Overview

Tasks and considerations: Speech-to-text transcription, speaker diarisation (who spoke when), paralinguistic aspects (intonation, voice quality, emotion), content understanding

Main (generative) objective:

$W^* = \underset{W}{\text{argmax}} P(W|X) = \underset{W}{\text{argmax}} P(X|W)P(W)$

Difficulties: Linguistic variations (speaker, discourse, vocab, accent, socioling, languages themselves, environmental acoustics); Machine learning components (high dimensional & long input, noisy data, limited training resource, hierarchical nature of speech needs multiple models to capture)

Approaches:

- Signal processing: Feature vectors (DFT, MFCC)
- Acoustic units: phone/phoneme/grapheme/syll...
- ML challenges: training (alignment) & recognition (searching)
- Training: Generative HMM/Discriminative MMI

Evaluation: WER = (S+D+I)/N

Signal processing

Waveform preprocessing (time domain):

- Dithering (avoid $log0$): $y[t] = x[t] + \epsilon \;\; \epsilon \sim \mathcal{N}(0,1)$
- Remove DC (mean) offset: $y[t] = x[t] - \frac{1}{T}\sum_{i=1}^{T} x[i]$
- Pre-emphasis (make high-freq energy more available): $y[t] = x[t] - 0.97 * x[t-1]$

Spectrogram:

- **DFT**: for $k = 0, ..., T-1$

  $X[k] = \sum_{t=0}^{T-1} x[t] e^{-i2\pi tk/T}$, where $i = \sqrt{-1}$

  $X[k]$ represents the similarity between the waveform $[x[0], x[1], ..., x[T-1]]$ and the k^{th} basis waveform $[w^0, w^k, w^{2k}, ..., w^{(T-1)k}]$ where $w = e^{-i2\pi/T}$.

  Linearity: $\mathcal{F}\{a_1x_1 + a_2x_2\} = a_1\mathcal{F}\{x_1\} + a_2\mathcal{F}\{x_2\}$

  Shift theo.: if $y[t] = x[t-1]$, $Y[k] = e^{i2\pi k/T} X[k]$

- **Spectrum**: Short-time Fourier transform of a window typically 25ms for ASR (with 16kHz sampling rate) with a 10ms hop.

  Windowing:(elementwise) $y[t] = w[t] \cdot x[t]$

Matrix rep of DFT in a window of length T:

$$\mathcal{F}\{x\} = \begin{bmatrix} 1 & 1 & 1 & . & 1 \\ 1 & w^1 & w^2 & . & w^{T-1} \\ 1 & w^2 & w^4 & . & w^{2(T-1)} \\ & & \cdots & & \\ 1 & w^{T-1} & w^{2(T-1)} & . & w^{T-1^2} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ . \\ x[T-1] \end{bmatrix} = \begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ . \\ X[T-1] \end{bmatrix}$$

- **Magnitude and phase:** Phase not as important as magnitude for speech intelligibility, usually used to locate sound. Energy = Mag^2

Signal processing cont.

To reveal formants:

- **Mel-filters:** asymmetric triangle-shaped filters, sensitive to energy & higher resolution in lower frequencies (reflect human auditory insensitiveness to difference in high frequencies). Equally placed on a mel(log)-scale

  In one analysis window, for each filter, we compute the sum of the magnitude in the frequencies falling in the filter: $Y[n] = \sum_{k=0}^{T-1} X[k] \cdot H_n[k]$, where $H_n[k]$ represents the co-efficient of the k-th frequency bin in the n-th filter. Applying the mel-filter bank H to a magnitude spectrum X as matrix multiplication to obtain the mel spectrum Y looks like:

$$Y = HX = \begin{bmatrix} H_1 \\ H_2 \\ \cdots \\ H_n \end{bmatrix} X = \begin{bmatrix} H_1[0] & H_1[1] & \cdots & H_1[T-1] \\ H_2[0] & H_2[1] & \cdots & H_2[T-1] \\ . & . & . & . \\ H_N[0] & H_N[1] & \cdots & H_N[T-1] \end{bmatrix} \begin{bmatrix} X[0] \\ X[1] \\ \cdots \\ X[T-1] \end{bmatrix}$$

- **MFCCs:** We take the log and apply DCT to the mel-spectrums to get the cepstrum. To reveal formants, truncate high-freq MFCCS; we can rebuild a smoother spectrum on low-freq MFCCs.

HMM basics

Hidden: We cannot observe the state sequence;

Markov: transition prob only depends on the previous state; emission depends only on the current state;

Parameters θ**:**Transition $a_{kj} = P(q_{t+1} = j | q_t = k)$; observation $b_j(x) = P(x|q = j)$

Typologies: left-to-right (3-state HMMs commonly used to model phones in ASR), parallel L2R (can go state $n \to n+2$), ergodic (full connections)

Forward algo: compute likelihood of an observation sequence $P(X|\theta) = \sum_{Q \in \mathcal{Q}} P(X, Q|\theta)$

- Forward prob: $\alpha_j(t) = p(x_1, ..., x_t, q_t = j|\theta)$
- Initialisation: $\alpha_j(t=0) = 1 \; if \; j = 0 \; else \; 0$
- Recursion: $\alpha_j(t) = \sum_{i=0}^{J} \alpha_i(t-1) a_{ij} b_j(x_t)$
- Termination: $P(X|\theta) = \alpha_E = \sum_{i=1}^{J} \alpha_i(T) a_{iE}$

Viterbi decoding: find the maximum likelihood state sequence $\hat{Q} = \underset{Q}{\text{argmax}} P(X, Q|\theta)$

- Likelihood of the most probable partial path in state j at t: $V_j(t) = \underset{Q_{len=t}}{\max} P(X[0:t], Q[t] = j|\theta)$
- **Backpointer** $B_j(t)$ indicates the previous state on the most probable path with likelihood $V_j(t)$
- **Init:** $V_j(t=0) = 1 \; if \; j = 0 \; else \; 0$; $B_j(t=0) = 0$
- **Recur:** $V_j(t) = \underset{i}{\max} V_i(t-1) a_{ij} b_j(x_t)$

  $B_j(t) = \underset{i}{\text{argmax}} V_i(t-1) a_{ij} b_j(x_t)$

HMM basics cont.

- **Term:** $V_E = \underset{i}{\max} V_i(T) a_{iE}$

  $B_E(T) = \underset{i}{\text{argmax}} V_i(T) a_{iE}$

Generative Training (HMM)

Training objective: $\hat{\theta} = \underset{\theta}{\text{argmax}} P(X, Q|\theta)$

Viterbi training of a_{ij} & $b_j(x)$:

1. Viterbi decodes the hard Maximum Likelihood state-time alignment on data based on current parameters;
2. Update transition probs based on the output $\hat{a}_{ij} = \frac{C(i \to j)}{\sum_{j'} C(i \to j')}$;
3. Update observation probs' mean $\hat{\mu_j} = \frac{\sum_t z_j(t) \mathbf{x}_t}{\sum_t z_j(t)}$ and covariance $\hat{\Sigma_j} = \frac{\sum_t z_j(t) (\mathbf{x}_t - \hat{\mu_j})(\mathbf{x}_t - \hat{\mu_j})^\top}{\sum_t z_j(t)}$, where $z_j(t)$ indicates if in the output $\mathbf{x}_t$ aligns with j;
4. Re-estimate the ML alignment and update parameters; iterate until the max $P(X, Q|\theta)$ doesn't improve.

Expectation Maximisation of a_{ij} & $b_j(x)$:

1. Forward-backward algo computes expected state-occupation probs $\gamma_j(t) = P(q_t = j|X, \theta)$ and arc-occupation probs $\xi_{ij}(t) = P(q_t = i, q_{t+1} = j|X, \theta)$;
2. Update transition probs as $\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_{ij}(t)}{\sum_{t=1}^{T} \sum_{j'=1}^{J} \xi_{ij'(t)}}$;
3. Update observation probs' $\hat{\mu_j} = \frac{\sum_t \gamma_j(t) \mathbf{x}_t}{\sum_t \gamma_j(t)}$ and $\hat{\Sigma_j} = \frac{\sum_t \gamma_j(t) (\mathbf{x}_t - \hat{\mu_j})(\mathbf{x}_t - \hat{\mu_j})}{\sum_t \gamma_j(t)}$; then iterate.

Backward prob: $\beta_j(t) = p(x_{t+1}, ..., x_T | q_t = j, \theta)$

- Init: $\beta_j(T) = a_{iE}$
- Recur: $\beta_i(t) = \sum_{j=1}^{J} \beta_j(t+1) a_{ij} b_j(x_{t+1})$
- Term: $P(X|\theta) = \beta_0 = \sum_{j=1}^{J} \beta_j(1) a_{0j} b_j(x_1) = \alpha_E$

Occupation probs:

$\gamma_j(t) = P(q_t = j|X, \theta) = \frac{p(X, q_t = j|\theta)}{p(X|\theta)} = \frac{\alpha_j(t)\beta_j(t)}{\alpha_E}$;

$\xi_{ij}(t) = P(q_t = i, q_{t+1} = j|X, \theta) = \frac{p(X, q_t = i, q_{t+1} = j|\theta)}{p(X|\theta)}$

$= \frac{\alpha_i(t) a_{ij} b_j(x_{t+1}) \beta_j(t+1)}{\alpha_E}$

Note. **Multivariate Gaussian Distribution:** To model D-dimensional data $\mathbf{x} = (x_1, \ldots, x_D)^\top$ with multivariate Gaussian distribution, we define the probability density function $p(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ as:

$$\frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

where the mean vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_D)^\top$ and

Gen Train cont. GMM

the symmetric ($\sigma_{ij} = \sigma_{ji}$) covariance matrix $\mathbf{\Sigma} = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1D} \\ . & . & . \\ \sigma_{D1} & \cdots & \sigma_{DD} \end{bmatrix}$. If $\sigma_{ij} > 0$ then $x_i \uparrow x_j \uparrow$.

Mean and covariance expectation:

$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$; $\hat{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\mu})(\mathbf{x}_n - \hat{\mu})^\top$, where n specifies which data vector.

Extend to GMM:

Mixture model: mix multiple pdfs into one model

$$p(x) = \sum_{m=1}^{M} P(C = m)p(x|C = m)$$

where C specifies which component, $P(C = m)$ the weight of component m, $p(x|C = m)$ the prob. den. of data x given by m. For Gaussian Mixture Model, $p(\mathbf{x}|m) = \mathcal{N}(\mathbf{x}; \mu_\mathbf{m}, \mathbf{\Sigma_m})$.

EM estimation of GMM parameters: The component-data alignment is unknown.

1. Estimate with current parameters the component occupation probabilities

$$P(m|x) = \frac{p(x|m)P(m)}{p(x)} = \frac{p(x|m)P(m)}{\sum_{m'=1}^{M} p(x|m')P(m')}$$

$N_m^* = \sum_{n=1}^{N} P(m|x_n)$ computes a soft measure of how much data m generates, given the estimation.

2. Update parameters: $\hat{P}(m) = \frac{N_m^*}{N}$, $\hat{\mu}(m) = \frac{\sum_n P(m|x_n)x_n}{N_m^*}$, $\hat{\Sigma}_m = \frac{\sum_n P(m|x_n)(x_n - \hat{\mu}_m)(x_n - \hat{\mu}_m)^\top}{N_m^*}$

Other notes:

- GMMs scale well: large GMM-based ASR can have 30k GMMs each with 32 components or much more;
- Ususally probabilities are computed in log domain to avoid underflow problems.

HMM/GMM Training:

- Observation probs as GMMs:

$$b_j(\mathbf{x}) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(\mathbf{x}; \mu_\mathbf{jm}, \mathbf{\Sigma_{jm}})$$

- Compute component-state occupation probabilities $\gamma_{jm}(t)$ of being in state j at time t with the m^{th} mixture component accounting for the observation $\mathbf{x_t}$;

Gen Train cont. HMM/GMM

- Update parameters (transition probs unaffected) of component m of state j:

$\hat{\mu_{\mathbf{jm}}} = \frac{\sum_t \gamma_{jm}(t)x_t}{\sum_t \gamma_{jm}(t)}$

$\hat{\mathbf{\Sigma}_{\mathbf{jm}}} = \frac{\sum_t \gamma_{jm}(t)(\mathbf{x_t} - \mu_{\hat{jm}})(\mathbf{x_t} - \mu_{\hat{jm}})^\top}{\sum_t \gamma_{jm}(t)}$

$\hat{c_{jm}} = \frac{\sum_t \gamma_{jm}(t)}{\sum_{m'=1}^{M}\sum_t \gamma_{jm'}(t)}$

Summary of HMM (generative) training:

- To maximise $p(X|W) = \sum_{Q \in B(W)} p(X, Q|W)$
- $p(X, Q|W) = p(q_1)p(x_1|q_1)) \prod_{t=2}^{T} p(q_t|q_{t=1})p(x_t|q_t)$
- $B(W)$ denotes acceptable state sequences for W, which can be represented as an FST $U \circ H \circ C \circ L \circ W$
- Search $\hat{\theta} = \text{argmax}_\theta P(X|W)$ with EM (GMM) or gradient descent (DNN)

Discriminative Training (MMI)

Discriminative approach:

- Solve $\hat{\theta} = \text{argmax}_\theta P(W|X)$
- Better solution where the decision boundary is easy to learn;
- For prediction task, learning the boundary is more intuitive.

Maximum Mutual Information (MMI):

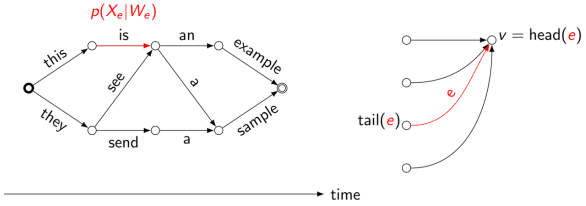
$$p(W|X) = \frac{p(X|W)p(W)}{p(X)} = \frac{p(X|W)p(W)}{\sum_{W'} p(X|W')p(W')}$$

To compute $p(X|W)p(W)$:

- Phone sequence HMMs to compute $p(X|W)$
- Language model to compute $p(W)$

To compute $\sum_{W'} p(X|W')p(W')$:

- Too many Ws exist which make the exact computation expensive, although it is possible to achieve lattice-free with GPU (since forward algorithm can be seen as matrix multiplication).
- We approximate with a set of high-probability word sequences D: $\sum_{W' \in D} p(X|W')p(W')$.
- D can be formalised as a lattice with edge probabilities $p(X_e|W_e)$ as parameters:



Disc Train cont. MMI

- Forward: $\alpha(v) = \sum_{e \in into(v)} p(X_e|W_e)\alpha(tail(e))$, where $\alpha(final) = \sum_{W' \in D} p(X|W')$
- Combine forward computation - $\sum_{W' \in D} p(X|W')$ and LM-$p(W')$ gives $\sum_{W' \in D} p(X|W')p(W')$.

Optimise edges with gradient descent:

$$\frac{\partial L}{\partial p(X_e|W_e)} = \sum_v \frac{\partial L}{\partial \alpha(v)} \frac{\partial \alpha(v)}{\partial p(X_e|W_e)}$$
$$= \sum_v \frac{\partial L}{\partial \alpha(v)} \alpha(\text{tail}(e)) \mathbb{1}_{v=\text{head}(e)}$$
$$= \frac{\partial L}{\partial \alpha(\text{head}(e))} \alpha(\text{tail}(e))$$

$$\frac{\partial L}{\partial \alpha(u)} = \sum_v \frac{\partial L}{\partial \alpha(v)} \frac{\partial \alpha(v)}{\partial \alpha(u)}$$
$$= \sum_v \frac{\partial L}{\partial \alpha(v)} \sum_e P(X_e|W_e) \mathbb{1}_{u=\text{tail}(e), v=\text{head}(e)}$$
$$= \sum_{e \in \text{out}(u)} \frac{\partial L}{\partial \alpha(\text{head}(e))} P(X_e|W_e)$$

(Lecture mentioned this is not examinable.) MMI objective considers the LM while training the HMMs.

Why name MMI? ↓ MMI definition ↓

$$\underset{\lambda}{\text{argmax}} \sum_{X,W} p(X, W) \log \frac{p(X, W)}{P(X)P(W)}$$
$$= \underset{\lambda}{\text{argmax}} \sum_{X,W} p(X, W) \log p(W|X) - \sum_{W,X} p(X, W) \log p(W)$$
$$= \underset{\lambda}{\text{argmax}} \sum_{X,W} p(X, W) \log p(W|X) - \sum_{W} p(W) \log p(W)$$

Entropy of the language model - irrelevant

$$\approx \underset{\lambda}{\text{argmax}} \frac{1}{N} \sum_{n=1}^{N} \log p(W_n|X_n)$$

But: MMI aims at minimising zero-one loss. We can replace it with other loss functions to make the objective more sophiticated.

$$\mathbb{E}_{W' \sim p(W'|X)}[\mathbb{1}_{W \neq W'}] = 1 - \mathbb{E}_{W' \sim p(W'|X)}[\mathbb{1}_{W = W'}]$$
$$= 1 - p(W|X)$$

MMI objective

$$\underset{\lambda}{\text{argmin}}\, \mathbb{E}_{W' \sim p(W'|X)}[\mathbb{1}_{W \neq W'}] = \underset{\lambda}{\text{argmax}}\, p(W|X)$$

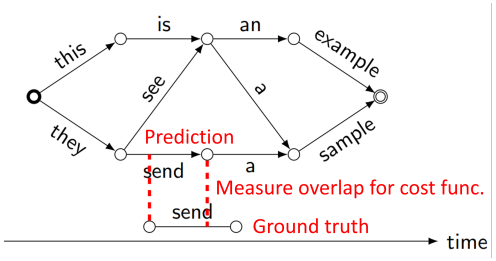
E.g., **Minimum Bayes Risk (MBR):** An alternative to MMI which allows partial credits and user-defined cost function. (see next page)

Disc Train cont. MMI & MBR

$$\mathbb{E}_{W'\sim p(W'|X)}[\underbrace{\text{cost}(W,W')}_{\text{Customised cost function}}] = \sum_{W'} p(W'|X)\text{cost}(W,W') = \frac{\sum_{W'} p(X|W')p(W')\text{cost}(W,W')}{\sum_{W''} p(X|W'')p(W'')}$$

as $p(W'|X) = \frac{p(X|W')p(W)}{p(X)} = \frac{p(X|W')p(W')}{\sum_{W''} p(X|W'')p(W'')}$, where both the nominator and denumerator need to be formalised as lattice.

- **We want to compute the cost along with edges in the lattice so WER won't work.** Instead, we can measure time overlap between the ground truth and the prediction to compute the cost.

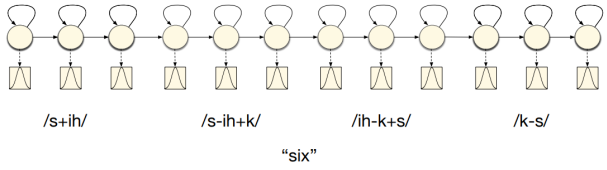


- If the cost formalises mismatches at phone level, it is called Minimum Phone Error (MPE). If at word level, then Minimum Word Error (MWE).

Context-dependent modelling

Back to HMM: transition $a_{ij} = P(q_{t+1} = j|q_t = k)$; observation $b_j(x) = P(x|q = j)$

Context-dependent phone models: The acoustic context may affect the segment's realisation through coarticulation. We may directly model the surface phonetics (e.g., "did you" [d ih jh y ah]), but this suffers from individual or demographic variations. Or, to model subword units with incorporated context (e.g., biphones, syllables) as **context-dependent phone models**. Triphone modelling of 'six' looks like:



For word sequences, the phone context can cross word boundaries, or be word-internal.

Advantages of CD models:

- More specific than context-independent ones, which compensate the naiveness of the Markov assumption;

Context-dependent modelling

- Each CD model becomes responsible for a smaller region of the acoustic-space, which can make the modelling more fitted;
- No need for more fine-grained context-independent models.

However, if we train an individual model for each CD phone, we won't have enough data. Many rare CD phones might not occur in the dataset at all. **Solutions** include smoothing and parameter sharing.

Smoothing

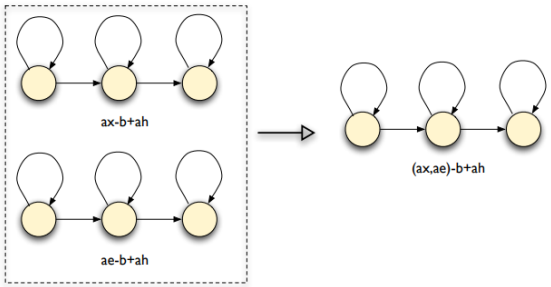
Backoff: Use a minimum training example count to determine whether a triphone should be modelled or backed-off to a biphone (likewise for biphones) e.g., sh-iy+l → iy+l. This results in few triphone models.

Interpolation: $\hat{\lambda}^{tri} = \alpha_3\lambda^{tri} + \alpha_2\lambda^{bi} + \alpha_1\lambda^{mono}$, where αs can be optimised through *deleted interpolation*. This enables more triphone models to be estimated.

Parameter sharing

- Share Gaussian (tied mixture): let distributions share the same set of Gaussians but have different mixture weights.

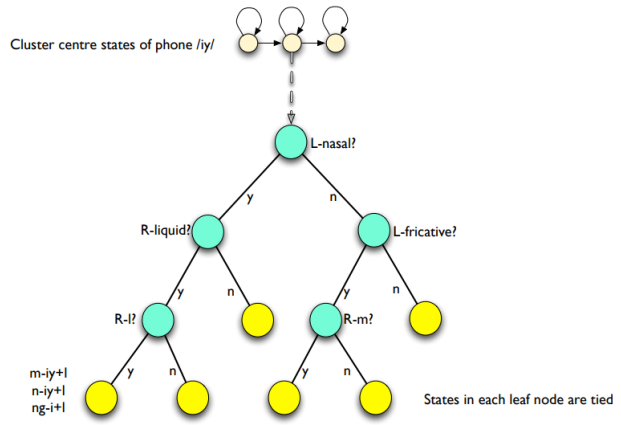
- Share models (generalised triphones): Merge similar context-dependent models so we have more available data for fewer models.



- Share states (clustering): We CANNOT just build every triphone model and then cluster states because there are unseen triphones. Instead, build a decision tree for each state of each base phone type, with questions on phonetic context to split data into pools of state clusters.

- At each node, iterate through all questions and choose the one bringing the largest increase in the likelihood of data $\max \Delta = L(S_{yes}) + L(S_{no}) - L(S)$.

CD modelling cont. state clustering



- Log likelihood of data associated with a cluster of state $L(S) = \sum_{s\in S}\sum_{x\in X}\gamma_s(x)\log p(x|\mu_{\mathbf{S}}, \Sigma_S)$, where $\gamma_s(x)$ the state occupation probability that x was generated by state s.
- If the output pdfs are Gaussian, then

$$L(S) = -\frac{1}{2}\left(\log\left((2\pi)^d|\Sigma_S|\right) + d\right)\sum_{s\in S}\sum_{x\in X}\gamma_s(x)$$

where d the data dimension. We compute Σ_S from individual states in the cluster and $\gamma_s(x)$ along with forward backward, and need not to iterate data again.

- We iteratively split clusters into smaller ones until the maximum $\Delta <$ a threshold, or there is no enough data to split.
- Since GMMs offer better estimation, we can post-process the original Gaussian distributions in state clusters into GMMs. To be specific, for each Gaussian, we clone it and perturbe the mean of the clone a bit, and retrain the original one and the copy as distinct mixture components. Then we repeat by splitting the dominant (highest state occupation count) mixture components in each state.

Large vocab modelling (efficient decoding)

Increased computational cost: Recognise connected words with n-gram language model massively extend the search space. Exact search not possible for large vocab tasks!

Solutions: beam search, tree-structured lexicons, language model look-ahead, dynamic search structures (like token passing I suppose), multipass search (two stage decoding), best-first search (stack decoding with a priority que).

Efficient decoding cont.

- **Beam search:** During Viterbi decoding, don't propagate paths whose probability falls below a threshold (e.g., a certain amount less than the current best path). NO guarantee to find the global optimum.
- **Tree-structured lexicon:** Reduces the number of state transition computations.
- **Language model look-ahead:** Know the best future in advance to prune unpromising paths earlier.

WFST Models & Compositions

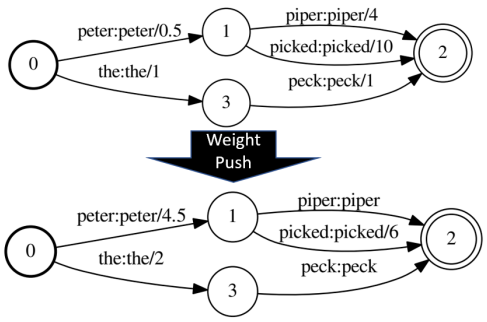
Overview: WFST are weighted finite state automaton that transduces an input sequence to an output sequence; each edge has an in label, an out label, and a weight as negative log likelihood ($w = -log(p)$).

Weight computation: $p_a * p_b \rightarrow w_a + w_b$; $p_a + p_b \rightarrow -log(e^{-w_a} + e^{-w_b})$; $\max P(path) \rightarrow \min W(path)$

WFST operations: determinisation, minimisation, composition, weight pushing.WFST operations help to **increase memory and run-time efficiency**. Standard approach is to determinise and minimizse after each composition as $H \circ C \circ L \circ G = min(det(H \circ min(det(C \circ min(det(L \circ G))))))$ for Kaldi.

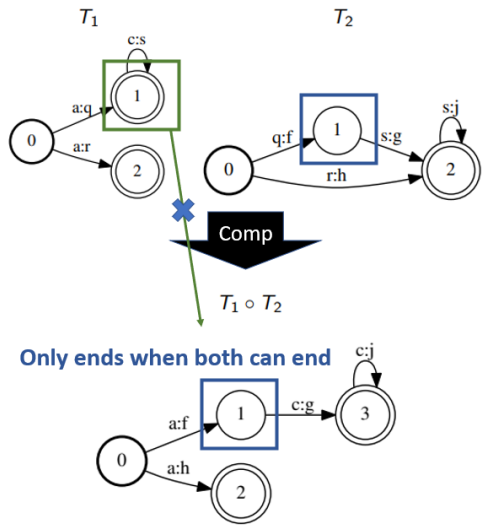
	transducer	input sequence	output sequence
G	word-level grammar	words	words
L	pronunciation lexicon	phones	words
C	context-dependency	CD phones	phones
H	HMM	HMM states	CD phones

- **Determinisation:** Ensure that each state has only one out transition for a given input label.
- **Minimisation:** Transforms a transducer to an equivalent transducer with the fewest possible states and transitions.
- **Weight pushing:** Apply look-ahead



- **Composition:** Combine transducers T1 and T2 into a single transducer acting as if the output of T1 was passed into T2.

WFST cont.



Neural Network

Linear regresson: $\mathbf{f(x) = Wx + b}$, where output phone score $\mathbf{f} = (f_{1=/aa/}, f_{2=/ae/}, ..., f_{J=/zh/})$, input vector $\mathbf{x} = (x_1, x_2, ..., x_D)$ (dim=D), weight matrix $\mathbf{W}$ of dimension $D \times J$, bias $\mathbf{b} = (b_1, ..., b_J)$ (dim=J).

- Mean square error $E = \frac{1}{2} \cdot \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{f(x_t)} - \mathbf{r}(t)\|^2$, which equals zero when the output $\mathbf{f}(\mathbf{x}_t)$ equals the target output $\mathbf{r}(t) = (0, ..., 1, ..., 0)$ for all t
- Stochasitic Gradient Descent: 1) Initialise parameters with small random numbers; for each epoch with a small batch of data, 2)Feed-forward to compute outputs; 3) Back-propagate to sum up gradients; 4) Update $w' = w - \eta \partial E / \partial w$; 5) iterate until the n-th epoch, or when the error stops decreasing.
- Gradient for E^t:

$$E^t = \frac{1}{2} \sum_{j=1}^{K} (f_j^t - r_j^t)^2 = \frac{1}{2} \sum_{j=1}^{J} \left(\sum_{d=1}^{D} (w_{jd} x_d^t + b_j) - r_j^t \right)^2$$

$$\frac{\partial E^t}{\partial w_{ji}} = (f_j^t - r_j^t) x_i^t = g_j^t x_i^t \qquad g_j^t = f_j^t - r_j^t \qquad \frac{\partial E^t}{\partial b_j} = g_j^t$$

Extensions:

Softmax: $y_j = \frac{\exp(f_j)}{\sum_{k=1}^{K} \exp(f_k)}$ turn real number into probability distribution.

Cross-entropy: $E^t = -\sum_{j=1}^{J} r_j^t \ln y_j^t = -\ln y_{true}^t$, where $\frac{\partial E^t}{\partial w_{jd}} = (y_j^t - r_j^t) x_d^t$ (logistic regression).

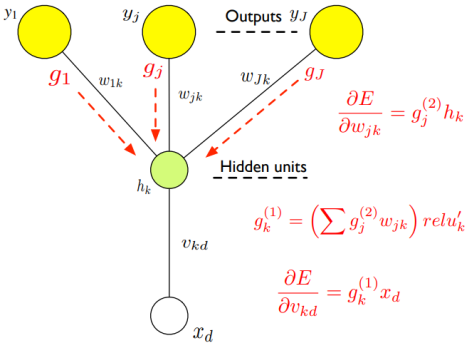
Add context: $x_t' = concat(x_{t-n}, ..., x_t, ..., x_{t+n})$.

Hidden layers: $\mathbf{h} = \alpha(\mathbf{Vx} + \mathbf{b^0})$-can stack to make multilayer DNN, $\mathbf{y} = softmax(\mathbf{Wh} + \mathbf{b^1})$

Neural Network cont. DNN

Gradient of $\alpha(\cdot)$: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, $relu'(x) = 1$ if $x > 0$ else 0

Back-propagation example: $g_j^{(2)} = y_j - r_j$



Hybrid DNN/HMM

We can replace GMMs with with outputs of neural networks. As $P(q_t|x_t) = \frac{p(x_t|q_t=j)P(q_t=j)}{p(x_t)}$, where $\frac{p(x_t|q_t=j)}{p(x_t)} = \frac{P(q_t=j|x_t)}{P(q_t=j)}$(scaled liklihood) can be computed by dividing the NN output with the relative frequency of class j in the training data.

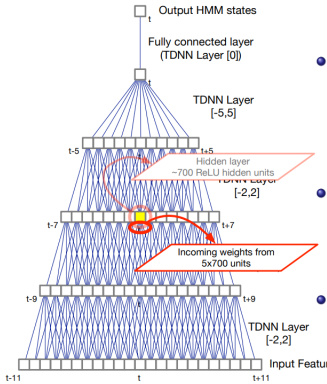
Modelling phonetic context: from NN to DNN

- Early approaches model the primary class, y, and its context, c with $p(y, c|x) = p(y|c, x)p(c|x)$ computed by two NNs. But we don't know the context and iterating all possibilities costs massive effort.
- **Tandem scheme:** use the output probabilities or intermidiate representations from the NN (good at modelling acoustic context) as input to CD-HMM-GMM system (good at speaker adaptation, phonetic context, and sequence-training).
- NN models correlated features more easily and flexibly (not require features to be uncorrelated) than GMM (requires covariance matrix). Later GPUs come to support the development of DNN systems.
- **Context-dependent hybrid HMM/DNN:** 1) train a CD-HMM/GMM system with tied states; 2) use Viterbi to do frame-state alignment; 3) pretrain an NN to capture this alignment (in-feature vector; out-distribution over context-dependent tied states); 4) train this NN with gradient descent; 5) decode with the CD scaled likelihood.
- **Compare with earlier NN/HMM**, DNN/HMM systems can model context-dependent tied states with a much wider output layer and can use **correlated features or higher resolution MFCCs**.

Neural Network cont. DNN models

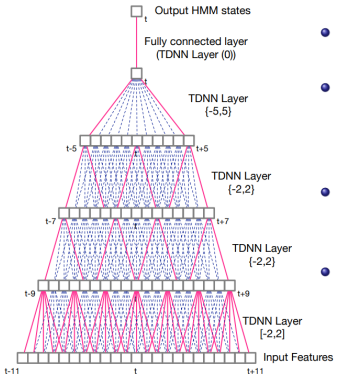
DNN architectures: extending context window

- **Time-delay neural networks (TDNNs)**: Each edge represents a weight matrix of shape input-dimension*number-of-hidden-units.



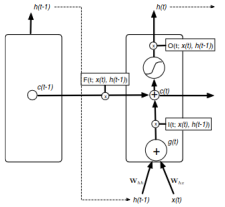
- View a TDNN as a 1D convolutional network with the transforms for each hidden unit tied across time
- TDNN layer with context [-2,2] has 5x as many weights as a regular DNN layer
- More computation, more storage required!

- **Sub-sampled TDNNs:** not all frames are used in forward pass. Window can also be **asymmetric** for cleaner sub-sampling.



- Sub sample window of hidden unit activations
- Large overlaps between input contexts at adjacent time steps – likely to be correlated
- Allow gaps between frames in a window (cf. dilated convolutions)
- Sub-sampling saves computation and reduces number of model size (number of weights)

- **RNNs** (simple, bidirectional, deep) and **LSTMs**:



$$\boldsymbol{I}(t) = \sigma\left(\boldsymbol{W}_{ix}\boldsymbol{x}(t) + \boldsymbol{W}_{ih}\boldsymbol{h}(t-1) + \boldsymbol{b}_i\right)$$
$$\boldsymbol{F}(t) = \sigma\left(\boldsymbol{W}_{fx}\boldsymbol{x}(t) + \boldsymbol{W}_{fh}\boldsymbol{h}(t-1) + \boldsymbol{b}_f\right)$$
$$\boldsymbol{O}(t) = \sigma\left(\boldsymbol{W}_{ox}\boldsymbol{x}(t) + \boldsymbol{W}_{oh}\boldsymbol{h}(t-1) + \boldsymbol{b}_o\right)$$
$$\boldsymbol{g}(t) = \boldsymbol{W}_{hx}\boldsymbol{x}(t) + \boldsymbol{W}_{hh}\boldsymbol{h}(t-1) + \boldsymbol{b}_h$$
$$\boldsymbol{c}(t) = \boldsymbol{F}(t) \circ \boldsymbol{c}(t-1) + \boldsymbol{I}(t) \circ \boldsymbol{g}(t)$$
$$\boldsymbol{h}(t) = \boldsymbol{O}(t) \circ \tanh\left(\boldsymbol{c}(t)\right)$$

- **Transformers:** Self-attention, multi-head, problematic on long input context.

End-to-End 1: CTC

Connectionist Temporal Classification (CTC) is a discriminative approach to sequence, used in Deep Speech - an bidirection LSTM end-to-end ASR system (maps from acoustic frames X to subword sequences Y consisting of characters/phones), with n-gram LM.

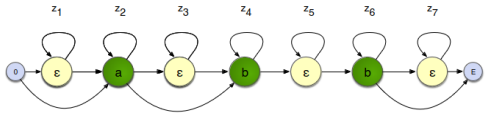
End-to-End 1: CTC

Alignment & compression: Each frame aligns with a symbol (a concrete one or the blank ϵ) monotonically, e.g., *hhϵϵϵϵlllϵlllϵoo* or *hhϵϵϵϵϵlϵϵoo* (label sequence C), respectively compressed into [hello] or [heelo] (output subword Y) as the output. A single frame cannot map to many outputs (a problem for e.g., 'th').

CTC loss function (NLL): $\mathcal{L}_{CTC} = -\log P(Y|X)$ for true Y, where $P(Y|X) = \sum_{C \in Y^*} P(C|X)$, with Y^* denoting all expansions of Y, and $P(C|X) = \prod_{t=1}^{T} P_t(c_t|X)$ with $P_t(c_t|X)$ denoting the probability of outputting c_t as the t-th label (alignment).

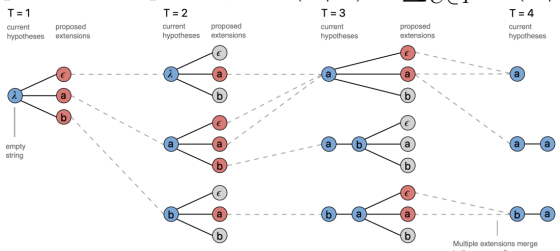
Dynamic programming to compute $P(Y|X)$:

- Define a symbol sequence $Z = (z_1, ..., z_j, ..., z_J) = (\epsilon, s_1, \epsilon, s_2, ..., \epsilon, s_M, \epsilon)$ where each s denotes a concrete symbol in Y.
- Build an L2R HMM on Z, e.g., below for $Y = [a, b, b]$, with extra transitions from $z_{i-2} \rightarrow z_i$ if $z_i \neq \epsilon$ and $z_i \neq z_{i-2}$.



- Forward algo: **Init.** $\alpha_i(t=0) = 1 \; if \; i = 0 \; else \; 0$; **Recur.** $\alpha_i(t) = p_t(z_i|X) \sum_{k \in islast(i)} \alpha_k(t-1)$; **Term.** $P(Z|X) = \alpha_{J-1}(T) + \alpha_J(T)$ (only the J-1th and Jth state can transit to the end state).

CTC decoding: Beam search, merge hypothesis with the same prefix so Cs of the same Y don't compete but cooperate, as $P(Y|X) = \sum_{C \in Y^*} P(C|X)$.

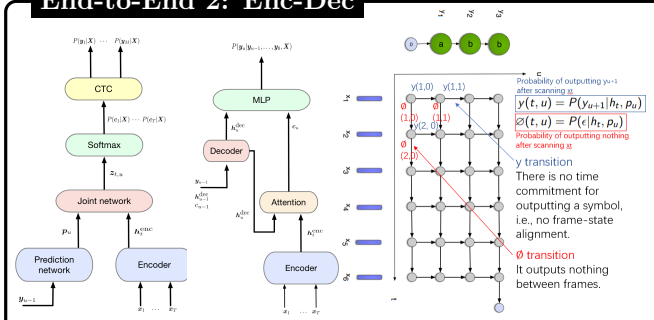


LM interpolation: last term penalises inappropriate lengths. (Optional modification: instead of LM $P(W)$, use subword LM $P(Y)$; use an FST T which transforms C into Y, then compose with L and G.)

$$\hat{W} = \underset{W}{\text{argmax}}(\log P(Y|X) + \lambda \log P(W)) + \eta L(W)$$

Cons of CTC: independence between outputs, need of an ad-hoc LM NOT trained together with the acoustic model

End-to-End 2: Enc-Dec



- **RNN Transducer** (left) can operate in online mode. *Encoder* maps acoustic frames to hidden vectors, *prediction network* takes the previous output label to predict next label (act like LM), *joint network* combines the outputs from both to compute a joint vector z used to predict c; CTC maps C to Y as before. **Everything** is optimised with the CTC loss. **Forward prob** (right) $\alpha_u(t) = P(\mathbf{y}_{<u+1}|\mathbf{h}^{enc}_{<t+1}) = \alpha_u(t-1)\phi(t-1,u) + \alpha_u(t-1)y(t,u-1)$

- **Attention-based enc-dec** (mid): encoder outputs h_t^{enc}; then compute attn weight $\alpha_{ut} = Attn(h_u^{dec}, h_t^{enc})$ for each frame with the decoder state $h_u^{dec} = RNN(y_{u-1}, h_{u-1}^{dec}, c_{u-1})$, where the contextual vector (not symbol!) $c_u = \sum_{t=1}^{T} \alpha_{ut} h_t^{enc}$; lastly estimate the subword distribution $y_u \sim MLP(c_u, h_u^{dec})$ with one feedforward layer and a softmax. $Attn(\cdot)$ can be a hidden layer followed by softmax.

- **Pyramid enc**: Deep RNN $h_t^j = RNN(h_t^{j-1}, h_{t-1}^j)$ can be hard to train due to long input context. Instead, let higher layers to reduce the resolution by factor 2 (concatenating 2 consecutive hidden states): $h_j^t = pyrRNN([h_{2t-1}^{j-1}, h_{2t}^{j-1}], h_{t-1}^j)$

Scheduled sampling: to max $\sum_u \log P(y_u|X, y_{<u})$, a) setting $y_{<u}$ to be previous predictions adds noise to training; b) teacher forcing trains faster but creates mismatch between training and testing; or, c) **sample a label from estimated distribution** to reduce noise at training and restrict train-test gap.

Decoding: With enough data, no acute need for pronunciation/language models. Simply decode the grapheme sequence with beam search.

Other refinements: Wordpiece models (BPE), multihead attn, interpolate WER for training, label smoothing, LM rescoring, and

- As attn doesn't impose monotonic constraint, use **windowed attn** (restricted to apply on a set of encoder hidden states) or **hybrid CTC/Attn** model (use both jointly during training and recognition).

Speaker Adaptation

Variations: accents, anatomy, channel conditions...
Adapataion can be supervised/unsupervised, static/dynamic. It should reduce the mismatch between speaker-dependent (SD) test data and the trained models within compact parameter space and low computational cost, and applicable to different models.

- Maximum a posteriori (MAP) of HMM/GMM parameters: interpolate speaker-independent (SI) parameters with those of the SD data. E.g., for GMM means, $\hat{\mu} = \frac{\tau\mu_0 + \sum_t \gamma(t)x_t}{\tau + \sum_n \gamma(t)}$, where τ (typically $0 \leq \tau \leq 20$) weights the SI μ_0 and the adaptation data x_t at time t, with $\gamma(t)$ the occupation probability of this Gaussian. **Cons:** only adapts parameters in observed components; most Gaussians are not adapted.

- Maximum likelihood linear regression (very effective and efficient): learn a linear transform from adaptation parameters to be shared across Gaussians in *a regression class* (e.g., one phone type/broad class/speech vs non-speech/all): $\hat{\mu} = A\mu + b, A \in \mathbb{R}^{d*d}$ where d stands for the observation dimension. If assume $W = [bA]$ and $\eta = [1\mu^\top]^\top$ then $\hat{\mu} = W\eta$, which maximizes the likelihood of the adaptation data.

- Constrained MLLR (cMLLR): use the same linear transform for both mean and covariance (equivalent to normalising data aka feature-space MLLR): $L = \mathcal{N}(x_t; A'\mu - b', A'\mathbf{\Sigma}A'^\top) = \mathcal{N}(Ax_t + b; \mu, \Sigma)$, with $A' = A^{-1}, b' = Ab$. Same tranformation can be used to normalise input **for SI HMM/NN models**.

- Speaker adaptive training: base model designed to be adapted on SD data. E.g. **GMM UBM system** concatenates the target speaker mean parameters to form a GMM supervector m_s typically with 2048*39-dimension components. An utterance u's SD supervector $m_u = m_0 + Tw_u$, where m_0 the SI UMB supervector of dimension D, w_u is the SD i-vector of dimension d for u (reduced from m_s?), and T is a SI $D \times d$ projection matrix, estimated with EM.

- Linear Input Network for SI HMM/NN system: instead of cMLLR, learn a linear layer for normalising SD data before feeding into the NN, with supervised training and all other layers frozen.

- Speaker codes: Learn a SD vector for each talker or use i-vector (very effective for dynamic adaptation) extracted from all SD data, add to adaptation NN layers as extra input.

- Learning Hidden Unit Contributions: learn an SD amplitude for each hidden unit (all set to 1 if SI).

Multilingual and low-resource ASR

Morphology challenges & solutions: compounding (decompose into morphemes), inflection (specific models to handle)
Code switching: common in low-resource languages, need to predict switching point, potential change in phonology, hard to model with monolingual data.
General principle: share model params across languages, learn a *multilingual representation* of speech.

- **Multilingual phone sets:** extend phone set to broader coverage; train multilingual representation for output prediction.
- **Hat-swap:** pretrain on monolingual data, then add an specific output layer for each other language and finetune the model on multilingual data. When returning to the original language, performance becomes better.
- **Multi-lingual networks (block softmax):** instead of finetune on languages one-by-one, train one network using all multilingual data in parallel. Each language has its own dedicated output layer, with other layers shared. (I suppose the input includes some language id.)
- **Bottleneck (BN) network:** extract intermediate feature from source langauge network or from multilingual network (including target langguage data during training) and augmented with target language input to train the target lang network.

Semi-supervised training uses limited data to seed a model and to transcribe more, with help from a strong LM. Traditionally, we apply data filtering based on confidence score to find useful data, but this would filter away those valuable hard-to-recognise ones. Or, we can use a lattice to incorporate uncertainty and train with MMI criterion (no idea how to do). Additionally, we can **pretrain** on multi or monolingual data and **finetune** on the target language (like mBART), or do **self-supervised training** (like masked LM for BERT).
Grapheme-based ASR: Instead of using a pronunciation model to convert phones into words, directly output graphemes rather than a sequence of phones. But might be problematic for sound-letter mismatch (e.g., English) and writing systems with unsupported characters (e.g., Xhosa). E.g. Large multiling model *Whisper* is trained for abt 0.7million hours on 97 langs, with a universal grapheme set as the output.
Bottom-up approaches: Cluster acoustics and map to phones.

Multilingual cont.

Top-down approaches: use a multilingual phone set & universal acoustic model; learn a pronunciation model to map graphemes with phones for each lang.

Self-supervised Learning

Early approaches separates feature extraction, acoustic modelling (GMM), and decoding (HMM). **Later** the latter two become end-to-end (CTC & enc-dec). Handcrafted features like MFCCs based on human perceptual space might not work well for statistic modelling, also prone to information loss from raw signals. Indeed, we can add the feature extract into the end-to-end pipeline (**feature learning**).

Supervised feature learning on raw waveform

- **CNN** can deal with high-resolution and temperal aspect of raw waveform;
- **LSTM** can directly do temporal modelling, but only focus on detailed local features, while higher-level modelling also helps;
- CNNs + LSTMs + DNNs = **CLDNN**, but nevertheless not as competitive as SOTA with handcrafted features.

Unsupervised feature learning: no available transciption so need to be trained separately from the acoustic model and the downstream task. (From here on, I understand nothing except that they are kinda like word embeddings.)

Pretext task types: masked acoustic modelling / auto-regression (prediction); **Learning algorithms:**

- *Contrastive Predictive Coding* (CPC): wav2vec (masked modelling), VQ-wav2vec (with an additional quantising layer to wav2vec), wav2vec 2.0 (prediction); basically just like skipgram (word2vec), maximise mutual information of different parts the signal, but minimise the probability of co-occurence with random negative samples.
- *Student-teacher*: Bootstrap Your Own Latent (denoising), Data2vec (masked modelling), Bootstrap Predictive Coding (prediction). Minimising loss between the model's output and the teacher/target.
- *Deep clustering*: HuBERT (do K-mean clustering on waveform's MFCCs first, use the clusters to train [CNN encoder + Transformer encoder] with masked modelling, then use the trained encoders to extract features, do K-mean clustering and masked training to update the encoders, iteratively.