Christian B. Mendl, Richard M. Milbradt due: 28 July 2022 (before tutorial)

Tutorial 13 (Tensor networks for generative modeling)
Tensor networks are capable of representing certain stochastic probability models. We first consider a *Markov chain*, which describes a stochastic sequence of discrete events $(s_0, s_1, \dots s_T)$, such that the probability distribution of the next event s_{t+1} only depends on the current event s_t. Formally, this setup is expressed by a transition probability

$$\mathbb{P}(s_{t+1} = j | s_t = i) = p_{ij}.$$

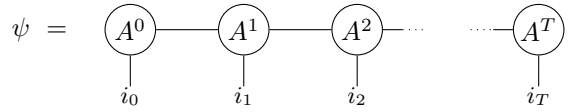
The left side is the conditional probability that the next event is j, given that the current event is i. The matrix $P = (p_{ij}) \in \mathbb{R}^{n\times n}$ is a stochastic matrix, i.e., it has non-negative entries and obeys $\sum_j p_{ij} = 1$ for all i. The initial event s_0 follows some prescribed probability distribution $p^{\text{init}} \in \mathbb{R}^n$.

In view of quantum computing, we now try to construct a so-called "Born machine": the squared entries $|\psi_j|^2$ of a quantum statevector ψ are interpreted as probabilities. For the Markov chain example, each entry corresponds to one possible sequence of events, and the squared amplitude to the probability that this sequence occurs:

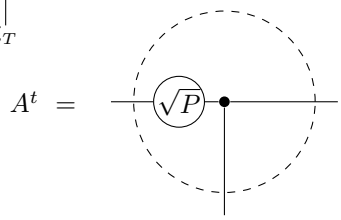
$$|\psi_{i_0,\dots,i_T}|^2 \stackrel{!}{=} \mathbb{P}(s_T = i_T, \dots, s_0 = i_0) = p^{\text{init}}_{i_0} \cdot p_{i_0,i_1} \cdot p_{i_1,i_2} \cdots p_{i_{T-1},i_T}.$$

Note that a quantum measurement of ψ is then equivalent to sampling a realization of the Markov chain.

(a) Consider a MPS Ansatz for ψ:



Show that the MPS tensors A^t for $1 \le t < T$ drawn on the right with $\sqrt{P}$ defined entrywise, together with suitably chosen tensors A^0 and A^T, give rise to the desired ψ.



For the second part we discuss decision tree-like models.[1]

(b) Consider now a scenario where $T = 2$ and event s_0 results in two independent events s_1, each of which results in two independent s_2. Draw the tree diagram which represents this scenario.

(c) It is possible to interpret the tree diagram as a tensor network where the leaves have a free index. Contracting the network would then result in the probability of the state $\{s_2^{(1)}, s_2^{(2)}, s_2^{(3)}, s_2^{(4)}\}$ occurring, without any knowledge of the states s_0 and s_1. Assume, for simplicity, that the probabilities for the events at each branch are the same and draw the tree tensor network, clearly defining the tensor at each node.

Exercise 13.1 (Derivative of matrix inversion)
Our goal is to "differentiate" the inversion of a matrix, $A \mapsto A^{-1}$, for non-singular $A \in \mathbb{R}^{n\times n}$. In *forward mode* differentiation, we imagine that the entries of A depend on some real parameter x, and use the notation $\dot{A} = \frac{\mathrm{d}}{\mathrm{d}x}A(x)$. The inverse matrix is referred to as $C = A^{-1}$ in the following.

(a) Observe that $CA = I$ is constant (independent of x), such that $\frac{\mathrm{d}}{\mathrm{d}x}(CA) = 0$. Combine this equation with the product rule for differentiation to derive that

$$\dot{C} = -C\dot{A}C. \tag{1}$$

To explain *reverse mode* differentiation, let us consider a vector-valued differentiable function $f : \mathbb{R} \to \mathbb{R}^n$, $y = f(x)$. Imagine that, possibly after a sequence of calculations, y determines the scalar output of a "cost" function $\mathcal{L}$. We use the notation $\overline{y} = \frac{\partial \mathcal{L}}{\partial y}$ for the gradient vector of $\mathcal{L}$ with respect to y. Indirectly $\mathcal{L}$ also depends on x through f. By the chain rule,

$$\overline{x} = \frac{\partial \mathcal{L}}{\partial x} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial y_i}\frac{\partial y_i}{\partial x} = \langle \overline{y}, \nabla f(x)\rangle\,. \tag{2}$$

(b) In the context of matrix inversion, we identify y with C (interpreted as vector) and x with one of the entries of A. Use that $\langle \mathrm{vec}(B), \mathrm{vec}(A)\rangle = \mathrm{tr}[B^T A]$ for real matrices A, B and Eqs. (1), (2), to derive that

$$\overline{A} = -C^T\,\overline{C}\,C^T.$$

Hint: First compute $\overline{a_{ij}}$, with a_{ij} one of the entries of A and the other entries assumed constant, using Eq. (1) for $\partial C/\partial a_{ij}$.

[1] S. Cheng, L. Wang, T. Xiang, P. Zhang: *Tree tensor networks for generative modeling*, PRB 99, 155131 (2019), arXiv:1901.02217

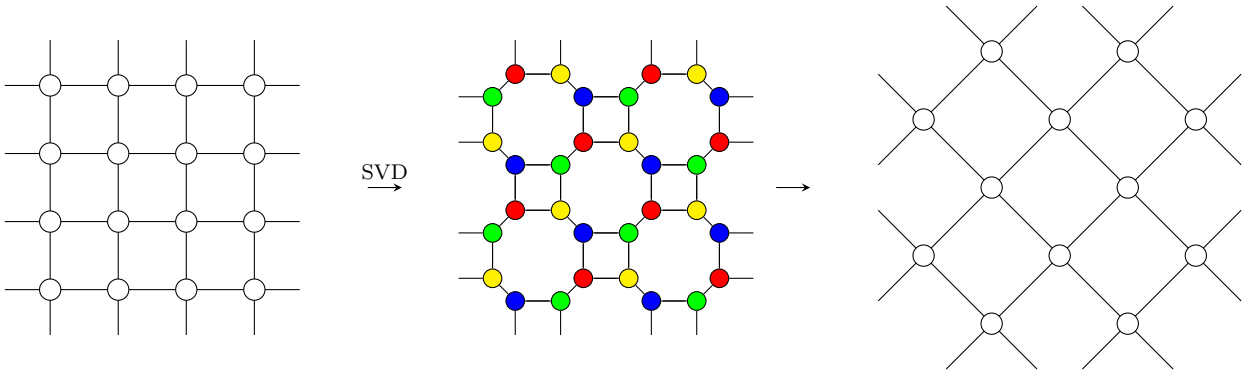
Exercise 13.2 (Tensor renormalization group algorithm and automatic differentiation[2])
The *tensor renormalization group* (TRG) algorithm evaluates the contraction of a large tensor network defined on a lattice by iteratively "coarse-graining" the network. As specific example, we consider here the classical Ising model on a square lattice with periodic boundary conditions.[3] After some preprocessing steps, the so-called partition function Z of this model results from contracting the tensor network shown below on the left. Each individual (white) tensor (with indices $u, \ell, d, r \in \{0, 1\}$ for "up", "left", "down", "right") is defined by

$$T_{u,\ell,d,r} = \quad \ell \text{—}\bigcirc\text{—} r \quad = \frac{1}{2}\sqrt{\lambda_u \lambda_\ell \lambda_d \lambda_r}\, \delta_{\mathrm{mod}(u+\ell+d+r,2),0}$$

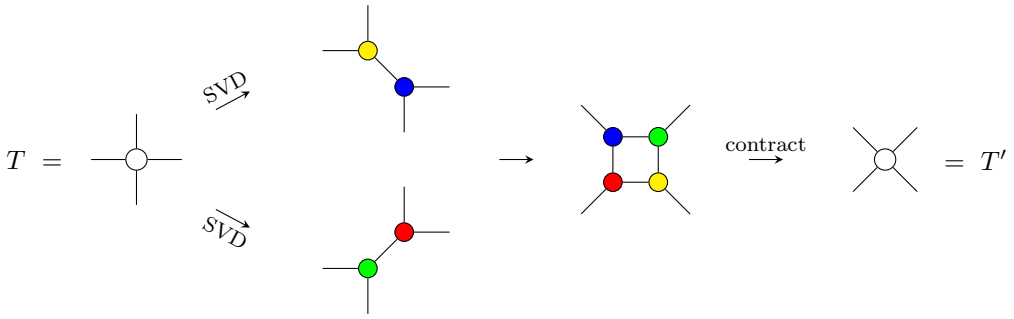
(with u above, d below the circle)

with $\lambda = (2\cosh(\beta), 2\sinh(\beta))$, given the "inverse temperature" $\beta > 0$. In this exercise all quantities are real-valued.

The main idea of the algorithm is to first split the T tensors by SVDs (with truncation), as illustrated by the diagram in the middle, then regroup and contract the resulting tensors according to a certain pattern. This again leads to a tensor network defined on a square lattice, but now a "coarser" one (scaled by a factor of $\sqrt{2}$), which means half the number of tensors. This "renormalization" step of the algorithm is repeated until only a single tensor remains. The final contraction is simply the trace of this tensor.



In our case the T tensors are the same for all lattice sites, such that it suffices to store a single tensor. A renormalization step is thus realized by:



with T' the tensor at the next iteration. We assume that both SVDs retain the same number of singular values, denoted D', such that $T' \in \mathbb{R}^{D' \times D' \times D' \times D'}$.

Implement the function `renormalize(T, Dmax, epsilon)` for a single renormalization step in the accompanying Jupyter notebook template using NumPy. D' should be determined by the number of singular values larger than `epsilon` and upper bounded by $D' \leq$ `Dmax`. For each SVD, you can absorb the square-root of the kept singular values, $(\sqrt{\sigma_j})_{j=1,\ldots,D'}$, both in the U and V matrices. Finally run the overall notebook, which should result in a relative error $\approx 4 \cdot 10^{-6}$.

Remark: The JAX toolbox (`https://github.com/google/jax`) can enrich NumPy code with automatic differentiation features. In principle, one only has to replace `import numpy as np` by `import jax.numpy as np` at the beginning of a source file. For the present example, an interesting quantity is the derivative of the logarithm of the partition function with respect to β, yielding the thermodynamic energy. However, differentiating through a SVD is currently not (yet) supported by JAX, and thus the toolbox not directly usable here. See the repository `https://github.com/wangleiphy/tensorgrad` for a demonstration using PyTorch.

[2] Exercise based on: H.-J. Liao, J.-G. Liu, L. Wang, T. Xiang: *Differentiable programming tensor networks*, Phys. Rev. X 9, 031041 (2019) (arXiv:1903.09650), `https://github.com/wangleiphy/tensorgrad`

[3] Familiarity with the physics of the classical Ising model and concepts like the partition function are not required for the exam.