Taxi Destination Prediction Challenge
Winner Team's Report

Montréal, July 2015

Alex Auvolat	Alexandre de Brébisson	Étienne Simon
ENS Paris	Université de Montréal	ENS Cachan
France	Québec, Canada	France
alexis211@gmail.com	adbrebs@gmail.com	esimon@esimon.eu

1 Summary

Our model is based on a multi-layer perceptron (MLP). Our MLP model is trained by stochastic gradient descent (SGD) on the training trajectories. The inputs of our MLP are the 5 first and 5 last positions of the known part of the trajectory, as well as embeddings for the context information (date, client and taxi identification). The embeddings are trained with SGD jointly with the MLP parameters. The MLP outputs probabilities for 3392 target points, and a mean is calculated to get a unique destination point as an output. We did no ensembling and did not use any external data.

2 Feature Selection/Extraction

We used a mean-shift algorithm on the destination points of all the training trajectories to extract 3392 classes for the destination point. These classes were used as a fixed softmax layer in the MLP architecture.

We used the embedding method which is common in neural language modeling approaches (see [1]) to take the metainformation into account in our model. The following embeddings were used (listed with corresponding dimensionnality):

Meta-data	**Embedding Dimension**	**Number of classes**
Unique caller number	10	57125
Unique stand number	10	64
Unique taxi number	10	448
Week of year	10	54
Day of week	10	7
1/4 of hour of the day	10	96
Day type (invalid data)	10	3

Table 1. Embeddings and corresponding dimensions used by the model

The embeddings were first initialized to random variables and were then let to evolve freely with SGD along with the other model parameters.

The geographical data input in the network is a centered and normalized version of the GPS data points.

We did no other preprocessing or feature selection.

3 Modelling Techniques and Training

Here is a brief description of the model we used:

- **Input.** The input layer of the MLP is the concatenation of the following inputs:
  - Five first and five last points of the known part of the trajectory.

    - Embeddings for all the metadata.

- **Hidden layer.** We use a single hidden layer MLP. The hidden layer is of size 500, and the activation function is a Rectifier Linear Unit (ie $f(x) = \max(0, x)$) [2].

- **Output layer.** The output layer predicts a probability vector for the 3392 output classes that we obtained with our clustering preprocessing step. If $\boldsymbol{p}$ is the probability vector output by our MLP (output by a softmax layer) and c_i is the centroid of cluster i, our prediciton is given by:

  $$\hat{y} = \sum_i p_i c_i$$

  Since $\boldsymbol{p}$ sums to one, this is a valid point on the map.

- **Cost.** We directly train using an approximation (Equirectangular projection) of the mean Haversine Distance as a cost.

- **SGD and optimization.** We used a minibatch size of 200. The optimization algorithm is simple SGD with a fixed learning rate of 0.01 and a momentum of 0.9.

- **Validation.** To generate our validation set, we tried to create a set that looked like the training set. For that we generated "cuts" from the training set, i.e. extracted all the taxi rides that were occuring at given times. The times we selected for our validation set are similar to those of the test set, only one year before:

```
1376503200, # 2013-08-14 18:00
1380616200, # 2013-10-01 08:30
1381167900, # 2013-10-07 17:45
1383364800, # 2013-11-02 04:00
1387722600  # 2013-12-22 14:30
```

4 Code Description

Here is a brief description of the Python files in the archive:

- `config/*.py` : configuration files for the different models we have experimented with

  The model which gets the best solution is `mlp_tgtcls_1_cswdtx_alexandre.py`

- `data/*.py` : files related to the data pipeline:

    - `__init__.py` contains some general statistics about the data
    - `csv_to_hdf5.py` : convert the CSV data file into an HDF5 file usable directly by Fuel
    - `hdf5.py` : utility functions for exploiting the HDF5 file
    - `init_valid.py` : initializes the HDF5 file for the validation set
    - `make_valid_cut.py` : generate a validation set using a list of time cuts. Cut lists are stored in Python files in `data/cuts/` (we used a single cut file)
    - `transformers.py` : Fuel pipeline for transforming the training dataset into structures usable by our model

- `data_analysis/*.py` : scripts for various statistical analyses on the dataset
  - `cluster_arrival.py` : the script used to generate the mean-shift clustering of the destination points, producing the 3392 target points
- `model/*.py` : source code for the various models we tried
  - `__init__.py` contains code common to all the models, including the code for embedding the metadata
  - `mlp.py` contains code common to all MLP models
  - `dest_mlp_tgtcls.py` containts code for our MLP destination prediction model using target points for the output layer
- `error.py` contains the functions for calculating the error based on the Haversine Distance
- `ext_saveload.py` contains a Blocks extension for saving and reloading the model parameters so that training can be interrupted
- `ext_test.py` contains a Blocks extension that runs the model on the test set and produces an output CSV submission file
- `train.py` contains the main code for the training and testing

In the archive we have included only the files listed above, which are the strict minimum to reproduce our results. More files for the other models we tried are available on GitHub at https://github.com/adbrebs/taxi.

5 Dependencies

We used the following packages developped at the MILA lab:

- **Theano.** A general GPU-accelerated python math library, with an interface similar to numpy (see [3, 4]). http://deeplearning.net/software/theano/
- **Blocks.** A deep-learning and neural network framework for Python based on Theano. https://github.com/mila-udem/blocks
- **Fuel.** A data pipelining framework for Blocks. https://github.com/mila-udem/fuel

We also used the `scikit-learn` Python library for their mean-shift clustering algorithm. `numpy`, `cPickle` and `h5py` are also used at various places.

6 How To Generate The Solution

1. Set the `TAXI_PATH` environment variable to the path of the folder containing the CSV files.
2. Run `data/csv_to_hdf5.py` to generate the HDF5 file (which is generated in `TAXI_PATH`, along the CSV files). This takes around 20 minutes on our machines.
3. Run `data/init_valid.py` to initialize the validation set HDF5 file.
4. Run `data/make_valid_cut.py test_times_0` to generate the validation set. This can take a few minutes.

5. Run `data_analysis/cluster_arrival.py` to generate the arrival point clustering. This can take a few minutes.

6. Create a folder `model_data` and a folder `output` (next to the training script), which will receive respectively a regular save of the model parameters and many submission files generated from the model at a regular interval.

7. Run `./train.py dest_mlp_tgtcls_1_cswdtx_alexandre` to train the model. Output solutions are generated in `output/` every 1000 iterations. Interrupt the model with three consecutive Ctrl+C at any times. The training script is set to stop training after 10 000 000 iterations, but a result file produced after less than 2 000 000 iterations is already the winning solution. We trained our model on a GeForce GTX 680 card and it took about an afternoon to generate the winning solution.

   When running the training script, set the following Theano flags environment variable to exploit GPU parallelism:

   ```
   THEANO_FLAGS=floatX=float32,device=gpu,optimizer=FAST_RUN
   ```

   Theano is only compatible with CUDA, which requires an Nvidia GPU. Training on the CPU is also possible but much slower.

7 Additional Comments and Observations

The training examples fed to the model are not full trajectories, since that would make no sense, but prefixes of those trajectories that are generated on-the-fly by a Fuel transformer, `TaxiGenerateSplits`, whose code is available in `data/transformers.py`. The data pipeline is as follows:

- Select a random full trajectory from the dataset
- Generate a maximum of 100 prefixes for that trajectory. If the trajectory is smaller than 100 data points, generate all possible prefixes. Otherwise, chose a random subset of prefixes. Keep the final destination somewhere as it is used as a target for the training.
- Take only the 5 first and 5 last points of the trajectory.
- At this points we have a stream of prefixes sucessively taken from different trajectories. We create batches of size 200 with the items of the previous stream, taken in the order in which they come. The prefixes generated from a single trajectory may end up in two sucessive batches, or all in a single batch.

8 References

1. Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, *3*, 1137-1155.

2. Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics* (pp. 315-323).

3. Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., ... & Bengio, Y. (2011). Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*.

4. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., ... & Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.