

Homework 3 Part 2 (Summer 2019)

MNIST Sequence Recognition

11-785: Introduction to Deep Learning

1 Introduction

In this homework you will be working with MNIST digit-sequences. As you might already be aware, MNIST is a hand-written digit dataset that contains numerous style examples for every digit (0-9). We will be using sequences of digits from the MNIST dataset padded vertically and shifted randomly within a given range. This task is different from HW1P2 as you must develop networks that recognize a sequence of digits rather than just one. Alignment and ordering are thus important. An example of the sequence before transformation (i.e., random shifting) is displayed below:

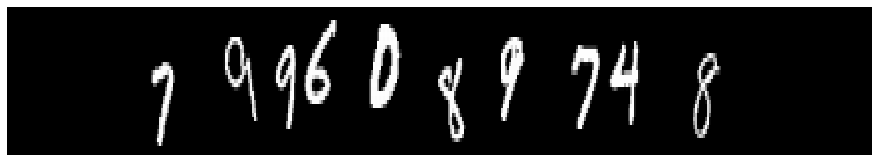


Figure 1: MNIST digit sequence from our dataset

2 Dataset

Sequence images provided to you are of the shape (36 x 432), they contain 10 MNIST digits in a random order and style. Your model must thus predict 10 digits for each sample.

2.1 Files

We have a total of 8 files for this assignment, 2 '.csv' files, 1 '.py' file and 5 '.npy' files:

- **mnist_train.npy**: This file contains your feature data for training the model. The training data contains 2000 images of shape (36, 432), therefore it will be of the shape (2000, 36, 432).
- **mnist_dev.npy**: This file is similar to **mnist_train.npy**, but should be used to calculate your validation losses and accuracy.

- **mnist_test.npy**: This file is similar to **mnist_train.npy**, but should be used to predict the sequence labels for the final Kaggle submission.
- **mnist_train_labels.npy**: This file contains the labels or sequence list for each image in **mnist_test.npy**. The dimensions of the data in this file will be of the form (2000, 10) since each image contains a sequence of 10 digits.
- **mnist_dev_labels.npy**: This file is similar to the one above, but instead will map the labels to the **mnist_dev.npy** file. You can use this for predicting validation losses and accuracy.
- **sample_submission.csv**: This is an empty submission file that contains test image Ids and dummy predictions for each image of test data.
- **gold_labels.csv**: This file contains the gold-standard predictions for the test data-set. You will be scored by calculating the Levenstein distance between your network predictions and the predictions in this file. You need not change or use this file, this file is used by the tester script that scores your predictions.
- **tester.py**: This python script must be used for scoring your predictions. you must run the script with your prediction csv file as the first argument:

python tester.py YourFile.csv

The program will print out the average Levenstein between your prediction strings and the actual digit strings from the gold-standard data-set.

3 Getting Started

3.1 Network Architecture

In this task, you are expected to use Recurrent Neural Networks which powerful type of neural network designed to handle sequence dependence. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning very frequently. Read up more about the LSTM from the resources section. You can use the pytorch implementaion of LSTM (nn.LSTM) for this assignment. Your architecture should take as the input a batch of digit sequence images, run them through 3-4 LSTM layers and finally use a classification layer (You can use a linear layer or an MLP network) to predict one of the 10 digits or a "-" which means **None** (you will know why we need a **None** symbol in the next section). Optionally, instead of directly using the images as input to the LSTM, you can use a CNN feature extractor and use the representation from the CNN as input to the LSTM layers. Simple cross-entropy loss will not work well in this case since we are not aware of digit boundaries and alignments, you must use CTC decoding and Loss as described in the next section. Finally, you will be evaluated on the basis of the average Levenstein distance between your predictions and the gold-standard on **mnist_test.npy**. Our baseline model uses a single CNN feature extractor (1D CNN, kernel size=2, stride=1) and 3 LSTM layers (using the most intuitive hidden size is left as an exercise).

3.2 CTC Loss

In this homework, you are expected to output a sequence of 10 digits present in the given image. This task can be thought of as a text-recognition system with 10 possible character outputs. In order to obtain ground truth, we could specify for each horizontal position of the image the corresponding character and then train a network for each position, much like an extension of HW1P2, however, it is very time-consuming to annotate a data-set on character-level in the real world and we only get character-scores and therefore need some further processing to get the final text from it. For instance, A single character can span multiple horizontal positions. We could get an output "mmeee" for "me" if "e" is a wide character in the image. We could remove duplicates, but doing this always as a rule might hurt in cases when there are repetitions in the source text itself.

CTC loss helps us get around this problem like magic! We are only required to provide the CTC loss function the characters that occur in the image and we ignore both the position and width of the characters in the image. You could read up more about CTC loss from the articles mentioned in the resources section.

Tensorflow has built-in CTC loss function. But for Pytorch, you can install the CTC-Loss library. The required CTC-loss library, Warp-CTC, can be installed [here](#). Follow the download instructions. NOTE: Make sure you download the GPU supported version.

3.3 Using Warp CTC in pytorch

The module assumes the blank position is at 0. If there are n (here, 10) labels, the model output should be of size $n+1$. The targets passed to the loss function should be from 1 to n instead of from 0 to $n-1$. The loss function takes four inputs. The first input is a tensor of your model network output. The second input is a tensor of all the labels should be concatenated into one single 1D tensor. The third input is a tensor containing the size of each output sequence from the network. The fourth input is a tensor containing the label of each target. Your submission should contain the predicted strings (i.e., digits) for each item in the test set.

3.4 CTC Decoding

If you are using PyTorch you can manually install the library, `ctcdecode`, [here](#). They have an implementation of beam search, use it in your code to decode the output of your model. If you are using Tensorflow, it has its own implementation of beam search.

4 Evaluation and Submission

You will be evaluated using character-level string edit distance. We are using Levenshtein distance, which counts how many additions, deletions and modifications are required to make one sequence into another.

Your submission should be a CSV file. The first column must contain the instance id in

numerical order and the second column must contain the predicted digit string. Refer to the sample submission file for more details. You have been provided with a python script for obtaining the average Levenshtein distance between your predictions and the gold-standard. Run the file with your prediction filename as the first argument (**python tester.py your-FileName.csv**).

5 Resources

1. **RNNs Blog:** <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>
2. **Pytorch RNN:** https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html
3. **Prof.Bhiksha's Lecture Slides:** [Click here](#)
4. **More about CTC:** <https://distill.pub/2017/ctc/>
5. **CTC Networks and Language Models:** <https://medium.com/corti-ai/ctc-networks-and-language-models-prefix-beam-search-explained-c11d1ee23306>