Deep Learning
and its application to CV and NLP

Fei Yan
University of Surrey
June 29, 2016
Edinburgh
Overview

- **Machine learning**
- Motivation: why go deep
- Feed-forward networks: CNN
- Recurrent networks: LSTM
- An example: geo-location prediction
- Conclusions
Machine learning

• Learn without explicitly programmed
• Humans are learning machines
• Supervised, unsupervised, reinforcement, transfer, multitask ...
ML for CV: image classification
ML for NLP: sentiment analysis

• “Damon has never seemed more at home than he does here, millions of miles adrift. Would any other actor have shouldered the weight of the role with such diligent grace?”

• “The warehouse deal TV we bought was faulty so had to return. However we liked the TV itself so bought elsewhere.”
ML for NLP: Co-reference resolution

• “John said he would attend the meeting.”

• “Barack Obama visited Flint Mich. on Wednesday since findings about the city’s lead-contaminated water came to light. ... The president said that ...”
Overview

• Machine learning
• Motivation: why go deep
• Feed-forward networks: CNN
• Recurrent networks: LSTM
• An example: geo-location prediction
• Conclusions
Motivation: why go deep

• A shallow cat/dog recogniser:
  – Convolve with fixed filters
  – Aggregate over image
  – Apply more filters
  – SVM

\[
\begin{pmatrix}
+1 & +2 & +1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{pmatrix}
\begin{pmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{pmatrix}
\]
Motivation: why go deep

• A shallow sentiment analyser:
  – Bag of words
  – Part-of-speech tagging
  – Named entity recognition
  – ...
  – SVM
Motivation: why go deep

• Shallow learner eg SVM
  – Convexity -> global optimum
  – Good performance
  – Small training sets

• But features manually engineered
  – Domain knowledge required
  – Representation and learning decoupled ie not end-to-end learning
Overview

• Machine learning
• Motivation: why go deep
• Feed-forward networks: CNN
• Recurrent networks: LSTM
• An example: geo-location prediction
• Conclusions
From shallow to deep

\[
\begin{pmatrix}
+1 & +2 & +1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{pmatrix}
\]
From shallow to deep

\[ W^{1,1} = \begin{pmatrix} w_{11}^{1,1} & w_{12}^{1,1} & w_{13}^{1,1} \\ w_{21}^{1,1} & w_{22}^{1,1} & w_{23}^{1,1} \\ w_{31}^{1,1} & w_{32}^{1,1} & w_{33}^{1,1} \end{pmatrix} \]

\[ W^{1,2} = \begin{pmatrix} w_{11}^{1,2} & w_{12}^{1,2} & w_{13}^{1,2} \\ w_{21}^{1,2} & w_{22}^{1,2} & w_{23}^{1,2} \\ w_{31}^{1,2} & w_{32}^{1,2} & w_{33}^{1,2} \end{pmatrix} \]

...
From shallow to deep

• 100x100x1 input
• 10 3x3x1 filters
• # of params:
  – 10x3x3x1=90
• Size of output:
  – 100x100x10 with padding and stride=1
From shallow to deep

- 100x100x10 input
- 8 3x3x10 filters
- # of params:
  - 8x3x3x10 = 720
- Size of output:
  - 100x100x8 with padding and stride=1
Other layers

• Rectified linear unit (ReLU)
• Max pooling
  – Location invariance
• Dropout
  – Effective regularisation
• Fully-connected (FC)
Complete network

• Loss:
  – Softmax loss for problem
  – How wrong current prediction is
  – How to change FC8 output to reduce error
Chain rule

• If $y$ is a function of $u$, and $u$ is a function of $x$

\[
\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}
\]

• DNNs are nested functions
  – Output of one layer is input of next
Back-propagation

• If a layer has parameters
  – Convolution, FC
  – O is function of Input I and parameters W
    \[
    \frac{\partial L}{\partial W} = \frac{dL}{dO} \cdot \frac{\partial O}{\partial W} \quad \frac{\partial L}{\partial I} = \frac{dL}{dO} \cdot \frac{\partial O}{\partial I}
    \]

• If a layer doesn’t have parameters
  – Pooling, ReLU, Dropout
  – O is a function of input I only
    \[
    \frac{dL}{dI} = \frac{dL}{dO} \cdot \frac{dO}{dI}
    \]
Stochastic gradient descent (SGD)

- Stochastic: random mini-batch
- **Weight update**: linear combination of
  - Negative gradient of current batch
  - Previous weight update

\[ V_{t+1} = \mu V_t - \alpha \nabla L(W_t) , \quad W_{t+1} = W_t + V_{t+1} \]

- \( \alpha \) : learning rate; \( \mu \) : momentum
- Other variants
  - Adadelta, AdaGrad, etc.
Why SGD works

• Deep NNs are non-convex
• Most critical points in high dimensional functions are saddle points
• SGD can escape from saddle points
Loss vs. iteration
ImageNet and ILSVRC

• **ImageNet**
  – # of images: 14,197,122, labelled
  – # of classes: 21,841

• **ILSVRC 2012**
  – # of classes: 1,000
  – # of train image: ~1,200,000, labelled
  – # of test image: 50,000
AlexNet

- [Krizhevsky et al. 2012]
- Conv1: 96 11x11x3 filters, stride=4
- Conv3: 384 3x3x256 filters, stride=1
- FC7: 4096 channels
- FC8: 1000 channels
AlexNet

- Total # of params: ~60,000,000
- Data augmentation
  - Translation, reflections, RGB shifting
- 5 days, 2 x Nvidia GTX 580 GPUs
- Significantly improves state-of-the-art
- Breakthrough in computer vision
More recent nets

AlexNet 2012 vs GoogleNet 2014
Hierarchical representation

Visualisation of learnt filters. [Zeiler & Fergus 2013]
Hierarchical representation

Visualisation of learnt filters. [Lee et al. 2012]
CNN as generic feature extractor

• Given:
  – CNN trained with eg ImageNet
  – A new recognition task/dataset

• Simply:
  – Forward pass, take FC7/ReLU7 output
  – SVM

• Often outperform hand crafted features
CNN as generic feature extractor

Image retrieval with trained CNN. [Krizhevsky et al. 2012]
Neural artistic style
Neural artistic style

• Key idea
  – Hierarchical representation
    => content and style are separable
  – Content: filter responses
  – Style: correlations of filter responses
Neural artistic style

• Input
  – Natural image: content
  – Image of artwork: style
  – Random noise image

• Define content loss and style loss

• Update a random image with BP to minimise:

\[ L_{total}(\vec{p}, \vec{d}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{d}, \vec{x}) \]
Neural artistic style

[Gatys et al. 2015]
Go game
CNN for Go game

- Treated as 19x19 image
- Convolution with zero-padding
- ReLU nonlinearity
- Softmax loss of size 361 (19x19)
- SGD as solver
- No Pooling
AlphaGo

- Policy CNN
  - Configuration -> choice of professional players
  - Trained with 30K+ professional games

- Simulate till end to get binary labels

- Value CNN
  - Configuration -> win/loss
  - Trained with 30M+ simulated games

- Reinforcement learning, Monte-Carlo tree search

- 1202 CPUs + 176 GPUs

- Beating 18 times world champion
Why it didn’t work

• Ingredients available in 80s
  – (Deep) Neural networks
  – Convolutional filters
  – Back-propagation

• But
  – Dataset thousands times smaller
  – Computers millions times slower

• Recent techniques/heuristics help
  – Dropout, ReLU
Overview

• Machine learning
• Motivation: why go deep
• Feed-forward networks: CNN
  • **Recurrent networks:** LSTM
• An example: geo-location prediction
• Conclusions
Why recurrent nets

• Feed-forward nets
  – Process independent vectors
  – Optimise over functions

• Recurrent nets
  – Process sequences of vectors
  – Internal state, or “memory”
  – Dynamic behaviour
  – Optimise over programs, much more powerful
Unfolding recurrent nets in time
LSTM

- Input, forget and output gates: $i, f, o$
- Internal state: $c$

\[ i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1}) \]
\[ f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1}) \]
\[ o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1}) \]
\[ g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1}) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot g_t \]
\[ h_t = o_t \odot \phi(c_t) \]

[Donahue et al. 2014]
Machine translation

• **Sequence to sequence mapping**
  – ABC<E> => WXYZ<E>

• **Traditional MT:**
  – Hand-crafted intermediate semantic space
  – Hand-crafted features
Machine translation

- **LSTM based MT:**
  - Maximise prob. of output given input
  - Update weights in LSTM by BP in time
  - End-to-end, no feature-engineering
  - Semantic information in LSTM cell

[Sutskever et al. 2014]
Image captioning

• Image classification
  – Girl/child, tree, grass, flower

• Image captioning
  – Girl in pink dress is jumping in the air
  – A girl jumps on the grass
Image captioning

• Traditional methods
  – Object detector
  – Surface realiser: objects => sentence

• LSTM
  – Inspired by neural machine translation
  – Translate image into sentence
Image captioning

[Vinyals et al. 2014]
Overview

• Machine learning
• Motivation: why go deep
• Feed-forward networks: CNN
• Recurrent networks: LSTM
• **An example: geo-location prediction**
• Conclusions
News article analysis

• BreakingNews dataset
  – 100k+ news articles
  – 7 sources: BBC, Yahoo, WP, Guardian, ...
  – Image + caption
  – Metadata: comments, geo-location, ...

• Tasks
  – Article illustration
  – Caption generation
  – Popularity prediction
  – Source prediction
  – Geo-location prediction
Geo-location prediction
Word2Vec embedding

- **Word embedding**
  - Words to vectors
  - Low dim. compared to vocabulary size

- **Word2Vec**
  - Unsupervised, neural networks [Mikolov et al. 2015]
  - Trained on large corpus eg 100+ billion words
  - Vectors close if similar context
Word2Vec embedding

• W2V arithmetic
  – King - Queen ~ man - woman
  – knee - leg ~ elbow - arm
  – China - Beijing ~ France - Paris
  – human - animal ~ ethics
  – library - book ~ hall
  – president - power ~ prime minister
Network

W2V matrix → Convolution → Pooling → FC → Nonlinearity → Dropout → FC_o → Loss → Label
Geoloc loss

- **Great circle**
  - Circle on sphere with same centre as the sphere

- **Great circle distance (GCD)**
  - Distance along great circle
  - Shortest distance on sphere
Geoloc loss

• Given two (lat, long) pairs

\[ y = [y_1, y_2]^T \quad z = [z_1, z_2]^T \]

• A good approximation to GCD

\[ \text{GCD} = R \cdot \arccos(\sin y_1 \sin z_1 + \cos y_1 \cos z_1 \cos \delta) \]

where R is radius of Earth, and

\[ \delta = z_2 - y_2 \]

• Geoloc loss

\[ L = \arccos(\sin y_1 \sin z_1 + \cos y_1 \cos z_1 \cos \delta) \]
Geoloc loss

\[ z = (z_1, z_2)^T \]

\[ y = (y_1, y_2)^T \]
Geoloc loss

- Gradient w.r.t. $z$

$$
\frac{\partial L}{\partial z} = \begin{pmatrix}
- \frac{1}{\sqrt{1-\phi^2}} (\sin y_1 \cos z_1 - \cos y_1 \sin z_1 \cos \delta) \\
- \frac{1}{\sqrt{1-\phi^2}} (-\cos y_1 \cos z_1 \sin \delta)
\end{pmatrix}
$$

where

$$\phi = \sin y_1 \sin z_1 + \cos y_1 \cos z_1 \cos \delta$$

- All other layers are standard
- Chain rule, back-propagation, etc.
Practical issues

• Hardware
  – Get a powerful GPU

• Software
  – Choose a library

• What code do I need to write?
  – Solver def. and net def.
  – Optionally: your own layer(s)
## Libraries

<table>
<thead>
<tr>
<th>Software</th>
<th>Written in</th>
<th>Interface</th>
<th>OpenMP support</th>
<th>OpenCL support</th>
<th>CUDA support</th>
<th>Has pretrained models</th>
<th>Recurrent Nets</th>
<th>Convolutional Nets</th>
<th>RBM/DBNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Designer</td>
<td>C++</td>
<td>Graphical user interface</td>
<td>Yes</td>
<td>No</td>
<td>No[24]</td>
<td>?</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OpenNN</td>
<td>C++</td>
<td>C++</td>
<td>Yes</td>
<td>No</td>
<td>No[25]</td>
<td>?</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SINGA[26]</td>
<td>C++, Python</td>
<td>Python, C++</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No[26]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>C++, Python</td>
<td>Python, C/C++</td>
<td>No</td>
<td>On roadmap[21][23]</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Theano</td>
<td>Python</td>
<td>Python</td>
<td>Yes</td>
<td>Under development[54]</td>
<td>Yes</td>
<td>Through Lasagne's model zoo[25]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Wikipedia: comparison of deep learning software
What you need to code

• solver.prototxt
  – Solver hyper-params

• train.prototxt
  – Network architecture
  – Layer hyper-params

• Layer implementation C++/CUDA
  – Forward pass
  – Backward propagation
  – Efficient GPU programming, CUDA kernel
solver.prototxt & train.prototxt

1 net: "train.prototxt"
2
3 solver_mode: GPU
4 device_id: 0
5
6 type: "SGD"
7 momentum: 0.9
8 weight_decay: 0.0005
9
10 base_lr: 0.1
11 lr_policy: "step"
12 gamma: 0.1
13 stepsize: 1000
14
15 max_iter: 1000000
16
17 display: 20
18 test_iter: 1000
19 test_interval: 500
20
21 snapshot: 500
22 snapshot_prefix: "geolo"
23

145 layer {
146    name: "PCo"
147    type: "InnerProduct"
148    bottom: "D0"
149    top: "PCo"
150    param {
151      lr_mult: 1
152      decay_mult: 1
153    }
154    param {
155      lr_mult: 2
156      decay_mult: 0
157    }
158    inner_product_param {
159      num_output: 2
160      weight_filler {
161        type: "xavier"
162      }
163      bias_filler {
164        type: "constant"
165      }
166    }
167    }  
168
169 layer {
170    name: "Geo_loss"
171    type: "GeoLoss"
172    bottom: "PCo"
173    bottom: "Geo"
174    top: "Geo_loss"
175    loss_weight: 1
176    geo_loss_param {
177      verbosity: 0
178    }
179  }
Overview

• Machine learning
• Motivation: why go deep
• Feed-forward networks: CNN
• Recurrent networks: LSTM
• An example: geo-location prediction
• Conclusions
Conclusions

• Why go deep
• CNN and LSTM
• Example: geo-location prediction
• Apply DL to my problem:
  – CNN or LSTM?
  – Network architecture, loss
  – Library and GPU
  – (Little) Coding
What’s not covered

• **Unsupervised learning**
  – Auto-encoder, restricted Boltzmann machine (RBM)

• **Reinforcement learning**
  – Actions in an environment that maximise cumulative reward

• **Transfer learning, Multitask learning**

• **Application to audio signal processing**