# Deep Learning

AlejandroMllo

This document serves as a very brief summary of the topics covered in each chapter of the book Deep Learning [1].

*Disclaimer: This document is completely extracted from [1], the author does not attribute any ownership over the material.*

# 17 Monte Carlo Methods

- Randomize algorithms fall into two rough categories:
  - Las Vegas algorithms, which always return precisely the correct answer (or report that they failed).
  - Monte Carlo algorithms, which return answers with a random amount of error.
- Many technologies used to accomplish ML goals are based on drawing samples from some PD and using these samples to form a Monte Carlo estimate of some desired quantity.
- When a sum or an integral cannot be computed exactly, it is often possible to approximate it using Monte Carlo sampling. The idea is to view the sum or integral as if it were an expectation under some distribution and to approximate the expectation by a corresponding average. Let

  $$s = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x}) = E_p[f(\mathbf{x})]$$

  or

  $$s = \int p(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = E_p[f(\mathbf{x})]$$

  be the sum or integral to estimate, rewritten as an expectation, with the constraint that $p$ is a PD (for the sum) or a probability density (for the integral) over random variable $\mathbf{x}$.
  It is possible to approximate $s$ by drawing $n$ samples $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n)}$ from $p$ and then forming the empirical average

  $$\hat{s}_n = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}^{(i)}).$$

  We compute both the empirical average of the $f(\boldsymbol{x}^{(i)})$ and the empirical variance, and then divide the estimated variance by the number of samples $n$ to obtain an estimator of $\mathrm{Var}[\hat{s}_n]$.
- Any Monte Carlo estimator

  $$\hat{s}_p = \frac{1}{n} \sum_{i=1, \mathbf{x}^{(i)} \sim p}^{n} f(\boldsymbol{x}^{(i)})$$

  can be transformed into an importance sampling estimator

  $$\hat{s}_q = \frac{1}{n} \sum_{i=1, \mathbf{x}^{(i)} \sim q}^{n} \frac{p(\boldsymbol{x}^{(i)}) f(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}.$$

- The family of algorithms that use Markov chains to perform Monte Carlo estimates is called Markov chain Monte Carlo methods (MCMC). They are used to approximately sample from $p_{model}(\mathbf{x})$ when there is no tractable method for drawing exact samples from the distribution $p_{model}(\mathbf{x})$ or from a good (low variance) importance sampling distribution $q(\mathbf{x})$. In the context of deep learning, this most often happens when $p_{model}(\mathbf{x})$ is represented by an undirected model.
- The most standard, generic guarantees for MCMC techniques are only applicable when the model does not assign zero probability to any state.
- The core idea of a Markov chain is to have a state $\boldsymbol{x}$ that begins as an arbitrary value. Over time, we randomly update $\boldsymbol{x}$. Eventually $\boldsymbol{x}$ becomes (very nearly) a fair sample from $p(\boldsymbol{x})$. Formally, it is defined by a random state $\boldsymbol{x}$ and a transition distribution $T(\boldsymbol{x}'|\boldsymbol{x})$ specifying the probability that a random update will go to state $\boldsymbol{x}'$ if it starts in state $\boldsymbol{x}$. Running the Markov chain means repeatedly updating the state $\boldsymbol{x}$ to a value $\boldsymbol{x}'$ sampled from $T(\boldsymbol{x}'|\boldsymbol{x})$.

- A conceptually simple and effective approach to building a Markov chain that samples from $p_{model}(\boldsymbol{x})$ is to use Gibbs sampling, in which sampling from $T(\boldsymbol{x}'|\boldsymbol{x})$ is accomplished by selecting one variable $\mathrm{x}_i$ and sampling it from $p_{model}$ conditioned on its neighbors in the undirected graph $\mathcal{G}$ defining the structure of the EBM. It is also possible to sample several variables at the same time as long as they are conditionally independent given all their neighbors.
- MCMC methods have a tendency to mix poorly.
- Markov chains based on tempered transitions temporarily sample from higher-temperature distributions to mix to different modes, then resume sampling from the unit temperature distribution.
- In parallel tempering, the Markov chain simulates many different states in parallel, at different temperatures. The highest temperature states mix slowly, while the lowest temperature states, at temperature 1, provide accurate samples from the model. The transition operator includes stochastically swapping states between two different temperature levels, so that a sufficiently high-probability sample from a high-temperature slot can jump into a lower temperature slot.
- Depth may help mixing.

## References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.