Tejas, Muskaan, Ojas
22B0909, 22B1058, 22B0965
Advanced Machine Unlearning

Learning with Hidden Variables

Advanced Machine Learning (Spring 2024)

2025-02-19

Some tasks, such as training Hidden Markov Models for natural language processing, require training with data available (supervision) only over a subset of the variables and prediction of all variables with the same, used in the generation of sentences.

We consider learning in an unconditional setting first since the conditional setting is just a variant with extra constraints. Further, we train our model as a whole using entire sentences rather than one variable at a time.

The Framework 1

We wish to learn $\vec{\theta}$, the parameters of the graphical model G with set of cliques C, in order to maximize

$$P_{\vec{\theta},G}\left(y_1, y_2, \ldots, y_n, z_1, z_2, \ldots z_m | \vec{x}\right) = \frac{1}{Z_{\vec{\theta}}(\vec{x})} \exp\left(\sum_C F_{\vec{\theta}}\left(\vec{y_C}, \vec{z_C}, \vec{x}\right)\right)$$

Specifically, given the dataset $D = \{(\vec{x_i}, \vec{y_i}) : i \in \{1 \ldots N\}\}$

$$\vec{\theta}_{ML} \;=\; \arg\max_{\vec{\theta}} \left(\sum_{i=1}^{N} \log\; P_{\vec{\theta}}(\vec{y_i}|\vec{x_i})\right) \;=\; \arg\max_{\vec{\theta}} \left(\sum_{i=1}^{N} \log\left(\sum_{\vec{z}} P_{\vec{\theta},G}(\vec{y_i}, \vec{z}|\vec{x_i})\right)\right)$$

This optimization problem, which contains a summation inside a logarithm, is difficult to solve. We will apply approximation algorithms such as the Expected Maximization (EM) algorithm to train such networks successfully.

1.a Variational Approach: Adding New Auxiliary Variables

If we have a variable z that takes k values, we show that

$$\log\left(\sum_{z=1}^{k} g(y, z)\right) \;=\; \max_{\vec{q}} Q(\vec{q}, g)$$

where $\vec{q} = \{q_1, q_2, \ldots q_k\}$ are auxiliary variables such that $\sum_z q_z = 1$ and $q_z \geq 0\ \forall z$ and

$$Q(\vec{q}, g) = \sum_z q_z \log g(y, z) - \sum_z q_z \log q_z$$

NOTE: $-\sum_z q_z \log q_z$ is the **entropy** over q_z.

Proof:

We introduce the Lagrangian parameter λ, and define

$$L(Q, \lambda) = Q(\vec{q}, g) + \lambda\left(\sum_z (q_z) - 1\right)$$

The second term evaluates to zero, but it eases calculation in the next step, where we attempt to find the maxima by differentiating w.r.t. all q_j. Specifically, we solve

$$\frac{\partial L(Q, \lambda)}{\partial q_j} = \log g(y, j) - \log q_j - 1 + \lambda = 0$$

This gives us

$$q_j* = \alpha \cdot g(y, j)$$

for some constant α which is common across all j and dependent on λ.
Using the normalization constraint on q_z, we get

$$q_j* = \frac{g(y, j)}{\sum_z g(y, z)}$$

Substituting this,

$$Q(\vec{q}*, g) \;=\; \sum_z q_z \log\left(\frac{g(y, z)}{q_z}\right) \;=\; \sum_z \left(\frac{g(y, z)}{\sum_z g(y, z)} \log\left(\sum_z g(y, z)\right)\right) \;=\; \log\left(\sum_z g(y, z)\right)$$

1.b Applying Variational Approach

We assumed z to be an integer variable in the proof, but the same does not use this fact and does not depend on the kind of z; z could take any values, but $\vec{q}$ has as many values in it as the number of possible values of z.

We now consider the i^{th} term in the outer sum alone, $\vec{z}$ instead of z and $P_{\vec{\theta},G}(\vec{y_i}, \vec{z}|\vec{x_i})$ as $g(y,z)$; q_z is replaced by $q_{i,\vec{z}}$. Using the same theorem,

$$\log\left(\sum_{\vec{z}} P_{\vec{\theta},G}(\vec{y_i},\vec{z}|\vec{x_i})\right) \;=\; \max_{\vec{q_i}\,:\,q_{i,\vec{z}}\geq 0,\;\sum_{\vec{z}} q_{i,\vec{z}}=1}\left(\sum_{\vec{z}} q_{i,\vec{z}}\log P_{\vec{\theta},G}(\vec{y_i},\vec{z}|\vec{z_i}) \;-\; \sum_{\vec{z}} q_{i,\vec{z}}\log q_{i,\vec{z}}\right)$$

This reduces our task to solving for θ_{ML}, the optimal parameters:

$$\vec{\theta}_{ML} \;=\; \arg\max_{\vec{\theta}}\sum_{i=1}^{N}\left(\max_{\vec{q_i}\,:\,q_{i,\vec{z}}\geq 0,\;\sum_{\vec{z}} q_{i,\vec{z}}=1}\left(\sum_{\vec{z}} q_{i,\vec{z}}\log P_{\vec{\theta},G}(\vec{y_i},\vec{z}|\vec{z_i}) \;-\; \sum_{\vec{z}} q_{i,\vec{z}}\log q_{i,\vec{z}}\right)\right)$$

Change of Notation

We now work only with generative models, and everything that follows can be extended to other models too.

Before we proceed to the methods for solving for $q_{i,\vec{z}}$, we would change our problem notation:

- Without loss of generality, that the earlier $\vec{x}$ is absent.
- We use just $\vec{x}$ to represent what was earlier $\vec{y}$
- We express $P_{\vec{\theta},G}(\vec{y_i},\vec{z}|\vec{x_i})$ as $P_{\vec{\theta}}(\vec{x_i},\vec{z}) = P_{\vec{\theta}}(\vec{x_i}|\vec{z})\cdot P_{\vec{\theta}}(\vec{z})$.
- Thus, our optimization problem can also be expressed as

$$\vec{\theta}_{ML} = \arg\max_{\vec{\theta}}\sum_{i=1}^{N}\max_{\vec{q_i}}\left(\sum_{\vec{z}} q_{i,\vec{z}}\log P_{\vec{\theta}}(\vec{x_i}|\vec{z}) - \sum_{\vec{z}} q_{i,\vec{z}}\log\left(\frac{P_{\vec{\theta}}(\vec{z})}{q_{i,\vec{z}}}\right)\right)$$

1.c Methods of solving $q_{i,\vec{z}}$

1. **Expectation Maximization (EM) algorithm**: we alternatively solve for $\vec{q_i}$ and $\vec{\theta}$ keeping the other fixed. This is not tractable for high-dimensional $\vec{z}$, though.
2. **Variational Auto-Encoders (VAE) algorithm**: we use a neural network which outputs $\vec{q_i}$ given input $\vec{x_i}$. Note that the neural network outputs the same in the form of a distribution $q_\phi(\vec{z}|\vec{x_i})$.
3. **Denoising Diffusion Models**: we break up $P_{\vec{\theta}}(\vec{x_i}|\vec{z})$ term into many smaller steps, assuming a fixed $q_{i,\vec{z}}$ for each step.
4. **Normalizing Flows (NF)**: We assume a simple, invertible function to implement $P_{\vec{\theta}}(\vec{x_i|\vec{z}})$ in order to compute the optimum $q_{i,\vec{z}}$ in closed form. This is the odd one out, and it relies on a deep generative model itself.

The NF method stands out among these, for reasons that we will see shortly.

The EM Algorithm 2

We repeatedly find the optimum $\vec{q_i}$, fixing $\vec{\theta}$ and vice-versa. At the t^{th} iteration,
The E-step: We solve for the posterior distribution of hidden variable $\vec{q_i}$ or $q_{i,\vec{z}}$ keeping $\vec{\theta} = \vec{\theta}_t$ fixed.

$$q_{t,i,\vec{z}} = \frac{P(\vec{x_i},\vec{z}\mid\vec{\theta}_t)}{\sum_{\vec{z}} P(\vec{x_i},\vec{z}\mid\vec{\theta}_t)} = P(\vec{z}\mid\vec{x_i},\vec{\theta}_t)$$

The M-step: We solve for $\vec{\theta}$ keeping fixed $\vec{q_i} = \vec{q_{t,i}}$.

$$\vec{\theta}_{t+1} = \arg\max_{\vec{\theta}}\sum_{i=1}^{N}\sum_{\vec{z}} q_{t,i,\vec{z}}\log P(\vec{x_i},\vec{z}|\vec{\theta})$$

This is concave in $\vec{\theta}$ and can often be solved in closed form, via factorization of $q_{\vec{t},i}$ over cliques in graphical models, or directly in closed form in HMMs.

The overall algorithm: This behaves somewhat like the block coordinate ascent algorithm, trying to optimize $\vec{q_i}$ and $\vec{\theta}$ alternatively in the E and M steps, and it can be shown that the algorithm is guaranteed to converge.

2.a Takeaways and Limitations of EM algorithm

- The algorithm is efficient when dealing with tractable graphical models such as HMMs or graphical models (where the distribution can be factorized) and a small number of dimensions for $\vec{z}$.
- However, this becomes intractable where $\vec{z}$ has a lot of dimensions, since the number of possible values of $\vec{z}$ is k^d and $\vec{q_i}$ has that many values. This cannot be used to train images, where each pixel constitutes three dimensions in the vector $\vec{z}$. This also cannot be used when $\vec{z}$ does not take discrete values.
- When not all training variables are observed in each training sample, learning of potentials that express interactions between variables becomes difficult.

High Dimensional Objects and Continuous Domains 3

Let $X \subset R^n$ be the space of all possible generated objects for a large n, and C be the space of possible contexts corresponding to objects in X. We let $Z \subseteq R^{k_0}$ be the space of latent variables, which we assume to fit into some known distribution $P(Z)$, such as the standard Gaussian distribution. Unlike in the EM algorithm, we deal with continuous variables here.

Further, our algorithm depends on the existence of a parametric network $F_{\vec{\theta}} : Z, C \to X$, which is a high-capacity, deep neural network.

Our objective is to

1. Train the model $F_{\vec{\theta}}$: learn its parameters $\vec{\theta}$ using the dataset $D = \{(\vec{c}_i, \vec{x}_i)\}$ which contains N objects, drawn from an unknown distribution $P(\vec{c}, \vec{x})$.
2. Generate, given a context $\vec{c}$, a sentence $\vec{x} = F_{\vec{\theta}}(\vec{z}, \vec{c})$ where z is sampled from $P(Z)$. Here, $\vec{x}$ may be a high-dimensional object that needs to be generated each time.

Since the context $\vec{c}$ can be included later, we keep things simple, i.e., drop the context and just focus on unconditional generation.

Variational Auto-Encoders 4

Train $F_{\vec{\theta}}$ to model $P_{\vec{\theta}}(\vec{x}|\vec{z})$.

We have prior knowledge of $P(\vec{z})$, and $\vec{\theta}$ are the model parameters.
We need to learn $\vec{\theta}$s to maximize the likelihood of the training data.
The BN for this is:

$$\vec{z} \longrightarrow \vec{x}$$

4.a What $P(\vec{x}|\vec{z})$ looks like?

The distribution can be very complicated, for tasks like image generation etc. So, it is assumed to be a conditional Gaussian distribution, with parameters depending on $\vec{z}$ and on $\vec{\theta}$, which can be obtained when training the **decoder** neural network $F_{\vec{\theta}}$.

$$P(\vec{x}|\vec{z}) = \mathcal{N}\left(\vec{\mu} = [\mu_{x_1|\vec{z}}, \mu_{x_2|\vec{z}}, \dots \mu_{x_n|\vec{z}}], \quad \mathbf{\Sigma} = diag[\sigma_{x_1|\vec{z}}, \sigma_{x_2|\vec{z}}, \dots \sigma_{x_n|\vec{z}}]\right)$$

Each $\mu_{x_i|\vec{z}}$ and $\sigma_{x_i|\vec{z}}$ depends on some of the parameters in $\vec{\theta}$.

4.b Training VAEs

We are given training data: $D = \{\vec{x_i}\}_{i=1}^{N}$ and $\vec{z}$ is hidden.

We wish to compute

$$\vec{\theta}_{ML} \;=\; \arg\max_{\vec{\theta}} \sum_{i=1}^{N} \log P_{\vec{\theta}}(\vec{x_i})$$

Marginalizing out z to get to get $P_{\vec{\theta}}(\vec{x_i})$

$$\arg\max_{\vec{\theta}} \sum_{i=1}^{N} \log P_{\vec{\theta}}(\vec{x_i}) \;=\; \arg\max_{\vec{\theta}} \sum_{i=1}^{N} \log \int_{\vec{z}} P_{\vec{\theta}}(\vec{x}|\vec{z}) P_{\vec{\theta}}(\vec{z}) d\vec{z}$$

Here, $z \in \mathbb{R}^k$ and $x \in \mathbb{R}^n$

Therefore, integrating over all hidden variables $\vec{z}$ is intractable.

So, we use the "variational approximation" (similar to the variational approach), i.e.

$$\log \int_{\vec{z}} P_{\vec{\theta}}(\vec{x}, \vec{z}) d\vec{z} \;=\; \max_{q_{i,\vec{z}}} \int_{\vec{z}} \left(q_{i,\vec{z}} \log(P_{\vec{\theta}}(\vec{x}, \vec{z}).P(\vec{z})) - q_{i,\vec{z}} \log q_{i,\vec{z}} \right) d\vec{z}$$

where $q_{\vec{z}} \geq 0, \ \int_{\vec{z}} q_{\vec{z}} d\vec{z} = 1$

We cannot compute the integral for all $i, \vec{z}$, so we have another intractable inner maximization problem.

To get $q_{i,\vec{z}}$, we use another neural network: $q_{\vec{\phi}}(\vec{z}|\vec{x_i})$ with parameters $\vec{\phi}$ shared across all i.

Thus,

$$\max_{\vec{\theta}} \sum_{i=1}^{N} \log P_{\vec{\theta}}(\vec{x_i}) \geq \max_{\vec{\theta}, \vec{\phi}} \sum_{i=1}^{N} \left(\underbrace{\int_{\vec{z}} q_{\vec{\phi}}(\vec{z}|\vec{x_i}) \log \left(P_{\vec{\theta}}(\vec{x_i}|\vec{z}) \right) d\vec{z}}_{\text{First term}} - \underbrace{\int_{\vec{z}} q_{\vec{\phi}}(\vec{z}|\vec{x_i}) \log \left(\frac{q_{\vec{\phi}}(\vec{z}|\vec{x_i})}{P(\vec{z})} \right) d\vec{z}}_{\text{Second term}} \right)$$

We have taken out the factor $P(\vec{z})$ as it is independent of θ, ϕ, and doesn't affect the maximization problem.

4.b.1 Estimating the first term

We use sampling to estimate the first term. Since

$$\int_{\vec{z}} q_{\vec{\phi}}(\vec{z}|\vec{x_i}) d\vec{z} \;=\; 1 \quad \text{and} \quad \forall \vec{z} \ \ q_{\vec{\phi}}(\vec{z}|\vec{x_i}) \geq 0$$

We can say that the first term resembles a probability distribution of the variable $\vec{z}$ given $\vec{x_i}$.

Using the expected value of a function of the variable

$$E_X[f(X)] = \int_{domain(X)} f(x) dx$$

the first term simplifies to

$$E_{q_{\vec{\phi}}(\vec{z}|\vec{x_i})} \left[\log P_{\vec{\theta}}(\vec{x_i}|\vec{z_j}) \right]$$

We now **approximate** this expectation as the average of $P_{\vec{\theta}}(\vec{x_i}|\vec{z})$ over r different samples of $\vec{z} \sim q_{\vec{\phi}}(\vec{z}|\vec{x_i})$:

$$E_{q_{\vec{\phi}}(\vec{z}|\vec{x_i})} \left[\log P_{\vec{\theta}}(\vec{x_i}|\vec{z}) \right] \;=\; \frac{1}{r} \sum_{j=1}^{r} \log P_{\vec{\theta}}(\vec{x_i}|\vec{z})$$

How do we sample $\vec{z}$ from $q_{\vec{\phi}}$?

Assume:

$$q_{\vec{\phi}}(\vec{z}|\vec{x_i}) = \prod_k q_{\vec{\phi}}(z_k|\vec{x_i}) \qquad \text{where} \quad q_{\vec{\phi}}(z_k|\vec{x_i}) \sim \mathcal{N}\left(\mu_{z_k|\vec{x_i}}, \sigma^2_{z_k|\vec{x_i}} \right)$$

In order to get the desired r samples of $\vec{z} \sim q_{\vec{\phi}}$ from this (multivariable) Gaussian distribution, we perform **re-parameterization**: Sample $v_{jk} \sim \mathcal{N}(0, 1)$ from the standard parameterless Gaussian distribution (mean 0, variance 1). Then, we express

$$z_{jk} = \mu_{z_k|\vec{x_i}} + v_{jk} \sigma_{z_k|\vec{x_i}} \implies \vec{z_j} = \vec{\mu}_{\vec{z}|\vec{x_i}} + \vec{v_j} \mathbf{\Sigma}_{\vec{z}|\vec{x_i}} \implies \vec{z_j} \in \mathcal{N}\left(\vec{\mu}_{\vec{z}|\vec{x_i}}, \mathbf{\Sigma}_{\vec{z}|\vec{x_i}} \right)$$

Therefore, the first term is simplified to

$$\sum_{i=1}^{N} \frac{1}{r} \sum_{j=1;\ v_{jk}\sim\mathcal{N}(0,1)}^{r} \log P_\theta\left(\vec{x_i} \mid \vec{\mu}_{\vec{z}|\vec{x_i}} + \vec{v_j}\boldsymbol{\Sigma}_{\vec{z}|\vec{x_i}}\right)$$

For this purpose, we assume an 'encoder' neural network $q_{\vec{\phi}}$, which accepts $\vec{x}$ as input and outputs $\vec{\mu}$ and $\boldsymbol{\Sigma}$.

4.b.2 Simplifying the second term

The second term (integral) is the KL distance between $q_{\vec{\phi}}(\vec{z}|\vec{x_i})$ and $P_{\vec{\theta}}(\vec{z})$.

Under the (already known) assumptions

$$P(\vec{z}) \sim \mathcal{N}(0,1)$$

$$q_{\vec{\phi}}(\vec{z}|\vec{x_i}) \sim \mathcal{N}\left(\vec{\mu_i}(\vec{x}), \boldsymbol{\Sigma}_\mathbf{i}(\vec{x})\right)$$

We have,

$$\int_{\vec{z}} \left(q_{\vec{\phi}}(\vec{z}|\vec{x_i}) \log \frac{q_{\vec{\phi}}(\vec{z}|\vec{x_i})}{P_{\vec{\theta}}(\vec{z})} \right) d\vec{z} \; = \; \int_{\vec{z}} \mathcal{N}\left(\vec{\mu_i}, \boldsymbol{\Sigma}_\mathbf{i}\right) \cdot \frac{1}{2} \left(-\frac{\|\vec{z}-\vec{\mu_i}\|_2^2}{|\boldsymbol{\Sigma}_\mathbf{i}|} + \|\vec{z}\|_2^2 - \log|\boldsymbol{\Sigma}_\mathbf{i}| \right) d\vec{z}$$

This evaluates to

$$\frac{1}{2}\left(\|\vec{\mu_i}\|_2^2 + |\boldsymbol{\Sigma}_\mathbf{i}| - 1 - \log|\boldsymbol{\Sigma}_\mathbf{i}| \right)$$

4.b.3 Putting it all together

$$\vec{\theta}_{ML}, \vec{\phi}_{ML} = \arg\max_{\vec{\theta},\vec{\phi}} \sum_{i=1}^{n} \left(\frac{1}{r} \sum_{j=1}^{r} \log P_{\vec{\theta}} \underbrace{(\vec{x_i} \mid \vec{\mu}(\vec{x_i}) + \vec{v_j}\boldsymbol{\Sigma}(\vec{x_i}))}_{\vec{v_j}\ :\ v_{j,k}\sim N(0,1)} \; - \; \frac{1}{2}\left(\|\vec{\mu}(\vec{x_i})\|_2^2 + |\boldsymbol{\Sigma}(\vec{x_i})| - 1 - \log|\boldsymbol{\Sigma}(\vec{x_i})| \right) \right)$$

where $P_{\vec{\theta}}(\vec{x}|\vec{z}) = \mathcal{N}\left(\vec{\mu}(\vec{z}), \boldsymbol{\Sigma}(\vec{z})\right)(\vec{x})$.
Here, $\vec{\mu}(\vec{z}), \boldsymbol{\Sigma}(\vec{z})$ are generated from the 'decoder' network $F_{\vec{\theta}}$, and $\vec{\mu}(\vec{x}), \boldsymbol{\Sigma}(\vec{x})$ are generated from the 'encoder' network $q_{\vec{\phi}}$.

4.c Training the Model: The Algorithm

We train $\vec{\theta}$ and $\vec{\phi}$ for the two networks. The overall algorithm uses batch gradient descent.

We start by initializing them **randomly**.

We train for a set number of iterations T. In each iteration t, we take a **mini-batch** of B data points. Without loss of generality, let the data points be $\vec{x_1}, \vec{x_2}, \ldots \vec{x_B}$.
In each iteration:

1. For $i \in \{1, 2, \ldots B\}$, compute $\vec{\mu}(\vec{x_i})$ and $\boldsymbol{\Sigma}(\vec{x_i})$ using the encoder $q_{\vec{\phi}}$.
2. Get Br samples of $\vec{v} : v_{i,j,k} \sim N(0,1)$, and $\vec{v_{i,j}}$ is a k_0-dimensional vector.
3. Compute $\vec{z_{i,j}} = \vec{\mu}(\vec{x_i}) + \vec{v_{i,j}}\boldsymbol{\Sigma}(\vec{x_i})$ for all Br values of (i,j).
4. For all Br values of (i,j) compute $\vec{\mu}(\vec{z_{i,j}})$ and $\boldsymbol{\Sigma}(\vec{z_{i,j}})$ from the decoder $F_{\vec{\theta}}$.
5. Compute the loss function:

$$loss\left(\vec{\theta}, \vec{\phi}\right) = \sum_{i=1}^{B} \left(-\frac{1}{r} \sum_{j=1}^{r} \left(\log N\big(\vec{\mu}(\vec{z_i}), \boldsymbol{\Sigma}(\vec{z_{i,j}})\big)(\vec{z_{i,j}}) \right) + \|\vec{\mu}(\vec{x_i})\|^2 + |\boldsymbol{\Sigma}(\vec{x_i})| - \log|\boldsymbol{\Sigma}(\vec{x_i})| \right)$$

6. Perform the gradient descent update using the derivative of the loss with respect to $\vec{\theta}$ and $\vec{\phi}$

4.d Limitation of VAE

Training $\vec{\theta}$ and $\vec{\phi}$ for the encoder and decoder separately is difficult, and the resultant quality is poor.