A Bayesian model for the inference of TF activation state:

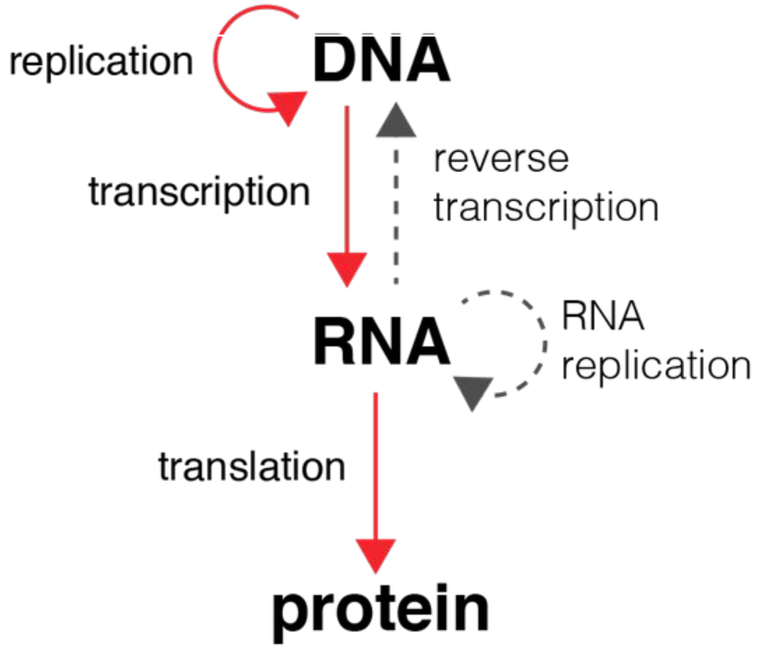
notes on implementation and preliminary results

University of Massachusetts Boston - November 9th 2018 - Argenis Arriojas

Outline

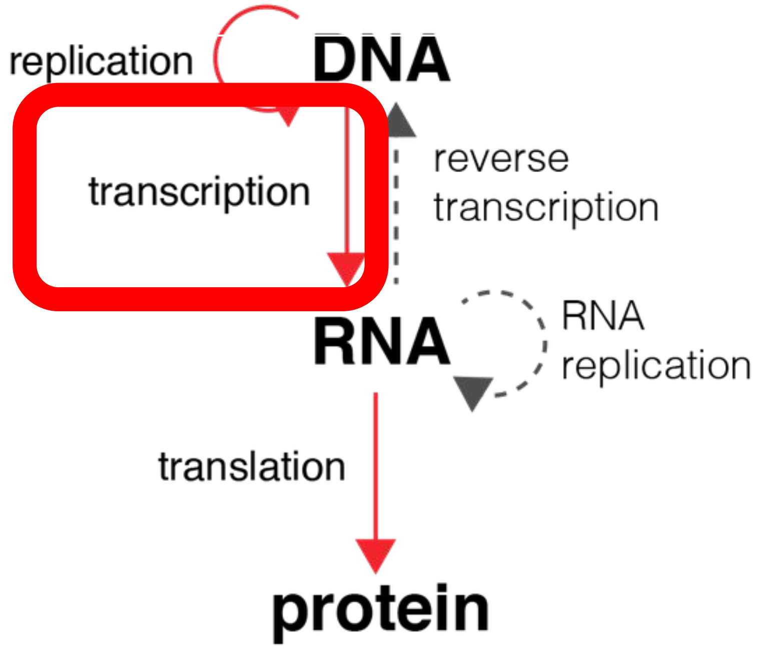
Introduction

Introduction



- Proteins are complex molecules produced from DNA
- These have many different and very specific functions within an organism
  - Antibody
  - Enzyme
  - Messenger
  - Structural component
  - Transport/storage
- Transcription factors (TFs) regulate when and when not to produce certain proteins
- This regulation depends on many factors and is harmoniously orchestrated to achieve cellular objectives

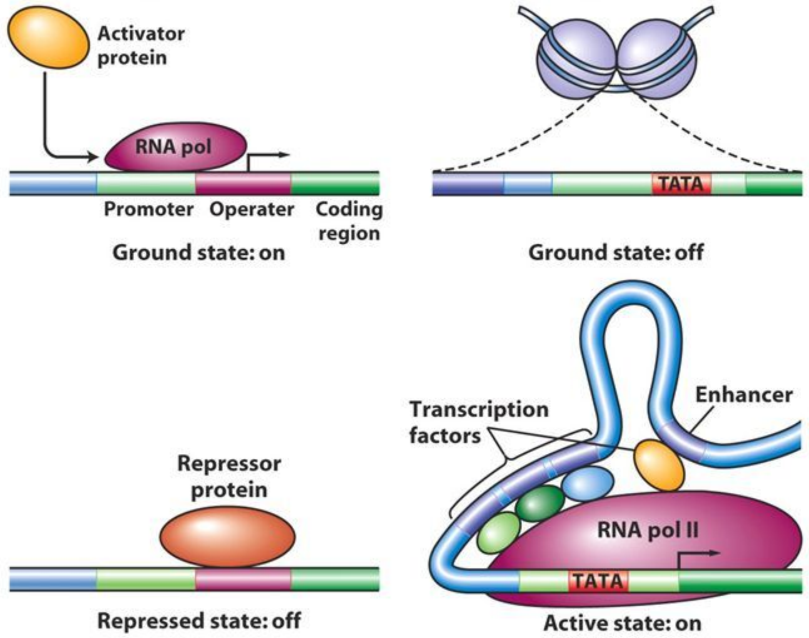
Introduction



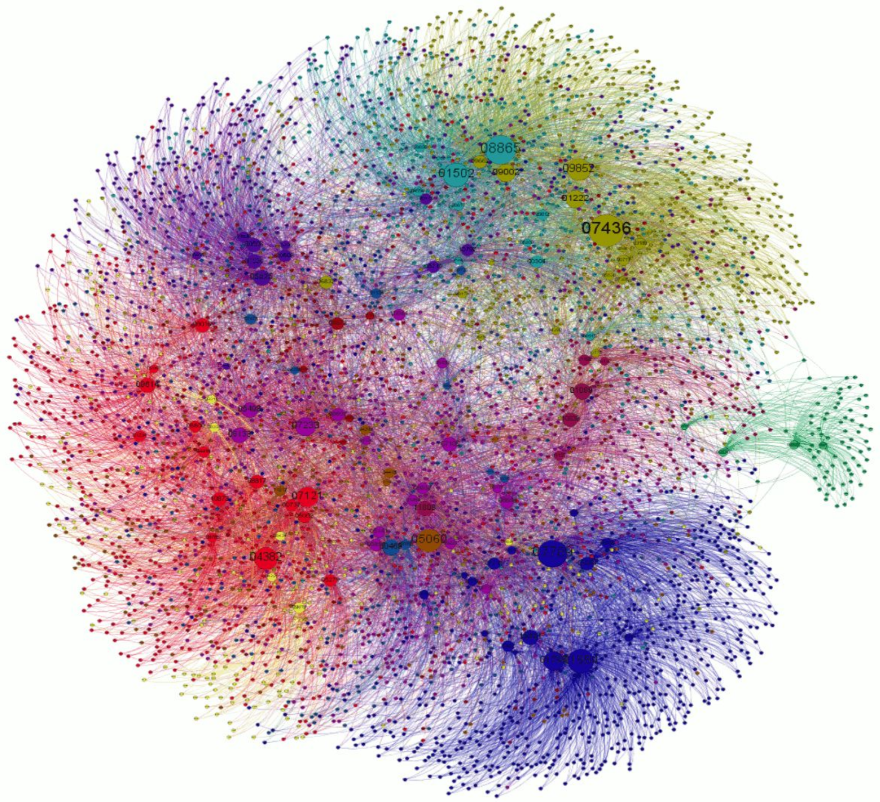
- Proteins are complex molecules produced from DNA
- These have many different and very specific functions within an organism
  - Antibody
  - Enzyme
  - Messenger
  - Structural component
  - Transport/storage
- Transcription factors (TFs) regulate when and when not to produce certain proteins
- This regulation depends on many factors and is harmoniously orchestrated to achieve cellular objectives

Transcriptional regulation

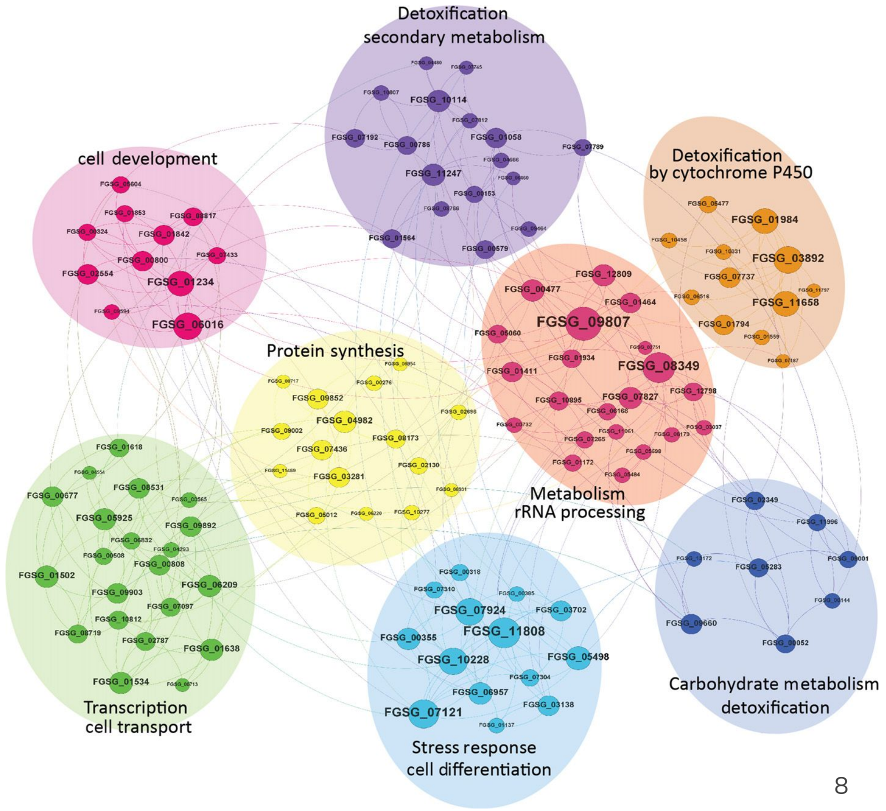
Transcription factors bind to promoter region of genes and may either activate or repress expression of a gene



Regulatory networks are complex



Regulatory networks are complex

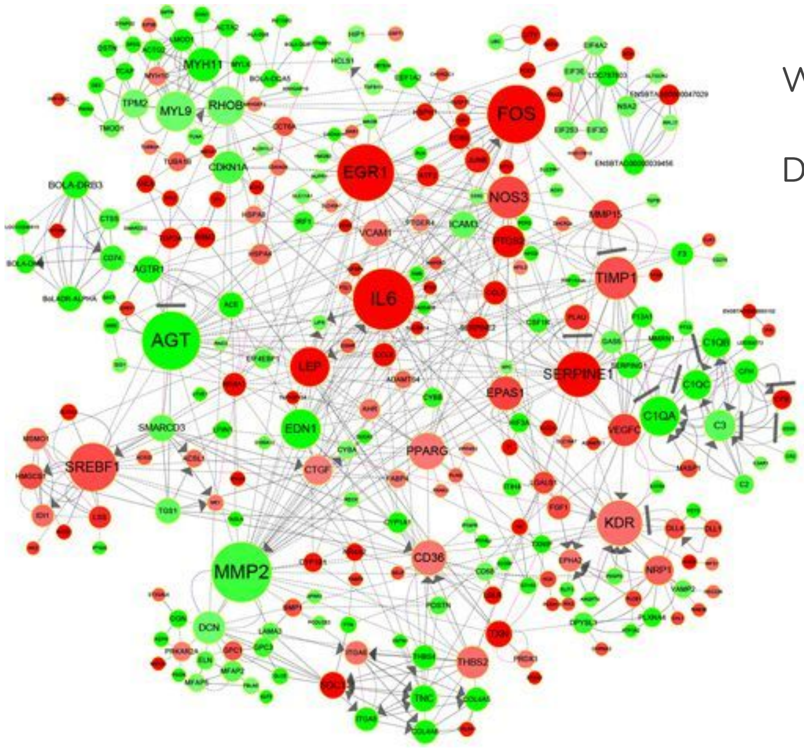


Detoxification secondary metabolism
cell development
Detoxification by cytochrome P450
Protein synthesis
Metabolism rRNA processing
Transcription cell transport
Stress response cell differentiation
Carbohydrate metabolism detoxification
FGSG_09807
FGSG_08349
FGSG_11808
FGSG_10114
FGSG_01234
FGSG_05016

Guo, L. , Zhao, G. , Xu, J. , Kistler, H. C., Gao, L. and Ma, L. (2016), Compartmentalized gene regulatory network of the pathogenic fungus Fusarium graminearum. New Phytol, 211: 527-541. doi:10.1111/nph.13912

Problem:
We want to identify relevant TFs given a cellular context

Using data to discover TFs



What data?

Differentially expressed genes

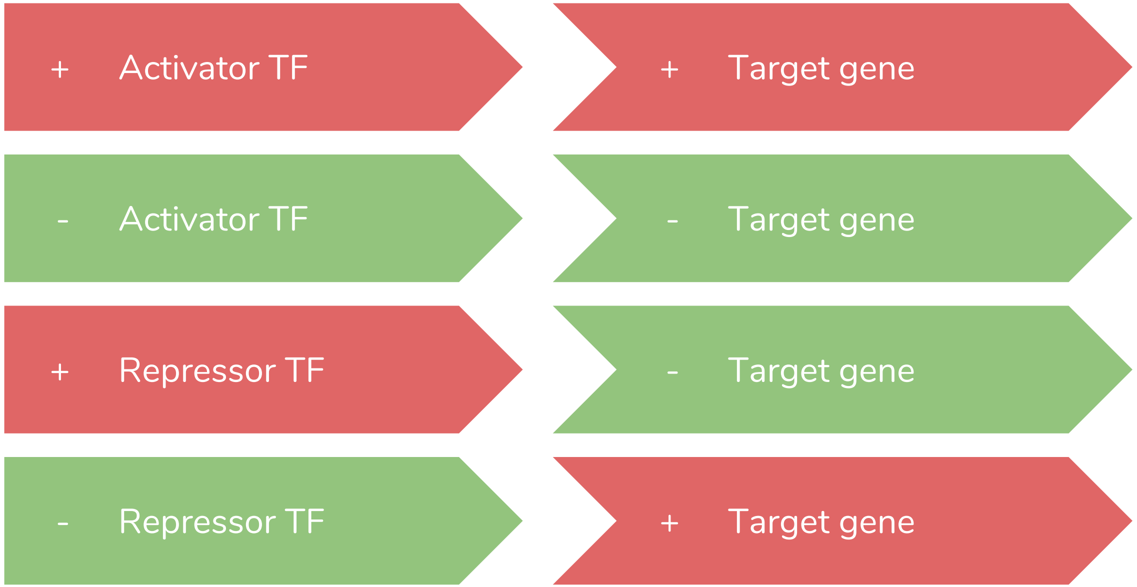
Differentially expressed genes

- Micro-arrays, RNA-seq
    - Measures gene expression levels at RNA level
    - This is a good measure of the activity of a gene
    - Contrast between two conditions (contexts)
        - Wild-type
        - Special condition

If there is a change in expression level of a gene, there may be some TF responsible for it

Differentially expressed genes

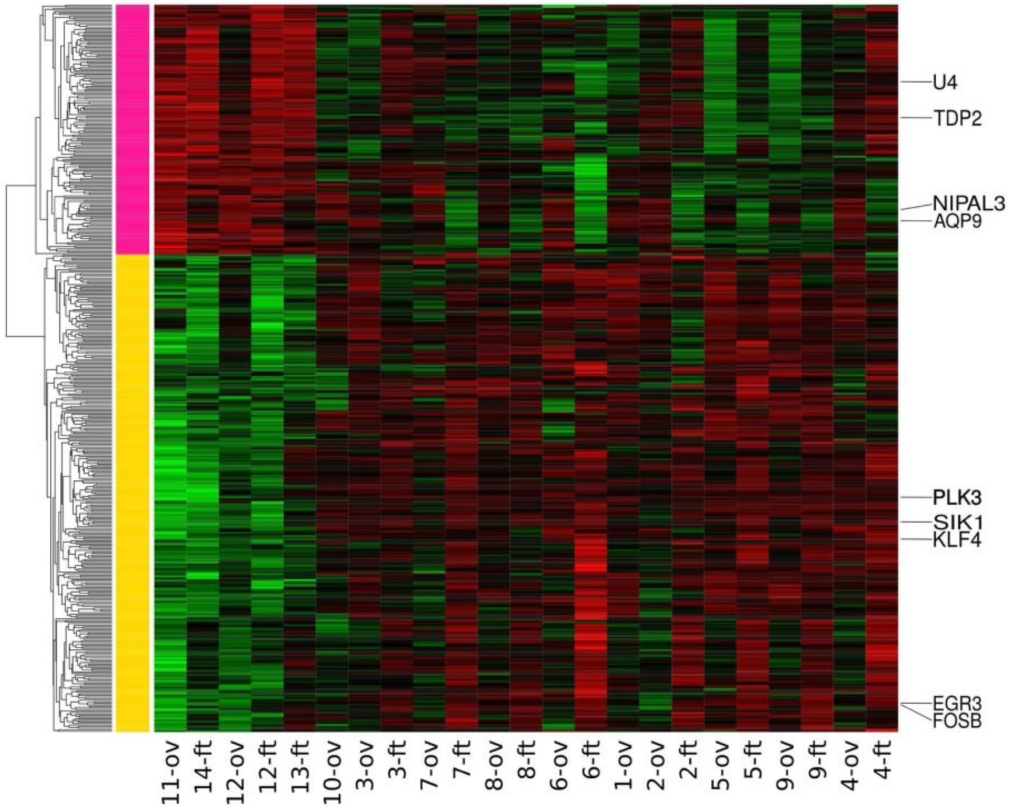
If there is a change in expression level of a gene, there may be some TF responsible for it



Differentially expressed genes

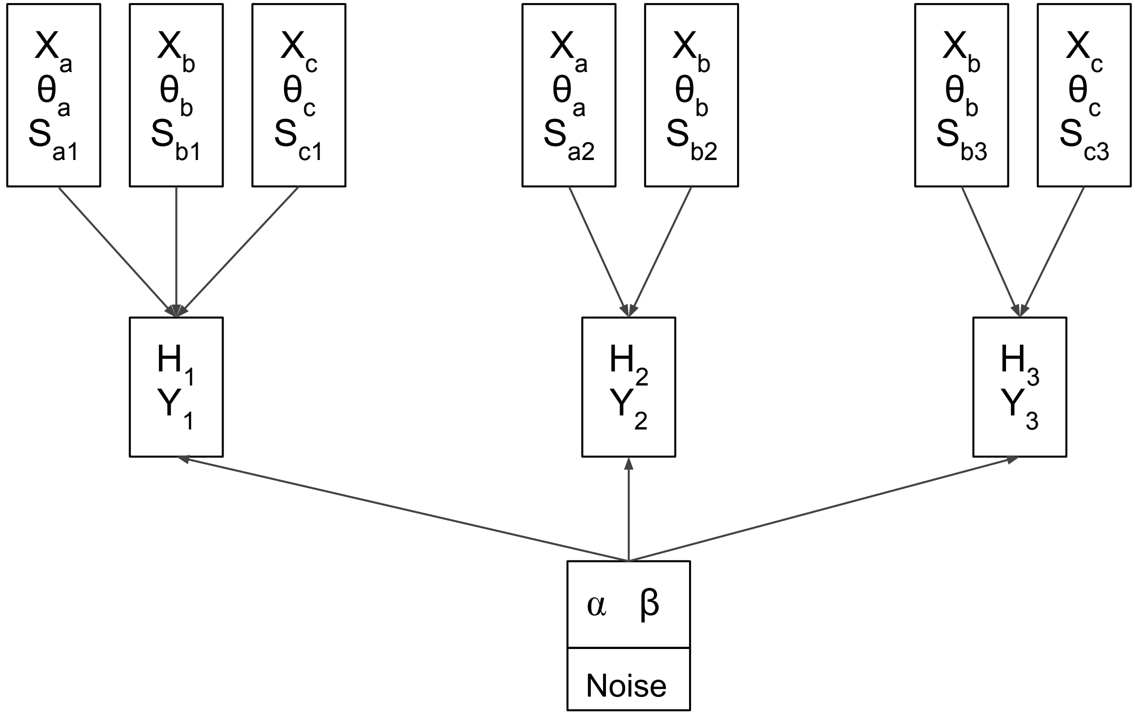
If there is a change in expression level of a gene, there may be some TF responsible for it

It is actually hard to infer from large DEG data



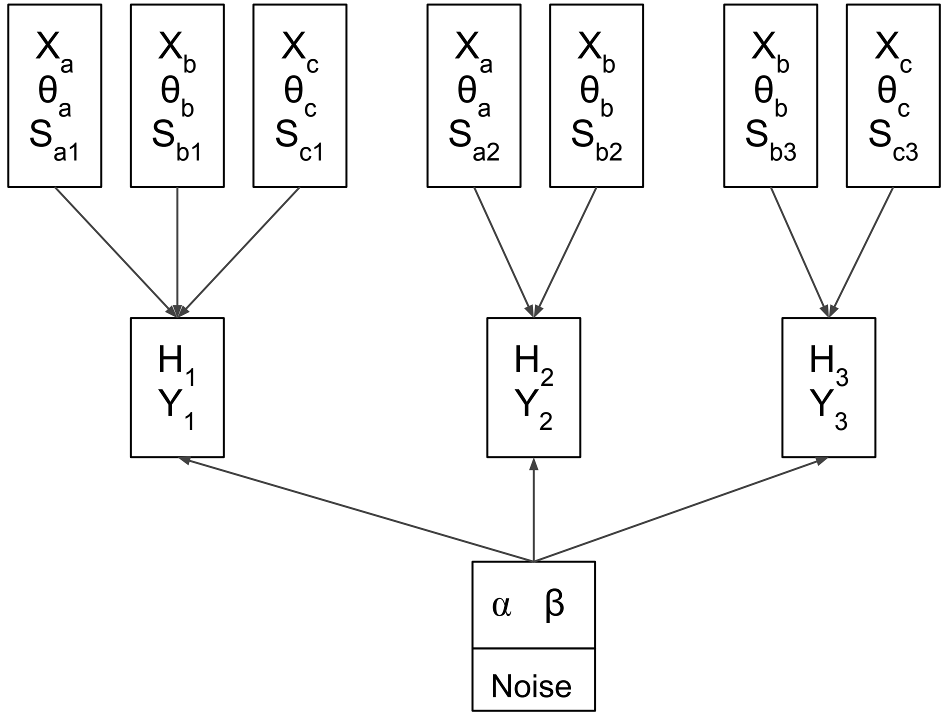
A causal inference model

A causal inference model



X_a θ_a S_{a1}
X_b θ_b S_{b1}
X_c θ_c S_{c1}
X_a θ_a S_{a2}
X_b θ_b S_{b2}
X_b θ_b S_{b3}
X_c θ_c S_{c3}
H_1 Y_1
H_2 Y_2
H_3 Y_3
α β
Noise

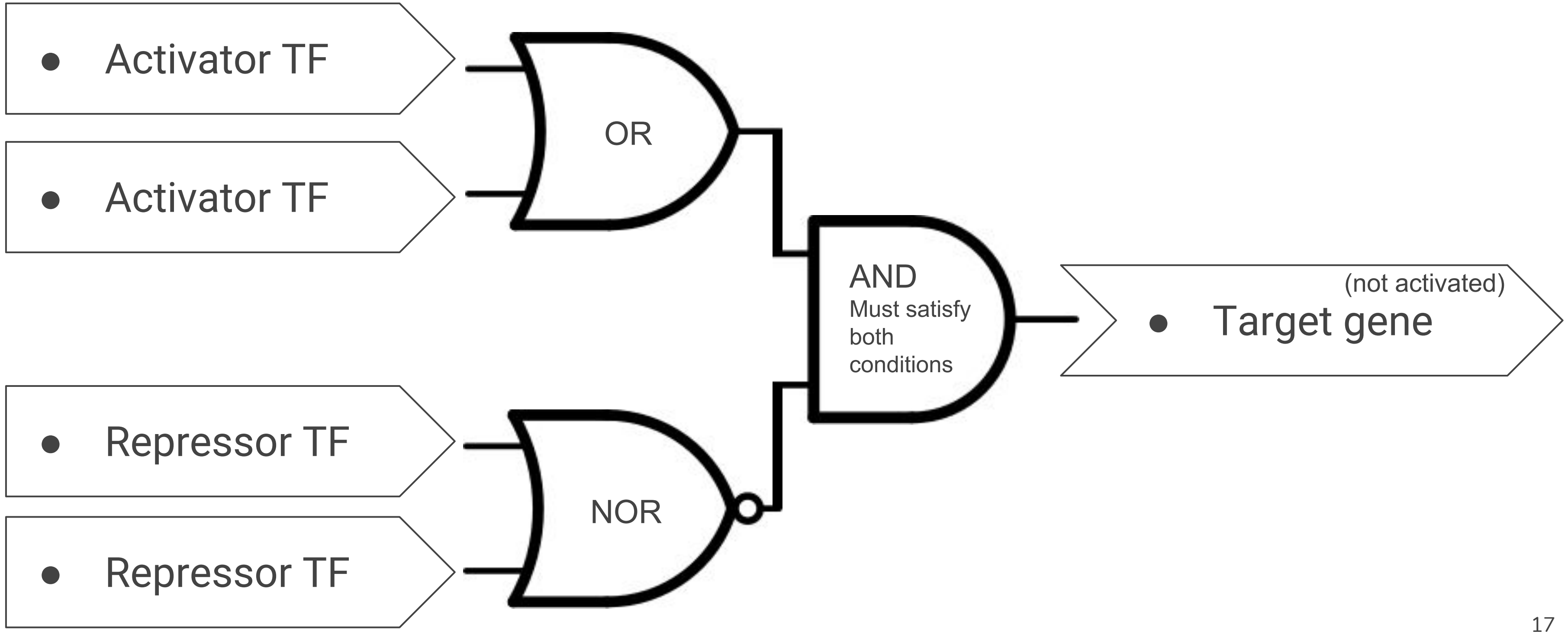
A causal inference model


X_a θ_a S_{a1}
X_b θ_b S_{b1}
X_c θ_c S_{c1}
X_a θ_a S_{a2}
X_b θ_b S_{b2}
X_b θ_b S_{b3}
X_c θ_c S_{c3}
H_1 Y_1
H_2 Y_2
H_3 Y_3
α β
Noise

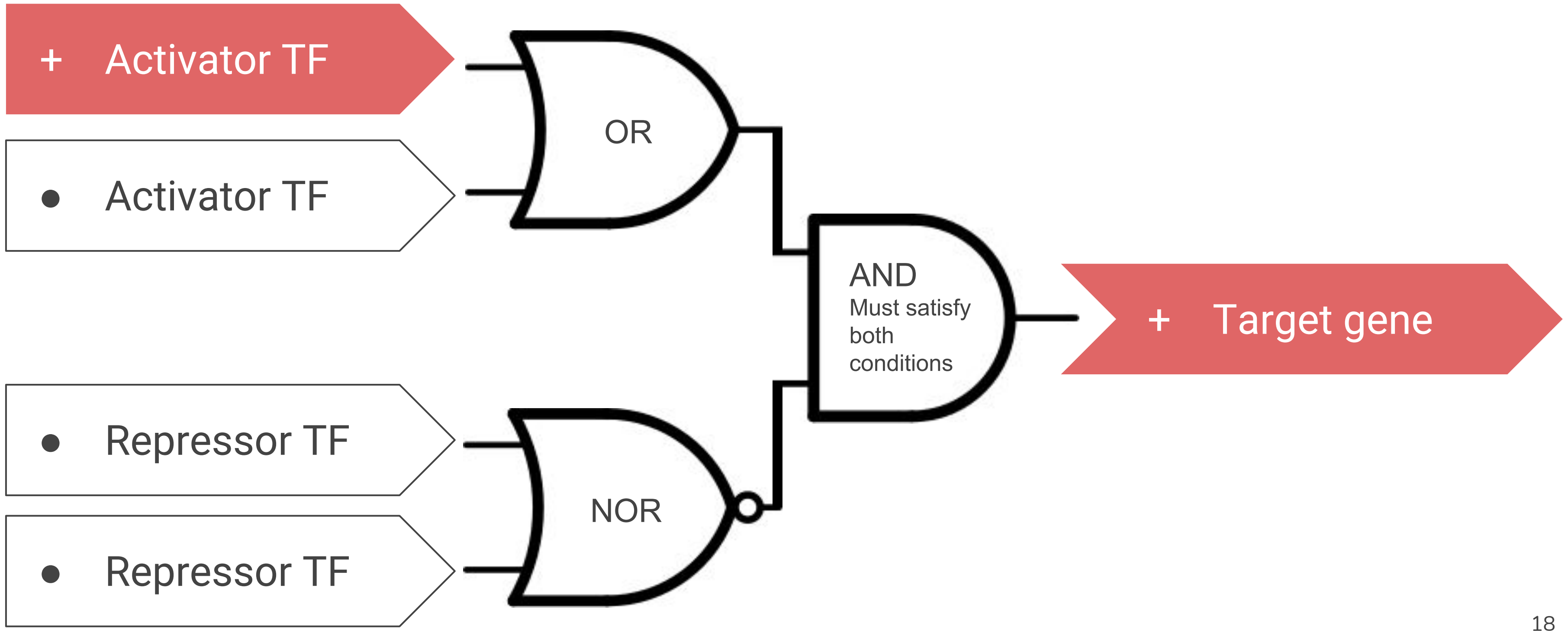
X: TF activation state {0, 1}
θ: TF activation strength
S: Mode of regulation {-1, 0, 1}
H: Hidden gene state {-1, 0, 1}
Y: Observed gene state {-1, 0, 1}
α: Observation noise parameter
β: Observation noise parameter

X ~ Binomial
θ ~ Beta
S ~ Multinomial
Y ~ Multinomial
α ~ Beta
β ~ Beta

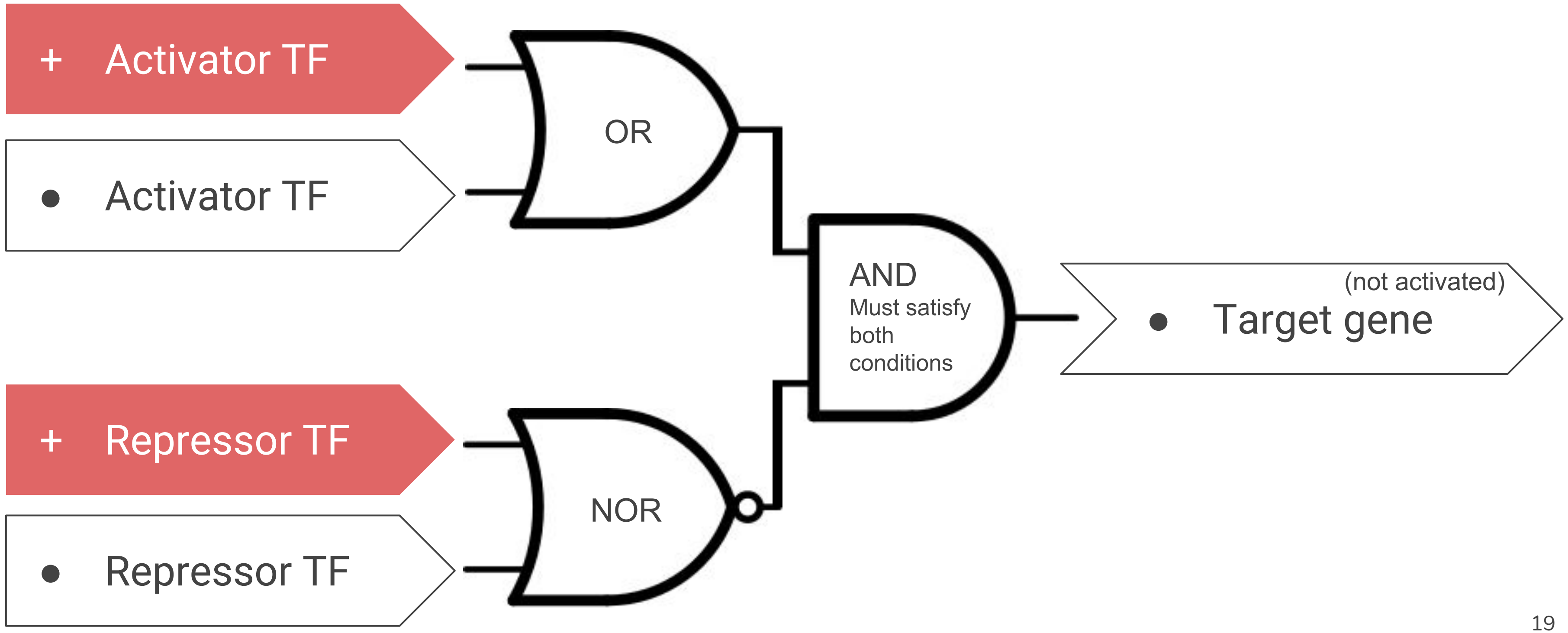
Regulatory logic - the OR-NOR model



Regulatory logic - the OR-NOR model



Regulatory logic - the OR-NOR model



Regulatory logic - the OR-NOR model

$$Y_j \sim Multinomial(P_{Y_j}^{(-1)}, P_{Y_j}^{(0)}, P_{Y_j}^{(+1)})$$

$$P_{Y_j}^{(-1)} = 1 - \prod_i (1 - X_i \theta_i S_{ij}^{(-1)})$$

$$P_{Y_j}^{(+1)} = [1 - \prod_i (1 - X_i \theta_i S_{ij}^{(+1)})] \prod_i (1 - X_i \theta_i S_{ij}^{(-1)})$$

$$P_{Y_j}^{(0)} = 1 - P_{Y_j}^{(+1)} - P_{Y_j}^{(-1)}$$

Regulatory logic - the OR-NOR model

$$Y_j \sim Multinomial(P_{Y_j}^{(-1)}, P_{Y_j}^{(0)}, P_{Y_j}^{(+1)})$$

$$P_{Y_j}^{(-1)} = 1 - \prod_i (1 - X_i \theta_i S_{ij}^{(-1)})$$

$$P_{Y_j}^{(+1)} = [1 - \prod_i (1 - X_i \theta_i S_{ij}^{(+1)})] \prod_i (1 - X_i \theta_i S_{ij}^{(-1)})$$

$$P_{Y_j}^{(0)} = \prod_i (1 - X_i \theta_i S_{ij}^{(+1)}) \prod_i (1 - X_i \theta_i S_{ij}^{(-1)})$$

Regulatory logic - the OR-NOR model

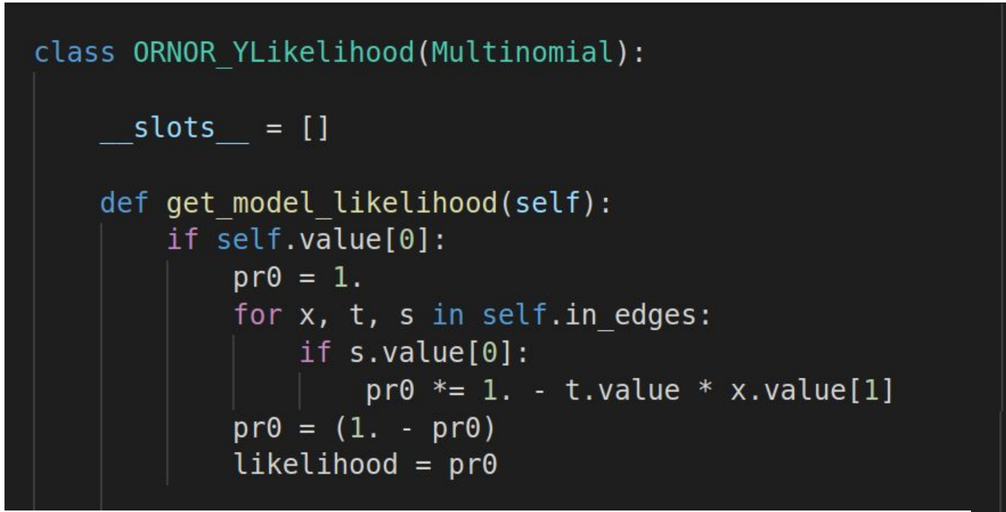
$$Y_j \sim Multinomial(P_{Y_j}^{(0)}, P_{Y_j}^{(1)}, P_{Y_j}^{(2)})$$

$$P_{Y_j}^{(0)} = 1 - \prod_i (1 - X_i \theta_i S_{ij}^{(0)})$$

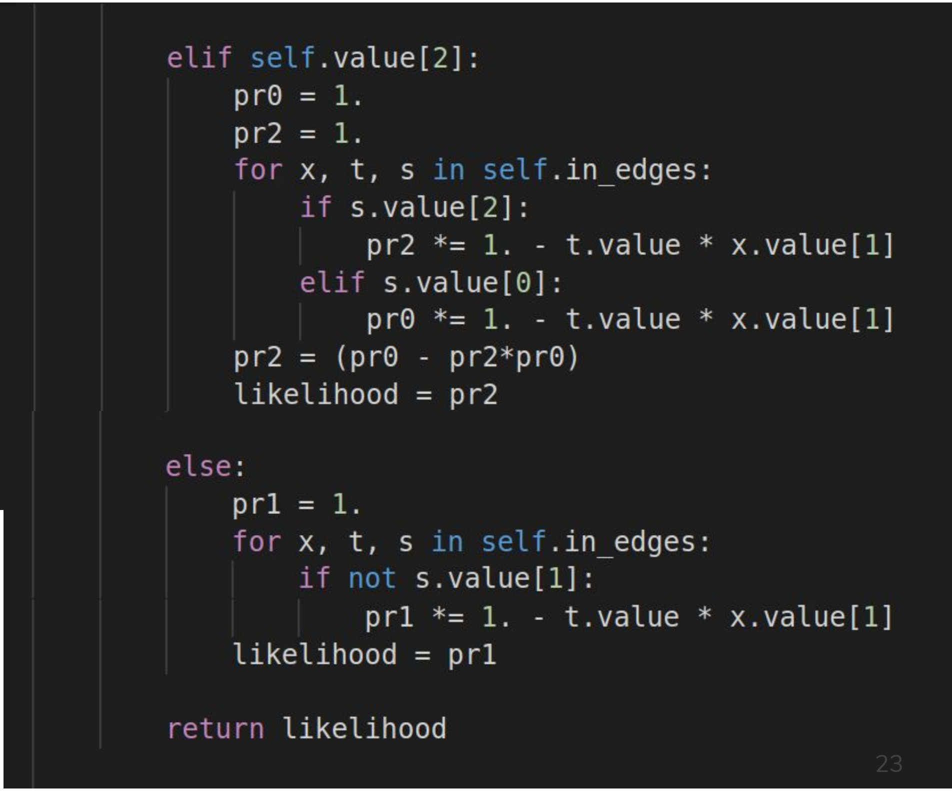
$$P_{Y_j}^{(2)} = [1 - \prod_i (1 - X_i \theta_i S_{ij}^{(2)})] \prod_i (1 - X_i \theta_i S_{ij}^{(0)})$$

$$P_{Y_j}^{(1)} = \prod_i (1 - X_i \theta_i S_{ij}^{(2)}) \prod_i (1 - X_i \theta_i S_{ij}^{(0)})$$

Regulatory logic - the OR-NOR model



```python
class ORNOR_YLikelihood(Multinomial):

    __slots__ = []

    def get_model_likelihood(self):
        if self.value[0]:
            pr0 = 1.
            for x, t, s in self.in_edges:
                if s.value[0]:
                    pr0 *= 1. - t.value * x.value[1]
            pr0 = (1. - pr0)
            likelihood = pr0
```



```python
        elif self.value[2]:
            pr0 = 1.
            pr2 = 1.
            for x, t, s in self.in_edges:
                if s.value[2]:
                    pr2 *= 1. - t.value * x.value[1]
                elif s.value[0]:
                    pr0 *= 1. - t.value * x.value[1]
            pr2 = (pr0 - pr2*pr0)
            likelihood = pr2

        else:
            pr1 = 1.
            for x, t, s in self.in_edges:
                if not s.value[1]:
                    pr1 *= 1. - t.value * x.value[1]
            likelihood = pr1

        return likelihood
```

Considering noise in DEG data

If there was no noise:

H_1
Y_1

Conditional probability table P(Y|H)

P(Y|H)	H = -1	H = 0	H = 1
Y = -1	1	0	0
Y = 0	0	1	0
Y = 1	0	0	1

Y ~ Multinomial

Considering noise in DEG data

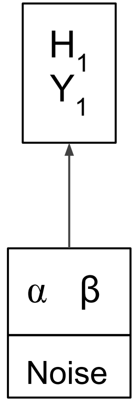
Incorporate noise:

Conditional probability table P(Y|H)

P(Y|H)	H = -1	H = 0	H = 1
Y = -1	1 - α - β	α	β
Y = 0	α	1 - 2α	α
Y = 1	β	α	1 - α - β

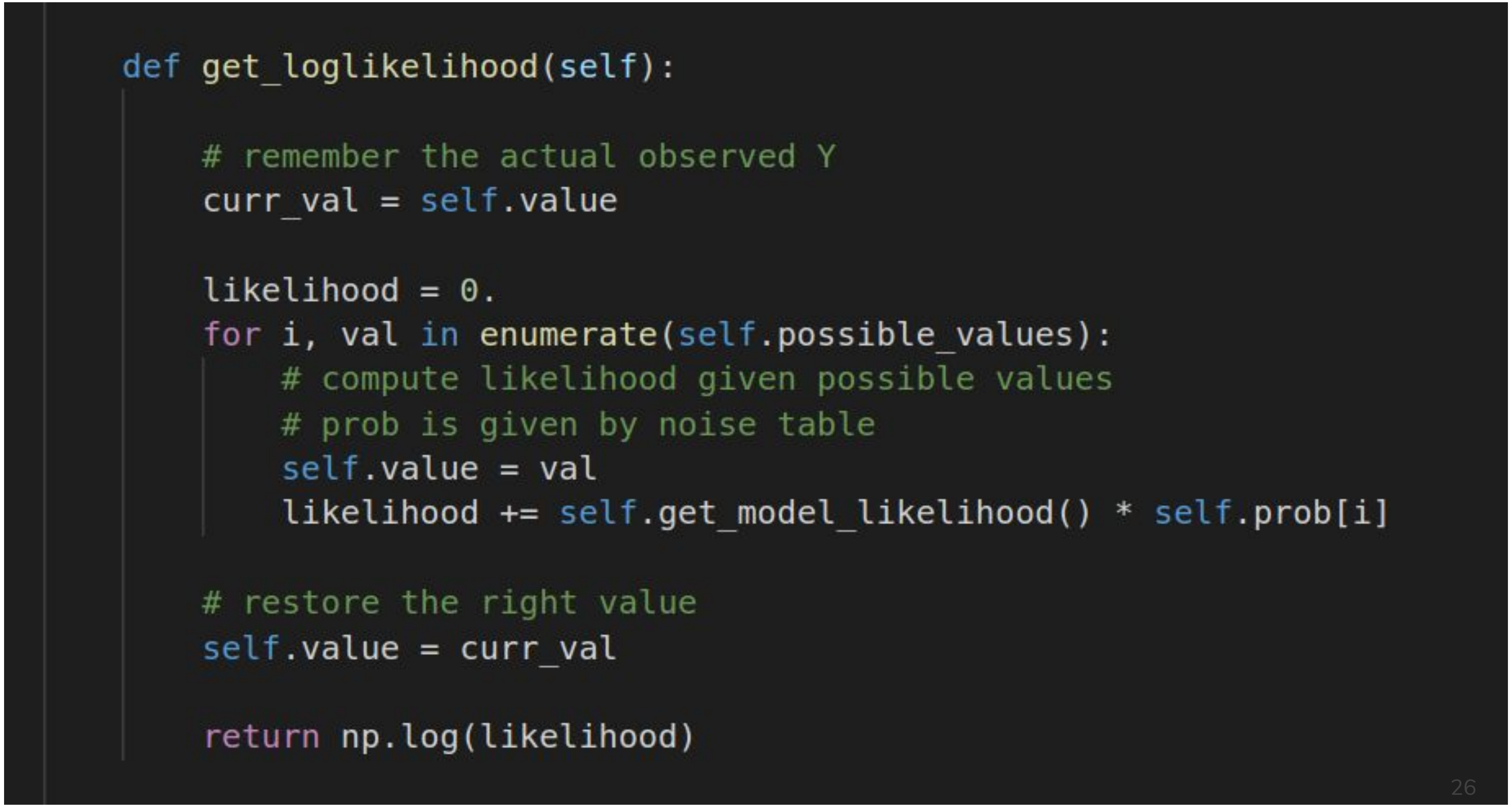
$\alpha > \beta$

?False positive rate = α
?False negative rate = (α + β) / 2



Y ~ Multinomial
α ~ Beta
β ~ Beta

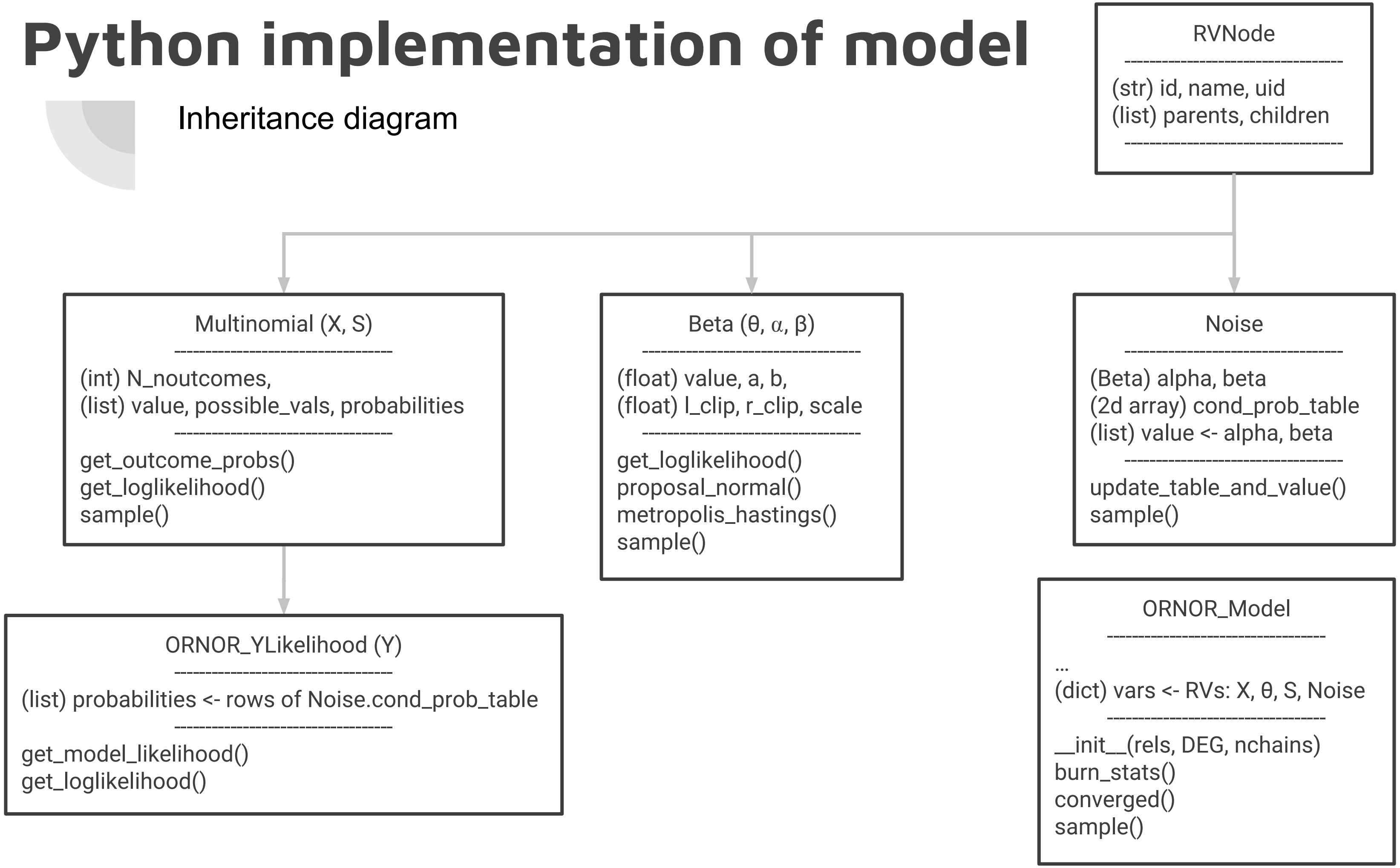
Considering noise in DEG data



```
def get_loglikelihood(self):

    # remember the actual observed Y
    curr_val = self.value

    likelihood = 0.
    for i, val in enumerate(self.possible_values):
        # compute likelihood given possible values
        # prob is given by noise table
        self.value = val
        likelihood += self.get_model_likelihood() * self.prob[i]

    # restore the right value
    self.value = curr_val

    return np.log(likelihood)
```

Python implementation of model

Inheritance diagram



RVNode
(str) id, name, uid
(list) parents, children

Multinomial (X, S)
(int) N_noutcomes,
(list) value, possible_vals, probabilities
get_outcome_probs()
get_loglikelihood()
sample()

Beta (θ, α, β)
(float) value, a, b,
(float) l_clip, r_clip, scale
get_loglikelihood()
proposal_normal()
metropolis_hastings()
sample()

Noise
(Beta) alpha, beta
(2d array) cond_prob_table
(list) value <- alpha, beta
update_table_and_value()
sample()

ORNOR_YLikelihood (Y)
(list) probabilities <- rows of Noise.cond_prob_table
get_model_likelihood()
get_loglikelihood()

ORNOR_Model
...
(dict) vars <- RVs: X, θ, S, Noise
__init__(rels, DEG, nchains)
burn_stats()
converged()
sample()

Results on real data

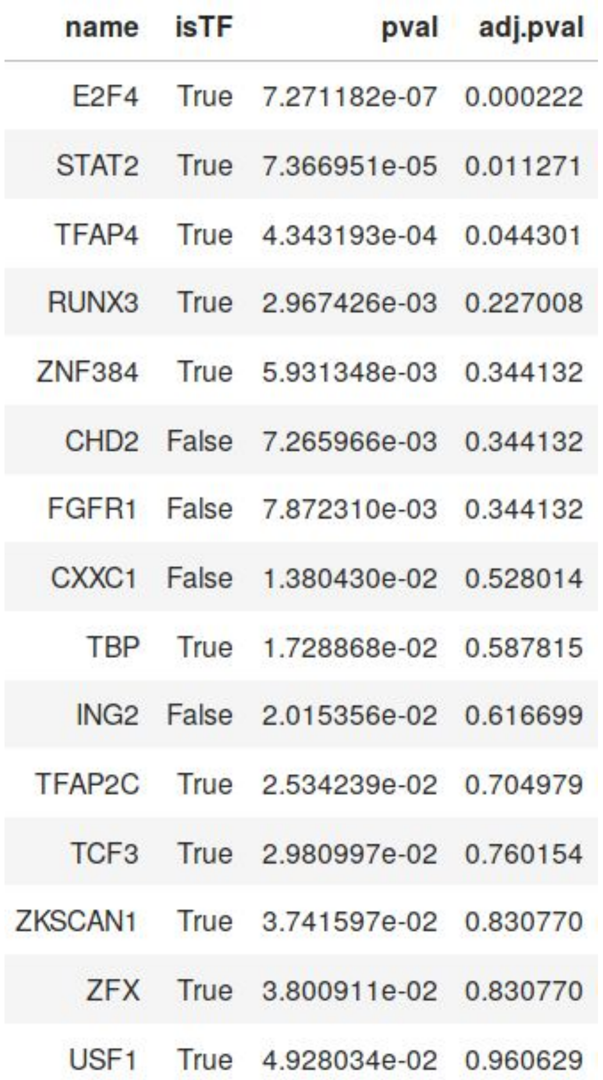
The experiment:

- TGFβ and CXCL12 treated cells: N1 cells which were derived from a stromal nodule of benign prostatic hyperplasia, exhibit a fibroblastic morphology, express fibroblastic markers vimentin and calponin, and demonstrate secretion and proliferation profiles consistent with aging primary prostate fibroblasts
- ?? Embryonic Mouse Hypothalamus Cell Line N1 (mHypoE-N1)
- The TGFβ/TGFβR and CXCL12/CXCR4 axes induce myofibroblast phenoconversion independently through Smads and MEK/Erk proteins, respectively

Can we identify TFs that are activated (or deactivated) by TGFβ and CXCL12 signalling?

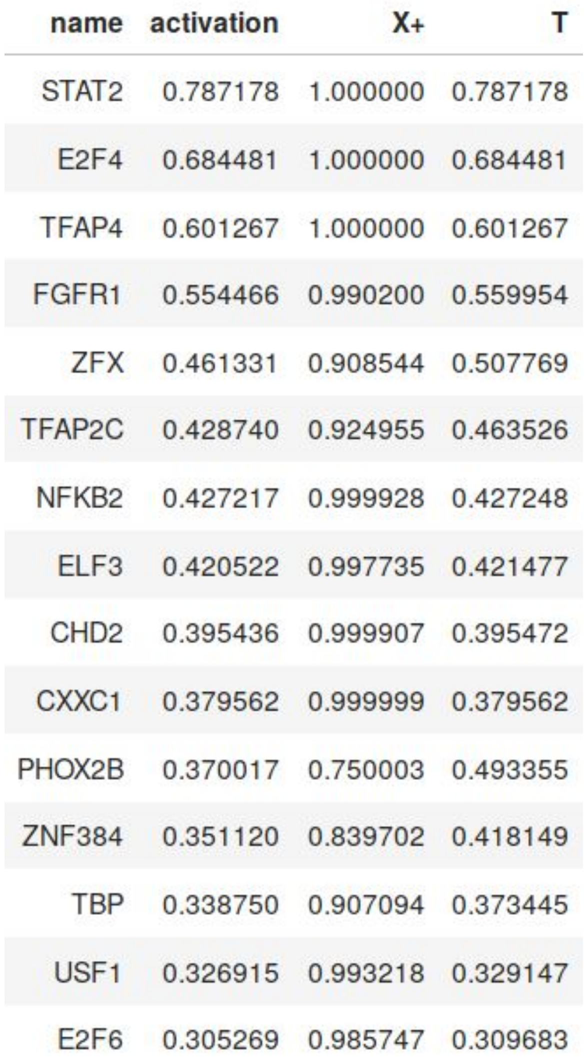
Fisher's test enrichment results

These results are from Causal Inference Engine developed in lab: Corey, Yasaman, Saman


TGFβ+ Fisher's test enrichment results

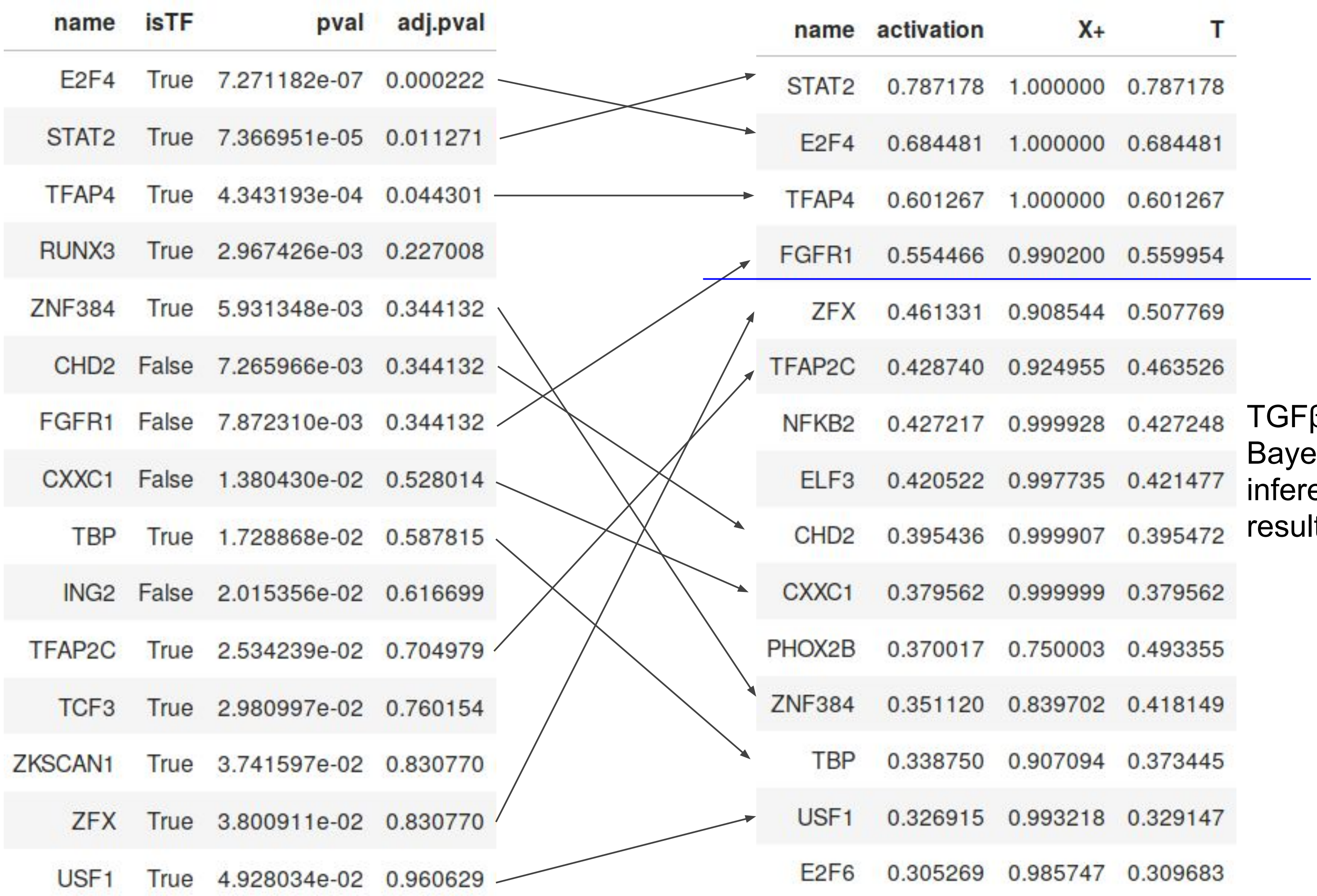
name	isTF	pval	adj.pval
E2F4	True	7.271182e-07	0.000222
STAT2	True	7.366951e-05	0.011271
TFAP4	True	4.343193e-04	0.044301
RUNX3	True	2.967426e-03	0.227008
ZNF384	True	5.931348e-03	0.344132
CHD2	False	7.265966e-03	0.344132
FGFR1	False	7.872310e-03	0.344132
CXXC1	False	1.380430e-02	0.528014
TBP	True	1.728868e-02	0.587815
ING2	False	2.015356e-02	0.616699
TFAP2C	True	2.534239e-02	0.704979
TCF3	True	2.980997e-02	0.760154
ZKSCAN1	True	3.741597e-02	0.830770
ZFX	True	3.800911e-02	0.830770
USF1	True	4.928034e-02	0.960629

Bayesian inference results



name	activation	X+	T
STAT2	0.787178	1.000000	0.787178
E2F4	0.684481	1.000000	0.684481
TFAP4	0.601267	1.000000	0.601267
FGFR1	0.554466	0.990200	0.559954
ZFX	0.461331	0.908544	0.507769
TFAP2C	0.428740	0.924955	0.463526
NFKB2	0.427217	0.999928	0.427248
ELF3	0.420522	0.997735	0.421477
CHD2	0.395436	0.999907	0.395472
CXXC1	0.379562	0.999999	0.379562
PHOX2B	0.370017	0.750003	0.493355
ZNF384	0.351120	0.839702	0.418149
TBP	0.338750	0.907094	0.373445
USF1	0.326915	0.993218	0.329147
E2F6	0.305269	0.985747	0.309683

TGFβ+ Bayesian inference results



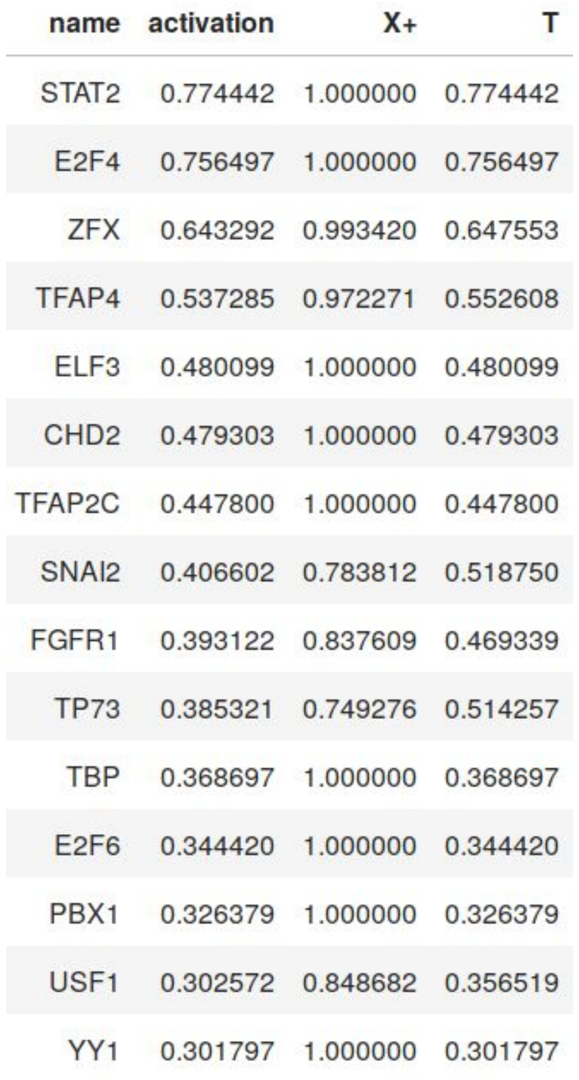
TGFβ+ Fisher's test enrichment results

name	isTF	pval	adj.pval
E2F4	True	7.271182e-07	0.000222
STAT2	True	7.366951e-05	0.011271
TFAP4	True	4.343193e-04	0.044301
RUNX3	True	2.967426e-03	0.227008
ZNF384	True	5.931348e-03	0.344132
CHD2	False	7.265966e-03	0.344132
FGFR1	False	7.872310e-03	0.344132
CXXC1	False	1.380430e-02	0.528014
TBP	True	1.728868e-02	0.587815
ING2	False	2.015356e-02	0.616699
TFAP2C	True	2.534239e-02	0.704979
TCF3	True	2.980997e-02	0.760154
ZKSCAN1	True	3.741597e-02	0.830770
ZFX	True	3.800911e-02	0.830770
USF1	True	4.928034e-02	0.960629

TGFβ+ Bayesian inference results

name	activation	X+	T
STAT2	0.787178	1.000000	0.787178
E2F4	0.684481	1.000000	0.684481
TFAP4	0.601267	1.000000	0.601267
FGFR1	0.554466	0.990200	0.559954
ZFX	0.461331	0.908544	0.507769
TFAP2C	0.428740	0.924955	0.463526
NFKB2	0.427217	0.999928	0.427248
ELF3	0.420522	0.997735	0.421477
CHD2	0.395436	0.999907	0.395472
CXXC1	0.379562	0.999999	0.379562
PHOX2B	0.370017	0.750003	0.493355
ZNF384	0.351120	0.839702	0.418149
TBP	0.338750	0.907094	0.373445
USF1	0.326915	0.993218	0.329147
E2F6	0.305269	0.985747	0.309683

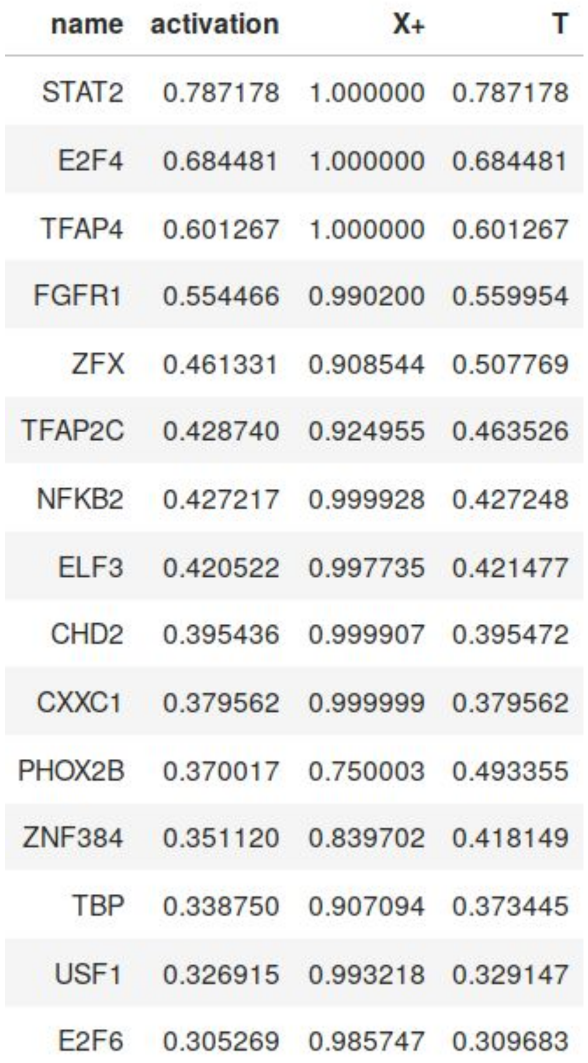
Bayesian inference results CXCL12+



name	activation	X+	T
STAT2	0.774442	1.000000	0.774442
E2F4	0.756497	1.000000	0.756497
ZFX	0.643292	0.993420	0.647553
TFAP4	0.537285	0.972271	0.552608
ELF3	0.480099	1.000000	0.480099
CHD2	0.479303	1.000000	0.479303
TFAP2C	0.447800	1.000000	0.447800
SNAI2	0.406602	0.783812	0.518750
FGFR1	0.393122	0.837609	0.469339
TP73	0.385321	0.749276	0.514257
TBP	0.368697	1.000000	0.368697
E2F6	0.344420	1.000000	0.344420
PBX1	0.326379	1.000000	0.326379
USF1	0.302572	0.848682	0.356519
YY1	0.301797	1.000000	0.301797

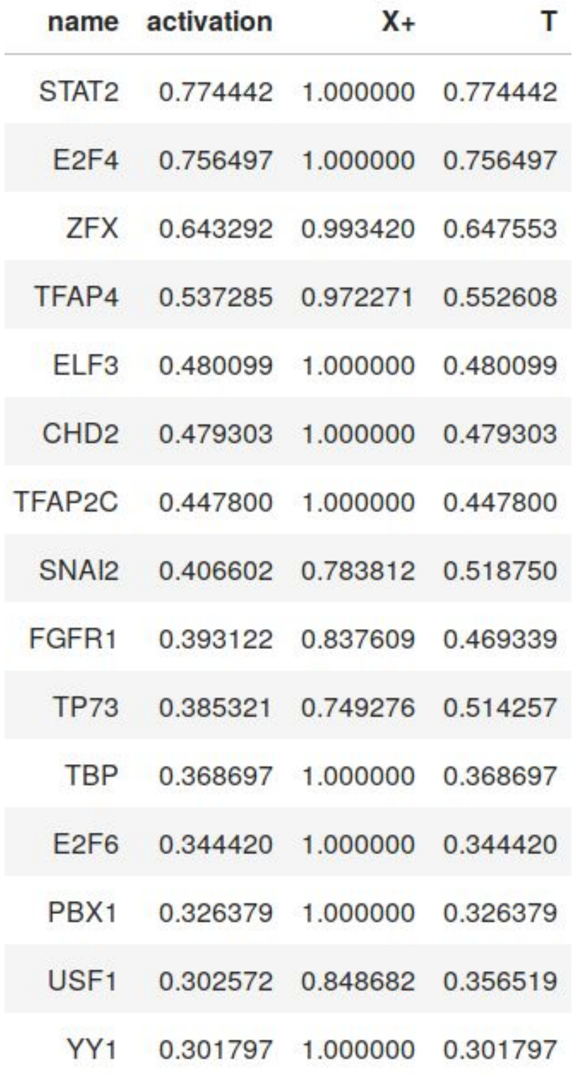
CXCL12+ Bayesian inference results

TGFβ+ Bayesian inference results



name	activation	X+	T
STAT2	0.787178	1.000000	0.787178
E2F4	0.684481	1.000000	0.684481
TFAP4	0.601267	1.000000	0.601267
FGFR1	0.554466	0.990200	0.559954
ZFX	0.461331	0.908544	0.507769
TFAP2C	0.428740	0.924955	0.463526
NFKB2	0.427217	0.999928	0.427248
ELF3	0.420522	0.997735	0.421477
CHD2	0.395436	0.999907	0.395472
CXXC1	0.379562	0.999999	0.379562
PHOX2B	0.370017	0.750003	0.493355
ZNF384	0.351120	0.839702	0.418149
TBP	0.338750	0.907094	0.373445
USF1	0.326915	0.993218	0.329147
E2F6	0.305269	0.985747	0.309683

CXCL12+ Bayesian inference results



name	activation	X+	T
STAT2	0.774442	1.000000	0.774442
E2F4	0.756497	1.000000	0.756497
ZFX	0.643292	0.993420	0.647553
TFAP4	0.537285	0.972271	0.552608
ELF3	0.480099	1.000000	0.480099
CHD2	0.479303	1.000000	0.479303
TFAP2C	0.447800	1.000000	0.447800
SNAI2	0.406602	0.783812	0.518750
FGFR1	0.393122	0.837609	0.469339
TP73	0.385321	0.749276	0.514257
TBP	0.368697	1.000000	0.368697
E2F6	0.344420	1.000000	0.344420
PBX1	0.326379	1.000000	0.326379
USF1	0.302572	0.848682	0.356519
YY1	0.301797	1.000000	0.301797

TGFβ+ Bayesian inference results

name	activation	X+	T
STAT2	0.787178	1.000000	0.787178
E2F4	0.684481	1.000000	0.684481
TFAP4	0.601267	1.000000	0.601267
FGFR1	0.554466	0.990200	0.559954
ZFX	0.461331	0.908544	0.507769
TFAP2C	0.428740	0.924955	0.463526
NFKB2	0.427217	0.999928	0.427248
ELF3	0.420522	0.997735	0.421477
CHD2	0.395436	0.999907	0.395472
CXXC1	0.379562	0.999999	0.379562
PHOX2B	0.370017	0.750003	0.493355
ZNF384	0.351120	0.839702	0.418149
TBP	0.338750	0.907094	0.373445
USF1	0.326915	0.993218	0.329147
E2F6	0.305269	0.985747	0.309683

CXCL12+ Bayesian inference results

name	activation	X+	T
STAT2	0.774442	1.000000	0.774442
E2F4	0.756497	1.000000	0.756497
ZFX	0.643292	0.993420	0.647553
TFAP4	0.537285	0.972271	0.552608
ELF3	0.480099	1.000000	0.480099
CHD2	0.479303	1.000000	0.479303
TFAP2C	0.447800	1.000000	0.447800
SNAI2	0.406602	0.783812	0.518750
FGFR1	0.393122	0.837609	0.469339
TP73	0.385321	0.749276	0.514257
TBP	0.368697	1.000000	0.368697
E2F6	0.344420	1.000000	0.344420
PBX1	0.326379	1.000000	0.326379
USF1	0.302572	0.848682	0.356519
YY1	0.301797	1.000000	0.301797

Improving performance

Python is an interpreted language

- Code is not pre-compiled
- Many uncertainties for the code interpreter
    - i.e. data types for variables
- Limited room for performance improvement
    - Many modules like Numpy are written in lower level code for performance improvement
    - First approach is to take advantage of highly optimized modules like Numpy

For the best performance we would need to use lower level programming like C/C++

However, there is a middle ground available for python code. Cython brings the opportunity to refactor key parts of Python code that are computationally intensive.

Cython

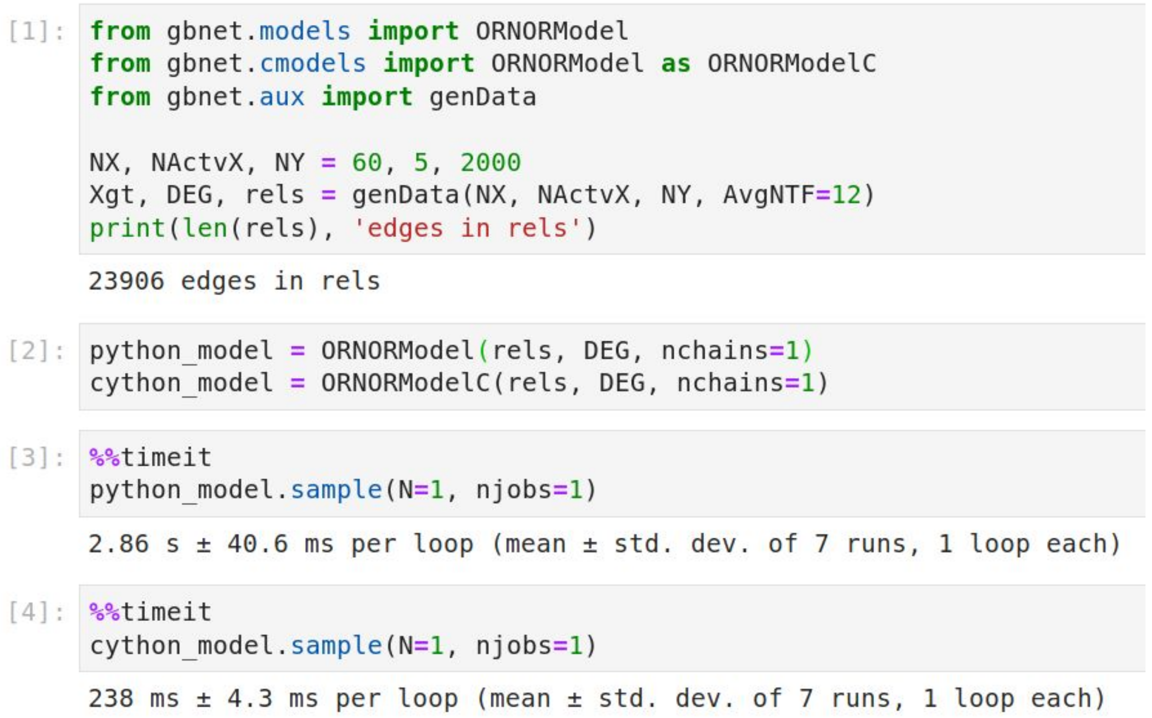
Some advantages:

- Reuse Python code
- Can use C libraries like GNU Scientific Library (GSL)
- Generates C/C++ compilable code
- Offers tools to identify computing bottlenecks, so we fix those first

Beware:

- It's not immediately magical

Performance comparison

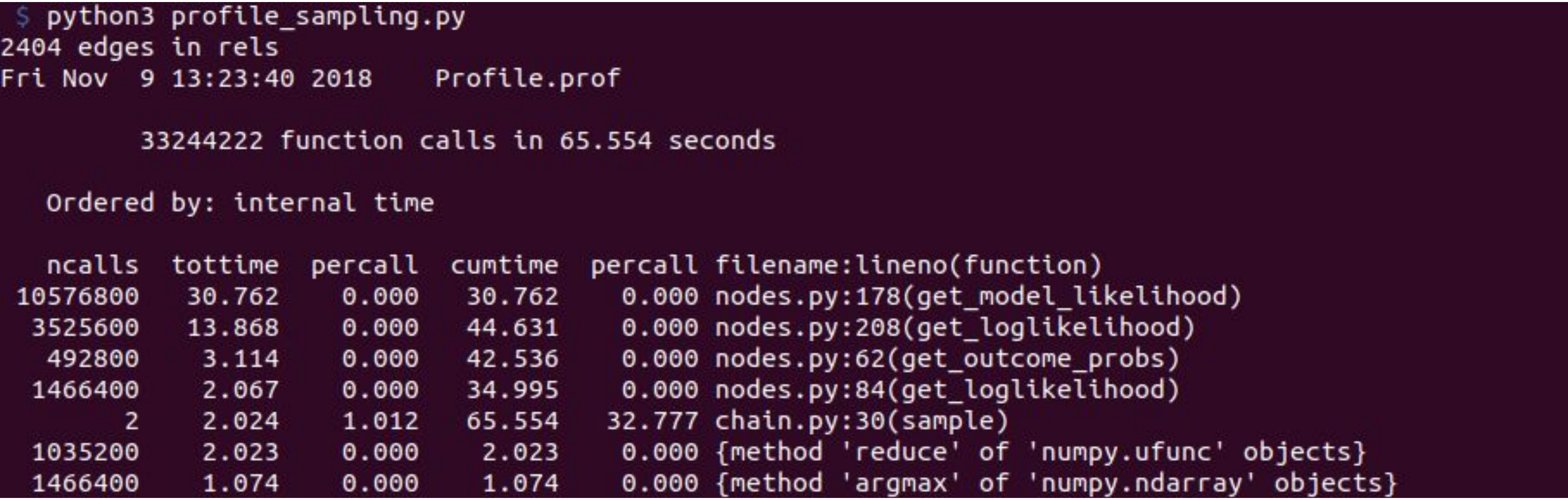


```
[1]: from gbnet.models import ORNORModel
     from gbnet.cmodels import ORNORModel as ORNORModelC
     from gbnet.aux import genData

     NX, NActvX, NY = 60, 5, 2000
     Xgt, DEG, rels = genData(NX, NActvX, NY, AvgNTF=12)
     print(len(rels), 'edges in rels')

     23906 edges in rels

[2]: python_model = ORNORModel(rels, DEG, nchains=1)
     cython_model = ORNORModelC(rels, DEG, nchains=1)

[3]: %%timeit
     python_model.sample(N=1, njobs=1)

     2.86 s ± 40.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

[4]: %%timeit
     cython_model.sample(N=1, njobs=1)

     238 ms ± 4.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Cython version is ~ 12 faster

Code profiling

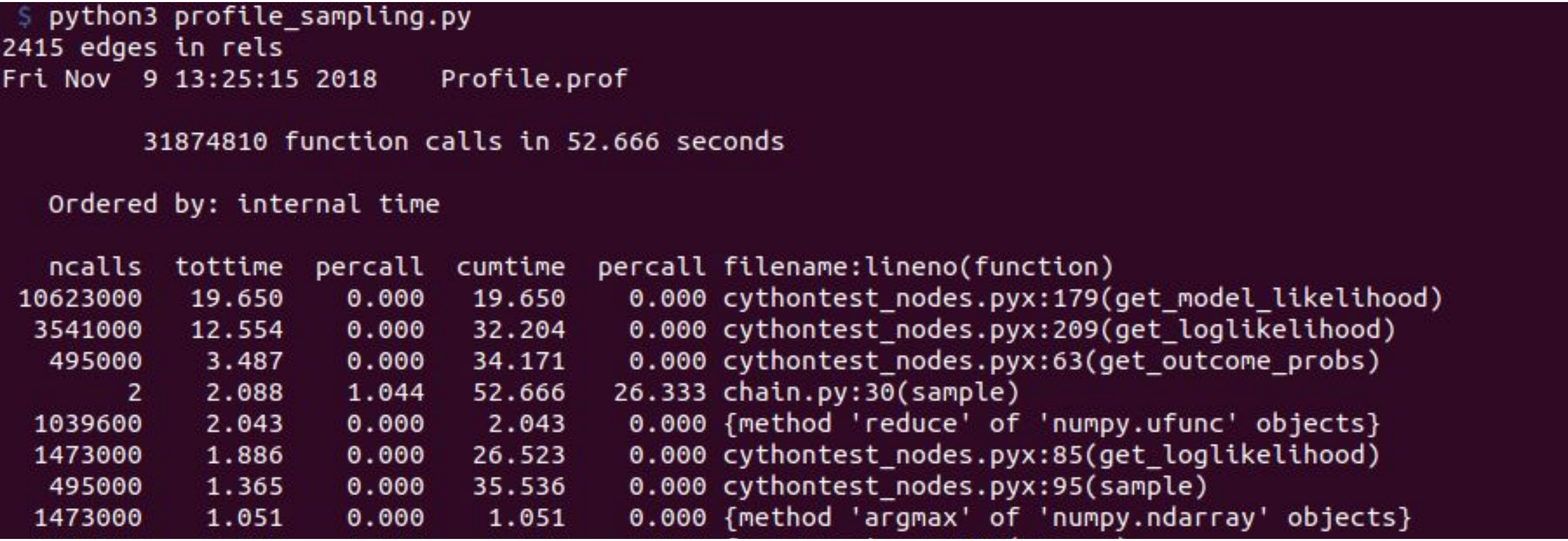
Original nodes.py file



```
 $ python3 profile_sampling.py
2404 edges in rels
Fri Nov  9 13:23:40 2018    Profile.prof

         33244222 function calls in 65.554 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 10576800   30.762    0.000   30.762    0.000 nodes.py:178(get_model_likelihood)
  3525600   13.868    0.000   44.631    0.000 nodes.py:208(get_loglikelihood)
   492800    3.114    0.000   42.536    0.000 nodes.py:62(get_outcome_probs)
  1466400    2.067    0.000   34.995    0.000 nodes.py:84(get_loglikelihood)
        2    2.024    1.012   65.554   32.777 chain.py:30(sample)
  1035200    2.023    0.000    2.023    0.000 {method 'reduce' of 'numpy.ufunc' objects}
  1466400    1.074    0.000    1.074    0.000 {method 'argmax' of 'numpy.ndarray' objects}
```

Code profiling

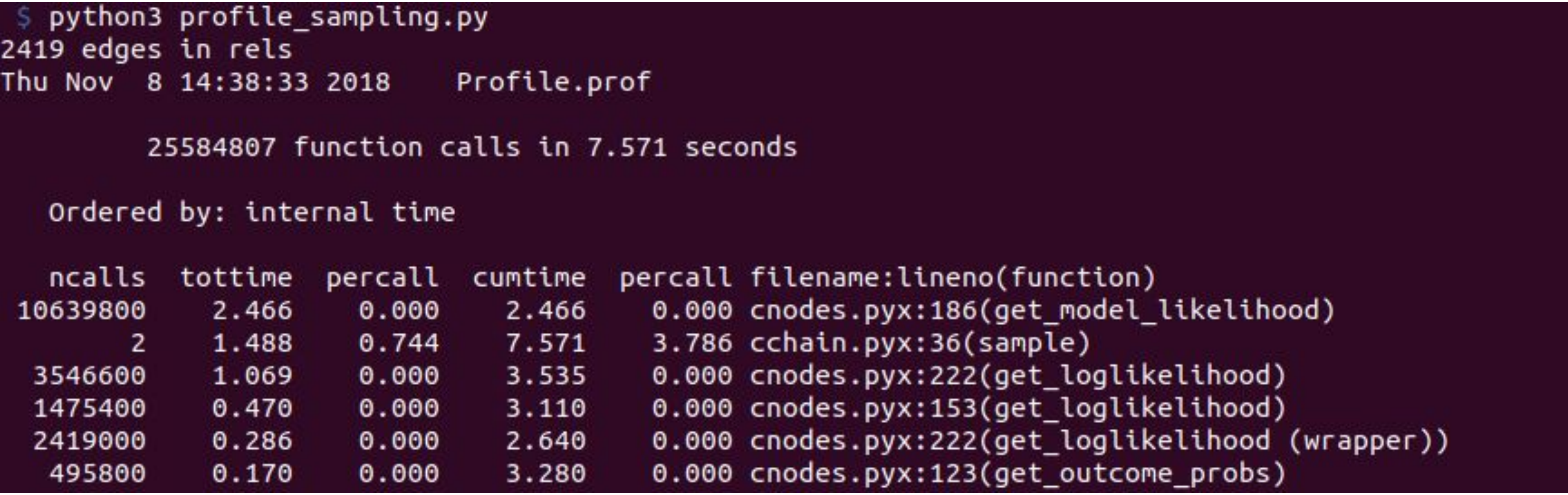
Only cythonize nodes.py file



```
$ python3 profile_sampling.py
2415 edges in rels
Fri Nov  9 13:25:15 2018    Profile.prof

         31874810 function calls in 52.666 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 10623000   19.650    0.000   19.650    0.000 cythontest_nodes.pyx:179(get_model_likelihood)
  3541000   12.554    0.000   32.204    0.000 cythontest_nodes.pyx:209(get_loglikelihood)
   495000    3.487    0.000   34.171    0.000 cythontest_nodes.pyx:63(get_outcome_probs)
        2    2.088    1.044   52.666   26.333 chain.py:30(sample)
  1039600    2.043    0.000    2.043    0.000 {method 'reduce' of 'numpy.ufunc' objects}
  1473000    1.886    0.000   26.523    0.000 cythontest_nodes.pyx:85(get_loglikelihood)
   495000    1.365    0.000   35.536    0.000 cythontest_nodes.pyx:95(sample)
  1473000    1.051    0.000    1.051    0.000 {method 'argmax' of 'numpy.ndarray' objects}
```

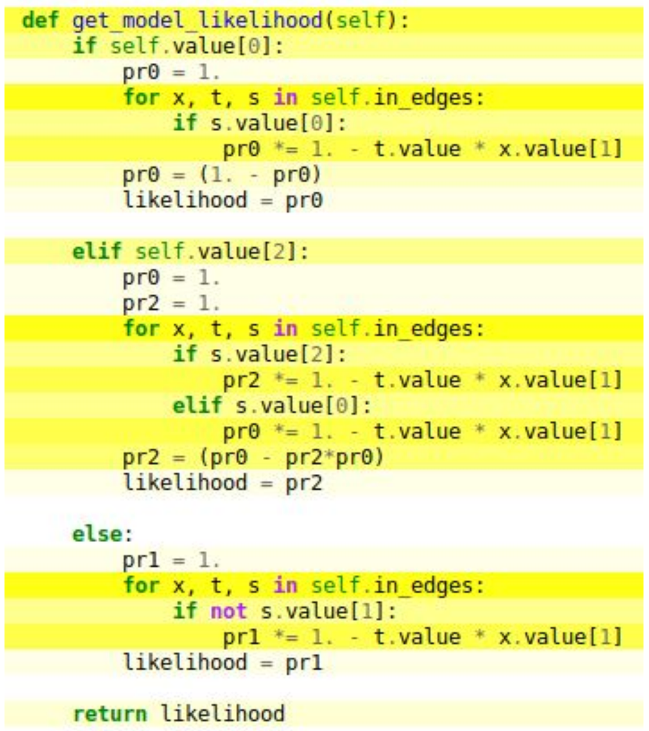
Code profiling

Craft most classes in nodes.py, also cythonize other files



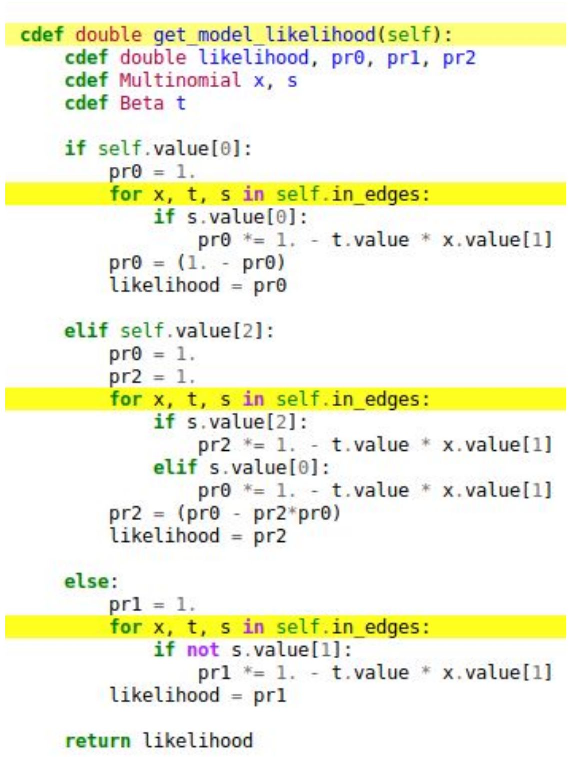
```
 $ python3 profile_sampling.py
2419 edges in rels
Thu Nov  8 14:38:33 2018    Profile.prof

         25584807 function calls in 7.571 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 10639800    2.466    0.000    2.466    0.000 cnodes.pyx:186(get_model_likelihood)
        2    1.488    0.744    7.571    3.786 cchain.pyx:36(sample)
  3546600    1.069    0.000    3.535    0.000 cnodes.pyx:222(get_loglikelihood)
  1475400    0.470    0.000    3.110    0.000 cnodes.pyx:153(get_loglikelihood)
  2419000    0.286    0.000    2.640    0.000 cnodes.pyx:222(get_loglikelihood (wrapper))
   495800    0.170    0.000    3.280    0.000 cnodes.pyx:123(get_outcome_probs)
```

Cython code annotation



```
def get_model_likelihood(self):
    if self.value[0]:
        pr0 = 1.
        for x, t, s in self.in_edges:
            if s.value[0]:
                pr0 *= 1. - t.value * x.value[1]
        pr0 = (1. - pr0)
        likelihood = pr0

    elif self.value[2]:
        pr0 = 1.
        pr2 = 1.
        for x, t, s in self.in_edges:
            if s.value[2]:
                pr2 *= 1. - t.value * x.value[1]
            elif s.value[0]:
                pr0 *= 1. - t.value * x.value[1]
        pr2 = (pr0 - pr2*pr0)
        likelihood = pr2

    else:
        pr1 = 1.
        for x, t, s in self.in_edges:
            if not s.value[1]:
                pr1 *= 1. - t.value * x.value[1]
        likelihood = pr1

    return likelihood
```

Yellow lines represent Python overhead. We aim at reducing this overhead.



```
cdef double get_model_likelihood(self):
    cdef double likelihood, pr0, pr1, pr2
    cdef Multinomial x, s
    cdef Beta t

    if self.value[0]:
        pr0 = 1.
        for x, t, s in self.in_edges:
            if s.value[0]:
                pr0 *= 1. - t.value * x.value[1]
        pr0 = (1. - pr0)
        likelihood = pr0

    elif self.value[2]:
        pr0 = 1.
        pr2 = 1.
        for x, t, s in self.in_edges:
            if s.value[2]:
                pr2 *= 1. - t.value * x.value[1]
            elif s.value[0]:
                pr0 *= 1. - t.value * x.value[1]
        pr2 = (pr0 - pr2*pr0)
        likelihood = pr2

    else:
        pr1 = 1.
        for x, t, s in self.in_edges:
            if not s.value[1]:
                pr1 *= 1. - t.value * x.value[1]
        likelihood = pr1

    return likelihood
```

What’s next

- Test our inference model against available DEG data
    - Harmonizome - http://amp.pharm.mssm.edu/Harmonizome/
    - CLUE.io - https://clue.io/cmap
- Incorporate prior knowledge on regulatory interactions
    - Use text mining to extract interactions
    - Use available public networks
- Implement the present work into C/C++
    - Seek the best possible performance