Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Introduction au Deep Learning

Réseaux convolutifs

J. Rynkiewicz

Université Paris 1

2020

Classification supervisée : exemple de CIFAR10

Introduction au Deep Learning

J. Rynkiewicz

Exemple de CIFAR10

Le modèle linéaire généralisé

Le MLP à une couche cachée

Réseau de neurones profond

Réseaux convolutifs

On veut prédire une classe (ou catégorie) Y en fonction d'une variable X : $Y = f(X)$. Y est à valeur dans E, un ensemble fini de cardinal K et X est à valeur dans $\mathbb{R}^d$.
Pour illustrer ce problème on va considérérer l'ensemble d'images CIFAR10, où E est un ensemble de dix catégories :

1. Avions
2. Voitures
3. Oiseaux
4. Chats
5. Cerfs
6. Chiens
7. Grenouilles
8. Chevaux
9. Bateaux
10. Camions

X est une image 32×32 sur 3 canaux de couleurs (RGB), donc $d \simeq 3000$.

Estimation (apprentissage) d'un modèle (1)

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

- On veut estimer f_θ à l'aide d'observations $\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right)$, (échantillon d'apprentissage).
- On veut que cette estimation soit performante sur de nouvelles observations (qui ne sont pas dans l'ensemble d'apprentissage). C'est la capacité de "généralisation" du modèle, elle est estimée sur une ensemble de "test" : $\left(\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix}, \cdots, \begin{pmatrix} x_{n+T} \\ y_{n+T} \end{pmatrix} \right)$.

Dans l'exemple de CIFAR10, $n = 50000$ et $T = 10000$. Si on note $f_{\hat{\theta}}$ l'estimation de la fonction de classification une mesure de la performance pourra être :

- $\frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{\{y_i\}}(f_{\hat{\theta}}(x_i))$ (taux de bonne classification pour l'ensemble d'apprentissage)
- $\frac{1}{T} \sum_{i=1}^{T} \mathbf{1}_{\{y_{n+i}\}}((f_{\hat{\theta}} x_{n+i}))$ (taux de bonne classification pour l'ensemble de test)

Estimation (apprentissage) d'un modèle (2)

- Le taux de bonne classification n'est pas facile à manipuler directement pour l'apprentissage.
- On utilise plutôt la vraisemblance conditionnelle des observations pour estimer f_θ.
- On note $g_\theta(k, x_i)$, la probabilité conditionnelle de $Y = k$, si on observe x_i : $g_\theta(k, x_i) = P_\theta(Y_i = k \,|\, X_i = x_i)$.
- La log-vraisemblance conditionnelle s'écrit :

$$\ln\left(L_\theta\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix}\right)\right) = \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbf{1}_{\{k\}}(y_i) \ln\left(g_\theta(k, x_i)\right).$$

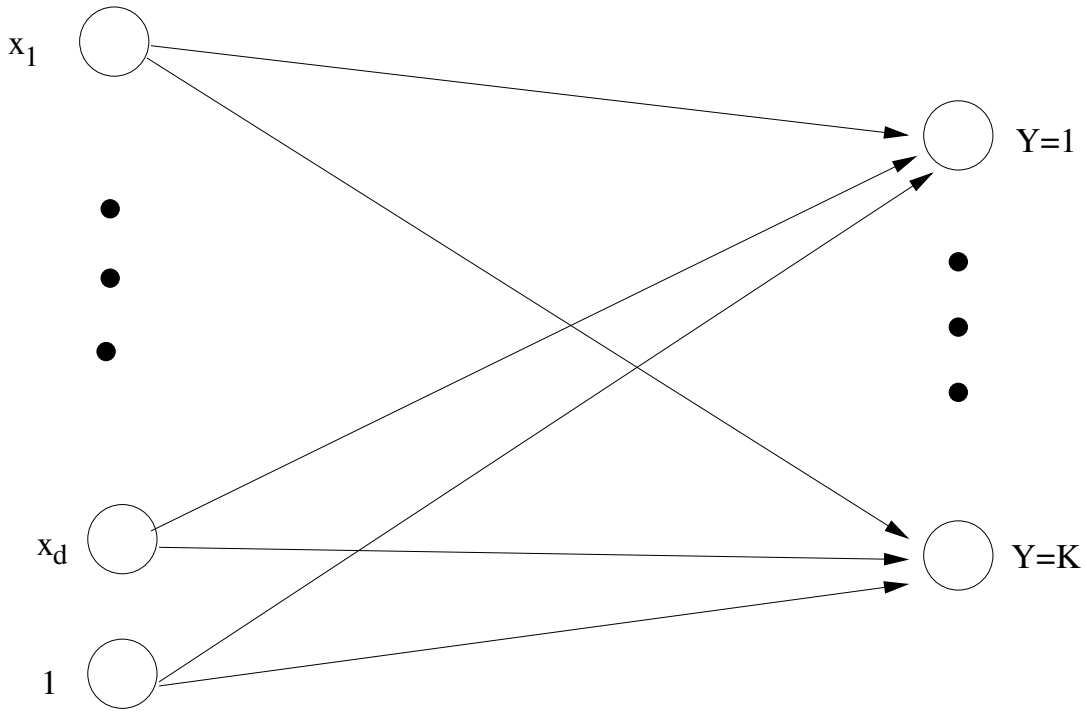
- On minimise l'opposé de la log-vraisemblance conditionnelle à l'aide d'une descente de gradient.

Le modèle linéaire généralisé

Le modèle le plus simple pour la fonction g est le modèle linéaire généralisé :

$$P(Y = k \,|X = x_i\,) = g_\theta(k, x_i) = \frac{\exp\left(\alpha_k + \beta_k^T x_i\right)}{\sum_{l=1}^{K} \exp\left(\alpha_l + \beta_l^T x_i\right)}$$

avec $\theta = (\alpha_1, \beta_1, \cdots, \alpha_K, \cdots, \beta_K)$. On peut schématiser ce modèle par :

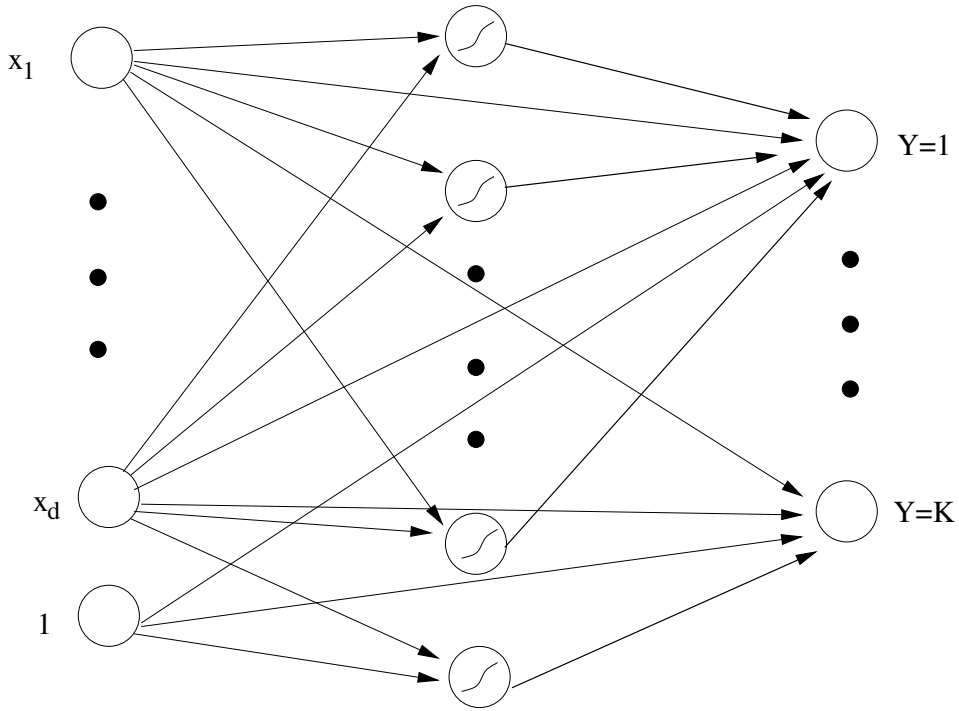


Le perceptron multicouche (MLP) à une couche cachée

On peut ajouter des non-linéarités dans la fonction g :

$$P(Y = k \,|X = x_i\,) = g_\theta(k, x_i) = \frac{\exp\left(F_{\theta_k}(x_i)\right)}{\sum_{l=1}^{K} \exp\left(F_{\theta_l}(x_i)\right)}$$

avec $F_{\theta_k}(x) = \alpha_k + \beta_k^T x + \sum_{h=1}^{H} a_h \sigma(b_k + W_k^T x)$ une fonction perceptron à une couche cachée et “skip layer”. On peut schématiser ce modèle par :



Application de ces deux modèles à CIFAR10

On estime ces modèles sur la base CIFAR10, grâce à la librairie “nnet” de R. Pour les deux modèles, l’estimateur du maximum de vraisemblance conditionnelle est calculé grâce à un algorithme de gradient “batch” (BFGS).

- Le modèle linéaire généralisé.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right).$$

  Le nombre de paramètres est de l’ordre de 30000, après plusieurs jours de calcul on obtient :

- Le MLP à une couche cachée avec “skip layer” et dix unités cachées. La log-vraisemblance conditionnelle est pénalisée par la norme au carré du vecteur paramètre pour limiter un peu l’éventuel sur-apprentissage.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right) + \mu \|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  Le nombre de paramètres est de l’ordre de 60000, après plus d’une semaine de calcul avec le BFGS on obtient :

Application de ces deux modèles à CIFAR10

On estime ces modèles sur la base CIFAR10, grâce à la librairie “nnet” de R. Pour les deux modèles, l’estimateur du maximum de vraisemblance conditionnelle est calculé grâce à un algorithme de gradient “batch” (BFGS).

- Le modèle linéaire généralisé.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right).$$

  Le nombre de paramètres est de l’ordre de 30000, après plusieurs jours de calcul on obtient :
  - Taux de bonne classification pour l’ensemble d’apprentissage : 54.19%.
  - Taux de bonne classification pour l’ensemble de test : 34.30%.
- Le MLP à une couche cachée avec “skip layer” et dix unités cachées. La log-vraisemblance conditionnelle est pénalisée par la norme au carré du vecteur paramètre pour limiter un peu l’éventuel sur-apprentissage.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right) + \mu \|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  Le nombre de paramètres est de l’ordre de 60000, après plus d’une semaine de calcul avec le BFGS on obtient :

Application de ces deux modèles à CIFAR10

On estime ces modèles sur la base CIFAR10, grâce à la librairie “nnet” de R. Pour les deux modèles, l’estimateur du maximum de vraisemblance conditionnelle est calculé grâce à un algorithme de gradient “batch” (BFGS).

- Le modèle linéaire généralisé.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right).$$

  Le nombre de paramètres est de l’ordre de 30000, après plusieurs jours de calcul on obtient :
  - Taux de bonne classification pour l’ensemble d’apprentissage : 54.19%.
  - Taux de bonne classification pour l’ensemble de test : 34.30%.
- Le MLP à une couche cachée avec “skip layer” et dix unités cachées. La log-vraisemblance conditionnelle est pénalisée par la norme au carré du vecteur paramètre pour limiter un peu l’éventuel sur-apprentissage.

$$\hat{\theta} = \arg\min_{\theta} - \ln \left(L_\theta \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right) \right) + \mu \|\theta\|^2, \text{ où } \mu = 10^{-9}.$$

  Le nombre de paramètres est de l’ordre de 60000, après plus d’une semaine de calcul avec le BFGS on obtient :
  - Taux de bonne classification pour l’ensemble d’apprentissage : 58.80%.
  - Taux de bonne classification pour l’ensemble de test : 32.47%.

Classification de CIFAR10 avec un réseau profond

- Le réseau a appris à l'aide du gradient stochastique.
- Comme les mise à jours de θ ne se font pas après le passage sur toute les données, il est facile de transformer les données de façon aléatoire pour enrichir l'ensemble d'entrainement. Ces transformations sont :
  - Renversement de la gauche et de la droite de l'image ("h-flip").
  - Petit recadrage aléatoire de l'image ("Random-crop").
- La log-vraisemblance conditionnelle est pénalisée par $10^{-5} \times ||\theta||^2$.
- On a coupé l'ensemble d'apprentissage en deux : 40000 exemples pour l'entrainement et 10000 exemples pour la validation.
  - Après chaque passage sur l'ensemble d'entrainement, on évalue l'erreur moyenne de classification sur la base de validation.
  - Au final ce sera le modèle avec la meilleur erreur de validation qui sera choisi (méthode du "hold-out").
- Après l'apprentissage (environ 1h30 avec une carte graphique) on obtient les résultats suivants :

Classification de CIFAR10 avec un réseau profond

- Le réseau a appris à l'aide du gradient stochastique.
- Comme les mise à jours de θ ne se font pas après le passage sur toute les données, il est facile de transformer les données de façon aléatoire pour enrichir l'ensemble d'entrainement. Ces transformations sont :
  - Renversement de la gauche et de la droite de l'image ("h-flip").
  - Petit recadrage aléatoire de l'image ("Random-crop").
- La log-vraisemblance conditionnelle est pénalisée par $10^{-5} \times ||\theta||^2$.
- On a coupé l'ensemble d'apprentissage en deux : 40000 exemples pour l'entrainement et 10000 exemples pour la validation.
  - Après chaque passage sur l'ensemble d'entrainement, on évalue l'erreur moyenne de classification sur la base de validation.
  - Au final ce sera le modèle avec la meilleur erreur de validation qui sera choisi (méthode du "hold-out").
- Après l'apprentissage (environ 1h30 avec une carte graphique) on obtient les résultats suivants :
  - Taux de bonne classification pour l'ensemble d'apprentissage : 96.93%

Classification de CIFAR10 avec un réseau profond

- Le réseau a appris à l'aide du gradient stochastique.
- Comme les mise à jours de θ ne se font pas après le passage sur toute les données, il est facile de transformer les données de façon aléatoire pour enrichir l'ensemble d'entrainement. Ces transformations sont :
  - Renversement de la gauche et de la droite de l'image ("h-flip").
  - Petit recadrage aléatoire de l'image ("Random-crop").
- La log-vraisemblance conditionnelle est pénalisée par $10^{-5} \times ||\theta||^2$.
- On a coupé l'ensemble d'apprentissage en deux : 40000 exemples pour l'entrainement et 10000 exemples pour la validation.
  - Après chaque passage sur l'ensemble d'entrainement, on évalue l'erreur moyenne de classification sur la base de validation.
  - Au final ce sera le modèle avec la meilleur erreur de validation qui sera choisi (méthode du "hold-out").
- Après l'apprentissage (environ 1h30 avec une carte graphique) on obtient les résultats suivants :
  - Taux de bonne classification pour l'ensemble d'apprentissage : 96.93%
  - Taux de bonne classification pour l'ensemble de validation : 95.44%

Classification de CIFAR10 avec un réseau profond

- Le réseau a appris à l'aide du gradient stochastique.
- Comme les mise à jours de θ ne se font pas après le passage sur toute les données, il est facile de transformer les données de façon aléatoire pour enrichir l'ensemble d'entrainement. Ces transformations sont :
  - Renversement de la gauche et de la droite de l'image ("h-flip").
  - Petit recadrage aléatoire de l'image ("Random-crop").
- La log-vraisemblance conditionnelle est pénalisée par $10^{-5} \times ||\theta||^2$.
- On a coupé l'ensemble d'apprentissage en deux : 40000 exemples pour l'entrainement et 10000 exemples pour la validation.
  - Après chaque passage sur l'ensemble d'entrainement, on évalue l'erreur moyenne de classification sur la base de validation.
  - Au final ce sera le modèle avec la meilleur erreur de validation qui sera choisi (méthode du "hold-out").
- Après l'apprentissage (environ 1h30 avec une carte graphique) on obtient les résultats suivants :
  - Taux de bonne classification pour l'ensemble d'apprentissage : 96.93%
  - Taux de bonne classification pour l'ensemble de validation : 95.44%
  - Taux de bonne classification pour l'ensemble de test : 92.26%

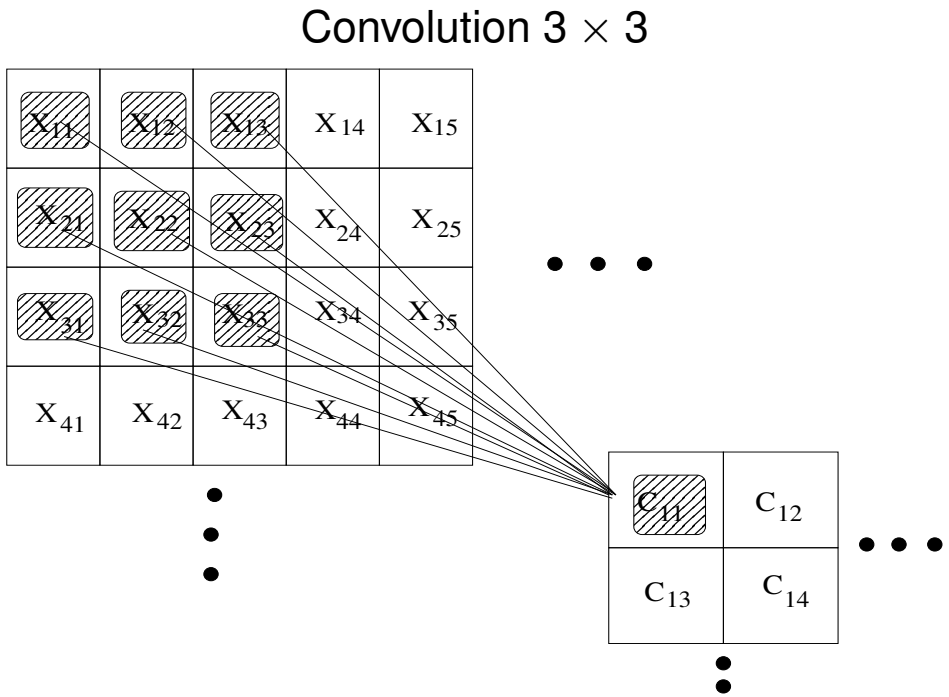
Réseaux convolutifs

- Ces résultats impressionnants, par rapport aux modèles classiques des années 1990, sont obtenus à l'aide d'un réseau convolutif.
- Le réseau utilisé dans cet exemple est appelé un VGG-16.
- Ce réseau enchaîne des couches convolutives et des fonctions de max-pooling.
- Il finit par une couche sans contrainte.
- Le réseau originel était utilisé sur des images bien plus détaillées, il a été adapté aux images CIFAR10.
- Il existe des réseaux encore plus performants (comme le Resnet) mais le VGG est particulièrement facile à étudier.

Couche convolutive

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Une couche convolutive revient à introduire des contraintes d'égalité entre de nombreux poids :



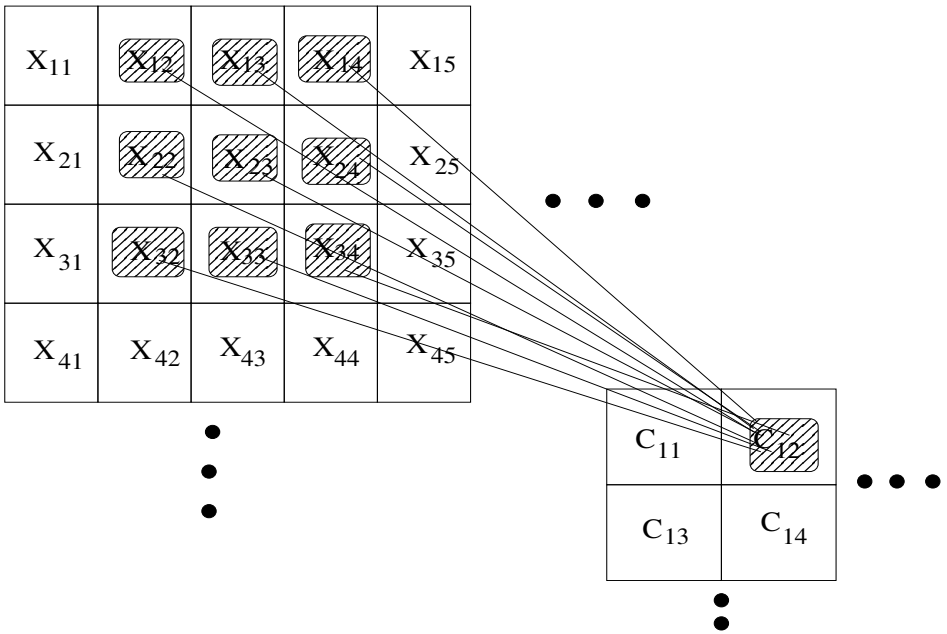
- $C_{11} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j}$

Couche convolutive

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Une couche convolutive revient à introduire des contraintes d'égalité entre de nombreux poids :

Convolution 3×3



- $C_{11} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j}$
- $C_{12} = \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ij} X_{i,j+1}$

Canaux de couches convolutives

- Les couches convolutives sont regroupées en canaux parallèles.
- Pour la n-ième couche, il y a $C(n)$ canaux parallèles.
- Les équations générales pour le neurone $C_{ijk}(n)$ de la couche n et du canal k sera donc :

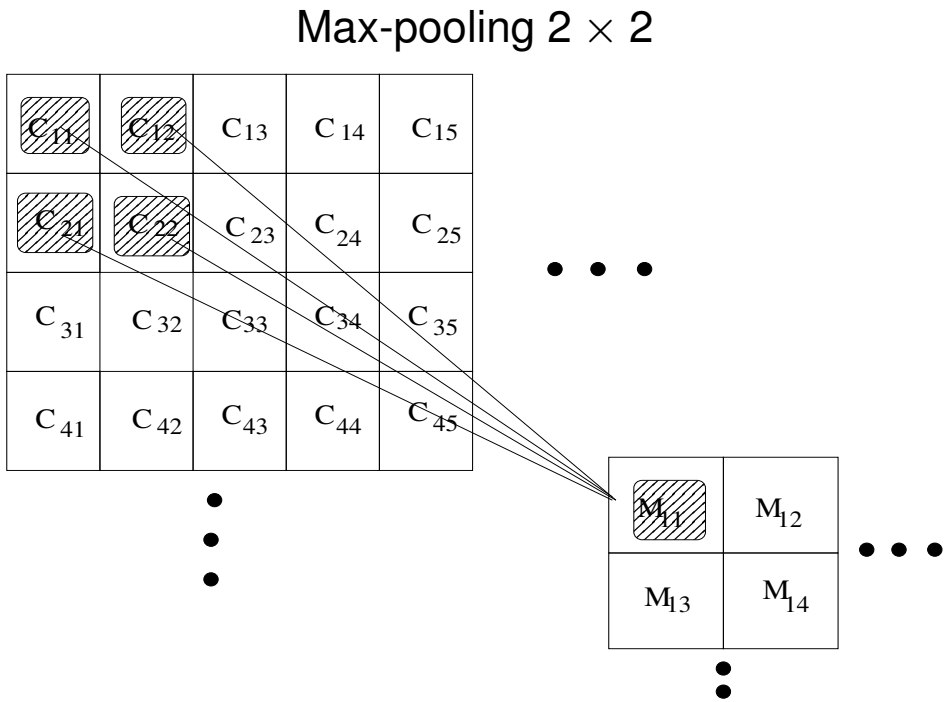
$$C_{ijk}(n) = \sum_{k=1}^{C(n-1)} \sum_{i=1}^{3} \sum_{j=1}^{3} W_{ijk} C_{i+i',j+j',k}(n-1)$$

- Pour les VGG, le nombre de canaux augmentent après chaque passage par la fonction de max-pooling car celle-ci réduit la surface de ces couches.
- Au final, il y a des millions de poids dans un VGG, mais ils sont extrêmement contraints dans les couches convolutives.

Couche de “max-pooling”

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Une couche de “max-pooling” calcule le maximum de plusieurs unités sur des petites zones :



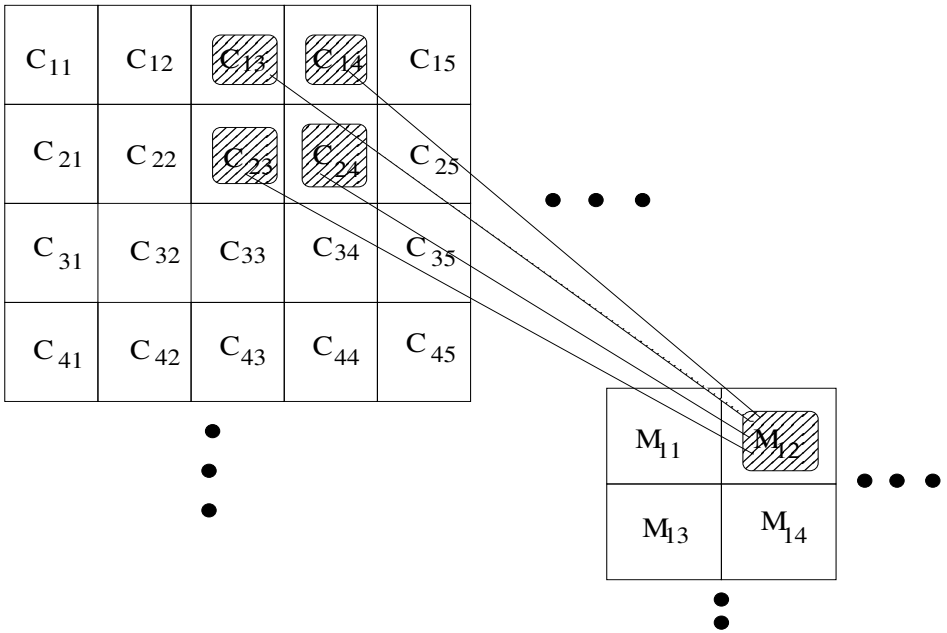
- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$

Couche de “max-pooling”

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Une couche de “max-pooling” calcule le maximum de plusieurs unités sur des petites zones :

Max-pooling 2×2



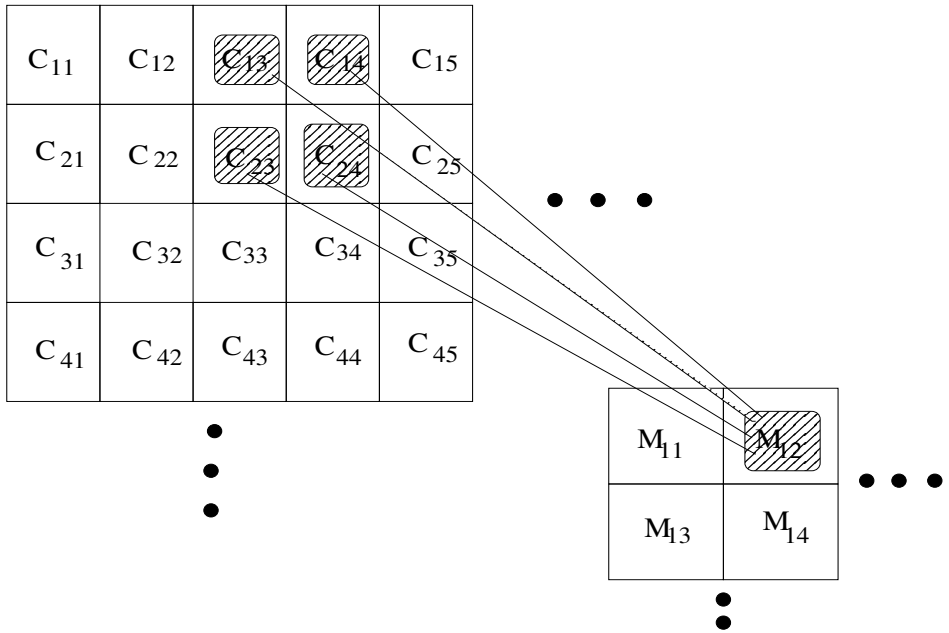
- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$
- $M_{12} = \max_{i,j=1}^{i,j=2} C_{i,j+2}$

Couche de “max-pooling”

Introduction au Deep Learning
J. Rynkiewicz
Exemple de CIFAR10
Le modèle linéaire généralisé
Le MLP à une couche cachée
Réseau de neurones profond
Réseaux convolutifs

Une couche de “max-pooling” calcule le maximum de plusieurs unités sur des petites zones :

Max-pooling 2×2



- $M_{11} = \max_{i,j=1}^{i,j=2} C_{ij}$
- $M_{12} = \max_{i,j=1}^{i,j=2} C_{i,j+2}$
- La sortie d’une telle couche aura une hauteur et une largeur divisée par deux !

VGG-16

Le réseau profond “VGG-16” enchaine des couches de convolution 3×3 (suivie d'une non-linéarité ReLU : $\sigma(x) = \max(0, x)$) et des couches de “max-pooling” 2×2. Pour ne pas réduire la dimension de la couche image par les convolutions on rajoute un cadre de 0 autour de la couche d'entrée (“padding”).
Si on note :

- CV3 les couches de convolution 3×3.
- ReLU, l'application de la non-linéarité ReLU sur chacune des unités.
- MP2 les couches de max-pooling 2×2.
- $NC \times W \times H$ la dimension des couches (nombre de canaux NC, largeur W, hauteur H)
- $LIN(NI, NO)$ une couche linéaire de dimension NI en entrée et NO en sortie.

l'architecture d'un VGG-16 sera :

$$(NC = 3, W = 32, H = 32) \overset{ReLU(CV3)}{\rightarrow} (NC = 64, W = 32, H = 32) \overset{ReLU(CV3)}{\rightarrow}$$
$$(NC = 64, W = 32, H = 32) \overset{MP2}{\rightarrow} (NC = 64, W = 16, H = 16) \overset{ReLU(CV3)}{\rightarrow}$$
$$(NC = 128, W = 16, H = 16) \overset{ReLU(CV3)}{\rightarrow} (NC = 128, W = 16, H = 16) \overset{MP2}{\rightarrow}$$
$$(NC = 128, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow} (NC = 256, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow}$$
$$(NC = 256, W = 8, H = 8) \overset{ReLU(CV3)}{\rightarrow} (NC = 256, W = 8, H = 8) \overset{MP2}{\rightarrow}$$
$$(NC = 256, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow}$$
$$(NC = 512, W = 4, H = 4) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 4, H = 4) \overset{MP2}{\rightarrow}$$
$$(NC = 512, W = 2, H = 2) \overset{ReLU(CV3)}{\rightarrow} (NC = 512, W = 2, H = 2) \overset{ReLU(CV3)}{\rightarrow}$$
$$(NC = 512, W = 2, H = 2) \overset{MP2}{\rightarrow} (NC = 512, W = 1, H = 1) \overset{LIN(512,10)}{\rightarrow} (NC = 1, W = 1, H = 10)$$

Estimation des paramètres du VGG-16

La mise à jours des paramètres se fait par mini batch de 128 observations. C'est une variante du l'algorithme du gradient stochastique :

- $\theta_{n+1} = \theta_n - \gamma \frac{1}{128} \sum_{i=1}^{128} \frac{\partial \ln(L_\theta(x_{t+i}, y_{t+i}))}{\partial \theta}$. Au cours de l'algorithme, γ décroît de 0.1, jusqu'à 0.01.
- Pour accélérer la descente du gradient, un momentum de 0.9 (cf le 2ème cours).
- La log-vraisemblance conditionnelle est pénalisée par $10^{-5} \times \|\theta\|^2$.
- On sélectionne le modèle avec la meilleur erreur de validation (méthode du "hold-out").
- On rappel que le taux de bonne classification pour l'ensemble de test de ce modèle est : 92.26%

Autres réseaux convolutifs

- Les principaux progrès dans les performances en classification de réseaux convolutifs sont liés à des modifications de l'architecture de ces modèles.
- Bien qu'ils diffèrent par leur architecture, tous ces réseaux combinent des couches convolutives et des couches de max-pooling.
- Un des plus célèbre est le Resnet, ce réseau introduit des connections "saute couche" qui permettent de mieux calculer le gradient.
- Avec le modèle Resnet, on peut augmenter le nombre de couches cachées (jusqu'à 150 !).
- Il existe aussi des modèles pour réduire le nombre de paramètres en conservant au possible les performances. Cela permet notamment de les utiliser dans de "petit" ordinateur (smartphone).