

Cours 5 - Les PMC avec Keras

Neila Mezghani

24 janvier 2022

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux librairies d'AM
 - PMC avec Keras

Configuration d'un perceptron multi-couche (PMC)

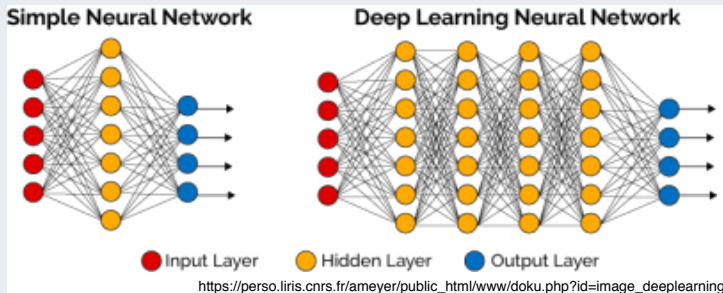
Configuration d'un PMC (1/4)

Les types de couches dans un MLP sont les suivants :

- Couche d'entrée : Variables d'entrée, parfois appelées couche visible.
- Couches cachés : Couches de nœuds entre les couches d'entrée et de sortie. Il peut y avoir une ou plusieurs de ces couches.
- Couche de sortie : Couche de nœuds qui produit les variables de sortie.

Configuration d'un PMC (2/4)

- Les réseaux neuronaux complexes, aux nombreuses couches de traitement, sont également appelés réseaux neuronaux profonds
- Ces réseaux alimentent les algorithmes d'apprentissage profond qui ont mené à des travaux révolutionnaires en IA.



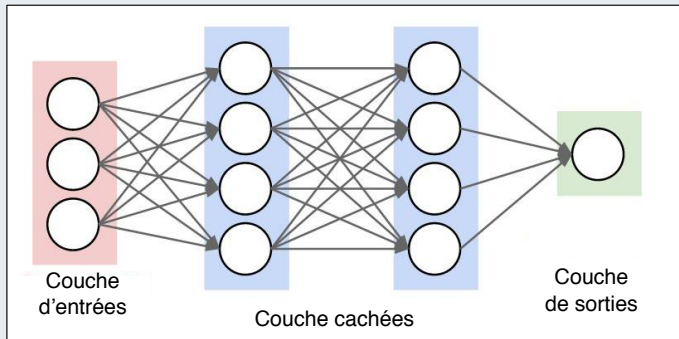
Configuration d'un PMC (3/4)

La forme d'un réseau neuronal peuvent être décrites par :

- Taille : Le nombre de nœuds dans le modèle.
- Largeur : Le nombre de nœuds dans une couche spécifique.
- Profondeur : Le nombre de couches dans un réseau neuronal.
- Architecture : La disposition spécifique des couches et des nœuds du réseau.

Configuration d'un PMC (4/4)

- Lorsque tous les neurones d'une couche sont connectés à chaque neurone de la couche précédente, la couche est une couche intégralement connectée, ou couche dense



PMC de classification et de régression

Retour sur le modèle de régression linéaire (1/5)

Un modèle linéaire effectue une prédiction en calculant une somme pondérée des variables d'entrée tout en y ajoutant un terme constant :

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \quad (1)$$

avec

- \hat{y} est la valeur prédite
- n est le nombre de variables
- θ_j est le j ème paramètre du modèle. Les θ_j sont l'équivalent des points synaptique dans les PMC.

Retour sur le modèle de régression linéaire (2/5)

- La régression logistique ou modèle logit est un modèle de régression binomiale
- Par opposition à la régression linéaire, la régression logistique fournit la probabilité du résultat (la logistique du résultat) au lieu du résultat lui même (\hat{y}).

Retour sur le modèle de régression linéaire (3/5)

- Le modèle de régression logistique permet d'estimer la probabilité qu'une observation x appartienne à la classe positive via la probabilité :

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\mathbf{x}^T \boldsymbol{\theta})$$

σ est une fonction sigmoïde

- Une fois la probabilité estimée, on peut faire une prédiction en se basant sur le modèle de régression logistique :

$$\hat{y} = \begin{cases} 0 & \text{si } \hat{p} < 0.5 \\ 1 & \text{si } \hat{p} \geq 0.5 \end{cases}$$

Retour sur le modèle de régression linéaire (4/5)

- La régression softmax, ou régression logistique multinomiale est considérée comme étant un modèle de régression logistique généralisé = Il permet de traiter plusieurs classes directement sans avoir à entraîner plusieurs classificateurs binaires puis à les combiner
- À partir des probabilités d'appartenance à chaque classe, le classificateur de régression softmax prédit la classe ayant la plus forte probabilité estimée
- Nous pouvons utiliser la fonction d'activation softmax pour l'intégralité de la couche de sortie.

Retour sur le modèle de régression linéaire (5/5)

- Pour la fonction cout, on utilise l'entropie croisée qui utilisée fréquemment pour mesurer l'adéquation entre un ensemble de probabilités estimées d'appartenance à des classes et les classes ciblées.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log \left(\hat{p}_k^{(i)} \right) \quad (2)$$

avec $y_k^{(i)}$ la probabilité cible que la i ème observation appartienne à la classe k

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux librairies d'AM
 - PMC avec Keras

PMC de classification et de régression

- Les PMC peuvent être utilisés pour des tâches de classification ou de régression.
- Si la couche de sortie comprend des valeurs continues alors on a une tâche de régression
- Si la couche de sortie contient des valeurs discrètes/catégorielles, on a une tâche de classification :
 - Binaire
 - Binaire à étiquettes multiples (multi-étiquettes)
 - Multi-classes

PMC de régression

- Les PMC peuvent être utilisés pour des tâches de régression.
- Exemple : On souhaite prédire le prix d'une maison en fonction de sa superficie et du nombre de chambres, on a besoin d'un seul neurone de sortie qui comprendra la valeur prédite.
- Pour réaliser une régression multi-variable (c'est-à-dire prédire de multiples valeurs à la fois), on a besoin d'un neurone de sortie pour chaque dimension de sortie (chaque variable).

PMC de régression : Fonction d'activation

- Lors de la construction d'un PMC pour la régression, généralement, il n'y a pas de fonction d'activation pour les neurones de sortie
⇒ avoir la possibilité de produire toute plage de valeurs en sortie.

PMC de régression : La fonction de perte

- La fonction de perte utilisée pendant l'entraînement est en général l'erreur quadratique moyenne (MSE)
- Si le jeu d'entraînement comprend un grand nombre de valeurs aberrantes, l'erreur absolue moyenne (MAE) peut-être utilisée.
- On peut fixer `loss` à `MeanSquaredError` ou bien `MeanAbsoluteError`

PMC de régression : Résumé

Hyperparamètre	Valeur type
Nombre de couches cachées	Dépend du problème, mais en général entre 10 et 100
Nombre de neurones de sortie	1 par dimension de prédiction
Fonction d'activation de la couche cachée	ReLU
Fonction d'activation de la sortie	Aucune ou ReLU par exemple
Fonction de perte	MSE ou MAE (en cas de valeurs aberrantes)

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux librairies d'AM
 - PMC avec Keras

PMC de classification : cas binaire (1/2)

- Dans le cas d'un problème de classification binaire, on a besoin d'un seul neurone de sortie
- Par exemple, un algorithme de détection de spam dans les courriers électroniques qui prédit si chaque message est un spam ou non
- La classe positive = le courrier électronique est un spam

PMC de classification : cas binaire (2/2)

- La fonction d'activation de la couche de sortie est une fonction logistique \implies On fixe activation à **tanh** ou bien activation à **ReLU**
- La sortie est une valeur entre 0 et 1 qui peut être interprétée comme la probabilité estimée de la classe cible = classe positive
- La probabilité estimée de la classe négative est égale à 1 moins la valeur de la classe positive

PMC de classification : cas binaire à étiquettes multiples

- Dans le cas d'un problème de classification binaire à étiquettes multiples, on a besoin de deux neurones de sortie = un neurone de sortie à chaque classe positive.
- La fonction d'activation de chaque neurone de sortie est une fonction logistique.
- Par exemple : un algorithme de détection de spam dans les courriers électroniques qui prédit si chaque message est un spam ou non tout en prédisant s'il est urgent ou non.
- Le premier neurone indique la probabilité que le courrier soit un spam, tandis que le second indique la probabilité qu'il soit urgent.

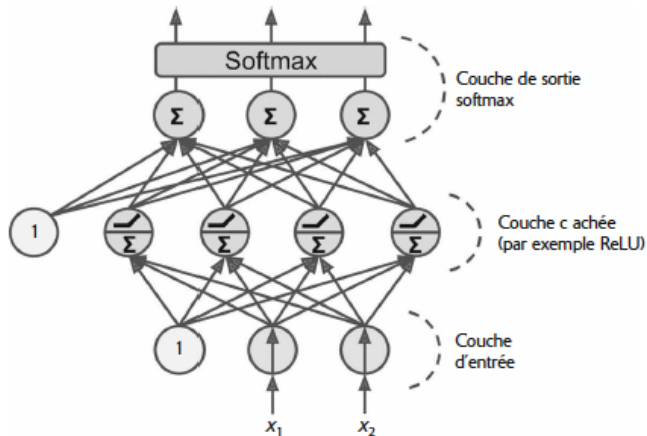
PMC de classification : cas multiclasse (1/2)

- Dans le cas de classification multi-classes, nous avons besoin d'un neurone par classe de sortie.
- Exemple : La classification des chiffres de 0 à 9.
- Nous pouvons utiliser la fonction d'activation softmax pour l'intégralité de la couche de sortie
⇒ On fixe activation à **softmax**

PMC de classification : Fonction perte

- La fonction de perte utilisée est la perte d'entropie croisée
- La fonction `loss` peut prendre les paramètres suivants :
 - ◇ Cas de classification binaire (Binary Cross-Entropy) :
`BinaryCrossentropy`
 - ◇ Cas de classification multi-classe (Categorical Cross-Entropy)
`CategoricalCrossentropy`

PMC de classification : cas multiclasse (2/2)



PMC de classification : Résumé

Hyperparamètre	Classification binaire	Classification binaire multiétiquette	Classification multi-classe
Couches d'entrée et couches cachées	Idem à la régression	Idem à la régression	Idem à la régression
Nombre de neurones de sortie	1	1 par étiquette	1 par classe
Fonction d'activation de la couche de sortie	Logistique	Logistique	Softmax
Fonction de perte	Entropie croisée	Entropie croisée	Entropie croisée

Contrôle de complexité

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux librairies d'AM
 - PMC avec Keras

Régularisation (1/3)

- La régularisation est une technique qui consiste à contrôler simultanément l'erreur du modèle sur l'ensemble des données d'entraînement et la complexité du modèle tout en évitant le sur-apprentissage.
- En effet, plus un modèle est « complexe », plus il est susceptible de sur-apprendre.
- L'objectif de la régularisation est de minimiser, non seulement, l'erreur du modèle, mais aussi, une nouvelle fonction objective qui est la somme d'un terme d'erreur et d'un terme mesurant la complexité du modèle.

Régularisation (2/3)

- La régularisation consiste en général à imposer des contraintes aux coefficients de pondération du modèle.
- Dans le cas de la régression linéaire, il s'agit donc de résoudre :

$$J(\theta) + \lambda \times \text{régulateur}(\theta)$$

- Plus λ est grand, plus le terme de régularisation est important.
- Plus λ est petit, plus l'erreur est importante.
- Si λ est suffisamment faible (et en particulier s'il est égal à zéro), on retrouvera la solution de la régression linéaire non-régularisée.

Régularisation (3/3)

- Dans les réseaux de neurones, une option simple pour éviter le sur-apprentissage consiste à introduire un terme de pénalisation dans le critère à optimiser $J(\theta)$.
- Le vecteur θ comprend les paramètres du modèle = Les poids synaptiques \mathbf{w} dans le cas d'un PMC.

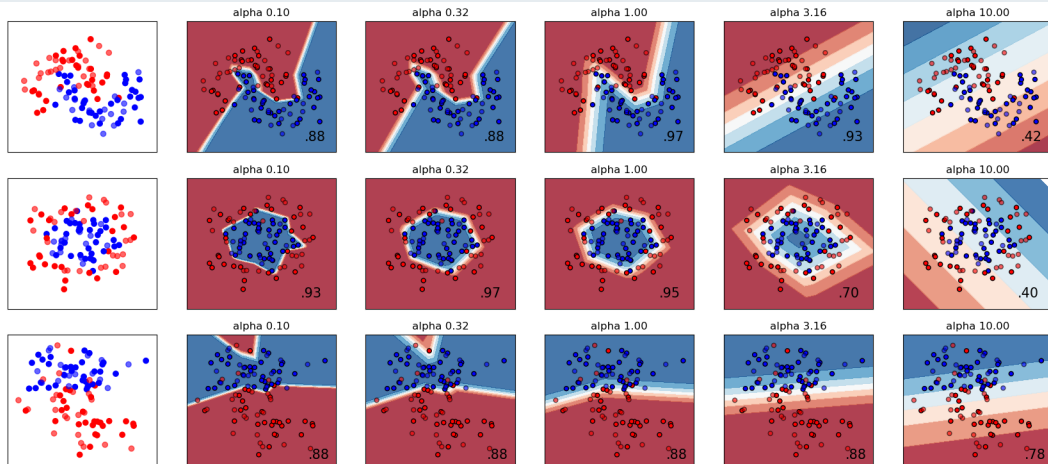
Régularisation de Tikhonov ou Ridge

- La régularisation de Tykhonov aussi appelée la régression ridge ou la régression de crête
- Cette régularisation utilise le carré de la norme du vecteur de poids synaptiques \mathbf{w}
- La fonction cout devient alors :

$$J(\mathbf{w}) = MSE(\mathbf{w}) + \lambda \sum_{i=1}^n \mathbf{w}_i^2$$

$\sum_{i=1}^n \mathbf{w}_i^2 = \|\mathbf{w}\|_2^2$ avec $\|\mathbf{w}\|_2$ qui représente la norme l2 du vecteur de poids synaptiques \mathbf{w} .

- On fixe `activity_regularizer` à `regularizers.l2(1e-5)`



Varying regularization in Multi-layer Perceptron

Régression Lasso (1/2)

- La régression ridge permet de réduire l'amplitude des coefficients d'une régression linéaire et d'éviter le sur-apprentissage.
- Maintenant, si on veut simplifier le modèle et annuler certains coefficients comme par exemple annuler les coefficients des variables non discriminante et leur attribuer un coefficient égal à zéro.
- Un modèle contenant plusieurs coefficients nuls est appelé un modèle parcimonieux (ou « sparse » en anglais).

Régression Lasso (2/2)

- La régularisation Lasso (Least Absolute Shrinkage and Selection Operator) ajoute un terme de régularisation à la fonction de coût en utilisant la norme l1 du vecteur de pondération au lieu de la moitié du carré de la norme l2.
- La fonction cout devient alors :

$$J(\mathbf{w}) = MSE(\mathbf{w}) + \lambda \sum_{i=1}^n |\mathbf{w}_i|$$

$|\mathbf{w}|$ représente la norme l1 du vecteur de pondération \mathbf{w}

- On fixe alors `activity_regularizer` à `regularizers.l1(1e-5)`

La régularisation Elastic net

- La régularisation Elastic net (« filet élastique ») est un compromis entre la régularisation Ridge et Lasso.
- Son terme de régularisation est un mélange des deux termes de régularisation contrôlé par le ratio de mélange (ou mix ratio) r :

$$J(\mathbf{w}) = MSE(\mathbf{w}) + r\lambda \sum_{i=1}^n |\mathbf{w}_i| + \frac{1-r}{2}\lambda \sum_{i=1}^n \mathbf{w}_i^2$$

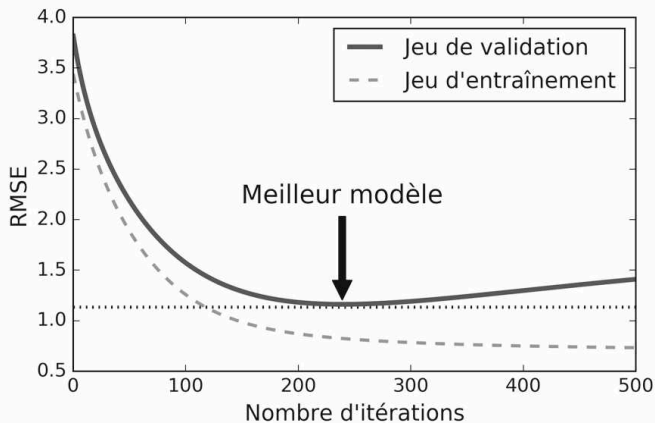
- Si $r = 0$, alors elastic net équivaut à une régression ridge
- Si $r = 1$, alors elastic net équivaut à une régression lasso.
- On fixe `activity_regularizer` à `regularizers.l1_l2(0.1,0.1)`

Régularisation ridge/lasso/elastic net ?

- Il est pratiquement toujours préférable d'avoir au moins un petit peu de régularisation.
- La régularisation ridge est un bon choix par défaut
- Si on veut annuler les coefficients de pondération des variables inutiles \Rightarrow La régression Lasso

Régularisation basée sur l'arrêt précoce

- L'arrêt précoce (early stopping) est une manière très différente de régulariser les algorithmes d'apprentissage itératifs tels que la descente de gradient
- Elle consiste à stopper l'apprentissage dès que l'erreur de validation atteint un minimum.



Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux librairies d'AM
 - PMC avec Keras

Choix et réglage des hyper-paramètres (1/2)

- La souplesse des PMC entraîne également une complexité relié aux nombreux hyperparamètres à ajuster.
- Une solution consiste simplement à tester de nombreuses combinaisons d'hyperparamètres et de voir celles qui donnent les meilleurs résultats sur l'ensemble des données de validation (idéalement via une K-fold cross-validation).
- Par exemple, nous pouvons employer **GridSearchCV** ou **RandomizedSearchCV** de manière à explorer l'espace des hyper-paramètres.

Choix et réglage des hyper-paramètres (2/2)

- La première étape est de créer une fonction qui construira et compilera un modèle Keras avec un ensemble donné d'hyperparamètres :
 - 1 Nombre de couches cachées
 - 2 Nombre de neurones par couche cachée
 - 3 Taux d'apprentissage
 - 4 Optimiseur
 - 5 Taille des lots
 - 6 Fonction d'activations
 - 7 Nombre d'itérations

Nombre de couches cachées (1/2)

- Un PMC doté d'une seule couche cachée peut théoriquement modéliser les fonctions assez complexes à condition de posséder suffisamment de neurones.
- Pour des problèmes complexes, les réseaux profonds ont une efficacité paramétrique beaucoup plus élevée que les réseaux peu profonds \implies Ils peuvent modéliser des fonctions complexes avec un nombre de neurones exponentiellement plus faible que les réseaux superficiels.

Nombre de couches cachées (2/2)

- La profondeur permet d'atteindre de meilleures performances avec la même quantité de données d'entraînement.
- Lorsque les problèmes sont plus complexes, on peut augmenter progressivement le nombre de couches cachées, jusqu'à ce qu'on arrive au sur-ajustement des données d'entraînement.
- Exemple : pour la classification d'image, on part de dizaines de couches mais ça peut atteindre des centaines.

Nombre de neurones par couche cachée (1/2)

- Comme pour le nombre de couches, on peut augmenter progressivement le nombre de neurones, jusqu'au sur-ajustement du réseau.
- En pratique, il est plus simple et plus efficace de choisir un modèle avec plus de couches et de neurones que nécessaire, puis d'utiliser l'arrêt prématuré et d'autres techniques de régularisation de façon à éviter le sur-ajustement.
- En général, il sera plus intéressant d'augmenter le nombre de couches que le nombre de neurones par couche.

Nombre de neurones par couche cachée (2/2)

Pourquoi ?

- ➊ L'ajout de nœuds supplémentaires dans la couche n'empêche pas le PMC de converger.
- ➋ Par contre, un nombre insuffisant de nœuds dans la couche cachée peut empêcher la convergence.

Taux d'apprentissage

- Le taux d'apprentissage est probablement l'hyperparamètre le plus important.
- Une bonne manière de déterminer un taux d'apprentissage approprié consiste à entraîner le modèle sur quelques centaines d'itérations, en commençant avec un taux d'apprentissage très bas (par exemple 10^{-5}) et de l'augmenter progressivement jusqu'à une valeur très grande (par exemple 10).

Optimiseur

- Le choix d'un optimiseur plus performant qu'une simple descente de gradient par mini-lots (avec l'ajustement de ses hyperparamètres) est également très important.
- On peut accélérer énormément l'entraînement en utilisant des solutions très répandues comme optimisation avec inertie, gradient accéléré de Nesterov, AdaGrad, RMSProp et optimisation Adam et Nadam.
- On fixe optimizer à une de ces valeurs **optimizers**.

Taille des lots

- La taille des lots peut également avoir un impact significatif sur les performances du modèle et le temps d'entraînement. Une taille de lot importante a l'avantage d'exploiter pleinement les accélérateurs matériels, comme les GPU
- Cependant un problème d'ordre pratique peut survenir : les tailles de lots élevées conduisent souvent à des instabilités de l'entraînement au début \implies la généralisation du modèle résultant risque d'être affectée.

Fonction d'activation

- La fonction ReLU est généralement un bon choix par défaut pour toutes les couches cachées.
- Pour la couche de sortie, cela dépend réellement de la tâche.

Nombre d'itérations

- Dans la plupart des cas, le nombre d'itérations d'entraînement n'a pas besoin d'être ajusté : il suffit d'utiliser, à la place, de l'arrêt prématuré.

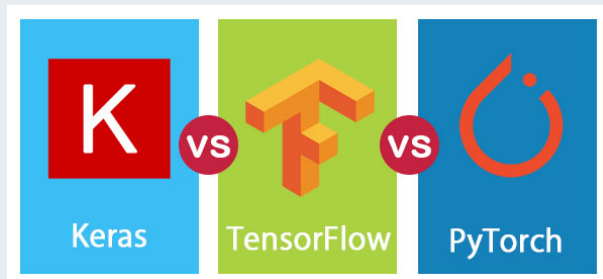
PMC avec Keras

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux bibliothèques d'AM
 - PMC avec Keras

TensorFlow, Keras et PyTorch

TensorFlow, Keras et PyTorch sont des bibliothèques très répandue parmi les professionnels du machine learning et du deep learning.



Tensorflow

- Tensorflow est une bibliothèque open-source développée par l'équipe Google Brain qui l'utilisait initialement en interne.
- Elle implémente plusieurs méthodes d'apprentissage machine basées sur le principe des réseaux de neurones profonds (deep learning).

Keras

- Keras est une bibliothèque Python qui encapsule l'accès aux fonctions proposées par plusieurs bibliothèques de machine learning, en particulier Tensorflow.
- Pourquoi ajouter une couche supplémentaire ? C'est parce qu'elle permet de construire, d'entraîner, d'évaluer et d'exécuter facilement toutes sortes de réseaux de neurones.
- Se référer à : [Keras API reference](#)

PyTorch

- PyTorch est une bibliothèque Python open-source d'apprentissage machine qui s'appuie sur Torch (en) développée par Facebook.
- PyTorch permet d'effectuer les calculs tensoriels nécessaires notamment pour l'apprentissage profond.
- La popularité de PyTorch a considérablement augmenté en 2018, essentiellement grâce à sa simplicité et à son excellente documentation contrairement à la première version de Tensorflow.

TensorFlow, Keras et PyTorch

Keras



- Prototypage rapide
- Petite base de données
- Multiple support Back-end

TensorFlow



- Large base de données
- Haute performances
- Détection d'objets

PyTorch



- Flexibilité
- Temps d'entraînement réduit
- Facilité de débogage

Plan du cours

- 1 Configuration d'un perceptron multi-couche (PMC)
- 2 PMC de classification et de régression
 - PMC de régression
 - PMC de classification
- 3 Contrôle de complexité
 - Régularisation
 - Choix et réglage des hyper-paramètres
- 4 PMC avec Keras
 - Introduction aux bibliothèques d'AM
 - PMC avec Keras

Structure séquentielle

- Keras présente une API séquentielle pour empiler les couches du réseau neuronal les unes sur les autres.
- Elle permet de créer un réseau de neurones vide dans lequel on peut ajouter des couches avec la fonction add.

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
model = Sequential()
```

Configuration d'un PMC


```
#instanciation du modele
PMC = Sequential()
# Ajout de la premiere couche "entree -> cachee"
# 30 neurones dans la premiere couche cachee
# ZTrain.shape[1] : dimension du vecteur de caracteristiques en entree
PMC.add(Dense(units=30,input_dim=ZTrain.shape[1],activation='relu'))
# Ajout de la seconde couche "cachee -> cachee"
# 30 neurones dans la deuxieme couche cachee
PMC.add(Dense(30,activation='relu'))
# Ajout de la troisieme couche "cachee -> sortie"
# mYTrain.shape[1] neurones dans la couche cachee =
# nb de modalites de la variable cible
PMC.add(Dense(units=mYTrain.shape[1],activation='softmax'))
```

Encodage des données (1/2)

- Keras ne sait pas manipuler directement une variable cible multivaluée.
- Il faut la transformer en une série d'indicateurs 0/1.
- Transformation du vecteur classe en matrice d'indicateurs

Encodage des données (2/2)

On peut utiliser la fonction `OneHotEncoder` ou bien `to_categorical()`

Index	Animal	One-Hot code	Index	Dog	Cat	Sheep	Lion	Horse
0	Dog		0	1	0	0	0	0
1	Cat		1	0	1	0	0	0
2	Sheep		2	0	0	1	0	0
3	Horse		3	0	0	0	0	1
4	Lion		4	0	0	0	1	0