Applying Boosting Techniques to the training of RBMs and VAEs

Lillian Huang, Jamie Matthews, Jessica Thompson

April 15, 2019

Abstract

Boosting algorithms have shown much success in the realm of supervised learning. As a natural next step, various papers have presented boosting-style algorithms for the unsupervised problem of density estimation [8] [9] [2]. However, they only learn simple models. In this paper, we apply the techniques used in [2] to the training of more complicated generative models, including restricted Boltzmann machines (RBMs) and variational autoencoders (VAEs).

1 Introduction

Boosting is an important component of machine learning. Commonly used for supervised learning, it has the power to take several weak learners and combine them into a strong learner. Boosting, specifically in regards to the original algorithm, AdaBoost [3], has seen much success in the realm of supervised learning; however, it has also shown promise when applied to unsupervised learning. Specifically, there are several papers that explore boosting in density estimation, and more recently, generative models.

1.1 Density Estimation

Some of the first uses of unsupervised boosting were for the problem of density estimation [8] [9]. The goal of this is to estimate the underlying probability density function of a given set of observed data. A commonly applied empirical principle with respect to density estimation is the idea of maximum entropy. The principle states that for all distributions that may encode the sample data, the distribution that has the maximal information entropy (closest to uniform) would be the best choice. Thus, the target distribution is estimated by the choice with maximum entropy that still satisfies all the given constraints. Often these constraints are represented by real-valued functions, called *features*, where each feature's theoretical expectation must match the empirical average.

1.2 Maximum Entropy Algorithm

As we've mentioned, it is a commonly held belief that, heuristically, the distribution with the highest entropy is the best distribution to describe a set of data. The goal of the algorithm presented in [2] is to find such a distribution. The authors show that the dual of this optimization problem is the problem of finding the Gibbs distribution which maximizes the log-likelihood, meaning it minimizes the log-loss. We have:

$$\mathcal{P}: \quad \max_{p \in \Delta} \mathrm{H}(p) \text{ subject to } p[f_j] = \tilde{\pi}[f_j] \qquad \mathcal{Q}: \quad \min_{\boldsymbol{\lambda} \in \mathbb{R}^n} \mathrm{L}_{\tilde{\pi}}(\boldsymbol{\lambda})$$

where $H(p)$ is the information entropy of an arbitrary probability distribution, and the f_j's are the features of the data. $\tilde{\pi}[f_i]$ is the expectation of a specific feature, with respect to the training data, and $L_\pi(\lambda)$ is the log-loss of the Gibbs distribution described by λ, again with respect to the training data.

The novelty of this paper is how the algorithm specifically learns this distribution. Rather than simply computing the Gibbs distribution which best fits the training data, which can be done analytically, it uses a boosting-style approach. It can be shown that the exact distribution tends to overfit the data. On the other hand, the numberically-found distribution tends to generalize better.

In each step of the algorithm for each coordinate of the training data, the best δ is found. When this δ is added to the corresponding coordinate of λ, a variable that characterizes the Gibbs distribution, this minimizes the log-loss. The authors describe a sequential algorithm which updates the coordinate with the lowest minimum, and a parallel algorithm which updates all of the coordinates individually at each iteration.

Input: Finite domain X
features $f_1, \ldots, f_n$ where $f_j : X \to [0, 1]$
examples $x_1, \ldots, x_m \in X$
nonnegative regularization parameters $\beta_1, \ldots, \beta_n$
Output: $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots$ minimizing $\mathrm{L}^{\boldsymbol{\beta}}_{\tilde{\pi}}(\boldsymbol{\lambda})$
Let $\boldsymbol{\lambda}_1 = \mathbf{0}$
For $t = 1, 2, \ldots$:
- let $(j, \delta) = \arg\min_{(j,\delta)} F_j(\boldsymbol{\lambda}_t, \delta)$
  where $F_j(\boldsymbol{\lambda}, \delta)$ is the expression appearing in Eq. (12)
- $\lambda_{t+1,j'} = \begin{cases} \lambda_{t,j} + \delta & \text{if } j' = j \\ \lambda_{t,j'} & \text{else} \end{cases}$

The authors show that the above δ, which minimizes a specific, constructed function related to

the log-loss, can be found analytically. Specifically, suppose δ is added to λ_j to give $\lambda'_J = \lambda_j + \delta$. Then, it can be shown that,

$$L^{\beta}_{\tilde{\pi}}(\lambda') - L^{\beta}_{\tilde{\pi}}(\lambda) \leq -\delta\tilde{\pi}[f_j] + \ln(1 + (e^{\delta} - 1)q_{\lambda}[f_j] + \beta_j(|\lambda_j + \delta| - |\lambda_j|)$$

where $q_{\lambda}[f_j]$ is the expectation of f_j with respect to the current Gibbs distribution model with parameter λ, and β is a hyper-parameter. Then, the best δ for λ_j must be

$$\ln\left(\frac{(\tilde{\pi}[f_j] \pm \beta_j)(1 - q_{\lambda}[f_j])}{(1 - \tilde{\pi}[f_j] \pm \beta_j)q_{\lambda}[f_j]}\right)$$

where the $\pm$ operation enforces that the value inside the log stays positive.

The authors then prove that the log-loss of the sequence of λ's, and thus, the log loss of the Gibbs distribution model produced at each iteration, decreases each step. Specifically, if all the β_j's are strictly positive, then,

$$\lim_{t\to\infty} L^{\beta}_{\tilde{\pi}}(\lambda_t) = \min_{\lambda} L^{\beta}_{\tilde{\pi}}(\lambda)$$

where λ_t is the λ produced by iteration t of the algorithm.

2 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are a special type of Markov random fields. They are shallow neural networks that contain only two layers—one visible and one hidden. There is no communication of information within neurons in a given layer. In other words, the model is a bipartite graph, hence the restriction in their name. Applications of RBMs include feature learning and dimensionality reduction, collaborative filtering, topic modelling, and more recently, many-body quantum mechanics. They are also used as components in deep belief networks.

There are several training algorithms for RBMs, each with its own benefits. The most commonly used algorithm is contrastive divergence, or CD [5]. Like many algorithms, the idea is to minimize the log-loss of the model with respect to the training data. The probability that the network assigns the visible layer a specific value, v, is given by

$$p(v) = \frac{1}{Z}\sum_{h} e^{-E(v,h)}$$

where the sum is taken over all possible values of the hidden layer, and Z is the partition function, i.e. a normalization constant. The energy configuration is

$$E(v,h) = -\sum_{i\in \text{ visible}} a_i v_i - \sum_{j\in \text{ hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

In essence, the algorithm uses Gibbs sampling to sample first the hidden layer, then the visible layer. The results describe the current distribution which are compared to the ideal distribution, using a similar idea to KL divergences. This is then used in a gradient ascent-like step to compute the weight updates, similar to error back-propagation in a neural network.

More specifically, a random training data point is first fed into the visible layer where each neuron's data is multiplied by various weights and added to a bias; these values are then passed into an activation function which results in the hidden neuron values. The outer product of the visible layer input, v, and the hidden layer output, h, is then computed. This value is called the positive gradient. The values of these hidden neurons are then passed backwards, where they are multiplied by the same weights as in the forward pass and added to a different bias. The output of this step is a reconstruction, v', of the original input. The first step is repeat with v' as input, to get another sample from the hidden layer, $'h$. Again, the outer product of these values are taken. This value is called the negative gradients. The weights and biases are updated as follows:

$$\Delta W = \epsilon(vh^T + v'h'^T) \quad \Delta a = \epsilon(v - v') \quad \Delta b = \epsilon(h - h')$$

where ϵ is a hyper-parameter similar to a step size.

2.1 Boosting Approach for RBM Training

To train an RBM model, the contrastive divergence algorithm seeks to maximize the log-likelihood for each training data point, v. The log-likelihood with a regularization term is given by

$$\ln p(v) = \ln \sum_h e^{-E(v,h)} - \ln Z + \sum_k \beta_k |\theta_k|$$

where θ represents the weights and biases of the model.

In a similar fashion as the Max Entropy boosting algorithm, we could find δ's which, when added to a specific parameter, maximizes the increase in this function. One hurdle to this calculation, however, is the sum over all possible hidden variable values. One way to potentially mitigate this is to compute the value of the corresponding hidden variables using the weights and the biases for each training data point.

Specifically, suppose δ is added to w_{ij} to give $w'_{ij} = w_{ij} + \delta$. Then, it can be shown that

$$\begin{aligned} L^{\beta}_{\tilde{\pi}}(w_{ij} + \delta) - L^{\beta}_{\tilde{\pi}}(w_{ij}) &\leq -\delta\tilde{\pi}[v_i] + \ln(1 + (e^{\delta} - 1)q_{\theta}[v_i h_j] + \beta_{ij}(|w_{ij} + \delta| - |w_{ij}|) \\ &= F(ij, \delta) \end{aligned}$$

Then, the best δ for w_{ij} must be

$$\ln\left(\frac{(\tilde{\pi}[v_i] \pm \beta_{ij})(1 - q_\theta[v_i h_j])}{(1 - \tilde{\pi}[v_i] \pm \beta_{ij}) q_\theta[v_i h_j]}\right)$$

A similar approach could be used to boost the biases for the visible and hidden layers.

3 Variational Autoencoders

One of the more popular latent variable models is a variational autoencoder, or a VAE. VAEs aim to produce "realistic" fake data by trying to learn a distribution of the latent variables $z^{(i)}$ such that it maximizes the probability of the input data $x^{(i)}$. It assumes a standard Gaussian prior for the distribution of $z^{(i)}$ and slowly adjusts a "decoder" function g_θ that maps $z^{(i)}$ to $x^{(i)}$ in order to maximize $P(x^{(i)})$. We want "decoder" model parameters θ that maximize the likelihood of observing $x^{(i)}$, so in math terms we are looking for:

$$\text{argmax}_\theta \prod_{i=1}^{m} P(x^{(i)}; \theta) = \text{argmax}_\theta \sum_{i=1}^{m} \log P(x^{(i)}; \theta)$$

Unfortunately, we cannot solve for this in closed form, so we use a trick: we find the maximum of the variational lower bound instead. Using Bayes' Rule, we know that

$$\sum_{i=1}^{m} \log P(x^{(i)}; \theta) = \frac{P(z|x^{(i)})P(x^{(i)})}{P(z)}$$

With some algebra, this ultimately gives us

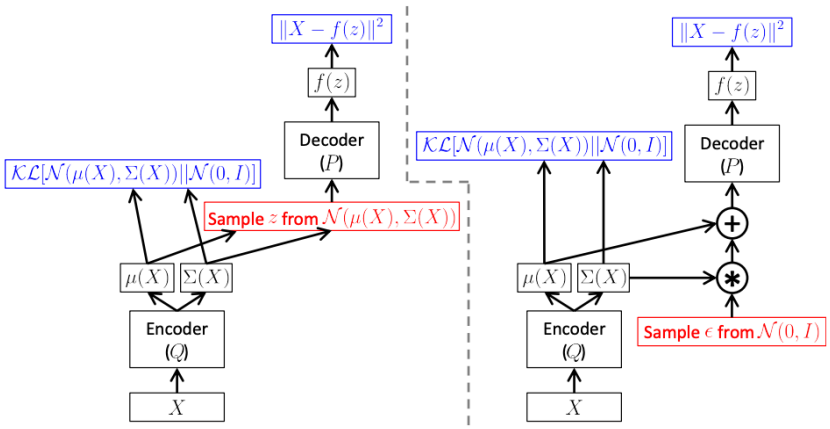
$$\log P(x^{(i)}) \geq \mathbb{E}_{z \sim Q_i}[\log P(x^{(i)}|z)] - \text{KL}(Q_i(z)||P(z))$$

where KL represents the KL divergence between the two distributions and Q is the prior we give to the latent variables $z^{(i)}$, which we choose to be a Gaussian for convenience. The right side of the equation above gives us the so-called variational lower bound, also known as the ELBO bound, and we can maximize this value instead of $P(x^{(i)})$. It's possible to find a closed-form solution for the ELBO, and thus we get

$$\log P(x^{(i)}) \geq \max_{\theta,\phi} \sum_{i=1}^{m} \mathbb{E}_{z \sim Q}\big[- \|x^{(i)} - g_\theta(z)\|^2 - \|z - f_\phi(x^{(i)})\|^2 - \|z\|^2\big]$$

where f_ϕ is the "encoder" function that maps $x^{(i)}$ values to latent space, and ϕ represents the parameters of this function. Using the variational lower bound, it possible to do gradient descent to optimize the network.

A full illustration of the VAE structure and loss functions are shown in the figure below. Note that the version on the right utilizes a "reparameterization" trick that allows the loss to be fully propagated back through the entire network. In the formulation described above, it is necessary to sample latent vectors from $Q(z|x)$, which is not differentiable and thus difficult to train. Instead, one can sample some small $\epsilon \sim N(0, I)$ and construct $z = \mu(X) + \epsilon * \Sigma(X)^{1/2}$, where μ, Σ were learned from Q to find parameters for an accurate Gaussian for the latent space. In this way, we make the VAE easier to train, and this trick also reduces the variance of gradient estimates. However, this only works if we haves a continuous latent distribution.



3.1 Discrete-Variable VAEs

VAEs with discrete latent distributions can be important for a number of real-life applications. However, as we've discussed above, it is not possible to do the traditional reparameterization trick with discrete-variable VAEs. Instead, [6] reparameterizes such models by using the Gumbel-Max perturbation model. When dealing with discrete-variable VAEs, the model distribution $p_\theta(x|z)$ follows the Gibbs distribution. The Gibbs distribution law is shown in [6] to be equivalent to Gumbel-Max perturbation models, and thus we can write the latent distribution in an alternative representation. Originally, we want

$$-\mathbb{E}_{z\sim q_\phi} \log p_\theta(x|z) = \sum_{z=1}^{k} \frac{e^{\phi(x,z)}}{\sum_{\hat{z}} e^{\phi(x,\hat{z})}} \theta(x, z)$$

Instead, the paper shows that

$$\sum_{\hat{z}} e^{\phi(x,\hat{z})} = P_{\gamma\sim g}[z^{\phi+\gamma} = z]$$

where $z^{\phi+\gamma} \equiv \text{argmax}_{\hat{z}=1,\ldots,k}\{\phi(x,\hat{z}) + \gamma(\hat{z})\}$, and where we can approximate

$$P_{\gamma\sim g}[z^{\phi+\gamma} = z] \approx \frac{e^{\phi(x,z)+\gamma(z)}}{\sum_{\hat{z}} e^{\phi(x,\hat{z})+\gamma(\hat{z})}}$$

This so-called "Gumbel-Softmax model" is smooth and thus a VAE with this reparameterized objective is easier to train. Ultimately, they show that they can directly optimize such a network with the following theorem:

Theorem 1. *Assume $\phi_\nu(x,z)$ is a smooth function of ν, where ν denotes the encoder parameters. Then*

$$\nabla_\nu \mathbb{E}_\gamma[\theta(x, z^{\phi_\nu+\gamma}) = \lim_{\epsilon\to 0} \frac{1}{\epsilon}\big(\mathbb{E}_\gamma[\nabla_\nu \phi_\nu(x, z^{\epsilon\theta+\phi_\nu+\gamma}) - \nabla_\nu \phi_\nu(x, z^{\phi_\nu+\gamma})]\big)$$

3.2 Boosting Approach to VAE training

Although they are not trained by trying to directly maximize likelihood, VAEs have a likelihood that we can attempt to modify in order to apply the Maximum Entropy boosting algorithm in training. Since the Maximum Entropy boosting algorithm finds the Gibbs distribution that minimizes the log loss, we wish to modify the VAEs current likelihood such that it becomes a problem of finding the Gibbs distribution that minimizes the log loss (or maximizes the likelihood). In doing so, we will be able to use the same boosting algorithm to train VAEs. Recall that the likelihood we want to maximize in a VAE is:

$$\mathbb{E}_{z\sim Q_i}[\log P(x^{(i)}|z)] - \text{KL}(Q_i(z)||P(z))$$

Although Q is typically a neural network that produces a multivariate Gaussian, the previous section showed that in discrete VAEs, Q can produce a Gibbs distribution. The first term in our likelihood ensures that the output distribution given the z's from Q resemble the input distribution. The second term usually serves to ensure that Q produces a distribution that is not too different from a multivariate Gaussian; however, since we are working with discrete VAEs, we would want Q to produce a distribution that is not too different from a Gibbs distribution. If we were to apply

the Maximum Entropy boosting algorithm, this KL divergence term will no longer be necessary. The boosting algorithm will force Q to be a Gibbs distribution by nature, thus we do not need to worry about this divergence term. The likelihood that we are now trying to maximize would be:

$$\mathbb{E}_{z\sim Q_i}[\log P(x^{(i)}|z)]$$

Our problem is now to find a Q such that we are likely to produce our original distribution given our latent variables. This is similar to the problem set in the Maximum Entropy paper, as both aim to find a Gibbs distribution that will maximize a likelihood. However, this likelihood is also dependent on $P(x|z)$, the posterior, which we do not know. Thus, in order to compute our likelihood, we need to have an estimate for our posterior. We know from the previous section that discrete VAEs can be optimized, thus this problem is potentially solvable. However, we are unsure how to combine methods such as variational inference and the Maximum Entropy boosting algorithm, to achieve such an optimization.

4 Conclusion

In this paper, we applied the techniques used in [2] to the training of RBMs. Specifically, we constructed a boosting-style algorithm, in which the weights and biases of the model are iteratively adjusted. Furthermore, we analytically re-derive this value, δ, in the context of this problem. Future work would include constructing and proving a convergence theorem for the algorithm, similar to the one presented in the original paper. Moreover, our algorithm could be run for some training example; the results of which, could be compared to accuracy and efficiency of the contrastive divergence algorithm.

We also explored the possibility of applying the maximum entropy boosting techniques could be applied to the training of VAEs. We discussed benefits and problems associated with introducing Gibbs distributions in different ways into the standar VAE model. However, more work needs to be done into constructing an algorithm to train these modified models. Again, if an algorithm was derived, then it could be tested and compared to standard algorithms.

References

[1] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[2] M. Dudik, S. J. Phillips, and R. E. Schapire. Performance guarantees for regularized maximum entropy density estimation. In *International Conference on Computational Learning Theory*,

pages 472–486. Springer, 2004.

[3] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[4] A. Grover and S. Ermon. Boosted generative models. In *ICLR 2017 conference submission*, 2016.

[5] G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[6] G. Lorberbom, A. Gane, T. Jaakkola, and T. Hazan. Direct optimization through arg max for discrete variational auto-encoder. *arXiv preprint arXiv:1806.02867*, 2018.

[7] S. J. Phillips, M. Dudík, and R. E. Schapire. A maximum entropy approach to species distribution modeling. In *Proceedings of the twenty-first international conference on Machine learning*, page 83. ACM, 2004.

[8] S. Rosset and E. Segal. Boosting density estimation. In *Advances in Neural Information Processing Systems*, pages 657–664, 2003.

[9] M. Welling, R. S. Zemel, and G. E. Hinton. Self supervised boosting. In *Advances in neural information processing systems*, pages 681–688, 2003.