(Deep) Neural Networks for Speech Processing

Kate Knill

September 2015

Cambridge University Engineering Department

DREAMS Summer School Tutorial 2015
Overview

• **Part 1:**
  – Motivation
  – Basics of Neural Networks
  – Voice Activity Detection
  – Automatic Speech Recognition

• **Part 2:**
  – Neural Networks for ASR Features and Acoustic Models
  – Neural Networks for Language Modelling
  – Other Neural Network Architectures
Motivation
Speech processing sequence-to-sequence mapping tasks

Speech (continuous time series) $\rightarrow$ Speech (continuous time series)
- Speech Enhancement, Voice Conversion

Speech (continuous time series) $\rightarrow$ Text (discrete symbol sequence)
- Automatic speech recognition (ASR), Voice Activity Detection (VAD)

Text (discrete symbol sequence) $\rightarrow$ Speech (continuous time series)
- Text-to-speech synthesis (TTS)

Text (discrete symbol sequence) $\rightarrow$ Text (discrete symbol sequence)
- Machine translation (MT)
Speech sequence-to-sequence mapping commonalities

- Variable length sequences

- Highly non-linear relationship

- Increasing quantities of data for training
  - Google Now, Siri, Cortana have gathered 1000s of hours of audio
  - A lot of the data is untranscribed or only has approximate labels

- Increasing diversity in the data
  - broader range of speakers - accents, first language
  - broader range of environmental noises

- Lots of room for improvement still!

Deep Neural Networks are very much part of the solution (cause?)
(Deep) Neural Networks

- Neural networks have increasingly been applied in speech since 2009
  - initially applied to speech recognition [1, 2, 3, 4]
  - “Neural Networks” in title of 8% INTERSPEECH 2015 sessions: feature extraction, modelling, speaker recognition, speech synthesis etc

- But we’ve been here before haven’t we?
  - alternative to GMM-HMMs for ASR in 1980s/early 90s e.g. [5, 6, 7, 8, 9, 10, 11]
    - ✓ smaller footprint than GMM-HMM-based systems
    - × did not perform as well - limited context modelling, adaptation

- What’s changed?
  - Significant increase in computing power: CPU and GPU
  - Big data
  → More powerful networks:
    - more layers (deep) and finer targets (wide)
Success of neural networks in ASR and TTS

- Speech recognition
  - Systems from Google and IBM reported in [12]

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours of data</th>
<th>HMM-DNN</th>
<th>HMM-GMM w/ same data</th>
<th>HMM-GMM w/ more data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice Input</td>
<td>5,870</td>
<td>12.3</td>
<td>N/A</td>
<td>16.0</td>
</tr>
<tr>
<td>YouTube</td>
<td>1,400</td>
<td>47.6</td>
<td>52.3</td>
<td>N/A</td>
</tr>
<tr>
<td>Switchboard</td>
<td>300</td>
<td>12.4</td>
<td>14.5</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Current best: Switchboard 10.4% using joint CNN/DNN and iVector features [13]

- Parametric speech synthesis [14]
  - Speech samples kindly provided by Heiga Zen, Google
Basics of Neural Networks
Where it started

- Early work by MuCulloch and Pitts [15]
- The Perceptron (Rosenblatt) [16] (early 1960s)
- Mostly halted by publication of “Perceptrons” by Minsky and Papert 1969 [17]
- Error back propagation training for multi-layer perceptrons mid 80s [18]
• Aim: map an input vector $\mathbf{x}$ into an output vector $\mathbf{y}$
  - Non-linear units “neurons” combined into one or more layers
  - **Intuition**: each layer produces a higher level feature representation and better classifier than its input
  - Combine simple building blocks to design more complex, non-linear systems
Hidden Layer Neuron

- Linearly weighted input is passed to a general activation function

- Assume \( n \) units at previous level \((k-1)\): \( x_j^{(k)} = y_j(x^{(k-1)}) \)

\[
y_i(x^{(k)}) = \phi(z_i) = \phi(w_i^T x^{(k)} + b_i) = \phi(\sum_{j=1}^{n} w_{ij} x_j^{(k)} + b_i)
\]

where \( \phi() \) is the activation function

- Note: activation function could be linear BUT then linear net i.e. lose power!
Traditional Activation Functions

- **Sigmoid** (or logistic regression) function:
  \[
  y_i(x) = \frac{1}{1 + \exp(-z_i)}
  \]

  Continuous output, \(0 \leq y_i(x) \leq 1\)

- **Softmax** (or normalised exponential or generalised logistic) function:
  \[
  y_i(x) = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}
  \]

  Positive output, sum of all outputs at current level is 1, \(0 \leq y_i(x) \leq 1\)

- **Hyperbolic tan** (tanh) function:
  \[
  y_i(x) = \frac{\exp(z_i) - \exp(-z_i)}{\exp(z_i) + \exp(-z_i)}
  \]

  Continuous output, \(-1 \leq y_i(x) \leq 1\)
Activation functions

- **step** activation function (green)
- **sigmoid** activation function (red)
- **tanh** activation function (blue)

Sigmoid or softmax often used at output layers as sum-to-one constraint enforced
**Possible Decision Boundaries**

- Nature of decision boundaries produced varies with network topology
- Using a threshold (step) activation function:

1. **Single layer**: position a hyperplane in the input space (SLP)
2. **Two layers**: surround a single convex region of input space
3. **Three layers**: generate arbitrary decision boundaries

- **Sigmoid**: arbitrary boundaries with two layers if enough hidden units
Number of Units per Layer

How many units to have in each layer?

- Number of output units = number of output classes

- Number of input units = number of input dimensions

- Number of hidden units - design issue
  - too few - network will not model complex decision boundaries
  - too many - network will have poor generalisation
Training Criteria (1)

Variety of training criteria may be used.

- Assume we have supervised training examples

\[
\{\{x_1, t_1\} \ldots, \{x_n, t_n\}\}
\]

- Compare outputs \(y\) with correct answer \(t\) to get error signal

- **Least squares error**: one of the most common training criteria

\[
E = \frac{1}{2} \sum_{p=1}^{n} ||y(x_p) - t_p||^2
\]

\[
= \frac{1}{2} \sum_{p=1}^{n} \sum_{i=1}^{K} (y_i(x_p) - t_{pi})^2
\]
Training Criteria (2)

- **Cross-Entropy for two classes**: consider case when \( t \) is binary (softmax output)

\[
E = - \sum_{p=1}^{n} \left( t_p \log(y(x_p)) + (1 - t_p) \log(1 - y(x_p)) \right)
\]

Goes to zero with the “perfect” mapping

- **Cross-Entropy for multiple classes**:

\[
E = - \sum_{p=1}^{n} \sum_{i=1}^{K} t_{pi} \log(y_i(x_p))
\]

- minimum value is non-zero
- represents the entropy of the target values
Single Layer Perceptron Training (1)

- Consider single layer perceptron initially

\[ y(x) = \sum w_d x_d + b \]

- Minimise (for e.g.) square error between target \( t_p \) and current output \( y(x_p) \)

- Least squares criterion with sigmoid activation function

\[
E = \frac{1}{2} \sum_{p=1}^{n} (y(x_p) - t_p)^T (y(x_p) - t_p) = \sum_{p=1}^{n} E^{(p)}
\]

- Simplify notation: single observation \( x \), target \( t \), current output \( y(x) \)
Single Layer Perceptron Training (2)

- How does the error change as \( y(x) \) changes?
  \[
  \frac{\partial E}{\partial y(x)} = y(x) - t
  \]

BUT we want to find the effect of varying the weights

- Calculate effect of changing \( z \) on the error using the chain rule
  \[
  \frac{\partial E}{\partial z} = \left( \frac{\partial E}{\partial y(x)} \right) \left( \frac{\partial y(x)}{\partial z} \right)
  \]

- What we really want is the change of the error with respect to the weights
  - the parameters that we want to learn
    \[
    \frac{\partial E}{\partial w_i} = \left( \frac{\partial E}{\partial z} \right) \left( \frac{\partial z}{\partial w_i} \right)
    \]
Single Layer Perceptron Training (3)

- The error function therefore depends on the weight as
  \[ \frac{\partial E}{\partial w_i} = \left( \frac{\partial E}{\partial y(x)} \right) \left( \frac{\partial y(x)}{\partial z} \right) \left( \frac{\partial z}{\partial w_i} \right) \]

- Noting that (the bias term \( b \) can be treated as the \( d + 1 \) element)
  \[ \frac{\partial y(x)}{\partial z} = y(x)(1 - y(x)) \]

  \[ \frac{\partial E}{\partial w_i} = (y(x) - t) y(x)(1 - y(x)) x_i \]

- In terms of the complete training set
  \[ \nabla E = \sum_{p=1}^{n} (y(x_p) - t_p) y(x_p)(1 - y(x_p)) \tilde{x}_p \]

- So for single layer can use gradient descent to find the “best” weight values
Single Layer Perceptron Training - Review

\[
\frac{\partial E}{\partial w_i} = \left( \frac{\partial E}{\partial y(x)} \right) \left( \frac{\partial y(x)}{\partial z} \right) \left( \frac{\partial z}{\partial w_i} \right)
\]
Error Back Propagation Algorithm

- Training Goal: minimise the cost between predicted output and target values
- Error back propagation [18] is an effective way to achieve this

- Use Gradient Descent to optimise the weight values
  - i.e. activation function must be differentiable
Training schemes

Modes

- **Batch** - update weights after all training examples seen

- **Sequential** - update weights after every sample
  
  Advantages:
  
  - Don’t need to store the whole training database
  - Can be used for online learning
  - In dynamic systems weight updates “track” the system

- **Mini-batch** - update weights after a subset of examples seen
  
  Practical compromise:
  
  - Estimate based on more data than sequential
  - Avoids expensive batch computation if poor current weight values
Voice Activity Detection
Voice Activity Detection

- Detect periods of human speech in an audio signal
Samples from MGB Challenge 2015 [19]
Voice Activity Detection

- Detect periods of human speech in an audio signal

- Sequence classification task
  - 2-class problem: speech or non-speech

- Standard approaches:
  - Unsupervised - threshold against a value e.g. energy, zero-crossing rate
  - Supervised - train a classifier with features such as MFCCs or PLPs, e.g. Gaussian mixture models (GMMs), support vector machines
DNNs for Speech Processing

VAD stages

1. Feature extraction
   - compact representation of signal
   - “uncorrelated” to allow diagonal covariance Gaussians

2. Decision making
   - probability of being speech/non-speech computed each frame

3. Hangover
   - smooth decisions
   - 2-state HMM in Viterbi decoding
Gaussian Mixture Models

- Gaussian mixture models (GMMs) are based on (multivariate) Gaussians
  - form of the Gaussian distribution:
    \[
    p(x) = \mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)
    \]

- For GMM each component modelled using a Gaussian distribution
  \[
  p(x) = \sum_{m=1}^{M} P(c_m) p(x|c_m) = \sum_{m=1}^{M} P(c_m) \mathcal{N}(x; \mu_m, \Sigma_m)
  \]
  - component prior: \( P(c_m) \)
  - component distribution: \( p(x|c_m) = \mathcal{N}(x; \mu_m, \Sigma_m) \)

- Highly flexible model, able to model wide-range of distributions
GMM-HMM based VAD

✓ Work well under stationary noise conditions

× Do not generalise to diverse domains e.g. meetings, YouTube

Source: Robust Speaker Diarization for Meetings, X. Anguera, Phd Thesis
DNN based VAD

- Replace GMM probability density function in HMM with DNN output [20]
  - First must convert output posteriors to likelihoods

\[
p(x_t | \text{spch}) = \frac{P(\text{spch} | x_t) p(x_t)}{P(\text{spch})}
\]

✓ Significantly more accurate in challenging environments
  e.g. 20% frame-wise error rate on YouTube vs 40% GMM system [21]
DNNs for Speech Processing

DNN-based VAD - training considerations

• Input features
  – Can use same MFCC or PLP features as for GMM
  – Gains shown when extending context [21]
  – Filterbanks show further gains [22]

• Targets
  – Each training frame is tagged as speech/non-speech
  – Following DNN training, data can be realigned including unlabelled data

• Example system: Cambridge University MGB Challenge 2015 VAD [22]
  – Input: 40-d filterbanks, 55 frames (±27)
  – Layers: $1000 \times 200^5 \times 2$
  – Activation functions: sigmoid
  – Targets: alignments derived from lightly supervised recognition
  – Training criterion: frame-based cross-entropy (CE)
Language Identification
Automatic Speech Recognition
Speech Production

- **Excitation source**
  - vocal cords vibrate producing quasi-periodic sounds (voiced sounds)
  - turbulence caused by forcing air through a constriction in the vocal tract (fricative sounds)

- **Acoustic tube**
  - articulators move: alter the shape of the vocal tract enable/disable nasal cavity
  - co-articulation effect.

- **Speech**
  - sound pressure wave.
Automatic Speech Recognition - Theory

- Speech recognition based on Bayes’ Decision Rule

\[ \hat{w} = \max_w \{ P(w|O) \} \]

\[ O = \{ x_1, \ldots, x_T \} \text{ and } w = \{ w_1, \ldots, w_L \} \]

- Two forms of classifier used:
  - **Generative model**: model joint distribution \( p(O, w) \)

\[ P(w|O) = \frac{p(O, w)}{p(O)} \propto p(O|w)P(w) \]

  - **Discriminative model**: directly model posterior distribution \( P(w|O) \)

Machine Learning underpins all ASR systems
Automatic Speech Recognition - Modules

- **Front-end** processing: transforms waveform into *acoustic vectors*
- **Acoustic** model: probability of observations given a word sequence
- **Lexicon**: maps from word to phone sequence
- **Language** model: computes the prior probability of any word sequence

Statistical approaches used to combine information sources
Front End Processing

Signal

Windowing

FFT

Mel Filterbank

DCT

Cambridge University Engineering Department
Acoustic Modelling

(a) Speech Production

(b) HMM Generative Model

- Not modelling the human production process!
Hidden Markov Model “Production”

- State evolution process
  - discrete state transition after each “observation”
  - probability of entering a state only dependent on the previous state

- Observation process
  - associated with each state is a probability distribution
  - observations are assumed independent given the current state

- Speech representation
  - feature vector every 10ms
Hidden Markov Model

- The likelihood of the data is

\[ p(x_1, \ldots, x_T) = \sum_{q \in Q_T} P(q)p(x_1, \ldots, x_T|q) = \sum_{q \in Q_T} P(q_0) \prod_{t=1}^{T} P(q_t|q_{t-1})p(x_t|q_t) \]

where \( q = \{q_0, \ldots, q_{T+1}\} \) and \( Q_T \) is all possible state sequences for \( T \) observations.

- Poor model of the speech process - piecewise constant state-space.
HMM Acoustic Units

John /jh/ /aa/ /n/ hit /hh/ /ih/ /t/ the /dh/ /ax/ ball /b/ /ao/ /l/
State Tying - Decision Tree

- Phone /ih/
- Left Nasal
  - Right Liquid
    - Right /l/
      - Yes: Model A
      - No: Model B
    - Left Fricative
      - Yes: Model C
      - No: Model D
  - No: Model E
State Output Distribution: Gaussian Mixture Model

A common form of distribution associated with each state:

- the Gaussian mixture model (or mixture of Gaussians).

- linear combination of components

\[
p(x_t) = \sum_{m=1}^{M} c^{(m)} N(x_t, \mu^{(m)}, \Sigma^{(m)})
\]

- Good modelling power:
  - implicitly models variability
- No constraints on component choice
HMM Training using EM

• Need to train HMM model parameters, $\lambda$, on 100s of millions of frames
  – transition probabilities
  – state output distribution

• Standard training criterion for generative models: Maximum Likelihood

$$\mathcal{F}_{m1}(\lambda) = \frac{1}{R} \sum_{r=1}^{R} \log(p(O^{(r)}|w_{\text{ref}}^{(r)}; \lambda))$$

  – yields most likely model parameters to generate training data!

• Challenging to handle vast amounts of data
  – Expectation Maximisation (EM) offers a solution
HMM Training using EM

- EM an iterative scheme involving two stages:
  - **Expectation**: accumulate statistics given current model parameters
  - **Maximisation**: estimate new model parameters

- Update formulae for GMM state output distributions

\[
\mu_{j}^{[l+1]} = \frac{\sum_{t=1}^{T} \gamma_{j}^{[l]}(t)x_t}{\sum_{t=1}^{T} \gamma_{j}^{[l]}(t)}
\]

\[
\Sigma_{j}^{[l+1]} = \frac{\sum_{t=1}^{T} \gamma_{j}^{[l]}(t)x_t x_t^T}{\sum_{t=1}^{T} \gamma_{j}^{[l]}(t)} - \mu_{j}^{[l+1]} \mu_{j}^{[l+1]T}
\]

where

\[
\gamma_{j}^{[l]}(t) = P(q_t = s_j|x_1, \ldots, x_T, \lambda^{[l]})
\]
Advantages of EM training

- EM is one of the reasons GMM-HMM systems dominated for many years
  - guaranteed not to decrease log-likelihood at each iteration
  - expectation stage can be parallelised

- Parallelising the expectation stage crucial
  - Enables handling of vast quantities of data
  - Can distribute across many cheap machines

- Would like ASR system to run in real-time
  - HMM structure enables this - Viterbi algorithm
Language Model

\[
P(\text{John hit the ball}) = P(\text{John}) \times P(\text{hit} \mid \text{John}) \times P(\text{the} \mid \text{John hit}) \times P(\text{ball} \mid \text{hit the})
\]

(e) Syntactic Parse Tree

(f) Trigram Model

- Syntactic/semantic information important
  - but hard to model robustly (especially for conversational style speech)
- Simple n-gram model-used: \( P(w_1w_2...w_n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-2}w_{i-1}) \)
  - don’t care about structure - just the probability - discuss later
Automatic Speech Recognition - Modules

Speech

Frontend Processing

Recognition Algorithm

Language Model

Lexicon

Acoustic Models

Recognised Hypothesis
Recognition Algorithm - Viterbi

- An important technique for HMMs (and other models) is the Viterbi Algorithm
  - here the likelihood is approximated as (ignoring dependence on class $\omega$)

\[
p(x_1, \ldots, x_T) = \sum_{q \in Q_T} p(x_1, \ldots, x_T, q) \approx p(x_1, \ldots, x_T, \hat{q})
\]

where

\[
\hat{q} = \{\hat{q}_0, \ldots, \hat{q}_{T+1}\} = \operatorname{argmax}_{q \in Q_T} \{p(x_1, \ldots, x_T, q)\}
\]

- This yields:
  - an approximate likelihood (lower bound) for the model
  - the best state-sequence through the discrete-state space
Viterbi Algorithm

- Need an efficient approach to obtaining the best state-sequence, \( \hat{q} \),
  - simply searching through all possible state-sequences impractical ...

- Consider generating the observation sequence \( x_1, \ldots, x_7 \)
  - HMM topology - 3 emitting states with strict left-to-right topology (left)
  - representation of all possible state sequences on the right
Best Partial Path to a State/Time

- Red possible partial paths
- Green state of interest

- Require best partial path to state $s_4$ at time 5 (with associated cost $\phi_4(5)$)
  - cost of moving from state $s_3$ and generating observation $x_5$: $\log(a_{34}b_4(x_5))$
  - cost of staying in state $s_4$ and generating observation $x_5$: $\log(a_{44}b_4(x_5))$
- Select “best: $\phi_4(5) = \max \{ \phi_3(4) + \log(a_{34}b_4(x_5)), \phi_4(4) + \log(a_{44}b_4(x_5)) \}$
Viterbi Algorithm for HMMs

- The Viterbi algorithm for HMMs can then be expressed as:
  
  - **Initialisation:** \((L\text{ZERO} = \log(0))\)
    \[
    \phi_1(0) = 0.0, \quad \phi_j(0) = L\text{ZERO}, \ 1 < j < N, \\
    \phi_1(t) = L\text{ZERO}, \ 1 \leq t \leq T
    \]
  
  - **Recursion:**
    \[
    \text{for } t = 1, \ldots, T \\
    \text{for } j = 2, \ldots, N - 1 \\
    \phi_j(t) = \max_{1 \leq k < N} \{ \phi_k(t - 1) + \log(a_{kj}) \} + \log(b_j(x_t))
    \]
  
  - **Termination:**
    \[
    \log(p(x_1, \ldots, x_T, \hat{q})) = \max_{1 < k < N} \{ \phi_k(T) + \log(a_{kN}) \}
    \]

- Can also store the best previous state to allow best sequence \(\hat{q}\) to be found.
Discriminative Training Criteria

- Bayes’ decision rule yields the minimum probability of error if:
  - infinite training data
  - models have the correct form
  - appropriate training criterion

  None of these are true for ASR!

- Motivates other discriminative criteria
  - use discriminative criteria to train generative models
  - ML people not that happy with use and term!

- Fortunately schemes related to EM can still be used
  - large scale discriminative training common for ASR
  - acoustic model still an HMM - Viterbi still possible
Simple MMIE Example

- HMMs are not the correct model - discriminative criteria a possibility

- Discriminative criteria a function of posteriors $P(w|O; \lambda)$
  - **NOTE**: same generative model, and conditional independence assumptions
Discriminative Training Criteria

- Discriminative training criteria commonly used to train HMMs for ASR
  - **Maximum Mutual Information** (MMI) [23, 24]: maximise
    \[
    F_{\text{mmi}}(\lambda) = \frac{1}{R} \sum_{r=1}^{R} \log(P(w_{\text{ref}}^{(r)}|O^{(r)}; \lambda))
    \]
  - **Minimum Classification Error** (MCE) [25]: minimise
    \[
    F_{\text{mce}}(\lambda) = \frac{1}{R} \sum_{r=1}^{R} \left( 1 + \left[ \frac{P(w_{\text{ref}}^{(r)}|O^{(r)}; \lambda)}{\sum_{w \neq w_{\text{ref}}^{(r)}} P(w|O^{(r)}; \lambda)} \right]^{e} \right)^{-1}
    \]
  - **Minimum Bayes' Risk** (MBR) [26, 27]: minimise
    \[
    F_{\text{mbr}}(\lambda) = \frac{1}{R} \sum_{r=1}^{R} \sum_{w} P(w|O^{(r)}; \lambda) \mathcal{L}(w, w_{\text{ref}}^{(r)})
    \]
MBR Loss Functions for ASR

- **Sentence (1/0 loss):**

\[
\mathcal{L}(w, w^{(r)}_{\text{ref}}) = \begin{cases} 
1; & w \neq w^{(r)}_{\text{ref}} \\
0; & w = w^{(r)}_{\text{ref}} 
\end{cases}
\]

When \( \phi = 1 \), \( F_{\text{mce}}(\lambda) = F_{\text{mbr}}(\lambda) \)

- **Word**: directly related to minimising the expected Word Error Rate (WER)
  - normally computed by minimising the Levenshtein edit distance.

- **Phone/State**: consider phone/state rather word loss
  - improved generalisation as more “errors” observed
  - this is known as Minimum Phone Error (MPE) training [28, 29].

- **Hamming (MPFE)**: number of erroneous frames measured at the phone level
Summary of Standard ASR Systems

• HMMs
  – efficiency of model training/decoding
  – approximate approach to modelling the signal
  – has limitations on features that can be used due to GMMs

• GMMs
  – OK but make lots of assumptions about feature vector
    - decorrelated and Gaussian
DNNs for Speech Processing

References


DNNs for Speech Processing


