Exercise 1: Convolutional Neural Networks (10 P)

Let $x = (x^{(i)})$, be a multivariate time series represented as a collection of one-dimensional signals. For simplicity of the following derivations, we consider these signals to be of infinite length. We feed x as input to a convolutional layer. The forward computation in this layer is given by:

$$z_t^{(j)} = \sum_i \left[w^{(ij)} * x^{(i)} \right]_t$$
$$= \sum_i \sum_{k=-\infty}^{\infty} w_k^{(ij)} \cdot x_{t-k}^{(i)} \qquad \text{for all } t \text{ and } t \in \mathbb{Z}$$

It results in $z = (z^{(j)})$, another multivariate time series again composed of a collection of output signals also assumed to be of infinite length. Convolution filters $w^{(ij)}$ have to be learned from the data. After passing the data through the convolutional layer, the neural network output is given as $y = \ell(z)$ where ℓ is some top-layer function assumed to be differentiable. To learn the model, parameters gradients need to be computed.

(a) *Express the gradient $\partial y / \partial w$ as a function of the input x and of the gradient $\partial y / \partial z$.*

$$\frac{\partial y}{\partial w_k^{(ij)}} = \sum_{j'} \sum_{t'} \frac{\partial y_{t'}^{(j')}}{\partial z_{t'}^{(j')}} \frac{\partial z_{t'}^{(j')}}{\partial w_k^{(ij)}} = \sum_{j'} \sum_{t'} \frac{\partial y}{\partial z_{t'}^{(j')}} \frac{\partial}{\partial w_k^{(ij)}} \sum_{i'} \sum_{k'} w_{k'}^{(i'j')} \cdot x_{t'-k'}^{(i')}$$
$$= \sum_{j'} \sum_{t'} \frac{\partial y}{\partial z_{t'}^{(j')}} \sum_{i'} \delta_{(i,i')} \delta_{(k,k')} x_{t'-k'}^{(i')} = \sum_{j'} \sum_{t'} \frac{\partial y}{\partial z_{t'}^{(j')}} \cdot x_{t'-k}^{(i)}$$
$$= \sum_{j'} \left[\frac{\partial y}{\partial z^{(j')}} * x^{(i)} \right]_k$$

$$\frac{\partial y}{\partial w^{(ij)}} = \sum_{j'} \frac{\partial y}{\partial z^{(j')}} * x^{(i)}$$

Exercise 2: Recursive Neural Networks (10 + 10 P)

Consider a recursive neural network that applies some function ϕ recursively from the leaves to the root of a binary parse tree. The function $\phi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^d$ takes two incoming nodes $a, b \in \mathbb{R}^d$ and some parameter vector $\theta \in \mathbb{R}^p$ as input, and produces an output $z_h \in \mathbb{R}^d$. Once we have reached the root node, we apply a function $\ell : \mathbb{R}^d \to \mathbb{R}$ that produces some real-valued prediction. We assume that both ϕ and ℓ are differentiable with their inputs.

Consider the sentence 'the cat sat on the mat', that is parsed as:

$$((\text{the}, \text{cat}), (\text{sat}, (\text{on}, (\text{the}, \text{mat}))))$$

The leaf nodes of the parsing tree are represented by word embeddings $z_{\text{the}}, z_{\text{cat}}, \ldots \in \mathbb{R}^d$.

(a) *Draw the computational graph associated to the application of the recursive neural network to this sentence, and write down the set of equations, e.g.*

$$z_1 = \phi(z_{\text{the}}, z_{\text{cat}}, \theta)$$
$$z_2 = \phi(z_{\text{the}}, z_{\text{mat}}, \theta)$$
$$z_3 = \phi(z_{\text{sat}}, z_2, \theta)$$
$$\vdots$$
$$y = \ell(z_h)$$

(b) *Express the total derivative $dy/d\theta$ (taking into account all direct and indirect dependencies on the parameter θ) in terms of local derivatives (relating adjacent quantities in the computational graph).*

(Hint: you can use for this the chain rule for derivatives that states that for some function $h(g_1(t), \ldots, g_N(t))$ we have:

$$\frac{dh}{dt} = \frac{\partial h}{\partial g_1} \cdot \frac{dg_1}{dt} + \cdots + \frac{\partial h}{\partial g_N} \cdot \frac{dg_N}{dt}$$

where $d(\cdot)$ and $\partial(\cdot)$ denote the total and partial derivatives respectively.)

$$\frac{dy}{d\theta} = \frac{\partial y}{\partial \theta} \left[\frac{\partial z_h}{\partial z_1} \frac{\partial z_1}{\partial \theta} + \frac{\partial z_h}{\partial z_2} \frac{dz_2}{d\theta} \right]$$
$$\frac{\partial z_2}{\partial \theta} = \frac{\partial z_2}{\partial z_3} \frac{dz_3}{d\theta}$$
$$\frac{dz_3}{d\theta} = \frac{\partial z_3}{\partial z_4} \frac{\partial z_4}{\partial \theta}$$

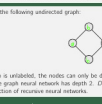
Exercise 3: Graph Neural Networks (10 + 10 P)

Graph neural networks are a fairly flexible class of neural networks that can sometimes be seen as a generalization of convolutional neural networks and recursive neural networks. We would like to study the equivalence of graph neural network with other neural networks. We will consider in this exercise graph neural network that map two consecutive layers using the equation:

$$\boldsymbol{H}_{l+1} = \rho(\hat{\boldsymbol{A}}\boldsymbol{H}_l\boldsymbol{W})$$

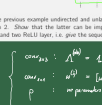
where ρ denotes the ReLU function. The adjacency matrix $\boldsymbol{A}$ is of size $N \times N$ with value 1 when there is a connection, and 0 otherwise. The matrix of parameters $\boldsymbol{W}$ is of size $d \times d$, where d is the number of dimensions used to represent each node. We also assume that the initial state $\boldsymbol{H}_0$ is a matrix filled with ones.

(a) *Consider first the following undirected graph:*



Because the graph is unlabeled, the nodes can only be distinguished by their connectivity to other nodes. Consider the case where the graph neural network has depth 2. Depict the multiple trees formed by viewing the graph neural network as a collection of recursive neural networks.

(b) *Consider now the following infinite lattice graph:*



which is like in the previous example undirected and unlabeled. We consider again the case where the graph neural network has depth 2. Show that the latter can be implemented as a 2D convolutional neural network with four convolution layers and two ReLU layer, i.e. give the sequence of layers and their parameters.

$$2\times \begin{cases} \text{conv}_{3\times3} : \Lambda^{(0)} = I_{3\times3} + \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ \text{conv}_{1\times1} : W^{(l)} = [w_{ij}] \\ \rho : \text{no parameter} \end{cases}$$