Privacy Proofs for OpenDP: Clamping

Sílvia Casacuberta

June 2021

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_clamp_vec` function implementing the clamping function. This is defined in lines 25-38 of the file `manipulation.rs` in the Git repository[1] (https://github.com/opendp/opendp/blob/58feb788ec78ce739caaf3cad8471c79fd5e7132/rust/opendp/src/trans/manipulation.rs#L25-L38).

```rust
pub fn make_clamp_vec<M, T>(lower: T, upper: T) -> Fallible<Transformation<VectorDomain<AllDomain<T>>, VectorDomain<IntervalDomain<T>>, M, M>>
    where M: Metric,
          T: 'static + Clone + PartialOrd,
          M::Distance: DistanceConstant + One {
    if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
    Ok(Transformation::new(
        VectorDomain::new_all(),
        VectorDomain::new(IntervalDomain::new(Bound::Included(lower.clone()), Bound::Included(upper.clone()))),
        Function::new(move |arg: &Vec<T>| arg.iter().map(|e| clamp(&lower, &upper, e)).collect()),
        M::default(),
        M::default(),
        // clamping has a c-stability of one, as well as a lipschitz constant of one
        StabilityRelation::new_from_constant(M::Distance::one())))
}
```

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".

(silvia) We could generalize the input domain below from float to a any general (unspecified) type that admits total ordering, and then add the corresponding precondition to assert that the type `T` for L and U has the trait `TotalOrd`. Note that partial ordering is not enough, and we might not be able to clamp in the domain. However, TotalOrd is still not implemented in the OpenDP library. For now, float is enough.

[1] As of June 16, 2021. Since then, the code has been updated to include a more general clampable domain, which is not yet finished.

Preconditions

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

(mike) The input metric can just be hardcoded to SymmetricDistance. This means there's no generic MI, so the precondition can be dropped (silvia) Same for output metric, no? Also, wouldn't it be safer not to hardcode it in case more metrics are added to the library in the future? (mike) After recent talks, I think we're thinking this should go in the preconditions. We can expand the proof in the future if we get more metrics. (silvia) Agreed

(mike) The type T can be any type that implements PartialOrd- for example floats or ints

(silvia) Fixed, but I think it should be "implements TotalOrd", no? (mike) Yes n I'm still seeing references to float throughout this document though. You should be able to just refer to T, where T is qualified by the trait bounds. (silvia) Agreed

(silvia) The domain/metric preconditions will be added once Prof. Vadhan agrees

- The type `T` must implement TotalOrd – for example, `floats` or `ints`.

```
def MakeClamp(L: T, U: T, metric):
    if L > U: raise Exception('Invalid parameters')
    input_domain = VectorDomain(AllDomain(T))
    output_domain = VectorDomain(IntervalDomain(L, U))
    input_metric = SymmetricDistance()
    output_metric = SymmetricDistance()
    def stability_relation(d_in: u32, d_out: u32) -> bool:
        return d_out >= d_in*1

    def function(data: Vec(T)) -> Vec(T):
        def clamp(x: T) -> T:
            return max(min(x, U), L)
        return list(map(clamp, data))
    return Transformation(input_domain, output_domain, function,
    input_metric, output_metric, stability_relation)
```

Conditions as specified in the pseudocode

(silvia) Redundant section with the pseudocode, but perhaps useful.

- Input domain: domain of all vectors of elements of type `T`.[2]
- Output domain: domain of all vectors of elements in `IntervalDomain(L, U)`, where `L` and `U` are of type `T`and stand for *lower bound* and *upper bound*, respectively.[3]
- Function: given v of type `Vec(T)`, it returns a list where each vector entry v_i has type `float` and is equal to `clamp`(v_i), as defined in line 11 of the pseudocode (Section 1.2).
- Input metric: symmetric distance.[4]

[2] In the future, this will be changed to the domain of all vectors of elements from an arbitrary data domain implementing total ordering (more concretely, for a generic type `T` that must have traits `static`, `Clone`, and `TotalOrd`). However, this is not yet implemented in the OpenDP library.

[3] The compiler infers `float` from the context, which is why it is omitted.

[4] As of June 24, SubstituteDistance is no longer a metric part of the OpenDP library.

- Output metric: same as input metric (which have to be consistent – this is checked by the preconditions).
- Stability relation: for `d_in`, `d_out` of type `u32`, the relation returns `True` if and only if `d_in` $\leq$ `d_out`. In particular, clamping has stability parameter $c = 1$.

More concisely, the stability relation can be stated as:[5] (silvia) Perhaps more concise: `d_in, d_out ∈ AllDomain<u32>`

$$\text{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \begin{cases} \texttt{True} & \text{if } \texttt{d\_out} \geq \texttt{d\_in} \text{ for } \texttt{d\_in}, \texttt{d\_out} \text{ of type } \texttt{u32} \\ \texttt{False} & \text{otherwise} \end{cases} \tag{1}$$

2 Proof

2.1 Symmetric Distance

Theorem 1. *For every setting of the input parameters $(L, U,$ metric$)$ to `MakeClamp` such that the given preconditions hold, the transformation returned by `MakeClamp` has the following properties:*

1. (Appropriate output domain). *For every vector v in the input domain, `function`(v) is in the output domain.*
2. (Stability guarantee). *For every input v, w of type `Vec(float)` and for every pair (`d_in`, `d_out`) (appropriately quantified), if v, w are d_{in}-close under the symmetric distance metric and* Relation(`d_in`, `d_out`) = `True`*, then `function`(v), `function`(w) are d_{out}-close under the symmetric distance metric.*

Proof. **(Appropriate output domain).** In the case of `MakeClamp`, this corresponds to showing that for every vector v of type `Vec(T)`, `function`(v) is an element of `VectorDomain (IntervalDomain(L, U))`. For that, we need to show two things: first, that the vector entries of `function`(v) have type `T`. Second, that they belong to the interval $[L, U]$.

That `function`(v) has type `Vec(T)` follows from the assumption that v is in the input domain, the precondition requirement that `T`has type `T`, and the type signature of `function` in line 10 of the pseudocode (Section 1.2), which takes in an element of type `Vec(T)` and returns an element of type `Vec(T)`. If the Rust code compiles correctly, then the type correctness follows from the definition of the type signature enforced by Rust. Otherwise, the code raises an exception for incorrect input type. Secondly, we need to show that the vector entries belong to the interval $[L, U]$. This follows from the definition of `clamp` in line 11. According to line 11 in the pseudocode, there are 3 possible cases to consider:

1. `x` $> U$: then `clamp` returns U.
2. `x` $\in [L, U]$: then `clamp` returns `x`.
3. `x` $< L$: then `clamp` returns L.

[5] See page 14 of *A Programming Framework for OpenDP* for more examples. Note that in the Rust implementation the metrics $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are *not* inputs to the relation, and they are instead fixed as attributes of the transformation.

In all three cases, the returned value of type `T` is contained in the interval $[L, U]$. Hence by the returned vector as defined in line 13 of the pseudocode, `function`(v) is an element of the output domain `VectorDomain(IntervalDomain(L, U))`.

Lastly, the necessary condition that $L \leq U$ is checked in line 2 of the pseudocode, hence correctness is guaranteed if no exception is raised. Both L and U have type `T` by their precondition requirement. Both the definition of `IntervalDomain` and that of the `clamp` function (line 11 in the pseudocode, which uses the `min` and `max` functions) require that the type of L, U, and of each vector entry in v admits a total ordering. In the case of `T`, this holds by the preconditions. (mike) floats don't have a total ordering. It only holds for non-null floats!

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

Since by assumption Relation(`d_in`, `d_out`) = `True`, by the `MakeClamp` stability relation (as defined in Equation (1)), we have that `d_in` $\leq$ `d_out`. Moreover, v, w are assumed to be `d_in`-close. By the definition of the symmetric difference metric, this is equivalent to stating that $d_{Sym}(v, w) = |\text{MultiSet}(v) \Delta \text{MultiSet}(w)| \leq$ `d_in`.

Further, applying the histogram notation,[6] it follows that

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

We now consider MultiSet(`function`(v)) and MultiSet(`function`(w)). For each element $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$, where z has type `float`, if $z \in \text{MultiSet}(v) \Delta \text{MultiSet}(w)$, we will assume wlog that $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$. We consider the following cases:

1. $z > U$ or $z < L$: then, in the former case, `clamp`$(z) = U$. First consider the case when $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets. Then, $|h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| = 0$ because we have both $h_{\texttt{function}(v)}(z) = 0$ and $h_{\texttt{function}(w)}(z) = 0$. Thus the sum

$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$$

remains invariant, because the quantity $|h_v(z) - h_w(z)|$ is added to $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$, given that `clamp`$(z) = U$.

Suppose z has multiplicity $k_v \geq 0$ in MultiSet(v) and multiplicity $k_w \geq 0$ in MultiSet(w), where $k_v \neq k_w$. After considering z, the value $h_{\texttt{function}(v)}(U)$ becomes $h_{\texttt{function}(v)}(U) + k_v$, and $h_{\texttt{function}(w)}(U)$ becomes $h_{\texttt{function}(w)}(U) + k_w$. Hence the quantity $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$ increases by at most $|h_v(z) - h_w(z)|$, since, by the triangle inequality,

$$\begin{aligned} &|(h_{\texttt{function}(v)}(U) + k_v) - (h_{\texttt{function}(w)}(U) + k_w)| \leq \\ &\leq |h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)| + |k_v - k_w| = \\ &= |h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)| + |h_v(z) - h_w(z)|. \end{aligned}$$

[6] 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.

The same argument applies whenever $z < L$.

(silvia) The first subcase discussed here, i.e., when $k_v = k_w$, is also proven by the triangle inequality expression above, but it seemed clean to separate the case where the total sum remains invariant.

2. $z \in (L, U)$: then, $\texttt{clamp}(z) = z$. Since $h_v(z) = h_{\texttt{function}(v)}(z)$ and $h_v(w) = h_{\texttt{function}(w)}(z)$, it follows that $|h_v(z) - h_w(z)| = |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$. Hence the histogram count, i.e., the quantity

$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)|$$

remains invariant.

3. $z = U$ or $z = L$: then, in the former case, $\texttt{clamp}(z) = U$. If $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$ with the same multiplicity in both multisets, then the histogram count remains invariant under the addition of element z. Otherwise, if $z \in \text{MultiSet}(v) \setminus \text{MultiSet}(w)$, or if z is in their union but with different multiplicity, then element z can increase the quantity $|h_{\texttt{function}(v)}(U) - h_{\texttt{function}(w)}(U)|$ by at most $|h_v(z) - h_w(z)|$, following the same reasoning with the triangle inequality as in case 2.

   The same argument applies whenever $z = L$.

By aggregating the three cases above, we conclude that

$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)|.$$

By the initial assumptions, we recall that $\texttt{d\_in} \leq \texttt{d\_out}$, and that v, w are $\texttt{d\_in}$-close. Then,

$$\sum_z |h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)| \leq \sum_z |h_v(z) - h_w(z)| \leq \texttt{d\_in} \leq \texttt{d\_out}.$$

Therefore,

$$|\text{MultiSet}(\texttt{function}(v)) \Delta \text{MultiSet}(\texttt{function}(w))| \leq \texttt{d\_out},$$

as we wanted to show. □

(silvia) Maybe add domain of z below the sum?