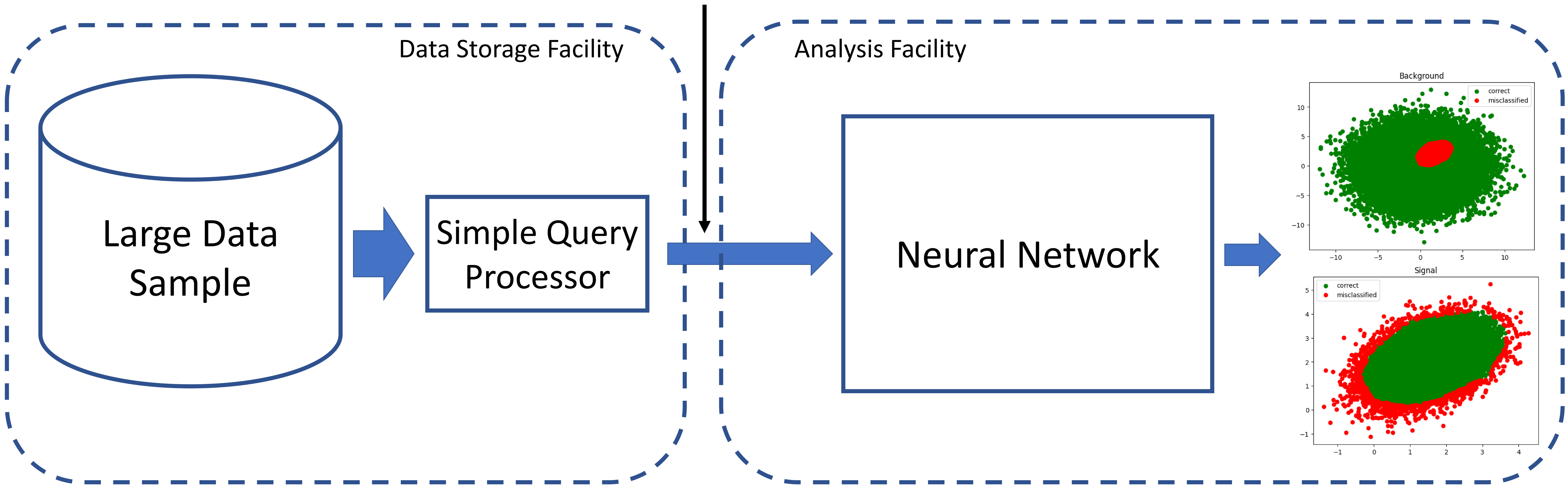
Differentiable Programming: Neural Networks and Selection Cuts Working Together

Gordon Watts

Goal: Minimize data flow along this arrow without compromising results (minimizing caching, networking, etc.)



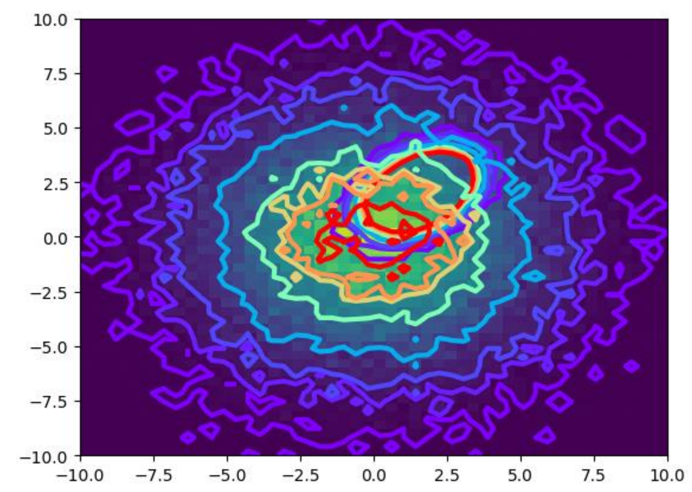
Train Simultaneously

Train & Cut: Could train the NN first, and then adjust cuts till they affected signal region. Does not work will with large number of cuts.

Simultaneously Train: Use gradient descent techniques to train the simple query cuts and the NN

Toy Test MC

- Two Axes
- Data centered at (0,0) and signal centered at 1.5, 2.0
- Data width is (9,9) with no correlation and signal width is (0.5, 0.5) with 20% correlation



```
# The NN MLP training
mlp = hk.nets.MLP(output_sizes=[2,15,30,15,1])

# The selection
selection = Selection(f_cut, initial_cuts=initial_cuts)

# Now the concat. Both these operate on the same input data (the tuple of
# values) - which is required for this concat to work.
combiner = ModuleConcat()

# And run the stuff through it. Use the sigmoid on the MLP result individually.
cut_result = jnp.reshape(selection(x), (x.shape[0], 1))
mlp_result = jax.nn.sigmoid(mlp(x))
combine_result = combiner((cut_result, mlp_result))

# And then the multiply
final = MultiplyRow()

# And put them together in the proper way
return final(combine_result)
```

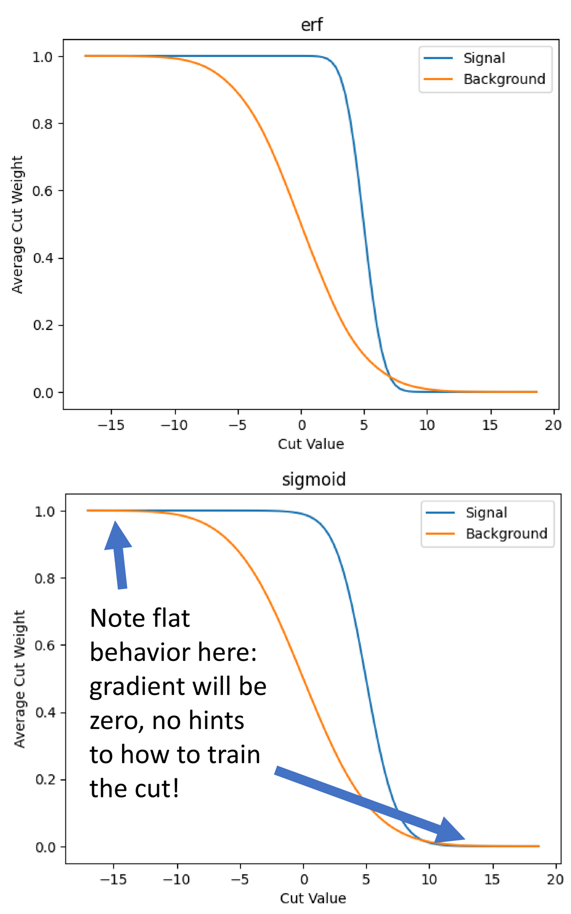
JAX & Haiku

- JAX is a Deep Learning framework from Google
- Encourages python idioms to construct networks
- Designed as a Toolkit
- **Makes it very easy to try new ideas**

P1: Cuts aren't Differentiable

Cuts, like $p_T > 30$ GeV, are discontinuous and differentiable training techniques do not work. We need a continuous weight, not a True/False.

The **Error Function & Sigmoid** were examined. Sigmoid functions trained slightly more quickly and resulting weight distributions were better behaved.

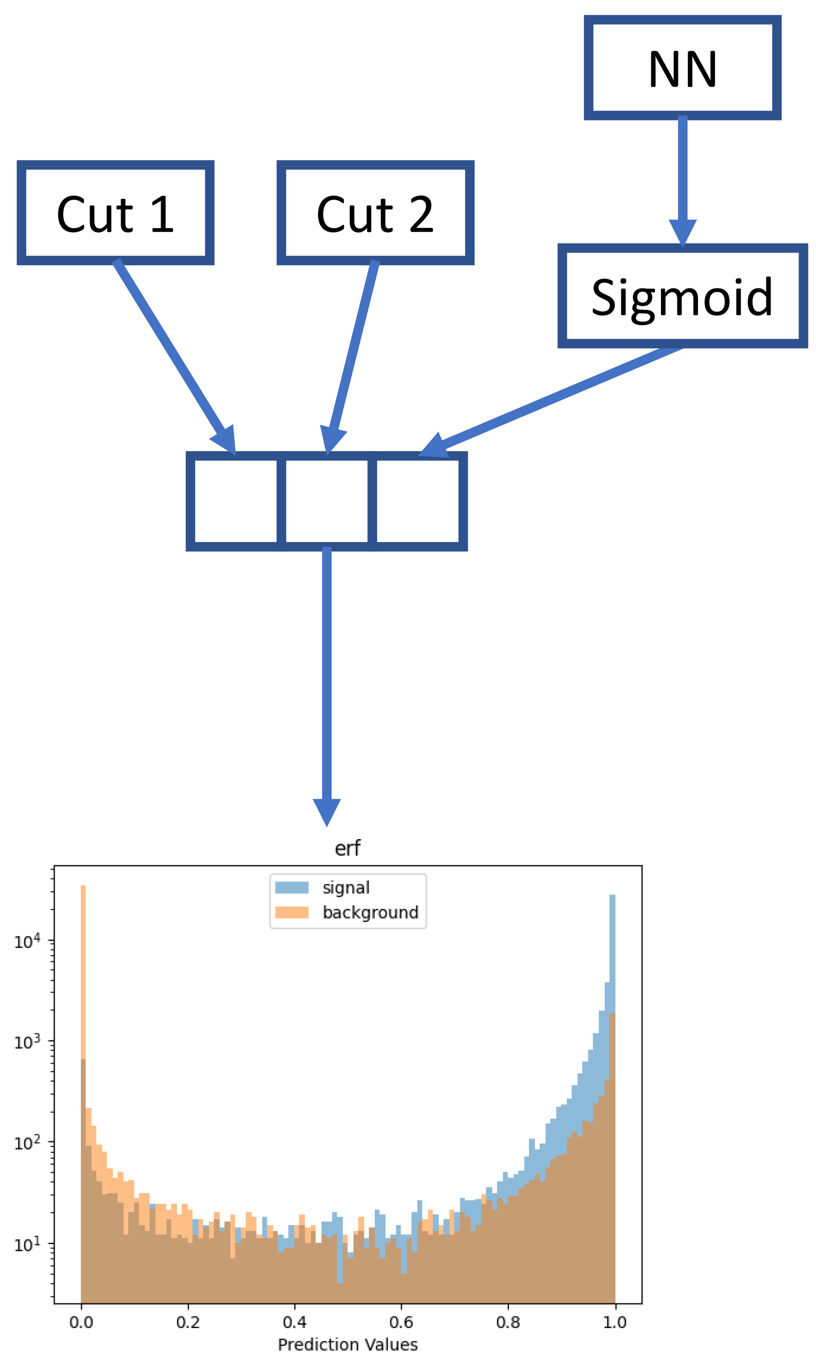


Trained along a single axis, fraction of sample surviving cut.

P2: NN Design

Each cut and the NN output are *weights*

- Treat a weights
- Multiply them together
- Normalize the NN with a sigmoid function only
- Do not normalize the whole thing by a sigmoid – the cut weights need to be cut weights



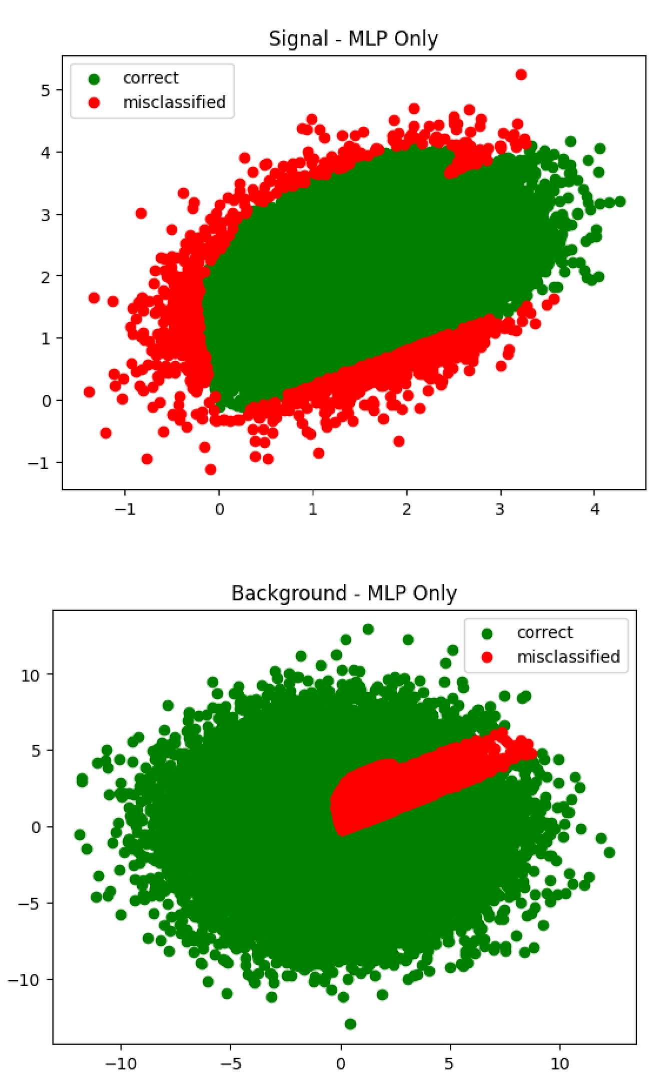
P3: Loss

Typical Loss functions all work.

Do not use the softmax function of the output of the combined NN

$$\frac{e^{x_i}}{\sum e^{x_i}}$$

Will cause the selection cuts to drift down towards very negative numbers!



Next Steps

- This works at **low dimension** and small test data
  - Try higher dimensional data
  - Try an actual analysis
- Use **JAX features to modify the gradient** of the selection functions, so we don't have to pre-specify them
- Test this with multidimensional space
- Add loss function term that is related to size of cached data
- There are other forms of cuts available in the AI community – expectation values, for example.

https://github.com/gordonwatts/diff-prog-intro