# CS 598: Communication Cost Analysis of Algorithms

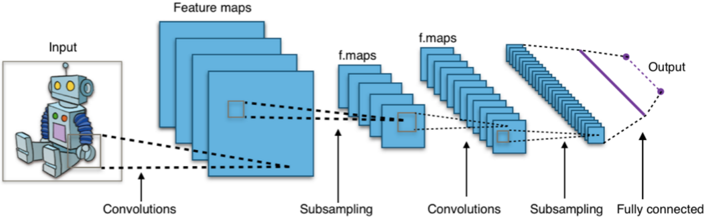
## Lecture 28: Convolutional neural networks

Edgar Solomonik

University of Illinois at Urbana-Champaign

December 5, 2016

# Convolutional neural networks (CNNs)




- CNNs consist of a set of layers, which convolve a 2D or 3D dataset with a set of 2D filters and subsample the result
- they are popular in machine learning with applications including image recognition and recommender systems

image source : https://commons.wikimedia.org/wiki/File:Typical_cnn.png

# 1D convolution

Given $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{f} \in \mathbb{R}^m$, $n \geq m$, compute $\mathbb{R}^{n+m-1} \ni \boldsymbol{y} = \boldsymbol{f} * \boldsymbol{x}$ so

$$\forall k \in [1, n+m-1], \quad \boldsymbol{y}(k) = \sum_{j=\max(1,k-n)}^{\min(m,k)} \boldsymbol{f}(j)\boldsymbol{x}(k-j+1)$$

- the convolution can also be interpreted as matrix-vector multiplication with a banded Toeplitz matrix, e.g. for $n=4, m=2$

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{f}(0) & 0 & 0 & 0 \\ \boldsymbol{f}(1) & \boldsymbol{f}(0) & 0 & 0 \\ 0 & \boldsymbol{f}(1) & \boldsymbol{f}(0) & 0 \\ 0 & 0 & \boldsymbol{f}(1) & \boldsymbol{f}(0) \\ 0 & 0 & 0 & \boldsymbol{f}(1) \end{bmatrix} \boldsymbol{x} = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{x}$$

  where $\mathcal{D}_n(\boldsymbol{f}) \in \mathbb{R}^{(m+n-1)\times n}$ is a Toeplitz matrix generated by $\boldsymbol{f}$
- we also have

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{f} = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{x} = \mathcal{D}_m(\boldsymbol{x})\boldsymbol{f}$$

  but it does not make sense to construct $\mathcal{D}_m(\boldsymbol{x})$ since $n \geq m$

# Computing a 1D convolution

In lecture 9, we proved that a convolution can be done via a $(n+m-1)$-dimensional Fourier transform

- assuming $n \geq m$ using an FFT would give the cost

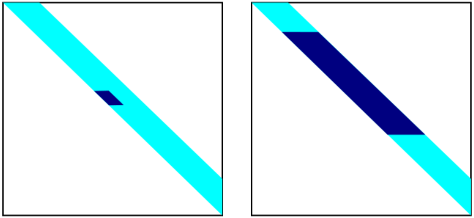
$$T_{\text{1D-FFT}}(n,P) = O(n\log(n)/P\cdot\gamma + n\log_{n/P}(n)/P\cdot\beta + \log_{n/P}(n)\cdot\alpha)$$

- alternatively, can compute the $mn$ scalar products directly and sum
- Q: what communication cost would this incur?
- A: it depends on the relative dimensions of $m,n$

$$T_{\text{1D-MM}}(m,n,P) = O(mn/P\cdot\gamma+\alpha) + \begin{cases} O(\sqrt{mn/P}\cdot\beta) & : n \leq mP \\ O(m\cdot\beta) & : n > mP \end{cases}$$

- when $n \gg m$, the direct approach requires less communication

# Computing a 1D convolution



- rather than communicating a block of $\mathcal{D}_n(\boldsymbol{f})$ we should always communicate a subset of $\boldsymbol{f}$ that implicitly represents it
- when $n > mP$, achieving $O(m)$ communication requires taking advantage of a good initial data layout
- Q: what cache complexity would this algorithm incur for cache size $H$?
- A: $O(n \max(1, m/H) \cdot \nu)$
- for comparison the FFT cache complexity is $O(n \log_H(n) \cdot \nu)$

# Many 1D convolutions

CNNs sometimes apply the same filter $\boldsymbol{f}$ to multiple datasets $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_r\}$

- given $\begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_r \end{bmatrix} = \boldsymbol{X} \in \mathbb{R}^{n \times r}$ and $\boldsymbol{f} \in \mathbb{R}^m$, $n \geq m$, compute $\begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_r \end{bmatrix} = \boldsymbol{Y} \in \mathbb{R}^{(n+m-1) \times r}$

$$\forall i \in [1, r], \quad \boldsymbol{y}_i = \boldsymbol{f} * \boldsymbol{x}_i = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{x}_i$$

- when fully expanded, the computation corresponds to

$$\forall i \in [1, r], k \in [1, n+m-1], \quad \boldsymbol{Y}(k, i) = \sum_{j=\max(1,k-n)}^{\min(m,k)} \boldsymbol{f}(j)\boldsymbol{X}(k-j+1, i)$$

- the set of convolutions can be computed via matrix multiplication

$$\boldsymbol{Y} = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{X}$$

# Computing many 1D convolutions

Performing $r$ FFTs achieves the complexity

$$T_{\text{r1D-FFT}}(r, n, P) = O(rn\log(n)/P \cdot \gamma + rn/P \cdot \beta + \alpha)$$

so long as $\log_{nr/P}(n) = O(1)$

- the matrix multiplication $\boldsymbol{Y} = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{X}$ corresponds to a band with dimensions $m \times n \times r$
- we can compute $\boldsymbol{Y} = \mathcal{D}_n(\boldsymbol{f})\boldsymbol{X}$ with complexity

  $$T_{\text{r1D-MM}}(r, m, n, P) = O(rmn/P \cdot \gamma + m \cdot \beta + \alpha)$$

  by replicating $\boldsymbol{f}$ on all processors (which makes sense when $rn \geq mP$)
- taking the FFT of every subvector locally yields computation cost

  $$O((rn/P + m)\log(rn/P + m) \cdot \gamma)$$

  Q: why is it not $r(n/P + m)\log(n/P + m)$?
- A: if we assign different columns of $\boldsymbol{A}$ to different processors, each column is subdivided over $P/r$ processors

# Convolving the same data with multiple filters

CNNs sometimes apply many filters $\{\boldsymbol{f}_1, \dots, \boldsymbol{f}_r\}$ to a dataset $\boldsymbol{x}$

- given $\boldsymbol{x} \in \mathbb{R}^n$ and $\begin{bmatrix} \boldsymbol{f}_1 & \cdots & \boldsymbol{f}_r \end{bmatrix} = \boldsymbol{F} \in \mathbb{R}^{m \times r}$, $n \geq m$, compute $\begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_r \end{bmatrix} = \boldsymbol{Y} \in \mathbb{R}^{(n+m-1) \times r}$

$$\forall i \in [1, r], \quad \boldsymbol{y}_i = \boldsymbol{f}_i * \boldsymbol{x} = \mathcal{D}_n(\boldsymbol{f}_i)\boldsymbol{x} = \mathcal{D}_m(\boldsymbol{x})\boldsymbol{f}_i$$

- when fully expanded, the computation corresponds to

$$\forall i \in [1, r], k \in [1, n+m-1], \quad \boldsymbol{Y}(k, i) = \sum_{j=\max(1,k-n)}^{\min(m,k)} \boldsymbol{f}(j, i)\boldsymbol{X}(k-j+1)$$

- the set of convolutions can be computed via matrix multiplication

$$\boldsymbol{Y} = \mathcal{D}_m(\boldsymbol{x})\boldsymbol{F}$$

# Computing convolutions with many filters

An FFT approach would still require an FFT for each filter and result vector

$$T_{\text{r1D-FFT}}(r,n,P) = O(rn\log(n)/P\cdot\gamma + rn/P\cdot\beta + \alpha)$$

so long as $\log_{nr/P}(n) = O(1)$

- the matrix multiplication $\boldsymbol{Y} = \mathcal{D}_m(\boldsymbol{x})\boldsymbol{F}$ corresponds to a band with dimensions $n\times m\times r$
- we can compute $\boldsymbol{Y} = \mathcal{D}_m(\boldsymbol{x})\boldsymbol{F}$ with complexity

  $$T_{\text{r1D-MM}}(r,m,n,P) = O(rmn/P\cdot\gamma + n\cdot\beta + \alpha)$$

  by replicating $\boldsymbol{x}$ on all processors (which makes sense when $rm \geq nP$)
- computing a sub-band of dimensions $b_n\times b_m\times b_r$ requires $b_n$ entries of $\boldsymbol{x}$, $b_m b_r$ entries of $\boldsymbol{X}$ and affects $O(b_n b_r + b_m b_r)$ entries of $\boldsymbol{Y}$
- for sufficiently large $P$, we can pick $b_r = 1$ and $b_n = b_m = \sqrt{rmn/P}$, yielding cost

$$T_{\text{r1D-MM}}(r,m,n,P) = O(rmn/P\cdot\gamma + \sqrt{rmn/P}\cdot\beta + \alpha)$$

# Many 1D convolutions with different filters

Other CNNs sum over applications of filters $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_r\}$ to datasets $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$

- compute $\boldsymbol{y} \in \mathbb{R}^{(n+m-1)\times r}$ where each $\boldsymbol{x}_i \in \mathbb{R}^n$ and each $\boldsymbol{f}_i \in \mathbb{R}^m$

$$\boldsymbol{y} = \sum_{i=1}^{r} \boldsymbol{f}_i * \boldsymbol{x}_i = \sum_{i=1}^{r} \mathcal{D}_n(\boldsymbol{f}_i)\boldsymbol{x}_i$$

- defining $\boldsymbol{G} = \begin{bmatrix} \mathcal{D}_n(\boldsymbol{f}_1) & \cdots & \mathcal{D}_n(\boldsymbol{f}_r) \end{bmatrix}$ and $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_r \end{bmatrix}$

$$\boldsymbol{y} = \boldsymbol{G}\,\mathrm{vec}\,(\boldsymbol{X})$$

- $\boldsymbol{X}$ can be sparse with $z$ nonzeros and $\mathrm{vec}\,(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_r \end{bmatrix}$

# Computing many 1D convolutions with different filters

If computing $\boldsymbol{y} = \boldsymbol{G}\boldsymbol{w}$ directly, we should leverage the implicit form of $\boldsymbol{G}$ (each $k \times k$ block can be formed with $O(k)$ entries of $\boldsymbol{f}$)

- the sparsity structure of $\boldsymbol{G}$ is easier to see by folding it into a $r \times (n+m-1) \times n$ tensor $\boldsymbol{H}$, then

$$\boldsymbol{y}(j) = \sum_{i=1}^{r} \sum_{k=1}^{n} \boldsymbol{H}(i,j,k)\boldsymbol{X}(k,i)$$

  $\boldsymbol{H}(i,j,k) = 0$ if $k \geq j$ or $j \geq k+n$ (each $\boldsymbol{H}(i,*,*)$ is banded Toeplitz)
- a sub-band-block of entries of $\boldsymbol{H}$ of volume $b_r \times b_m \times b_n$ is
  - represented by $b_r b_m$ entries of $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_r\}$
  - multiplied by $b_r b_n$ entries of $\boldsymbol{X}$
  - sums into $b_m + b_n$ entries of $\boldsymbol{y}$
- fine-grained case: $b_m \leq m$ seek $b_r b_m b_n = \Theta(rmn/P)$ to minimize

$$O(b_r b_m + b_r b_n) = O((rmn/P)(1/b_n + 1/b_m))$$

- minimized by $b_m = b_n = \sqrt{rmn/P}$, possible so long as $rn \leq mP$, yielding communication complexity $O(\sqrt{rmn/P} \cdot \beta)$

# Cost of many 1D convolutions with different filters

The overall cost of the outlined approach is (so long as $rn \leq mP$)

$$T_{\text{CNN-1D}}(r, m, n, P) = O(rmn/P \cdot \gamma + \sqrt{rmn/P} \cdot \beta + \alpha)$$

- coarse-grained case: $b_m = m$ (assign each processor a subset of whole filter vectors), seek $b_r b_n = \Theta(rn/P)$ to minimize

$$W = O(b_r m + b_n)$$

- minimized when $b_n = \sqrt{rmn/P}$, $b_r = \sqrt{rn/(Pm)}$, again with $W = \Theta(\sqrt{rmn/P})$
- when $\boldsymbol{X}$ is sparse with $z = frn$ nonzeros ($f < 1$) distributed randomly, the term $b_r b_n$ becomes $fb_r b_n$ in the fine-grained case, and we set $fb_n = b_m = \sqrt{frmn/P}$ yielding

$$T_{\text{sparse-CNN-1D}}(z, m, P) = O(zm/P \cdot \gamma + \sqrt{zm/P} \cdot \beta + \alpha)$$

- these costs hold only in certain regimes of values $r, m, n, z, P$

# 1D convolution cross product

The most general variant of CNNs uses $rs$ filters, $sq$ input datasets, and produces $rq$ output datasets

- each output dataset is a sum of one of $r$ sets of $s$ filters applied to one of $q$ sets of $s$ input datasets
- this can be interpreted as a matrix multiplication with dimensions $r \times s \times q$ where each product is a convolution
- if $rsq \geq p$, each convolution can be performed independently
- we should choose $b_r b_s b_q = rsq/p$ to minimize

$$O(b_r b_s m + b_s b_q n + b_r b_q n)$$

- taking into account initial data layout, this becomes

$$O((b_r b_s - rs/P)m + (b_s b_q - sq/P)n + (b_r b_q - rq/P)n)$$

- the appropriate blocking, as well as the benefit of using FFT depends on relative values of $r, s, q, m, n, P$

# 2D convolutions

Given $\boldsymbol{X} \in \mathbb{R}^{n \times n}$, $\boldsymbol{F} \in \mathbb{R}^{m \times m}$, $m \leq n$, compute $\boldsymbol{Y} \in \mathbb{R}^{(n+m-1)\times(n+m-1)}$ so

$$\forall k_1, k_2 \in [1, n+m-1],$$

$$\boldsymbol{Y}(k_1, k_2) = \sum_{j_1=\max(1,k_1-n)}^{\min(m,k_1)} \sum_{j_2=\max(1,k_2-n)}^{\min(m,k_2)} \boldsymbol{F}(j_1, j_2)\boldsymbol{X}(k_1 - j_1 + 1, k_2 - j_2 + 1)$$

- we can evaluate the 2D convolution directly via $O(m^2 n^2)$ operations
- alternatively, we can leverage the FFT, if $\boldsymbol{D}_k$ is the DFT matrix

$$\boldsymbol{D}_{n+m-1}\boldsymbol{Y}\boldsymbol{D}_{n+m-1} = \left(\boldsymbol{D}_{n+m-1}\begin{bmatrix}\boldsymbol{F} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{D}_{n+m-1}\right) \circ \left(\boldsymbol{D}_{n+m-1}\begin{bmatrix}\boldsymbol{X} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{D}_{n+m-1}\right)$$

where $\circ$ is the Hadamard product: $(\boldsymbol{A} \circ \boldsymbol{B})(i,j) = \boldsymbol{A}(i,j)\boldsymbol{B}(i,j)$

- the resulting computation cost is $O(n^2 \log(n))$

# Communication complexity of 2D convolutions

Lets compare direct 2D convolution to FFT-based convolution

- we can block the 4D index space, the total size of which is approximately $n \times n \times m \times m$
- when $n < m\sqrt{P}$, we should subdivide this space into 4D blocks with equal dimensions

$$T_{\text{2D-MM}}(m, n, P) = O(m^2 n^2 / P \cdot \gamma + \frac{mn}{\sqrt{P}} \cdot \beta + \alpha)$$

- Q: what approach should we take when $n \geq m\sqrt{P}$?
- A: we should replicate the filter $\boldsymbol{F}$ on all processors yielding complexity

$$T_{\text{2D-MM}}(m, n, P) = O(m^2 n^2 / P \cdot \gamma + m^2 \cdot \beta + \alpha)$$

- computing the 2D FFTs, when $\log_{n^2/P}(n) = O(1)$ has complexity

$$T_{\text{2D-FFT}}(n, P) = O(n^2 \log(n)/P \cdot \gamma + n^2/P \cdot \beta + \alpha)$$

- the computation complexity of the first approach can sometimes be improved by doing FFT locally

# Many 2D convolutions

We can also consider doing $r$ convolutions with the same filter $\boldsymbol{F}$

- in this case $\boldsymbol{X}$ and $\boldsymbol{Y}$ become order 3 tensors
- performing 2D FFTs would usually yield the overall complexity

$$T_{\text{r2D-FFT}}(r, n, P) = O(rn^2 \log(n)/P \cdot \gamma + rn^2/P \cdot \beta + \alpha)$$

- replicating the filter would be faster for sufficiently faster $r, m$

$$T_{\text{r2D-MM}}(r, m, n, P) = O(rm^2 n^2/P \cdot \gamma + m^2 \cdot \beta + \alpha)$$

- if we use 2D FFT to compute each 4D block, the cost becomes

$$O((n\sqrt{r/P} + m)^2 \log(n) \cdot \gamma + m^2 \cdot \beta + \alpha)$$

- generally, we can observe that the transition from 1D to 2D convolutions does not significantly affect the parallelization strategies and communication cost, a 2D CNN is about as hard as a 1D CNN with datasets of size $n^2$ and a filter of size $m^2$