# ADVANCED LEARNING FOR TEXT AND GRAPH DATA

# Lab session 7: Learning on Sets / Influence Maximization

Lecture: Prof. Michalis Vazirgiannis
Lab: Giannis Nikolentzos, George Panagopoulos

Tuesday, January 19, 2021

---

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit here a **.zip** file (max 10MB in size) containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet named `firstname_lastname.pdf`, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is January 26, 2021 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late $\rightarrow$ -5 pts; $]24, 48]$ hours late $\rightarrow$ -10 pts; $> 48$ hours late $\rightarrow$ not graded (zero).

---

## 1 Learning objective

The goal of this lab is to introduce you to machine learning models for data represented as sets. Furthermore, you will be introduced to the influence maximization problem. Specifically, in the first part of the lab, we will implement the DeepSets model. We will evaluate the model in the task of computing the sum of sets of digits and compare it against traditional models such as LSTMs. In the second part of the lab, we will implement a greedy algorithm and use it to identify a set of influential users in a social network. We will use Python 3.6, and the following two libraries: (1) PyTorch (`https://pytorch.org/`), and (2) NetworkX (`http://networkx.github.io/`).

## 2 DeepSets

Typical machine learning algorithms, such as the Logistic Regression classifier or Multi-layer Perceptrons, are designed for fixed dimensional data samples. Thus, these models cannot handle input data that takes the form of sets. The cardinalities of the sets are not fixed, but they are allowed to vary. Therefore, some sets are potentially larger in terms of the number of elements than others. Furthermore, a model designed for data represented as sets needs to be invariant to the permutations of the elements of the input sets. Formally, it is well-known that a function $f$ transforms its domain $\mathcal{X}$ into its range $\mathcal{Y}$. If the input is a set $X = \{x_1, \ldots, x_M\}$, $x_m \in \mathfrak{X}$, i.e., the input domain is the power set $\mathcal{X} = 2^{\mathfrak{X}}$, and a function $f : 2^{\mathfrak{X}} \rightarrow Y$ acting on sets must be permutation invariant to the order of objects in the set, i.e., for any permutation $\pi : f(\{x_1, \ldots, x_M\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(M)}\})$. Learning on sets emerges in several real-world applications, and has attracted considerable attention in the past years.

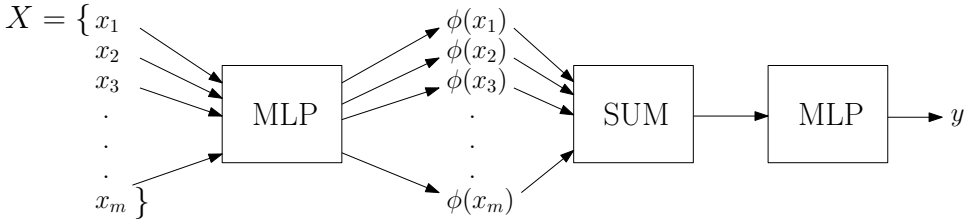


**Figure 1:** The DeepSets model.

## 2.1 Dataset Generation

For the purposes of this lab, we consider the task of finding the sum of a given set of digits, and we will create a synthetic dataset as follows: Each sample is a set of digits and its target is the sum of its elements. For instance, the target of the sample $X_i = \{8, 3, 5, 1\}$ is $y_i = 17$. We will generate $100,000$ training samples by randomly sampling between $1$ and $10$ digits ($1 \leq M \leq 10$) from $\{1, 2, \ldots, 10\}$. With regards to the test set, we will generate $200,000$ test samples of cardinalities from $5$ to $100$ containing again digits from $\{1, 2, \ldots, 10\}$. Specifically, we will create $10,000$ samples with cardinalities exactly $5$, $10,000$ samples with cardinalities exactly $10$, and so on.

**Task 1**
Fill in the body of the `create_train_dataset()` function in the `utils.py` file to generate the training set (consisting of $100,000$ samples) as discussed above. Each set contains between $1$ and $10$ digits where each digit is drawn from $\{1, 2, \ldots, 10\}$. To train the models, it is necessary that all training samples have identical cardinalities. Therefore, we pad sets with cardinalities smaller than $10$ with zeros. For instance, the set $\{4, 5, 1, 7\}$ is represented as $\{0, 0, 0, 0, 0, 0, 4, 5, 1, 7\}$ (Hint: use the `randint()` function of NumPy to generate random integers from $\{1, 2, \ldots, 10\}$).

**Task 2**
Fill in the body of the `create_test_dataset()` function in the `utils.py` file to generate the test set (consisting of $200,000$ samples) as discussed above. Each set contains from $5$ to $100$ digits again drawn from $\{1, 2, \ldots, 10\}$. Specifically, the first $10,000$ samples will consist of exactly $5$ digits, the next $10,000$ samples will consist of exactly $10$ digits, and so on.

## 2.2 Implementation of DeepSets

It can be shown that if $\mathfrak{X}$ is a countable set and $\mathcal{Y} = \mathbb{R}$, then a function $f(X)$ operating on a set $X$ having elements from $\mathfrak{X}$ is a valid set function, i.e., invariant to the permutation of instances in $X$, if and only if it can be decomposed in the form $\rho(\sum_{x \in X} \phi(x))$, for suitable transformations $\phi$ and $\rho$.

DeepSets achieves permutation invariance by replacing $\phi$ and $\rho$ with multi-layer perceptrons (universal approximators). Specifically, DeepSets consists of the following two steps:

- Each element $x_i$ of each set is transformed (possibly by several layers) into some representation $\phi(x_i)$.
- The representations $\phi(x_i)$ are added up and the output is processed using the $\rho$ network in the same manner as in any deep network (e.g., fully connected layers, nonlinearities, etc.).

An illustration of the DeepSets model is given in Figure 1.

**Task 3**
Implement the DeepSets architecture in the `models.py` file. More specifically, add the following layers:
– an embedding layer which projects each digit to a latent space of dimensionality $h_1$
– a fully-connected layer with $h_2$ hidden units followed by a tanh activation function
– a sum aggregator which computes the sum of the elements of each set
– a fully-connected layer with 1 unit since the output of the model needs to be a scalar (i.e., the prediction of the sum of the digits contained in the set)

We have now defined the DeepSets model. We will compare the DeepSets model against an LSTM, an instance of the family of recurrent neural networks. The next step is thus to define the LSTM model. Given an input set, we will use the LSTM hidden state output for the last time step as the representation of entire set.

**Task 4**
Implement the LSTM architecture in the `models.py` file. More specifically, add the following layers:
– an embedding layer which projects each digit to a latent space of dimensionality $h_1$
– an LSTM layer with $h_2$ hidden units
– a fully-connected layer which takes the LSTM hidden state output for the last time step as input and outputs a scalar

## 2.3 Model Training

Next, we will train the two models (i.e., DeepSets and LSTM) on the dataset that we have constructed. We will store the parameters of the trained models in the disk and retrieve them later on to make predictions.

**Task 5**
Fill in the missing code in the `train.py` file, and then execute the script to train the two models. Specifically, you need to generate all the necessary tensors for each batch.

## 2.4 Predicting the Sum of a Set of Digits

We will now evaluate the two models on the test set that we have generated. We will compute the accuracy and mean absolute error of the two models on each subset of the test set separately. We will store the obtained accuracies and mean absolute errors in a dictionary.

**Task 6**
In the `eval.py` file, for each batch of the test set, generate the necessary tensors for making predictions. Compute the output of two models. Compute the accuracy and mean absolute error achieved by the two models and append the emerging values to the corresponding lists of the `results` dictionary (Hint: use the `accuracy_score()` and `mean_absolute_error()` functions of scikit-learn).

We will next compare the performance of DeepSets against that of the LSTM. Specifically, we will visualize the accuracies of the two models with respect to the maximum cardinality of the input sets.

**Task 7**
Visualize the accuracies of the two models with respect to the maximum cardinality of the input sets (Hint: you can use the `plot()` function of Matplotlib).

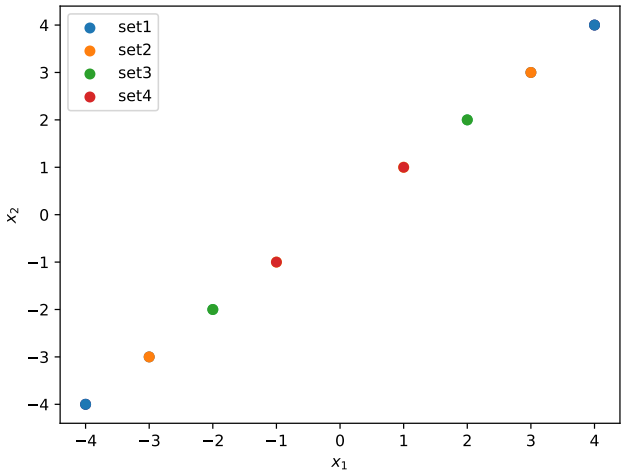


**Figure 2:** A synthetic example consisting of four sets.

**Question 1 (5 points)**
Consider the dataset shown in Figure 2 which consists of four sets of 2-dimensional vectors. The cardinality of all sets is equal to 2, while for all sets, the sum of their elements is equal to $[0, 0]$. Can the DeepSets model learn different representations for the four sets?

**Question 2 (5 points)**
In a graph classification problem (i.e., where each sample is a graph), could DeepSets correspond to a submodule of a graph neural network architecture?

# 3 Influence Maximization

In this part of the lab, we will study the problem of influence maximization in graphs. Influence maximization has attracted a lot of attention recently since it has many applications, ranging from viral marketing to disease modeling and public health interventions. Influence maximization seeks for a small subset of nodes in a network such that the resulting "influence" propagating from that subset reaches the largest number of nodes in the network. In the case of viral marketing, "influence" would correspond to the adoption of some product. In recent times, discovering influencers on social media is becoming increasingly important. For a small brand, finding a social media influencer with thousands of loyal followers to promote their products is much more economical and fruitful than spending their advertising budget on TV ads. Formally, the problem is formulated as follows: given a social network $G = (V, E)$, a diffusion model with some parameters and a number $k$, find a seed set $S \subset V$ of size $k$ such that the influence spread is maximized.

We will first introduce a diffusion model which can be used to simulate a spreading process that takes place over the network, and then, we will implement a greedy algorithm that utilizes this measure to choose the optimum nodes to maximize the spread of information.

## 3.1 Spread Process - Independent Cascade

Independent Cascade is a function that simulates the spread from a given set of nodes across the network. In an independent cascade model, a probability $p_e$ is specified for each edge of a network in ad-

vance. A node is influenced by its neighboring nodes with the predefined probability, independently. In this model, nodes can have two states: (1) active (influenced) or (2) inactive (not influenced). The model runs for a number of time steps and at each time step, some nodes have the opportunity to activate their neighbors. More specifically, all the nodes that were activated (influenced) at time step $t$ have a single chance to activate each of their neighbors. The success depends on the probability assigned to the edge connecting the two nodes. If a neighbor is successfully activated, then at time $t+1$, its state is switched from inactive (not influenced) to active (influenced). Since the vlaue of spread is not deterministic, we usually calculate the expected spread of a given seed set by taking the average over a large number of Monte Carlo simulations.

## 3.2 Greedy Algorithm

Given a spreading function (such as the Independent Cascade presented above), to find the set of most influential nodes, we can just find the set of nodes that maximizes the function. Unfortunately, for the Independent Cascade model, solving exactly this problem is NP-complete. Therefore, we will resort to an approximation algorithm. It has been shown that the function of the influence spread under the Independent Cascade models is monotone non-decreasing and submodular. A monotone non-decreasing submodular function can be maximized by a greedy algorithm with a $(1 - 1/e)$ approximation ratio. Hence, we can use the following greedy algorithm that approximates the optimal solution with a theoretical guarantee:

**Algorithm 1** Greedy Algorithm

**Input:** Graph $G = (V, E)$ and number of influential nodes $k$
**Output:** Set $S$ containing the most influential nodes

1: Initialize $S = \emptyset$
2: **for** $i = 1, \ldots, k$ **do**
3: $\quad v = \arg\max_{u \in V \setminus S} f(S \cup \{u\}) - f(S)$
4: $\quad S = S \cup \{v\}$
5: **end for**

What is exactly a monotone non-decreasing submodular function? If $V$ is a finite set:

- A function $f$ is monotone non-decreasing if and only if for all $S \subseteq T \subseteq V$, $f(S) \leq f(T)$
- A function $f : 2^V \rightarrow \mathbb{R}$ is submodular if it satisfies the following: $\forall S \subseteq T \subseteq V \backslash v$, we have that: $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$

**Task 8**
Fill in the body of the `greedy_algorithm()` function in the `inf_max.py` file. The algorithm takes as input a graph $G$, the size of the set to produce $k$, the probability $p$ of activating a neighbor (for the Independent Cascade model), and the number of Monte Carlo simulations $mc$ (for the Independent Cascade model). The algorithm returns the set of nodes identified by the greedy algorithm and the average spread for each subset of this set encountered during the different iterations of the algorithm.

**Question 3 (5 points)**
Suppose we need to compute exactly the influence spread instead of approximating it. What is the number of subgraphs we need to evaluate the influence on?

**Question 4 (5 points)**
Instead of employing the above greedy algorithm, we could alternatively create a set that contains the top-$k$ nodes according to some centrality measure (e.g., degree). However, such an approach is very likely to extract seed sets whose influence spread is smaller than that of the greedy algorithm. Why?

### 3.3 Identifying Influential Users

We will apply the greedy algorithm that we implemented to Socrates's social network, a graph derived from Plato's dialogues and some letters. The graph consists of 187 individuals, of whom 120 have direct relations with Socrates.

**Task 9**
Use the function `greedy_algorithm()` to find a set of 5 influential nodes. Set the number of Monte Carlo simulations to 100 and the probability that a node influences a neighbor equal to 0.1. Print the 5 nodes.

Finally, we will plot the expected spread for each seed set size from 1 to 5.

## References


[1] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the Spread of Influence through a Social Network. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[2] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3744–3753, 2019.

[3] George Panagopoulos, Fragkiskos D Malliaros, and Michalis Vazirgianis. Influence Maximization Using Influence and Susceptibility Embeddings. In *Proceedings of the 14th International AAAI Conference on Web and Social Media*, volume 14, pages 511–521, 2020.

[4] Konstantinos Skianis, Giannis Nikolentzos, Stratis Limnios, and Michalis Vazirgiannis. Rep the Set: Neural Networks for Learning Set Representations. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pages 1410–1420, 2020.

[5] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.