Outrepasser les limites des techniques classiques de Prise d'Empreintes grâce aux Réseaux de Neurones

Javier Burroni - Carlos Sarraute

{ javier, carlos } @ coresecurity.com

Core Security Technologies

SSTIC 2006

PLAN

1. Introduction

2. Mappeur d'endpoint DCE-RPC

3. Détection d'OS basée sur les signatures de Nmap

4. Réduction des dimensions et entraînement

1. Introduction

2. Mappeur d'endpoint DCE-RPC

3. Détection d'OS basée sur les signatures de Nmap

4. Réduction des dimensions et entraînement

Identification de systèmes d'exploitation

- Identification d'OS = Détection d'OS = Fingerprinting d'OS
- Étape cruciale d'un test d'intrusion
  - envoyer activement des paquets de test et étudier la réponse de la machine
- Première génération : analyse des différences entre les implémentations de la pile TCP/IP
- Génération suivante : analyse de données au niveau de la couche d'applications (DCE RPC endpoints)
  - pour raffiner la détection de versions / éditions / service packs de Windows

Limitations des outils de OS Fingerprinting

- L'information est analysée en utilisant une variation de l'algorithme consistant à chercher le point le plus proche ("best fit")
  - Ne marche pas dans les situations non standard
  - Est incapable de reconnaître les éléments clefs de la réponse
- Notre proposition:
  - Se concentrer sur les techniques utilisées pour analyser l'information
  - Nous avons développé des outils utilisant des réseaux de neurones
  - Qui ont été intégrés dans un logiciel commercial : Core Impact

1. Introduction

2. Mappeur d’endpoint DCE-RPC

3. Détection d’OS basée sur les signatures de Nmap

4. Réduction des dimensions et entraînement

Service DCE-RPC de Windows

- En envoyant une requête RPC au port 135 d'une machine
  il est possible de déterminer quels services ou programmes sont registrés

- La réponse inclut:
  - UUID = 'universal unique identifier' pour chaque programme
  - Nom annoté
  - Protocole utilisé par chaque programme
  - Adresse de réseau à laquelle le programme est lié
  - Point final du programme (endpoint)

Endpoints d'un Windows 2000 Édition professionnelle service pack 0

- uuid="5A7B91F8-FF00-11D0-A9B2-00C04FB6E6FC"
  annotation="Messenger Service"
  - protocol="ncalrpc" endpoint="ntsvcs" id="msgsvc.1"
  - protocol="ncacn_np" endpoint="\PIPE\ntsvcs" id="msgsvc.2"
  - protocol="ncacn_np" endpoint="\PIPE\scerpc" id="msgsvc.3"
  - protocol="ncadg_ip_udp" id="msgsvc.4"
- uuid="1FF70682-0A51-30E8-076D-740BE8CEE98B"
  - protocol="ncalrpc" endpoint="LRPC" id="mstask.1"
  - protocol="ncacn_ip_tcp" id="mstask.2"
- uuid="378E52B0-C0A9-11CF-822D-00AA0051E40F"
  - protocol="ncalrpc" endpoint="LRPC" id="mstask.3"
  - protocol="ncacn_ip_tcp" id="mstask.4"

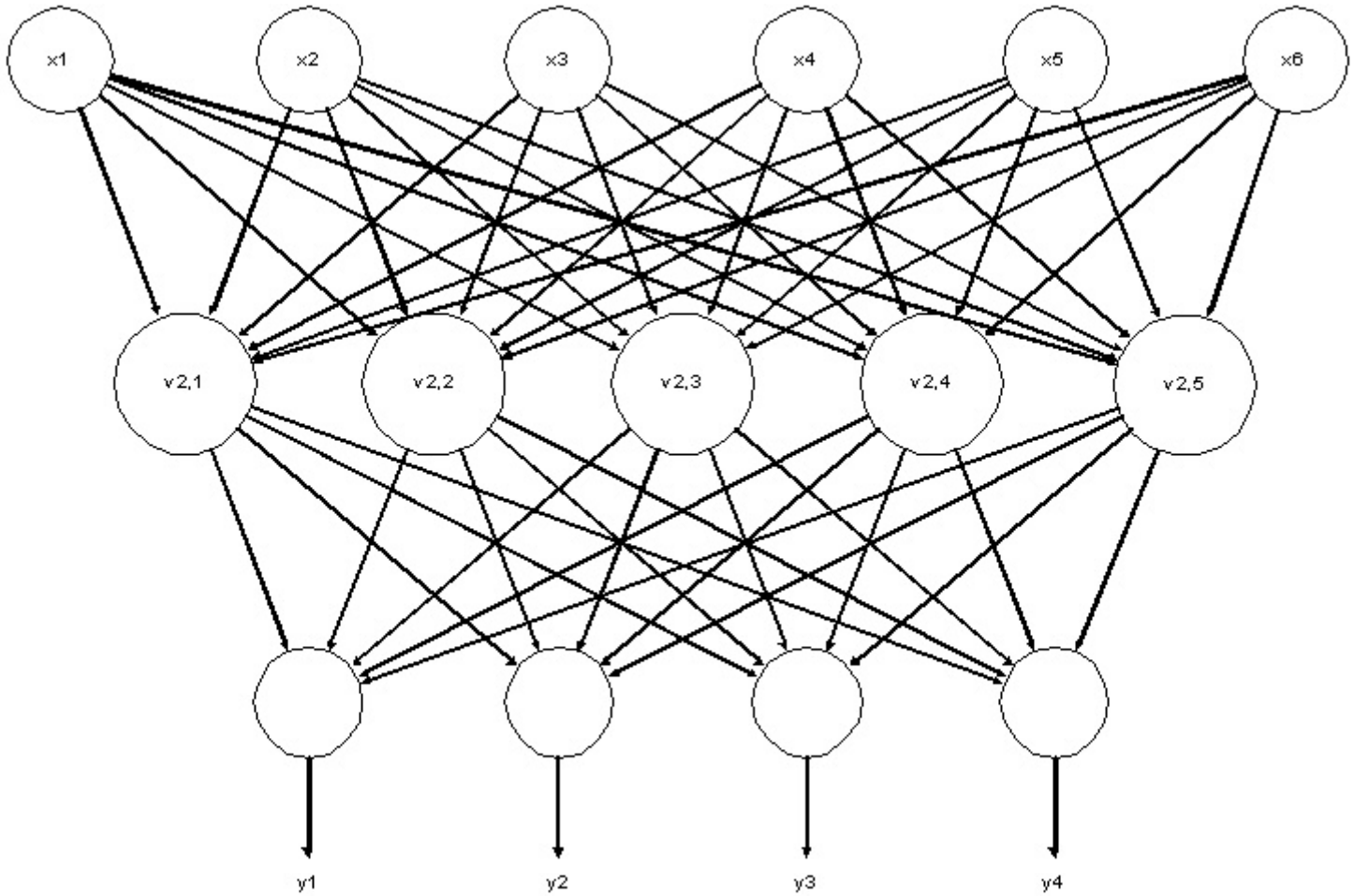
Les réseaux de neurones entrent en jeu…

- Il est possible de reconnaître les versions, éditions et service packs de Windows a partir de la combinaison de points finaux fournis par le service DCE-RPC

- Idée: modeler la fonction qui fait correspondre les combinaisons de points finaux aux versions du système d'exploitation avec un réseau de neurones

- Plusieurs questions se posent:
  - quel genre de réseau de neurones allons nous utiliser ?
  - comment organiser les neurones ?
  - comment faire correspondre les combinaisons de points finaux avec les neurones d'entrée du réseau ?
  - comment entraîner le réseau ?

Réseau de Neurones Perceptron Multicouche



413 neurones

42 neurones

25 neurones

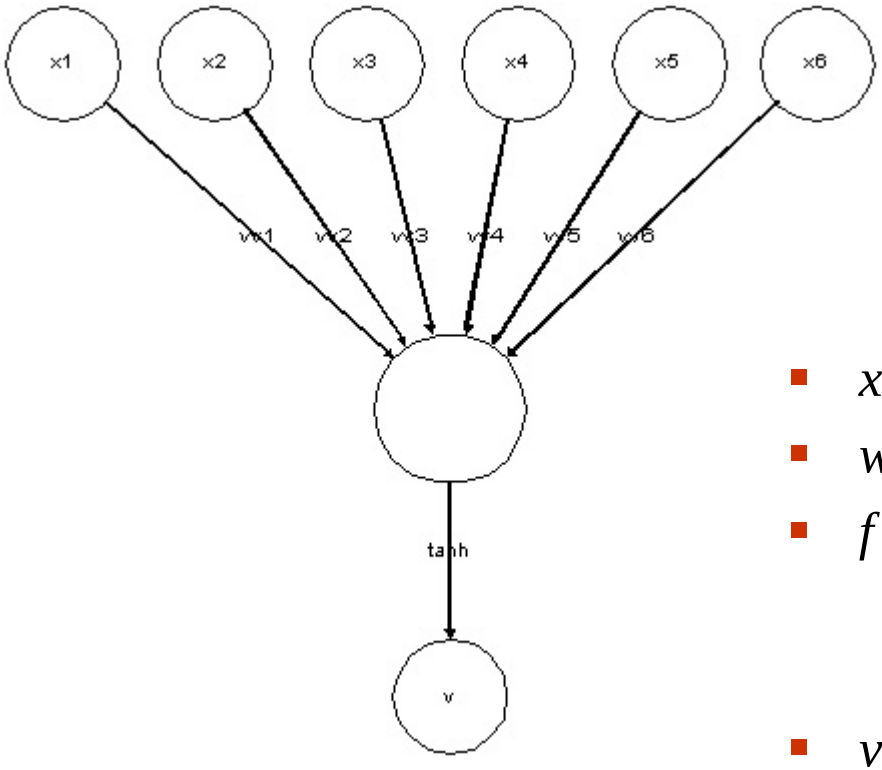
Topologie avec 3 couches

- Couche d'entrée : 413 neurones
  - un neurone pour chaque UUID
  - un neurone pour chaque point final qui correspond à cet UUID
  - permet de répondre avec flexibilité à l'apparition d'un point final inconnu

- Couche de neurones cachés : 42 neurones
  - chaque neurone représente une combinaison des neurones d'entrée

- Couche de sortie : 25 neurones
  - un neurone pour chaque version de Windows
  - un neurone pour chaque version et édition de Windows
    - » Windows 2000 édition professionnelle
  - un neurone pour chaque version et service pack de Windows
    - » Windows 2000 service pack 2
  - les erreurs dans une dimension n'affectent pas les erreurs dans l'autre dimension

Qu'est-ce qu'un perceptron?



$$v_{i,j} = f(\sum_{k=0}^{n} w_{i,j,k} \cdot x_k)$$

- $x_1 \ldots x_n$ sont les entrées du neurone
- $w_{i,j,0} \ldots w_{i,j,n}$ sont les poids synaptiques
- f est une fonction d'activation non linéaire
  - nous utilisons la tangente hyperbolique tanh
- $v_{i,j}$ es la sortie du neurone

Entraînement du réseau = calculer les poids pour chaque neurone

Rétro-propagation

- Entraînement par rétro-propagation
- Pour la couche de sortie
  - à partir de la sortie recherchée $y_1 \dots y_m$
  - calculer une estimation de l'erreur

$$\delta_{i,j} = f'(v_{i,j})\ (y_j - v_{i,j})$$

- Celle-ci est propagée aux couches précédentes par:

$$\delta_{i,j} = f'(v_{i,j}) \sum_k w_{i,j,k} \cdot \delta_{i+1,j}$$

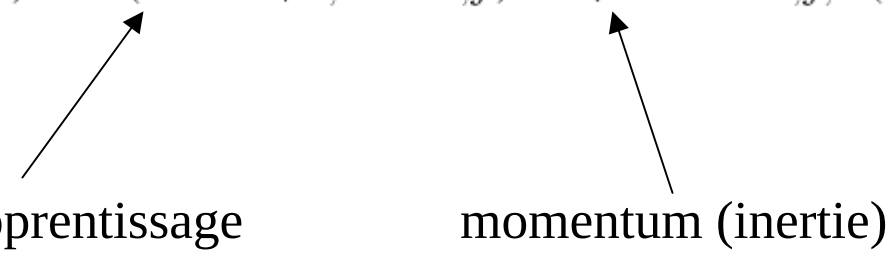
Nouveaux poids synaptiques

- Les nouveaux poids, au temps *t+1*, sont:

$$w_{i,j,k}(t+1) = w_{i,j,k}(t) + \Delta w_{i,j,k}(t)$$

- où:

$$\Delta w_{i,j,k}(t) = (\lambda \cdot \delta_{i+1,k} \cdot v_{i,j}) + \mu \cdot \Delta w_{i,j,k}(t-1)$$



Apprentissage supervisé

- Nous avons un jeu de données avec des entrées et des sorties recherchées
- Une génération : recalculer les poids pour chaque paire d'entrée / sortie
- Entraînement complet = 10350 générations
  - nous tardons 14 heures pour entraîner le réseau (code python)
- Pour chaque génération du processus d'entraînement, les entrées sont réordonnées à l'hasard (pour que l'ordre des données n'affecte pas l'entraînement)

Exemple de résultat (sortie du module d'Impact)

```
Neural Network Output (close to 1 is better):
Windows NT4: 4.87480503763e-005
Editions:
    Enterprise Server: 0.00972694324639
    Server: -0.00963500026763
Service Packs:
    6: 0.00559659167371
    6a: -0.00846224120952
Windows 2000: 0.996048928128
Editions:
    Server: 0.977780526016
    Professional: 0.00868998746624
    Advanced Server: -0.00564873813703
Service Packs:
    4: -0.00505441088081
    2: -0.00285674134367
    3: -0.0093665583402
    0: -0.00320117552666
    1: 0.921351036343
```

Exemple de résultat (sortie du module d'Impact) (cont.)

```
Windows 2003: 0.00302898647853
Editions:
    Web Edition: 0.00128127138728
    Enterprise Edition: 0.00771786077082
    Standard Edition: -0.0077145024893
Service Packs:
    0: 0.000853988551952
Windows XP: 0.00605168045887
Editions:
    Professional: 0.00115635710749
    Home: 0.000408057333416
Service Packs:
    2: -0.00160404945542
    0: 0.00216065240615
    1: 0.000759109188052
Setting OS to Windows 2000 Server sp1
Setting architecture: i386
```

Comparaison des résultats

- Résultats de notre laboratoire:

	Vieux module DCE-RPC	DCE-RPC avec réseau de neurones
Concordance parfaite	6	7
Concordance partielle	8	14
Erreur	7	0
Pas de réponse	2	2

1. Introduction

2. Mappeur d’endpoint DCE-RPC

3. Détection d’OS basée sur les signatures de Nmap

4. Réduction des dimensions et entraînement

Les tests de Nmap

- Nmap est un outil d'exploration du réseau et un scanner de sécurité
- Inclut la détection d'OS basée sur la réponse d'une machine aux 9 tests:

Test	envoyer paquet	à un port	avec les flags
T1	TCP	TCP ouvert	SYN, ECN-Echo
T2	TCP	TCP ouvert	aucun flag
T3	TCP	TCP ouvert	URG, PSH, SYN, FIN
T4	TCP	TCP ouvert	ACK
T5	TCP	TCP fermé	SYN
T6	TCP	TCP fermé	ACK
T7	TCP	TCP fermé	URG, PSH, FIN
PU	UDP	UDP fermé	
TSeq	TCP * 6	TCP ouvert	SYN

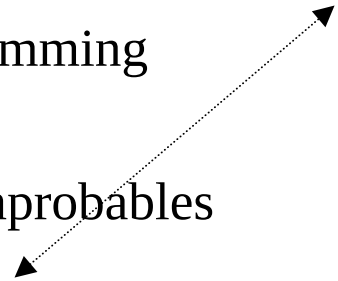
La base de signatures de Nmap

- Notre método utilise la base de signatures de Nmap
- Une signature est un ensemble de règles décrivant comment une version / édition spécifique d'un système d'exploitation répond aux tests. Par exemple:

```
# Linux 2.6.0-test5 x86
Fingerprint Linux 2.6.0-test5 x86
Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<2D3CFA0&>73C6B%IPID=Z%TS=1000HZ)
T1(DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%UL
  EN=134%DAT=E)
```

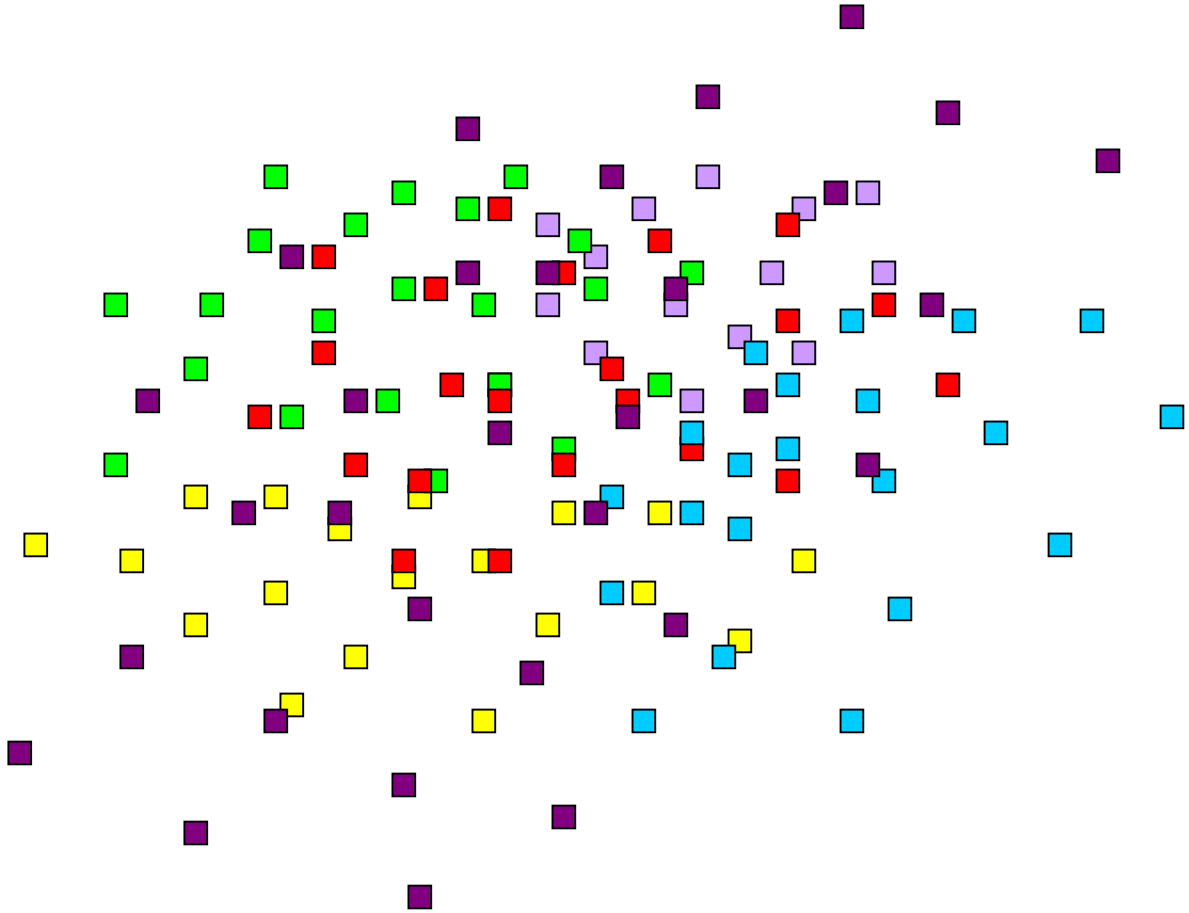
Richesse et faiblesse de Nmap

- La base de Nmap contient 1684 signatures
- Nmap fonctionne en comparant la réponse d'une machine avec chaque signature de la base de données :
  - un score est assigné à chaque signature
  - score = nombre de règles qui concordent / nombre de règles considérées
  - "best fit" basé sur une distance de Hamming
- Problème : les systèmes d'exploitation improbables
  - génèrent moins de réponses aux tests
  - et obtiennent un meilleur score!
  - p.ex. un Windows 2000 détecté comme Atari 2600 ou HPUX …



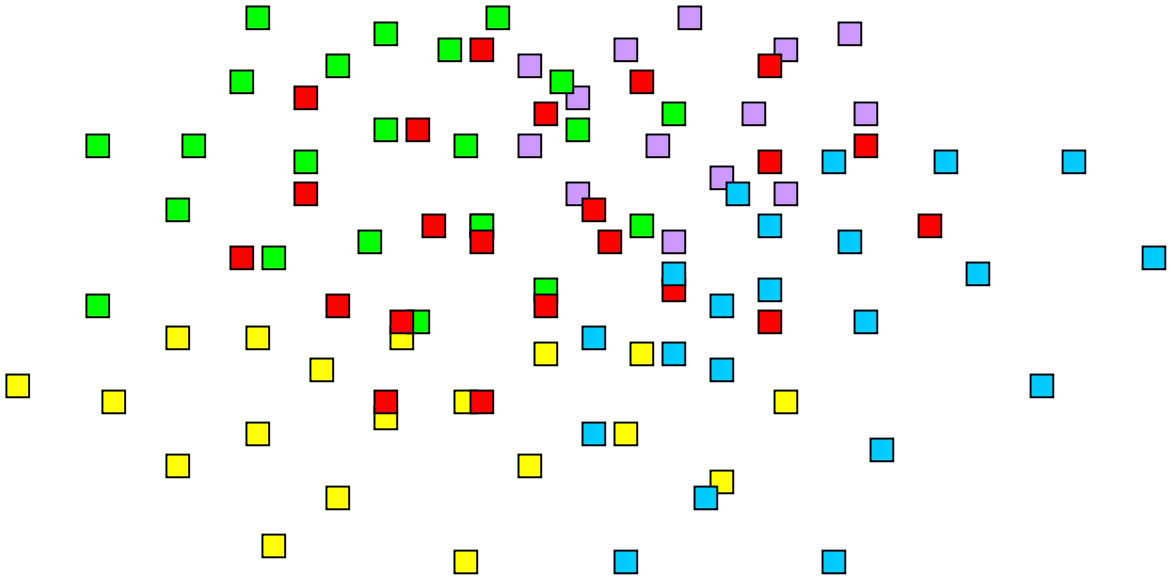
Representation symbolique de l'espace d'OS

- L'espace des réponses de machines a 560 dimensions
- Les couleurs représentent différentes familles d'OS

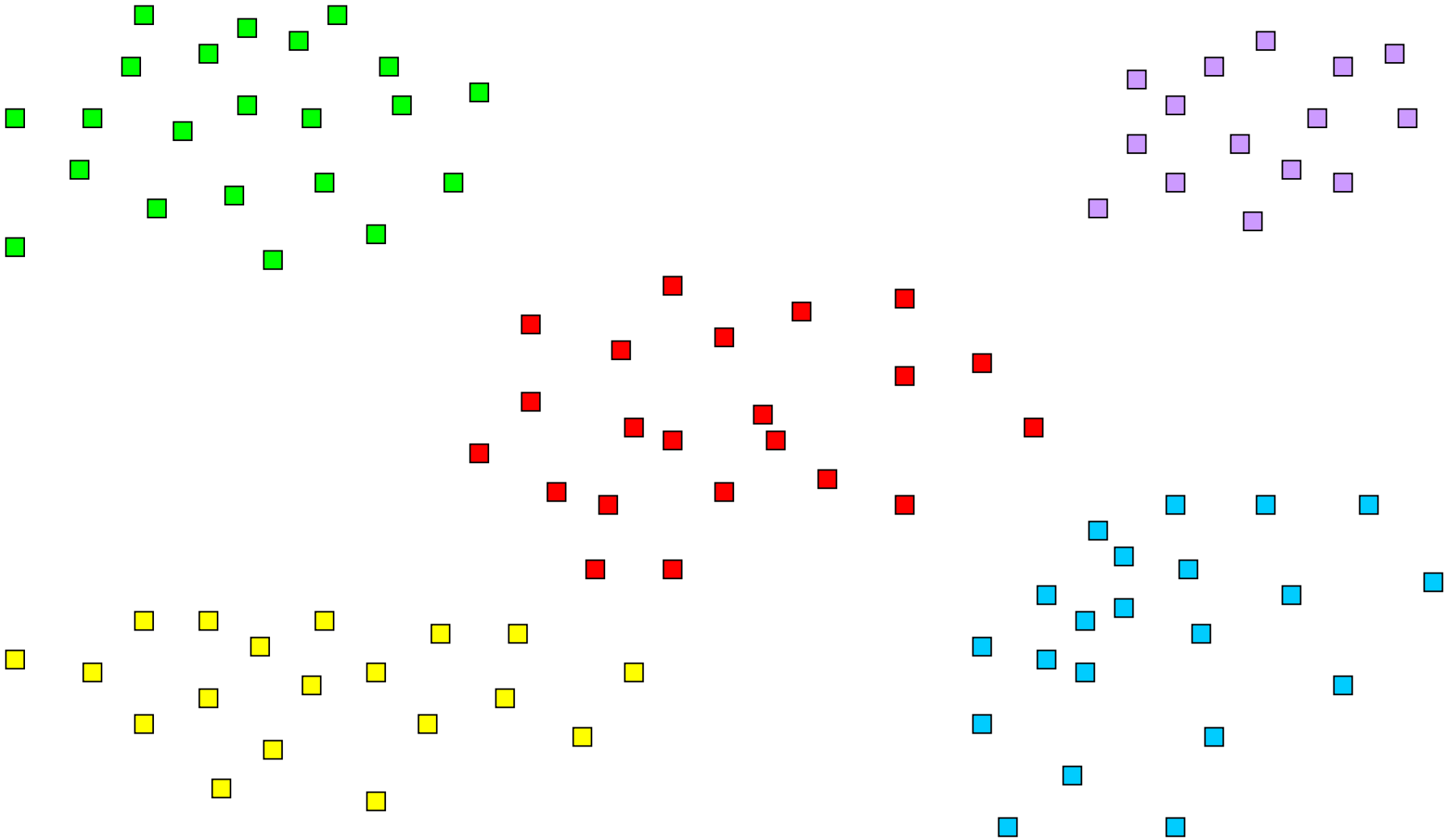


Représentation après avoir filtré les OS non pertinents

- La détection d'OS est une étape d'un test d'intrusion
  - nous voulons seulement détecter Windows, Linux, Solaris, OpenBSD, FreeBSD, NetBSD

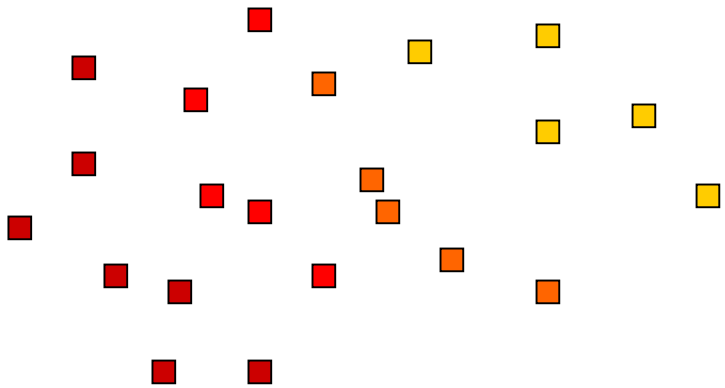


Représentation après avoir séparé les familles d'OS



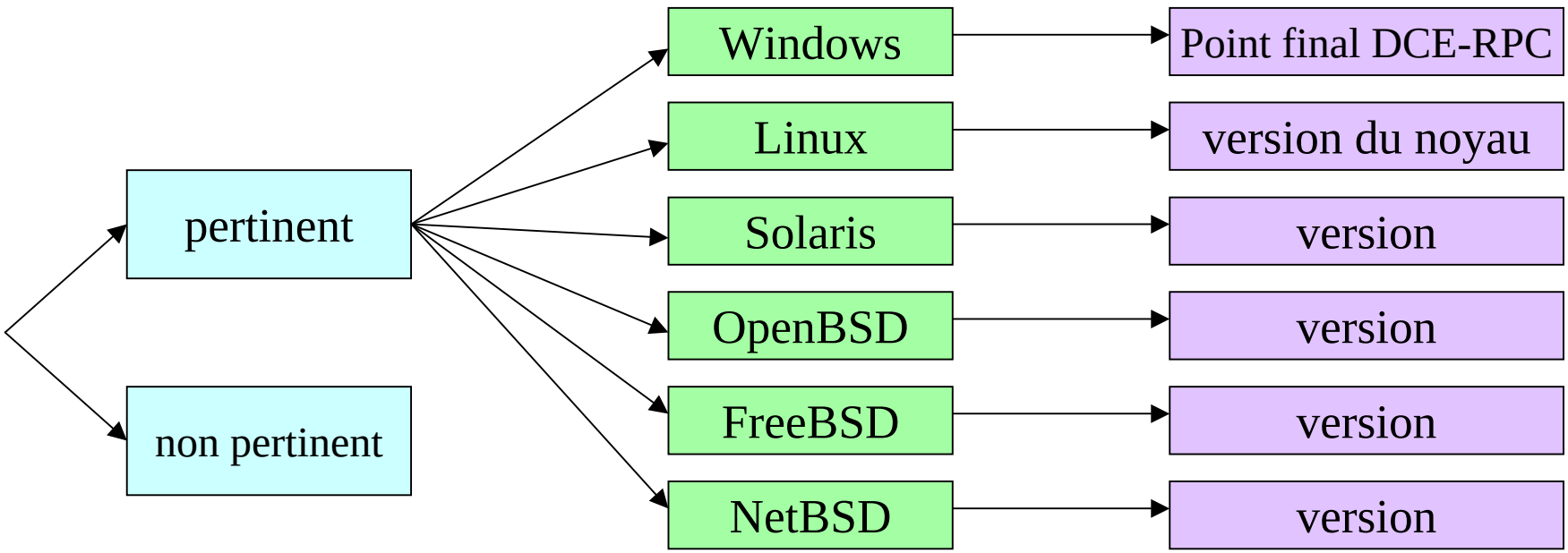
Distinguer les versions au sein de chaque famille

- L'analyse pour distinguer les différentes versions est réalisée une fois que nous connaissons la famille
  - par exemple, nous savons que la machine est un OpenBSD et voulons connaître la version



Structure de Réseaux Hiérarchiques

- Idée : utiliser plusieurs réseaux de neurones organisés de façon hiérarchique



Ainsi nous avons 5 réseaux de neurones…

- Un réseau de neurones pour décider si l'OS est important ou non
- Un réseau de neurones pour décider la famille de l'OS:
  - Windows, Linux, Solaris, OpenBSD, FreeBSD, NetBSD
- Un réseau de neurones pour décider la version de Linux
- Un réseau de neurones pour décider la version de Solaris
- Un réseau de neurones pour décider la version de OpenBSD

- Chaque réseau de neurones requiert une topologie et un entraînement spécial
  - chaque réseau résout un problème plus simple
  - un seul réseau qui résout le problème complet a donné de mauvais résultats

Les entrées du Réseau de Neurones

- Comment assigner un ensemble de neurones d'entrée à chaque test?
- Détails pour les tests T1 … T7:

- un neurone pour le flag ACK
  - un neurone pour chaque réponse: S, S++, O
- un neurone pour le flag DF
  - un neurone pour la réponse: yes/no
- un neurone pour le champs 'Flags'
  - un neurone pour chaque flag: ECE, URG, ACK, PSH, RST, SYN, FIN
- 10 groupes de 6 neurones pour le champs 'Options'
  - nous activons un seul neurone dans chaque groupe suivant l'option

    EOL, MAXSEG, NOP, TIMESTAMP, WINDOW, ECHOED
- un neurone pour le champs W (window size)

Exemple d'entrées du réseau de neurones

- Pour les flags ou les options : l'entrée est 1 ou –1 (présent ou absent)
- Certains ont une entrée numérique
  - le champs W (window size)
  - le champs GCD (plus grand commun diviseur des numéros de séquence initiaux)

- Par exemple, la réponse d'un Linux 2.6.0 :
  `T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)`

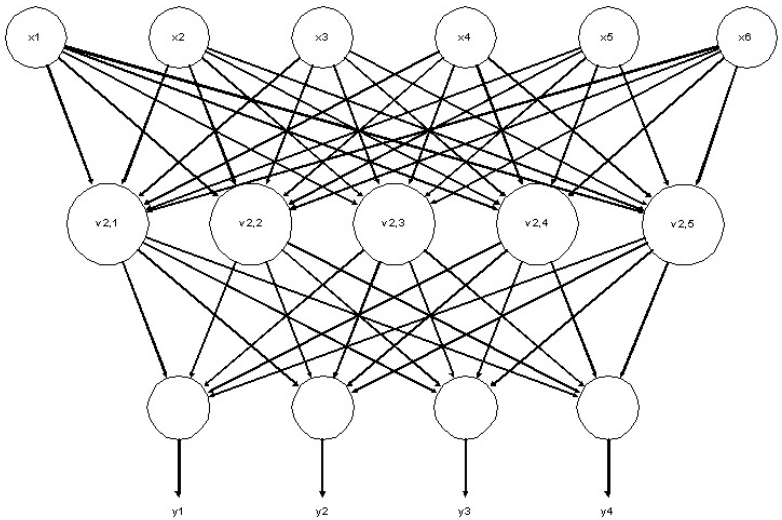
- se transforme en :

ACK	S	S++	O	DF	Yes	Flags	E	U	A	P	R	S	F	…
1	-1	1	-1	1	1	1	-1	-1	1	-1	-1	1	-1	…

La topologie du Réseau de Neurones

- La couche d'entrée a 560 dimensions
  - beaucoup de redondance
  - nous permet de traiter de façon flexible les réponses inconnues
  - mais introduit des problèmes de performance !
  - il est nécessaire de réduire le nombre de dimensions …

- Les réseaux de neurones ont 3 couches, par exemple le premier réseaux de neurones (le filtre de pertinence) contient :



couche d'entrée : 96 neurones

couche cachée : 20 neurones

couche de sortie : 1 neurone

Génération du jeu de données

- Pour entraîner le réseau de neurones nous avons besoin
  - d'entrées (réponses de machines)
  - avec les sorties correspondantes (OS de la machine)
- La base de signatures contient 1684 règles
  - requiert une population de 15000 machines pour entraîner le réseau!
  - nous n'avons pas accès à une telle population …
  - scanner l'Internet n'est pas une option !

- Générer les entrées par une simulation Monte Carlo
  - pour chaque règle, générer des entrées correspondant à cette règle
  - le nombre d'entrées dépend de la distribution empirique des OS
    - » basée sur des données statistiques
  - quand la règle spécifie des options ou un intervalle de valeurs
    - » choisir une valeur en suivant une distribution uniforme

1. Introduction

2. Mappeur d’endpoint DCE-RPC

3. Détection d’OS basée sur les signatures de Nmap

4. Réduction des dimensions et entraînement

Les entrées comme variables aléatoires

- Nous avons été généreux avec les entrées
  - 560 dimensions, avec une redondance importante
  - le jeu de données est très grand
  - la convergence de l'entraînement est lente …

- Considérer chaque dimension d'entrée comme une variable aléatoire X_i
  - les dimensions d'entrée ont des ordres de grandeur différents
    - » les flags prennent comme valeur 1/-1
    - » le champs ISN (numéro de séquence initial) es un entier de 32 bits
  - il faut normaliser les variables aléatoires :

$$\frac{X_i - \mu_i}{\sigma_i}$$

μ_i ← espérance mathématique

σ_i ← écart type

La matrice de corrélation

- Nous calculons la matrice de corrélation *R* :

$$R_{i,j} = \frac{E[(X_i - \mu_i)(X_j - \mu_j)]}{\sigma_i \, \sigma_j}$$

- Après avoir normalisé les variables c'est simplement :

$$R_{i,j} = E[X_i \, X_j]$$

espérance mathématique

- La corrélation est une mesure de la dépendance statistique
  - proche de 1 ou –1 indique une plus grande dépendance
  - la dépendance linéaire entre des colonnes de *R* indique des variables dépendantes
  - nous en gardons une et éliminons les autres
  - les constantes ont une variance nulle et sont aussi éliminées

Exemple de fingerprints d'OpenBSD

```
Fingerprint OpenBSD 3.6 (i386)
Class OpenBSD | OpenBSD | 3.X | general purpose
T1(DF=N%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

Fingerprint OpenBSD 2.2 - 2.3
Class OpenBSD | OpenBSD | 2.X | general purpose
T1(DF=N%W=402E%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=N%W=402E%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=4000%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
```

Champs permettant de distinguer les versions d'OpenBSD

New index	Original index	Field name
0	20	T1:TCP_OPT_1_EOL
1	26	T1:TCP_OPT_2_EOL
2	29	T1:TCP_OPT_2_TIMESTAMP
3	74	T1:W_FIELD
4	75	T2:ACK_FIELD
5	149	T2:W_FIELD
6	150	T3:ACK_FIELD
7	170	T3:TCP_OPT_1_EOL
8	179	T3:TCP_OPT_2_TIMESTAMP
9	224	T3:W_FIELD
10	227	T4:SEQ_S
11	299	T4:W_FIELD
12	377	T6:SEQ_S
13	452	T7:SEQ_S

Champs permettant de distinguer les versions d'OpenBSD (2)

New index	Original index	Field name
14	525	TSeq:CLASS_FIELD
15	526	TSeq:SEQ_TD
16	528	TSeq:SEQ_RI
17	529	TSeq:SEQ_TR
18	532	TSeq:GCD_FIELD
19	533	TSeq:IPID_FIELD
20	535	TSeq:IPID_SEQ_BROKEN_INCR
21	536	TSeq:IPID_SEQ_RPI
22	537	TSeq:IPID_SEQ_RD
23	540	TSeq:SI_FIELD
24	543	TSeq:TS_SEQ_2HZ
25	546	TSeq:TS_SEQ_UNSUPPORTED
26	555	PU:UCK_RID_RIPCK_EQ
27	558	PU:UCK_RID_RIPCK_ZERO

Analyse en Composantes Principales (ACP)

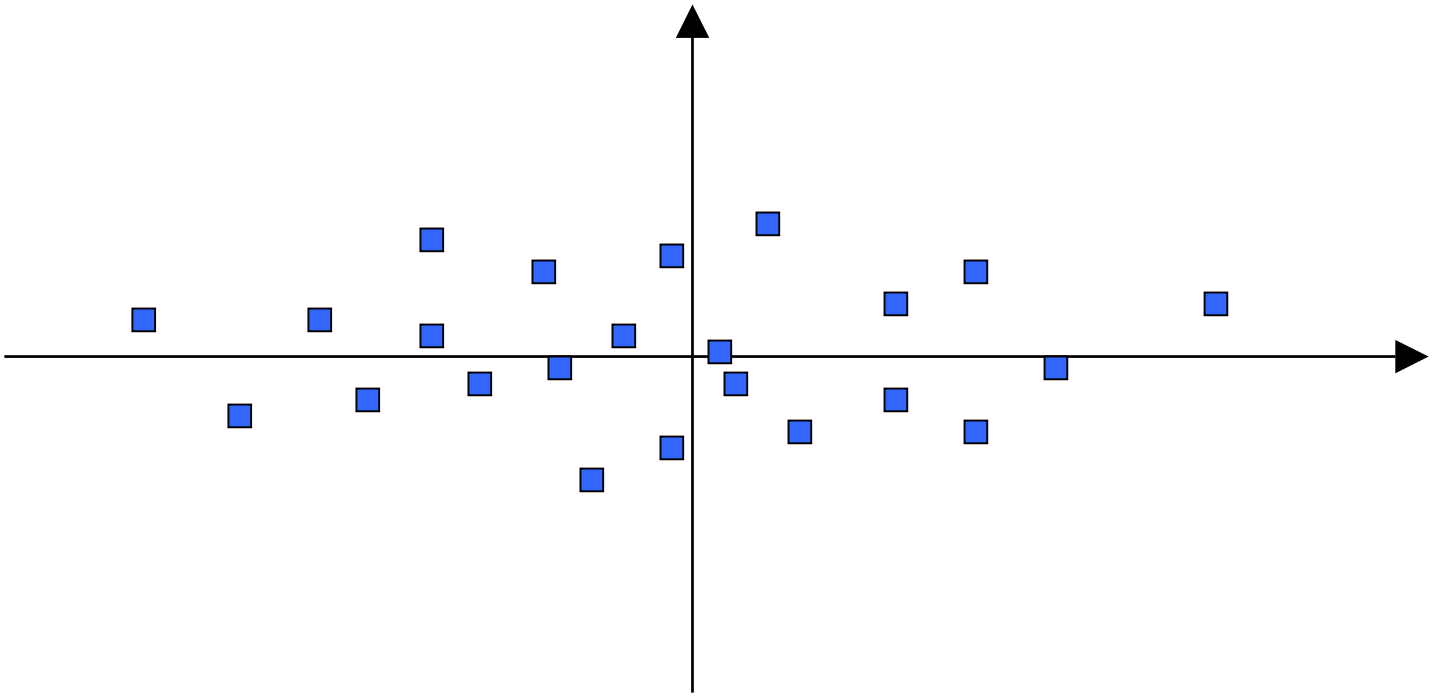
- Réduction ultérieure en utilisant l'Analyse en Composantes Principales (ACP)

- Idée : calculer une nouvelle base (système de coordonnées) de l'espace d'entrée
  - la majeure variance de toute projection du jeu de données dans un sous-espace de k dimensions
  - provient de projeter sur les k premiers vecteurs de cette base

- Algorithme ACP :
  - calculer les vecteurs propres et valeurs propres de R
  - trier par valeur propre décroissante
  - garder les k premiers vecteurs pour projeter les données
  - le paramètre k est choisi pour maintenir 98% de la variance totale

Idée de l'Analyse en Composantes Principales

- Garder les dimensions qui ont une plus grande variance
  - valeurs propres les plus hautes de la matrice de corrélation



Topologie des Réseaux de Neurones résultants

- Après avoir réalisé l'ACP nous avons obtenu les topologies suivantes pour les réseaux de neurones (la taille de l'entrée originale était de 560 dans tous les cas)

Analyse	Couche d'entrée (après réduction de la matrice R)	Couche d'entrée (après ACP)	Couche cachée	Couche de sortie
Pertinence	204	96	20	1
Système d'exploitation	145	66	20	6
Linux	100	41	18	8
Solaris	55	26	7	5
OpenBSD	34	23	4	3

Taux d'apprentissage adaptatif

- Stratégie pour accélérer la convergence de l'entraînement

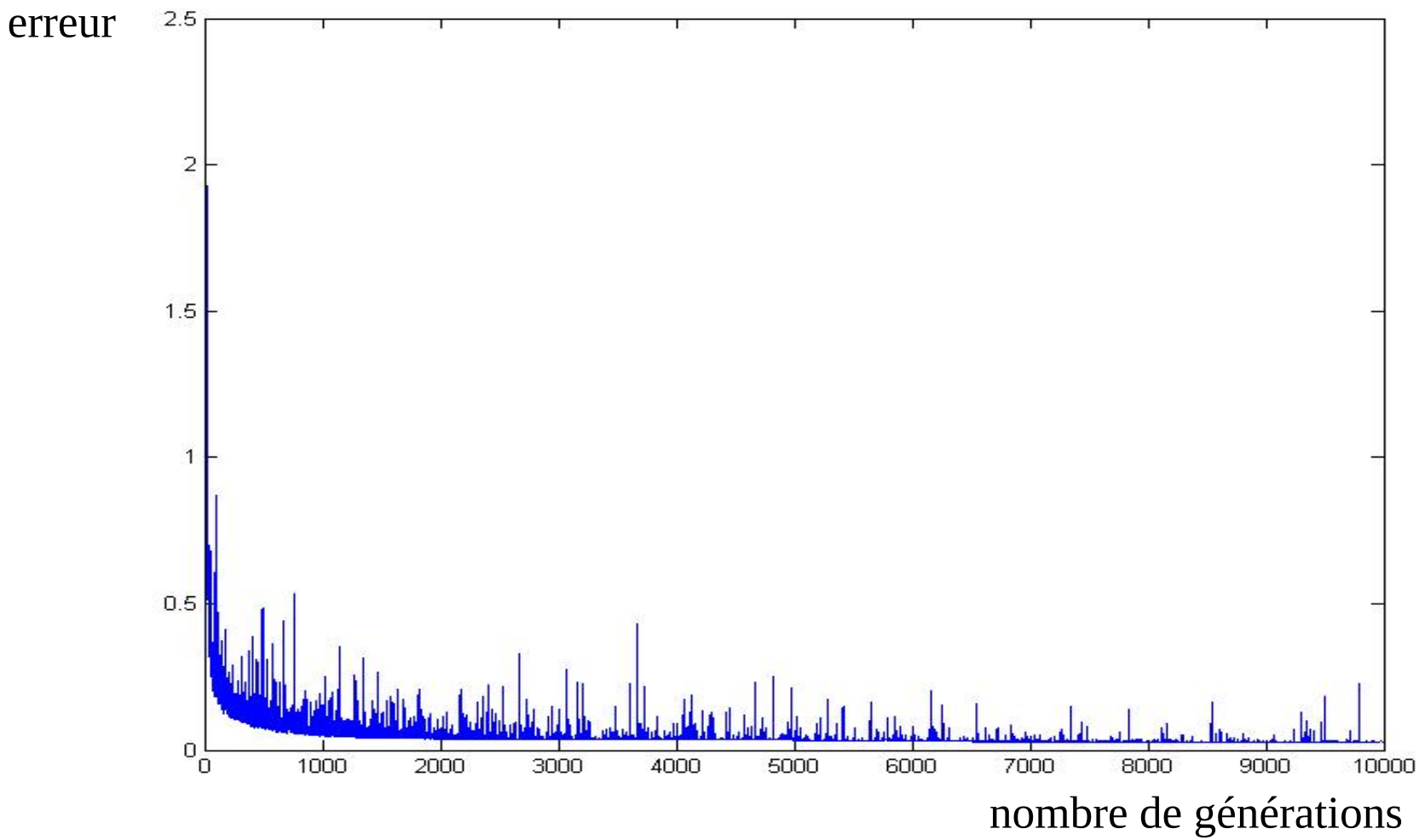
- Calculer une estimation de l'erreur quadratique
  (y_i sont les sorties recherchée, v_i sont les sorties du réseau):

$$\frac{\sum_{i=1}^{n}(y_i - v_i)^2}{n}$$

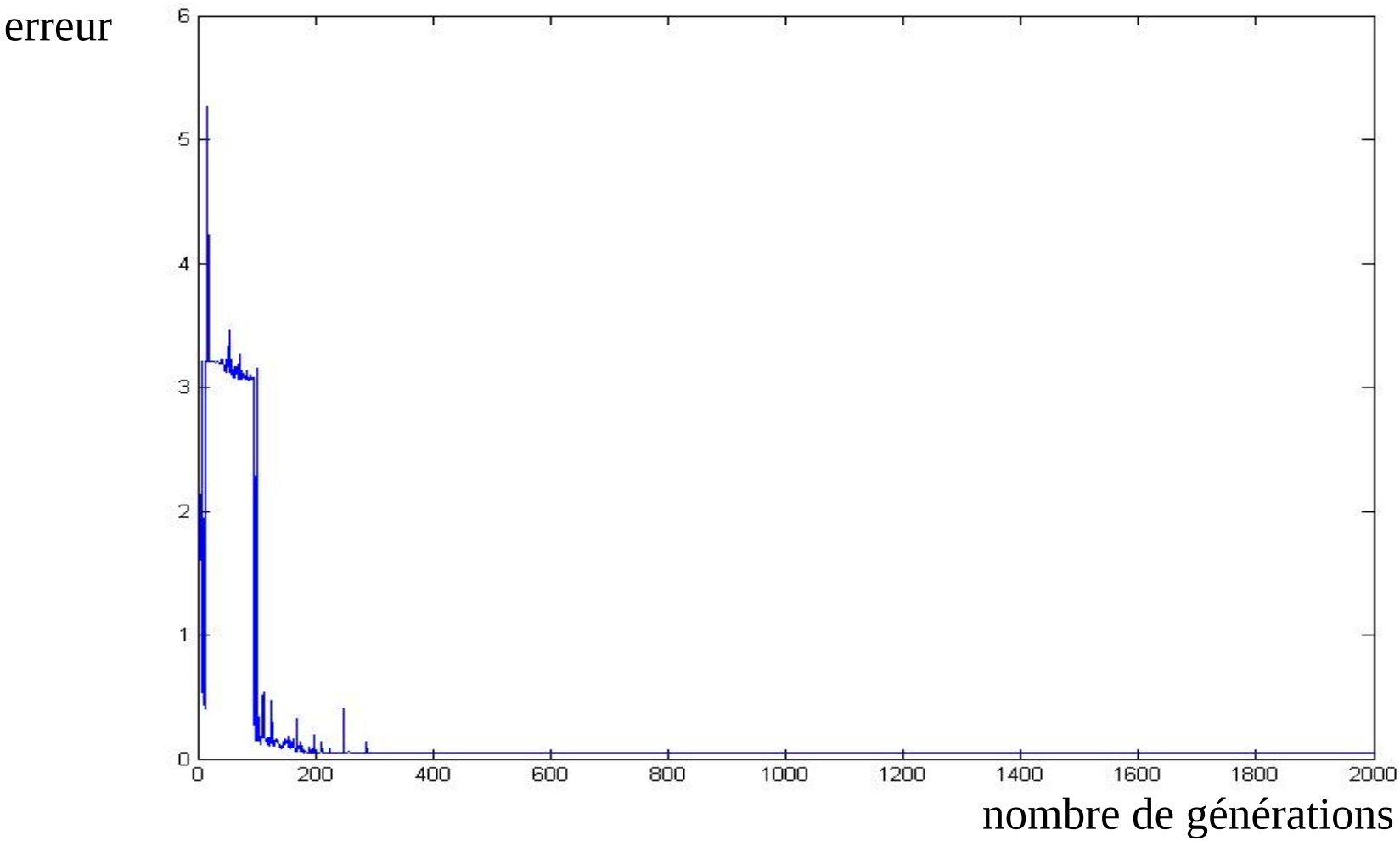
- Après chaque génération (après avoir fait les calculs pour toutes les paires d'entrée / sortie)
  - si l'erreur est plus petite, alors nous augmentons le taux d'apprentissage
  - si l'erreur est plus grande, alors nous diminuons le taux d'apprentissage

- Idée : se déplacer plus rapidement si nous allons dans la direction correcte

Évolution de l'erreur (taux d'apprentissage fixe)



Évolution de l'erreur (taux d'apprentissage adaptatif)



Entraînement par sous-ensembles du jeu de données

- Autre stratégie pour accélérer la convergence de l'entraînement
- Entraîner le réseau avec plusieurs sous-ensembles plus petits du jeu de données
- Pour estimer l'erreur, nous calculons une mesure d'adéquation G
  - si la sortie est 0/1 :

    G = 1 – (Pr[faux positif] + Pr[faux négatif])
  - autres sorties :

    G = 1 – nombre d'erreurs / nombre de sortie
- Taux d'apprentissage adaptatif :
  - si la mesure d'adéquation G augmente, alors nous augmentons le taux d'apprentissage initial

Exemple de résultats (machine avec Solaris 8)

- Relevant / not relevant analysis

0.9999999999999789	**relevant**

- Operating System analysis

-0.9999999999999434	Linux
0.9999999921394744	**Solaris**
-0.9999999999998057	OpenBSD
-0.9999964651426454	FreeBSD
-1.0000000000000000	NetBSD
-1.0000000000000000	Windows

- Solaris version analysis

0.9817278032507448 2	**Solaris 8**
-0.9928138245833577 6	Solaris 9
-0.9935758690614388 0	Solaris 7
-0.9998378968003799	Solaris 2.X
-0.9999999977837983	Solaris 2.5.X

Idées pour le futur (1)

- Analyser les éléments clefs des tests de Nmap
  - grâce à l'analyse des poids de convergence du réseau
  - grâce à la réduction de la matrice de corrélation
  - grâce à l'Analyse des Composants Principaux
- Optimiser Nmap pour générer moins de trafic
- Ajouter du bruit et le filtre d'un firewall
  - détecter la présence du firewall
  - identifier différents firewalls
  - faire des tests plus robustes

Idées pour le futur (2)

- Cette analyse peut s'appliquer à d'autres méthodes de détection:
- xprobe2 – Ofir Arkin, Fyodor Yarochkin & Meder Kydyraliev
  - détection par ICMP, SMB, SNMP
- p0f (Passive OS Identification) – Michal Zalewski
- détection d'OS par SUN RPC / Portmapper
  - Sun / Linux / autres versions de System V
- détection de MUA (Outlook / Thunderbird / etc) en utilisant les Mail Headers

Merci!

javier.burroni@coresecurity.com
carlos.sarraute@coresecurity.com

http://www.coresecurity.com/corelabs/