THE DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF FLORIDA

The Deep Neural Networks: Formulation and Implementation

Dihong Gong
May 14, 2015

1 Model

We model the deep neural networks as a stack of single-layer feed-forward neural networks, where each layer is modelled a function $f_\theta(x) : R^D \rightarrow R^d$ transforming a set of feature maps $X = \{X_1 \dots X_m\}$ into another set of feature maps $Y = \{Y_1 \dots Y_n\}$. The dimension of input feature maps is $h_X \times w_X$ and the dimension of the output feature maps is $h_Y \times w_Y$. This model can construct a wide variety of feed-forward networks, such as Multiple Layer Perceptrons (MLP), GoogleNet, Deep Boltzmann Machines (DBM) and Deep Autoencoders. The figure 2.2 shows a mapping example.

2 Formulations

The model in section 1 consists of a series of connecting blocks mapping one set of feature maps to another. Different networks have different connecting blocks that are connected in different manners. The blocks, which we refer as "functional units", represent a wide variety of operations such as convolution, response normalization, dropout, etc.

The constructed networks are trained with gradient-based algorithms such as stochastic gradient descend, using back propagation. The back propagation consists of two phases: forward signal passing and backward error propagation. Thus, any functional units $f_\theta(x) : R^D \rightarrow R^d$ can be identified by triple $\left(f_\theta(x), \frac{\partial f_\theta(x)}{\partial \theta}, \frac{\partial f_\theta(x)}{\partial x}\right)$. In this section, we will formulate triples for different types of functional units.

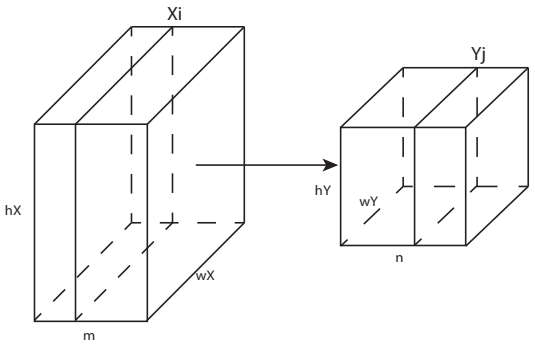

Figure 1.1: An illustration for a functional unit.

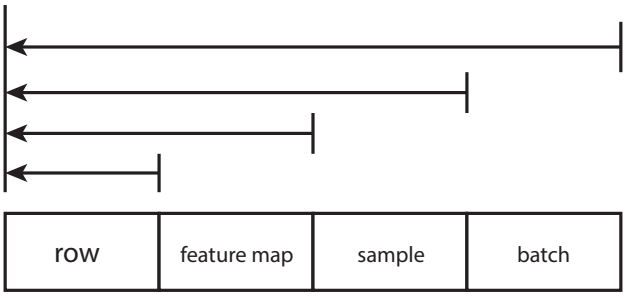

Figure 2.1: Feature maps vectorization.

2.1 Input

The Input unit admits training samples, which initiates the feature maps.

Notations	Values	Dimension	Comments
Input	X	$R^{M\times h_X\times w_X}$	The input training signals
Parameter	$mean(X)$	$R^{M\times h_X\times w_X}$	Mean-center normalization

2.2 Output

The output unit calculates the cost of the input based on supervision signals.

2.2.1 softmax

Notations	Values	Dimension	Comments
Input	X and labels T	$X\in R^{D\times 1\times 1}$, $T\in R^{D\times 1\times 1}$	$T_d=1$ if the sample is of class d D is equal to number of classes
$E=f(x)$	$-\sum_{d=1}^{D} T_d log P_d$	scalar	$P_d=\frac{e^{X_d}}{\sum_{j=1}^{D} e^{X_j}}$
$\frac{\partial E}{\partial \theta}$	N/A	N/A	No parameters
$\frac{\partial E}{\partial X_d}$	P_d-T_d	$R^{1\times 1\times D}$	$d=1\ldots D$

2.3 Full

The fully-connected unit transmits input signals via densely connected networks.

Notations	Values	Dimension	Comments
Input	X	$X\in R^{M\times h_X\times w_X}$	M is the number of feature maps. $h_X\times w_X$ is dimension of feature maps.
Parameters	W, b	$W\in R^{Q\times P}$ $b\in R^{Q\times 1}$	$P=M\times h_X\times w_X$ is # input variables. Q is # output variables.
$Y=f(x)$	$h(W\,vec(X)+b)$	$Q\times 1$	$vec(\cdot)$ is a vectorization operator. Vectorization is in row-major manner (Figure 2.1).
$\frac{\partial E}{\partial Y}\cdot\frac{\partial f(x)}{\partial b_q}$	$\frac{\partial E}{\partial Y_q}\odot h^{'}(x)$	$Q\times 1$	$q=1\ldots Q$ Operator $\odot$ represents element-wise product. Tanh: $h(x)=tanh(x), h^{'}(x)=1-Y_q^2$ ReLU: $h(x)=max(0,x), h^{'}(x)=\delta(Y_q>0)$
$\frac{\partial E}{\partial Y}\cdot\frac{\partial f(x)}{\partial W_{qp}}$	$\frac{\partial E}{\partial Y}\cdot\frac{\partial f(x)}{\partial b_q}\otimes vec(X)$	$Q\times P$	Operator $\otimes$ represents outer product $vec(X)\in R^{1\times P}$
$\frac{\partial E}{\partial Y}\cdot\frac{\partial f(x)}{\partial X}$	$\left(\frac{\partial E}{\partial Y}\cdot\frac{\partial f(x)}{\partial b_q}\right)^T\cdot W$	$1\times P$	$d=1\ldots D$

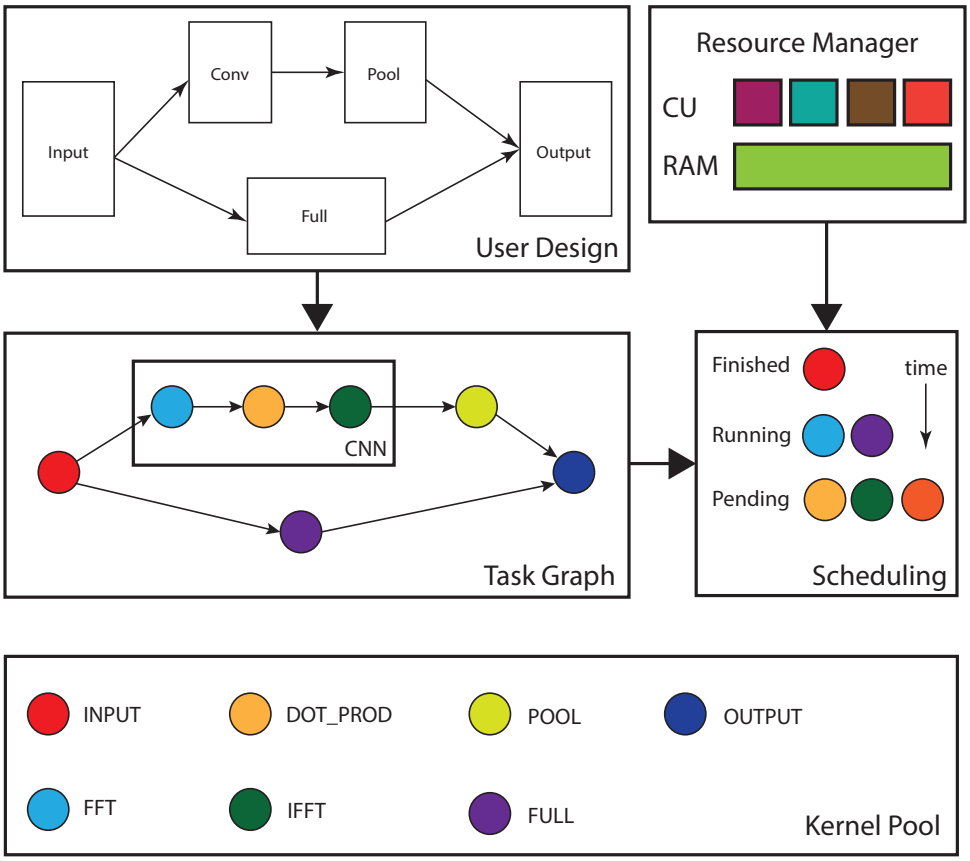

Figure 2.2: An illustration for OpenCL implementation paradigm.

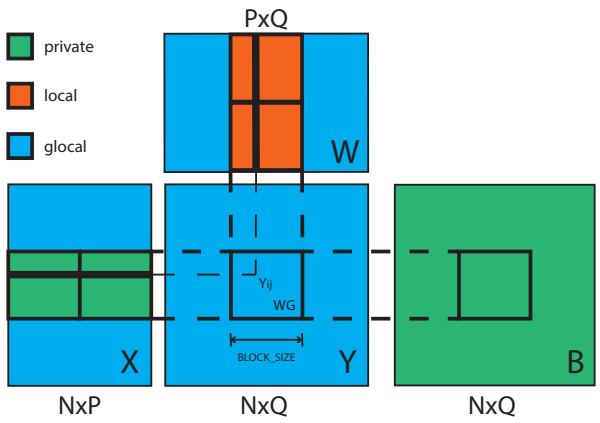

Figure 3.1: OpenCL implementation for forward pass of a fully-connected unit.

3 Implementations

In this part, we elaborate the details about implementations. Specifically, the following problems will be addressed:

- Map a feed-forward neural network into a sequence of tasks.
- Implement the tasks as OpenCL kernels.
- Estimate the runtime resource requirements of each task.
- Schedule the tasks base on availability of resources and task dependency.
- Dynamically allocate/deallocate resources (e.g. memory and compute units).
- Benchmarking the kernel implementation and scheduling algorithm.

3.1 Functional Units

3.1.1 FULL_FORWARD_TANH

```
__kernel void full_forward_tanh(__global float * X,
                                __global float * W,
                                __global float * B,
                                __constant N,
                                __constant P,
                                __constant Q,
                                __global float* Y)
```

This kernel implements forward pass of fully-connected linear layer with $tanh$ activation:

$$Y = tanh(WX + B) \tag{3.1}$$

X[In]: $N \times P$ matrix, containing all feature maps of N samples, with each row vector containing one sample whose vectorization is shown in Figure 2.1.
W[In]: $P \times Q$ matrix, network weights.
B[In]: $1 \times Q$ vector, network biases.
Y[Out]: $N \times Q$ matrix, output of N samples passing through the unit.
The Figure 3.1 illustrates the OpenCL parallel implementation of this functional unit.

3.1.2 FULL_BACKWARD_TANH

```
__kernel void full_forward_tanh(__global float * X,
                                __global float * W,
                                __global float * B,
                                __global float * Y,
                                __constant N,
                                __constant P,
                                __constant Q,
                                __global float* dEdY,
                                __global float* dEdB,
                                __global float* dEdW,
                                __global float* dEdX)
```

This kernel implements backward pass of fully-connected linear layer with $tanh$ activation. It calculates the derivatives $w.r.t.$ parameters and inputs.
X[In]: $N \times P$ matrix, containing all feature maps of N samples, with each row vector containing one sample whose vectorization is shown in Figure 2.1.
W[In]: $P \times Q$ matrix, network weights.
B[In]: $1 \times Q$ vector, network bias.
Y[In]: $N \times Q$ matrix, calculated outputs in the forward pass.
dEdY[In]: $N \times Q$ matrix, each row represents derivatives of one sample.
dEdB[Out]: $1 \times Q$ vector, the $\frac{\partial E}{\partial B}$.
dEdW[Out]: $P \times Q$ matrix, the $\frac{\partial E}{\partial W}$.
dEdX[Out]: $N \times P$ matrix, each row represents derivatives $w.r.t.$ inputs based on one sample.
Implementation details:

$$\text{DOT\_PROD\_MEAN\_1}\left(\frac{\partial E}{\partial Y} \odot (1 - Y^2)\right) \rightarrow \left(\frac{\partial E}{\partial B}, \frac{\partial E}{\partial B_{ex}}\right) \tag{3.2}$$

$$\text{OUT\_PROD}\left(\frac{\partial E}{\partial B} \otimes X\right) \rightarrow \frac{\partial E}{\partial W} \tag{3.3}$$

$$\text{INN\_PROD}\left(\frac{\partial E}{\partial B_{ex}} \cdot W^T\right) \rightarrow \frac{\partial E}{\partial X} \tag{3.4}$$

3.1.3 SOFTMAX

```
float softmax(float * X, float * T, float * dEdX)
```

This unit is implemented in the host, instead of device.
X[In]: $N \times D$ matrix, representing the input into the softmax unit, with N samples.
T[In]: $N \times 1$ vector, where $T[n]$ is index of the true class for the n^{th} sample.
dEdX[In]: $N \times D$ matrix, representing $\frac{\partial E}{\partial X}$.
Return value: It returns the negative log-likelihood $E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} T_{dn} log P_{dn}$.