Intelligence Artificielle 3

Cours 3 - Compilation avancée modèles

Table des matières

La fonction de perte (Loss function)

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 courrants et populaires sont dans le domaine de la classification (i.e. MNIST) ou de la regression (i.e. Real Estate $).

La fonction de perte (Loss function)

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)

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**.

La fonction de perte (Loss function) - Classification (Cross-Entropy)

Lorsque l'étiquette vraie t est 1, la perte d'entropie croisée se rapproche de 0 lorsque la probabilité prédite p s'approche de 1.

Lorsque l'étiquette vraie t est 0, la perte d'entropie croisée se rapproche de 0 lorsque la probabilité prédite p s'approche de 0.

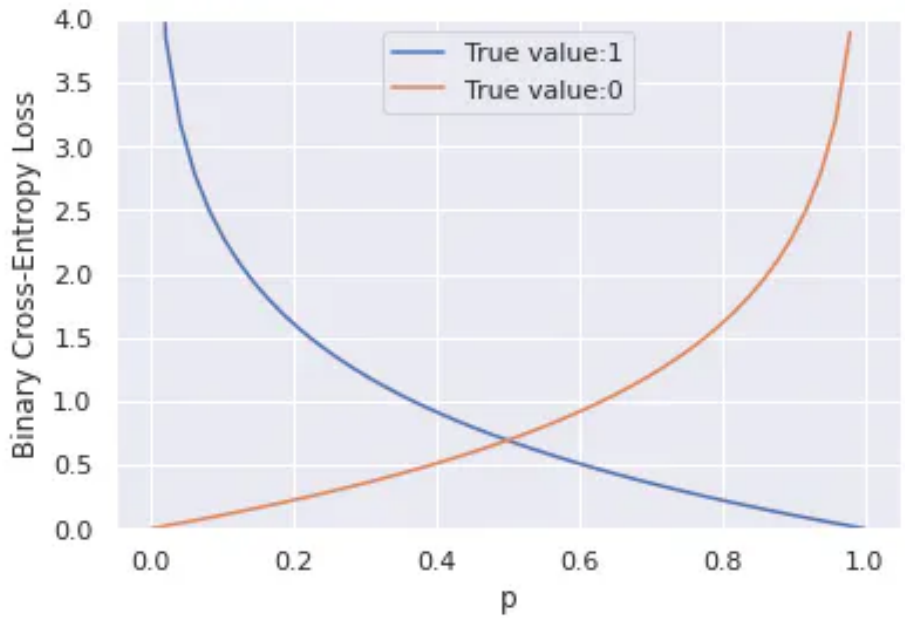


Figure: https://www.pinecone.io/learn/cross-entropy-loss/

La fonction de perte (Loss function) - Classification (Cross-Entropy)

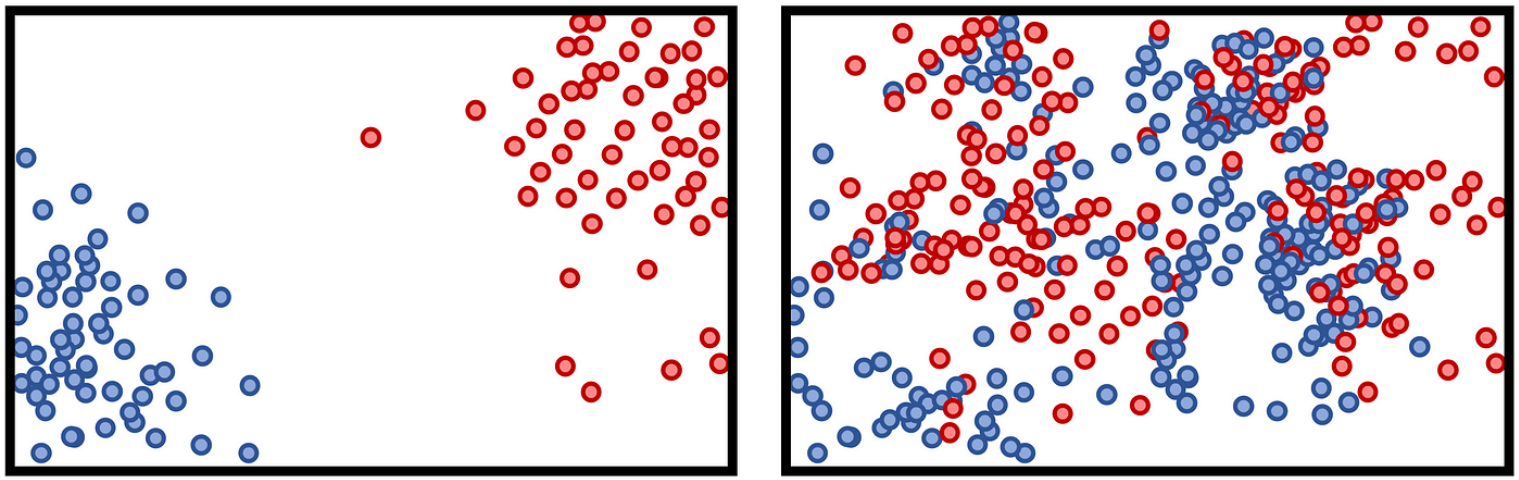


Figure: https://towardsdatascience.com/understanding-entropy-the-golden-measurement-of-machine-learning-4ea97c663dc3

La fonction de perte (Loss function) - Régression (MAE, MSE, etc.)

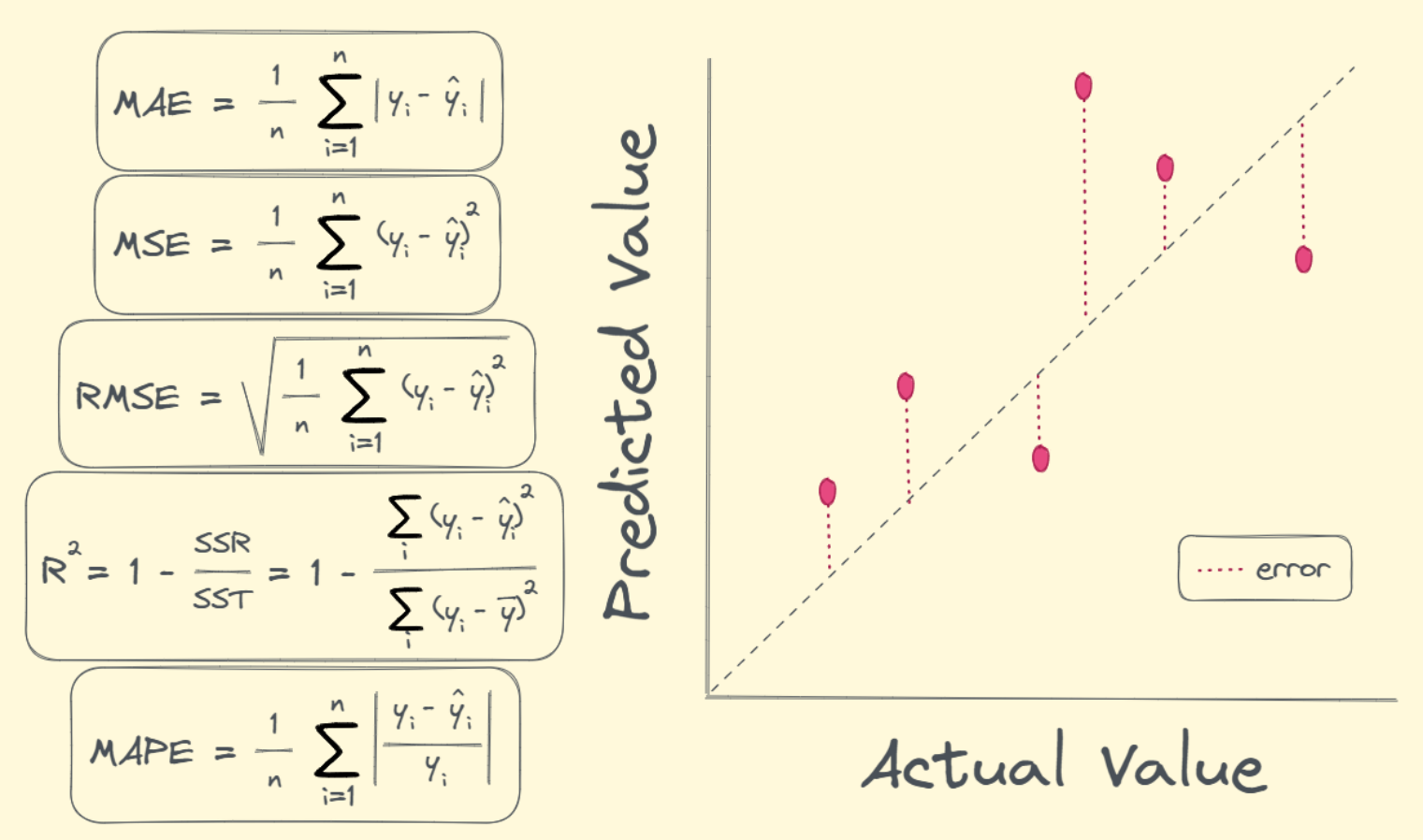


Figure: https://towardsdatascience.com/the-expressive-power-of-the-scatter-plot-c2f3354d3d97

L'optimisateur (Optimizer)

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.

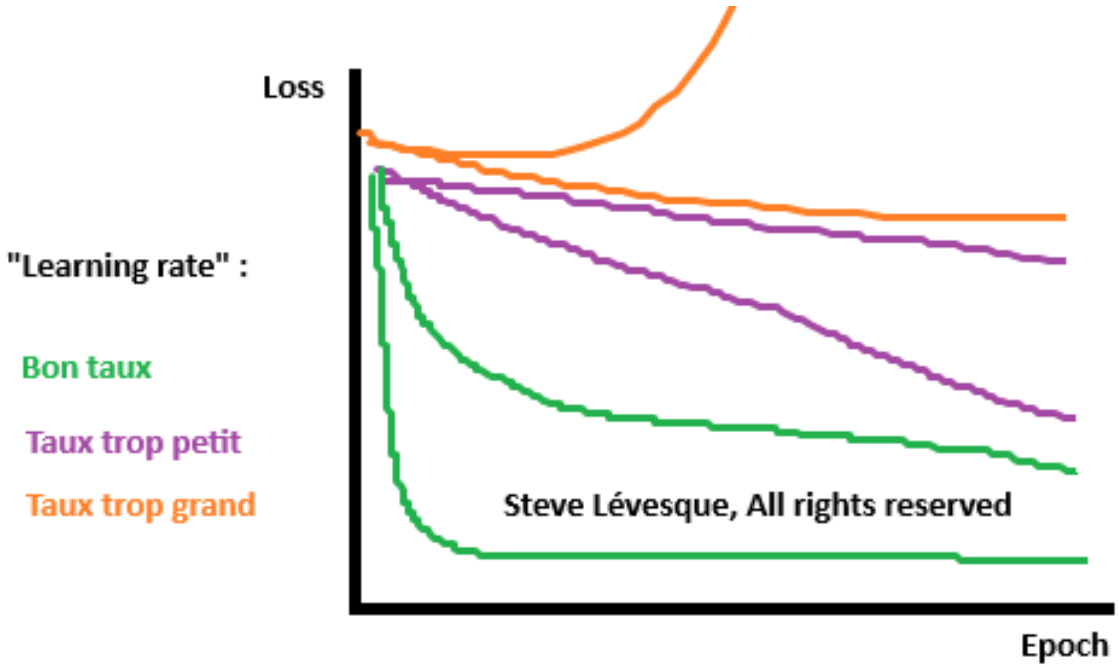
L'optimisateur (Optimizer)

Ceci est le problème de la déscente du gradient, veuillez retenir le principe, les maths ne sont pas sujet de ce cours d'AEC.

Arriver le plus proche possible du minimum aide à réduire la "loss". Le minimum global est souhaité, mais desfois il est difficile/impossible d'y arriver.

L'optimisateur (Optimizer)

Voici le résultat d'une bonne (vert) et mauvaise (orange et mauve) optimisation.



Métrique (Metric)

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

- Classification (Maximiser = meilleur) :
  - Précision (Accuracy)
  - F1
  - Etc.
- Régression (Minimiser = meilleur) :
  - MAE
  - MSE
  - Root Mean Squared Error (RMSE)
  - R2
  - Etc.

Métrique (Metric) - Précision (Accuracy)

Cette mesure crée deux variables locales, total et count, qui sont utilisées pour calculer la fréquence à laquelle y_pred correspond y_true. Cette fréquence est finalement renvoyée sous forme de précision binaire : une opération idempotente qui divise simplement le total par le nombre.

Tensorflow : `https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy`

Métrique (Metric) - Précision (Accuracy)

Exemple de prédiction de spam courriels (emails) : rouge = spam, gris = non-spam.

NB : On peut prédire de manière mauvaise le spam (en vrai est non-spam) ou non-spam (en vrai est spam).

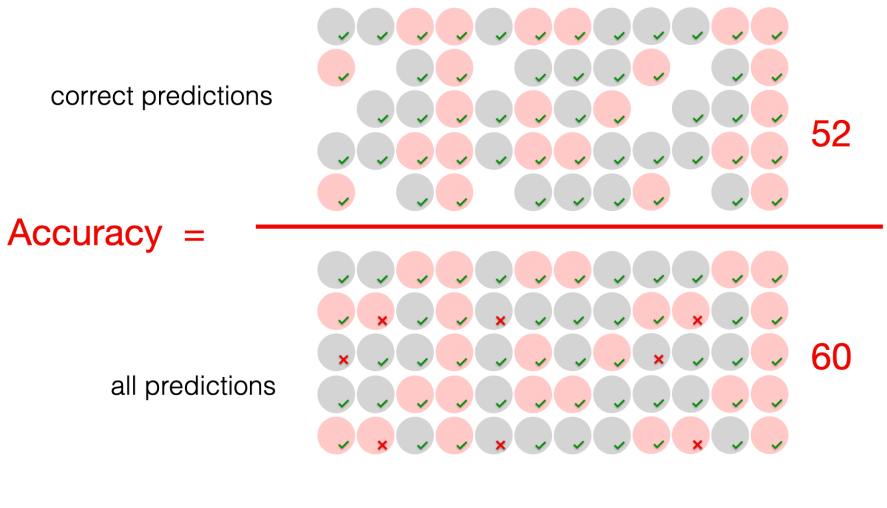


Figure: `https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall`

Régression (MAE, MSE, etc.)

Puisque la métrique en régression est de réduire l'erreur, il est possible d'utiliser les mêmes formules que la "loss" (MAE, MSE, etc.).

L'éfficacité dépends du problème. Si il est correct d'avoir des valeurs abbérentes (outliers), on peut utiliser MAE.

Sinon MSE est un meilleur choix puisqu'il va **punir** avec son amplification au **carré**, et ceci va être visible dans le résultat de la métrique et nous alarmer que finalement, notre entraînement contient trop de données abberantes.

NB : Amplifier les valeurs permet une meilleure détection des bons/mauvais (dépendamment du problème) comportement que nous voulons surveiller. Ceci est un paradigme réccurent pour le "pre-processing" des données.

Régression (MAE, MSE, etc.)

Par exemple, pour 3 = valeur réelle (y) et 2 = valeur prédite (y^*) :

MAE : $3 - 2 = 1$

MSE : $(3 - 2)^2 = 1^2 = 1$

Même réponse, MSE ne réagit pas trop lorsque la différence est minime. C'est ce qu'on veut en général.

Régression (MAE, MSE, etc.)

Par contre :

MAE : $100 - 2 = 98$

MSE : $(100 - 2)^2 = 98^2 = 9604$

Si on sait que les valeurs abbérantes n'ont pas lieu d'être dans notre modèle, il serait judicieux qu'on prenne MSE pour **fortement pénaliser les erreurs de grande envergure.**

Alors, pour un problème de ce type (i.e. Real Estate Prices) on devrait utiliser MSE plutôt que MAE puisqu'il est gentil lors de faibles erreurs, mais punitif pour les cas complètement faux.

Exercice - Régression (MAE, MSE, etc.)

Exemple 1 : Pour une maison qui peut se vendre à 500 000 $ (y) et une prédiction de notre modèle à 100 000 $ (y^*) :
MAE : $500000 - 100000 = 400000$
MSE : $(500000 - 100000)^2 = 400000^2 = 160000000000$

Exemple 2 : Pour une maison qui peut se vendre à 500 000 $ (y) et une prédiction de notre modèle à 499 000 $ (y^*) :
MAE : $500000 - 499000 = 1000$
MSE : $(500000 - 499000)^2 = 1000^2 = 1000000$

Question 1 : D'après vous, devrait-on utiliser MAE ou MSE ?

Question 2 : Si une perte de 1000$ est négligable sur un montant aussi gros (500k$), MAE ou MSE ?

Exercice - Régression (MAE, MSE, etc.)

Question 3 : Pourrait-on faire une normalisation des données pour réduire la disparité et réduire l'erreur des cas négligables ?

NB : 500 000 est la valeur la plus grande (1), et le reste est proportionnelement réduit jusqu'à 0.

Exemple 1 : Pour une maison qui peut se vendre à 500 000 $ (y) et une prédiction de notre modèle à 100 000 $ (y^*) :
MAE : $1 - 0.2 = 0.8$
MSE : $(1 - 0.2)^2 = 0.8^2 = 0.64$

Exemple 2 : Pour une maison qui peut se vendre à 500 000 $ (y) et une prédiction de notre modèle à 499 000 $ (y^*) :
MAE : $1 - 0.998 = 0.002$
MSE : $(1 - 0.002)^2 = 0.002^2 = 0.000004$

Exercice - Régression (MAE, MSE, etc.) - Conclusion

On peut maintenant utiliser MSE sans problème ! Notre exemple 1 est puni puisque 400k$ de perte est trop, mais 1k$ de perte est **négligable** vis-à-vis 500k$.

Bien que 0.8 (MAE) puni plus que 0.64 (MSE) puisqu'on doit **minimiser**, ils sont beaucoup plus proches entre eux que la borne inférieure : 0.000004 est beaucoup plus bas que 0.002.

Les 0.8 (MAE) et 0.64 (MSE) vont punir tout de même la mauvaise prédiction et la perte de 400k$, mais 0.000004 (MSE) ne va pas pénaliser la perte de 1000$ tandis que le 0.002 (MAE) aura un petit impact.

À cause que la normalisation est **proportionnelle**, ceci est une solution qui fonctionne pour une très grande majorité des problèmes.

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

```
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'autre 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

```
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 des nouvelles données inconnues au modèle.

En temps normal, les données d'entraînement (train set) sont connus 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

```
1  # Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
2
3  # Evaluate against the train set.
4  print("\n Evaluate the new model against the train set:")
5  my_model.evaluate(x=x_train_normalized, y=y_train, batch_size=batch_size)
6  # Evaluate against the test set.
7  print("\n Evaluate the new model against the test set:")
8  my_model.evaluate(x=x_test_normalized, y=y_test, batch_size=batch_size)
9
10 """
11  Evaluate the new model against the train set:
12 15/15 ------------------ 0s 2ms/step - accuracy: 0.9727 - loss: 0.0995
13
14  Evaluate the new model against the test set:
15 3/3 -------------------- 0s 3ms/step - accuracy: 0.9567 - loss: 0.1503
16 """
```

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 haut niveau administrative/gestion).

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

```python
# Copyright 2020 Google LLC. https://www.apache.org/licenses/LICENSE-2.0
def plot_curve(epochs, hist, list_of_metrics):
    """Plot a curve of one or more classification metrics vs. epoch."""
    # list_of_metrics should be one of the names shown in:
    # https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#defin
    plt.figure()
    plt.xlabel("Epoch")
    plt.ylabel("Value")

    for m in list_of_metrics:
        # Show accuracy as 0 to 100% for better reading
        if 'acc' in m:
            x = hist[m] * 100
        else:
            x = hist[m]
        plt.plot(epochs[1:], x[1:], label=m)

    plt.legend()
    plt.savefig(list_of_metrics[0] + '.png')
    plt.show()
```

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

NB : Ca diffère d'un problème à un autre

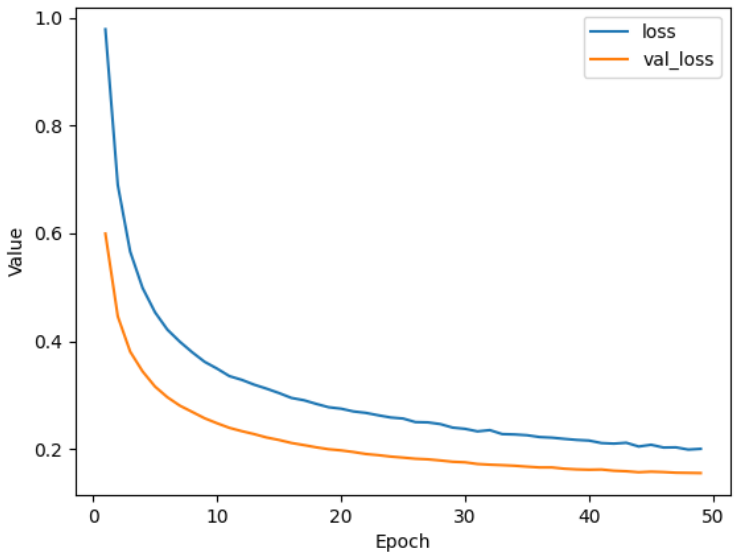


Figure: Entraînement - Perte (loss) - Le plus minimal, le meilleur

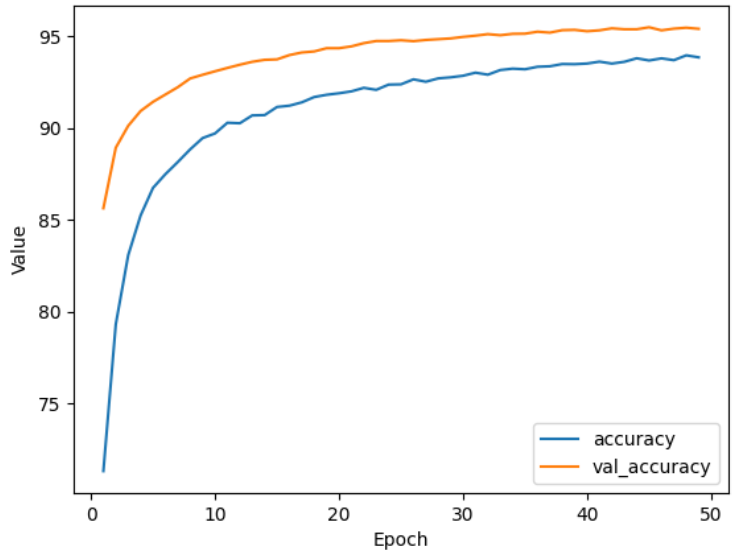


Figure: Évaluation - Précision en % (accuracy) - Le plus haut, le meilleur

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.

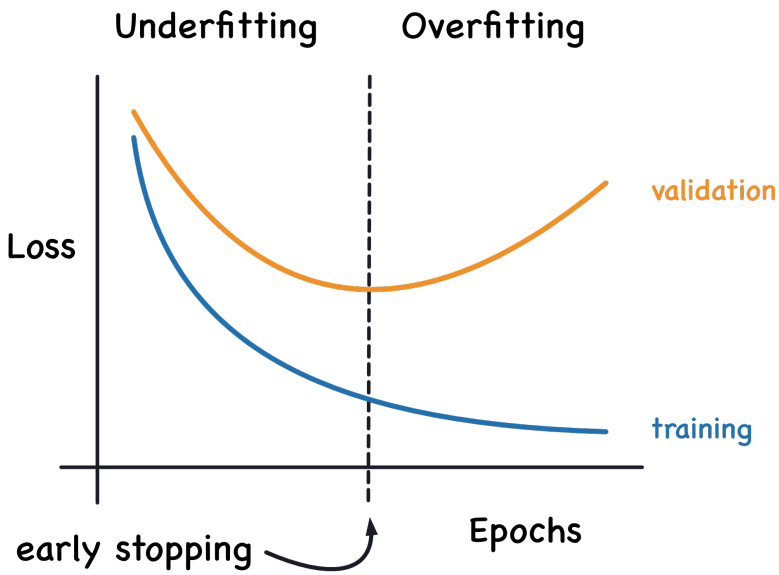


Figure: https://sourestdeeds.github.io/blog/overfitting-and-underfitting/

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.

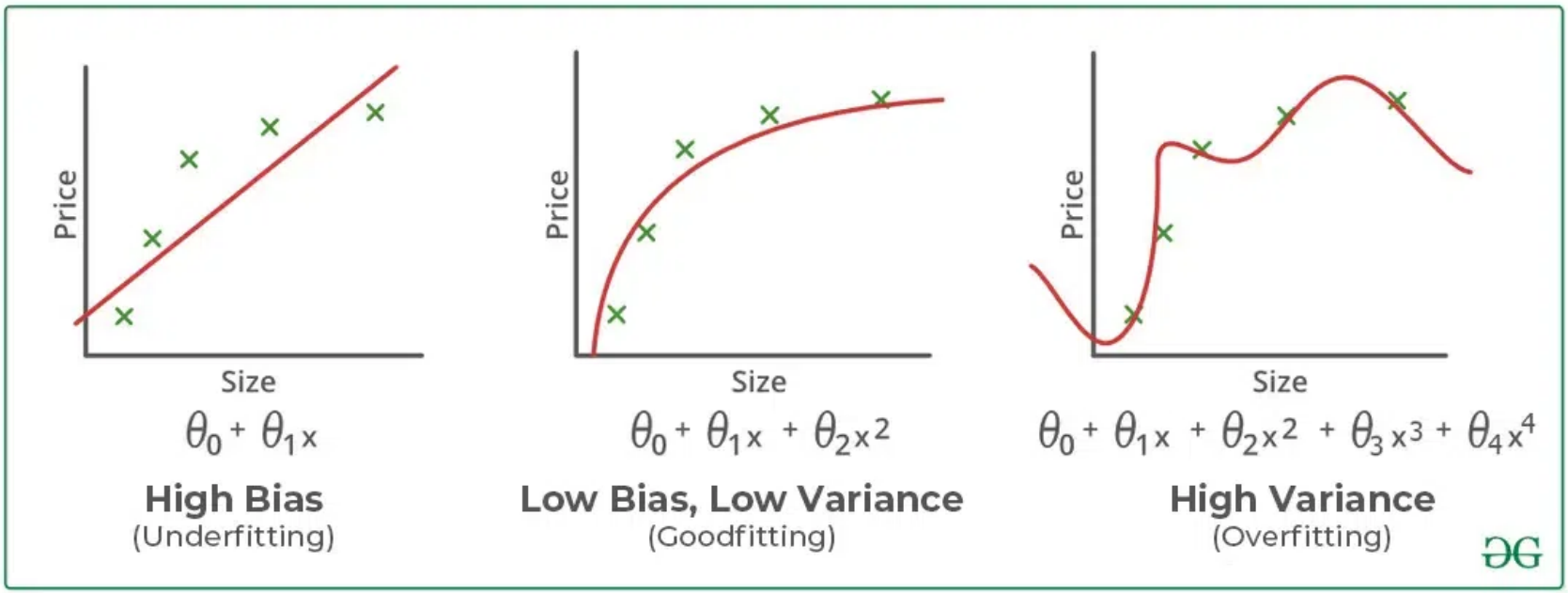


Figure: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Partie d'exécution globale

```
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')
```

Bibliographie

- https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning
- 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/
- https://medium.com/@LayanSA/complete-guide-to-adam-optimization-1e5f29532c3d
- https://www.datacamp.com/tutorial/tutorial-gradient-descent