Hands on: Tools for Deep Learning

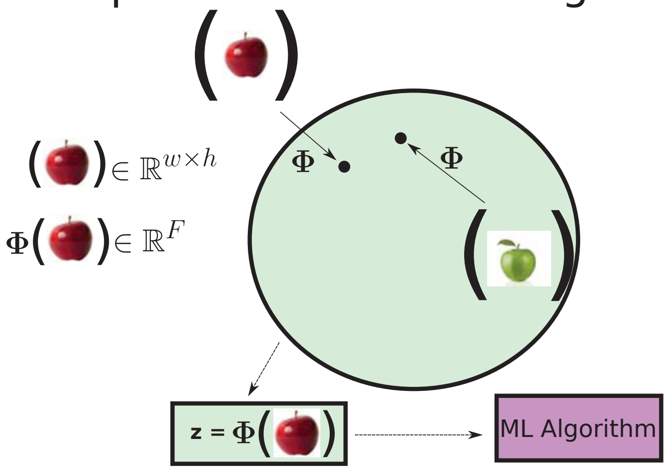
Tobias Springenberg
Machine Learning Lab

University of Freiburg

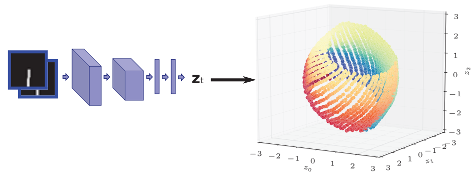
November 10, 2016

Machine Learning Groups I work with in Freiburg

Representation Learning



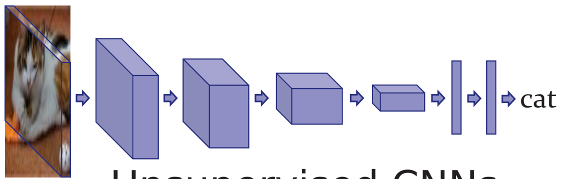
Reinforcement Learning and Control



Joschka Boedecker

Martin Riedmiller

Computer Vision



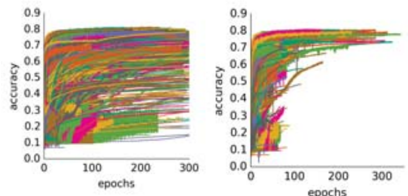
Unsupervised CNNs

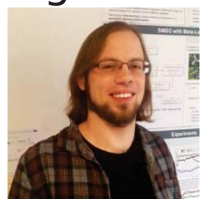
Alexey Dosovitskiy

Generative models

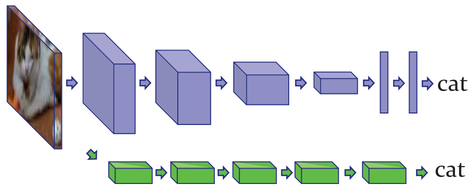
Thomas Brox

Automated Machine Learning





speed up hyperparameter search



simplify networks

Outline

Ich werde ...

Outline

Ich werde ...

- Differenzen zwischen verschiedenen “deep learning frameworks” aufzeigen

Outline

Ich werde ...

- Differenzen zwischen verschiedenen “deep learning frameworks” aufzeigen
- Tipps für Neuanfänger im “deep learning” geben

Outline

Ich werde ...

- Differenzen zwischen verschiedenen “deep learning frameworks” aufzeigen
- Tipps für Neuanfänger im “deep learning” geben
- Die folgenden Folien sind in englischer Sprache

Deep Learning Frameworks

An explosion of frameworks over the last years ...

Deep Learning Frameworks

Packages

C++

- Caffe (Python interface)
- MxNet
  (Python, R, Julia interfaces)

Python

- Theano
  - Lasagne
  - Keras
- Tensorflow
  - Keras
- Neon

Lua

- Torch

... and many more

Deep Learning Frameworks

Packages

C++

- Caffe (Python interface)
- MxNet
  (Python, R, Julia interfaces)

Python

- Theano
  - Lasagne
  - Keras
- Tensorflow
  - Keras
- Neon

Lua

- Torch

... and many more

Deep Learning Frameworks

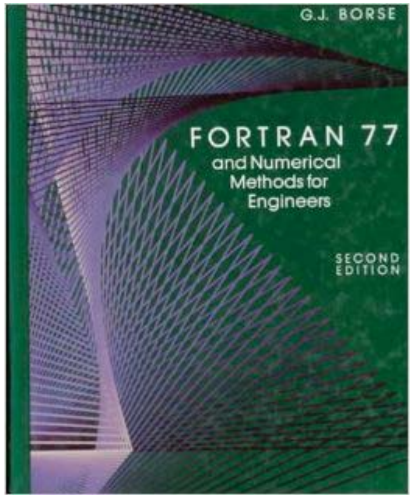
An explosion of tools and frameworks over the last years ...

Deep Learning Frameworks

An explosion of tools and frameworks over the last years ...
Why is this happening now ?

A story of increasing abstraction ...

It happened before for scientific code



1960s

Standard Specification

BLAS
LINPACK
LAPACK
GSL

1980s

matplotlib
python
SciPy
NumPy

Eigen

Today

A story of increasing abstraction in Deep Learning ...

The same is happening in deep learning

- Early 90s: experts only
  - take BLAS, write specialized C++ code to implement your network

A story of increasing abstraction in Deep Learning ...

The same is happening in deep learning

- Early 90s: experts only
  - take BLAS, write specialized C++ code to implement your network
- 2012: lots of re-implementations of AlexNet (ImageNet winner)
  - Caffe, cuda-convnet used as black-box in Computer Vision
  - limited modularity in the scalable solutions
  - Theano is flexible and works well for researchers on small problems

A story of increasing abstraction in Deep Learning ...

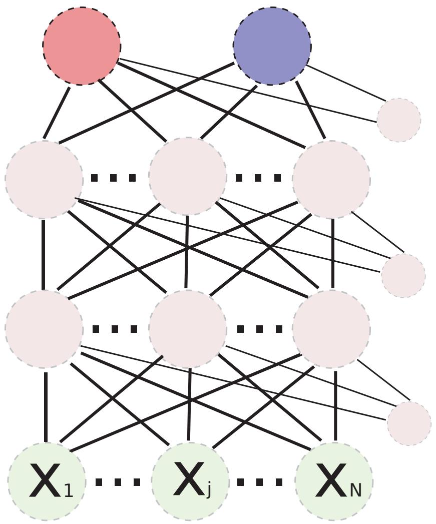
The same is happening in deep learning

- Early 90s: experts only
  - take BLAS, write specialized C++ code to implement your network
- 2012: lots of re-implementations of AlexNet (ImageNet winner)
  - Caffe, cuda-convnet used as black-box in Computer Vision
  - limited modularity in the scalable solutions
  - Theano is flexible and works well for researchers on small problems
- Since 2014: easy to use scalable frameworks
  - community figured out how to write faster device independent code
  - NVIDIA cuDNN forms the basis for GPU computations (all frameworks use it)
  - Several frameworks with **full automatic differentiation** exist
  - scaling **from research code** to **production** in one framework is now possible

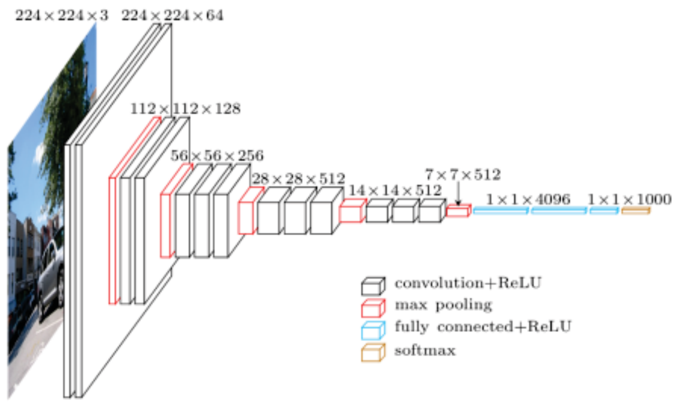
Deep Learning Models

Multiple ways to define a parametric function $f_\theta(x)$ with a network:

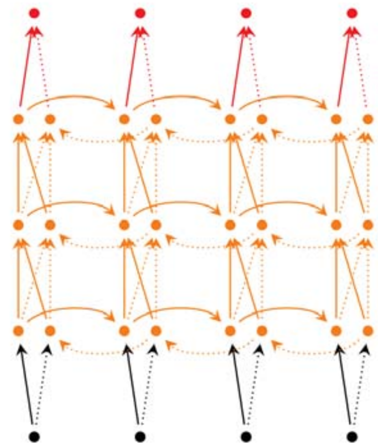
Fully connected



Convolutional



Recurrent



Deep Learning Models

Tip 

- if you are just starting
- forget CNN / RNNs, implement a simple NN **from scratch** once!
- the exercise I give my students:
  `https://github.com/mllfreiburg/dl_lab_2016`

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \dots (\mathbf{x}_N, \mathbf{y}_N)\}$

$$\min_\theta L(f_\theta, D) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \tag{1}$$

where $l(\cdot, \cdot)$ is a per example loss

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$

$$\min_\theta L(f_\theta, D) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \tag{1}$$

  where $l(\cdot, \cdot)$ is a per example loss
- For regression often use the squared loss:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \sum_{j=1}^{M} (f_{j,\theta}(\mathbf{x}) - y_j)^2$$

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$

$$\min_\theta L(f_\theta, D) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \tag{1}$$

where $l(\cdot, \cdot)$ is a per example loss
- For regression often use the squared loss:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \sum_{j=1}^{M} (f_{j,\theta}(\mathbf{x}) - y_j)^2$$

- The simplest approach to finding $\min_\theta L(f_\theta, D)$ is gradient descent

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$

$$\min_\theta L(f_\theta, D) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \tag{1}$$

  where $l(\cdot, \cdot)$ is a per example loss
- For regression often use the squared loss:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \sum_{j=1}^{M} (f_{j,\theta}(\mathbf{x}) - y_j)^2$$

- The simplest approach to finding $\min_\theta L(f_\theta, D)$ is gradient descent

$\rightarrow$ iteratively follow gradient to minimum $\theta^{t+1} = \theta^t - \gamma_t \frac{\partial L(f_\theta, D)}{\partial \theta}$

Training supervised feed-forward neural networks

- Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- Via minimizing the empirical risk on a dataset $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_N, \mathbf{y}_N)\}$

$$\min_\theta L(f_\theta, D) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \tag{1}$$

  where $l(\cdot, \cdot)$ is a per example loss
- For regression often use the squared loss:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \sum_{j=1}^{M} (f_{j,\theta}(\mathbf{x}) - y_j)^2$$

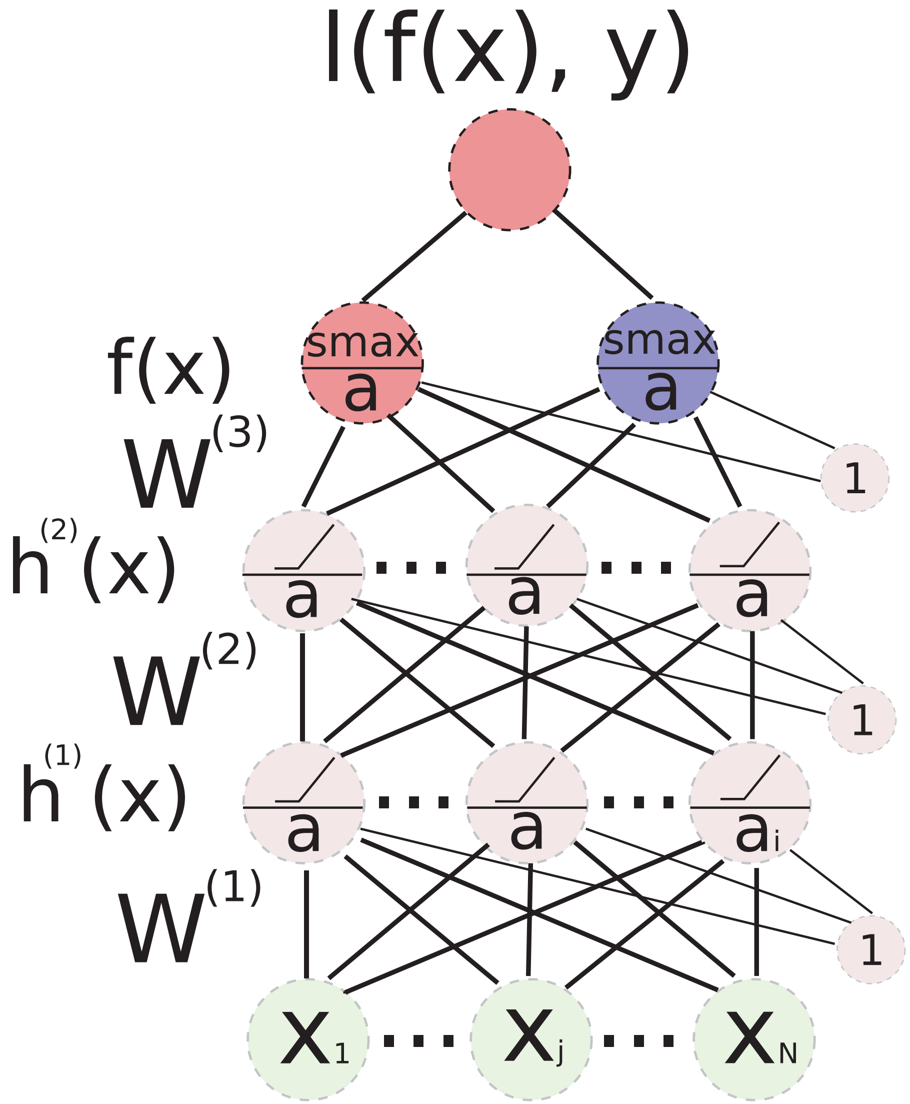
- The simplest approach to finding $\min_\theta L(f_\theta, D)$ is gradient descent

$\rightarrow$ iteratively follow gradient to minimum $\theta^{t+1} = \theta^t - \gamma_t \frac{\partial L(f_\theta, D)}{\partial \theta}$

$\rightarrow$ Computing the gradient is expensive if the training dataset is large!

Neural Network backward pass

→ Now how do we compute the gradient for a network ?



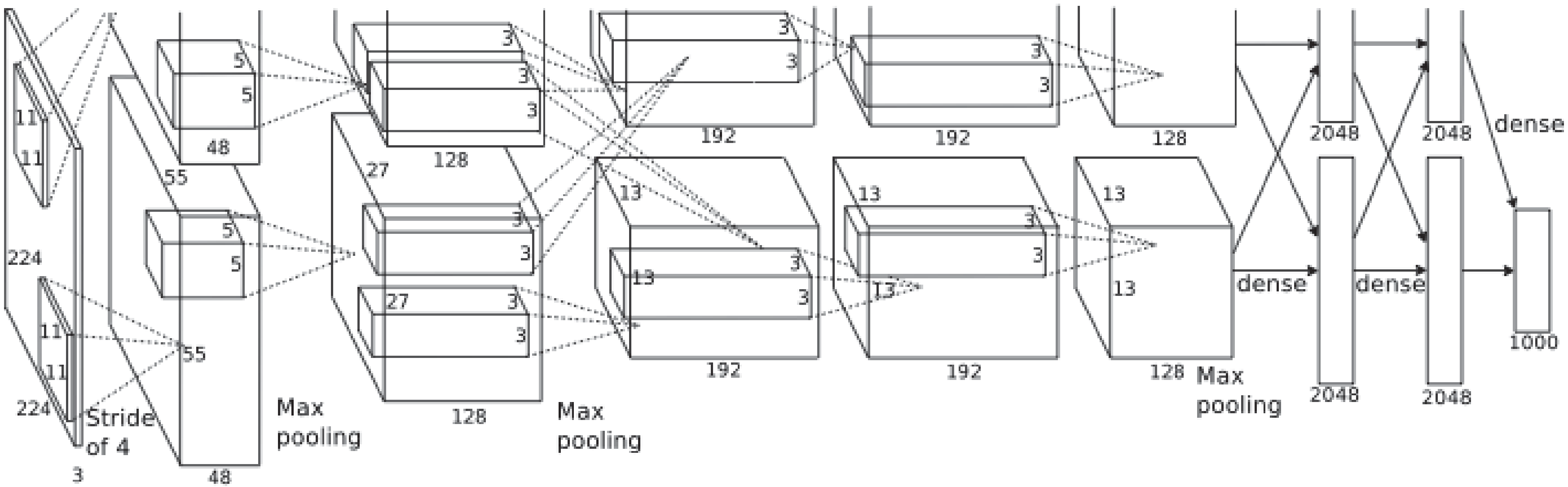
- Use the chain rule:

$$\frac{\partial a(b(x))}{\partial x} = \frac{\partial a(b(x))}{\partial b(x)} \frac{\partial b(x)}{\partial x}$$

- compute gradient on output layer
- then backpropagate to get $\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}}$

Neural Network backward pass

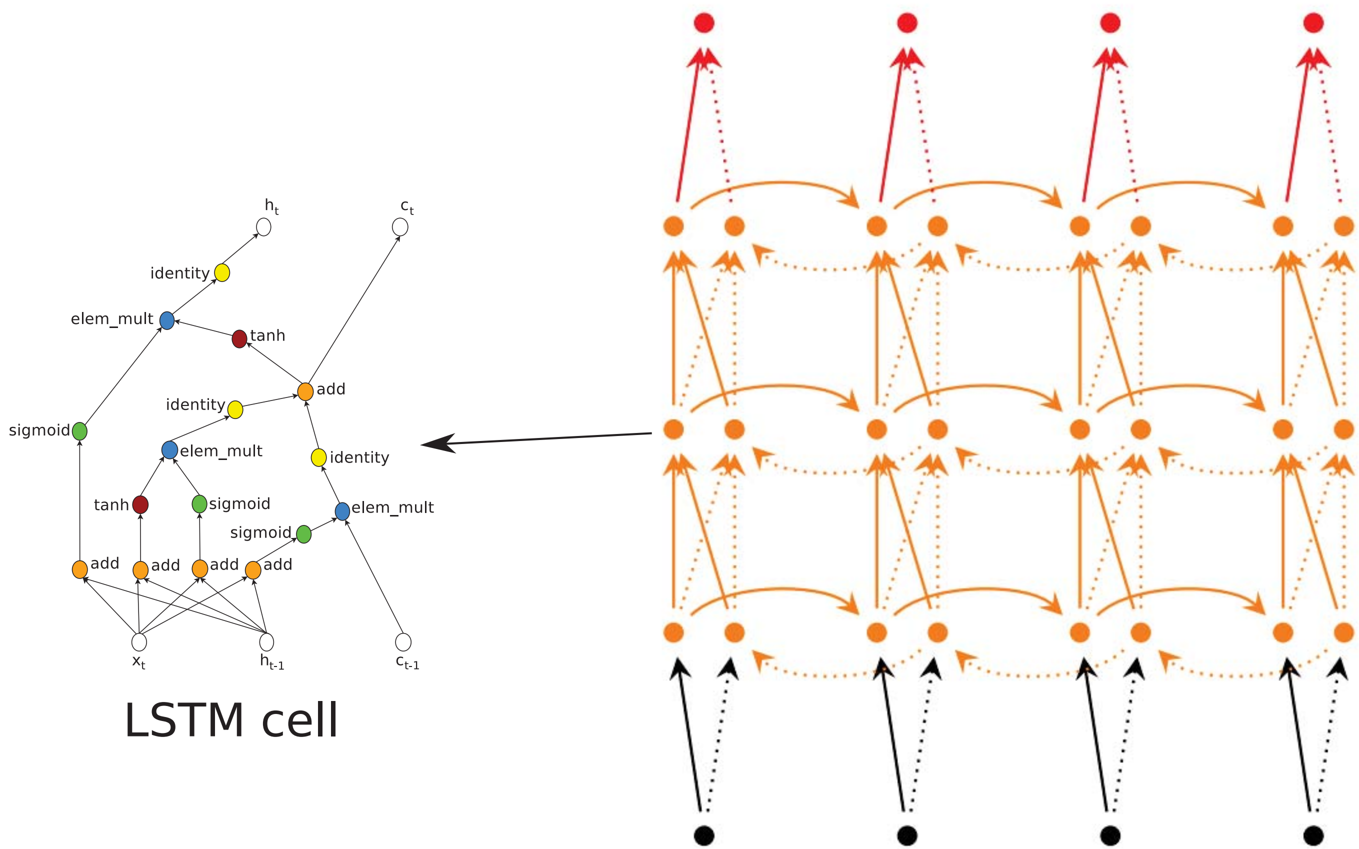
- Can you imagine implementing a backward pass for AlexNet ?



taken from (Krizhevsky, 2012)

Neural Network backward pass

- And what about an LSTM ? Not too much fun!

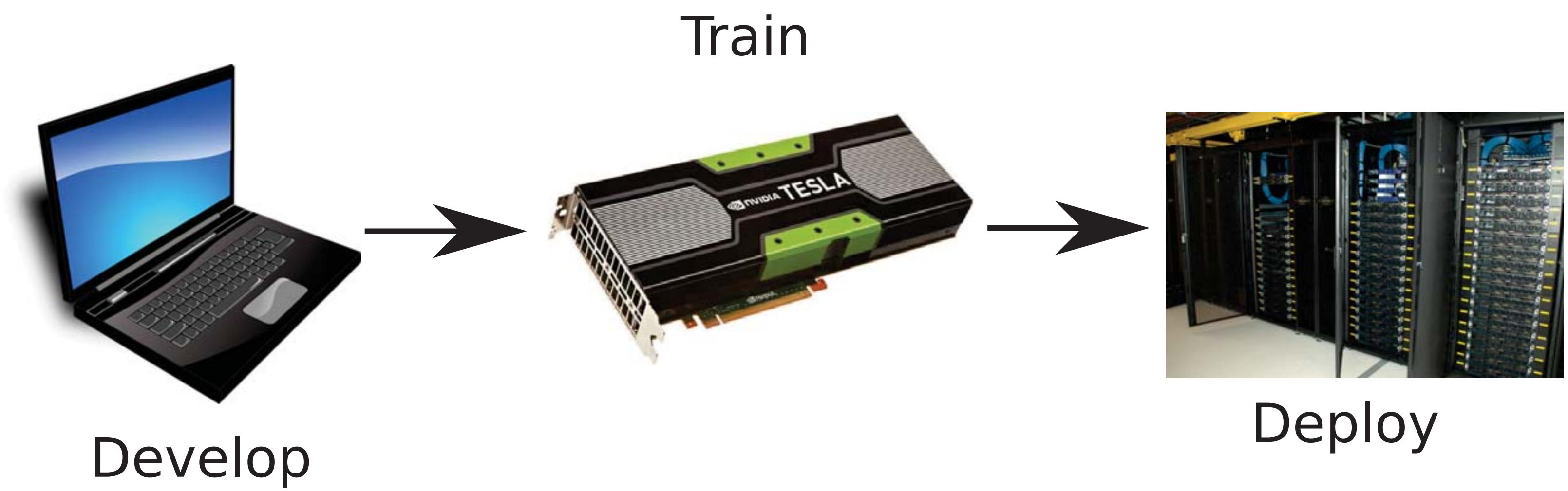


LSTM cell

- good luck implementing that without bugs first try ...

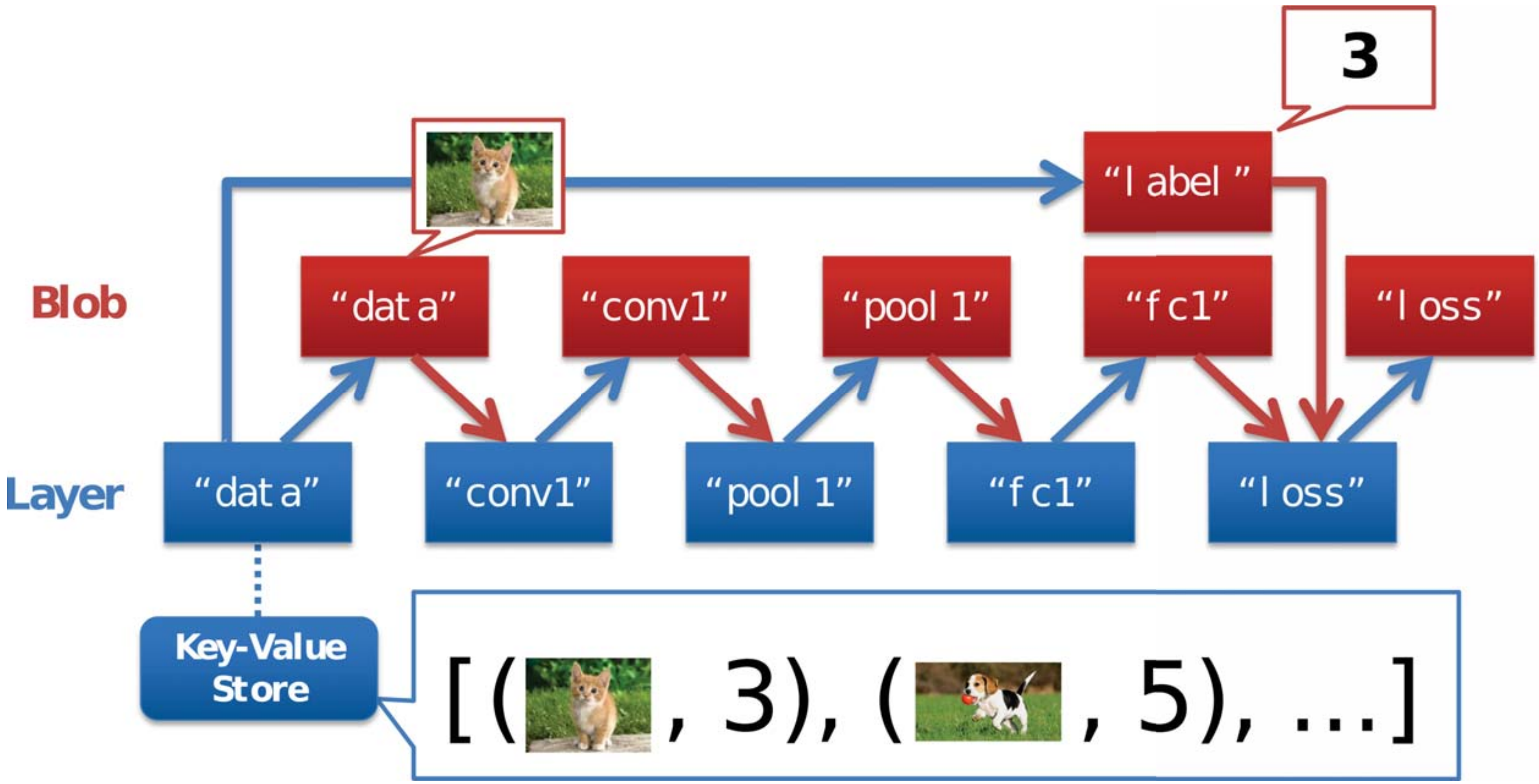
Frameworks to the rescue

- The main purpose of all DL frameworks is: **compute derivatives for you**
- And allow execution on **many different devices**



Caffe

Traditional layer wise computation in **Caffe**



Caffe

Network definition (train_val.prototxt)

```
name: "AlexNet"
layer {
 name: "data"
 type: "Input"
 top: "data"
 input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
layer {
 name: "conv1"
 type: "Convolution"
 bottom: "data"
 top: "conv1"
 param { lr_mult: 1  decay_mult: 1 }
 param { lr_mult: 2  decay_mult: 0 }
 convolution_param {
   num_output: 96  kernel_size: 11  stride: 4 }
}
layer { name: "relu1” type: "ReLU"
   bottom: "conv1" top: "conv1" }
```

solver.prototxt

```
net: "train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 100000
display: 20
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/my_model"
```

On commandline run

```
./caffe train --solver=solver.prototxt
```

Caffe with python (fast but limited flexibility)

Network definition (in python)

```
from caffe import params as P

def lenet(lmdb, batch_size):
    # Define a CNN that mimics the LeNet network
    n = caffe.NetSpec()
    n.data, n.label = L.Data(batch_size=batch_size,
                         backend=P.Data.LMDB, source=lmdb,
                         transform_param=dict(scale=1./255), ntop=2)
    n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20)
    n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50)
    n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.ip1 = L.InnerProduct(n.pool2, num_output=500)
    n.relu1 = L.ReLU(n.ip1, in_place=True)
    n.ip2 = L.InnerProduct(n.relu1, num_output=10)
    n.loss = L.SoftmaxWithLoss(n.ip2, n.label)
    return n.to_proto()

with open('conv.prototxt', 'w') as f:
    f.write(str(lenet('images_database_lmdb', 64)))
```

training in python

```
import caffe
caffe.set_mode_gpu()
net = caffe.Net('conv.prototxt', caffe.TEST)
solver = caffe.SGDSolver('solver.prototxt')
for i in range(iterations):
    solver.net.forward()
    solver.step(1)
```

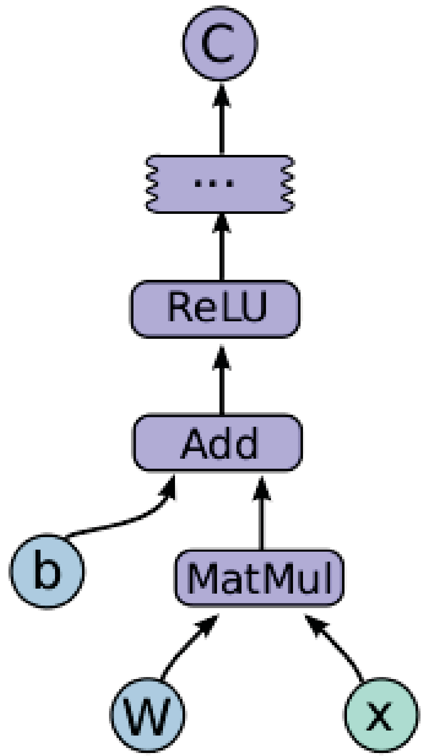
On commandline run (still works)

```
./caffe train --solver=solver.prototxt
```

Graph based frameworks

- Graph based frameworks (theano,tensorflow) give fine grained control over computations in a graph
- The graph can then be executed on different devices, optimized, etc.
- Let us look at a simple example (tensorflow)

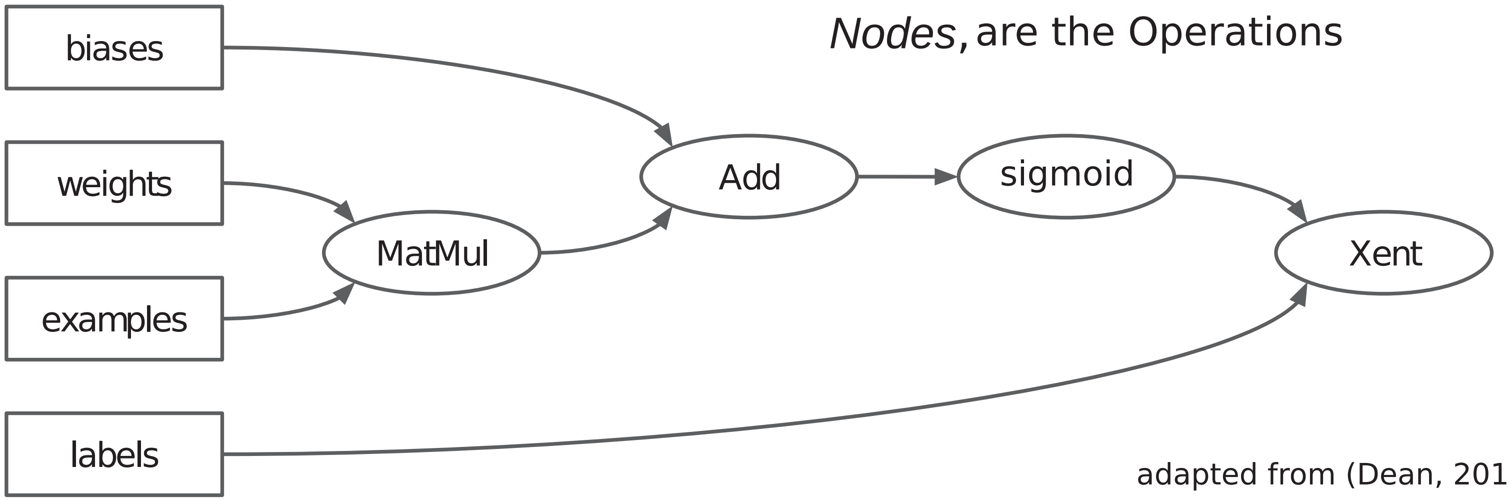
```
tf.nn.sigmoid(tf.matmul(x, weights) + biases)
```



Graph based frameworks

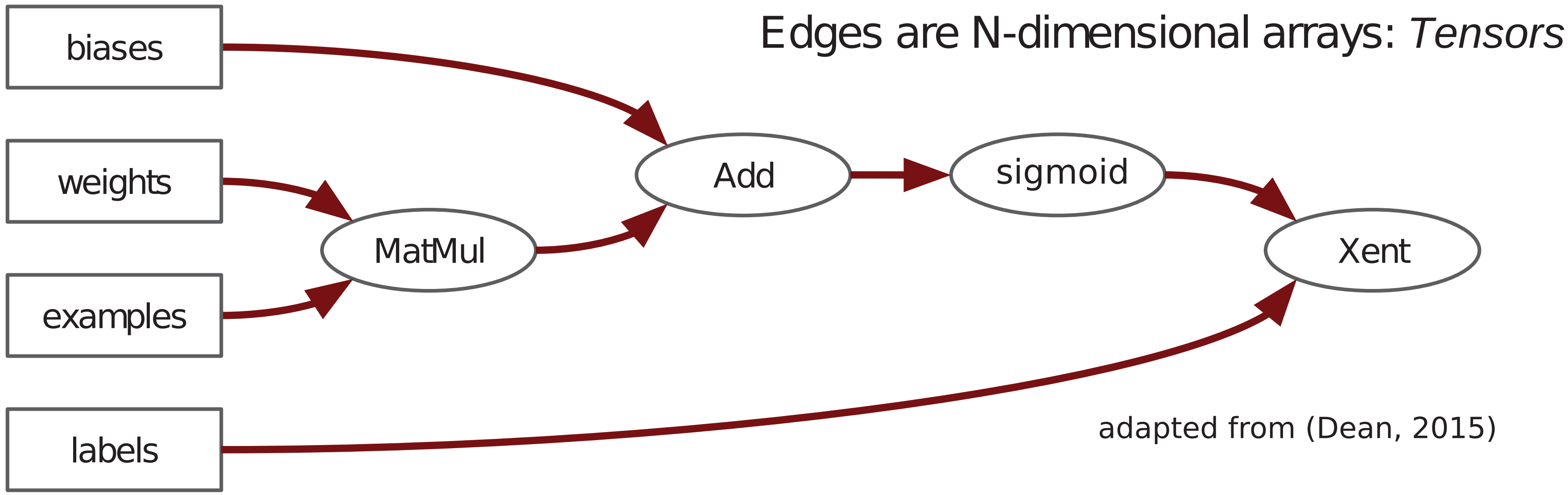
Convert code to a graph

Nodes, are the Operations



adapted from (Dean, 2015)

Graph based frameworks



adapted from (Dean, 2015)

Graph based frameworks

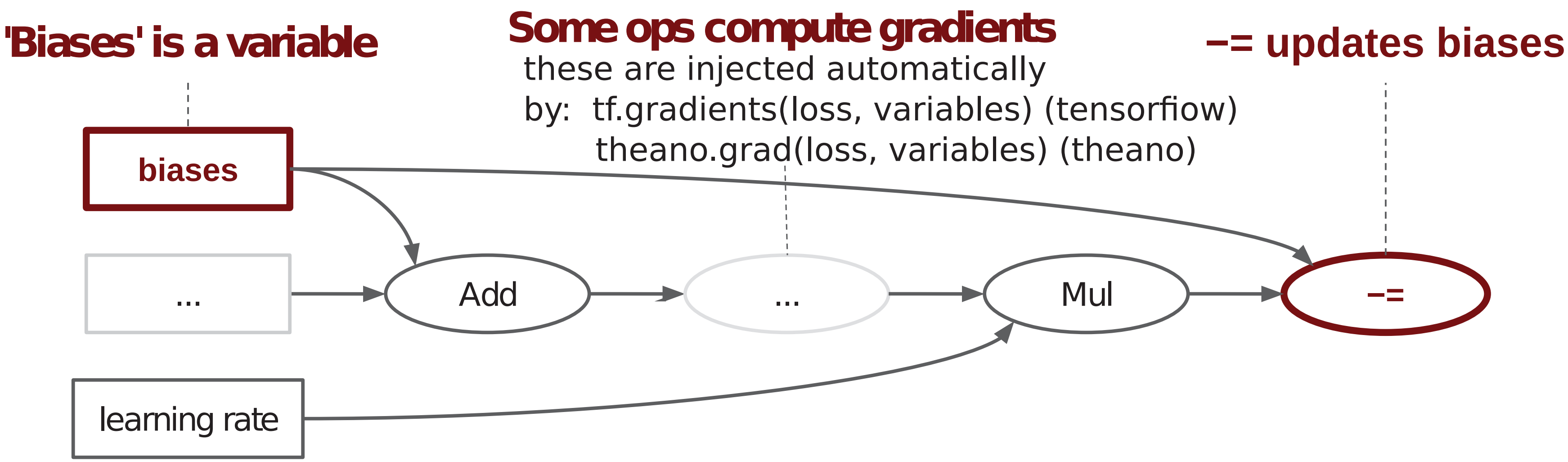
The graph can have state:

```
loss = binary_cross_entropy(out, y)
updates = tf.gradients(loss, b)
b.assign(b + learning_rate * updates)
```

Graph based frameworks

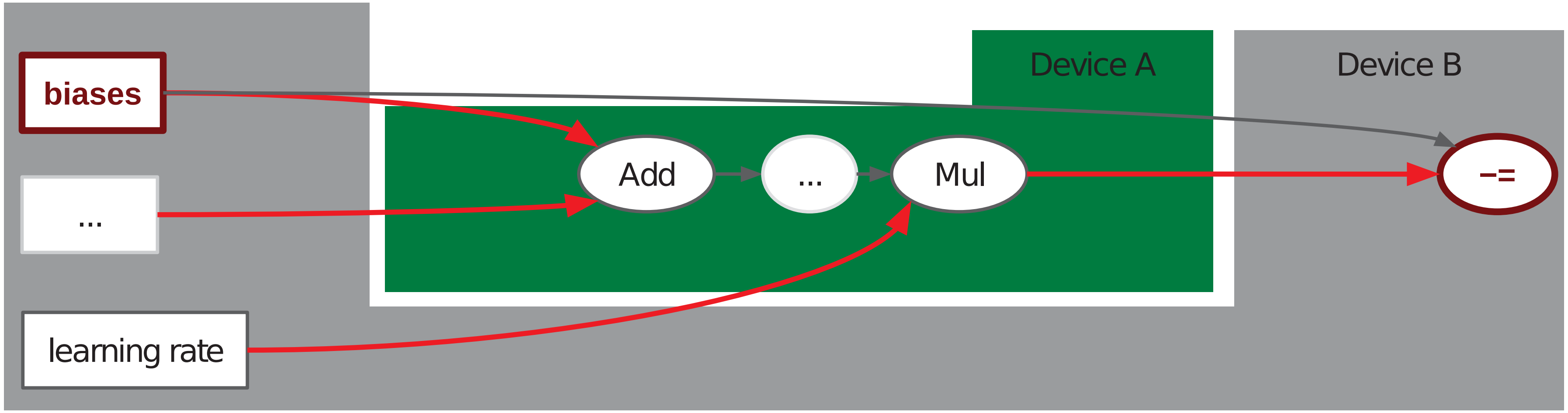
The graph can have state:

```
loss = binary_cross_entropy(out, y)
updates = tf.gradients(loss, b)
b.assign(b + learning_rate * updates)
```



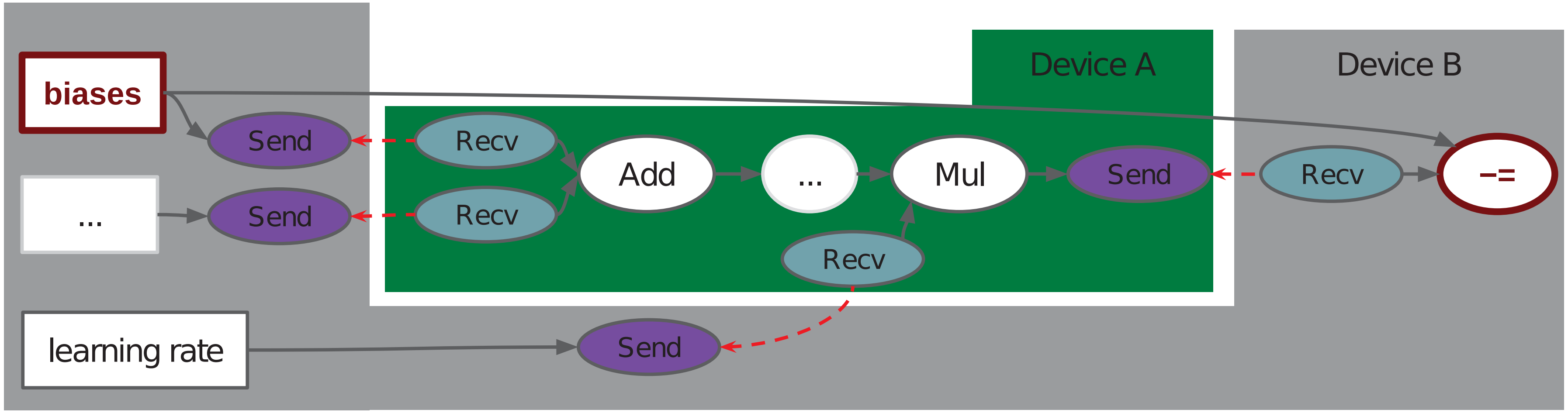
Graph based frameworks

In tensorflow the graph makes distributed processing easy



Devices: Processes, Machines, GPUs, etc

inject send and receive nodes

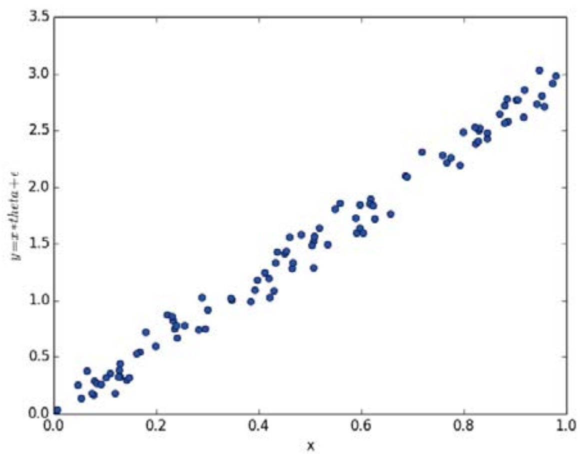


Devices: Processes, Machines, GPUs, etc

Tensorflow a simple example

Let us look at a simple example in tensorflow

```
# Define data size and batch size
n_samples = 1000
batch_size = 100

X_data, y_data =load_data()

# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))
```



Tensorflow a simple example

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
  W = tf.get_variable("weights", (1, 1),
                      initializer=tf.random_normal_initializer())
  b = tf.get_variable("bias", (1,),
                      initializer=tf.constant_initializer(0.0))
  y_pred = tf.matmul(X, W) + b
  loss = tf.reduce_sum((y - y_pred)**2/n_samples)
```

This creates a variable with state within the graph

$$L(W, b, \tilde{D}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b)))^2$$

Tensorflow a simple example

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())

  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

This calls tf.gradient()
for all variables in the graph
i.e. the magic happens here

Tensorflow a simple example

machine learning lab

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())

  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

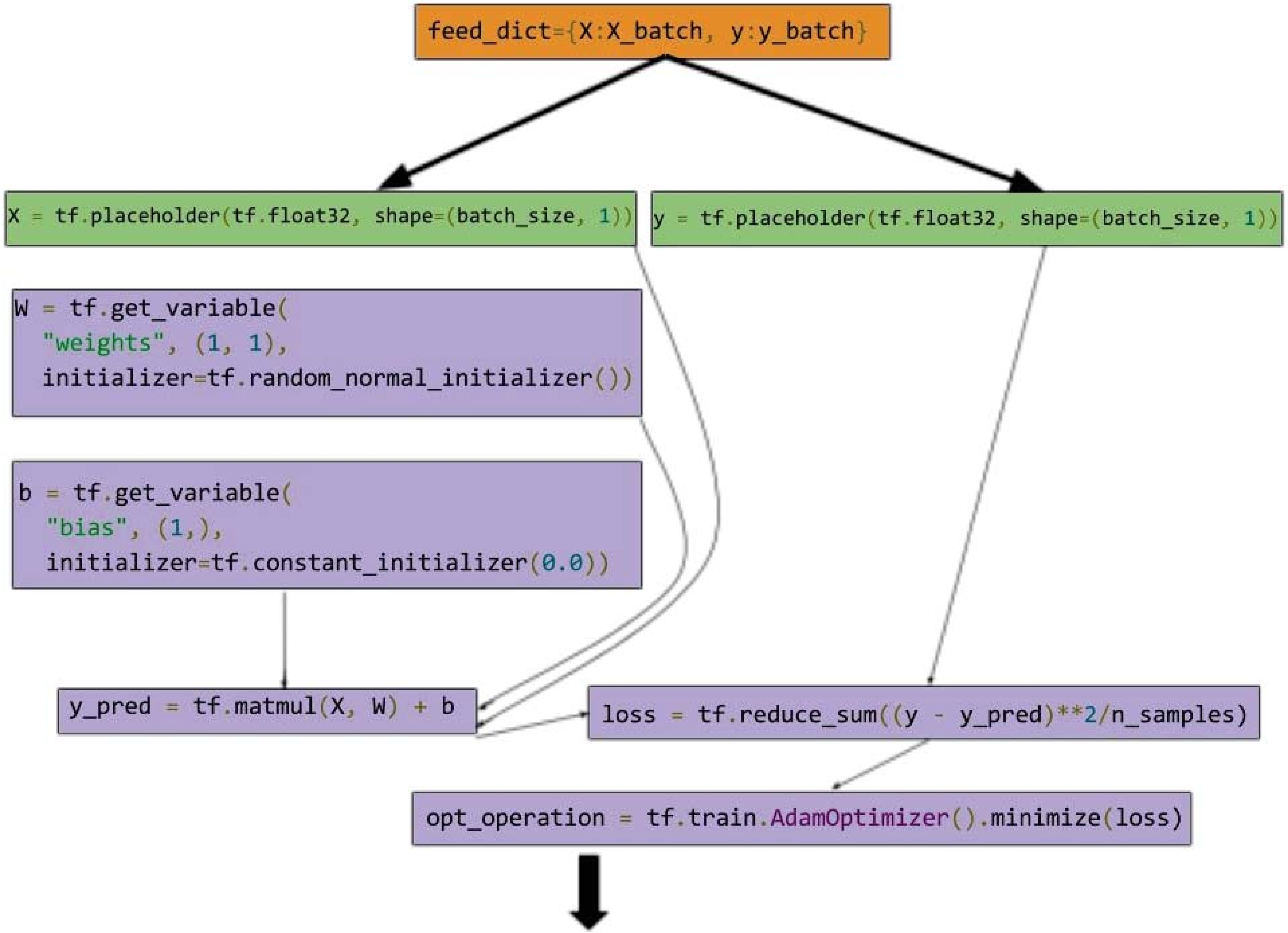
This creates gives us a session
to run the graph on a device

Tensorflow a simple example

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())

  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

This bridges python and the
graph execution, feeding variables
let us have another look

Tensorflow a simple example



Tensorflow a simple example

→ Simulation, for SGD $K = 1$, assuming that gradient evaluation on all data takes 4 times as much time as evaluating a single datapoint (gradient descent $(\gamma = 2)$, stochastic gradient descent $(\gamma_t = 0.01\frac{1}{t})$)

(Video sgd)

Stochastic Gradient descent (SGD)

- Whenever possible, solve your learning problem using SGD!!

Stochastic Gradient descent (SGD)

- Whenever possible, solve your learning problem using SGD!!

$\rightarrow$ remaining problem: We have to find a good **learning rate**

Tensorflow a simple neural net

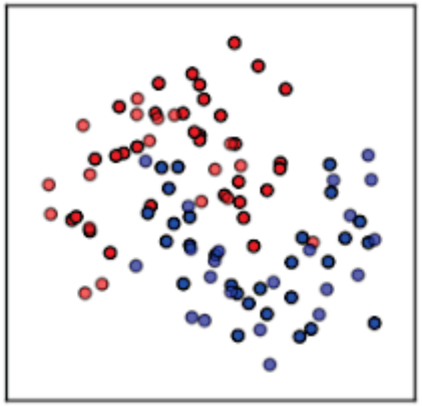
- **OK, cool** but what about a neural network ?

Tensorflow a simple neural net

- **OK, cool** but what about a neural network ?

→ just change the model

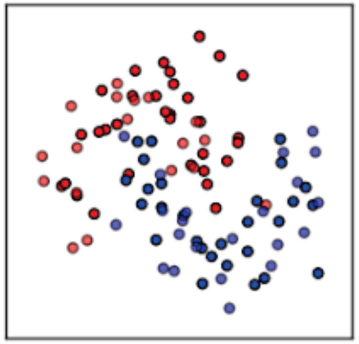
```
layer_size = 50
n_classes = 2
with tf.variable_scope('neural_network'):
    # Define variable nodes in the graph
    W1 = tf.get_variable("weights_1", (2, layer_size),
                         initializer=tf.random_normal_initializer())
    b1 = tf.get_variable("bias_1", (layer_size,),
                         initializer=tf.constant_initializer(0.0))
    W2 = tf.get_variable("weights_2", (layer_size, n_classes),
                         initializer=tf.random_normal_initializer())
    b2 = tf.get_variable("bias_2", (n_classes,),
                         initializer=tf.constant_initializer(0.0))

    # Compute network prediction
    hidden = tf.nn.relu(tf.matmul(X, W1) + b1)
    y_pred = tf.nn.softmax(tf.matmul(hidden, W2) + b2)
    # Define cross-entropy loss
    loss = tf.reduce_mean(tf.reduce_sum(-tf.log(y_pred) * y, 1))
```



train on
2D moons dataset

Tensorflow a simple neural net

train on
2D moons dataset



- And does it work ?
- let's look at the training accuracy and a visualization
- **typically** look at loss / accuracy over time

(Video)

- Visualize input weights (hidden size=2)
- Thanks to Alec Radford

Stochastic Gradient descent (SGD)

Tip ❸

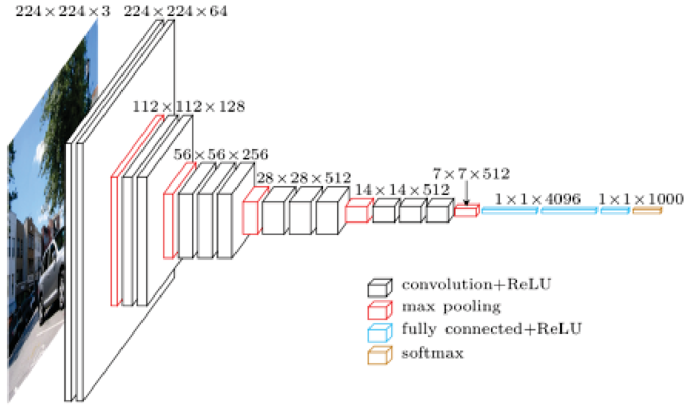
- How to choose a good learning rate / optimizer ?
- hand tuning learning rate for SGD ? **takes too much time**

$\rightarrow$ use Adam/RMSprop they typically **just work**

Tensorflow, more complicated networks ?

One option: TensorFlow slim

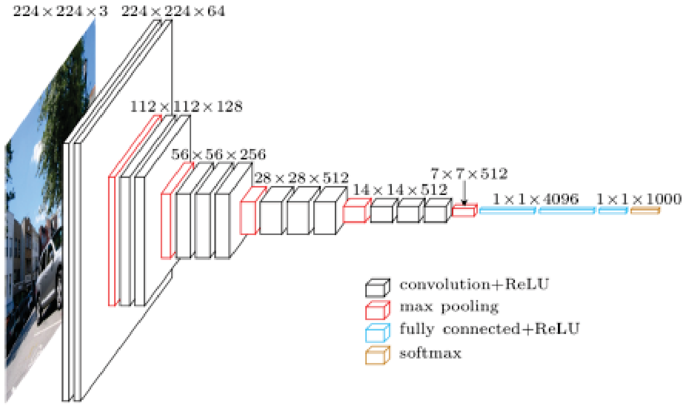
- Removes boilerplate
- Definition of a network from (Simonyan, 2015)
- Works with other code



Tensorflow, more complicated networks ?

One option: TensorFlow slim

- Removes boilerplate
- Definition of a network from (Simonyan, 2015)
- Works with other code



A good model for ImageNet

```
def vgg16(inputs):
  with slim.arg_scope([slim.ops.conv2d, slim.ops.fc], stddev=0.01, weight_decay=0.0005):
    net = slim.ops.repeat_op(2, inputs, slim.ops.conv2d, 64, [3, 3], scope='conv1')
    net = slim.ops.max_pool(net, [2, 2], scope='pool1')
    net = slim.ops.repeat_op(2, net, slim.ops.conv2d, 128, [3, 3], scope='conv2')
    net = slim.ops.max_pool(net, [2, 2], scope='pool2')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 256, [3, 3], scope='conv3')
    net = slim.ops.max_pool(net, [2, 2], scope='pool3')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 512, [3, 3], scope='conv4')
    net = slim.ops.max_pool(net, [2, 2], scope='pool4')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 512, [3, 3], scope='conv5')
    net = slim.ops.max_pool(net, [2, 2], scope='pool5')
    net = slim.ops.flatten(net, scope='flatten5')
    net = slim.ops.fc(net, 4096, scope='fc6')
    net = slim.ops.dropout(net, 0.5, scope='dropout6')
    net = slim.ops.fc(net, 4096, scope='fc7')
    net = slim.ops.dropout(net, 0.5, scope='dropout7')
    net = slim.ops.fc(net, 1000, activation=None, scope='fc8')
  return net
```

Keras

- Originally built on theano, also supports tensorflow now
- Scikit-learn style interface (fit and predict)
- Hides away theano/tensorflow internals
  (can still be combined with custom tensorflow code)

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))
```

```
model.compile(loss='categorical_crossentropy',
       optimizer='sgd', metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5,  batch_size=32)

loss_and_metrics = model.evaluate(
       X_test, Y_test, batch_size=32)

classes = model.predict_classes(X_test, batch_size=32)
proba = model.predict_proba(X_test, batch_size=32)
```

Frameworks to the rescue

Tip 4

- When developing use flexible tools for rapid prototyping, transition to production level code afterwards
- Do you need to run C++ in production ?

Frameworks to the rescue

Tip 4

- When developing use flexible tools for rapid prototyping, transition to production level code afterwards
- Do you need to run C++ in production ?
- Can you at least get training data in python/lua ?
  - If possible prototype in python
  - then deploy in C++ (tensorflow, Caffe, Caffe embedded)

Choosing a Framework

- Reasons why you might want to use one over the other

	Speed	Memory	Distributed	Languages	Deploy C++	Flexibility	Simplicity
Caffe	XXX	XXX	Somewhat	C++/Python	Easy	X	XX
Theano	XX		Somewhat	Python	Hard	XXXX	X
Lasagne	XX		No	Python	Hard	XXXX	XXX
Keras	XX		No	Python	Easy (TF)	XXX	XXXX
Torch	XXX		Yes	Lua	Embed lua	XXX	XXX
TensorFlow	XXX		Yes	C++/Python	Easy	XXXX	XXX

Thanks

Thank you for your attention!

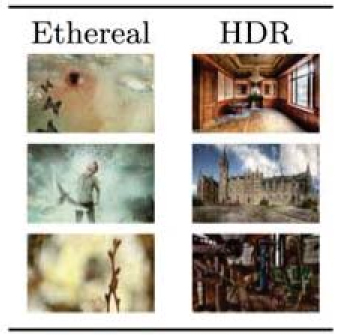
Model Zoo

- And how should I invent such an architecture ?

Lots of Data

image by Andrej Karpathy

Your Task



Style Recognition

© kaggle.com

Dogs vs. Cats
top 10 in
10 minutes

(Thanks to Evan Shelhammer for slides)

From ImageNet to Image Style

machine learning lab

Simply change a few lines in the model definition

```
layer {
  name: "data"
  type: "Data"
  data_param {
    source: "ilsvrc12_train_lmdb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layer {
  name: "fc8"
  type: "InnerProduct"
  inner_product_param {
    num_output: 1000
    ...
  }
}
```

```
layer {
  name: "data"
  type: "Data"
  data_param {
    source: "style_train_lmdb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layer {
  name: "fc8-style"
  type: "InnerProduct"
  inner_product_param {
    num_output: 20
    ...
  }
}
```

Input:
A different source

new name =
new params

Last Layer:
A different classifier

From ImageNet to Image Style

```
> caffe train -solver models/finetune_flickr_style/solver.prototxt
              -weights bvlc_reference_caffenet.caffemodel
```

Step-by-step in pycaffe:

```
pretrained_net = caffe.Net(
  "net.prototxt", "net.caffemodel")
solver = caffe.SGDSolver("solver.prototxt")
solver.net.copy_from(pretrained_net)
solver.solve()
```