Algorithme TAGGER : reconstruire les images par morceaux

Quentin Lenet, Elliot Drees, Jérémie Boulic, Zhenliang Mu

[1]Ecole Centrale de Marseille – Parcours SISN

Nous allons expliciter l'Algorithme Tagger, écrit par Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hotloo Hao, Jürgen Schmidhuber et Harri Valpola. Son but est de reconstituer une image altérée par un bruit. Pour cela, l'algorithme repère et sépare les formes constituant la scène, et les débruite indépendamment les unes des autres. L'opération de débruitage est réalisée par un réseau de neurones sans supervision ou semi-supervisée.

1. Mathématisation du problème

L'image est représentée par un vecteur $x \in \mathbb{R}^N$, où N est le nombre de pixels. C'est ce vecteur qu'il faut retrouver.
En entrée du système, on envoie une version bruitée de ce vecteur, noté $\tilde{x}$. Dans le cas d'une image binaire (noir et blanc), $\tilde{x} = x + b$ où $b[i]$ suit la loi $\mathcal{B}(\beta)$ pour $i \in [1, N]$. Dans le cas d'une image couleurs (on approxime que le champ des couleurs est ici continu), $b[i]$ suit la loi $\mathcal{N}(0, \sigma^2)$ (bruit sous forme de vecteur gaussien avec matrice de covariance diagonale). Nous nous intéresserons seulement au deuxième cas.

L'algorithme, qui ne connaît que $\tilde{x}$, va chercher à déterminer $p(x|\tilde{x})$, qui représente la fonction de débruitage de l'image. Son estimation est la fonction q. Si l'estimation est précise, on peut aisément déduire x et donc l'image de départ. q est donc un descripteur de l'image reconstruite. $q(x)$ est donc la sortie de l'algorithme Tagger.

On définit $K \in \mathbb{N}$, le nombre de classes - de formes - à séparer, $k \in [1, K]$ l'indicateur de classe et $g_{k,j}$ l'indicatrice d'appartenance au groupe k du pixel x_j. Dans l'algorithme Tagger tel que présenté, ce nombre de classes est fixé à l'avance : on a donc un discriminateur et non un classifieur.

2. Description de l'algorithme

L'algorithme a été conçu autour du réseau d'apprentissage neuronal dit "Ladder", dans le but d'augmenter son efficacité.
Il est donc possible de le traiter en 2 parties : Une partie d'exploitation des données, qui définit les grandeurs à optimiser. Cette partie nourrit ensuite la partie du réseau de neurones, qui optimise les valeurs définies dans le première partie. Celui-ci débruite l'image et retourne alors l'estimation de la solution voulue. Il faut alors définir la forme de $q(x)$ (section 2.1) pour aider au mieux le réseau.

Le procédé est itératif : les données de sortie du réseau Ladder sont réinjectées dans la partie d'exploitation des données, qui renvoient à leur tour les paramètres actualisés au réseau neuronal. La fonction utilisée pour entraîner le réseau est la log-vraisemblance de

la fonction de débruitage au cours des itérations, soit $C(x) = \sum_i log(q_i(x))$. L'apprentissage peut se faire de manière non-supervisée ou semi-supervisée.
L'algorithme sépare l'image en différentes parties selon les formes qui la constituent. La boucle décrite ci-dessus est alors effectuée indépendemment pour chaque forme.

2.1. Exploitation de l'entrée

Lors de la partie d'exploitation, le but est de définir $q(x)$ et d'aider le réseau à optimiser les bons paramètres.

Pour cela, il faut définir z_k, qui représente la valeur attendue (l'espérance en quelque sorte) pour chaque groupe de l'image, et donc l'estimation de l'image **débruitée**. Cette valeur est mise à jour après chaque passage dans le réseau de neurones. z_k est initialisé à la valeur $E[x]$.
On définit également v, la variance de la valeur chaque pixel autour de leur valeur prévue. Cette valeur est **apprise** lors de la phase d'entraînement.
Cela nous permet de construire une estimation de la fonction de débruitage pour chaque classe : $q(x|g_k) = \mathcal{N}(z_k, vI)$.

Pour chaque classe, les 4 grandeurs sont :

- $\tilde{z}_k = q(\tilde{x}|g_k) \in \mathbb{R}^N$: il s'agit du descripteur de la classe k, i.e la projection de l'image **bruitée** reconstruite sur la classe k. En pratique, cela permet de rappeler au réseau de neurones la variance σ^2 du bruit, tout en donnant l'information de l'espérance z_k : on en déduit $\tilde{z}_k = \mathcal{N}(z_k, (v + \sigma^2)I)$
- $m_k = q(g_k) \in \mathbb{R}^N$: il s'agit de la probabilité pour chaque pixel d'appartenir à la classe k. En pratique, cela renseigne sur la localisation générale de la forme cherchée.
- δz_k, l'erreur de modélisation, qui est défini par le gradient de la fonction de coût : $\delta z_k = \frac{\partial C(\tilde{x})}{\partial z_k}$. En pratique, cette grandeur est proportionnelle à $\tilde{x} - z_k$, c'est-à-dire l'écart entre l'image bruitée et la reconstruction débruitée.
- $L(m_k)$ est le ratio de vraisemblance décrivant l'assignation de chaque pixel au groupe k.

On définit alors q comme suit : $q(x) = \sum_{k=1}^{K} q(x|g_k)q(g_k) = \sum_{k=1}^{K} \mathcal{N}(z_k, vI)m_k$.
Il s'agit donc des valeurs attendues de chaque pixel (avec une variance v) pour chaque classe, pondérée par la probabilité d'assignation à la classe. On voit donc bien que z_k et m_k sont les valeurs à optimiser par le réseau Ladder pour chaque classe.

2.2. Réseau neuronal Ladder

Le but du réseau neuronal, à partir des 4 grandeurs définies ci-dessus, est d'affiner le calcul de z_k et m_k, qui sont les deux paramètres de la fonction de débruitage présentée ci-dessus.
Le réseau Ladder, créé par Harri Valpola, étendu par Antti Rasmus et al, a pour but de combiner apprentissage supervisé et apprentissage non-supervisé, et est destiné à des opérations de débruitage.
Ainsi, le réseau maintient 2 fonctions de coût : une supervisée et une non-supervisée.

Dans notre cas, la fonction de coût non-supervisée prévue par Ladder est remplacée par notre $C(x)$. La fonction de coût supervisée est ajoutée à C durant la phase d'apprentissage.

Comme son nom l'indique, le réseau ladder pourrait se représenter par une échelle, avec un synapse en guise de barreau. Il s'agit donc d'un réseau unidirectionnel sans cycle, allant de la couche d'entrée, à la couche de sortie. Durant l'apprentissage, deux échelles sont mises côte à côte : une phase de propagation, et une phase de rétropropagation, avec les 2 échelles qui sont reliées. Une troisième échelle est utilisée pour la supervision (apprentissage semi-supervisé).

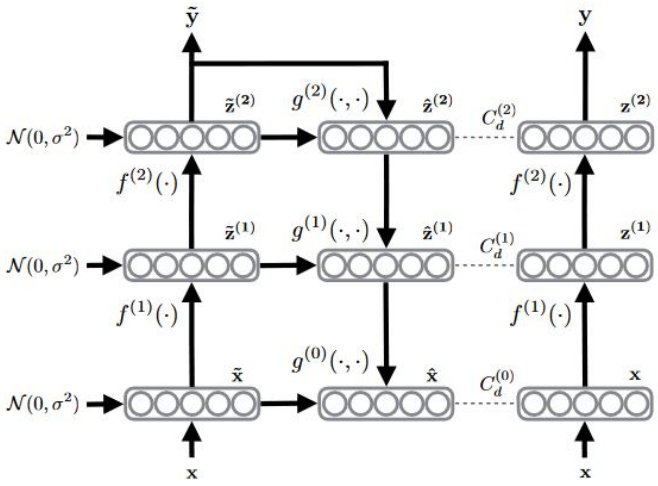

FIGURE 1. Représentation graphique du réseau Ladder de hauteur 2, par les auteurs de l'article arXiv :1507.02672

Appelons $\tilde{z}^{(l)}$ l'estimation de l'image bruitée à la couche l, et $\hat{z}^{(l)}$, l'estimation débruitée à la couche l. Pour arriver à une bonne estimation, on définit $C_d^{(l)} = ||z^{(l)} - \hat{z}^{(l)}||^2$ (Fonction de coût **supervisée**), à minimiser pour chaque barreau. Dans le cas semi-supervisé, cela permet de construire un bon $\hat{z}^{(l)}$.
Dans le cas non-supervisé, cette fonction n'est pas connue (on ne connaît pas la vraie entrée), il faut alors créer une fonction approximative de débruitage. Celle-ci est définie comme $\hat{z}^{(l)} = (\tilde{z}^{(l)} - \mu)v + \mu$, où v et μ sont respectivement des espérances et variances des données entrées dans le réseau à partir de l'image non-bruitée. Ces paramètres n'étant pas connus exactement, on en construit une approximation grâce à des paramètres d'apprentissage.

Lors de la propagation, les couches supérieures reçoivent les informations des précédentes couches pour faire de nouvelles estimations. Lors de la phase de rétropropagation (débruitage effectif), les couches supérieures permettent aux couches inférieures de débruiter : $\hat{z}^{(l)} = g^{(l)}(\tilde{z}^{(l)}, \hat{z}^{(l+1)})$

Dans **Tagger**, seule la dernière couche est concernée par cette fonction de coût, et

la présence de δz_k et de $L(m_k)$ dans les données d'entrée pourrait aider le réseau de neurones à déterminer la fonction de débruitage, et à bien séparer les classes, surtout dans le cas non-supervisé, où plus d'approximations sont faites. Par un système de projections, l'ensemble des 4 données traitées est renvoyé sous la forme d'une nouvelle reconstruction de l'image : z_k et m_k mis à jour, et donc le $q(x)$ recherché.

Concernant l'attribution des classes, elle est réalisée par la couche supérieure du réseau grâce à y.

3. Exemple et résultats obtenus

3.1. Illustration de l'algorithme

Voici un exemple de reconstruction non-supervisée du système Tagger. L'image à retrouver est un assemblage de textures rayées (Dataset Texture MNIST2).

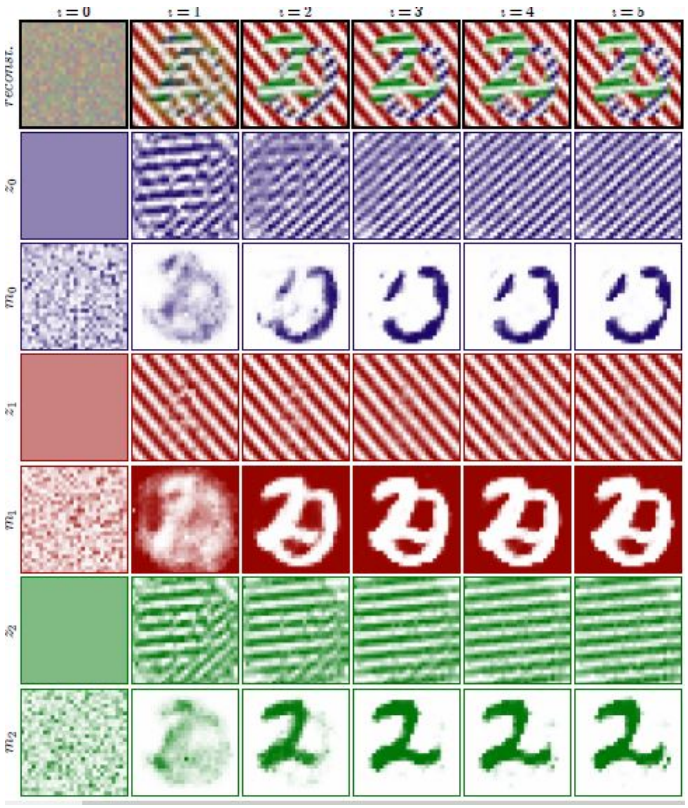

FIGURE 2. Résultats de l'algorithme sur une image texturée

On voit que z_k isole la texture utilisée pour le groupe qu'il reconstruit. C'est donc bien l'image projetée sur la classe étudiée. On voit également que m_k est une sorte de masque qui montre la localisation spatiale de chaque groupe. À chaque itération, les contours de chaque forme sont de plus en plus nets et précis. On remarque que 5 itérations suffisent à reconstruire l'image. Les auteurs indiquent que seuls 3 classes étaient nécessaires, mais que l'ajout d'une quatrième inutile accélère le processus, sans donner d'explication.

3.2. Résultats de tests

Concernant la performance, les auteurs la décrivent comme bien supérieure au réseau neuronal Ladder seul. La création d'un cadre permettant de donner des indicateurs précis au réseau permet donc une amélioration des performances.
Après des tests avec 5 humains confrontés aux mêmes images, il a été déterminé que Tagger était au moins aussi bons que les humains, et ce sur 2 Datasets différents.