0.1 More complete

Variational inference

Problem

Family of techniques for approximating intractable integrals arising in Bayesian inference. Usually situations of modeling latent variables. (Why?)

Approximation of posterior $P(Z|X)$ over a set of unobserved variables $Z = (Z_1, \ldots, Z_n)$ given observables X using a so-called variational distribution $Q(Z)$: $Q(Z) \approx P(Z|X)$

$Q(Z)$ is a family of distributions of simpler form than $P(Z|X)$ (eg. a family of Gaussian distributions), selected with the intention of selecting $q^* \in Q$ s.t. q^* is optimally similar to the true posterior $P(Z|X)$. This is achieved by minimizing a dissimilarity function $d(Q; P)$ for parametrized $Q(Z; \Phi)$, wherein $\phi^* \in \Phi$ is found s.t. $q^*(Z|\phi^*) \approx P(Z|X)$.

KL divergence

KL-divergence (Kullback-Leibler) is a common choice for $d(Q; P)$:

$$D_{KL}(Q||P) \coloneqq \sum_Z Q(Z) log \frac{Q(Z)}{P(Z|X)}$$

- Another understanding is to think of GMM (Gaussian mixture models). GMMs are weighted sums of Gaussian pdfs, however in practice, with a user-fixed number of basis functions K: $\pi_K(x) = \sum_i^K w_i \mathcal{N}(x|\mu_i, \sigma_i^2)$. A variational approach would be to optimize the number of basis components K^*. (Not sure if this is correct perspective)
- c.f. mean-field approximation: $Q(Z)$ is usually factorized into a partition over $Z = (Z_1, \ldots, Z_m)$: $Q(Z) = \prod_i^M q_i(Z_i|X)$. It can be shown using calculus of variations that the optimal distribution q_i^* for each of the factors q_i, in terms of minimizing the KL-divergence (for what exactly?) is an expression ... (record). Basically, it's the log-likelihood term I saw in VAE loss function, and reading this, it's clear why it's called mean-field.

Likelihood function

Definition

$\mathcal{L}(\theta; X)$ is viewed as a function over parameters rather than data. A typical view of probability functions is as a function over data with fixed parameters. Indeed the fixedness of the parameters - the parametrization of a distribution - makes for a given distribution generating data.

MLE

Example - MLE of a sample of Binomial variables

$X_1, \ldots, X_N$ sample with $X_i \sim \mathcal{D}(\theta)$ (here $\mathcal{B}(n,p)$)

$$\mathcal{L}(\theta; X) = \prod_i^N p_\theta(x_i)$$

$$\max_\theta \mathcal{L}(\theta; X) = \max_\theta \log p(\theta; X)$$

$$\text{often } \frac{\partial}{\partial\theta} \log p(\theta; X) := 0$$

$$\mathcal{L}(\theta; X) = \prod_i^N \binom{n}{x} p^{x_i}(1-p)^{(n-x_i)}$$

$$\log \mathcal{L}(\theta; X) = \log \prod_i^N \binom{n}{x} p^{x_i}(1-p)^{(n-x_i)} = \sum_i^N \log \binom{n}{x} p^{x_i}(1-p)^{(n-x_i)}$$

$$= \sum_i^N (\log \binom{n}{x} + x_i \log p + (n-x_i)\log(1-p))$$

$$0 := \frac{\partial}{\partial\theta} \left(N \log \binom{n}{x} + \sum_i^N x_i \log p + (n-x_i)\log(1-p)) \right)$$

$$0 := \sum_i^N \frac{x_i}{p} - \frac{(n-x_i)}{(1-p)}$$

$$\boxed{\hat{p}_{MLE} = \sum_i^N \frac{x_i}{nN}}$$

* e.g. generated binomial sample from python: `np.random.binomial(n, p, size=N)`, with `n=5`, `p=0.3`, `N=20`, s.t. we have 20 binomial RVs with $X_i \sim \mathcal{B}(n,p)$. This generated `[2,2,3,1,1,2,2,2,3,2,2,1,0,1,3,1,1,0,0,1]`. $\hat{p}_{MLE} = 0.28$ in this case, very close to $p_{true} = 0.3$.

* Computationally, we see MLE in practice, further illustrating how this works:

  Generate a binomial sample $X_1, \ldots, X_{20}$, w/ p unknown parameter to be inferred from geenerated data ($p = 0.3$ here f.ex.) and n=5 known parameter (is there a way to do MLE/estimation over multiple unknown parameters?)

  `X = np.random.binomial(5, 0.3, 20)` (n, p, N)

  `bin = lambda x, p, n:  factorial_choice(n, x)*p**x*(1-p)**(n-x)`

  `P = np.linspace(0, 1, 20)`

```
pdfs = [bin(X, p, 5) for p in P]

L_X = np.prod(pdfs_X, axis=1)

for row, p in zip(L_X, P): ; print(row, p)
```

```
8.662790562900144e-25 0.05263157894736842
8.03624640569313e-18 0.10526315789473684
1.3878582387533934e-14 0.15789473684210525
5.96388271219428e-13 0.21052631578947367
2.8742901812567957e-12 0.2631578947368421
2.9489962977318594e-12 0.3157894736842105
8.770444046689676e-13 0.3684210526315789
8.745813689574647e-14 0.42105263157894735
3.0643807546084483e-15 0.47368421052631576
3.6688345109700304e-17 0.5263157894736842
1.358585260115384e-19 0.5789473684210527
1.2927868586939663e-22 0.631578947368421
2.324908934955231e-26 0.6842105263157894
4.763139432910712e-31 0.7368421052631579
4.636773153938482e-37 0.7894736842105263
4.0588082506887503e-45 0.8421052631578947
7.403862837914546e-57 0.894736842105263
1.6451768904419834e-77 0.9473684210526315
```

Key takeaway: MLE is a fitting of a family of distributions (i.e. $\mathcal{D}(\theta)$ for known $\mathcal{D}$ and (at least/most?) one unknown $\theta \in \Theta$, *for given data* X. $\mathcal{L}(\theta; X)$ is a function where data is treated as fixed and θ is variable; $\mathcal{L}(\theta)$ is maximized - as is $\log \mathcal{L}$, often easier to work with in deriving $\hat{\theta}$ - for $\min_{\hat{\theta}} d(\theta_{true}, \hat{\theta})$ i.e. for θ very close to θ_{true}. Now the question is, why is likelihood function maximized for this situation?

'Baby Bayes': Bayesian classifier example, illustrating some concepts
$\Rightarrow$ Goal = introduce how variational inference + NNs = powerful image generation?

- Dataset used is MNIST (thru `keras`) handwritten digits: $X_i = \{0, \ldots, 9\}$, 28×28 images; $X = N \times 28 \times 28$
- Algorithm summary:
  - for each class $y_k \in Y$, model $P(X|Y)$ and use this to model posterior $P(Y|X)$ rather than modeling $P(Y|X)$ directly.
  - To model $P(X|Y)$, assume a class-specific Gaussian to get $(\mu^{(k)}, \sigma^{2(k)})$ for each y_k - i.e. calculate the parameters and use to generate data. This is a first example of generative modeling.

  - With $P(X|Y)$, calculate $P(X), P(Y)$ and use $P(y_k|x_i) = \dfrac{P(x_i|y_k)P(y_k)}{P(x_i)}$ to get predictions $P(Y|x_i)$.

- Implementation details:
  - X = $N \times 28 \times 28$
  - I chose to model each pixel within a class $X_i^{(k)}$ as a separate random variable, so $X_i^{(k)} \sim \mathcal{N}^{(k)}(\mu_i, \sigma_i^2)$.
  - For each y_k, calculated $\mu^{(k)} = (\mu_1, \ldots, \mu_{784})^{(k)}$ and $\mathbb{V}[X]^{(k)} = (\mathbb{V}[X_1], \ldots, \mathbb{V}[X_{728}])^{(k)}$ and input these as parameters into $\mathcal{N}^{(k)}$. Thus we get $P(X|y_k) = \mathcal{N}^{(k)}(\mu^{(k)}, \sigma^{2(k)})$ for all y_k.
  - $P(Y)$ is $\frac{\#y_k}{\#Y}$; the approach I took for $P(X)$ was to rebin data into $\tilde{x_i} \in \mathbb{Z}_{255}$ (i.e. round the data) and calculate $P(\tilde{x}_i), \forall i \in \{0, \ldots, 255\}$, as an approximation for a given $P(x_i)$.

Neural networks fundamentals

- NNs at the simplest are decision boundaries formed by linear hyperplanes.
- The next layer of complexity is nonlinear activation functions applied to these linear hyperplanes.
- Below is a sketch of deriving the optimal weightings for a linear classifier
  - The model is a transformation of $x^n \in X \mapsto y(x^n) \in c$ using a linear function: $y_k(x^n; w) = w_k^T x^n + w_0$
  - Here, we have basis functions $\phi_j^n = \phi_j(x^n)$ transforming $X = \{x^n\}_n^N$ input features into an output space of dimension c: $y(x; w) = \sum_n^N \sum_k^c \sum_j^M w_{kj}\phi_j^n$
  - Deriving the optimal weight vector w using ordinary least squares error function:
    - $E(w) = \dfrac{1}{2}\sum_n^N \sum_k^c (\sum_j^M w_{kj}\phi_j^n - t_k^n)^2$
    - $= \dfrac{1}{2}\sum_n \sum_k (w_{k1}\phi_1^n + \ldots + w_{kM}\phi_M^n - t_k^n)^2$
    - $= \dfrac{1}{2}\sum_k (w_{k1}\phi_1^1 + \ldots + w_{kM}\phi_M^N - (t_k^1 + \ldots t_k^N))$
    - $\dfrac{\partial E}{\partial w_k} = \dfrac{\partial}{\partial w_k}(\dfrac{1}{2}\sum_n^N (w_{k1}\phi_1^n + \ldots + w_{kM}\phi_M^n - t_k^n)^2) := 0$

- $= \nabla_{w_k}(\ldots) = \sum_n (\sum_j w_{kj}\phi_j^n - t_k^n)\nabla_{w_k}(\sum_j w_{kj}\phi_j^n - t_k^n)$
- $= \sum_n (\sum_j w_{kj}\phi_j^n - t_k^n)(\sum_j \phi_j^n) := 0$
- $\{0 = (y_k(x) - t_k)\phi(x), \forall k\} \implies$ convert this solution to matrix form

- I implemented both a closed form solution and gradient-descent solution for w^*, applied to the iris dataset, which is a $(150, 4)$ dimension feature-space (sepal lengths/widths, petal lengths/widths $\times$ classes). Starting w/ the closed form solution, if I were to complete the above calculation, I might arrive at $\Phi^\dagger = (\Phi^T\Phi)^{-1}\Phi^T$ for $W^T = \Phi^\dagger T$, and would use this projection vector/matrix W^T on $x \in X$ to output classifications. This is achieved by deciding on a labeling scheme for the labels $t \in T$. There are 3 classes (species) in the iris dataset, so I used:

$$t = \begin{cases} -1 & \text{if } x \in C_1 \\ 0 & \text{if } x \in C_2 \\ -1 & \text{if } x \in C_3 \end{cases} \tag{1}$$

- Working out the dimensionality of the problem and projecting in my case a vector $w, dim(w) = (2, 1)$, I got the following average projections when applied to a test set of $x_{C_1}, x_{C_2}, x_{C_3}$: , $\langle w^T x_{C_1}\rangle = -0.7769$, $\langle w^T x_{C_2}\rangle = 0.2176$, $\langle w^T x_{C_3}\rangle = 0.3407$, the idea ostensibly being for $y(x_{C_k})$ to approach the value for t_{C_k}.
- Gradient descent solution: $w_{kj}^{r+1} = w_{kj}^r - \eta\frac{\partial E}{\partial w_{kj}}$. Write out notes sometime, but I got $= \sum_n (x^{nT}(w_k^T x^n - t_k^n))$, which I then viewed as $(\sum_n x_1^n(\ldots), \ldots, \sum_n x_4^n(\ldots))$. With the same labeling scheme for t, I set $\eta = 10^-3$ (too high), then $\eta = 10^-5$ (seemed to work a lot better). Started out with $w^0 = (0, 0, 0, 0)^T$ and used an expression I derived `w_k = w_k0 - eta*(X.T@(X@w_k0 - t))`. Here, the mean projections seem to do a bit better: $\langle w^T x_{C_1}\rangle = -1.1055$, $\langle w^T x_{C_2}\rangle = 0.2613$, $\langle w^T x_{C_3}\rangle = 0.7376$. This was over 10^5 weight update iterations. The gradient descent scheme was of course a lot more satisfying to figure out and implement.

0.2 Drafts

Self-information

Mutual information