```cpp
#include "LossActivation.hpp"

namespace LossActivation {

using namespace std;
```

Avoid using namespace std; prefer explicit std:: to avoid naming conflicts

```cpp
const int threshold = 3;
```

Avoid using global variables

```cpp
template <typename T> SoftmaxCCE<T>::SoftmaxCCE() : loss{}, activation{} {}

template <typename T>
T SoftmaxCCE<T>::compute(Vec2d<T>* inputs, Vec2d<T>* actualY) {
```

Prefer smart pointers to raw pointers

```cpp
    activation.compute(inputs);
    output = activation.getOutput();
    return loss.calculate(output, actualY);
}

template <typename T>
void SoftmaxCCE<T>::backward(Vec2d<T>* dValues, Vec2d<T>* actualY) {
```

Prefer smart pointers to raw pointers

```cpp
    int numSamples = dValues.size();
```

Add comments to document this block of logic

```cpp
    Vec2d<T> maxIdxs;
    if (actualY.size() > threshold) {
        maxIdxs.push_back({});
        for (const auto &row : actualY) {
            T maxVal = row[0];
            T maxIdx = 0;
            for (int i = 1; i < row.size(); i++) {
                if (row[i] > maxVal) {
                    maxVal = row[i];
                    maxIdx = i;
```

```
            }
        }
        maxIdxs[0].push_back(maxIdx);
    }
} else {
    maxIdxs = actualY;
}

dInputs = dValues;

for (int i = 0; i < numSamples; i++) {
    T y = maxIdxs[0][i];
    dInputs[i][y] -= 1;
}

dInputs = dInputs / (T)numSamples;
```

Avoid C-style casts, use static_cast instead

```
}

template <typename T> Vec2d<T> SoftmaxCCE<T>::getOutput() { return output; }

template <typename T> Vec2d<T> SoftmaxCCE<T>::getDInputs() { return dInputs; }
```

getDInputs and getOutput should return by reference instead of by value in order to avoid unnecessary copying

```
// Explicit instantiations
template class SoftmaxCCE<double>;
template class SoftmaxCCE<float>;

} // namespace LossActivation
```