Neural Expectation Maximization

Klaus Greff[*] & Sjoerd van Steenkiste[*] & Jürgen Schmidhuber
Dalle Molle Institute for Artificial Intelligence (IDSIA)
Lugano, Switzerland
`{klaus,sjoerd,juergen}@idsia.ch`

Abstract

We introduce a novel framework for clustering that combines generalized EM with neural networks and can be implemented as an end-to-end differentiable recurrent neural network. It learns its statistical model directly from the data and can represent complex non-linear dependencies between inputs. We apply our framework to a perceptual grouping task and empirically verify that it yields the intended behavior as a proof of concept.

1 Introduction

Many real world tasks such as reasoning and physical interaction require identification and manipulation of conceptual entities. A first step towards solving these tasks is the automated discovery of symbol-like representations that are both distributed and disentangled (Hinton, 1984).

In this paper we are concerned with the domain of images where entities (such as objects) naturally form groups of pixels that share mutual information. We are interested in *perceptual grouping* (or *clustering*) to recover these entities, and build a structured representation that can be used in a symbol-like fashion. This task relies on regularities in the data-generating distribution and should generalize across different images to arbitrary combinations of known entities.

Perceptual grouping has been formalized as inference in a generative compositional model of images, and tackled in several ways. For example Jojic & Frey (2001) use generalized Expectation Maximization (EM) to infer how to split the frames of a video into (inferred) image patches. Masked RBMs (Le Roux et al., 2011) model the objects using a Restricted Boltzmann Machine, and use Block-Gibbs sampling to infer the clustering of pixels. In contrast, we parametrize the distribution over objects using a neural network embedded in an EM procedure for grouping. This framework, which we call *Neural-EM* (N-EM), can be implemented as a recurrent neural network (RNN) and trained end-to-end using gradient descent, similar to Tagger (Greff et al., 2016). However, in comparison we maintain a close connection to the EM framework, which offers theoretical insights and guarantees, thus helping to build a deeper understanding for this class of trainable clustering methods.

2 Neural Expectation Maximization

In this section we derive the *Neural Expectation Maximization* (N-EM) framework based on generalized Expectation Maximization (EM; Dempster et al. 1977). Starting with a brief summary of EM for mixture models, we then describe how the distribution over possible components can be parametrized with a neural network. Subsequently we adapt the steps of EM to the neural model, and show that the resulting computational graph takes the form of a *recurrent neural network* (RNN). Finally we provide an objective function for training this clustering method end-to-end on many images using gradient descent. Note that the following is not restricted to images, but also applies to other modalities.

Mixture Models Clustering is a classic application of EM in which we consider a mixture model, where each cluster k is represented by a distribution with parameters θ_k. A set of binary latent variables $\boldsymbol{Z}$ encodes the cluster assignments, such that $z_{n,k} = 1$ iff point n was generated by cluster k.

[*]Both authors contributed equally to this work

Assuming the points to be iid. given the parameters, the data likelihood for each point x_n becomes:

$$P(x_n|\boldsymbol{\theta}) = \sum_{\mathbf{z}} P(x_n,\mathbf{z}|\theta_k) = \sum_{k=1}^{K} \pi_k P(x_n|z_{n,k}=1,\theta_k), \quad \text{where} \quad P(z_{n,k}=1)=\pi_k. \tag{1}$$

Because of the summation over $\mathbf{z}$, directly optimizing $\log P(\mathbf{x}|\boldsymbol{\theta})$ is often difficult, while the optimization of $\log P(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})$ is usually straightforward. In these cases EM can be used to iteratively compute a *Maximum Likelihood Estimate (MLE)*, by alternating between re-estimation of the latent variables $\mathbf{Z}$ (*E-step*) and of the parameters $\boldsymbol{\theta}$ (*M-step*). This procedure is initialized with random $\boldsymbol{\theta}$ and is guaranteed to converge (albeit only to a local maximum).

Parametrizing Clusters In order to cluster pixels into natural entities, we need to represent complex dependencies between them. Assuming all pixels to be identically distributed is therefore clearly insufficient and instead we choose to parametrize each cluster k by a vector $\theta_k = \{h_1,\dots,h_D\}$. A neural network f transforms this representation into a separate distribution for each pixel. In the simplest case we assume a Gaussian distribution with fixed σ around a mean $\mu_n = f(\theta)_n$ generated by the network for each pixel n:

$$P(x_n|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|f(\theta_k)_n,\sigma). \tag{2}$$

In contrast to (1) we no longer assume the pixels of the image to be identically distributed given $\boldsymbol{\theta}$ and thus are able to represent complex non-linear dependencies between them. Note that the parameters θ refer to the representation, and not to the weights of the neural network (which we treat as fixed for now).

E-Step In the E-step we simply use (2) in order to compute the posterior distribution of the latent variables for each pixel given the last estimate of the parameters: $\boldsymbol{\gamma}_n = P(\mathbf{z}_n|\mathbf{x},\boldsymbol{\theta}^{\text{old}})$.

M-Step The M-Step aims to find the value of $\boldsymbol{\theta}$ that would maximize the expected log-likelihood using the parameter posteriors computed in the E-Step. This quantity presents a lower bound for the log-likelihood and is often written as the Q function:

$$Q(\boldsymbol{\theta},\boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} P(\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{\text{old}}) \log P(\mathbf{x},\mathbf{Z}|\boldsymbol{\theta}). \tag{3}$$

Unfortunately there is no analytical form for finding the maximum of $Q(\boldsymbol{\theta},\boldsymbol{\theta}^{\text{old}})$ for our parametrization. However, since f is differentiable, we can simply improve $\boldsymbol{\theta}$ using gradient ascent[1]:

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \eta \frac{\partial Q}{\partial \boldsymbol{\theta}} \qquad \text{where} \qquad \frac{\partial Q}{\partial \theta_k} \propto \sum_{n=1}^{N} \gamma_{k,n}(f(\theta_k)_n - x_n)\frac{\partial f(\theta_k)_n}{\partial \theta_k}. \tag{4}$$

For sufficiently small learning rate η this will increase the Q function. Neural-EM therefore belongs to the class of generalized EM algorithms, and thus always converges to a (local) optimum of the data log likelihood (Wu, 1983).

N-EM & RNN-EM We observe that the unrolled gradient ascent updates in (4) form a computational graph that is end-to-end differentiable, which allows us to train the network weights by back-propagation through time (eg. Werbos (1988); Williams (1989)). We refer to this trainable procedure as *Neural Expectation Maximization* (N-EM).

The structure of N-EM resembles K copies of a recurrent neural network with hidden states θ_k that, at each timestep, receive $\boldsymbol{\gamma}_k \odot (\boldsymbol{\mu}_k - \mathbf{x})$ as their input. Each generates a new $\boldsymbol{\mu}_k$, which are then used by the E-step to re-estimate $\boldsymbol{\gamma}$. In order for an RNN to accurately mimic the M-Step from (4) to update its hidden state, we must impose several restrictions on its weights and structure. Instead in the RNN-EM procedure we choose to substitute that part of the computational graph of N-EM with a standard RNN, without imposing any restrictions. Although RNN-EM can no longer guarantee convergence of the data log likelihood, its recurrent weights are likely to increase the representational power of the clustering procedure. Figure 1 presents the computational graph of a single RNN-EM (time) step.

[1]Here we assumed the parametrization from (2), yet a similar update arises for many typical parametrizations.

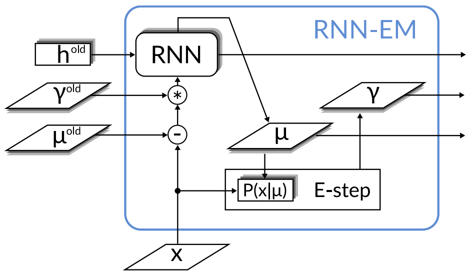


Figure 1: Illustration of a single step of RNN-EM. It receives the prediction error as $(\boldsymbol{x} - \boldsymbol{\mu}^{\text{old}})$ masked with $\boldsymbol{\gamma}^{\text{old}}$ (obtained from the previous E-step) as input. The RNN implements the M-step by combining the input with its previous state to generate a new $\boldsymbol{\mu}$, from which a new $\boldsymbol{\gamma}$ is computed by taking the E-step. Shaded areas indicate K parallel operations (with shared RNN weights).

Training EM In order to cluster the pixels of an image into objects, we require information about the statistical regularities of the data-distribution. In N-EM and RNN-EM, this knowledge is encoded by the weights of the neural network. We train them with gradient descent to minimize a two part loss function:

$$L = -\log \sum_k P(\boldsymbol{X}|\theta_k, \boldsymbol{Z}) \quad + \quad \mathbb{E}_{\boldsymbol{X}}[\log \sum_k P(\boldsymbol{X}|\theta_k, \bar{\boldsymbol{Z}})]. \tag{5}$$

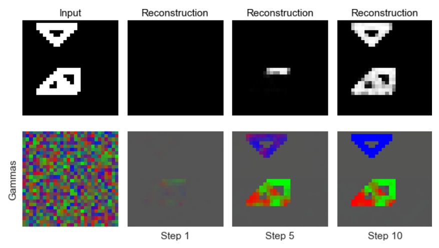
The first term maximizes the data log likelihood (the same as for EM) and pushes each cluster to better reconstruct its pixels. However, by itself, this term only considers the intra-cluster likelihood, which can lead to degenerate solutions like color-quantization where the system relies entirely on the E-step for reconstruction.

The second term is about minimizing the expected out-of-cluster data log likelihood, where $\bar{\boldsymbol{Z}} = 1 - \boldsymbol{Z}$ denotes a point *not* being part of a cluster. This part is equivalent to the cross entropy error between the predictions and the prior for $\boldsymbol{X}$ masked by $1 - \boldsymbol{\gamma}$. It pushes each cluster to specialize and not make strong predictions about points in other clusters.

3 Empirical Validation and Future Directions

We present preliminary results of N-EM and RNN-EM on *Shapes* (Reichert & Serre, 2013) and *Pascal VOC* (Everingham et al., 2015) as a proof of concept of the introduced framework.

Figure 2 displays the grouping learned by N-EM and RNN-EM on *Shapes*. The groupings learned by RNN-EM are markedly better, averaging an *Adjusted Mutual information (AMI)* score of 0.83 across the validation set, compared to a less powerfull N-EM with an *AMI* score of 0.41. We observe that RNN-EM outperforms Reconstruction Clustering (Greff et al., 2015) and its performance is comparable to Tagger (Greff et al., 2016), even though a much simpler architecture (single-layer RNN with 250 neurons) was used.



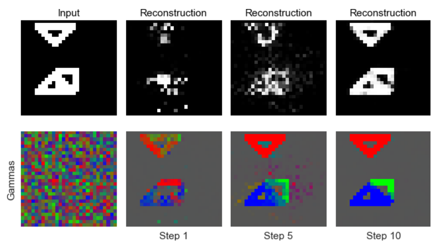


Figure 2: Learned groupings by *N-EM* (left) and *RNN-EM* (right) on the Shapes dataset.

Figure 3 in Appendix A displays the groupings learned by RNN-EM on Pascal VOC. These are far from optimal and we hypothesize that this is the result of insufficient representational capacity by the neural network used (single layer RNN with 2000 neurons). Nevertheless, it is promising that even in suboptimal conditions RNN-EM makes an attempt at grouping, in this case mostly based on color.

Future Directions In future work we continue to improve RNN-EM on the perceptual grouping task. Additionally we want to adapt RNN-EM to learn object-like representations for sequential data (e.g. by considering a next-step prediction objective in the recurrence) and to compose these representations hierarchically.

References

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society.*, pp. 1–38, 1977.

Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Binding via Reconstruction Clustering. *arXiv:1511.06418 [cs]*, November 2015.

Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hotloo Hao, Jürgen Schmidhuber, and Harri Valpola. Tagger: Deep Unsupervised Perceptual Grouping. *arXiv:1606.06724 [cs]*, June 2016.

Geoffrey E. Hinton. Distributed representations. 1984.

Nebojsa Jojic and Brendan J. Frey. Learning flexible sprites in video layers. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference On*, volume 1, pp. I–I. IEEE, 2001.

Nicolas Le Roux, Nicolas Heess, Jamie Shotton, and John Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

David P. Reichert and Thomas Serre. Neuronal Synchrony in Complex-Valued Deep Networks. *arXiv:1312.6115 [cs, q-bio, stat]*, December 2013.

Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

Ronald J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical report, Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

CF Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of statistics*, pp. 95–103, 1983.

A Pascal VOC Visualizations

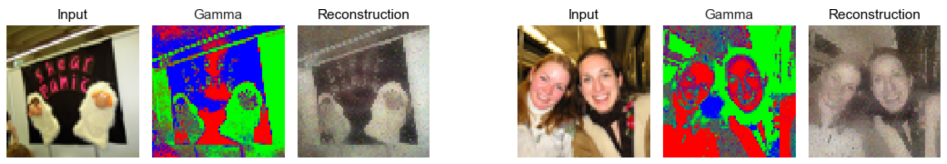


Figure 3: Learned groupings by RNN-EM on Pascal VOC.

B Experimental Setup

The following subsections provide detailed information about the experimental setup of our emperical evaluation.

B.1 Shapes

For RNN-EM we use a single layer RNN with 250 sigmoidal neurons followed by a sigmoid output layer that corresponds to the number of pixels in the image. Similarly for N-EM we use a hidden layer with 250 sigmoidal units and a sigmoid output layer. At each M-step we propagate the corresponding gradient ascent update for the Q-function from Equation 3 once (as described in Equation 4 but adapted to the binary case and corresponding likelihood based on binomial cross-entropy).

We set $K = 3$ and use 10 EM steps for both architectures. We find that further increasing this number no longer provides a substantial improvement. When decreasing the number of EM steps we find performance drop exponentially until performance reaches an AMI score of approximately 0.1.

Both models are trained with Adam using the default learning rate of 0.001. We stop training after the validation reconstruction error[2] no longer decreased for 10 epochs. We have evaluated each model with different random seeds and found the reported results to be stable. In all cases we were able to obtain the reported AMI score within 100-120 epochs.

B.2 Pascal VOC

We used the same RNN-EM configuration as described for *Shapes* except for an increase in the number of neurons in the hidden layer from 250 to 2000, usage of linear output units, and a decrease in the learning rate of Adam from 0.001 to 0.0001. Our observations regarding stability across different random seeds remains the same.

[2]Note that we do not stop on the AMI score as this is not part of our objective function and only measured to evaluate the performance *after training*.