CS324: Deep Learning
Assignment 2

Luca Rossi

April 11, 2019

1 Part I: PyTorch MLP (30 points)

In the first part of this second assignment you will implement the same MLP architecture you implemented in Assignment 1 but this time using PyTorch.

1.1 Task 1

Implement the MLP architecture and the training procedure by completing the files **pytorch_mlp.py** and **pytorch_train_mlp.py**.

1.2 Task 2

Train both your numpy version of MLP and the PyTorch version on the same data. To this end, I suggest you to create a jupyter notebook file where you first create the training and set data using `make_moons` (as you did in Assignment 1), import the definition of the numpy and PyTorch MLP implementations, train them, and test them on the same data. You should see similar accuracy rates with both implementations of the MLP architecture. You are encouraged to test your MLPs with different datasets (see `https://scikit-learn.org/stable/modules/classes.html#samples-generator`), plotting whenever possible for each chosen dataset the sampled points, highlighting with different colours and shapes the different classes for both training and test set, and showing the accuracy achieved with the numpy and PyTorch versions of the MLP architecture. It's up to you if you want to use stochastic gradient descent, batch gradient descent or mini-batch gradient descent: just make sure to train the two models using the same strategy.

1.3 Task 3

Using **torchvision.datasets.CIFAR10** load the CIFAR10 dataset. Using PyTorch and the units, optimisation methods, regularisation methods, etc., studied in these weeks, try to obtain the highest accuracy you can on this dataset. Whenever possible use validation sets, but don't worry too much about it at this stage. You're free to implement your architecture in a separate .py file, but you should use a jupyter notebook to run the experiments, illustrate them, and comment on the results.

2 Part II: PyTorch CNN (30 points)

In the second part of this assignment you will try to improve your neural architecture by using convolutional neural networks (CNNs). You will use again the CIFAR10 dataset but this time you will implement a CNN to classify the images in the dataset.

2.1 Task 1

Implement the CNN architecture and the training procedure by completing the files **cnn_model.py** and **cnn_train.py**. The architecture of the network you need to create is detailed in the slides at the end of **Lesson 5.pdf** (today's lecture) and it's a reduced version of the well-known VGG network.

2.2 Task 2

Analyse the performance of the model by plotting accuracy and loss curves in a Jupyter notebook. To this end, use the Adam optimizer with default learning rate and default PyTorch parameters to initialize convolutional and linear layers. To compute the gradient, use mini-batch gradient descent.

3 Part III: PyTorch RNN (40 points) - NEW

In this assignment you will implement a RNN to predict the last digit of a palindrome. Palindromes in general are strings that look the same when read forward and backward, e.g., 123454321, 1111, 4224, etc. The file **dataset.py** contains the class PalindromeDataset, which extends PyTorch datasets and contains the function *generate_palindrome* to generate palindromes of numbers. You will use this dataset to sample the mini-batches and train your RNN.

You RNN will have to follow the structure defined by the equations

$$h^{(t)} = \tanh(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \tag{1}$$

$$o^{(t)} = (W_{ph}h^{(t)} + b_o)\,. \tag{2}$$

$$\tilde{y}^{(t)} = \text{softmax}(o^{(t)})\,. \tag{3}$$

Initialise $h^{(0)}$ to the vector of all zeros. Since the task we're considering here is that of predicting only the last digit of a palindrome number, we compute the cross-entropy loss only over the last time-step, i.e.,

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log(\tilde{y}_k^{(T)})\,, \tag{4}$$

where k runs over the classes ($K = 10$ digits in total), y_k is a one-hot encoding vector.

3.1 Task 1

Implement the RNN without using *torch.nn.RNN* and *torch.nn.LSTM*. Follow the skeleton provided in **train.py** (for the training) and **vanilla_rnn.py** (to define the model). For the forward pass you will need to use a *for* loop to step through time and apply the recurrence equations that define the network behaviour. For the backward pass you can rely on Pytorch's automatic differentiation and use the RMSProp optimizer for tuning the weights.

3.2 Task 2

Given the RNN implemented in Task 1 and a palindrome of length T, the network should be able to predict the T-th digit given the preceeding $T-1$ ones. Now, as studied in the lecture this morning, RNN have a limited memory. So we expect the prediction accuracy of the RNN become lower as we challenge it with longer and longer palindromes. Using a jupyter notebook create a plot that shows accuracy versus palindrome length. You should be able to obtain close to perfect accuracy with $T = 5$ and the default parameters provided in the python files.

4 Submission instructions

Create a ZIP archive with the results of Assignment 2 (all parts and tasks). Give the ZIP file the name **studentnumber_assignment2.zip**, where you insert your student number. Please submit the archive through Sakai. Make sure all files needed to run your code are included or you may be given 0 points for it.

The deadline for assignment 1 (all parts and all tasks) is the 18th of April 2019 at 23:55 (Beijing Time).