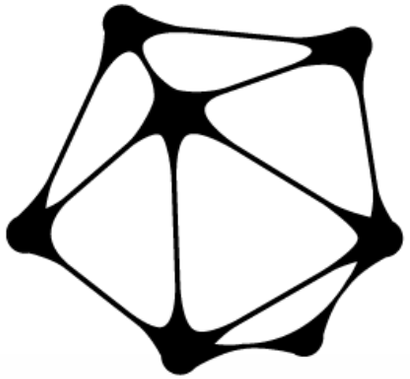
Microsoft



Cognitive Toolkit

Emad Barsoum
Principal Engineering Manager, AI & Research

Agenda

- Introduction
  - Deep learning at Microsoft
  - What is Cognitive toolkit (CNTK)?
  - Introduction to Deep Neural Networks
  - Cognitive toolkit basics
- CNTK Workflow
  - Data reading and augmentations
  - Modeling (MLP, CNN, RNN)
  - Training-Test-Eval workflow
- Models via layer APIs
  - Image: ResNet, Inception, ConvNet, Emotion and GAN.
  - Video: 3D Convolution and RNN.
  - Customization

Agenda

- Inference performance
  - Compression
  - Pruning
- Parallel training
  - 1-bit SGD
  - Block momentum
  - Variable sized mini-batch
  - DC-ASGD
- Conclusion / Q&A

Deep learning at Microsoft

Deep learning at Microsoft

Microsoft Cognitive Services

Skype Translator

Cortana

Bing

HoloLens

Microsoft Research

Cortana

Bing

Microsoft Research

Microsoft HoloLens

XBOX

Microsoft

Microsoft's new speech breakthrough

Microsoft 2017 research system for conversational speech recognition

5.1% word-error rate

Enabled by CNTK's multi-level scalability

W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, A. Stolcke:

"The Microsoft 2017 Conversational Speech Recognition System,"

Microsoft researchers achieve new conversational speech recognition milestone

August 20, 2017 | By Microsoft blog editor

Share Tweet Share

By Xuedong Huang, Technical Fellow, Microsoft

Last year, Microsoft's speech and dialog research group announced a milestone in reaching human parity on the Switchboard conversational speech recognition task, meaning we had created technology that recognized words in a conversation as well as professional human transcribers.

After our transcription system reached the 5.9 percent word error rate that we had measured for humans, other researchers conducted their own study, employing a more involved multi-transcriber process, which yielded a 5.1 human parity word error rate. This was consistent with prior research that showed that humans achieve higher levels of agreement on the precise words spoken as they expend more care and effort. Today, I'm excited to announce that our research team reached that 5.1 percent error rate with our speech recognition system, a new industry milestone, substantially surpassing the accuracy we achieved last year. A technical report published this weekend documents the details of our system.

Chat

"Hey Cortana, tell me a joke."

Switchboard is a corpus of recorded telephone conversations that the speech research community has used for more than 20 years to benchmark speech recognition systems. The task involves transcribing conversations between strangers discussing topics such as sports and politics.

We reduced our error rate by about 12 percent compared to last year's accuracy level, using a series of improvements to our neural net-based acoustic and language models. We introduced an additional CNN-BLSTM (convolutional neural network combined with bidirectional long-short-term memory) model for improved acoustic modeling. Additionally, our approach to combine predictions from multiple acoustic models now does so at both the frame/senone and word levels.

Moreover, we strengthened the recognizer's language model by using the entire history of a dialog session to predict what is likely to come next, effectively allowing the model to adapt to the topic and local context of a conversation.

Our team also has benefited greatly from using the most scalable deep learning software available, Microsoft Cognitive Toolkit 2.1 (CNTK), for exploring model architectures and optimizing the hyper-parameters of our models. Additionally, Microsoft's investment in cloud compute infrastructure, specifically Azure GPUs, helped to improve the effectiveness and speed by which we could train our models and test new ideas.

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

What is Cognitive Toolkit (CNTK)?

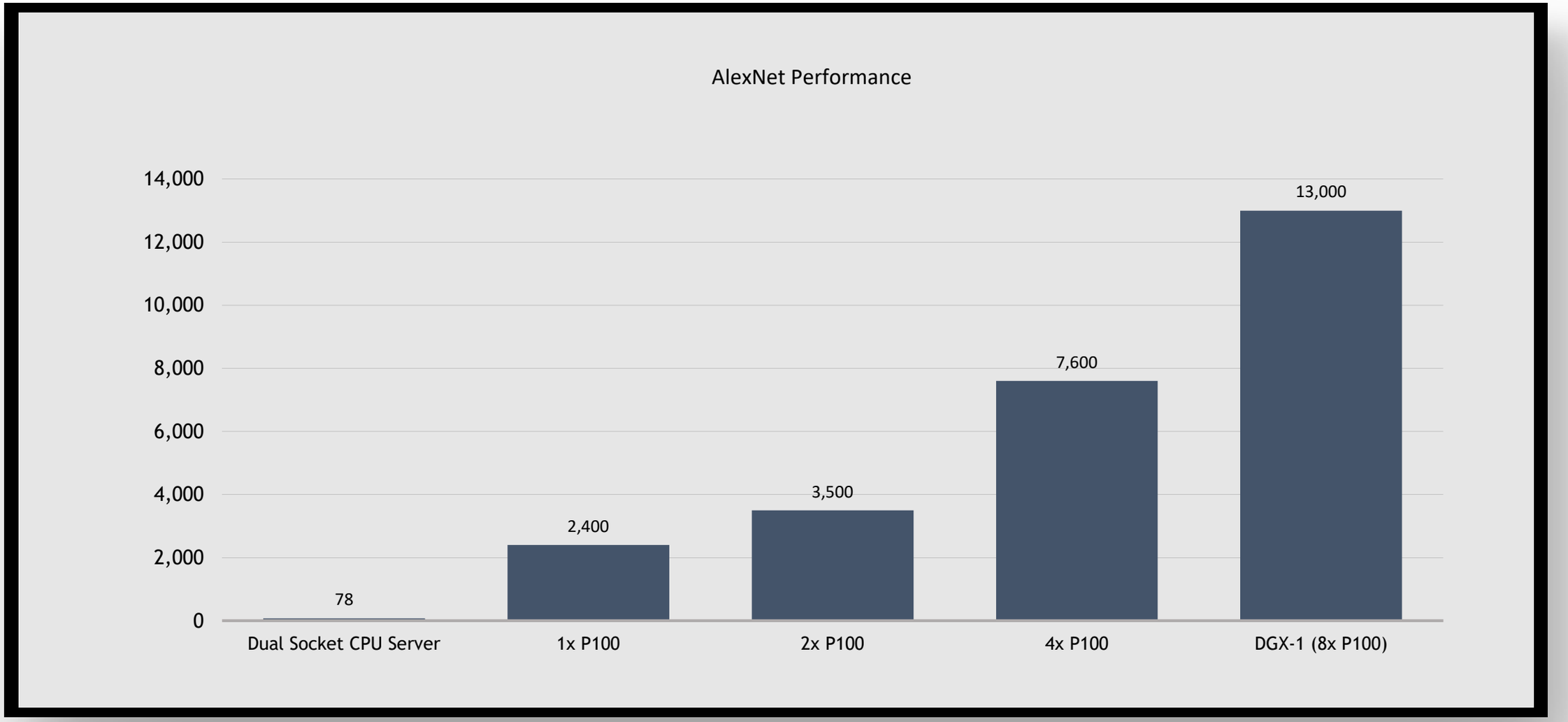
What is Cognitive Toolkit

- Microsoft's open-source deep-learning toolkit
  - https://github.com/Microsoft/CNTK
- Created by Microsoft Speech researchers in 2012
- On GitHub since Jan 2016 under MIT license
- 13,000+ GitHub stars and 140+ contributors
- Internal == External
- 1st-class on Linux and Windows, docker support

What is Cognitive Toolkit

- Core implemented in C++ and CUDA
- Low level + high level Python API
- C++, C# and Python API for train and eval
- Easy extensible at multiple level
- Built-in high performance readers
- Float16 support on NVidia Volta GPU
- Model compression (Fast binarized evaluation)

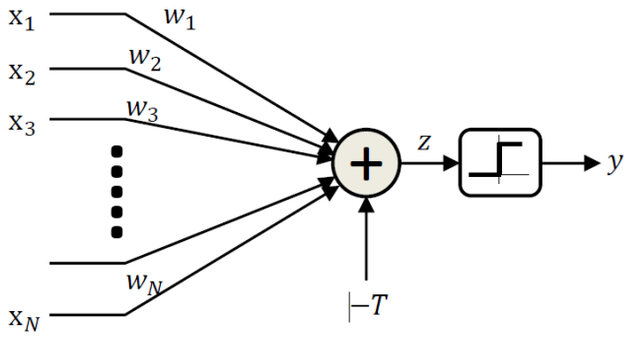
Near-Linear GPU Scaling



Introduction to Deep Neural Networks

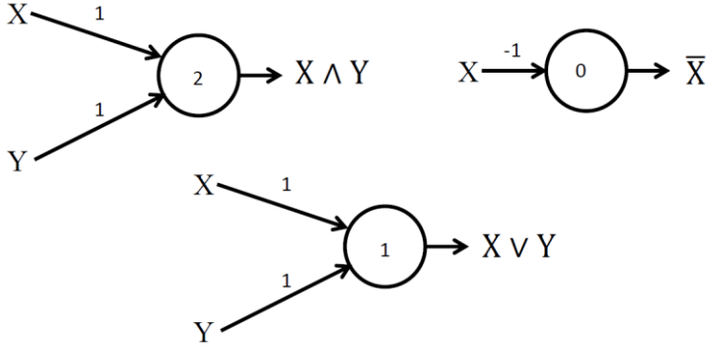
What is Neural Network

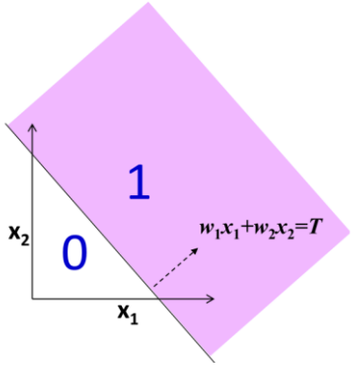
Neural networks are universal function approximators: Boolean functions, regressions, classifiers

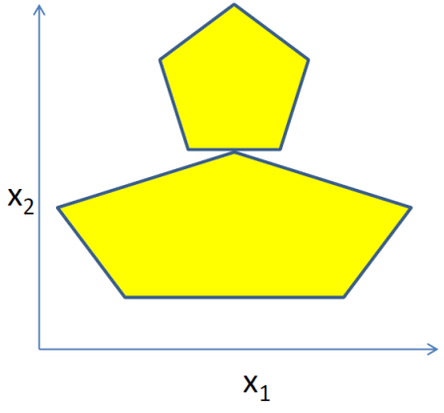


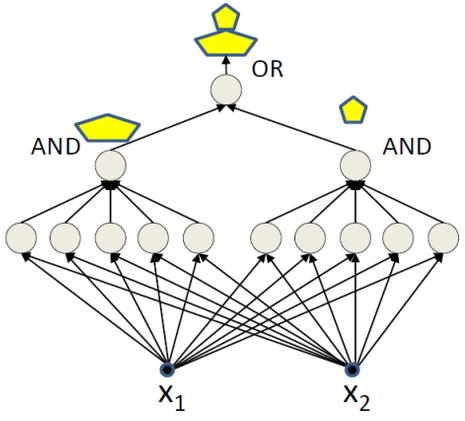
$$z = \sum_i w_i x_i - T$$

$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; else \end{cases}$$

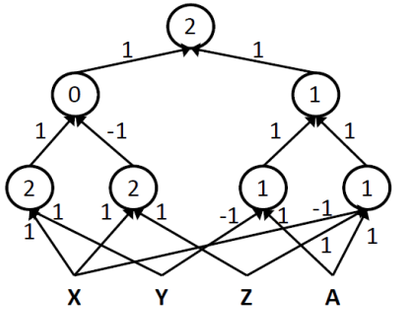


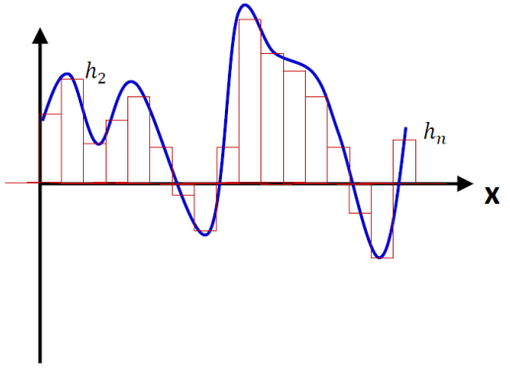


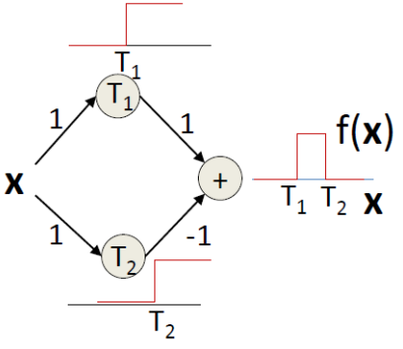




$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$



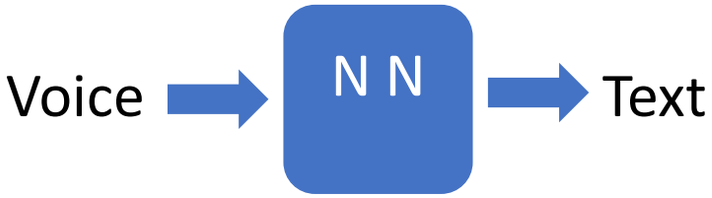


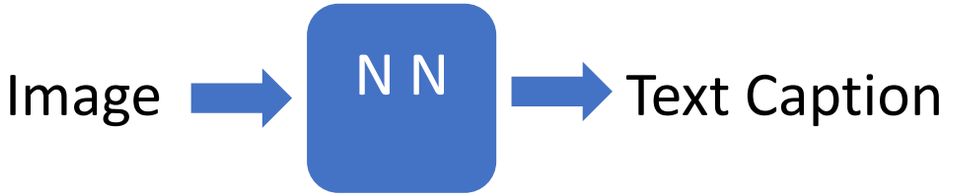


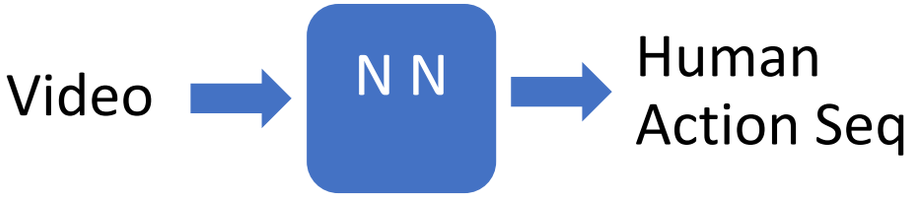
Figures in this slide are from © B. Raj, CMU with Bhiksha's courtesy permission

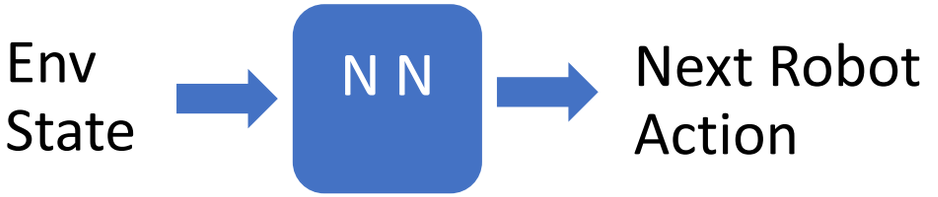
Microsoft

Tasks







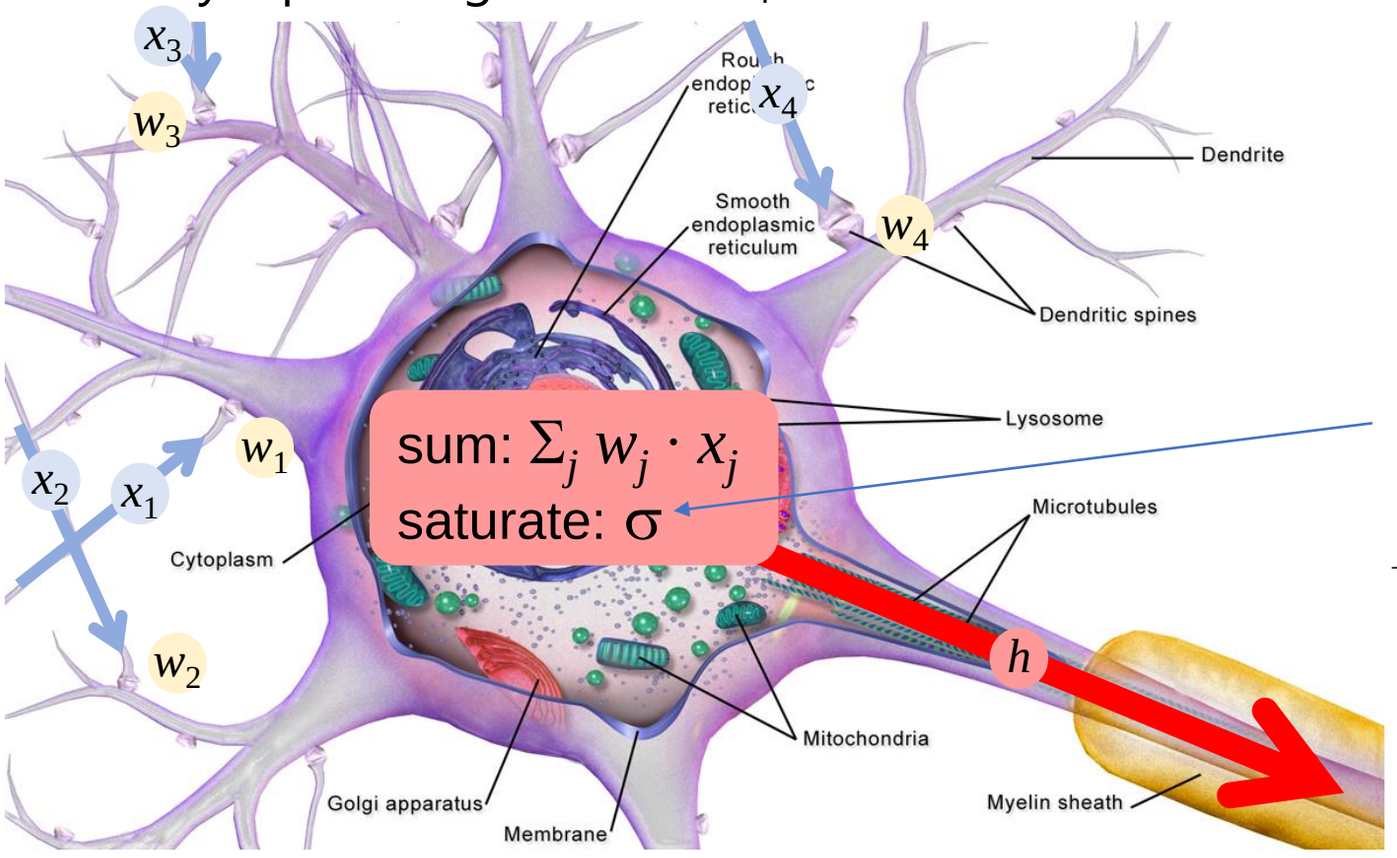


These tasks can be formulated as functions, e.g.

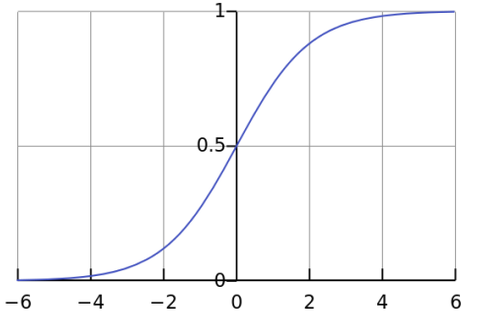
- f: Voice -> Text
- f: Video -> Action sequence

Neurons

- neurons are simple pattern detectors, **measure how well** inputs x_j **correlate** with synaptic weights w [Perceptron, Rosenblatt 1957]



example saturation: sigmoid function



[images from Wikipedia]

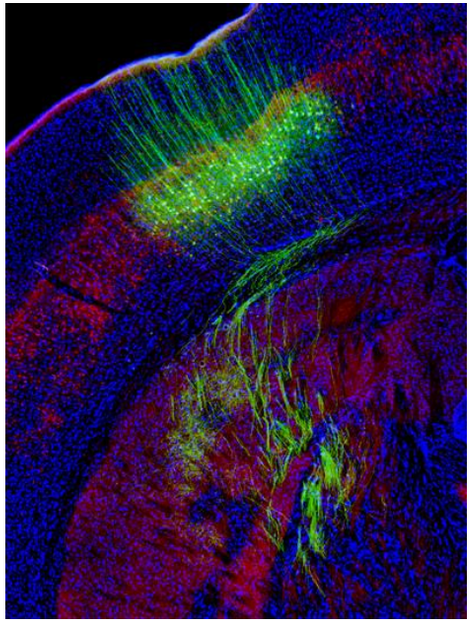
Neural Networks Pattern Matching

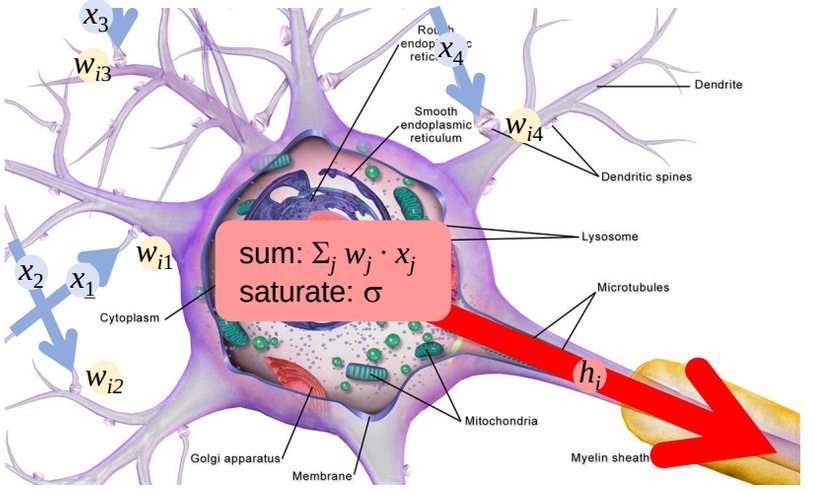
- Neurons are simple pattern detectors, measure how well inputs x_j **correlate** with synaptic weights w

$$h_i = \sigma(\Sigma_j\, w_{ij} \cdot x_j + b_i)$$

- Operate as **collections**, or vectors

$$h = \sigma(\mathbf{W}\, x + b)$$





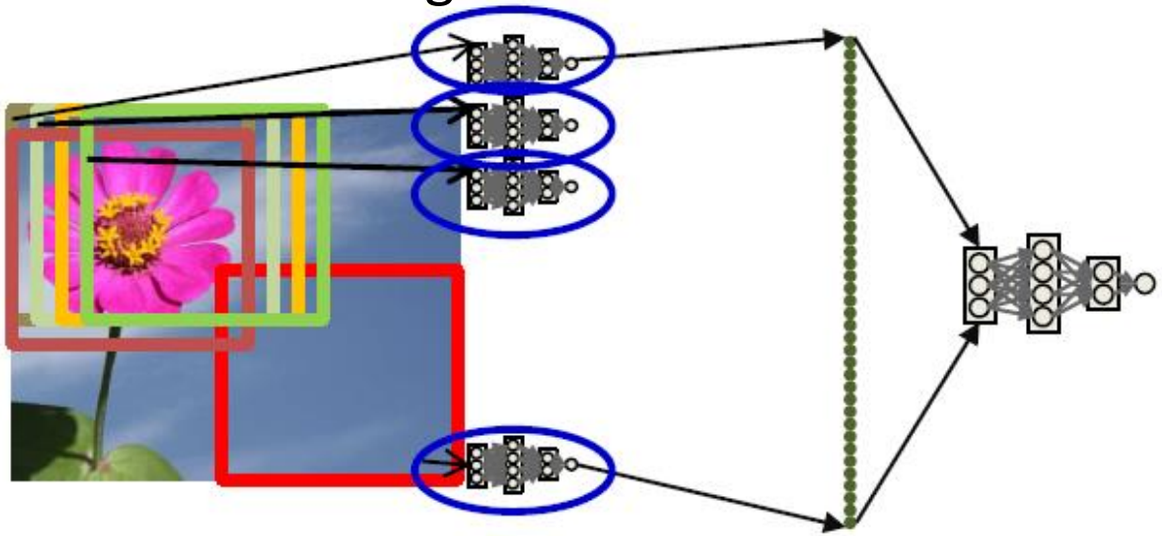
correlate with patterns

$$\mathbf{W}\, x = \begin{pmatrix} w_{11}, w_{12}, ..w_{1N} \\ w_{21}, w_{22}, ..w_{2N} \\ \dots \\ w_{M1}, w_{M2}, ..w_{MN} \end{pmatrix} \mathrm{x} = \begin{pmatrix} \mathrm{pattern}_1 \\ \mathrm{pattern}_2 \\ \dots \\ \mathrm{pattern}_N \end{pmatrix} \mathrm{x}$$

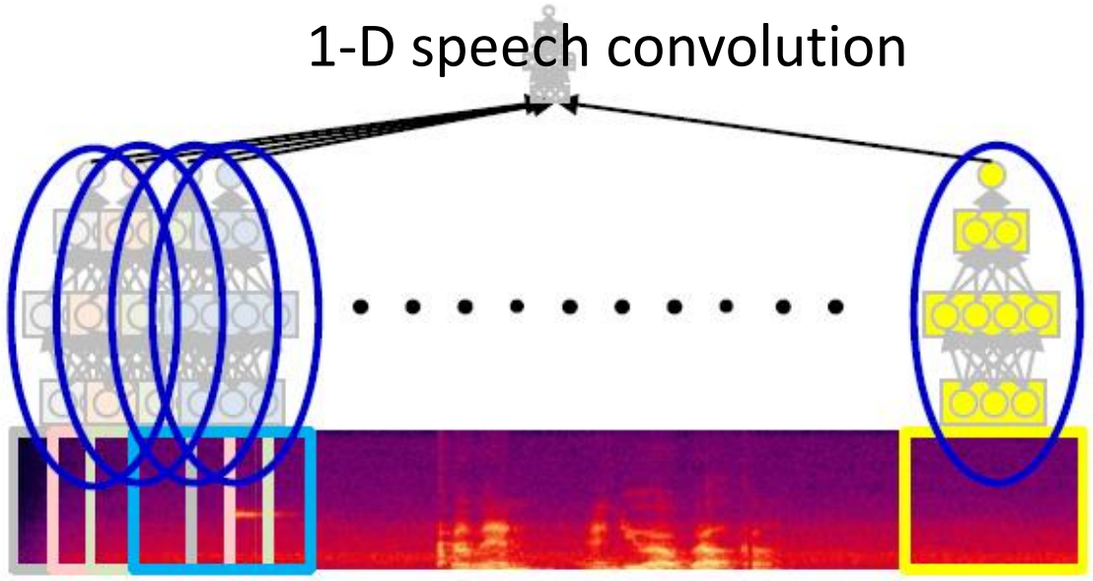
e.g. 2000 neurons

Parameters Sharing: Convolution and Recurrent Networks

2-D image convolution



1-D speech convolution



- **Convolution**: The same shared sub-network is applied **independently** over sliding windows of inputs; detecting **translation invariant patterns**
- **Recurrent**: The same shared sub-network is applied **independently** over sliding windows of inputs and previous outputs; detecting **iterated structure**, e.g. **long-term dependency on the "past"**

Figures in this slide are from © B. Raj, CMU with Bhiksha's courtesy permission

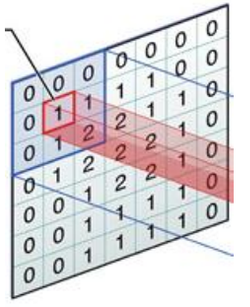
Summary

- Fully connected

$$h = \sigma(\mathbf{W}\, x + b)$$

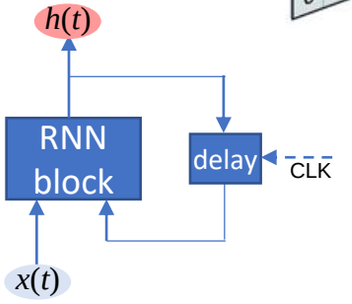
- Convolution: connectivity **can be local** (spatial receptive fields)

$$h(c,r) = \sigma(\mathbf{W}\, x(c\text{-}\Delta c..c\text{+}\Delta c, r\text{-}\Delta r..r\text{+}\Delta r) + b)$$



- Recurrent network: can form **feedback loops**

$$h(t) = \sigma(\mathbf{W}\, x(t) + \mathbf{R}\, h(t\text{-}1) + b)$$



- Arranged in **layers** → increasingly abstract representation

$$h^{(1)} = \sigma(\mathbf{W}^{(1)}\, x \quad + b^{(1)})$$

$$h^{(2)} = \sigma(\mathbf{W}^{(2)}\, h^{(1)} + b^{(2)})$$

$$\mathbf{W}^{(1)} = \begin{pmatrix} \text{pattern}_1 \\ \text{pattern}_2 \\ \dots \\ \text{pattern}_N \end{pmatrix}$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} \text{abstract pattern}_1 \\ \text{abstract pattern}_2 \\ \dots \\ \text{abstract pattern}_N \end{pmatrix}$$

Learning

- A constructed network represents a class of functions $Y = f(X; W)$ which is parameterized by weights W
- Learning a target function: $g(X)$ which has the desired outputs, e.g. classify objects, generate texts, and etc.
- Looking for appropriate W with desired behaviors so that $f(X; W)$ **approximates the unknown $g(X)$ as much as possible**
- We need a **loss** function to "measure" the distance between the neural network function and the target function: e.g. squared errors, cross entropy, KL-divergence and so on

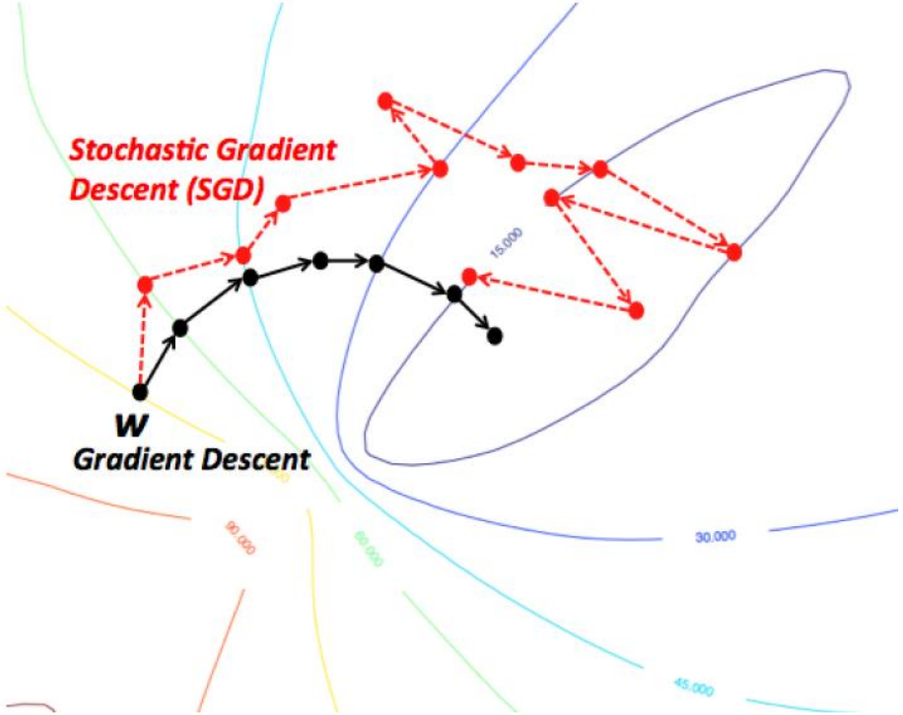
SGD

SGD

- For (X_t, Y_t) in the label data set (randomized)
  - $W_{t+1} = W_t - \eta_t \nabla_W loss(f(X_t; W_t), Y_t)$

MiniBatch SGD

- For $\mathrm{B_t} = \{(X_i, Y_i)\}$ in the label data set (randomized)
  - $W_{t+1} = W_t - \frac{1}{|B_t|} \eta_t \sum \nabla_W loss(f(X_i; W_t), Y_i)$



Key requirements to success

- Loss functions are differentiable or sub-gradients can be defined w.r.t. weights

Cognitive Toolkit Basics

Basics

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

Basics

Example: 2-hidden layer feed-forward NN

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

Basics

Example: 2-hidden layer feed-forward NN

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

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

Basics

Example: 2-hidden layer feed-forward NN

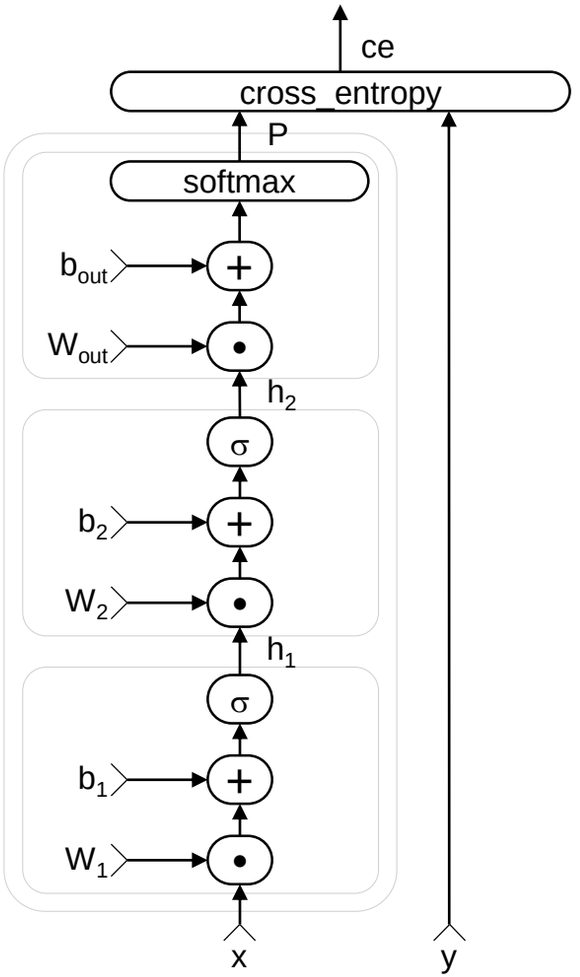
$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \text{softmax}(W_{\text{out}} h_2 + b_{\text{out}})$$

```
h1 = sigmoid (x  @ W1   + b1)
h2 = sigmoid (h1 @ W2   + b2)
P  = softmax (h2 @ Wout + bout)
```

$$ce = L^T \log P$$

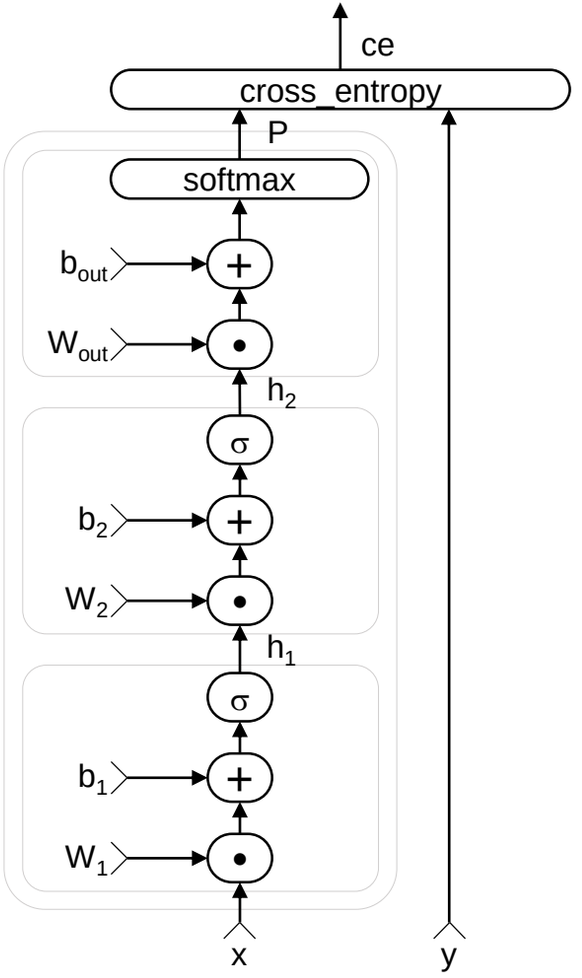
```
ce = cross_entropy (L, P)
```

Basics




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

Basics



- 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

Recurrent Network

Extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1\, x(t) + H_1\, h_1(t\text{-}1) + b_1)$$
$$h_2(t) = \sigma(\mathbf{W}_2\, h_1(t) + H_2\, h_2(t\text{-}1) + b_2)$$
$$P(t) = \text{softmax}(\mathbf{W}_{\text{out}}\, h_2(t) + b_{\text{out}})$$
$$ce(t) = L^{\text{T}}(t) \log P(t)$$

Recurrent Network

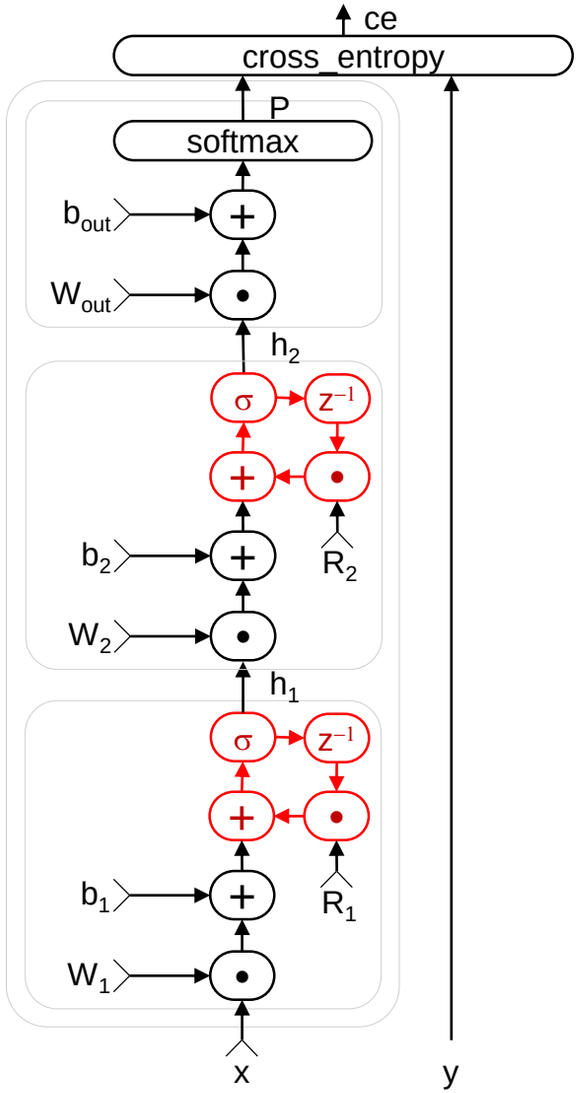
Extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1\, x(t) + H_1\, h_1(t\text{-}1) + b_1)$$
$$h_2(t) = \sigma(\mathbf{W}_2\, h_1(t) + H_2\, h_2(t\text{-}1) + b_2)$$
$$P(t) = \mathrm{softmax}(\mathbf{W}_{\mathrm{out}}\, h_2(t) + b_{\mathrm{out}})$$
$$ce(t) = L^{\mathrm{T}}(t) \log P(t)$$
$$\sum_{\mathrm{corpus}} ce(t) = \max$$

```
h1 = sigmoid(x  @ W1 + past_value(h1) + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ H2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

→ no explicit notion of time

Recurrent Network

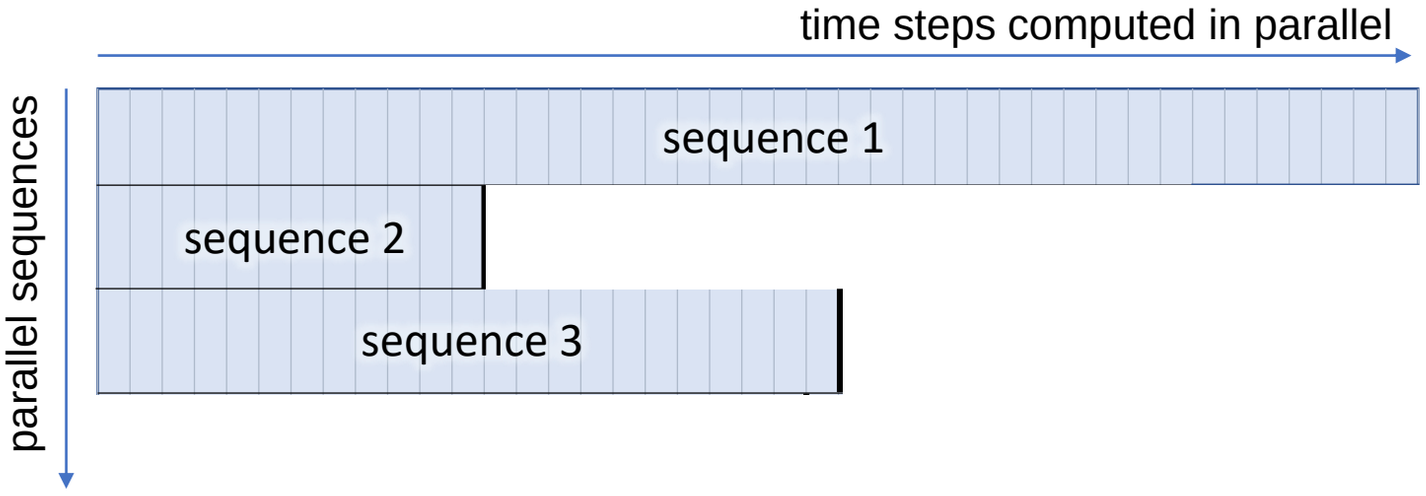



```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

Microsoft

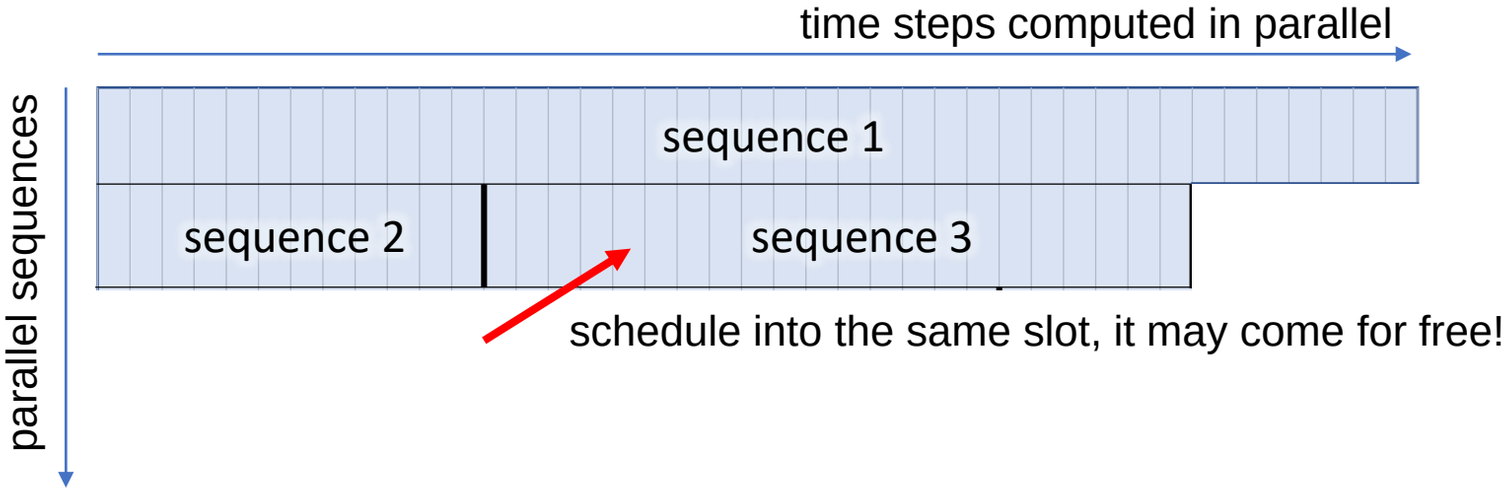
Variable length sequences

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



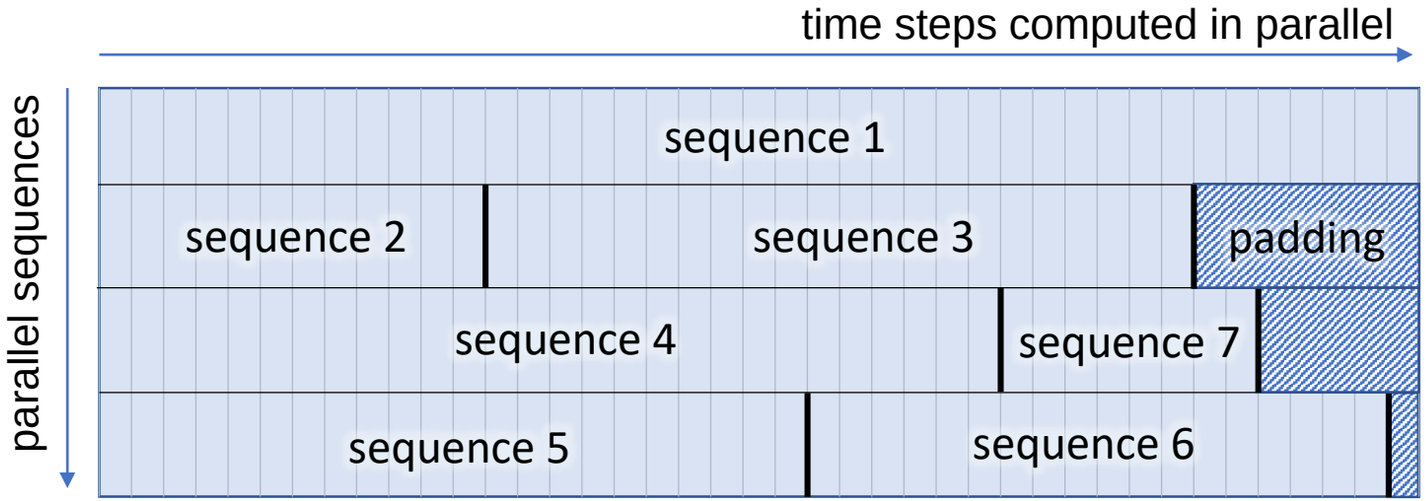
Variable length sequences

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



Variable length sequences

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



CNTK model update difference

- The parameters are specified in a way that is agnostic of minibatch size
- By default SGD with momentum uses Unit Gain
  - Standard SGD
    - $x = x - \lambda \nabla x$
  - SGD with momentum
    - $v = \mu v - \lambda \nabla x$
    - $x = x + v$
- By default gradient is sum not average
  - `use_mean_gradient`

CNTK model update difference

- The parameters are specified in a way that is agnostic of minibatch size
- By default SGD with momentum uses Unit Gain
  - Standard SGD
    - $x = x - \lambda \nabla x$
  - SGD with momentum
    - $v = \mu v - \lambda \nabla x \longrightarrow v = \mu v - (1 - \mu)\lambda \nabla x$
    - $x = x + v$
- By default gradient is sum not average
  - `use_mean_gradient`

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

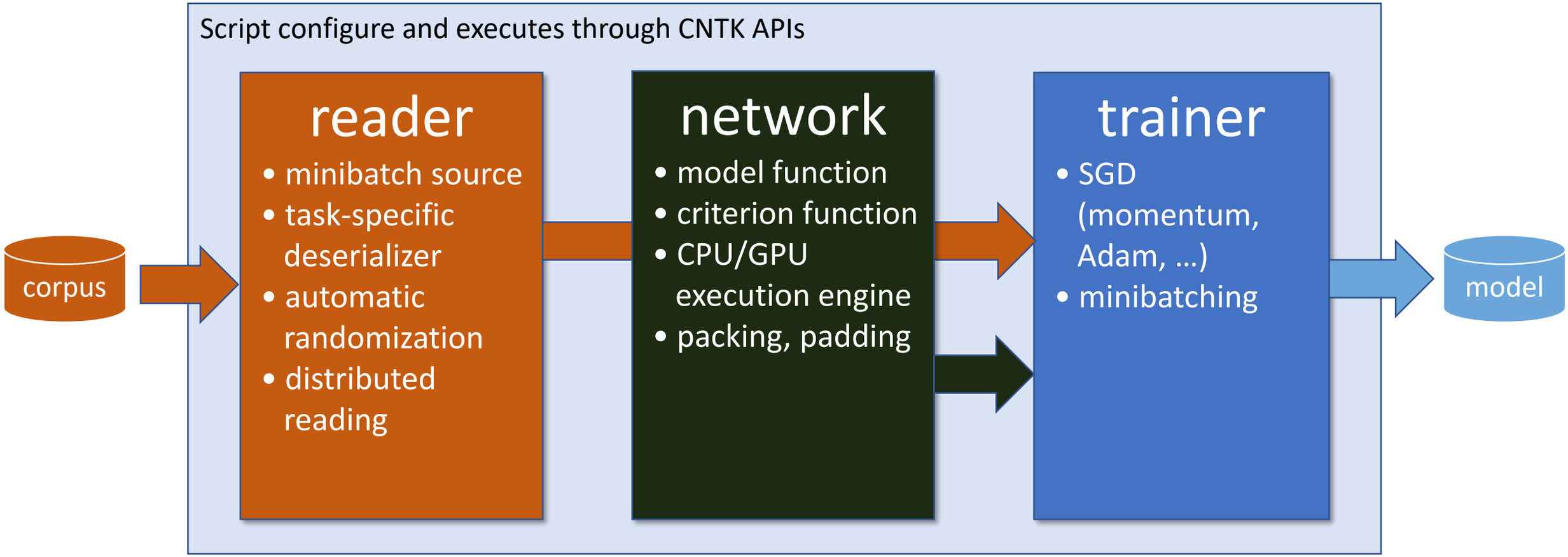
- Automatic MB sizing [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "ON Parallelizability of Stochastic Gradient Descent...", ICASSP 2014]
- Block momentum [K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training...," ICASSP 2016]
  - Very recent, very effective parallelization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

Asynchronous training

- DC-ASGD: [Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z. M., & Liu, T. Y.: "Asynchronous Stochastic Gradient Descent with Delay Compensation", ICML2017]

Training workflow

Training workflow



Training

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

```
python my_cntk_script.py
```

Distributed training

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

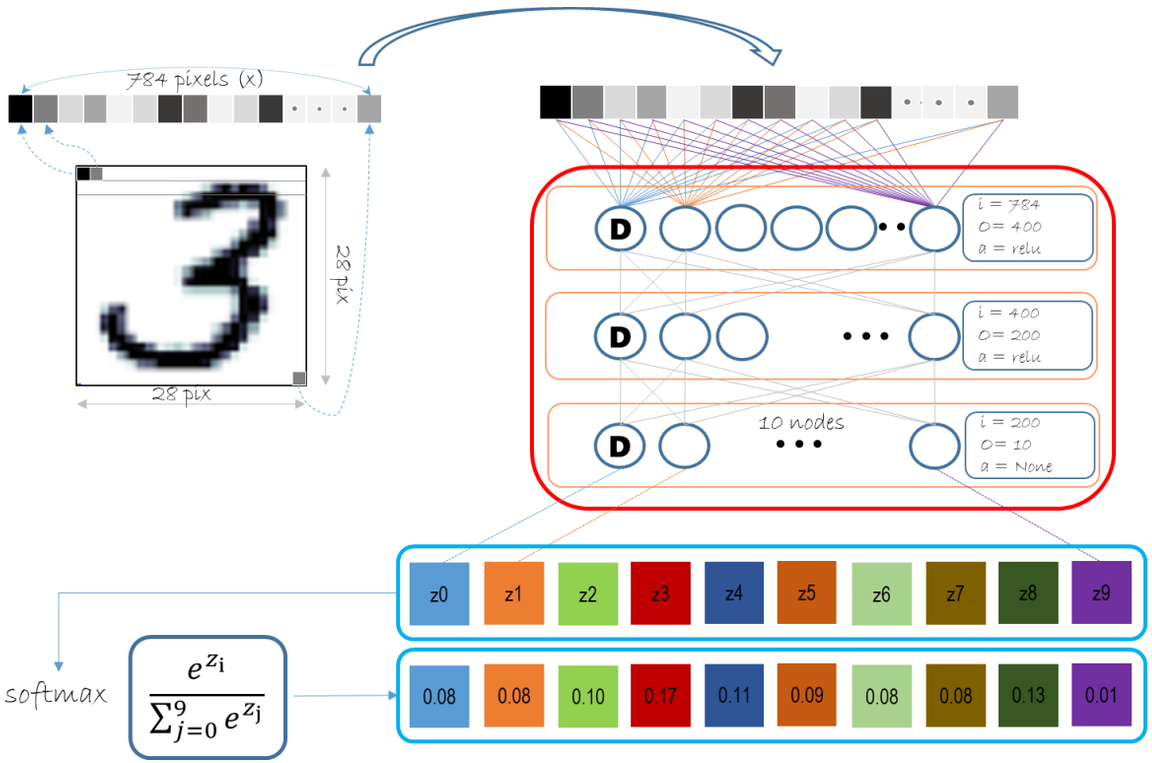
```
mpiexec --np 16 --hosts server1,server2,server3,server4    \
python my_cntk_script.py
```

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

Multi-Layer Perceptron



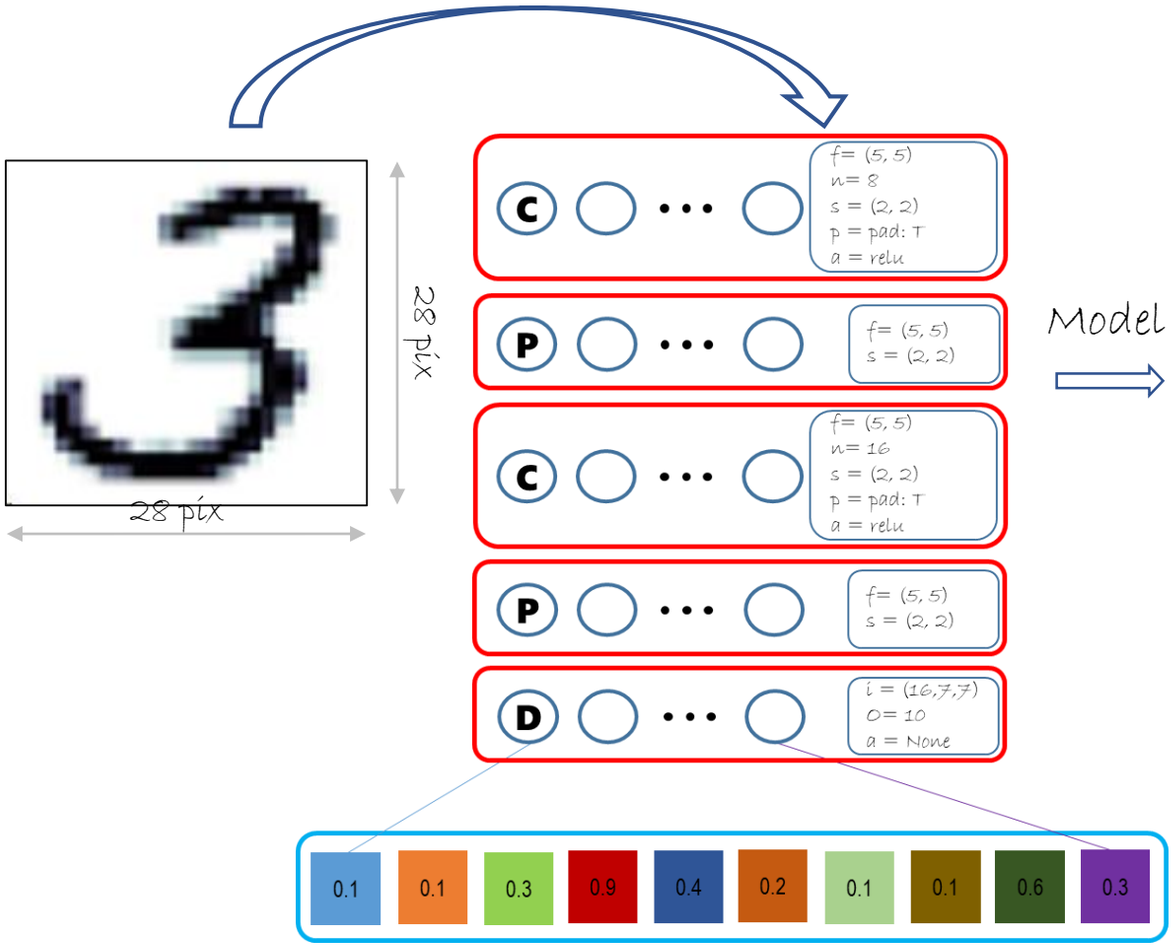
Model

```
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)
```

Convolution Neural Network



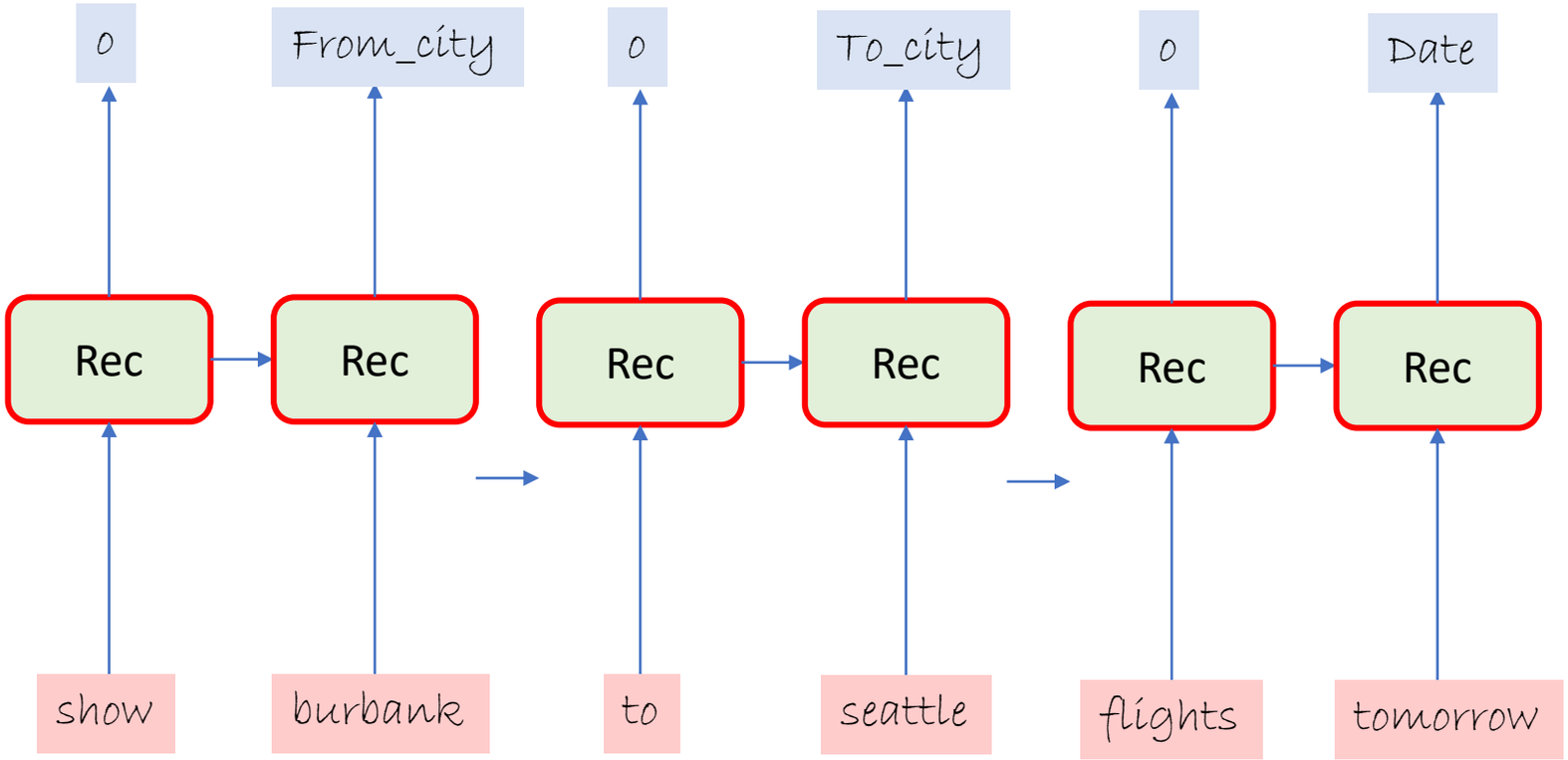
Model

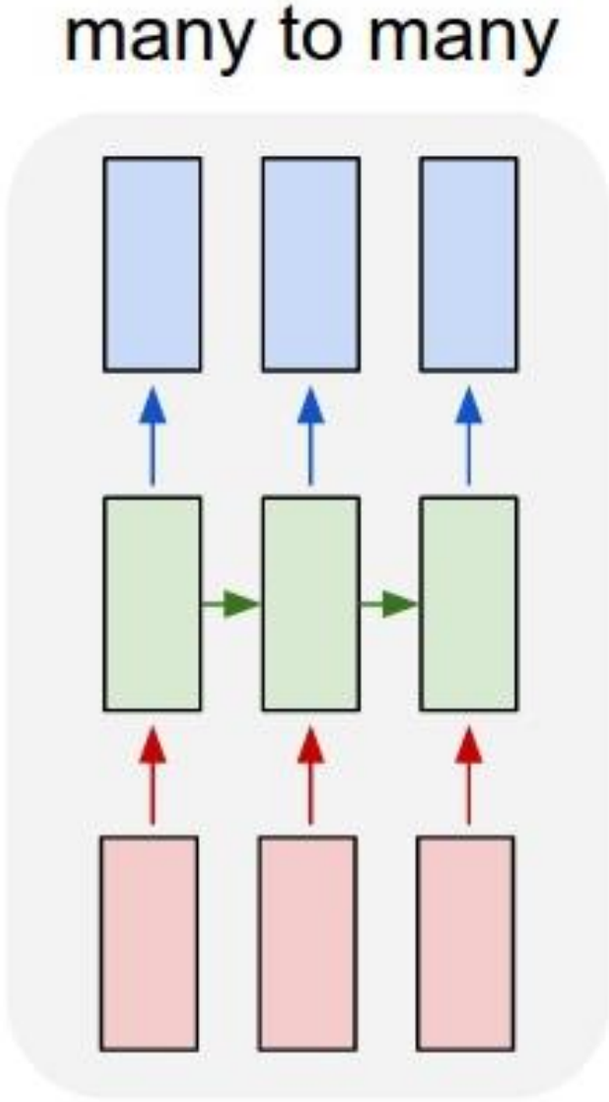
```
z = model(x):
    h = Convolution2D((5,5),filt=8, …)(x)
    h = MaxPooling(…)(h)
    h = Convolution2D ((5,5),filt=16, …)((h)
    h = MaxPooling(…)(h)
    r = Dense(output_classes, act= None)(h)
    return r
```

Microsoft

Sequence (many to many)

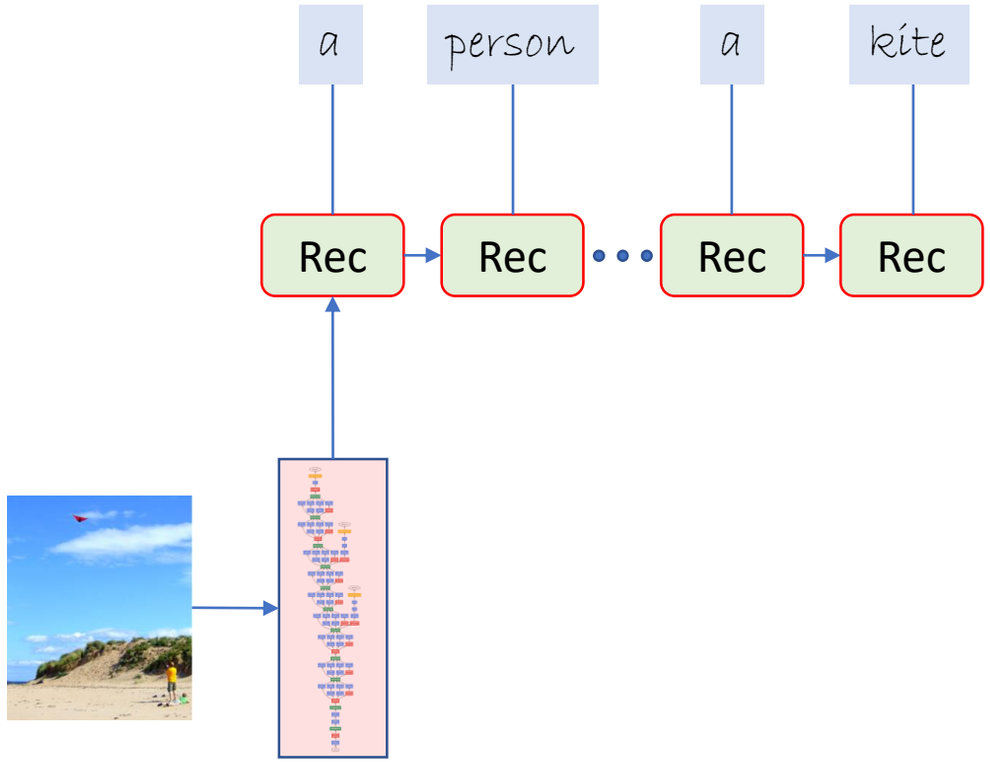
Problem: Tagging entities in Air Traffic Controller (ATIS) data

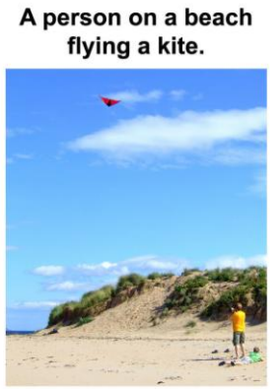


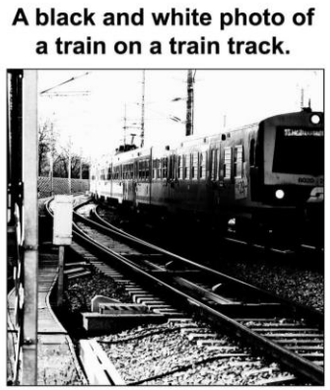


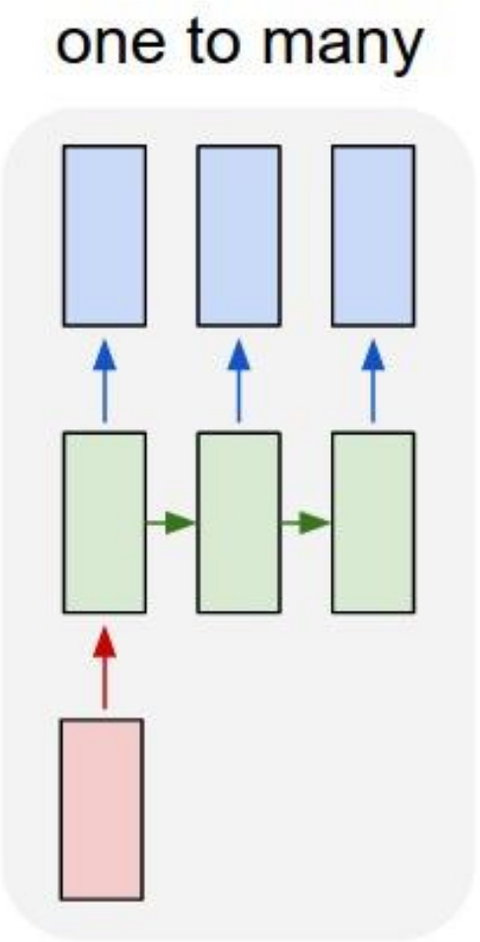
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sequence (one to many)





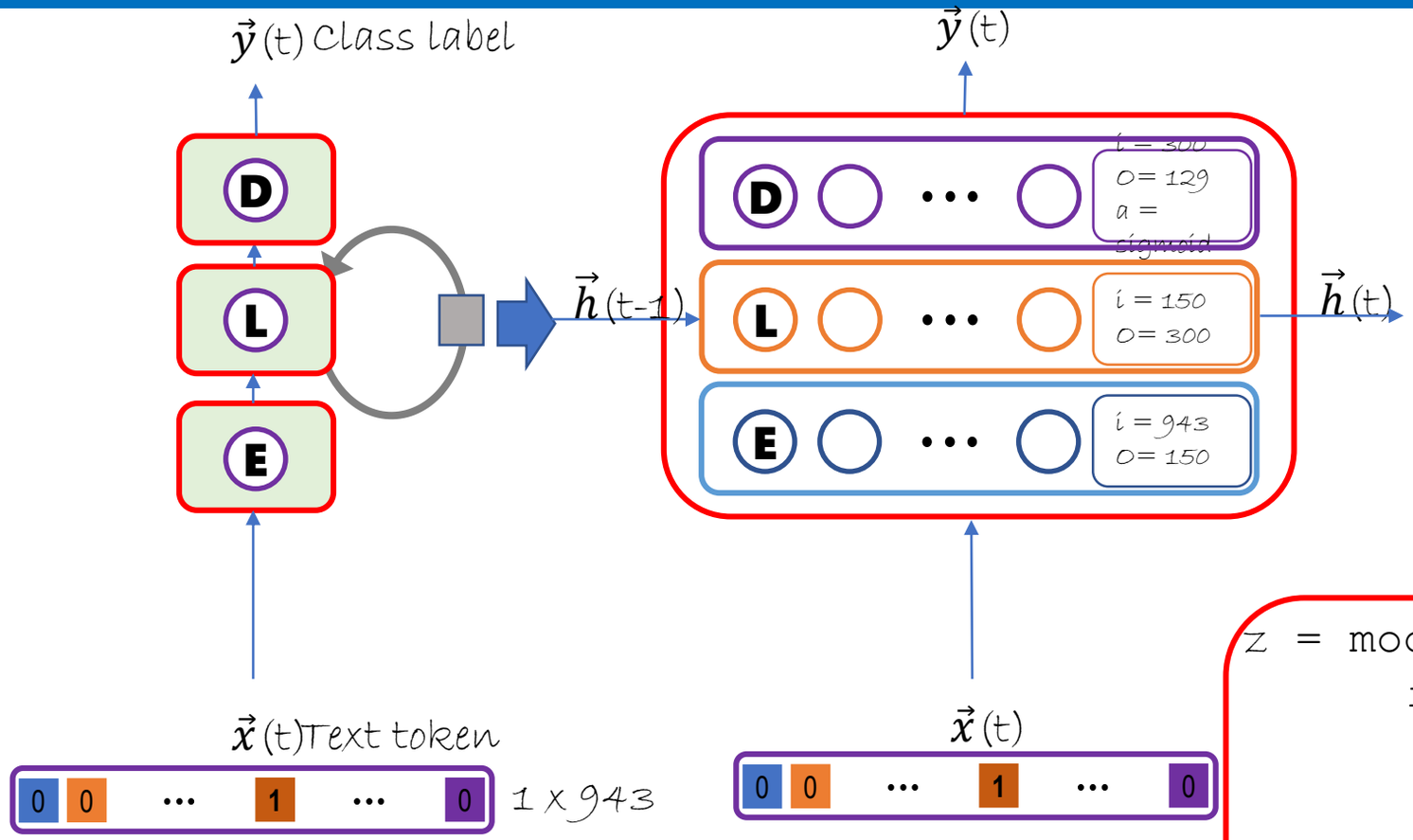




http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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

Recurrent Neural Network




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

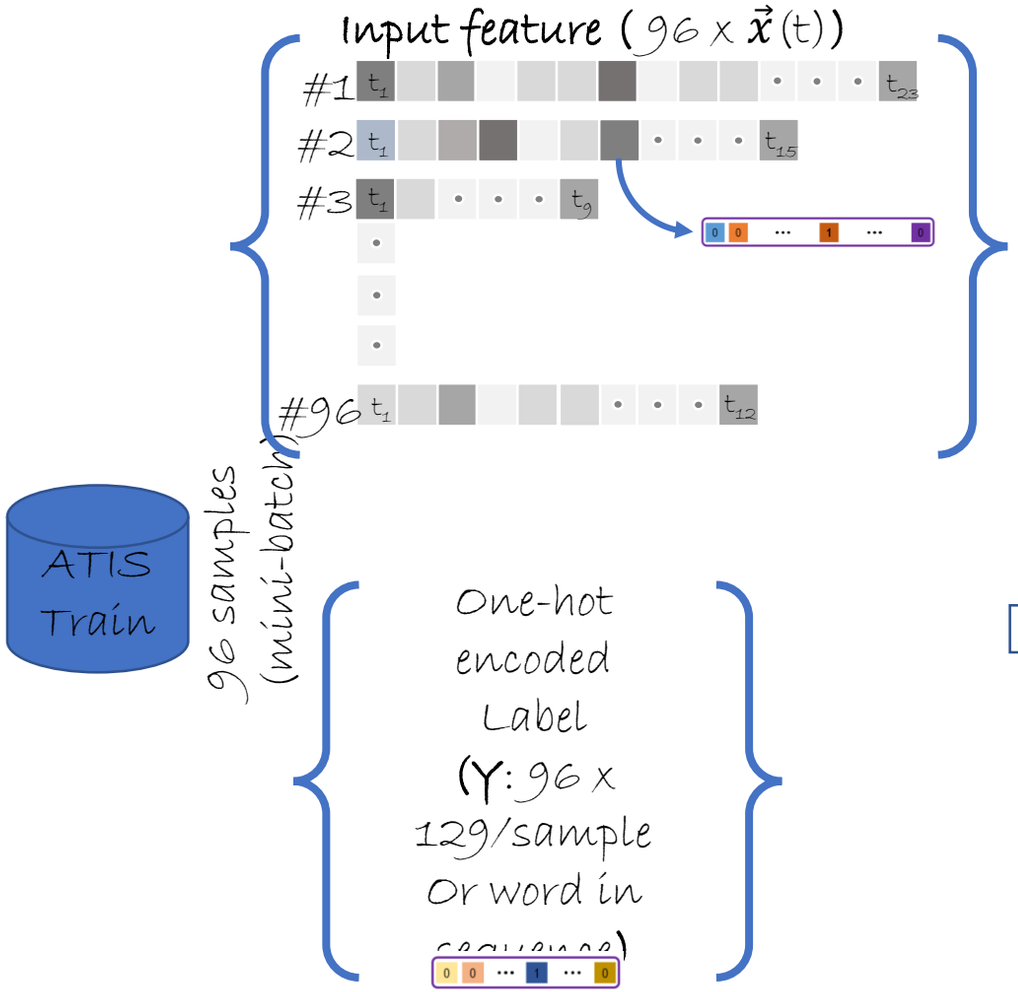
Microsoft

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],
                        minibatch_size=C.learners.IGNORE, epoch_size=100)
sgd_learner = C.sgd(z.parameters, lr_schedule)
```

Train workflow




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

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})
```

Microsoft

Models via layer APIs

FeedForward models

Take away

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

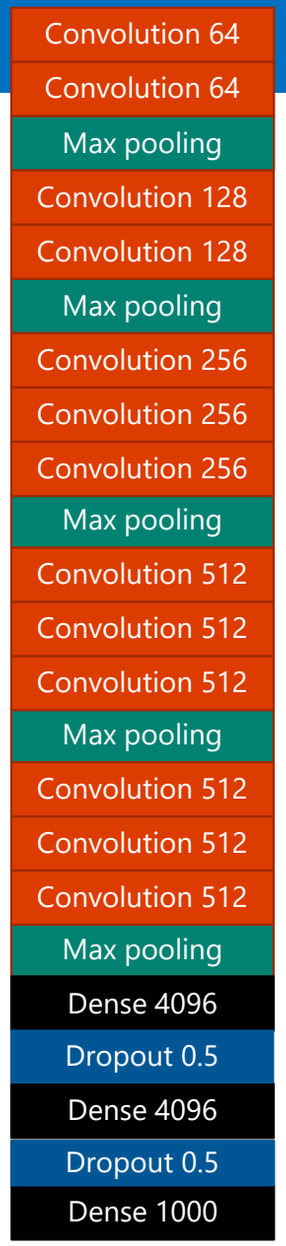
Visual Geometry Group (VGG network)

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)
```



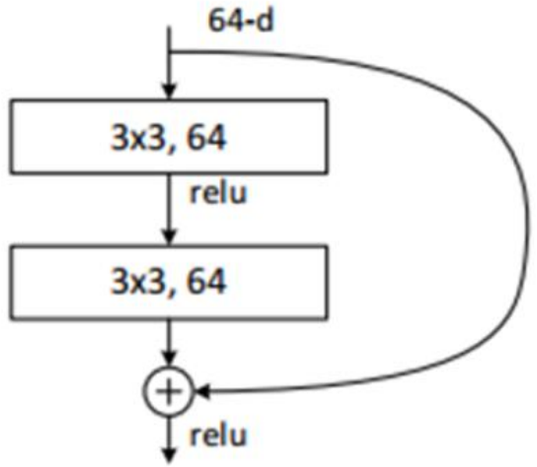
Residual Network (ResNet)

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
```

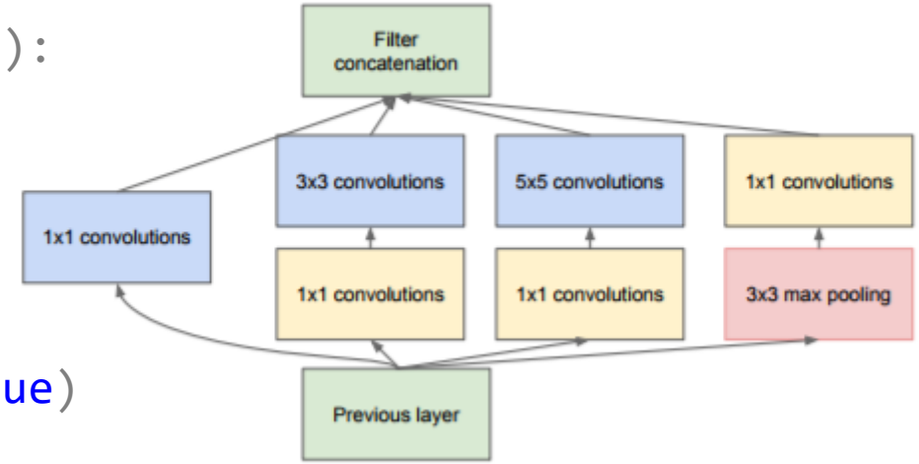
Inception Network (GoogleNet)

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 (GoogleNet)

```
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

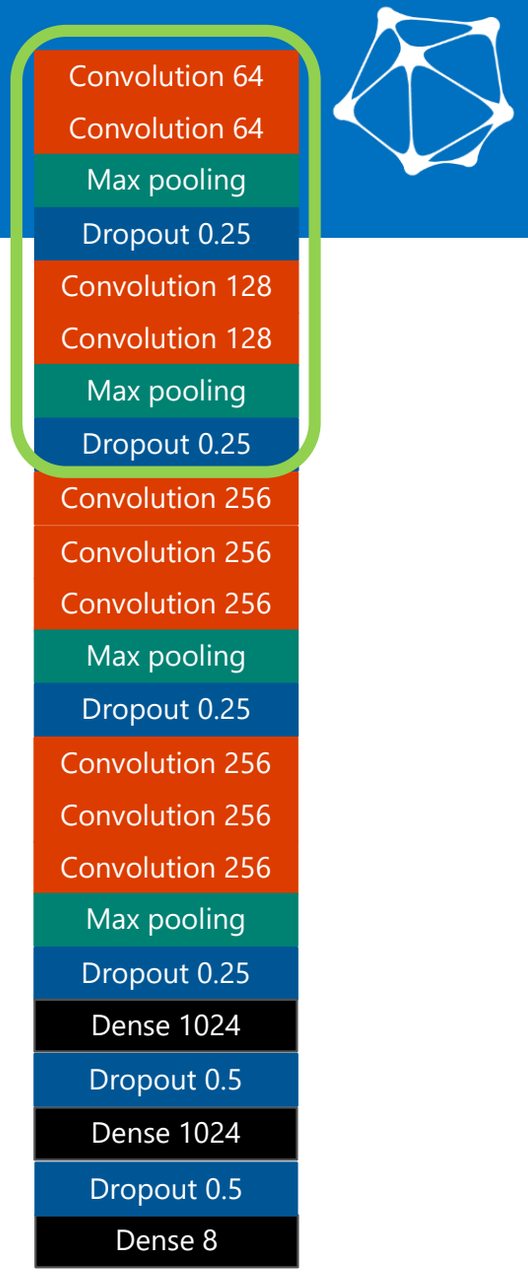
https://github.com/Microsoft/FERPlus

Emotion Recognition

- 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.

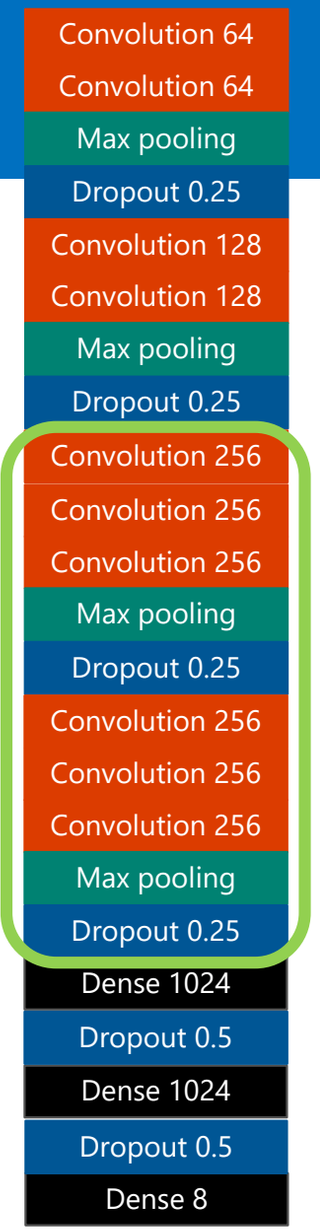
Emotion Recognition

```
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)])
```



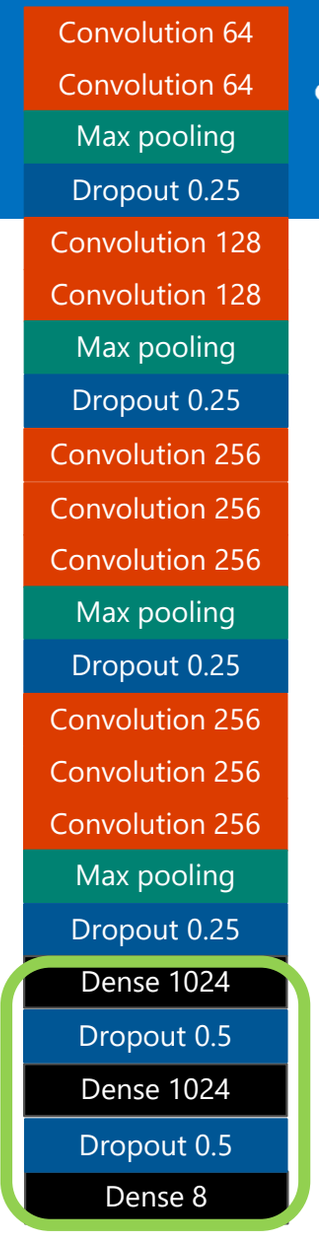
Emotion Recognition

```
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 Recognition

```
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

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

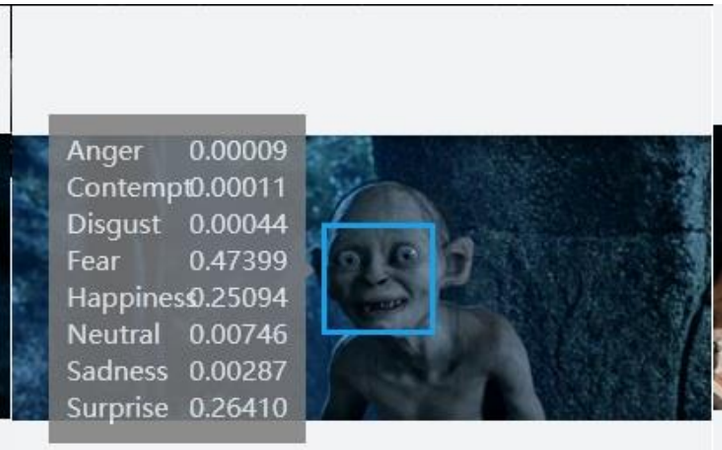
```
train_loss = -C.reduce_sum(C.element_times(target, C.log(prediction)))
```

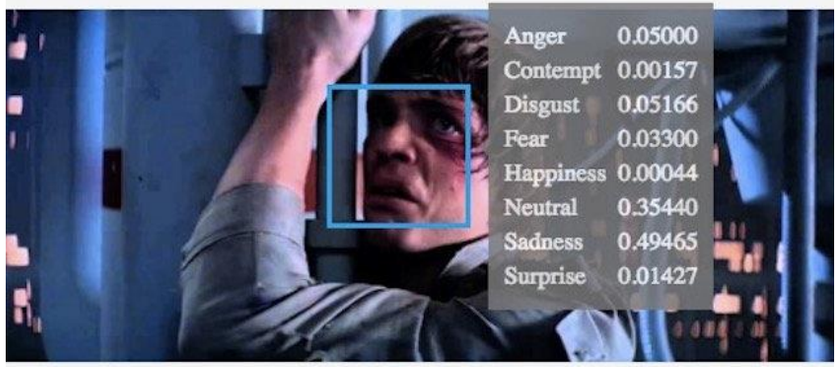
- Multi-label learning

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

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

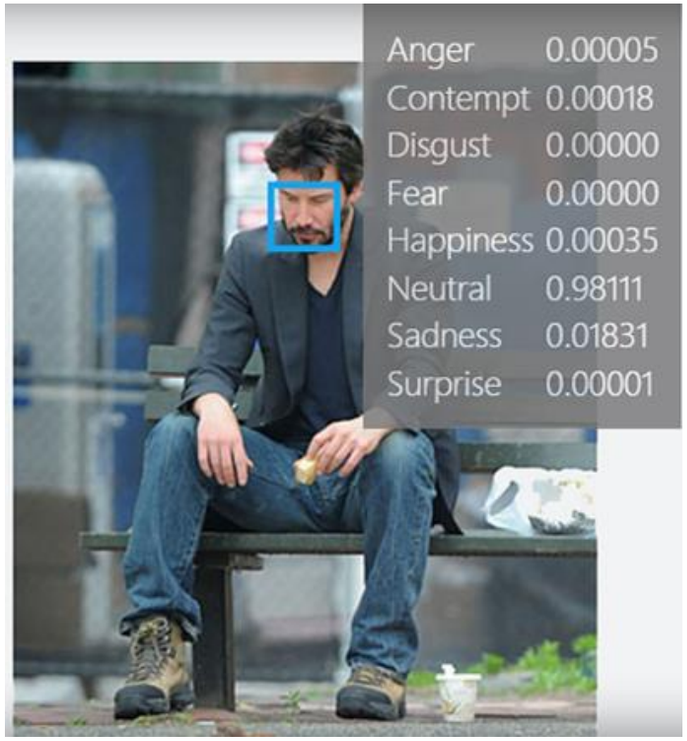
Examples





Microsoft

Examples



“Microsoft thinks Sad Keanu is only 0.01831 sad”

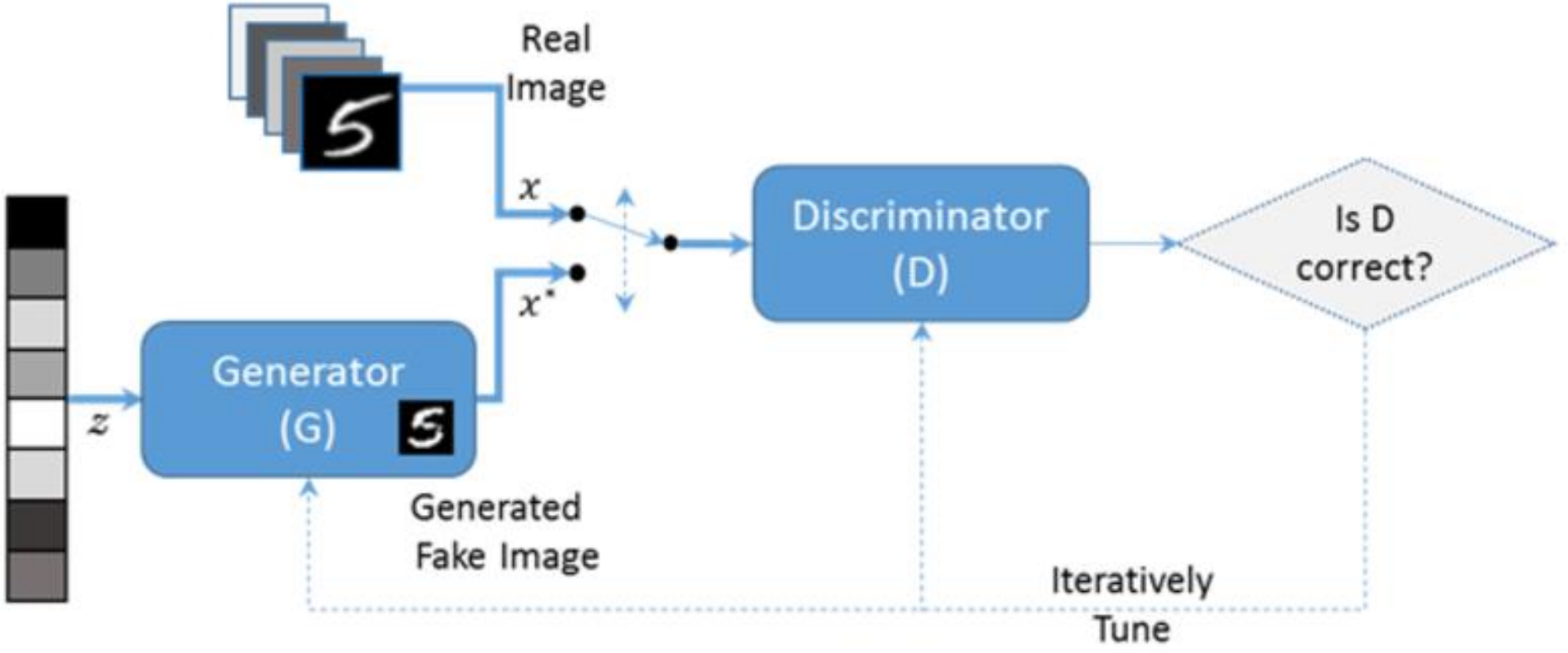
“According to Microsoft's Emotion API, Sidney Crosby was rather angry about scoring the goal that won Canada a gold medal at 2010's Olympics in Vancouver.”

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)



Generative Adversarial Networks (GAN)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

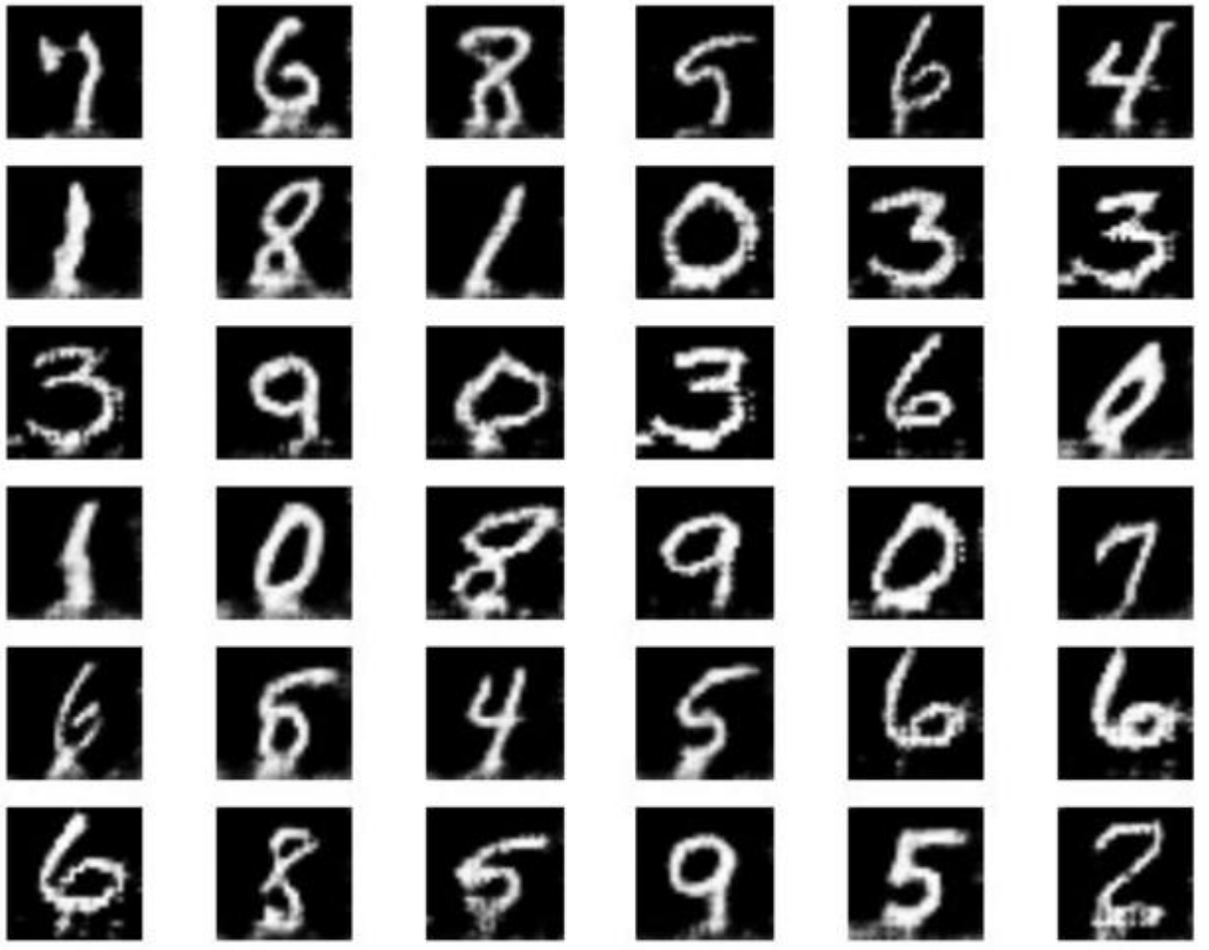
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Generative Adversarial Networks (GAN)

```
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, minibatch_size=C.learners.IGNORE),
                   momentum=C.momentum_schedule(momentum))

d_learner = C.adam(parameters=d_real.parameters,
                   lr=C.learning_rate_schedule(lr, minibatch_size=C.learners.IGNORE),
                   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)
```

Generative Adversarial Networks (GAN)



Video Classification: 3D Convolution

Video Classification

Two main problems:

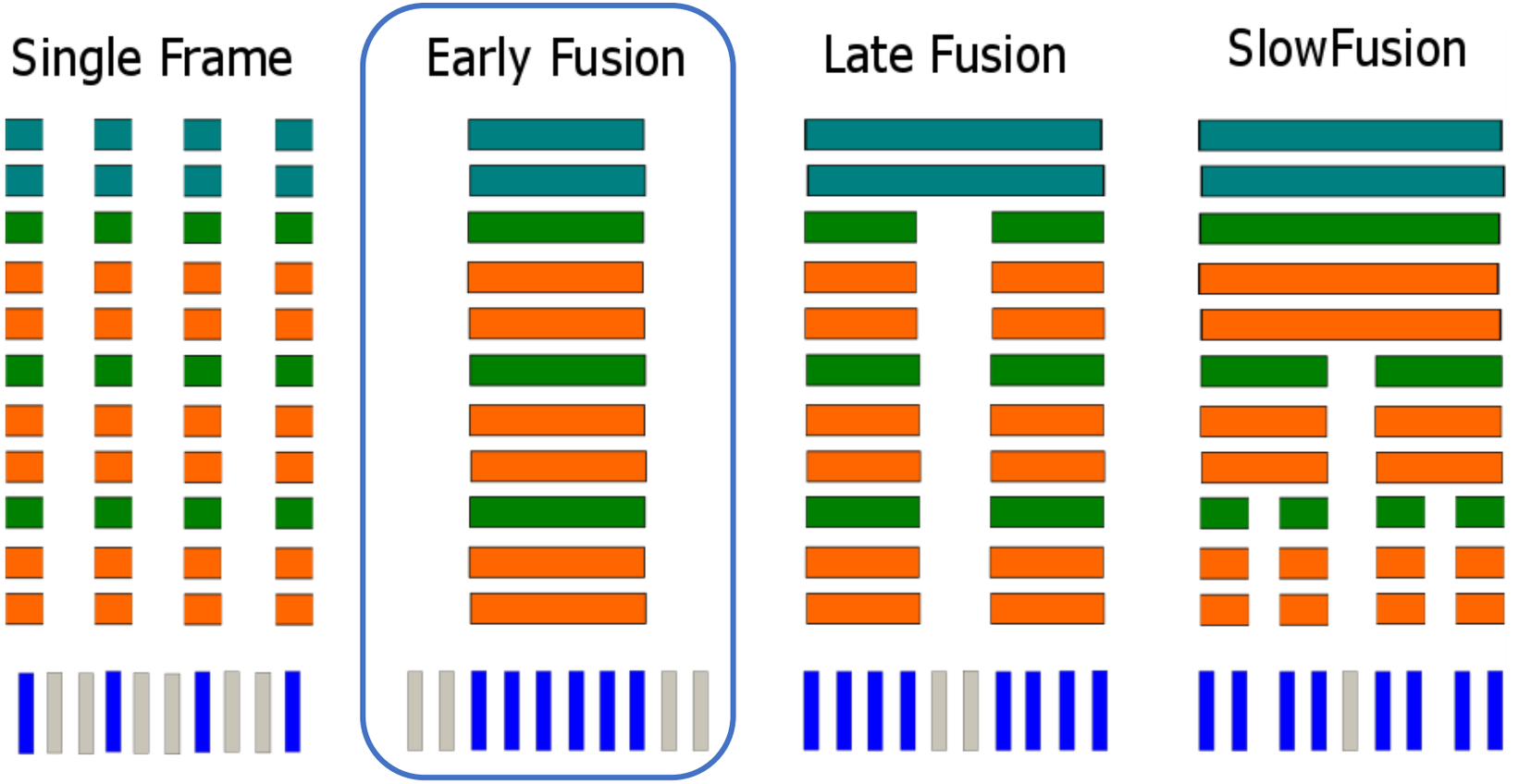
- Action classification
  - Input: a trimmed video clip with a single action.
  - Output: classify the action in the clip.
- Action detection
  - Input: an untrimmed video clip with multiple actions and possible no action.
  - Output: location of each action and its corresponding classification.

Video 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.

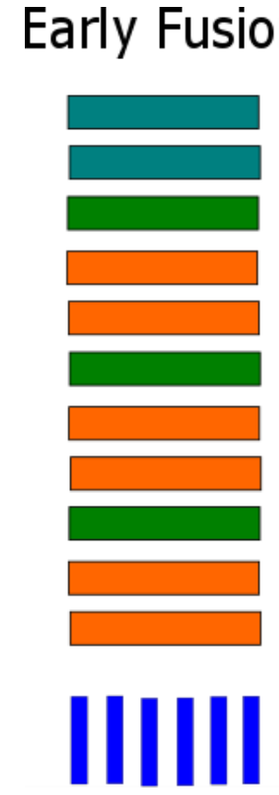
Video Classification



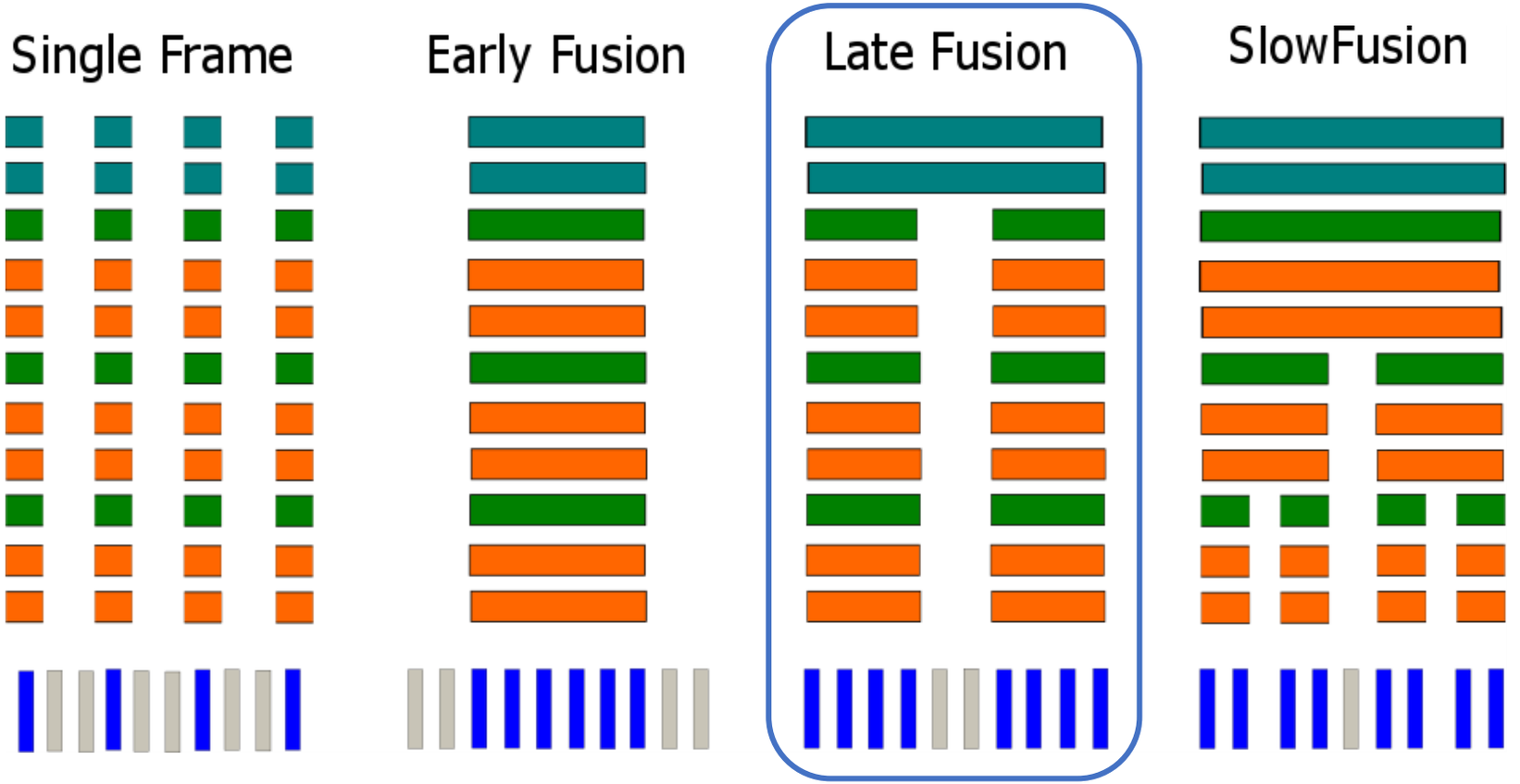
[Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014]

Video Classification

```
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)
```



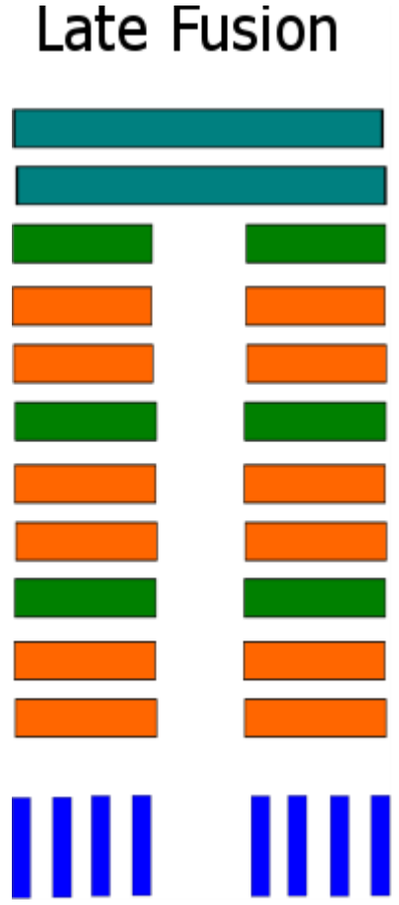
Video Classification



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

Video Classification

```
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))
```



Video Classification: RNN

Video Classification

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

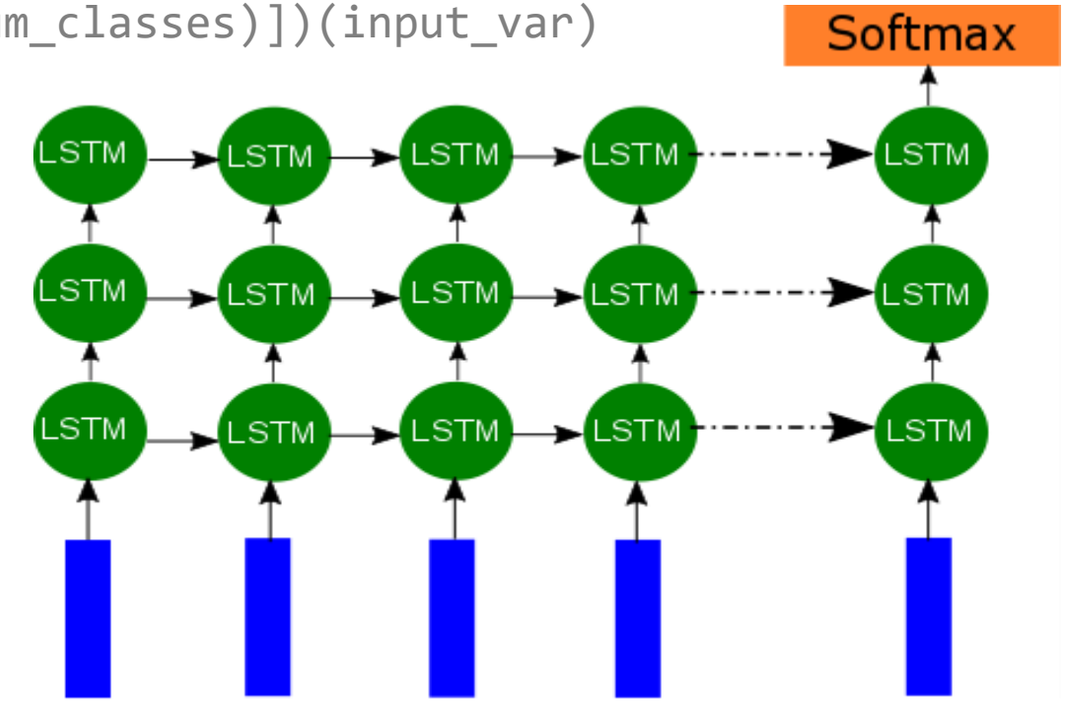
Video Classification

- 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>)
```

Video 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)
```



Video Classification

```
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))
```

Customization

Custom Layer using CNTK expression

```
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))
```

Custom Layer using 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)]
```

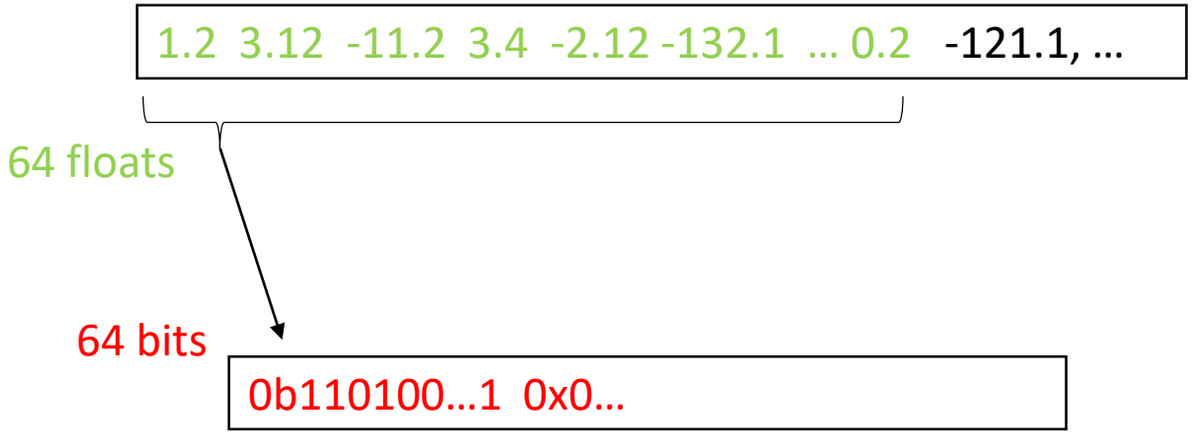
Custom Learner

```
def my_rmsprop(parameters, gradients):
    rho = 0.999
    lr = 0.01

    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):
        accum_new = cntk.assign(a, rho * a + (1-rho) * g * g)
        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)
```

Inference performance

Binarization [Bengio et al. 2015/16]

- Quantize floats to +/-1
- 1.122 * -3.112 ==> 1 * -1
- Notice:
  - 1 * 1 = 1
  - 1 * -1 = -1
  - -1 * 1 = -1
  - -1*-1 = 1
- Replacing -1 with 0, this is just XNOR
- Retrain model to convergence



A[:64] . W[:64] == popc($A_{/64}$ XNOR $W_{/64}$)

1-bit Matrix Operations

```
float x[], y[], w[];
...
for i in 1…N:
    y[j] += x[i] * w[i];
```

2N ops

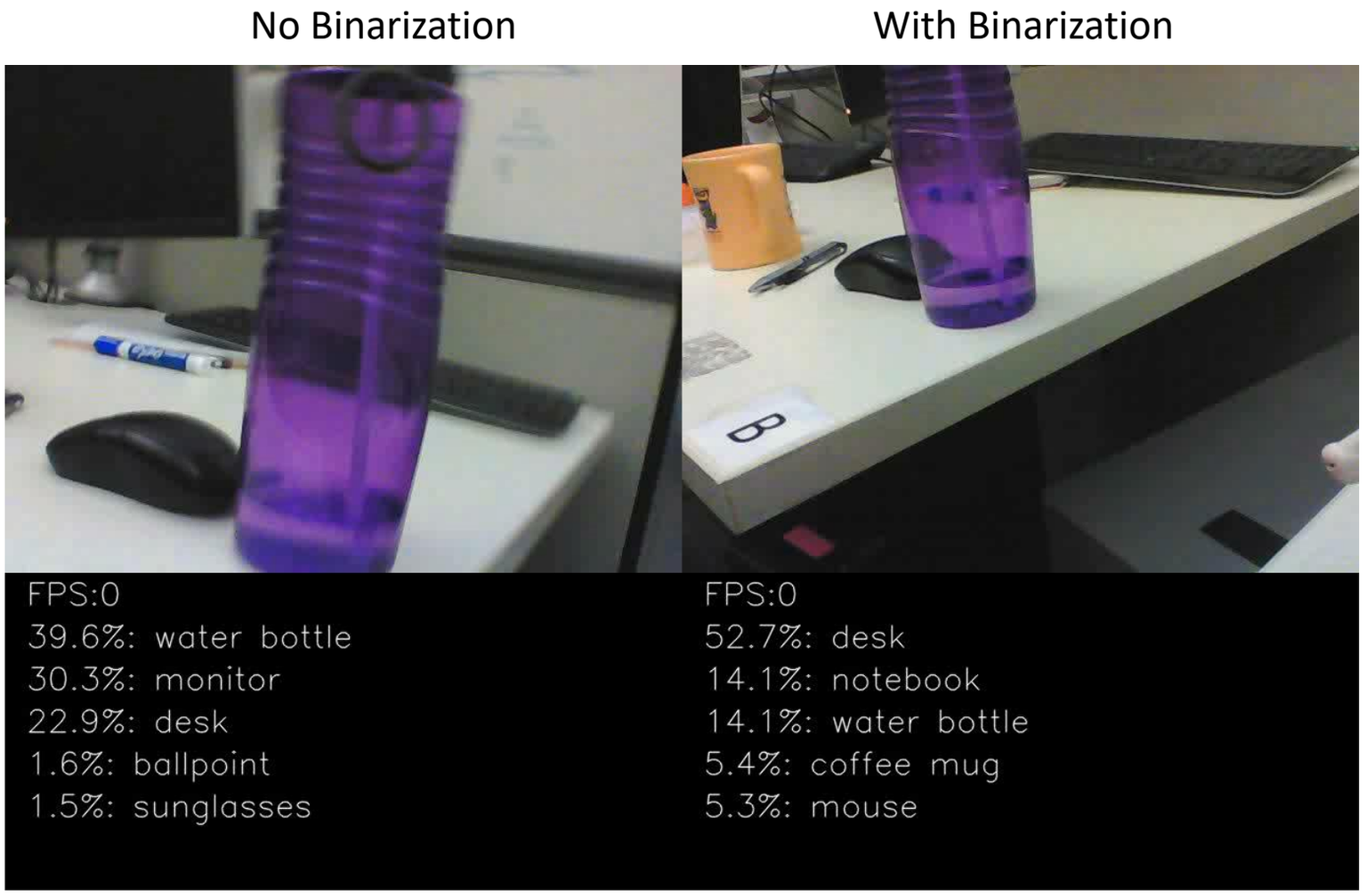
```
unsigned long x[], y[], w[];
…
for i in 1…N/64:
    y[j] += 64 – 2*popc(not(x_b[i] xor w_b[i]));
```

3N/64 ops

~40x fewer ops

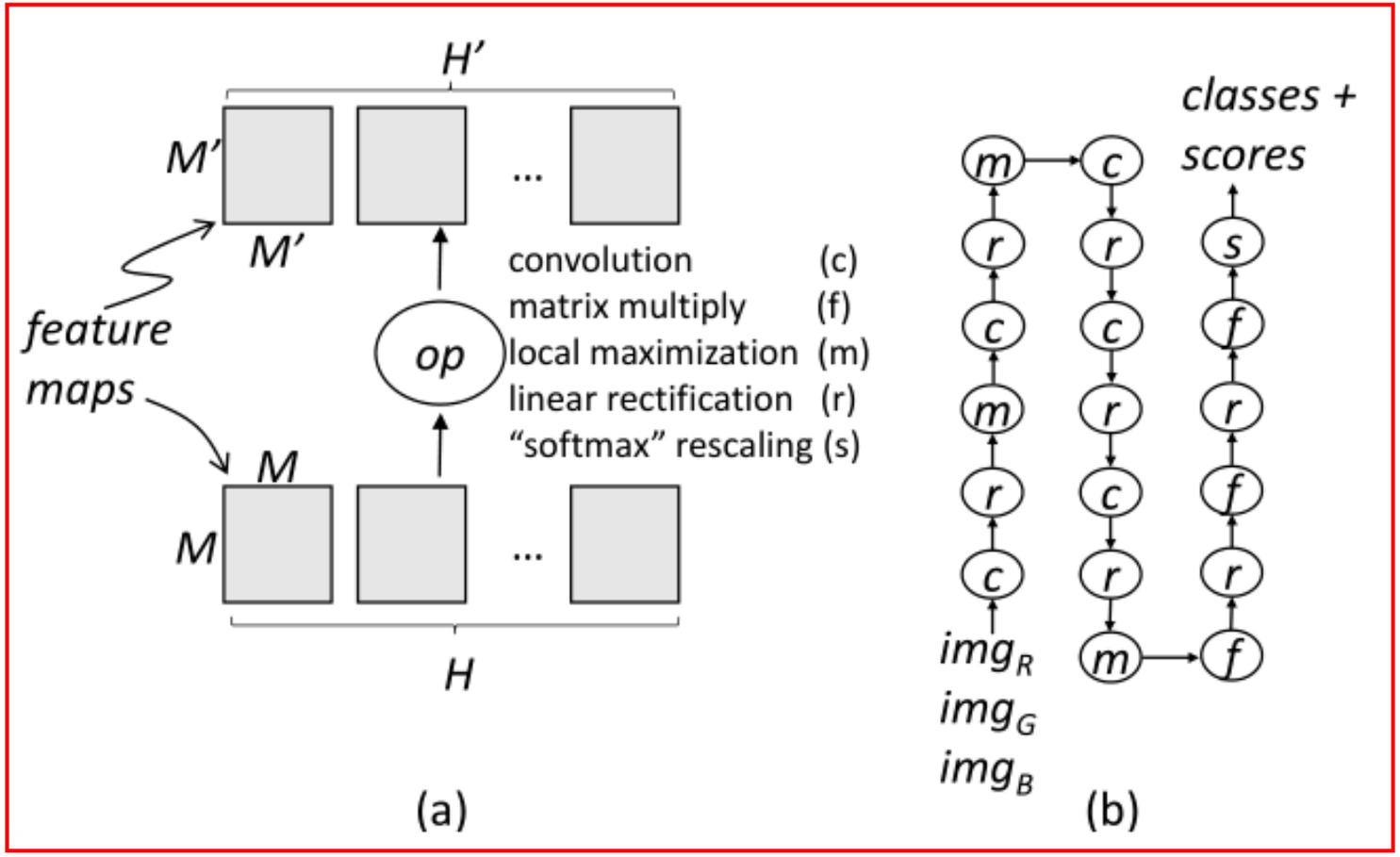
32x smaller

Object Recognition on RaspberryPi3



- Quadcore Cortex A53 ARM CPU @ 2W
- 8.1x end-end speedup
- No noticeable accuracy loss

DNN are Statistically-specified linear algebra (SSLAs)



+ training/test dataset

+ cost function

$sum_i (label_i - y_l)^2$

Microsoft

Recipe for Optimizing SSLAs

Do until enough "good" versions obtained:

1. Transform network to make it cheaper*.
2. Retrain the network.
3. If speed/accuracy on test dataset is good enough, add result to a *catalog*.

**Even if network accuracy collapses!*

Sample transformations (#instructions reduced)

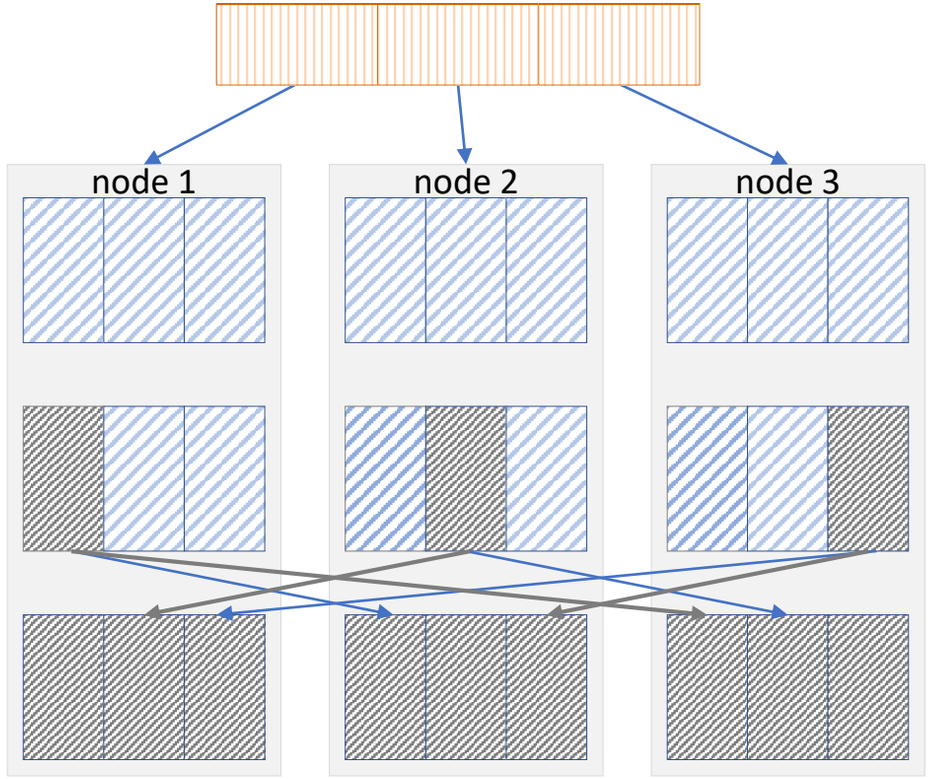
- **30x** [Prune] Reduce layers, parameters
- **9x** [Separate] Replace 3-D convolution with 2-D + 1-D
- **3x** [Factorize] Replaces matrices by low-rank factors
- **2x** [FFT-ize] Move to frequency domain
- **35x** [Dictionarize] Replace ops by lookups
- **32x** [Quantize] Replace floats by bits
- … many more in the learning literature

Distributed training

Data-Parallel training

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

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



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

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

Asynchronous training

- DC-ASGD: [Zheng, S., Meng, Q., Wang, T., Chen, W., Yu, N., Ma, Z. M., & Liu, T. Y.: "Asynchronous Stochastic Gradient Descent with Delay Compensation", ICML2017]

Data-Parallel training

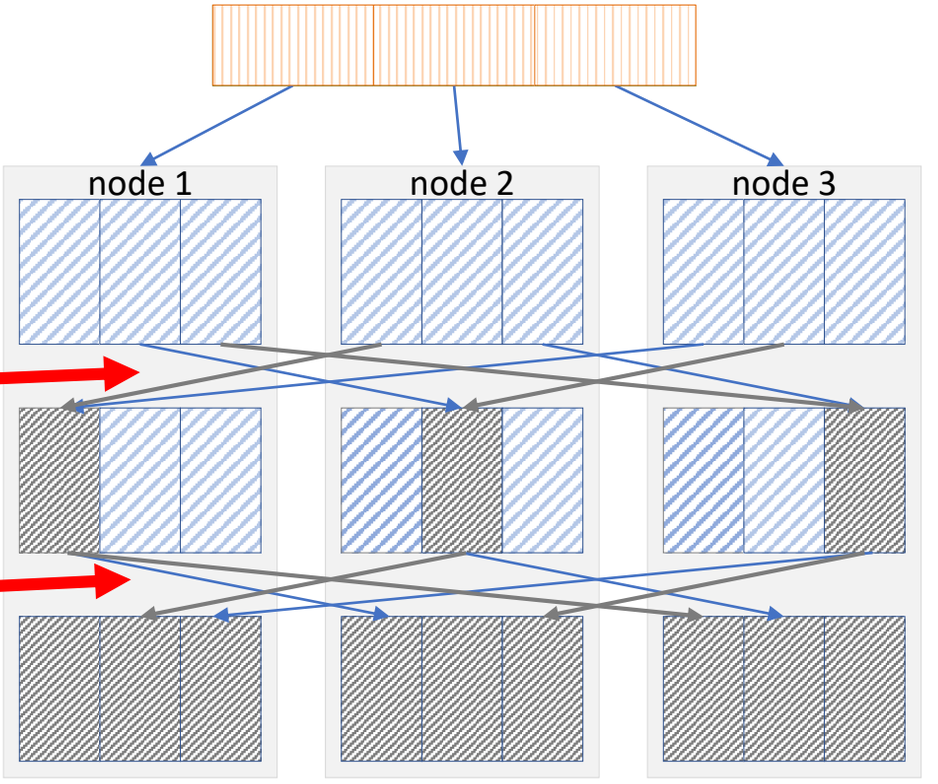
How to reduce communication cost:

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



Data-Parallel training: 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]

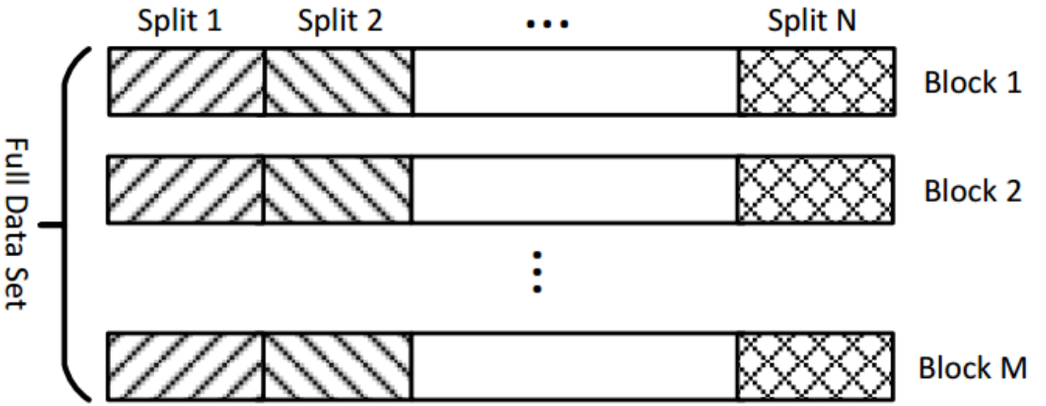
Block Momentum

- 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

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

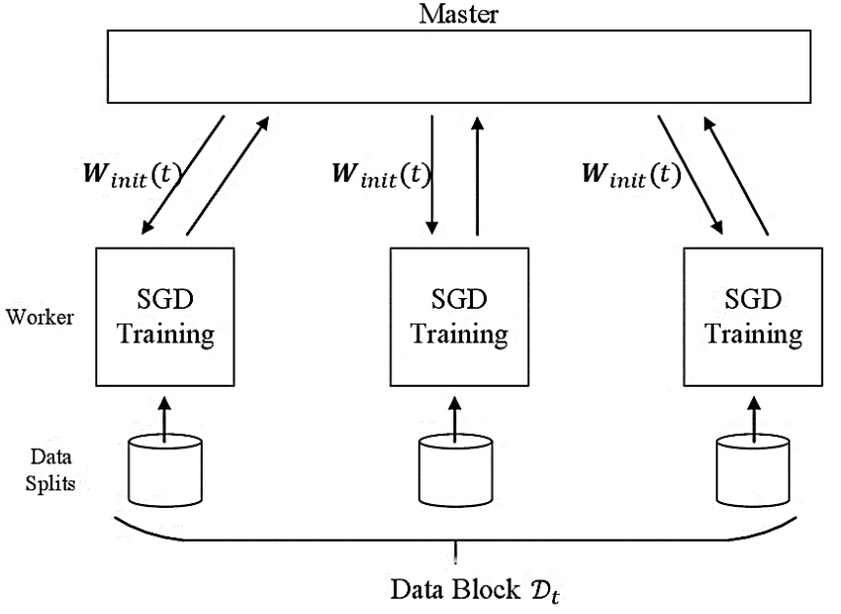


→Training dataset is processed block-by-block

→**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)

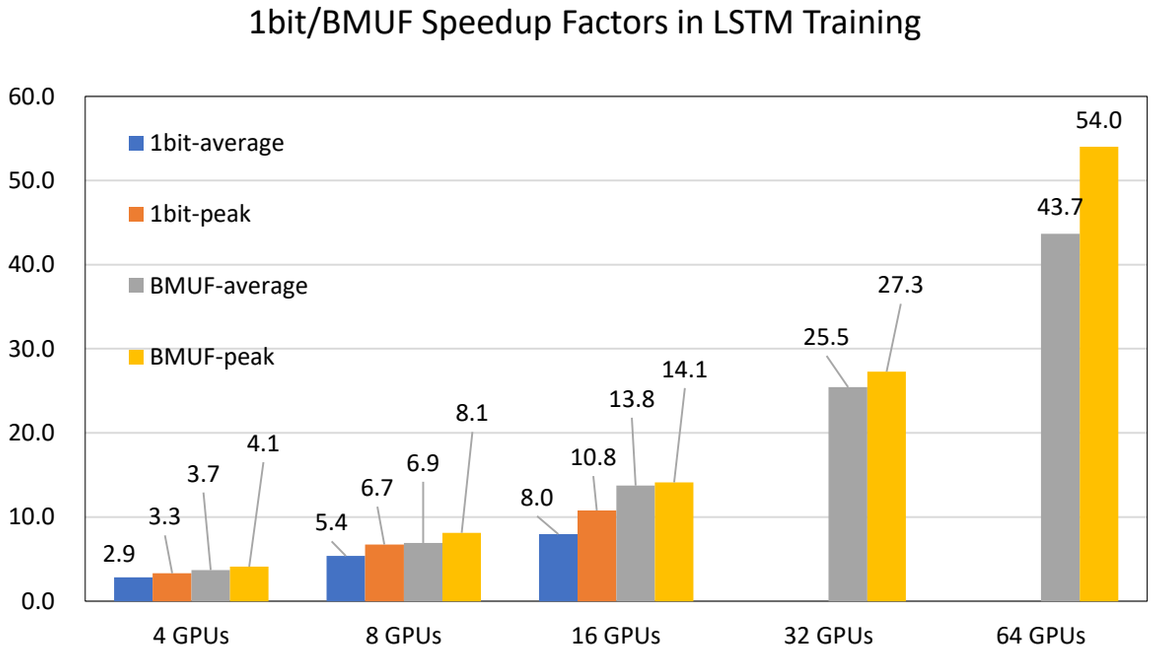
$$\boldsymbol{W}_{init}(t+1) = \boldsymbol{W}(t) + \eta_{t+1} \cdot \boldsymbol{\Delta}(t)$$

Block Momentum

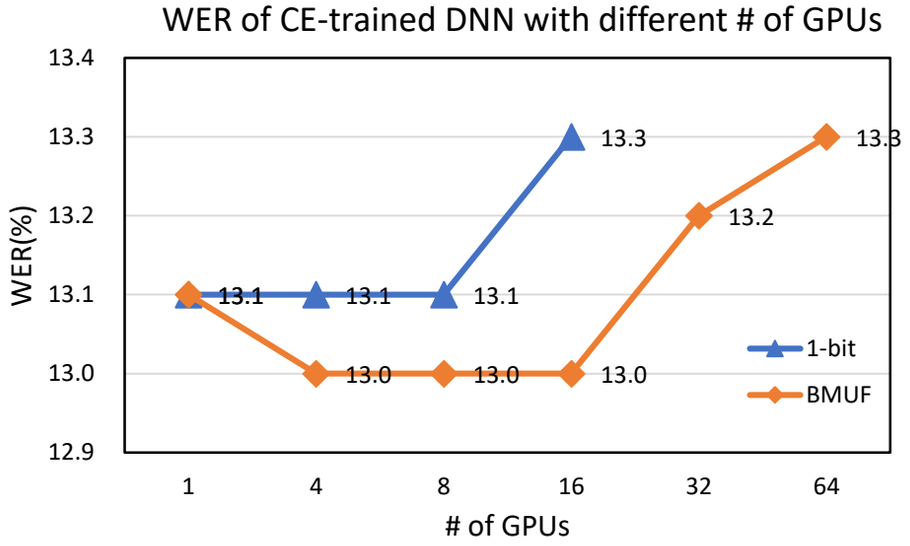
- Repeat IBPO and BMUF until all data blocks are processed
  - So-called “one sweep”
- Re-partition training set for a new sweep, repeat the above step
- Repeat the above step until a stopping criterion is satisfied
  - Obtain the final global model $\boldsymbol{W}_{final}$

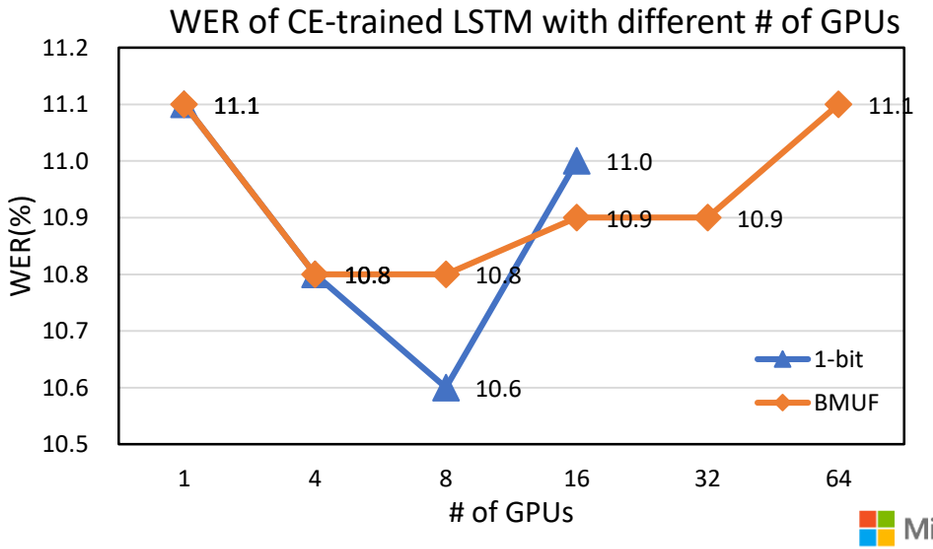
Comparison

- 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



Credit: Yongqiang Wang, Kai Chen, Qiang Huo





Microsoft

Results

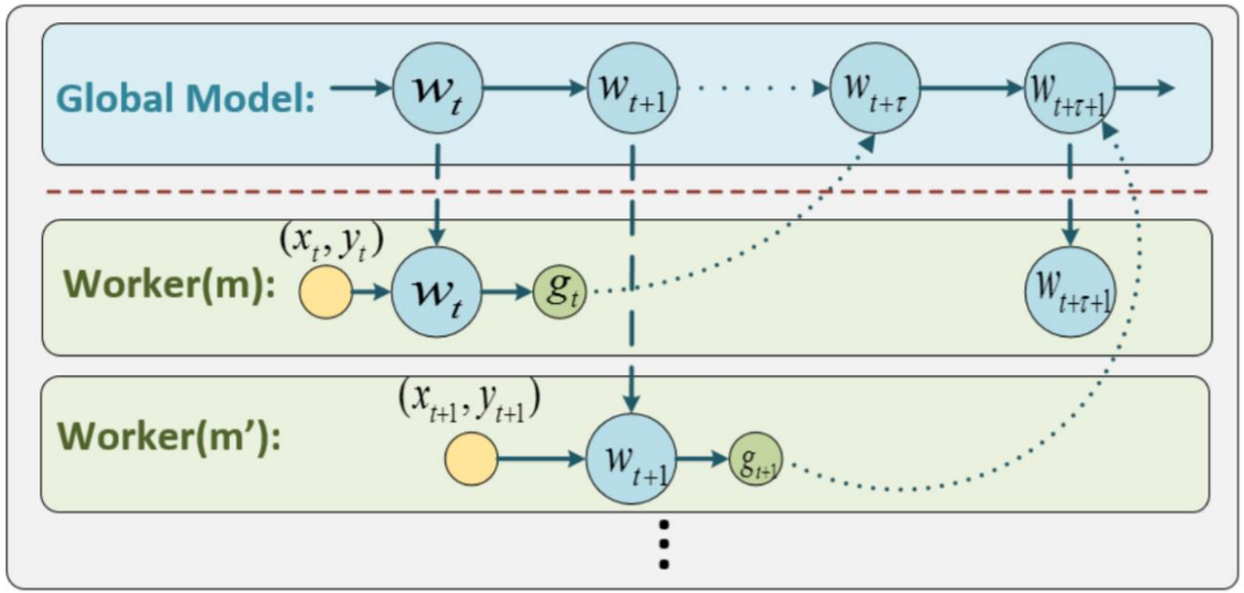
- Almost linear speedup without degradation of model quality
- Verified for training DNN, CNN, LSTM up to 64 GPUs for speech recognition, image classification, OCR, and click prediction tasks
- Used for enterprise scale production data loads
- Production tools in other companies such as iFLYTEK and Alibaba

Block Momentum

```
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)
```

Asynchronous Algorithm

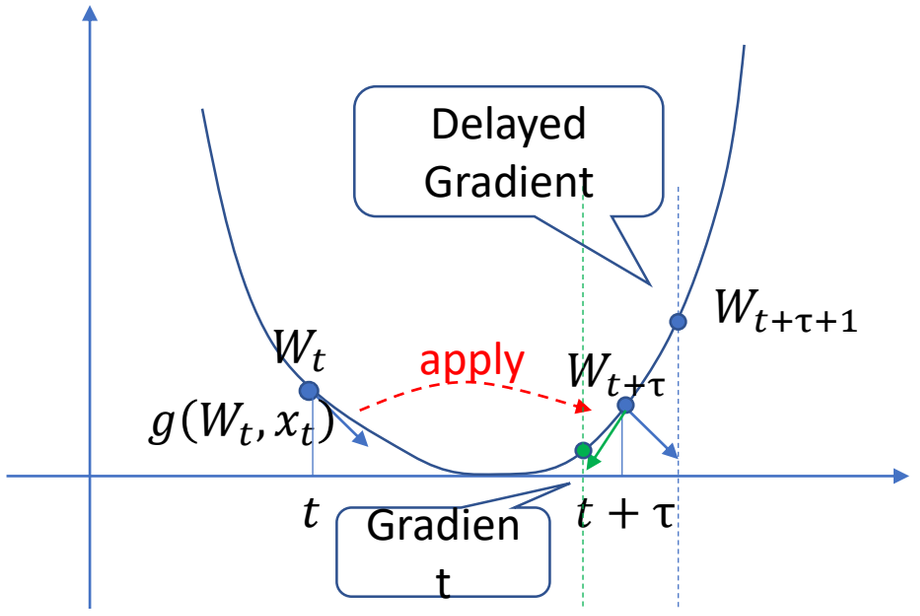
- Asynchronous SGD
  - No more all-reduce, no more barrier, no more wait. Linear speed-up.



Asynchronous Algorithm

- Problem in Asynchronous SGD.
  - Delayed Gradient.
    - SGD
      - $W_{t+\tau+1} = W_{t+\tau} - \eta * g(W_{t+\tau}, x_t)$
    - Async SGD
      - $W_{t+\tau+1} = W_{t+\tau} - \eta * g(W_t, x_t)$

$$g(W_{t+\tau}, x_t) \neq g(W_t, x_t)$$



Reducing delay

$$g(W_{t+\tau}) \neq g(W_t)$$

- Taylor Expansion at $g(W_t)$

$$g(W_{t+\tau}) = g(W_t) + \nabla g(W_t) \cdot (W_{t+\tau} - W_t) + \frac{1}{2}(W_{t+\tau} - W_t)^T \boldsymbol{H} g(W_t) \cdot (W_{t+\tau} - W_t) + \mathrm{O}(||W_{t+\tau} - W_t||^3)$$

Delay Compensated Gradient: 1st Order Approximation

$$g(W_{t+\tau}) = g(W_t) + \nabla g(W_t) \cdot (W_{t+\tau} - W_t)$$

where $g(W_t) = \nabla f(W_t)$ and $\nabla g(W_t) = \boldsymbol{H} f(W_t)$,

$\boldsymbol{H}$ is Hessian Matrix.

DC-ASGD

ASGD:

$$W_{t+\tau+1} = W_{t+\tau} - \eta\, g(W_t)$$

Delay Compensated ASGD:

$$W_{t+\tau+1} = W_{t+\tau} - \eta\big(g(W_t) - \lambda_t g(W_t) \odot g(W_t) \odot (W_{t+\tau} - W_t)\big)$$

DC Gradient

$diag(\lambda G)$

Training ImageNet fast with high accuracy

- ResNet 50
- ImageNet1K
- 16 GPUs

# workers	algorithm	error(%)
16	ASGD	25.64
	SSGD	25.30
	DC-ASGD-a	**25.18**

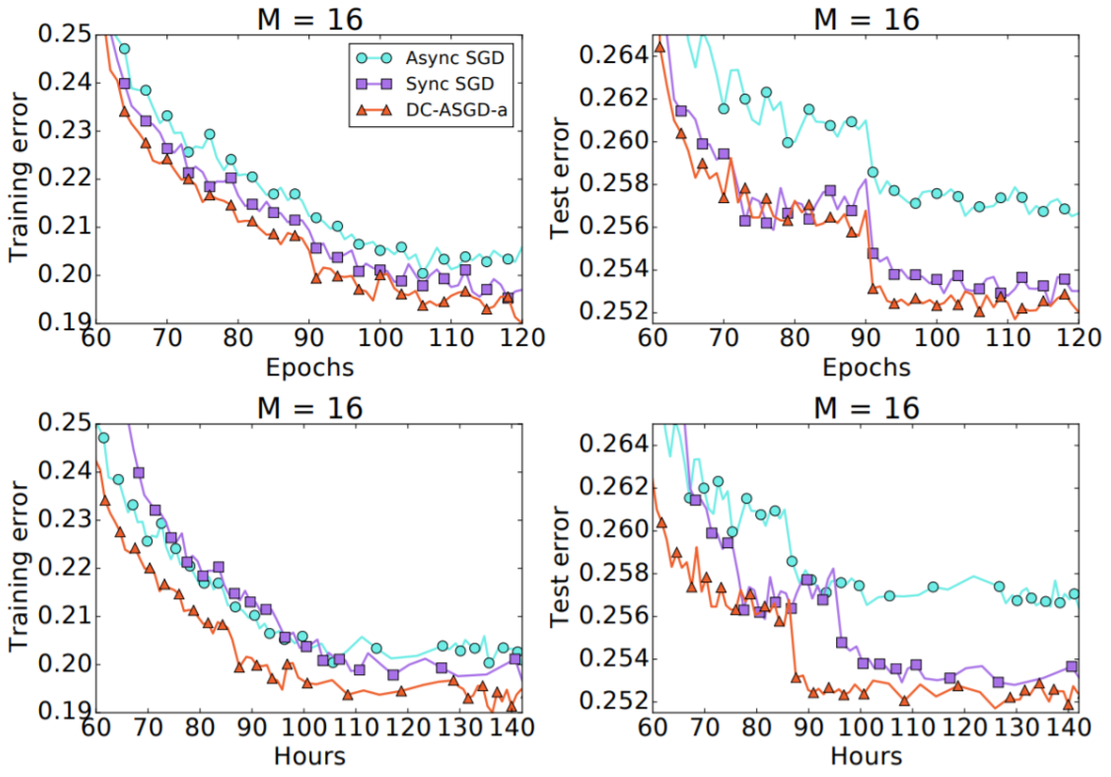


Figure 4. Error rates of the global model w.r.t. both number of effective passes and wallclock time on ImageNet

Thank you for attending

Questions?