Régularisation

Introduction à l'apprentissage automatique
Master Sciences Cognitives
Aix Marseille Université

Alexis Nasr

Plan

Introduction

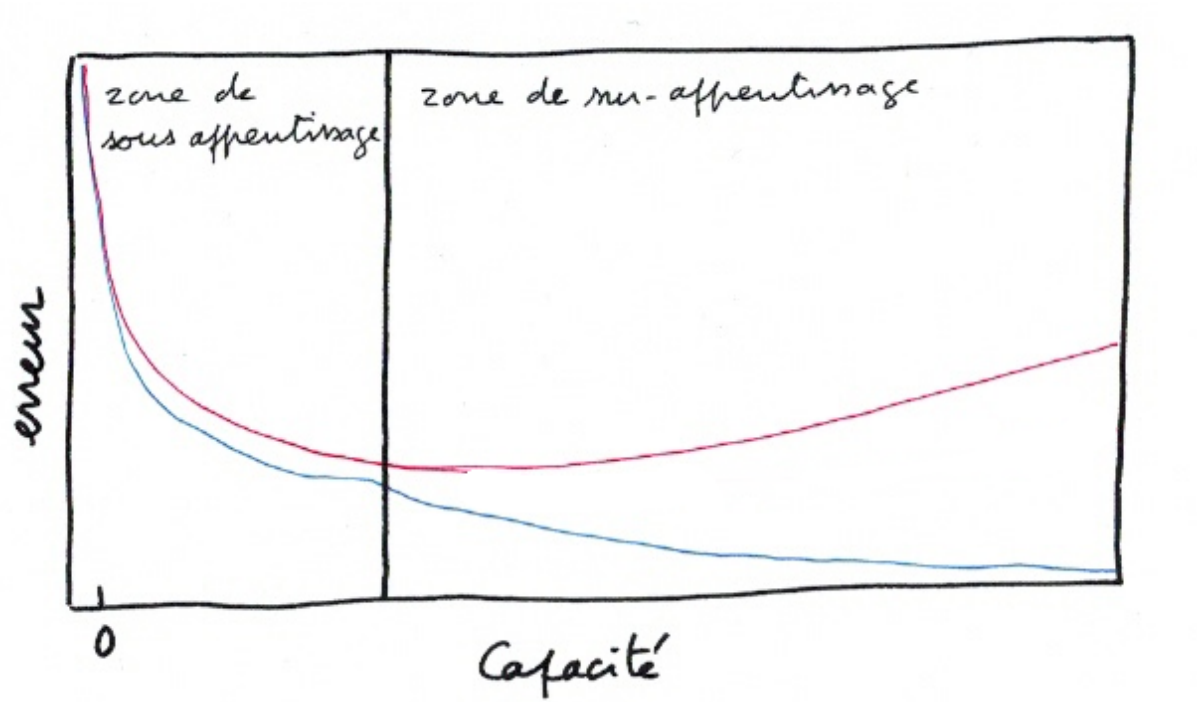
- Comment faire pour qu'un modèle appris sur des données d'apprentissage obtienne de bons résultats sur des données non vues ?
- L'aptitude d'un modèle à obtenir de bons résultats sur des données non observées est applée **généralisation** du modèle.
- Deux notions importantes :
  - **Erreur d'apprentissage** : valeur de la fonction d'erreur sur les données d'apprentissage.
  - **Erreur de généralisation** (ou erreur de test) : valeur de la fonction d'erreur sur des données de test.
- Lorsque l'erreur d'apprentissage est élevée, on parle de **sous-apprentissage**.
- Lorsque la différence entre l'erreur d'apprentissage et l'erreur de généralisation est élevée, on parle de **sur-apprentissage**.

Capacité d'un modèle

- On peut contrôler à quel point un modèle va avoir tendance à sous-apprendre ou à sur-apprendre en modifiant sa **capacité**.
- La capacité mesure la possibilité d'approximer une grande variété de fonctions.
- Exemples :
  - Un modèle de régression linéaire possède une capacité moindre qu'un modèle de régression polynomiale.
  - La capacité d'un perceptron multicouche possédant une couche cachée augmente avec le nombre de neurones de sa couche cachée.
- Un modèle de **faible capacité** aura tendance à **sous-apprendre**.
- Un modèle de **capacité élevée** aura tendance à **sur-apprendre**, en mémorisant les caractéristiques des données d'apprentissage, dont certaines sont absentes des données de test.
- La capacité d'un modèle doit être adapté à la **complexité du problème d'apprentissage** et à la **quantité de données**.

Régularisation

Toute modification apportée à un algorithme d'apprentissage qui a pour but de diminuer son erreur de généralisation.



- en rouge : erreur de généralisation
- en bleu : erreur d'apprentissage

Comment faire ?

- Modifier la fonction d'erreur
  - Ajout de pénalités à la fonction d'erreur
- Modifier les données
  - Augmentation de données
  - Ajout de bruit
- Modifier le modèle
  - Ajout de buit
  - Apprentissage multi-tâche
  - Early stopping
  - Bagging
  - Dropout

Pénalités

- Idée générale : diminuer la capacité d'un modèle en **contraignant** les valeurs possibles de ses paramètres.
- **Contraintes souples** : on favorise les modèles vérifiant les contraintes, sans éliminer la possibilité que les modèles violent ces contraintes.
- Mise en œuvre du principe du rasoir d'Occam (ou principe de parcimonie) :

  Pluralitas non est ponenda sine necessitate

  ou

  les multiples ne doivent pas être utilisés sans nécessité

  ou

  les hypothèses suffisantes les plus simples doivent être préférées
- La simplicité est représentée ici par des valeurs faibles des paramètres d'un modèle.

Pénalités

- On ajoute un terme ($\Omega(\mathbf{w})$) à la fonction d'erreur $E(\mathbf{w}; \mathcal{D})$ de sorte à **pénaliser** les valeurs trop élevées des paramètres :

$$\tilde{E}(\mathbf{w}; \mathcal{D}) = E(\mathbf{w}; \mathcal{D}) + \lambda \Omega(\mathbf{w})$$

- L'hyperparamètre λ permet de moduler l'importance donnée à la régularisation ($\lambda = 0$ pas de régularisation)
- Lors de la minimisation de $\tilde{E}$, on cherche à minimiser l'erreur E et, dans une certaine mesure, la pénalité Ω :

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w} \in \mathcal{W}} \tilde{E}(\mathbf{w}; \mathcal{D}) \\ &= \arg\min_{\mathbf{w} \in \mathcal{W}} (E(\mathbf{w}; \mathcal{D}) + \lambda \Omega(\mathbf{w})) \end{aligned}$$

- Quels choix possibles pour Ω ?

Régularisation L^2

$$\Omega(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} \mathbf{w}_i^2$$

- Calcul du gradient de $\tilde{E}$:

$$\begin{aligned}
\nabla_{\mathbf{w}}\tilde{E}(\mathbf{w};\mathcal{D}) &= \nabla_{\mathbf{w}}(E(\mathbf{w};\mathcal{D}) + \lambda\Omega(\mathbf{w})) \\
&= \nabla_{\mathbf{w}}E(\mathbf{w};\mathcal{D}) + \nabla_{\mathbf{w}}\lambda\Omega(\mathbf{w}) \\
&= \nabla_{\mathbf{w}}E(\mathbf{w};\mathcal{D}) + \nabla_{\mathbf{w}}\frac{\lambda}{2}\sum_{i=1}^{n}\mathbf{w}_i^2 \\
&= \nabla_{\mathbf{w}}E(\mathbf{w};\mathcal{D}) + \lambda\mathbf{w}
\end{aligned}$$

- Mise à jour de $\mathbf{w}$

$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}}E(\mathbf{w};\mathcal{D}) + \lambda\mathbf{w}) \\
&\leftarrow (1-\eta\lambda)\mathbf{w} - \eta(\nabla_{\mathbf{w}}E(\mathbf{w};\mathcal{D}))
\end{aligned}$$

- Avant la mise à jour des paramètres, la valeur de ces derniers est diminuée d'un facteur $\eta\lambda$: $(1-\eta\lambda)\mathbf{w}$

Régularisation L^1

$$\Omega(\mathbf{w}) = \sum_{i=1}^{n} |w|$$

- Calcul du gradient de $\tilde{E}$:

$$\begin{aligned}
\nabla_{\mathbf{w}} \tilde{E}(\mathbf{w}; \mathcal{D}) &= \nabla_{\mathbf{w}} (E(\mathbf{w}; \mathcal{D}) + \lambda \Omega(\mathbf{w})) \\
&= \nabla_{\mathbf{w}} E(\mathbf{w}; \mathcal{D}) + \nabla_{\mathbf{w}} \lambda \Omega(\mathbf{w}) \\
&= \nabla_{\mathbf{w}} E(\mathbf{w}; \mathcal{D}) + \nabla_{\mathbf{w}} \lambda \sum_{i=1}^{n} |\mathbf{w}_i| \\
&= \nabla_{\mathbf{w}} E(\mathbf{w}; \mathcal{D}) + \lambda signe(\mathbf{w})
\end{aligned}$$

- Mise à jour de $\mathbf{w}$

$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}} E(\mathbf{w}; \mathcal{D}) + \lambda signe(\mathbf{w})) \\
&\leftarrow (1 - \eta\lambda) signe(\mathbf{w}) - \eta(\nabla_{\mathbf{w}} E(\mathbf{w}; \mathcal{D}))
\end{aligned}$$

- La valeur du gradient est diminué d'une constante $((1 - \eta\lambda) signe(\mathbf{w}))$ et non pas d'une valeur proportionnelle à $\mathbf{w}$.

Augmentation de données

- La meilleure façon de faire en sorte qu'un modèle généralise mieux et de réaliser l'apprentissage sur une quantité **plus importante** de données.
- En pratique, la quantité de données à notre disposition est limitée.
- Dans certains cas, il est facile de créer de nouvelles données automatiquement, en modifiant les données d'apprentissage.
- Etant donné une paire $(\mathbf{x}, y)$, on peut générer la paire $(\mathbf{x}', y)$ en applicant une transformation à $\mathbf{x}$ pour produire $\mathbf{x}'$.

Augmentation de données

- L'augmentation n'est pas applicable à tout type de données.
- Elle est très efficace, par exemple, pour la reconnaissance d'objets à partir d'images, pour lesquelles il est facile de simuler de nombreuses variantions, tel que la translation [1], la rotation, le changement d'échelle ou la symétrie.
- Il faut néanmoins faire attention, en appliquant des variations, de ne pas modifier la classe correcte (une rotation de 180^o n'est pas conseillé pour reconnaître des chiffres manuscrits).

1. Même si les réseaux de neurones convolutionnels permettent déjà, en partie, d'introduire de l'invariance à la translation.

Robustesse au bruit

- Le bruit prend des formes différentes :
  - Dans les entrées, il peut être dû à des erreurs de mesure lors de la collection de données.
  - Dans les sorties, il peut être dû à des erreurs de mesure ou à des erreurs d'annotation.
- Il est possible d'ajouter du bruit à différents endroits :
  - sur les entrées des données d'apprentissage
  - sur les sorties des données d'apprentissage
  - sur les paramètres des modèles

Ajout de bruit sur les entrées

- Ajout de bruit sur les entrées des données d'apprentissage avant l'apprentissage.
- Peut aussi être vu comme un cas d'augmentation de données.
- Le bruit ajouté peut être aléatoire, mais il peut aussi être lié à des erreur spécifiques aux données.
- Exemple : erreur d'OCR lors de l'identification de la langue d'un document manuscrit.

Ajout de bruit sur les sorties

- La majorité des jeux de données contiennent des erreurs d'étiquetage.
- Exemple, dans un couple (x, y) d'un jeu d'apprentissage d'un classifieur, y n'est pas la catégorie correcte.
- Label smoothing :
  - On peut estimer que l'étiquette y est correcte avec une probabilité de $1 - \varepsilon$.
  - La probabilité des autres classes vaut $\frac{\varepsilon}{k-1}$, où k est le nombre de classes.
- Une telle distribution de probabilité est naturellement prise en compte par l'entropie croisée.

Ajout de bruit sur les paramètres

- Les paramètres sont vus comme des variables aléatoires (un paramètre peut être vu comme une variable qui peut prendre différentes valeurs proches, avec une certaine probabilité).
- On ajoute du bruit aux paramètres lors de l'apprentissage.
- Voir le Dropout.

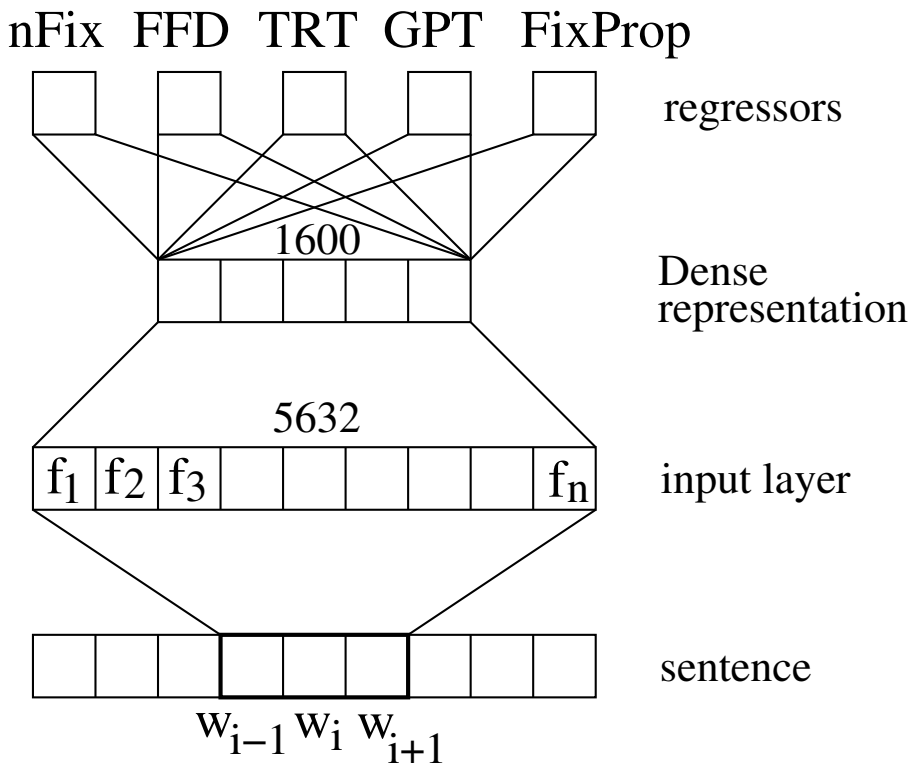
Apprentissage multi-tâches

- L'apprentissage multi-tâche consiste à réaliser n prédictions simultanément à partir d'un même réseau.
- Ce réseau est composé de $n + 1$ parties, n parties dédiées spécifiquement à chacune des n tâches et une partie partagée par tous.
- Les paramètres de la partie partagée seront mis à jour à partir de la propagation d'erreur des n différentes tâches.
- Il faut en général que les tâches soient corrélées.

Exemple : prédiction du mouvement oculaire

- Tâche : prédire le mouvement des yeux du lecteur d'un texte.
- $\mathbf{x}$: variables extraites du texte
  - longueur des mots
  - fréquence des mots
  - structure linguistique de la phrase
  - ...
- $\mathbf{y}$: variables oculométriques
  - durée de première fixation d'un mot
  - nombre de fixations d'un mot
  - ...

Exemple : prédiction du mouvement oculaire



Early stopping

- Lors de l'apprentissage de modèles ayant sufisamment de paramètre pour sur-apprendre, on observe souvent que l'erreur d'apprentissage diminue avec le temps mais que l'erreur sur les données de validation se met à augmenter lorsque l'on atteint la zone de sur-apprentissage.
- Il est donc possible d'obtenir un modèle ayant une meilleure erreur sur les données de validation (et, on l'espère, sur les données de test) en revenant aux paramètres minimisant l'erreur sur les données de validation.
- On garde une copie des paramètres ayant permis d'obtenir l'erreur la plus basse, jusque là, sur les données de validation.
- A l'issue de l'apprentissage, on renvoie ces paramètres plutôt que les derniers.
- L'algorithme s'arrête lorsque l'erreur sur les données de développement n'a pas diminué à l'issue des p dernières itérations.
- L'hyperparamètre p est appelé **patience**.

Bagging (Boostrap aggregating)

- Méthode de diminution de l'erreur de généralisation fondée sur la **combinaison de plusieurs modèles**.
- L'idée consiste à apprendre plusieurs modèles séparément puis à les faire voter.
- La raison pour laquelle une telle méthode fonctionne est que les différents modèles ne vont généralement pas faire tous les mêmes erreurs sur les données de test.
- Une manière simple de mettre en œuvre cette idée consiste à construire, à partir des données d'apprentissage, k jeux de données différents contenant le même nombre d'exemples.
- Puis d'entraîner un modèle sur chaque sous-ensemble.
- Etant donné un ensemble de données d'apprentissage de taille n, les différents jeux de données sont formés en choisissant aléatoirement n exemples des données d'apprentissage, avec remise.

Dropout

- Etant donné un réseau R, le dropout consiste à entraîner des sous-réseaux de R, obtenus en enlevant de manière aléatoire des neurones du réseau
- Lors de l'apprentissage, à chaque étape on choisit de manière aléatoire un certain nombre de neurones du réseau.
- Les neurones choisis verront leur sortie multipliée par zéro lors de cette étape de l'apprentissage.
- La proportion de neurones à neutraliser est un hyper-paramètre de l'apprentissage.

Sources

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016.