Deep Learning Libraries

Badri Narayana Patro[1]

[1]Department of Electrical Engieering
Indian Institute of Technology, Kanpur

August 8, 2016

Plagiarism

- I have refered few slides of Prof. Fei-Fei Li and Prof. Anoop M. Namboodiri.
- This presentation is meant for educational purpose only.

Table of Contents

Table of Contents

Deep Network: CNN

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 60,000,000 params)
  - More data (10^6 vs 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week

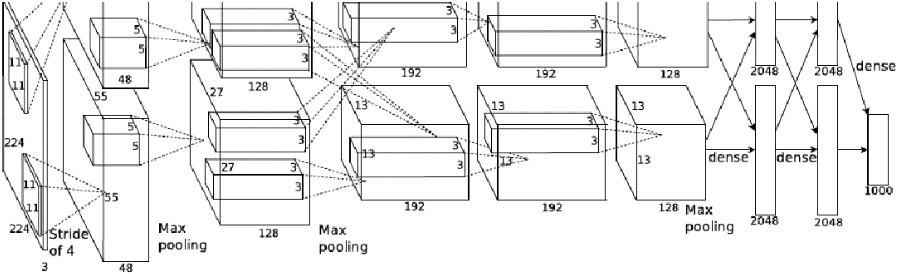
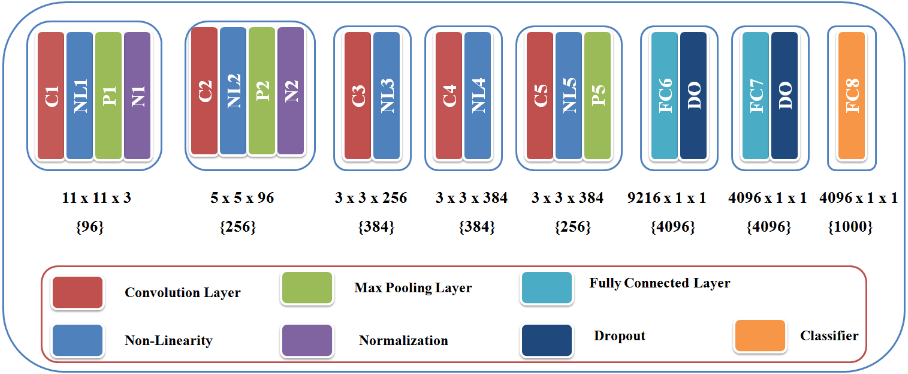
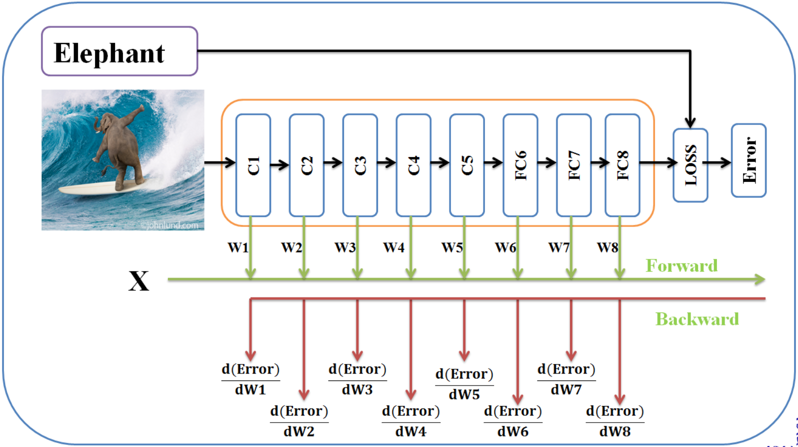


Image credit: A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks,NIPS 2012.

Deep Network: CNN



Training CNN



How to code for such a network?

- How do I code each type of layer(what is a layer ?)

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer
- Take care of normalization

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer
- Take care of normalization
- Take care of learning rates

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer
- Take care of normalization
- Take care of learning rates
- Code the optimizer/solver (SGD?)

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer
- Take care of normalization
- Take care of learning rates
- Code the optimizer/solver (SGD?)
- Several tricks to improve learning (Batch-norm, Dropout, etc.)

How to code for such a network?

- How do I code each type of layer(what is a layer ?)
- How do I connect them together
- Define a loss function
- Compute the gradient for each layer
- Code for updating the weights of each node/layer
- Take care of normalization
- Take care of learning rates
- Code the optimizer/solver (SGD?)
- Several tricks to improve learning (Batch-norm, Dropout, etc.)
- Do this on CPU/GPU/Distributed-systems

Would be nice if we could say

- Add a convolutional layer (params)

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer
- Add a max-pooling layer

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer
- Add a max-pooling layer
- Add a soft-max layer

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer
- Add a max-pooling layer
- Add a soft-max layer
- Use cross-entropy loss function

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer
- Add a max-pooling layer
- Add a soft-max layer
- Use cross-entropy loss function
- Train in batches of 64 images

Would be nice if we could say

- Add a convolutional layer (params)
- Add a ReLU activation layer
- Add a max-pooling layer
- Add a soft-max layer
- Use cross-entropy loss function
- Train in batches of 64 images
- Use the following learning rate schedule

Why Libraries

- Easily configure a network(Sequential and parallel conntction)

Why Libraries

- Easily configure a network(Sequential and parallel conntction)
- Easily Add and remove layers in a network

Why Libraries

- Easily configure a network(Sequential and parallel conntction)
- Easily Add and remove layers in a network
- Make use of GPUs, CPUs

Why Libraries

- Easily configure a network(Sequential and parallel conntction)
- Easily Add and remove layers in a network
- Make use of GPUs, CPUs
- Easily train a network

Why Libraries

- Easily configure a network(Sequential and parallel conntction)
- Easily Add and remove layers in a network
- Make use of GPUs, CPUs
- Easily train a network
- Change various parameter

Deep Learning Libraries

Some of Popular Libraries

- Torch [Lua]

Deep Learning Libraries

Some of Popular Libraries

- Torch [Lua]
- Theano [Python]

Deep Learning Libraries

Some of Popular Libraries

- Torch [Lua]
- Theano [Python]
- Caffe

Deep Learning Libraries

Some of Popular Libraries

- Torch [Lua]
- Theano [Python]
- Caffe
- TensorFlow

Deep Learning Libraries

Some of Popular Libraries

- Torch [Lua]
- Theano [Python]
- Caffe
- TensorFlow
- Matconvnet

Table of Contents

Caffe

- Created at UC Berkeley

Caffe

- Created at UC Berkeley
- Written in C++, Python wrappers.

Caffe

- Created at UC Berkeley
- Written in C++, Python wrappers.
- Creator:
  Yangqing Jia(UCB, Google, Facebook)

Caffe

- Created at UC Berkeley
- Written in C++, Python wrappers.
- Creator:
  Yangqing Jia(UCB, Google, Facebook)
- Current Lead Developer: Evan Shelhamer: (UCB)

Caffe

- Created at UC Berkeley
- Written in C++, Python wrappers.
- Creator:
  Yangqing Jia(UCB, Google, Facebook)
- Current Lead Developer: Evan Shelhamer: (UCB)
- In short
  - Simple and fast
  - Very easy for beginners/users

Caffe Pros / Cons

Pros:

- Good for feedforward networks
- Good for finetuning existing networks
- Train models without writing any code!
- Python interface is pretty useful!

Cons:

- Need to write C++ / CUDA for new GPU layers
- Not good for recurrent networks
- Not Good big networks (GoogLeNet, ResNet)

Table of Contents

Torch

- Developed at NYU and IDIAP
- Written in C and Lua.
- Current Version: Torch7 (1,3,5,7)
- Current Maintainers:
  - Ronan Collobert: Facebook [IDIAP]
  - Clement Farabert: Twitter [NYU]
  - Koray Kavukcuoglu: Google DeepMind
  - Soumith Chintala: Facebook
- In short
  - Flexible and fast

Torch

- Main data structure:
  - table in Lua; like object in Javascript
  - tensor in Torch: like Numpy array in Python
- Core Torch just has Tensors; others in packages
  - neural network layers in nn package
  - optimizers in optim package
  - nngraph for complex archiectures
- Torch layers are called modules; operate on Tensor

Easy to build and train neural nets

- Build a two-layer ReLU net

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network
- Use a softmax loss function

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network
- Use a softmax loss function
- Generate random data

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network
- Use a softmax loss function
- Generate random data
- Forward pass: compute scores and loss

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network
- Use a softmax loss function
- Generate random data
- Forward pass: compute scores and loss
- Backward pass: Compute gradients. Remember to set weight gradients to zero

Easy to build and train neural nets

- Build a two-layer ReLU net
- Get weights and gradient for entire network
- Use a softmax loss function
- Generate random data
- Forward pass: compute scores and loss
- Backward pass: Compute gradients. Remember to set weight gradients to zero
- Update: Make a gradient descent step

Torch: Training

```
--Step 0:----------- Load all the necessary packages----------------------------------
require 'cutorch'
require 'cunn'
require 'optim'
require 'gnuplot'
-- load file which have user defined functions
dofile './util.lua'

--step 1:------------Initilize variables-------------------------------------------------

--Batch size, input dimension, hidden dimension, number of classes
local batch_size, input_data_dim, No_of_neurons_hidden_layer, no_of_classes = 100, 1000, 100, 10

--step 2:------------Build Network-: two layer ReLU Network-----------------------------------

--note: variable cann't start with number, 2l_network== worng, correct one is l2_network.
local l2_networks = nn.Sequential()
l2_networks:add(nn.Linear(input_data_dim, No_of_neurons_hidden_layer))
l2_networks:add(nn.ReLU())
l2_networks:add(nn.Linear(No_of_neurons_hidden_layer, no_of_classes))

--step 3:-----------Get Weight and gradient for entire Network--------------------------------

--Collect all the weights and gradients in a single tensor
local weights, gradient_weights = l2_networks:getParameters()

--step 4:------------Use Softmax Loss function-------------------------------------------------

-- loss function are called criterions
local criterion= nn.CrossEntropyCriterion() --Softmax loss

--step 5:------------Generate Random Data------------------------------------------------------

-- Generate some Random input Data
local input_x = torch.randn(batch_size, input_data_dim)
local output_y= torch.Tensor(batch_size):random(no_of_classes)

--step 6:------------Forward pass -: compute scores and loss------------------------------------

local scores = l2_networks:forward(input_x)
local loss = criterion:forward(scores,output_y)

--step 7:------------Backward pass-: Compute gradients-----------------------------------------

gradient_weights:zero()-- set gradient weight to zero

local dscores = criterion:backward(scores,output_y) --ist find grad of loss functionn then
local dx = l2_networks:backward(input_x,dscores) -- find grad of network

--step 8:------------update weights-: Make a gradient descent step----------------------------

local learning_rate = 1e-3
weights:add(-learning_rate,gradient_weights)
```

Torch: Pros / Cons

Pros:

- Lots of modular pieces that are easy to combine
- Easy to write your own layer types and run on GPU
- Most of the library code is in Lua, easy to read
- Lots of pretrained models!

Cons:

- Lua
- Less plug-and-play than Caffe
- You usually write your own training code
- Not great for RNNs

Table of Contents

Theano

- Developed at Toronto and Montreal

Theano

- Developed at Toronto and Montreal
- BSD License

Theano

- Developed at Toronto and Montreal
- BSD License
- Written in Python.

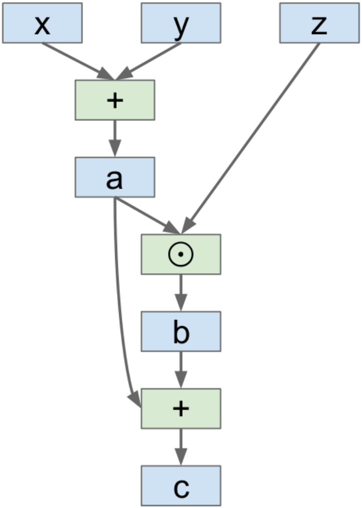
Theano

- Developed at Toronto and Montreal
- BSD License
- Written in Python.
- Current Maintainers: Yoshua Bengio's group at University of Montreal

Theano

- Developed at Toronto and Montreal
- BSD License
- Written in Python.
- Current Maintainers: Yoshua Bengio's group at University of Montreal
- In short
  - Complex code to do simple things
  - High-level wrappers: Keras, Lasagne

Theano: Computational Graphs




```python
import theano
import theano.tensor as T

# Define symbolic variables
x = T.matrix('x')
y = T.matrix('y')
z = T.matrix('z')

# Compute some other values symbolically
a = x + y
b = a * z
c = a + b

# Compile a function that computes c
f = theano.function(
        inputs=[x, y, z],
        outputs=c
    )

# Evaluate the compiled function
# on some real values
xx = np.random.randn(4, 5)
yy = np.random.randn(4, 5)
zz = np.random.randn(4, 5)
print f(xx, yy, zz)

# Repeat the same computation
# explicitly using numpy ops
aa = xx + yy
bb = aa * zz
cc = aa + bb
print cc
```

Image courtesy: Fei-Fei Li et al.

Theano: Computational Graphs code

```lua
require 'torch'
require 'nn'


-- Batch size, input dim, hidden dim, num classes
local N, D, H, C = 100, 1000, 100, 10

-- Build a one-layer ReLU network
local net = nn.Sequential()
net:add(nn.Linear(D, H))
net:add(nn.ReLU())
net:add(nn.Linear(H, C))

-- Collect all weights and gradients in a single Tensor
local weights, grad_weights = net:getParameters()

-- Loss functions are called "criterions"
local crit = nn.CrossEntropyCriterion()  -- Softmax loss

-- Generate some random input data
local x = torch.randn(N, D)
local y = torch.Tensor(N):random(C)

-- Forward pass: Compute scores and loss
local scores = net:forward(x)
local loss = crit:forward(scores, y)

-- Backward pass: compute gradients
grad_weights:zero()
local dscores = crit:backward(scores, y)
local dx = net:backward(x, dscores)

-- Make a gradient step
local learning_rate = 1e-3
weights:add(-learning_rate, grad_weights)
```

Image credit: Fei-Fei Li. et al.

Theano: Pros / Cons

Pros:

- Python + numpy
- Computational graph is nice abstraction
- RNNs fit nicely in computational graph
- High level wrappers (Keras, Lasagne) ease the pain

Cons:

- Raw Theano is somewhat low-level
- Error messages can be unhelpful
- Large models can have long compile times
- Much “fatter” than Torch; more magic
- Patchy support for pretrained models

Table of Contents

Tensor Flow

- Developed by Google

Tensor Flow

- Developed by Google
- Written in C++ and Python

Tensor Flow

- Developed by Google
- Written in C++ and Python
- BSD License

Tensor Flow

- Developed by Google
- Written in C++ and Python
- BSD License
- Released in 2015

Tensor Flow

- Developed by Google
- Written in C++ and Python
- BSD License
- Released in 2015
- In short
  - Good for large, distributed systems
  - Data and model parallelism

Tensor Flow

- Very similar to Theano - all about computation graphs

Tensor Flow

- Very similar to Theano - all about computation graphs
- Easy visualisation tool: TensorBoard

Tensor Flow

- Very similar to Theano - all about computation graphs
- Easy visualisation tool: TensorBoard
- Multi-GPU and multi-node training

Tensor Flow

- Very similar to Theano - all about computation graphs
- Easy visualisation tool: TensorBoard
- Multi-GPU and multi-node training
- TensorBoard makes it easy to visualise training process, in terms of loss, weights, activations, and even the whole DAG can be seen!

Tensor Flow

- Very similar to Theano - all about computation graphs
- Easy visualisation tool: TensorBoard
- Multi-GPU and multi-node training
- TensorBoard makes it easy to visualise training process, in terms of loss, weights, activations, and even the whole DAG can be seen!
- Matconvnet

TensorFlow: Tensorboard

Start Tensorboard server, and we get graphs!

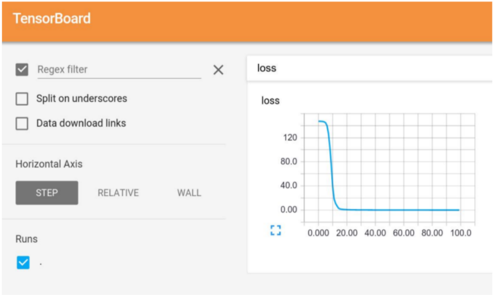


Image courtesy: Fei-Fei Li et al.

Tensorboard shows the graph!

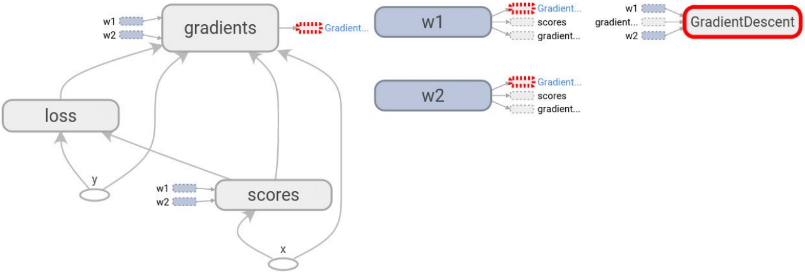


Image courtesy: Fei-Fei Li et al.

TensorFlow: TensorBoard

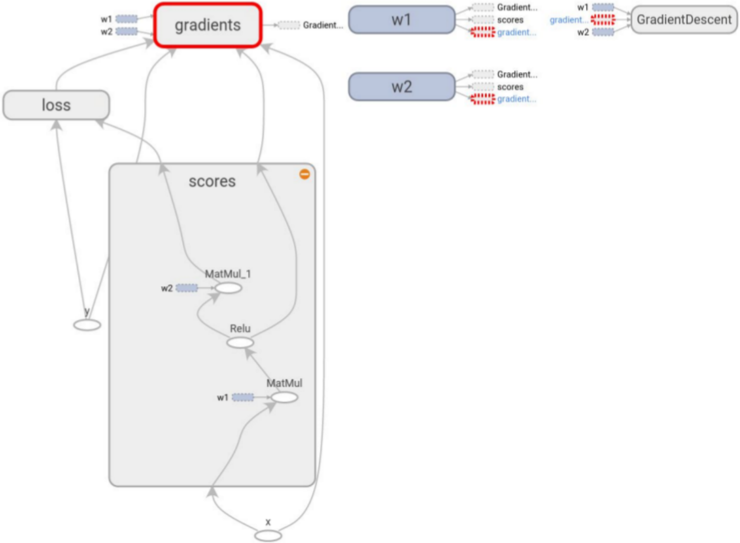
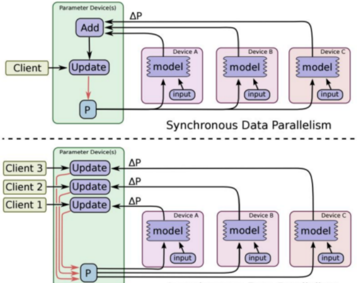


Image courtesy: Fei-Fei Li et al.

TensorFlow: Multi-GPU

Data parallelism:
synchronous or asynchronous



Asynchronous Data Parallelism

Model parallelism:
Split model across GPUs

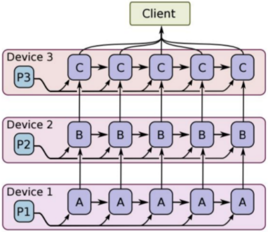
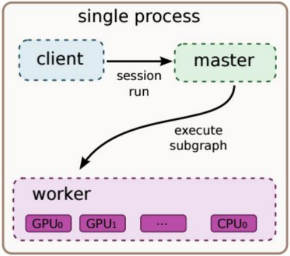


Image courtesy: Fei-Fei Li et al.

Single machine:
Like other frameworks



Many machines:
Not open source (yet) =(

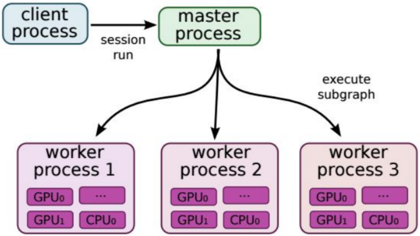
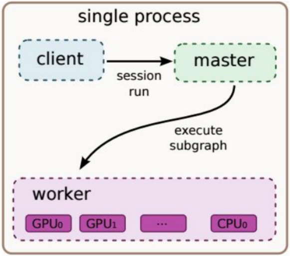


Image courtesy: Fei-Fei Li et al.

TensorFlow: Distributed

Single machine:
Like other frameworks



Many machines:
Not open source (yet) =(

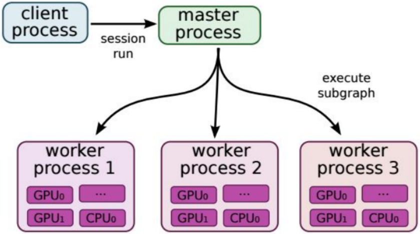


Image courtesy: Fei-Fei Li et al.

Tensor Flow

Pros:

- Python + numpy
- Computational graph abstraction, like Theano; great for RNNs
- Much faster compile times than Theano
- Slightly more convenient than raw Theano?
- TensorBoard for visualization
- Data AND model parallelism; best of all frameworks

Cons:

- Distributed models, but not open-source yet
- Slower than other frameworks right now
- Much “fatter” than Torch; more magic
- Not many pretrained models

Comparison

	Caffe	Torch	Theano	TensorFlow
Language	C++,Python	Lua	Python	Python
Pretrained	Very good	Very good	OK	Poor
Multi-GPU: Data parallel	Yes	Yes	Ok	Yes
Multi-GPU: Model parallel	No	Yes	Experimental	Yes (best)
Readable	Yes(C++)	Yes(lua)	V. Poor	V. Poor
Good at RNN	No	Mediocre	Yes	Yes (best)
Visualization	No	OK	No	Yes
Automatic differentiation	No	Yes(nngraph)	Yes	Yes

Use Cases

- Use pretrained models for features? Caffe

Use Cases

- Use pretrained models for features? Caffe
- Fine-tune AlexNet for new classes? Use Caffe

Use Cases

- Use pretrained models for features? Caffe
- Fine-tune AlexNet for new classes? Use Caffe
- Segmentation? (Classify every pixel)
  - Need pretrained model (Caffe, Torch, Lasagna)
  - If loss function exists in Caffe: Use Caffe
  - If you want to write your own loss: Use Torch

Use Cases

- Use pretrained models for features? Caffe
- Fine-tune AlexNet for new classes? Use Caffe
- Segmentation? (Classify every pixel)
  - Need pretrained model (Caffe, Torch, Lasagna)
  - If loss function exists in Caffe: Use Caffe
  - If you want to write your own loss: Use Torch
- Object Detection?
  - Need pretrained model (Torch, Caffe, Lasagne)
  - Need lots of custom imperative code (NOT Lasagne)
  - Use Caffe + Python or Torch

Use Cases

- Use pretrained models for features? Caffe
- Fine-tune AlexNet for new classes? Use Caffe
- Segmentation? (Classify every pixel)
  - Need pretrained model (Caffe, Torch, Lasagna)
  - If loss function exists in Caffe: Use Caffe
  - If you want to write your own loss: Use Torch
- Object Detection?
  - Need pretrained model (Torch, Caffe, Lasagne)
  - Need lots of custom imperative code (NOT Lasagne)
  - Use Caffe + Python or Torch
- Implement BatchNorm?
  - Don't want to derive gradient? Theano or TensorFlow
  - Implement efficient backward pass? Use Torch

Use Cases

- Image Captioning with finetuning?
  - Need pretrained models (Caffe, Torch, Lasagne)
  - Need RNNs (Torch or Lasagne)
  - Use Torch or Lasagna

Use Cases

- Image Captioning with finetuning?
  - Need pretrained models (Caffe, Torch, Lasagne)
  - Need RNNs (Torch or Lasagne)
  - Use Torch or Lasagna
- Language modeling with new RNN structure?
  - Need easy recurrent nets (NOT Caffe, Torch)
  - No need for pretrained models
  - Use Theano or TensorFlow

Use Cases

- Image Captioning with finetuning?
  - Need pretrained models (Caffe, Torch, Lasagne)
  - Need RNNs (Torch or Lasagne)
  - Use Torch or Lasagna
- Language modeling with new RNN structure?
  - Need easy recurrent nets (NOT Caffe, Torch)
  - No need for pretrained models
  - Use Theano or TensorFlow
- Huge parallelism, distributed training required? Tensorflow

Use Cases

- Image Captioning with finetuning?
  - Need pretrained models (Caffe, Torch, Lasagne)
  - Need RNNs (Torch or Lasagne)
  - Use Torch or Lasagna
- Language modeling with new RNN structure?
  - Need easy recurrent nets (NOT Caffe, Torch)
  - No need for pretrained models
  - Use Theano or TensorFlow
- Huge parallelism, distributed training required? Tensorflow
- Simple parallelism required? Torch or Caffe

Use Cases

- Image Captioning with finetuning?
  - Need pretrained models (Caffe, Torch, Lasagne)
  - Need RNNs (Torch or Lasagne)
  - Use Torch or Lasagna
- Language modeling with new RNN structure?
  - Need easy recurrent nets (NOT Caffe, Torch)
  - No need for pretrained models
  - Use Theano or TensorFlow
- Huge parallelism, distributed training required? Tensorflow
- Simple parallelism required? Torch or Caffe
- Reinforcement learning? Torch or Tensorflow

Recommendation

- Feature extraction / finetuning existing models: Use Caffe

Recommendation

- Feature extraction / finetuning existing models: Use Caffe
- Complex uses of pretrained models: Use Lasagne or Torch

Recommendation

- Feature extraction / finetuning existing models: Use Caffe
- Complex uses of pretrained models: Use Lasagne or Torch
- Write your own layers: Use Torch

Recommendation

- Feature extraction / finetuning existing models: Use Caffe
- Complex uses of pretrained models: Use Lasagne or Torch
- Write your own layers: Use Torch
- Crazy RNNs: Use Theano or Tensorflow

Recommendation

- Feature extraction / finetuning existing models: Use Caffe
- Complex uses of pretrained models: Use Lasagne or Torch
- Write your own layers: Use Torch
- Crazy RNNs: Use Theano or Tensorflow
- Huge model, need model parallelism: Use TensorFlow

Thank you

Thank you.