Hello!

Practical deep neural nets for detecting marine mammals

daniel.nouri@gmail.com
@dnouri
Kaggle competitions

- 2 sec sounds → right whale upcall?
ICML2013 comp results (1)

• 47k examples, 10% positive
  – AUC: 0.988 (Kaggle valid set)
  – Accuracy: 97.3%

• 62k examples, 19% positive
  – AUC: 0.992 (Kaggle valid set)
  – Accuracy: 97.3%
ICML2013 comp results (2)

Confusion matrix

<table>
<thead>
<tr>
<th></th>
<th>no</th>
<th>3152</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>29</td>
<td>740</td>
<td></td>
</tr>
</tbody>
</table>
ICML2013 comp results (3)

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg</td>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
<td>3231</td>
</tr>
<tr>
<td>pos</td>
<td>0.90</td>
<td>0.96</td>
<td>0.93</td>
<td>769</td>
</tr>
<tr>
<td>avg</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>4000</td>
</tr>
</tbody>
</table>

Precision-Recall curve
Predictions
This presentation

1. Quick overview: deep learning
2. An implementation: cuda-convnet
3. Practical tips for better results
Neural networks

- Neural Networks
- find weights so that $h$ produces desired output

$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$
Deep neural networks

- “Deep” because many hidden layers
Deep learning: and the brain

- Fascinating idea: “one algorithm” hypothesis
- Rewire sensors auditory cortex → visual cortex, visual cortex will learn to hear
Deep learning: so what

- DNN not just a classifier, but also a very powerful feature extractor
- signal processing, filtering
- noise reduction
- contour extraction, per species
- (sometimes uninformed) assumptions
Deep learning: say what

- DNN not just a classifier, but also a very powerful feature extractor
- signal processing, filtering
- noise reduction
- contour extraction, per species
- (sometimes uninformed) assumptions
Deep learning: claim

• Big bold claim
  – less work
  – better results
• Challenge me!
Deep Learning: breakthrough

- recent breakthroughs in many fields:
  - Image recognition
  - Image search (autoencoder)
  - Speech recognition
  - Natural Language Processing
  - Passive acoustics for detecting mammals!
Deep learning: old ideas

- Backprop for training weights
- but training used to be hard
Deep learning: new things

• New developments that enabled breakthrough

• Much larger (deeper) nets; able to train them better through
  – GPUs (huge jump in performance)
  – more (labeled) data
  – 'relu' activation function
  – Dropout
Implementation: cuda-convnet

- by Alex Krizhevsky, Hinton's group
- Open Source and good docs
- examples included (CIFAR)
- code.google.com/p/cuda-convnet/
- very fast implementation of convolutional DNNs based on CUDA
- C++, Python
cuda-convnet: ILSVRC 2012

- Large Scale Visual Recognition Challenge 2012
- 1.2 million high-resolution training images
- 1000 object classes
- Winner code based on cuda-convnet
- Trained for a week on two GPUs
- 60 million parameters and 650,000 neurons
- 16.4% error versus 26.1% (2nd place)
cuda-convnet: ILSVRC 2012
**layers.cfg** defines architecture

```
[fc4]  # layer name
  type=fc  # type of layer
  inputs=fc3  # layer input
  outputs=512  # number of units
  initW=0.01  # weight initialization
  neuron=relu  # activation function
```
• **layers.cfg** defines many layers

```plaintext
[data] [fc3]
[resize] [fc4]
[conv1] [fc5]
[pool1] [probs]
[conv2] [logprob]
[pool2]
```
- `layer-params.cfg`
- defines additional params for layers in `layers.cfg`
- params that *may change* during training
- e.g. learning rate, regularization
cuda-convnet: input file format

- actual training data: data_batch_1, data_batch_2, ..., data_batch_n
- statistics (mean): batches_meta
- data_batch_1: “pickled dict” with {'data': Numpy array, 'labels': list}
- a few lines of Python
cuda-convnet: data provider

- Python class responsible for
  - reading data
  - passing it on to neural net
- example data layer included
- can adjust e.g. when dealing with grayscale, different cropping
python convnet.py
--data-path=../cifar-10-batches-py-colmajor/
--save-path=../tmp
--test-range=5
--train-range=1-4
--layer-def=layers.cfg
--layer-params=layer-params.cfg
--data-provider=cifar-cropped
--test-freq=13
--crop-border=4
--epochs=100
cuda-convnet: training (2)

- continue training from a snapshot
  python convnet.py -f 
  ../tmp/ConvNet__2013-06-14_15.5 4.31
  --epochs=110
cuda-convnet: prediction

- **input:** data_bach_x
- **output:** csv file, other formats
- [github.com/dnouri/noccn](https://github.com/dnouri/noccn)
  - predict script
Practical tips for better results

- Lots of hyperparameters
- most important params:
  - number and type of layers
  - number of units in layers
  - number of convolutional filters and their size
  - weight initialization
  - learning rates: epsW
  - weight decay
  - number of input dims
  - convolutional filter size
Practical: where to start

• Lots of parameters
• Automated grid search not feasible, at least not for bigger nets
• Need to start with “reasonable defaults”
• Standard architectures go a long way
Practical: try examples

- CIFAR-10 examples
- I worked on image classification problem when I started with upcall detection challenge
- feeding a spectogram into a very similar net gave great results already
Practical: overfit first

- Configure net to overfit first
- Add regularization later
- except maybe weight decay in conv layers: helps with learning
- Hinton: if your deep neural net isn't overfitting, it isn't big enough
Practical: init weights (1)

• fine-tuning net hyperparameters can take a long time

• net with better initialized weights trains much faster, thus reducing round-trip time for fine-tuning

• we initialize weights from a random distribution
Practical: init weights (2)

• play a little, compare training error of first epoch

• whatever trains faster, wins

• if you change number of units, you'll probably want to change scale of weight initialization, too
Practical: check filters

Noisy convolutional filters are bad for generalization
Practical: check weights

- make sure that all/many filters are active
- here: second conv layer
• DBNs: pre-training to learn weights
• use if you don't have a lot of labeled data
Practical: learning rate

- relatively easy to find good values
- too high: training error doesn't decrease
- too low: training error decreases slowly, gets stuck in local optimum
- reduce at end of training to get little more gain
Practical: weight decay

- pulls weights towards zero
- makes for “cleaner filters”
- don't use them for fully connected layers; instead use...
Practical: Dropout

- recent development
- effect similar to averaging many individual nets
- but faster to train and test
- dropout 0.5 in fully connected layers; sometimes 0.2 in input layers
- my best model uses dropout and overfits very little
Practical: data augmentation

- more data $\rightarrow$ better generalization
- augment data
  - at train time, mix example together with random negative example
Practical: cropping

- another way to augment data
- crop from 120x100 spectrogram window of 100x100
References (1)

- ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky 2012]
- Improving neural networks by preventing co-adaptation of feature detectors [Hinton 2012]
- Practical recommendations for gradient-based training of deep architectures [Bengio 2012]
References (2)

- code.google.com/p/cuda-convnet/
- github.com/dnouri/cuda-convnet
- github.com/dnouri/noccn
- daniel.nouri@gmail.com

• Thanks!