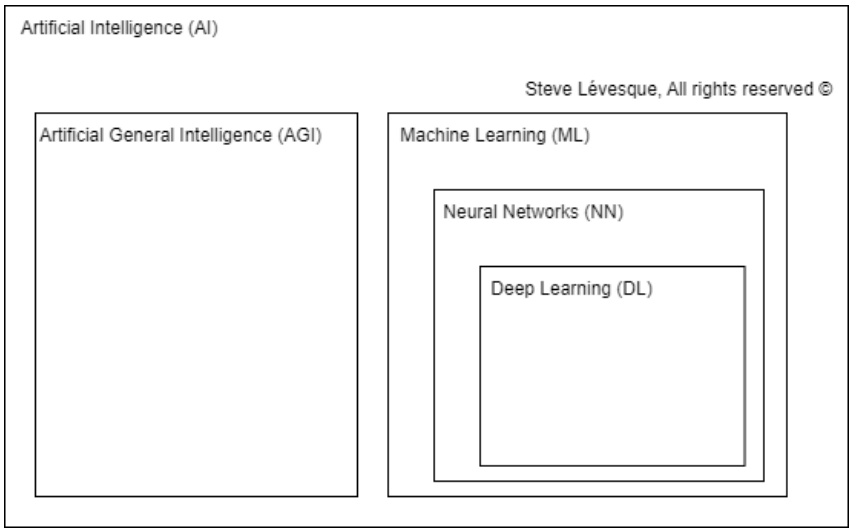
Intelligence Artificielle 1

Cours 4 à 6 - Introduction à l'AI

Table des matières

Introduction à la définition de l'intelligence artificielle et sa globalité

L'intelligence artificielle (AI) rassemble la globalité des techniques et théories ayant pour but de rendre les machines capables de faire des tâches simulant la capacité humaine.



Introduction à des tâches programmables reliées à l'intelligence artificielle

L'intelligence artificielle (AI) disponible dans les produits est fonctionnelle grâce à de la programmation.

Pour faire de la recherche appliquée en intelligence artificielle, il est nécessaire de programmer pour pouvoir générer des résultats d'expériences.

Introduction à des tâches programmables reliées à l'intelligence artificielle

Tâches courantes :

- Importation des données, séparation des ensembles de données (train/test sets) et nettoyage des données
- La compilation d'un modèle préétabli avant l'entraînement/l'évaluation (étapes et distinctions brèves, vu en détail au cours AI 3) :
  - La topologie du modèle (couches, etc.)
  - La fonction de perte (Loss function)
  - L'optimisateur (Optimizer)
  - Métrique (Metric)
- L'entraînement d'un modèle à partir de données
- L'évaluation d'un modèle à partir de données
- La représentation graphique des données et résultats d'évaluation
- La sauvegarde et chargement de modèles ML

Importation des données, séparation des ensembles de données (train/test sets) et nettoyage des données

Les données sont une composante importante de l'apprentissage automatique ("Machine Learning", ML).

Un modèle apprend à partir des données fournies à celui-ci pour prédire les résultats ("prédictions") grâce aux tendances qu'il établit en entraînant sur les données.

Sur ce, le nettoyage des données est une des étapes les plus importantes d'un projet de Machine Learning.

Importation des données, séparation des ensembles de données (train/test sets) et nettoyage des données

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  # Import data
4  (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
5
6  # Data cleaning - To make the training easier. Easier = less time and resources
7  # Normalisation is the task of scaling all values in a same level between 0 and
8  # We divide by 255 because the color spectrum is between 0 and 255.
9  x_train_normalized = x_train / 255.0
10 x_test_normalized = x_test / 255.0
```

La compilation d'un modèle avant l'entraînement/l'évaluation : étapes et distinctions brèves

Il y a quatre (4) composantes importantes lors de la compilation d'un modèle avant l'entraînement ou l'évaluation de celui-ci :

- La topologie du modèle (couches, etc.)
- La fonction de perte (Loss function)
  - Différence entre la fonction de perte (Loss function) et la fonction de coût (Cost function)
- L'optimisateur (Optimizer)
- Métrique (Metric)

La topologie du modèle (couches, etc.) - Résumé bref

Un modèle de réseaux de neurones est constitué de plusieurs couches de manière séquentielle.

Cette partie est complexe, il est généralement possible de se baser sur l'état de l'art (littérature) et d'utiliser les découvertes prouvées à cet effet.

La topologie du modèle (couches, etc.)

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  # All models in this course are sequential.
4  model = tf.keras.models.Sequential()
5
6  # The features are stored in a two-dimensional 28X28 array. Flatten that
7  # two-dimensional array into a one-dimensional 784-element array.
8  model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
9
10 # Define the first hidden layer.
11 model.add(tf.keras.layers.Dense(units=32, activation='relu'))
12
13 # Define a dropout regularization layer.
14 model.add(tf.keras.layers.Dropout(rate=0.2))
15
16 # Define the output layer. The units parameter is set to 10 because
17 # the model must choose among 10 possible output values (representing
18 # the digits from 0 to 9, inclusive).
19 # Don't change this layer.
20 model.add(tf.keras.layers.Dense(units=10, activation='softmax'))
```

La fonction de perte (Loss function) - Résumé bref

Cette fonction permet à chaque étape (epoch) de calculer et quantifier la marge d'erreur entre la valeur et la prédiction.

Un problème d'optimisation cherche à minimiser une fonction de perte.

Les problèmes courants et populaires sont dans le domaine de la classification (MNIST) ou de la régression (Real Estate $)

Les loss functions peuvent être réservées à seulement la classification ou la régression :

- Classification : Categorical cross-entropy, Hinge Loss, Log Loss
- Régression : Mean Absolute Error (MAE ou L1 Loss), Mean Squared Error (MSE ou L2 Loss), Uber Loss

Différence entre la fonction de perte (Loss function) et la fonction de coût (Cost function) - Résumé bref

Quelle est la différence entre la “Loss function” (i.e. L1 Loss) et la “Cost function” (i.e. MAE) ?

La “Loss function” est pour une prédiction et sa valeur réelle (“label”).

La “Cost function” est pour la totalité des observations (dataset / jeu de données) par aggrégation des valeurs des “Loss function”.

Des abus de langue peuvent être fait avec loss et cost, mais vous êtes invité.e.s à seulement vous rappeler que **les retours des functions de loss/cost permettent de mieux guider l’optimisateur**.

L'optimisateur (Optimizer) - Résumé bref

Un optimisateur permet d'obtenir graduellement le minimum le plus petit possible vis-à-vis les retours de la loss function après chaque étape (epoch) en définissant le réarrangement des paramètres pour y arriver.

Le but de l'optimisateur (et l'acte d'optimisation) est de trouver les paramètres optimaux pour le modèle qui réduit le minimum de la loss function le plus possible.

Métrique (Metric) - Résumé bref

La métrique permet de faire une évaluation du modèle et savoir s'il est compétent ou non (dépendamment des standards du milieu d'application).
Types de métriques possibles :

- Classification (maximiser = meilleur) : Précision (Accuracy)
- Régression (minimiser = meilleur) : MAE, MSE, Root Mean Squared Error (RMSE), R2

La fonction de perte (Loss function), l'optimisateur (Optimizer), et la métrique (Metric)

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  def create_model(my_learning_rate):
4
5      model = tf.keras.models.Sequential()
6      model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
7      model.add(tf.keras.layers.Dense(units=32, activation='relu'))
8      model.add(tf.keras.layers.Dropout(rate=0.2))
9      model.add(tf.keras.layers.Dense(units=10, activation='softmax'))
10
11     # Construct the layers into a model that TensorFlow can execute.
12     # Notice that the loss function for multi-class classification
13     # is different than the loss function for binary classification.
14     model.compile(
15         optimizer=tf.keras.optimizers.Adam(learning_rate=my_learning_rate),
16         loss="sparse_categorical_crossentropy",
17         metrics=['accuracy'])
18
19     return model
```

L'entraînement d'un modèle à partir de données

Il suffit de "fitter" le modèle avec les données X (features), y (labels), le nombre d'epochs et d'autres hyper-paramètres pour arranger les composantes du modèle en question.

L'action d'entraîner permet au modèle d'itérer les epochs pour réduire la Loss, optimiser les paramètres (optimisateur) et potentiellement améliorer la métrique.

L'entraînement d'un modèle à partir de données

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  def train_model(model, train_features, train_label, epochs,
4                  batch_size=None, validation_split=0.1):
5      """Train the model by feeding it data."""
6
7      history = model.fit(x=train_features, y=train_label, batch_size=batch_size,
8                          epochs=epochs, shuffle=True,
9                          validation_split=validation_split)
10
11     # To track the progression of training, gather a snapshot
12     # of the model's metrics at each epoch.
13     epochs = history.epoch
14     hist = pd.DataFrame(history.history)
15
16     return epochs, hist
```

L'évaluation d'un modèle à partir de données

L'évaluation est une fonction simple du modèle pour permettre d'évaluer celui-ci avec de nouvelles données inconnues au modèle.

En temps normal, les données d'entraînement (train set) sont connues du modèle, donc il faut utiliser des données de test (test set) sinon c'est de la "triche" au sens de la littérature et de la communauté de l'intelligence artificielle puisqu'il connait déjà le contenu.

L'évaluation d'un modèle à partir de données

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
# Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0

# Evaluate against the train set.
print("\n Evaluate the new model against the train set:")
my_model.evaluate(x=x_train_normalized, y=y_train, batch_size=batch_size)
# Evaluate against the test set.
print("\n Evaluate the new model against the test set:")
my_model.evaluate(x=x_test_normalized, y=y_test, batch_size=batch_size)

"""
 Evaluate the new model against the train set:
15/15 ------------------ 0s 2ms/step - accuracy: 0.9727 - loss: 0.0995

 Evaluate the new model against the test set:
3/3 -------------------- 0s 3ms/step - accuracy: 0.9567 - loss: 0.1503
"""
```

La représentation graphique des données : résultats d'entraînement et d'évaluation

Il est très pertinent de représenter graphiquement les résultats d'entraînement et d'évaluation pour pouvoir faire des conclusions sur la performance d'un modèle (décisions de haut niveau administratives/gestion).

La représentation graphique des données et/ou résultats d'évaluation

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2  def plot_curve(epochs, hist, list_of_metrics):
3      """Plot a curve of one or more classification metrics vs. epoch."""
4      # list_of_metrics should be one of the names shown in:
5      # https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#defin
6      plt.figure()
7      plt.xlabel("Epoch")
8      plt.ylabel("Value")
9
10     for m in list_of_metrics:
11         # Show accuracy as 0 to 100% for better reading
12         if 'acc' in m:
13             x = hist[m] * 100
14         else:
15             x = hist[m]
16         plt.plot(epochs[1:], x[1:], label=m)
17
18     plt.legend()
19     plt.savefig(list_of_metrics[0] + '.png')
20     plt.show()
```

La représentation graphique des données : résultats d'entraînement et d'évaluation

NB : Ça diffère d'un problème à un autre

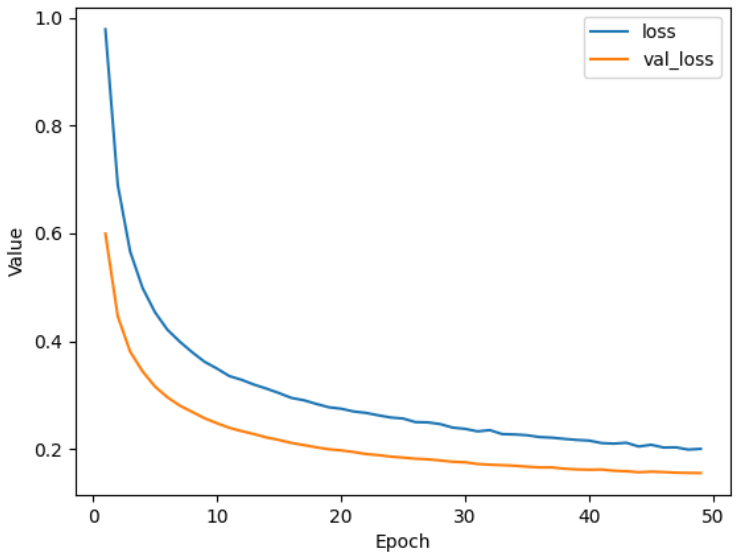


Figure: Entraînement - Perte (loss) - Le plus minimal, le meilleur

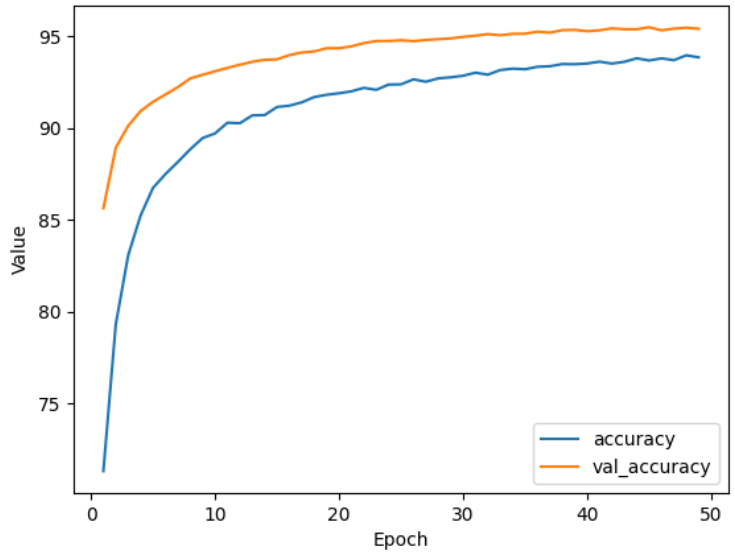


Figure: Évaluation - Précision en % (accuracy) - Le plus haut, le meilleur

La représentation graphique des données : résultats d'entraînement et d'évaluation - Overfitting/Underfitting

Underfitting : Lorsqu'un modèle est trop simple pour capturer la complexité des données parce qu'il n'y a pas assez de données et/ou puisque l'entraînement était trop court au niveau des itérations (epochs).
Overfitting : Lorsqu'un modèle est entraîné avec trop de données et/ou pendant trop d'itérations (epochs), il commence à apprendre du bruit et des entrées de données inexactes de notre ensemble de données.

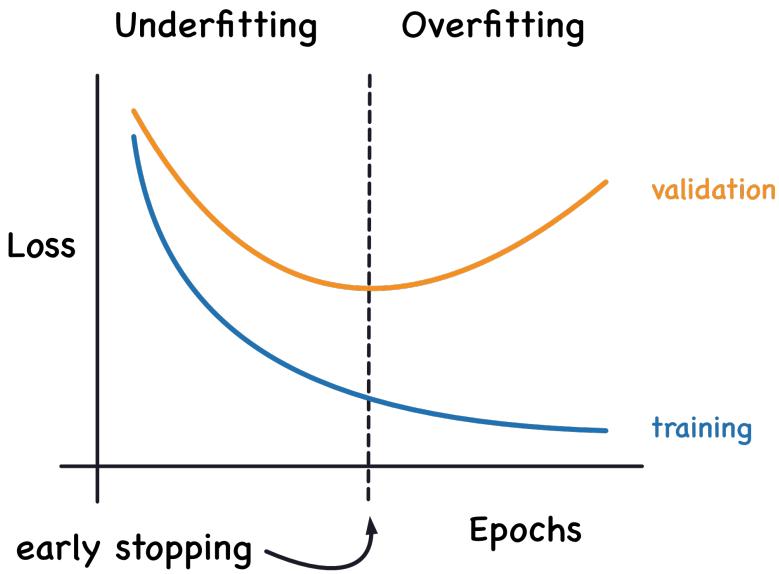


Figure: https://sourestdeeds.github.io/blog/overfitting-and-underfitting/

La représentation graphique des données : résultats d'entraînement et d'évaluation - Biais/Variance

Biais (Bias) : Le biais fait référence à l'erreur due à des hypothèses trop simplistes dans l'algorithme d'apprentissage.

Variance : La variance est l'erreur due à la sensibilité du modèle aux fluctuations des données d'entraînement.

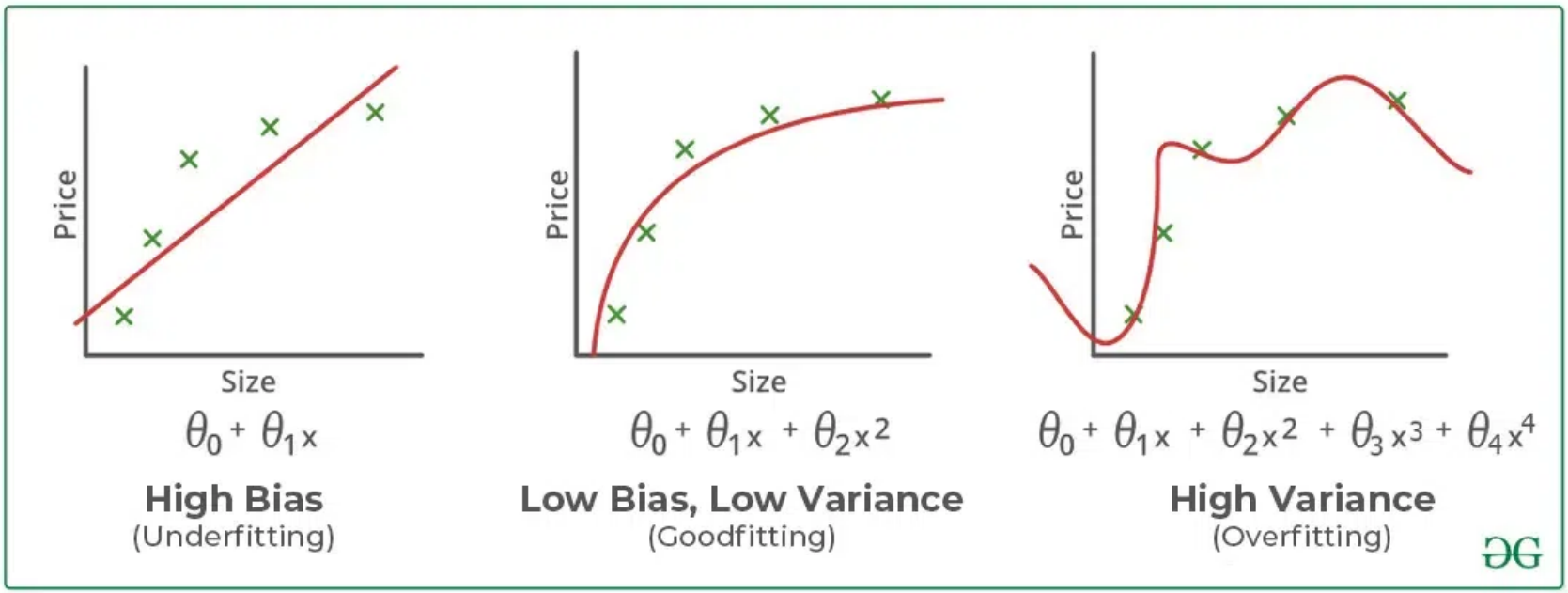


Figure: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Partie d'exécution globale

Listing: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  learning_rate = 0.003
4  epochs = 50
5  batch_size = 4000
6  validation_split = 0.2
7
8  # Establish the model's topography.
9  my_model = create_model(learning_rate)
10
11 # Train the model on the normalized training set.
12 epochs, hist = train_model(my_model, x_train_normalized, y_train,
13                            epochs, batch_size, validation_split)
14 # Plot a graph of the metric vs. epochs. (Omitted for simplicity)
15
16 # Eval model
17 my_model.evaluate(x=x_test_normalized, y=y_test, batch_size=batch_size)
18 # Save model for quick reuse in another app (without training again)
19 my_model.save('mnist_trained.keras')
```

La sauvegarde et chargement de modèles ML

Après avoir sauvegardé le modèle, il est possible de l'utiliser sur n'importe quel appareil qui a assez de mémoire sans avoir à l'entraîner à nouveau puisque les poids sont sauvegardés.

Un IoT peut être utilisé ! Les modèles chargés demandent seulement l'espace de stockage embarqué et une RAM minimale pour son exécution.

Les prédictions sont quasi instantanées (normalement à moins que le modèle soit vraiment complexe).

La sauvegarde et chargement de modèles ML

```
1  # Steve Levesque, All rights reserved
2  import tensorflow as tf
3  import numpy as np
4  from matplotlib import pyplot as plt
5
6  def show_mnist_graphic_number(img):
7      img = np.array(img, dtype='float')
8      pixels = img.reshape((28, 28))
9      plt.imshow(pixels, cmap='gray')
10     plt.show()
11
12 # Get data of MNIST
13 (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
14 # Load model
15 new_model = tf.keras.models.load_model('mnist_trained.keras')
16 # Show the model architecture
17 new_model.summary()
18 # Show a prediction
19 img1 = x_train[1]
20 print(new_model.predict(np.reshape(img1, (1, 28, 28))))
21 show_mnist_graphic_number(img1)
```

Bibliographie

- https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en
- https://stephenallwright.com/loss-function-vs-cost-function/
- https://stephenallwright.com/l1-loss-function/
- https://stephenallwright.com/interpret-mae/
- https://sourestdeeds.github.io/blog/overfitting-and-underfitting/
- https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/