C++

The Beast is back !!

Agenda

- Présentation
- Technologie C++
- Application au Machine Learning
- Petite démo

Qui sommes-nous ?

- 11 ans d'expérience
- Monitoring passif du coeur de réseau d'opérateurs télécom
- Les opérateurs mobiles et les entreprises de chemin de fer (GSM-R)
- Analyse / Business Intelligence

@ExpandiumSAS

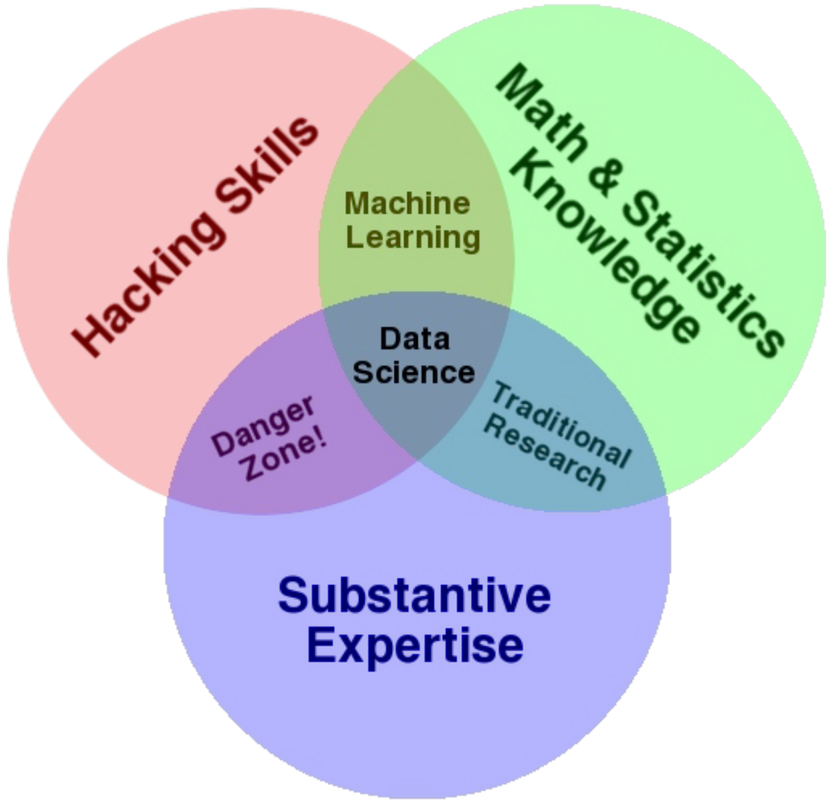
Qui sommes-nous ?

Pierre Salmon

https://github.com/ddway2

Qui sommes-nous ?

Nicolas Greffard



Quelles sont nos problématiques ?

- Gestion de gros volume
- Capacité de traitement
- Contrainte hardware
- Données sensibles
- Coût de maintenance

Pourquoi les technologies C++ ?

Au commencement : 1979 - 1983

C with Classes

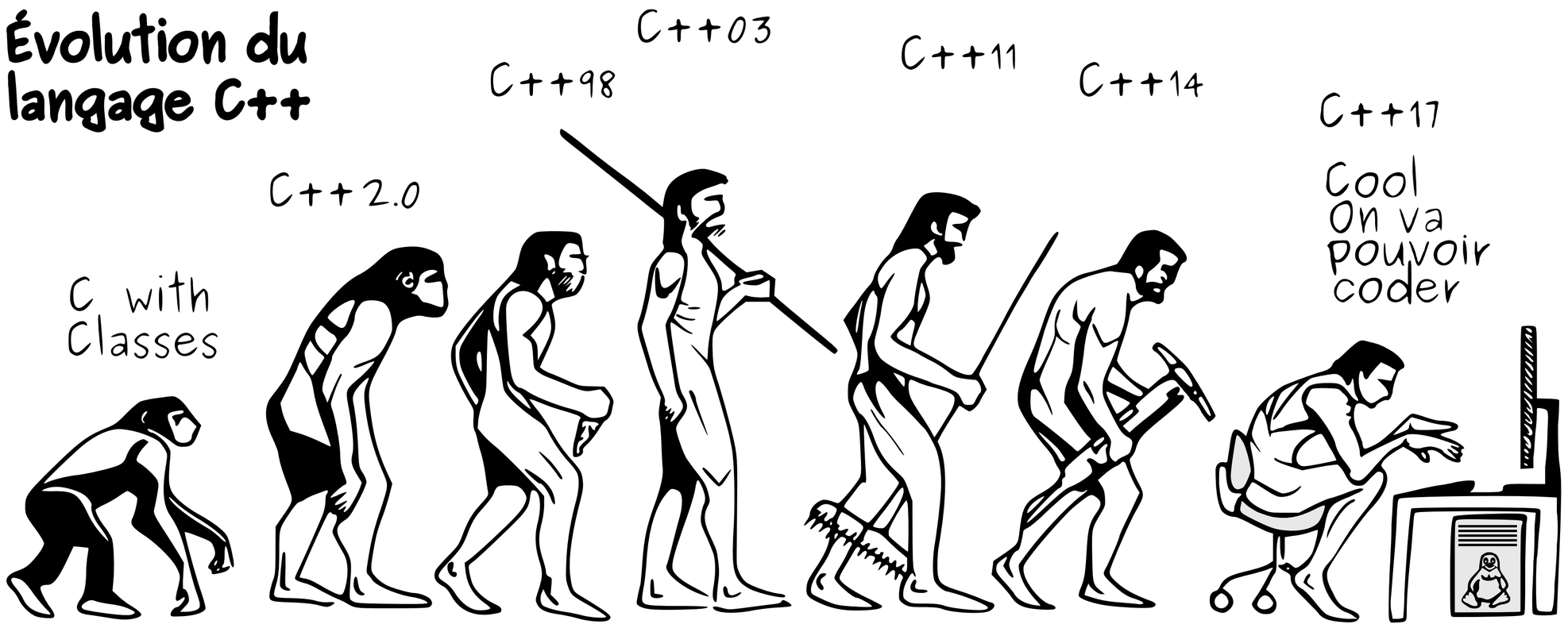
Bjarne Stroustrup

1998 : standard ISO

2011 - 2014 : Le renouveau

- Le langage C++
- La bibliothèque Standard (STL)
- La bibliothèque Boost

Évolution du langage C++



Luttons contre les idées reçues

Idée reçue n°1 :

”Il n’y a pas de Garbage Collector : il faut gérer soi-même la mémoire et les objets”

Un pool de serveurs TCP

Comment gérer une instance partagée ?

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    // ...
};

std::vector<server_tcp*>            server_tcp_pool;

server_tcp* new_server() {
    server_tcp* srv = new server_tcp();

    server_tcp_pool.push_back(srv);
    return srv;
}

// ... Un peu de traitement
void shutdown() {
    for (server_tcp* srv : server_tcp_pool) {
        delete srv;
    }
    server_tcp_pool.clear();
}
```

Smart Pointer

Le pointeur intelligent pour partager les variables

```
#include <memory>

std::vector<std::shared_ptr<server_tcp>>            server_tcp_pool;

std::shared_ptr<server_tcp> new_server() {
    std::shared_ptr<server_tcp> srv =  std::make_shared<server_tcp>();

    server_tcp_pool.push_back(srv);
    return srv;
}

/// Live your life server TCP variable...

void shutdown() {
    server_tcp_pool.clear();
}
```

Focus sur l'envoi de paquets

```
class server_tcp {
public:
    struct packet {
        uint64_t          id;
        std::vector<uint8_t> data(2000000);        // Big packet
        // ...
    };
public:
    void send(packet p) {  /* ... */ }     // Passage par valeur

    void run() {
        while (!done()) {
            packet pkt;

            // ... un peu de traitement

            send(pkt);
        }
    }
};
```

Une copie de 2'000'000 octets... Ce n'est pas l'idéal

Améliorons cela avec un passage par référence

```
class server_tcp {
public:
    void send(packet& p) {  /* ... */ }     // Passage par référence

    void run() {
        while (!done()) {
            packet pkt;

            // ... un peu de traitement

            send(pkt);
        }
    }
};
```

Mais que ce passe-t-il si send est asynchrone ?

Move Semantic

...ou comment passer le controle d'une variable à un autre scope

```
class server_tcp {
public:
    void send(packet&& p) {  /* ... */ }     // Passage par rvalue

    void run() {
        while (!done()) {
            packet pkt;

            // ... un peu de traitement

            send(std::move(pkt));
        }
    }
};
```

Idée reçue n°2

”Il y a du typage partout !
Sans IDE c’est trop dur”

Management d'un pool de serveur

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    // ...
};

std::vector<std::shared_ptr<server_tcp>>            server_tcp_pool;
std::map<uint32_t, std::shared_ptr<server_tcp>>     server_tcp_by_id;

...

std::map<uint32_t, std::shared_ptr<server_tcp>>::iterator found = server_pool.find(42);
socket s = found->second->sock;
```

C'est un peu verbeux non ?

On peut toujours utiliser un alias (using ou typedef)

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    // ...
};

typedef  std::vector<std::shared_ptr<server_tcp>> server_tcp_pool_type;
using server_tcp_by_id_type = std::map<uint32_t, std::shared_ptr<server_tcp>>;

server_tcp_pool_type      server_tcp_pool;
server_tcp_by_id_type     server_tcp_by_id;

...

server_tcp_by_id_type::iterator found = server_tcp_by_id.find(42);
socket s = found->second->sock;
```

Le mot clé auto à la rescousse...

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    // ...
};

typedef std::vector<std::shared_ptr<server_tcp>> server_tcp_pool_type;
using server_tcp_by_id_type = std::map<uint32_t, std::shared_ptr<server_tcp>>;

server_tcp_pool_type      server_tcp_pool;
server_tcp_by_id_type     server_tcp_by_id;

...

auto found = server_tcp_by_id.find("toto");
auto sock = found->second->name;
```

Idée reçue n°3

”C’est un langage limité et bas niveau”

Exemple : liste des instances server_tcp fermées

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    bool        closed;
    // ...
};

using server_tcp_pool_type = std::vector<std::shared_ptr<server_tcp>>;
using server_list_type = std::list<std::shared_ptr<server_tcp>>;
server_tcp_pool_type      server_tcp_pool;

server_list_type closed_socket_list(server_tcp_pool_type& srv)
{
    server_list_type result;
    for (size_t i = 0 ; i < srv.size() ; ++i) {
        if (srv[i].closed) {
            result.push_back(srv[i]);
        }
    }
    return result;
}
```

Avec des algorithmes, des itérateurs et des lambdas

```
class server_tcp {
public:
    uint32_t    id;
    socket      sock;
    bool        closed;
    // ...
};

using server_tcp_pool_type = std::vector<std::shared_ptr<server_tcp>>;
using server_list_type = std::list<std::shared_ptr<server_tcp>>;
server_tcp_pool_type      server_tcp_pool;

template<typename Container>
server_list_type closed_socket_list(Container& c)
{
    server_list_type result;
    std::for_each(
        c.begin(),
        c.end(),
        [&result](const auto& s){
            if (s.closed) {

result.push_back(s);
            }
        }
    );
    return result;
}
```

Les constantes ”magiques”

Qui n'a pas déjà vu cela dans du code :

```
const uint64_t delay = 200;      // 200 quoi ? secondes, millisecondes ?

process_data(42, packet);        //  La réponse à la grande question sur la vie ?
```

User defined literal

```
const duration_type delay = 200_ms;

process_data(42_mo, packet);
```

Pour la mise en oeuvre :

```
struct duration_type {
    uint64_t        value_ms = 0;
};

constexpr duration_type operator ""_s(unsigned long long int v) {
    duration_type result;
    result.value_ms = v * 1000;
    return result;
}

constexpr duration_type operator ""_ms(unsigned long long int v) {
    duration_type result;
    result.value_ms = v;
    return result;
}
```

Idée reçue n°4

”Il y a pas grand chose dans la librairie standard”

Le ”standard” C++ en 2014

La bibliothèque STL

container, algorithme, thread, filesystem, regexp, chrono, math,...

La bibliothèque BOOST

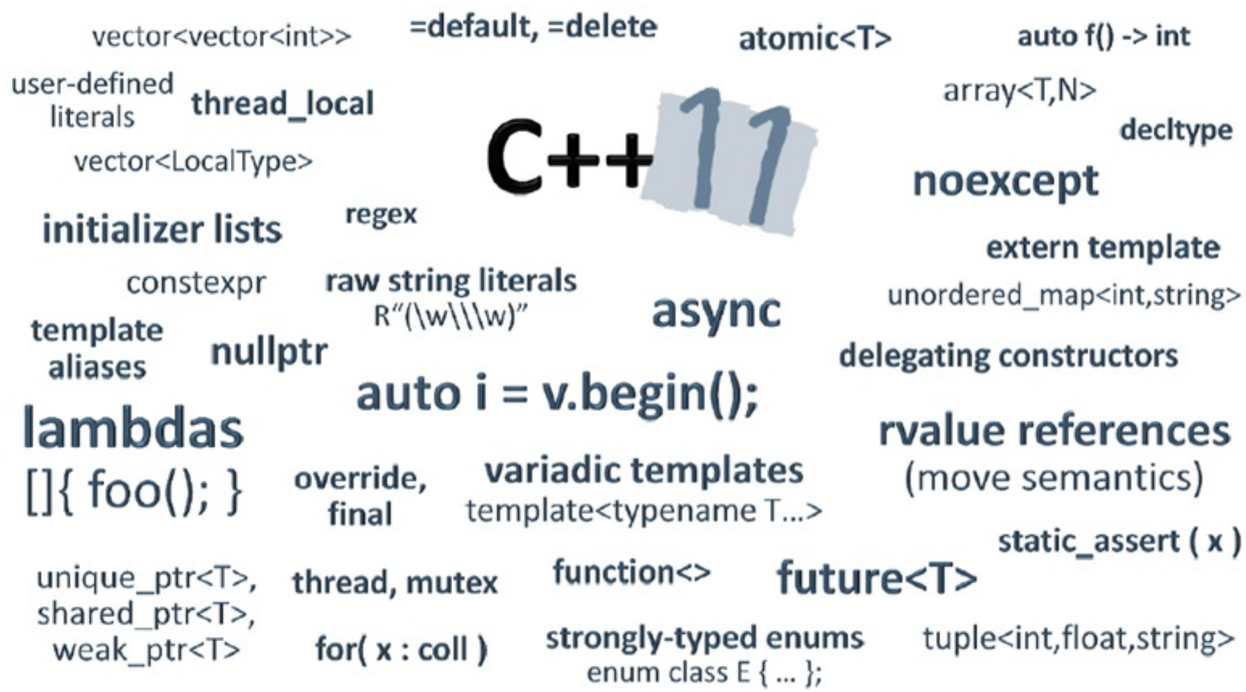
le reste... (ASIO, GPU, coroutine, fiber, Date, graph, ...) 142 modules

Et aussi : l'intégralité des bibliothèques C/C++

Quelques exemples :

- Expandium NX - Librairie HTTP asynchrone
- Facebook Folly - Collection d'utilitaire
- nghttp2 - librairie HTTP/2
- Librairie Cloud (MapR)
- ...

Et ce n'est qu'un aperçu



Comment faire du C++ aujourd'hui ?

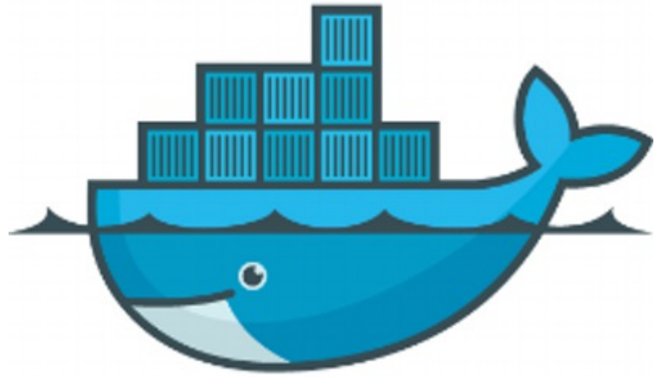
Phoenix

Concepts

- Gérer un projet C++
- Gérer ses dépendances de build sans déclarations
- "Multi plateforme" -> Utilisation de docker
- Génération d'images docker utilisables dans un cloud

Solution

- Docker
- Debian apt
- cbuild (Remy Chibois)



Un projet phoenix ?

Un fichier de description : cbuild.conf

```
PKG_VER=1.0
PKG_REV=1
PKG_SHORTDESC="Caffe C++ Datamining"
PKG_LONGDESC="
Caffe server for C++ Datamining with phoenix integration
"

# cbuild settings
PRJ_NAME=caffe-datamining
PRJ_SERVER=server
PRJ_OPTS["std"]=c++14
```

Un projet phoenix ?

Une arboresence

```
my-project
├── DEBS
│   └── libcaffe.deb
├── sources
│   ├── tests
│   │   ├── check_server
│   │   │   └── check_server.cpp
│   ├── binaries
│   │   ├── predicate
│   │   │   └── main.cpp
│   │   └── server
│   │       └── main.cpp
│   ├── include
│   │   └── caffe-datamining
│   │       └── devfest2016
│   │           ├── classifier.hpp
│   │           └── signal.hpp
│   └── libraries
│       └── caffe-datamining
│           ├── classifier.cpp
│           └── signal.cpp
└── cbuild.conf
```

Et quelques commandes...

```
phoenix-build configure
phoenix-build install
phoenix-build run

phoenix-build image
```

That's All...

Un petit exemple d'utilisation concret

Problématiques data ?

- ~~$ Learn from the past to predict the future~~
- No BS : essayer d'aller au delà des KPIs
- Ex: #dropped calls => churn probability
- #calls + #calls_fhz => fraudeur

Données sensibles

Prenons un exemple classique

- Classification: **étiqueter**/labelliser des objets (données) comme appartenant à une catégorie parmi plusieurs
- Ex : un utilisateur satisfait ou non; une image comme étant un visage ou un poisson

Classification supervisée

Le modèle apprend à discriminer les catégories à partir de données déjà étiquetées. C’est la phase d’apprentissage

Apprentissage

Plane

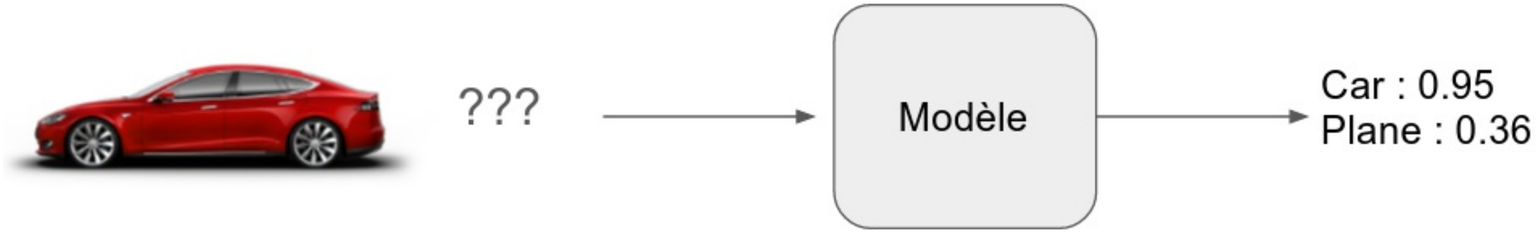
Car

Modèle

Classification supervisée

Une fois le modèle obtenu, on lui envoie des images non étiquetées et il prédit l'étiquette correspondant

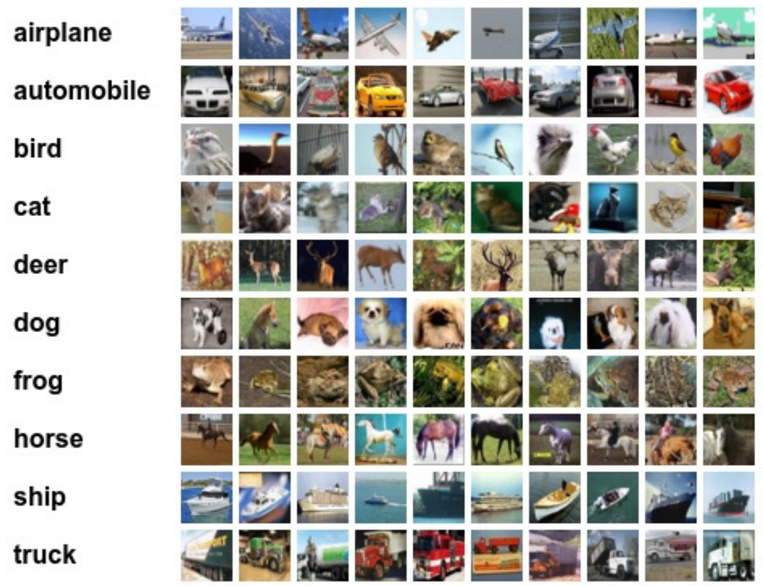
Prédictions



Webservice de classification d'images

C++ versus Python

CIFAR10



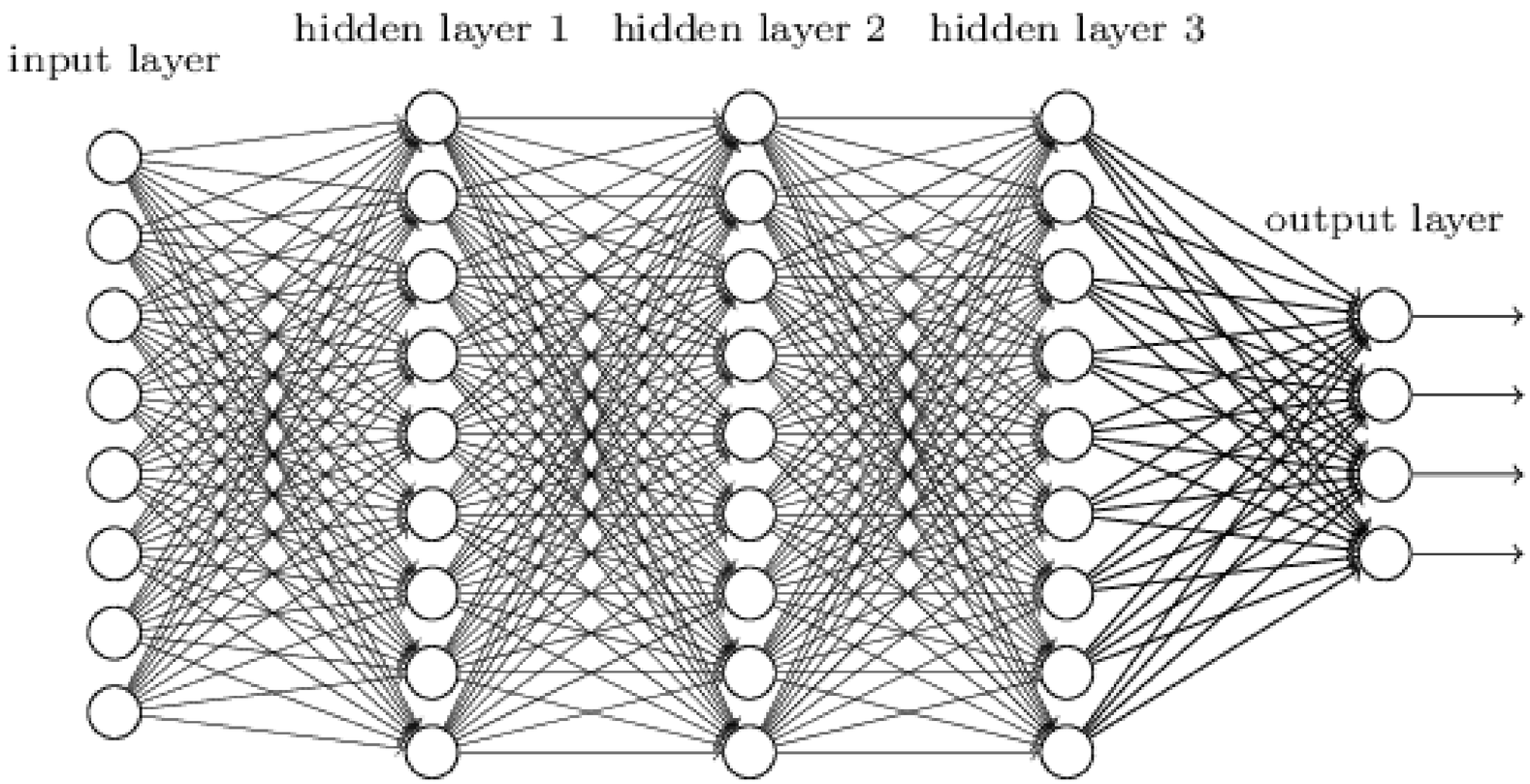
Deep learning : CNN (réseau neuronal convolutif)

Derrière un serveur http qui reçoit des images et qui renvoie ce qu'elles représentent

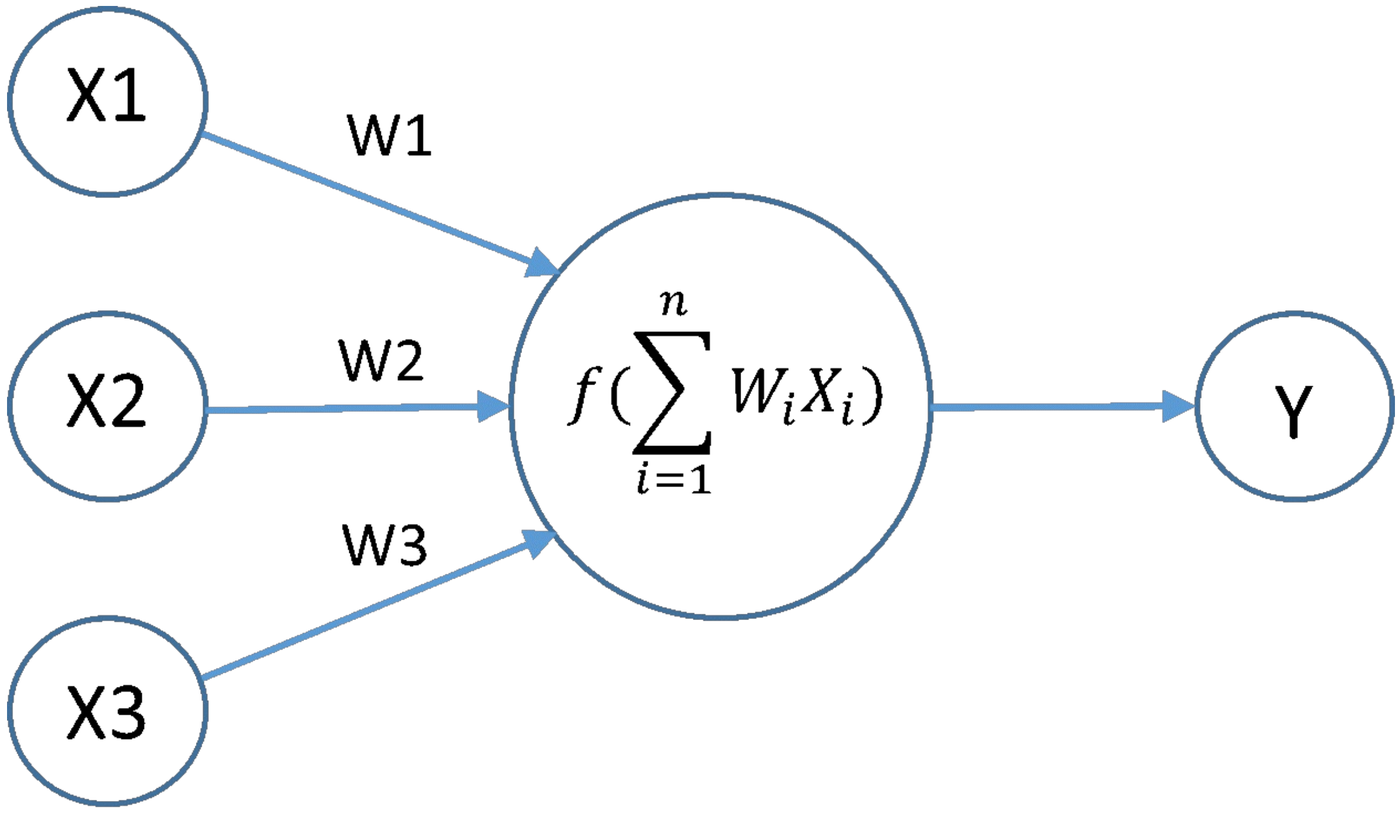
Réseau neuronal

- Algorithme d'apprentissage
- **Inspiré** du fonctionnement du cerveau humain
- Perceptron en 1957 par F. Rosenblatt

Réseau neuronal



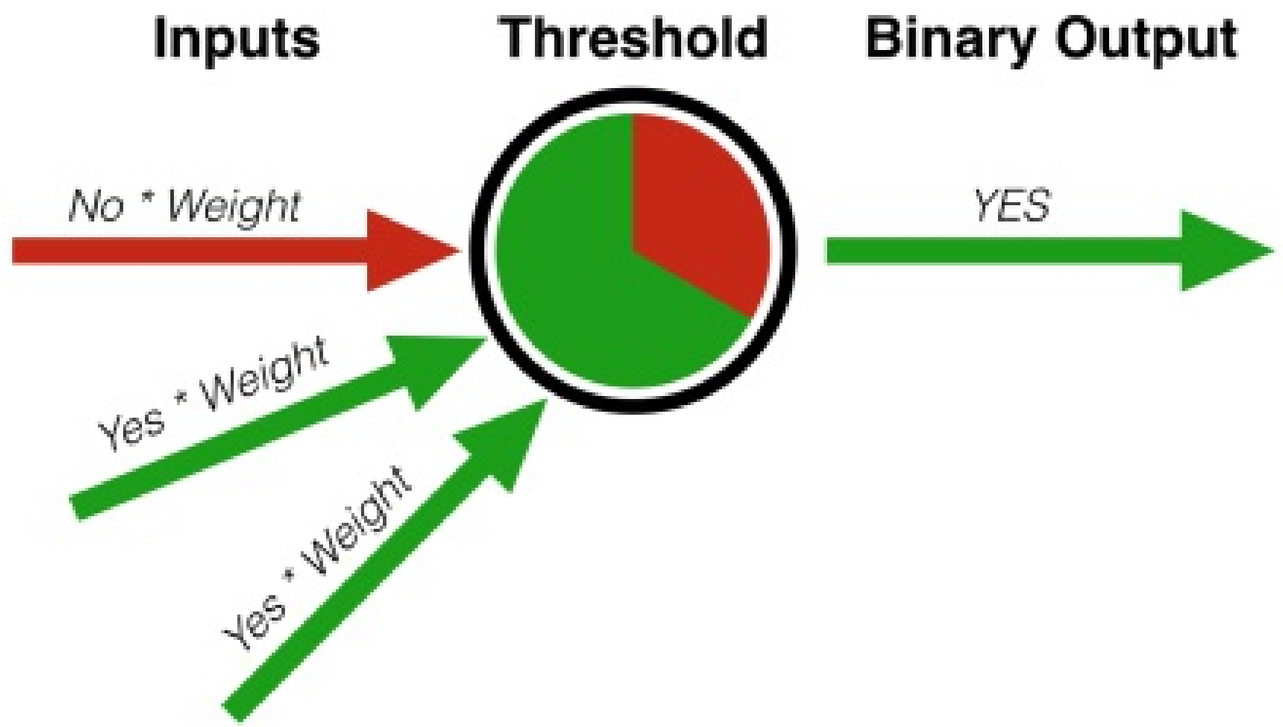
Réseau neuronal



Intuition :

- Si la somme des entrées tend vers +∞ alors la sortie tend vers 1
- Si la somme des entrées tend vers -∞ alors la sortie tend vers 0

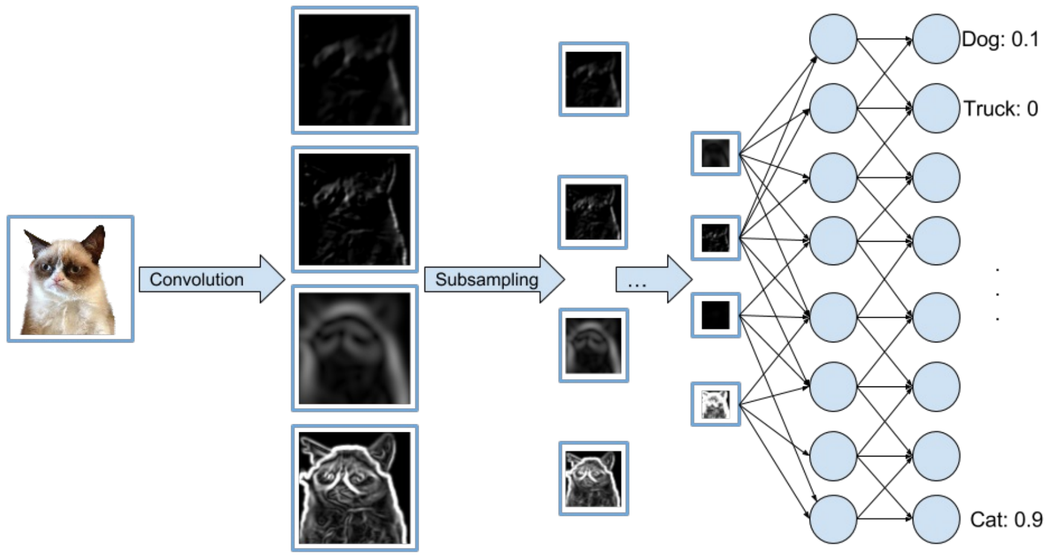
Intuition


Inputs
Threshold
Binary Output
No * Weight
Yes * Weight
Yes * Weight
YES

Backpropagation

- On calcule les sorties à partir des entrées
- On mesure l'erreur
- On fait remonter l'erreur dans le réseau via backpropagation (dérivées partielles) et on modifie les poids en conséquence

Réseau neuronal convolutif



Réseau neuronal convolutif

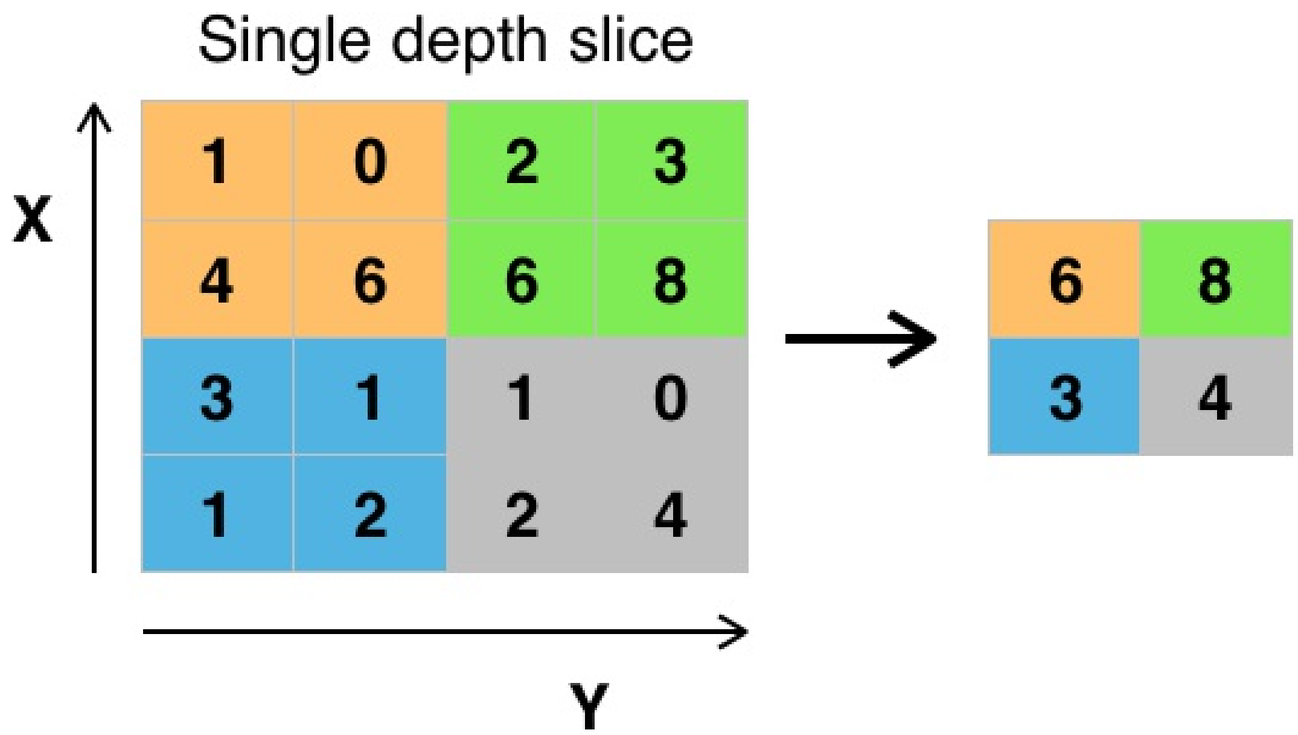
Input image

Convolution Kernel

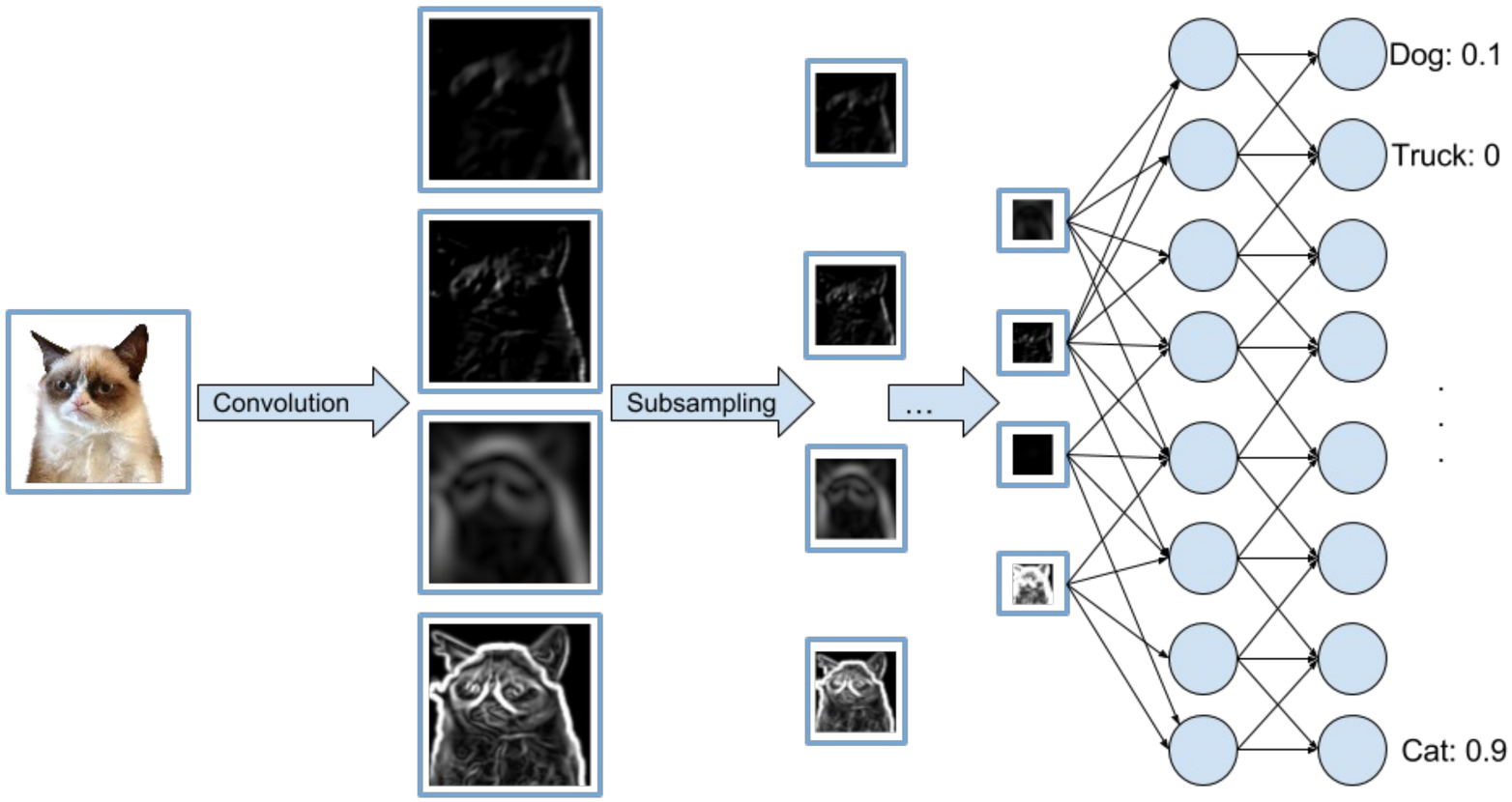
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

Réseau neuronal convolutif



Réseau neuronal convolutif



Lib : Caffe

- Développé en C++ avec wrapper python
- Comme la majorité des solutions...
- La définition du modèle se fait via fichier de config

Http serveur python

Du classique : Flask

```
import sys
import os
from flask import Flask
from flask import request
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = '/tmp'

import caffe

def loadNet():
    # ... Loading Net

    return(net)

net = loadNet()

@app.route("/predict_from_file", methods = ['POST'])
def predict_from_file():
    app.logger.info('Predict OK')
    if 'upload_file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['upload_file']
    app.logger.info('file found')
    if file.filename == '':
        flash('No Selected file')
        return redirect(request.url)

    filename = file.filename
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

    x   = net.predict( [ caffe.io.load_image(file) ])
    return json.dumps(x.tolist())

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Http serveur C++ sur Phoenix

```
    auto classifier = std::make_unique<devfest2016::classifier>();

    using namespace nx;

    httpd srv;
    MPFD::Parser  part_parser;
    part_parser.SetTempDirForFileUpload("/tmp");

    srv(POST) / "predict_from_file" = [&](const request& req, buffer& data, reply& rep) {
        part_parser.SetContentType(req.h("Content-Type"));
        part_parser.AcceptSomeData(&(data[0]), data.size());
        auto file = part_parser.GetFieldsMap()[0];

        if (file->GetType() == MPFD::Field::FileType) {
            auto result = classifier->classify(file->GetTempFileName());
            cxxu::rmfile(file->GetTempFileName());
            rep <<
nx::json(result)
;
        }
    };

    cxxu::info() << "Serve at " << bind_addr << ":" << bind_port;
    srv(make_endpoint(bind_addr, bind_port));
```

Demo :

Et les perfs ?

- Python+Caffe+Flask : 30req/s
- Phoenix+Caffe+Nx : 60 req/s

Conclusion

- Utilisation des nouvelles technologies
- Industrialisation des POC DataMining
- Disponibilité des compétences (Go/R)
- Obtenir le maximum d'une architecture hardware

Remerciements

- Remy Chibois - cbuild
- Anthony Garreau - GUI
- Expandium
- Et l'équipe du DevFest

Des questions ?

- https://github.com/ExpandiumSAS
- https://github.com/ddway2/devfest2016-phoenix-caffe
- https://github.com/ddway2/neo-phoenix
- https://github.com/chybz/cbuild