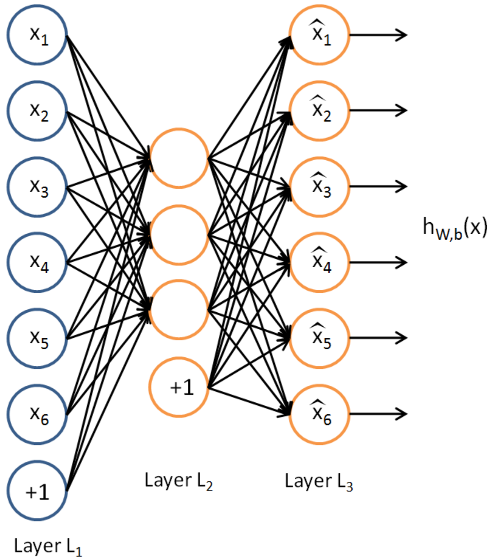
Deep Learning for Computer Vision – IV

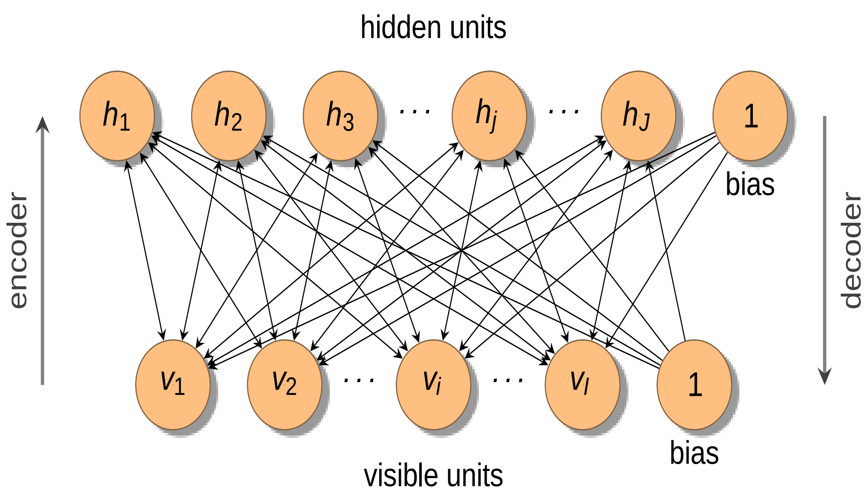
C. V. Jawahar

Popular DL Architectures

Auto Encoder



RBM



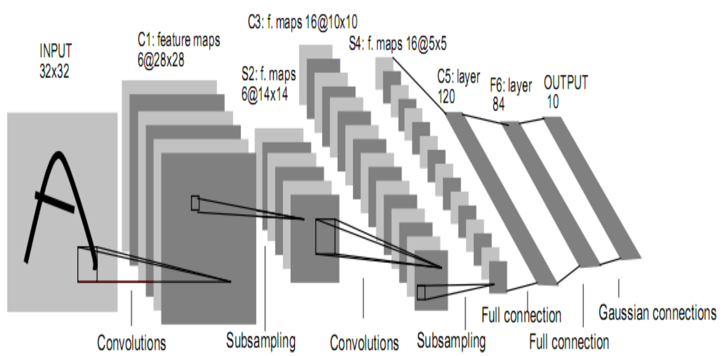
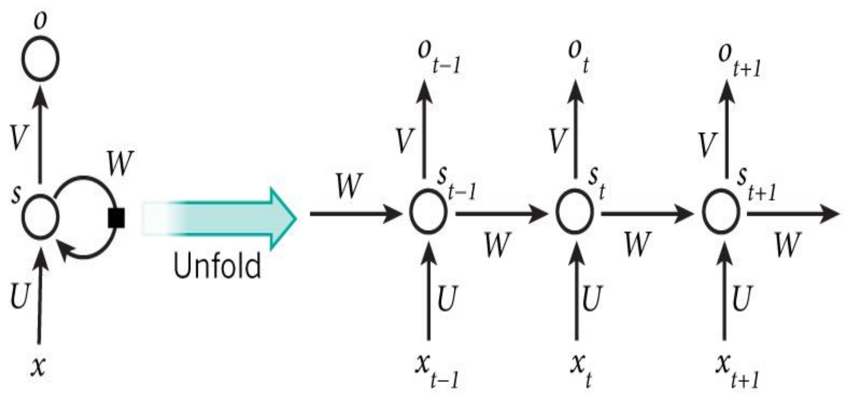


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

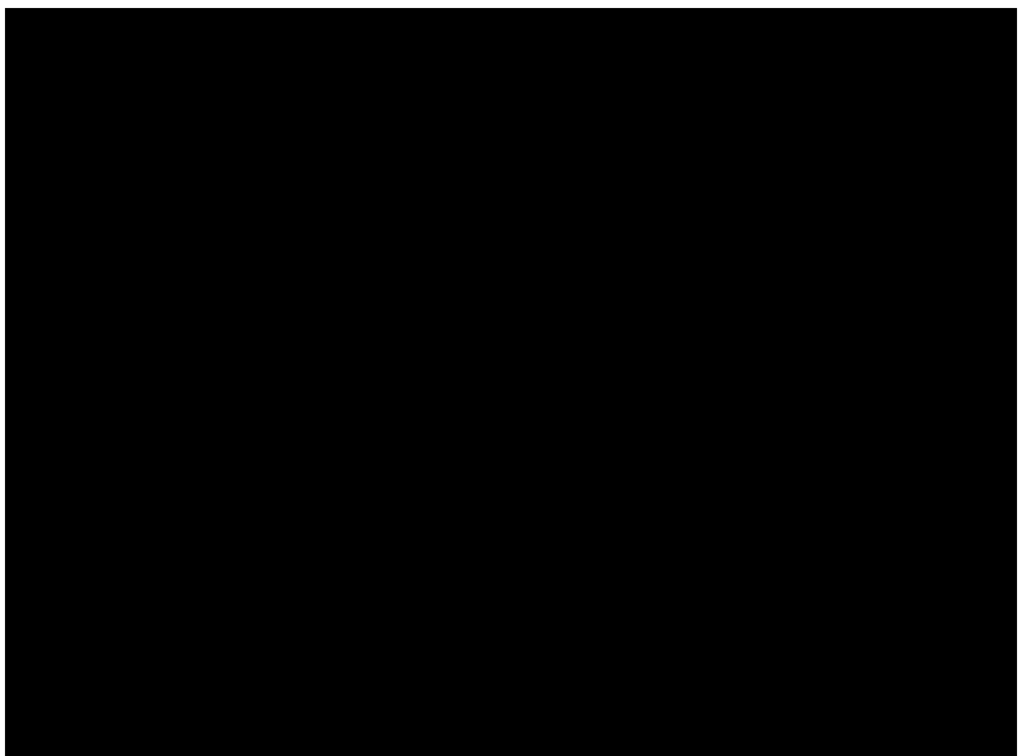
CNN



RNN

Deep Learning Saga

- Before you get to serious work, don’t forget to watch the comical tribute to Prof. Geoff Hinton (by Prof. Yoshua Bengio) :
- https://www.youtube.com/watch?v=mlXzufEk-2E



Must visit

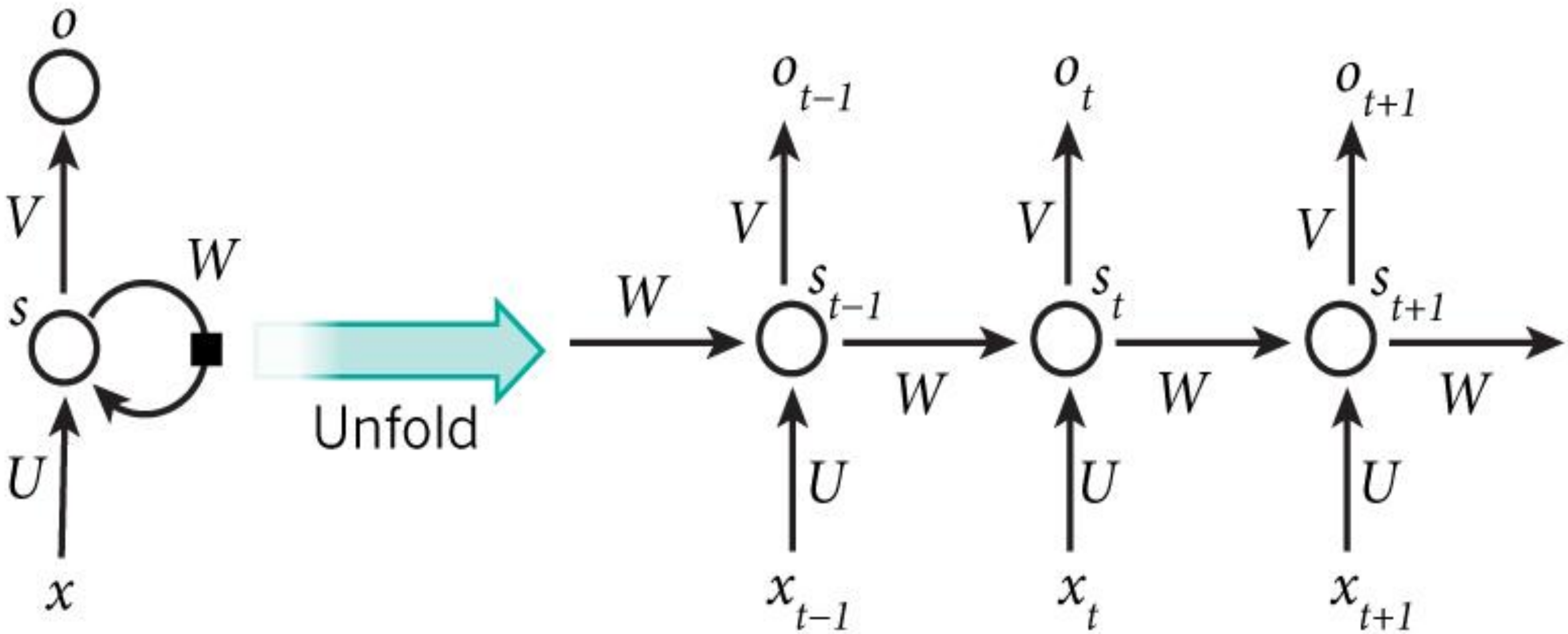
- http://deeplearning.net/
- has curated list of tutorials, reading materials, references, papers, research groups, datasets, startups etc

Basics: Take online courses with coding assignments

Professor	Course URL	Notes
Hugo Larochelle	Universite de Sherbrooke, IFT 725, Automne 2013	- Detailed explanation of theory - Exercises in python
Nando de Frietas	University of Oxford, Machine Learning, Jan 2015	- Fast paced but overview of recent developments - Maps concepts to Torch implementation
Fei Fei Li, Andrej Karpathy	Stanford University, CS231N, Jan-March 2015	- Explanations mapped to python code

Online courses for machine learning basics: Andrew Ng, Tom Mitchell, Nando de Freitas, Fred Hamprecht, Geoff Hinton, Yaser S. Abu-Mostafa, Patrick H Winston or NPTEL

Recurrent Neural Networks



Vanishing Gradients

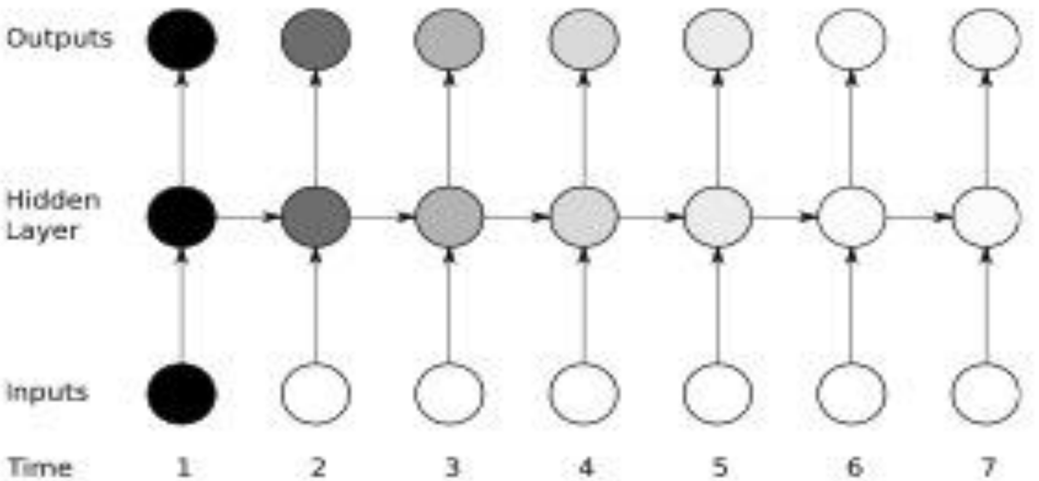


Figure 4.1: **The vanishing gradient problem for RNNs.** The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Vanishing Gradients

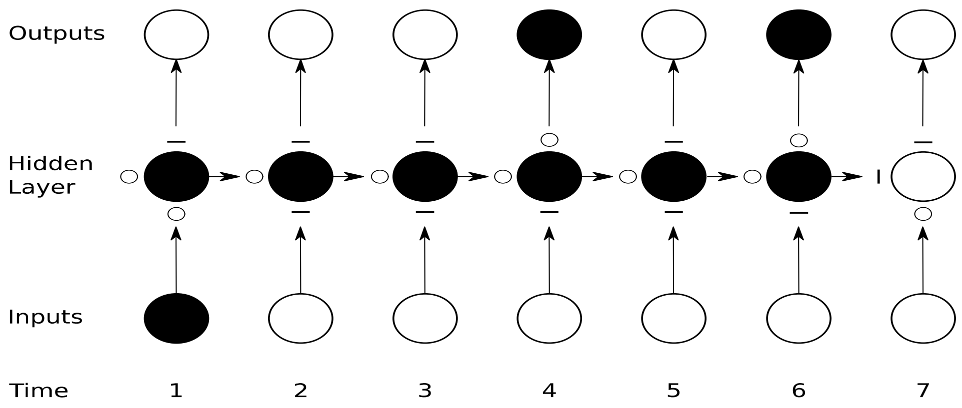
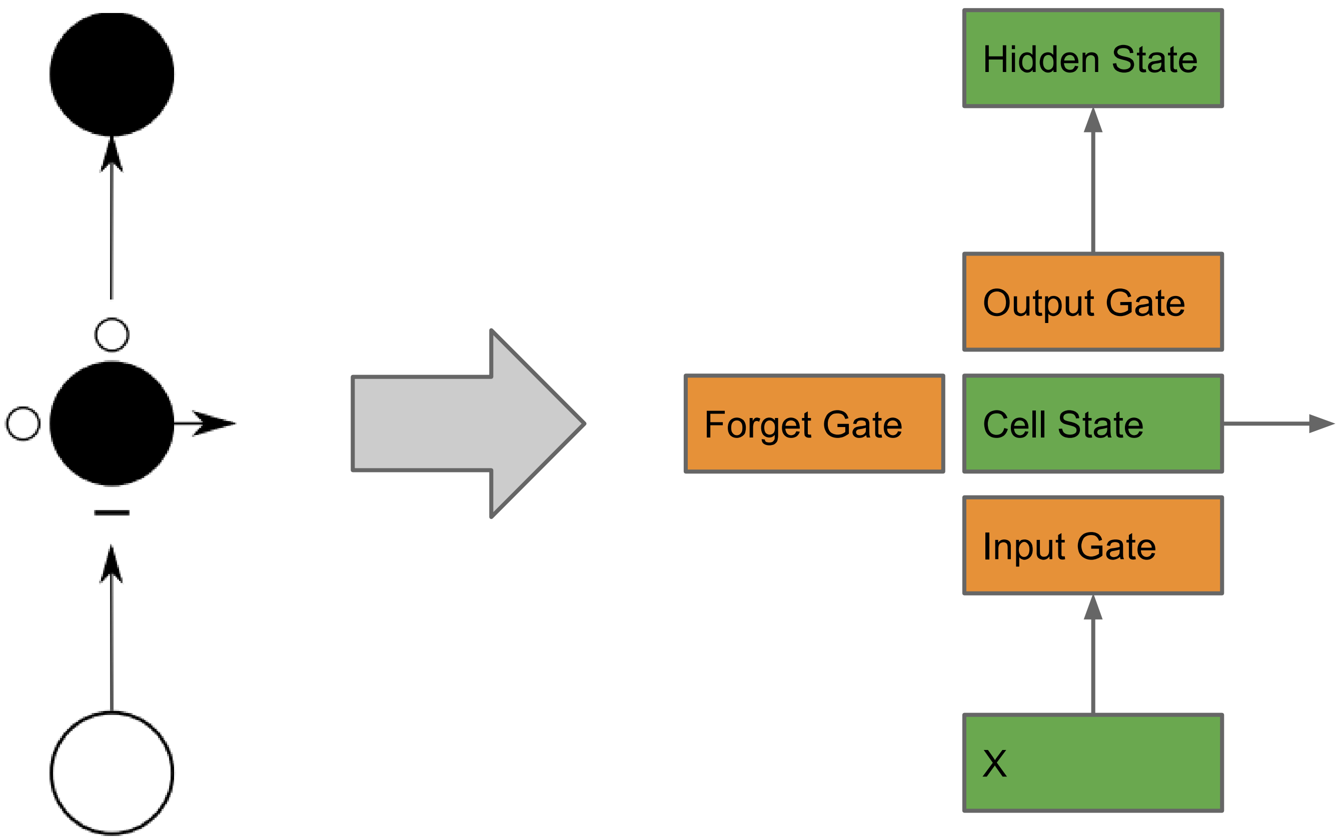
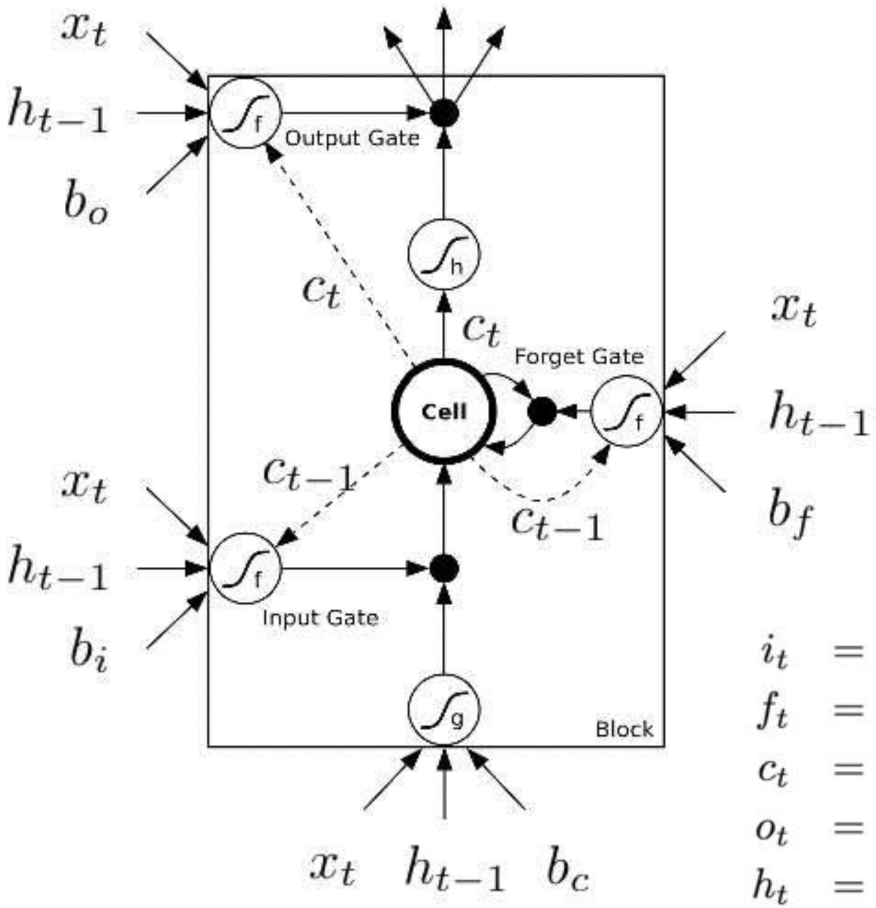


Figure 4.4: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

LSTM Node


Hidden State
Output Gate
Forget Gate
Cell State
Input Gate
X

LSTM Node



$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t tanh(c_t)
\end{aligned}
$$

LSTN Network

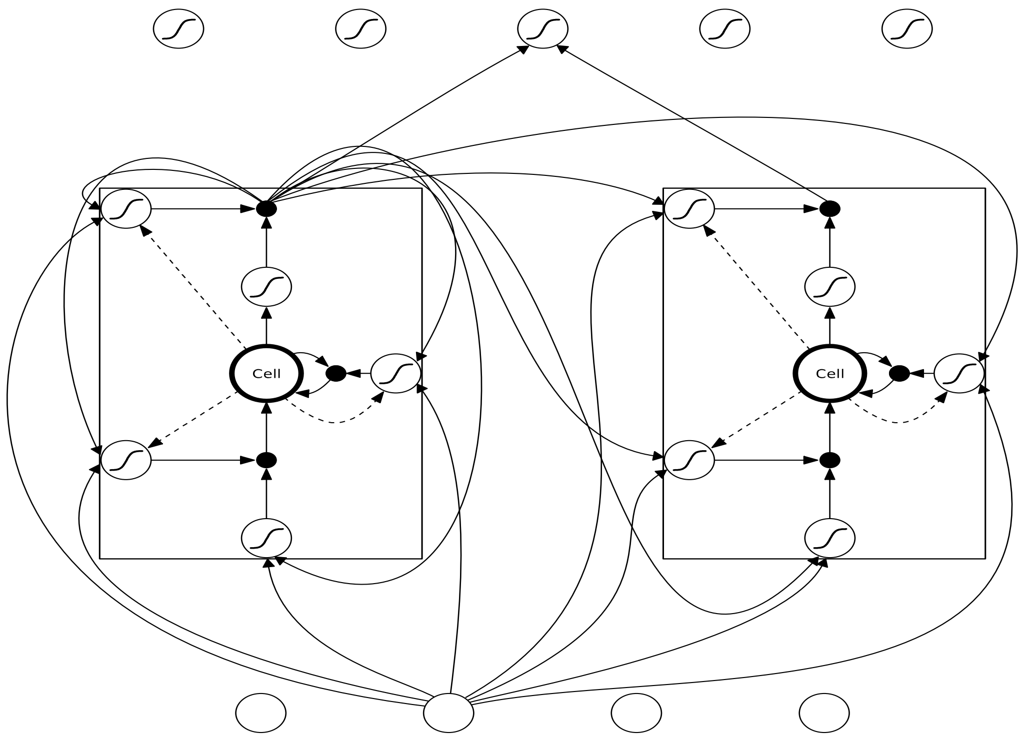
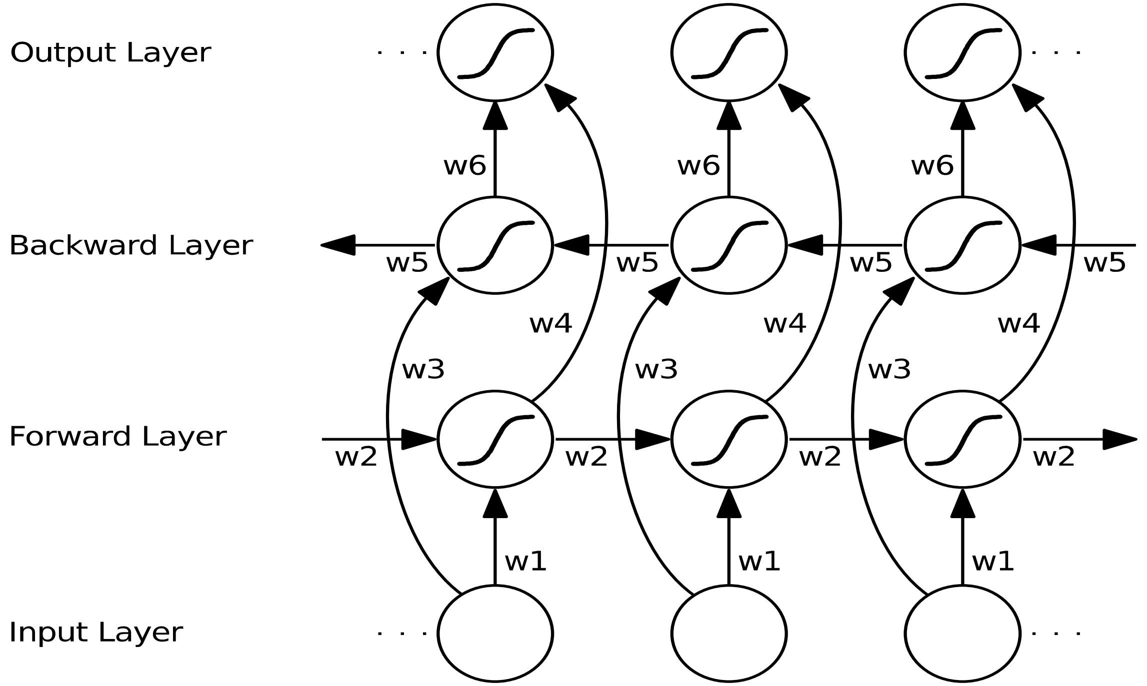


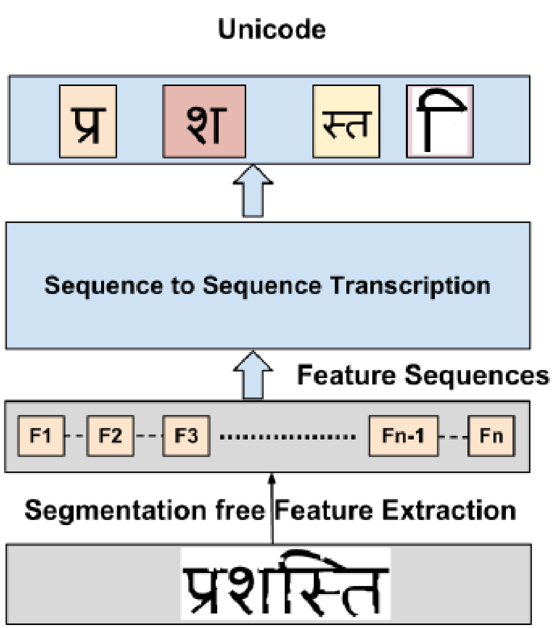
Figure 4.3: **An LSTM network.** The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output.

Bidirectional Network

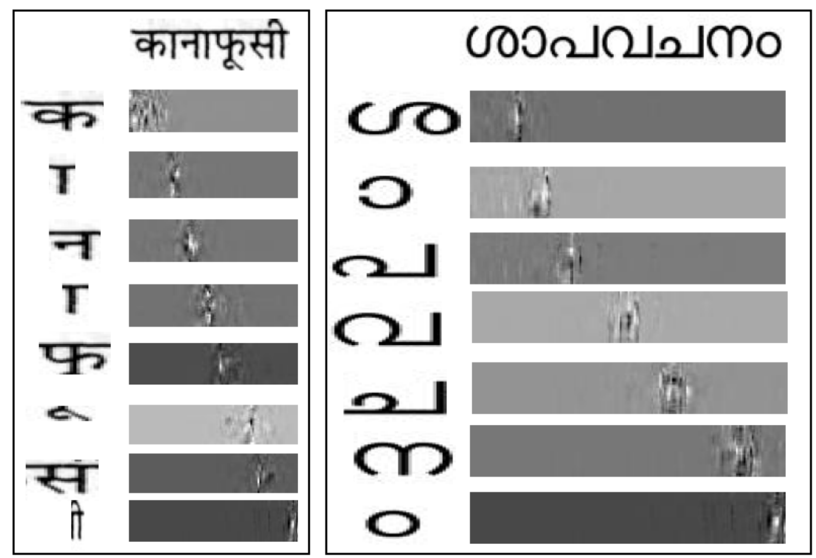


Deep OCR

- Formulated as a sequence-2-sequence transcription utilizing the context.
- Raw features.
- Segmentation free approach.
- Robust to common degradation and font styles.



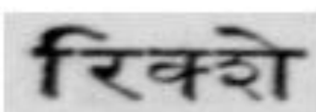
OCR architecture

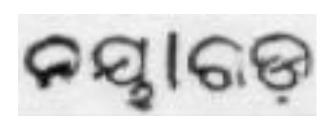


Visualizing use of context for recognizing each symbol

Deep OCR

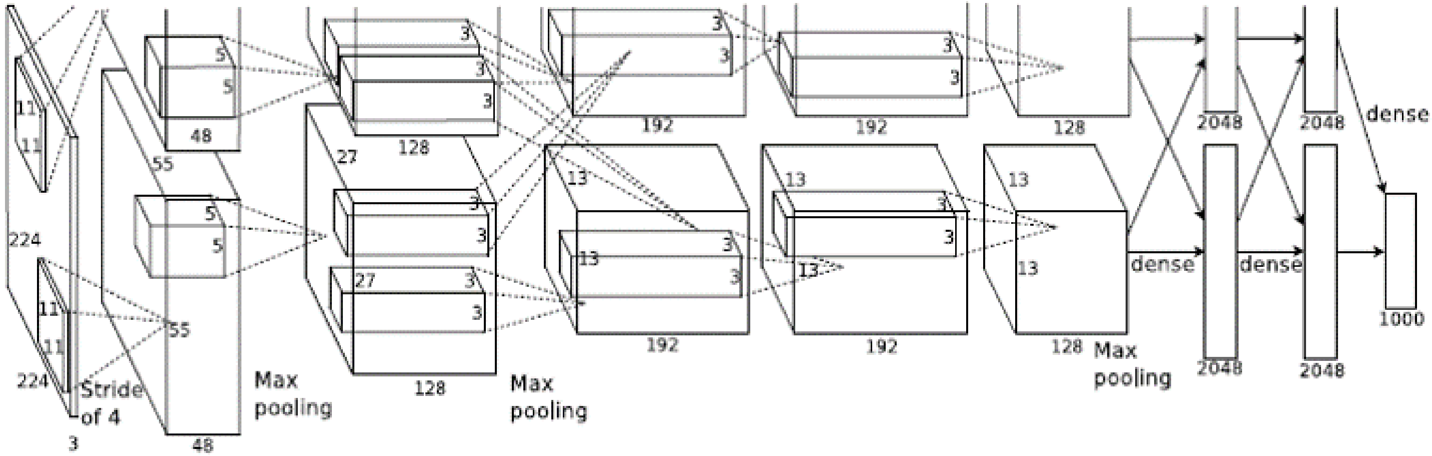
Language	# Pages Tested	Only Recognition		Segmentation + Recognition
		Word	Line	Line
Assamese	1000	1.78	1.65	2.10
Bengali	1300	2.13	2.22	2.30
Gujarati	3500	3.42	3.00	4.70
Gurmukhi	3500	1.28	1.22	2.30
Hindi	3000	2.30	2.00	3.90
Kannada	3500	4.10	4.16	5.60
Malayalam	3500	0.88	0.74	3.60
Manipuri	2000	1.30	1.21	2.30
Marathi	3500	1.29	1.10	3.80
Oriya	3500	3.49	2.40	3.30
Tamil	3500	2.44	4.00	5.60
Telugu	3500	2.00	1.90	4.86
English	300	0.93	0.65	1.25

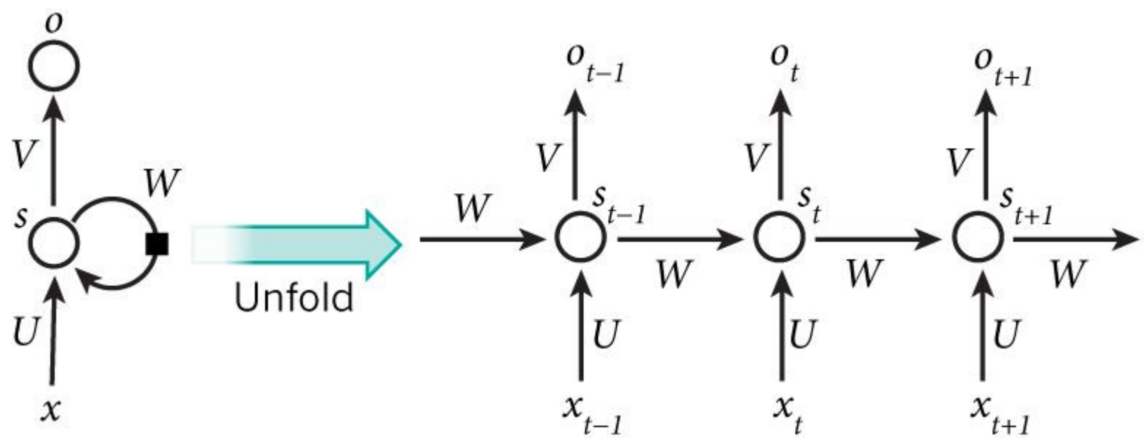
Hindi  Tamil  Malayalam 

Telugu  Kannada  Oriya 

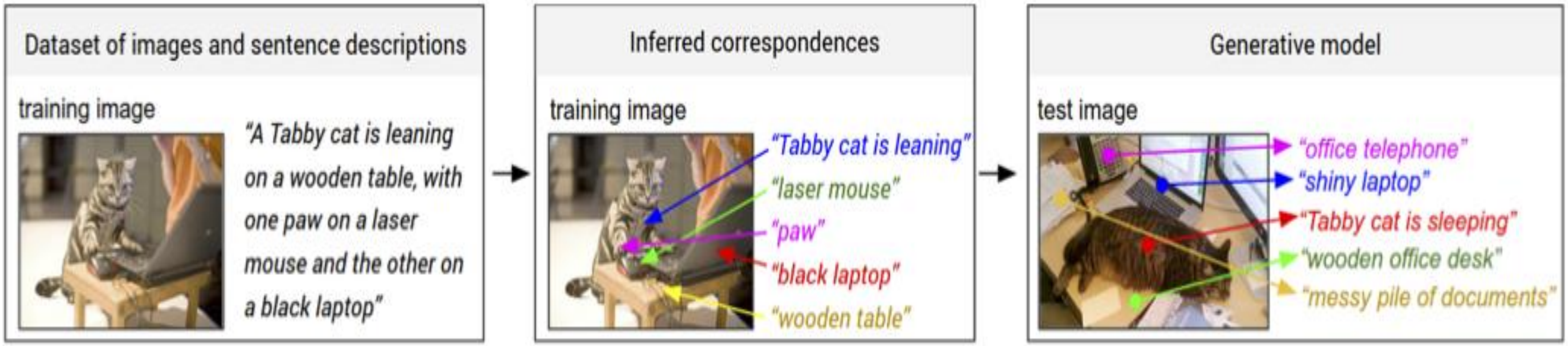
Sample images which were correctly recognized

Two powerful networks: CNNs and RNNs



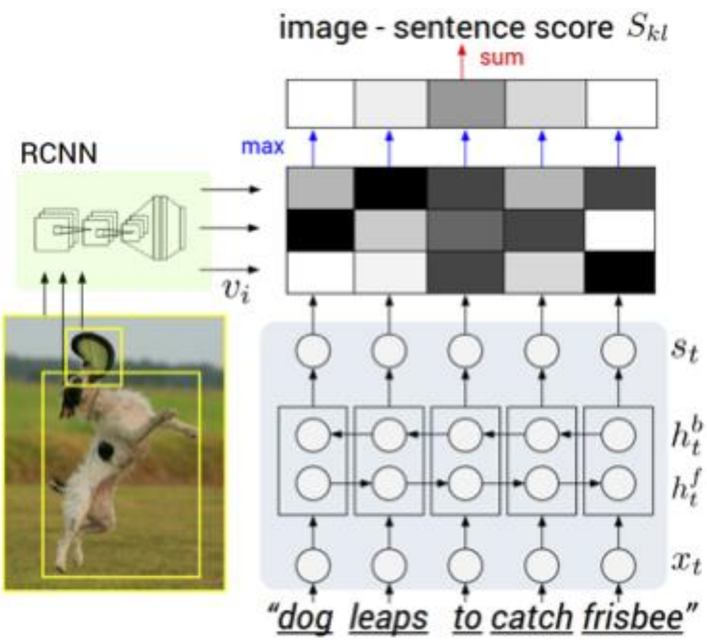


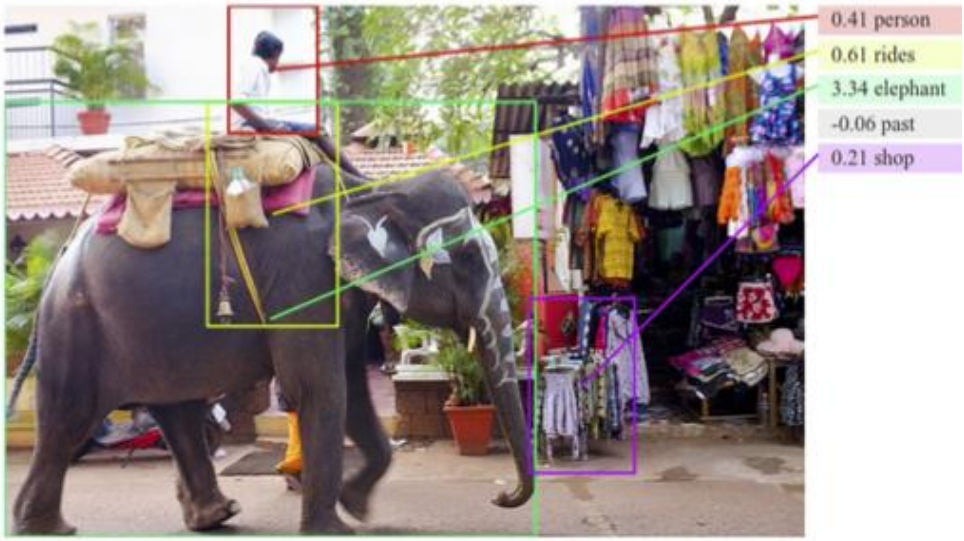
Captions with Deep Learning



A. Karpathy and L Fei-Fei, Deep visual semantic alignment for generating image descriptions, CVPR 2015

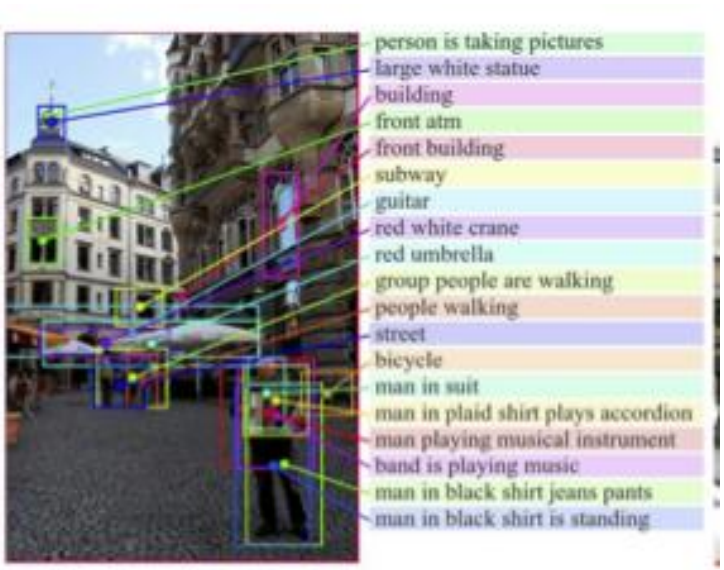
Captions with Deep Learning

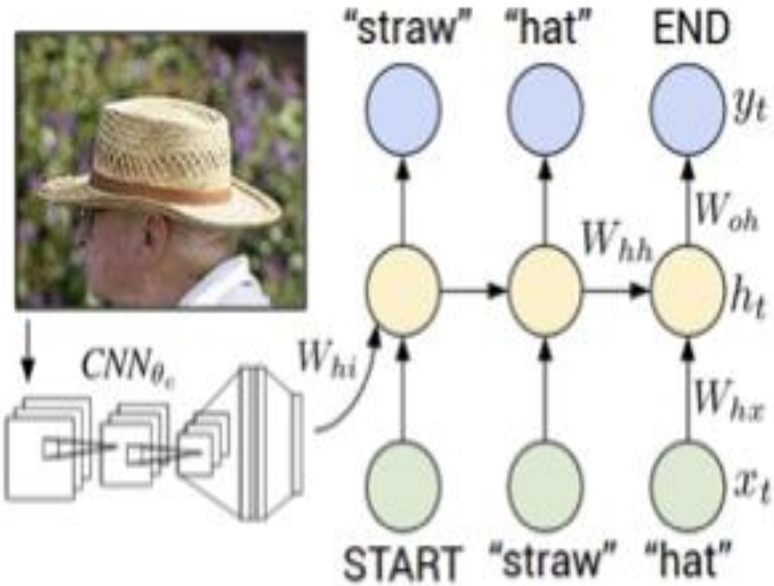




A. Karpathy and L Fei-Fei, Deep visual semantic alignment for generating image descriptions, CVPR 2015

Captions with Deep Learning



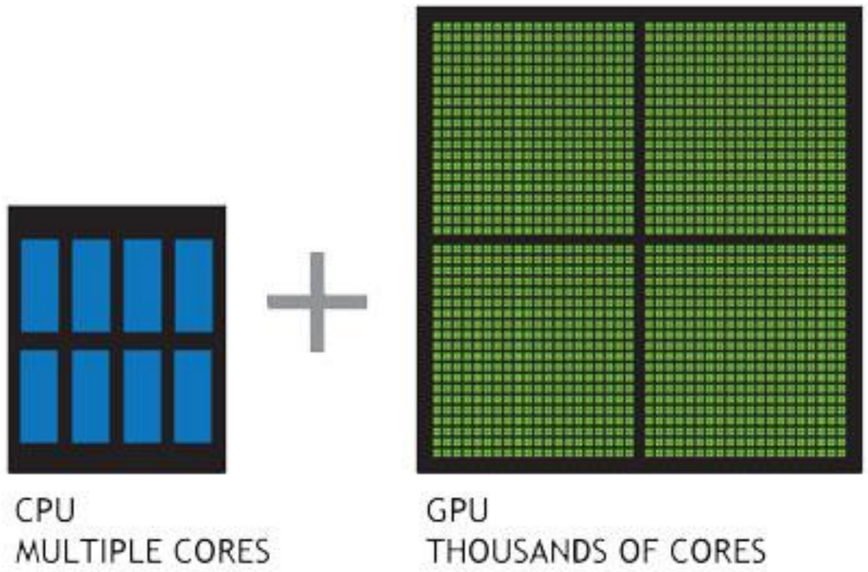


A. Karpathy and L Fei-Fei, Deep visual semantic alignment for generating image descriptions, CVPR 2015

Getting Started with Deep Learning

Hardware choice: CPU vs GPU

- CPU: few (less than hundred) cores optimized for sequential serial processing
- GPU: thousands of small, efficient cores for parallel processing

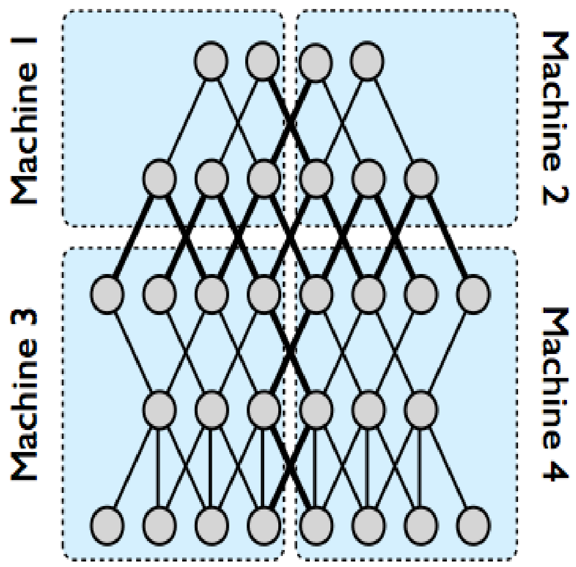


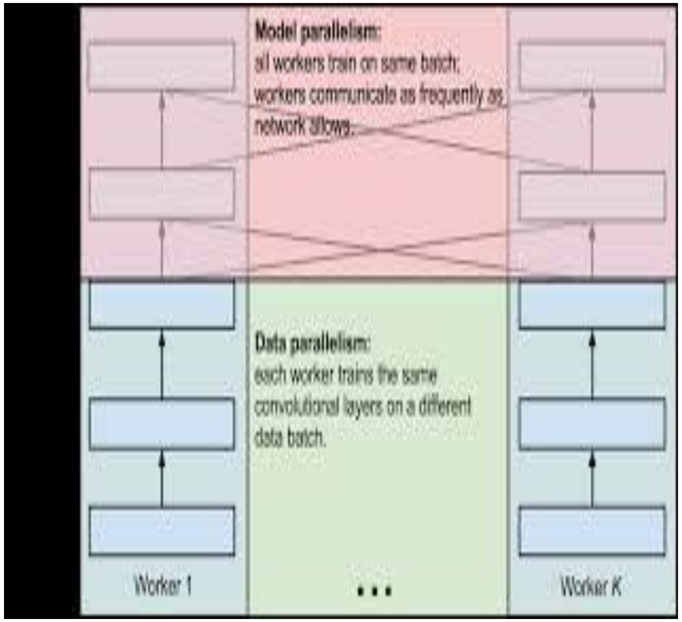
Funny explanation: https://www.youtube.com/watch?v=-P28LKWTzrI

(Image credit: NVIDIA)

DistBelief and Alex Khrizhevsky's weird trick

- http://research.google.com/pubs/pub40565.html
- http://arxiv.org/pdf/1404.5997v2.pdf





10000 CPU cores vs 2 graphics cards : naturally GPUs have become popular in deep learning

CPU vs GPU

- Training of ImageNet (1 million images) winning networks like Alexnet, VGGNet, GoogLeNet take several hours to weeks, using best available GPUs
- Training on CPUs is believed to be impractical for such datasets
- For accurate benchmark timings:
  - Soumith Chintala: https://github.com/soumith/convnet-benchmarks
  - https://github.com/BVLC/caffe/issues/1317

Imagenet 2016

- The only one submission from India was from Intel bangalore:
  - Using 32 nodes of the Xeon nodes we can train VGGA network in 30 hours and alexnet in 7.5 hours upto the best accuracy.

PCL Bangalore

- Description: We jointly train image classification and object localization on a single CNN using cross entropy loss and L2 regression loss respectively. The network predicts both the location of the object and a corresponding confidence score. We use a variant of the network topology (VGG-A) proposed by [1]. This network is initialized using the weights of classification only network. This network is used to identify bounding boxes for the objects, while a 144-crop classification is used to classify the image. The network has been trained on Intel Parallel Computing Lab's deep learning library (PCL-DNN) and all the experiments were performed on 32-node Xeon E5 clusters. A network of this size typically takes about 30 hrs for training on our deep learning framework. Multiple experiments for fine-tuning were performed in parallel on NERSC's Edison and Cori clusters, as well as Intel's Endeavor cluster.

Credit: Nataraj Jammalamadaka et al. , Intel

Machine learning : layered perspective

- **Machine Learning Libraries**
  - Torch
  - Caffe
  - Scikit-learn
- **Tensor**
  - Theano
  - TensorFlow
  - Numpy
- **Systems Programming 1**
  - CUDNN
  - CUBLAS
- **Systems Programming 2**
  - CUDA
- **OS**
  - NVIDIA Tesla drivers
  - NVIDIA GeForce drivers
  - Linux kernel
- **HW**
  - NVIDIA Tesla K40, K80
  - NVIDIA GeForce GTX 950, 960, 970, 980
  - CPU : Intel i7, AMD, ARM

Machine learning : layered perspective

Machine Learning Libraries
- Caffee

Systems Programming 1
- MKL

Systems Programming 2
- OPENMP
- MPI
- AVX Instructions

OS
- Linux kernel

HW
- CPU : Intel i7, Xeon, Xeon Phi

Intel’s Deep learning Library

Intel® Xeon® Processor Single Node Classification

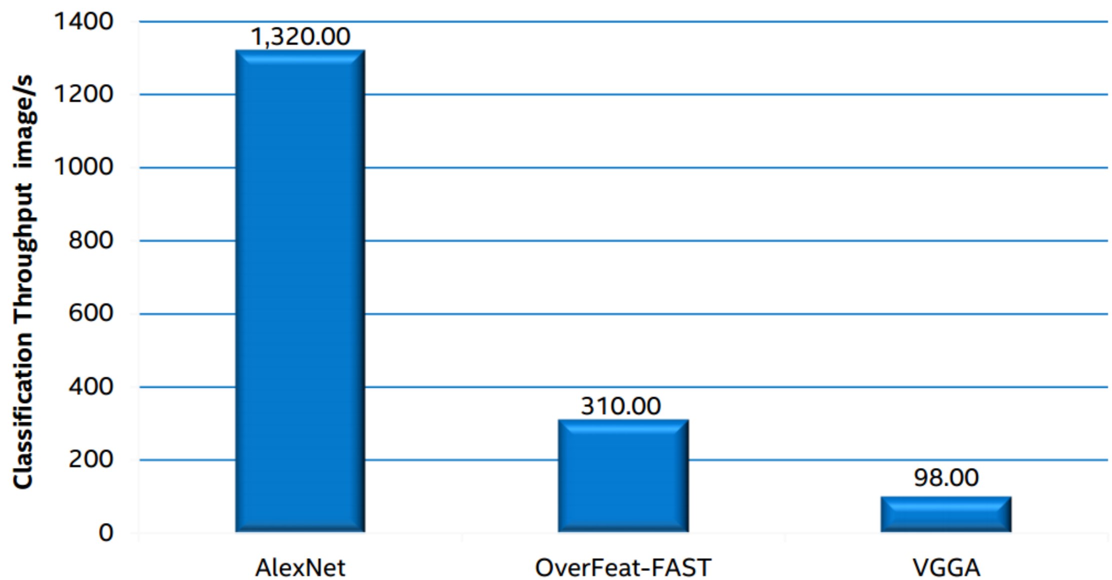
Intel® Xeon® processor performance for classification across a wide variety of image classification networks.

Experimental setup:
2 x Intel Xeon processor E5-2699v3 @ 2.30 GHz (HSW) Dual socket, 18 cores and 2 threads/socket; Cache size: 45MB
Memory: DDR4, 2133GHz, 64 GB, CentOS Linux* Release 7.0.1406

All performance reported for a batch of 32,000 images.

Benchmark networks are available at links in: https://github.com/soumith/convnet-benchmarks

Performance for Classification (img/s)



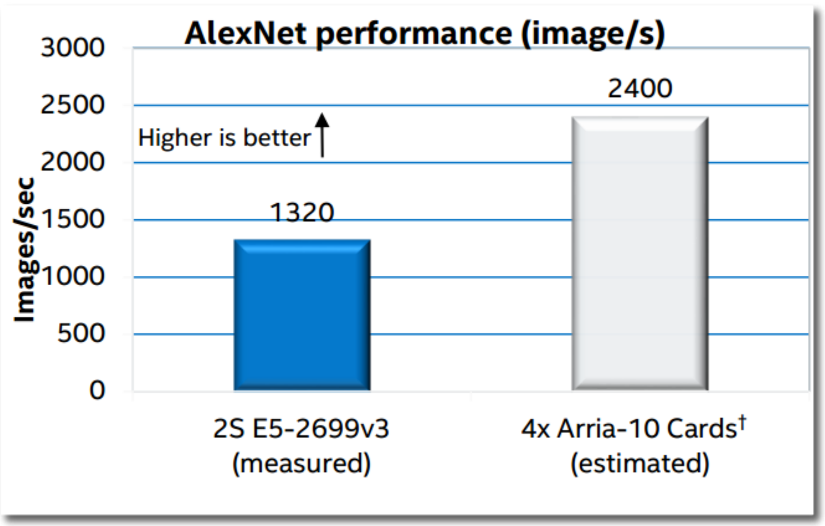
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

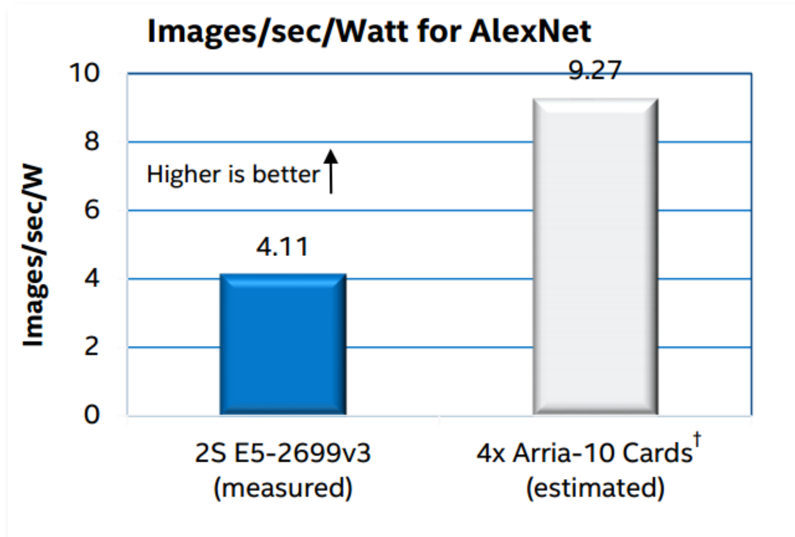
IDF15 INTEL DEVELOPER FORUM

Courtesy: Pradeep Dubey, Intel Labs

Intel's Deep learning Library

Single Node Classification with Intel® Xeon® Processor + FPGA





Power-performance of CNN classification boosted up to **2.2X**

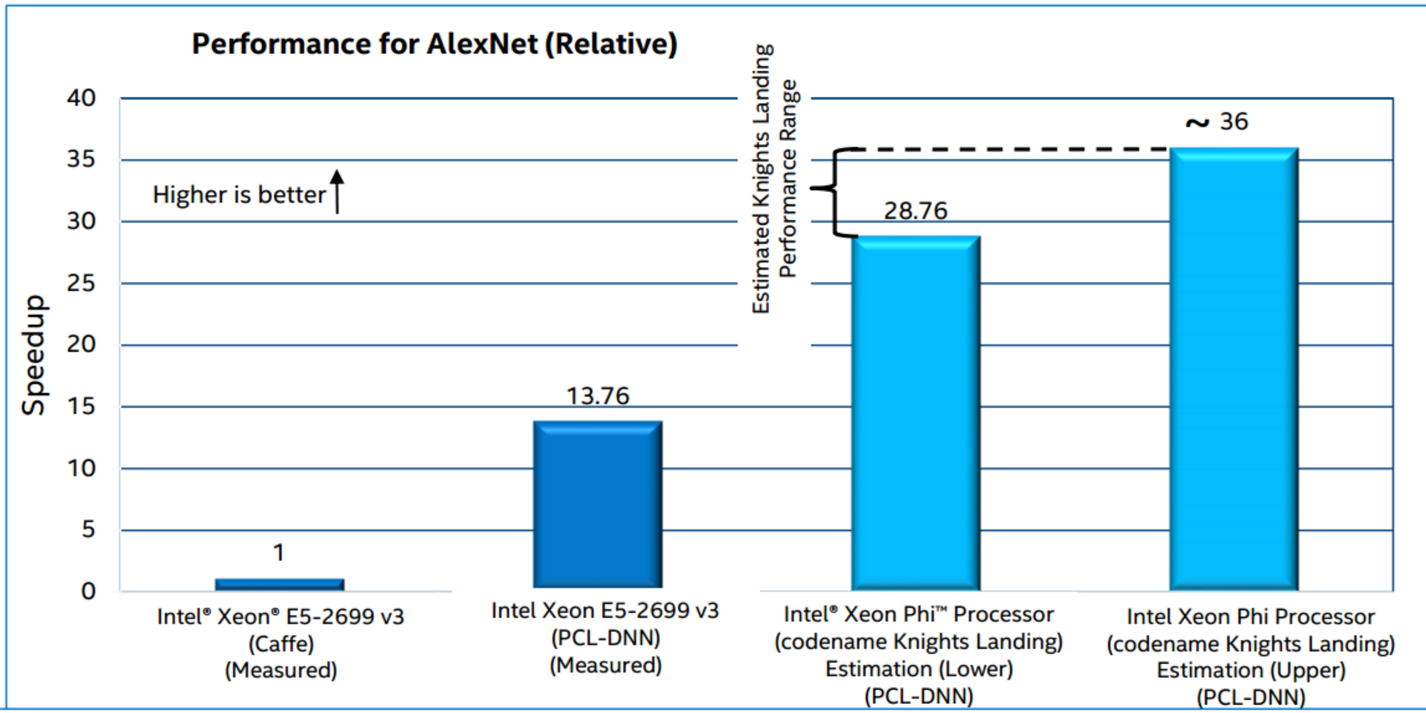
Source: Intel Measured (E5-2699v3 results); Altera* Estimated (4x Arria* 10 results)
†2S E5-2699v3 + 4x GX1150PCIe cards. Most computations executed on Arria-10 FPGA's, 2S E5-2699v3 host assumed to be near idle, doing misc. networking/housekeeping functions. Arria-10 results estimated by Altera with Altera custom classification network. 2x E5-2699v3 power estimated @ 139W while doing "housekeeping" for GX1150 cards based on Intel measured microbenchmark. In order to sustain ~2400 img/s we need a I/O bandwidth of ~500 MB/s, which can be supported by a 10GigE link and software stack

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configuration Details: See System Configurations slide For more information go to http://www.intel.com/performance Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Courtesy: Pradeep Dubey, Intel Labs

Intel's Deep learning Library

Intel Performance: Single Node Training



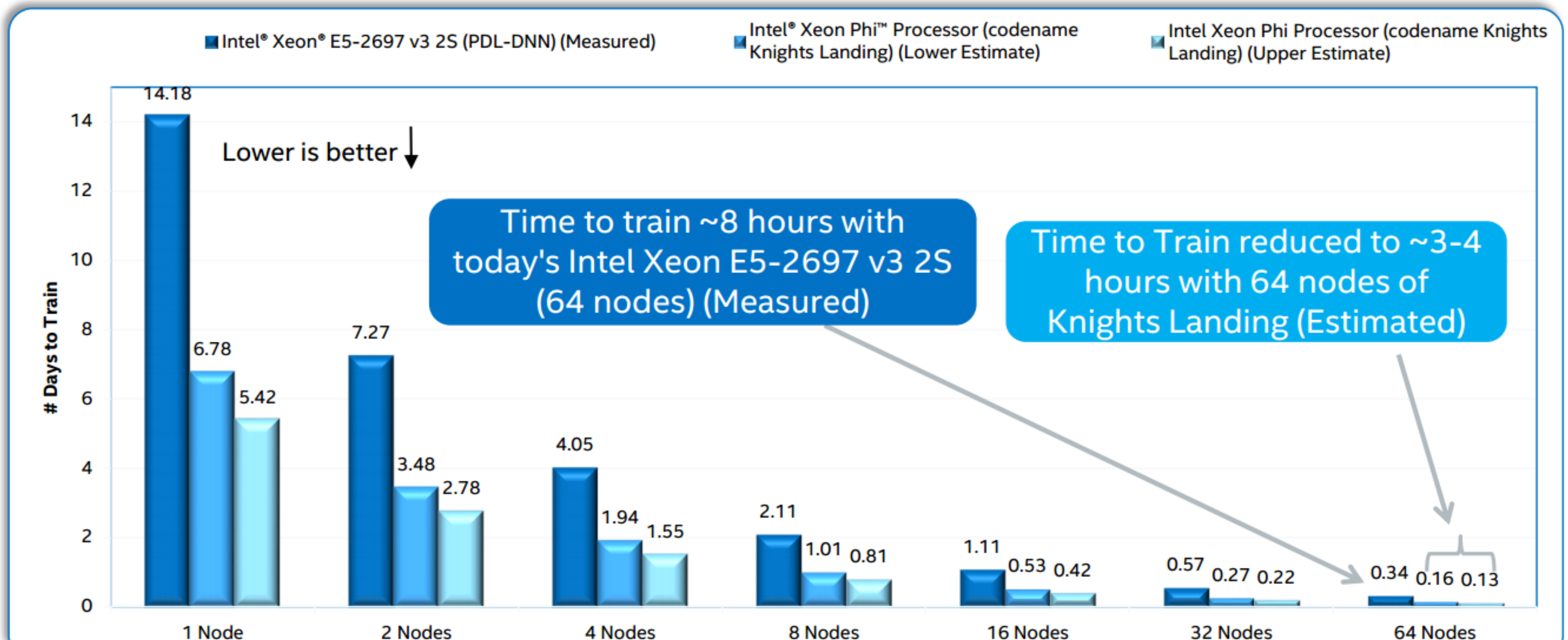
Source: Results were measured by Intel in Q1 2015 for Intel Xeon E5-2697 v3 or results were estimated by Intel for Intel® Xeon Phi™ processor codename Knights Landing.

Intel Xeon processor E5-2699v3 2S measured: 8 x 8GB DDR4 2133, AlexNet on randomly generated inputs (32,000 images) Intel® C Compiler: 15.0.2, OS: CentOS 7.0.1406

Courtesy: Pradeep Dubey, Intel Labs

Intel's Deep learning Library

Time to Train (OverFeat-FAST Network)



Source: Results were measured by Intel in Q1 2015 for Intel Xeon E5-2697 v3 or results were estimated by Intel for Intel Xeon Phi Processor codename Knights Landing.

2 x Intel Xeon processor E5-2697 v3 @ 2.60GHz, DDR4, 2133GHz, 64 GB; RHEL 6.5, Network interface: InfiniBand* FDR, Intel® C Compiler 15.0.2 with Intel® Advanced Vector Extensions 2 (Intel® AVX2), OpenMP*, Intel® MPI library, DNN Library: PCL-DNN Library, PCL-DNN Harness & PCL-CML Library, randomly generated inputs (64000 images), training on 1.3M images of ImageNet-1k

Courtesy: Pradeep Dubey, Intel Labs

Deep Learning libraries

- AlexNet won ImageNet in 2012
- Atleast 67 libraries in 3 years !
- Comprehensive list:
- https://docs.google.com/spreadsheets/d/1XvGfi3TxWm7kuQ0DUqYrO6cxva196UJDxKTxccFqb9U/htmlview?pli=1&sle=true

Selecting a NVIDIA GPU

Card	Memory (GB)	Cores	Price (USD)	Power Requirements(W)	Type
GeForce GTX 950	2	768	159	350	Desktop
GeForce GTX 960	2	1024	199	400	Desktop
GeForce GTX 970	4	1664	329	500	Desktop
GeForce GTX 980	4	2048	549	500	Desktop
GeForce GTX 980 Ti	6	2816	649	600	Desktop
GeForce GTX Titan X	12	3072	999	600	Desktop
Tesla K40	12	2880	2999	NA	Server
Tesla K80	24	4992	4199	NA	Server
Tesla M40	12	3072	NA	250	Server
Tesla M60	16	4096	NA	NA	Server

What hardware to buy ?

Building GPU System

Component	Requirement	List Price (MSRP) range in USD (lowest - highest)	
		Desktop class	Server class
GPU	- Most libraries only support CUDA (and not OpenCL). - CUDA is NVIDIA technology - This means you can use only NVIDIA GPUs	Geforce GTX series: 150 - 1000	Tesla series K40-K80: 3000 - 4200
CPU	- NVIDIA Maxwell GPUs require atleast fifth generation Intel i7 or Xeon processors	i7 Haswell / Skylake family: 303 - 623	Xeon E5-E7 family: 213 - 7174
RAM	- Larger RAM is beneficial as most ML algorithms use in memory data structures like vectors, matrices, tensors	Crucial 64GB DDR4: 550 - 1080	Crucial 128 GB DDR4: 1140 - 1424
Hard Disk	- Large datasets like ImageNet necessitates higher storage capacities	Seagate 7200 RPM 1-3 TB 70 - 135	Seagate 7200 RPM 3-6 TB: 135 - 300
Assembled System rough price estimate		**1-1.25 lakh INR**	**3-5 lakh INR**

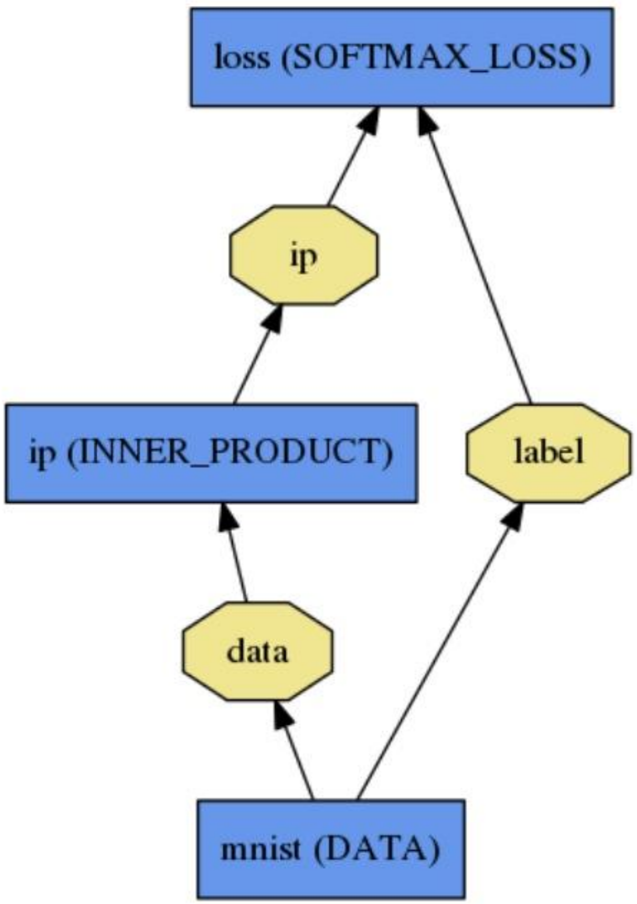
Caffe

- https://github.com/BVLC/caffe/
- C++, python, matlab, command line executables
- Started with Prof. Trevor Darell's BVLC grouf at Berkeley
- Key developers: Yanqing Jia, Jeff Donahue, Evan Shelhmer, Jonathan Long et al
- **To get started:**
  - https://github.com/BVLC/caffe/tree/master/examples
  - http://caffe.berkeleyvision.org/
  - http://tutorial.caffe.berkeleyvision.org/

Caffe concepts

- Blob:
  - Abstraction for N-dimensional array holding data
- Layer:
  - Forward function: function
  - Backward function: gradient
  - Predefined layers like InnerProduct, Convolution ☺
  - Create custom layers by inheriting caffe.Layer class ☺
  - Have to write CPU and GPU versions explicity ☹
- Net:
  - DAG of layers and loss function

Caffe: network: define a DAG



- Network specified in a JSON like prototxt format
- Model parameters serialized to Google Protobuf format : allows checkpointing and using pretrained model
- Learning rate, momentum are specified in 'solver.txt' file
- Also, supports databases like LMDB and LevelDB

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source:
"input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param
{
    num_output: 2
  }
}
layer {
  name: "loss"
  type:
"SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Source: http://caffe.berkeleyvision.org/tutorial/net_layer_blob.html

Caffe: training and testing

- caffe train –solver solver.prototxt –gpu 0,1
- caffe test –model model.prototxt –weights model.caffemodel -gpu 0 -iterations 100
- For Python interface:
  - Check ipython notebooks in the examples folder

Theano

- Theano is a tensor library: define, optimize, evaluate mathematical operations involving multi dimensional arrays
- Started with Prof. Yoshua Bengio's LISA lab at Universite de Montreal
- Key developers: Fedric Bastien, Pascal Lamblin, A Berger, Ian GoodFellow, Razvan Pascanu, James Bergstra et al
- **To get started:**
  - http://deeplearning.net/software/theano/
  - http://www.deeplearning.net/tutorial/

Theano concepts

- Symbolic programming ☹
  - Similar to mathematica, sympy, maple
- Automatic differentiation ☺
  - Symbolic expressions converted into graphs and differentiation is done symbolically
- Transparent use of GPU ☺
  - Symbolic expression generates GPU CUDA code
- Python language / numpy compatibility ☺

Theano: network: write expressions

[Source J Bergstra: http://bit.ly/1Qn8Qnm]

```
# Initial imports
import numpy as np
import theano.tensor as T
from theano import shared, function
rng = np.random.RandomState(123)
```

```
# Create a sample logistic regression problem.
true_w = rng.randn(100)
true_b = rng.randn()
xdata = rng.randn(50, 100)
ydata = (np.dot(xdata, true_w) + true_b) > 0.0
```

Input data and labels

```
# Step 1. Declare Theano variables
x = T.dmatrix()
y = T.dvector()
w = shared(rng.randn(100))
b = shared(numpy.zeros(()))
print "Initial model"
print w.get_value()
print b.get_value()
```

- Symbolic variables
- "Shared" variables for state persistence

```
# Step 2. Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))
xent = -y * T.log(p_1) - (1 - y) * T.log(1 - p_1)
prediction = p_1 > 0.5
cost = xent.mean() + 0.01 * (w ** 2).sum()
gw, gb = T.grad(cost, [w, b])
```

- Expressions of symbolic vars
- Automatic differentiation
- Can define complex NN models

```
# Step 3. Compile expressions to functions
train = function(inputs=[x, y],
                 outputs=[prediction, xent],
                 updates={w:w - 0.1 * gw,
                          b:b - 0.1 * gb})
```

SGD update rule

```
# Step 4. Perform computation
for loop in range(100):
    pval, xval = train(xdata, ydata)
    print xval.mean()
```

Run over multiple epochs

Theano based libraries

VGG_CNN_S model definition in Lasagne

```
net = {}
net['input'] = InputLayer((None, 3, 224, 224))
net['conv1'] = ConvLayer(net['input'], num_filters=96, filter_size=7, stride=2)
net['norm1'] = NormLayer(net['conv1'], alpha=0.0001) # caffe has alpha = alpha * pool_size
net['pool1'] = PoolLayer(net['norm1'], pool_size=3, stride=3, ignore_border=False)
net['conv2'] = ConvLayer(net['pool1'], num_filters=256, filter_size=5)
net['pool2'] = PoolLayer(net['conv2'], pool_size=2, stride=2, ignore_border=False)
net['conv3'] = ConvLayer(net['pool2'], num_filters=512, filter_size=3, pad=1)
net['conv4'] = ConvLayer(net['conv3'], num_filters=512, filter_size=3, pad=1)
net['conv5'] = ConvLayer(net['conv4'], num_filters=512, filter_size=3, pad=1)
net['pool5'] = PoolLayer(net['conv5'], pool_size=3, stride=3, ignore_border=False)
net['fc6'] = DenseLayer(net['pool5'], num_units=4096)
net['drop6'] = DropoutLayer(net['fc6'], p=0.5)
net['fc7'] = DenseLayer(net['drop6'], num_units=4096)
net['drop7'] = DropoutLayer(net['fc7'], p=0.5)
net['fc8'] = DenseLayer(net['drop7'], num_units=1000, nonlinearity=lasagne.nonlinearities.softmax)
output_layer = net['fc8']
```

- Libraries with higher level of abstraction can make network construction simpler
- Examples: PyLearn2, Lasagne, Blocks, Keras, nolearn,
- Tradeoff between mathematical expressiveness and easy-of-use

[Source: Lasagne github : http://bit.ly/1Rj5zpn]

Torch

- http://torch.ch/
- Torch is a machine learning library which has good support for neural networks and CUDA (GPU)
- Language : LuaJIT
- Lua is lightweight scripting language which can be embedded in C programs and is popular in gaming community
- Key developers: Ronan Collobert, Soumith Chintala, Koray Kavukcuoglu, Clement Farabet et al
- Supported by: NYU, FAIR, Purdue e-Lab

- **To get started:**
  - http://torch.ch/
  - https://github.com/torch/torch7/wiki/Cheatsheet
  - http://tylerneylon.com/a/learn-lua/
  - Prof. Nando de Frietas, Oxford ML course, 2015
    - https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/
  - Code: https://github.com/torch
  - Prof. Eugenio Culurciello: Artificial and Robotic vision course, Purdue, 2013
    - https://www.youtube.com/playlist?list=PLNgy4gid0G9e0-SiJWEdNEcHUYjo1T5XL

Torch concepts

- Base class nn.Module represents NN layers
  - forward() : function
  - backward() : gradient
- Base class: nn.Criterion
  - Similar but represents loss functions
- Modules can be composed into a DAG using contianers like nn.Sequential(), nn.Parallel(), nngraph to create complex networks
- Backpropagation and optimization algorithms provided
- Custom layers and criterions can be built easily ☺
- CUDA support for tensor and NN operations ☺

Torch: network: build modularly

```
x = torch.rand(128,200)*10;
y = torch.rand(128);
print('Successfully created Tensors')

torch.manualSeed(101);
model = nn.Sequential();
model:add(nn.Linear(200,100))
model:add(nn.Sigmoid())
model:add(nn.Linear(100,1))
crit = nn.MSECriterion();
w,df_dw = model:getParameters();
print('Successfully created model')


function f(w)
        pred = model:forward(x);
        fw = crit:forward(pred,y)
        grad = crit:backward(pred,y);
        model:zeroGradParameters();
        model:backward(x,grad);
        return fw, df_dw
end

print("Starting gradient descent from 'optim' on CPU...")
maxIter = 100;
timer = torch.Timer();
for i=1,maxIter do
        _,fw = optim.sgd(f,w);
        if i%(torch.floor(maxIter/10))==0 then print(string.format('MSE = %f',fw[1])) end
end
print(string.format('Success! Average iteration time was %f',timer:time().real/maxIter))
```

Input data, labels

1. Create Sequential NN module
2. Add layers to the module
3. Define criterion (loss) function

model -> forward
criterion -> forward
criterion -> backward
model -> backward

Run SGD over multiple epochs

Source: http://wiki.epfl.ch/lions/simbaclustertorch

Others

- MatConvNet
  - http://www.vlfeat.org/matconvnet/
  - For Matlab users
  - Project started by Oxford VGG
  - Catching up with other libraries
- TensorFlow
  - https://www.tensorflow.org/
  - Open sourced by Google
  - Like Theano, not a rigid neural network library
  - Works on data flow graphs and tensors

How do I start with DL?

- Play with Pretrained models in your favorite software environments
- Do not start with imagenet training or heavy computational task, if you have not done many such in the past.

Modern Empirical Vision

- In modern vision:
  - Use downloaded software of successful projects
  - Reproduce their results
  - Change only one at a time
    - Data
    - Problem
    - Specific modules f the solution
    - Etc.

Comments

- Pretrained models
  - Save training time, efforts and computational resources by using pre-trained models from model zoo:
    - http://caffe.berkeleyvision.org/model_zoo.html
- DAG networks
  - Back propagation generalizes beyond sequential networks to DAGs. Support for DAGs is fast improving in all libraries

Comments

- Gradient checking
  - If library does not support automatic differentiation, it is useful to implement numerical gradient checking function to ascertain correctness
- Optimization methods
  - Most libraries allow many optimization methods and parameters
  - SGD, momemtum, adagrad, Nesterov, L-BGFS

Comments

- Off the shelf features
  - We can CNN as a feature extraction tool, by passing input image through the network and using output blobs of intermediate fully connected layers like fc6, fc7 of AlexNet as feature vectors
- Transfer learning
  - CNN trained on dataset like ImageNet can be finetuned to learn to classify other problems
  - In Caffe:
    - weights of new model ar initialized from serialized .caffemodel file and
    - 'prototxt' network architecture definitionis copied and last layer is modified to have number of classes as per the new problem
    - Learning rate is reduced

Comments

- Layer specific learning rates
  - Libraries like caffe allow for layer specific learning rates
- Minibatch size
  - Determines the amount of data transferred from CPU to GPU
- BoW features
  - CNN features can replace HoG / SIFT BoW features

Summary

- RNNs are becoming popular.
  - Direct contrast with HMMs and sequence/variable length representations.
  - Very powerful with lots of memory
- Many Libraries/options to start
  - Choice depends on use cases; many going strong.
- Hardware
  - Choice, Cost
  - GPUs are very popular
  - CPUs/Distributed solutions are also getting tried.

Thanks!!