10-708: Probabilistic Graphical Models 10-708, Spring 2017

18 : An Overview of Deep Learning building blocks

Lecturer: Maruan Al-Shedivat *Scribes: Lisa Lee, Chaoyang Wang*

1 An overview of the DL components

First we start with motivation: Why is DL so popular these days? There is a lot of investment in DL (e.g., DeepMind, DNNResearch, etc.). It's always in the news every month. And rightfully so, because DL has won numerous pattern recognition competitions using only minimal feature engineering. These models have been around since the 1960s, but there has recently been new tricks discovered to make DL models successful, such as: using more hidden units; better online optimization; new non-linear activation functions; etc.

McCulloch & Pitts (1943) came up with an artificial neuron (perceptron) which was inspired from biological neural networks. The real biological neural networks are a physical substrate, and communicate via voltage pulses (spikes). They do not use specific threshold values or work with real values. They are oscillating systems and can be described using systems of ODEs. However, "computation" performed by real neural networks is still not understood well. We only use spiking models for simulating brain activity, not for computing. There are various hypothesis of what exactly spiking networks compute, but it is not well understood.

Consider the perceptron learning algorithm. The sigmoid function satisfies $\sigma' = \sigma(1-\sigma)$.

$$w = \arg\min_{w} \sum_i \frac{1}{2}(y_i - \hat{f}(x_i; w))^2$$

The batch mode update is given by $w \leftarrow w - \eta \nabla E_D[w]$.

We can think of the perceptron as combined logistic regression models, where we have inputs, we regress them into intermediate, more high-level variables, and then output probabilities.

We use backpropagation, which is a reverse-mode differentiation, to train a neural network. Artificial neural networks are nothing more than complex functional compositions that can be represented by computation graphs.

By applying the chain rule and using reverse accumulation, we get

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x_{i_1}}$$

If functions are stochastic (e.g., f is not deterministic, but instead samples from some distribution), then we use stochastic backpropagation (to be covered in the next lecture).

(Auto-reverse-mode differentiation) One of the reasons DL is becoming popular these days is because there is a lot of engineering effort into packages that can automatically compute derivatives for a given computational graph. (E.g., TensorFlow, Caffe, Torch, etc.)

Here are some of the basic building blocks of deep networks:

- Activation functions: linear, ReLU, sigmoid, tanh, etc.
- Layers: fully connected, convolutional & pooling, recurrent (Problem: The graph is not acyclic. To compute the gradients, we unroll the computational cycle over timesteps, and backpropagate through this structure.), resnet (skip connections over layers), etc.

Putting everything together, we have arbitrary combinations of the basic building blocks of deep networks. Having multiple loss functions allows us to do multi-target prediction, transfer learning, and more. Given enough data, deeper architectures just keep improving.

Moreover, the networks learn increasingly more abstract representations of the data that are amenable to linear separation. This is inspired from signal processing in the brain, where our hypothesis is that the brain builds hierarchies of more and more abstract representations.

An alternative to hierarchical representations is to use hand-crafted features. This approach has been used in computer vision and NLP communities. People have spent their careers coming up with features that are invariant. However, the drawbacks of feature engineering are that it requires a priori expert knowledge, time-consuming hand-tuning, and poor reproducibility.

Intuitively, good representations should be hierarchical. For example, in language, there is hierarchy in syntax (words→parts of speech→sentences→text) and semantics (objects, actions, attributes→phrases→statements→stories). In vision, there is the part-whole hierarchy (pixels→edges→textons→parts→objects→scenes).

2 Similarities and differences between Graphical Models and Deep nets

Graphical models learn representations for encoding meaning knowledge and the associated uncertainty in a graphical form. For example, in a latent variable model, we have some meaning that we assign to each variable. On the contrary, neural networks learn representations that facilitate computation and performance on the end-metric; the intermediate representations may not be meaningful or interpretable.

In GMs, learning and inference are based on a rich toolbox of well-studied (structure-dependent) techniques, such as EM, message passing, VI, MCMC, etc. However, in NNs, learning is predominantly based on backpropagation, and inference is done via a "forward pass".

In GMs, graphs represent models, whereas in NNs, graphs represent computation.

In GMs, the graph is a vehicle for: (1) synthesizing a global loss function from a local structure (e.g., potential function, feature function, etc.); (2) for designing sound and efficient inference algorithms (e.g., sum-product, mean-fie.d, etc.); (3) to inspire approximation and penalization (structured MF, tree-approximation, etc.); (4) for monitoring theoretical and empirical behavior and accuracy of inference. The loss function is a major measure of quality of the learning algorithm and the model.

In DNNs, the network is a vehicle (1) to conceptually synthesize complex decision hypothesis (stage-wise projection and aggregation); (2) for organizing computational operations (stage-wise update of latent states); (3) designing processing steps/computing modules (layer-wise parallelization). There is no obvious utility in evaluating DL inference algorithms. The loss function is complex and non-convex.

So far, neural nets are flexible function approximators. Some of the neural nets are actually graphical models where units/neurons represent proper random variables (e.g., boltzmann machines, RBMs, sigmoid belief networks, deep belief networks, deep Boltzmann machines). Note that in all these models, the primary goal is to represent the distribution of the observables. The hidden variables are secondary (auxiliary) elements

used to facilitate learning of complex dependencies between the observables.

The quality of hidden representations is judged by the marginal likelihood. In contrast, graphical models are often concerned with the correctness of learning and inference of all variables.

Restricted Boltzmann Machines (RBMs): Let v be the visible units, and h the hidden units. There are no intralayer connections. The hidden units are independent given the data: $h_j \perp\!\!\!\perp h_k \mid v$.

In the clamped/wake phase, we condition on the data. In the sleep phase, the model hallucinates the visible inputs. The Contrastive-Divergence algorithm computes the gradient step and estimates the second term with a monte Carlo estimate from 1-step of a Gibbs sampler. The negative phase is problematic due to slow convergence and extra computation. There are several tricks, e.g., use Markov chains from previous steps, or parallel tampering, to speed up computation and improve optimization of RBMs.

Sigmoid belief networks are simply Bayes networks with conditionals represented in a particular form:

$$P(S_i = x \mid S_j = s_j : j \neq i) \propto P(S_i = x \mid X_j = s_j : j < i) \prod_{j>i} P(S_j = s_j \mid S_i = x, S_k = s_k : k < j, k \neq i),$$

$$P(S_i = s_i \mid S_j = s_j : j < i) = \sigma \left(s_i^* \sum_{j<i} s_j w_{ij} \right).$$

Here, σ denotes the sigmoid function. Learning a DBN requires estimating the gradient of some objective $\mathcal{L}$ over the network weights w_{ij}:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{ij}} &= \sum_{\tilde{v} \in \mathcal{T}} \frac{1}{P(\tilde{V} = \tilde{v})} \frac{\partial P(\tilde{V} = \tilde{v})}{\partial w_{ij}} \\
&= \sum_{\tilde{v} \in \mathcal{T}} \frac{1}{P(\tilde{V} = \tilde{v})} \sum_{\tilde{h}} \frac{\partial P(\tilde{S} =< \tilde{h}, \tilde{v} >)}{\partial w_{ij}} \\
&= \sum_{\tilde{v} \in \mathcal{T}} \sum_{\tilde{h}} P(\tilde{S} =< \tilde{h}, \tilde{v} > | \tilde{V} = \tilde{v}) \cdot \frac{1}{P(\tilde{S} =< \tilde{h}, \tilde{v} >)} \frac{\partial P(\tilde{S} =< \tilde{h}, \tilde{v} >)}{\partial w_{ij}} \\
&= \sum_{\tilde{v} \in \mathcal{T}} \sum_{\tilde{s}} P(\tilde{S} = \tilde{s} | \tilde{V} = \tilde{v}) \frac{1}{P(\tilde{S} = \tilde{s})} \frac{\partial P(\tilde{S} = \tilde{s})}{\partial w_{ij}} \\
&= \sum_{\tilde{v} \in \mathcal{T}} \sum_{\tilde{s}} P(\tilde{S} = \tilde{s} | \tilde{V} = \tilde{v}) \frac{1}{\sigma(s_i^* \sum_{k<i} s_k w_{ik})} \frac{\partial \sigma(s_i^* \sum_{k<i} s_k w_{ik})}{\partial w_{ij}} \\
&= \sum_{\tilde{v} \in \mathcal{T}} \sum_{\tilde{s}} P(\tilde{S} = \tilde{s} | \tilde{V} = \tilde{v}) s_i^* s_j \sigma(-s_i^* \sum_{k<i} s_k w_{ik}),
\end{aligned} \tag{1}$$

Direct computing the gradient is intractable since it requires to do inference which is problematic in BNs. Ranford Neal proposed to use Monte Carlo methods to do inference (Neal, 1992). Due to the directed structure, there is no negative phase as in RBMs. However, convergence is very slow, especially for large belief nets, due to the intricate "explain-away" effects.

Despite sigmoid belief nets are directed graphical models, and RBMs are undirected, there actually exists an equivalence between the two models: **RBMs are infinitely deep belief networks**. Consider the sampling procedure of a RBM(Fig. 1 left), which iteratively perform the following two steps: sample hidden variables conditioned on visible ones, and then sample visible variables from hidden ones. This sampling procedure is equivalent to sampling from an infinitely deep belief networks whose weights of all layers are tied(Fig. 1 right).

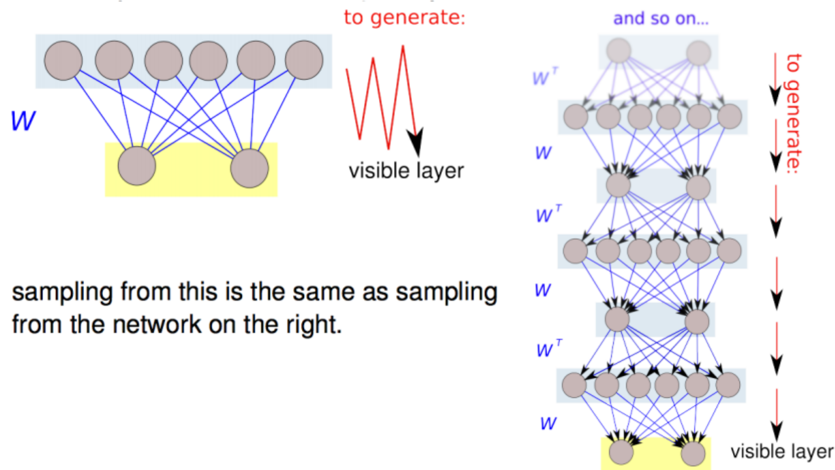

Figure 1: RBMs are infinitely deep belief networks. Left: iterative sampling from RBMs; right: feedforward sampling from an infinitely DN.

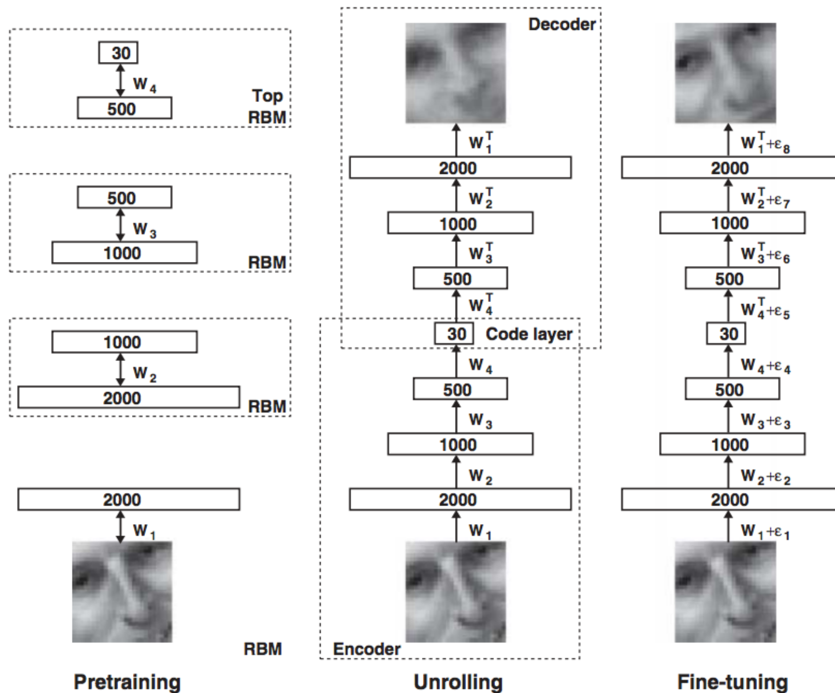

Figure 2: DBN autoencoder training. Left: pretraining consists of learning a stack of RBMs, each having only one layer of feature detectors. The learned features from one RBM are used as the input for training the next RBM in the stack; middle: after the pretraining, the RBMs are “unrolled” to create a deep autoencoder; right, joint fine-tune the weights by backpropagation of reconstruction error derivatives.

Based on this equivalency, Hinton & Salakhutinov [1] proposed a pre-training and finetuning framework to train a DBN autoencoder(see Fig. 2):

1. Pretrain a stack of RBMs in greedy layerwise fashion
2. Unroll the RBMs to create an autoencoder (i.e., untie bottom-up and top-down weights)
3. Fine-tune the parameters using backpropagation.

One intuition to justify this pre-training and finetuning framework is that, in the layerwise pre-training phase, the network learns to abstract the input signal at each layer, and thus gains a high level abstract feature at the end of the encoder. However, the autoencoder loses some information because the input data's dimensionality is bigger than what we can pass through the network, we try to jointly fine-tune the parameters so as to store as much information as possible.

Deep Bayesian Networks (DBNs) are hybrid graphical models (chain graphs) with several layers of latent variables. The latent variables are typically binary, while the visible units may be binary or real. There are no intralayer connections. The connections between the top two layers are undirected, while the connections between all other layers are directed, with the arrows pointed toward the layer that is closest to the data.

Inference in DBNs is problematic due to the "explaining away" effect. Training is done by greedy pre-training and ad-hoc fine-tuning; there is no proper joint training method. Approximate inference is feed-forward. In this case, both the wake phase and sleep phase are intractable.

On the other hand, **Deep Boltzmann Machines (DBMs)** are purely undirected graphical models. Unlike the RBM, the DBM has several layers of latent variables (RBMs have just one). But like the RBM, within each layer, each of the variables are mutually independent, conditioned on the variables in the neighboring layers. Connections are only between units in neighboring layers; there are no intralayer connections. DBMs typically contain only binary units, but it is straightforward to include real-valued visible units. A DBM is an energy-based model, i.e., the joint probability distribution over the model variables is parameterized by an energy E. For a DBM with one visible layer v and three hidden layers $h^{(1)}, h^{(2)}, h^{(3)}$, the joint probability is

$$P(v, h^{(1)}, h^{(2)}, h^{(3)}) = \frac{1}{Z(\theta)} \exp\left(-E(v, h^{(1)}, h^{(2)}, h^{(3)}; \theta)\right)$$

where the energy function is defined by

$$E(v, h^{(1)}, h^{(2)}, h^{(3)}; \theta) = -v^\top W^{(1)} h^{(1)} - h^{(1)\top} W^{(t)} h^{(2)} - h^{(2)\top} W^{(3)} h^{(3)}$$

DBMs' energy function includes connections between the hidden units, which have significant consequences for the model behavior and the inference method. The DBM layers can be organized into a bipartite graph (A, B), with odd layers in A and even layers in B. Thus, the Markov blanket of a hidden layer is the two layers which are below and above it. This layerwise bipartite structure makes Gibbs sampling in a DBM efficient: it can be divided into two blocks of updates, one including all even layers (including the visible layer), and the other including all odd layers.

Similar to DBNs, training DBM can be divided into two phases: a layer-wise pre-training phase which trains each layer of the DBM as a stack of isolated RBMs. And a joint fine-tuning phase which will be discussed in the next lecture.

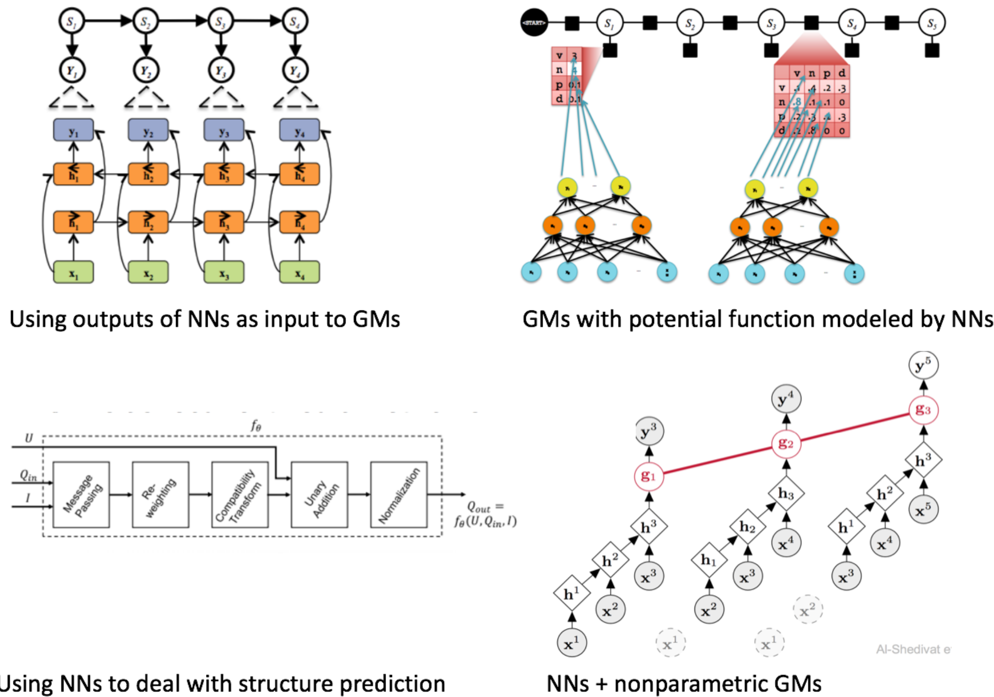

Figure 3: Four approaches of combining deep learning with graphical models.

3 Combining DL methods and GMs

As discussed in the previous section, graphical models and neural networks share many similarities but also have major difference in utilities and learning objectives. Therefore, it's natural to consider the possibility of combining the two models so as to gain benefits from both sides. In the following, we briefly discuss several typical approaches to combine DL with GMs.

Using outputs of NNs as inputs to GMs

One way of combining GMs and NNs is using the outputs of NNs as inputs to GMs. An example to this is hybriding recurrent neural network (RNN) with HMM (Graves *et al.*, 2013). In this hybrid model(see Fig. 3 top-left), HMM takes sequential observations from the output of a bidirectional RNN, and the learning objective is to maximize the log-likelihood of sequential data. The inference of this hybrid model follows exactly the forward-backward algorithm used in HMM, and the learning is done by SGD with backpropagation.

GMs with potential functions by NNs

Another way of combining GMs and NNs is through representing potential functions of GMs with NNs. Hybriding NNs with CRFs belongs to this approach. As illustrated in Fig. 3(top-right), the simple tabular representation of unary and pairwise potential of a CRF is replaced with DNNs. Given this sequential model structure, we can write down the objective function, do inference via CRF inference algorithms, and perform optimization by backpropagation of the same gradient computed for this CRF structure. One thing needs to be emphasized is that, the backpropagation doesn't stop at just computing the gradient of CRF parameters, but also goes through the DNNs' parameters.

One example of this approach is, in Collobert & Weston, 2011 [2], the unary and pairwise potential functions of a linear chain CRF are modeled as CNNs, and the training objective is set to maximize the sentence-level

likelihood.

Dealing with structured prediction
The inference problem of many graphical models can be viewed as an energy-based modelling of structured prediction:

$$\mathbf{y}^*(\mathbf{x};\mathbf{w}) = \arg\min_{\mathbf{y}} E(\mathbf{y},\mathbf{x};\mathbf{w}).$$

Solving such optimization problem for GM inference usually resorts to iterative methods, such as, mean fields, loopy belief propagations, *etc.*. Suppose that the optimization algorithm is run for a fixed number of steps, the optimization algorithm can be unrolled as some deterministic non-linear function as:

$$\mathbf{y}^*(\mathbf{x};\mathbf{w}) = \text{opt-alog}_{\mathbf{y}} E(\mathbf{y},\mathbf{x};\mathbf{w}).$$

In most cases, function $\mathbf{y}^*$ are differentiable over the inputs and model parameters, thus it can be optimized over some loss function as the standard computation graph using backpropagation. This type of approach which unrolls the inference procedure of a graphical model into a DNN is a hot topic in current research literature.

For example, in computer vision, CRF is commonly used for pixel-level labeling tasks, like semantic segmentation. Zheng *et al.* [3] propose to formulate mean-field inference of dense CRF with Gaussian pairwise potentials as a RNN which can refine coarse outputs from a traditional CNN in the forward pass, while passing error differentials back to the CNN during training. Importantly, with this formulation, the whole deep network, which comprises a traditional CNN and an RNN for CRF inference, can be trained end-to-end utilizing the usual back-propagation algorithm. Their experimental evaluation confirms that the proposed network outperforms a system where CRF inference is only applied as a post-processing method on independent pixel-level predictions produced by a pre-trained CNN.

NNs with nonparametric GMs
Non-parametric graphical models can also be combined with neural networks. Consider a sequential model which has finite window that looks back in time, and tries to predict some sequence $\{y_t\}$ given an input sequence $\{x_t\}$. The problem of this approach is that it gives you point-wise predictions, and the outputs are not correlated with each other. Al-Shedivat, *et al.* [4] proposed to use Gaussian processes to combine with DNNs so as to correlate outputs across the samples and also provide uncertainty estimates(see Fig. 3 bottom-right).

4 Conclusion

Deep learning and graphical models are similar in the beginning (structure, energy, etc.) and then diverge to their own signature pipelines. In deep learning, most effort is directed to comparing different architectures and their components (based on empirical performance on a downstream task). DL models are good at learning robust hierarchical representations from the data and suitable for simple reasoning ("low-level cognition"). On the other hand in graphical models, a lot of efforts are directed to improve inference accuracy and convergence speed. GMs are best for provably correct inference and suitable for high-level complex reasoning tasks ("high-level cognition"). Combining both fields is a very promising direction.

References

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[2] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[3] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[4] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. Learning scalable deep kernels with recurrent structure. *arXiv preprint arXiv:1610.08936*, 2016.