osknExample

February 26, 2024

In this notebook, I'll go through a full example of using BPReveal to analyze some chip-nexus data. These are the same data that were used in our original paper: Avsec, Ž., Weilert, M., Shrikumar, A. et al. Base-resolution models of transcription-factor binding reveal soft motif syntax. Nat Genet 53, 354–366 (2021). https://doi.org/10.1038/s41588-021-00782-6

You can download the data from https://zenodo.org/record/3371216#.Y0muwFLMKAQ , but be aware that it's 30 GB of data. I (Charles McAnany) also have a local copy at Stowers, so if you're at Stowers, just let me know and I'll point you in the right direction. I've copied the idr-optimal-set.summit.bed files and the counts.neg.bw and counts.pos.bw files from the downloaded data into my working directory.

1 Setup

Here, I'm just importing stuff and configuring global variables. This is the section you'll probably want to adjust if you want to work through this tutorial yourself. In particular, you'll want to change BASE_DIRECTORY, WORKING_DIRECTORY, DATA_DIRECTORY, and SLURM_CONFIG.

[70]:
```python
# This is specific to running the job on teak (my workstation) since I need to
#add bedtools to my path. You will probably not need to do this.
import os
os.environ["PATH"] = os.environ["PATH"] + ":/n/apps/CentOS7/bin/"
import bpreveal.utils as utils
import bpreveal.tools.plots as bprplots
from bpreveal.tools.slurm import configSlurmLocal, jobsLocal
import json
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,8]
plt.rcParams['figure.dpi'] = 150
import numpy as np

import pybedtools
import pysam
import pyBigWig
import h5py
```

[71]:
```python
#Here, I'll set a few constants that will be applicable throughout the project.
```

```
BASE_DIRECTORY="/n/projects/cm2363/bpreveal"
WORKING_DIRECTORY="/bigscratch/bpreveal/oskn"
#The data directory is where I've unpacked the zenodo archive.
DATA_DIRECTORY="/bigscratch/oskn"
#I have a little script here that renames motifs in modiscolite. This will be␣
 ↪unnecessary with new versions of modiscolite.
SCRIPTS_DIR="/n/projects/cm2363/manuscript-bpreveal/src"
SLURM_CONFIG=configSlurmLocal(["/home/cm2363/.zshrc"],
                         "bpreveal-teak", WORKING_DIRECTORY, cpus=24, memory=50)

SLURM_CMD_CPU = jobsLocal
SLURM_CMD_GPU = jobsLocal
NUM_THREADS = 24

GENOME_FASTA="/bigscratch/genomes/mm10.fa"
TF_NAMES = ["oct4", "sox2", "klf4", "nanog"] #The names of the factors we'll␣
 ↪use.
                                             #For consistency, I'm always going
                                             #to use lowercase.
TEST_CHROMS = ["chr" + str(x) for x in [1,8,9]]
VAL_CHROMS = ["chr" + str(x) for x in [2,3,4]]
TRAIN_CHROMS = ["chr" + str(x) for x in [5,6,7,10,11,12,13,14,15,16,17,18,19]]
```

[3]:
```
!mkdir -p {WORKING_DIRECTORY}/input
!mkdir -p {WORKING_DIRECTORY}/bed
!mkdir -p {WORKING_DIRECTORY}/json
!mkdir -p {WORKING_DIRECTORY}/logs
!mkdir -p {WORKING_DIRECTORY}/models
!mkdir -p {WORKING_DIRECTORY}/modisco
!mkdir -p {WORKING_DIRECTORY}/pred
!mkdir -p {WORKING_DIRECTORY}/shap
!mkdir -p {WORKING_DIRECTORY}/slurm
!mkdir -p {WORKING_DIRECTORY}/scan
!ls -l {WORKING_DIRECTORY}
```

```
total 12
drwxr-xr-x 2 cm2363 domain users  255 Feb 21 15:26 bed
drwxr-xr-x 2 cm2363 domain users   92 Feb 21 15:27 input
drwxr-xr-x 2 cm2363 domain users 4096 Feb 22 10:23 json
drwxr-xr-x 2 cm2363 domain users    6 Feb 21 15:23 logs
drwxr-xr-x 9 cm2363 domain users  325 Feb 22 10:23 models
drwxr-xr-x 2 cm2363 domain users    6 Feb 21 15:23 modisco
drwxr-xr-x 2 cm2363 domain users 4096 Feb 22 10:26 pred
drwxr-xr-x 2 cm2363 domain users    6 Feb 21 15:23 scan
drwxr-xr-x 2 cm2363 domain users    6 Feb 21 15:23 shap
drwxr-xr-x 2 cm2363 domain users 4096 Feb 22 10:55 slurm
```

[4]:
```
!ls -lh {DATA_DIRECTORY}/
```

```
total 4.0K
drwxr-xr-x 2 cm2363 domain users 4.0K Feb 21 15:21 bigwigs
drwxr-xr-x 2 cm2363 domain users  182 Feb 21 15:20 klf4
drwxr-xr-x 2 cm2363 domain users  141 Feb 21 15:20 nanog
drwxr-xr-x 2 cm2363 domain users  141 Feb 21 15:20 oct4
drwxr-xr-x 2 cm2363 domain users   48 Feb 21 15:20 patchcap
drwxr-xr-x 2 cm2363 domain users   94 Feb 21 15:20 peaks-bak
drwxr-xr-x 2 cm2363 domain users  141 Feb 21 15:20 sox2
```

[5]:
```
#The first thing I need to do is prepare input files in order to train a bias␣
 ↪model.
#But what shall I use for bias? I have two options: I can either use background
#regions from the actual chip-nexus experiments, or I can use the patchcap␣
 ↪track.
#If I were to use background regions, I'd have to have a stringent way to␣
 ↪determine
#when a region is not bound, and the data are noisy enough that this might be a
#tough call. I'll train up the bias model on patchcap data instead.

#In order to train that model, I'll need a couple things:
# 1. The bias data. I'm going to just use the patchcap bigwigs from the paper,␣
 ↪no biggie.
# 2. A set of regions to train on. I'll make these in a minute.
# 3. A model architecture. I have to decide on this right now, because
#    it will determine the size of the regions I train on.

#I'll use a standard BPNet architecture, but with few filters since it's␣
 ↪learning
#something so simple. With a 9-layer network, and a 25 bp input filter and 25 bp
#output filter, using 1000 bp output windows, I can calculate the input size:
```

[6]:
```
OUTPUT_LENGTH=1000
input_length_str = !lengthCalc --output-len {OUTPUT_LENGTH} \
                               --n-dil-layers 9 \
                               --conv1-kernel-size 7 \
                               --profile-kernel-size 7
INPUT_LENGTH=int(input_length_str[0])
print(INPUT_LENGTH)
RECEPTIVE_FIELD=INPUT_LENGTH - OUTPUT_LENGTH+1
print(RECEPTIVE_FIELD)
MAX_JITTER = 100
BUFFER = (INPUT_LENGTH - OUTPUT_LENGTH) // 2
print(BUFFER)
```

```
3056
2057
```

```
1028
```

[7]:
```
#Okay, great. I need to make sure that the regions I train on have valid DNA
#within 3092/2 bases of the middle of the window.
#For clarity, here are some dimensions:

#
#
#            |<---  2057 bp (Receptive field) --->|
#   |<--------------------   3056 bp (Input length) ----------------------->|
#   SEQUENCESEQUENCESEQUENCESEQUENCESEQUENCESEQUENCESEQUENCESEQUENCESEQUENCE
#   \        \                                    /                        /
#     \        \                                /                        /
#       \        \                            /                        /
#         \        \                        /                        /
#           \        \                    /                        /
#             \        \                /                        /
#               \        \            /                        /
#                 \        \        /                        /
#                   \        \    /                        /
#                     \        \/                        /
#                      PROFILEPROFILEPROFILEPROFILEPROFIL
#                      |<--- 1000 bp (Output length) --->|
#    |<--- 1028 bp --->|
#
# During training, we also shift the regions around by a little bit, a process
# called jittering. We'll use a maximum jitter of 100.
```

[8]:
```
#In order to generate bias regions, I need to get the actual training regions.
#This is not really part of bpreveal, but I do have a few utility scripts in
#the repo to help with this.
#I'm going to combine the called peaks, make sure there's valid genome under
#all of them (i.e., no "N" nucleotides within the receptive field.),
#then split them into train, validation, and test splits.
```

[9]:
```
bigwigFileNames = [[DATA_DIRECTORY + "/" + tfName + "/counts." + strand + ".bw"
                   for strand in ["pos", "neg"]]
                  for tfName in TF_NAMES]
print(bigwigFileNames)
summitBedFnames = [DATA_DIRECTORY + "/" + tfName + "/idr-optimal-set.summit.bed"
                   for tfName in TF_NAMES]
#I'm using all of the peaks we ever called, so here are some more. It rarely␣
 ↪hurts to provide the model with
# additional peaks.
summitBedFnames += [DATA_DIRECTORY + "/peaks-bak/" + tfName + ".bed"
                    for tfName in TF_NAMES]
print(summitBedFnames)
```

```
#And I need to make bigwig specs, for the upcoming json.
#The bigwig spec needs to list max and min quantiles.
headSpec = [{"bigwig-names" : flist, "max-quantile" : 1, "min-counts" : 1}
        for flist in bigwigFileNames]
print(headSpec)
```

```
[['/bigscratch/oskn/oct4/counts.pos.bw', '/bigscratch/oskn/oct4/counts.neg.bw'],
['/bigscratch/oskn/sox2/counts.pos.bw', '/bigscratch/oskn/sox2/counts.neg.bw'],
['/bigscratch/oskn/klf4/counts.pos.bw', '/bigscratch/oskn/klf4/counts.neg.bw'],
['/bigscratch/oskn/nanog/counts.pos.bw',
'/bigscratch/oskn/nanog/counts.neg.bw']]
['/bigscratch/oskn/oct4/idr-optimal-set.summit.bed', '/bigscratch/oskn/sox2/idr-
optimal-set.summit.bed', '/bigscratch/oskn/klf4/idr-optimal-set.summit.bed',
'/bigscratch/oskn/nanog/idr-optimal-set.summit.bed', '/bigscratch/oskn/peaks-
bak/oct4.bed', '/bigscratch/oskn/peaks-bak/sox2.bed', '/bigscratch/oskn/peaks-
bak/klf4.bed', '/bigscratch/oskn/peaks-bak/nanog.bed']
[{'bigwig-names': ['/bigscratch/oskn/oct4/counts.pos.bw',
'/bigscratch/oskn/oct4/counts.neg.bw'], 'max-quantile': 1, 'min-counts': 1},
{'bigwig-names': ['/bigscratch/oskn/sox2/counts.pos.bw',
'/bigscratch/oskn/sox2/counts.neg.bw'], 'max-quantile': 1, 'min-counts': 1},
{'bigwig-names': ['/bigscratch/oskn/klf4/counts.pos.bw',
'/bigscratch/oskn/klf4/counts.neg.bw'], 'max-quantile': 1, 'min-counts': 1},
{'bigwig-names': ['/bigscratch/oskn/nanog/counts.pos.bw',
'/bigscratch/oskn/nanog/counts.neg.bw'], 'max-quantile': 1, 'min-counts': 1}]
```

[10]:
```
prepareBedPeaksConfig = {
    "heads" : headSpec,
    "splits" : {"test-chroms"  : TEST_CHROMS,
                "val-chroms"   : VAL_CHROMS,
                "train-chroms" : TRAIN_CHROMS,
                "regions" : summitBedFnames},
    "genome" : GENOME_FASTA,
    "output-length" : OUTPUT_LENGTH,
    "input-length" : INPUT_LENGTH,
    "max-jitter" : MAX_JITTER,
    "output-prefix" : WORKING_DIRECTORY + "/bed/peak",
    "resize-mode" : "center",
    "remove-overlaps" : True,
    "num-threads" : NUM_THREADS,
    "overlap-max-distance" : 100,
    "verbosity" : "INFO"}

with open(WORKING_DIRECTORY + "/json/prepareBedPeaks.json", "w") as fp:
    json.dump(prepareBedPeaksConfig, fp, indent=4)
    print(json.dumps(prepareBedPeaksConfig, indent=4))
```

```
{
    "heads": [
    {
        "bigwig-names": [
            "/bigscratch/oskn/oct4/counts.pos.bw",
            "/bigscratch/oskn/oct4/counts.neg.bw"
        ],
        "max-quantile": 1,
        "min-counts": 1
    },
    {
        "bigwig-names": [
            "/bigscratch/oskn/sox2/counts.pos.bw",
            "/bigscratch/oskn/sox2/counts.neg.bw"
        ],
        "max-quantile": 1,
        "min-counts": 1
    },
    {
        "bigwig-names": [
            "/bigscratch/oskn/klf4/counts.pos.bw",
            "/bigscratch/oskn/klf4/counts.neg.bw"
        ],
        "max-quantile": 1,
        "min-counts": 1
    },
    {
        "bigwig-names": [
            "/bigscratch/oskn/nanog/counts.pos.bw",
            "/bigscratch/oskn/nanog/counts.neg.bw"
        ],
        "max-quantile": 1,
        "min-counts": 1
    }
],
"splits": {
    "test-chroms": [
        "chr1",
        "chr8",
        "chr9"
    ],
    "val-chroms": [
        "chr2",
        "chr3",
        "chr4"
    ],
    "train-chroms": [
        "chr5",
        "chr6",
        "chr7",
                "chr10",
                "chr11",
                "chr12",
                "chr13",
                "chr14",
                "chr15",
                "chr16",
                "chr17",
                "chr18",
                "chr19"
            ],
            "regions": [
                "/bigscratch/oskn/oct4/idr-optimal-set.summit.bed",
                "/bigscratch/oskn/sox2/idr-optimal-set.summit.bed",
                "/bigscratch/oskn/klf4/idr-optimal-set.summit.bed",
                "/bigscratch/oskn/nanog/idr-optimal-set.summit.bed",
                "/bigscratch/oskn/peaks-bak/oct4.bed",
                "/bigscratch/oskn/peaks-bak/sox2.bed",
                "/bigscratch/oskn/peaks-bak/klf4.bed",
                "/bigscratch/oskn/peaks-bak/nanog.bed"
            ]
        },
        "genome": "/bigscratch/genomes/mm10.fa",
        "output-length": 1000,
        "input-length": 3056,
        "max-jitter": 100,
        "output-prefix": "/bigscratch/bpreveal/oskn/bed/peak",
        "resize-mode": "center",
        "remove-overlaps": true,
        "num-threads": 24,
        "overlap-max-distance": 100,
        "verbosity": "INFO"
    }
```

```
[11]: SLURM_CMD_CPU(SLURM_CONFIG, ["prepareBed {0:s}/json/prepareBedPeaks.json".
       ↪format(WORKING_DIRECTORY)],
                   "prepareBedPeaks", ntasks=24, mem=20, parallel=True)
```

```
[11]: '/bigscratch/bpreveal/oskn/slurm/prepareBedPeaks.zsh'
```

```
[12]: backgroundBase = "tileGenome --genome {0:s} --output-length {1:d}␣
       ↪--input-length {2:d} "\
                        "--chrom-edge-boundary 100000 --spacing 10000 --output-bed {3:
       ↪s} "\
                        "{4:s} {5:s}"
      blacklistArgs = "--blacklist {0:s} --blacklist {1:s}".format(
          WORKING_DIRECTORY + "/bed/peak_all.bed",
          WORKING_DIRECTORY + "/bed/peak_reject.bed")

      chromArgs = ' '.join(["--allow-chrom {0:s}".format(c) for c in (TRAIN_CHROMS +␣
       ↪TEST_CHROMS + VAL_CHROMS)])

      cmdGenBackground = backgroundBase.format(
          GENOME_FASTA, OUTPUT_LENGTH, INPUT_LENGTH, WORKING_DIRECTORY + "/bed/
       ↪tiling_all.bed",
          blacklistArgs, chromArgs)

      slurmNameGenBackground = SLURM_CMD_CPU(SLURM_CONFIG, [cmdGenBackground],␣
       ↪"genBackground", ntasks=2, mem=10, parallel=True)
```

[13]:
```
#Now that we have a bed file with all of our training regions in it, we can
#generate the background regions that we'll train the bias model on.
#Note that even though I'm using patchcap data for my bias track, I'm still
#going to train the bias model on unbound regions, so that any effect
#TF binding has on patchcap doesn't show up in my bias model.
#This is another script I wrote, it generates tiling regions across the
#whole genome and then removes regions that overlap your peak set,
#and also only outputs regions that fall in a set percentile of counts.

biasBigwigFnames = [DATA_DIRECTORY + "/patchcap/counts." + strand + ".bw"
                    for strand in ["pos", "neg"]]
print(biasBigwigFnames)
```

```
['/bigscratch/oskn/patchcap/counts.pos.bw',
'/bigscratch/oskn/patchcap/counts.neg.bw']
```

[14]:
```
biasHeadSpec = [{"bigwig-names" : flist, "max-quantile" : 0.6, "min-quantile" :␣
↪0.01}
          for flist in bigwigFileNames]
biasHeadSpec = biasHeadSpec + [{"bigwig-names" : biasBigwigFnames,
                                "max-quantile" : 0.95,
                                "min-quantile" : 0.1} ]
prepareBedNonPeaksConfig = {
    "heads" : biasHeadSpec,
    "splits" : {"test-chroms"  : TEST_CHROMS,
                "val-chroms"   : VAL_CHROMS,
                "train-chroms" : TRAIN_CHROMS,
                "regions" : [WORKING_DIRECTORY + "/bed/tiling_all.bed"]},
    "genome" : GENOME_FASTA,
    "output-length" : OUTPUT_LENGTH,
    "input-length" : INPUT_LENGTH,
    "max-jitter" : MAX_JITTER,
    "output-prefix" : WORKING_DIRECTORY + "/bed/nonpeak",
    "remove-overlaps" : False,
    "resize-mode" : "center",
    "num-threads": NUM_THREADS,
    "verbosity" : "INFO"}

with open(WORKING_DIRECTORY + "/json/prepareBedNonPeaks.json", "w") as fp:
    json.dump(prepareBedNonPeaksConfig, fp)
    print(json.dumps(prepareBedNonPeaksConfig, indent=4))
```

```
{
    "heads": [
        {
            "bigwig-names": [
                "/bigscratch/oskn/oct4/counts.pos.bw",
                "/bigscratch/oskn/oct4/counts.neg.bw"
            ],
            "max-quantile": 0.6,
            "min-quantile": 0.01
        },
        {
            "bigwig-names": [
                "/bigscratch/oskn/sox2/counts.pos.bw",
                "/bigscratch/oskn/sox2/counts.neg.bw"
            ],
            "max-quantile": 0.6,
            "min-quantile": 0.01
        },
        {
            "bigwig-names": [
                "/bigscratch/oskn/klf4/counts.pos.bw",
                "/bigscratch/oskn/klf4/counts.neg.bw"
            ],
            "max-quantile": 0.6,
            "min-quantile": 0.01
        },
        {
            "bigwig-names": [
                "/bigscratch/oskn/nanog/counts.pos.bw",
                "/bigscratch/oskn/nanog/counts.neg.bw"
            ],
            "max-quantile": 0.6,
            "min-quantile": 0.01
        },
        {
            "bigwig-names": [
                "/bigscratch/oskn/patchcap/counts.pos.bw",
                "/bigscratch/oskn/patchcap/counts.neg.bw"
            ],
            "max-quantile": 0.95,
            "min-quantile": 0.1
        }
    ],
    "splits": {
        "test-chroms": [
            "chr1",
            "chr8",
            "chr9"
        ],
        "val-chroms": [
            "chr2",
            "chr3",
            "chr4"
        ],
        "train-chroms": [
            "chr5",
            "chr6",
            "chr7",
            "chr10",
            "chr11",
            "chr12",
            "chr13",
            "chr14",
            "chr15",
            "chr16",
            "chr17",
            "chr18",
            "chr19"
        ],
        "regions": [
            "/bigscratch/bpreveal/oskn/bed/tiling_all.bed"
        ]
    },
    "genome": "/bigscratch/genomes/mm10.fa",
    "output-length": 1000,
    "input-length": 3056,
    "max-jitter": 100,
    "output-prefix": "/bigscratch/bpreveal/oskn/bed/nonpeak",
    "remove-overlaps": false,
    "resize-mode": "center",
    "num-threads": 24,
    "verbosity": "INFO"
}
```

[15]:
```
SLURM_CMD_CPU(SLURM_CONFIG, ["prepareBed {0:s}/json/prepareBedNonPeaks.json".
 ↪format(WORKING_DIRECTORY)],
           "prepareBedNonPeaks", ntasks=2, mem=20, parallel=True)
```

```
[15]: '/bigscratch/bpreveal/oskn/slurm/prepareBedNonPeaks.zsh'
```

Whenever I have a slurm jobs call (jobsGpu, jobsNonGpu, jobsLocal) in this notebook, assume that I've run it. If you run every cell of this notebook without running the batched commands, you will encounter lots of missing file errors.

2 Building the training dataset

```
[16]: #This next step is pretty easy; we just need to pull the sequence and profile
#information into a single hdf5-format file for the training programs to use.

#We'll need to make training and validation sets for both the nonpeaks and
#peaks bed files.

configFnames = []
for split in ["train", "val"]:
    for dataset in ["peak", "nonpeak"]:
        heads = []
        for tfId, tfName in enumerate(TF_NAMES):
            if(dataset == 'peak'):
                heads.append({
                    "revcomp-task-order" : "auto",
                    "bigwig-files" : bigwigFileNames[tfId]})
            else:
                heads.append({
                    "revcomp-task-order" : "auto",
                    "bigwig-files" : biasBigwigFnames})
        config = {"genome" : GENOME_FASTA,
                  "input-length" : INPUT_LENGTH,
                  "output-length" : OUTPUT_LENGTH,
                  "max-jitter" : MAX_JITTER,
                  "regions" : WORKING_DIRECTORY + "/bed/" + dataset + "_" +␣
 ↪split + ".bed",
                  "output-h5" : WORKING_DIRECTORY + "/input/" + dataset + "_" +␣
 ↪split + ".h5",
                  "reverse-complement" : True,
                  "heads" : heads,
                  "verbosity" : "INFO"}
        configFname =WORKING_DIRECTORY + "/json/prepareInput" + dataset + "_" +␣
 ↪split+ ".json"
        with open(configFname, "w") as fp:
            json.dump(config, fp, indent=2)
        configFnames.append(configFname)

SLURM_CMD_CPU(SLURM_CONFIG, ["prepareTrainingData {0:s}".format(configFname)
                for configFname in configFnames],
            "prepareTrainingData", ntasks=2, mem=10, parallel=True)
```

```
```

```
[16]: '/bigscratch/bpreveal/oskn/slurm/prepareTrainingData.zsh'
```

3 Training the bias model

```
[17]: #Okay, so the bed preparation step is done. I didn't spend much time
      #on that since it will be specific to every system you deal with.
      #But now comes the common stuff. And it's (honestly) easier.
```

```
[18]: #To make the model config file, I'll assemble the heads first.
      heads = []
      for tfName in TF_NAMES:
          heads.append({"num-tasks" : 2,
                        "profile-loss-weight" : 1,
                        "head-name" : "patchcap_" + tfName,
                        "counts-loss-weight" : 10,
                        "counts-loss-frac-target" : 0.1})


      #And now the whole config file:
      biasTrainConfig = {
          "settings" : {
              "output-prefix" : WORKING_DIRECTORY + "/models/solo",
              "epochs" : 200,
              "max-jitter" : 100,
              "early-stopping-patience" : 20,
              "batch-size" : 128,
              "learning-rate" : 0.004,
              "learning-rate-plateau-patience" : 5,
              "architecture" : {
                  "architecture-name" : "bpnet",
                  "input-length" : INPUT_LENGTH,
                  "output-length" : OUTPUT_LENGTH,
                  "model-name" : "patchcap",
                  "model-args" : "",
                  "filters" : 16,
                  "layers" : 9,
                  "input-filter-width" : 7,
                  "output-filter-width" : 7
              }
          },
          "train-data" : WORKING_DIRECTORY + "/input/nonpeak_train.h5",
          "val-data" : WORKING_DIRECTORY + "/input/nonpeak_val.h5",
          "heads" : heads,
          "verbosity" : "DEBUG"
      }

      print(json.dumps(biasTrainConfig, indent=4))

      with open(WORKING_DIRECTORY + "/json/trainBias.json", "w") as fp:
          json.dump(biasTrainConfig, fp, indent=4)
```

```
{
    "settings": {
        "output-prefix": "/bigscratch/bpreveal/oskn/models/solo",
        "epochs": 200,
        "max-jitter": 100,
        "early-stopping-patience": 20,
        "batch-size": 128,
        "learning-rate": 0.004,
        "learning-rate-plateau-patience": 5,
        "architecture": {
            "architecture-name": "bpnet",
            "input-length": 3056,
            "output-length": 1000,
            "model-name": "patchcap",
            "model-args": "",
            "filters": 16,
            "layers": 9,
            "input-filter-width": 7,
            "output-filter-width": 7
        }
    },
    "train-data": "/bigscratch/bpreveal/oskn/input/nonpeak_train.h5",
    "val-data": "/bigscratch/bpreveal/oskn/input/nonpeak_val.h5",
    "heads": [
        {
            "num-tasks": 2,
            "profile-loss-weight": 1,
            "head-name": "patchcap_oct4",
            "counts-loss-weight": 10,
            "counts-loss-frac-target": 0.1
        },
        {
            "num-tasks": 2,
            "profile-loss-weight": 1,
            "head-name": "patchcap_sox2",
            "counts-loss-weight": 10,
            "counts-loss-frac-target": 0.1
        },
        {
            "num-tasks": 2,
            "profile-loss-weight": 1,
            "head-name": "patchcap_klf4",
            "counts-loss-weight": 10,
            "counts-loss-frac-target": 0.1
        },
        {
            "num-tasks": 2,
            "profile-loss-weight": 1,
            "head-name": "patchcap_nanog",
            "counts-loss-weight": 10,
            "counts-loss-frac-target": 0.1
        }
    ],
    "verbosity": "DEBUG"
}
```

[19]:
```
SLURM_CMD_GPU(SLURM_CONFIG, ["trainSoloModel {0:s} |& showTrainingProgress".
 ↪format(WORKING_DIRECTORY + "/json/trainBias.json")],
       "trainSolo", 10, 30)
```

[19]:
```
'/bigscratch/bpreveal/oskn/slurm/trainSolo.zsh'
```

4 Evaluating the bias model

[82]:
```
#First, we need to make predictions with the bias model. That's another json␣
 ↪file...
biasPredictConfig = {
    "settings" : {
        "output-h5" : WORKING_DIRECTORY + "/pred/patchcap.h5",
        "batch-size" : 128,
        "heads" : 4,

        "architecture" : {
            "model-file" : WORKING_DIRECTORY + "/models/solo.model",
            "input-length" : INPUT_LENGTH,
            "output-length" : OUTPUT_LENGTH
        }
    },
    "genome" : GENOME_FASTA,
    "bed-file" : WORKING_DIRECTORY + "/bed/peak_all.bed",
    "num-threads" : 4,
    "verbosity" : "INFO"
}
```

```
with open(WORKING_DIRECTORY + "/json/predictBias.json", "w") as fp:
    json.dump(biasPredictConfig, fp)
```

[83]:
```
SLURM_CMD_GPU(SLURM_CONFIG, ["makePredictions {0:s}".format(WORKING_DIRECTORY +␣
↪"/json/predictBias.json")],
        "predictSolo", 1, 50)
```

[83]:
```
'/bigscratch/bpreveal/oskn/slurm/predictSolo.zsh'
```

[24]:
```
#And now I need to convert that hdf5 file into a bigwig.
predCmd = "predictToBigwig " +\
          "--h5 {0:s}/pred/patchcap.h5 " +\
          "--bw {0:s}/pred/patchcap_{1:s}.bw "+\
          "--head-id 0 --task-id {2:d} --mode profile "+\
          "--threads 10 --verbose"

SLURM_CMD_CPU(SLURM_CONFIG, [predCmd.format(WORKING_DIRECTORY, strand[0],␣
↪strand[1]) for strand in [("positive", 0), ("negative", 1)]],
           "predToBigwigBias", 10, 20, parallel=True)
```

[24]:
```
'/bigscratch/bpreveal/oskn/slurm/predToBigwigBias.zsh'
```

[ ]:
```

```

[25]:
```
#We should look at how well the model did.
!makeLossPlots --json {WORKING_DIRECTORY}/models/solo.history.json \
               --output {WORKING_DIRECTORY}/models/solo.png
#It's pretty clear that the model overlearned, even with only sixteen filters.
#Interesting. It would be great if the training and validation losses were
#more similar, but it's not a lethal flaw since we don't need to interpret
#the bias model. We should, however, make predictions from it and calculate
#some metrics.
```

[26]:
```
#Note that I've only written bigwigs for the first head - since all heads were␣
↪trained on the same data, I'm going to assume each head performed equally␣
↪well.
```

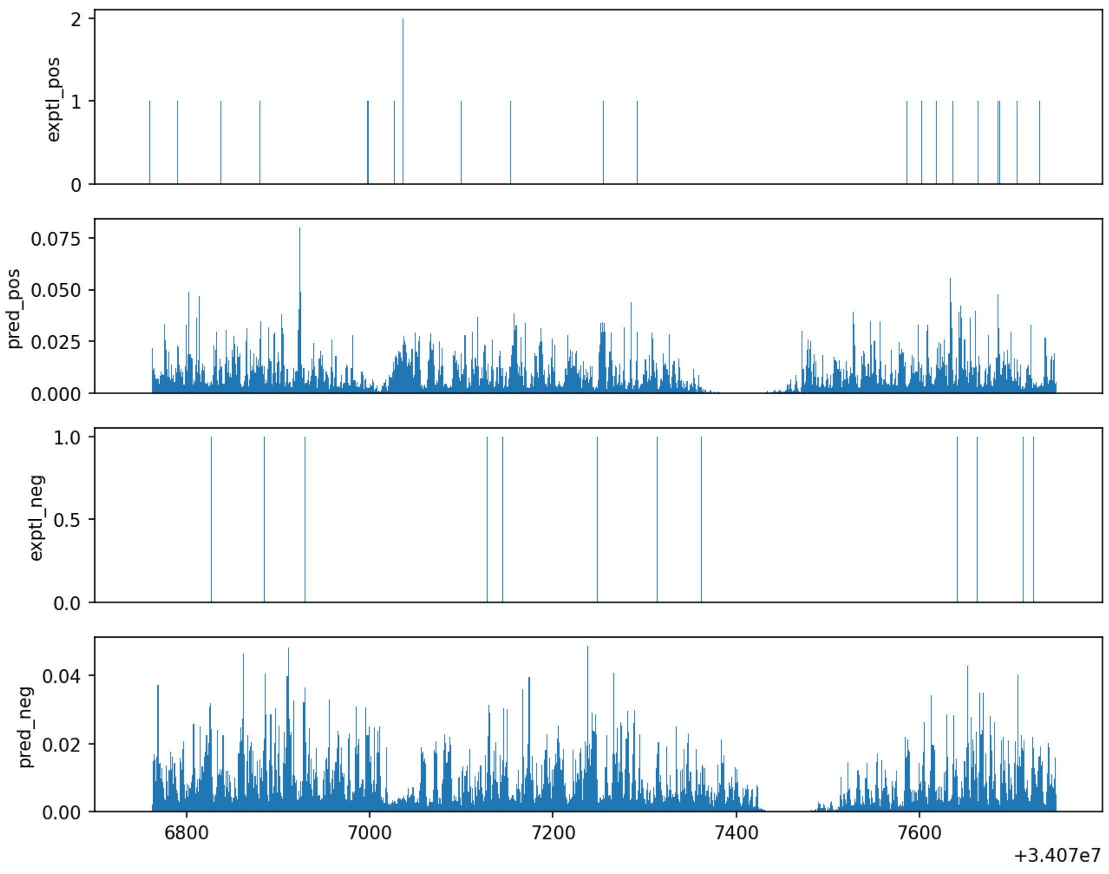
[27]:
```
#We can now calculate some standard metrics on our predictions, though we don't␣
↪yet have anything to compare these to.
!metrics --reference {DATA_DIRECTORY}/patchcap/counts.pos.bw\
         --pred {WORKING_DIRECTORY}/pred/patchcap_positive.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_all.bed \
         --threads 20 --apply-abs --skip-zeroes
```

```
reference /bigscratch/oskn/patchcap/counts.pos.bw predicted
/bigscratch/bpreveal/oskn/pred/patchcap_positive.bw regions
/bigscratch/bpreveal/oskn/bed/peak_all.bed
metric                 0.000000%      25.000000%      50.000000%      75.000000%
```

```
100.000000% regions
mnll                  -2537.942752       -83.252447       -68.759964       -54.629996
-5.458253 116033
jsd                       0.651337         0.794447         0.801281         0.807855
0.832382 116033
pearsonr                 -0.054262         0.056208         0.084920         0.115402
0.348581 116033
spearmanr                -0.062017         0.057837         0.079521         0.100292
0.279783 116033
Counts pearson    0.144117
Counts spearman   0.151393
```

[28]:
```python
#Let's also take a quick look at the generated bigwigs.

def plotBws(bwNames, titles, chrom, start, stop):

    for i, bwName in enumerate(bwNames):
        plt.subplot(100*len(bwNames)+10+(i+1))
        bw = pyBigWig.open(bwName)
        bwVals = np.nan_to_num(bw.values(chrom, start, stop))
        #plt.xlim(0,stop-start)
        plt.bar(range(start, stop), bwVals, width=1)
        plt.ylabel(titles[i])
        if(i < len(bwNames)-1):
            plt.xticks([])
```

[29]:
```python
plotBws([DATA_DIRECTORY + "/patchcap/counts.pos.bw",
         WORKING_DIRECTORY + "/pred/patchcap_positive.bw",
         DATA_DIRECTORY + "/patchcap/counts.neg.bw",
         WORKING_DIRECTORY + "/pred/patchcap_negative.bw"],
        ["exptl_pos", "pred_pos", "exptl_neg", "pred_neg"], "chr1", 34076750,␣
 ↪34077750)
```



```
[30]: #Huh. With so little patchcap data, it's really hard to tell if the model is␣
      ↪doing a good job.
      #In any event, it's time to train the transformation model up.
```

5 Training the transformation model

```
[31]: heads = []
      for tfName in TF_NAMES:
          heads.append({"num-tasks" : 2,
                        "profile-loss-weight" : 1,
                        "head-name" : "patchcap_" + tfName,
                        "counts-loss-weight" : 100,
                        "counts-loss-frac-target" : 0.1})

      transformationTrainConfig = {
          "settings" : {
              "output-prefix" : WORKING_DIRECTORY + "/models/transformation",
              "epochs" : 200,
              "early-stopping-patience" : 20,
              "batch-size" : 128,
              "learning-rate" : 0.04, #Note the very aggressive LR; we can do this␣
      ↪because there are so few parameters.
              "learning-rate-plateau-patience" : 5,
              "solo-model-file" : WORKING_DIRECTORY + "/models/solo.model",
              "input-length" : INPUT_LENGTH,
              "output-length" : OUTPUT_LENGTH,
              "max-jitter" : 100,
              "profile-architecture" : {
                  "name" : "simple",
                  "types" : ["linear", "sigmoid"]},
              "counts-architecture" : {
                  "name" : "simple",
                  "types" : ["linear", "sigmoid"]}},

          "train-data" : WORKING_DIRECTORY+ "/input/peak_train.h5",
          "val-data" : WORKING_DIRECTORY + "/input/peak_val.h5",
          "heads" : heads,
          "verbosity" : "DEBUG"
      }

      print(json.dumps(transformationTrainConfig, indent=2))

      with open(WORKING_DIRECTORY + "/json/trainTransformation.json", "w") as fp:
          json.dump(transformationTrainConfig, fp)
```

```
{
  "settings": {
    "output-prefix": "/bigscratch/bpreveal/oskn/models/transformation",
    "epochs": 200,
    "early-stopping-patience": 20,
    "batch-size": 128,
    "learning-rate": 0.04,
    "learning-rate-plateau-patience": 5,
    "solo-model-file": "/bigscratch/bpreveal/oskn/models/solo.model",
    "input-length": 3056,
    "output-length": 1000,
    "max-jitter": 100,
    "profile-architecture": {
      "name": "simple",
      "types": [
        "linear",
        "sigmoid"
      ]
    },
    "counts-architecture": {
      "name": "simple",
      "types": [
        "linear",
        "sigmoid"
      ]
    }
  },
  "train-data": "/bigscratch/bpreveal/oskn/input/peak_train.h5",
  "val-data": "/bigscratch/bpreveal/oskn/input/peak_val.h5",
  "heads": [
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "patchcap_oct4",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "patchcap_sox2",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "patchcap_klf4",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "patchcap_nanog",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1
    }
  ],
  "verbosity": "DEBUG"
}
```

[41]:
```
jobSpecs = []
SLURM_CMD_CPU(SLURM_CONFIG, ["trainTransformationModel {0:s} |&␣
↪showTrainingProgress".format(WORKING_DIRECTORY + "/json/trainTransformation.
↪json")],
        "trainTransformation")
```

[41]:
```
'/bigscratch/bpreveal/oskn/slurm/trainTransformation.zsh'
```

```
[85]: #Let's go ahead and make predictions...
      transformPredictConfig = {
          "settings" : {
              "output-h5" : WORKING_DIRECTORY + "/pred/transform.h5",
              "batch-size" : 128,
              "heads" : 4,

              "architecture" : {
                  "model-file" : WORKING_DIRECTORY + "/models/transformation.model",
                  "input-length" : INPUT_LENGTH,
                  "output-length" : OUTPUT_LENGTH
              }
          },
          "genome" : GENOME_FASTA,
          "bed-file" : WORKING_DIRECTORY + "/bed/peak_all.bed",
          "num-threads" : 4,
          "verbosity" : "DEBUG"
      }

      print(transformPredictConfig)

      with open(WORKING_DIRECTORY + "/json/predictTransformation.json", "w") as fp:
          json.dump(transformPredictConfig, fp)
```

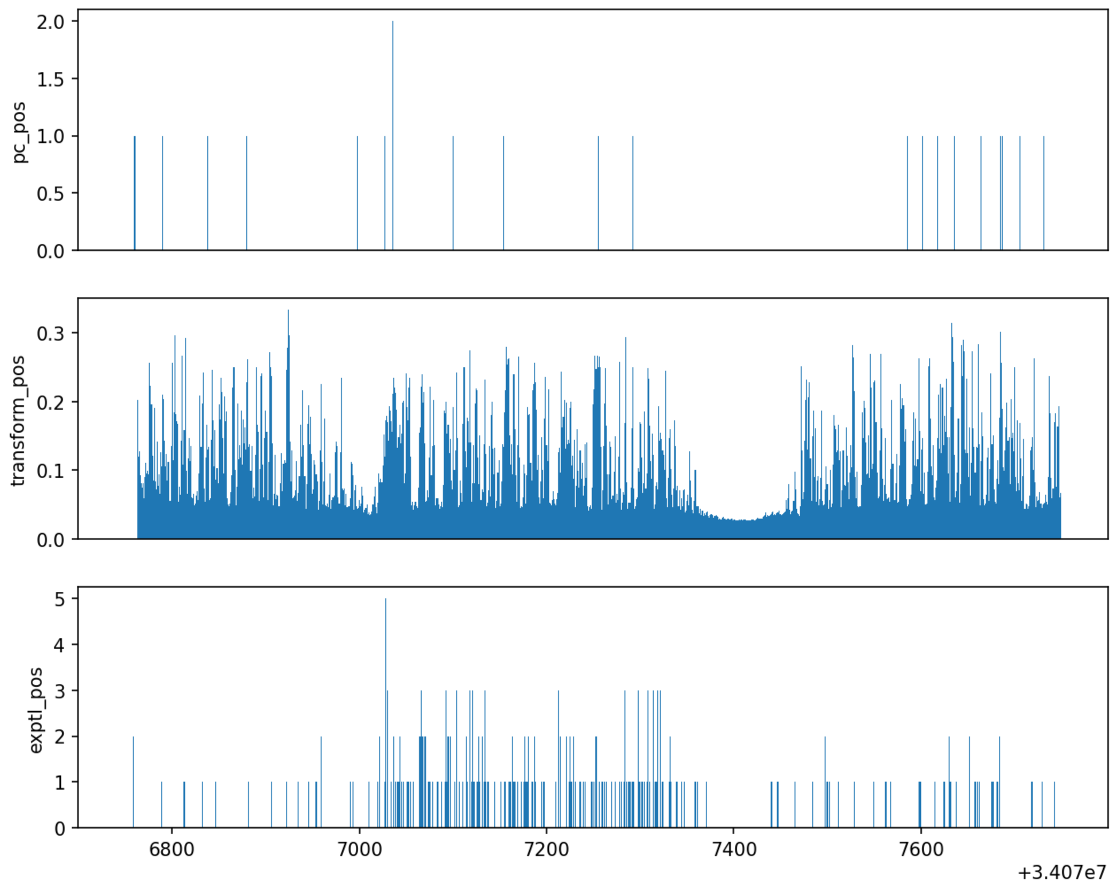
```
{'settings': {'output-h5': '/bigscratch/bpreveal/oskn/pred/transform.h5',
'batch-size': 128, 'heads': 4, 'architecture': {'model-file':
'/bigscratch/bpreveal/oskn/models/transformation.model', 'input-length': 3056,
'output-length': 1000}}, 'genome': '/bigscratch/genomes/mm10.fa', 'bed-file':
'/bigscratch/bpreveal/oskn/bed/peak_all.bed', 'num-threads': 4, 'verbosity':
'DEBUG'}
```

```
[43]: SLURM_CMD_GPU(SLURM_CONFIG, ["makePredictions {0:s}".format(WORKING_DIRECTORY +␣
       ↪"/json/predictTransformation.json")],
              "predictTransformation")

      predCmd = "predictToBigwig " +\
                "--h5 {0:s}/pred/transform.h5 " +\
                "--bw {0:s}/pred/transform_{1:s}.bw "+\
                "--head-id 0 --task-id {2:d} --mode profile "+\
                "--threads 10 --verbose"

      SLURM_CMD_CPU(SLURM_CONFIG,
          [predCmd.format(WORKING_DIRECTORY, strand[0], strand[1]) for strand in␣
       ↪[("positive", 0), ("negative", 1)]],
          "predToBigwigTransform", 10, 10, parallel=True)
```

```
[43]: '/bigscratch/bpreveal/oskn/slurm/predToBigwigTransform.zsh'
```

```
[ ]:
```

```
[44]: plotBws([DATA_DIRECTORY + "/patchcap/counts.pos.bw",
               WORKING_DIRECTORY + "/pred/transform_positive.bw",
               DATA_DIRECTORY + "/nanog/counts.pos.bw"],
              ["pc_pos", "transform_pos", "exptl_pos"], "chr1", 34076750, 34077750)
```



```
[45]: #Of course these aren't a good match, but that's partly the point - the bald␣
      ↪spot that the patchcap model predicts seems to also be present in the Nanog␣
      ↪and Oct4 experimental data,
      #and this suggests that that bald spot is an artifact.
```

6 Training the combined model

```
[46]: heads = []
      for tfName in TF_NAMES:
          heads.append({"num-tasks" : 2,
                        "profile-loss-weight" : 1,
                        "head-name" : "combined_" + tfName,
                        "counts-loss-weight" : 100,
                        "counts-loss-frac-target" : 0.1,
                        "use-bias-counts" : False})


      #And now the whole config file:
      combinedTrainConfig = {
          "settings" : {
              "output-prefix" : WORKING_DIRECTORY + "/models/joint",
              "epochs" : 200,
              "early-stopping-patience" : 20,
              "batch-size" : 128,
              "learning-rate" : 0.004,
              "learning-rate-plateau-patience" : 5,
              "max-jitter" : 100,
              "transformation-model" : {
                  "transformation-model-file" : WORKING_DIRECTORY + "/models/
       ↪transformation.model"
              },
              "architecture" : {
                  "architecture-name" : "bpnet",
                  "input-length" : INPUT_LENGTH,
                  "output-length" : OUTPUT_LENGTH,
                  "model-name" : "joint",
                  "model-args" : "",
                  "filters" : 64,
                  "layers" : 9,
                  "input-filter-width" : 7,
                  "output-filter-width" : 7
              }
          },
          "train-data" : WORKING_DIRECTORY + "/input/peak_train.h5",
          "val-data" : WORKING_DIRECTORY + "/input/peak_val.h5",
          "heads" : heads,
          "verbosity" : "DEBUG"
      }

      print(json.dumps(combinedTrainConfig, indent=2))

      with open(WORKING_DIRECTORY + "/json/trainCombined.json", "w") as fp:
          json.dump(combinedTrainConfig, fp)
```

```
{
  "settings": {
    "output-prefix": "/bigscratch/bpreveal/oskn/models/joint",
    "epochs": 200,
    "early-stopping-patience": 20,
    "batch-size": 128,
    "learning-rate": 0.004,
    "learning-rate-plateau-patience": 5,
    "max-jitter": 100,
    "transformation-model": {
      "transformation-model-file":
"/bigscratch/bpreveal/oskn/models/transformation.model"
    },
    "architecture": {
      "architecture-name": "bpnet",
      "input-length": 3056,
      "output-length": 1000,
      "model-name": "joint",
      "model-args": "",
      "filters": 64,
      "layers": 9,
      "input-filter-width": 7,
      "output-filter-width": 7
    }
  },
  "train-data": "/bigscratch/bpreveal/oskn/input/peak_train.h5",
  "val-data": "/bigscratch/bpreveal/oskn/input/peak_val.h5",
  "heads": [
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "combined_oct4",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1,
      "use-bias-counts": false
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "combined_sox2",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1,
      "use-bias-counts": false
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "combined_klf4",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1,
      "use-bias-counts": false
    },
    {
      "num-tasks": 2,
      "profile-loss-weight": 1,
      "head-name": "combined_nanog",
      "counts-loss-weight": 100,
      "counts-loss-frac-target": 0.1,
      "use-bias-counts": false
    }
  ],
  "verbosity": "DEBUG"
}
```

[47]:
```
SLURM_CMD_GPU(SLURM_CONFIG, ["trainCombinedModel {0:s} |& showTrainingProgress".
  ↪format(WORKING_DIRECTORY + "/json/trainCombined.json")],
        "trainCombined")
```

[47]: '/bigscratch/bpreveal/oskn/slurm/trainCombined.zsh'

[48]:
```
#Let's look at the losses...
!makeLossPlots --json {WORKING_DIRECTORY}/models/joint.history.json --output␣
  ↪{WORKING_DIRECTORY}/models/joint.png
```

[49]:
```
#It's overfitting a bit, maybe next time I'll try with fewer filters.
#But now's the time to make predictions.
```

[86]:
```
combinedPredictConfig = {
    "settings" : {
        "output-h5" : WORKING_DIRECTORY + "/pred/combined.h5",
        "batch-size" : 128,
        "heads" : 4,

        "architecture" : {
            "model-file" : WORKING_DIRECTORY + "/models/joint_combined.model",
            "input-length" : INPUT_LENGTH,
            "output-length" : OUTPUT_LENGTH
        }
    },
    "genome" : GENOME_FASTA,
    "bed-file" : WORKING_DIRECTORY + "/bed/peak_all.bed",
    "num-threads" : 4,
    "verbosity" : "DEBUG"
}

print(combinedPredictConfig)

with open(WORKING_DIRECTORY + "/json/predictCombined.json", "w") as fp:
    json.dump(combinedPredictConfig, fp)
#For the residual model, I just need to change a few terms:
residualPredictConfig = combinedPredictConfig
```

```
residualPredictConfig["settings"]["output-h5"] = WORKING_DIRECTORY + "/pred/
 ↪residual.h5"
residualPredictConfig["settings"]["architecture"]["model-file"] =␣
 ↪WORKING_DIRECTORY + "/models/joint_residual.model"
with open(WORKING_DIRECTORY + "/json/predictResidual.json", "w") as fp:
    json.dump(residualPredictConfig, fp)
```

```
{'settings': {'output-h5': '/bigscratch/bpreveal/oskn/pred/combined.h5', 'batch-
size': 128, 'heads': 4, 'architecture': {'model-file':
'/bigscratch/bpreveal/oskn/models/joint_combined.model', 'input-length': 3056,
'output-length': 1000}}, 'genome': '/bigscratch/genomes/mm10.fa', 'bed-file':
'/bigscratch/bpreveal/oskn/bed/peak_all.bed', 'num-threads': 4, 'verbosity':
'DEBUG'}
```

[88]:
```
SLURM_CMD_GPU(SLURM_CONFIG, ["makePredictions {0:s}".format(WORKING_DIRECTORY +␣
 ↪"/json/predictCombined.json"),
         "makePredictions {0:s}".format(WORKING_DIRECTORY + "/json/
 ↪predictResidual.json")],
        "predictCombined", 1, 50, "10:00:00")

bwCmdBase = "predictToBigwig " +\
          "--h5 {wd:s}/pred/{inf:s}.h5 " +\
          "--bw {wd:s}/pred/{outf:s}.bw "+\
          "--head-id {hid:d} --task-id {tid:d} --mode profile "+\
          "--threads 5 --verbose"
bwCmds = []
for modelType in ["residual", "combined"]:
    for headid, tfname in enumerate(TF_NAMES):
        for tid, strand in enumerate(["positive", "negative"]):
            cmd = bwCmdBase.format(wd=WORKING_DIRECTORY,
                                   inf=modelType,
                                   outf=tfname + "_" + modelType + "_" + strand,
                                   hid=headid, tid=tid)
            bwCmds.append(cmd)

SLURM_CMD_CPU(SLURM_CONFIG, bwCmds,
           "predToBigwigCombined", 5, 5, parallel=True)
```

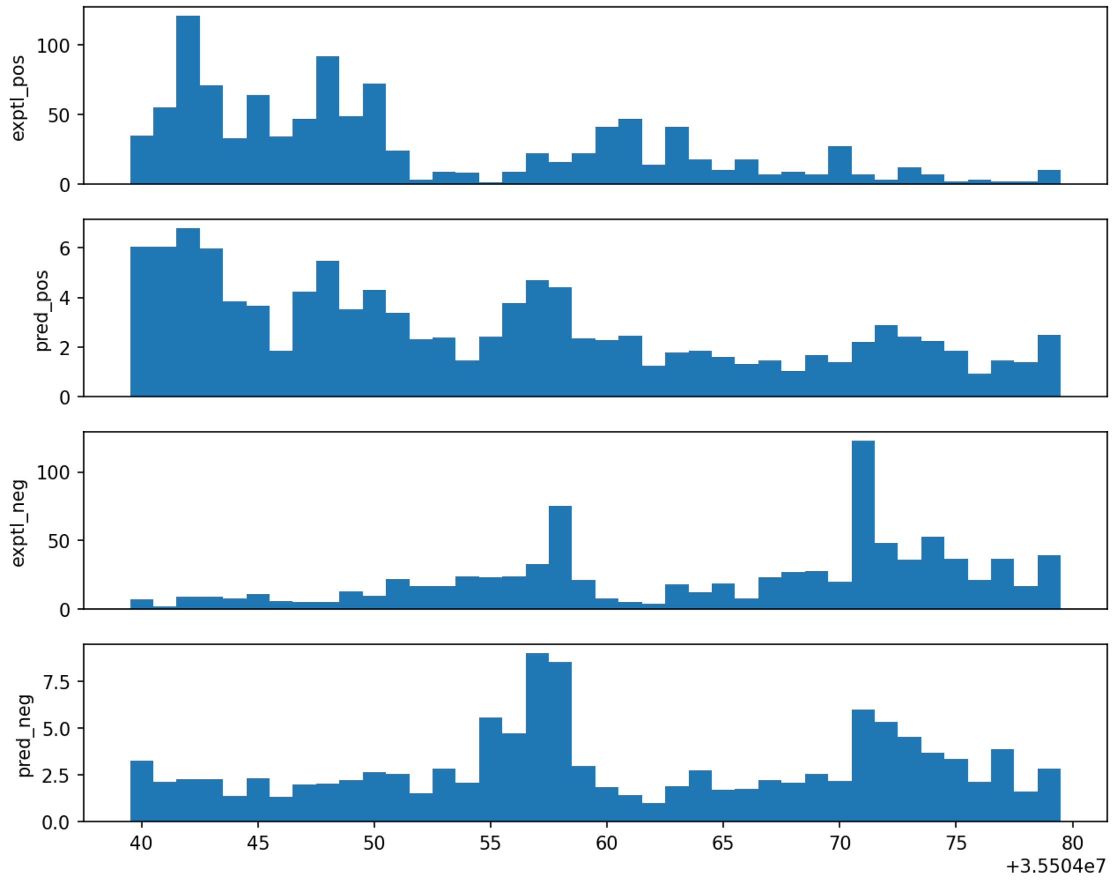
[88]:
```
'/bigscratch/bpreveal/oskn/slurm/predToBigwigCombined.zsh'
```

[ ]:

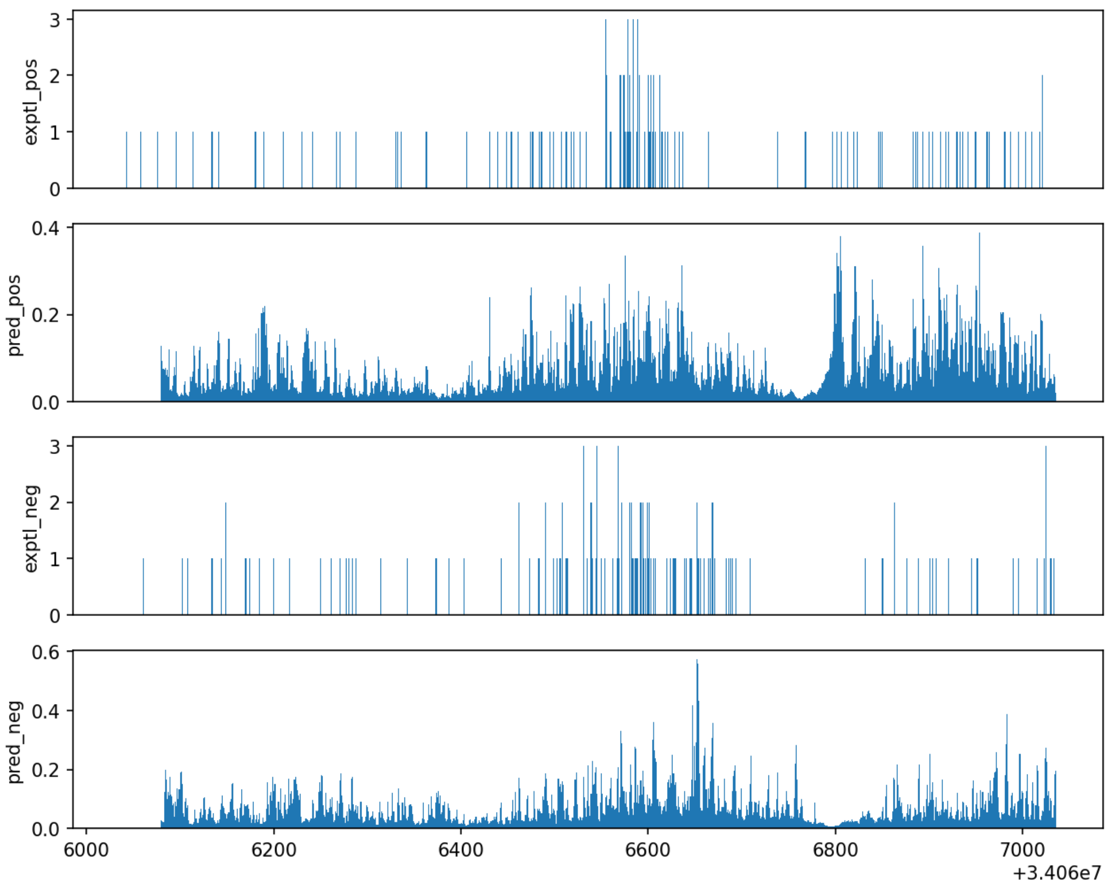
[ ]:

[54]:
```
def plotTfBigwigs(tfName, exptName, startPos = 34066036, span=1000,␣
 ↪chrom="chr1"):
    plotBws([DATA_DIRECTORY + "/" + tfName + "/counts.pos.bw",
             WORKING_DIRECTORY + "/pred/" + tfName + "_" + exptName +␣
 ↪"_positive.bw",
             DATA_DIRECTORY + "/" + tfName + "/counts.neg.bw",
             WORKING_DIRECTORY + "/pred/" + tfName + "_" + exptName +␣
 ↪"_negative.bw"],
            ["exptl_pos", "pred_pos", "exptl_neg", "pred_neg"], chrom,␣
 ↪startPos, startPos+span)
```

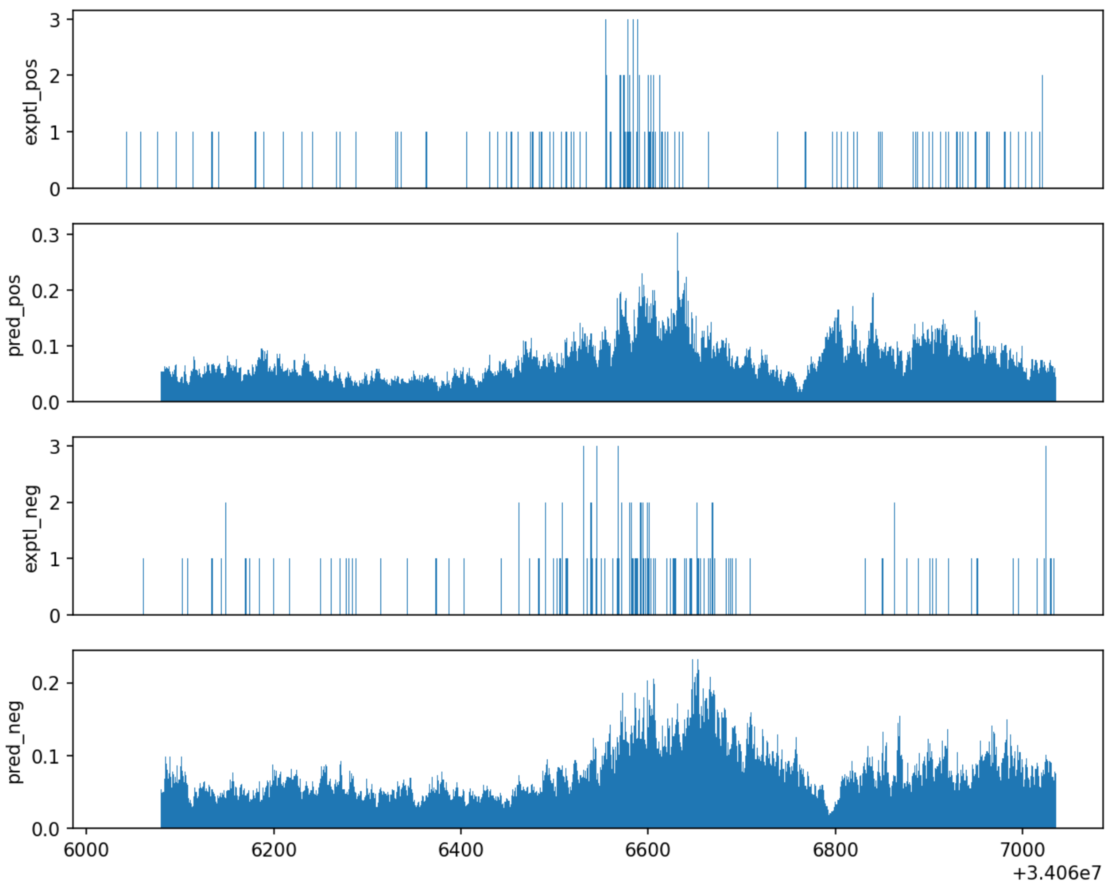
```
[55]: plotTfBigwigs('oct4', 'residual', startPos = 35504040, span=40, chrom="chr17")
```



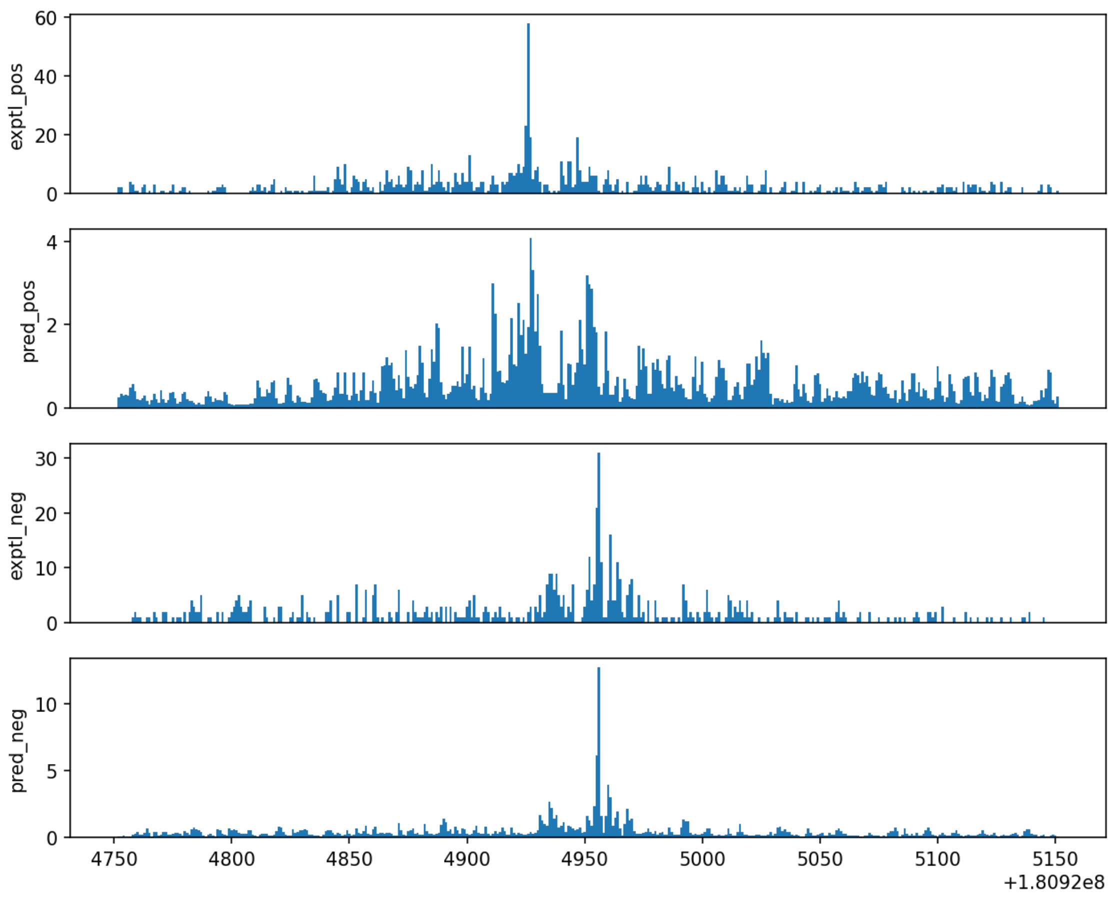
```
[56]: plotTfBigwigs('oct4', 'combined')
```



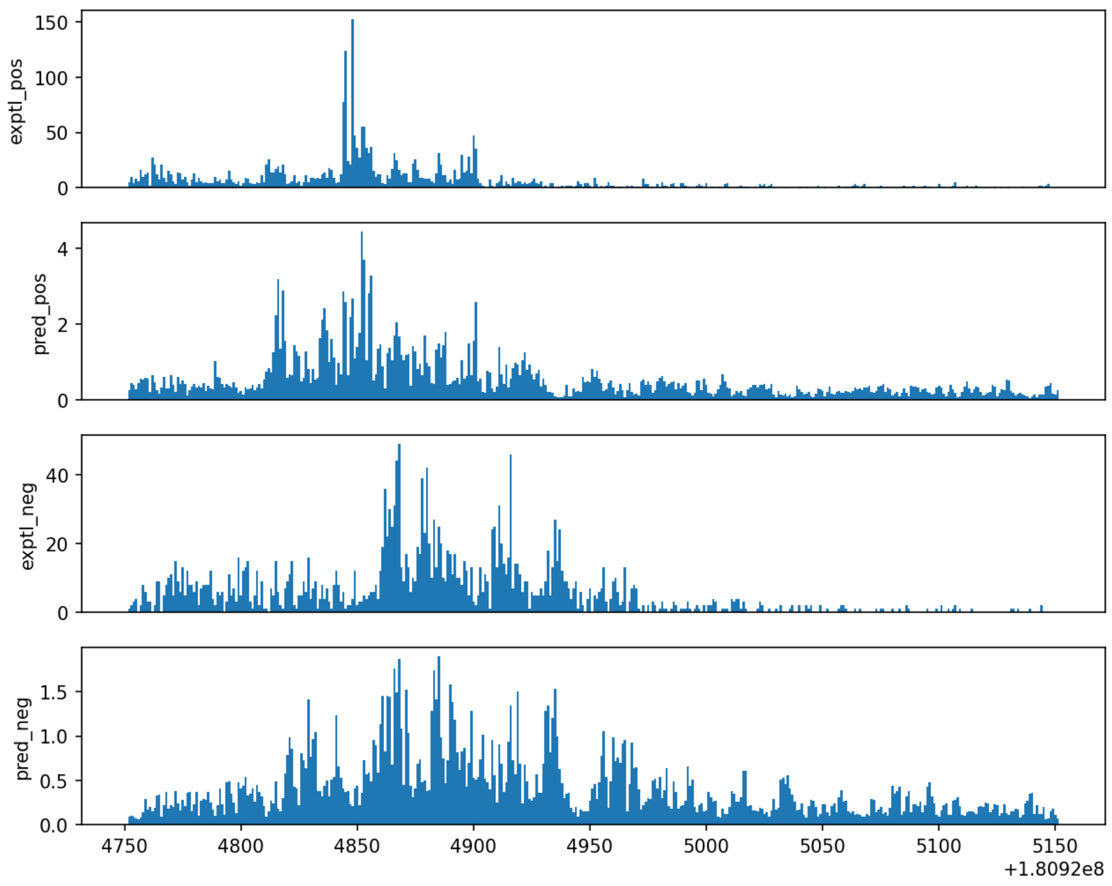
```
[57]: plotTfBigwigs('oct4', 'residual')
```



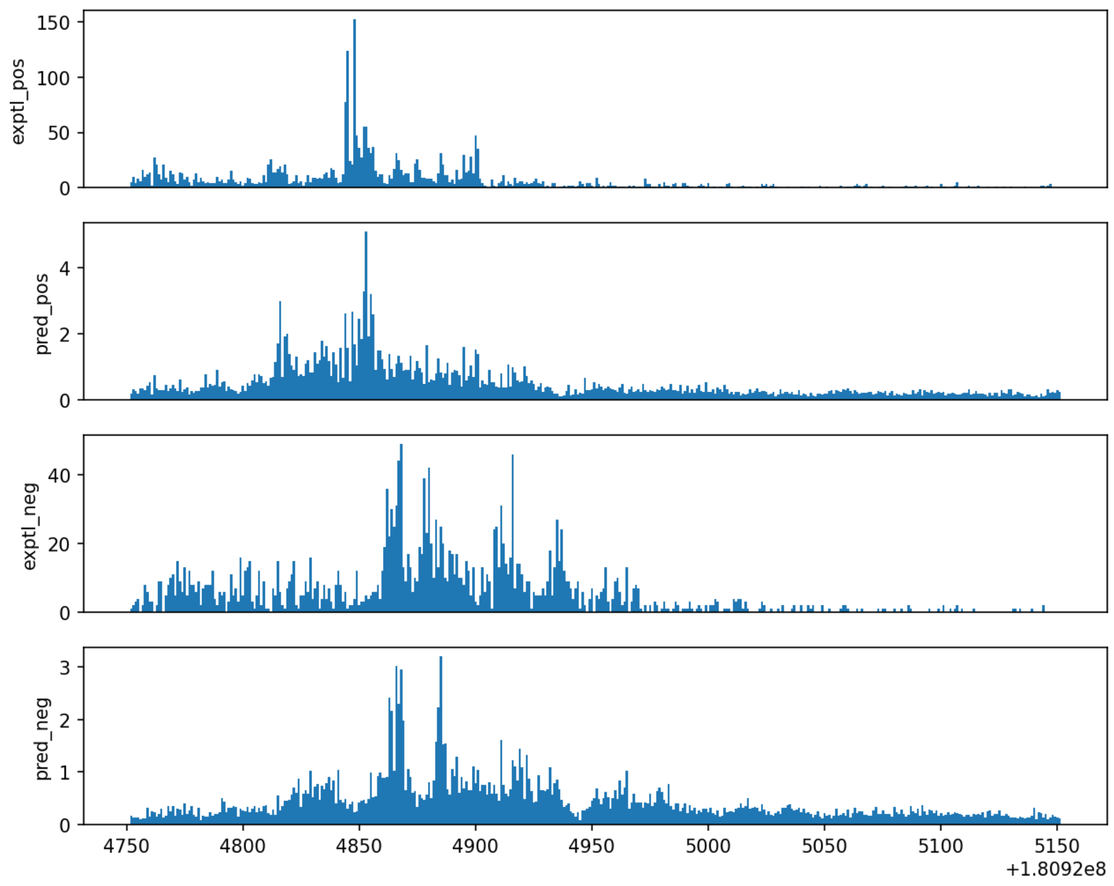
```
[58]: #Let's look around our favorite Lefty1 enhancer...
      plotTfBigwigs('oct4', 'combined', startPos = 180924752, span=400)
```



```
[59]: plotTfBigwigs('nanog', 'combined', startPos = 180924752, span=400)
```



```
[60]: plotTfBigwigs('nanog', 'residual', startPos = 180924752, span=400)
```



[61]:
```
!metrics --reference {DATA_DIRECTORY}/oct4/counts.pos.bw \
         --pred {WORKING_DIRECTORY}/pred/oct4_combined_positive.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_train.bed \
         --threads 20 --apply-abs --skip-zeroes

!metrics --reference {DATA_DIRECTORY}/oct4/counts.pos.bw \
         --pred {WORKING_DIRECTORY}/pred/oct4_combined_positive.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_val.bed \
         --threads 20 --apply-abs --skip-zeroes
```

```
reference /bigscratch/oskn/oct4/counts.pos.bw predicted
/bigscratch/bpreveal/oskn/pred/oct4_combined_positive.bw regions
/bigscratch/bpreveal/oskn/bed/peak_train.bed
metric                0.000000%       25.000000%       50.000000%       75.000000%
100.000000% regions
mnll               -4863.968228      -425.038306      -332.418318      -262.705183
-5.776033 71371
jsd                    0.287421         0.661698         0.698218         0.725974
0.830603 71371
pearsonr              -0.115368         0.157548         0.202830         0.265696
```

```
0.899057 71371
spearmanr             -0.188634        0.157078        0.194144        0.243795
0.786776 71371
Counts pearson    0.508207
Counts spearman   0.615764
reference /bigscratch/oskn/oct4/counts.pos.bw predicted
/bigscratch/bpreveal/oskn/pred/oct4_combined_positive.bw regions
/bigscratch/bpreveal/oskn/bed/peak_val.bed
metric                0.000000%      25.000000%      50.000000%      75.000000%
100.000000% regions
mnll               -4870.122599     -428.913687     -333.733298     -264.609612
-23.809449 23032
jsd                    0.309911        0.660102        0.697738        0.725359
0.831215 23032
pearsonr              -0.095459        0.158507        0.203768        0.267762
0.828873 23032
spearmanr             -0.064361        0.157711        0.194711        0.244532
0.739891 23032
Counts pearson    0.392344
Counts spearman   0.544193
```

[62]:
```
!metrics --reference {DATA_DIRECTORY}/oct4/counts.pos.bw \
         --pred {DATA_DIRECTORY}/bigwigs/Oct4.preds.pos.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_train.bed \
         --threads 20 --apply-abs --skip-zeroes
!metrics --reference {DATA_DIRECTORY}/oct4/counts.pos.bw \
         --pred {DATA_DIRECTORY}/bigwigs/Oct4.preds.pos.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_val.bed \
         --threads 20 --apply-abs --skip-zeroes
```

```
reference /bigscratch/oskn/oct4/counts.pos.bw predicted
/bigscratch/oskn/bigwigs/Oct4.preds.pos.bw regions
/bigscratch/bpreveal/oskn/bed/peak_train.bed
metric                0.000000%      25.000000%      50.000000%      75.000000%
100.000000% regions
mnll               -4035.381696     -472.540420     -371.122058     -298.897697
-44.751772 32883
jsd                    0.232497        0.637601        0.679668        0.710612
0.832555 51843
pearsonr              -0.145809        0.176634        0.233113        0.320228
0.931734 51843
spearmanr             -0.254353        0.167713        0.213578        0.276336
0.813174 51843
Counts pearson    0.442374
Counts spearman   0.469483
reference /bigscratch/oskn/oct4/counts.pos.bw predicted
/bigscratch/oskn/bigwigs/Oct4.preds.pos.bw regions
/bigscratch/bpreveal/oskn/bed/peak_val.bed
```

```
metric                  0.000000%      25.000000%      50.000000%      75.000000%
100.000000% regions
mnll                 -4403.891311     -482.707397     -377.738172     -302.637765
-34.572364 10742
jsd                     0.289047        0.635485        0.678711        0.709827
0.832555 16698
pearsonr               -0.148314        0.176564        0.234603        0.319725
0.893884 16698
spearmanr              -0.212549        0.168115        0.215135        0.276653
0.764641 16698
Counts pearson     0.379145
Counts spearman    0.428746
```

[63]:
```
!metrics --reference {WORKING_DIRECTORY}/pred/oct4_combined_positive.bw \
         --pred {DATA_DIRECTORY}/bigwigs/Oct4.preds.pos.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_train.bed \
         --threads 20 --apply-abs --skip-zeroes
!metrics --reference {WORKING_DIRECTORY}/pred/oct4_combined_positive.bw \
         --pred {DATA_DIRECTORY}/bigwigs/Oct4.preds.pos.bw \
         --regions {WORKING_DIRECTORY}/bed/peak_val.bed \
         --threads 20 --apply-abs --skip-zeroes
```

```
reference /bigscratch/bpreveal/oskn/pred/oct4_combined_positive.bw predicted
/bigscratch/oskn/bigwigs/Oct4.preds.pos.bw regions
/bigscratch/bpreveal/oskn/bed/peak_train.bed
metric                  0.000000%      25.000000%      50.000000%      75.000000%
100.000000% regions
mnll                    0.000000        0.000000        0.000000        0.000000
0.000000 5
jsd                     0.076613        0.130853        0.155033        0.191316
0.831779 51843
pearsonr               -0.322994        0.750164        0.827035        0.875522
0.961081 51843
spearmanr              -0.683330        0.793708        0.861657        0.903284
0.979249 51843
Counts pearson     0.633014
Counts spearman    0.489929
reference /bigscratch/bpreveal/oskn/pred/oct4_combined_positive.bw predicted
/bigscratch/oskn/bigwigs/Oct4.preds.pos.bw regions
/bigscratch/bpreveal/oskn/bed/peak_val.bed
metric                  0.000000%      25.000000%      50.000000%      75.000000%
100.000000% regions
mnll                    0.000000        0.000000        0.000000        0.000000
0.000000 5
jsd                     0.080074        0.131596        0.156246        0.192478
0.831647 16698
pearsonr               -0.329339        0.747608        0.825062        0.873779
0.956333 16698
```

```
spearmanr              -0.614163        0.793789        0.860093        0.902568
0.980839 16698
Counts pearson    0.315478
Counts spearman   0.496745
```

7 Deriving flat importance scores

[69]:
```
#Importance scores are needed to run motif discovery, but they're also a great␣
↪way to analyze what the model learned.
#Unfortunately for us, they take a while to generate.
jobSpecs = []
def makeInterpretJson(tfNum):
    return {
        "genome" : GENOME_FASTA,
        "bed-file" : WORKING_DIRECTORY + "/bed/peak_all.bed",
        "model-file" : WORKING_DIRECTORY + "/models/joint_residual.model",
        "input-length" : INPUT_LENGTH,
        "output-length" : OUTPUT_LENGTH,
        "heads" : 4,
        "head-id": tfNum,
        "profile-task-ids" : [0,1],
        "profile-h5" : WORKING_DIRECTORY + "/shap/" + TF_NAMES[tfNum] +␣
↪"_profile.h5",
        "counts-h5" : WORKING_DIRECTORY + "/shap/" + TF_NAMES[tfNum] + "_counts.
↪h5",
        "num-shuffles" : 20,
        "verbosity" : "DEBUG"}
cmds = []
for tfNum in range(len(TF_NAMES)):
    fname = WORKING_DIRECTORY + "/json/shap_" + TF_NAMES[tfNum] + ".json"
    cmds.append("interpretFlat {0:s}".format(fname))
    with open(fname, "w") as fp:
        json.dump(makeInterpretJson(tfNum), fp)
SLURM_CMD_GPU(SLURM_CONFIG, cmds, "interpretFlat")
```

[69]:
```
'/bigscratch/bpreveal/oskn/slurm/interpretFlat.zsh'
```

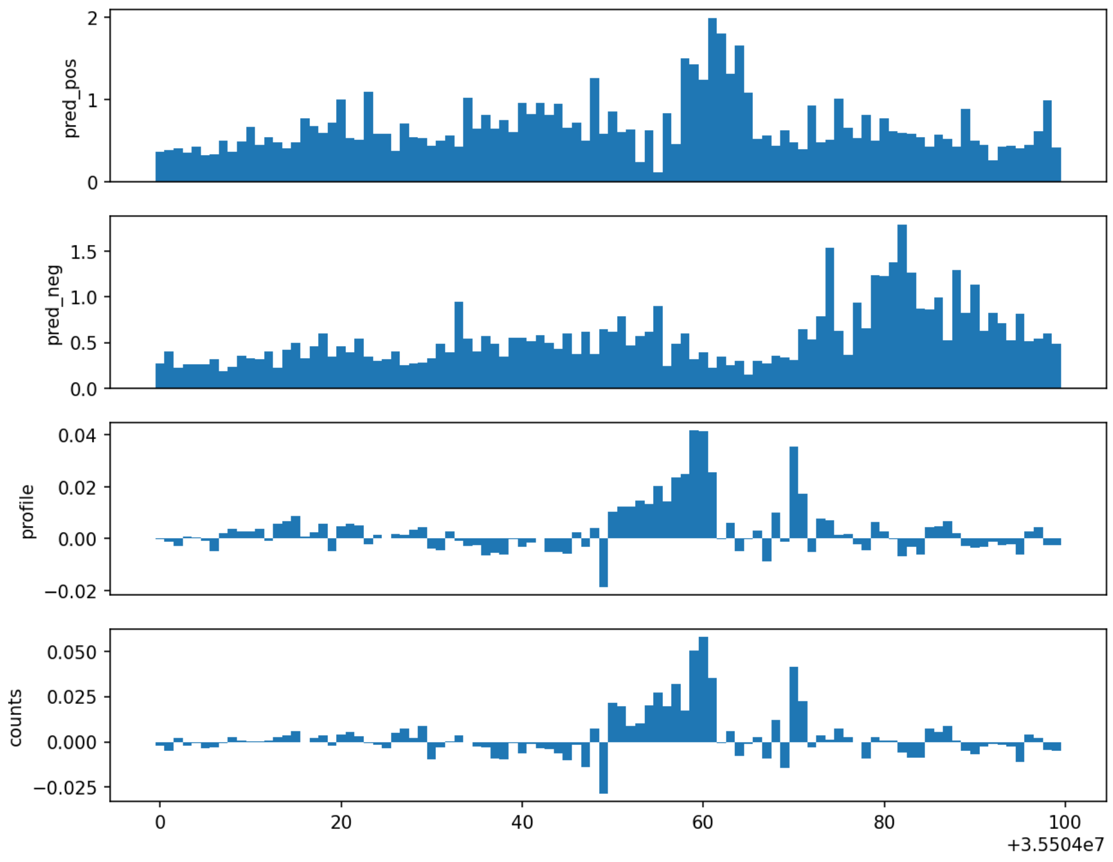
[73]:
```
shapBwCmdBase = "shapToBigwig " +\
          "--h5 {wd:s}/shap/{tf:s}_{readout:s}.h5 " +\
          "--bw {wd:s}/shap/{tf:s}_{readout:s}.bw "+\
          "--verbose"
shapBwCmds = []
for tfname in TF_NAMES:
    for readout in ["profile", "counts"]:
        cmd = shapBwCmdBase.format(wd=WORKING_DIRECTORY,
                                   tf=tfname,
                                   readout=readout)
        shapBwCmds.append(cmd)

SLURM_CMD_CPU(SLURM_CONFIG, shapBwCmds,
           "shapToBigwig", 5, 10, parallel=True)
```

[73]: '/bigscratch/bpreveal/oskn/slurm/shapToBigwig.zsh'

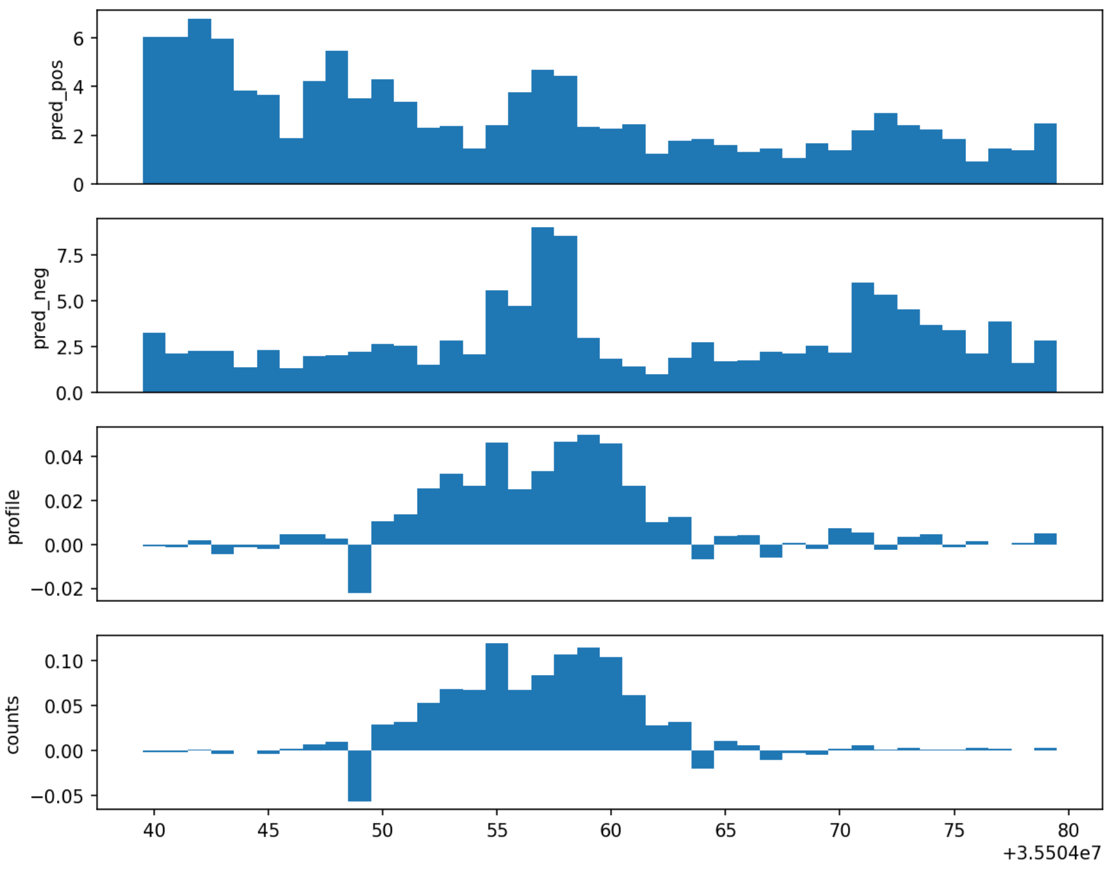
[ ]:

```
[74]: def plotShapBigwigs(tfName, exptName, startPos = 34066036, span=1000,␣
  ↪chrom="chr1"):
    plotBws([WORKING_DIRECTORY + "/pred/" + tfName + "_" + exptName +␣
  ↪"_positive.bw",
             WORKING_DIRECTORY + "/pred/" + tfName + "_" + exptName +␣
  ↪"_negative.bw",
             WORKING_DIRECTORY + "/shap/" + tfName + "_profile.bw",
             WORKING_DIRECTORY + "/shap/" + tfName + "_counts.bw"],
            ["pred_pos", "pred_neg", "profile", "counts"], chrom, startPos,␣
  ↪startPos+span)
```

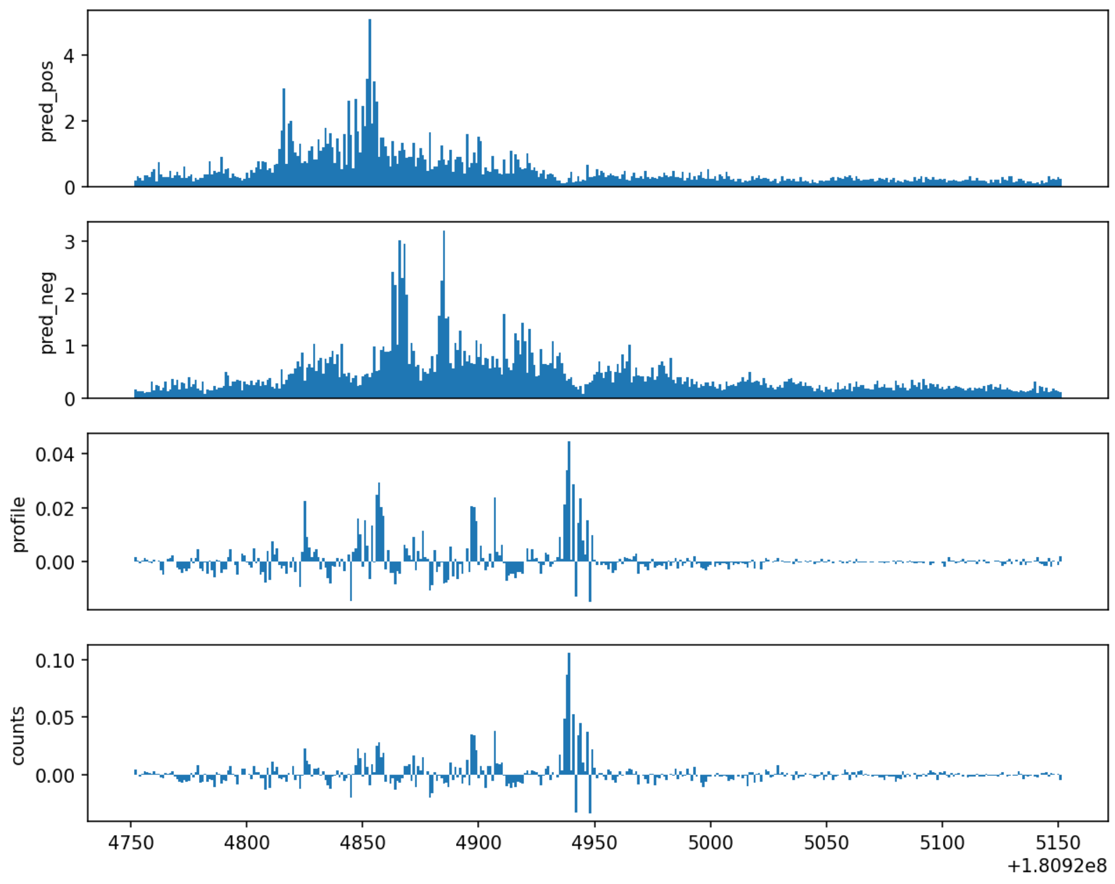
```
[89]: plotShapBigwigs("nanog", "residual", startPos = 35504000, span=100,␣
  ↪chrom="chr17")
```



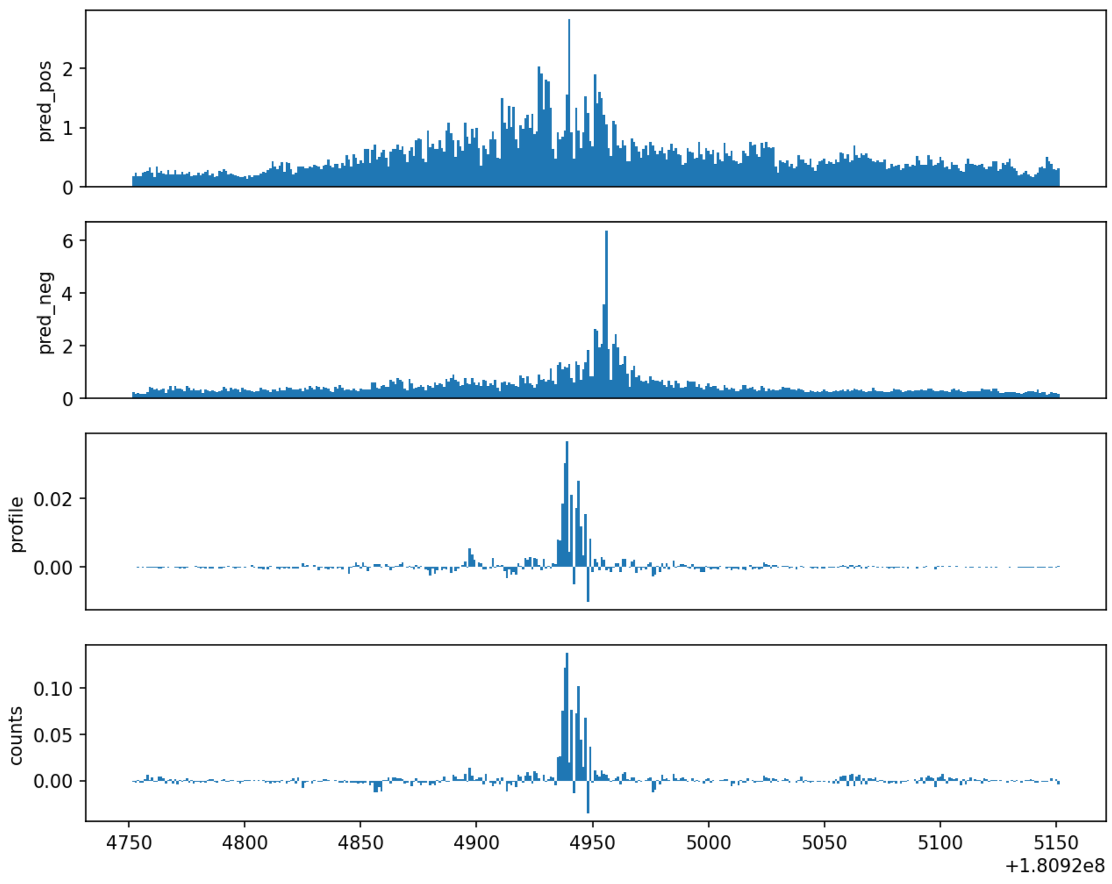
```
[90]: plotShapBigwigs('oct4', 'residual', startPos = 35504040, span=40, chrom="chr17")
```



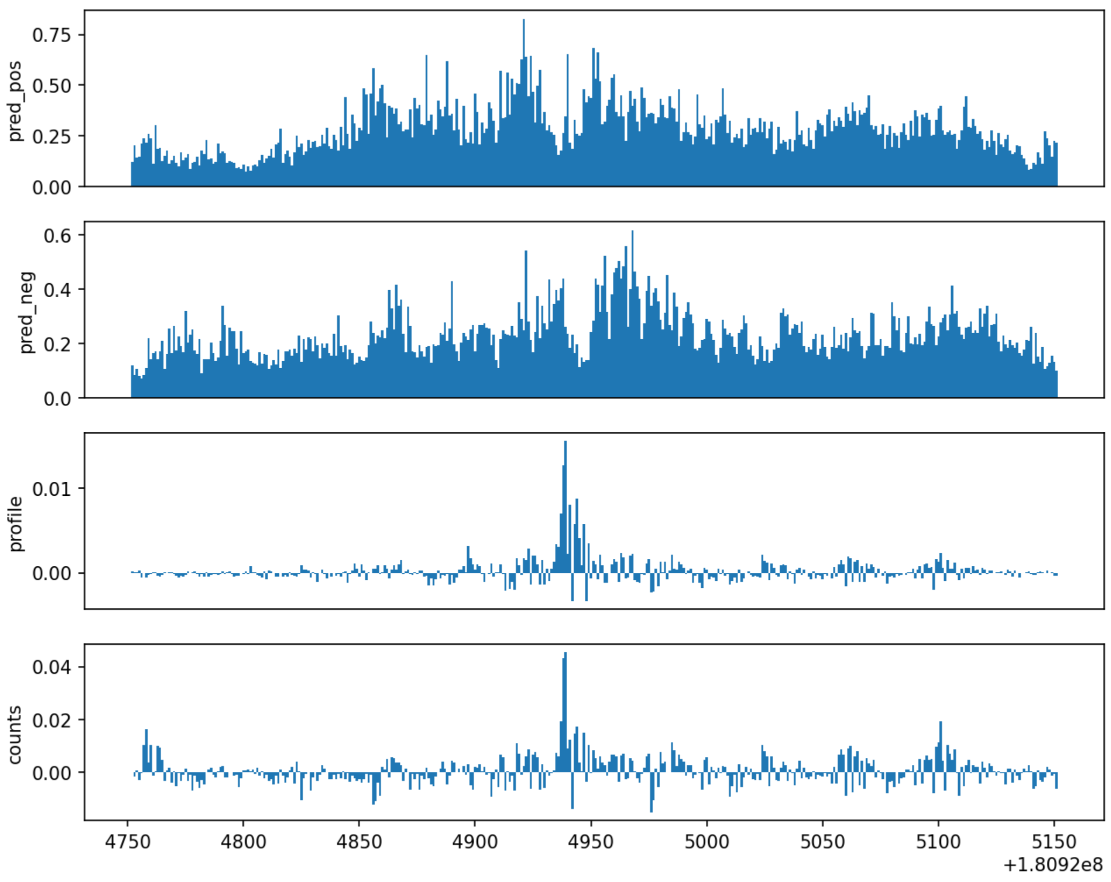
```
[91]: plotShapBigwigs('nanog', 'residual', startPos = 180924752, span=400)
```



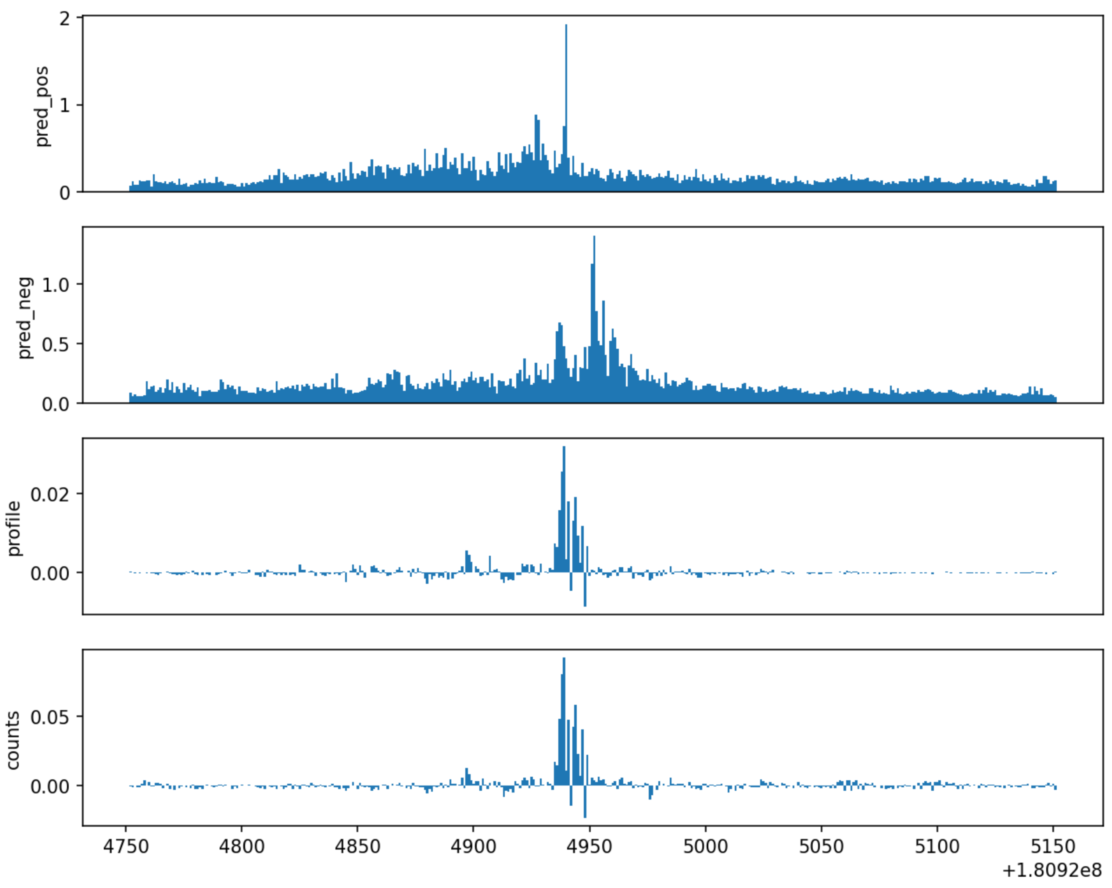
```
[92]: plotShapBigwigs('oct4', 'residual', startPos = 180924752, span=400)
```



```
[93]: plotShapBigwigs('klf4', 'residual', startPos = 180924752, span=400)
```



```
[94]: plotShapBigwigs('sox2', 'residual', startPos = 180924752, span=400)
```



[ ]:

```

```

[ ]:

```

```

[76]:

```
#Great, so we have those bigwigs and the importance hdf5. I can run Modisco now!
↪
#The first thing to do is to export the shap scores as numpy arrays, then I run␣
↪Modisco proper,
#and finally I generate reports.

shapToNumpyCmdBase = "shapToNumpy " +\
          "--h5 {wd:s}/shap/{tf:s}_{readout:s}.h5 " +\
          "--seqs {wd:s}/shap/seqs_{tf:s}_{readout:s}.npz "+\
          "--scores {wd:s}/shap/scores_{tf:s}_{readout:s}.npz "+\
          "--verbose"
shapToNumpyCmds = []
for tfname in TF_NAMES:
    for readout in ["profile", "counts"]:
        cmd = shapToNumpyCmdBase.format(wd=WORKING_DIRECTORY,
                                                tf=tfname,
                                                readout=readout)
        shapToNumpyCmds.append(cmd)

SLURM_CMD_CPU(SLURM_CONFIG, shapToNumpyCmds,
           "shapToNumpy", 1, 3, parallel=True)
```

[76]: '/bigscratch/bpreveal/oskn/slurm/shapToNumpy.zsh'

[77]:
```
modiscoCmdBase = "mkdir -p {wd:s}/modisco/{tf:s}_{readout:s}\n" +\
          "modisco motifs " +\
              "-s {wd:s}/shap/seqs_{tf:s}_{readout:s}.npz " +\
              "-a {wd:s}/shap/scores_{tf:s}_{readout:s}.npz "+\
              "-n 100000 " +\
              "-w 1000 "+\
              "-v " +\
              "-o {wd:s}/modisco/{tf:s}_{readout:s}/modisco.h5 "
modiscoCmds = []
for tfname in TF_NAMES:
    for readout in ["profile", "counts"]:
        cmd = modiscoCmdBase.format(wd=WORKING_DIRECTORY,
                                    tf=tfname,
                                    readout=readout)
        modiscoCmds.append(cmd)

SLURM_CMD_CPU(SLURM_CONFIG, modiscoCmds,
           "modisco", 80, 600)
```

[77]: '/bigscratch/bpreveal/oskn/slurm/modisco.zsh'

[98]:
```
reportCmdBase = "modisco report " +\
              "-i {wd:s}/modisco/{tf:s}_{readout:s}/modisco.h5 " +\
              "-o {wd:s}/modisco/{tf:s}_{readout:s}/ "+\
              "-n 2 " +\
              "-m /n/data1/JASPAR/2022/
 ↪JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt " +\
              "\n\n{sd:s}/annotateModiscoHtml " +\
              " {wd:s}/modisco/{tf:s}_{readout:s}/motifs.html " + \
              " > {wd:s}/modisco/{tf:s}_{readout:s}/motifs_names.html"

reportCmds = []
for tfname in TF_NAMES:
    for readout in ["profile", "counts"]:
        cmd = reportCmdBase.format(wd=WORKING_DIRECTORY,
                                   sd=SCRIPTS_DIR,
                                   tf=tfname,
                                   readout=readout)
        reportCmds.append(cmd)

SLURM_CMD_CPU(SLURM_CONFIG, reportCmds,
           "modiscoReport", 2, 5)
```

[98]:
```
'/bigscratch/bpreveal/oskn/slurm/modiscoReport.zsh'
```

[ ]:
```

```

In order to do motif scanning, I need to know what patterns Modisco has identified. I've looked at the report files and annotated the motifs that looked most promising to me.

[99]:
```
bgProbs = [(1-0.42) /2, 0.21, 0.21, (1-0.42) /2]
patternsToScan = {
    "klf4_counts" : {
        "pos" : [
            [0, "klf4"],
            [1, "esrrb"],
            [2, "octsox"],
            [3, "sox2"]]},
    "klf4_profile" : {
        "pos" : [
            [0, "klf4"],
            [1, "octsox"],
            [3, "esrrb"]]},
    "nanog_counts" : {
        "pos" : [
            [0, "nanog"],
            [1, "octsox"],
            [2, "sox2"],
            [3, 'nanog?'],
            [5, "esrrb"]]},
    "nanog_profile" : {
        "pos" : [
            [0, "nanog"],
            [1, "sox2"],
            [2, "octsox"],
            [4, "nanog2"],
            [5, "nanog2"]]},
    "oct4_counts" : {
        "pos" : [
            [0, "octsox"],
            [1, "sox2"],
            [2, "klf4"],
            [4, "esrrb"]]},
    "oct4_profile" : {
        "pos" : [
            [0, "klf4"],
            [1, "sox2"],
            [2, "octsox"],
            [4, "nanog"]]},
    "sox2_counts" : {
        "pos" : [
            [0, "sox2"],
            [1, "octsox"],
            [2, "octsox"],
            [3, "klf4"],
            [5, "esrrb"]]},
    "sox2_profile" : {
        "pos" : [
            [0, "sox2"],
            [1, "octsox"],
            [2, "klf4"],
            [3, "nanog"]]}}
```

[100]:
```
import bpreveal.tools.plots as bprplots
import bpreveal.motifUtils as motifUtils
import tqdm
```

[101]:
```
#I'll generate all of those figures and save them.
for runName, run in patternsToScan.items():
    for clusterName, cluster in run.items():
        for motif in cluster:
            pat = motifUtils.Pattern(clusterName + "_patterns", "pattern_{0:d}".
 ↪format(motif[0]), motif[1])
            with h5py.File(WORKING_DIRECTORY + "/modisco/" + runName + "/
 ↪modisco.h5", "r") as fp:
                pat.loadCwm(fp, 0.3, 3, bgProbs)
                pat.loadSeqlets(fp)
            fig = plt.figure()
            bprplots.plotModiscoPattern(pat, fig, sortKey = pat.
 ↪seqletContribMatches)
            fig.savefig(WORKING_DIRECTORY + "/modisco/" + runName + "/" +␣
 ↪clusterName + "_" + motif[1] + ".png")
            plt.close(fig)
```

[102]:
```
cmds = []
SCAN_BASE = "motifSeqletCutoffs {cutoffFname:s} \n" +\
            "motifScan {scanFname:s} \n    " +\
            "motifAddQuantiles --seqlet-tsv {seqletTsv:s} --scan-tsv {scanTsv:
 ↪s}\n" +\
            "   cat {scanTsv:s} | " + \
                "cut -f 1-6 | " + \
                "tail -n +2 | " + \
                "sort -k1,1 -k2,2n -k3,3n -k4,4 -k5,5nr | "+\
                "awk '!_[$1,$2,$3,$4,$6]++' > {scanBed:s}"
#This is a four-part command.
#First, we analyze the modisco results to extract pssms and cwms for the motifs␣
 ↪of interest.
#Then, we run the actual scan step.
#next, we use the motifAddQuantiles.py script to add quantile information,␣
 ↪which can be useful for
#analyzing the motifs.
#The last part, starting with `cat {scanTsv:s}` is scary, but it's just␣
 ↪extracting a bed file from
#the generated tsv files. The sort and awk lines are just there to remove␣
 ↪duplicate maps, where
#a motif is mapped to the same region several times. If you want all of the␣
 ↪called motifs,
#or don't want to deal with awk, you can remove the sort and awk parts of that␣
 ↪command.

for pat in patternsToScan.keys():
    curPats = patternsToScan[pat]
    patternSpec = []
    for mcName in curPats.keys():
        patternSpec.append({
            "metacluster-name" : mcName + "_patterns",
            "pattern-names" : ["pattern_{0:d}".format(x[0]) for x in␣
 ↪curPats[mcName]],
            "short-names" : [x[1] for x in curPats[mcName]]})
    seqletTsv =  WORKING_DIRECTORY + "/modisco/" + pat + "/seqlets_" + pat + ".
 ↪tsv"
    hitsTsv = WORKING_DIRECTORY + "/scan/" + pat + ".tsv"
    hitsBed = WORKING_DIRECTORY + "/scan/" + pat + ".bed"
    cutoffConfigDict = {
            "seqlets-tsv" : seqletTsv,
            "modisco-h5" : WORKING_DIRECTORY + "/modisco/" + pat + "/modisco.
 ↪h5",
            "modisco-contrib-h5" : WORKING_DIRECTORY + "/shap/" + pat + ".h5",
            "patterns" : patternSpec,
            "seq-match-quantile" : 0.2,
            "contrib-match-quantile" : 0.2,
            "contrib-magnitude-quantile" : 0.2,
            "trim-threshold" : 0.3,
            "trim-padding" : 1,
            "background-probs" : bgProbs,
            "quantile-json" : WORKING_DIRECTORY + "/scan/" + pat + "_motifs.
 ↪json",
            "verbosity" : "INFO"}
    scanConfigDict = {
        "scan-settings" : {
            "scan-contrib-h5" : WORKING_DIRECTORY + "/shap/" + pat + ".h5",
            "hits-tsv" : hitsTsv,
            "num-threads" : NUM_THREADS},
        "seqlet-cutoff-json" : WORKING_DIRECTORY + "/scan/" + pat + "_motifs.
 ↪json",
        "verbosity" : "INFO"}
    scanFname = WORKING_DIRECTORY + "/json/scan_" + pat + ".json"
    cutoffFname = WORKING_DIRECTORY + "/json/cutoffs_" + pat + ".json"
    cmdStr = SCAN_BASE.format(scanFname = scanFname, cutoffFname = cutoffFname,␣
 ↪seqletTsv = seqletTsv, scanTsv = hitsTsv, scanBed = hitsBed)
    cmds.append(cmdStr)
    with open(scanFname, "w") as fp:
        json.dump(scanConfigDict, fp, indent=4)
    with open(cutoffFname, "w") as fp:
        json.dump(cutoffConfigDict, fp, indent=4)
SLURM_CMD_CPU(SLURM_CONFIG, cmds, "motifScan", NUM_THREADS, 20)
```

[102]:
```
'/bigscratch/bpreveal/oskn/slurm/motifScan.zsh'
```

8 Making a PISA plot

[106]:
```
#In order to make a pisa plot, I need to get a list of regions I want to␣
 ↪analyze. The way the PISA script works is that I give it a fasta-format file
#of genomic regions, each region being INPUT_LENGTH long. The PISA tool will␣
 ↪then assign contributions to all of the bases in the input relative to the
#*leftmost* base in the output.
#This is important, so let me phrase it differently:



#|<- Receptive field ->|
#INPUTSEQUENCEINPUTSEQUENCEINPUTSEQUENCEINPUTSEQUENCE
#\                     /                            /
#  \                 /                            /
#    \             /                            /
#      \         /                            /
#        \     /                            /
#          \ /                            /
#           OUTPUTPROFILEOUTPUTPROFILEOUTP
#           ^
#           | This O is the base that will be used to calculate the␣
 ↪contribution scores.
#It's important to not have any off-by-one problems here, so let's work it out␣
 ↪manually.
```

```python
print(INPUT_LENGTH)
print(RECEPTIVE_FIELD)
print(BUFFER)
```

```
3056
2057
1028
```

[104]:
```python
#Since I don't feel like doing ascii art that's quite so wide, I'm going to say that the network is quite a bit smaller:
!lengthCalc --output-len 20 --n-dil-layers 3 --conv1-kernel-size 3 --profile-kernel-size 3
```

```
52
```

[ ]:
```python
#So in this example the receptive field would be 52-20+1=33.
#And there are 16 bases of slop on each side that need to be seen by the model.
#-30     -20       -10         0        10        20        30        40        50
#V         V         V         V         V         V         V         V         V
#098765432109876543210987654321012345678901234567890123456789012345678901234567890123456789
#Output:                       01234567890123456789
#Input:        6543210987654321012345678901234567890123456789012345
#Receptive:    654321098765432101234567890123456

#So in this case, if we want shap scores for a base at position zero, we need sequence from -16 to +35 (inclusive)
```

[ ]:
```python

```

[105]:
```python
windowStart = 180924752-1000
windowEnd = 180925152+1000
windowLen = windowEnd - windowStart
windowChrom = "chr1"
```

[107]:
```python
#So I need to get windows that are 3092 bases wide, and the first 2093 bases are the only ones that have a chance of affecting the output
#(since that's the receptive field for the first base.)
#The slop is (2093-1)/2 = 1046
#I want to shap starting at chr1:180924752 and I want to take 400 bases worth of calculations.
def writeRegion(genome, outFp, regionStart):
    genomeStart = regionStart - BUFFER
    genomeEnd = genomeStart + INPUT_LENGTH
    seq = genome.fetch(windowChrom, genomeStart, genomeEnd)
    outFp.write(">{0:d}\n".format(regionStart))
    outFp.write(seq.upper())
    outFp.write("\n")

with open(WORKING_DIRECTORY + "/shap/pisa_regions.fa", "w") as fp:
    with pysam.FastaFile(GENOME_FASTA) as genome:
        for regionStart in range(windowStart, windowEnd):
            writeRegion(genome, fp, regionStart)
```

[109]:
```
#And now we bulid the json file for the PISA analysis.
cmds = []
for tfid in [0,3]:
    for strand in [0,1]:
        task_name = TF_NAMES[tfid] + "_" + ["positive", "negative"][strand]
        pisa_config = {"model-file" : WORKING_DIRECTORY + "/models/
 ↪joint_residual.model",
                       "fasta-file" : WORKING_DIRECTORY + "/shap/pisa_regions.
 ↪fa",
                       "num-shuffles" : 20,
                       "head-id" : tfid, #(That's the nanog head)
                       "task-id" : strand,
                       "output-h5" : WORKING_DIRECTORY + "/shap/pisa_" +␣
 ↪task_name + ".h5",
                       "input-length" : INPUT_LENGTH,
                       "output-length" : OUTPUT_LENGTH,
                       "make-predictions" : True,
                       "num-batchers" : 2,
                       "verbosity" : "INFO"}
        jsonFname = WORKING_DIRECTORY + "/json/pisa_" + task_name + ".json"
        with open(jsonFname, "w") as fp:
            json.dump(pisa_config, fp)
        cmds.append("interpretPisa {0:s}".format(jsonFname))

SLURM_CMD_GPU(SLURM_CONFIG, cmds, "interpretPisa", 5, 20, "10:00:00")
```

[109]:
```
'/bigscratch/bpreveal/oskn/slurm/interpretPisa.zsh'
```

[ ]:
```

```

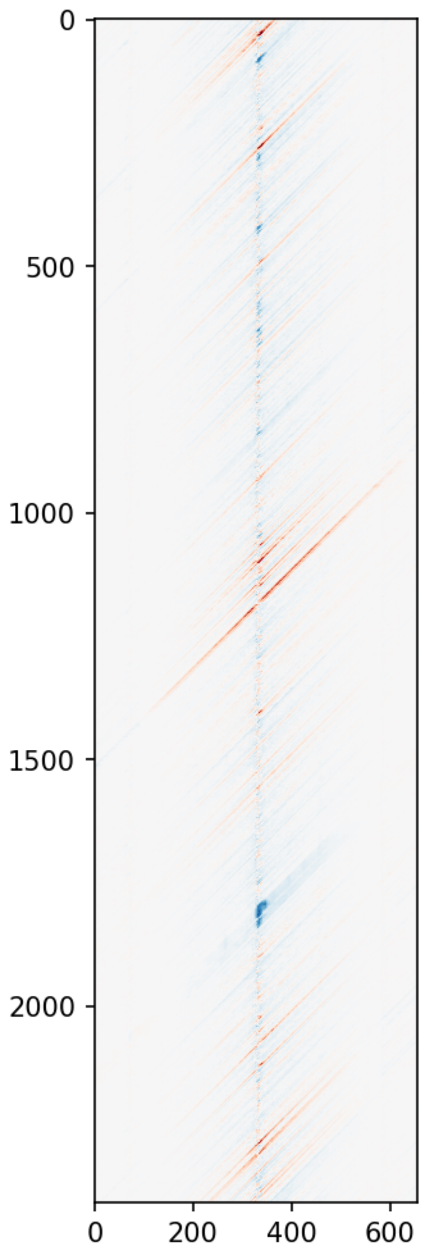
[124]:
```
#Let's take a look at the pisa results!
with h5py.File(WORKING_DIRECTORY + "/shap/pisa_nanog_positive.h5", "r") as fp:
    pisaDescriptions = list(fp["descriptions"])
    pisaSequences = np.array(fp["sequence"])
    pisaShap = np.array(fp["shap"])
    pisaInputPred = np.array(fp["input_predictions"])
    pisaInputPred = np.array(fp["shuffle_predictions"])
```

[125]:
```
pisaVals = np.sum(pisaShap,axis=2)
print(pisaVals.shape)
```

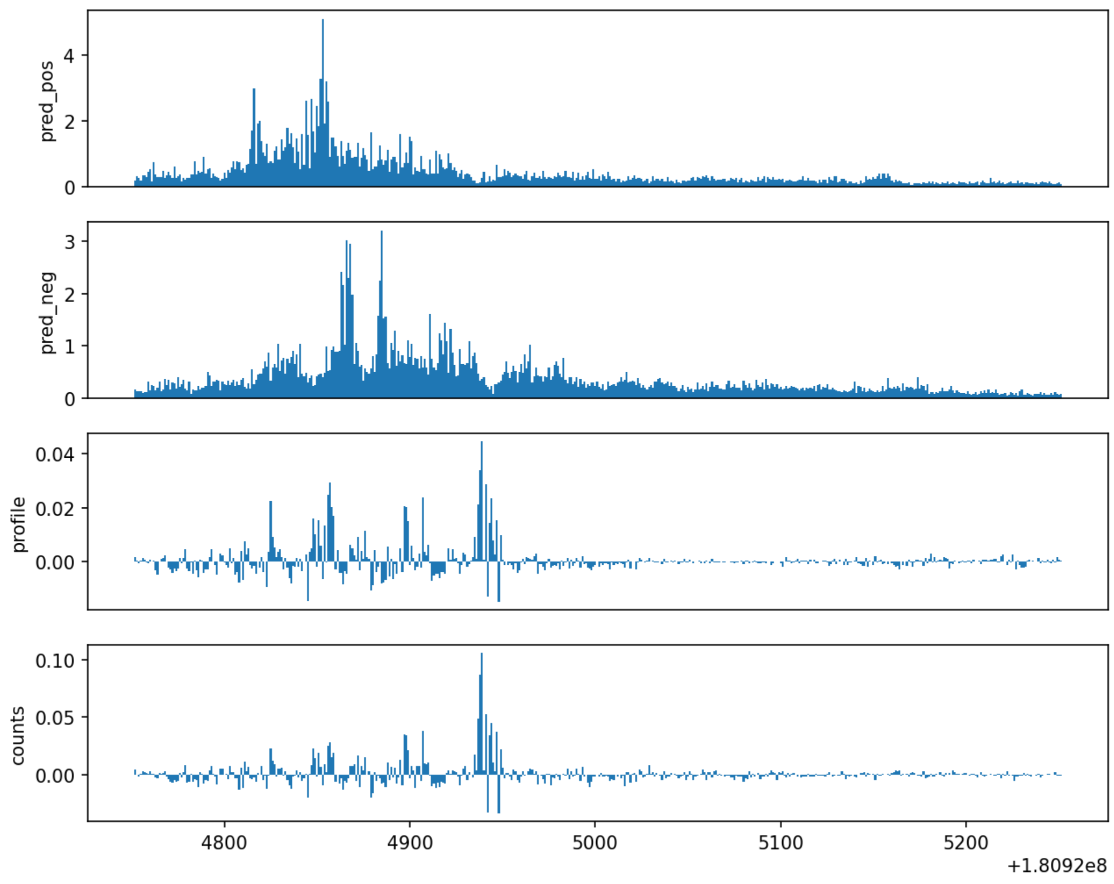
```
(2400, 2056)
```

[128]:
```
pisaSpan = 0.2
plt.imshow(pisaVals[:,700:-700], vmin=-pisaSpan, vmax=pisaSpan, cmap='RdBu_r')
```

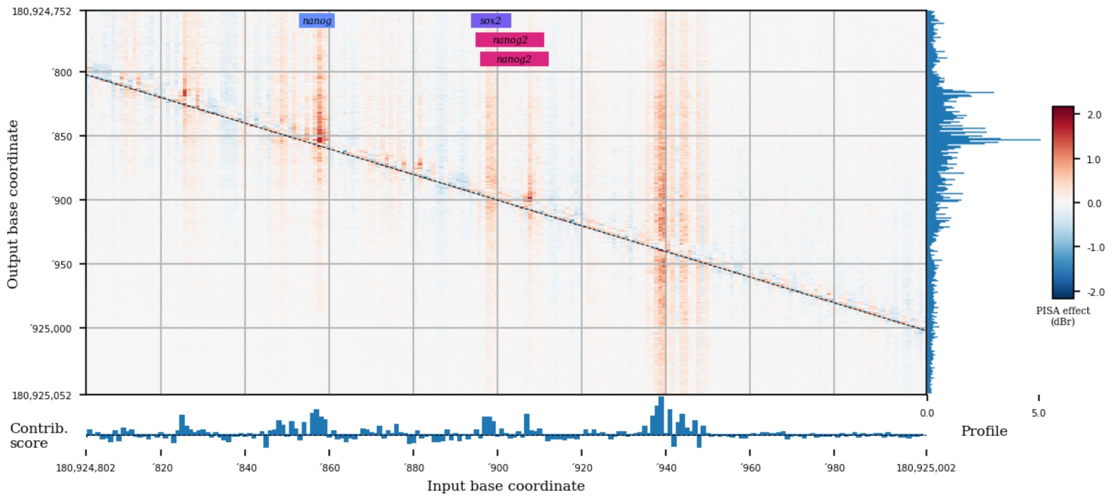
[128]:
```
<matplotlib.image.AxesImage at 0x7f50434c4090>
```

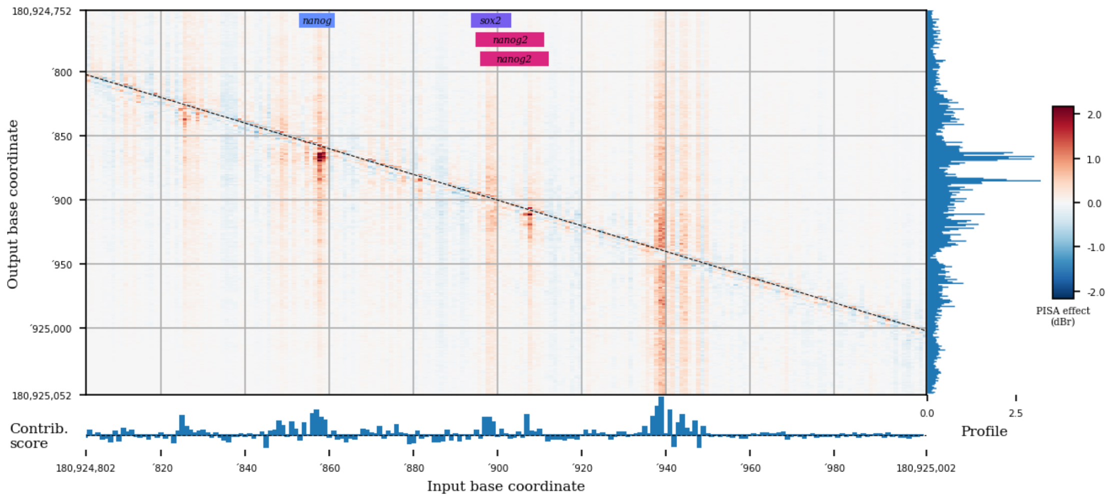


[113]:
```
#Let's remind ourselves of what the nanog binding looked like...
plotShapBigwigs('nanog', 'residual', startPos = windowStart + 1000, span=500)
```

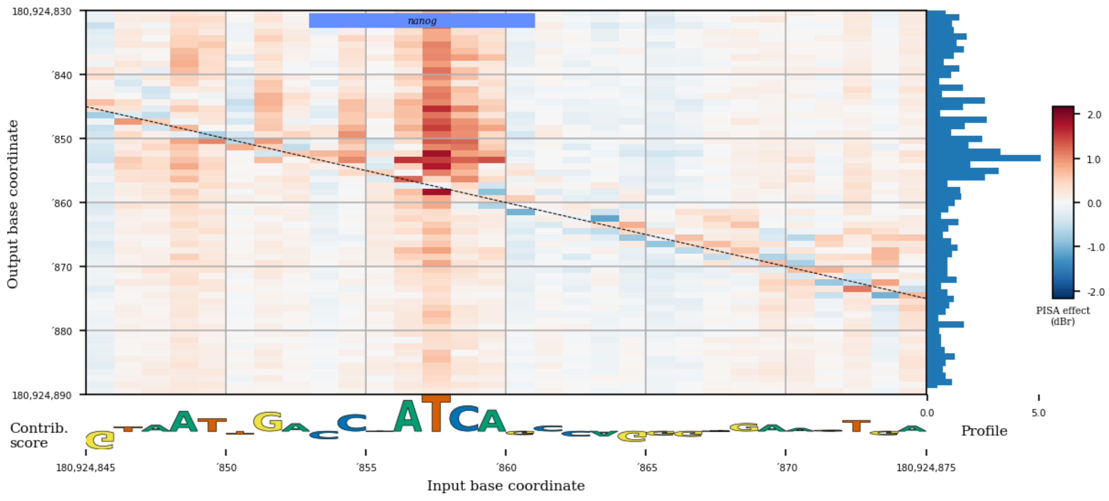


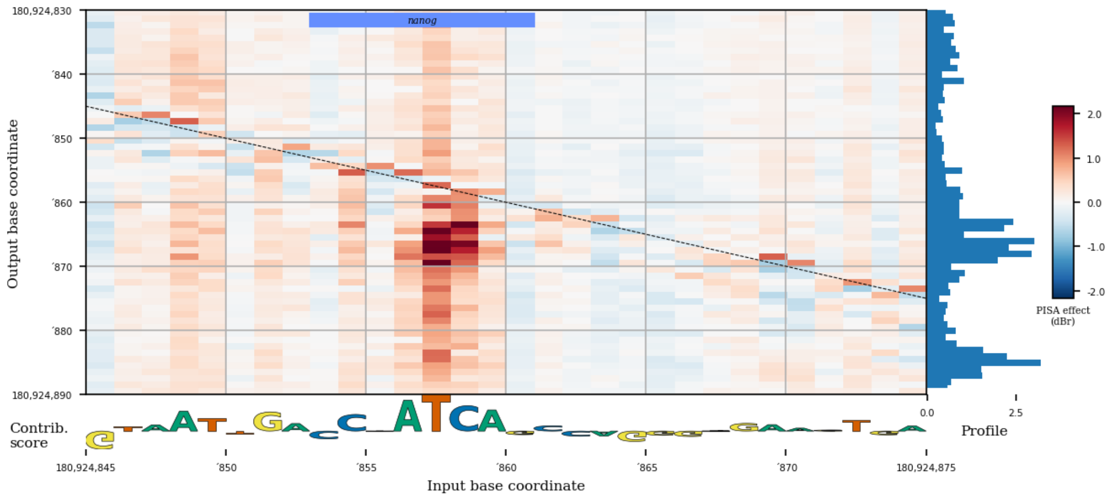
[114]:
```
nameColors = dict()
fig = plt.figure()

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_positive.h5",
                  1150, 200, 300, RECEPTIVE_FIELD,
                  windowStart, "chr1", GENOME_FASTA,
                  WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                  WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                  WORKING_DIRECTORY + "/pred/nanog_residual_positive.bw",
                  nameColors,
                  fig, [0.1, 0.55, 0.9, 0.4],
                  colorSpan = 0.5,
                  fontsize=5)

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_negative.h5",
                  1150, 200, 300, RECEPTIVE_FIELD,
                  windowStart, "chr1", GENOME_FASTA,
                  WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                  WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                  WORKING_DIRECTORY + "/pred/nanog_residual_negative.bw",
                  nameColors,
                  fig, [0.1, 0.05, 0.9, 0.4],
                  colorSpan = 0.5);
```



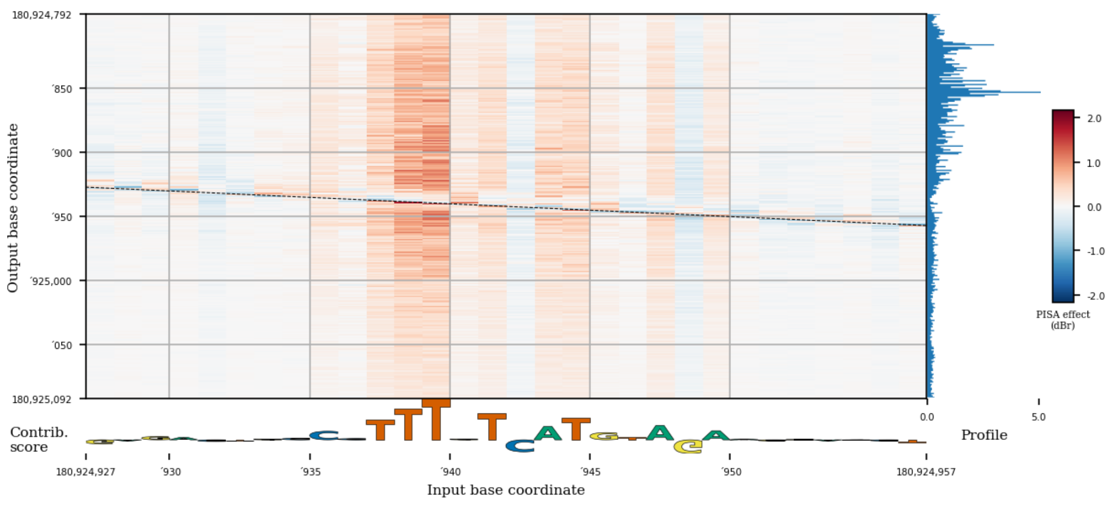


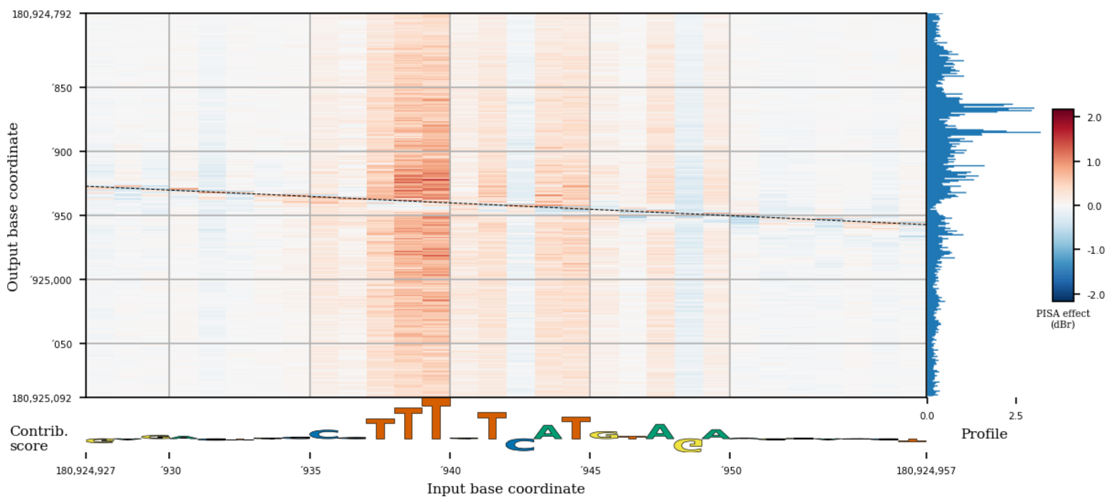
```
[115]: #One thing is pretty striking. The motif at ~100 bp has a directional effect,␣
         ↪that is, the importance toward the positive peak is upstream of the
       #motif and the importance of the negative peak is downstream. But the motif at␣
         ↪~190 doesn't seem to be directional, and it seems to have a larger
       #reach. Let's zoom in!
       fig = plt.figure()

       bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_positive.h5",
                                  1108, 30, 60, RECEPTIVE_FIELD,
                                  windowStart, "chr1", GENOME_FASTA,
                                  WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                                  WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                                  WORKING_DIRECTORY + "/pred/nanog_residual_positive.bw",
                                  nameColors,
                                  fig, [0.1, 0.55, 0.9, 0.4],
                                  colorSpan = 0.5)

       bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_negative.h5",
                                  1108, 30, 60, RECEPTIVE_FIELD,
                                  windowStart, "chr1", GENOME_FASTA,
                                  WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                                  WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                                  WORKING_DIRECTORY + "/pred/nanog_residual_negative.bw",
                                  nameColors,
                                  fig, [0.1, 0.05, 0.9, 0.4],
                                  colorSpan = 0.5);
```



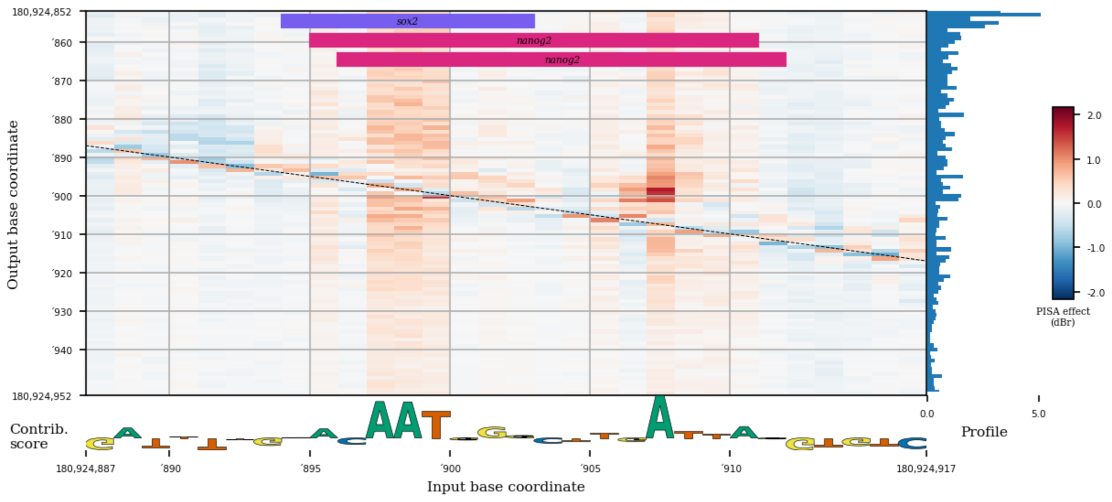


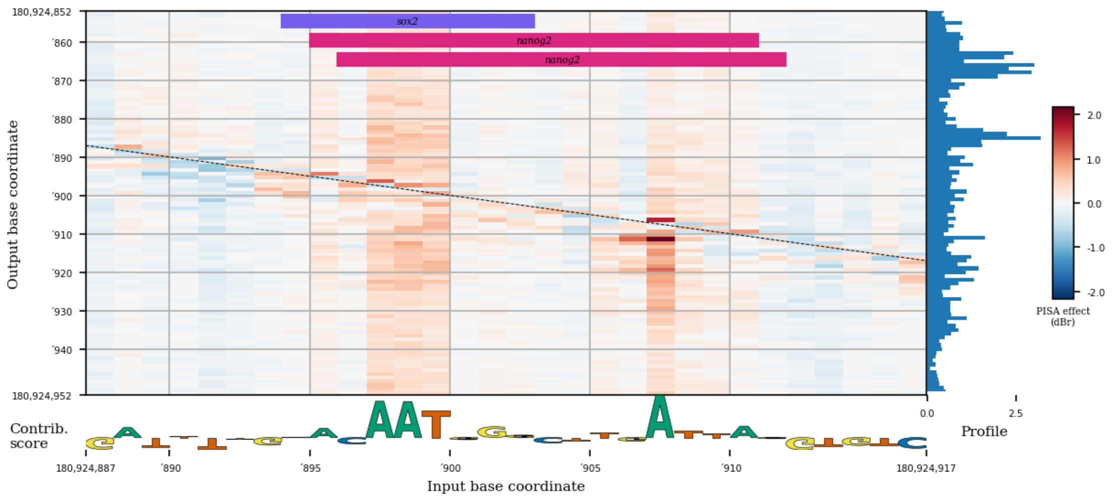
```
[116]: fig = plt.figure()

       bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_positive.h5",
                          1190, 30, 300, RECEPTIVE_FIELD,
                          windowStart, "chr1", GENOME_FASTA,
                          WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                          WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                          WORKING_DIRECTORY + "/pred/nanog_residual_positive.bw",
                          nameColors,
                          fig, [0.1, 0.55, 0.9, 0.4],
                          colorSpan = 0.5)

       bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_negative.h5",
                          1190, 30, 300, RECEPTIVE_FIELD,
                          windowStart, "chr1", GENOME_FASTA,
                          WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                          WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                          WORKING_DIRECTORY + "/pred/nanog_residual_negative.bw",
                          nameColors,
                          fig, [0.1, 0.05, 0.9, 0.4],
                          colorSpan = 0.5);
```





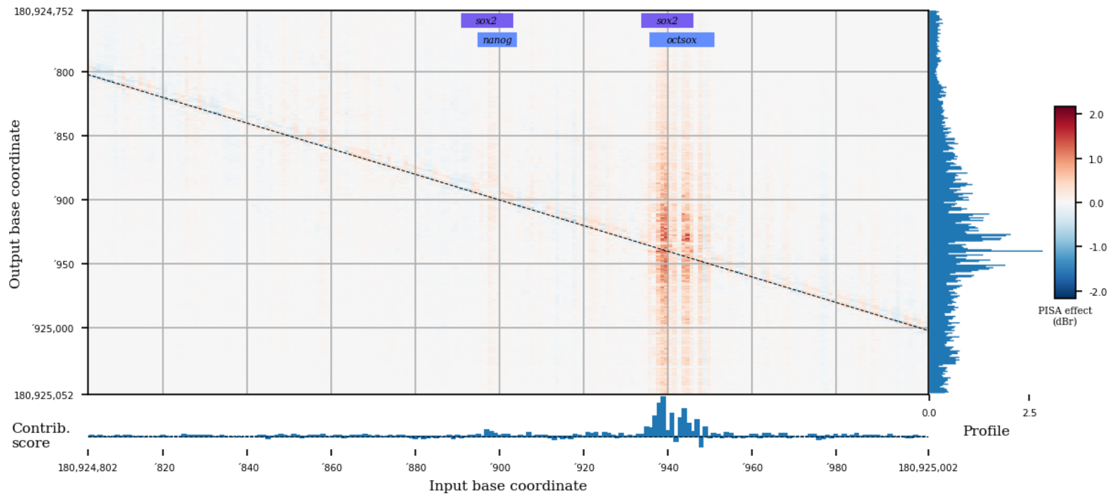
[117]:
```
fig = plt.figure()

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_positive.h5",
                   1150, 30, 100, RECEPTIVE_FIELD,
                   windowStart, "chr1", GENOME_FASTA,
                   WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                   WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                   WORKING_DIRECTORY + "/pred/nanog_residual_positive.bw",
                   nameColors,
                   fig, [0.1, 0.55, 0.9, 0.4],
                   colorSpan = 0.5)

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_nanog_negative.h5",
                   1150, 30, 100, RECEPTIVE_FIELD,
                   windowStart, "chr1", GENOME_FASTA,
                   WORKING_DIRECTORY + "/shap/nanog_profile.bw",
                   WORKING_DIRECTORY + "/scan/nanog_profile.bed",
                   WORKING_DIRECTORY + "/pred/nanog_residual_negative.bw",
                   nameColors,
                   fig, [0.1, 0.05, 0.9, 0.4],
                   colorSpan = 0.5);
```

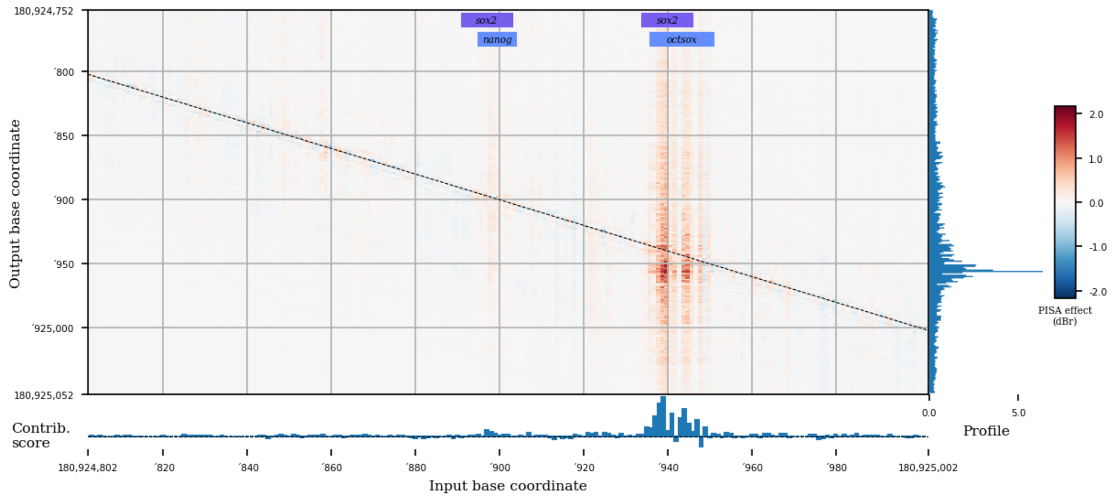




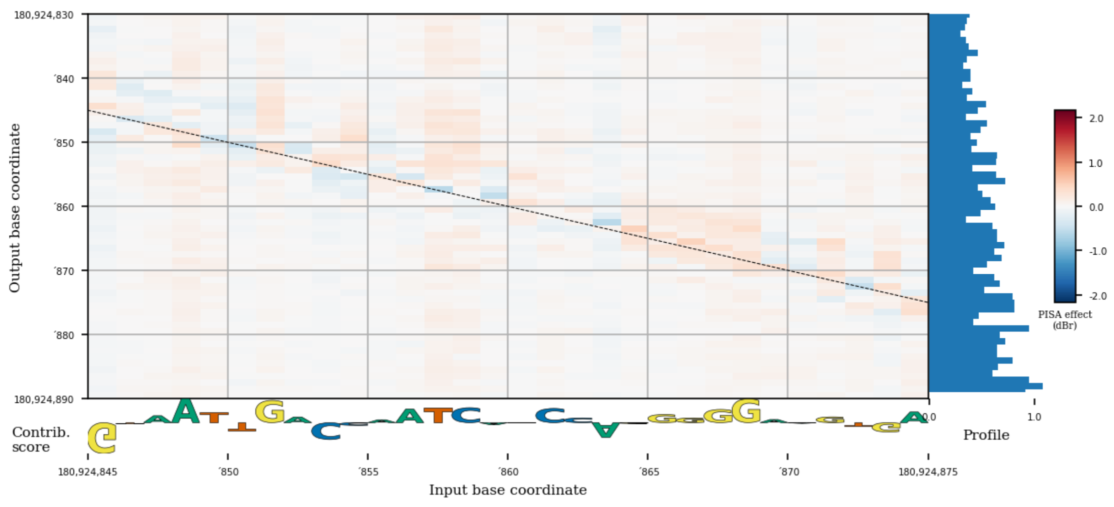
[118]:
```
#Indeed, this motif looks very different!
```

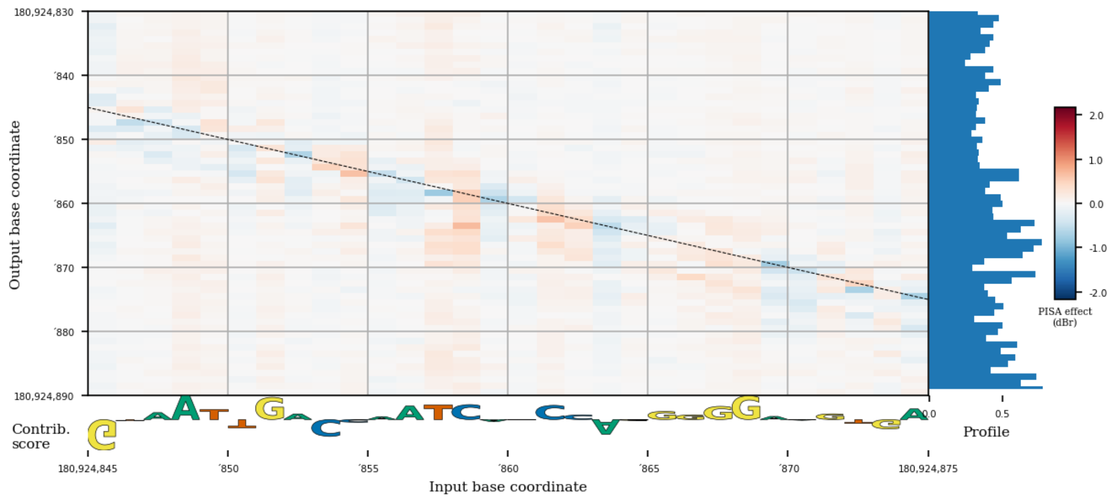
[119]:
```
fig = plt.figure()

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_positive.h5",
                    1150, 200, 300, RECEPTIVE_FIELD,
                    windowStart, "chr1", GENOME_FASTA,
                    WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                    WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                    WORKING_DIRECTORY + "/pred/oct4_residual_positive.bw",
                    nameColors,
                    fig, [0.1, 0.55, 0.9, 0.4],
                    colorSpan = 0.5)

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_negative.h5",
                    1150, 200, 300, RECEPTIVE_FIELD,
                    windowStart, "chr1", GENOME_FASTA,
                    WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                    WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                    WORKING_DIRECTORY + "/pred/oct4_residual_negative.bw",
                    nameColors,
                    fig, [0.1, 0.05, 0.9, 0.4],
                    colorSpan = 0.5);
```





[120]:
```
fig = plt.figure()

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_positive.h5",
                    1108, 30, 60, RECEPTIVE_FIELD,
                    windowStart, "chr1", GENOME_FASTA,
                    WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                    WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                    WORKING_DIRECTORY + "/pred/oct4_residual_positive.bw",
                    nameColors,
                    fig, [0.1, 0.55, 0.9, 0.4],
                    colorSpan = 0.5)

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_negative.h5",
                    1108, 30, 60, RECEPTIVE_FIELD,
                    windowStart, "chr1", GENOME_FASTA,
                    WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                    WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                    WORKING_DIRECTORY + "/pred/oct4_residual_negative.bw",
                    nameColors,
                    fig, [0.1, 0.05, 0.9, 0.4],
                    colorSpan = 0.5);
```

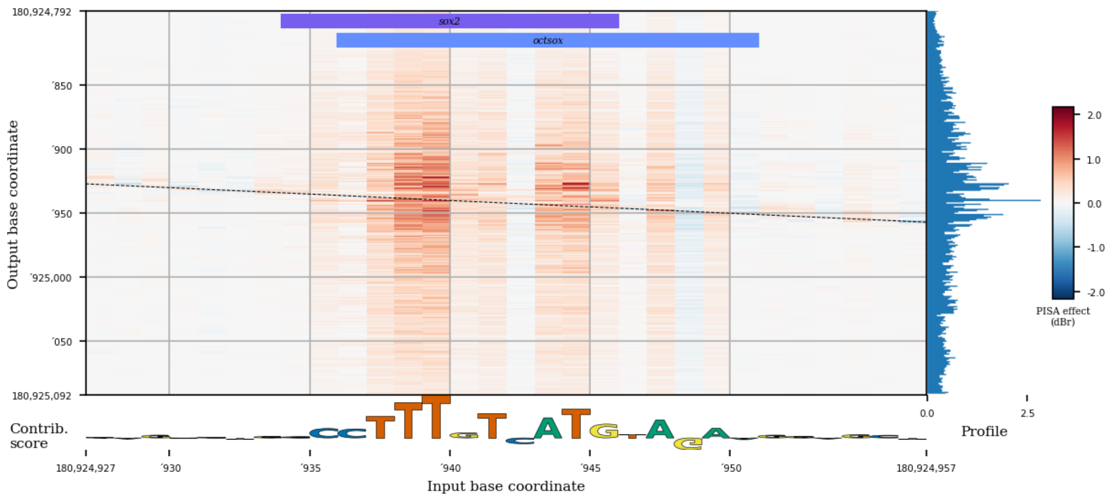


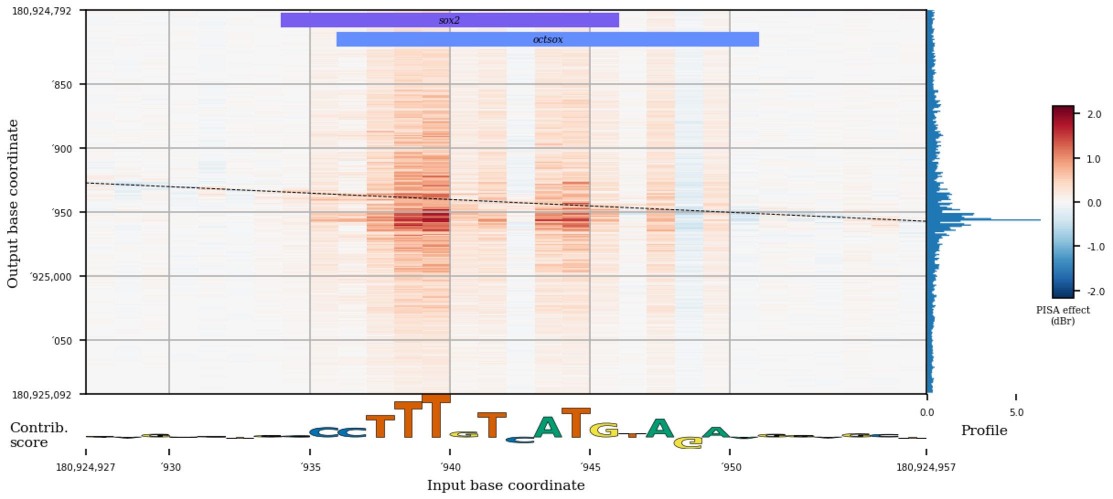


[121]:
```
#Ah, that might explain why the motif acts differently - it's a motif for a␣
↪different protein altogether! Let's zoom in on the Oct4 motif.
```

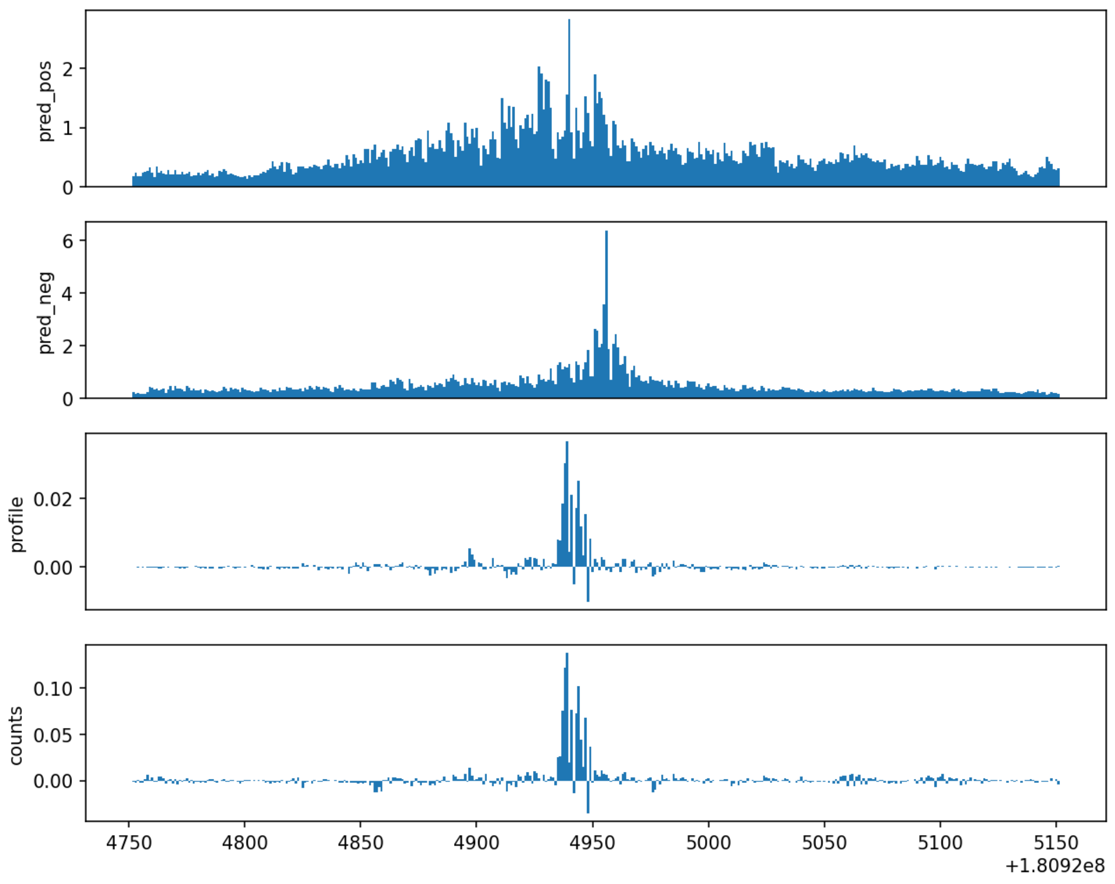
[122]:
```
fig = plt.figure()

bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_positive.h5",
                     1190, 30, 300, RECEPTIVE_FIELD,
                     windowStart, "chr1", GENOME_FASTA,
                     WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                     WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                     WORKING_DIRECTORY + "/pred/oct4_residual_positive.bw",
                     nameColors,
                     fig, [0.1, 0.55, 0.9, 0.4],
                     colorSpan = 0.5)
```

```
bprplots.plotPisaWithFiles(WORKING_DIRECTORY + "/shap/pisa_oct4_negative.h5",
                  1190, 30, 300, RECEPTIVE_FIELD,
                  windowStart, "chr1", GENOME_FASTA,
                  WORKING_DIRECTORY + "/shap/oct4_profile.bw",
                  WORKING_DIRECTORY + "/scan/oct4_profile.bed",
                  WORKING_DIRECTORY + "/pred/oct4_residual_negative.bw",
                  nameColors,
                  fig, [0.1, 0.05, 0.9, 0.4],
                  colorSpan = 0.5);
```





[123]:
```
#As a reminder, let me pull up the Oct4 tracks:
plotShapBigwigs('oct4', 'residual', startPos = 180924752, span=400)
```


pred_pos
0
1
2
pred_neg
0
2
4
6
profile
0.00
0.02
counts
0.00
0.05
0.10
4750
4800
4850
4900
4950
5000
5050
5100
5150
+1.8092e8

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```