Date de remise: 17 Octobre (23h00), 2022

Instructions

- *Montrez vos traces pour toutes les questions !*
- *Utilisez LaTeX et le modèle que nous vous fournissons pour rédiger vos réponses. Vous pouvez réutiliser la plupart des raccourcis de notation, des équations et/ou des tableaux. SVP voir la politique des devoirs sur le site web du cours pour plus de détails.*
- *Soumettez ce devoir sur Gradescope. Assurez-vous que vous sélectionnez les pages contenant la réponse pour chaque question sur Gradescope. Nous ne noterons pas les questions sans les pages sélectionnées.*
- *Les auxiliaires d'enseignement pour ce devoir sont* ***Nanda Harishankar Krishna et Sarthak Mittal****.*

Question 1 (5-2-3-5)**.** (**La fonction d'activation**)

1. Considérez la fonction d'activation $h(x) = x\sigma(\beta x)$ où $\sigma(\cdot)$ définit la fonction d'activation sigmoïde $\sigma(z) = \frac{1}{1+\exp(-z)}$. Montrez que, pour un choix approprié de β, cette fonction d'activation peut rapprocher (a) la fonction d'activation linéaire, et (b) la fonction d'activation du ReLU.
2. Considérer la fonction d'activation linéaire continue $h(x)$ qui présente les pentes suivantes à différentes parties de l'entrée, en particulier, $h'(x) = \begin{cases} a & x >= \alpha \\ 0 & \beta < x < \alpha \\ -b & x <= \beta \end{cases}$ et $h(x) = 0$ pour $\beta < x < \alpha$. Supposons que vous n'ayez accès qu'aux opérations arithmétiques (l'addition, la soustraction, la multiplication et la division) ainsi que la fonction d'activation ReLU $ReLU(x) = max(0, x)$. Pouvez-vous construire $h(x)$ en utilisant ces opérations élémentaires et l'activation ReLU ? Pouvez-vous penser à une fonction d'activation populaire similaire à $h(\cdot)$?
   Conseil : Essayez de visualiser la fonction d'activation
3. Rappeler la définition de la fonction softmax $\mathcal{S}(\boldsymbol{x})_i = e^{x_i} / \sum_j e^{x_j}$, où $\boldsymbol{x} \in \mathbb{R}^d$. Montrez que c'est traduction invariante, c'est-à-dire $\mathcal{S}(\boldsymbol{x} + c) = \mathcal{S}(\boldsymbol{x})$ pour toute $c \in \mathbb{R}$.
4. Donné un i il est souvent nécessaire pour calculer $\log \mathcal{S}(\boldsymbol{x})_i = x_i - \log \sum_j e^{x_j}$. Cependant, cela peut souvent conduire à des incertitudes numériques car (a) si tous les x_j sont petits, alors il devient proche de $\log 0$ (soupassement), et b) si tous les x_j sont grands, leurs exposants peuvent entraîner un débordement. Pouvez-vous utiliser la propriété d'invariance de traduction de softmax pour le rendre numériquement stable ?
   Conseil : Pensez à $\max_j e^{x_j}$.

Answer 1. Q.1.1.

(a). Pour $\beta < 0$ et $|\beta|$ assez grand :
Pour $x > 0$, $\sigma(\beta x) \approx 1$ donc $h(x) \approx x$.

Pour $x < 0$, $\sigma(\beta x) \approx 0$ donc $h(x) \approx 0$
Ainsi, On a $h(x) \approx max(0, x) = ReLU(x)$

Pour $|\beta|$ suffisamment petit, on a $exp(-\beta x) \approx 1$, donc $h(x) \approx x/2$
Q.1.2. Posons $g(x) = max(0, a(x-\alpha)) + max(0, b(\beta - x)) = relu(a(x-\alpha)) + relu(b(\beta - x))$

1. g est une fonction d'activation linéaire et continue.
2. $(x - \alpha < 0) and (\beta - x < 0) \longrightarrow g(x) = 0$
3. $g'(x) = \begin{cases} a & x >= \alpha \\ 0 & \beta < x < \alpha \\ -b & x <= \beta \end{cases}$

Donc

$$g = h$$

Q.1.3. La fonction sigmoid S de x où $x = (x_1, x_2, \ldots, x_d)$ est la fonction $S(x) = (S(x)_1, S(x)_2, \ldots, S(x)_d)$, où

$$S(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ pour tout } j \in \{1, \ldots, d\}$$

De plus, la somme des $S(x)_i$ est égale à 1.

Soit i fixé dans $1, 2, \ldots, d$ et c dans $\mathbb{R}$:

$$S(x+c)_i = \frac{e^{x_j + c}}{\sum_{k=1}^{K} e^{x_k + c}} \tag{1}$$

$$= \frac{e^c e^{x_j}}{e^c \sum_{k=1}^{K} e^{x_k}} \tag{2}$$

$$= S(x)_i \tag{3}$$

Ainsi

$$S(x + c) = S(x)$$

Q.1.4.

En posant $C = -max_i x_i = -log(max_i e^{x_i})$, on calcule $S(x + C)$

On a une des entrées, celle qui est égale au max qui sera nulle avec l'exposant exponentielle, i.e. $\sum_j e^{x_j} > 1$. Donc pas de soupassement.

De plus, tous les exposant des $e^{x_j - max_j x_j}$ seront négatifs, on aura donc pas de débordement.

Question 2 (10). (**Les modèles de mélange rencontrent les réseaux neuronaux**)

Considérez de modéliser certaines données $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^d$, $y_n \in \{0, 1\}$, en utilisant un mélange de modèles de régression logistique, où nous modélisons chaque étiquette binaire y_n en sélectionnant d'abord l'un des modèles de régression logistique K, basée sur la valeur d'une variable latente $z_n \sim$ Categorical$(\pi_1, ..., \pi_K)$ puis générant y_n *conditionné* sur z_n en tant que $y_n \sim$ Bernoulli$[\sigma(\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n)]$, où $\sigma(\cdot)$ est la fonction d'activation sigmoïde.

Maintenant, veuillez considérer la probabilité *marginale* de l'étiquette $y_n = 1$, étant donné $\boldsymbol{x}_n$, c.-à-d., $p(y_n = 1|\boldsymbol{x}_n)$, et montrez que cette quantité peut aussi être considérée comme la sortie d'un réseau neuronal. Spécifiez clairement ce qu'est la couche d'entrée, le(s) calque(s) caché(s), les activations, la couche de sortie.

Answer 2. Dans l'espace de probabilité conditionné à x_n. On a

$$p(y_n = 1|x_n) = \sum_{\mathbf{z}} p(\mathbf{z}) p(y_n = 1 \mid x_n, \mathbf{z}) = \sum_{k=1}^{K} \mathbb{P}(z_n = k)\mathbb{P}(B_{n,k} = 1)$$

avec $B_{n,k}$ qui suit une loi de Bernoulli$[\sigma(\boldsymbol{w}_{z_n=k}^T \boldsymbol{x}_n)]$,

Ainsi,

$$p(y_n = 1|x_n) = \sum_{k=1}^{K} \pi_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x}_n)$$

The neural network .

- **Input Layer**
  - **Number of Neurons** : N*d.
  - **Input :** $x_{i,j} = x_i[j]$ **for** i, j **in** $[|1, N|] * [|1, d|]$
  On considère le vecteur $X = \begin{bmatrix} x_{1,1} & x_{1,2} & ... & x_{1,d} & x_{2,1} & ... & x_{n,d-1} & x_{n,d} \end{bmatrix}$,
- **First Hidden Layer Pre-activation** On pose le vecteur de poids W de taille (dN, KN), défini comme suit pour $i \leq NK - 1$ et $j \leq N - 1$:

  $$W[dj : d(j+1), i] = {w_{\text{int}(\frac{i+1}{K})}}^T$$

  Le reste des la matrice W est rempli par des zéros.
  Une visualisation de la matrice des poids donne :

  $$W = \begin{pmatrix} w_1^T & w_2^T & \cdots & w_K^T & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & w_1^T & \cdots & w_K^T & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & w_1^T & w_2^T & \cdots & w_K^T \end{pmatrix}$$

  Ainsi $A^{(1)} = XW = \begin{bmatrix} w_1^T x_1 & w_2^T x_1 & ... & w_K^T x_1 & w_1^T x_2 & ... & w_{K-1}^T x_N & w_K^T x_N \end{bmatrix}$ de taille $(1, NK)$

- **First Hidden Layer Activation**
  On applique la fonction d'activation Sigmoid, ainsi :
  $H^{(1)} = \begin{bmatrix}\sigma(w_1^T x_1) & \sigma(w_2^T x_1) & ... & \sigma(w_K^T x_1) & \sigma(w_1^T x_2) & ... & \sigma(w_{K-1}^T x_N) & \sigma(w_K^T x_N)\end{bmatrix}$
- **Output Layer**
  On pose $W^{(2)}$ la nouvelle matrice de poids de taille (NK, N). Elle est définie comme suit, pour tout j dans $[0, NK-1]$ et i dans $[0, N]$:

$$W^{(2)}[i,j] = \begin{cases} \pi_{(j+1)\%d} & \text{si } j \in \{id, id+1, ..., id+d-1\} \\ 0 & \text{sinon.} \end{cases}$$

  visualisation de $W^{(2)}$ donne :

$$W^{(2)} = \begin{pmatrix} \pi_1 & \cdots & 0 \\ \pi_2 & \cdots & 0 \\ \vdots & \vdots & \vdots \\ \pi_K & \cdots & 0 \\ 0 & \cdots & \pi_1 \\ 0 & \cdots & \pi_2 \\ \vdots & \vdots & \vdots \\ 0 & \cdots & \pi_K \end{pmatrix}$$

$A^{(2)} = H^{(1)}W^{(2)} = \begin{bmatrix}\sum_{k=1}^K \pi_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x}_1) & ... & \sum_{k=1}^K \pi_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x}_n)\end{bmatrix}$

En effet ,

$$A^{(2)} = \begin{bmatrix}p(y_1 = 1|x_1) & p(y_2 = 1|x_2) & ... & p(y_n = 1|x_n)\end{bmatrix}$$

Ce qui conclut le réseau neuronal.

Question 3 (5-2-5-2-2-4)**.** (**Schémas d'optimisation**)

L'optimisation est l'un des piliers les plus importants d'apprentissage profond. Avec l'augmentation de la puissance de calcul ainsi que des schémas d'optimisation plus complexes comme Momentum et Adam, nous sommes en mesure d'optimiser des modèles d'apprentissage profond complexes à grande échelle. Cependant, tous les algorithmes que nous avons vus utilisent seulement des informations dérivées de premier ordre. Il y a mieux ?

Laissez $f(\boldsymbol{w})$ être la fonction que vous voulez minimiser. Une façon simple est de faire une descente de gradient, c'est-à-dire de mettre à jour itérativement $\boldsymbol{w}$ en $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \nabla f(\boldsymbol{w}^{(t)})$. Essayons de faire mieux maintenant.

1. Considérez l'approximation Taylor de deuxième ordre de la fonction f à $\boldsymbol{w}^{(t)}$.

$$f(\boldsymbol{w}) \approx f(\boldsymbol{w}^{(t)}) + \left\langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \right\rangle + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Quelle est la solution du problème d'optimisation suivant ?

$$\arg\min_{\boldsymbol{w}} \left[f(\boldsymbol{w}^{(t)}) + \left\langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \right\rangle + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)}) \right] \tag{4}$$

Où $\boldsymbol{H}$ est la matrice hessienne (considérer défini positif), c'est-à-dire la matrice dont les éléments sont $\frac{\partial^2}{\partial w_i w_j} f(\boldsymbol{w}^{(t)})$. Ça vous dit quelque chose sur le "taux d'apprentissage" ?

2. Considérez un model perceptron multicouche composé de deux couches cachées de 512 neurones chacune. L'entrée se compose de 32 fonctionnalités, et la sortie est 10 dimensions pour représenter la probabilité de chaque classe dans un problème de classification de 10 classes. Combien de paramètres le modèle a-t-il (veuillez également inclure les termes de biais) ? Quelle est la taille de la matrice $\boldsymbol{H}$?
3. Compte tenu de l'échelle actuelle des architectures (p. ex., GPT, Dall-E, etc.), C'est facile de voir que le calcul de $\boldsymbol{H}$ est souvent intraitable. Cependant, souvent nous n'avons pas besoin de la matrice complète de $\boldsymbol{H}$, et au lieu de ça, on a juste besoin de produits Hessian-vector, c'est-à-dire $\boldsymbol{H}\boldsymbol{v}$ pour certains vecteurs $\boldsymbol{v}$. Laissez $g(\cdot)$ être la fonction de gradient de f, et considérez l'expansion de la série Taylor.

$$g(\boldsymbol{w} + \Delta\boldsymbol{w}) \approx g(\boldsymbol{w}) + \boldsymbol{H}\Delta\boldsymbol{w}$$

Pouvons-nous utiliser cette approximation pour estimer le produit $\boldsymbol{H}\boldsymbol{v}$, donné vecteurs $\boldsymbol{v}$ quelconques. (*Conseil : Considérez* $\Delta\boldsymbol{w} = r\boldsymbol{v}$ *avec petit* r)

4. L'expansion réelle de la série Taylor de la fonction de gradient est plutôt $g(\boldsymbol{w} + \Delta\boldsymbol{w}) = g(\boldsymbol{w}) + \boldsymbol{H}\Delta\boldsymbol{w} + O(||\Delta\boldsymbol{w}||^2)$. Dans ce cas, quelle est l'estimation de l'erreur lors du calcul de $\boldsymbol{H}\boldsymbol{v}$ comme dérivé ci-dessus, où $\Delta\boldsymbol{w} = r\boldsymbol{v}$. Vous pouvez utiliser la notation $O(\cdot)$ pour décrire ce que serait l'erreur.
5. Le problème avec cette solution approximative est qu'elle est souvent sujette à des erreurs numériques. Cela est dû au fait que nous avons besoin de r pour réduire l'erreur, mais quand r est très petit alors $r\boldsymbol{v}$ perd de la précision lorsqu'il est ajouté à vw. Cependant, peut-être pouvons-nous faire mieux et obtenir une solution plus exacte plutôt qu'une solution approximative ?
Considérez l'opérateur $\mathcal{R}-$, qui est défini comme

$$\mathcal{R}_v\{f(\boldsymbol{w})\} = \frac{\partial}{\partial r} f(\boldsymbol{w} + r\boldsymbol{v})\Big|_{r=0}$$

Montrez que

$$\mathcal{R}_v\{cf(\boldsymbol{w})\} = c\mathcal{R}_v\{f(\boldsymbol{w})\} \qquad \text{(Linearity under Scalar Multiplication)}$$
$$\mathcal{R}_v\{f(\boldsymbol{w})g(\boldsymbol{w})\} = \mathcal{R}_v\{f(\boldsymbol{w})\}g(\boldsymbol{w}) + f(\boldsymbol{w})\mathcal{R}_v\{g(\boldsymbol{w})\} \qquad \text{(Product Rule)}$$

6. Enfin, Utilisez l'opérateur $\mathcal{R}-$ pour dériver une équation pour calculer exactement le produit $\boldsymbol{H}\boldsymbol{v}$ à vecteur hessien.

L'algorithme que nous avons dérivé est le populaire *Newton Method*, il a été prouvé que la convergence est plus rapide que les méthodes de premier ordre sous certaines hypothèses. Non seulement avons-nous dérivé cette procédure d'optimisation, mais nous avons également découvert des moyens d'estimer de façon réaliste les produits vectoriels matriciels nécessaires.

Answer 3. Q.3.1. Psons :

$$l(\boldsymbol{w}) = \left[f(\boldsymbol{w}^{(t)}) + \left\langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \right\rangle + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)}) \right]$$

On calcule le gradient de la fonction $g(\boldsymbol{w})$ par rapport à $\boldsymbol{w}$.

On a pour la fonction

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}^\top \boldsymbol{x} = \left(\boldsymbol{A} + \boldsymbol{A}^\top\right) \boldsymbol{x}$$

Ainsi

$$\nabla_{\boldsymbol{w}} \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)}) = \frac{1}{2}\left(\boldsymbol{H} + \boldsymbol{H}^\top\right)(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Or $\boldsymbol{H}$ est symmétrique car c'est une hessienne donc

$$\nabla_{\boldsymbol{w}} \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

et on a

$$\nabla_{\boldsymbol{w}} \left\langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \right\rangle = \nabla f(\boldsymbol{w}^{(t)})$$

Ainsi,

$$\nabla_{\boldsymbol{w}} l(\boldsymbol{w}) = \nabla f(\boldsymbol{w}^{(t)}) + \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Ainsi vu que $\boldsymbol{H}$ est définie positive, donc g est convexe donc :

$$argmin(l(\boldsymbol{w})) = \boldsymbol{w}^{(t)} - \boldsymbol{H}^{-1} \nabla f(\boldsymbol{w}^{(t)})$$

Ce taux minimise la fonction l, il est donc l'optimale contre le taux d'apprentissage qui est approximatif.

Posons λ_{max} et λ_{min} le min et le max des valeurs propres de $\boldsymbol{H}$.

Donc ,

$$\frac{1}{\lambda_{max}} \nabla f(\boldsymbol{w}^{(t)}) \leq \ \boldsymbol{H}^{-1} \nabla f(\boldsymbol{w}^{(t)}) \leq \frac{1}{\lambda_{min}} \nabla f(\boldsymbol{w}^{(t)})$$

Nous pouvons donc borner le taux d'apprentissage en s'aidant des valeurs propres de $\boldsymbol{H}$

Q.3.2

Pour la couche d'entrée, on passe de 32 à 512 neurones, la matrice est de taille $(32, 512)$ et les biais $(1, 32)$, donc 16416 paramètres

Pour la couche 1, on passe de 512 à 512 neurones, la matrice est de taille $(512, 512)$ et les biais $(1, 512)$, donc 262656 paramètres

Pour la couche 2, on passe de 512 à 512 neurones, la matrice est de taille $(512, 512)$ et les biais $(1, 512)$, donc 262656 paramètres

Pour la couche finale, on passe de 512 à 10 neurones, la matrice est de taille $(512, 10)$ et les biais $(1, 10)$, donc 5130 paramètres

Donc en total, nous avons 546858 paramètres

La taille de la matrice hessienne $\boldsymbol{H}$ est $(546858, 546858)$

Q.3.3.

Soit r suffisamment petit, on pose $\Delta \boldsymbol{v} = r\boldsymbol{w}$

$$\begin{aligned} g(\boldsymbol{v} + \Delta \boldsymbol{v}) &\approx g(\boldsymbol{v}) + \boldsymbol{H}\Delta \boldsymbol{v} \\ &\approx g(\boldsymbol{v}) + r\boldsymbol{H}\boldsymbol{v} \end{aligned}$$

Ainsi,

$$\boldsymbol{H}\boldsymbol{v} \approx \frac{g(\boldsymbol{v} + \Delta \boldsymbol{v}) - g(\boldsymbol{v})}{r}$$

Q.3.4. On a $g(\boldsymbol{v} + \Delta \boldsymbol{v}) = g(\boldsymbol{v}) + \boldsymbol{H}\Delta \boldsymbol{v} + O(r^2||\boldsymbol{v}||^2)$

Ainsi, $\frac{g(\boldsymbol{v}+\Delta \boldsymbol{v})-g(\boldsymbol{v})}{r} - H\boldsymbol{v} = O(r||\boldsymbol{v}||^2)$, l'erreur est donc sz l'ordre $O(r||\boldsymbol{v}||^2)$, donc on prend le r le plus petit possible, en faisant attention au calcul numérique de $1/r$

Q.3.5.

$$\begin{aligned} \mathcal{R}_v\{cf(\boldsymbol{w})\} &= \frac{\partial}{\partial r} cf(\boldsymbol{w} + r\boldsymbol{v})\Big|_{r=0} \\ &= c\frac{\partial}{\partial r} f(\boldsymbol{w} + r\boldsymbol{v})\Big|_{r=0} \\ &= c\mathcal{R}_v\{f(\boldsymbol{w})\} \end{aligned}$$

Aussi ,

$$\begin{aligned}\mathcal{R}_v\{g(\boldsymbol{w})f(\boldsymbol{w})\} &= \frac{\partial}{\partial r} cf(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} \\ &= \frac{\partial}{\partial r} g(\boldsymbol{w}+r\boldsymbol{v}) * f(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} \\ &= g(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} * \frac{\partial}{\partial r} f(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} + f(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} * \frac{\partial}{\partial r} g(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} \\ &= \mathcal{R}_v\{f(\boldsymbol{w})\}g(\boldsymbol{w}) + f(\boldsymbol{w})\mathcal{R}_v\{g(\boldsymbol{w})\}\end{aligned}$$

Q.3.6.

On a

$$\boldsymbol{H}\boldsymbol{v} = \frac{\partial}{\partial r}\nabla f(\boldsymbol{w}+r\boldsymbol{v})\Big|_{r=0} = \mathcal{R}_v\{\nabla f(\boldsymbol{w})\}$$

Ainsi nous pouvons propager cette opérateur R sur le reste du réseau comme nous faisons avec la back propagation vue en cours.

Les équations utilisées sont donc celles de la backpropagation :

$$\begin{aligned}\nabla_{\mathbf{W}^{(k)}} f(\mathbf{x}) &\Longleftarrow \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} f(\mathbf{x})\right) nn\mathbf{h}^{(k-1)}(\mathbf{x})^\top \\ \nabla_{\mathbf{b}^{(k)}}(\mathrm{x}) &\Longleftarrow \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} f(\mathrm{x})\end{aligned}$$

$$\begin{aligned}\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} f(\mathbf{x}) &\Longleftarrow \mathbf{W}^{(k)^\top}\left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} f(\mathbf{x})\right) \\ \nabla_{\mathbf{a}^{(k-1)(\mathbf{x})}} f(\mathbf{x}) &\Longleftarrow \left(\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} f(\mathbf{x})\right) \odot \left[\ldots, g'\left(a^{(k-1)}(\mathbf{x})_j\right), \ldots\right]\end{aligned}$$

On implémente l'opérateur R sur ces équations et on suit le schéma de la propagation qui nous permettra de retrouve $\mathcal{R}_v\{\nabla f(\boldsymbol{w})\}$ et ensuite $\boldsymbol{H}\boldsymbol{v}$

Question 4 (3-5-5-7)**.** (**Base de convolution**)

1. Considérez un Réseau neuronal convolutif avec l'architecture suivante

   Image → Conv-3x3 → ReLU → Conv-3x3 → MaxPool-2x2 → Conv-3x3 → ReLU → Output

   où Conv-3x3 fait référence à une couche convolutionnelle avec un noyau 3x3 taille, et MaxPool-2x2 fait référence à l'opération max-pooling sur une fenêtre noyau 2x2. Les convolutions utiliser la foulée 1 et les opérations de mutualisation utiliser la foulée 2 et n'utilisent aucun remplissage. Image de résolution en entrée (32×32), trouvez la résolution de la sortie lors du passage de cette image à travers le réseau défini ci-dessus.
2. Considérez un autre Réseau neuronal convolutif avec l'architecture.:

   Image → Conv-5x5 → ReLU → Conv-5x5 → ReLU → Global Avg. Pool → FC-128 → FC-1 (Output)

où nous réutilisons la notation Conv-axa pour représenter les couches convolutionnelles avec la taille du noyau axa et foulée 1, et FC-b se réfère à un réseau neuronal Feedforward avec des neurones b. Cette architecture peut-elle gérer des images de taille variable ? Les Perceptron multicouches (sans couches de regroupement convolutionnelles ou globales) sont-ils capables de gérer des entrées de taille variable ? Si oui, pourquoi ; sinon, pourquoi pas ?

3. Couches de convolution *intrinsèquement* calculer une fonction linéaire (Similaire aux réseaux de transmission sans couches d'activation non linéaires !). Rappelez-vous que les fonctions d'entrée X, une fonction linéaire est n'importe quelle fonction qui peut être calculée comme, pour une matrice A.

   Considérez une couche de convolution avec le noyau W et entrez X donné par

$$W = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \quad X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix}$$

   Trouvez la carte de sortie en convoquant l'entrée X avec le noyau W, **Utilisation de la foulée 2 et zéro rembourrage de taille 1**. Pouvez-vous réécrire cette fonction linéaire (c'est-à-dire, quelle est la matrice A pour cette transformation ?). Vous pouvez envisager une extension des fonctionnalités en ligne, qui représente tous les éléments de la première rangée, puis tous les éléments de la deuxième rangée, et ainsi de suite comme un vecteur aplati, par exemple $X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{1,0} & \dots & x_{2,1} & x_{2,2} \end{bmatrix}$

4. Les convolutions transposées peuvent aider à améliorer les fonctionnalités spatialement. Considérez une couche de convolution transposée avec noyau $W = \begin{bmatrix} w_{0,0} & w_{0,1} \\ w_{1,0} & w_{1,1} \end{bmatrix}$, utilisé sur l'entrée $X = \begin{bmatrix} x_{0,0} & x_{0,1} \\ x_{1,0} & x_{1,1} \end{bmatrix}$, avec **stride = 2 et padding = 0**. La sortie est-elle toujours une transformation linéaire des caractéristiques d'entrée ?Pouvez-vous montrer ce que devrait être la matrice linéaire, compte tenu de l'expansion principale ligne pour les caractéristiques comme avant ?

Answer 4. Q.4.1. On utilise la formule

$$o = \left[\frac{i + 2p - k}{s} \right] + 1$$

la première couche Conv 3x3, donne une sortie $32 + 2 * 0 - 3 + 1 = 30$ donc de taille (30,30)

La deuxième couche Conv 3x3, donne une sortie $30 + 2 * 0 - 3 + 1 = 30$ donc de taille (28,28)

La couche MAxPool 2x2, donne une sortie $\left[\frac{28+2*0-2}{2} \right] + 1$, docn de taille (14,14)

La dernière couche Conv 3x3, donne une sortie $14 + 2 * 0 - 3 + 1 = 12$ donc de taille (12,12)

La sortie du modèle est donc de taille (12,12)

Q.4.2.

L'entrée de la convolution 5x5 avec stride 1 et padding nul, doit au minimum être égale à 5. car $output + 2 * padding \geq kernel$.Ainsi la dimension d'entrée de la dernière convolution doit être au minimum 5. En remontant le réseau, l'entrée de la première convolution doit donc être supérieur ou égal à 9.

Le reste du réseau ne donne pas d'importance à la taille de ses entrées.

Le réseau ne peut donc que traiter des images avec au moins (9,9) comme taille.

Pour les réseaux MLP, on ne peut pas faire passer une entrée qui ne respecte pas la première couche du modèle. La taille d'entrée est imposée par la taille in_channels de la qui est le nombre de neurones de la première couche. Q.4.3.

Posons

$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & x_{0,0} & x_{0,1} & x_{0,2} & 0 \\ 0 & x_{1,0} & x_{1,1} & x_{1,2} & 0 \\ 0 & x_{2,0} & x_{2,1} & x_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

On a donc l'output qui aura comme taille $o = \left\lfloor \frac{3+2*1-3}{2} \right\rfloor + 1 = 2$ Soit Z la matrice output de taille (2,2)

Pour tout (a,b) :

$$Z[a,b] = \sum_{i=1}^{3} \sum_{j=1}^{3} W[i,j] * Y[2a+i, 2b+j]$$

Ainsi,

$$Z[0,0] = x_{0,0} * w_{1,1} + x_{0,1} * w_{1,2} + x_{1,0} * w_{2,1} + x_{1,1} * w_{2,2}$$

$$Z[0,1] = x_{0,1} * w_{1,0} + x_{0,2} * w_{1,1} + x_{1,1} * w_{2,0} + x_{1,2} * w_{2,1}$$

$$Z[1,0] = x_{1,0} * w_{0,1} + x_{1,1} * w_{0,2} + x_{2,0} * w_{1,1} + x_{2,1} * w_{1,2}$$

$$Z[1,1] = x_{1,1} * w_{0,0} + x_{1,2} * w_{0,1} + x_{2,1} * w_{1,0} + x_{2,2} * w_{1,1}$$

On pose

$$A = \begin{pmatrix} w_{1,1} & 0 & 0 & 0 \\ w_{1,2} & w_{1,0} & 0 & 0 \\ 0 & w_{1,1} & 0 & 0 \\ w_{2,1} & 0 & w_{0,1} & 0 \\ w_{2,2} & w_{2,0} & w_{0,2} & w_{0,0} \\ 0 & w_{2,1} & 0 & w_{0,1} \\ 0 & 0 & w_{1,1} & 0 \\ 0 & 0 & w_{1,2} & w_{1,0} \\ 0 & 0 & 0 & w_{1,1} \end{pmatrix}$$

Ainsi donc

$$[Z_{0,0}, Z_{0,1}, Z_{1,0}, Z_{1,1}] = XA$$

Q.4.4. L'output de la convolution transversée de stride 2 est le suivant.

$$O = \begin{pmatrix} w_{0,0} * x_{0,0} & w_{0,1} * x_{0,0} & w_{0,0} * x_{0,1} & w_{0,1} * x_{0,1} \\ w_{1,0} * x_{0,0} & w_{1,1} * x_{0,0} & w_{1,0} * x_{0,1} & w_{1,1} * x_{0,1} \\ w_{0,0} * x_{1,0} & w_{0,1} * x_{1,0} & w_{0,0} * x_{1,1} & w_{0,1} * x_{1,1} \\ w_{1,0} * x_{1,0} & w_{1,1} * x_{1,0} & w_{1,0} * x_{1,1} & w_{1,1} * x_{1,1} \end{pmatrix}$$

Donc en posant $X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{1,0} & x_{1,1} \end{bmatrix}$

Et en posant

$$A' = \begin{pmatrix} w_{0,0} & w_{0,1} & 0 & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{0,0} & w_{0,1} & 0 & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & 0 & w_{1,0} & w_{1,1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & 0 & w_{1,0} & w_{1,1} \end{pmatrix}$$

Ainsi

$$O_{aplati} = XA'$$

, avec O_{aplati} le vecteur aplati venant de le matrice d'output O. En effet, nous avons la sortie une transformation linéaire du vecteur d'entrée .

Question 5 (3-2-3-7)**.** (**Champ récepteur**)

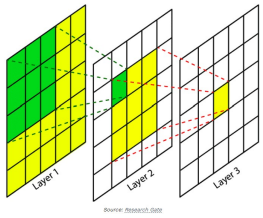

FIGURE 1 – Illustration of Receptive Field

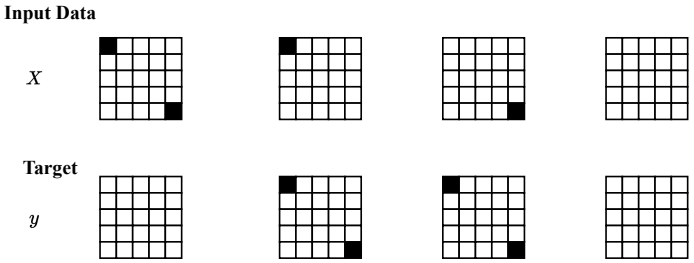

FIGURE 2 – Données illustratives pour la question 5, où X sont les images d'entrée et y sont les images de sortie de la vérité au sol. *Ceci n'est qu'une figure illustrative, les images réelles pourraient ne pas être de 5* $\times 5$

Le champ réceptif est défini comme la taille de la région dans l'entrée qui produit une caractéristique.. Par Exemple, Lorsque vous utilisez une couche Conv-3x3, chaque fonction est générée par une grille 3x3 dans l'entrée. Avoir deux couches Conv-3x3 successivement conduit à un champ réceptif 5x5. La figure ci-dessous en donne un exemple 1. Notez qu'il y a un champ réceptif associé à chaque fonctionnalité (par exemple, ici c'est pour la fonctionnalité de couleur jaune au Calque 3), où une fonctionnalité est une cellule dans la carte d'activation de sortie.

Considérez une architecture réseau composée uniquement de **4 Couches de convolutions, chacune avec la taille du noyau = 3, foulée = 1, zero-padding d'un pixel sur les quatre côtés, canaux intermédiaires = 16, et à un canal de sortie.**

1. Trouvez la résolution de la carte de sortie lorsque l'entrée du modèle est de taille 32×32. La taille de sortie est-elle la même que la taille d'entrée ?
2. Existe-t-il des cas où l'on pourrait envisager des extrants de la même taille que l'intrant (dans la résolution spatiale) ? Veuillez donner des exemples de tels cas.
3. Quel est le champ réceptif des caractéristiques finales obtenues à travers cette architecture, qui est, quelle est la taille de l'entrée qui correspond à une position de pixel particulière dans la sortie ?
4. Supposons qu'on vous donne des données, comme le montre la figure 2. qui est, vous avez 7×7 resolution images, où si les deux pixels de coin opposés en diagonale dans l'entrée sont les mêmes, puis la carte de sortie (de la même taille), a les coins correspondants en blanc. Si les deux coins de l'entrée sont différents, les coins correspondants de la carte de sortie sont noir. Tous les cas possibles de saisie sont décrits dans la figure 2.

   Compte tenu de l'architecture décrite ci-dessus, serait-il possible d'apprendre à bien faire avec ces données ? Autrement dit, le modèle peut-il obtenir une précision de 100% sur ces données ? Est-ce que quelque chose change si vous ajoutez une connexion résiduelle entre deux couches ?

Answer 5. Q.5.1.

la première couche Conv 3x3, donne une sortie $32 + 2 * 1 - 3 + 1 = 32$ donc de taille (30,30). Et pareillement pour les autres trois couches. Ainsi, nous avons une sortie final de taille (32,32).

Et en effet, pour ces couche de convolution 3x3, avec un padding = 1, un stride = 1. On a $o = i + 2 * 1 - 3 + 1 = i$

Donc la taille de sortie est égale à la taille d'entrée.

Q.5.2. Afin de détecter les contours par exemple, on voudra bien conserver la taille de l'image. On peut également utiliser la préservation de la taille de sortie pour mieux détecter les formes par le biais du gradient des images.

Q.5.3.

En remontant les différentes couches du réseau, on a :

Pour la dernière couche : 1x1 remonte d'une entrée de taille 3x3.

Pour la couche n 3 : 3x3 remonte d'une entrée de taille 7x7.

Pour la couche n 2 : 7x7 remonte d'une entrée de taille 15x15.

Pour la couche n 1 : 15x15 remonte d'une entrée de taille 31x31

Q.5.4.