Advanced Deep Learning

Lab 8: Symmetry in Deep Learning Representations

Authored by: Giannis Nikolentzos & Johannes Lutzeyer
Taught with: Guillaume Lachaud & Robin Courant

04 March 2025

1 Learning objective

The goal of this lab is to introduce you to machine learning models for data represented as sets. Specifically, 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.

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.

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 multiset 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.

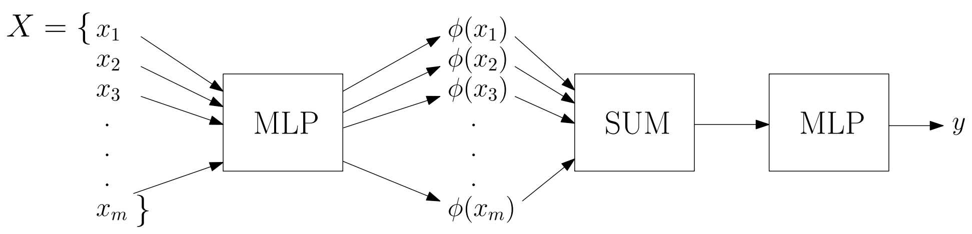

Figure 1: The DeepSets model.

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 sample should be a multiset of a length that is uniformly at random sampled from $\{1, 2, \ldots, 10\}$. Further, each element of the multisets should also be uniformly at random sampled from $\{1, 2, \ldots, 10\}$. To train the models, it is necessary that all training samples have identical cardinalities. Therefore, we pad multisets with cardinalities smaller than 10 with zeros. For instance, the multiset $\{4, 5, 1, 7, 5\}$ is represented as $\{0, 0, 0, 0, 0, 4, 5, 1, 7, 5\}$ (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. Note that here we would like you to return a list of numpy arrays, where the first list element contains a numpy array containing $10,000$ samples with cardinality 5, the second list element should contain a numpy array with $10,000$ samples of cardinality 10, and the final list element should containing $10,000$ samples with cardinality 100.

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 ϕ and ρ.

DeepSets achieves permutation invariance by replacing ϕ and ρ with multi-layer perceptrons (universal approximators). Specifically, DeepSets consists of the following two steps [1]:

- 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 ρ 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. When calculating the accuracy of our models you may want to use numpy's `round()` function to set all predictions to integer values.).

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).

References

[1] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances on Neural Information Processing Systems*, pages 3394–3404, 2017.