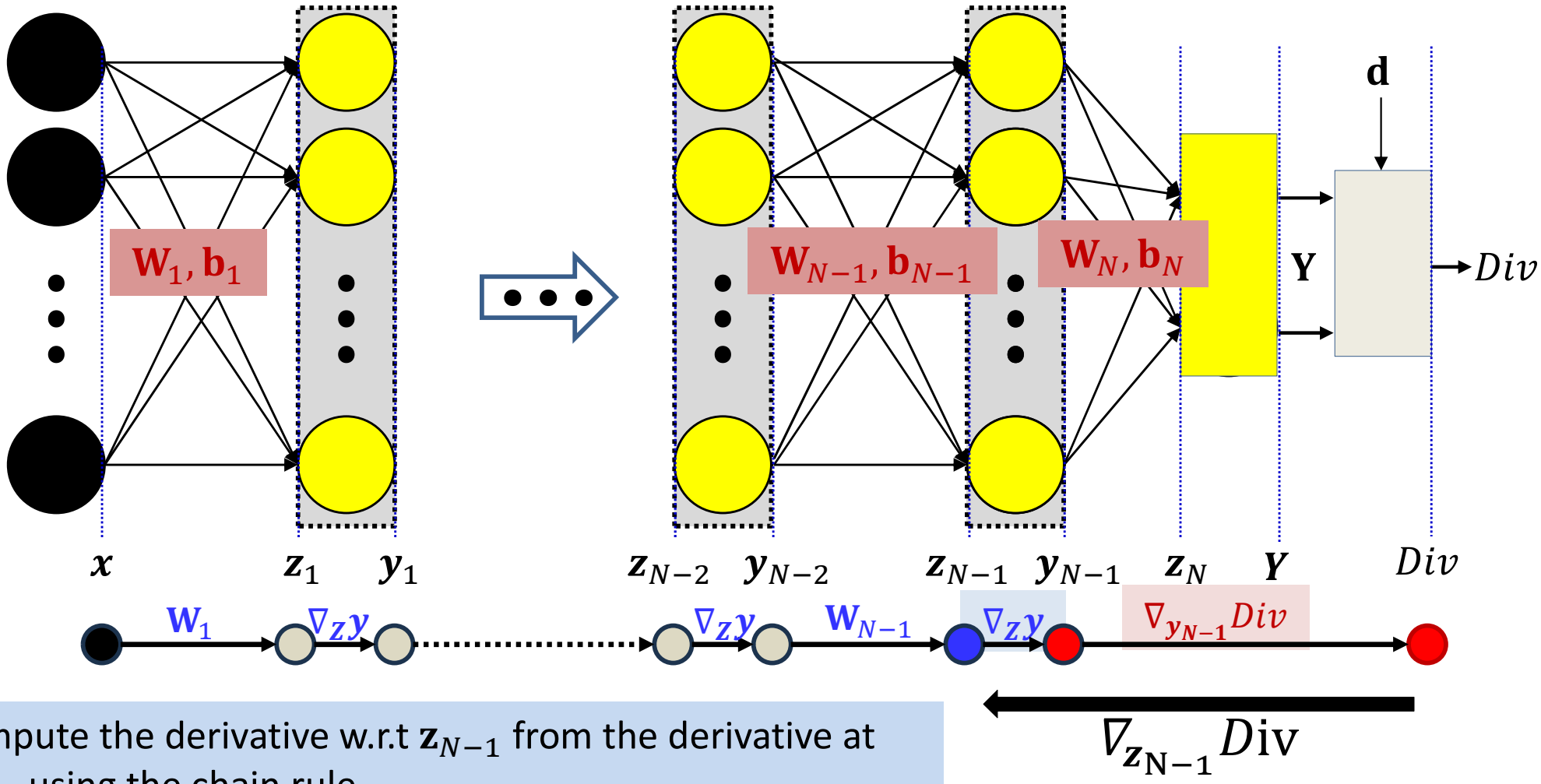

$$\nabla_{\mathbf{y}_{N-1}} Div = \nabla_{\mathbf{z}_N} Div \, \mathbf{W}_N$$

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

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

We now have the derivative for $y_{N-1}$

# The backward pass

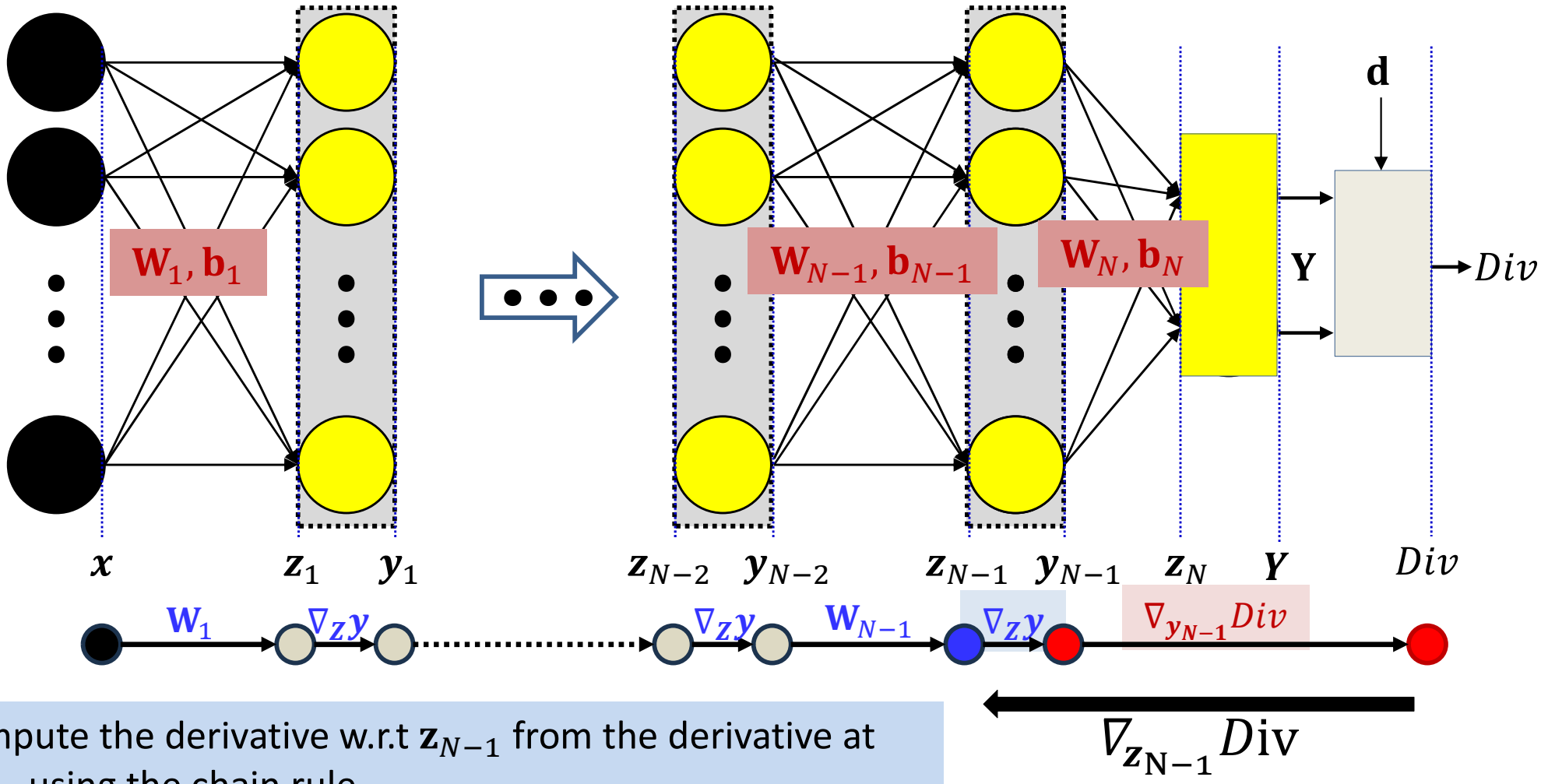



Compute the derivative w.r.t $\mathbf{z}_{N-1}$ from the derivative at $\mathbf{y}_{N-1}$ using the chain rule

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

Already computed — New term

# The backward pass



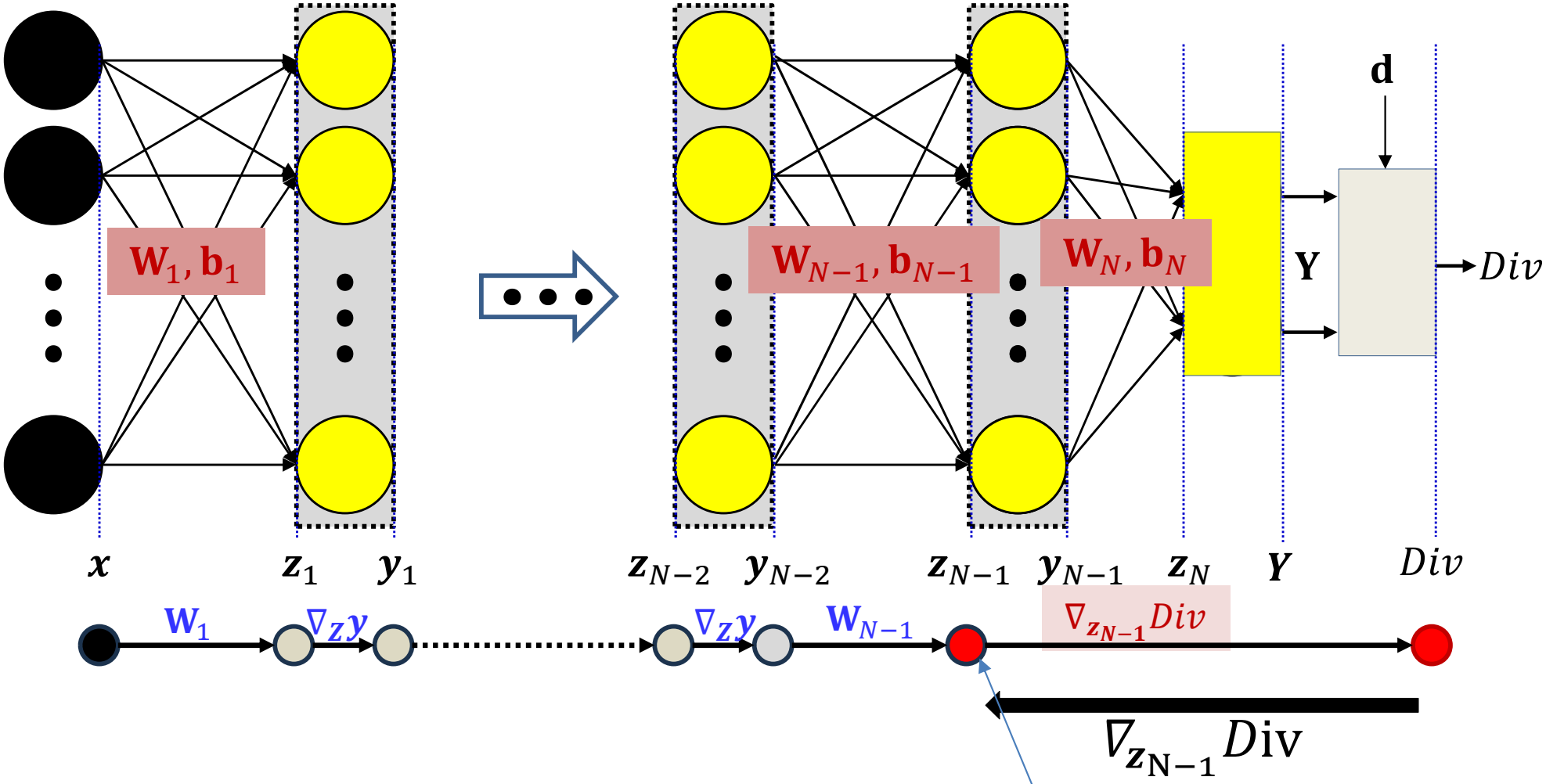

Compute the derivative w.r.t $\mathbf{z}_{N-1}$ from the derivative at $\mathbf{y}_{N-1}$ using the chain rule

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

Already computed  Jacobian

$\nabla_{\mathbf{z}_{N-1}} \mathbf{y}_{N-1}$ is the Jacobian of the activation function.
It is a diagonal matrix for scalar activations

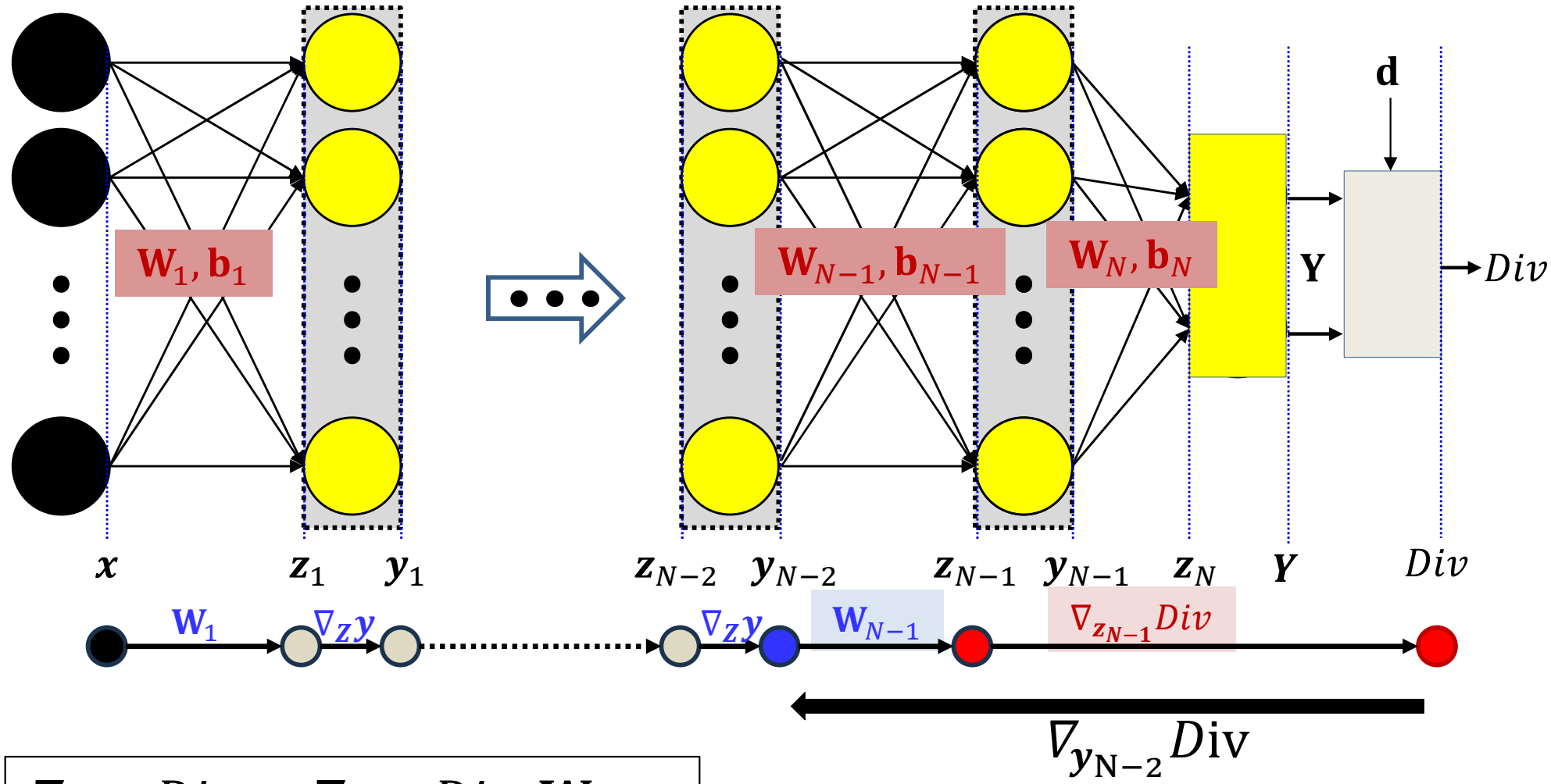
# The backward pass




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

We now have the derivative for $z_{N-1}$

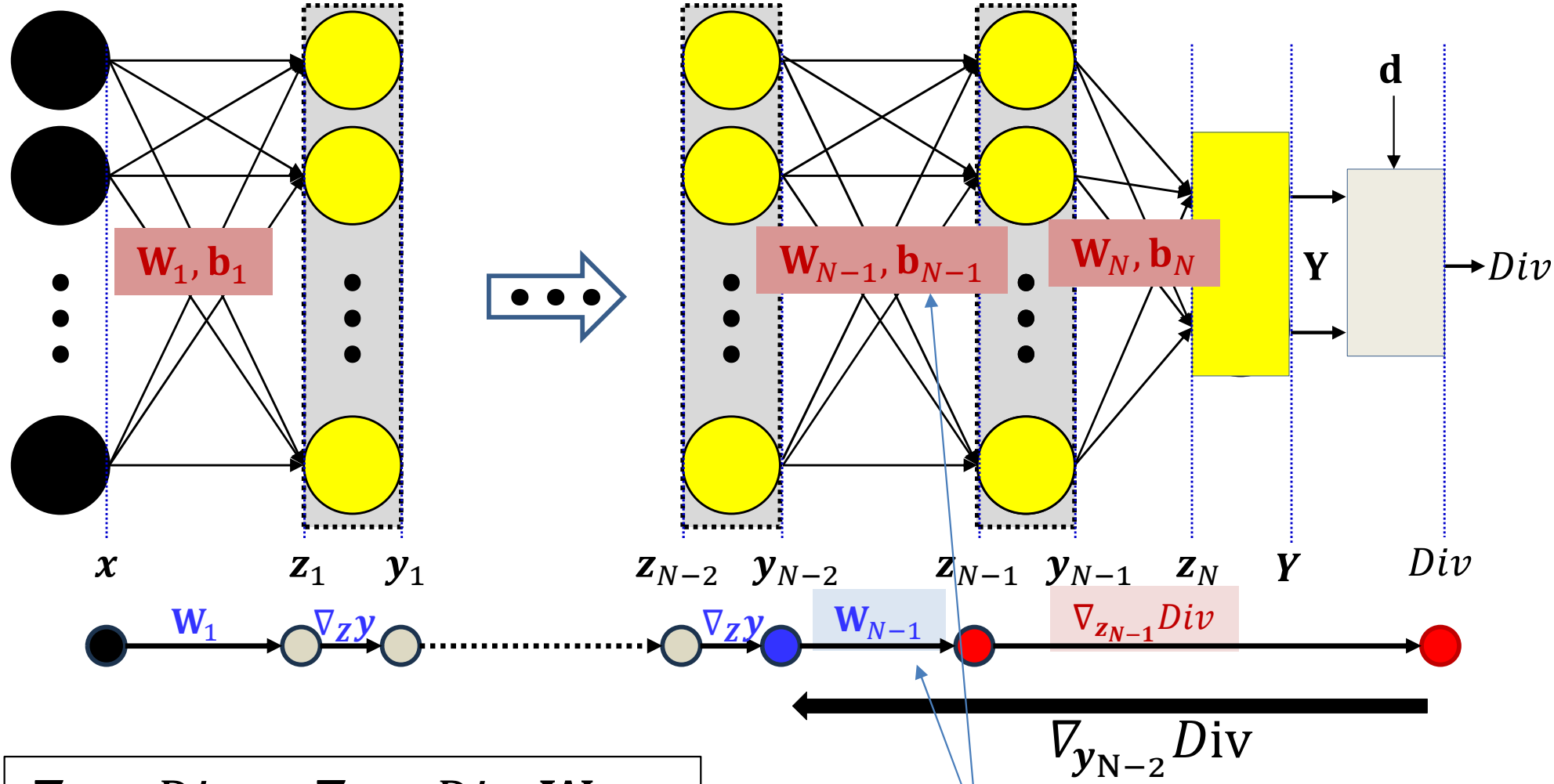
# The backward pass




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

Already computed  New

# The backward pass



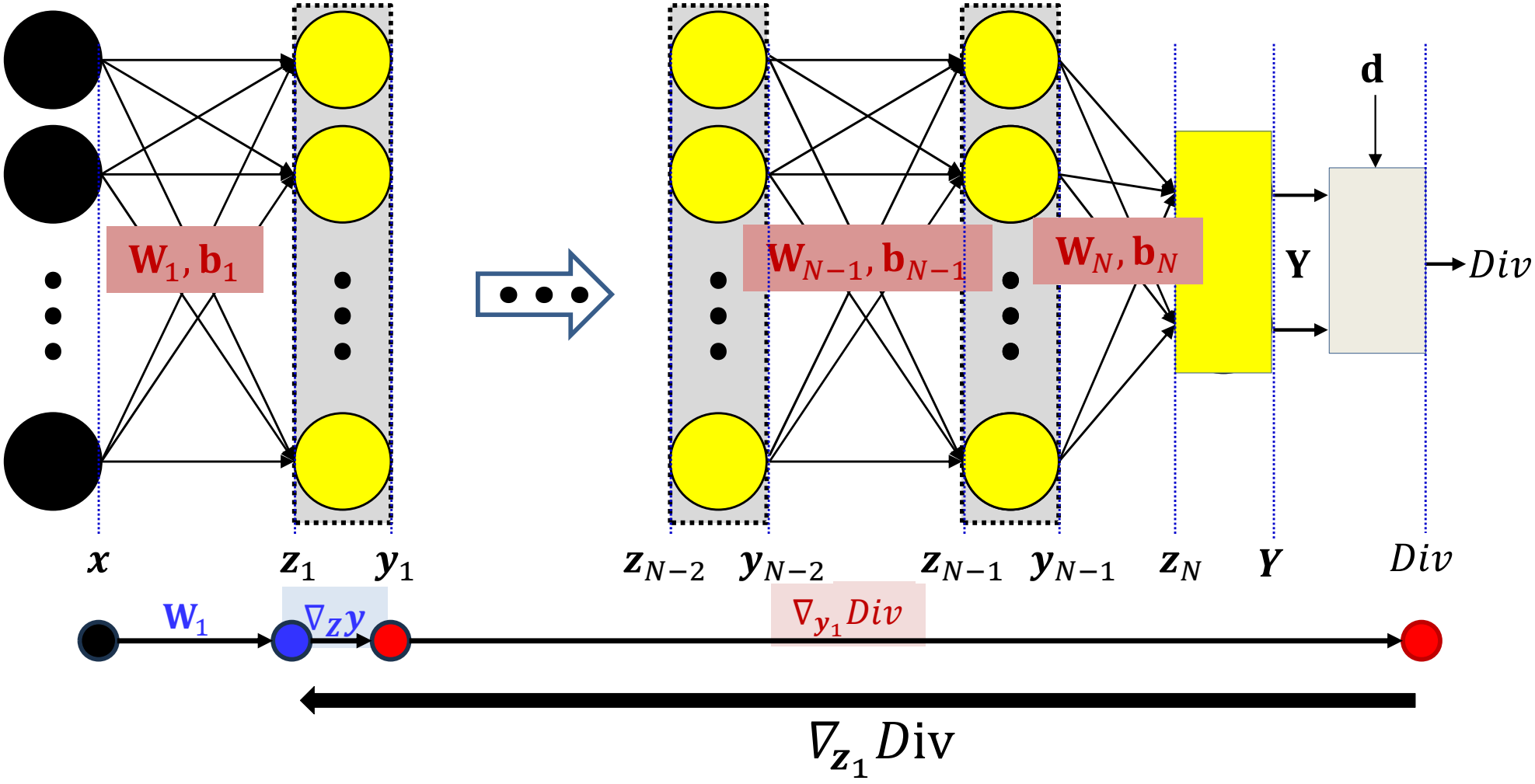

$$\nabla_{\mathbf{y}_{N-2}} Div = \nabla_{\mathbf{z}_{N-1}} Div . \mathbf{W}_{N-1}$$

Already computed  New

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

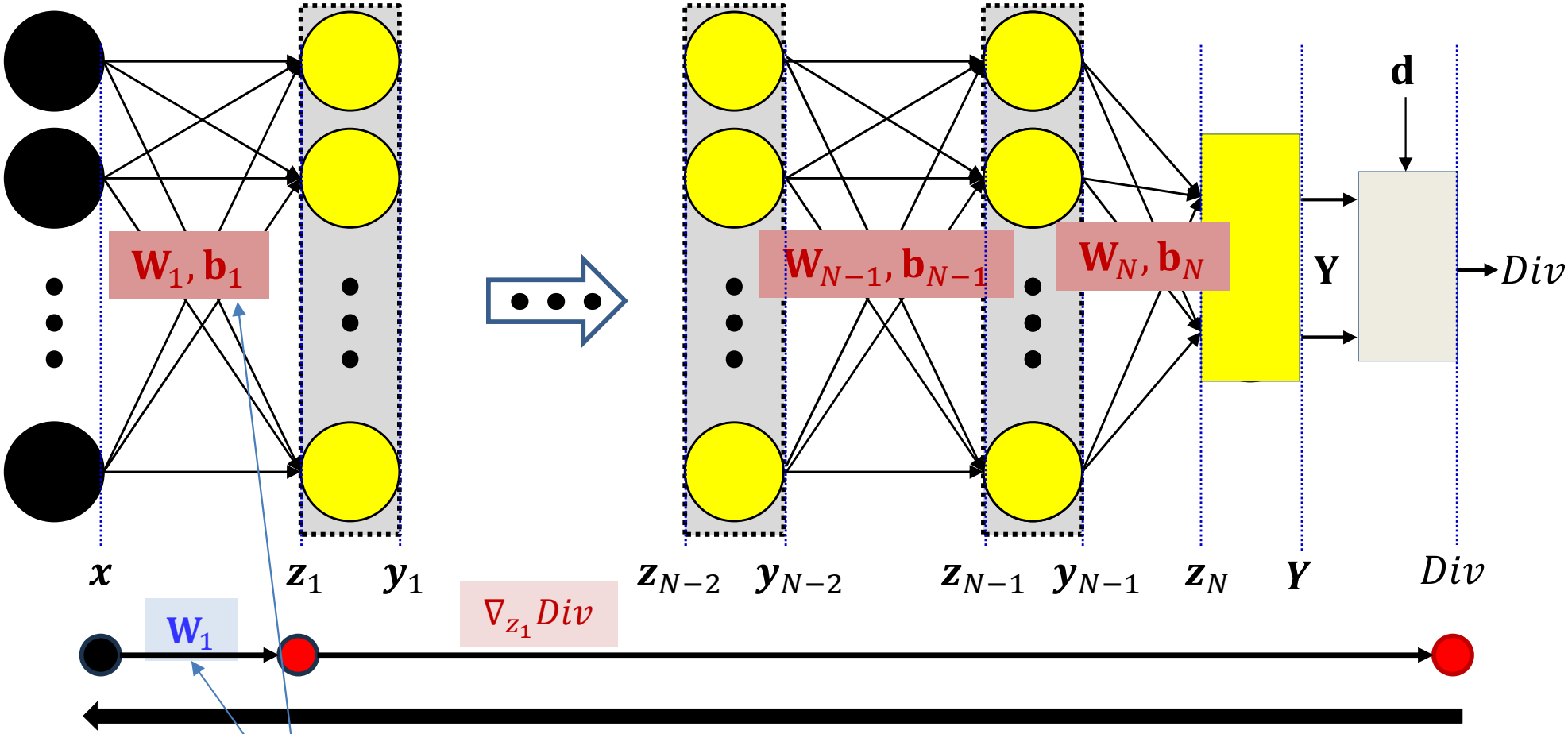
$$\nabla_{\mathbf{b}_{N-1}} Div = \nabla_{\mathbf{z}_{N-1}} Div$$

# The backward pass




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

# The backward pass

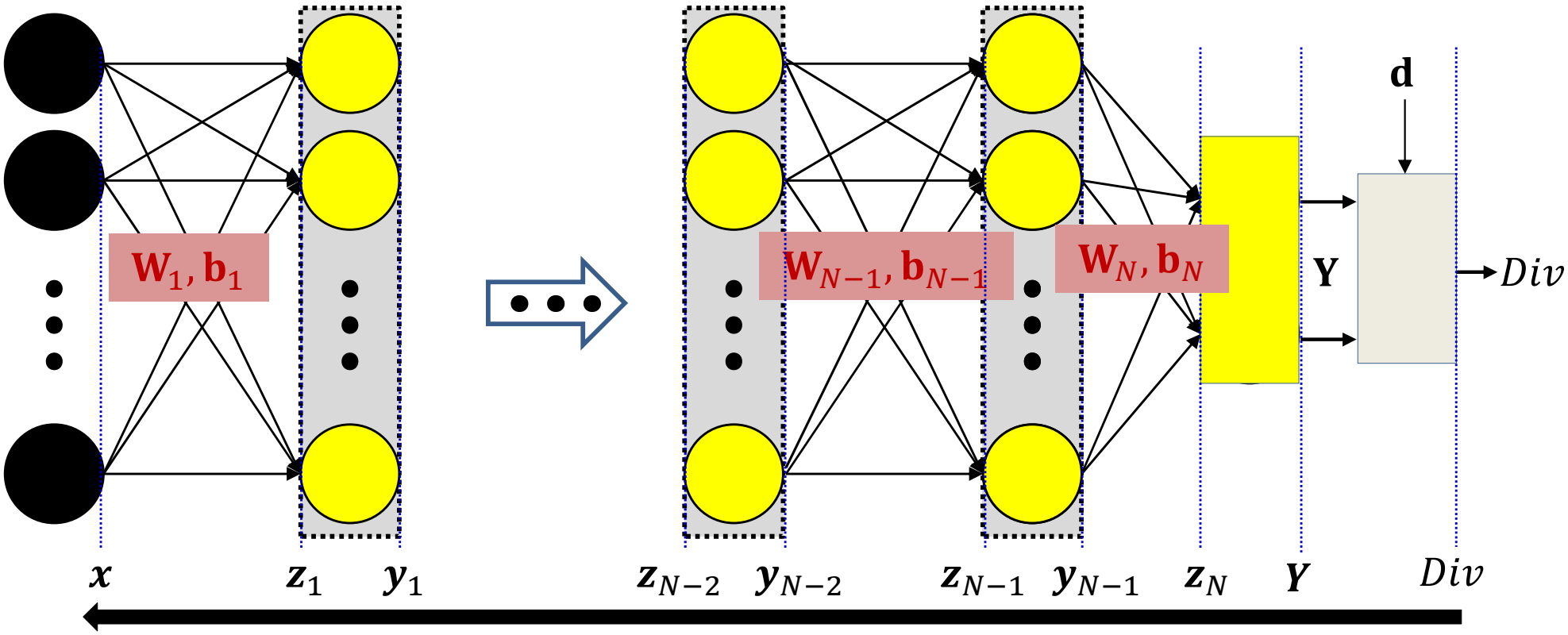



$$\nabla_{\mathbf{W}_1} Div = \mathbf{x} \nabla_{\mathbf{z}_1} Div$$

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

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

# The backward pass




Initialize:

$$\nabla_{\mathbf{y}_N} Div = \nabla_Y Div$$

For k = N downto 1:

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

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

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

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

# 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
  - Backward recursion step:

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

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

  - 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
  - Backward recursion step:

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 :
  - Forward recursion step:

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

- Output:

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

# Neural network training algorithm

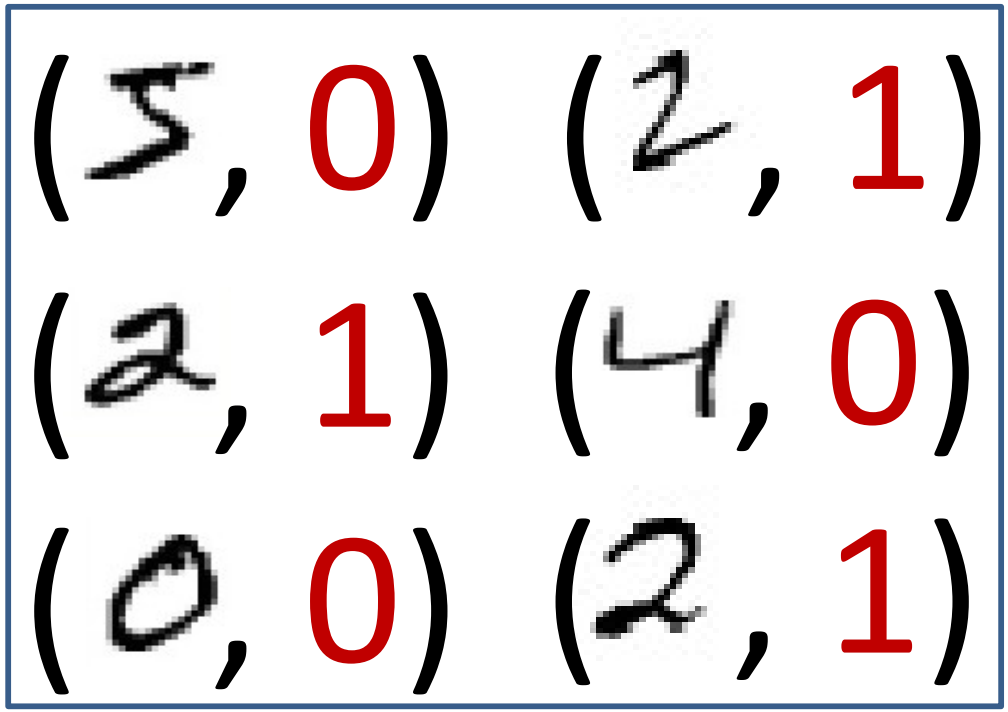
- Initialize all weights and biases $(\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \ldots, \mathbf{W}_N, \mathbf{b}_N)$
- Do:
  - $Loss = 0$
  - For all $k$, initialize $\nabla_{\mathbf{W}_k} Loss = 0$, $\nabla_{\mathbf{b}_k} Loss = 0$
  - For all $t = 1:T$ # Loop through training instances
    - Forward pass : Compute
      - Output $\boldsymbol{Y}(\boldsymbol{X_t})$
      - Divergence $\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$
      - $Loss \mathrel{+}= \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$
    - Backward pass: For all $k$ compute:
      - $\nabla_{\mathbf{y}_k} Div = \nabla_{\mathbf{z}_{k+1}} Div \, \mathbf{W}_{k+1}$
      - $\nabla_{\mathbf{z}_k} Div = \nabla_{\mathbf{y}_k} Div \, J_{\mathbf{y}_k}(\mathbf{z}_k)$
      - $\nabla_{\mathbf{W}_k} \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t}) = \mathbf{y}_{k-1} \nabla_{\mathbf{z}_k} Div$; $\nabla_{\mathbf{b}_k} \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t}) = \nabla_{\mathbf{z}_k} Div$
      - $\nabla_{\mathbf{W}_k} Loss \mathrel{+}= \nabla_{\mathbf{W}_k} \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$; $\nabla_{\mathbf{b}_k} Loss \mathrel{+}= \nabla_{\mathbf{b}_k} \boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})$
  - For all $k$, update:

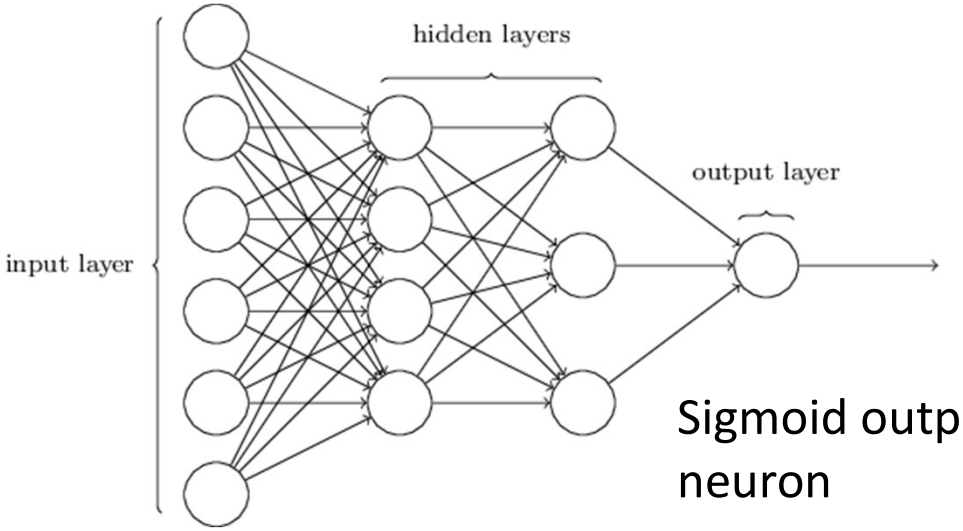
$$\mathbf{W}_k = \mathbf{W}_k - \frac{\eta}{T} \left(\nabla_{\mathbf{W}_k} Loss\right)^T; \qquad \mathbf{b}_k = \mathbf{b}_k - \frac{\eta}{T} \left(\nabla_{\mathbf{W}_k} Loss\right)^T$$

- Until $Loss$ has converged

# Setting up for digit recognition

Training data



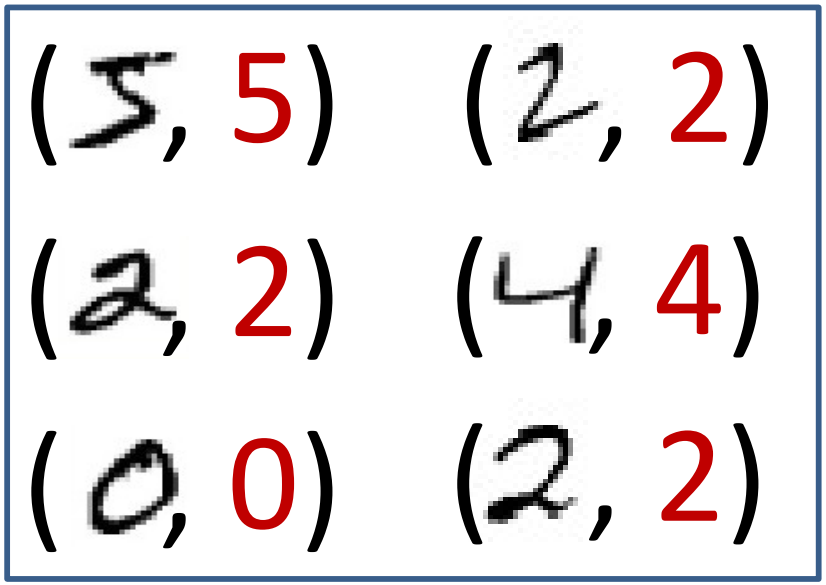



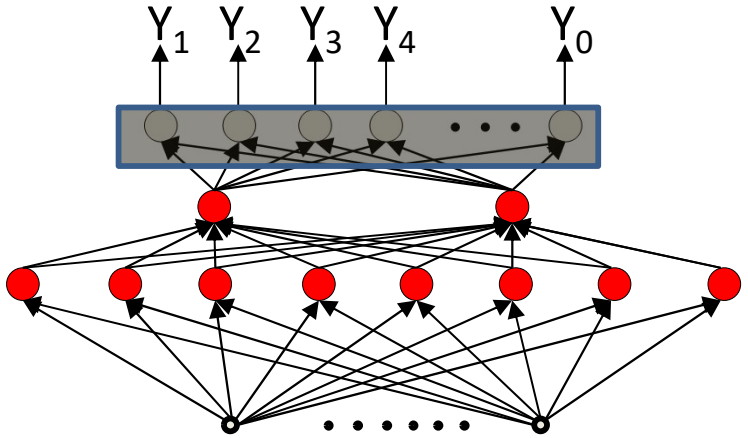

Sigmoid output neuron

- Simple Problem: Recognizing "2" or "not 2"
- Single output with sigmoid activation
  - $Y \in (0,1)$
  - $d$ *is either* $0$ *or* $1$
- Use KL divergence
- Backpropagation to compute derivatives
  - To apply in gradient descent to learn network parameters

# Recognizing the digit

Training data




- More complex problem: Recognizing digit
- Network with 10 (or 11) outputs
  - First ten outputs correspond to the ten digits
    - Optional 11th is for none of the above
- Softmax output layer:
  - Ideal output: One of the outputs goes to 1, the others go to 0
- Backpropagation with KL divergence
  - To compute derivatives for gradient descent updates to learn network

# Story so far

- Neural networks must be trained to minimize the average divergence between the output of the network and the desired output over a set of training instances, with respect to network parameters.
- Minimization is performed using gradient descent
- Gradients (derivatives) of the divergence (for any individual instance) w.r.t. network parameters can be computed using backpropagation
  - Which requires a “forward” pass of inference followed by a “backward” pass of gradient computation
- The computed gradients can be incorporated into gradient descent

# Issues

- Convergence: How well does it learn
  - And how can we improve it
- How well will it generalize (outside training data)
- What does the output really mean?
- *Etc.*.

# Next up

- Convergence and generalization