Hardware and Software for Deep Learning

Master Seminar Deep Learning

Maximilian Kaiser

Institut für Statistik
Ludwig-Maximilians-Universität

27.1.2017

Table of Contents

Hardware

CPUs In General
Central Processing Units

- Main component of any PC (and various other things).
  Main competitors: Intel, AMD, TSMC and Qualcomm.
- Processing Speed depends on many things:
  1. Clock speed.
     Low Gigahertz.
  2. May cache interim results in the processors cache (L1<L2<L3).
     Size: Low Megabytes. Extreme speed.
  3. Writes and reads on the Random Access Memory (RAM).
     Size: Low Gigabytes. Very high speed.
- Modern CPUs consist of multiple cores. Clusters consist of multiple multi-core CPUs.
- Standard option for most software is to use the CPU.

CPU in Deep Learning

- Pro
  - Widely spread in terms of hardware and software support.
  - Versatile to complex functions (gradients?)
  - Parallization (even Hyperthreading) is almost standard...
- Con
  - ... but very limited with a small amount of threads.
  - Scalability for clusters depends on various factors.

Short Video Demonstration

GPUs in General
Whats a GPU

- Driven by the video game industry (among others), graphics processing units have been increasing steadily in their performance and versatility. Main competitors: Intel, nVidia and ATI.
- They are in part designed to calculate high dimensional matrix operations, i.e.:

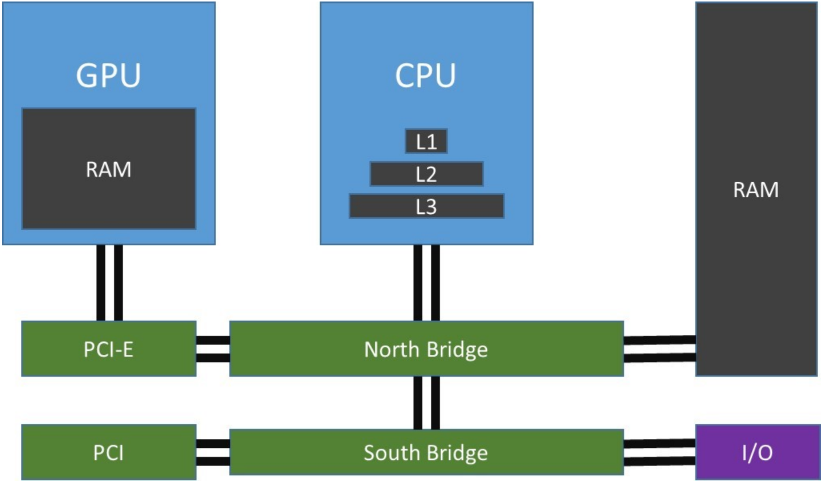
$$\begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{pmatrix} \begin{pmatrix} T \\ U \\ V \\ W \end{pmatrix} = \begin{pmatrix} a_1 T + a_2 U + a_3 V + a_4 W \\ b_1 T + b_2 U + b_3 V + b_4 W \\ c_1 T + c_2 U + c_3 V + c_4 W \\ d_1 T + d_2 U + d_3 V + d_4 W \end{pmatrix}$$

- GPGPU - General Purpose Graphical Processing Unit. (e.g. CUDA).
- A dedicated GPU does not rely on the RAM of the CPU but has it's own memory (low GB). They're clocked at around 1.5 to 2 GHz.
- A GPU consists of many shader cores (in the low thousands). Multi GPU settings are possible in private and commercial sector.

GPUs in Deep Learning

- Pro
  - PARALLELIZATION. Each hidden layer may be computed in parallel.
  - High speed increases.
- Con
  - Scalability of multi-GPU systems still in development.
  - Specialized software is needed.
  - Cost intensive.

CPU and GPU scheme



Specialized Hardware

Double digits are overrated

- Changes in Deep Learning Development:
  - In the "first neural network era", CPUs were rapidly evolving and simultaneously getting cheaper, while the implementation of neural network chips may took up to two years.
  - In this "nn-era", three main factors of the industry turned this trend around:
    1. GPUs are far more powerful and popular.
    2. Parallelization across cores > Single core improvement (Both CPU and GPU).
    3. Low-power devices with new requirements are everywhere (phones).
- It was shown in 1991 that double precision is not necessary for deep learning, see [4].
  - Google developed TPUs specially for TensorFlow machine learning. It uses only 8-Bit Precision and achieved 10x efficiency compared to GPUs.
- VPUs - Visual Processing Units (Movidius)
  FPGAs - Field Programmable Gate Arrays

Model Compression and Dynamic Structures

- Problem: Training and deployment of models are inherently different.
- Solution: Compression
  - Applicable if size is driven by the need to prevent overfitting.
  - Rather use one large model than an ensemble of many, many small ones
- Problem: Calculation speed is much slower on less powerful devices.
- Solution: Dynamic structure
  - Use cascade classifiers. Start with low capacity and high recall. Finish with high precision classifiers.
  - Train a specialized gater neural network that chooses which expert neural network is to be used in each case.

Software

Popular Open Source Software - Python Affiliated

- Theano by the Université de Montréal
  - Python library.
  - Optimized Speed and Stability.
  - Dynamic C code generation.
  - Uses the concepts of computational graphs similarly to TensorFlow.
- TensorFlow by Google Brain
  - Python and C++ Intefaces. Written in Python and C++, as well.
  - We will focus on this later.
- Keras by François Chollet
  - Python library.
  - May be used as Frontend for TensorFlow or Thenao - therefore inherits their properties. Extends both by hyperparameter optimization and more.
  - Focused on meaningful, modular and easily extensible coding.

Keras Example

```
from keras.models import Sequential
from keras.layers import Dense, Activation


model = Sequential()

model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy', optimizer='sgd',
   metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

loss_and_metrics = model.evaluate(X_test, Y_test, batch_size=32)
```

Popular Open Source Software - in General

- Microsoft Cognitive Toolkit (previously CNTK)
  - Became open source in 2016.
  - Claims to have best scalability on the market.
  - Command line, python and C++ interfaces.
- Caffe by the Berkeley Vision and Learning Center.
  - Main goals: Expressive architecture, extensible code, speed and community.
  - Pure C++/Cuda library.
  - Command line, python and matlab interfaces.
  - Used in industry as well as science.
- Torch
  - Developed by a team of Facebook, Twitter and Google Research scientists.
  - Written in C and Lua. Interfaces in C and Lua.

Caffe Example

Data Layer

```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
```

Convolutional Layer

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

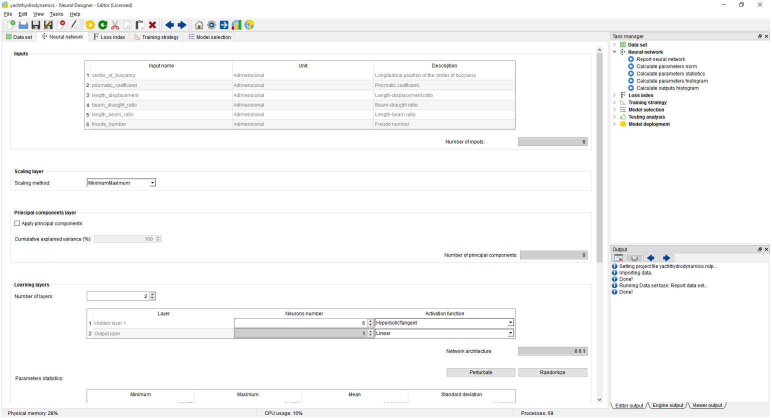
Pooling and Loss

```
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
}
```

Proprietary Software

- Neural Designer
  - Developed from the open source library OpenNN by Artelnics - Written in C++
  - Covers data mining and machine learning tools. Called a "general predictive analytics software".
- Wolfram Mathematica
  - Developed by Wolfram Research - written in Wolfram language, C/C++, Java and Mathematica.
  - Covers many and many more fields of computational mathematics, machine learning, data mining etc.

Examples - Neural Designer



Tensor Flow

Overview

- Grew out of the DistBelief project inside the Google Brain Project.
- Controlled via Python API
- There is also a C++ API for execution while R, Java, Ruby and Go APIs are in the making.
- TF both has CPU and GPU implementations (through CUDA). You can use multiple GPUs through multi towering.
- Not only used for neural networks but especially suited for them, cuDNN (Nvidia Cuda Deep Neural Network library).
- Tensors in this case refer to multidimensional arrays (not the mathematical concept of tensors)

Computation Graph Concept

On the board

Example MNIST in Tensor Flow - R API (1/2)

```
# Load Data
library(tensorflow)
datasets = tf$contrib$learn$datasets
mnist = datasets$mnist$read_data_sets("MNIST-data", one_hot = T)

# Start an interactive Session
sess = tf$InteractiveSession()

# Define (empty) placeholders for inputs
x = tf$placeholder(tf$float32, shape(NULL, 784L))
y_ = tf$placeholder(tf$float32, shape(NULL, 10L))

# Define model parameters as variables
W = tf$Variable(tf$zeros(shape(784L, 10L)))
b = tf$Variable(tf$zeros(shape(10L)))

# Initiate all variables
sess$run(tf$initialize_all_variables())
```

Example MNIST in Tensor Flow - R API (2/2)

```
# Define the model and Loss
y = tf$nn$softmax(tf$matmul(x,W) + b)
cross_entropy = tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y),
    reduction_indices=1L))

# Train the model
optimizer = tf$train$GradientDescentOptimizer(0.5)
train_step = optimizer$minimize(cross_entropy)
for (i in 1:1000) {
  batches = mnist$train$next_batch(100L)
  batch_xs = batches[[1]]
  batch_ys = batches[[2]]
  sess$run(train_step,
           feed_dict = dict(x = batch_xs, y_ = batch_ys))
}

# Evaluate Model
correct_prediction = tf$equal(tf$argmax(y, 1L), tf$argmax(y_, 1L))
accuracy = tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
accuracy$eval(feed_dict=dict(x = mnist$test$images, y_ = mnist$
    test$labels))
```

Benchmarks

Results

- Trial network as seen in your Handout
- Classic MNIST example: 28x28 pictures of number 0-9
- 2 Convolutional Layers and 1 Fully connected
- Batchsize: 50 , Epochs: 1, Train sets: 60000, Test sets: 10000

			TensorFlow		Keras	
System	Processor	RAM	Runtime	Accuracy	Runtime	Accuracy
Virtual Box	1 CPU	12 Gb	383	0.9668	326	0.602
AWS t2.xlarge	4 CPU	16 Gb	334	0.9654	130	0.9756

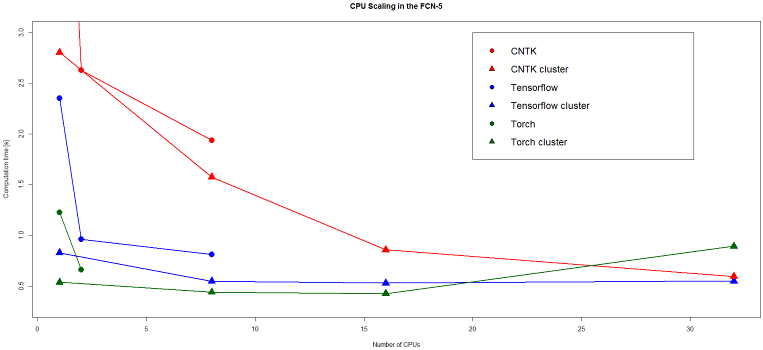
Benchmarking Set Up

Benchmarking State-of-the-Art Deep Learning Software Tools

Computational Unit	Cores	Memory	OS	CUDA
Intel CPU i7-3280	4	64 GB	Ubuntu 14.04	-
Intel CPU E5-2630x2	16	128 GB	CentOS 7.2	-
NVIDIA GTX 980	2048	4 GB	Ubuntu 14.04	7.5
NVIDIA GTX 1080	2560	8 GB	Ubuntu 14.04	8.0
NVIDIA Tesla K80	2496	12 GB	CentOS 7.2	7.5

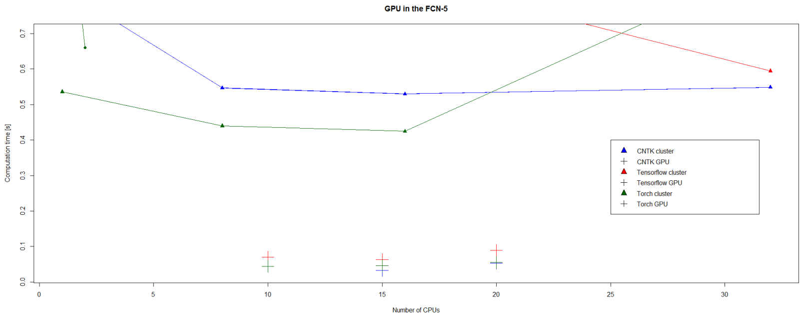
Type	Network	Input	Output	Layers	Parameters
FCN	FCN-5	26,752	26,752	5	55 millions
	FCN-8	26,752	26,752	8	58 millions
CNN	AlexNet	150,528	1,000	4	61 millions
	ResNet-50	150,528	1,000	50	3.8 billions
RNN	LSTM-32	10,000	10,000	2	13 millions
	LSTM-64	10,000	10,000	2	13 millions

Benchmarking Results
FCN-5 - CPUs



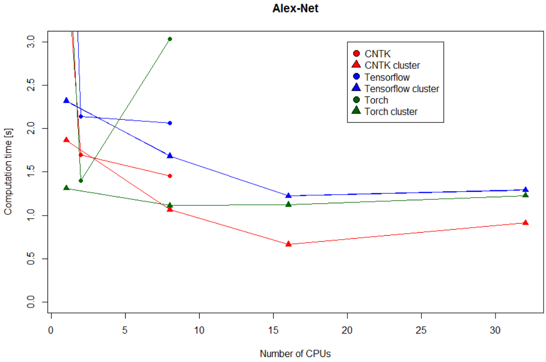
Network	Software	CPU Threads			CPU Server Threads			
		1	2	8	1	8	16	32
FCN-5	CNTK	2.351	.962	.810	.828	.547	.530	.549
FCN-5	TF	7.206	2.626	1.934	2.804	1.574	.857	0.595
FCN-5	Torch	1.227	.661	-	.536	.440	.425	0.892

Benchmarking Results
FCN-5 - GPUs



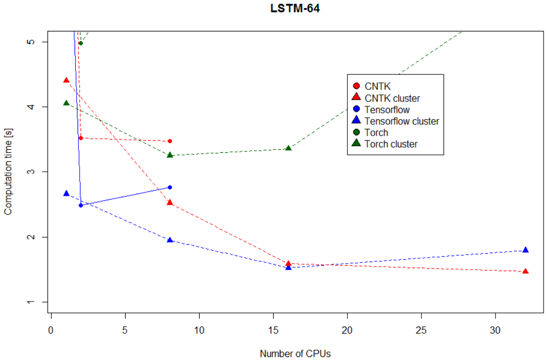
Network	Software	CPU Threads		GPUs		
		16	32	G.980	G.1080	T.K80
FCN-5	CNTK	.530	.549	.044	.033	.053
FCN-5	TF	.857	.595	.070	.063	.089
FCN-5	Torch	.425	.892	.044	.046	.055

Benchmarking Results
AlexNet



Network	Software	CPU Threads		GPUs		
		16	32	G.980	G.1080	T.K80
AlexNet	CNTK	1.223	1.292	.054	.040	.091
AlexNet	TF	.666	.914	.058	-	.086
AlexNet	Torch	1.122	1.229	.038	.033	.081

Benchmarking Results
LSTM-64



Network	Software	CPU Threads		GPUs		
		16	32	G.980	G.1080	T.K80
LSTM-64	CNTK	1.527	1.798	.171	.122	.249
LSTM-64	TF	1.590	1.469	.178	.144	.232
LSTM-64	Torch	3.358	5.815	.269	.194	.407

Sources and Discussion

Sources

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, www.deeplearningbook.org, 2016.
- Shaohuai Shi, Qiang Wang, Pengfei Xu, Xiaowen Chu, *Benchmarking State-of-the-Art Deep Learning Software Tools*, arxiv.org, v5, 19.9.2016.
- S. Bahrampour, N. Ramakrishnan, L. Schott, M. Shah *Comparative Study of Deep Learning Software Frameworks*, Mar. 2016.
- J. Holt, Thomas Baker, *Back Propagation Simulations using Limited Precision Calculations*, IJCNN-91 on Neural Networks, 1991.
- TensorFlow with Python:
  https://www.tensorflow.org, 3.1.17.
  TensorFlow with R:
  https://github.com/rstudio/tensorflow, 3.1.17.
  Pro and Cons of Frameworks:
  https://deeplearning4j.org/compare-dl4j-torch7-pylearn, 7.7.17.
  Keras:
  https://github.com/fchollet/keras/tree/master/examples, 5.1.17

Questions and Discussion

Benchmarking Results as of Sept. 2016

Hong Kong Baptist University

Network	Software	CPU Threads			CPU Server Threads		
		1	2	8	1	8	16
FCN-5	CNTK	2.351	.962	.810	.828	.547	.530
FCN-5	TF	7.206	2.626	1.934	2.804	1.574	.857
FCN-5	Torch	1.227	.661	-	.536	.440	.425
FCN-8	CNTK	2.641	1.393	.919	.885	.633	.580
FCN-8	TF	7.167	2.630	1.955	2.896	1.577	.892
FCN-8	Torch	.1317	.448	.881	.560	.475	.444
AlexNet	CNTK	6.541	2.140	2.063	2.319	1.684	1.223
AlexNet	TF	3.935	1.694	1.453	1.865	1.067	.666
AlexNet	Torch	4.621	1.400	3.034	1.312	1.114	1.122
ResNet-50	CNTK	-	-	-	-	-	-
ResNet-50	TF	26.707	10.093	8.187	9.989	6.048	3.773
ResNet-50	Torch	12.101	-	-	5.145	4.043	3.770
LSTM-32	CNTK	4.393	1.220	1.369	1.331	.964	.773
LSTM-32	TF	9.306	2.021	1.723	2.168	1.229	.770
LSTM-32	Torch	4.872	2.366	3.645	2.067	1.706	1.763
LSTM-64	CNTK	8.218	2.483	2.762	2.662	1.949	1.527
LSTM-64	TF	11.699	3.516	3.477	4.402	2.525	1.590
LSTM-64	Torch	9.623	4.980	6.976	4.054	3.252	3.358

Benchmarking Results as of Sept. 2016
Hong Kong Baptist University

Network	Software	CPU Threads		GPUs		
		16	32	G.980	G.1080	T.K80
FCN-5	CNTK	.530	.549	.044	.033	.053
FCN-5	TF	.857	.595	.070	.063	.089
FCN-5	Torch	.425	.892	.044	.046	.055
FCN-8	CNTK	.580	.653	.049	.037	.059
FCN-8	TF	.892	.620	.071	.063	.107
FCN-8	Torch	.444	.976	.047	.048	.057
AlexNet	CNTK	1.223	1.292	.054	.040	.091
AlexNet	TF	.666	.914	.058	-	.086
AlexNet	Torch	1.122	1.229	.038	.033	.081
ResNet-50	CNTK	-	-	.245	.207	.475
ResNet-50	TF	3.773	4.060	.346	-	.486
ResNet-50	Torch	3.770	4.428	.215	.188	.435
LSTM-32	CNTK	.773	.897	.088	.062	.133
LSTM-32	TF	.770	.706	.087	.070	.123
LSTM-32	Torch	1.763	2.901	.135	.098	.205
LSTM-64	CNTK	1.527	1.798	.171	.122	.249
LSTM-64	TF	1.590	1.469	.178	.144	.232
LSTM-64	Torch	3.358	5.815	.269	.194	.407

How to set it up

1. Install Python 2.7 or 3.3+
2. Install Cuda Toolkit 8.0
3. Register and install cuDNN 5.1
4. Install cuDNN dependencies
5. Install TensorFlow (various methods)
6. Install TensorFlow-GPU (various methods)

Keras
Example MNIST in Keras

```
batch_size = 128; nb_classes = 10; nb_epoch = 12;

img_rows, img_cols = 28, 28 # input image dimensions
nb_filters = 32             # number of convolutional filters
pool_size = (2, 2)          # size of pooling area for max pooling
kernel_size = (3, 3)        # convolution kernel size

# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

if K.image_dim_ordering() == 'th':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows,
        img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols
        )
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols
        , 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols,
        1)
    input_shape = (img_rows, img_cols, 1)
```

Keras
Example MNIST in Keras

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

Keras

Example MNIST in Keras

```
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size
    [1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size
    [1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Keras

Example MNIST in Keras

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=
    nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test␣score:', score[0])
print('Test␣accuracy:', score[1])
```