metaPath Cookbook

document version 0.1, software version 0.4.37

Created for the 6th network modeling workshop of the workgroup „network modeling“ (Prof. Dr. König, Jena, Germany)

Patrick Michl

05.06.2013

Introduction

Why?

From wikipedia, the free enzykopedia (https://en.wikipedia.org/wiki/Deep_learning, version from 03.06.2013):

> *The term "deep learning" gained traction in the mid-2000s after a publication by Geoffrey Hinton showed how a many-layered neural network could be effectively trained one layer at a time, treating each layer in turns as a restricted Boltzmann machine. Although the backpropagation algorithm had been available for training neural networks since 1986, it was often considered too slow for practical use. As a result, neural networks fell out of favor in practical machine learning and simpler models such as support vector machines dominated much of the field in the 1990s.*
>
> *Apart from improvements in algorithms, advances in hardware have been an important enabling factor for the resurgence of neural networks and the advent of deep learning, in particular the availability of powerful and inexpensive graphics processing units (GPUs) also suitable for general-purpose computing. GPUs are highly suited for the kind of "number crunching" involved in machine learning, and have been shown to speed up training algorithms by orders of magnitude, bringing running times of weeks back to days.*

What?

metaPath is a python package that uses GPU computing to create deep neuronal networks from gene expression data. At the moment it implements: RBMs, GRBMs, deep neuronal networks, autoencoders, Data Annotation with BioConductor, Data Stratification, Data Normalization, dimensionality reducing transformation, Knockout tests on various properties of systems, corralation analysis, bicorrelation cluster analysis, …

Contents!

Software configuration

Base configuration

Configuration example (*Base Configuration*)

File: `metapath.ini`

```
# metaPath base configuration

[folders]
user = ./projects/
common = ./share/
```

user

User projects directory. Folders in this directory contain projects with networks, datasets, models, plots, reports, logfiles etc. and can be opened inside metapath workspaces. Configuration files ('.ini') in the top level of a projects directory (for example './projects/example') are parsed when opening a project. All objects of the project can then be accessed via the projects namespace (for example 'example.testnetwork'). To create new models, datasets etc. read and write permissions are necessary.

common

Shared projects directory. All projects inside this directory are automatically parsed when opening any project in the user projects directory. This allows for example sharing datasets, networks etc. on a network share. New ressources inside common projects are not created by metapath workspaces, therefore only read permissions are necessary.

Network Configuration

Networks are configured via '.ini' files in the top level of a projects directory. The filename is arbitrary. Within configuration files networks are identified with a section starting with 'network'.

Static networks

Static networks define stacked networks with a given structure of layers, nodes and edges. For the purpose to automatically match network nodes with dataset colums

Configuration example (*3-layer GAG-Pathway regulation*)

from File: `projects/example/networks.ini`

```
[network gag]
description:
  3-layer GAG-Pathway Regulation
source:
  file = '%networks%/gag.ini'
```

source

The network 'example.gag' is associated with the static network configuration file 'gag.ini' in the networks directory of the current project ('example/networks')

File: `projects/example/networks/gag.ini`

```
# metaPath network configuration
# GAG Pathway, virtual TFs, bunch of SigMols, full connected

[network]
type = layer
layers: visible:e, hidden:tf, visible:s
nodes = gene:alias

[e-tf binding]
GUSB: h0,h1,h2,h3,h4,h5
NAGLU: h0,h1,h2,h3,h4,h5
GNS: h0,h1,h2,h3,h4,h5
HYAL2: h0,h1,h2,h3,h4,h5
GLB1: h0,h1,h2,h3,h4,h5
GALNS: h0,h1,h2,h3,h4,h5
SGSH: h0,h1,h2,h3,h4,h5

[tf-s binding]
h0: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
h1: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
h2: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
h3: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
h4: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
h5: SHC1,HK1,CHD5,ANXA2,HDAC1,KALRN,LIMS1,YBX1,PTBP1,ELF4,RASGRF1,MYL12A
```

type

The network is layered (bipartite topology). At the moment only layered networks are supported by metaPath

layers

The number, the order and the labels of the layers are given by a comma-separated list. The first part (before ':') of the labels define the *layergroup*. The special layergroups *visible* and *hidden* are used by Hidden Markov Models like RBMs, GRBMs and ANNs like autoencoders to structure the system: *RBMs*, *GRBMs* and autoencoders group all *visible* tagged layers in one single *visible* layer and all *hidden* tagged layers in one single *hidden* layer. Autoencoders define this *hidden* layer as bottleneck. The second part (after ':') of the labels define the *layernames* which have to be unique.

nodes

With the purpose to automatically match network nodes with dataset columns annotation can be necessary. Therefore metaPath provides annotation modules. One oft them is *gene* which basically wraps the request to the bioconductor annotation data packages. For example *gene: hgu133a*, *gene:symbol* or *gene:alias*

sections

Sections of the format [<layerA>-<layerB> binding] where layerA and layerB are neighbouring layers contain node labels and describe edges. The single label left of ':' is the label of a node in layerA, right of ':' a list of nodes in LayerB. Depending on the system the edges are bidirectional (energy based models: RBM, GRBM) or directed from layerA to layerB (feed worward ANNS: Autoencoder).

Automatic networks

metaPath supports generated networks (for example for simulations)

Configuration example (generated 3-Layer regulation networks)

from File: `projects/example/networks.ini`

```
[network etfs]
type: autolayer
layers:
  e = visible, tf = hidden, s = visible
params:
  e.size = 4, tf.size = 4, s.size = 4
```

type

The name of the generator / algorithm which is used to create node labels and edges

layers

Similar to the static network configuration file. At the moment still with different systax but with same effect.

params

All params which are defined within the params dictionary are default values. Within a metaPath script in this example the network 'example.etfs' would create a 3-layer network with 4 'e'-nodes (e1, e2, …), 4 'tf' nodes and 4 's'-nodes. All values inside a params dictionary can be changed easily within a metapath script: for example 'example.etfs(e.size = 8, tf.size = 4, s.size = 12)' is a valid name for a generated 'etfs' network.

Dataset Configuration

Like networks, datasets are configured via arbitrary '.ini' files in the top level of a projects directory. Within those files datasets are identified with a section starting with dataset. At the moment metaPath supports csv-files and binary cache files for data storage.

Configuration example (*gbm.tcga*)

File: `share/gbm.tcga/datasets.ini`

```
# metaPath dataset configuration
# Glioblastoma multiforme, The Cancer Genome Atlas
```

```
[dataset normal]
description:
  Glioblastoma multiforme normal
source:
  file = '%datasets%/normal.tab',
  csvtype = 'r-table',
  columns = 'gene:hthgu133a'
preprocessing:
  normalize = 'z-trans'

[dataset wt]
description:
  Glioblastoma multiforme wildtype
source:
  file = '%datasets%/wt.tab',
  csvtype = 'r-table',
  columns = 'gene:hthgu133a'
preprocessing:
  normalize = 'z-trans'

[dataset mut]
description:
  Glioblastoma multiforme mutated
source:
  file = '%datasets%/mut.tab',
  csvtype = 'r-table',
  columns = 'gene:hthgu133a'
preprocessing:
  normalize = 'z-trans'

[dataset all]
description:
  Glioblastoma multiforme
source:
  datasets = 'normal, wt, mut'
preprocessing:
  stratify  = 'equal',
  normalize = 'z-trans'
```

source

The source of the data. This can be a file (file = …) or an other datasets (datasets = …).

In the case of a file, the filetype hast o be specified. The *csvtype* 'r-table' defines a csv dialect produced ba csv files which has been exported from R. The *columns* define the annotation module which is used to match network nodes. See also ‚nodes' in the static network configuration.

In the case of other datasets all dataset names are passed comma-separated in a single string. If a dataset in this string is not passed with its project namespace (for example 'normal' instead of 'gbm.tcga.normal') the namespace oft he configuration file is assumed.

preprocessing

Steps in preprocessing. The data preprocessing steps are in this order: *stratification* 'stratify = …', *normalization* 'normalize = …' and *transformation* 'transform = …'.

Stratification: Supported for source from other datasets. The canonical arguments are 'none' for no stratification, 'equal' for equal stratification and 'auto' for stratification which respects the hierarchic definition of the files.

Normalization: ANN based models usually represent input data with units with given probability distributions. Therefore normalization oft he input data can be necessary. Valid normalizations: 'z-trans', 'gauss' (which are the same), 'mean-t', … If data is stratified the normalization of the sources is ignored instead the compound is normalized

Transformation: A simple transformation is 'binary' which creates 0,1 values from gauss normalized data. A more complex transformation can be provided by naming previous created models like autoencoders which can be used as dimensionality reducing transformations or any other complex transformation from models. If data is stratified the transformation of the sources is ignored instead the compound is transformed.

System Configuration

Systems define the dynamics that are assumed in the data. Depending on the system there can be a big bunch of parameters or maybe none. Systems are configured via arbitrary '.ini' files in the top level of a projects directory. Within those files systems are identified with a section starting with 'system'.

Configuration example (systems of *boltzmann* package)

from File: `share/boltzmann/systems.ini`

```
[system RBM]
package = boltzmann
class = RBM

[system GRBM]
package = boltzmann
class = GRBM

[system autoencoder]
package = boltzmann
class = autoencoder
```

Optimization schedule configuration

Optimization schedule configuration sections start with 'schedule' ... there are so many parameters to explain for every single optimization … In the example below there are default configurations which still do not include all parameters that can be used in the RBM and GRBM optimization … A good starting point would be:

[1] A Practical Guide to Training Restricted Boltzmann Machines, Geoffrey Hinton

[2] Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines, KyungHyun Cho

Configuration example (schedules of *boltzmann* package)

from File: `share/boltzmann/systems.ini`

```
# metaPath optimization schedules

[schedule CD]
system GRBM:
  iterations = 1,
  iterationReset = False,
  iterationLiftLinks = False,
  updates = 100000,
  updateAlgorithm = 'CD',
  updateSamplingSteps = 1,
  updateSamplingIterations = 1,
  updateRate = 0.01,
  updateFactorWeights = 1.0,
  updateFactorHbias = 0.1,
  updateFactorVbias = 0.1,
  updateFactorVlvar = 0.01,
  minibatchSize = 1000,
  minibatchInterval = 10,
  useAdjacency = False,
  inspect = True,
  inspectFunction = 'error',
  inspectInterval = 1000,
  estimateTime = True,
  estimateTimeWait = 5.0
system RBM:
  iterations: 1,
  iterationReset: False,
  iterationLiftLinks: False,
  minibatchSize: 1000,
  minibatchInterval: 10,
  updates: 20000,
  updateAlgorithm: 'CD',
  updateSamplingSteps: 1,
  updateSamplingIterations: 1,
  updateRate: 0.1,
  updateFactorWeights: 1.0,
  updateFactorHbias: 0.1,
  updateFactorVbias: 0.1,
  useAdjacency: False,
  inspect: True,
  inspectFunction: 'error',
  inspectInterval: 1000,
  estimateTime: True,
  estimateTimeWait': 5.0
```

Plot Configuration

Plots are great! Therefore many plot packages have been implemented. The following to example plot configuration files give an overview how to use the plot packages. Plots that are defined in ‚share/base‘ can directly be accessed without namespace (For example ‚Histogram‘ instead of ‚base.Histogram‘)

Configuration example (*base plots*)

File: `share/base/systems.ini`

```
# metaPath plot configuration

[plot Histogram]
description:
  Plot all dataset values as histogram
package = histogram
class = dataHistogram
params:
  bins = 120,
  facecolor = 'lightgrey',
  edgecolor = 'black',
  histtype = 'bar',
  linewidth = 0.5

[plot Network]
package = layergraph
class = adjacencyGraph
params:
  edge_weight = 'adjacency',
  edge_zoom = 5.0,
  dpi = 300

[plot CorrelationHeatmap]
package = heatmap
class = unitRelation
params:
  relation = 'correlation',
  statistics = 100000

[plot CorrelationNetwork]
description:
  Plot correlations of units as network
type = unitrelationgraph
params:
  relation = 'correlation',
  threshold = 2.0

[plot sampleCorrelationHistogram]
description:
  Plot correlation of samples as histogram
type = samplerelationhistogram
params:
  relation = 'correlation',
  bins = 120,
  facecolor = 'lightgrey',
  edgecolor = 'black',
  histtype = 'bar',
  linewidth = 0.5

[plot sampleCorrelationHeatmap]
description:
  Plot correlation of samples as heatmap
type = samplerelationmatrix
params:
  relation = 'correlation'
```

```
[plot sampleCorrelationGraph]
description:
  Plot correlation of samples as graph
type = samplerelationgraph
params:
  relation = 'correlation',
  filter_threshold = 1.55,
  edge_zoom = 1.0
```

Configuration example (*plots of boltzmann package*)

File: `share/boltzmann/plots.ini`

```
# metaPath plot configuration

# MODEL WEIGHT GRAPH

[plot HiddenLayerGraph]
package = layergraph
class = weightGraph
params:
  node_caption = 'performance',
  edge_caption = 'none',
  edge_weight = 'weights',
  edge_threshold = 0.0,
  edge_zoom = 2.0

# DIMENSIONALITY REDUCTION VALIDATION

[plot CorrelationHeatmap]
package = heatmap
class = unitRelation
params:
  relation = "correlation",
  preprocessing = 'corrStableFilter',
  statistics = 100000

[plot CorrelationHistogram]
package = histogram
class = unitRelation
params:
  relation = "correlation",
  preprocessing = 'corrStableFilter',
  statistics = 100000

# PREDICTION OF INTERACTIONS

[plot InteractionHeatmap]
package = heatmap
class = unitRelation
params:
  relation = "causality(measure = 'performance', modify = 'setmean',
transform = '-M')",
  statistics = 100000

[plot InteractionHistogram]
package = histogram
class = unitRelation
params:
  relation = "causality(measure = 'performance', modify = 'setmean',
transform = '-M')",
  statistics = 100000

# PREDICTION OF INTERACTION DIRECTIONS

[plot DirectionHeatmap]
package = heatmap
class = unitRelation
params:
  relation = "causality(measure = 'intperformance', modify = 'setmean',
transform = '-(M - M.T)')",
  statistics = 100000

[plot DirectionHistogram]
package = histogram
class = unitRelation
params:
  relation = "causality(measure = 'intperformance', modify = 'setmean',
transform = '-(M - M.T)')",
  statistics = 100000

# PREDICTION OF INTERACTION STRENGTHS

[plot ConnectionHeatmap]
package = heatmap
class = unitRelation
params:
  relation = "causality(measure = 'extperformance', modify = 'setmean',
transform = '-(M + M.T)')",
  statistics = 100000

[plot ConnectionHistogram]
package = histogram
class = unitRelation
params:
  relation = "causality(measure = 'extperformance', modify = 'setmean',
transform = '-(M + M.T)')",
  statistics = 100000
```

Scripting

The following examples should be creatd and executed in the metapath main directory.

Scripting example (*create new model*)

File: `exampleCode1.ini`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys

def main():

    sys.path.append('./package')
    import metapath

    # create workspace and open project

    mp = metapath.open('example')

    model = mp.model(
        name     = 'sim1Gauss',
        network  = 'example.etfs(e.size = 4, tf.size = 4, s.size = 4)',
        dataset  = 'example.sim1',
        system   = 'example.GRBM',
        optimize = 'example.CD')

    mp.saveModel(model)
    mp.plot(model, 'boltzmann.HiddenLayerGraph')
```