```julia
using PlutoUI
```

Table of Contents

```julia
TableOfContents()
```

```julia
using LuxorGraphPlot, Luxor
```

The backward rule of matrix multiplication

Let $\mathcal{T}$ be a stack, and $x \rightarrow \mathcal{T}$ and $x \leftarrow \mathcal{T}$ be the operation of pushing and poping an element from this stack. Given $A \in R^{l\times m}$ and $B \in R^{m\times n}$, the forward pass computation of matrix multiplication is

$$\begin{aligned}&C = AB\\&A \rightarrow \mathcal{T}\\&B \rightarrow \mathcal{T}\\&\ldots\end{aligned}$$

Let the adjoint of x be $\overline{x} = \frac{\partial \mathcal{L}}{\partial x}$, where $\mathcal{L}$ is a real loss as the final output. The backward pass computes

$$\begin{aligned}&\ldots\\&B \leftarrow \mathcal{T}\\&\overline{A} = \overline{C}B\\&A \leftarrow \mathcal{T}\\&\overline{B} = A\overline{C}\end{aligned}$$

The rules to compute $\overline{A}$ and $\overline{B}$ are called the backward rules for matrix multiplication. They are crucial for rule based automatic differentiation.

Deriving the backward rules

Let us introduce a small perturbation δA on A and δB on B,

$$\delta C = \delta AB + A\delta B$$

$$\delta \mathcal{L} = \mathrm{tr}(\delta C^T \overline{C}) = \mathrm{tr}(\delta A^T \overline{A}) + \mathrm{tr}(\delta B^T \overline{B})$$

It is easy to see

$$\delta L = \mathrm{tr}((\delta AB)^T \overline{C}) + \mathrm{tr}((A\delta B)^T \overline{C}) = \mathrm{tr}(\delta A^T \overline{A}) + \mathrm{tr}(\delta B^T \overline{B})$$

We have the backward rules for matrix multiplication as

$$\begin{aligned}&\overline{A} = \overline{C}B^T\\&\overline{B} = A^T\overline{C}\end{aligned}$$

The backward rule of eigen decomposition

Ref: https://arxiv.org/abs/1710.08717

Given a symmetric matrix A, the eigen decomposition is

$$A = UEU^\dagger$$

We have

$$\overline{A} = U\left[\overline{E} + \frac{1}{2}\left(\overline{U}^\dagger U \circ F + h.c.\right)\right]U^\dagger$$

Where $F_{ij} = (E_j - E_i)^{-1}$.

If E is continuous, we define the density $\rho(E) = \sum\limits_k \delta(E - E_k) = -\frac{1}{\pi}\int_k \Im[G^r(E, k)]$ (check sign!). Where $G^r(E, k) = \frac{1}{E - E_k + i\delta}$.

We have

$$\overline{A} = U\left[\overline{E} + \frac{1}{2}\left(\overline{U}^\dagger U \circ \Re[G(E_i, E_j)] + h.c.\right)\right]U^\dagger$$

Tensors and tensor decomposition

Ref: Golub, Section 12

Tensor contraction

We use the `einsum` function to compute the tensor contraction.

The diagramatic representation

Higher order SVD

Quiz: How do you use matrix singular value decomposition (or principle componnt analysis) in your own research?

Tucker decomposition

Suppose $A \in R^{n_1 \times n_2 \times n_3}$ and assume that $r \leq \textbf{rank}(A)$with inequality in at least one component. Prompted by the optimality properties of the matrix SVD, let us consider the following optimization problem:

$$\min_X \|A - X\|_F$$

such that

$$X_{lmn} = \sum_{j_1=j_2=j_3=1}^{r_1,r_2,r_3} S_{j_1j_2j_3}(U_1)_{lj_1}(U_2)_{mj_2}(U_3)_{nj_3}.$$

We refer to this as the Tucker approximation problem.

The pseudocode for Tucker decomposition algorithm:

```
Repeat:
  for $k = l, \ldots, d$
    Compute the SVD
      $A(k)(U_d \otimes \ldots \otimes U_{k+1} \otimes U_{k-1} \otimes \ldots \otimes U_1) = \tilde{U}_k \Sigma_k V_k^T$
    $U_k = \tilde{U}_k(:, 1 : r_k)$
  end
```

```
using LinearAlgebra
```

```
tucker_decomp (generic function with 1 method)
```

```julia
function tucker_decomp(X::AbstractArray{T,N}, rs::Vector{Int}; nrepeat::Int) where {T, N}
    # the first sweep, to generate U_k
    Us = [Matrix{T}(I, size(X, i), size(X, i)) for i=1:N]
    Ak = X
    for n=1:nrepeat
        for i=1:N
            Ak = tucker_movefirst(X, Us, i)
            ret = svd(reshape(Ak, size(Ak, 1), :))
            Us[i] = ret.U[:,1:rs[i]]
        end
        Ak = permutedims(Ak, (2:N..., 1))
        dist = norm(tucker_project(tucker_project(X, Us), Us; inverse=true) .- X)
        @info "The Frobenius norm distance is: $dist"
    end
    return tucker_project(X, Us), Us
end
```

```
tucker_movefirst (generic function with 1 method)
```

```julia
function tucker_movefirst(X::AbstractArray{T, N}, Us, k::Int) where {N, T}
    Ak = X
    for i=1:N
        # move i-th dimension to the first
        if i!=1
            pm = collect(1:N)
            pm[1], pm[i] = pm[i], pm[1]
            Ak = permutedims(Ak, pm)
        end
        if i != k
            # multiply Uk on the i-th dimension
            remain = size(Ak)[2:end]
            Ak = Us[i]' * reshape(Ak, size(Ak, 1), :)
            Ak = reshape(Ak, size(Ak, 1), remain...)
        end
    end
    A_ = permutedims(Ak, (2:N..., 1))
    return permutedims(A_, (k, setdiff(1:N, k)...))
end
```

```
tucker_project (generic function with 1 method)
```

```julia
function tucker_project(X::AbstractArray{T, N}, Us; inverse=false) where {N, T}
    Ak = X
    for i=1:N
        # move i-th dimension to the first
        if i!=1
            pm = collect(1:N)
            pm[1], pm[i] = pm[i], pm[1]
            Ak = permutedims(Ak, pm)
        end
        remain = size(Ak)[2:end]
        Ak = (inverse ? Us[i] : Us[i]') * reshape(Ak, size(Ak, 1), :)
        Ak = reshape(Ak, size(Ak, 1), remain...)
    end
    return permutedims(Ak, (2:N..., 1))
end
```

```julia
X = randn(20, 10, 15);
```

```
(4×5×6 Array{Float64, 3}:                                    , [20×4 Matrix{Float64}:
 [:, :, 1] =                                                     0.318988     0.298372    -0.16
```

```julia
Cor, Us = tucker_decomp(X, [4, 5, 6]; nrepeat=10)
```

```
The Frobenius norm distance is: 50.17417148667813

The Frobenius norm distance is: 49.74678459550737

The Frobenius norm distance is: 49.68685891896426

The Frobenius norm distance is: 49.66997991980766

The Frobenius norm distance is: 49.66282175475892

The Frobenius norm distance is: 49.659172743918596

The Frobenius norm distance is: 49.65707756646826

The Frobenius norm distance is: 49.65574198537893

The Frobenius norm distance is: 49.65480535771437

The Frobenius norm distance is: 49.65409424110672
```

Quiz: compare the size of storage before/after the tucker decomposition.

CP decomposition

Given $X \in R^{n_1 \times n_2 \times n_3}$ and an integer r, we consider the problem

$$\min_X \|A - X\|$$

such that

$$X_{lmn} = \sum_{j=1}^{r} \lambda_j F_{lj} G_{mj} H_{nj}$$

where $F \in R^{n_1 \times r}$, $G \in R^{n_2 \times r}$, and $H \in R^{n_3 \times r}$. This is an example of the CP approximation problem. We assume that the columns of F, G, and H have unit 2-norm.

```
Repeat:
  for $k = l : d$
    Minimize $\|A_{(k)} - \tilde{F}^{(k)}(F^{(d)} \odot \ldots \odot F^{(k+l)} \odot F^{(k-1)} \odot \ldots \odot F^{(1)})\|_F$
        with respect to $\tilde{F}(k)$.
    for $j = l : r$
      $\lambda_j = \|\tilde{F}_{(k)}(:, j)\|$
      $F^{(k)}(:, j) = \tilde{F}_k(:, j)/\lambda_j$
    end
  end
```

Tensor contraction

The `einsum` notation.

- The `einsum` notation for matrix multiplication $C_{ik} := A_{ij}B_{jk}$ is `"ij,jk->ik"`.
- The `einsum` notation for element-wise multiplication is `i,i->i`.
- Guess, what are
  - `ii->`
  - `ii->i`
  - `i->ii`
  - `,,->`
  - `ijb,jkb->ikb`
  - `ij,ik,il->jkl`

```julia
using OMEinsum
```

```
2×2 Matrix{Int64}:
 19  22
 43  50
```

```julia
ein"ij, jk -> ik"([1 2; 3 4], [5 6; 7 8])
```

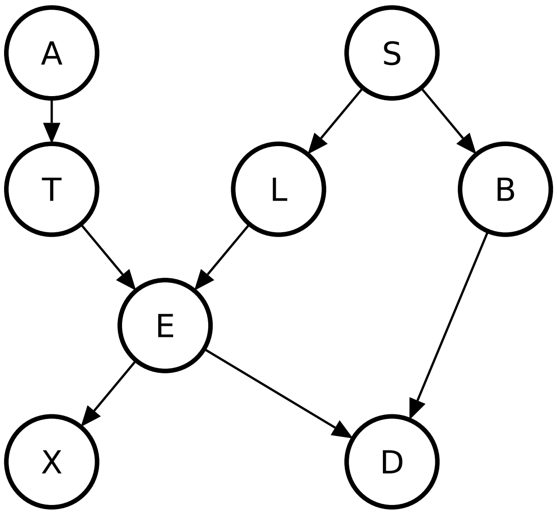
The backward rule of tensor contraction

The backward rule for matrix multiplication is

- `C = ein"ij,jk->ik"(A, B)`
  - `Ā = ein"ik,jk->ij"(C̄, B)`
  - `B̄ = ein"ik,jk->ij"(A, C̄)`
- `v = ein"ii->i"(A)`
  - `Ā = ein"?"(v̄)`

Probability graph

Random variable	Meaning
A	Recent trip to Asia
T	Patient has tuberculosis
S	Patient is a smoker
L	Patient has lung cancer
B	Patient has bronchitis
E	Patient hast T and/or L
X	Chest X-Ray is positive
D	Patient has dyspnoea



A probabilistic graphical model (PGM) illustrates the mathematical modeling of reasoning in the presence of uncertainty. Bayesian networks (above) and Markov random fields are popular types of PGMs. Consider the Bayesian network shown in the figure above known as the *ASIA network*. It is a simplified example from the context of medical diagnosis that describes the probabilistic relationships between different random variables corresponding to possible diseases, symptoms, risk factors and test results. It consists of a graph $G = (V, \mathcal{E})$ and a probability distribution $P(V)$ where G is a directed acyclic graph, V is the set of variables and $\mathcal{E}$ is the set of edges connecting the variables. We assume all variables to be discrete (0 or 1). Each variable $v \in V$ is quantified with a *conditional probability distribution* $P(v \mid pa(v))$ where $pa(v)$ are the parents of v. These conditional probability distributions together with the graph G induce a *joint probability distribution* over $P(V)$, given by

$$P(V) = \prod_{v \in V} P(v \mid pa(v)).$$

The tensor network

A tensor network in physics is also known as the **factor graph** in machine learning, and the **sum-product network** in mathematics.

A tensor network is a multi-linear map from a collection of labelled tensors $\mathcal{T}$ to an output tensor. It is formally defined as follows.

Definition: A tensor network is a multi-linear map specified by a triple of $\mathcal{N} = (\Lambda, \mathcal{T}, \boldsymbol{\sigma}_o)$, where Λ is a set of symbols (or labels), $\mathcal{T} = \{T^{(1)}_{\boldsymbol{\sigma}_1}, T^{(2)}_{\boldsymbol{\sigma}_2}, \ldots, T^{(M)}_{\boldsymbol{\sigma}_M}\}$ is a set of tensors as the inputs, and $\boldsymbol{\sigma}_o$ is a string of symbols labelling the output tensor. Each $T^{(k)}_{\boldsymbol{\sigma}_k} \in \mathcal{T}$ is labelled by a string $\boldsymbol{\sigma}_k \in \Lambda^{r(T^{(k)})}$, where $r\left(T^{(k)}\right)$ is the rank of $T^{(k)}$. The multi-linear map or the **contraction** on this triple is

$$O_{\boldsymbol{\sigma}_o} = \sum_{\Lambda \setminus \sigma_o} \prod_{k=1}^{M} T^{(k)}_{\boldsymbol{\sigma}_k},$$

where the summation runs over all possible configurations over the set of symbols absent in the output tensor. \end{definition} For example, the matrix multiplication can be specified as a tensor network

$$\mathcal{N}_{\rm matmul} = (\{i,j,k\}, \{A_{ij}, B_{jk}\}, ik),$$

where A_{ij} and B_{jk} are input matrices (two-dimensional tensors), and (i, k) are labels associated to the output. The contraction is defined as $O_{ik} = \sum_j A_{ij}B_{jk}$, where the subscripts are for tensor indexing, and the tensor dimensions with the same label must have the same size. The graphical representation of a tensor network is an open hypergraph that having open hyperedges, where an input tensor is mapped to a vertex and a label is mapped to a hyperedge that can connect an arbitrary number of vertices, while the labels appearing in the output tensor are mapped to open hyperedges.

The partition function

https://uaicompetition.github.io/uci-2022/competition-entry/tasks/

The optimal contraction of a tensor network

```
eincode = at, ex, sb, sl, tle, ebd, a, s, t, l, b, e, x, d ->
```

```
eincode = ein"at,ex,sb,sl,tle,ebd,a,s,t,l,b,e,x,d->"
```

```
optimized_eincode =   SlicedEinsum([], e, e ->                         )
                                       └─ e, e -> e
```

```
optimized_eincode = optimize_code(eincode, uniformsize(eincode, 2), TreeSA())
```

```
Time complexity (number of element-wise multiplications) = 2^5.954196310386875
Space complexity (number of elements in the largest intermediate tensor) = 2^2.0
Read-write complexity (number of element-wise read and write) = 2^6.965784284662087
```

```
contraction_complexity(optimized_eincode, uniformsize(optimized_eincode, 2))
```

```
contract (generic function with 1 method)
```

```julia
function contract(ancillas...)
    # 0 -> NO
    # 1 -> YES
    AT = [0.98 0.02; 0.95 0.05]
    EX = [0.99 0.01; 0.02 0.98]
    SB = [0.96 0.04; 0.88 0.12]
    SL = [0.99 0.01; 0.92 0.08]
    TLE = zeros(2, 2, 2)
    TLE[1,:,:] .= [1.0 0.0; 0.0 1.0]
    TLE[2,:,:] .= [0.0 1.0; 0.0 1.0]
    EBD = zeros(2, 2, 2)
    EBD[1,:,:] .= [0.8 0.2; 0.3 0.7]
    EBD[2,:,:] .= [0.2 0.8; 0.05 0.95]
    return optimized_eincode(AT, EX, SB, SL, TLE, EBD, ancillas...)[]
end
```

The backward rule for factor graph contraction

```
0.009999999999999998
```

```julia
contract([0.0, 1.0], [1.0, 0.0], [1.0, 1.0], # A, S, T
        [0.0, 1.0], [1.0, 1.0], # L, B
        [1.0, 1.0], # E
        [1.0, 1.0], [1.0, 1.0] # X, D
        )
```

Random variable	Meaning
A	Recent trip to Asia
T	Patient has tuberculosis
S	Patient is a smoker
L	Patient has lung cancer
B	Patient has bronchitis
E	Patient hast T and/or L
X	Chest X-Ray is positive
D	Patient has dyspnoea

Homework

1. What is the einsum notation for outer product of two vectors?
2. What does the einsum notation `"jk,kl,lm,mj->"` stands for?
3. The tensor network `"abc,cde,efg,ghi,ijk,klm,mno->abdfhjlno"` is known as matrix product state in physics or tensor train in mathematics. Please
    1. Draw a diagramatic representation for it.
    2. If we contract it with another tensor network `"pbq,qdr,rfs,sht,tju,ulv,vnw->pbdfhjlnw"`, i.e., computing `abc,cde,efg,ghi,ijk,klm,mno,pbq,qdr,rfs,sht,tju,ulv,vnw->apow`. What is the optimal contraction order in the diagram, and estimate the contraction complexity (degree of freedoms have the same size n).
    3. Using `OMEinsum` (check the section "Probability graph") to obtain a contraction order and compare it with your answer.