

Deep Learning in Apache MxNet Gluon

Imperative, Dynamic and Distributed

gluon.mxnet.io

Aran Khanna, Alex Smola

AWS Machine Learning

imperative

symbolic

theano

Caffe

Microsoft
CNTK

before

2012

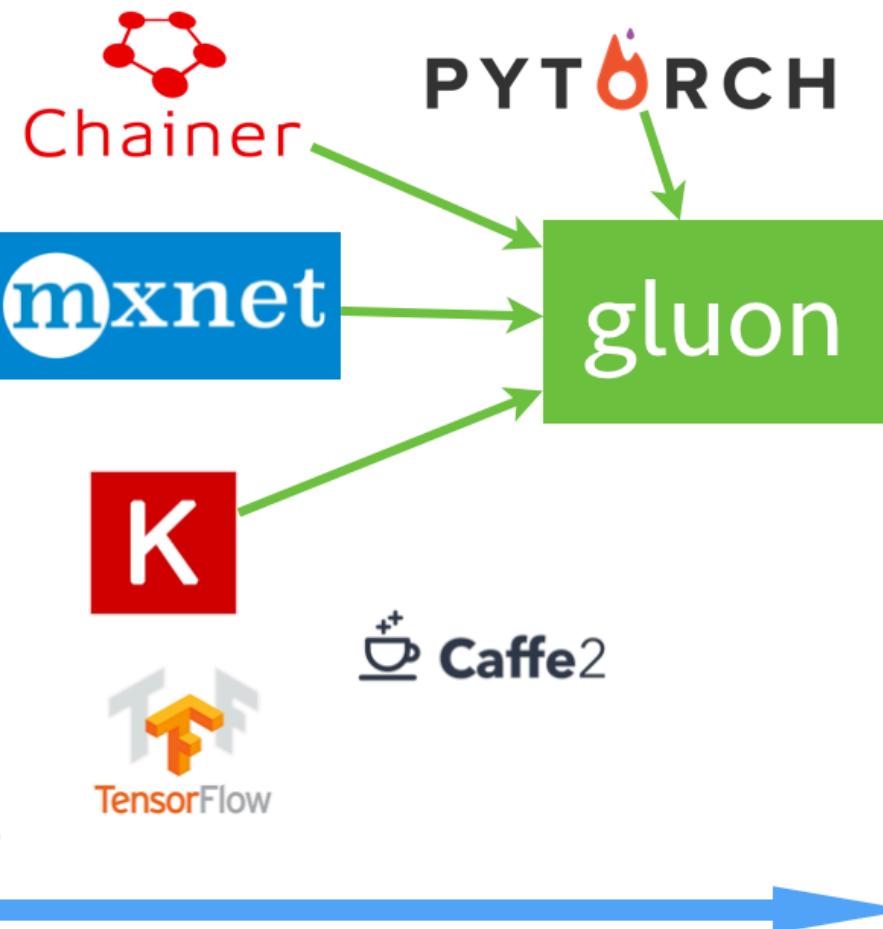
2013

2014

2015

2016

2017



Outline

1. Deep Learning 101

2. NDArray
3. Multilayer Perceptrons
4. Convolutional Neural Networks
5. Generative Adversarial Networks
6. LSTMs
7. TreeLSTMs
8. Distributed Optimization
9. Parallel Training

1. Getting started with MxNet

2. Automatic Differentiation
3. Training from scratch/Gluon
4. Gluon layers
5. More than one gradient
6. Operator fusion
7. Dynamic graphs
8. Synchronous / asynchronous
9. (key, value) store

Installation

- You need to update to the latest version of MxNet
- Build from source

```
git clone git@github.com:apache/incubator-mxnet.git  
git submodule update --init  
mxnet.io/get\_started/install.html
```

- PIP install (you should use Python 3)

```
pip install --upgrade pip  
pip install --upgrade setuptools  
pip install mxnet -pre
```

- Deep Learning AMI

bit.ly/deepami and bit.ly/deepubuntu



Programming FizzBuzz

Code

```
var o='';  
for (var i=1; i<=100; i++) {  
    i%3 || (o+='Fizz ');  
    i%5 || (o+='Buzz ');  
    !(i%3 && i%5) || (o+=(i+' '));  
}  
console.log(o);
```

That was silly.
Why would you do this?

Inspired by joelgrus.com

Programming with Data

- **Generate training data**
(9,'Fizz'),(10,'Buzz'),(2,'2'),
(15,'Fizz Buzz') ...
- **Extract input features**
 $x \rightarrow (x \% 3, x \% 5, x \% 15)$
- **Train a classifier**
mapping input x to output y
using training data



Programming with Data

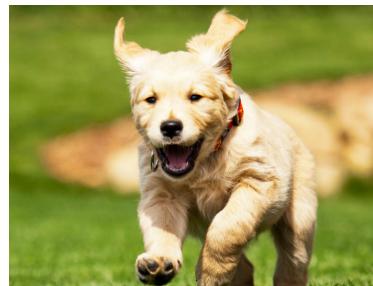
- Given some input x we want to estimate output y
 - If it's a simple deterministic rule, probably don't bother
 - Often quite complicated, vague but lots of examples**
 - Impossible to write code but can train estimator**



cat



cat



dog



dog

image credit - wikipedia

Programming with Data

- **Data** (patterns, labels)
 - Images (pictures / cat, dog, dinosaur, chicken, human)
 - Text (email / spam, ham)
 - Sound (recording / recognized speech)
 - Video (sequence / break-in, OK)
 - Images (X-ray image / size of cancer)

- **Goal** Learn mapping data to outputs

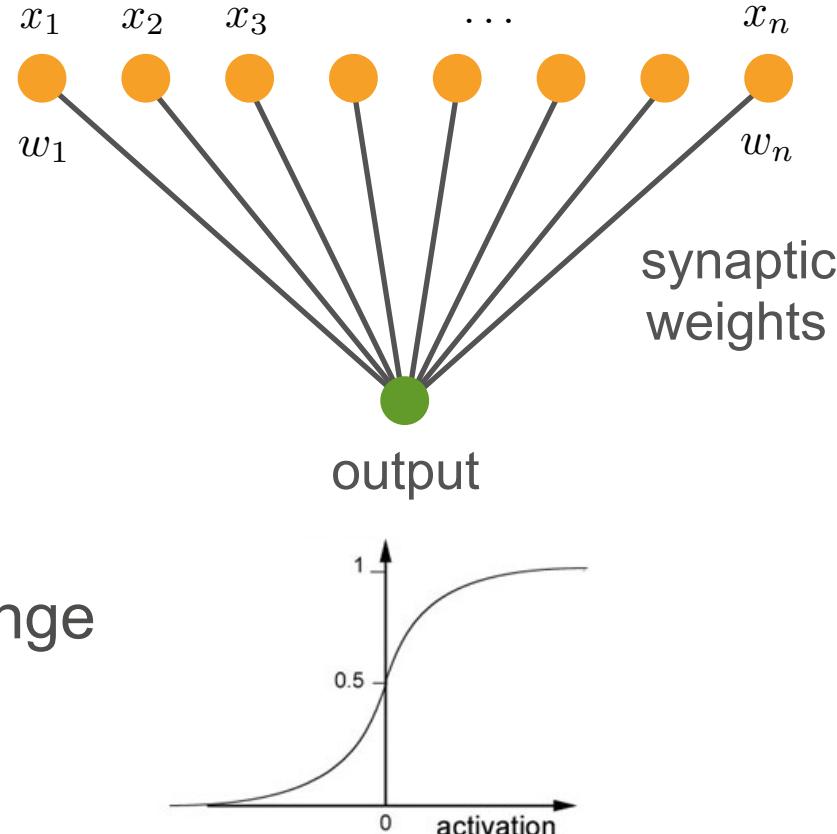
- **Loss function**

measure how well we're doing

	Edible	Poison
Truffle		0
Death cap		1,000,000

Neural Networks

- **Input**
Data vector x
- **Output**
Linear function of inputs
- **Nonlinearity**
Transform output into desired range of values, e.g. probabilities $[0, 1]$
- **Training**
Learn the weights w and bias b

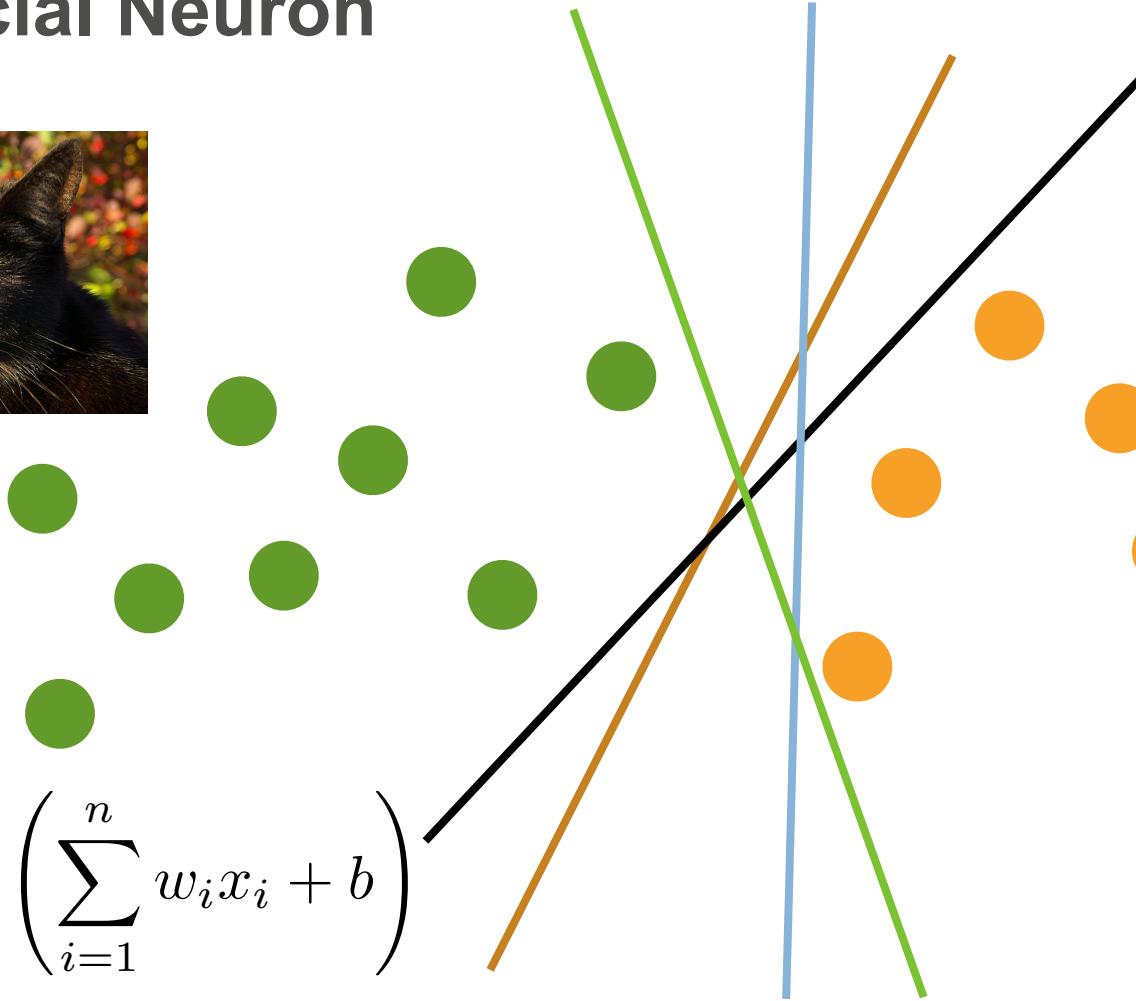


$$f(x) = \sigma \left(\sum_{i=1}^n w_i x_i + b \right)$$

Artificial Neuron



$$f(x) = \sigma \left(\sum_{i=1}^n w_i x_i + b \right)$$



MxNet Code

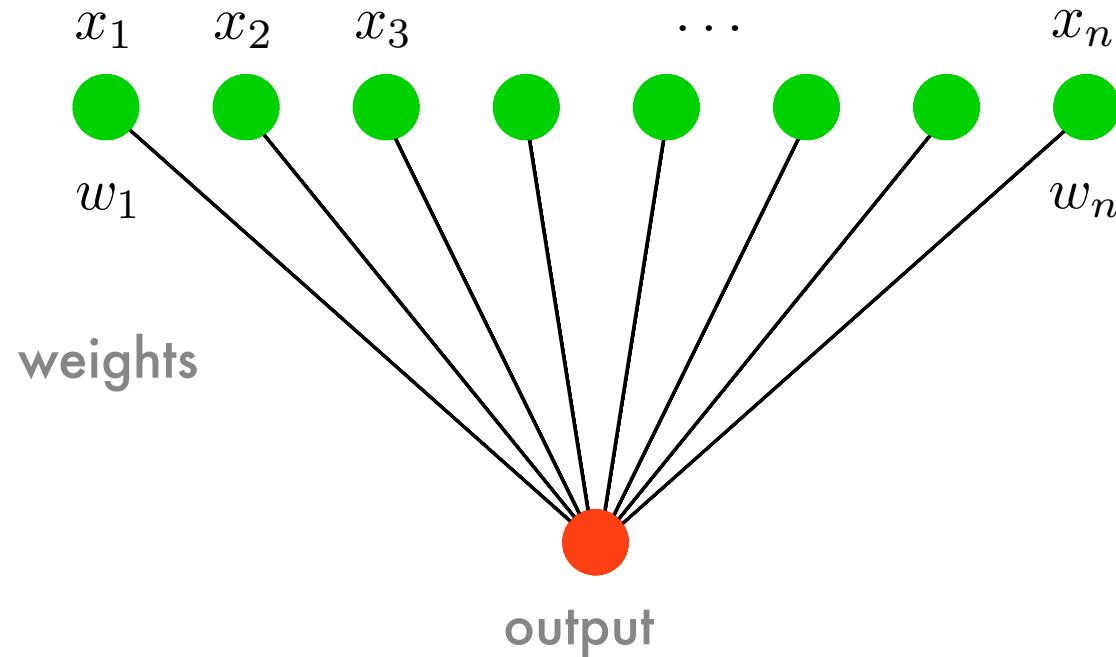
```
import mxnet as mx  
from mxnet import gluon  
  
net = gluon.nn.Sequential()  
net.add(gluon.nn.Dense(1))  
loss = gluon.loss.SoftmaxCrossEntropyLoss()
```

fully connected
layer, 1 output

+ iterator over data (and loader)
+ training loop

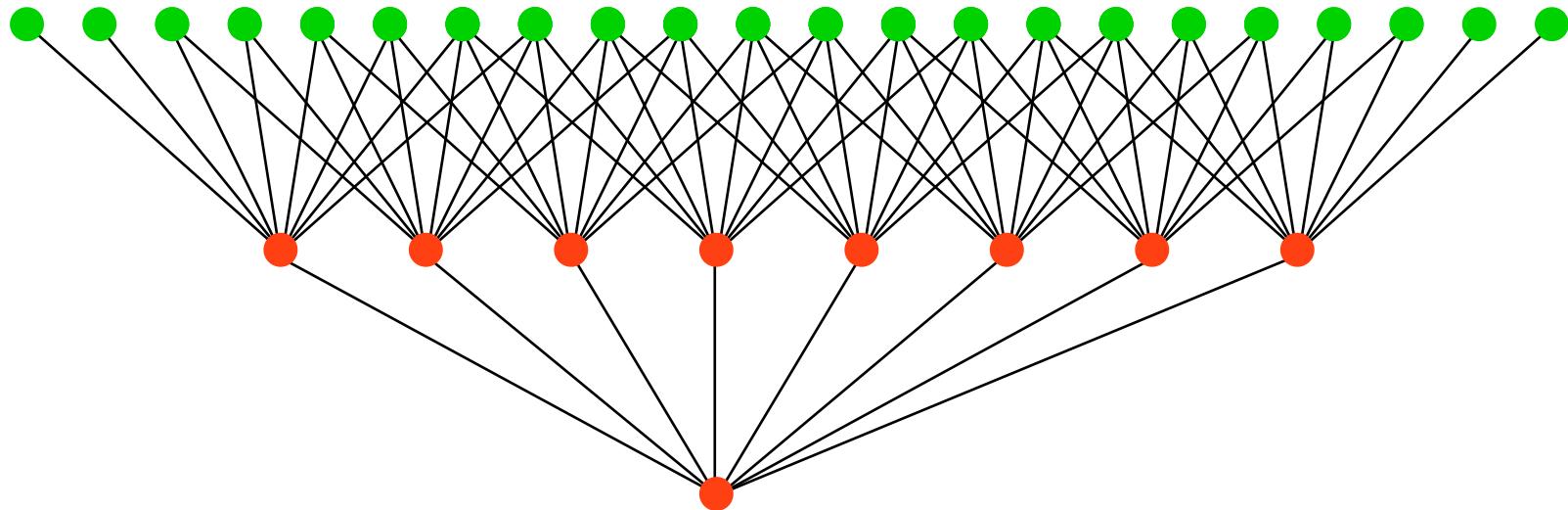
Multilayer Perceptron and Backprop

Perceptron



$$y(x) = \sigma(\langle w, x \rangle)$$

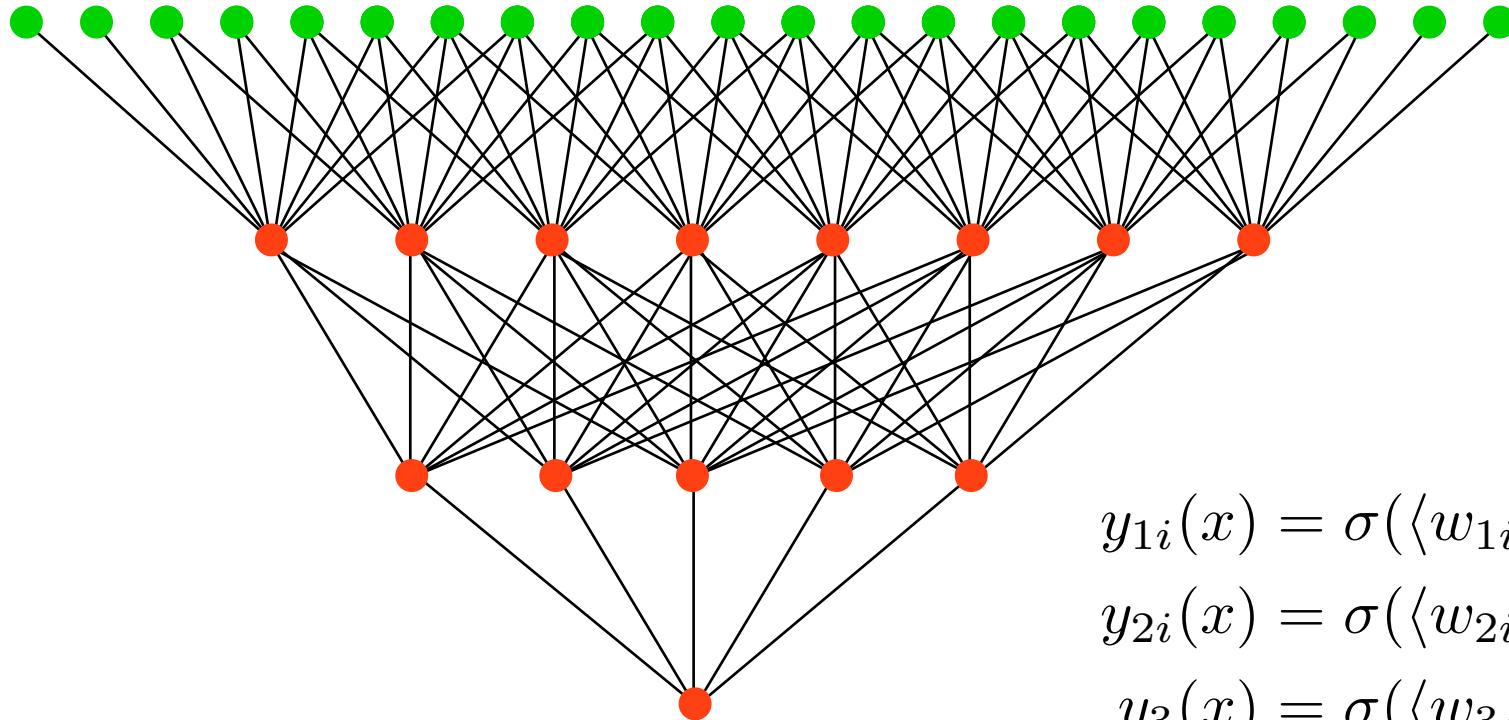
Multilayer Perceptron



$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$

$$y_2(x) = \sigma(\langle w_2, y_1 \rangle)$$

Multilayer Perceptron



Multilayer Perceptron Training

Layer Representation

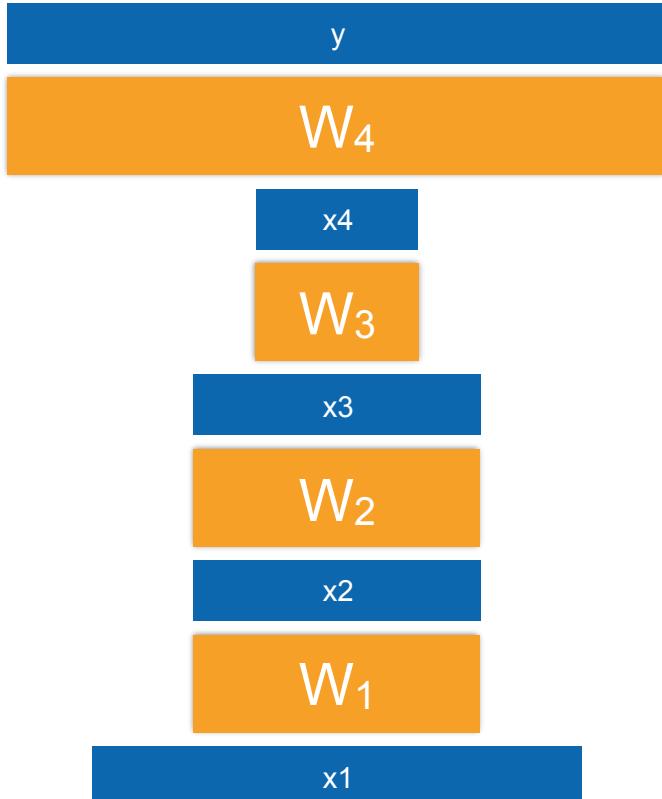
$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

Gradient descent

$$W_i \leftarrow W_i - \eta \partial_{W_i} l(y, y_n)$$

- **Stochastic gradient descent**
(use only one sample)
- **Minibatch** (small subset)



Multilayer Perceptron Training

Layer Representation

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

Change in Objective

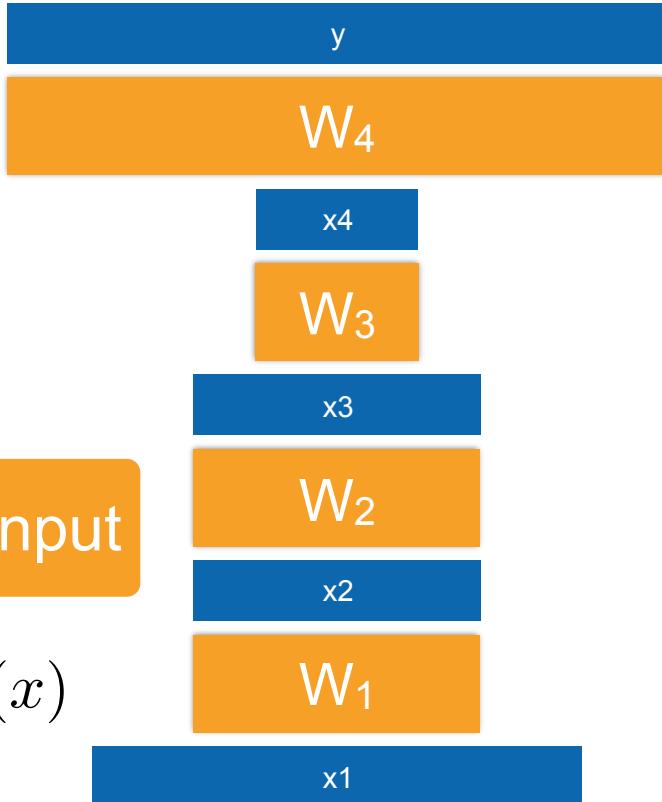
$$g_j = \partial_{W_j} l(y, y_i)$$

Chain Rule

$$\partial_x [f_2 \circ f_1] (x) = [\partial_{f_1} f_2 \circ f_1] (x) [\partial_x f_1] (x)$$

effect of input

second layer



Multilayer Perceptron Training

Layer Representation

$$y_i = W_i x_i$$

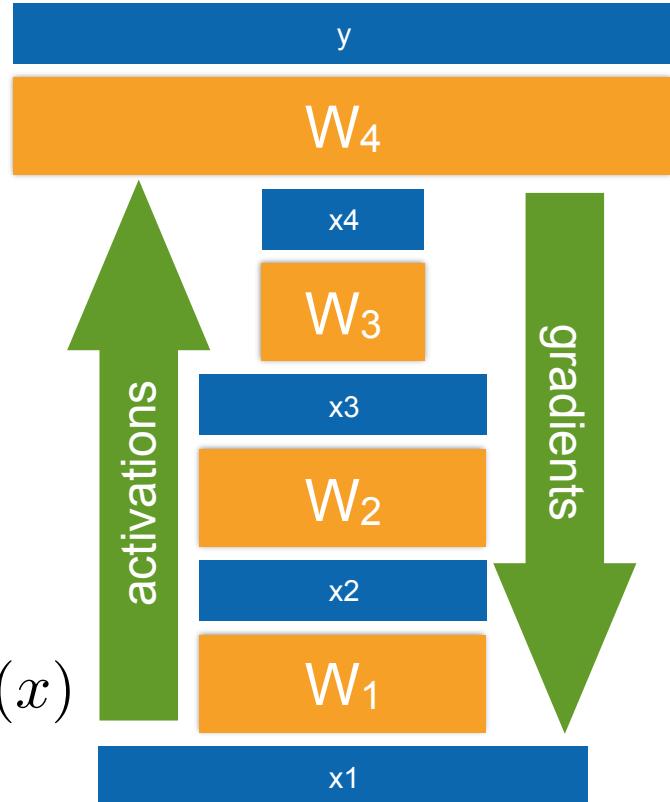
$$x_{i+1} = \sigma(y_i)$$

Change in Objective

$$g_j = \partial_{W_j} l(y, y_i)$$

Chain Rule

$$\partial_x [f_2 \circ f_1](x) = [\partial_{f_1} f_2 \circ f_1](x) [\partial_x f_1](x)$$



MxNet computes this automatically

That was complicated! MXNet takes care of this for you ...

```
import mxnet as mx  
from mxnet import gluon  
  
net = gluon.nn.Sequential()  
with net.name_scope():  
    net.add(gluon.Dense(128, activation='relu'))  
    net.add(gluon.Dropout(dropout=0.5))  
    net.add(gluon.Dense(64, activation='relu'))  
    net.add(gluon.Dropout(dropout=0.5))  
    net.add(gluon.Dense(1))  
  
loss = gluon.loss.SoftmaxCrossEntropyLoss()
```

nonlinearity

regularization

second layer

Outline

1. Deep Learning 101

2. NDArray

3. Multilayer Perceptrons

4. Convolutional Neural Networks

5. Generative Adversarial Networks

6. LSTMs

7. TreeLSTMs

8. Distributed Optimization

9. Parallel Training

1. Getting started with MxNet

2. Automatic Differentiation

3. Training from scratch/Gluon

4. Gluon layers

5. More than one gradient

6. Operator fusion

7. Dynamic graphs

8. Synchronous / asynchronous

9. (key, value) store

Matrices and Vectors

- A Simple ML algorithm

```
initialize function f(x,w)
```

```
for (x,y) in data:
```

```
    l = loss(y, f(x,w))
```

```
    g = gradient(l)
```

```
    w = w - eta * g
```

- Need vectors to deal with w
- Need matrices to deal with chain rule for loss gradient
- Need high performance linear algebra



NDArray (Linear Algebra on GPUs in Python)

- NumPy
 - Limited to CPUs
 - No automatic differentiation
 - **Blocking (returns only once computation is performed)**
- NDArray
 - Multiple CPUs / GPUs via device context
 - Distributed systems in the cloud
 - **Nonblocking Python frontend (no GIL) & lazy evaluation**

<http://mxnet.io/tutorials/basic/ndarray.html>

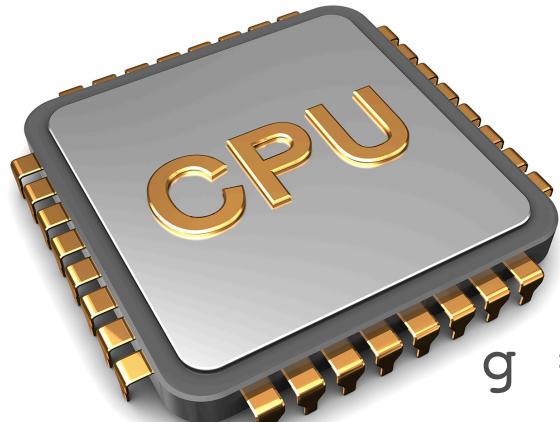


NDArray Device Contexts

```
context = mx.cpu()
```

```
context = mx.gpu(0)
```

```
context = mx.gpu(1)
```



Caution!
NDArrays are
immutable on GPUs

```
g = copyto(c)
```

```
g = c.as_in_context(mx.gpu(0))
```

Automatic Differentiation

```
b = a * 2
while (nd.norm(b) < 1000).asscalar():
    b = b * 2
if (mx.nd.sum(b) > 0).asscalar():
    c = b
else :
    c = 100 * b
```

Compute dc/da ?

Links & Notebooks

gluon.mxnet.io

github.com/zackchase/mxnet-the-straight-dope
mxnet.io/api/python/gluon.html

<http://gluon.mxnet.io/P01-C03-linear-algebra.html>

<http://gluon.mxnet.io/P01-C05-autograd.html>

Outline

- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons**
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon**
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store

Linear Regression

- Model

$$f(x) = \langle w, x \rangle + b$$

- Loss

$$l(y, f(x)) = \frac{1}{2}(y - f(x))^2$$

- Optimization algorithm - SGD

- Initialize w, b at random (or at zero)
 - Iterate over data

$$(w, b) \leftarrow (w, b) - \eta \partial_{(w,b)} l(y, f(x))$$

Logistic Regression (Multiclass)

- Model (single layer)

$$f(x) = Wx + b$$

- Multilayer Perceptron

$$x_1 = x_{\text{input}} \text{ and } x_{l+1} = \sigma(W_l x_l + b_l)$$

- Loss (exponential family)

$$-\log p(y|x) = \log \sum_{y'} \exp(f(x)_{y'}) - f(x)_y$$

Links & Notebooks

<http://gluon.mxnet.io/P02-C01-linear-regression-scratch.html>
<http://gluon.mxnet.io/P02-C02-linear-regression-gluon.html>
<http://gluon.mxnet.io/P02-C04-softmax-regression-gluon.html>
<http://gluon.mxnet.io/P03-C02-mlp-gluon.html>

Outline

- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks**
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers**
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store

Where is Waldo?



image credit - wikipedia

Where is Waldo?

Waldo model is global

Waldo information is local.

Convolutional Layers

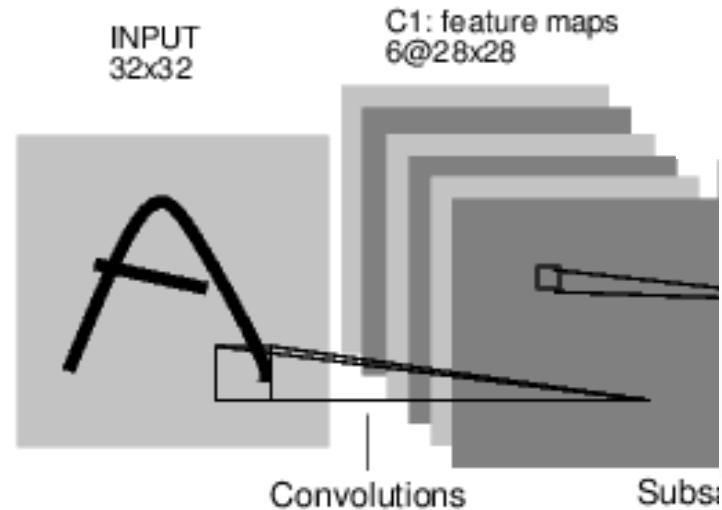
- Images have shift invariance
- Low level mostly edge and feature detectors
- **Key Idea**

Replace matrix multiplication
with local computation

$$y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ab} x_{i+a, j+b}$$

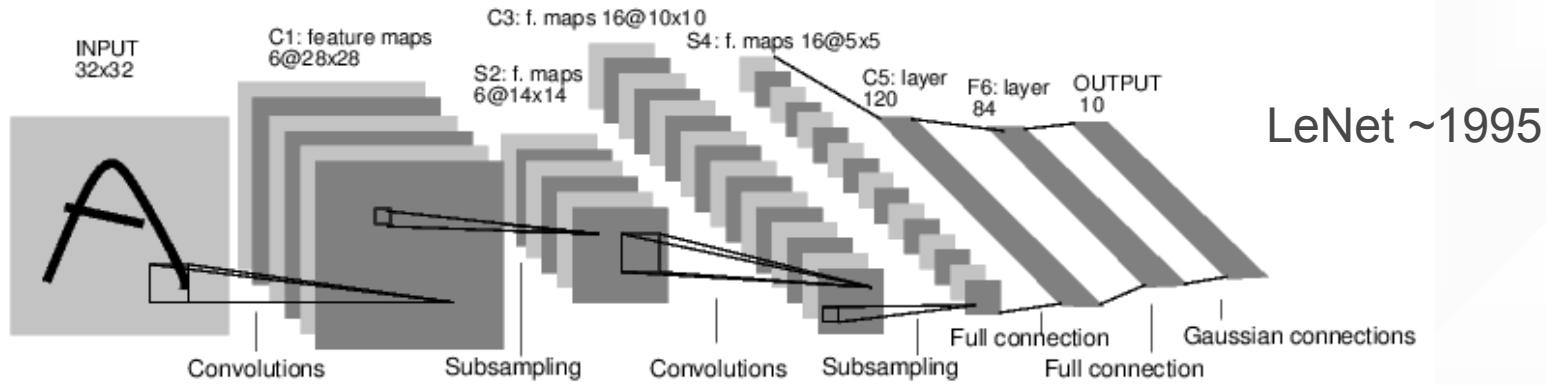
- Backprop automatic in MXNet

image credit - Le Cun et al, 1998, Olshausen and Field, 1998



Subsampling & MaxPooling

- Multiple convolutions blow up dimensionality



- Subsampling - average over patches (works OK)
- MaxPooling - pick the maximum over patches (much better)

image credit - Le Cun et al, 1998

LeNet in MXNet

```
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Flatten())
    net.add(gluon.nn.Dense(500, activation='tanh'))
    net.add(gluon.nn.Dense(10))

loss = gluon.loss.SoftmaxCrossEntropyLoss()

(size and shape inference is automatic)
```

More Layers

- **The usual suspects**

- gluon.nn.Dense(units=50, ...)
- gluon.nn.Activation(activation='relu')
- gluon.nn.Dropout(rate=0.3)
- gluon.nn.BatchNorm(...)
- gluon.nn.Embedding(...) for text

- **Convolutions**

- gluon.nn.Conv1D, 2D, 3D
- gluon.nn.Conv1DTranspose, 2D, 3D for Deconvolution

- **Pooling Layers**

- gluon.nn.MaxPool1D, 2D, 3D, gluon.nn.AvgPool1D, 2D, 3D

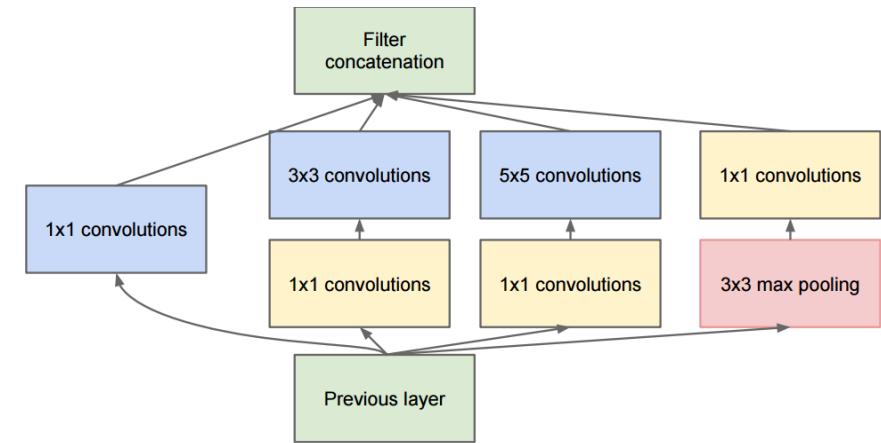
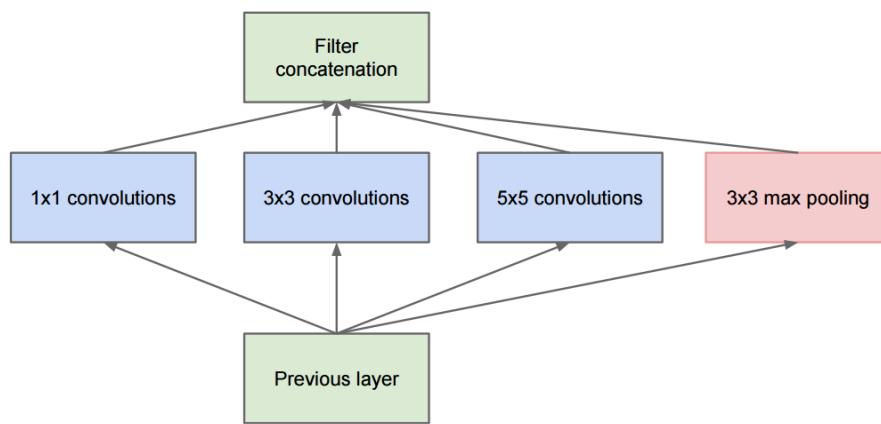
Links & Notebooks

<http://gluon.mxnet.io/P04-C02-cnn-gluon.html>

<http://gluon.mxnet.io/P14-C05-hybridize.html>

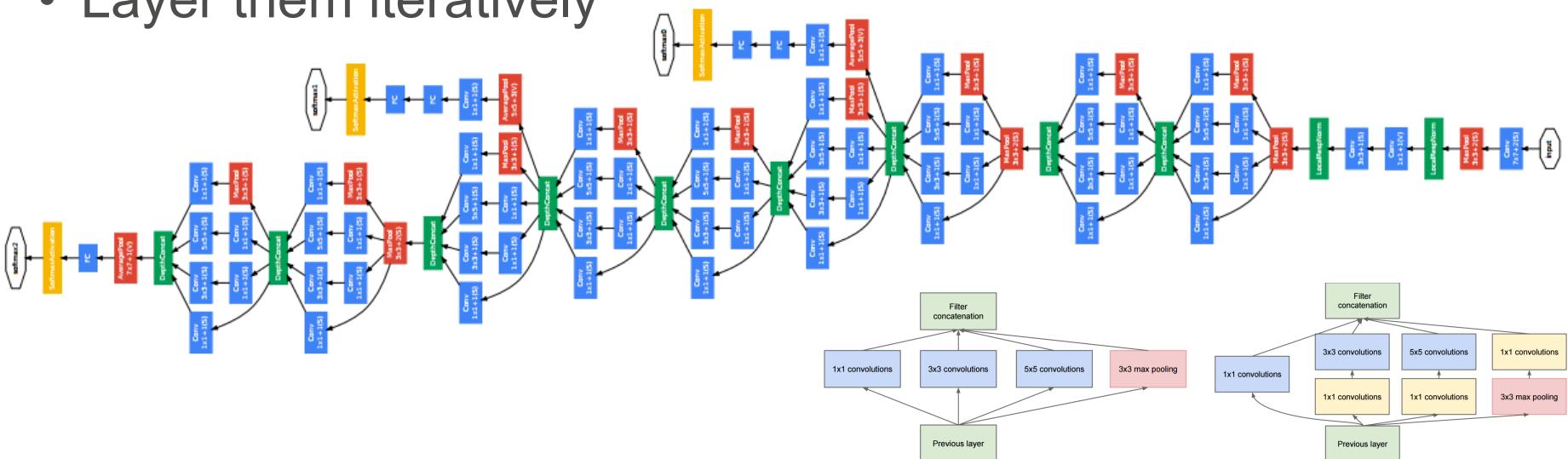
Fancy structures

- Compute different filters
- Compose one big vector from all of them
- Layer them iteratively



Fancy structures

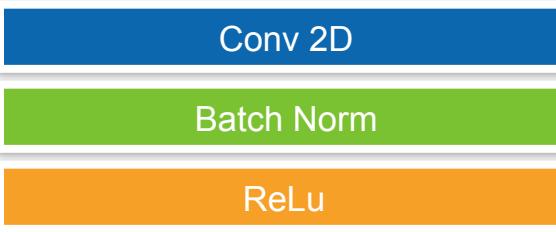
- Compute different filters
- Compose one big vector from all of them
- Layer them iteratively



Networks of networks - Sequential Composition

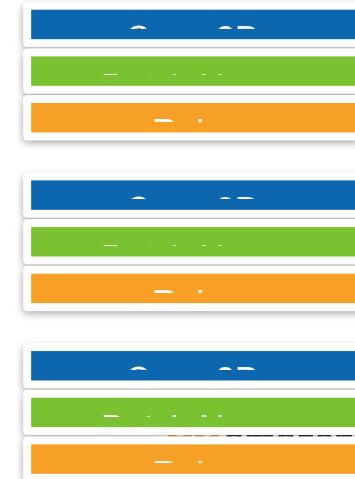
- Define Single Block

```
def _make_basic_conv(**kwargs):  
    out = nn.HybridSequential(prefix='')  
    out.add(nn.Conv2D(use_bias=False, **kwargs))  
    out.add(nn.BatchNorm(epsilon=0.001))  
    out.add(nn.Activation('relu'))  
    return out
```



- Compose Multiple Blocks into Branch

```
def _make_branch(*conv_settings):  
    out = nn.HybridSequential(prefix='')  
    out.add(nn.MaxPool2D(pool_size=3, strides=2))  
    setting_names = ['channels', 'kernel_size', 'strides', 'padding']  
    for setting in conv_settings:  
        kwargs = {}  
        for i, value in enumerate(setting):  
            if value is not None:  
                kwargs[setting_names[i]] = value  
        out.add(_make_basic_conv(**kwargs))  
    return out
```



Networks of networks - Parallel Composition

- Parallel composition of branches

```
def _make_A(pool_features, prefix):
    out = HybridConcurrent(concat_dim=1, prefix=prefix)
    with out.name_scope():
        out.add(_make_branch(None, (64, 1, None, None)))
        out.add(_make_branch(None, (48, 1, None, None),
                            (64, 5, None, 2)))
        out.add(_make_branch(None, (64, 1, None, None),
                            (96, 3, None, 1),
                            (96, 3, None, 1)))
    out.add(_make_branch('avg', (pool_features, 1, None, None)))
    return out
```



Model Zoo

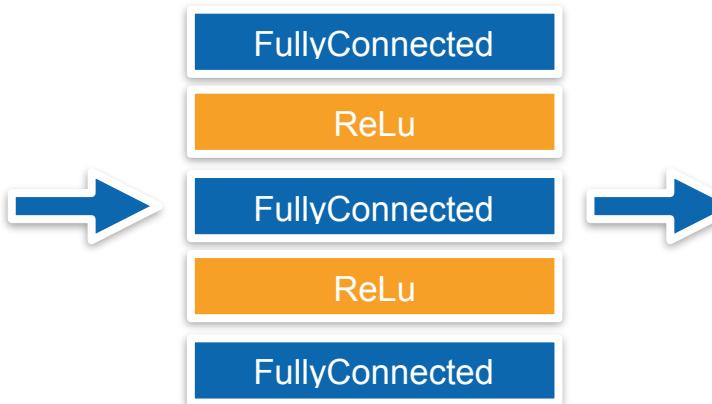
- Lots more networks at `mxnet.gluon.model_zoo.vision` (with all the model parameters you want)
 - AlexNet (because, why not)
 - SqueezeNet
 - Inception
 - VGG
 - DenseNet
 - ResNet

... import from PyTorch is trivial ...

Hybridization (a JIT compiler for Networks)

- Loops and control flow are easy to understand
- Runtime engine needs to parse them each time (slow)
- Use JIT compiler to convert into compute graph
(no need to parse the code more than once)

```
x = F.relu(self.fc1(x))  
x = F.relu(self.fc2(x))  
return self.fc3(x)
```



JSON
Network definition
Serialize weights

Outline

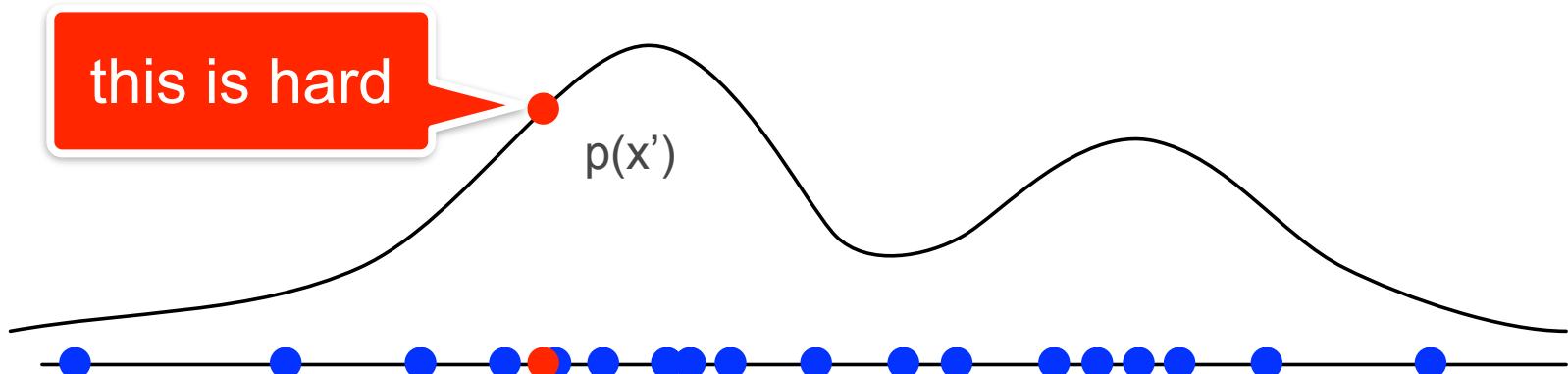
- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks**
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient**
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store

Generative Adversarial Networks Primer

- **Traditional Statistics**
 - Given data X , try to find some distribution p that would have generated X .
- **Why?**
 - Assess how well some observation x' fits with the data, by evaluating $p(x')$.
 - Draw new fake data x' that looks like it came from X
 - Summarize X concisely (sufficient statistics)

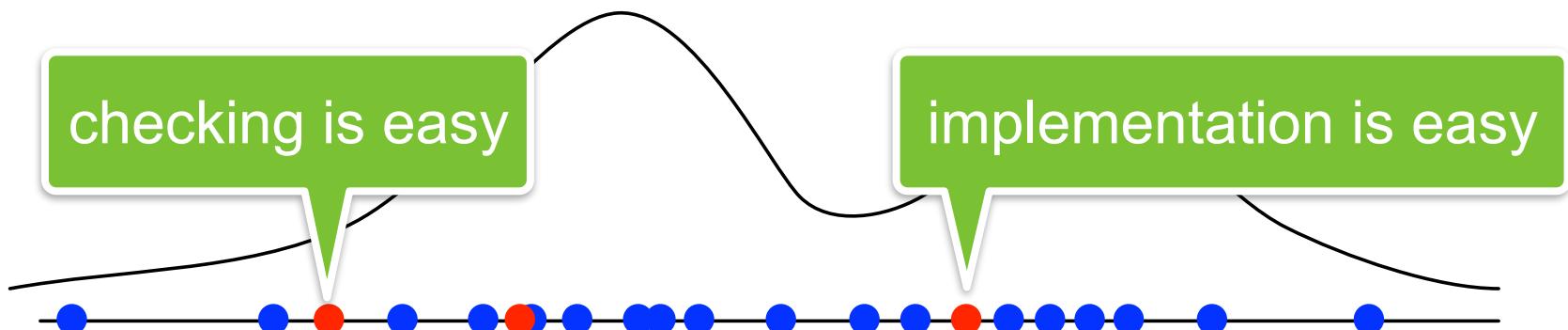
Generative Adversarial Networks Primer

- Assess how well some observation x' fits with the data, by evaluating $p(x')$.
- Draw new fake data x' that looks like it came from X
- Summarize X concisely (sufficient statistics)



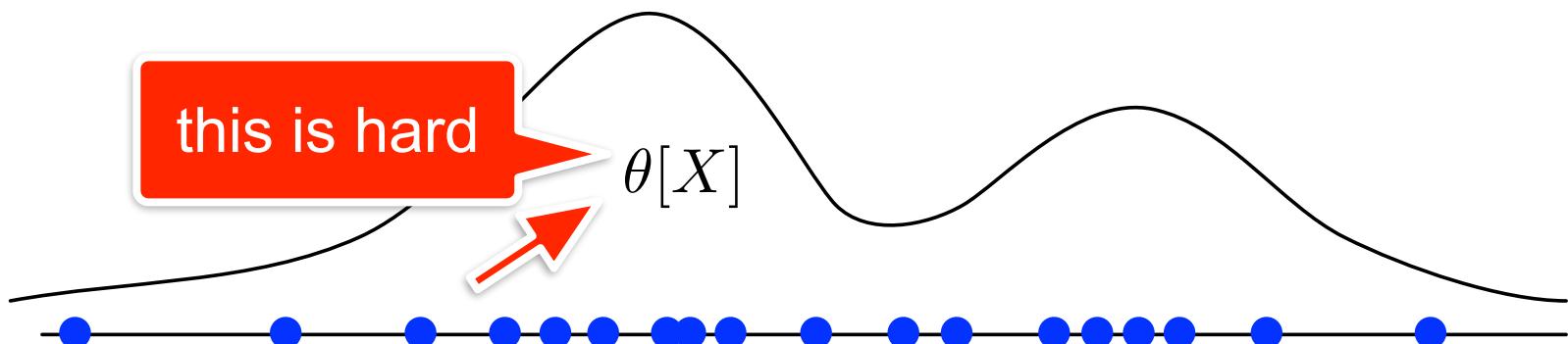
Generative Adversarial Networks Primer

- Assess how well some observation x' fits with the data, by evaluating $p(x')$.
- Draw new fake data x' that looks like it came from X
- Summarize X concisely (sufficient statistics)



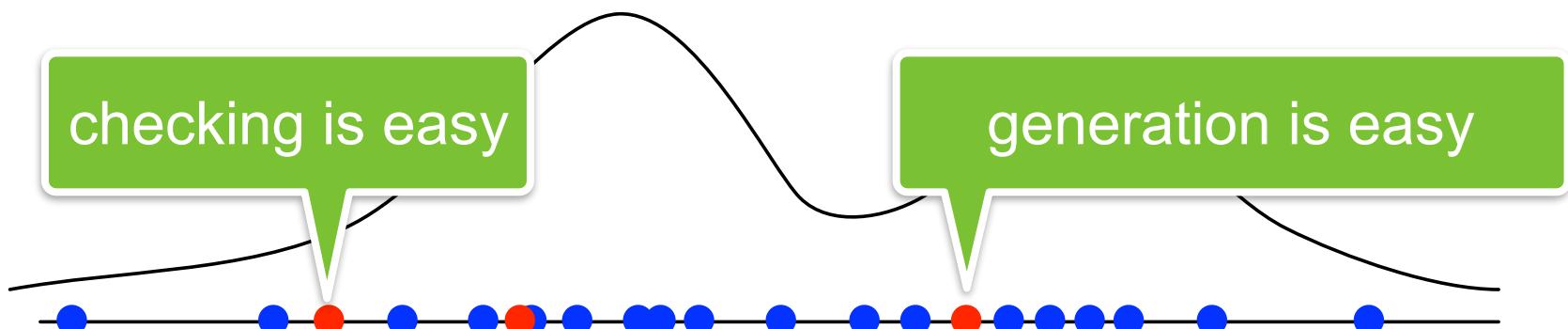
Generative Adversarial Networks Primer

- Assess how well some observation x' fits with the data, by evaluating $p(x')$.
- Draw new fake data x' that looks like it came from X
- Summarize X concisely (sufficient statistics)



Key Idea - Focus on generating data

- **Generator network** $x' = g(z)$ that produces ‘fake’ data
 - Start with some random z
 - Transform it via neural network
- **Discriminator network** $d(x)$ distinguishing x from fake x'



Objective Functions

- Discriminator Network uses logistic regression
(binary classification between real and fake data)

$$\frac{1}{n} \sum_{i=1}^n \log(1 + \exp(d(x_i))) + \mathbf{E}_{x' \sim q(x')} [\log(1 + \exp(-d(x')))]$$

real data

fake data

- Generative Network tries to fool data
(maximize acceptance or minimize rejection)

$$\mathbf{E}_u [\log(1 + \exp(d(g(z)))]$$

Multiple Gradients

- ‘Normal’ SGD loop
 - Iterate over data and compute loss
 - Update parameters w.r.t. loss gradient
- GAN loop
 - Iterate over data and sample from generator
 - Compute loss and gradient on real & fake data
 - Update parameters for discriminator
 - Sample from generator
 - Compute loss and gradient on real data
 - Update parameters for generator



different
parameters

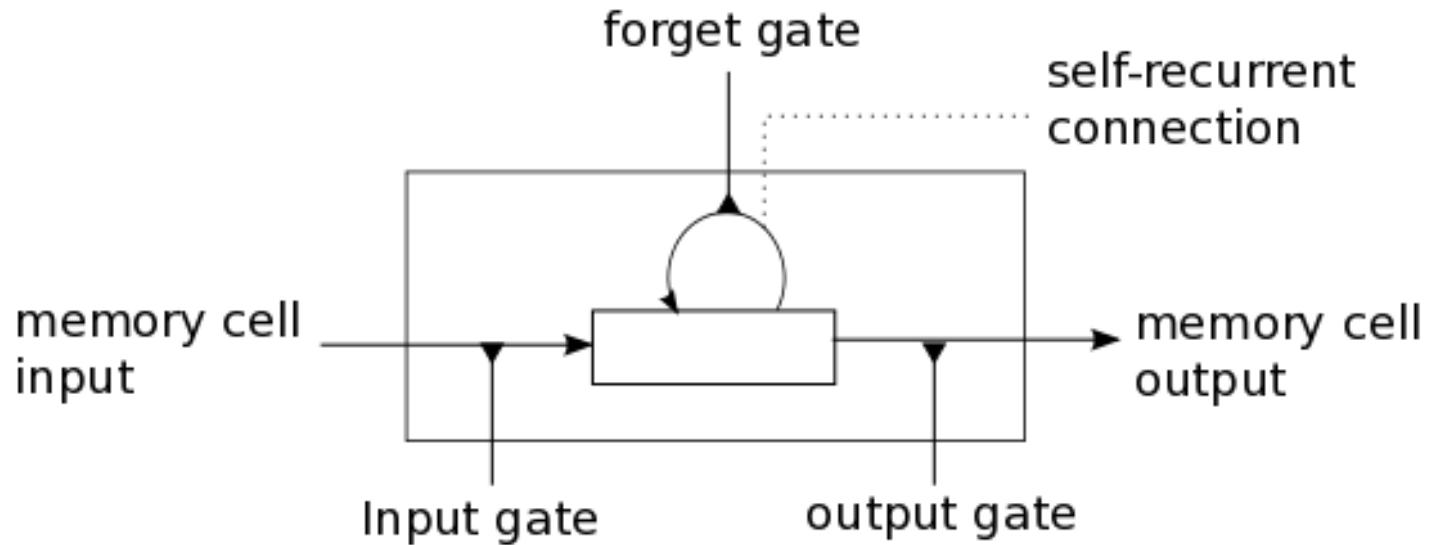
Links & Notebooks

<http://gluon.mxnet.io/P10-C01-gan-intro.html>

Outline

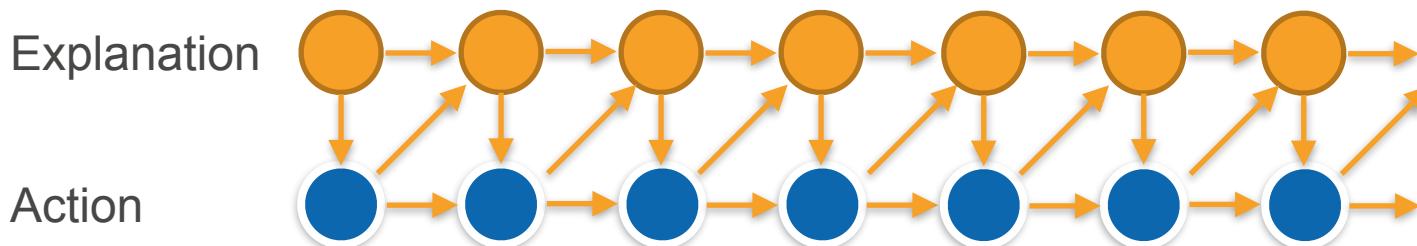
- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs**
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion**
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store

Sequence Models



Latent Variable Models

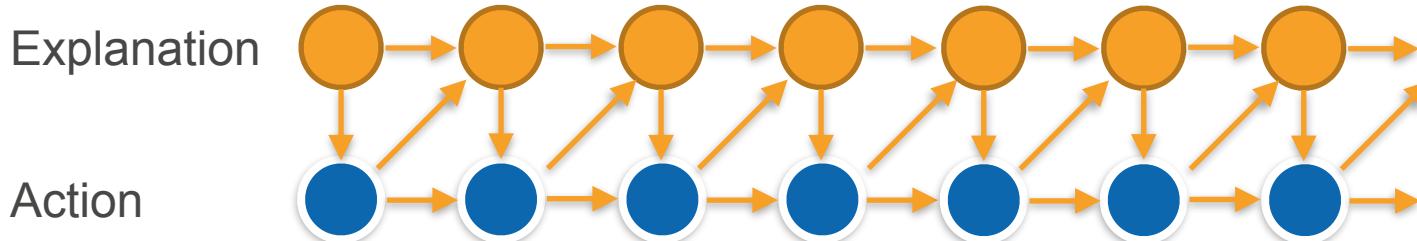
- **Temporal sequence of observations**
Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
 - Clusters (navigational, informational queries in search)
 - Topics (interest distributions for users over time)
 - Kalman Filter (trajectory and location modeling)



Latent Variable Models

- **Temporal sequence of observations**
Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**

Are the parametric models really true?

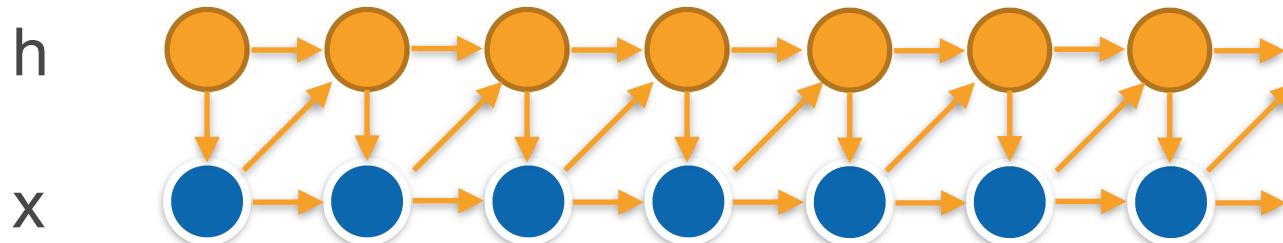


Latent Variable Models

- **Temporal sequence of observations**
Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
 - Nonparametric model / spectral
 - Use data to determine shape
 - Sidestep approximate inference

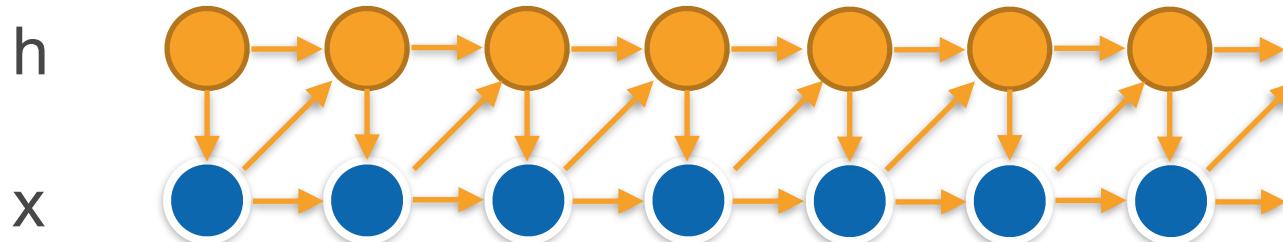
$$h_t = f(x_{t-1}, h_{t-1})$$

$$x_t = g(x_{t-1}, h_t)$$



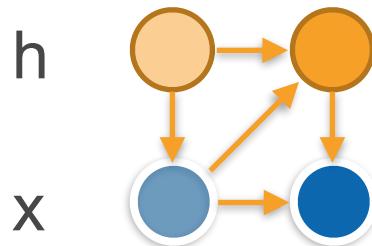
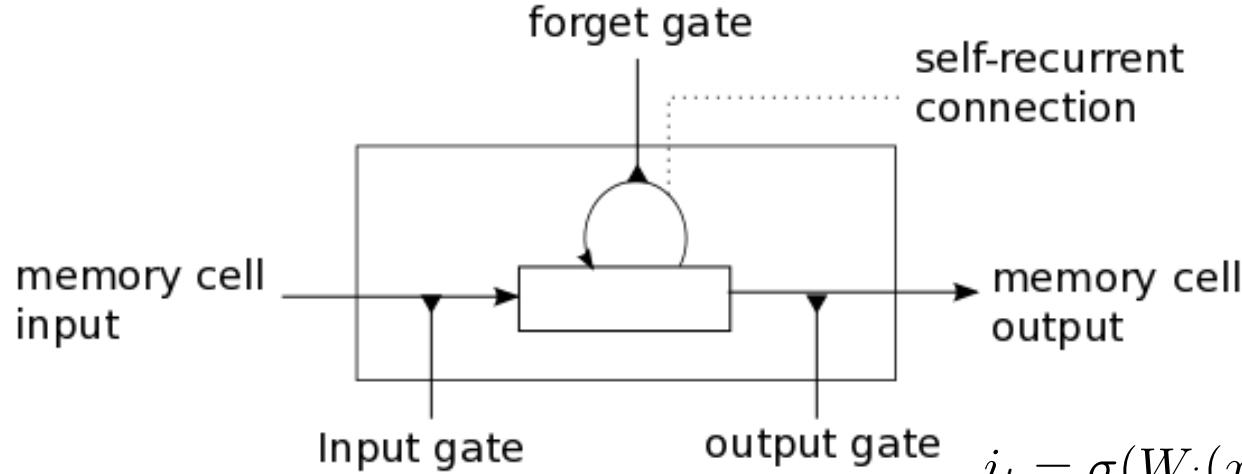
Latent Variable Models

- **Temporal sequence of observations**
Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
 - Plain deep network = RNN
 - Deep network with attention = LSTM / GRU ...
(learn when to update state, how to read out)



Long Short Term Memory

Hochreiter & Schmidhuber, 1997



$$i_t = \sigma(W_i(x_t, h_t) + b_i)$$

$$f_t = \sigma(W_f(x_t, h_t) + b_f)$$

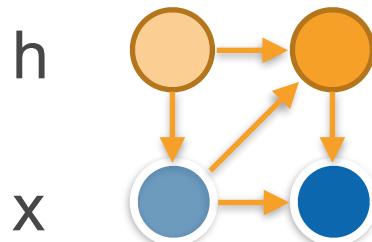
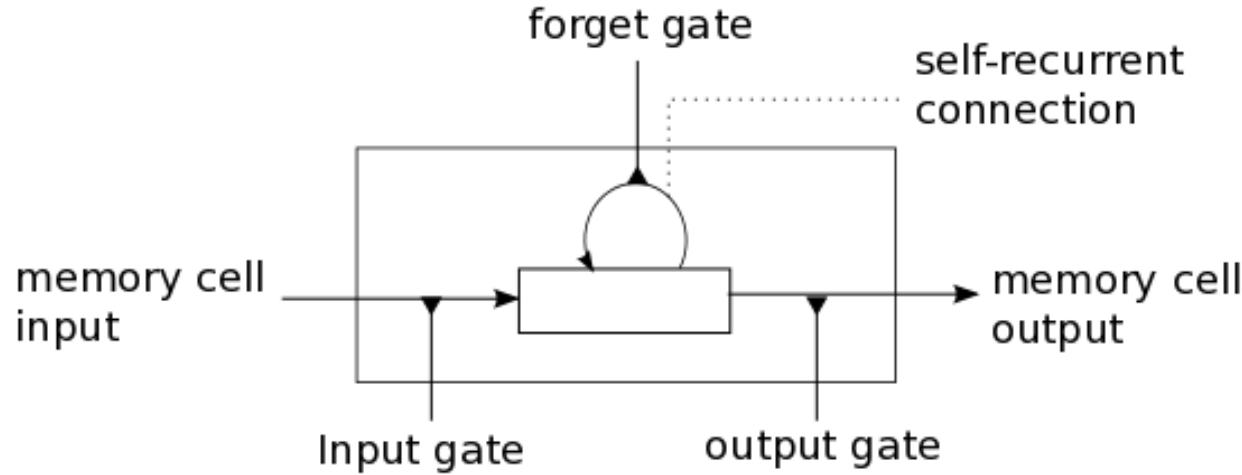
$$z_{t+1} = f_t \cdot z_t + i_t \cdot \tanh(W_z(x_t, h_t) + b_z)$$

$$o_t = \sigma(W_o(x_t, h_t, z_{t+1}) + b_o)$$

$$h_{t+1} = o_t \cdot \tanh z_{t+1}$$

Long Short Term Memory

Hochreiter & Schmidhuber, 1997



$$(z_{t+1}, h_{t+1}, o_t) = \text{LSTM}(z_t, h_t, x_t)$$

```
rnn.LSTM(num_hidden, num_layers, dropout, input_size)
```

The anatomy of an RNN language model

- **Statistical Model**

$$p(x) = \prod_{i=0}^{n-1} p(x_{i+1} | x_i \dots x_1) = \prod_{i=0}^{n-1} p(x_{i+1} | \text{LSTM}(x_i, h_i))$$

- **Truncated Backpropagation Through Time (BPTT)**

- Sequence is often too long for exact computation
- Truncate and predict smaller segments (approximation)

$$p(x) = \prod_{j=0}^{\lfloor n/L \rfloor} \prod_{i=0}^{L-1} p(x_{i+1+jL} | \text{LSTM}(x_{i+jL}, h_{i+jL}))$$

The anatomy of an RNN language model

- **Data**
 - Some sequence to predict the next ...
 - Break into batches
 - Tokenize words (or characters)
 - Truncate for limited Backprop through Time (BPTT)
- **Encoder**
 - Map token i into vector v_i
(one-hot encoding followed by an MLP)

The anatomy of an RNN language model

- **RNN**
 - Initialize with some state s
 - Recurse through sequence to compute
$$(h_{i+1}, o_{i+1}) = \text{LSTM}(h_i, x_i)$$
- **Decoder**
 - Softmax decoding for output, i.e
$$p(x_i | o_i) = \text{softmax}(\text{MLP}(o_i))$$
 - Better decoders possible (beam search, WFST ...)

Links & Notebooks

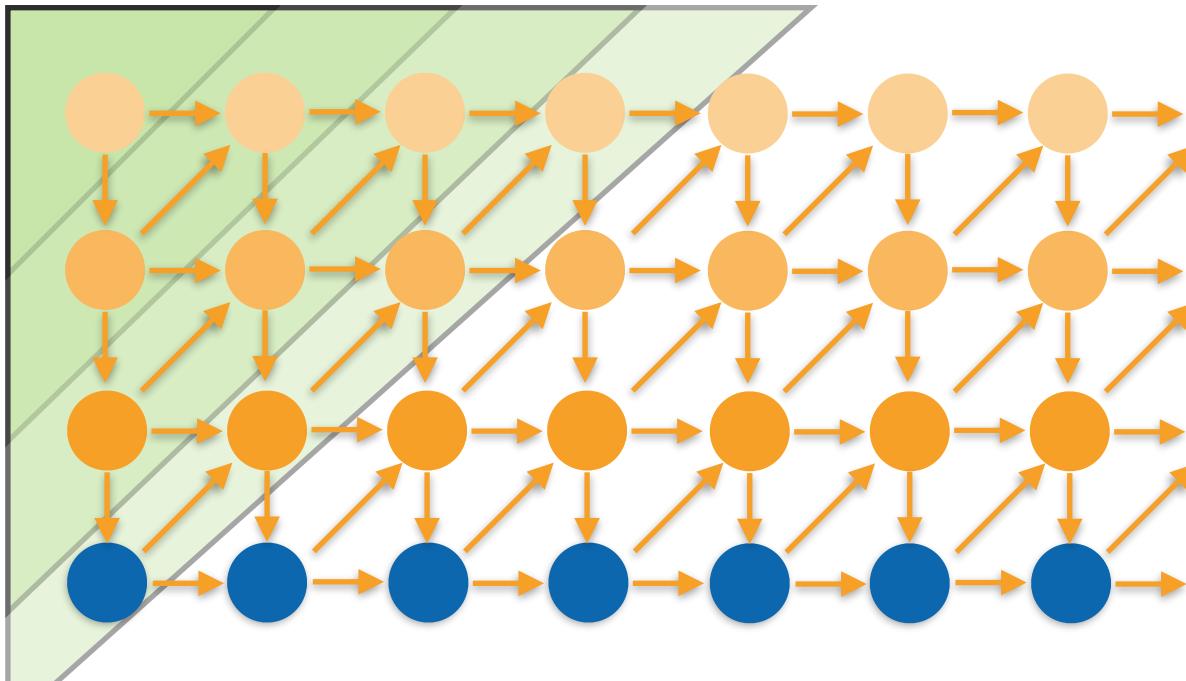
gluon.mxnet.io/P05-C01-simple-rnn.html

gluon.mxnet.io/P05-C03-lstm-scratch.html

gluon.mxnet.io/P05-C04-rnns-gluon.html

Fusion for RNNs

- GPUs too fast to call a matrix-vector product every time
- Fuse CUDA kernels (automatic if you use Gluon RNNs)



Outline

- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs**
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs**
- 8. Synchronous / asynchronous
- 9. (key, value) store

Tree LSTMs

- **LSTM**
sequential dependence
between states

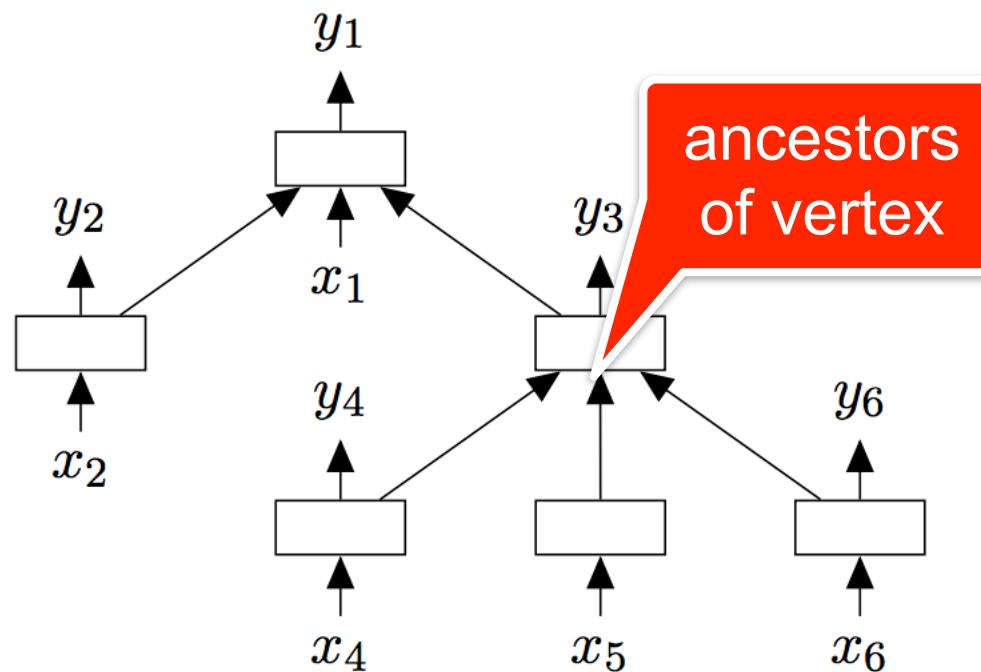
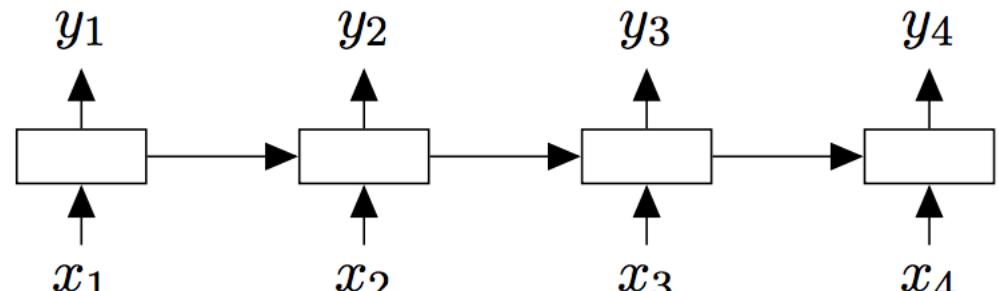
$$o_t = f(x_t, h_{t-1})$$

$$(c_t, h_t) = g(x_t, h_{t-1}, c_{t-1})$$

- **Tree LSTM**
hierarchical dependence

$$o_t = f(x_t, H_{t-1})$$

$$(c_t, h_t) = g(x_t, H_{t-1}, C_{t-1})$$



Tree LSTMs (Tsai et al, 2015)

- **Applications**

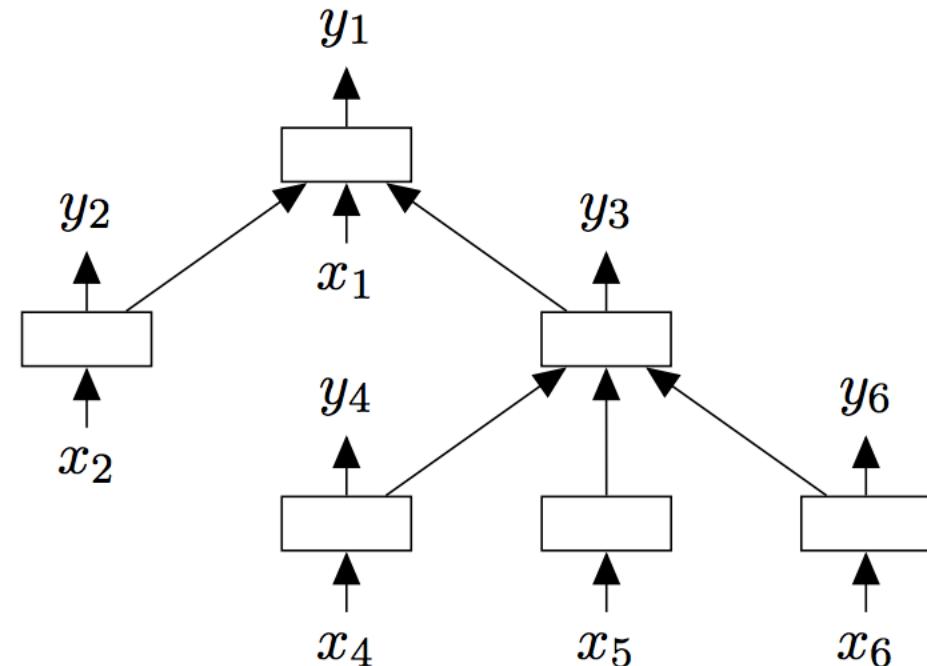
- dependency parsing
- sentiment
- document similarity

- **Problems**

- **Variable inputs H and C**
- Specialization (left, right)
- **Model has variable dependency structure**

$$o_t = f(x_t, H_{t-1})$$

$$(c_t, h_t) = g(x_t, H_{t-1}, C_{t-1})$$



LSTM

$$z = Wx_t + Uh_{t-1} + b$$

$$[i_t, f_t, o_t, u_t] = [\sigma(z_i), \sigma(z_f), \sigma(z_o), \tanh(z_u)]$$

$$c_t = i_t \circ u_t + f_t \circ c_{t-1}$$

$$h_t = o_t \circ \tanh(c_t)$$

Tree LSTM

individual
forget weights

$$\tilde{h} = \sum_{j \in C(t)} h_j$$

aggregate
hidden state

$$z = Wx_t + U\tilde{h} + b$$

$$f_{tj} = \sigma(W_f x_t + U_f h_j + b_f)$$

$$[i_t, o_t, u_t] = [\sigma(z_i), \sigma(z_o), \tanh(z_u)]$$

$$c_t = i_t \circ u_t + \sum_{j \in C(t)} f_{tj} \circ c_j$$

$$h_t = o_t \circ \tanh(c_t)$$

LSTM

$$z = Wx_t + Uh_{t-1} + b$$

$$[i_t, f_t, o_t, u_t] = [\sigma(z_i), \sigma(z_f), \sigma(z_o), \tanh(z_u)]$$

$$c_t = i_t \circ u_t + f_t \circ c_{t-1}$$

$$h_t = o_t \circ \tanh(c_t)$$

individual weight

Tree LSTM (individual weights)

$$z = Wx_t + U [h_j, \dots h_{j'}] + b$$

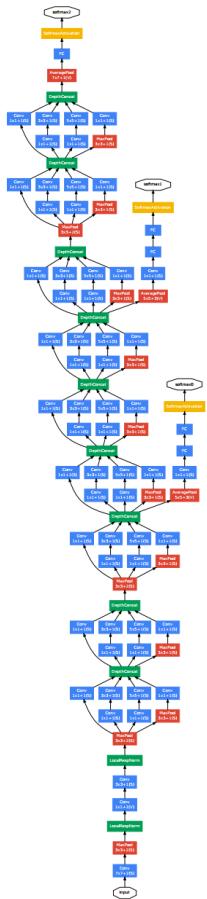
$$[i_t, o_t, [f_{tj} \dots f_{tj'}], u_t] = [\sigma(z_i), \sigma(z_o), [\sigma(z_{fj}) \dots \sigma(z_{fj'})], \tanh(z_u)]$$

$$c_t = i_t \circ u_t + \sum_{j \in C(t)} f_{tj} \circ c_{tj}$$

individual forget weight

$$h_t = o_t \circ \tanh(c_t)$$

Static and Dynamic Graphs



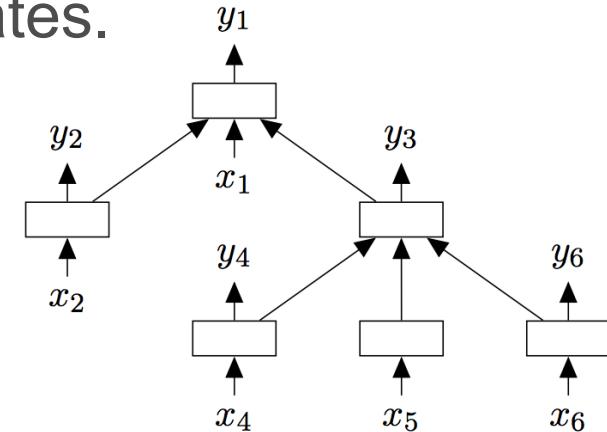
- **Static Graph**

Predefined concatenation of layers. Easy to do with a symbolic framework.

- **Dynamic Graph**

Implicitly defined via code in **imperative** framework.
Automatic differentiation for updates.

- Compute solution
- Update parameters



Links & Notebooks

gluon.mxnet.io/P07-C08-tree-lstm.html

gluon.mxnet.io/P14-C05-hybridize.html

Hybridization - (JIT for Deep Networks)

- **Forward Definition**

```
def hybrid_forward(self, F, x):  
    print('type(x): {}, F: {}'.format(type(x).__name__, F.__name__))  
    x = F.relu(self.fc1(x))  
    x = F.relu(self.fc2(x))  
    return self.fc3(x)
```

- **Without Hybridization**

Executes the full program (including print)

- **With Hybridization**

Executes only the compute graph that was **implicitly** defined. JIT for compute graphs (no need to parse repeatedly), as efficient as symbolic definition.



Outline

- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization**
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous**
- 9. (key, value) store

Optimization Basics

- **Optimization Problem**

$$\underset{w}{\text{minimize}} \frac{1}{m} \sum_{i=1}^m l(y_i, f(x_i, w)) + \lambda \|w\|_2^2$$

- **Stochastic Gradient Approximation**

- **Gradient on Minibatch**

$$g_B := \frac{1}{b} \sum_{i \in B} \partial_w l(y_i, f(x_i, w)) \text{ and } w \leftarrow \text{Optimizer}(w, g_B)$$

- **Update w with advanced first order solver**

(Adam, AdaGrad, Momentum, Eve, SGD, SGLD ...)



Optimization Basics

- Stochastic Gradient Descent (SGD)

$$w \leftarrow (1 - 2\eta\lambda)w - \eta g_b$$

- SGD with Momentum and Clipping

$$s \leftarrow \mu s + \eta \text{clip}(g_b) + \eta\lambda w$$

$$w \leftarrow w - s$$

Quite often learning rate is piecewise constant (this yields better accuracy in practice, albeit convergence is slower)

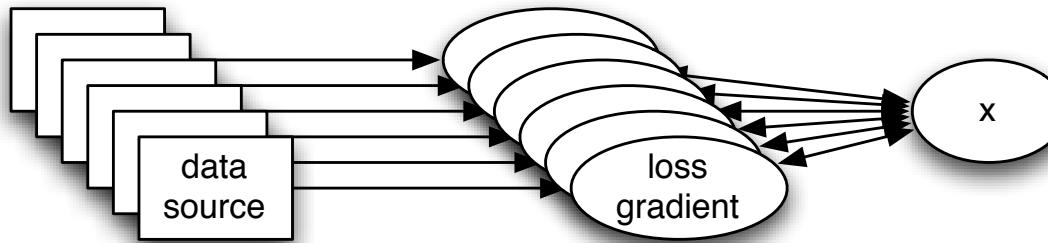
- Improved conditioning, momentum, precision ...

- SGD
- DCASGD
- NAG
- Adam
- SGLD
- AdaGrad
- RMSProp
- AdaDelta
- Ftrl
- Adamax
- Nadam

Parallelization

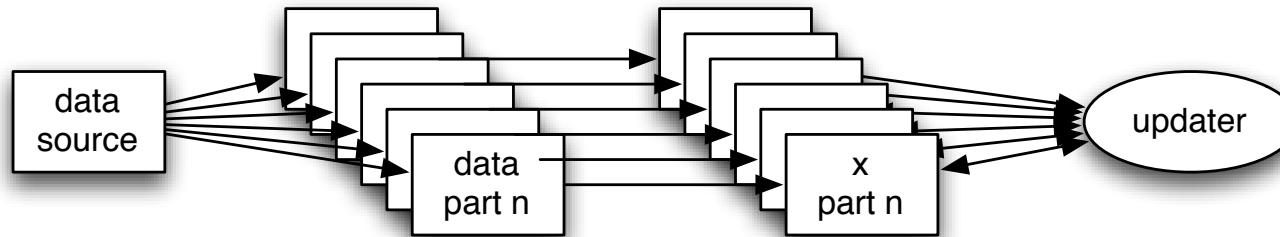
- **Data Parallel (easier & more efficient)**

Distribute data over multiple GPUs / CPUs / machines



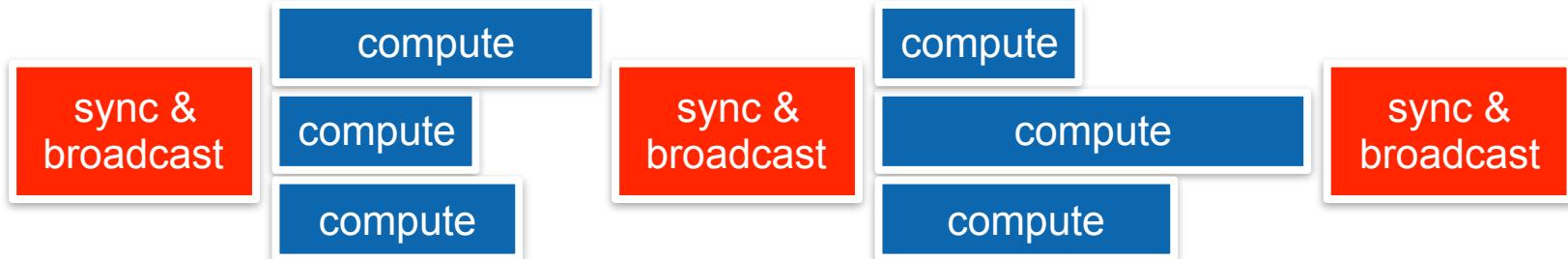
- **Model Parallel (only for huge models)**

Distribute parts of the computation over multiple GPUs / CPUs

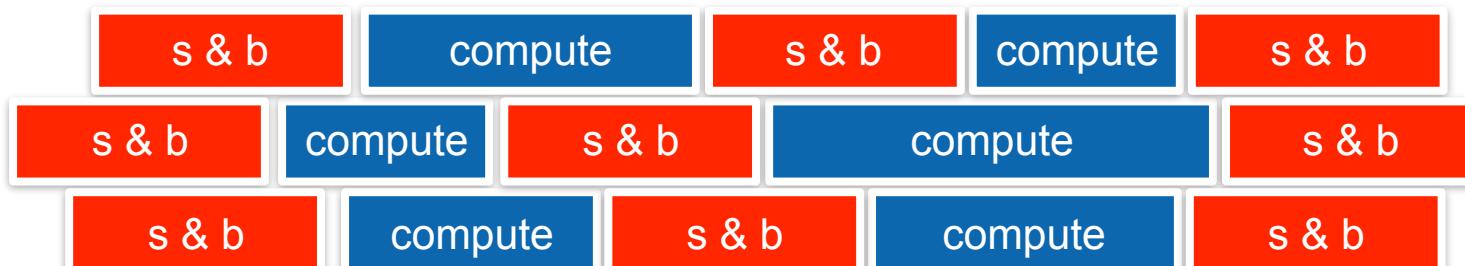


Parallelization

- **Synchronous - easy for homogeneous jobs**



- **Asynchronous - efficient for high time dispersion**



Parallelization - Pragmatic Strategy

- **Data Parallel & Synchronous**
 - Even for 512 GPUs you only need 32 P2.16xlarge (so the speed dispersion is negligible).
 - GPUs have plenty of RAM
(only an inefficient framework needs model parallel)
- **Algorithm Template** (shard data over machines)
 - Compute gradient on mini batch on each GPU
 - Aggregate gradients into (key,value) store
 - Broadcast them back to GPUs

Links & Notebooks

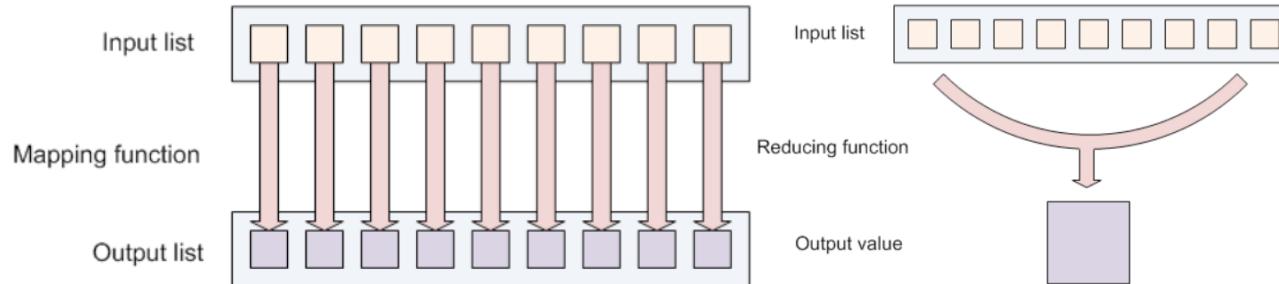
gluon.mxnet.io/P14-C02-multiple-gpus-scratch.html
gluon.mxnet.io/P14-C03-multiple-gpus-gluon.html

Outline

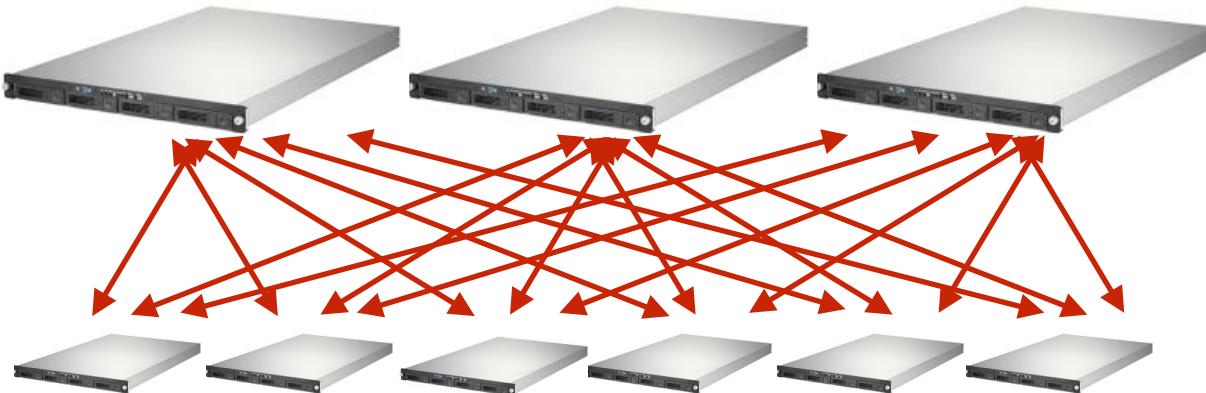
- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training**
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store**

Why (not) MapReduce?

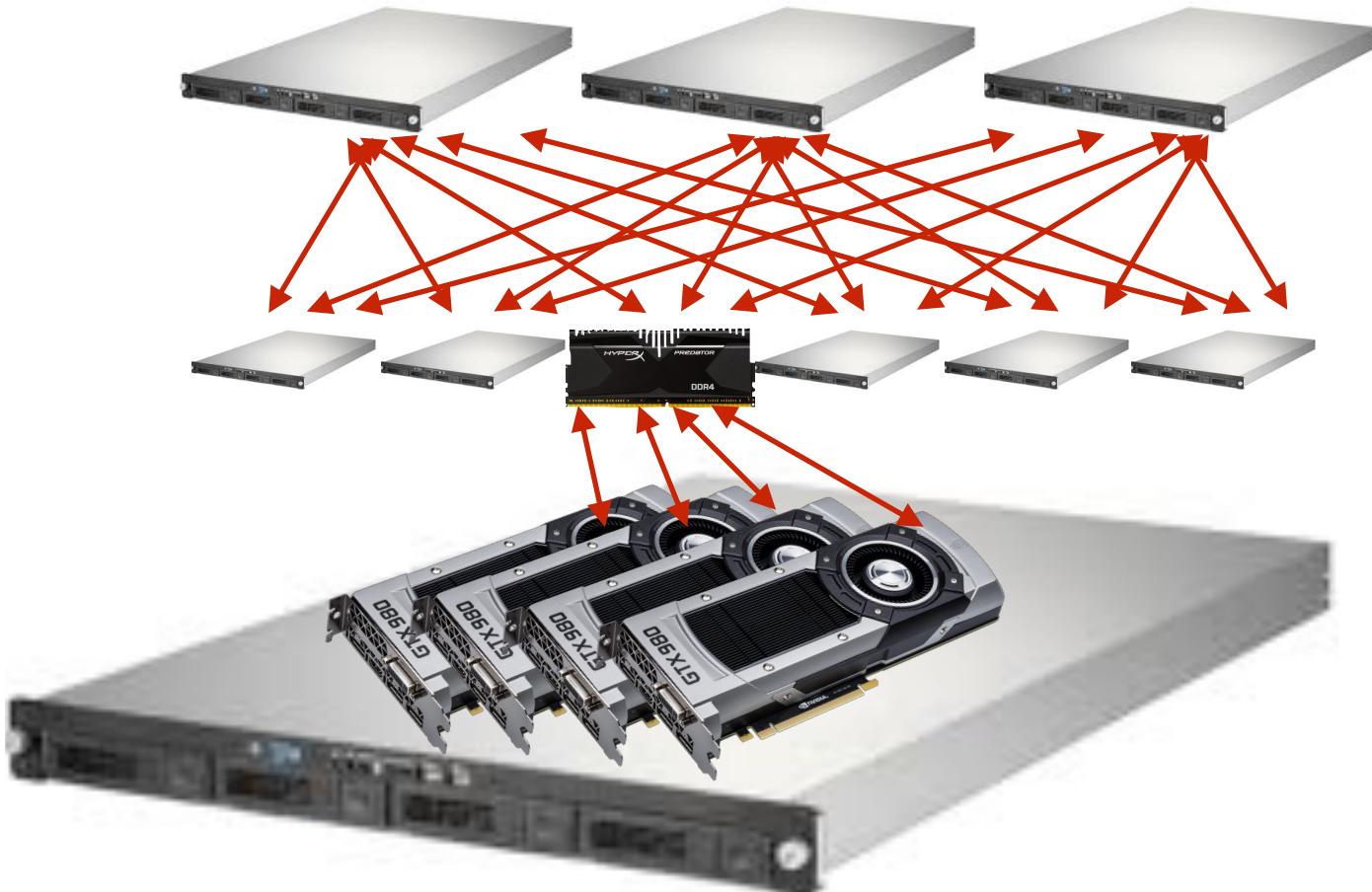
- **Map(key, value)**
Process subset of the data (and emit gradients)
- **Reduce(key, value)**
Aggregate gradients from machines and update parameters
- **Simple parameter exchange**
 - Great if only small number of syncs needed (**slow mixing**)
 - Huge overhead per iteration
 - Fully synchronous (**bad for large number of machines**)



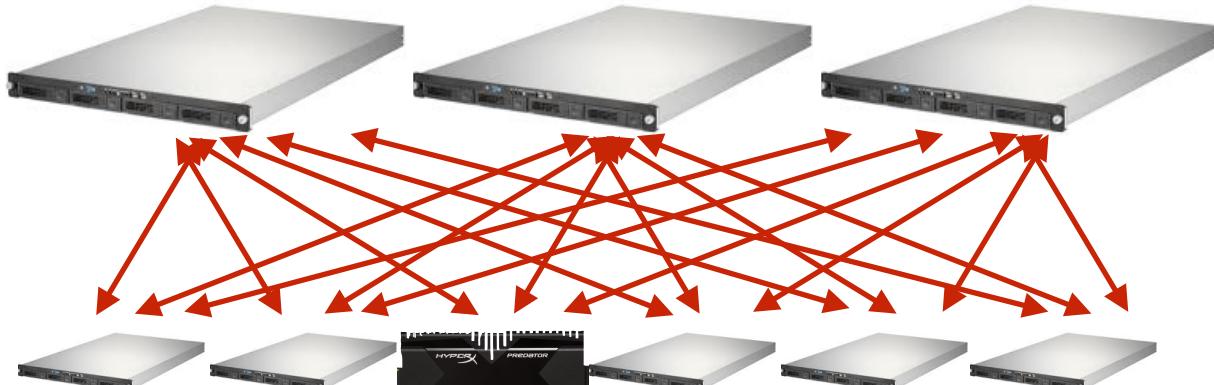
Parameter Server aka (key,value) store



Parameter Server aka (key, value) store



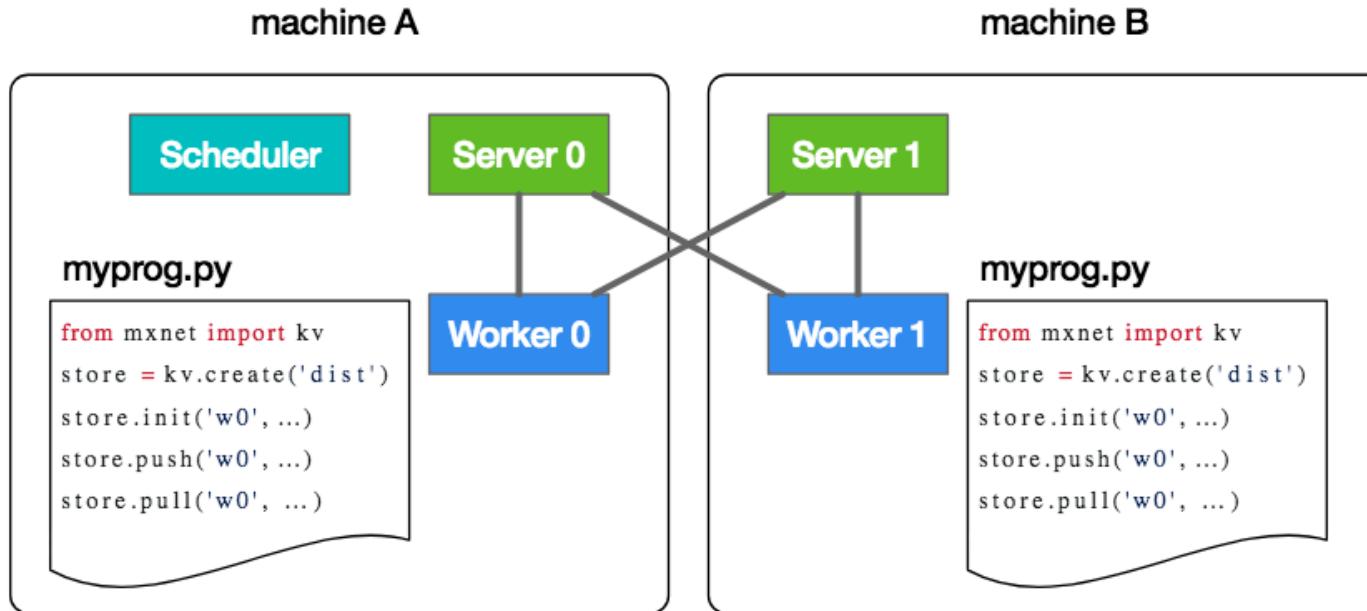
Parameter Server aka (key, value) store



1. Read a data partition
2. Pull the parameters (key, value)
3. Compute the gradient
4. Push the gradient (key, value)
5. Update the parameters (on server)

Multiple Machines (in practice)

- Each machine has server and worker job
- One machine has scheduler for task distribution



Links & Notebooks

gluon.mxnet.io/P14-C04-training-with-multi-machines.html
mxnet.io/api/python/kvstore.html

Notebooks github.com/zackchase/mxnet-the-straight-dope

- 1. Deep Learning 101
- 2. NDArray
- 3. Multilayer Perceptrons
- 4. Convolutional Neural Networks
- 5. Generative Adversarial Networks
- 6. LSTMs
- 7. TreeLSTMs
- 8. Distributed Optimization
- 9. Parallel Training
- 1. Getting started with MxNet
- 2. Automatic Differentiation
- 3. Training from scratch/Gluon
- 4. Gluon layers
- 5. More than one gradient
- 6. Operator fusion
- 7. Dynamic graphs
- 8. Synchronous / asynchronous
- 9. (key, value) store