Cuts and NN Living together in Peace

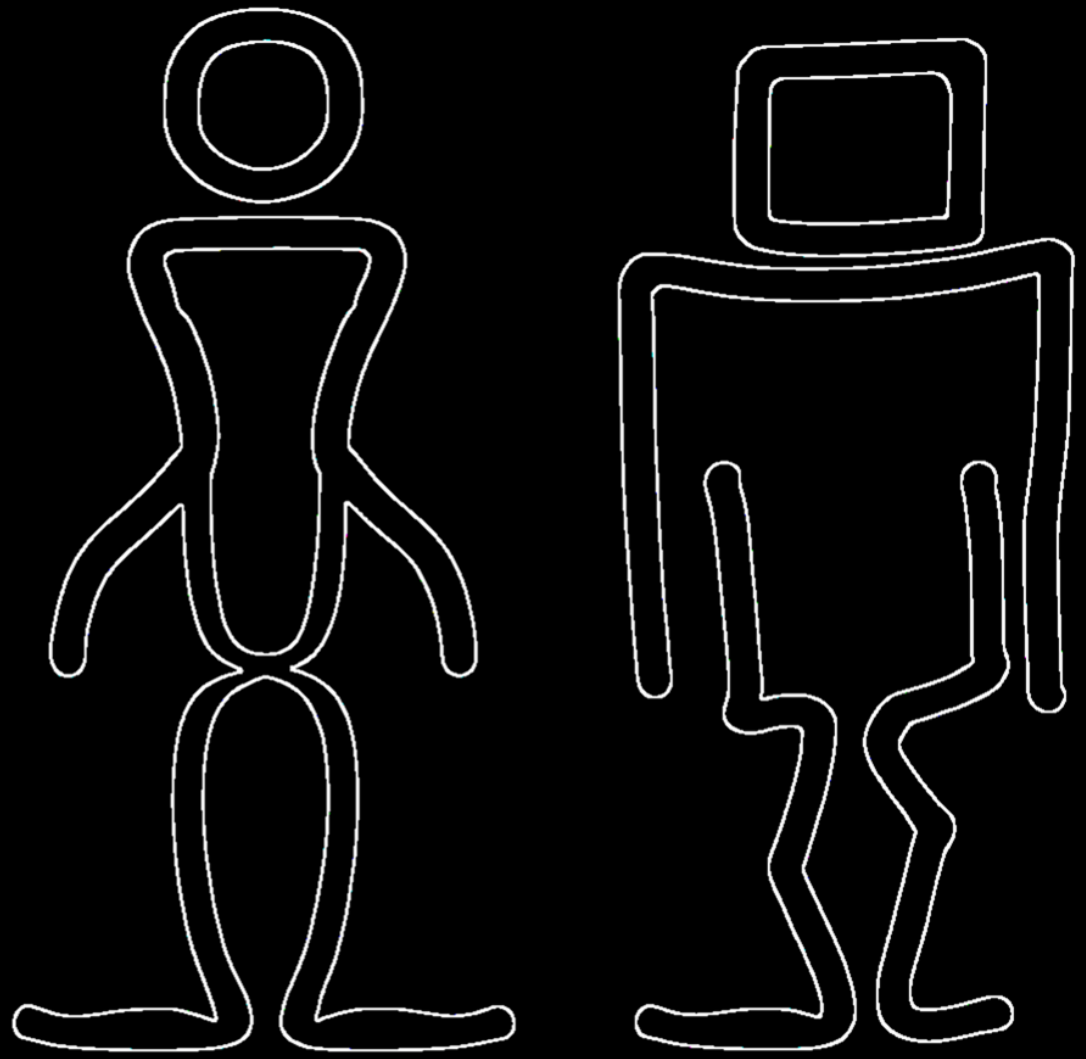
G. Watts (UW/Seattle)

7/25/2023

MODE Workshop

Mr. Cut

Mr. NN



Introduction

G. Watts (UW/Seattle) 3

Training Selection Cuts

G. Watts (UW/Seattle) 10

Training a NN

G. Watts (UW/Seattle) 22

Train Together

G. Watts (UW/Seattle) 25

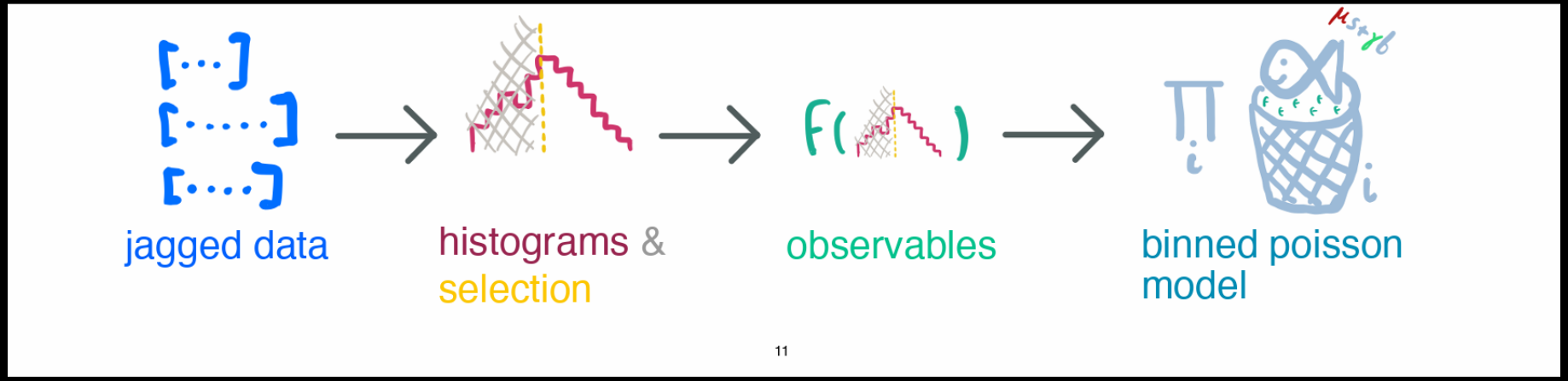
Generated by Bing Chat

Introduction

From Nathan Simpson's Talk

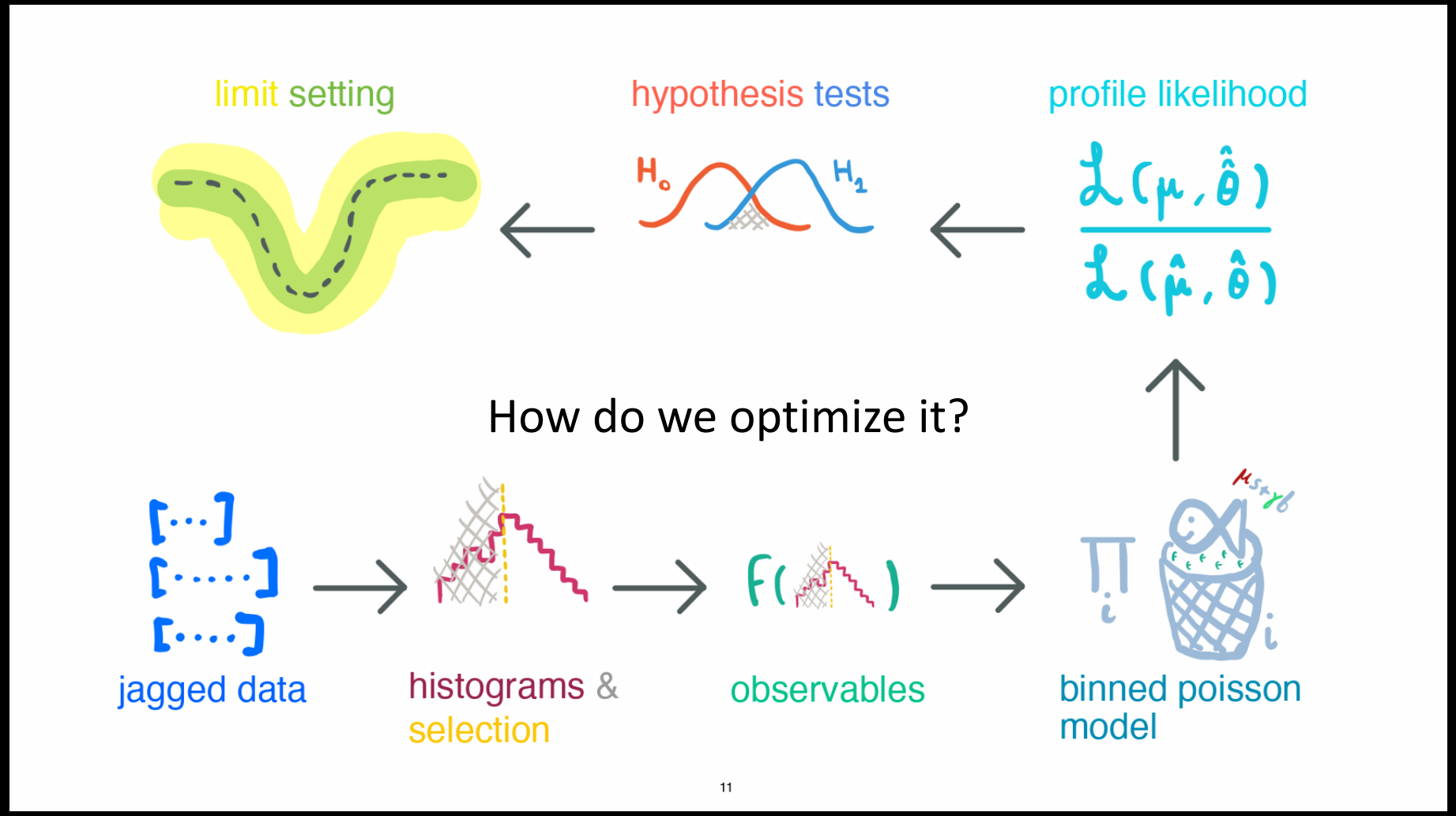
From the first MODE workshop

What is a physics Analysis?

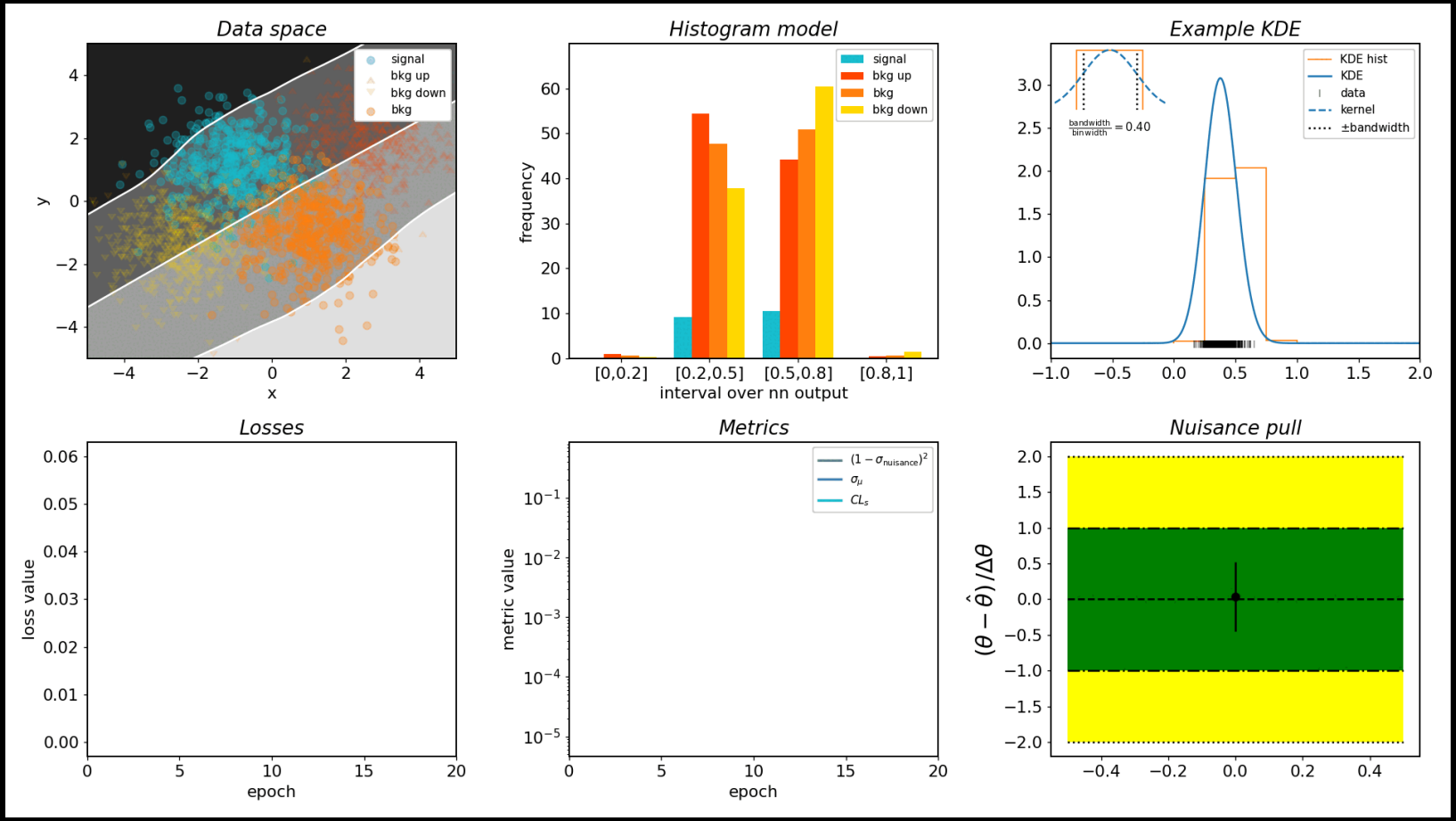


From Nathan Simpson's Talk

From the first MODE workshop



NEOS


Data space
signal
bkg up
bkg down
bkg
x
y
Histogram model
signal
bkg up
bkg
bkg down
frequency
[0,0.2]
[0.2,0.5]
[0.5,0.8]
[0.8,1]
interval over nn output
Example KDE
KDE hist
KDE
data
kernel
±bandwidth
bandwidth / binwidth = 0.40
Losses
loss value
epoch
Metrics
$(1-\sigma_{\text{nuisance}})^2$
σ_μ
CL_s
metric value
epoch
Nuisance pull
$(\theta-\hat{\theta})/\Delta\theta$

An Analysis

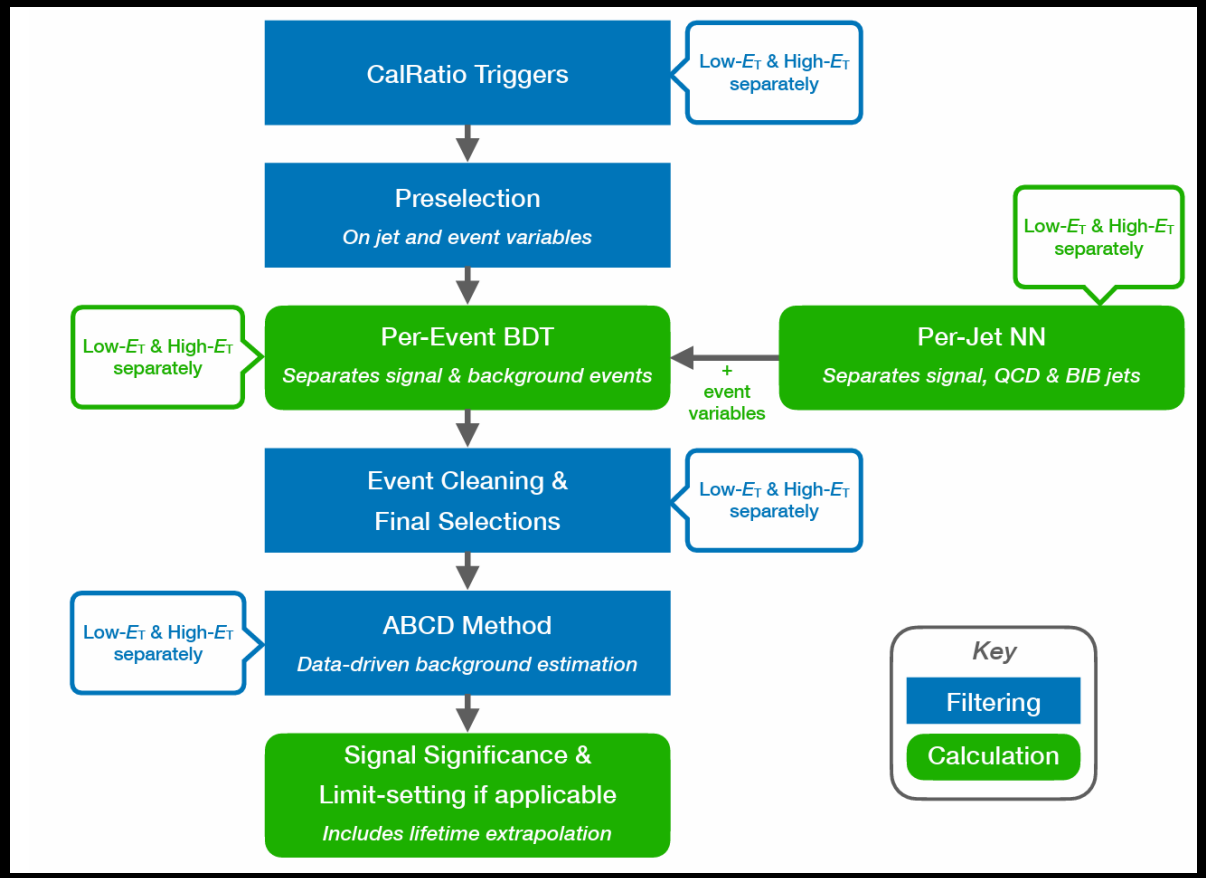
PBs of reconstructed data

Reduce PBs of data to something manageable without cutting out physics

Mercilessly kill background to bring out the signal, measurement, etc.

Histograms, etc., to statistically calculate how well you did

Example from Published ATLAS Analysis



Data

Simple Cuts

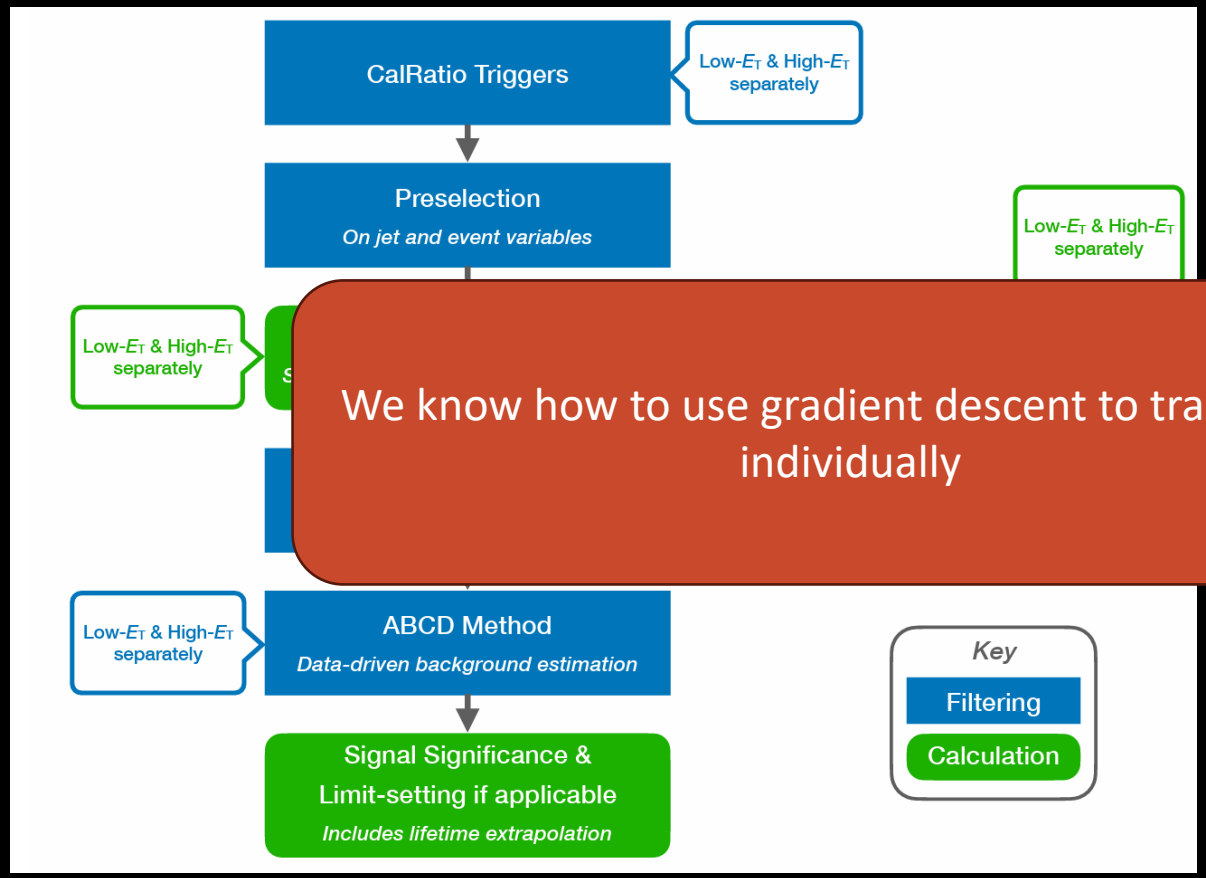
NN and BDT

More cuts

ABCD (histogram)

Statistical Model

Example from Published ATLAS Analysis



We know how to use gradient descent to train these individually

Data

Simple Cuts

NN and BDT

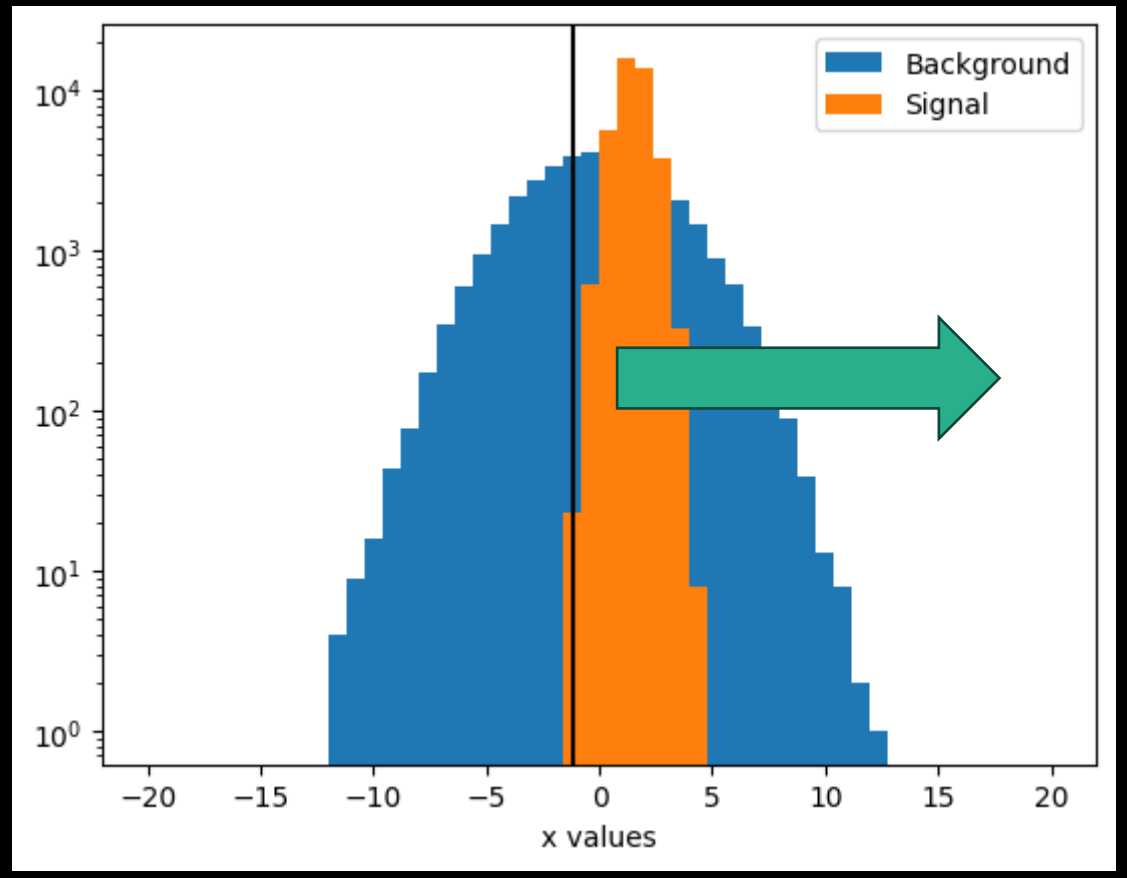
More cuts

ABCD (histogram)

Statistical Model

Training Selection Cuts

What is a Simple Cut?



Keep data this side of the line…

Keep only the data on one side of a line.

In HEP:

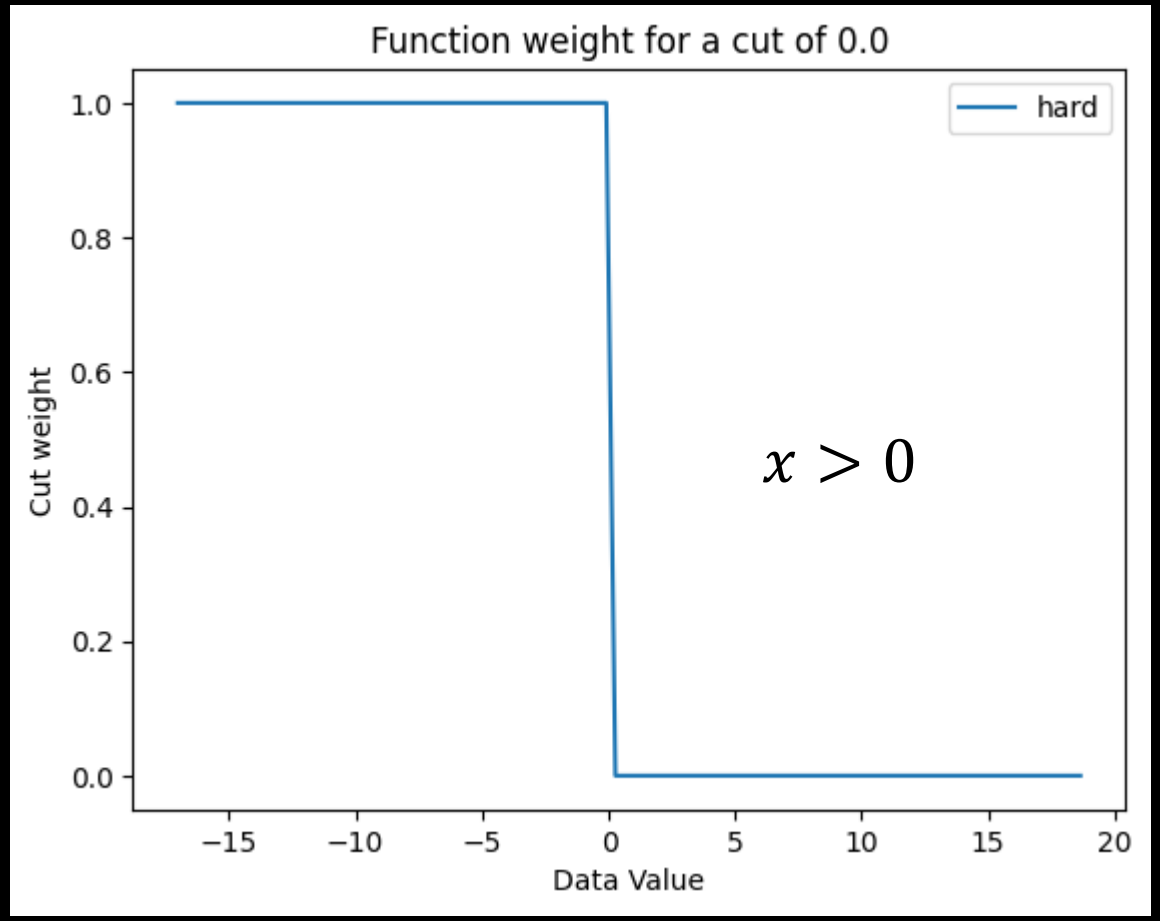
$$x > 1.127$$

If we think of this in terms of probability, then:

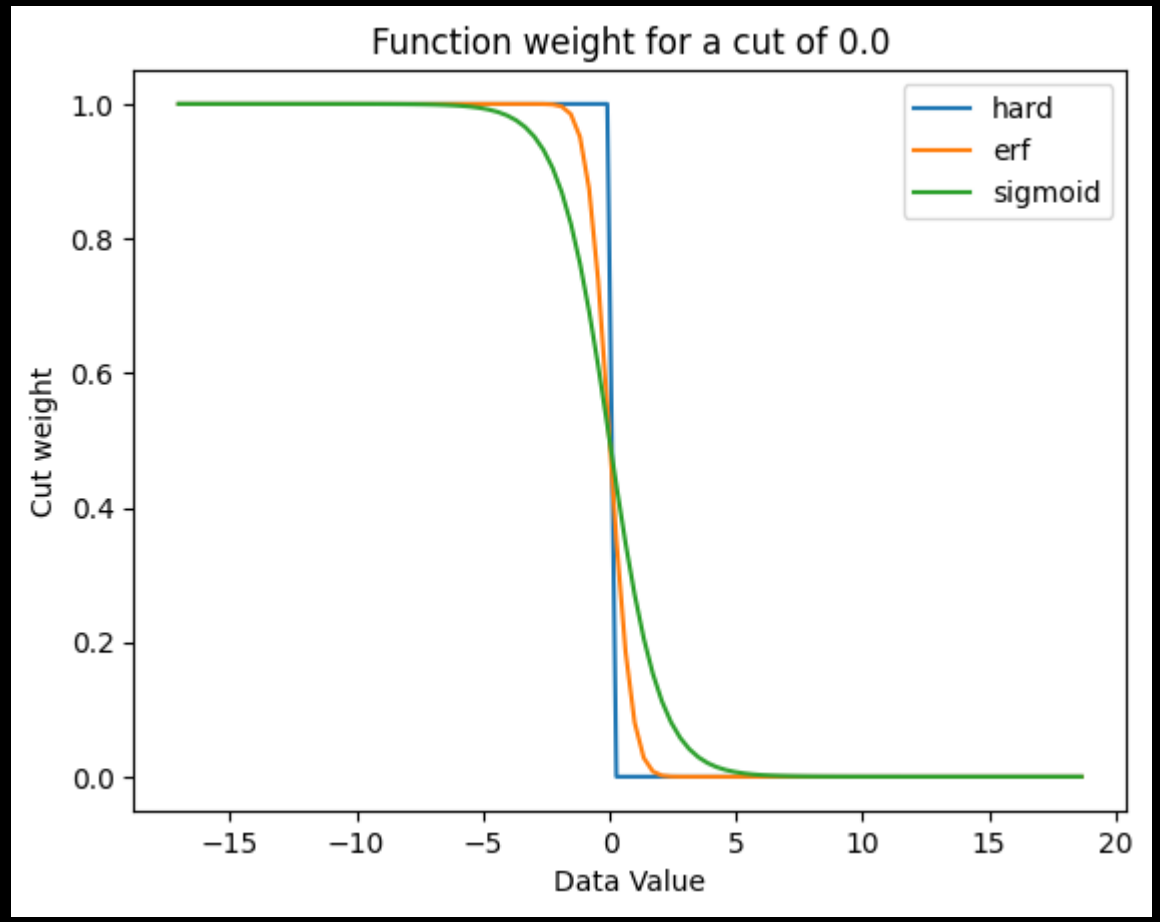
$$weight(x) = \begin{matrix} x > -1.127 & 1.0 \\ x \leq -1.127 & 0.0 \end{matrix}$$

Cuts Do Not A Smooth Gradient Make…

All code is on github!



The Fix is Well Known…

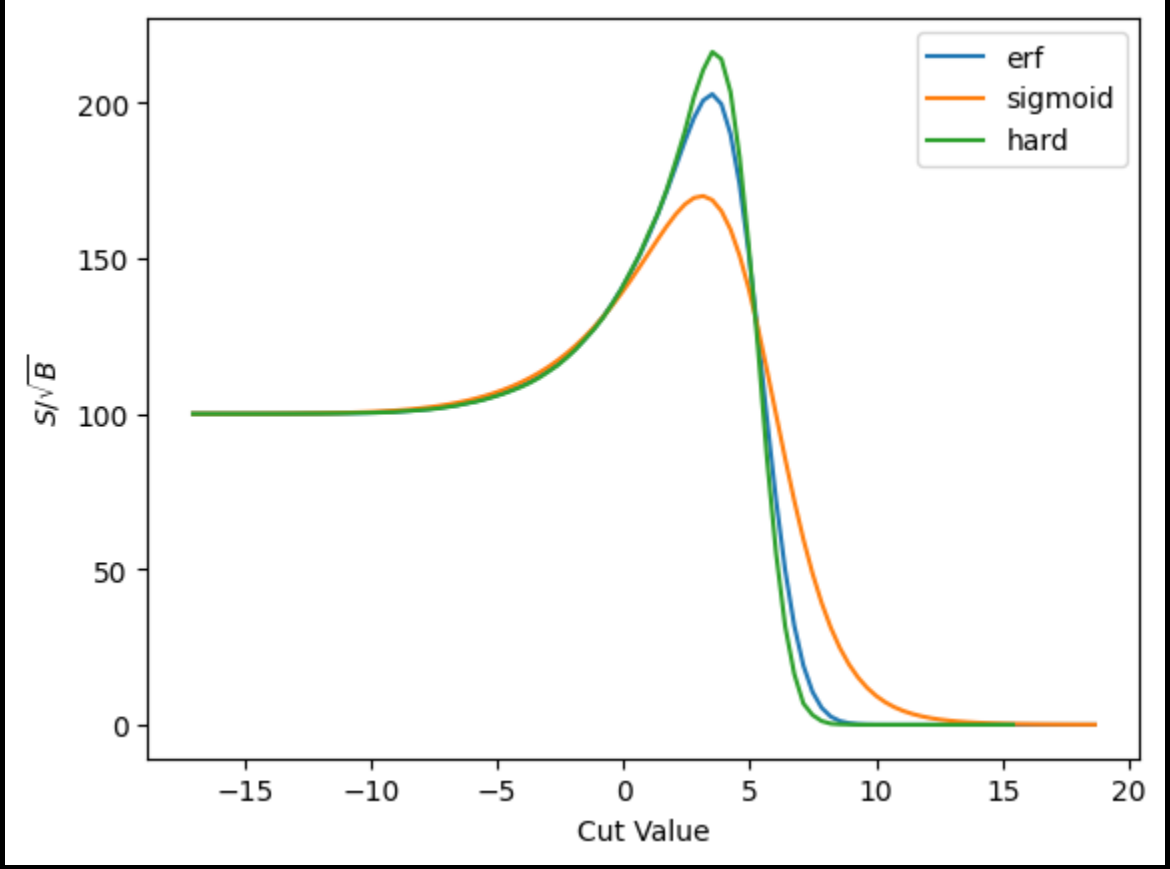


$$\mathrm{ERF}(x) = \frac{2}{\sqrt{2}} \int_0^x e^{-x^2}\, dx$$

(renormalize to [0,1))

$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

It’s close…



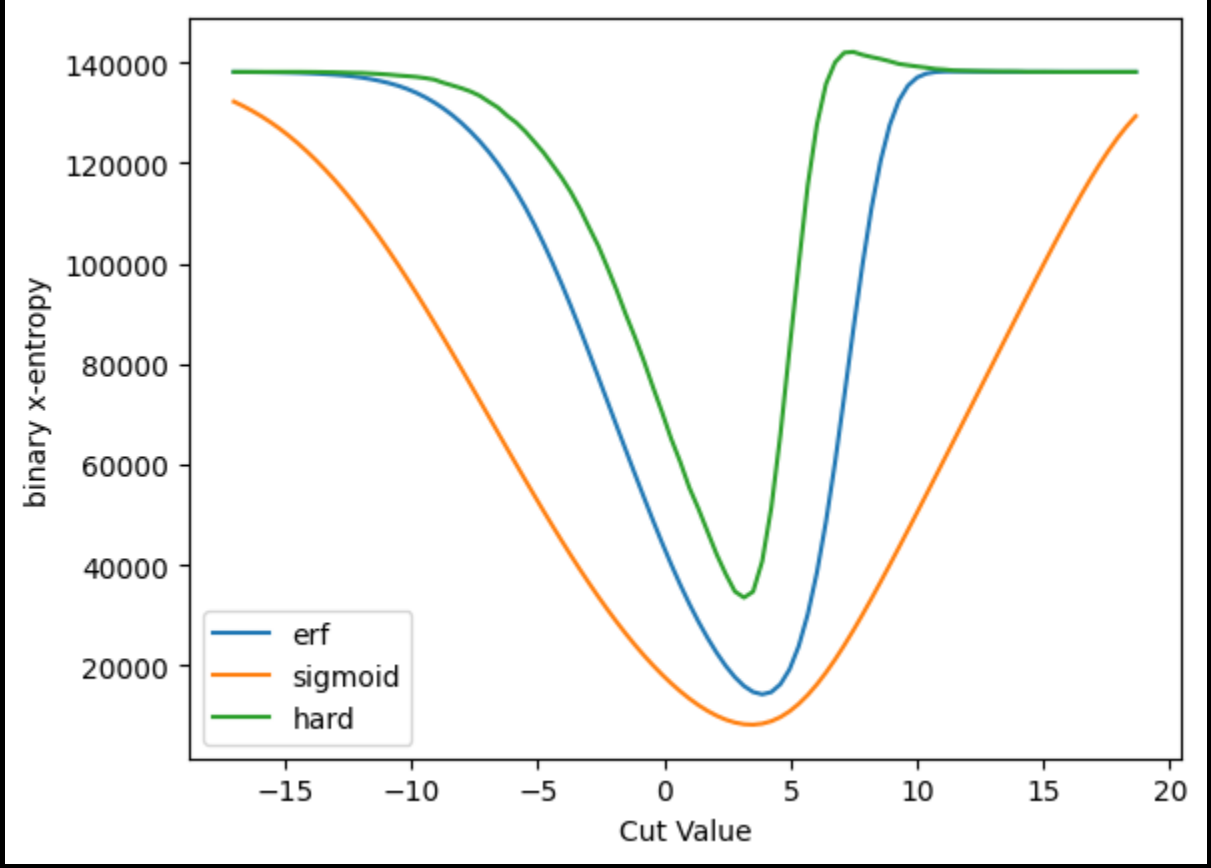
Use the “easy” $S/\sqrt{B}$

Note the peaks do not *quite* line up

Generate some fake data sampling from a gaussian

- Signal $\mu = 4.0, \sigma = 1$
- Background $\mu = 1.0, \sigma = 4$

More obvious with binary x-entropy

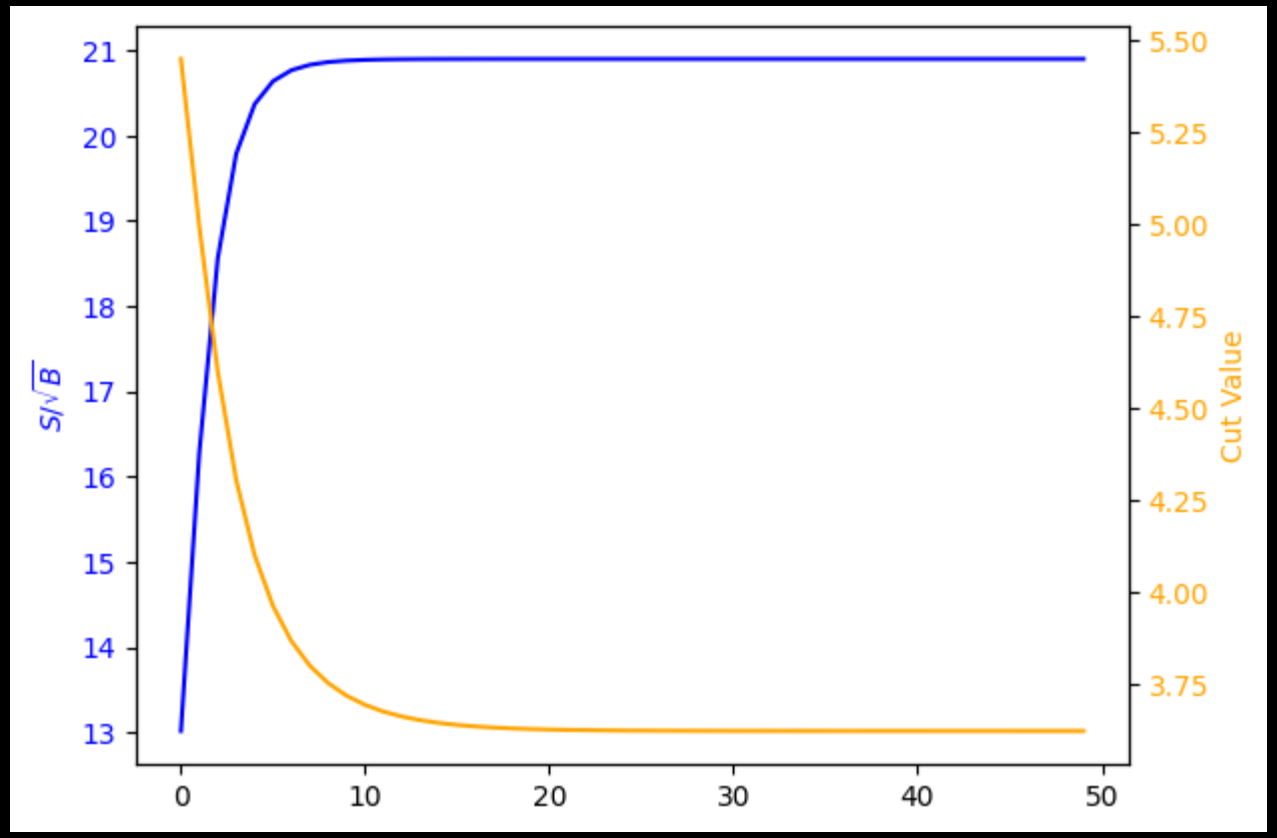


1

The “best” cut values are not the same as a straight cut.

These are emulations!

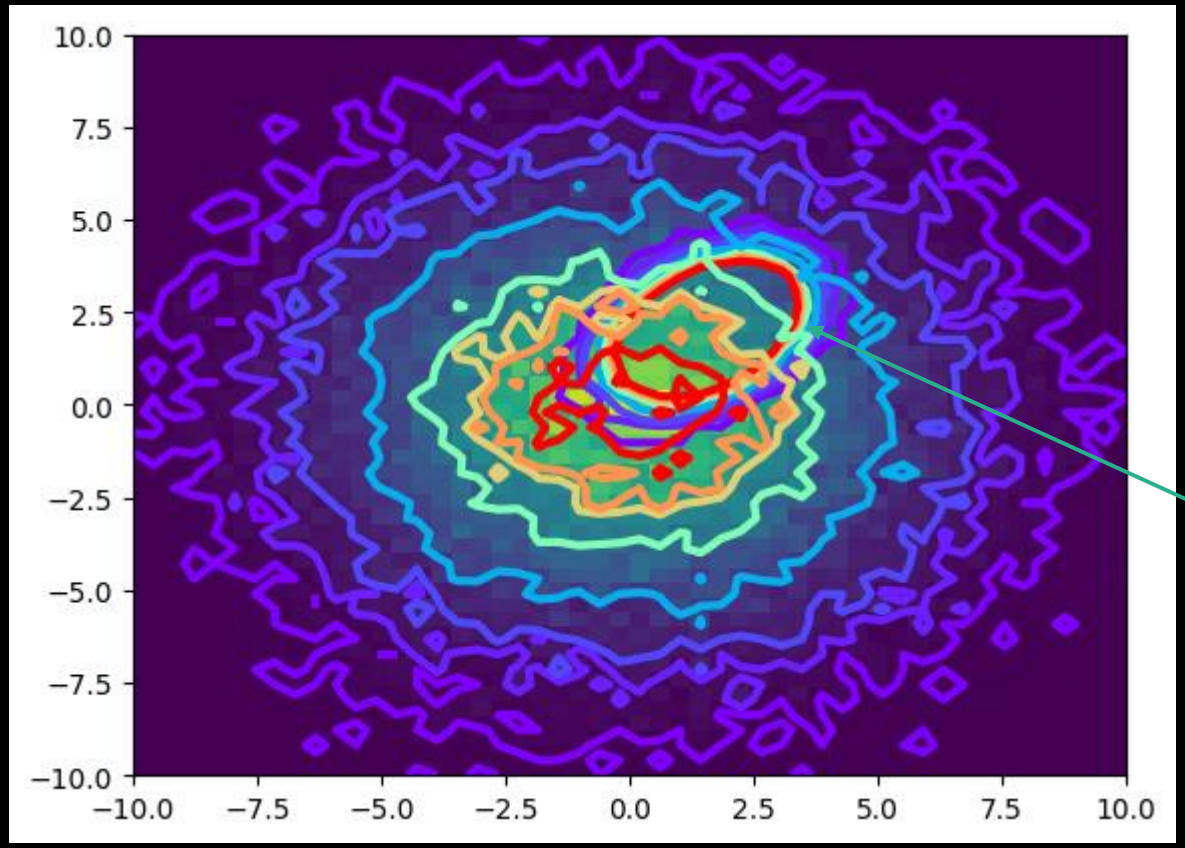
Training



Build a very simple gradient descent training loop using raw JAX

- Finds the proper value quickly, ~20 iterations
- Gets the right value (even).

Training...

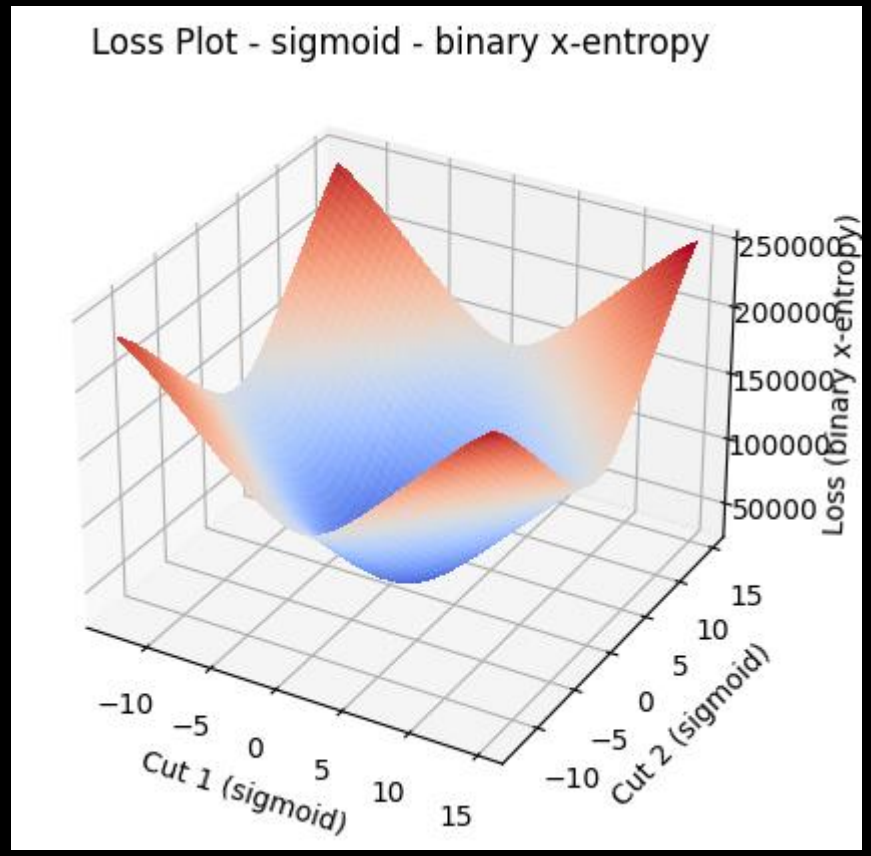


Simple 2D data source:

- (x,y) are both gauusians
- Averages are close, different widths
- And some correlation for signal, but not background.

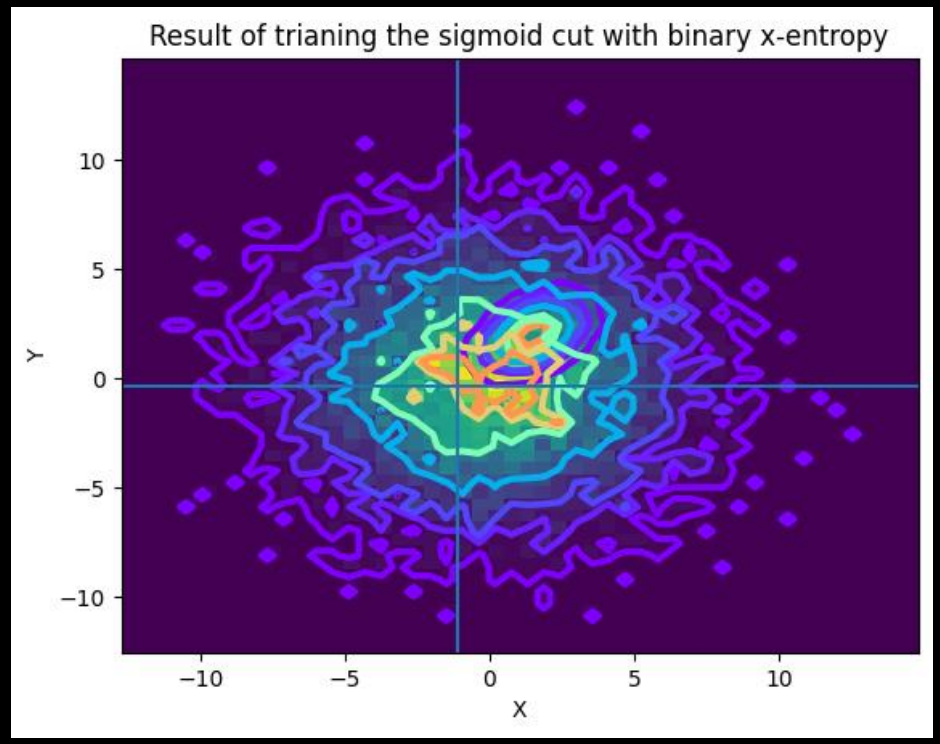
Signal

Loss Plot for 2D cuts



Very well behaved!

Graphical Results of the training…



Haiku

One of the more common DL libraries for JAX

- I do not want to re-implement all the DL tools out there
- I want cuts to be "**first class**" citizens in the JAX eco-system
- Cuts are implemented as a Haiku module

Few notes:

- This takes the cut function as an argument
  - The exact function listed earlier (sigmoid, erf, etc.)
- Initial value of cut is important, as we will see!!

```python
class Selection(hk.Module):
    """Apply a selection cut to each input, output is a weight,
    zero if the cut would reject, one if it would not"""

    def __init__(
        self,
        f_cut: Callable,
        initial_cuts: Optional[List[float]] = None,
        name="SelectionCut",
    ):
        super().__init__(name=name)
        self._initial_cuts = initial_cuts
        self._f_cut = f_cut

    def __call__(self, x):
        "Apply a selection cut for all the incoming data."

        # See if we have a decent set of initializers
        cuts_initial = (
            jnp.asarray(self._initial_cuts)
            if self._initial_cuts is not None
            else jnp.ones(x.shape[1])
        )
        assert (
            cuts_initial.shape[0] == x.shape[1]
        ), f"Incorrect number of initial cut values specified - need {x.shape[1]}"

        # Get the parameters to apply here
        cuts = hk.get_parameter(
            "cuts",
            shape=[x.shape[1]],
            dtype=jnp.float32,
            init=lambda shp, dtyp: cuts_initial,
        )

        # Next, apply the cut
        cut_data_pairing = [
            (cuts[index], x[:, index]) for index in range(0, x.shape[1])
        ]
        wts = jnp.stack([self._f_cut(c, x_col) for c, x_col in cut_data_pairing],
                        axis=1)

        return jnp.prod(wts, axis=1)
```

Training isn’t as fast…

```
1  sigmoid_cut_results = train_cut(training_data, training_truth, cut_sigmoid, 5000)['SelectionCut']['cuts']
2  print(f'Sigmoid cut results = {sigmoid_cut_results}')
```

```
NegLogLoss : 98698.24, epoch: 1
NegLogLoss : 98554.80, epoch: 500
NegLogLoss : 98512.15, epoch: 1000
NegLogLoss : 98504.36, epoch: 1500
NegLogLoss : 98503.14, epoch: 2000
NegLogLoss : 98502.99, epoch: 2500
NegLogLoss : 98502.98, epoch: 3000
NegLogLoss : 98502.98, epoch: 3500
NegLogLoss : 98502.98, epoch: 4000
NegLogLoss : 98502.98, epoch: 4500
NegLogLoss : 98502.98, epoch: 5000
Sigmoid cut results = [-1.0773739 -0.3744329]
```

Stable around 2500 epochs…

- Error function trains a bit more slowly

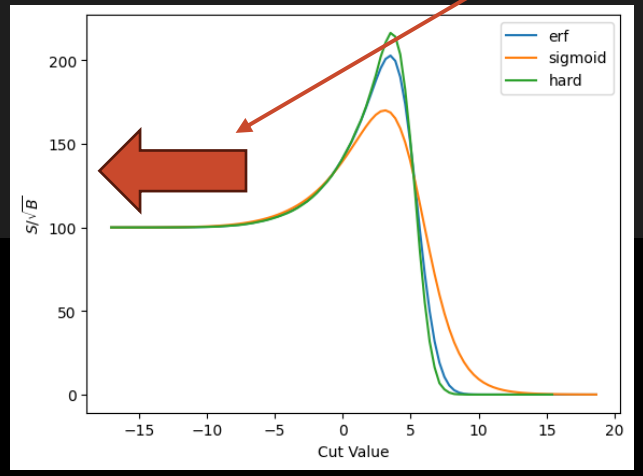
Since there is no data, the gradient is flat!!!!

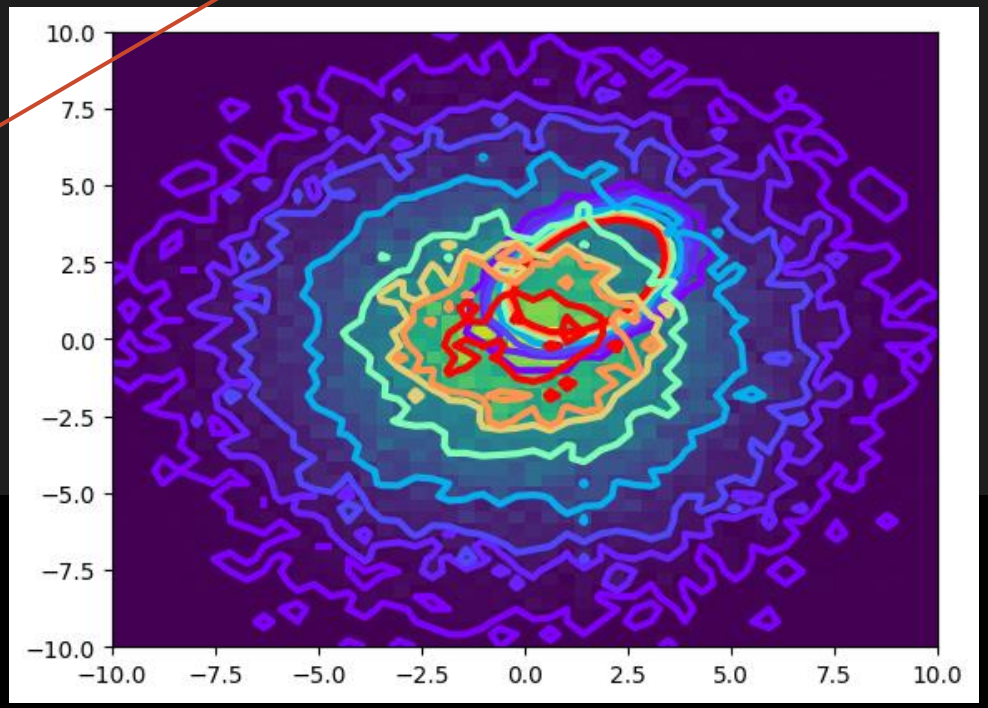
But...

Way off the scale – no data!!!!

```
1  sigmoid_cut_results_faraway = train_cut(training_data, training_truth, cut_sigmoid, 5000, initial_cuts=(60.0, 60.0))['SelectionCut']['cuts']
2  print(f'Sigmoid cut results = {sigmoid_cut_results_faraway}')
```

```
NegLogLoss : 99034.87, epoch: 1
NegLogLoss : 99034.87, epoch: 500
NegLogLoss : 99034.87, epoch: 1000
NegLogLoss : 99034.87, epoch: 1500
NegLogLoss : 99034.87, epoch: 2000
NegLogLoss : 99034.87, epoch: 2500
NegLogLoss : 99034.87, epoch: 3000
NegLogLoss : 99034.87, epoch: 3500
NegLogLoss : 99034.87, epoch: 4000
NegLogLoss : 99034.87, epoch: 4500
NegLogLoss : 99034.87, epoch: 5000
Sigmoid cut results = [60. 60.]
```





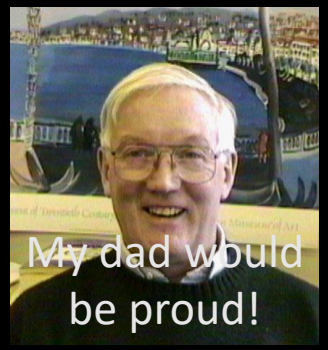
2 Choose your initial cuts carefully!

Training a NN

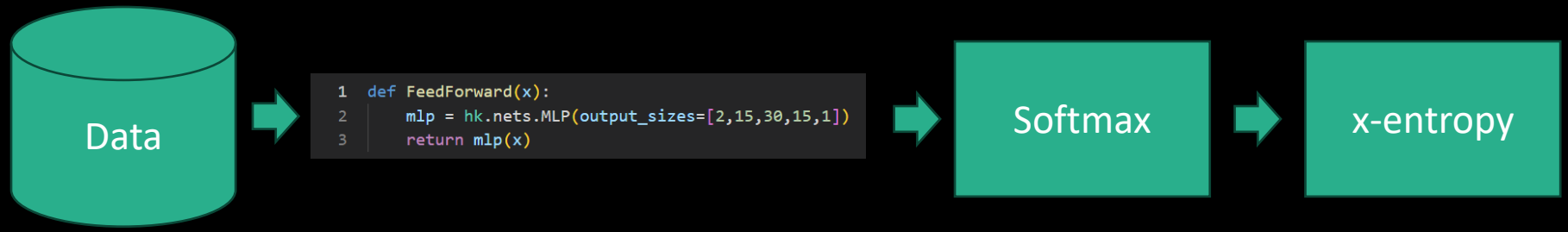
Network is very simple!

Using a very very simple fully connected network:

- 2 inputs
- 1 output
- 3 hidden layers



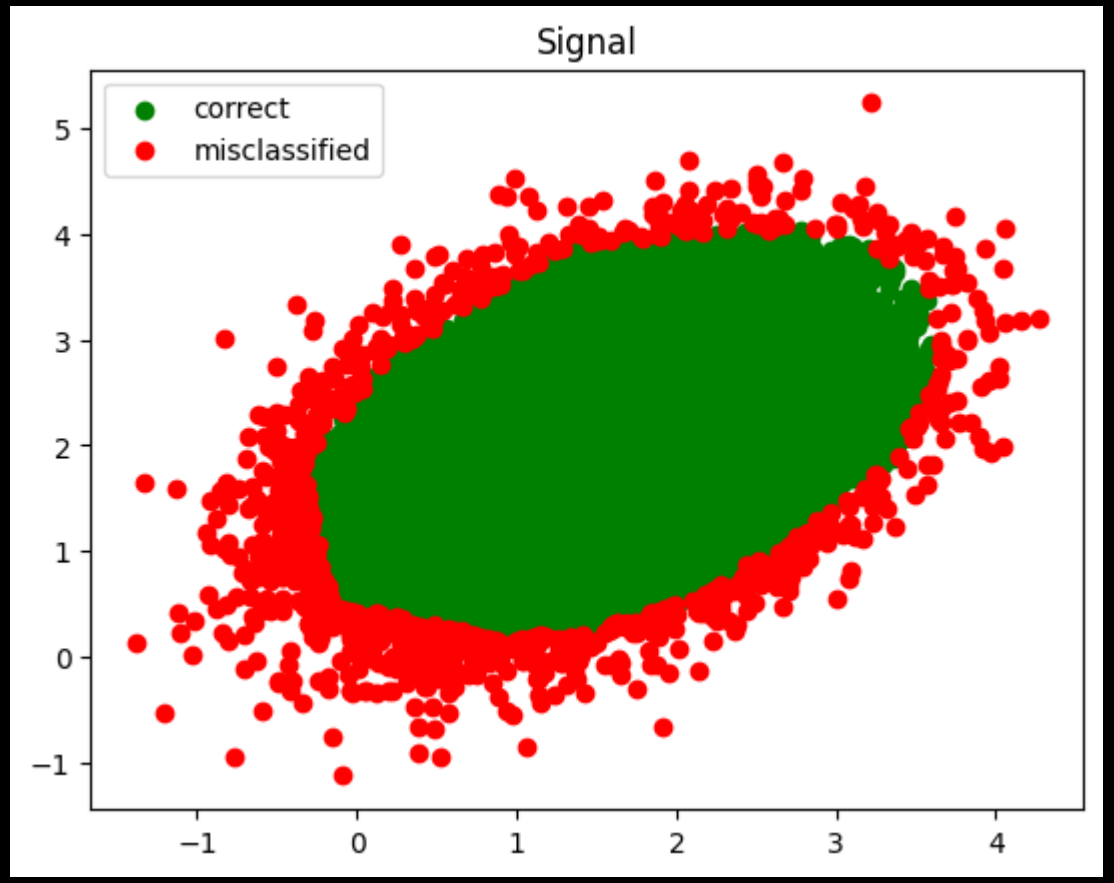

```python
def FeedForward(x):
    mlp = hk.nets.MLP(output_sizes=[2,15,30,15,1])
    return mlp(x)

model = hk.transform(FeedForward)
```

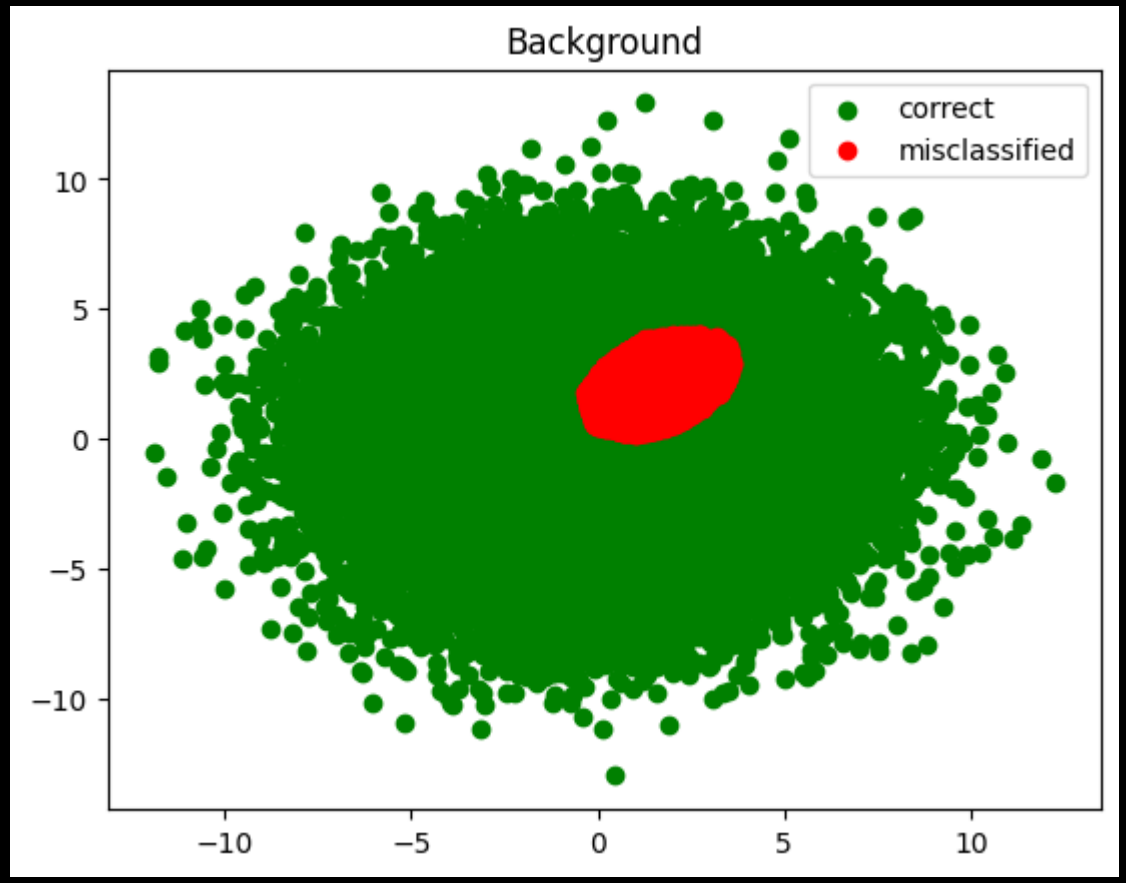


No problem!

Trains faster than the cuts-only version!

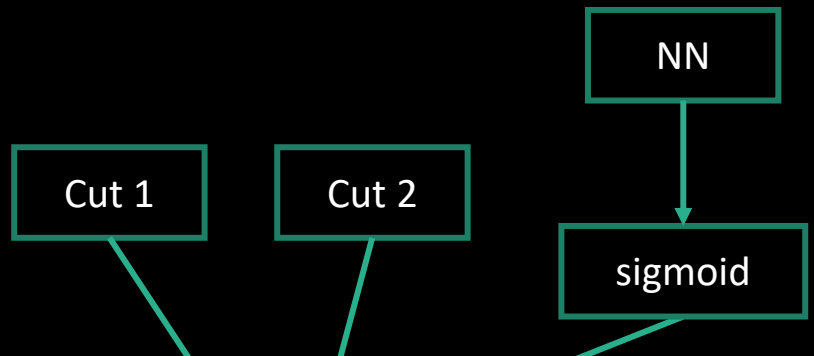
```
NegLogLoss : 450086.47, epoch: 1
NegLogLoss : 440696.31, epoch: 200
NegLogLoss : 440081.81, epoch: 400
NegLogLoss : 439910.03, epoch: 600
NegLogLoss : 439677.06, epoch: 800
NegLogLoss : 439542.91, epoch: 1000
NegLogLoss : 439481.50, epoch: 1200
NegLogLoss : 439340.09, epoch: 1400
NegLogLoss : 439237.56, epoch: 1600
NegLogLoss : 439129.94, epoch: 1800
NegLogLoss : 439072.69, epoch: 2000
```





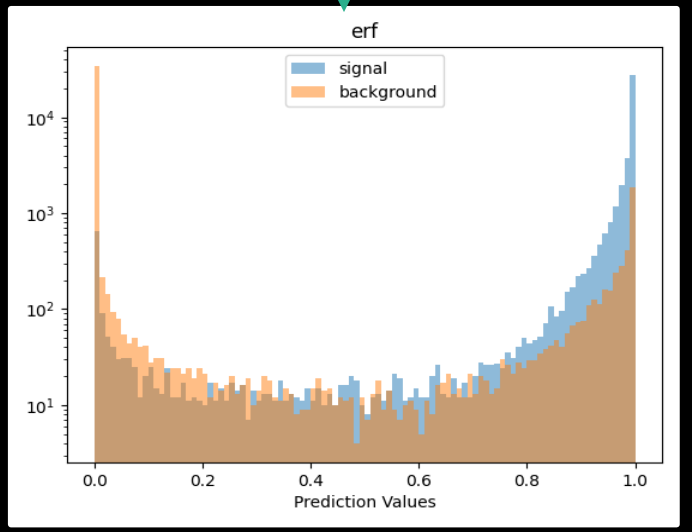
Train Together

What does it mean to be a cut?



1. Calculate the weighs of the cuts
2. Calculate the value of the NN
3. Multiply them together

If a cut's weight is zero, the NN's result does not affect the loss function or back-prop!



3

Apply the softmax function only to the NN outputs, not the cuts!!

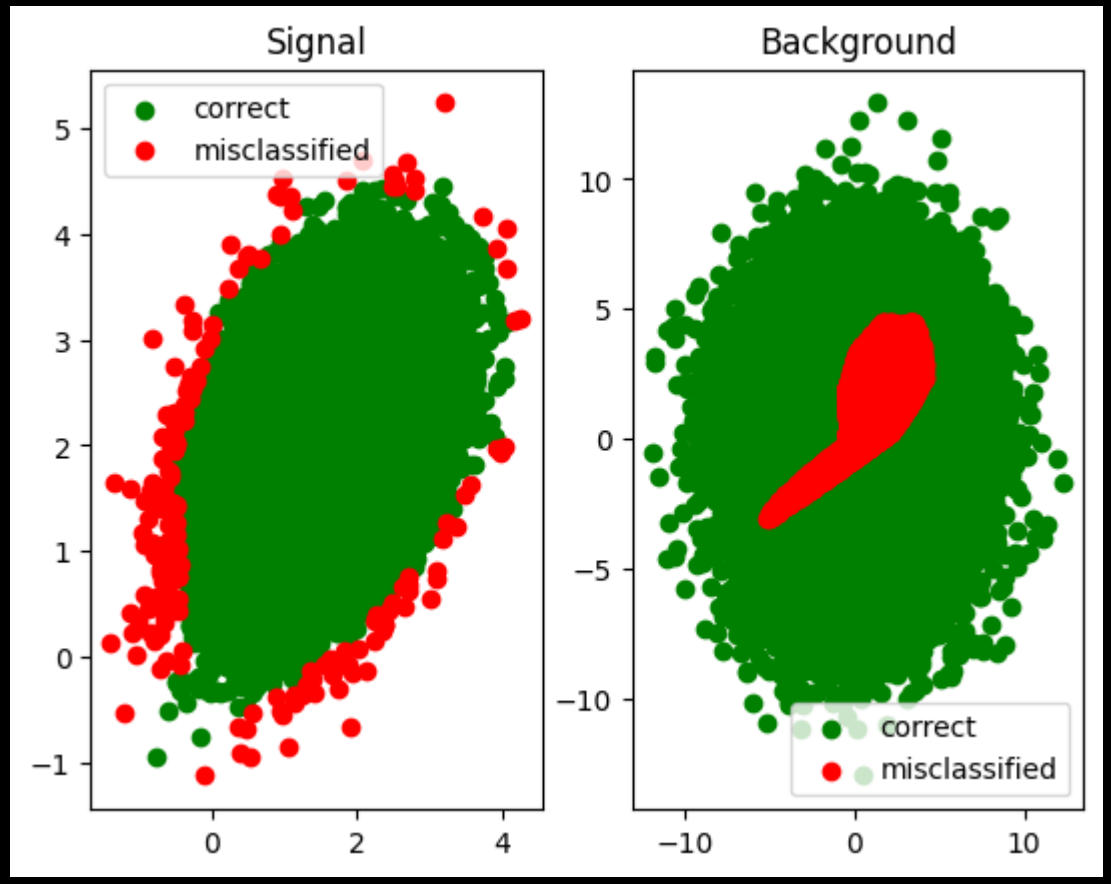
Your cuts will otherwise always be pushed down to zero!

Final network design just for reference

Note the actual code a physicist has to write is very clean!

```
# 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)
```

And we get most of the way there…



This is after 8000 epochs

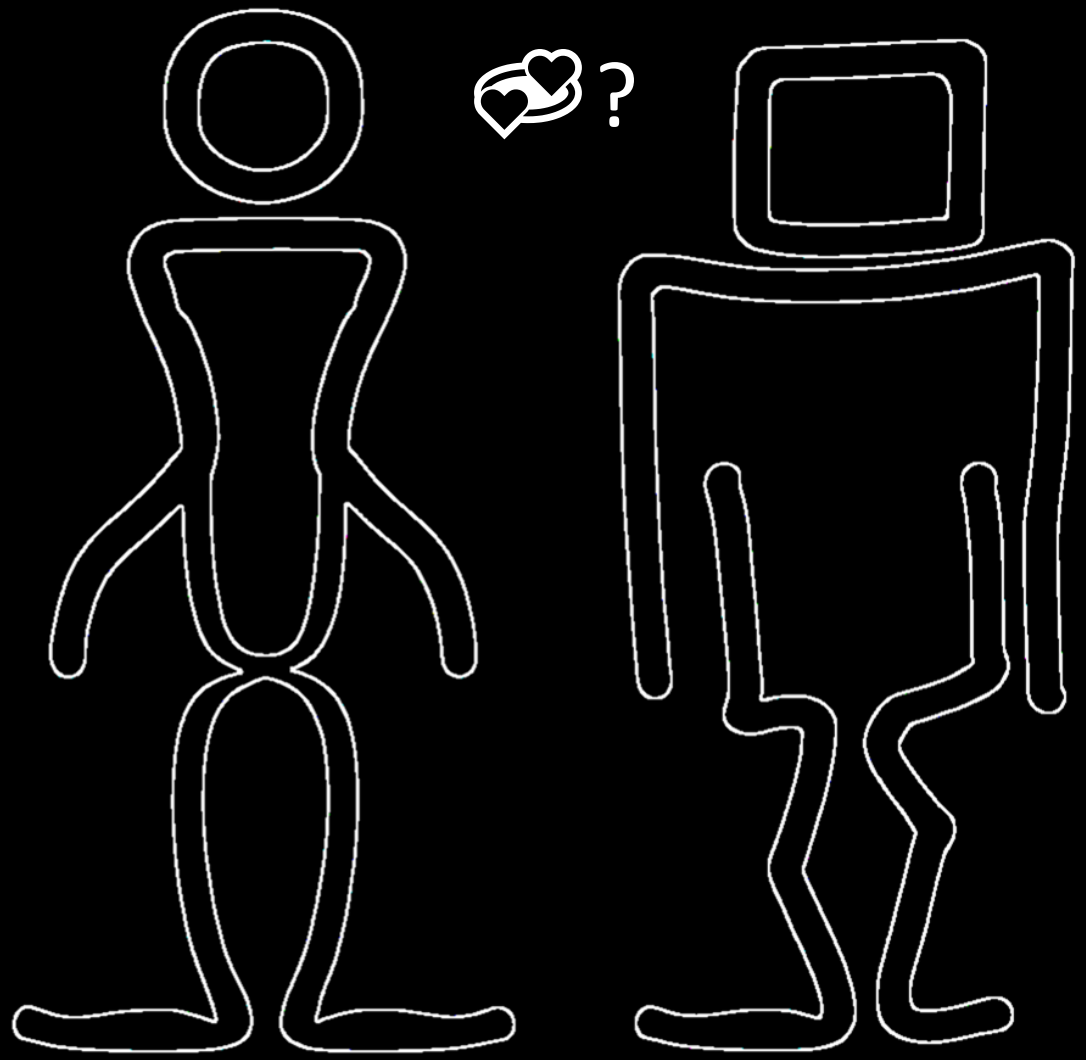
- The current value of the cuts just touches the edge of the red

4 Training takes too long

For this job, why not just train with the NN, and then crank up the cuts until there is an effect?

Lessons

1. The “best” cut values are not the same as a straight cut. ← Annealing?
2. Choose your initial cuts carefully! ← Automate initial data scan? Add fake gradient outside of data bounds?
3. Apply the softmax function only to the NN outputs, not the cuts!! ← Easy fix here… But what about multiple cut layers?
4. Training takes too long ← Train separately? Too complex? Different approach?



Generated by Bing Chat

Conclusions

- I, at least, learned a lot
  - JAX, Haiku
  - Ripping apart a NN to study how the components work
  - I can get at individual gradients during training to understand why they have the values they have!
- How to make this easy for a physicist?
  - Should be like adding a cut to be a one-liner
  - Haiku/PyTorch and other frameworks give us the coding infrastructure
  - But cuts clearly have effects beyond their bounds (viral!)
  - But one technique is easy to implement in other contexts
- What is next:
  - Study the slow-training a bit further
  - Stochastic estimation of a hard cut