

# **CS 190I**

# **Deep Learning**

# **Learning FFN**

Lei Li (leili@cs)

UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and  
Mu Li & Alex Smola's 157 courses on Deep Learning, with  
modification

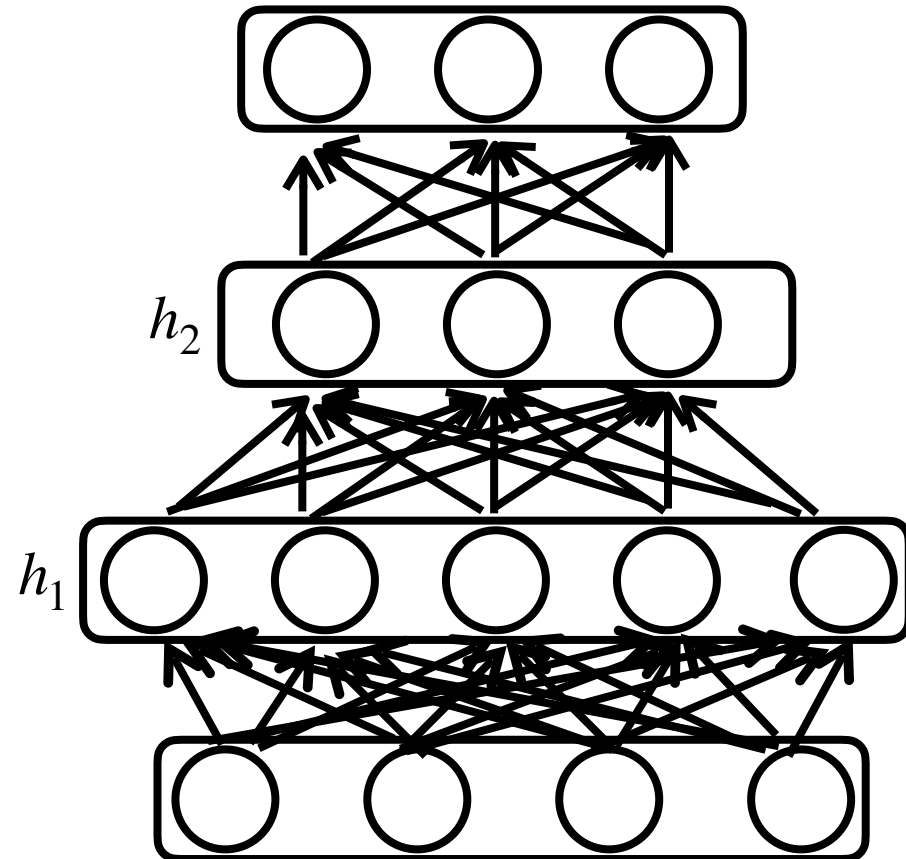
# Recap

---

- Feedforward network (also Multilayer Perceptron)
- Empirical Risk minimization framework for learning
- Loss function for classification and regression
- First-order optimality condition:  $\text{gradient}=0$
- Gradient descent is an iterative algorithm to update the parameter towards the opposite direction of gradient

# Feedforward Neural Net (FFN)

- also known as multilayer perceptron (MLP)
- Layers are connected sequentially
- Each layer has full-connection (each unit is connected to all units of next layer)
  - Linear project followed by
  - an element-wise nonlinear activation function
- There is no connection from output to input



# Empirical Risk Minimization

---

- The expected risk is the average risk (loss) over the entire  $(x, y)$  data space

$$R(\theta) = E_{\langle x, y \rangle \in P} [\ell(y, f(x; \theta))] = \int \ell(y, f(x; \theta)) dP(x, y)$$

- Instead, given a training set of empirical data

$$D = \{(x_n, y_n)\}_{n=1}^N$$

- Minimize **the empirical risk** over training data

$$\hat{\theta} \leftarrow \arg \min_{\theta} L(\theta) = \frac{1}{N} \sum_n \ell(y_n, f(x_n; \theta))$$

# Gradient Descent

---

- Update rule:  $x_{t+1} = x_t - \eta \nabla f|_{x_t}$
- $\eta$  is a hyper-parameter to control the learning rate
- How to compute gradient for FFN?

# Quiz

---

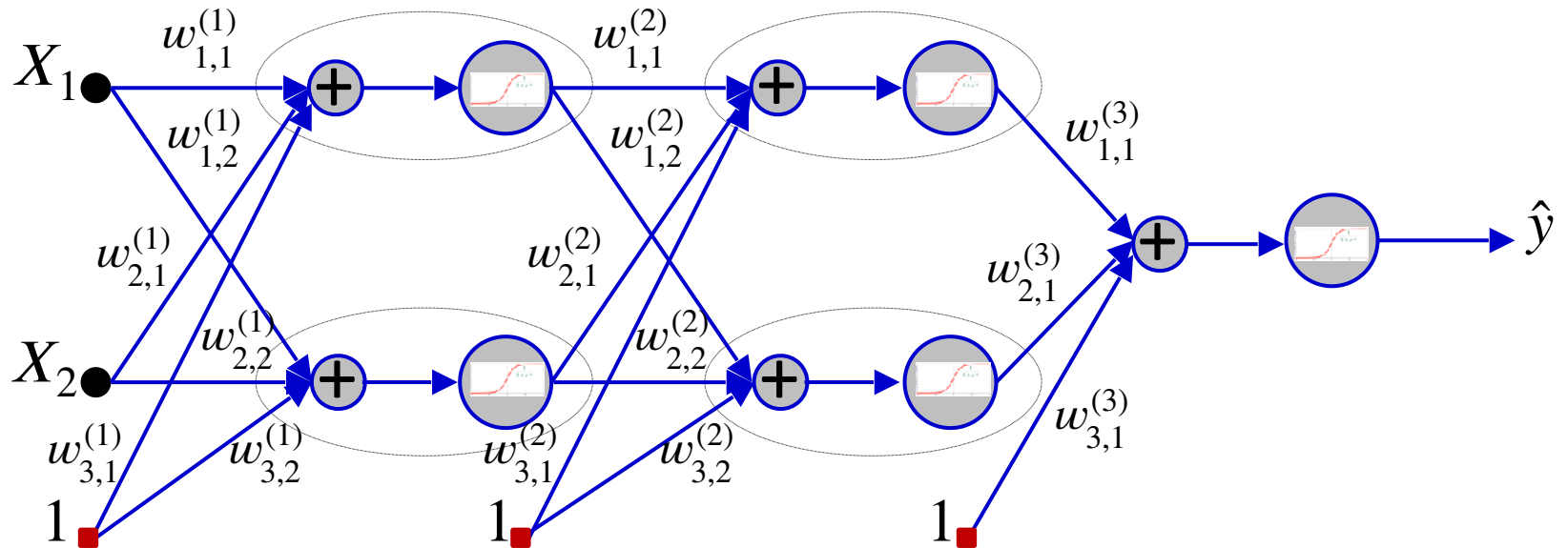
- <https://edstem.org/us/courses/31035/lessons/54638/slides/309014>

# Computing Gradient for Neural Net

---

- Forward and back-propagation
- Suppose  $y=f(x)$ ,  $z=g(y)$ , therefore  $z=g(f(x))$
- Use the chain rule,
$$\nabla g(f(x)) \big|_x = (\nabla f \big|_x)^T \cdot \nabla g \big|_y$$
- For a neural net and its loss  $\ell(\theta)$
- First compute gradient with respect to last layer
- then using chain-rule to back propagate to second last, and so on

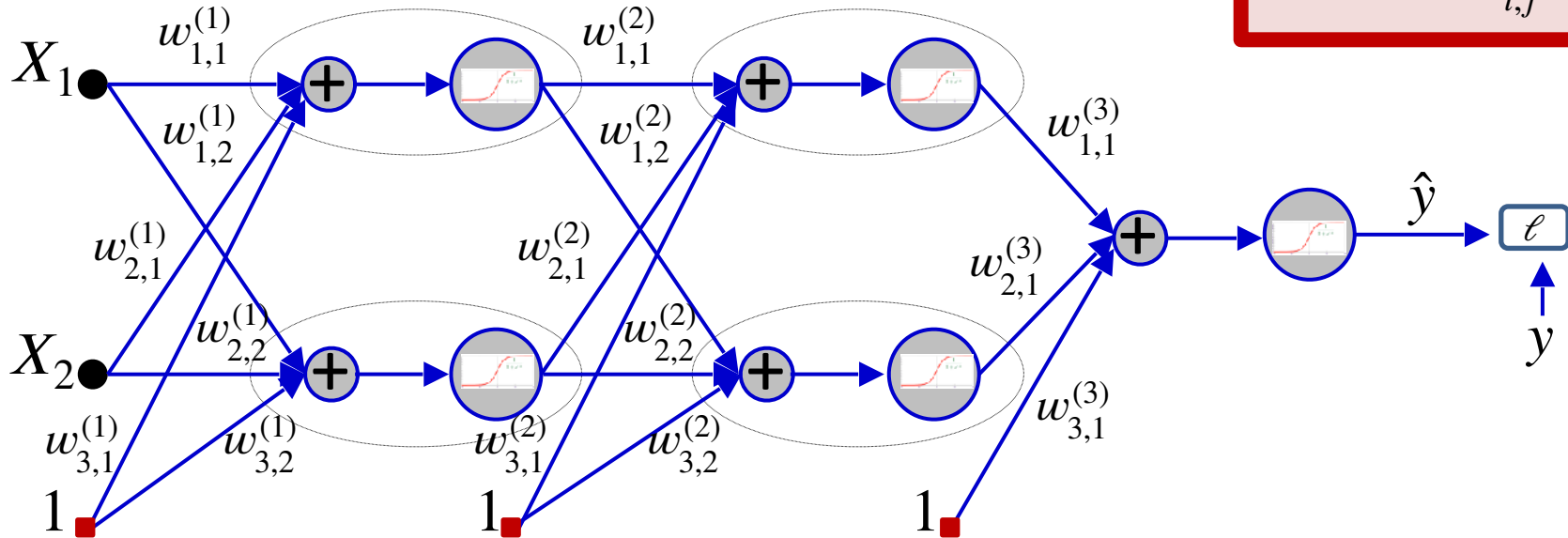
# Example





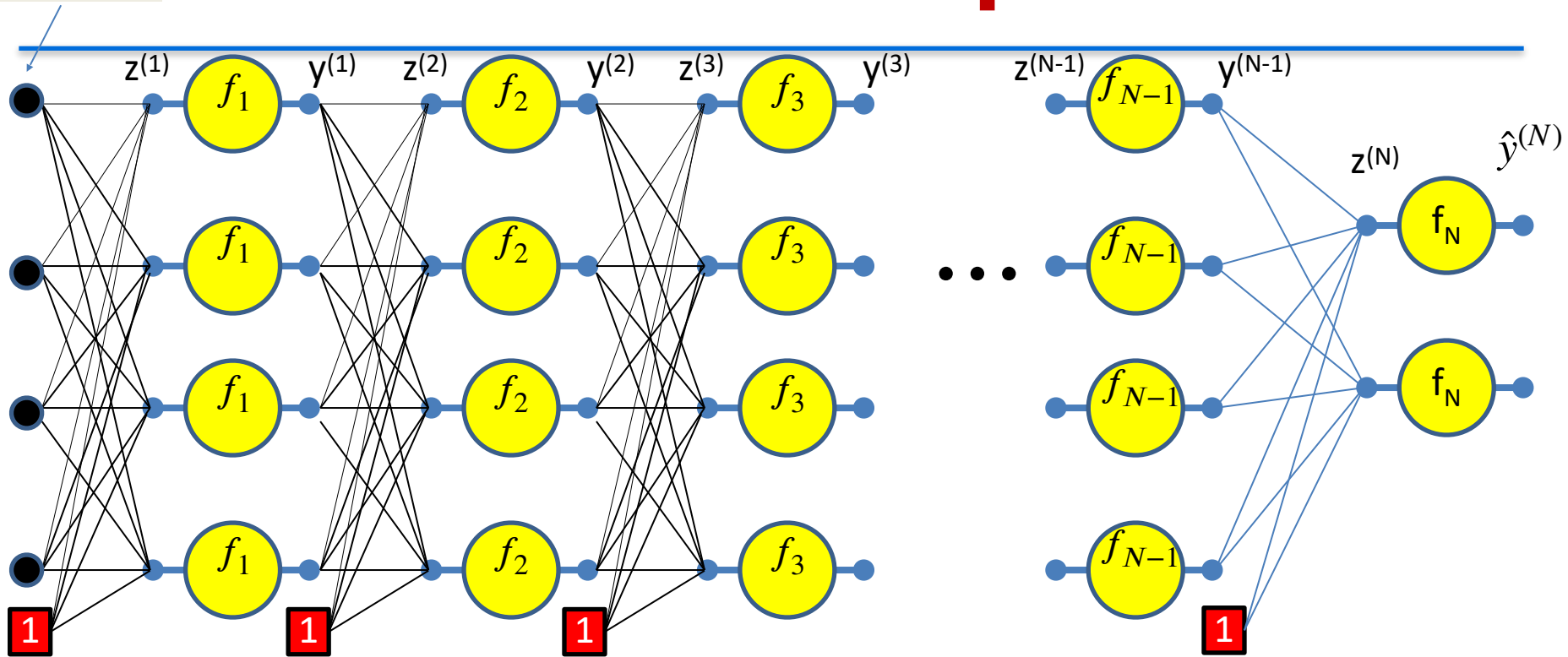
# Computing the Gradient

What is:  $\frac{\partial \ell(y, \hat{y})}{\partial w_{i,j}^{(k)}}$



# The “forward pass”

$$y(0) = x$$

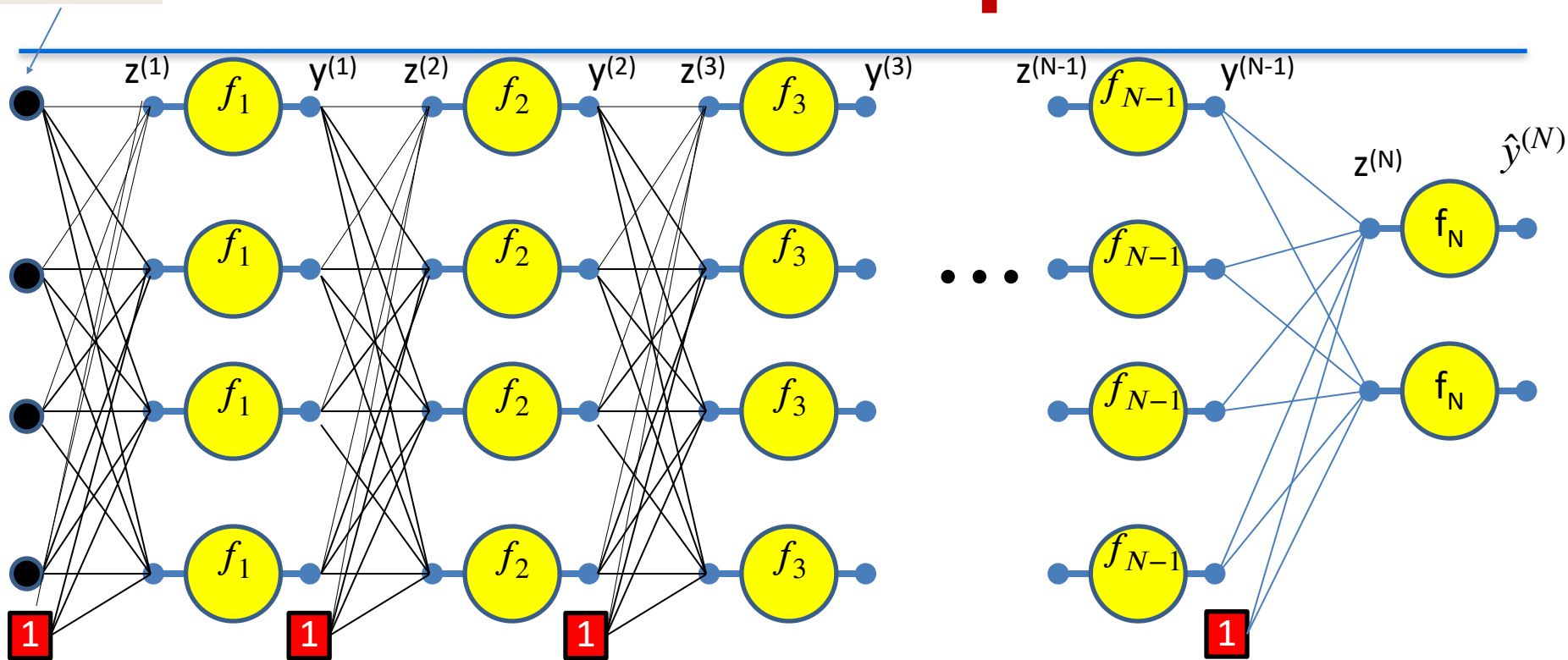


We will refer to the process of computing the output from an input as the forward pass

We will illustrate the forward pass in the following slides

# The “forward pass”

$$y(0) = x$$

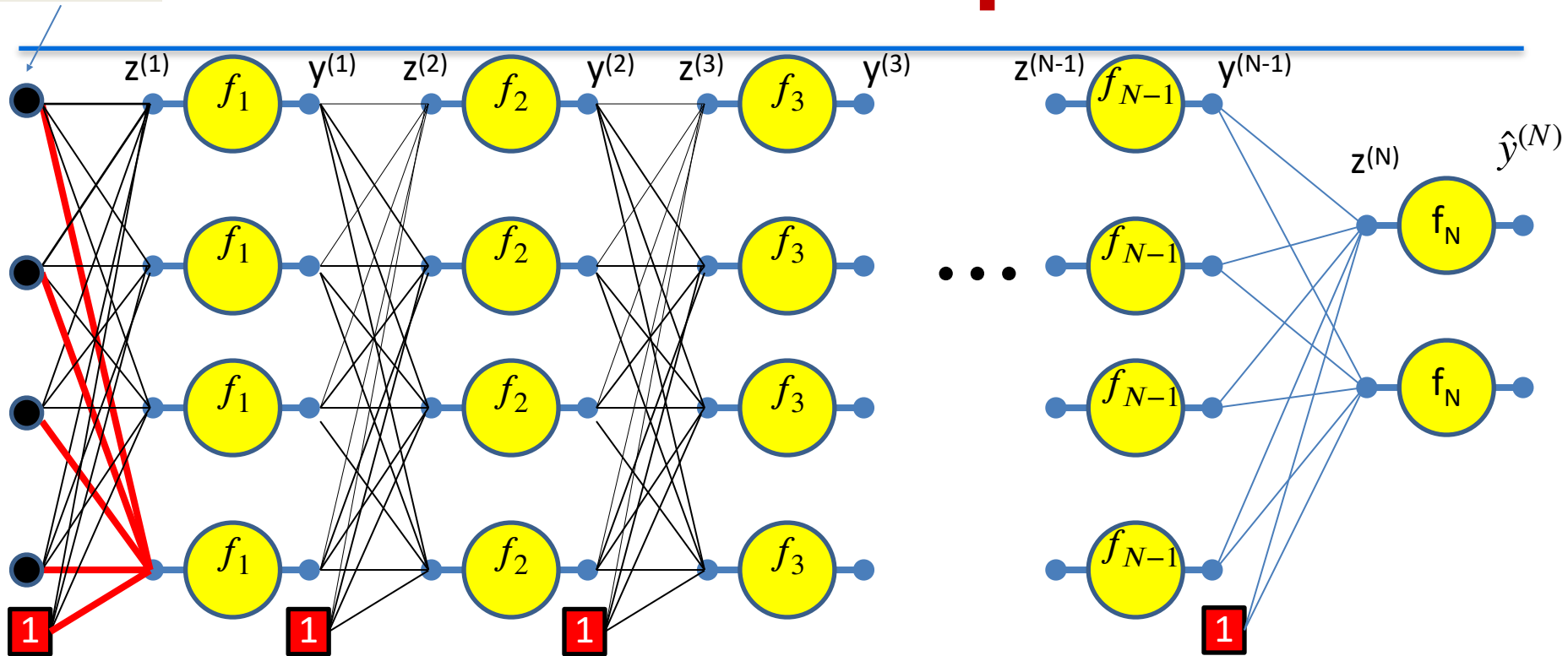


Setting  $y_i^{(0)} = x_i$  for notational convenience

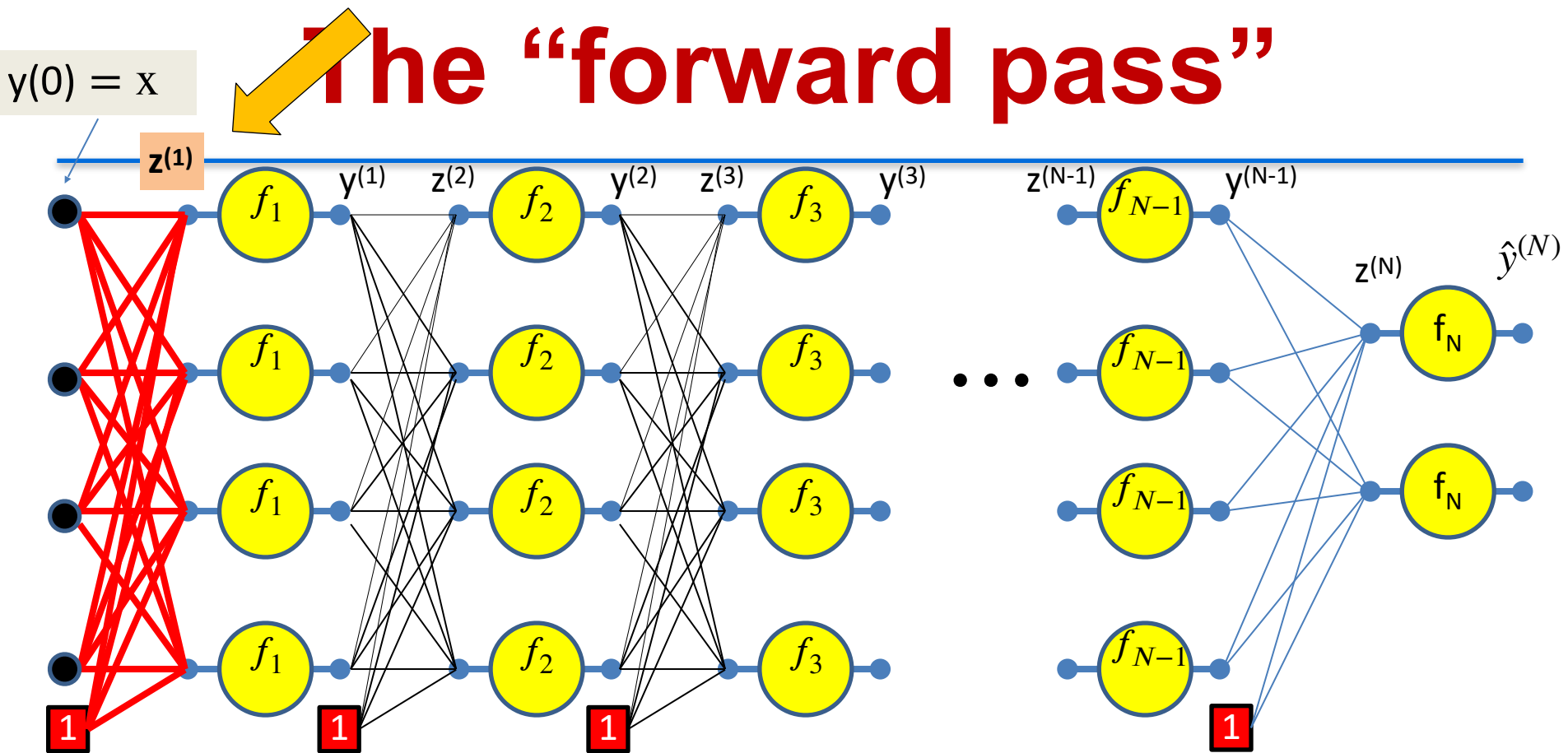
Assuming  $w_{0j}^{(k)} = b_j^{(k)}$  and  $y_0^{(k)} = 1$  -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases

# The “forward pass”

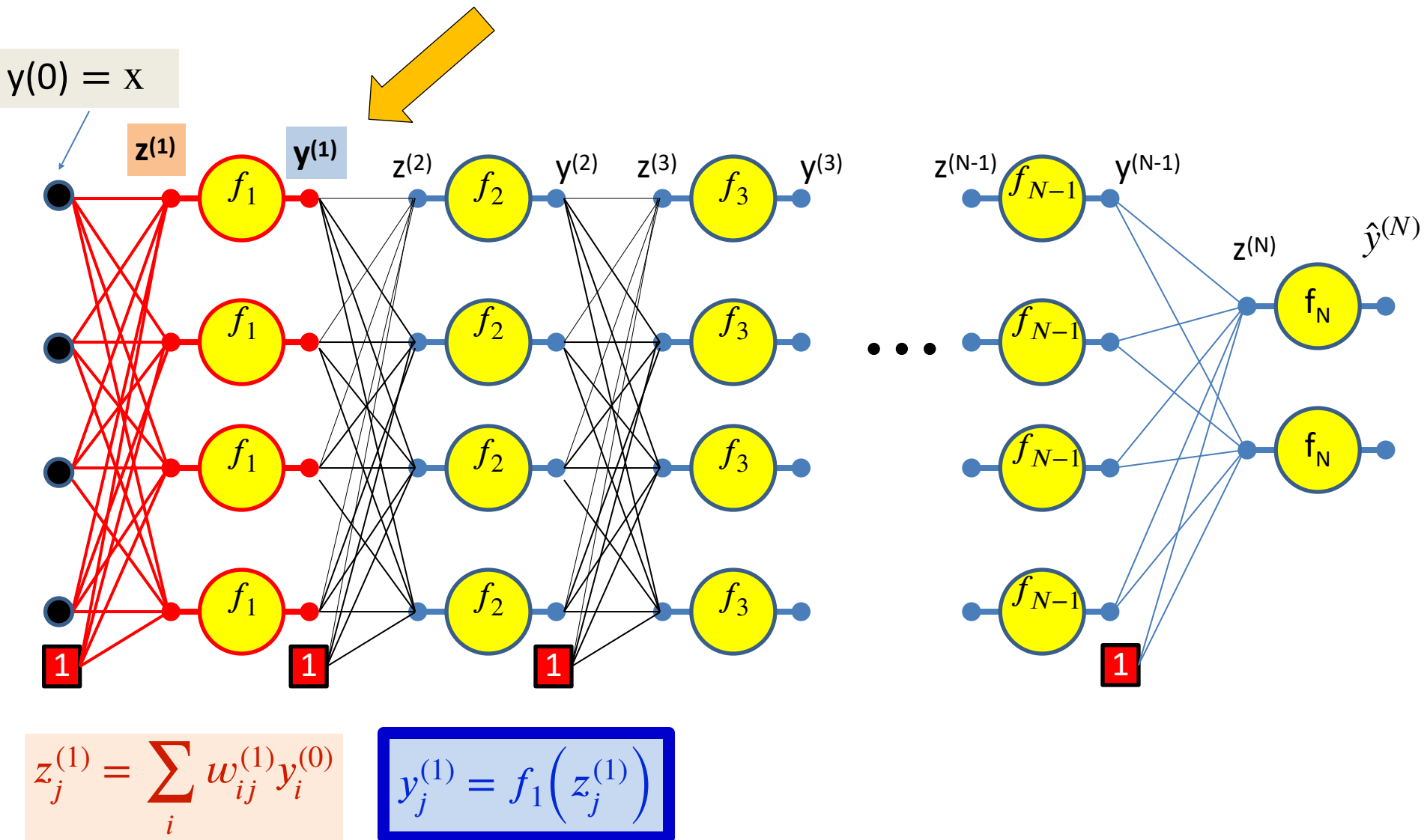
$$y(0) = x$$

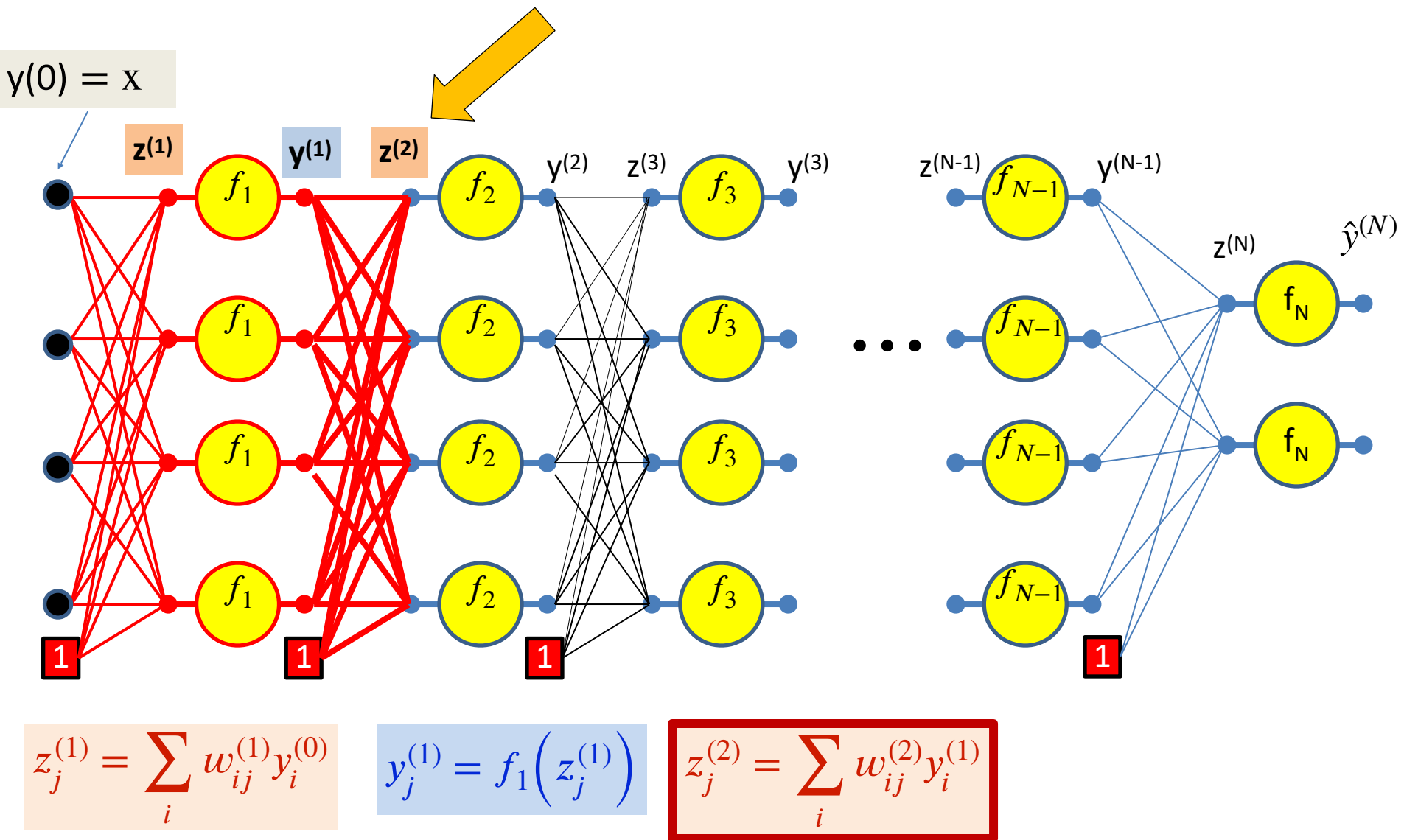


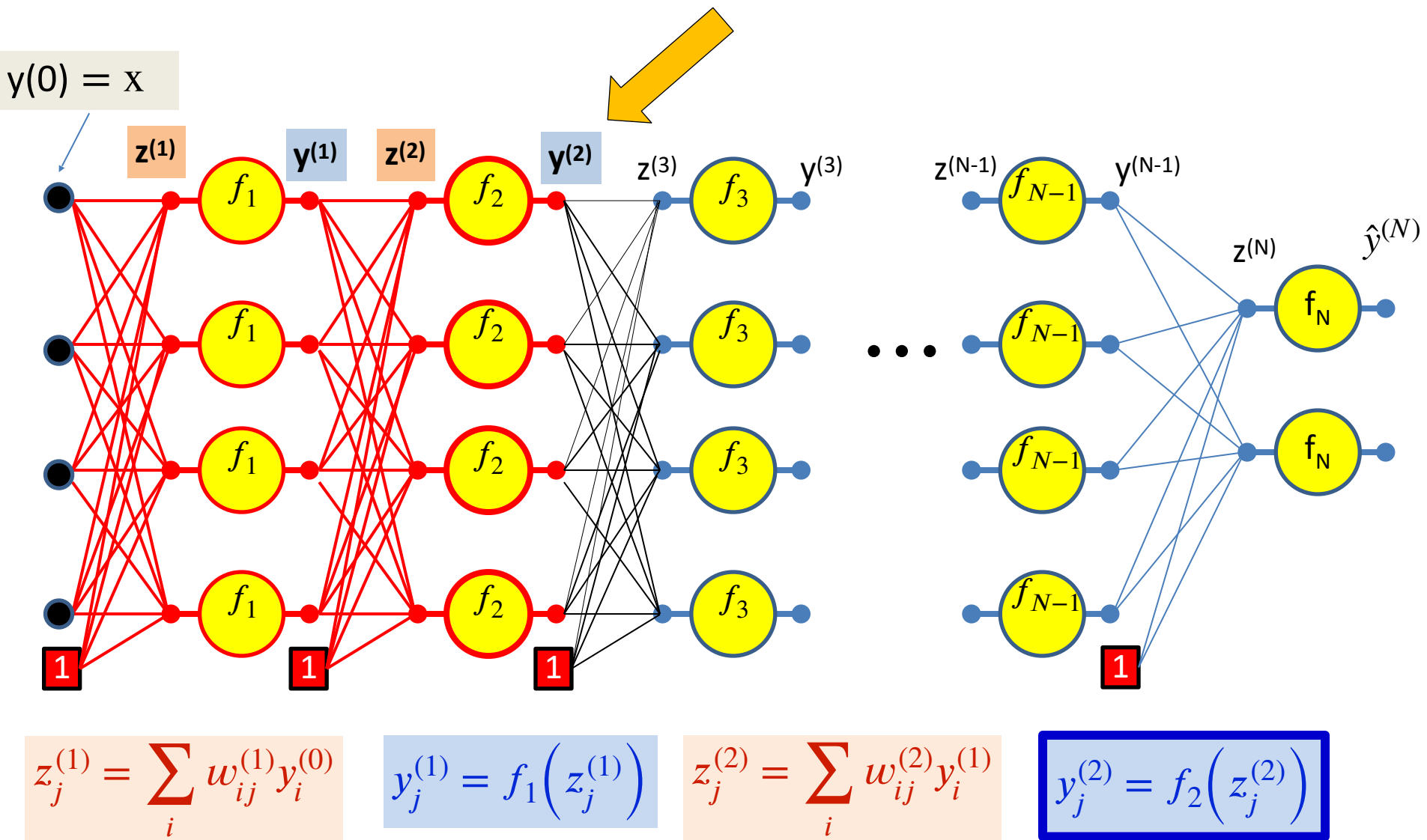
$$z_1^{(1)} = \sum_i w_{i1}^{(1)} y_i^{(0)}$$



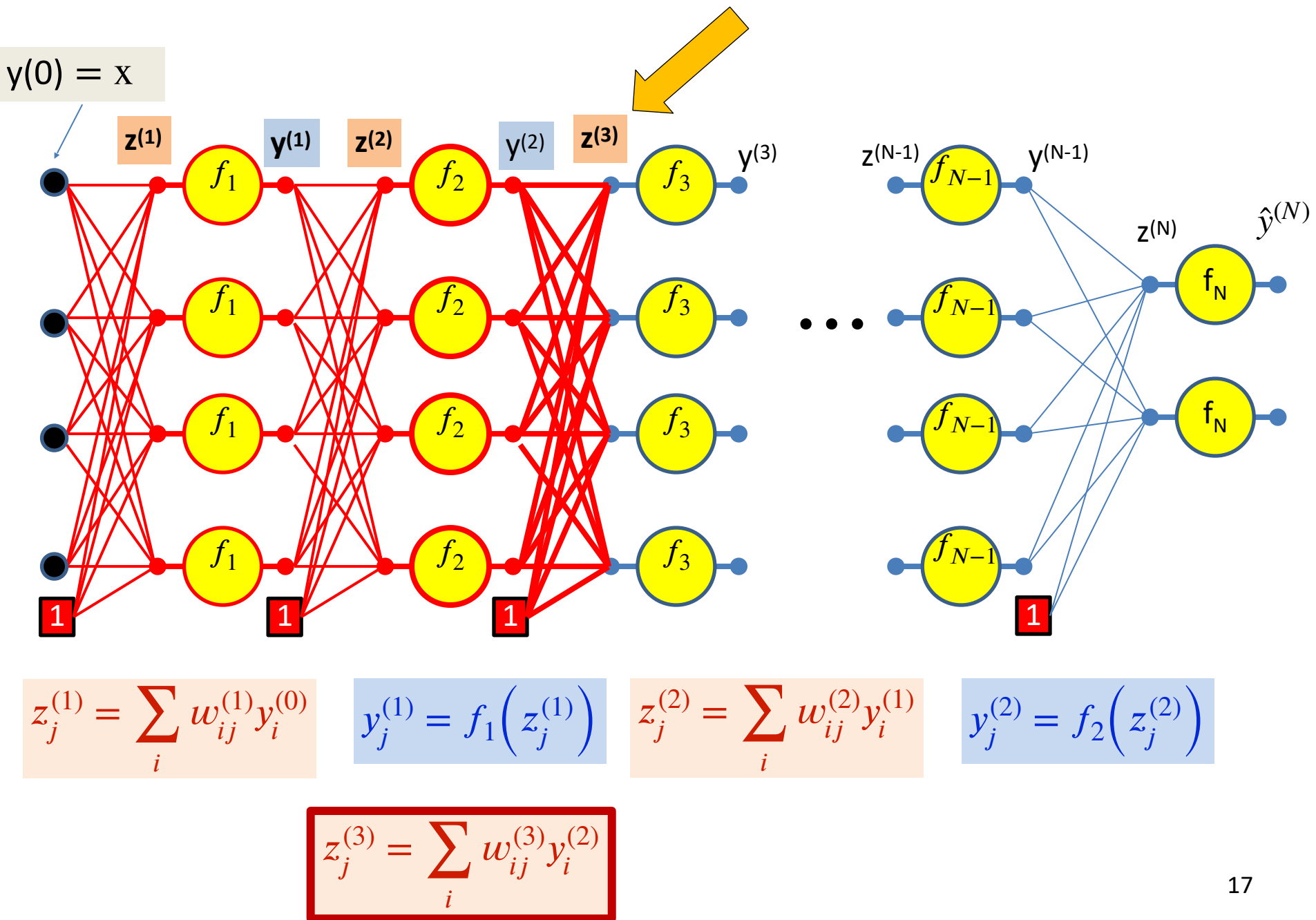
$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$



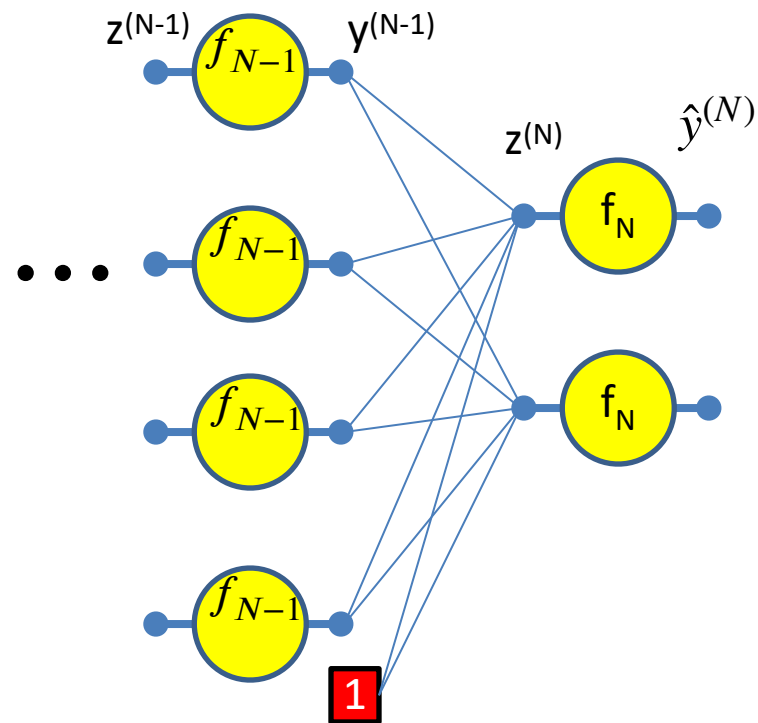
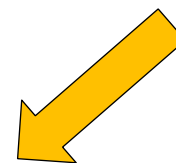
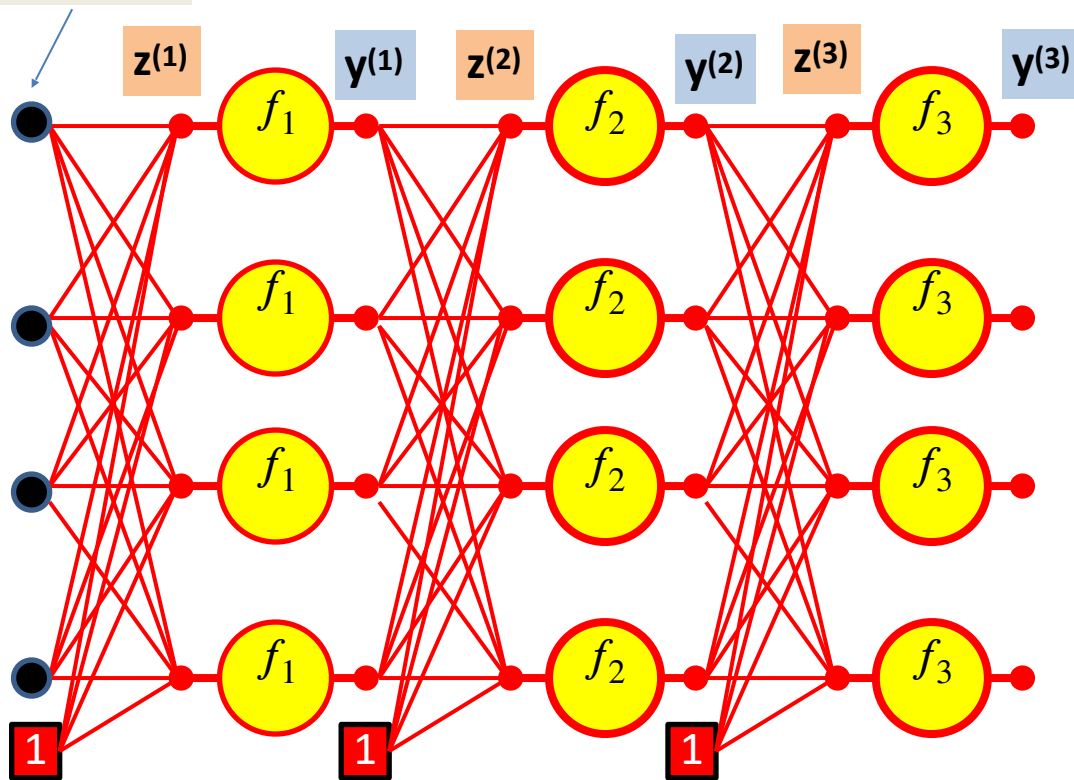








$$y(0) = x$$



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

$$y_j^{(1)} = f_1(z_j^{(1)})$$

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

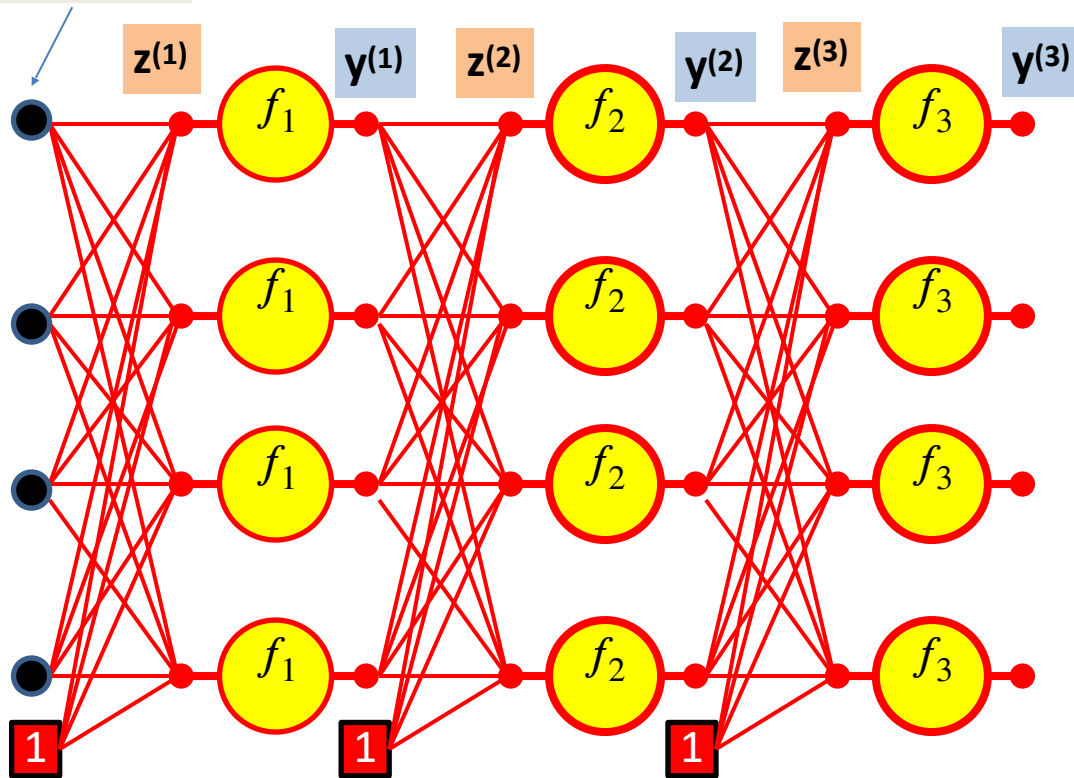
$$y_j^{(2)} = f_2(z_j^{(2)})$$

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

$$y_j^{(3)} = f_3(z_j^{(3)})$$

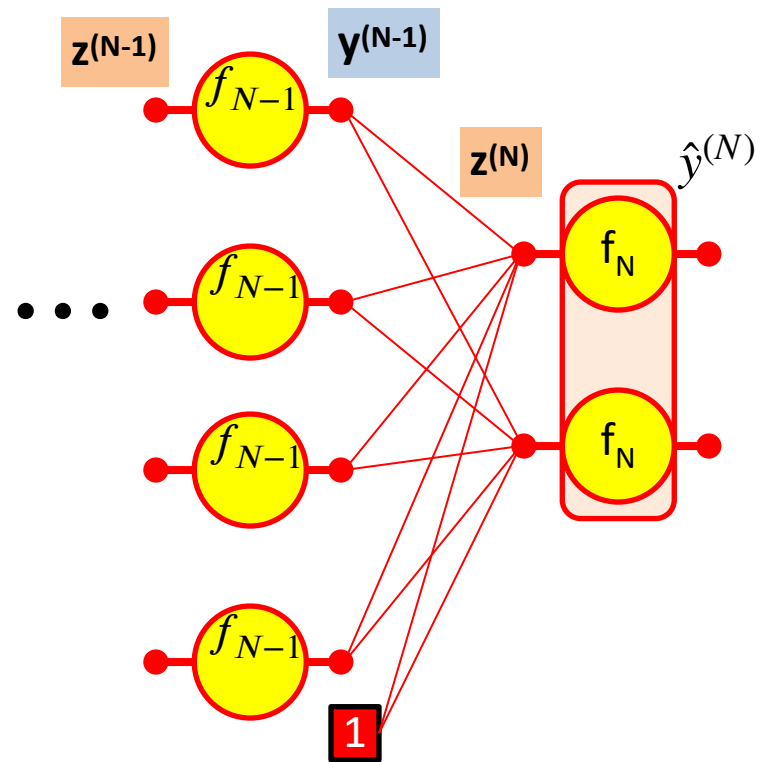
...

$$y(0) = x$$



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

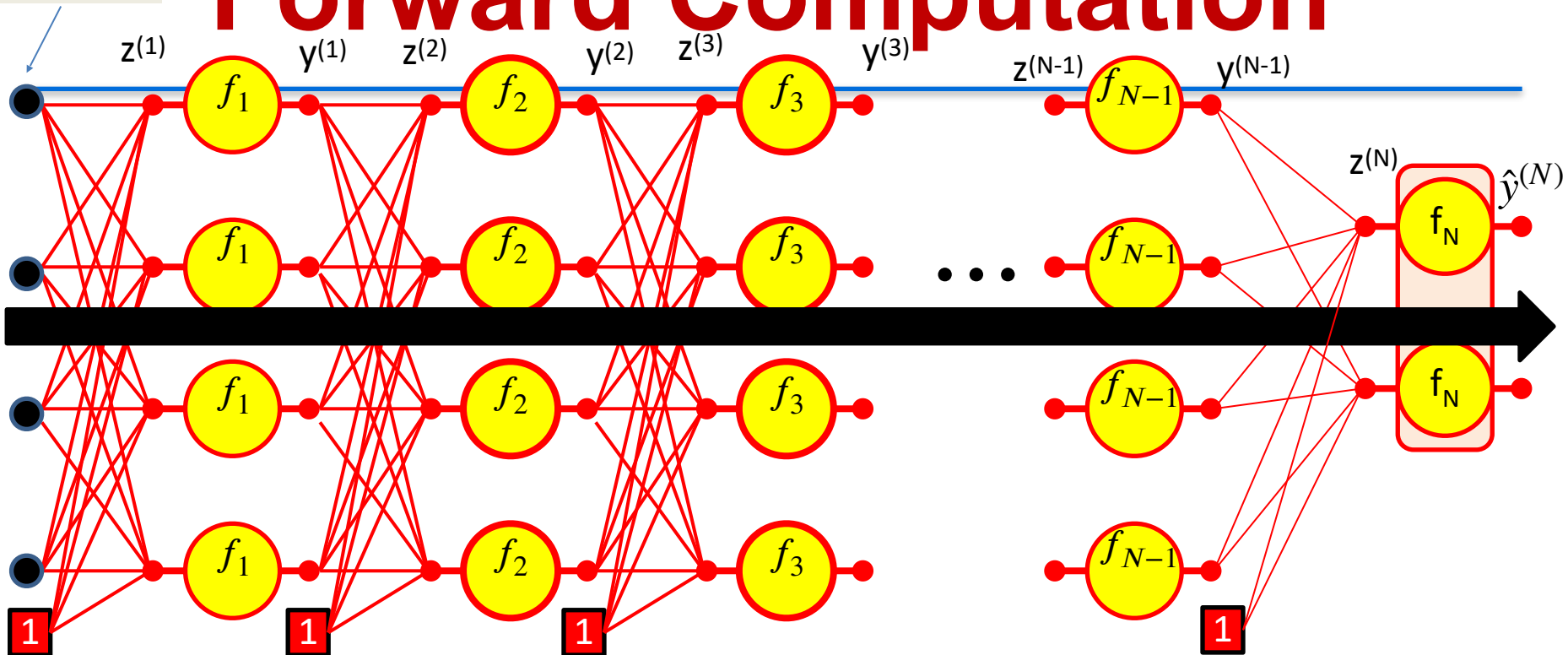
$$z_j^{(N)} = \sum_i w_{ij}^{(N)} y_i^{(N-1)}$$



$$y^{(N)} = f_N(z^{(N)})$$

$$y(0) = x$$

# Forward Computation



ITERATE FOR  $k = 1:N$

for  $j = 1:\text{layer-width}$

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

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

$$y_j^{(k)} = f_k(z_j^{(k)})$$

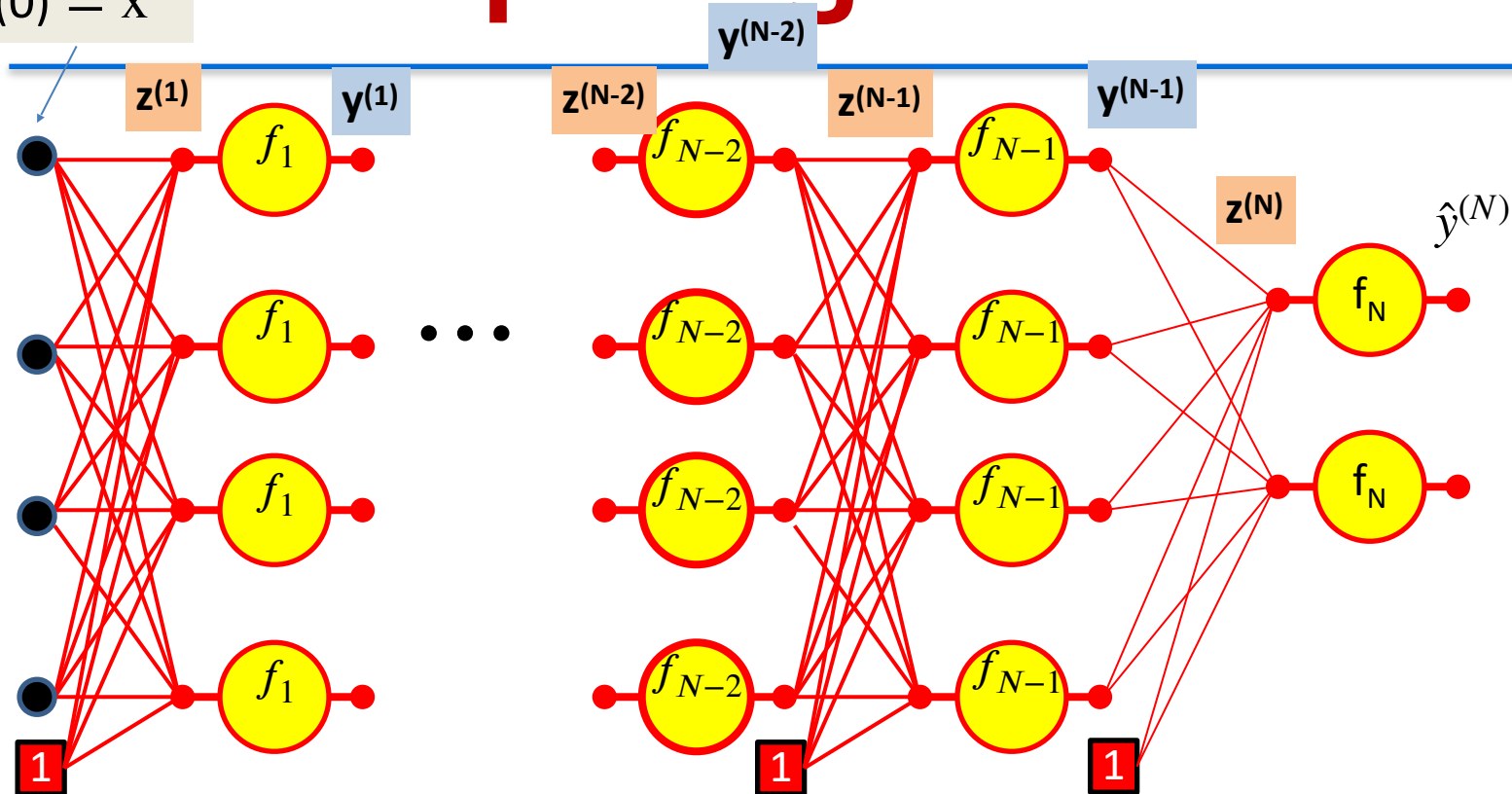
# Forward “Pass”

- Input:  $D$  dimensional vector  $\mathbf{x} = [x_j, j = 1 \dots D]$
- Set:
  - $D_0 = D$ , is the width of the 0<sup>th</sup> (input) layer
  - $y_j^{(0)} = x_j, j = 1 \dots D; \quad y_0^{(k=1 \dots N)} = x_0 = 1$
- For layer  $k = 1 \dots N$ 
  - For  $j = 1 \dots D_k$ 

$D_k$  is the size of the  $k$ th layer

$$\begin{aligned} \triangleright z_j^{(k)} &= \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)} \\ \triangleright y_j^{(k)} &= f_k(z_j^{(k)}) \end{aligned}$$
- Output:
  - $Y = y_j^{(N)}, j = 1 \dots D_N$

# Computing derivatives



We have computed all these intermediate values in the forward computation

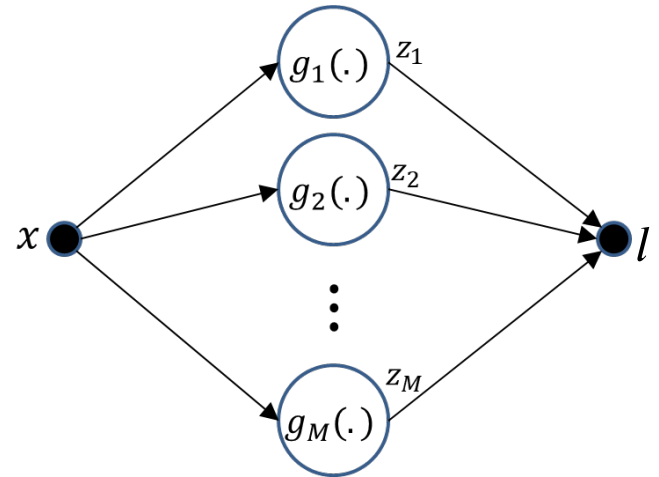
We must remember them - we will need them to compute the derivatives

# Calculus Refresher: Chain rule

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

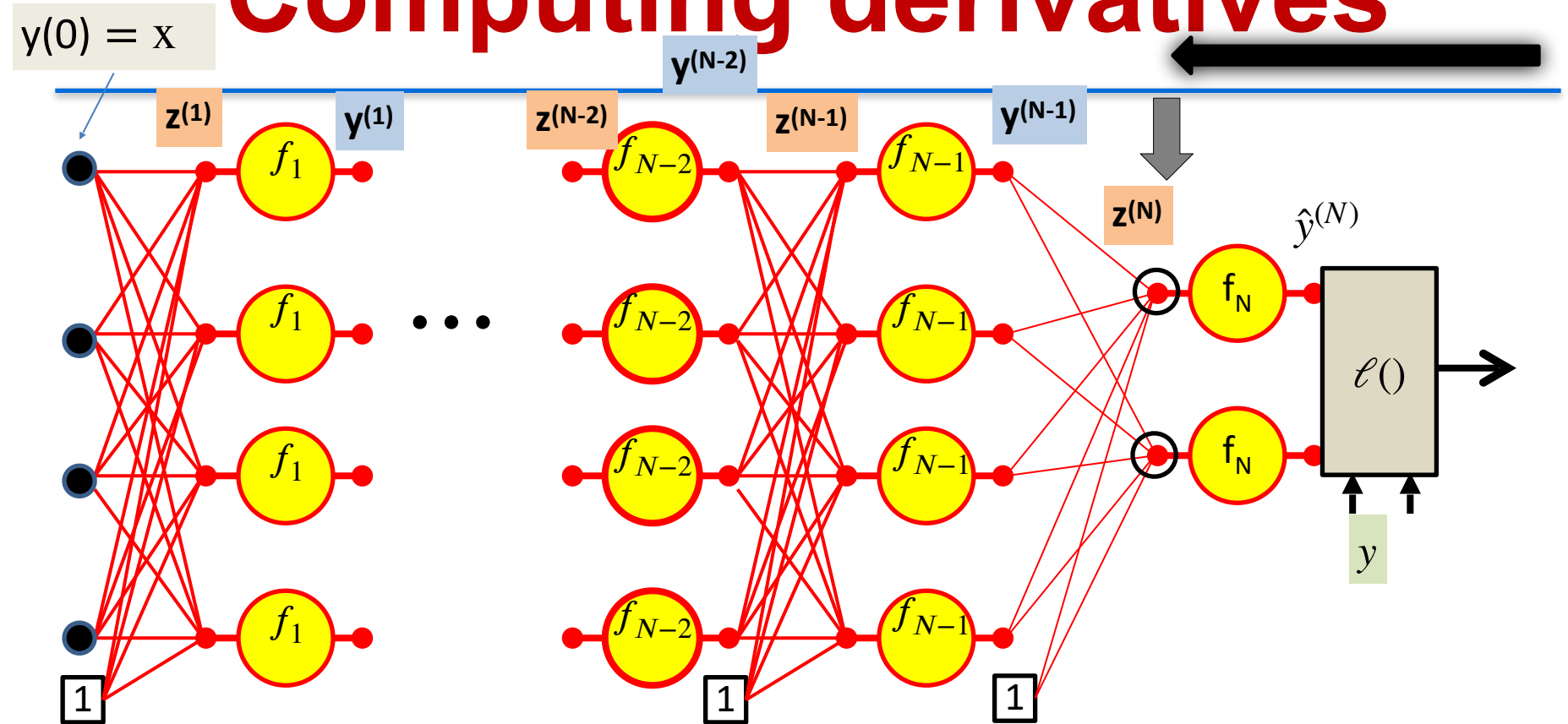
$$\frac{dl}{dz} = \frac{dl}{dy} \frac{dy}{dz}$$

For  $l = f(z_1, z_2, \dots, z_M)$   
where  $z_i = g_i(x)$



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

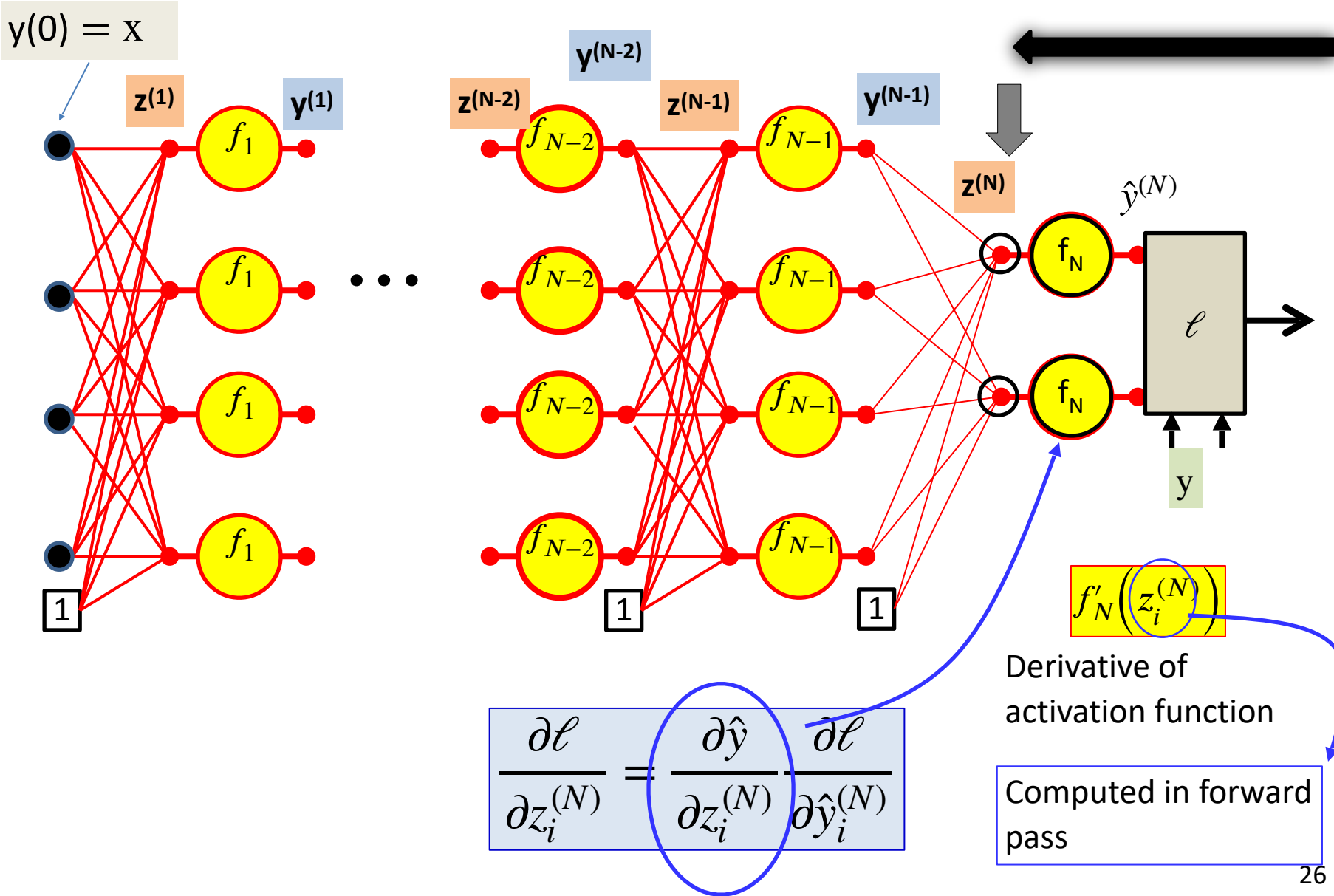
# Computing derivatives

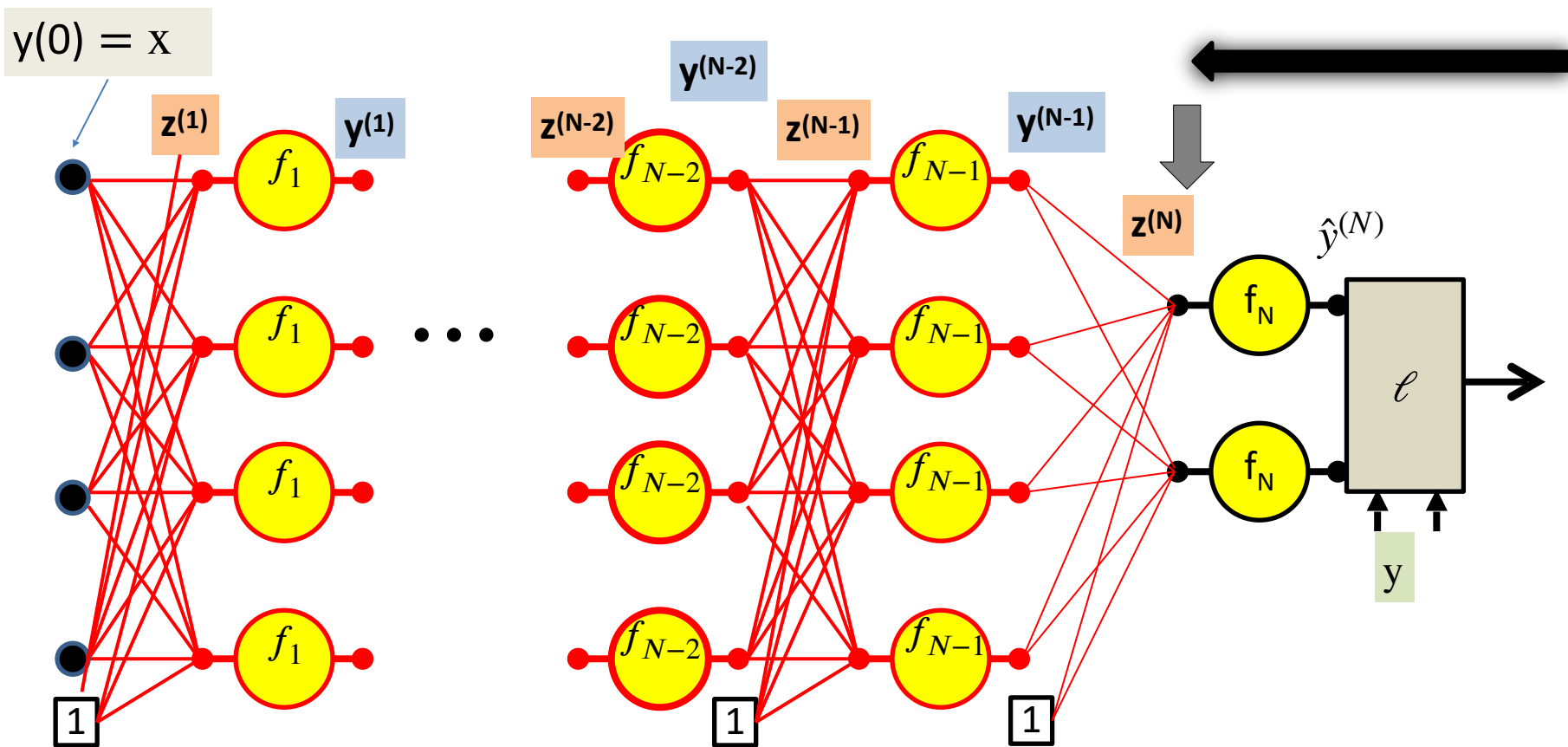


$$\frac{\partial \ell}{\partial z_i^{(N)}} = \frac{\partial \hat{y}}{\partial z_i^{(N)}} \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$

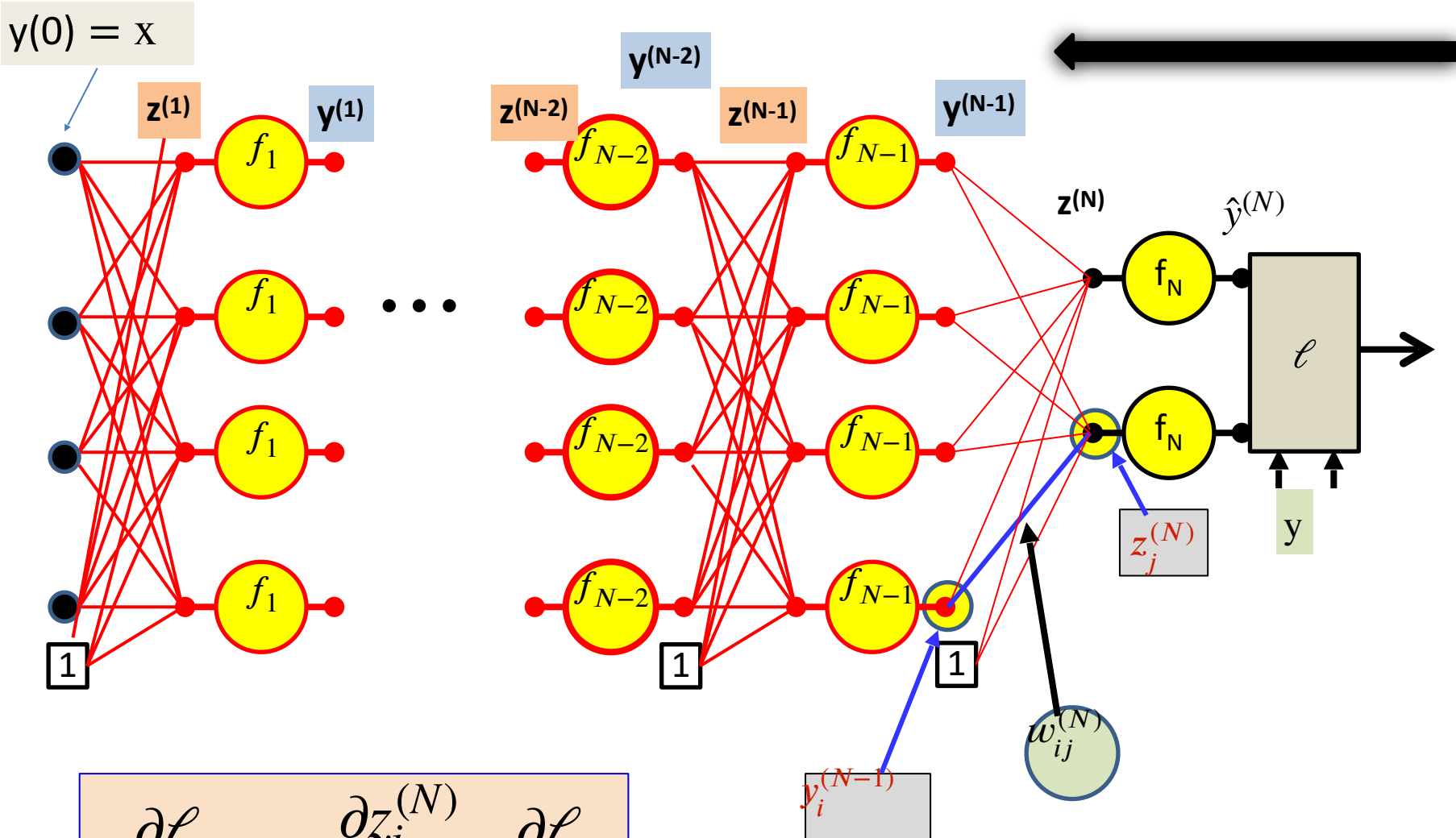




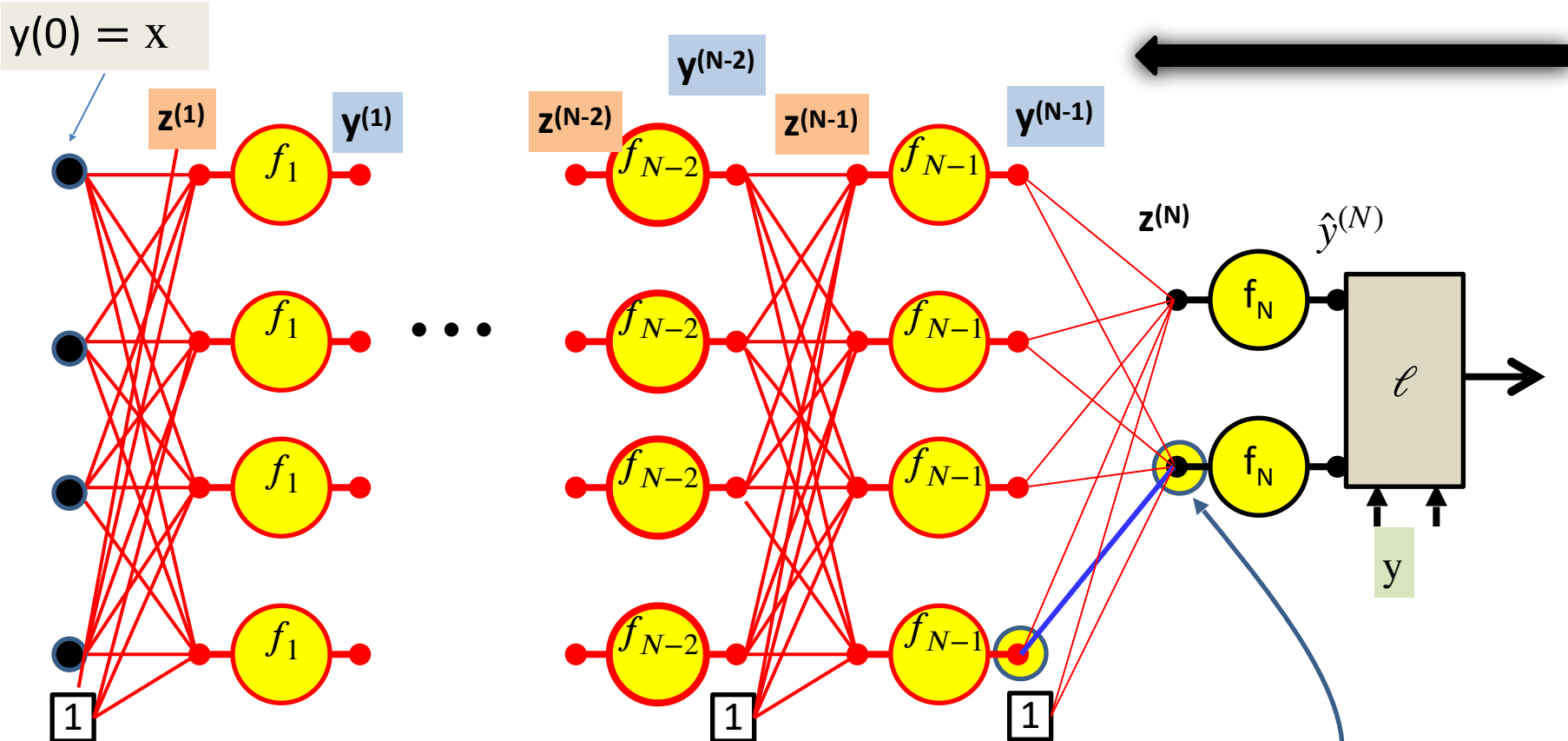




$$\frac{\partial \ell}{\partial z_i^{(N)}} = f'_N(z_i^{(N)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$

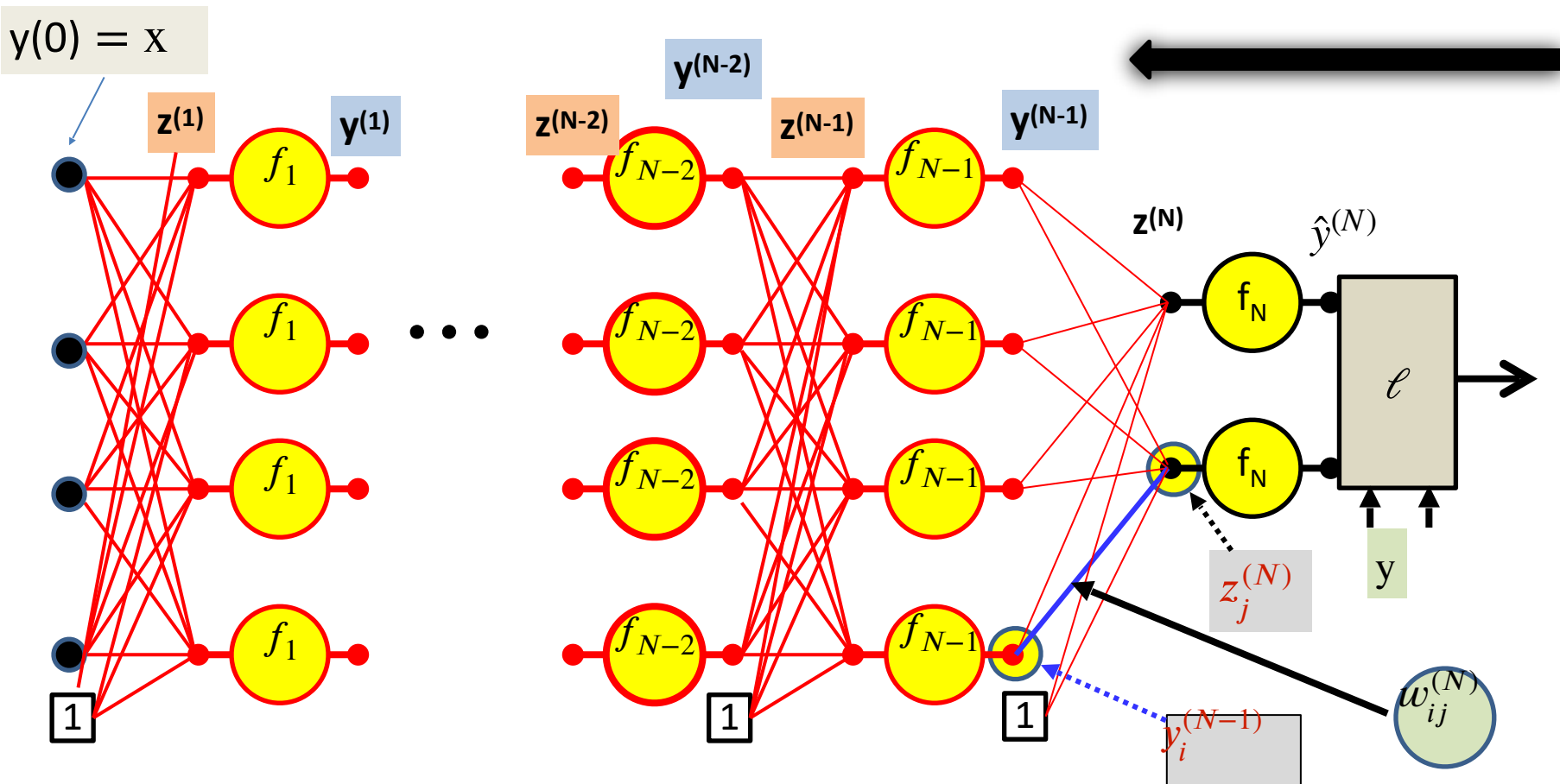


$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$



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

Just computed




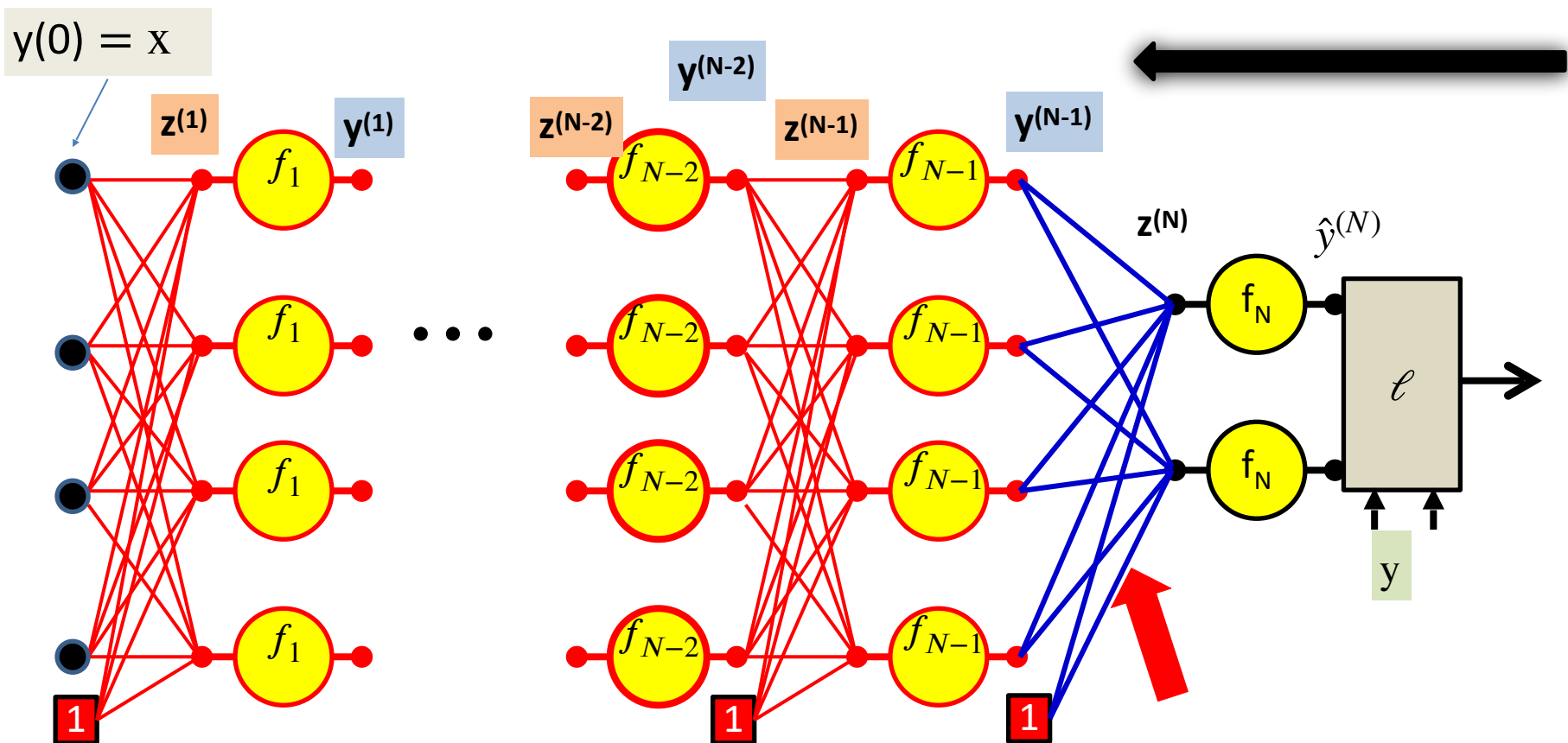
$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$

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

Computed in forward pass

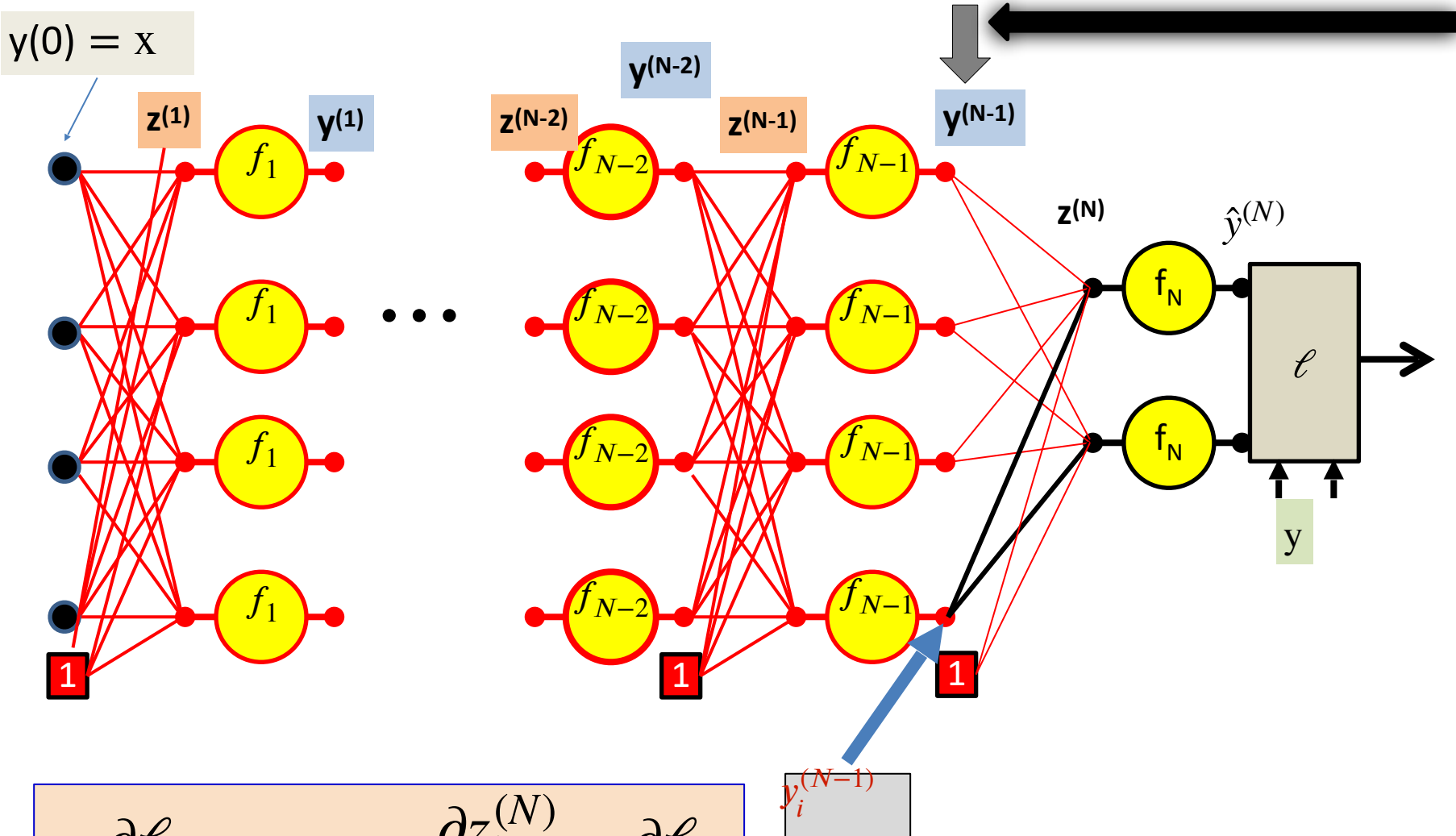




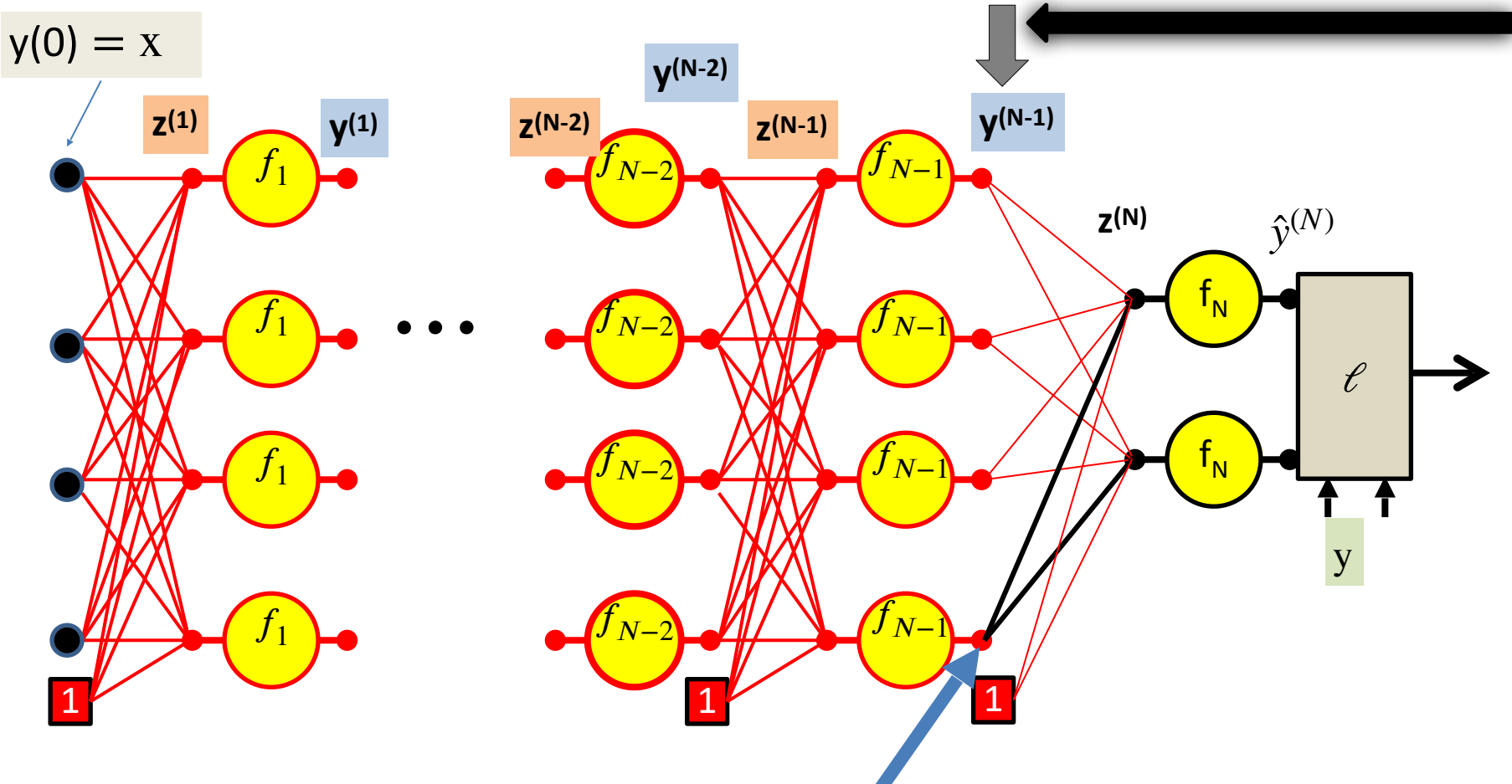


$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}} \quad \text{For the bias term } y_0^{(N-1)} = 1$$



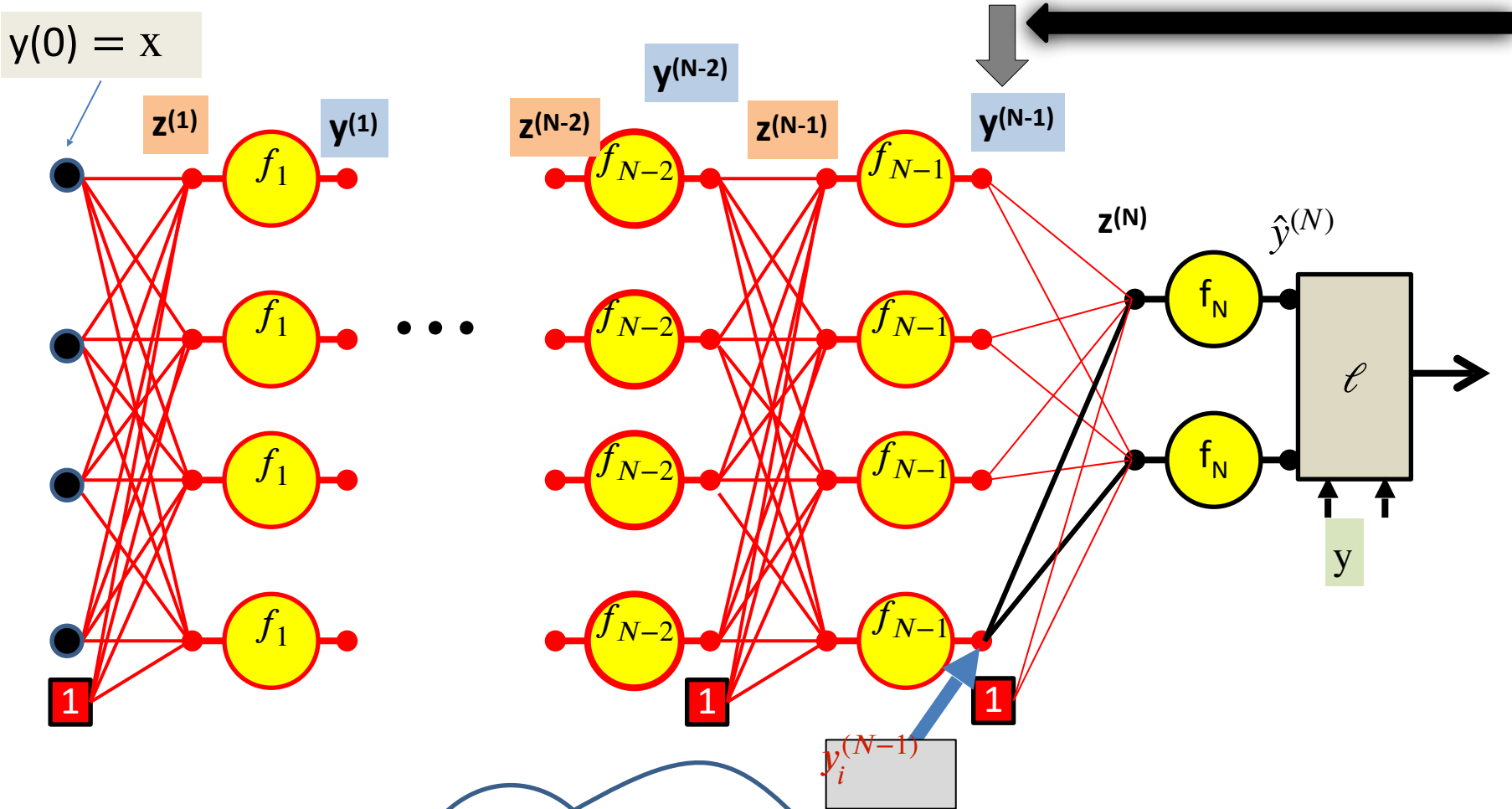


$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$



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

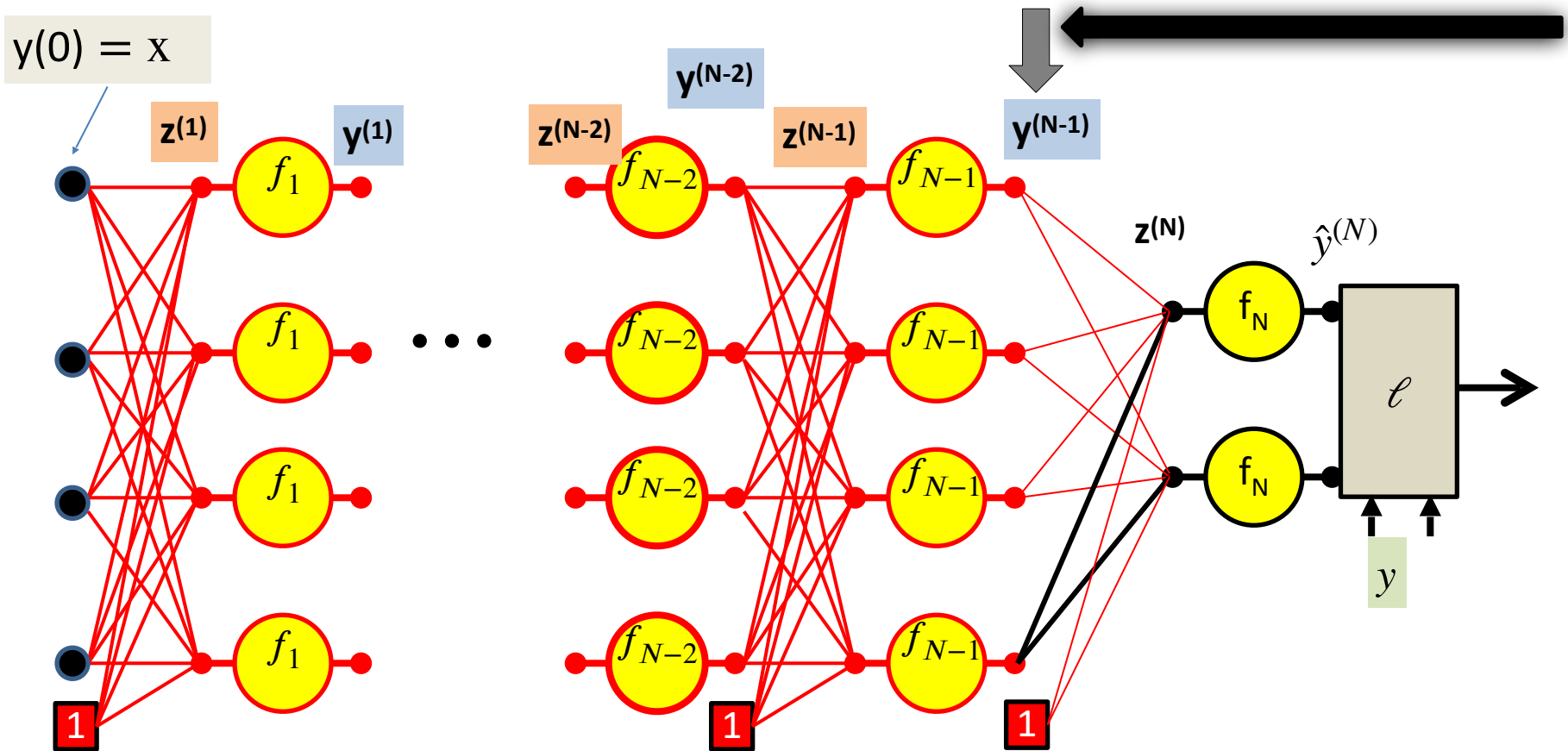
Already computed



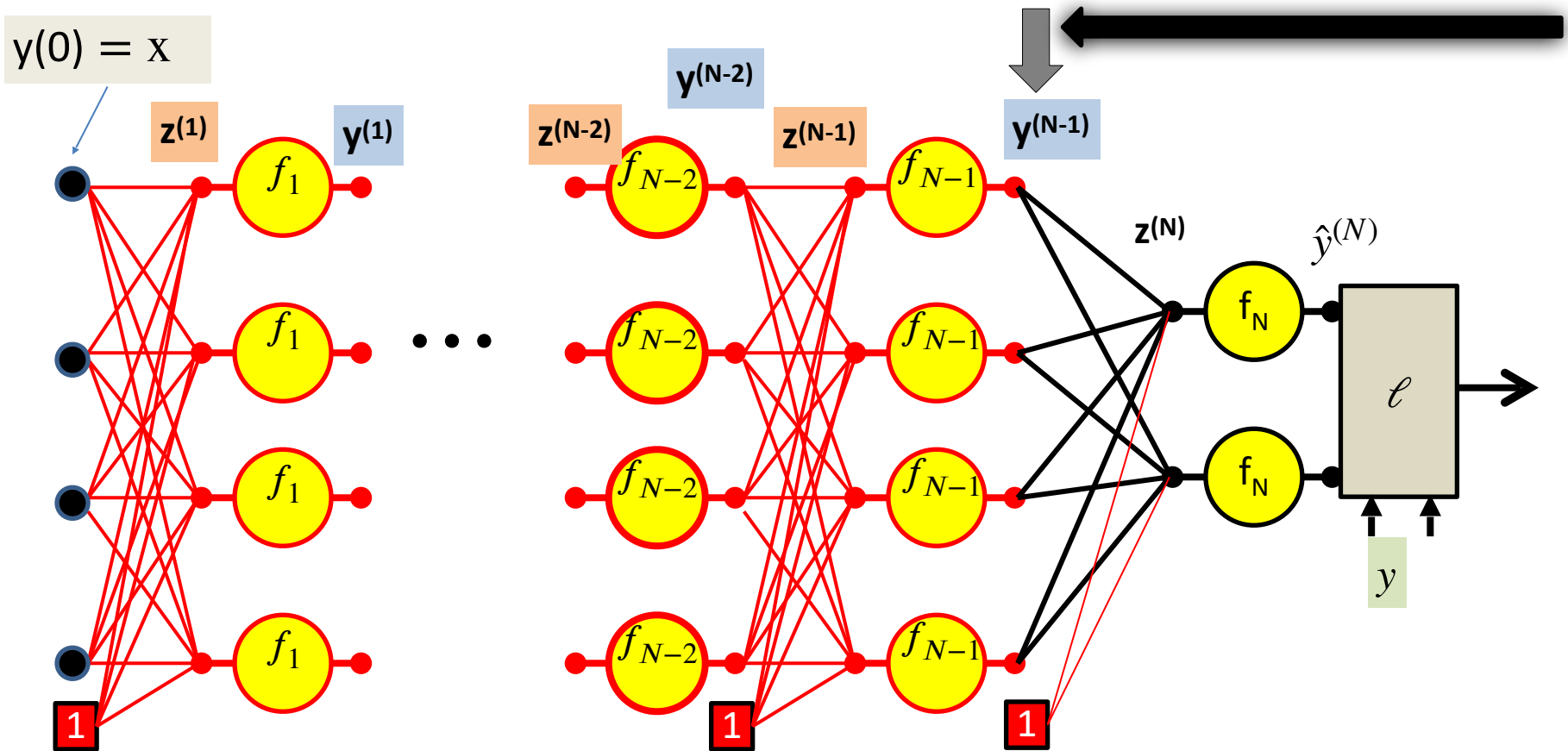
$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$   
 $w_{ij}^{(N)}$

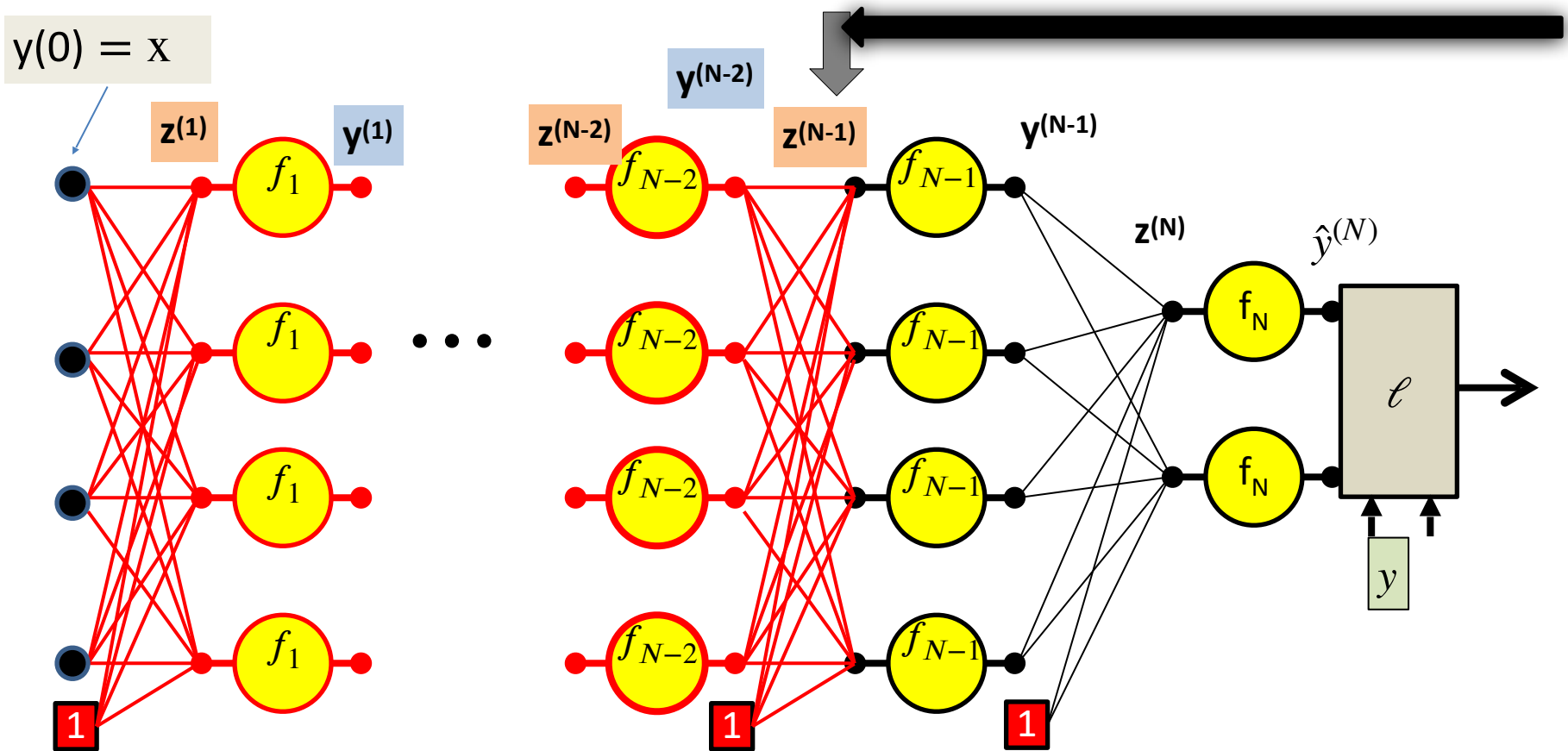
Because  $z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$



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

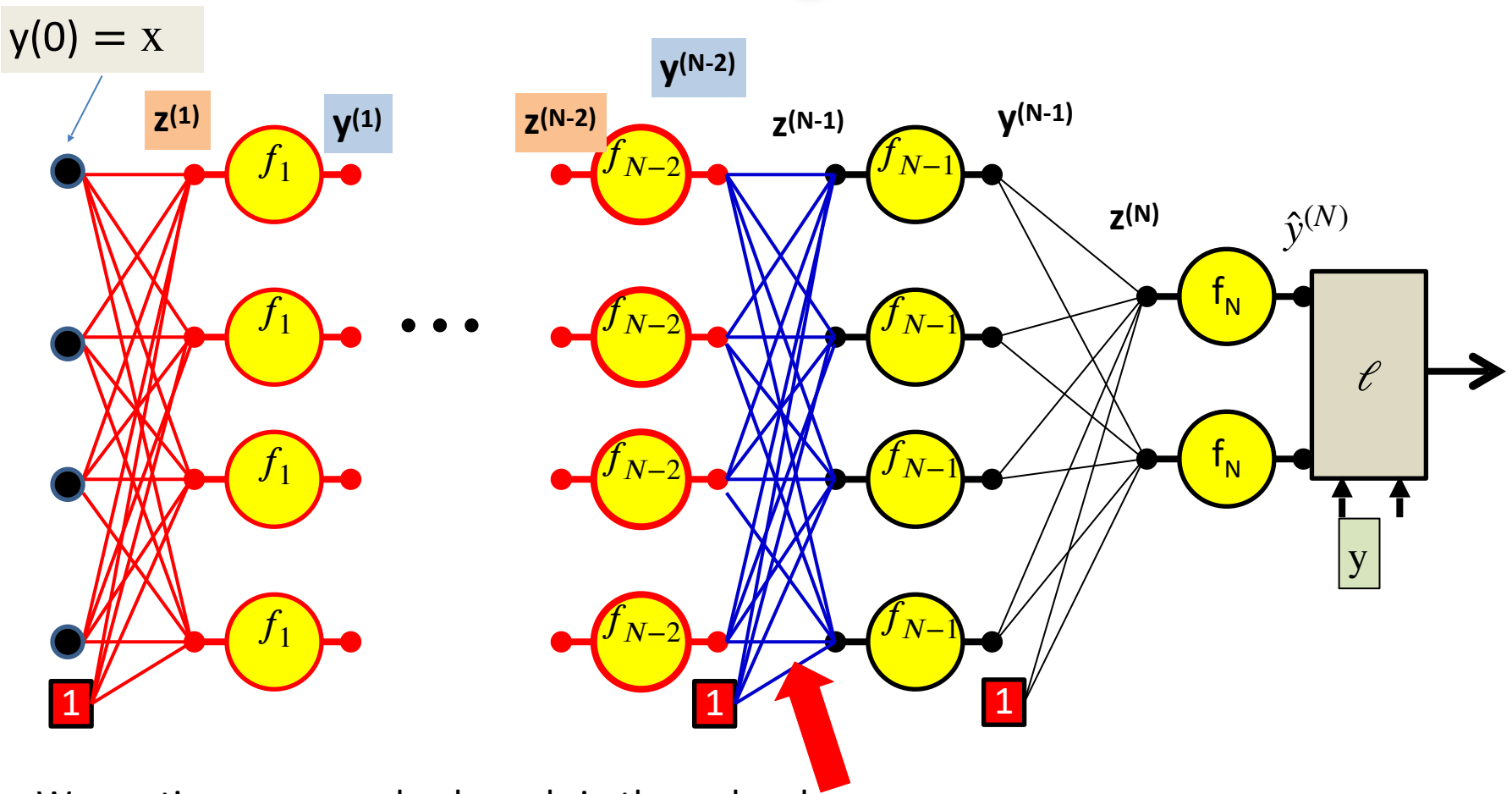


$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial \ell}{\partial z_j^{(N)}}$$



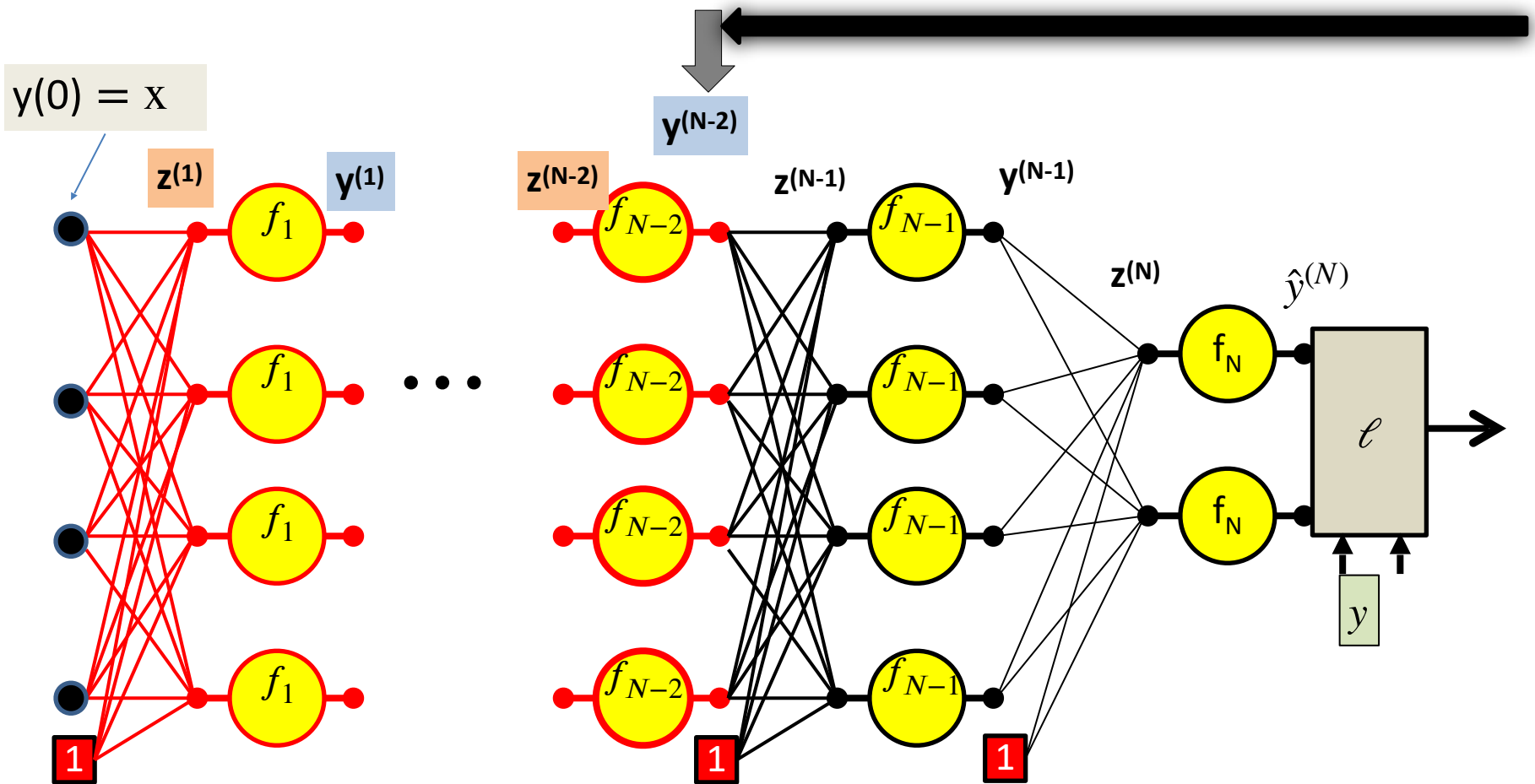
We continue our way backwards in the order shown

$$\frac{\partial \ell}{\partial z_i^{(N-1)}} = f'_{N-1}(z_i^{(N-1)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N-1)}}$$



We continue our way backwards in the order shown

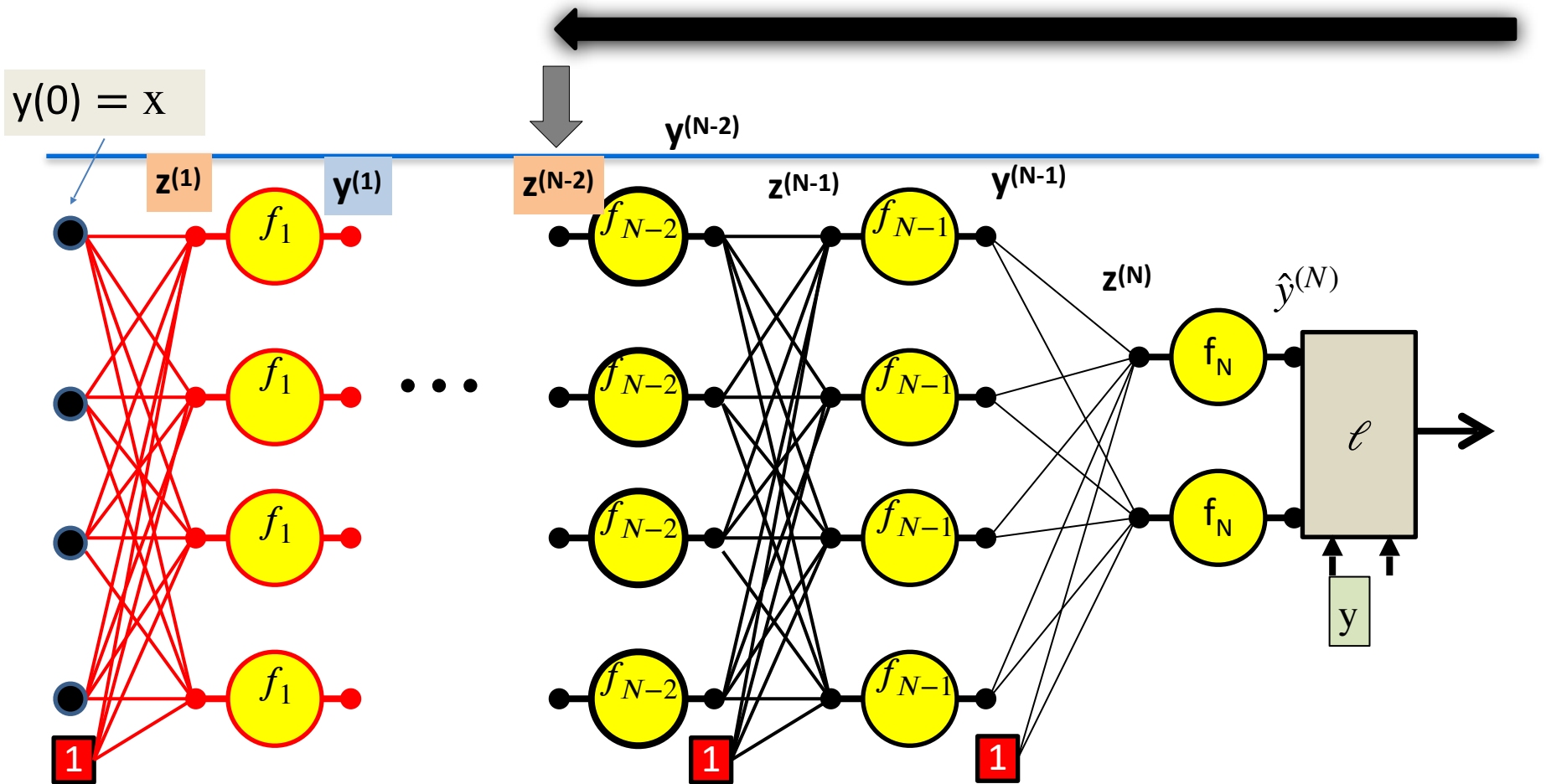
$$\frac{\partial \ell}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial \ell}{\partial z_j^{(N-1)}} \quad \text{the bias term } y_0^{(N-2)} = 1$$



We continue our way backwards in the order shown

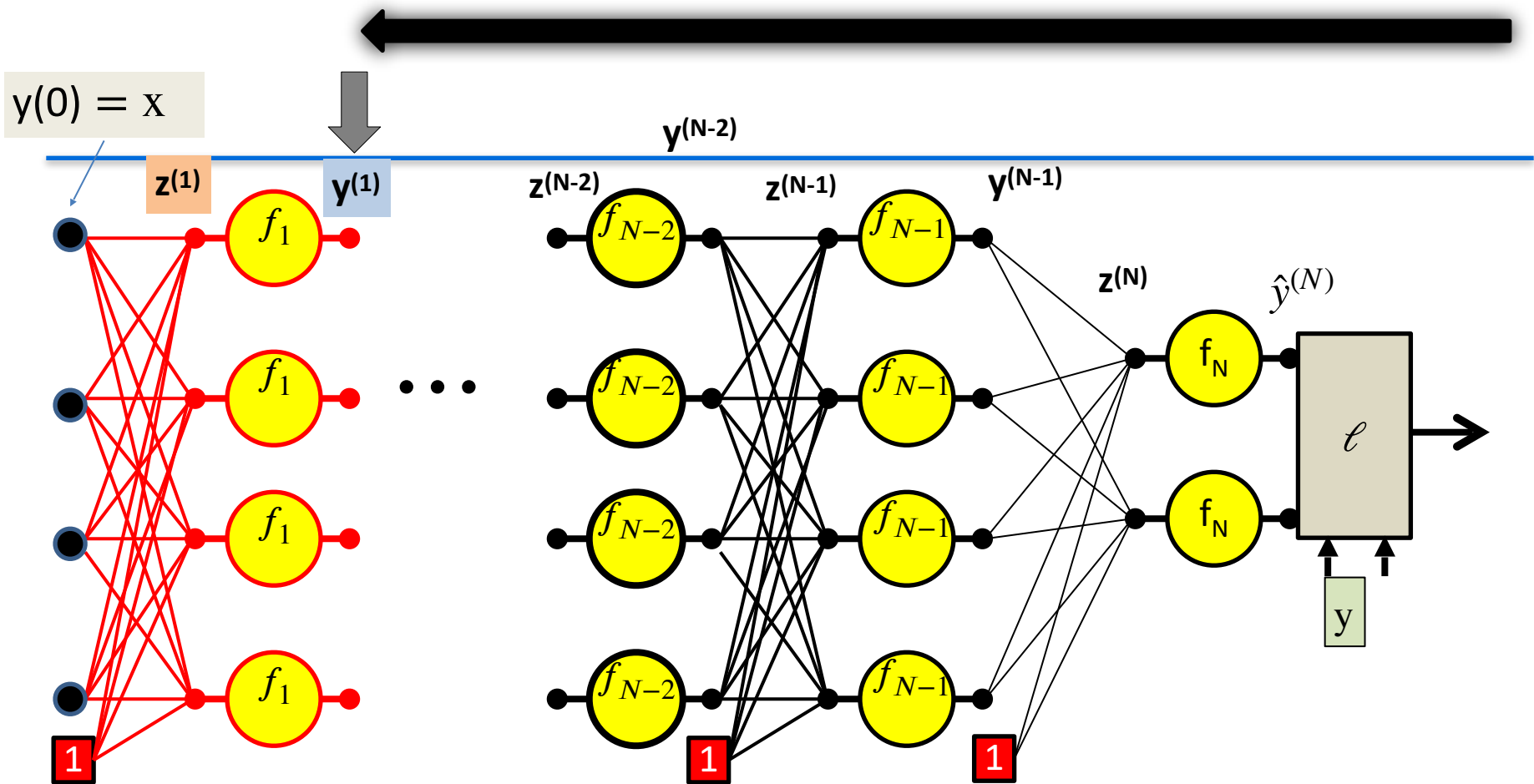
$$\frac{\partial \ell}{\partial y_i^{(N-2)}} = \sum_j w_{ij}^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N-1)}}$$





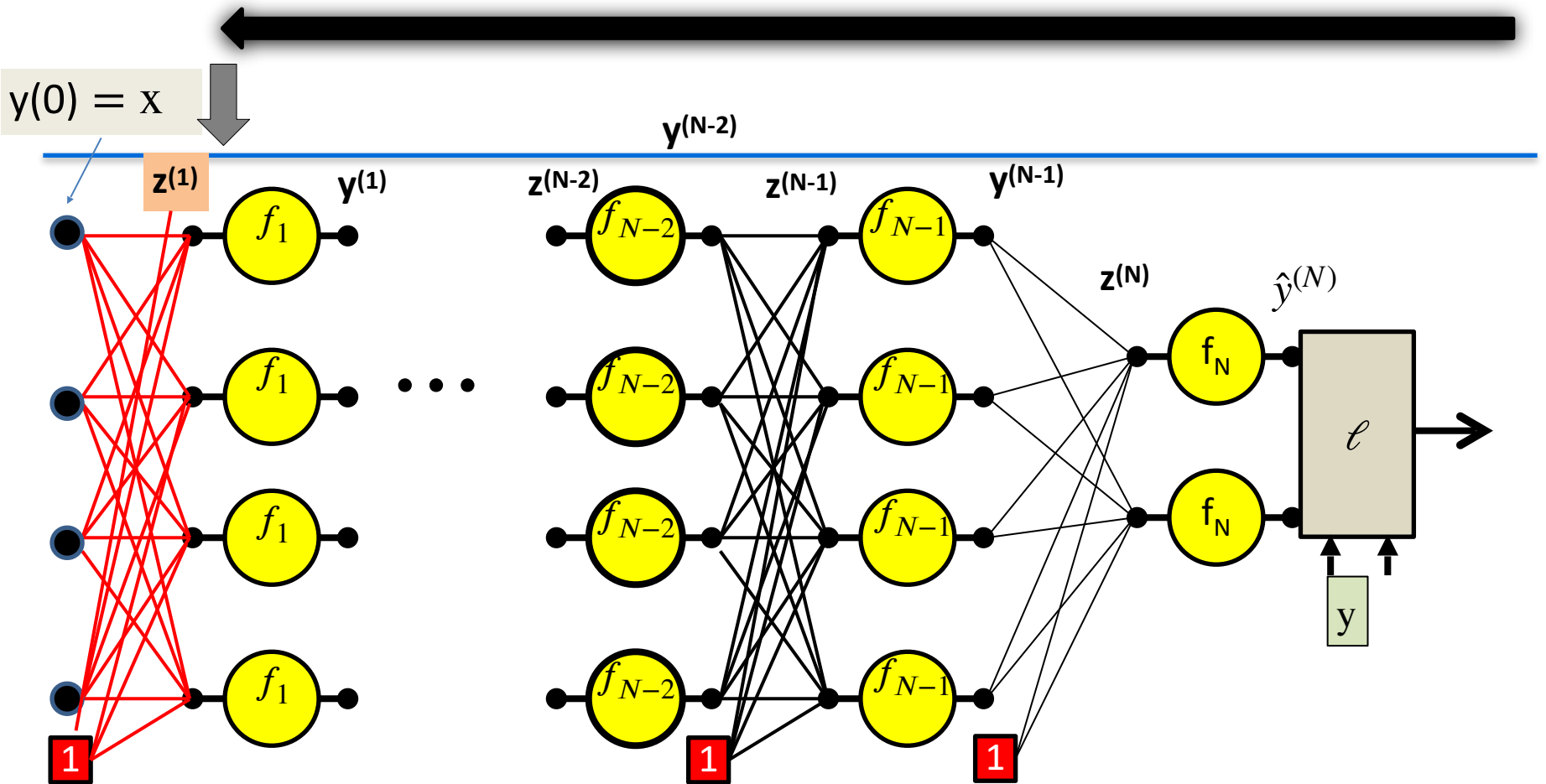
We continue our way backwards in the order shown

$$\frac{\partial \ell}{\partial z_i^{(N-2)}} = f'_{N-2}(z_i^{(N-2)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N-2)}}$$



We continue our way backwards in the order shown

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

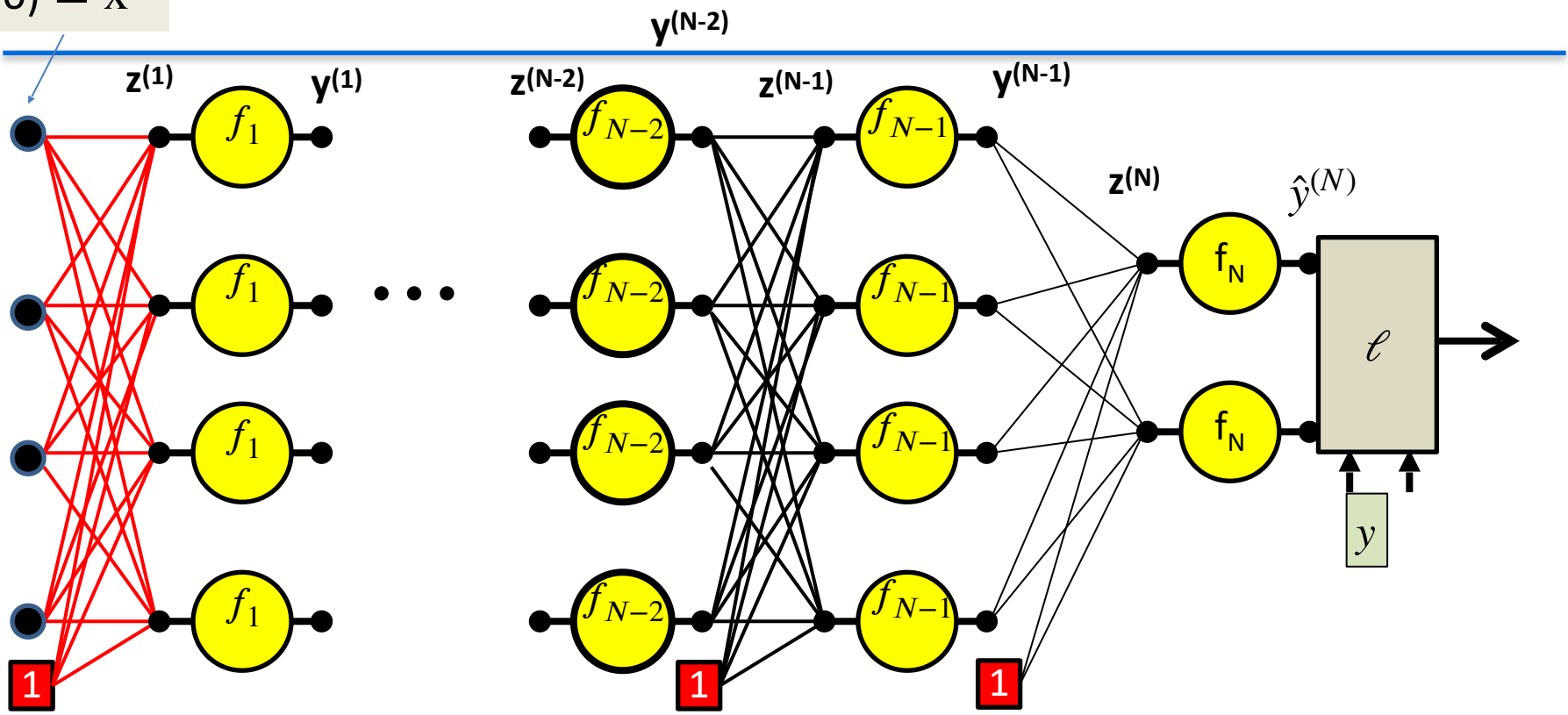


We continue our way backwards in the order shown

$$\frac{\partial \ell}{\partial z_i^{(1)}} = f_1'(z_i^{(1)}) \frac{\partial \ell}{\partial \hat{y}_i^{(1)}}$$



$$y(0) = x$$



We continue our way backwards in the order shown

$$\frac{\partial \ell}{\partial w_{ij}^{(1)}} = x_i \frac{\partial \ell}{\partial z_j^{(1)}}$$

# Backward Pass

- Output layer ( $N$ ) :
  - For  $i = 1 \dots D_N$ 
    - $\frac{\partial \ell}{\partial z_i^{(N)}} = f'_N(z_i^{(N)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$
    - $\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}}$  for each  $j$
- For layer  $k = N - 1$  *downto* 1
  - For  $i = 1 \dots D_k$

Called “**Backpropagation**” because the derivative of the loss is propagated “backwards” through the network

Very analogous to the forward pass:

$$\begin{aligned}
 \bullet \quad \frac{\partial \ell}{\partial y_i^{(k-1)}} &= \sum_j w_{ij}^{(k)} \frac{\partial \ell}{\partial z_j^{(k)}} \\
 \bullet \quad \frac{\partial \ell}{\partial z_i^{(k)}} &= f'_k(z_i^{(k)}) \frac{\partial \ell}{\partial y_i^{(k)}} \\
 \bullet \quad \frac{\partial \ell}{\partial w_{ij}^{(k)}} &= y_i^{(k-1)} \frac{\partial \ell}{\partial z_j^{(k)}} \text{ for each } j
 \end{aligned}$$

Backward weighted combination of next layer

Backward equivalent of activation

# Recap

---

- Compute the gradient through Back-propagation algorithm
  - with forward pass and backward pass
  - backward pass is application of chain rule

# Example

---

- `simple_model.html`

# Autograd

---

- No need to write forward and backward explicitly
- Only need to specify the network
- Supported in pytorch and tensorflow



# FFN in Pytorch

```
import torch
import math

x = ...
y = ...

model = torch.nn.Sequential(
    torch.nn.Linear(2, 3),
    torch.nn.ReLU(),
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)

loss_fn =
torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer =
torch.optim.SGD(model.parameters()
, lr=learning_rate)
for t in range(2000):

    # Forward pass
    y_pred = model(xx)
```

```
loss = loss_fn(y_pred, y)
if t % 100 == 99:
    print(t, loss.item())
model.zero_grad()
```

```
# Backward pass
loss.backward()
```

```
# Update the weights using
stochastic gradient descent.
optimizer.step()
```

```
# You can access the first layer
linear_layer = model[0]
```

```
# For linear layer, its parameters
are stored as `weight` and `bias`.
print(f'Result: y =
{linear_layer.bias.item()} +
{linear_layer.weight[:, 0].item()}
x + {linear_layer.weight[:,
1].item()} x^2 +
{linear_layer.weight[:, 2].item()}
x^3')
```

# Recitation

---

- 1/27
- Danqing will give a hands-on tutorial on pytorch
- You will need it for Machine Problems.

# Next Up

---

- More on optimization
- Training/testing procedure
- Generalization problem
- Regularization tricks