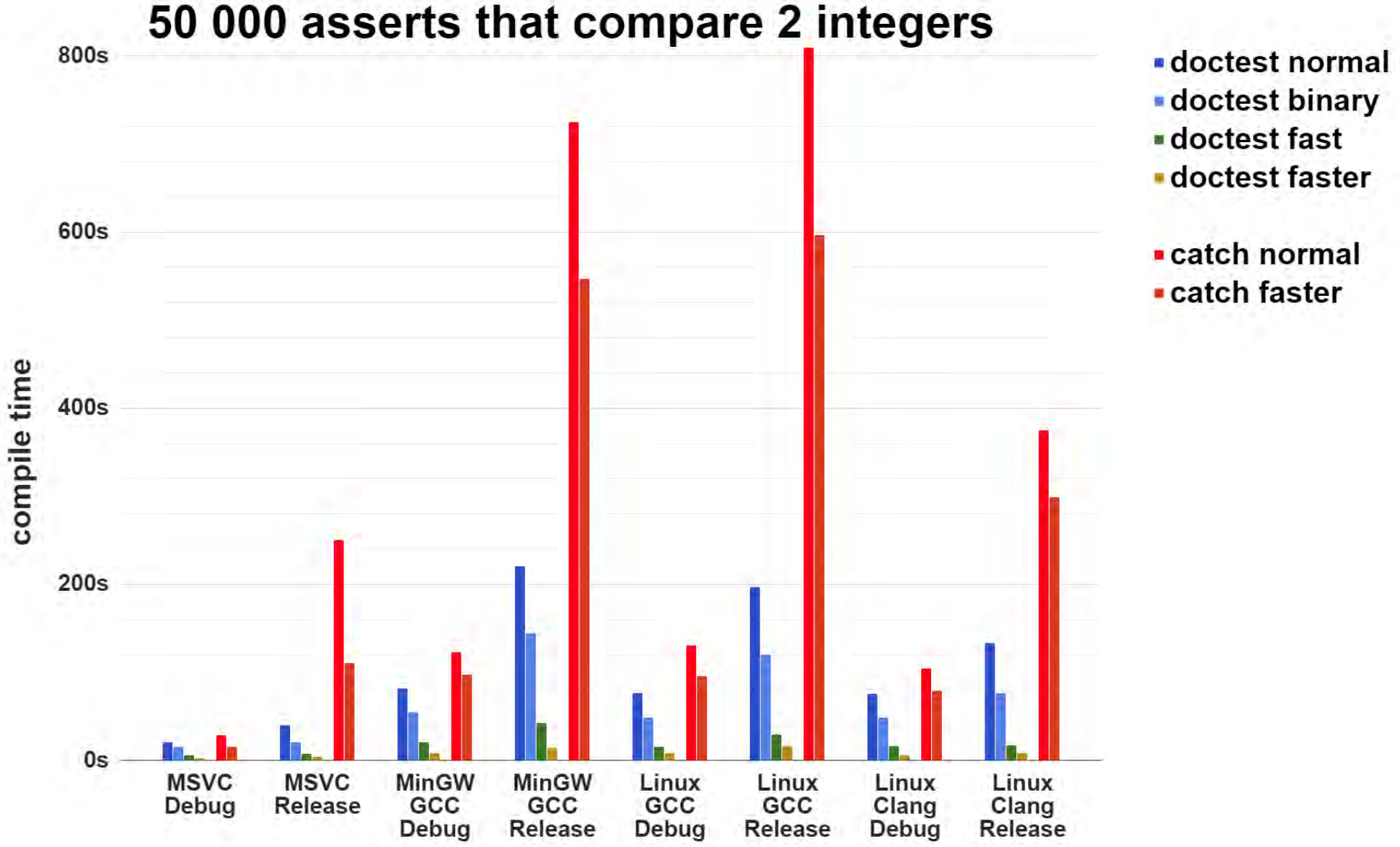
Compile times - assert macros



doctest 1.0 - CHECK(a == b);

```
do {
    Result res;
    bool threw = false;
    try {
        res = ExpressionDecomposer() << a == b;
    } catch(...) { threw = true; }
    if(res || GCS()->success) {
        do {
            if(!GCS()->hasLoggedCurrentTestStart) {
                logTestStart(GCS()->currentTest->m_name,
                             GCS()->currentTest->m_file,
                             GCS()->currentTest->m_line);
                GCS()->hasLoggedCurrentTestStart = true;
            }
        } while(false);
        logAssert(res.m_passed, res.m_decomposition.c_str(),
                  threw, "a == b", "CHECK", "a.cpp", 76);
    }
    GCS()->numAssertionsForCurrentTestcase++;
    if(res) {
        addFailedAssert("CHECK");
        BREAK_INTO_DEBUGGER();
    }
} while(doctest::always_false());
```

doctest 1.1 asserts


```
// CHECK(a == b)                    << THIS EXPANDS TO:
do {
    ResultBuilder rb("CHECK", "a.cpp", 76, "a == b");
    try {
        rb.setResult(ExpressionDecomposer() << a == b);
    } catch(...) { rb.exceptionOccurred(); }
    if(rb.log()) BREAK_INTO_DEBUGGER();
} while((void)0, 0)
```

```
// FAST_CHECK_EQ(a, b)              << THIS EXPANDS TO:
do {
    int res = fast_binary_assert<equality>("FAST_CHECK_EQ", "a.cpp",
                                           76, "a", "b", a, b);
    if(res) BREAK_INTO_DEBUGGER();
} while((void)0, 0)
```

```
// FAST_CHECK_EQ(a, b)      with #define DOCTEST_CONFIG_SUPER_FAST_ASSERTS
fast_binary_assert<equality>("FAST_CHECK_EQ", "a.cpp", 76, "a", "b", a, b);
```

Compile times - assert macros

50 000 asserts spread in 500 test cases compile for:

- normal: 20-220 secs (roughly 30-75% faster than Catch)
- fastest: 3-16 secs 10-15 times faster than the normal

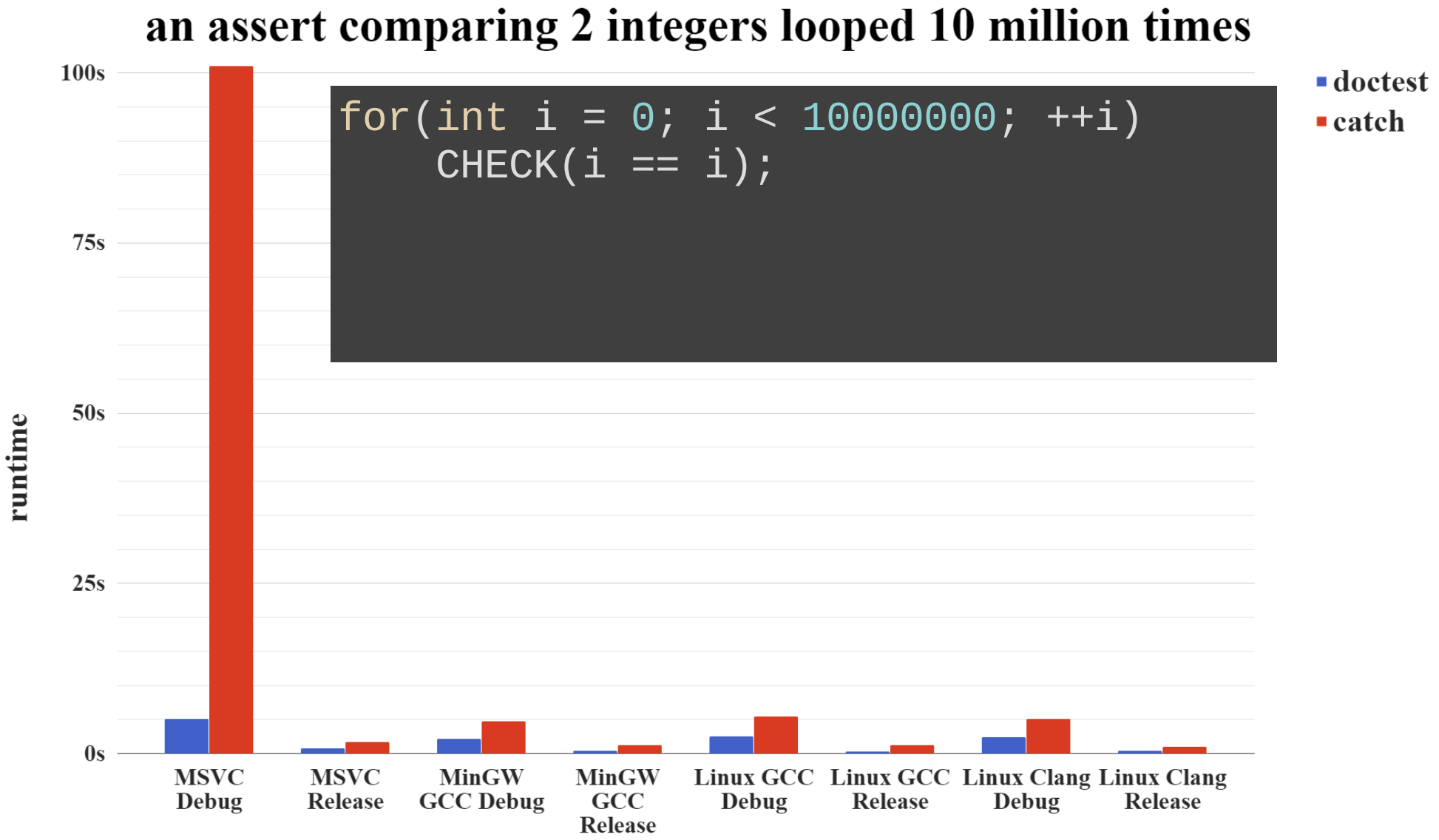
extensive use of **__declspec(noinline)** / **__attribute__((noinline))**

The benchmarks were done on 2017.09.10 with versions:

- doctest: 1.2.2 (released on 2017.09.05)
- Catch: 2.0.0-develop.3 (released on 2017.08.30)

https://github.com/onqtam/doctest/blob/master/doc/markdown/benchmarks.md

Runtime performance



Runtime performance

doctest 1.2 is more than 30 times faster than doctest 1.1

(talking only about the common case where all tests pass)

- constructs strings only if asserts fail (big gains)
- small string optimization for doctest::String (huge gains)
- move semantics (doesn't matter if nothing fails though...)
- not accessing local statics on the hot path (<1% gain)

CppCon 2016: Nicholas Ormrod “The strange details of std::string at Facebook"

Mixing tests and production code

When developing end products (not libraries for developers):

- just mix code and tests
- supply your own main() with **DOCTEST_CONFIG_IMPLEMENT**
- build the final release version with **DOCTEST_CONFIG_DISABLE**

OR ship the tests in the binary:

- disabled by default by setting the **no-run** option to **true**

```
#define DOCTEST_CONFIG_IMPLEMENT
#include <doctest.h>

// later in main()
context.setOption("no-run", true); // don't run by default
context.applyCommandLine(argc, argv); // parse command line
```

Tests in header-only libraries

```cpp
// fact.h

#pragma once
inline int fact(int n) {
    return n <= 1 ? n : fact(n - 1) * n;
}

#ifdef FACT_WITH_TESTS
#ifndef DOCTEST_LIBRARY_INCLUDED
#include <doctest.h>
#endif // DOCTEST_LIBRARY_INCLUDED

TEST_CASE("[fact] testing fact") {
    CHECK(fact(0) == 1);
    CHECK(fact(1) == 1);
}
#endif
```

```cpp
// fact_usage.cpp

#include "fact.h"
```

```cpp
// fact_tests.cpp

//#define DOCTEST_CONFIG_IMPLEMENT
#define FACT_WITH_TESTS
#include "fact.h"
```

```cpp
// fact_tests.cpp

//#define DOCTEST_CONFIG_IMPLEMENT
#include <doctest/doctest.h>
#define FACT_WITH_TESTS
#include "fact.h"
```

add a tag in your test case names if shipping a library

```
--test-case-exclude=*[fact]*
```

or use a test suite

Tests in compiled libraries

Many binaries (shared objects and executables) can share the same test runner - a single test case registry

```
#define DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
```

There are issues with self-registering test cases in static libraries which are common to all testing frameworks - for more information visit this link from the FAQ:

https://github.com/onqtam/doctest/blob/master/doc/markdown/faq.md#why-are-my-tests-in-a-static-library-not-getting-registered

Getting the most out of the framework

```
// doctest_proxy.h - use this header instead of doctest.h

#define DOCTEST_CONFIG_NO_SHORT_MACRO_NAMES // prefixed macros
#define DOCTEST_CONFIG_SUPER_FAST_ASSERTS   // speed junkies
#include <doctest.h>

#define test_case       DOCTEST_TEST_CASE
#define subcase         DOCTEST_SUBCASE
#define test_suite      DOCTEST_TEST_SUITE
#define check_throws    DOCTEST_CHECK_THROWS
#define check_throws_as DOCTEST_CHECK_THROWS_AS
#define check_nothrow   DOCTEST_CHECK_NOTHROW

#define check_eq        DOCTEST_FAST_CHECK_EQ
#define check_ne        DOCTEST_FAST_CHECK_NE
#define check_gt        DOCTEST_FAST_CHECK_GT
#define check_lt        DOCTEST_FAST_CHECK_LT
#define check           DOCTEST_FAST_CHECK_UNARY
#define check_not       DOCTEST_FAST_CHECK_UNARY_FALSE
```

Where most of the effort went

- Familiarizing myself with testing and other frameworks
- Not dragging any headers
- The 330+ different CI builds (and the .travis.yml file...)
- The usual suspects (problematic warnings):
  - -Winline - especially with gcc 4.7
  - -Wstrict-overflow (level 5 - without real file/line in release)
  - -Weffc++
- Took me 3-4 days to track down and workaround a valgrind error - only with g++4.8 in Release
- My first unique compiler bug report in GCC (sanitizer related)
- Hit MANY other toolchain problems

Roadmap

- reporters - xml, xUnit, compact and user defined
- logging levels
- test execution in separate processes - UNIX fork()
- death tests
- symbolizer for stack traces
- generators for data-driven testing
- matchers
- more command line options
- IDE integration (VS (MSTest), XCode...)
- thread safe assertions / subcases / logging
- spreading the word about **doctest** - marketing
- and many many other small things!

History of doctest

- **August 2014** - initial concept
- **01.01.2016** - development accelerated
- **22.05.2016** - released **1.0** - focus on compile time of the header
- **21.09.2016** - released **1.1** - compile time of asserts, bug fixes
- **16.05.2017** - released **1.2** - features and runtime performance

A bit late to the party...

Such results would not have been possible without starting from scratch.

A "modest" goal for doctest - make it the de-facto standard for unit testing in C++ (almost as a language feature).

Q&A

- Slides: http://slides.com/onqtam/2017_november_doctest
- Project: https://github.com/onqtam/doctest
- Personal site: http://onqtam.com
- GitHub: https://github.com/onqtam
- Twitter: https://twitter.com/KirilovVik
- E-Mail: vik.kirilov@gmail.com

2017 CPP-Summit

元编程在深度学习上的应用

—— C++模板元神经网络库

陈明辉

群熵金融CTO

```cpp
#include "BPNN.hpp"
int main()
{
    /// 1.
    typedef mtl::BPNN<4, 2, 4, 2, 4, 2, 4> MyNN;
    MyNN bpnn;

    /// 2.
    bpnn.init()
        .set_aberration(0.0001)
        .set_learnrate(0.8)
        .set_sigfunc(mtl::logsig)
        .set_dsigfunc(mtl::dlogsig);

    /// 3.
    MyNN::InMatrix inMx;
    MyNN::OutMatrix outMx;
    MyNN::OutMatrix expectMx;

    /// 4.
    bpnn.train(inMx, outMx, 100);

    /// 5.
    bpnn.simulate(inMx, outMx, expectMx);
}
```

开篇

五步神经网络

2017 CPP-Summit

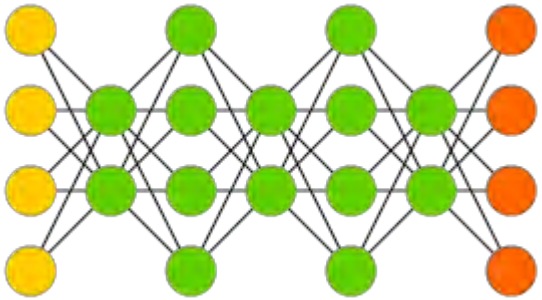
打开深度学习的大门

机器学习、神经网络、深度学习

元编程？

一个“全连接”、“简单”、“深度”神经网络

BPNN<4, 2, 4, 2, 4, 2, 4>



```
BPNN<IN<4>,
     CP<2, 4, 4>,
     CP<4, 2, 2>,
     CP<2, 4, 4>,
     CP<4, 2, 2>,
     CP<2, 4, 4>,
     OUT<4>>
```

neuron：[4], [2], [4], [2], [4], [2], [4]

weight： [4×2], [2×4], [4×2], [2×4], [4×2], [2×4]

threshold： [2], [4], [2], [4], [2], [4]

delta： [2], [4], [2], [4], [2], [4]

预先创建神经元、权值、阈值、Delta四种数据结构

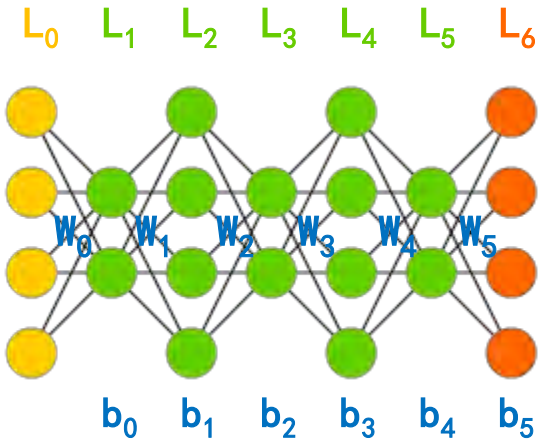

```
tuple<Matrix<double, 1, Layers>...> layers;

typename BPNNType<make_index_sequence<N - 1>, Layers...>::Weights weights;

typename BPNNType<make_index_sequence<N - 1>, Layers...>::Thresholds thresholds;

tuple<Matrix<double, 1, Layers>...> deltas;
```

```cpp
template<typename I, int... Layers> struct BPNNType;

template<std::size_t... I, int... Layers>
struct BPNNType<std::index_sequence<I...>, Layers...>
{
    typedef /// Weights type
    std::tuple<
            Matrix<
                    double,
                    UnpackInts<I, Layers...>::value,
                    UnpackInts<I + 1, Layers...>::value
            >...
    > Weights;

    typedef /// Thresholds type
    std::tuple<
            Matrix<
                    double,
                    1,
                    UnpackInts<I + 1, Layers...>::value
            >...
    > Thresholds;
};
```

权值和阈值的辅助类

展开 I... 的时候 Layers... 被嵌套展开

正向传播



$$L_1 = f(L_0 \times W_0 + b_0)$$
$$L_2 = f(L_1 \times W_1 + b_1)$$
$$L_3 = f(L_2 \times W_2 + b_2)$$
$$L_4 = f(L_3 \times W_3 + b_3)$$
$$L_5 = f(L_4 \times W_4 + b_4)$$
$$L_6 = f(L_5 \times W_5 + b_5)$$

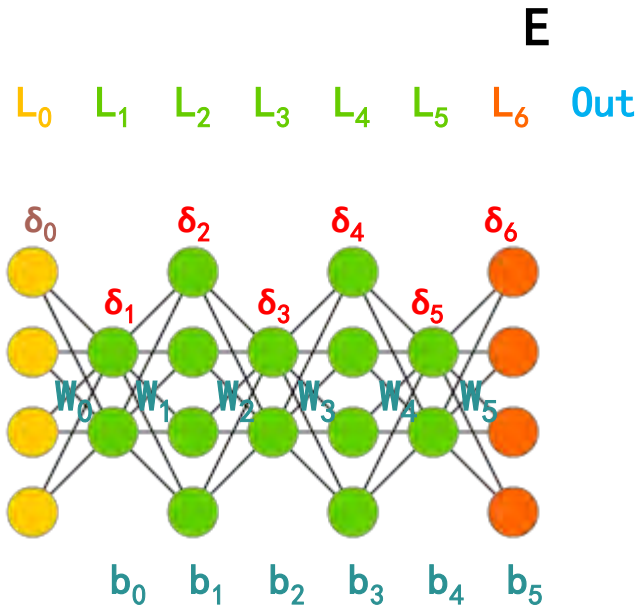
```cpp
using expander = int[];


train(std::make_index_sequence<N - 1>());


template<int... Layers>
template<std::size_t... I>
void BPNN<Layers...>::train(std::index_sequence<I...>)
{
    expander {(forward(std::get<I>(m_layers),
                       std::get<I + 1>(m_layers),
                       std::get<I>(m_weights),
                       std::get<I>(m_thresholds)),
                       0)...};
}


template<int... Layers>
template<class LX, class LY, class W, class T>
void BPNN<Layers...>::forward(LX& layerX, LY& layerY,
                              W& weight, T& threshold)
{
    layerY.multiply(layerX, weight);
    layerY += threshold;
    layerY.foreach(m_sigfunc);
}
```

正向传播

expander

逗号运算符

反向修正



$$E = (Out - L_6)^2 / 2$$
$$\delta_6 = (Out - L_6) \cdot f'(L_6)$$

$$W_5 = W_5 + L_5 \times \delta_6 \cdot \eta$$
$$b_5 = b_5 + \delta_6 \cdot \eta$$
$$\delta_5 = (W_5 \times \delta_6)^T \cdot f'(L_5)$$

$$W_4 = W_4 + L_4 \times \delta_5 \cdot \eta$$
$$b_4 = b_4 + \delta_5 \cdot \eta$$
$$\delta_4 = (W_4 \times \delta_5)^T \cdot f'(L_4)$$

...

$$W_0 = W_0 + L_0 \times \delta_1 \cdot \eta$$
$$b_0 = b_0 + \delta_1 \cdot \eta$$
$$\delta_0 = (W_0 \times \delta_1)^T \cdot f'(L_0)$$

```cpp
template<int... Layers>
template<std::size_t... I>
void BPNN<Layers...>::train(std::index_sequence<I...>)
{
    deltaN.hadamard(m_aberrmx, layerN.foreach(dlogsig));

    expander {(backward(std::get<N - I - 2>(m_layers),
                        std::get<N - I - 2>(m_weights),
                        std::get<N - I - 2>(m_thresholds),
                        std::get<N - I - 2>(m_deltas),
                        std::get<N - I - 1>(m_deltas)),
                        0)...};
}


template<int... Layers>
template<class LX, class W, class T, class DX, class DY>
void BPNN<Layers...>::backward(LX& layerX, W& weight,
                               T& threshold, DX& deltaX, DY& deltaY)
{
    weight.adjustW(layerX, deltaY, m_learnrate);
    threshold.adjustT(deltaY, m_learnrate);

    deltaX.mult_trans(weight, deltaY);
    layerX.foreach(m_dsigfunc);
    deltaX.hadamard(layerX);
}
```

反向修正

expander

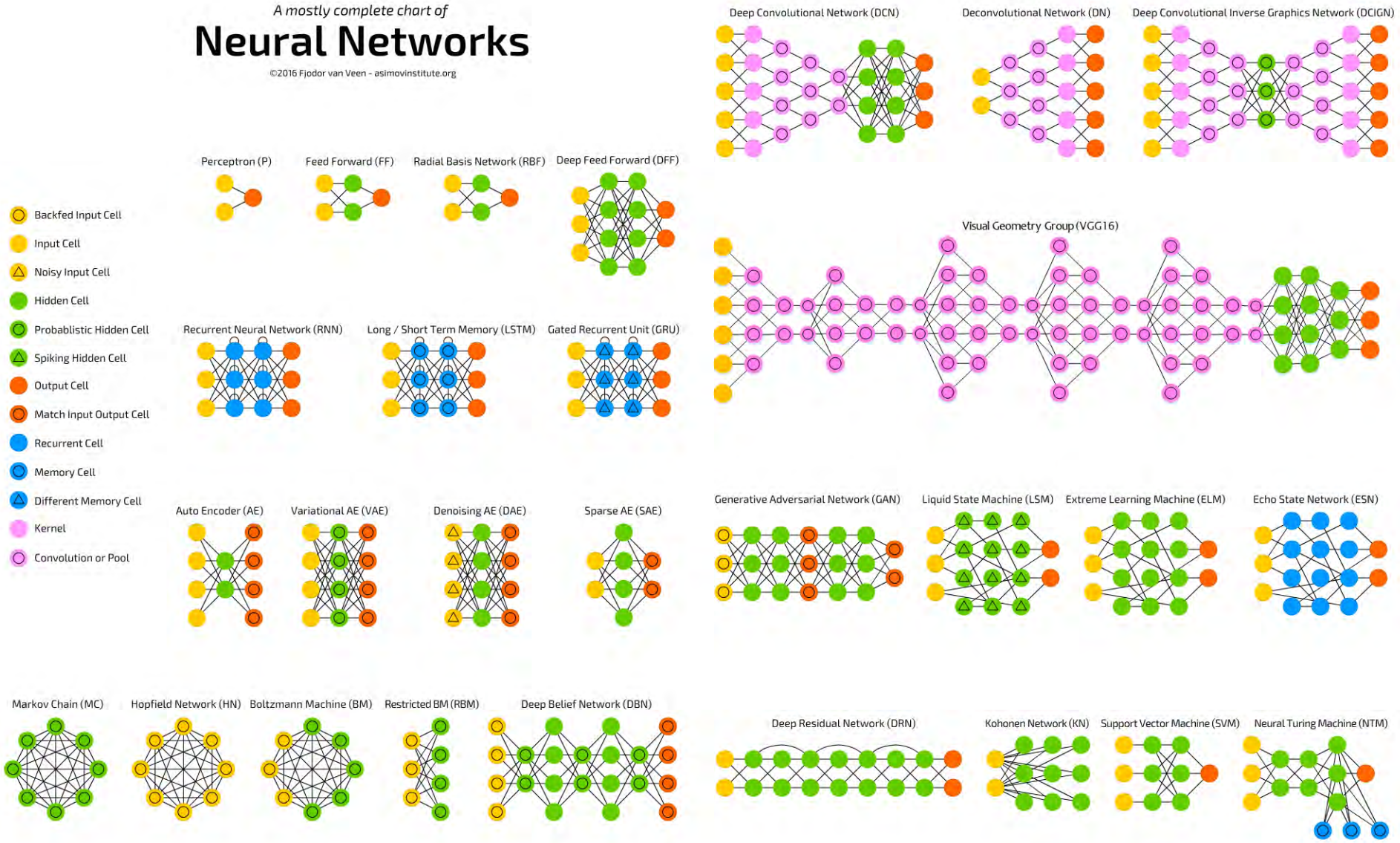
逗号运算符

“惬意”的编译期错误提示

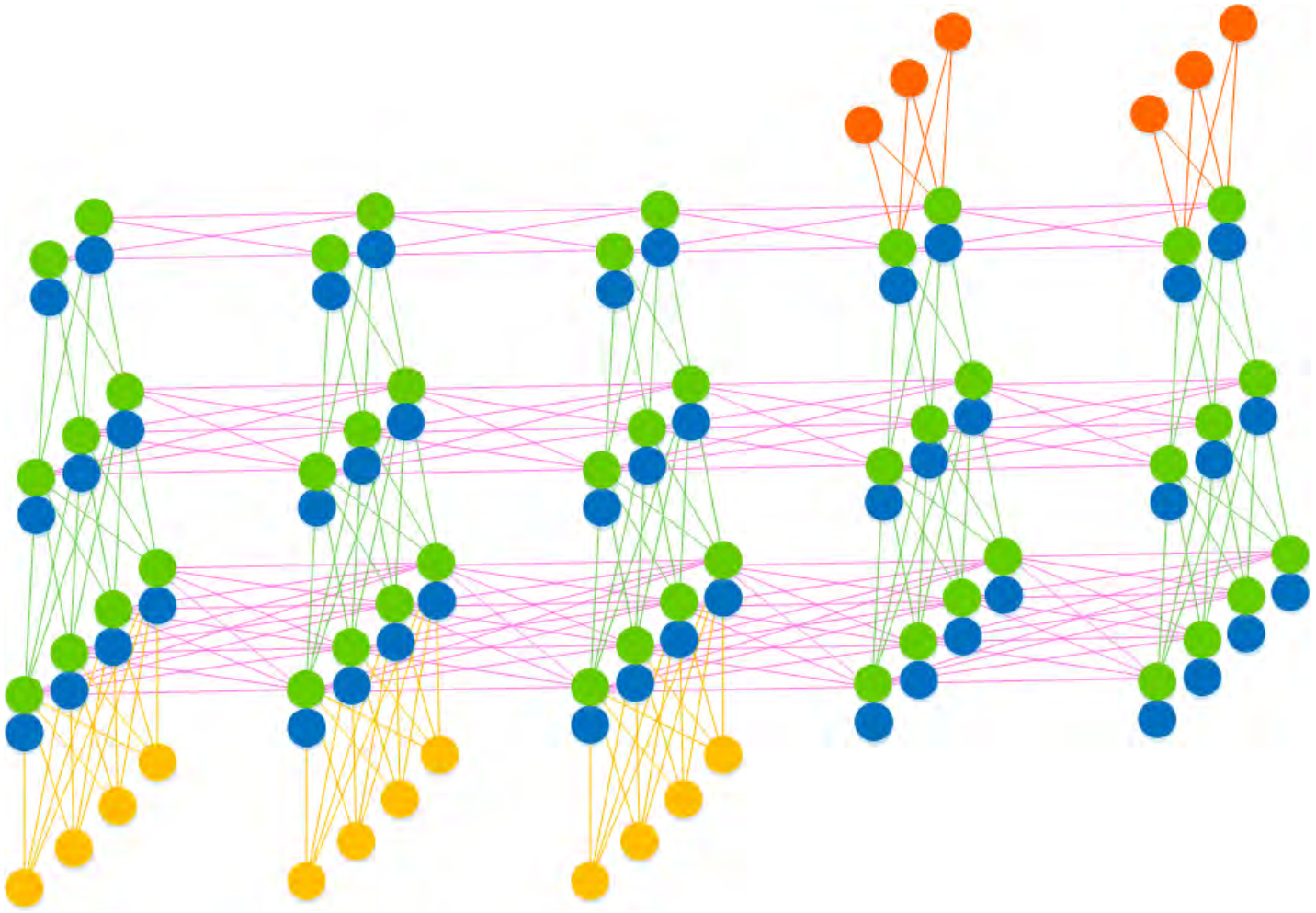

```
expander {(forward(std::get<I>(m_layers),
                   std::get<I>(m_layers), //std::get<I + 1>(m_layers),
                   std::get<I>(m_weights),
                   std::get<I>(m_thresholds)),
                   0)...};
```

```
 required from ‘bool mtl::BPNN<Layers>::train(const InMatrix&, const OutMatrix&, int, double) [with int ...Layers = {4, 2, 4, 2, 4, 2, 4};
 from here
47:5: error: no matching function for call to ‘mtl::Matrix<double, 1, 4>::multiply(mtl::Matrix<double, 1, 4>&, mtl::Matrix<double, 4, 2>&)’
 required from ‘bool mtl::BPNN<Layers>::train(const InMatrix&, const OutMatrix&, int, double) [with int ...Layers = {4, 2, 4, 2, 4, 2, 4};
 from here
47:5: error: no matching function for call to ‘mtl::Matrix<double, 1, 2>::multiply(mtl::Matrix<double, 1, 2>&, mtl::Matrix<double, 2, 4>&)’

 error: no match for ‘operator+=’ (operand types are ‘mtl::Matrix<double, 1, 2>’ and ‘mtl::Matrix<double, 1, 4>’)
 error: no match for ‘operator+=’ (operand types are ‘mtl::Matrix<double, 1, 4>’ and ‘mtl::Matrix<double, 1, 2>’)
```

常见的神经网络


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Visual Geometry Group (VGG16)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

最复杂的神经网络——LSTM



“门” 和新增的辅助类型

输入门，输出门，遗忘门和记忆门

```
enum Gate : unsigned char
{
    f = 0,
    i,
    C,
    o,
};
```

权重数组类型

```
typedef /// Weights type
std::tuple<
        Matrix<
                double,
                UnpackInts<I, Layers...>::value,
                UnpackInts<I + 1, Layers...>::value
        >[o]...
> Weights;
```

包含时间序列的临时类型

```
template<int R>
  using Temps = std::tuple<
                    Matrix<
                            double,
                            1,
                            UnpackInts<I + 1, Layers...>::value
                    >[R][o]...
                >;
```

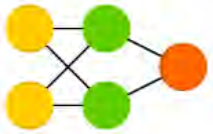
```
/// lstm 需要创建临时矩阵，用来保存当前系列输入的states
const int r = input.Row() + output.Row();
typename Type<std::make_index_sequence<N - 1>, Layers...>
        ::template RCells<r> rCells;
typename Type<std::make_index_sequence<N - 1>, Layers...>
        ::template Temps<r> states;
typename Type<std::make_index_sequence<N - 1>, Layers...>
        ::template Temps<r> rDeltas;
OUT trainOut;
OUT aberration;

auto& layer0 = std::get<0>(m_layers);
auto& layerN = std::get<N - 1>(m_layers);
auto& deltaN = std::get<N - 1>(m_deltas);
for(int i = 0; i < times; ++i)
{
    /// 1. 正向传播
    for(int t = 0; t < r; ++t)
    {
        /// 1.1 依次取input的每一层作为当前输入层
        layer0.subset(input, t, 0);
        layer0.normalize(nor);
        expander {(forward(std::get<I>(m_layers),
                           std::get<I + 1>(m_layers),
                           std::get<I>(m_weights),
                           std::get<I>(m_thresholds),
                           std::get<I + 1>(m_rWeights),
                           std::get<I + 1>(m_cells),
                           std::get<I>(rCells),
                           std::get<I>(states),
                           t, input.Row()),
                           0)...};
        /// 1.2 计算出的out依次赋给output的每一层
        if(t >= input.Row())
        {
            trainOut.set(layerN, t - input.Row(), 0);
```

允许可以变多输入

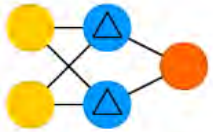
大量临时对象

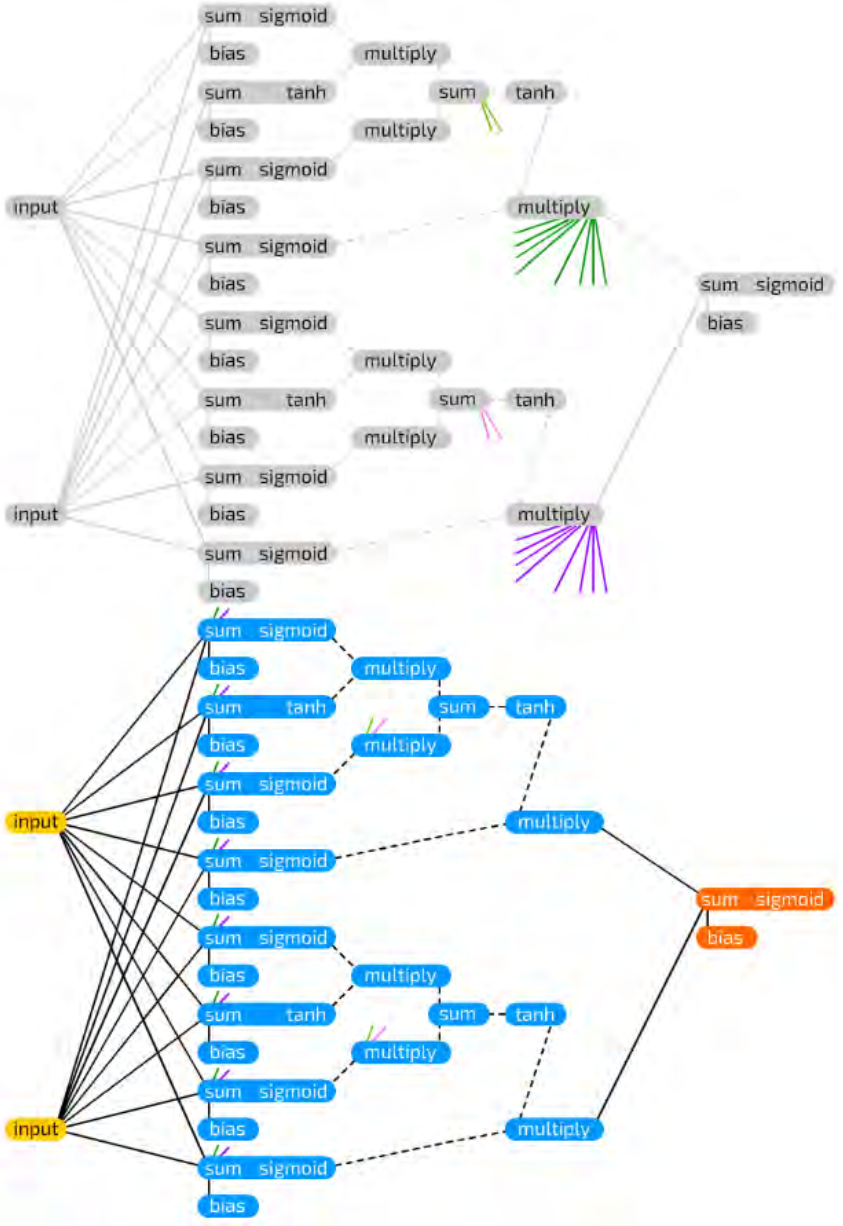
最复杂的神经网络——LSTM

Feed forward example



LSTM example





前方高能！！

前方高能！！

前方高能！！

LSTM公式及推导过程（看看就行，不懂也不要紧）

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(W_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(W_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
\tilde{\mathbf{c}}_t &= \tanh(W_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \\
\mathbf{c}_t &= f_t \circ \mathbf{c}_{t-1} + i_t \circ \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma(W_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t)
\end{aligned}
$$

$$
\delta_t \overset{def}{=} \frac{\partial E}{\partial \mathbf{h}_t}
$$

$$
\begin{aligned}
\mathbf{net}_{f,t} &= W_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f \\
&= W_{fh}\mathbf{h}_{t-1} + W_{fx}\mathbf{x}_t + \mathbf{b}_f \\
\mathbf{net}_{i,t} &= W_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i \\
&= W_{ih}\mathbf{h}_{t-1} + W_{ix}\mathbf{x}_t + \mathbf{b}_i \\
\mathbf{net}_{\tilde{c},t} &= W_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c \\
&= W_{ch}\mathbf{h}_{t-1} + W_{cx}\mathbf{x}_t + \mathbf{b}_c \\
\mathbf{net}_{o,t} &= W_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o \\
&= W_{oh}\mathbf{h}_{t-1} + W_{ox}\mathbf{x}_t + \mathbf{b}_o
\end{aligned}
$$

$$
\delta_{f,t} \overset{def}{=} \frac{\partial E}{\partial \mathbf{net}_{f,t}}
$$

$$
\delta_{i,t} \overset{def}{=} \frac{\partial E}{\partial \mathbf{net}_{i,t}}
$$

$$
\delta_{\tilde{c},t} \overset{def}{=} \frac{\partial E}{\partial \mathbf{net}_{\tilde{c},t}}
$$

$$
\delta_{o,t} \overset{def}{=} \frac{\partial E}{\partial \mathbf{net}_{o,t}}
$$

$$
\begin{aligned}
\delta_{t-1}^T &= \frac{\partial E}{\partial \mathbf{h}_{t-1}} \\
&= \frac{\partial E}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \\
&= \delta_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}
\end{aligned}
$$

$$
\delta_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \delta_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t}\frac{\partial \mathbf{o}_t}{\partial \mathbf{net}_{o,t}}\frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{h}_{t-1}} + \delta_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{f}_t}\frac{\partial \mathbf{f}_t}{\partial \mathbf{net}_{f,t}}\frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{h}_{t-1}} + \delta_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{i}_t}\frac{\partial \mathbf{i}_t}{\partial \mathbf{net}_{i,t}}\frac{\partial \mathbf{net}_{i,t}}{\partial \mathbf{h}_{t-1}}
$$

$$
\begin{aligned}
&= \delta_{o,t}^T \frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{h}_{t-1}} + \delta_{f,t}^T \frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{h}_{t-1}} + \delta_{i,t}^T \frac{\partial \mathbf{net}_{i,t}}{\partial \mathbf{h}_{t-1}} + \delta_{\tilde{c},t}^T \frac{\partial \mathbf{net}_{\tilde{c},t}}{\partial \mathbf{h}_{t-1}} \\
\delta_{t-1} &= \delta_{o,t}^T \frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{h}_{t-1}} + \delta_{f,t}^T \frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{h}_{t-1}} + \delta_{i,t}^T \frac{\partial \mathbf{net}_{i,t}}{\partial \mathbf{h}_{t-1}} + \delta_{\tilde{c},t}^T \frac{\partial \mathbf{net}_{\tilde{c},t}}{\partial \mathbf{h}_{t-1}} \\
&= \delta_{o,t}^T W_{oh} + \delta_{f,t}^T W_{fh} + \delta_{i,t}^T W_{ih} + \delta_{\tilde{c},t}^T W_{ch}
\end{aligned}
$$

$$
\begin{aligned}
\delta_{o,t}^T &= \delta_t^T \circ \tanh(\mathbf{c}_t) \circ \mathbf{o}_t \circ (1 - \mathbf{o}_t) \\
\delta_{f,t}^T &= \delta_t^T \circ \mathbf{o}_t \circ (1 - \tanh(\mathbf{c}_t)^2) \circ \mathbf{c}_{t-1} \circ \mathbf{f}_t \circ (1 - \mathbf{f}_t) \\
\delta_{i,t}^T &= \delta_t^T \circ \mathbf{o}_t \circ (1 - \tanh(\mathbf{c}_t)^2) \circ \tilde{\mathbf{c}}_t \circ \mathbf{i}_t \circ (1 - \mathbf{i}_t) \\
\delta_{\tilde{c},t}^T &= \delta_t^T \circ \mathbf{o}_t \circ (1 - \tanh(\mathbf{c}_t)^2) \circ \mathbf{i}_t \circ (1 - \tilde{\mathbf{c}}^2)
\end{aligned}
$$

$$
\delta_k^T = \prod_{j=k}^{t-1} \delta_{o,j}^T W_{oh} + \delta_{f,j}^T W_{fh} + \delta_{i,j}^T W_{ih} + \delta_{\tilde{c},j}^T W_{ch}
$$

$ht = Ot \cdot \tanh(Ct)$

$ht = \sigma(W \cdot Xt) \cdot \tanh(Ct)$

$ht = \sigma(W \cdot Xt) \cdot \tanh(it \cdot \tilde{Ct})$

$ht = \sigma(Wo \cdot Xt) \cdot \tanh(\sigma(Wi \cdot Xt) \cdot \tanh(Wt \cdot Xt))$

$ht = \sigma(Xt) \cdot \tanh(\sigma(Xt) \cdot \tanh(Xt))$

$$
\begin{aligned}
\frac{\partial E}{\partial \mathbf{net}_t^{l-1}} &= \frac{\partial E}{\partial \mathbf{net}_{f,t}^l}\frac{\partial \mathbf{net}_{f,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\partial \mathbf{net}_t^{l-1}} + \frac{\partial E}{\partial \mathbf{net}_{i,t}^l}\frac{\partial \mathbf{net}_{i,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\partial \mathbf{net}_t^{l-1}} + \frac{\partial E}{\partial \mathbf{net}_{\tilde{c},t}^l}\frac{\partial \mathbf{net}_{\tilde{c},t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\partial \mathbf{net}_t^{l-1}} + \frac{\partial E}{\partial \mathbf{net}_{o,t}^l}\frac{\partial \mathbf{net}_{o,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\partial \mathbf{net}_t^{l-1}} \\
&= \delta_{f,t}^T W_{fx} \circ f'(\mathbf{net}_t^{l-1}) + \delta_{i,t}^T W_{ix} \circ f'(\mathbf{net}_t^{l-1}) + \delta_{\tilde{c},t}^T W_{cx} \circ f'(\mathbf{net}_t^{l-1}) + \delta_{o,t}^T W_{ox} \circ f'(\mathbf{net}_t^{l-1}) \\
&= (\delta_{f,t}^T W_{fx} + \delta_{i,t}^T W_{ix} + \delta_{\tilde{c},t}^T W_{cx} + \delta_{o,t}^T W_{ox}) \circ f'(\mathbf{net}_t^{l-1})
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial W_{oh,t}} &= \frac{\partial E}{\partial \mathbf{net}_{o,t}}\frac{\partial \mathbf{net}_{o,t}}{\partial W_{oh,t}} \\
&= \delta_{o,t}\mathbf{h}_{l-1}^T \\
\frac{\partial E}{\partial W_{ox}} &= \frac{\partial E}{\partial \mathbf{net}_{o,t}}\frac{\partial \mathbf{net}_{o,t}}{\partial W_{ox}} \\
&= \delta_{o,t}\mathbf{x}_t^T \\
\frac{\partial E}{\partial W_{oh}} &= \sum_{j=1}^{t} \delta_{o,j}\mathbf{h}_{j-1}^T \\
\frac{\partial E}{\partial \mathbf{b}_{o,t}} &= \frac{\partial E}{\partial \mathbf{net}_{o,t}}\frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{b}_{o,t}} \\
&= \delta_{o,t}
\end{aligned}
\qquad
\frac{\partial E}{\partial \mathbf{b}_o} = \sum_{j=1}^{t} \delta_{o,j}
$$

$h' = \sigma(X)' \cdot \tanh(\sigma(X) \cdot \tanh(X)) + \sigma(X) \cdot \tanh(\sigma(X) \cdot \tanh(X))'$ 莱布尼兹公式

$h' = dsig(X) \cdot \tanh(sig(X) \cdot \tanh(X)) + sig(X) \cdot dtanh((sig(X) \cdot \tanh(X)) \cdot (dsig(X) \cdot \tanh(X) + sig(X) \cdot dtanh(X))$ 链式法则 + 莱布尼兹公式

$h' = sig(X) \cdot (1 - sig(X)) \cdot \tanh(\Delta) + sig(X) \cdot (1 - \tanh(\Delta)^2) \cdot (sig(X) \cdot (1 - sig(X)) \cdot \tanh(X) + sig(X) \cdot (1 - \tanh(X)^2))$

倒序计算循环过程中产生的delta（时间方向上的）


```
auto delta_cal = [&](unsigned char g, auto func)
{
    if(t >= r - 1)
    {
        /// 倒数第一个时刻
        rDelta[t][g] = gDelta[g];
        gDelta[g].hadamard(rDelta[t][g], state[t][g]);
    }
    else if(t >= rIn)
    {
        /// 对应输出的时刻
        rDelta[t][g] = gDelta[g];
        rDelta[t][g].mult_trans_sum(rWeight[g], state[t + 1][g].foreach(func));
        gDelta[g].hadamard_sum(rDelta[t][g], state[t][g]);
        rDelta[t][g] += rDelta[t + 1][g];
    }
    else
    {
        /// 对应输入的时刻
        rDelta[t][g].mult_trans(rWeight[g], state[t + 1][g].foreach(func));
        gDelta[g].hadamard_sum(rDelta[t][g], state[t][g]);
        rDelta[t][g] += rDelta[t + 1][g];
    }
};
delta_cal(f, dlogsig);
delta_cal(i, dlogsig);
delta_cal(C, dtansig);
delta_cal(o, dlogsig);
```

LSTM应用实例——实时数据序列