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. Fonctions d'activation

1. La fonction d'activation donnée est $h(x) = \frac{x}{1+\exp(-\beta x)}$. Nous pouvons voir que $\beta = 0$ mène à une fonction d'activation linéaire, tandis que $\beta \to \infty$ se rapproche de la fonction d'activation ReLU.

2. Notez que $a \cdot \text{ReLU}(x-\alpha) + b \cdot \text{ReLU}(\beta - x)$ satisfait aux conditions. Supposer $\alpha = \beta = 0$, $a = 1.0$ et $b = -0.2$ ou une petite valeur. Il s'agit de la fonction d'activation LeakyReLU, qui est une fonction d'activation très populaire.
3. Nous pouvons voir la propriété d'invariance de traduction du softmax comme suit

$$\begin{aligned}\mathcal{S}(\boldsymbol{x}+c)_i &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^c \cdot e^{x_i}}{e^c \cdot \sum_j e^{x_j}} \\ &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\ &= \mathcal{S}(\boldsymbol{x})_i\end{aligned}$$

4. Supposer $j \in \arg\max x_j$. Alors, nous savons que $\log \mathcal{S}(\boldsymbol{x})_i = \log \mathcal{S}(\boldsymbol{x} - x_j)_i = (x_i - x_j) - \log \sum_k e^{(x_k - x_j)}$. On voit alors que cela équivaut à $(x_i - x_j) - \log\left(1 + \sum_{k \neq j} e^{x_k - x_j}\right)$. Cela résout-il les instabilités numériques ? Oui, parce que nous avons maintenant $(1 + +\text{ve terms})$ dans le journal résoudre le problème de sous-flux, ainsi que $(x_k - x_j) < 0$, Résoudre le problème 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 $z_n \sim \text{Categorical}(\pi_1, ..., \pi_K)$ puis générant y_n *conditionné* sur z_n en tant que $y_n \sim \text{Bernoulli}[\sigma(\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n)]$, 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. Le marginal

$$\begin{aligned}p(y_n = 1|\boldsymbol{x}_n) &= \sum_{k=1}^K p(y_n = 1|z_n = k, \boldsymbol{x}_n) p(z_n = k) \\ &= \sum_{k=1}^K \sigma(\boldsymbol{w}_k^T \boldsymbol{x}_n)\pi_k \\ &= \sum_{k=1}^K \frac{\pi_k}{1 + e^{-\boldsymbol{w}_k^T \boldsymbol{x}_n}}\end{aligned}$$

Architecture de réseau neuronal

Considérez une entrée $\boldsymbol{x}$ au réseau neural. Nous maintenons la couche cachée pour avoir la taille K. Nous maintenons également tous les termes de biais comme 0 dans le réseau neural.

Couche d'entrée	$x_1\ x_2\ \ldots\ x_D$
Couche masquée 1	$\sigma(\boldsymbol{w}_1^T\boldsymbol{x})\ \ldots\ \sigma(\boldsymbol{w}_K^T\boldsymbol{x})$
Couche de sortie	$\sum_{k=1}^{K} \frac{\pi_k}{\sigma(\boldsymbol{w}_k^T\boldsymbol{x})}$

TABLE 1 – Architecture de réseau neuronal équivalente

Notez que pour cette équivalence, nous définissons les paramètres suivants du réseau neuronal:

- $W^{(1)}$ est la matrice des poids reliant la couche d'entrée à la première couche cachée. Il a la structure suivante:

$$W^{(1)}_{D\times K} = \begin{bmatrix} . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ \boldsymbol{w}_1 & \boldsymbol{w}_2 & - & - & - & \boldsymbol{w}_{K-1} & \boldsymbol{w}_K \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \end{bmatrix}$$

  Cela nous donne les valeurs à la couche cachée $\boldsymbol{a}^{(1)}$ pour entrée $\boldsymbol{x}_n$ sans activation comme:

$$\begin{aligned} \boldsymbol{a}_n^{(1)} &= {W^{(1)}}^T \boldsymbol{x}_n \\ &= \begin{bmatrix} \boldsymbol{w}_1^T\boldsymbol{x}_n & \boldsymbol{w}_2^T\boldsymbol{x}_n & . & . & . & \boldsymbol{w}_{K-1}^T\boldsymbol{x}_n & \boldsymbol{w}_K^T\boldsymbol{x}_n \end{bmatrix}^T \end{aligned}$$

- Ensuite, nous appliquons le **sigmoid** non-linéarité sur la couche cachée. Ensuite, nous obtenons la valeur finale $\boldsymbol{z}^{(1)}$ au premier calque caché comme suit:

$$\begin{aligned} \boldsymbol{z}_n^{(1)} &= \sigma(\boldsymbol{a}_n^{(1)}) \\ &= \begin{bmatrix} \sigma(\boldsymbol{w}_1^T\boldsymbol{x}_n) & \sigma(\boldsymbol{w}_2^T\boldsymbol{x}_n) & . & . & . & \sigma(\boldsymbol{w}_{K-1}^T\boldsymbol{x}_n) & \sigma(\boldsymbol{w}_K^T\boldsymbol{x}_n) \end{bmatrix}^T \end{aligned}$$

- $W^{(2)}$ est la matrice des poids reliant la couche cachée au noeud de sortie. Il a la structure suivante:

$$W^{(2)} = \begin{bmatrix} \pi_1 & \pi_2 & . & . & . & \pi_{K-1} & \pi_K \end{bmatrix}^T$$

  Nous n'appliquons aucune activation sur la couche de sortie.

- Cela nous donne la valeur à la couche de sortie comme suit:

$$\begin{aligned} \text{out} &= {W^{(2)}}^T \boldsymbol{z}_n^{(1)} \\ &= \sum_{k=1}^{K} \pi_k \sigma(\boldsymbol{w}_k^T\boldsymbol{x}_n) \end{aligned}$$

Cela nous montre que le réseau neuronal conçu ci-dessus nous donne la sortie requise exactement. Par conséquent, le marginal requis peut également être considéré comme la sortie d'un réseau neuronal avec la structure ci-dessus.

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 ?

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

   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 $\boldsymbol{Hv}$ 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. schémas d'optimisation

1. Soit

$$L(\mathbf{w}) = f(\mathbf{w}^{(t)}) + \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^T \mathbf{H} (\mathbf{w} - \mathbf{w}^{(t)})$$

En réglant le gradient de $L(\mathbf{w})$ à 0, on obtient

$$\begin{aligned}
\nabla_{\mathbf{w}} L(\mathbf{w}) &= \nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}) + \nabla_{\mathbf{w}} \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle + \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^T \mathbf{H} (\mathbf{w} - \mathbf{w}^{(t)}) \\
&= 0 + \nabla f(\mathbf{w}^{(t)}) + \frac{1}{2} \nabla_{\mathbf{w}} (\mathbf{w}^T \mathbf{H} \mathbf{w} - \mathbf{w}^{(t)T} \mathbf{H} \mathbf{w} - \mathbf{w}^T \mathbf{H} \mathbf{w}^{(t)} + \mathbf{w}^{(t)T} \mathbf{H} \mathbf{w}^{(t)}) \\
&= \nabla f(\mathbf{w}^{(t)}) + \frac{1}{2}((\mathbf{H} + \mathbf{H}^T)\mathbf{w} - \mathbf{H}^T \mathbf{w}^{(t)} - \mathbf{H}\mathbf{w}^{(t)} + 0) \\
&= \nabla f(\mathbf{w}^{(t)}) + \frac{1}{2}((\mathbf{H} + \mathbf{H}^T)\mathbf{w} - (\mathbf{H} + \mathbf{H}^T)\mathbf{w}^{(t)}) \\
&= \nabla f(\mathbf{w}^{(t)}) + \frac{1}{2}(\mathbf{H} + \mathbf{H}^T)(\mathbf{w} - \mathbf{w}^{(t)}) \\
&= \nabla f(\mathbf{w}^{(t)}) + \mathbf{H}(\mathbf{w} - \mathbf{w}^{(t)}) \qquad \text{because } \mathbf{H} = \mathbf{H}^T \\
&= 0 \qquad \text{mettre le dégradé à 0} \\
\implies \mathbf{w} &= \mathbf{w}^{(t)} - \mathbf{H}^{-1} \nabla f(\mathbf{w}^t)
\end{aligned}$$

Notez que cela vous fournit un taux d'apprentissage adaptatif pour chaque paramètre en descente de pente standard, notamment $\mathbf{H}^{-1}$ qui dépend de votre situation.

2. La première couche est constituée de 512×32, la deuxième couche est constituée d' 512×512 poids et la couche de sortie finale se compose de 512×10 poids. Les biais sont respectivement $512, 512$ and 10 dimensionnel chacun. Ainsi, le nombre total de paramètres est: $512 \times 32 + 512 \times 512 + 512 \times 10 + 512 + 512 + 10$ qui sort de 284682. Par conséquent, le hessien serait une matrice de taille 284682×284682, ce qui est **Enorme** même pour un si petit réseau.
3. Supposer que nous perturbons $\mathbf{w}$ en direction de $\mathbf{v}$ par une petite quantité ; c.-à-d. $\delta\mathbf{w} = r\mathbf{v}$ où r est petit. Ensuite, on obtient

$$\begin{aligned} g(\mathbf{w} + r\mathbf{v}) &\approx g(\mathbf{w}) + rH\mathbf{v} \\ \implies H\mathbf{v} &\approx \frac{g(\mathbf{w} + r\mathbf{v}) - g(\mathbf{w})}{r} \end{aligned}$$

4. On peut voir que

$$\implies H\mathbf{v} = \frac{g(\mathbf{w} + r\mathbf{v}) - g(\mathbf{w})}{r} + \frac{O(r^2||\mathbf{v}||^2)}{r}$$

   Considérant $\mathbf{v}$ comme vecteur unitaire (ou tout vecteur fixe ; peu importe), cela se résume à $O(r)$.
5. Nous montrons d'abord la linéarité sous multiplication scalaire:

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

   Ensuite, nous montrons la règle du produit

$$\begin{aligned} \mathcal{R}_v\{f(\boldsymbol{w})g(\boldsymbol{w})\} &= \frac{\partial}{\partial r} f(\boldsymbol{w} + r\boldsymbol{v})g(\boldsymbol{w} + r\boldsymbol{v})|_{r=0} \\ &= \left[\left(\frac{\partial}{\partial r} f(\boldsymbol{w} + r\boldsymbol{v})\right) g(\boldsymbol{w} + r\boldsymbol{v}) f(\boldsymbol{w} + r\boldsymbol{v}) \left(\frac{\partial}{\partial r} g(\boldsymbol{w} + r\boldsymbol{v})\right)\right] \Bigg|_{r=0} \\ &= \mathcal{R}_v\{f(\boldsymbol{w})\}g(\boldsymbol{w}) + f(\boldsymbol{w})\mathcal{R}_v\{g(\boldsymbol{w})\} \end{aligned}$$

   ce qui achève la preuve.
6. Nous savons que

$$H\boldsymbol{v} = \frac{g(\boldsymbol{w} + r\boldsymbol{v}) - g(\boldsymbol{w})}{r} + O(r)$$

   Prendre la limite avec $r \to 0$, nous obtenons

$$\begin{aligned} H\boldsymbol{v} &= \lim_{r\to 0} \frac{g(\boldsymbol{w} + r\boldsymbol{v}) - g(\boldsymbol{w})}{r} + \lim_{r\to 0} O(r) \\ &= \lim_{r\to 0} \frac{g(\boldsymbol{w} + r\boldsymbol{v}) - g(\boldsymbol{w})}{r} \\ &= \frac{\partial}{\partial r} g(\boldsymbol{w} + r\boldsymbol{v})|_{r=0} \\ &= \mathcal{R}_v\{g(\boldsymbol{w})\} \end{aligned}$$

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

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

   $$\text{Image} \to \text{Conv-3x3} \to \text{ReLU} \to \text{Conv-3x3} \to \text{MaxPool-2x2} \to \text{Conv-3x3} \to \text{ReLU} \to \text{Output}$$

   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.:

   $$\text{Image} \to \text{Conv-5x5} \to \text{ReLU} \to \text{Conv-5x5} \to \text{ReLU} \to \text{Global Avg. Pool} \to \text{FC-128} \to \text{FC-1 (Output)}$$

   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. Bases de la convolution

1. La résolution de sortie est 12×12.
2. Oui, il peut gérer des entrées de taille variable. Oui, les systèmes MLP peuvent également gérer les entrées de taille variable en utilisant des outils de vision par ordinateur comme redimensionner les images à une résolution fixe. Cependant, ce n'est pas une solution aussi agréable que d'utiliser des couches convolutionnelles qui peuvent, de par leur conception, gérer des entrées de taille variable.
3. Le résultat de l'utilisation de la convolution est
$$\begin{bmatrix} w_{1,1}x_{0,0} + w_{1,2}x_{0,1} + w_{2,1}x_{1,0} + w_{2,2}x_{1,1} & w_{1,0}x_{0,1} + w_{1,1}x_{0,2} + w_{2,0}x_{1,1} + w_{2,1}x_{1,2} \\ w_{0,1}x_{1,0} + w_{0,2}x_{1,1} + w_{1,1}x_{2,0} + w_{1,2}x_{2,1} & w_{0,0}x_{1,1} + w_{0,1}x_{1,2} + w_{1,0}x_{2,1} + w_{1,1}x_{2,2} \end{bmatrix}$$
qui, une fois vectorisé, peut être écrit comme
$$\begin{bmatrix} w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & w_{1,0} & w_{1,1} & 0 & w_{2,0} & w_{2,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,1} & w_{0,2} & 0 & w_{1,1} & w_{1,2} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{1,0} & w_{1,1} & \end{bmatrix} \begin{bmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{2,0} \\ x_{2,1} \\ x_{2,2} \end{bmatrix}$$
Notez qu'il s'agit d'une opération linéaire.
4. Le résultat de l'application de la convolution transposée serait
$$\begin{bmatrix} w_{0,0}x_{0,0} & w_{0,1}x_{0,0} & w_{0,0}x_{0,1} & w_{0,1}x_{0,1} \\ w_{1,0}x_{0,0} & w_{1,1}x_{0,0} & w_{1,0}x_{0,1} & w_{1,1}x_{0,1} \\ w_{0,0}x_{1,0} & w_{0,1}x_{1,0} & w_{0,0}x_{1,1} & w_{0,1}x_{1,1} \\ w_{1,0}x_{1,0} & w_{1,1}x_{1,0} & w_{1,0}x_{1,1} & w_{1,1}x_{1,1} \end{bmatrix}$$
qui, lorsqu'il est vectorisé, peut être vu comme
$$\begin{bmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & 0 & 0 & 0 \\ 0 & w_{0,0} & 0 & 0 \\ 0 & w_{0,1} & 0 & 0 \\ w_{1,0} & 0 & 0 & 0 \\ w_{1,1} & 0 & 0 & 0 \\ 0 & w_{1,0} & 0 & 0 \\ 0 & w_{1,1} & 0 & 0 \\ 0 & 0 & w_{0,0} & 0 \\ 0 & 0 & w_{0,1} & 0 \\ 0 & 0 & 0 & w_{0,0} \\ 0 & 0 & 0 & w_{0,1} \\ 0 & 0 & w_{1,0} & 0 \\ 0 & 0 & w_{1,1} & 0 \\ 0 & 0 & 0 & w_{1,0} \\ 0 & 0 & 0 & w_{1,1} \end{bmatrix} \begin{bmatrix} x_{0,0} \\ x_{0,1} \\ x_{1,0} \\ x_{1,1} \end{bmatrix}$$
qui est aussi une opération linéaire.

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

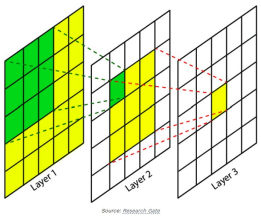

FIGURE 1 – Illustration of Receptive Field

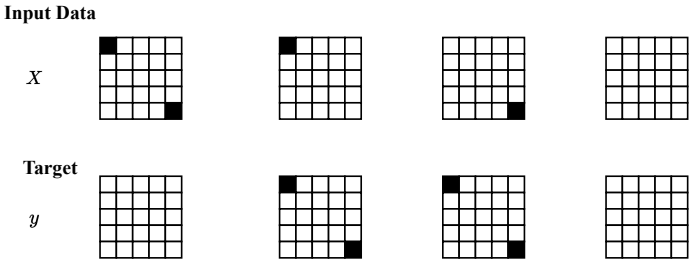

FIGURE 2 – Données illustratives pour la question 5, où X sont les images d'entrée et y sont les images de sortie de la vérité au sol. *Ceci n'est qu'une figure illustrative, les images réelles pourraient ne pas être de 5* $\times 5$

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. Champs réceptifs

1. La résolution de sortie est 32×32. Oui, ça reste pareil.
2. Exemples : tâches basées sur la segmentation et modélisation des scores / modèles de diffusion entre autres.
3. Le Champs réceptifs est 9×9.
4. Non ce n'est pas possible. Pour résoudre cette tâche, vous devez intégrer l'information des deux coins. Cela nécessite un champ réceptif d'au moins 13×13. L'architecture donnée a un champ réceptif plus petit, et donc le modèle ne peut pas faire cette tâche. Les connexions résiduelles n'impactent pas le champ récepteur.