Réponses aux questions

IFT780 - tp3

mnist_autoencoders.ipynb

Question 1 :

- Quelle est la relation entre l'architecture de l'encodeur et du décodeur ?
    - L'architecture du décodeur est la réciproque inverse de celle de l'encodeur, elle transforme la taille de la sortie de l'encodeur en la taille de son entrée initiale. L'encodeur permet de réduire la taille de l'input et le décodeur permet de repasser à cette taille origine : l'un effectue une réduction et l'autre une augmentation. Ils ont un rôle inverse de l'un l'autre.

- Pourquoi la fonction d'activation en sortie est-elle une **sigmoïde** ?
    - La sortie du décodeur est 28*28 soit de taille 784 et chacun de ces neurones représentent un pixel dans l'image de sortie. Etant donné que l'image est en noir et blanc, on peut coder chaque pixel comme son niveau de noir ou de blanc d'où l'utilisation d'une sigmoïde permettant de passer à un intervalle de [0,1].

- Quelle est la taille de l'espace latent du modèle?
    - La taille de l'espace latent est de 32.

Question 2 :

- Quel est l'utilité du dataloader?
    - Le dataloader est un « wrapper » qui utilise un itérable permettant de faciliter l'accès aux échantillons du dataset récupéré.

- Que fait la propagation avant considérant qu'elle utilise 2 réseaux de neurones? Quelles sont les entrées et sorties de cette fonction?
    - Notre propagarion avant permet de faire circuler nos données dans tout le réseau d'encodeur, pour obtenir notre représentation de nos images dans un espace latent. Puis, la représentation obtenue est un injectée dans le réseau décodeur. A la fin de ce ces deux réseaux, nous calculons une cross entropie binaire pour établir une métrique de différence entre notre image de départ et notre image décodée de l'espace latent.

Question 3 :

Pourquoi pensez-vous que l'espace latent de deux dimensions donne des reconstructions plus floues et moins précises ?

En comparant avec un espace latent d'une taille plus grande (32), on remarque que les données d'entrée projetées dans un espace d'une taille inférieure sont moins séparables et forment moins des clusters de points distincts d'où une reconstruction précise. Nous pouvons souligner que ce système est comparable à une compression des données dans un sous espace.

Question 4:

- La cellule précédente multiplie par 2 la taille de l'espace latent lors de l'instanciation de l'encodeur. Pourquoi ?
    - La taille de notre espace latent est multipliée par 2. En effet, dans le cas d'un auto-encodeur variationnel, le VAE projete les données en entrée en un sous espace latent symbolisant une gaussienne. Pour représenter cette gaussienne il est donc nécessaire d'allouer deux fois plus d'espace sur la couche de sortie de l'encodeur (pour le couple variance-moyenne).

- Dans la prochaine cellule, une certaine opération, une astuce, est effectuée afin d'échantillonner l'espace latent. Quelle est cette astuce et pourquoi est-elle utilisée?
    - Pour échantillonner notre espace latent, nous le considérons comme un couple de paramètres (variance, moyenne) définissant une gaussienne. Pour conserver des paramètres différentiables lors de notre backpropagation, nous construisons notre nouveau point comme une addition de la moyenne de notre gaussienne, auquelle nous additionnons l'écart type, multiplié par un terme aléatoire. Cette astuce est appelée reparametrization-trick/

- Quelle est la taille de notre espace latent ?
    - La taille de notre espace latent est de 2.

Question 5:

Quelle différence remarquez-vous entre les deux espaces latents ?

Nous remarquons que notre premier espace latent est bien plus éclaté que le premier. En effet, dans notre auto-encodeur variationnel, nos points intéressants sont contenus dans un petit périmètre autour de l'origine, contraire à un simple auto-encodeur. Cette approche est plus pratique puisqu'elle nous permet pour générer une donnée, de piocher dans un petit espace bien défini de notre espace latent.

cardiac_mri_autoencoders.ipynb

Question 1 :

- Combien de neurones contient l'encodeur ?

$$128 \times 128 \times 48 \times 2 + 96 \times 4 \times 64 \times 2 + 192 \times 32 \times 32 \times 2 + 48 \times 16 \times 16 \times 2 + 4 = 2\,777\,092$$

L'encodeur comprend 2 777 092 neurones. Notons qu'ici, nous ne comptons pas l'image d'entrée.

- Combien de paramètres contient l'encodeur ?
    - L'encodeur contient 798 148 paramètres.
- Quelle est la taille de l'espace latent ?
    - La taille est de l'espace latent est de 4 puisqu'il s'agit de la dimension de la sortie de l'encodeur.

Question 2 :

En quoi consiste l'opération ConvTranspose2d? Pourquoi est-elle nécessaire?

Elle sert à agrandir la taille des features map. Elle double la grandeur (largeur et longueur) à chaque fois qu'elle est utilisée. Elle est nécessaire puisque dans le décodeur on veut augmenter la taille des features map pour les ramener à la taille de l'image d'entrée de l'encodeur (256 par 256 ici). Elle consiste à multiplier chaque élément de la matrice par un filtre. C'est une multiplication scalaire-matrice qui donne ainsi une matrice en sortie. On fait ca pour chaque case (dépendamment du stride). C'est comme une convolution inverse. Voici un exemple de convolution transposé avec un stride de 1 :

1	2
0	3

*

2	3
4	1

=

2	3	4	6
4	1	8	2
0	0	6	9
0	0	12	3

Question 3:

- Quelles sont les différences entre les fonctions *vae_forward_pass* de ce notebook et celui de l'autoencodeur sur MNIST ?
    - Dans mnist, on linéarise l'entrée de l'encodeur et on utilise une cross-entropy binaire. On reshape aussi la sortie de l'encoudeur pour qu'il ait la même dimension que l'entrée. Dans MRI, on convertit l'entrée de l'encodeur en un one-hot vector multicouche (matrice).

Question 4:

- Réentraînez l'autoencodeur avec un espace latent à 2 dimensions. Comme dans le cas de MNIST, les reconstructions sont de bien moins bonne qualité. En quoi selon vous ces résultats sont de moins bonne qualité et pourquoi ne sont-ils pas flous comme pour MNIST?
    - Comme la dimension de l'espace latent diminue (4 vs 2), il y a une perte d'information.
    - Les images d'entrée de cet encodeur ont 4 channels d'entrée, elles contiennent plus d'informations que celle de mnist, qui n'en contient qu'un seul. Le réseau ne perd ainsi moins d'information quand il réduit ses images dans l'espace latent. De plus, le réseau s'entraine ainsi mieux puisqu'il a 4 fois plus de données.

rnn_image_captioning.ipynb

Question 1 :

- À quoi servent les jetons spéciaux [START], [END], [PAD], [UNK] ? Donnez une réponse par jeton.
    - [START] = Ce jeton représente le début de la phrase
    - [END] = Ce jeton représente la fin de la phrase
    - [PAD] = Ce jeton permet de compléter une phrase afin qu'elle puisse atteindre une taille fixe si elle est courte. (padding)
    - [UNK] = Ce jeton représente un mot inconnu jamais rencontré lors de l'entrainement.

Question 2:

- Pourquoi utiliser un modèle pré-entraîné ? Pourquoi ne pas entraîner aussi l'encodeur ?
    - Comme vu dans le cours, on utilise le transfert d'apprentissage en utilisant déjà un modèle pré-entrainé : cela permet d'éviter d'entrainer un modèle et de gagner donc considérablement du temps. Ici l'encodeur est une version modifiée de l'architecture ResNet50 dont on a chargé le modèle déjà pré-entrainé d'où l'inutilité d'entrainer l'encodeur.

- Expliquez dans vos propres mots quelle est la représentation de l'image en sortie de l'encodeur. Considérez la taille du tenseur, ce qu'il représente, et l'architecture Resnet50.
    - La sortie de l'encodeur contient les features de l'image et en utilisant ResNet50 spécialisé dans la classification d'image, nous obtenons en sortie un tenseur de taille 14x14x2048 contenant les caractéristiques de l'image. Et celle-ci va être ensuite utilisée par le décodeur pour obtenir une description de l'image.

Question 3:

- Quel est l'utilité de la "embedding layer"?
    - La couche « embedding » permet de représenter les mots en tant qu'un vecteur dense au lieu d'un one hot vecteur par exemple. Cela permet de réduire considérablement la taille d'entrée de la couche. En rêgle général, une couche d'embedding sert à réduire la dimensionnalité de l'espace de départ.

- Que représente la sortie du modèle dans la fonction forward? Et dans la fonction predict?

    - Dans la fonction forward, la sortie du modèle sont les variables c et h. c correspond au « cell state » et h correspond à l'« hidden state ». h contient l'information de mémoire à court terme, celle qu'on utilise dans les RNN régulier. c contient l'information à long terme, celle qui n'est pas là nécessairement dans l'évènement précédent. Dans la fonction prédire, la sorite du modèle est la même.

- Quelles sont les différences entre l'entrée du LSTM dans la fonction forward et la fonction predict? Quels sont les avantages d'avoir ces différences ?

    - En entrée du LSTM pour la fonction forward, nous avons un embedding global calculé une seule fois en amont, sur lequel on itère pour le fournir en entrée du LSTM. En revanche, dans la fonction predict, à chaque itération de Timestamp, l'embedding est regénéré pour être passé en paramètre d'entrée au LSTM. La liste à partir de laquelle ces embeddings sont générés est ensuite mise à jour. Les autres entrées sont h et c de la couche précédente.

Question 4

- Pourquoi utiliser l'entropie-croisée comme fonction de perte ? Pourquoi pas une fonction de perte L2, par exemple?

    - Comme le LSTM recourt à un softmax (donc des probabilité) en sortie, il est indispensable d'utiliser la cross entropie comme fonction de perte. Le softmax est calculé directement dans la fonction nn.CrossEntropyLoss().

Questions 5

- Les descriptions générées à la cellule précédente devraient parfois manquer de précision, inclure des éléments de trop ou en omettre ("a group of cows standing ..." versus "a cow standing ...", par exemple). Pourquoi croyez-vous que cela se produit ? Prenez en compte l'architecture des deux modèles et les tenseurs échangés entre ceux-ci. Indice: ce n'est pas dû à l'entraînement ou parce que le modèle n'a pas convergé ou par manque de données. Si vous êtes courageux.euses, vous pouvez entraîner le modèle plus longtemps pour vous en convaincre.

    - Il est plus que probable que ce problème provienne de l'architecture de notre modèle : il se pourrait qu'en prenant un encodeur plus performant ou en augmentant le nombre de décodeurs LSTM nous pourrions améliorer les performances.

- Pour cette implémentation, nous avons utilisé un ensemble de jetons correspondant grosso modo à chaque mot dans l'ensemble de toutes les descriptions. Combien de jetons font partie du vocabulaire du modèle ?

    - Le nombre de mots appartenant au vocabulaire est de 27964. (vocab_size = 27964)

- Quel aurait été l'impact sur le modèle, sur son nombre de paramètres et sur la difficulté du problème si nous avions choisi d'avoir plutôt un jeton par caractère ?

- Nous aurions perdu le sens entre plusieurs mots. Les mots auraient probablement été correct 2 par 2, mais on aurait perdu le sens avec les plus longues chaines de mots. Le nombre de paramètre aurait explosé et comme expliqué plus haut, cela aurait été difficile d'écrire quelque chose qui a du sens. Le contexte aura donc tendance à se perdre. Les modèles de langages basés sur les charactères sont plus longs à entrainer, mais sont plus aptes à gérer la ponctuation et les structures internes aux mots.