DeepSurv in Social Science: Modelling Refugee Journey Duration

---

Purpose of this Notebook

- This notebook applies *DeepSurv*, a Cox proportional hazards deep neural network, in a social science setting. I study whether *DeepSurv* outperforms the predictive accuracy of traditional Cox models. The basis for comparison is Harrel's c-index, which in this case measures how well a model ranks the arrival times of refugees. The c-index of the best traditional Cox model is equal to 0.72.

Step 1: Import Packages

- The script first imports required packages. Those include the deep learning package *Lasagne*, the widely-used Python packages *numpy*, *pandas* and *matplotlib* as well as *DeepSurv*. You need to download and install [DeepSurv from Github (https://github.com/jaredleekatzman/DeepSurv)](https://github.com/jaredleekatzman/DeepSurv) prior to the analysis.

In [1]:
```
import sys
sys.path.append('../deepsurv')
import deep_surv

from deepsurv_logger import DeepSurvLogger, TensorboardLogger
import utils
import viz

import numpy as np
import pandas as pd

import lasagne
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

```
/Users/timfingerhut/anaconda3/lib/python3.6/site-packages/h5py/__i
nit__.py:36: FutureWarning: Conversion of the second argument of i
ssubdtype from `float` to `np.floating` is deprecated. In future,
it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Step 2: Select the Dataset

The following three lines of code

> 1. load the dataset,
> 2. transform the data into a *Pandas* Dataframe (the most common data structure in Python, analogous to a csv file with column headers),
> 3. and output the first lines of the dataset.

The data appears to be in good shape. Please note that continuous variables, such as age, need to be normalized to a 0 to 1 scale prior to employing *DeepSurv* (mmx refers to the minimum-maximum normalization procedure, the 'imp' ending signifies that missing data was imputed).

In [2]:
```
train_dataset_fp = './2006_final.csv' # make sure to set the Python working directory to the path of your dataset and insert the name of your csv file
train_df = pd.read_csv(train_dataset_fp)
train_df.head()
```

Out[2]:

	fail_1	dur	country1	country2	country4	country5	country6	country7	country8	count
0	1	30	1	0	0	0	0	0	0	
1	1	1	1	0	0	0	0	0	0	
2	1	60	1	0	0	0	0	0	0	
3	1	60	1	0	0	0	0	0	0	
4	1	45	1	0	0	0	0	0	0	

5 rows × 67 columns

Step 3: Prepare the Data for Event History Analysis

- The data is almost ready for *DeepSurv*. As in traditional event history analysis, the *failure* and *duration* variables still need to be indicated (in this case 'fail_1' and 'dur'). The event / failure indicator is saved as 'e', time 't' is coded in days in this case and the covariates are saved under 'x').

In [3]:
```
# event_col is the header in the df that represents the 'Event / St
atus' indicator
# time_col is the header in the df that represents the event time
def dataframe_to_deepsurv_ds(df, event_col = 'fail_1', time_col = '
dur'):
    # Extract the event and time columns as numpy arrays
    e = df[event_col].values.astype(np.int32)
    t = df[time_col].values.astype(np.float32)

    # Extract the patient's covariates as a numpy array
    x_df = df.drop([event_col, time_col], axis = 1)
    x = x_df.values.astype(np.float32)

    # Return the deep surv dataframe
    return {
        'x' : x,
        'e' : e,
        't' : t
    }

# If the headers of the csv change, you can replace the values of
# 'event_col' and 'time_col' with the names of the new headers
# You can also use this function on your training dataset, validati
on dataset, and testing dataset
train_data = dataframe_to_deepsurv_ds(train_df, event_col = 'fail_1
', time_col= 'dur')
```

Step 4: Set Hyperparameters

- Based on a manual hyperparameter tuning focused on the *learning rate*, I retain the following hyperparameters. Advanced users can implement a random or Bayesian optimization procedure. For an example of a random hyperparameter search, visit Jared Lee Katzman's GitHub (https://github.com/jaredleekatzman/DeepSurv/tree/master/hyperparam_search).

In [4]:
```
hyperparams = {
    'L2_reg': 10.0,
    'batch_norm': True,
    'dropout': 0.4,
    'hidden_layers_sizes': [100, 100],
    'learning_rate': 3e-2,
    'lr_decay': 0.001,
    'momentum': 0.9,
    'n_in': train_data['x'].shape[1],
    'standardize': True
}
```

Step 5: Train _DeepSurv_

- The dataframe and hyperparameters being defined, it is now time to **train *DeepSurv***. Tensorboard allows you to monitor the training progress in real-time.
- The code below also defines the **update function** (= amsgrad) as well as the **number of training epochs** (= 2000).

In [5]:
```
# Create an instance of DeepSurv using the hyperparams defined abov
e
model = deep_surv.DeepSurv(**hyperparams)

# DeepSurv can now leverage TensorBoard to monitor training and val
idation
# This section of code is optional. If you don't want to use the te
nsorboard logger
# Uncomment the below line, and comment out the other three lines:
# logger = None

experiment_name = 'test_experiment_tim'
logdir = './logs/tensorboard/'
logger = TensorboardLogger(experiment_name, logdir=logdir)

# Now we train the model
update_fn=lasagne.updates.amsgrad # The type of optimizer to use. \
                                  # Check out http://lasa
gne.readthedocs.io/en/latest/modules/updates.html \
                                  # for other optimizers
to use
n_epochs = 2000

# If you have validation data, you can add it as the second paramet
er to the function
metrics = model.train(train_data, n_epochs=n_epochs, logger=logger,
update_fn=update_fn)
```

```
2019-07-30 13:53:45,675 - Training step 0/2000    |
| - loss: 23.3311 - ci: 0.6639
2019-07-30 13:54:43,876 - Training step 250/2000  |***
| - loss: 6.3844 - ci: 0.8368
2019-07-30 13:55:44,226 - Training step 500/2000  |******
| - loss: 6.2731 - ci: 0.8734
2019-07-30 13:56:42,873 - Training step 750/2000  |*********
| - loss: 6.2329 - ci: 0.8915
2019-07-30 13:57:41,429 - Training step 1000/2000 |************
| - loss: 6.1705 - ci: 0.8971
2019-07-30 13:58:42,531 - Training step 1250/2000 |***************
| - loss: 6.1805 - ci: 0.9046
2019-07-30 13:59:41,270 - Training step 1500/2000 |***************
***        | - loss: 6.1290 - ci: 0.9070
2019-07-30 14:00:39,638 - Training step 1750/2000 |***************
******     | - loss: 6.1346 - ci: 0.9047
2019-07-30 14:01:37,559 - Finished Training with 2000 iterations i
n 472.25s
```

Step 6: Output and visualize the Results

- Finally, it is time to print and visualize *DeepSurv*'s metrics.
- Compared to the traditional Cox model (c-index = 0.72), DeepSurv orders refugees' arrival times far more accurately (c-index = 0.9102).

In [6]:
```
# Print the final metrics
print('Train C-Index:', metrics['c-index'][-1])
# print('Valid C-Index: ',metrics['valid_c-index'][-1])

# Plot the training / validation curves
viz.plot_log(metrics)
```

```
Train C-Index: (1999, 0.9101721615528376)
```