CNTK: deep learning framework

Alexey Kamenev

CNTK Overview

- CNTK: Computational Network ToolKit
  - Created by MSR Speech researchers several years ago
- Unified framework for building:
  - Deep Neural Networks (DNNs)
  - Recurrent Neural Networks (RNNs)
  - Long Short Term Memory networks (LSTMs)
  - Convolutional Neural Networks (CNNs)
  - Deep Structured Semantic Models (DSSMs)
  - and few other things…
- All types of deep learning applications: speech, vision and text

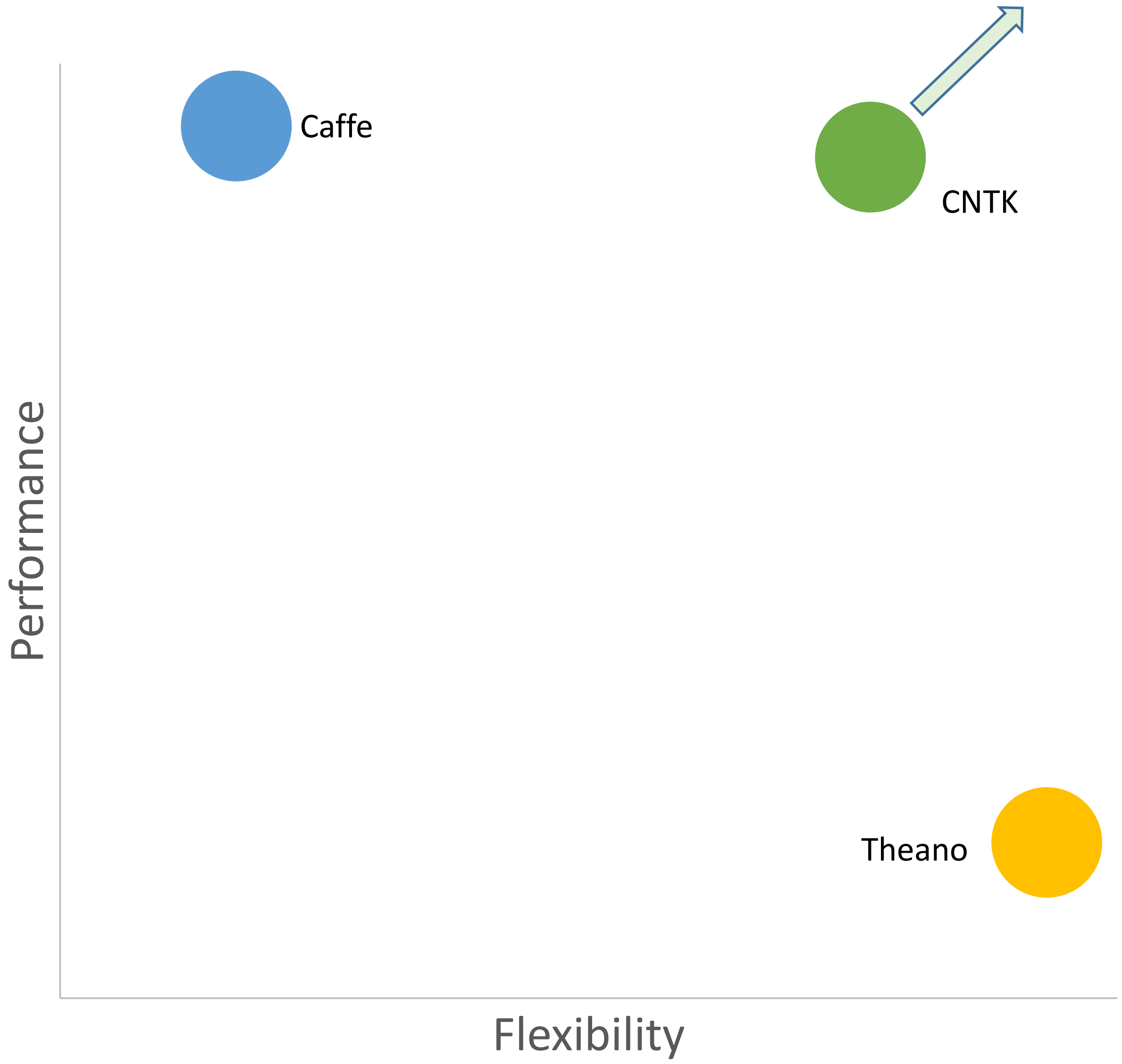
CNTK Overview

- Open source
  - Currently hosted on CodePlex, GitHub migration to be done soon
  - Contributors from Microsoft and external (MIT, Stanford etc)
- Runs on Linux and Windows
  - Project Philly runs 100% on Linux
- Efficient GPU and CPU implementations
- GPU implementation uses highly-optimized libraries from NVIDIA:
  - CUB
  - cuDNN
  - and of course other things like cuBLAS, cuSPARSE, cuRAND etc.

CNTK Overview

- Distributed training
  - Can scale to hundreds of GPUs
  - Supports 1-bit SGD and model averaging
  - ASGD is coming soon
- Supports most popular input data formats
  - Plain text (e.g. UCI datasets)
  - Speech formats (HTK)
  - Images
  - Binary
  - DSSM file format
  - New formats can be added by creating DataReader
- State of the art results on speech and image workloads

CNTK vs …



Configuring CNTK

- Define computation graph
  - Feed forward
  - Recurrent
  - Convolutional nets.
- Define training parameters:
  - Command definitions
  - Learner (SGD) parameters
    - Adaptive learning algorithms supported (AdaGrad, RmsProp, FSAdaGrad)
  - Data reader settings
- Optional: define model transformations

NDL example

Simple, one-hidden layer network (MNIST)

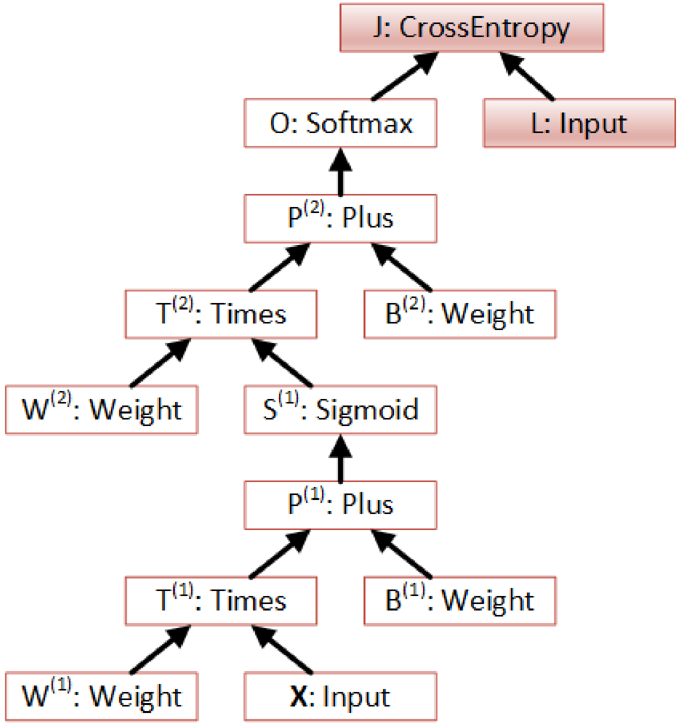
Main file:

```
# input dimension
FeatDim = 784
# label dimension
LabelDim = 10

features = Input(FeatDim, tag = feature)
featScale = Const(0.00390625)
featScaled = Scale(featScale, features)

labels = Input(LabelDim, tag = label)

HiddenDim = 200

h1=DNNLayer(FeatDim, HiddenDim, featScaled, 1)
ol=DNNLastLayer(labelDim, HiddenDim, h1, 1)

CE = CrossEntropyWithSoftmax(labels, ol, tag = Criteria)
Err = ErrorPrediction(labels, ol, tag = Eval)
OutputNodes = ol
```

Macros file:

```
DNNLayer(inDim, outDim, x, parmScale)
{
    W = Parameter(outDim, inDim, init = Uniform,
      initValueScale = parmScale)
    b = Parameter(outDim, init = Uniform,
      initValueScale = parmScale)
    t = Times(W, x)
    z = Plus(t, b)
    y = sigmoid(z)
}

DNNLastLayer(LabelDim, hiddenDim, x, parmScale)
{
    W = Parameter(LabelDim, hiddenDim, init = Uniform,
      initValueScale = parmScale)
    b = Parameter(LabelDim, init = Uniform,
      initValueScale = parmScale)
    t = Times(W, x)
    z = Plus(t, b)
}
```

NDL example (cont.)

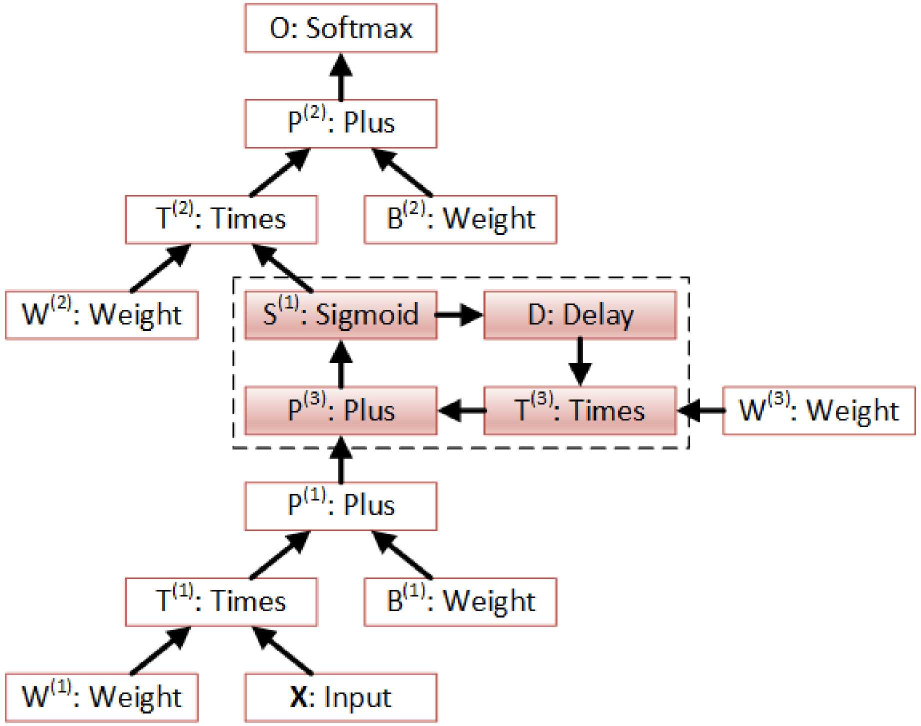


- Data flows through the graph, computations performed in the nodes
- Automatic differentiation is enabled by recursive gradient computation algorithm

NDL example: LSTM

```
LSTMPComponent(inputDim, outputDim, cellDim, inputx)
{
        Wxo = Parameter(cellDim, inputDim, init=uniform, initValueScale=1);
        Wxi = Parameter(cellDim, inputDim, init=uniform, initValueScale=1);
        Wxf = Parameter(cellDim, inputDim, init=uniform, initValueScale=1);
        Wxc = Parameter(cellDim, inputDim, init=uniform, initValueScale=1);
...

        dh = PastValue(outputDim, output, timeStep=1);
        dc = PastValue(cellDim, ct, timeStep=1);

        Wxix = Times(Wxi, Scale(expsWxi, inputx));
        Whidh = Times(Whi, Scale(expsWhi, dh));
        Wcidc = DiagTimes(Wci, Scale(expsWci, dc));

        it = Sigmoid (Plus ( Plus (Plus (Wxix, bi), Whidh), Wcidc));
...
        Wxox  = Times(Wxo, Scale(expsWxo, inputx));
        Whodh = Times(Who, Scale(expsWho, dh));
        Wcoct = DiagTimes(Wco, Scale(expsWco, ct));

        ot = Sigmoid( Plus( Plus( Plus(Wxox, bo), Whodh), Wcoct));

        mt = ElementTimes(ot, Tanh(ct));

        output = Times(Wmr, Scale(expsWmr, mt));
}
```

- LSTM layer of arbitrary complexity can be defined in NDL
- PastValue node is the key
- No need to write from scratch – plenty of examples are available



CNTK in production: Project Philly

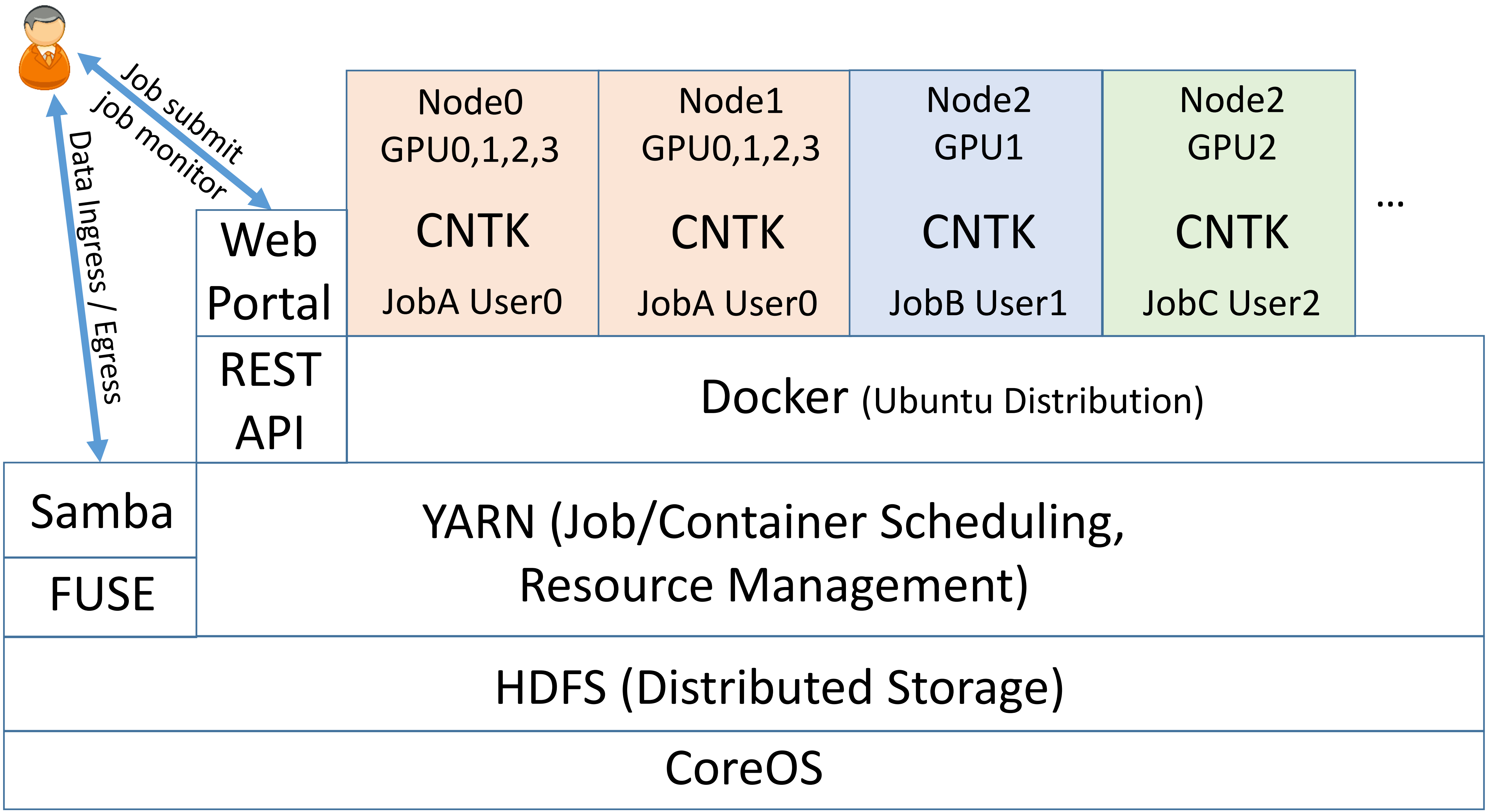
- Turnkey GPU DNN training cluster
- Scalable to hundreds of NVIDIA GPUs
- Rapid, no-hassle, DNN experimentation
- Larger models and training data sets
- Multitenant
- Fault tolerant
- Open source friendly
- 3rd party accessible

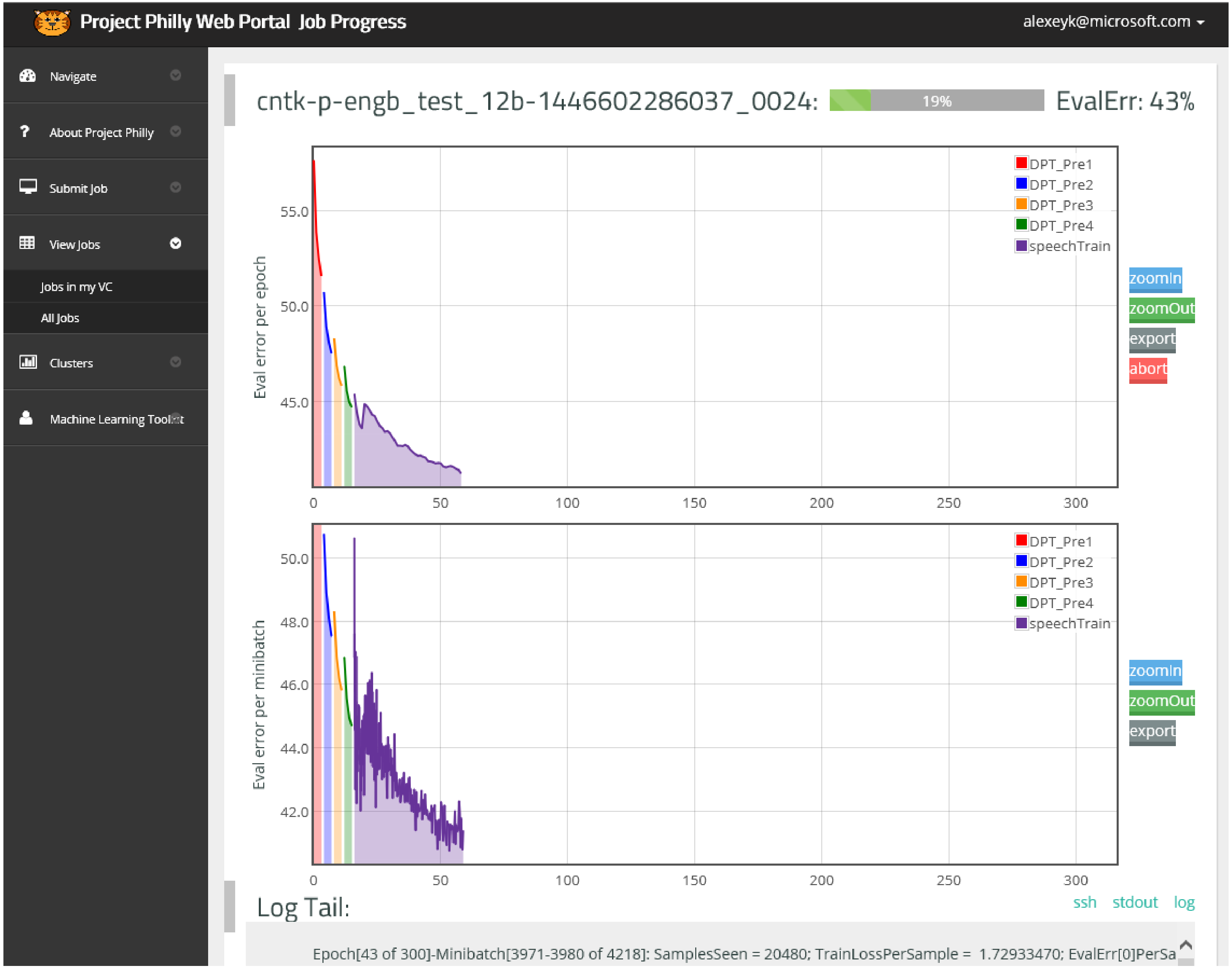
How we use Project Philly

- Massive improvement in training time with greater scale
- Enable bigger data and more complicated algorithms
- Used company-wide for DNN training
  - Available for experimental and production jobs
- Future plan to make it a public service
  - Partnership with Azure

Why Linux and OSS?

- OSS infrastructure components
  - Allows for rapid build out a turnkey, capable system with fault tolerance
  - Industry recognized innovative, state of the art cloud tools
- Components easily compose into solid platform
- GPUDirect RDMA and CUDA aware MPI are currently available on Linux only
  - Key to low latency networking and scale out


Job submit
job monitor
Data Ingress / Egress
Node0
GPU0,1,2,3
CNTK
JobA User0
Node1
GPU0,1,2,3
CNTK
JobA User0
Node2
GPU1
CNTK
JobB User1
Node2
GPU2
CNTK
JobC User2
...
Web
Portal
REST
API
Docker (Ubuntu Distribution)
Samba
FUSE
YARN (Job/Container Scheduling,
Resource Management)
HDFS (Distributed Storage)
CoreOS


Project Philly Web Portal Job Progress
alexeyk@microsoft.com
Navigate
About Project Philly
Submit Job
View Jobs
Jobs in my VC
All Jobs
Clusters
Machine Learning Toolkit
cntk-p-engb_test_12b-1446602286037_0024:
19%
EvalErr: 43%
Eval error per epoch
DPT_Pre1
DPT_Pre2
DPT_Pre3
DPT_Pre4
speechTrain
zoomIn
zoomOut
export
abort
Eval error per minibatch
Log Tail:
ssh
stdout
log
Epoch[43 of 300]-Minibatch[3971-3980 of 4218]: SamplesSeen = 20480; TrainLossPerSample = 1.72933470; EvalErr[0]PerSa

Backup

NDL example: CNN (AlexNet)

Main file:

```
# conv2
kW2 = 5
kH2 = 5
cMap2 = 192
hStride2 = 1
vStride2 = 1
# weight[cMap2, kW2 * kH2 * cMap1]
conv2_act = ConvReLULayer(pool1, cMap2, 1600, kW2, kH2,
  hStride2, vStride2, conv2WScale, conv2BValue)

# pool2
pool2W = 3
pool2H = 3
pool2hStride = 2
pool2vStride = 2
pool2 = MaxPooling(conv2_act, pool2W, pool2H,
  pool2hStride, pool2vStride)
```

Macros file:

```
ConvReLULayer(inp, outMap, inWCount, kW, kH, hStride, vStride,
  wScale, bValue)
{
    convW = Parameter(outMap, inWCount, init = Gaussian,
      initValueScale = wScale)
    conv = Convolution(convW, inp, kW, kH, outMap, hStride,
      vStride, zeroPadding = true)
    convB = Parameter(outMap, 1, init = fixedValue, value = bValue)
    convPlusB = Plus(conv, convB);
    act = RectifiedLinear(convPlusB);
}

DNNReLULayer(inDim, outDim, x, wScale, bValue)
{
    W = Parameter(outDim, inDim, init = Gaussian,
      initValueScale = wScale)
    b = Parameter(outDim, init = fixedValue, value = bValue)
    t = Times(W, x)
    z = Plus(t, b)
    y = RectifiedLinear(z)
}
```

CNTK configuration files

Defines sequence of commands:

```
command=DPT_Pre1:AddLayer2:DPT_Pre2:AddLayer3:DPT_Pre3:AddLayer4:DPT_Pre4:AddLayer5:Train

...

DPT_Pre1=[
    action=train
    modelPath=$ModelDir$\Pre1\cntkSpeech

    NDLNetworkBuilder=[
        networkDescription=$WorkDir$\lib\ndl\dnn_1layer.txt
    ]
]

...
```

CNTK configuration files

And data reader(s):

```
Test=[
    action=test
    modelPath=$ModelDir$/AlexNet.Top5
    # Set minibatch size for testing.
    minibatchSize=128

     NDLNetworkBuilder=[
        networkDescription=$WorkDir$/AlexNet.ndl
    ]

    reader=[
        readerType=ImageReader
        file=$WorkDir$/val_map.txt
        randomize=None
        features=[
            width=224
            height=224
            channels=3
            cropType=Center
            meanFile=$WorkDir$/ImageNet1K_mean.xml
        ]
        labels=[
            labelDim=1000
        ]
    ]
]
```

Next steps: BrainScript

- 3 similar but incompatible languages:
  - NDL
  - Config file
  - MEL
- Combine all of them into a single language:
  - Change syntax as little as possible (backward compatibility)
  - Eliminate inconsistencies
- BrainScript is still in development but already used to build exotic LSTM models.
  - Other examples: defining new adaptive learning algorithm (e.g. Adam or ESGD) completely in BS

“BrainScript??”

- *full name* perfectly expresses our grand *long-term ambition*
- *two-letter acronym* perfectly expresses *today’s state* of the degree that artificial neural networks actually implement brains

☺

© Frank Seide