Proabilistic/Bayesian Models for Deep Learning

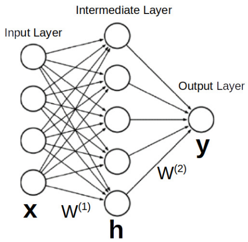
Piyush Rai

Topics in Probabilistic Modeling and Inference (CS698X)

April 12, 2018

Neural Networks

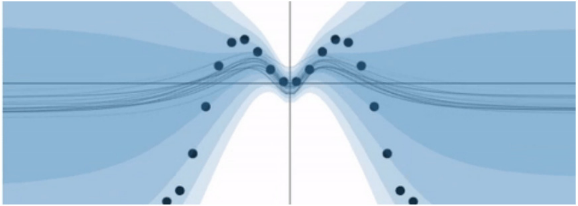
- A simple neural network with one intermediate (also called “hidden”) layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- In traditional neural nets, $\boldsymbol{h}$ is a Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid, ReLU, tanh, etc)
- Neural nets are awesome but brittle in many ways
  - Lots of parameters, difficult to train, need lots of data to train
  - Do not provide uncertainty estimates

What We Want..

- Neural networks with additional benefits of probabilistic/Bayesian modeling



- Basically, nonlinear models with estimates of uncertainty in the model/its predictions
- Note: We already have seen something that accomplishes this - Gaussian Processes
- Probabilistic/Bayesian neural nets are another alternative to this

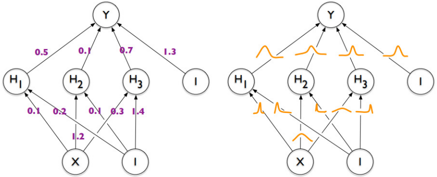
Neural Networks as Probabilistic Models

- A probabilistic model for neural network for supervised learning

$$
\begin{aligned}
y_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \beta^{-1}) && \text{(for real-valued responses)} \\
y_n &\sim \text{Bernoulli}(\sigma(\text{NN}(\boldsymbol{x}_n; \mathbf{W}))) && \text{(for binary responses)} \\
y_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{x}_n; \mathbf{W})) && \text{(for general types of responses modeled by exp-family)}
\end{aligned}
$$

where $\text{NN}(\boldsymbol{x}_n; \mathbf{W})$ is a neural network with features $\boldsymbol{x}_n$ as inputs and parameters $\mathbf{W}$

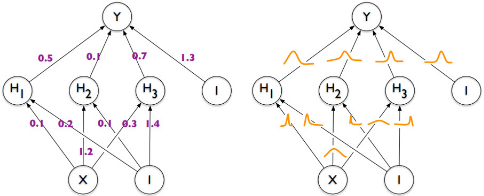
- This enables learning probabilistic nonlinear input-to-output mappings
- We can perform point estimation or fully Bayesian inference for such probabilistic neural networks



Left: Standard NN or NN with point estimation, Right: Bayesian Neural Network

"Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Learning Bayesian Neural Networks

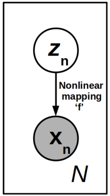


- Even if the prior $p(\mathbf{W})$ on NN weights is Gaussian, the model is not conjugate
- MCMC methods can be used to learn the posterior $p(\mathbf{W}|\mathcal{D})$ but can be slow
  - However, methods such as SGLD† allow efficient MCMC inference for such models (recall that SGLD only requires gradient expressions of the log-joint probability $\log p(\mathcal{D}, \mathbf{W})$ of the model)
- Variational inference is another popular alternative to MCMC for such models
  - But ELBO is intractable (due to non-conjugacy); has to be approximated via Monte-Carlo/BBVI
- Note: Hybrid architectures also possible (only last layer modeled in a fully Bayesian way)

† "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks" (Li et al, 2016)

Constructing Generative Models using Neural Nets

- Probabilistic view enables neural net based latent variable models ("deep generative models")
- Useful for unsupervised learning. Nonlinear latent variable to data mapping f modeled by NN



- Example: A probabilistic neural network for latent variable modeling (e.g., PPCA)

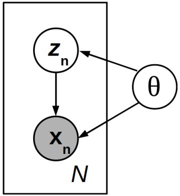
$$\boldsymbol{x}_n \sim \mathcal{N}(\text{NN}(\boldsymbol{z}_n; \mathbf{W}), \sigma^2 \mathbf{I}_D) \qquad \text{(for real-valued features)}$$
$$\boldsymbol{x}_n \sim \text{ExpFam}(\text{NN}(\boldsymbol{z}_n; \mathbf{W})) \qquad \text{(for general types of features modeled by exp-family)}$$

  where $\text{NN}(\boldsymbol{z}_n; \mathbf{W})$ is a neural network with latent variables $\boldsymbol{z}_n$ as inputs and and parameters $\mathbf{W}$

- The NN enables learning a nonlinear latent variable to data mapping f
- If $\boldsymbol{z}_n$ has a Gaussian prior, such models are called "Deep Latent Gaussian Models" (DLGM)

Inference for Deep Latent Gaussian Models

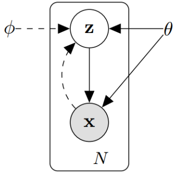
- Assume θ to be the global parameters of the model (params defining $p(\boldsymbol{z})$, $p(\boldsymbol{x}|\boldsymbol{z})$, etc.)



- The usual approach for inference in such models (as in most Bayesian models) is iterative, e.g.,
- Initialize θ. Then iterate until converence
  - For $n = 1, \ldots, N$
    - Infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ using MCMC. If doing VB, update variational parameters ϕ_n of $q(\boldsymbol{z}_n|\phi_n)$
  - Infer θ (its full posterior using MCMC or VB, or a point estimate)
- This iterative approach can be slow for large N
- Also, inferring $\boldsymbol{z}$ for new data point(s) $\boldsymbol{x}$ would require using the same iterative procedure

Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$
- Key idea: For each point $\boldsymbol{x}_n$, instead of learning a separate $q(\boldsymbol{z}_n|\phi_n)$ with local params ϕ_n, assume

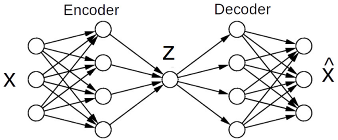
$$q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\text{NN}(\boldsymbol{x}_n; \phi))$$

  so, basically, each ϕ_n is computed by a neural net with global parameters ϕ and input $\boldsymbol{x}_n$

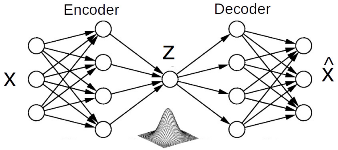
- Once ϕ is learned, we can get $q(\boldsymbol{z}_*|\boldsymbol{x}_*) = q(\boldsymbol{z}_*|\phi_*)$ for any $\boldsymbol{x}_*$ by just using $\phi_* = \text{NN}(\boldsymbol{x}_*; \phi)$
- $p(\boldsymbol{x}|\boldsymbol{z})$ is known as decoder and $q(\boldsymbol{z}|\boldsymbol{x})$ is known as encoder

Standard Auto-encoder vs Variational Auto-encoder

- A standard auto-encoder learns to (nonlinearly) compress and uncompress an input



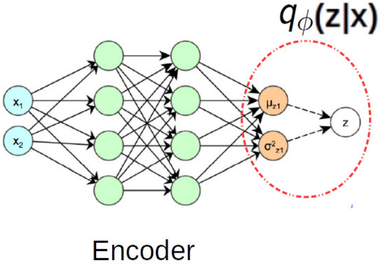
- Model is trained to minimize the reconstruction error (difference b/w $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$)
- However, it can't "generate" a "realistic" input from a random $\boldsymbol{z}$ (the model isn't trained for that)
- VAE allows this by assuming a distribution (e.g., Gaussian) over $\boldsymbol{z}$ and learning to generate $\boldsymbol{x}$ from random $\boldsymbol{z}$'s drawn from that distribution (so the model is trained to do this!)



- Note: Simple generative models like PPCA or factor analysis also have this ability to generate data from random $\boldsymbol{z}$ but the linear map from $\boldsymbol{z}$ to $\boldsymbol{x}$ limits the type of data that can be generated well

VAE: The Encoder

- Role of encoder: Take $\boldsymbol{x}$ as input and generate an encoding $\boldsymbol{z}$



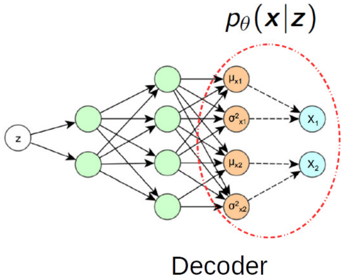
- Unlike standard autoencoders, for each $\boldsymbol{x}$, VAE gives us a distribution $q(\boldsymbol{z}|\boldsymbol{x})$ over its encoding
- Assume $q(\boldsymbol{z}|\boldsymbol{x})$ to be Gaussian whose mean/var are computed by a NN with global params ϕ

$$\mu_z = \text{NN}(\boldsymbol{x}; \phi) \qquad \sigma_z^2 = \text{NN}(\boldsymbol{x}; \phi)$$

- Since μ_z, σ_z are outputs of neural networks, the $\boldsymbol{x}$ to $\boldsymbol{z}$ mapping is nonlinear

VAE: The Decoder

- Role of decoder: Generate $\boldsymbol{x}$ given $\boldsymbol{z}$. Defined by the likelihood model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$



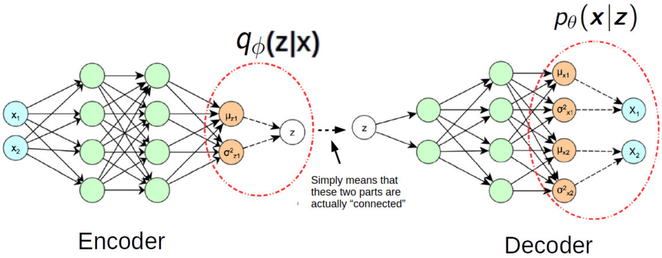
- Unlike PPCA, the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping is nonlinear (modeled by a neural network)
- Assume $p(\boldsymbol{x}|\boldsymbol{z})$ to be Gaussian whose mean/var are computed by a NN with global params θ

$$\mu_x = \text{NN}(\boldsymbol{z};\theta) \qquad \sigma_x^2 = \text{NN}(\boldsymbol{z};\theta)$$

- Thus in the VAE, both $\boldsymbol{x}$ to $\boldsymbol{z}$ (encoder) and $\boldsymbol{z}$ to $\boldsymbol{x}$ (decoder) mappings are nonlinear

Inference for VAE

- VAE uses variational inference (hence the name!) to learn the model parameters θ and ϕ



- Typically a prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ is assumed on $\boldsymbol{z}$. The ELBO for a single $\boldsymbol{x}_n$ will be

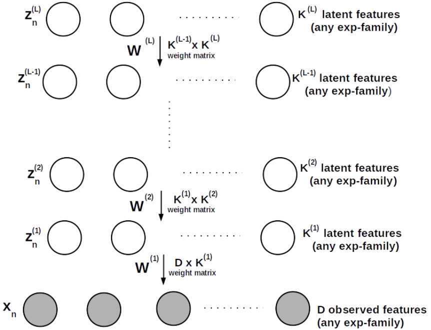
$$\text{ELBO} \quad = \quad \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \qquad \text{(note: } q_\phi \text{ and } q(\boldsymbol{z}_n|\boldsymbol{x}_n) \text{ mean the same)}$$

- Variational inference uses the reparametrization trick† for computing ELBO derivatives

† "Auto-encoding Variational Bayes" (Kingma and Welling, 2013)

Some Other Architectures: Deep Exponential Families

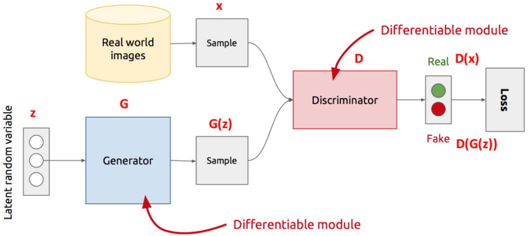
- Standard VAE has only one layer of latent $\boldsymbol{z}$ and a neural net to transform $\boldsymbol{z}$ into $\boldsymbol{x}$
- Many other deep architectures have multiple layers of latent variables
- Deep Exponential Family (DEF) is one such recently proposed popular model
- in DEF, latent variables in every layer, as well as observations, are from exp. family distributions



- Overall model not conjugate but BBVI (Ranganath et al, 2013) or MCMC methods can be used

Some Other Architectures: Generative Adversarial Networks

- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)



- Generator (a neural net) generates realistic looking "fake" data $\boldsymbol{x}$ from random $\boldsymbol{z}$
- Discriminator tries to detect fake data from real data
- At game's equilibrium, the $p_{gen} = p_{data}$ and success rate of discriminator $= 50\%$ (i.e., random)
- Originally designed mainly for synthetic data generation tasks but recent work extends GANs for many other problems such as latent variable inference, semi-supervised learning, etc.

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Learning the right size of a deep neural network

- How to decide the number of layers and width of each layer?
- Nonparametric Bayesian methods can help here

- A cascaded Indian Buffet Prior can model the relationships between nodes in adjacent layers
  - The bottom-most layer is the data layer (fixed/known size)
  - Width of each intermediate layer and active connections can be inferred by the IBP prior
- Another option is to use sparsity inducing priors on the connection weights

"Learning the Structure of Deep Sparse Graphical Models" (Adams et al, 2009), "Model Selection in Bayesian Neural Networks via Horseshoe Priors", Ghosh and Doshi-Velez (2017)

Neural Nets vs Gaussian Processes

- Both can be used learn nonlinear input to output mappings, e.g.,

$$\begin{aligned} y_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \beta^{-1}) \\ y_n &\sim \mathcal{N}(f(\boldsymbol{x}_n), \beta^{-1}) \qquad \text{where } f \sim \text{GP} \end{aligned}$$

- Both have their pros and cons
- NN pros: Fast to train (e.g., using SGD methods) and also fast at test time
- NN cons: Difficult to train, also not Bayesian (but can be made Bayesian)
- GP pros: Simple formulation, especially for regression settings; natively Bayesian in formulation
- GP cons: Slow to train and also slow at test time
- Nowadays Bayesian NN and GPs are competitive in many applications

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)
- Several other models such as Deep Boltzmann Machines, Neural Autoregressive Density Estimator, etc. that we didn't cover here
- An important distinction between explicit and implicit generative models
  - Models like PPCA, FA, DLGM, SBN, VAE, etc. have an explicit likelihood model for data
  - A model like GAN only defines $p(\boldsymbol{x})$ implicitly (no "likelihood" model for data)