CNTK

Mithun Prasad, PhD
miprasad@Microsoft.com

The Microsoft Cognition Toolkit (CNTK)

Microsoft's open-source deep-learning toolkit

- Ease of use: what, not how
- Fast
- Flexible

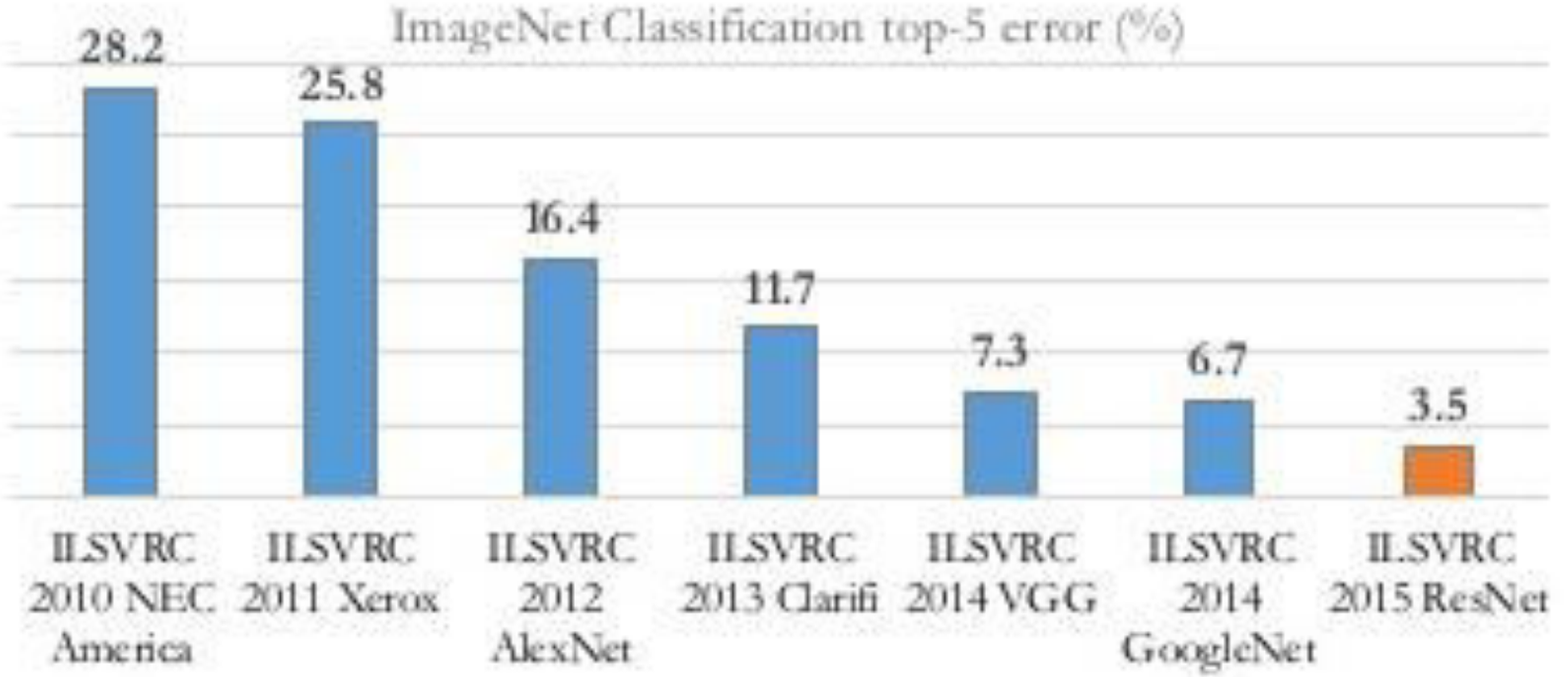
- 1st –class Windows support
- Internal=external version

Microsoft

Deep Learning at Microsoft

- Microsoft Cognitive Services
- Skype Translator
- Cortana
- Bing
- HoloLens
- Microsoft Research

ImageNet: Microsoft 2015 ResNet



Microsoft had all **5 entries** being the 1-st places this year: ImageNet classification, ImageNet localization, ImageNet detection, COCO detection, and COCO segmentation

Microsoft

I. What is CNTK
II. How to use CNTK
III. Deep dive into CNTK technologies
IV. Examples source-code walkthroughs

CNTK “Cognition Toolkit”

- CNTK IS Microsoft’s open-source, cross-platform toolkit for learning and evaluating deep neural networks
- CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications
- CNTK is production-ready: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server

Microsoft

CNTK – Open-Source, Cross-Platform Toolkit

Open-source model inside and outside the company

- Created by Microsoft speech researchers (Dong Yu et al.) in 2012; open-sourced (Codeplex) in early 2015
- On GitHub since Jan 2016 under permissive license
- “working out loud:” virtually all code development is out in the open

Used by Microsoft product groups

- CNTK-trained models power more and more Microsoft products
- Several teams have full-time employees on CNTK that actively contribute

External contributions e.g. from MIT and Stanford

Linux, Windows, docker, cudnn5

- Python and c++ API beta in october; followed by c#/. Net

Microsoft

CNTK – Simple Building Blocks

Example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(w_1 x + b_1)$$
$$h_2 = \sigma(w_2 h_1 + b_2)$$
$$P = softmax\,(w_{out} h_2 + b_{out})$$

$$h1 = Sigmoid\,(w_1 * x + b_1)$$
$$h2 = Sigmoid\,(w_2 * h_1 + b_2)$$
$$p = Softmax\,(Wout * h_2 + bout)$$

With input $x \in R^M$ and label $y \in R^J$

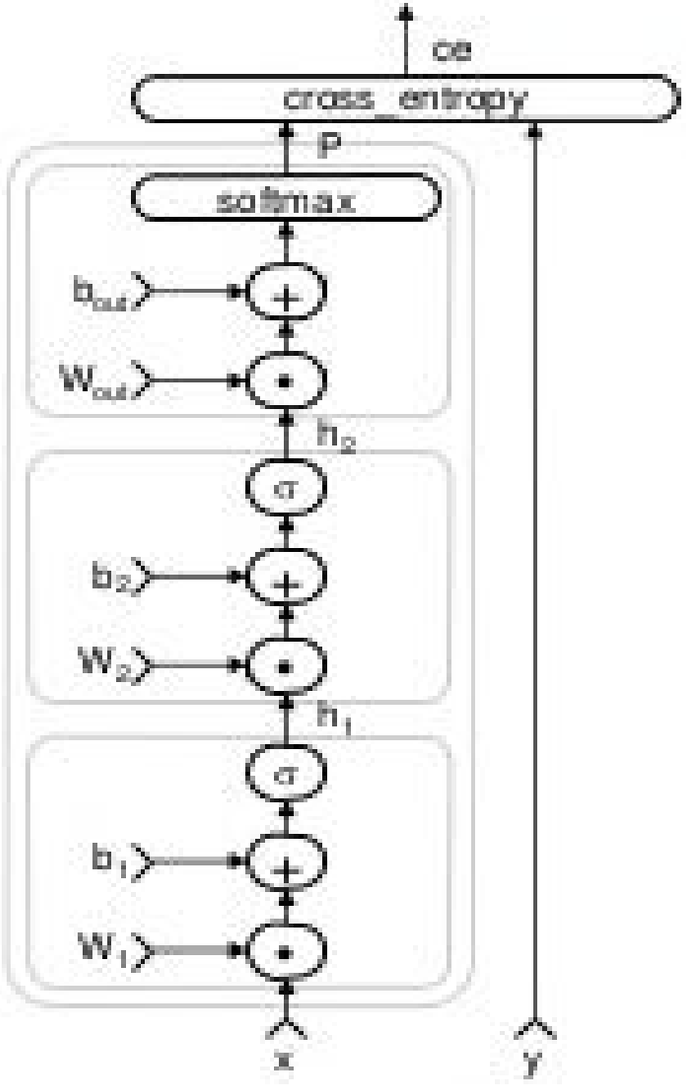
And cross-entropy training criterion

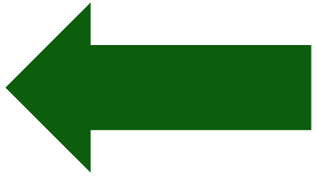
$$ce = y^T \log P$$
$$\sum_{corpus} ce = max$$

$$ce = cross\_entropy(y, p)$$

Microsoft

CNTK – Simple Building Blocks

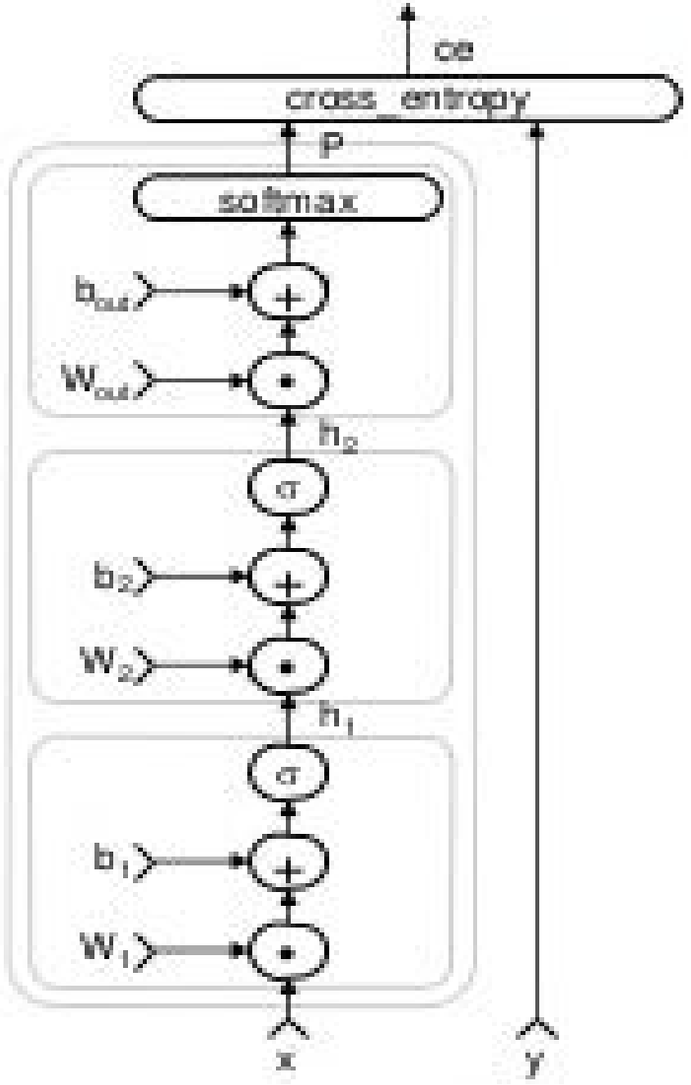


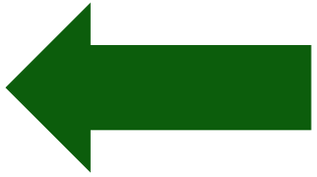


$$h1 = Sigmoid\ (w_1 * x + b_1)$$
$$h2 = Sigmoid\ (w_2 * h_1 + b_2)$$
$$p = Softmax\ (Wout\ *\ h_2 + bout)$$
$$ce = CrossEntropy(y, p)$$

Microsoft

CNTK – Simple Building Blocks





- nodes: functions (primitives)
  - can be composed into reusable composites
- edges: values
- deferred computation → execution
  - optimized execution
  - memory sharing

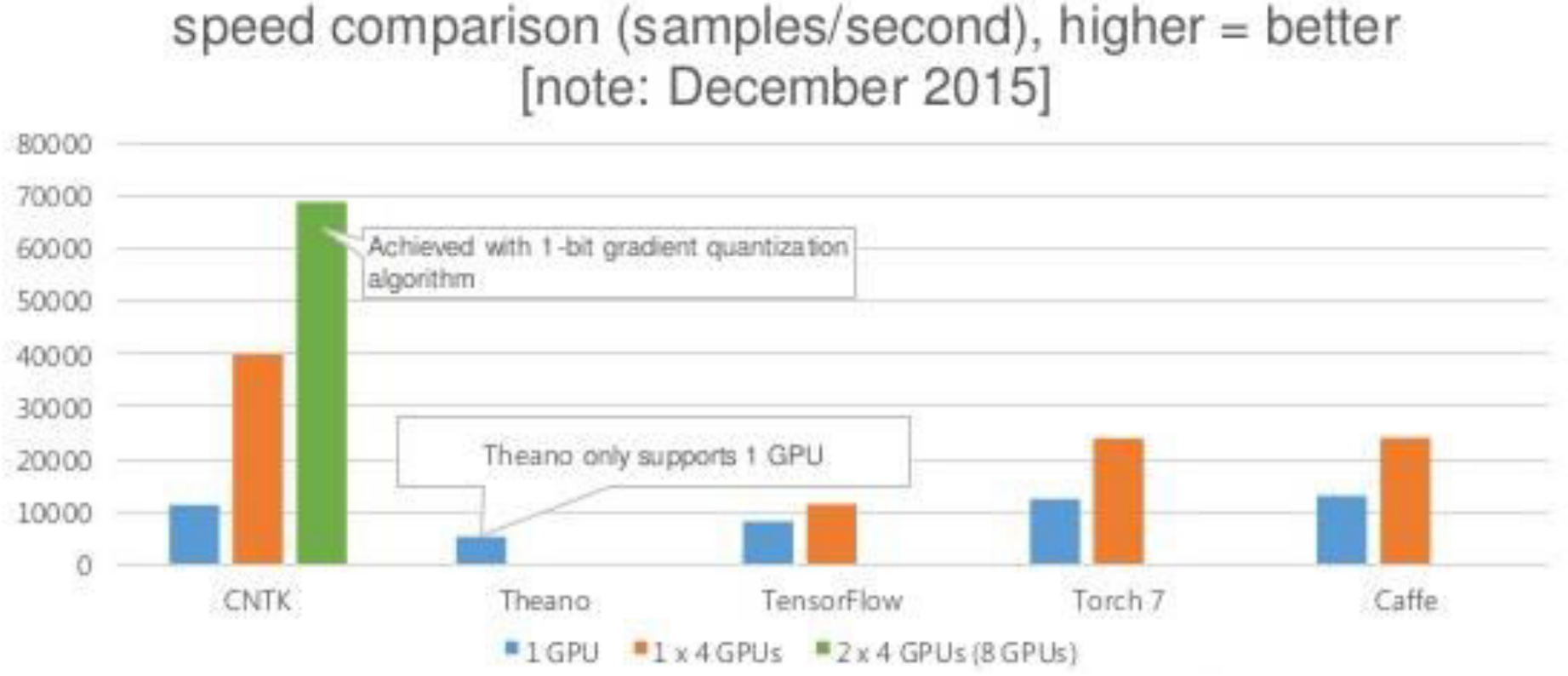
Microsoft

Support for a Range of Networks

- Feed-forward DNN
- RNN, LSTM
- Convolution
- DSSM
- Sequence-to-sequence
- For a range of applications including
  - Speech
  - Vision
  - Text

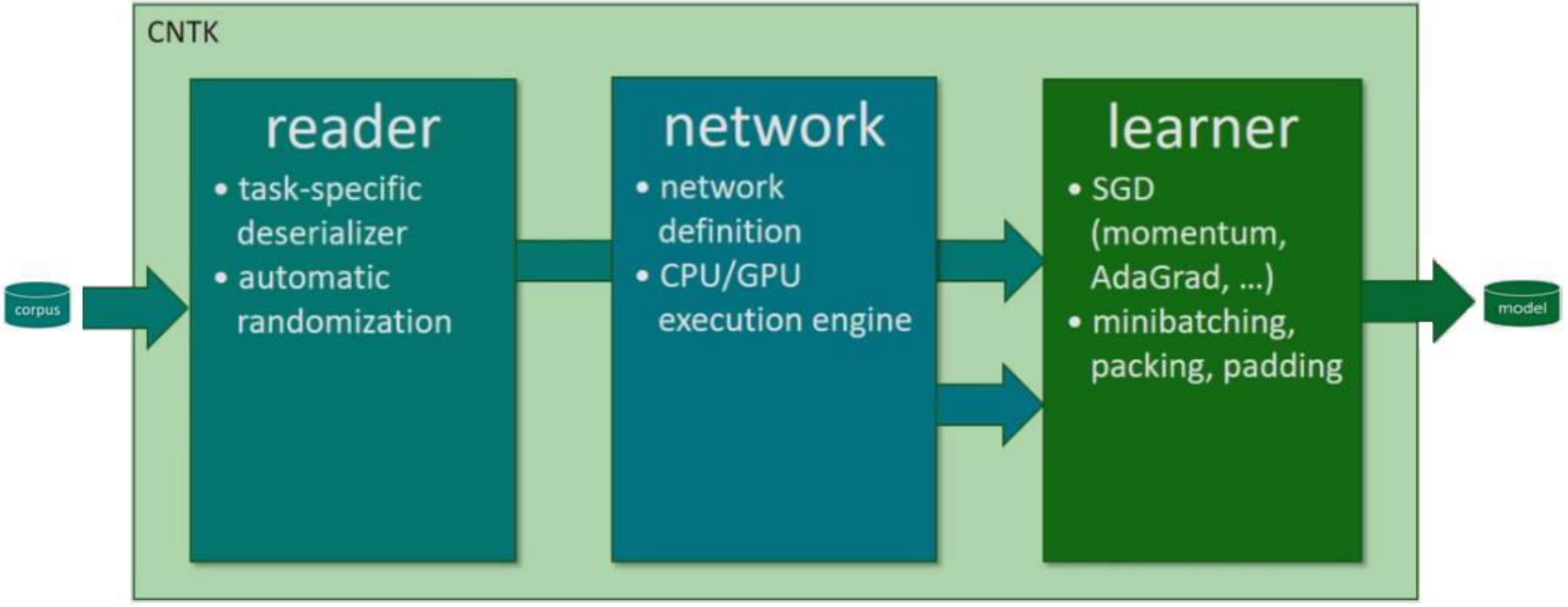
Production-Ready

- State-of-art accuracy on benchmarks and production models
- Multi-GPU/multi-server parallel training on production-size corpora



Microsoft

How to: CNTK Architecture

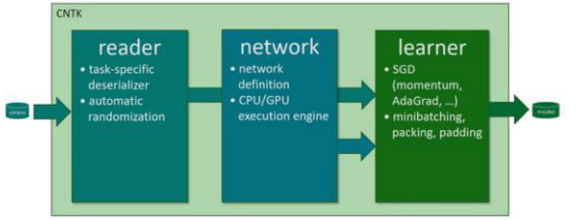


How to: Top-Level Script Outline

```
from cntk import *

# reader
def create_reader(path, is_training):
    …

# network
def create_model_function():
    …
def create_criterion_function(model):
    …

# trainer(and evaluator)
def train(reader, model):
    …
def evaluate(reader, model):
    …

# main function
model=create_model_function()

reader=create_reader(…, is_training=True)
train(reader, model)

reader=create reader(…, is_training=False)
evaluate(reader, model)
```

Microsoft

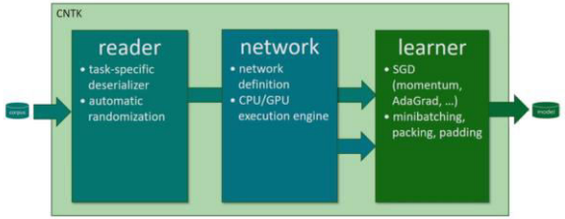
How to: Reader




```
def create_reader (map_file, mean_file, is _training):



    #  deserializer
    return MinibatchSource (ImageDeserializer (map_file,  StreamDefs(
           features = StreamDef (field = “image”, transforms = transforms),
           labels   = StreamDef (field = “label", shape = num_classes)
    )), randomize=is_training, epoch_size = INFINITELY_REPEAT if  is_training else
FULL_DATA_SWEEP)
```

Microsoft

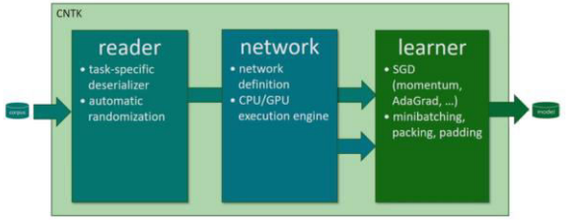
How to: Reader




```
def create_reader(map_file,mean_file,is _training):
    # image preprocessing pipeline
    transforms = [
        ImageDeserializer.crop(crop_type ='Random', ratio=0.8, jitter_type='uniRatio')
        ImageDeserializer.scale(width=image width. height=image_height, channels=num_channels,
                                interpolations='linear'),
        ImageDeserializer.mean(mean_file)
    ]
    # deserialize
    return MinibatchSource(ImageDeserializer.(map_file,StreamDefs(
        features =StreamDef(field='image', transforms = transforms
        labels = StreamDef(field='label', shape=num_classes)
    )), randomize=is_training, epoch_size= INFINITELY_REPEAT if is_trainig else FULL_DATA_SWEEP)
```

Microsoft

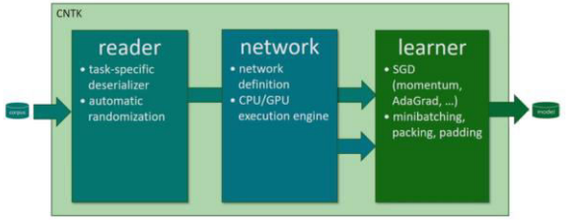
How to: Network




```
M = 40 ; N = 512 ;J = 9000 # feat/hid/out dim
X = Input(M); y = Input(J) # feat/labels
def layer (x, out, in, act): # reusable block
    W = parameter(in,out));b = Parameter(out)
    h = act(x @ w + b)
    return h

h1 = sigmoid (x @ w1  + b1)
h2 = sigmoid (h1 @ w2  + b2)
P = softmax (h2 @ Wout + bout)
ce = cross_entropy(P, y)
```

Microsoft

How to: Network




```
M = 40 ; N = 512 ; J =9000 ; L=2
X = Input (M) ; y = Input (J) # feat/labels
def layer (x, out, in, act): # reusable block
    W = parameter (in, out)) ; b = parameter (out)
    h = act (x @ w + b)
    return h
r = x
for i in range (L):
    r = layer (r, N, M if i = 0 else N, sigmoid)
P = layer (r, J, N, softmax)
ce = cross_entropy (P, y)
```

Microsoft

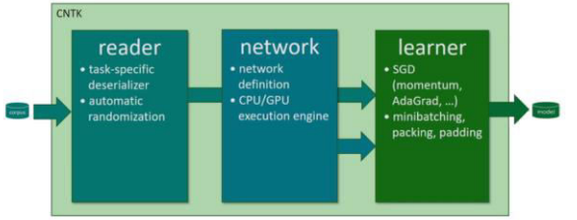
How to: Network (Compact Representation)

```
M = 40 ; N = 512; J = 9000  # feat/hid/out dim
X = Input(M) ; y = Input (J) # feat/labels

from layer import *

h1 = Dense (N, activation=sigmoid) (x)
h1 = Dense (N, activation=sigmoid) (h1)

P = Dense (J, activation = softmax) (h2)
ce = cross_entropy(p, y)
```

How to: Network (Sequential Representation)

```
M = 40 ; N = 512 ; J = 9000      # feat/hid/out dim
X = Input (M)  ;  y = Input (J) # feat/labels

from layers import *
Model = sequential ([
    Dense(N, activation=sigmoid),
    Dense(N, activation=sigmoid),
    Dense(J, activation=softmax)
])
P = model (x)
ce=cross_entropy (P, y)
```

How to: Network



Layers
DenseLayer
ConvolutionLayer
MaxPoolingLayer
AveragePoolingLayer
DropOut
…

https://github.com/Microsoft/CNTK/wiki/Layers-Reference#convolutionallayer

Layers Reference

Philipp Kranen edited this page on Jan 20 · 72 revisions

CNTK predefines a number of common "layers," which makes it very easy to write simple networks that consist of standard layers layered on top of each other. Layers are function objects that can be used like regular BrainScript functions but hold learnable parameters and have an additional pair of `{}` to pass construction parameters or attributes.

For example, this is the network description for a simple 1-hidden layer model using the `DenseLayer{}` layer:

Microsoft

How to: Network

```
# model function: feat -> prediction
model = Sequential ([
    for (range(L), lambda: Dense(N,…
    Dense(J), softmax
])
# criterion function: (feat,label) -> (ce,errs)
y = Placeholder( ) # label argument
ce = cross_entropy_with_softmax (model, y)
errs = classification_error(model, y)
criterion = combine ( [ce, erras])
# trainer
trainer = Trainer(model, criterion, learner)
```

How to: Network

Learning algorithms
AdaGrad
Adam (a low memory variant)
MomentumSGD
Nesterov
RMSProp
SGD