DEEP LEARNING WITH PYTHON

PROF. DR. – ING. MORRIS RIEDEL, UNIVERSITY OF ICELAND / JUELICH SUPERCOMPUTING CENTRE
HEAD OF CROSS-SECTIONAL TEAM DEEP LEARNING & RDA CO-CHAIR INTEREST GROUP BIG DATA
18TH APRIL – 8TH JLESC WORKSHOP, BARCELONA, SPAIN
JUELICH SUPERCOMPUTING CENTRE & DEEP LEARNING

Convergence: Simulation and Data Labs (SDL)

Research Groups

Communities

Cross-Sectional Teams

Domain-specific SDLs

Simulation Labs

Data Life Cycle Labs

Exascale co-Design

Facilities

Research Group High Productivity Data Processing

Cross-Sectional Team Deep Learning

Increasing number of Deep Learning Applications in HPC Computing Time Grants

HPC Systems JURECA & JUQUEEN

Modular Supercomputer JUWELS

DEEP-EST EU PROJECT

19th April 2018
DEEP LEARNING 101
Short Overview & Role of Cross-Sectional Team Deep Learning at Juelich Supercomputing Centre (JSC)

- Innovative & disruptive approach

- Provide deep learning tools that work with JSC HPC machines (e.g. Python/Keras/Tensorflow)
- Advance deep learning applications and research on HPC prototypes (e.g. DEEP-EST)
- Engage with industry (industrial relations team) & support SMEs (e.g. Soccerwatch)
- Offer tutorials & application enabling support for commercial & scientific users (e.g. YouTube)
PYTHON

SIMPLE AND FLEXIBLE PROGRAMMING LANGUAGE

- **Selected Benefits**
  - Work with many students reveal: quicky & easy to learn
  - Is an interpreted powerful programming language
  - Has Efficient high-level data structures
  - Provides a simple but effective approach to object-oriented programming
  - Great libraries & community support (e.g. numpy)

- Python is an ideal language for fast scripting and rapid application development that in turn makes it interesting for the machine learning modeling process
- The machine learning modeling process in general and the deep learning modeling process in particular requires iterative and highly flexible approaches
- E.g. network topology prototyping, hyper-parameter tuning, etc.


(this talk is not about the book)
DEEP LEARNING
Programming with TensorFlow

- Tensorflow is an open source library for deep learning models using a flow graph approach
- Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)
- The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPU versions)
- Tensorflow work with the high-level deep learning tool Keras in order to create models fast

(Lessons learned: Installing Tensorflow with right versions of CUDA, Python, and other dependencies on a specific system is NOT a trivial task)


**DEEP LEARNING**

Programming with TensorFlow – What is a Tensor?

- A Tensor is nothing else than a multi-dimensional array often used in scientific & engineering environments
- Tensors are best understood when comparing it with vectors or matrices and their dimensions
- Those tensors ‘flow‘ through the deep learning network during the optimization / learning & inference process

---


---

[7] Big Data Tips, What is a Tensor?
Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano.

The key idea behind the Keras tool is to enable faster experimentation with deep networks. Created deep learning models run seamlessly on CPU and GPU via low-level frameworks.

```
keras.layers.Dense(units,
                                 activation=None,
                                 use_bias=True,
                                 kernel_initializer='glorot_uniform',
                                 bias_initializer='zeros',
                                 kernel_regularizer=None,
                                 bias_regularizer=None,
                                 activity_regularizer=None,
                                 kernel_constraint=None,
                                 bias_constraint=None)
```

```
keras.optimizers.SGD(lr=0.01,
                                momentum=0.0,
                                decay=0.0,
                                nesterov=False)
```

Tool Keras supports inherently the creation of artificial neural networks using Dense layers and optimizers (e.g. SGD).

Includes regularization (e.g. weight decay) or momentum.


Create an Artificial Neural Network (ANN) using Python Scripts

- Feed-forward neural network: nodes in one layer are connected only to the nodes in the next layer (i.e. ‘a constraint of network construction’)
- Think each hidden node as a ‘simple perceptron’ that each creates one hyperplane
- Think the output node simply combines the results of all the perceptrons to yield the ‘decision boundary’ above

(28*28 pixels, 60000 training samples ~47MB, 10000 test samples ~7.8 MB)
ANN Parameters & Data Normalization in Python Script

```python
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2

# dataset 28 x 28 pixels = 784 reshaped
(X_train, y_train), (X_test, y_test) = mnist.load_data()
RESHAPED = 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# data output
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

- **NB_CLASSES**: 10 Class Problem
- **NB_EPOCH**: number of times the model is exposed to the training set – at each iteration the optimizer adjusts the weights so that the objective function is minimized
- **BATCH_SIZE**: number of training instances taken into account before the optimizer performs a weight update
- **OPTIMIZER**: Stochastic Gradient Descent (`SGD`) – only one training sample/iteration

- Data load shuffled between training and testing set
- Data preparation, e.g. `X_train` is 60000 samples / rows of 28 x 28 pixel values that are reshaped in 60000 x 784 including type specification (i.e. `float32`)
- Data normalization: divide by 255 – the max intensity value to obtain values in range `[0,1]`
DEEP LEARNING – PYTHON SCRIPT PART 2
Dynamic & Flexible Modeling of the ANN in Python Script

- The Sequential() Keras model is a linear pipeline (aka ‘a stack’) of various neural network layers including Activation functions of different types (e.g. softmax).

- Dense() represents a fully connected layer used in ANNs that means that each neuron in a layer is connected to all neurons located in the previous layer.

- The non-linear Activation function ‘softmax’ represents a generalization of the sigmoid function – it squashes an n-dimensional vector of arbitrary real values into a n-dimensionsal vector of real values in the range of 0 and 1 – here it aggregates 10 answers provided by the Dense layer with 10 neurons.

```python
# convert vectors to binary matrices of classes
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# Simple ANN model
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()

# Compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# Fit the model
history = model.fit(X_train, Y_train, batch_size=BATCHSIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split=VALIDATION_SPLIT)

# Evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])

L_i = -\sum_j t_{i,j} \log(p_{i,j})
```

- Loss function is a multiclass logarithmic loss: target is $t_{i,j}$ and prediction is $p_{i,j}$.

19th April 2018
DEEP LEARNING – RUNNING THE PYTHON SCRIPT

Using a Job Script for High Performance Computing (HPC) Machine with the Python Script & ANN Model

#!/bin/bash
#PBS -l nodes=1:ppn=all
#PBS -l walltime=1:0:0
#PBS -N KERAS_MNIST_ANNN_HIDDEN

module load TensorFlow/1.4.0-intel-2017b-Python-3.6.3
module load Keras/2.1.1-intel-2017b-Python-3.6.3

# make sure Keras is using TensorFlow as backend
export KERAS_BACKEND=tensorflow

export WORKDIR=$VSC_SCRATCH/${PBSJOBNAME}-${PBSJOBDIR}
mkdir -p $WORKDIR

cd $WORKDIR

export OMP_NUM_THREADS=1
python $PBS_O_WORKDIR/KERAS_MNIST_ANNN_HIDDEN.py

echo "Working directory was $WORKDIR"

[vsc42544@glicher03 deeplearning]$ more KERAS_MNIST_ANNN_HIDDEN.o1179466
60000 train samples
10000 test samples

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 128)</td>
<td>100480</td>
</tr>
<tr>
<td>activation_1 (Activation)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 128)</td>
<td>16512</td>
</tr>
<tr>
<td>activation_2 (Activation)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 10)</td>
<td>1290</td>
</tr>
<tr>
<td>activation_3 (Activation)</td>
<td>(None, 10)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
Train on 48000 samples, validate on 12000 samples

[vsc42544@glicher03 deeplearning]$ tail KERAS_MNIST_ANNN_HIDDEN.o1179466

32/10000 [==========]  - ETA: 0s
2272/10000 [=====>]  - ETA: 0s
4544/10000 [============]  - ETA: 0s
6784/10000 [==============]  - ETA: 0s
9088/10000 [================]  - ETA: 0s
10000/10000 [==================]  - 0s 22us/step
Test score: 0.0772481116249
Test accuracy: 0.97783

Working directory was /user/scratch/gent/vsc425/vsc42544/KERAS_MNIST_ANNN_HIDDEN_1179466.master19.goeltt.gent.vsc

A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer.

The non-linear Activation function 'relu' represents a so-called Rectified Linear Unit (ReLU) that only recently became very popular because it generates good experimental results in ANNs and more recent deep learning models – it just returns 0 for negative values and grows linearly for only positive values.

```python
# data output
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert vectors to binary matrices of classes
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# ANN model with hidden layers
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()

# Compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# Fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split=VALIDATION_SPLIT)

# Evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```
DEEP LEARNING – ARCHITECTURES

Each of the Architectures provide Unique Characteristica (e.g. ‘smart layers‘)

- Deep Neural Network (DNN)
  - ‘Shallow ANN‘ approach with many hidden layers between input/output

- Convolutional Neural Network (CNN, sometimes ConvNet)
  - Connectivity pattern between neurons is like animal visual cortex

- Deep Belief Network (DBN)
  - Composed of multiple layers of variables; only connections between layers

- Recurrent Neural Network (RNN)
  - ‘ANN‘ but connections form a directed cycle; state and temporal behaviour

- Deep Learning architectures can be classified into Deep Neural Networks, Convolutional Neural Networks, Deep Belief Networks, and Recurrent Neural Networks all with unique characteristica
- Deep Learning needs ‘big data‘ to work well & for high accuracy – works not well on sparse data
- Classify pixels in a hyperspectral remote sensing image having groundtruth/labels available
- Created CNN architecture for a specific hyperspectral land cover type classification problem
- Used dataset of Indian Pines (compared to other approaches) using all labelled pixels/classes
- Performed no manual feature engineering to obtain good results (aka accuracy)
DEEP LEARNING – EXPERIMENTING WITH TOPOLOGIES

Programming with Python & TensorFlow & Keras – Supervised Classification Example – Network Topology

# standard model that achieved 85% accuracy on test in best of 5
# The pool_size is adapted to 220 channels
# The first two dimensions of kernel_size are adapted to 9x9 window tensors

def build_model_standard(input_shape, activation, num_classes):
    model = Sequential()
    model.add(Conv3D(48, kernel_size=(3, 3, 5), activation=activation, input_shape=input_shape))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format='None'))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format='None'))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(MaxPooling3D(pool_size=(1, 1, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation=activation))
    model.add(Dense(128, activation=activation))
    model.add(Dense(num_classes, activation='softmax'))
    return model


Using Python with TensorFlow & Keras easily enables changes in hyper-parameter tuning

Various runs on different topologies add up to computational demand of GPUs

Need for HPC machines with good GPUs and good deep learning software stacks required

Blue: correctly classified  
Red: incorrectly classified

<table>
<thead>
<tr>
<th>Feature</th>
<th>Representation / Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv. Layer Filters</td>
<td>48, 32, 32</td>
</tr>
<tr>
<td>Conv. Layer Filter size</td>
<td>(3, 3, 5), (3, 3, 5), (3, 3, 5)</td>
</tr>
<tr>
<td>Dense Layer Neurons</td>
<td>128, 128</td>
</tr>
<tr>
<td>Optimizer</td>
<td>SGD</td>
</tr>
<tr>
<td>Loss Function</td>
<td>mean squared error</td>
</tr>
<tr>
<td>Activation Functions</td>
<td>ReLU</td>
</tr>
<tr>
<td>Training Epochs</td>
<td>600</td>
</tr>
<tr>
<td>Batch Size</td>
<td>50</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>1</td>
</tr>
<tr>
<td>Learning Rate Decay</td>
<td>$5 \times 10^{-6}$</td>
</tr>
</tbody>
</table>


COMPARE TRADITIONAL MACHINE LEARNING

Supervised Classification Example – Remote Sensing Dataset & Results

- Traditional Methods
  - Support Vector Machine (SVM)
  - 52 classes of different land cover, 6 discarded, mixed pixels, rare groundtruth
  - Substantial manual feature engineering, e.g. Self Dual Attribute Profile (SDAP)
  - 10-fold cross-validation
  - Achieved 77.02% accuracy


COMPARE TRADITIONAL MACHINE LEARNING

Supervised Classification Example – Speed-up Cross-Validation & MPI C Code

- Traditional Methods
  - Support Vector Machine (SVM)
  - piSVM C code that can be improved (taken from ML experts – not parallel experts, tuned @ JSC)
  - Message Passing Interface (MPI)

- Feature Engineering
  - Working also on parallel feature engineering using tree-based approach (MPI/OpenMP C)

---

First Result: best parameter set from 14.41 min to 1.02 min
Second Result: all parameter sets from ~9 hours to ~35 min

---

[16] piSVM Website, 2011/2014 code
SELECTED COMPARISONS

Supervised Classification Example – Remote Sensing Dataset & Results

- **Traditional Methods**
  - C MPI-based Support Vector Machine (SVM)
  - Substantial manual feature engineering
  - 10-fold cross-validation for model selection
  - Achieved **77.02% accuracy**

- **Convolutional Neural Networks (CNNs)**
  - Python/TensorFlow/Keras
  - Automated feature learning
  - Achieved **84.40% accuracy on all 58 classes**

**SVM + Feature Engineering (~3 years) vs. CNN architecture setup (~1 month)**

DEEP SERIES OF PROJECTS
EU Projects Driven by Co-Design of HPC Applications

- 3 EU Exascale projects
  - DEEP
  - DEEP-ER
  - DEEP-EST

- 27 partners
  Coordinated by JSC

- EU-funding: 30 M€
  JSC-part > 5,3 M€

- Nov 2011 – Jun 2020


Juelich Supercomputing Centre implements the DEEP projects designs in its production infrastructure
JSC – MODULAR SUPERCOMPUTING ARCHITECTURE

Roadmap

- **Deep learning**
- **ML Training**
- **ML Testing, Inference**
- **Deep learning**
- **Data Models**
- **‘Big data‘ / parallel I/O**

- **Innovative Ideas, e.g. trained models in memory**
- **Data storage module for geospatial datasets?**
- **Innovative Ideas, e.g. use of deep learning optimized chip designs**
IBM Power 6
JUMP, 9 TFlop/s

JUROPA
200 TFlop/s
HPC-FF
100 TFlop/s

IBM Power 4+
JUMP (2004), 9 TFlop/s

IBM Blue Gene/L
JUBL, 45 TFlop/s

IBM Blue Gene/P
JUGENE, 1 PFlop/s

IBM Blue Gene/Q
JUQUEEN (2012)
5.9 PFlop/s

JUWELS Scalable Module (2019/20)
50+ PFlop/s

JUWELS Cluster Module (2018)
12 PFlop/s

JURECA Cluster
(2015) 2.2 PFlop/s

JURECA Booster
(2017) 5 PFlop/s

IBM Power 4+
JUMP (2004), 9 TFlop/s

IBM Blue Gene/L
JUBL, 45 TFlop/s

IBM Blue Gene/P
JUGENE, 1 PFlop/s

IBM Blue Gene/Q
JUQUEEN (2012)
5.9 PFlop/s

JUWELS Scalable Module (2019/20)
50+ PFlop/s
TRADITIONAL MACHINE LEARNING
Supervised Classification – Modular Supercomputing Architecture

- (1) The training dataset and testing dataset of the remote sensing application is used many times in the process and make sense to put into the DEEP-EST Network Attached Memory (NAM) module.

- (2) Training with piSVM in order to generate a model requires powerful CPUs with good interconnection for the inherent optimization process and thus can take advantage of the DEEP-EST CLUSTER module (use of training dataset, requires piSVM parameters for kernel and cost).

- (3) Instead of dropping the trained SVM model (i.e. file with support vectors) to disk it makes sense to put this model into the DEEP-EST NAM module.

- (4) Testing with piSVM in order to evaluate the model accuracy requires not powerful CPUs and not a good interconnection but scales perfectly (i.e. nicely parallel) and thus can take advantage of the BOOSTER module (use of testing dataset & model file residing in NAM), prediction & inference using models is largely usable on the BOOSTER module too.

- (5) If accuracy too low back to (2) to change parameters.

DEEP LEARNING (WORK IN PROGRESS)

Supervised Classification – CNN Design & Setup

- (1) The training dataset and testing dataset of the remote sensing application is used many times in the process and make sense to put into the DEEP-EST Network Attached Memory (NAM) module.

- (2) Training with CNNs in Tensorflow works best for many-core CPUs for the inherent optimization process based on Stochastic Gradient Descent (SGD) MPI collective available in the DEEP-EST Global Collective Environment (GCE) and thus can take advantage of the DEEP-EST BOOSTER module (use of training dataset, requires CNN architectural design parameters).

- (3) Trained models of selected architectural CNN setups need to be compared and thus can be put in the DEEP-EST NAM module.

- (4) Testing with Tensorflow in order to evaluate the model accuracy works also quite well for many-core architectures and scales perfectly (i.e. nicely parallel) and thus can take advantage of the BOOSTER module (use of testing dataset & CNN models residing in NAM).

- (5) If accuracy too low back to (2) to change parameters.

- (Upcoming: potentially exploring the use of Intel Nervana Chips & Neon with Tensorflow)
DEEP LEARNING

Supervised Classification – Transfer Learning

- (1) Studies have shown that Transfer Learning works well especially for remote sensing data without groundtruth or labelled data (i.e. unsupervised) and pre-trained networks trained on general images like ImageNet (e.g. like Overfeat) are available and are put into the DEEP-EST NAM module to be re-used for unsupervised deep learning CNN training.

- (2) Based on pre-trained features another CNN architectural setup is trained with the real remote sensing data whereby the DEEP-EST DATA ANALYTICS module is an interesting approach since the FPGA might be used to compute the transformation from pre-trained features as suitable inputs to the real training process of the CNN based on remote sensing data.

- (3) Trained models of selected architectural CNN setups that have been used with pre-trained features need to be compared and thus can be put in the DEEP-EST NAM module.

- (4) Testing with Tensorflow in order to evaluate the model accuracy works also quite well for many-core architectures and scales perfectly (i.e. nicely parallel) and thus can take advantage of the BOOSTER module (use of testing dataset & CNN models residing in NAM).

- (5) Testing results are written back to the DEEP-EST NAM per CNN architectural design, the FPGA in the NAM can compute the best obtained accuracy for all the different setups.

- (6) If accuracy is too low consider to move back to step (1) to change the pre-trained network or step (2) to create a better CNN architectural setup.
OTHER MODELS – LSTM FOR TIME SERIES ANALYSIS

Python with TensorFlow/Keras support easy development of LSTM network

- Long Short Term Memory (LSTM) networks are a special kind of Recurrent Neural Networks (RNNs)
- LSTMs learn long-term dependencies in data by remembering information for long periods of time
- The LSTM chain structure consists of four neural network layers interacting in a specific way

[1] M. Riedel, Invited
YouTube Tutorial on Deep Learning, Ghent University

(x uses sigmoid \( \sigma \))

(each line carries an entire vector)
OTHER MODELS – LSTM EXAMPLE

Python with TensorFlow/Keras & LSTM Models

- Prototyping sequence networks with LSTM models is easy using Python with Tensorflow and Keras library
- LSTM models work quite well to predict power but needs to be trained and tuned for different power stations
- Observing that some peaks can not be ‘learned’

```python
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# design network
model = Sequential()
model.add(LSTM(1, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1, activation='tanh'))
model.compile(loss='mean_squared_error', optimizer='adam')

# fit network
history = model.fit(
    train_X, train_y,
    epochs=args.epochs,
    batch_size=args.batch_size,
    validation_data=(test_X, test_y),
    verbose=2,
    shuffle=True
)
```
DEEP LEARNING & GPU PARALLELIZATION

Simple Image Benchmark on JURECA JSC HPC System (75 x 2 NVIDIA Tesla K80/node – dual GPU design)

Setup
- TensorFlow 1.4
- Python 2.7
- CUDA 8
- cuDNN 6
- Horovod 0.11.2
- MVAPICH-2.2-GDR
- ‘Simple’ 1.2 million images with 224 x 224 pixels

Open source tool Horovod enables distributed deep learning using Python with TensorFlow (and Keras)
- Machine & Deep Learning: speed-up is just secondary goal after primary goal accuracy
- Speed-up & parallelization nice to have for faster hyperparameter tuning, model creation, and inference
- Third goal is avoiding much feature engineering through ‘feature learning’ of deep learning networks

![Speedup Graph](image)

<table>
<thead>
<tr>
<th>#GPUs</th>
<th>Images/s</th>
<th>Speedup</th>
<th>Performance per GPU [images/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55</td>
<td>1.0</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>178</td>
<td>3.2</td>
<td>44.5</td>
</tr>
<tr>
<td>8</td>
<td>357</td>
<td>6.5</td>
<td>44.63</td>
</tr>
<tr>
<td>16</td>
<td>689</td>
<td>12.5</td>
<td>43.06</td>
</tr>
<tr>
<td>32</td>
<td>1230</td>
<td>22.4</td>
<td>38.44</td>
</tr>
<tr>
<td>64</td>
<td>2276</td>
<td>41.4</td>
<td>35.56</td>
</tr>
<tr>
<td>128</td>
<td>5562</td>
<td>101.1</td>
<td>43.45</td>
</tr>
</tbody>
</table>

[19] JURECA HPC System


(absolute number of images per second and relative speedup normalized to 1 GPU are given)
DEEP LEARNING – OTHER ACTIVITIES @ JSC

Selected Projects related to Deep Learning

- SoccerWatch.TV
  - SME created/joined by a ‘exit-ing’ PHD Student
  - Besides upper leagues: 80k matches/week
  - Recording too expensive (amateurs) with camera man needed
  - Approach: Find X,Y center and zoom on panorama camera using Deep Learning

- SMITH
  - Virtual Patient models with Markov Chains Monte Carlo (MCMC) mapped to Deep Learning networks
SUMMARY

- **Mindset**
  - Think traditional machine learning still relevant for deep learning
  - Using interpreted languages like Python enable easy & flexible parameter-tuning
  - Selected new approaches with specific deep learning per problem (CNN, LSTM, etc.)

- **Skillset**
  - Basic knowledge of machine learning required for deep learning
  - Faster experimentation and use of various topologies and architectures through Python
  - Validation (i.e. model selection) and regularization still valid(!)

- **Toolset**
  - Parallel versions of traditional machine learning methods exist (piSVM, HPDBSCAN)
  - Python with Tensorflow & Keras just one example & good performance (good install!)
  - Explore technology trends, e.g. specific chips for deep learning, NAM, etc.
REFERENCES (1)

- [1] M. Riedel, ‘Deep Learning using a Convolutional Neural Network’, Ghent University, Invited YouTube Tutorial, Online: https://www.youtube.com/watch?v=gOL1_YlosYk&list=PLrmNhuZo9sgZUdaZ-f6OHK2yFW1kTS2qF
- [8] Keras Python Deep Learning Library, Online: https://keras.io/
REFERENCES (2)

- [16] Original (not JSC tuned) piSVM tool, Online: http://pisvm.sourceforge.net/
REFERENCES (3)

- [17] SoccerWatch.TV,
  Online: https://soccerwatch.tv/

- [18] Smart Medical Information Technology for HealthCare (SMITH),
  Online: http://www.smith.care/

- [19] JURECA HPC System @ Juelich Supercomputing Centre (JSC),
  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/Configuration/Configuration_node.html;jsessionid=CF5F5276AE7CFF0B185DCA048A2D0010

  Online: https://arxiv.org/abs/1802.05799
ACKNOWLEDGEMENTS

Previous & current members of the High Productivity Data Processing Research Group

PD Dr. G. Cavallaro
Senior PhD Student A.S. Memon
Senior PhD Student M.S. Memon
PhD Student E. Erlingsson
PhD Student S. Bakarat
MSc Student G.S. Guðmundsson (Landsverkjun)

Dr. M. Goetz (now KIT)
MSc M. Richerzhagen
MSc P. Glock (now INM-1)
DEEP Learning Startup
MSc C. Bodenstein (now Soccerwatch.tv)
THANKS

Talk shortly available under www.morrisriedel.de