Learning Sequence Neighbourhood Metrics

Justin Bayer, `bayer.justin@googlemail.com` [*1], Christian Osendorfer,
`osendorf@in.tum.de`[1], and Patrick van der Smagt, `smagt@dlr.de`[2]

[1] Chair for Robotics and Embedded Systems, Insitut für Informatik, Technische
Universität München
[2] Institute of Robotics and Mechatronics, DLR German Aerospace Center

Abstract. Recurrent neural networks (RNNs) in combination with a
pooling operator and the neighbourhood components analysis (NCA)
objective function are able to detect the characterizing dynamics of se-
quences and embed them into a fixed-length vector space of arbitrary
dimensionality. Subsequently, the resulting features are meaningful and
can be used for visualization or nearest neighbour classification in linear
time. This kind of metric learning for sequential data enables the use of
algorithms tailored towards fixed length vector spaces such as $\mathbb{R}^n$.

1. Introduction

Sequential data is found in many domains including medical applications, robot
control, neuroscience, financial information or text processing. This data is fun-
damentally different from static data vectors.

When considering a single sequence over T time steps $\mathbf{x} = (x_1, x_2, \ldots, x_T) \in X^*$ with $X \subset \mathbb{R}^n$, the order of the individual elements x_i is relevant for the
interpretation[3]. Conversely, in the case of static data $\mathbf{x}' \in \mathbb{R}^n$, an ordering on
the n components is not even defined. Indeed, the key element of structured data
is that the context (i.e., the immediate predecessors and successors) contains
essential information to make learning on the data possible.

[5] note that metrics and features are actually closely related: by measuring
pairwise distances between the data points $\mathbf{x}' \in \mathbb{R}^n$, the data can be embed-
ded into a metric space. They learn a Mahalanobis distance by mapping the
high-dimensional data set X to a metric space Z in which k-nearest neighbour
classification performance is maximized. The resulting objective function is dif-
ferentiable with respect to the embedding.

Similar to [16], we use a different model than a linear map for learning the
embedding function. Our choice, recurrent neural networks (RNNs), are rich
models for sequence learning. They have been successfully used for handwriting
recognition [7], audio processing [6], and text modelling [14].

---

[*] Contacting author.
[3] We let X^* denote the set of all sequences over the space X.

Related Work

Only a few principled approaches exist for extracting fixed length features from sequential data. If we were given some kind of distance measure, a classic technique such as multi-dimensional scaling could be used. This is however rarely the case. A commonly used practice is based on a set of fixed basis functions (e.g. Fourier or wavelet basis). While it has strong mathematical guarantees, it is sometimes too inflexible: in order to work with arbitrarily long sequences, a sliding time window has to be employed, limiting the capability to model context. Furthermore, the fixed set of basis functions implies that the problem of identifying usefull factors of variations remains unresolved in general. Fisher kernels [10], a combination of probabilistic generative models with kernel methods, provide another commonly vectorial representation of sequences. The basic idea is that two similar objects induce similar gradients of the likelihood for the parameters of the model. Thus, the features for a sequence are the elements of the gradient of the log-likelihood of this sequence with respect to the model paramters. This choice can presumably be very bad: if the distribtution represented by the trained model closely resembles the data distribution the gradients for all sequences in the data set will be nearly zero. A recent paper [17] alleviates this problem by exploiting label information and employing ideas from metric learninig. Obviously, this only works if class information is available.

A fully unsupervised approach is to use the parameters estimated by a system identification method (e.g., a linear dynamical system) as features. Recent work includes [12], in which a complex numbers based system successfully clusters motion capture data.

The last two approaches clearly suffer from the fact that the number of features is directly connected with the complexity of the model. In particular it is not given that the important factors of variation are captured by these methods.

In principle, any sequenctial clustering technique can be used as a feature extractor by treating the scores (e.g. the posterior likelihood in case of a generative model or the distances to a node of a self-organizing map) as features.

2. Recurrent Neural Networks

Recurrent neural networks are an extension of feedforward networks to deterministic state space models. The inputs to an RNN are given as a sequence $(x_1, x_2, \ldots, x_T)$. Subsequently, a sequence of hidden states $(h_1, h_2, \ldots, h_T)$ and a sequence of outputs $(o_1, o_2, \ldots, o_T)$ is calculated via the following equations:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$
$$o_t = W_{ho}h_t + b_o \tag{2}$$

where $t = 1, 2, \ldots, T$ and σ is a suitable transfer function, typically the tangent hyperbolic, applied element-wise. $W_\diamond$ are weight matrices, $b_\diamond$ bias vectors and x_i, h_i and o_i real-valued vectors. For the calculation of h_1 a special initial hidden state h_0 has to be used which can be optimized during learning as well.

RNNs have a lot of expressive power since their states are distributed and nonlinear dynamics can be modelled. The calculation of their gradients is astonishingly easy via Backpropagation Through Time (BPTT) [15] or Real-Time Recurrent Learning [18]. However, first order gradient methods completely fail to capture relations that are more than as little as ten time steps apart of each other. This problem is called the *vanishing gradient* and has been studied by [8] and [1]. The previous state of the art method to overcome this has been the Long Short-Term Memory (LSTM) [9] up until recently, when [13] introduced a second-order optimization method for RNNs, a Hessian free optimizer (HF-RNN), which is able to cope with aforementioned long term dependencies even better. In this work, we stick to LSTM since the HF-RNN is tailored towards convex loss functions— neighbourhood component analysis (NCA), the objective function of choice in this paper, is however not convex.

Another neural model for nonlinear dynamical systems is the echo state network approach introduced in [11]. The drawback of this method is that the dynamics that are to be modelled have to be already present in the network's random initialization.

Recurrent Networks are Differentiable Sequence Approximators

One consequence of the differentiability of RNNs is that we can optimize their parameters with respect to an objective function.[4] Stochastic gradient descent or higher order techniques are the techniques of choice to fit the weights.

Similar to [3] we reduce output sequences to a single vector with a *pooling operation*. A pooling operation is a function $p : X^* \to X$ that reduces an undefined amout of inputs to a single output of the same set, e.g. taking the sum or picking the maximum. Similar to convolutional neural networks, we can use this technique to reduce a sequence to a point. If our pooling operation is differentiable as well, we can use it as a gateway to arbitrary objective functions that are defined on real vectors. Given a network f parametrized by W, a data set $\mathcal{D} = \{x_i\}$, a pooling operation p and an objective function $\mathcal{O}$ we proceed as follows:

1. Process input sequences $\mathbf{x}_i = (x_{i1}, \ldots, x_{iT}), x_{it} \in \mathbb{R}^n$ to produce output sequences $f(\mathbf{x}_i; W) = \mathbf{o}_i = (o_{i1}, \ldots, o_{iT}), o_{it} \in \mathbb{R}^m$,
2. Use a pooling operation p to reduce the output sequences to a point via $p(o_{i1}, \ldots, o_{iT}) = e_i$,

---

[4] The authors recommend to use automatic or symbolic differentiation. In this work, Theano [2] was used.

3. Calculate the objective function $\mathcal{O}(\{e_i\})$.

Since the whole calculation is differentiable, we can evaluate the derivative of the objective function with respect to the parameters of the RNN via

$$\frac{\partial \mathcal{O}}{\partial p} \frac{\partial p}{\partial f} \frac{\partial f}{\partial W}. \tag{3}$$

Subsequently, we can use the gradients to find embeddings $\{e_i\}$ of our data which optimize the objective function. We apply this insight to combine RNNs with neighbourhood components analysis (NCA), which we will review in a later section.

Long Short-Term Memory

LSTM cells are special stateful transfer functions for RNNs which enable the memorization of events hundreds of time steps apart. We review them shortly because the usage of LSTM cells plays a crucial role in problems where long term dependencies are an important characteristic of the data at hand, necessary to make usable predictions.

The power of LSTM cells is mostly attributed to a special building block, the so called *gating units*. We define $\phi(c, v) = v\sigma(c)$ with σ being the sigmoid function $\frac{1}{1+e^{-x}}$ ranging from 0 to 1.

Another central concept are the *states* $(s_1, s_2, \ldots, s_T)$ of the cell. These can be altered by the inputs via the *input*, *forget* and *output gate*. To keep the notation uncluttered, we concatenate the four different inputs $a_t^{(\cdot)}$ to the cell into a single vector. As indicated by the superscript, each of the $a_t^{(\cdot)}$ represents an input to one of the gates i, f and o. The superscript x represents the input to the cell itself.

$$[a_t^{(x)} \; a_t^{(i)} \; a_t^{(f)} \; a_t^{(g)}] = W_{ha}h_{t-1} + W_{xa}x_t + b_a$$

$$s_t = \underbrace{\phi(a_t^{(i)}, a_t^{(x)})}_{\text{input gate}} + \underbrace{\phi(a^{(f)}, s_{t-1})}_{\text{forget gate}}$$

$$h_t = \sigma(\underbrace{\phi(a_t^{(o)}, s_t)}_{\text{output gate}})$$

$$o_t = W_{ho}h_t + b_h$$

Since all the operations are differentiable, gradient-based can be employed.

3. Sequential Neighbourhood Components Analysis

The central assumption of neighbourhood components analysis [5,16] is that items of the same class lie near each other on a lower-dimensional manifold. To exploit this, we want to learn a function $f : X^* \to Z$ from the sequence space X to a metric space Z that reflects this. Recall that in our case, the embedding function is $e(\mathbf{x}; W) = p(f(\mathbf{x}; W))f$. Given a set of sequences with an associated class label $\mathcal{D} = \{x_i, c_i\}$ mapped to a set of embeddings $\mathcal{E} = \{e_i\}$, we define the probability that a point a selects another point b as its neighbour based on Euclidean pairwise distances as

$$p_{ab} = \frac{\exp(-||e_a - e_b||_2^2)}{\sum_{z \neq a} \exp(-||e_a - e_z||_2^2)},$$

while the probability that a point selects itself as a neighbour is set to zero: $p_{aa} = 0$. The probability that a point i is assigned to class k depends on the classes of the points in its neighbourhood $p(c_i = k) = \sum_j p_{ij} \mathbb{I}(c_j = k)$, where $\mathbb{I}$ is the indicator function.function. The overall objective function is then the expected number of correctly classified points

$$\mathcal{O} = \sum_i \sum_j p_{ij} \mathbb{I}(c_i = c_j).$$

Although NCA has a computational complexity that is quadratic in the number of samples in the training set for training, using batches containing roughly 1000 samples made this negligible. We did not observe any decrease of test performance.

Classifying Sequences

We first train an RNN on our data set with the NCA objective function. Afterwards, all training sequences are propagated through the network and the pooling operator to obtain embeddings $\mathcal{E} = \{e_i\}$ for each of them. We then build a nearest neighbour classifier for which we use all embeddings of the training set. A new sequence $(x_1, x_2, \ldots, x_T)$ is classified by first forward propagating it through the RNN and obtaining an embedding. We then find the k-nearest neighbours and obtain the class by a majority vote.

Note that this method has two appealing characteristics from a computational perspective: first, finding a descriptor for a new sequence has a complexity in the order of the length of that sequence. Furthermore, the memory requirements for that descriptor are invariant of the length of the sequence and can thus be tailored towards memory requirements. Indeed, millions of such descriptors can easily be held in main memory to allow fast similarity search.

4. Experiments

To show that our algorithm works as a classifier we present results on several data sets from the UCR Time Series archive [4]. Due to space limitations, we refer the reader to the corresponding web page for detailed descriptions of each data set. The data sets from UCR are restricted in the sense that all are univariate and of equal length. Since our method is well suited to high dimensional sequences, we we proceed to the well known TIDIGITS benchmark afterwards.

UCR Time Series Data

The hyper parameters for each experiment were determined by random search. We did 200 experiments for each data set, reporting the test error for those parameters which performed best on the training set. The hyper parameters were the number of hiddens, the used transfer function (sigmoid, tangent hyperbolicus rectified linear units or lstm cells), the optimization algorithm (either RPROP or LBFGS), the pooling operator (either sum, max or mean), whether to center and whiten each sequence or the whole data set and the size of the batch to perform gradient calculations on.

Data set	Train	Test	our 1NN	DWT 1NN
Wafers	0.984	0.987	0.987	0.995
Two Patterns	0.992	0.996	0.99725	0.9985
Swedish Leaf	0.797	0.772	0.848	0.843
OSU Leaf	0.684	0.457	0.579	0.616
Face (all)	0.938	0.833	0.647	0.808
Synthetic Control	0.999	0.962	0.96	0.983
ECG	0.999	0.846	0.88	0.88
Yoga	0.684	0.73	0.699	0.845

The training and test performances stated are the average probabilities that a point is correctly classified by the stochastic classifier used in the formulation of NCA. We also report the error for 1-nearest neighbour classification on the test set as 1NN with the training set as a data base to perform nearest neighbour queries on. 1NN-DWT corresponds to the best DWT classification results on the UCR page. If a certrain data set from the UCR repository is not listed, performance was not satisfactory. We attribute this to small training set sizes in comparison with the number of classes with which our method seems to struggle. This is not at all surprising, as the number of parameters is sometimes exceeded by the number of training samples.

TIDIGITS Data

TIDIGITS is a data set consisting of spoken digits by adult and child speakers. We restricted ourselves to the adult speakers. The audio was preprocessed with mel-frequency cepstrum coefficient analysis to yield a 13-dimensional vector at each time step.
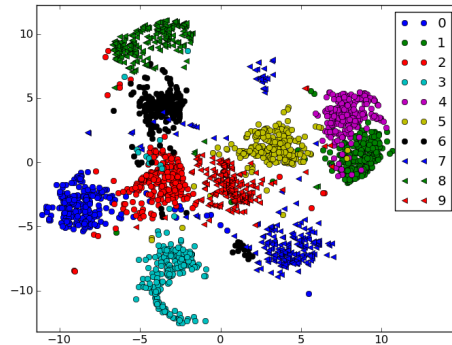
Fig. 1. The output of our method after applying tSNE to the found embeddings. The data is arranged into mostly distinct clusters. Interestingly, the NCA objective also makes it possible for points of the same class to arrange in several clusters. This is not the case for objectives that try to separate the data with a functional form such as a hyperplane.

During training we went along with the official split into a set of 2240 training and 2260 testing samples. 240 samples from the training set were used for validation. We trained the networks until convergence and report the test error with the parameters achieving the best validation error. We used 40 LSTM [9] units to get 30 dimensional embeddings. For comparison, we also trained LSTM-RNNs of similar size with the cross entropy error function for comparison. Since both methods yield discriminative models, we can report the the average probability that a point from the testing set is correctly classified, which was 97.9% for NCA and 92.6% for cross entropy. For a visualization of the found embeddings, see figure 1.

5. Conclusion

We presented a solution to an important problem—by combining two well established methods we introduced a method to embed sequential data into a semantically meaningful feature space: it leads to interpretable features naturally and can be used out of the box as a visualization method and data exploration tool.

The techniques presented here are usable with any RNN structure—we believe that the usage of echo state networks [11] or multiplicative RNNs [14] to NCA might yield even better results.

We also want to stress the applicability of our method to big data: while classifcation has the downside of quadratic complexity, the resulting embeddings are extremly well compressed representations of the data. Also, finding a new representation for an unseen sequence is a single forward pass of an RNN, which is extremly efficient.

References

1. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

2. James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral.

3. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, (to appear), 2011.

4. Keogh E., X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series classification/clustering homepage. 2006.

5. Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2004.

6. Alex Graves and Juergen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18:602–610, 2005.

7. Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *Neural Information Processing Systems*, pages 545–552, 2009.

8. S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 1991.

9. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

10. Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *In Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1998.

11. Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004.

12. Lei Li and B. Aditya Prakash. Time series clustering: Complex is simpler! 2011.

13. James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

14. James Martens, Ilya Sutskever, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

15. M. C Mozer. A focused backpropagation algorithm for temporal pattern recognition. 1989.

16. Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. 2007.

17. Laurens van der Maaten. Learning discriminative fisher kernels. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

18. Ronald J. Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. 1995.