

NVIDIA - Drive PX

Deep learning
via GPU acceleration

BOSSHARD
LAROCHE
TOURNIAIRE
VIANES

Florentin
Richard
Thomas
Arthur

X IESE 5
Option CSI
15 janvier 2018

Sommaire

- I. Réseau de neurone
 - ✗ Reconnaissance de chiffre
 - ✗ Opérations nécessaires
 - ✗ Phase d'apprentissage
- II. DRIVE PX 2
 - ✗ Série DRIVE PX
 - ✗ Caractéristique et Architecture
- III. Implémentation



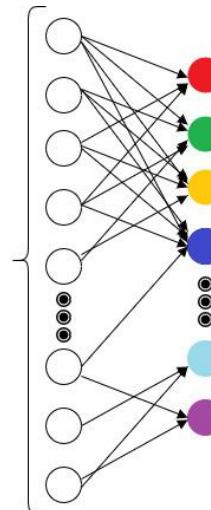
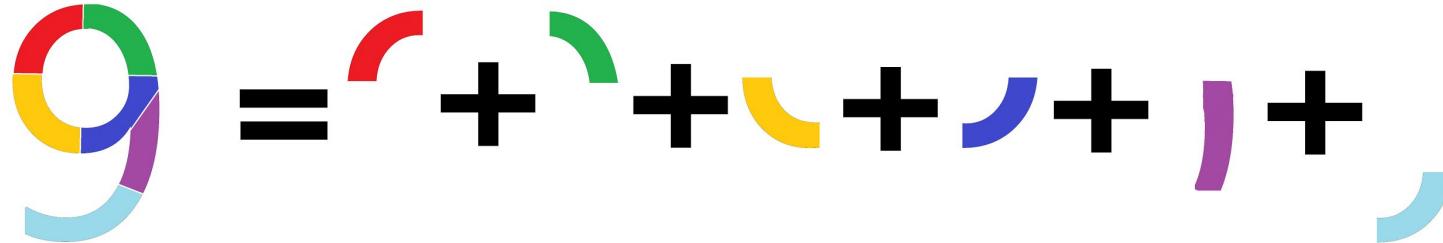
Réseau de neurones

Approche intuitive
Reconnaissance de chiffre

Reconnaissance de chiffre

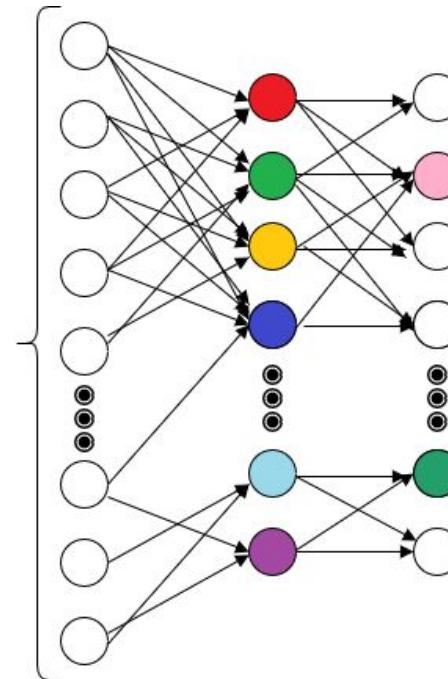
9 ?

Reconnaissance de chiffre - Intuition



Reconnaissance de chiffre - Intuition

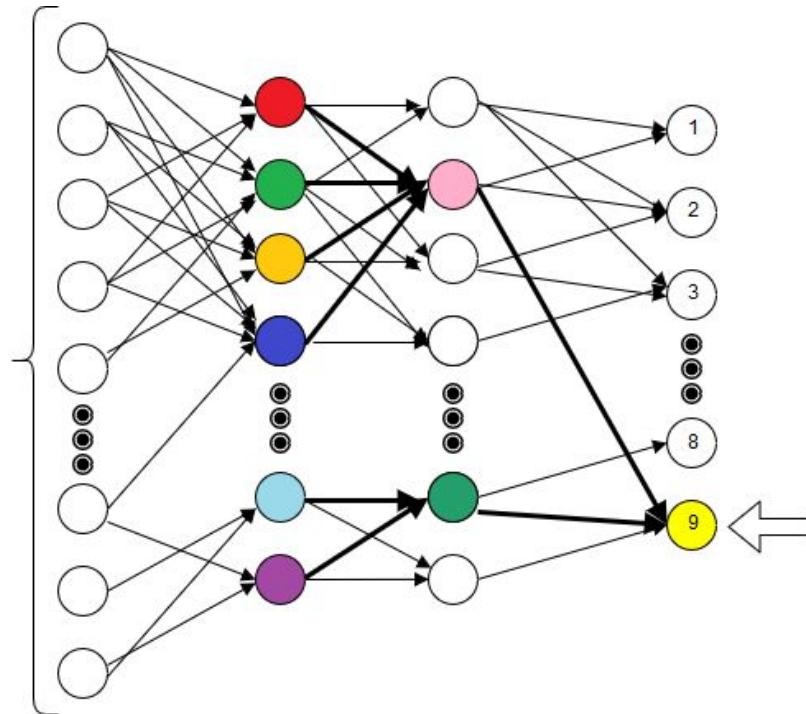
$$O = \textcolor{red}{\lrcorner} + \textcolor{green}{\lrcorner} + \textcolor{yellow}{\lrcorner} + \textcolor{blue}{\lrcorner}$$
$$\lrcorner = \textcolor{violet}{|} + \textcolor{cyan}{\lrcorner}$$



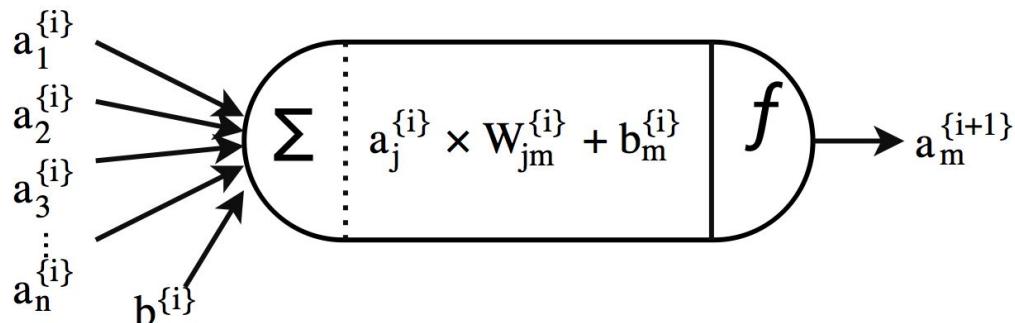
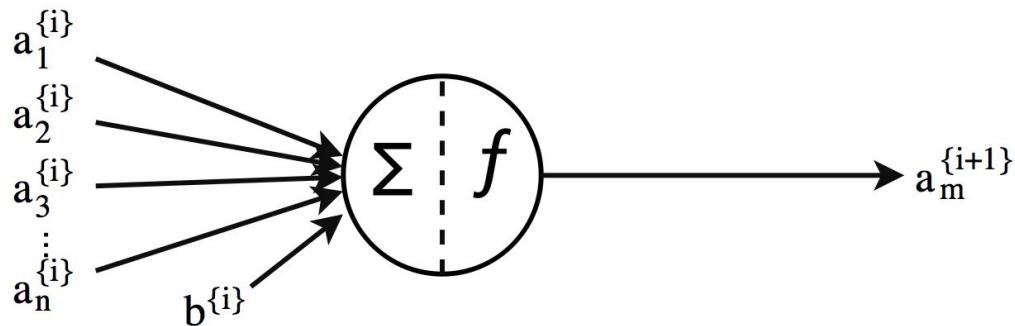
Reconnaissance de chiffre - Intuition

Réseau de neurones

- Entrée : Image
- Première couche : Pattern
- Seconde couche : Combinaison de pattern
- Sortie : Chiffre



Un neurone



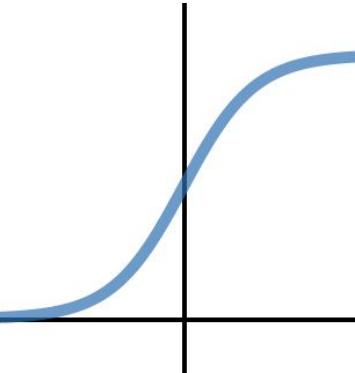
$$f(a^{(i)} \times W^{(i)} + b^{(i)}) = a^{(i+1)}$$

$$W^{(i)} = \begin{pmatrix} w_{11}^{(i)} & w_{12}^{(i)} & \cdots & w_{1m}^{(i)} \\ w_{21}^{(i)} & \ddots & & \\ \vdots & & \ddots & \\ w_{n1}^{(i)} & & & w_{nm}^{(i)} \end{pmatrix}$$

$$a^{(i)} = \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_n^{(1)} \end{pmatrix}$$

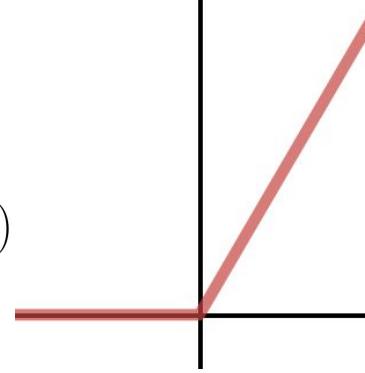
Fonctions d'activations

Sigmoid

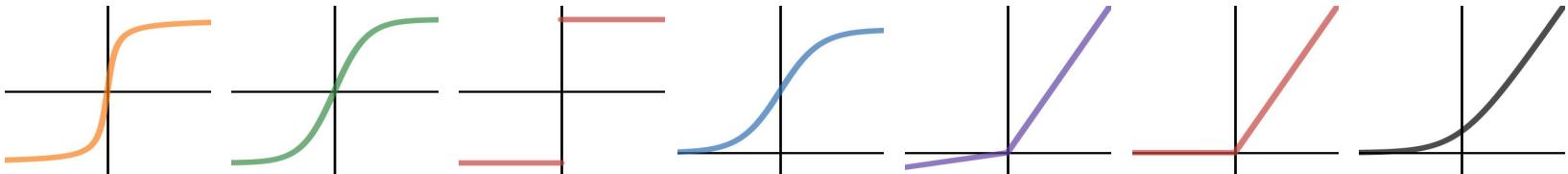

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x) \times (1 - f(x))$$

ReLU


$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{sinon} \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{sinon} \end{cases}$$



Opérations nécessaires

$$a^{\{i+1\}} = f(a^{\{i\}} \times W^{\{i\}} + b^{\{i\}})$$



Fonction
d'activation



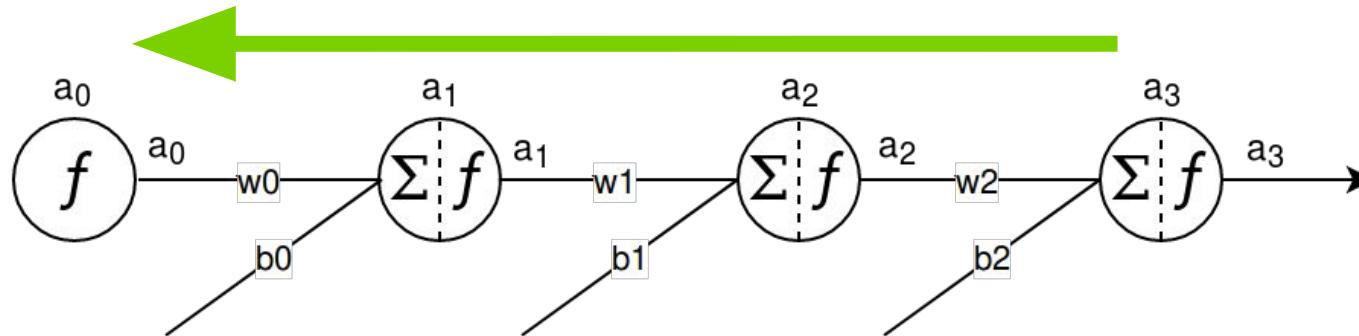
Produit de
matrice



Somme de
matrice

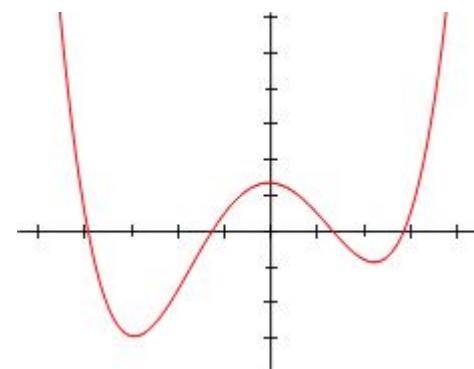
Phase d'apprentissage

Back propagation - Descente de gradient



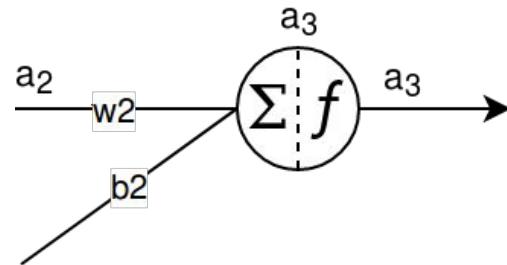
But :

- ✖ Minimiser l'erreur
- ✖ $E = (a_3 - y)^2$
- ✖ Redéfinir les coefficients



Phase d'apprentissage

Back propagation - Descente de gradient



Variation de w_2 sur la sortie :

$$E = (a_3 - y)^2 \Rightarrow \frac{\partial E}{\partial a_3} = 2(a_3 - y)$$

$$a_{i+1} = f(z_i) \Rightarrow \frac{\partial a_{i+1}}{\partial z_i} = f'(z_i)$$

$$z_i = a_i w_i + b_i \Rightarrow \frac{\partial z_i}{\partial w_i} = a_i$$

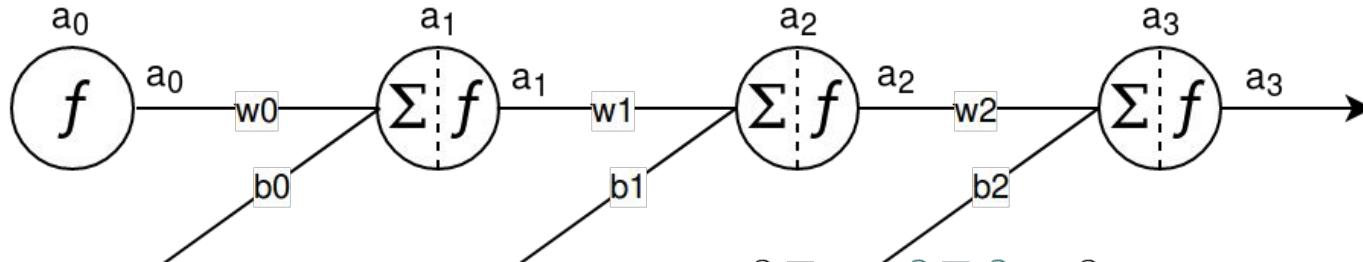
$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_3} \frac{\partial a_3}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = 2(a_3 - y) f'(z_2) a_2$$

Phase d'apprentissage

Back propagation - Descente de gradient



- Variation de w_2 sur la sortie :

$$\frac{\partial E}{\partial w_2} = \underbrace{\frac{\partial E}{\partial a_3} \frac{\partial a_3}{\partial z_2}}_{\epsilon_3} \frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \epsilon_3 \frac{\partial z_2}{\partial w_2}$$

- Variation de w_1 sur la sortie :

$$\frac{\partial E}{\partial w_1} = \epsilon_3 \underbrace{\frac{\partial z_2}{\partial a_2} \frac{\partial a_2}{\partial z_1}}_{\epsilon_2} \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \epsilon_3 \epsilon_2 \frac{\partial z_1}{\partial w_1}$$

Drive PX2



Tegra X2
Architecture Pascal

Série Drive PX

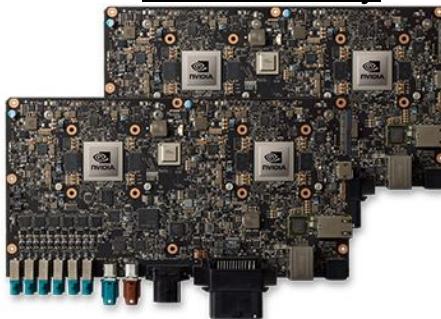
AutoCruise



AutoChauffeur



Full Autonomy



Technologies d'architectures

- ✖ Chronologie : Fermi, Kepler, Maxwell, Pascal, Volta
- ✖ Multiplication du parallélisme

Spécificités des cartes

- ✖ Drive CX, Drive PX, Drive PX 2 (AutoCruise ou Tesla)
- ✖ Drive PX 2 AutoChauffeur
- ✖ Drive PX Xavier AutoChauffeur
- ✖ Drive PX Pegasus : Level 5 Self-Driving Hardware

Drive PX2 AutoChauffeur

Performance et Informations générales

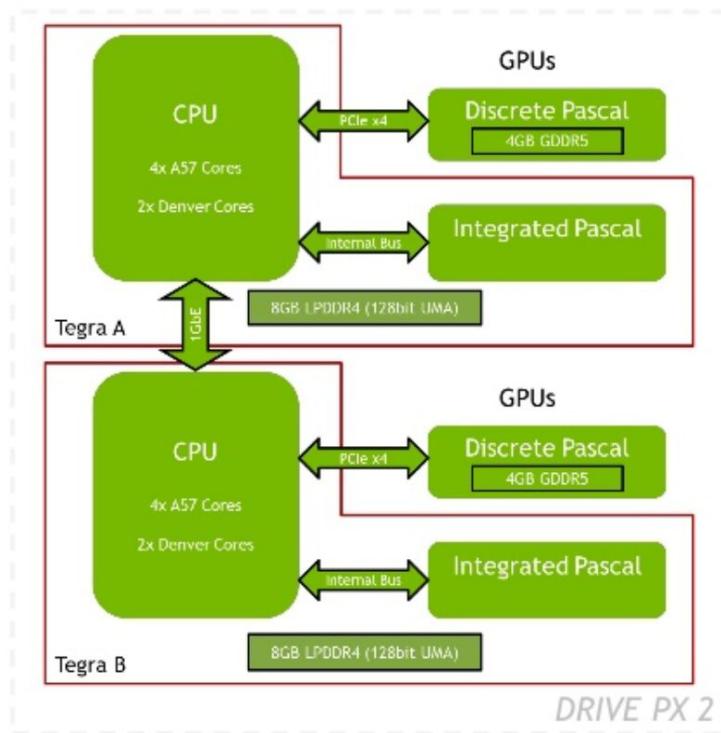
DRIVE PX 2 (Autochauffeur)

- ✗ Janvier 2016
- ✗ Intègre **12 coeurs** de CPU
- ✗ **2 GPU Pascal** dédiés
- ✗ Puissance de calculs:
8 TFLOPS
24 DL TOPS
- ✗ Taille d'intégration: 16 nm
- ✗ Puissance consommée: 250 W

Comparatif:

- ✗ **DRIVE PX 2 (autocruise)**
 - ✗ Septembre 2016
 - ✗ Intègre **6 coeurs** de CPU
 - ✗ Puissance de calcul **1,3 TFLOPS**
 - ✗ Puissance consommée: 10 W
- ✗ **DRIVE PX**
 - ✗ Janvier 2015
 - ✗ Intègre **2 CPU Tegra X1**
 - ✗ Puissance de calcul:
2,3 TFLOPS
1,3 Gpix/s

Architecture global DRIVE PX 2 (Autochauffeur)



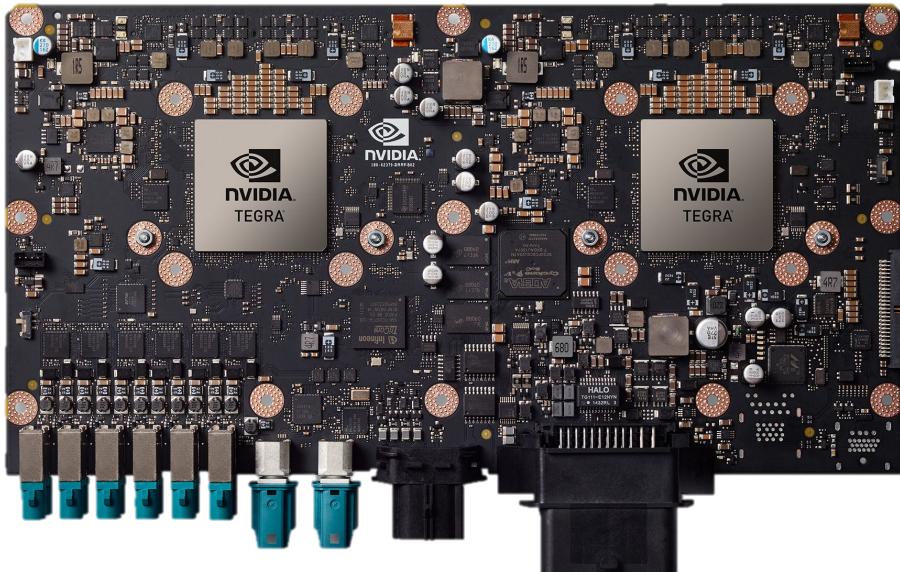
2 CPU Tegra X2

- × CPU (4x A57 cores et 2x Denver cores)
- × GPU Pascal intégré
- × 8 GB RAM LPDDR4 (externe)
- × Communication : 1 GB ethernet (CPU)
- × Communication : bus interne

2 GPU Pascal dédiés (GP10B)

- × 4 GB RAM GDDR5 (externe)
- × Communication : PCIe x4 (8 GB/s)

Architecture global DRIVE PX 2 (Autochauffeur)



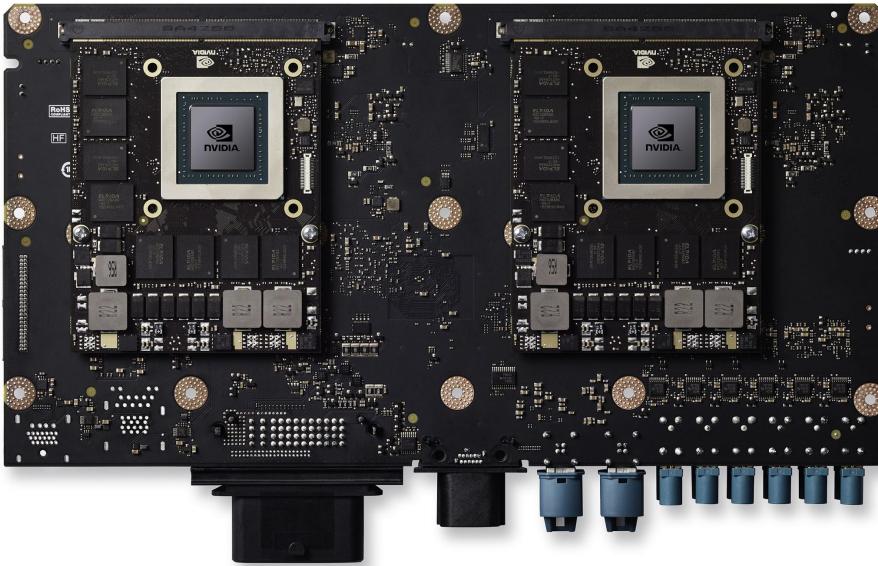
2 CPU Tegra X2

- ✗ CPU (4x A57 cores et 2x Denver cores)
- ✗ GPU Pascal intégré
- ✗ 8 GB RAM LPDDR4 (externe)
- ✗ Communication : 1 GB ethernet (CPU)
- ✗ Communication : bus interne

2 GPU Pascal dédiés (GP10B)

- ✗ 4 GB RAM GDDR5 (externe)
- ✗ Communication : PCIe x4 (8 GB/s)

Architecture global DRIVE PX 2 (Autochauffeur)



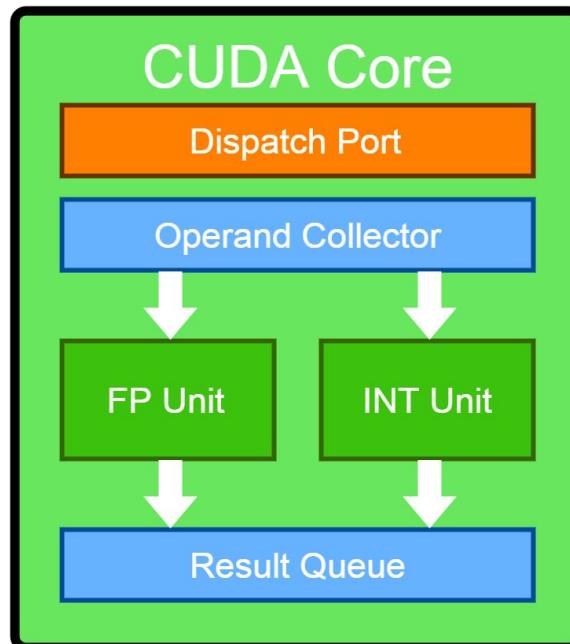
2 CPU Tegra X2

- ✗ CPU (4x A57 cores et 2x Denver cores)
- ✗ GPU Pascal intégré
- ✗ 8 GB RAM LPDDR4 (externe)
- ✗ Communication : 1 GB ethernet (CPU)
- ✗ Communication : bus interne

2 GPU Pascal dédiés (GP10B)

- ✗ 4 GB RAM GDDR5 (externe)
- ✗ Communication : PCIe x4 (8 GB/s)

Architecture d'un cœur CUDA et SFU



Cœur CUDA:

- ✖ Unité de calcul de base GPU Nvidia
- ✖ Architecture simple
- ✖ Sans cache
- ✖ Sans banc de registre

Special Function Unit :

- ✖ Calcul de **fonction spéciales** en 1 cycle
- ✖ Précision moins grande (**FP32**)
- ✖ SM Pascal GP104 :
32 Résultats/cycle (32 SFUs)
- ✖ Activation (nvcc) : `-use_fast_math`

Architecture d'un cœur CUDA et SFU

Operator/Function	Device Function
x/y	<code>_fdividef(x,y)</code>
$\sin(x)$	<code>_sinf(x)</code>
$\cos(x)$	<code>_cosf(x)</code>
$\tan(x)$	<code>_tanf(x)</code>
$\sin\cos(x, \text{sptr}, \text{cptr})$	<code>_sincosf(x,sptr,cptr)</code>
$\log(x)$	<code>_logf(x)</code>
$\log_2(x)$	<code>_log2f(x)</code>
$\log_{10}(x)$	<code>_log10f(x)</code>
$\exp(x)$	<code>_expf(x)</code>
$\exp_{10}(x)$	<code>_exp10f(x)</code>
$\text{pow}(x,y)$	<code>_powf(x,y)</code>

Cœur CUDA:

- Unité de calcul de base GPU Nvidia
- Architecture simple
- Sans cache
- Sans banc de registre

Special Function Unit :

- Calcul de fonction spéciales en 1 cycle
- Précision moins grande (FP32)
- SM Pascal GP104 :
32 Résultats/cycle (32 SFUs)
- Activation (nvcc) : `-use_fast_math`

Fonctionnement de la mémoire



Architecture Pascal (GP104) :

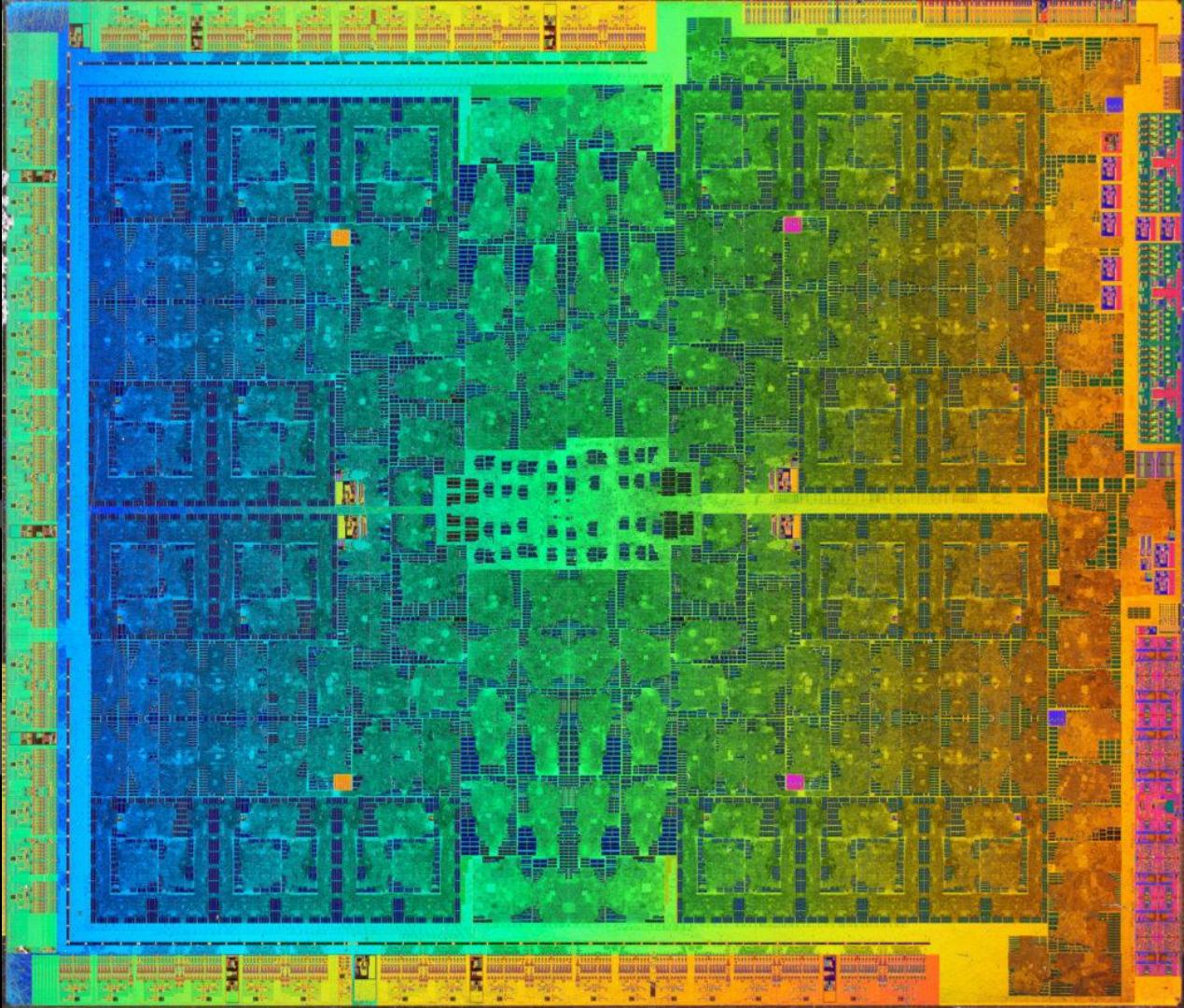
- ✖ Mémoires
 - ✖ Mémoire partagée dédié **96 Ko** commun dans un SM
 - ✖ Cache L1 :
Commun dans un SM
48 Ko, ON/OFF (Maxwell)
 - ✖ Cache L2 :
Commun pour tous les SM
2048 Ko
 - ✖ Unitées (par SM)
 - ✖ **128 coeurs CUDA (2560 au total)**
 - ✖ **32 unitées lecture/écriture mémoire**
 - ✖ **32 SFUs**
 - ✖ **4 unitées de calculs FP64**
- Evolution : Pas de unitées FP64 sur Maxwell

Fonctionnement de la mémoire



Architecture Pascal (GP104) :

- Mémoires
 - Mémoire partagée dédié **96 Ko** commun dans un SM
 - Cache L1 :
 - Commun dans un SM **48 Ko**, ON/OFF (Maxwell)
 - Cache L2 :
 - Commun pour tous les SM **2048 Ko**
 - Unitées (par SM)
 - **128 coeurs CUDA (2560 au total)**
 - **32 unitées lecture/écriture mémoire**
 - **32 SFUs**
 - **4 unitées de calculs FP64**
- Evolution : Pas de unitées FP64 sur Maxwell



Et le Deep Learning dans tout ça ?

Parallélisme de coeurs CUDA : *Streaming Multiprocessors*

- Beaucoup de calculs
- Calculs parallèles
- Calculs simples

Special Function Unit (SFUs) :

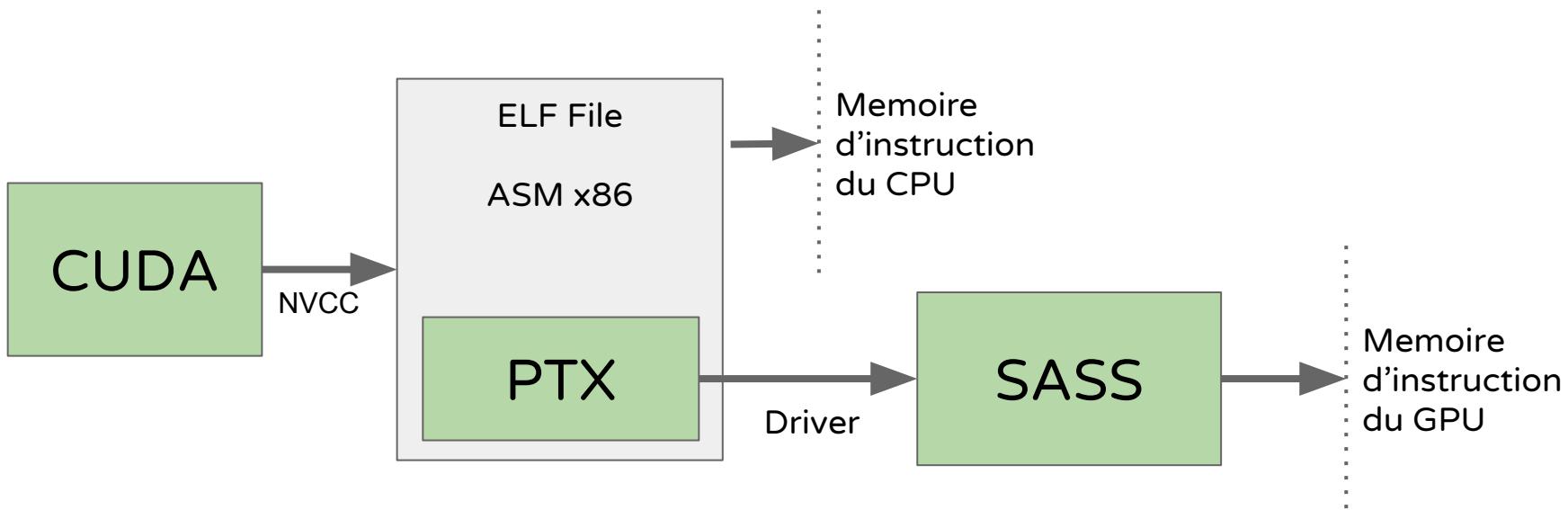
- Calcul de la fonction d'activation



Implementation

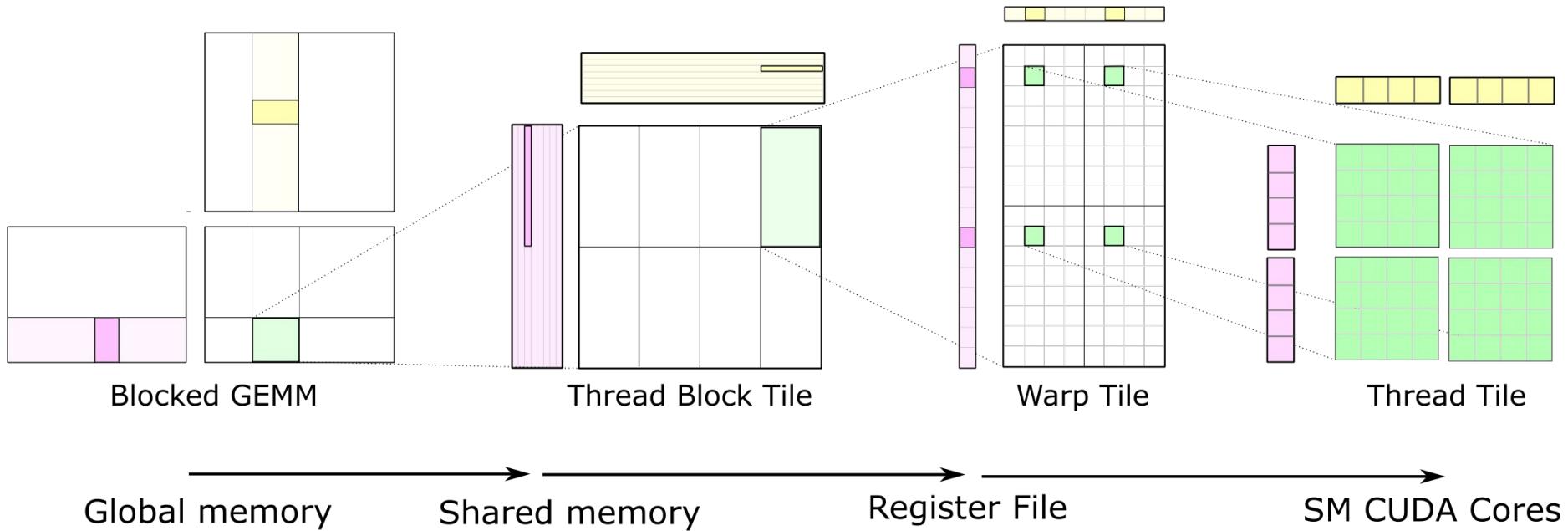
Produits matrices
Fonctions d'activations

Chaine de compilation



Produit de matrice (tenseur)

Optimisation GPU - GEMM



Instructions intéressante

Loop:

```
ld.shared.f32 f2, [rd5 + 0]; // f2 ← Mem[addr1]
ld.shared.f32 f3, [rd8 + 0]; // f3 ← Mem[addr2]
mad.f32       f1, f2, f3, f1; // f1 ← f2*f3 + f1

add.u64       rd5, rd5, rd6; // addr1 ← addr1+size
add.u64       rd8, rd8, rd6; // addr2 ← addr2+size

setp.le.s32   p1, r6, 33791;
@p1 bra        Loop;
bra.uni       Stop_loop;
```

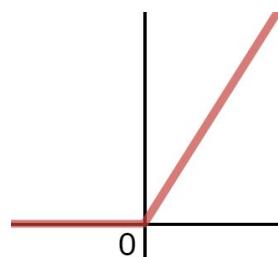
Tenseur Core (Volta)

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} + \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

FSU et fonction d'activation

ReLU

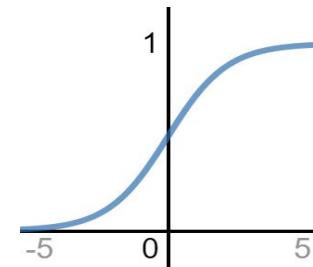
$$f(x) = \max(0, x)$$



VS

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$



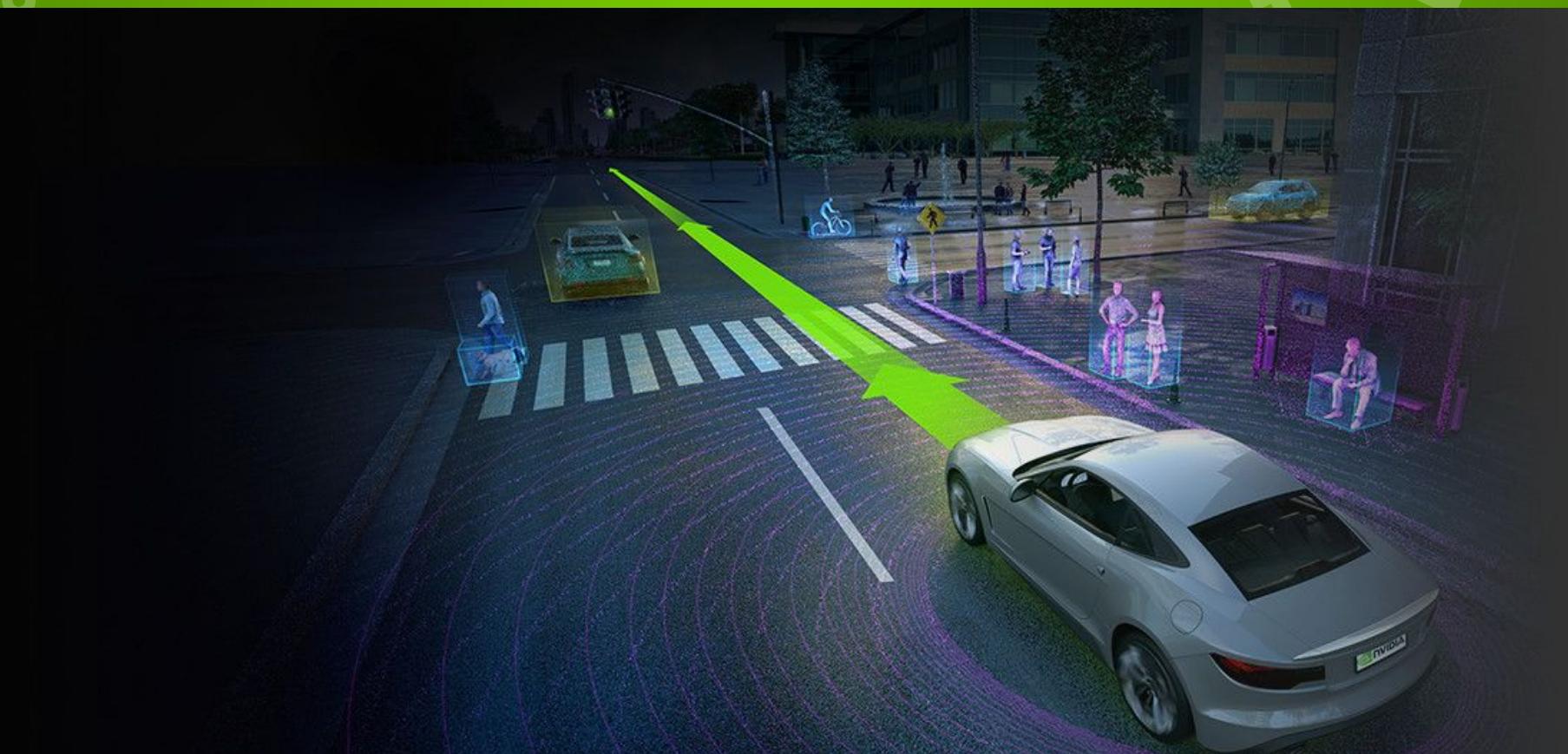
```
// Activation function ReLU
    setp.lt.f32  p, a, 0; //      p ← X<0
@ p mov.f32    s, 0; // Si  p  S ← 0
@!p mov.f32    s, a; // Si !p  S ← a
```

```
// Activation function Sigmoid
ex2.approx.f32 b, a; // B ← 2^A ⇔      e^(A)
add.f32        c, 1, b; // C ← 1+B ⇔      1 + e^(A)
div.approx.f32 s, 1, c; // S ← 1/C ⇔ 1 / ( 1 + e^(A) )
```

Registres	2 fp32 + 1 conditionnel
SFU call	0

Registres	4 fp32
SFU call	2

Applications & Conclusion



Merci pour votre attention.
Avez vous des questions ?

Références

Réseaux de neurones

- ✖ [Videos 3Blue1Brown - neural networks](#)

Optimisation produit de matrices

- ✖ [Cutlass Linear Algebra cuda](#)

Documentations

- ✖ [A List of Chip/IP for Deep Learning - NVIDIA](#)
- ✖ [Nvidia Tesla P100 whitepaper](#)
- ✖ <https://arxiv.org/pdf/1509.02308.pdf>
- ✖ <http://docs.nvidia.com/drive/index.html>

Drive PX

- ✖ Wikipédia
- ✖ <https://www.nvidia.fr/self-driving-cars/>

Architecture

- ✖ [Image de gravure](#)
- ✖ [Détail de l'architecture](#)
- ✖ [SFU - fonctions intrinsèques](#)

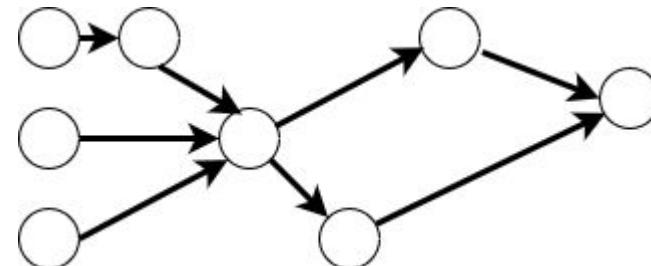
Optimisation produit de matrices

- ✖ [Cutlass Linear Algebra cuda](#)

Pour aller plus loin

NEAT : Neuroevolution of augmenting topologies

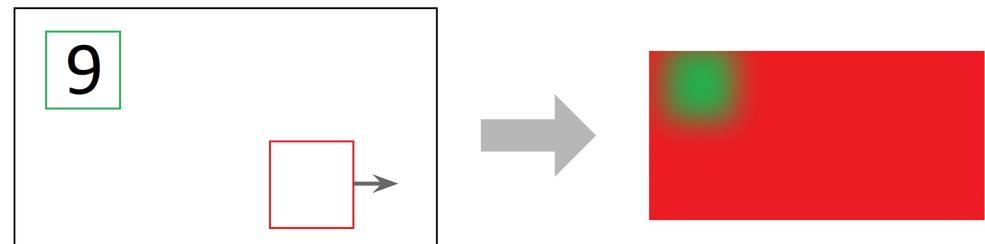
- ✗ algorithme génétique -> choix de la topologie
- ✗ Réseau de neurone -> calcul



Convolutif

Réseau de neurone convolutif

- ✗ Traitement d'image
- ✗ Détection de pattern



Merci !