Let’s say we have 2 segments, and , , then the likelihood in the CPL paper can be written as (please refer to formula 6 in the paper)

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

When you want to maximize it in the codebase, you can simply minimize a Binary Cross Entropy loss, i.e.,

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Here if human operators annotate , otherwise .

If and you use CPL with behaviour cloning loss as the constraint (see Appendix B of the paper), then , and Pytorch can achieve (2) with 3 lines of code,

**>>> m** **=** **nn.Sigmoid()**

**>>> criterion** **=** **nn.BCELoss()**

**>>> loss** **=** **criterion(m(A1-lambda\*A2), y)**

But if you have as the constraint, may not be equal to . The nn.BCELoss does not support this situation. You must write your custom loss function. You can rewrite (1) as

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

A problem of (3) is that may be too large and lead to numerical instability. You can set:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

And (3) becomes:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Take the logarithm of both sides, then:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Due to symmetry,

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

You can substitute (6) and (7) back to (2) to get your custom BCE loss. biased\_bce\_with\_logits() in research/algs/cpl.py takes the same implementation.