Implementing matrix-matrix multiplication

Suggested completion date: March 1, 2011

In this note, we discuss how to implement matrix-matrix multiplication in terms of matrix-vector operations.

An “all purpose” matrix matrix multiplication operation implements all of the operations $C := \alpha \text{op}(A)\text{op}(B) + \beta C$, where $\text{op}(X)$ can be $X$ or $X^T$. Thus, it implements all four of the following operations:

\[
C := \alpha AB + \beta C \\
C := \alpha A^T B + \beta C \\
C := \alpha AB^T + \beta C \\
C := \alpha A^T B^T + \beta C
\]

Because of this, you are asked to create a wrapper routine `SLAP_Gemm` that has the calling sequence

\[
\text{SLAP_Gemm( transA, transB, alpha, A, B, beta, C )}
\]

For example,

\[
C = \text{SLAP_Gemm( SLAP_NO_TRANSPOSE, SLAP_TRANSPOSE, alpha, A, B, beta, C )}
\]

will overwrite matrix $C$ with $\alpha A \ast B + \beta C$.

In the below discussion, you will instead be asked to implement routines that compute $C := \alpha \text{op}(A)\text{op}(B) + C$ (notice: no $\beta$!). The reason is that in the wrapper routine, in Exercise 13, you can first set $C = \beta C$ and then call one of the routines you will write to compute $C := \alpha \text{op}(A)\text{op}(B) + C$, which will have the same net effects as computing $C := \alpha \text{op}(A)\text{op}(B) + \beta C$.

1 SLAP_Gemm_nn

The first routine you will write comes $C := \alpha A B + C$, where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$.

There are three algorithmic variants for computing this.

1.1 Variant 1

Partition $C$ and $B$ by columns. Then $C := \alpha A B + C$ means that

\[
\begin{pmatrix}
c_0 & c_1 & \cdots & c_{n-1}
\end{pmatrix} := \alpha A \begin{pmatrix}
b_0 & b_1 & \cdots & b_{n-1}
\end{pmatrix} + \begin{pmatrix}
c_0 & c_1 & \cdots & c_{n-1}
\end{pmatrix}.
\]

Perform a blocked matrix-matrix multiplication and you get

\[
\begin{pmatrix}
c_0 & c_1 & \cdots & c_{n-1}
\end{pmatrix} := \begin{pmatrix}
\alpha Ab_0 & \alpha Ab_1 & \cdots & \alpha Ab_{n-1}
\end{pmatrix} + \begin{pmatrix}
c_0 & c_1 & \cdots & c_{n-1}
\end{pmatrix},
\]

or,

\[
\begin{pmatrix}
c_0 & c_1 & \cdots & c_{n-1}
\end{pmatrix} := \begin{pmatrix}
\alpha Ab_0 + c_0 & \alpha Ab_1 + c_1 & \cdots & \alpha Ab_{n-1} + c_{n-1}
\end{pmatrix}.
\]

What you find is that $c_j$, the $j$th column of $C$ must be updated by $c_j := \alpha Ab_j + c_j$. But you have a routine for doing that in your library, `SLAP_Gemv`.

Exercise 2 Implement $C := \alpha A B + C$ using the above insights. Use the Spark webpage to generate the code skeleton. Call the resulting routine `SLAP_Gemm_nn_unb_var1`. (Note: the nn stands for “no transpose, no transpose”.)
2.1 Variant 2

Partition $C$ and $A$ by rows. Then $C := \alpha AB + C$ means that

$$
\begin{pmatrix}
\hat{c}_0^T \\
\hat{c}_1^T \\
\vdots \\
\hat{c}_{m-1}^T
\end{pmatrix} = \alpha
\begin{pmatrix}
\hat{a}_0^T \\
\hat{a}_1^T \\
\vdots \\
\hat{a}_{m-1}^T
\end{pmatrix} B +
\begin{pmatrix}
\hat{c}_0^T \\
\hat{c}_1^T \\
\vdots \\
\hat{c}_{m-1}^T
\end{pmatrix}.
$$

Perform a blocked matrix-matrix multiplication and you get

$$
\begin{pmatrix}
\hat{c}_0^T \\
\hat{c}_1^T \\
\vdots \\
\hat{c}_{m-1}^T
\end{pmatrix} = \begin{pmatrix}
\alpha\hat{a}_0^T B \\
\alpha\hat{a}_1^T B \\
\vdots \\
\alpha\hat{a}_{m-1}^T B
\end{pmatrix} +
\begin{pmatrix}
\hat{c}_0^T \\
\hat{c}_1^T \\
\vdots \\
\hat{c}_{m-1}^T
\end{pmatrix},
$$
or,

$$
\begin{pmatrix}
\hat{c}_0^T \\
\hat{c}_1^T \\
\vdots \\
\hat{c}_{m-1}^T
\end{pmatrix} = \begin{pmatrix}
\alpha\hat{a}_0^T B + \hat{c}_0^T \\
\alpha\hat{a}_1^T B + \hat{c}_1^T \\
\vdots \\
\alpha\hat{a}_{m-1}^T B + \hat{c}_{m-1}^T
\end{pmatrix}.
$$

What you find is that $\hat{c}_i^T$, the $i$th row of $C$ must be updated by $\hat{c}_i^T := \alpha\hat{a}_i^T B + \hat{c}_i^T$.

You may THINK you do not have a routine for doing computing a row times a matrix in your library, but you do! Consider

$$y^T = \alpha x^T A + y^T.$$

Transpose both sides:

$$(y^T)^T = (\alpha x^T A + y^T)^T = \alpha(x^T A)^T + (y^T)^T = \alpha A^T(x^T)^T + (y^T)^T = \alpha A^T x + y.$$

So, you can compute $y^T = \alpha x^T A + y^T$ as $y = \alpha A^T x + y$, one of the special cases of SLAP_Gemv!

Exercise 3 Implement $C := \alpha AB + C$ using the above insights. Use the Spark webpage to generate the code skeleton. Call the resulting routine SLAP_Gemm_nn_unb_var2.

3.1 Variant 3

Partition $A$ by columns and $B$ by rows. Then $C := \alpha AB + C$ means that

$$C := \alpha \begin{pmatrix}
a_0 & a_1 & \cdots & a_{k-1}
\end{pmatrix}
\begin{pmatrix}
\hat{b}_0^T \\
\hat{b}_1^T \\
\vdots \\
\hat{b}_{k-1}^T
\end{pmatrix} + C.$$

Perform a blocked matrix-matrix multiplication and you get

$$C := \alpha \left( a_0\hat{b}_0^T + a_1\hat{b}_1^T + \cdots + a_{k-1}\hat{b}_{k-1}^T \right) + C = \alpha a_0\hat{b}_0^T + \alpha a_1\hat{b}_1^T + \cdots + \alpha a_{k-1}\hat{b}_{k-1}^T + C$$

which can be rewritten as

$$C := \left( \alpha a_{k-1}\hat{b}_{k-1} + (\cdots (\alpha a_1\hat{b}_1^T + (\alpha a_0\hat{b}_0^T + C))) \right).$$

Thus, one compute this as a loop where each iteration of the loop updates $C := \alpha a_p\hat{b}_p + C$, $p = 0, \ldots, k - 1$. Each of these operation is a rank-1 update, as we will see in the next section.

You will want to skip this next exercise, and get back to it after you have completed the next section, on implementation of the rank-1 update.

Exercise 4 Implement $C := \alpha AB + C$ using the above insights. Use the Spark webpage to generate the code skeleton. Call the resulting routine SLAP_Gemm_nn_unb_var3. You will first want to implement $C := \alpha a_1\hat{b}_1^T + C$ as $C = \text{alpha} * a * \text{bit} + C.$
5 Rank-1 update

Notice that $C := \alpha a_1 b_1^T + C$ is a special case of the operation $A := \alpha y x + A$, where $A \in \mathbb{R}^{m \times n}$. This itself is an important operation known as a rank-1 update and will be part of your library.

5.1 Variant 1

Let’s examine it a bit: Pick $\alpha = -1$ and partition

$$A = \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$  

Then $A := -y x^T + A$ becomes

$$\begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} := - \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} x^T + \begin{pmatrix} \hat{a}_0^T \\ \hat{a}_1^T \\ \vdots \\ \hat{a}_{m-1}^T \end{pmatrix} = \begin{pmatrix} -\psi_0 x^T + \hat{a}_0^T \\ -\psi_1 x^T + \hat{a}_1^T \\ \vdots \\ -\psi_{m-1} x^T + \hat{a}_{m-1}^T \end{pmatrix} = \begin{pmatrix} \hat{a}_0^T - \psi_0 x^T \\ \hat{a}_1^T - \psi_1 x^T \\ \vdots \\ \hat{a}_{m-1}^T - \psi_{m-1} x^T \end{pmatrix} x^T + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}. $$

So, each row of $A$ is updated by subtracting a multiple of a row vector from it. Where have we seen that before??! Gaussian elimination!

Exercise 6 Implement $A := \alpha y x^T + A$ using the above insights. Use the Spark webpage to generate the code skeleton. Call the resulting routine `SLAP_Ger_unb_var1` with calling sequence `SLAP_Ger_unb_var1( alpha, y, x, A )` (notice the order of the operands!!) What vector-vector operation should you use to implement it?

Now, alternatively one can partition

$$A = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \quad x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}.$$  

Then $A := \alpha y x^T + A$ equals

$$\begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} \end{pmatrix} := \alpha y \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} (a_0 | a_1 | \cdots | a_{n-1})$$

$$\begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} \end{pmatrix} := \alpha y \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} (a_0 | a_1 | \cdots | a_{n-1}) + \begin{pmatrix} \alpha y_0 & \alpha y_1 & \cdots & \alpha y_{n-1} \end{pmatrix} a_0 a_0 + a_1 a_1 + \cdots + a_{n-1} a_{n-1}$$

So, each column of $A$ is updated by adding a multiple of a column vector to it.

Exercise 7 Implement $A := \alpha y x^T + A$ using the above insights. Use the Spark webpage to generate the code skeleton. Call the resulting routine `SLAP_Ger_unb_var2` with calling sequence `SLAP_Ger_unb_var2( alpha, y, x, A )`. What vector-vector operation should you use to implement it?

Exercise 8 Write a wrapper routine `SLAP_Ger( alpha, y, x, A )` that allows $y$ and/or $x$ to be column and/or row vectors. In other words, in the two variants discussed above, you can assume that both $y$ and $x$ are column vectors. Then, in the wrapper routine, you detect if they are row or column vectors, and if necessary you transpose them to make them row or column vectors, similar to what you did for `SLAP_Gemv`.

Which of the two variants will give you better performance if $A$ is stored using column-major order?
9 The other cases of Gemm

Exercise 10 Implement $C := \alpha A^T B + C$ using the insights from Section 1. Use the Spark webpage to generate the code skeleton. Call the resulting routines SLAP_Gemm_tn_unb_var1, SLAP_Gemm_tn_unb_var2, and SLAP_Gemm_tn_unb_var3.

Note: Carefully think about what the $^T$ means in terms of rows become columns and columns becoming rows! What does this mean about how you should march through the matrix?

Exercise 11 Implement $C := \alpha A B^T + C$ using the insights from Section 1. Use the Spark webpage to generate the code skeleton. Call the resulting routines SLAP_Gemm_nn_unb_var1, SLAP_Gemm_nn_unb_var2, and SLAP_Gemm_nn_unb_var3.

Exercise 12 Implement $C := \alpha A^T B^T + C$ using the insights from Section 1. Use the Spark webpage to generate the code skeleton. Call the resulting routines SLAP_Gemm_tn_unb_var1, SLAP_Gemm_tn_unb_var2, and SLAP_Gemm_tn_unb_var3.


To be clear: by the time you are finished with the above exercises, you will have

- For $A := \alpha y^T x + A$:
  - SLAP_Ger_unb_var1( alpha, y, x, A );
  - SLAP_Ger_unb_var2( alpha, y, x, A );

  For these routines, you assume that $y$ and $x$ are column vectors.

  You will also create a wrapper routine SLAP_Ger( alpha, y, x, C ) for which $y$ and $x$ can be row or column vectors.

- For $C := \alpha A B + C$:
  - SLAP_Gemm_nn_unb_var1( alpha, A, B, C );
  - SLAP_Gemm_nn_unb_var2( alpha, A, B, C );
  - SLAP_Gemm_nn_unb_var3( alpha, A, B, C );

- For $C := \alpha A^T B + C$:
  - SLAP_Gemm_tn_unb_var1( alpha, A, B, C );
  - SLAP_Gemm_tn_unb_var2( alpha, A, B, C );
  - SLAP_Gemm_tn_unb_var3( alpha, A, B, C );

- For $C := \alpha A B^T + C$:
  - SLAP_Gemm_nt_unb_var1( alpha, A, B, C );
  - SLAP_Gemm_nt_unb_var2( alpha, A, B, C );
  - SLAP_Gemm_nt_unb_var3( alpha, A, B, C );

- For $C := \alpha A^T B^T + C$:
  - SLAP_Gemm_tn_unb_var1( alpha, A, B, C );
  - SLAP_Gemm_tn_unb_var2( alpha, A, B, C );
  - SLAP_Gemm_tn_unb_var3( alpha, A, B, C );