Semi-Supervised Learning with Ladder Networks (2015)

Rasmus, Antti Valpola, Harri Honkala, Mikko Berglund, Mathias Raiko, Tapani

Resume

March 29, 2019

1 Introduction

In this work, the authors propose a new usage to the Ladder network architecture in a semi supervised setting. Ladder networks combine supervised learning with unsupervised learning in deep neural networks. Often, unsupervised learning was used only for pre-training the network, followed by normal supervised learning. In case of ladder networks, it is trained to simultaneously minimize the sum of supervised and unsupervised cost functions by backpropagation, avoiding the need for layer-wise pre-training. Ladder network is able to achieve state-of-the-art performance in semi-supervised MNIST and CIFAR-10 classification, in addition to permutation-invariant MNIST classification with all labels.

1.1 Key Aspects

Compatibility with supervised methods It can be added to existing feedforward neural networks. The unsupervised part focuses on relevant details found by supervised learning. It can also be extended to be added to recurrent neural networks.

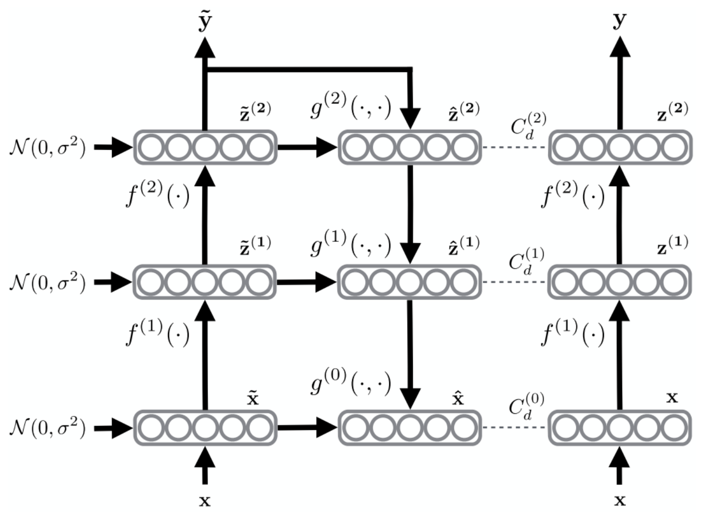
Scalability resulting from local learning In addition to a supervised learning target on the top layer, the model has local unsupervised learning targets on every layer, making it suitable for very deep neural networks.

Computational efficiency Adding a decoder (part of the ladder network) approximately triples the computation during training but not necessarily the training time since the same result can be achieved faster through the better utilization of the available information.

1.2 Justification

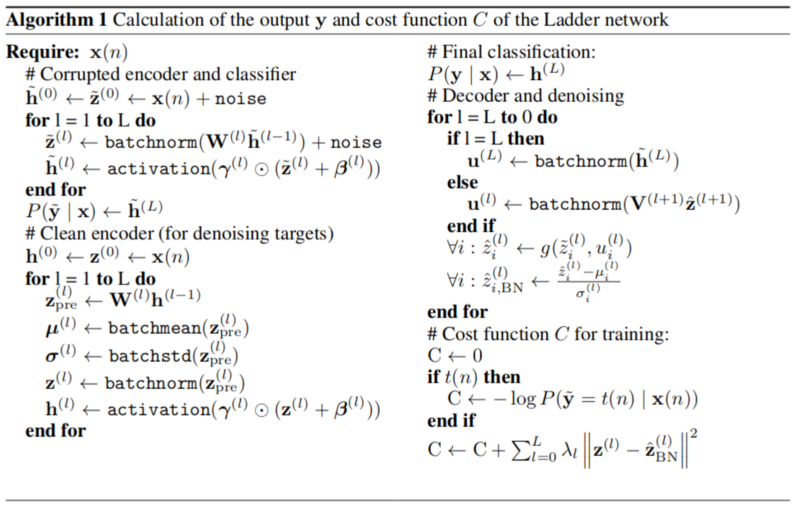
In an semi supervised setting, we will use latent variable models given that it combines both a supervised and an unsupervised leaning in a principled way, the training process of latent variable model can be split into training and inferent, let's take the EM algorithm as an example, we start by E step finding the expectation of the latent variable (finding which class each variable belong to), and based on this expectation we apply an M step to maximise the probability model (the parameters of each probability distribution / prior distributions), in our case, we will use denoise autoencoder as a gateaway to modeling these latent variable models, the goal is to reconstruct a latent z using a prior $p(z)$ and an observation $\tilde{z} = z + \text{noise}$, the prior can be modeled by the output of a network (a decoder), in this paper, tha authors use a ladder network, that is more of denoising source sepration (DSS), given that we'll reconstruct / denoise all the intermediate activation in the network $\hat{z} = g(\tilde{z})$ and the objective is to reconstuct the latent variable using the corupted version of the activation $\hat{z} = z$, unlike dAE (denoising autoencoders) where we normally trained to denoise the observations only and learning is based simply on minimising the norm of the different between the original input x and its reconstructed version $\hat{x}$ from the corrupted input $\tilde{x}$, that is the cost $||\hat{x} - x||^2$.

2 The Model



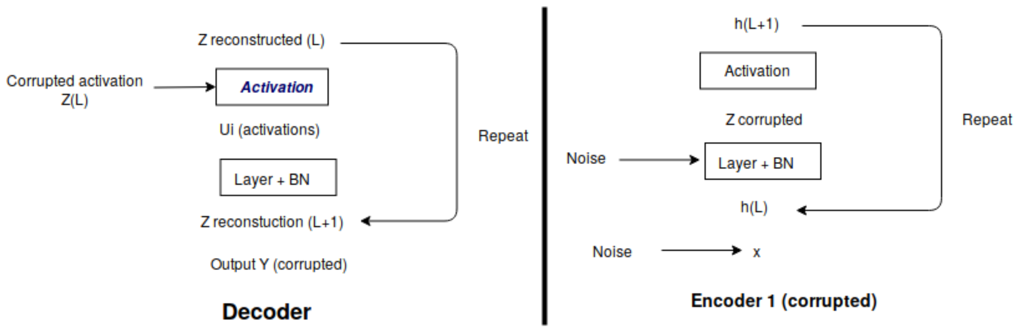
The model contains two encoders, one taking as an input a corrupted version $\tilde{x}$, and where we also add gaussian noise at each layers outputs (after batch norm and beforf the non linearity) and the final output is $\tilde{y}$, and a second encoder where we input the clean version of x and we don't add any noise in the intermediate layer, thus obtaining a clean prediction y. And an important part of the network is a decoder, which is reponsible of computing the reconstruction of each layers activations based on the corrupted versions, taking as inputs in each layers, the upcoming activatio as usual, plus the activation of the encoder using a set of skip connections, and we then compute the unsupervised loss $C_d^{(l)} = \left\|\mathbf{z}^{(l)} - \hat{\mathbf{z}}^{(l)}\right\|^2$, plus the supervised loss comping from the clean encoder in case of labeled examples.

3 Algorithm



Algorithm 1 Calculation of the output $\mathbf{y}$ and cost function C of the Ladder network

```
Require: $\mathbf{x}(n)$
  # Corrupted encoder and classifier
  $\tilde{\mathbf{h}}^{(0)} \leftarrow \tilde{\mathbf{z}}^{(0)} \leftarrow \mathbf{x}(n) + \texttt{noise}$
  for l = 1 to L do
    $\tilde{\mathbf{z}}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)}) + \texttt{noise}$
    $\tilde{\mathbf{h}}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  $P(\tilde{\mathbf{y}} \mid \mathbf{x}) \leftarrow \tilde{\mathbf{h}}^{(L)}$
  # Clean encoder (for denoising targets)
  $\mathbf{h}^{(0)} \leftarrow \mathbf{z}^{(0)} \leftarrow \mathbf{x}(n)$
  for l = 1 to L do
    $\mathbf{z}_{\text{pre}}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$
    $\boldsymbol{\mu}^{(l)} \leftarrow \texttt{batchmean}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\boldsymbol{\sigma}^{(l)} \leftarrow \texttt{batchstd}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\mathbf{z}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{z}_{\text{pre}}^{(l)})$
    $\mathbf{h}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  # Final classification:
  $P(\mathbf{y} \mid \mathbf{x}) \leftarrow \mathbf{h}^{(L)}$
  # Decoder and denoising
  for l = L to 0 do
    if l = L then
      $\mathbf{u}^{(L)} \leftarrow \texttt{batchnorm}(\tilde{\mathbf{h}}^{(L)})$
    else
      $\mathbf{u}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)})$
    end if
    $\forall i : \hat{z}_i^{(l)} \leftarrow g(\tilde{z}_i^{(l)}, u_i^{(l)})$
    $\forall i : \hat{z}_{i,\text{BN}}^{(l)} \leftarrow \frac{\hat{z}_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}}$
  end for
  # Cost function $C$ for training:
  $\text{C} \leftarrow 0$
  if $t(n)$ then
    $\text{C} \leftarrow -\log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x}(n))$
  end if
  $\text{C} \leftarrow \text{C} + \sum_{l=0}^{L} \lambda_l \left\|\mathbf{z}^{(l)} - \hat{\mathbf{z}}_{\text{BN}}^{(l)}\right\|^2$
```



One important note, is that the activations in the decoder are different, and are computed as follows:

$$\hat{z}_i^{(l)} = g_i\left(\tilde{z}_i^{(l)}, u_i^{(l)}\right) = \left(\tilde{z}_i^{(l)} - \mu_i\left(u_i^{(l)}\right)\right) v_i\left(u_i^{(l)}\right) + \mu_i\left(u_i^{(l)}\right)$$