Neural Autoregressive Distribution Estimators (NADEs)

Isaac De Vlugt

May 11, 2020

NADE = Neural Autoregressive Distribution Estimator. It's inspired by an RBM and aims to provide a tractable distribution estimator. The probability of a given configuration as modelled by an RBM requires the calculation of the partition function, which is intractable. Recall that for an RBM,

$$Z = \sum_{h \in \mathcal{H}_h} \sum_{v \in \mathcal{H}_v} e^{-E(v,h)},$$

and

$$p(\mathbf{v}) = \frac{e^{-\sum_{h \in \mathcal{H}_h} E(\mathbf{v},h)}}{Z}.$$

An RBM as a Bayesian Network

If we can maybe write $p(\mathbf{v})$ as a product of tractable conditionals, then we can bypass calculating Z entirely. We can write $p(\mathbf{v})$ exactly as a product of conditionals that aren't tractable as follows.

$$\begin{aligned} p(\mathbf{v}) =& \prod_i p(v_i|\mathbf{v}_{<i}) \\ =& \prod_i \frac{p(v_i, \mathbf{v}_{<i})}{p(\mathbf{v}_{<i})} \end{aligned}$$

If we can approximate the numerator and denominator, then there may be instances where the above expression is tractable.

Consider a mean-field distribution approach for the approximation (recall that a mean-field approximation just relates to the idea that our variables are independent, e.g. $p(a,b) = p(a)p(b)$): approximate $p(v_i|\mathbf{v}_{<i})$ by finding a tractable approximation for $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i}) \approx q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ such that $q(v_i|\mathbf{v}_{<i})$ is obtainable. In our mean-field approximation, $q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$, for $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$,

$$\begin{aligned} q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i}) =& q(v_i|\mathbf{v}_{<i}) q(\mathbf{v}_{>i}|\mathbf{v}_{<i}) q(\mathbf{h}|\mathbf{v}_{>i}) \\ =& q(v_i|\mathbf{v}_{<i}) \prod_{j>i} q(v_j|\mathbf{v}_{<i}) \prod_k q(h_k|\mathbf{v}_{<i}). \end{aligned}$$

Let us now write the conditional distribution $q(v_j|\mathbf{v}_{<i})$ as binomial with a success probability $\mu_j(i)$. Noting again that $v_i \in 0, 1$,

$$q(v_j|\mathbf{v}_{<i}) = \mu_j(i)^{v_j}(1 - \mu_j(i))^{1-v_j}.$$

We may assume a similar form for $q(h_k|\mathbf{v}_{<i})$:

$$q(h_k|\mathbf{v}_{<i}) = \tau_k(i)^{h_k}(1-\tau_k(i))^{1-h_k}.$$

Therefore,

$$q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i}) = \mu_i(i)^{v_i}(1-\mu_i(i))^{1-v_i}\prod_{j>i}\mu_j(i)^{v_j}(1-\mu_j(i))^{1-v_j}\prod_k \tau_k(i)^{h_k}(1-\tau_k(i))^{1-h_k},$$

with

$$\begin{aligned}\mu_j(i) &= q(v_i = 1|\mathbf{v}_{<i}) \approx p(v_i = 1|\mathbf{v}_{<i}) \\ \tau_k(i) &= q(h_k = 1|\mathbf{v}_{<i}) \approx p(h_k = 1|\mathbf{v}_{<i}).\end{aligned}$$

Our approximation proceeds by finding $\mu_j(i)$ for $j \geq i$ and $\tau_k(i)$ which minimize the KL divergence between $q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ and $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$. There exists an algebraic solution for this problem:

$$\begin{aligned}\tau_k(i) &= \text{sigm}\left(c_k + \sum_{j\geq i} W_{jk}\mu_j(i) + \sum_{j<i} W_{kj}v_j\right) \\ \mu_j(i) &= \text{sigm}\left(b_j + \sum_k W_{kj}\tau_k(i)\right) \forall j \geq i.\end{aligned}$$

Note that these expressions depend on their counterparts (i.e. $\tau_k(i)$ depends on $\mu_j(i)$ and visa versa), and there is no exact solution for this set of non-linear equations. Similar to a Gibbs sampling procedure where we bounce back and forth between the hidden and visible layers to infer the other conditioned upon the current, we can bounce back and forth between $\mu_j(i)$ and $\tau_k(i)$ (initialize them to 0) and we are guaranteed to come to some equilibrium values of $\mu_j(i)$ and $\tau_k(i)$. Then, $\mu_j(i)$ is used to approximate $p(v_j = 1|\mathbf{v}_{<i})$. This takes about 20 iterations are each iteration is quite costly when the RBM gets larger. Moreover, we'd need to do this for every v_i. So, this mean-field approximation does not actually end up being tractable.

NADEs: Building off of the mean-field RBM

Let's build off of this mean-field idea presented in the previous section. However, it's at this point where the physical ties to an RBM sort of vanish. How the NADE architecture is formed is simply "inspired by" this mean-field approximation for an RBM.

For one iteration with $\mu_j(i)$ initialized to zero,

$$\mu_j(i)^{(0)} = 0 \rightarrow \tau_k(i)^{(0)} = \text{sigm}\left(c_k + \sum_{j<i} W_{kj}v_j\right) \rightarrow \mu_j(i)^{(1)} = \text{sigm}\left(b_j + \sum_k W_{kj}\tau_k(i)^{(0)}\right)$$

This is simply a feed-forward network with one hidden layer and shared weights going across the networks. There are N networks to train, where N is the number of sites. The input to the i^{th} network is the $j < i$ preceeding parts of the visible layer (i.e. all spin values before the i^{th} site) as dictated by $\sum_{j<i}$. The activation / hidden layer is given by $\tau_k(i)^{(0)}$, and the ouput of the network is $\mu_j(i)^{(1)}$ which we take to be the desired conditional $p(v_i = 1|\mathbf{v}_{<i})$. The model is autoregressive!

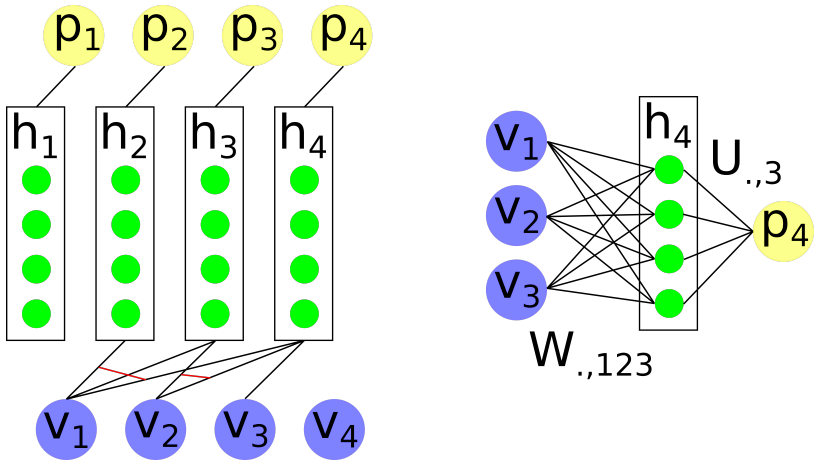

Figure 1: Illustration of a NADE. Here, $p_i = p(v_i = 1|\mathbf{v}_{<i})$. Red lines correspond to shared connections. The network to the right is simply an unfolded version of the fourth feed-forward neural network.

It turns out to be benefitial computationally to "undo" the shared parameters between the output and input (they share the same weight matrix!). Instead, we can just train a seperate set of weights connecting the output of the networks with the hidden layers. Each of the N networks look like the following.

$$\begin{aligned} \text{input} =& \mathbf{v}_{<i} \\ \text{activation} =& \mathbf{h}_i = \text{sigm}(\mathbf{c} + \mathbf{W}_{\cdot,<i}\mathbf{v}_{<i}) \\ \text{output} =& p(v_i = 1|\mathbf{v}_{<i}) = \text{sigm}(b_i + \mathbf{U}_{i,\cdot}\mathbf{h}_i) \qquad \text{not} \qquad \text{sigm}(b_i + \mathbf{W}_{i,\cdot}^T\mathbf{h}_i) \end{aligned}$$

There's a lot of "reusing" parameters in each of the N networks. Let's just jot some things down to understand the architecture better.

- For N sites, there are N hidden layers that each comprise of n_h neurons.
- $\mathbf{W}$ ($n_h \times N$ matrix) and $\mathbf{c}$ are shared by all N networks.
- $\mathbf{U}$ is $N \times n_h$.

Training the NADE boils down to minimizing the negative log-likelihood of the parameters given the training set.

$$\begin{aligned} \text{NLL} &= -\frac{1}{|D|}\sum_{\mathbf{v}\in D}\log p(\mathbf{v}) \\ &= -\frac{1}{|D|}\sum_{\mathbf{v}\in D}\sum_{i=1}^{N}\log p(v_i = 1|\mathbf{v}_{<i}) \end{aligned}$$

The autoregressive probability, $p(\mathbf{v})$, is calculated via the following procedure.

Algorithm 1: Calculating $p(\mathbf{v})$

$p(\mathbf{v}) = 1$
$\mathbf{a} = \mathbf{c}$
for $i = 1$ to N **do**
  $\mathbf{h}_i \leftarrow \text{sigm}(\mathbf{a})$
  $p(v_i = 1|\mathbf{v}_{<i}) \leftarrow \text{sigm}(b_i + \mathbf{U}_{i,\cdot}\mathbf{h}_i)$
  $p(\mathbf{v}) \leftarrow p(\mathbf{v}) \times p(v_i = 1|\mathbf{v}_{<i})^{v_i} \times (1 - p(v_i = 1|\mathbf{v}_{<i}))^{1-v_i}$
  $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{W}_{\cdot,i}v_i$
end
return $p(\mathbf{v})$

Now, we can calculate gradients of the NLL w.r.t. the NADE parameters $(\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{c})$. Note, $\bigodot$ refers to an element-wise multiplication. Here is the algorithm for gradient calculation.

Algorithm 2: Gradients of NLL w.r.t. NADE parameters from a data point, $\mathbf{v}$.

$\delta\mathbf{a} = 0$
$\delta\mathbf{c} = 0$
for $i = 1$ to N **do**
  $\delta b_i \leftarrow p(v_i = 1|\mathbf{v}_{<i}) - v_i$
  $\delta\mathbf{U}_{i,\cdot} \leftarrow (p(v_i = 1|\mathbf{v}_{<i}) - v_i)\,\mathbf{h}_i$
  $\delta\mathbf{h}_i \leftarrow (p(v_i = 1|\mathbf{v}_{<i}) - v_i)\,\mathbf{U}_{i,\cdot}$
  $\delta\mathbf{c} \leftarrow \delta\mathbf{c} + \delta\mathbf{h}_i \bigodot \mathbf{h}_i \bigodot (1 - \mathbf{h}_i)$
  $\delta\mathbf{W}_{\cdot,i} \leftarrow \delta\mathbf{a}v_i$
  $\delta\mathbf{a} \leftarrow \delta\mathbf{a} + \delta\mathbf{h}_i \bigodot \mathbf{h}_i \bigodot (1 - \mathbf{h}_i)$
end
return $\delta\mathbf{b}, \delta\mathbf{c}, \delta\mathbf{W}, \delta\mathbf{U}$