Backpropagation with Tensors

AI - Winter 2020

Basic Definitions: What is a Tensor?

(11)

SCALAR

5	3	7

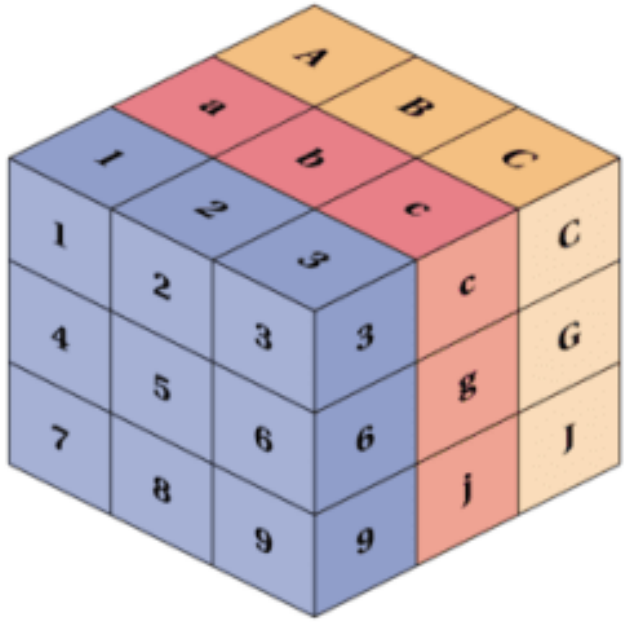
Row Vector
(shape 1x3)

5
1.5
2

Column Vector
(shape 3x1)

$$\begin{bmatrix} 4 & 19 & 8 \\ 16 & 3 & 5 \end{bmatrix}$$

MATRIX



TENSOR

Scaler in – Scaler out

$$f : \mathbb{R} \to \mathbb{R} \qquad f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$y = f(x)$$

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x} \Delta x$$

Vector in – Scaler Out (Gradient)

$$f : \mathbb{R}^N \to \mathbb{R} \qquad \nabla_x f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{\|h\|}$$

$$y = f(x)$$

$$\frac{\partial y}{\partial x} = \left(\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_N} \right)$$

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x} \cdot \Delta x$$

Vector in – Vector Out (Jacobian)

$$f : \mathbb{R}^N \to \mathbb{R}^M$$

$$y = f(x)$$

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{pmatrix}$$

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x}\Delta x$$

Tensor in – Tensor out (Generalized Jacobian)

$$f : \mathbb{R}^{N_1 \times \cdots \times N_{D_x}} \rightarrow \mathbb{R}^{M_1 \times \cdots \times M_{D_y}}$$

$$y = f(x)$$

$\frac{\partial y}{\partial x}$ is a *generalized Jacobian*, which is an object with shape

$$(M_1 \times \cdots \times M_{D_y}) \times (N_1 \times \cdots \times N_{D_x})$$

Confused?!!!! Just imagine it as a matrix which each of its rows has the same shape as y and each of its columns has a save shape of x.

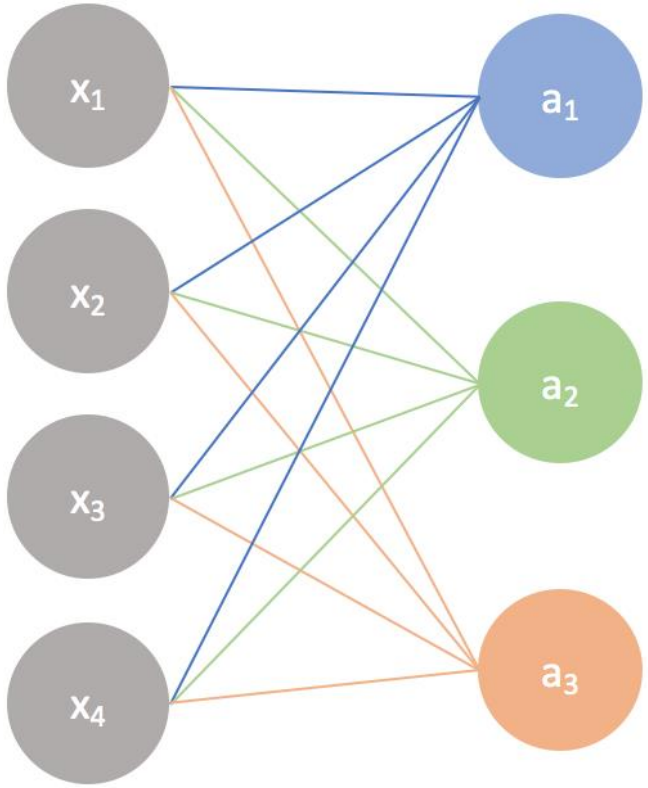
Confused?!!!! Just imagine it as a matrix which each of its rows has the same shape as y and each of its columns has a save shape of x.

Now if we let $i \in \mathbb{Z}^{D_y}$ and $j \in \mathbb{Z}^{D_x}$ be vectors of integer indices, then we can write

$$\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_i}{\partial x_j}$$

In this equation note that y_i and x_j are scalars, so the derivative $\frac{\partial y_i}{\partial x_j}$ is also a scalar.

Weights and Biases in Matrix Form



$$x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b = a_1$$
$$x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b = a_1$$
$$x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b = a_1$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} w_1 & w_1 & w_1 \\ w_2 & w_2 & w_2 \\ w_3 & w_3 & w_3 \\ w_4 & w_4 & w_4 \end{bmatrix} + \begin{bmatrix} b & b & b \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$$

Notation

In a single layer:

- $\underline{x}$ is the input vector of the layer
- $\underline{\underline{W}}$ is the weight matrix
- $\underline{b}$ is the bias vector
- f is the activation function
- y is the output vector

We can write:

$$a = xW + b \quad , \quad y = f(a)$$

Notation

In a feedforward network:

- $\underline{x_i}$ is the input vector of the i'th layer
- $\underline{\underline{W_i}}$ is the weight matrix of the i'th layer
- $\underline{b_i}$ is the bias vector of the i'th layer
- $\underline{f_i}$ is the activation function of the i'th layer
- y_i is the output vector of the i'th layer
- $\mathcal{L}$ is the loss function
- L is the output of the loss function

For a n-layer network:

$$a_i = x_i W_i + b_i \quad , \qquad y_i = f(a_i) \qquad i \in \{1, \dots, n\}$$

$$x_i = y_{i-1} \qquad i \in \{2, \dots, n\}$$

$$L = \mathcal{L}(y_n)$$

Backpropagation in Matrix Form

Our goal is to find:

$$\frac{\partial L}{\partial W_i}, \frac{\partial L}{\partial b_i}, \frac{\partial L}{\partial x_i} \qquad i \in \{1, \dots, n\}$$

So that we can update the parameters of the network using SGD algorithm.

Using the chain rule we can write:

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial W_i}$$

We cannot directly use the generalized Jacobian in computations as it is usually very big !!!

Consider a single layer with input of size $N \times D$ and output of size $N \times M$. The size of the Jacobian tensor is $(N \times D) \times (N \times M)$.

For $N = 64$ and $M = D = 4096$ and while using 32-bit floating point, the Jacobian occupies 256GB of memory!!!

Solution????

Most of the elements of the Jacobian are zero! This means that we are not supposed to compute the whole Generalize Jacobian!

Lets say N=1, D=2, and M=3 for an arbitrary layer of the network. Also lets assume that there is no activation function for simplicity.

$$
\begin{aligned}
y &= \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} \end{pmatrix} = xw \\
&= \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} \\
&= \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} \\ x_{1,1}w_{1,2} + x_{1,2}w_{2,2} \\ x_{1,1}w_{1,3} + x_{1,2}w_{2,3} \end{pmatrix}^T
\end{aligned}
$$

$$
\frac{\partial L}{\partial y} = \begin{pmatrix} dy_{1,1} & dy_{1,2} & dy_{1,3} \end{pmatrix}
\qquad
\frac{\partial L}{\partial x} = \begin{pmatrix} \frac{\partial L}{\partial x_{1,1}} & \frac{\partial L}{\partial x_{1,2}} \end{pmatrix}
$$

$$\frac{\partial L}{\partial x_{1,1}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_{1,1}}$$

$$= \begin{pmatrix} dy_{1,1} & dy_{1,2} & dy_{1,3} \end{pmatrix} \begin{pmatrix} \frac{\partial y_{1,1}}{\partial x_{1,1}} & \frac{\partial y_{1,2}}{\partial x_{1,1}} & \frac{\partial y_{1,3}}{\partial x_{1,1}} \end{pmatrix}$$

$$= \begin{pmatrix} dy_{1,1} & dy_{1,2} & dy_{1,3} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{pmatrix}$$

$$= dy_{1,1}w_{1,1} + dy_{1,2}w_{1,2} + dy_{1,3}w_{1,3}$$

$$\frac{\partial L}{\partial x} = \begin{pmatrix} \frac{\partial L}{\partial x_{1,1}} & \frac{\partial L}{\partial x_{1,2}} \end{pmatrix}$$

$$= \begin{pmatrix} dy_{1,1}w_{1,1} + dy_{1,2}w_{1,2} + dy_{1,3}w_{1,3} \\ dy_{1,1}w_{2,1} + dy_{1,2}w_{2,2} + dy_{1,3}w_{2,3} \end{pmatrix}^T$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} W^T$$

$$y = \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} \end{pmatrix} = xw$$

$$= \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} = \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} \\ x_{1,1}w_{1,2} + x_{1,2}w_{2,2} \\ x_{1,1}w_{1,3} + x_{1,2}w_{2,3} \end{pmatrix}^T$$

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \frac{\partial L}{\partial w_{1,3}} \\ \frac{\partial L}{\partial w_{2,1}} & \frac{\partial L}{\partial w_{2,2}} & \frac{\partial L}{\partial w_{2,3}} \end{bmatrix}$$

$$\frac{\partial L}{\partial w_{1,1}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{1,1}} = \left(dy_{1,1}, dy_{1,2}, dy_{1,3}\right)\left(\frac{\partial y_{1,1}}{\partial w_{1,1}}, \frac{\partial y_{1,2}}{\partial w_{1,1}}, \frac{\partial y_{1,3}}{\partial w_{1,1}}\right)$$

$$= \left(dy_{1,1}, dy_{1,2}, dy_{1,3}\right)\left(x_{1,1}, 0, 0\right) = dy_{1,1}\, x_{1,1}$$

$$\frac{\partial L}{\partial W} = \begin{bmatrix} x_{1,1}dy_{1,1} & x_{1,1}dy_{1,2} & x_{1,1}dy_{1,3} \\ x_{1,2}dy_{1,1} & x_{1,2}dy_{1,2} & x_{1,2}dy_{1,3} \end{bmatrix}$$

$$= \left(x_{1,1}, x_{1,2}\right)^T \left(dy_{1,1}, dy_{1,2}, dy_{1,3}\right)$$

$$\boxed{\frac{\partial L}{\partial W} = x^T \frac{\partial L}{\partial y}}$$

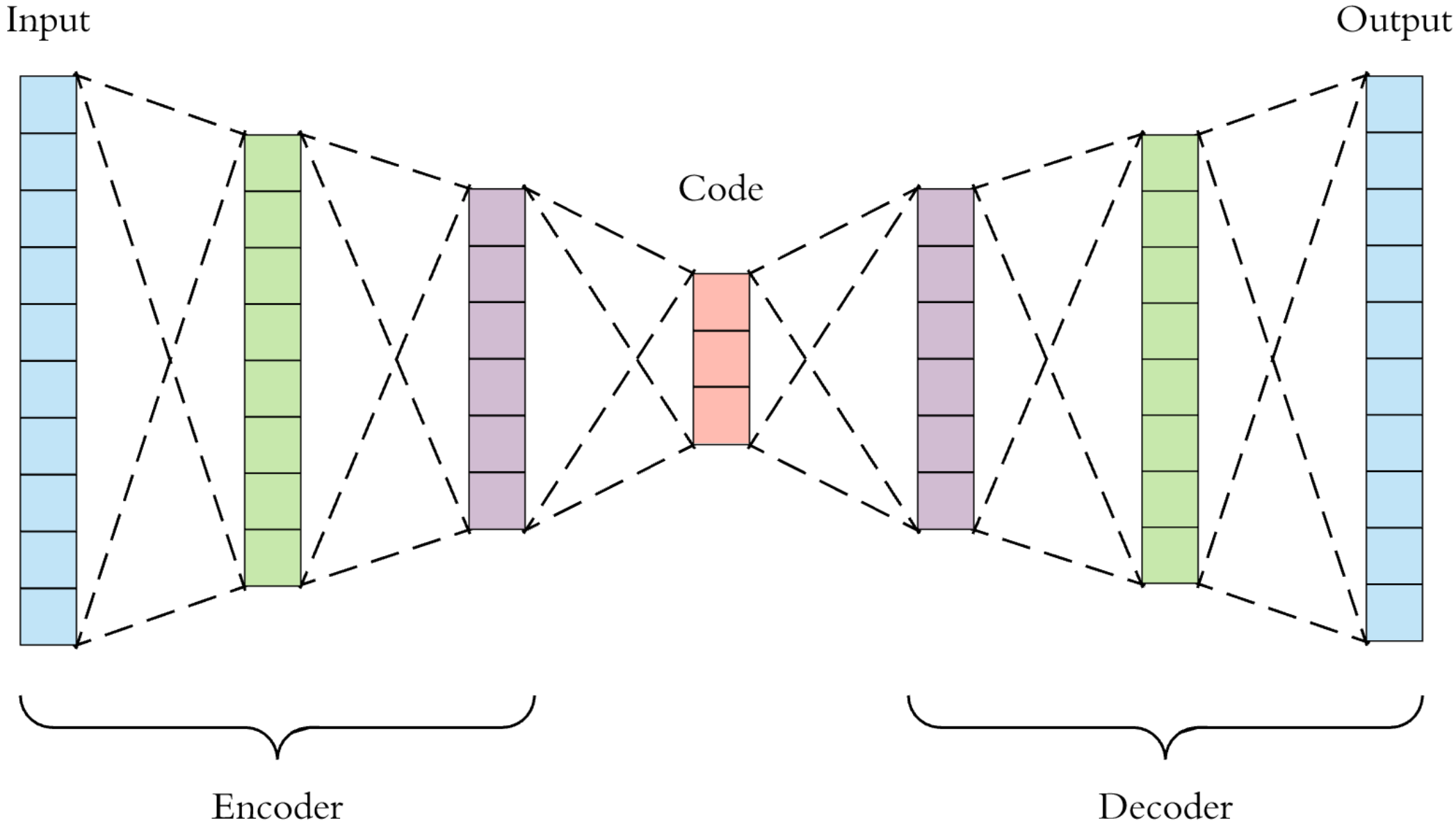
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} W^T \qquad\qquad \frac{\partial L}{\partial W} = x^T \frac{\partial L}{\partial y}$$

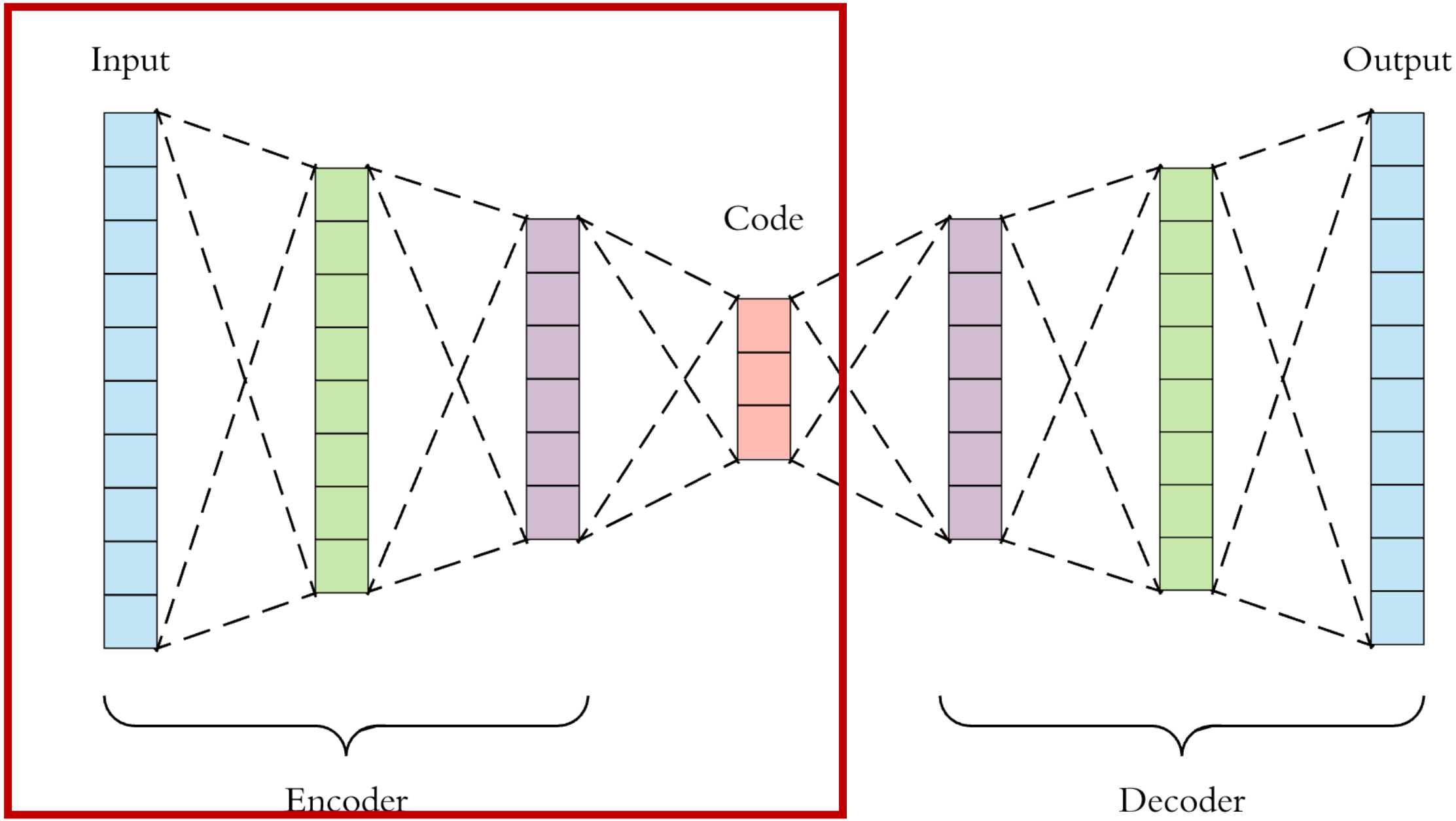
In a network with n layers:

$$\frac{\partial L}{\partial W_i} = x_i^T \frac{\partial L}{\partial y_i} \quad , \quad i \in \{1, \dots, n\}$$

$$\frac{\partial L}{\partial y_n} \; can\ be\ computed\ directly$$

$$\frac{\partial L}{\partial y_{i-1}} = \frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_i} W_i^T \quad , \quad i \in \{1, \dots, n-1\}$$


Input
Output
Code
Encoder
Decoder


Input
Output
Code
Encoder
Decoder