PeakSQL

Dynamic deep learning database for genomics

PeakSQL

Dynamic deep learning database for genomics

Or

Some machine learning buzzwords

PeakSQL

Dynamic deep learning database for genomics

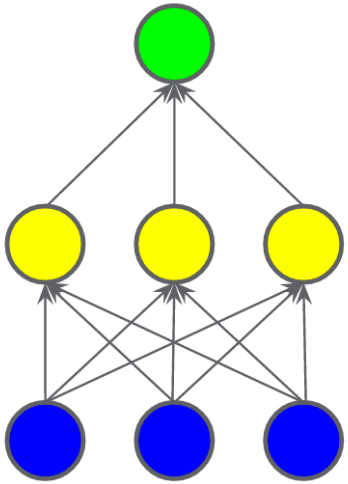
Or

Some machine learning buzzwords

Or

Do we really need another database?

Machine learning

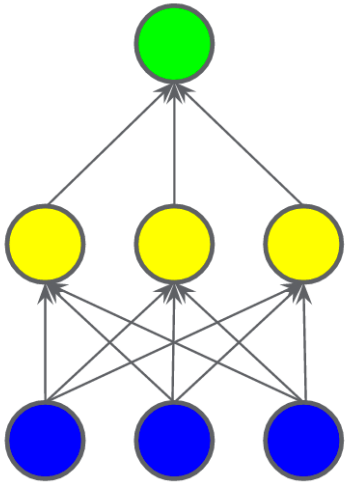


Output

Hidden Layer

Input

Machine learning



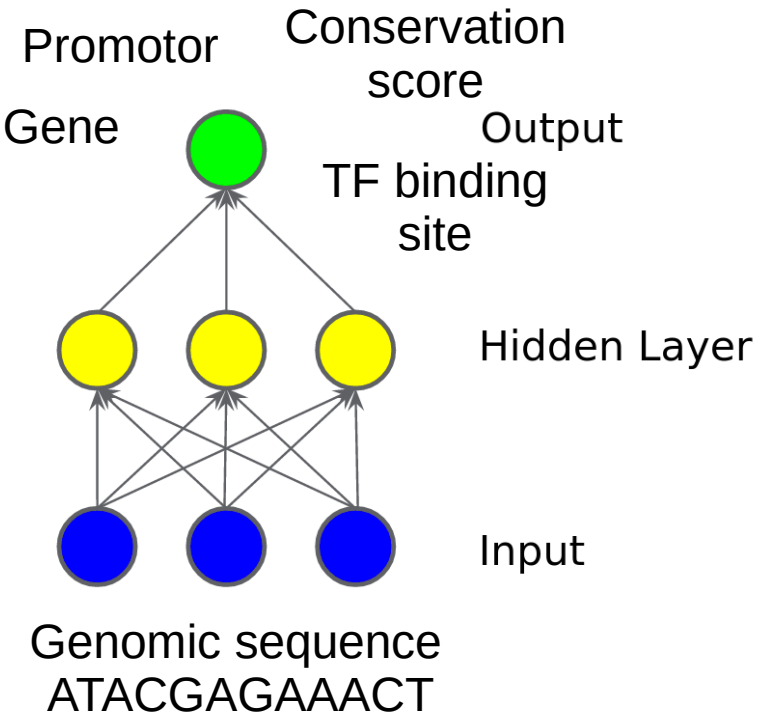
Output

Hidden Layer

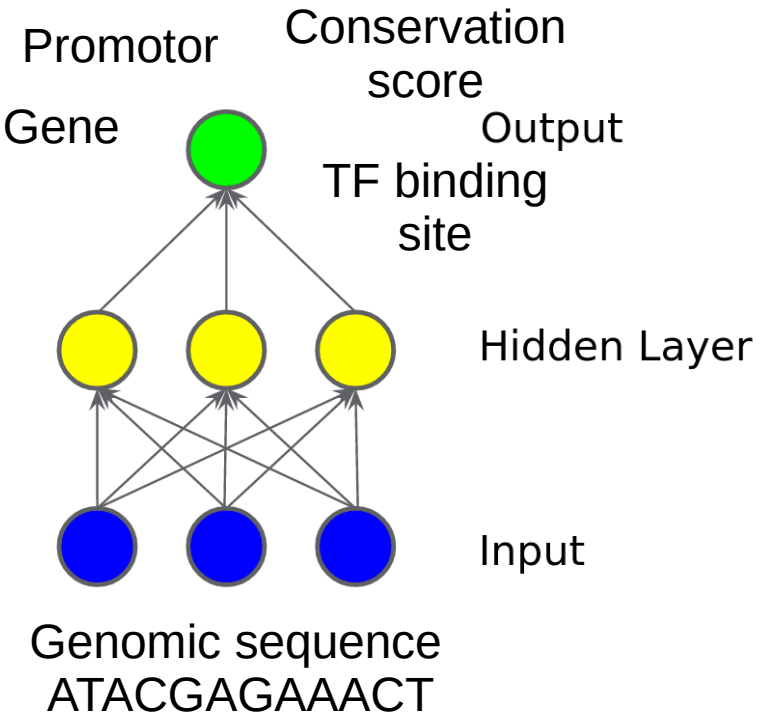
Input

Genomic sequence
ATACGAGAAACT

Machine learning

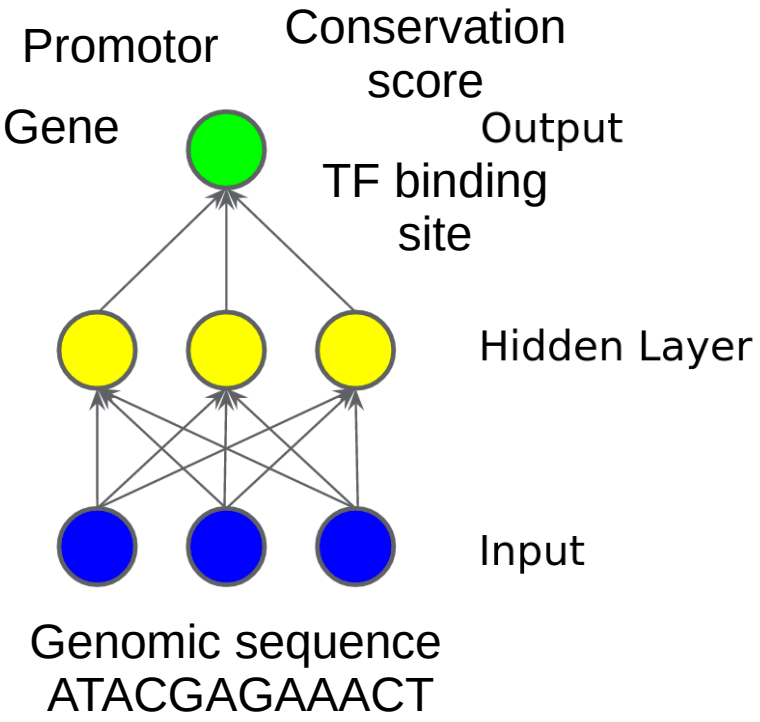


Machine learning



- Uncovering tissue-specific binding features from differential deep learning
- Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks
- Sequential regulatory activity prediction across chromosomes with convolutional neural networks
- Multi-scale deep tensor factorization learns a latent representation of the human epigenome
- Large-scale imputation of epigenomic datasets for systematic annotation of diverse human tissues
- Denoising genome-wide histone ChIP-seq with convolutional neural networks
- Etc. etc. etc.

Machine learning



- Uncovering tissue-specific binding features from differential deep learning
- Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks
- Sequential regulatory activity prediction across chromosomes with convolutional neural networks
- Multi-scale deep tensor factorization learns a latent representation of the human epigenome
- Large-scale imputation of epigenomic datasets for systematic annotation of diverse human tissues
- Denoising genome-wide histone ChIP-seq with convolutional neural networks
- Etc. etc. etc.

I spent the whole first year of my PhD getting these inputs and outputs!

Input

ATCGGCAT

- Label encoding:
  - [1, 4, 2, 3, 3, 2, 4, 1]
- One-hot encoding:
  - 0 1 2 3 4 5 6 7

A -> [[True False False False False False True False]

C -> [False False True False False True False False]

G -> [False False False True True False False False]

T -> [False True False False False False False True]]

Output

- Class:
  - Binary: True or False (promotor?)
  - Multi-class: [False, False, True, False]
    - Promotor? Gene? TFBS? Nucleosome?
  - **Q**: Label encoded or one-hot encoded?
- Continuous:
  - Pileup
  - Level of expression
- Level of detail
  - Per nucleotide
  - Per sequence
  - Anything in between

Online learning

- Most statistical methods require the complete dataset as input
- Neural networks can be trained ‘online’
  - This means adding subsets of data to the model
  - Advantages:
    - Faster
    - Less memory usage
  - Disadvantage:
    - Methods have no memory between subsets

Why a (another) database?

- Genomelake
  - Fast (O(1), no out-of-the-box multiprocessing)
    - Toy dataset: 24.3 sec
- Kipoiseq
  - Slow (O(1), multiprocessing support)
    - Toy dataset (01 workers): 49.8 sec
    - Toy dataset (20 workers): 3.8 s
- Both:
  - Requires file (BED) of positions and its label
    - Makes filtering, changing, and playing with the data relatively hard
  - Support only one genome (“easy” workaround is to combine all genomes)

Why a (another) database?

- Genomelake
  - Fast (O(1), no out-of-the-box multiprocessing)
    - Toy dataset: 24.3 sec
- Kipoiseq
  - Slow (O(1),
    - Toy datase
    - Toy datase
- Both:
  - Requires file (BED) of positions and its label
    - Makes filtering, changing, and playing with the data relatively hard
  - Support only one genome (“easy” workaround is to combine all genomes)

Chromosome	Chromstart	Chromend	Label
chr22	238	12379	1
chr10	43	89	3
chrM	7823	89912	0
chr22	3218	3219	2

Why a (another) database?

- Genomelake
  - Medium speed (O(1), no out-of-the-box multiprocessing)
    - Toy dataset: 24.3 sec
- Kipoiseq
  - Fast when multiprocessing (O(1), multiprocessing support)
    - Toy dataset (01 workers): 49.8 sec
    - Toy dataset (20 workers): 3.8 s
- Both:
  - Requires file (BED) of positions and its label
    - Makes filtering, changing, and playing with the data relatively hard
  - Support only one genome (“easy” workaround is to combine all genomes)

PeakSQL

- PeakSQL
  - Fast (O(log(n)), multiprocessing support)
    - Toy dataset (01 workers): 13.1 sec
    - Toy dataset (20 workers): 1.5 sec
  - “Fully” dynamic constraints
    - Specify at run-time
    - Allows for filtering on whatever (chromosome, condition, pval, length, etc.)
    - *Dynamic* is a potential bottleneck with large databases
  - Multiple genomes
  - *Concerns* about scalability
    - How it scales to many experimental conditions
    - How it scales to per-nucleotide info

```python
import peaksql
from torch.utils.data import DataLoader


# files we need
db_file = './peakSQL.sqlite'  # where to store the database
genome_fasta = "/home/sande/.local/share/genomes/hg19/hg19.fa"
intervals_file = "kundaje.bed"

# pre-process
db = peaksql.database.DataBase(db_file)
db.add_assembly(genome_fasta, assembly="hg19", species="human")
db.add_data(intervals_file, assembly="hg19")
db.create_index()

# set up a dataloader, accepts an SQL clause as argument
train = peaksql.datasets.BedDataSet(db_file, seq_length=101, stride=100,
                                    where="WHERE chromosome='chr22'",
                                    in_memory=False)
trainload = iter(DataLoader(train, batch_size=128, shuffle=True, num_workers=20))

# now let keras do all the heavy lifting
from keras.models import Sequential
from keras.layers import Conv1D, Flatten, Dense

model = Sequential()
model.add(Conv1D(15, 25, input_shape=(101, 4)))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# feed our dataloader to the model and learn
model.fit_generator(trainload, steps_per_epoch=100)

```

PeakSQL


Assembly
AssemblyId int
Assembly text
Species text
AbsPath text
Chromosome
ChromosomeId INTEGER
Chromosome TEXT
Size INT
AssemblyId int
BED
BedId int
AssemblyId int
ConditionId int
ChromosomeId int
ChromStart int
ChromEnd int
Name text
Score int
Strand text
SignalValue text
PValue float
QValue float
Peak int
Condition
ConditionId int
Condition text

PeakSQL

Sybren?

Optimizations

- Numba
- lru_cache
- line_profiler
- Binary search

Numba

```
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

```
0.28 secs for 1.000.000 calls
```

Numba

```
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

```
0.28 secs for 1.000.000 calls
```

```
@jit(nopython=True)
def monte_carlo_pi(nsamples):
```

```
0.007 secs for 1.000.000 calls
```

lru_cache

```
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)
```

Takes 2.34 secs for 35th fibonacci nr

lru_cache

```
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)
```

Takes 2.34 secs for 35th fibonacci nr

```
@lru_cache()
def fib(n):
```

Takes 0.000016 secs for 35th fibonacci nr

Line profiler

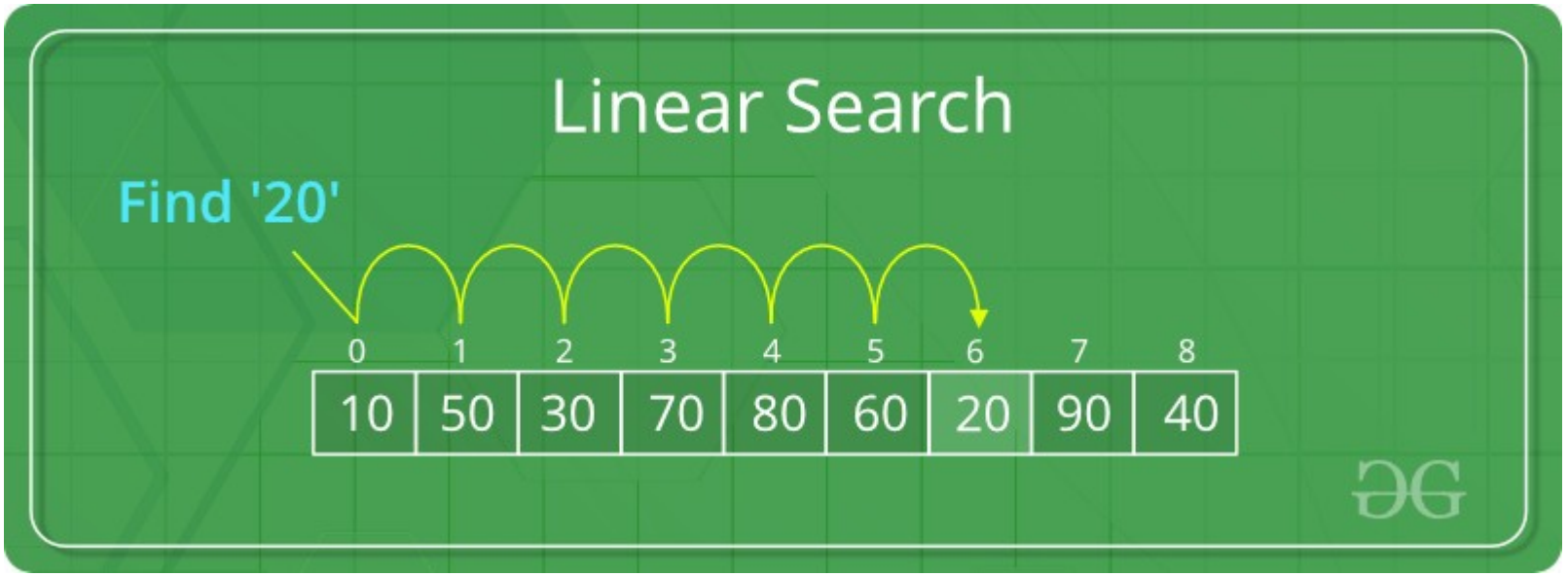
- https://github.com/rkern/line_profiler
  - Simply add @profile function

Line profiler

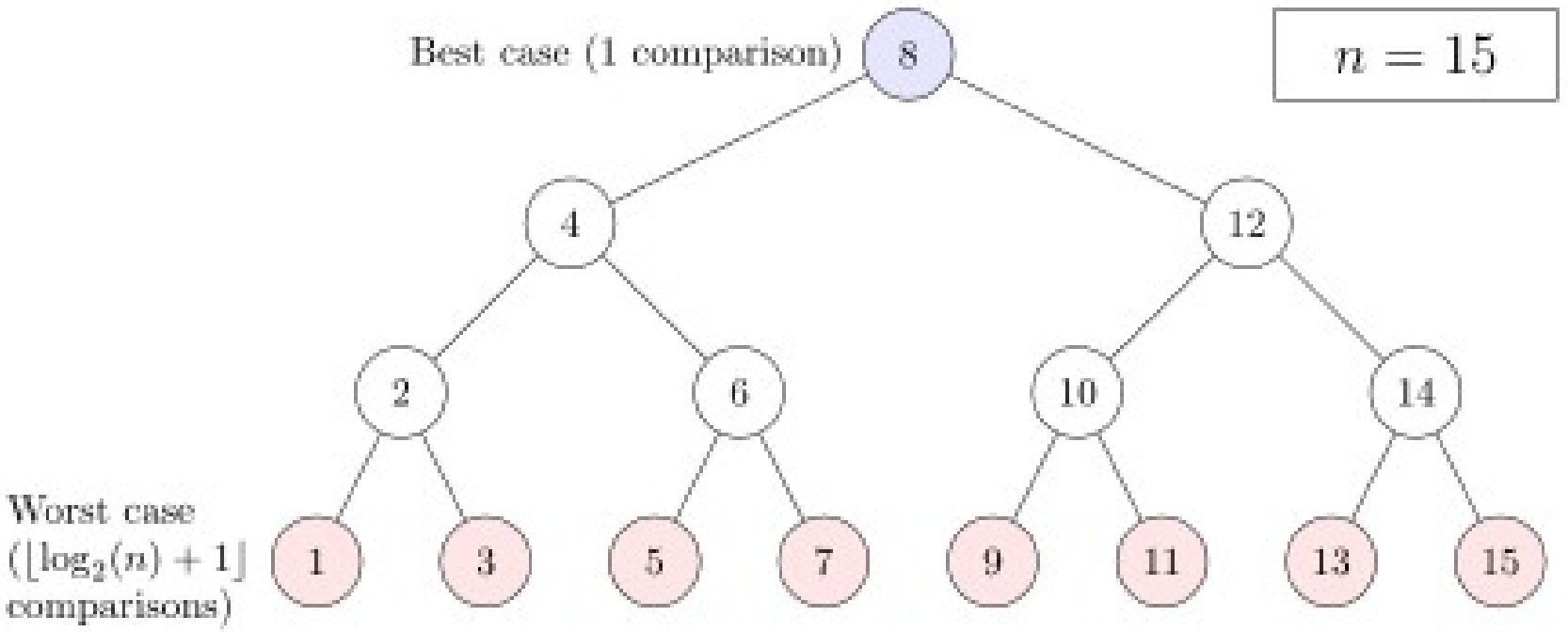
- https://github.com/rkern/line_profiler
  - Simply add @profile function

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    72                                           @profile
    73                                           def sequence_to_onehot(sequence):
    74                                               """
    75                                               Convert a sequence of length n to one-hot encoding of shape (4 x n).
    76                                               """
    77     10000   17576387.0   1757.6     59.6      seq = np.fromiter(str(sequence).upper(), (np.unicode, 1))
    78     10000   11931301.0   1193.1     40.4      return _sequence_to_onehot(seq).T
```

Binary search



Binary search



In short: Another database

- KipoSeq & genomelake fulfill the same function
  - Static input from BED
- PeakSQL
  - Dynamic data generation
  - Sample genome(s) evenly with stride; or get N random positions
  - Multi genome & multiple conditions (classes)
  - *Future*: support for per nucleotide info