CSE6243: Advanced Machine Learning **Fall 2023**

Lecture 9: Neural Network Revisit

Lecturer: Bo Dai *Scribes: Zhaoyu Xu & Mian Wu*

9.1 Recap

Last week we mainly discussed about the *Density/Distribution Parametrization*, specifically –

- Exponential Family, which is defined by:

$$p(x) = h(x)\exp\left(\eta^T T(x) - A(\eta)\right)$$
$$A(\eta) = \log\left(\int h(x)\exp\left(\eta^T T(x)\right)dx\right)$$

- Generalizations: Energy-based Models (EBM), Autoregressive Model (ARM), VAE (and Diffusion)
- Topics mentioned but haven't been thoroughly discussed yet:
  - Deep version of these models
  - Claim: Most of the learning problems can be reformulated into distribution fitting **(Today)**

9.2 New Content

We will start from linear models, and then we will see how we can generalize the linear models to neural network, and what is the generic formation of neural network.

9.2.1 Learning as Distribution Fitting

Recall that we have talked about two major components in learning - $\begin{cases}\text{(1) optimization}\\ \text{(2) posterior estimation}\end{cases}$

Example 9.2.1 (Regression as Conditional Gaussian Fitting) Given n data $\{x_i, y_i\}_{i=1}^n$ with condition model $P(y|x;w) = N\left(w^\top x, \sigma\right)$, we have that the Maximum Likelihood Estimation (MLE) of this problem: $\max_w \frac{1}{n}\sum_{i=1}^n \log P(y|x;w) \iff \min_w \frac{1}{n}\sum_{i=1}^n \left(y_i - w^\top x_i\right)^2$, which is the minimization of the Squared Loss (i.e., Least Square).

Example 9.2.2 (Binary Classification) Given n data $\{x_i, y_i\}_{i=1}^n$ with condition model $P(y|x;w) = \pi(x)^y \cdot (1-\pi(x))^{(1-y)}$ (i.e., Bernoulli Distribution), where $\pi(x) = \sigma\left(w^\top x\right)$, $\sigma(a) = \frac{1}{1+e^{-a}} \in (0,1)$ is the sigmoid function, we have that the MLE: $\max_w \frac{1}{n}\sum_{i=1}^n \log p\left(y_i \mid x_i, w\right) \iff \min_w -\frac{1}{n}\sum_{i=1}^n y_i \log\sigma\left(w^\top x_i\right) + \left(1-y_i\right)\log\left(1-\sigma\left(w^\top x_i\right)\right)$, which is the minimization of the log-loss/binary cross-entropy loss.

Example 9.2.3 (Multi-Class Classification) Given n data $\{x_i, y_i\}_{i=1}^n$ with condition model $P(y|x;w) = \prod_{i=1}^k \pi_i(x)^{\mathbf{1}_i(y)}$, where $y \in \{1,\cdots,k\}$, $\mathbf{1}_i(y)$ is the index function $\mathbf{1}_i(y) = \begin{cases} 1 & \text{if } y = i \\ 0 & \text{otherwise} \end{cases}$. Note that here we want to simulate the model as in Binary Classification, thus naturally we want the $\pi_i(x)$ to indicate the probability that a sample x is from class i, which requires us: $\pi_i(x) \in (0,1)$ and $\sum_i \pi_i(x) = 1$. A possible choice would be the softmax function: $\pi_i(x) = \frac{\exp\left(w_i^\top x\right)}{\sum_j \exp\left(w_j^\top x\right)}$, under which the MLE of this problem would be equivalent to the minimization of the cross-entropy loss.

Example 9.2.4 (Unsupervised Learning/Word2vec) (Supplementary Content) For the Word2vec model, we want to find a way to represent a word with a fixed-length vector, thus to "quantify" those words for future uses (for instance, calculate the similarity). Under the Word2vec structure, we have two models: skip-gram (uses the conditional probability given a central word to generate other context words that surround it) and CBOW (continuous bag of words, uses the conditional probability given the surrounding context words to generate the central word). Specifically, given a word dictionary, $\mathcal{V} = \{0, 1, \ldots, |\mathcal{V}|-1\}$, for a word indexed as i, its vector is represented as $\mathbf{v}_i \in \mathbb{R}^d$ when it is the central target word, and $\mathbf{u}_i \in \mathbb{R}^d$ when it is a context word.

Then, for the skip-gram model, the conditional probability of generating the context word w_o for a given central target word w_c can be formulated in the softmax manner: $P\left(w_o \mid w_c\right) = \frac{\exp\left(\mathbf{u}_o^\top \mathbf{v}_c\right)}{\sum_{i\in\mathcal{V}} \exp\left(\mathbf{u}_i^\top \mathbf{v}_c\right)}$. For a sentence of length T with words $w^{(1)}, \cdots, w^{(T)}$, assuming independent generation, with context window size m, the likelihood function of the skip-gram model is then the joint probability of generating all the context words given any center word: $\prod_{t=1}^T \prod_{-m\le j\le m, j\ne 0} P\left(w^{(t+j)} \mid w^{(t)}\right)$.

Likewise, for the CBOW model, the conditional probability of generating a central target word from the given context word is: $P\left(w_c \mid w_{o_1}, \ldots, w_{o_{2m}}\right) = \frac{\exp\left(\frac{1}{2m}\mathbf{u}_c^\top\left(\mathbf{v}_{o_1}+\ldots,+\mathbf{v}_{o_{2m}}\right)\right)}{\sum_{i\in\mathcal{V}}\exp\left(\frac{1}{2m}\mathbf{u}_i^\top\left(\mathbf{v}_{o_1}+\ldots,+\mathbf{v}_{o_{2m}}\right)\right)} = \frac{\exp\left(\mathbf{u}_c^\top \overline{\mathbf{v}}_o\right)}{\sum_{i\in\mathcal{V}}\exp\left(\mathbf{u}_i^\top \overline{\mathbf{v}}_o\right)}$, where $\mathcal{W}_o = \{w_{o_1}, \ldots, w_{o_{2m}}\}$ and $\overline{\mathbf{v}}_o = \left(\mathbf{v}_{o_1} + \ldots, +\mathbf{v}_{o_{2m}}\right)/(2m)$. Then, for a sentence of length T, the likelihood function of the CBOW model is the probability of generating all center words given their context words: $\prod_{t=1}^T P\left(w^{(t)} \mid w^{(t-m)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+m)}\right)$. Maximizing these likelihood functions will eventually lead us to the minimization of the corresponding losses of the Word2vec model.

9.2.2 Beyond Linear to Nonlinear - Neural Network

Recall that while we generalize the exponential family to EBM, the process is like:

$$\left\{\begin{array}{l} p(x) = h(x)\exp\left(\eta^T T(x) - A(\eta)\right) \\ A(\eta) = \log\left(\int h(x)\exp\left(\eta^T T(x)\right)dx\right) \end{array}\right. \rightarrow \left\{\begin{array}{l} p(x) = \exp(f(x) - A(f)) \\ A(f) = \log\int\exp(f(x))dx \end{array}\right.$$

Now for a basic linear representation $f(x) = w^\top x$, we can also generalize it to non-linear conditions by introducing non-linear basis $\varphi(x)$, i.e., $f(x) = w^\top x \rightarrow f(x) = w^\top\varphi(x)$, and that's basically how we organize a neural network, where the basis $\varphi(x)$ is usually called the activation function, commonly used ones are–

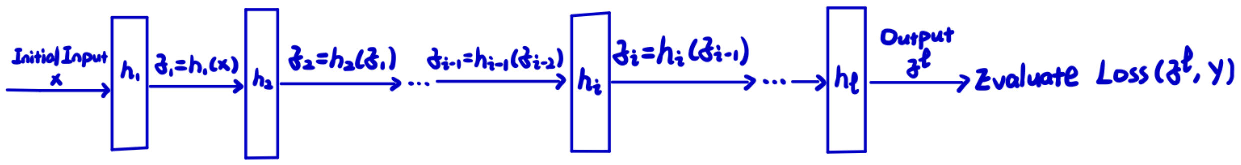
- Sigmoid: $\varphi(x) = \frac{1}{1+e^{-x}}$

- Radial Basis Function (RBF): $\varphi(x) = \exp\left(-\frac{\|x-\mu\|^2}{\sigma^2}\right)$
- Rectified Linear Units (ReLU): $\varphi(x) = \max(0, x)$

To further generalize such 1-layer naive neural network, we can simply stack more layers by function composition (Multilayer Perceptron, MLP): $f_w(x) = w_3^\top \cdot \varphi_3\left(w_2^\top \cdot \varphi_2\left(w_1^\top \cdot \varphi_1(x) + b_1\right) + b_2\right) + b_3 = h_3 \circ h_2 \circ h_1(x)$, where one can keep doing this by defining: $h_i(x) = \omega_i^\top \cdot \varphi_i(x) + b_i$. However, since such general formation of Neural Network is sometimes too flexible, thus in practice we usually introduce additional structures to the network (e.g., CNN/RNN/Transformer, etc.), all of which basically only changes how to compose the functions.

9.2.3 "Solving" Neural Networks from a New Perspective

Having formulated the model of Neural Network, now we naturally care about how to find the "best" parameters (i.e., "solve" the neural network) to minimize the corresponding loss function, i.e., $\min_w \text{Loss}\left(f_w(x), y\right)$. From the first section (Convex Optimization), we have introduced the first-order method (Stochastic) Gradient Descent, which is currently (one of) the dominant algorithms, i.e., $w_{t+1} = w_t - \eta \nabla_w \text{Loss}\left(f_w(x), y\right)$. To find the gradient $\nabla_w \text{Loss}\left(f_w(x), y\right)$, as introduced in Introductory ML courses (for instance, Notes on Back-Prop), normally we shall use back-propagation (i.e., Chain Rule), which can be somewhat tedious. Now we shall provide a new (and even more generalized) perspective to "solve" the neural network - Converting it to an Optimization Problem.



Recall that for an l-layer neural network as shown in the figure above, we have the forward-propagation relationship: $z_i = h_i\left(z_{i-1}\right)$ for $i = 1, \cdots, l$ (denote $x = z_0$), thus we can represent the solving process of such neural network by:

$$\begin{aligned}
\min \ & \text{Loss}\left(z^l\right) \\
\text{s.t.} \ & z^l = h_l\left(z^{l-1}\right) \\
& z^{l-1} = h_{l-1}\left(z^{l-2}\right) \\
& \vdots \\
& z^1 = h_1(x)
\end{aligned}$$

which is actually quite intuitive, as the objective function defines our target of the solving process (i.e., minimizing the loss function), where the conditions define the structure of the problem that we have to satisfy (i.e., the forward-propagation relationship).

To solve such (equality) constrained optimization problem, we can use the Lagrange Multiplier Method by constructing Lagrange Function: $\mathcal{L}\left(\left\{z^i\right\}, \left\{\lambda^i\right\}\right) = \text{Loss}\left(z^\ell\right) - \sum_{i=1}^{\ell}\left(\lambda^i\right)^\top\left(z^i - h_i\left(z^{i-1}\right)\right)$, and solving the system with KKT conditions, i.e.,

$$\begin{cases} \nabla_{z^l}\mathcal{L} = \nabla_{z^l}\,\text{Loss}\left(z^l\right) - \lambda^l = 0 & \\ \nabla_{z^i}\mathcal{L} = \left(\nabla_{z^i} h_{i+1}\left(z^i\right)\right)^\top \lambda^{i+1} - \lambda^i = 0 & 1 \leqslant i \leqslant l-1 \\ \nabla_{\omega^i}\mathcal{L} = \left(\nabla_{\omega^i} h_i\left(z^{i-1}\right)\right)^\top \lambda^i = 0 & 1 \leqslant i \leqslant l \\ \nabla_{\lambda^i}\mathcal{L} = z^i - h_i\left(z^{i-1}\right) = 0 & 1 \leqslant i \leqslant l \end{cases}$$

which would bring us the same result as solving with back-propagation (chain rule).

Furthermore, one may have also noticed that the above paradigm of:

$$\begin{aligned} &\textbf{min}\;\; \text{Loss}\;\; (\textbf{max}\;\; \text{Obj.}) \\ &\;\text{s.t.}\;\; \text{Problem Structures/Relationships} \end{aligned}$$

can be quite universal. Actually, we can apply such re-formulation (to optimization problems) to a whole bunch of models, not just limited to Neural Networks. Even for traditional algorithmic problems that are seemingly not directly differentiable (for instance, Sorting & Ranking, Top-k, etc.), we can still find a way to convert it to a proper optimization problem (see [FP22]), which we shall further discuss in Module III.