Objectifs du TP :

- Découvrir l'apprentissage par perceptron multi-couches (les réseaux de neurones)
- Utiliser scikit-learn pour déterminer une architecture MLP performante pour le jeu de données MNIST.

Apprentissage par perceptron multi-couche sous sklearn

L'architecture des perceptrons multicouches (en anglais MLP, multi-layer perceptron) est un algorithme d'apprentissage supervisé fondé sur le perceptron, où des couches de neurones, dites couches cachées, sont ajoutées entre les entrées et la couche de sortie. La sortie d'un neurone caché (a_i) est obtenue par une combinaison linéaire des *p* variables d'entrées et d'un biais, toujours pondérés par des poids (Figure 1). L'intérêt de cette évolution est de surpasser les limitations du perceptron, afin d'optimiser la capacité des réseaux à apprendre à effectuer une tâche.

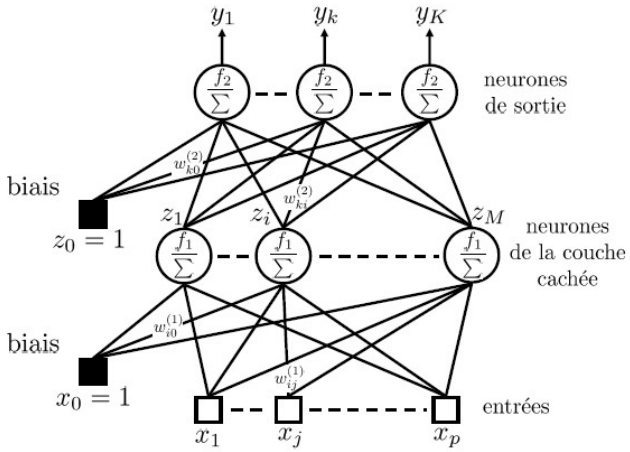

Figure 1 – Illustration d'un réseau de neurones à une couche cachée (MLP) (source : [Bishop, 1995])

1.1 La classe *MLPClassifier* : mise en pratique

Ce modèle optimise la fonction entropie croisée par la méthode du gradient stochastique. Les paramètres principaux de cette classe sont les suivants :

- **hidden_layer_sizes** est un tuple qui spécifie le nombre de neurones de chaque couche cachée; de l'entrée vers la sortie. Par exemple, une couche cachée de 55 neurones, hidden_layer_sizes = (55); pour trois couches cachées de taille respectivement 50, 12 et 100 neurones, hidden_layer_sizes =(50,12,100).
- **activation** définit la fonction d'activation pour les couches cachées : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'
  'identity', no-op activation, linear bottleneck, returns f(x) = x
  'logistic', logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
  'tanh', hyperbolic tan function, returns f(x) = tanh(x).
  'relu', rectified linear unit function, returns f(x) = max(0, x)
- **solver** {'lbfgs', 'sgd', 'adam'} spécifie l'algorithme utilisé pour minimiser la fonction de perte en sortie.
  'adam' est implémenté par défaut pour les grandes bases de données. Pour les petites bases de données, 'lbfgs' peut mieux converger.
- **alpha** par défaut 0.0001, est la magnitude de la régularisation L2.

C'est une méthode de régularisation (pour contrôler la capacité, empêcher l'overfitting) dont le but est de pénaliser les poids forts.
On rajoute la pénalité $\lambda \sum_i \theta_i^2$ à la fonction de coût.

- **max_iter** indique le nombre max d'itérations du solveur.
- **tol** est un facteur de tolérance qui permet d'arrêter le solveur précocement lorsque qu'il n'y a pas d'amélioration.

Les fonctions usuelles d'apprentissage et de test de sklearn sont les suivantes : (**fit, predict, score**).

Travail à effectuer.

Nous avons déjà utilisé le jeu de données MNIST. lors de TP précédent.
Ecrivez un programme TP2_prog1.py qui permet de :

- Charger le jeu de données MNIST..
- Diviser la base de données en 49000 lignes pour l'apprentissage (training) et le reste pour les tests.
- Construire un modèle de classification ayant comme paramètre : hidden_layer_sizes = (50), puis calculez la précession du classifieur ;
- Afficher la classe de l'image 4 et sa classe prédite.
- Calculez la précession en utilisant le package : metrics.precision_score(ytest_pr, ypredTest_pr,average='micro').
- Varier le nombre de couches de 1 entre (2 et 100) couches, et recalculer la précision du classifieur.
- Construire cinq modèles de classification des données mnist, avec des réseaux qui ont respectivement de 1 à 10 couches cachées, et des tailles de couches entre 10 et 300 neurones au choix d'une façon aléatoire. Quelles sont les performances en taux de bonne classification et en temps d'apprentissage obtenus pour chaque modèle ? Utilisez la fonction time() du package **time** pour mesurer le temps d'apprentissage d'un modèle.
- Étudier la convergence des algorithmes d'optimisation disponibles : L-BFGS, SGD et Adam.
- Varier les fonctions d'activation {'identity', 'logistic', 'tanh', 'relu'}.
- Changer la valeur de la régularisation L2 (paramètre α).
- Choisissez le modèle qui propose de meilleurs résultats.

- Sur chacun des cas précédents, tracez les différentes courbes d'erreurs.
  - Temps d'apprentissage,
  - Précision, rappel
  - Erreur avec le package metrics.zero_one_loss(ytest, ypredTest).
- A votre avis, quels sont les avantages et les inconvénients des A-nn : optimalité ? temps de calcul ? passage à l'échelle ?