CSC 561: Neural Networks and Deep Learning

Learning (part 3)

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2025

THINK BIG WE DO℠

Neural Networks
Learning the network: Part 3

11-785, Spring 2025

Lecture 5

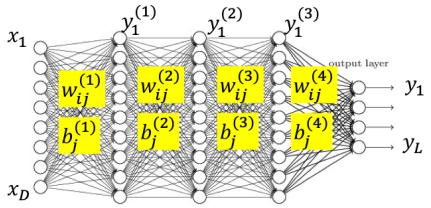
Training neural nets through Empirical Risk Minimization: Problem Setup

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

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

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

Notation



- The input layer is the 0[th] layer
- We will represent the output of the i-th perceptron of the k[th] 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 $b_j^{(k)}$

Training Neural Nets through Gradient Descent

Total training Loss:

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

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

Assuming the bias is also represented as a weight

  - Using the extended notation: the bias is also a weight
- Do:
  - For every layer k for all i, j, update:
    - $w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{dLoss}{dw_{i,j}^{(k)}}$
- Until $Loss$ has converged

The derivative

Total training Loss:

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

- Computing the derivative

Total derivative:

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

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

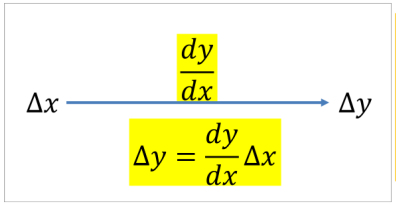
Calculus Refresher: Basic rules of calculus

For any differentiable function
$y = f(x)$
with derivative
$\frac{dy}{dx}$
the following must hold for sufficiently small Δx ⇨ $\Delta y \approx \frac{dy}{dx}\Delta x$

Calculus Refresher: Basic rules of calculus

For any differentiable function
$y = f(x)$
with derivative
$\frac{dy}{dx}$
the following must hold for sufficiently small Δx ⇨ $\Delta y \approx \frac{dy}{dx}\Delta x$

Introducing the "influence" diagram: x influences y



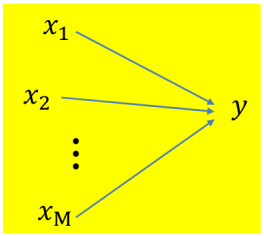
The derivative graph: The edge carries the derivative.

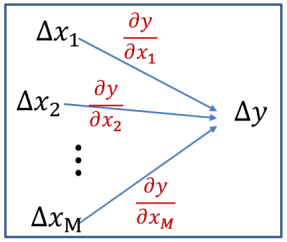
Node and edge weights multiply

Calculus Refresher: Basic rules of calculus

For any differentiable function
$y = f(x_1, x_2, \dots, x_M)$
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 $\Delta x_1, \Delta x_2, \dots, \Delta x_M$

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

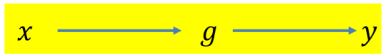


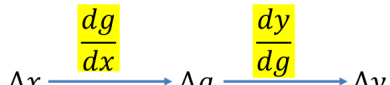


Calculus Refresher: Chain rule

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

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

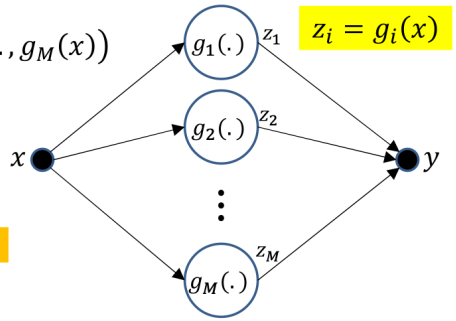




$$\Delta y = \frac{dy}{dg(x)}\frac{dg(x)}{dx}\Delta x$$

Distributed Chain Rule: Influence Diagram

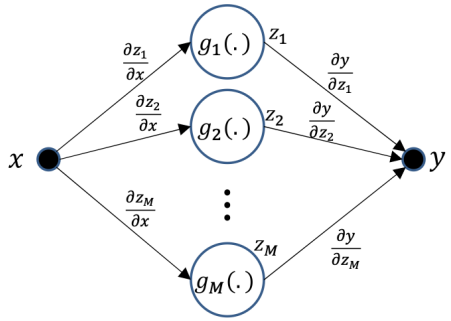
$y = f\big(g_1(x), g_1(x), \dots, g_M(x)\big)$



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

Distributed Chain Rule: Influence Diagram

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



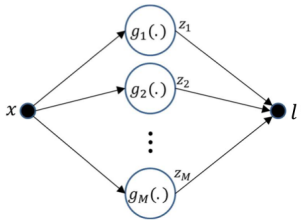
Calculus Refresher: Chain rule summary

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

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

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

where $z_i = g_i(x)$



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

23

3

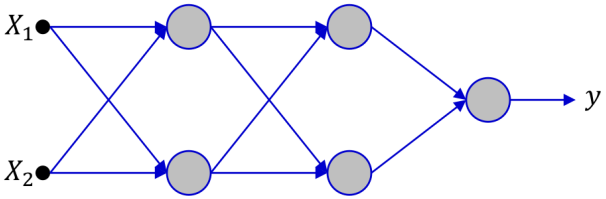
Our problem for today

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

24

4

A first closer look at the network

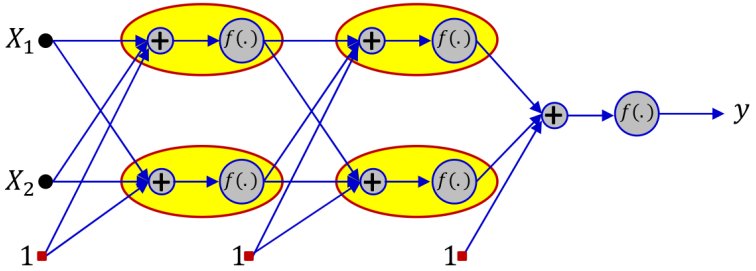


- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs

27

5

A first closer look at the network

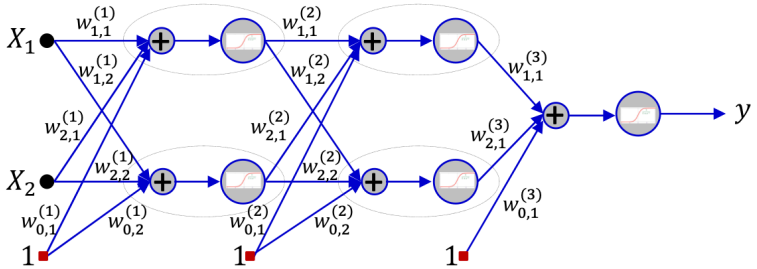


- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Explicitly separating the affine function of inputs from the activation

28

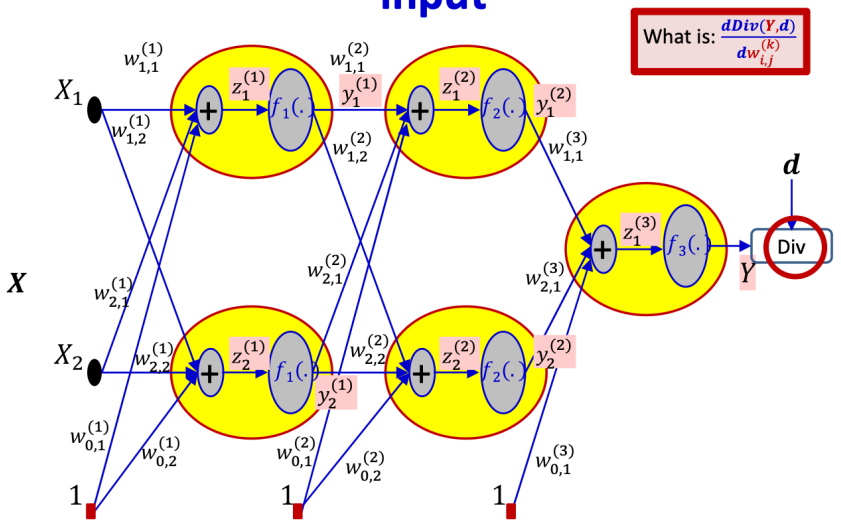
6

A first closer look at the network

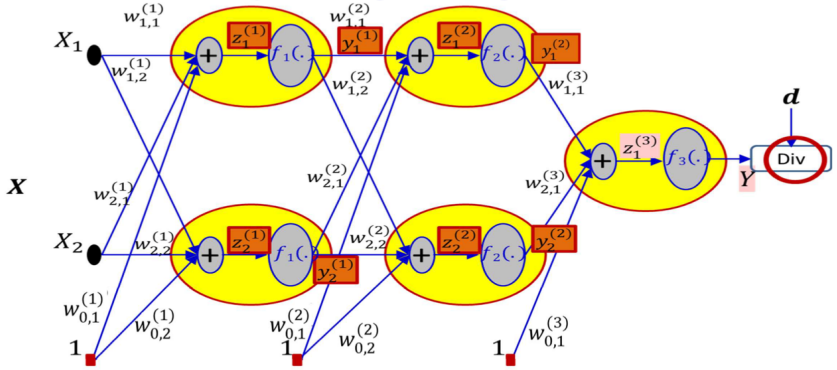


- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Expanded **with all weights shown**
- Let's label the other variables too…

Computing the derivative for a *single* input

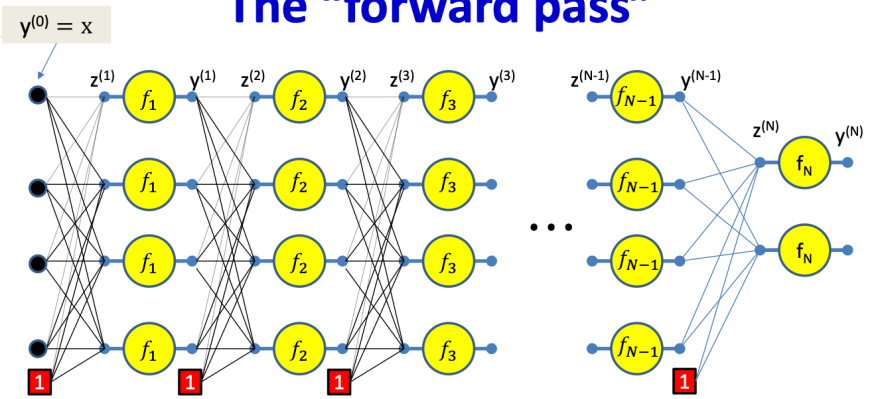


Computing the gradient



- Note: computation of the derivative $\frac{dDiv(Y,d)}{dw_{i,j}^{(k)}}$ requires intermediate and final output values of the network in response to the input

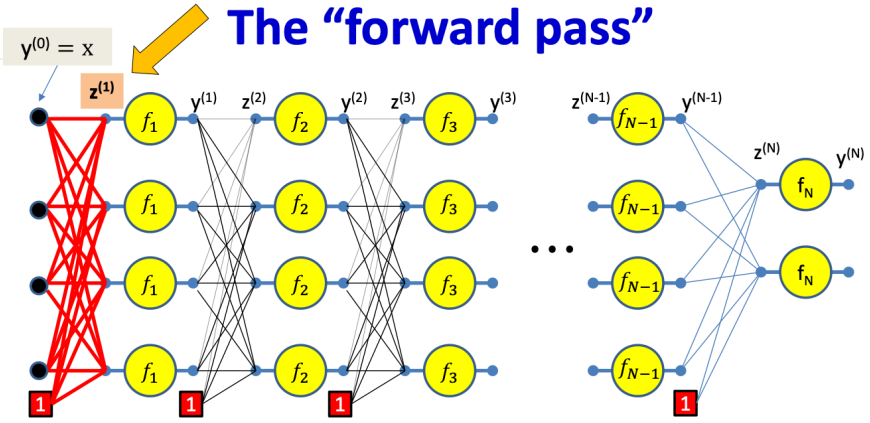
The "forward pass"



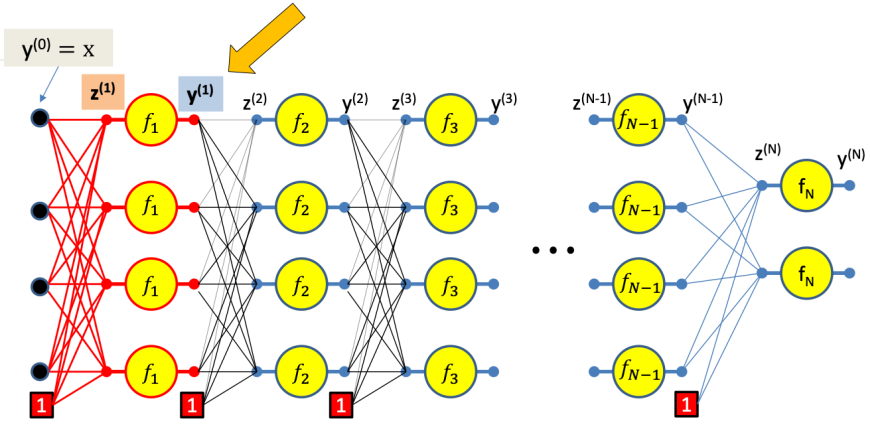
Setting $y_i^{(0)} = x_i$ for notational convenience

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

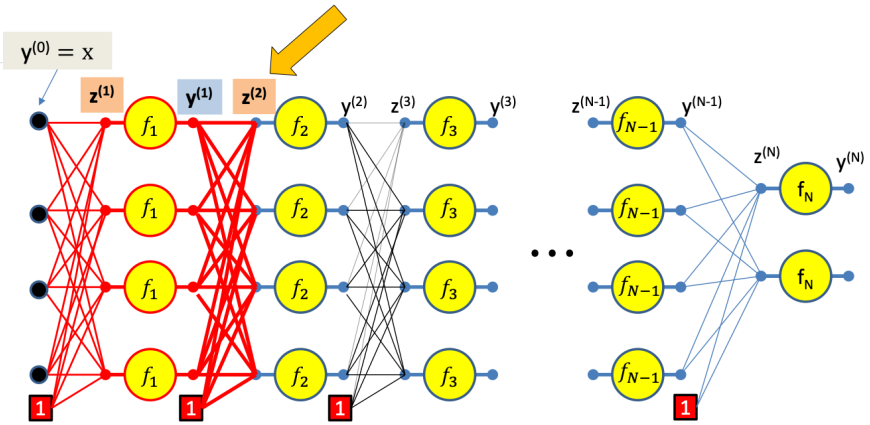
The "forward pass"



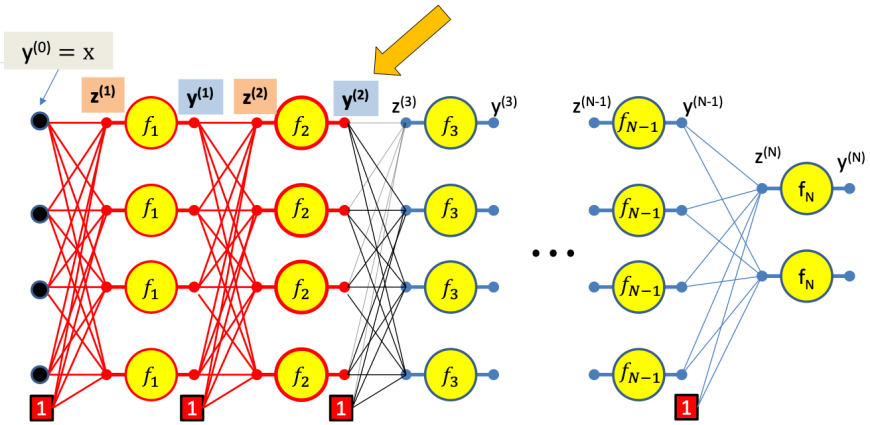
$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$$



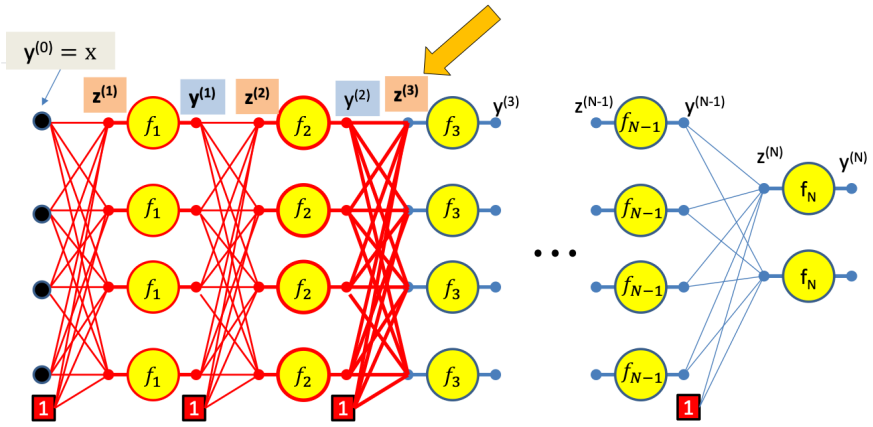
$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \qquad y_j^{(1)} = f_1\left(z_j^{(1)}\right)$$



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



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

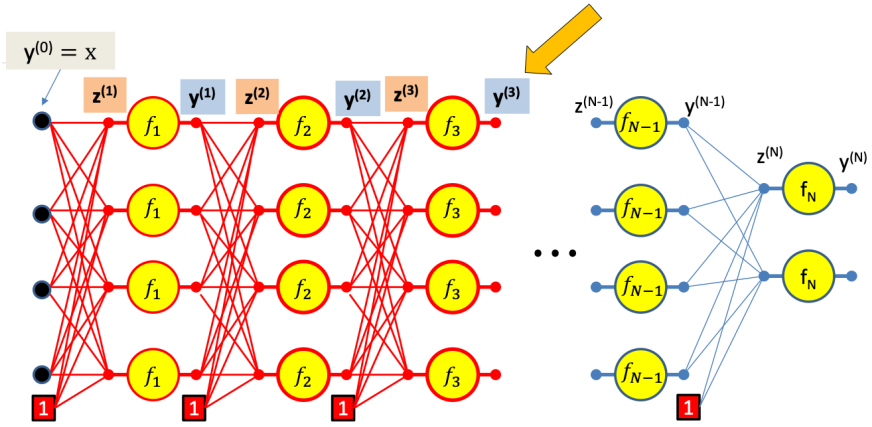


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \qquad y_j^{(1)} = f_1\left(z_j^{(1)}\right) \qquad z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \qquad y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$

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

40

5

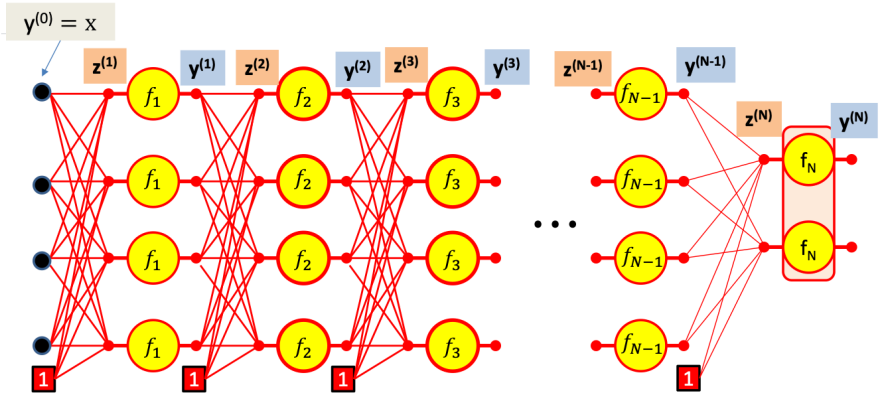


$$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \qquad y_j^{(1)} = f_1\left(z_j^{(1)}\right) \qquad z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \qquad y_j^{(2)} = f_2\left(z_j^{(2)}\right)$$

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

41

6

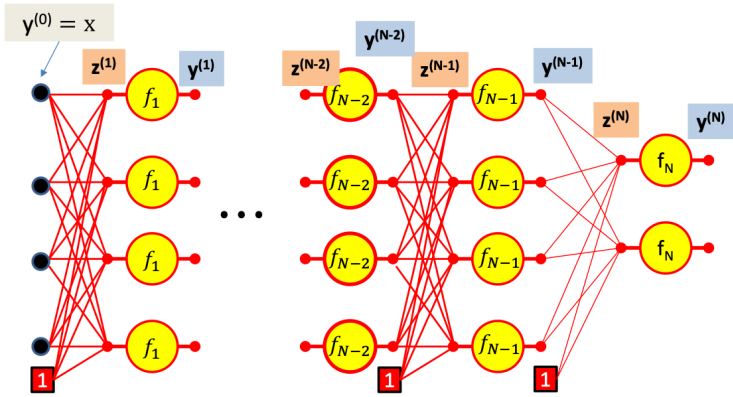


$$y_j^{(N-1)} = f_{N-1}\left(z_j^{(N-1)}\right) \qquad z_j^{(N)} = \sum_i w_{ij}^{(N)} y_i^{(N-1)} \qquad \boldsymbol{y}^{(N)} = f_N\left(\boldsymbol{z}^{(N)}\right)$$

42

7

Computing derivatives



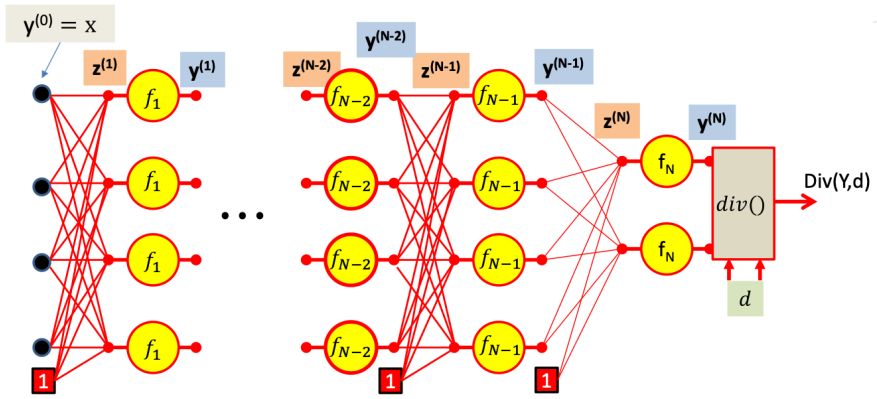
We have computed all these intermediate values in the forward computation

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

45

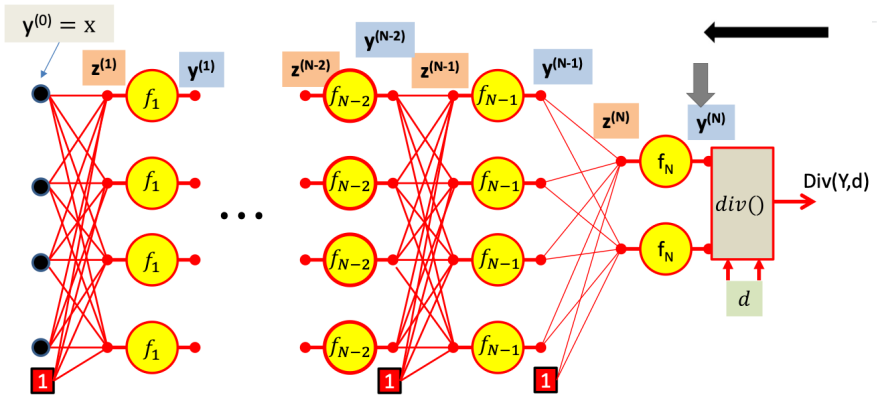
8

Computing derivatives



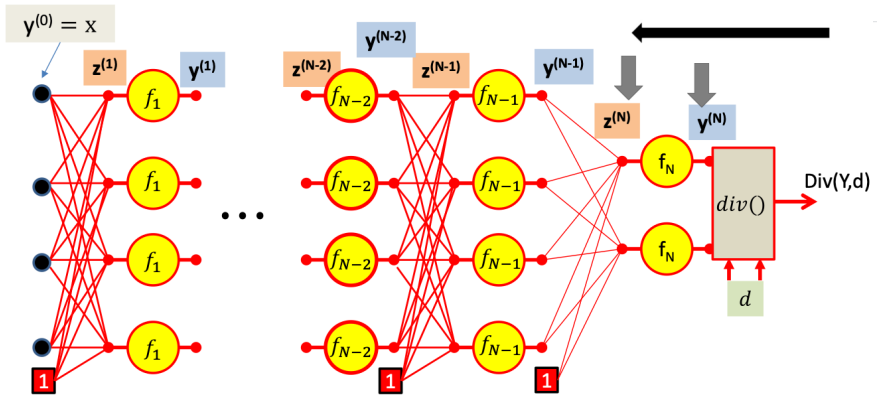
First, we compute the divergence between the output of the net y = $y^{(N)}$ and the desired output d

Computing derivatives



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

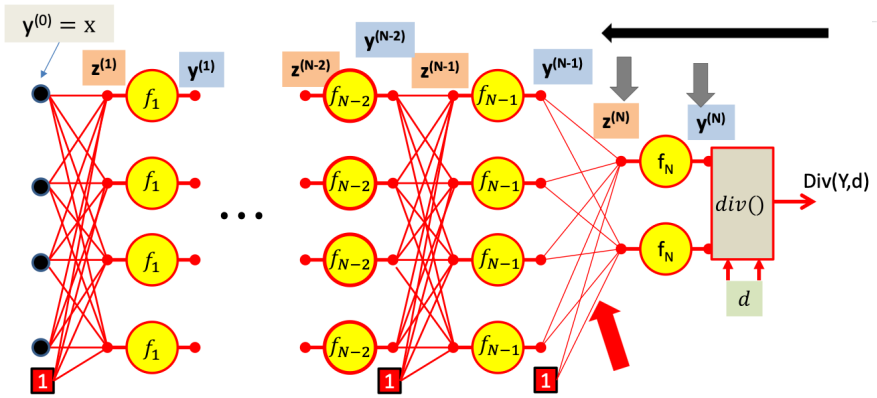
Computing derivatives



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

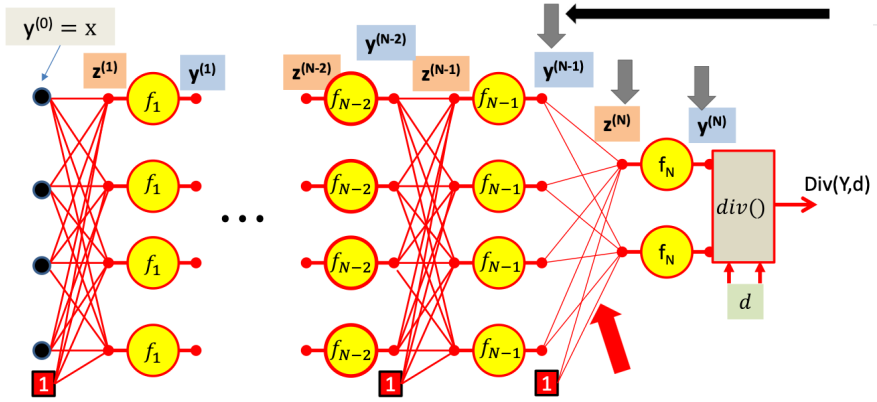
We then compute $\nabla_{z^{(N)}} div(.)$ the derivative of the divergence w.r.t. the *pre-activation* affine combination $z^{(N)}$ using the chain rule

Computing derivatives



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

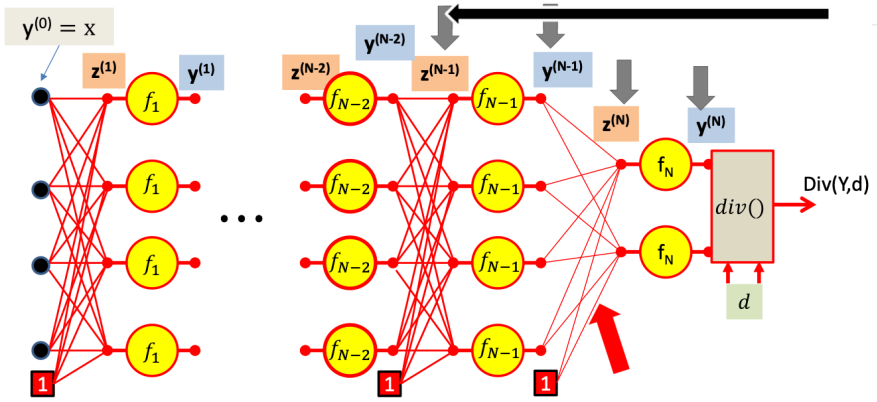
Computing derivatives



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

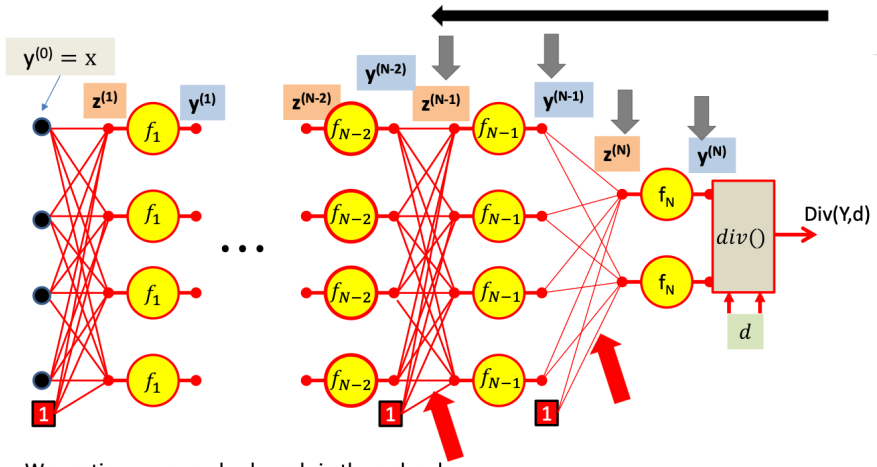
Then continue with the chain rule to compute $\nabla_{Y^{(N-1)}} div(.)$ the derivative of the divergence w.r.t. the output of the N-1th layer

Computing derivatives



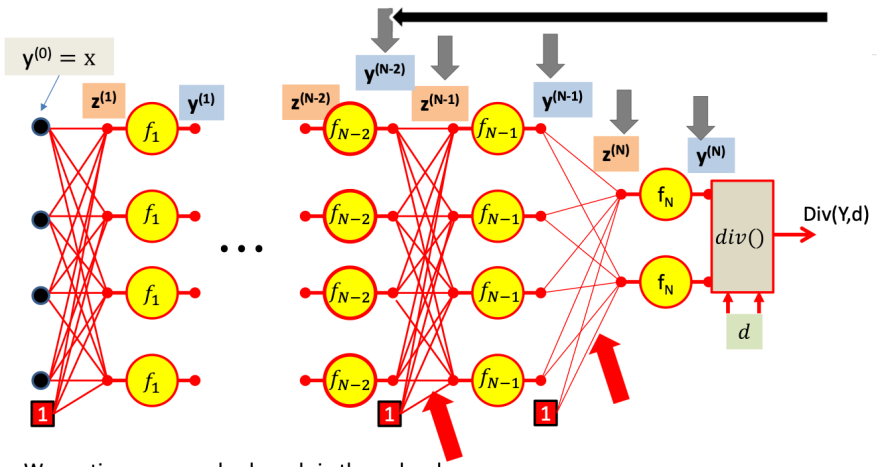
We continue our way backwards in the order shown

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



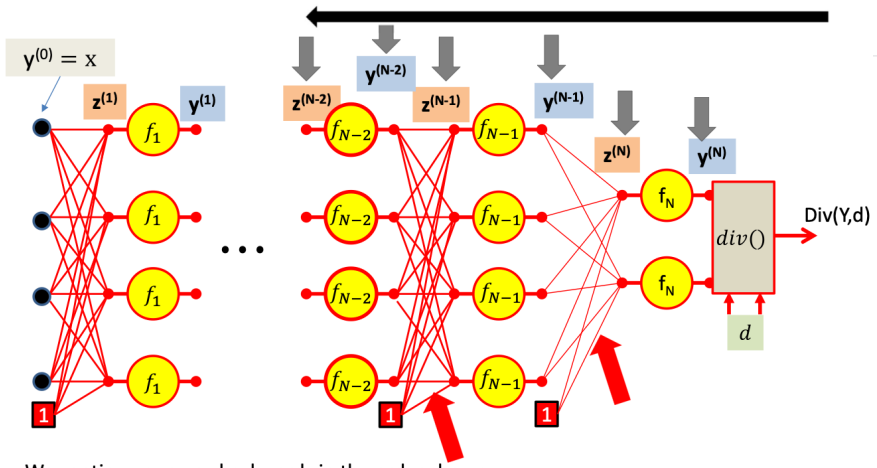
We continue our way backwards in the order shown

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



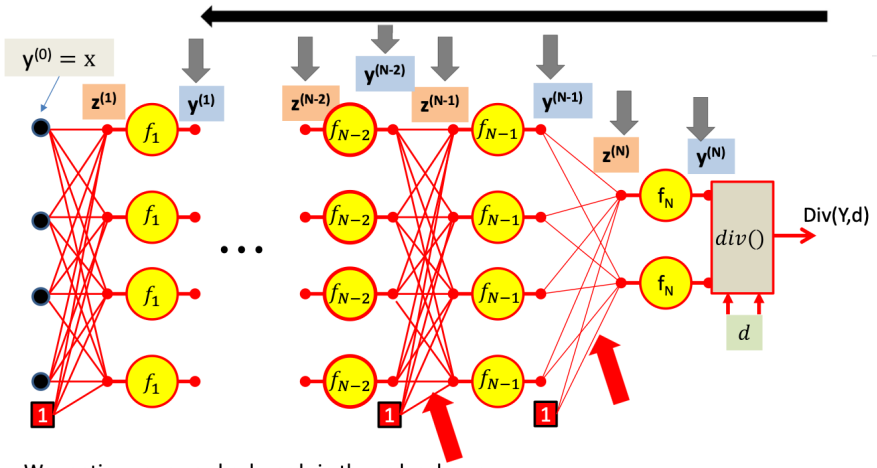
We continue our way backwards in the order shown

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



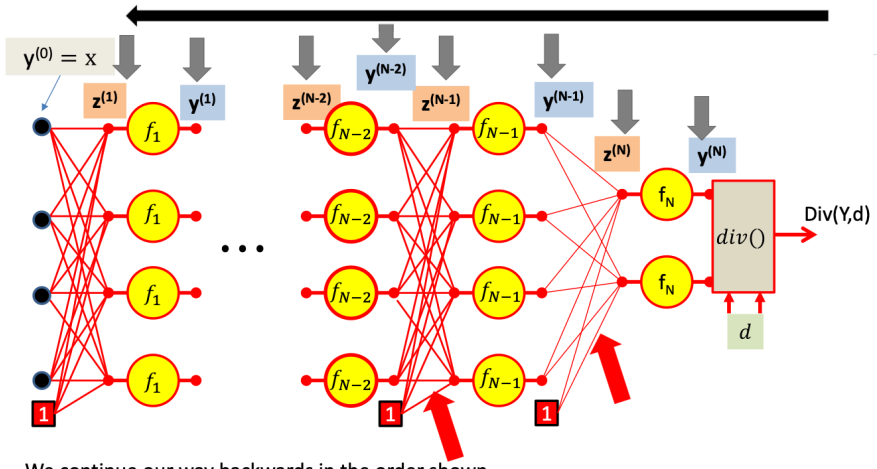
We continue our way backwards in the order shown

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



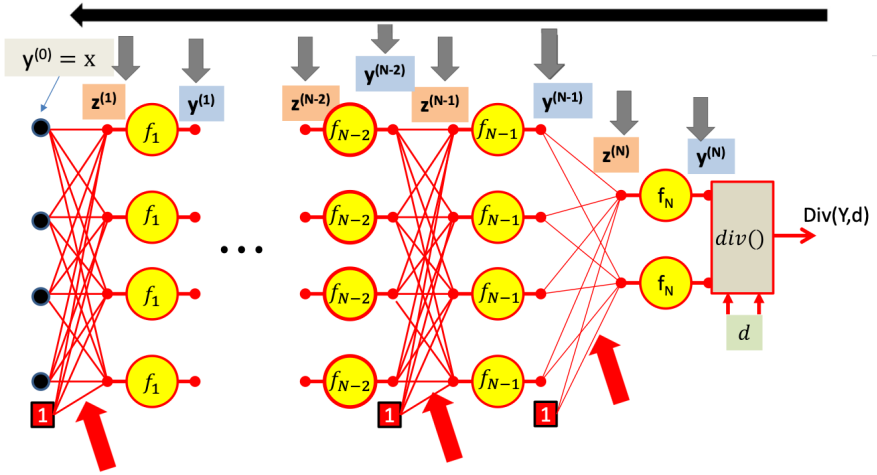
We continue our way backwards in the order shown

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



We continue our way backwards in the order shown

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



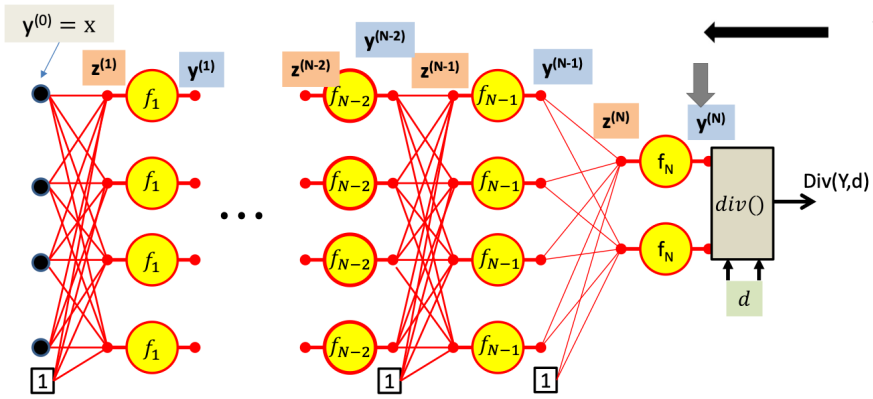
We continue our way backwards in the order shown

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

Backward Gradient Computation

- Let's actually see the math..

Computing derivatives



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

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

Calculus Refresher: Chain rule

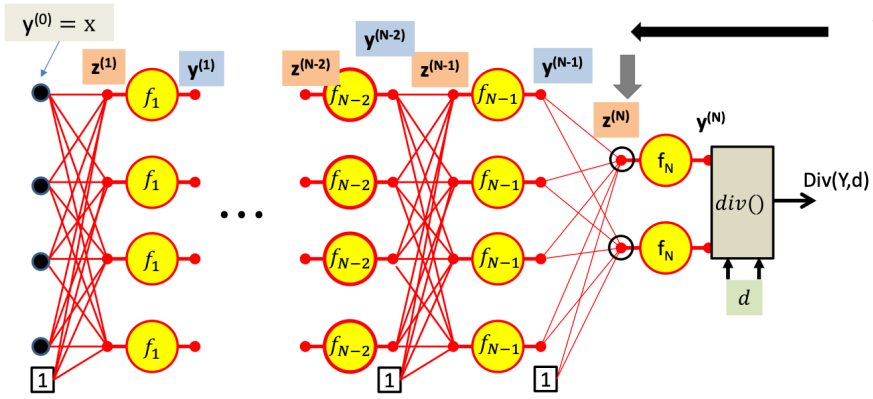
For any nested function $l = f(y)$ where $y = g(z)$

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

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

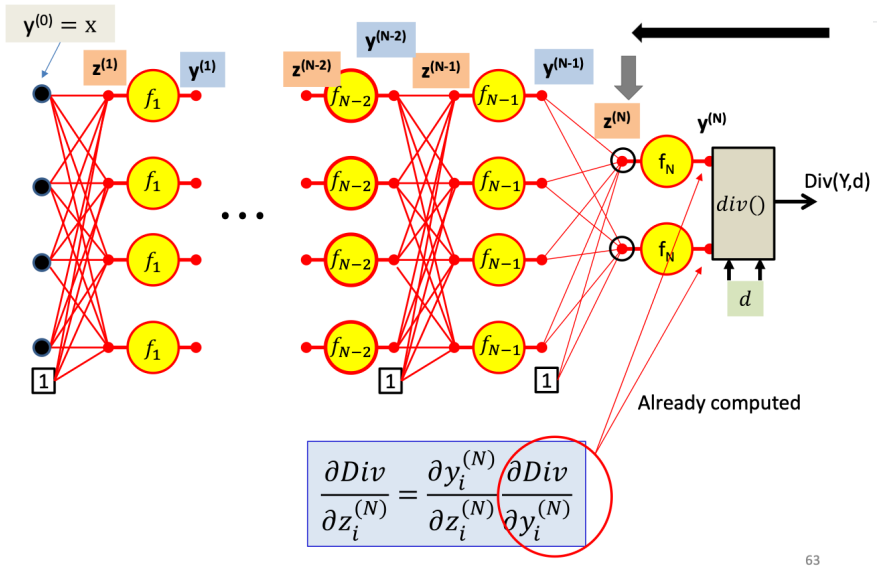
$z \longrightarrow y \longrightarrow l$

Computing derivatives



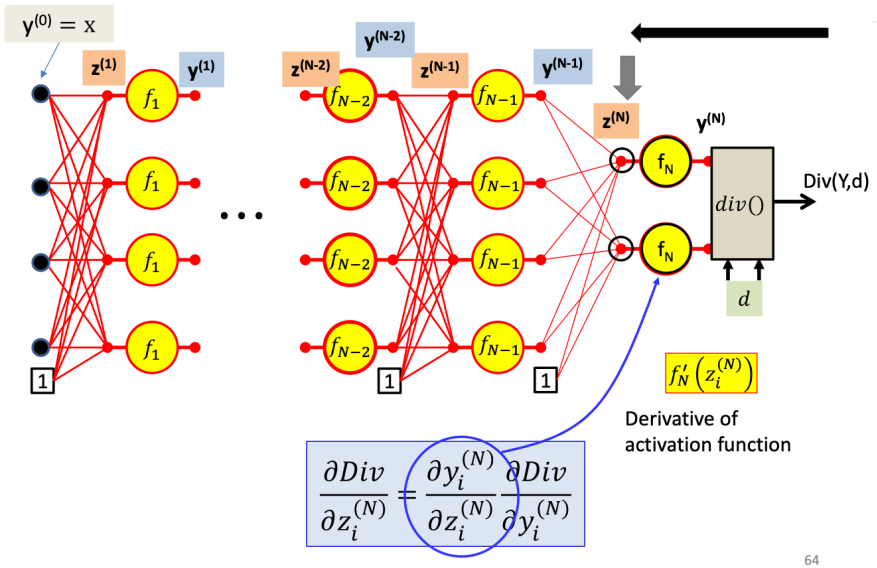
$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}\frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



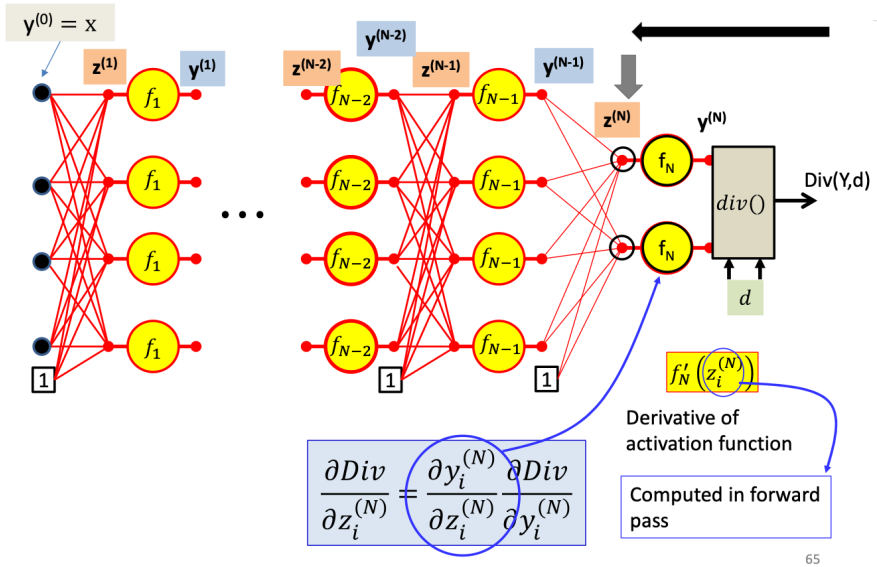
$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



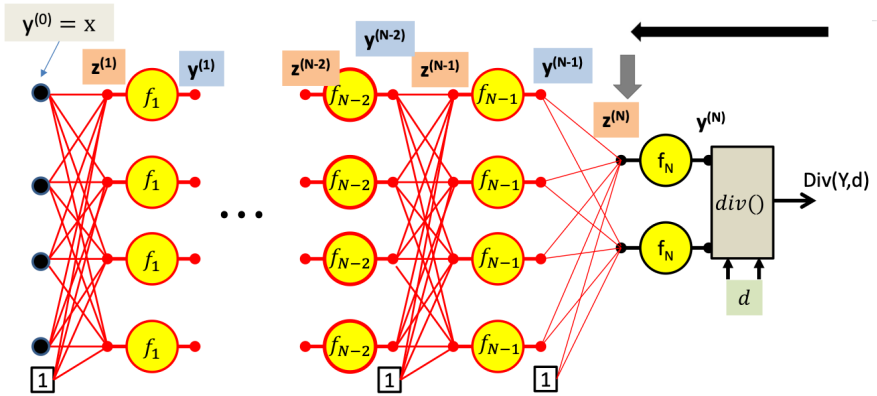
$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



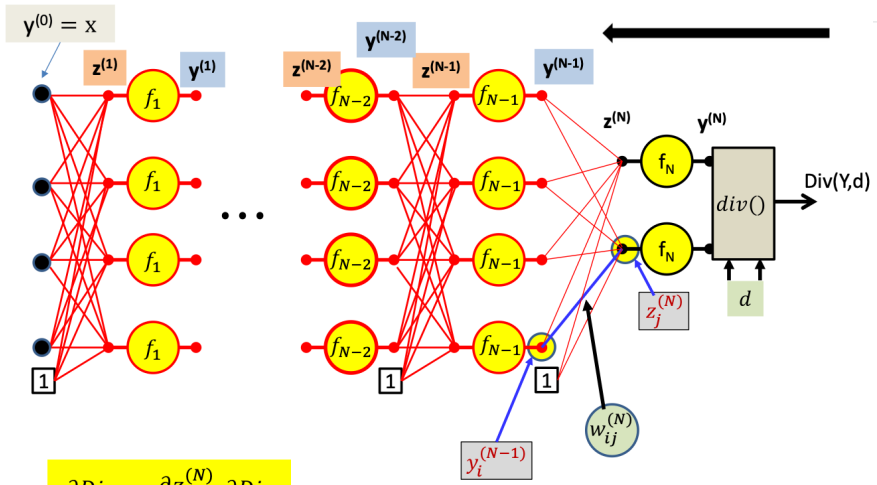
$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



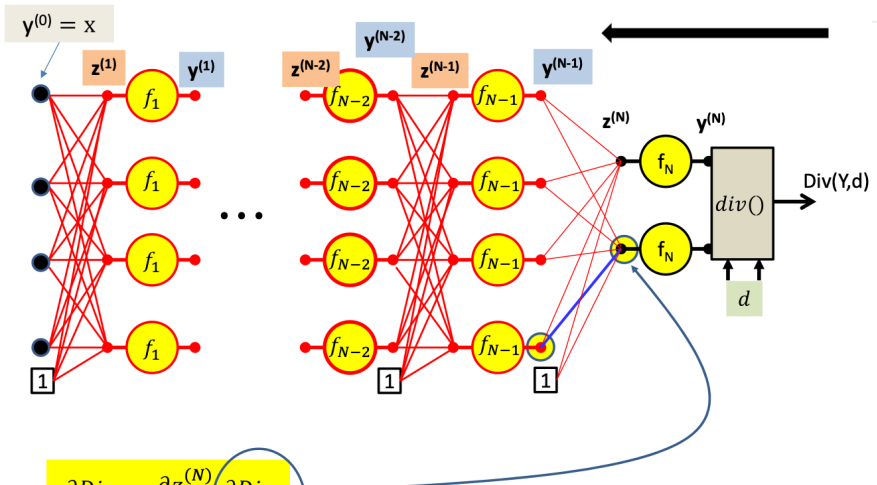
$$\frac{\partial Div}{\partial z_i^{(N)}} = f_N'\left(z_i^{(N)}\right) \frac{\partial Div}{\partial y_i^{(N)}}$$

Computing derivatives



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

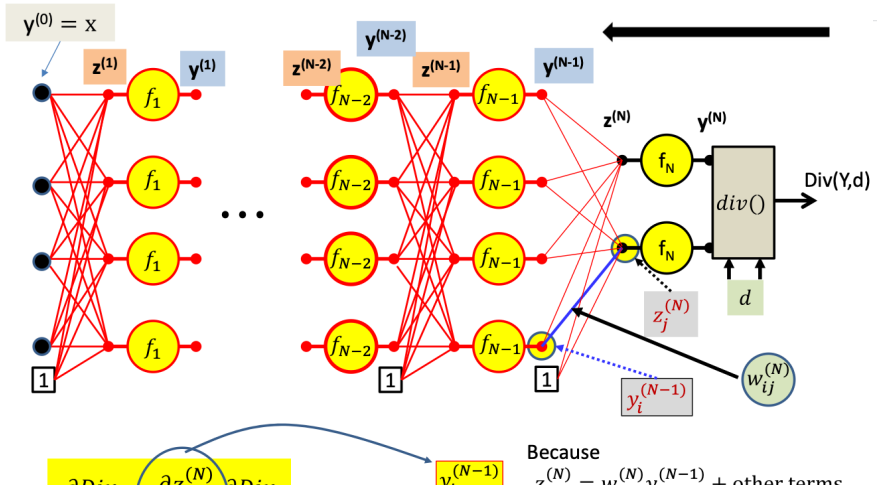
Computing derivatives



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

Just computed

Computing derivatives



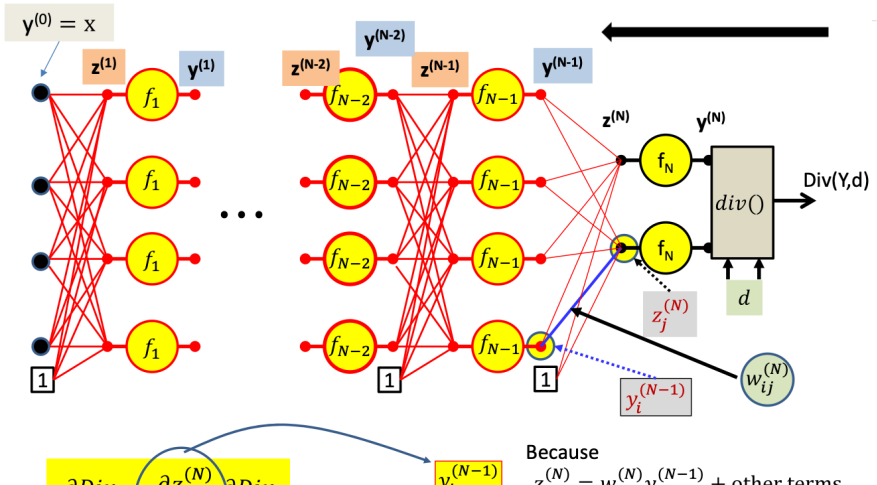
$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$y_i^{(N-1)}$

Because

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

Computing derivatives



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

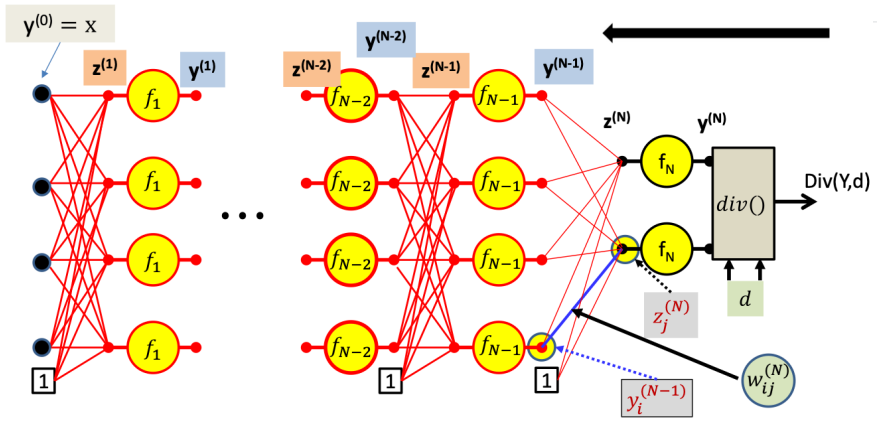
$y_i^{(N-1)}$

Because

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

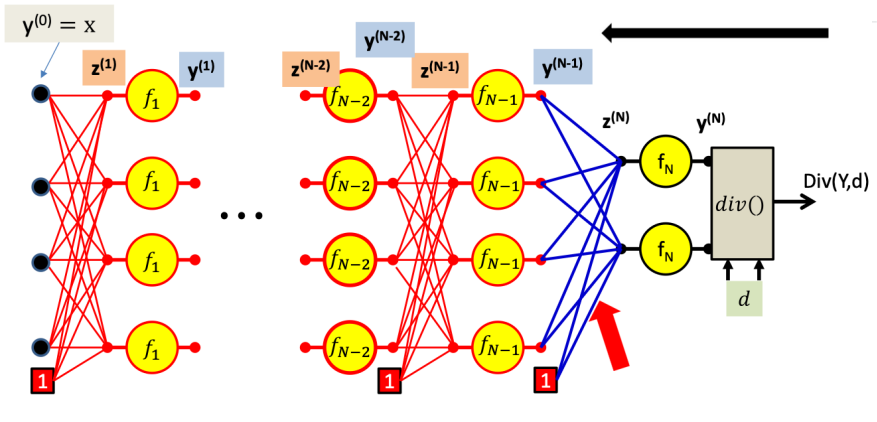
Computed in forward pass

Computing derivatives



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

Computing derivatives



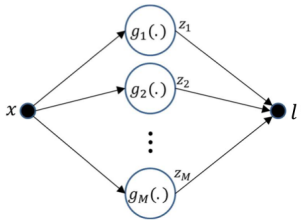
$$\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$$

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

Calculus Refresher: Chain rule

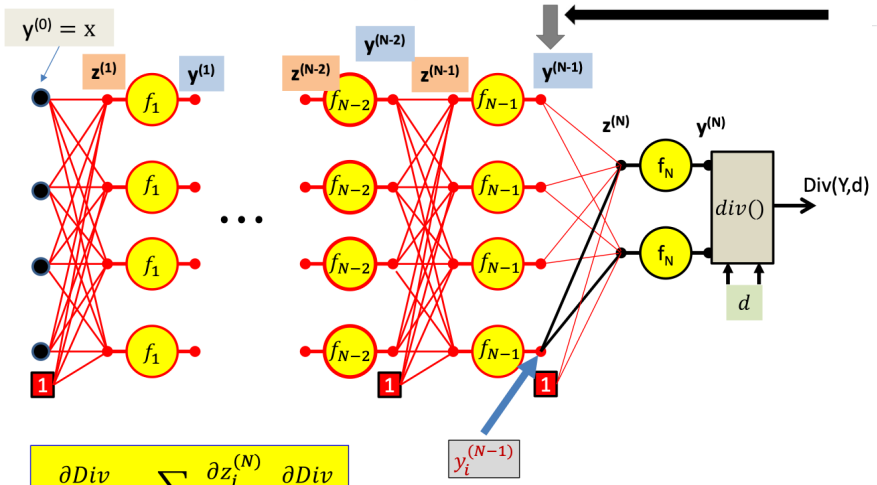
For $l = f(z_1, z_2, \ldots, z_M)$

where $z_i = g_i(x)$



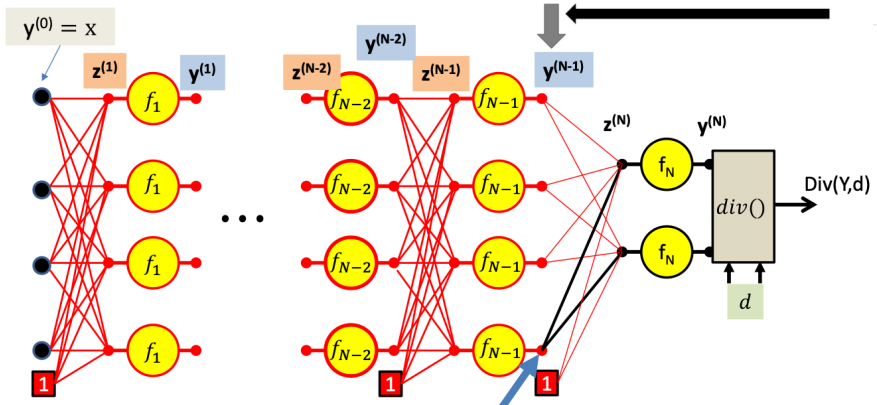
$$\frac{dl}{dx} = \frac{\partial l}{\partial z_1}\frac{dz_1}{dx} + \frac{\partial l}{\partial z_2}\frac{dz_2}{dx} + \cdots + \frac{\partial l}{\partial z_M}\frac{dz_M}{dx}$$

Computing derivatives



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

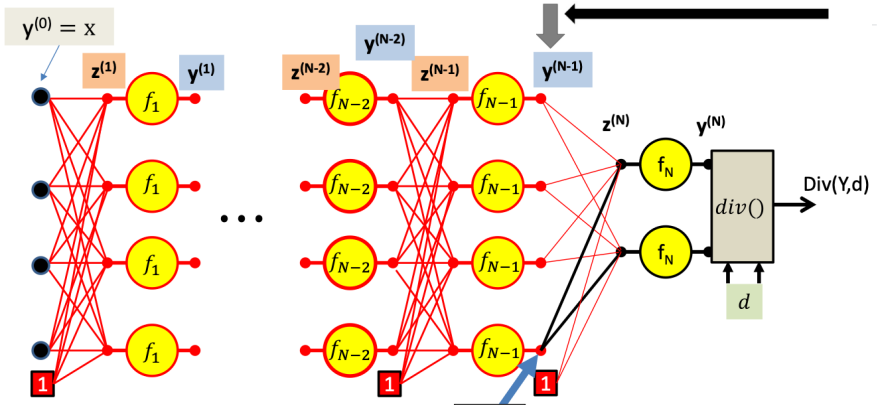
Computing derivatives



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

$y_i^{(N-1)}$ Already computed

Computing derivatives

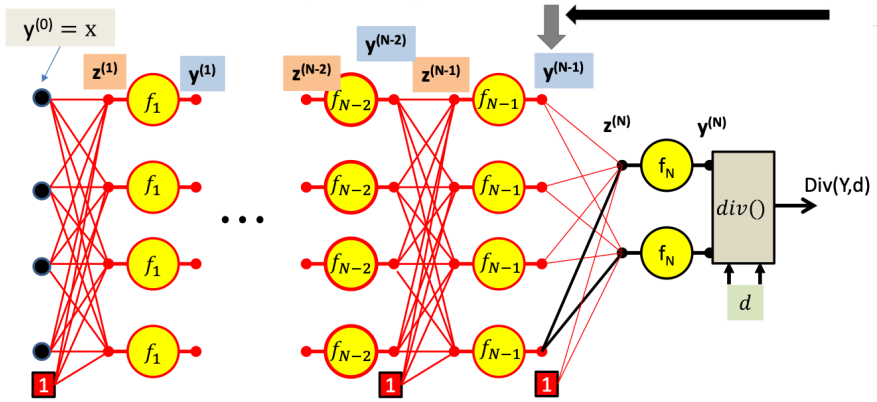


$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}}$$

$w_{ij}^{(N)}$

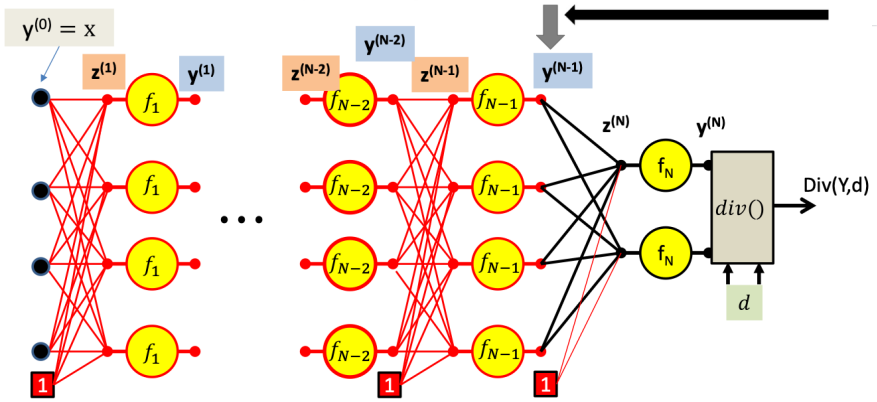
Because $z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms}$

Computing derivatives



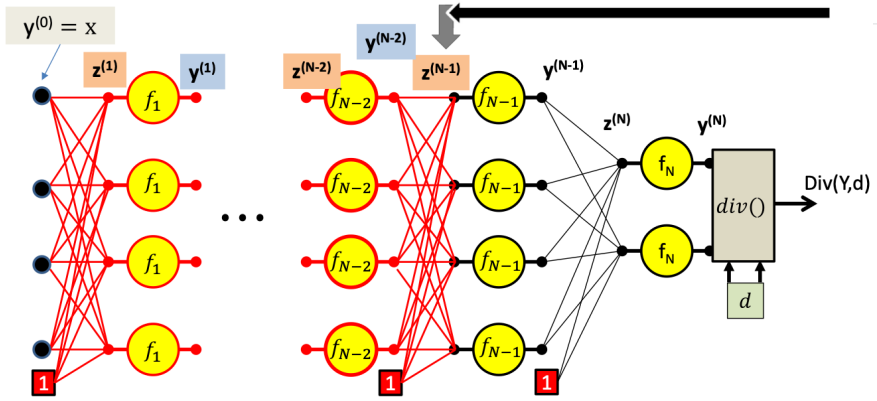
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



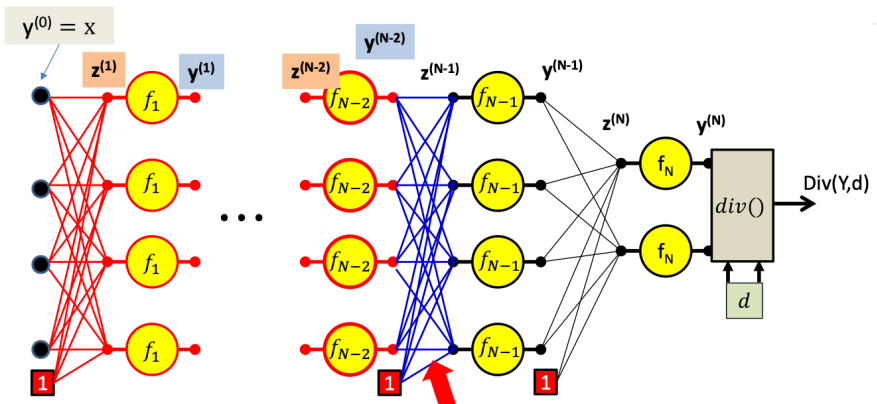
$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

Computing derivatives



We continue our way backwards in the order shown

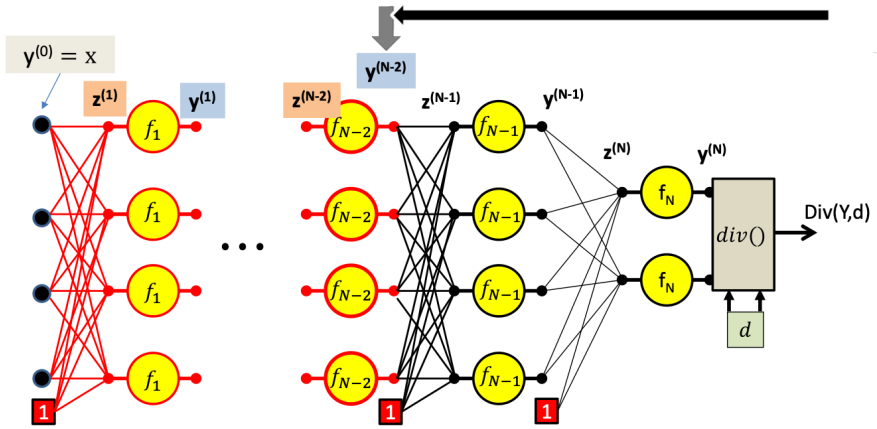
$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f'_{N-1}\left(z_i^{(N-1)}\right)\frac{\partial Div}{\partial y_i^{(N-1)}}$$



We continue our way backwards in the order shown

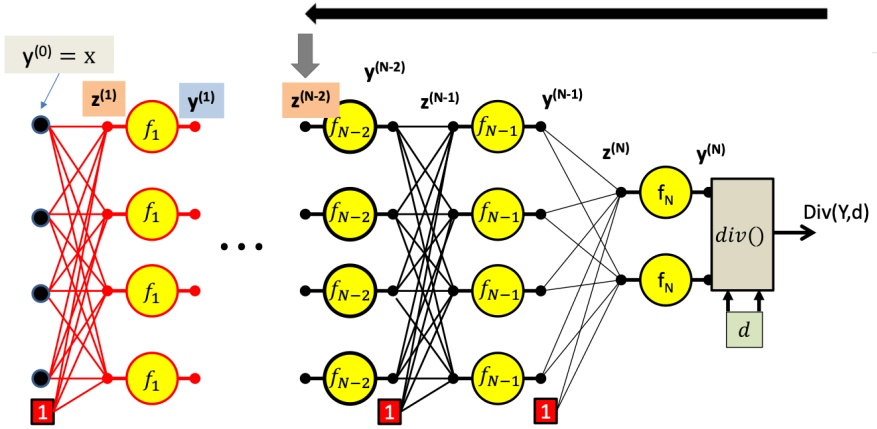
$$\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)}\frac{\partial Div}{\partial z_j^{(N-1)}}$$

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



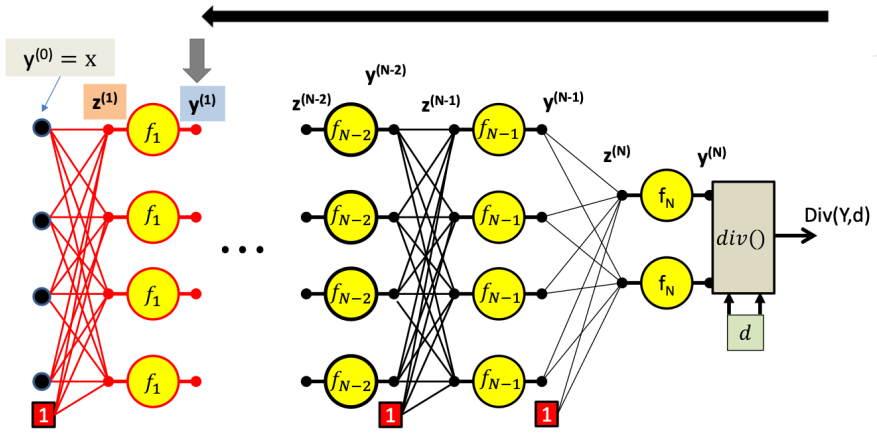
We continue our way backwards in the order shown

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



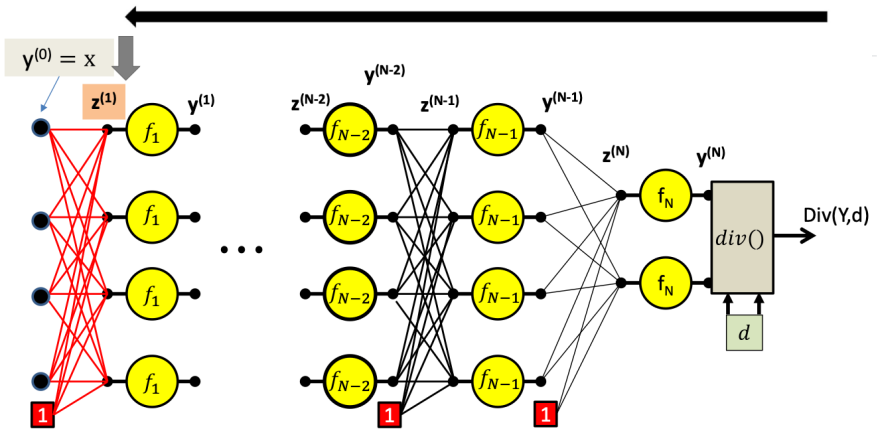
We continue our way backwards in the order shown

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



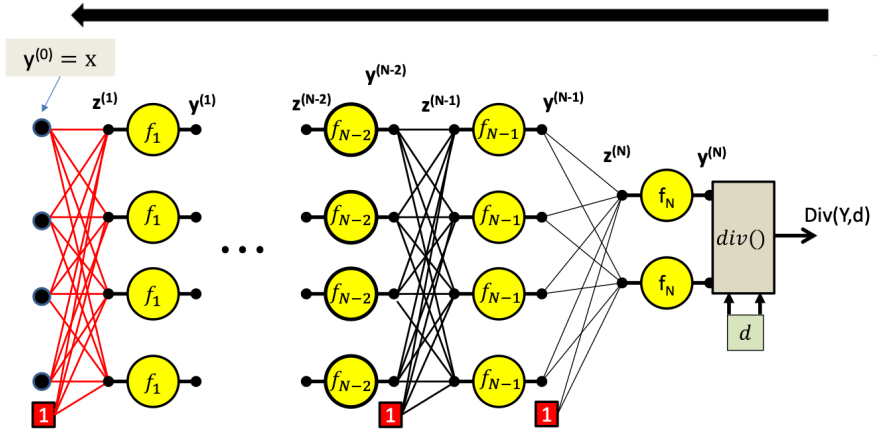
We continue our way backwards in the order shown

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



We continue our way backwards in the order shown

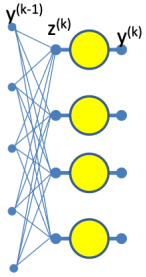
$$\frac{\partial Div}{\partial z_i^{(1)}} = f_1'\left(z_i^{(1)}\right) \frac{\partial Div}{\partial y_i^{(1)}}$$

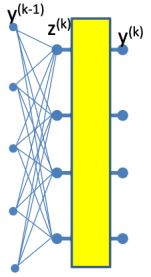


We continue our way backwards in the order shown

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

Special Case 1. Vector activations





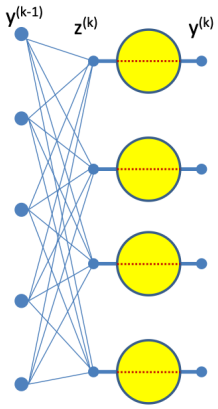
Scalar activation: Modifying a z_i only changes corresponding y_i

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

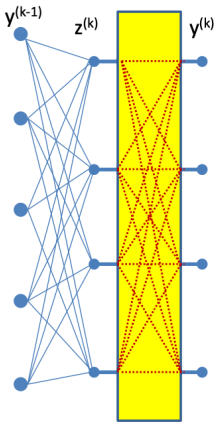
$$y_i^{(k)} = f\left(z_i^{(k)}\right)$$

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

"Influence" diagram

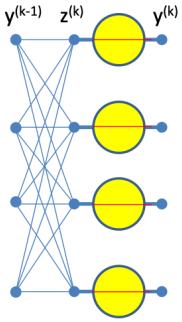


Scalar activation: Each z_i influences *one* y_i



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

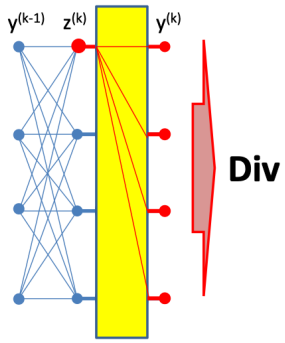
Scalar Activation: Derivative rule



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

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

Derivatives of vector activation



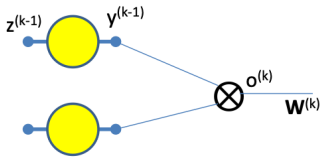
$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

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

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

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

Special Case 2: Multiplicative networks

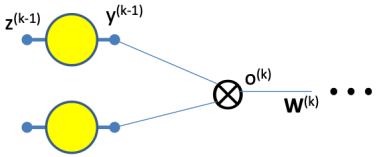


$\ldots$

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

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

Backpropagation: Multiplicative Networks



Forward:

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

Backward:

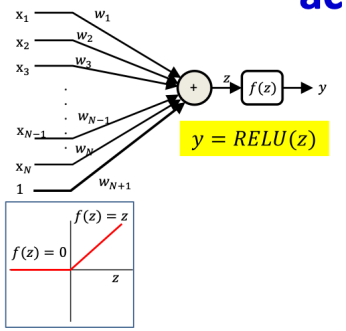
$$\frac{\partial Div}{\partial o_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$$

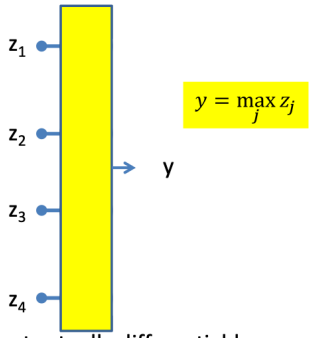
$$\frac{\partial Div}{\partial y_j^{(k-1)}} = \frac{\partial o_i^{(k)}}{\partial y_j^{(k-1)}} \frac{\partial Div}{\partial o_i^{(k)}} = y_l^{(k-1)} \frac{\partial Div}{\partial o_i^{(k)}}$$

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

- Some types of networks have *multiplicative* combination

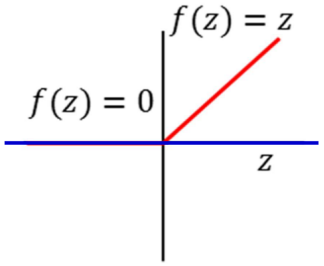
Special Case : Non-differentiable activations

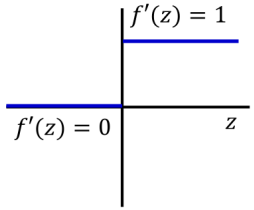




- Activation functions are sometimes not actually differentiable
  - E.g. The RELU (Rectified Linear Unit)
    - And its variants: leaky RELU, randomized leaky RELU
  - E.g. The "max" function
- Must use "subgradients" where available
  - Or "secants"

Non-differentiability: RELU



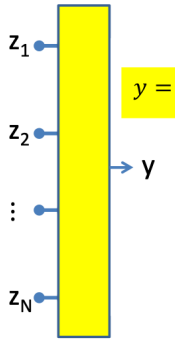


$$f'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

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

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

Subgradients and the Max



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

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

Overall Approach

- For each data instance
  - **Forward pass**: Pass instance forward through the net. Store all intermediate outputs of all computation.
  - **Backward pass**: Sweep backward through the net, iteratively compute all derivatives w.r.t weights
- Actual loss is the sum of the divergence over all training instances

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

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

$$\nabla_W \mathbf{Loss} = \frac{1}{|\{X\}|}\sum_{X} \nabla_W Div(Y(X), d(X)) \quad W \leftarrow W - \eta \nabla_W \mathbf{Loss}^{\mathrm{T}}$$

Training by BackProp

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

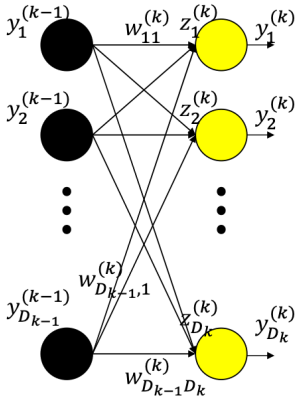
$$w_{i,j}^{(k)} = w_{i,j}^{(k)} - \frac{\eta}{T}\frac{dLoss}{dw_{i,j}^{(k)}}$$

- Until $Loss$ has converged

Vector formulation

- For layered networks it is generally simpler to think of the process in terms of vector operations
  - Simpler arithmetic
  - Fast matrix libraries make operations *much* faster
- We can restate the entire process in vector terms
  - This is what is *actually* used in any real system

Vector formulation



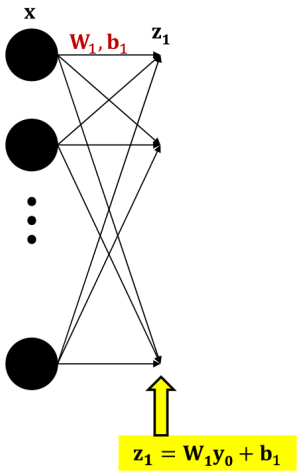
$$\mathbf{z_k} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y_k} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

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

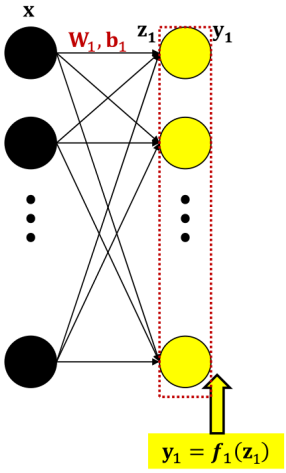
- Arrange the *inputs* to neurons of the kth layer as a vector $\mathbf{z_k}$
- Arrange the outputs of neurons in the kth layer as a vector $\mathbf{y_k}$
- Arrange the weights to any layer as a matrix $\mathbf{W_k}$
  - Similarly with biases

The forward pass: Evaluating the network

The forward pass



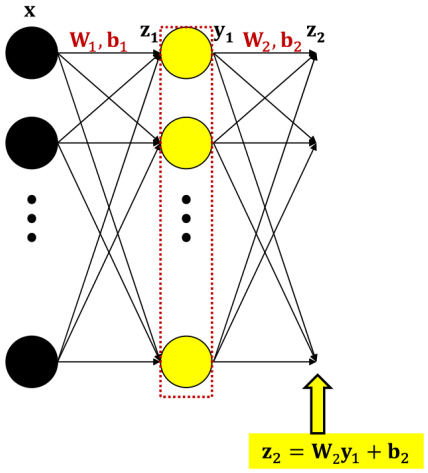
The forward pass



The Complete computation

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

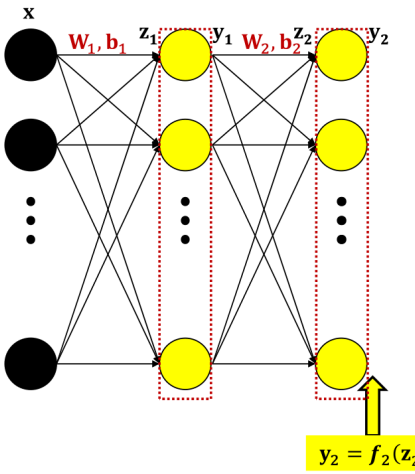
The forward pass



The Complete computation

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

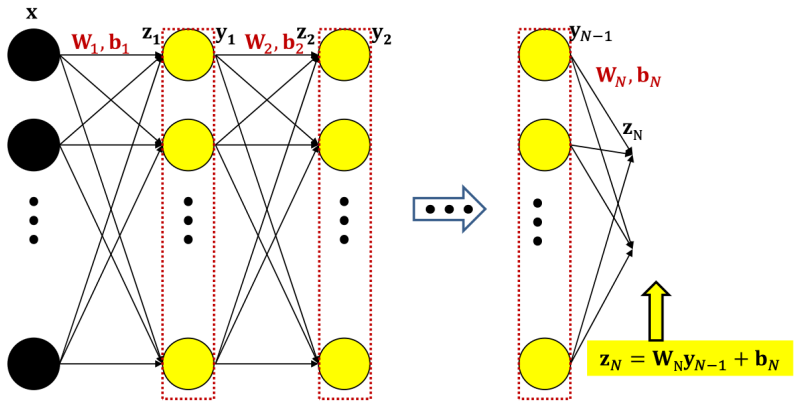
The forward pass



The Complete computation

$$y_2 = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

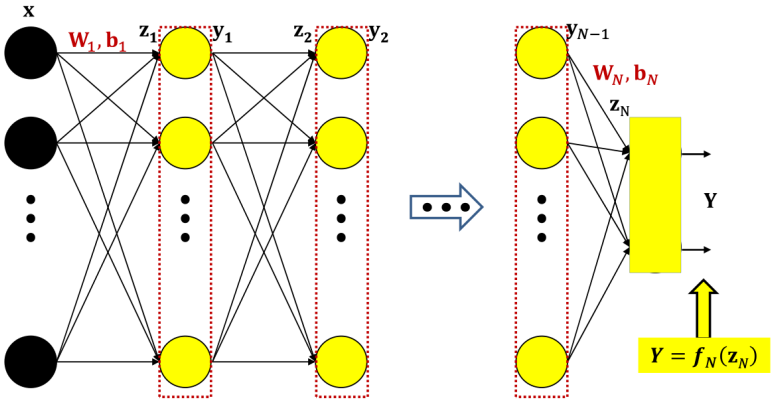
The forward pass



The Complete computation

$$z_N = W_N f_{N-1}(\ldots f_2(W_2 f_1(W_1 x + b_1) + b_2)\ \ldots) + b_N$$

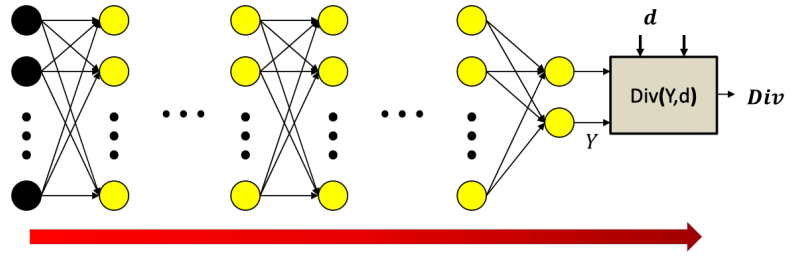
The forward pass



The Complete computation

$$Y = f_N(W_N f_{N-1}(\ldots f_2(W_2 f_1(W_1 x + b_1) + b_2)\ \ldots) + b_N)$$

Forward pass



Forward pass:

Initialize $y_0 = x$

For k = 1 to N: $z_k = W_k y_{k-1} + b_k$ $y_k = f_k(z_k)$

Output $Y = y_N$

The Backward Pass

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

Calculus Notes: The Jacobian

- The derivative of a vector function w.r.t. vector input is called a *Jacobian*
- It is the matrix of partial derivatives given below

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

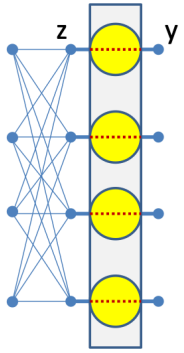
$$J_{\mathbf{y}}(\mathbf{z}) = \begin{bmatrix} \frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_D} \\ \frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_D} \\ \cdots & \cdots &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

Using vector notation

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

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

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

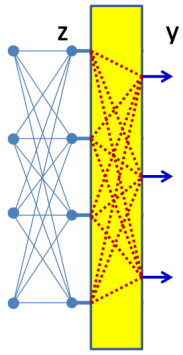


$$y_i = f(z_i)$$

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

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

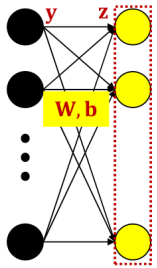
For *Vector* activations



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

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

Special case: Affine functions



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

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

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

- Matrix **W** and bias **b** operating on vector **y** to produce vector **z**
- The Jacobian of **z** w.r.t **y** is simply the matrix **W**

Vector Calculus Notes 2: Chain rule

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

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

Check

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

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

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

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

Extended Chain rule

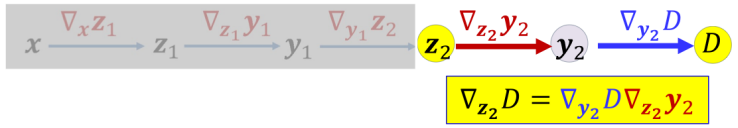
$$x \xrightarrow{\nabla_x z_1} z_1 \xrightarrow{\nabla_{z_1} y_1} y_1 \xrightarrow{\nabla_{y_1} z_2} z_2 \xrightarrow{\nabla_{z_2} y_2} y_2 \xrightarrow{\nabla_{y_2} D} D$$

How do we compute the derivative of D w.r.t. x, $\mathbf{z}_1$, y_1, $\mathbf{z}_2$ and y_2, from the local derivatives shown on the edges?

Extended Chain rule

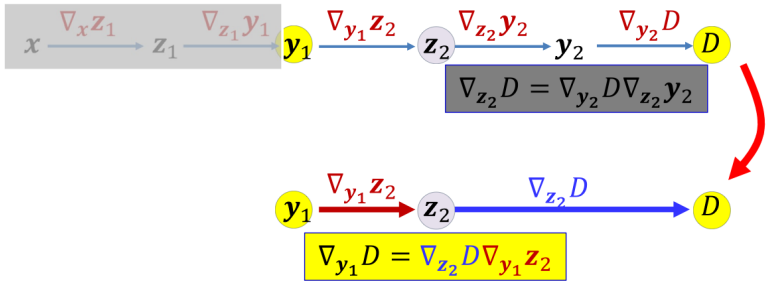
$$x \xrightarrow{\nabla_x z_1} z_1 \xrightarrow{\nabla_{z_1} y_1} y_1 \xrightarrow{\nabla_{y_1} z_2} z_2 \xrightarrow{\nabla_{z_2} y_2} y_2 \xrightarrow{\nabla_{y_2} D} D$$

Extended Chain rule



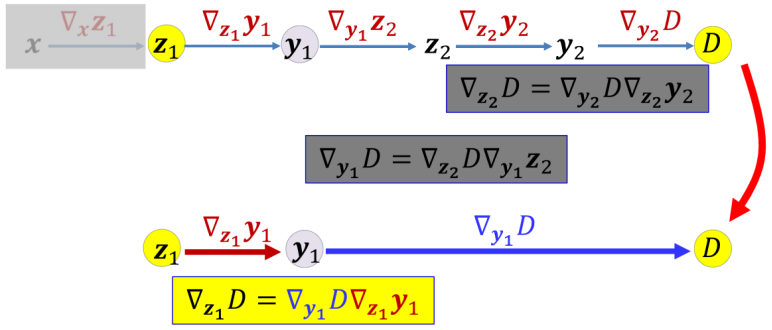
Note the order: The derivative of the outer function comes first

Extended Chain rule



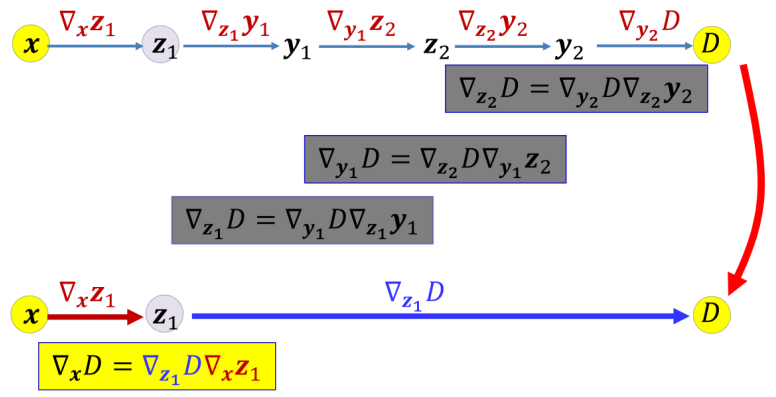
Note the order: The derivative of the outer function comes first

Extended Chain rule



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

Extended Chain rule



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

Extended Chain rule

$$x \xrightarrow{\nabla_x z_1} z_1 \xrightarrow{\nabla_{z_1} y_1} y_1 \xrightarrow{\nabla_{y_1} z_2} z_2 \xrightarrow{\nabla_{z_2} y_2} y_2 \xrightarrow{\nabla_{y_2} D} D$$

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

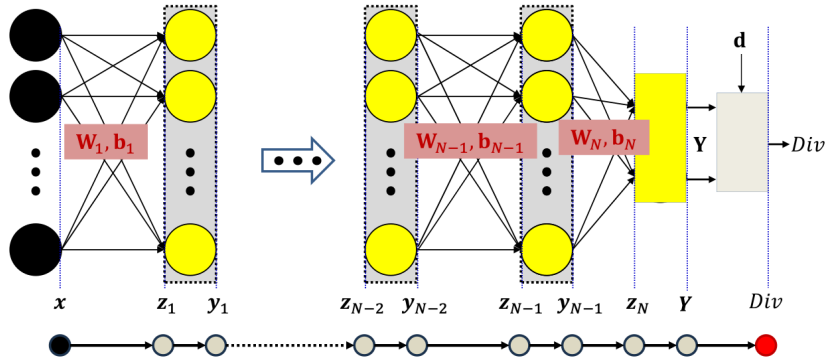
$$\nabla_{y_1} D = \nabla_{z_2} D \nabla_{y_1} z_2$$

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

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

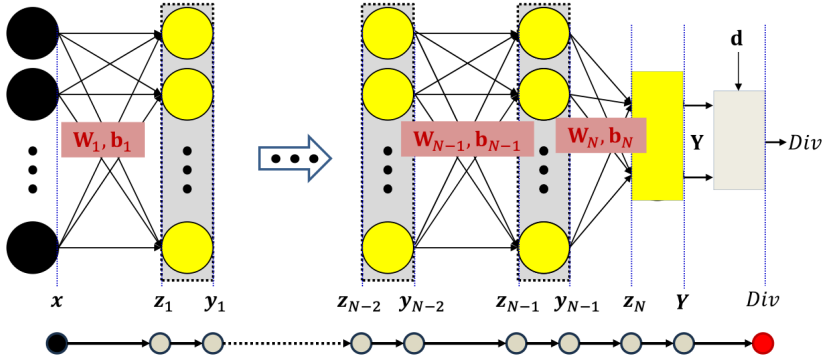
Note the order: The derivative of the outer function comes first

The backward pass



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

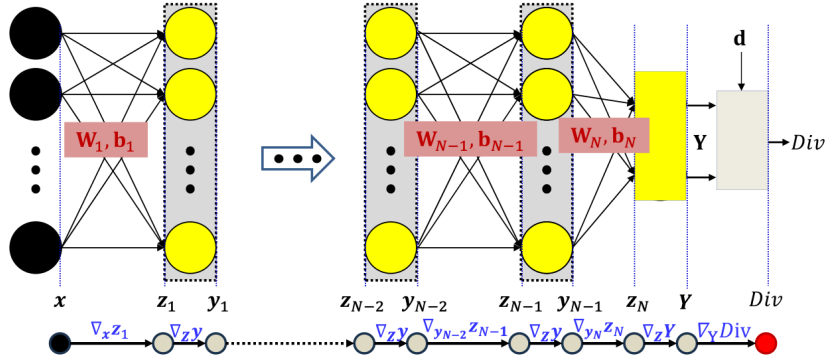
The backward pass



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

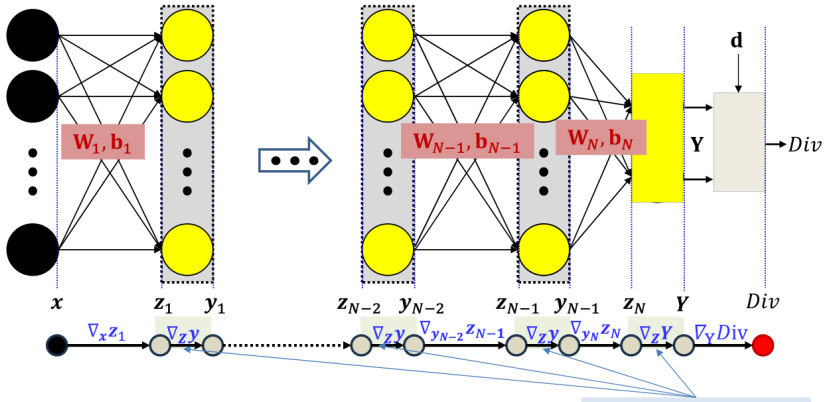
Note that for activation functions, these are actually Jacobians

The backward pass



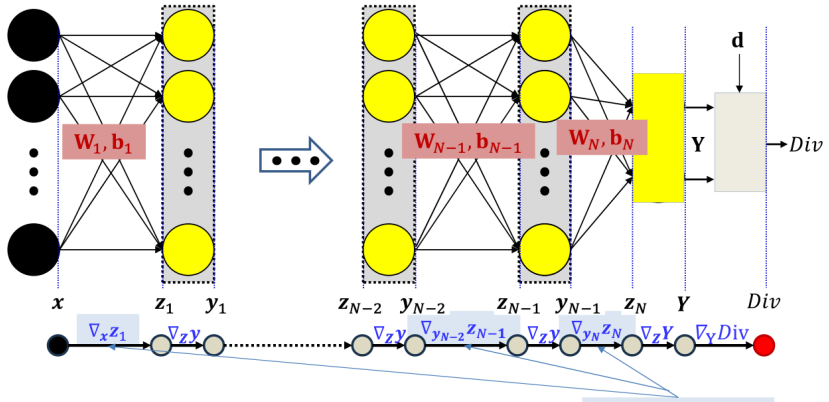
- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

The backward pass



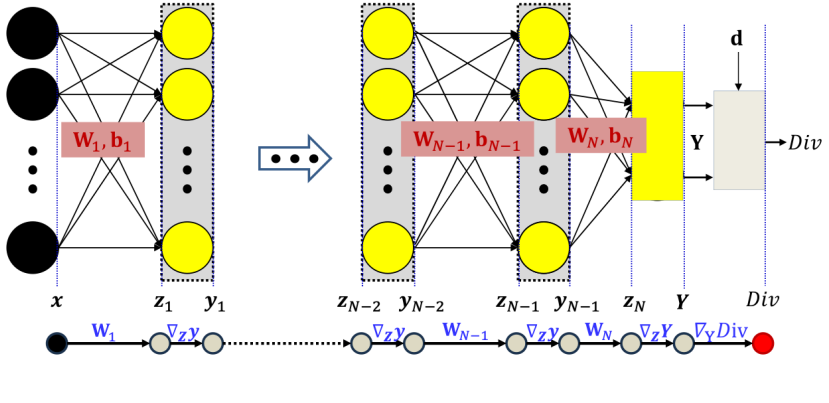
- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

The backward pass



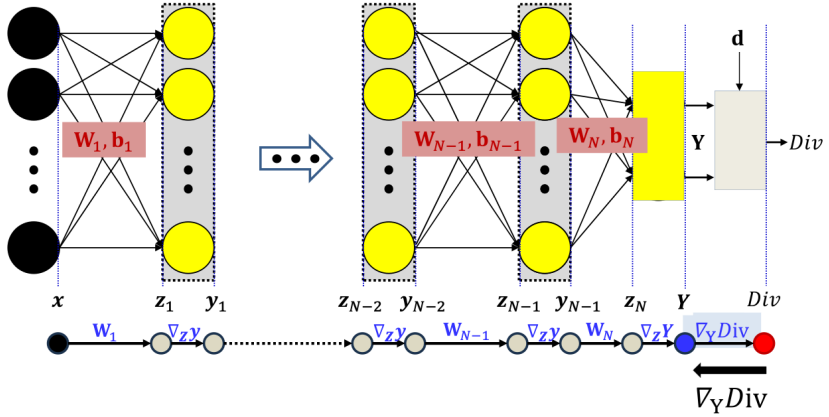
- The network again (with variables shown)…
- With the divergence we will minimize…
- And the entire influence diagram (with derivatives)
  - Variable subscripts not shown in $\nabla_z y$ for brevity

The backward pass



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

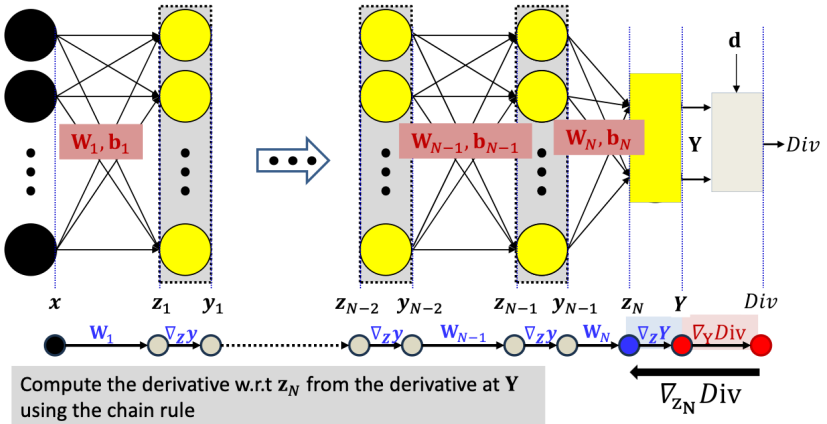
The backward pass



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

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

The backward pass

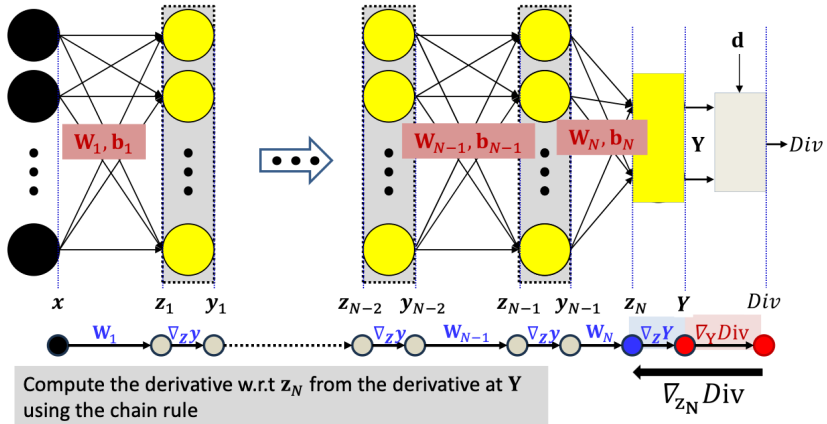


Compute the derivative w.r.t $\mathbf{z}_N$ from the derivative at $\mathbf{Y}$ using the chain rule

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

Already computed — New term

The backward pass



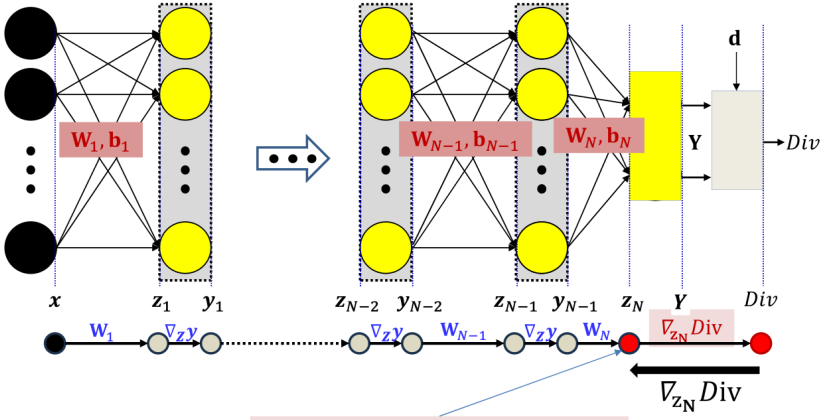
Compute the derivative w.r.t $\mathbf{z}_N$ from the derivative at $\mathbf{Y}$ using the chain rule

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

Already computed — Jacobian

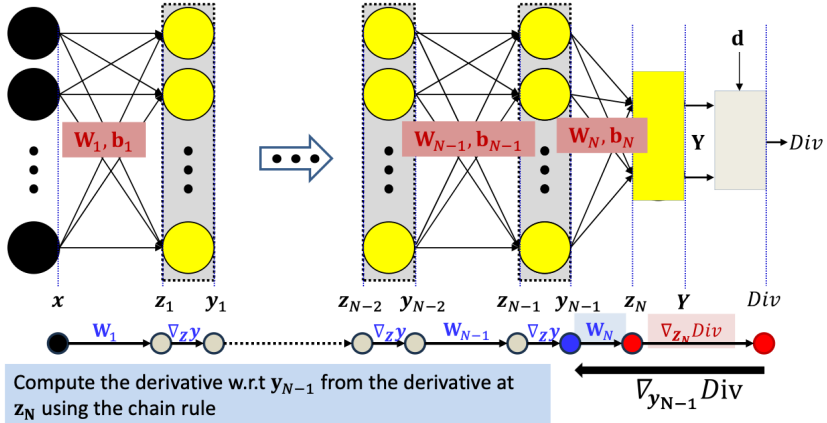
$\nabla_{\mathbf{z}_N} \mathbf{Y}$ is just the Jacobian of the activation function

The backward pass



We now have the derivative for $\mathbf{z}_N$

The backward pass

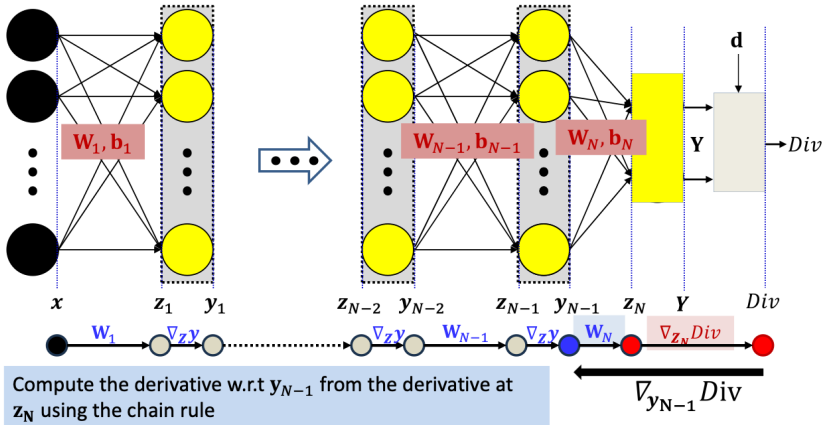


Compute the derivative w.r.t $\mathbf{y}_{N-1}$ from the derivative at $\mathbf{z}_N$ using the chain rule

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

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

The backward pass



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

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

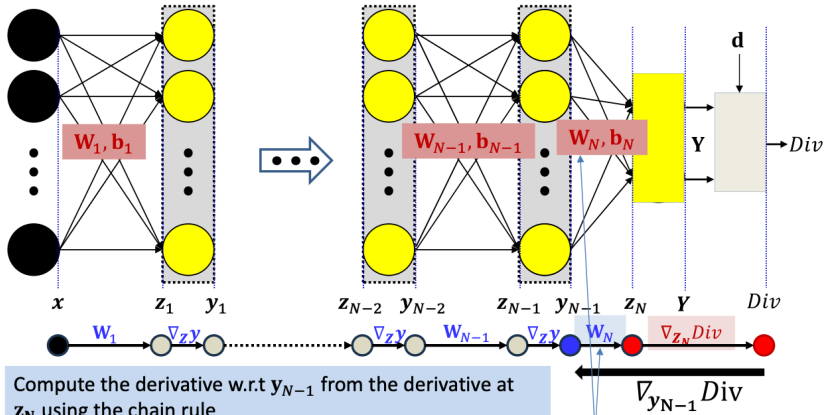
Already computed — New term

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

178

3

The backward pass



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

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

Already computed — New term

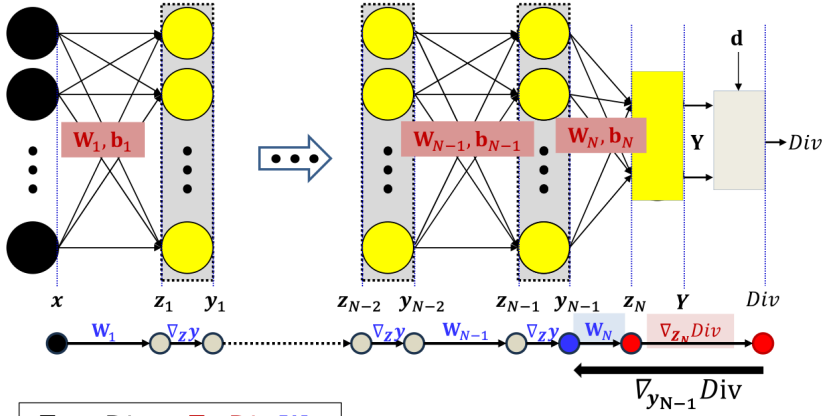
$$\mathbf{z}_N = \mathbf{W}_N \mathbf{y}_{N-1} + \mathbf{b}_N \quad \Rightarrow \quad \nabla_{\mathbf{y}_{N-1}} \mathbf{z}_N = \mathbf{W}_N$$

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

179

4

The backward pass



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

Affine parameter rules

$\mathbf{z} = \mathbf{W}\mathbf{y} + \mathbf{b}$ → $\nabla_{\mathbf{b}} Div = \nabla_{\mathbf{z}} Div$, $\nabla_{\mathbf{W}} Div = \mathbf{y} \nabla_{\mathbf{z}} Div$

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

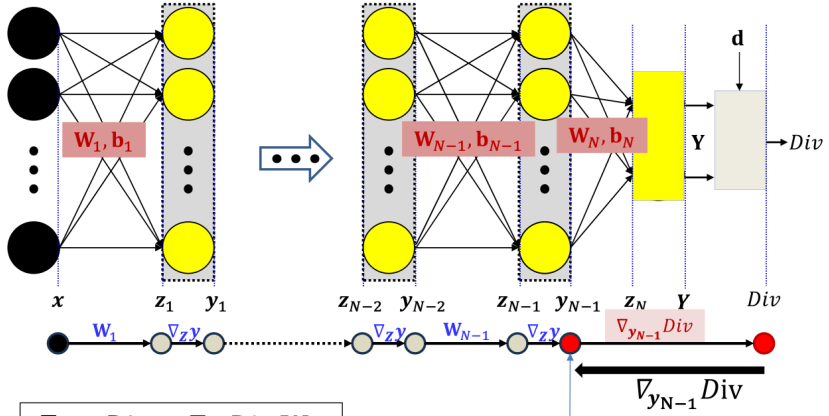
$$\nabla_{\mathbf{W}_N} Div = \mathbf{y}_{N-1} \nabla_{\mathbf{z}_N} Div$$

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

180

5

The backward pass



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

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

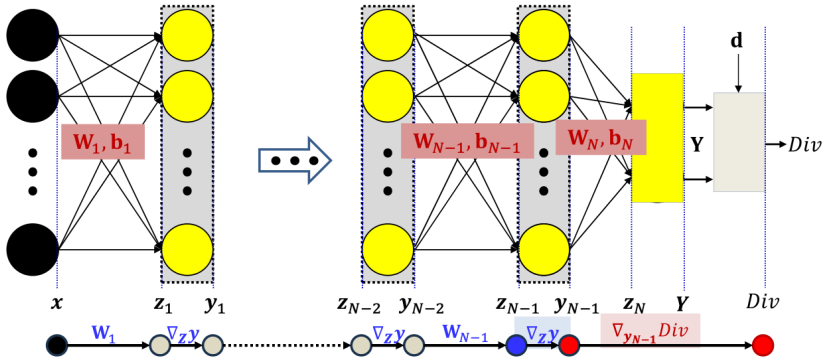
$$\nabla_{\mathbf{b}_N} Div = \nabla_{\mathbf{z}_N} Div$$

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

181

6

The backward pass

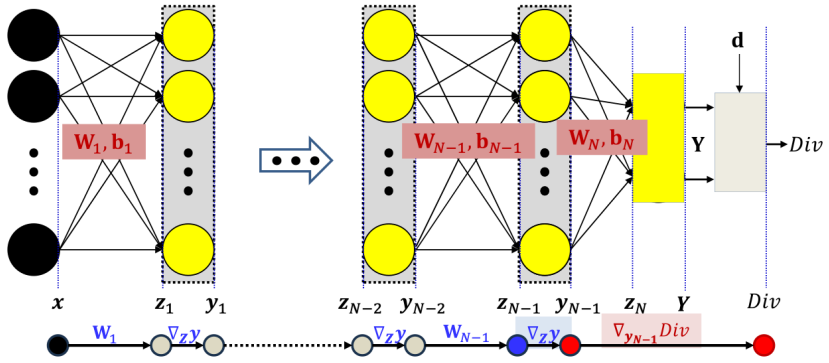


Compute the derivative w.r.t $\mathbf{z}_{N-1}$ from the derivative at $\mathbf{y}_{N-1}$ using the chain rule

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

Already computed — New term

The backward pass



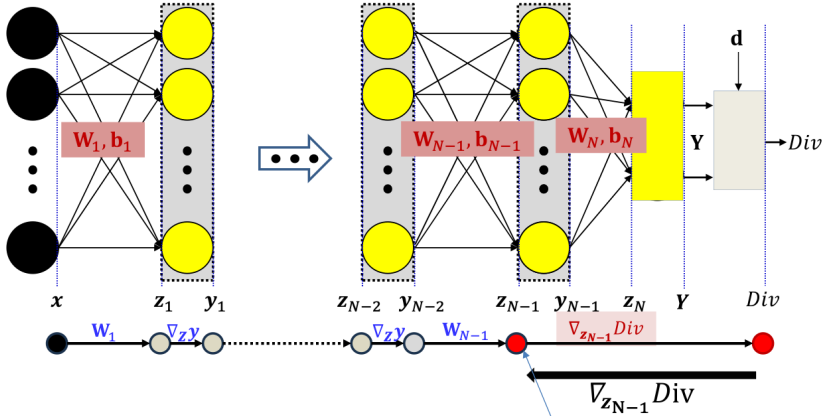
Compute the derivative w.r.t $\mathbf{z}_{N-1}$ from the derivative at $\mathbf{y}_{N-1}$ using the chain rule

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

Already computed — Jacobian

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

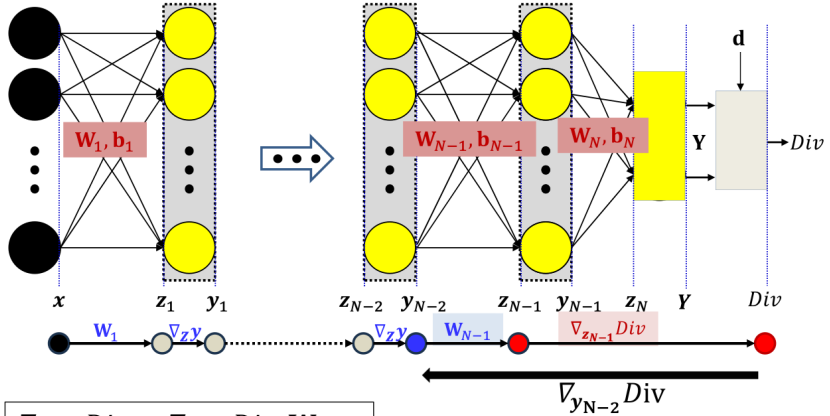
The backward pass



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

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

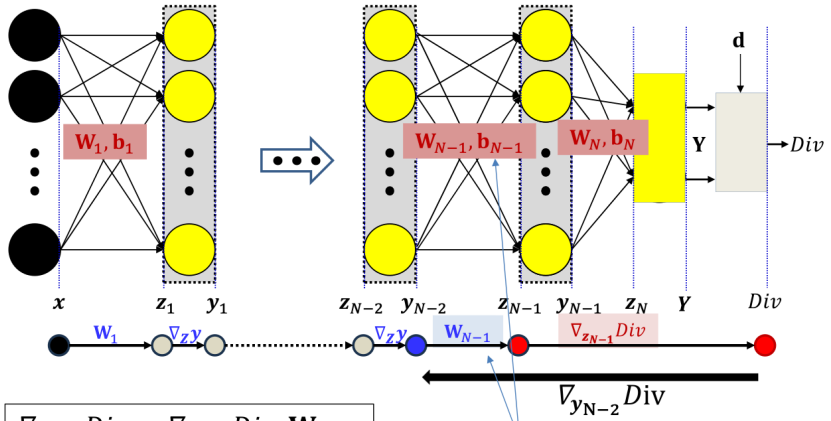
The backward pass



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

Already computed — New

The backward pass



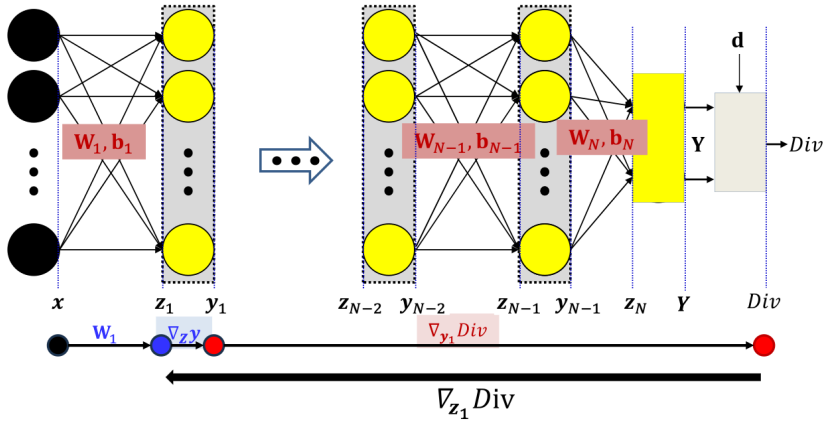
$$\nabla_{\mathbf{y}_{N-2}}Div = \nabla_{\mathbf{z}_{N-1}}Div.\mathbf{W}_{N-1}$$

Already computed New

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

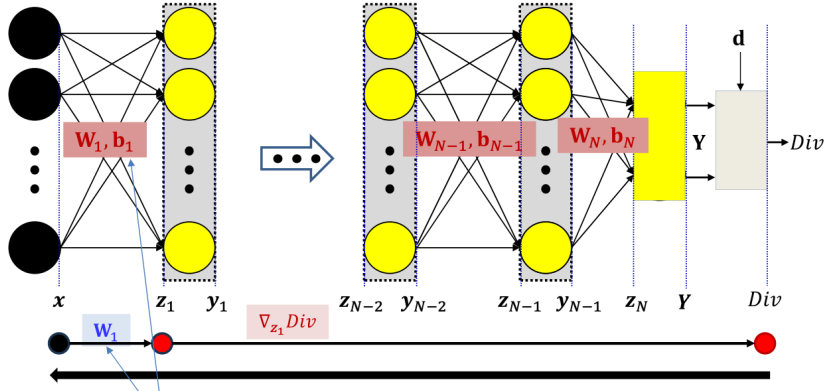
$$\nabla_{\mathbf{b}_{N-1}}Div = \nabla_{\mathbf{z}_{N-1}}Div$$

The backward pass



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

The backward pass



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

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

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

Neural network training algorithm

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

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

- Until $Loss$ has converged

Issues

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