Privacy Proofs for OpenDP: Bounded Sum with Unknown n

Sílvia Casacuberta

Summer 2021

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the transformation `make_bounded_sum_n` implementing bounded sum unknown n. This is defined in lines 53-68 of the file `sum.rs` in the Git repository[1] (`https://github.com/opendp/opendp/blob/b936c74223b4e319698fa51837b5f8f40f3126d3/rust/opendp/src/trans/sum.rs#L53-L68`).

```rust
pub fn make_bounded_sum<MI, T>(
    lower: T, upper: T
) -> Fallible<Transformation<VectorDomain<IntervalDomain<T>>, AllDomain<T>, MI, AbsoluteDistance<T>>>
    where MI: BoundedSumConstant<T> + DatasetMetric,
          T: DistanceConstant + Sub<Output=T>,
          for <'a> T: Sum<&'a T> {

    Ok(Transformation::new(
        VectorDomain::new(IntervalDomain::new(
            Bound::Included(lower.clone()), Bound::Included(upper.clone()))?),
        AllDomain::new(),
        Function::new(|arg: &Vec<T>| arg.iter().sum()),
        MI::default(),
        AbsoluteDistance::default(),
        StabilityRelation::new_from_constant(MI::get_stability_constant(lower, upper)?)))
}
```

[1] As of July 1, 2021.

Update: after conversations with Mike about the possible overflow problems in Bounded Sum, the Rust code has been updated. The new version can be found at `https://github.com/opendp/opendp/blob/53e8d67b8dde4425930fb8bc397c126cd4f18370/rust/opendp/src/trans/sum.rs#L18-L33`. However, we also keep the code snippet above because this change is still a pull request and is still not part of the OpenDP library.[2] The proof below now refers to this most updated version.

```rust
pub fn make_bounded_sum<T>(
    lower: T, upper: T
) -> Fallible<Transformation<VectorDomain<IntervalDomain<T>>, AllDomain<T>, SymmetricDistance, AbsoluteDistance<T>>>
    where T: DistanceConstant<IntDistance> + Sub<Output=T> + Abs + SaturatingAdd + Zero,
          IntDistance: InfCast<T> {

    Ok(Transformation::new(
        VectorDomain::new(IntervalDomain::new(
            Bound::Included(lower.clone()), Bound::Included(upper.clone()))?),
        AllDomain::new(),
        Function::new(|arg: &Vec<T>| arg.iter().fold(T::zero(), |sum, v| sum.saturating_add(v))),
        SymmetricDistance::default(),
        AbsoluteDistance::default(),
        StabilityRelation::new_from_constant(max(lower.abs(), upper.abs())
            .ok_or_else(|| err!(InvalidDistance, "lower and upper must be comparable"))?)))
}
```

1.2 Pseudocode in Python

We present a simplified Python-like pseudocode of the Rust implementation below. The necessary definitions for the pseudocode can be found at "List of definitions used in the pseudocode".

Preconditions

To ensure the correctness of the output, we require the following preconditions:

- **User-specified types:**
  - Type `T` must implement `DistanceConstant(IntDistance)`, `TotalOrd`,[3] `Abs`, `Sub(Output=T)`, `SaturatingAdd`, and `Zero`.
  - `IntDistance` must have trait `InfCast(T)`. **Question:** Same question that Connor asked in `make_count` – this is not needed for the proof.

Postconditions

- Either a valid `Transformation` is returned or an error is returned.

[2] As of July 20.

[3] For now, the OpenDP library only implements `PartialOrd`, but `TotalOrd` will soon be implemented. Then, `TotalOrd` will be redundant, since the trait `TotalOrd` is part of the trait `DistanceConstant`.

```
1  def MakeBoundedSum(L: T, U: T):
2      input_domain = VectorDomain(IntervalDomain(L, U))
3      output_domain = AllDomain(T)
4      input_metric = SymmetricDistance()
5      output_metric = AbsoluteDistance(T)
6
7      def Relation(d_in: u32, d_out: T) -> bool:
8          return d_out >= inf_cast(d_in, T) * max(abs(U), abs(L))
9
10     def function(data: Vec[T]) -> T:
11         result:T = 0
12         for i in data:
13             result = saturating_add(result, i)
14         return result
15
16     return Transformation(input_domain, output_domain, function,
       input_metric, output_metric, stability_relation = Relation)
```

2 Proof

2.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters* `(L, U)` *to* `MakeBoundedSum`*, the transformation returned by* `MakeBoundedSum` *has the following properties:*

1. (Appropriate output domain). *For every vector v in the input domain,* `function`(v) *is in the output domain.*

2. (Domain-metric compatibility). *The domain* `input_domain` *matches one of the possible domains listed in the definition of* `input_metric`*, and likewise* `output_domain` *matches one of the possible domains listed in the definition of* `output_metric`*.*

3. (Stability guarantee). *For every pair of elements v, w in* `input_domain` *and for every pair* (`d_in`, `d_out`)*, where* `d_in` *is of the associated type for* `input_metric` *and* `d_out` *is the associated type for* `output_metric`*, if v, w are* `d_in`*-close under* `input_metric` *and* `Relation`(`d_in`, `d_out`) $=$ `True`*, then* `function`(v)*,* `function`(v) *are* `d_out`*-close under* `output_metric`*.*

Proof. **(Appropriate output domain).** In the case of `MakeBoundedSum`, this corresponds to showing that for every vector v in `VectorDomain(IntervalDomain(L, U))`, where `L` and `U` have type `T`, the element `function(v)` belongs to `AllDomain(T)`. The type signature of `function` as defined in line 10 automatically enforces that `function(v)` has type `T`. Since the Rust code successfully compiles, by the type signature the appropriate output domain property must hold. Otherwise, the code will raise an exception for incorrect input type. It is also necessary to check that `function(v)` is contained within the interval `[get_min_value(T), get_max_value(T)]`. This is enforced by the use of the function `saturating_add` in line 13, as described in "List of definitions used in the pseudocode".

If the sum of all the vector elements in `data` is greater than `get_max_value(T)`, then `result` will be equal to `get_max_value(T)`. If the sum of all the vector elements in `data` is less than `get_min_value(T)`, then `result` will be equal to `get_min_value(T)`. Otherwise, `result` will be equal to the sum of all the vector elements in `data`, and it will be contained within the interval `[get_min_value(T), get_max_value(T)]`. Therefore, `function(v)` is guaranteed to be in `output_domain` in all cases.

(Domain-metric compatibility). For `MakeBoundedSum`, this corresponds to showing that `VectorDomain(IntervalDomain (L, U))` is compatible with symmetric distance, and that `AllDomain(T)` is compatible with absolute distance. Both follow directly from the definition of symmetric distance and absolute distance, as stated in "List of definitions used in the pseudocode", along with the *appropriate output domain property* shown above, which ensures that `output_domain` is indeed `AllDomain(T)`.

(Stability guarantee). Throughout the stability guarantee proof, we can assume that $\texttt{function}(v)$ and $\texttt{function}(w)$ are in the correct output domain, by the appropriate output domain property shown above.

Since by assumption $\texttt{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, by the `MakeBoundedSum` stability relation (as defined in line 7 in the pseudocode), we have that $\texttt{d\_out} \geq \texttt{d\_in} \cdot \max(|\texttt{U}|, |\texttt{L}|)$. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v) \Delta \text{MultiSet}(w)| \leq \texttt{d\_in}$.

Further, applying the histogram notation,[4] it follows that

$$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in}.$$

We want to show that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \max(|\texttt{U}|, |\texttt{L}|).$$

This would imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w) \cdot \max(|\texttt{U}|, |\texttt{L}|) \leq \texttt{d\_in} \cdot \max(|\texttt{U}|, |\texttt{L}|),$$

and by the stability relation this will imply that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \texttt{d\_out},$$

as we want to see. □

Let u denote the vector formed by all the elements of v and w *without multiplicities* (i.e., u contains exactly once each of the elements in $\text{MultiSet}(v) \cup \text{MultiSet}(w)$, in any order). Let u_i denote the i-th element of u, and similarly for v and w, and let m denote the length of u. Then, by definition,

$$d_{Sym}(v, w) = \sum_z \Big|h_v(z) - h_w(z)\Big| = \sum_i \Big|h_v(u_i) - h_w(u_i)\Big|;$$

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) = |\texttt{function}(v) - \texttt{function}(w)| \leq \Big|\sum_i v_i - \sum_i w_i\Big| =$$

$$= \Big|\sum_i u_i \cdot h_v(u_i) - \sum_i u_i \cdot h_w(u_i)\Big| = \Big|\sum_i u_i \cdot (h_v(u_i) - h_w(u_i))\Big|.$$

[4] See *A Programming Framework for OpenDP*, footnote 1 in page 3. Note that there is a bijection between multisets and histograms, which is why the proof can be carried out with either notion. For further details, please consult `https://www.overleaf.com/project/60d214e390b337703d200982`.

Note that we have the inequality $|\texttt{function}(v) - \texttt{function}(w)| \leq |\sum_i v_i - \sum_i w_i|$ above (instead of an equality) due to the definition of `saturating_add`. The equality case holds whenever $\sum_i v_i \in$ `[get_min_value(T), get_max_value(T)]` and $\sum_i w_i \in$ `[get_min_value(T), get_max_value(T)]`. In any of the possible cases where $\sum_i v_i >$ `get_max_value(T)` or $\sum_i v_i <$ `get_min_value(T)` and $\sum_i w_i >$ `get_max_value(T)` or $\sum_i w_i <$ `get_min_value(T)`, the difference $|\sum_i v_i - \sum_i w_i|$ will always upper bound the value $|\texttt{function}(v) - \texttt{function}(w)|$, and hence it is sufficient to carry our proof by only considering the quantity $|\sum_i v_i - \sum_i w_i|$.

(silvia) Might be easier to just state this in terms of absolute clamp and refer to that transformation proof.

By the definition of absolute distance and symmetric distance, and by applying the triangle inequality, we obtain:

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \left| \sum_i u_i \cdot (h_v(u_i) - h_w(u_i)) \right| \leq |u_i| \cdot \sum_i |h_v(u_i) - h_w(u_i)|.$$

By the appropriate output domain property $u_i \in$ `[L, U]` $\forall i$ it follows that $|u_i| \leq \max(|\texttt{U}|, |\texttt{L}|)$ for all i. Hence,

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq |u_i| \cdot \sum_i |h_v(u_i) - h_w(u_i)| \leq$$

$$\leq \max(|\texttt{U}|, |\texttt{L}|) \cdot \sum_i |h_v(u_i) - h_w(u_i)| \leq \max(|\texttt{U}|, |\texttt{L}|) \cdot d_{Sym}(v, w).$$

Lastly, since by assumption v and w are `d_in`-close, by the defined `Relation(d_in, d_out)` (line 10 in the pseudocode) it follows that

$$d_{Abs}(\texttt{function}(v), \texttt{function}(w)) \leq \max(|\texttt{U}|, |\texttt{L}|) \cdot d_{Sym}(v, w) \leq$$

$$\leq \max(|\texttt{U}|, |\texttt{L}|) \cdot \texttt{d\_in} \leq \texttt{d\_out},$$

as we wanted to show.

(silvia) Flag: need to account for rounding errors in the stability relation given the non-closure of float addition, as discussed on the week of August 2nd. We are figuring this out.