Scalable Deep Learning with Microsoft Cognitive Toolkit (CNTK)

Emad Barsoum, Prin. Software Dev. Engineer
Sayan Pathak, Prin. ML Scientist
Cha Zhang, Prin. Researcher

With 140+ contributors

Microsoft

Deep Learning Explained

Learn an intuitive approach to building the complex models that help machines solve real-world problems with human-like intelligence.

Microsoft

Self-Paced

Enroll Now

☑ I would like to receive email from Microsoft and learn about other offerings related to Deep Learning Explained.

Meet the instructors

Sayan Pathak PhD.
Principal ML Scientist and AI School Instructor, CNTK team
Microsoft

Roland Fernandez
Senior Researcher and AI School Instructor, Deep Learning Technology Center
Microsoft Research AI

Jonathan Sanito
Senior Content Developer
Microsoft


Tutorial Agenda

Agenda

- Introduction (30 min)
  - What is Cognitive toolkit
  - Why use Cognitive Toolkit
- Basic operations (25 min)
  - Data reading and augmentations, Modeling (MLP, CNN, RNN), Training-Test-Eval workflow
- Image application (40 min)
  - ResNet, Inception, ConvNet/Emotion, Faster R-CNN, Segmentation, etc.
- Break (20 min)

Agenda

- Image application (cont. 20 min)
  - Neural style, GAN/Pixel CNN, etc.
- Video application (15 min)
  - Action classification
- Parallel training: (15 min)
  - 1-bit SGD, Block momentum, variable sized mini-batch
- Porting models from Caffe (10 min)
- Reinforcement learning in simulated environment (20 min)
  - Keras flappy bird, RL framework
- Conclusion / Q&A (20 min)


What is Cognitive Toolkit
Why use Cognitive Toolkit

Intro - Microsoft Cognitive Toolkit (CNTK)

- Microsoft's open-source deep-learning toolkit
  - https://github.com/Microsoft/CNTK 

  - Created by Microsoft Speech researchers (Dong Yu et al.) in 2012, "Computational Network Toolkit"
  - Open sourced on CodePlex in Apr 2015
  - On GitHub since Jan 2016 under MIT license, and renamed to "Cognitive Toolkit"
  - Community contributions from MIT, Stanford, Nvidia and many others

Microsoft Cognitive Toolkit

- Runs over 80% Microsoft internal DL workload
- 1st-class on Linux and Windows, docker support
- New in v2.0 GA (Jun 2017):
  - Keras backend support (Beta)
  - Java support, Spark support
  - Model compression (Fast binarized evaluation)

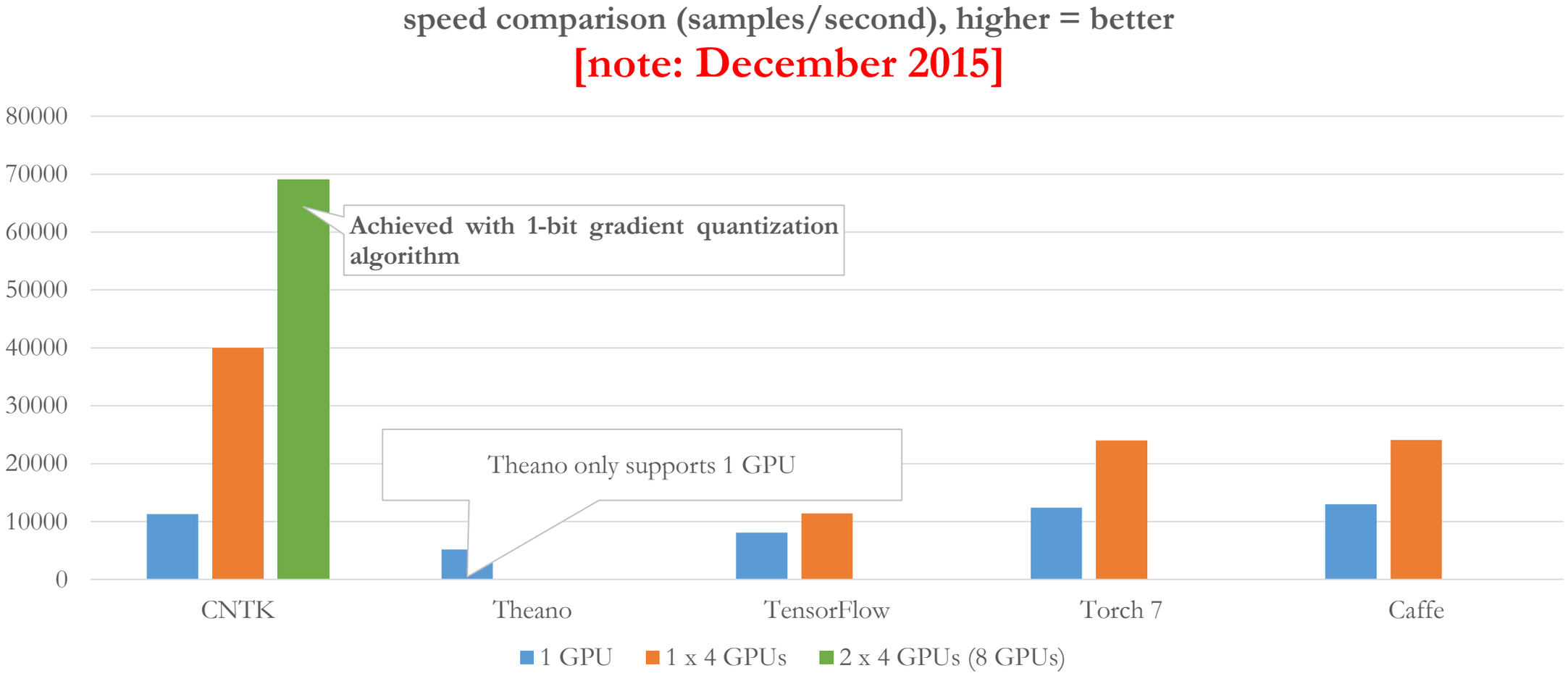
Cognitive Toolkit – The Fastest Toolkit

http://dlbench.comp.hkbu.edu.hk/
Benchmarking by HKBU, Version 8
Single Tesla K80 GPU, CUDA: 8.0 CUDNN: v5.1

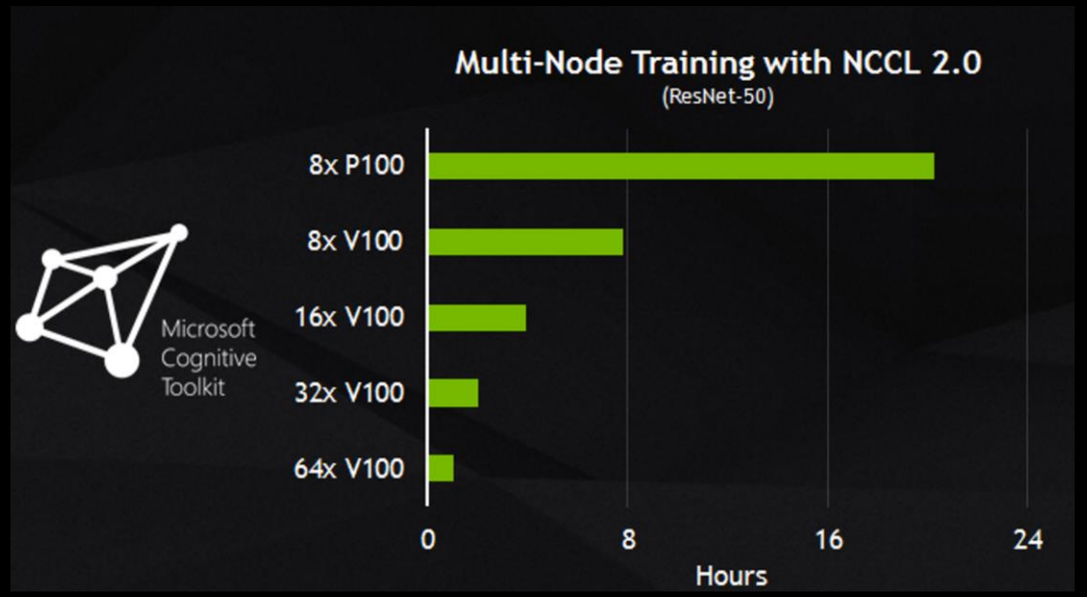
Caffe: 1.0rc5(39f28e4)
CNTK: 2.0 Beta10(1ae666d)
MXNet: 0.93(32dc3a2)
TensorFlow: 1.0(4ac9c09)
Torch: 7(748f5e3)

	Caffe	Cognitive Toolkit	MxNet	TensorFlow	Torch
FCN5 (1024)	55.329ms	**51.038ms**	60.448ms	62.044ms	52.154ms
AlexNet (256)	36.815ms	**27.215ms**	28.994ms	103.960ms	37.462ms
ResNet (32)	143.987ms	**81.470ms**	84.545ms	181.404ms	90.935ms
LSTM (256) (v7 benchmark)	-	**43.581ms** **(44.917ms)**	288.142ms (284.898ms)	- (223.547ms)	1130.606ms (906.958ms)

"CNTK is production-ready: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server."



Superior performance



GTC, May 2017

nvidianews.nvidia.com/news/nvidia-and-microsoft-accelerate-ai-together

NVIDIA

DRIVERS | PRODUCTS | DEEP LEARNING AND AI | COMMUNITIES | SUPPORT | SHOP

NEWSROOM | Multimedia | Executive Bios | Media Contacts | In

News

NVIDIA and Microsoft Accelerate AI Together

Monday, November 14, 2016

GPU-Accelerated Microsoft Cognitive Toolkit Now Available in the Cloud on Microsoft Azure and On-Premises with NVIDIA DGX-1

SC16 -- To help companies join the AI revolution, NVIDIA today announced a collaboration with Microsoft to accelerate AI in the enterprise.

Using the first purpose-built enterprise AI framework optimized to run on NVIDIA® Tesla® GPUs in Microsoft Azure or on-premises, enterprises now have an AI platform that spans from their data center to Microsoft's cloud.

"Every industry has awoken to the potential of AI," said Jen-Hsun Huang, founder and chief executive officer, NVIDIA. "We've worked with Microsoft to create a lightning-fast AI platform that is available from on-premises with our DGX-1™ supercomputer to the Microsoft Azure cloud. With Microsoft's global reach, every company around the world can now tap the power of AI to transform their business."

"We're working hard to empower every organization with AI, so that they can make smarter products and solve some of the world's most pressing problems," said Harry Shum, executive vice president of the Artificial Intelligence and Research Group at Microsoft. "By working closely with NVIDIA and harnessing the power of GPU-accelerated systems, we've made Cognitive Toolkit and Microsoft Azure the fastest, most versatile AI platform. AI is now within reach of any business."

This jointly optimized platform runs the new Microsoft Cognitive Toolkit (formerly CNTK) on NVIDIA GPUs, including the NVIDIA DGX-1™ supercomputer, which uses Pascal™ architecture GPUs with NVLink™ interconnect technology, and on Azure N-Series virtual machines, currently in preview. This combination delivers unprecedented performance and ease of use when using data for deep learning.

As a result, companies can harness AI to make better decisions, offer new products and services faster and provide better customer experiences. This is causing every industry to implement AI. In just two years, the number of companies NVIDIA collaborates with on deep learning has jumped 194x to over 19,000. Industries such as healthcare, life sciences, energy, financial services, automotive and manufacturing are benefiting from deeper insight on extreme amounts of data.

Scalability

Image: Cray

Microsoft, Cray claim deep learning breakthrough on supercomputers

Steve Ranger
ZDNet

A team of researchers from Microsoft, Cray, and the Swiss National Supercomputing Centre (CSCS) have been working on a project to speed up the use of deep learning algorithms on supercomputers.

The team have scaled the Microsoft Cognitive Toolkit -- an open-source suite that trains deep learning algorithms -- to more than 1,000 Nvidia Tesla P100 GPU accelerators on the Swiss centre's Cray XC50 supercomputer, which is nicknamed Piz Daint.

Cognitive Toolkit Other Benefits

- Accuracy
  - Verified training scripts for common networks (AlexNet, ResNet, Inception V3, Faster RCNN, etc.)
- Python and C++ API
  - Mostly implemented in C++ (train and test)
  - Low level + high level Python API
- Extensibility
  - User functions and learners in Python or C++
- Readers
  - Distributed, highly efficient built-in data readers

Cognitive Toolkit Other Benefits

- Keras interoperability
  - Switching your backend to CNTK and your LSTM will be 2-3x faster immediately
- Binary evaluation
  - 10x speed-up in model execution
- Internal == External

What is CNTK?

- CNTK expresses (nearly) **arbitrary neural networks** by composing simple building blocks into complex **computational networks**, supporting relevant network types and applications.

What is CNTK?

Example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \mathrm{softmax}(W_{\mathrm{out}} h_2 + b_{\mathrm{out}})$$

with input $x \in \mathrm{R}^M$

What is CNTK?

Example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \mathrm{softmax}(W_{\mathrm{out}} h_2 + b_{\mathrm{out}})$$

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

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

What is CNTK?

example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \text{softmax}(W_{\text{out}} h_2 + b_{\text{out}})$$

```
h1 = sigmoid (x  @ w1   + b1)
h2 = sigmoid (h1 @ w2   + b2)
P  = softmax (h2 @ wout + bout)
```

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

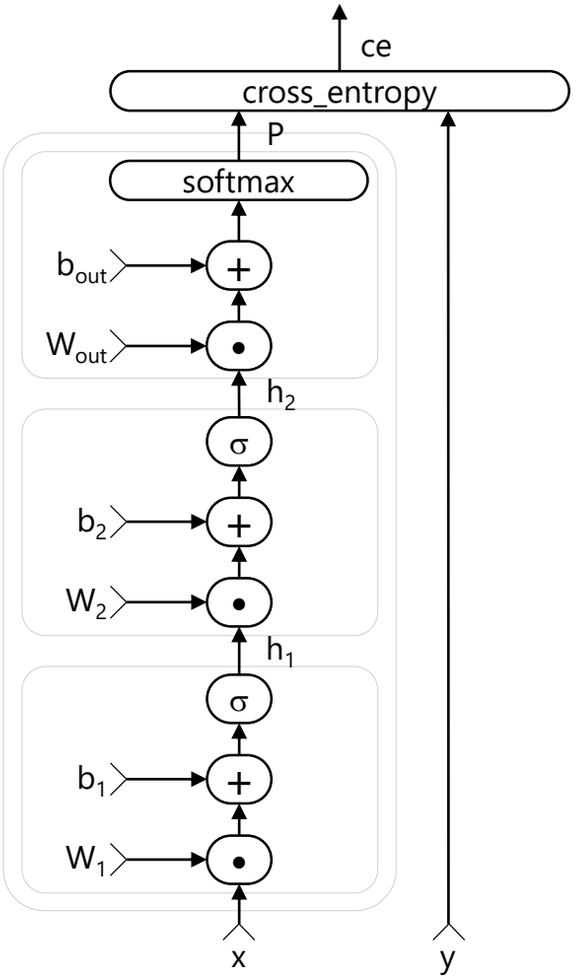
$$ce = y^{\mathrm{T}} \log P$$
$$\sum_{\text{corpus}} ce = \max$$

```
ce = cross_entropy (P, y)
```

What is CNTK?

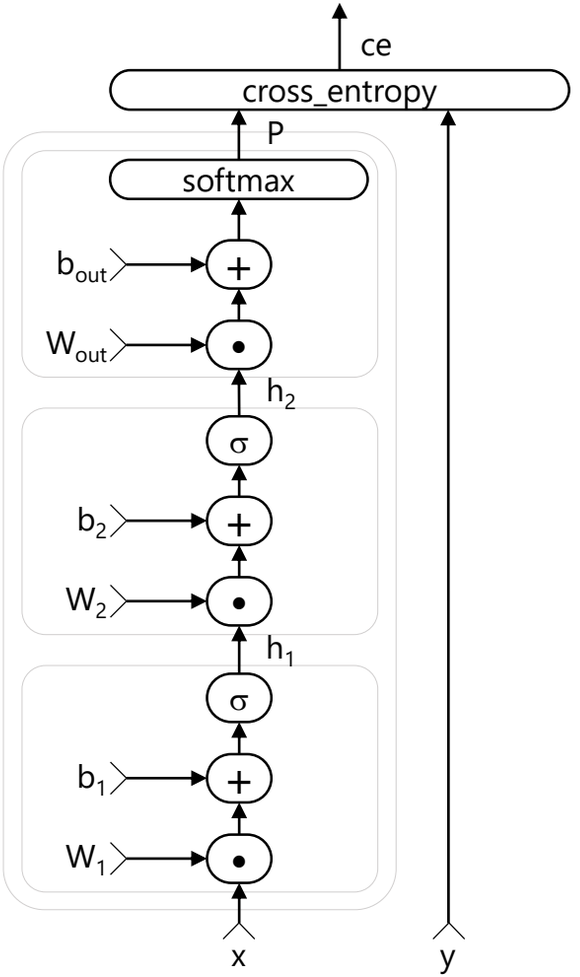
```
h1 = sigmoid (x  @ w1   + b1)
h2 = sigmoid (h1 @ w2   + b2)
P  = softmax (h2 @ wout + bout)
ce = cross_entropy (P, y)
```

What is CNTK?




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

What is CNTK?



- Nodes: functions (primitives)
  - Can be composed into reusable composites
- Edges: values
  - Incl. tensors, sparse
- Automatic differentiation
  - $\partial \mathcal{F} / \partial \text{in} = \partial \mathcal{F} / \partial \text{out} \cdot \partial \text{out} / \partial \text{in}$
- Deferred computation → execution engine
- Editable, clonable

LEGO-like composability allows CNTK to support wide range of networks & applications

CNTK Unique Features

- Symbolic loops over sequences with dynamic scheduling
- Turn graph into parallel program through minibatching
- Unique parallel training algorithms (1-bit SGD, Block Momentum)

Symbolic Loops over Sequential Data

Extend our example to a recurrent network (RNN)

$$
\begin{aligned}
h_1 &= \sigma(\mathbf{W}_1 \, x + b_1) \\
h_2 &= \sigma(\mathbf{W}_2 \, h_1 + b_2) \\
P &= \mathrm{softmax}(\mathbf{W}_{\mathrm{out}} \, h_2 + b_{\mathrm{out}}) \\
ce &= L^{\mathrm{T}} \log P \\
\sum_{\mathrm{corpus}} ce &= \max
\end{aligned}
$$

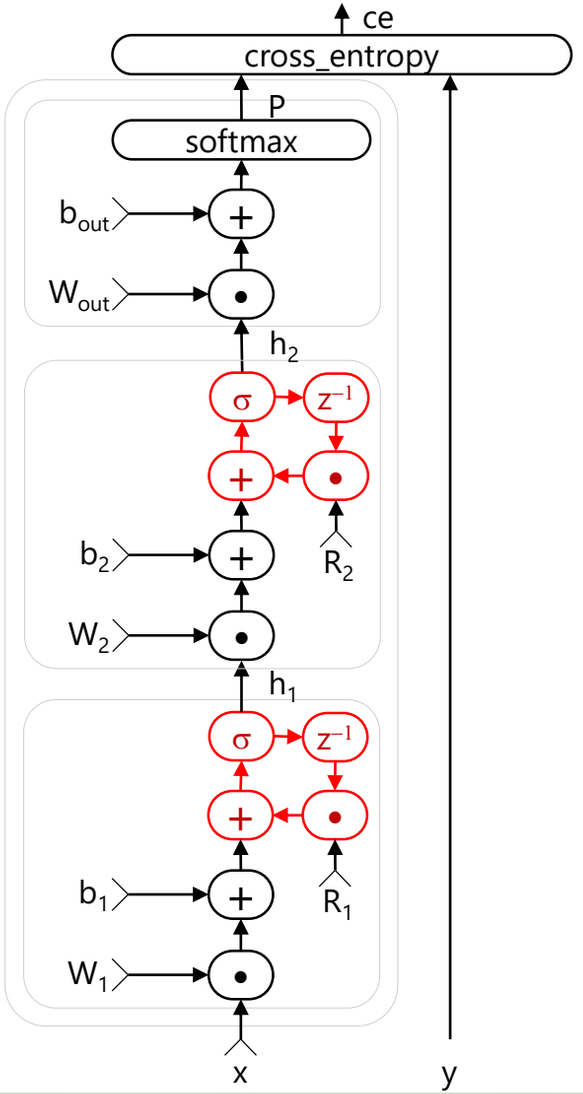
Symbolic Loops over Sequential Data

Extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1\, x(t) + \mathbf{R}_1\, h_1(t\text{-}1) + b_1)$$
$$h_2(t) = \sigma(\mathbf{W}_2\, h_1(t) + \mathbf{R}_2\, h_2(t\text{-}1) + b_2)$$
$$P(t) = \text{softmax}(\mathbf{W}_{\text{out}}\, h_2(t) + b_{\text{out}})$$
$$ce(t) = L^{\text{T}}(t) \log P(t)$$
$$\sum_{\text{corpus}} ce(t) = \max$$

```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

Symbolic Loops over Sequential Data

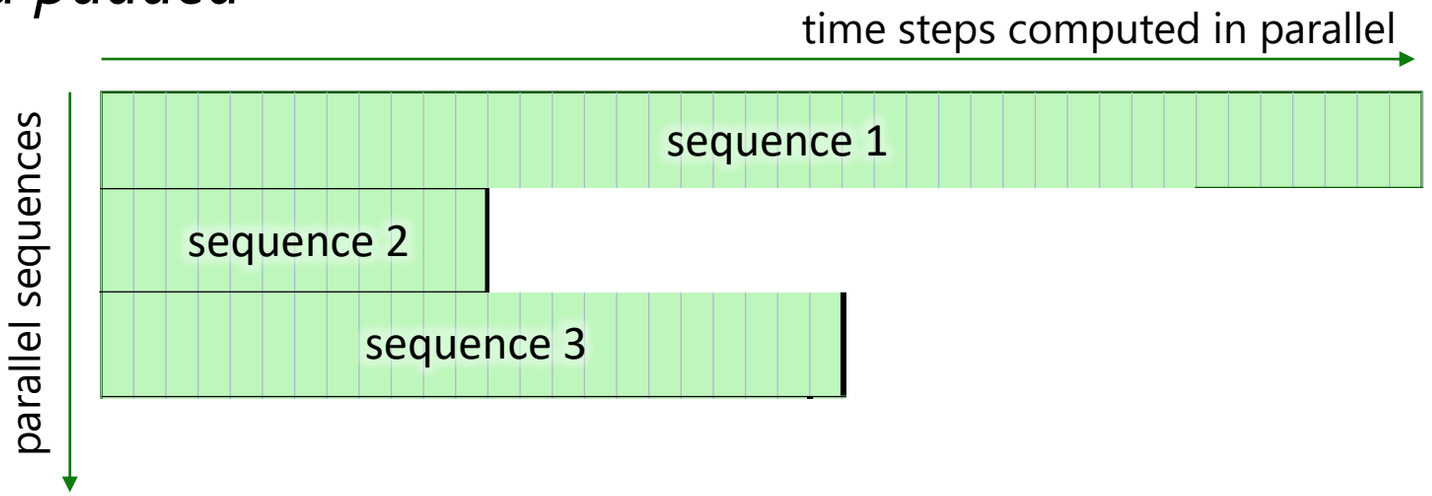



```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

- CNTK automatically unrolls cycles at *execution time*
  - cycles are detected with Tarjan's algorithm
- Efficient and composable

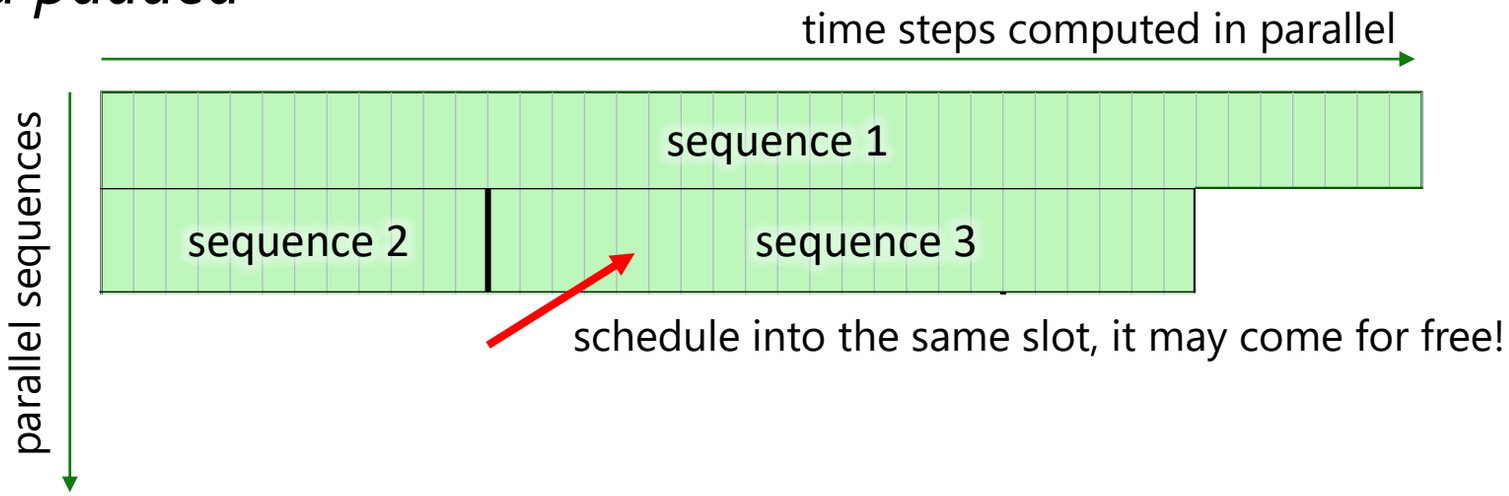
Batch-Scheduling of Variable-Length Sequences

- Minibatches containing sequences of different lengths are automatically packed *and padded*



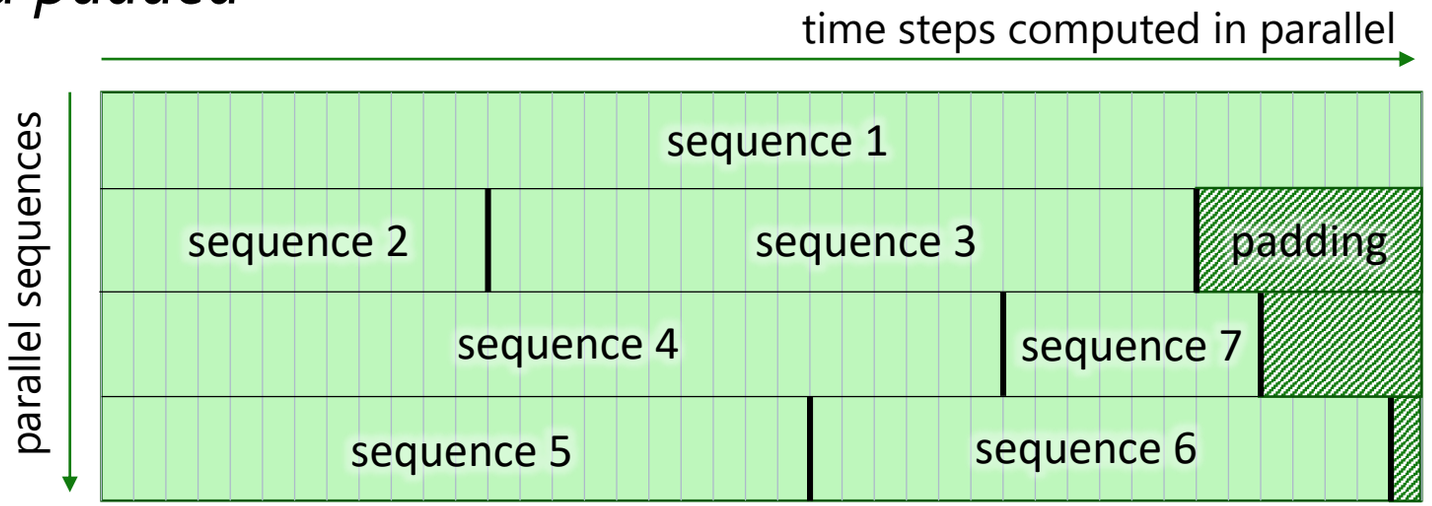
Batch-Scheduling of Variable-Length Sequences

- Minibatches containing sequences of different lengths are automatically packed *and padded*



Batch-Scheduling of Variable-Length Sequences

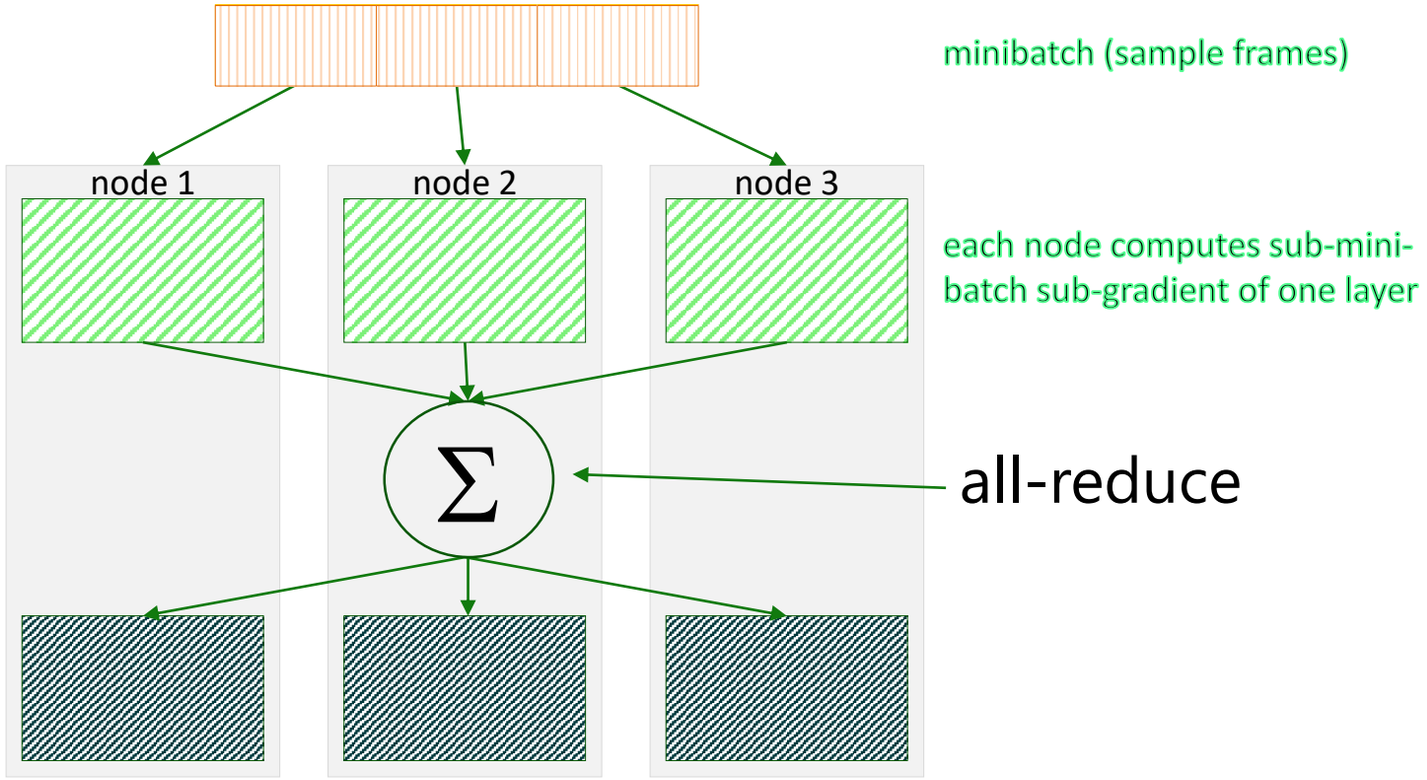
- Minibatches containing sequences of different lengths are automatically packed *and padded*



- Fully transparent batching
  - Recurrent → CNTK unrolls, handles sequence boundaries
  - Non-recurrent operations → parallel
  - Sequence reductions → mask

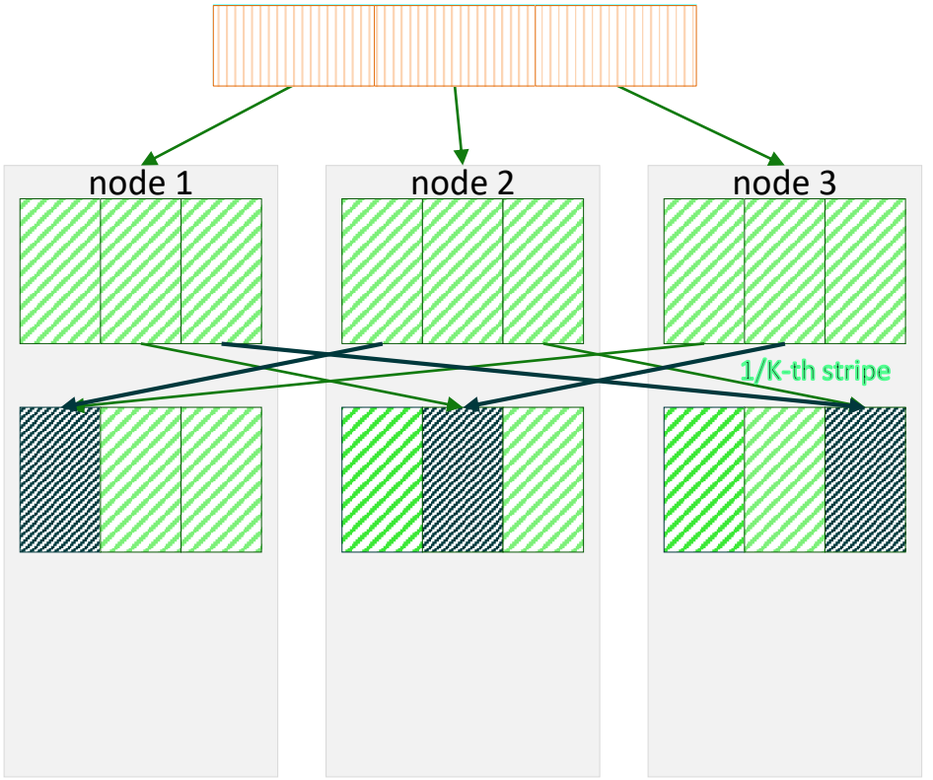
Data-Parallel Training

- Data-parallelism: distribute minibatch over workers, all-reduce partial gradients



Data-Parallel Training

- Data-parallelism: distribute minibatch over workers, all-reduce partial gradients



minibatch (sample frames)

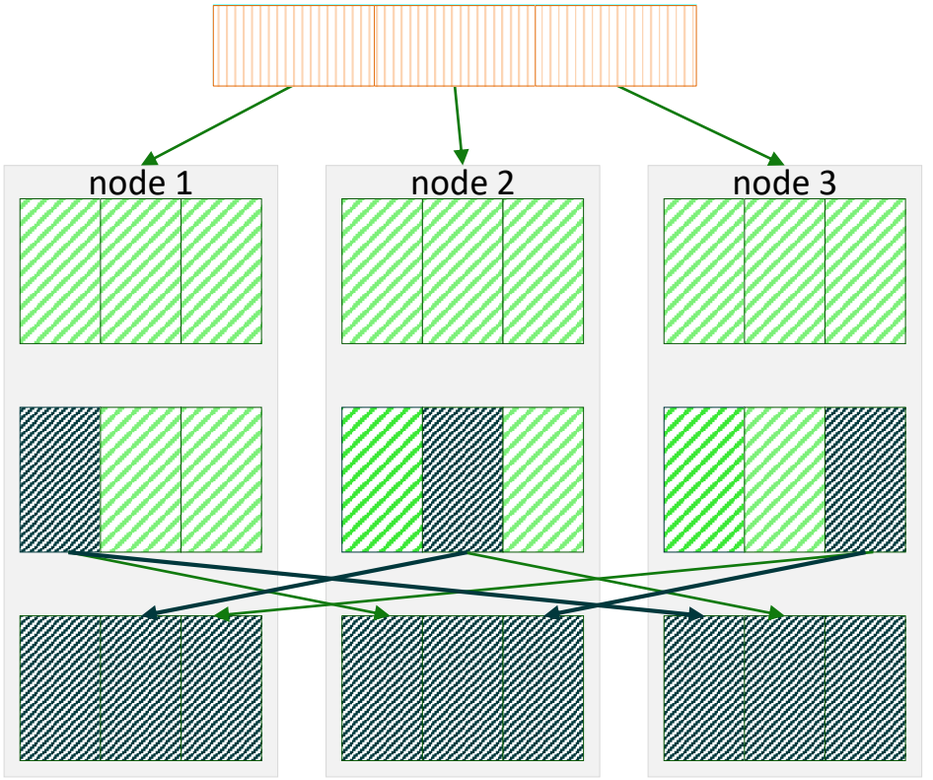
each node computes sub-mini-batch sub-gradient of one layer

(K-1) concurrent transfers of 1/K-th data

step 1: each node owns aggregating a stripe

Data-Parallel Training

- Data-parallelism: distribute minibatch over workers, all-reduce partial gradients



minibatch (sample frames)

each node computes sub-mini-batch sub-gradient of one layer

step 1: each node owns aggregating a stripe

step 2: aggregated stripes are redistributed

ring algorithm
$O(2\ (K\text{-}1)/K\ M)$
→ $O(1)$ w.r.t. K

Data-Parallel Training

- Data-parallelism: distribute minibatch over workers, all-reduce partial gradients
- $O(1)$ — enough?
- Example: DNN, MB size 1024, 160M model parameters
  - compute per MB: → 1/7 second
  - communication per MB: → 1/9 second (640M over 6 GB/s)
  - can't even parallelize to 2 GPUs: communication cost already dominates!
- How about doing it asynchronously?
  - HogWild! [Feng *et al.*, 2011], DistBelief ASGD [Dean *et al.*, 2012]
  - Helps with latency and jitter, could hide some communication cost with pipeline
  - Does not change the problem fundamentally

Data-Parallel Training

How to reduce communication cost:

communicate less each time

- 1-bit SGD: [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "1-Bit Stochastic Gradient Descent...Distributed Training of Speech DNNs", Interspeech 2014]
  - quantize gradients to 1 bit per value
  - trick: carry over quantization error to next minibatch

communicate less often

- Automatic MB sizing [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "ON Parallelizability of Stochastic Gradient Descent...", ICASSP 2014]
- Block momentum [K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training...," ICASSP 2016]
  - Very recent, very effective parallelization method
  - Combines model averaging with error-residual idea

Evaluation of CNTK Models

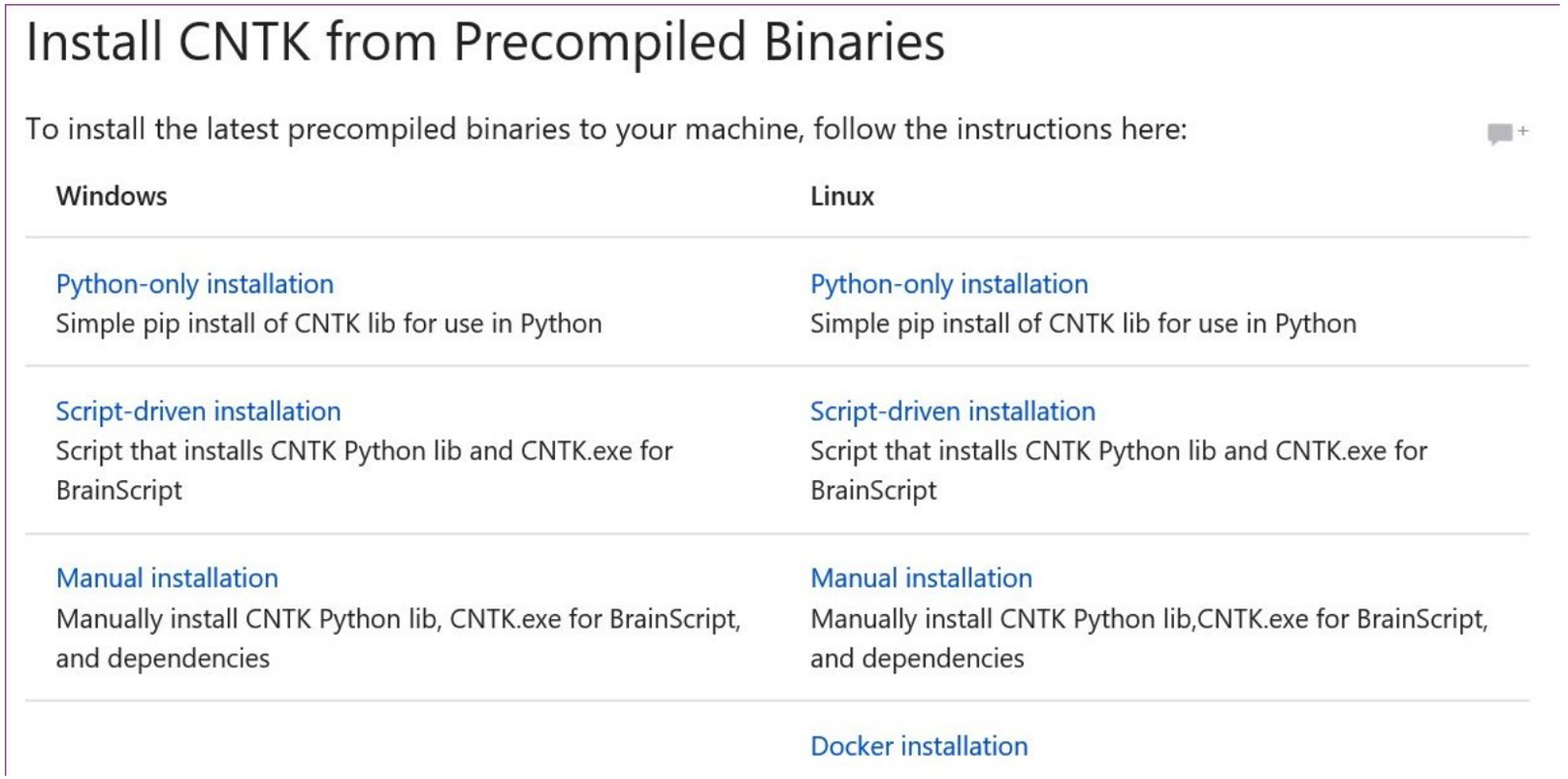
- Multi-language support
  - C++, C#/.NET, Python, Java, etc.
- More focused on direct integration of CNTK evaluation into user applications
- Parallel evaluation of multiple requests with very limited memory overhead

Toolkit Basics:

- Where to start
- Model train workflow
- Data Readers (with Augmentation)
- Extensibility
- Modeling components (MLP/CNN/RNN)

Installation

$ pip install <url>



Install CNTK from Precompiled Binaries

To install the latest precompiled binaries to your machine, follow the instructions here:

Windows	Linux
Python-only installation Simple pip install of CNTK lib for use in Python	Python-only installation Simple pip install of CNTK lib for use in Python
Script-driven installation Script that installs CNTK Python lib and CNTK.exe for BrainScript	Script-driven installation Script that installs CNTK Python lib and CNTK.exe for BrainScript
Manual installation Manually install CNTK Python lib, CNTK.exe for BrainScript, and dependencies	Manual installation Manually install CNTK Python lib,CNTK.exe for BrainScript, and dependencies
	Docker installation

https://docs.microsoft.com/en-us/cognitive-toolkit/Setup-CNTK-on-your-machine

Tutorials

Tutorials

1. *Classify cancer using simulated data (Logistic Regression)*
   CNTK 101: Logistic Regression with NumPy

2. *Classify cancer using simulated data (Feed Forward, FFN)*
   CNTK 102: Feed Forward network with NumPy

3. *Recognize hand written digits (OCR) with MNIST data*
   CNTK 103 Part A: MNIST data preparation, Part B: Multi-class logistic regression classifier
   Part C: Multi-layer perceptron classifier Part D: Convolutional neural network classifier

4. *Learn how to predict the stock market*
   CNTK 104: Time Series basics with finance data

5. *Compress (using autoencoder) hand written digits from MNIST data with no human input (unsupervised learning, FFN)*
   CNTK 105 Part A: MNIST data preparation, Part B: Feed Forward autoencoder

6. *Forecasting using data from an IOT device*
   CNTK 106: LSTM based forecasting - Part A: with simulated data, Part B: with real IOT data

7. *Recognize objects in images from CIFAR-10 data (Convolutional Network, CNN)*
   CNTK 201 Part A: CIFAR data preparation, Part B: VGG and ResNet classifiers

8. *Infer meaning from text snippets using LSTMs and word embeddings*
   CNTK 202: Language understanding

9. *Train a computer to perform tasks optimally (e.g., win games) in a simulated environment*
   CNTK 203: Reinforcement learning basics with OpenAI Gym data

10. *Translate text from one domain (grapheme) to other (phoneme)*
    CNTK 204: Sequence to sequence basics with CMU pronouncing dictionary

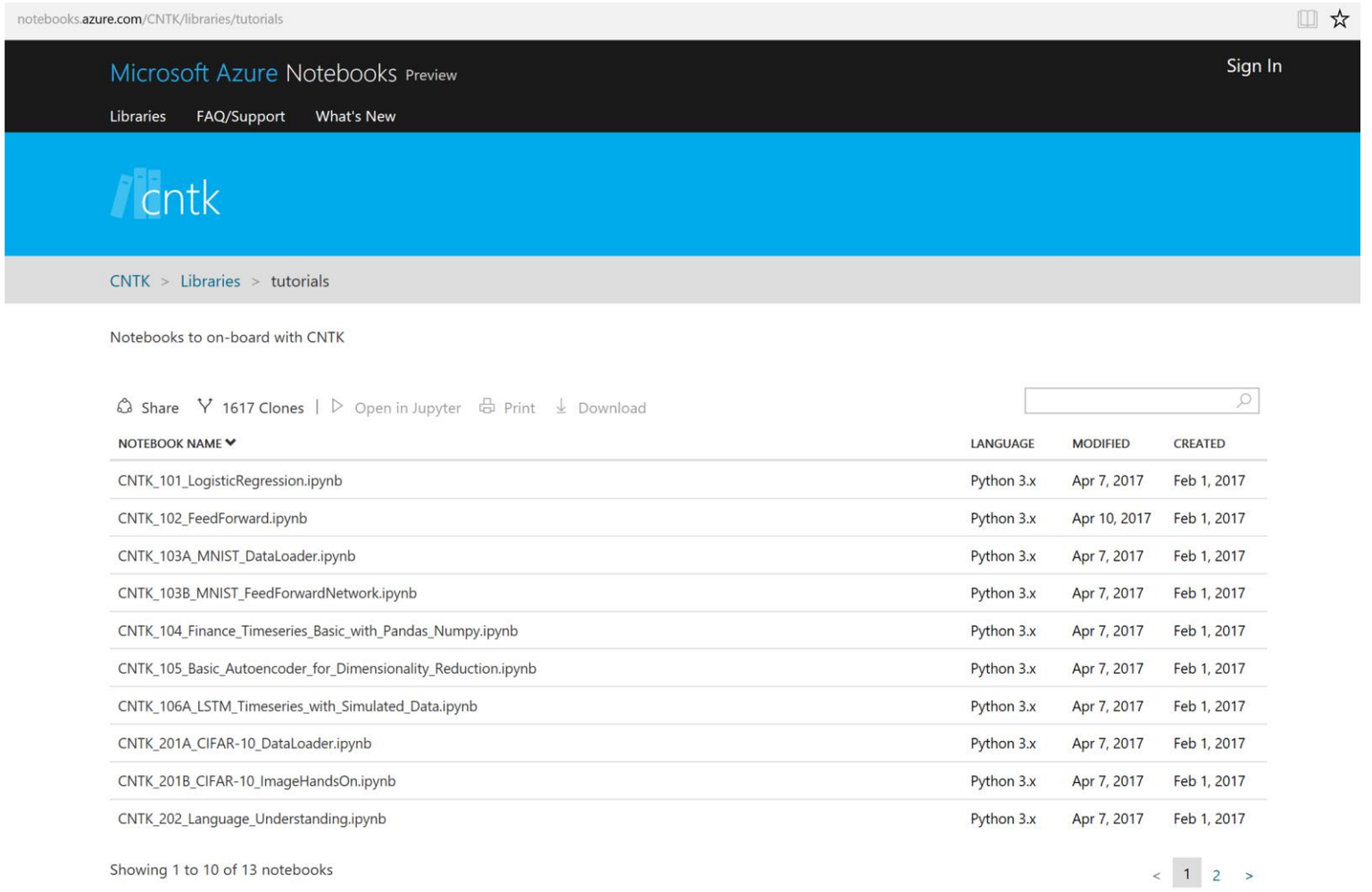
11. *Teach a computer to paint like Picasso or van Gogh*
    CNTK 205: Artistic Style Transfer

12. *Produce realistic data (MNIST images) with no human input (unsupervised learning)*
    CNTK 206 Part A: MNIST data preparation, Part B: Basic Generative Adversarial Networks (GAN), Part B: Deep Convolutional GAN

13. *Training with Sampled Softmax*
    CNTK 207: Training with Sampled Softmax

14. *Recognize flowers and animals in natural scene images using deep transfer learning*
    CNTK 301: Deep transfer learning with pre-trained ResNet model

Tutorials on Azure (Free Pre-Hosted)

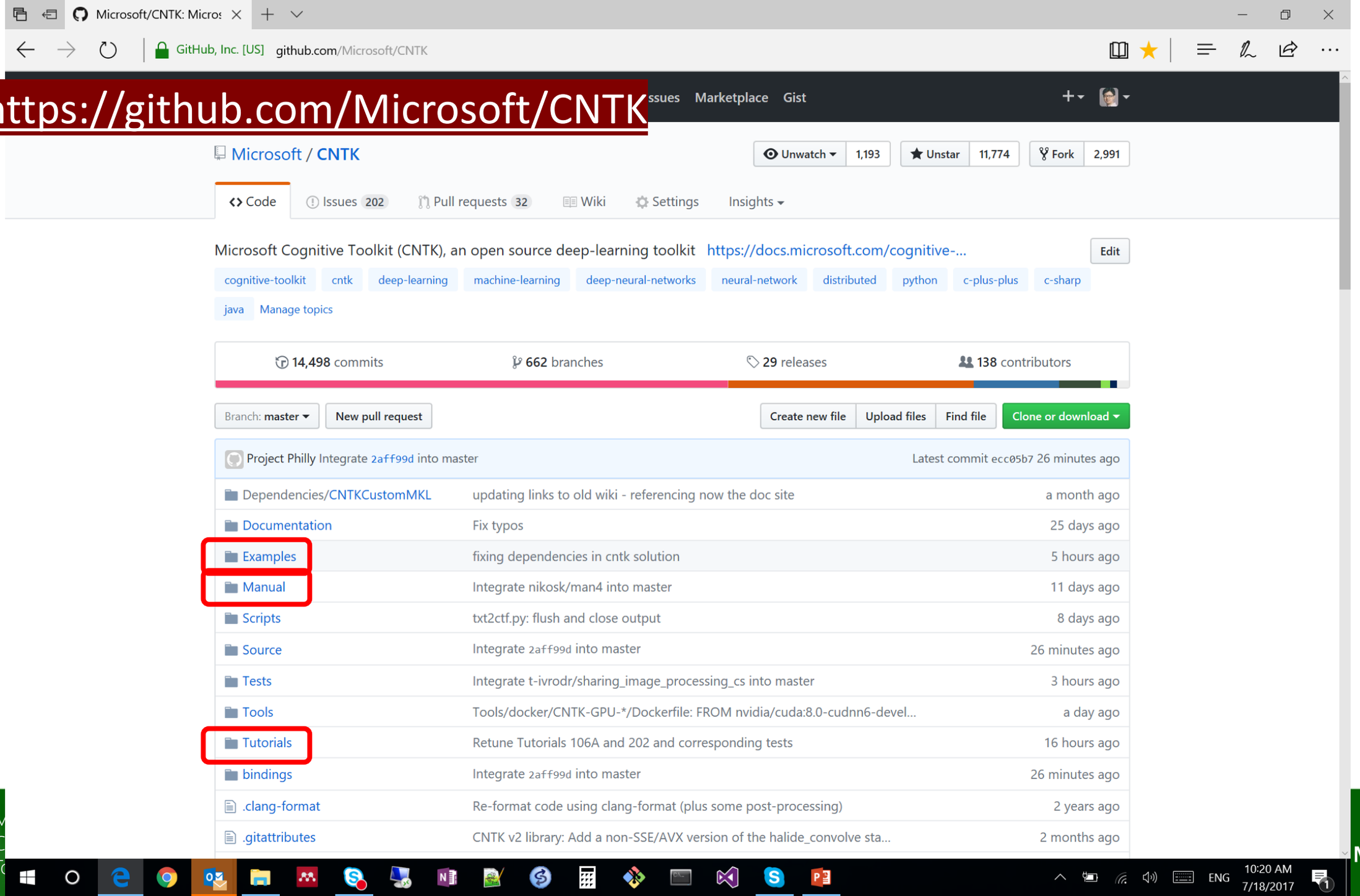


Microsoft Cognitive Toolkit

https://notebooks.azure.com/cntk/libraries/tutorials

Microsoft

https://github.com/Microsoft/CNTK



Microsoft

The Microsoft Cognitive Toolkit

2017-6-1 • 1 min to read • Contributors all

The Microsoft Cognitive Toolkit - CNTK - is a unified deep-learning toolkit by Microsoft. This video provides a high-level overview of the toolkit.

The latest release of the Microsoft Cognitive Toolkit is 2.0.

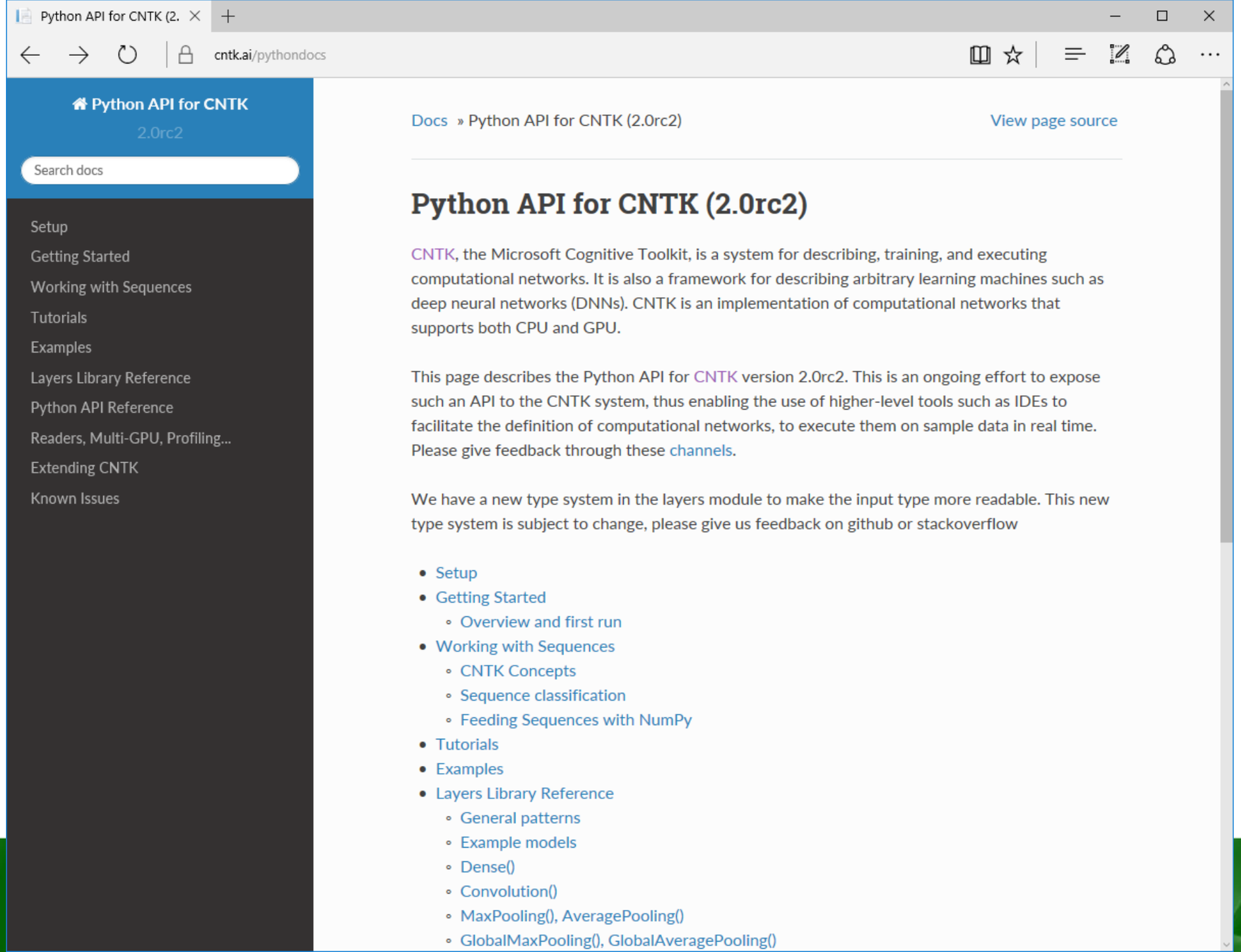
CNTK can be included as a library in your Python or C++ programs, or used as a standalone machine learning tool through its own model description language (BrainScript). In addition you can use the CNTK model evaluation functionality from your C# or Java program.

CNTK supports 64-bit Linux or 64-bit Windows operating systems. To install you can either choose pre-compiled binary packages, or compile the toolkit from the source provided in GitHub.

Here are a few pages to get started:

- Reasons to switch from TensorFlow to CNTK
- Setting up CNTK on your machine
- Tutorials, Examples, Tutorials on Azure
- The CNTK Library APIs
  - Using CNTK from Python
    - CNTK with Keras
  - Using CNTK from C++
- CNTK using BrainScript
- CNTK Model Evaluation
- How to contribute to CNTK
- Give us feedback through these channels

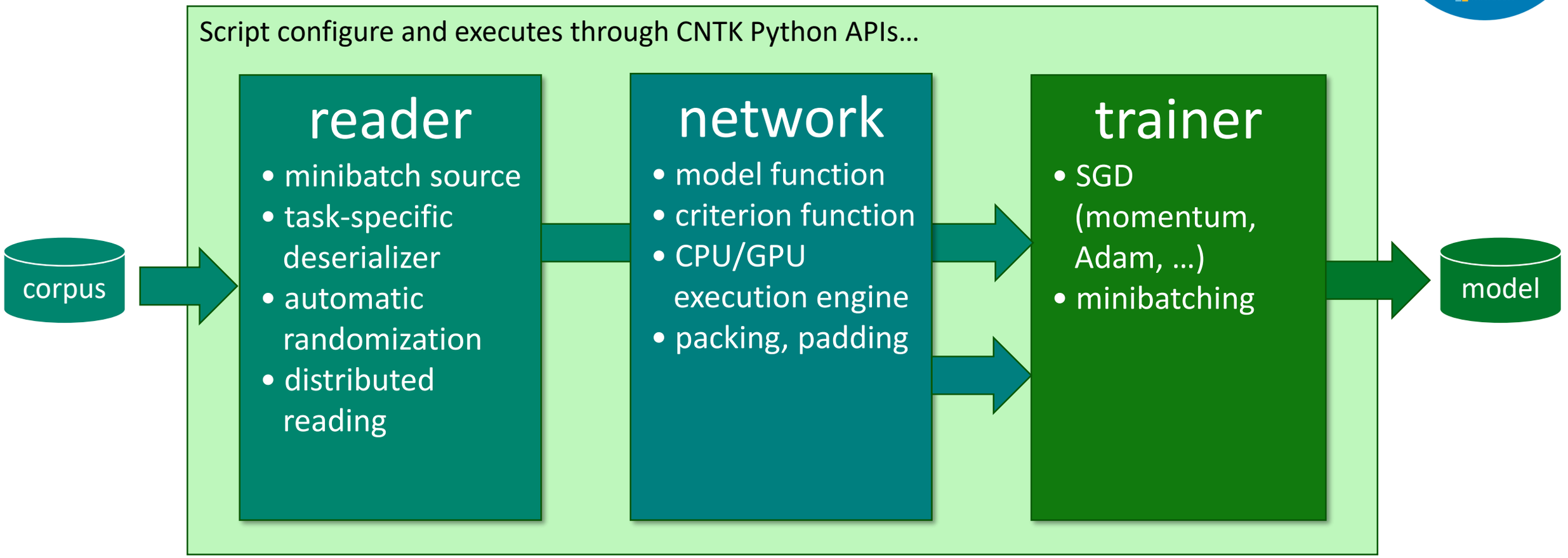
0 Comments



Microsoft Cognitive Toolkit

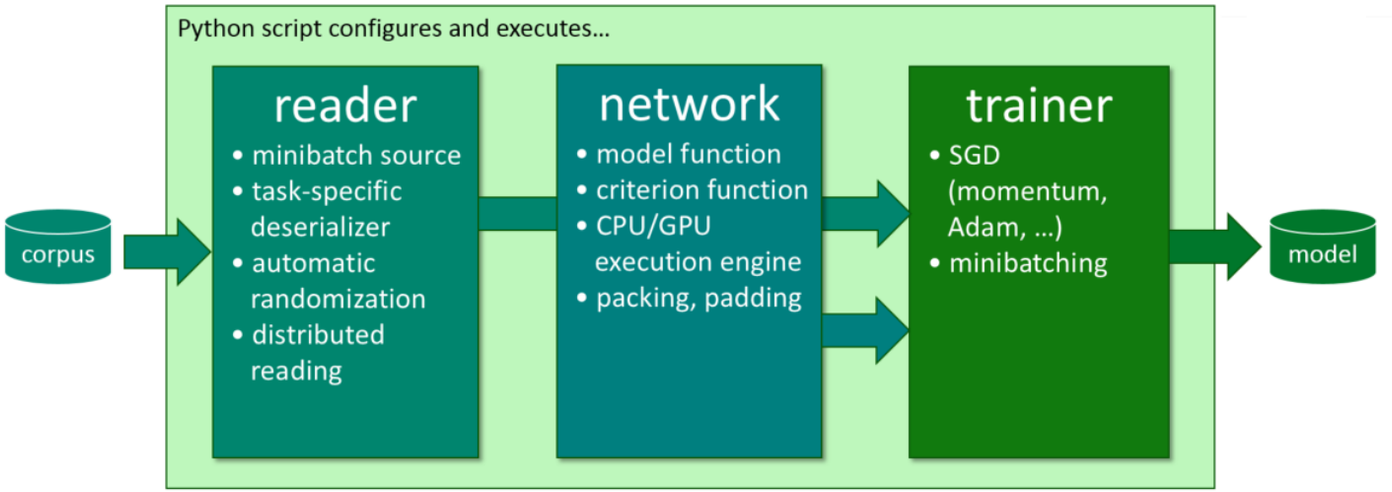
Microsoft

CNTK Workflow

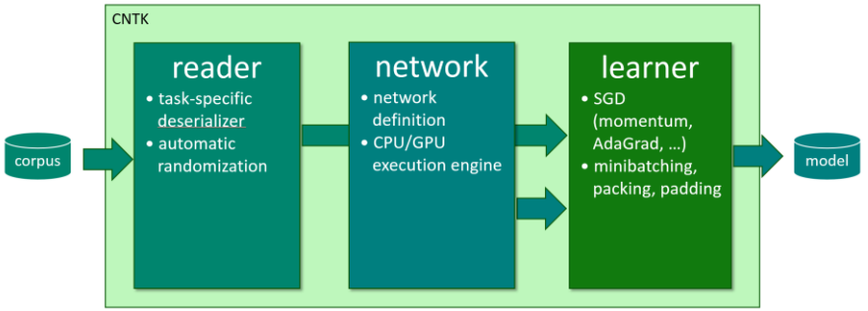


As Easy as 1-2-3

```
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)
```



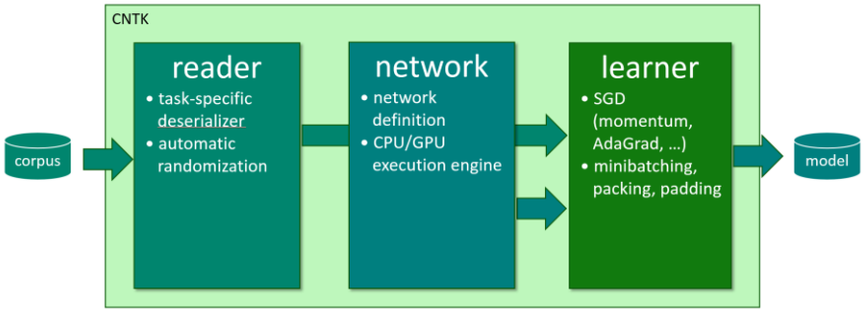
Workflow



- Prepare data
- Configure reader, network, learner (Python)
- Train:

```
python my_cntk_script.py
```

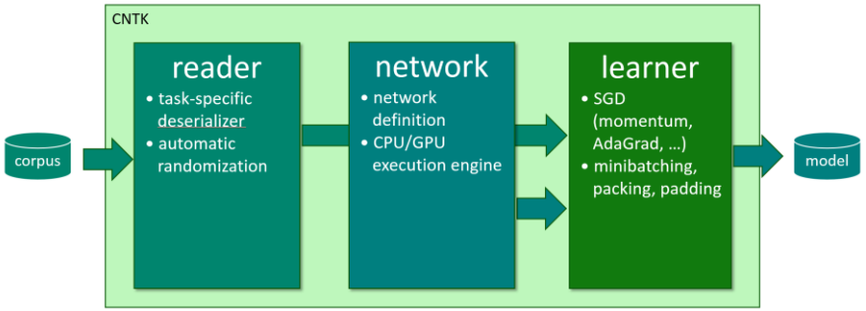
Prepare Data: Reader




```
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)
```

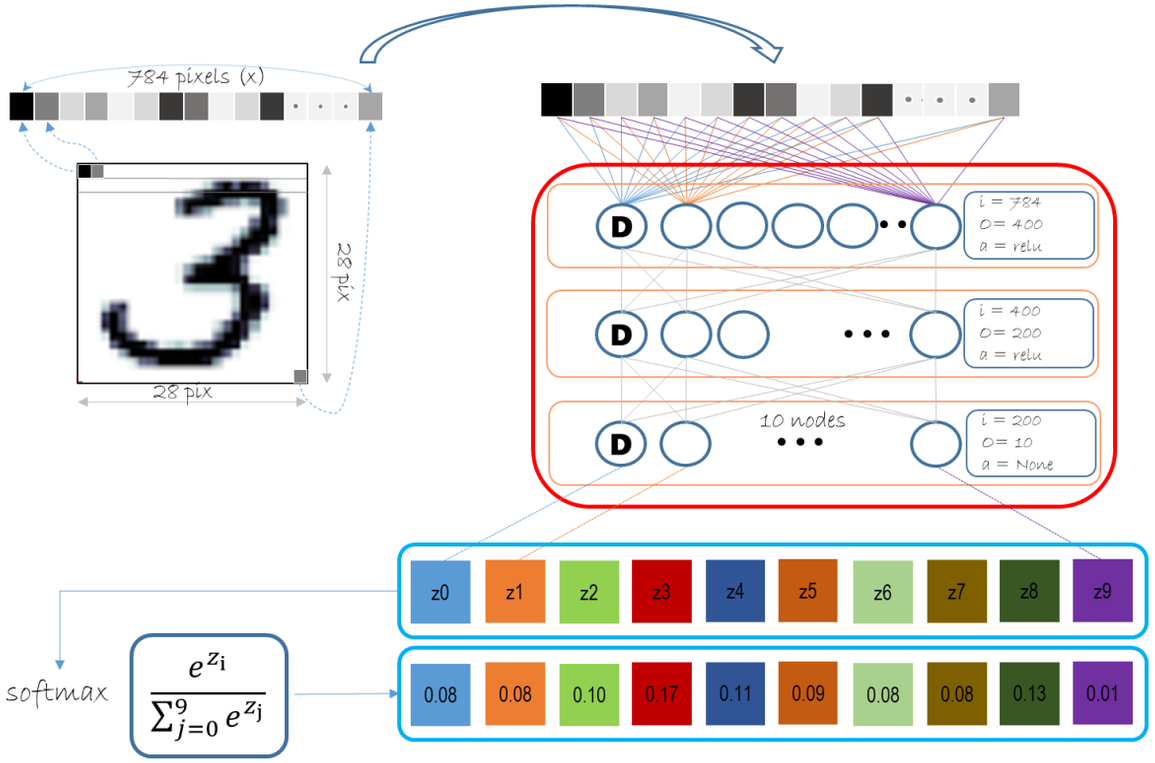
Prepare Data: Reader




```
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)
    ]
    # 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)
```

- Automatic on-the-fly randomization important for large data sets
- Readers compose, e.g. image → text caption

Prepare Network: Multi-Layer Perceptron



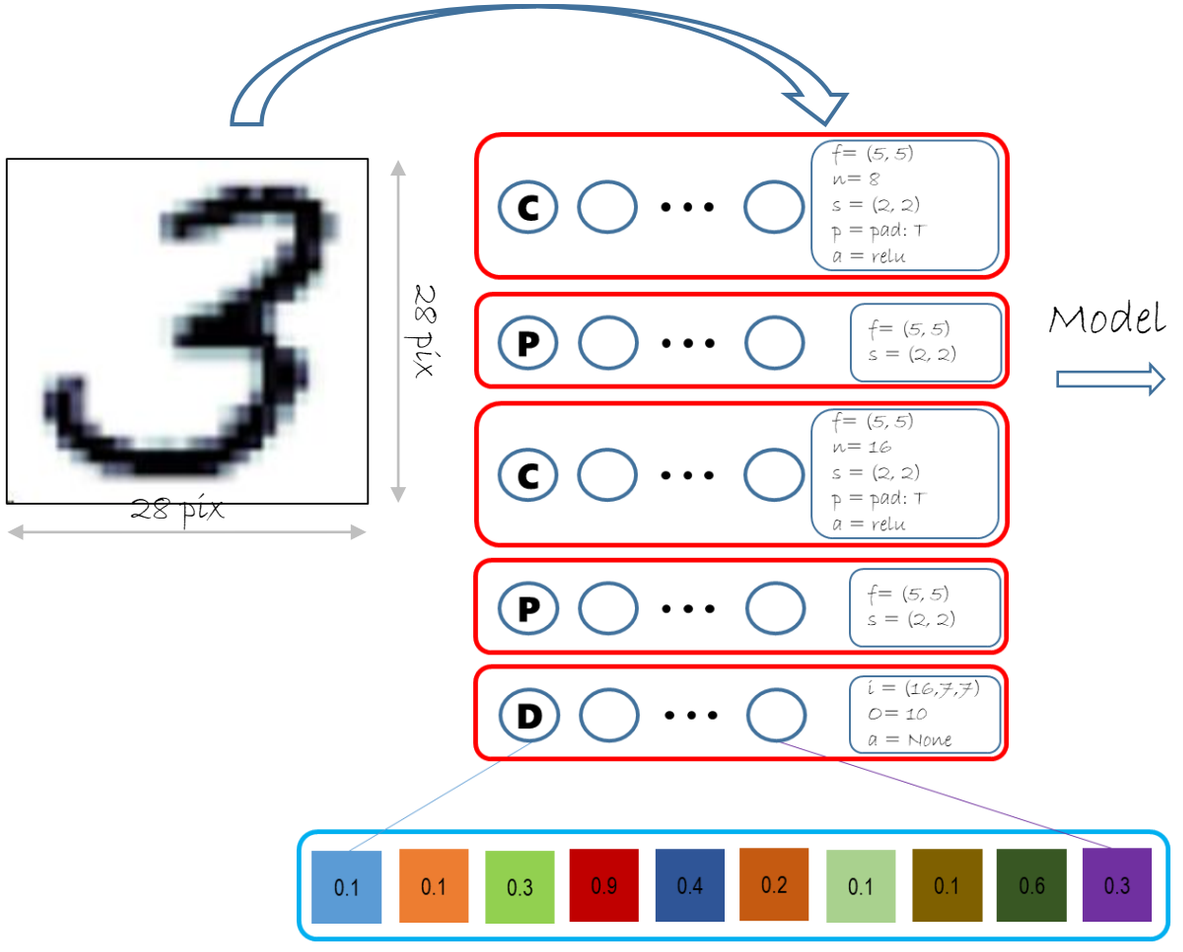
Model

```
z = model(x):
        h1 = Dense(400, act = relu)(x)
        h2 = Dense(200, act = relu)(h1)
        r  = Dense(10,  act = None)(h2)
        return r
```

Loss

```
cross_entropy_with_softmax(z,Y)
```

Prepare Network: Convolutional Neural Network

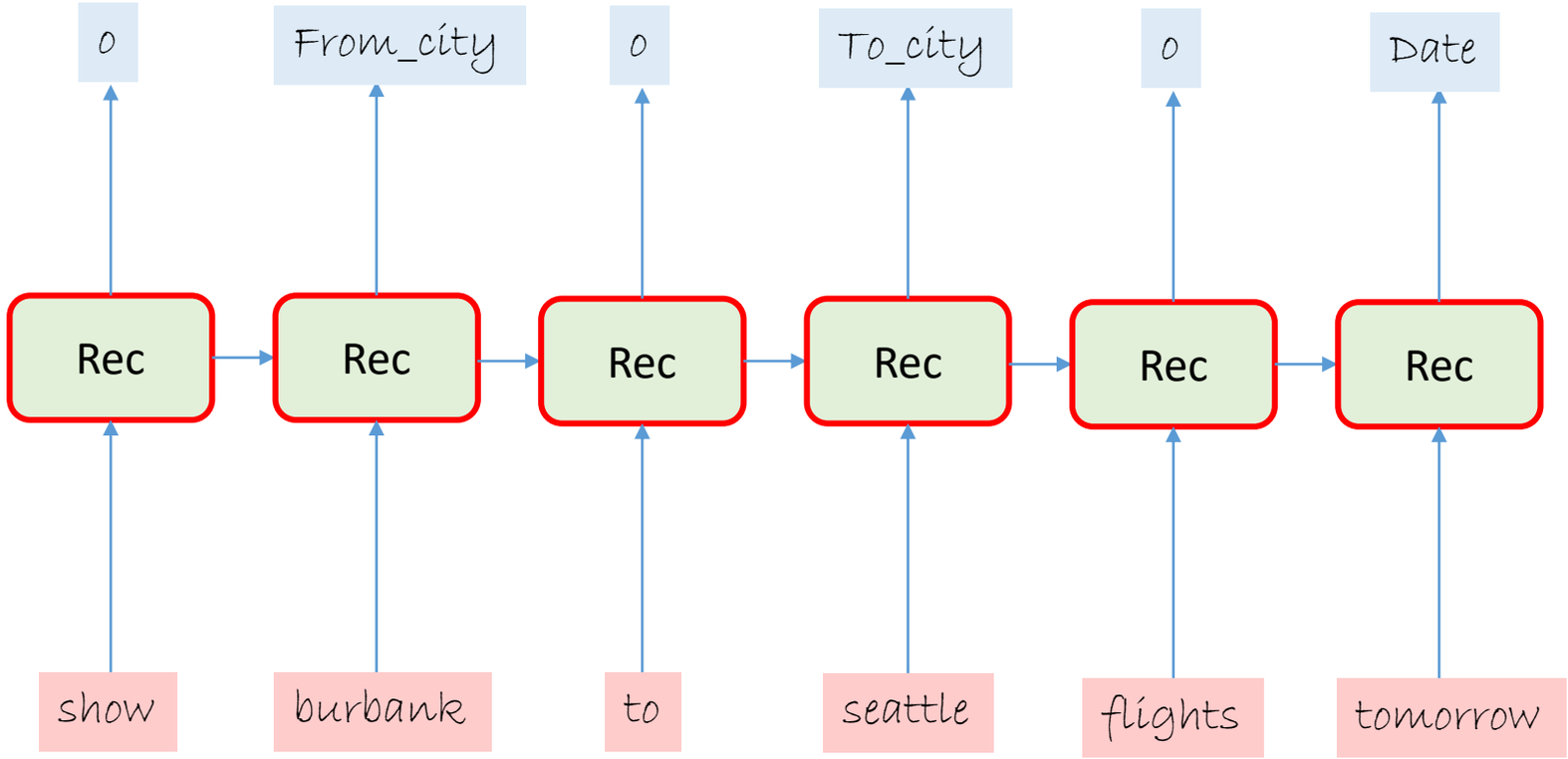


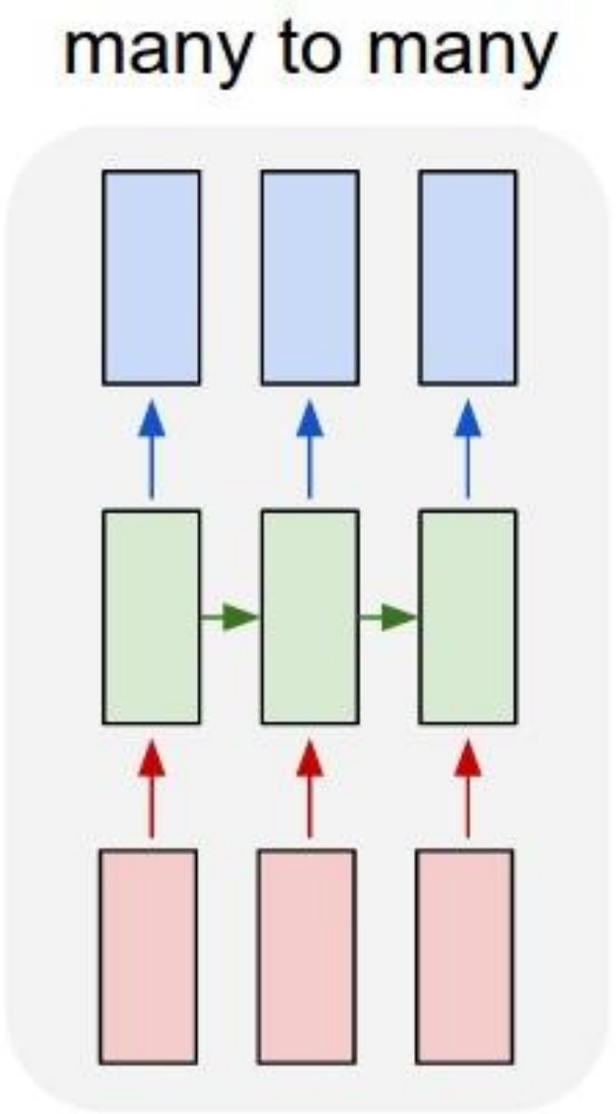
Model

```
z = model(x):
    h = Convolution2D((5,5),filt=8, …)(x)
    h = MaxPooling(…)(h)
    h = Convolution2D ((5,5),filt=16, …)((h)
    h = MaxPooling(…)(h)
    r = Dense(output_classes, act= None)(h)
    return r
```

An Sequence Example (many to many + 1:1)

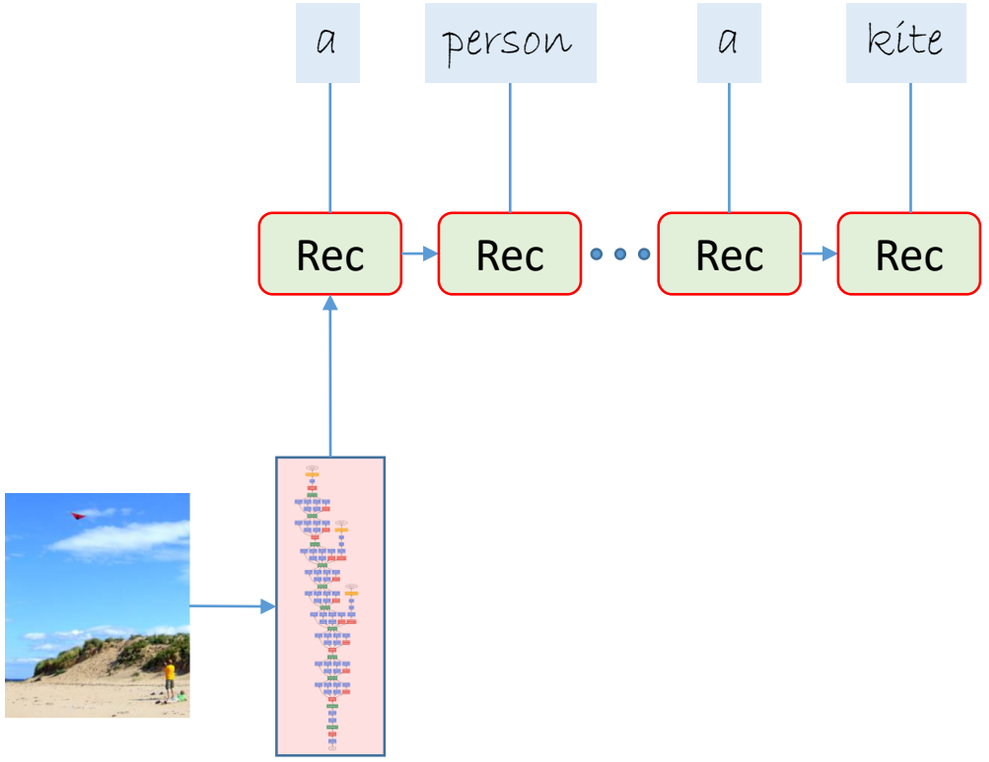
Problem: Tagging entities in Air Traffic Controller (ATIS) data

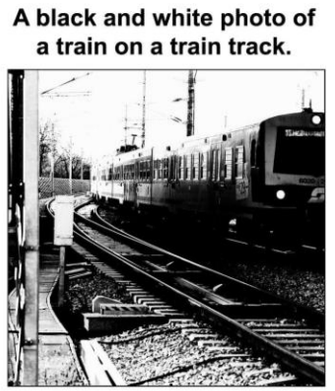


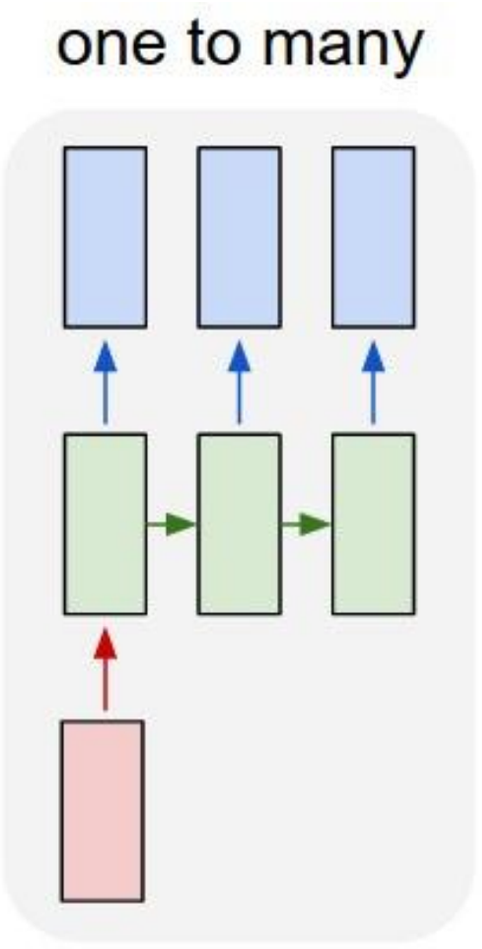


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Another Sequence Example (one to many)



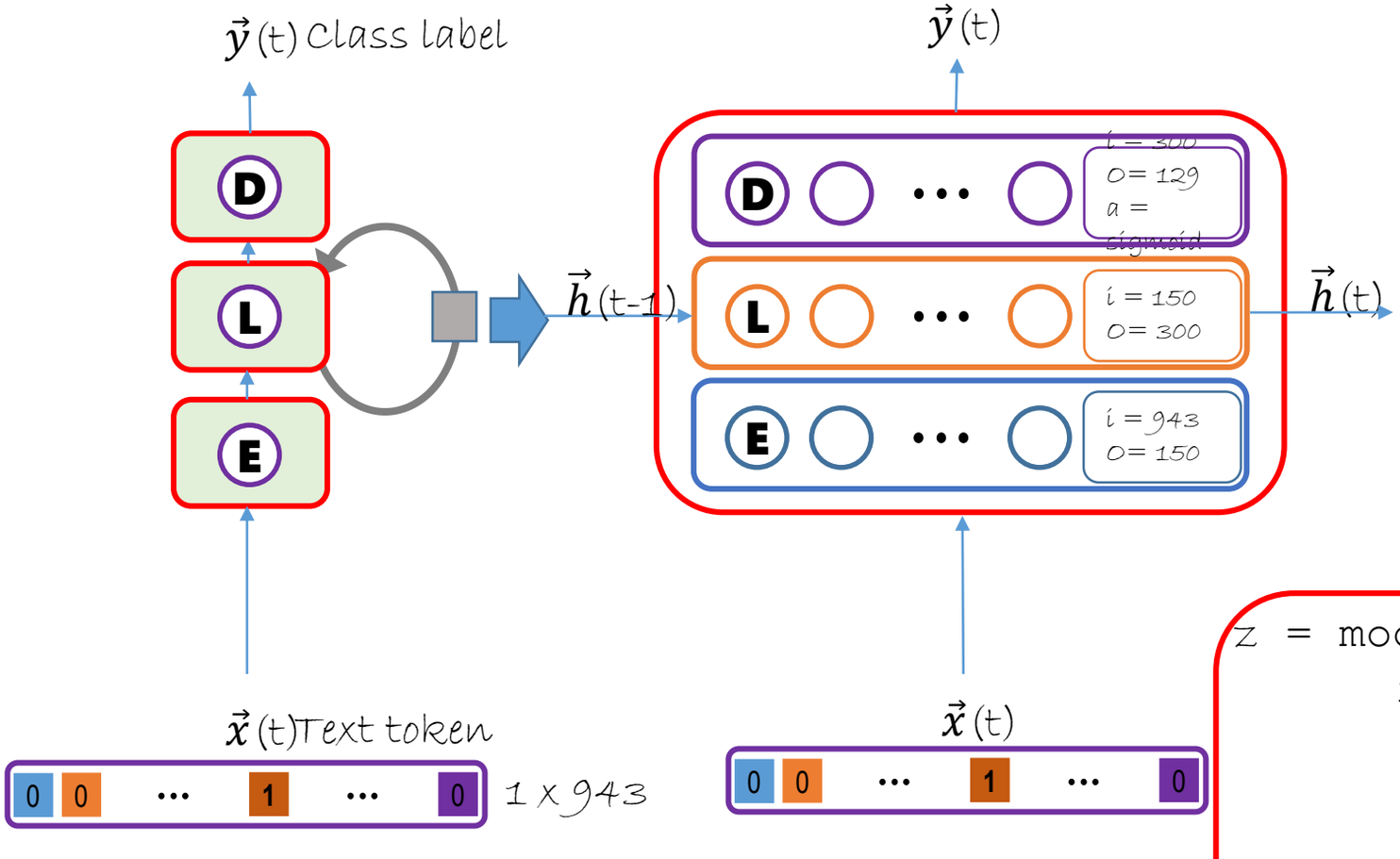




http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Vinyals et al (https://arxiv.org/abs/1411.4555)

Prepare Network: Recurrent Neural Network



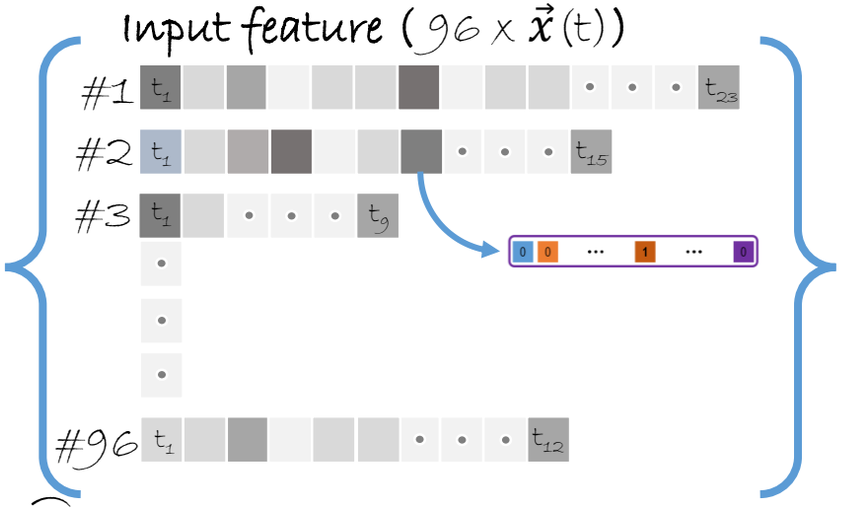

```
z = model():
      return
        Sequential([
            Embedding(emb_dim=150),
            Recurrence(LSTM(hidden_dim=300),
                       go_backwards=False),
            Dense(num_labels = 129)
        ])
```

Prepare Learner

- Many built-in learners
  - SGD, SGD with momentum, Adagrad, RMSProp, Adam, Adamax, AdaDelta, etc.
- Specify learning rate schedule and momentum schedule
- If wanted, specify minibatch size schedule

```
lr_schedule = C.learning_rate_schedule([0.05]*3 + [0.025]*2 + [0.0125],
                                       C.UnitType.minibatch, epoch_size=100)
sgd_learner = C.sgd(z.parameters, lr_schedule)
```

Overall Train Workflow




```
z = model():
        return
          Sequential([
              Embedding(emb_dim=150),
              Recurrence(LSTM(hidden_dim=300),
                         go_backwards=False),
              Dense(num_labels = 129)
          ])
```

96 samples (mini-batch)

One-hot encoded Label (Y: 96 x 129/sample Or word in sequence)

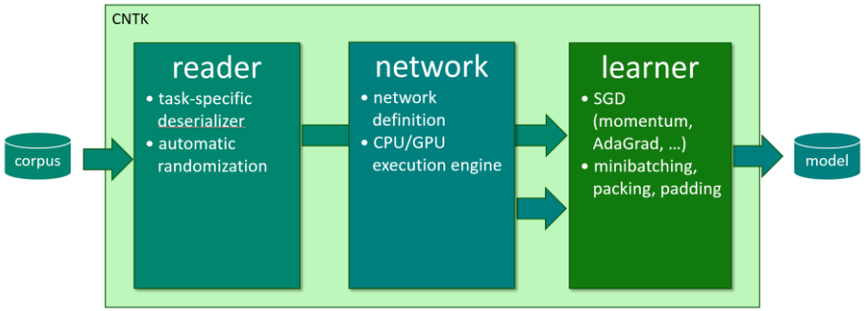
Loss `cross_entropy_with_softmax(z,Y)`

Error `classification_error(z,Y)`

Choose a learner
(SGD, Adam, adagrad etc.)

```
Trainer(model,  (loss, error),  learner)

Trainer.train_minibatch({X,  Y})
```

Distributed training



- Prepare data
- Configure reader, network, learner (Python)
- Train: -- distributed*!*

```
mpiexec --np 16 --hosts server1,server2,server3,server4    \
python my_cntk_script.py
```

Extensibility: Custom Layer with Built-in Ops

```
def concat_elu(x):
    """ like concatenated ReLU (http://arxiv.org/abs/1603.05201), but then with ELU """
    return cntk.elu(cntk.splice(x, -x, axis=0))

def selu(x, scale, alpha):
    return cntk.element(scale, cntk.element_select(cntk.less(x, 0), alpha * cntk.elu(x), x))


def log_prob_from_logits(x, axis):
    """ numerically stable log_softmax implementation that prevents overflow """
    m = cntk.reduce_max(x, axis)
    return x - m - cntk.log(cntk.reduce_sum(cntk.exp(x-m), axis=axis))
```

Extensibility: Custom Layer with Pure Python

```
class MySigmoid(UserFunction):
    def __init__(self, arg, name='MySigmoid'):
        super(MySigmoid, self).__init__([arg], name=name)

    def forward(self, argument, device=None, outputs_to_retain=None):
        sigmoid_x = 1/(1+numpy.exp(-argument))
        return sigmoid_x, sigmoid_x

    def backward(self, state, root_gradients):
        sigmoid_x = state
        return root_gradients * sigmoid_x * (1 - sigmoid_x)

    def infer_outputs(self):
        return [cntk.output_variable(self.inputs[0].shape,
                 self.inputs[0].dtype, self.inputs[0].dynamic_axes)]
```

Extensibility: Custom Learner

```
def my_rmsprop(parameters, gradients):
    rho = 0.999
    lr = 0.01
    # We use the following accumulator to store the moving average of every squared gradient
    accumulators = [C.constant(1e-6, shape=p.shape, dtype=p.dtype) for p in parameters]
    update_funcs = []
    for p, g, a in zip(parameters, gradients, accumulators):
        # We declare that `a` will be replaced by an exponential moving average of squared gradients
        # The return value is the expression rho * a + (1-rho) * g * g
        accum_new = cntk.assign(a, rho * a + (1-rho) * g * g)
        # This is the rmsprop update.
        # We need to use accum_new to create a dependency on the assign statement above.
        # This way, when we run this network both assigns happen.
        update_funcs.append(cntk.assign(p, p - lr * g / cntk.sqrt(accum_new)))
    return cntk.combine(update_funcs)

my_learner = cntk.universal(my_rmsprop, z.parameters)
```

Image Networks:

- VGG, ResNet and Inception
- Emotion Recognition
- Faster R-CNN
- Artistic Neural Style
- GAN / PixelCNN
- Image Captioning

VGG, ResNet and Inception

Take away

- Show CNTK high level API
- Show how to implement these popular network models

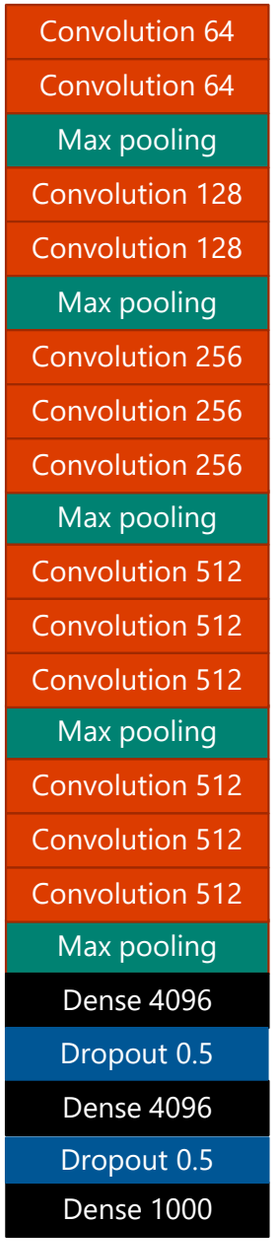
Visual Geometry Group (VGG network)

K. Simonyan, A. Zisserman
“Very Deep Convolutional Networks for Large-Scale Image Recognition”,
CoRR 2014

https://github.com/Microsoft/CNTK/tree/master/Examples/Image/Classification/VGG

VGG16

```
with C.layers.default_options(activation=C.relu, init=C.glorot_uniform()):
 return C.layers.Sequential([
            C.layers.For(range(2), lambda i: [
                C.layers.Convolution((3,3), [64,128][i], pad=True),
                C.layers.Convolution((3,3), [64,128][i], pad=True),
                C.layers.MaxPooling((3,3), strides=(2,2))
            ]),
             C.layers.For(range(3), lambda i: [
                C.layers.Convolution((3,3), [256,512,512][i], pad=True),
                C.layers.Convolution((3,3), [256,512,512][i], pad=True),
                C.layers.Convolution((3,3), [256,512,512][i], pad=True),
                C.layers.MaxPooling((3,3), strides=(2,2))
            ]),
            C.layers.For(range(2), lambda : [
                C.layers.Dense(4096),
                C.layers.Dropout(0.5)
            ]),
            C.layers.Dense(out_dims, None)])(input)
```



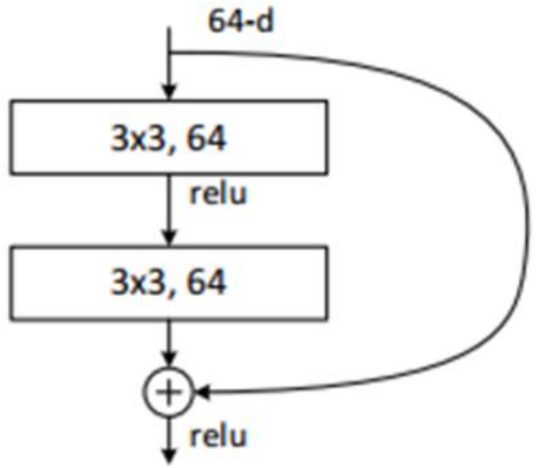
Residual Network (ResNet)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
“Deep Residual Learning for Image Recognition”, CVPR 2016

https://github.com/Microsoft/CNTK/tree/master/Examples/Image/Classification/ResNet

Residual Network (ResNet)

```
def conv_bn(input, filter_size, num_filters, strides=(1,1), activation=C.relu):
    c = Convolution(filter_size, num_filters, None)(input)
    r = BatchNormalization(…)(c)
    if activation != None:
        r = activation(r)
    return r

def resnet_basic(input, num_filters):
    c1 = conv_bn(input, (3,3), num_filters)
    c2 = conv_bn(c1, (3,3), num_filters, activation=None)
    return C.relu(c2 + input)

def resnet_basic_inc(input, num_filters, strides=(2,2)):
    c1 = conv_bn (input, (3,3), num_filters, strides)
    c2 = conv_bn(c1, (3,3), num_filters, activation=None)
    s  = conv_bn(input, (1,1), num_filters, strides, None)
    p  = c2 + s
    return relu(p)
```



Residual Network (ResNet)

```
def create_resnet_model(input, out_dims):
    c = conv_bn(input, (3,3), 16)
    r1_1 = resnet_basic_stack(c, 16, 3)

    r2_1 = resnet_basic_inc(r1_1, 32)
    r2_2 = resnet_basic_stack(r2_1, 32, 2)

    r3_1 = resnet_basic_inc(r2_2, 64)
    r3_2 = resnet_basic_stack(r3_1, 64, 2)

    pool = C.layers.GlobalAveragePooling()(r3_2)
    net = C.layers.Dense(out_dims,
                         C.he_normal(),
                         None)(pool)
```

```
def resnet_basic_stack(input,
                       num_filters,
                       num_stack):
    r = input
    for _ in range(num_stack):
        r = resnet_basic(r, num_filters)
    return r
```

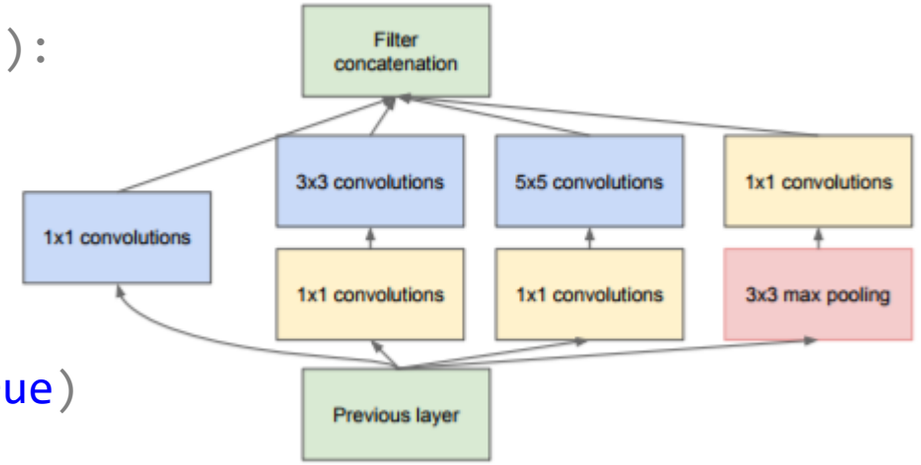
Inception Network (GoogLeNet)

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich
“Going Deeper with Convolutions”, CVPR 2015

https://github.com/Microsoft/CNTK/tree/master/Examples/Image/Classification/GoogLeNet

Inception Network

```
inception_block(input, num1x1, num3x3, num5x5, num_pool):
    # 1x1 Convolution
    branch1x1 = conv_bn(input, num1x1, (1,1), True)

    # 3x3 Convolution
    branch3x3 = conv_bn(input, num3x3[0], (1,1), True)
    branch3x3 = conv_bn(branch3x3, num3x3[1], (3,3), True)

    # 5x5 Convolution
    branch5x5 = conv_bn(input, num5x5[0], (1,1), True)
    branch5x5 = conv_bn(branch5x5, num5x5[1], (5,5), True)

    # Max pooling
    branch_pool = C.layers.MaxPooling((3,3), True)(input)
    branch_pool = conv_bn(branch_pool, num_pool, (1,1), True)

    return C.splice(branch1x1, branch3x3, branch5x5, branch_pool)
```



Emotion Recognition

Emad Barsoum, Cha Zhang, Cristian Canton Ferrer and Zhengyou Zhang
"Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution", ICMI 2016

https://github.com/Microsoft/FERPlus

Emotion Recognition

Take away

- Show how to implement a custom loss in CNTK.
- Show a end-to-end application in CNTK.

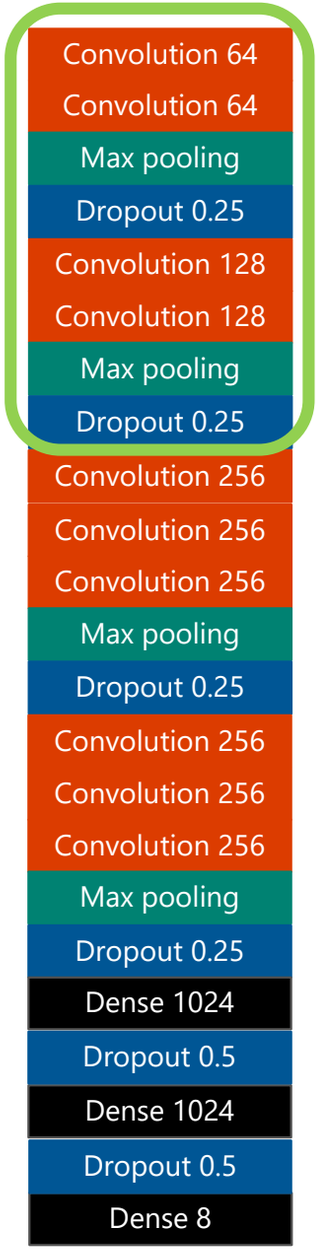
Emotion Recognition

- Recognize emotion from facial appearance.
- Each face can express multiple emotions.
- For each training image and for all the 8 emotion:
  - we compute a per emotion probability for that image
- We will show how to implement 4 different custom loss functions in CNTK.



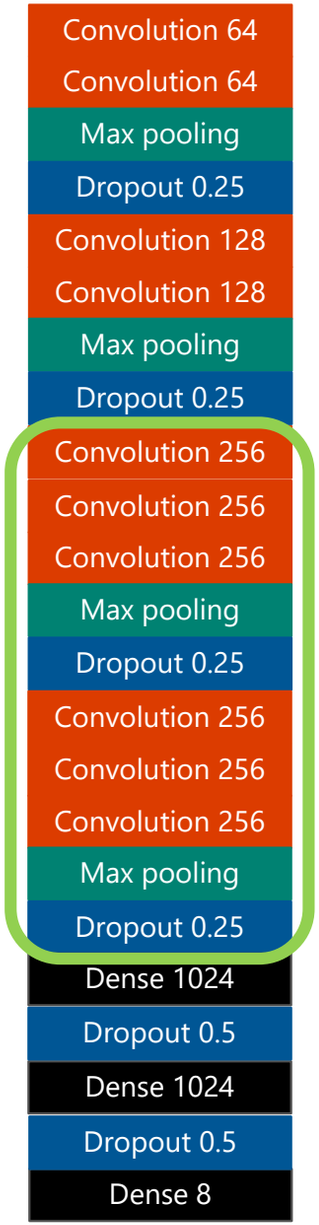

Emotion Network in CNTK

```
with C.default_options(activation=C.relu, init=C.glorot_uniform()):
    model = C.layers.Sequential([
                C.layers.For(range(2), lambda i: [
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2): lambda :[
                   C.layers.For(range(3)): lambda :[
                        C.layers.Convolution((3,3), 256, pad=True)
                    ]),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2), lambda : [
                    C.layers.Dense(1024),
                    C.layers.Dropout(0.5)
                ]), C.layers.Dense(num_classes, None)])
```



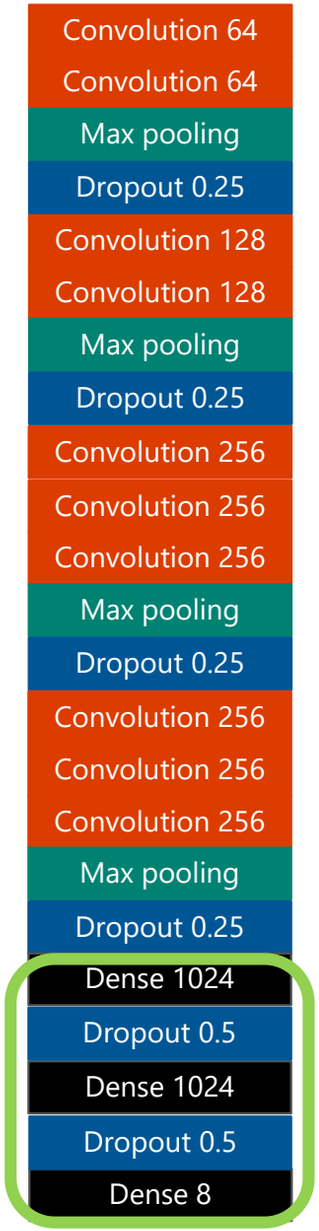
Emotion Network in CNTK

```
with C.default_options(activation=C.relu, init=C.glorot_uniform()):
    model = C.layers.Sequential([
                C.layers.For(range(2), lambda i: [
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2): lambda :[
                   C.layers.For(range(3)): lambda :[
                       C.layers.Convolution((3,3), 256, pad=True)
                    ]),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2), lambda : [
                    C.layers.Dense(1024),
                    C.layers.Dropout(0.5)
                ]), C.layers.Dense(num_classes, None)])
```



Emotion Network in CNTK

```
with C.default_options(activation=C.relu, init=C.glorot_uniform()):
    model = C.layers.Sequential([
                C.layers.For(range(2), lambda i: [
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.Convolution((3,3), [64,128][i], pad=True),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2): lambda :[
                   C.layers.For(range(3)): lambda :[
                        C.layers.Convolution((3,3), 256, pad=True)
                    ]),
                    C.layers.MaxPooling((2,2), strides=(2,2)),
                    C.layers.Dropout(0.25)
                ]),
                C.layers.For(range(2), lambda : [
                    C.layers.Dense(1024),
                    C.layers.Dropout(0.5)
                ]), C.layers.Dense(num_classes, None)])
```



Four Loss Functions

- Cross-entropy loss
  - Prediction matches label distribution
- Majority voting
  - Choose the majority label + cross-entropy loss
- Multi-label learning
  - Prediction is correct as long as the model predicts one of the labels voted more than n times
- Probabilistic label drawing
  - Randomly draw a label as temporary GT + cross-entropy loss

Custom Loss in CNTK

- For majority voting, probabilistic label drawing and cross entropy

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{k=1}^{8} p_k^i \log q_k^i.$$

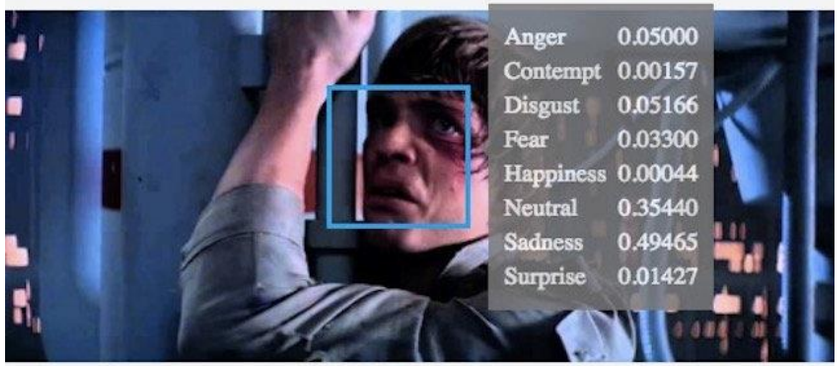
```
train_loss = -C.reduce_sum(C.element_times(target, C.log(prediction)))
```

- Multi-label learning

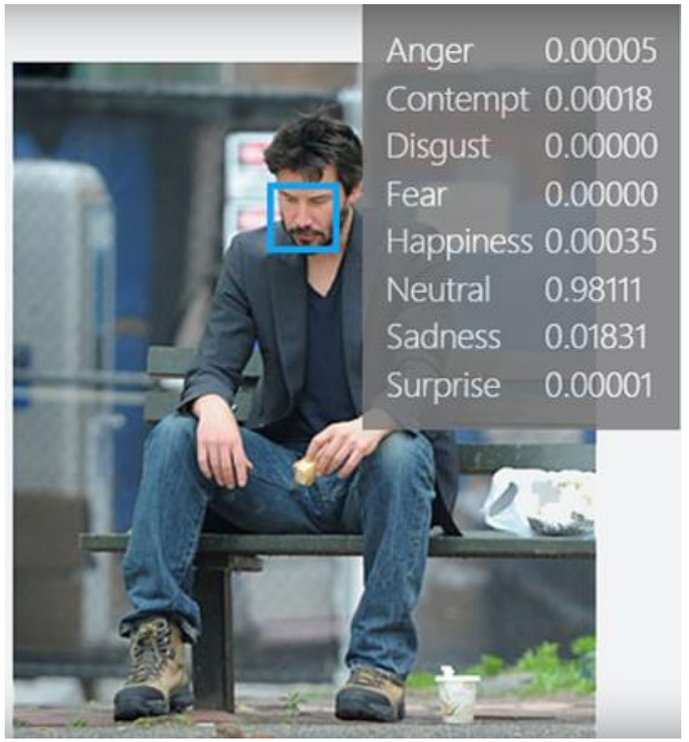
$$\mathcal{L} = -\sum_{i=1}^{N} \arg\max_k [I_\theta(p_k^i) \log q_k^i]$$

```
train_loss = -C.reduce_max(C.element_times(target, C.log(prediction)))
```

Some Examples



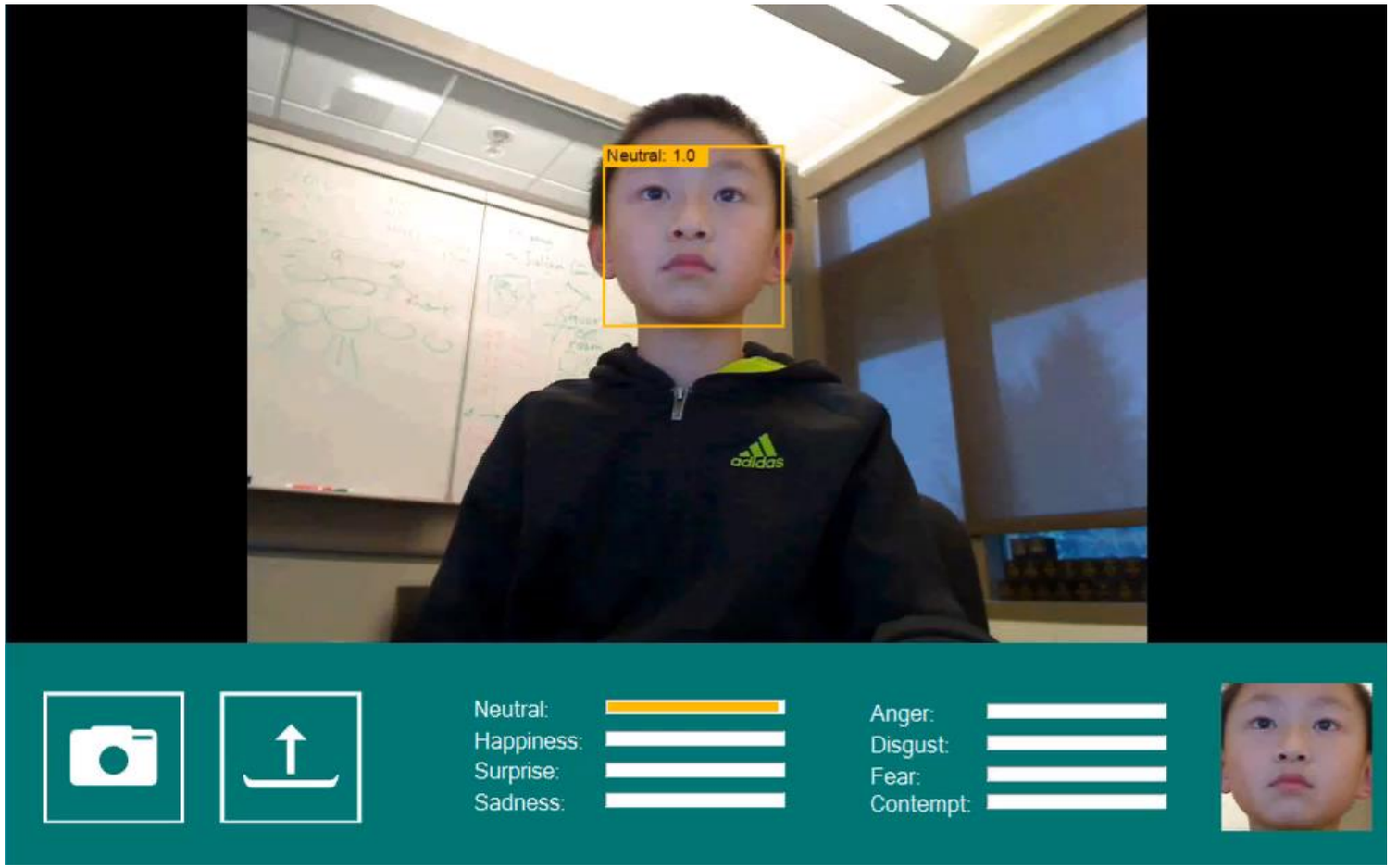
More Examples



“Microsoft thinks Sad Keanu is only 0.01831 sad”

“According to Microsoft's Emotion API, Sidney Crosby was rather angry about scoring the goal that won Canada a gold medal at 2010's Olympics in Vancouver.”

Emotion from Video



Faster R-CNN in CNTK

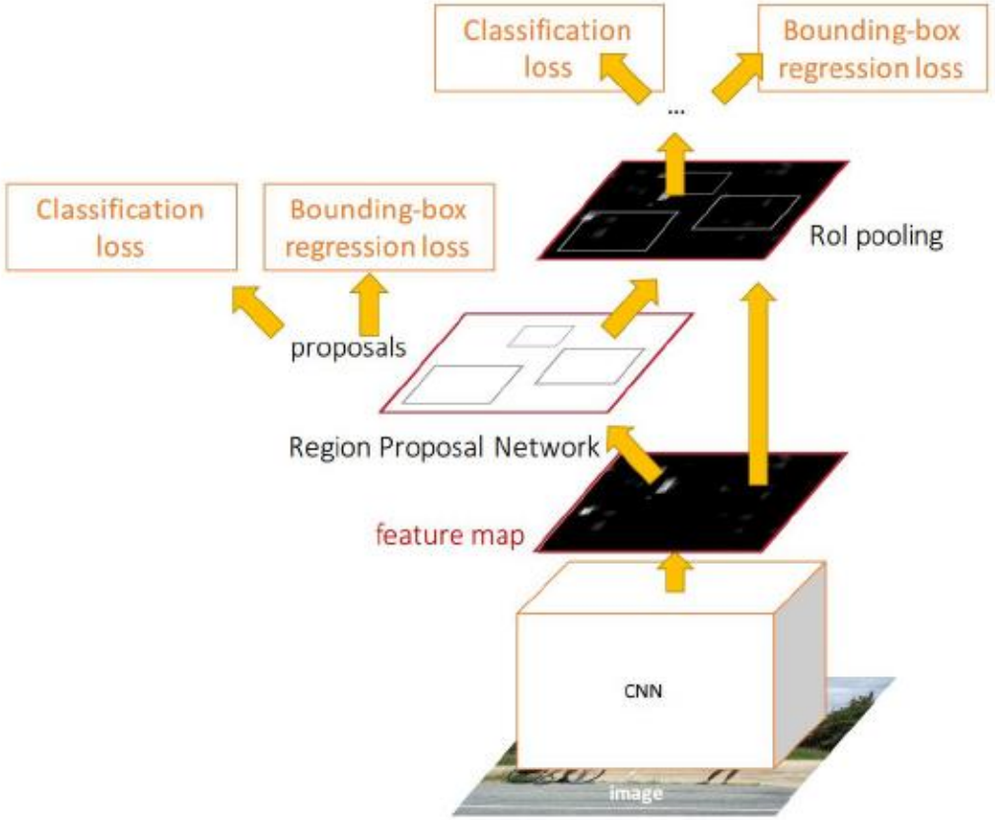
Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun
"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks",
NIPS 2015

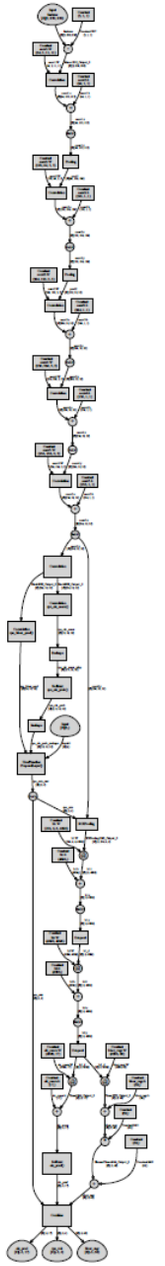
Faster R-CNN in CNTK

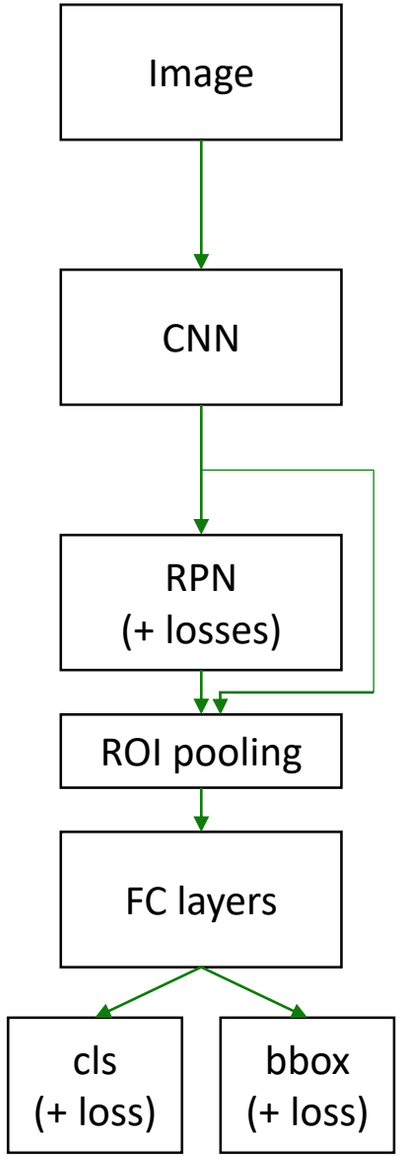
Take away

- Show use of pretrained model
- Show how to concatenate multiple networks
- Show model cloning

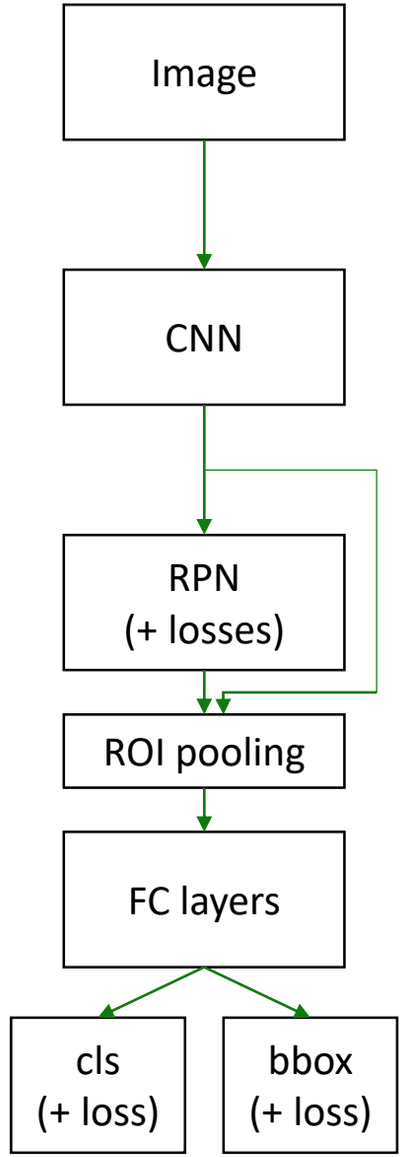
Faster R-CNN in CNTK



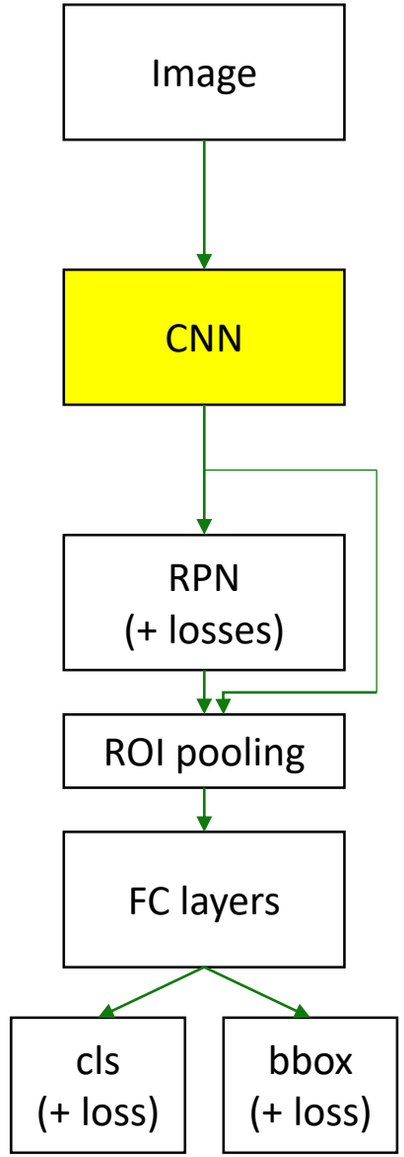




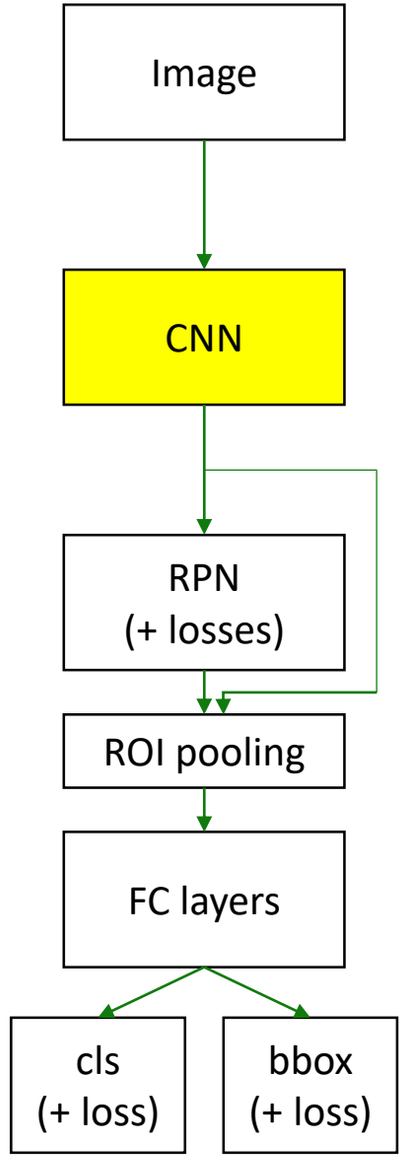
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model  = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out   = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers   = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out    = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



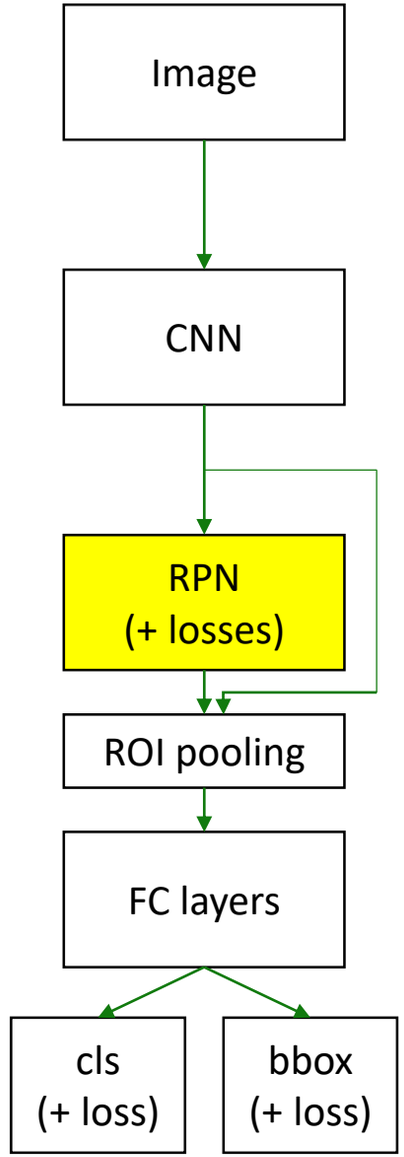
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model  = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out   = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers   = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out    = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



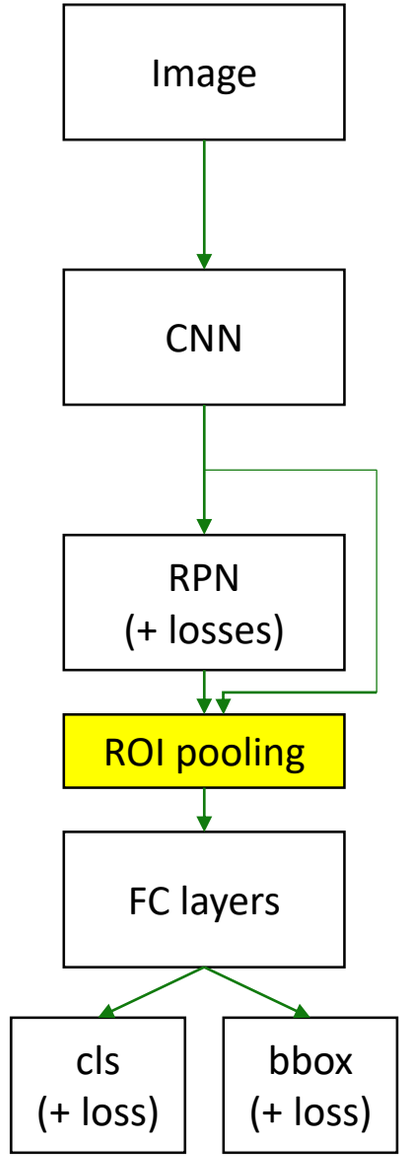
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model   = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out    = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers    = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out     = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



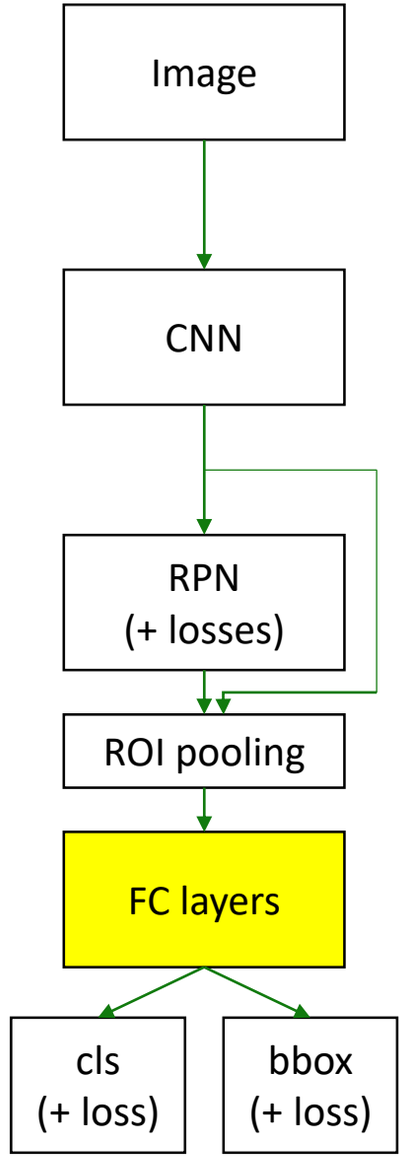
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model   = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out    = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers    = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out     = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



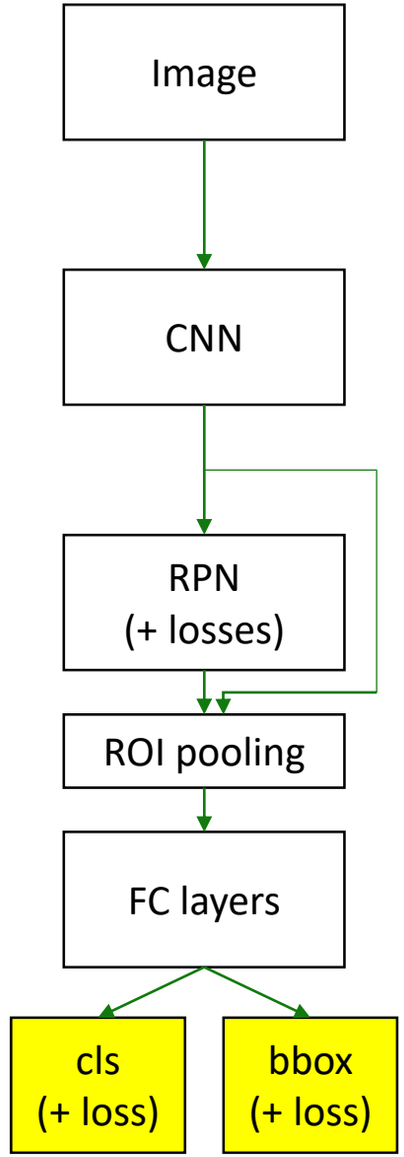
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model   = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out   = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers   = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out     = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



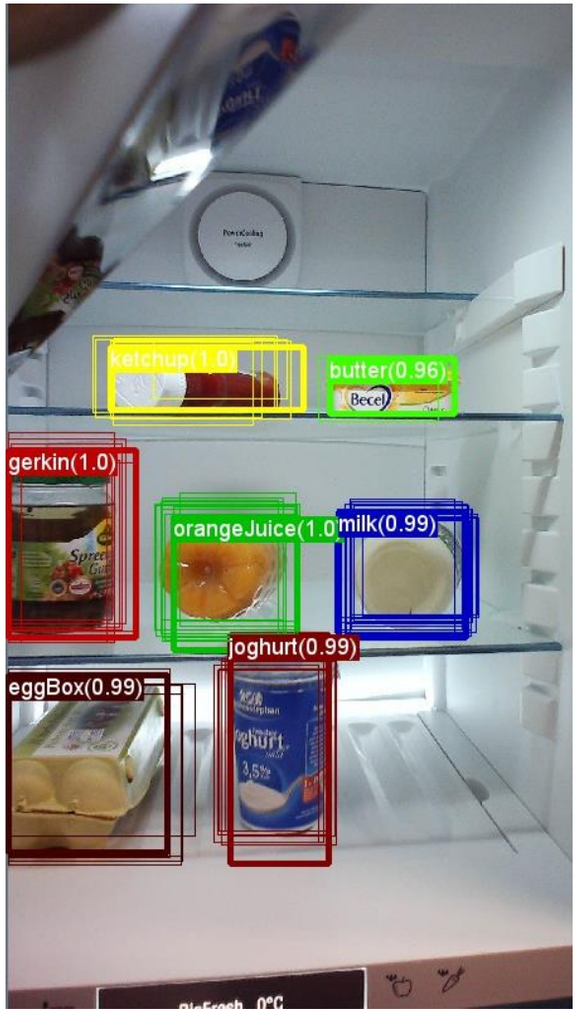
```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model   = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out   = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers   = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out    = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```

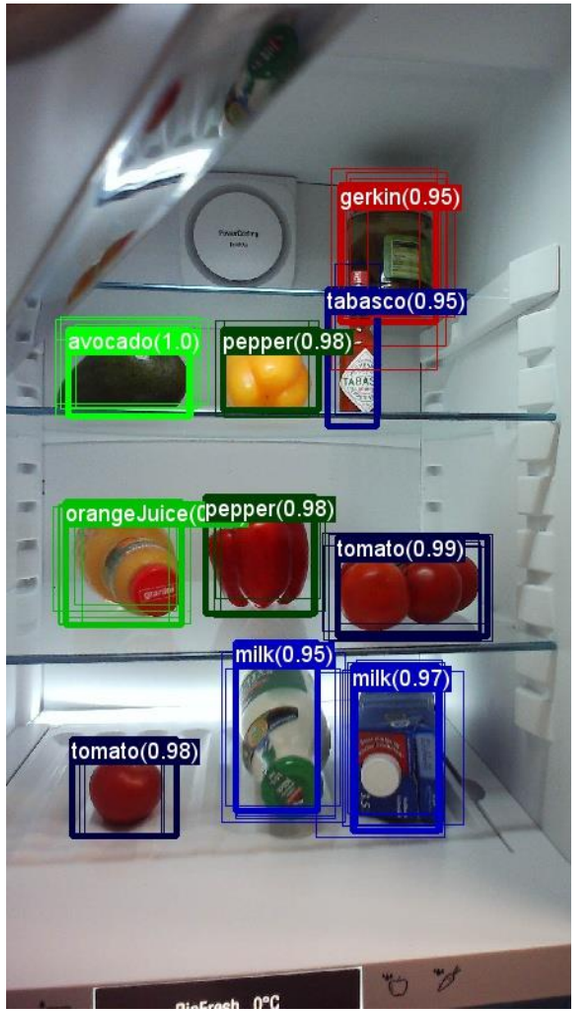


```
def create_faster_rcnn_predictor(input_var, gt_boxes, img_dims):

    base_model   = load_model(base_model_file)
    conv_layers = clone_model(base_model, ['data'], ['relu5_3'], CloneMethod.freeze)

    conv_out = conv_layers(input_var)

    rpn_rois, rpn_losses = create_rpn(conv_out, gt_boxes, img_dims)

    roi_out    = roipooling(conv_out, rpn_rois, cntk.MAX_POOLING, (roi_dim, roi_dim))

    fc_layers    = clone_model(base_model, ['pool5'], ['drop7'], CloneMethod.clone)
    fc_out     = fc_layers(roi_out)

    cls_score = Dense(shape=(4096, num_classes), None)(fc_out)
    bbox_pred = Dense(shape=(4096, num_classes*4), None)(fc_out)

    det_losses = create_detector_losses(cls_score, bbox_pred, rpn_rois, gt_boxes)
    loss = rpn_losses + det_losses
    pred_error = classification_error(cls_score, label_targets, axis=1)

    return loss, pred_error
```



Faster R-CNN in CNTK: Example





Grocery

- small data set in CNTK repo for playing
- 17 classes (food items)
- 20 train, 5 test images
- 96.8% mAP using VGG16

Artistic Neural Style

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge
“A Neural Algorithm of Artistic Style”, CoRR 2015

Roman Novak, Yaroslav Nikulin
“Improving the Neural Algorithm of Artistic Style”, CoRR 2016

https://github.com/Microsoft/CNTK/blob/master/Tutorials/CNTK_205_Artistic_Style_Transfer.ipynb

Artistic Neural Style

Take away

- Show how to implement a complex loss function.

Artistic Neural Style

- Given two images
  - A input image in which you want to preserve its contents.
  - A style image.

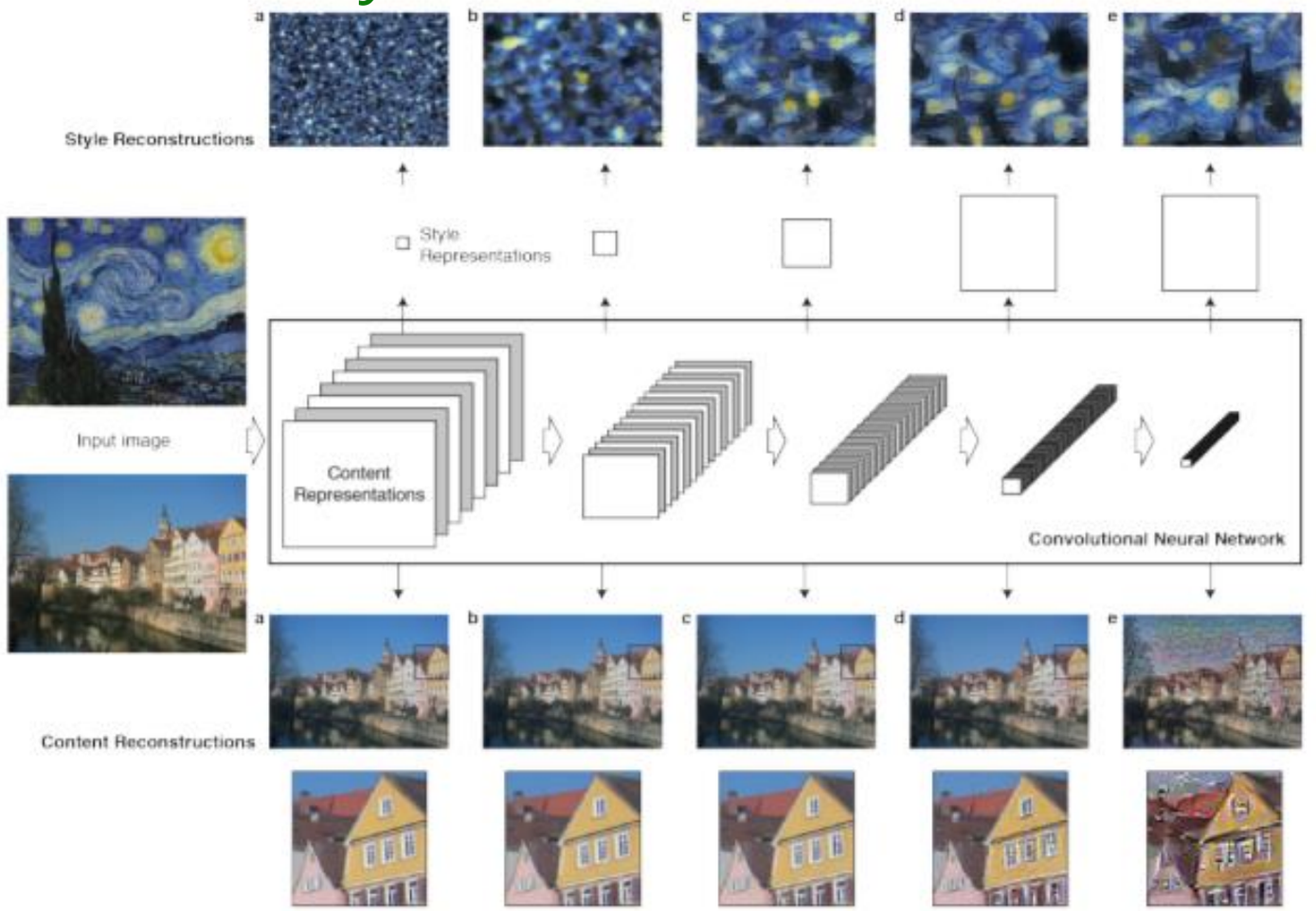
- Goal: change the style of the input image to match the style image, while keeping its content intact.


+

=


https://en.wikipedia.org/wiki/The_Starry_Night

Artistic Neural Style



Artistic Neural Style

Loss: $L(x) = \alpha C(x, c) + \beta S(x, s) + T(x)$

- **Content loss** (C): Match the generated image with the content image. Main idea: keep changing the generated image until its feature match the content image.

```
def content_loss(x, c):
    return C.squared_error(x, c)/np.prod(list(x.shape))
```

Artistic Neural Style

Loss: $L(x) = \alpha C(x, c) + \beta S(x, s) + T(x)$

- **Style loss** (S): Match the correlation between feature of the generated image to the correlation between feature of the style image.

```
def gram(x):
    features = C.minus(flatten(x), C.reduce_mean(x))
    return C.times_transpose(features, features)

def style_loss(x, s):
    X = gram(x)
    S = gram(s)
    return C.squared_error(X, S)/(x.shape[0]**2 * x.shape[1]**4)
```

Artistic Neural Style

Loss: $L(x) = \alpha C(x, c) + \beta S(x, s) + T(x)$

- **Total variation loss** (T): Measure the smoothness of the image, reduce $T(x)$ smooth the generated image.

```
def total_variation_loss(x):
    xx = C.reshape(x, (1,)+x.shape)
    delta = np.array([-1, 1], dtype=np.float32)
    kh = C.constant(value=delta.reshape(1, 1, 1, 1, 2))
    kv = C.constant(value=delta.reshape(1, 1, 1, 2, 1))
    dh = C.convolution(kh, xx)
    dv = C.convolution(kv, xx)
    avg = 0.5 * (C.reduce_mean(C.square(dv)) + C.reduce_mean(C.square(dh)))
    return avg
```

Generative Adversarial Networks (GAN)

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
"Generative Adversarial Networks", NIPS 2014

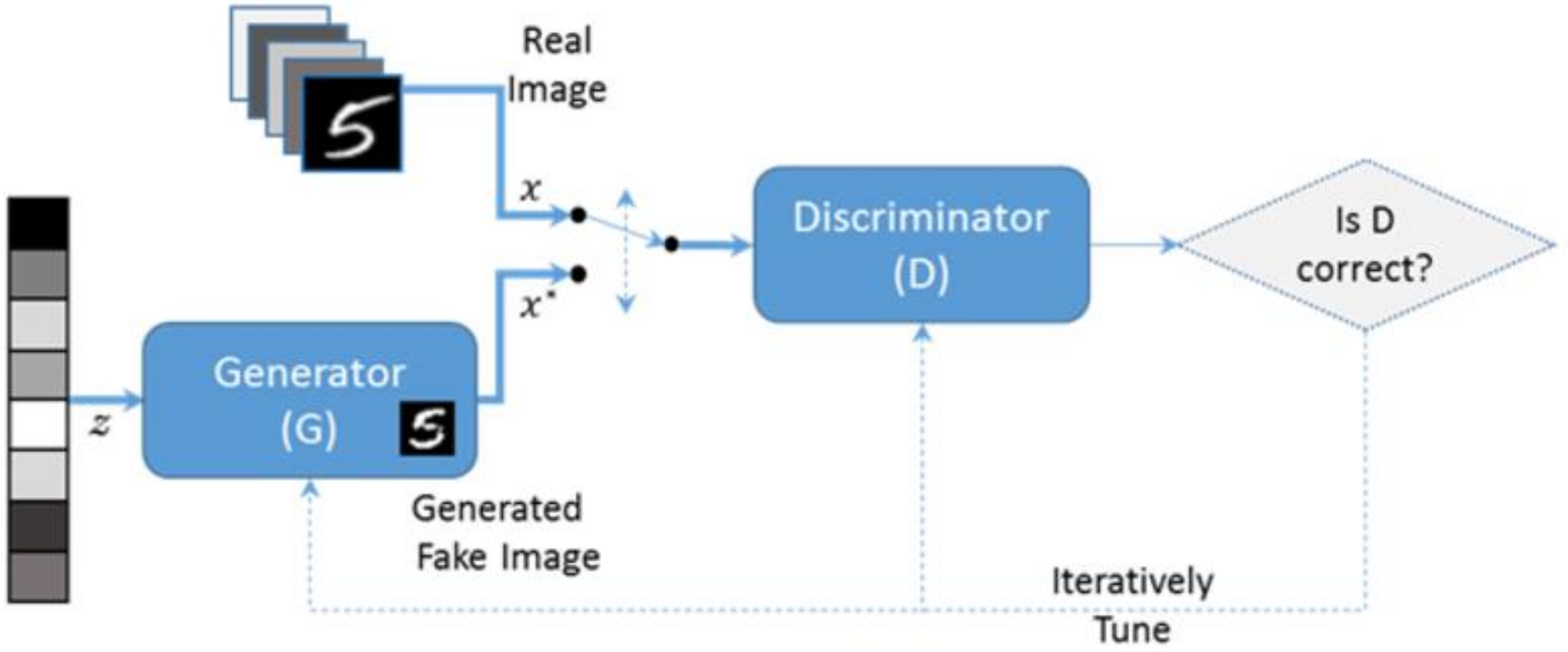
https://github.com/Microsoft/CNTK/blob/master/Tutorials/CNTK_206A_Basic_GAN.ipynb
https://github.com/Microsoft/CNTK/blob/master/Tutorials/CNTK_206B_DCGAN.ipynb

Generative Adversarial Networks (GAN)

Take away

- Show how to update specific parameters in the network.
- Show how to implement a non-standard learning algorithm

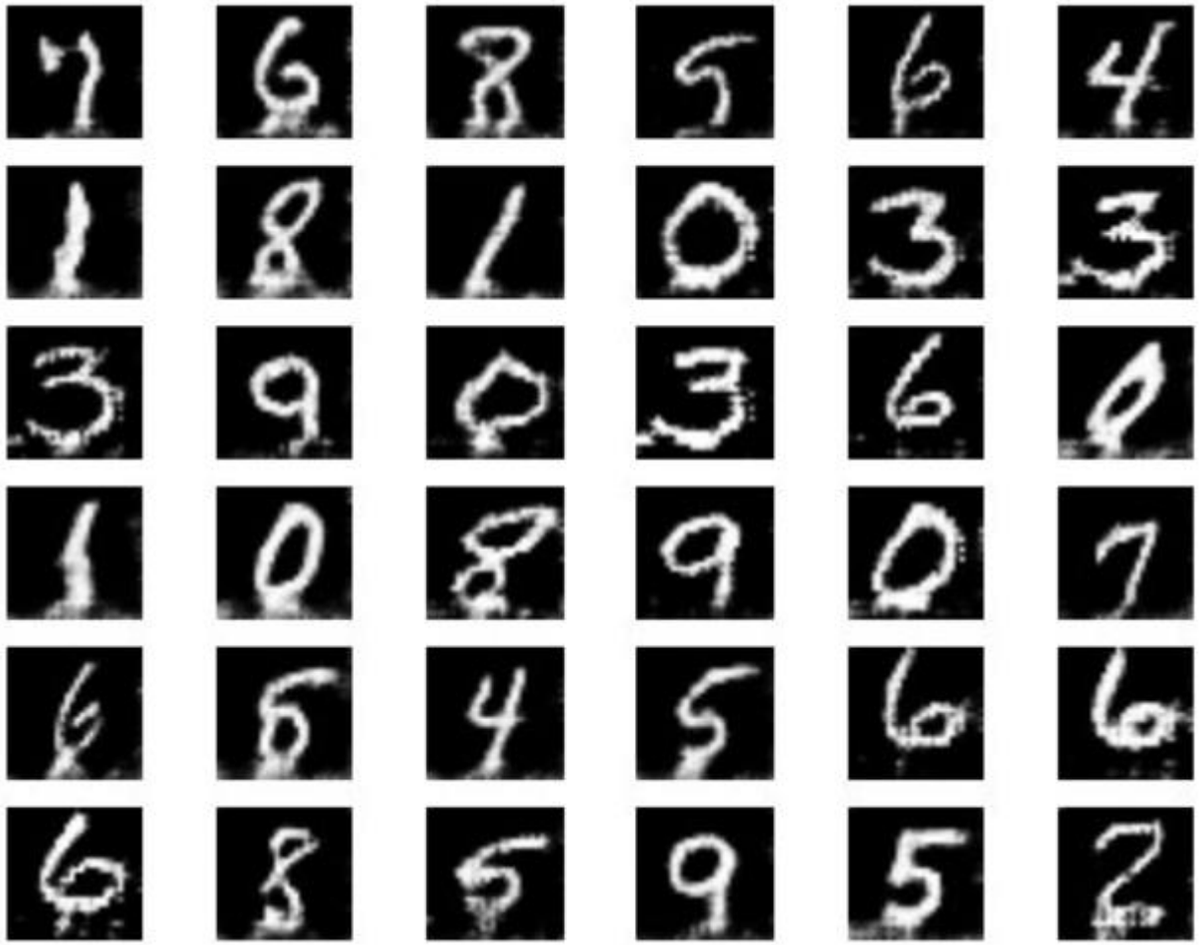
GAN


Real
Image
x
x*
Discriminator
(D)
Is D
correct?
z
Generator
(G)
Generated
Fake Image
Iteratively
Tune

GAN in CNTK

```
g = generator(z)
d_real = discriminator(real_input)
d_fake = d_real.clone(method='share', substitutions={real_input.output:g.output})

g_loss = 1.0 - C.log(d_fake)
d_loss = -(C.log(d_real) + C.log(1.0 - d_fake))

g_learner = C.adam(parameters=g.parameters,
                   lr=C.learning_rate_schedule(lr, C.UnitType.sample),
                   momentum=C.momentum_schedule(momentum))

d_learner = C.adam(parameters=d_real.parameters,
                   lr=C.learning_rate_schedule(lr, C.UnitType.sample),
                   momentum=C.momentum_schedule(momentum))

g_trainer = C.Trainer(g, (g_loss, None), g_learner)
d_trainer = C.Trainer(d_real, (d_loss, None), d_learner)
```

MNIST GAN with CNTK



PixelCNN++

Tim Salimans, Andrej Karpathy, Xi Chen, Diederik P. Kingma
“PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications”, ICLR 2017

https://github.com/openai/pixel-cnn

PixelCNN++

Take away

- Show how to use low level CNTK APIs (for advanced users)
- Show CNTK usage akin to NumPy usage
- Provide a high level view of a probabilistic generative model

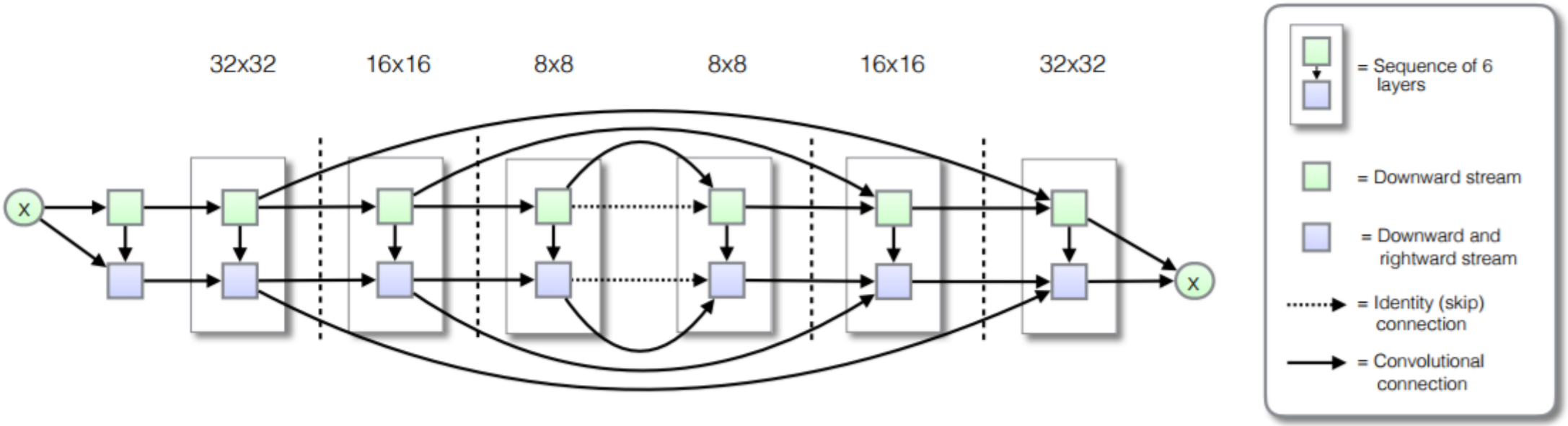
PixelCNN++

- Model each pixel as a mixture of logistic probability.

$$P(x|\pi,\mu,s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x+0.5-\mu_i)/s_i) - \sigma((x-0.5-\mu_i)/s_i)\right]$$

- Learn a conditional probability of each pixel channel given previous pixels.
- Can generate full image by sample pixels from the learned probability distribution.
- Can generate images for specific label, when trained condition on the label.

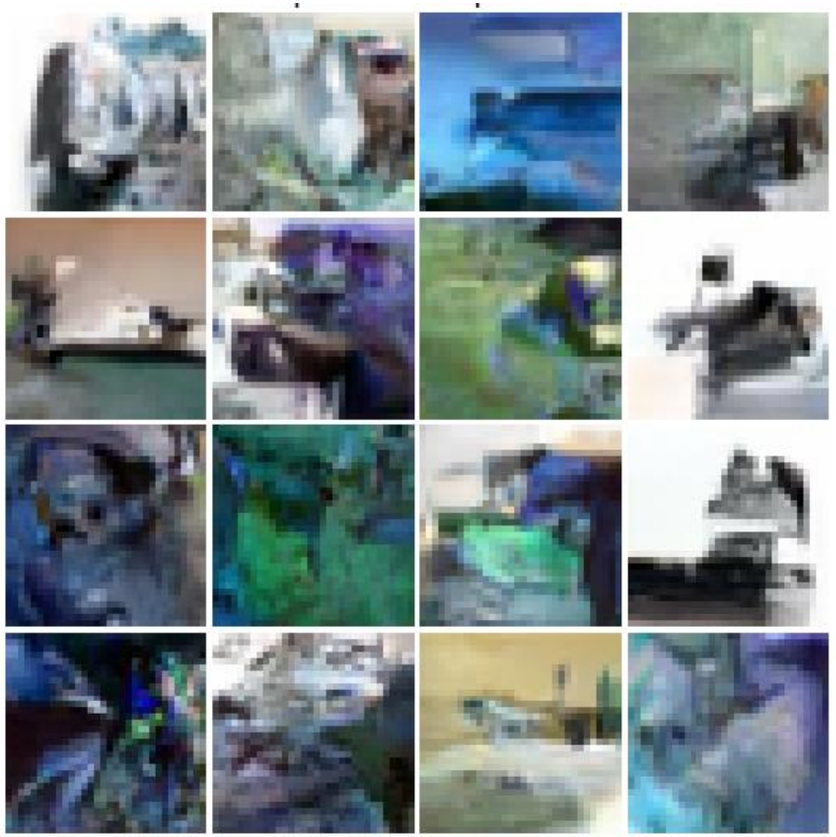
PixelCNN++


32x32
16x16
8x8
8x8
16x16
32x32
x
x
= Sequence of 6 layers
= Downward stream
= Downward and rightward stream
= Identity (skip) connection
= Convolutional connection

PixelCNN++: Using Low Level APIs

```
def gated_resnet(x, a=None, h=None, nonlinearity=concat_elu, conv=conv2d,…):
    …
    c1 = conv(nonlinearity(x), num_filters)
    if a is not None: # add short-cut connection if auxiliary input 'a' is given
        c1 += nin(nonlinearity(a), num_filters)
    c1 = nonlinearity(c1)
    if dropout_p > 0:
        c1 = C.dropout(c1, dropout_p)
    c2 = conv(c1, num_filters * 2, init_scale=0.1)

    if h is not None:
       Wh = C.parameter(h.shape + (2 * num_filters,), init=init, name='Wh')
       c2 = c2 + C.reshape(C.times(h, Wc), (2 * num_filters, 1, 1))
    a = c2[:num_filters,:,:]
    b = c2[num_filters:2*num_filters,:,:]
    c3 = a * C.sigmoid(b)
    return x + c3
```

PixelCNN++ Samples



PixelCNN++ Samples

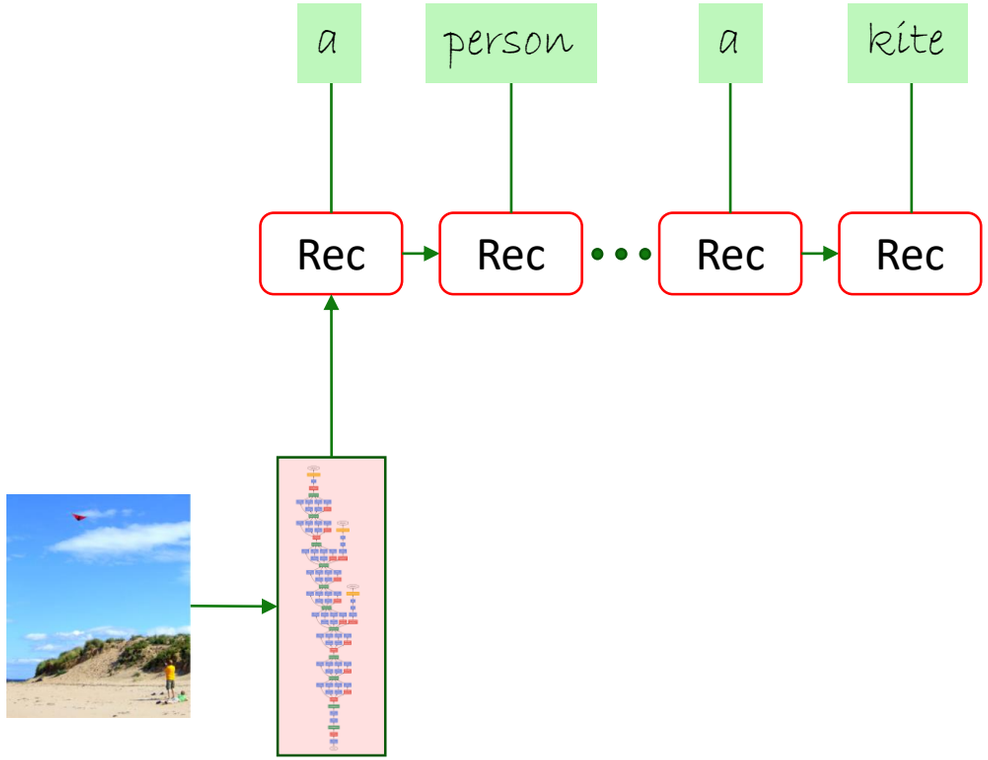
Image Captioning

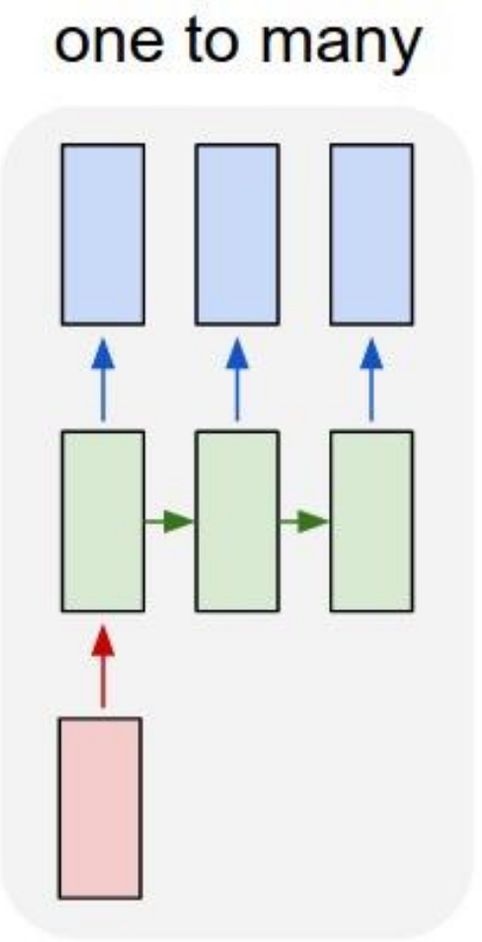
Image Captioning

Take away

- Show how one reduces the burden (on developer) to handle variable length sequences
- Show how to use advanced block functions in layers library

Image Captioning (one to many)

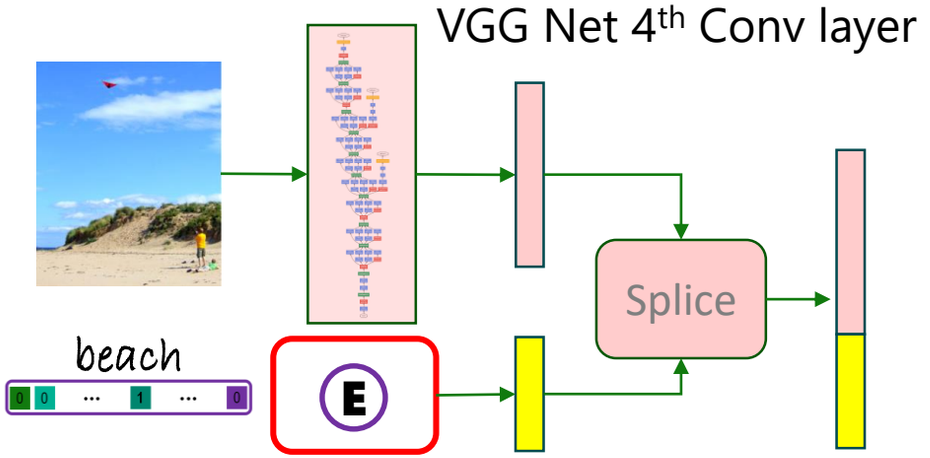




http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Image Captioning with CNTK

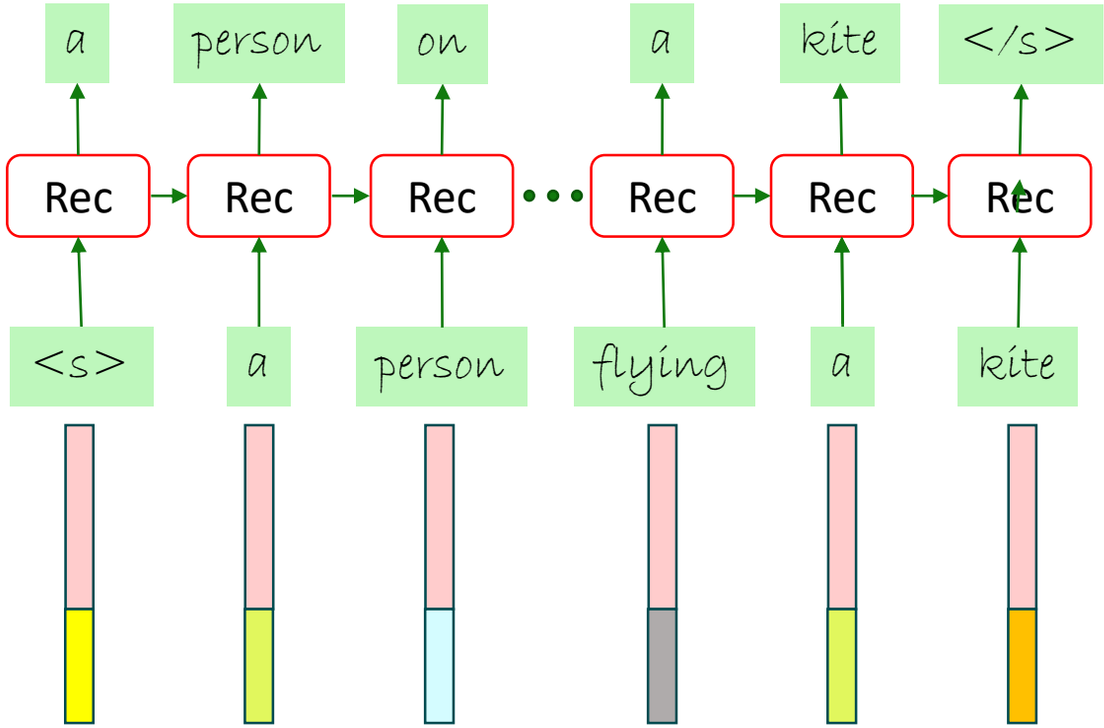
- Input:
  - Image features
  - Word token in the caption




```
# Input image feature
img_fea = VGG_model(img)

# Caption input
cap_in = C.input_variable(shape=(V), is_sparse=True)


img_txt_feature = C.splice(C.reshape(img_fea, …),
                           C.Embedding(EMB_DIM)(cap_in))
```

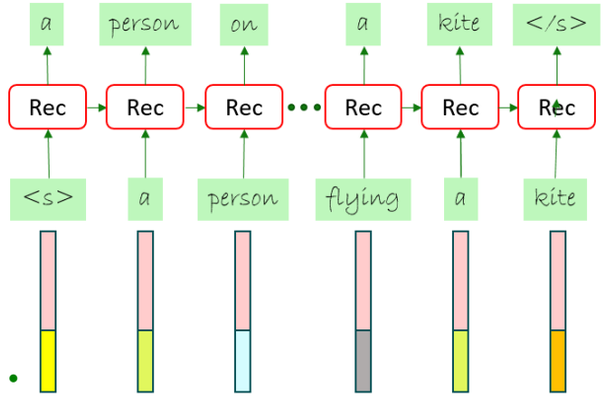
Image Captioning with CNTK

- Use Sequence to Sequence generation machinery
  - Input: Image Feature + word
  - Output: next word




```
img_fea_broadcasted = C.splice(
        C.sequence.broadcast_as(img_fea),
        C.Embedding(EMB_DIM)(cap_in))
```

Image Captioning (Decoder)




```
def eval_greedy(input): # (input*) --> (word sequence*)

    # Decoding is an unfold() operation starting from sentence_start.
    # We must transform (history*, input* -> word_logp*) into
    # a generator (history* -> output*)
    # which holds 'input' in its closure.
    unfold = C.layers.UnfoldFrom(lambda history: model(history, input) >> C.hardmax,
                                 # stop once sentence_end_index is reached
                                 until_predicate=lambda w: w[...,sentence_end_index])

    return unfold(initial_state=sentence_start, dynamic_axes_like=input)
```


https://github.com/Microsoft/CNTK/blob/master/Tutorials/CNTK_204_Sequence_To_Sequence.ipynb

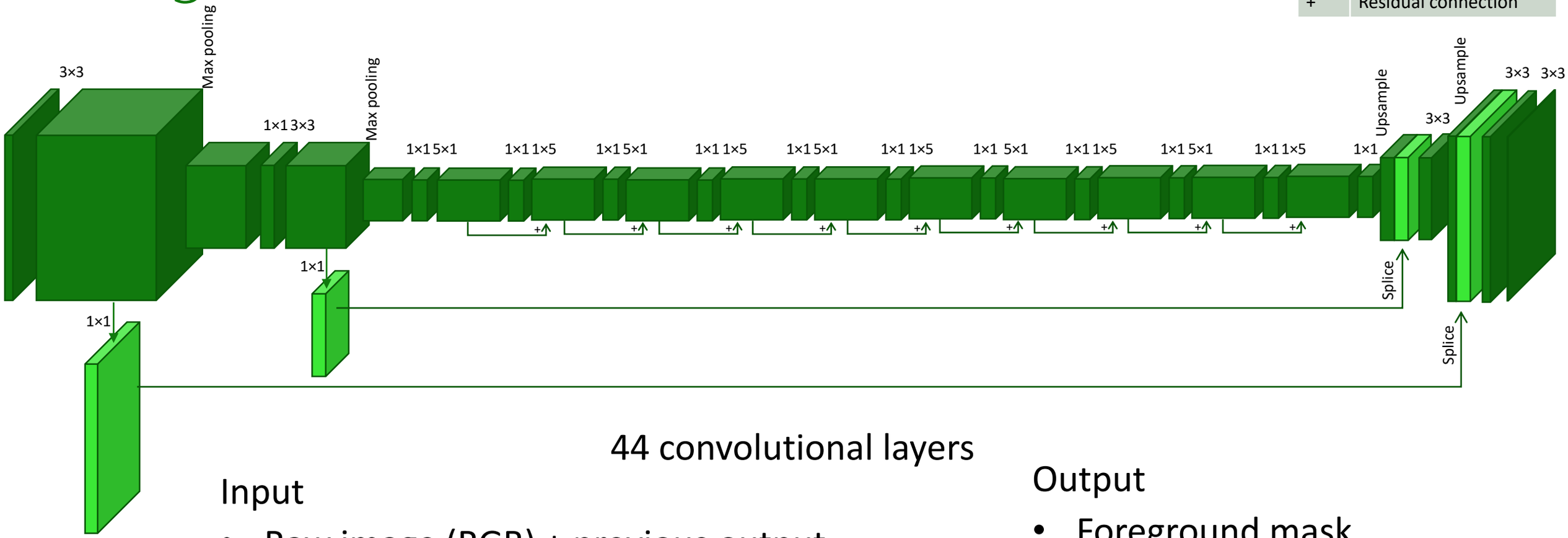
Image Segmentation

Take away

- Show how to segment images with CNTK

Segmentation Model

1×1	1×1 convolution
3×3	3×3 convolution
1×5	1×5 convolution
5×1	5×1 convolution
+	Residual connection



44 convolutional layers

Input

- Raw image (RGB) + previous output
- 96×128×4

Output

- Foreground mask
- 96×128×1

Segmentation Model

```
def resnet_skip_block(input, d1, d2):
    with C.layers.default_options(activation=None, pad=True, init=C.glorot_uniform()):
        z = C.layers.Sequential([
                C.layers.For(range(2), lambda i: [
                    C.layers.Convolution([(1,1), (3,3)][i], d1),
                    C.layers.BatchNormalization(map_rank=1),
                    C.layers.Activation('relu')
                ]),
                C.layers.Convolution((1,1), d2),
                C.layers.BatchNormalization(map_rank=1)
            ])(input)
    return C.relu(z + input)
```

Image Segmentation Demo


Video Networks:
3D Convolution
Late Fusion
Pretraining + LSTM

Action Classification

Two main problems:

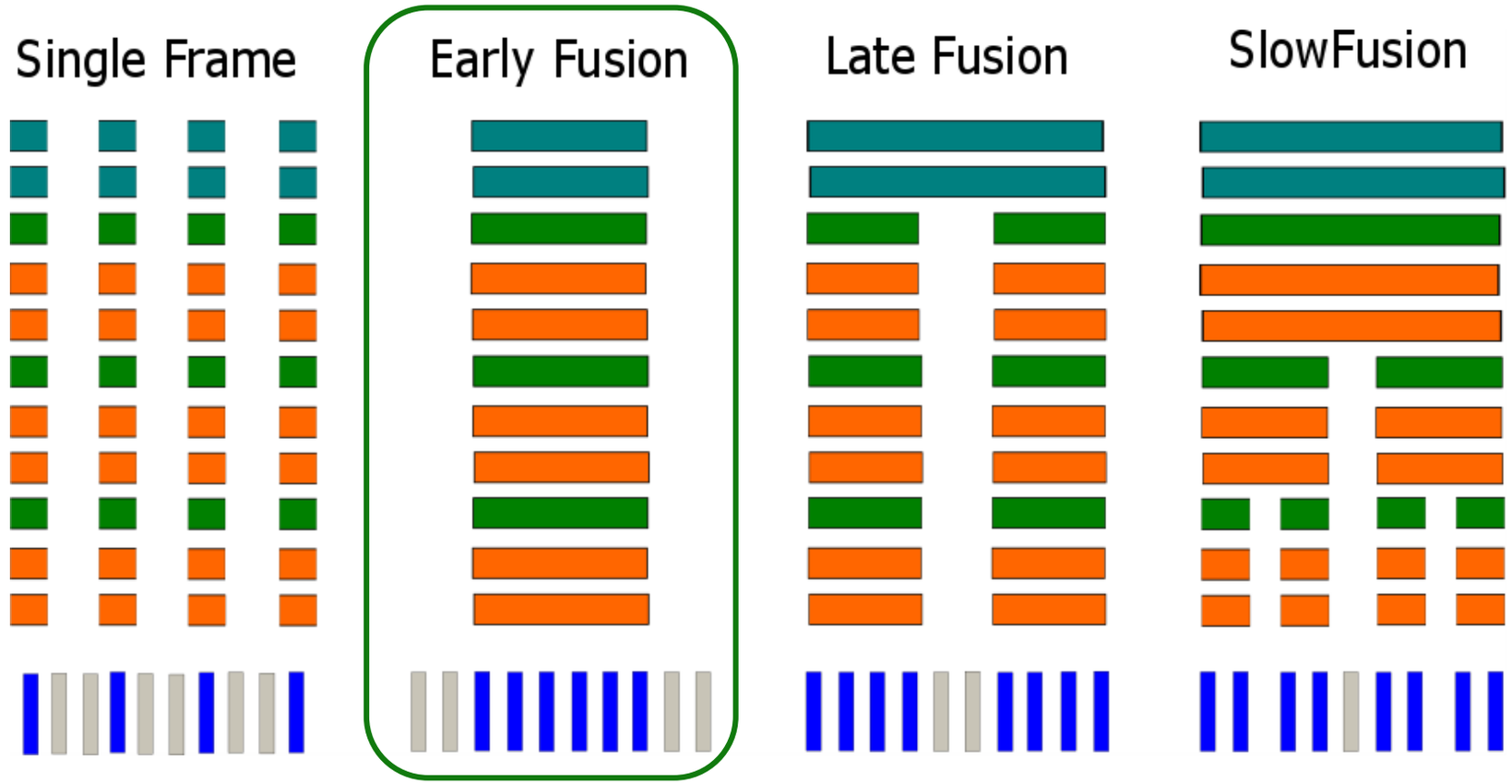
- Action classification
  - Input: a trimmed video clip with a single action.
  - Output: classify the action in the clip.
- Action detection
  - Input: an untrimmed video clip with multiple actions and possible no action.
  - Output: location of each action and its corresponding classification.

Action Classification

Two possible approaches:

- 3D Convolution network
  - Extend 2D convolution into temporal axis.
  - Pick at random a sequence of frames from the video clip.
  - Use the 3D cube as input to the 3D convolution network.
- Pretraining + RNN
  - Use a pretrained model.
  - Extract a feature vector from each frame.
  - Pass the sequence of features into a recurrent network.

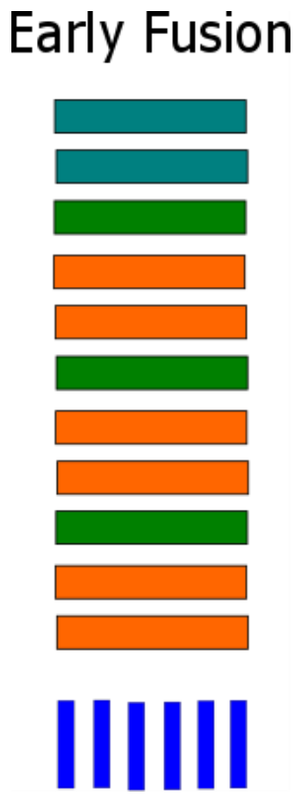
Video Classification Using Feedforward Networks



[Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014]

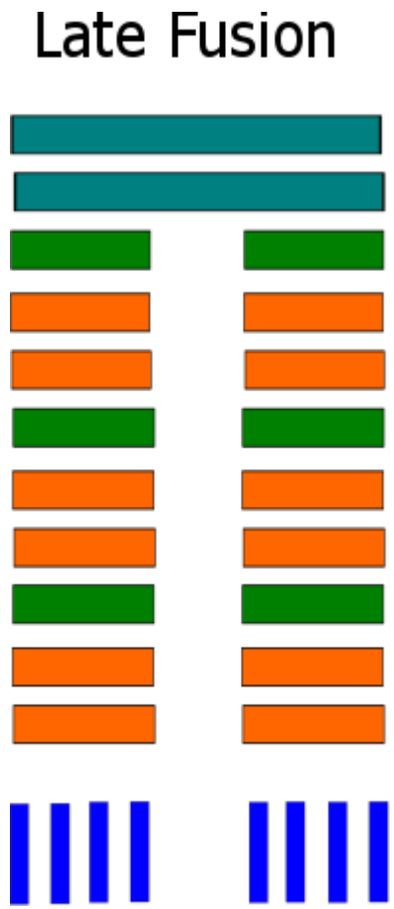
3D Convolution Network

```
input_var = C.input_variable((num_channels, sequence_length, image_height, image_width))

with C.default_options (activation=C.relu):
    z = C.layers.Sequential([
            C.layers.Convolution3D((3,3,3), 64),
            C.layers.MaxPooling((1,2,2), (1,2,2)),
            C.layers.For(range(3), lambda i: [
                C.layers.Convolution3D((3,3,3), [96, 128, 128][i]),
                C.layers.Convolution3D((3,3,3), [96, 128, 128][i]),
                C.layers.MaxPooling((2,2,2), (2,2,2))
            ]),
            C.layers.For(range(2), lambda : [
                C.layers.Dense(1024),
                C.layers.Dropout(0.5)
            ]),
            C.layers.Dense(num_output_classes, activation=None)
        ])(input_var)
```



Late Fusion

```
with C.default_options (activation=C.relu):
        z1 = C.layers.Sequential([
                C.layers.Convolution3D((3,3,3), 64),
                C.layers.MaxPooling((1,2,2), (1,2,2)),
                C.layers.For(range(3), lambda i: [
                    C.layers.Convolution3D((3,3,3), [96, 128, 128][i]),
                    C.layers.Convolution3D((3,3,3), [96, 128, 128][i]),
                    C.layers.MaxPooling((2,2,2), (2,2,2))
                ]),
            ])(input_var1)
        z2 = C.layers.Sequential([…])(input_var2)
        z  = C.layers.Sequential([
                C.layers.For(range(2), lambda : [
                    C.layers.Dense(1024),
                    C.layers.Dropout(0.5)
                ]),
                C.layers.Dense(num_output_classes, None)
            ])(C.splice(z1, z2, axis=0))
```



Pretraining + RNN

- Loading a pretrained model.
- Extract feature from each frame in a video.
- Feed those frames to LSTM.
- Classify the last output of the LSTM.

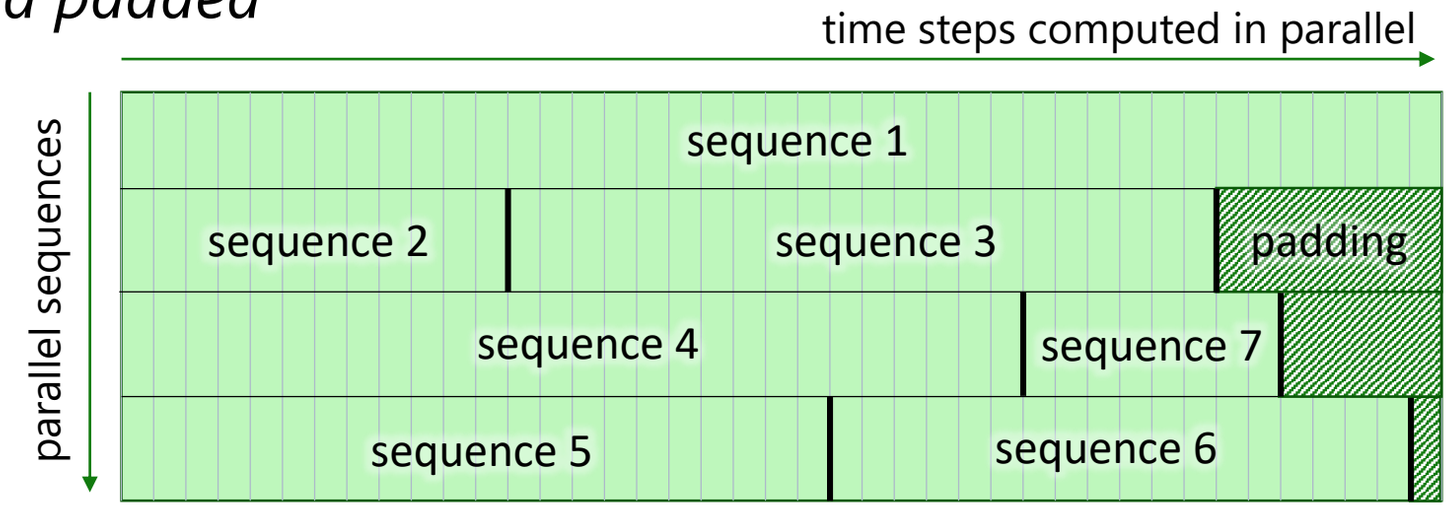
Loading Model and Extract Feature

- Download a pretrained model from CNTK site.
- Convert a pretrained model from another toolkit such as Caffe.
- Train you own network from scratch.
- Loading a model and extract feature is trivial, as shown below:

```
loaded_model  = C.load_model(model_file)
node_in_graph = loaded_model.find_by_name(node_name)
output_node   = C.as_composite(node_in_graph)

output = output_node.eval(<image>)
```

Variable-Length Sequences in CNTK

- Minibatches containing sequences of different lengths are automatically packed *and padded*



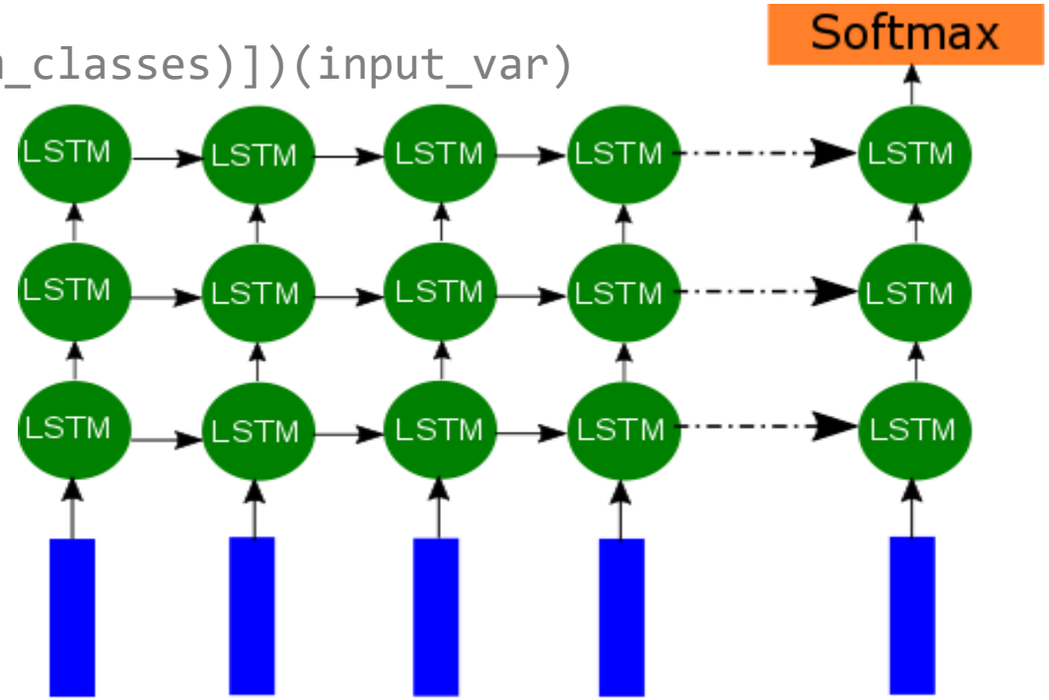
- Fully transparent batching
- Recurrent → CNTK unrolls, handles sequence boundaries
- Non-recurrent operations → parallel
- Sequence reductions → mask

Feature sequence classification

- Use cntk.sequence.input instead of cntk.input.
- Define the network for a single sample.
- No explicit handling of batch or sequence axes.
- Use cntk.sequence.last to get the last item in the sequence.
- Use cntk.sequence.first to get the first item in the sequence, in case of bidirection.

Feature Sequence Classification

```
input_var = C.sequence.input_variable(shape=input_dim)
label_var = C.input_variable(num_classes)

z = C.layers.Sequential([C.For(range(3), lambda : Recurrence(LSTM(hidden_dim))),
                         C.sequence.last,
                         C.layers.Dense(num_classes)])(input_var)
```



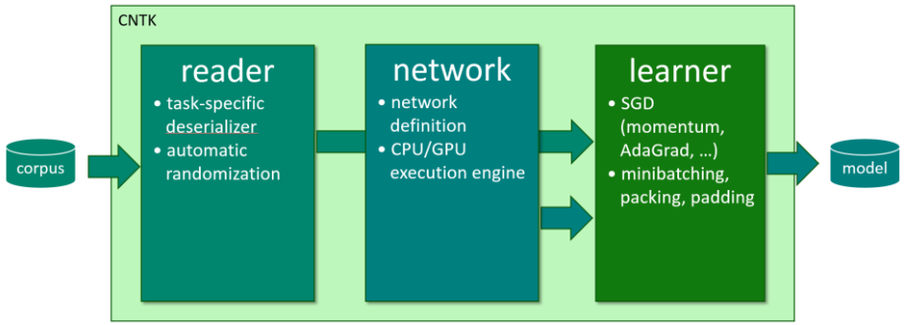
Feature Sequence Classification (Bi-Directional)

```
input_var = C.sequence.input_variable(shape=input_dim)
label_var = C.input_variable(num_classes)

fwd = C.layers.Sequential(
          [C.For(range(3), lambda : Recurrence(GRU(hidden_dim))),
           C.sequence.last])(input_var)

bwd = C.layers.Sequential(
          [C.For(range(3), lambda : Recurrence(GRU(hidden_dim), go_backwards=True)),
           C.sequence.first])(input_var)

z = C.layers.Dense(num_classes)(C.splice(fwd, bwd))
```

Parallel Training:

- Enable Parallel Training
- 1-bit SGD and Block Momentum
- Fast ResNet50 and Inception V3 Training

Distributed training

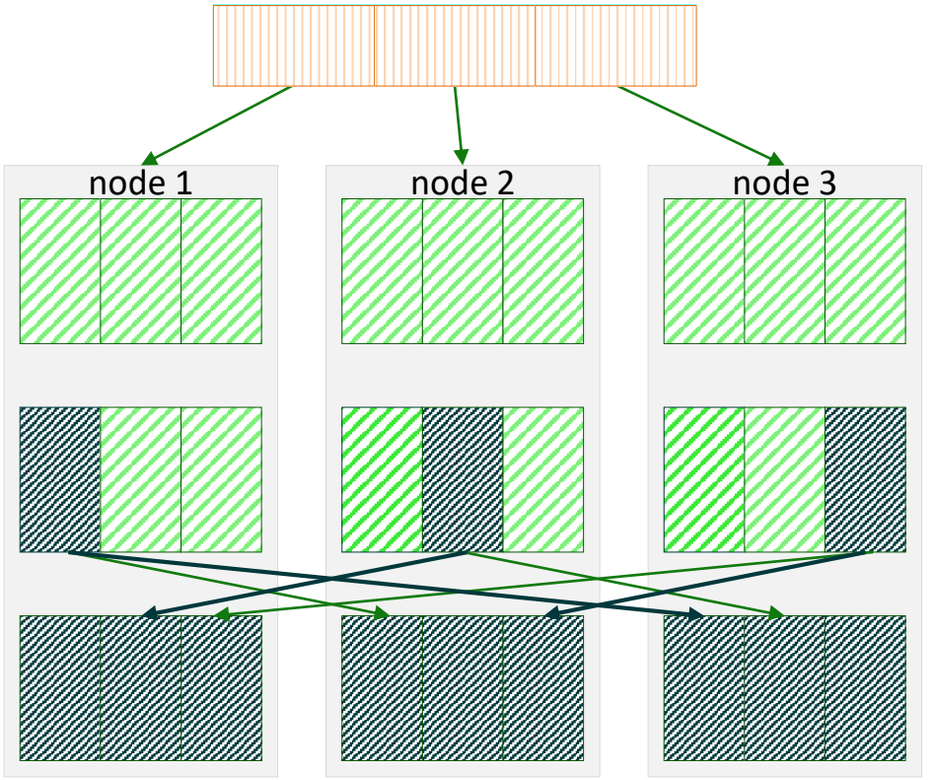


- Prepare data
- Configure reader, network, learner (Python)
- Train: -- distributed*!*

```
mpiexec --np 16 --hosts server1,server2,server3,server4    \
python my_cntk_script.py
```

Data-Parallel Training

- Data-parallelism: distribute minibatch over workers, all-reduce partial gradients



minibatch (sample frames)

each node computes sub-mini-batch sub-gradient of one layer

step 1: each node owns aggregating a stripe

step 2: aggregated stripes are redistributed

ring algorithm
$O(2\ (K\text{-}1)/K\ M)$
→ $O(1)$ w.r.t. K

Data-Parallel Training

How to reduce communication cost:

communicate less each time

- 1-bit SGD: [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "1-Bit Stochastic Gradient Descent...Distributed Training of Speech DNNs", Interspeech 2014]
  - quantize gradients to 1 bit per value
  - trick: carry over quantization error to next minibatch

communicate less often

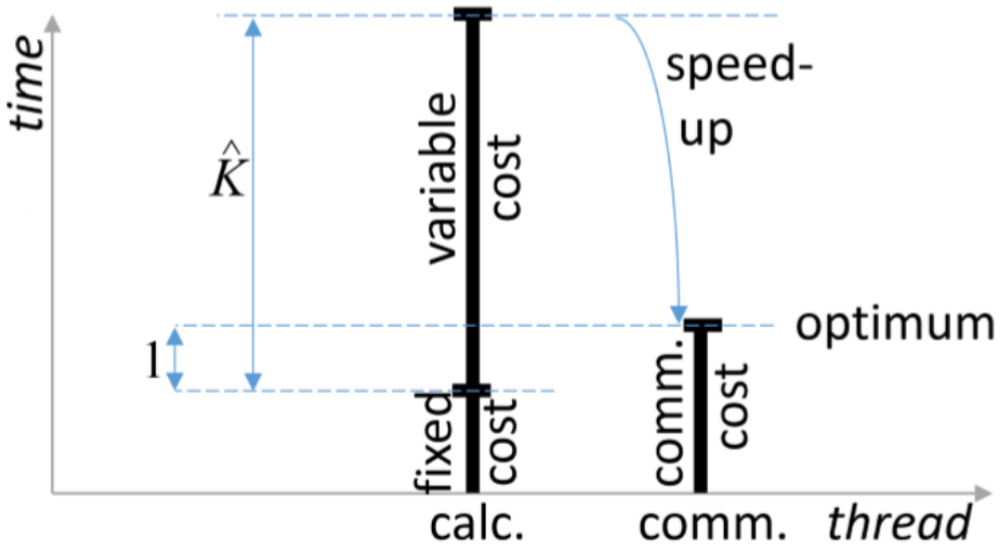
- Automatic MB sizing [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "ON Parallelizability of Stochastic Gradient Descent...", ICASSP 2014]
- Block momentum [K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training...," ICASSP 2016]
  - Very recent, very effective parallelization method
  - Combines model averaging with error-residual idea

Calculation and Communication Cost Analysis

- Calculation cost
  - Variable cost
    - Gradient computation (scalable by # of nodes)
  - Fixed cost
    - Gradient post processing (momentum, AdaGrad accumulation, etc.)
    - Add gradient to model
- Communication cost

Goal:

- Reduce #bits exchanged between servers



Data-Parallel Training

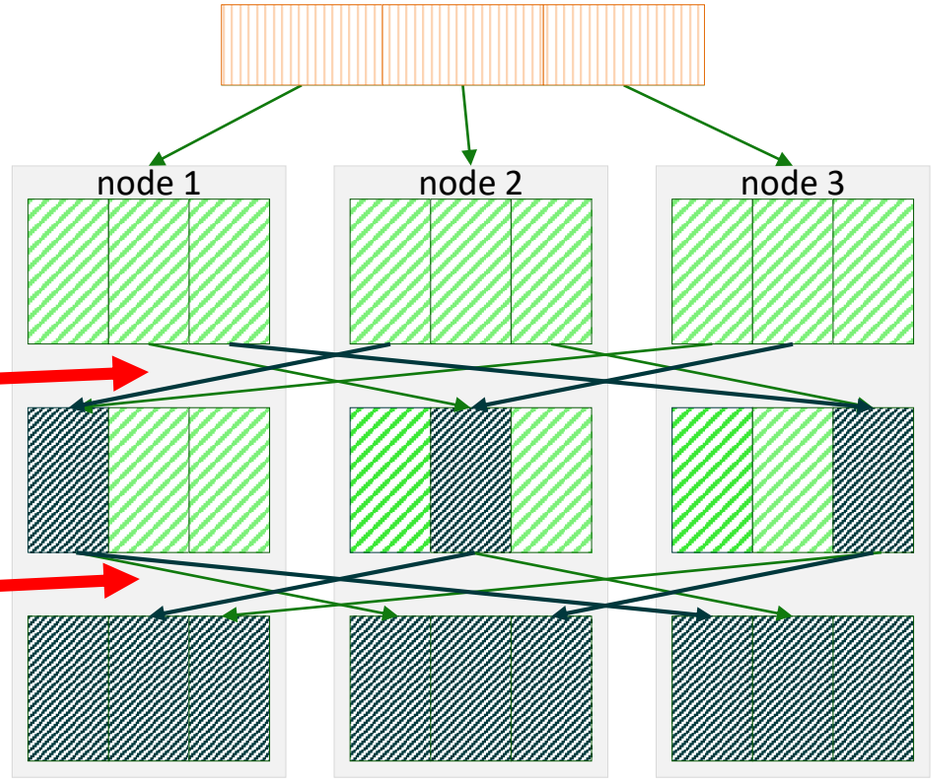
How to reduce communication cost:

communicate less each time

- 1-bit SGD:
  [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "1-Bit Stochastic Gradient Descent... Distributed Training of Speech DNNs", Interspeech 2014]
  - Quantize gradients to 1 bit per value
  - Trick: carry over quantization error to next minibatch

1-bit quantized with residual

1-bit quantized with residual



1-Bit SGD

```
local_learner = momentum_sgd(network['output'].parameters,
                             lr_schedule, mm_schedule,
                             l2_regularization_weight = l2_reg_weight)
learner = data_parallel_distributed_learner(local_learner,
                                            num_quantization_bits=num_quantization_bits,
                                            distributed_after=warm_up)
Trainer(network['output'], (network['ce'], network['pe']), learner, progress_printer)
```

Automatic MB Sizing

- Observation: given a learning rate, there is a maximum MB size that ensures model convergence
  - Note: we uses sum-gradient instead of average gradient for this work
- Learning rate shrinking during training
- At any learning rate change point:
  - Try a range of minibatch size on a small data block and pick the largest feasible one

[F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: “ON Parallelizability of Stochastic Gradient Descent...”, ICASSP 2014]

Train ImageNet in One Hour

- After a warm up, use large minibatch sizes
  - Note: average gradient is used with Caffe 2
- Linear scaling the learning rate

- Same as use sum gradient with fixed learning rate
  - Default behavior of CNTK with sum-gradient
  - Special case of the previous automatic MB sizing approach

[Goyal et al.: “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour,” ArXiv. 2017]

ResNet50 Experiments

GPUs	Facebook Results	Our Results
8-GPUs	23.60 ± 0.12	23.34
128-GPUs	23.56 ± 0.12	23.52
256-GPUs	23.74 ± 0.09	23.71

- On 8-GPUs, our baseline is slightly better than FB
- We observe slight degradation when scaling to more GPUs

BN-Inception Experiments

GPUs	Without Warm-up	With Warm-up
8-GPUs	25.110	-
16-GPUs	25.270	-
64-GPUs	25.958	25.228
128-GPUs	27.446	25.722
256-GPUs	35.934	27.690

- Exponentially decaying learning rate
- Gradual warm-up trick is also useful in GoogleNet, but cannot achieve baseline's accuracy.

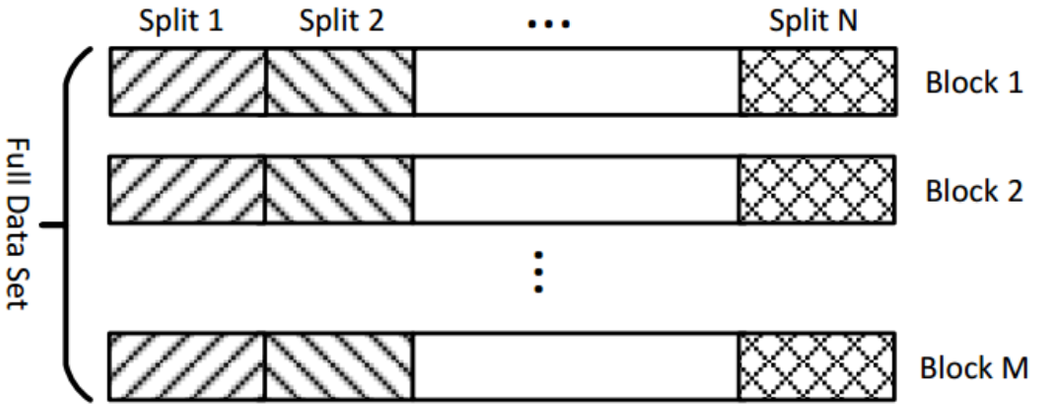
Block Momentum: Data Partition

- Partition randomly training dataset $\mathcal{D}$ into S mini-batches

$$\mathcal{D} = \{\mathcal{B}_i | i = 1,2, \dots, S\}$$

- Group every τ mini-batches to form a split
- Group every N splits to form a data block
- Training dataset $\mathcal{D}$ consists of M data blocks

$$S = M \times N \times \tau$$

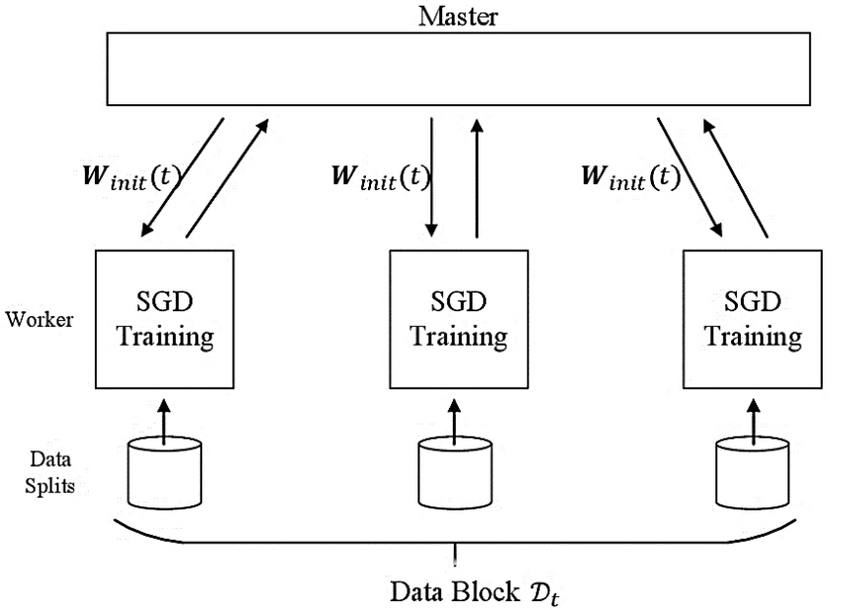


→Training dataset is processed block-by-block

→**Incremental Block Training (IBT)**

Intra-Block Parallel Optimization (IBPO)

- Select randomly an unprocessed data block denoted as $\mathcal{D}_t$
- Distribute N splits of $\mathcal{D}_t$ to N parallel workers
- Starting from an initial model denoted as $\boldsymbol{W}_{init}(t)$, each worker optimizes its local model independently by 1-sweep mini-batch SGD with momentum trick
- Average N optimized local models to get $\overline{\boldsymbol{W}}(t)$



Blockwise Model-Update Filtering (BMUF)

- Generate model-update resulting from data block $\mathcal{D}_t$:

$$\boldsymbol{G}(t) = \overline{\boldsymbol{W}}(t) - \boldsymbol{W}_{init}(t)$$

- Calculate global model-update:

$$\boldsymbol{\Delta}(t) = \eta_t \cdot \boldsymbol{\Delta}(t-1) + \varsigma_t \cdot \boldsymbol{G}(t)$$

  - ς_t: Block Learning Rate (BLR)
  - η_t: Block Momentum (BM)
  - When $\varsigma_t = 1$ and $\eta_t = 0$ → MA

- Update global model

$$\boldsymbol{W}(t) = \boldsymbol{W}(t-1) + \boldsymbol{\Delta}(t)$$

- Generate initial model for next data block
  - Classical Block Momentum (CBM)

$$\boldsymbol{W}_{init}(t+1) = \boldsymbol{W}(t)$$

  - Nesterov Block Momentum (NBM)

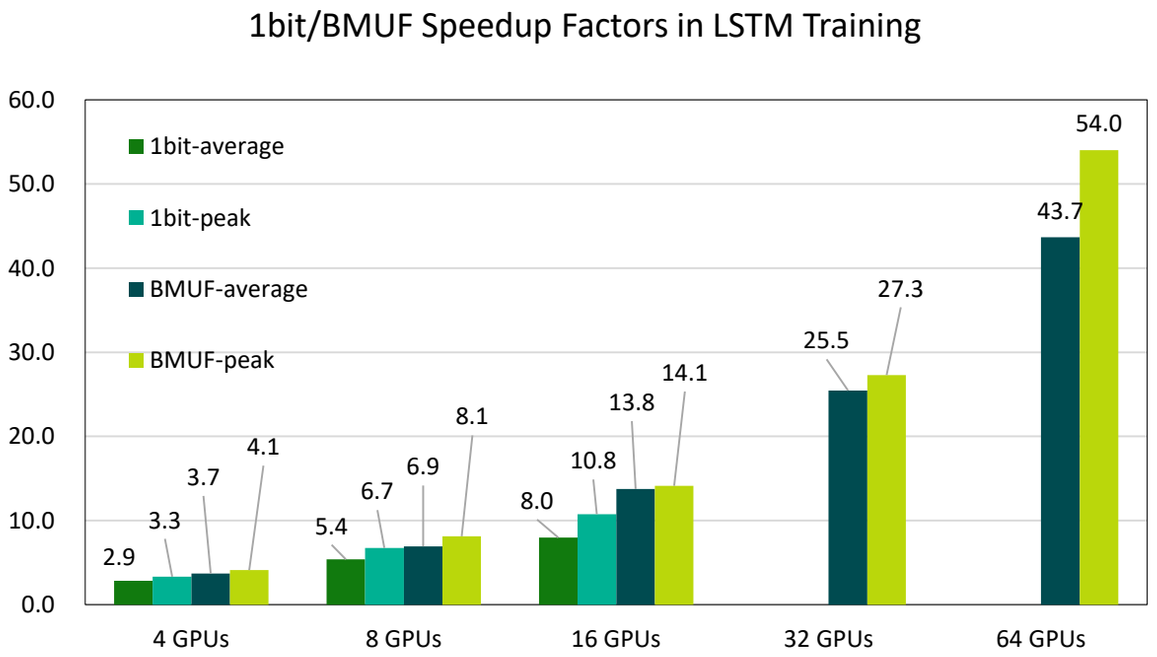
$$\boldsymbol{W}_{init}(t+1) = \boldsymbol{W}(t) + \eta_{t+1} \cdot \boldsymbol{\Delta}(t)$$

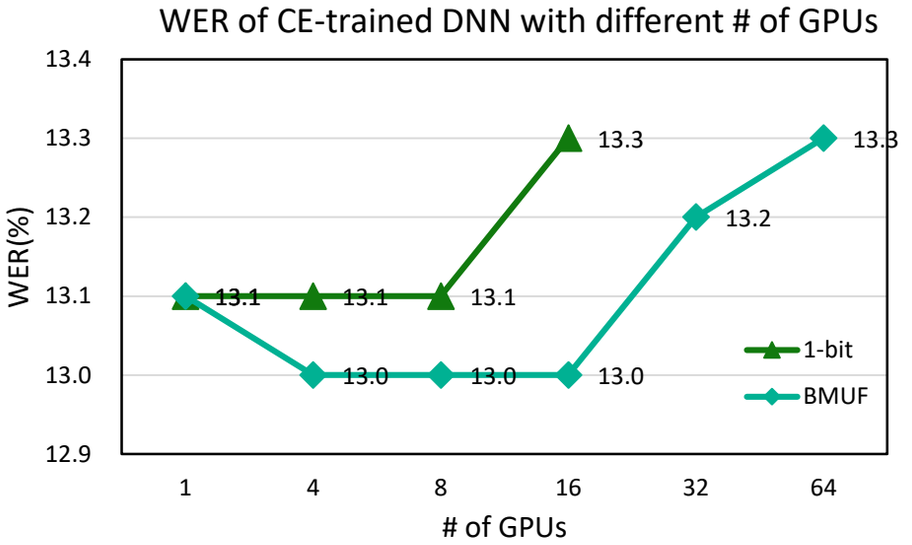
Iteration

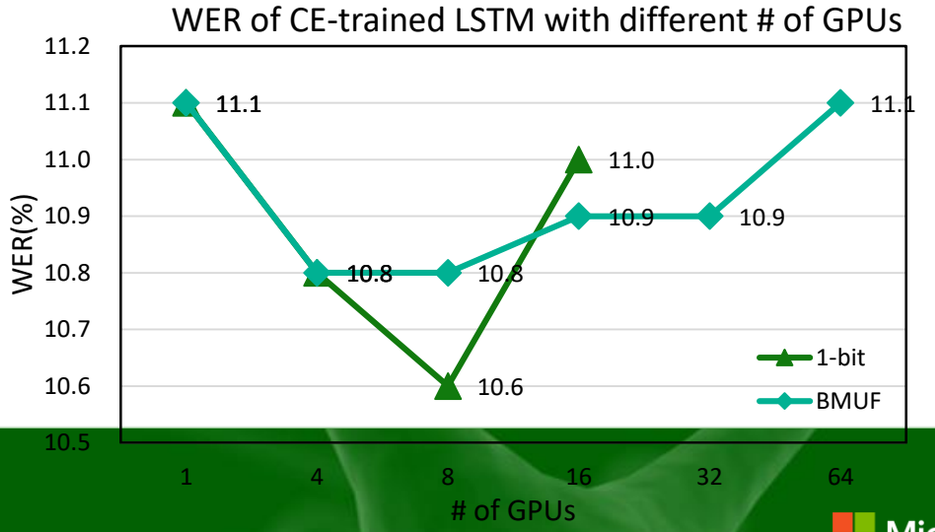
- Repeat IBPO and BMUF until all data blocks are processed
  - So-called “one sweep”
- Re-partition training set for a new sweep, repeat the above step
- Repeat the above step until a stopping criterion is satisfied
  - Obtain the final global model $\boldsymbol{W}_{final}$

Benchmark Result of Parallel Training on CNTK

- Training data: 2,670-hour speech from real traffics of VS, SMD, and Cortana
  - About 16 and 20 days to train DNN and LSTM on 1-GPU, respectively







Microsoft Cognitive Toolkit

Credit: Yongqiang Wang, Kai Chen, Qiang Huo

Microsoft

Results

- Almost linear speedup without degradation of model quality
- Verified for training DNN, CNN, LSTM up to 64 GPUs for speech recognition, image classification, OCR, and click prediction tasks
- Used for enterprise scale production data loads
- Production tools in other companies such as iFLYTEK and Alibaba

Block Momentum SGD

```
local_learner = momentum_sgd(network['output'].parameters,
                             lr_schedule, mm_schedule,
                             l2_regularization_weight = l2_reg_weight)
learner = block_momentum_distributed_learner(local_learner,
                                             block_size=block_size)
Trainer(network['output'], (network['ce'], network['pe']), learner, progress_printer)
```


Porting Models from Caffe

CrossTalkCaffe

- An easy-to-use tool for converting Caffe models to CNTK
  - Both training scripts and run time model
  - No knowledge on CNTK/Caffe required
  - Minimize migration time and cost
- A general framework for CNTK converters
  - Model → Intermediate presentation → Model
- A debugger for validating the converted model
  - Quantitative analysis for each component (1e-6 numeric accuracy)
- Release to CNTK contrib (Jul/Aug 2017)

Three Steps

- Step1: Prepare the Caffe prototxt and model
- Setp2: Configure the *global_conf* file
  - Source solver: configuration about Caffe model
  - Model solver: configuration about CNTK model
  - Validation solver: configuration about identity check
- Step3: Do converting with command

  ModelConverter.convert_model(global_conf.json)

global_conf.json for AlexNet

Basic information of the Caffe model

Basic information of the CNTK converted model

Validation nodes configuration

```
{
    "SourceSolver" : {
        "Source": "Caffe",
        "ath": "./Classification/AlexNet_ImageNet/deploy.prototxt",
        sPath": "./Classification/AlexNet_ImageNet/bvlc_alexnet.caffemodel",
        "PHASE": 1
    },
    "ModelSolver": {
        "CNTKModelPath": "./Classification/AlexNet_ImageNet/AlexNet_ImageNet.cntkmodel"
    },
    "ValidSolver": {
        "SavePath": "./Classification/AlexNet_ImageNet/Valid",
        "ValInputs": {
            "data": [[0, 255], []]
        },
        les": {
            ob": "prob"
        }
    }
}
```

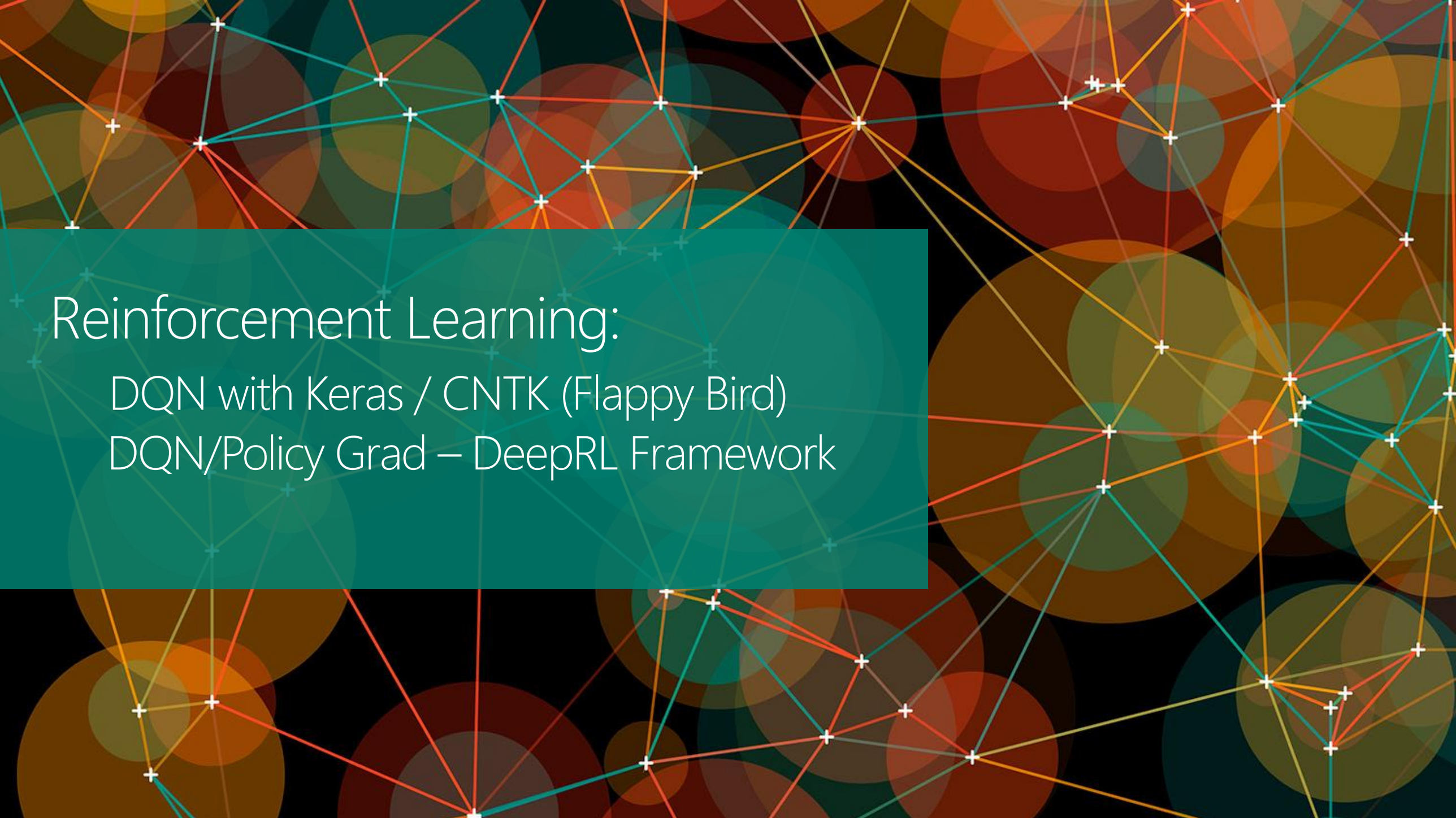
Another Example – ResNet50

```
{
    "SourceSolver" : {
        "Source": "Caffe",
        "ModelPath": "./Classification/ResNet_ImageNet/ResNet-50-deploy.prototxt",
        "WeightsPath": "./Classification/ResNet_ImageNet/ResNet-50-model.caffemodel",
        "PHASE": 1
    },

    "ModelSolver": {
        "CNTKModelPath": "./Classification/ResNet_ImageNet/ResNet50_ImageNet.cntkmodel"
    },

    "ValidSolver": {
        "SavePath": "./Classification/ResNet_ImageNet/Valid",
        "ValInputs": {
            "data": [[0, 255], []]
        },

        "ValNodes": {
            "prob": "prob"
        }
    }
}
```

CNTK model download path	https://www.cntk.ai/Models/Caffe_Converted/ResNet50_ImageNet_Caffe.model
Last updated	April, 28th, 2017
Source Caffe model website	https://github.com/KaimingHe/deep-residual-networks
Single crop top 5 error	7.75%

Current Status

- Executable tool and source code
  - Tested with all mainstream CNN models
    - AlexNet, NIN, VGG, GoogLeNet, ResNet for classification
    - Fast/Faster RCNN, RFCN for segmentation
  - Support CNTK user define function
  - A validation module is provided to monitor arbitrary nodes between Caffe and CNTK
- Widely used for internal model conversion and debugging


Reinforcement Learning:
DQN with Keras / CNTK (Flappy Bird)
DQN/Policy Grad – DeepRL Framework

Overview

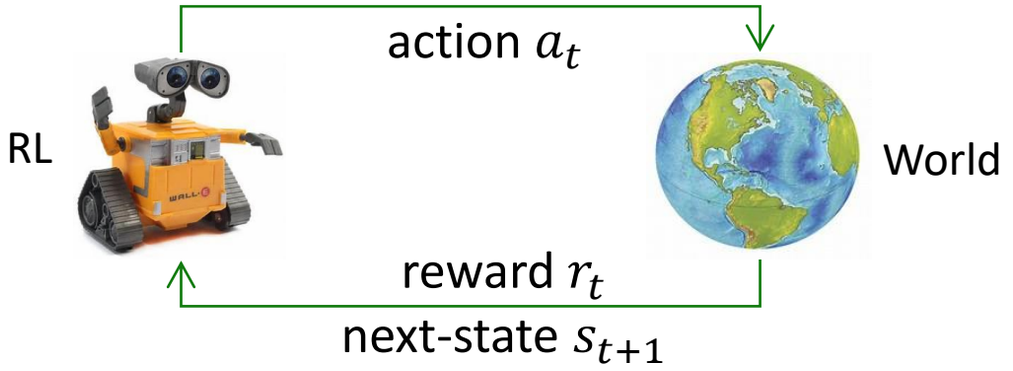
- Short review RL basics
- CNTK backend for Keras
  - Interoperability with other toolkits
  - Demo: DQN based game using pre-trained model from TensorFlow
- In-depth modeling
  - Compute forward and backward passes
  - Demo: Reinforcement learning tutorial (CNTK 203)
- Extensible framework
  - DeepRL framework (CNTK/bindings/python/cntk/contrib/deeprl)

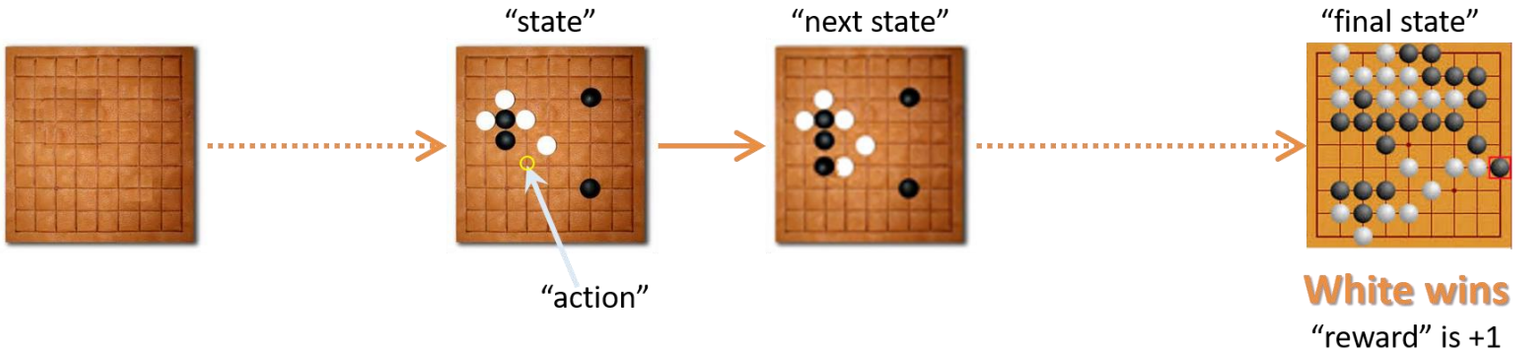
Reinforcement Learning

Take away

- CNTK's Keras support enables interoperability with other toolkits
- Use low-level API for custom RL modeling
- An extensible RL framework with out-of-the-box popular model support

RL Problem


action a_t
RL
World
reward r_t
next-state s_{t+1}


"state"
"next state"
"final state"
"action"
White wins
"reward" is +1

Q-Learning Algorithm

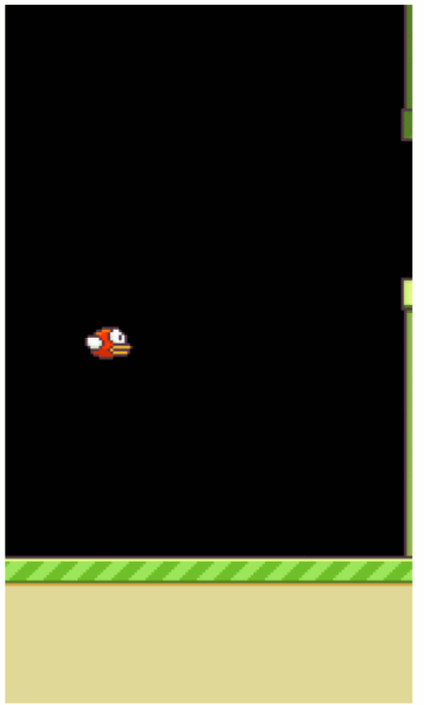
```
# Initialize
Initialize Q[number of states, number of actions]

# Initial statue from the environment
Observe s

# Iterate
while( not terminate ):
    Choose an action (a) given a state (s)
    Receive reward (r) and new state (s')
    Compute:
    Q[s,a] += α(R + γ(max_a, (Q[s',a']-Q[s,a]))
    Update state s = s'
```

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \ldots)) = r_t + \gamma R_{t+1}$$

Reinforcement Learning with Flappy Bird

Steps:

- Receive the Game screen input (as image array)
- Pre-process the image
- Process the image using a CNN
- Predict action: Flap vs. no Flap
- Use Q-learning to maximize the future reward



Code

- https://github.com/yanpanlau/Keras-FlappyBird

Reinforcement Learning with Flappy Bird

- Input: `a_t`
- Game API: return
  - Next frame: `x_t1_colored`
  - Reward: `r_t` (0.1 if alive, +1 if pass pipe, -1 die)
  - Game state: `terminal` (bool flag: Finished or Not)

```
import game.wrapped_flappy_bird as game

# open up a game state to communicate with emulator
game_state = game.GameState()

#run the selected action, and observe next state and reward (unflap/downflap: a_t)
x_t1_colored, r_t, terminal = game_state.frame_step(a_t)
```

Image Preprocessing

- Convert image to grayscale
- Crop the image to 80 x 80 pixel
- Stack 4 frames together
  - Allows the model to infer bird velocity
  - `x_t1`: (1 x 1 x 80 x 80)
  - `s_t1`: (1 x 4 x 80 x 80)

```
x_t1 = skimage.color.rgb2gray(x_t1_colored)
x_t1 = skimage.transform.resize(x_t1,(80,80))
x_t1 = skimage.exposure.rescale_intensity(x_t1,
                                          out_range=(0, 255))

# Reshape x_t1 to (1x1x80x80) and generate s_t1 (1x4x80x80)
x_t1 = x_t1.reshape(1, x_t1.shape[0], x_t1.shape[1], 1)
s_t1 = numpy.append(x_t1, s_t[:, :, :, :3], axis=3)
```

CNN for Action Prediction

- Predicts whether the bird should flap or not
- Note:
  - Initialization is key: use `normal` distribution
  - The image data should be `(4 x 80 x 80)`
  - `subsample=(2,2)` is equivalent to `stride`
  - `Adam` optimizer (learner): Learning rate is `1-e6`

Model

```
def buildmodel():
    print("Now we build the model")
    model = Sequential()
    model.add(Convolution2D(32, 8, 8, subsample=(4, 4), border_mode='same',
                            input_shape=(img_rows,img_cols,img_channels)))  #80*80*4
    model.add(Activation('relu'))

    model.add(Convolution2D(64, 4, 4, subsample=(2, 2), border_mode='same'))
    model.add(Activation('relu'))

    model.add(Convolution2D(64, 3, 3, subsample=(1, 1), border_mode='same'))
    model.add(Activation('relu'))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dense(2))

    adam = Adam(lr=LEARNING_RATE)
    model.compile(loss='mse',optimizer=adam)
    return model
```

Deep Q Network

`Q(s,a)` - function:
a future reward for choosing
action `a` in state `s`

L – loss function ( )

$$L = [r + \gamma max_{a'} Q(s', a') - Q(s, a)]^2$$

```
#sample a minibatch to train on
minibatch = random.sample(exp_replay, BATCH)

inputs = numpy.zeros((BATCH, s_t.shape))
targets = numpy.zeros((inputs.shape[0], ACTIONS))

for i in range(0, len(minibatch)):
    state_t, action_t = minibatch[i][0:1]
    reward_t, terminal = minibatch[i][2:3]
    inputs[i:i + 1] = state_t

    targets[i] = model.predict(state_t)
    Q_sa = model.predict(state_t1)

    if terminal:
        targets[i, action_t] = reward_t
    else:
        targets[i, action_t] = reward_t + \
            GAMMA * numpy.max(Q_sa)

loss += model.train_on_batch(inputs, targets)
```

Experience Replay

- During game play episodes `(s, a, r, s')` are stored in a buffer
  - A.k.a: Replay memory
- During training, random minibatches from replay memory (D) are used
  - Instead of recent states and action

```
exp_replay.append((s_t, action_index, r_t, s_t1, terminal))

if len(exp_replay) > REPLAY_MEMORY:
    exp_replay.popleft()

#only train if done observing
if t > OBSERVE:
    #sample a minibatch to train on
    minibatch = random.sample(exp_replay, BATCH)
```

- Advantages:
  - Randomization prevents the optimizer from getting trapped in local minima
  - Makes the task similar to supervised learning, helps with troubleshooting

Explore-Exploit

- Initially the agent generate random Q-value and max value is picked
- The agent randomly explores
- Eventually the Q-function is learnt but it is greedily selected
- ε-greedy approach:
  - Agent selects random actions with a certain probability (exploration)
  - Else greedily select an action that gives highest reward (exploitation)
- Practically: Start with ε =1 and reduce it to 0.1

```
if random.random() <= epsilon:
    print("----------Random Action----------")
    action_index = random.randrange(ACTIONS)
    a_t[action_index] = 1
else:
    # input a stack of 4 images,
    # get the prediction
    q = model.predict(s_t)
    max_Q = np.argmax(q)
    action_index = max_Q
    a_t[max_Q] = 1
```

Policy Gradient

Goal

- Average reward collected per episode, by running policy (set of actions in an episode) with parameter θ

$$J(\theta) = \mathbf{E}_{\theta}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t\right]$$

- Maximize the expected discounted reward across entire episode

Approach

- Collect experience (sample a bunch of trajectories through (s, a) space)
- Update the policy so that good experiences become more probable

Step-by-Step Tutorial

CNTK 203: Reinforcement Learning Basics

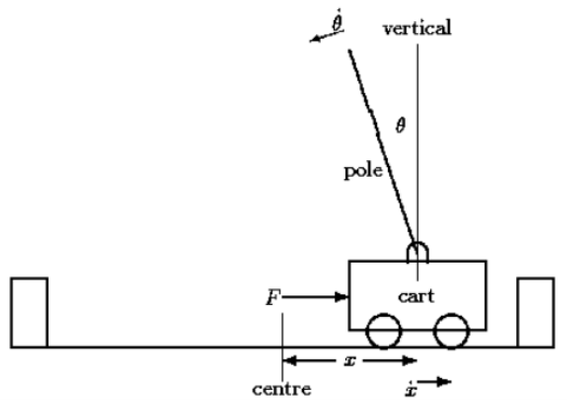
Reinforcement learning (RL) is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. In machine learning, the environment is typically formulated as a Markov decision process (MDP) as many reinforcement learning algorithms for this context utilize dynamic programming techniques.

In some machine learning settings, we do not have immediate access to labels, so we cannot rely on supervised learning techniques. If, however, there is something we can interact with and thereby get some feedback that tells us occasionally, whether our previous behavior was good or not, we can use RL to learn how to improve our behavior.

Unlike in supervised learning, in RL, labeled correct input/output pairs are never presented and sub-optimal actions are never explicitly corrected. This mimics many of the online learning paradigms which involves finding a balance between exploration (of conditions or actions never learnt before) and exploitation (of already learnt conditions or actions from previous encounters). Multi-arm bandit problems is one of the category of RL algorithms where exploration vs. exploitation trade-off have been thoroughly studied. See figure below for reference.

In [2]:
```
# Figure 1
Image(url="https://cntk.ai/jup/polecart.gif", width=300, height=300)
```

Out[2]:



Microsoft Cognitive Toolkit

Microsoft

Policy Gradient in CNTK

- Simple illustrative model

```
H = 10 # number of hidden layer neurons
observations = C.sequence.input_variable(STATE_COUNT, numpy.float32, name="obs")

W1 = C.parameter(shape=(STATE_COUNT, H), init=C.glorot_uniform(), name="W1")
b1 = C.parameter(shape=H, name="b1")
layer1 = C.relu(C.times(observations, W1) + b1)

W2 = C.parameter(shape=(H, ACTION_COUNT), init=C.glorot_uniform(), name="W2")
b2 = C.parameter(shape=ACTION_COUNT, name="b2")
score = C.times(layer1, W2) + b2

probability = C.sigmoid(score, name="prob")
```

Policy Search

- Use of classic `loss.forward` and `loss.backward`

```
gradBuffer = dict((var.name, np.zeros(shape=var.shape)) \
                  for var in loss.parameters if var.name in ['W1', 'W2', 'b1', 'b2'])

# Forward pass
state, outputs_map = loss.forward(arguments, outputs=loss.outputs,
                                  keep_for_backward=loss.outputs)

# Backward pass
root_gradients = {v: np.ones_like(o) for v, o in outputs_map.items()}
vargrads_map = loss.backward(state, root_gradients, variables=set([W1, W2]))

for var, grad in vargrads_map.items():
    gradBuffer[var.name] += grad
```

Deep RL Framework (Preview)

- Flexible and extensible framework for Deep RL Algorithms
- Out of the box support for different RL agents:
  - Random
  - Tabular Q-learning
  - Deep Q-Learning
  - Policy Gradient (Actor-Critic method)

DeepRL: Actor-Critic Method

- Learn approximation to both policy and value fns.
  - Policy approximation: Actor
  - Value approximation: Critic
- Deep RL Framework

`source: CNTK/bindings/python/cntk/contrib/deepRL`

```
python bin/run.py
    --env=CartPole-v0
    --max_steps=100000
    --agent=actor_critic
    --agent_config=config_examples/policy_gradient.cfg
    --eval_period=1000
    --eval_steps=20000
```

Extensible Agent

```
@abstractmethod
def start(self, state):
    """Start a new episode. return (action) """

@abstractmethod
def step(self, reward, next_state):
    """Observe one transition and choose an action. return (action) """

@abstractmethod
def end(self, reward, next_state):
    """Last observed reward/state of the episode(which then terminates).
       return (last_reward)"""
```


https://github.com/Microsoft/CNTK/blob/master/bindings/python/cntk/contrib/deeprl/agent/agent.py


Conclusion / Q & A

Conclusions – CNTK Advantages

- Performance
  - Speed: faster than others, 5-10x faster on recurrent networks
  - Accuracy: validated examples/recipes
  - Scalability: few lines of change to scale to thousands of GPUs
  - Built-in readers: efficient distributed readers
- Programmability
  - Powerful C++ library for enterprise users
  - Intuitive and performant Python APIs
  - C#/.NET/Java inference support
  - Extensible via user custom layers, learners, readers, etc.

Q&A

https://github.com/Microsoft/CNTK