MERCIER Julien

Séminaire GreenAI 2

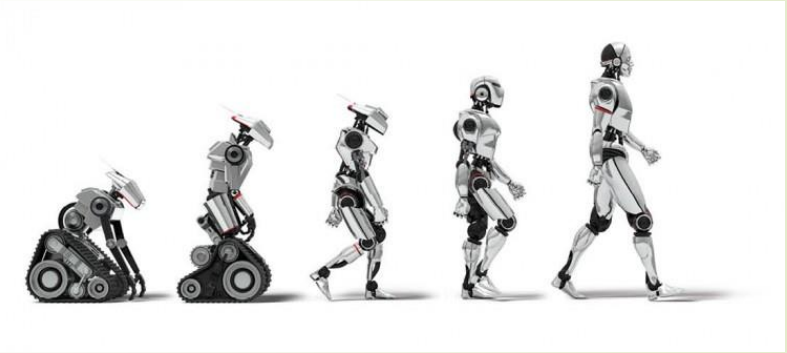
Sujet: Comment intégrer de la binarisation de réseau dans les algorithmes de deep learning?

UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

L'intelligence artificielle

- Voitures autonomes

- Robots intelligents



- Enceintes connectées

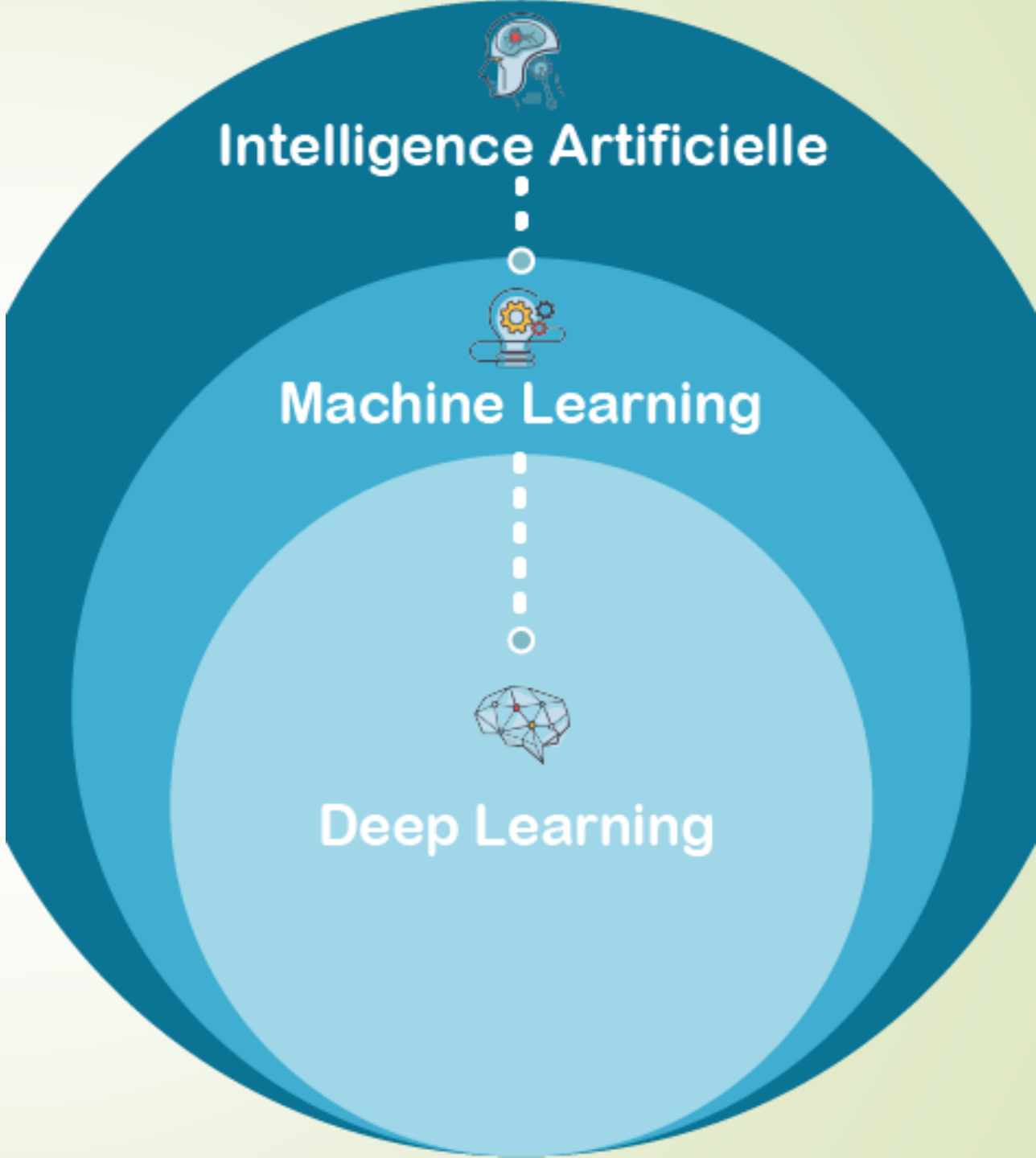
Qu'est ce que l'intelligence artificielle?

- Machines ou systèmes visant à imiter l'intelligence humaine
- Objectif:
  - Assister l'Homme dans des tâches de la vie courante

Deep Learning, c'est quoi?

- Apprentissage par couches successives
- Minimiser l'effort d'ingénierie
- Apprentissage des paramètres à partir des données

- Cela demande:
  - Un grand nombre de paramètres
  - De gros moyen de calcul



Problèmes / Motivations

- Algorithme difficile à embarquer
- Empreinte carbone grandissante

Solutions

- Algorithme binaire
- Léger
- Rapide

Plan

- Méthodes
- Expériences
- Gains

Définition des variables et algorithme général

- **K :** correspondra au nombre de couches
- a_k : sortie de la couche k
- W_k : paramètre à apprendre
- a_0 : données d'entrée
- **L(a_K, a^*)** : perte entre la prédiction du réseau a_k et la vérité terrain a^*.
- **lr** : taux d'apprentissage utilisé pour la mise à jour des poids
- $couche_k$: couche k paramétrée par des poids W_k à apprendre(linéaire, convolution, …)
- $activation_k$: non linéarité de la couche k (softmax, sigmoid, …)

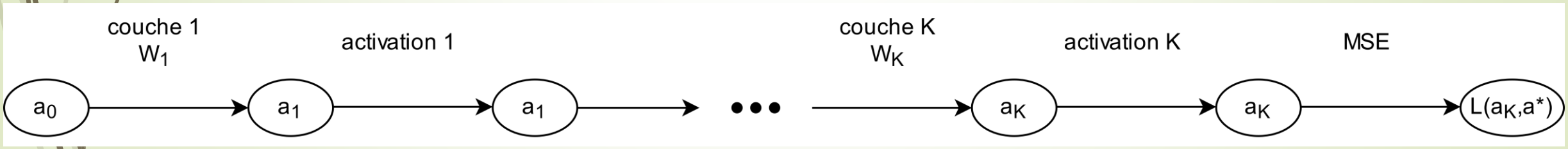


Schéma réseau Deep Learning classique à K couches

Algorithme de Deep Learning général

- Prédiction (passe forward)
- Descente de gradient
  - Calcul du gradient par rétropropagation
  - Mise à jour des poids

1- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$x \leftarrow couche_k\ (W_k, a_{k-1})$$
$$a_k \leftarrow activation_k(x)$$

2- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ \text{à}\ 1:$

$$\frac{dL}{dW_k} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k}$$

$Si\ k > 1:$

$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k}\ \frac{dW_k}{da_{k-1}}$$

3- Mise à jour des poids

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k}$$

BinaryConnect (2016, Courbariaux)

Titre: Training Deep Neural Networks with binary weights during propagations

- Binarisation des poids (W^b)

Introduction à l'algorithme de BinaryConnect

- Ajout d'une étape d'initialisation des poids binaires
- Utilisations des poids binaires
- Stockage et binarisation des poids

Initialisation des poids binaires

1- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$x \leftarrow couche_k(W_k, a_{k-1})$$
$$a_k \leftarrow activation_k(x)$$

2- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ \text{à}\ 1:$

$$\frac{dL}{dW_k} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k}$$

$Si\ k > 1:$

$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k}\ \frac{dW_k}{da_{k-1}}$$

3- Mise à jour des poids

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k}$$

Binarisation des poids

Algorithme de BinaryConnect

- Définition de la fonction sign():

$$\text{Sign(x)} = \begin{cases} -1 \; si \; x < 0 \\ +1 \; si \; x \geq 0 \end{cases}$$

- Utilisations des poids binaires lors de la prédiction

1- Initialisation des poids binaires

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k{}^b = sign(W_k)$$

2- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$x \leftarrow couche_k\ (W_k{}^b, a_{k-1})$$
$$a_k \leftarrow activation_k(x)$$

3- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ \text{à}\ 1:$

$$\frac{dL}{dW_k} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k}$$

$Si\ k > 1:$

$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k}\ \frac{dW_k}{da_{k-1}}$$

3- Mise à jour des poids

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k}$$

Binarisation des poids

Algorithme de BinaryConnect

- Gradient calculé sur les poids binaires
- Stockage des poids réels pour la mise à jour par descente de gradient
- Mise à jour des poids binaires à partir des nouveaux poids réels

1- Initialisation des poids binaires

$Pour\ k\ allant\ de\ 1\ à\ K:$

$$W_k{}^b = sign(W_k)$$

2- Prédiction

$Pour\ k\ allant\ de\ 1\ à\ K:$

$$x \leftarrow couche_k\ (W_k{}^b, a_{k-1})$$
$$a_k \leftarrow activation_k(x)$$

3- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ à\ 1:$

$$\frac{dL}{dW_k{}^b} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k{}^b}$$

$Si\ k > 1:$

$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k{}^b}\ \frac{dW_k{}^b}{da_{k-1}}$$

4- Mise à jour des poids

$Pour\ k\ allant\ de\ 1\ à\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k{}^b}$$

$$W_k{}^b = sign(W_k)$$

Etape importante de BinaryConnect

- Pourquoi calculer le gradient avec des poids binaires et mettre à jour les poids reels?

1- Initialisation des poids binaires

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k{}^b = sign(W_k)$$

2- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$x \leftarrow couche_k\ (W_k{}^b, a_{k-1})$$
$$a_k \leftarrow activation_k(x)$$

3- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ \text{à}\ 1:$

$$\frac{dL}{dW_k{}^b} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k{}^b}$$

$Si\ k > 1:$

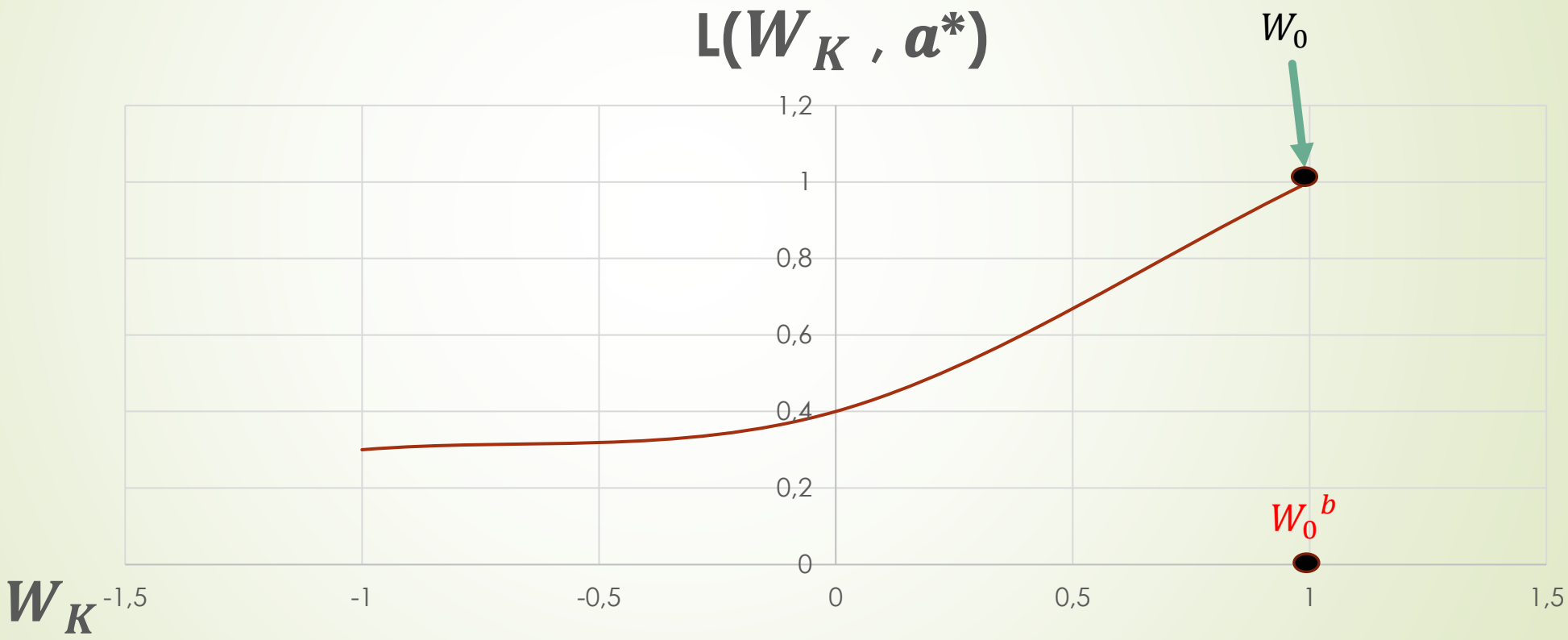
$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k{}^b}\ \frac{dW_k{}^b}{da_{k-1}}$$

4- Mise à jour des poids

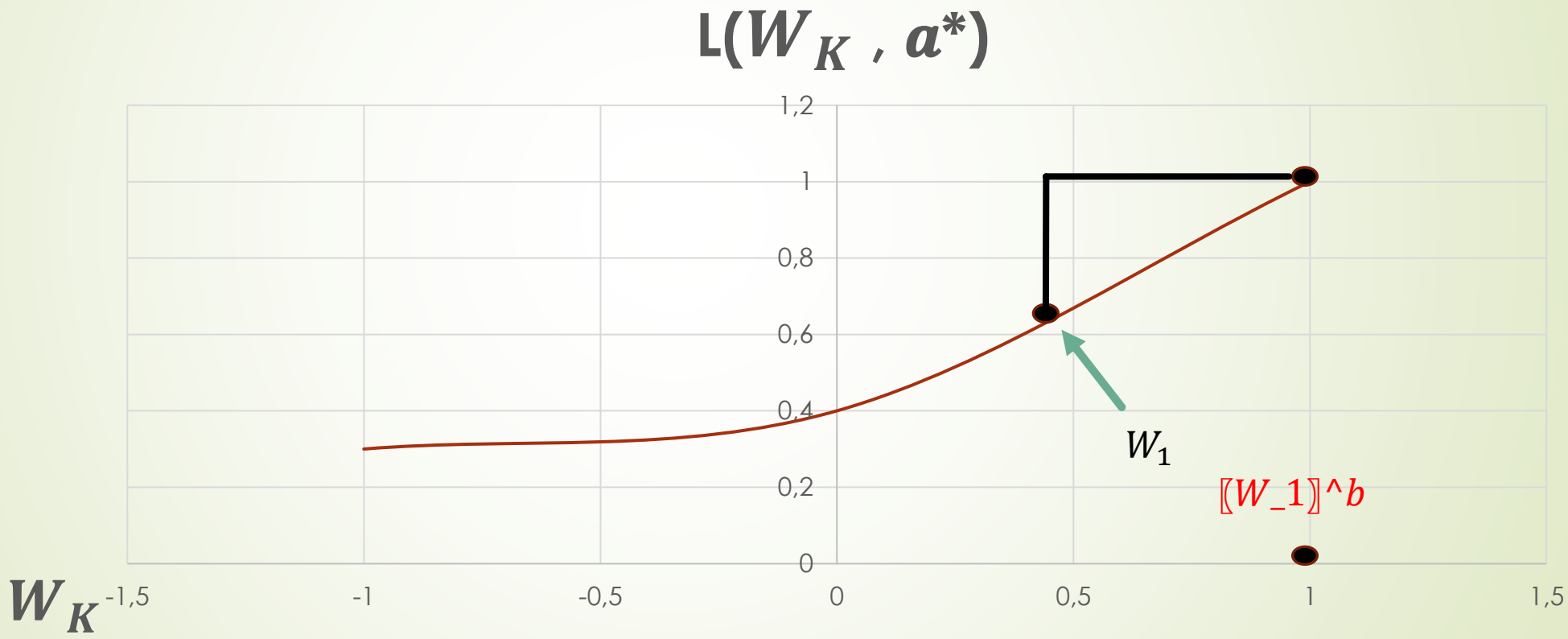
$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k{}^b}$$
$$W_k{}^b = sign(W_k)$$

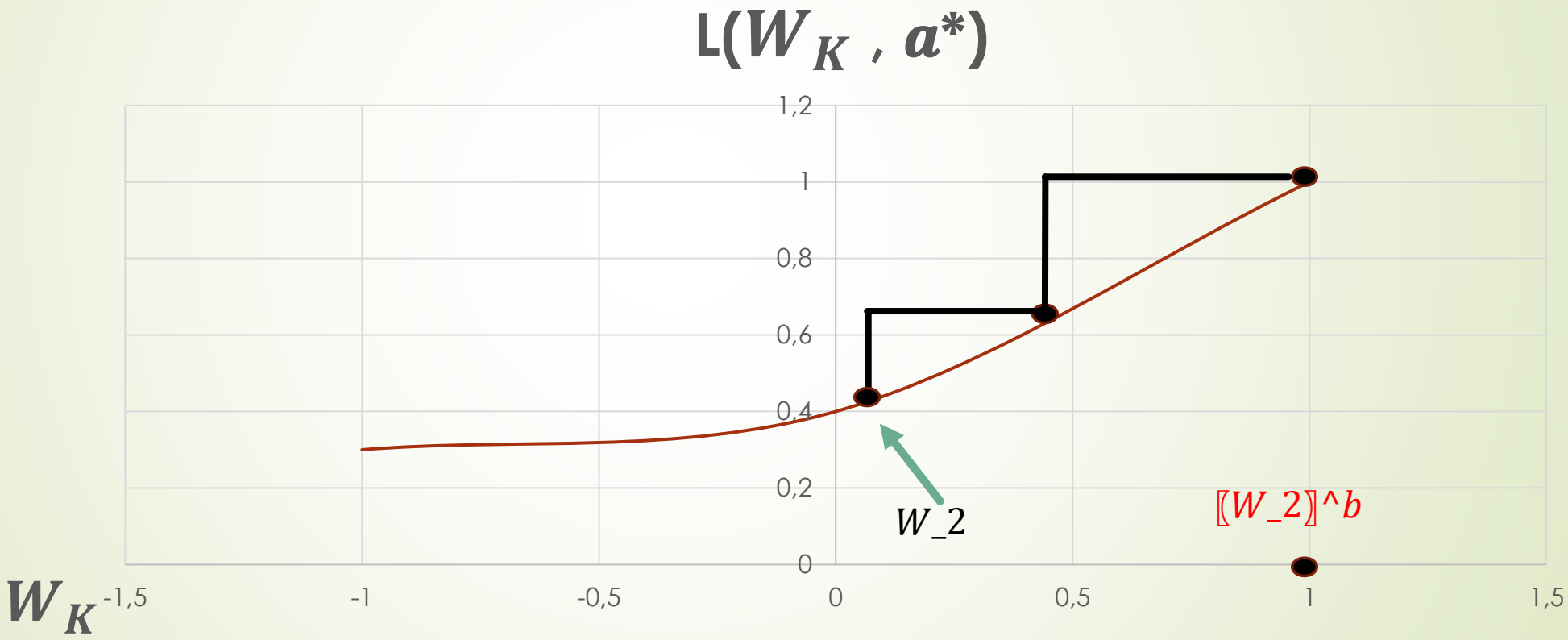
Descente de gradient pour les réseaux de neurones



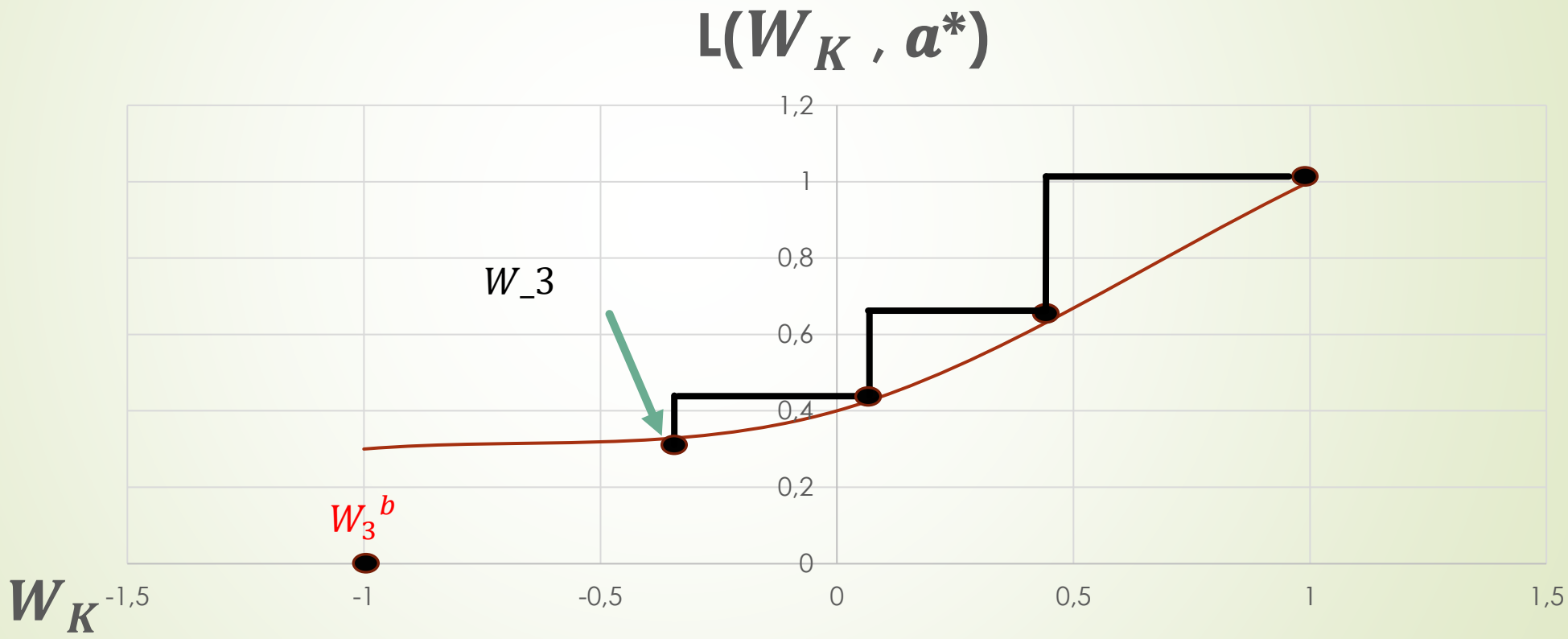
Descente de gradient pour les réseaux de neurones



Descente de gradient pour les réseaux de neurones



Descente de gradient pour les réseaux de neurones



BinaryNetwork (2016, Courbariaux)

Titre: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1

- Binarisation des poids (W_k^b)
- Binarisation des activations (a_k^b)

Introduction à l'algorithme de BinaryNetwork

- Ajout d'une binarisation des activations (a_k)
- Modification de la descente de gradient

1- Initialisation des poids binaires

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k{}^b = sign(W_k)$$

2- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$x \leftarrow couche_k\left(W_k{}^b, a_{k-1}\right)$$

$$a_k \leftarrow activation_k(x)$$

3- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k};$

$Pour\ k\ allant\ de\ K\ \text{à}\ 1:$

$$\frac{dL}{dW_k{}^b} \leftarrow \frac{dL}{da_k}\ \frac{da_k}{dW_k{}^b}$$

$Si\ k > 1:$

$$\frac{dL}{da_{k-1}} \leftarrow \frac{dL}{dW_k{}^b}\ \frac{dW_k{}^b}{da_{k-1}}$$

4- Mise à jour des poids

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$W_k \leftarrow W_k - lr * \frac{dL}{dW_k}$$

$$W_k{}^b = sign(W_k)$$

Définition de la prédiction de BinaryNetwork

- Ajout d'une binarisation des activations
  - Les données d'entrée a_0 et la dernière couche a_K ne sont pas binarisées

2- Prédiction

$Pour\ k\ allant\ de\ 1\ à\ K:$

$$Si\ k == 1 :$$
$$a_k \leftarrow couche_k\left(W_k{}^b, a_{k-1}\right)$$
$$Sinon :$$
$$a_k \leftarrow couche_k\left(W_k{}^b, a_{k-1}{}^b\right)$$
$$Si\ k == 1 :$$
$$a_k{}^b \leftarrow sign(a_k)$$
$$Sinon :$$
$$a_k \leftarrow activation_k(x)$$

Définition de la prédiction de BinaryNetwork

- Ajout d'une binarisation des activations
  - Les données d'entrée a_0 et la dernière couche a_K ne sont pas binarisées

2- Prédiction

$Pour\ k\ allant\ de\ 1\ \text{à}\ K:$

$$Si\ k == 1 :$$
$$a_k \leftarrow couche_k\left(W_k{}^b, a_{k-1}\right)$$
$$Sinon :$$
$$a_k \leftarrow couche_k\left(W_k{}^b, a_{k-1}{}^b\right)$$
$$Si\ k == 1 :$$
$$a_k{}^b \leftarrow sign(a_k)$$
$$Sinon :$$
$$a_k \leftarrow activation_k(x)$$

Problème: la fonction sign() n'est pas derivable

Problème avec la rétropropagation du gradient de BinaryNetwork

- Calcul du gradient:

$$\frac{dL}{dW_k{}^b} = \frac{dL}{da_k{}^b}\boxed{\frac{da_k{}^b}{da_k}}\frac{da_k}{dW_k{}^b}$$

- Définition d'un straight-through:

$$ST(x) = x * 1_{\{|x|<1\}}$$

- Rédefinition du gradient:

$$\frac{dL}{dW_k{}^b} \approx \frac{dL}{da_k{}^b}\ ST\left(\frac{da_k}{dW_k{}^b}\right)$$

Définition de la rétropagation du gradient de BinaryNetwork

- Modification de la descente de gradient

3- Rétropropagation du gradient

$Calculer\ \frac{dL}{da_k}$;

$Pour\ k\ allant\ de\ K$ à 1:

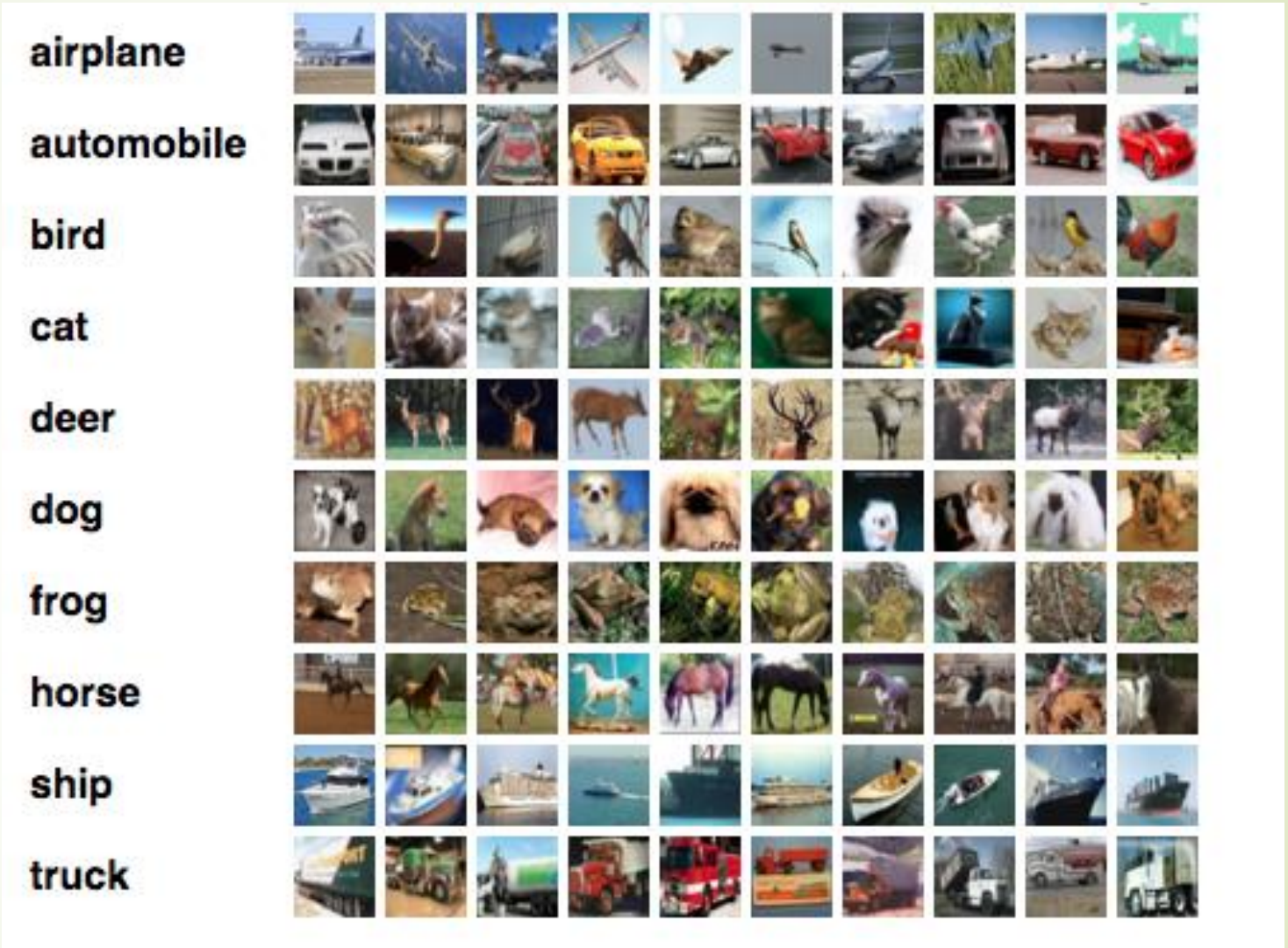
$$\frac{dL}{dW_k{}^b} \leftarrow \frac{dL}{da_k{}^b}\ ST\left(\frac{da_k}{dW_k{}^b}\right)$$

$Si\ k > 1$:

$$\frac{dL}{da_{k-1}{}^b} \leftarrow \frac{dL}{dW_k{}^b}\ \frac{dW_k{}^b}{da_{k-1}{}^b}$$

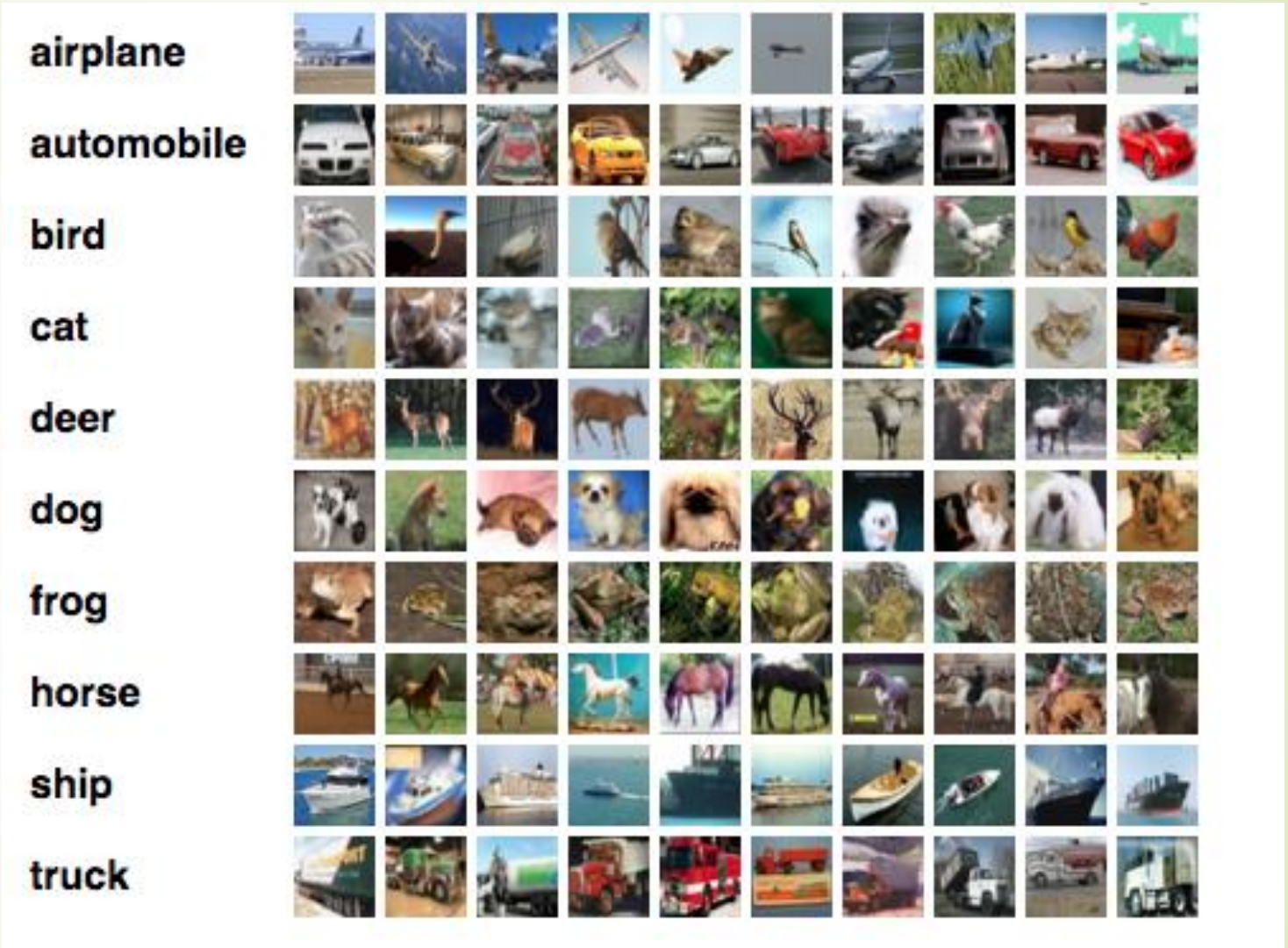
Expériences

- Jeu de données: Cifar 10
- Taille des images: 32x32x3
- Nombre de données: 50000



Protocole expérimental

- Taille du batch: 64
- Observation de la perte et de la justesse de prédiction
- Couche cachée à 100 neurones
- Clipping des poids et biais réels entre -1 et 1
- 5 modèles différents



Expériences

Réseau Deep learning général

- Poids réels
- Activations réelles
- Modèle linéaire et perceptron à une couche cachée

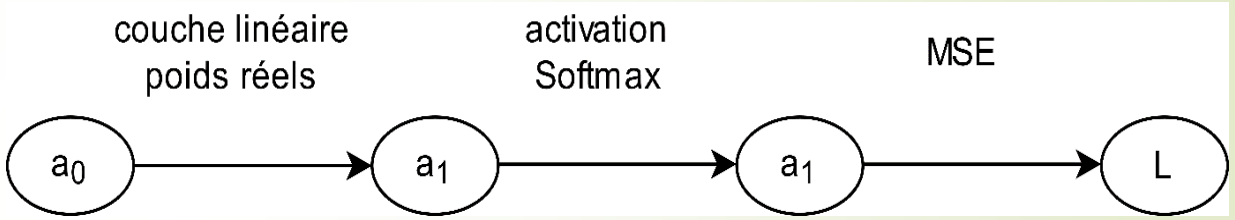


Schéma modèle linéaire poids réels

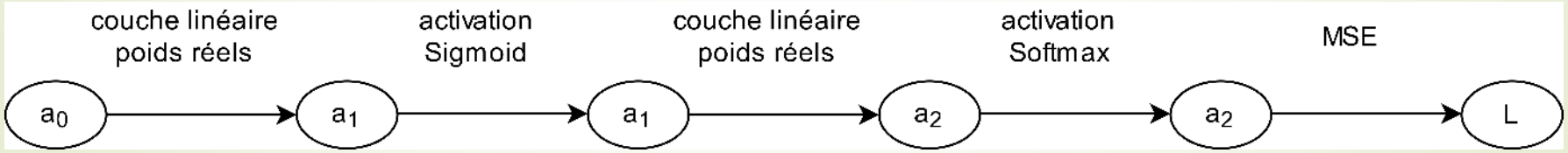


Schéma perceptron à une couche cachée

Expériences

BinaryConnect

- Poids binaires
- Modèle linéaire et perceptron à une couche cachée

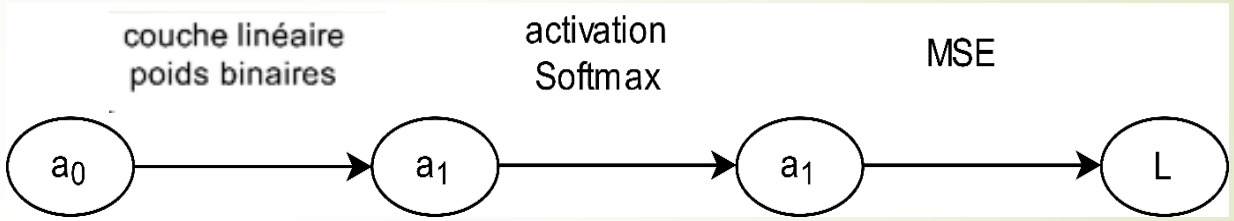


Schéma modèle linéaire poids binaires

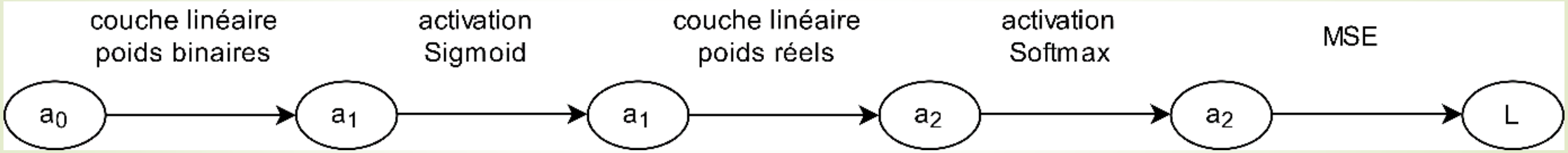


Schéma perceptron à une couche cachée BinaryConnect

Expériences

BinaryNetwork

- Poids binaires
- Activation binaire
- Architecture du perceptron à une couche cachée
- Modèle linéaire non disponible

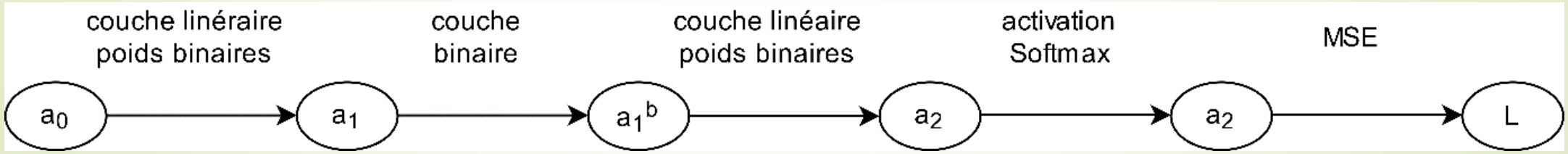


Schéma perceptron à une couche cachée BinaryNetwork

Tableau des résultats après 5 itérations

	Modèle linéaire		Perceptron à 1 couche cachée		
	Poids réels	**Poids binaires**	**Poids réels**	**BinaryConnect**	**BinaryNetwork**
Moyenne de la précision	39,36	29,66	51,58	31,7	35,2
Ecart-type	0,65	3,78	0,70	0,68	0,52
Empreinte mémoire (bytes)	0,12	0,004	1,233	0,04	0,04
Nb param	30730	30730	308310	308310	318410

Remarque: L'empreinte mémoire est 32 fois plus petite.

Gain énergétique

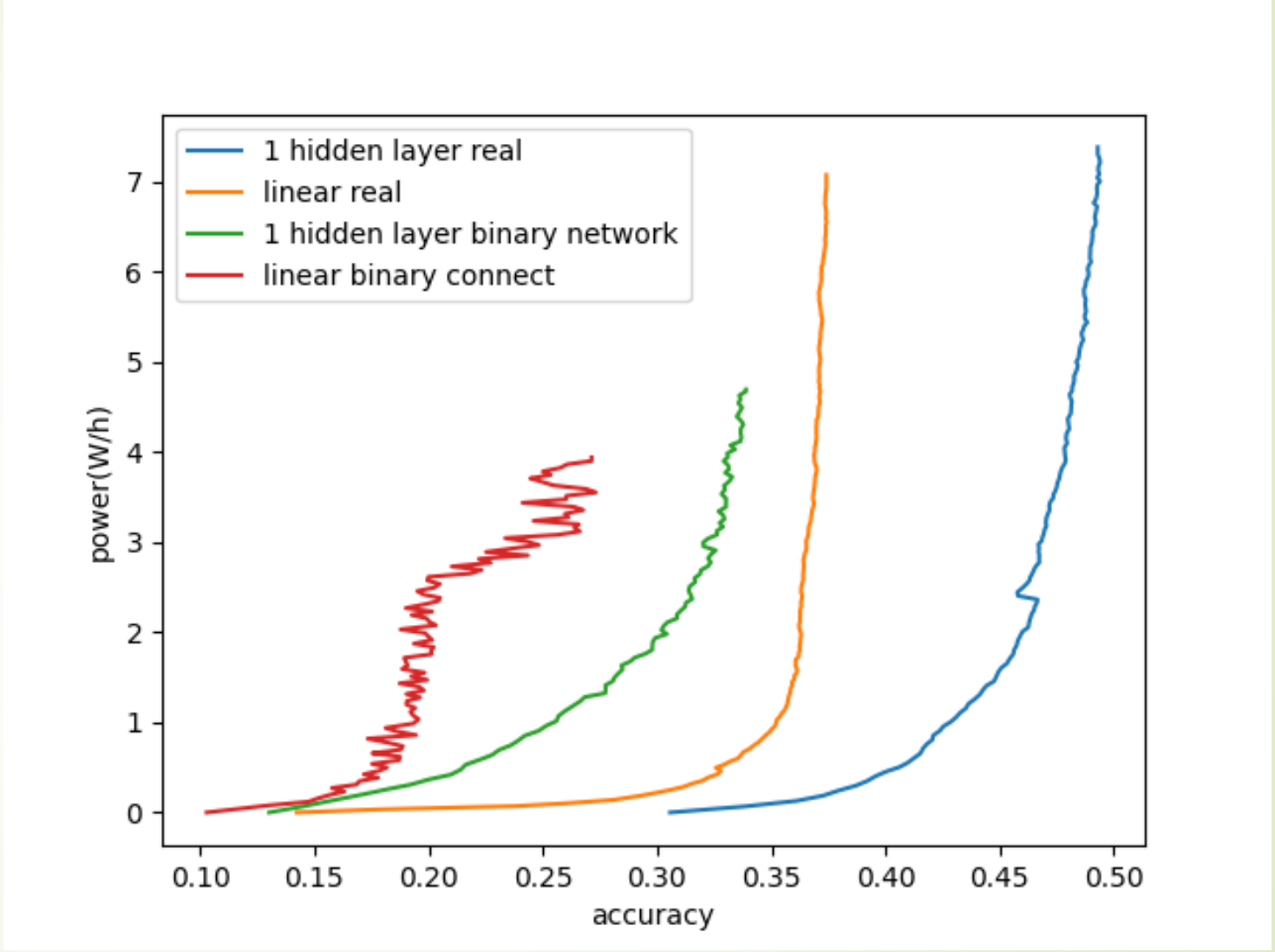
L'empreinte carbone et la consommation énergétique de nos reseaux binaires sont améliorées à plusieurs niveaux

Ce qu'on a avec notre implémentation de BinaryNetwork

- Empreinte mémoire 32 fois plus petite

Consommation énergétique

- Les derniers pourcentages sont durs à atteindre
- A budget énergétique equivalent, il est plus rentable d'entrainer des modèles plus gros
- La difficulté d'entrainer des reseaux binaires et leurs performances inférieures en termes de precision rend leur entrainement plus couteux en terme d'énergie

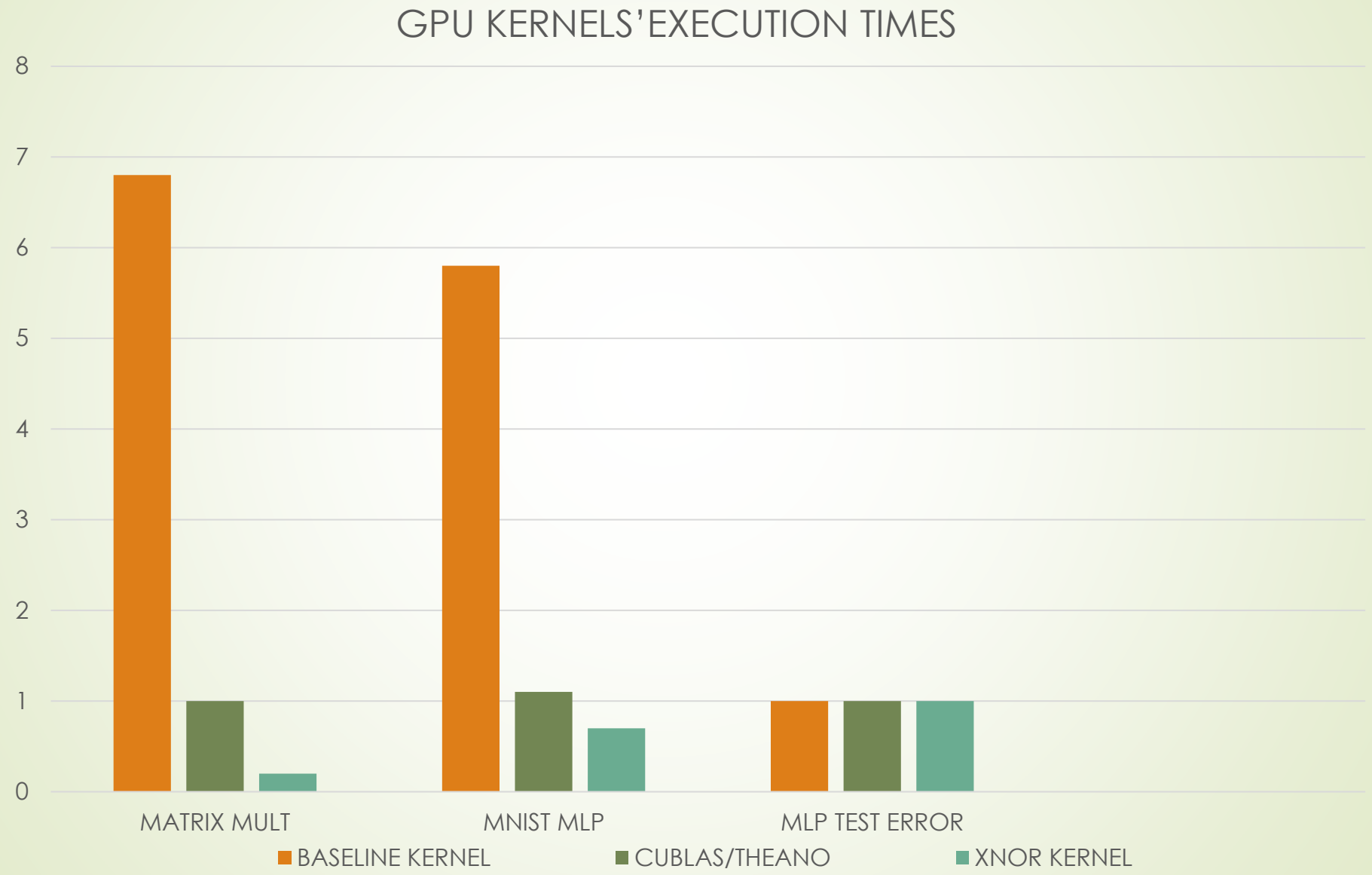


Gain énergétique

Potentiel du produit matriciel optimisé entre les poids binaires w_k^b et les activations binaires a_k^b

- Vitesse d'exécution
  - x58 sur CPU (ref xnornet, 2016)
  - x7 sur GPU (ref Courbariaux, 2016)
- Suppose une ré-implementation de bas niveau
  - Instructions binaires supportées par l'architecture du processeurs
  - Calcul matriciel optimisé (complexité algorithmique, gestion de la mémoire)

BinaryNetwork (Courbariaux, 2016)



Gain sur le produit scalaire

- Différence entre les produits scalaires classique et binaire

Produit scalaire classique: $x.y = \sum_i^N x_i y_i$

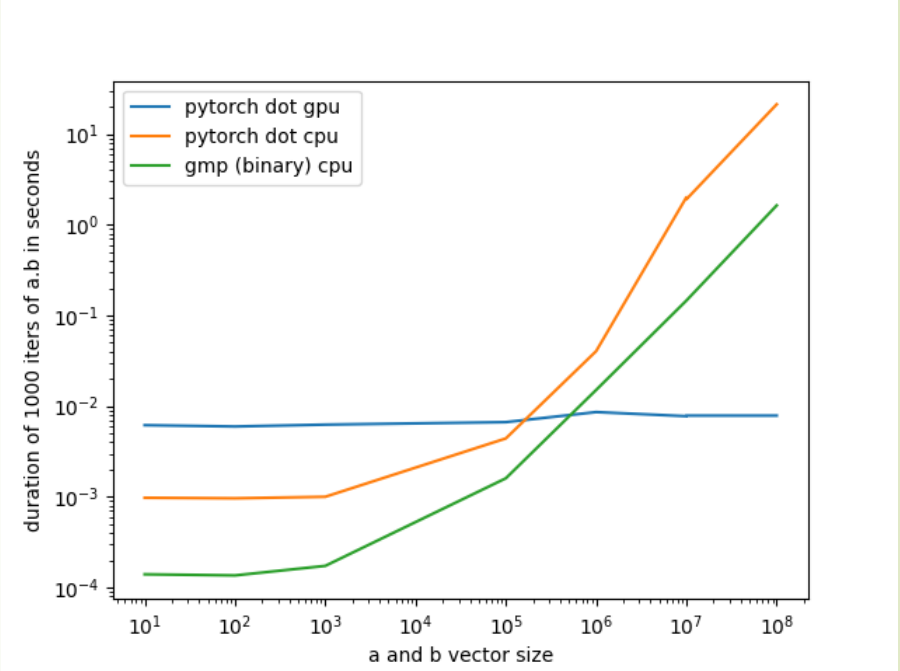
Produit scalaire binaire: $x.y = N - 2 * hamming(x,y)$

$$= N - 2 * popcount\big(xor(x,y)\big)$$

Pour $x_i \in \{-1,1\}\ et\ y_i \in \{-1,1\}$

Implémentation des opérateurs binaires

- Tester sur le produit scalaire mais peu d'information sur l'accés mémoire et la parallélisation
- Utilisation de la librairie optimisé GMPY2
- Comparaison avec le .dot() de Pytorch car les fonctions bitwise_xor() et torch.sum() sont moins efficace



Graphe du temps de calcul d'un produit scalaire en fonction de la taille des vecteurs en seconde

Implémentation des opérateurs binaires

- Implémentation de multiplications binaires de matrices disponible
  - Larq de Tensorflow
  - Bmxnet de Microsoft

Perspectives

- Packager les “class” implémentées pour leur reutilisation par le personnel du Lmap.
- Test des libraries larq et bmxnet

Conclusion