Privacy Proofs for OpenDP: Clamping

Sílvia Casacuberta

Summer 2021

Contents

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`).

```
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".

[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:

- **User-specified types:**
  - Type `T` must have trait `TotalOrd`.[2]

Postconditions

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

```
def MakeClamp(L: T, U: T):
    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 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)
```

2 Proof

The necessary definitions for the proof can be found at "List of definitions used in the proofs".

2.1 Symmetric Distance

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

1. (Appropriate output domain). *For every element v in* `input_domain`*,* `function`(v) *is in* `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`*.*

[2]For now, the OpenDP library only implements `PartialOrd`, but `TotalOrd` will soon be implemented.

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 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`(w) *are d_{out}-close under* `output_metric`*.*

Proof. **(Appropriate output domain).** In the case of `MakeClamp`, this corresponds to showing that for every vector v of elements of type `T`, `function`(v) is a vector of elements of type `T` which are contained in the interval `[L, U]`. For that, we need to show two things: first, that `function(v)` has type `Vec(T)`. Second, that they belong to the interval `[L, U]`.

Firstly, that `function`(v) has type `Vec(T)` follows from the assumption that element v is in `input_domain` and from the type signature of `function` in line 12 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 13. According to line 13 in the pseudocode, there are 3 possible cases to consider:

1. `x` $>$ `U`: then `clamp(x)` returns `U`.

2. `x` $\in$ `[L, U]`: then `clamp(x)` returns `x`.

3. `x` $<$ `L`: then `clamp(x)` returns `L`.

In all three cases, the returned value of type `T` is contained in the interval `[L, U]`. Hence, the vector `function`(v) returned in line 15 of the pseudocode is an element of `output_domain`.

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 13 in the pseudocode, which uses the `min` and `max` functions) require that `T` implement `TotalOrd`, which holds by the preconditions. In the case of `T`, this holds by the preconditions.

(Domain-metric compatibility). For `MakeClamp`, both cases correspond to showing that `VectorDomain(T)` is compatible with symmetric distance. This follows directly from the definition of symmetric distance, as stated in "List of definitions used in the proofs".

(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 line 9 in the pseudocode), 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,[3] 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($\texttt{function}(v)$) and MultiSet($\texttt{function}(w)$). For each element $z \in \text{MultiSet}(v) \cup \text{MultiSet}(w)$, where z has type `T`, 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. $z > \texttt{U}$ or $z < \texttt{L}$: then, in the former case, $\texttt{clamp}(z) = \texttt{U}$. 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 $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$, given that $\texttt{clamp}(z) = \texttt{U}$.

   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 $h_{\texttt{function}(v)}(\texttt{U})$ becomes $h_{\texttt{function}(v)}(\texttt{U}) + k_v$, and $h_{\texttt{function}(w)}(\texttt{U})$ becomes $h_{\texttt{function}(w)}(\texttt{U}) + k_w$. Hence the quantity $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$ increases by at most $|h_v(z) - h_w(z)|$, since, by the triangle inequality,

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

   The same argument applies whenever $z < \texttt{L}$.

   (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. $z \in \texttt{(L, U)}$: 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. $z = \texttt{U}$ or $z = \texttt{L}$: then, in the former case, $\texttt{clamp}(z) = \texttt{U}$. 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 $z \in \text{MultiSet}(v) \setminus$

[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.

MultiSet(w), or if z is in their union but with different multiplicity, then element z can increase the quantity $|h_{\texttt{function}(v)}(\texttt{U}) - h_{\texttt{function}(w)}(\texttt{U})|$ 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 $z = \texttt{L}$.

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,

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

as we wanted to show. □

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