Deep Learning

Michèle Sebag
TAO

Université Paris-Saclay

Jan. 21st, 2016

Credit for slides: Yoshua Bengio, Yann Le Cun, Nando de Freitas, Christian Perone, Honglak Lee, Ronan Collobert, Tomas Mikolov, Rich Caruana
Overview

Neural Nets
   Main ingredients
   Invariances and convolutional networks

Deep Learning

Deep Learning Applications
   Computer vision
   Natural language processing
   Deep Reinforcement Learning
   The importance of being deep, revisited

Take-home message
History

1943  A neuron as a computable function $y = f(x)$  Pitts, McCullough  
      Intelligence $\rightarrow$ Reasoning $\rightarrow$ Boolean functions

1960  Connexionism $+$ learning algorithms  Rosenblatt

1969  AI Winter  Minsky-Papert

1989  Back-propagation  Amari, Rumelhart & McClelland, LeCun

1995  Winter again  Vapnik

2005  Deep Learning  Bengio, Hinton
One neuron: input, weights, activation function

\[ x \in \mathbb{R}^d \quad z = \sum_i w_i x_i \quad f(z) \in \mathbb{R} \]

Activation functions

- Thresholded: \( 0 \) if \( z < \text{threshold} \), \( 1 \) otherwise
- Linear: \( z \)
- Sigmoid: \( \frac{1}{1 + e^{-z}} \)
- Tanh: \( \frac{e^z - e^{-z}}{e^z + e^{-z}} \)
- Radius-based: \( e^{-z^2/\sigma^2} \)
- Rectified linear (ReLU): \( \max(0, z) \)
Learning the weights

An optimization problem: Define a criterion

- Supervised learning classification, regression

\[ \mathcal{E} = \{(x_i, y_i), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1 \ldots n\} \]

- Reinforcement learning

\[ \pi : \text{State space } \mathbb{R}^d \mapsto \text{Action space } \mathbb{R}^{d'} \]

Mnih et al., 2015

Main issues

- Requires a differentiable / continuous activation function
- Non convex optimization problem
Back-propagation, 1

**Notations**

Input \( x = (x_1, \ldots x_d) \)

From input to the first hidden layer

\[
    z_j^{(1)} = \sum w_{jk} x_k \\
    x_j^{(1)} = f(z_j^{(1)})
\]

From layer \( i \) to layer \( i + 1 \)

\[
    z_j^{(i+1)} = \sum w_{jk}^{(i)} x_k^{(i)} \\
    x_j^{(i+1)} = f(z_j^{(i+1)})
\]

(\( f \): e.g. sigmoid)
Input \((x, y)\), \(x \in \mathbb{R}^d\), \(y \in \{-1, 1\}\)

**Phase 1** Propagate information forward

- For layer \(i = 1 \ldots \ell\)
  - For every neuron \(j\) on layer \(i\)
    \[
    z^{(i)}_j = \sum_k w^{(i)}_{j,k} x^{(i-1)}_k \\
    x^{(i)}_j = f(z^{(i)}_j)
    \]

**Phase 2** Compare the target output \((y)\) to what you get \((x^{(\ell)}_1)\)

*assuming scalar output for simplicity*

- Error: difference between \(\hat{y} = x^{(\ell)}_1\) and \(y\).
  Define
  \[
  e^{\text{output}} = f'(z^{\ell}_1) [\hat{y} - y]
  \]
  where \(f'(t)\) is the (scalar) derivative of \(f\) at point \(t\).
Back-propagation, 3

Phase 3 retro-propagate the errors

\[ e_j^{(i-1)} = f'(z_j^{(i-1)}) \sum_k w_{kj}^{(i)} e_k^{(i)} \]

Phase 4: Update weights on all layers

\[ \Delta w_{ij}^{(k)} = \alpha e_i^{(k)} x_j^{(k-1)} \]

where \( \alpha \) is the learning rate \(< 1\).

Adjusting the learning rate is a main issue
Properties of NN

**Good news**
- MLP, RBF: universal approximators
  For every decent function $f (\equiv f^2$ has a finite integral on every compact of $\mathbb{R}^d$) for every $\epsilon > 0$, there exists some MLP/RBF $g$ such that $||f - g|| < \epsilon$.

**Bad news**
- Not a constructive proof (the solution exists, so what ?)
- Everything is possible $\rightarrow$ no guarantee (overfitting).

**Very bad news**
- A non convex (and hard) optimization problem
- Lots of local minima
- Low reproducibility of the results
The curse of NNs

The NIPS community has suffered of an acute convexivitis epidemic

▷ ML applications seem to have trouble moving beyond logistic regression, SVMs, and exponential-family graphical models.
▷ For a new ML model, convexity is viewed as a virtue
▷ Convexity is sometimes a virtue
▷ But it is often a limitation

▷ ML theory has essentially never moved beyond convex models
  ☑ the same way control theory has not really moved beyond linear systems

▷ Often, the price we pay for insisting on convexity is an unbearable increase in the size of the model, or the scaling properties of the optimization algorithm \[O(n^2), O(n^3)\]...

http://videolectures.net/eml07_lecun_wia/
Old Key Issues (many still hold)

Model selection
- Selecting number of neurons, connexion graph
- Which learning criterion

More $\nRightarrow$ Better
avoid overfitting

Algorithmic choices
- a difficult optimization problem
  - Enforce stability through relaxation
    \[
    W_{\text{neo}} \leftarrow (1 - \alpha)W_{\text{old}} + \alpha W_{\text{neo}}
    \]
  - Decrease the learning rate $\alpha$ with time
  - Stopping criterion

Tricks
- Normalize data
- Initialize $W$ small!
Overview

Neural Nets
- Main ingredients
- Invariances and convolutional networks

Deep Learning

Deep Learning Applications
- Computer vision
- Natural language processing
- Deep Reinforcement Learning
- The importance of being deep, revisited

Take-home message
Toward deeper representations

Invariances matter

- The label of an image is invariant through small translation, homothety, rotation...
- Invariance of labels $\rightarrow$ Invariance of model

$$y(x) = y(\sigma(x)) \rightarrow h(x) = h(\sigma(x))$$

Enforcing invariances

- by augmenting the training set:
  $$\mathcal{E} = \{(x_i, y_i)\} \bigcup \{ (\sigma(x_i), y_i) \}$$
- by structuring the hypothesis space

Convolutional networks
Hubel & Wiesel 1968

Visual cortex of the cat

- cells arranged in such a way that
- ... each cell observes a fraction of the visual field
- ... their union covers the whole field

- Layer \( m \): detection of local patterns (same weights)
- Layer \( m + 1 \): non linear aggregation of output of layer \( m \)
Ingredients of convolutional networks

1. Local receptive fields (aka kernel or filter)

2. Sharing weights
   through adapting the gradient-based update: the update is averaged over all occurrences of the weight.
   Reduces the number of parameters by several orders of magnitude
Ingredients of convolutional networks, 2

3. Pooling: reduction and invariance

- Overlapping / non-overlapping regions
- Return the max / the sum of the feature map over the region
- Larger receptive fields (see more of input)
Convolutional networks, summary

Properties

- Invariance to small transformations (over the region)
- Reducing the number of weights

LeCun 1998
Convolutional networks, summary

Properties

- Invariance to small transformations (over the region)
- Reducing the number of weights
- Usually many convolutional layers
Overview

Neural Nets
  Main ingredients
  Invariances and convolutional networks

Deep Learning

Deep Learning Applications
  Computer vision
  Natural language processing
  Deep Reinforcement Learning
  The importance of being deep, revisited

Take-home message
Manifesto for Deep

1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on human labor

3. Abstraction is mandatory

Bengio, Hinton 2006
Manifesto for Deep

1. Grand goal: AI

2. Requisites
   ▶ Computational efficiency
   ▶ Statistical efficiency
   ▶ Prior efficiency: architecture relies on student labor

3. Abstraction is mandatory
Manifesto for Deep

1. Grand goal: AI

2. Requisites
   ▶ Computational efficiency
   ▶ Statistical efficiency
   ▶ Prior efficiency: architecture relies on student labor

3. Abstraction is mandatory

4. Compositionality principle:
Manifesto for Deep

1. Grand goal: AI

2. Requisites
   - Computational efficiency
   - Statistical efficiency
   - Prior efficiency: architecture relies on student labor

3. Abstraction is mandatory

4. Compositionality principle:
   *build skills on the top of simpler skills*  
   
   Piaget
The importance of being deep

A toy example: \( n \)-bit parity

Pros: efficient representation

Deep neural nets are (exponentially) more compact

Cons: poor learning

- More layers \( \rightarrow \) more difficult optimization problem
- Getting stuck in poor local optima.
Overcoming the learning problem

Long Short Term Memory


Deep Belief Networks


Auto-Encoders

Auto-encoders

\[ \mathcal{E} = \{(x_i, y_i), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1 \ldots n\} \]

First layer

\[ x \rightarrow h_1 \rightarrow \hat{x} \]

- An auto-encoder:

\[
\text{Find } W^* = \arg\min_W \left( \sum_i \|W^t oW(x_i) - x_i\|^2 \right)
\]

\[
\sum_j x_{i,j} \log \hat{x}_{i,j} + (1 - x_{i,j}) \log (1 - \hat{x}_{i,j})
\]

\((*)\) Instead of min squared error, use cross-entropy loss:
Auto-encoders, 2

First layer

Second layer

\[ x \rightarrow h_1 \rightarrow \hat{x} \]

\[ h_1 \rightarrow h_2 \rightarrow \hat{h}_1 \]

same, replacing \( x \) with \( h_1 \)
Discussion

Layerwise training

- Less complex optimization problem (compared to training all layers simultaneously)
- Requires a local criterion: e.g. reconstruction
- Ensures that layer $i$ encodes same information as layer $i + 1$
- But in a more abstract way:
  - layer 1 encodes the patterns formed by the (descriptive) features
  - layer 2 encodes the patterns formed by the activation of the previous patterns
- When to stop? trial and error.
Discussion

Layerwise training

▶ Less complex optimization problem (compared to training all layers simultaneously)

▶ Requires a local criterion: e.g. reconstruction

▶ Ensures that layer $i$ encodes same information as layer $i + 1$

▶ But in a more abstract way:
  layer 1 encodes the patterns formed by the (descriptive) features
  layer 2 encodes the patterns formed by the activation of the previous patterns

▶ When to stop? Trial and error.

Now pre-training is almost obsolete

Gradient problems better understood

▶ Initialization

▶ New activation

▶ Regularization

▶ More data

▶ Better optimization algorithms
**Dropout**

**Why**
- Ensemble learning is effective
- But training several Deep NN is too costly
- The many neurons in a large DNN can “form coalitions”.
- Not robust!

**How**
- **During training**
  - For each hidden neuron, each sample, each iteration
  - For each input (of this hidden neuron)
  - with probability $p$ (.5), zero the input
  - (double the # iterations needed to converge)
- **During validation/test**
  - use all input
  - rescale the sum ($\times p$) to preserve average
Recommendations

Ingredients

- ReLU non-linearities
- Cross-entropy loss for classification
- Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- If you cannot overfit, increase the model size; if you can, regularize.

Regularization

- $L_2$ penalizes large weights
- $L_1$ penalizes non-zero weights

Adaptive learning rate

- adjusted per neuron to fit the moving average of the last gradients

Hyper-parameters

- Grid search
- Continue training the most promising model

Not covered

- Long Short Term Memory
- Restricted Boltzman Machines
- Natural gradient
Overview

Neural Nets
- Main ingredients
- Invariances and convolutional networks

Deep Learning

Deep Learning Applications
- Computer vision
- Natural language processing
- Deep Reinforcement Learning
- The importance of being deep, revisited

Take-home message
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton,
Advances in Neural Information Processing Systems 2012

ImageNet

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazons Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (1000 per category)
  - 50,000 validation images
  - 150,000 testing images
- RGB images with variable-resolution
ImageNet

Evaluation

- Guess it right
top-1 error
- Guess the right one among the top 5
top-5 error
What is new?

Former state of the art

SIFT

Spin image

HoG

RIFT

Textons

GLOH

SIFT: scale invariant feature transform
HOG: histogram of oriented gradients
Textons: “vector quantized responses of a linear filter bank”
What is new, 2

Traditional approach

→ Manually crafted features → Trainable classifier

Deep learning

→ Trainable feature extractor → Trainable classifier
DNN. 1, Tractability

Activation function
- On CIFAR-10: Relu 6 times faster than tanh

Data augmentation
- Translation and horizontal symmetries
- Alter RGB intensities
  - PCA, with \((p, \lambda)\) eigen vector, eigen value
  - Add: \((p_1, p_2, p_3) \times (\alpha \lambda_1, \alpha \lambda_2, \alpha \lambda_3)^t\) to each image, with \(\alpha \sim U[0, 1]\)

4 layers convolutional
DNN. 2, Architecture

- 1st layer: 96 kernels ($11 \times 11 \times 3$; stride 3)
- Normalized, pooled
- 2nd layer: 256 kernels ($5 \times 5 \times 48$).
- Normalized, pooled
- 3rd layer: 384 kernels ($3 \times 3 \times 256$)
- 4th layer: 384 kernels ($3 \times 3 \times 192$)
- 5th layer: 256 kernels ($3 \times 3 \times 192$)
- followed by 2 fully connected layers, 4096 neurons each
DNN. 3, Details

Pre-processing
  ▶ Variable-resolution images → i) down-sampling; ii) rescale
  ▶ subtract mean value for each pixel

Results on the test data
  ▶ top-1 error rate: 37.5%
  ▶ top-5 error rate: 17.0%

Results on ILSVRC-2012 competition
  ▶ 15.3% accuracy
  ▶ 2nd best team: 26.2% accuracy
“Understanding” the result

Interpreting a neuron:
Plotting the input (image) which maximally excites this neuron.

20 millions image from YouTube
“Understanding” the result, 2

Interpreting the representation: Plotting the induced topology
http://cs.stanford.edu/people/karpathy/cnnembed/
Overview

Neural Nets
- Main ingredients
- Invariances and convolutional networks

Deep Learning

Deep Learning Applications
- Computer vision
- Natural language processing
- Deep Reinforcement Learning
- The importance of being deep, revisited

Take-home message
Natural Language Processing
Dimensionality: 20K (speech) 50K (Penn TB) 500K (big vocab) 13M (Google)

Bag-of-words

\[
\text{motel} \left[ \begin{array}{cccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \text{ AND } \\
\text{hotel} \left[ \begin{array}{cccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \right] = 0
\]

Latent representations

Latent Semantic analysis

▶ Input: matrix (documents × words)
▶ You know a word by the company it keeps
▶ Dimensionality reduction
  – high dimensional, sparsity issue, scales quadratically, update problematic

also, something non additive is needed: not bad \(\not\equiv\) not + bad
NLP: which learning criterion?

Criterion for learning
1. predict a linguistic label
2. predict a class
3. predict the neighborhood of words

The labelling cost
- 1, 2 requires labels
- 3: can be handled in an unsupervised way

Criterion for evaluation
- evaluate relationships
Continuous language models

Bengio et al. 2001

Principle

- Input: 10,000-dim boolean input (words)
- Hidden layer: 500 continuous neurons
- Output: from a text window \( w_i \ldots w_{i+k} \), predict
  - The grammatical tag of the central word \( w_{i+k} \)
  - Other: see next

Trained embeddings

- Hidden layer defines a mapping from a text window onto \( \mathbb{R}^{500} \)
- Applicable to any discrete space
Continuous language models

The window approach
- Fixed size window works fine for some tasks
- Does not deal with long-range dependencies

The sentence approach
- Feed the whole sentence to the network
- Convolutions to handle variable-length inputs
- Convert network outputs into probabilities

\[ p(i) = \frac{\exp(f(i,x,\theta))}{\sum_j \exp(f(j,x,\theta))} \]

- Maximize log likelihood

Find \( \theta^* = \arg \max \log(p(y|x, \theta)) \)

Results
- Small improvements
- 15% of most frequent words in the dictionary are seen 90% of the time...
Going unlabelled

Idea: a lesion study

- Take a sentence from Wikipedia: label true
- Replace middle word with random word: label false
- Tons of labelled data, 0-cost labels
- Captures semantics and syntax
Training Language Model

- Two window approach (11) networks (100HU) trained on two corpus:
  - LM1: Wikipedia: **631M** of words
  - LM2: Wikipedia + Reuters RCV1: **631M + 221M = 852M** of words

- Massive dataset: cannot afford classical training-validation scheme

- Like in biology: breed a couple of network lines

- Breeding decisions according to 1M words validation set

- LM1
  - order dictionary words by frequency
  - increase dictionary size: 5000, 10,000, 30,000, 50,000, 100,000
  - 4 weeks of training

- LM2
  - initialized with LM1, dictionary size is 130,000
  - 30,000 additional most frequent Reuters words
  - 3 additional weeks of training
<table>
<thead>
<tr>
<th>Country</th>
<th>Word</th>
<th>MDB</th>
<th>Color</th>
<th>Action</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Jesus</td>
<td>Xbox</td>
<td>Reddish</td>
<td>Scratched</td>
<td>Megabits</td>
</tr>
<tr>
<td>Austria</td>
<td>God</td>
<td>Amiga</td>
<td>Greenish</td>
<td>Nailed</td>
<td>Octets</td>
</tr>
<tr>
<td>Belgium</td>
<td>Sati</td>
<td>Playstation</td>
<td>Bluish</td>
<td>Smashed</td>
<td>MB/s</td>
</tr>
<tr>
<td>Germany</td>
<td>Christ</td>
<td>MSX</td>
<td>Pinkish</td>
<td>Punched</td>
<td>Bit/s</td>
</tr>
<tr>
<td>Italy</td>
<td>Satan</td>
<td>iPod</td>
<td>Purplish</td>
<td>Popped</td>
<td>Baud</td>
</tr>
<tr>
<td>Greece</td>
<td>Kali</td>
<td>Sega</td>
<td>Brownish</td>
<td>Crimped</td>
<td>Carats</td>
</tr>
<tr>
<td>Sweden</td>
<td>Indra</td>
<td>PSNUMBER</td>
<td>Greyish</td>
<td>Scraped</td>
<td>Kbit/s</td>
</tr>
<tr>
<td>Norway</td>
<td>Vishnu</td>
<td>HD</td>
<td>Grayish</td>
<td>Screwed</td>
<td>Megahertz</td>
</tr>
<tr>
<td>Europe</td>
<td>Ananda</td>
<td>Dreamcast</td>
<td>Whitish</td>
<td>Sectioned</td>
<td>Megapixels</td>
</tr>
<tr>
<td>Hungary</td>
<td>Parvati</td>
<td>GeForce</td>
<td>Silvery</td>
<td>Slashed</td>
<td>Gbit/s</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Grace</td>
<td>Capcom</td>
<td>Yellowish</td>
<td>Ripped</td>
<td>Amperes</td>
</tr>
</tbody>
</table>
Continuous language models, Collobert et al. 2008

MTL: Semantic Role Labeling

We get: 14.30%. State-of-the-art: 16.54% – Pradhan et al. (2004)

250× faster than state-of-the-art. ~ 0.01 s to label a WSJ sentence.
Word to Vec

Continuous bag of words model

- input, projection layer, hidden layer (linear), output

- Adds input from window to predict the current word
- Shares the weights for different positions
- Very efficient

Mikolov et al., 13, 14
https://code.google.com/p/word2vec/
Word to Vec, two models
Computational aspects

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training Words</th>
<th>Training Time</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert NNLM</td>
<td>50</td>
<td>660M</td>
<td>2 months</td>
<td>11</td>
</tr>
<tr>
<td>Turian NNLM</td>
<td>200</td>
<td>37M</td>
<td>few weeks</td>
<td>2</td>
</tr>
<tr>
<td>Mnih NNLM</td>
<td>100</td>
<td>37M</td>
<td>7 days</td>
<td>9</td>
</tr>
<tr>
<td>Mikolov RNNLM</td>
<td>640</td>
<td>320M</td>
<td>weeks</td>
<td>25</td>
</tr>
<tr>
<td>Huang NNLM</td>
<td>50</td>
<td>990M</td>
<td>weeks</td>
<td>13</td>
</tr>
<tr>
<td>Skip-gram (hier.s.)</td>
<td>1000</td>
<td>6B</td>
<td>hours</td>
<td>66</td>
</tr>
<tr>
<td>CBOW (negative)</td>
<td>300</td>
<td>1.5B</td>
<td>minutes</td>
<td>72</td>
</tr>
</tbody>
</table>

Tricks

- Undersample frequent words (the, is, ...)
- Linear hidden layer
- Negative sampling: only the output neuron that represents the positive class + few randomly sampled neurons are evaluated
- Output neurons: independent logistic regression classifiers
- → training speed independent of vocabulary size
Word vectors – nearest neighbors

<table>
<thead>
<tr>
<th></th>
<th>Redmond</th>
<th>Havel</th>
<th>graffiti</th>
<th>capitulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert NNLM</td>
<td>conyers</td>
<td>plauen</td>
<td>cheesecake</td>
<td>abdicate</td>
</tr>
<tr>
<td></td>
<td>lubbock</td>
<td>dzerzhinsky</td>
<td>gossip</td>
<td>accede</td>
</tr>
<tr>
<td></td>
<td>keene</td>
<td>osterreich</td>
<td>dioramas</td>
<td>rearm</td>
</tr>
<tr>
<td>Turian NNLM</td>
<td>McCarthy</td>
<td>Jewell</td>
<td>gunfire</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Alston</td>
<td>Arzu</td>
<td>emotion</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Cousins</td>
<td>Ovitz</td>
<td>impunity</td>
<td>-</td>
</tr>
<tr>
<td>Mnih NNLM</td>
<td>Podhurst</td>
<td>Pontif</td>
<td>anaesthetics</td>
<td>Mavericks</td>
</tr>
<tr>
<td></td>
<td>Harlang</td>
<td></td>
<td>monkeys</td>
<td>planning</td>
</tr>
<tr>
<td></td>
<td>Agarwal</td>
<td>Rodionov</td>
<td>Jews</td>
<td>hesitated</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>Redmond Wash.</td>
<td>Vaclav Havel</td>
<td>spray paint</td>
<td>capitulation</td>
</tr>
<tr>
<td>(phrases)</td>
<td>Redmond Washington</td>
<td>president Vaclav Havel</td>
<td>graffiti</td>
<td>capitulated</td>
</tr>
<tr>
<td></td>
<td>Microsoft</td>
<td>Velvet Revolution</td>
<td>taggers</td>
<td>capitulating</td>
</tr>
</tbody>
</table>

- More training data helps the quality a lot!
Word vectors – more examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris - France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>bigger - big + cold</td>
<td>colder</td>
</tr>
<tr>
<td>sushi - Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu - copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows - Microsoft + Google</td>
<td>Android</td>
</tr>
<tr>
<td>Montreal Canadiens - Montreal + Toronto</td>
<td>Toronto Maple Leafs</td>
</tr>
</tbody>
</table>
Word vectors – visualization using PCA
Overview

Neural Nets
  Main ingredients
  Invariances and convolutional networks

Deep Learning

Deep Learning Applications
  Computer vision
  Natural language processing
  Deep Reinforcement Learning
  The importance of being deep, revisited

Take-home message
Deep Reinforcement Learning

Reinforcement Learning in one slide

- State space $S$
- Action space $A$
- Transition model $p(s, a, s') \mapsto [0, 1]$
- Reward $r(s)$

Value functions and policies

$$V^\pi(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s') V^\pi(s')$$

$$V^*(s) = \max_\pi V^\pi(s')$$

$$\pi^*(s) = \arg\max_{a \in A} \left\{ \sum_{s'} p(s, a, s') V^*(s') \right\}$$
Playing Atari

**Input**: 4 consecutive frames
- $84 \times 84$ (reduced, gray-scaled) pixels $\times$ 4 (last four frames)

**Architecture**
- 1st hidden layer: 16 $8 \times 8$ filters with stride 4, ReLU
- 2nd hidden layer: 32 $4 \times 4$ filters with stride 2, ReLU
- last hidden layer, fully connected, 256 ReLU
- output layer: fully connected, one output per valid action $\#A$ in 4..18
- decision: select action with max. output
Playing Atari, 2

Training

Algorithm 1 Deep Q-learning with Experience Replay

1. Initialize replay memory $D$ to capacity $N$
2. Initialize action-value function $Q$ with random weights
3. for episode $= 1, M$ do
4. \hspace{1em} Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
5. \hspace{1em} for $t = 1, T$ do
6. \hspace{2em} With probability $\epsilon$ select a random action $a_t$
7. \hspace{2em} otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
8. \hspace{2em} Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
9. \hspace{2em} Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10. \hspace{2em} Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
11. \hspace{2em} Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
12. \hspace{2em} Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
13. \hspace{2em} Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation
14. \hspace{1em} end for
15. \hspace{1em} end for

$$y = Q(s, a, \theta)$$

$$Q(s, a, \theta) = \mathbb{E} \left[ r(s, a) + \arg\max_{a' \in A} \{Q(s', a', \theta)\} \right]$$
Playing Atari, 3

Tricks

- Experience replay: store \( \{ (s_t, a_t, r_t, s_{t+1}) \} \)
- Inner loop, minibatch of 32 uniformly drawn samples (avoids correlated updates)
- All positive rewards = 1; negative = -1
- Select an action every 4 time frames and apply it for 4 time frames

Results

<table>
<thead>
<tr>
<th></th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>-20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>-17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>-3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>-16</td>
<td>1325</td>
<td>800</td>
<td>1145</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
</tbody>
</table>

Table 1: The upper table compares average total reward for various learning methods by running an \( \epsilon \)-greedy policy with \( \epsilon = 0.05 \) for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an \( \epsilon \)-greedy policy with \( \epsilon = 0.05 \).
Playing Atari, 4

What is impressive

- Several games
- Single architecture
- Same hyper-parameters !!!
Overview

Neural Nets
Main ingredients
Invariances and convolutional networks

Deep Learning

Deep Learning Applications
Computer vision
Natural language processing
Deep Reinforcement Learning
The importance of being deep, revisited

Take-home message
Do Deep Nets Really Need To Be Deep?

Caruana


**Principle**

- Train an ensemble of deep NN
- Use the ensemble as teacher
- Find (optimize) a shallow NN to approximate the teacher
Contents
TIMIT speech corpus: 462 speakers in the training set, 50 speakers in validation set, 24 speakers in test set.

Pre-processing
The raw waveform audio data were pre-processed using 25ms Hamming window shifting by 10ms to extract Fourier-transform-based filter-banks with 40 coefficients (plus energy) distributed on a mel-scale, together with their first and second temporal derivatives. We included +/- 7 nearby frames to formulate the final 1845 dimension input vector. The data input features were normalized by subtracting the mean and dividing by the standard deviation on each dimension. All 61 phoneme labels are represented in tri-state, i.e., three states for each of the 61 phonemes, yielding target label vectors with 183 dimensions for training. At decoding time these are mapped to 39 classes as in [13] for scoring.
Results

NNs

- DNN, three fully connected feedforward hidden layers (2000 rectified linear units per layer).
- CNN convolutional architecture
- Shallow neural nets with 8000, 50 000, and 400 000 hidden units.

Architecture of shallow NN

- A linear bottleneck followed by a non-linear layer

<table>
<thead>
<tr>
<th>Architecture</th>
<th># Param.</th>
<th># Hidden units</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNN-8k</td>
<td>8k + dropout trained on original data</td>
<td>~12M</td>
<td>~8k</td>
</tr>
<tr>
<td>SNN-50k</td>
<td>50k + dropout trained on original data</td>
<td>~100M</td>
<td>~50k</td>
</tr>
<tr>
<td>SNN-400k</td>
<td>250L-400k + dropout trained on original data</td>
<td>~180M</td>
<td>~400k</td>
</tr>
<tr>
<td>DNN</td>
<td>2k-2k-2k + dropout trained on original data</td>
<td>~12M</td>
<td>~6k</td>
</tr>
<tr>
<td>CNN</td>
<td>c-p-2k-2k-2k + dropout trained on original data</td>
<td>~13M</td>
<td>~10k</td>
</tr>
<tr>
<td>ECNN</td>
<td>ensemble of 9 CNNs</td>
<td>~125M</td>
<td>~90k</td>
</tr>
<tr>
<td>SNN-MIMIC-8k</td>
<td>250L-8k no convolution or pooling layers</td>
<td>~12M</td>
<td>~8k</td>
</tr>
<tr>
<td>SNN-MIMIC-400k</td>
<td>250L-400k no convolution or pooling layers</td>
<td>~180M</td>
<td>~400k</td>
</tr>
</tbody>
</table>
What matters is not the deep architecture, after all...

On the validation set
What matters is not the deep architecture, after all...

On the test set
Why does this work?

- Label much more informative: input \((x, p)\) with \(p\) the log probability of each class (before the softmax).
  This gives much more information than the softmax.
- Data augmentation: teacher can label anything, no extra label cost.
Neural Turing Machines
http://msrvideo.vo.msecnd.net/rmcvideos/260037/dl/260037.mp4
Morphing of representations

Leon Gatys
Morphing of representations

Leon Gatys

Used for Content

Used for Style

Decrease $\alpha/\beta$
Spatial transformer

Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.
Overview

Neural Nets
- Main ingredients
- Invariances and convolutional networks

Deep Learning

Deep Learning Applications
- Computer vision
- Natural language processing
- Deep Reinforcement Learning
- The importance of being deep, revisited

Take-home message
DNN as a representation builder

Features learned from large datasets e.g. ImageNet

- Can be useful for many other problems
- As initialization for another DNN
  - Higher layers are more specific: can be tuned on *your* data while reusing general features from lower layers (e.g. edge detectors)
- As indices for a large db see Locally Sensitive Hashing
- As a feature layer for e.g. SVMs

Faces | Cars | Elephants | Chairs

![Images of faces, cars, elephants, and chairs]
DNN as a massive computer science technology

DNN training is made possible
  ▶ With tons of data
  ▶ With specific computational platforms

The entry ticket is expensive
  ▶ See TensorFlow
DNN as a functional primitive

Huge models

<table>
<thead>
<tr>
<th>Published source</th>
<th>Application</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hinton et al., 2006</td>
<td>Digit images</td>
<td>1.6mn</td>
</tr>
<tr>
<td>Hinton &amp; Salakhutdinov</td>
<td>Face images</td>
<td>3.8mn</td>
</tr>
<tr>
<td>Salakhutdinov &amp; Hinton</td>
<td>Sem. hashing</td>
<td>2.6mn</td>
</tr>
<tr>
<td>Ranzato &amp; Szummer</td>
<td>Text</td>
<td>3mn</td>
</tr>
<tr>
<td>Using GPU (Raina et al., 2009)</td>
<td></td>
<td>100mn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training Words</th>
<th>Training Time</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert NNLM</td>
<td>50</td>
<td>660M</td>
<td>2 months</td>
<td>11</td>
</tr>
<tr>
<td>Turian NNLM</td>
<td>200</td>
<td>37M</td>
<td>few weeks</td>
<td>2</td>
</tr>
<tr>
<td>Mnih NNLM</td>
<td>100</td>
<td>37M</td>
<td>7 days</td>
<td>9</td>
</tr>
<tr>
<td>Mikolov RNNLM</td>
<td>640</td>
<td>320M</td>
<td>weeks</td>
<td>25</td>
</tr>
<tr>
<td>Huang NNLM</td>
<td>50</td>
<td>990M</td>
<td>weeks</td>
<td>13</td>
</tr>
<tr>
<td>Skip-gram (hier.s.)</td>
<td>1000</td>
<td>6B</td>
<td>hours</td>
<td>66</td>
</tr>
<tr>
<td>CBOW (negative)</td>
<td>300</td>
<td>1.5B</td>
<td>minutes</td>
<td>72</td>
</tr>
</tbody>
</table>
Next frontiers

Questions

- Interpretation
- Do we still need (relational) logic?

Next applications

- Signal processing?