Développement d'une bibliothèque de machine learning

Soutenance de projet

Aurélien DELVAL
IATIC5

17 février 2021

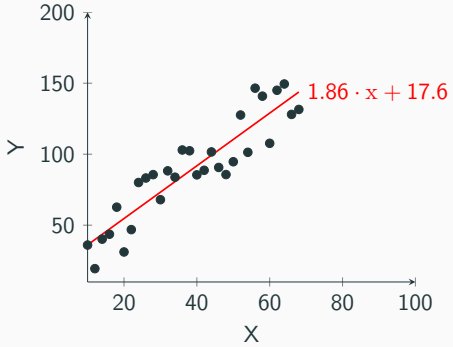
UVSQ université PARIS-SACLAY | ISTY Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Table des matières

Introduction théorique

Modèles mathématiques et régression

Principe de base : **régression**



On veut minimiser le **coût** : $c = \sum_{i=1}^{n} (a.x_i + b - y_i)^2$

Introduction aux réseaux de neurones : MNIST

- On va considérer l'exemple de la base de données MNIST

Introduction aux réseaux de neurones : MNIST

- On va considérer l'exemple de la base de données MNIST
- Reconnaissance de chiffres manuscrits (problème très simple)

Introduction aux réseaux de neurones : MNIST

- On va considérer l'exemple de la base de données MNIST
- Reconnaissance de chiffres manuscrits (problème très simple)

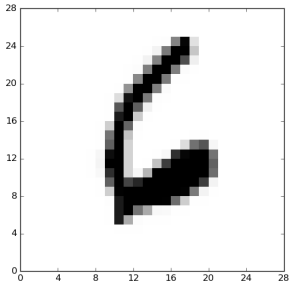
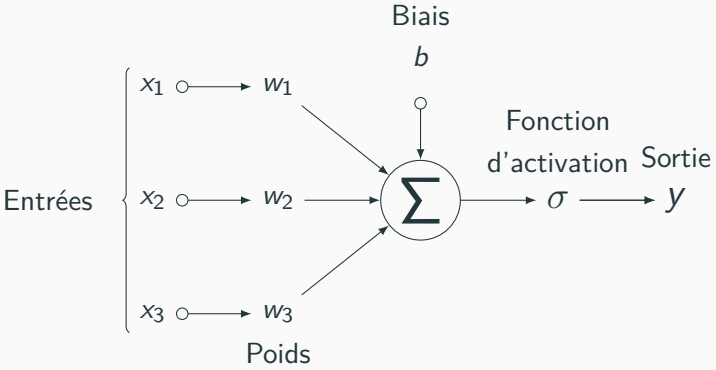


Image d'entrée (28×28 px $= 784$ entrées)

$$y^{pred} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

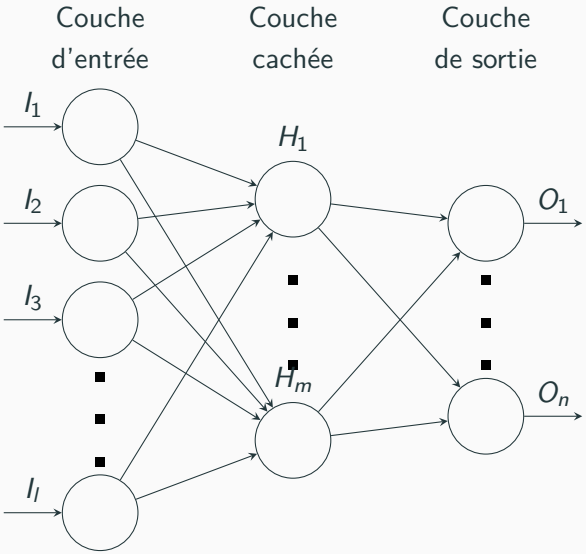
Vecteur de sortie attendu

Introduction aux réseaux de neurones : Neurone artificiel



$$f : \begin{cases} \mathbb{R} \to [0,1] \\ x \mapsto f_{act}\left(\sum_{i=1}^{n} x_i w_i \ + b\right) \end{cases}$$

Introduction aux réseaux de neurones : MLP



Valeur d'un neurone i :

$$H_i = f_{act}(\sum_{j=1}^{I} I_j w_{j,i} + b_i)$$

Valeur d'un neurone i :

$$H_i = f_{act}(\sum_{j=1}^{I} I_j w_{j,i} \ + b_i)$$

Plutôt que de considérer chaque neurone individuellement, on peut remarquer que :

$$H = f_{act}\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,I} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,I} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,I} \end{bmatrix} \cdot \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_I \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\right)$$

Comment exprimer le coût du modèle ?

$$\vec{c} = \begin{pmatrix} (\hat{y}_1 - y_1^{pred})^2 \\ (\hat{y}_2 - y_2^{pred})^2 \\ \vdots \\ (\hat{y}_n - y_n^{pred})^2 \end{pmatrix}$$

On exprime la norme de ce vecteur :

$$c = \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} (\hat{y}_i - y_i^{pred})^2$$

On obtient la **fonction de coût du réseau**

- On calculera le gradient ∇ de la fonction de coût.
- en prenant $-\nabla$, on s'approchera d'un minimum.
- Cependant, la fonction de coût est très complexe.

- On calculera le gradient ∇ de la fonction de coût.
- en prenant $-\nabla$, on s'approchera d'un minimum.
- Cependant, la fonction de coût est très complexe.

On peut calculer ∇ grâce à la **différentiation automatique**

Différentiation automatique

Principe et chain rule

La différentiation automatique ne permet pas d'exprimer, mais seulement de calculer un gradient.
Elle repose sur la **chain rule** :

$$(f \circ g)' = (f' \circ g).g'$$

Que l'on peut aussi écrire :

$$\frac{dz}{dx} = \frac{dz}{dy}.\frac{dy}{dx}$$

Exemple en mode forward

Prenons la fonction

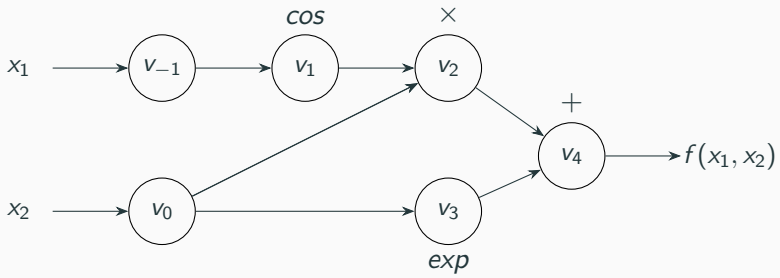
$$f : \begin{cases} \mathbb{R}^2 \to \mathbb{R} \\ (x_1, x_2) \mapsto x_2.cos(x_1) + e^{x_2} \end{cases}$$

Exemple en mode forward

Prenons la fonction

$$f : \begin{cases} \mathbb{R}^2 \to \mathbb{R} \\ (x_1, x_2) \mapsto x_2.cos(x_1) + e^{x_2} \end{cases}$$

Son **graphe de calcul** est :



Exemple en mode forward

$$\begin{cases} f(\pi, 1) = 1.cos(\pi) + e^1 = e - 1 \\ \frac{\partial f}{\partial x_1} = -1.sin(\pi) = 0 \\ \frac{\partial f}{\partial x_2} = cos(\pi) + e^1 = e - 1 \end{cases}$$

Exemple en mode forward

$$\begin{cases} f(\pi,1) = 1.cos(\pi) + e^1 = e-1 \\ \frac{\partial f}{\partial x_1} = -1.sin(\pi) = 0 \\ \frac{\partial f}{\partial x_2} = cos(\pi) + e^1 = e-1 \end{cases}$$

Evaluation de f	Evaluation de $\frac{\partial f}{\partial x_1}$	Evaluation de $\frac{\partial f}{\partial x_2}$
$v_{-1} = x_1 = \pi$ $v_0 = x_2 = 1$	$\dot{v_{-1}} = \dot{x_1} = 1$ $\dot{v_0} = \dot{x_2} = 0$	$\dot{v_{-1}} = \dot{x_1} = 0$ $\dot{v_0} = \dot{x_2} = 1$
$v_1 = cos(v_{-1}) = -1$ $v_2 = v_0.v_1 = -1$ $v_3 = e^{v_0} = e$	$\dot{v_1} = -\dot{v_{-1}}.sin(v_{-1}) = 0$ $\dot{v_2} = \dot{v_0}.v_1 + v_0.\dot{v_1} = 0$ $\dot{v_3} = \dot{v_0}.e^{v_0} = 0$	$\dot{v_1} = -\dot{v_{-1}}.sin(v_{-1}) = 0$ $\dot{v_2} = \dot{v_0}.v_1 + v_0.\dot{v_1} = -1$ $\dot{v_3} = \dot{v_0}.e^{v_0} = e$
$v_4 = v_2 + v_3 = e-1$	$\dot{v_4} = \dot{v_2} + \dot{v_3} = 0$	$\dot{v_4} = \dot{v_2} + \dot{v_3} = e-1$

Exemple en mode backward

Le principe reste le même, mais en réécrivant la chain rule :

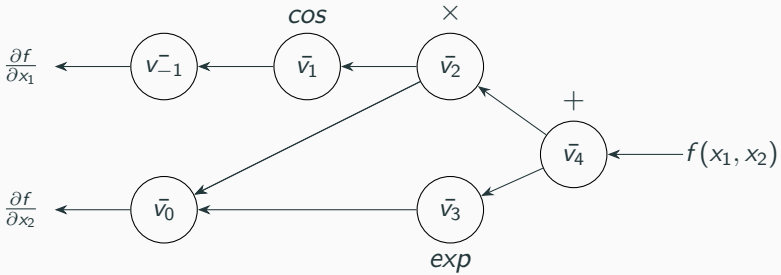
$$\frac{dz}{dx} = \frac{dz}{dy}.\frac{dy}{dx}$$

Devient :

$$\frac{dx}{dz} = \frac{dy}{dz}.\frac{dx}{dy}$$

Exemple en mode backward

On va parcourir le graphe de calcul depuis la fin :



Exemple en mode backward

Evaluation de f	Evaluation des $\frac{\partial f}{\partial x_i}$
$v_{-1} = x_1 = \pi$	$\bar{v_{-1}} = \bar{v_1}.\frac{\partial v_1}{\partial v_{-1}} = 0$
$v_0 = x_2 = 1$	$\bar{v_0} = \bar{v_3}.\frac{\partial v_3}{\partial v_0} + \bar{v_2}.\frac{\partial v_2}{\partial v_0} = e - 1$
$v_1 = cos(v_{-1}) = -1$	$\bar{v_1} = \bar{v_2}.\frac{\partial v_2}{\partial v_1} = -1$
$v_2 = v_0.v_1 = -1$	$\bar{v_2} = \bar{v_4}.\frac{\partial v_4}{\partial v_2} = -1$
$v_3 = e^{v_0} = e$	$\bar{v_3} = \bar{v_4}.\frac{\partial v_4}{\partial v_3} = e$
$v_4 = v_2 + v_3 = e - 1$	$\bar{v_4} = 1$

On n'a fait qu'un seul passage !

Dans le cas général, si on a m entrées et n sorties :

Dans le cas général, si on a m entrées et n sorties :

- $n > m$, il est plus facile de calculer ∇ avec le mode forward (m passages)

Dans le cas général, si on a m entrées et n sorties :

- $n > m$, il est plus facile de calculer ∇ avec le mode forward (m passages)
- $n < m$, il est plus facile de calculer ∇ avec le mode backward (n passages)

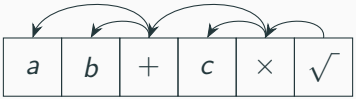
Dans le cas général, si on a m entrées et n sorties :

- $n > m$, il est plus facile de calculer ∇ avec le mode forward (m passages)
- $n < m$, il est plus facile de calculer ∇ avec le mode backward (n passages)

Dans notre contexte, seul le **mode backward** est intéressant.

Implémentation

On va construire le graphe de calcul au fur et à mesure en utilisant une **liste de Wengert** :



Chaque noeud contient :

- les dérivées partielles de l'opération associée
- les indices des noeuds dont il dépend

Implémentation

Pour calculer le gradient, l'algorithme est le suivant :

Algorithme 1 : Calcul de gradient depuis une liste de Wengert

Entrée : liste de Wengert W, variable v associée au noeud W_k
Résultat : gradient ∇

1 $\nabla \leftarrow \{0\}_0^{|W|}$
2 $\nabla_k \leftarrow 1$
`/* Parcours depuis la fin */`
3 **pour** $i \leftarrow k$ **à** 1 **faire**
4 $\quad C \leftarrow enfants(W_k)$
5 $\quad$ **pour** $j \leftarrow 0$ **à** $|C|$ **faire**
`/* Indice dans C != Indice dans W ou ∇ */`
6 $\quad\quad j' \leftarrow indice_{\nabla}(C_j)$
7 $\quad\quad \nabla_{j'} \leftarrow \nabla_{j'} + \frac{\partial C_j}{\partial W_k} \cdot \frac{\partial C_j}{\partial C_i}$
8 $\quad$ **fin**
9 **fin**

Implémentation

Dans la pratique, on construira la liste au fur et à mesure des calculs avec de la **surcharge d'opérateur**.
Cette approche a plusieurs avantages :

Implémentation

Dans la pratique, on construira la liste au fur et à mesure des calculs avec de la **surcharge d'opérateur**.
Cette approche a plusieurs avantages :

- Quasiment transparente

Implémentation

Dans la pratique, on construira la liste au fur et à mesure des calculs avec de la **surcharge d'opérateur**.
Cette approche a plusieurs avantages :

- Quasiment transparente
- Calcul du gradient en un appel de fonction

Implémentation

Dans la pratique, on construira la liste au fur et à mesure des calculs avec de la **surcharge d'opérateur**.
Cette approche a plusieurs avantages :

- Quasiment transparente
- Calcul du gradient en un appel de fonction
- Rapide à calculer

Généralisation aux tenseurs

- Pour un réseau de neurones “réaliste”, la liste de Wengert sera énorme.

Généralisation aux tenseurs

- Pour un réseau de neurones “réaliste”, la liste de Wengert sera énorme.
- On peut encapsuler non plus des scalaires mais des tenseurs.

Généralisation aux tenseurs

- Pour un réseau de neurones “réaliste”, la liste de Wengert sera énorme.
- On peut encapsuler non plus des scalaires mais des tenseurs.
- Tout au long du développement, on utilisera la bibliothèque Eigen.

Généralisation aux tenseurs

- Pour un réseau de neurones “réaliste”, la liste de Wengert sera énorme.
- On peut encapsuler non plus des scalaires mais des tenseurs.
- Tout au long du développement, on utilisera la bibliothèque Eigen.
- On encapsulera le type `Eigen::Array` dans `ts::Tensor`.

Implémentation des modèles

Principe de l'implémentation

On veut pouvoir créer des modèles :

Principe de l'implémentation

On veut pouvoir créer des modèles :

- de manière simple (un utilisateur pourrait le faire)

Principe de l'implémentation

On veut pouvoir créer des modèles :

- de manière simple (un utilisateur pourrait le faire)
- utilisables avec tous les algorithmes d'optimisation

Principe de l'implémentation

On veut pouvoir créer des modèles :

- de manière simple (un utilisateur pourrait le faire)
- utilisables avec tous les algorithmes d'optimisation

On va créer une classe virtuelle `ts::Model` avec une méthode `compute` qui va encapsuler tous les calculs.

Principe de l'implémentation

On veut pouvoir créer des modèles :

- de manière simple (un utilisateur pourrait le faire)
- utilisables avec tous les algorithmes d'optimisation

On va créer une classe virtuelle `ts::Model` avec une méthode `compute` qui va encapsuler tous les calculs.

Il suffira de définir un constructeur et cette méthode de calcul pour créer un modèle.

Exemple du MLP

On veut stocker :

Exemple du MLP

On veut stocker :

- les matrices de poids

Exemple du MLP

On veut stocker :

- les matrices de poids
- les vecteurs de biais

Exemple du MLP

On veut stocker :

- les matrices de poids
- les vecteurs de biais

L'algorithme de calcul est :

Algorithme 2 : Calcul de la sortie d'un MLP

Entrée : vecteur d'entrée v, matrices de poids $\{w\}_{i=0}^{n}$, vecteurs de biais $\{b\}_{i=0}^{n}$, fonction f_{act}

Résultat : vecteur H_n

1 $H_0 \leftarrow v$
2 **pour** $i \leftarrow 1$ **à** n **faire**
3 $\quad H_i \leftarrow f_{act}(w_i.H_{i-1} + b_i)$
4 **fin**

Conclusion sur les modèles

On est déjà capables

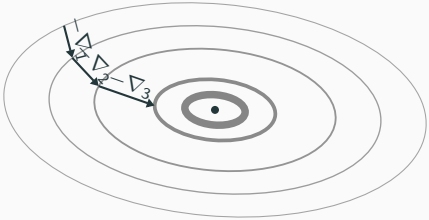
- de calculer la sortie de réseaux de neurones simples
- d'en implémenter de plus complexes
- d'obtenir très facilement leurs gradients (un appel de fonction)

Il reste à utiliser les calculs de gradient pour ajuster les paramètres.

Optimisation

Principe de la descente de gradient

Le vecteur $-\nabla$ nous indique la direction d'un minimum de la fonction de coût :



En ajustant petit à petit les paramètres du modèle, nos prédictions deviendront meilleures.

Algorithme SGD

L'algorithme SGD (*stochastic gradient descent*) de base est la version la plus simple :

Algorithme SGD

L'algorithme SGD (*stochastic gradient descent*) de base est la version la plus simple :

- On définit un taux d'apprentissage scalaire η (très faible)

Algorithme SGD

L'algorithme SGD (*stochastic gradient descent*) de base est la version la plus simple :

- On définit un taux d'apprentissage scalaire η (très faible)
- On calcule ∇ sur des données d'entraînement

Algorithme SGD

L'algorithme SGD (*stochastic gradient descent*) de base est la version la plus simple :

- On définit un taux d'apprentissage scalaire η (très faible)
- On calcule ∇ sur des données d'entraînement
- On incrémente le modèle de $-\eta\nabla$

Algorithme SGD

L'algorithme SGD (*stochastic gradient descent*) de base est la version la plus simple :

- On définit un taux d'apprentissage scalaire η (très faible)
- On calcule ∇ sur des données d'entraînement
- On incrémente le modèle de $-\eta\nabla$

Après des centaines ou milliers d'itérations, la fonction de coût converge.

Algorithme SGD

Dans la pratique, on utilise souvent des **batches** et plusieurs **epochs**.

Algorithme 3 : Descente stochastique de gradient

Entrée : fonction f, paramètres θ_0, ensemble de batches B, nombre d'epochs e, taux d'apprentissage η

Résultat : paramètres θ_t

pour $i \leftarrow 1$ **à** e **faire**
  pour $t \leftarrow 1$ **à** $|B|$ **faire**
    $b \leftarrow B_t$
    $\nabla_{sum} \leftarrow \{0\}_{k=1}^{|\theta_0|}$
    pour $j \leftarrow 1$ **à** $|b|$ **faire**
      $\nabla_{sum} \leftarrow \nabla_{sum} + \nabla f_t(\theta_{t-1}, b_j)$
    fin
    $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\nabla_{sum}}{|b|}$
  fin
fin

Mise en pratique

Résolution de MNIST

On a maintenant tous les éléments pour résoudre MNIST.

Voici comment se fait l'utilisation de la bibliothèque sur ce problème :

```
1    ts::MultiLayerPerceptron<float> model(
2        EXPECTED_IMAGE_SIZE,
3        {512, 128, 10}
4    );
5
6    ts::AdamOptimizer<float> optimizer;
7
8    std::vector<std::vector<std::vector< float >>> losses =
9    optimizer.run(model, trainingData);
```

Listing 1 – Définition et utilisation du MLP pour MNIST

Résolution de MNIST

Phase d'entraînement : **10** epochs et **1000** batches de taille **5**.

Résolution de MNIST

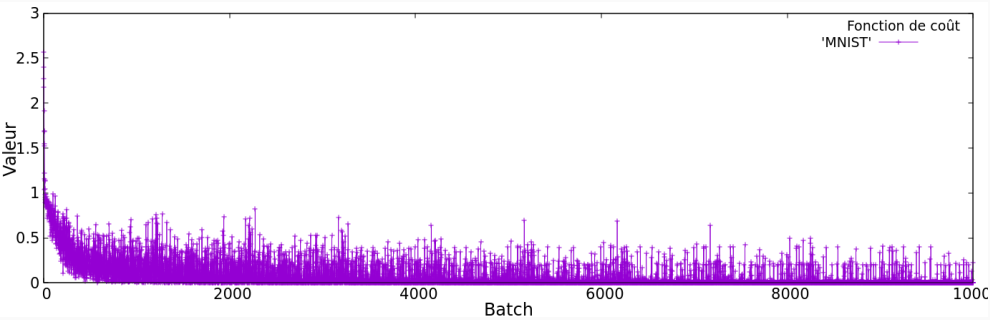
Phase d'entraînement : **10** epochs et **1000** batches de taille **5**.

Après quelques minutes d'entraînement, on obtient une précision de **96%**.

Résolution de MNIST

Phase d'entraînement : **10** epochs et **1000** batches de taille **5**.

Après quelques minutes d'entraînement, on obtient une précision de **96%**.
On observe aussi que la fonction de coût a convergé comme prévu :

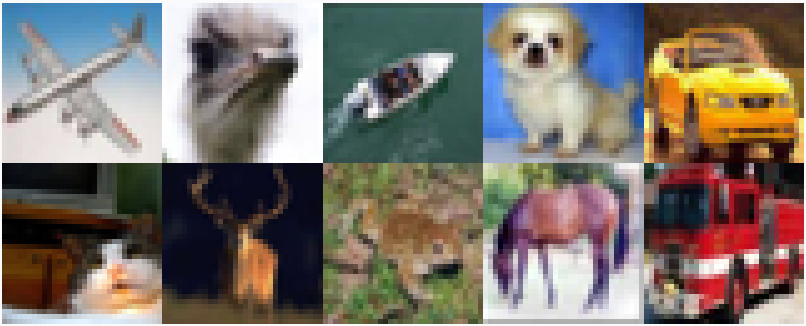


CIFAR et CNN

On va maintenant essayer de résoudre CIFAR, un problème similaire à MNIST, mais plus complexe :

CIFAR et CNN

On va maintenant essayer de résoudre CIFAR, un problème similaire à MNIST, mais plus complexe :



CIFAR et CNN

Phase d'entraînement : **15** epochs et **1000** batches de taille **5**.

CIFAR et CNN

Phase d'entraînement : **15** epochs et **1000** batches de taille **5**.

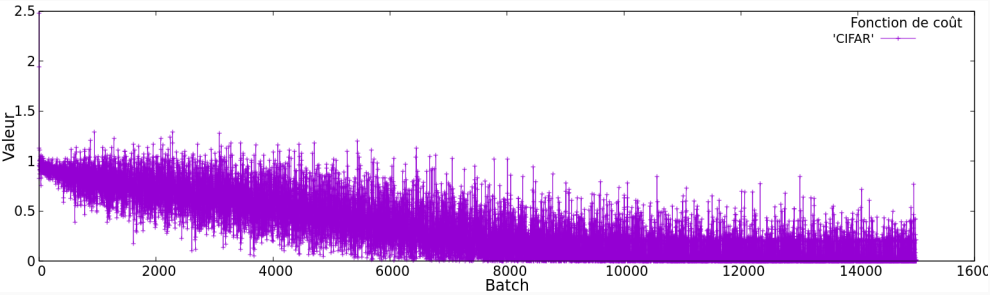
On obtient une précision de **49%**.

CIFAR et CNN

Phase d'entraînement : **15** epochs et **1000** batches de taille **5**.

On obtient une précision de **49%**.

La fonction de coût converge, mais plus lentement :



Détection d'objets avec un CNN

Dans MNIST, on se contentait de classifier des images : on va maintenant essayer de faire de la **détection d'objets**. On va :

Détection d'objets avec un CNN

Dans MNIST, on se contentait de classifier des images : on va maintenant essayer de faire de la **détection d'objets**. On va :

- entraîner un CNN à reconnaître des véhicules grâce à CIFAR

Détection d'objets avec un CNN

Dans MNIST, on se contentait de classifier des images : on va maintenant essayer de faire de la **détection d'objets**. On va :

- entraîner un CNN à reconnaître des véhicules grâce à CIFAR
- utiliser un autre jeu de données (TrafficNet) pour les prédictions

Détection d'objets avec un CNN

Dans MNIST, on se contentait de classifier des images : on va maintenant essayer de faire de la **détection d'objets**. On va :

- entraîner un CNN à reconnaître des véhicules grâce à CIFAR
- utiliser un autre jeu de données (TrafficNet) pour les prédictions
- isoler plusieurs zones des images de test pour détecter les véhicules

Détection d'objets avec un CNN

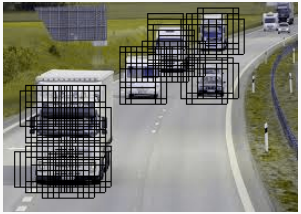
Dans MNIST, on se contentait de classifier des images : on va maintenant essayer de faire de la **détection d'objets**. On va :

- entraîner un CNN à reconnaître des véhicules grâce à CIFAR
- utiliser un autre jeu de données (TrafficNet) pour les prédictions
- isoler plusieurs zones des images de test pour détecter les véhicules

Remarque : La mise en place sera une version très simplifiée d'algorithmes tels que R-CNN ou YOLO.

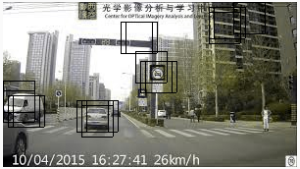
Détection d'objets avec un CNN

Voici quelques exemples de détection satisfaisants :



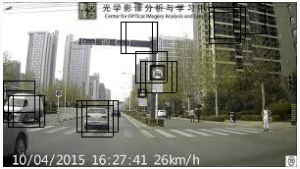
Détection d'objets avec un CNN

Et d'autres comportant des erreurs



Détection d'objets avec un CNN

Et d'autres comportant des erreurs



Il est normal que les résultats ne soient pas parfaits car :

- Les jeux de données utilisés n'étaient pas prévus à cet effet.
- Le CNN utilisé n'est pas tout à fait optimal (87% de précision sur la classification binaire).

Conclusion

Processus de développement

Utilisation de plusieurs bibliothèques :

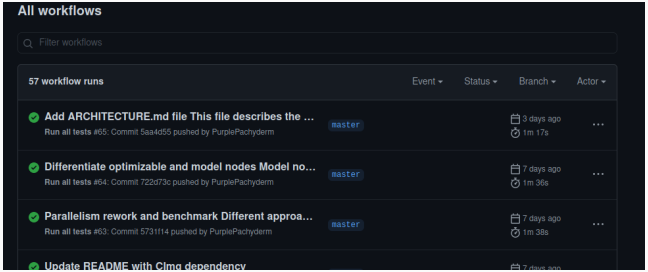
- Eigen
- Google Benchmarks
- Google Tests

Processus de développement

Utilisation de plusieurs bibliothèques :

- Eigen
- Google Benchmarks
- Google Tests

Intégration continue des tests unitaires avec les Github Workflows :



Conclusion sur la bibliothèque

- Son approche est très générale.

Conclusion sur la bibliothèque

- Son approche est très générale.
- Elle n'est évidemment pas complète...

Conclusion sur la bibliothèque

- Son approche est très générale.
- Elle n'est évidemment pas complète...
- ... mais fournit des outils pour de nombreuses utilisations potentielles.