Homework 01 – Linear Algebra

Arthur J. Redfern
arthur.redfern@utdallas.edu

---

0 Outline

1 Reading
2 Theory
3 Practice

1 Reading

1. Linear algebra
   Motivation: a xNN related linear algebra refresher
   https://github.com/arthurredfern/UT-Dallas-CS-6301-CNNs/blob/master/Lectures/xNNs_010_LinearAlgebra.pdf

   Complete

2. A guide to convolution arithmetic for deep learning
   Motivation: an alternative presentation of CNN style 2D convolution
   https://arxiv.org/abs/1603.07285

   Complete

2 Theory

Matrix multiplication

3. [**2nd part is optional**] What is the arithmetic intensity for matrix matrix multiplication with sizes M, N and K (BLAS notation)? Prove that arithmetic intensity for matrix matrix multiplication is maximized when M = N = K (all matrices are square).

   Part 1: Arithmetic intensity = MNK / (MN + MK + NK)

Part 2: Let the number of MACs be $MNK = c^3$. Given this constraint, arithmetic intensity is maximized when data movement $MN + MK + NK$ is minimized. Use LaGrange multiplier λ to transform the constrained minimization problem into an unconstrained minimization problem of objective function $J(M, N, K, \lambda)$

$\min_{M,N,K,\lambda} J(M, N, K, \lambda) = MN + MK + NK - \lambda(MNK - c^3)$

To find the critical point set partial derivatives to 0

$\partial J/\partial M = N + K - \lambda NK = 0$ (1)
$\partial J/\partial N = M + K - \lambda MK = 0$ (2)
$\partial J/\partial K = M + N - \lambda MN = 0$ (3)
$\partial J/\partial \lambda = -MNK + c^3 = 0$ (4)

Solve (1) for λ

$\lambda = (N + K)/NK$ (5)

then substitute (5) into (2) and simplify to find $M = N$

$M + K - ((N + K)/NK)MK = 0$
$M + K - (MNK + MK^2)/NK = 0$
$M + K - M + MK/N = 0$
$K - MK/N = 0$
$NK = MK$
$M = N$ (6)

Solve (2) for λ

$\lambda = (M + K)/MK$ (7)

then substitute (7) into (3) and simplify to find $K = N$

$M + N - ((M + K)/MK)MN = 0$
$M + N - (MNK + M^2N)/MK = 0$
$M + N - N + MN/K = 0$
$M - MN/K = 0$
$MK = MK$
$M = N$ (8)

Combining (7) and (8) with the constraint (4) yields

$M = N = K = c$

Now all that's left to show is that the choice of M = N = K = c for c^3 MACs results in a minimum for data movement MN + MK + NK and not a maximum or saddle point. Consider a small perturbation of this M, N and K such that their product is still c^3. Specifically, let a_M, a_N and a_K be numbers close to 1 with the product $a_M\, a_N\, a_K = 1$ and write arithmetic intensity as

$$(a_M\, c)\,(a_N\, c)\,(a_K\, c) \,/\, ((a_M\, c\, a_N\, c) + (a_M\, c\, a_K\, c) + (a_N\, c\, a_K\, c))$$
$$= c^3 \,/\, ((a_M\, a_N\, c^2) + (a_M\, a_K\, c^2) + (a_N\, a_K\, c^2)) \qquad (9)$$

Using the inequality of arithmetic and geometric means and moving the arithmetic average value of 3 to the right hand side we know that the denominator

$$((a_M\, a_N\, c^2) + (a_M\, a_K\, c^2) + (a_N\, a_K\, c^2))$$
$$\geq 3\,((a_M\, a_N\, c^2)(a_M\, a_K\, c^2)(a_N\, a_K\, c^2))^{1/3}$$
$$= 3\,(a_M^2\, a_N^2\, a_K^2\, c^6)^{1/3}$$
$$= 3\, c^2$$

or

$$((a_M\, a_N) + (a_M\, a_K) + (a_N\, a_K)) \geq 3$$

with equality corresponding to minimum of data movement occurring when

$$a_M = a_N = a_K = 1 \qquad (10)$$

This completes the proof.

A neglected point in all of the above: M, N and K must be integers. As such, the above only holds for matrix matrix multiplication problems with c^3 MACs for positive integer values of c. However, using "as square as possible" matrices is still the correct intuition to maximize the arithmetic intensity for matrix matrix multiplication with numbers of MACs that don't have 3 repeated positive integer factors.

Dense layers

4. Consider a dense layer that transforms input feature vectors to output feature vectors of the same length ($N_o = N_i$). Ignoring the bias and pointwise nonlinearity, what is the complexity (MACs and memory) of this layer applied to inputs created from vectorized versions of the following:

MNIST:	1 x 28 x 28
CIFAR:	3 x 32 x 32
ImageNet:	3 x 224 x 224 (typical use)
Quasi 1/4 HD:	3 x 512 x 1024
Quasi HD:	3 x 1024 x 2048

Let $N = N_i = N_o$. Memory is composed of the N x 1 input, N x N weight matrix and N x 1 output for a total of $N^2 + 2N$ elements (for large N this is ~ N^2) and matrix vector multiplication results in N^2 MACs.

	N	MACs	Memory
MNIST:	784	614656	617008
CIFAR:	3072	9437184	9446400
ImageNet:	150528	22658678784	22659130368
Quasi 1/4 HD:	1572864	2473901162496	2473905881088
Quasi HD:	6291456	39582418599936	39582437474304

5. In practice, why can't you flatten a quasi HD input image (3 x 1024 x 2048) to a 6291456 x 1 vector and use densely connected layers to transform from data to weak features to strong features to classes?

An impractical number of MACs and memory elements for current computers. Likely also an insufficient amount of training data to effectively find that many coefficients.

6. Say I have trained a dense layer for an input of size 1024 x 1. Can this dense layer be applied to an input of size 2048 x 1? What about 512 x 1?

No and no. A dense layer expects an input of a specific size to maintain compatible matrix vector multiplication dimensions.

CNN style 2D convolution layers

7. [**Optional**] Prove that CNN style 2D convolution with a N_o x N_i x F_r x F_c filter, N_i x L_r x L_c input and N_o x ($L_r - F_r + 1$) x ($L_c - F_c + 1$) output size can be lowered to the sum of (F_r*F_c) matrix matrix multiplications with N_o x N_i matrices made of filter coefficients where the elements of each matrix comes from a single $f_r \in \{0, \ldots, F_r - 1\}$ and single $f_c \in \{0, \ldots, F_c - 1\}$ index and N_i x (M_r*M_c) matrices made from inputs.

Starting from the definition of CNN style 2D convolution

$$Y(n_o, m_r, m_c) = \Sigma_{ni} \Sigma_{fr} \Sigma_{fc} H(n_o, n_i, f_r, f_c) X(n_i, m_r + f_r, m_c + f_c) \forall n_o, m_r, m_c$$

Reorder the summations to

$$Y(n_o, m_r, m_c) = \Sigma_{fr} \Sigma_{fc} (\Sigma_{ni} H(n_o, n_i, f_r, f_c) X(n_i, m_r + f_r, m_c + f_c)) \forall n_o, m_r, m_c$$

and note that the term in the newly introduced parenthesis is a vector inner product for a single f_r and f_c of a length 1 x N_i row $\mathbf{H}(n_o, :, f_r, f_c)$ with a N_i x 1 column $\mathbf{X}(:, m_r + f_r, m_c +$

f_c). Define $\mathbf{H}_{fr,fc}^{2D}$ as the N_o x N_i matrix formed via stacking N_o row vectors from $\mathbf{H}(n_o, :, f_r, f_c)$ on top of each other $\forall$ n_o and define $\mathbf{X}_{fr,fc}^{2D}$ as the N_i x (M_r*M_c) matrix formed via stacking (M_r*M_c) column vectors $\mathbf{X}(:, m_r + f_r, m_c + f_c)$ shoulder to shoulder $\forall$ m_r, m_c. The 2D lowered output $\mathbf{Y}^{2D}$ defined in the usual way as a No x (Mr*Mc) matrix can now be written as

$$\mathbf{Y}^{2D} = \Sigma_{fr} \Sigma_{fc} (\mathbf{H}_{fr,fc}^{2D} \mathbf{X}_{fr,fc}^{2D})$$

For an alternative proof of the same result, start from the definition of $\mathbf{Y}^{2D} = \mathbf{H}^{2D} \mathbf{X}^{2D}$ in the linear algebra notes. Reorder columns in $\mathbf{H}^{2D}$ and corresponding rows of $\mathbf{X}^{2D}$ according to the following pattern of indices: $0:(F_r*F_c):(N_i*F_r*F_c - 1)$, $1:(F_r*F_c):(N_i*F_r*F_c - 1)$, …, $(F_r*F_c - 1):(F_r*F_c):(N_i*F_r*F_c - 1)$. This results in

$$\begin{aligned}\mathbf{Y}^{2D} &= \mathbf{H}^{2D} \mathbf{X}^{2D} \\ &= \mathbf{H}_{reorder}^{2D} \mathbf{X}_{reorder}^{2D} \\ &= [\mathbf{H}_{0,0}^{2D}, \mathbf{H}_{0,1}^{2D}, \ldots, \mathbf{H}_{Fr-1,Fc-1}^{2D}] [\mathbf{X}_{0,0}^{2D}; \mathbf{X}_{0,1}^{2D}; \ldots; \mathbf{X}_{Fr-1,Fc-1}^{2D}] \\ &= \Sigma_{fr} \Sigma_{fc} (\mathbf{H}_{fr,fc}^{2D} \mathbf{X}_{fr,fc}^{2D})\end{aligned}$$

8. Consider a CNN style 2D convolution layer with filter size N_o x N_i x F_r x F_c. How many MACs are required to compute each output point?

$N_i*F_r*F_c$

9. How does CNN style 2D convolution complexity (MACs and memory) scale as a function of
   Product of the image rows and cols (L_r*L_c)?
   Product of the filter rows and cols (F_r*F_c), assume N_i and N_o are fixed?
   Product of the number of input and output feature maps (N_i*N_o)?

Assume L_r, L_c >> F_r, F_c

MACs $= N_o*(L_r - F_r + 1)*(L_c - F_c + 1)*N_i*F_r*F_c$
$\approx N_o*L_r*L_c*N_i*F_r*F_c$

Filter memory $= N_o*N_i*F_r*F_c$

Feature map memory $= N_o*(L_r - F_r + 1)*(L_c - F_c + 1) + N_i*L_r*L_c$
$\approx (N_i + N_o)*L_r*L_c$

L_r*L_c: MACs and feature map memory scale proportionally
Filter memory is independent

F_r*F_c: MACs and filter memory scale proportionally
Feature map memory is independent

N_i*N_o: MACs and filter memory scale proportionally
Feature map memory scales proportional to $(N_i*N_o)^{1/2}$ or
~ proportional to N_i or N_o

10. Consider a CNN style 2D convolution layer with filter size N_o x N_i x F_r x F_c. How many 0s do I need to pad the input feature map with such that the output feature map is the same size as the input feature map (before 0 padding)? What is the size of the border of 0s for $F_r = F_c = 1$? What is the size of the border of 0s for $F_r = F_c = 3$? What is the size of the border of 0s for $F_r = F_c = 5$?

The total row pad is $F_r - 1$ zeros and the total column pad is $F_c - 1$ zeros. For $F_r = F_c = 1$ no border of 0s is added, for $F_r = F_c = 3$ a border of 1 pixel of 0s is added on all sides, for $F_r = F_c = 5$ a border of 2 pixels of 0s is added on all sides.

11. Consider a CNN style 2D convolution layer with N_o x N_i x F_r x F_c filter, N_i x L_r x L_c input (L_r and L_c both even) and $P_r = F_r - 1$ and $P_c = F_c - 1$ zero padding. What is the size of the output feature map with striding $S_r = S_c = 1$ (no striding)? What is the size of the output feature map with striding $S_r = S_c = 2$? How does this change the shape of the equivalent lowered matrix equation?

$S_r = S_c = 1$: the output feature map is of size N_o x L_r x L_c and the equivalent lowered matrix equation has BLAS notation dimensions of

$M = N_o$
$N = L_r*L_c$
$K = N_i*F_r*F_c$

$S_r = S_c = 2$: the output feature map is of size N_o x $(L_r/2)$ x $(L_c/2)$ and the equivalent lowered matrix equation has BLAS notation dimensions of

$M = N_o$
$N = (L_r/2)*(L_c/2) = L_r*L_c/4$
$K = N_i*F_r*F_c$

As such, for $S_r = S_c = 2$ the matrices $\mathbf{Y}^{2D}$ and $\mathbf{X}^{2D}$ have 1/4 the number of columns vs the $S_r = S_c = 1$ case.

12. Say I have trained a CNN style 2D convolution layer for an input of size 3 x 1024 x 2048. Can this CNN style 2D convolution layer be applied to an input of size 3 x 512 x 1024? What about 3 x 512 x 512?

Yes and yes. A CNN style 2D convolution layer works for inputs with arbitrarily sized rows and cols, but expects a specific number of input channels N_i to maintain compatible matrix matrix multiplication dimensions.

RNN layers

13. In a standard RNN, if the state update matrix is constrained to a diagonal, what does this do for the mixing of the previous state with new inputs?

Let $\mathbf{y}_t = f(\mathbf{H}\,\mathbf{x}_t + \mathbf{G}\,\mathbf{y}_{t-1} + \mathbf{v})$ and constrain $\mathbf{G} = \text{diag}(g(0), g(1), \ldots, G(M-1, M-1))$. The mth element of the M x 1 output vector can be written as $y_t(m) = f(\mathbf{H}(m, :)\,\mathbf{x}_t + g(m)\,y_{t-1}(m) + v(m))$. The output feature at the previous time step $y_{t-1}(m)$ only interacts with the corresponding feature at the current time step via the scale g(m). There is not mixing from multiple previous features to current feature.

Attention layers

14. Consider single headed self attention where input $\mathbf{X}^T$ is a M x K matrix composed of M input vectors with K features per vector, $\mathbf{A}_i^T$ is a M x M attention matrix where each element is non negative and each row sums to 1 (think each row is a pmf), $\mathbf{W}_{v,i}$ is a K x L weight matrix and output $\mathbf{Y}_i^T$ is a M x L matrix composed of M output vectors with L features per vector

$$\mathbf{Y}_i^T = \mathbf{A}_i^T\,\mathbf{X}^T\,\mathbf{W}_{v,i}.$$

Can $\mathbf{A}_i^T$ be computed once and then used for all inputs? What does it mean for the output if $\mathbf{A}_i^T$ is an identity matrix? What does it mean for the output if $\mathbf{W}_{v,i}$ is an identity matrix?

No, because $\mathbf{A}_i^T = \text{softmax}_{row}(\mathbf{X}^T\,\mathbf{W}_{q,i}\,\mathbf{W}_{k,i}^T\,\mathbf{X} / P^{1/2})$ is a function of each input $\mathbf{X}^T$. Note that it may be beneficial to pre compute the $\mathbf{W}_{q,i}\,\mathbf{W}_{k,i}^T$ term depending on the matrix dimensions.

If $\mathbf{A}_i^T$ is an identity matrix then each output vector is only a function of the corresponding input vector (no mixing across input vectors to create output vectors, very similar to a dense layer).

If $\mathbf{W}_{v,i}$ is an identity matrix then each output feature is only a function of the corresponding input feature (no mixing across input features to create output features).

Average pooling layers

14. The size of the input to a global average pooling layer is 1024 x 16 x 32. What is the size of the output? What is the complexity (operations) of the layer?

1024 x 1. Each element of the vector output requires summing the 16*32 = 512 elements of the corresponding feature map and dividing the result by 16*32 = 512. Calling this 512 OPs per feature map, there are 1024*512 = 524288 OPs total.

3 Practice

15. For network specification, training and evaluation, this class will use TensorFlow 2.0 (https://www.tensorflow.org/beta) with the Keras functional API (https://keras.io). Begin to familiarize yourself with these via reviewing the following documentation and running the embedded examples in Google Colab:

- Keras overview: https://www.tensorflow.org/beta/guide/keras/overview
- Keras functional API: https://www.tensorflow.org/beta/guide/keras/functional
- Train and evaluate (Part I only): https://www.tensorflow.org/beta/guide/keras/training_and_evaluation

Complete

16. The features of TensorFlow are easier to understand and the problems are easier to debug with small datasets and simple networks (re: Andrej Karpathy's blog post you read for the previous homework assignment). This coding example will use MNIST (60k training and 10k testing 1 x 28 x 28 images of {0, …, 9} digits) with a few layer neural network for classification. Understand all lines of code in the following example (https://github.com/arthurredfern/UT-Dallas-CS-6301-CNNs/blob/master/Code/xNNs_Code_010_MNIST.py) and run it in Google Colab. Note how the code specifies a graph comprised of:

- Data
  - Source
  - Transformation
- Forward path
  - Encoder
  - Decoder
- Loss
- Backward path
  - Implicit
- Weight update

Training specifies a set of hyper parameters, runs the full graph and as a byproduct updates the weights. Evaluation runs the graph from the data to the loss and testing runs the graph from the data to the decoder.

Feel free to modify the code and experiment.

Complete