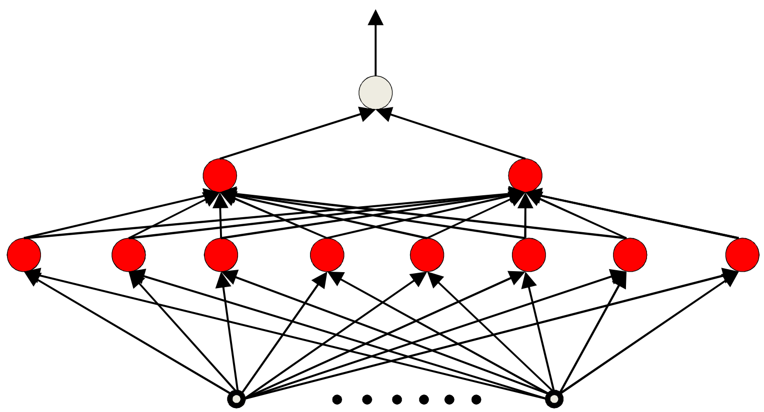
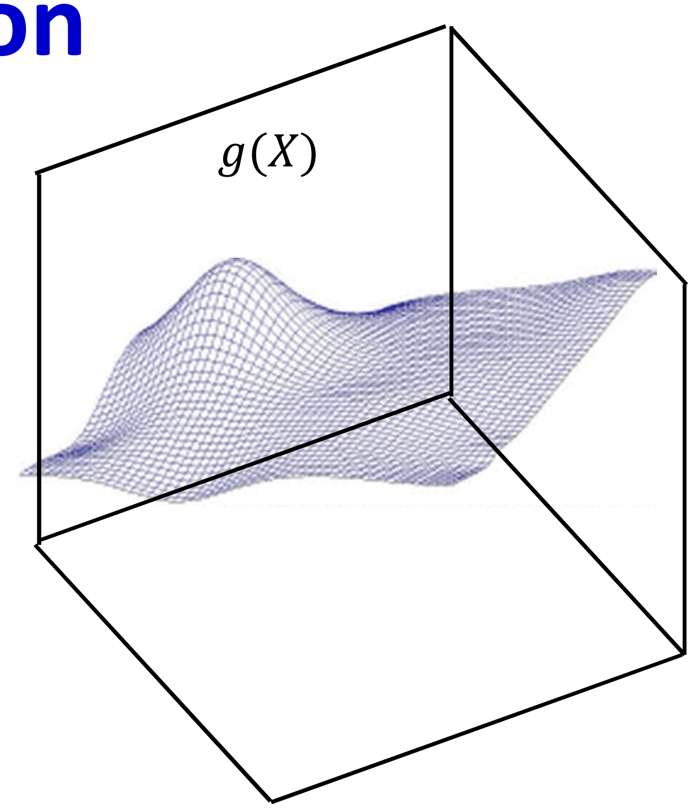
Neural Networks
Learning the network: Backprop

11-785, Spring 2020

Lecture 4

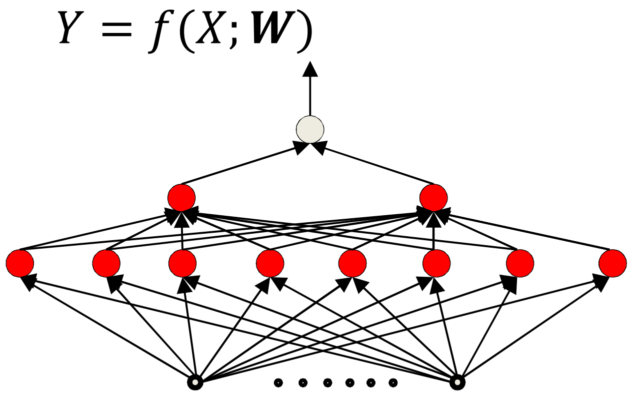
Recap: The MLP *can* represent any function

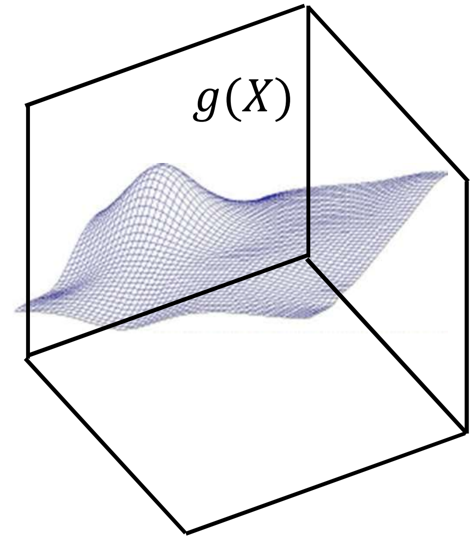




- The MLP *can be constructed* to represent anything
- But *how* do we construct it?
  - *I.e.* how do we determine the weights (and biases) of the network to best represent a target function
    - *Assuming that the architecture of the network is given*

Recap: How to learn the function



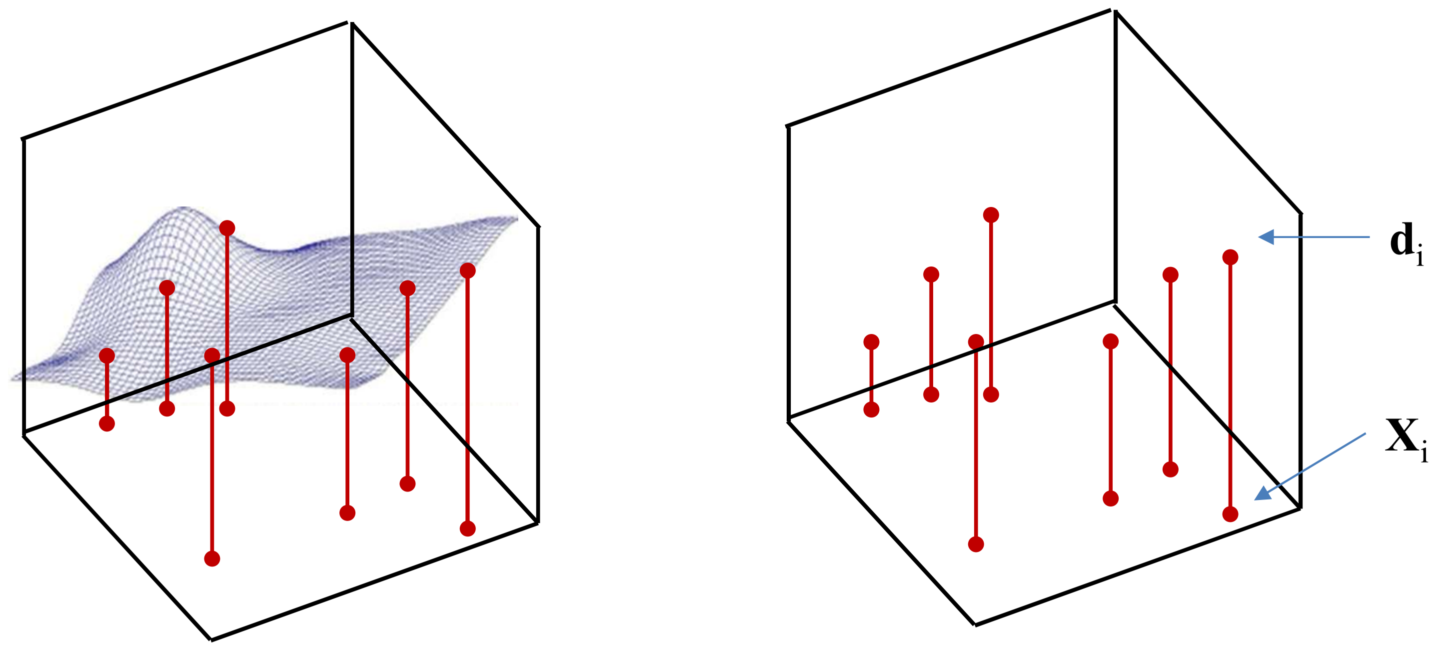


- By minimizing expected error

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}} \int_X div\big(f(X; W), g(X)\big)P(X)dX$$

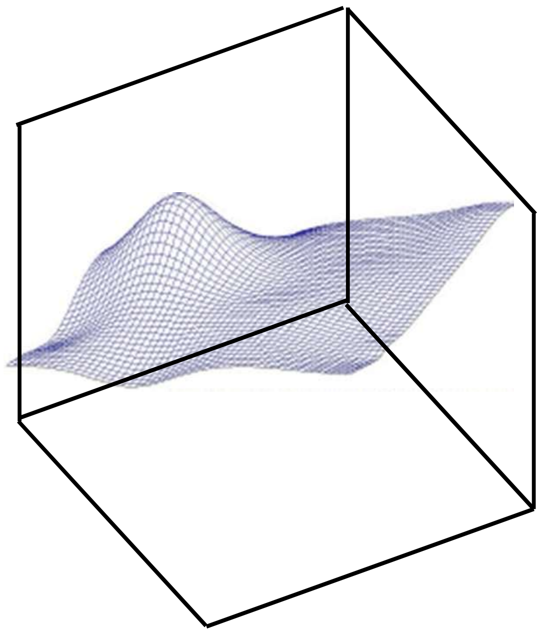
$$= \underset{W}{\operatorname{argmin}}\, E\big[div\big(f(X; W), g(X)\big)\big]$$

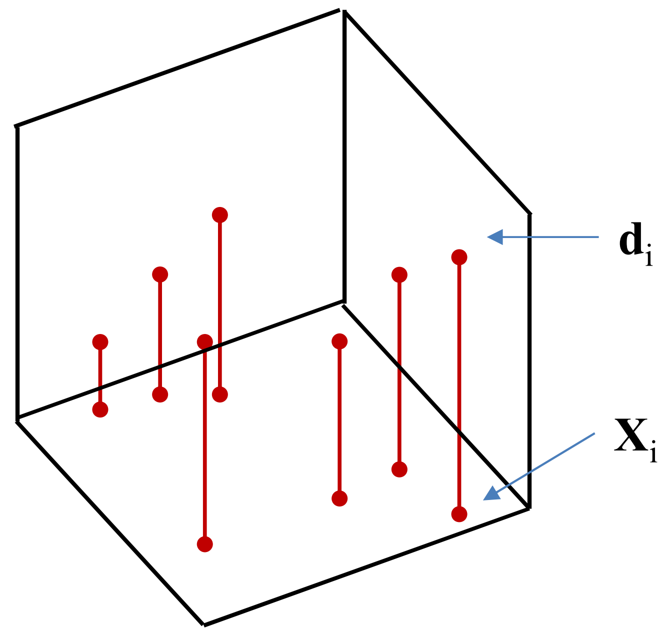
Recap: Sampling the function



- *$g(X)$ is unknown, so sample it*
  - Basically, get input-output pairs for a number of samples of input X_i
  - Good sampling: the samples of X will be drawn from $P(X)$
- Estimate function from the samples

The *Empirical* risk



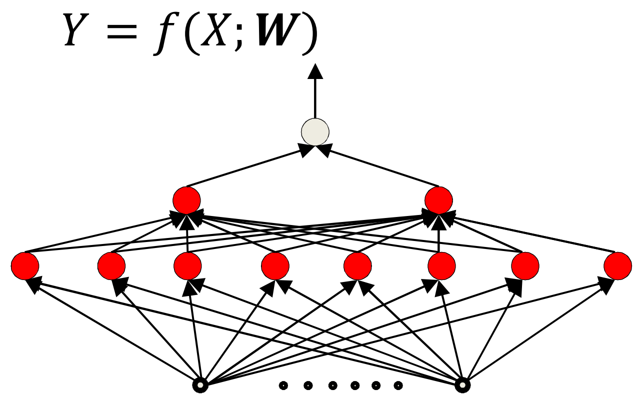


- The *empirical estimate* of the expected error is the *average* error over the samples

$$E\left[div\left(f(X;W),g(X)\right)\right] \approx \frac{1}{T}\sum_{i=1}^{T} div(f(X_i;W),d_i)$$

- This approximation is an unbiased estimate of the *expected* divergence that we *actually* want to estimate
  – We can *hope* that minimizing the empirical loss will minimize the true loss
  – Caveat: This hope is generally not based on anything but, well, hope..

Empirical Risk Minimization



- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
  - Error on the i-th instance: $div(f(X_i; W), d_i)$
  - Empirical average error on all training data:

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

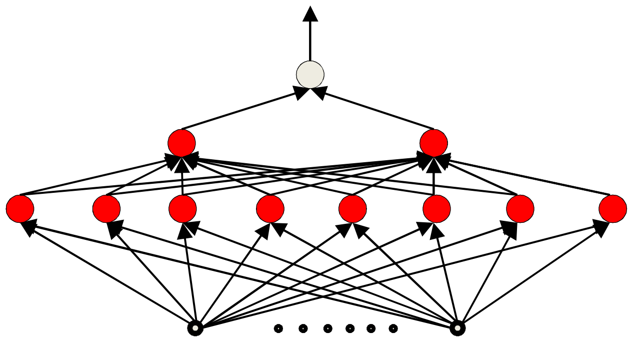
- Estimate the parameters to minimize the empirical estimate of expected error

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}}\, Loss(W)$$

  - I.e. minimize the *empirical error* over the drawn samples

Empirical Risk Minimization

$Y = f(X; \boldsymbol{W})$



This is an instance of function minimization (optimization)

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
  - Error on the i-th instance: $div(f(X_i; W), d_i)$
  - Empirical average error on all training data:

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

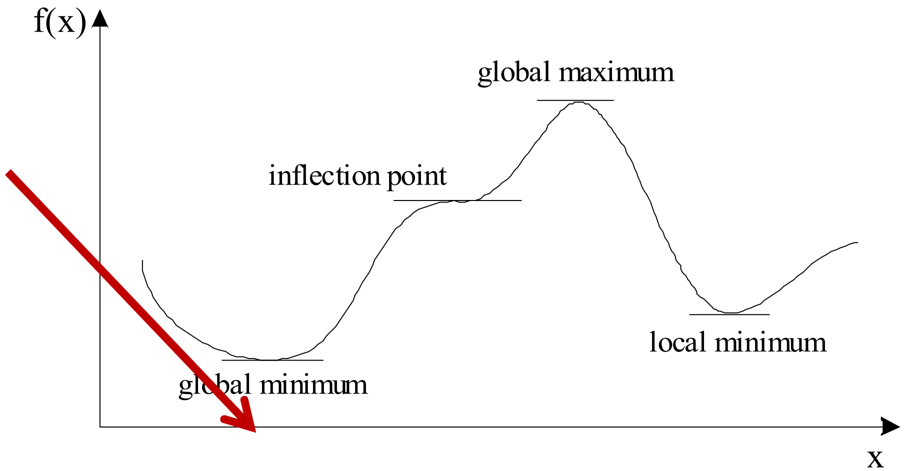
- Estimate the parameters to minimize the empirical estimate of expected error

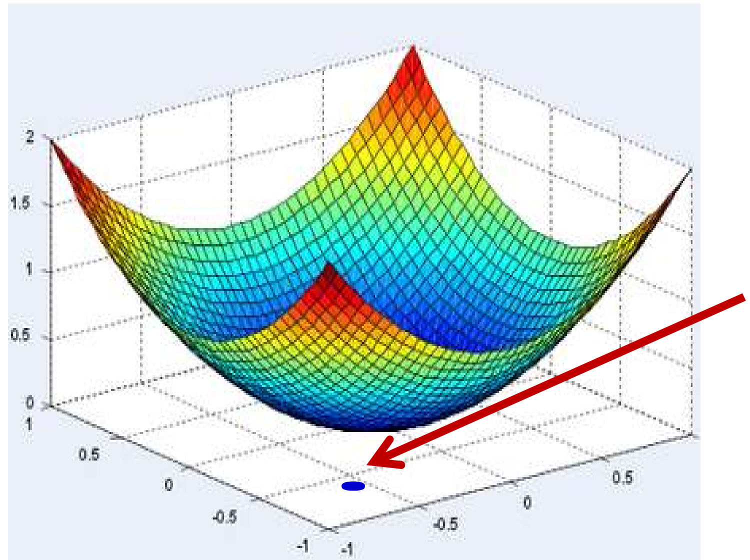
$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}}\, Loss(W)$$

  - I.e. minimize the *empirical error* over the drawn samples

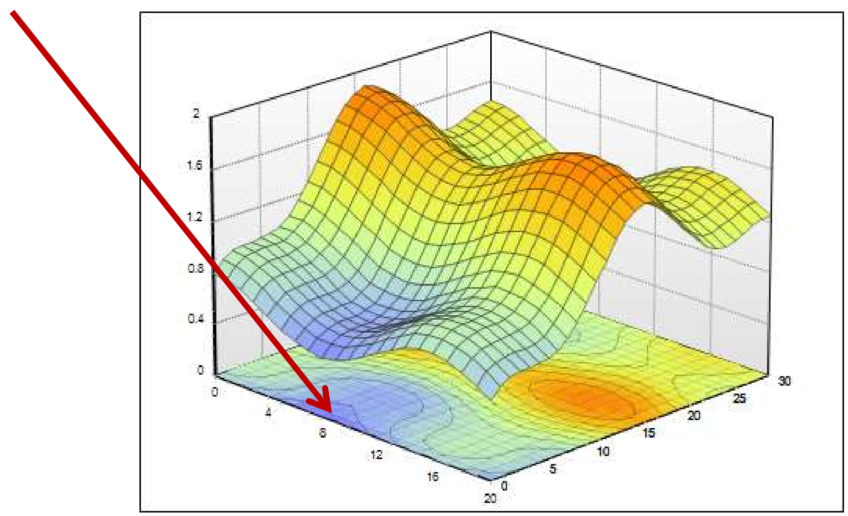
- **A CRASH COURSE ON FUNCTION OPTIMIZATION**

The problem of optimization

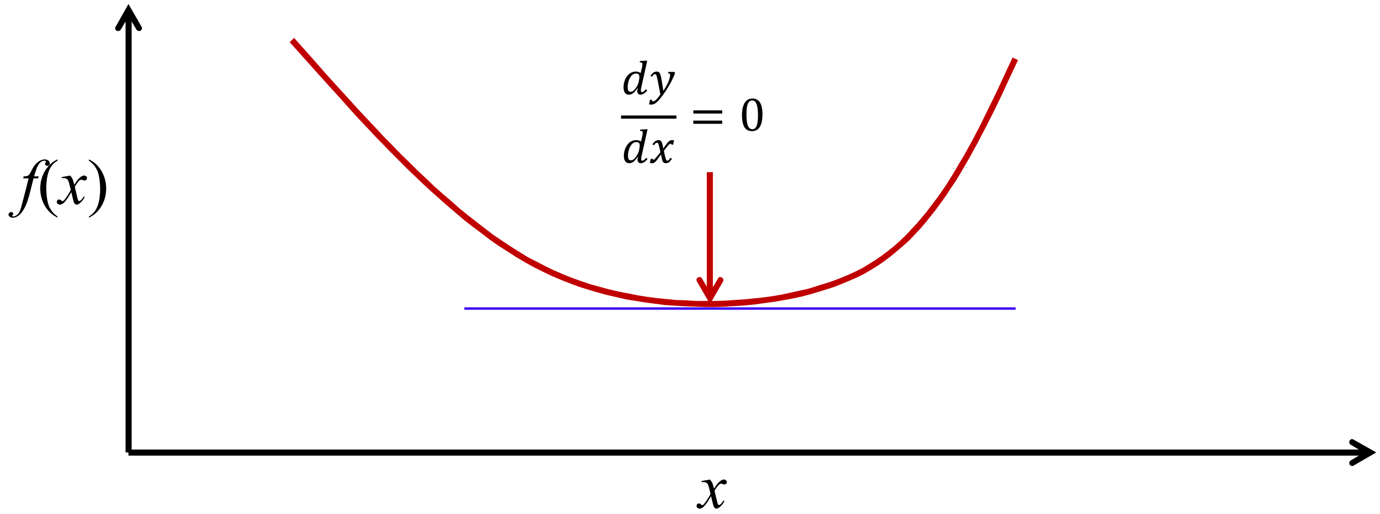




- General problem of optimization: find the value of x where $\mathbf{f}(x)$ is minimum



Finding the minimum of a function

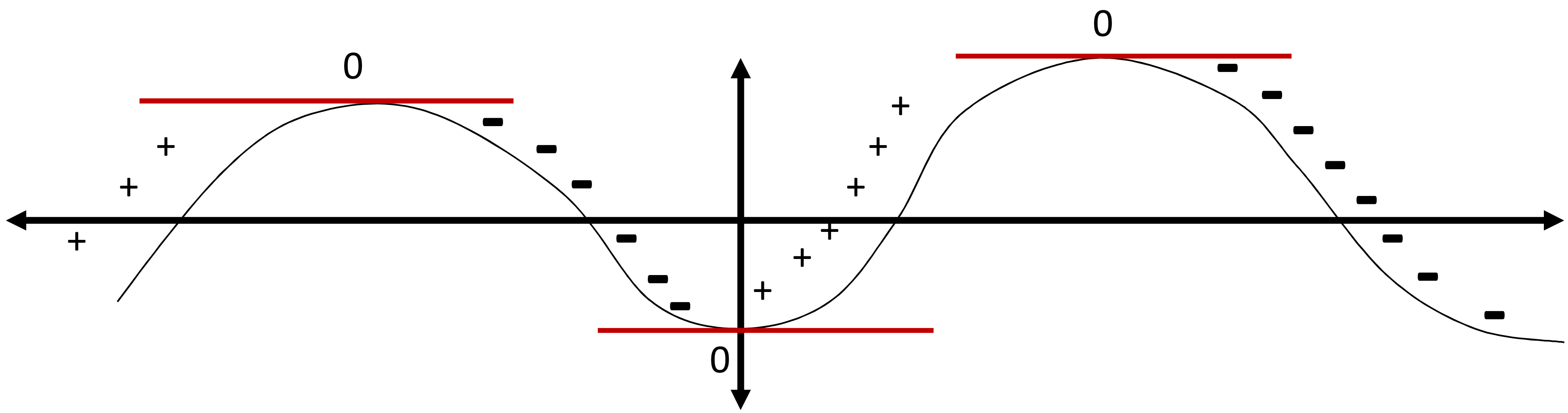


- Find the value x at which $f'(x) = 0$
  - Solve

$$\frac{df(x)}{dx} = 0$$

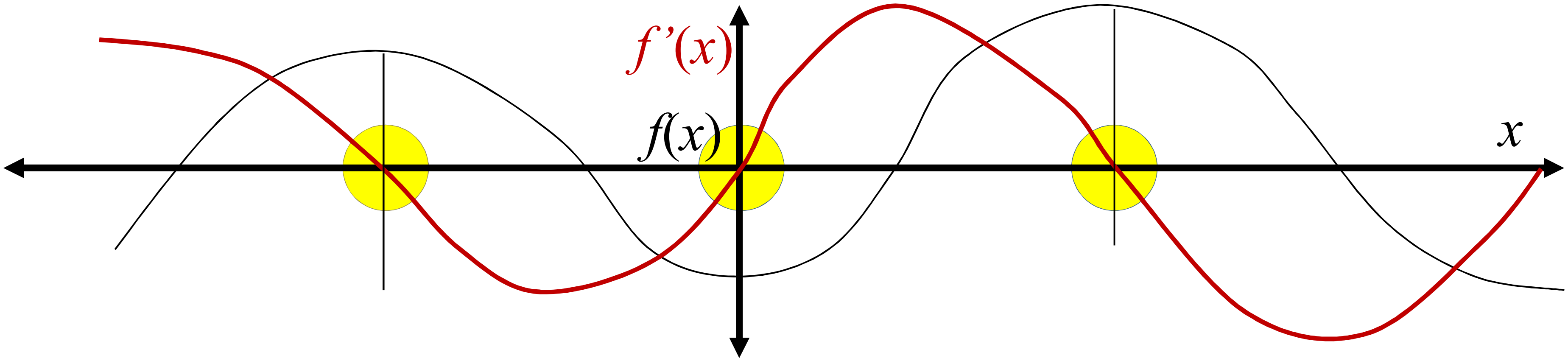
- The solution is a "turning point"
  - Derivatives go from positive to negative or vice versa at this point
- But is it a minimum?

Turning Points



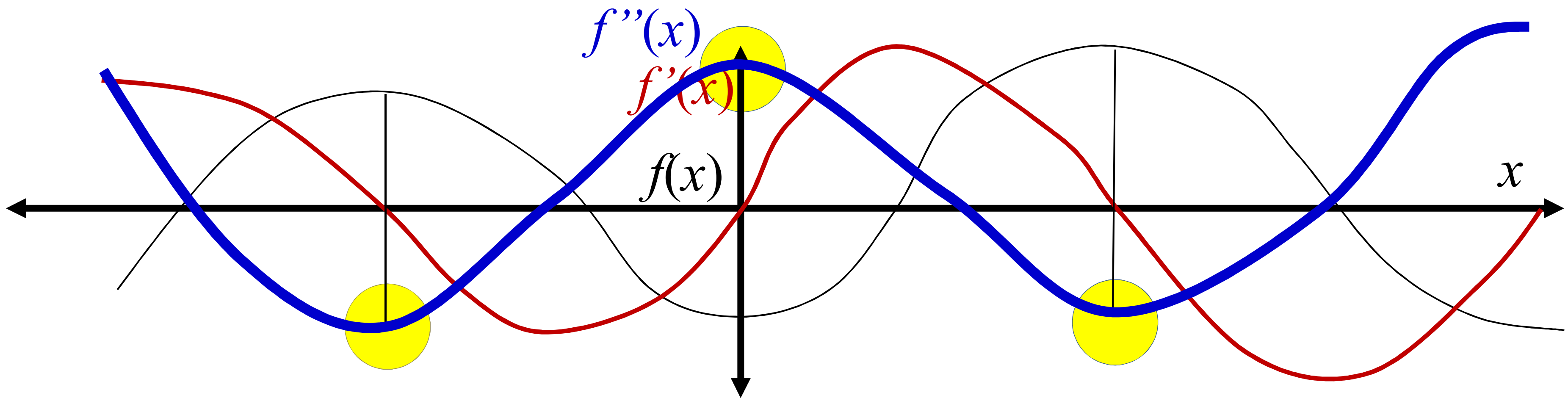
- Both *maxima* and *minima* have zero derivative
- Both are turning points

Derivatives of a curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

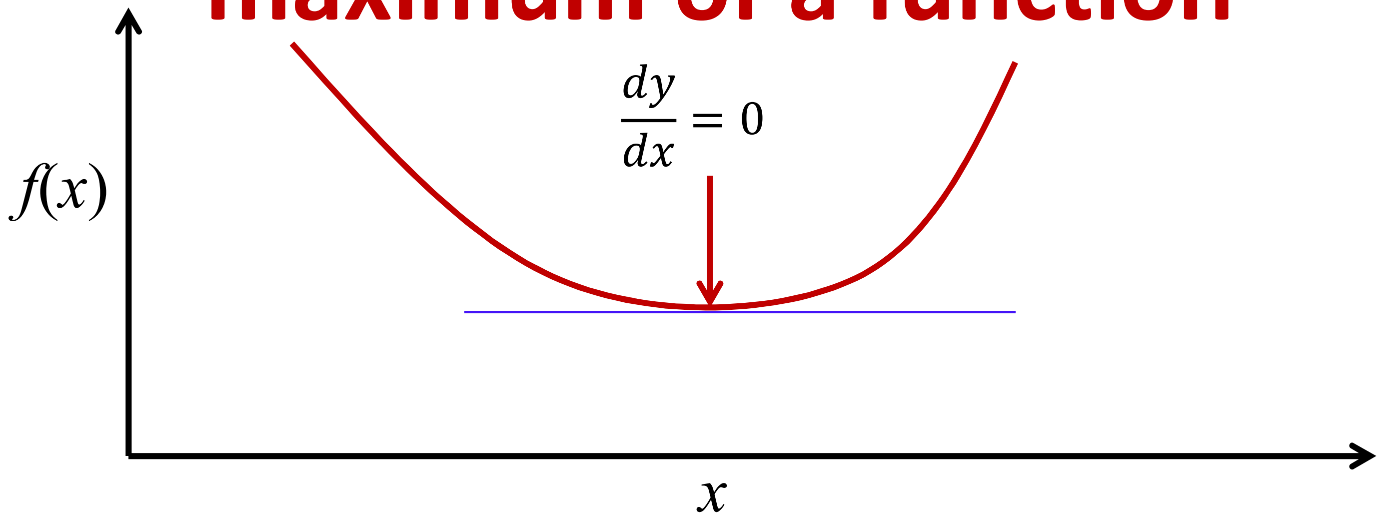
Derivative of the derivative of the curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

- The *second derivative f''*(*x*) is –ve at maxima and +ve at minima!

Soln: Finding the minimum or maximum of a function



- Find the value x at which $f'(x) = 0$: Solve

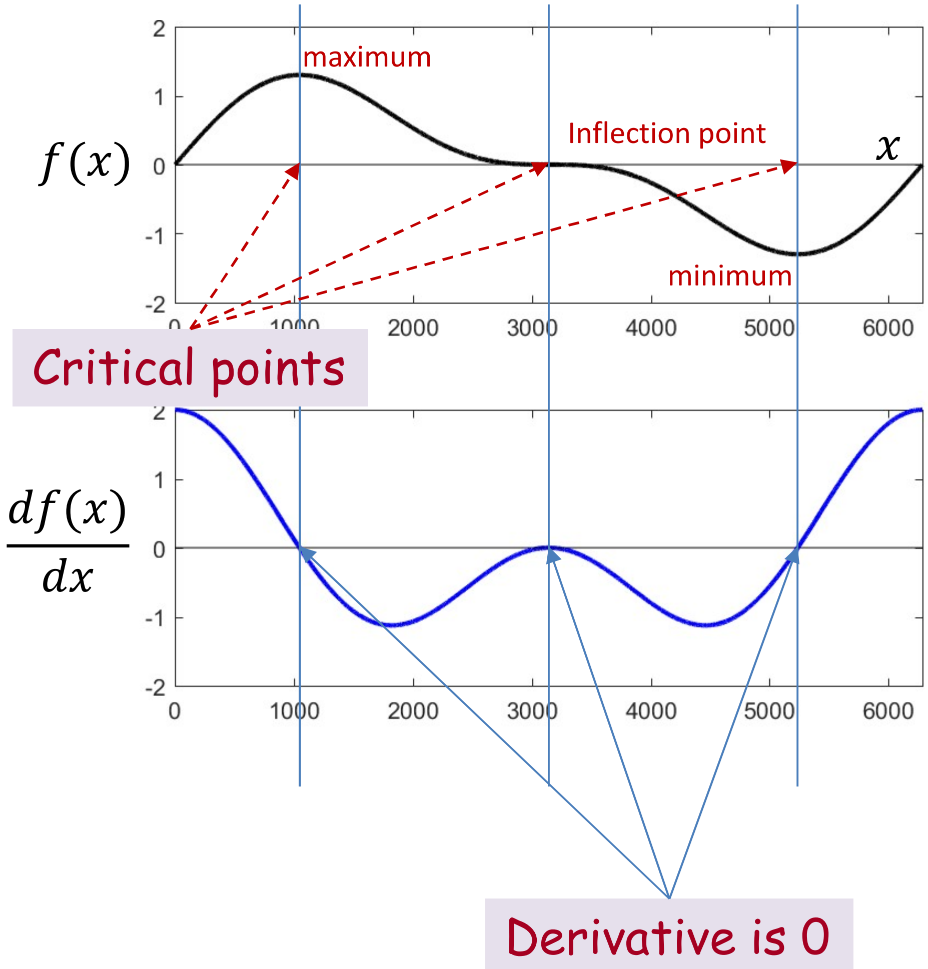
$$\frac{df(x)}{dx} = 0$$

- The solution x_{soln} is a turning point
- Check the double derivative at x_{soln} : compute

$$f''(x_{soln}) = \frac{df'(x_{soln})}{dx}$$

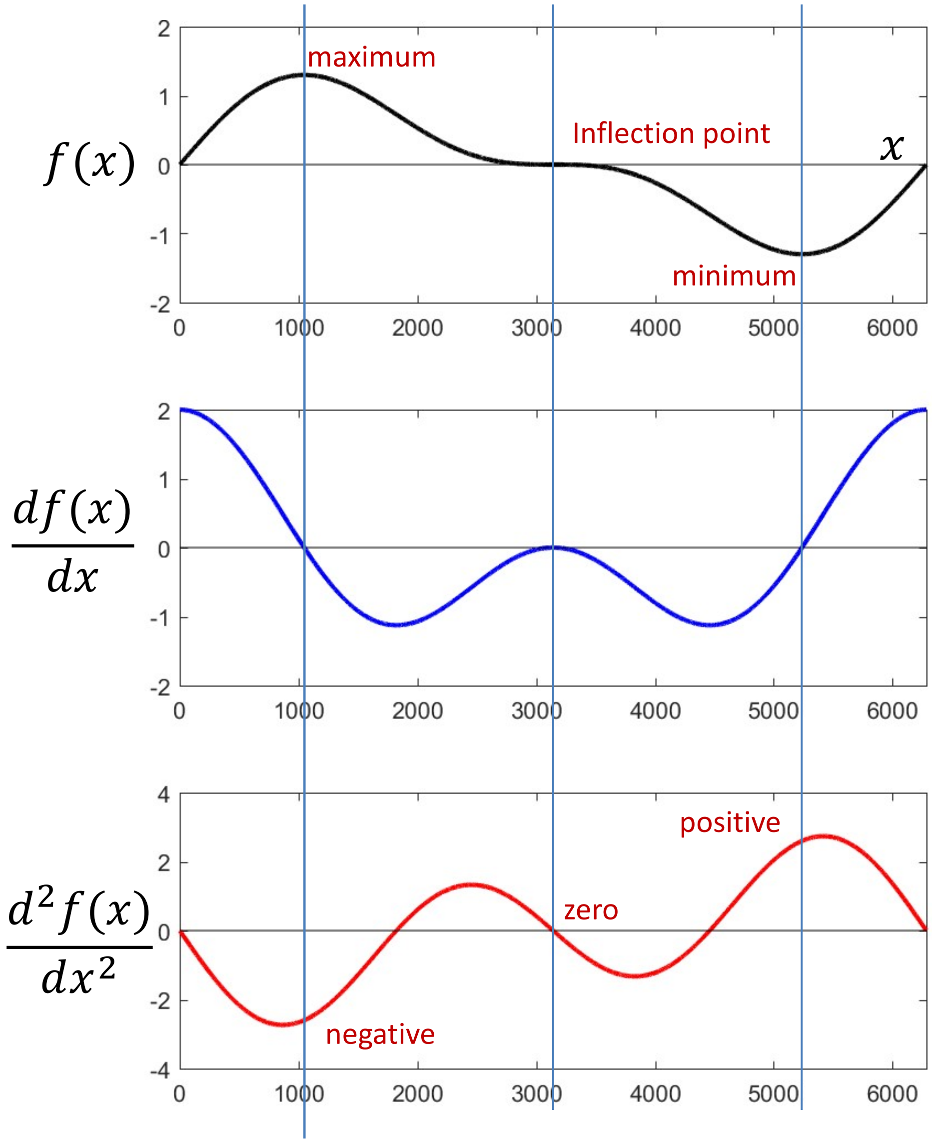
- If $f''(x_{soln})$ is positive x_{soln} is a minimum, otherwise it is a maximum

A note on derivatives of functions of single variable



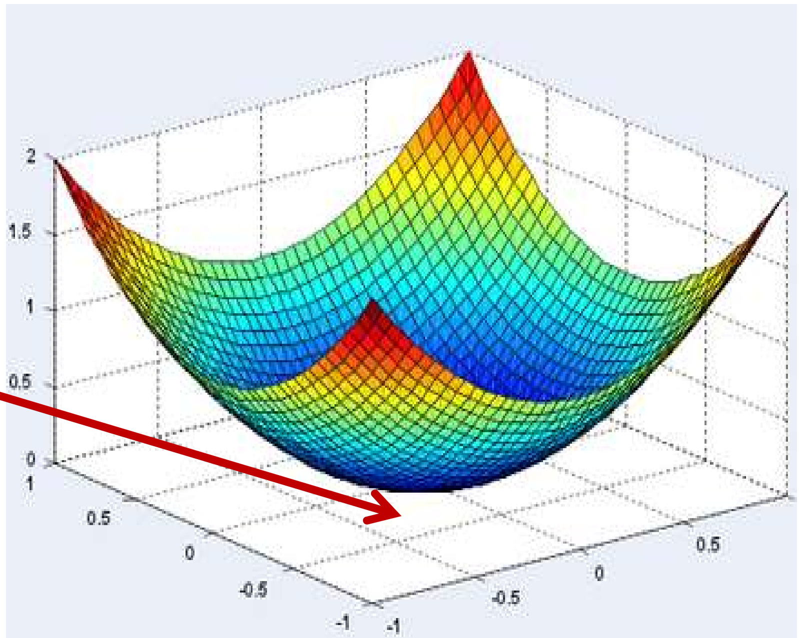
- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points

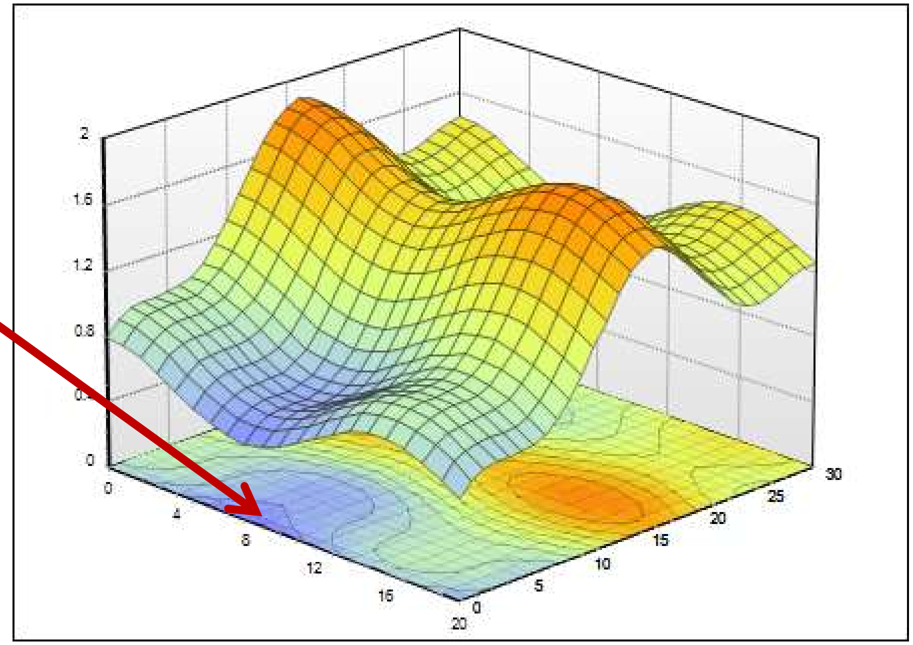
A note on derivatives of functions of single variable



- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - ≥ 0 at minima
  - ≤ 0 at maxima
  - Zero at inflection points
- It's a little more complicated for functions of multiple variables..

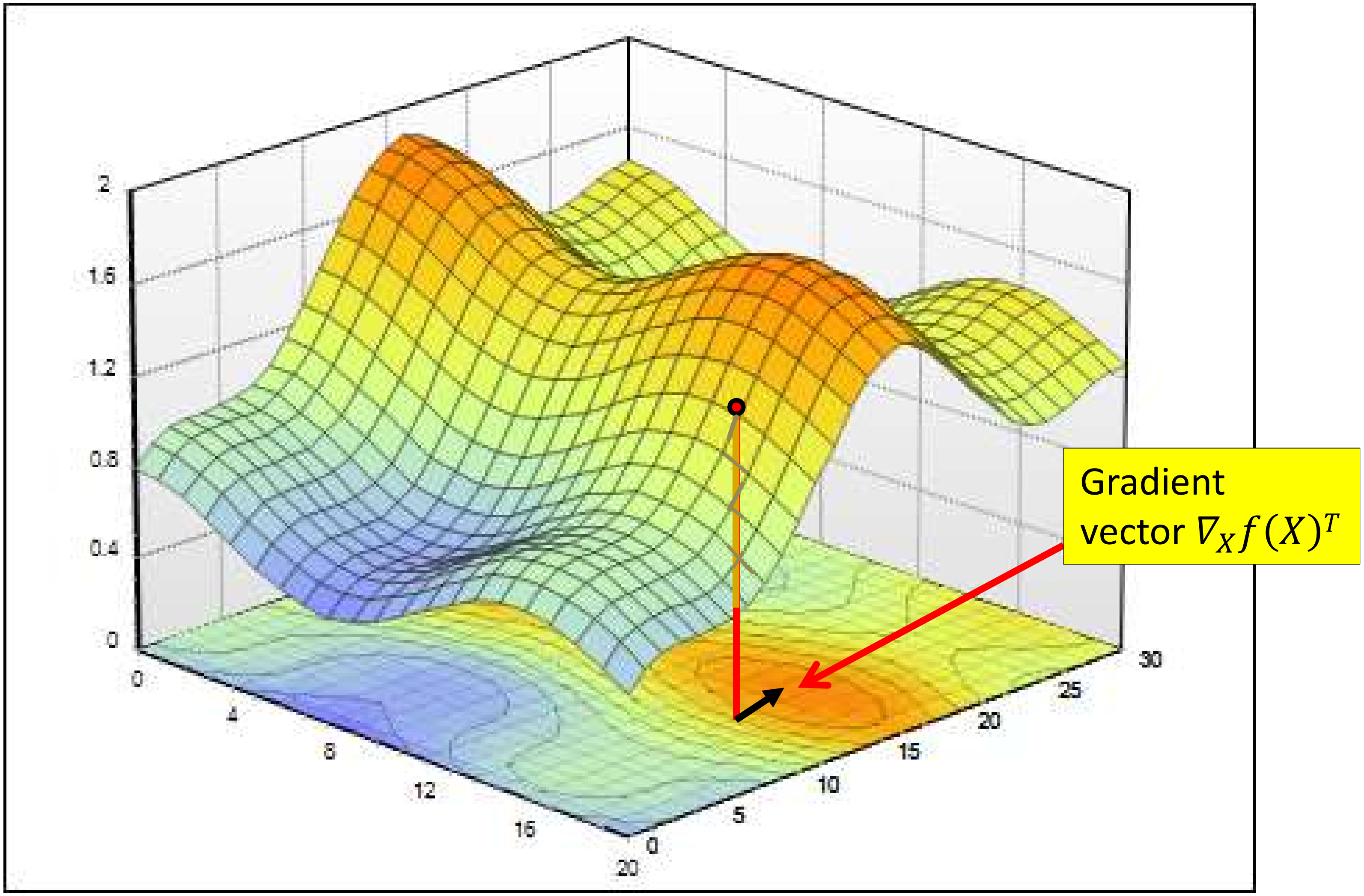
What about functions of multiple variables?





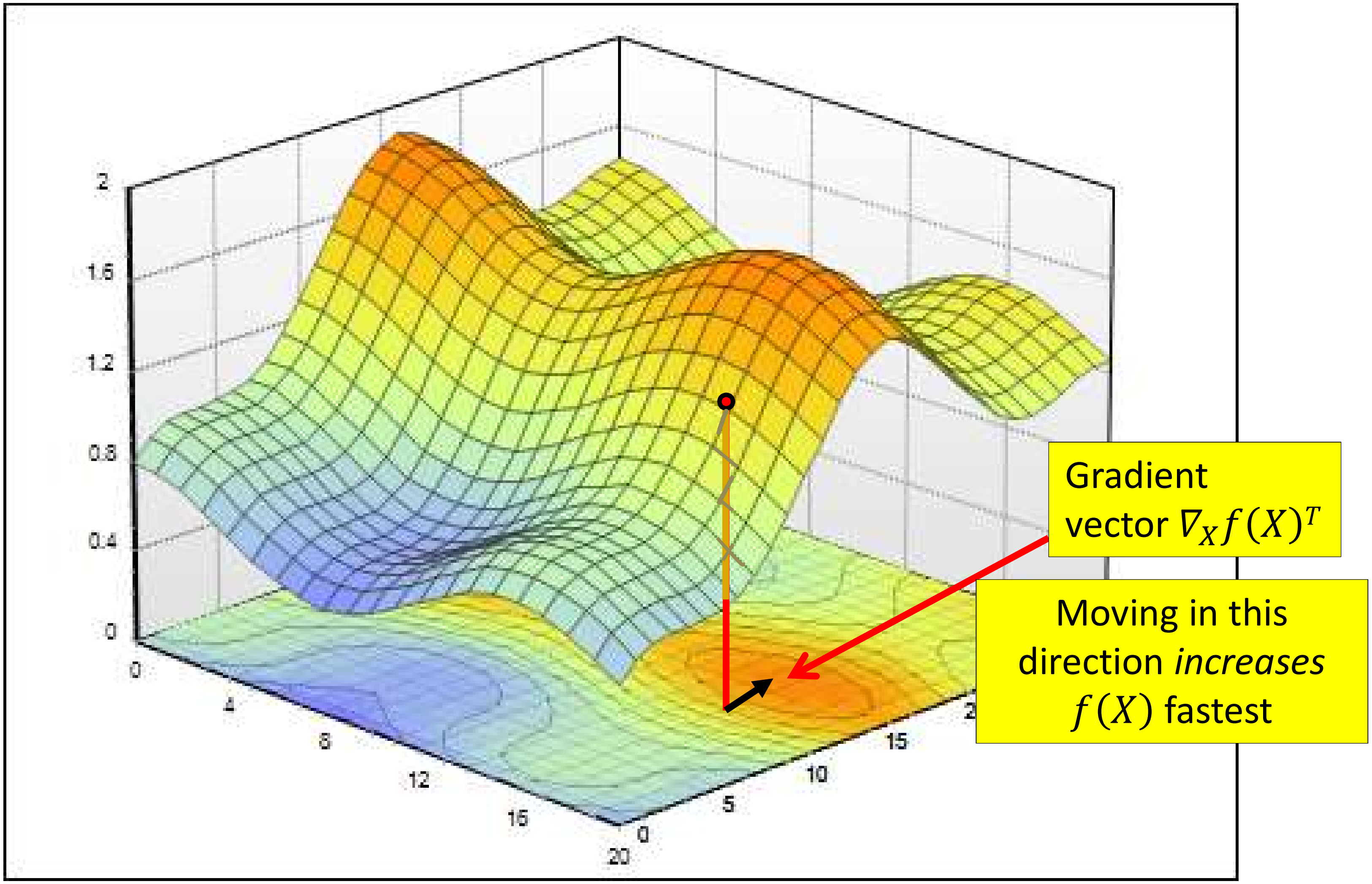
- The optimum point is still "turning" point
  - Shifting in any direction will increase the value
  - For smooth functions, miniscule shifts will not result in any change at all
- We must find a point where shifting in any direction by a microscopic amount will not change the value of the function

Gradient

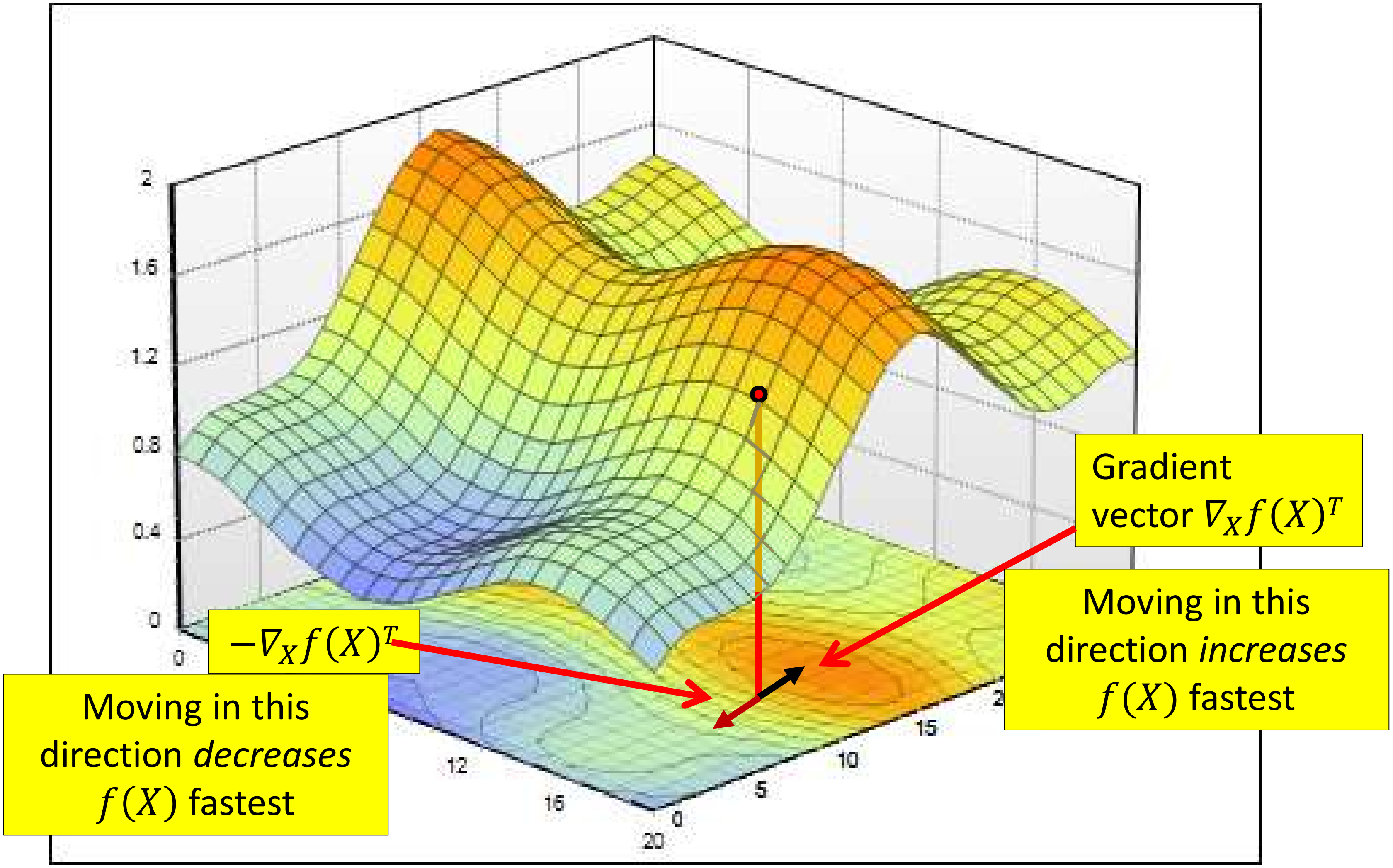


The gradient is the direction of fastest increase of the function

Gradient

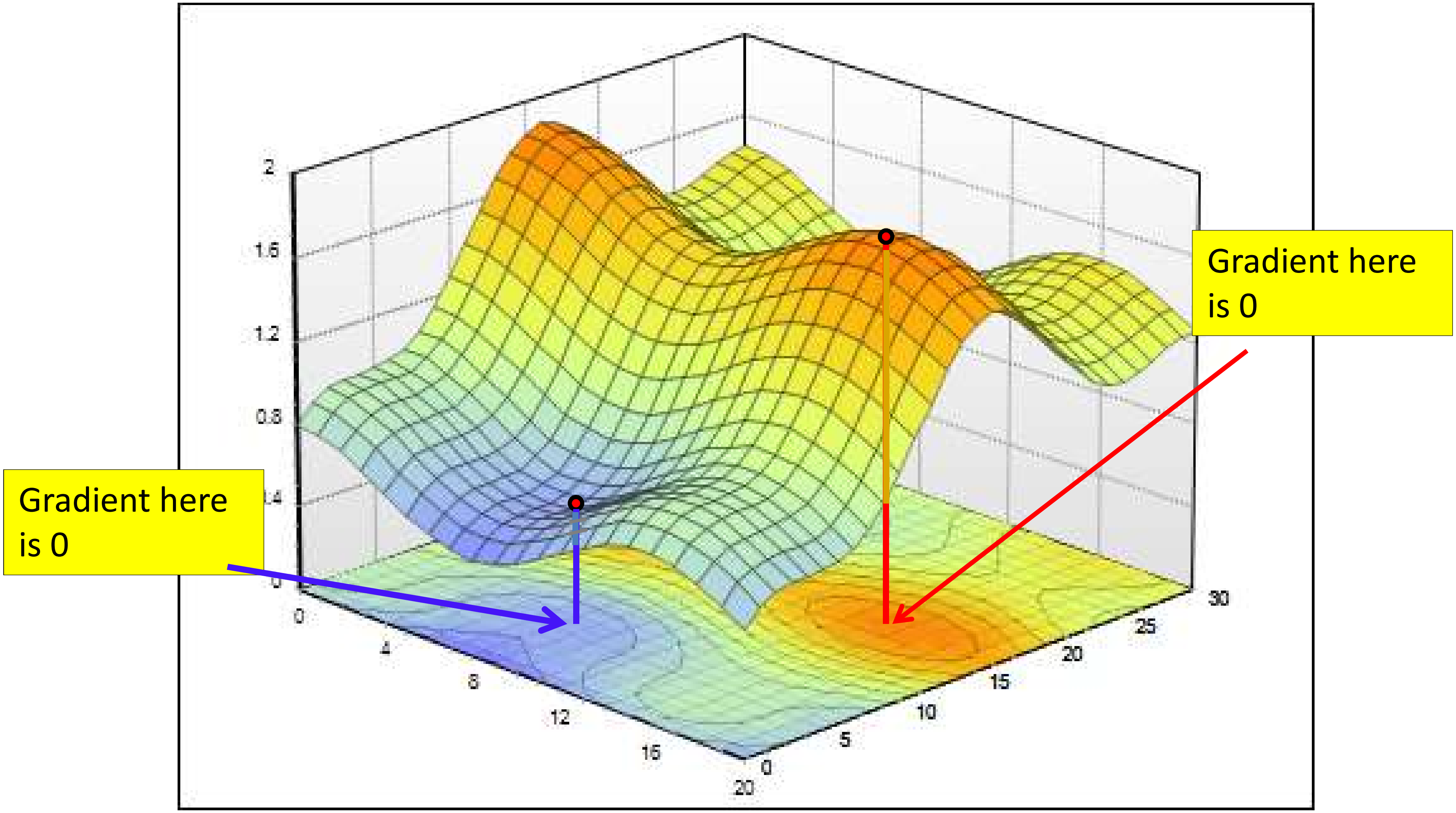

2
1.6
1.2
0.8
0.4
0
0
4
8
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest

Gradient



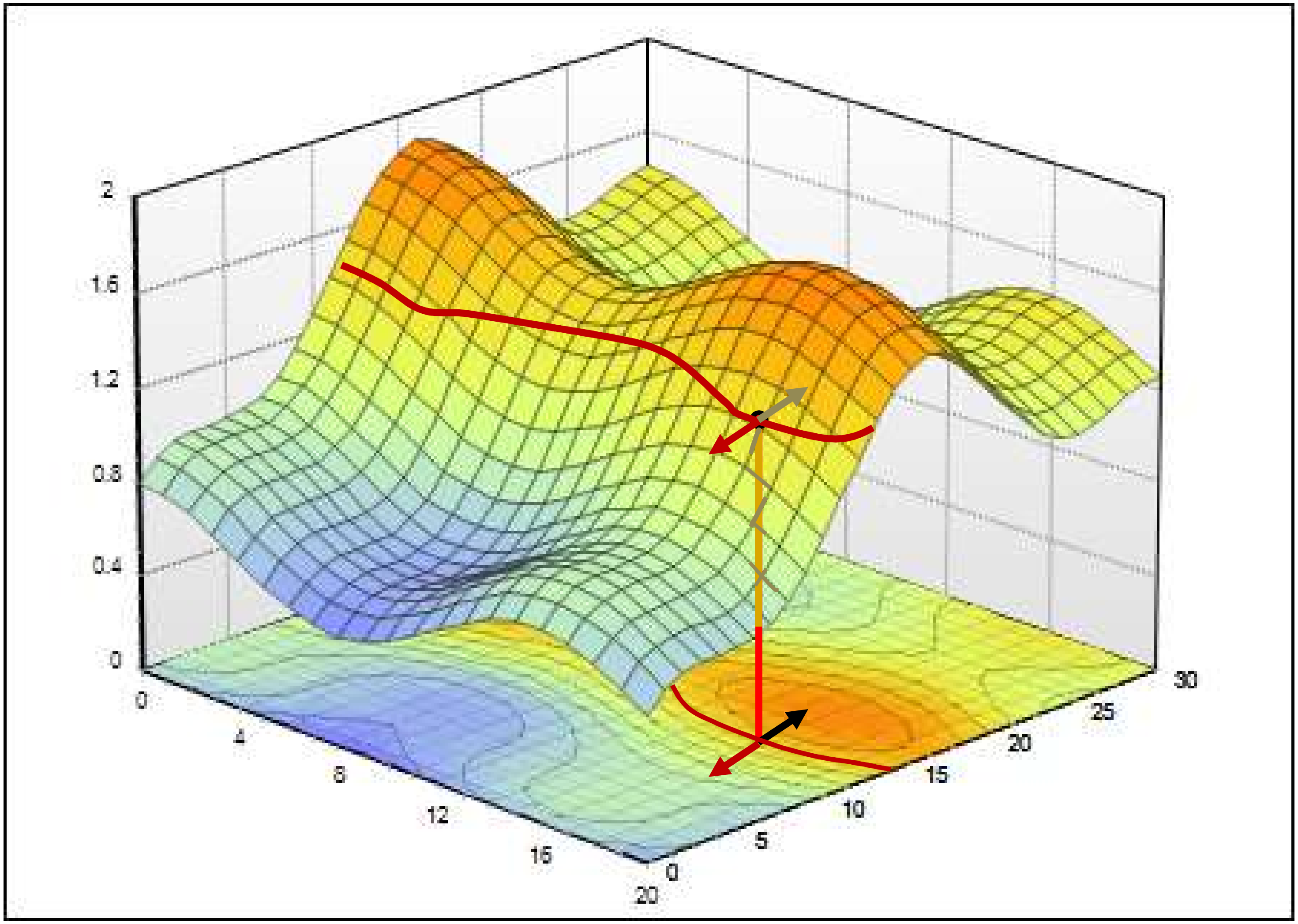
2
1.6
1.2
0.8
0.4
0
0
12
16
20
0
5
10
15
Gradient vector $\nabla_X f(X)^T$
Moving in this direction *increases* $f(X)$ fastest
$-\nabla_X f(X)^T$
Moving in this direction *decreases* $f(X)$ fastest

Gradient



Gradient here is 0
Gradient here is 0
2
1.6
1.2
0.8
0
0
4
8
12
16
20
0
5
10
15
20
25
30

Properties of Gradient: 2



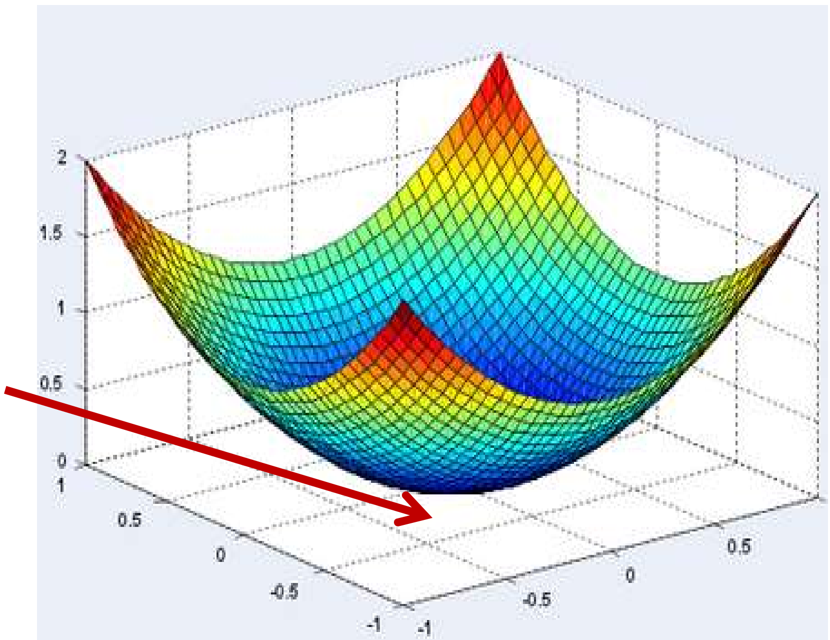
- The gradient vector $\nabla_X f(X)^T$ is perpendicular to the level curve

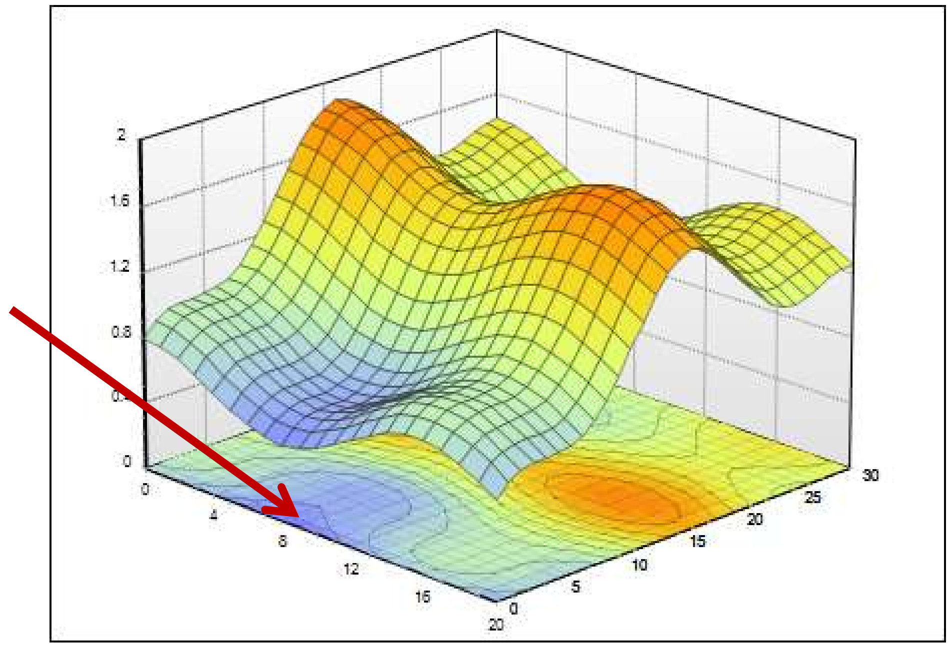
The Hessian

- The Hessian of a function $f(x_1, x_2, \ldots, x_n)$ is given by the second derivative

$$\nabla_X^2 f(x_1,\ldots,x_n) := \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1{}^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdot & \cdot & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2{}^2} & \cdot & \cdot & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdot & \cdot & \dfrac{\partial^2 f}{\partial x_n{}^2} \end{bmatrix}$$

Finding the minimum of a scalar function of a multi-variate input





- The optimum point is a turning point – the gradient will be 0

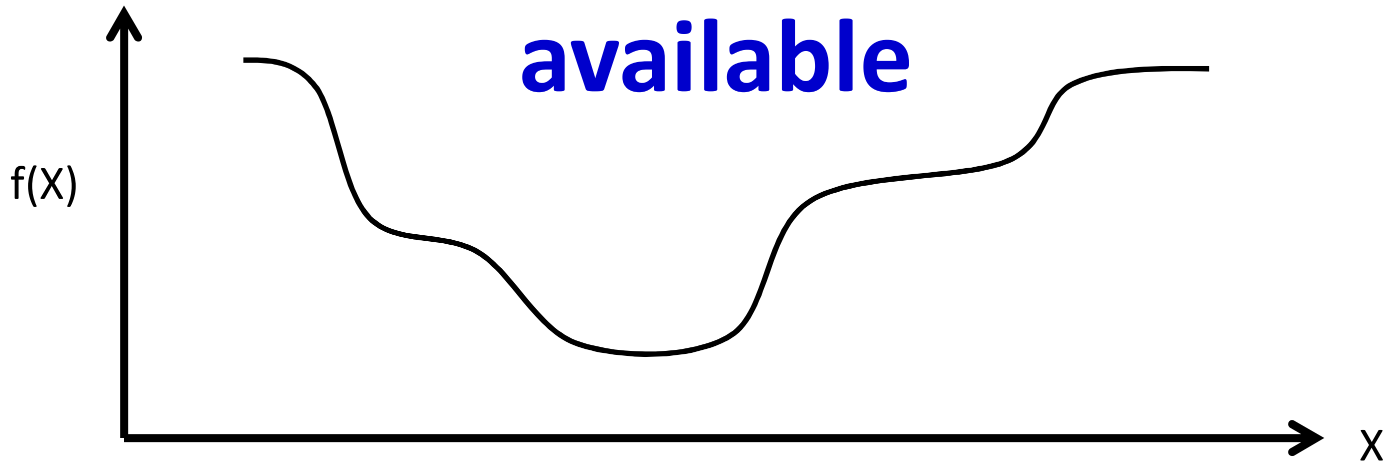
Unconstrained Minimization of function (Multivariate)

1. Solve for the X where the derivative (or gradient) equals to zero

$$\nabla_X f(X) = 0$$

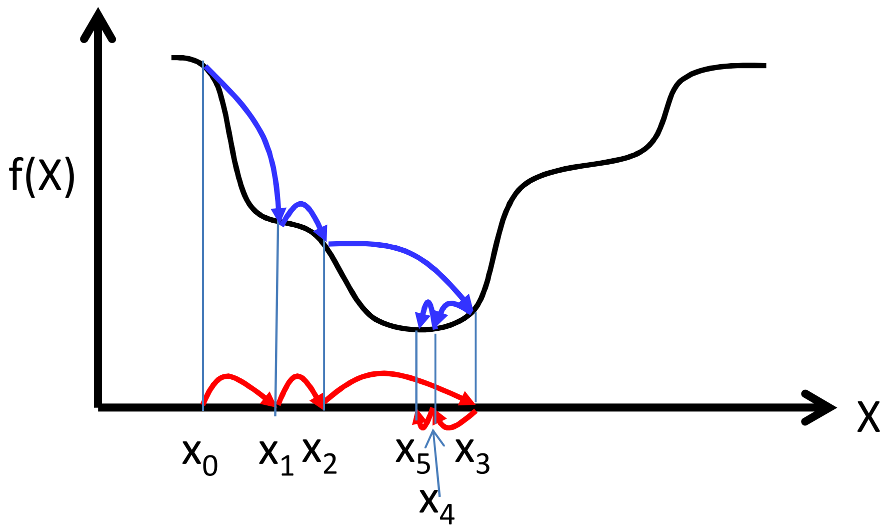
2. Compute the Hessian Matrix $\nabla_X^2 f(X)$ at the candidate solution and verify that
   - Hessian is positive definite (eigenvalues positive) -> to identify local minima
   - Hessian is negative definite (eigenvalues negative) -> to identify local maxima

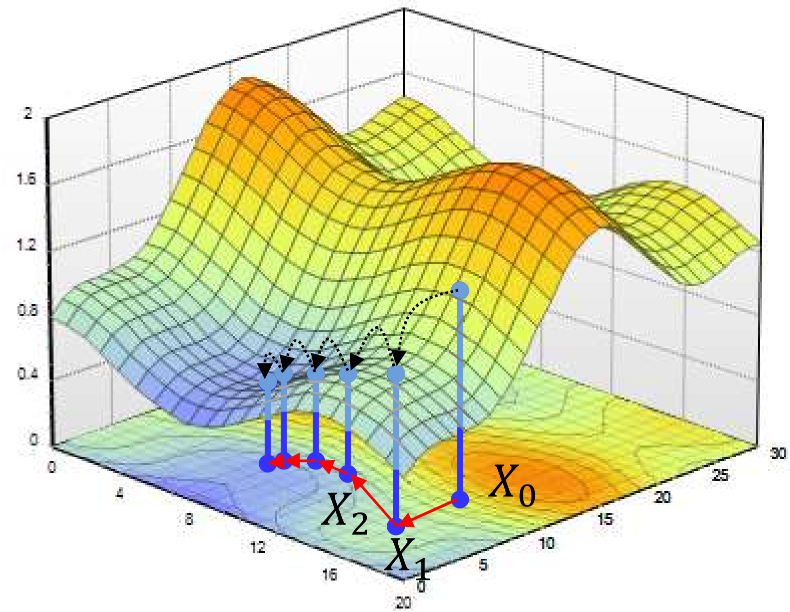
Closed Form Solutions are not always available



- Often it is not possible to simply solve $\nabla_X f(X) = 0$
  - The function to minimize/maximize may have an intractable form
- In these situations, iterative solutions are used
  - Begin with a “guess” for the optimal X and refine it iteratively until the correct value is obtained

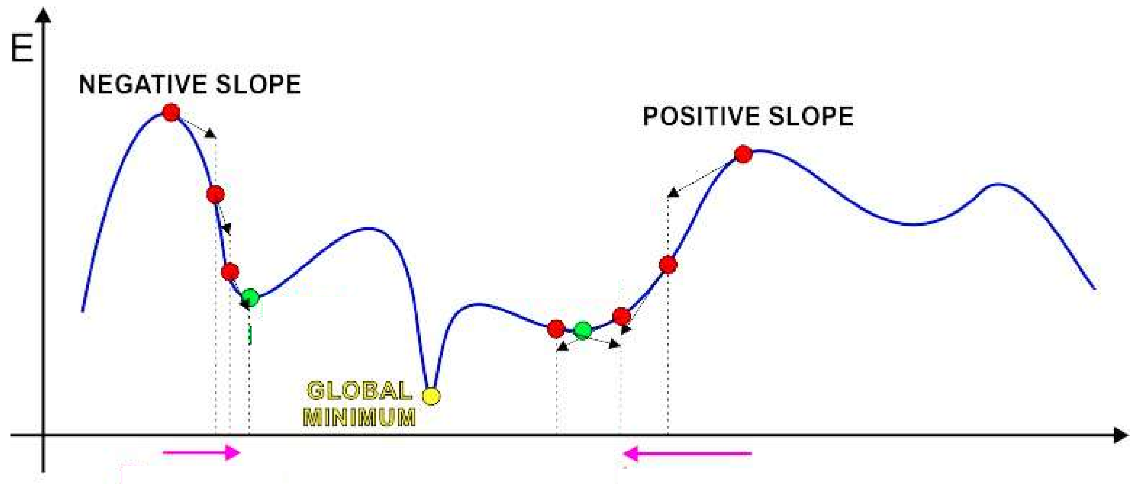
Iterative solutions





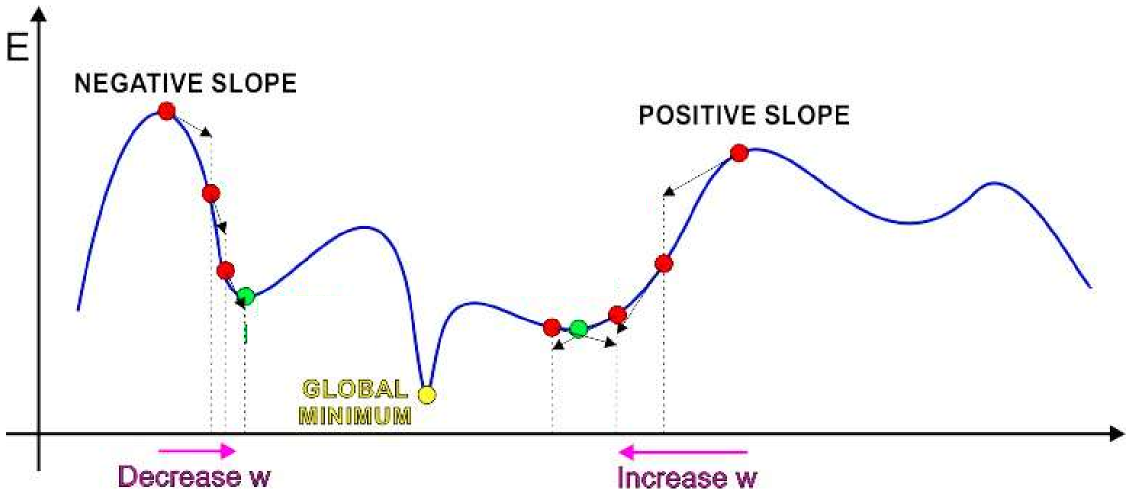
- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $\mathrm{f}(X)$
  - Stop when $\mathrm{f}(X)$ no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

The Approach of Gradient Descent



- Iterative solution:
  - Start at some point
  - Find direction in which to shift this point to decrease error
    - This can be found from the derivative of the function
      - A positive derivative → moving left decreases error
      - A negative derivative → moving right decreases error
  - Shift point in this direction

The Approach of Gradient Descent



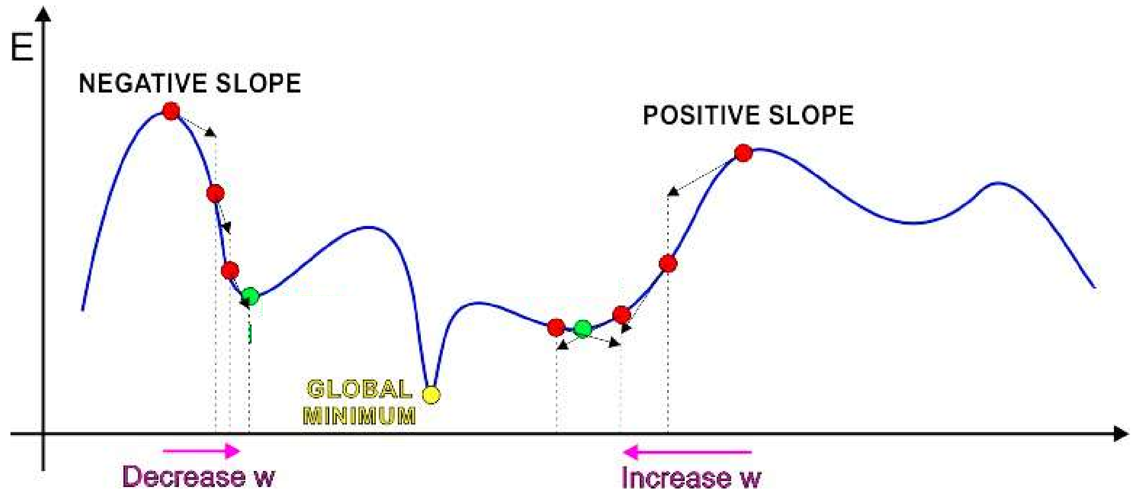
- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'\left(x^k\right) \neq 0$
    - If $sign\left(f'\left(x^k\right)\right)$ is positive:

$$x^{k+1} = x^k - step$$

    - Else

$$x^{k+1} = x^k + step$$

The Approach of Gradient Descent

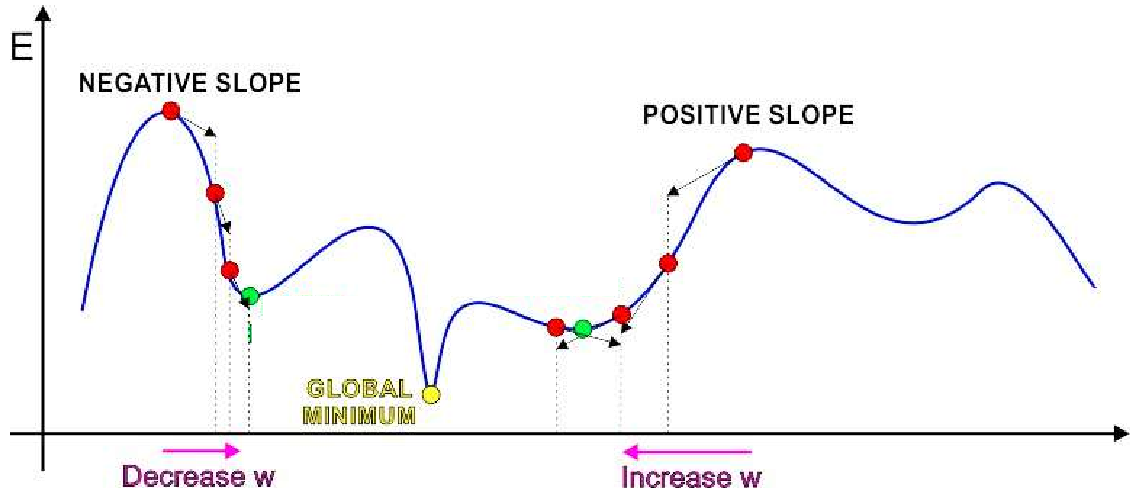


- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'\left(x^k\right) \neq 0$

$$x^{k+1} = x^k - sign\left(f'\left(x^k\right)\right).step$$

- Identical to previous algorithm

The Approach of Gradient Descent



- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'\left(x^k\right) \neq 0$

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

- η^k is the “step size”

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

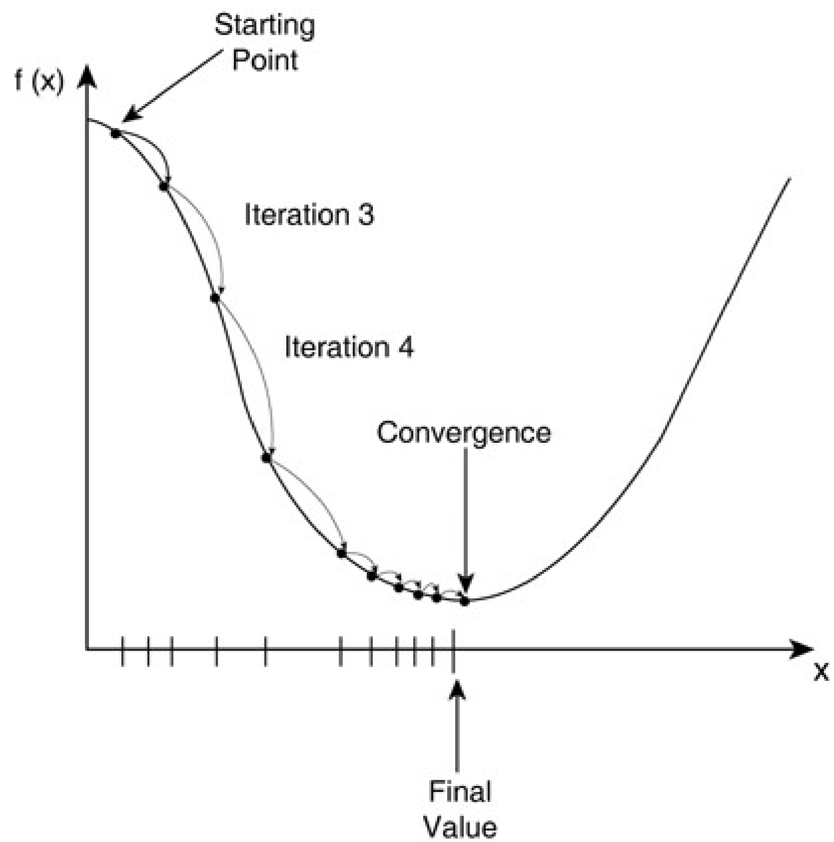
Gradient descent convergence criteria

- The gradient descent algorithm converges when one of the following criteria is satisfied

$$\left| f(x^{k+1}) - f(x^k) \right| < \varepsilon_1$$

- Or

$$\left\| \nabla_x f(x^k) \right\| < \varepsilon_2$$

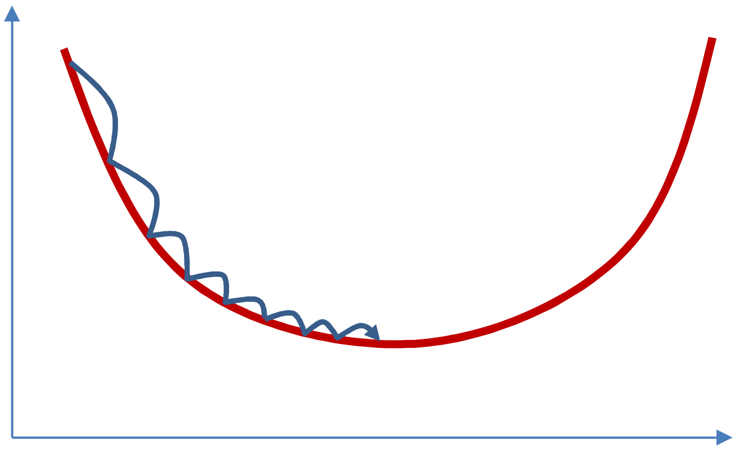


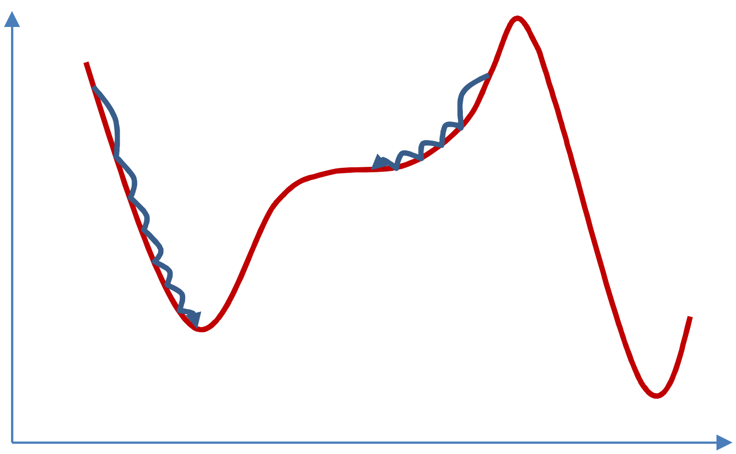
Overall Gradient Descent Algorithm

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

- do
  - $x^{k+1} = x^k - \eta^k \nabla_x f\left(x^k\right)^T$
  - $k = k + 1$
- while $\left|f\left(x^{k+1}\right) - f\left(x^k\right)\right| > \varepsilon$

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

- Returning to our problem..

Problem Statement

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- Minimize the following function

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

w.r.t W

- This is problem of function minimization
  - An instance of optimization

Preliminaries

- Before we proceed: the problem setup

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

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

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

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

What is f() and what are its parameters W?

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

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

What is the divergence div()?

What is f() and what are its parameters W?

Problem Setup: Things to define

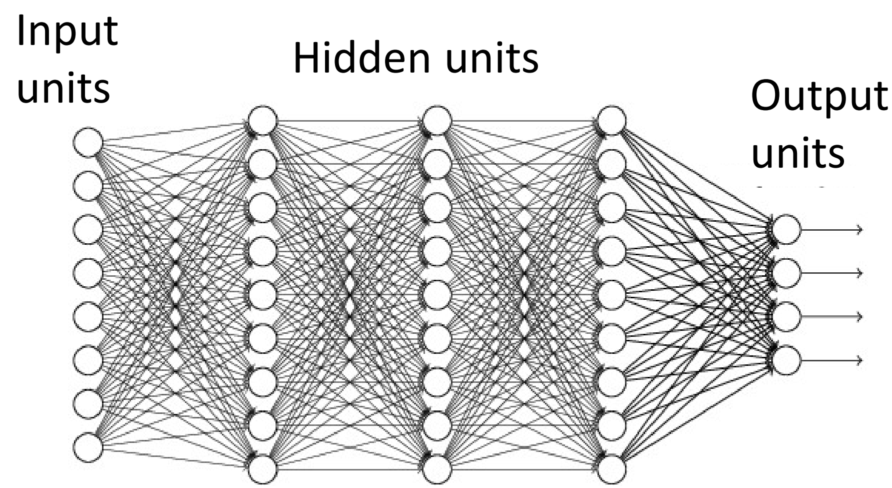
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

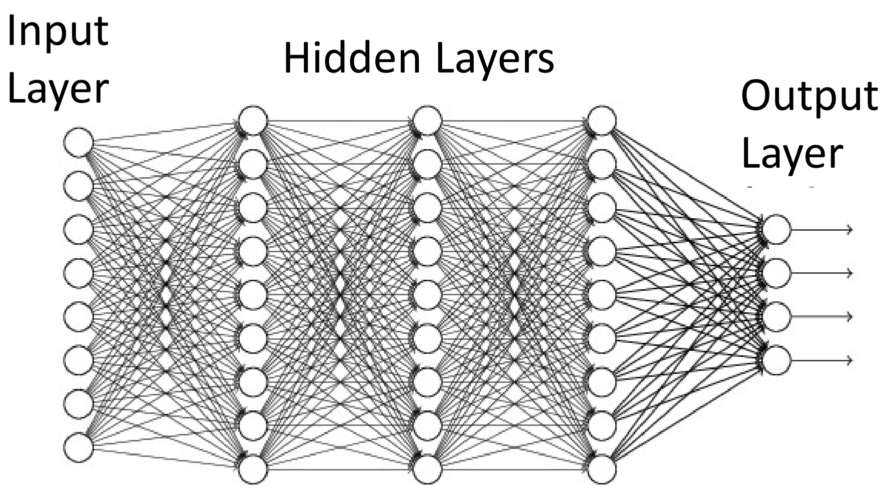
What is f() and what are its parameters W?

What is f()? Typical network



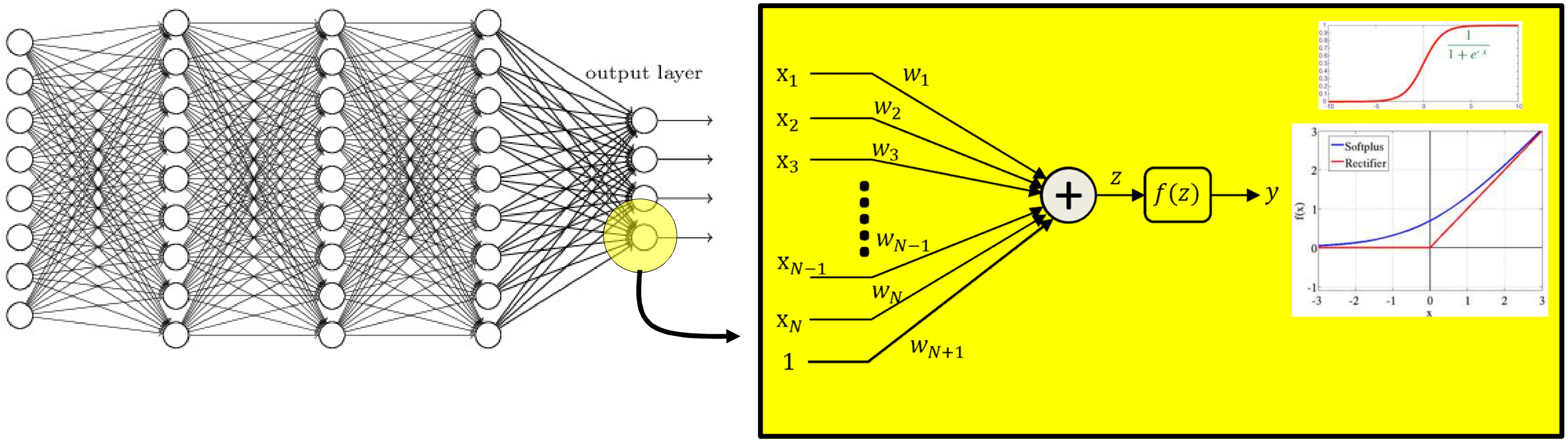
- Multi-layer perceptron
- A *directed* network with a set of inputs and outputs
  - No loops

Typical network



- We assume a "layered" network for simplicity
  - Each "layer" of neurons only gets inputs from the earlier layer(s) and outputs signals only to later layer(s)
  - We will refer to the inputs as the ***input layer***
    - No neurons here – the "layer" simply refers to inputs
  - We refer to the outputs as the ***output layer***
  - Intermediate layers are ***"hidden" layers***

The individual neurons



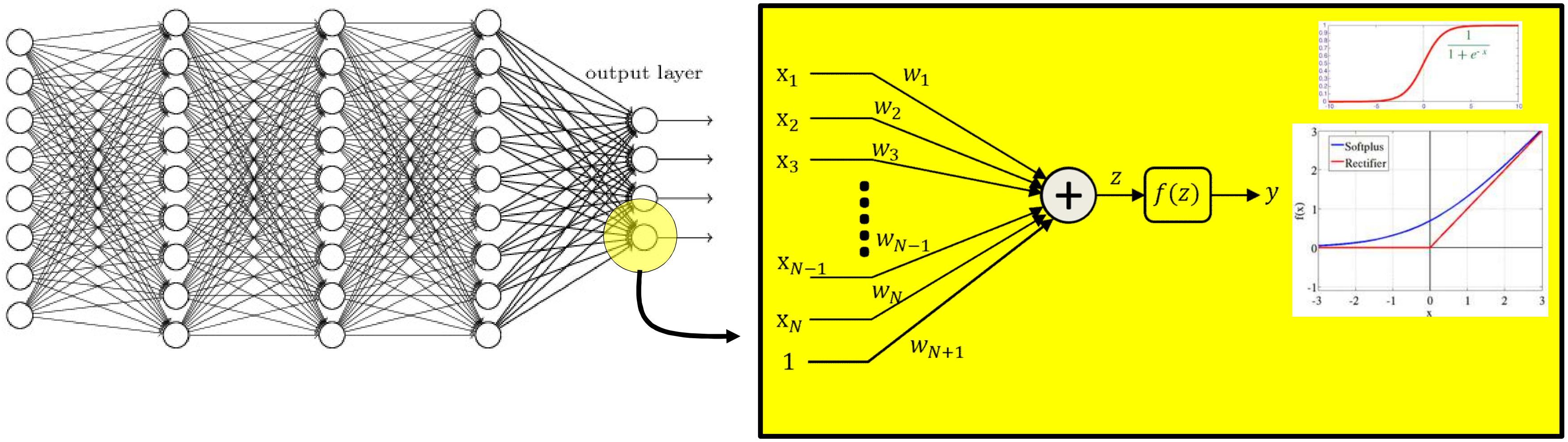
- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A differentiable activation function applied to an affine combination of the inputs

$$y = f\left(\sum_i w_i x_i + b\right)$$

  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

The individual neurons



- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A differentiable activation function applied to an affine combination of the input

$$y = f\left(\sum_i w_i x_i + b\right)$$

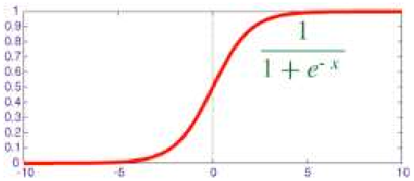
  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

We will assume this unless otherwise specified

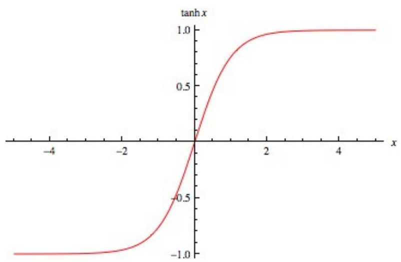
Parameters are weights w_i and bias b

Activations and their derivatives



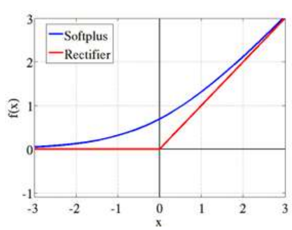
$$f(z) = \frac{1}{1+\exp(-z)}$$

$$f'(z) = f(z)(1 - f(z))$$



$$f(z) = \tanh(z)$$

$$f'(z) = (1 - f^2(z))$$



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

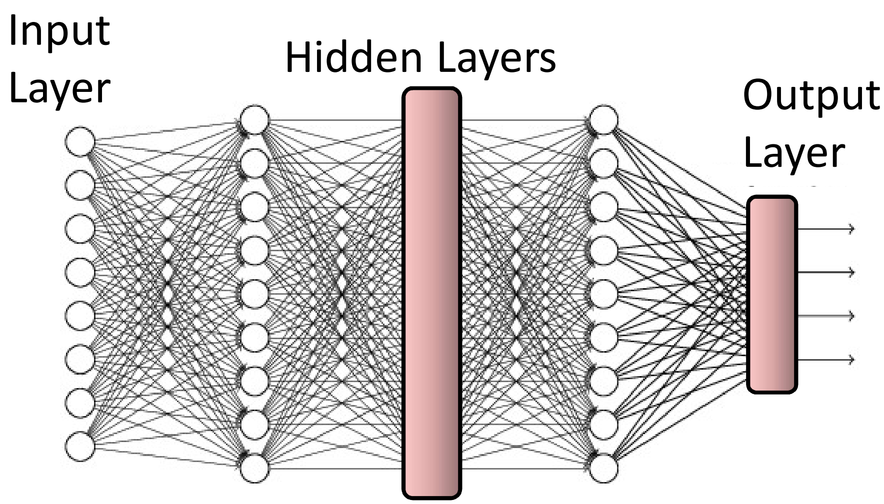
[*] $$f'(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

$$f(z) = \log(1 + \exp(z))$$

$$f'(z) = \frac{1}{1+\exp(-z)}$$

- Some popular activation functions and their derivatives

Vector Activations



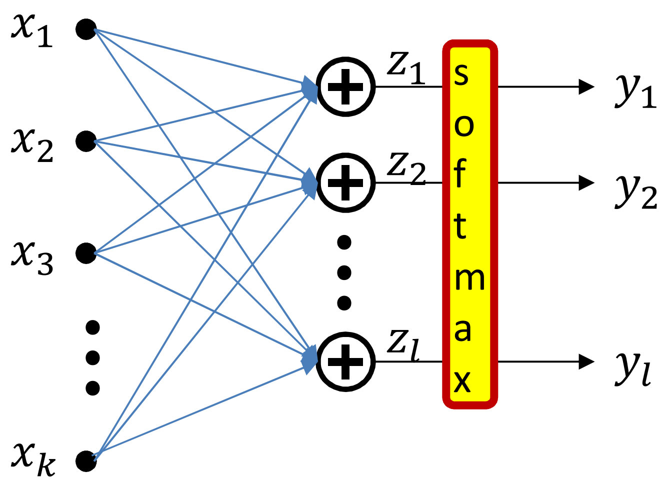
- We can also have neurons that have *multiple coupled* outputs

$$[y_1, y_2, \ldots, y_l] = f(x_1, x_2, \ldots, x_k; W)$$

  – Function $f()$ operates on set of inputs to produce set of outputs

  – Modifying a single parameter in W will affect *all* outputs

Vector activation example: Softmax



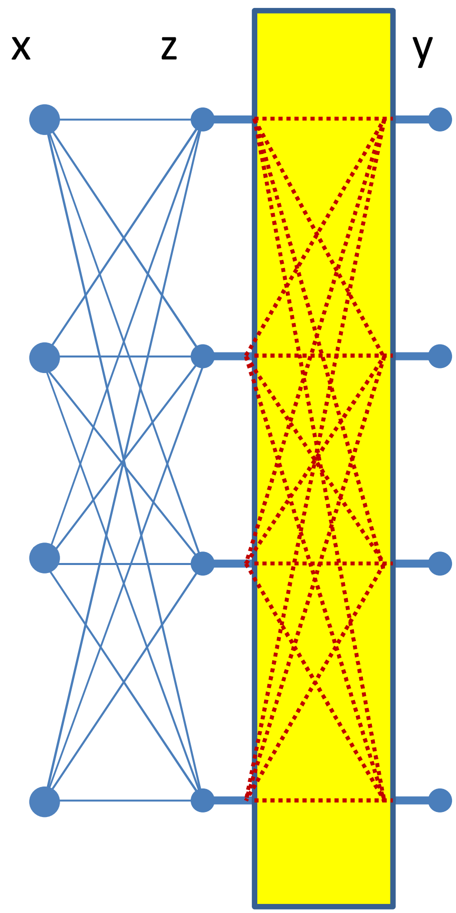
- Example: Softmax *vector* activation

$$z_i = \sum_j w_{ji} x_j + b_i$$

$$y = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Parameters are weights w_{ji} and bias b_i

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



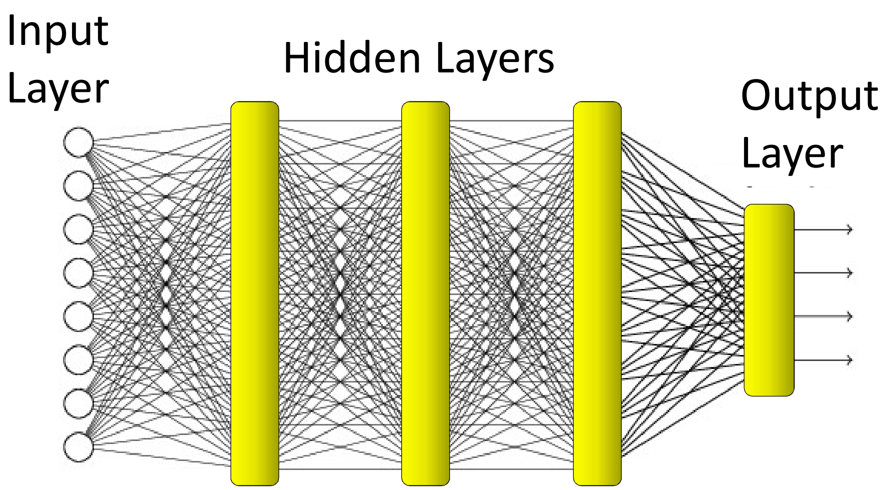
$$z_i = \sum_j w_{ji} x_j + b_i$$

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

Parameters are weights w_{ji} and bias b_i

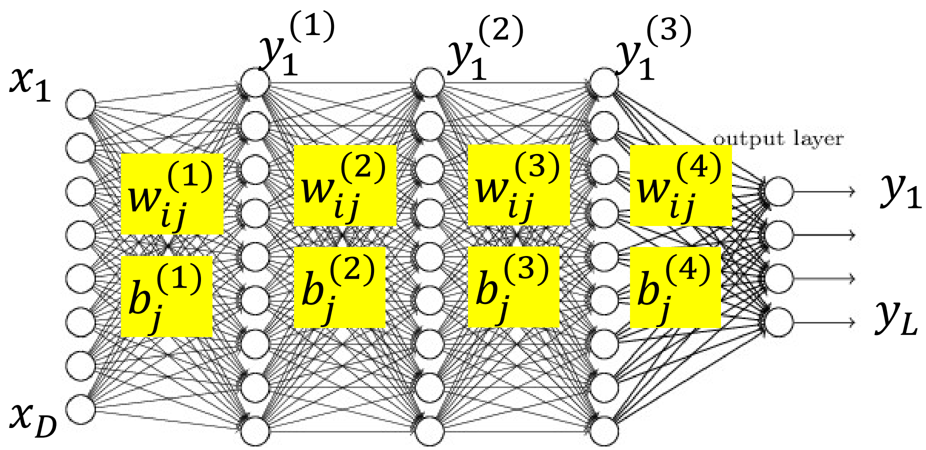
- A layer of multiplicative combination is a special case of vector activation

Typical network



- In a layered network, each layer of perceptrons can be viewed as a single vector activation

Notation



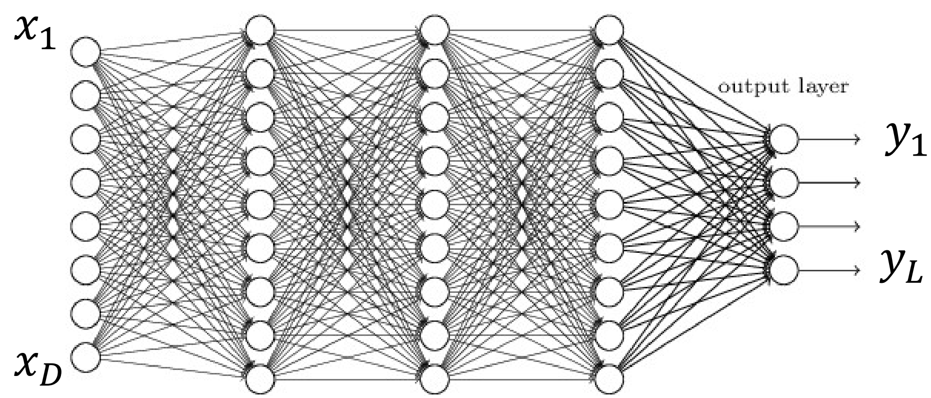
- The input layer is the 0th layer
- We will represent the output of the i-th perceptron of the kth layer as $y_i^{(k)}$
  - **Input to network:** $y_i^{(0)} = x_i$
  - **Output of network:** $y_i = y_i^{(N)}$
- We will represent the weight of the connection between the i-th unit of the k-1th layer and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

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

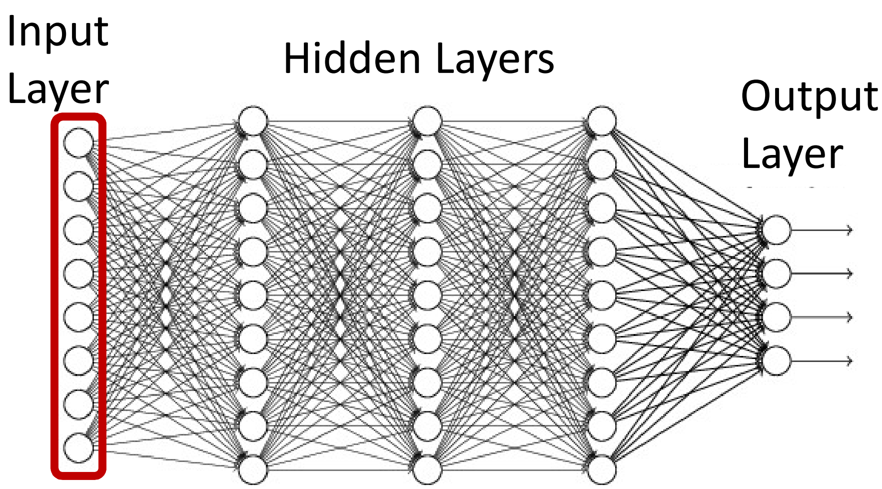
$$Loss(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

Vector notation



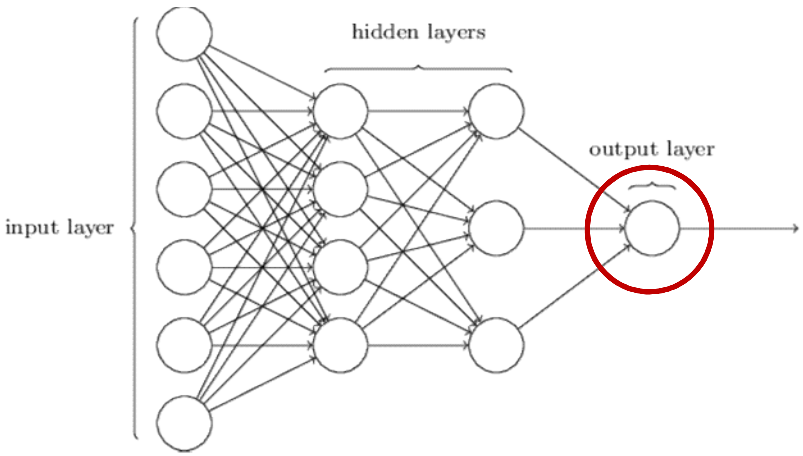
- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- $X_n = [x_{n1}, x_{n2}, \dots, x_{nD}]$ is the nth input vector
- $d_n = [d_{n1}, d_{n2}, \dots, d_{nL}]$ is the nth desired output
- $Y_n = [y_{n1}, y_{n2}, \dots, y_{nL}]$ is the nth vector of *actual* outputs of the network
- We will sometimes drop the first subscript when referring to a *specific* instance

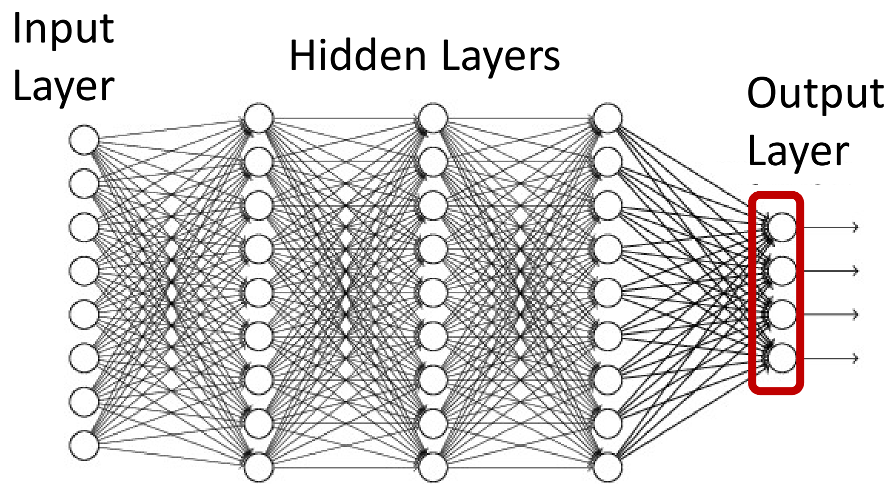
Representing the input



- Vectors of numbers
  - (or may even be just a scalar, if input layer is of size 1)
  - E.g. vector of pixel values
  - E.g. vector of speech features
  - E.g. real-valued vector representing text
    - We will see how this happens later in the course
  - Other real valued vectors

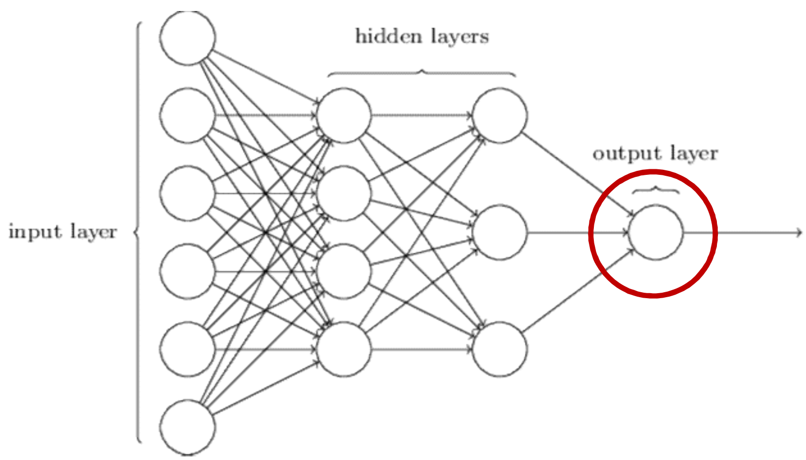
Representing the output





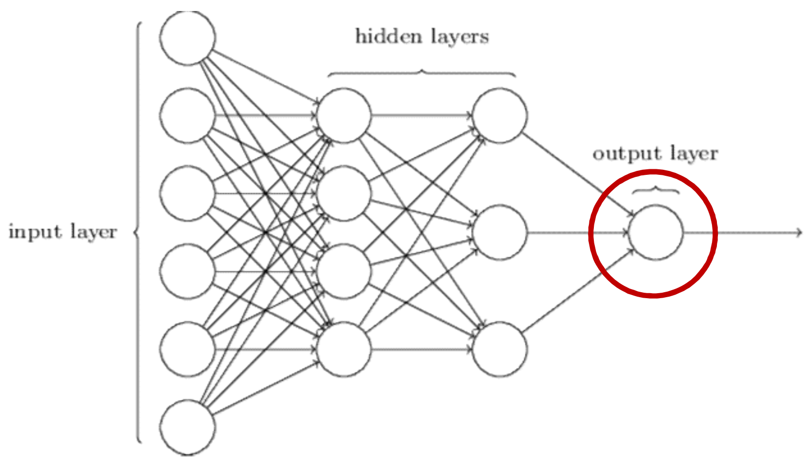
- If the desired *output* is real-valued, no special tricks are necessary
  - Scalar Output : single output neuron
    - d = scalar (real value)
  - Vector Output : as many output neurons as the dimension of the desired output
    - $d = [d_1\ d_2\ ..\ d_L]$ (vector of real values)

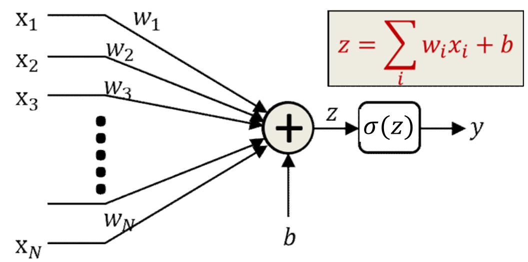
Representing the output

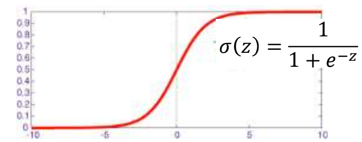


- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
  – 1 = Yes it's a cat
  – 0 = No it's not a cat.

Representing the output

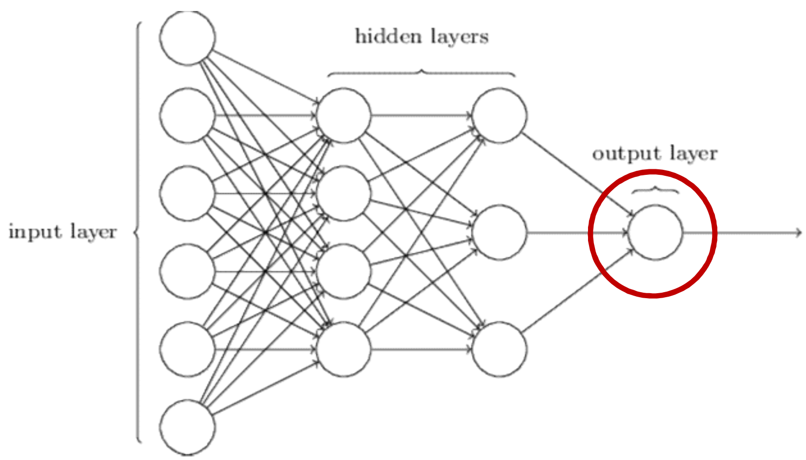


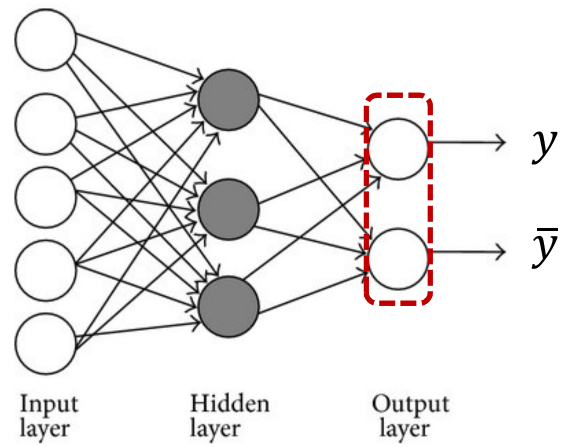




- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically a sigmoid
  - Viewed as the *probability* $P(Y = 1|X)$ of class value 1
    - Indicating the fact that for actual data, in general a feature value X may occur for both classes, but with different probabilities
    - Is differentiable

Representing the output





- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
  - 1 = Yes it's a cat
  - 0 = No it's not a cat.

- Sometimes represented by *two* outputs, one representing the desired output, the other representing the *negation* of the desired output
  - Yes: → [1 0]
  - No: → [0 1]
- The output explicitly becomes a 2-output softmax

Multi-class output: One-hot representations

- Consider a network that must distinguish if an input is a cat, a dog, a camel, a hat, or a flower
- We can represent this set as the following vector:

  $[\text{cat dog camel hat flower}]^T$

- For inputs of each of the five classes the desired output is:

  cat: $[1\ 0\ 0\ 0\ 0]^T$

  dog: $[0\ 1\ 0\ 0\ 0]^T$

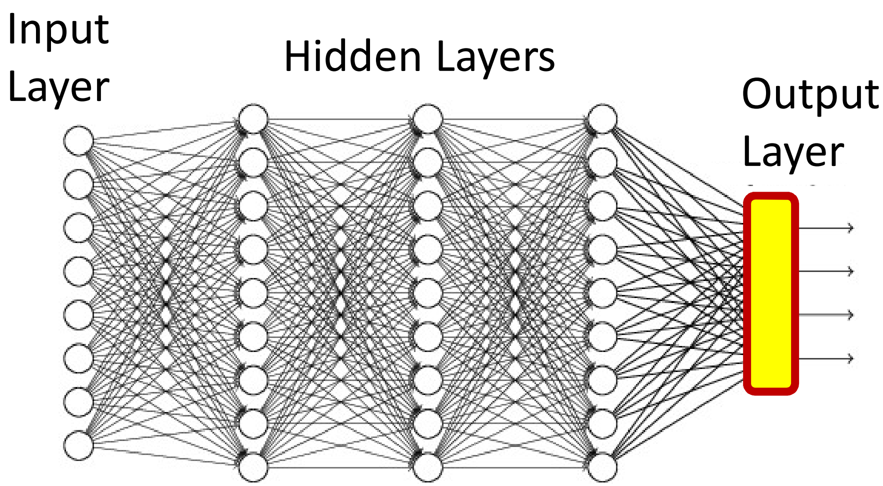
  camel: $[0\ 0\ 1\ 0\ 0]^T$

  hat: $[0\ 0\ 0\ 1\ 0]^T$

  flower: $[0\ 0\ 0\ 0\ 1]^T$

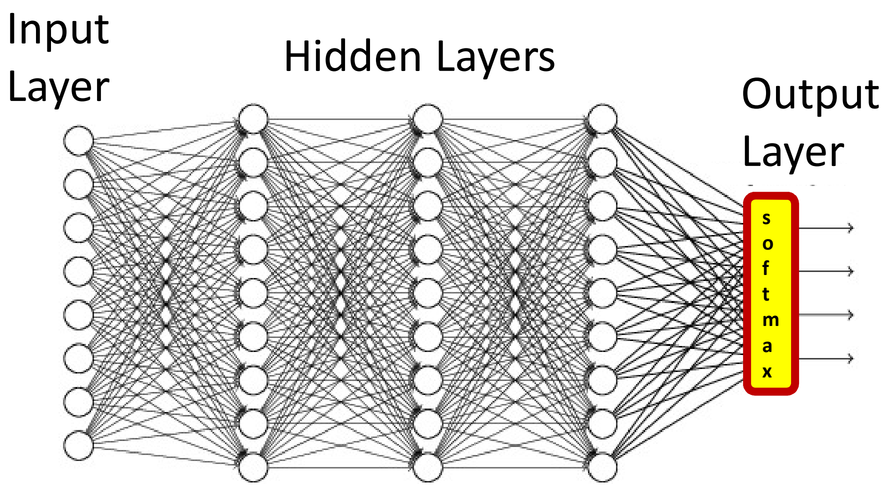
- For an input of any class, we will have a five-dimensional vector output with four zeros and a single 1 at the position of that class
- This is a *one hot vector*

Multi-class networks



- For a multi-class classifier with N classes, the one-hot representation will have N binary target outputs (d)
  – An N-dimensional binary vector
- The neural network's output too must ideally be binary (N-1 zeros and a single 1 in the right place)
- More realistically, it will be a probability vector
  – N probability values that sum to 1.

Multi-class classification: Output



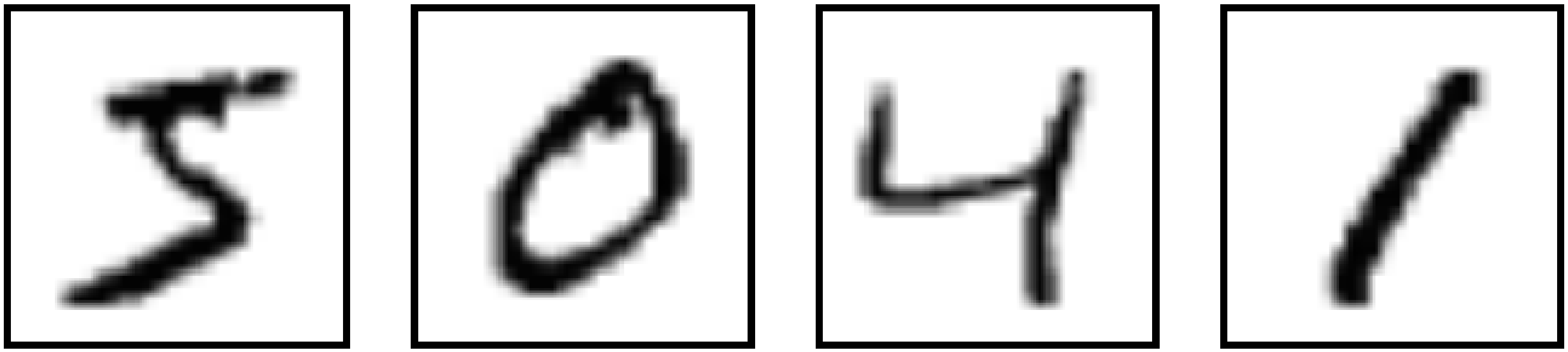
- Softmax *vector* activation is often used at the output of multi-class classifier nets

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

$$y_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

- This can be viewed as the probability $y_i = P(class = i|X)$

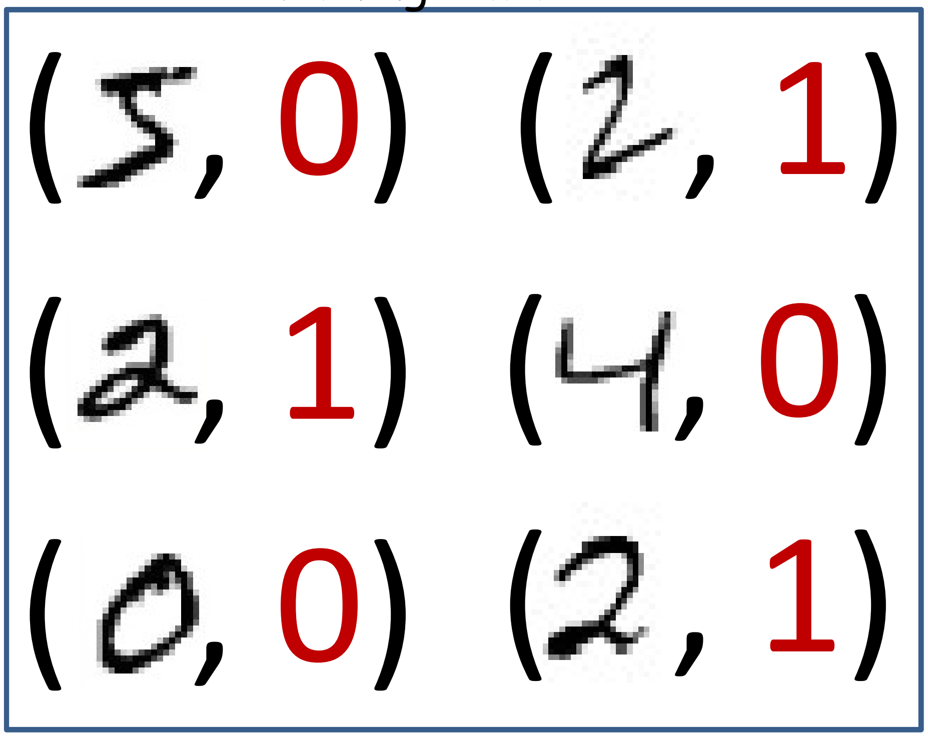
Typical Problem Statement

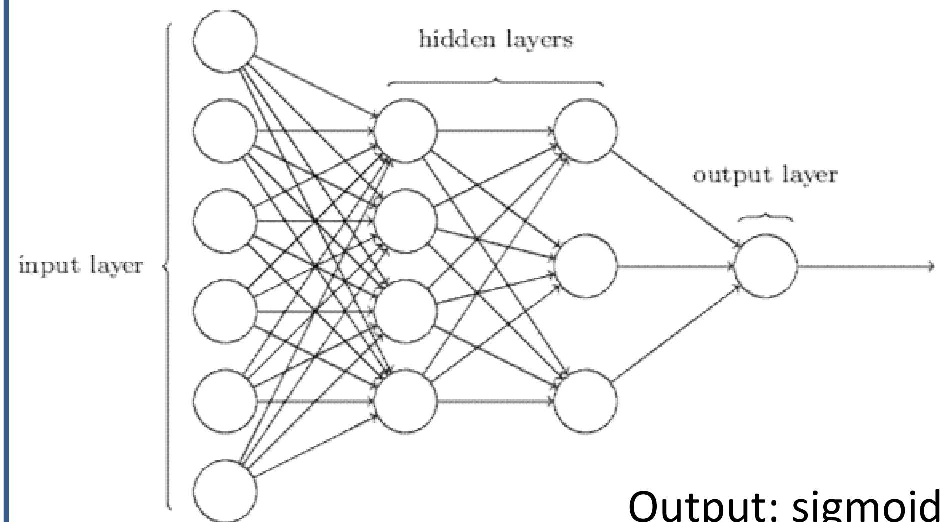


- We are given a number of “training” data instances
- E.g. images of digits, along with information about which digit the image represents
- Tasks:
  - Binary recognition: Is this a “2” or not
  - Multi-class recognition: Which digit is this? Is this a digit in the first place?

Typical Problem statement: binary classification

Training data





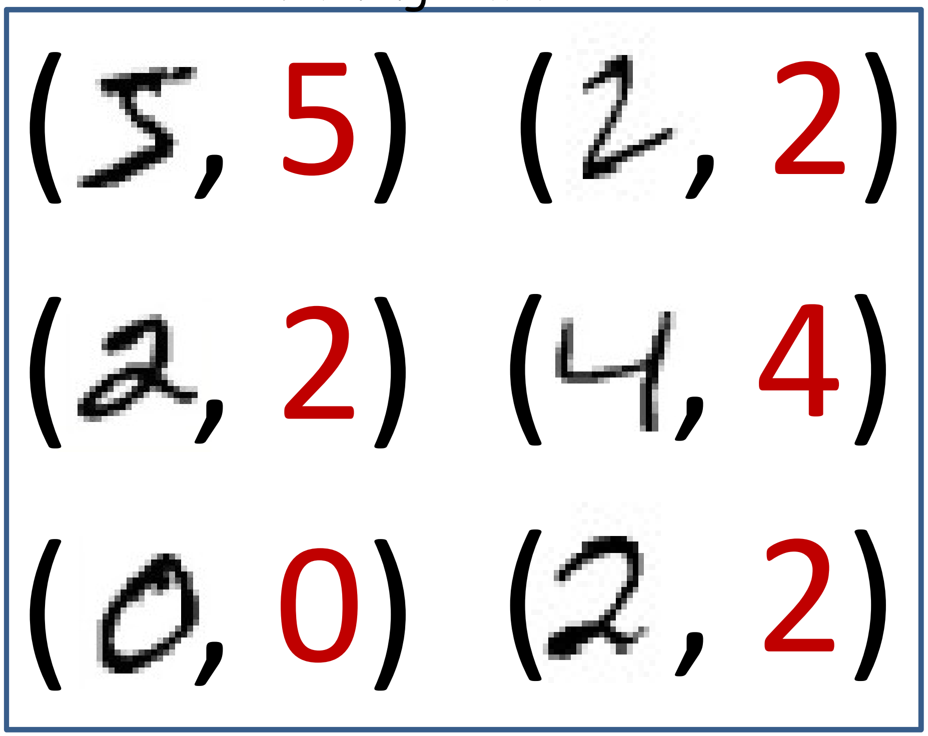
Output: sigmoid

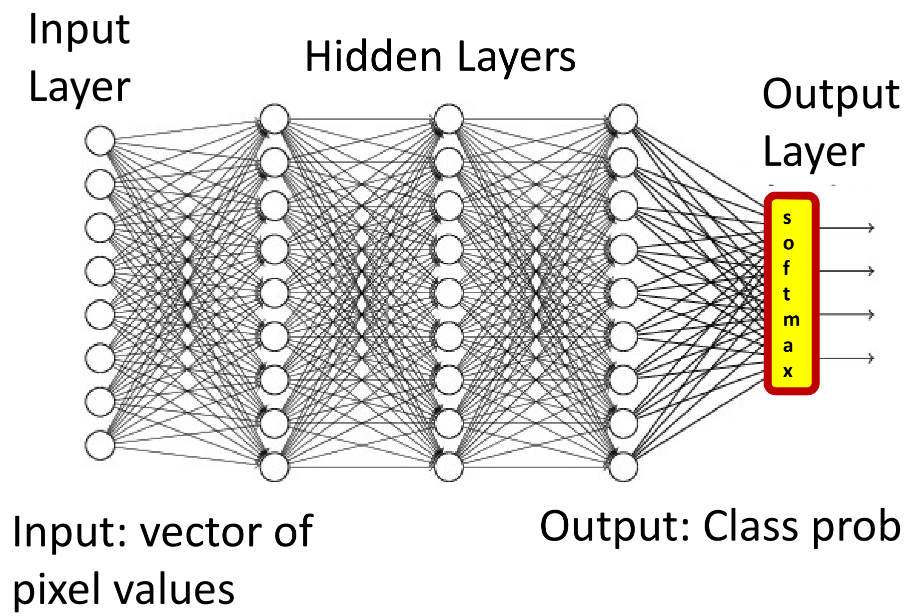
Input: vector of pixel values

- Given, many positive and negative examples (training data),
  - learn all weights such that the network does the desired job

Typical Problem statement: multiclass classification

Training data





- Given, many positive and negative examples (training data),
  - learn all weights such that the network does the desired job

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

What is the divergence div()?

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \dots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

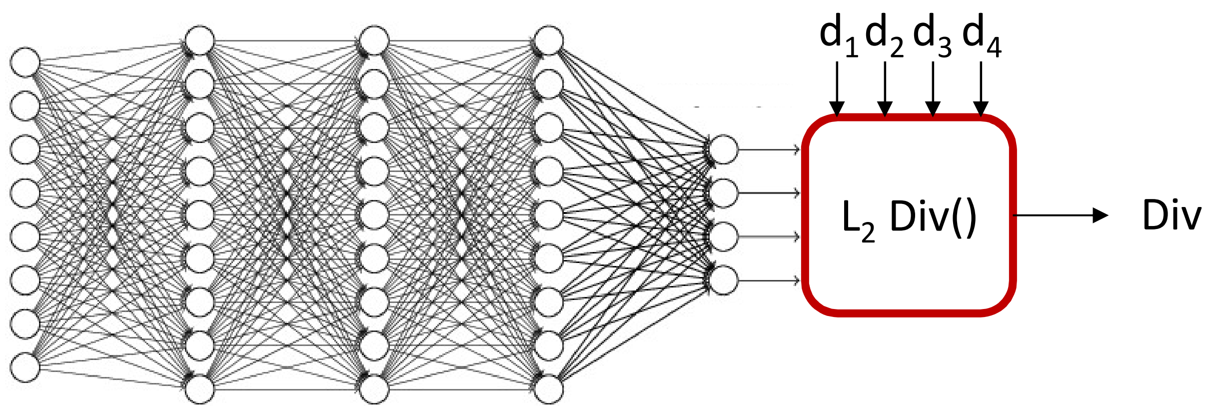
- Minimize the following function

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

What is the divergence div()?

Note: For Loss(W) to be differentiable w.r.t W, div() must be differentiable

Examples of divergence functions



- For real-valued output vectors, the (scaled) L_2 divergence is popular

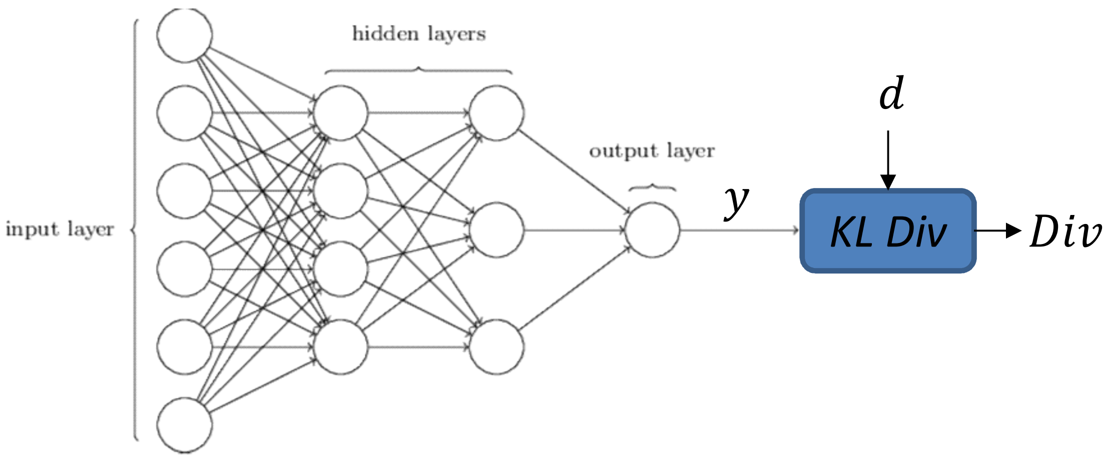
$$Div(Y, d) = \frac{1}{2}\|Y - d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

  - Squared Euclidean distance between true and desired output
  - Note: this is differentiable

$$\frac{dDiv(Y, d)}{dy_i} = (y_i - d_i)$$

$$\nabla_Y Div(Y, d) = [y_1 - d_1, y_2 - d_2, \dots]$$

For binary classifier



- For binary classifier with scalar output, $Y \in (0,1)$, d is $0/1$, the cross entropy between the probability distribution $[Y, 1 - Y]$ and the ideal output probability $[d, 1 - d]$ is popular

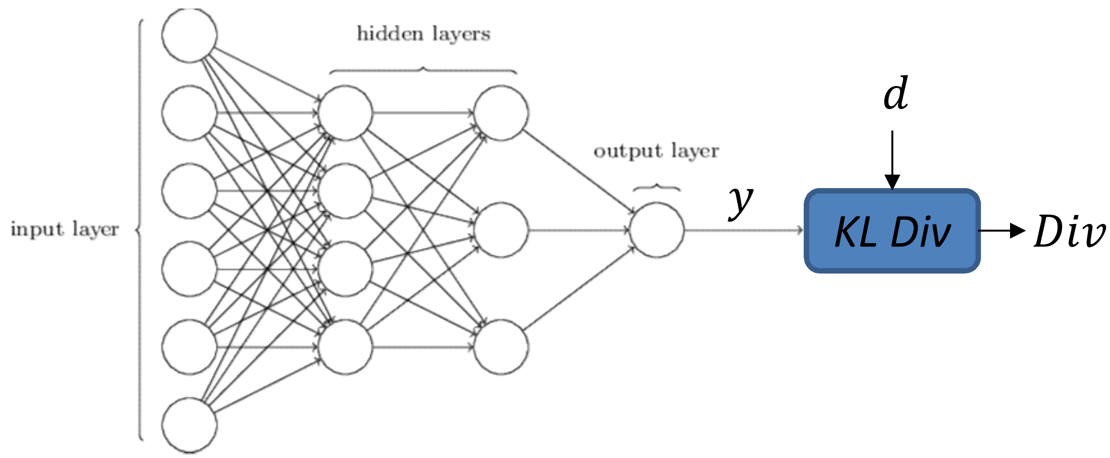
$$Div(Y, d) = -dlogY - (1 - d)\log(1 - Y)$$

  – Minimum when $d = Y$

- Derivative

$$\frac{dDiv(Y,d)}{dY} = \begin{cases} -\frac{1}{Y} & if \;\; d = 1 \\ \frac{1}{1 - Y} & if \; d = 0 \end{cases}$$

For binary classifier



- For binary classifier with scalar output, Y ∈ $(0,1)$, d is $0/1$, the cross entropy between the probability distribution $[Y, 1-Y]$ and the ideal output probability $[d, 1-d]$ is popular

$$Div(Y,d) = -dlogY - (1-d)\log(1-Y)$$

  – Minimum when d $= Y$

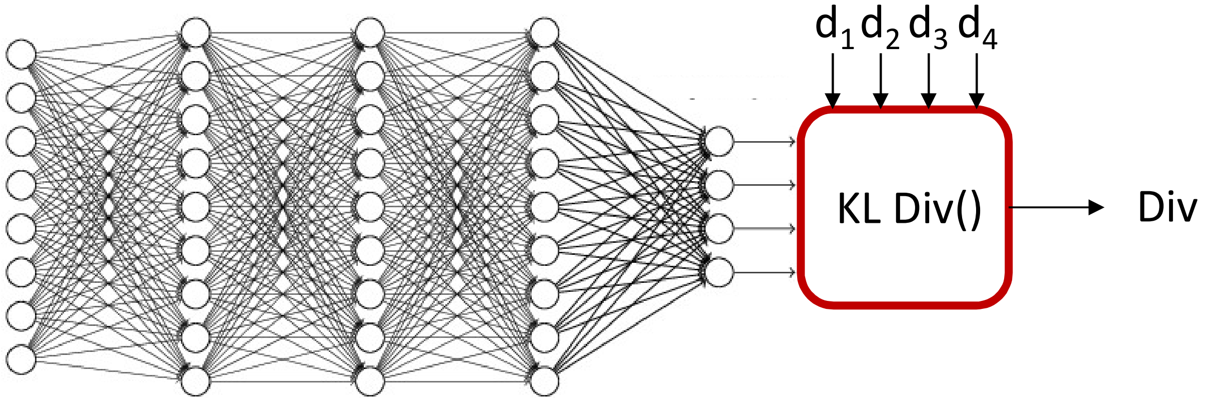
- Derivative

$$\frac{dDiv(Y,d)}{dY} = \begin{cases} -\frac{1}{Y} & if \;\; d = 1 \\ \frac{1}{1-Y} & if \; d = 0 \end{cases}$$

Note: when $y = d$ the derivative is *not* 0

Even though $div() = 0$ (minimum) *when y = d*

For multi-class classification



- Desired output d is a one hot vector $[0\ 0 \ldots 1\ \ldots 0\ 0\ 0\ ]$ with the 1 in the c-th position (for class c)
- Actual output will be probability distribution $[y_1, y_2, \ldots]$
- The cross-entropy between the desired one-hot output and actual output:

$$Div(Y, d) = -\sum_i d_i \log y_i = -\log y_c$$

- Derivative

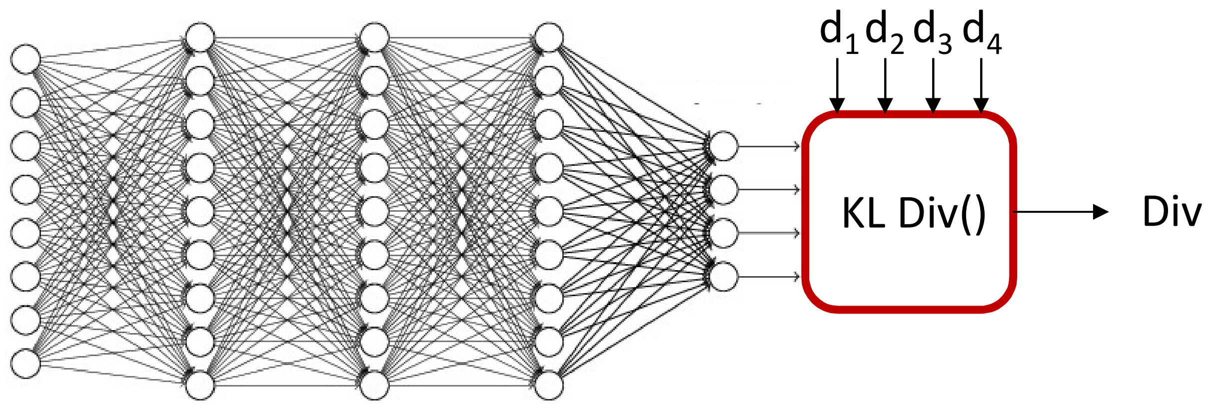
$$\frac{dDiv(Y, d)}{dY_i} = \begin{cases} -\frac{1}{y_c} & for\ the\ c-th\ component \\ 0 & for\ remaining\ component \end{cases}$$

$$\nabla_Y Div(Y, d) = \left[0\ 0\ \ldots \frac{-1}{y_c} \ldots 0\ 0\right]$$

If $y_c < 1$, the slope is negative w.r.t. y_c

Indicates *increasing* y_c will *reduce* divergence

For multi-class classification



- Desired output d is a one hot vector $[0\ 0 \ldots 1\ \ldots 0\ 0\ 0\ ]$ with the 1 in the c-th position (for class c)
- Actual output will be probability distribution $[y_1, y_2, \ldots]$
- The cross-entropy between the desired one-hot output and actual output:

$$Div(Y, d) = -\sum_i d_i \log y_i = -\log y_c$$

- Derivative

$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\frac{1}{y_c} & for\ the\ c-th\ component \\ 0 & for\ remaining\ component \end{cases}$$

$$\nabla_Y Div(Y,d) = \left[0\ 0\ \ldots \frac{-1}{y_c} \ldots 0\ 0\right]$$

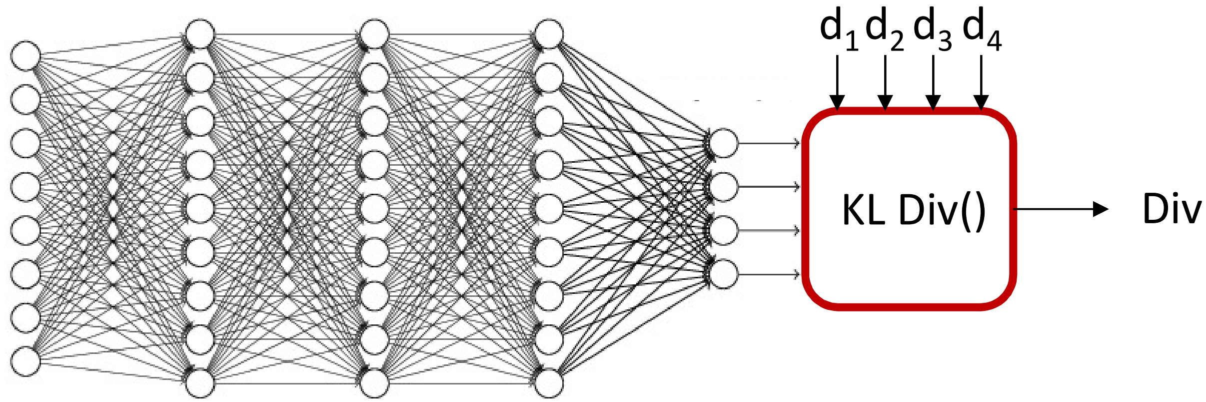
If $y_c < 1$, the slope is negative w.r.t. y_c

Indicates *increasing* y_c will *reduce* divergence

Note: when $y = d$ the derivative is *not* 0

Even though $div() = 0$ (minimum) *when y = d*

For multi-class classification



- It is sometimes useful to set the target output to $[\epsilon \ \ \epsilon \ldots (1-(K-1)\epsilon) \ldots \epsilon \ \ \epsilon \ \ \epsilon]$ with the value $1-(\mathrm{K}-1)\epsilon$ in the c-th position (for class c) and ϵ elsewhere for some small ϵ
  - "Label smoothing" -- aids gradient descent
- The cross-entropy remains:

$$Div(Y,d) = -\sum_i d_i \log y_i$$

- Derivative

$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\dfrac{1-(K-1)\epsilon}{y_c} & for\ the\ c-th\ component \\ -\dfrac{\epsilon}{y_i} for\ remaining\ components & \end{cases}$$

Problem Setup: Things to define

- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$

- Minimize the following function

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

ALL TERMS HAVE BEEN DEFINED

Problem Setup

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$
- The error on the i^th instance is $div(Y_i, d_i)$
  - $Y_i = f(X_i; W)$
- The loss

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

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

Recap: Gradient Descent Algorithm

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

To minimize any function f(x) w.r.t x

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

Recap: Gradient Descent Algorithm

- In order to minimize any function $f(x)$ w.r.t. x
- Initialize:
  - x^0
  - $k = 0$

- do
  - For every component i
    - $x_i^{k+1} = x_i^k - \eta^k \frac{\partial f}{\partial x_i}$
  - $k = k + 1$
- while $\left|f\left(x^k\right) - f\left(x^{k-1}\right)\right| > \varepsilon$

Explicitly stating it by component

Training Neural Nets through Gradient Descent

Total training Loss:

$$Loss = \frac{1}{T}\sum_t Div(\mathbf{Y}_t, \mathbf{d}_t)$$

- 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

Training Neural Nets through Gradient Descent

Total training Loss:

$$Loss = \frac{\mathbf{1}}{\boldsymbol{T}} \sum_{\boldsymbol{t}} Div(\boldsymbol{Y}_{\boldsymbol{t}}, \boldsymbol{d}_{\boldsymbol{t}})$$

- Gradient descent algorithm:
- Initialize all weights $\left\{w_{ij}^{(k)}\right\}$
- 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 Err has converged

The derivative

Total training Loss:

$$Loss = \frac{1}{T}\sum_{t} Div(\boldsymbol{Y}_t, \boldsymbol{d}_t)$$

- Computing the derivative

Total derivative:

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

Training by gradient descent

- Initialize all weights $\left\{w_{ij}^{(k)}\right\}$
- Do:
  - For all $i, j, k,$ initialize $\frac{dLoss}{dw_{i,j}^{(k)}} = 0$
  - For all $t = 1: T$
    - For every layer k for all i, j:
      - Compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
      - $\frac{dLoss}{dw_{i,j}^{(k)}} += \frac{d\boldsymbol{Div}(\boldsymbol{Y_t}, \boldsymbol{d_t})}{dw_{i,j}^{(k)}}$
  - For every layer k for all i, j:

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

- Until Err has converged

The derivative

Total training Loss:

$$Loss = \frac{\mathbf{1}}{\boldsymbol{T}} \sum_{\boldsymbol{t}} Div(\boldsymbol{Y}_{\boldsymbol{t}}, \boldsymbol{d}_{\boldsymbol{t}})$$

Total derivative:

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

- 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 Δx $\Rightarrow$ $\Delta y \approx \frac{dy}{dx} \Delta x$

For any differentiable function

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

with partial derivatives

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

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

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

Both by the definition

$\Delta y = \nabla f \Delta x$

Calculus Refresher: Chain rule

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

$$\boxed{\frac{dy}{dx} = \frac{\partial f}{\partial g(x)} \frac{dg(x)}{dx}}$$

Check – we can confirm that : $\Delta y = \frac{dy}{dx} \Delta x$

$$z = g(x) \Rightarrow \Delta z = \frac{dg(x)}{dx} \Delta x$$

$$y = f(z) \Rightarrow \Delta y = \frac{df}{dz} \Delta z = \frac{df}{dz} \frac{dg(x)}{dx} \Delta x$$

Calculus Refresher: Distributed Chain rule

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

$$\boxed{\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)}\frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)}\frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)}\frac{dg_M(x)}{dx}}$$

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

Let $z_i = g_i(x)$

$$\Delta y = \frac{\partial f}{\partial z_1}\Delta z_1 + \frac{\partial f}{\partial z_2}\Delta z_2 + \cdots + \frac{\partial f}{\partial z_M}\Delta z_M$$

$$\boxed{\Delta y = \frac{\partial f}{\partial z_1}\frac{dz_1}{dx}\Delta x + \frac{\partial f}{\partial z_2}\frac{dz_2}{dx}\Delta x + \cdots + \frac{\partial f}{\partial z_M}\frac{dz_M}{dx}\Delta x}$$

$$\boxed{\Delta y = \left(\frac{\partial f}{\partial g_1(x)}\frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)}\frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)}\frac{dg_M(x)}{dx}\right)\Delta x}$$

Calculus Refresher: Distributed Chain rule

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

$$\boxed{\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)}\frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)}\frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)}\frac{dg_M(x)}{dx}}$$

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

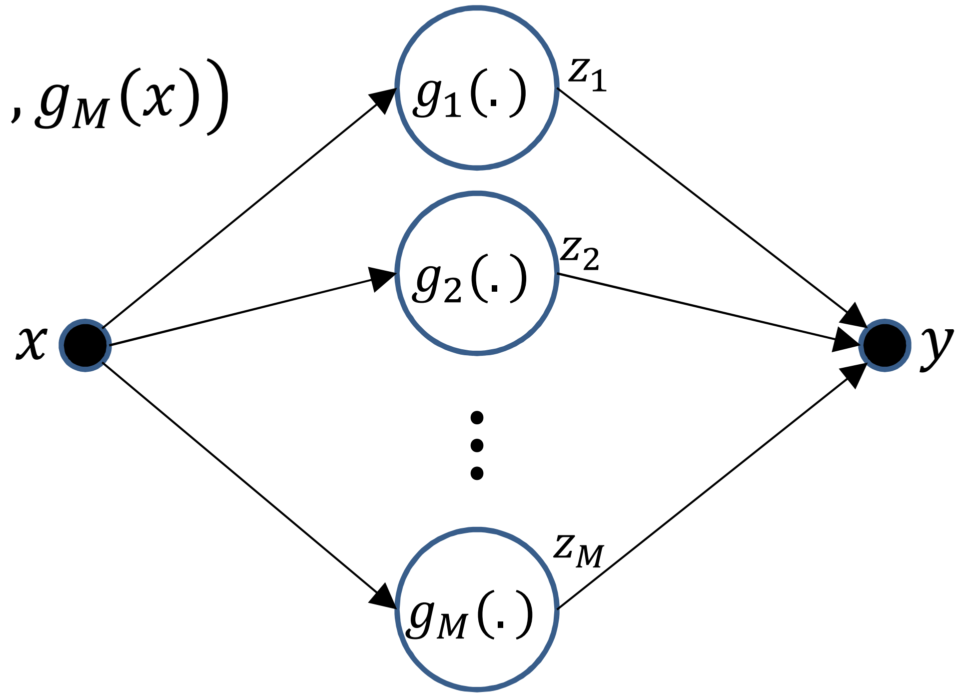
$$\Delta y = \frac{\partial f}{\partial g_1(x)}\Delta g_1(x) + \frac{\partial f}{\partial g_2(x)}\Delta g_2(x) + \cdots + \frac{\partial f}{\partial g_M(x)}\Delta g_M(x)$$

$$\boxed{\Delta y = \frac{\partial f}{\partial g_1(x)}\frac{dg_1(x)}{dx}\Delta x + \frac{\partial f}{\partial g_2(x)}\frac{dg_2(x)}{dx}\Delta x + \cdots + \frac{\partial f}{\partial g_M(x)}\frac{dg_M(x)}{dx}\Delta x}$$

$$\boxed{\Delta y = \left(\frac{\partial f}{\partial g_1(x)}\frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)}\frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)}\frac{dg_M(x)}{dx}\right)\Delta x}$$

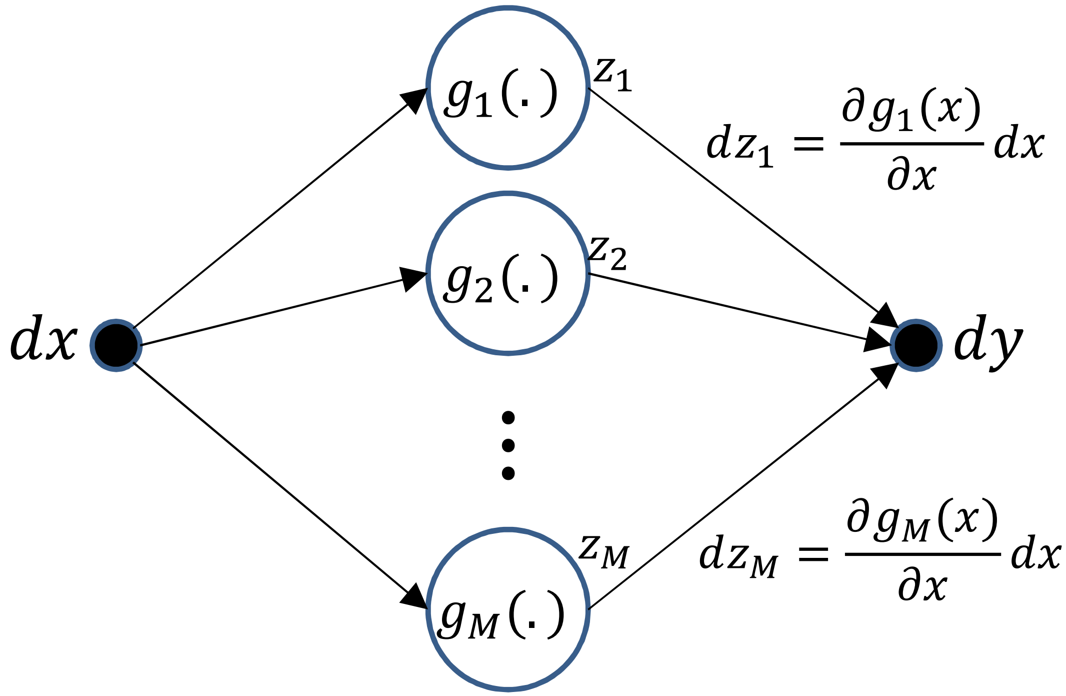
Distributed Chain Rule: Influence Diagram

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



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

Distributed Chain Rule: Influence Diagram

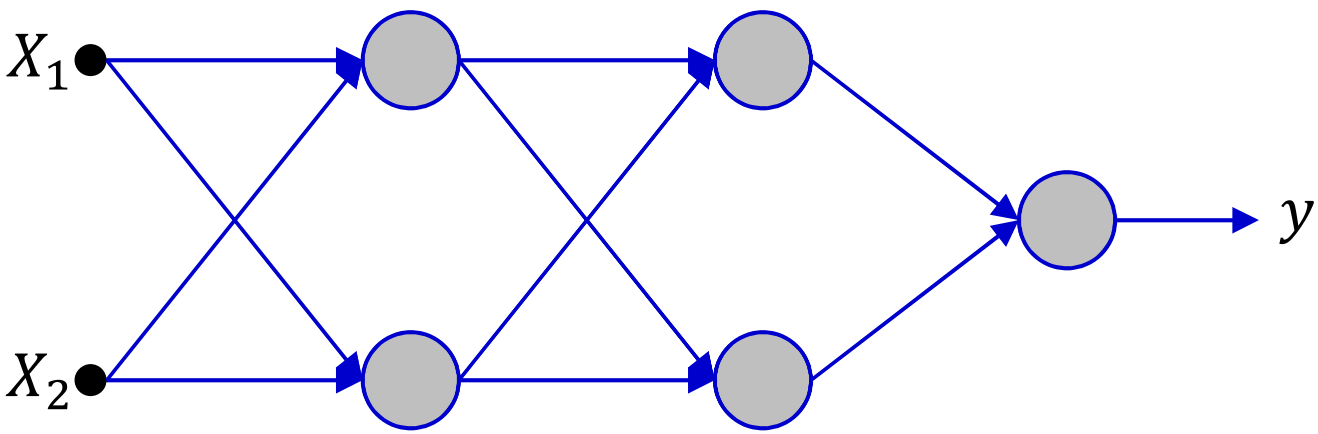


- Small perturbations in x cause small perturbations in each of $g_1 \ldots g_M$, each of which individually additively perturbs y

Returning to our problem

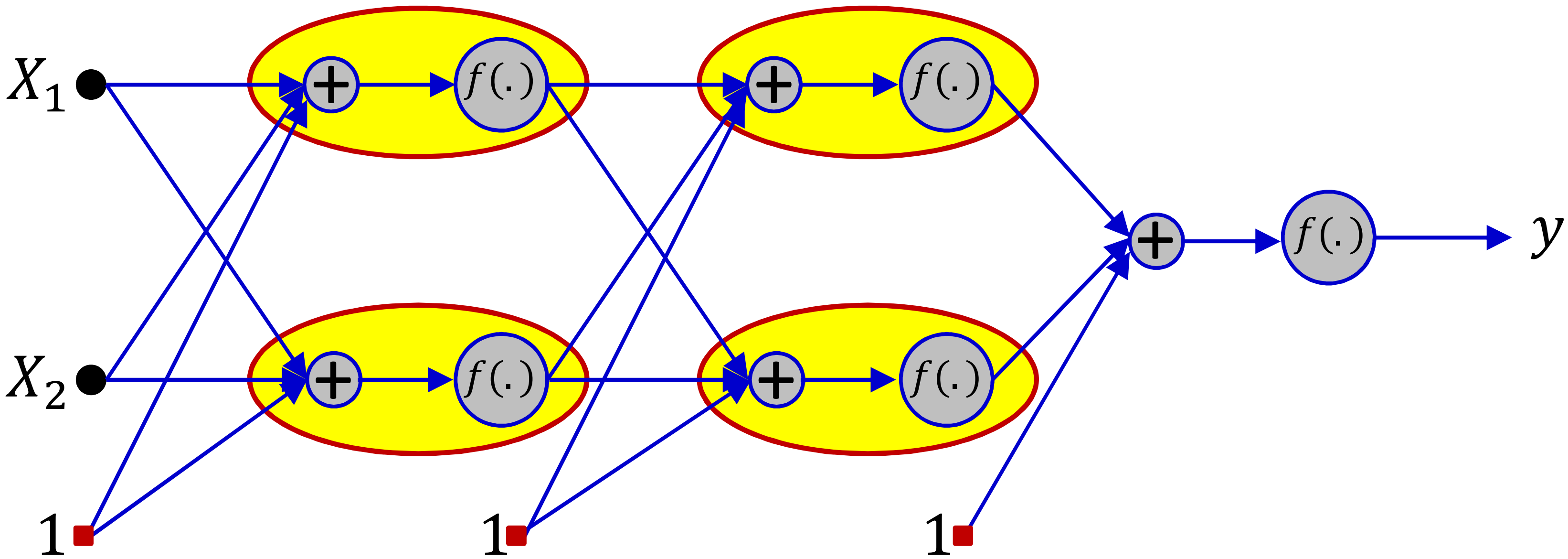
- How to compute $\frac{d\boldsymbol{Div}(\boldsymbol{Y},\boldsymbol{d})}{dw_{i,j}^{(k)}}$

A first closer look at the network



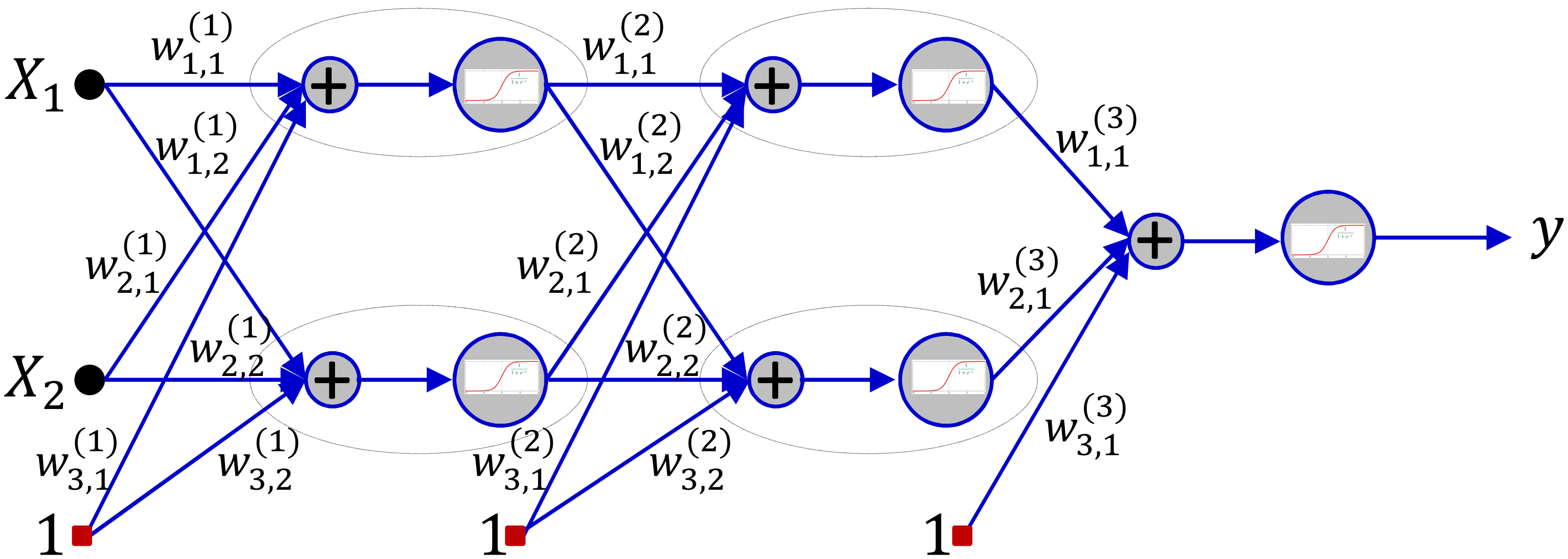
- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs

A first closer look at the network



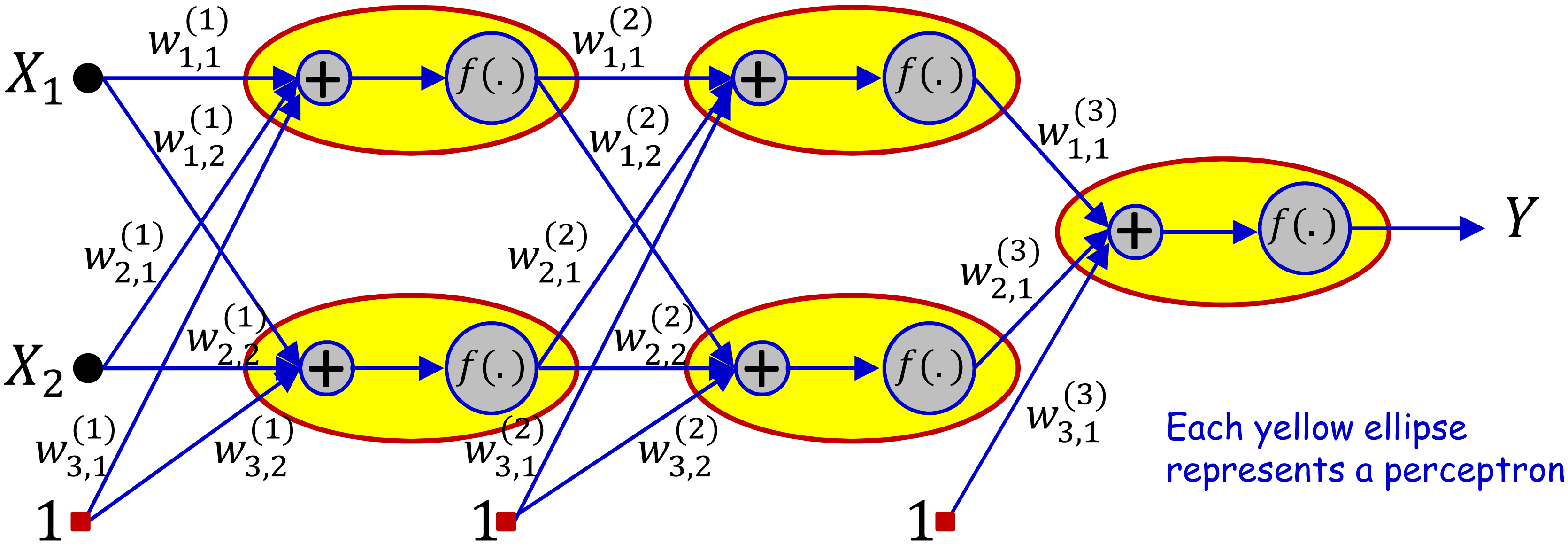
- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Explicitly separating the weighted sum of inputs from the activation

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 and activations shown**
- The overall function is differentiable w.r.t every weight, bias and input

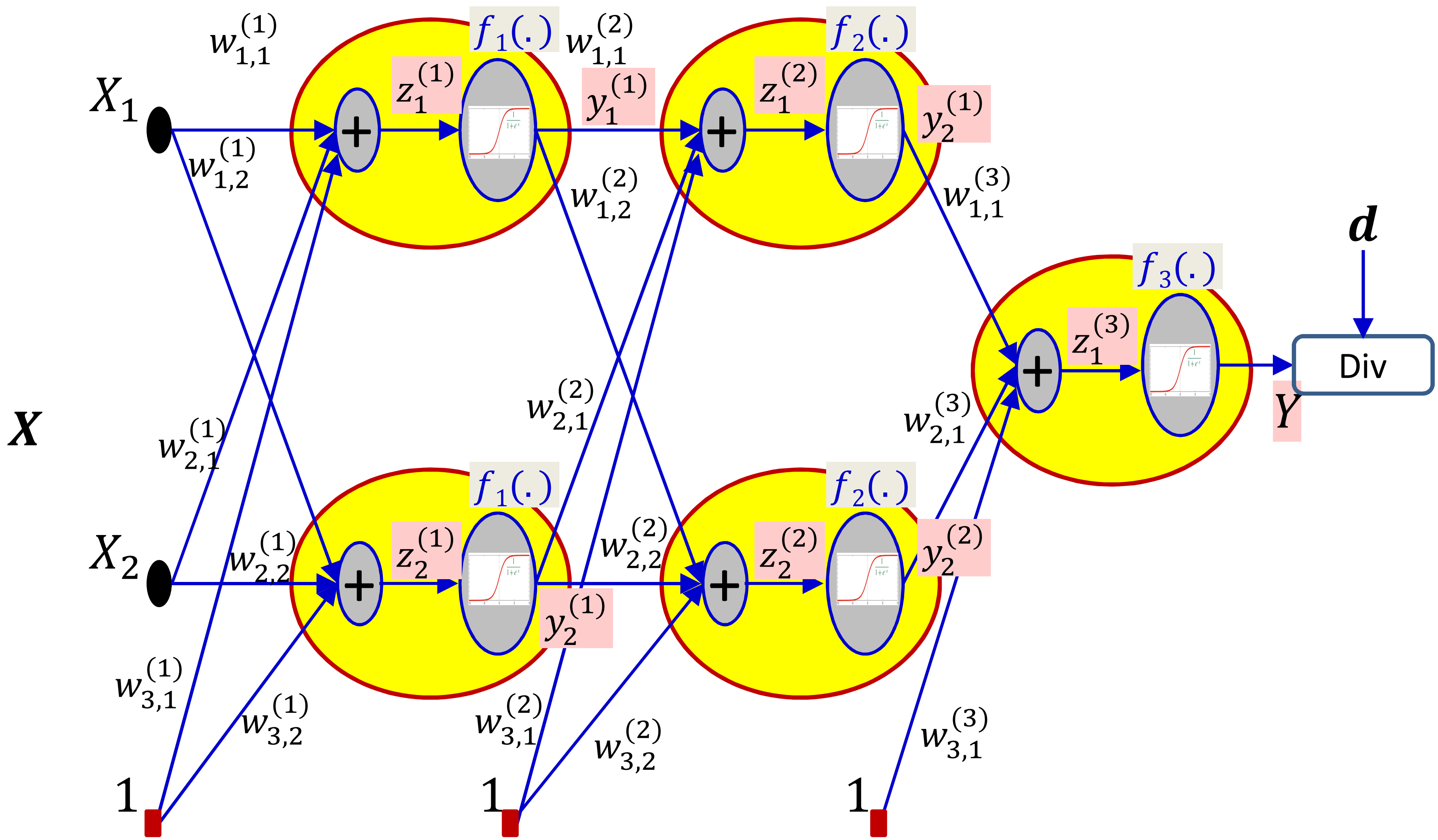
Computing the derivative for a *single* input



- Aim: compute derivative of $Div(Y, d)$ w.r.t. each of the weights

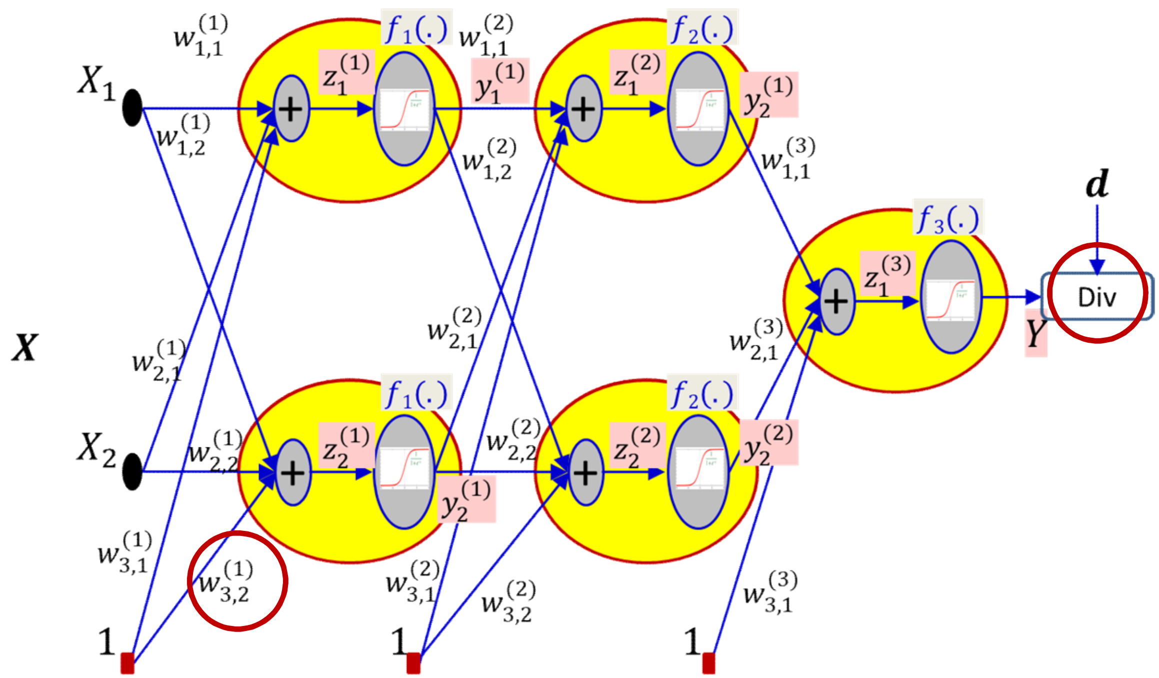
- But first, lets label *all* our variables and activation functions

Computing the derivative for a *single* input



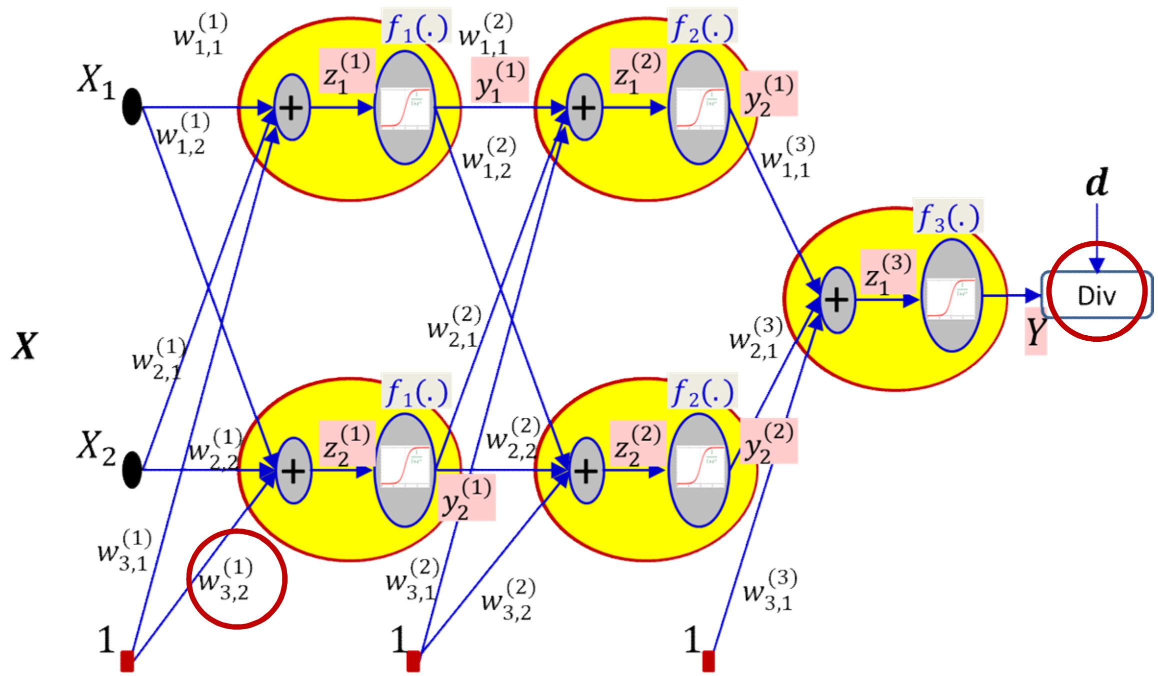
Computing the gradient

- What is: $\frac{dDiv(Y,d)}{dw_{i,j}^{(k)}}$



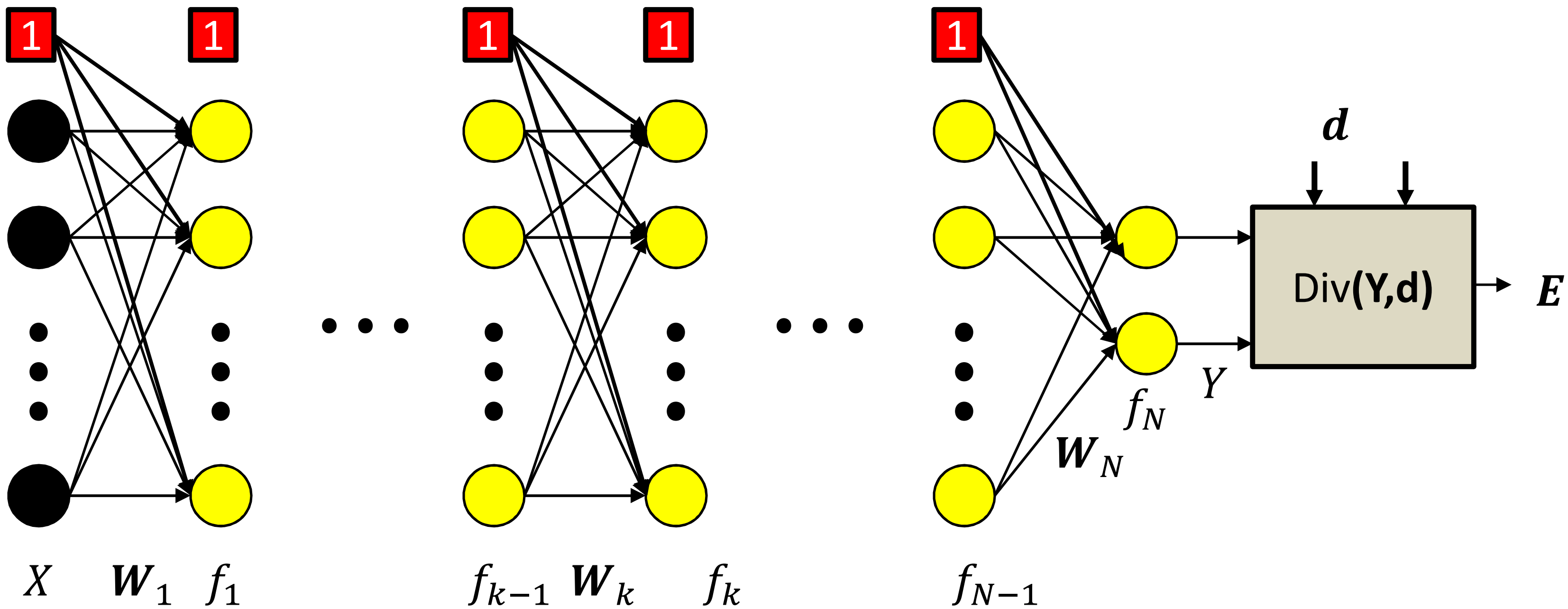
Computing the gradient

- What is: $\frac{dDiv(Y,d)}{dw_{i,j}^{(k)}}$



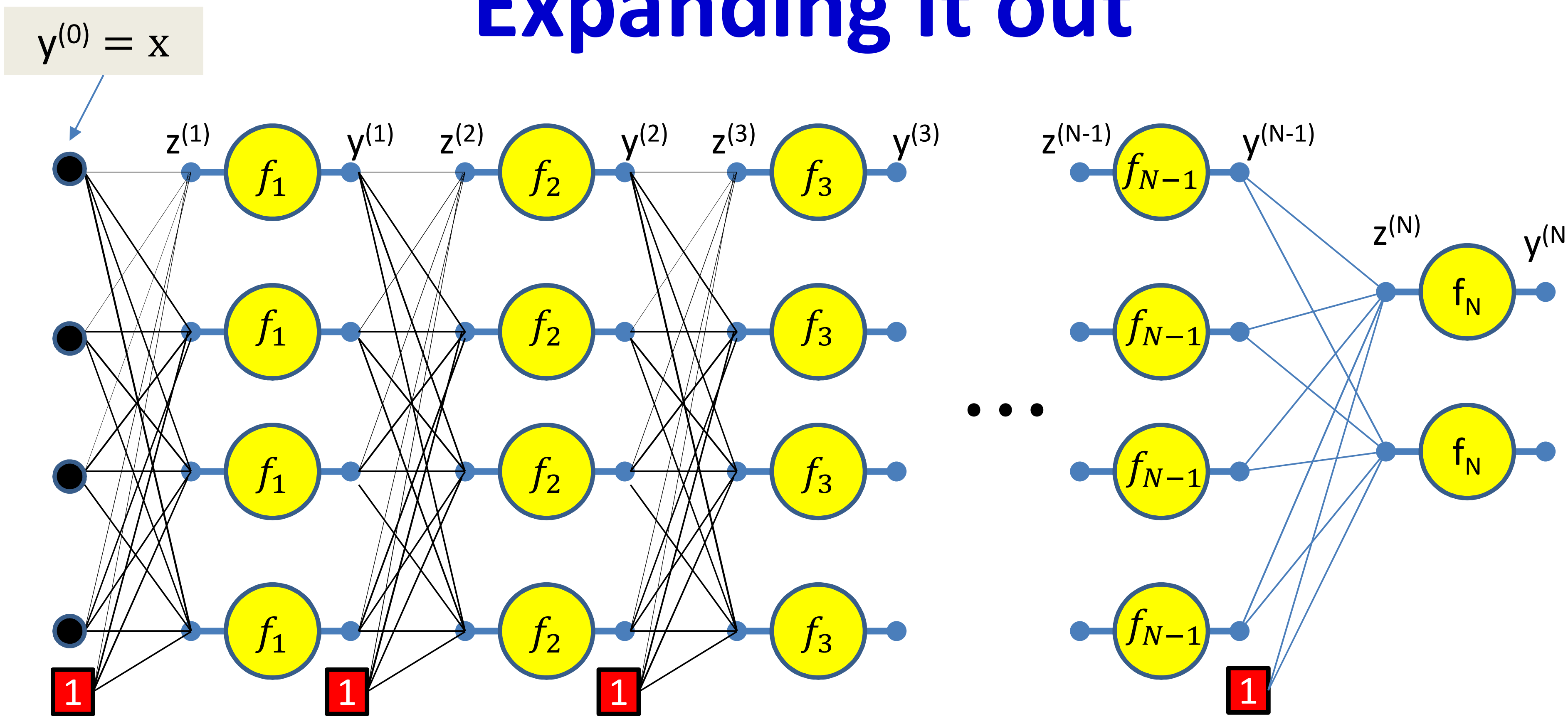
- Note: computation of the derivative requires intermediate and final output values of the network in response to the input

BP: Scalar Formulation



- The network again

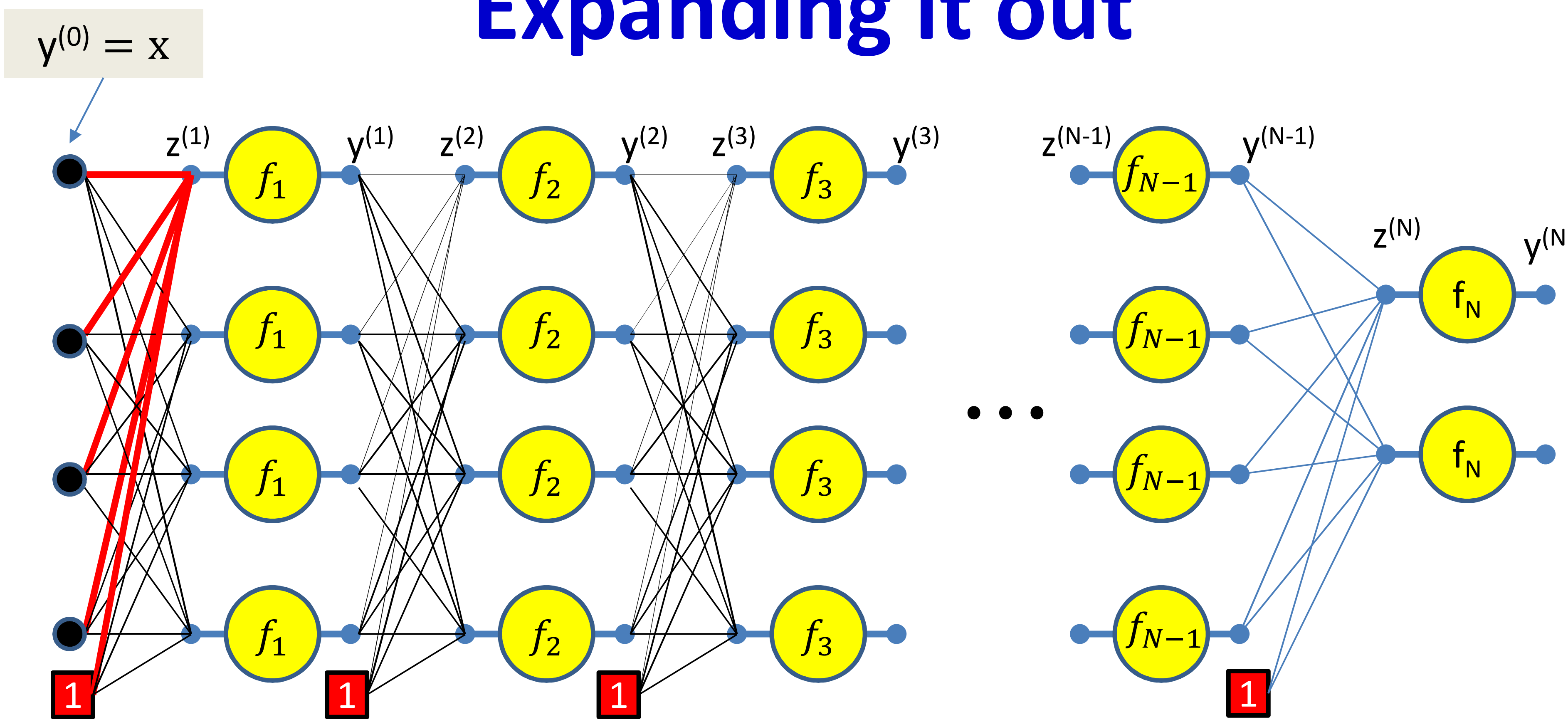
Expanding it out



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

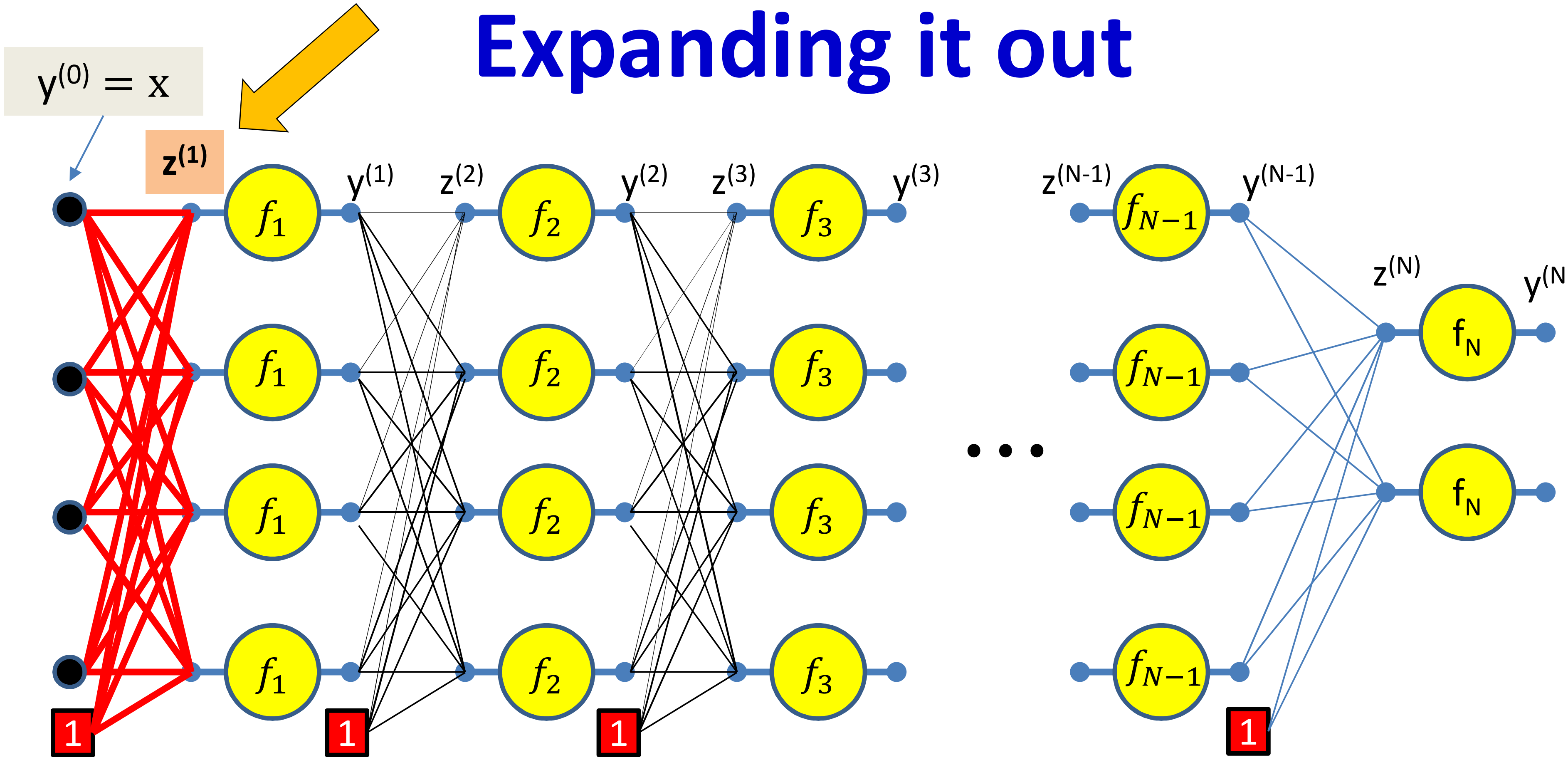
Assuming $w_{0j}^{(k)} = b_j^{(k)}$ and $y_0^{(k)} = 1$ -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases

Expanding it out

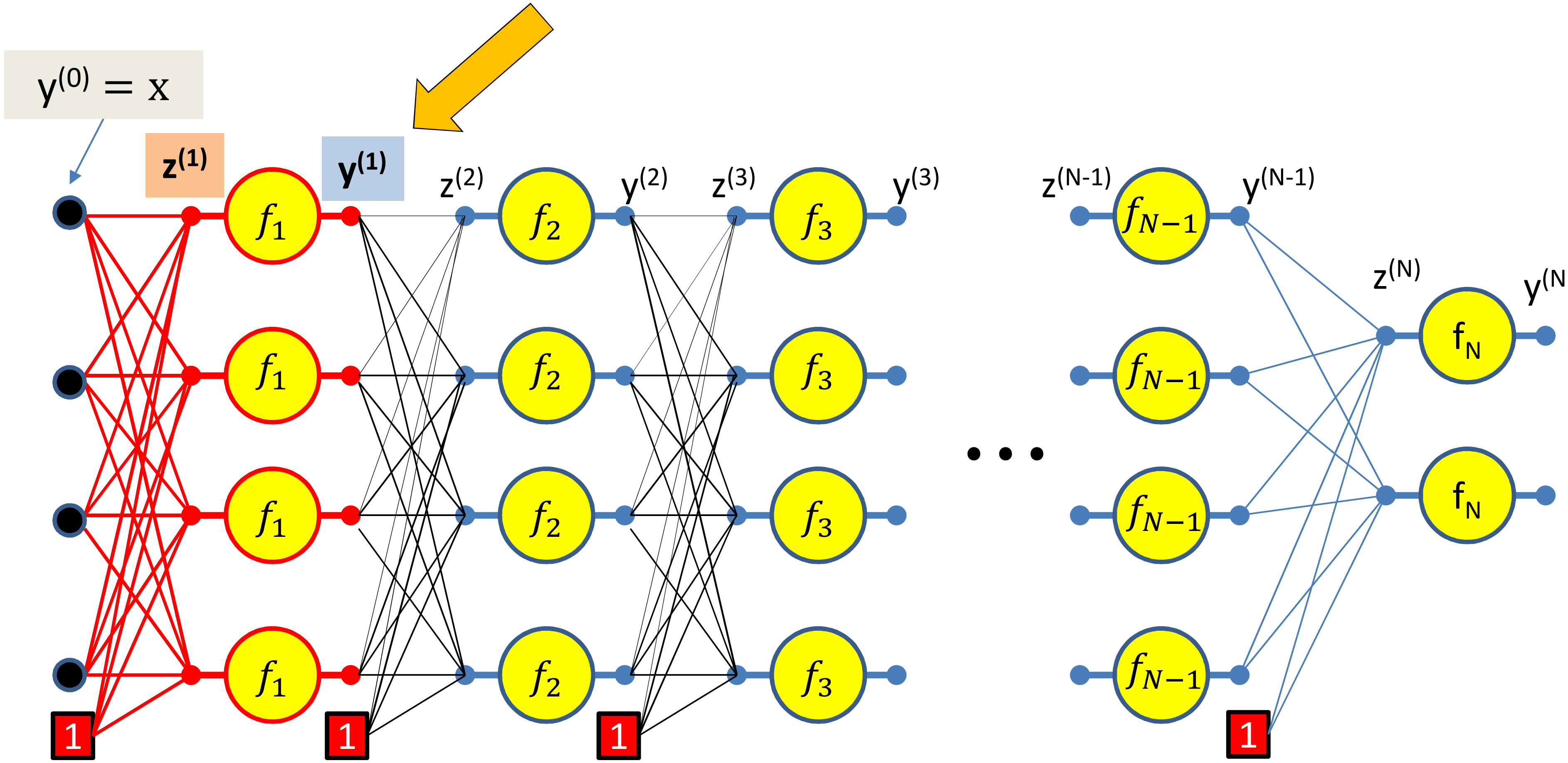


$$z_1^{(1)} = \sum_i w_{i1}^{(1)} y_i^{(0)}$$

Expanding it out

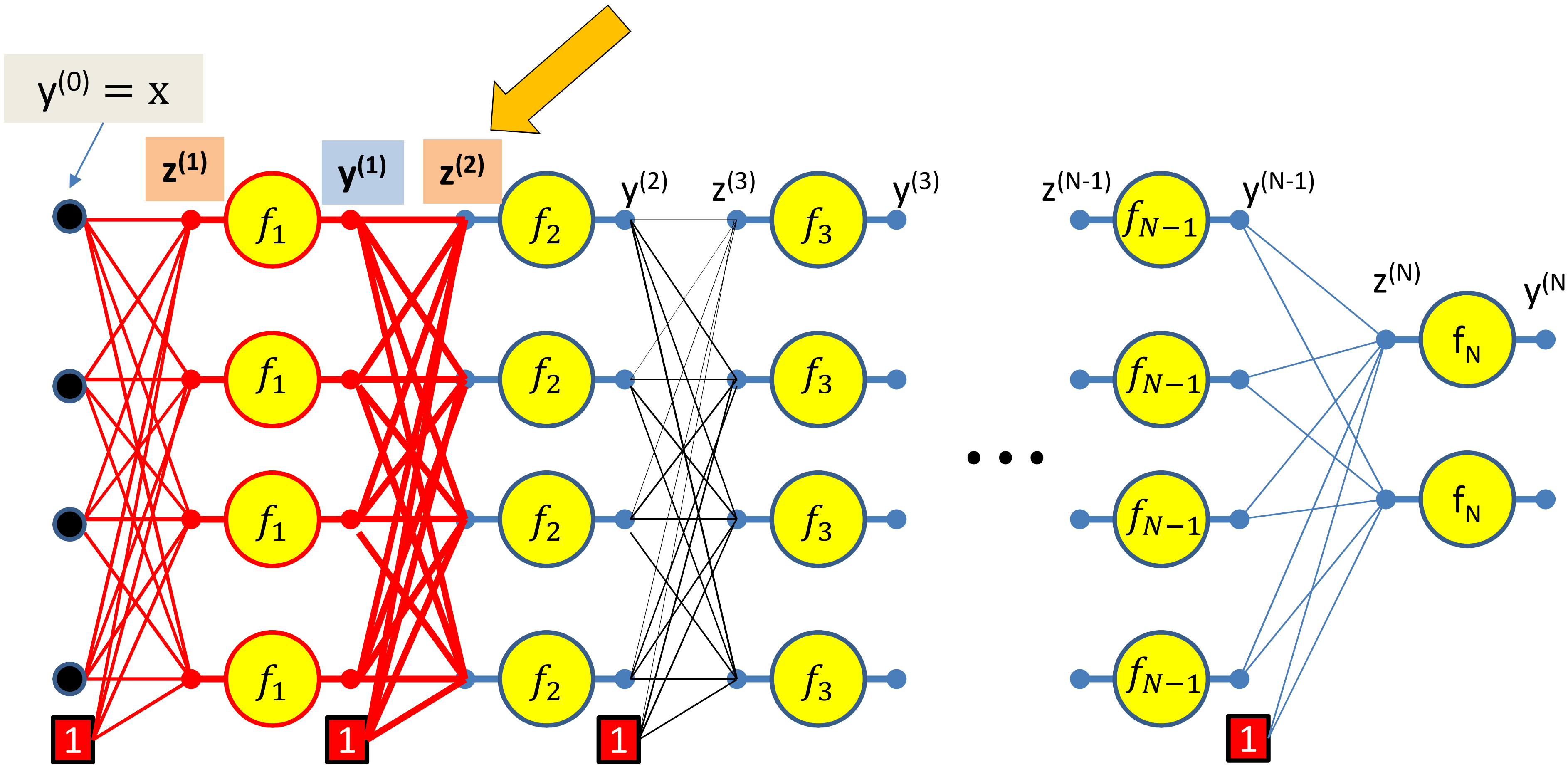


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$



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

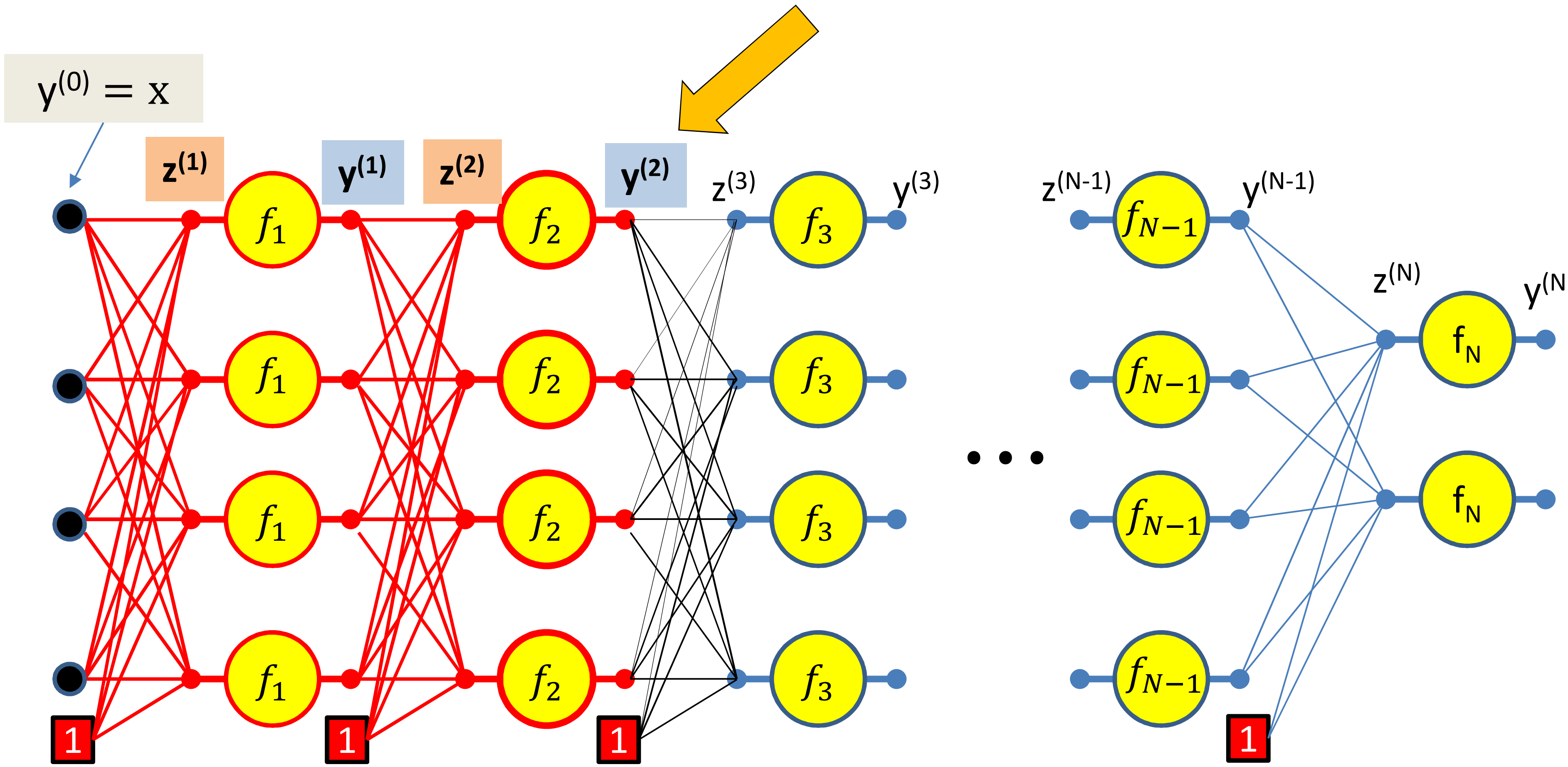
$$\boxed{y_j^{(1)} = f_1\left(z_j^{(1)}\right)}$$



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

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

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

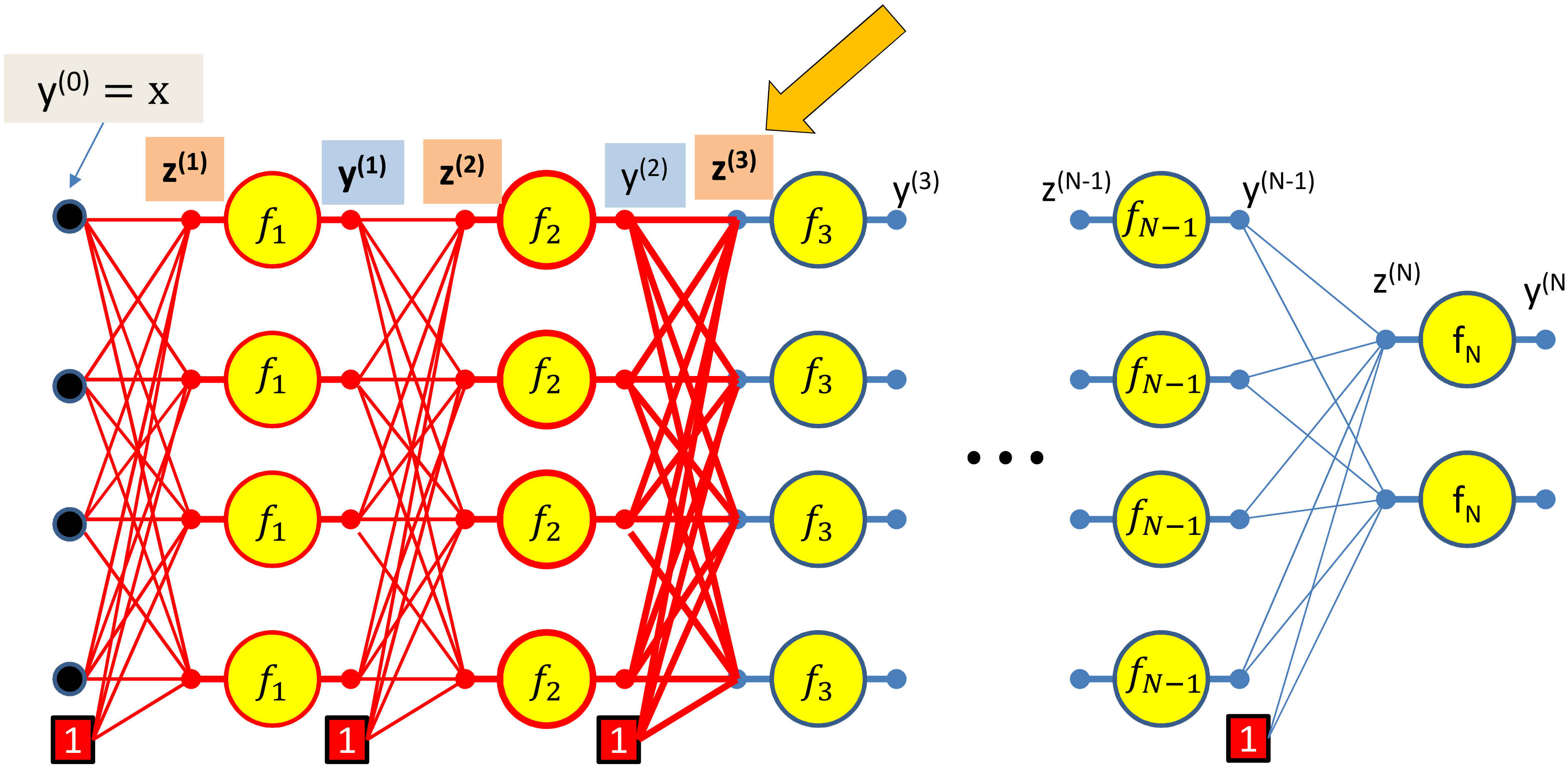


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

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

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

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



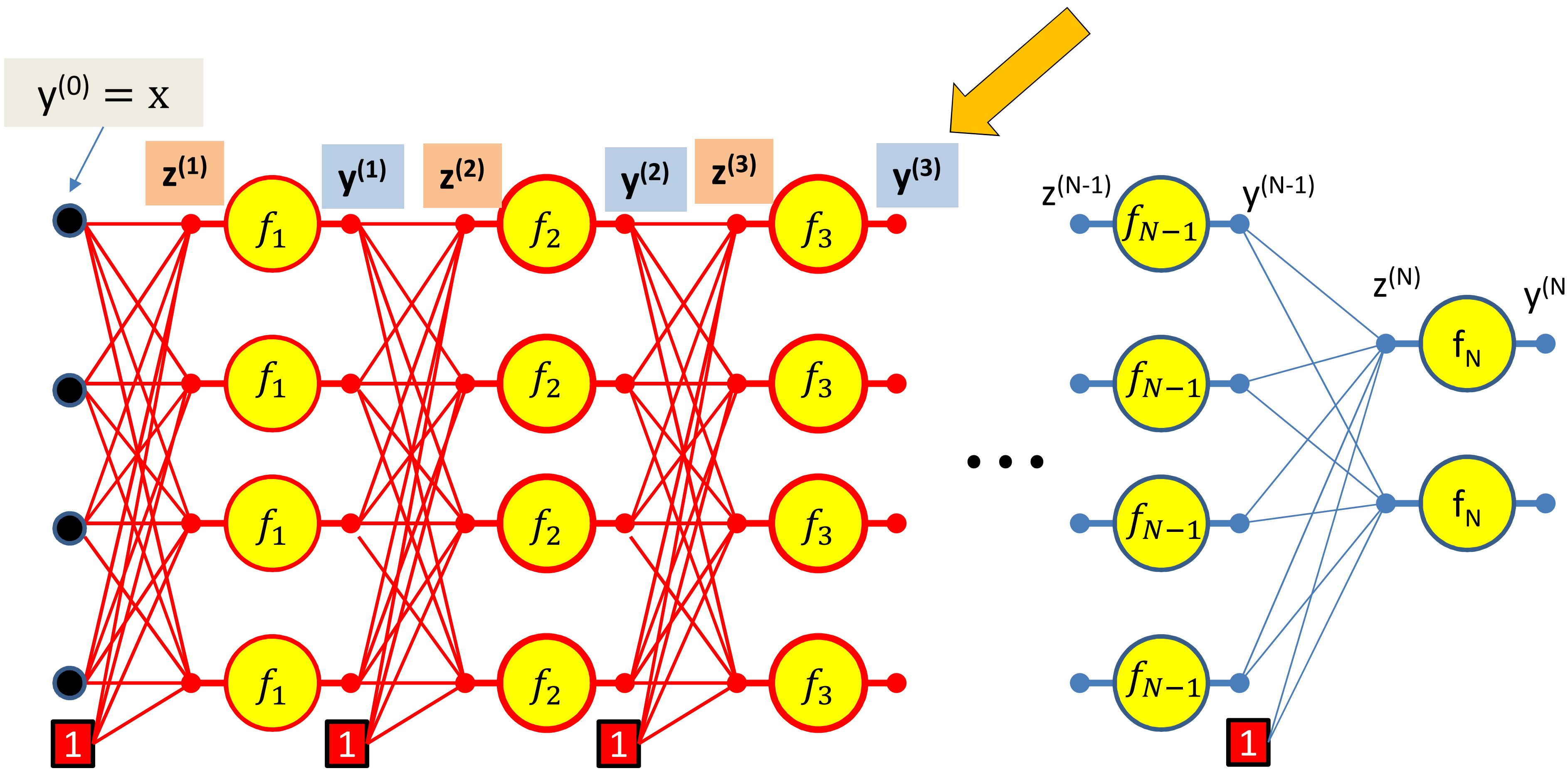
$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$

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

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

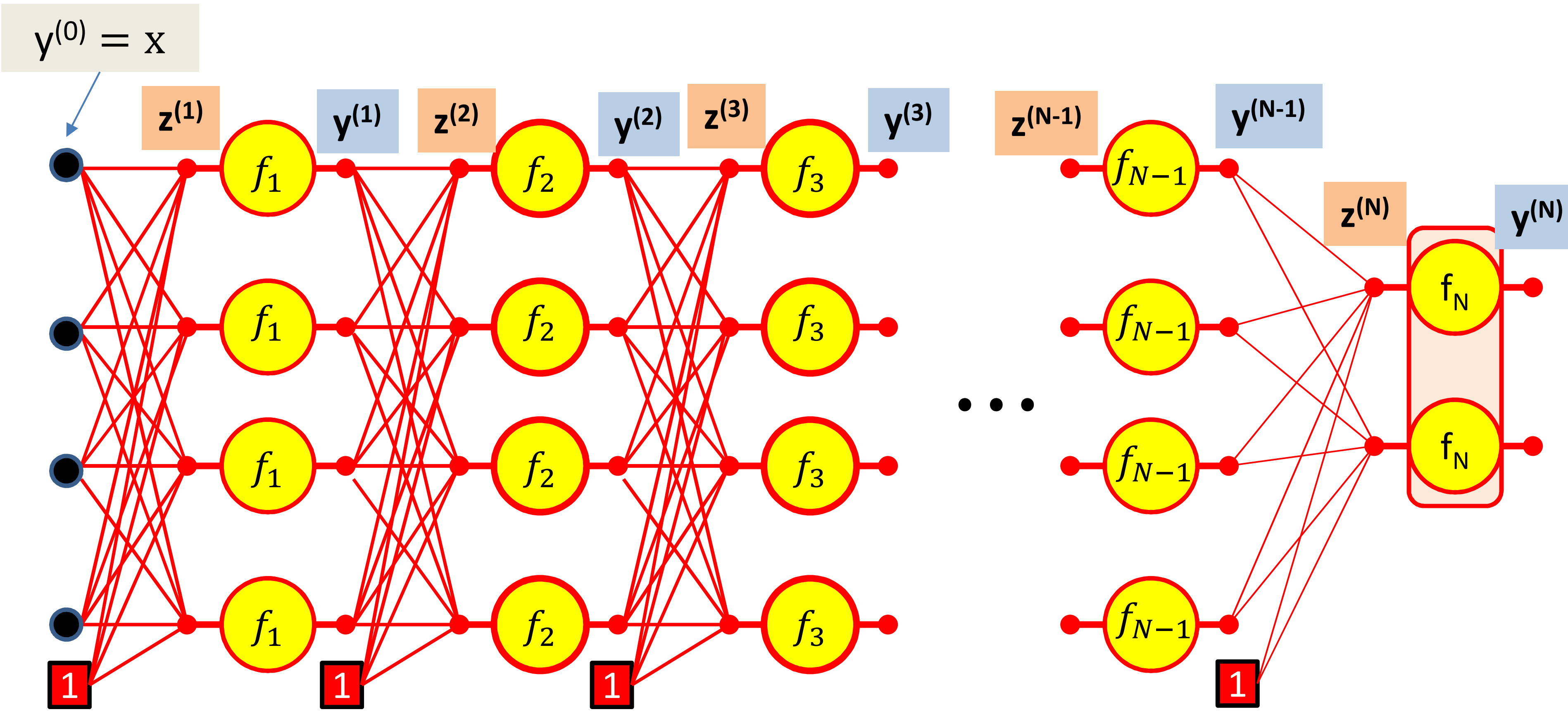
$$y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$

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



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

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

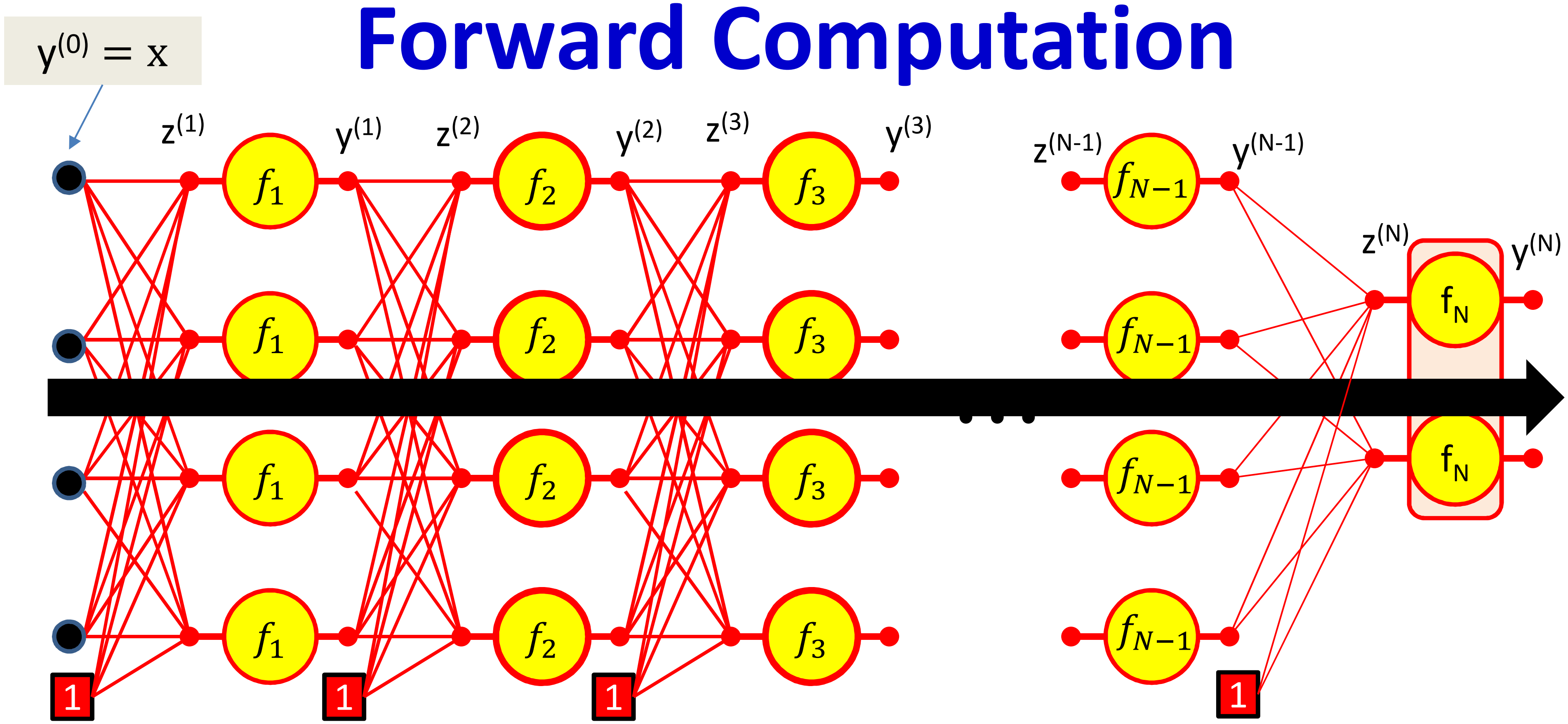


$$y_j^{(N-1)} = f_{N-1}\left(z_j^{(N-1)}\right)$$

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

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

Forward Computation



ITERATE FOR k = 1:N

for j = 1:layer-width

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

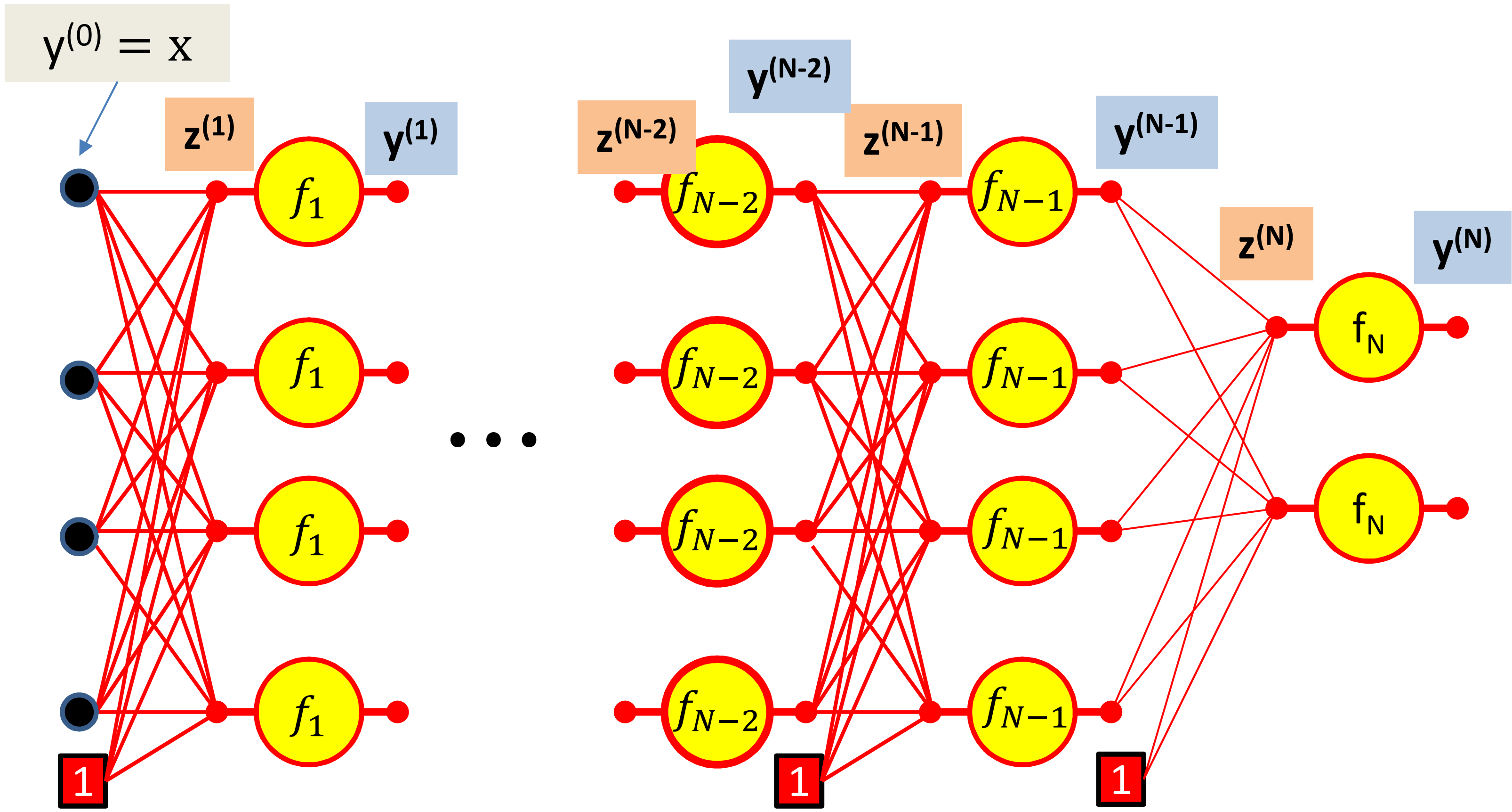
$$z_j^{(k)} = \sum_i w_{ij}^{(k)} y_i^{(k-1)}$$

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

Forward "Pass"

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

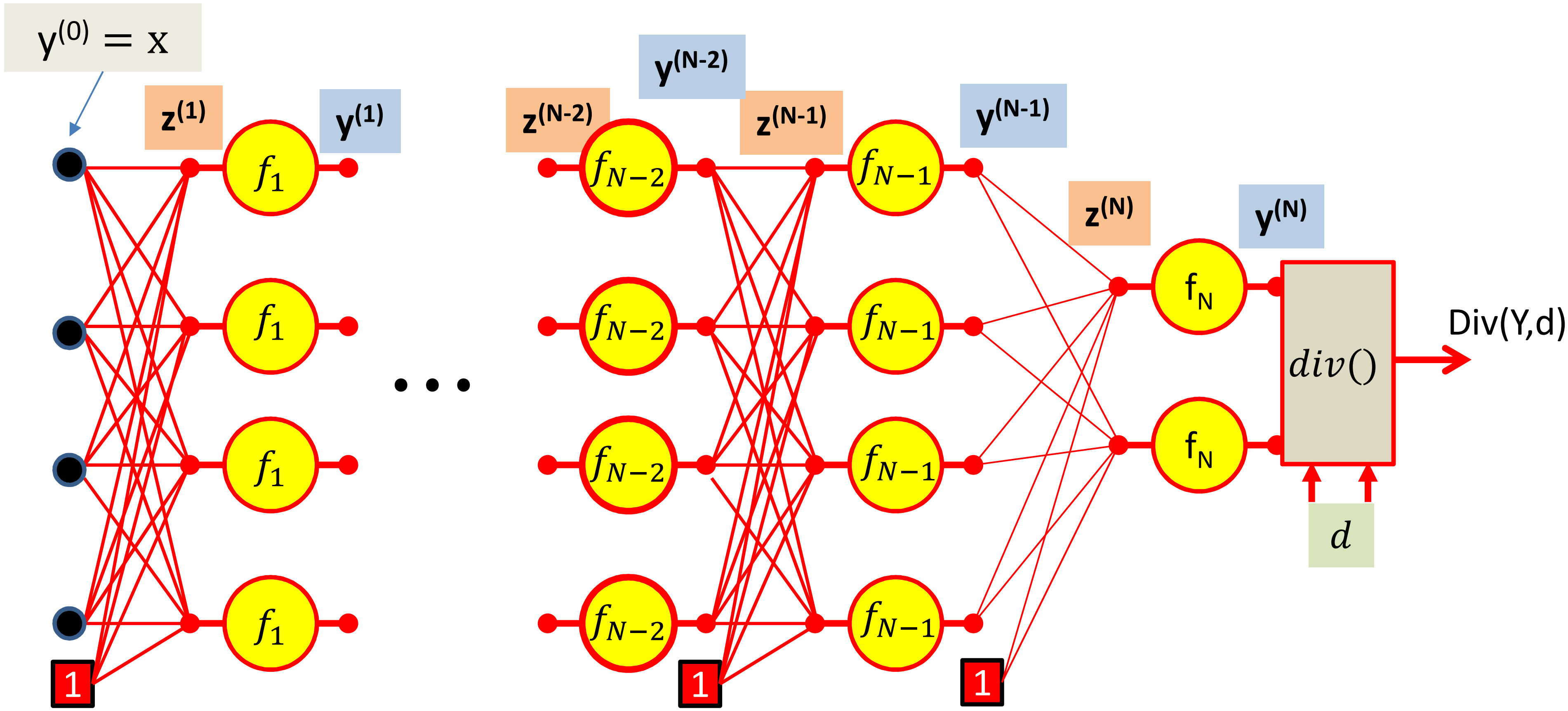
Computing derivatives



We have computed all these intermediate values in the forward computation

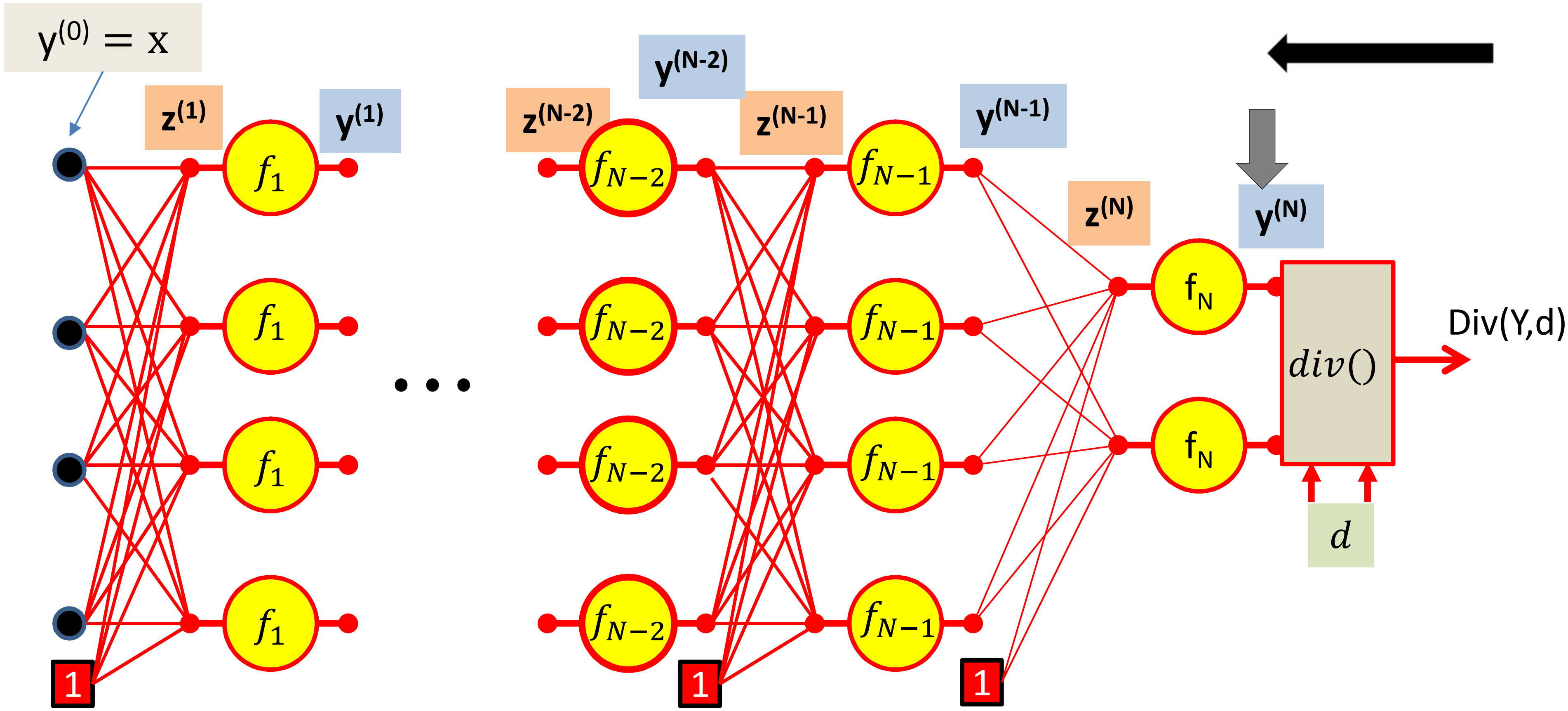
We must remember them – we will need them to compute the derivatives

Computing derivatives



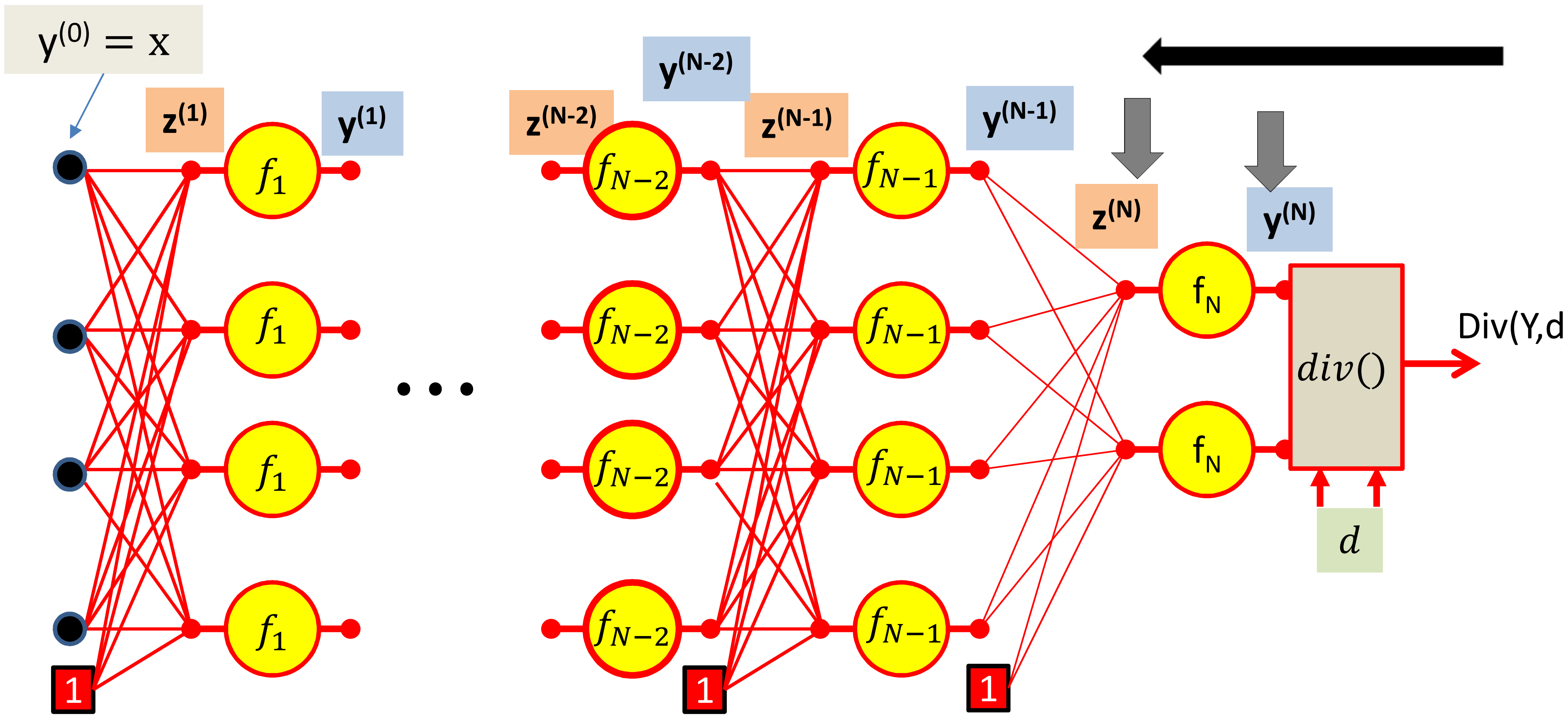
First, we compute the divergence between the output of the net y = $y^{(N)}$ and the desired output d

Computing derivatives



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

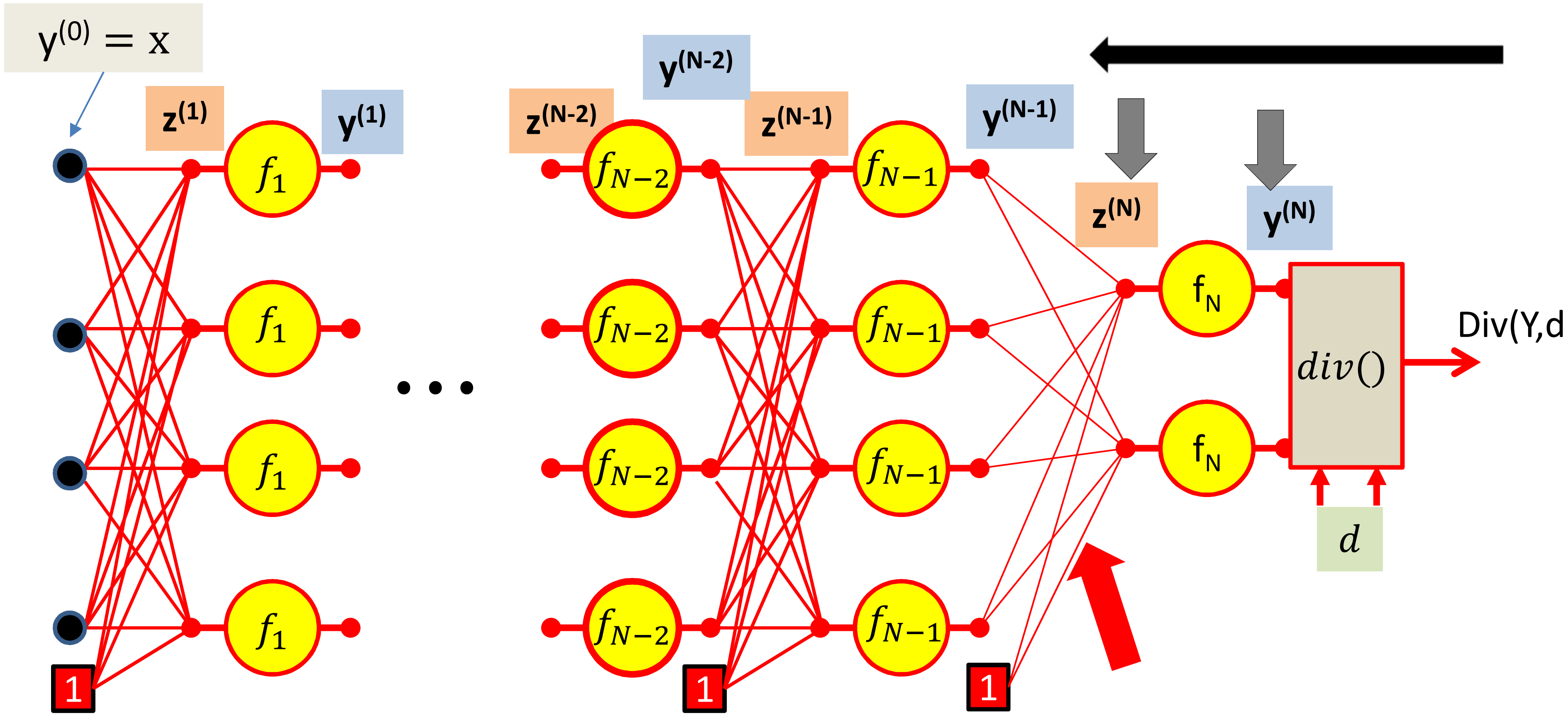
Computing derivatives



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

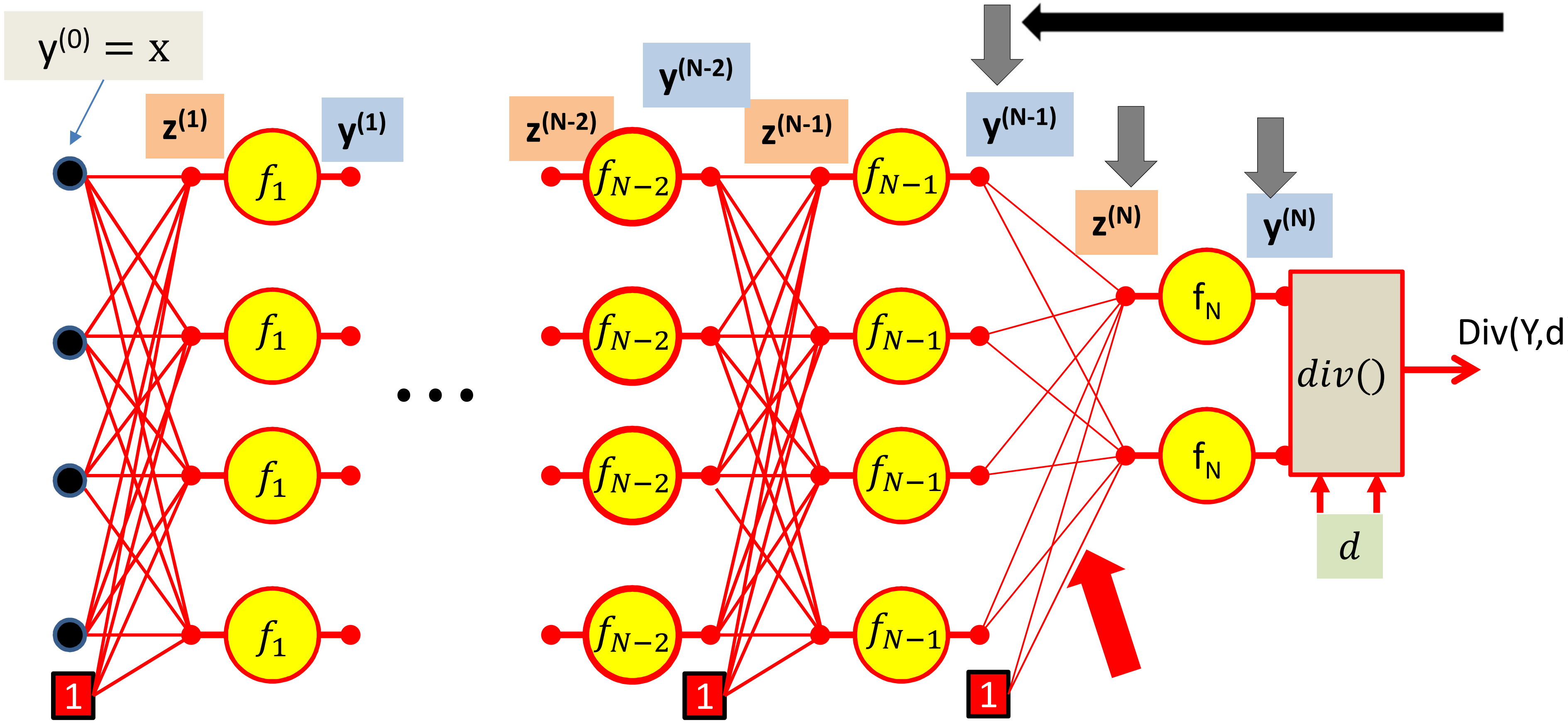
We then compute $\nabla_{z^{(N)}} div(.)$ the derivative of the divergence w.r.t. the *pre-activation* affine combination $z^{(N)}$ using the chain rule

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

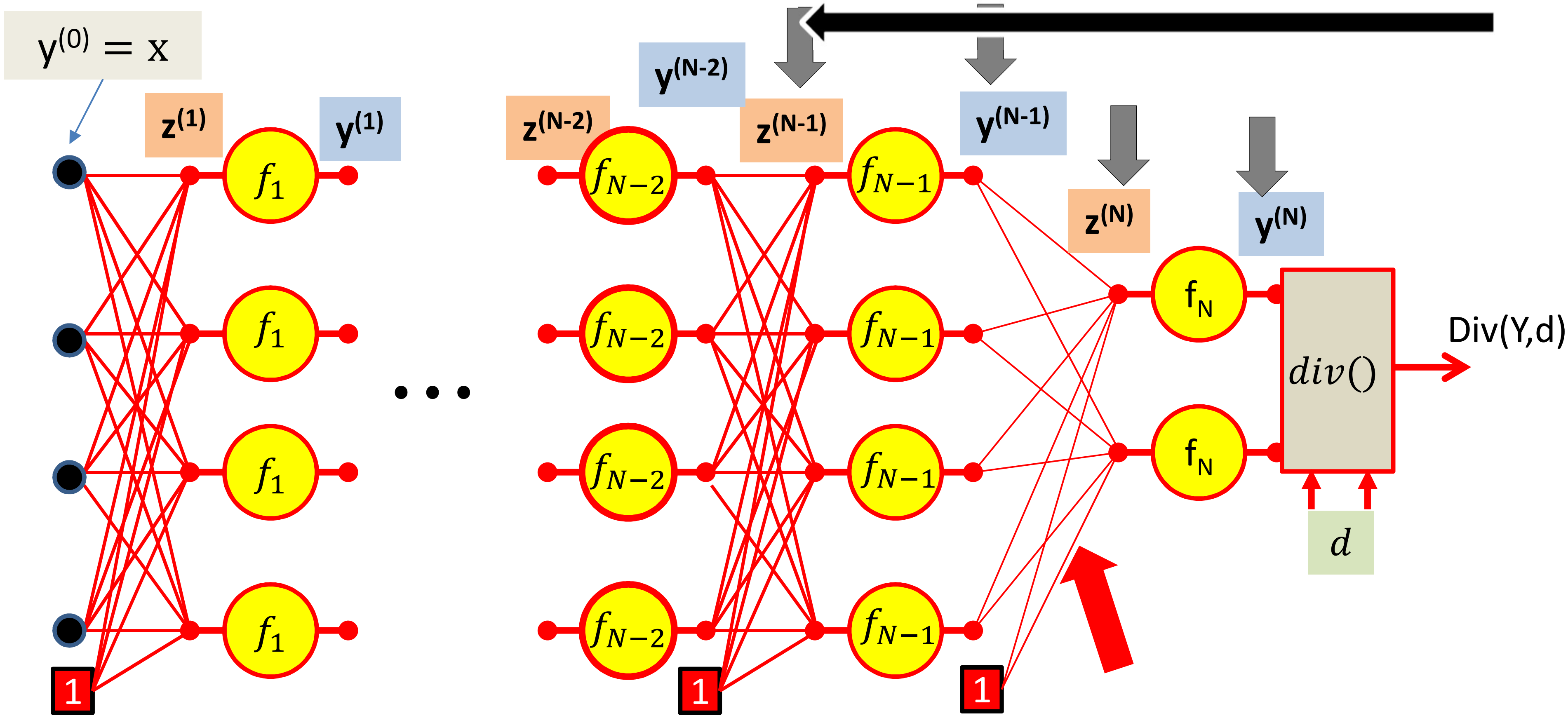
Computing derivatives



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

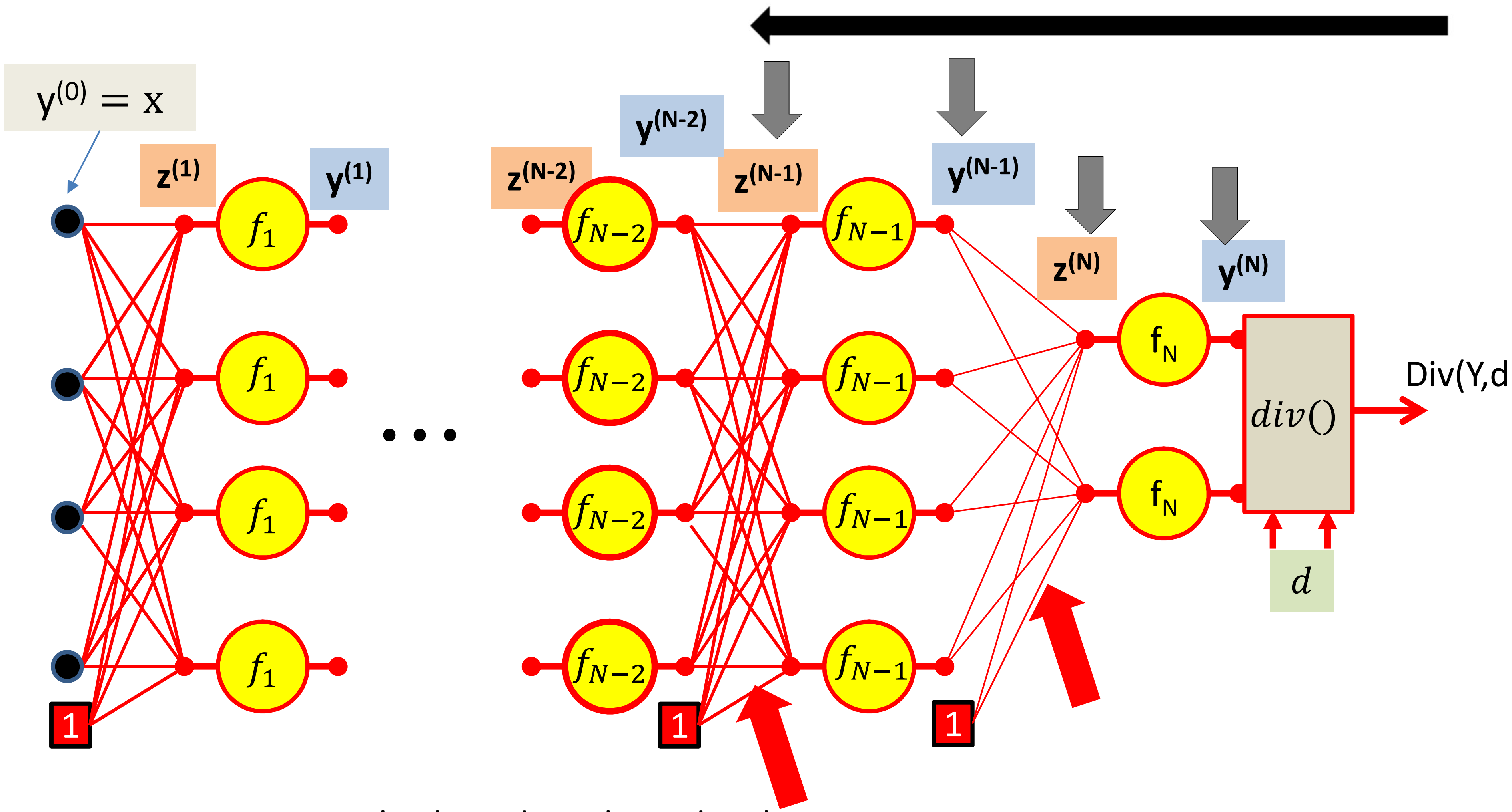
Then continue with the chain rule to compute $\nabla_{Y^{(N-1)}} div(.)$ the derivative of the divergence w.r.t. the output of the N-1th layer

Computing derivatives



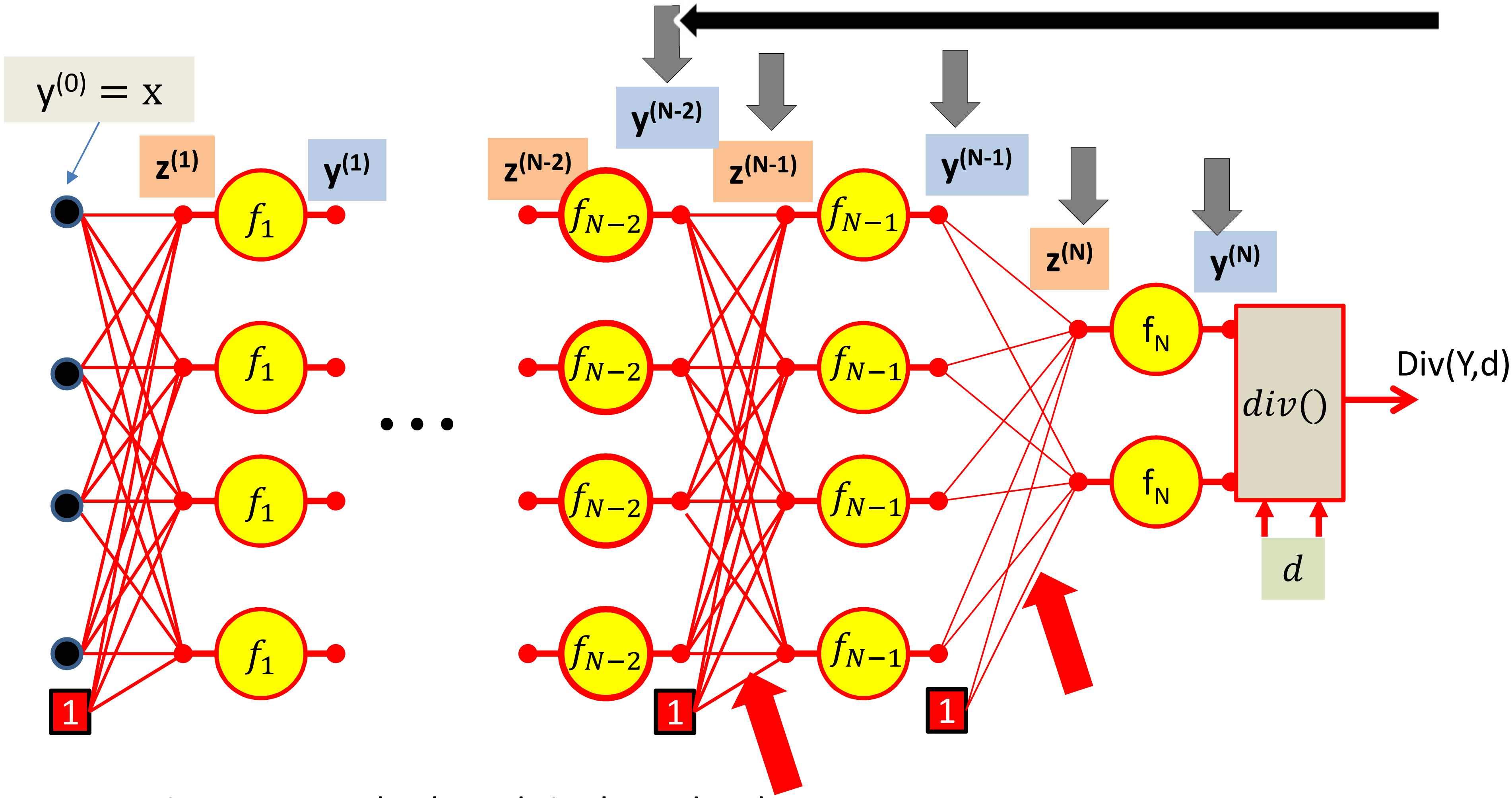
We continue our way backwards in the order shown

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



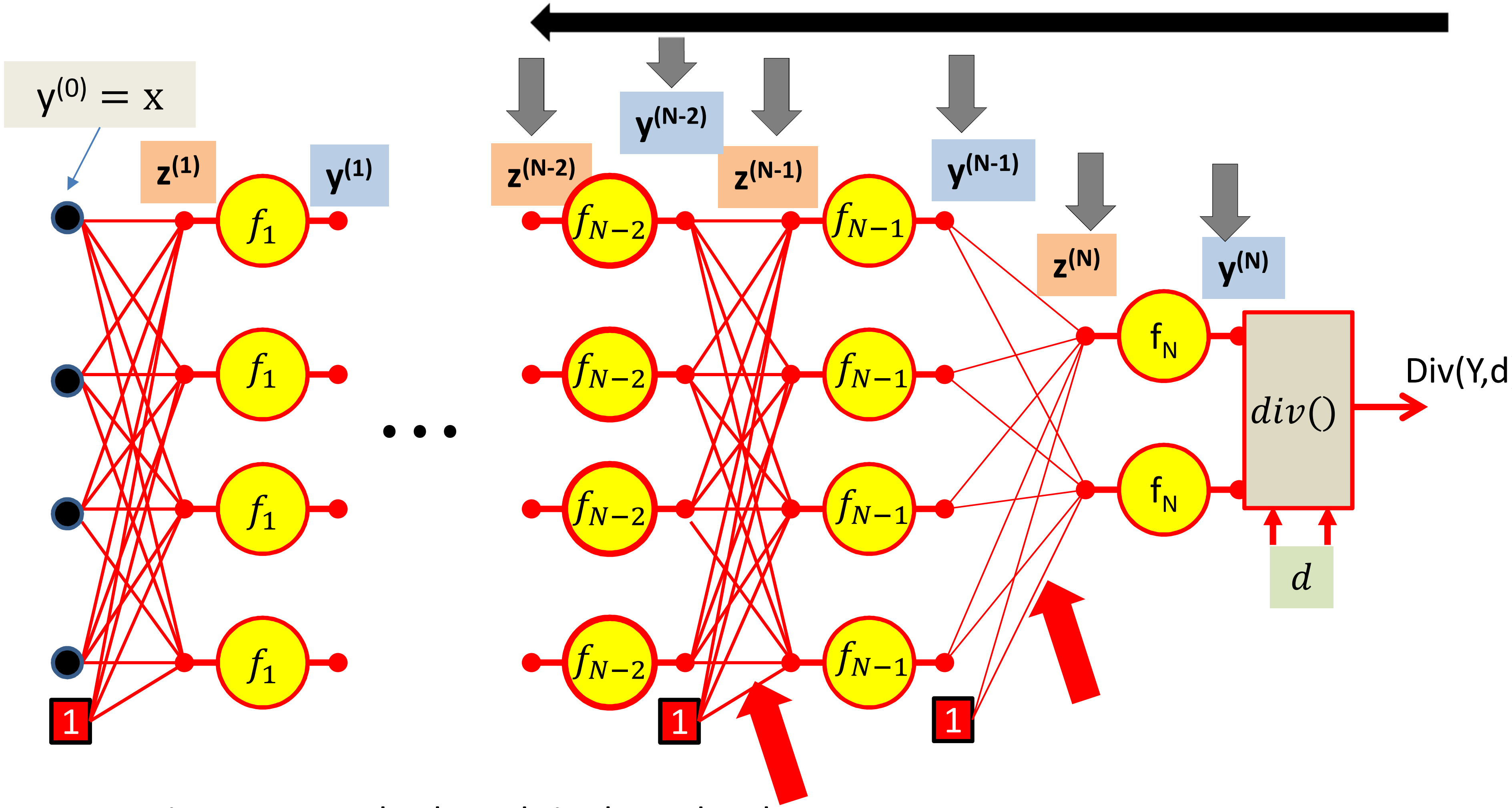
We continue our way backwards in the order shown

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



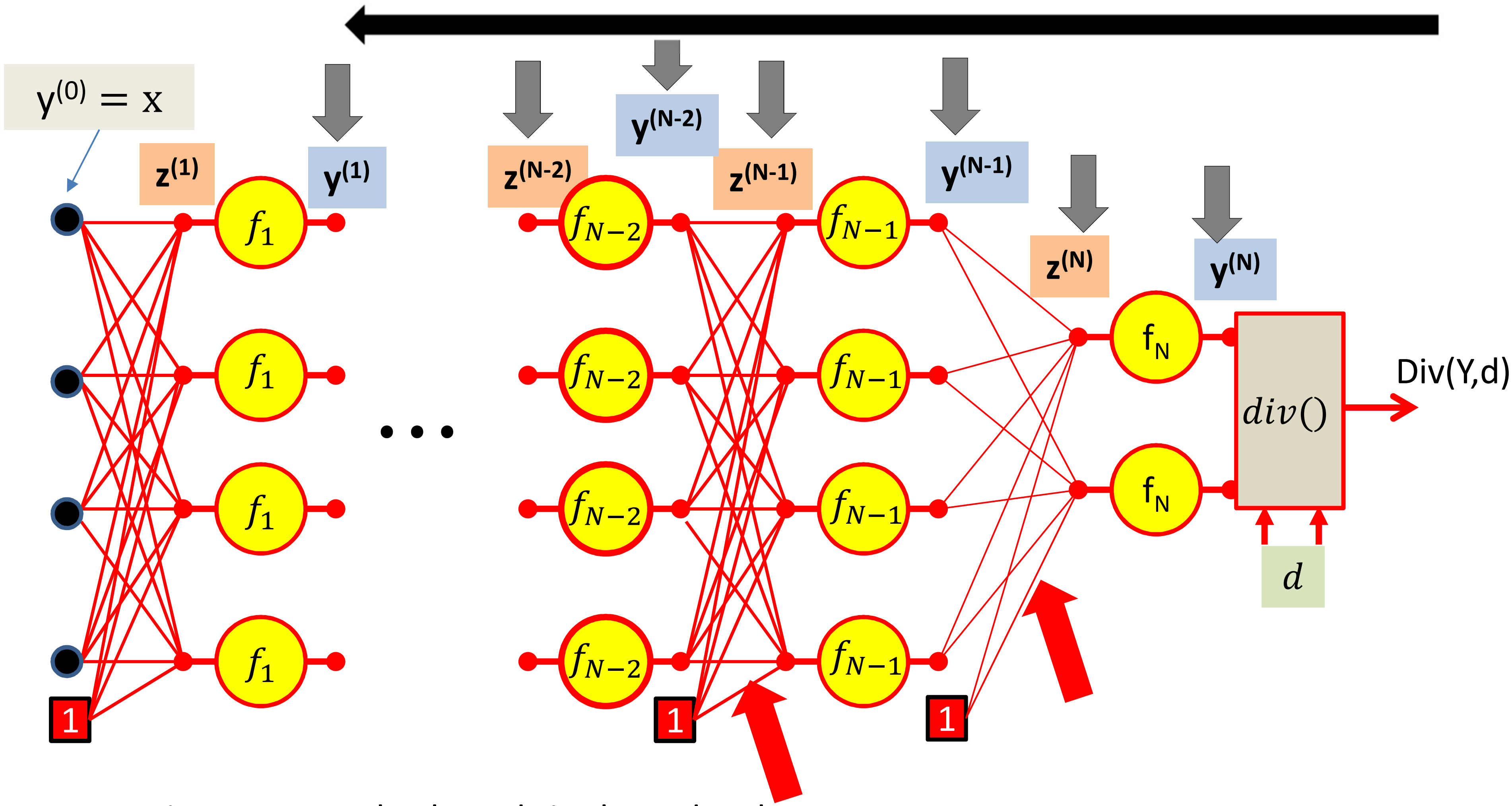
We continue our way backwards in the order shown

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



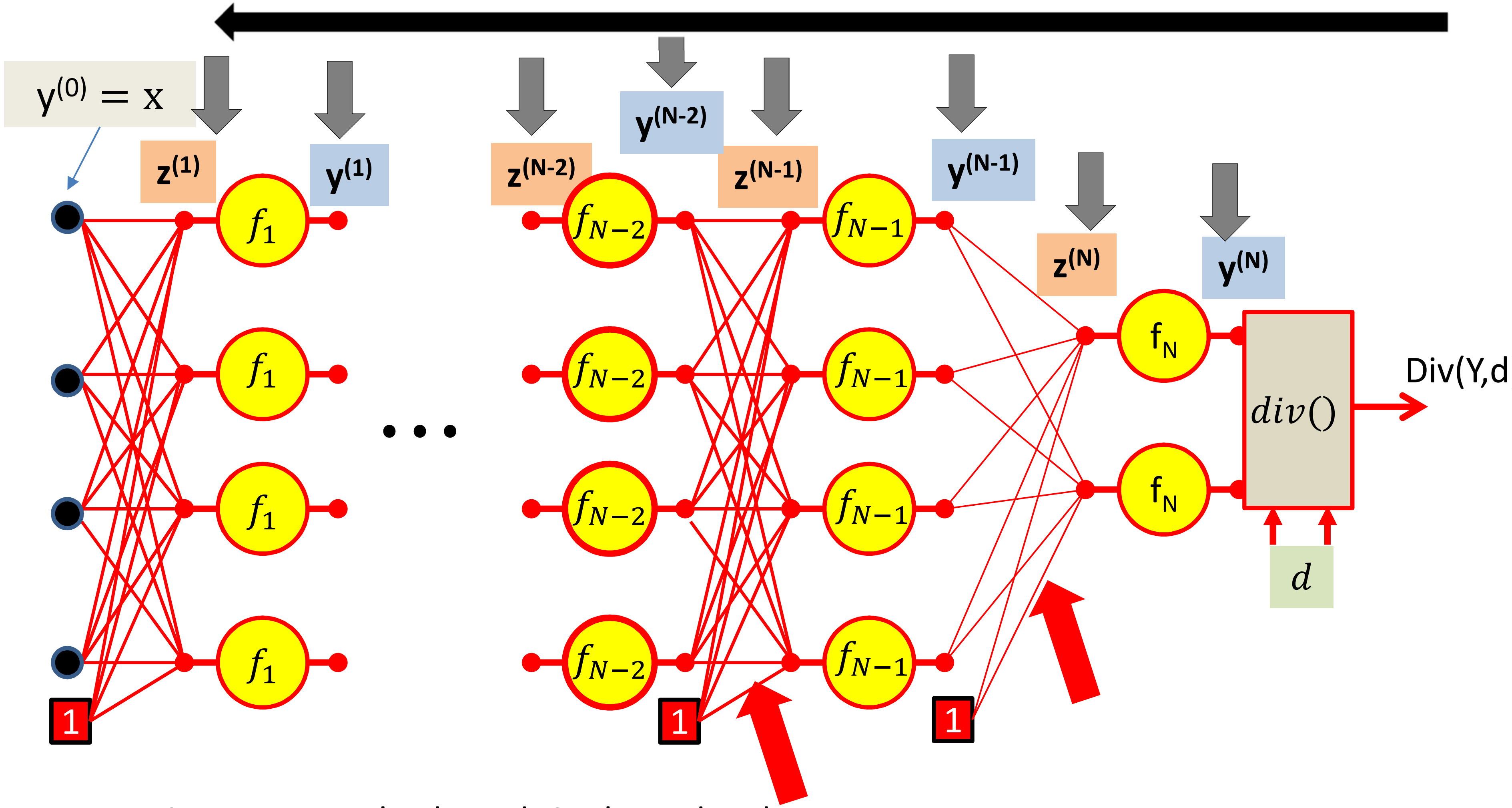
We continue our way backwards in the order shown

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



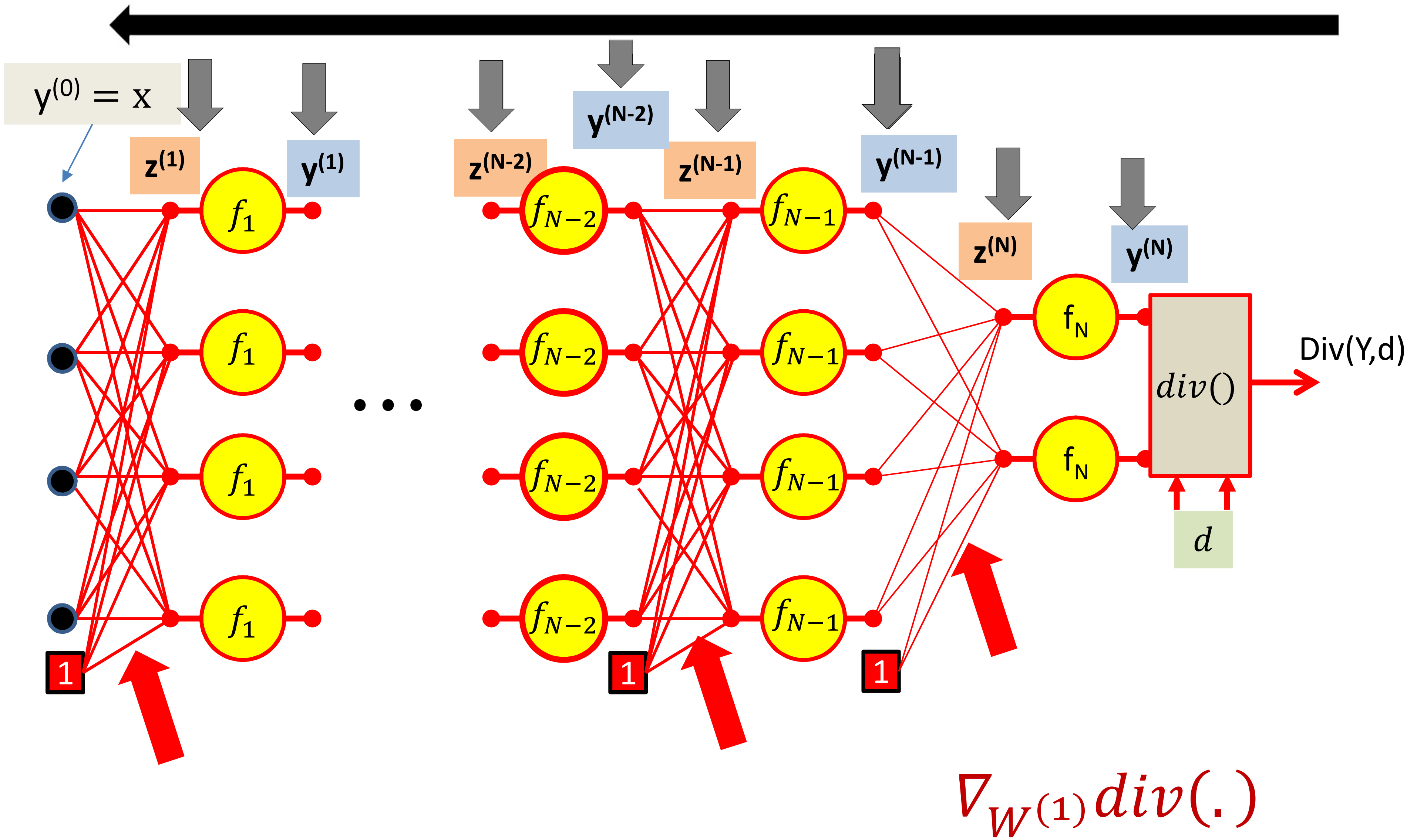
We continue our way backwards in the order shown

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



We continue our way backwards in the order shown

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

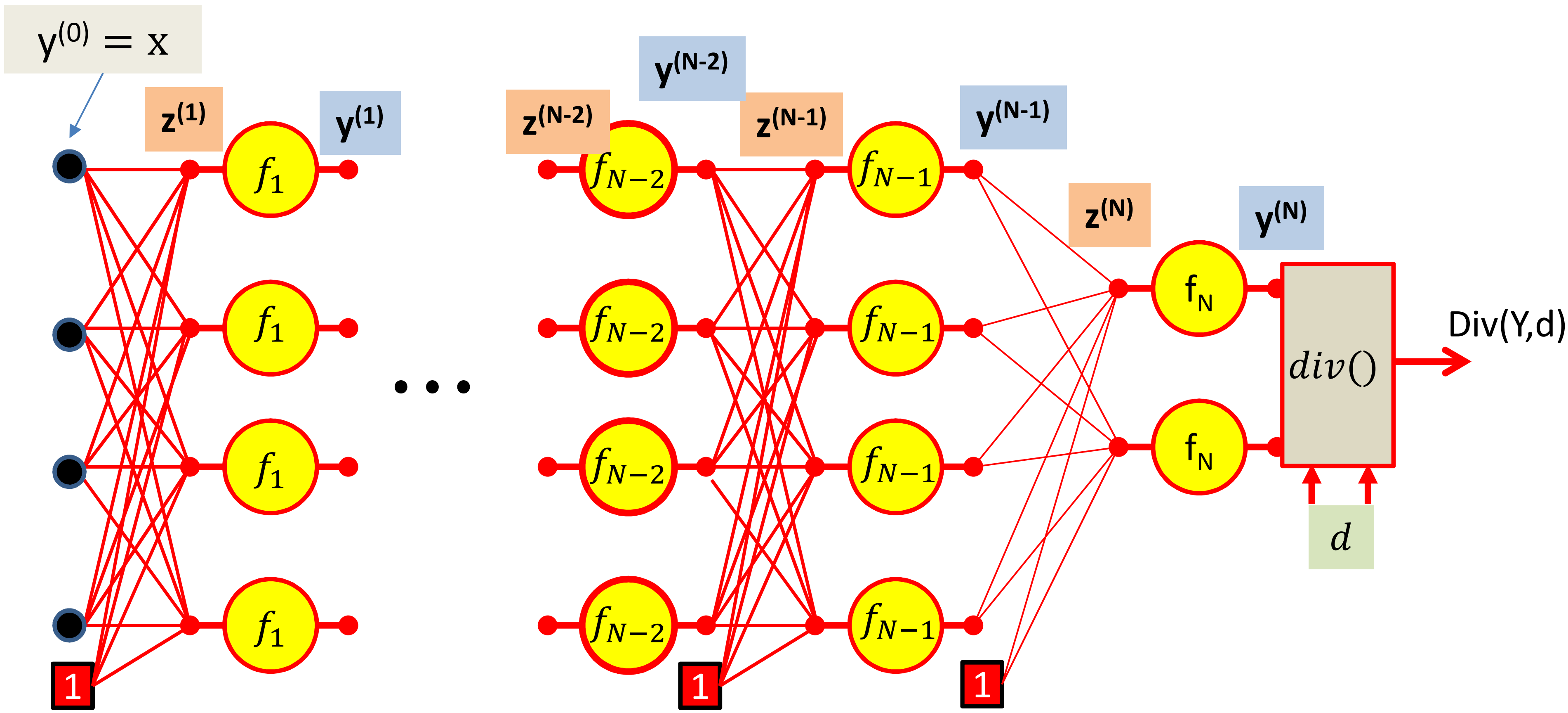


We continue our way backwards in the order shown

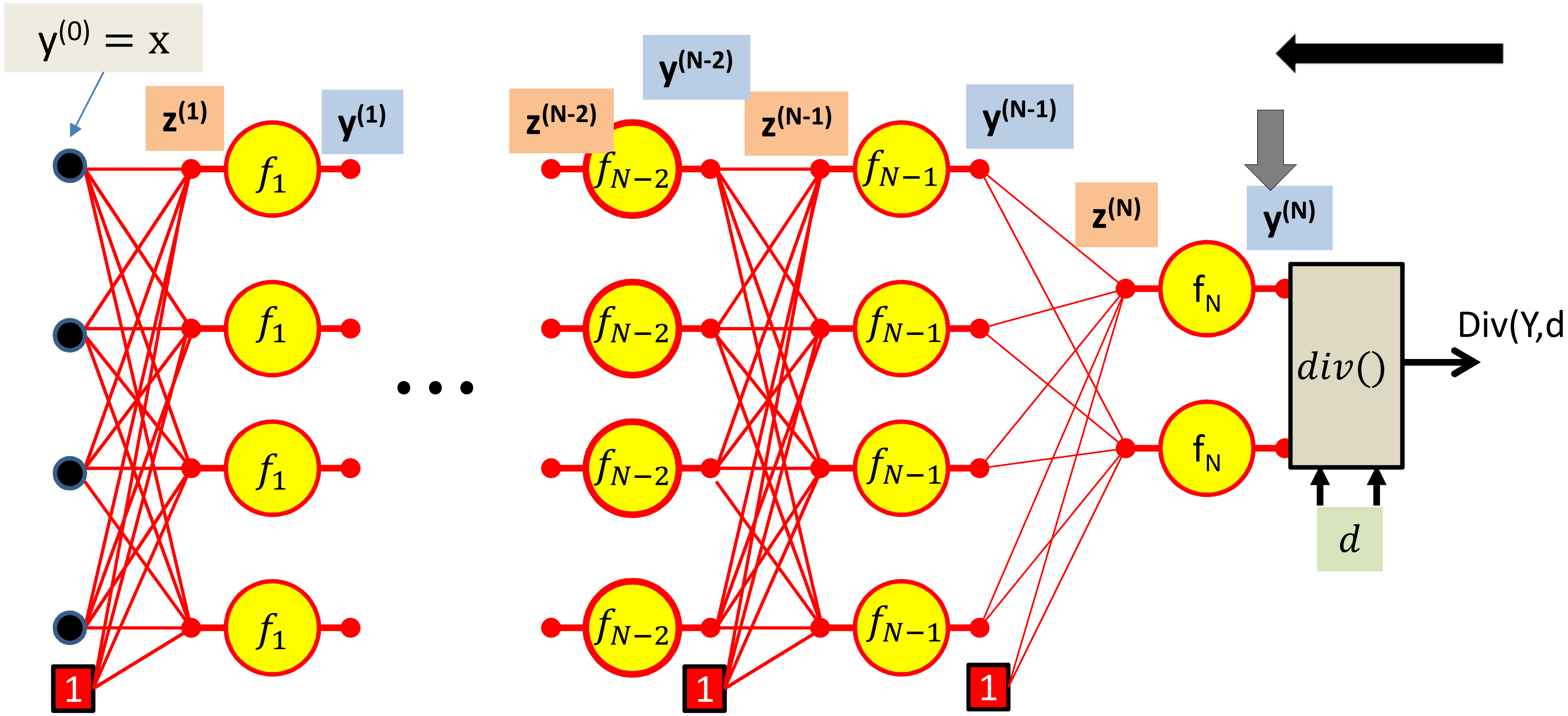
Backward Gradient Computation

- Lets actually see the math..

Computing derivatives



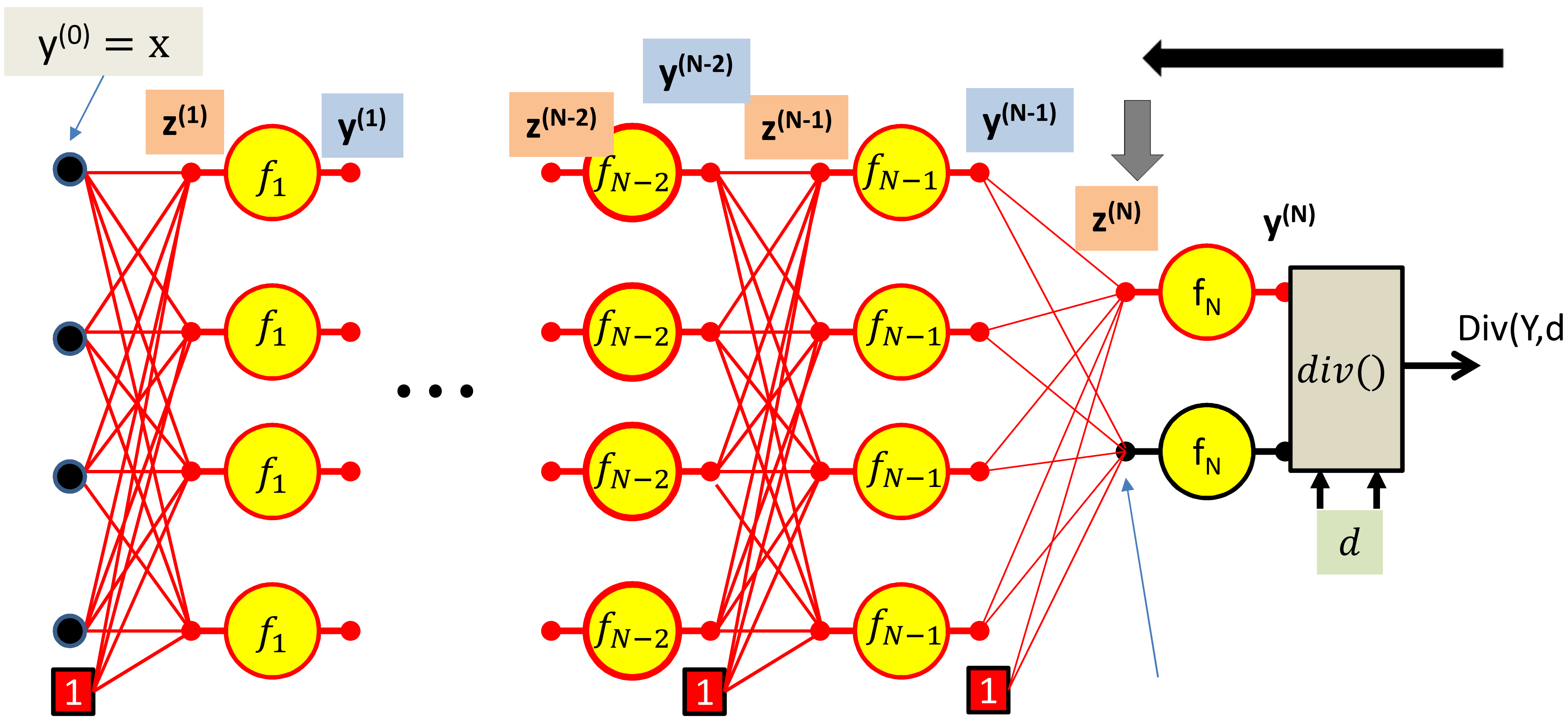
Computing derivatives



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

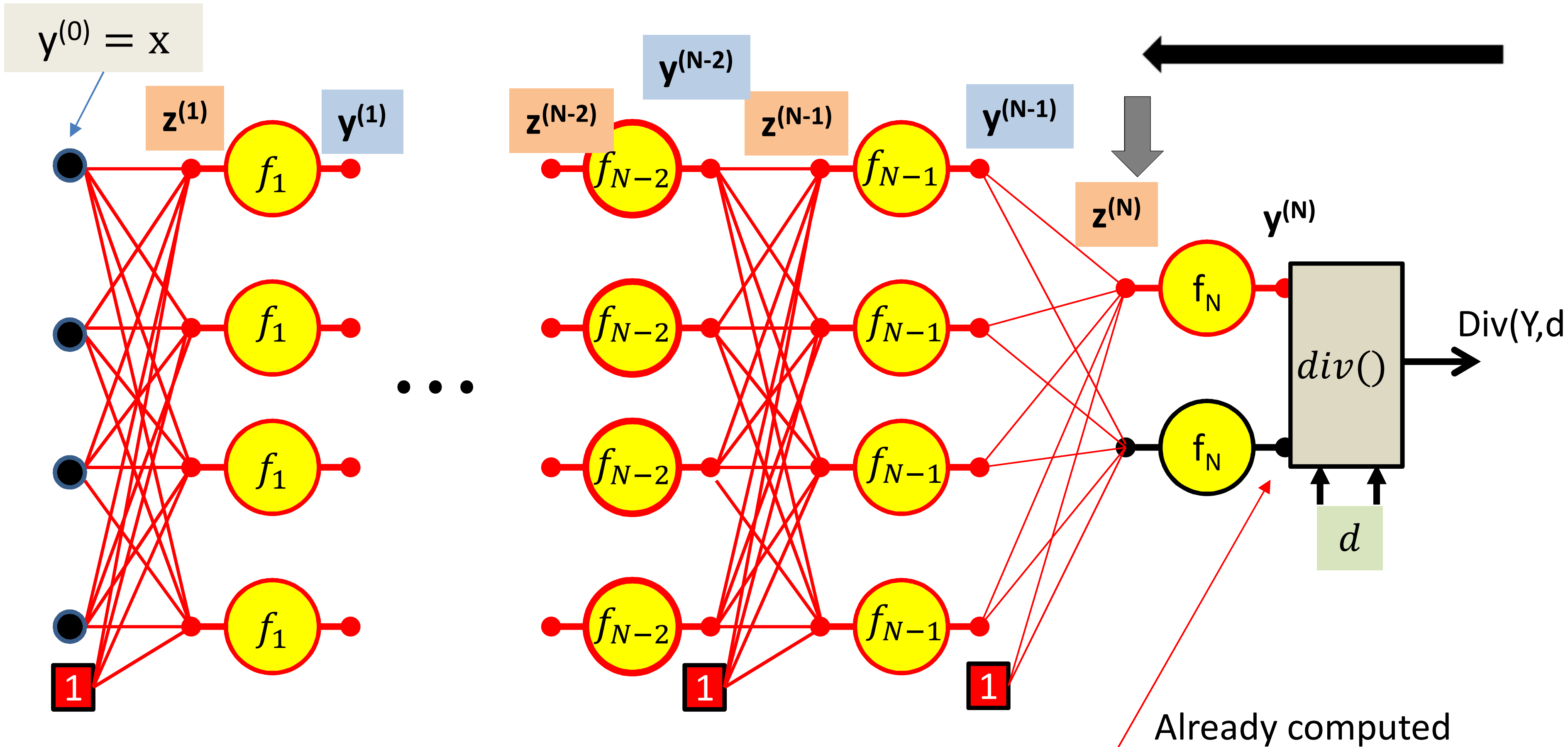
$$\frac{\partial Div(Y,d)}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}}$$

Computing derivatives



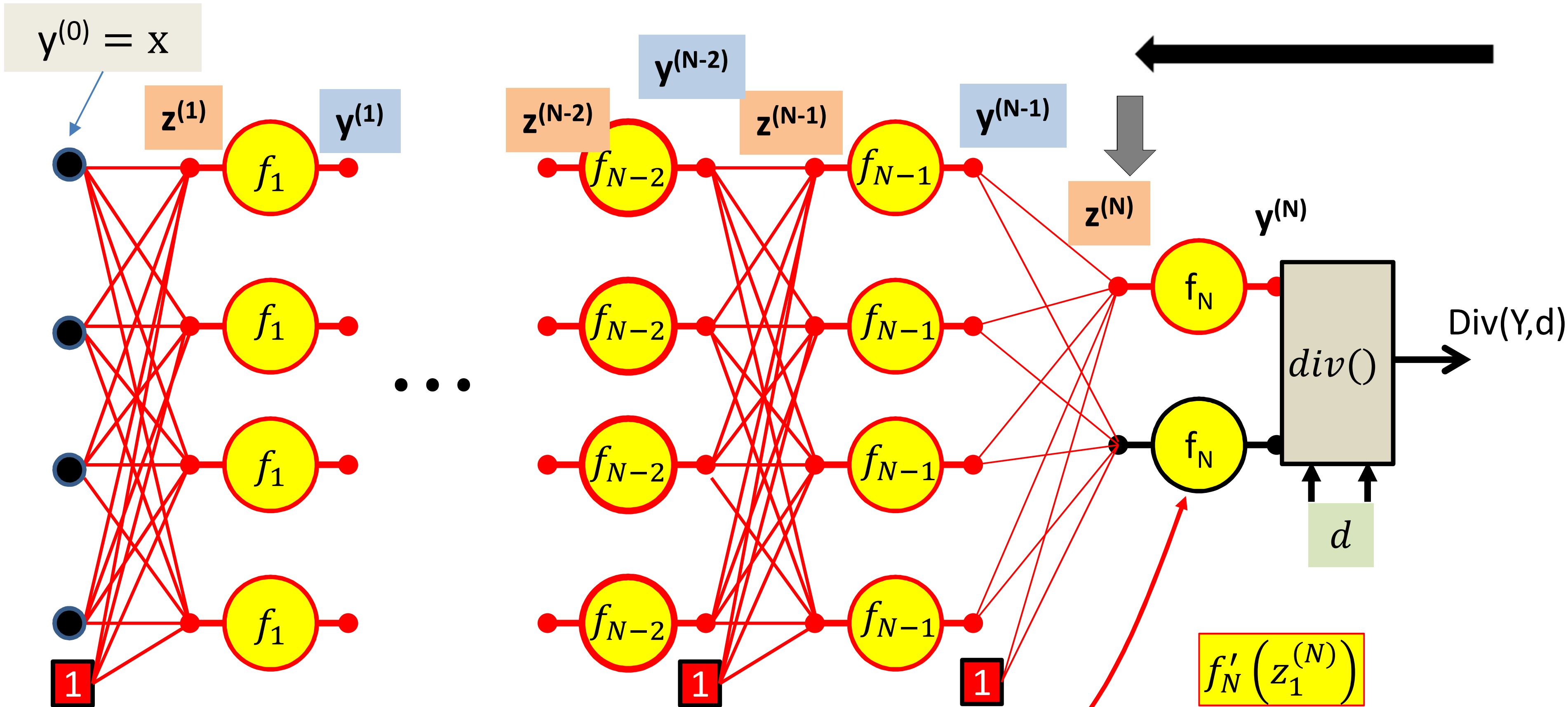
$$\frac{\partial Div}{\partial z_1^{(N)}} = \frac{\partial y_1^{(N)}}{\partial z_1^{(N)}} \frac{\partial Div}{\partial y_1^{(N)}}$$

Computing derivatives



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

Computing derivatives

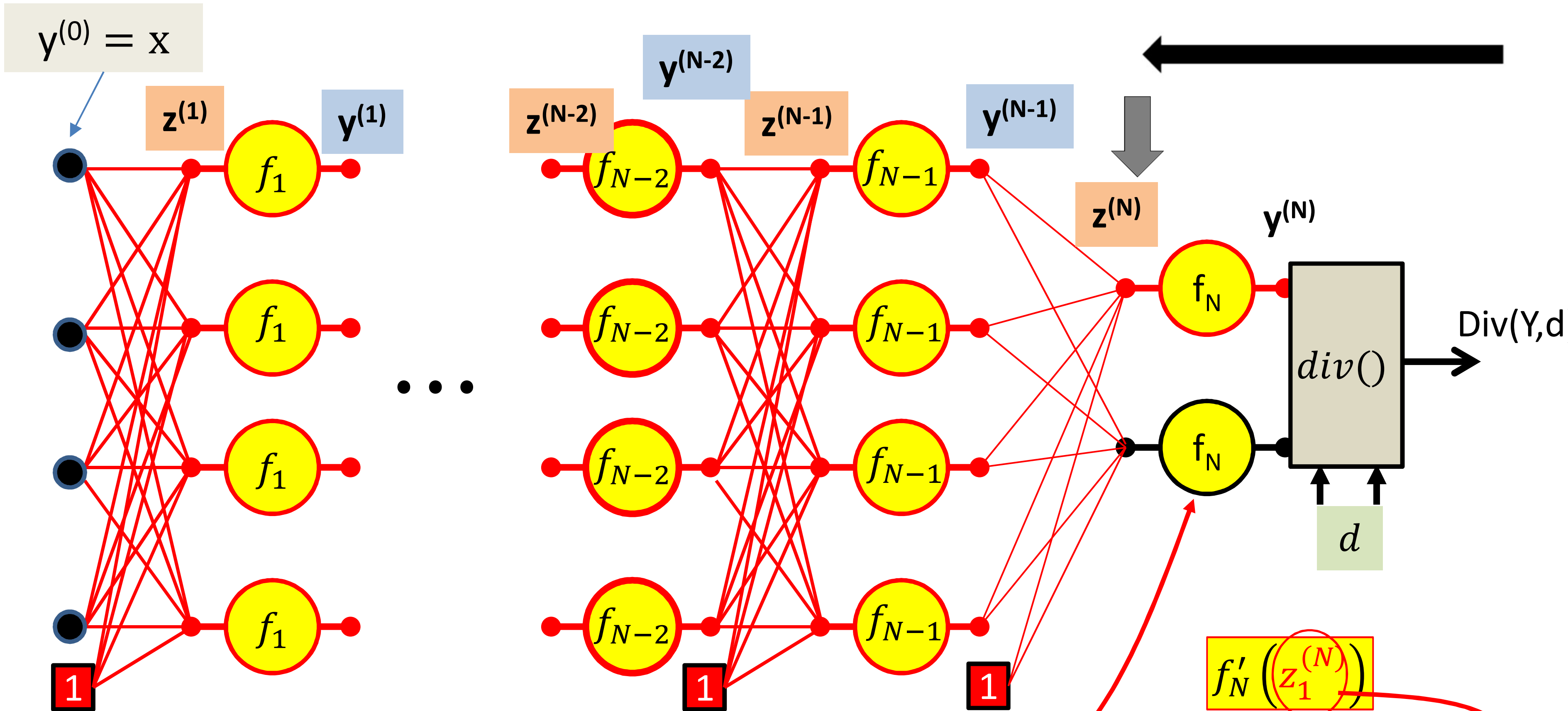


$$f_N'\left(z_1^{(N)}\right)$$

Derivative of activation function

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

Computing derivatives



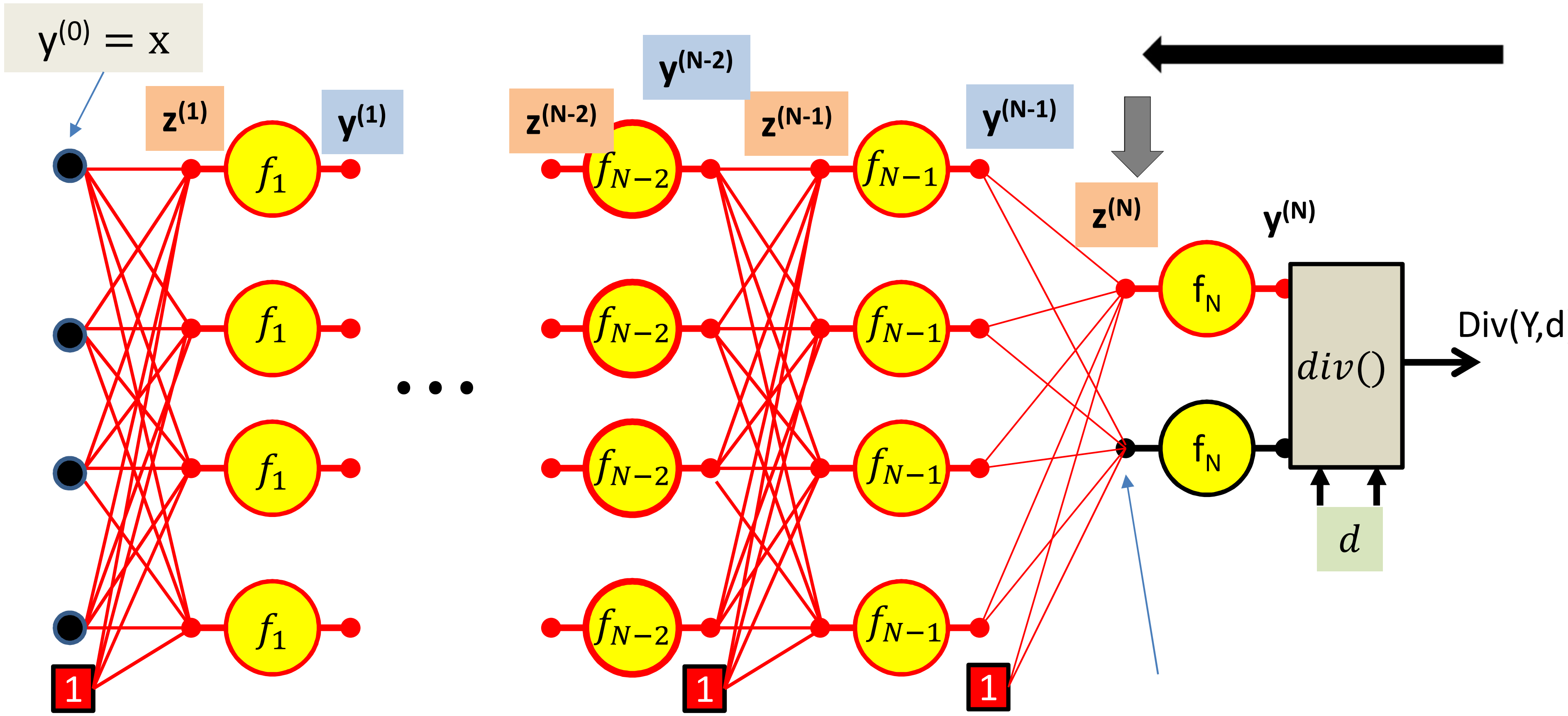
$$\frac{\partial Div}{\partial z_1^{(N)}} = \frac{\partial y_1^{(N)}}{\partial z_1^{(N)}} \frac{\partial Div}{\partial y_1^{(N)}}$$

$f'_N\left(z_1^{(N)}\right)$

Derivative of activation function

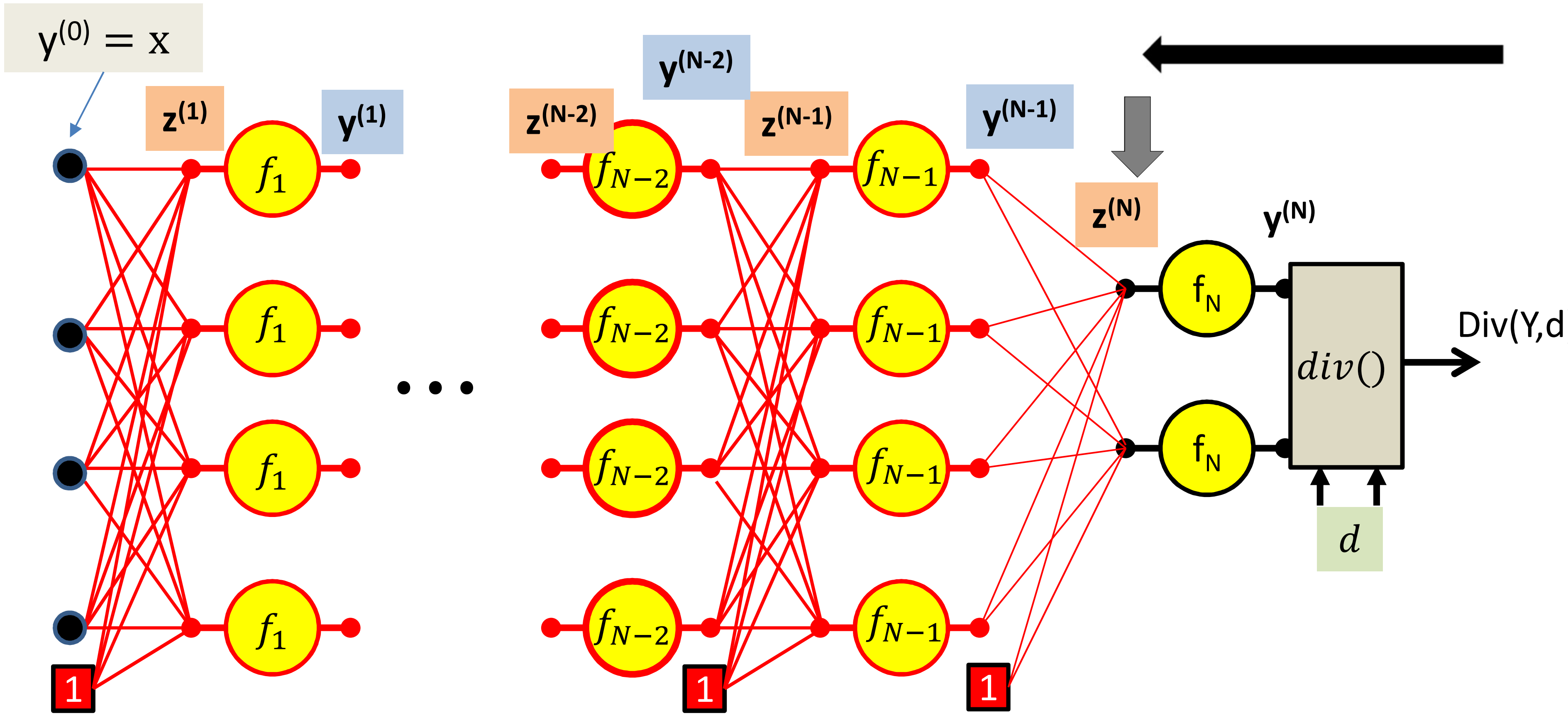
Computed in forward pass

Computing derivatives



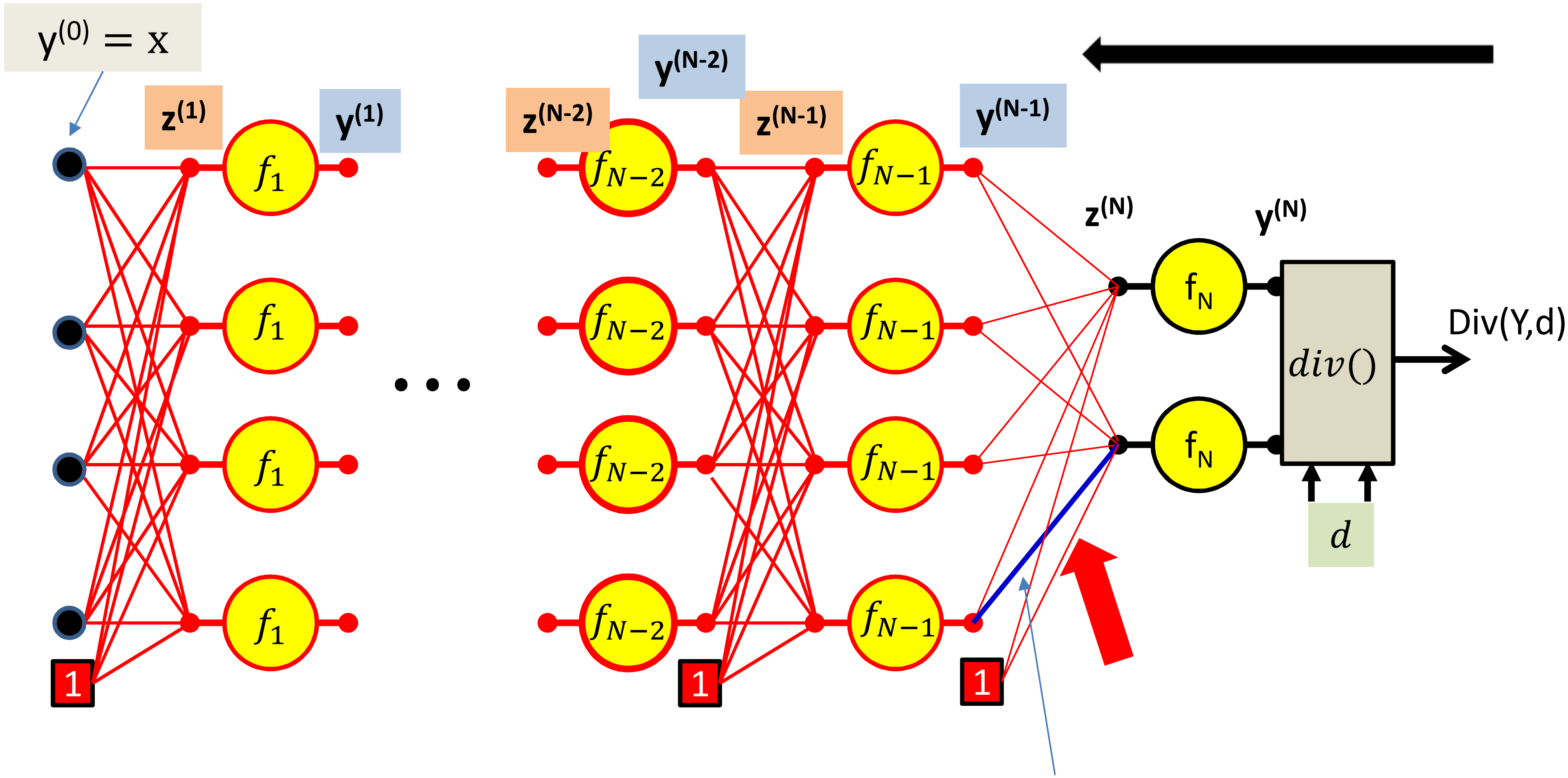
$$\frac{\partial Div}{\partial z_1^{(N)}} = f_N'\left(z_1^{(N)}\right)\frac{\partial Div}{\partial y_1^{(N)}}$$

Computing derivatives



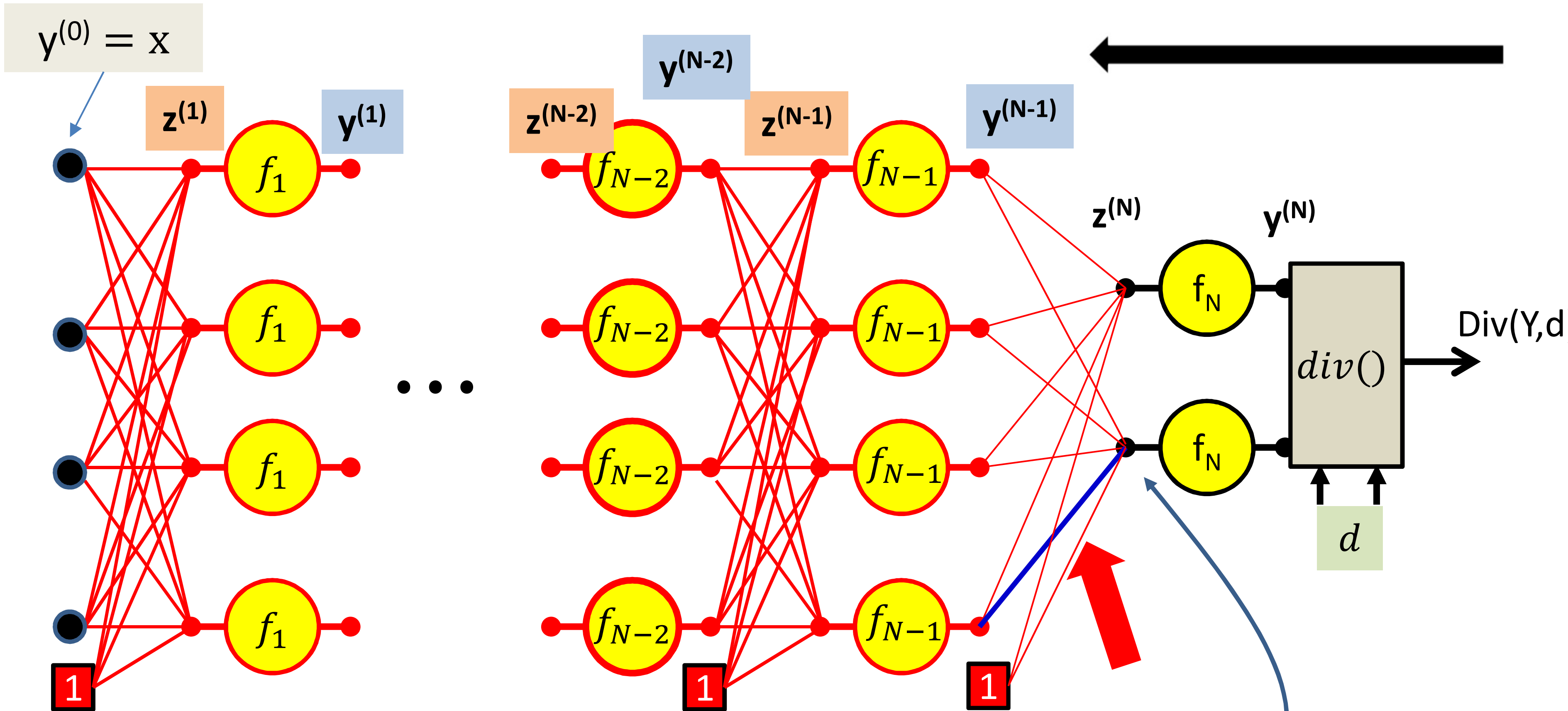
$$\frac{\partial Div}{\partial z_i^{(N)}} = f_N'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



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

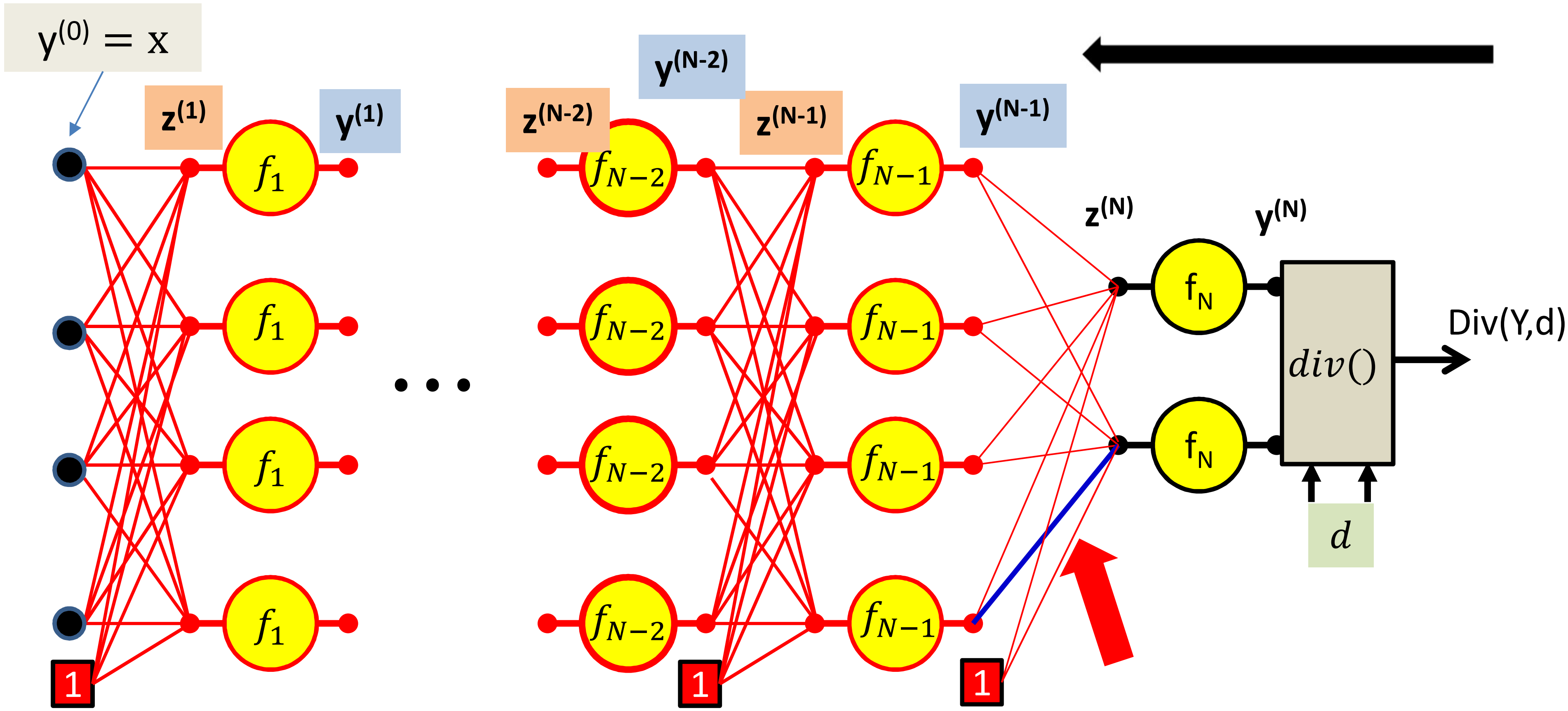
Computing derivatives



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

Just computed

Computing derivatives



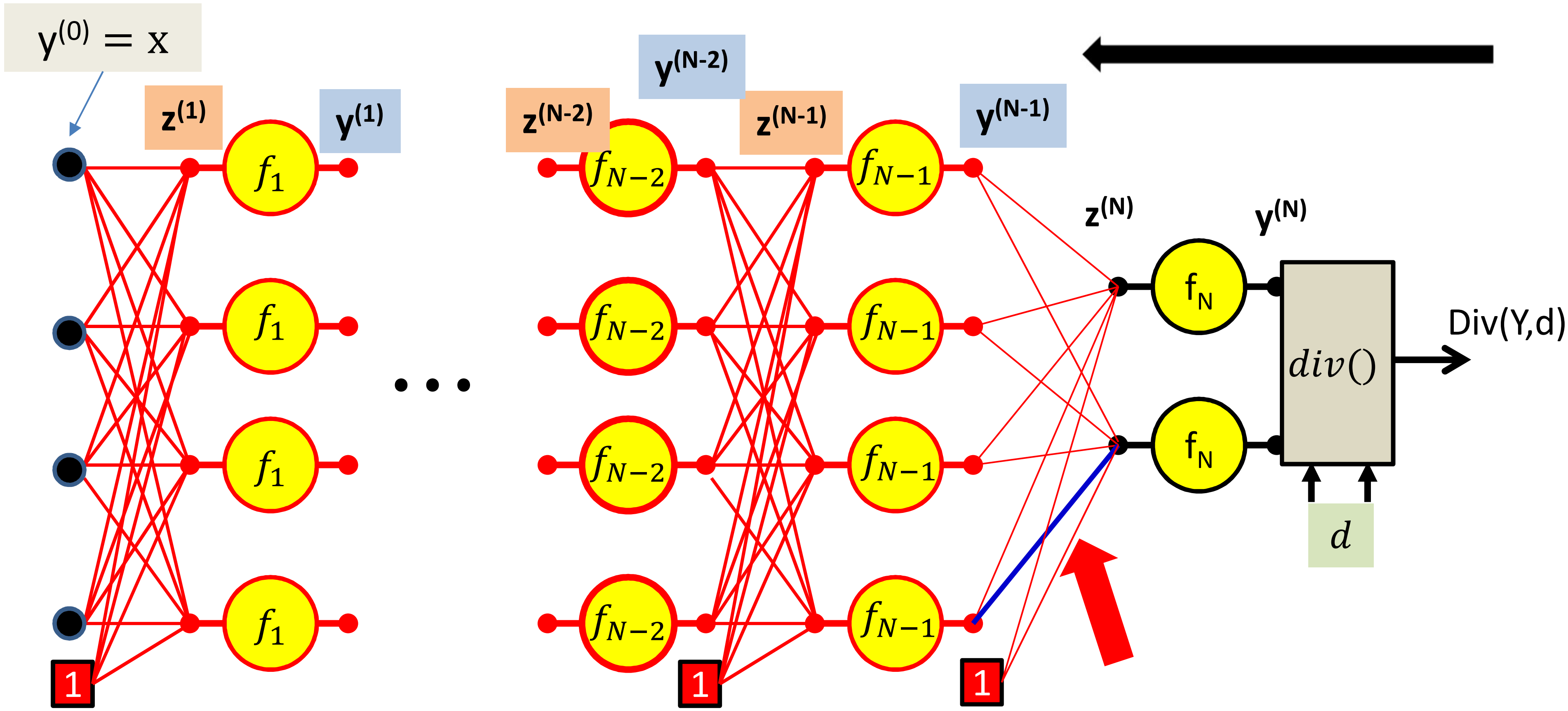
$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}}$$

$y_1^{(N-1)}$

Because

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

Computing derivatives



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

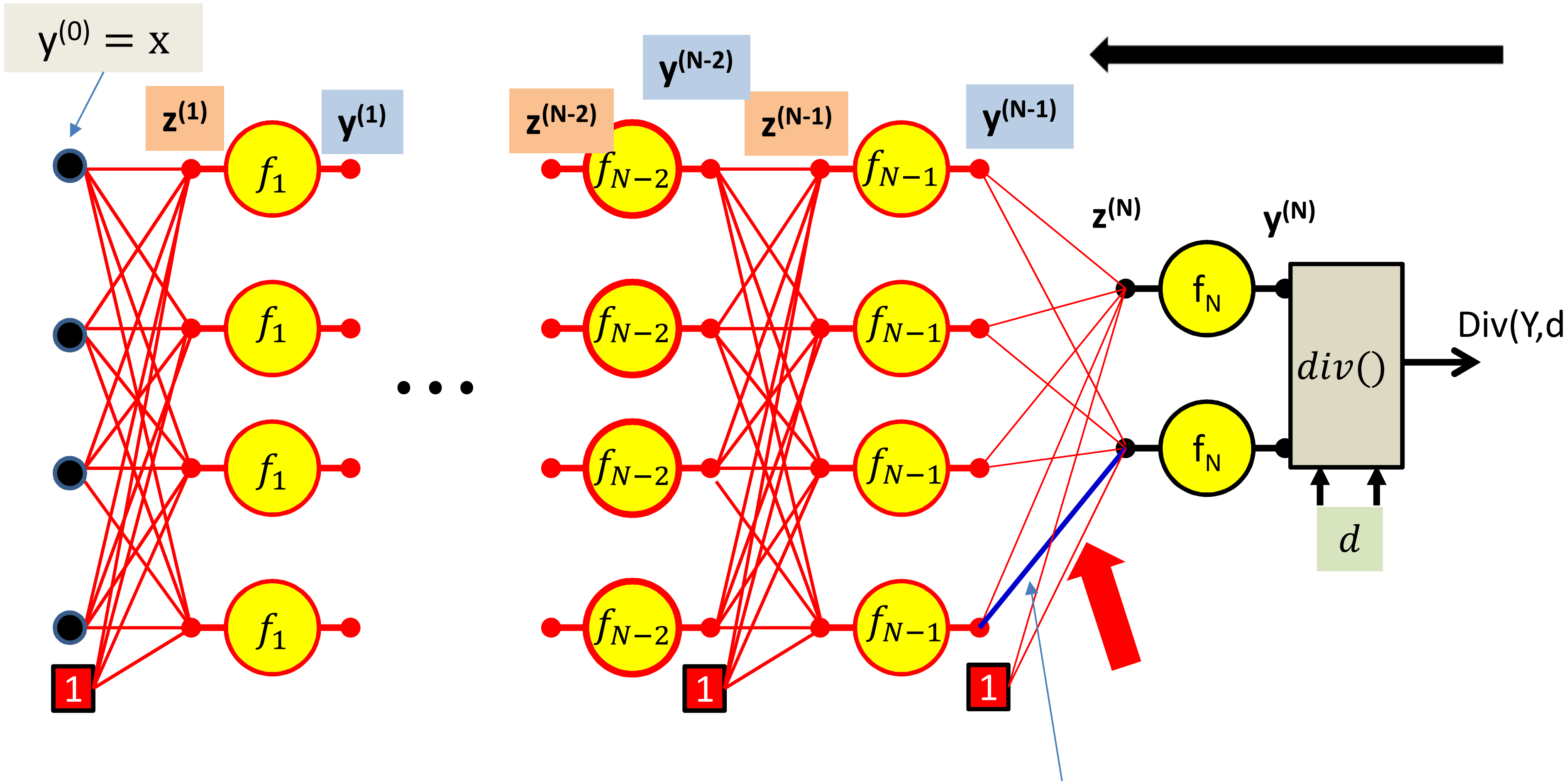
$y_1^{(N-1)}$

Because

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

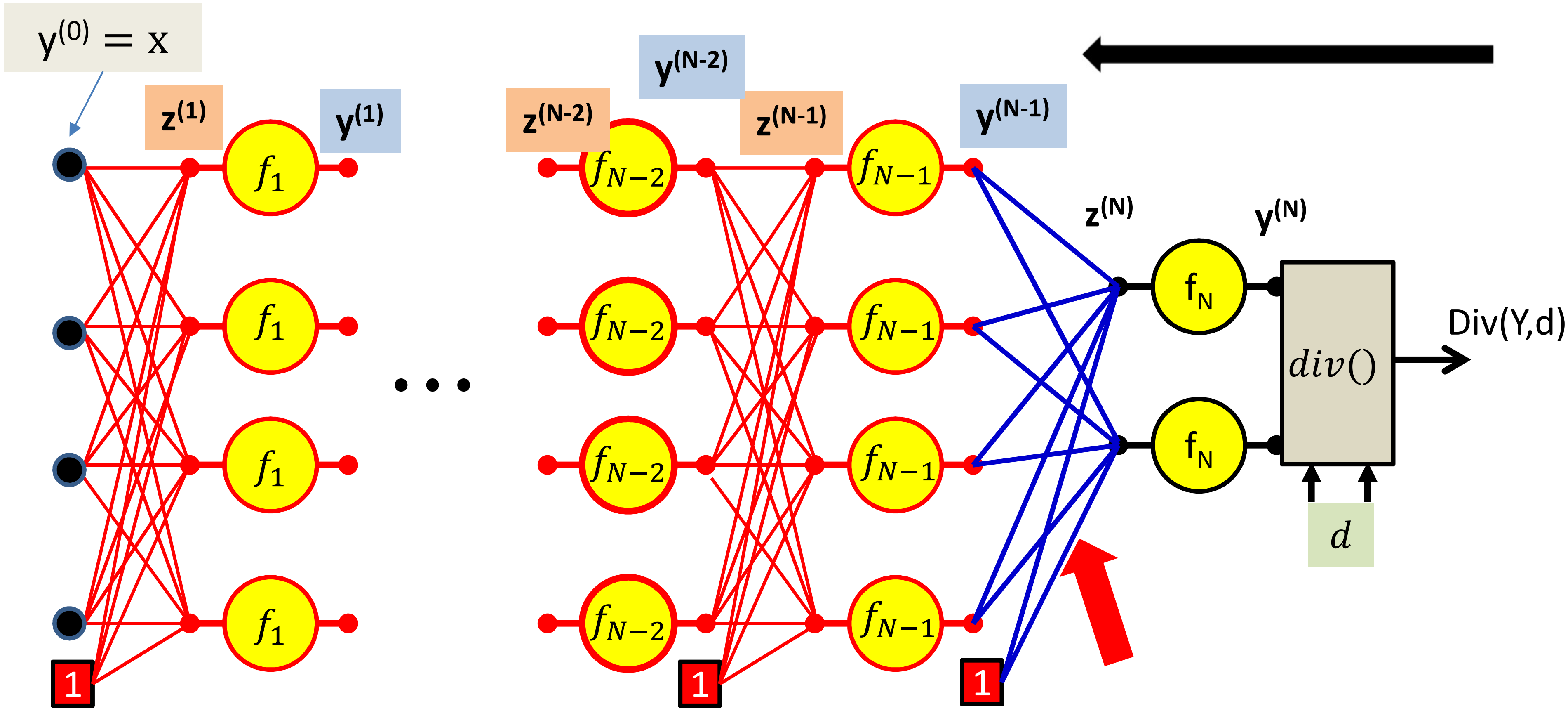
Computed in forward pass

Computing derivatives



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

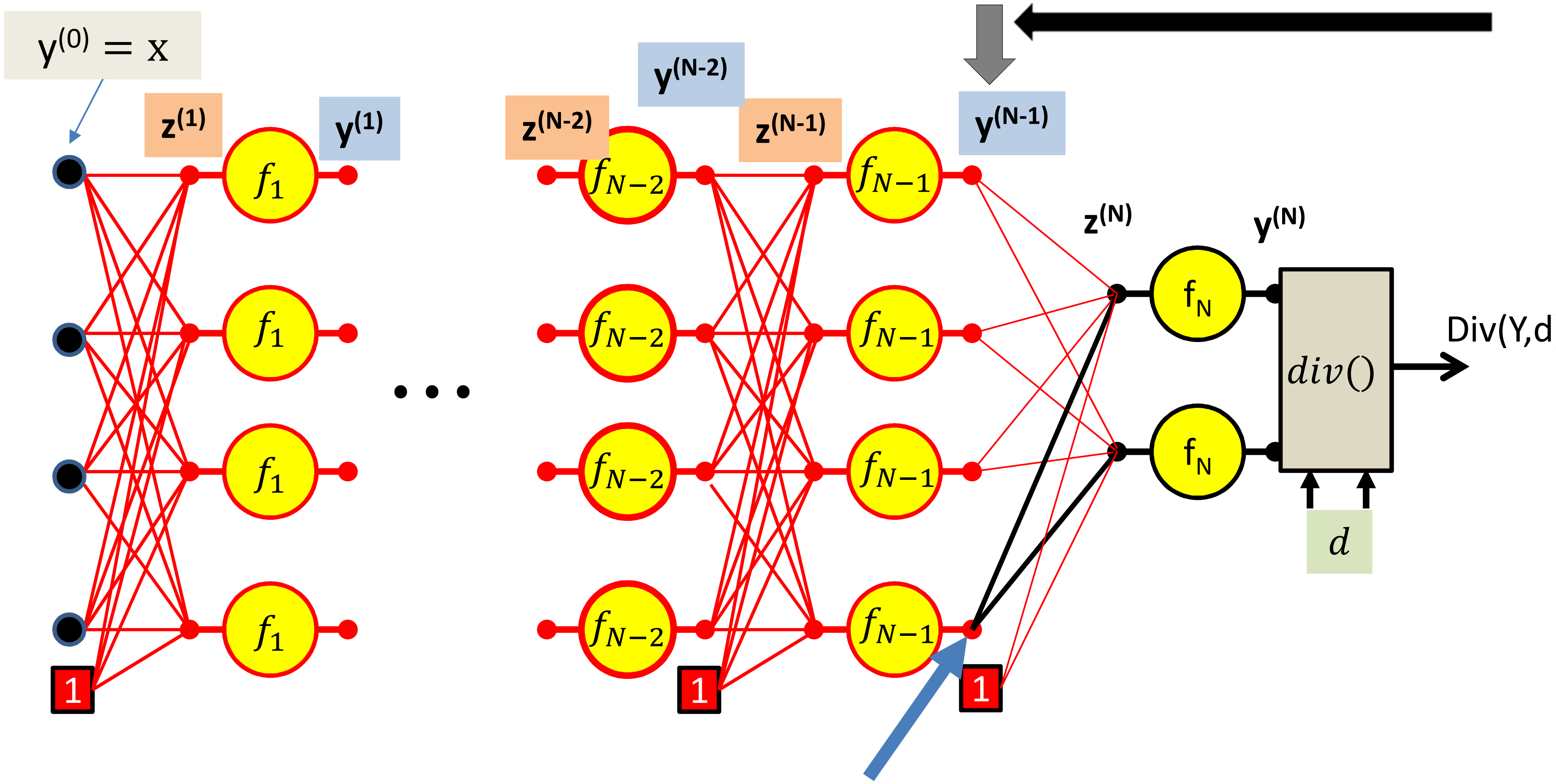
Computing derivatives



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

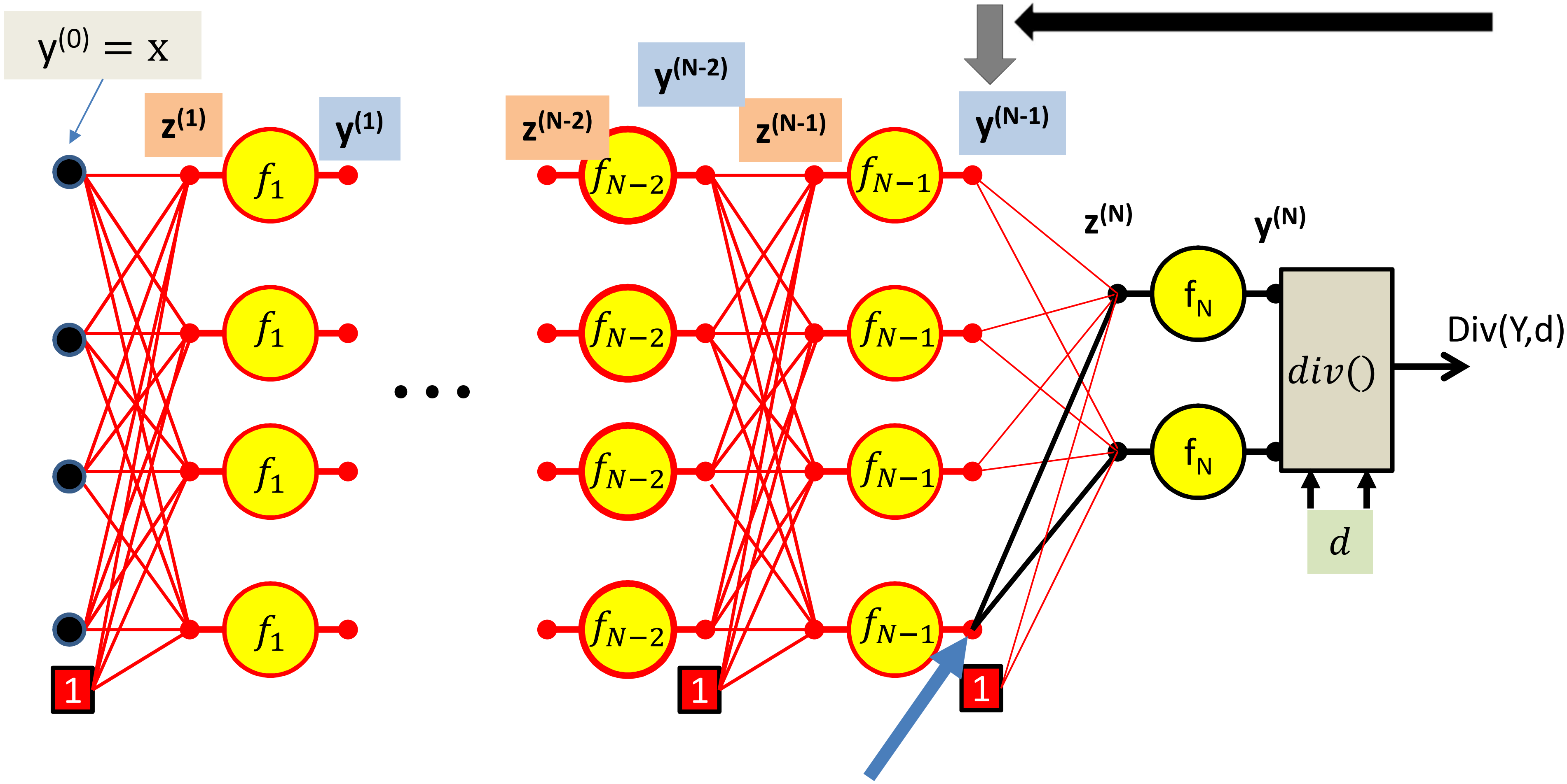
For the bias term $y_0^{(N-1)} = 1$

Computing derivatives



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

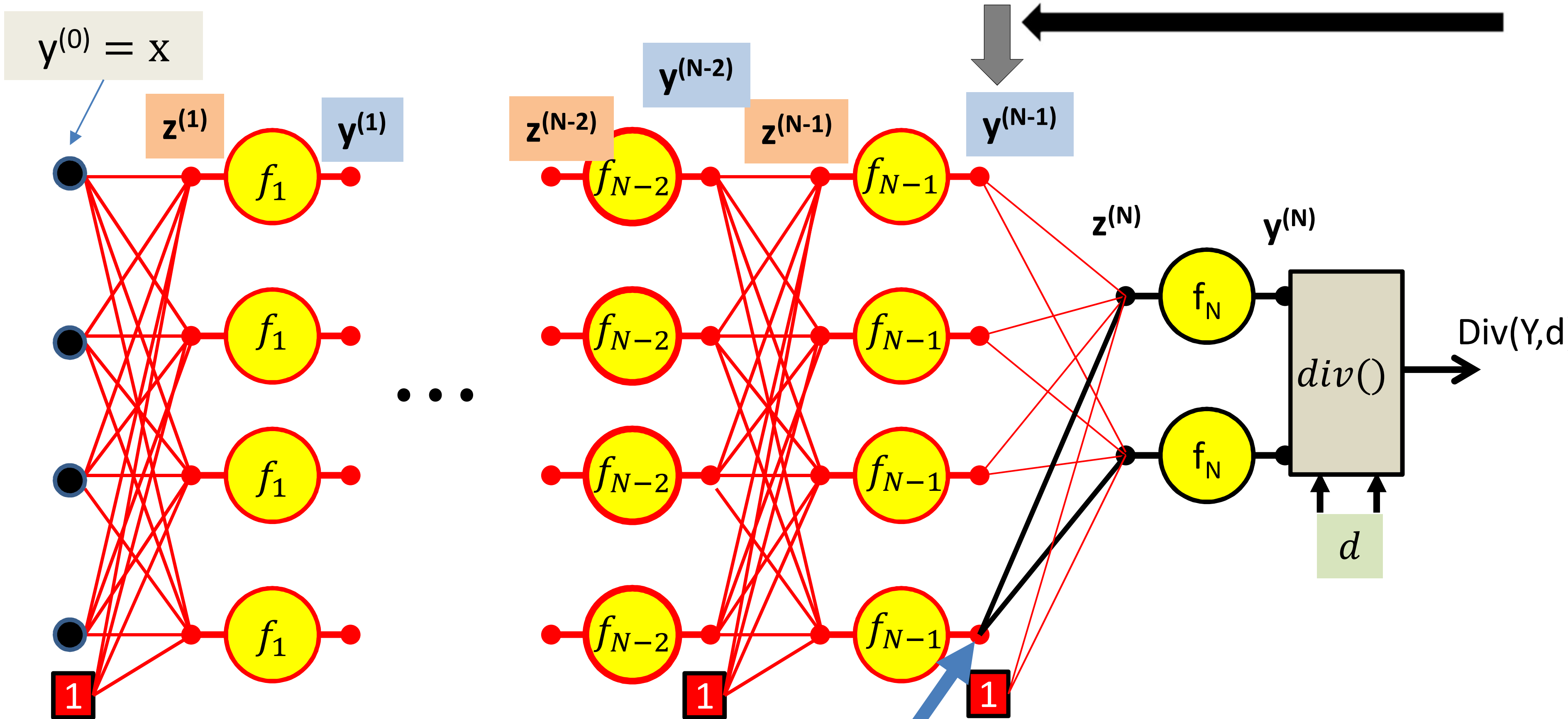
Computing derivatives



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

Already computed

Computing derivatives



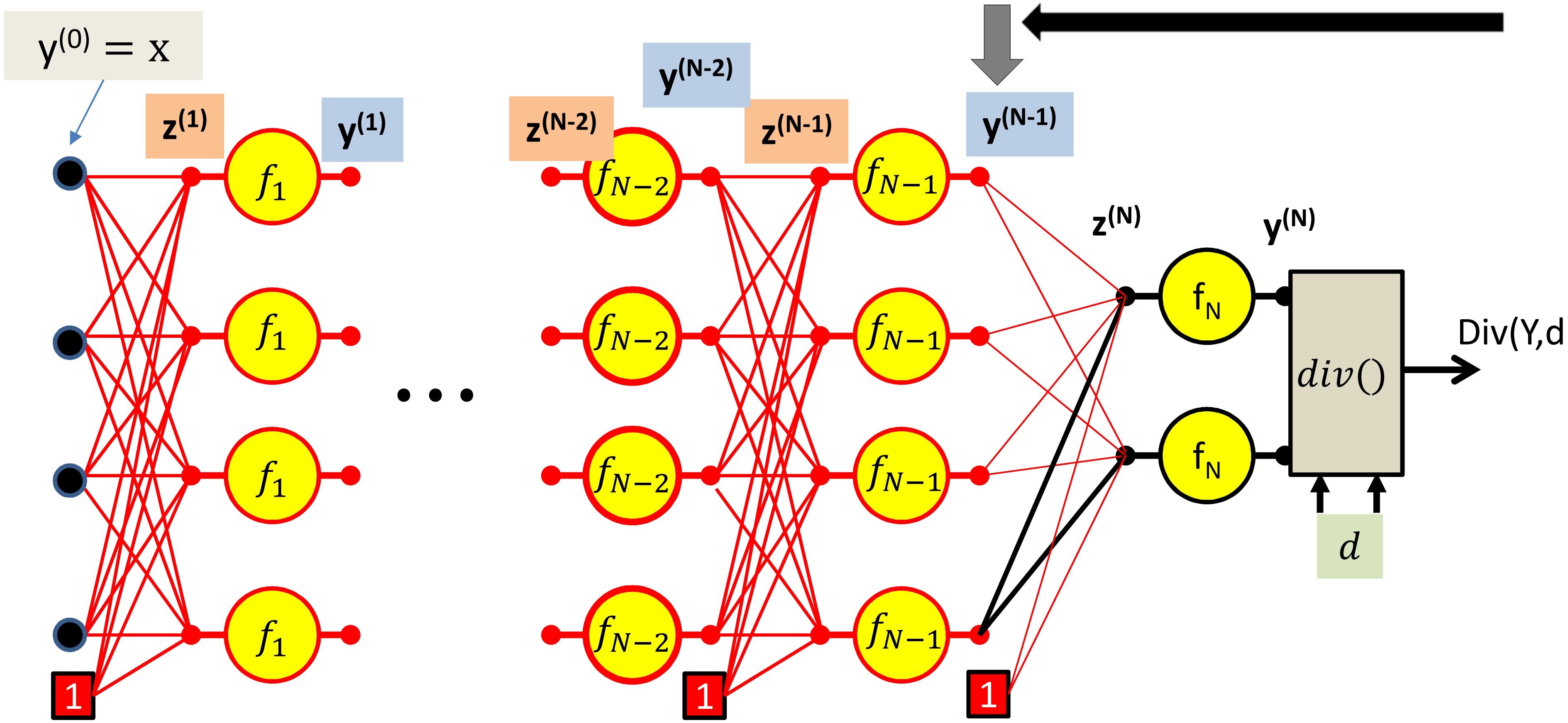
$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

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

Because

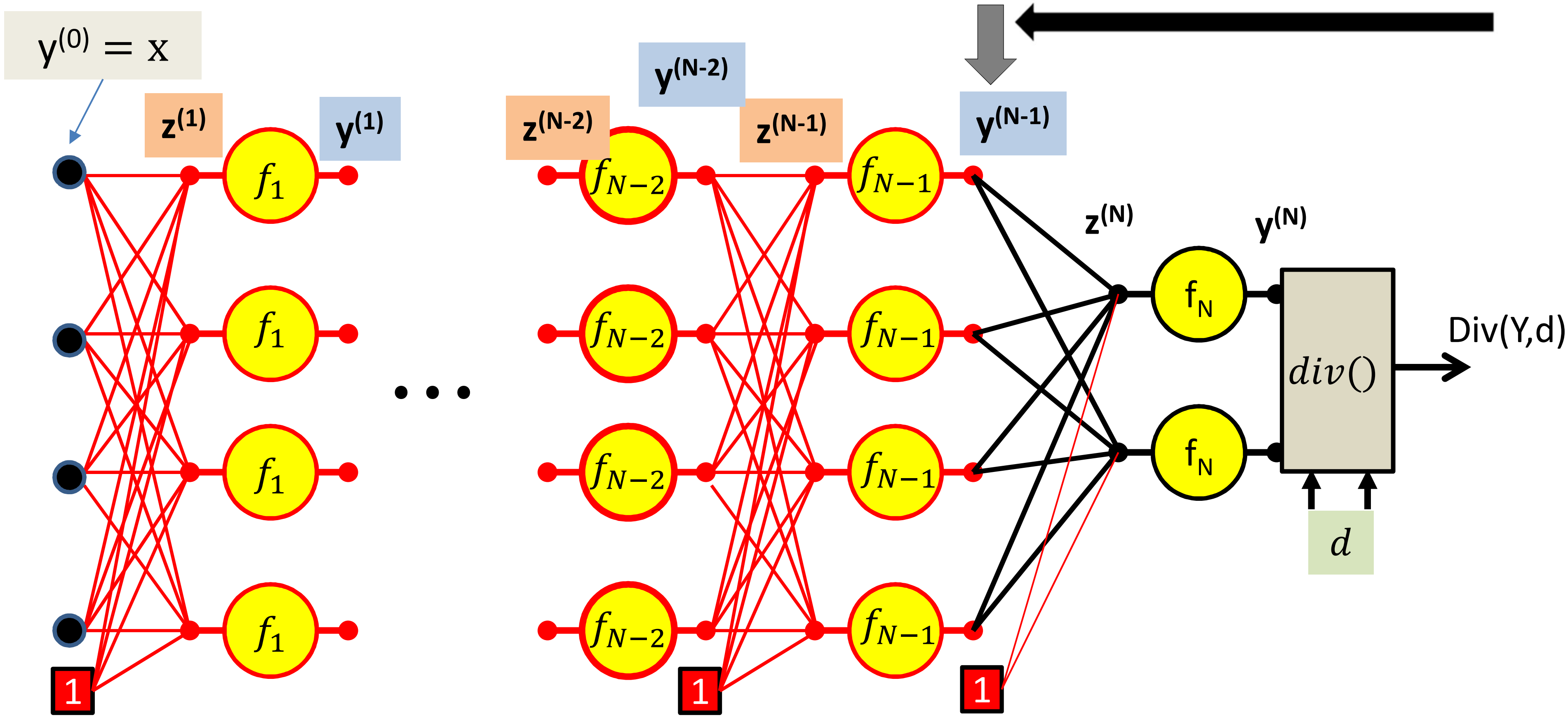
$$z_j^{(N)} = w_{1j}^{(N)} y_1^{(N-1)} + \text{other terms}$$

Computing derivatives



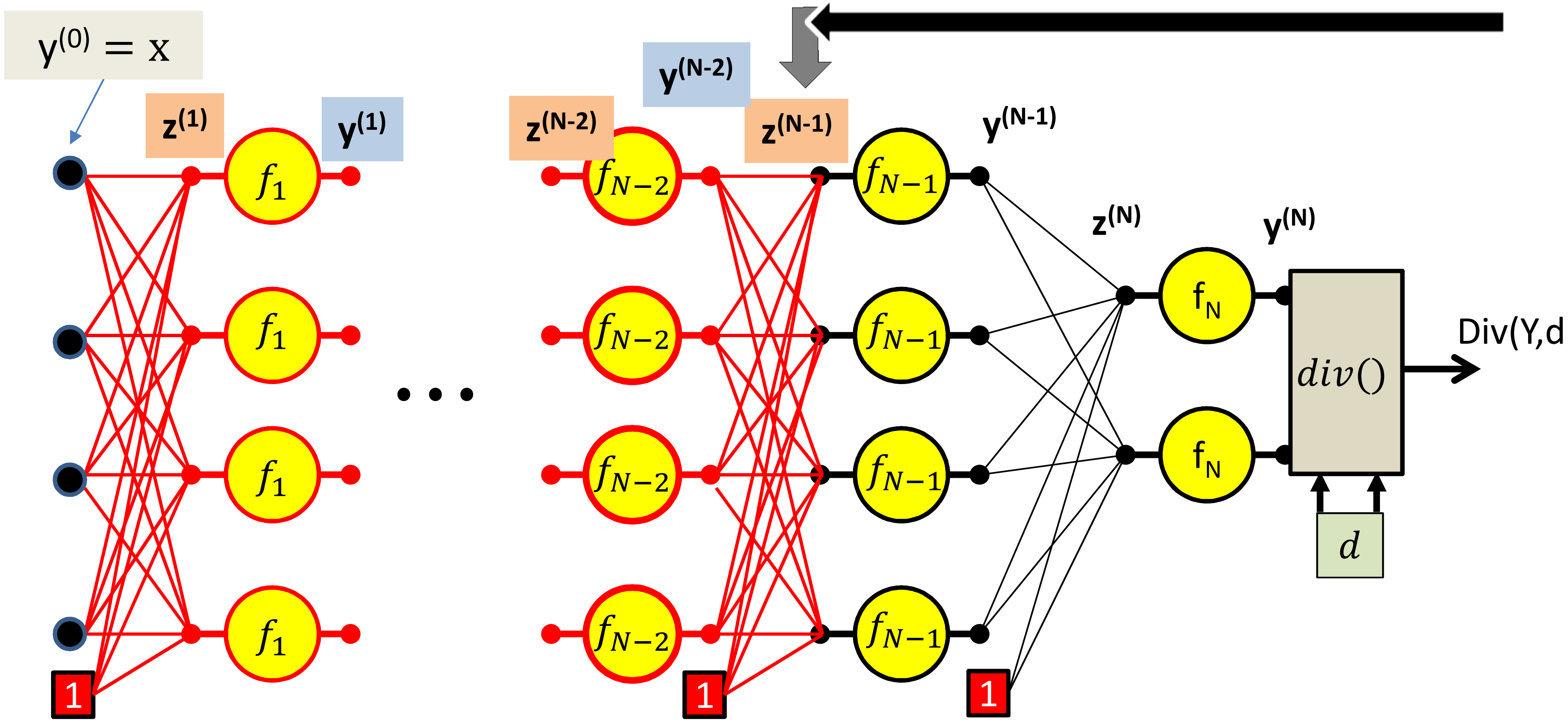
$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j w_{1j}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



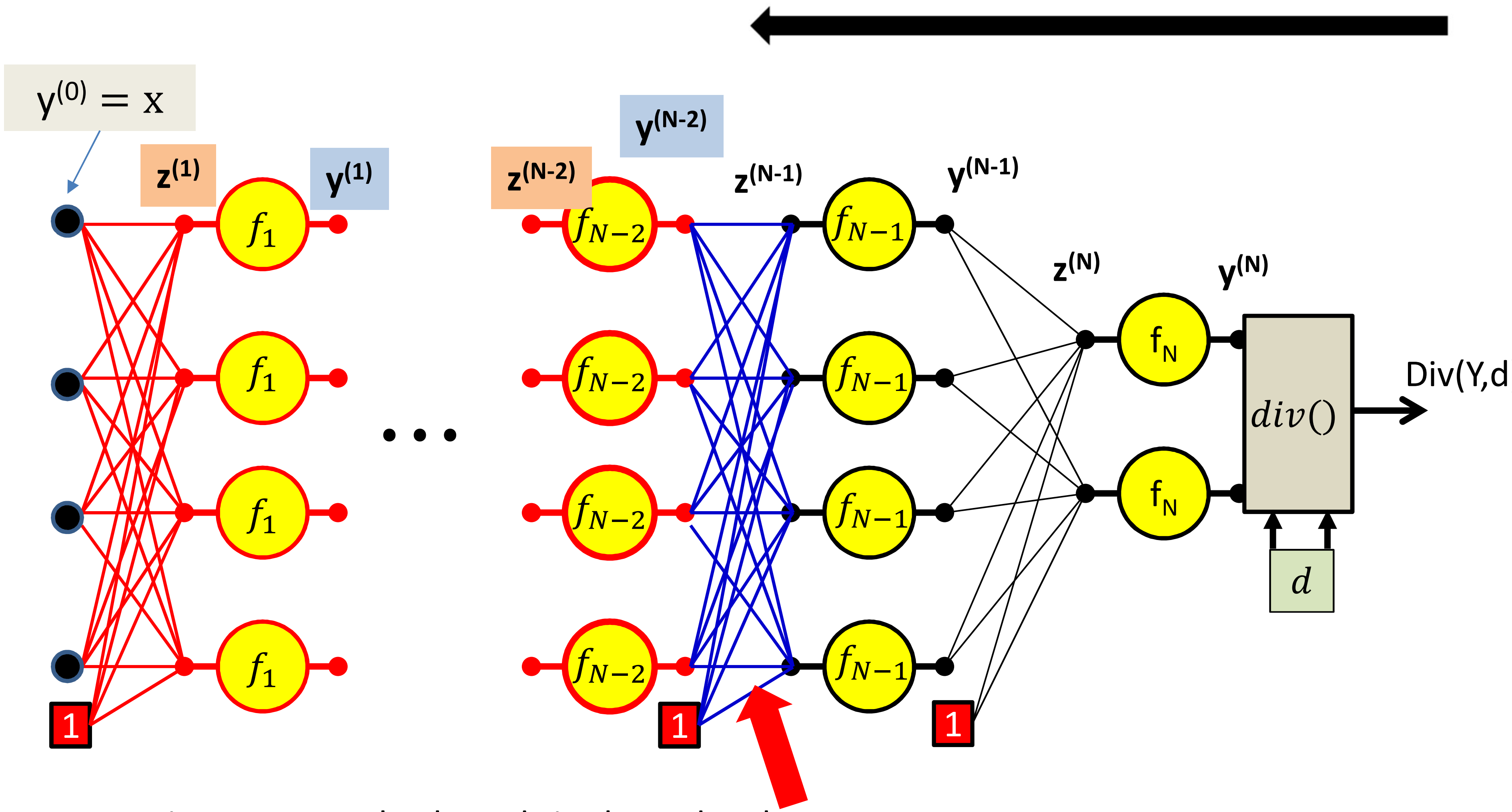
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



We continue our way backwards in the order shown

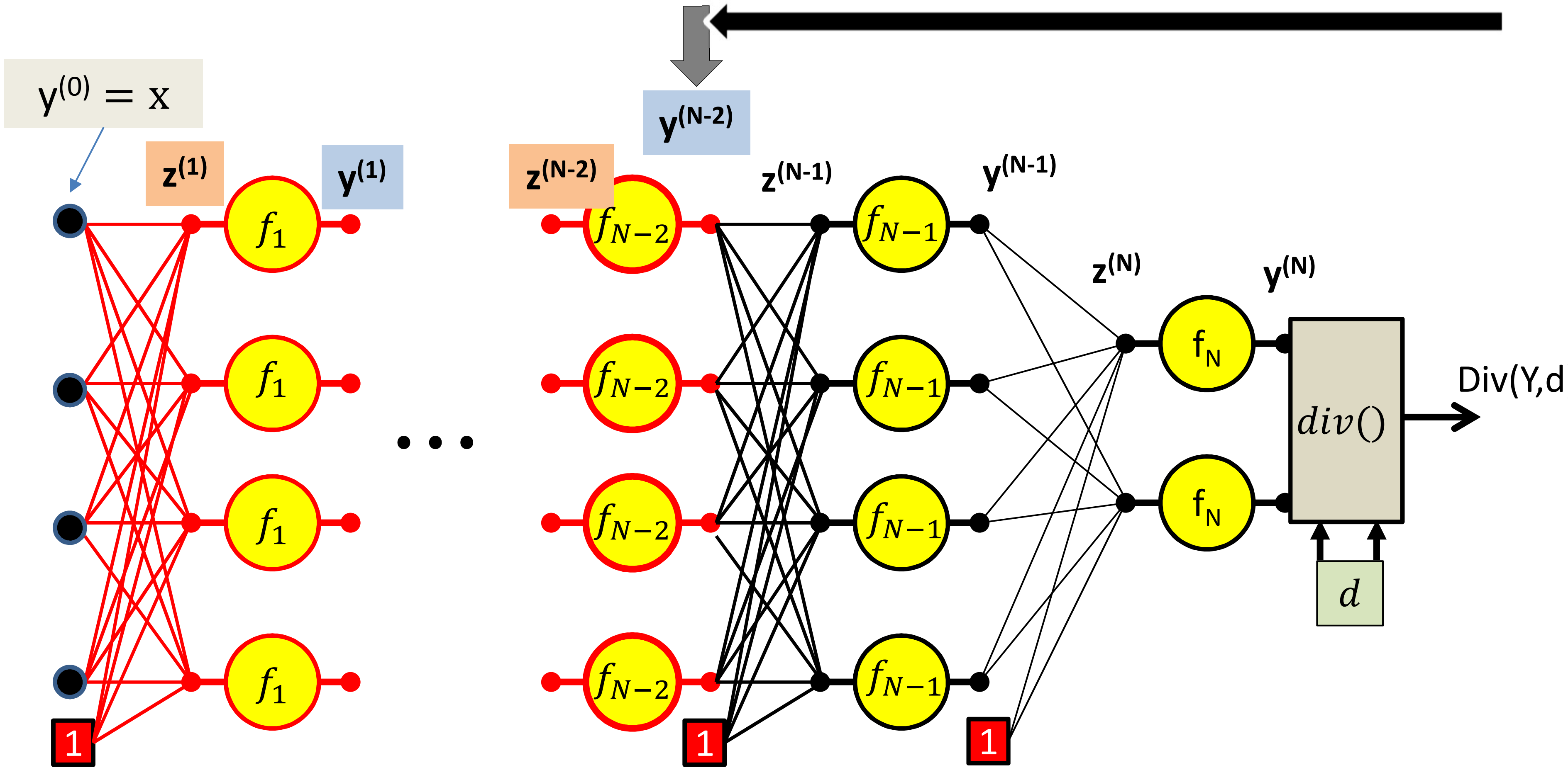
$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f'_{N-1}\left(z_i^{(N-1)}\right)\frac{\partial Div}{\partial y_i^{(N-1)}}$$



We continue our way backwards in the order shown

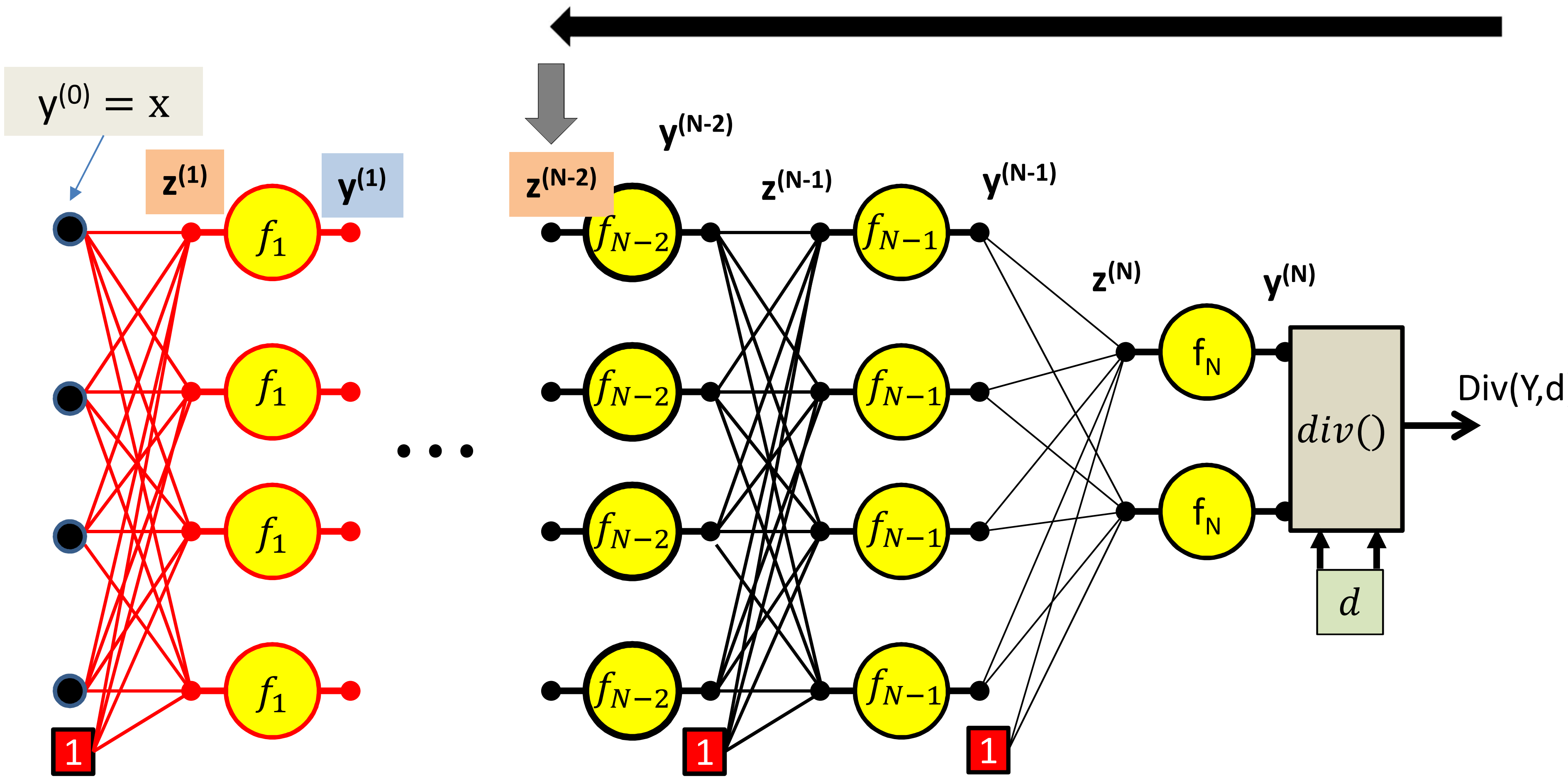
$$\boxed{\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial Div}{\partial z_j^{(N-1)}}}$$

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



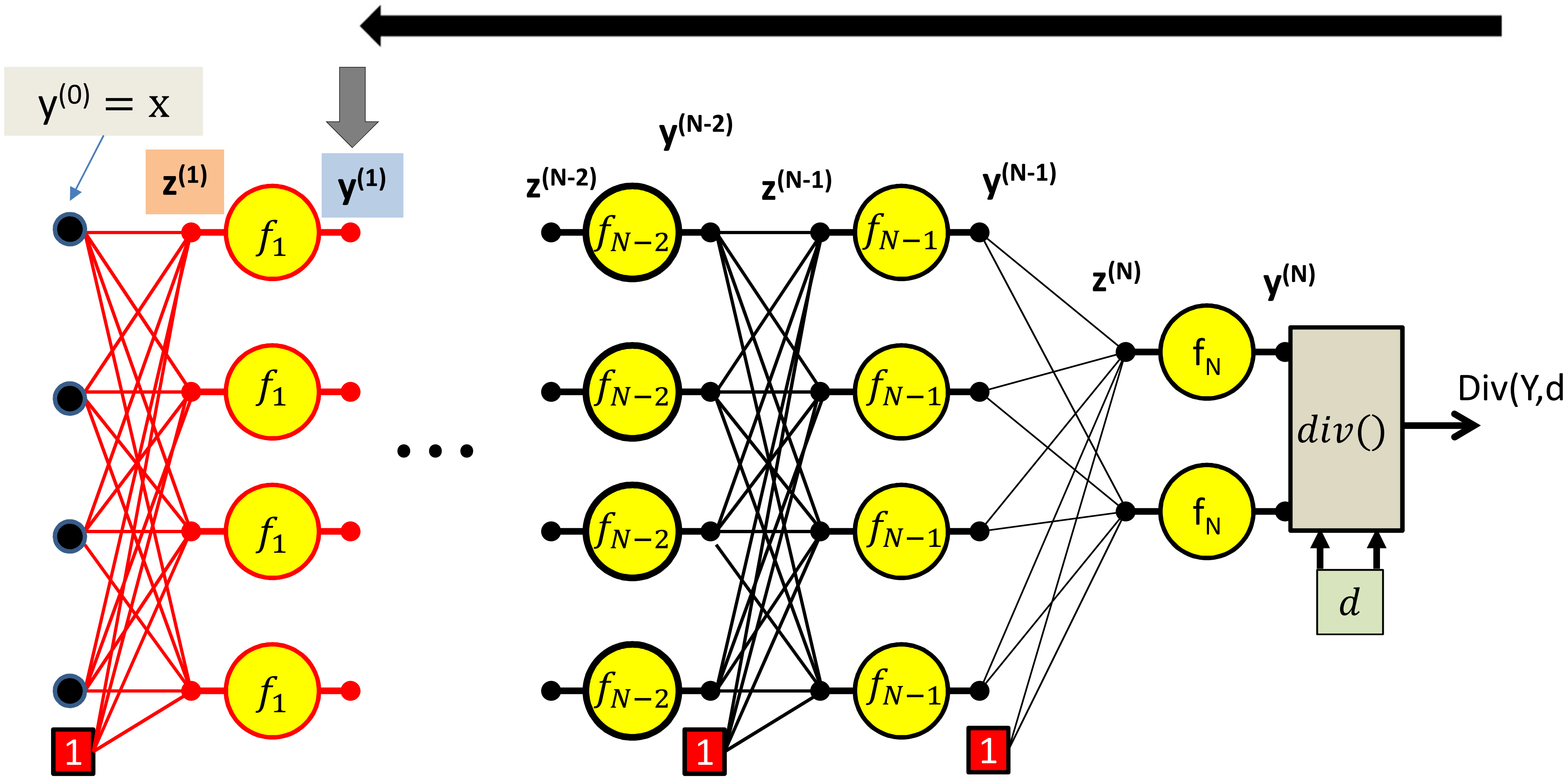
We continue our way backwards in the order shown

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



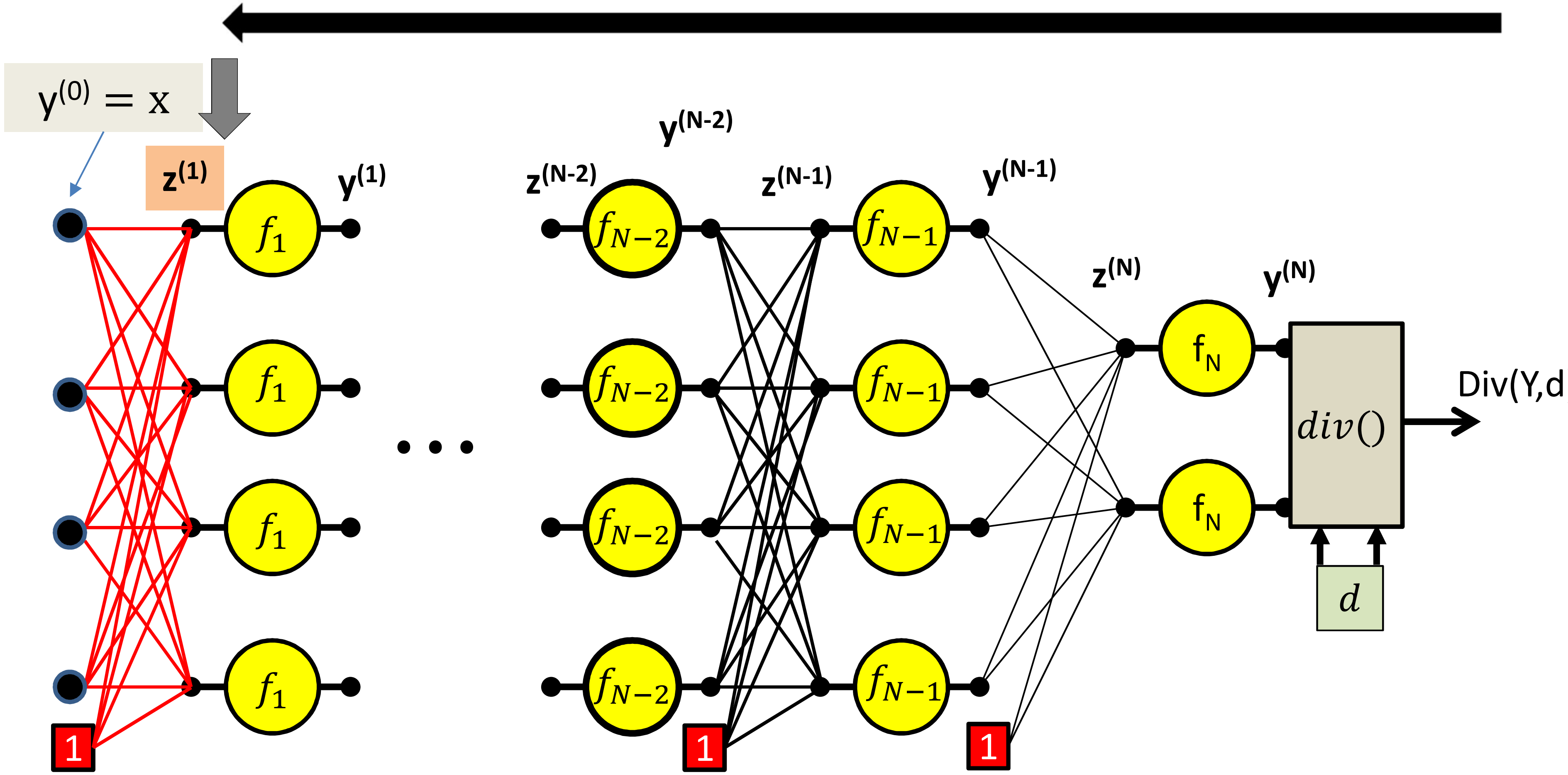
We continue our way backwards in the order shown

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



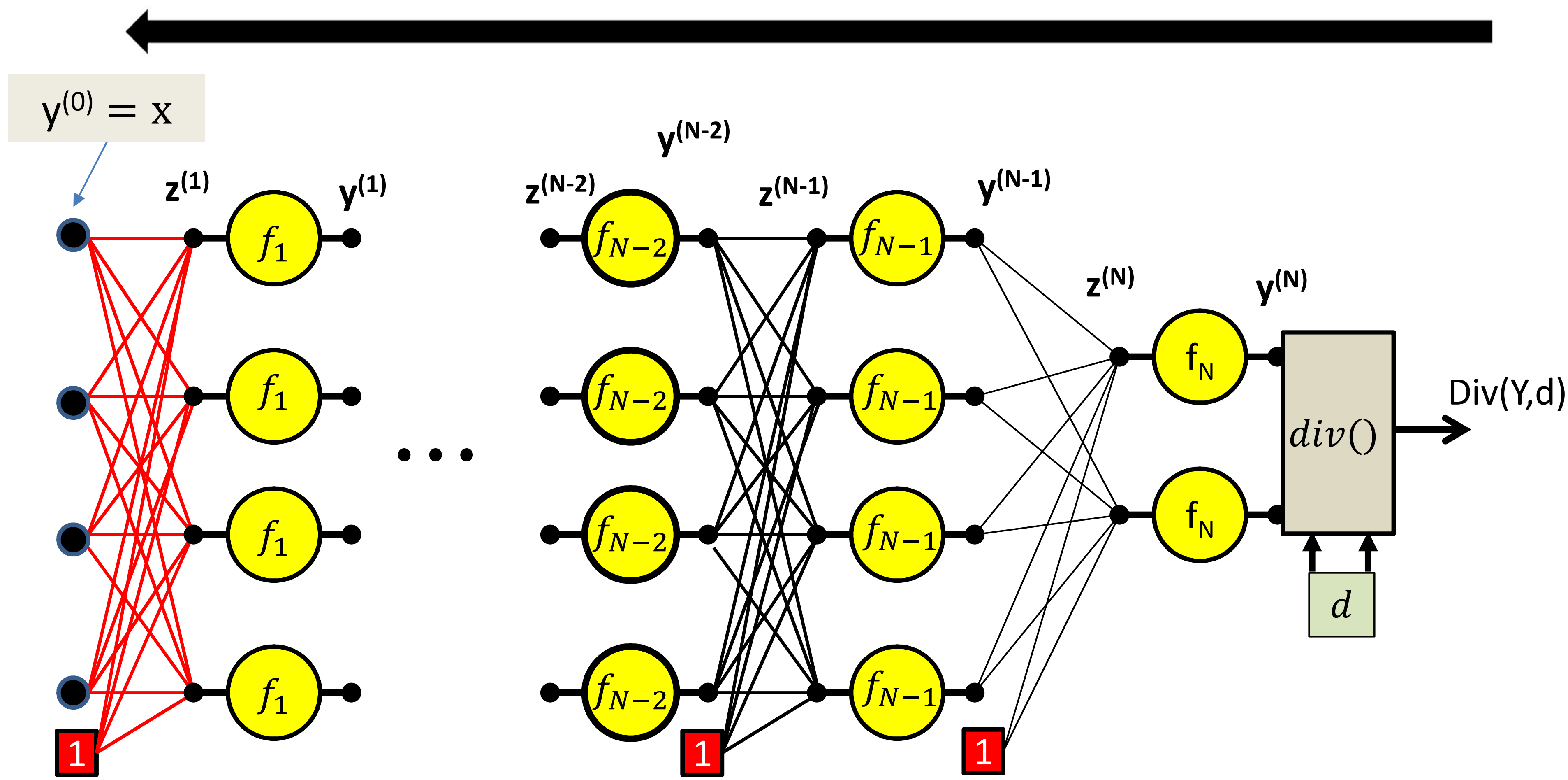
We continue our way backwards in the order shown

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



We continue our way backwards in the order shown

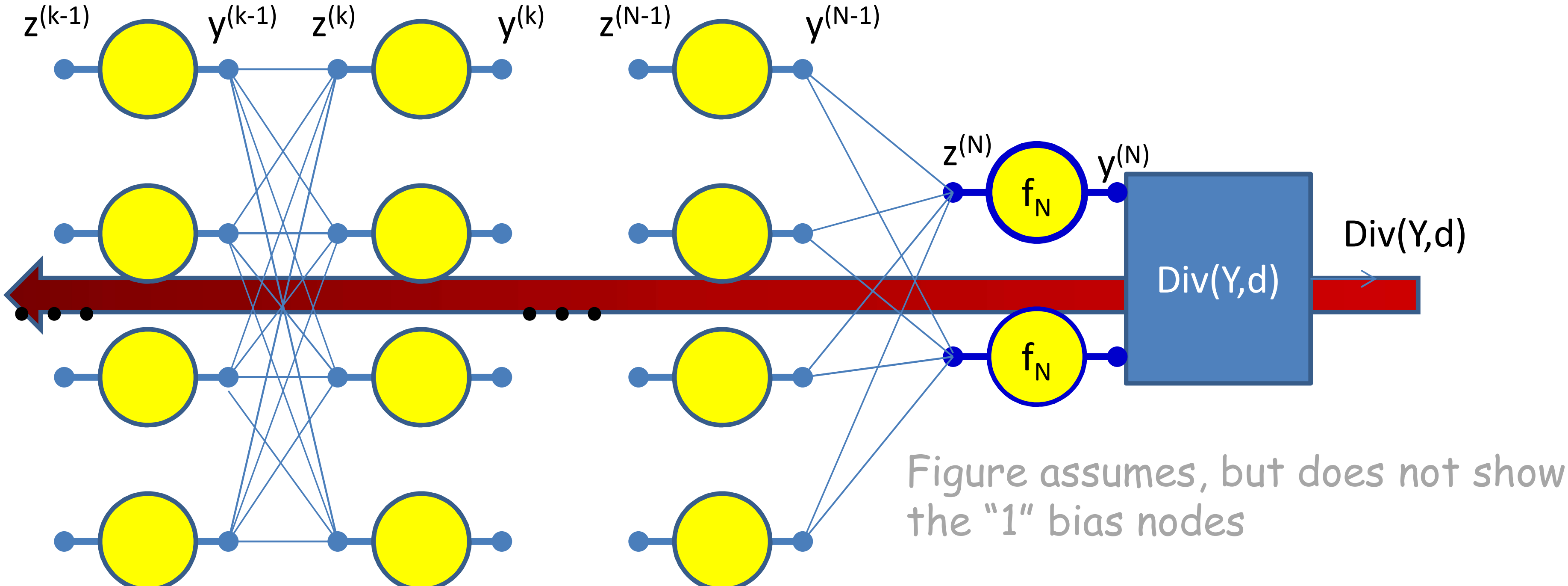
$$\frac{\partial Div}{\partial z_i^{(1)}} = f_1'\left(z_i^{(1)}\right)\frac{\partial Div}{\partial y_i^{(1)}}$$



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



Initialize: Gradient w.r.t network output

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

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

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

$\quad For\ i\ =\ 1{:}\ layer\ width$

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

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

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

Backward Pass

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

Backward Pass

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

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

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

Very analogous to the forward pass:

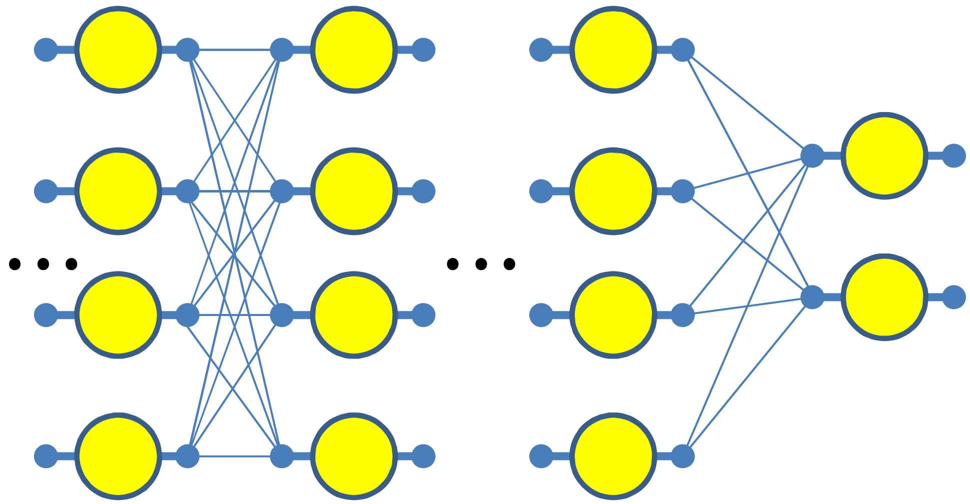
Backward weighted combination of next layer

Backward equivalent of activation

For comparison: the forward pass again

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

Special 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. Outputs of neurons only combine through weighted addition
  3. Activations are actually differentiable
  – All of these conditions are frequently not applicable
- Will not dwell on the topic in class, but explained in slides
  – Will appear in quiz. Please read the slides