Application of Deep Convolutional Neural Networks to Speech Recognition

Dr. László Tóth

University of Szeged
Department of Computer Algorithms and Artificial Intelligence
Neural networks in speech recognition

- Hidden Markov modelling (HMM) has been the standard speech recognition technology since the early 80’s.
- But from the late 80’s attempts have been made to use ANNs.
- The most successful was the HMM/ANN hybrid model:
  - The ANN is responsible for the local labeling (probability estimates).
  - The utterance level combination/search is performed by the HMM.
  - Slightly better results than with HMMs, but no breakthrough.
- Deep neural networks use the same HMM/ANN scheme:
  - The breakthrough is from using DNNs instead of ANNs.
  - Although there are approaches to replace the HMM part as well by neural models (end-to-end speech recognition), these are worse yet.
The deep learning revolution

- 2006: The first deep learning paper (Science, image data)
- 2009: First application to speech recognition
  - Immediately a new record on the TIMIT dataset
  - It still holds that the new deep learning ideas are first tried on image data
- 2011: Google and Microsoft also applies deep learning
  - They report an error rate decrease of 10-30% in their products
- 2015-16: They already talk about „superhuman performance”
  - „Achieving Human Parity in Conversational Speech Recognition” (Microsoft, 2016)
  - “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification” (Microsoft, 2015)
The effect of deep learning on ASR
(Slide by Li Deng, ICASSP 2016 keynote talk)
The Sources of this Success

- New learning algorithms and activation functions
  - E.g. RBM-pretraining, ReLU activation
- The availability of enormous data sets
  - The advantages of deep learning do not show up on small data
- The availability or fast hardware
  - The invention of GPUs made deep learning accessible to everyone
- Many of the algorithmic ideas (like the convolutional model) were present decades ago
  - But the lack of hardware and large training datasets did not allow to convincingly prove the advantages of deep models
We applied only 1-2 hidden layers

- Trained by error backpropagation (gradient-based optimization)

Why only 1-2 hidden layers?

- theoretically, it can achieve arbitrary accuracy by increasing the amount of hidden neurons
- training was already slow
Deep networks

• **Deep network: many (>2) hidden layers**
  - With a *given* number of neurons, arranging them into many layers is more efficient than using only one big hidden layer!

• **The problem of speed**
  - Solved by the invention of GPUs (20-40 times speedup!)

• **The problem of training: backpropagation is not efficient for many hidden layers**
  - Hinton invented the RBM pre-training algorithm in 2006
  - Many new refinements since then (ReLU activation, new initialization schemes, batch normalization…)
  - Current wisdom: pre-training is not necessary if you have enough training data
Rectifier Neural Nets (Glorot, 2011)

- We modify the activations function of the neurons
  - Replace sigmoid (or tanh) with the max(0,x) function (ReLU)
- These neurons seem to be more suitable for building deep nets
  - The activation does not saturate → no “vanishing gradient” effect
  - Weight normalization is required to prevent the weights from “blowing up”
Results on TIMIT

**DBN**: DBN pre-training

**DPT**: discriminative pre-training

**RECT**: rectifier activation (no pre-training)
Results on Broadcast News Data

Comment: HMM: 19.9%
A Comparison of Training Times (Broadcast news dataset)

<table>
<thead>
<tr>
<th>Method</th>
<th>Pre-training</th>
<th>Backpropagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBN pretraining</td>
<td>48 hours</td>
<td>14 hours</td>
</tr>
<tr>
<td>DPT pretraining</td>
<td>9 hours</td>
<td>11 hours</td>
</tr>
<tr>
<td>ReLU neurons</td>
<td>0 hours</td>
<td>14.5 hours</td>
</tr>
</tbody>
</table>

- We now almost exclusively use only rectifier networks
- „the new de-facto standard of deep learning” (Sonoda & Murata)
- „the most popular non-linear function is the rectified linear unit (ReLU)” (Hinton et al., Nature 2015)
Yet another method taken from image processing
• The basic concept was already present in the 80’s, 90’s!
• But it’s only now that we can efficiently train deep structures
  (A convolutional network is necessarily deep)
• CNNs have a special network structure
  • It assumes that the input builds up hierarchically
  • Lower levels: extraction of local, but detailed information
  • Higher levels: detection of wide-spreading abstract structures
• It was first applied to speech recognition only in 2012
  • (with the exception of TDNN, 1989!)
  • it can be combined with all the previously mentioned techniques
An Example: Face Recognition

- A face is built up hierarchically: lines, circles,… → nose, eye,… → face
- The 3 main features of convolutional processing:
  - Locality: each neuron processes only a small part of the picture
  - Weight sharing: the same neuron is evaluated at several positions
  - Pooling: the resulting values are pooled (e.g. taking the max)
- Example: a “nose detector”
- Of course, there may be further, convolutional of fully connected layers
- Main advantages:
  - Hierarchical processing
  - Shift-invariance
Convolutional Neural Networks

- Image recognition may require a lot of layers
  - Lowest levels: local, high resolution details
  - Higher levels: covering wider and wider areas with lower and lower resolution, detecting more and more abstract components
How to Apply CNNs to Speech?

- **HMMs:** the conventional input is the MFCC representation
  - A short-term spectral representation plus a DCT to decorrelate the features
  - The time context is not taken into consideration (only by the “delta” vectors)

- **DNNs:**
  - DNNs do not require the decorrelation of features
  - They can efficiently make use of a wider context (9-33 neighboring frames)

- From MFCCs we returned to a spectro-temporal input representation
  - $f$: 23-40 mel bands
  - $t$: 9-33 frames

- This is an image, so we can apply CNNs!
Convolution along the frequency axis

- Basic idea: Abdel-Hamid (2012), Sainath (2013)
- How to exploit the shift invariance of CNNs?
- The frequency axis is divided into wider bands (the optimum was at 7 bands)
- We allow small shifts along the freq. axis
- The output of the convolutional layers is processed by further fully connected layers
- Why convolution (shift invariance) helps: decreases the speaker and speaking-style variance (e.g. tolerates small differences in the formant positions)
Results (TIMIT)

Phone error rate as a function of the pooling size

- The optimal size for „pooling” (shifting) is 4-5 mel-channels
- Convolution reduces the error rate by about 9% relative
- There is error reduction already at pooling size = 1
  (there is no pooling, just a local processing of spectral parts!)
Convolution along the time axis

- Basic idea: Vesely (2011)
- We divide the input along the time axis
- Why convolution helps:
  - Allowing shifts is not important (the HMM handles time shifts)
  - It allows the hierarchical processing of a wider input with fewer neurons
- Very similar to the Time-Delay Neural Network of Waibel et al from 1989!
- As by “convolutional” people mean convolution by frequency, I prefer calling it the hierarchical model
- The Kaldi implementation of TDNN consists of several such layers
## Results (TIMIT)

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully connected ReLU network</td>
<td>20.6%</td>
</tr>
<tr>
<td>Convolutional network (along time)</td>
<td>18.6%</td>
</tr>
</tbody>
</table>

- Convolution along time brings an error rate reduction of about 9%
Convolution along both axes

- The two concepts of convolution (time domain – Vesely vs. freq. domain – Abdel Hamid) are totally different, but can be easily combined
  - and this combination results in a significant reduction of the recognition error rates.
Convolution along both axes

- The input is divided along both time and frequency
- The lowest layer perform the convolution along frequency
- A higher layer performs the fusion along time
- The are several further, fully connected layers
Error rates (TIMIT)

<table>
<thead>
<tr>
<th>Convolution method</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution only along the freq. axis</td>
<td>18.8%</td>
</tr>
<tr>
<td>Convolution only along the time axis</td>
<td>18.6%</td>
</tr>
<tr>
<td>Convolution along both axes</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

• Compared to the previous best result, by combining the two convolution methods we obtained a further error rate reduction of 6% relative
The Maxout Activation Function

- “Maxout” can be interpreted as a generalization of the rectifier activation function
- The neurons are divided into groups (e.g., 2 neurons/groups)
- There is one output per group, defined as the maximum of the linear activations within the group
The convolutional step can be easily combined with the maxout activation:

- **Convolution**: The „pooling” step fuses the outputs of the same neuron obtained at different positions.
- **Maxout**: The „pooling” step fuses the outputs of different neurons on the same input.
- The two pooling operations can be executed in one step.
Other authors have found that Maxout outperforms ReLU most importantly in low-resource conditions (<30h) (Miao et al, 2013)

Since then, newer variants of the ReLU activation have been proposed, but these are not convincingly better.

Currently, ReLU is the most popular activation function for DNNs

<table>
<thead>
<tr>
<th>Network type</th>
<th>PhER (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv. ReLU</td>
<td>17.6%</td>
</tr>
<tr>
<td>Conv. Maxout</td>
<td>17.0%</td>
</tr>
</tbody>
</table>
The „Dropout” Method (Hinton et al, 2012)

- During training, a group of randomly selected neurons (10-50%) are discarded (their output is replaced by zeros)
  - Effect: the neurons within the same layer are forced to rely less on each other
  - Result: decreases the risk of overfitting
  - It can be combined with all the previous network types
  - The only drawback is that training takes 3-5 time longer

- Results (TIMIT):

<table>
<thead>
<tr>
<th>Network type</th>
<th>PhER (%)</th>
<th>PhER with dropout (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv. ReLU</td>
<td>17,6%</td>
<td>16,7%</td>
</tr>
<tr>
<td>Conv. Maxout</td>
<td>17,0%</td>
<td>16,5%</td>
</tr>
</tbody>
</table>
Current trends #1: Recognition from Raw Waveforms

• Goal: get rid of hand-crafted features (MFCC, PLP, …)
  • They might be suboptimal
  • They require expert knowledge

• Long-term Goal: “End-to-end” speech recognition
  • No separate modules, just one big network
  • Input: raw sound file, output: text

• It would be a very big theoretical achievement
  • However, a lot of experts are very skeptical if it’s possible
Recognition from Raw Waveforms

- The first step of current feature extraction methods is to process the signal by a filter bank (e.g. mel-filters)
  - The operation of a filter is very similar to the operation of a neuron
  - This gives the idea to learn the filter parameters by a special convolutional network structure

[PDF] Acoustic modeling with deep neural networks using raw time signal for LVCSR.
Z Tüske, P Golik, R Schlüter, H Ney - INTERSPEECH, 2014 - 193.6.4.39

[PDF] Convolutional neural networks for acoustic modeling of raw time signal in lvcsr
P Golik, Z Tüske, R Schlüter... - ... Speech ...., 2015 - www-i6.informatik.rwth-aachen.de

[PDF] Analysis of cnn-based speech recognition system using raw speech as input
D Palaz, R Collobert - Proceedings of Interspeech, 2015 - infoscience.epfl.ch

[PDF] Learning the speech front-end with raw waveform cldnns
TN Sainath, RJ Weiss, A Senior, KW Wilson... - Proc. ..., 2015 - ee.columbia.edu
Learning Filter Banks - Results

- Tüske et al (Interspeech 2014):

- Google (Interspeech 2015):

<table>
<thead>
<tr>
<th>Features</th>
<th>model</th>
<th># hidden layers</th>
<th>dev</th>
<th>eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFCC</td>
<td>GMM</td>
<td>-</td>
<td>24.4</td>
<td>31.6</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>9</td>
<td>16.9</td>
<td>22.1</td>
</tr>
<tr>
<td>time signal</td>
<td>DNN</td>
<td>9</td>
<td>20.7</td>
<td>26.3</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>12</td>
<td>20.3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filter Size (N (ms))</th>
<th>Window Size (M (ms))</th>
<th>Init</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 (25ms)</td>
<td>400 (25ms)</td>
<td>random</td>
<td>19.9</td>
</tr>
<tr>
<td>400 (25ms)</td>
<td>560 (35ms)</td>
<td>random</td>
<td>16.4</td>
</tr>
<tr>
<td>400 (25ms)</td>
<td>560 (35ms)</td>
<td>gammatone</td>
<td>16.2</td>
</tr>
<tr>
<td>400 (25ms)</td>
<td>560 (35ms)</td>
<td>gammatone untrained</td>
<td>16.4</td>
</tr>
</tbody>
</table>
Current Trends #2: Very Deep CNNs

- **DNNs:**
  - We did not obtain significant improvements above 5-6 layers
  - Most people in the literature do not go beyond 5-9 layers
- **CNNs:**
  - We applied only 1+1 convolutional layer (along freq+time)
  - “Early” literature: 2 conv. layers is slightly better than 1, no further improvements with 3 (Sainath et al., 2013)
  - But nowadays, in image processing, people experiment with CNNs of 50-150 layers!
- The training of these very deep networks require special solutions
Current Trends #2: Very Deep CNNs

- Methods to efficiently backpropagate the error to lower layers:
  - „linear pass-through” or „jump” connections
  - „Highway networks”
  - „Linearly augmented” layers
  - „Residual network”

- These all operate quite similarly:
  - They introduce direct connections between layers farther away
  - This allows the direct propagation of the error to deeper layers, thus alleviates the “vanishing gradient” problem
Example #1: Linearly Augmented Layers

- One layer of a normal net (V is an extra linear transformation):
  \[ f_i(x) = V_i \phi(U_i x + b_i) \]
- In comparison, a linearly augmented layer:
  \[ f_i(x) = V_i \phi(U_i x + b_i) + T_i x \]
  - T is not necessarily a full transformation matrix, good results were obtained using a diagonal matrix, or a fixed (non-trainable) unity matrix

<table>
<thead>
<tr>
<th>Num of H.Layers</th>
<th>Layers Size</th>
<th># Params</th>
<th>PER %</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1024X256</td>
<td>2.9M</td>
<td>22.39</td>
</tr>
<tr>
<td>12</td>
<td>512X256</td>
<td>3.8M</td>
<td>21.8</td>
</tr>
<tr>
<td>48</td>
<td>256X128</td>
<td>3.4M</td>
<td>21.7</td>
</tr>
</tbody>
</table>

(Results on TIMIT, taken from Droppo et al, ICASSP 2016)
Example #2: Highway Networks

- The networks has „highway” connections which allow the information to flow without transformation
  - Layer of a conventional net: \( y = H(x, W_H) \)
  - Layer of a highway net: \( y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T)) \)

- \( T \) is a „transfer gate”: \( T(x) = \sigma(W_T^T x + b_T) \)
- \( T \) can take values between 0 and 1
- The output in the cases of \( T=0 \) or \( T=1 \):
  \[
  y = \begin{cases} 
  x, & \text{if } T(x, W_T) = 0, \\
  H(x, W_H), & \text{if } T(x, W_T) = 1.
  \end{cases}
  \]

- That is, \( T \) controls the ratio of how much is let through from the non-transformed \( x \) and the transformed \( H(x) \)
Current trends #3: Combination with Recurrent Networks

• Currently, the fastest developing trend in speech recognition is the application of Recurrent Neural Networks (RNNs)
  • While images have no “directions”, speech is a left-to-right process, which is definitely a very important factor
  • Currently, very good results are achieved by RNNs, or its more refined variants (LSTM, GRU, …)
• Right now, in speech recognition RNNs seems to be more promising for future improvements than CNNs
  • But the two may be combined, eg. using convolution in the deeper layers and recurrent layers at higher levels!
Thank you for your attention!