Lecture 5
Feedforward Neural Network

Lei Li and Yuxiang Wang
UCSB

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

Project Ideas

- https://docs.google.com/document/d/1evwPACHg96UKzOSdNQ629cYl-RUzIIPnU9uSJl_YiNw/
- Proposal due today:
  - 1 page
  - your team members
  - What problem you will do
  - Why is it important
  - Rough exploration direction
  - How will you evaluate (dataset, metric)
    - if you are reading/interpreting a classic paper, try to apply to a new dataset
- You are encouraged to discuss your project at any of our office hours

Scribing Notes

- Volunteers to scribe the lecture notes, and type it in Latex
- Earn 10% bonus points for each scribed note

Recap

- General framework to formulate a learning task is through empirical risk minimization (ERM)
- Risk Bound for general bounded loss functions

$$R(\hat{h}) - R(h^*) = O\left(\sqrt{\frac{\log|\mathcal{H}| + \log(1/\delta)}{n}}\right)$$

  – using Hoeffding's inequality and union bounds
- Model selection, cross validation
- Optimization algorithm:
  – Stochastic Gradient Descent

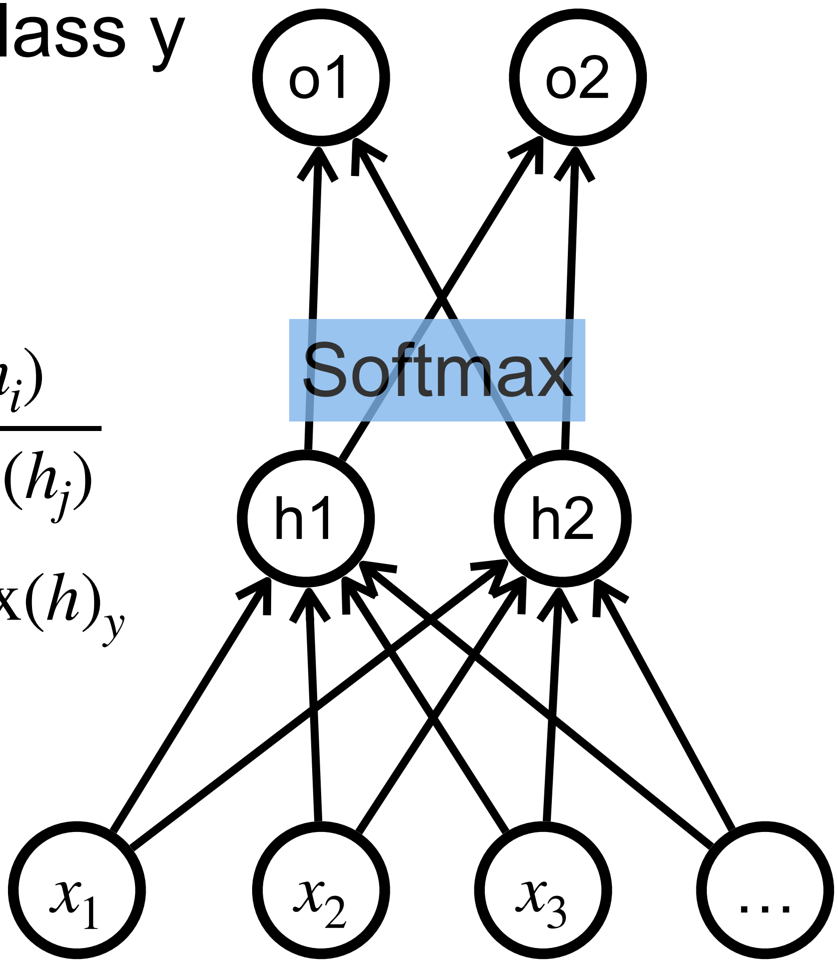
Logistic Regression

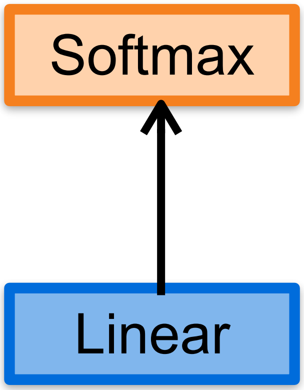
output: prob. of class y

$$h = \mathbf{W} \cdot \mathbf{x}$$

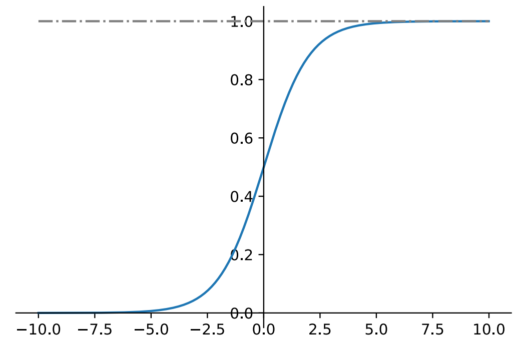
$$\text{softmax}(h)_i = \frac{\exp(h_i)}{\sum_j \exp(h_j)}$$

$$p(y \mid h) = \text{softmax}(h)_y$$

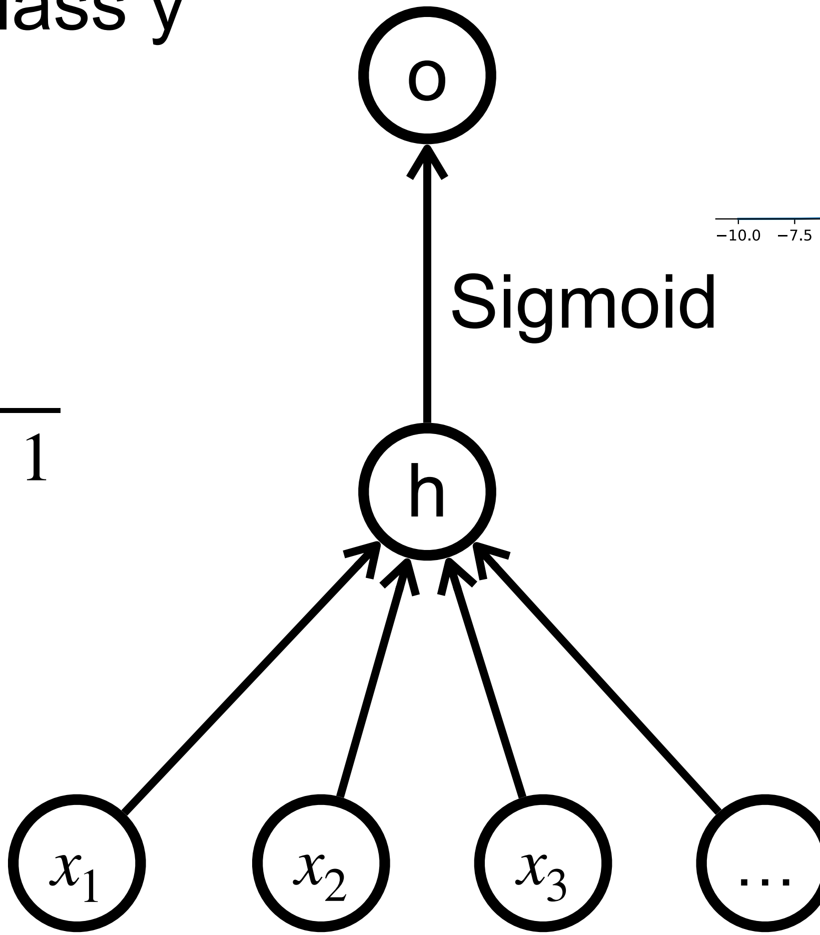




Logistic Regression for Binary Classification

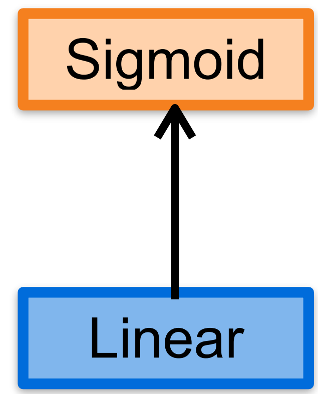


output: prob. of class y



$$h = \mathbf{w} \cdot \mathbf{x}$$

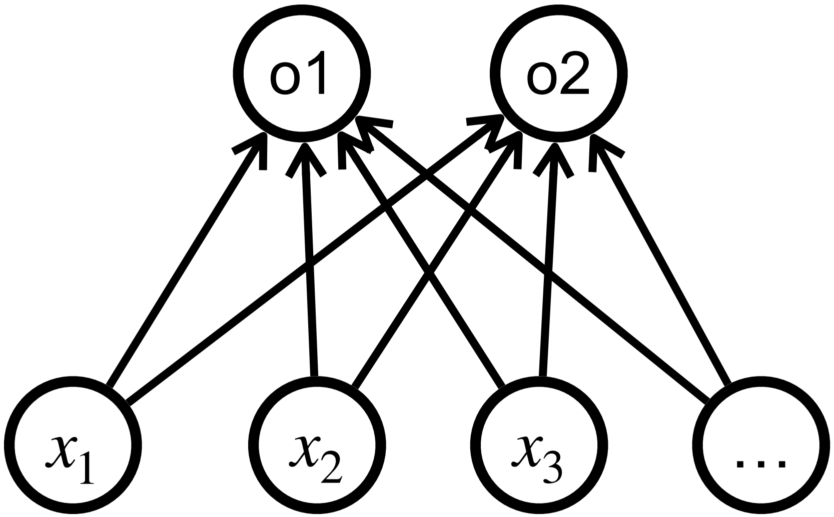
$$p(y \mid h) = \sigma(h) = \frac{e^h}{e^h + 1}$$



Cross-Entropy Loss for Classification

$$\boxed{\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} -\log f(x_n)_{y_n}}$$

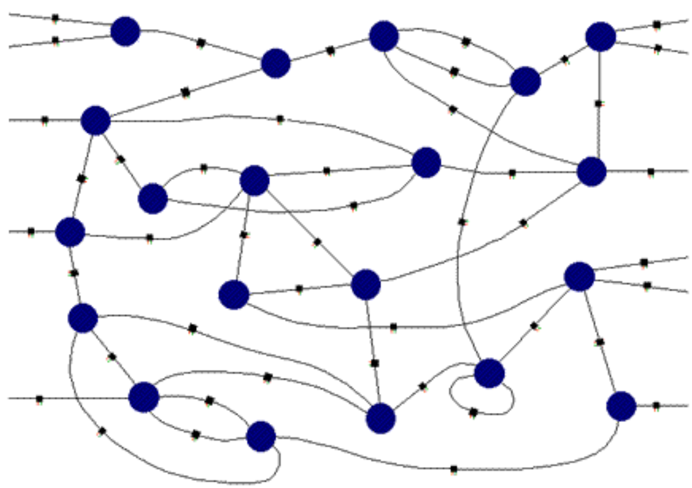
Limitation of Logistic Regression

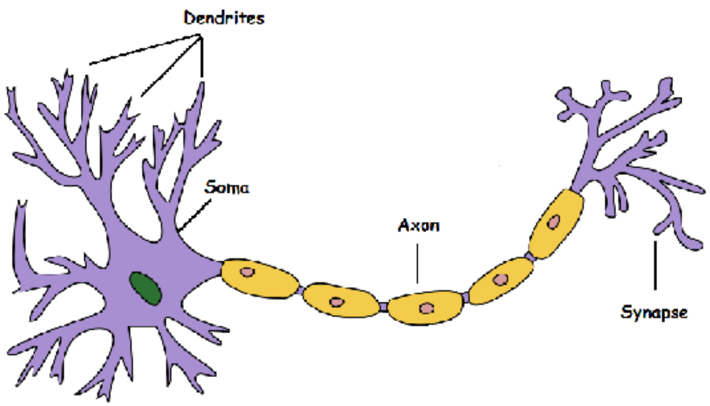


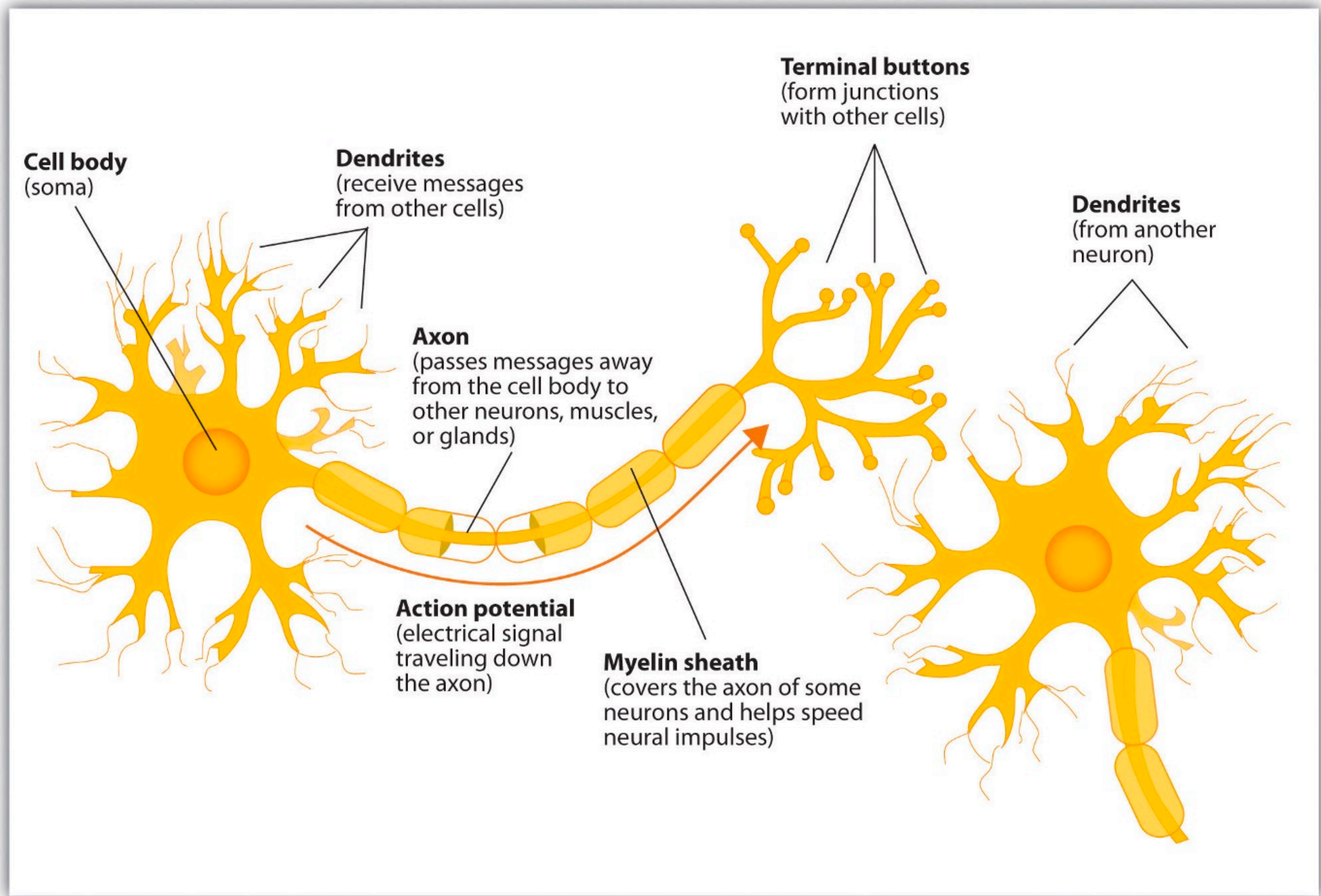
- Single layer has limited capability
  - cannot learn XOR
- The decision boundary is linear
  - cannot learn a nonlinear decision boundary
  - why?

How to build more expressive models?

- Inspiration from human brain
  - The human brain is a connectionist machine
  - neuron

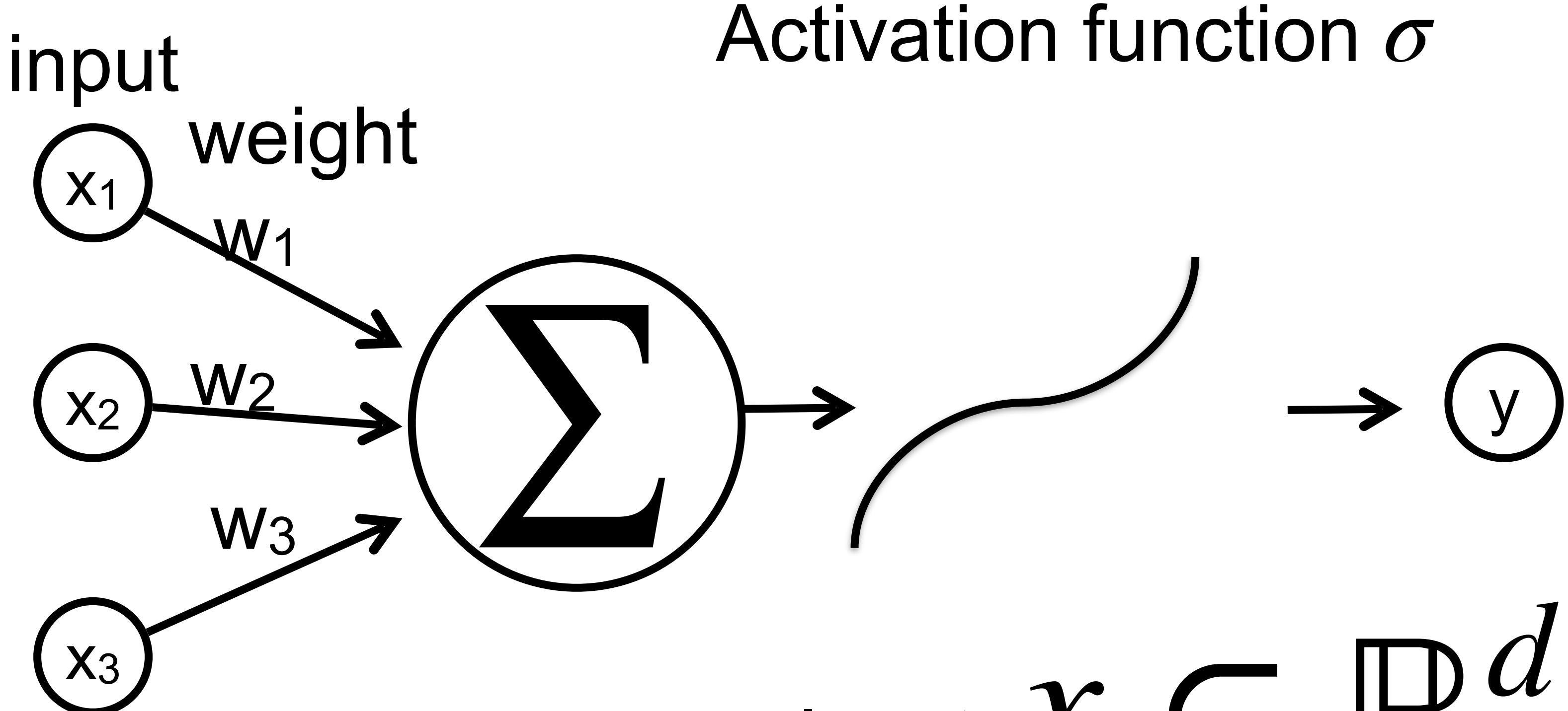






https://en.wikipedia.org/wiki/File:Components_of_neuron.jpg

A single Artificial Neuron



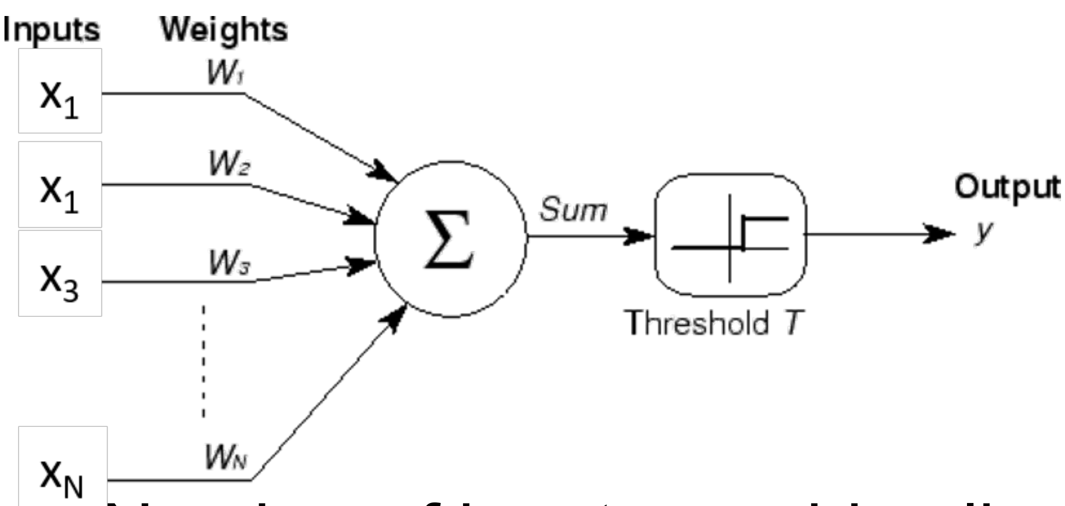
Input: $x \in \mathbb{R}^d$

Weight:

$$w \in \mathbb{R}^d, b \in \mathbb{R}$$

Perceptron

- Frank Rosenblatt
  - Psychologist, Logician
  - Inventor of the solution to everything, aka the Perceptron (1958)

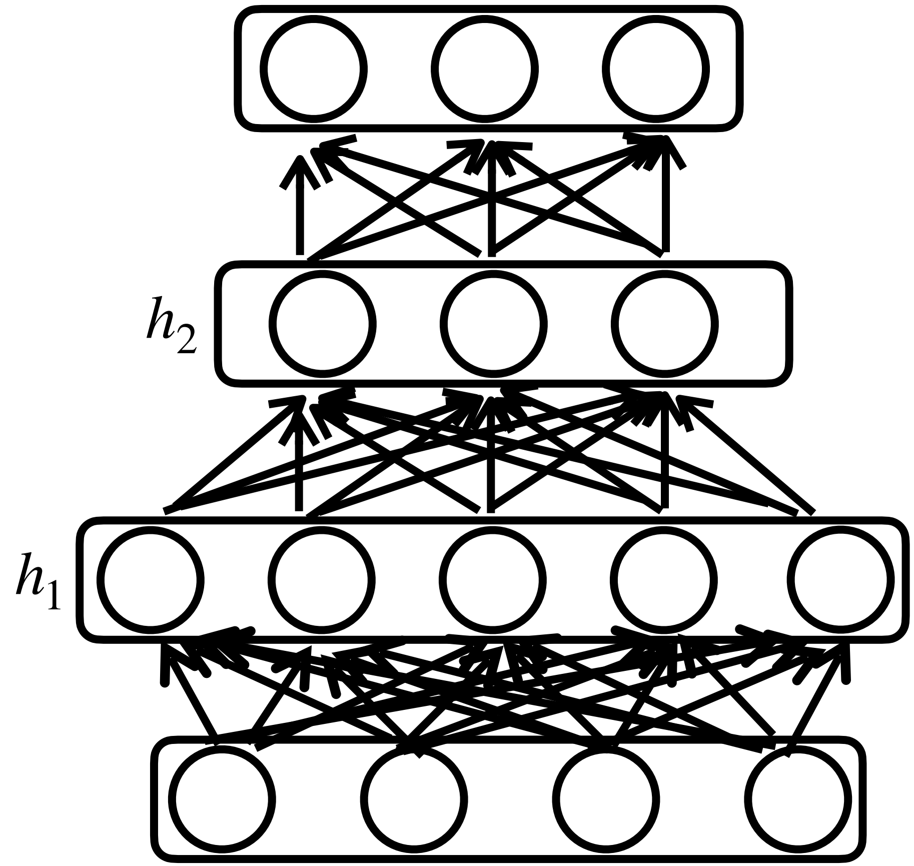


- Number of inputs combine linearly
  - Threshold logic: Fire if combined input exceeds threshold

$$Y = \begin{cases} 1 & if \sum_i w_i x_i - T \geq 0 \\ 0 & else \end{cases}$$

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



Feedforward Neural Net (FFN)

- also known as multilayer perceptron (MLP)

$$x \in \mathbb{R}^d$$

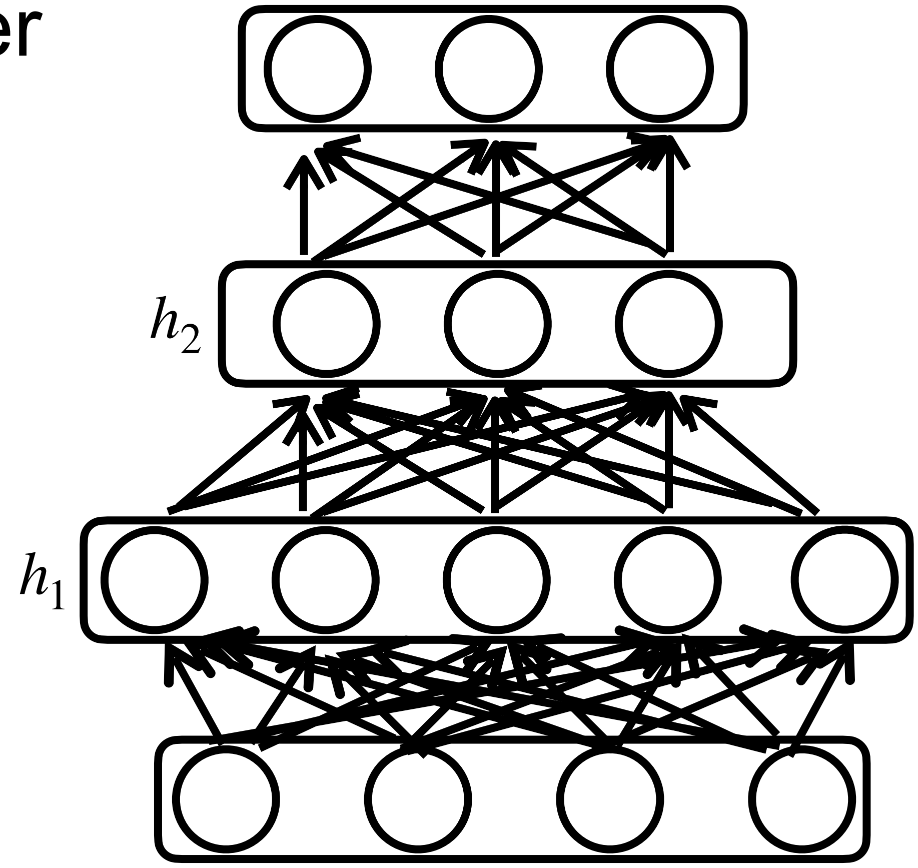
$$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$$

$$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$$

$$o = \text{Softmax}(w_L \cdot h_{L-1} + b_L)$$

Parameters

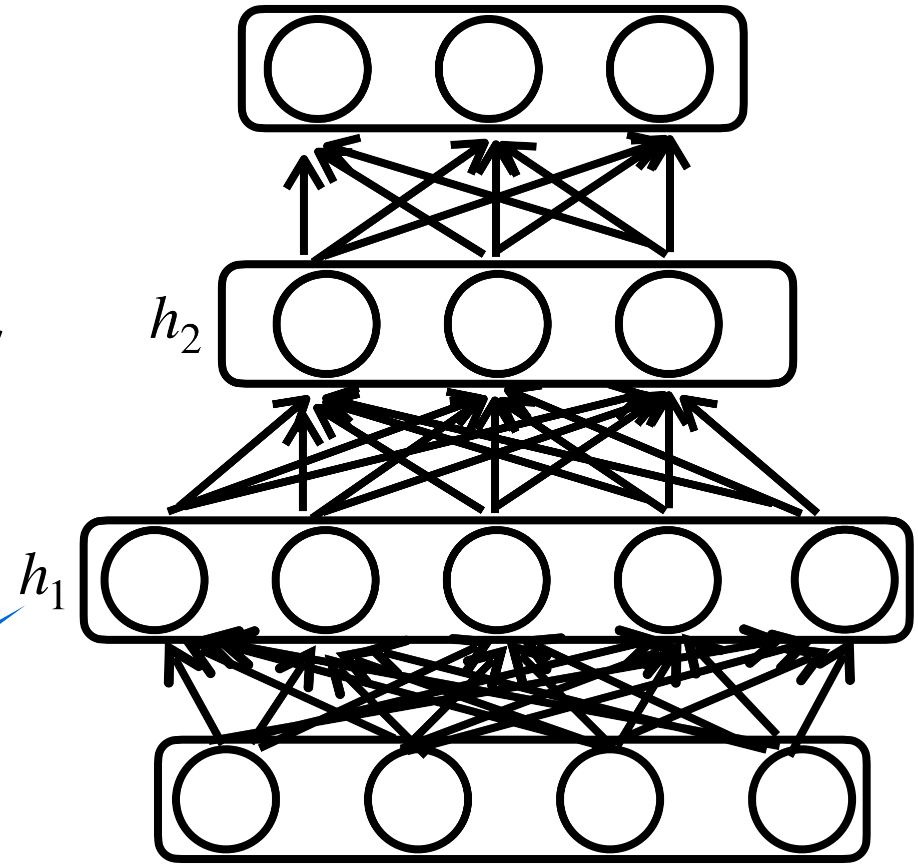
$$\theta = \{w_1, b_1, w_2, b_2, \ldots\}$$



Hidden layers

- $$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$$
  $$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$$

σ is element-wise nonlinear activation function



Why do we need an a nonlinear

What-if Layer with no activation?

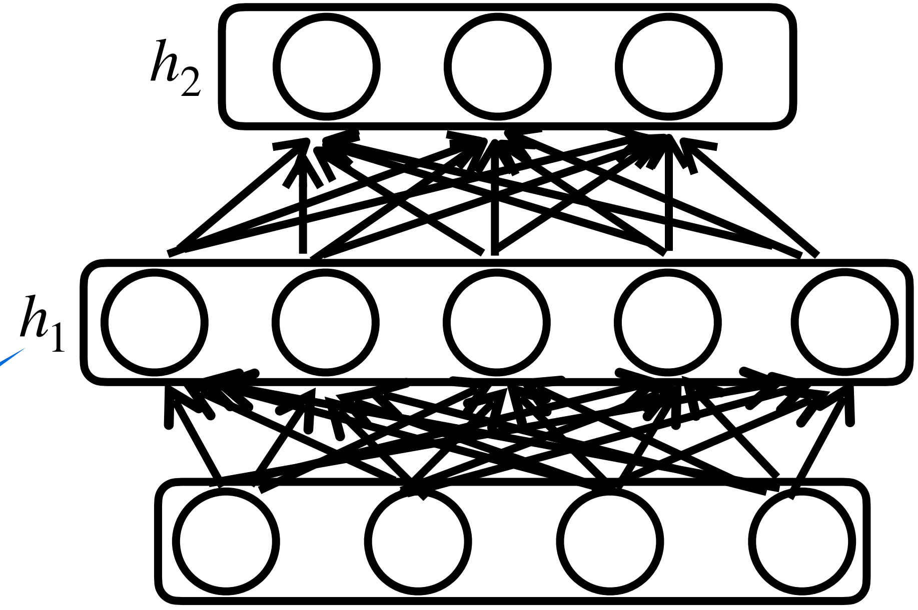
Linear ...

$$\mathbf{h}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{h}_2 = \mathbf{w}_2^T\mathbf{h}_1 + b_2$$

hence $h_2 = \mathbf{w}_2^\top \mathbf{W}_1\mathbf{x} + b'$

Why do we need an a nonlinear

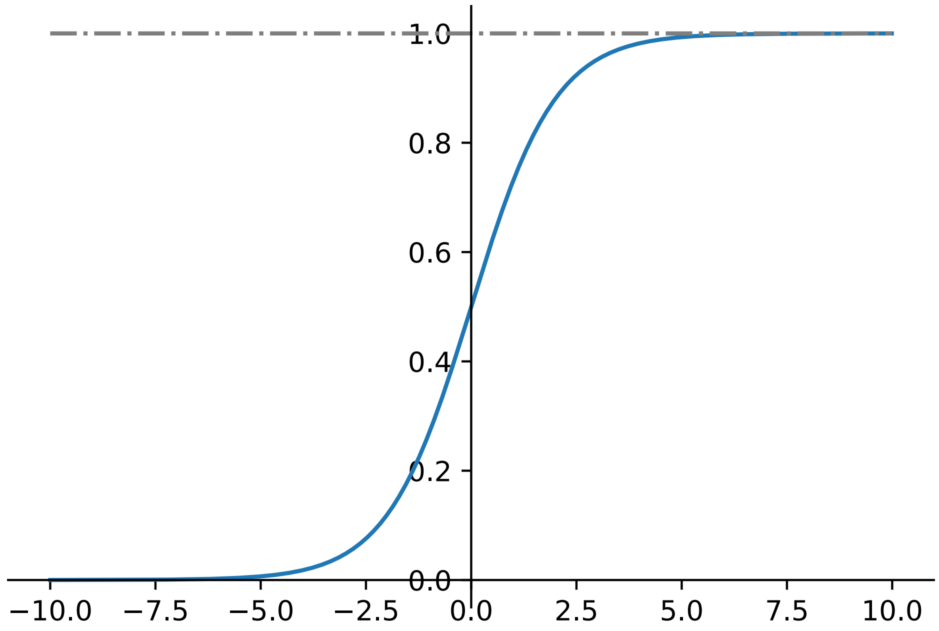


Sigmoid Activation

Map input into (0, 1), a soft version of

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

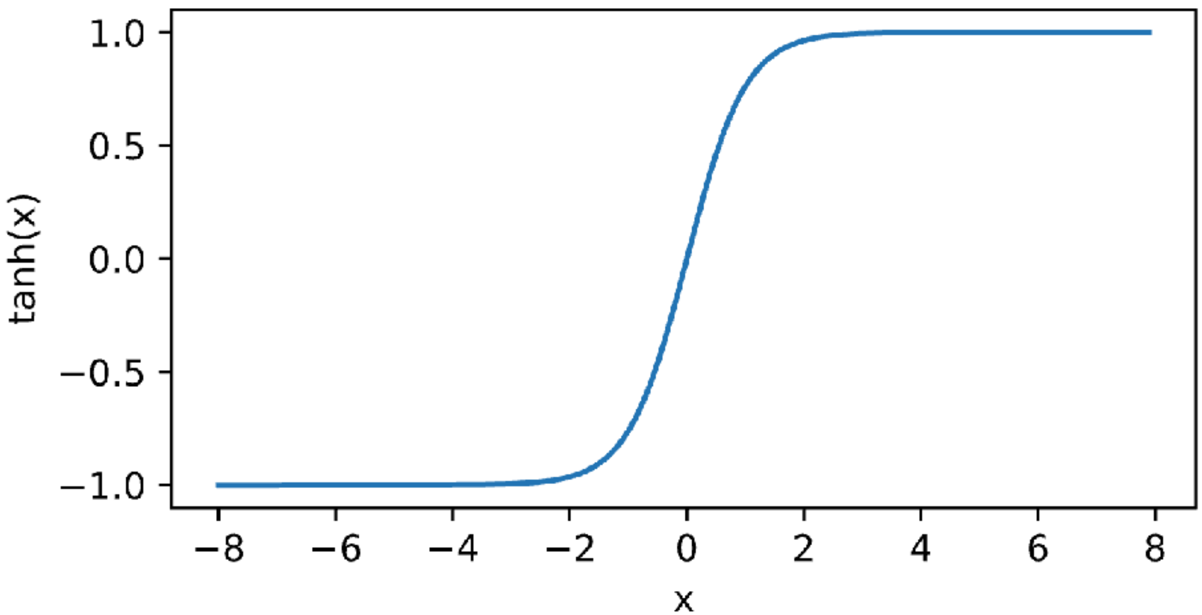
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Tanh Activation

Map inputs into (-1, 1)

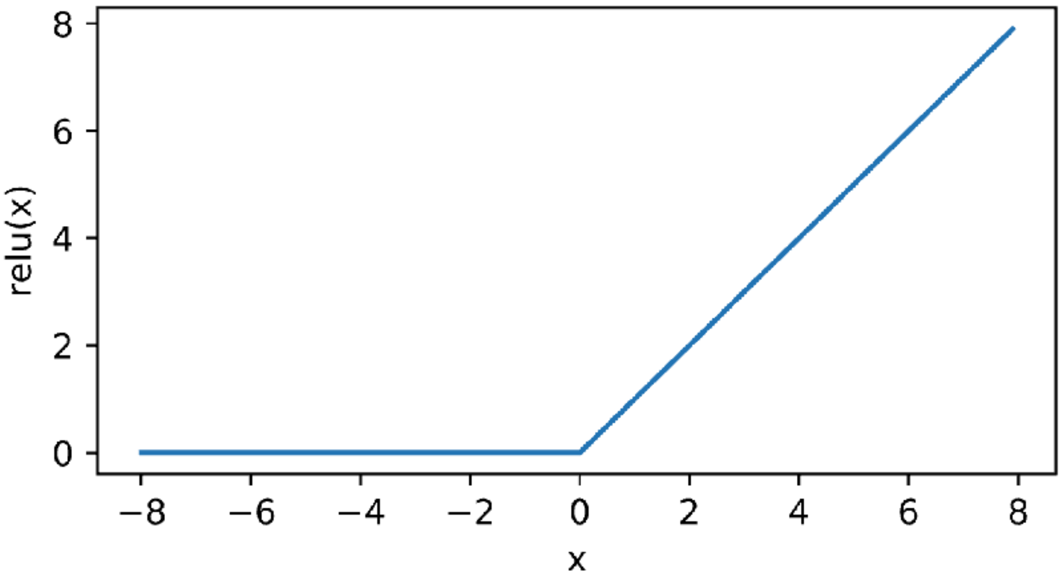
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



ReLU Activation

ReLU: rectified linear unit

$$\mathrm{ReLU}(x) = \max(x, 0)$$

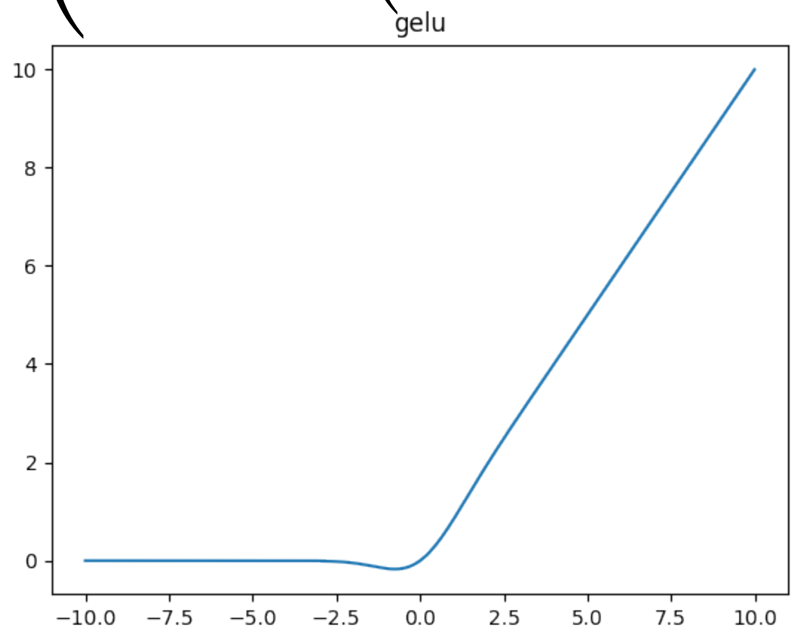


Gaussian Error Linear Units (GELU)

smoothed version of RELU

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right]$$

$$\text{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right)$$



Feedforward Network for Classification

Softmax as the final output layer.

$x \in \mathbb{R}^d$

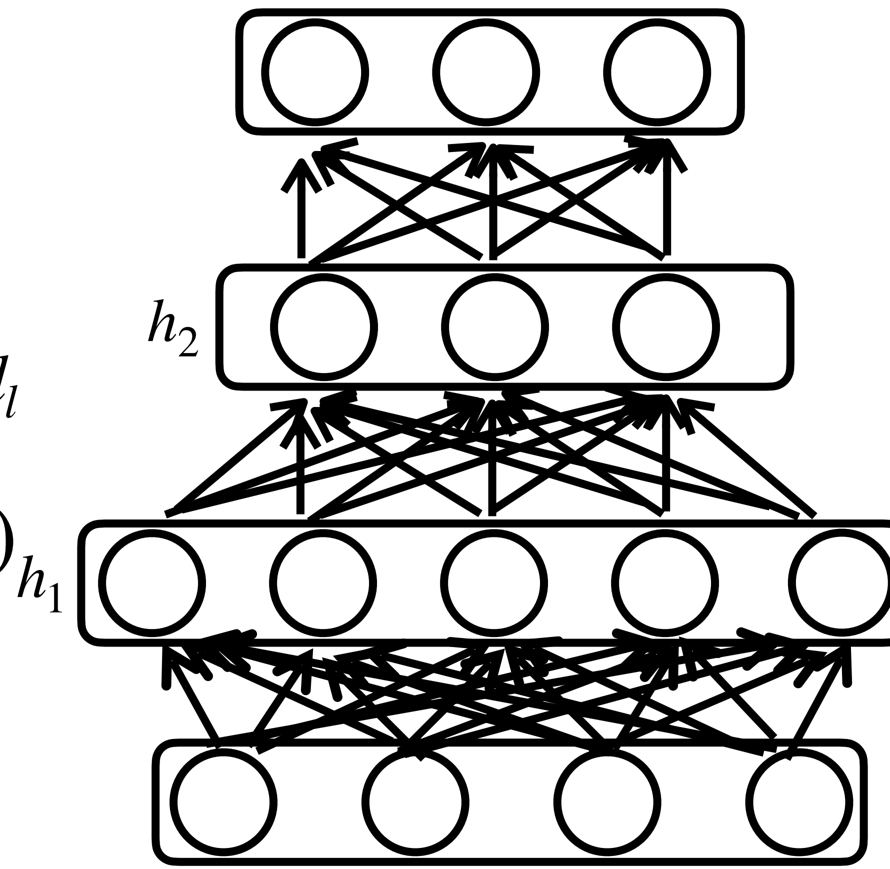
$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$

$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$

$o = \mathrm{Softmax}(w_L \cdot h_{L-1} + b_L)$

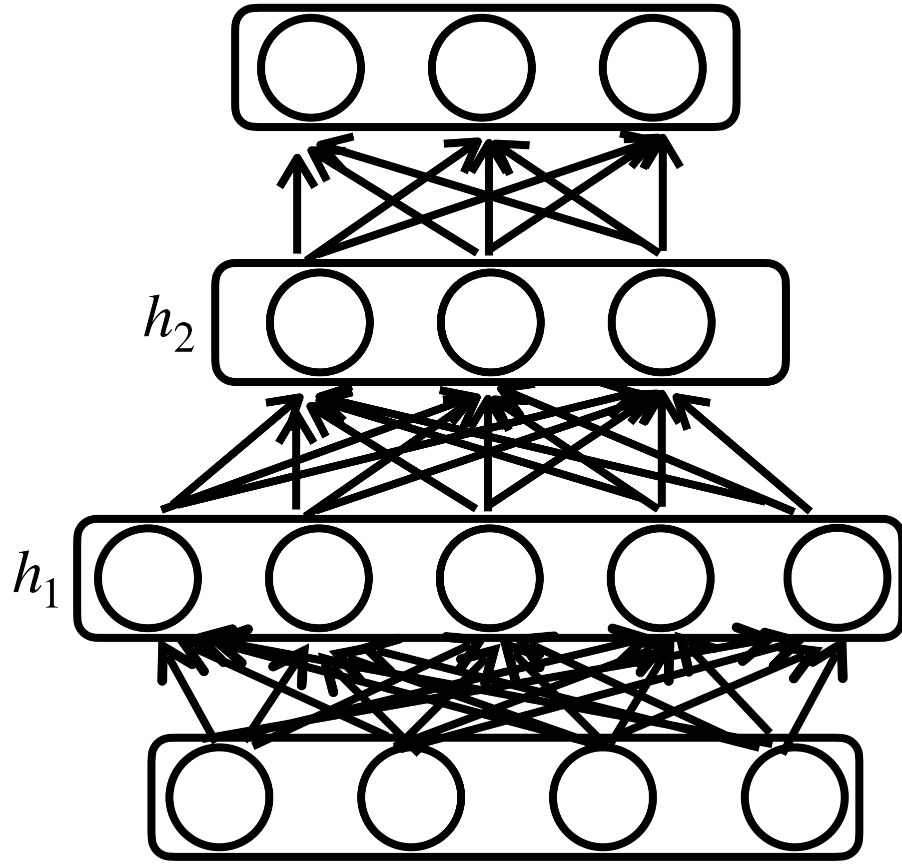
Parameters

$\theta = \{w_1, b_1, w_2, b_2, \ldots\}$



Hyperparameters for FFN

- Number of layers
- Number of hidden dimension for each layer
- These determine the hypothesis class



Application: Sentiment Analysis

"We enjoyed our stay so much. The weather was not great, but everything else was perfect." 😄

"There were no clean linens when I got to my room and the breakfast options were not that many." ☹️

"Best weekend in the countryside I've ever had." 😄

"Terrible. Slow staff, slow town. Only good thing was being surrounded by nature." ☹️

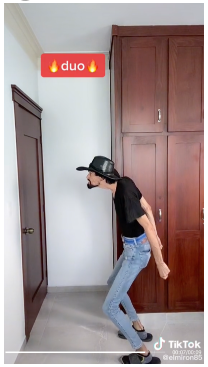
"It was a peaceful getaway in the countryside." 😄

Pytorch implementation:

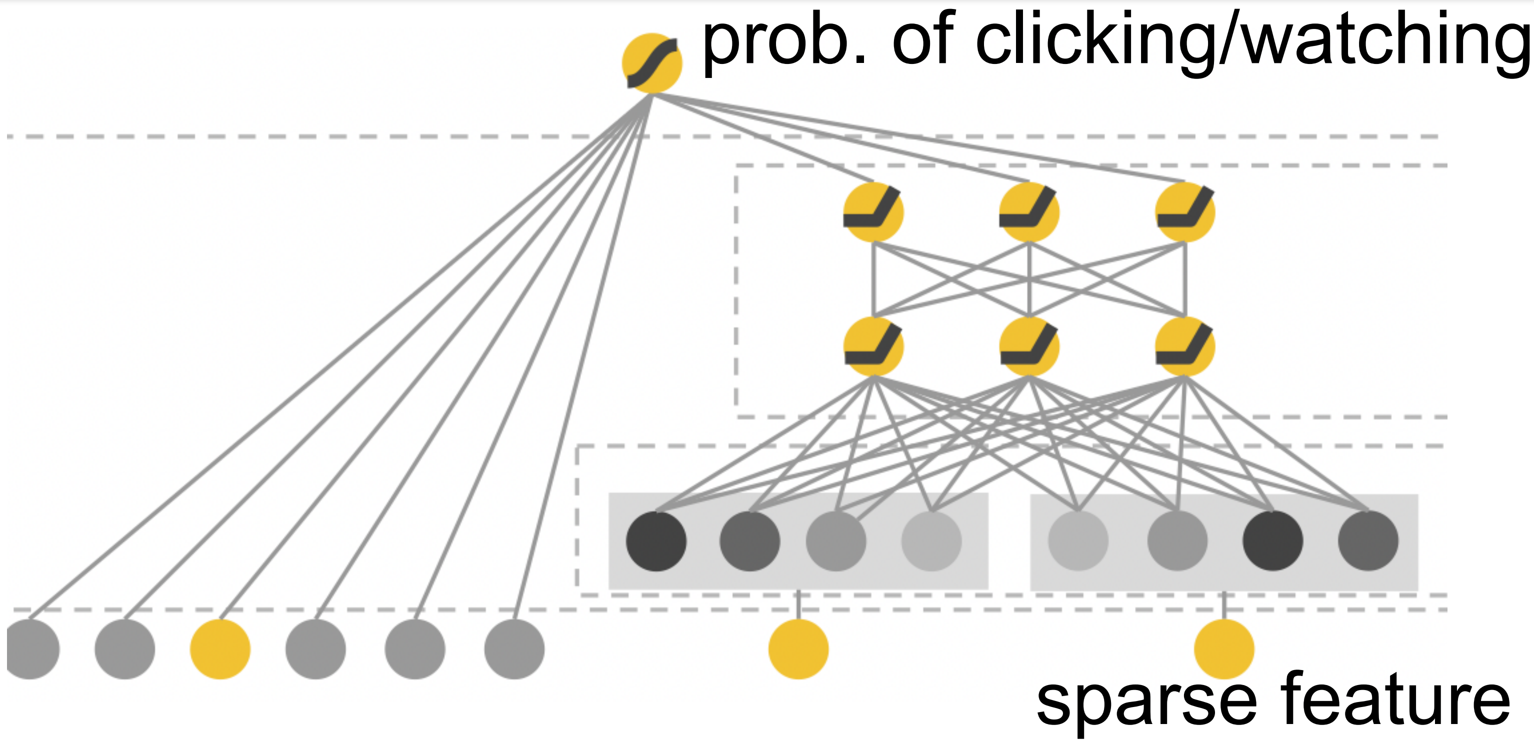
```
model = torch.nn.Sequential(
    torch.nn.Linear(10000, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 2)
)
```

Application: Recommendation Model

- Recommending videos on Tiktok
- 1.5 billion users (growing from 10 million in 2017)
- > 10 million videos for recommendation
- Response time: < 50ms



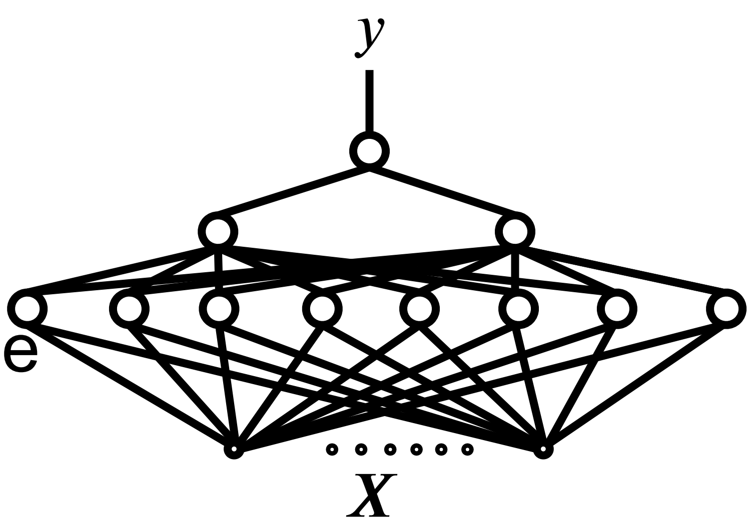
Deep&Wide Model



- user features (e.g., country, language, demographics),
- contextual features (e.g., device, hour of the day, day of the week)
- impression features (e.g., historical statistics)
- Content features (e.g. item id, extracted image feature, title feature)

The Learning Problem

- Given a training set of input-output pairs $D = \{(x_n, y_n)\}_{n=1}^{N}$
  - x_n and y_n may both be vectors
- To find the model parameters such that the model produces the most accurate output for each training input
  - Or a close approximation of it
- Learning the parameter of a neural network is an instance!
  - The network architecture is given



Recap: Risk

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

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

Empirical Risk Minimization (ERM)

- Ideally, we want to minimize the expected risk
  - but, unknown data distribution …
- Instead, given a training set of empirical data $D = \{(x_n, y_n)\}_{n=1}^{N}$
- Minimize the empirical risk over training data

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

Learning the Model

- Ideally, we want to minimize the expected Risk

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

- Finding the parameter θ to minimize the empirical risk over training data $D = \{(x_n, y_n)\}_{n=1}^{N}$

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

- This is an instance of function optimization problem
  - optimization algorithms from previous lecture

Loss for Classification

- The empirical risk (loss) is determined by the error function
- Ideal error for classification: 0-1 loss

$$l(y, f(x)) = \begin{cases} 0 & \text{if } y = \arg\max_k f(x)_k \\ 1 & \text{otherwise} \end{cases}$$

- Cross entropy is one common error function for classification

$$\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} -y_n \cdot \log f(x_n)$$

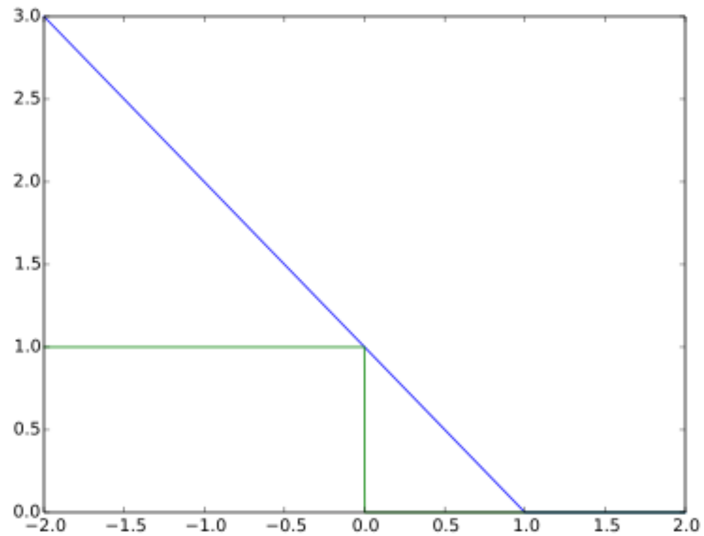
Other Loss for Classification

- Hinge loss

  - Binary classification:

$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

When ground-truth y is +1, prediction $\hat{y}$<0 lead to larger penalty



  - Multi-class

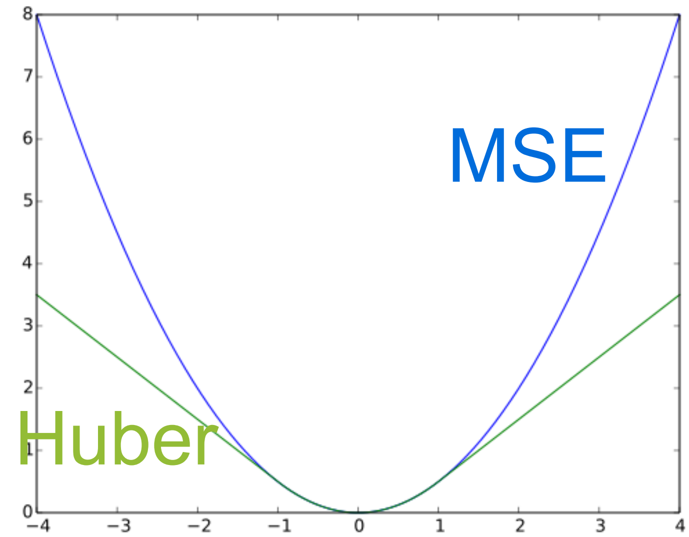
$$\ell(y, \hat{y}) = \sum_{k \neq y} \max(0, 1 - \hat{y}_y + \hat{y}_k)$$

Loss for Regression

- Continuous outcome

$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(x_n))$$

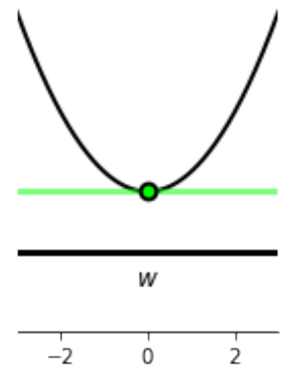
- squared loss: $\ell(y, f) = \frac{1}{2}|f - y|_2^2$
- L1 loss: $\ell(y, f) = \frac{1}{2}|f - y|$
- Huber loss: $\ell(y, f) = \begin{cases} \frac{1}{2}|f - y|_2^2 & \text{if } |f - y|_2 \leq \delta \\ \delta(|f - y| - \frac{\delta}{2}) & \text{otherwise} \end{cases}$

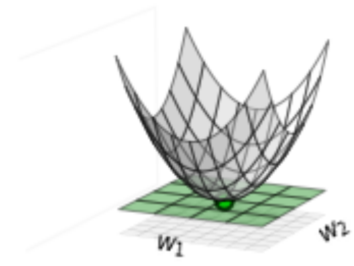


Recap: Optimization

- Consider a generic function minimization problem

$$\min_x f(x) \text{ where } f : \mathbb{R}^d \to \mathbb{R}$$





- Optimality condition:

$$\nabla f|_x = 0, \text{ where i-th element of } \nabla f|_x \text{ is } \frac{\partial f}{\partial x_i}$$

- Linear regression has closed-form solution
- In general, no closed-form solution for the equation.

Stochastic Gradient Descent

- $f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t}$
- To make $\Delta x^T \nabla f|_{x_t}$ smallest
- $\Rightarrow \Delta x$ in the opposite direction of $\nabla f|_{x_t}$ i.e. $\Delta x = -\nabla f|_{x_t}$
- Update rule: $x_{t+1} = x_t - \eta \nabla f|_{x_t}$
- Gradient $\nabla f|_{x_t}$ is computed over a minibatch of samples.
- η is a hyper-parameter to control the learning rate

How to compute the Gradient for ~~Feedforward~~ Any Neural Network

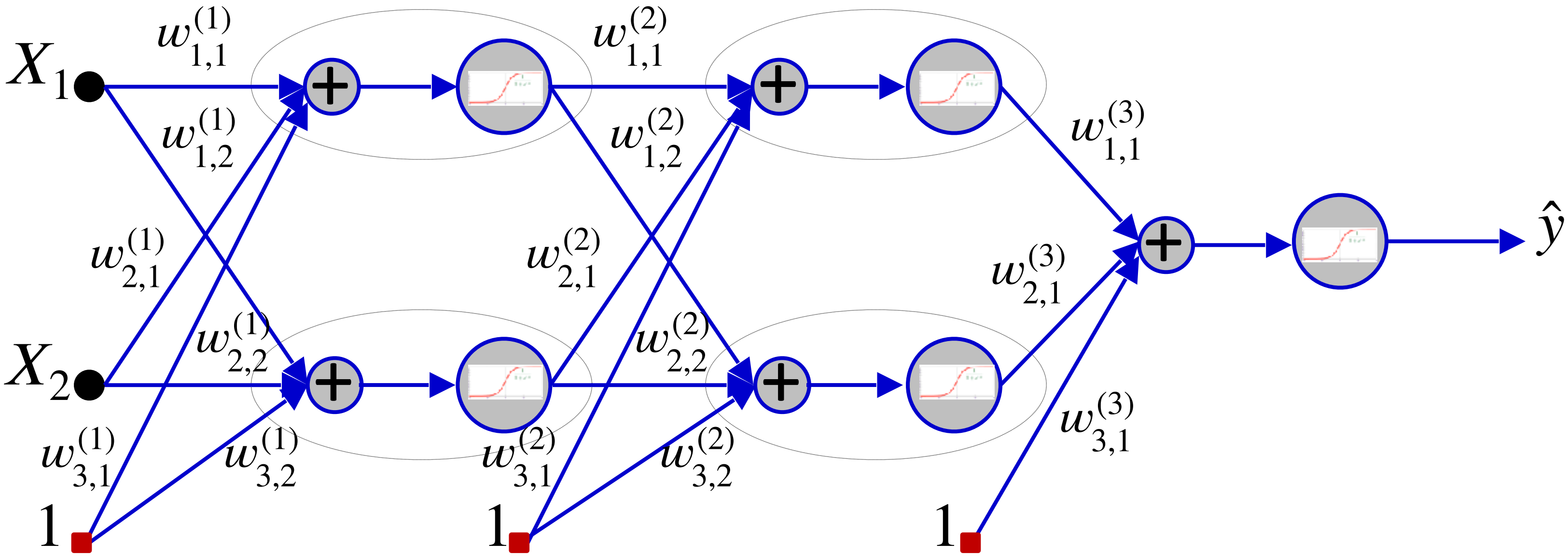
Computing Gradient for Neural Net

- Forward and back-propagation
- Suppose y=f(x), z=g(y), therefore z=g(f(x))
- Use the chain rule,

$$\nabla g(f(x))|_x = (\nabla f|_x)^T \cdot \nabla g|_y$$

- 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

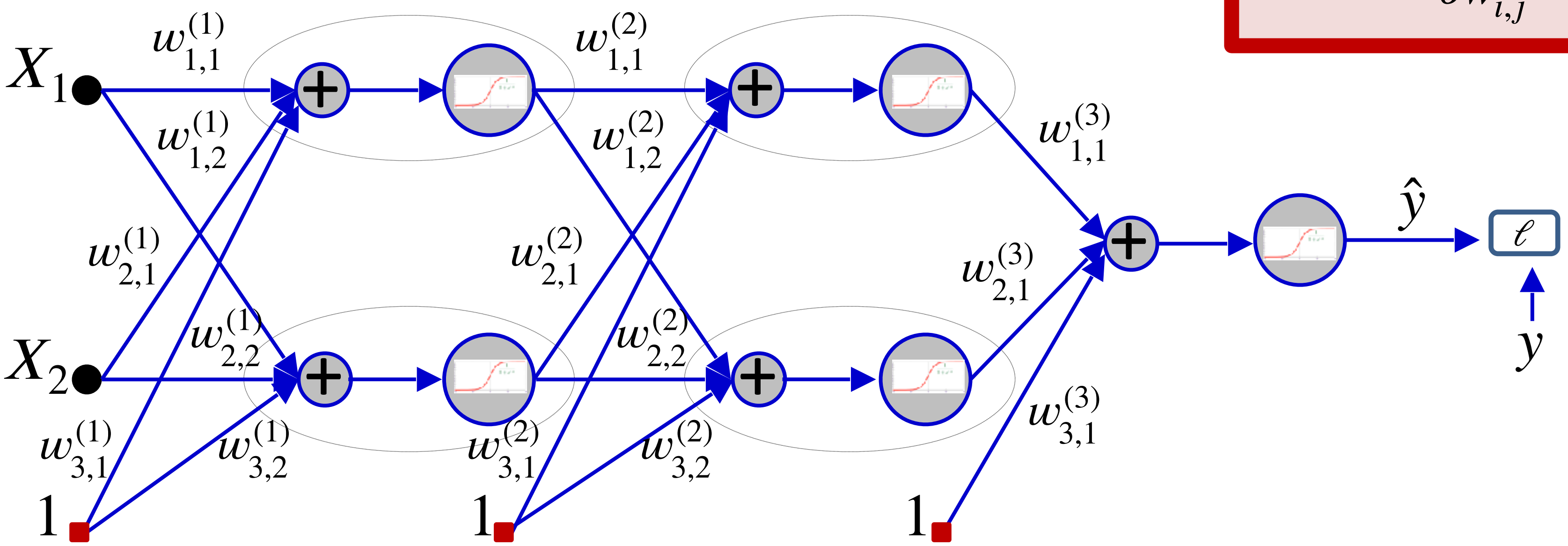
Example



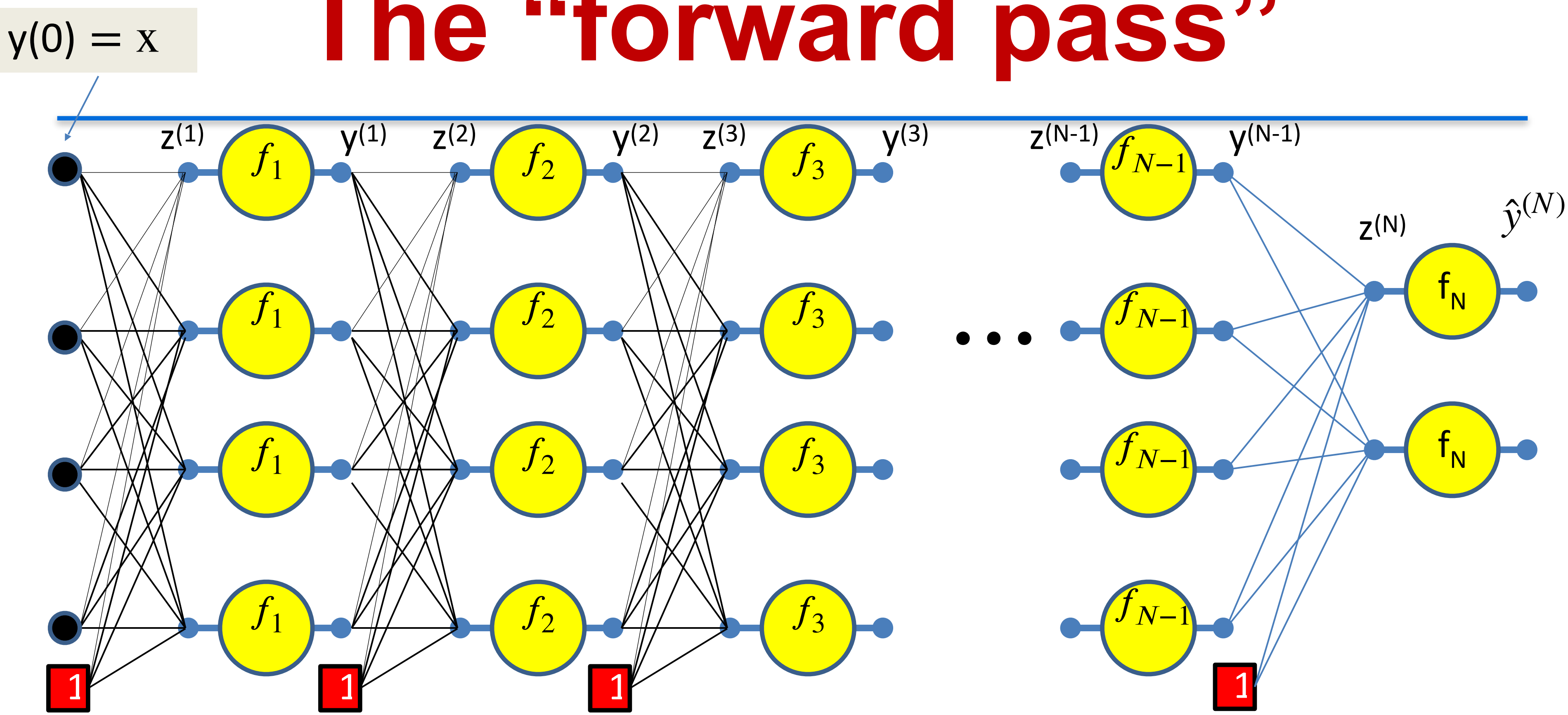
X_1
X_2
1
1
1
$w_{1,1}^{(1)}$
$w_{1,2}^{(1)}$
$w_{2,1}^{(1)}$
$w_{2,2}^{(1)}$
$w_{3,1}^{(1)}$
$w_{3,2}^{(1)}$
$w_{1,1}^{(2)}$
$w_{1,2}^{(2)}$
$w_{2,1}^{(2)}$
$w_{2,2}^{(2)}$
$w_{3,1}^{(2)}$
$w_{3,2}^{(2)}$
$w_{1,1}^{(3)}$
$w_{2,1}^{(3)}$
$w_{3,1}^{(3)}$
$\hat{y}$

Computing the Gradient

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



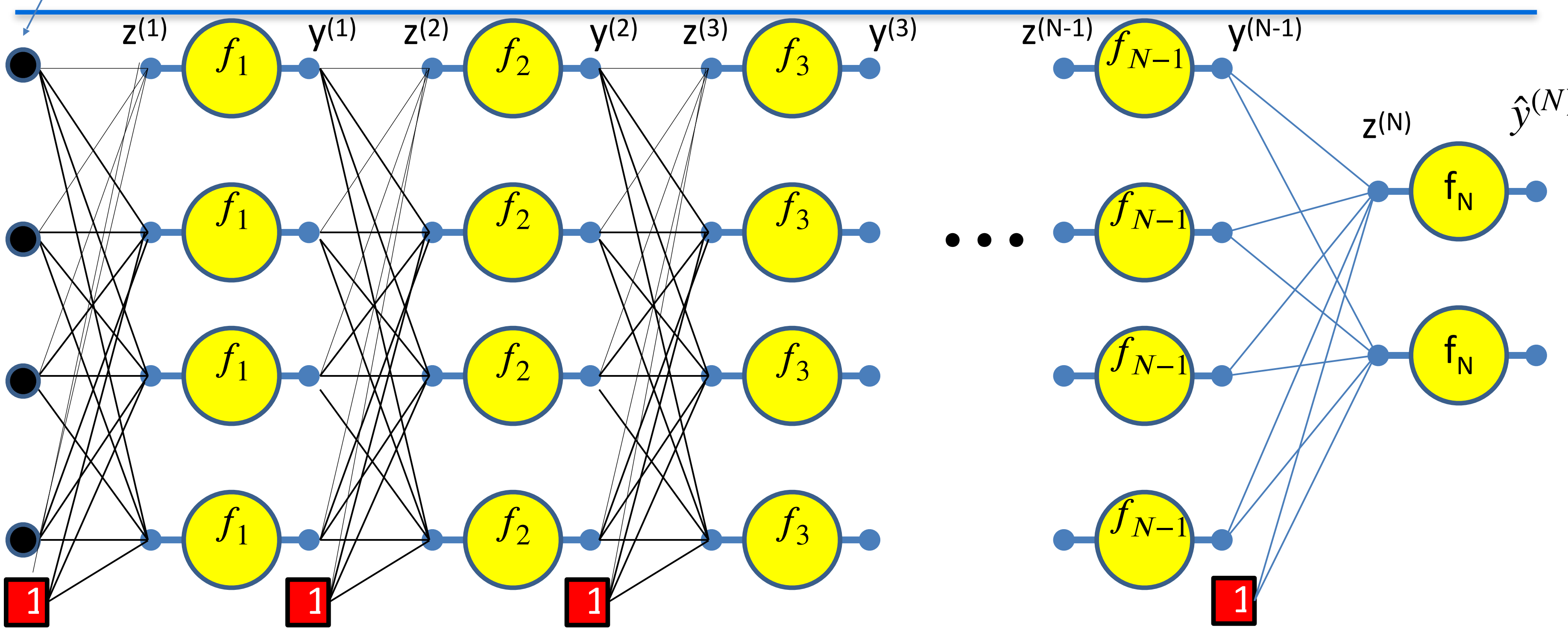
The "forward pass"



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

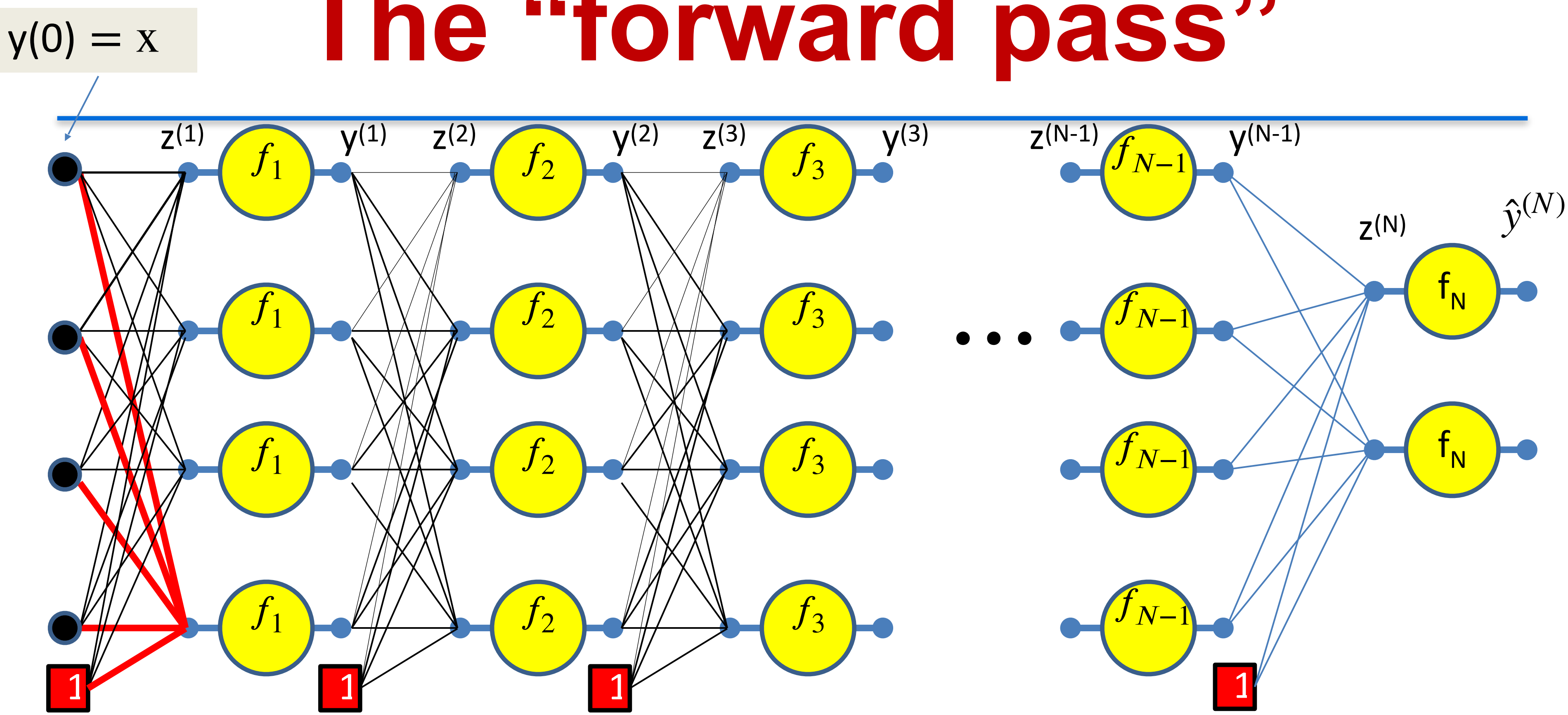
The “forward pass”



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

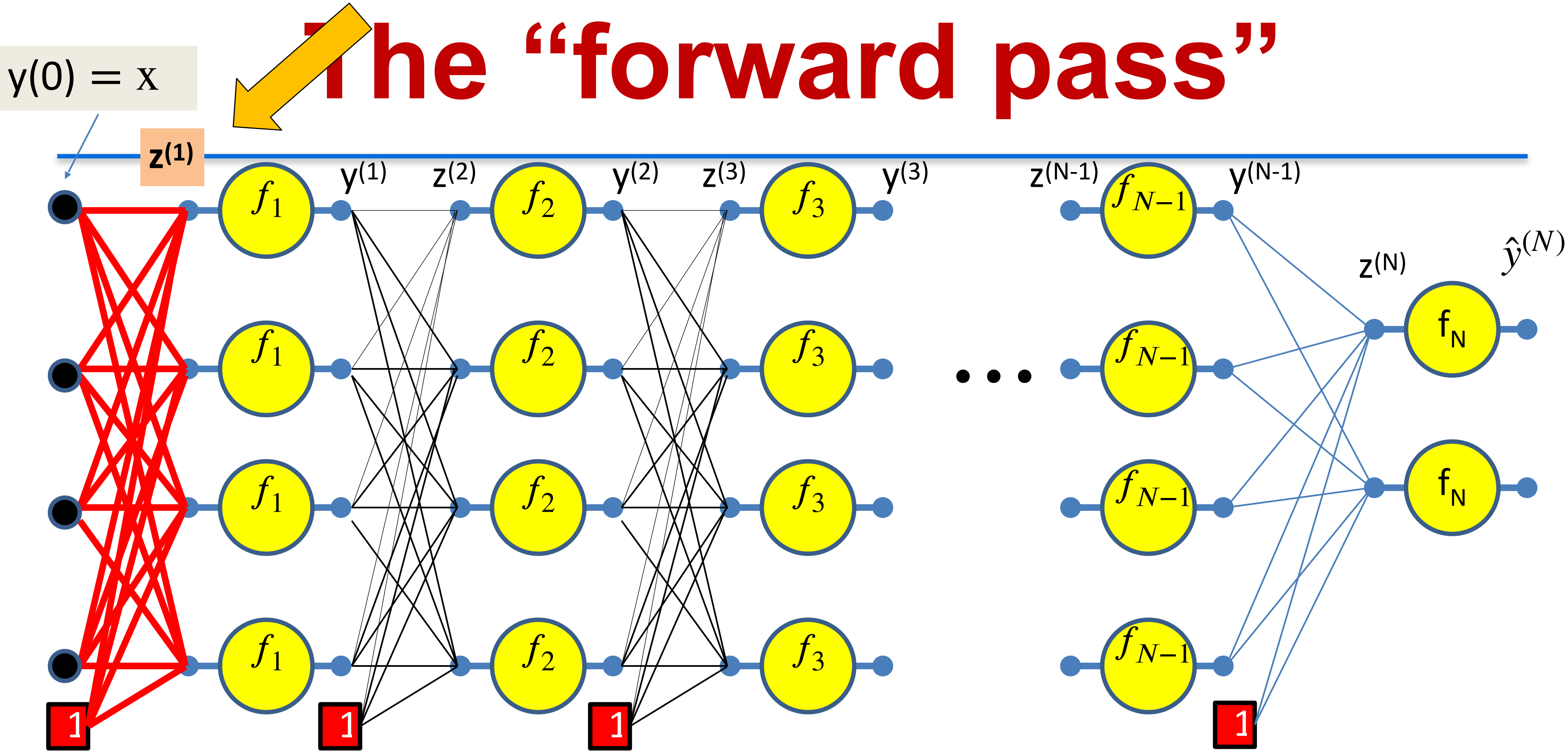
Assuming $w_{0j}^{(k)} = b_j^{(k)}$ and $y_0^{(k)} = 1$ -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases

The “forward pass”

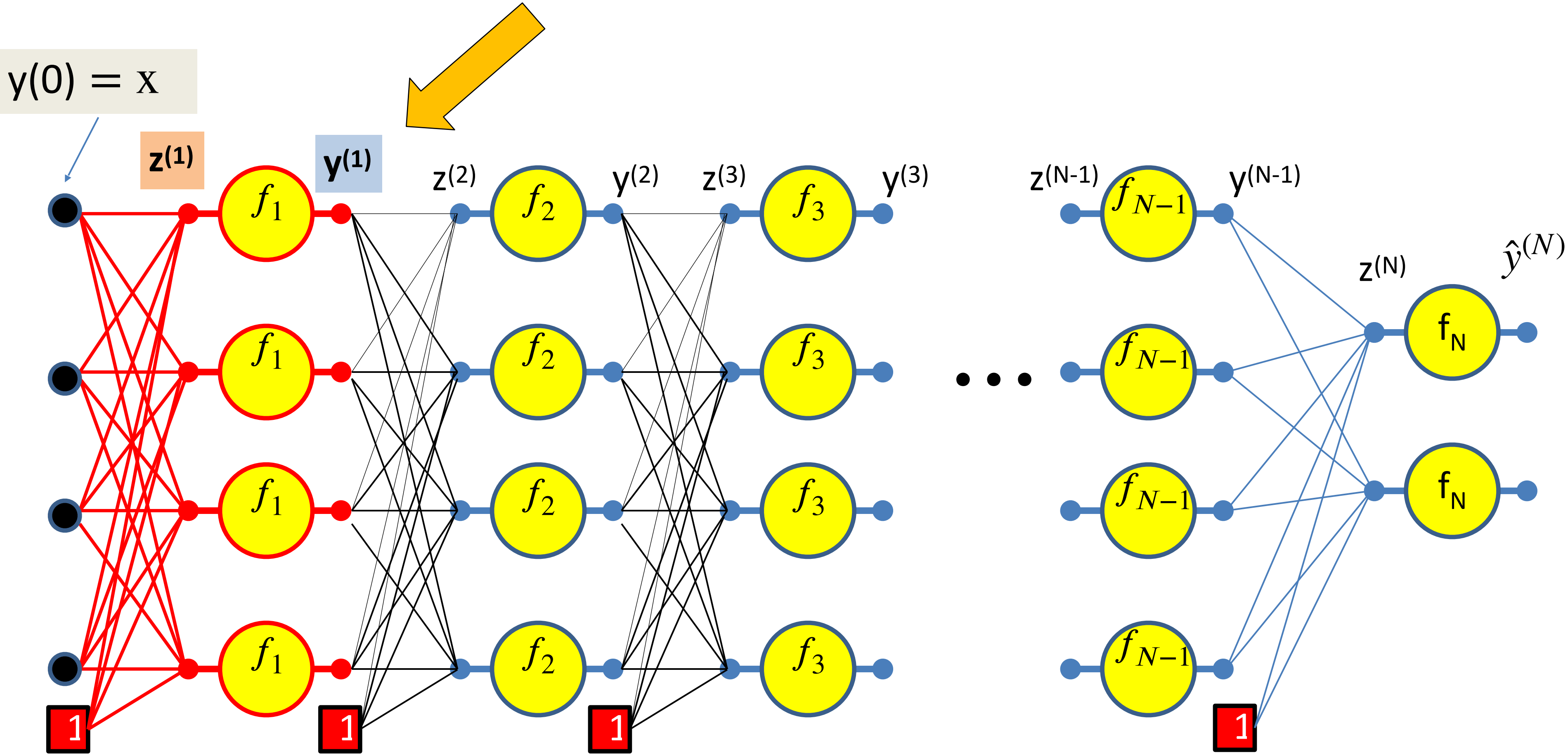


$$z_1^{(1)} = \sum_i w_{i1}^{(1)} y_i^{(0)}$$

The “forward pass”

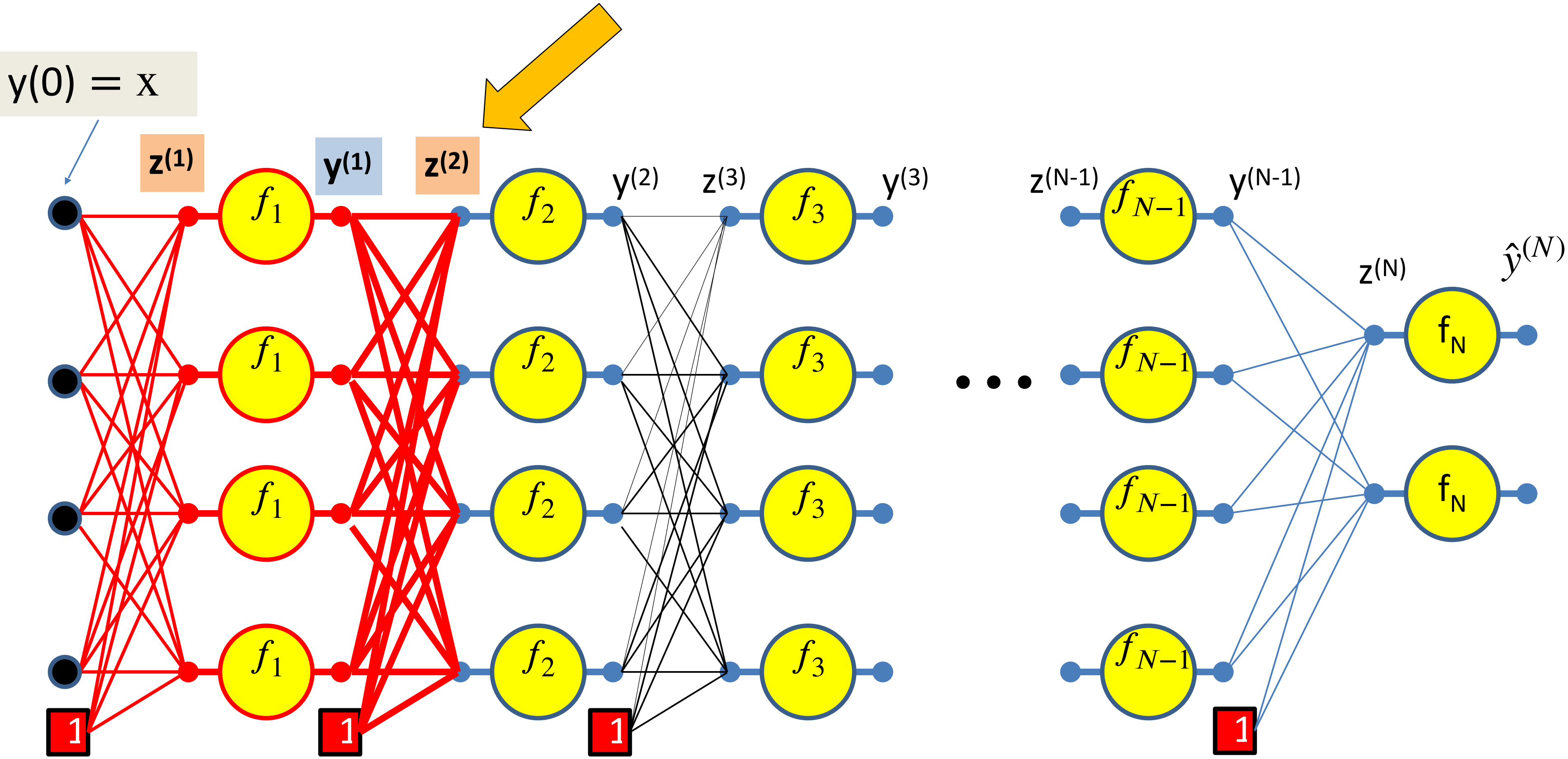


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$



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

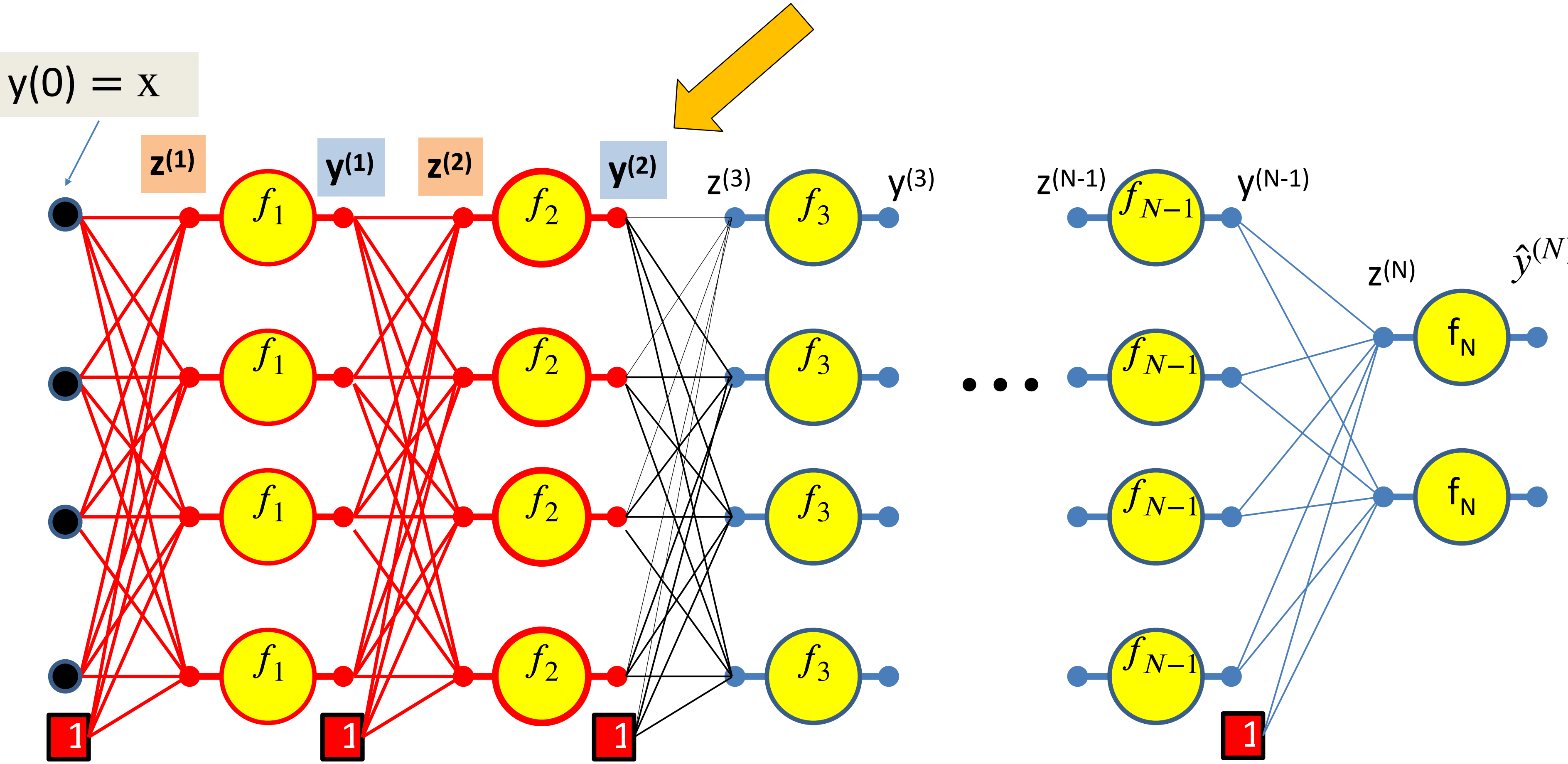
$$\boxed{y_j^{(1)} = f_1\left(z_j^{(1)}\right)}$$



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

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

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

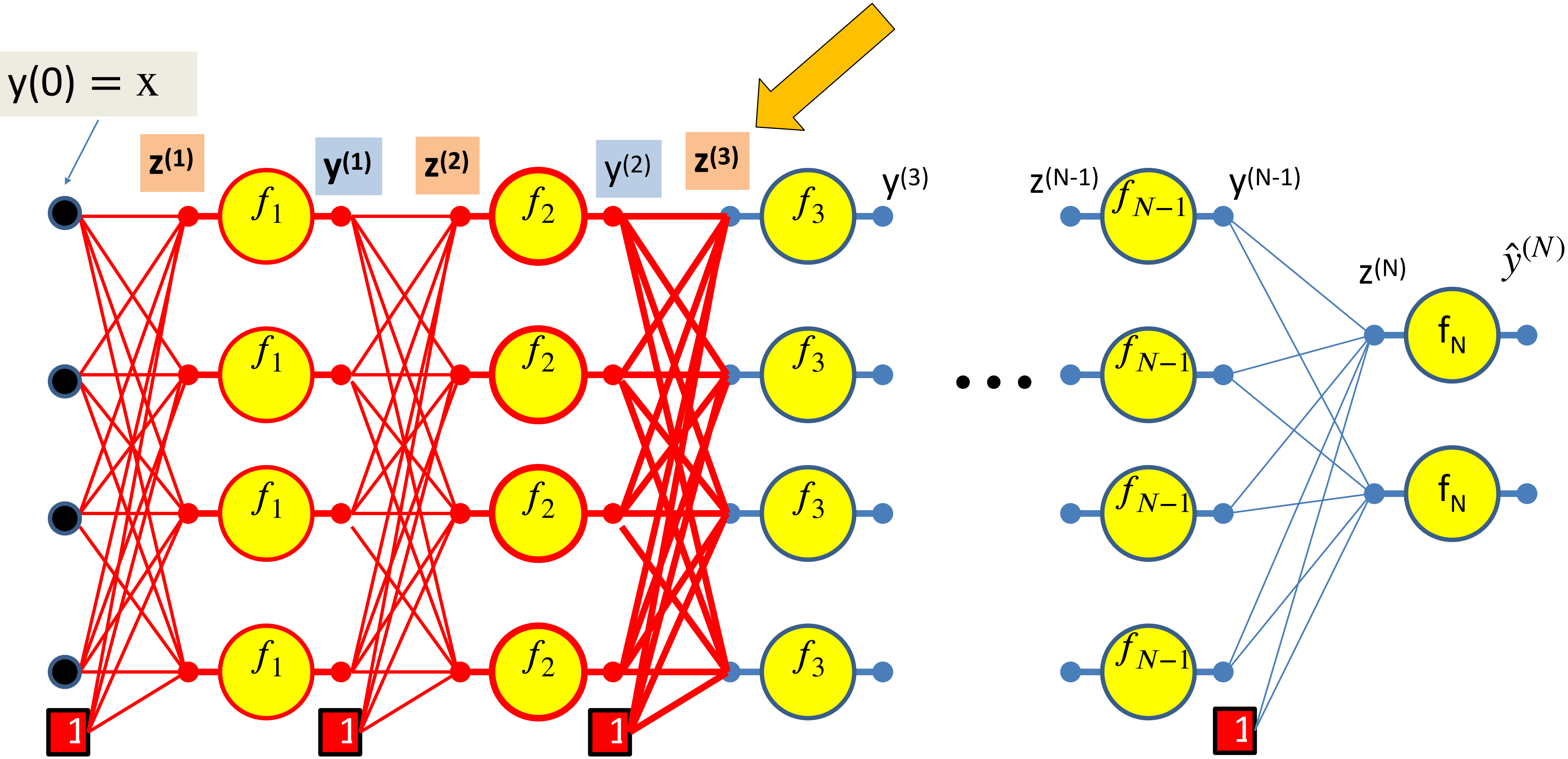


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

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

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

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



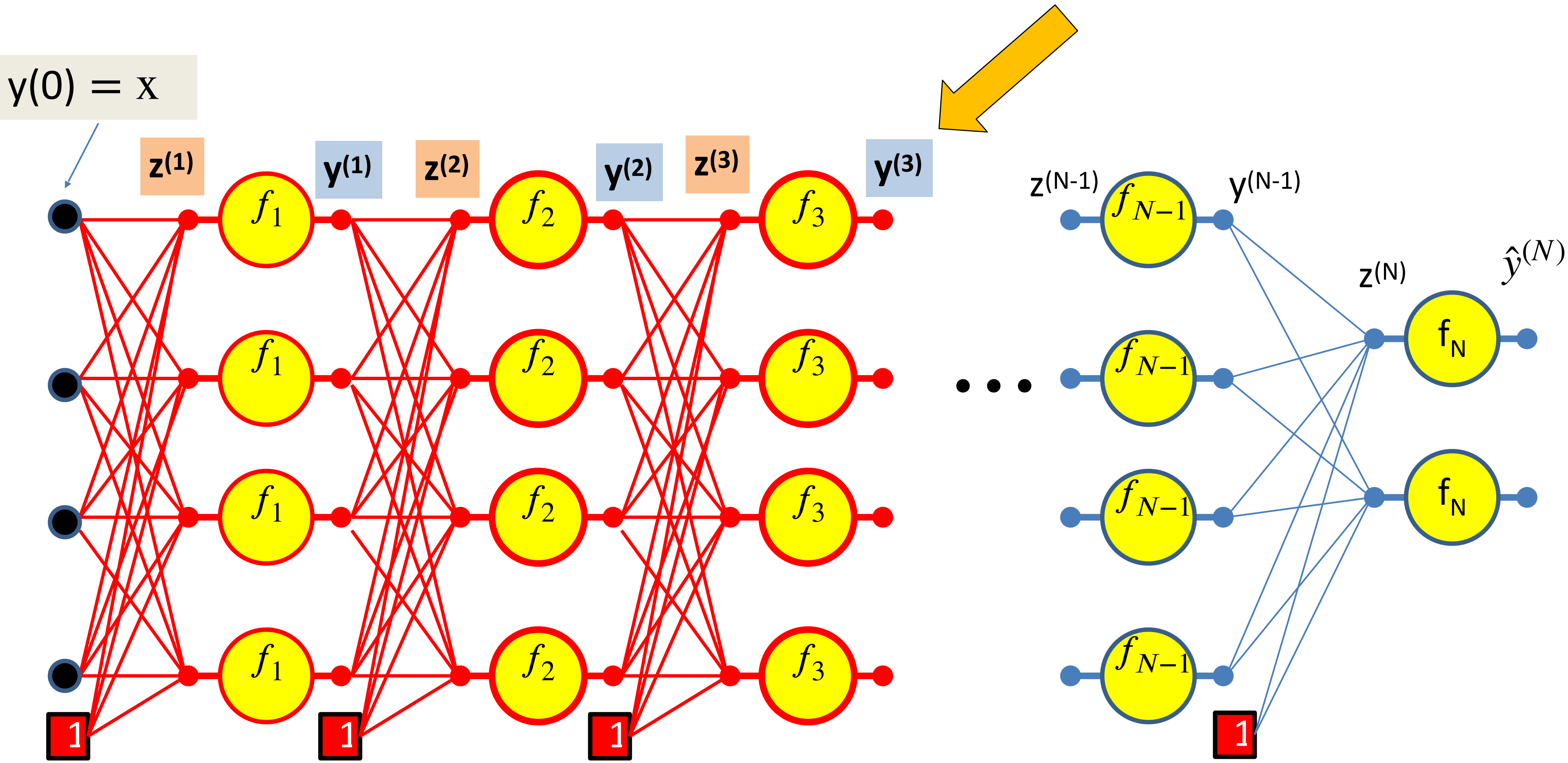
$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

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

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

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

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



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

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

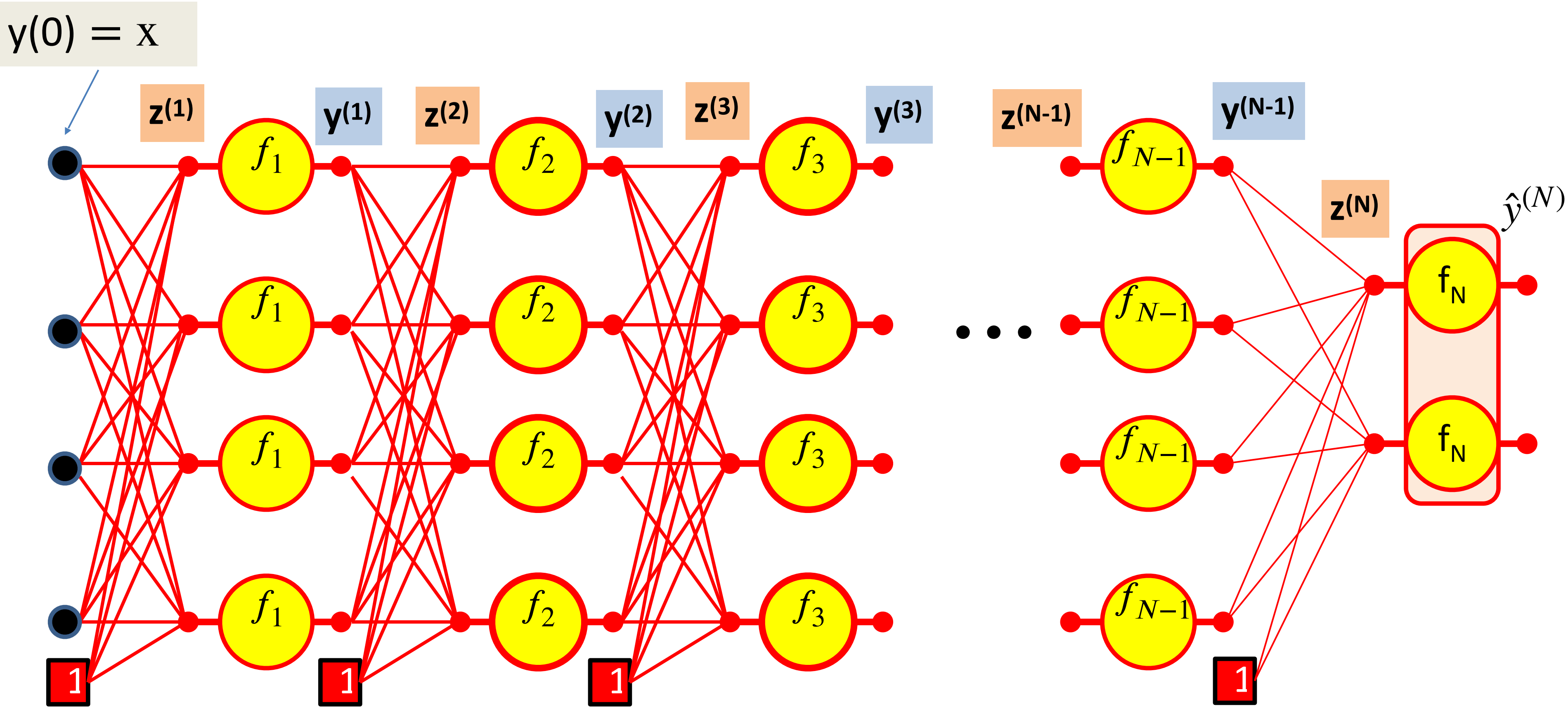
$$z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)}$$

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

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

$$\boxed{y_j^{(3)} = f_3\left(z_j^{(3)}\right)}$$

. . .

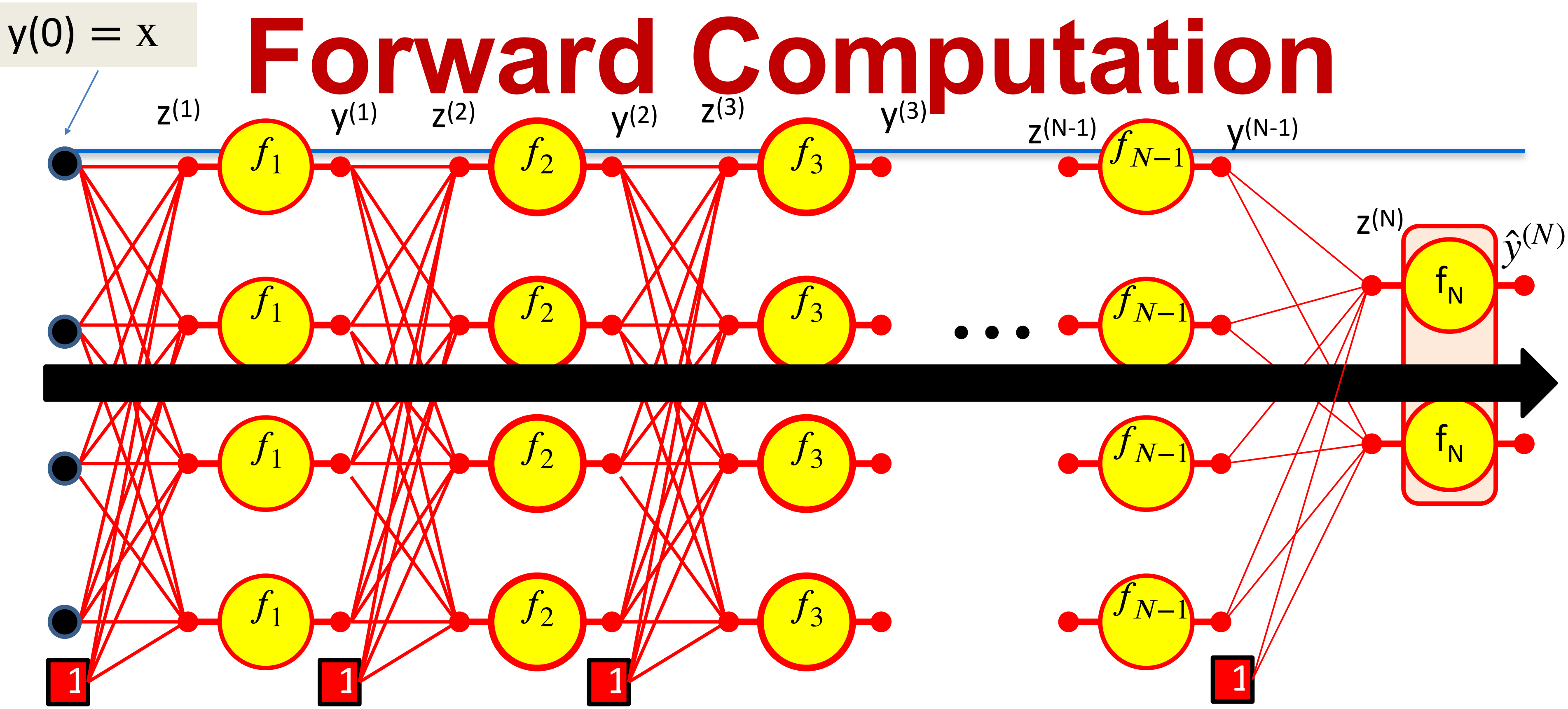


$$y_j^{(N-1)} = f_{N-1}\left(z_j^{(N-1)}\right)$$

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

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

Forward Computation



ITERATE FOR k = 1:N

for j = 1:layer-width

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

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

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

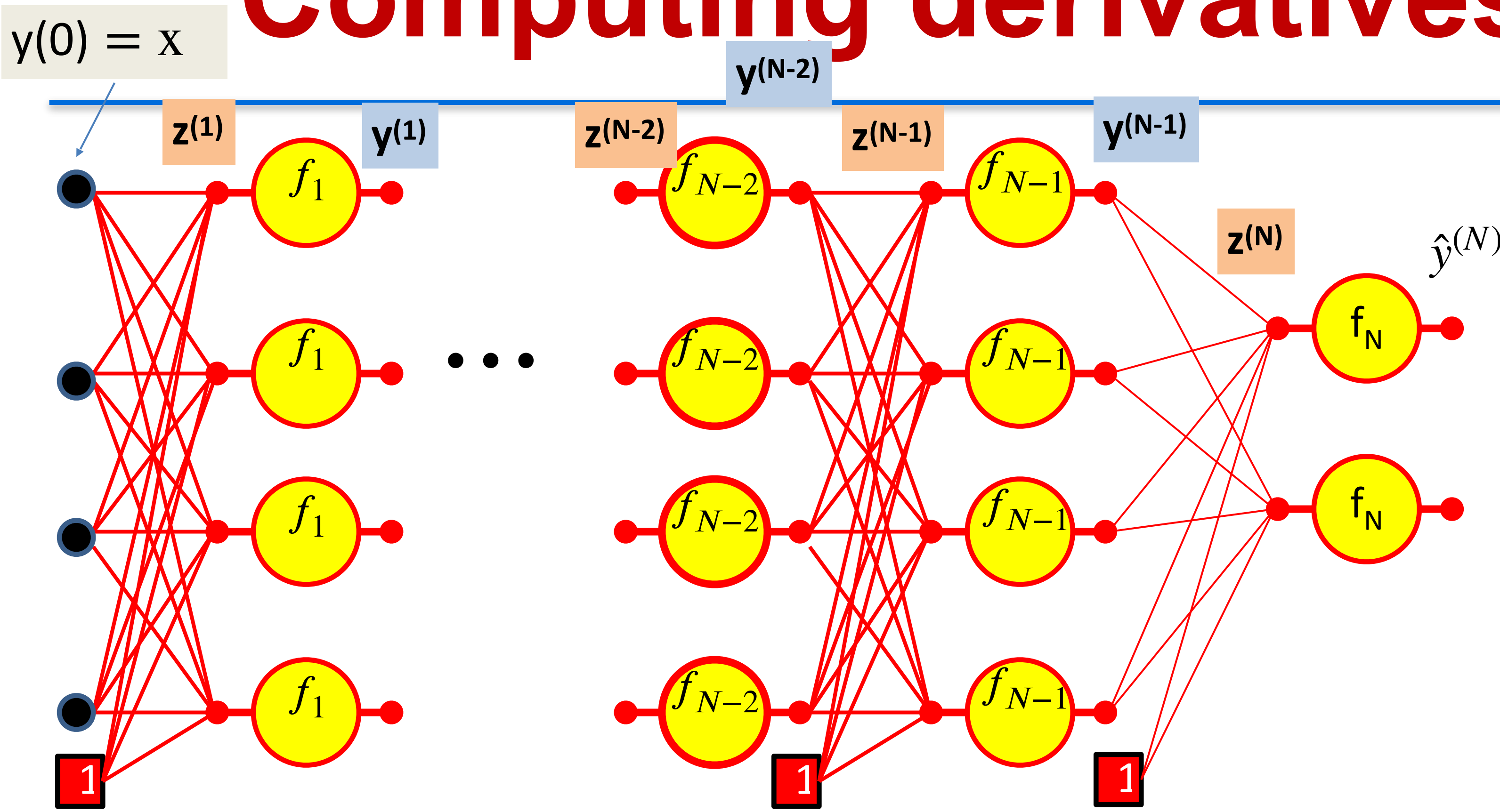
Forward "Pass"

- 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

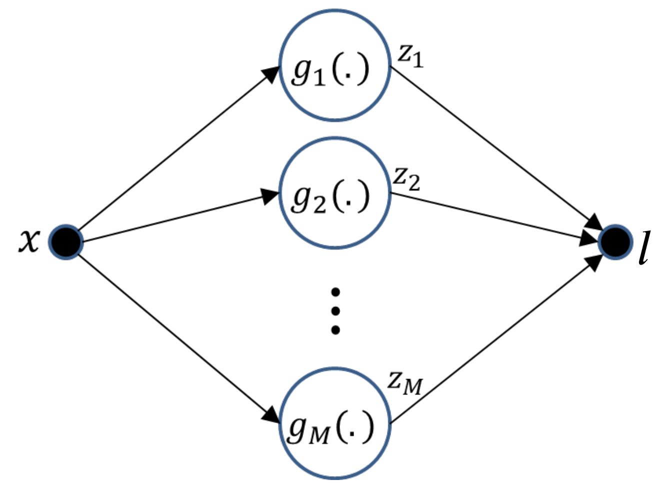
We must remember them – we will need them to compute the derivatives

Calculus Refresher: Chain rule

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

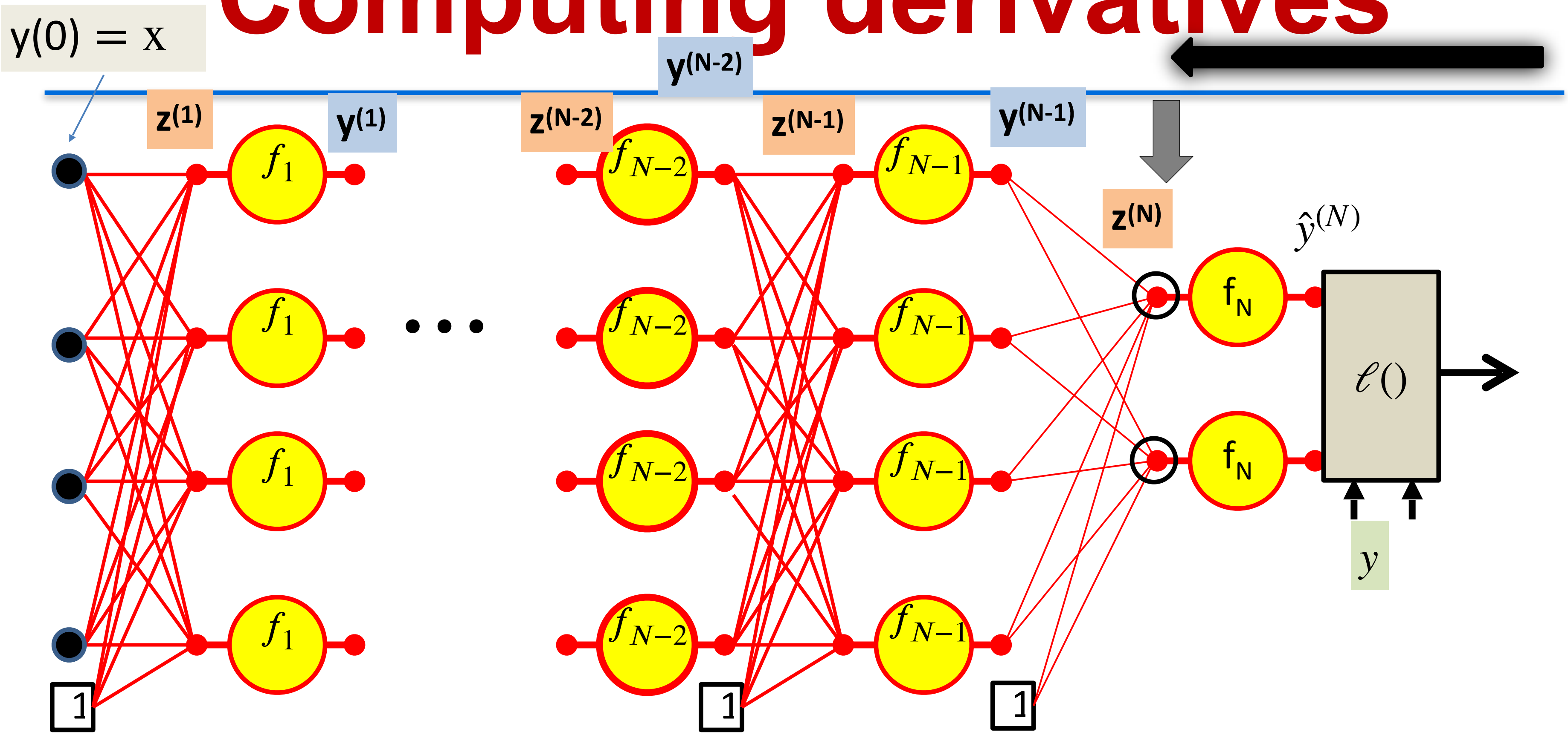
$$\boxed{\frac{dl}{dz} = \frac{dl}{dy}\frac{dy}{dz}}$$

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

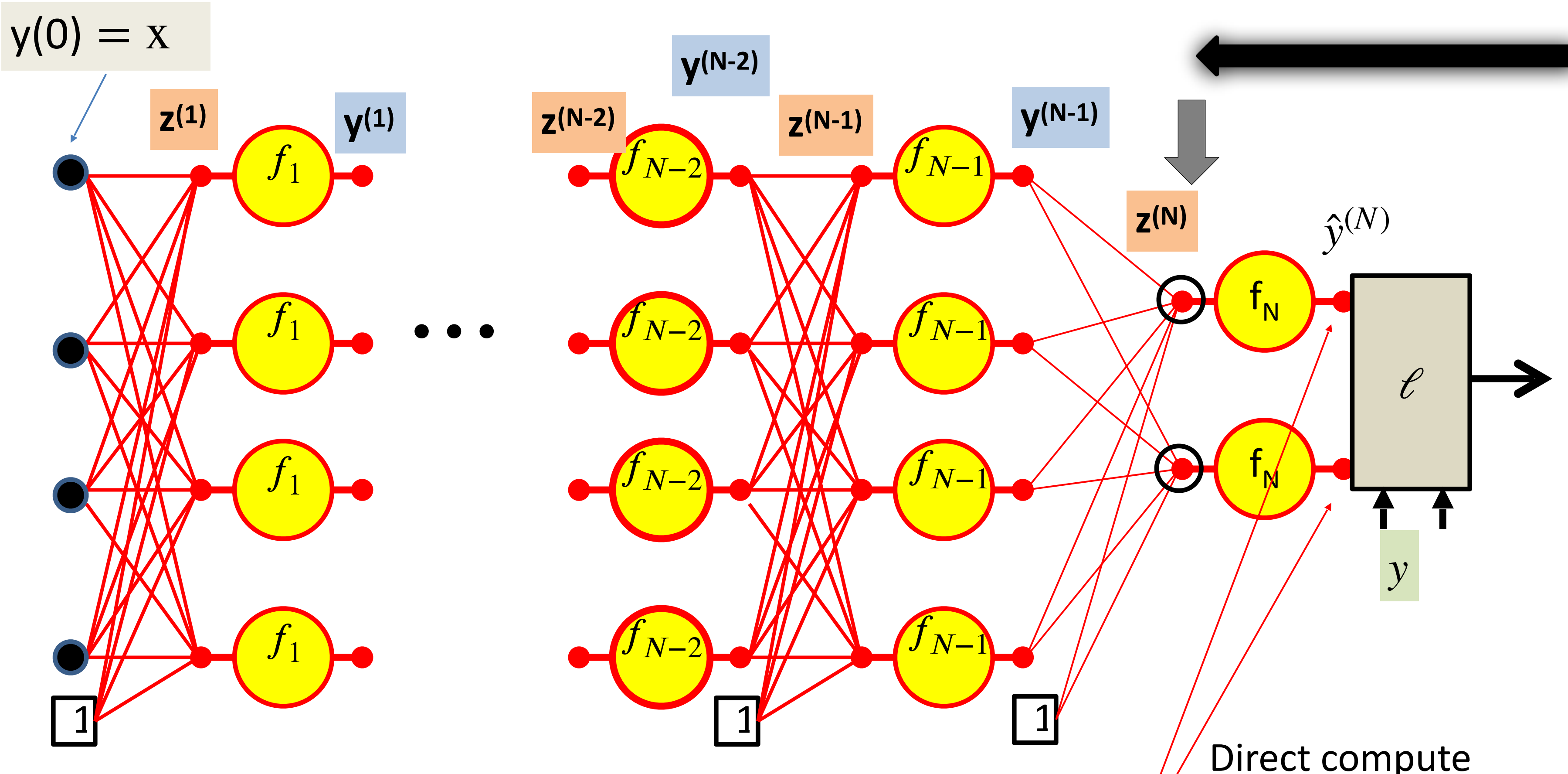


$$\boxed{\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx} + \ldots + \frac{\partial l}{\partial z_M}\frac{dz_M}{dx}}$$

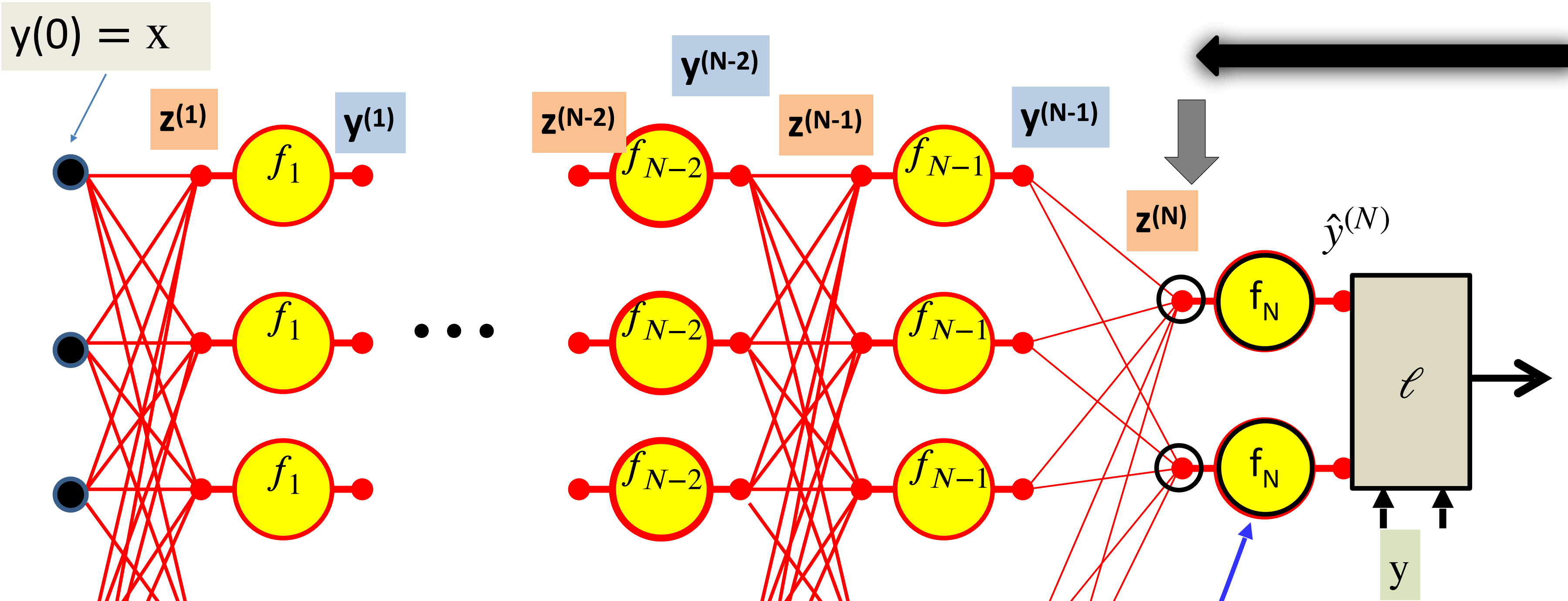
Computing derivatives



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



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

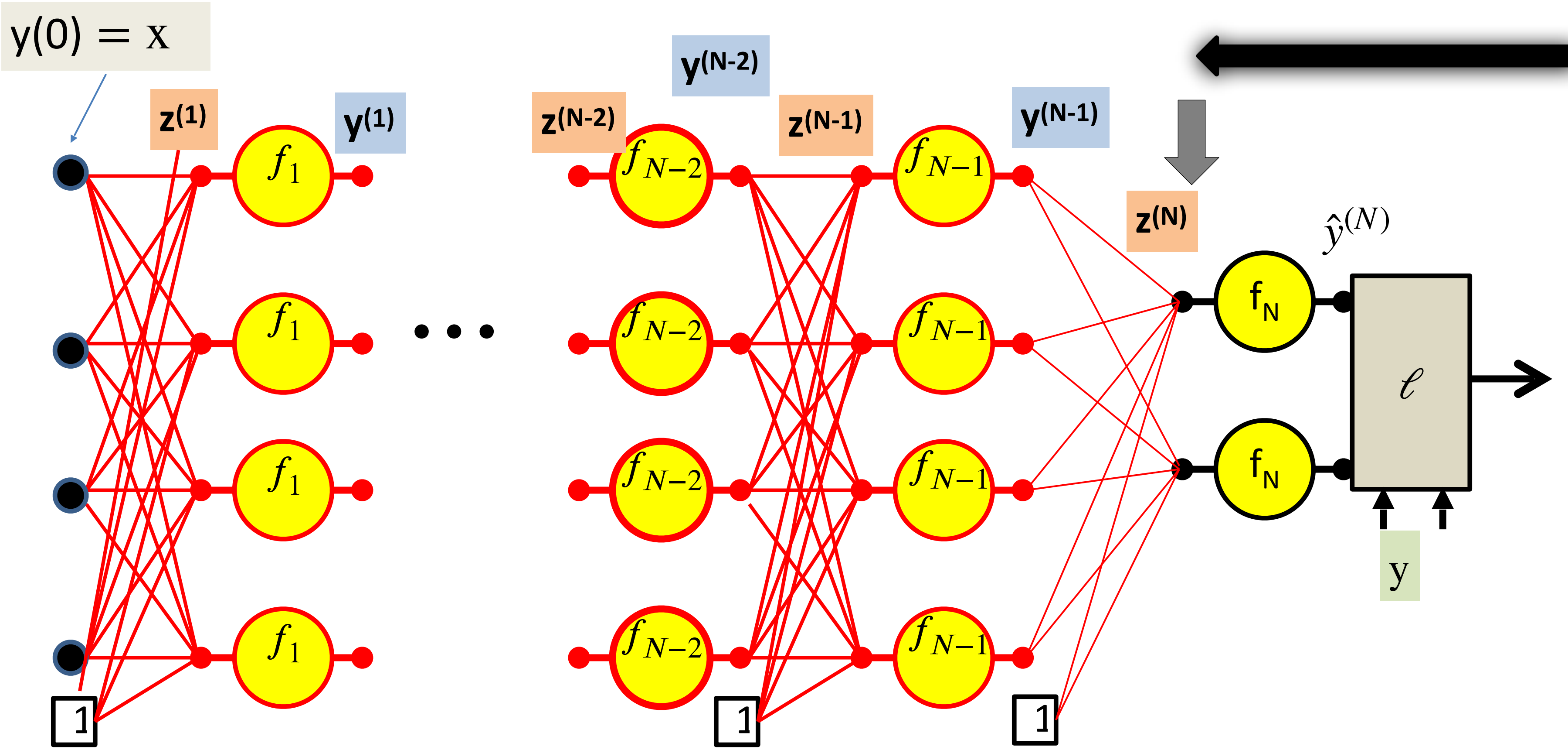


$$\frac{\partial \ell}{\partial z_i^{(N)}} = \frac{\partial \hat{y}}{\partial z_i^{(N)}} \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$

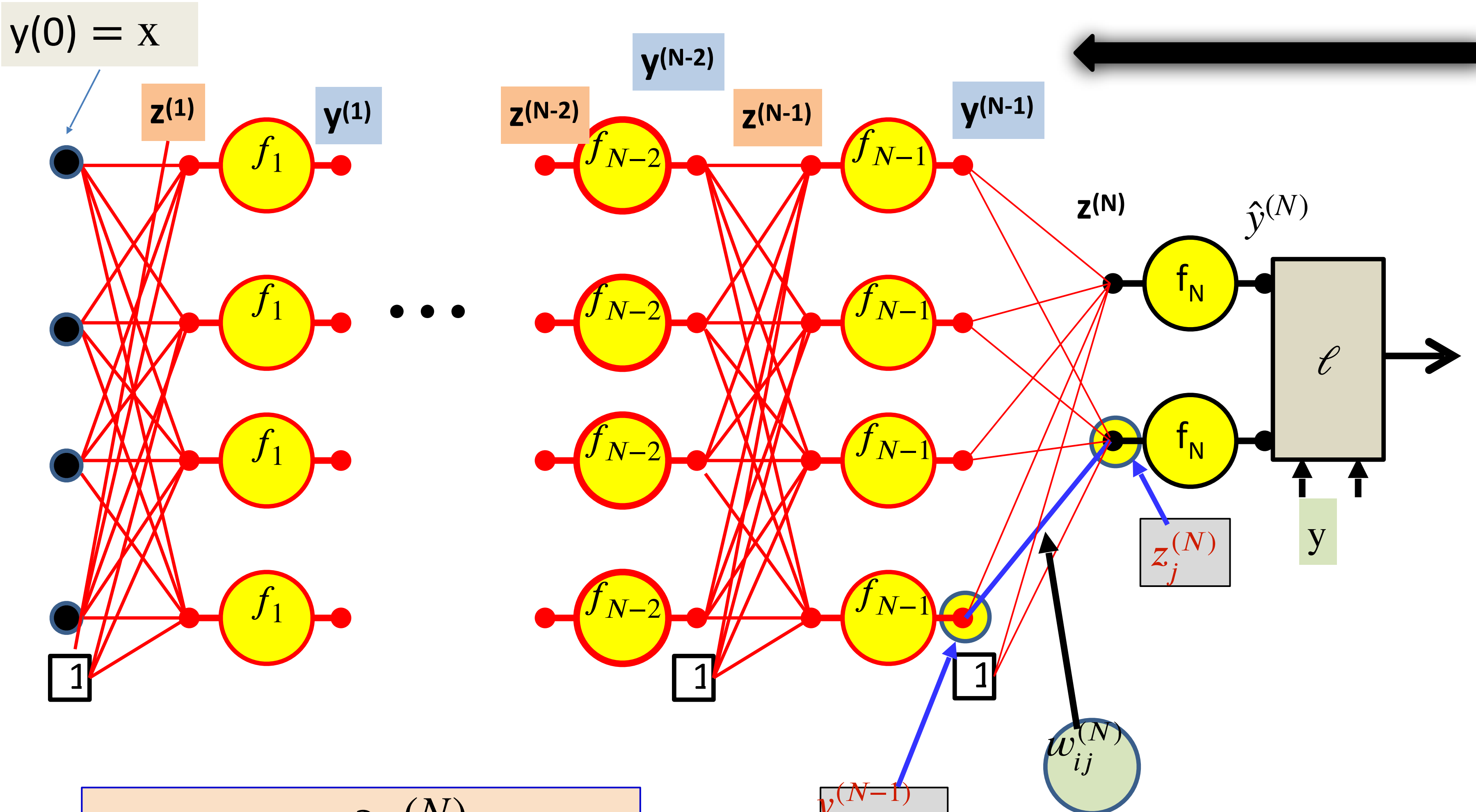
$f'_N\left(z_i^{(N)}\right)$

Derivative of activation function

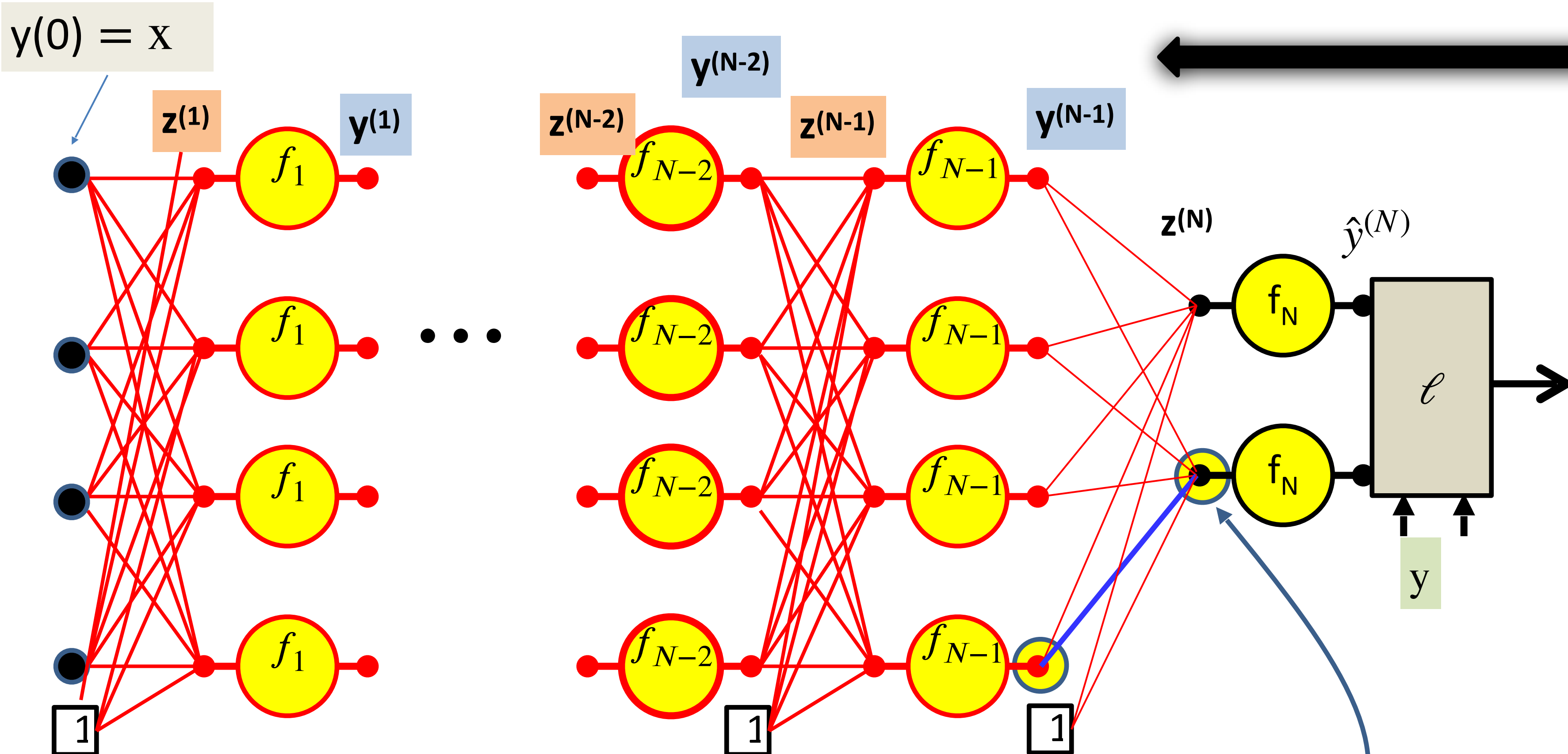
Computed in forward pass



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

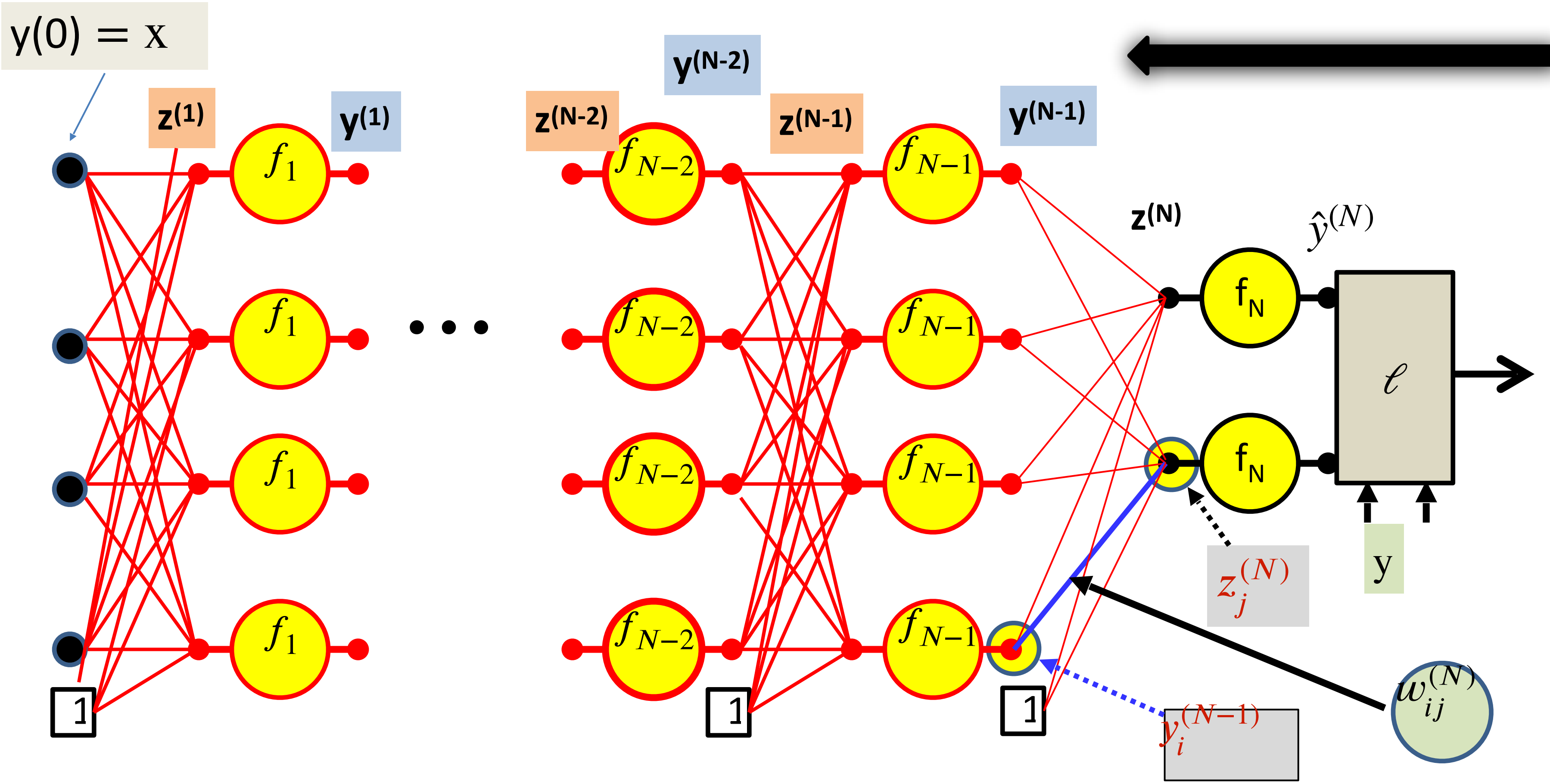


$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$



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

Just computed



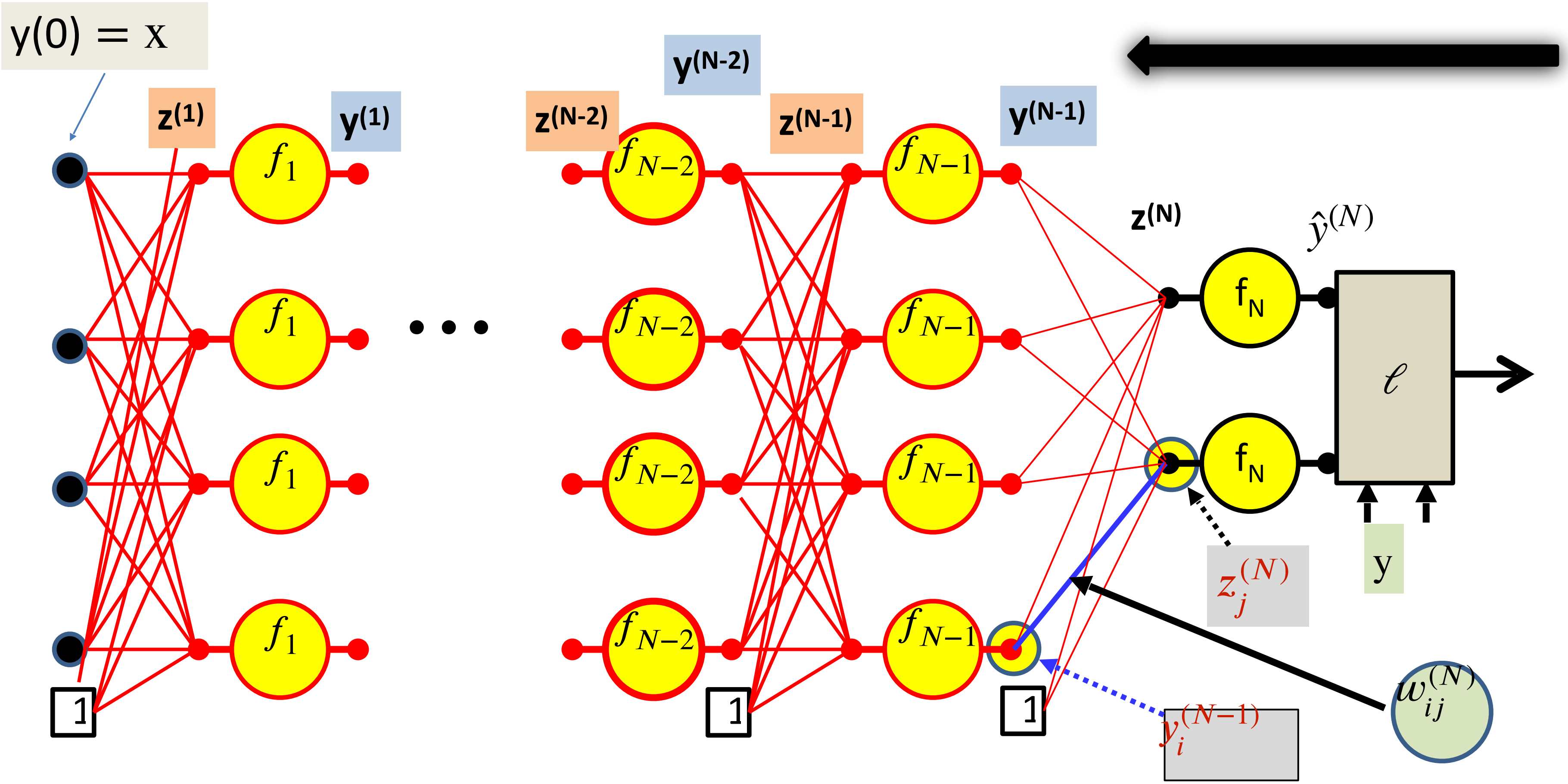
$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$

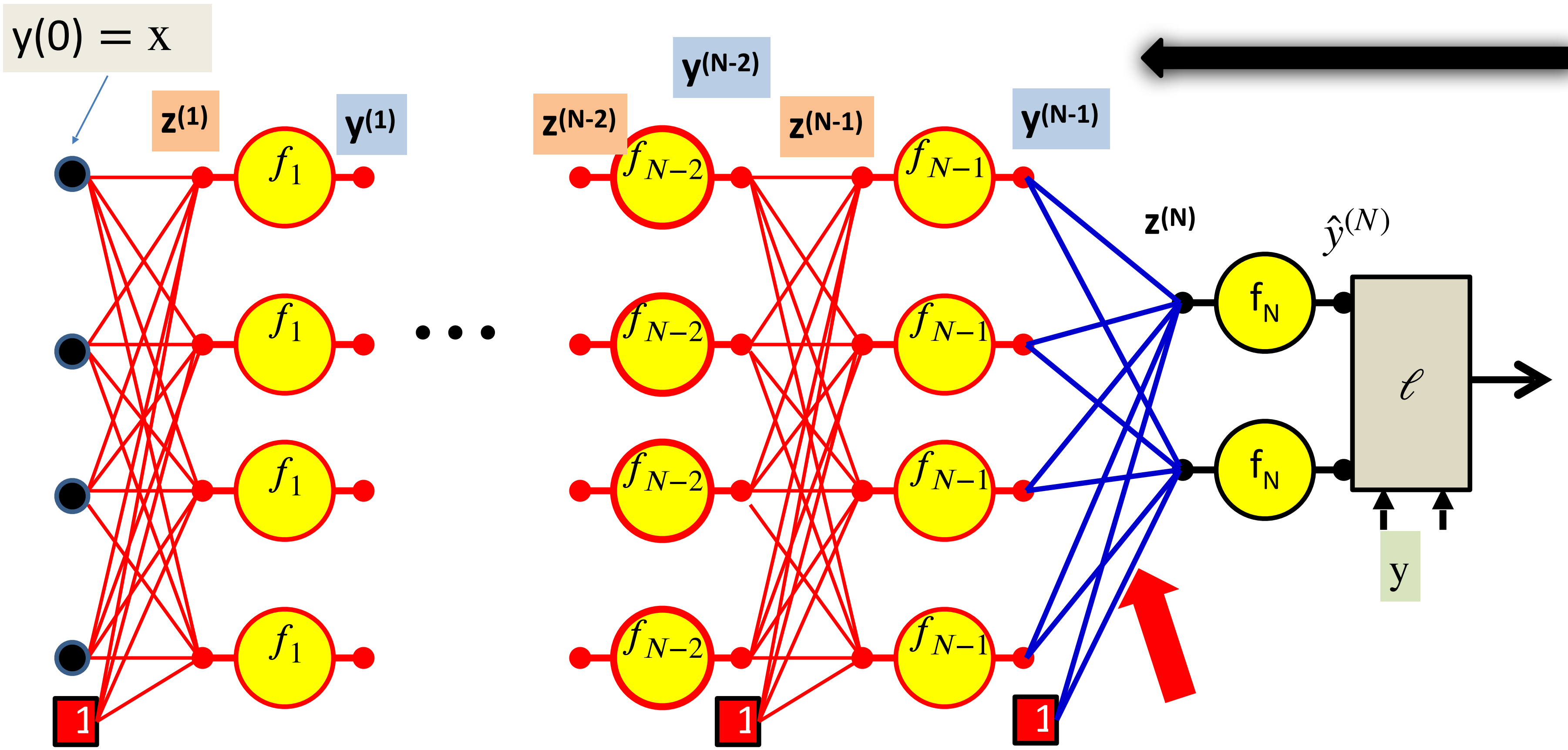
Computed in forward pass

Because

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

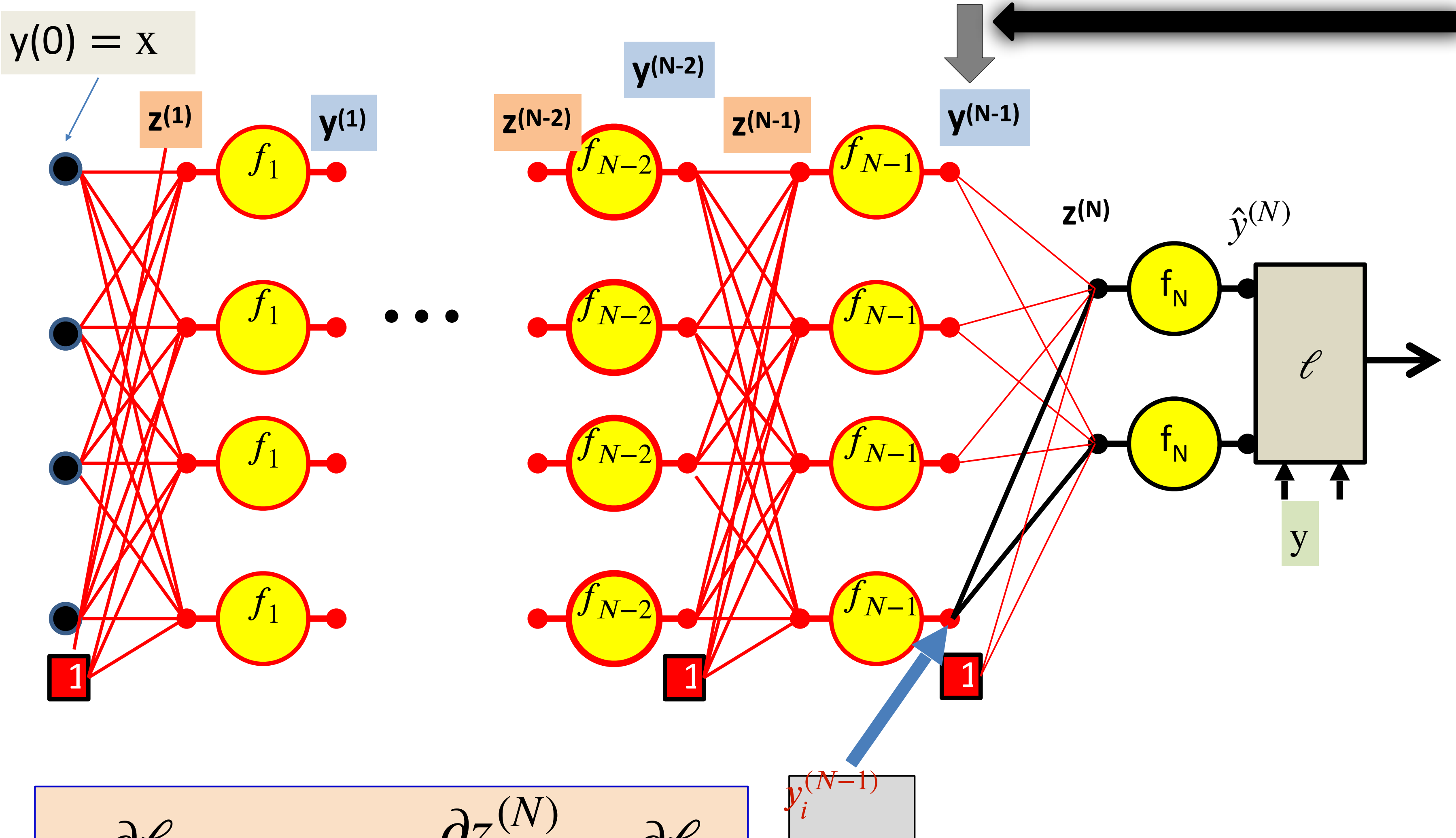


$$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}}$$

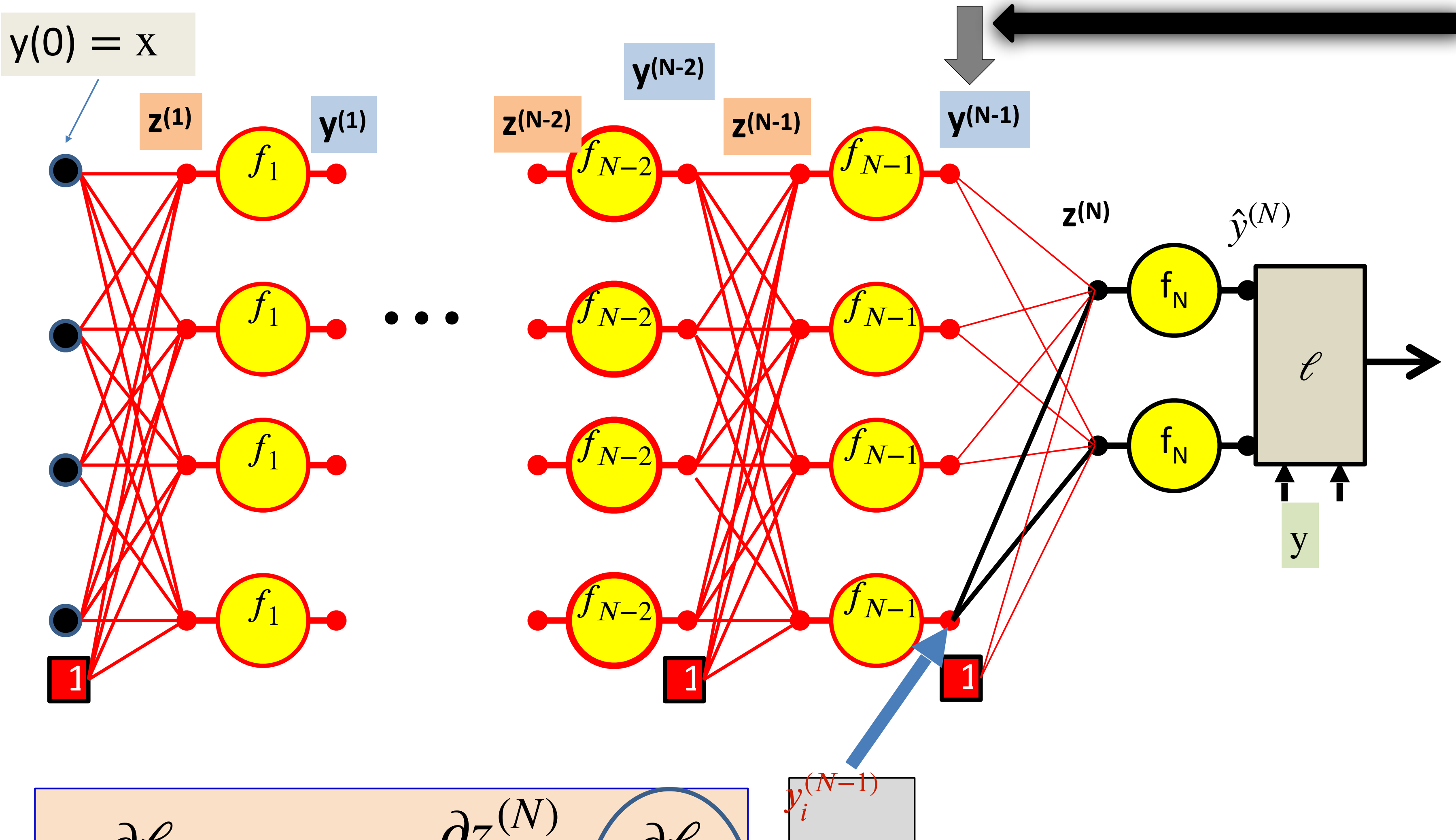


$$\boxed{\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}}}$$

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

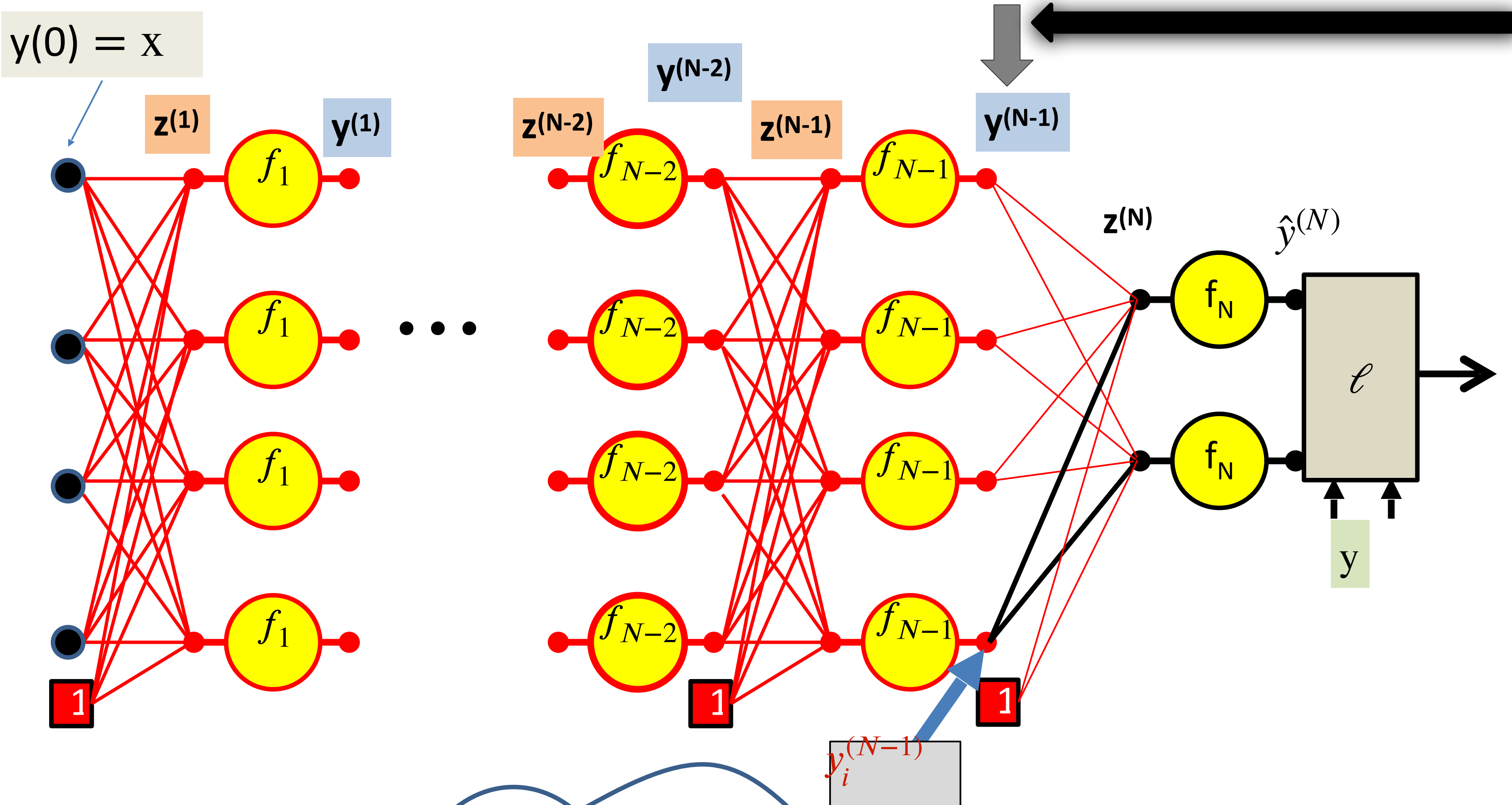


$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$



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

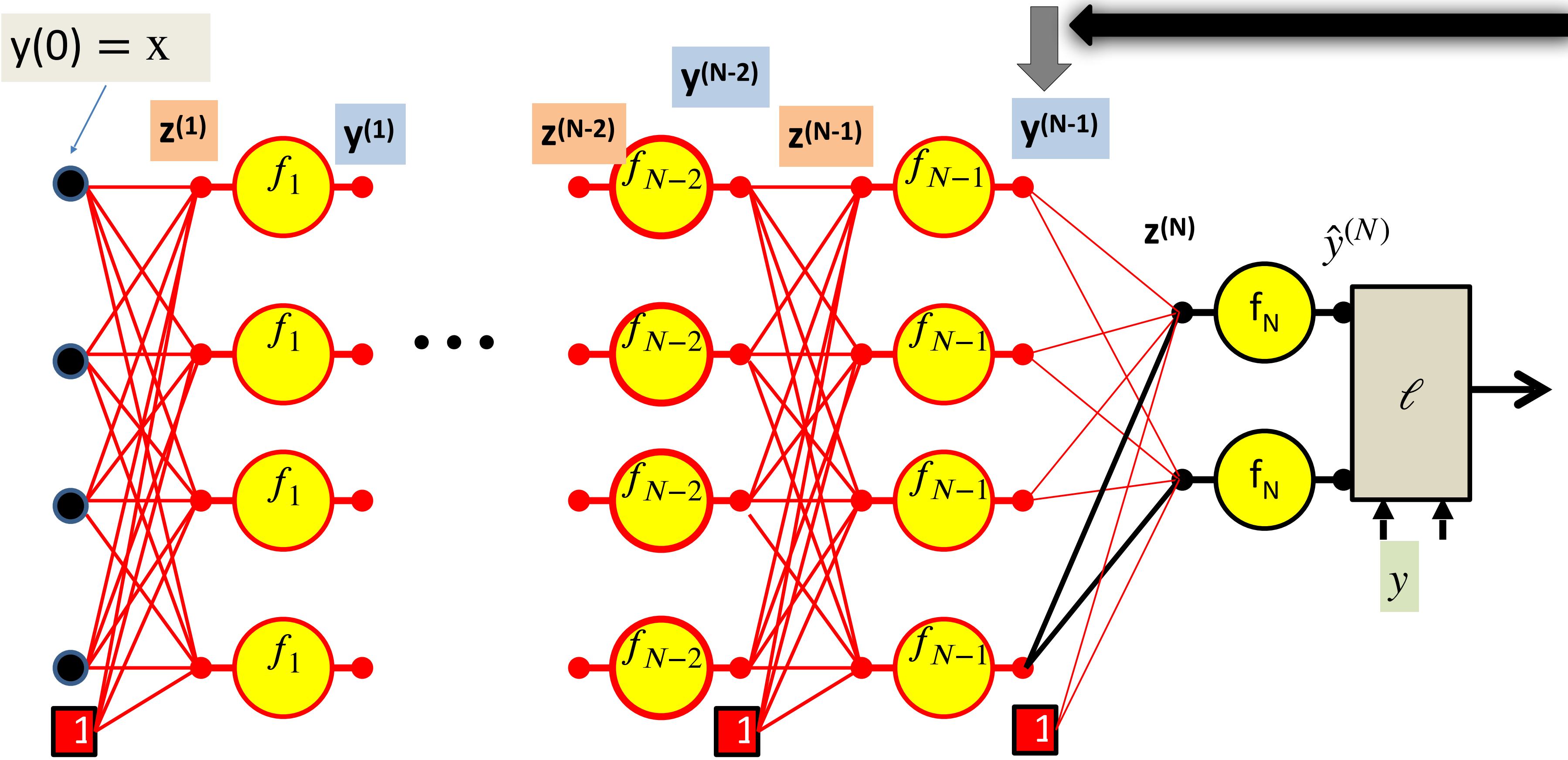
Already computed



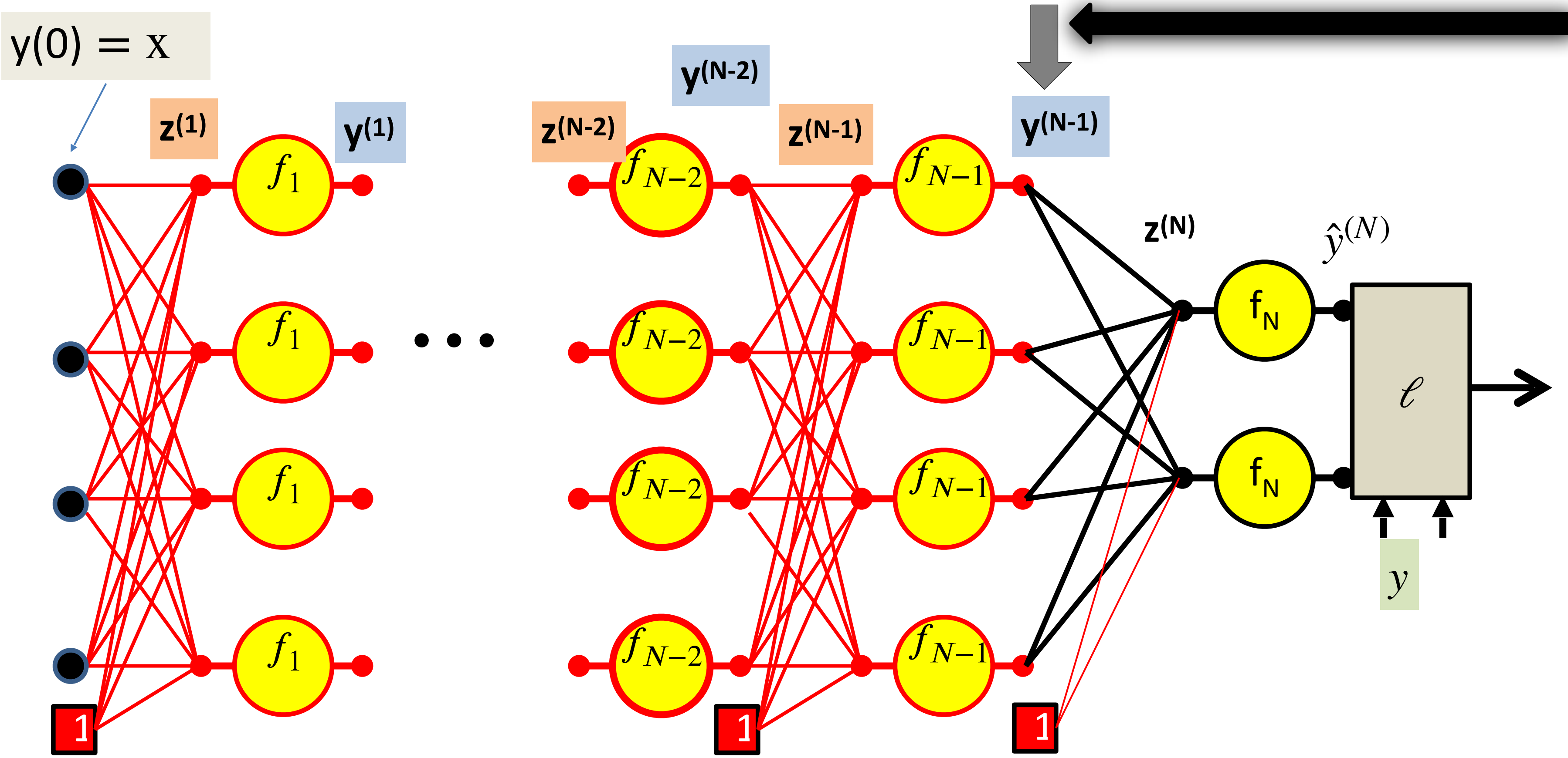
$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial \ell}{\partial z_j^{(N)}}$$

Because

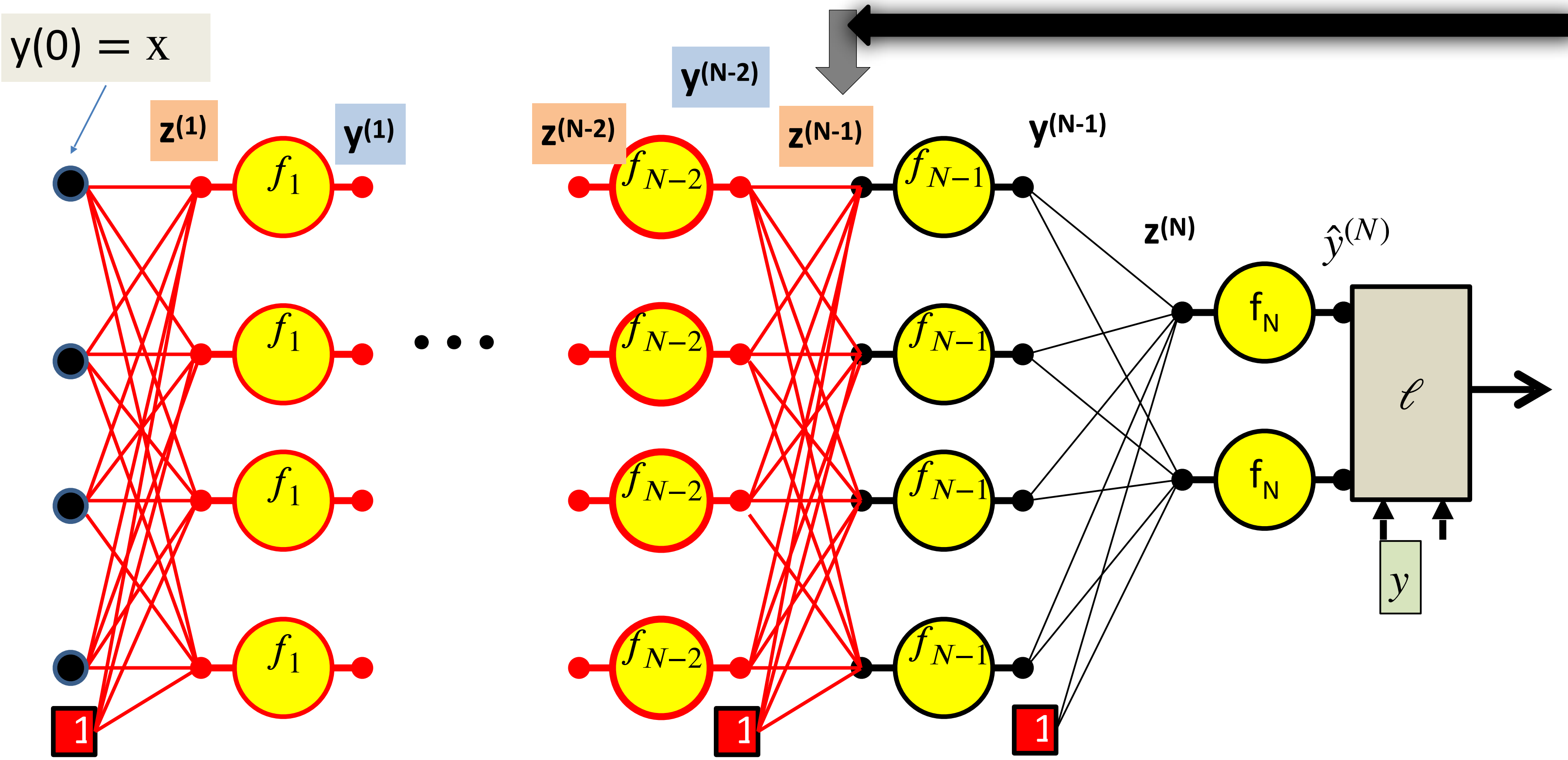
$$z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$$



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

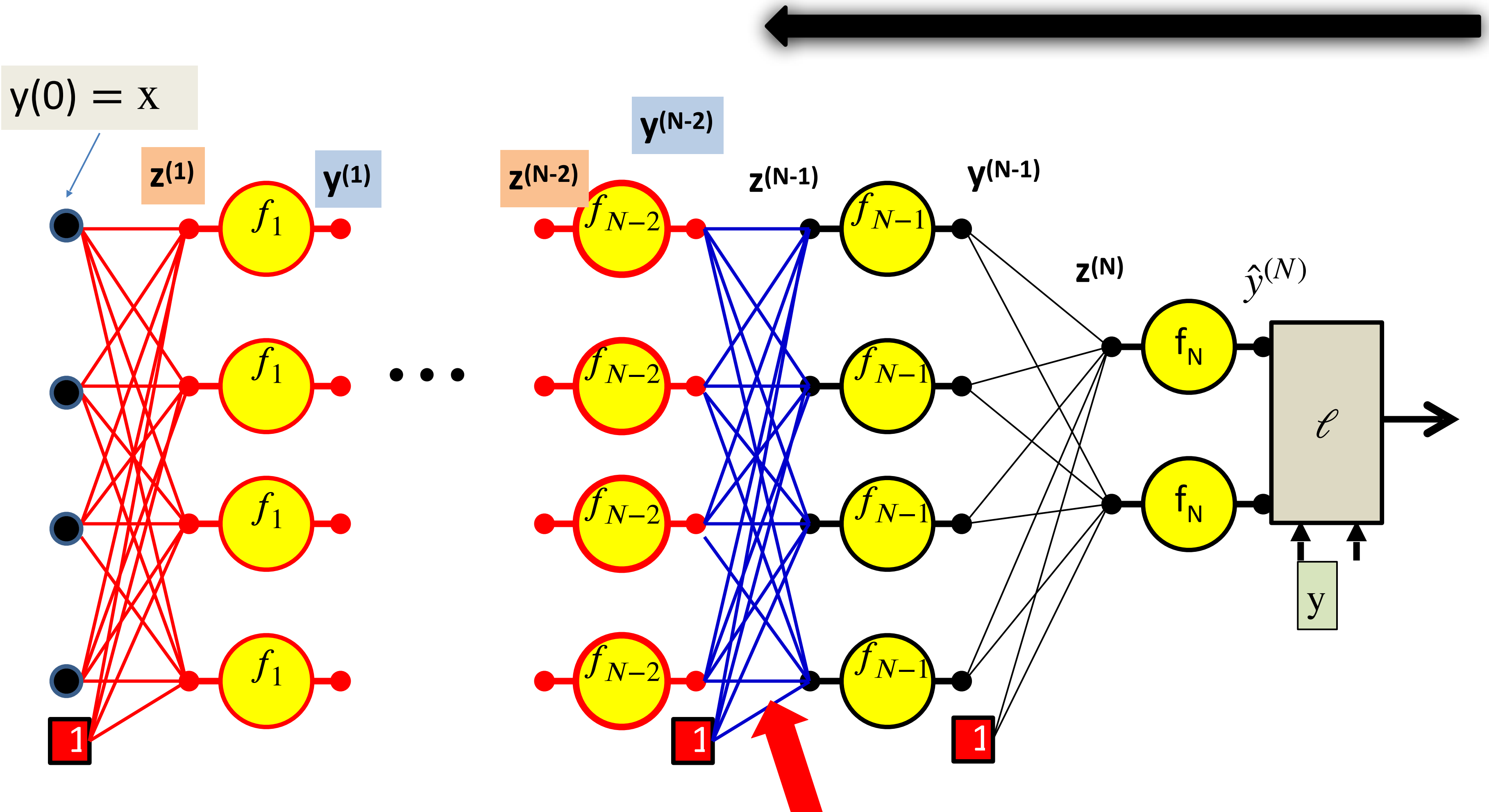


$$\frac{\partial \ell}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial \ell}{\partial z_j^{(N)}}$$



We continue our way backwards in the order shown

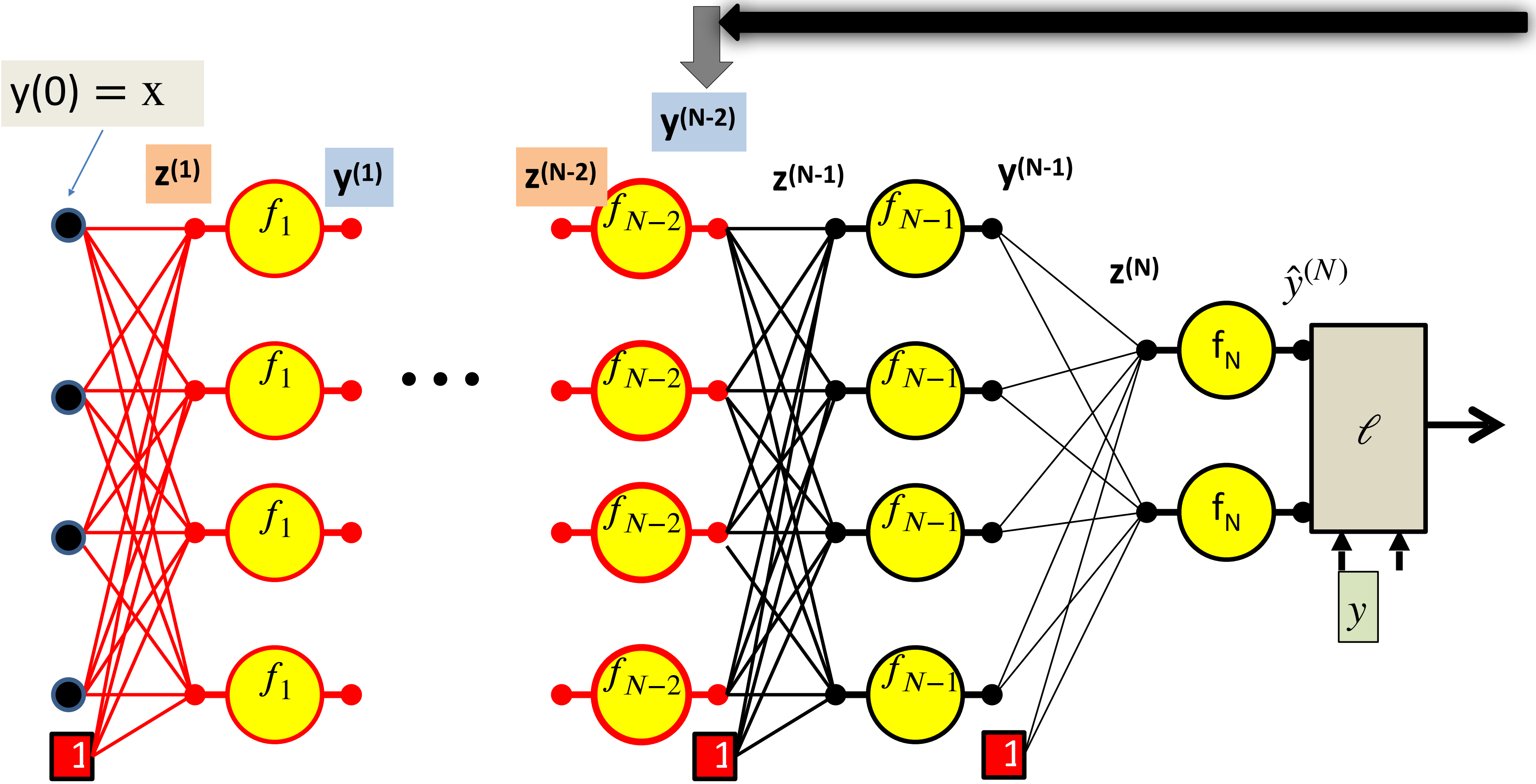
$$\frac{\partial \ell}{\partial z_i^{(N-1)}} = f'_{N-1}(z_i^{(N-1)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N-1)}}$$



We continue our way backwards in the order shown

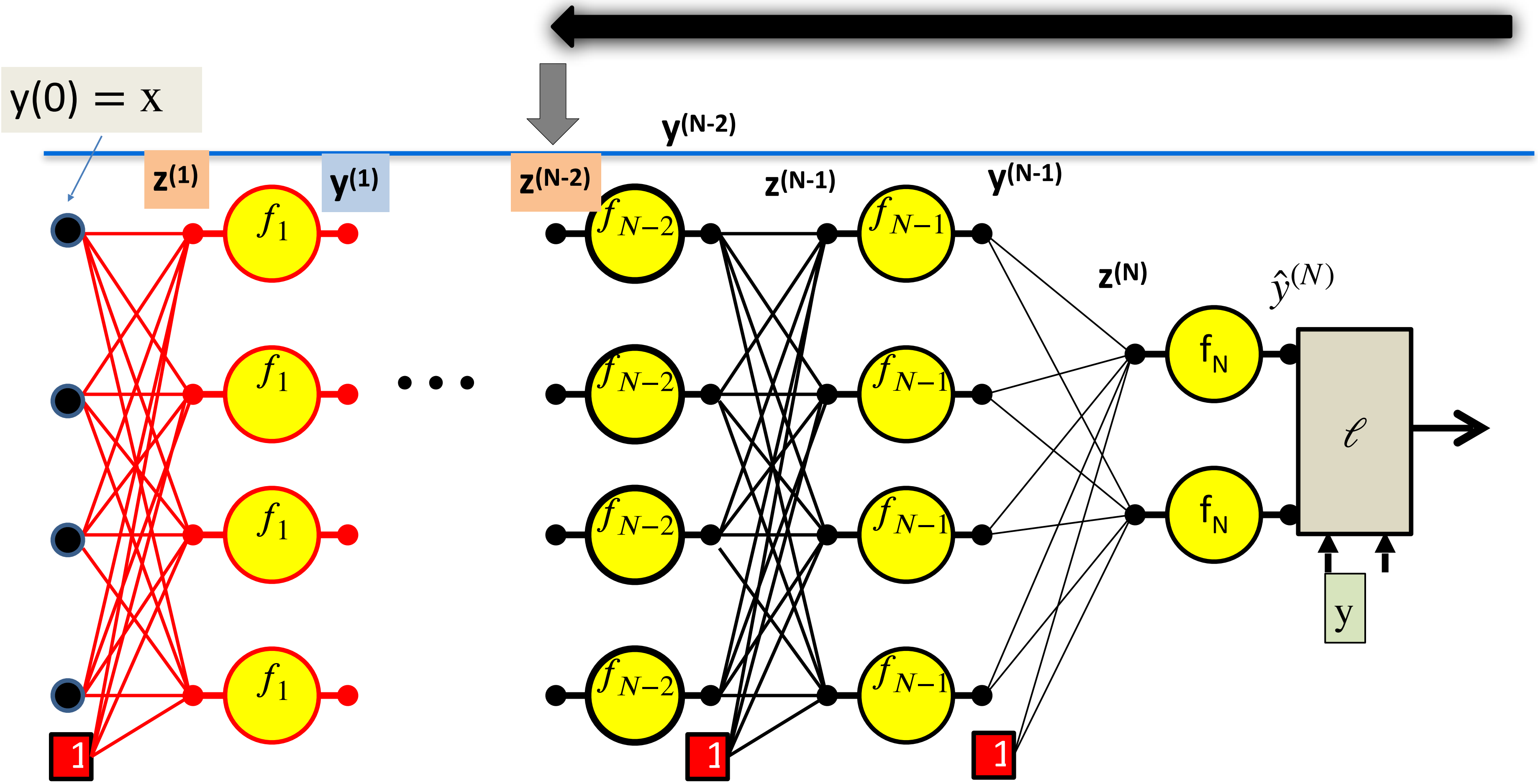
$$\boxed{\frac{\partial \ell}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial \ell}{\partial z_j^{(N-1)}}}$$

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



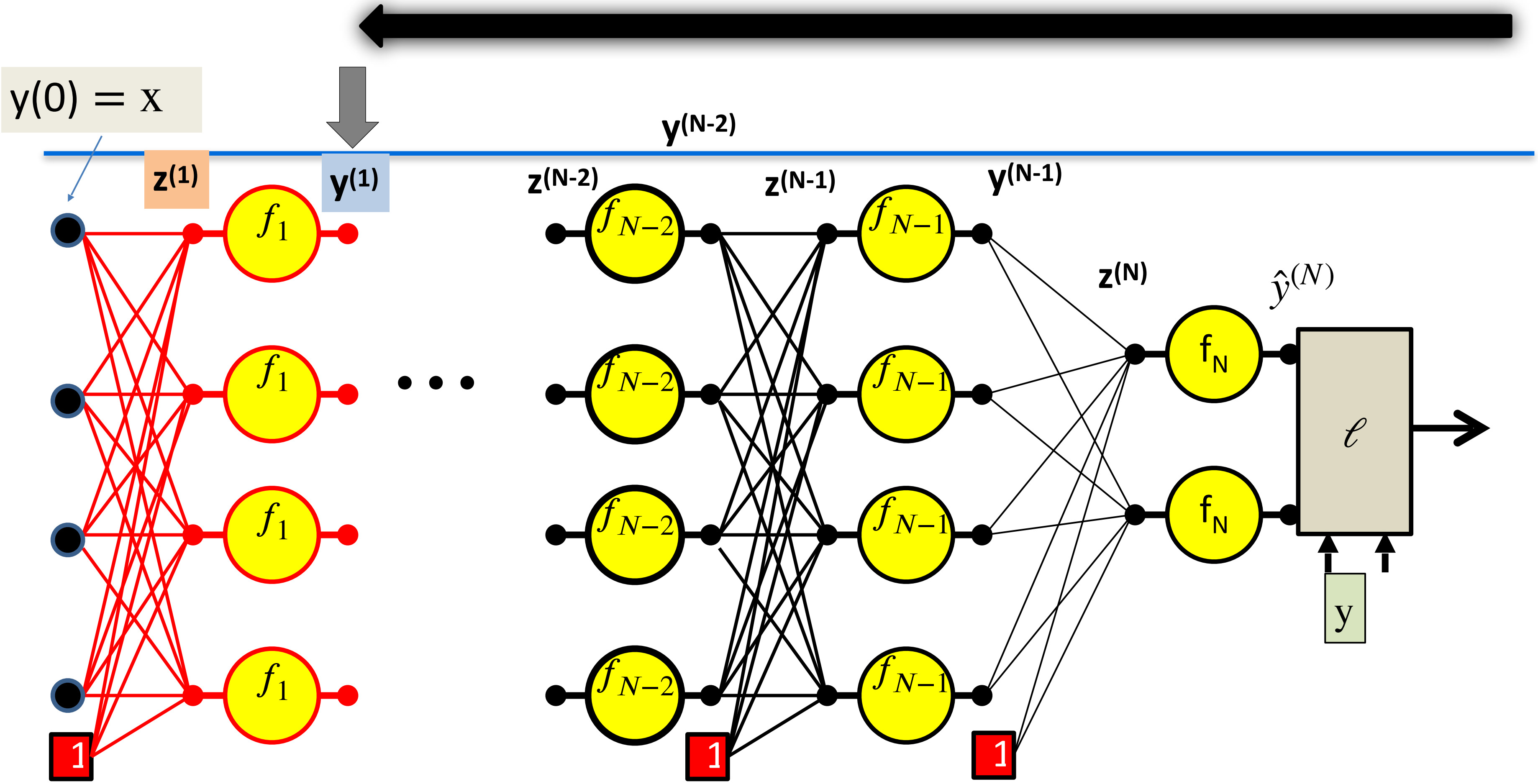
We continue our way backwards in the order shown

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



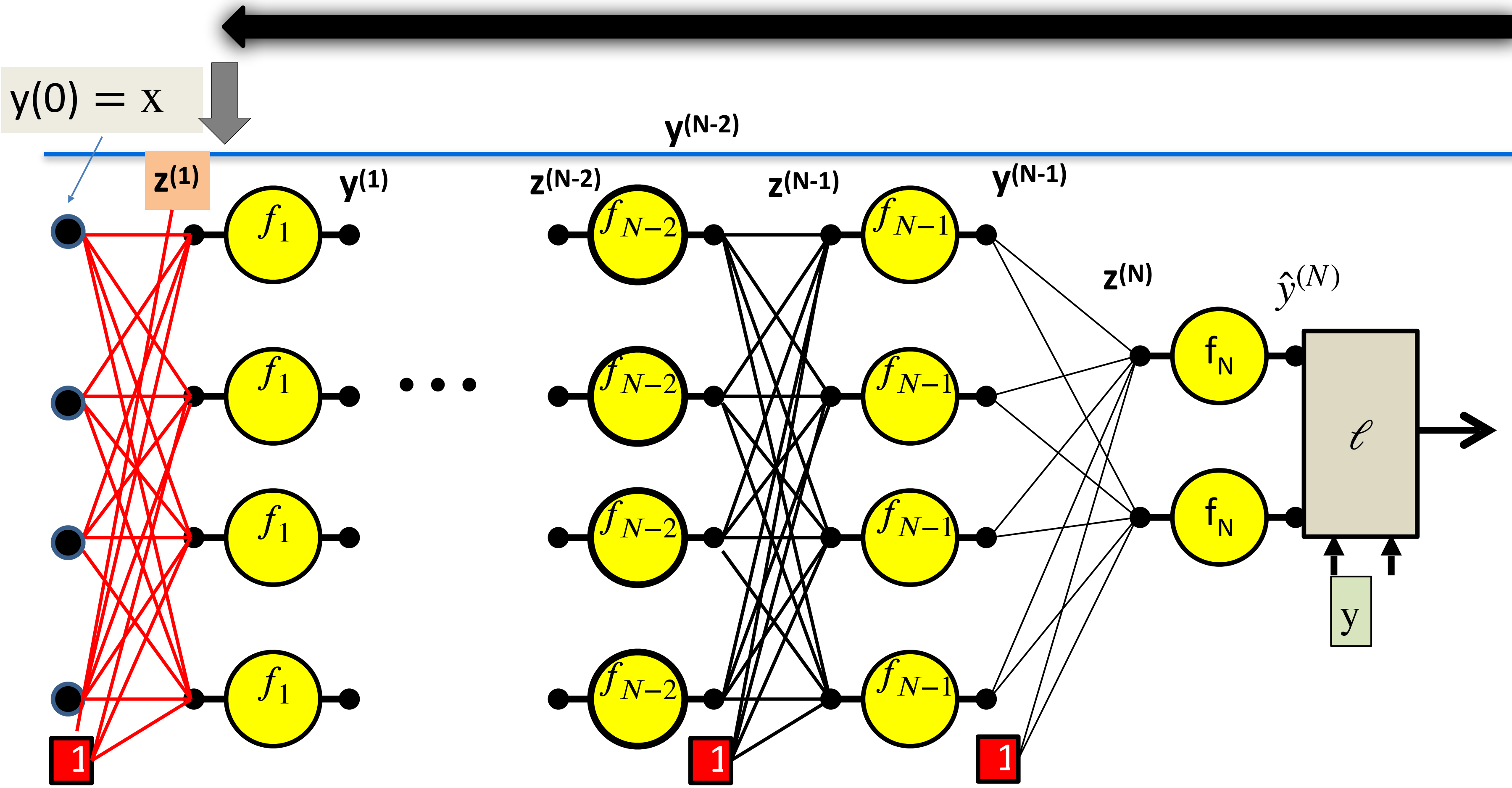
We continue our way backwards in the order shown

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



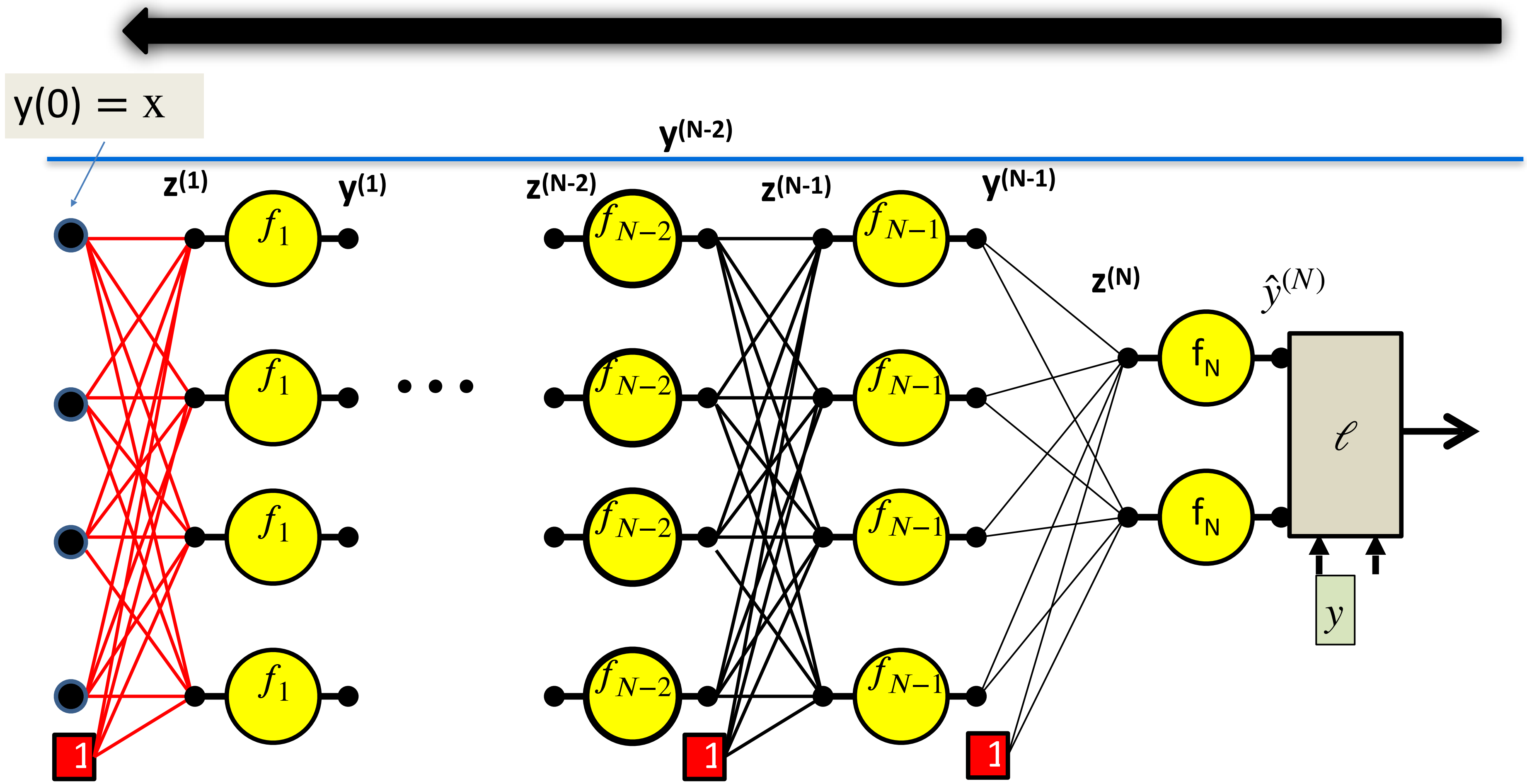
We continue our way backwards in the order shown

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



We continue our way backwards in the order shown

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



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 $i = 1 \dots D_N$
    - $$\frac{\partial \ell}{\partial z_i^{(N)}} = f_N'(z_i^{(N)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$
    - $$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}} \text{ for each j}$$
- For layer $k = N - 1 \; downto \; 1$
  - For $i = 1 \dots D_k$
    - $$\frac{\partial \ell}{\partial y_i^{(k-1)}} = \sum_j w_{ij}^{(k)} \frac{\partial \ell}{\partial z_j^{(k)}}$$
    - $$\frac{\partial \ell}{\partial z_i^{(k)}} = f_k'(z_i^{(k)}) \frac{\partial \ell}{\partial y_i^{(k)}}$$
    - $$\frac{\partial \ell}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial \ell}{\partial z_j^{(k)}} \text{ for each j}$$

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

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)
)

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')
```

Summary

- Single artificial neuron
- Logistic Regression and its limitation
- Feedforward neural network (multilayer perceptron)
- Successful example of FFN: Deep&Wide model
- Computing Gradient for FFN — backpropagation

Next Up

- Lecture 6: Convolutional neural network
  - Application in image classification and object detection
- Lecture 7: Sequence modelling, recurrent neural networks
- Lecture 8: Transformer (very powerful model)
  - pretraining