Mini-Project (ML for Time Series) - MVA 2025/2026

Romain Planchon romain.planchon@gmail.com
Kilian Berlivet kilian.berlivet@gmail.com

December 13, 2025

1 Introduction and contributions

Avec l'essor de l'auto-supervision en vision et en NLP, des approches similaires ont émergé pour apprendre des représentations de séries temporelles. Le problème de fond reste le même : construire une représentation compacte mais exploitable pour les tâches usuelles (classification, clustering, etc.). Dans ce cadre, la plupart des méthodes apprennent surtout des représentations *instance-level*, i.e. un vecteur par série (ou par fenêtre).

Ces représentations globales marchent bien pour classifier une série entière, mais elles deviennent vite limitées dès qu'il faut garder une résolution temporelle : prédiction, détection d'anomalies, ou robustesse aux *missing values*.TS2Vec s'attaque précisément à ce point : au lieu d'apprendre seulement une représentation globale, il apprend des représentations *timestamp-level* multi-échelles, cohérentes entre vues de contexte, et utilisables ensuite avec des têtes simples selon la tâche.

Nos contributions sont les suivantes : - (i) Nous réimplémentons le framework TS2VEC (85% toutes les features non utiles pour nos expériences ne sont pas implémentés). - (ii) Nous validons expérimentalement notre implémentation sur le jeu de données RefrigerationDevices en comparant aux performances rapportées dans l'article original. - (iii) Nous nous focalisions ensuite sur la robustesse des representations aux missing values. Les expériences ont été définies à deux, puis implémentées par l'un de nous avant une analyse commune, avec Romain en charge de l'implémentation de l'architecture et du prétraitement, et Kilian de celle de l'expérience sur les valeurs manquantes.

2 Méthode

Nous considérons un ensemble de séries temporelles $(x^{(i)})_{i=1}^{N}$, où chaque $x^{(i)} \in \mathbb{R}^{T_i \times F}$ est de longueur T_i avec F canaux. On apprend un encodeur

$$f_\theta : \mathbb{R}^{T \times F} \to \mathbb{R}^{T \times K}, \qquad f_\theta(x^{(i)}) = r^{(i)} = (r_t^{(i)})_{t=1,\dots,T_i},$$

tel que chaque $r_t^{(i)} \in \mathbb{R}^K$ soit une représentation *timestamp-level* de l'instant t, et qu'une agrégation temporelle simple $z^{(i)} = \max_t r_t^{(i)} \in \mathbb{R}^K$ fournisse une représentation *instance-level* de la série entière.

L'encodeur f_θ est un réseau convolutionnel 1D le long de l'axe temporel, avec des convolutions de plus en plus dilatées : les premières couches capturent des motifs très locaux, les couches dilatées agrègent un contexte de plus en plus large autour de chaque timestamp.

Les auteurs introduisent aussi le **timestamp masking** dans le but d'apprendre des représentations plus robustes à une observation partielles. En pratique, chaque x_t est d'abord projeté (pointwise) dans l'espace latent de dimension K pour obtenir z_t, puis un masque binaire sur l'axe temporel est appliqué directement sur ces vecteurs latents. L'utilité et l'efficacité de ce mécanisme seront analysées dans la section 4.

2.1 Entraînement de l'encodeur discriminatif

On rappelle que l'idée des auteurs de TS2Vec est de définir une tâche contrastive qui force, simultanément : (i) la stabilité des représentations d'un même « événement temporel » observé sous des contextes légèrement différents, (ii) la séparation des timestamps au sein d'une même série et (iii) la discrimination entre séries différentes au même instant.

Pour matérialiser ces trois contraintes sans supervision explicite, ils construisent, pour chaque série, deux *vues de contexte recadrées* : c'est le recadrage qui fournit à la fois des paires « même événement, contexte différent » pour (i), des timestamps distincts dans une même série pour (ii), et, via le batch, des séries différentes au même temps pour (iii).

Vues de contexte recadrées. Pour une série $x^{(i)}$ de longueur T_i, on échantillonne à chaque itération deux intervalles

$$[a_1, b_1],\ [a_2, b_2] \subset \{1, \dots, T_i\}, \qquad 0 < a_1 \leq a_2 \leq b_1 \leq b_2 \leq T_i,$$

qui définissent deux sous-séries $x^{(i,1)} = x^{(i)}_{a_1:b_1}, x^{(i,2)} = x^{(i)}_{a_2:b_2}$

et une zone de recouvrement $\Omega_i = \{t : a_2 \leq t \leq b_1\}$.

Les deux vues correspondent donc au même signal observé sur des fenêtres décalées, partageant un sous-intervalle Ω_i où les timestamps représentent le « même » contenu sémantique, modulo le contexte.

On passe ensuite ces deux vues dans l'encodeur :

$$r^{(1)}_{i,t} = f_\theta(x^{(i,1)})_t, \qquad r^{(2)}_{i,t} = f_\theta(x^{(i,2)})_t, \qquad t \in \Omega_i.$$

On note pour alléger : $r_{i,t} = r^{(1)}_{i,t}, \qquad r'_{i,t} = r^{(2)}_{i,t}, \qquad t \in \Omega_i.$

Les auteurs construisent ensuite une loss contrastive qui reflète exactement les contraintes précédentes :

- **Cohérence contextuelle (positifs).** Pour un i et un $t \in \Omega_i$, les deux vecteurs $r_{i,t}$ et $r'_{i,t}$ représentent le même timestamp de la même série, vu dans deux contextes temporels légèrement différents. Ils sont donc considérés comme une *paire positive* qui doit être rapprochée dans l'espace latent. Cela impose que f_θ encode le contenu local de la série plutôt que la position exacte dans la fenêtre.
- **Séparation dans le temps (négatifs temporels).** Pour un i fixé, deux timestamps $t \neq t'$ correspondent en général à des états du signal différents (même sur un sinus périodique, on ne veut pas que toutes les phases soient confondues). Les paires $(r_{i,t}, r_{i,t'})$ et $(r_{i,t}, r'_{i,t'})$ avec $t' \neq t$ sont donc utilisées comme *négatifs temporels*. Cette composante de la loss décourage le collapse où toutes les positions d'une même série partageraient la même représentation.

- **Séparation entre séries (négatifs instance-level).** À timestamp fixé t, deux séries $i \neq j$ peuvent se trouver dans des régimes ou des classes différentes. Les paires $(r_{i,t}, r_{j,t})$ et $(r_{i,t}, r'_{j,t})$ sont donc des *négatifs instance-level*. Cette partie de la loss force f_θ à encoder des informations discriminantes entre séries, ce qui est crucial pour obtenir de bons embeddings globaux $z^{(i)}$ pour la classification.

On obtient ainsi une tâche discriminative auto-supervisée : à chaque timestamp (i, t), l'encodeur doit « reconnaître » la bonne vue positive au milieu d'un ensemble de négatifs construits dans le temps (même série) et dans le batch (autres séries).

Ainsi la **perte temporelle** (i, t) et la **perte instance-level** (i, t) s'écrivent :

$$\ell_{\text{temp}}^{(i,t)} = -\log \frac{\exp(r_{i,t}^\top r'_{i,t})}{\sum_{t' \in \Omega_i} \left[\exp(r_{i,t}^\top r'_{i,t'}) + \mathbb{1}_{t' \neq t} \exp(r_{i,t}^\top r_{i,t'}) \right]}$$

$$\ell_{\text{inst}}^{(i,t)} = -\log \frac{\exp(r_{i,t}^\top r'_{i,t})}{\sum_{j=1}^{B} \left[\exp(r_{i,t}^\top r'_{j,t}) + \mathbb{1}_{j \neq i} \exp(r_{i,t}^\top r_{j,t}) \right]}.$$

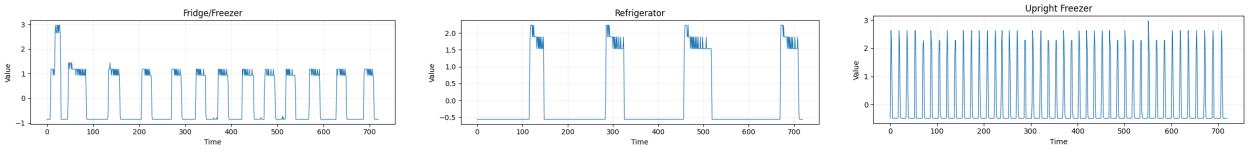
La perte contrastive totale au niveau des timestamps est enfin

$$L_{\text{dual}} = \frac{1}{\sum_i |\Omega_i|} \sum_{i=1}^{B} \sum_{t \in \Omega_i} \left(\ell_{\text{temp}}^{(i,t)} + \ell_{\text{inst}}^{(i,t)} \right).$$

En pratique, on réapplique cette construction à des représentations obtenues par max-pooling successifs le long du temps (contraste hiérarchique multi-échelle), de manière à contraindre f_θ à rester discriminatif à la fois au niveau local (timestamp) et au niveau global (instance).

3 Data

Nous utilisons le jeu de données *RefrigerationDevices* qui contient des séries temporelles univariées mesurant la consommation électrique individuelle de trois types d'appareils frigorifiques: *Fridge/Freezer*, *Refrigerator* et *Upright Freezer*. Chaque série correspond à une journée d'observation échantillonnée toutes les deux minutes. Les séries sont déjà *standardisées indépendamment les unes des autres*. Quelques exemples (Fig. 1).



(a) *Fridge/Freezer* (b) *Refrigerator* (c) *Upright Freezer*

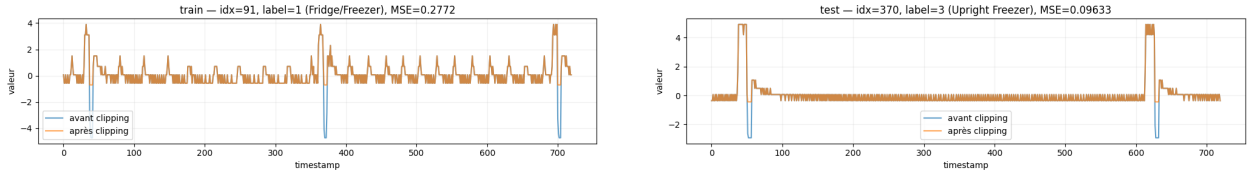
Figure 1: Exemples de séries normalisées pour chaque classe du dataset *RefrigerationDevices*.

Pour les trois types d'appareils, on s'attend à observer des alternances de phases *ON* (compresseur actif) et *OFF* (consommation de veille). Cependant, chaque constructeur garde des degrés de

liberté pour réguler la température (compresseur plus ou moins puissant, cycles plus ou moins longs ou variation de la période d'activation). On observe ainsi une forte variabilité intra-classe des signaux temporels. (Fig. 4)

On s'attend tout de même à des différences : physiquement, un *Refrigerator* devrait présenter des cycles plus modestes (moins fréquents ou moins intenses) qu'un *Freezer*. Un *Fridge/Freezer* lui, combine un compartiment réfrigérateur et un compartiment congélateur, on peut donc observer jusqu'à trois niveaux d'activation distincts dans le signal brut : OFF, ON « faible » (frigo seul) et ON « fort » (freezer ou les deux compartiments), comme on le remarque sur la Fig. 1a.

Bien que le dataset provienne de mesures réelles, il reste selon nous globalement très propre : nous n'avons pas observé de bruit excessif ni de segments manifestement dus à des données manquantes. Cependant comme nous sommes face à une dynamique essentiellement ON/OFF, on est tenté de considérer que des valeurs très en dessous du plateau bas (OFF) sont « anormales ». Quelques signaux présentent en effet des dents fortement négatives, isolées, bien en dessous de ce qu'on identifie visuellement comme l'état OFF. Nous avons donc testé une détection d'outliers négatifs (Fig. 2), basée sur l'estimation d'un niveau OFF typique et le *clipping* des valeurs situées en dessous. Il n'est cependant pas clair si ces points sont de vrais artefacts de mesure ou simplement le reflet d'appareils dont le compresseur est très rarement à l'arrêt.



(a) Exemple sur le train.

(b) Exemple sur le test.

Figure 2: Exemples clipping des valeurs fortement négatives (avant/après) sur le train et le test.

Enfin, l'inspection qualitative a révélé quelques séries du split de train dont la forme globale n'a, à nos yeux, physiquement aucun sens pour ce type d'appareils. Nous avons choisi de retirer ce petit nombre de séries aberrantes du jeu d'entraînement, afin de ne pas polluer l'apprentissage.

4 Results

Nous évaluons notre implémentation de TS2Vec sur le dataset *RefrigerationDevices* décrit dans la section Data. Nous suivons la configuration de l'implémentation officielle pour avoir des résultats comparables. Dans nos expériences nous faisons varier la probabilité de masking $p_{\text{mask}} \in \{0.0, 0.1, 0.2, \ldots, 0.8\}$ qui restait fixé à 0.0 ou 0.5 dans l'article (ON ou OFF). Pour chaque valeur de p_{mask}, nous entraînons un modèle distinct, avec deux variantes de données d'entrée : *brutes* et *prétraitées*. (cf chapitre 3)

Pour l'évaluation, nous suivons exactement le protocole du papier : une fois l'encodeur entraîné, nous gelons le backbone et extrayons une représentation *instance-level* par max-pooling global sur la dimension temporelle. Sur ces vecteurs, nous entraînons un classifieur SVM à noyau RBF, avec recherche en grille sur le paramètre de régularisation C par validation croisée sur le train.

Dans l'article original, TS2Vec atteint une accuracy test de 0.589 sur *RefrigerationDevices* avec la configuration standard, c'est-à-dire avec $p_{\text{mask}} = 0.5$. À architecture et protocole d'évaluation

identiques, nos expériences produisent des scores du même ordre de grandeur. Le Tableau 1 synthétise l'accuracy test en fonction de p_{mask}, pour les données brutes et pour la variante prétraitée.

p_{mask}	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Test acc. (brut)	0.600	0.580	0.580	0.570	0.580	0.550	0.560	0.520	0.500
Test acc. (prétraité)	0.610	0.621	0.624	0.597	0.595	0.592	0.540	0.528	0.520

Table 1: Accuracy classification test set en fonction de p_{mask} pour notre implémentation TS2Vec

Le résultat le plus notable est donc que la configuration *sans* timestamp masking ($p_{\text{mask}} = 0$) est déjà compétitive et légèrement meilleure que la configuration standard du paper, et un masquage modéré, combiné à un léger nettoyage des données, permet d'obtenir la meilleure accuracy globale (0.624 pour $p_{\text{mask}} = 0.2$). À l'inverse, un masking trop agressif finit par effacer trop d'information sur la structure et détériore systématiquement la classification.

Le tableau met également en évidence **l'impact du preprocessing**. Les différences sont marginales mais globalement, ces résultats suggèrent que :

- les représentations apprises par TS2Vec sont raisonnablement robustes aux défauts de données présents dans *RefrigerationDevices* : même sans preprocessing, les accuracies restent proches de celles du papier ;
- un preprocessing léger, motivé par une analyse qualitative des séries peut néanmoins apporter un gain de performances.

On rappel ici que le preprocessing effctué reste très faible et ne permet pas de conclure sur si cette méthode permet de s'affranchir d'un prétraitement d'une manière plus générale.

4.1 Robustesse au missing Value

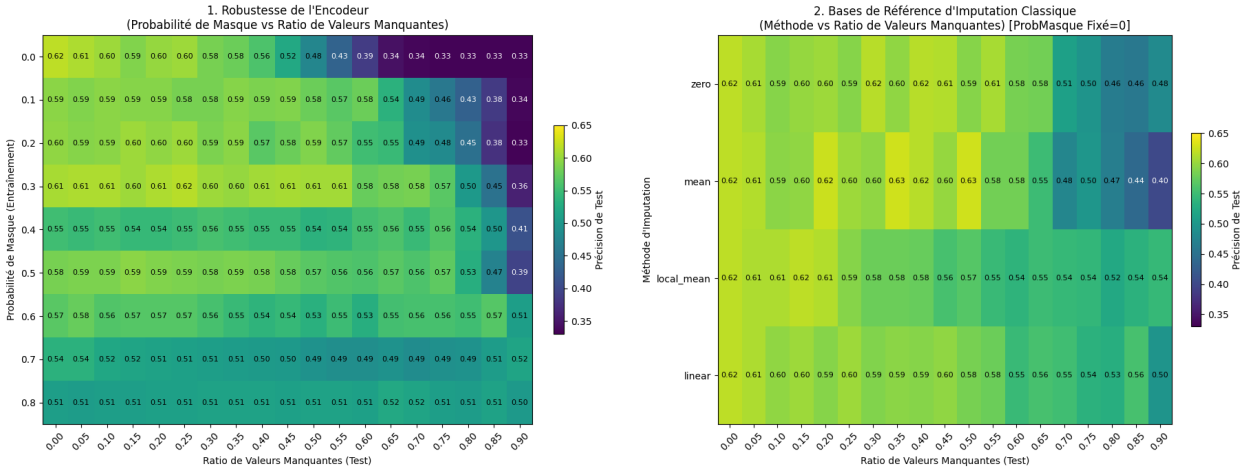
Avec l'expérience précédente, l'utilité du timestamp masking en termes de *gain brut* de performance n'est pas évidente. Les auteurs eux motivent aussi le timestamp masking comme un moyen de rendre l'encodeur plus robuste aux **missing values**. Dans l'article, ils montrent que si l'on entraîne *sans* timestamp masking, la performance s'effondre très vite dès que le pourcentage de valeurs manquantes au test augmente. Nous reproduisons cette expérience en faisant varier p_{mask} (pas seulement 0 et 0.5) et retrouvons le même comportement (Fig. 3(a)).

On observe bien que sans timestamp masking ($p_{\text{mask}} = 0$), les performances décroissent rapidement dès que p_{miss} augmente. À l'inverse, plus p_{mask} diminue plus la décroissance est faible : le modèle est forcé pendant l'entraînement à construire des représentations qui restent cohérentes même quand une partie du contexte temporel disparaît.

Nous avons donc voulu voir comment cela se compare à des méthodes classiques d'imputation : pour un modèle entraîné sans timestamp masking, nous appliquons des imputations simples directement sur le test set avant encodage (Fig. 3(b)). On observe que ces imputations sont, dans nos expériences, plus robustes que le timestamp masking seul. Au final, revenir à des baselines d'imputation reste plus performants.

Cela dit, on peut tout de même défendre leur approche : elle ne suppose *aucun* schéma d'interpolation ni hypothèse forte sur la dynamique entre deux échantillons. On indique simplement cette valeur est manquante via un masque, et l'encodeur apprend à produire une représentation exploitable malgré ces absences.

A Annexe



(a) Variation de p_{mask}.

(b) Comparaison avec imputation.

Figure 3: Missing data experiments.

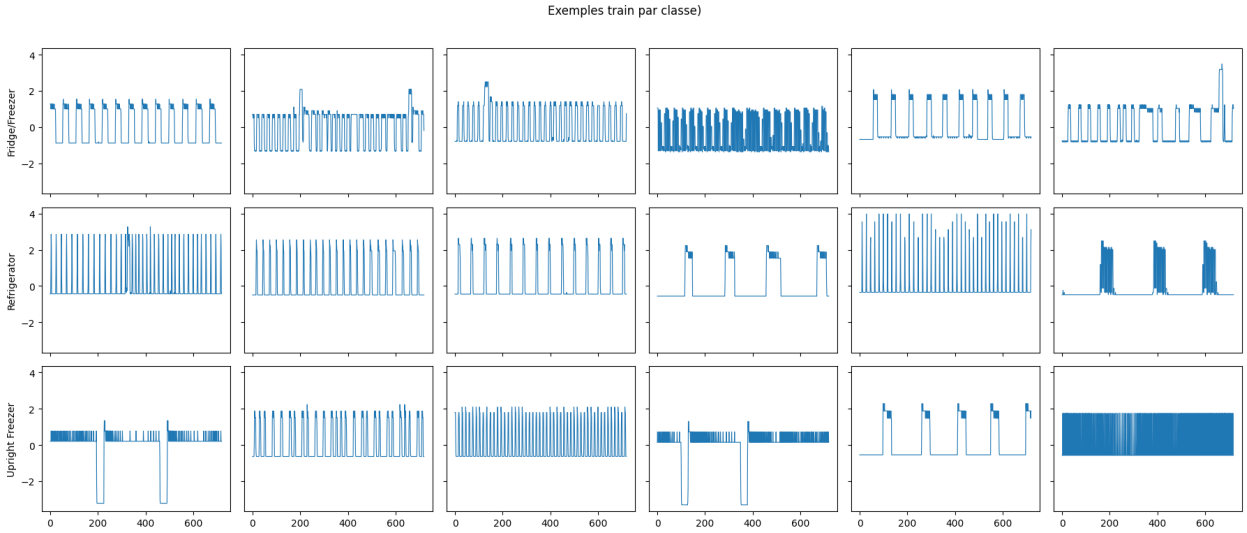


Figure 4: Variabilité intra-classe des signaux temporels.