A Computational Viewpoint for Deep Learning

Liangliang Cao
Jan 28, 2014

http://llcao.net/cu-deeplearning15/
Outline

• An abstract view of deep network

• An abstract view of deep network solver

• Three cases
  – Logistic regression
  – Multiple-layer perceptron (MLP)
  – Convolutional neural network (CNN)

• Q/A; Discussions on course project ideas
Homework and Course Arrangement

• Very easy homework. Deadline passed. Late submissions will be NOT accepted.

• Those who did a good hw#1 will be notified

• Please consider dropping the course if you fail with hw#1
  – Coz you will receive a VERY low score with the course going

• Homework will be explained in class#4

• Why do we assign this homework?
Outline

• **An abstract view of deep network**

• **An abstract view of deep network solver**

• **Three case studies**
  – Logistic regression
  – Multiple-layer perceptron (MLP)
  – Convolutional neural network (CNN)

• **Q/A; Discussions on course project ideas**
An abstract view of deep network

- Estimate the output
  \[ o_1 = L_1(x) \]
  \[ o_2 = L_2(L_1(x)) \]
  \[ \ldots \]
  \[ o_5 = L_5(L_4(L_3(L_2(L_1(x)))))) \]

- Compute the loss function
  \[ C = \text{Loss}(o_5, y) \]

- Compute the gradient
  \[ \frac{\partial C}{\partial o_i} = \frac{\partial C}{\partial o_{i+1}} \frac{\partial o_{i+1}}{\partial o_i} \]
  \[
  \frac{\partial C}{\partial o_1} = \frac{\partial C}{\partial o_5} \cdot \frac{\partial o_5}{\partial o_4} \cdot \frac{\partial o_4}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1}
  \]
An abstract view of deep network (2)

• Estimate the output (Forward propagation)

\[ o_5 = L_5( L_4( L_3( L_2( L_1(x) ) ) ) ) \]

• Compute the gradient (Backward propagation)

\[ \frac{\partial C}{\partial o_1} = \frac{\partial C}{\partial o_5} \cdot \frac{\partial o_5}{\partial o_4} \cdot \frac{\partial o_4}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \]
An abstract view of deep network (3)

• Suppose a layer is in the form of

$$o_l = L_l(x) = f_l(w^T x + b)$$

• We can compute the gradients s.t. parameters

$$\frac{\partial C}{\partial w} = \sum_i \frac{\partial C}{\partial o_l} \cdot f_l' \cdot x_i \quad \frac{\partial C}{\partial b} = \sum_i \frac{\partial C}{\partial o_l} \cdot f_l'$$

• Updating parameters by gradient descent

$$w \leftarrow w - \alpha \frac{\partial C}{\partial w} \quad b \leftarrow b - \alpha \frac{\partial C}{\partial b}$$
• There are many ways to define layers and cost functions

• Layer definitions may differ from field to field
  – Computer vision
  – NLP
  – Speech
  – ...

• But there are only **three key steps** in deep network
1. Forward propagation

\[ o_5 = L_5( L_4( L_3( L_2( L_1(x) )) )) ) \]

2. Backward propagation

\[ \frac{\partial C}{\partial o_1} = \frac{\partial C}{\partial o_5} \cdot \frac{\partial o_5}{\partial o_4} \cdot \frac{\partial o_4}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \]

3. Updating

\[ w \leftarrow w - \alpha \frac{\partial C}{\partial w} \quad b \leftarrow b - \alpha \frac{\partial C}{\partial b} \]
Outline

• An abstract view of deep network

• **An abstract view of deep network solver**

• Three cases
  – Logistic regression
  – Multiple-layer perceptron (MLP)
  – Convolutional neural network (CNN)

• Q/A; Discussions on course project ideas
Brainstorming question:

Deep neural networks have been studied by Hinton, LeCun, Bengio, Schmidhuber, and many others since 1990s.

Why *only recently* it becomes hot (again)?
My understanding

Two reasons

- In 1990s we do not have large scale datasets
- In 2000s we have not only large scale datasets but also powerful computers (w/o GPUs) to compute them

But what is the algorithm to learn from the large scale datasets?
A typical optimization problem

Very often, a machine learning model with parameter $w$ aims to minimize

$$\min_w \frac{1}{N} \sum_{i=1}^{N} C(x_i, y_i | w)$$

with the hope that the cost in the testing set $T$ will be small too.

$$\frac{1}{|T|} \sum_{j \in T} C(x_j, y_j | w)$$
A typical optimization problem

Very often, a machine learning model with parameter $w$ aims to minimize

$$\min_w \frac{1}{N} \sum_{i=1}^{N} C(x_i, y_i | w)$$

If $C$ is convex and continuous, we can try

1) gradient descent

$$w \leftarrow w - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial C(x_i, y_i | w)}{\partial w}$$
A typical optimization problem

Very often, a machine learning model with parameter \( \mathbf{w} \) aims to minimize

\[
\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} C(x_i, y_i | \mathbf{w})
\]

If \( C \) is convex and continuous, we can try

1) gradient descent

2) Newton’s method and its variants

\[
\mathbf{w}_{t+1} = \mathbf{w}_t - \Gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} C(x_i, y_i | \mathbf{w})
\]

inverse of Hessian
A typical optimization problem

Very often, a machine learning model with parameter $\mathbf{w}$ aims to minimize

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} C(x_i, y_i|\mathbf{w})$$

If $C$ is convex and continuous, we can try
1) gradient descent
2) Newton’s method and its variants
3) Coordinate descent
4) ...
A typical optimization problem

Very often, a machine learning model with parameter $\mathbf{w}$ aims to minimize

$$
\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} C(x_i, y_i | \mathbf{w})
$$

when $N$ is big, we can see that

- The gradient $\frac{1}{N} \sum_{i=1}^{N} \frac{\partial C(x_i, y_i | \mathbf{w})}{\partial \mathbf{w}}$ becomes very expensive.
- Even worse, we may not be able to load all $(x_i, y_i)$ in to memory!
Stochastic Gradient Descent (SGD)

Idea: estimate the gradient on a randomly picked sample

- Gradient descent

\[ w \leftarrow w - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial C(x_i, y_i | w)}{\partial w} \]

- Stochastic gradient descent

\[ w \leftarrow w - \alpha_t \frac{\partial C(x_t, y_t | w)}{\partial w} \]

**Theoretical** requirement for convergence:

\[ \sum_{t} \alpha_t^2 < \infty \quad \sum_{t} \alpha_t = \infty \]

*in deep learning practice we just choose a small rate and then decrease it*
Stochastic gradient descent (SGD) on single machines is much easier to program than many optimization methods!
Example: traditional SVM optimization (SMO)

Classifier  \[ F(x) = \sum_{i=1}^{N} \alpha_i K(x,x_i) \]  with the cost function

\[
\text{arg max}_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i,x_j)
\]

\[ 0 \leq \alpha_i \leq \frac{C}{N}, \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \]

SMO algorithm:

1. Heuristically picks 2 variables, say \( \alpha_i, \alpha_j \), and freeze the other variables.
2. Analytically update \( \alpha_i, \alpha_j \)
3. Iterate until converges.

You may write hundreds or even thousands of lines of codes to implement SMO.
But you can implement a stochastic SVM in 10 lines

```
function w = pegasos_SVM(X,Y,lambda,nepochs)
    [m,d] = size(X); w = zeros(d,1); t = 1;
    for (i=1:nepochs)  % iterations over the full data
        for (tau=1:m)   % pick a single data point
            if (Y(tau)*X(tau,:) * w < 1)  % data too close or wrongly separated
                w = (1-1/t)*w + 1/(lambda*t)*Y(tau)*X(tau,:);  
            else
                w = (1-1/t)*w;
            end
        end
        t=t+1;  % increment counter
    end
end
```

Can you find the problem of this code?

*Pegasos SVM by Shai Shalev-Shwartz*
Philosophy of SGD

• One iteration of SGD is way faster than one iteration of GD

• SGD relies on randomness to reduce the cost although it may not find the global minimum

• But SGD fits better data + local minimum than global minimum, esp when
  – Cost function is not convex
  – Training set is not the same distribution as testing set
SGD as a typical deep learning solver

```
for patch = uttl
    self.mlp_.forward(xs(patch,:));
    self.mlp_.backward(ys(patch,:));
    self.mlp_.update();
end
```

For every layer, compute the gradient and update.

```
```
SGD and GPUs

```python
for patch = uttl
    self.mlp_.forward(xs(patch,:))
    self.mlp_.backward(ys(patch,:))
    self.mlp_.update()
end
```

For every layer, compute the gradient and update.


- Within every batch, SGD is mainly matrix multiplication: perfect task for GPU!
- Beyond every batch, SGD is sequential: so multiple GPUs may help!
Outline

• An abstract view of deep network

• An abstract view of deep network solver

• Three case studies
  – Logistic regression
  – Multiple-layer perceptron (MLP)
  – Convolutional neural network (CNN)

• Q/A; Discussions on course project ideas
Logistic regression

Logistic regression = one layer neural network + neg-loglikelihood cost

LR can be used separately or as the last layer of MLP.

Consider the binary case:

- Decision function: \( p(x) = Pr(y = 1|x) = \frac{\exp(\beta^T x)}{1 + \exp(\beta^T x)} \)

- Cost \( \sum_{i=1}^{N} C(x_i, y_i) = \sum_{i=1}^{N} [y_i \log p(x) + (1 - y_i) \log(1 - p(x))] \)
Solving logistic regression (traditional methods)

• Compute the gradient
\[ L(\beta) = \sum_{i=1}^{N} C(x_i, y_i) \]

• Solver 1: gradient descent
\[ \frac{\partial L(\beta)}{\partial \beta} = x^T(y - p) \]
\[ \beta = \beta - \alpha \frac{\partial L(\beta)}{\partial \beta} \]

• Solver 2: Newton’s method
\[ H = \nabla^2 L(\beta) = -x^TWWx \]
\[ \beta = \beta - H^{-1} \frac{\partial L(\beta)}{\partial \beta} \]

\( W \) is a diagonal matrix with the element as \( p(x)(1-p(x)) \).
Analyzing the traditional solver

• Solver 1: gradient descent

\[ \frac{\partial L(\beta)}{\partial \beta} = X^T (y - p) \]

\[ \beta = \beta - \alpha \frac{\partial L(\beta)}{\partial \beta} \]

• Solver 2: Newton’s method

\[ H = \nabla^2 L(\beta) = -X^T WX \]

\[ \beta = \beta - H^{-1} \frac{\partial L(\beta)}{\partial \beta} \]

Newton’s method converges faster than gradient descent, but it requires more time to compute the Hessian matrix.

And Newton’s method is expensive in large scale.
Comparing optimization methods for Logistic Reg.

<table>
<thead>
<tr>
<th></th>
<th>Global minimum</th>
<th>Memory consumption</th>
<th>Convergence speed</th>
<th>More criteria?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Newton’s method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SGD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

- An abstract view of deep network
- An abstract view of deep network solver

- Three case studies
  - Logistic regression
  - Multiple-layer perceptron (MLP)
  - Convolutional neural network (CNN)

- Q/A; Discussions on course project ideas
Multi-layer perceptron

• Generalized from single layer perceptron

\[ f(x) = \begin{cases} 
1 & \text{if } w^T x + b > 0 \\
0 & \text{otherwise} 
\end{cases} \]

There is an interesting story between single layer perceptron and multi-layer perceptron. See [Minsky and Papert, 1969]
Multi-layer perceptron

The last layer is often logistic regression
The hidden layer is a perceptron with nonlinear function

\[ \phi(w^T x + b) \]

\( \phi \) can be sigmoid, tanh, or rectifier
## Comparing optimization methods for MLP

<table>
<thead>
<tr>
<th></th>
<th>Global minimum</th>
<th>Memory consumption</th>
<th>Convergence speed</th>
<th>More criteria?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Newton’s method</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SGD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tricks to train MLP with SGD

• Initialize the neurons with random weights

• Randomly shuffle the data

• Use a batch in every SGD iteration

• Choose the learning rate by multiple trials.

More details will be covered by Feb 11 class.
Outline

• An abstract view of deep network

• An abstract view of deep network solver

• Three case studies
  – Logistic regression
  – Multiple-layer perceptron (MLP)
  – Convolutional neural network (CNN)

• Q/A; Discussions on course project ideas
Convolutional Layer

- Almost all image filters can be represented as 2D convolution

\[ y[m, n] = x[m, n] \ast h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] \cdot h[m-i, n-j] \]
## A Nice Illustration of Convolution

### Image

```
1_1 1_0 1_1 0 0
0_0 1_1 1_0 1 0
0_0 0_0 1_1 1 1
0 0 1 1 0
0 0 1 1 0
```

### Convolved Feature

```
4
```

*Gif picture courtesy to ufldl.Stanford.edu/wiki*
Forward and Backward Propagation for Conv Layer

- **Forward propagation**

\[
y(s,j) = \sum_{i \in f} x(s,i) \ast w(j,i)
\]

- **Backward propagation**

\[
\frac{\partial L}{\partial x(s,i)} = \sum_{j \in f'} \frac{\partial L}{\partial y(s,j)} \ast w(j,i) \\
\frac{\partial L}{\partial w(j,i)} = \sum_{s \in S} \frac{\partial L}{\partial y(s,j)} \ast x(s,i)
\]
Why Deep CNN Is Powerful?

Conceptually, three reasons:

1. Many many filters
2. A number of layers
3. Conv + Pooling lead to local invariance
An Example of Deep CNN

Input: 224*224*3

(11 x 11 x 96) conv3D

(4 x 4) max pooling

1st Conv Output 55*55*96

more layers of convolution

2 fully connected layers

Classification output

Krizhevsky, Sutskever and Hinton
1st place, ImageNet
LSVRC 2012
Local Invariance

• Convolution is translation invariant:
  – any translation invariant operation can be represented as a convolution.

• Convolution + max pooling can find local invariant features
Number of filters in the Alex’ CNN

• Filters in 1st conv layer: 3 x 96 (neighborhood 11 x 11)
• Filters in 2nd conv layer: 96 x 128 (neighborhood 5 x 5)
• Filters in 3rd conv layer: 256 x 384 (neighborhood 3 x 3)
• Filters in 4th conv layer: 384 x 192 (neighborhood 3 x 3)
• Filters in 5th conv layer: 384 x 128 (neighborhood 3 x 3)

Millions of parameters!
Make the CNN Even Deeper

- [Simonyan and Zisserman 2014] suggests to use replace one conv layer (big filter size) with several concatenated conv layers (small filter size).

- [Szegedy et al 2014] proposes to replace one conv layer with concatenated inceptions.
From 1D convolution to 2D convolution

1D convolution is widely used in speech and NLP
- Computational complexity: $O(M \times m)$

2D/3D convolution is mainly used for image/video
- Computational complexity: $O(M \times N \times m \times n)$

Convolution with 2D Gaussian is efficient by separating 2D into 2*1D
- Computational complexity $O(M \times N \times m \times 2)$
- But most CNN filters cannot be separated
How Hard to Implement 2D Convolution?

• It is not super hard at the first glance

```plaintext
for w in 1..W
    for h in 1..H
        for x in 1..K
            for y in 1..K
                for m in 1..M
                    for d in 1..D
                        output(w, h, m) += input(w+x, h+y, d) * filter(m, x, y, d)
                    end
                end
            end
        end
    end
end
```

• But we overlooked cache, parallelism, or any fancy SSE2 command
• And it becomes 10 times tricky with GPUs!
Three Ways to Implement Fast Convolution in GPU

1. Directly implement convolution algorithm
   • Extremely demanding with memory, data transportation, and model sharing
   • Very challenging for GPU programming skills

2. Change convolution to matrix multiplication
   • Make good use of existing BLAS or cuBLAS library
   • Maybe memory demanding

3. Use FFT instead of directly convolution
   • Convolution in image domain is equivalent to multiplication in frequency domain
   • Performance may depends on the image/filter size.
What kind of projects would you like to take in this class?

- Theory
- Applications
  1. NLP
  2. Vision
  3. NLP + Vision
  4. Your own data or problem?