Practical Deep Neural Networks

GPU computing perspective

Introduction

Yuhuang Hu Chu Kiong Loo

Advanced Robotic Lab
Department of Artificial Intelligence
Faculty of Computer Science & IT
University of Malaya

Outline

Objectives

- Light introduction of numerical computation.
- Fundamentals of Machine Learning.
- Support Vector Machine, Softmax Regression.
- Feed-forward Neural Network.
- Convolutional Networks.
- Recurrent Neural Networks.

Prerequisites

- ⋆ Basic training in Calculus
- ⋆ Basic training in Linear Algebra
  - Matrix operations
  - Matrix properties: transform, rank, norm, determinant, etc
  - Eigendecomposition, Singular Value Decomposition.
- ⋆ Basic programming skills
  - If-else conditioning
  - Loops
  - Function, class, library
  - Source code control: Git (optional)

References

- Deep Learning: An MIT Press book in preparation
  Main reference in this workshop, still in development, awesome structure, awesome contents.
- Machine Learning: A probabilistic perspective
  One of the best Machine Learning books on the market.
- CS231n Convolutional Neural Networks for Visual Recognition Notes
  Nice structured, well written, loads of pictures.
- CS229 Machine Learning Course Materials
  For basic knowledge, well written, easy-to-read.

Software and Tools

- Ubuntu 14.04
- CUDA Toolkit 7
- Python 2.7.9 (Why not 3.*?)
- Theano
- numpy, scipy, etc
- Eclipse+PyDev

Reading List

A reading list is prepared for this workshop, all reading materials can be found at:
`http://rt.dgyblog.com/ref/ref-learning-deep-learning.html`

The list keeps expanding!!

Scalar, Vector, Matrix and Tensor

Scalars A scalar is a single number.

Vectors A vector is an array of numbers.

Matrices A matrix is a 2-D array of numbers.

Tensors A tensor is an array of numbers arranged on a regular grid with a variable number of axes.

Matrix Operations and Properties

Identity	$\boldsymbol{I}_n \in \mathbb{R}^{n \times n}$, $\boldsymbol{I}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{I} = \boldsymbol{A}$.
Transpose	$(\boldsymbol{A}^\top)_{ij} = A_{ji}$.
Addition	$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ where $C_{ij} = A_{ij} + B_{ij}$.
Scalar	$\boldsymbol{D} = a \cdot \boldsymbol{B} + c$ where $D_{ij} = a \cdot B_{ij} + c$.
Multiply	$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ where $C_{ij} = \sum_k A_{ik} B_{kj}$.
Element-wise Product	$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$ where $C_{ij} = A_{ij} B_{ij}$.
Distributive	$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{A}\boldsymbol{C}$.
Associative	$\boldsymbol{A}(\boldsymbol{B}\boldsymbol{C}) = (\boldsymbol{A}\boldsymbol{B})\boldsymbol{C}$.
Transpose of Matrix Product	$(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$.

Inverse Matrices

The *matrix inverse* of $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^{-1}$, and it is defined as the matrix such that

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = I_n$$

if $\boldsymbol{A}$ is not singular matrix.
If $\boldsymbol{A}$ is not square or is square but singular, it's still possible to find it's generalized inverse or pseudo-inverse.

Norms

We usually measure the size of vectors using an L^p norm:

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

There are few commonly used norms

- L_1 norm

$$\|\boldsymbol{x}\|_1 = \sum_i |x_i|.$$

- l_∞ (Max norm)

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|.$$

- Frobenius norm: Measure the size of matrix, analogy to the L^2 norm.

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$$

Unit vector, Orthogonal, Orthonormal, Orthogonal Matrix

A *unit vector* is a vector with *unit norm*:

$$\|\boldsymbol{x}\|_2 = 1$$

A vector $\boldsymbol{x}$ and $\boldsymbol{y}$ are *orthogonal* to each other if $\boldsymbol{x}^\top \boldsymbol{y} = 0$. In $\mathbb{R}^n$, at most n vectors may be mutually orthogonal with nonzero norm. If vectors are not only orthogonal but also have unit norm, we call them *orthonormal*.
An *orthogonal matrix* is a square matrix whose rows are mutually orthonormal and whose columns are mutually orthonormal:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}$$

This implies that

$$\boldsymbol{A}^{-1} = \boldsymbol{A}^\top$$

Eigendecomposition

An *eigenvector* of a square matrix $\boldsymbol{A}$ is a non-zero vector $\boldsymbol{v}$ such that

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

where scalar λ is known as the *eigenvalue* corresponding to the eigenvector. If $\mathbf{v}$ is an eigenvector of $\boldsymbol{A}$, so is any rescaled vector $s\boldsymbol{v}$ for $s \in \mathbb{R}, s \neq 0$. Therefore, we usually only look for unit eigenvectors.
We can represent the matrix $\boldsymbol{A}$ using an *eigendecomposition*.

$$\boldsymbol{A} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$$

where $\boldsymbol{V} = [\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(n)}]$ (one column per eigenvector) and $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]$.
Every real symmetric matrix can be decomposed into

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}$$

where $\boldsymbol{Q}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix, with λ_{ii} being the eigenvalues.

Singular Value Decomposition (SVD)

SVD is a general purpose matrix factorization method that decompose an $n \times m$ matrix $\boldsymbol{X}$ into:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{W}^\top$$

where $\boldsymbol{U}$ is an $n \times n$ matrix whose columns are mutually orthonormal, $\boldsymbol{\Sigma}$ is a retangular diagonal matrix and $\boldsymbol{W}$ is an $m \times m$ matrix whose columns are mutually orthonormal. Elements alone the main diagonal of $\boldsymbol{\Sigma}$ are referred to as the singular values. Columns of $\boldsymbol{U}$ and $\boldsymbol{W}$ are referred as the *left-singular vectors* and *right-singular vectors* of $\mathbf{X}$ respectively.

- Left-singular vectors of $\boldsymbol{X}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top$.
- Right-singular vectors of $\boldsymbol{X}$ are the eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}$.
- Non-zero singular values of $\boldsymbol{X}$ are the square roots of the non-zero eigenvalues for both $\boldsymbol{X}\boldsymbol{X}^\top$ and $\boldsymbol{X}^\top\boldsymbol{X}$.

Trace, Determinant

The trace operator is defined as

$$\mathrm{Tr}(\boldsymbol{A}) = \sum_i A_{ii}$$

Frobenius norm in trace operator:

$$\|\boldsymbol{A}\|_F = \sqrt{\mathrm{Tr}(\boldsymbol{A}^\top \boldsymbol{A})}$$

The determinant of a square matrix, denoted $\det(\boldsymbol{A})$ is a function mapping matrices to real scalars. The determinant is equal to the product of all matrix's eigenvalues.

MNIST

MNIST dataset contains 60,000 training samples and 10,000 testing samples (Lecun, Bottou, Bengio, & Haffner, 1998). Each sample is a hand-written digit from 0-9.

CIFAR-10

CIFAR-10 dataset consists 60,000 images in 10 classes (Krizhevsky, 2009). There are 50,000 training images and 10,000 testing images.

Dataset: How to split the data?

- If dataset has been splitted initially, please keep that way.
- If dataset comes as a whole, usually use 60% as trianing data, 20% as cross-validation and 20% as testing data.
- Another alternative is using 70% as training data, 30% as testing data.

Dataset: Cross validation

- Cross validation is used to keep track the performance of learning algorithm (overfitting? underfitting?).
- Used to be a small portion of training dataset, randomly selected.
- Serve as testing data when testing data is not visible.
- Help to choose and adjust parameters of the learning model.

Mean subtraction, Unit Variance

Let μ as mean image

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

And then subtract mean image on every image:

$$x^{(i)} = x^{(i)} - \mu$$

Let

$$\sigma_j^2 = \frac{1}{m}\sum_{i}\left(x_j^{(i)}\right)^2$$

and then

$$x_j^{(i)} = \frac{x_j^{(i)}}{\sigma_j}$$

This is also called PCA whitening if it's applied to rotated data, the variance here can also be viewed as eigenvalues.

PCA

Principle Components Analysis (PCA) is a dimension reduction algorithm that can be used to significantly speed up unsupervised feature learning algorithm. PCA will find a lower-dimensional subspace onto which to project our data.
Given a set of data $\boldsymbol{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in \mathbb{R}^n$, covariance is computed by

$$\Sigma = \frac{\boldsymbol{X}\boldsymbol{X}^\top}{m}$$

Use eigendecomposition, we can obtain the eigenvectors $\boldsymbol{U}$:

$$\boldsymbol{U} = \begin{bmatrix} | & | & \cdots & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & \cdots & | \end{bmatrix}$$

and corresponding eigenvalues $\boldsymbol{\Lambda} = \{\lambda_1, \ldots, \lambda_n\}$. Note that usually $\boldsymbol{\Lambda}$ is a sorted in descending order.

PCA

Columns in $\boldsymbol{U}$ are principle bases, so that we can rotate our data to the new bases:

$$\boldsymbol{X}_{\text{rot}} = \boldsymbol{U}^T \boldsymbol{X};$$

Noted that $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{X}_{\text{rot}}$.
To one of the data x, we can reduce the dimension by

$$\tilde{x} = \begin{bmatrix} x_{\text{rot},1} \\ \vdots \\ x_{\text{rot},k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} x_{\text{rot},1} \\ \vdots \\ x_{\text{rot},k} \\ x_{\text{rot},k+1} \\ \vdots \\ x_{\text{rot},n} \end{bmatrix} = x_{\text{rot}}$$

The resulted $\tilde{\boldsymbol{X}}$ can approximate the distribution of $\boldsymbol{X}_{\text{rot}}$ since it's dropping only small components. And we reduced $n - k$ dimensions of the original data

PCA

We can recover the approximation of original signal $\hat{\boldsymbol{X}}$ from $\tilde{X}$:

$$\hat{x} = U \begin{bmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

PCA: Number of components to retain

To determine k, we usually look at the *percentage of variance retained*,

$$p = \frac{\sum_{i=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j} \tag{1}$$

One common heuristic is to choose k so as to retain 99% of the variance ($p \geq 0.99$).

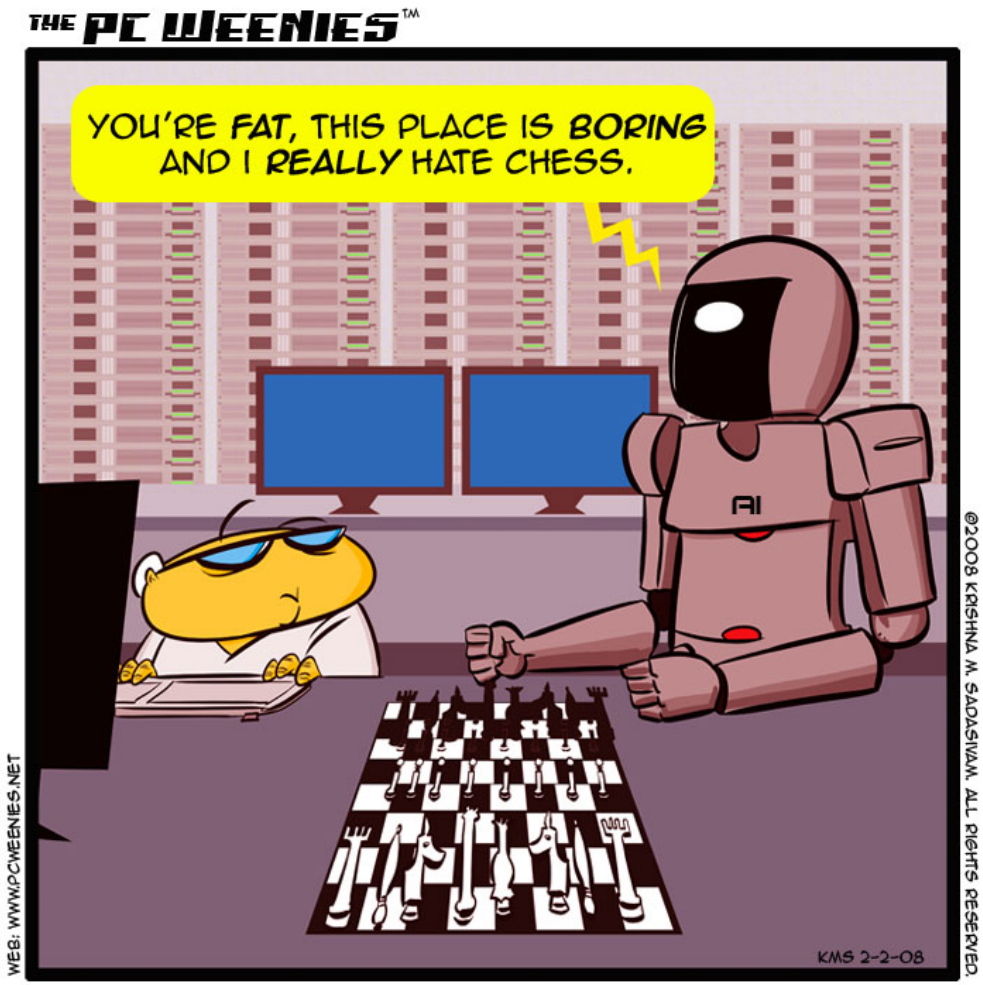
ZCA Whitening

ZCA whitening is just a recover version of PCA whitening:

$$x_{\text{ZCA}} = U x_{\text{PCA}} = U \frac{x_{rot}}{\mathbf{\Lambda} + \epsilon}$$

where ϵ is a regularization term. When x takes values around $[-1, 1]$, a value of $\epsilon \approx 10^{-5}$ might be typical.

Q&A



HOW YOU'LL KNOW WHEN YOU'VE TRULY SUCCEEDED IN THE FIELD OF A.I. RESEARCH.

References

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (Master Thesis).

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, Nov). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.