9 Matlab Tricks that You Probably Want to Know

Wittawat Jitkrittum

Gatsby Tea Talk

17 Dec 2015

1. Matrix storage is column-major order

- Physical memory is linear.
- To store a multi-dimensional array, need to arrange it linearly.

Matlab:

- $A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \in \mathbb{R}^{r \times c}$ is internally stored as $(1, 2, 3, 4, 5, 6)^\top$ (column-major).

Tricks/Facts:

- $A(1, 2) == 3$. Can also use **linear index**. $A(3) == 3$
- To flatten A, do $A(:) == (1, 2, 3, 4, 5, 6)^\top$. Get a column vector.
- Internally, Matlab does $A((j-1)r + i)$ for $A(i, j)$.
- C/C++, Python use row-major order.

2. Set diagonal elements

Task:

- $A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \in \mathbb{R}^{r \times r}$. Want to set the diagonal to 0.
- Don't want to use (slow)

```
for i=1:r
    A(i, i) = 0;
end
```

Tricks:

- Use linear indexing. $A(1 : (r+1) : \text{end}) = 0$.
- "end" == 9.
- $1 : (r+1) : \text{end} == 1 : 4 : 9 == [1, 5, 9] ==$ indices of the diagonal elements.

2. Set diagonal elements

Task:

- $A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \in \mathbb{R}^{r \times r}$. Want to set the diagonal to 0.
- Don't want to use (slow)

```
for i=1:r
    A(i, i) = 0;
end
```

Tricks:

- Use linear indexing. $A(1 : (r+1) : \mathrm{end}) = 0$.
- "end" == 9.
- $1 : (r+1) :$ end $== 1 : 4 : 9 == [1, 5, 9] ==$ indices of the diagonal elements.

3. reshape

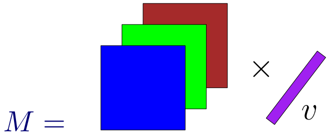
reshape(..) is used to change the shape of an array.

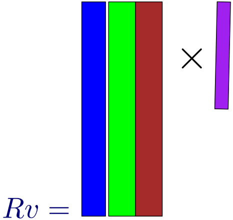
- Read elements in linear order (column-wise).
- $A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$
- reshape(A, 1, 6) == $(1, 2, 3, 4, 5, 6)$. Row vector.
- reshape(A, 3, 2) == $\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$.
- reshape(A, 3, 3). Get an error.
- reshape(A, 3, 2) == reshape(A(:), 3, 2)
- reshape(..) is computationally very cheap.

4. Weighted average on a 3D array

Task:

- $T \in \mathbb{R}^{r\times c\times d}$, a 3d array e.g., d images of size $r \times c$.
- $v \in \mathbb{R}^{d}$, a weight vector.
- Want to multiply to get $M = \sum_{i=1}^{d} T(:,:,i) * v(i) \in \mathbb{R}^{r\times c}$.





- Do not want to use a loop.

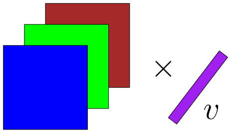
Tricks

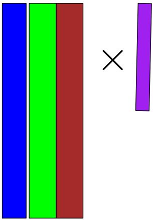
- Use reshape
- $R = \text{reshape}(T, r * c, d)$
- $M = \text{reshape}(R * v, r, c)$

4. Weighted average on a 3D array

Task:

- $T \in \mathbb{R}^{r \times c \times d}$, a 3d array e.g., d images of size $r \times c$.
- $v \in \mathbb{R}^{d}$, a weight vector.
- Want to multiply to get $M = \sum_{i=1}^{d} T(:,:,i) * v(i) \in \mathbb{R}^{r \times c}$.

$M =$ 

$Rv =$ 

- Do not want to use a loop.

Tricks

- Use reshape
- $R = \text{reshape}(T,\ r * c,\ d)$
- $M = \text{reshape}(R * v,\ r,\ c)$

5. Minimum element of a multi-dimensional array

Task:

- $E \in \mathbb{R}^{r \times c \times d}$ e.g., validation errors of param.1 $\times$ param.2 $\times$ param.3
- Find the minimum error, and the corresponding three parameters.

Problem:

- Matlab's **min** operates along one dimension.
- Tedious to find min three times.

Tricks:

```
[minerr, ind] = min(E(:));
[p1_ind, p2_ind, p3_ind] = ind2sub(size(E), ind);
```

- Flatten the array E(:). Find min and its linear index (ind).
- Convert the linear index back to the subscript index.

6. $\text{tr}(A^\top B)$

Task:

- $A, B \in \mathbb{R}^{m\times n}$. Want $\text{tr}(A^\top B)$.
- Inefficient to compute $A^\top B$ and take the trace.

Tricks:

- Let $A := (\boldsymbol{a}_1|\cdots|\boldsymbol{a}_n)$ and $B := (\boldsymbol{b}_1|\cdots|\boldsymbol{b}_n)$.

$$\begin{aligned}\text{tr}(A^\top B) &= \text{sum}(\text{diag}(A^\top B))\\ &= \sum_{j=1}^{n} \boldsymbol{a}_j^\top \boldsymbol{b}_j = \sum_{j=1}^{n}\sum_{i=1}^{m} a_{ij}b_{ij}\\ &= \text{sum}(\text{sum}(A.*B))\\ &= A(:)' * B(:) \text{ in Matlab}\end{aligned}$$

- trace(A'*B) costs $O(mn^2)$.
  - Compute A'*B. Then, throw away off-diagonal entries.
- A(:)'*B(:) = sum(sum(A.*B)) costs $O(mn)$.

7. log-sum-exp trick (not specific to Matlab)

- Want $r^{(k)} = \frac{\prod_{d=1}^{D} p_d^{(k)}}{\sum_{k'=1}^{K} \prod_{d=1}^{D} p_d^{(k')}}$ where $p_d^{(k)} \in (0,1)$ and D is big.
- Example: Posterior probability of the k^{th}-component of a mixture of Bernoulli.

Problem:

- $\prod_{d=1}^{D} p_d^{(k)}$ leads to numerical underflow. Try prod(rand(1, 1000)).

Tricks:

1. Store log prob. $\log r^{(k)} = \sum_d \log p_d^{(k)} - \log \sum_{k'} \prod_d p_d^{(k')}$
2. Introduce c

$$\log \sum_{k'} \prod_{d} p_d^{(k')} = \log \exp(c) + \log \exp(-c) + \log \sum_{k'} \exp \left(\log \prod_{d} p_d^{(k')} \right)$$

$$= c + \log \sum_{k'} \exp \left(\sum_{d} \log p_d^{(k')} - c \right),$$

   choose c so that $\exp\left(\sum_d \log p_d^{(k')} - c\right) > 0$.

3. One way is $c := \max_{k'} \sum_d \log p_d^{(k')} < 0$.

7. log-sum-exp trick (not specific to Matlab)

- Want $r^{(k)} = \frac{\prod_{d=1}^{D} p_d^{(k)}}{\sum_{k'=1}^{K} \prod_{d=1}^{D} p_d^{(k')}}$ where $p_d^{(k)} \in (0,1)$ and D is big.
- Example: Posterior probability of the k^{th}-component of a mixture of Bernoulli.

Problem:

- $\prod_{d=1}^{D} p_d^{(k)}$ leads to numerical underflow. Try prod(rand(1, 1000)).

Tricks:

1. Store log prob. $\log r^{(k)} = \sum_d \log p_d^{(k)} - \log \sum_{k'} \prod_d p_d^{(k')}$
2. Introduce c

$$\log \sum_{k'} \prod_d p_d^{(k')} = \log \exp(c) + \log \exp(-c) + \log \sum_{k'} \exp\left(\log \prod_d p_d^{(k')}\right)$$
$$= c + \log \sum_{k'} \exp\left(\sum_d \log p_d^{(k')} - c\right),$$

choose c so that $\exp\left(\sum_d \log p_d^{(k')} - c\right) > 0$.

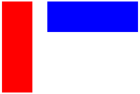
3. One way is $c := \max_{k'} \sum_d \log p_d^{(k')} < 0$.

8. bsxfun and repmat

Task:

- $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$.
- Want $B = f(A, v)$ (f: element-wise) such that $B_{ij} = f(A_{ij}, v_i)$.
- Example: Subtract mean from each column.

Tricks:

- Trick 1: f(A, repmat(v, [1, n]))
- Trick 2: bsxfun(@f, A, v)
  - Same effect as Trick 1 without replicating v. Memory efficient.
- bsxfun can only take in simple f
  - $f \in \{$@plus, @minus, @times, @max, @eq, ...$\}$, not any arbitrary f
- See "doc bsxfun".
- bsxfun also works for  .

9. Embarassingly parallel for-loop

- Want to run an embarassingly parallel for-loop on multiple machines.
- Example: validation_error(θ_i) for i in a long list.

Tricks:

- Download Multicore package (open source).

  http://uk.mathworks.com/matlabcentral/fileexchange/13775-multicore-parallel-processing-on-multiple-cores
- Master/slave machines need to share **temp_dir** for passing information.
- On slave Matlab's, run

```
startmulticoreslave(temp_dir);
```

- On the master,

```
v_error_func = .. (some func. of theta) ..
thetas = {t1, t2, ...}
resultCell = startmulticoremaster(v_error_func, thetas, setting);
```

- resultCell{i} == validation error of θ_i.
- Master/slave machines can be on the same or different machines. Need to share the same file system. Work at Gatsby.
- Should launch slave Matlab's through the job queue (slurm).

References I