COMP417 Lecture 7

Learning the Network: Backpropagation Part 2

Dr. Hend Dawood

Training neural nets: Problem Setup

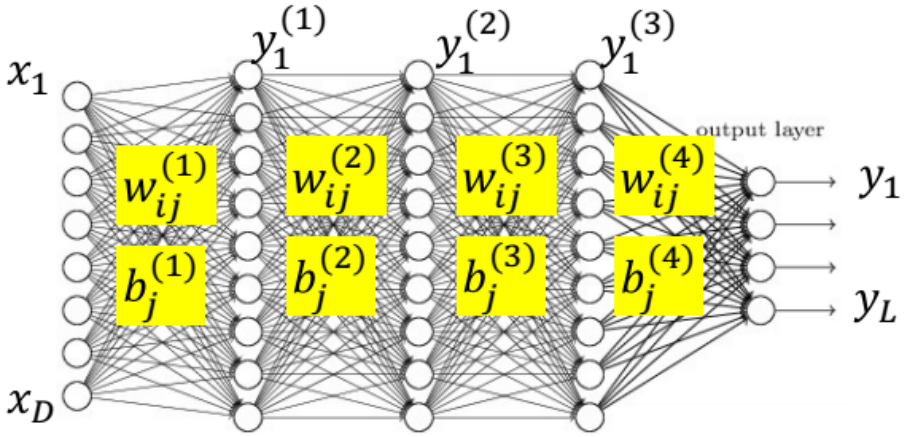
- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$

- 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 $b_j^{(k)}$

Gradient descent/ascent (multivariate)

- The gradient descent/ascent method to find the minimum or maximum of a function f iteratively
  - To find a *maximum* move *in the direction of the gradient*

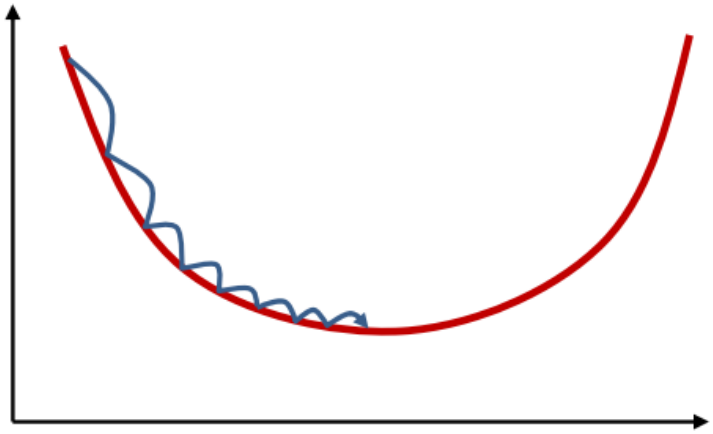
$$x^{k+1} = x^k + \eta^k \nabla_x f\left(x^k\right)^T$$

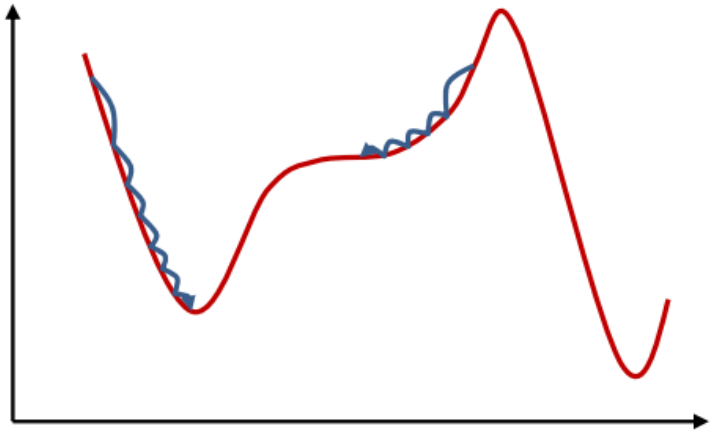
  - To find a *minimum* move *exactly opposite the direction of the gradient*

$$x^{k+1} = x^k - \eta^k \nabla_x f\left(x^k\right)^T$$

- Many solutions to choosing step size η^k

Convergence of Gradient Descent





- For appropriate step size, for convex (bowl-shaped) functions gradient descent will always find the minimum.
- For non-convex functions it will find a local minimum or an inflection point

Gradient Descent Algorithm

- Initialize:
  - W^0
  - $k = 0$

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

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

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

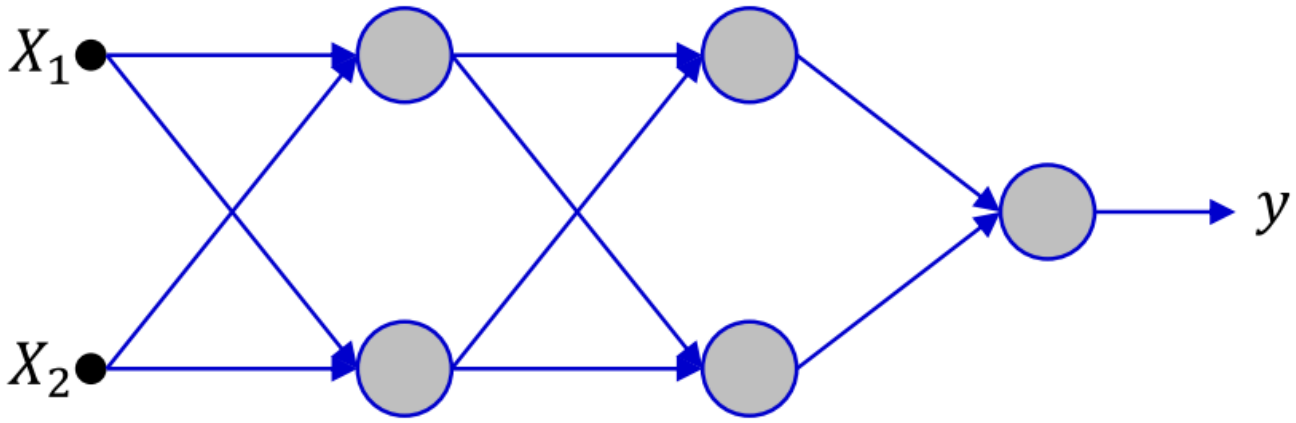
$$\frac{dLoss}{dw_{i,j}^{(k)}} = \frac{1}{T}\sum_{t} \frac{dDiv(\boldsymbol{Y}_t, \boldsymbol{d}_t)}{dw_{i,j}^{(k)}}$$

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

Our problem for today

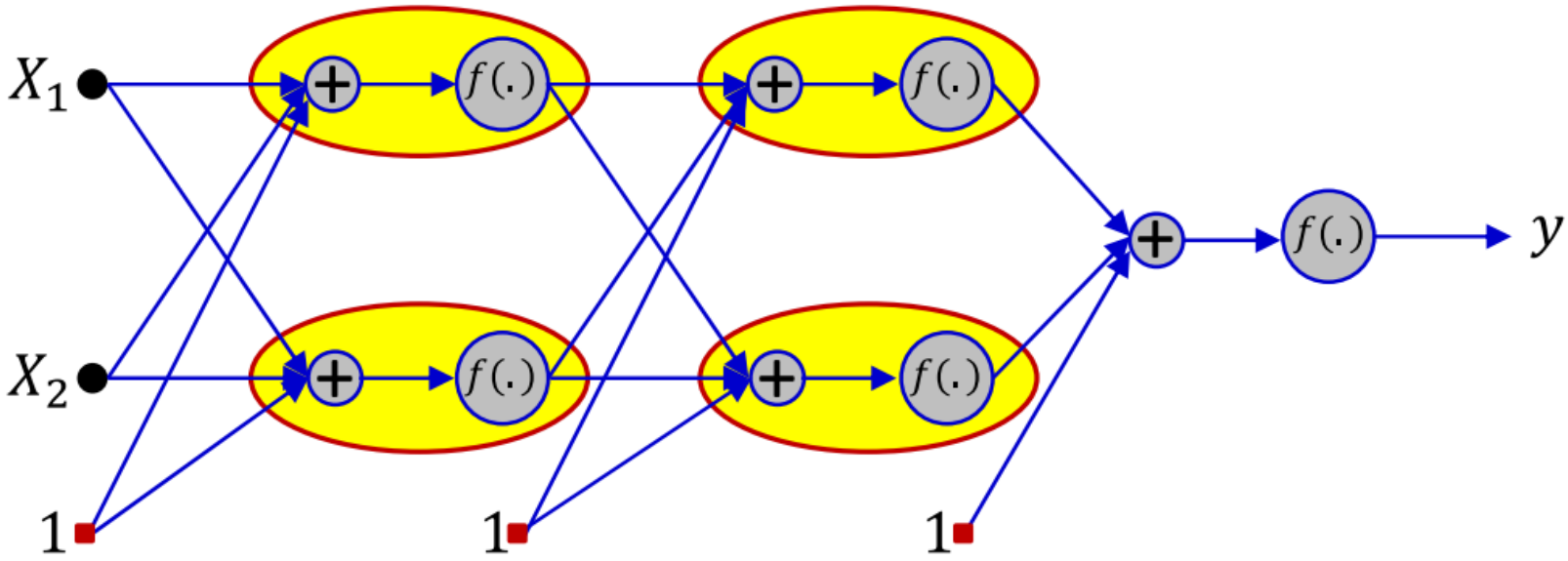
- How to compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y},\boldsymbol{d})}{dw_{i,j}^{(k)}}$ for a single data instance

A first closer look at the network



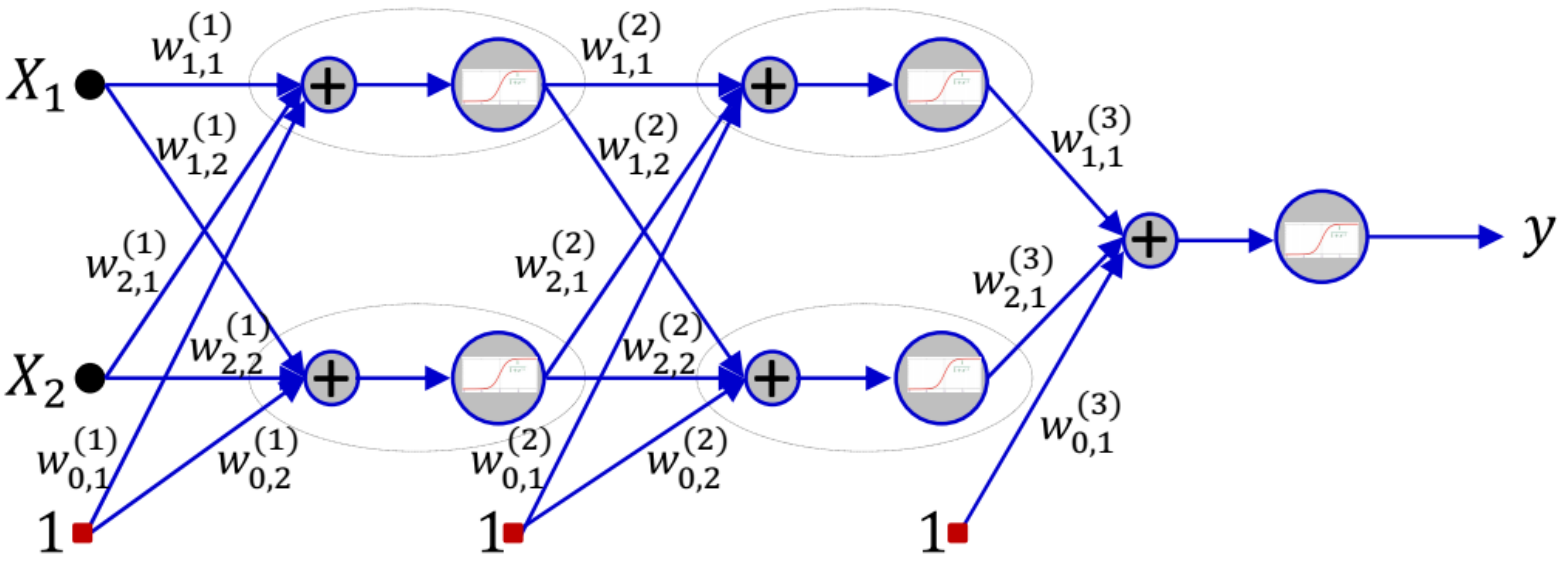
- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs

A first closer look at the network



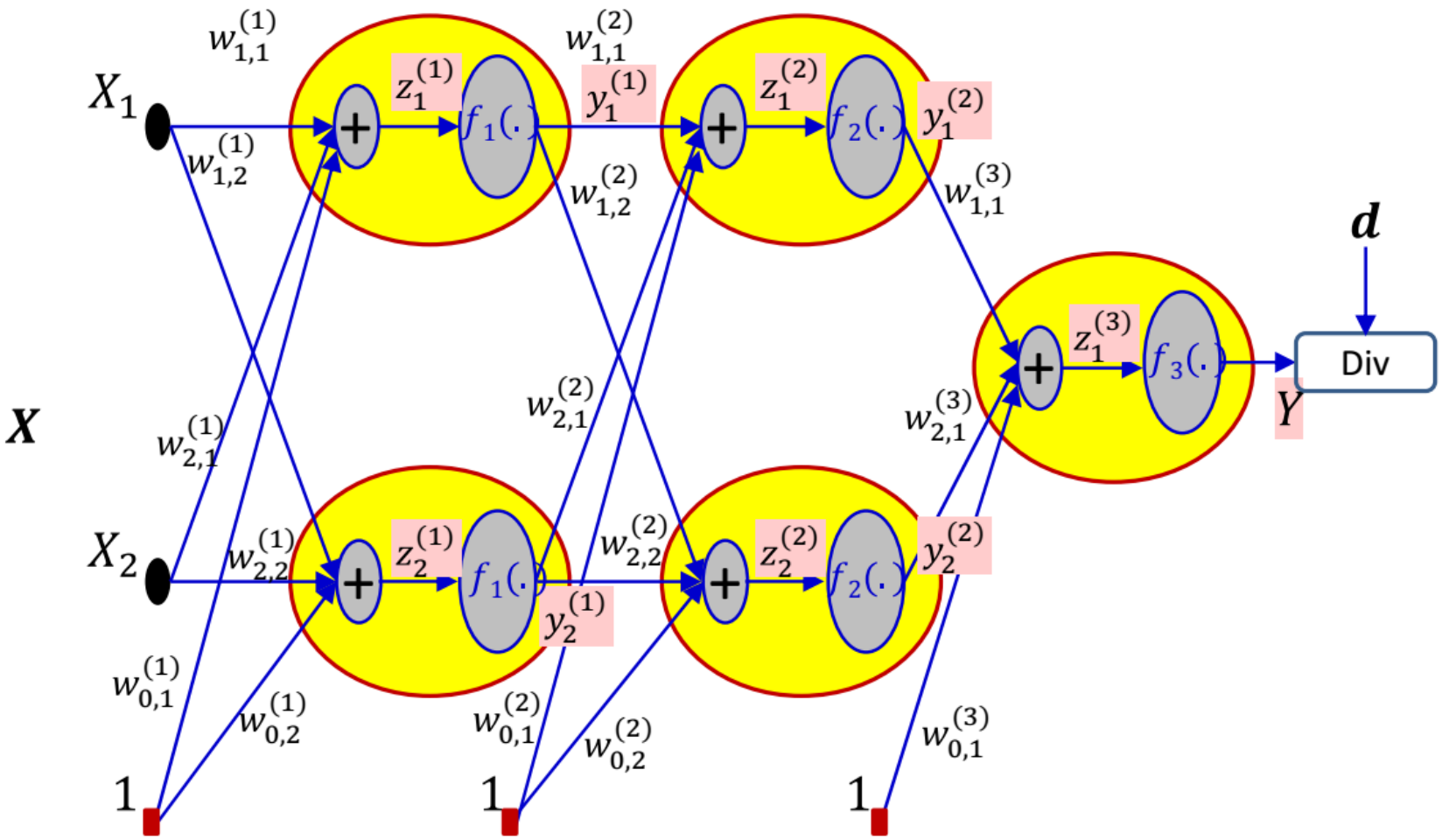
- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Explicitly separating the affine function of inputs from the activation

A first closer look at the network

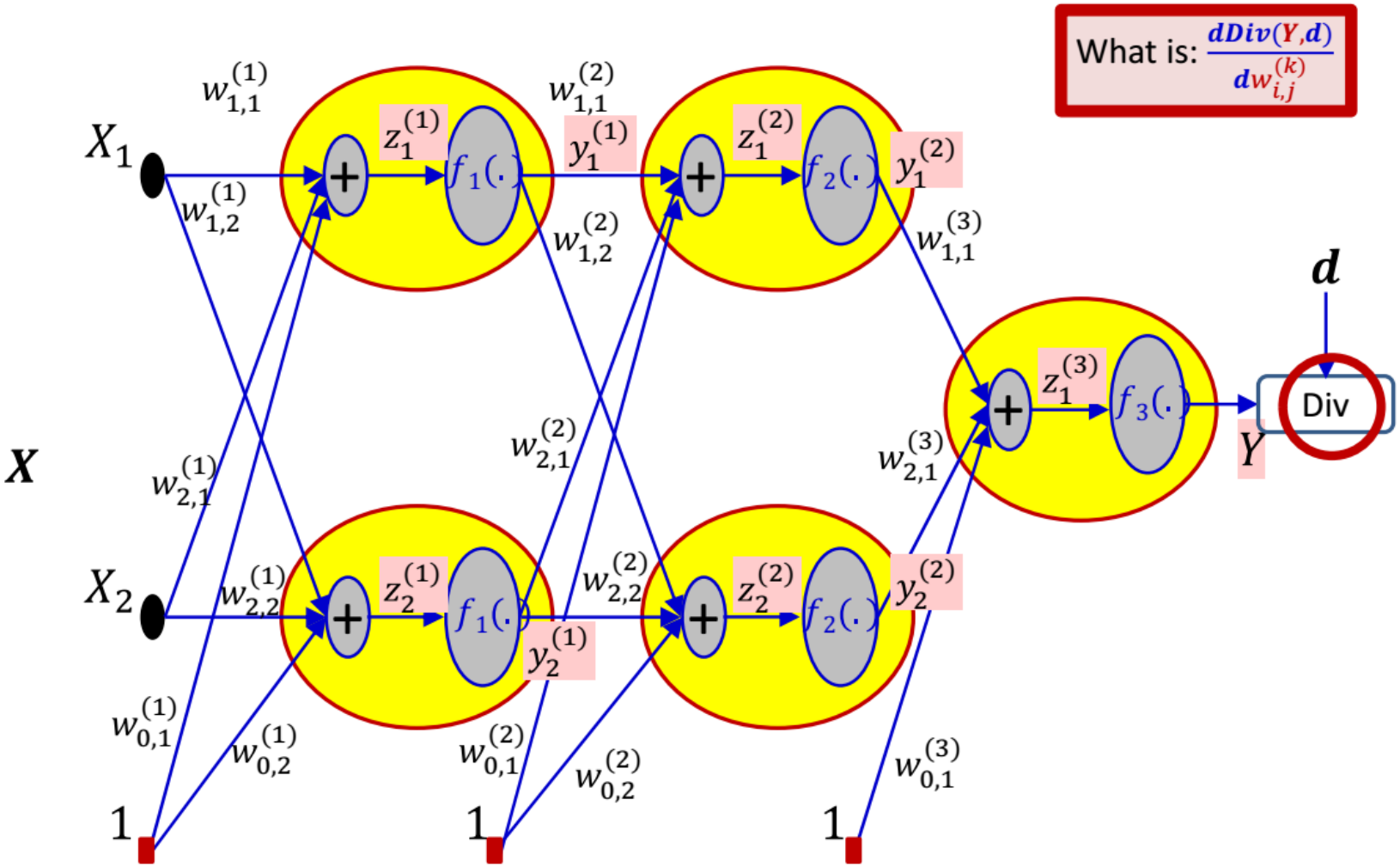


- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Expanded **with all weights shown**
- Let's label the other variables too…

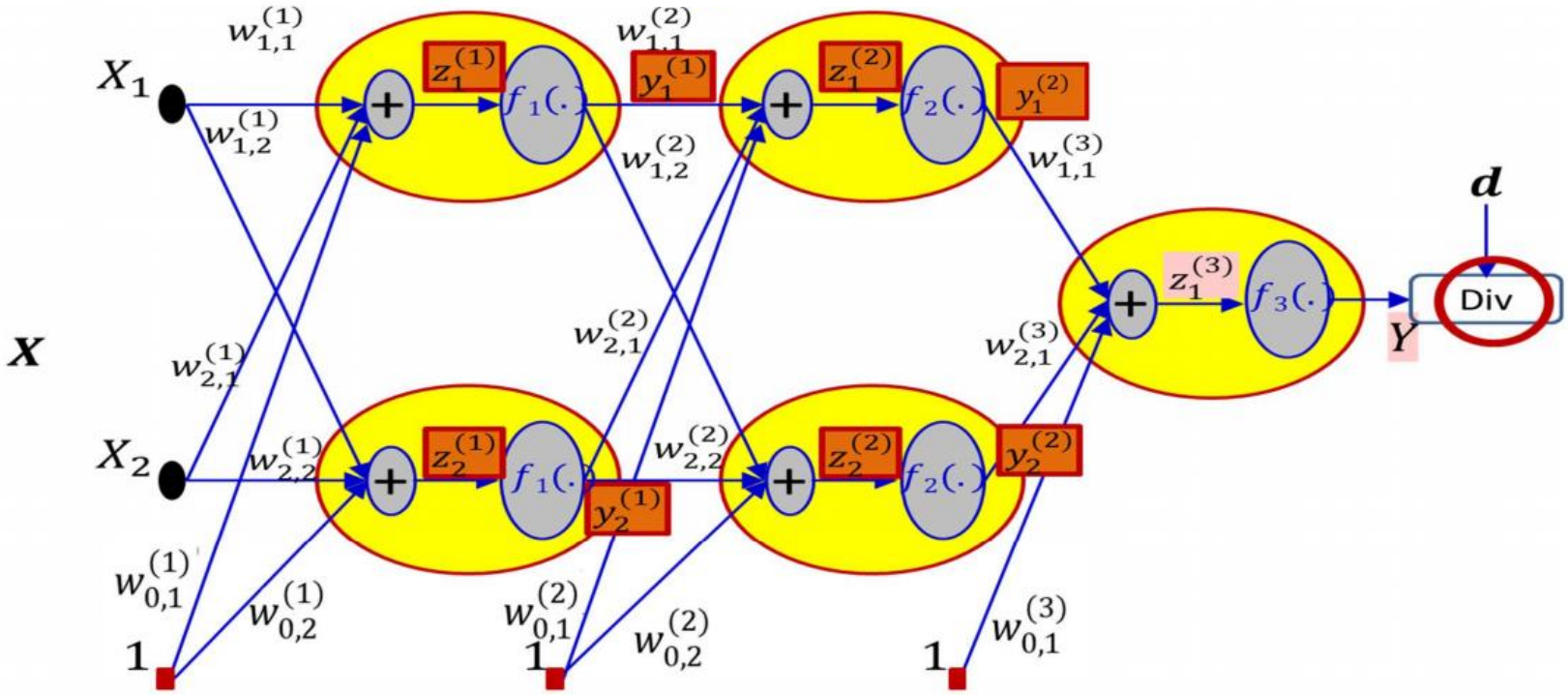
Computing the derivative for a single input



Computing the derivative for a single input



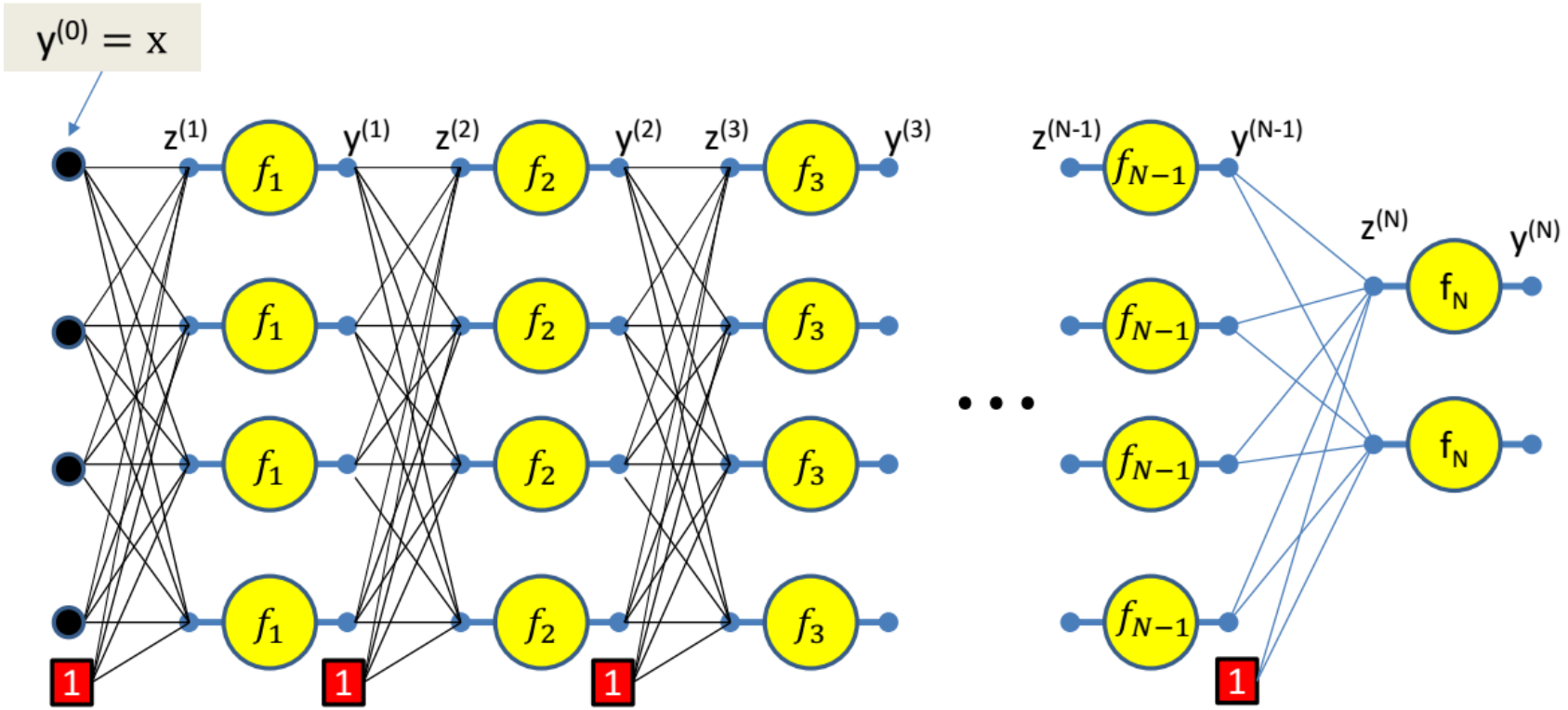
Computing the gradient



- 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

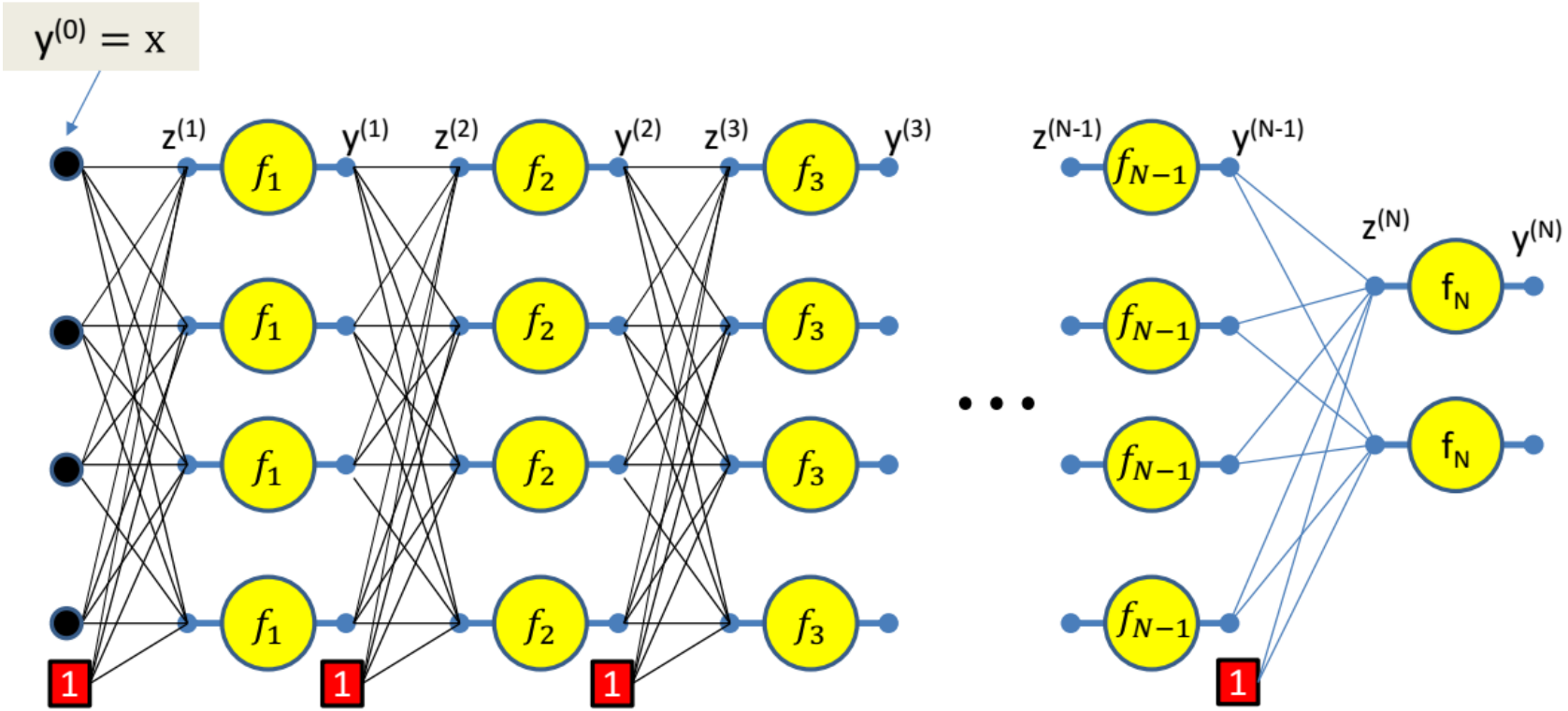
The "forward pass"



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

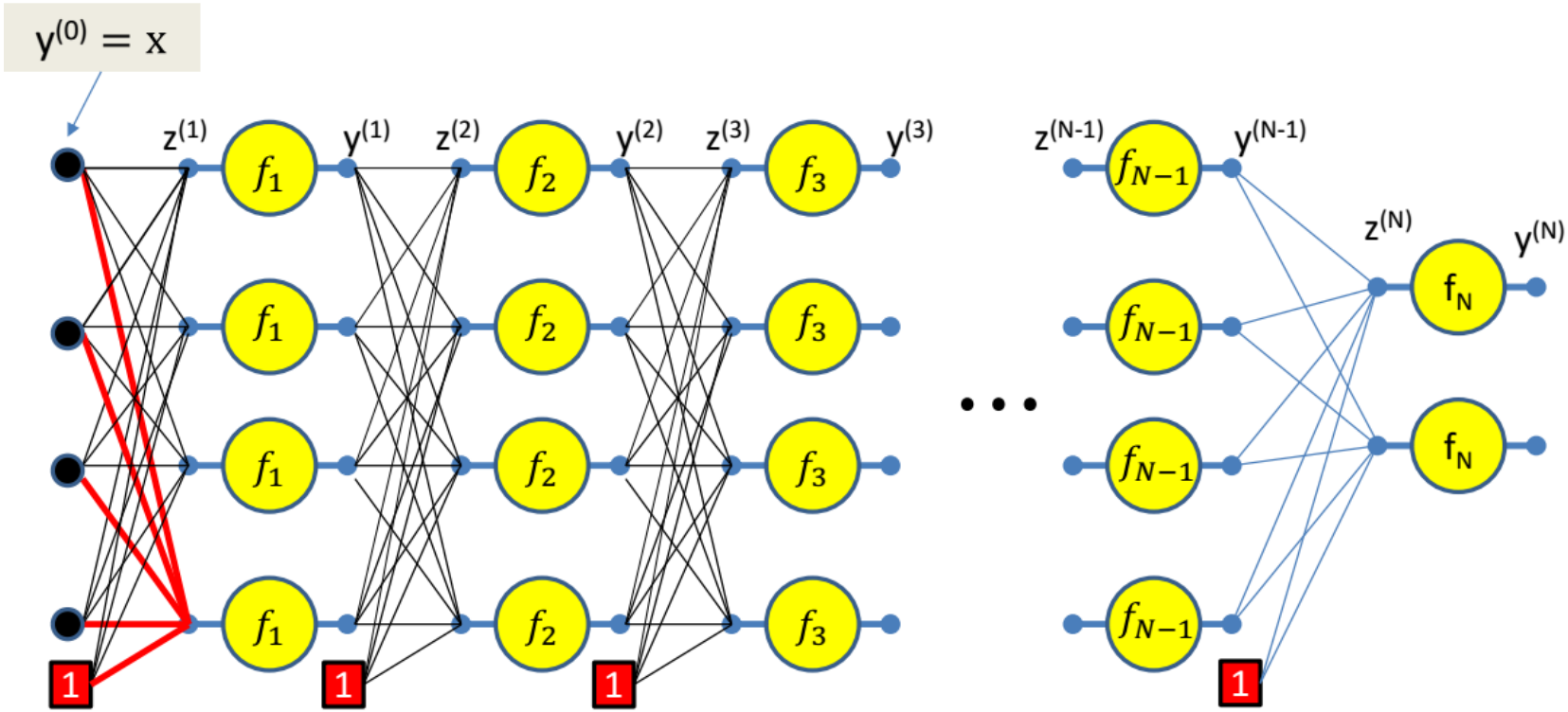
The "forward pass"



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

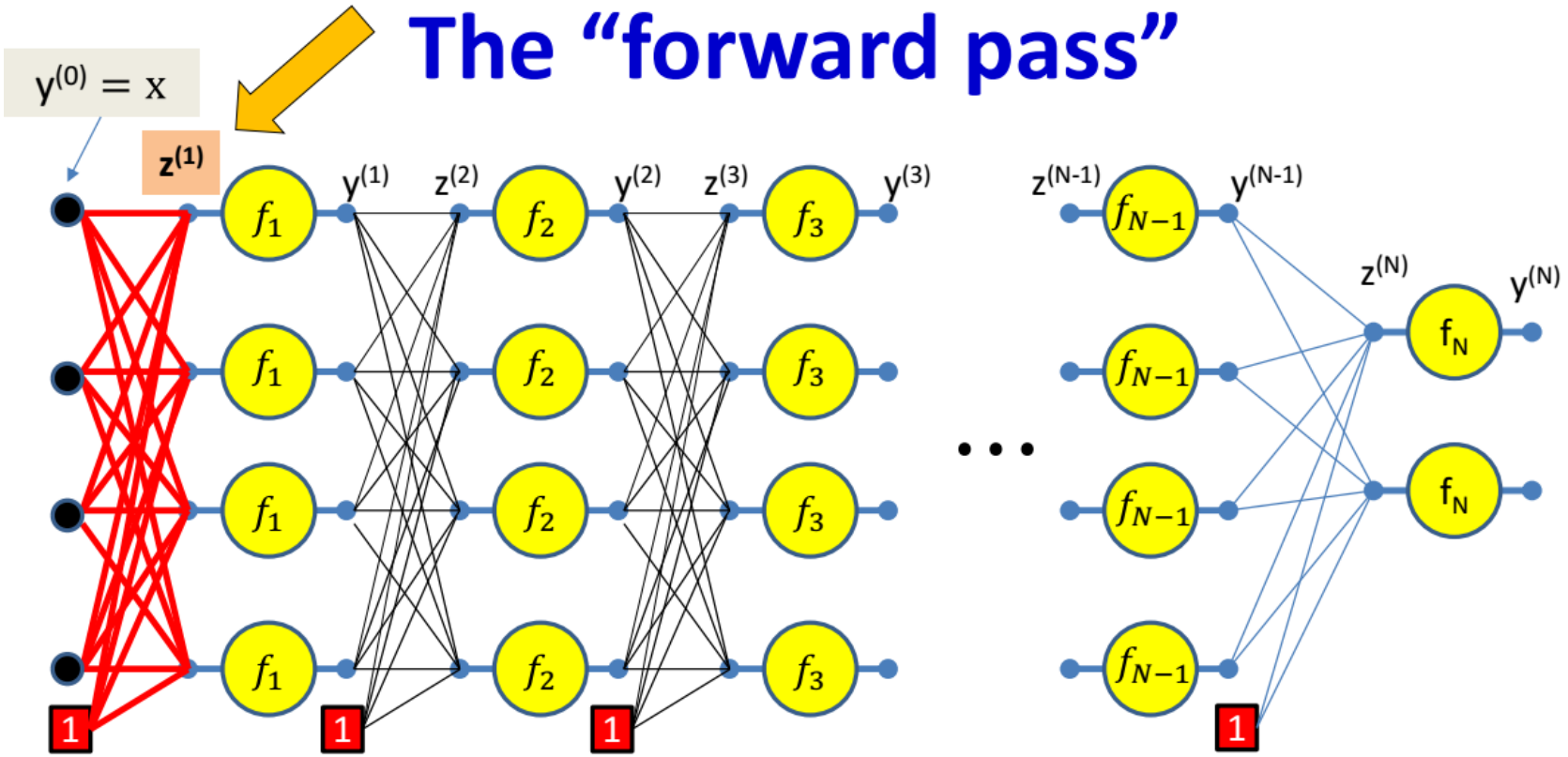
Assuming $w_{0j}^{(k)} = b_j^{(k)}$ and $y_0^{(k)} = 1$ -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases

The "forward pass"



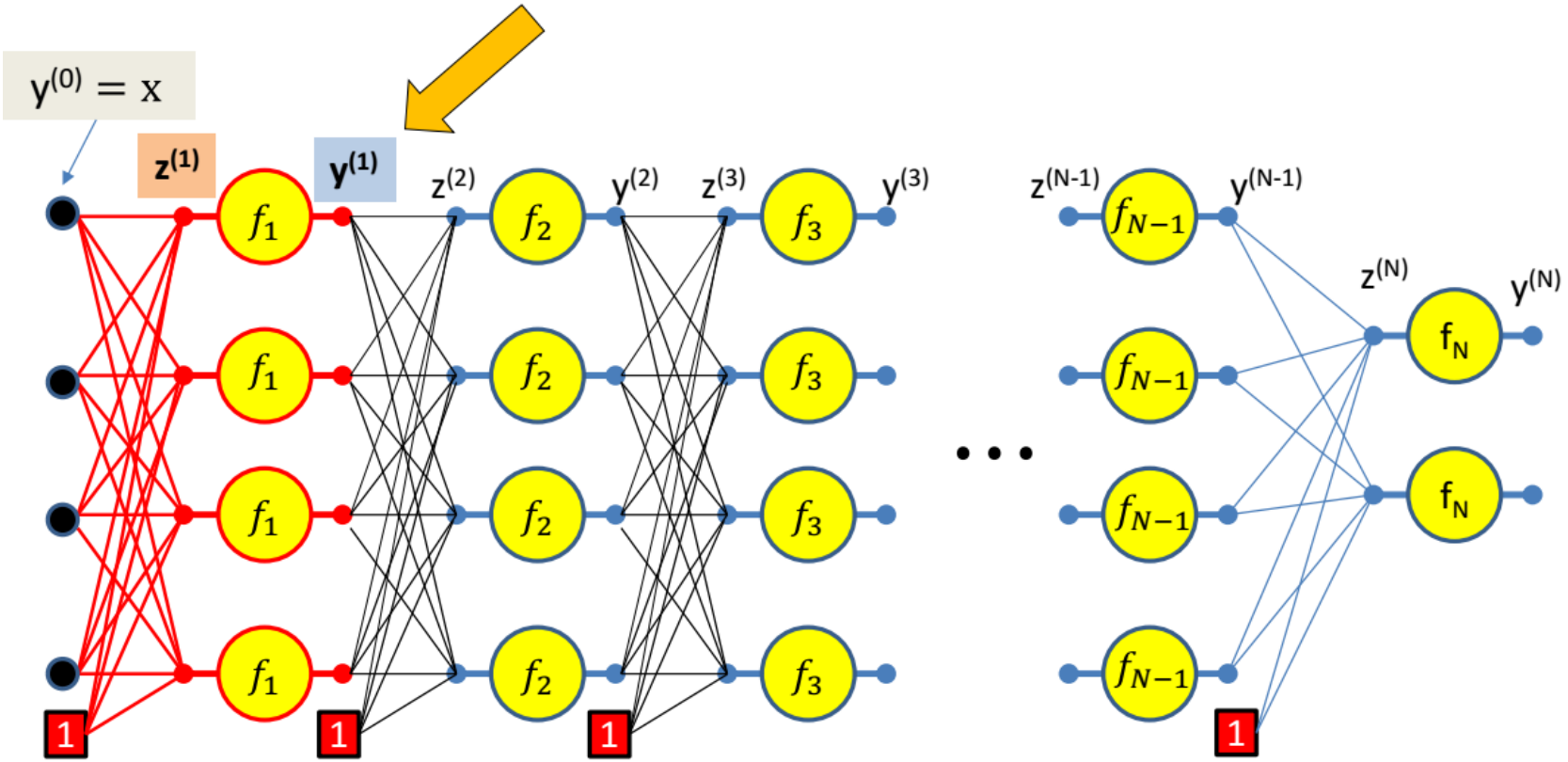
$$z_1^{(1)} = \sum_i w_{i1}^{(1)} y_i^{(0)}$$

The "forward pass"



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

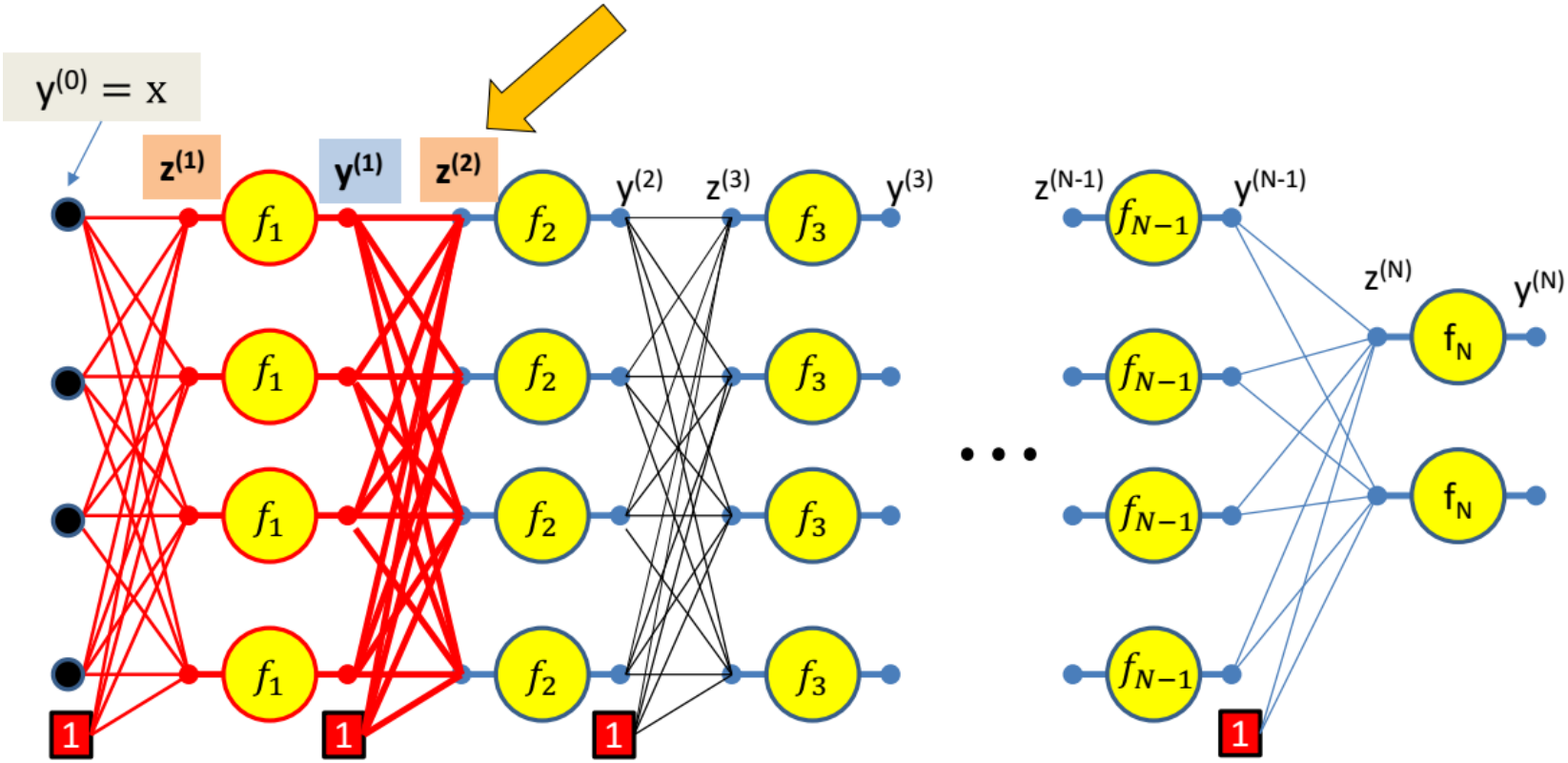
The "forward pass"



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

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

The "forward pass"

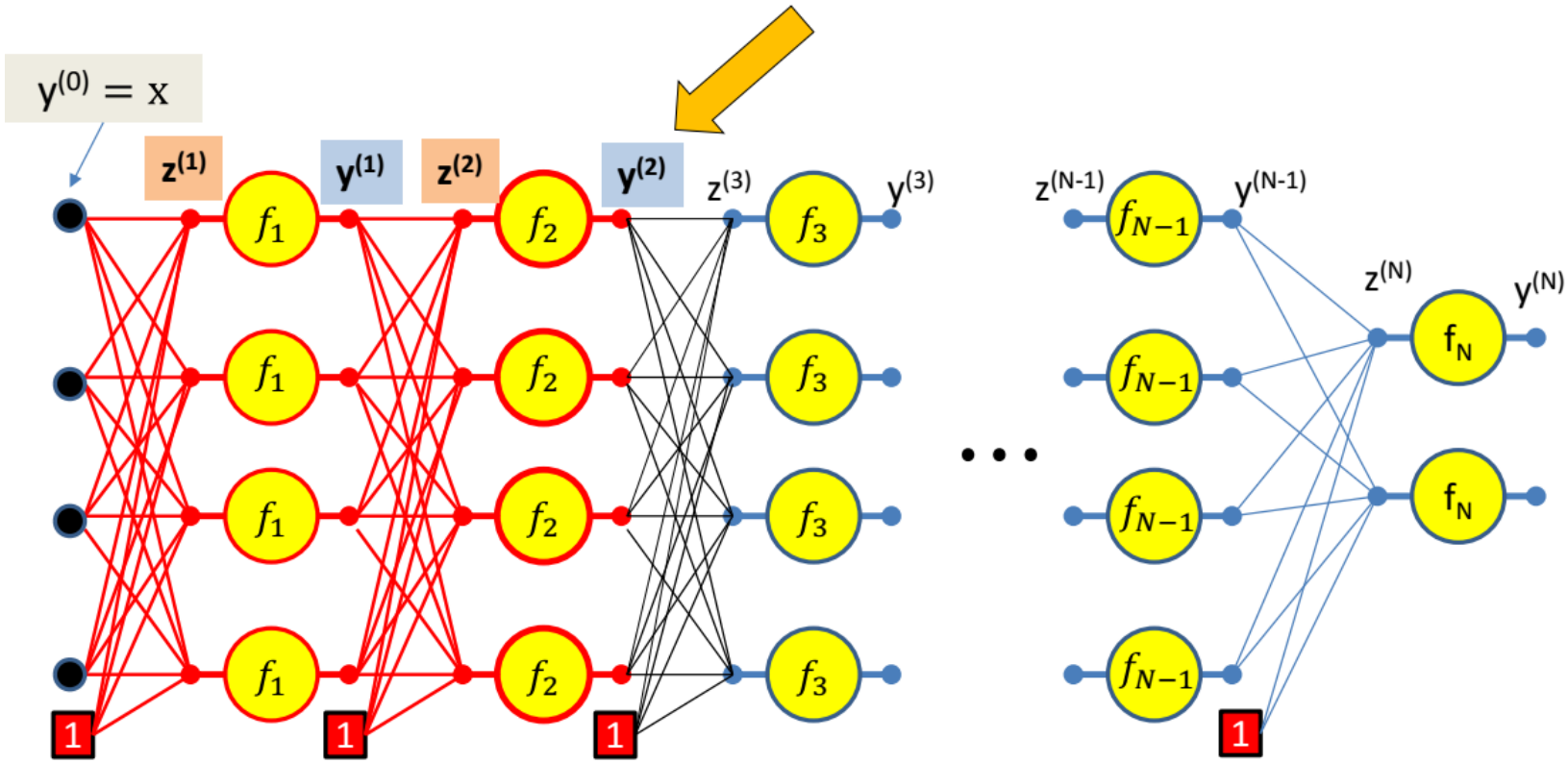


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

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

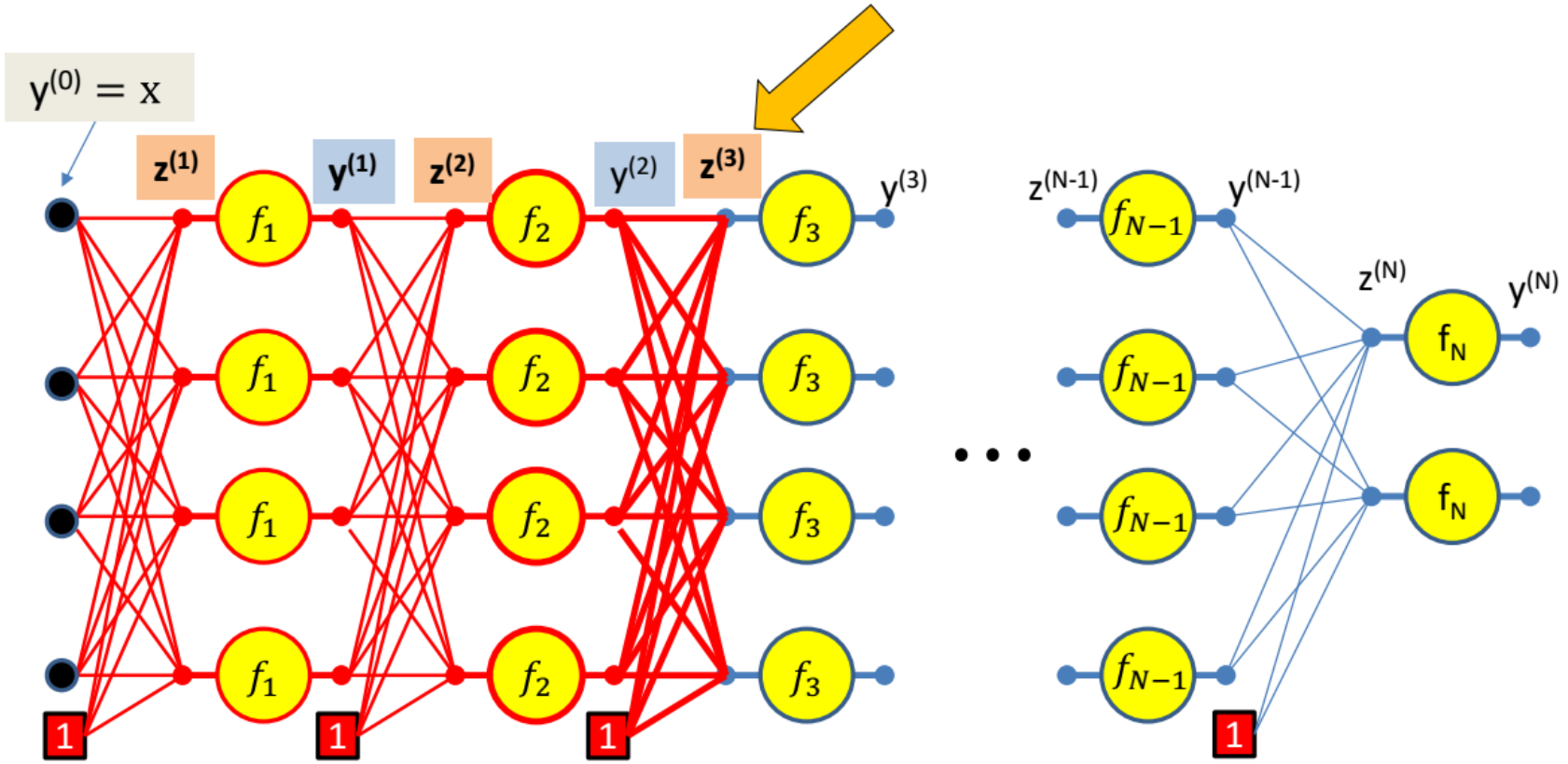
$$z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)}$$

The "forward pass"



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

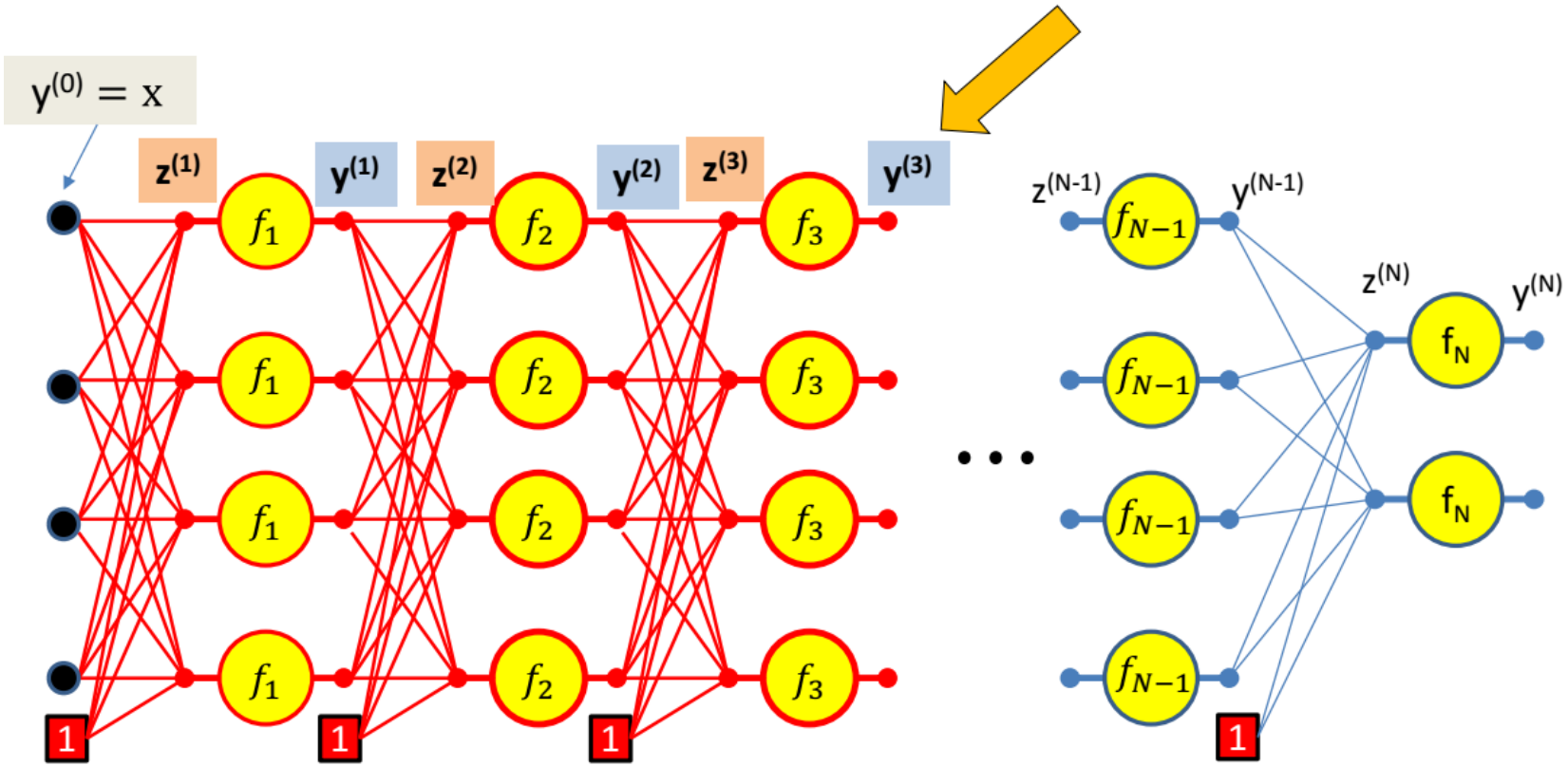
The "forward pass"



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

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

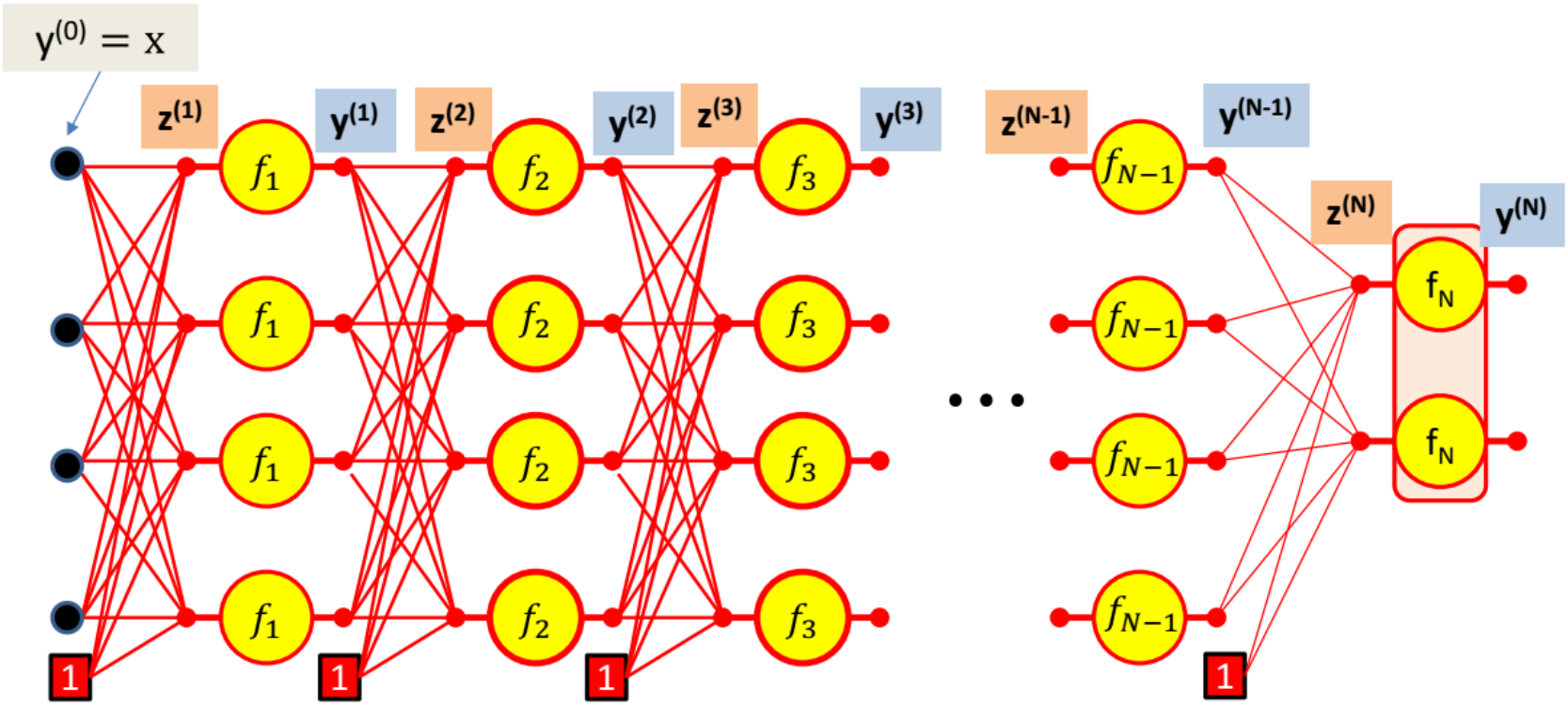
The "forward pass"



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

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

The "forward pass"

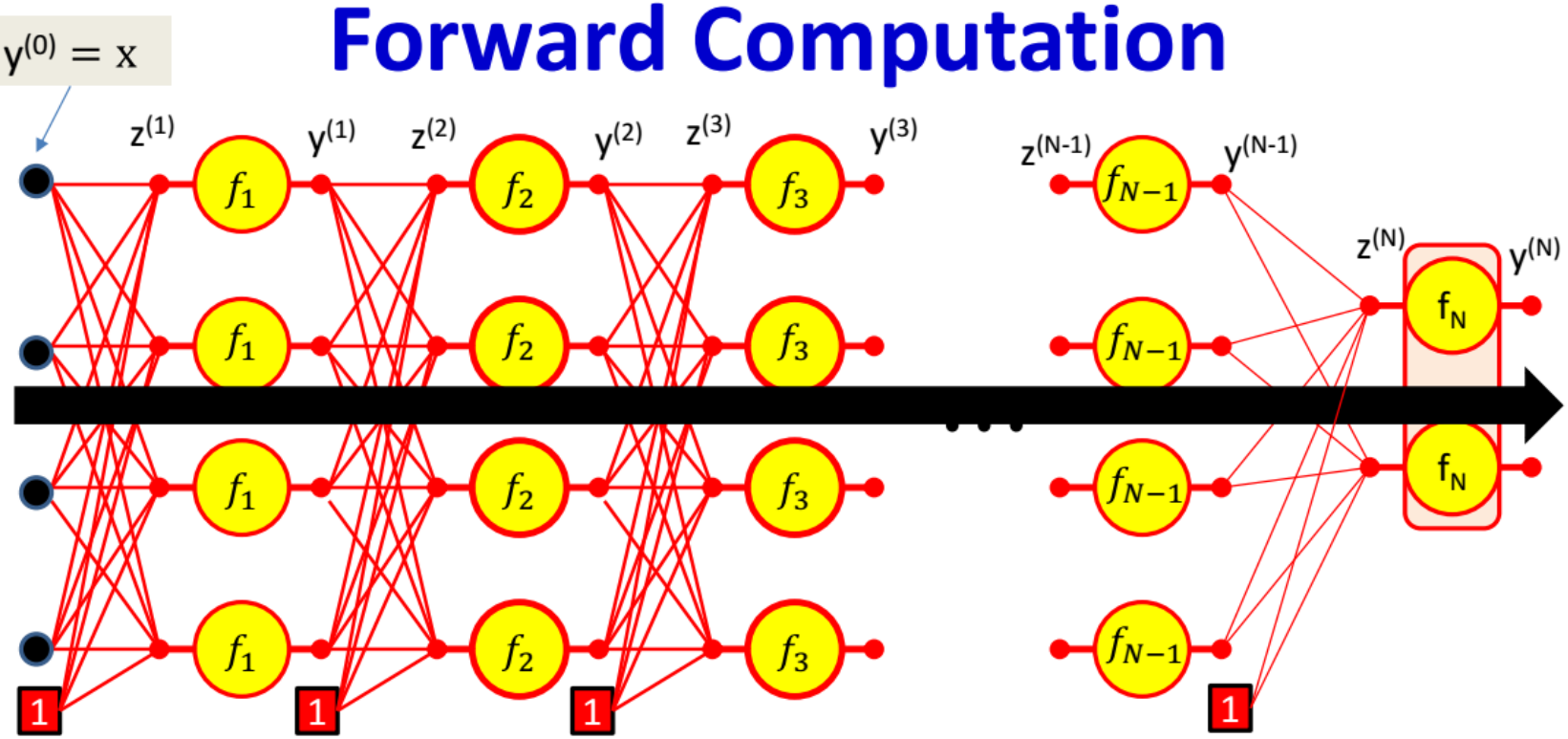


$$y_j^{(N-1)} = f_{N-1}\left(z_j^{(N-1)}\right) \qquad z_j^{(N)} = \sum_i w_{ij}^{(N)} y_i^{(N-1)}$$

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

The "forward pass"

Forward Computation



ITERATE FOR k = 1:N

for j = 1:layer-width

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

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

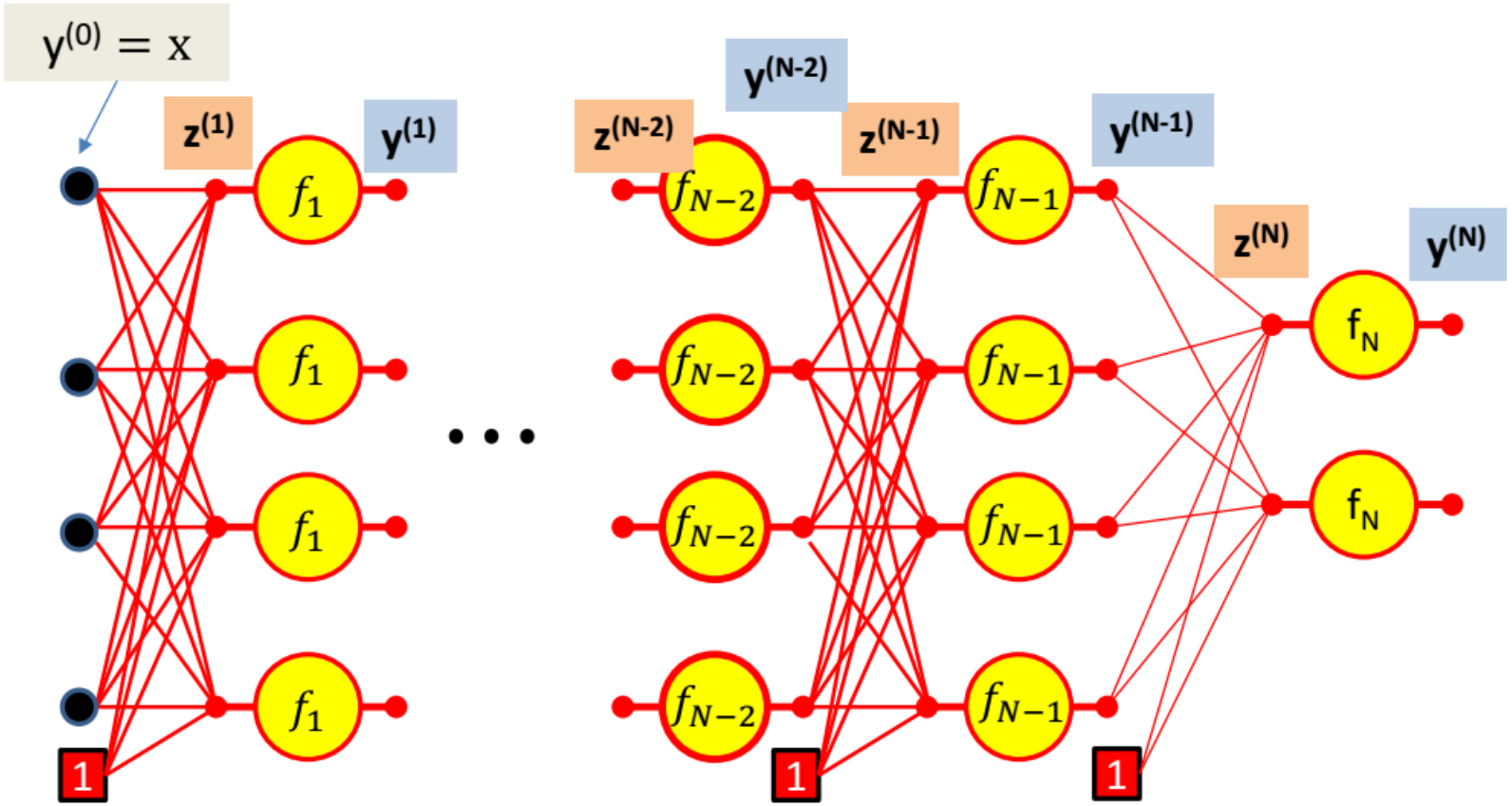
$$y_j^{(k)} = f_k\left(z_j^{(k)}\right)$$

The "forward pass"

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

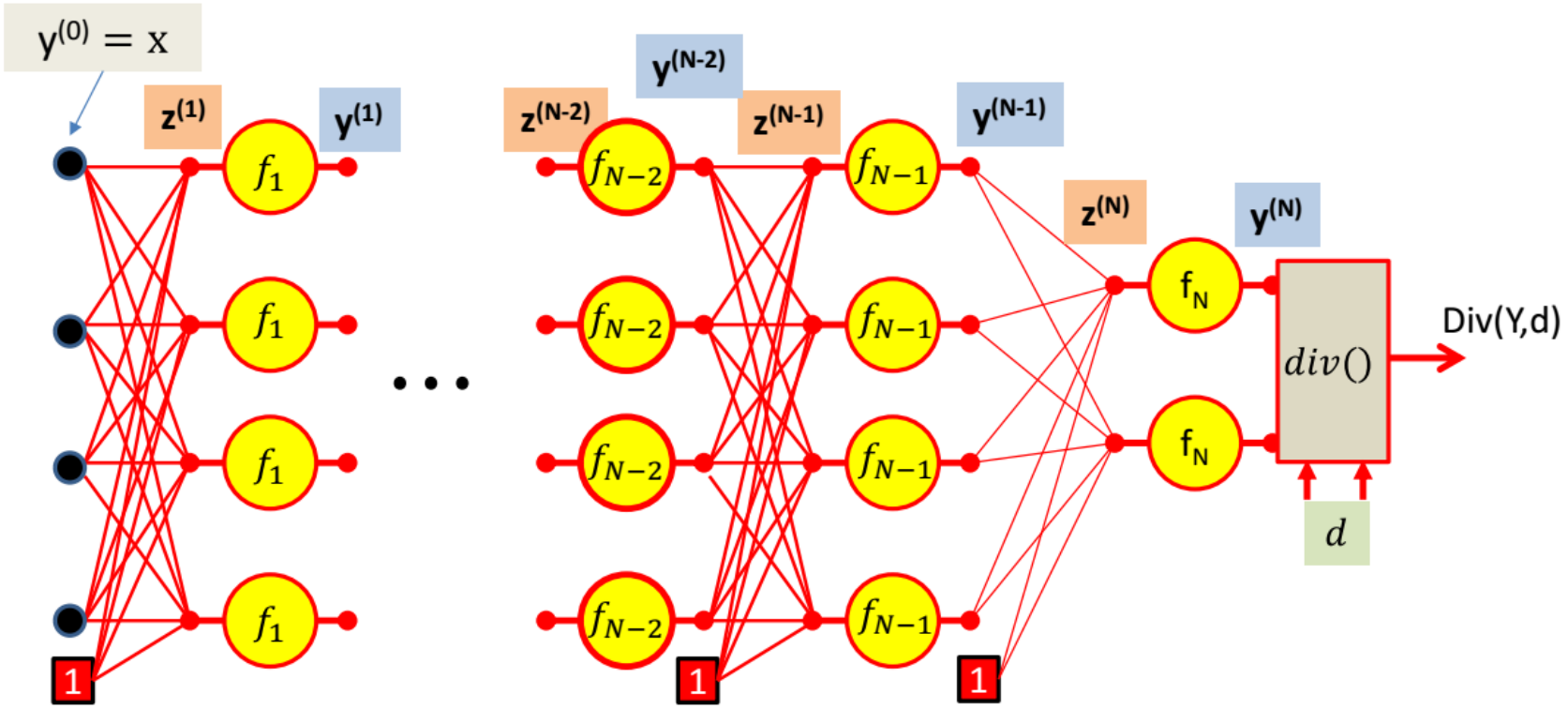
Computing derivatives



We have computed all these intermediate values in the forward computation

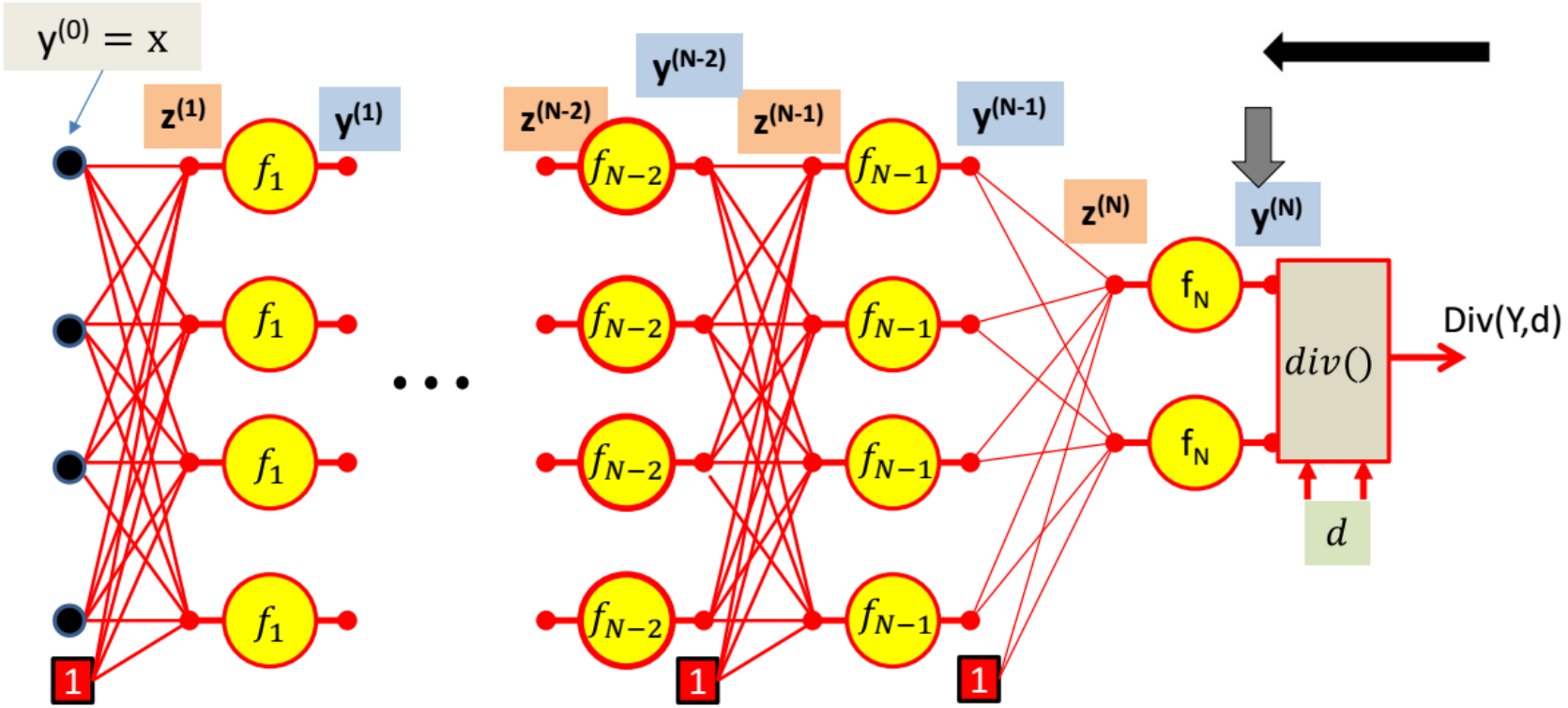
We must remember them – we will need them to compute the derivatives

Computing derivatives



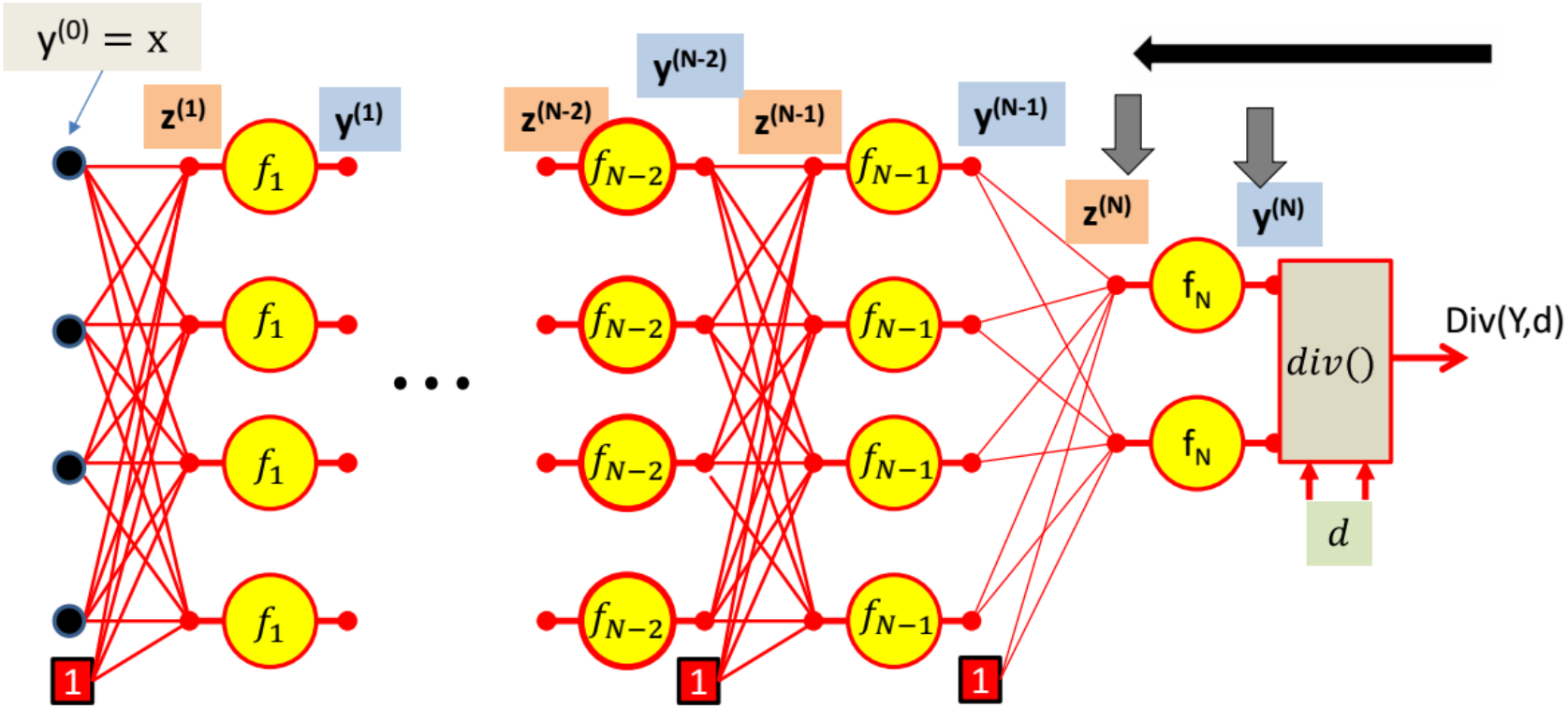
First, we compute the divergence between the output of the net $y = y^{(N)}$ and the desired output d

Computing derivatives



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

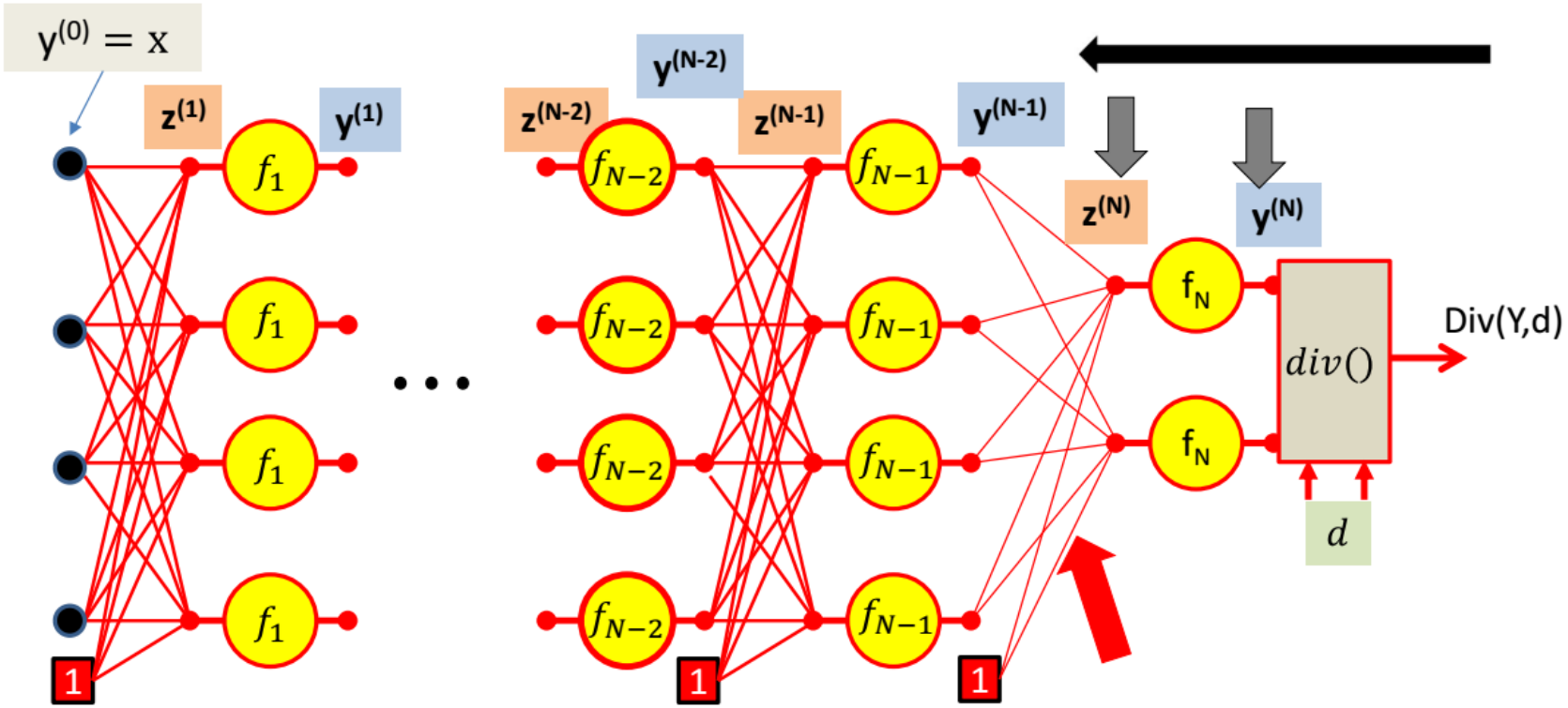
Computing derivatives



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

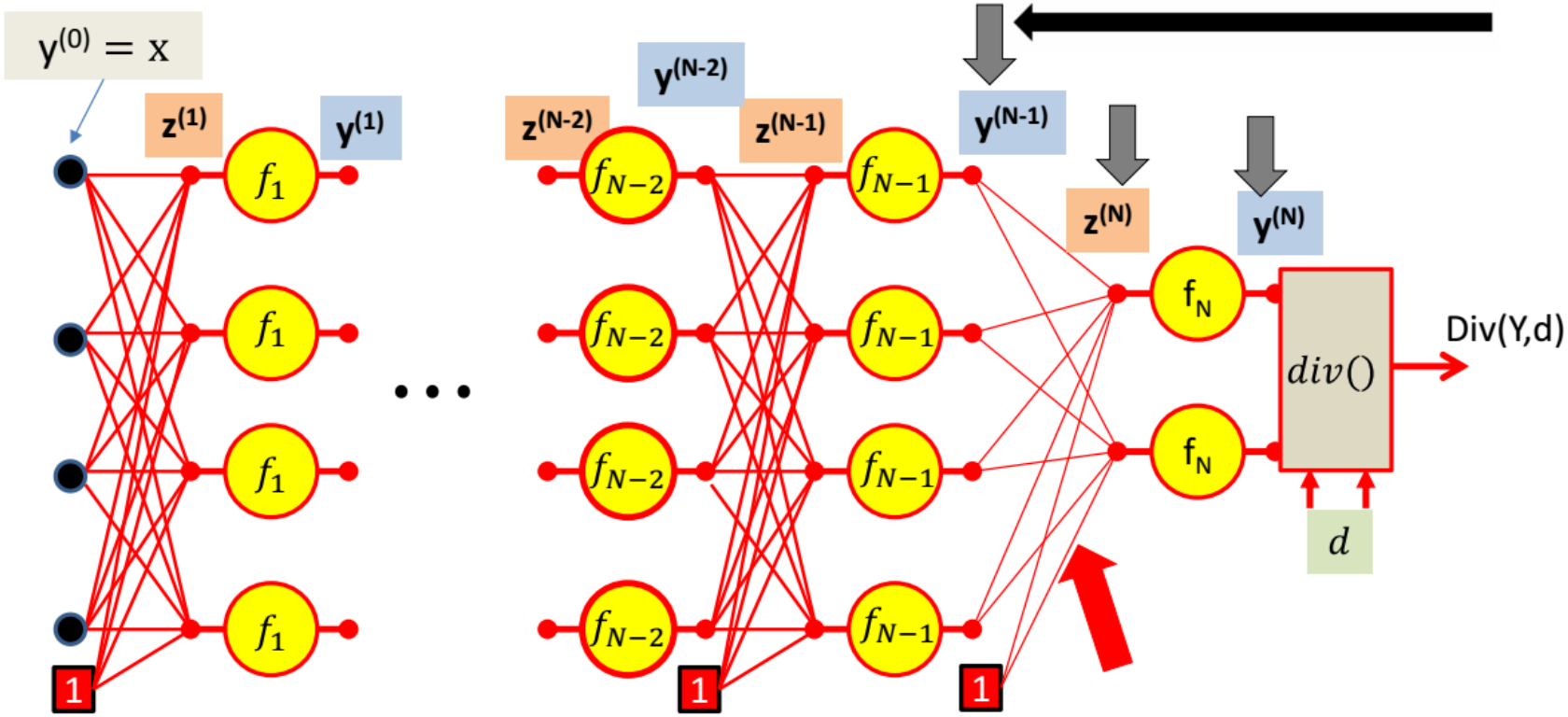
We then compute $\nabla_{z^{(N)}} div(.)$ the derivative of the divergence w.r.t. the *pre-activation* affine combination $z^{(N)}$ using the chain rule

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

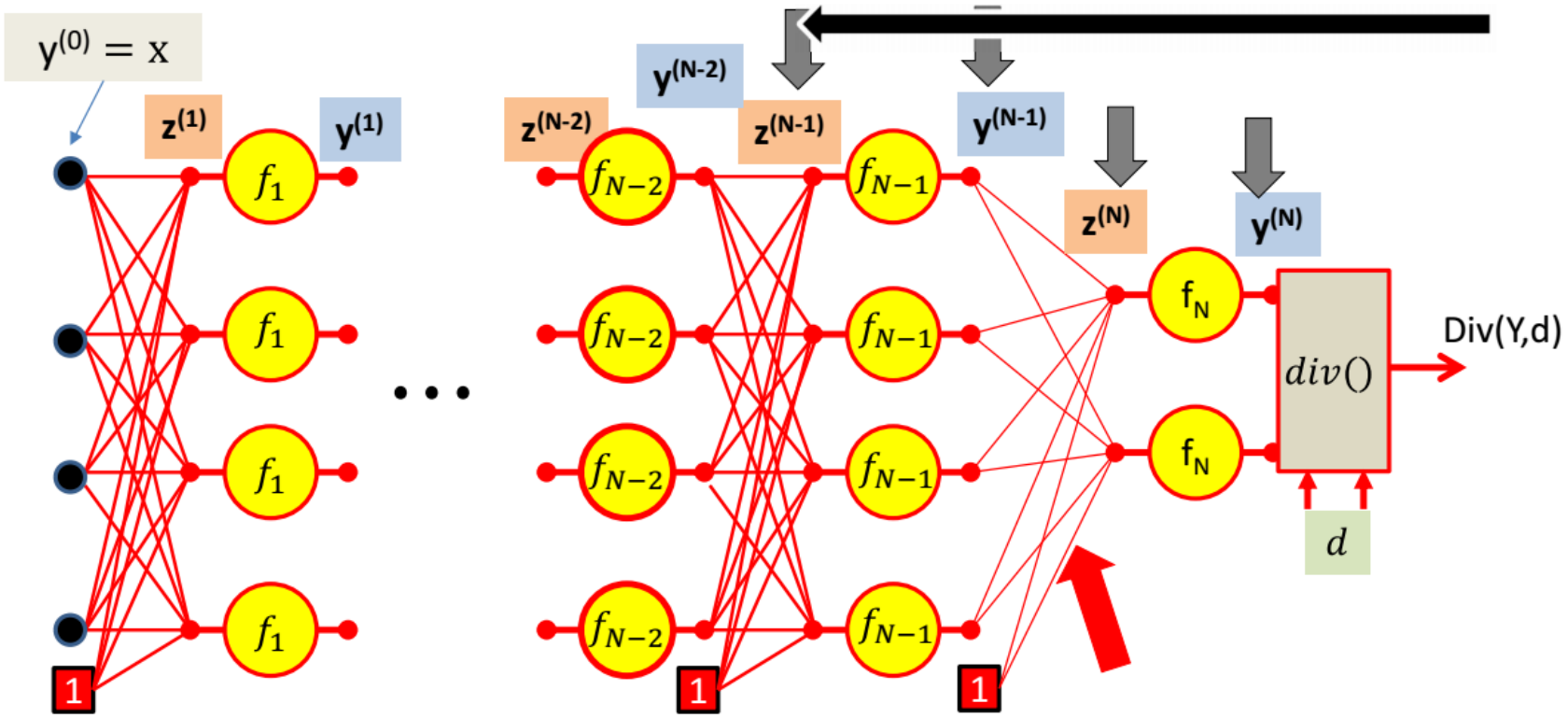
Computing derivatives



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

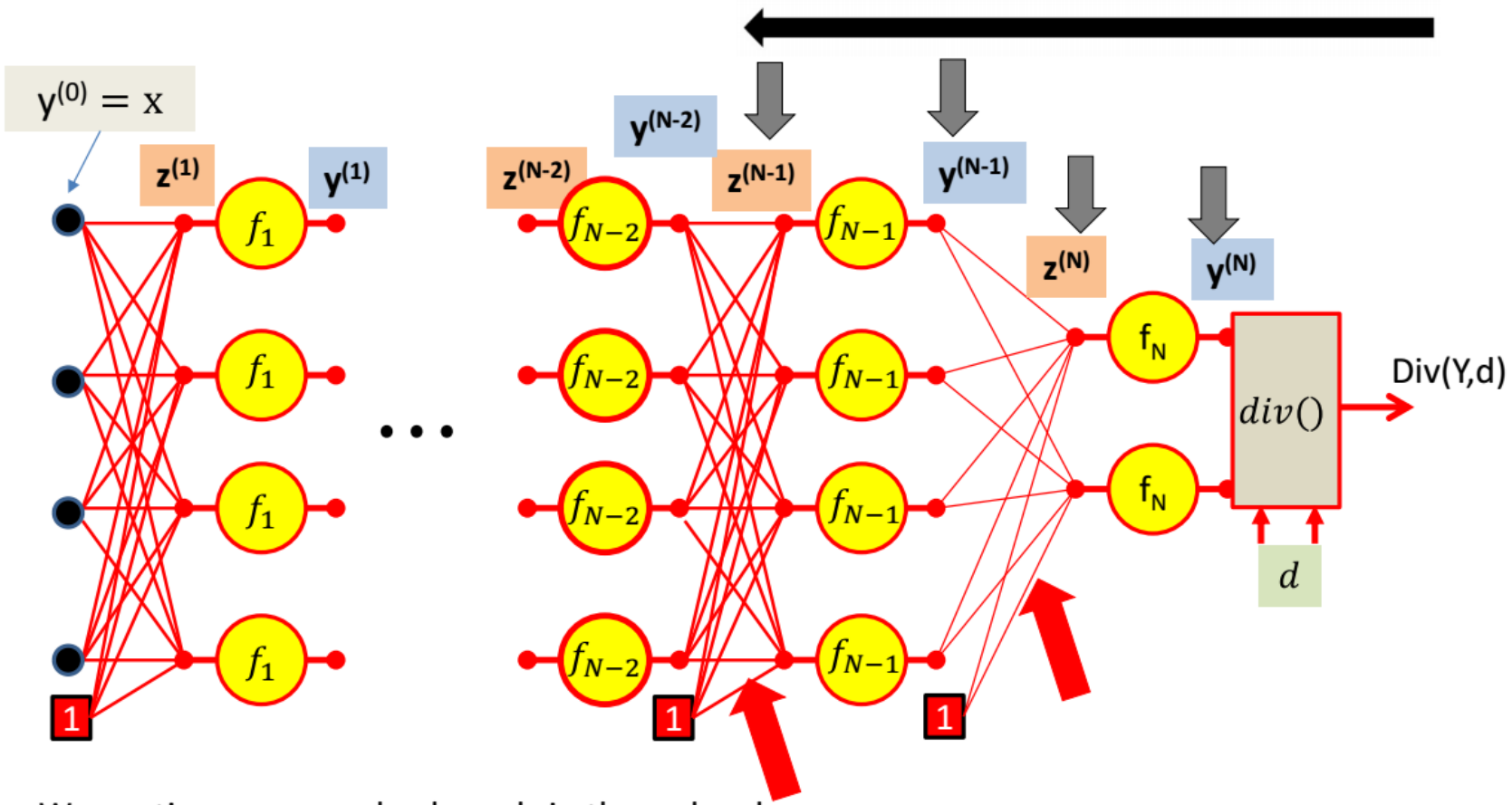
Computing derivatives



We continue our way backwards in the order shown

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

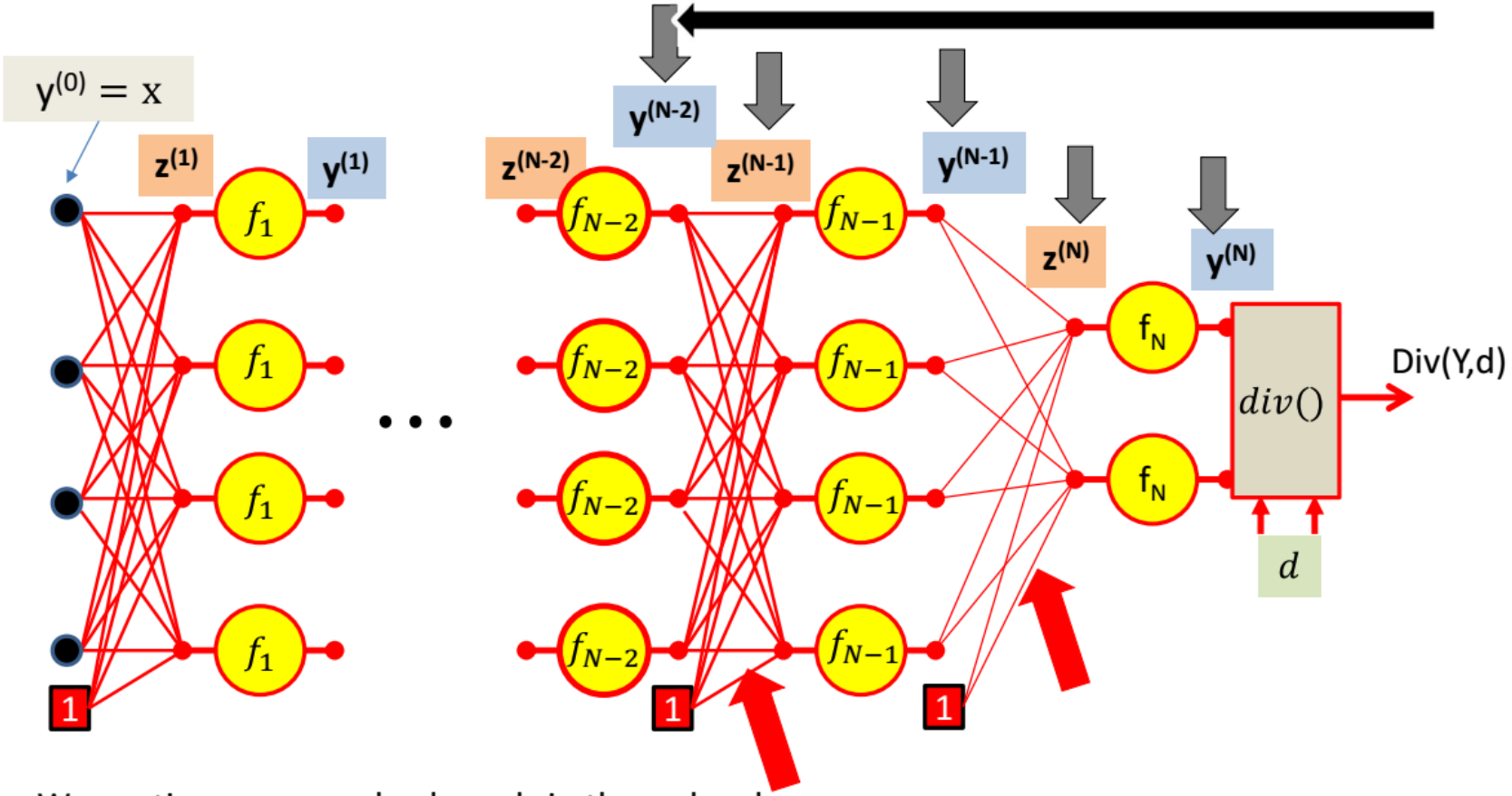
Computing derivatives



We continue our way backwards in the order shown

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

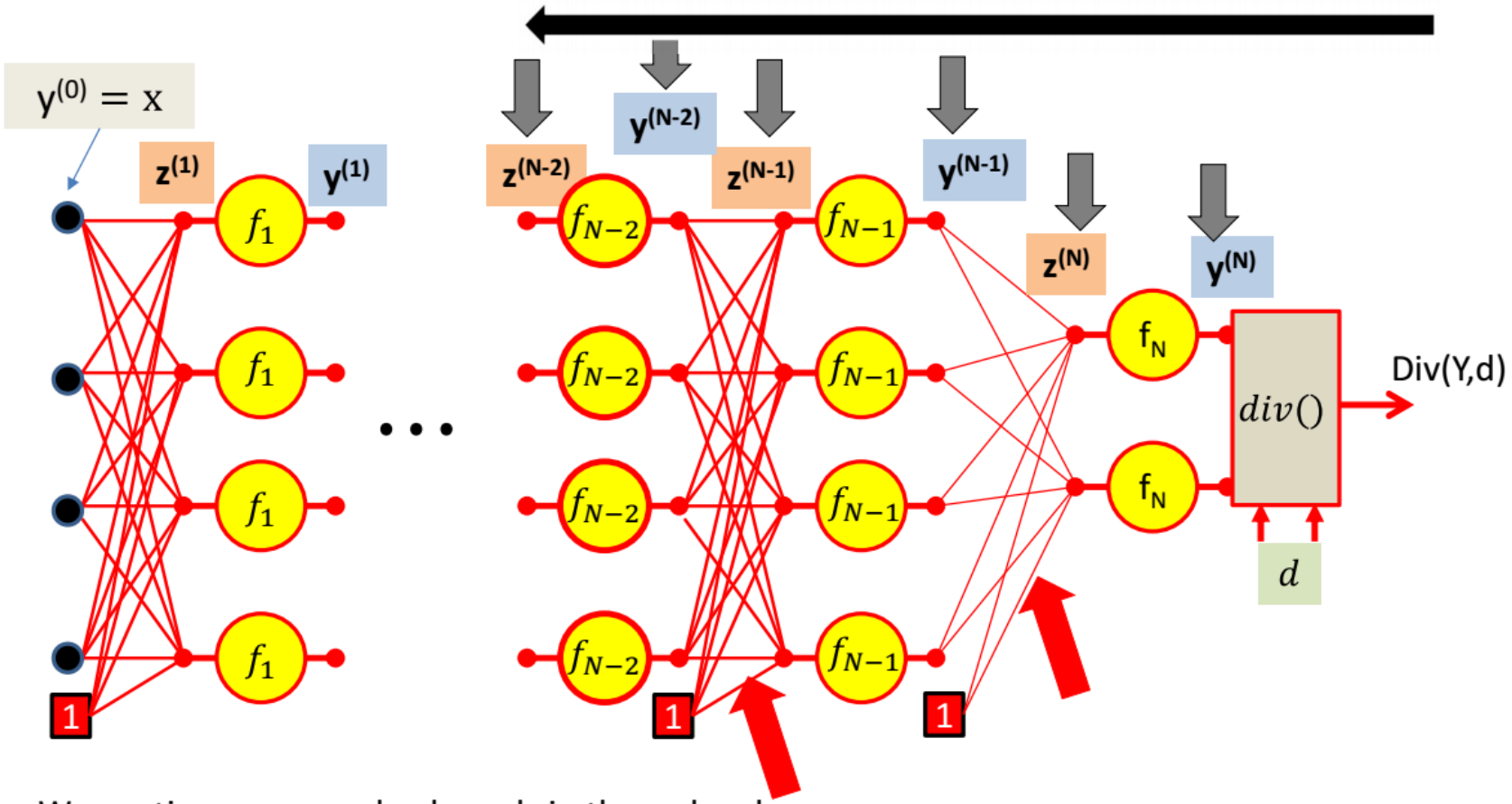
Computing derivatives



We continue our way backwards in the order shown

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

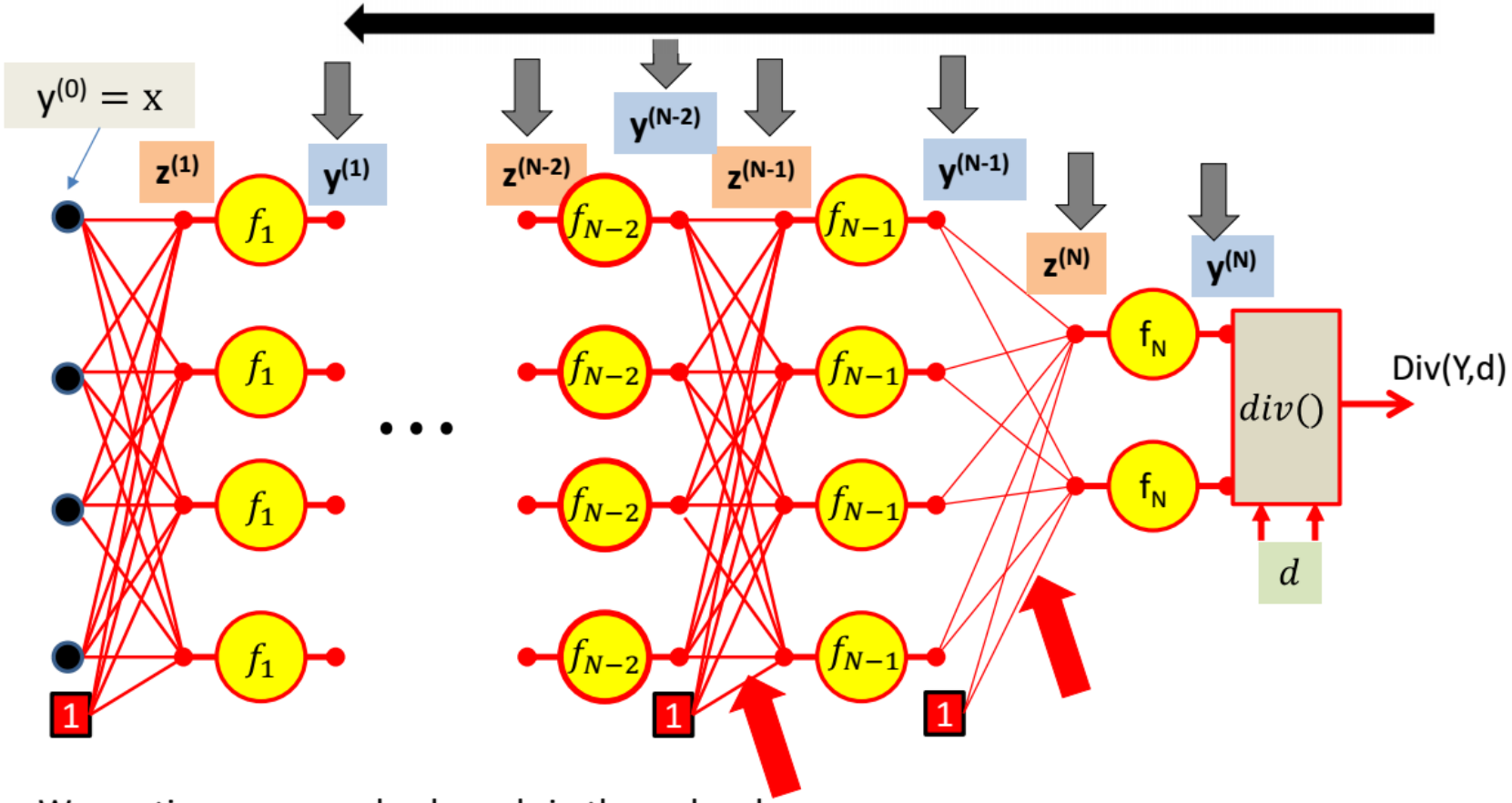
Computing derivatives



We continue our way backwards in the order shown

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

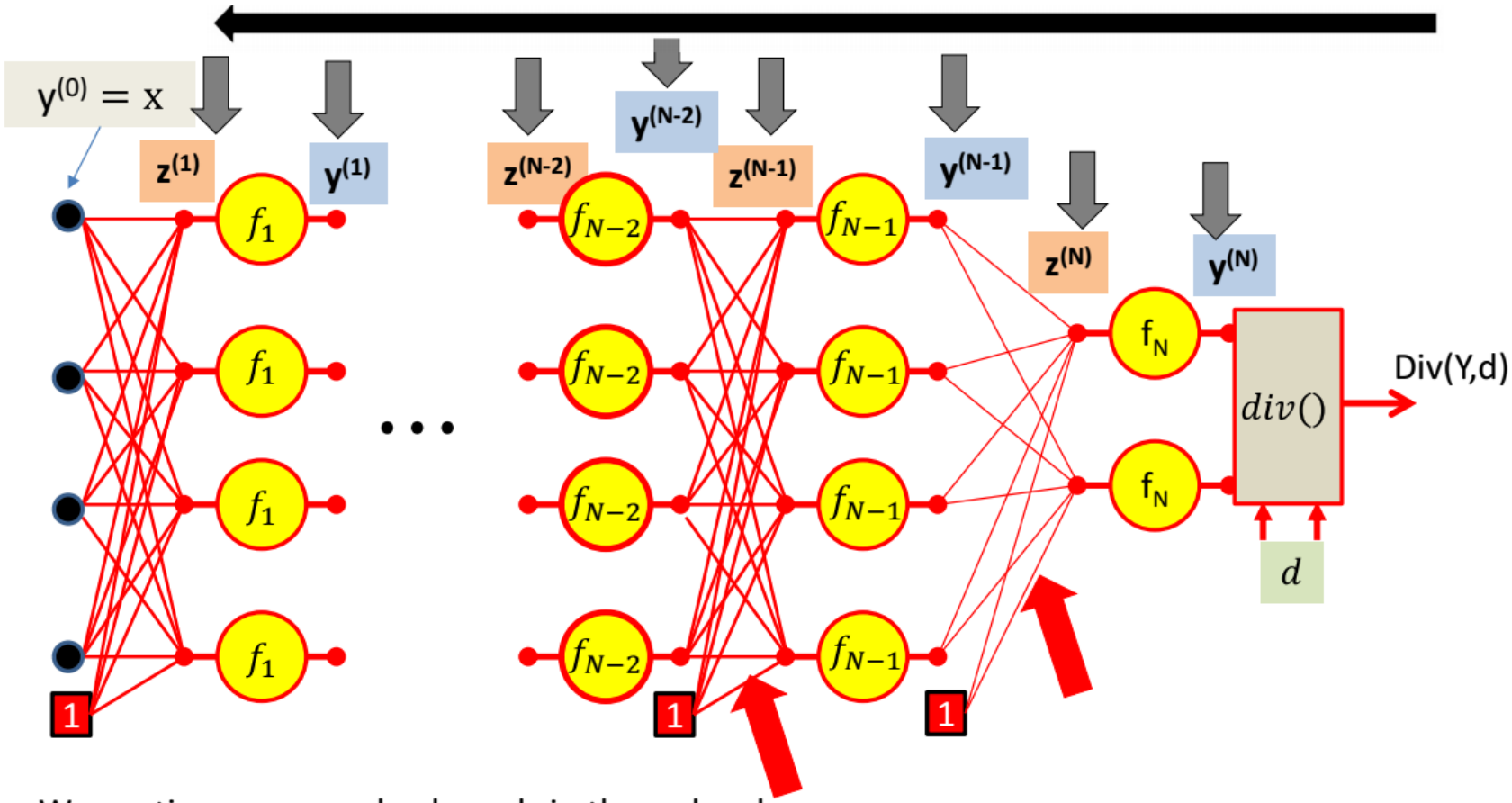
Computing derivatives



We continue our way backwards in the order shown

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

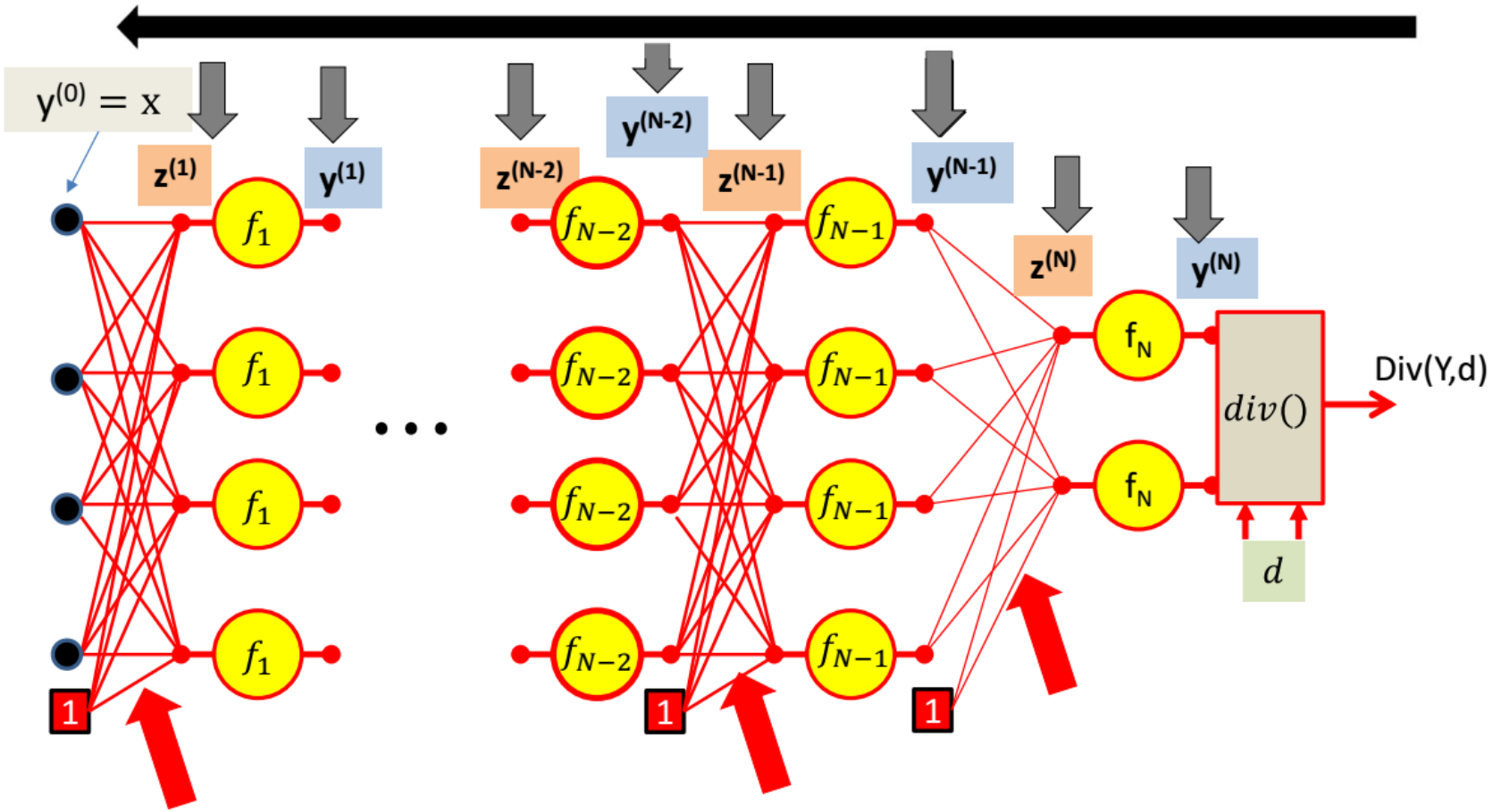
Computing derivatives



We continue our way backwards in the order shown

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

Computing derivatives



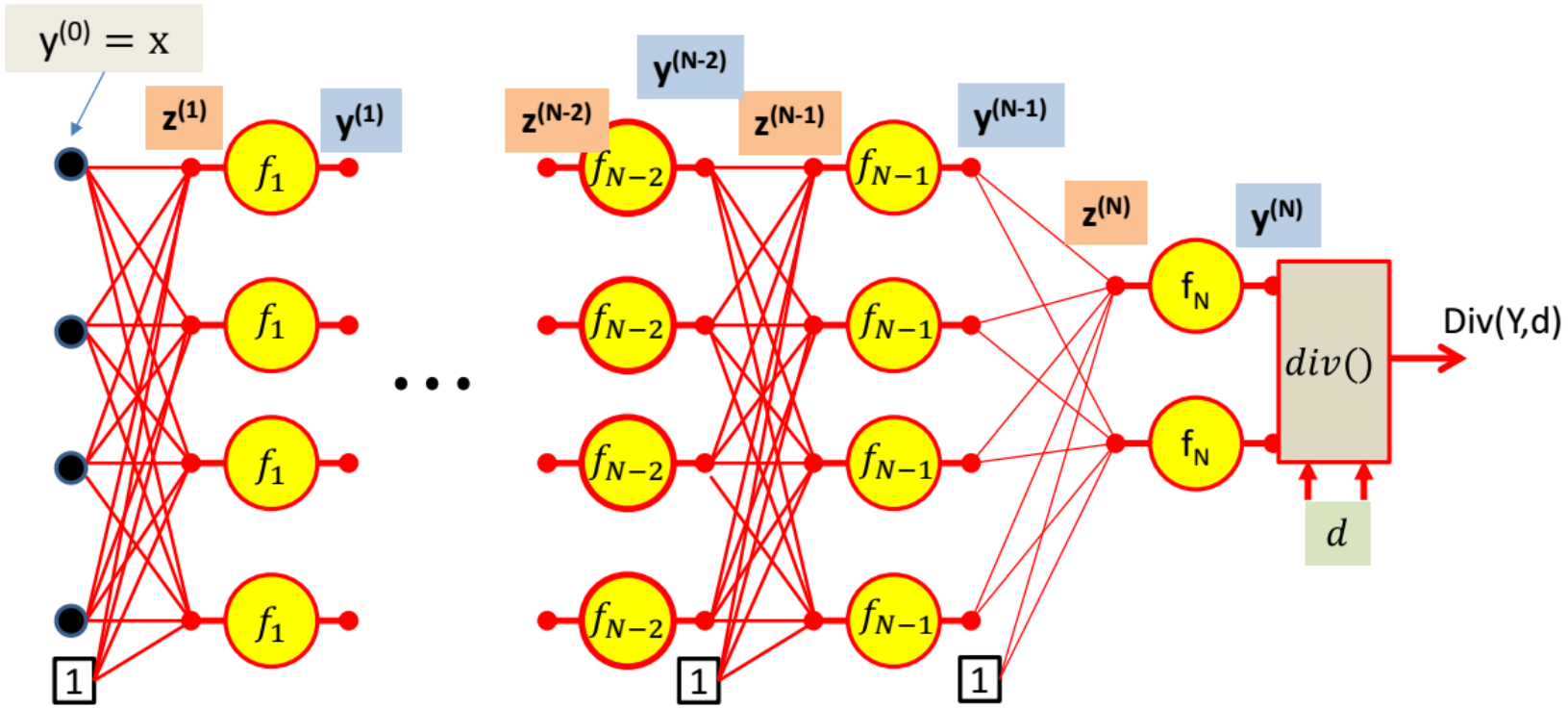
We continue our way backwards in the order shown

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

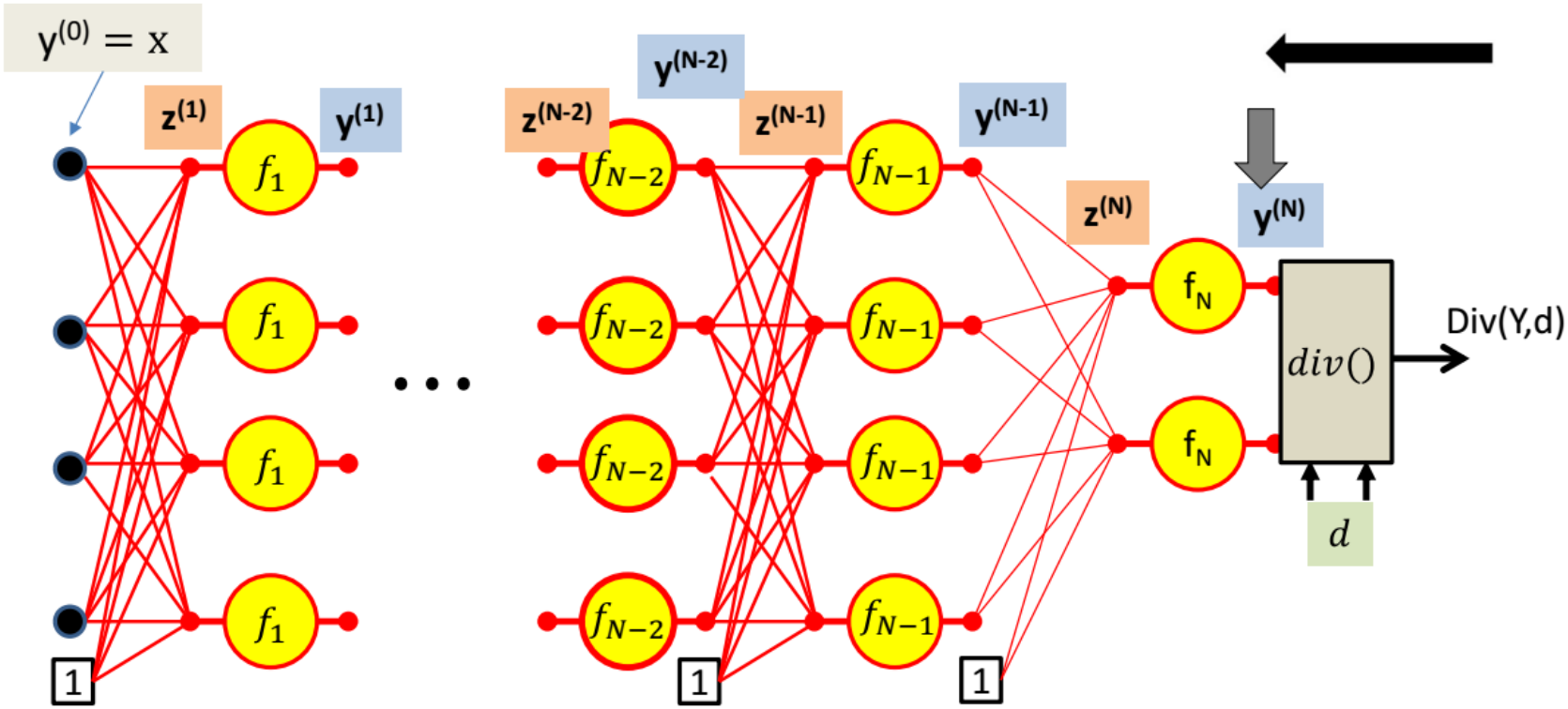
Backward Gradient Computation

- Let's actually see the math..

Computing derivatives



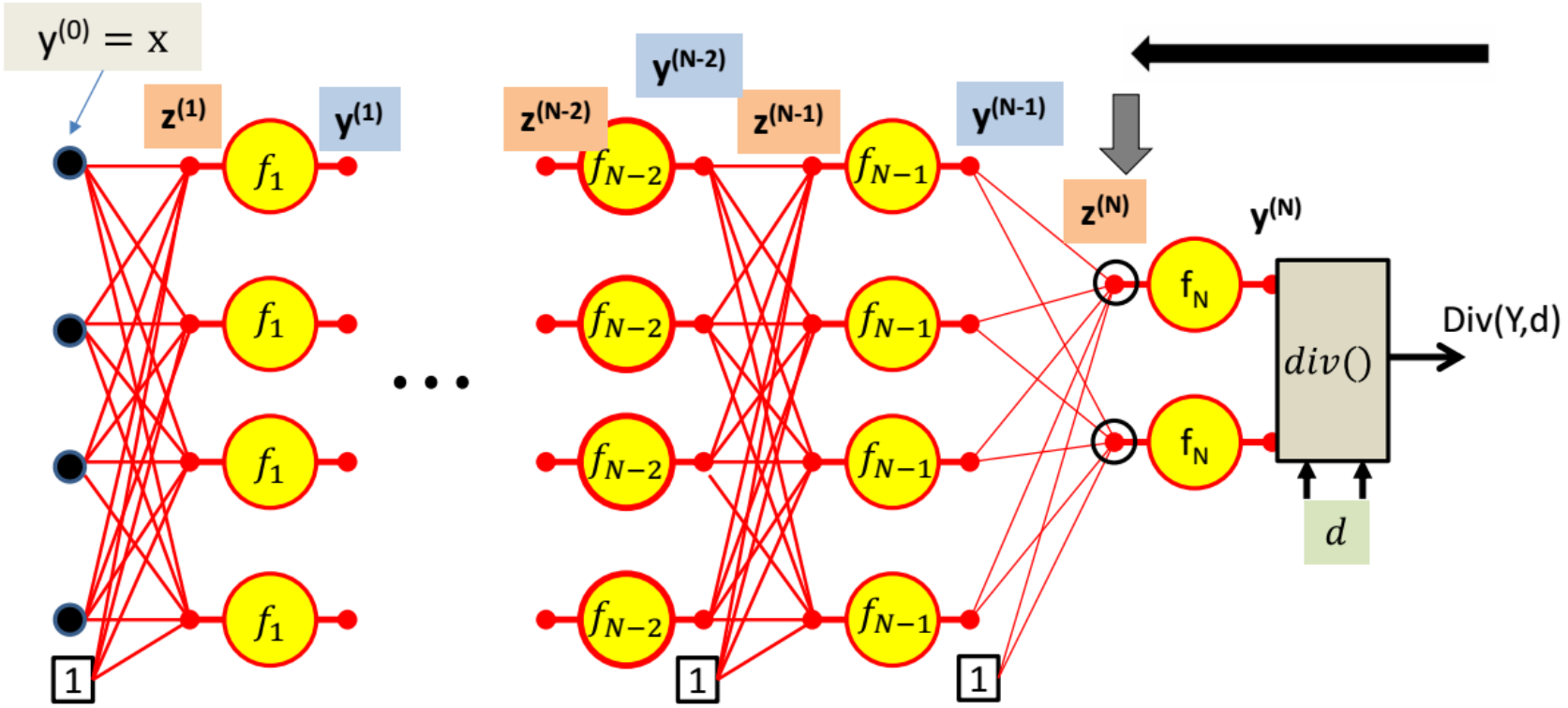
Computing derivatives



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

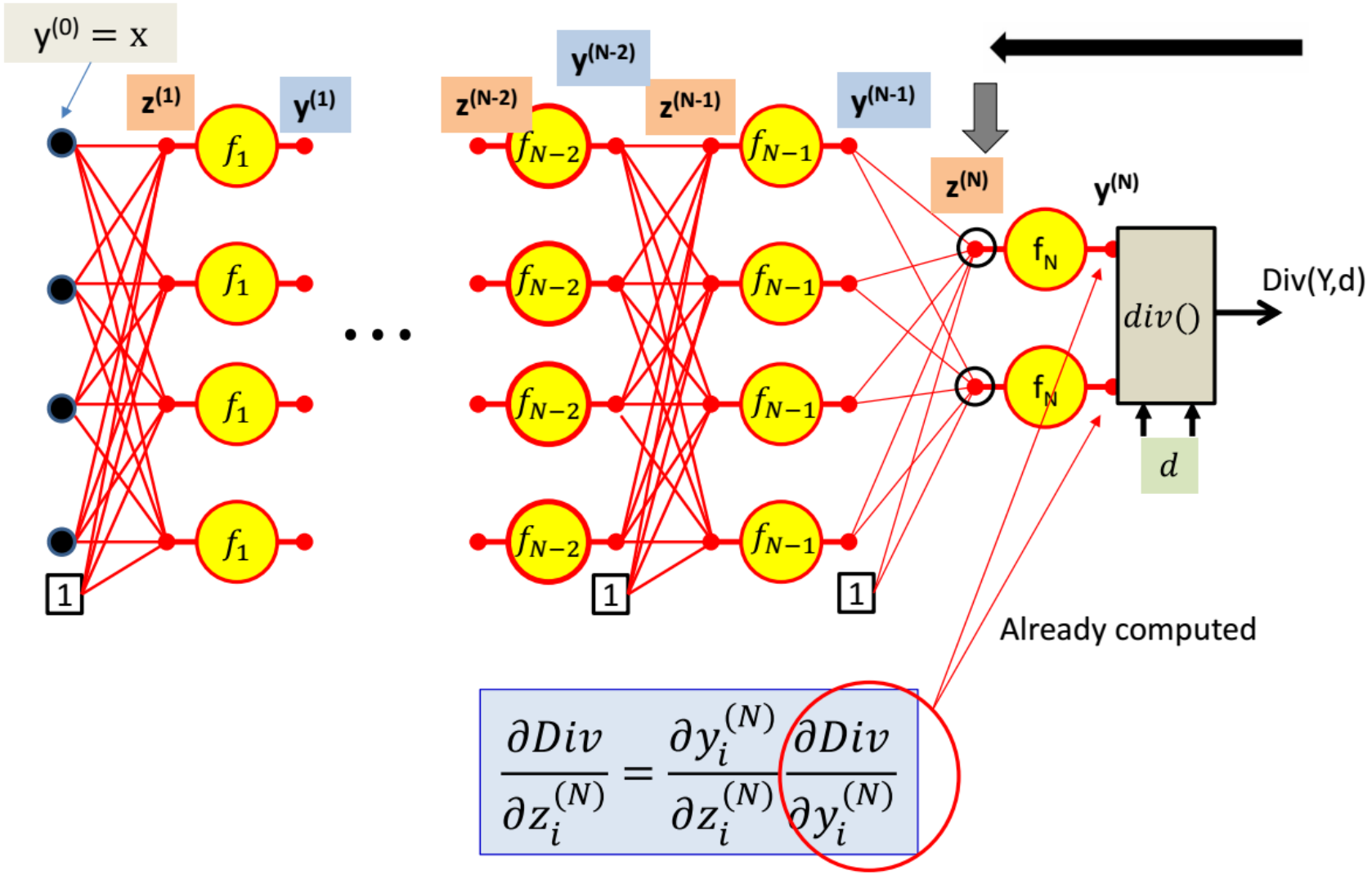
$$\frac{\partial Div(Y,d)}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$$

Computing derivatives

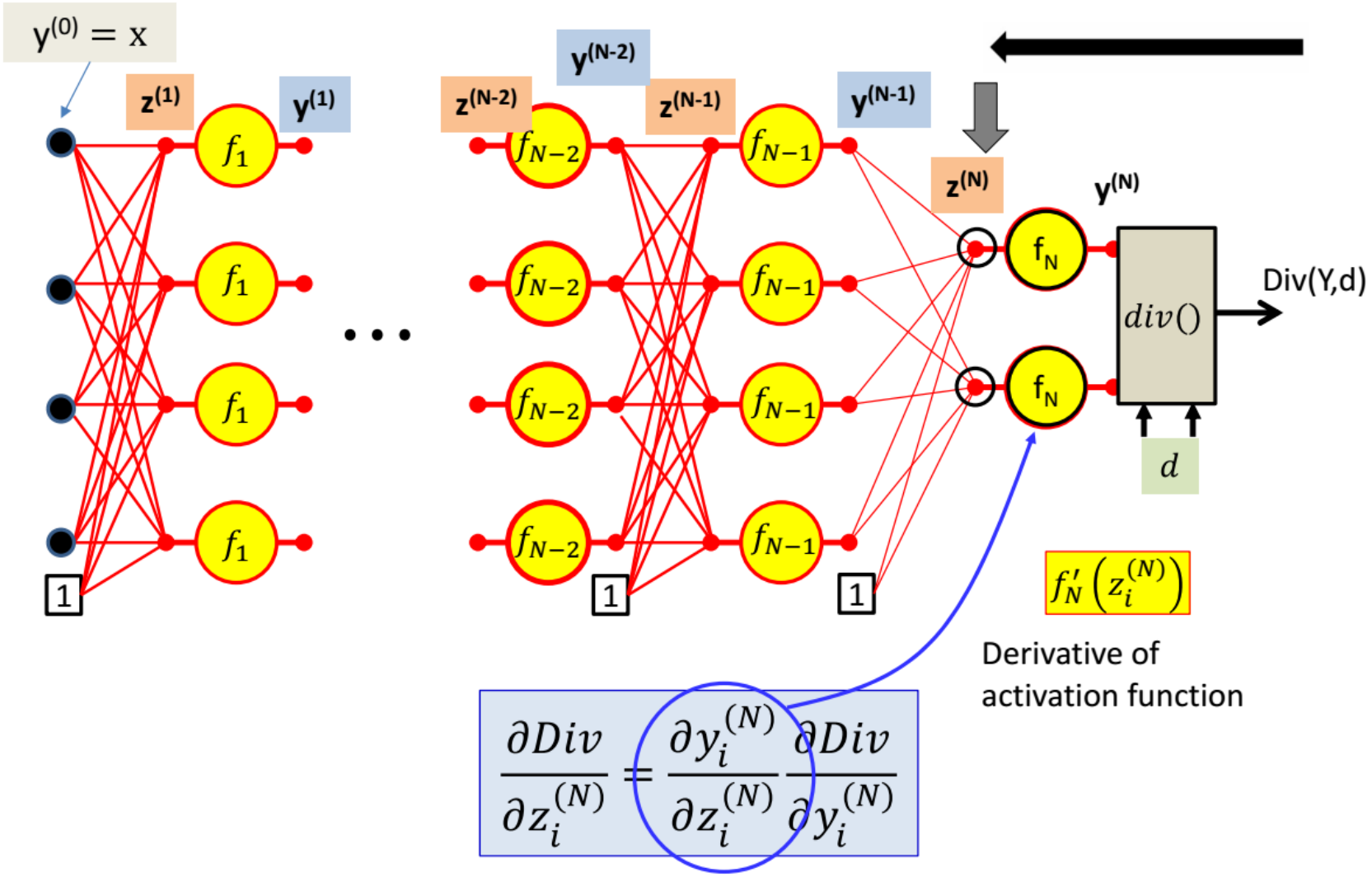


$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

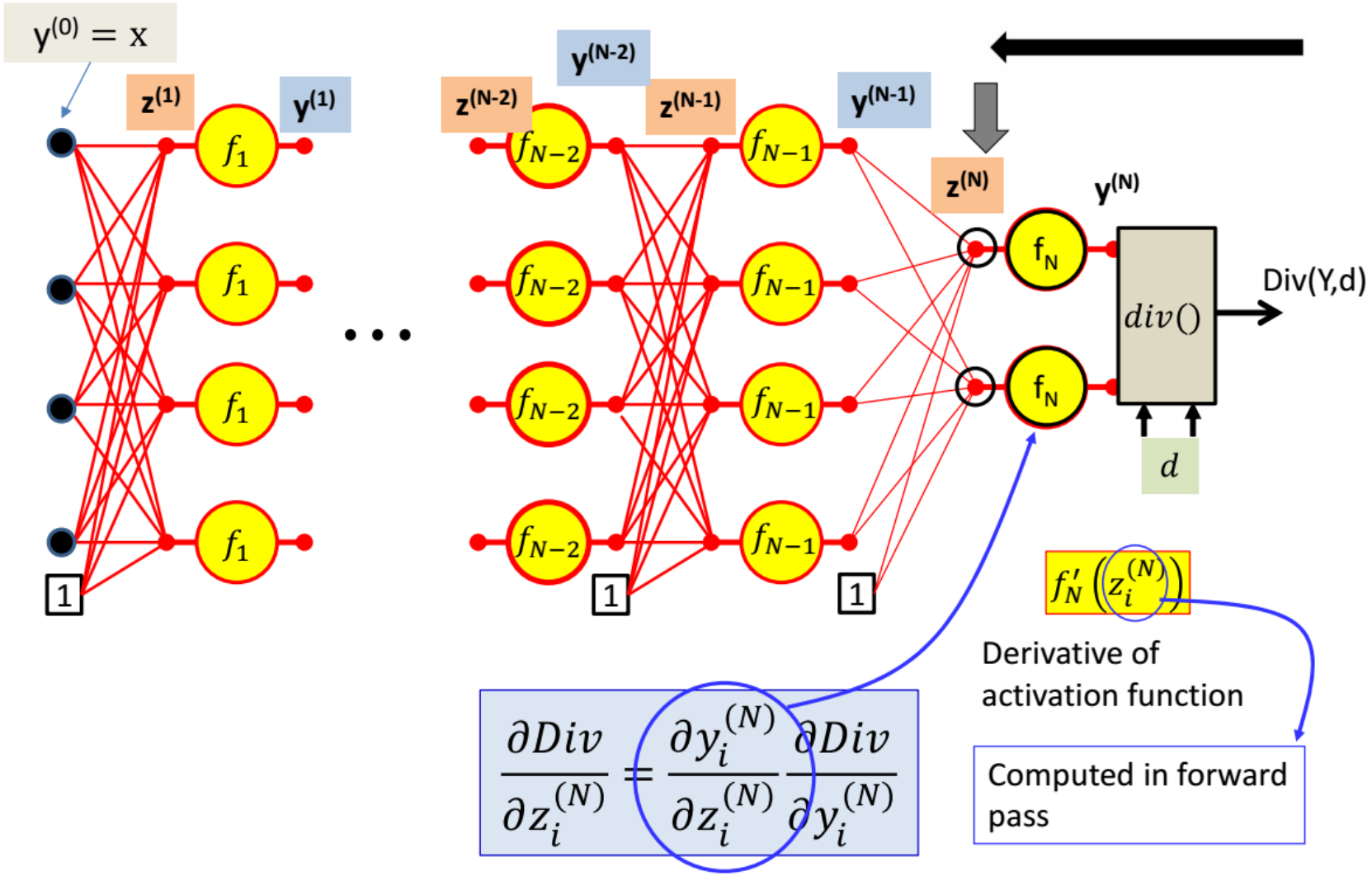
Computing derivatives



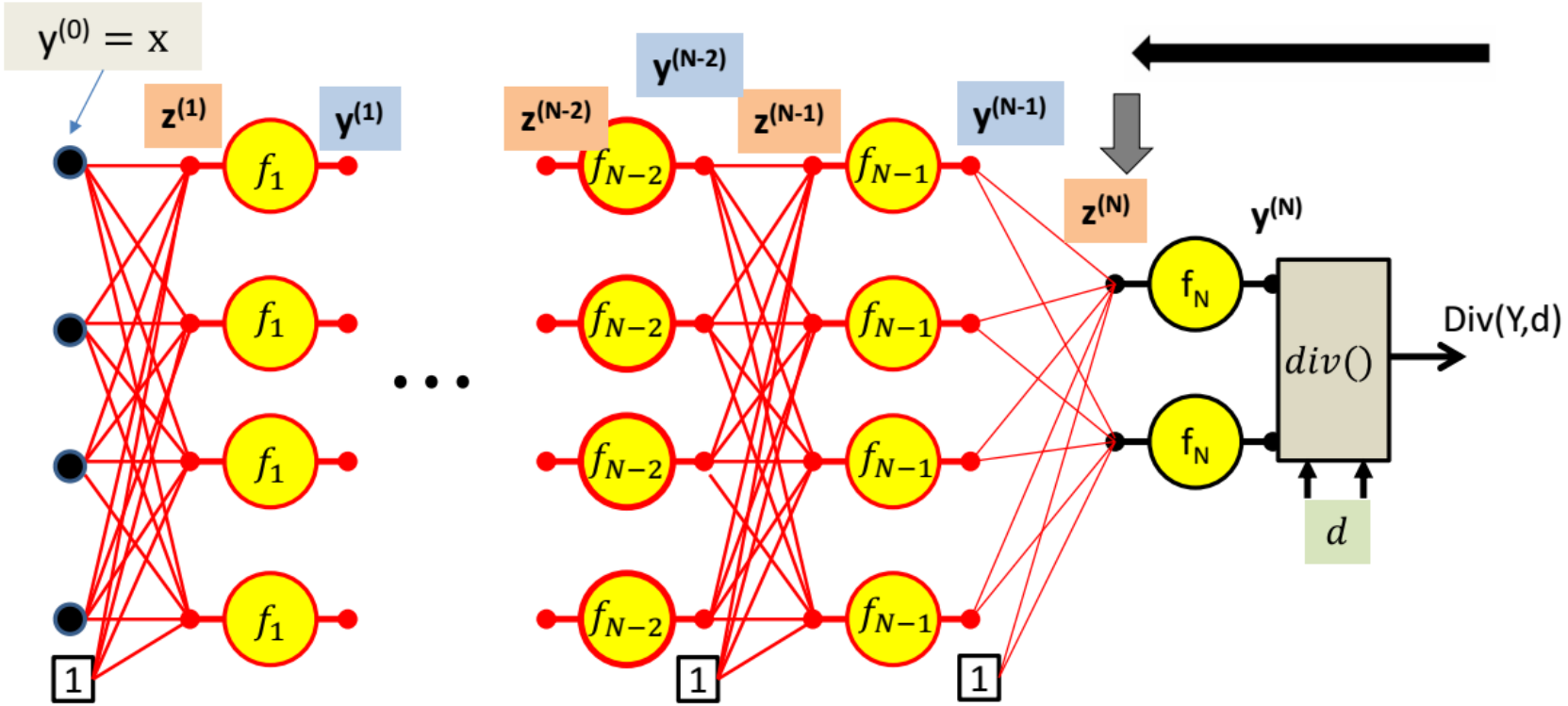
Computing derivatives



Computing derivatives

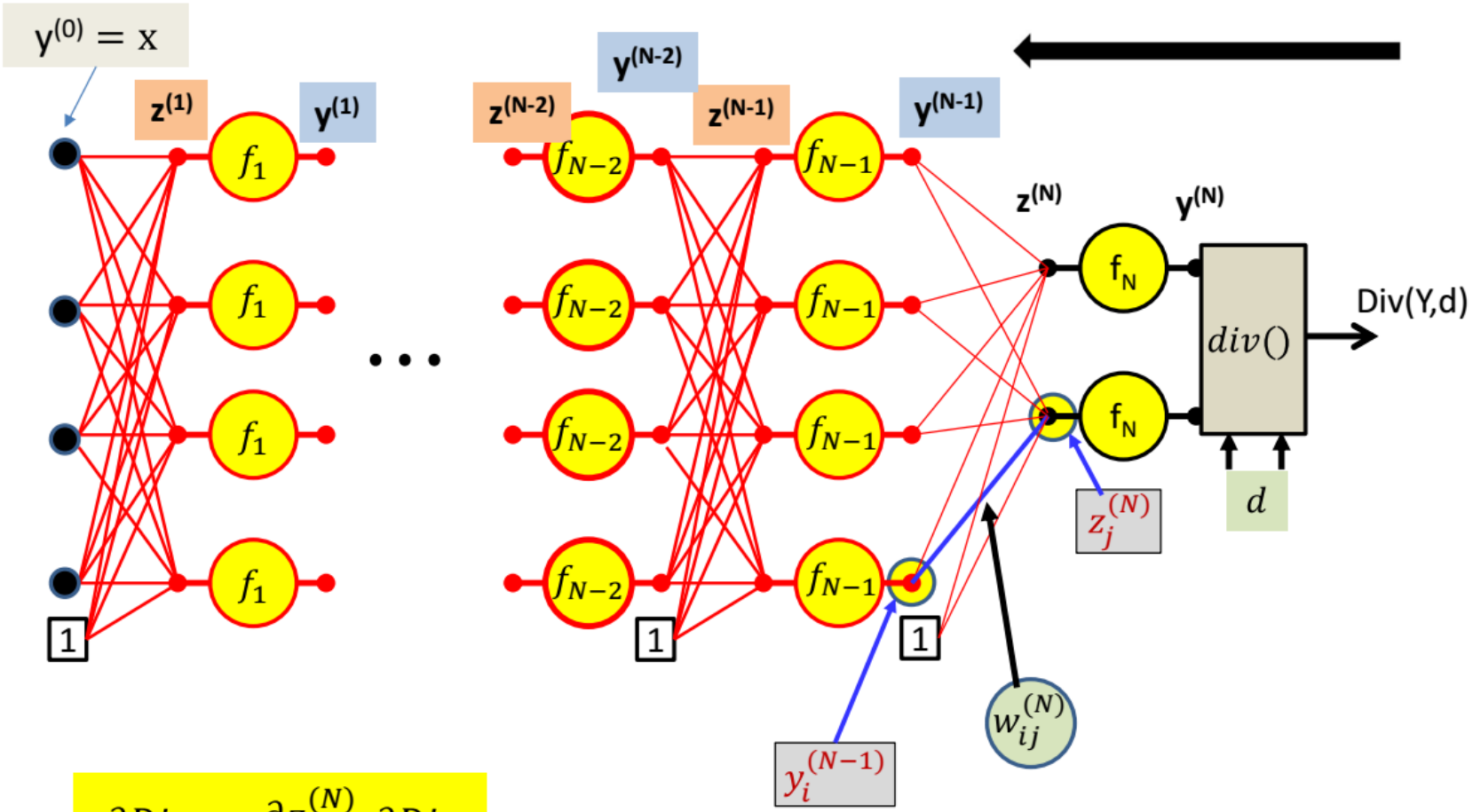


Computing derivatives



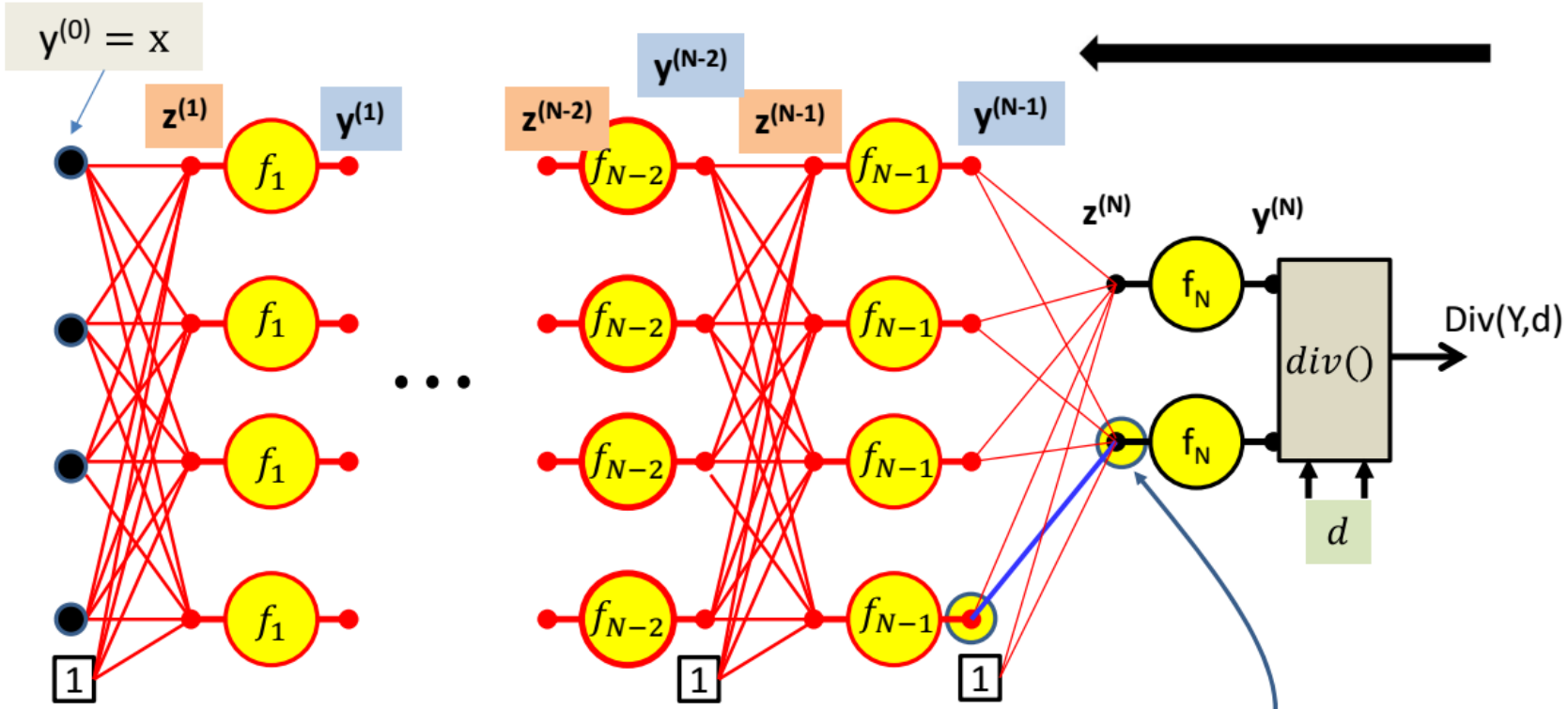
$$\frac{\partial Div}{\partial z_i^{(N)}} = f_N'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



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

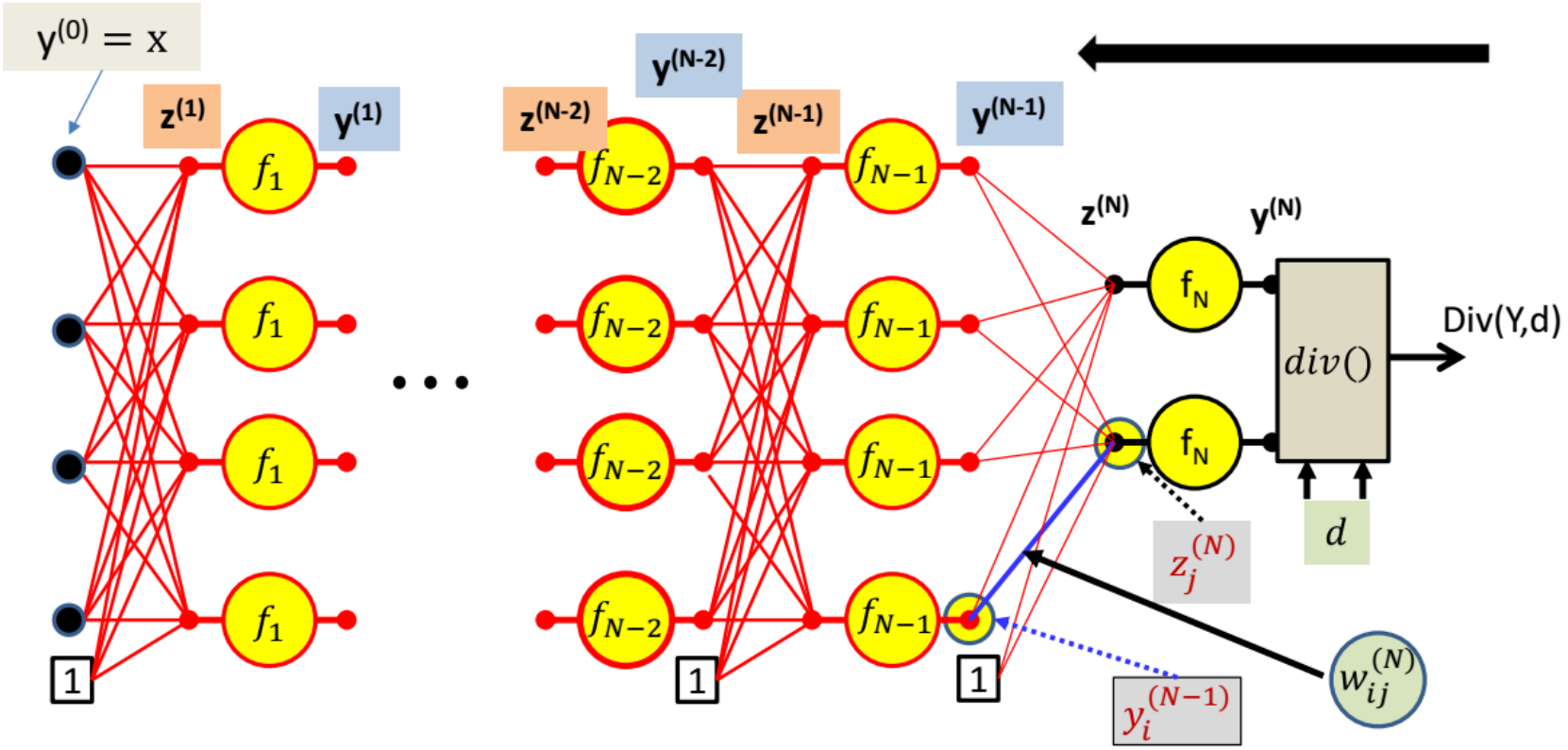
Computing derivatives



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

Just computed

Computing derivatives



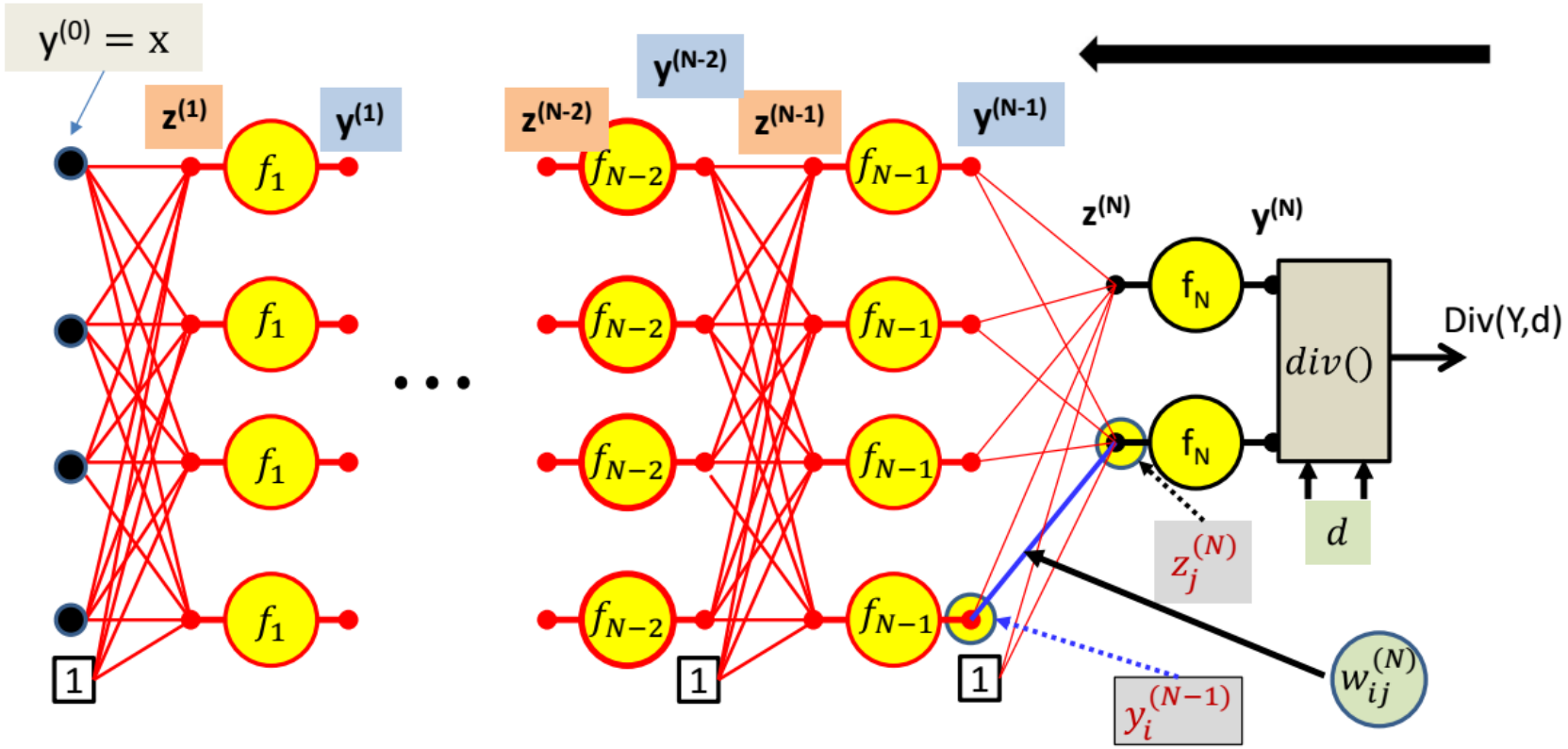
$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$

Because

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

Computing derivatives



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

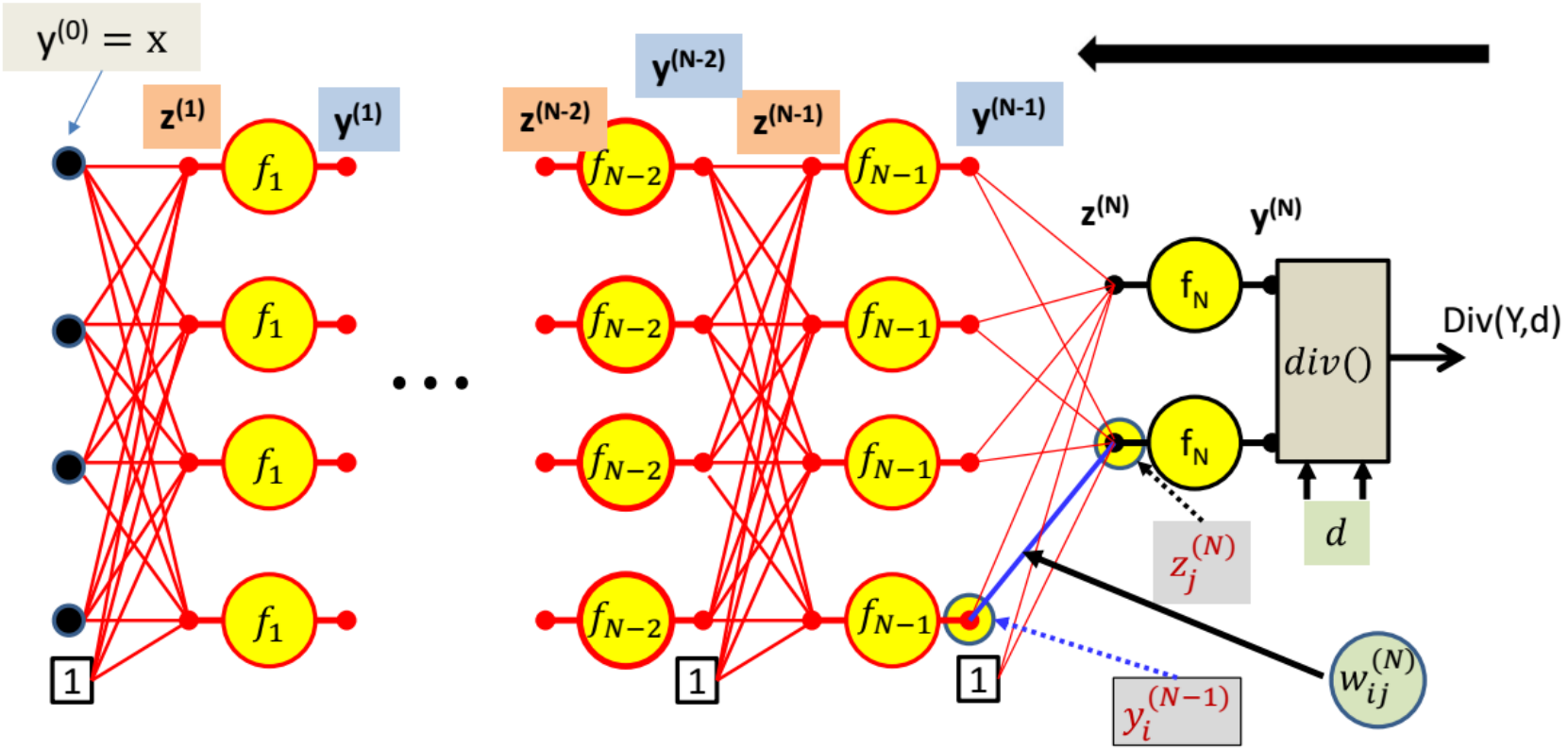
$y_i^{(N-1)}$

Computed in forward pass

Because

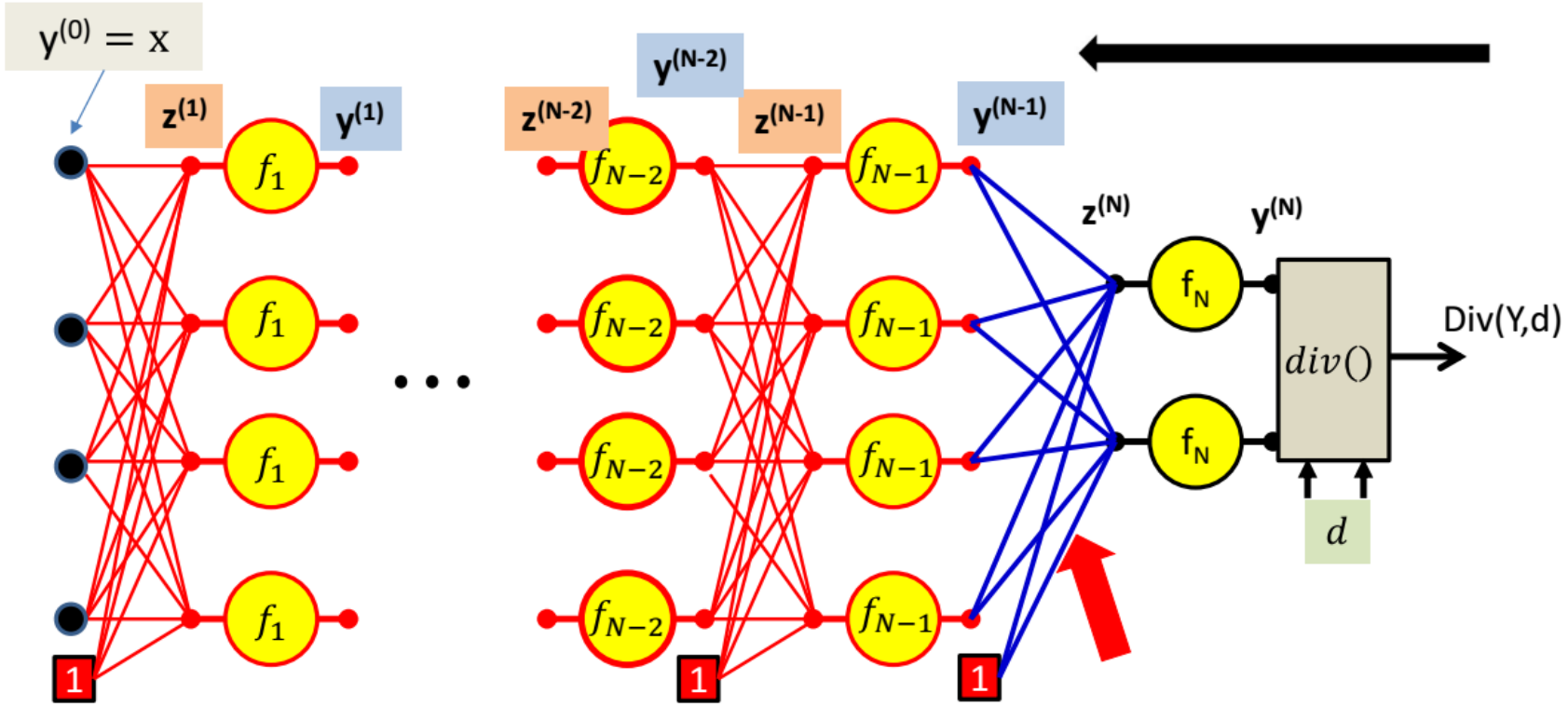
$$z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$$

Computing derivatives



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

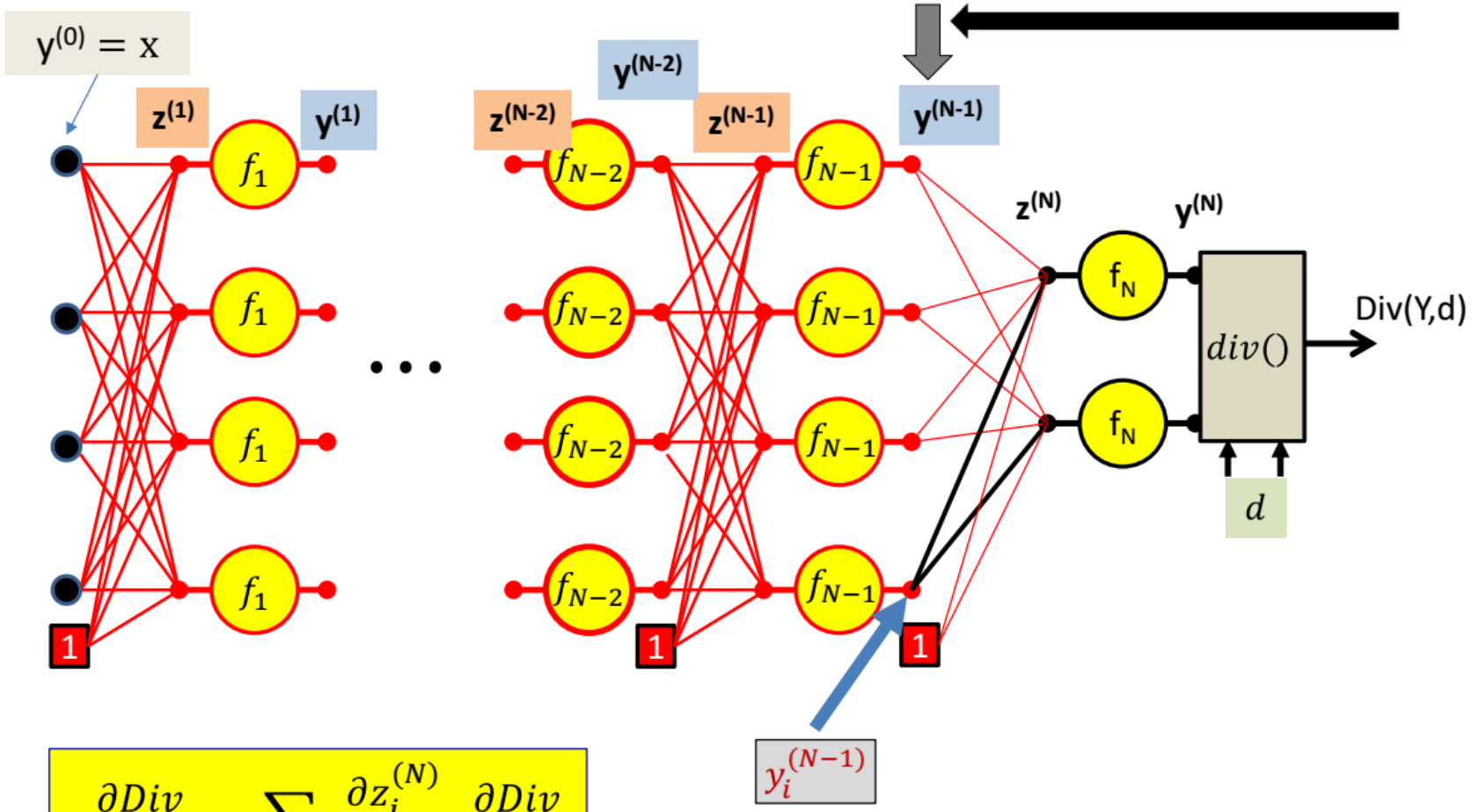
Computing derivatives



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

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

Computing derivatives



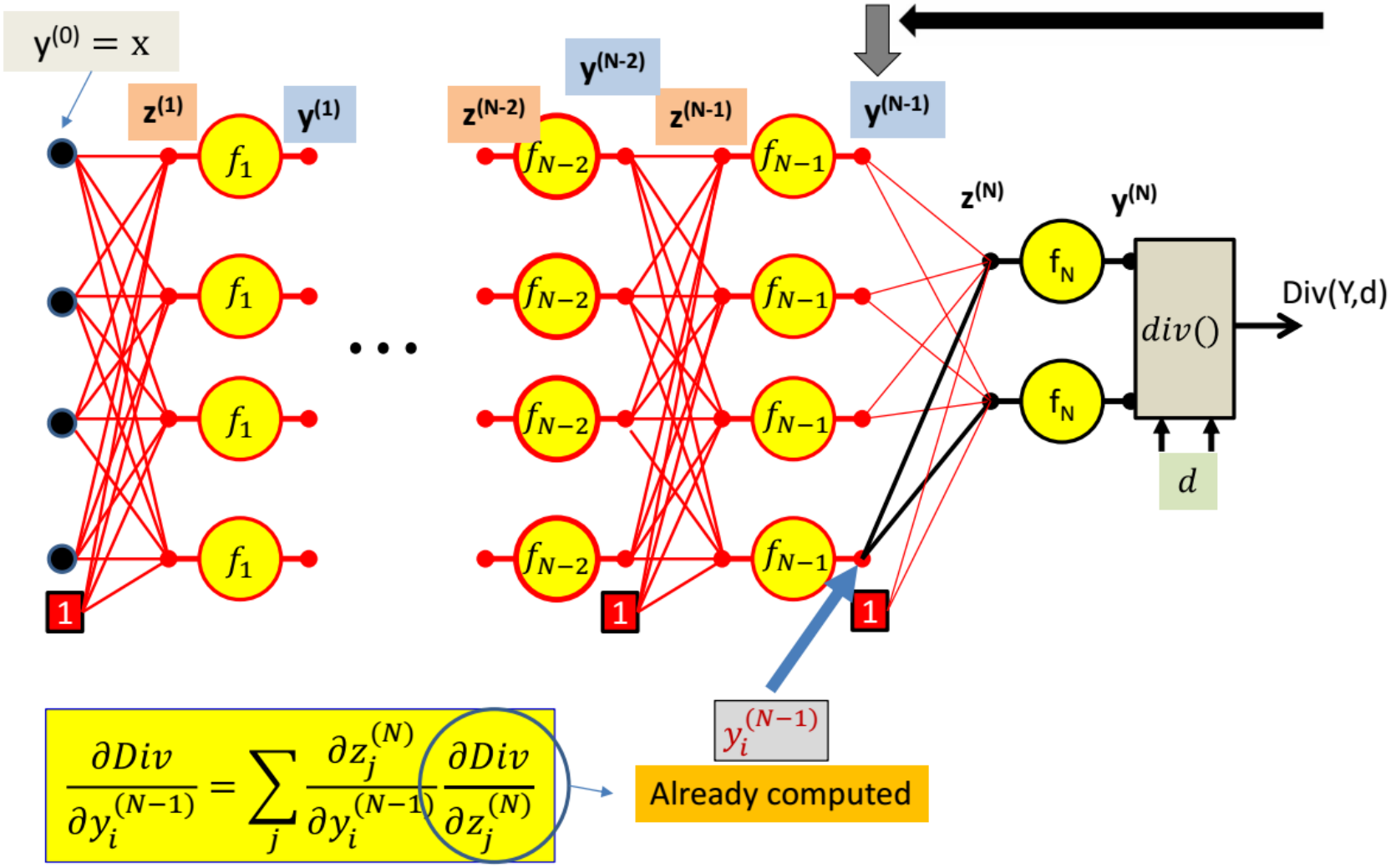
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

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

where $z_i = g_i(x)$

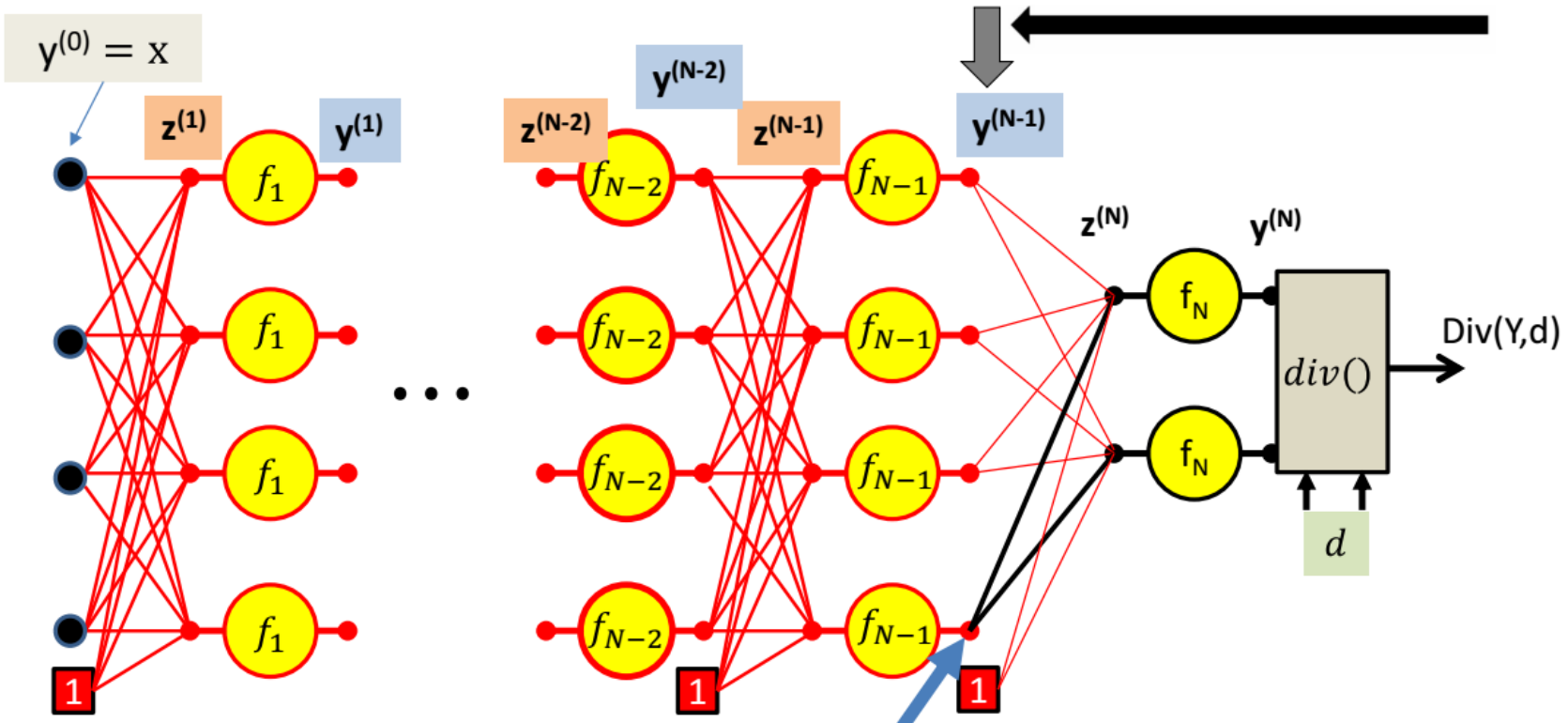
$$\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx} + \cdots + \frac{\partial l}{\partial z_M}\frac{dz_M}{dx}$$

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

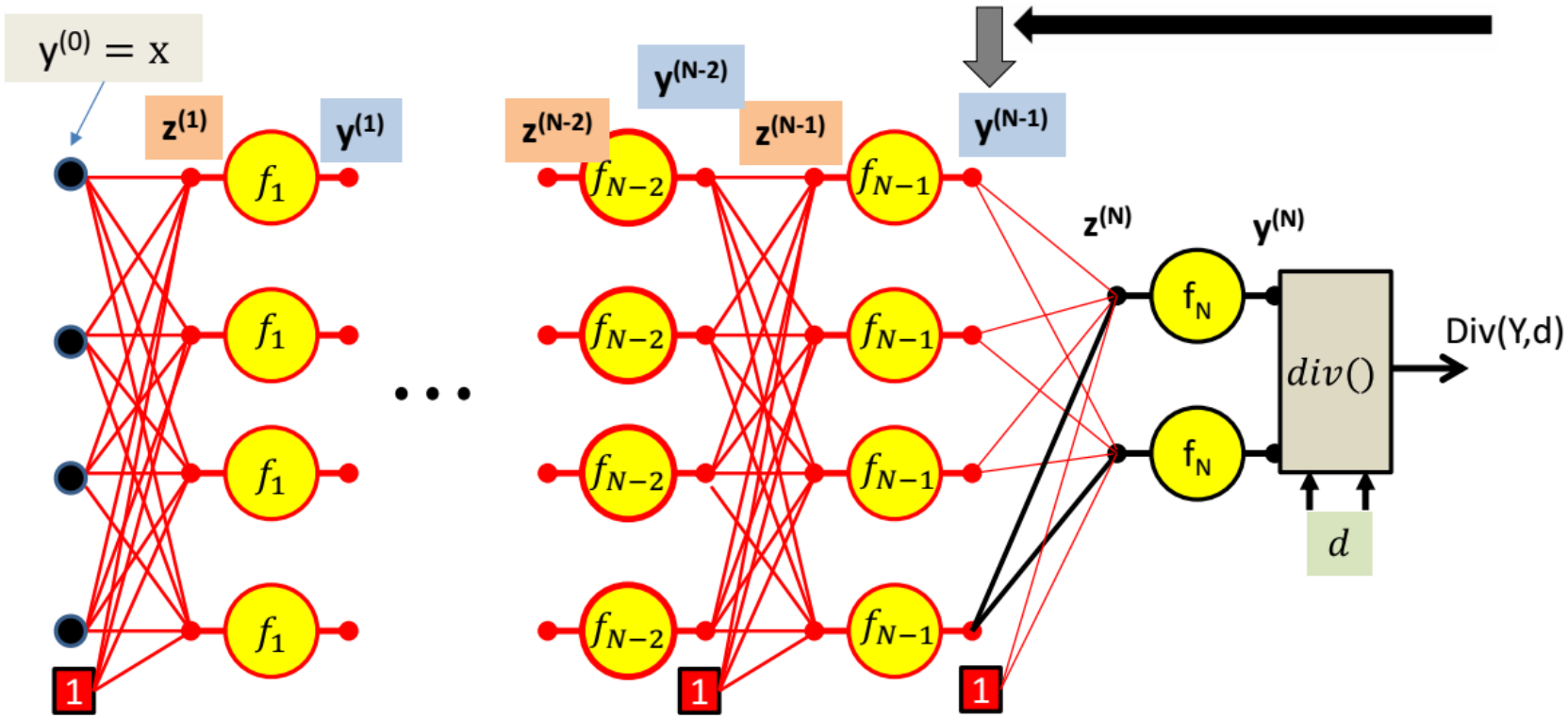


$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

Because

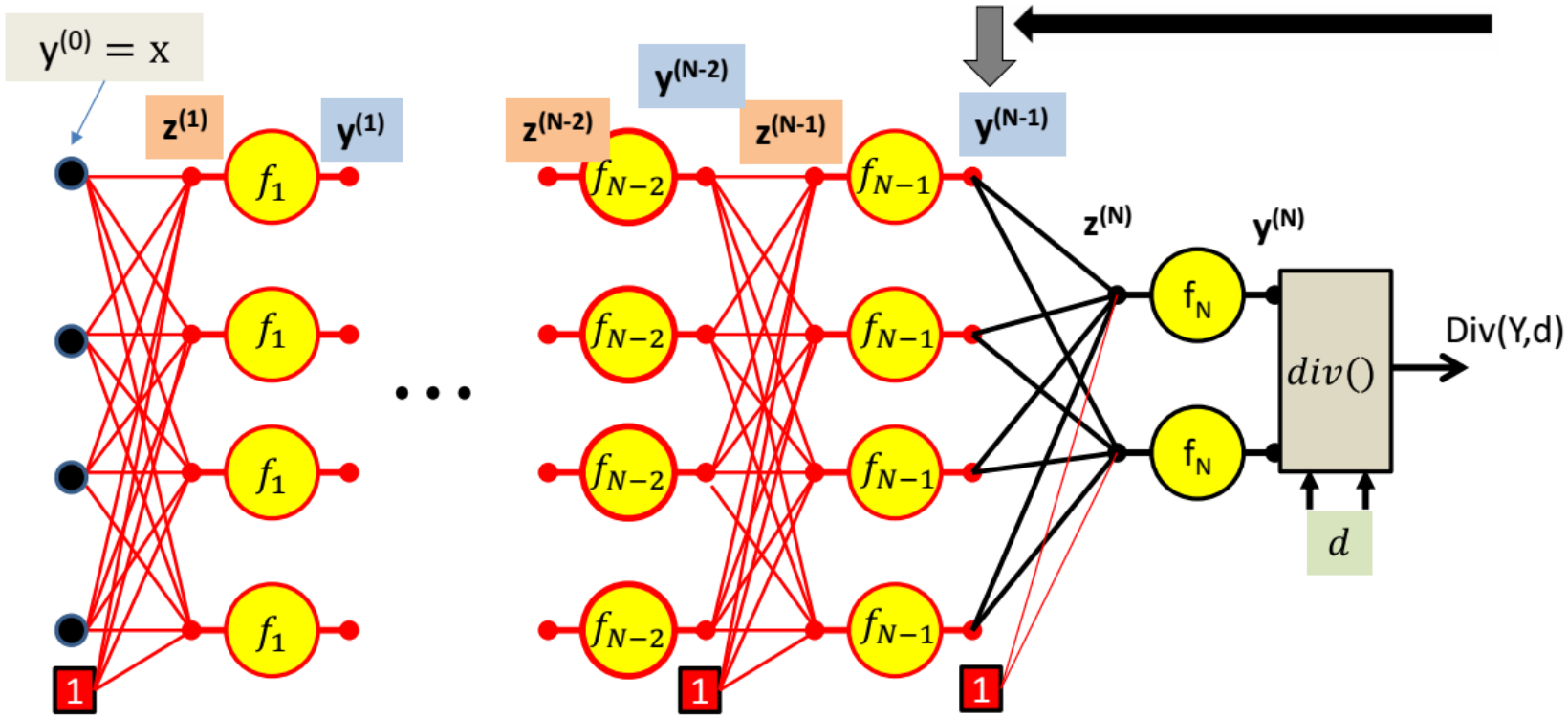
$$z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$$

Computing derivatives



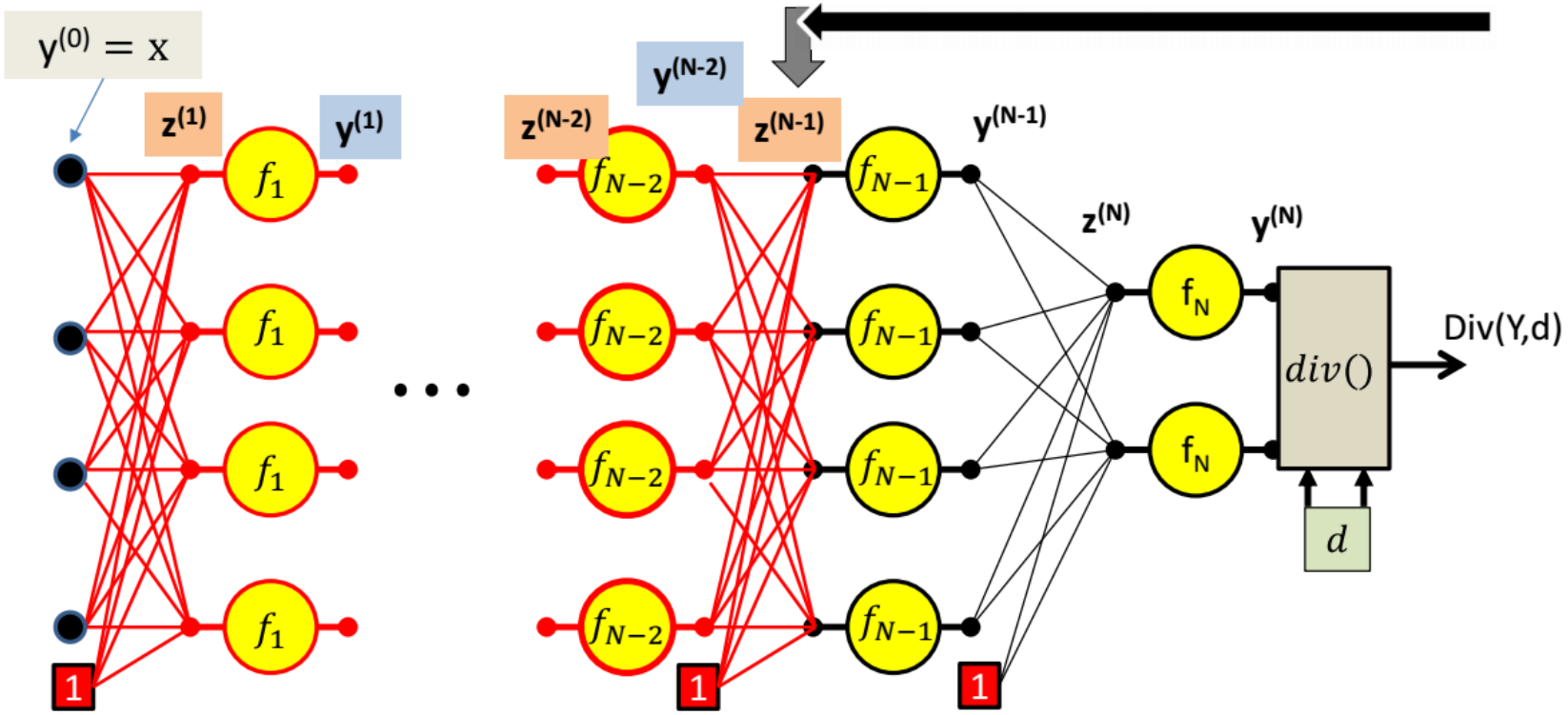
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



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

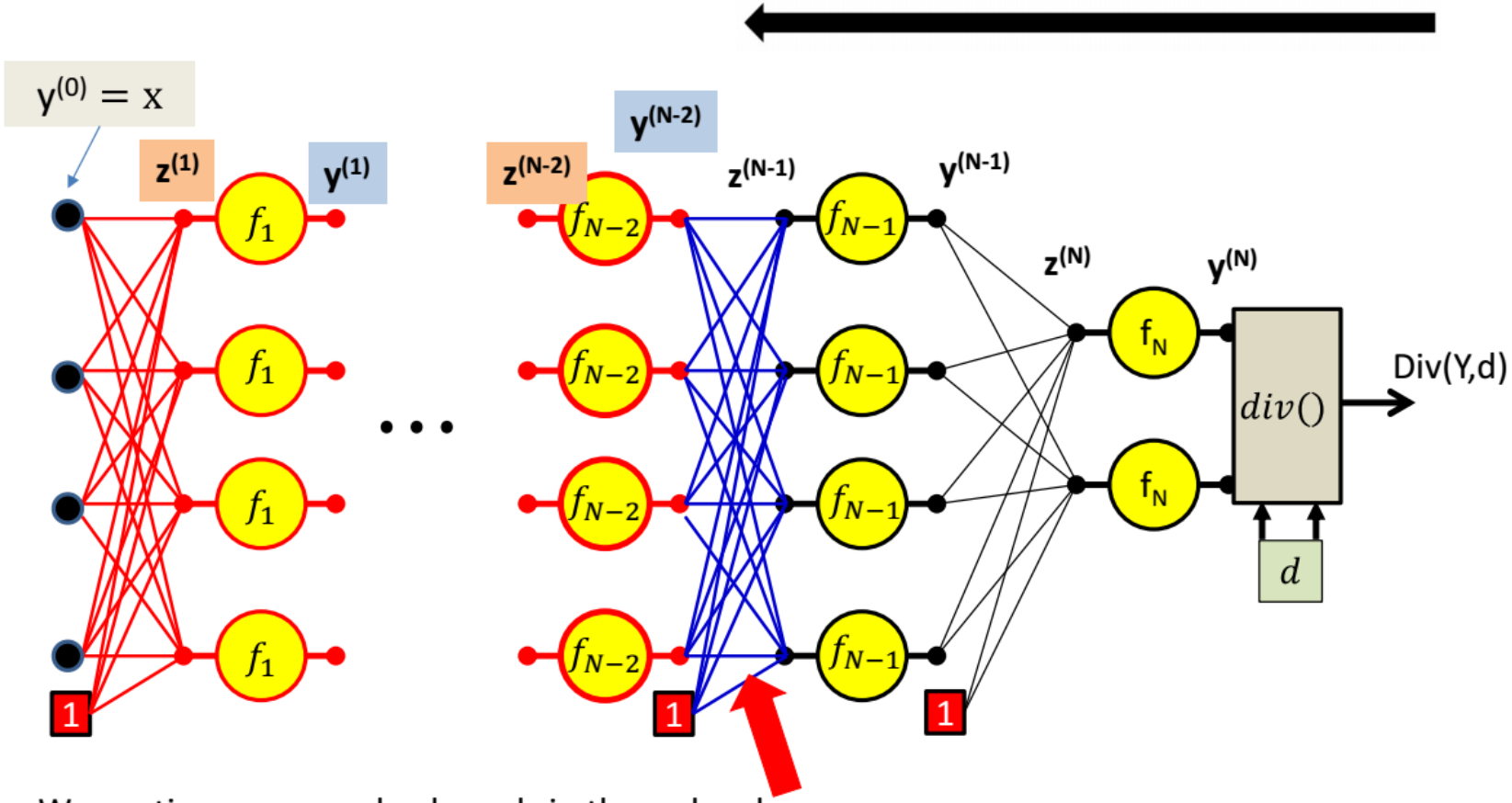
Computing derivatives



We continue our way backwards in the order shown

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

Computing derivatives

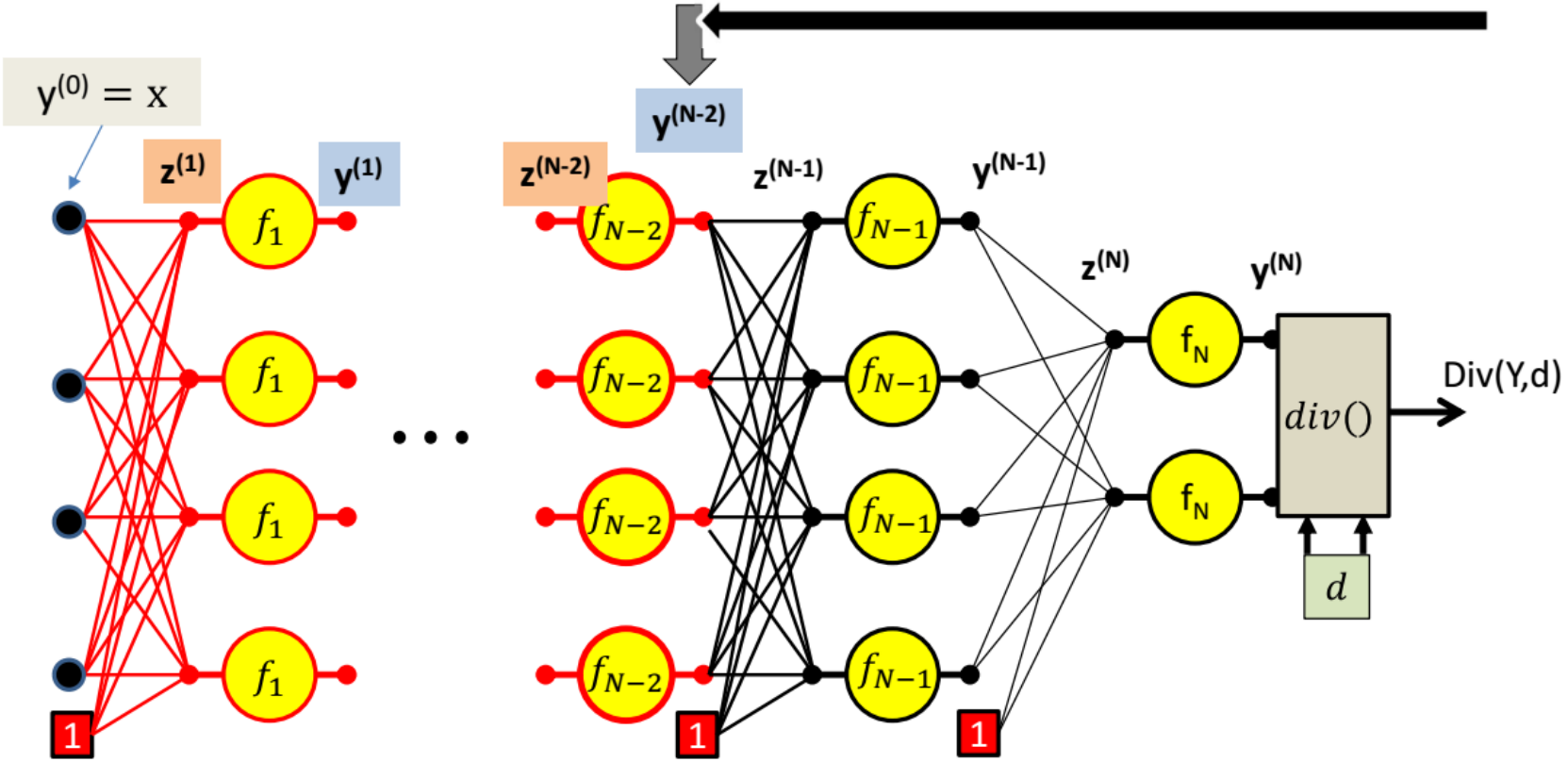


We continue our way backwards in the order shown

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

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

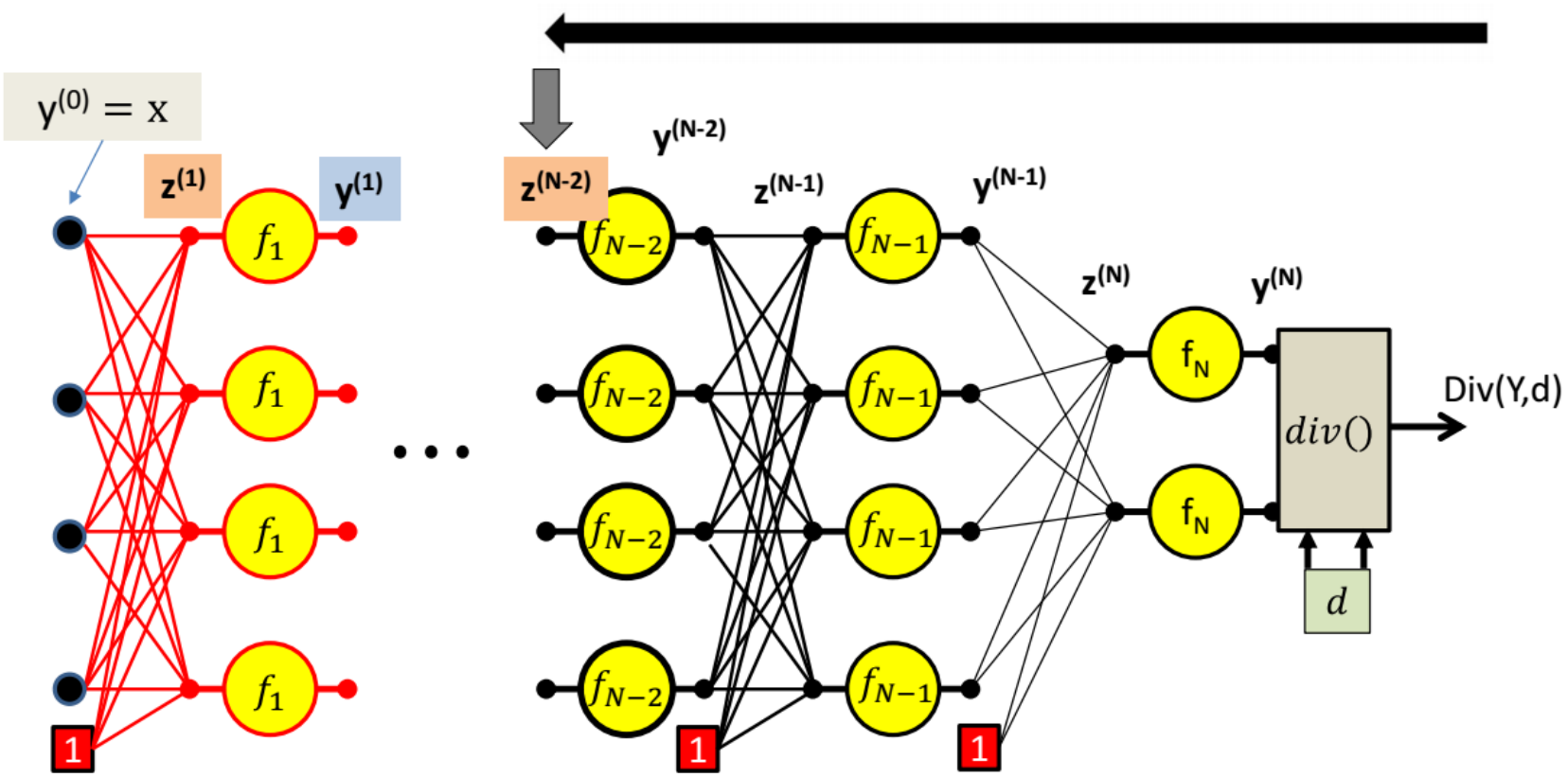
Computing derivatives



We continue our way backwards in the order shown

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

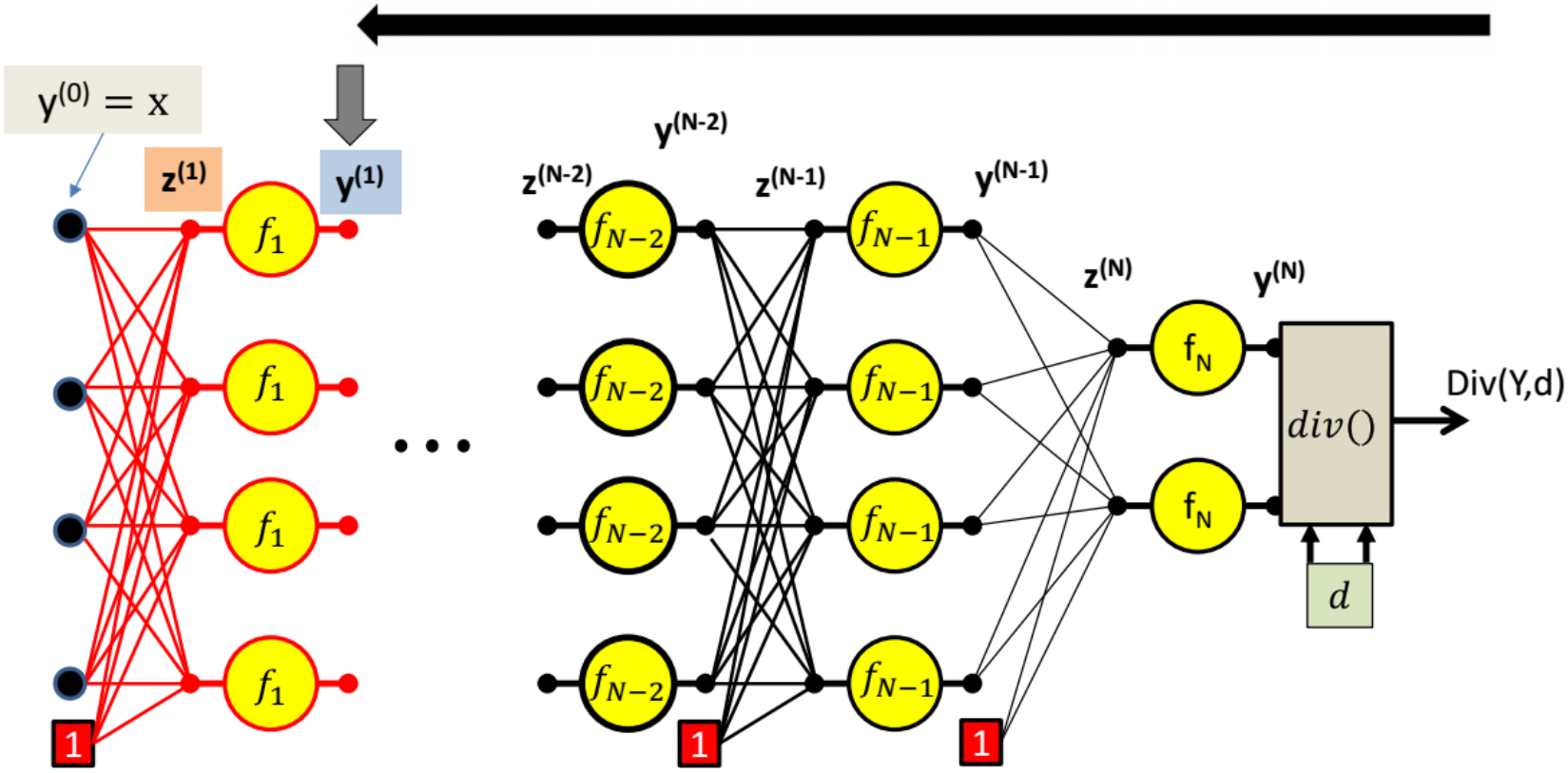
Computing derivatives



We continue our way backwards in the order shown

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

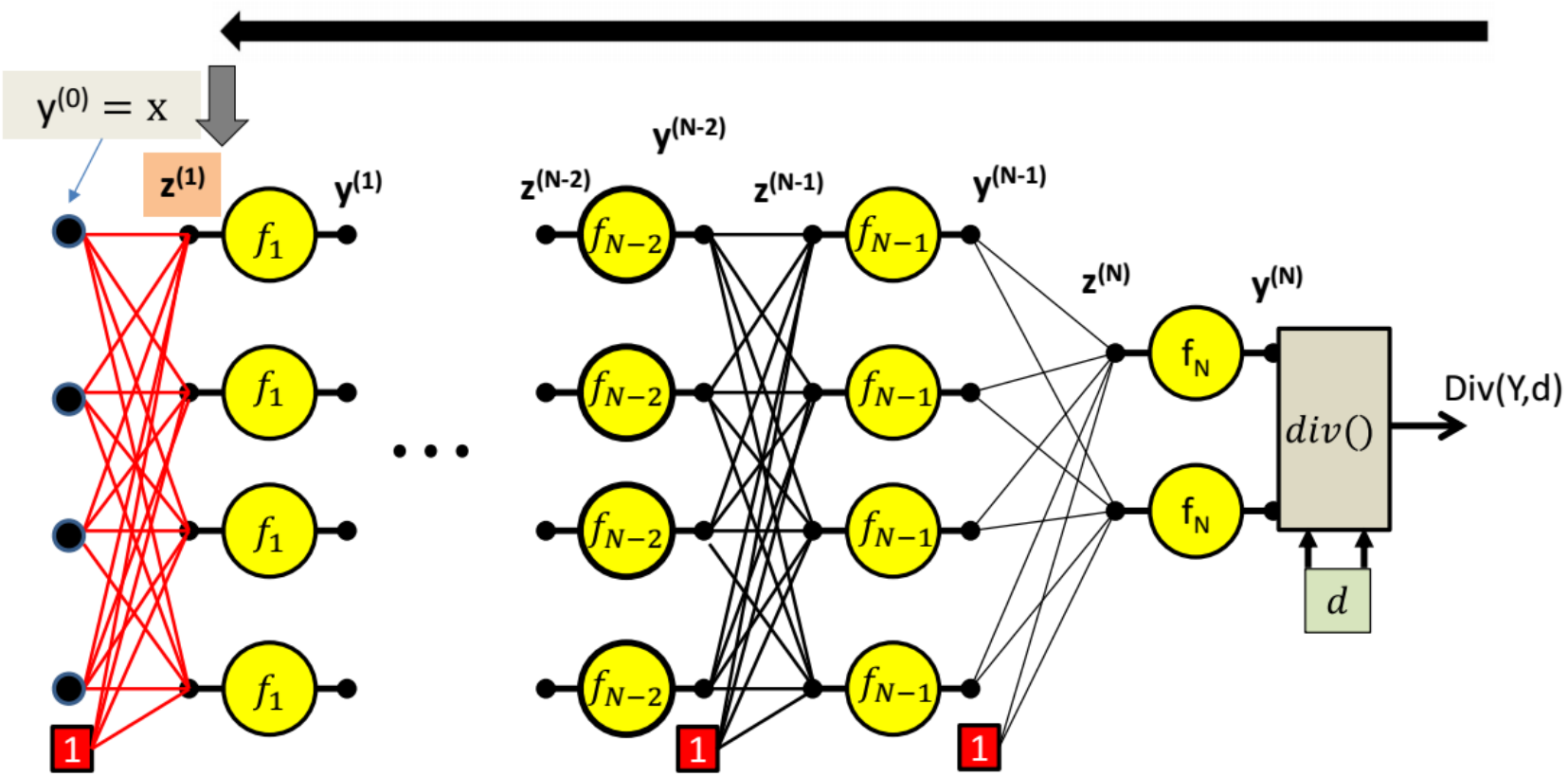
Computing derivatives



We continue our way backwards in the order shown

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

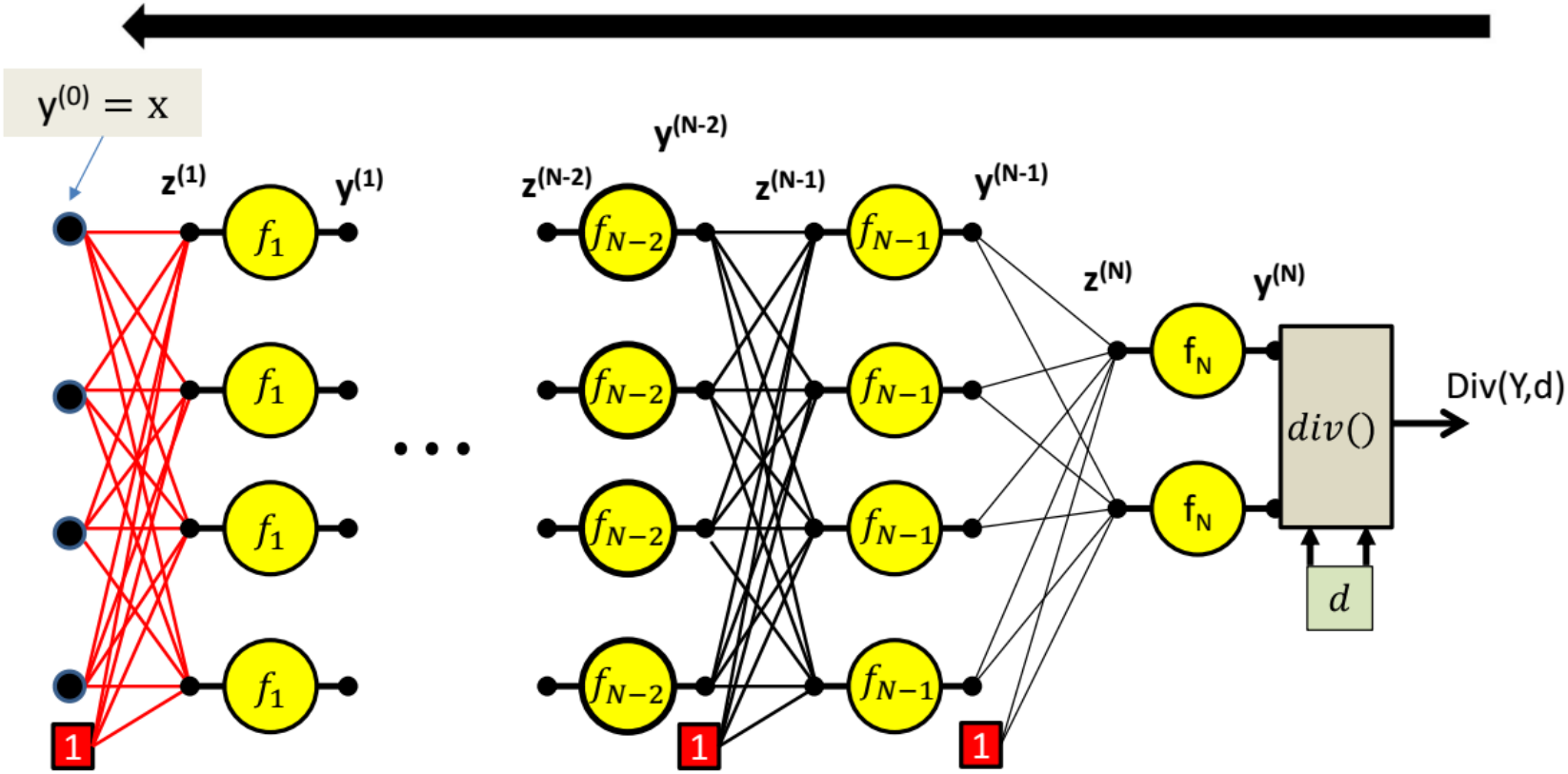
Computing derivatives



We continue our way backwards in the order shown

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

Computing derivatives



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

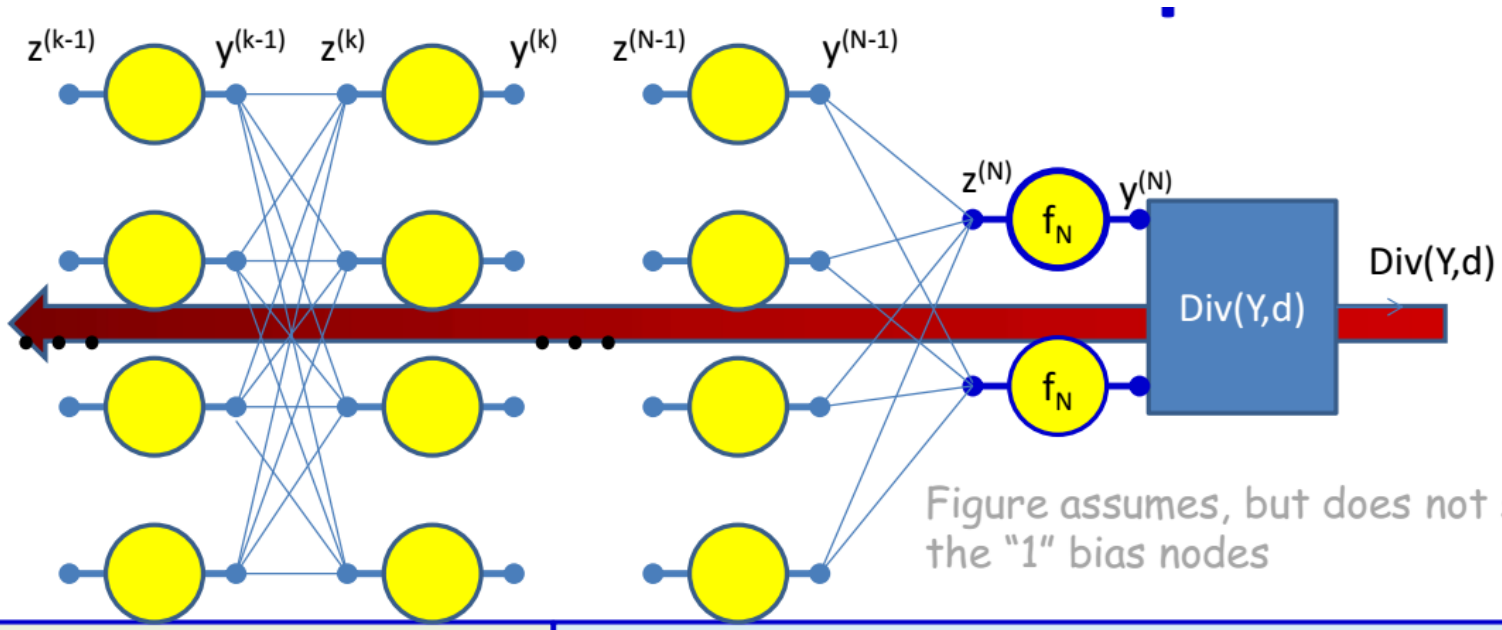


Figure assumes, but does not show the "1" bias nodes

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

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

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

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

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]
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Di}{\partial y_i^{(N)}} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$ for i = $0 \ldots D_{N-1}$, j = $1 \ldots D_N$
- 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 i = $0 \ldots D_{k-1}$, j = $1 \ldots D_K$

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 i = 0 … D_{N-1} , j = 1 … D_N

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

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

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 $j = 1 \ldots D_k$
    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k\left(z_j^{(k)}\right)$
- Output:
  - $Y = y_j^{(N)}, j = 1..D_N$

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

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

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

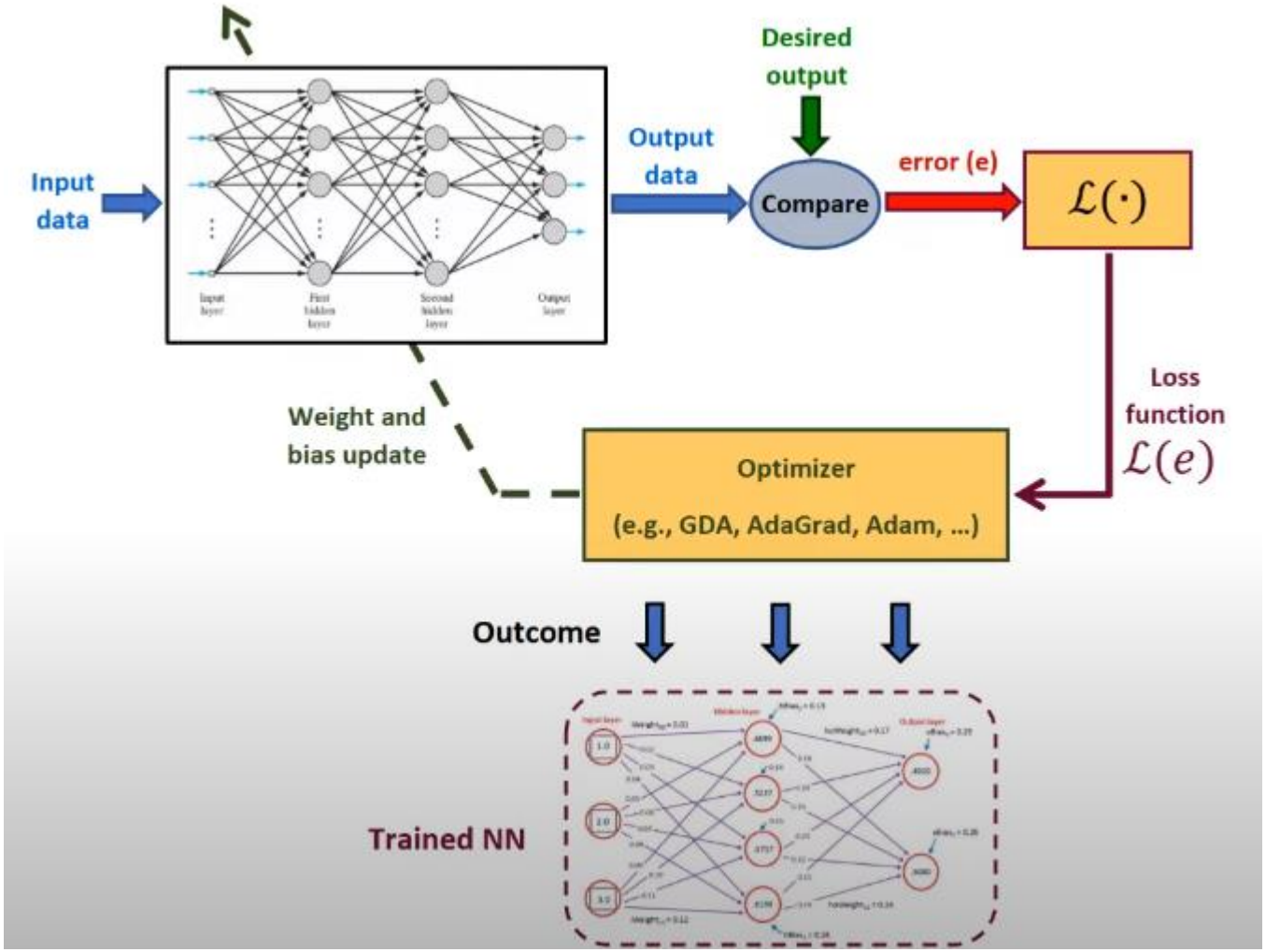
Training by BackProp

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

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

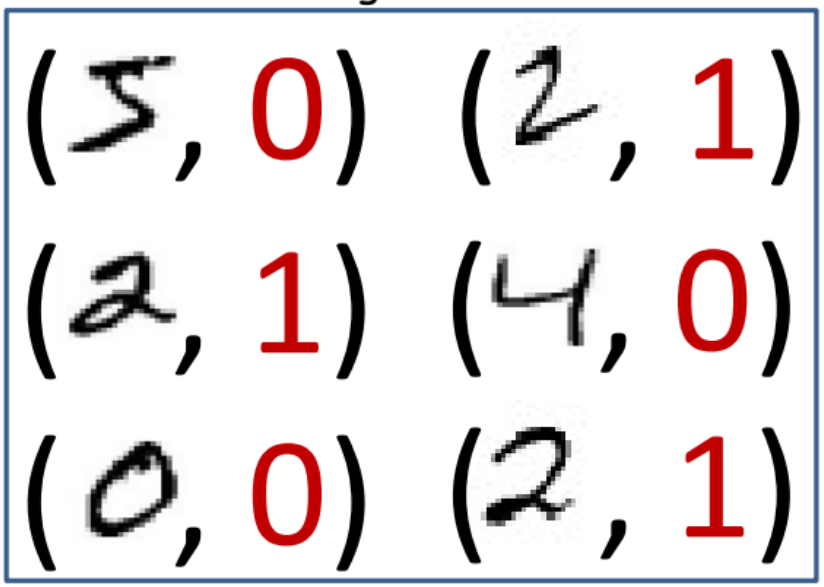
- Until $Loss$ has converged

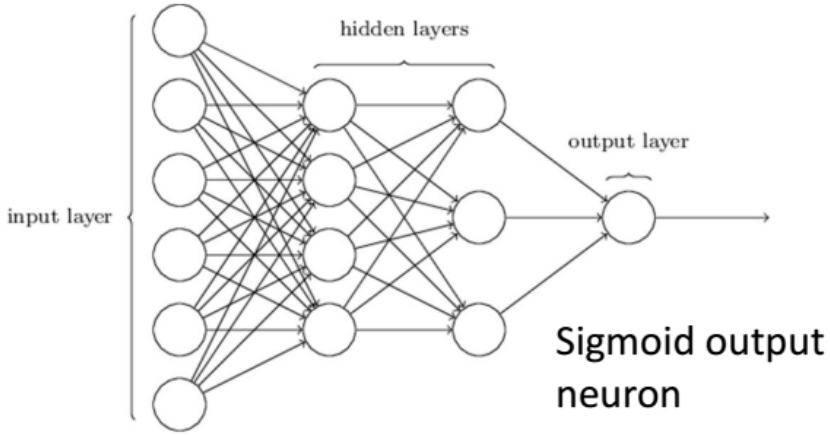
Training by BackProp



Setting up for digit recognition

Training data

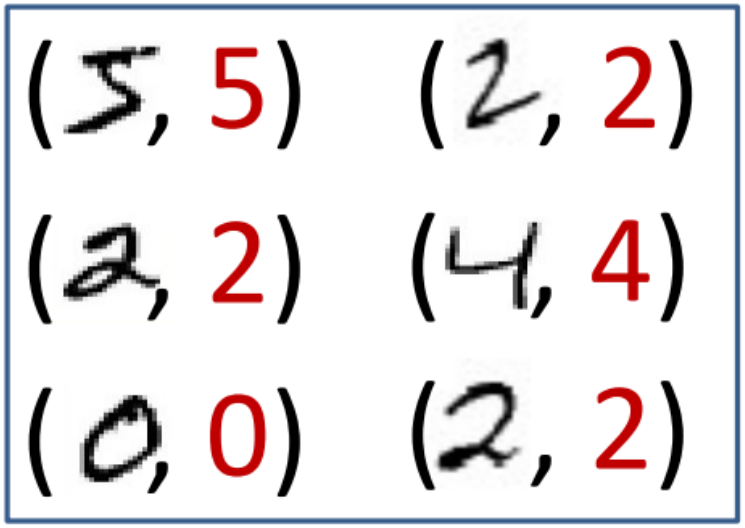


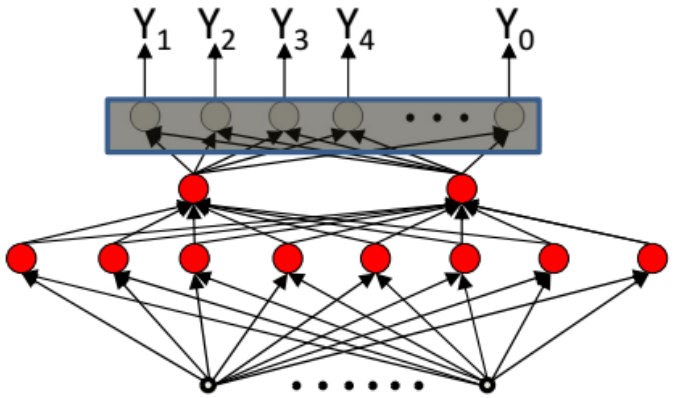


- Simple Problem: Recognizing "2" or "not 2"
- Single output with sigmoid activation
  - $Y \in (0,1)$
  - d *is either* 0 *or* 1
- Use KL divergence
- Backpropagation to compute derivatives
  - To apply in gradient descent to learn network parameters

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

Next

- Backpropagation in practice…