TTIC 31110
Speech Technologies

May 5, 2020

Announcements

- HW3 due Friday 5/8 7pm
- Tutorial 3 (yesterday) materials available online
- Coming up: Term project
  - End of this week: Materials and guidelines available
  - Week 7: Project proposals due
  - Week 9: Project updates
  - Finals week: Final project presentations & reports
- Feel free to discuss topic ideas with me/Ankita at office hours, via email
- Seek out partners for term project (see survey results, discuss on canvas, etc.)

Questions from last week/year

We mainly discussed training HMMs for the case of discrete observations; what about continuous observations (e.g. MFCCs)?

- Discrete HMMs can be used with vector quantized features (see Tutorial 2, Lecture 6)
- In a *continuous-density* HMM, the discrete observation probabilities, $b_i(k)$, are replaced by continuous densities $b_i(\mathbf{o})$
- The observation distributions are typically Gaussian or mixture of Gaussians: $b_i(\mathbf{o}) = \sum_{k=1}^{K} c_{ik}\mathcal{N}(\mathbf{o}|\mu_{ik}, \mathbf{\Sigma}_{ik}), \;\; 1 \leq i \leq N$
- In the forward/backward/Viterbi algorithms: same algorithms, just replace $b_i(k)$ by the value of the corresponding density $b_i(\mathbf{o})$
- For EM training, the update equations look a bit different.

Recall: The Baum-Welch re-estimation formulas

M step, with multiple observation sequences $\mathbf{O}^1, \ldots, \mathbf{O}^L$

$$\hat{a}_{ij} = \frac{\displaystyle\sum_{l=1}^{L} \sum_{t=1}^{T-1} \xi_t^l(i,j)}{\displaystyle\sum_{l=1}^{L} \sum_{t=1}^{T-1} \gamma_t^l(i)}$$

$$\hat{b}_i(k) = \frac{\displaystyle\sum_{l=1}^{L} \sum_{t=1, o_t=v_k}^{T} \gamma_t^l(i)}{\displaystyle\sum_{l=1}^{L} \sum_{t=1}^{T} \gamma_t^l(i)}$$

$$\hat{\pi}_i = \frac{1}{L} \sum_{l=1}^{L} \gamma_1^l(i)$$

Baum-Welch for continuous-density HMMs

Single-Gaussian case: $b_i(\mathbf{o}) = \mathcal{N}(\mathbf{o}|\mu_i, \mathbf{\Sigma}_i), \;\; 1 \leq i \leq N$

$$\begin{aligned}
\hat{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \text{(same as for discrete HMMs!)} \\
\hat{\mu}_i &= \frac{1}{\sum_{t=1}^{T} \gamma_t(i)} \sum_{t=1}^{T} \gamma_t(i)\mathbf{o}_t \\
&= \text{(same as Gaussian mixture update for "component" } i) \\
\hat{\mathbf{\Sigma}}_i &= \frac{1}{\sum_{t=1}^{T} \gamma_t(i)} \sum_{t=1}^{T} \gamma_t(i)(\mathbf{o}_t - \hat{\mu}_i)(\mathbf{o}_t - \hat{\mu}_i)^T
\end{aligned}$$

Baum-Welch for continuous-density HMMs

Gaussian mixture case:

- This is often referred to as an HMM/GMM
- Now state and Gaussian component index are both latent variables
- Update equations now involve $\gamma_t(i,k) =$ posterior probability of being in component k in state i at time t
- See Rabiner tutorial for equations

Questions from last week/year

How does it all fit together? How do I go from a pile of data to a speech recognizer?

Meta-algorithm 1: Training a (whole-word) HMM/GMM-based speech recognizer

(1) Given:

- Training set of L utterances (acoustic features + corresponding word transcriptions)
- Hyperparameters: # states per word, # Gaussians per state, HMM “topology” (which transition probabilities are 0)
- Initial parameter values (guess)

(2) Repeat until convergence:

- E step: For each training utterance l, run forward and backward algorithms and compute the ξs
- M step: Update parameters according to the Baum-Welch equations
- Check convergence (e.g., likelihood not higher than previous iteration by some amount δ)

Meta-algorithm 2: Training and tuning a (whole-word) HMM/GMM-based speech recognizer

(1) Given:

- Training set of L utterances (acoustic features + corresponding word transcriptions)
- Development (held out/tuning) set of D utterances
- Set of allowed hyperparameters: range of # states per word, range of # Gaussians per state

(2) For each allowed combination of hyperparameters:

- Train recognizer using meta-algorithm 1
- Record performance (error rate) on dev set

(3) Choose trained recognizer with best dev-set performance

(In practice, there are more efficient ways to tune hyperparameters than the for-loop above (the above is a "grid search"))

Questions from last week/year

What about silence?

- Treat it like just another word in the vocabulary
- Often we have one “word” for utterance-initial/utterance-final silence, and one for short inter-word silences
- One twist: We usually don’t have silences marked in training data, so we allow for optionally skipping the silences

Details: Measuring performance

- Most common measure: Word error rate (WER)
- WER = (# substitutions + # deletions + # insertions)/(# wds in reference script)
- Example:

REF:	The	*	dogs	are	barking	now
HYP:	The	uh	smogs	*	barking	*
		I	S	D		D

- WER = (1+1+2)/5 = 80%
- Can be computed efficiently using dynamic programming (like DTW, Viterbi)
- Note: WER can be above 100%

Questions from last time

How are HMMs used for speech technologies besides speech recognition?

- **Unsupervised** learning, e.g. discovering sound units in a low-resource language
  - Take a pile of speech without transcriptions
  - Train a single HMM on all of it
  - Each state is a "sound unit"
  - Look for short/long repeated sequences of states to discover phones/words
- Speech synthesis
  - Much like HMMs for speech recognition, but trained on a **single** speaker's speech
  - Some care needed to ensure continuity across synthesized frames
  - Possibly replaced by neural methods

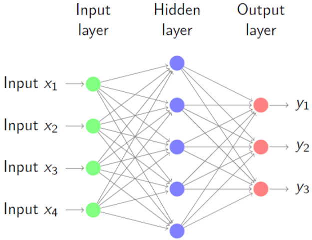
Recall: Hybrid generative/discriminative models

Typical approach:

1. Train a frame-based discriminative classifier of sub-word units (e.g. phones, phone states, triphone states) given some labeled training data, $c^* = f_c(\mathbf{o})$, where c is the class and $\mathbf{o}$ is a frame feature vector
2. The output is a posterior probability $p(c|\mathbf{o})$
3. Convert $p(c|\mathbf{o})$ to something like an observation model (a "likelihood"): $p(\mathbf{o}|c) \propto \frac{p(c|\mathbf{o})}{p(c)}$
4. Use the result in place of the observation model in an HMM
5. Most popular type of frame classifier by far: neural network

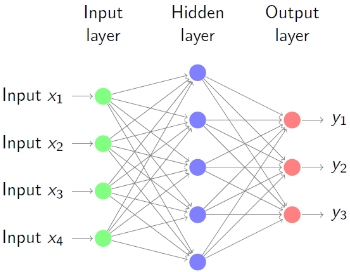
Recall: Feedforward neural networks

A feedforward neural network (NN, or DNN) is any vector function $f(\mathbf{x})$ of a vector input $\mathbf{x}$ that can be written as a composition of simple "layers"



- Each node i in each layer l outputs $y_i^l = \sigma(\mathbf{w}_i^l \cdot \mathbf{y}_i^{l-1} + b_i^l)$
- Or writing each layer's output as a vector: $\mathbf{y}^l = \sigma_l(\mathbf{W}_l \mathbf{y}^{l-1} + \mathbf{b}^l)$, where σ is applied element-wise
- (Letting $\mathbf{y}^0 = \mathbf{x}$)
- Final output: $f(\mathbf{x}) = \mathbf{y} = \mathbf{y}^L$ for an L-layer network

Recall: Multi-class outputs



Typical activation function for final layer: Softmax

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}$$

where $z_i = \mathbf{w}_i \cdot \mathbf{x} + b_i$ (Note: layer indexing dropped; $\mathbf{x}$ refers to the input of the current layer)

- Outputs are ≥ 0 and sum to 1, so can be thought of as class posterior probabilities $p(\text{class } i|\mathbf{x})$

Recap: Training neural networks

The parameters are learned to minimize some loss, or measure of badness of the outputs

- A NN is an MLP if it is trained with perceptron loss (though often "MLP" is used to refer to any feedforward NN)
- For multi-class classification (softmax output layer activation function), most common loss is cross-entropy loss
  $\ell_{CE} = -\sum_c y_c \log f_c(\mathbf{x})$, where
  $\mathbf{x}$ is input vector for one example in training set
  $y_c = 1$ if ground-truth label $= c$, 0 otherwise
  $f_c(\mathbf{x})$ is our estimate of $p(c|\mathbf{x})$
- Cross-entropy loss also called *log loss*, because
  $\ell_{CE} = -\log f_{c^*}(\mathbf{x})$ where c^* is the ground-truth label
- Total loss is the sum of the loss over all training examples

Aside: A teeny bit of information theory

- If X is a discrete random variable taking one of N values with probabilities $p_1, \ldots, p_N$, respectively, then the **entropy** of X is
$$H(X) = -\sum_{i=1}^{N} p_i \log_2 p_i$$
- This is the average number of bits needed to represent X
- If the distribution of X is uniform, then $H(X) = \log_2 N$
- A related term is **perplexity** $PP_p(X) = 2^{H(X)}$
- If the distribution of X is uniform, then what is $PP(X)$

Aside: A teeny bit of information theory

- The **cross-entropy** of a model distribution q with respect to a true distribution p is
  $$H(p,q) = -\sum_{i=1}^{N} p_i \log_2 q_i$$
- This is the average number of bits needed to represent X drawn from p using a code optimized for q
- Going back to cross-entropy loss:
  $\ell_{CE} = -\sum_c y_c \log f_c(\mathbf{x})$
- This is the cross-entropy between the true distribution y_c and our estimate of it $f_c(\mathbf{x})$
- y_c happens to be a very simple distribution: $y_c = 1$ if true label $= c$, 0 otherwise
- If we had some other ground-truth distribution ("soft" labels), could still use cross-entropy
- But then it would not be equivalent to log loss

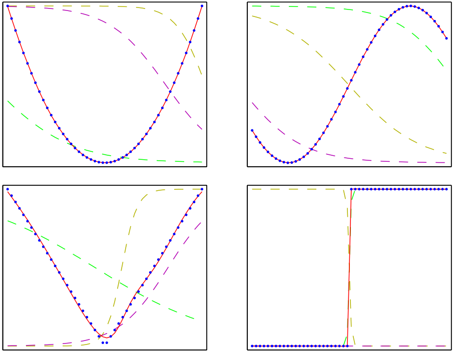
Aside: A teeny bit of information theory

Cross-entropy loss: $\ell_{CE} = -\sum_c y_c \log f_c(\mathbf{x})$

- Viewing the ground-truth label as a distribution over labels c, this is the cross-entropy between that distribution and the network's output distribution $f_c(\mathbf{x})$
- This is a measure of dissimilarity between distributions
- (For our purposes, equivalent to KL divergence)
- What is the minimum of this loss? (Note if needed: $x \log x \to 0$ as $x \to 0$

Power of two layers

- Theoretical result [Cybenko 1989]: 2-layer net with sigmoid hidden units can approximate any continuous function over compact domain to arbitrary accuracy, **given enough hidden units**
- Examples: 3 hidden units with $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ activation



[from Bishop]

Back to speech recognition...

Reminder: Hybrid ASR systems

- Use a DNN to produce a posterior for each class c (= HMM state) given an input frame of acoustic features $\mathbf{o}$
- Posterior is converted to a scaled likelihood via $p(\mathbf{o}|c) \propto \frac{p(c|\mathbf{o})}{p(c)}$

Where do class labels for all frames come from?

- Frames may have ground-truth (human) labels
- ... Or labels can be produced via a Viterbi alignment ("forced alignment") using an existing HMM/GMM system
- ... Or we can use "soft labels" = posteriors produced by running forward-backward using an existing HMM/GMM system
- (The latter makes sense if using cross-entropy loss)

Meta-algorithm 1a: Training a (whole-word) HMM/DNN-based speech recognizer

(1) Given:

- Training set of L utterances (acoustic features, corresponding word transcriptions, **state label per frame**)
- Hyperparameters: # states per word, # Gaussians per state, HMM "topology", **# DNN layers, # DNN hidden units, learning rate, regularization parameters...**
- Initial parameter values for a_{ij}, π_i **(not $b_i(\mathbf{o})$), Θ**

(2) Train DNN: Repeat until convergence

- One step of gradient descent
- Check convergence (e.g., loss not improved by at least δ_{DNN})

(3) Train HMM: Repeat until convergence

- E step: For each training utterance l, run forward and backward algorithms and compute the ξs
- M step: Update parameters according to the Baum-Welch equations, **except** $b_i(\mathbf{o})$

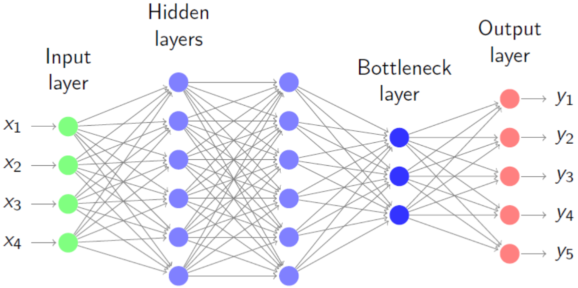
Tandem models

Never mind the whole posterior conversion business: Use the NN outputs as features!

- Then use standard HMM/GMMs with these features as inputs
- Idea developed at ICSI Berkeley (e.g., Hermansky et al. 2000)
- $\mathbf{o}' = [y_1(\mathbf{o})\ y_2(\mathbf{o})\ \ldots y_n(\mathbf{o})]$
- If the $y_i(\mathbf{o})$ represent probabilities, then we typically take their logs: $\mathbf{o}' = [\log(f_1(\mathbf{o}))\ \log(f_2(\mathbf{o}))\ \ldots]$

Tandem models (2)

Alternatively, use outputs from a lower layer, and make that layer narrow (a "bottleneck layer") to reduce dimensionality

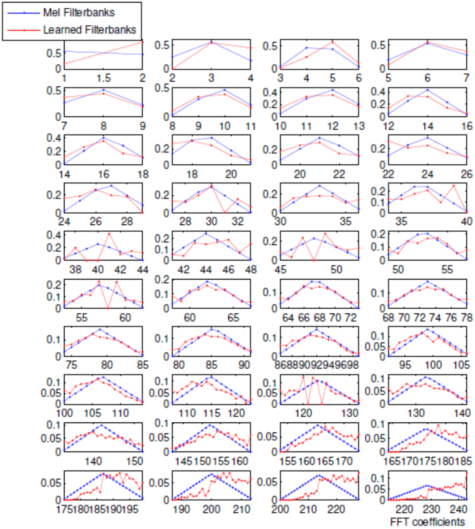


Tandem models: More tricks (3)

- These features are often appended to the original features, e.g. MFCCs, so the new feature vector is $[\mathbf{o}\ \mathbf{o}']$ (hence, "tandem"!)
- Typically, the input is a concatenation of acoustic vectors over a window of 7-20 frames around the current frame (very high-dimensional!)

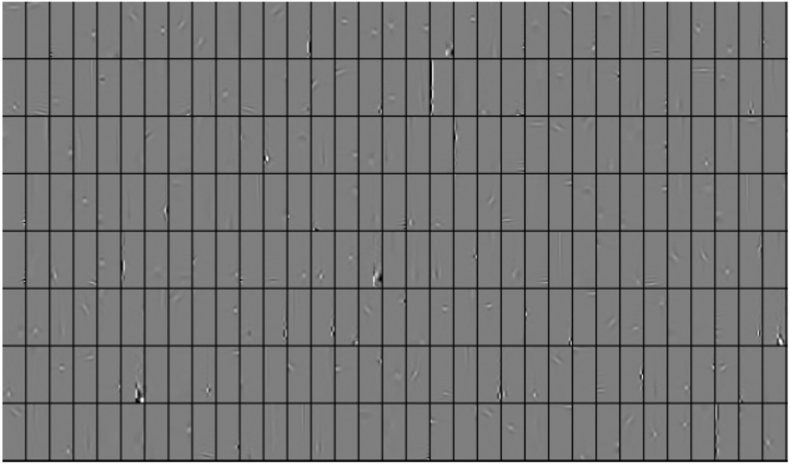
Visualizing learned acoustic features

Given raw spectrum input (Sainath *et al.* ASRU 2013):



Visualizing learned acoustic features

Given a mel-spectrogram patch as input:



Current state of hybrid and tandem models

As of 7-8 years ago:

- Depending on the task, HMM/NN models may or may not outperform HMM/GMM-based models
- Tandem models typically outperform their HMM/GMM-based counterparts

Now:

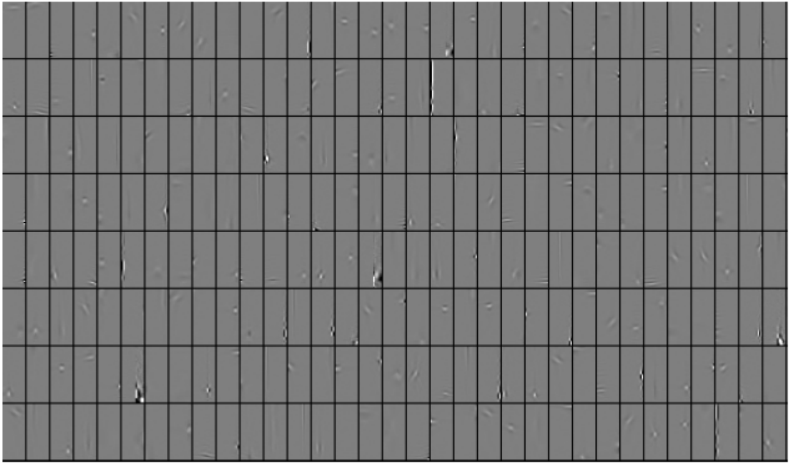
- DNNs have caused a revolution in ASR
- The state of the art is now often* hybrid HMM/NN systems, with DNN-based tandem systems somewhat behind
- Tandem models have some advantages, e.g. easier to adapt to new speakers
- *And for some domains, end-to-end neural network models are now the state-of-the-art

What changed?

- More data
- More compute (GPUs)
- $\Longrightarrow$ deeper networks
- $\Longrightarrow$ wider network outputs
- Pretraining (that's probably not important, but was useful in getting NNs into the mainstream)
- Better regularization

Recall: Visualizing learned acoustic features

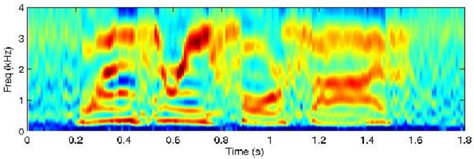
Given a mel-spectrogram patch as input:



Convolutional neural networks (CNNs)

But if we start from spectrograms, then it might help to consider:

- Many of the useful patterns are **local** in time-frequency
- Many of the useful patterns **repeat** in different time-frequency locations
- But they don't **exactly** repeat... they move around a bit between speakers, contexts, etc.

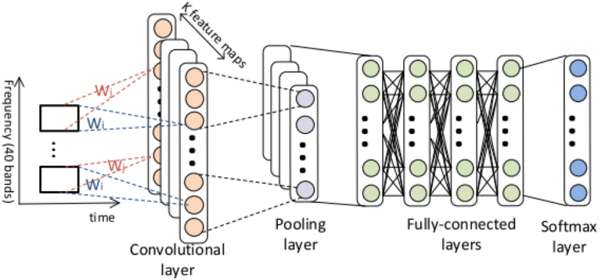


Convolutional neural networks (CNNs)

CNNs are DNNs with some twists to take into account these considerations

- They encode **local** patterns with subsets of nodes that only consider small patches of the input (**filters**)
- They encode **repetition** of patterns by applying the same weights to different patches of the input (weight **sharing**)
- They normalize for **inexact repetition** by **pooling** information over multiple areas in the input
- Developed in the mid-1990s (really even the 1980s...) by Yann LeCun and colleagues
- Became hugely popular for image processing/computer vision starting in 2012
- Borrowed into speech recognition shortly thereafter

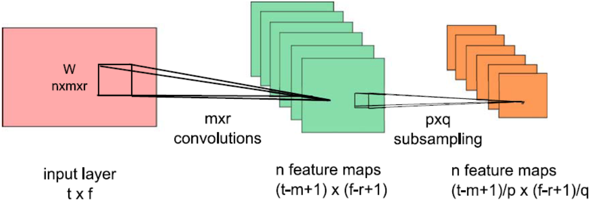
CNNs



Key ingredients: local filters, sharing for repetition, pooling for inexact repetition

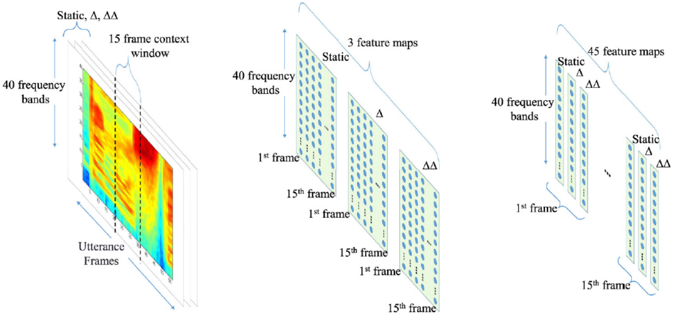
CNNs

Ofter easier to think about convolutional layers in 2D

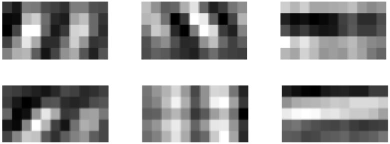


CNNs

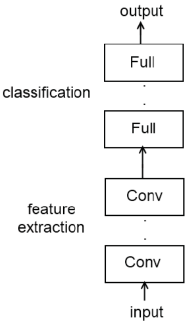
Convolutions can be applied to different kinds of features at the same time:



CNNs: Example filters



Dependence on number/types of layers



# of convolutional vs. fully connected layers	WER
No conv, 6 full (DNN)	21.6
1 conv, 5 full	21.3
2 conv, 4 full	18.9
3 conv, 3 full	20.2

Dependence on activation type

Table 12
WER on broadcast news, 50 hr.

Model	Feature	Non-linearity	dev04f
GMM/HMM	fBMMI		18.8
DNN	fMLLR	sigmoid	16.3
CNN	log-mel	sigmoid	15.8
CNN+DNN	log-mel+(fMLLR+i-vectors)	sigmoid	14.2
CNN+DNN	log-mel+(fMLLR+i-vectors)	ReLU	**13.6**
DNN	log-mel+(fMLLR+i-vectors)	ReLU	14.2

Table 13
WER on broadcast news, 400 hr.

Model	Feature	Non-linearity	dev04f
GMM/HMM	fBMMI		16.0
DNN	fMLLR	sigmoid	15.1
CNN	log-mel	sigmoid	13.5
CNN+DNN	log-mel+(fMLLR+i-vectors)	ReLU	**12.7**

CNNs applied to MFCCs