Project report: Unsupervised feature alignment with SOFT DTW

Alexia Allal & Anaïs Khemliche

The project focuses on unsupervised feature learning from time-series data using a Siamese network trained with a differentiable version of Dynamic Time Warping (DTW) as a loss function. The objective is to align feature representations for similar activities while maintaining separability for dissimilar ones.

Here is a detail of the methodology used to solve the task, some examples of encountered difficulties and ideas of potential limitations and extensions to which the DTW-based Siamese network method can be applied.

I. Methodology

As presented in the notebook, the main steps we carried out to solve the task are as follows:

1. Dataset handling

a) Data preparation

The UCI HAR dataset contains time-series sensor data from smartphones for various human activities. The main characteristics of the dataset are as follows:

- Training set shape: (7352, 128, 9)
- Test set shape: (2947, 128, 9)
- Number of classes = 6

N.B. Because the data seemed to be already organised in classes before, we did a shuffling after the dataset load to ensure the examples (data, labels) were balanced in each of the training and testing sets.

b) Define a custom dataset class for Siamese network

We build a class custom dataset class to dynamically generate pairs of sequences during training, representing positive (similar) and negative (dissimilar) examples. Then we apply it to create train and test datasets using the previous training/testing data and labels from the UCI HAR dataset preprocessing in step 1.

2. Dynamic Time Warping (DTW) implementation

As mentioned in the notebook presentation, DTW being a technique to measure the similarity between time-series data by aligning them temporally, we can use it as a loss function on the Siamese network. However, the problem is that the traditional DTW is non-differentiable, making it unsuitable as a loss function. This is why we use Soft DTW, a differentiable approximation of DTW, enabling its integration into the backpropagation-based training of the network.

We use the pysdtw library to implement a pairwise distance function that computes the DTW distance between a pair of sequences. Then, we implement a differentiable Soft DTW function to calculate this distance.

3. Siamese Network implementation

The Siamese network is designed to process two inputs (time-series sequences) through two identical branches, generating feature embeddings for each input. The two time-series are thus projected into a learned feature space. The goal is to compare these embeddings using the Soft DTW function as a loss function. The underlying idea is that the distance between them reflects their similarity.

a) Network definition

Key Components of the SiameseNetwork Class

- Input dimension: The network starts with inputs shaped (128 x 6), representing the length and number of features of the time-series data. The goal is to reduce this to a compact representation of hidden_size=128.
- Layers
  - first_layers: used to process the raw input features using fully connected lnear layers interspersed with the Relu activation functions. It reduces the data into a preliminary embedding of size (128x128).
  - last_layers: used to further compress the concatenated embeddings (flattened from (128x128) to (1x16384)) into a final embedding of size 128.
- **Forward method**
  - The forward takes two inputs (input1 and input2) and passes each through forward_one.
  - Branch logic (forward one)
    - A single branch processes one input sequence through the network's layers.
    - After first_layers, the sequence is reshaped (flattened) and last_layers finalizes the embedding.
  - The forward outputs two embeddings (output1, output2) of size 128 for the two inputs, which are then compared using a loss function.

b) Network training strategy

The objective is to train the Siamese network so that positive pairs of time-series are encouraged to have smaller distances in the feature space, while negative pairs are encouraged to have larger distances. To do this, the Soft DTW loss is used to compute the alignment distance between outputs of the Siamese network.

In the code, the train_network function encapsulates the training process of the network, with flexibility for using either a standard contrastive loss or a DTW-based loss. We used the following training logic:

i. Preparation
- The network is set to training mode using model.train().
- An optimizer (e.g., Adam or SGD) is used to update the network parameters.
- Loss is accumulated across mini-batches to monitor progress.

ii. **Batch processing**. For each batch:
- The network computes embeddings for the input pairs.
- If use_dtw is True, the custom ContrastiveLossDtw loss is applied, otherwise the standard contrastive loss is used.
- Gradients are computed using loss.backward(), and the optimizer updates the parameters.

iii. **Performance evaluation**
- After each epoch, the total loss is printed for progress tracking.
- The training time is recorded and stored in the network for reference.

c) Loss functions

Two loss functions are defined to train the Siamese network:

« Standard » contrastive coss function

- The function encourages embeddings of similar pairs (label=0) to be close and dissimilar pairs (label=1) to be far apart.
- Logic
  - For positive pairs (label = 0) → penalize the squared Euclidean distance.
  - For negative pairs (label=1) → penalize the difference between the margin and the Euclidean distance, ensuring the embeddings are sufficiently far apart.
- Goal: the goal is to minimize the intra-class variance while maximizing inter-class separability.

DTW-based contrastive Loss

- Incorporates DTW to compute the ground truth distance between input sequences.
- Logic: the loss penalizes the squared difference between the Euclidean distance of the embeddings and the DTW distance of the inputs.
- Goal: to align the learned feature space with the DTW-aligned distances of the original inputs, ensuring that similar sequences are represented closer in the feature space.

c) Dataset handling / Data Loaders

- DataLoader configuration
    - The train_loader and test_loader are created to iterate through the dataset.
    - Batch size: set to 64 to balance memory usage and computational efficiency.
    - Shuffling: enabled for training data to improve generalization and avoid biases during training.
- Pair-based dataset: the Siamese network requires positive and negative pairs for training where:
    - Positive pairs are sequences from the same class/activity.
    - Negative pairs are sequences from different classes/activities.

This pairing logic is managed outside the network, ensuring balanced representation of positive and negative samples in each mini-batch.

4. PART 4 & 5 : FEATURES EVALUATION WITH KNN + FINE-TUNING WITH LINEAR LAYER

We implemented a KNN classifier on the learned representations to evaluate the classification performance of the Siamese network and we added a simple linear layer for fine-tuning for better performance evaluation.

Finally, in part 6, we compare the performance of our Siamese network with a classic neural network.

II. ENCOUNTERED DIFFICULTIES

We did encounter some difficulties to solve the task:

- Understanding of the dataset and the nature of the data

The dataset included time-series data, some of which were pre-processed with Fourier transformations, resulting in mismatched dimensions and a loss of temporal information. This required careful understanding and adjustment to align the data properly with the model's expectations. We used the code put on Moodle to handle this issue.

- Definition of the loss function

It took some time for us to define a propre loss function to train the network. We used a kind of contrastive loss and adapted its structure to the task. We also used the hint put on Moodle in this step.

- Understanding of the network architecture

Here, the Siamese network design must return feature representations only, while discarding the temporal dimension of the time-series, it was something we didn’t grasp in the beginning.

But overall, after the Siamese network was implemented, everything else went smoothly : kNN classifier training + Multi-layer perceptron training on the features returned by the Siamese network.

III. LIMITATIONS

We can also mention some limitations to this approach, linked with some specificities of the method that make it less suitable to other tasks.

- Scalability to other datasets: this approach assumes time-series data with clear temporal alignment, making it less suitable for highly noisy or irregularly sampled data.
- Dependence on pair sampling: the quality of the learned representation heavily depends on the quality and diversity of the sampled pairs.
- Loss function sensitivity: the Soft DTW loss relies on hyperparameters (e.g., the smoothing factor), which may require extensive tuning for different datasets.
- Computational costs: we can imagine that the Soft DTW computation can be computationally expensive, particularly for long sequences or large datasets.

IV. Potential improvements and extensions to other tasks

We can think of some potential improvements that could be added to the model to improve the overall performance.

- Multi-task learning: combine the Siamese network with auxiliary tasks, such as predicting activity labels, to improve the learned feature space's generalization.
- Clustering-based pre-training: we could use clustering methods to create pseudo-labels for entirely unsupervised pre-training, followed by fine-tuning on a downstream task.
- Attention mechanisms: integrate attention layers to focus on salient features of the time-series, improving the quality of the learned representations.

Finally, regarding the range of potential applications of this approach, we could also try to extend the framework to align time-series data from different domains (e.g., aligning sensor data from different devices or users).