fastISM

Jun 18, 2022

Contents:

A Keras implementation for fast in-silico saturated mutagenesis (ISM) for convolution-based architectures. It speeds up ISM by 10x or more by restricting computation to those regions of each layer that are affected by a mutation in the input.

CHAPTER 1

Quickstart

A Keras implementation for fast in-silico saturated mutagenesis (ISM) for convolution-based architectures. It speeds up ISM by 10x or more by restricting computation to those regions of each layer that are affected by a mutation in the input.

1.1 Installation

Currently, fastISM is available to download from PyPI. Bioconda support is expected to be added in the future. fastISM requires TensorFlow 2.3.0 or above.

```
pip install fastism
```

1.2 Usage

fastISM provides a simple interface that takes as input Keras models. For any Keras `model` that takes in sequence as input of dimensions `(B, S, C)`, where

- `B`: batch size
- `S`: sequence length
- `C`: number of characters in vocabulary (e.g. 4 for DNA/RNA, 20 for proteins)

Perform ISM as follows:

```
from fastism import FastISM

fast_ism_model = FastISM(model)

for seq_batch in sequences:
    # seq_batch has dim (B, S, C)
```

(continues on next page)

(continued from previous page)

```
    ism_seq_batch = fast_ism_model(seq_batch)
    # ism_seq_batch has dim (B, S, num_outputs)
```

fastISM does a check for correctness when the model is initialised, which may take a few seconds depending on the size of your model. This ensures that the outputs of the model match that of an unoptimised implementation. You can turn it off as `FastISM(model, test_correctness=False)`. fastISM also supports introducing specific mutations, mutating different ranges of the input sequence, and models with multiple outputs. Check the Examples section of the documentation for more details. An executable tutorial is available on Colab.

1.3 Benchmark

You can estimate the speedup obtained by comparing with a naive implementation of ISM.

```
# Test this code as is
>>> from fastism import FastISM, NaiveISM
>>> from fastism.models.basset import basset_model
>>> import tensorflow as tf
>>> import numpy as np
>>> from time import time

>>> model = basset_model(seqlen=1000)
>>> naive_ism_model = NaiveISM(model)
>>> fast_ism_model = FastISM(model)

>>> def time_ism(m, x):
        t = time()
        o = m(x)
        print(time()-t)
        return o

>>> x = tf.random.uniform((1024, 1000, 4),
                          dtype=model.input.dtype)

>>> naive_out = time_ism(naive_ism_model, x)
144.013728
>>> fast_out = time_ism(fast_ism_model, x)
13.894407
>>> np.allclose(naive_out, fast_out, atol=1e-6)
True
>>> np.allclose(fast_out, naive_out, atol=1e-6)
True # np.allclose is not symmetric
```

See `notebooks/ISMBenchmark.ipynb` for benchmarking code that accounts for initial warm-up.

1.4 Getting Help

fastISM supports the most commonly used subset of Keras for biological sequence-based models. Occasionally, you may find that some of the layers used in your model are not supported by fastISM. Refer to the Supported Layers section in Documentation for instructions on how to incorporate custom layers. In a few cases, the fastISM model may fail correctness checks, indicating there are likely some issues in the fastISM code. In such cases or any other bugs, feel free to reach out to the author by posting an Issue on GitHub along with your architecture, and we'll try to work out a solution!

1.5 Citation

fastISM: Performant *in-silico* saturation mutagenesis for convolutional neural networks; Surag Nair, Avanti Shrikumar*, Jacob Schreiber*, Anshul Kundaje (Bioinformatics 2022) http://doi.org/10.1093/bioinformatics/btac135.

*equal contribtion

Preprint available on bioRxiv.

CHAPTER 2

Examples

This section covers some of the common use cases and functionalities of fastISM.

fastISM provides a simple interface that takes as input Keras model For any Keras `model` that takes in sequence as input of dimensions `(B, S, C)`, where

- `B`: batch size
- `S`: sequence length
- `C`: number of characters in vocabulary (e.g. 4 for DNA/RNA, 20 for proteins)

2.1 Alternate Mutations

By default, inputs at the ith position are set to zero. It is possible to specify mutations of interest by passing them to `replace_with` in the call to the fastISM model. To perform ISM with all possible mutations for DNA:

```
fast_ism_model = FastISM(model)

mutations = [[1,0,0,0],
             [0,1,0,0],
             [0,0,1,0],
             [0,0,0,1]]

for seq_batch in sequences:
    # seq_batch has dim (B, S, C)
    for m in mutations:
        ism_seq_batch = fast_ism_model(seq_batch, replace_with=m)
        # ism_seq_batch has dim (B, S, num_outputs)
        # process/store ism_seq_batch
```

Each `ism_seq_batch` has the same dimensions `(B, S, num_outputs)`. The outputs of the model are computed on the mutations only for those positions where the base differs from the mutation. Where the base is the same as the mutation, the output is the same as for the unperturbed sequence.

2.2 Alternate Ranges

By default, mutations are introduced at every single position in the input. You can also set a list of equal-sized ranges as input instead of single positions. Consider a model that takes as input 1000 length sequences, and we wish to introduce a specific mutation of length 3 in the central 150 positions:

TODO: test this

```
# specific mutation to introduce
mut = [[0,0,0,1],
       [0,0,0,1],
       [0,0,0,1]]

# ranges where mutation should be introduced
mut_ranges = [(i,i+3) for i in range(425,575)]

fast_ism_model = FastISM(model,
                         change_ranges = mut_ranges)

for seq_batch in sequences:
    ism_seq_batch = fast_ism_model(seq_batch, replace_with=mut)
```

2.3 Multi-input Models

fastISM supports models which have other inputs in addition to the sequence input that is perturbed. These alternate inputs are assumed to stay constant through different perturbations of the primary sequence input. Consider the model below in which an addition vector is concatenated with the flattened sequence output:

```
def get_model():
    rna = tf.keras.Input((100,)) # non-sequence input
    seq = tf.keras.Input((100,4))

    x = tf.keras.layers.Conv1D(20, 3)(seq)
    x = tf.keras.layers.Conv1D(20, 3)(x)
    x = tf.keras.layers.Flatten()(x)

    rna_fc = tf.keras.layers.Dense(10)(rna)

    x = tf.keras.layers.Concatenate()([x, rna_fc])
    x = tf.keras.layers.Dense(10)(x)
    x = tf.keras.layers.Dense(1)(x)
    model = tf.keras.Model(inputs=[rna,seq], outputs=x)

    return model
```

To inform fastISM that the second input is the primary sequence input that will be perturbed:

```
>>> model = get_model()
>>> fast_ism_model = FastISM(model, seq_input_idx=1)
```

Then to obtain the outputs:

```
for rna_batch, seq_batch in data_batches:
   ism_batch = fast_ism_model([rna_batch, seq_batch])
```

(continues on next page)

(continued from previous page)

```
# or equivalently without splitting inputs
for data_batch in data_batches
    ism_batch = fast_ism_model(data_batch)
```

NOTE: Currently, multi-input models in which descendants of alternate inputs interact directly with descendants of primary sequence input *before* a *Stop Layer* are not supported, i.e. a descendant of an alternate input in general should only interact with a flattened version of primary input sequence.

2.4 Recursively Defined Models

Keras allows defining models in a nested fashion. As such, recursively defined models should not pose an issue to fastISM. The example below works:

```
def res_block(input_shape):
    inp = tf.keras.Input(shape=input_shape)
    x = tf.keras.layers.Conv1D(20, 3, padding='same')(inp)
    x = tf.keras.layers.Add()([inp, x])
    model = tf.keras.Model(inputs=inp, outputs=x)
    return model

def fc_block(input_shape):
    inp = tf.keras.Input(shape=input_shape)
    x = tf.keras.layers.Dense(10)(inp)
    x = tf.keras.layers.Dense(1)(x)

    model = tf.keras.Model(inputs=inp, outputs=x)
    return model

def get_model():
    res = res_block(input_shape=(108,20)))
    fcs = fc_block(input_shape=(36*20,))

    inp = tf.keras.Input((108, 4))
    x = tf.keras.layers.Conv1D(20, 3, padding='same')(inp)
    x = res(x)
    x = tf.keras.layers.MaxPooling1D(3)(x)
    x = tf.keras.layers.Flatten()(x)
    x = fcs(x)

    model = tf.keras.Model(inputs=inp, outputs=x)

    return model

>>> model = get_model()
>>> fast_ism_model = FastISM(model)
```

CHAPTER 3

fastISM on DeepSEA Beluga

fastISM is a faster way to perform *in-silico* saturation mutagenesis. This tutorial uses the DeepSEA Beluga model (Zhou et al 2018), which predicts 2002 chromatin features for a 2000 bp input sequence. This tutorial covers the following:

- Installations and downloading required files for tutorial
- Benchmarking fastISM on the Beluga model against a standard ISM implementation
- Running fastISM on custom input sequences
- Visualizing fastISM output across all tasks (outputs)
- Selecting a task, visualizing the fastISM scores, and zooming in to visualize the underlying sequence features.

3.1 Installations and Data

We use `pip` to install `fastISM` and `vizsequence` (to visualize sequence importance scores). In addition we download a trained Beluga model and a tsv with all the model's outputs.

[4]:
```
# install fastISM
!pip install fastism
```

```
Collecting fastism
  Downloading https://files.pythonhosted.org/packages/4f/93/
↪26f83f7197d92b0c502d7f7af32cbd5e0d0f0b52a4bfb51b29162e860fc8/fastism-0.4.0-py3-none-
↪any.whl
Requirement already satisfied: tensorflow<3.0.0,>=2.3.0 in /usr/local/lib/python3.6/
↪dist-packages (from fastism) (2.3.0)
Collecting pydot<2.0.0,>=1.4.1
  Downloading https://files.pythonhosted.org/packages/33/d1/
↪b1479a770f66d962f545c2101630ce1d5592d90cb4f083d38862e93d16d2/pydot-1.4.1-py2.py3-
↪none-any.whl
Requirement already satisfied: numpy<1.19.0,>=1.16.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.18.5)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.12.1)
Requirement already satisfied: scipy==1.4.1 in /usr/local/lib/python3.6/dist-packages␣
↪(from tensorflow<3.0.0,>=2.3.0->fastism) (1.4.1)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.32.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.6/dist-packages␣
↪(from tensorflow<3.0.0,>=2.3.0->fastism) (1.15.0)
Requirement already satisfied: keras-preprocessing<1.2,>=1.1.1 in /usr/local/lib/
↪python3.6/dist-packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.1.2)
Requirement already satisfied: astunparse==1.6.3 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.6.3)
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (0.10.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages␣
↪(from tensorflow<3.0.0,>=2.3.0->fastism) (0.35.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (3.3.0)
Requirement already satisfied: google-pasta>=0.1.8 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (0.2.0)
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.6/dist-packages␣
↪(from tensorflow<3.0.0,>=2.3.0->fastism) (0.3.3)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (3.12.4)
Requirement already satisfied: tensorboard<3,>=2.3.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (2.3.0)
Requirement already satisfied: tensorflow-estimator<2.4.0,>=2.3.0 in /usr/local/lib/
↪python3.6/dist-packages (from tensorflow<3.0.0,>=2.3.0->fastism) (2.3.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (1.1.0)
Requirement already satisfied: h5py<2.11.0,>=2.10.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorflow<3.0.0,>=2.3.0->fastism) (2.10.0)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.6/dist-
↪packages (from pydot<2.0.0,>=1.4.1->fastism) (2.4.7)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages␣
↪(from protobuf>=3.9.2->tensorflow<3.0.0,>=2.3.0->fastism) (50.3.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/
↪python3.6/dist-packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->
↪fastism) (1.7.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/
↪python3.6/dist-packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->
↪fastism) (0.4.1)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-
↪packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (1.0.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-
↪packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (3.2.2)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-
↪packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (1.17.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-
↪packages (from tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (2.23.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/
↪dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<3,>=2.3.0->
↪tensorflow<3.0.0,>=2.3.0->fastism) (1.3.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/
↪local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard<3,>=2.3.0->
↪tensorflow<3.0.0,>=2.3.0->fastism) (1.7.0)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/
↪dist-packages (from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->tensorflow<3.0.0,>
↪=2.3.0->fastism) (4.1.1)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/local/lib/
↪python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->
↪tensorflow<3.0.0,>=2.3.0->fastism) (4.6)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-
↪packages (from google-auth<2,>=1.6.3->tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.
↪0->fastism) (0.2.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-
↪packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0-
↪>fastism) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages␣
↪(from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->
↪fastism) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
↪lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->
↪tensorflow<3.0.0,>=2.3.0->fastism) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-
↪packages (from requests<3,>=2.21.0->tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0-
↪>fastism) (2020.6.20)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-
↪packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->
↪tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (3.1.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages␣
↪(from importlib-metadata; python_version < "3.8"->markdown>=2.6.8->tensorboard<3,>
↪=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (3.1.0)
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.6/dist-
↪packages (from rsa<5,>=3.1.4; python_version >= "3"->google-auth<2,>=1.6.3->
↪tensorboard<3,>=2.3.0->tensorflow<3.0.0,>=2.3.0->fastism) (0.4.8)
Installing collected packages: pydot, fastism
  Found existing installation: pydot 1.3.0
    Uninstalling pydot-1.3.0:
      Successfully uninstalled pydot-1.3.0
Successfully installed fastism-0.4.0 pydot-1.4.1
```

[ ]:
```
#for visualizing the per-position importance
!pip install vizsequence
```

```
Collecting vizsequence
  Downloading https://files.pythonhosted.org/packages/a6/10/
↪b3b210eba27de588fba3c261b80317413e18ac3e371df9578b3cdc61096c/vizsequence-0.1.1.0.
↪tar.gz
Requirement already satisfied: numpy>=1.9 in /usr/local/lib/python3.6/dist-packages␣
↪(from vizsequence) (1.18.5)
Requirement already satisfied: matplotlib>=2.2.2 in /usr/local/lib/python3.6/dist-
↪packages (from vizsequence) (3.2.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/
↪lib/python3.6/dist-packages (from matplotlib>=2.2.2->vizsequence) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages␣
↪(from matplotlib>=2.2.2->vizsequence) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-
↪packages (from matplotlib>=2.2.2->vizsequence) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-
↪packages (from matplotlib>=2.2.2->vizsequence) (1.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from␣
↪cycler>=0.10->matplotlib>=2.2.2->vizsequence) (1.15.0)
Building wheels for collected packages: vizsequence
  Building wheel for vizsequence (setup.py) ... done
  Created wheel for vizsequence: filename=vizsequence-0.1.1.0-cp36-none-any.whl␣
↪size=3269 sha256=f5c7dd7708c4186bbdadb6e2e428969bec77fb97b43315a0eb60a4c0f1f66c62f
```

(continues on next page)

(continued from previous page)

```
  Stored in directory: /root/.cache/pip/wheels/08/eb/de/
↪6b398b439ba39c278e5c341bdeed57d66280910e096496eaef
Successfully built vizsequence
Installing collected packages: vizsequence
Successfully installed vizsequence-0.1.1.0
```

[ ]:
```
# download trained model
! wget http://mitra.stanford.edu/kundaje/surag/fastISM/deepseabeluga_keras_
↪nopermutelayer.h5 -O deepseabeluga.h5
```

```
--2020-09-20 06:25:21--  http://mitra.stanford.edu/kundaje/surag/fastISM/
↪deepseabeluga_keras_nopermutelayer.h5
Resolving mitra.stanford.edu (mitra.stanford.edu)... 171.67.96.243
Connecting to mitra.stanford.edu (mitra.stanford.edu)|171.67.96.243|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 598186116 (570M)
Saving to: 'deepseabeluga.h5'

deepseabeluga.h5    100%[===================>] 570.47M  48.5MB/s    in 12s

2020-09-20 06:25:33 (46.9 MB/s) - 'deepseabeluga.h5' saved [598186116/598186116]
```

[ ]:
```
# download output annotation
! wget https://raw.githubusercontent.com/FunctionLab/ExPecto/20b99d1278678/resources/
↪deepsea_beluga_2002_features.tsv -O outputs.tsv
```

```
--2020-09-20 06:25:35--  https://raw.githubusercontent.com/FunctionLab/ExPecto/
↪20b99d1278678/resources/deepsea_beluga_2002_features.tsv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.
↪101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 203001 (198K) [text/plain]
Saving to: 'outputs.tsv'

outputs.tsv         100%[===================>] 198.24K  --.-KB/s    in 0.04s

2020-09-20 06:25:36 (5.37 MB/s) - 'outputs.tsv' saved [203001/203001]
```

3.2 Init

[ ]:
```
import fastism
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
import vizsequence

# for some seaborn warnings
import warnings; warnings.simplefilter('ignore')
```

(continues on next page)

(continued from previous page)

```
import time
```

[ ]:
```
tf.__version__
```

```
'2.3.0'
```

[5]:
```
! pip freeze | grep fastism
```

```
fastism==0.4.0
```

[1]:
```
!nvidia-smi
```

```
Wed Sep 23 09:42:18 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.66       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P0    25W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[ ]:
```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_
↪devices('GPU')))
```

```
Num GPUs Available:  1
```

[ ]:
```
device = 'GPU:0' if tf.config.experimental.list_physical_devices('GPU') else␣
↪'/device:CPU:0'
device
```

```
'GPU:0'
```

3.3 Load Model

[ ]:
```
model = tf.keras.models.load_model("deepseabeluga.h5")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was␣
↪*not* compiled. Compile it manually.
```

[ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 1993, 320)         10560
_________________________________________________________________
activation_1 (Activation)    (None, 1993, 320)         0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1986, 320)         819520
_________________________________________________________________
activation_2 (Activation)    (None, 1986, 320)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1986, 320)         0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 496, 320)          0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 489, 480)          1229280
_________________________________________________________________
activation_3 (Activation)    (None, 489, 480)          0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 482, 480)          1843680
_________________________________________________________________
activation_4 (Activation)    (None, 482, 480)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 482, 480)          0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 120, 480)          0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 113, 640)          2458240
_________________________________________________________________
activation_5 (Activation)    (None, 113, 640)          0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 106, 640)          3277440
_________________________________________________________________
activation_6 (Activation)    (None, 106, 640)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 67840)             0
_________________________________________________________________
altereddense (Dense)         (None, 2003)              135885523
_________________________________________________________________
activation_7 (Activation)    (None, 2003)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2002)              4012008
_________________________________________________________________
activation_8 (Activation)    (None, 2002)              0
=================================================================
Total params: 149,536,251
Trainable params: 149,536,251
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
model.input.shape
```

```
TensorShape([None, 2000, 4])
```

[ ]:
```
# a look at the 2002 model outputs
outputs = pd.read_csv("./outputs.tsv", sep="\t")
```

(continues on next page)

(continued from previous page)

```
outputs
```

```
      Unnamed: 0  ...                                          Unnamed: 6
0              1  ...  ./wgEncodeAwgDnaseUniform/wgEncodeAwgDnaseDuke...
1              2  ...  ./wgEncodeAwgDnaseUniform/wgEncodeAwgDnaseDuke...
2              3  ...  ./wgEncodeAwgDnaseUniform/wgEncodeAwgDnaseDuke...
3              4  ...  ./wgEncodeAwgDnaseUniform/wgEncodeAwgDnaseDuke...
4              5  ...  ./wgEncodeAwgDnaseUniform/wgEncodeAwgDnaseDuke...
...          ...  ...                                                ...
1997        1998  ...                          E129-H3K4me2.narrowPeak.gz
1998        1999  ...                          E129-H3K4me3.narrowPeak.gz
1999        2000  ...                         E129-H3K79me2.narrowPeak.gz
2000        2001  ...                          E129-H3K9me3.narrowPeak.gz
2001        2002  ...                         E129-H4K20me1.narrowPeak.gz

[2002 rows x 7 columns]
```

3.4 Benchmark

Calculate approximate speedup of fastISM vs a naive implementation.

[ ]:
```
def time_ism(ism_model, batch_sizes, seqlen):
    times = []
    per_100 = []
    for b in batch_sizes:
        # using randomly initialized tensors for benchmarking
        x = np.random.random((b,seqlen,4))
        x = tf.constant(x, dtype=ism_model.model.input.dtype)

        t = time.time()
        # each position is substituted with 0s for benchmarking
        ism_model(x)
        times.append(time.time()-t)

        per_100.append((times[-1]/b)*100)

        print("BATCH: {}\tTIME: {:.2f}\tPER 100: {:.2f}".format(b, times[-1], (times[-
↪1]/b)*100))

    print("BEST PER 100: {:.2f}".format(min(per_100)))
```

[ ]:
```
fast_ism_model = fastism.FastISM(model, test_correctness=False)
```

[ ]:
```
# 512 works with 16Gb GPU memory
time_ism(fast_ism_model, [128, 512], 2000)
```

```
BATCH: 128      TIME: 42.58     PER 100: 33.26
BATCH: 512      TIME: 85.09     PER 100: 16.62
BEST PER 100: 16.62
```

[ ]:
```
naive_ism_model = fastism.NaiveISM(model)
```

[ ]:
```
time_ism(naive_ism_model, [128, 512], 2000)
```

```
BATCH: 128      TIME: 142.26    PER 100: 111.14
BATCH: 512      TIME: 513.30    PER 100: 100.25
BEST PER 100: 100.25
```

Naive ISM takes around 100 seconds per 100 sequence, while fastISM takes around 16.6 seconds. That's a speedup of about 6x!

3.5 Run on Custom Sequences

We run fastISM on selected custom sequences, check correctness against a standard implementation and visualize the fastISM importance score matrix across the top 200 outputs.

[ ]:
```
# chr3:138862274-138864274 (hg38)
chr3_enhancer =␣
↪"CCGGTGATTTTCTGGAGTCTATATCCTTCATCAGATTTTCCAAGGGGTGTCTGTCCCCTCAAAAGAATGATTGTCATTATTTGAAAGACTAGTTCCAC

# reproduce to have a "batch" of sequences
# you can add more custom sequences to this list
sequences = [chr3_enhancer]*5

#We define a function to do the one-hot encoding
onehot_mapping = {
    'A': [1,0,0,0],
    'C': [0,1,0,0],
    'G': [0,0,1,0],
    'T': [0,0,0,1],
    'N': [0,0,0,0],
    'a': [1,0,0,0],
    'c': [0,1,0,0],
    'g': [0,0,1,0],
    't': [0,0,0,1],
}
def one_hot_encode(sequence):
  return np.array([onehot_mapping[x] for x in sequence])

onehot_sequences = np.array([one_hot_encode(x) for x in sequences])

onehot_sequences.shape
```

```
(5, 2000, 4)
```

[ ]:
```
x = tf.constant(onehot_sequences, dtype=model.input.dtype)
mutations = [[1,0,0,0],
             [0,1,0,0],
             [0,0,1,0],
             [0,0,0,1]]

# get outputs with all mutations (A/C/G/T)
# in general, fastISM gives the most speedup for larger batch sizes
# here we use a small batch size for illustration
fast_ism_out = [fast_ism_model(x, replace_with=mut) for mut in mutations]
naive_ism_out = [naive_ism_model(x, replace_with=mut) for mut in mutations]
```

[ ]:
```
# make into a batch_size x seqlen x num_outputs x 4 (bases)
fast_ism_out = np.transpose(np.array(fast_ism_out), (1,2,3,0))
```

(continues on next page)

(continued from previous page)

```
naive_ism_out = np.transpose(np.array(naive_ism_out), (1,2,3,0))

print(naive_ism_out.shape)
```

```
(5, 2000, 2002, 4)
```

[ ]:
```
# check correctness
print(np.allclose(fast_ism_out, naive_ism_out, atol=1e-6))
print(np.allclose(naive_ism_out, fast_ism_out, atol=1e-6))
```

```
True
True
```

[ ]:
```
# reference output (should return using ISM itself)
ref = model(x, training=False).numpy()
print(ref.shape)

# repeat to make shape compatible with ism_out (for broadcasting)
ref = np.expand_dims(np.repeat(np.expand_dims(ref, 1), 2000, 1), -1)
print(ref.shape)
```

```
(5, 2002)
(5, 2000, 2002, 1)
```

[ ]:
```
mut_minus_ref = fast_ism_out - ref

# euclidean distance from reference for each output at each position
mut_minus_ref_euclidean =  np.sqrt(np.sum(np.square(mut_minus_ref), -1)/3)

print(mut_minus_ref_euclidean.shape)
```

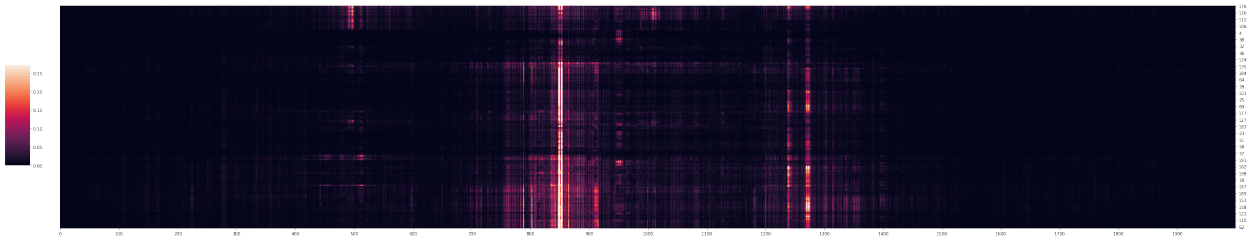
```
(5, 2000, 2002)
```

[ ]:
```
# pick index of sequence to explore
SEQ_IDX = 0
```

[ ]:
```
# heatmap of importance scores [tasks (clustered) x position]
# for top 200 tasks by reference predictions

top_tasks = np.argsort(ref[SEQ_IDX][0][:,0])[::-1][:200]
to_plot = mut_minus_ref_euclidean[SEQ_IDX][:, top_tasks].T

cg = sns.clustermap(to_plot,
                    row_cluster=True, # cluster tasks
                    col_cluster=False, # not position
                    # standard_scale=0,
                    dendrogram_ratio=0.04,
                    figsize=(40,8),
                    xticklabels=100,
                    vmax = np.quantile(to_plot, 0.998), # set max limit
                    cbar_pos=(0, .3, .02, .4))
cg.ax_row_dendrogram.set_visible(False)
```



Positions in the input have different importance scores depending on the task. For example, regions around 500 bp have high importance scores for the top few tasks but not for the rest of the tasks.

3.6 Sequence Features for a Single Output

Let's pick a task and zoom in to visualize the underlying sequence with high importance scores for that task.

[ ]:
```
# pick task with highest prediction
TASK_IDX = np.argsort(ref[SEQ_IDX][0][:,0])[::-1][0]

# or pick another task of your choice
# TASK_IDX = 1777

TASK_IDX
```

```
1777
```

[ ]:
```
# predicted value for that task
ref[SEQ_IDX, 0, TASK_IDX, 0]
```

```
0.98828876
```

[ ]:
```
# output (task) description
outputs.iloc[TASK_IDX]
```

```
Unnamed: 0                      1778
Cell type            Small_Intestine
Assay                      DNase.hot
Treatment                        NaN
Assay type                     DNase
Source           Roadmap Epigenomics
Unnamed: 6     E109-DNase.hot.bed.gz
Name: 1777, dtype: object
```

The output is DNase-seq prediction in small intestine.

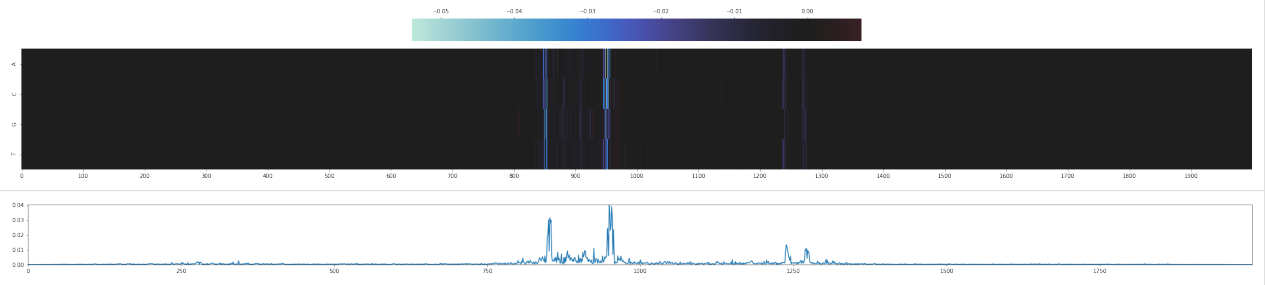
[ ]:
```
# heatmap of base x position difference from ref
plt.figure(figsize=(40,5))
to_plot = mut_minus_ref[SEQ_IDX, :, TASK_IDX].T
sns.heatmap(to_plot,
            center=0,
            vmin = np.quantile(to_plot, 0.0001),
            vmax = np.quantile(to_plot, 0.9999),
            xticklabels = 100,
            yticklabels = ['A','C','G','T'],
            cbar_kws = dict(use_gridspec=False,location="top"))
```

(continues on next page)

(continued from previous page)

```
plt.show()

plt.figure(figsize=(40,2))
plt.plot(mut_minus_ref_euclidean[SEQ_IDX, :, TASK_IDX])
plt.margins(0, 0)
plt.show()
```



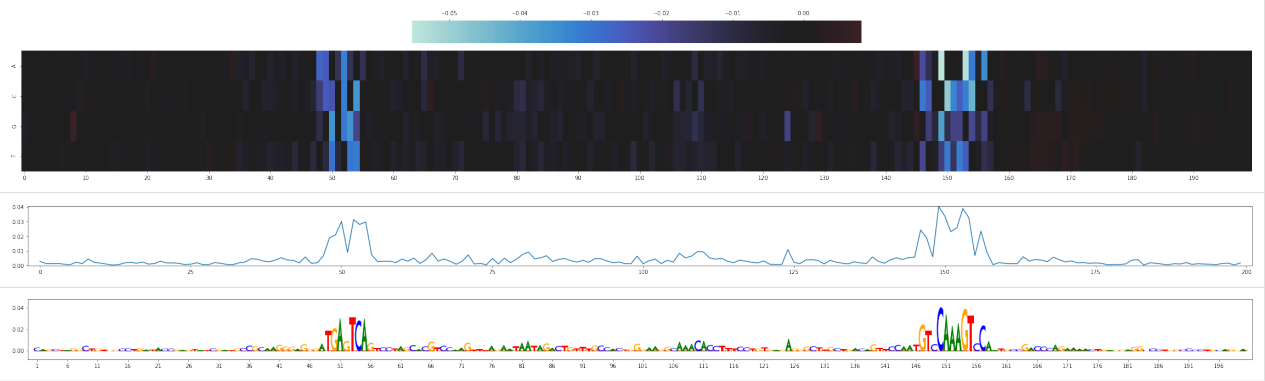
Let's zoom into the central bases and visualize the sequence as well.

[ ]:
```
# Zoom in heatmap of base x position
ZOOM_IN = range(800, 1000)

plt.figure(figsize=(40,5))
to_plot = mut_minus_ref[SEQ_IDX, ZOOM_IN, TASK_IDX].T
sns.heatmap(to_plot,
            center=0,
            vmin = np.quantile(to_plot, 0.0001),
            vmax = np.quantile(to_plot, 0.9999),
            xticklabels = 10,
            yticklabels = ['A','C','G','T'],
            cbar_kws = dict(use_gridspec=False,location="top"))
plt.show()

plt.figure(figsize=(40,2))
plt.plot(mut_minus_ref_euclidean[SEQ_IDX, ZOOM_IN, TASK_IDX])
plt.margins(0.01, 0.01)
plt.show()

vizsequence.plot_weights(
      np.expand_dims(mut_minus_ref_euclidean[SEQ_IDX, ZOOM_IN, TASK_IDX], -1)*onehot_
↪sequences[SEQ_IDX, ZOOM_IN],
      figsize=(40,2))
```



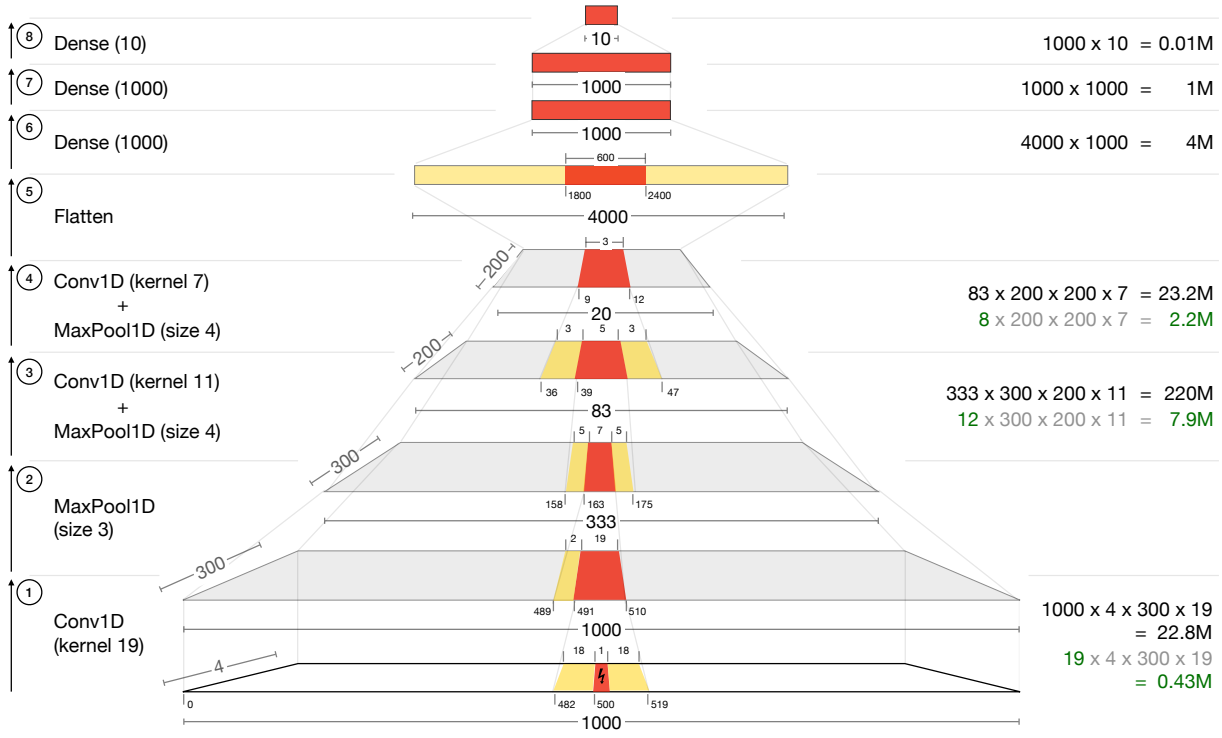
On the left is an AP1 motif and on the right is HNF4 motif. HNF4 has a known role in intestinal development.

CHAPTER 4

How fastISM Works

This section gives a high level overview of the fastISM algorithm. For more detail, check out the paper, or better still, take a look at the source code!

fastISM is based on the observation that neural networks spend the majority of their computation time in convolutional layers and that point mutations in the input sequence only affect limited a range of intermediate layers. As a result, most of the computation in ISM is redundant and avoiding it can result in significant speedups.



Consider the above annotated diagram of a Basset-like architecture (Kelley et al., 2016) on an input DNA sequence of length 1000, with a 1 base-pair mutation at position 500. Positions marked in red indicate the regions that are affected by the point mutation in the input. Positions marked in yellow, flanking the positions in red, indicate unaffected regions that contribute to the output of the next layer. Ticks at the bottom of each layer correspond to position indices. Numbers on the right in black indicate the approximate number of computations required at that layer for a naive implementation of ISM. For convolution layers, the numbers in gray and green indicate the minimal computations required.

For a single position change in the middle of the input sequence, the output of the first convolution, which has a kernel size of 19, is perturbed at 19 positions which can be computed from just 37 positions in the input. It then goes on to affect 7 out of 333 positions after the first Max Pool layer (Layer 2) and 5 out of 83 positions after the second Max Pool (Layer 3). Once the output of the final Max Pool layer is flattened and passed through a fully connected layer, all the neurons are affected by a single change in the input, and thus all subsequent computations must be recomputed entirely.

fastISM works by restricting computation in the convolution layers to only those positions that are affected by the mutation in the input. Since the most time is spent in convolution layers, fastISM avoids down a major amount of redundant computation and speeds up ISM. See *API* for more details on how this is achieved.

CHAPTER 5

Supported Layers

This sections covers the layers that are currently supported by fastISM. fastISM supports a subset of layers in `tf.keras.layers` that are most commonly used for sequence-based models.

NOTE: Restrictions on layers apply only till *Stop Layers*, beyond which all layers are allowed.

The layers below have been classified by which positions of the output are a function of the input at the `i` th position.

5.1 Local Layers

For Local layers, the input at the `i` th position affects a fixed interval of outputs around the `i` th position.

Supported: Conv1D, Cropping1D

Currently, custom Local Layers are not supported as they may require additional logic to be incorporated into the code. Please post an Issue on GitHub to work out a solution.

5.2 See Through Layers

See through layers are layers for which the output at the `i` th position depends on the input at the `i` th position only.

Supported: Activation, BatchNormalization, Dropout, ELU, InputLayer, LeakyReLU, PReLU, ReLU

To add a custom see-through layer: `fastism.fast_ism_utils.SEE_THROUGH_LAYERS.add("YourLayer")`

5.3 Aggregation Layers

Aggregation layers are also See Through Layers as the output at the `i` th position depends on the input at the `i` th position only. The main difference is that Aggregation layers take in multiple inputs, and thus their output at the `i` th position depends on the `i` th position of all their inputs.

Supported: Add, Average, Maximum, Minumum, Multiply, Subtract

To add a custom aggregation layer: `fastism.fast_ism_utils.AGGREGATE_LAYERS.add("YourLayer")`

5.4 Stop Layers

Layers after which output at `i` th position depends on inputs at most or all positions in the input. However, this is not strictly true for Flatten/Reshape, but it is assumed these are followed by Dense or similar.

Supported: Dense, Flatten, GlobalAveragePooling1D, GlobalMaxPool1D, Reshape

To add a custom stop layer: `fastism.fast_ism_utils.STOP_LAYERS.add("YourLayer")`

5.5 Pooling Layers

Pooling layers are also Local Layers but are special since they are typically used to reduce the size of the input.

Supported: AveragePooling1D, MaxPooling1D

To add a custom pooling layer: `fastism.fast_ism_utils.POOLING_LAYERS.add("YourLayer")`

Custom pooling layers must have the class attributes `pool_size`, `strides` (which must be equal to `pool_size`), `padding` (which must be `valid`), `data_format` (which must be `channels_last`). Here is an example of a custom pooling layer.

```
class AttentionPooling1D(tf.keras.layers.Layer):
    # don't forget to add **kwargs
    def __init__(self, pool_size = 2, **kwargs):
        super().__init__()
        self.pool_size = pool_size

        # need for pooling layer
        self.strides = self.pool_size
        self.padding = "valid" # ensure it behaves like MaxPooling1D with valid␣
↪padding
        self.data_format = "channels_last"

    def build(self, input_shape):
        _, length, num_features = input_shape
        self.w = self.add_weight(
            shape=(num_features, num_features),
            initializer="random_normal",
            trainable=True,
        )

    # implement so that layer can be duplicated
    def get_config(self):
        config = super().get_config()
        config.update({
            "pool_size": self.pool_size,
            "data_format": self.data_format,
            "strides": self.strides,
            "padding": self.padding
        })
```

(continues on next page)

(continued from previous page)

```
        return config

    def call(self, inputs):
        _, length, num_features = inputs.shape

        if length == None: # this can happen at when creating fast_ism_model
            return inputs  # don't do anything for now

        inputs = tf.reshape(
            inputs,
            (-1, length // self.pool_size, self.pool_size, num_features))

        return tf.reduce_sum(
            inputs * tf.nn.softmax(tf.matmul(inputs, self.w), axis=-2),
            axis=-2)
```

Code adapted from Enformer. Note that pooling layers can contain weights.

CHAPTER 6

fastISM package

fastISM takes a Keras model as input. The main steps of fastISM are as follows:

1. One-time Initialization (*fastISM.fast_ism_utils.generate_models()*):

- Obtain the computational graph from the model. This is done in *fastISM.flatten_model.get_flattened_graph()*.
- Chunk the computational graph into segments that can be run as a unit. This is done in *fastISM.fast_ism_utils.segment_model()*.
- Augment the model to create an "intermediate output model" (referred to as `intout_model` in the code) that returns intermediate outputs at the end of each segment for reference input sequences. This is done in *fastISM.fast_ism_utils.generate_intermediate_output_model()*.
- Create a second "mutation propagation model" (referred to as `fast_ism_model` in the code) that largely resembles the original model, but incorporates as additional inputs the necessary flanking regions from outputs of the IntOut model on reference input sequences between segments. This is done in *fastISM.fast_ism_utils.generate_fast_ism_model()*.

2. For each batch of input sequences:

- Run the `intout_model` on the sequences (unperturbed) and cache the intermediate outputs at the end of each segment. This is done in *fastISM.fast_ism.FastISM.pre_change_range_loop_prep()*.
- For each positional mutation:
  - Introduce the mutation in the input sequences
  - Run the `fast_ism_model` feeding as input appropriate slices of the `intout_model` outputs. This is done in *fastISM.fast_ism.FastISM.get_ith_output()*.

See *How fastISM Works* for a more intuitive understanding of the algorithm.

6.1 ism_base module

This module contains a *base ISM* class, from which the *NaiveISM* and *FastISM* classes inherit. It also includes implementation of *NaiveISM*.

class fastISM.ism_base.**ISMBase**(*model*, *seq_input_idx=0*, *change_ranges=None*)
Bases: object

get_ith_output(*inp_batch*, *i*, *idxs_to_mutate*)

pre_change_range_loop_prep(*inp_batch*, *num_seqs*)

set_perturbation(*replace_with*)

class fastISM.ism_base.**NaiveISM**(*model*, *seq_input_idx=0*, *change_ranges=None*)
Bases: *fastISM.ism_base.ISMBase*

cleanup()

get_ith_output(*inp_batch*, *i*, *idxs_to_mutate*)

pre_change_range_loop_prep(*inp_batch*, *num_seqs*)

run_model(*x*)

6.2 fast_ism module

This module contains the *FastISM* class.

class fastISM.fast_ism.**FastISM**(*model*, *seq_input_idx=0*, *change_ranges=None*, *early_stop_layers=None*, *test_correctness=True*)
Bases: *fastISM.ism_base.ISMBase*

cleanup()

get_ith_output(*inp_batch*, *i*, *idxs_to_mutate*)

pre_change_range_loop_prep(*inp_batch*, *num_seqs*)

prepare_intout_output(*intout_output*, *num_seqs*)

prepare_ith_input(*padded_inputs*, *i*, *idxs_to_mutate*)

run_model(*inputs*)

test_correctness(*batch_size=10*, *replace_with=0*, *atol=1e-06*)
Verify that outputs are correct by matching with Naive ISM. Running on small examples so as to not take too long.

Hence not comparing runtime against Naive ISM implementation, which requires bigger inputs to offset overheads.

TODO: ensure generated data is on GPU already before calling either method (for speedup)

time_batch(*seq_batch*)

6.3 fast_ism_utils module

class fastISM.fast_ism_utils.**GraphSegment**(*start_node*, *input_seqlen*, *input_perturbed_ranges*)

Bases: object

input_unperturbed_width()

output_perturbed_width()

update_forward_output(*input_unperturbed_slices*, *input_unperturbed_padding*, *output_seqlen*, *output_perturbed_ranges*)

update_num_filters(*num_out_filters*)

class fastISM.fast_ism_utils.**SliceAssign**(*a_dim*, *b_dim*)

Bases: tensorflow.python.keras.engine.base_layer.Layer

call(*inputs*)

GOAL: a[:,i:min(i+b.shape[1], a.shape[1])] = b clip b if i+b.shape[1] exceeds width of a, guarantee width of output is same as a. This could happen when a layer's output (b) feeds into multiple layers, but some layers don't need all positions of b (can happen near the edges). See test_skip_then_mxp of test/test_simple_skip_conn_architectures.py

For Cropping1D layers, i can also be negative, which needs to be handled separately.

Parameters **inputs** (*[type]*) – [description]

Returns [description]

Return type [type]

fastISM.fast_ism_utils.**compute_segment_change_ranges**(*model*, *nodes*, *edges*, *inbound_edges*, *node_to_segment*, *stop_segment_idxs*, *input_seqlen*, *input_filters*, *input_change_ranges*, *seq_input_idx*)

for each segment, given input change range compute (ChangeRangesBase.forward_compose):

- input range of intermediate output required
- offsets for input tensor wrt intermediate output
- output seqlen
- output change range
- number of filters in output.

Starts only from sequence input that is changed. Does not deal with alternate inputs.

Forward propagation through network one segment at a time till a segment in stop_segments_idxs is hit. Computes the change ranges for each segment and propagates to the next segment.

fastISM.fast_ism_utils.**generate_fast_ism_model**(*model*, *nodes*, *edges*, *inbound_edges*, *outputs*, *node_to_segment*, *stop_segment_idxs*, *alternate_input_segment_idxs*, *segments*)

fastISM.fast_ism_utils.**generate_fast_ism_subgraph**(*current_node*, *node_edge_to_tensor*, *input_tensors*, *input_specs*, *nodes*, *edges*, *inbound_edges*, *node_to_segment*, *stop_segment_idxs*, *alternate_input_segment_idxs*, *segments*)

fastISM.fast_ism_utils.**generate_intermediate_output_model**(*model*, *nodes*, *edges*, *inbound_edges*, *outputs*, *node_to_segment*, *stop_segment_idxs*)

fastISM.fast_ism_utils.**generate_intermediate_output_subgraph**(*current_node*, *node_to_tensor*, *output_tensor_names*, *nodes*, *edges*, *inbound_edges*, *node_to_segment*, *stop_segment_idxs*)

fastISM.fast_ism_utils.**generate_models**(*model*, *seqlen*, *num_chars*, *seq_input_idx*, *change_ranges*, *early_stop_layers=None*)

fastISM.fast_ism_utils.**label_alternate_input_segment_idxs**(*current_node*, *nodes*, *edges*, *node_to_segment*, *stop_segment_idxs*, *alternate_input_segment_idxs*, *segment_idx*)

fastISM.fast_ism_utils.**label_stop_descendants**(*current_node*, *nodes*, *edges*, *node_to_segment*, *segment_idx*)

fastISM.fast_ism_utils.**process_alternate_input_node**(*current_node*, *node_edge_to_tensor*, *input_tensors*, *input_specs*, *nodes*, *edges*, *inbound_edges*, *node_to_segment*, *alternate_input_segment_idxs*)

fastISM.fast_ism_utils.**resolve_multi_input_change_ranges**(*input_change_ranges_list*)

For AGGREGATE_LAYERS such as Add, the different inputs have different change ranges. For the change ranges, take the largest range over all input ranges:

e.g. [[(1,3), (4,6)], [(2,4), (4,5)]] -> [(1,4), (3,6)] input1 -^ input2 -^

Parameters **input_change_ranges_list** – list of list of tuples. Inner lists must

have same length, where each ith tuple corresponds to ith mutation in the input (ith input change range). :type input_change_ranges_list: list[list[tuple]] :return: Resolved input change ranges. All ranges must have the same width. :rtype: list[tuple]

fastISM.fast_ism_utils.**segment_model**(*model*, *nodes*, *edges*, *inbound_edges*, *seq_input_idx*, *early_stop_layers*)

fastISM.fast_ism_utils.**segment_subgraph**(*current_node*, *nodes*, *edges*, *inbound_edges*, *node_to_segment*, *stop_segment_idxs*, *segment_idx*, *num_convs_in_cur_segment*)

fastISM.fast_ism_utils.**update_stop_segments**(*current_node*, *nodes*, *edges*, *node_to_segment*, *stop_segment_idxs*)

6.4 change_range module

class fastISM.change_range.**ChangeRangesBase**(*config*)

Bases: object

Base class for layer-specific computations of which indices of the output are changed when list of input changed indices are specified. Conversely, given output ranges of indices that need to be produced by the layer, compute the input ranges that will be required for the same.

In addition, given an input. . . .

TODO: document better and with examples!

backward(*output_select_ranges*)

forward(*input_seqlen*, *input_change_ranges*)

list of tuples. e.g. [(0,1), (1,2), (2,3). . .] if single bp ISM

static forward_compose(*change_ranges_objects_list*, *input_seqlen*, *input_change_ranges*)

validate_config()

class fastISM.change_range.**Conv1DChangeRanges**(*config*)

Bases: *fastISM.change_range.ChangeRangesBase*

backward(*output_select_ranges*)

forward(*input_seqlen*, *input_change_ranges*)

list of tuples. e.g. [(0,1), (1,2), (2,3). . .] if single bp ISM

validate_config()

class fastISM.change_range.**Cropping1DChangeRanges**(*config*)

Bases: *fastISM.change_range.ChangeRangesBase*

backward(*output_select_ranges*)

forward(*input_seqlen*, *input_change_ranges*)

list of tuples. e.g. [(0,1), (1,2), (2,3). . .] if single bp ISM

validate_config()

class fastISM.change_range.**Pooling1DChangeRanges**(*config*)

Bases: *fastISM.change_range.ChangeRangesBase*

backward(*output_select_ranges*)

forward(*input_seqlen*, *input_change_ranges*)

list of tuples. e.g. [(0,1), (1,2), (2,3). . .] if single bp ISM

validate_config()

fastISM.change_range.**get_int_if_tuple**(*param*, *idx=0*)

fastISM.change_range.**not_supported_error**(*message*)

6.5 flatten_model module

This module implements functions required to take an arbitrary Keras model and reduce them to a graph representation that is then manipulated by *fast_ism_utils*.

fastISM.flatten_model.**get_flattened_graph**(*model*, *is_subgraph=False*)
[summary]

Parameters

- **model** (*[type]*) – [description]
- **is_subgraph** (*bool, optional*) – [description], defaults to False

Returns [description]

Return type [type]

fastISM.flatten_model.**is_bipartite**(*edges*)

fastISM.flatten_model.**is_consistent**(*edges*, *inbound_edges*)

fastISM.flatten_model.**is_input_layer**(*layer*)
Checks if layer is an input layer

Parameters **layer** (*tf.keras.layers*) – A Keras layer

Returns True if layer is input layer, else False

Return type bool

fastISM.flatten_model.**list_replace**(*l*, *old*, *new*)

fastISM.flatten_model.**node_is_layer**(*node_name*)

fastISM.flatten_model.**strip_subgraph_names**(*name*, *subgraph_names*)
subgraph_name1/subgraph_name2/layer/name -> layer/name

fastISM.flatten_model.**viz_graph**(*nodes*, *edges*, *outpath*)

CHAPTER 7

Changelog

7.1 Unreleased

7.2 0.5.0 - 2022-02-08

7.2.1 Added

- Cropping1D Support
- User specified stop layers (undocumented)
- Support for MultiHeadAttention layers

7.2.2 Changed

- Refinements to segmenting
- Segment starting with see-through layers followed by Conv1Ds with valid padding are kept in one segment
- Layers are duplicated with `from_config` and `get_config`
- Generalized pooling layers and added ability to add custom pooling layers

7.2.3 Fixed

- Runs for batch size 1
- Multi-input layers that had the same input twice (e.g. Add()([x,x])) would not run, fixed this
- Support for newer versions of tensorflow which changed sub-models class from keras to tf.keras (in `flatten_model`)
- Stop layers were traversed redundantly

7.3 0.4.0 - 2020-09-16

7.3.1 Added

- Sequences for benchmarking in notebooks dir and a notebook to process the sequence
- Benchmarking notebooks
- Notebook to time Basset conv and fc separately
- Ability to specify custom mutations
- For each mutation, models only run on input sequences for which character is different from mutation. As a result, each batch usually has a different size. This is slow for the first few batches as it entails a one-time cost.
- Lots of documentation and a logo!

7.3.2 Changed

- Models updated:
  - Activation added to Basset
  - Num output for Basset and Factorized Basset
  - For BPNet, only one channel output and one counts instead of two

7.3.3 Fixed

- FastISM object would keep intermediate outputs of a batch even after it was used, as a result it would occupy extra memory. Get rid of such objects now through a `cleanup()` function. This has stopped GPU Resource errors that popped up after running a few batches

7.4 0.3.0 - 2020-08-24

7.4.1 Added

- Support for multi-input models where alternate input does not merge with primary sequence input before a stop layer.
- Support for layers that dependend on exact order of inputs, e.g. Subtract and Concat.

7.5 0.2.0 - 2020-08-22

7.5.1 Added

- Support for recursively defined networks with 3 test cases
- This Changelog file.

7.5.2 Changed

- BPNet test cases atol changed to 1e-5 so they pass deterministically

7.6 0.1.3 - 2020-08-21

7.6.1 Added

- First PyPI release and tagged version
- Tested and working on non-recursively defined single-input, single and multi-output architectures
- Tested and working on arcitectures with skip connections

The format is based on Keep a Changelog.

CHAPTER 8

Citation

fastISM: Performant *in-silico* saturation mutagenesis for convolutional neural networks; Surag Nair, Avanti Shrikumar, Anshul Kundaje (Bioinformatics 2022) http://doi.org/10.1093/bioinformatics/btac135

Python Module Index

f

Index

L

N

O

P

R

S

T

U

V