Lecture 27: Exam 3 Review

Mark Hasegawa-Johnson

ECE 537, Fall 2022

Outline

When, Where, What, How

- Exam 3 will be 12/16, 13:30-16:30, here.
- If you need a conflict exam or an on-line exam, tell me in advance.
- 2/3 of the exam (about 6 questions) will cover voice conversion, CTC, Transformers, and self-supervised learning.
- 1/3 of the exam (about 3 questions) will be review of topics from exam 1 and exam 2.
- You can bring two pages of handwritten notes, front and back.
- No calculators.

Topics to be covered

- Material from exam 1: loudness, voder, pitch, speech production
- Material from exam 2: DTW, LPC, HMM
- New material:
    - Voice conversion: formant synthesis, neural nets
    - CTC: RNNs, CTC labeling, CTC forward-backward, loss gradient
    - Transformer: Dot-product similarity, Attention, Masking, Positional encoding
    - Self-supervised learning: CPC, HuBERT, VQ-VAE, Transposed Convolution

Outline

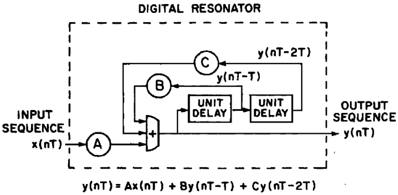
Formant-Based Speech Synthesis

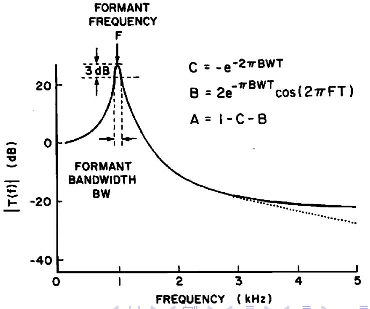
Formant synthesis filters the excitation with

$$R_k(z) = \frac{a_k}{1 - b_k z^{-1} - c_k z^{-2}},$$

The coefficients are related to the formant frequency, F_k, formant bandwidth, B_k, and sampling frequency $1/T$ by

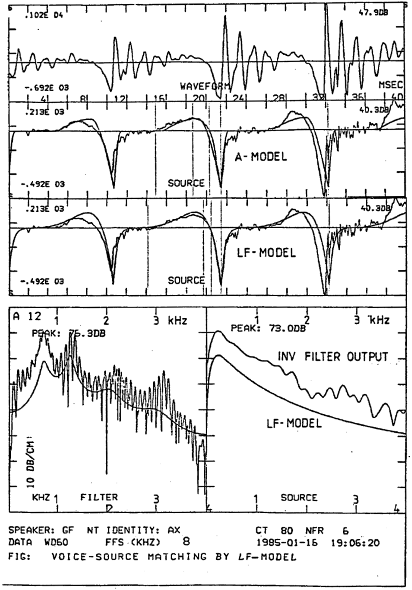
$$c_k = -e^{-2\pi B_k T}$$
$$b_k = 2e^{-\pi B_k T} \cos(2\pi F_k T)$$
$$a_k = 1 - b_k - c_k$$





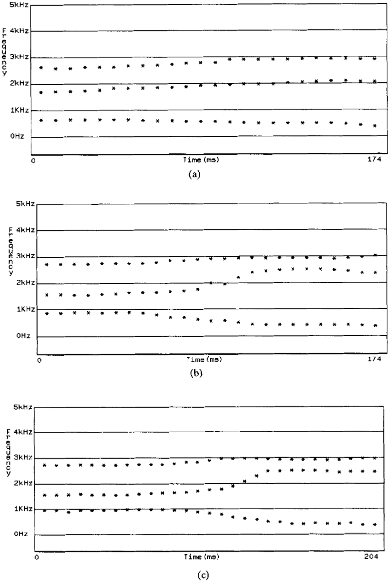
An example voice source is the LF model, which is determined by T_0 (the pitch period) plus four other parameters:

- E_e, amplitude of excitation
- t_e, time of the excitation
- time from upward-going zero-crossing, t_c, to downward-going zero-crossing, t_p
- slope of the return part, $\frac{E_e}{t_a}$



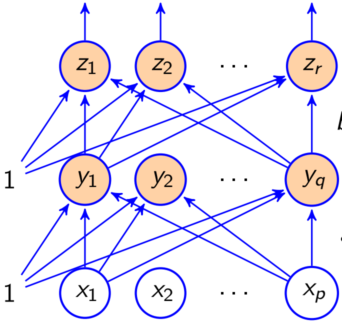
(c) http://www.speech.kth.se/qpsr

Formants can be converted from one voice to another using a neural network. Narendranath et al. (1995), Figure 4: (a) original voice, (c) conversion target, (b) conversion result.



(c) http://www.speech.kth.se/qpsr

Two-Layer Feedforward Neural Network



$$\vec{z} = h(\vec{x}, U, V)$$

$$z_\ell = g(b_\ell) \qquad \vec{z} = g(\vec{b})$$

$$b_\ell = v_{k0} + \sum_{k=1}^{q} v_{\ell k} y_k \qquad \vec{b} = V\vec{y}$$

$$y_k = f(a_k) \qquad \vec{y} = f(\vec{a})$$

$$a_k = u_{k0} + \sum_{j=1}^{p} u_{kj} x_j \qquad \vec{a} = U\vec{x}$$

$\vec{x}$ is the input vector

Gradient Descent = Local Optimization

Given an initial U, V, find $\hat{U}$, $\hat{V}$ with lower error.

$$\begin{aligned}\hat{u}_{kj} &= u_{kj} - \eta\frac{\partial\mathcal{L}}{\partial u_{kj}}\\ \hat{v}_{\ell k} &= v_{\ell k} - \eta\frac{\partial\mathcal{L}}{\partial v_{\ell k}}\end{aligned}$$

η =Learning Rate

- If η too large, gradient descent won't converge. If too small, convergence is slow. Usually we pick $\eta \approx 0.001$ and cross our fingers.
- Second-order methods like L-BFGS and Adam choose an optimal η at each step, so they're MUCH faster.

Back-Propagation = Use Chain Rule to Find the Derivatives

$$\mathcal{L} = \frac{1}{2}\sum_i \sum_\ell (z_{\ell i} - \zeta_{\ell i})^2 \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial b_{\ell i}} = \frac{2}{n}(z_{\ell i} - \zeta_{\ell i}) g'(b_{\ell i})$$

$$\frac{\partial \mathcal{L}}{\partial u_{kj}} = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{L}}{\partial a_{ki}}\right) \left(\frac{\partial a_{ki}}{\partial u_{kj}}\right) = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{L}}{\partial a_{ki}}\right) x_{ji}$$

where... $\left(\frac{\partial \mathcal{L}}{\partial a_{ki}}\right) = \sum_{\ell=1}^{r} \left(\frac{\partial \mathcal{L}_n}{\partial b_{\ell i}}\right) v_{\ell k} f'(a_{ki})$

Outline

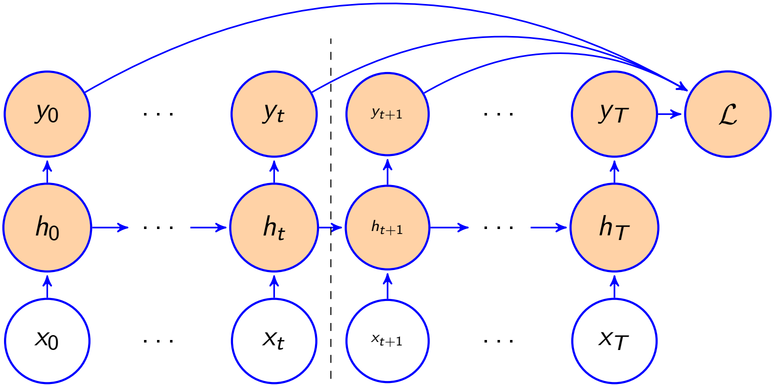
Recurrent Neural Net (RNN)

A recurrent neural net defines nonlinear recurrence of a hidden vector, h_t:

$$h_t = \sigma\left(Ux_t + Vh_{t-1}\right)$$
$$y_t = \mathrm{softmax}\left(Wh_t\right)$$

The weight matrices, U, V, and W, are chosen to minimize the loss function. For example, suppose we're using a cross-entropy loss with target sequence $\mathbf{z}$, then

$$\mathcal{L} = -\sum_{t=1}^{T} \ln y_{z_t}^{t}$$



Back-propagation through time does this:

$$\frac{d\mathcal{L}}{dh_t} = \frac{d\mathcal{L}}{dy_t}\frac{\partial y_t}{\partial h_t} + \frac{d\mathcal{L}}{dh_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}$$

Many-to-One Mapping

The key idea of CTC is that, since $U \leq T$, the mapping from $\mathbf{y}$ to $\mathbf{z}$ is many-to-one. CTC makes repeated letters possible by using a blank character, –. The many-to-one mapping now has two steps: (1) eliminate all duplicate characters, (2) THEN eliminate all blanks.

$$\mathbf{y} = [\text{f}, -, \text{e}, \text{e}, \text{d}] \longrightarrow \boxed{\mathcal{B}} \longrightarrow \mathbf{z} = [\text{f}, \text{e}, \text{d}]$$

$$\mathbf{y} = [\text{f}, \text{e}, -, \text{e}, \text{d}] \longrightarrow \boxed{\mathcal{B}} \longrightarrow \mathbf{z} = [\text{f}, \text{e}, \text{e}, \text{d}]$$

$$\mathbf{y} = [\text{f}, -, \text{f}, \text{e}, \text{d}] \longrightarrow \boxed{\mathcal{B}} \longrightarrow \mathbf{z} = [\text{f}, \text{f}, \text{e}, \text{d}]$$

Temporal Classification

The temporal classification problem is now just:

$$\begin{aligned} h(\mathbf{x}) &= \operatorname*{argmax}_{\mathbf{l}\in L^{\leq T}} p(\mathbf{l}|\mathbf{x}) \\ &= \operatorname*{argmax}_{\mathbf{l}\in L^{\leq T}} \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \\ &= \operatorname*{argmax}_{\mathbf{l}\in L^{\leq T}} \sum_{\pi\in\mathcal{B}^{-1}(\mathbf{l})} \prod_{t=1}^{T} y^t_{\pi_t} \end{aligned}$$

- $\mathbf{l} = [l_1, \ldots, l_V]$ is a label sequence of any length $V \leq T$ where $l_v \in L$.
- $\pi = [\pi_1, \ldots, \pi_T]$ is a path of length T where $\pi_t \in L \cup \{-\}$.

CTC Forward Algorithm: The Modified Label Sequence

In order to express the CTC forward algorithm, we need to define a modified label sequence, $\mathbf{l}'$. $\mathbf{l}'$ is equal to $\mathbf{l}$ with blanks inserted between every pair of letters. Thus if

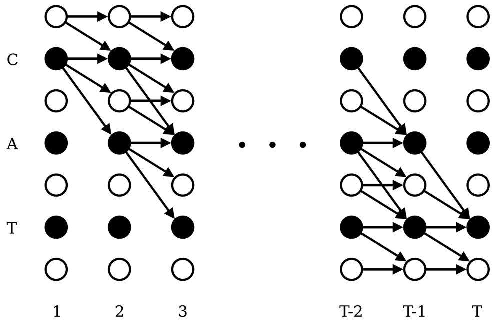
$$\mathbf{l} = [\text{f}, \text{e}, \text{d}],$$

then

$$\mathbf{l}' = [-, \text{f}, -, \text{e}, -, \text{d}, -].$$

If the length of $\mathbf{l}$ is $|\mathbf{l}|$, then the length of $\mathbf{l}'$ is $2|\mathbf{l}| + 1$.

The CTC Forward Algorithm



Graves et al., 2006, Fig. 3. (c) ICML

Repeating the same character ($\alpha_{t-1}(\mathbf{l}'_{1:s})$) or adding one more character ($\alpha_{t-1}(\mathbf{l}'_{1:s-1})$) are always possible. Adding two more characters ($\alpha_{t-1}(\mathbf{l}'_{1:s-2})$) is OK if the current character is not a blank or a repeat.

Differentiating the CTC Loss

$$\frac{d\mathcal{L}}{dy_k^\tau} = \left(\frac{-1}{p(\mathbf{z}|\mathbf{x})}\right)\left(\frac{1}{y_k^\tau}\sum_{\pi\in\mathcal{B}^{-1}(\mathbf{z}),\pi_\tau=k}\prod_{t=1}^{T} y_{\pi_t}^t\right)$$
$$= -\frac{\gamma_\tau(k)}{y_k^\tau},$$

where

$$\gamma_\tau(k) = p(\pi_\tau = k, \mathbf{z}|\mathbf{x}) = \frac{1}{y_k^\tau}\sum_{s:z'_s=k}\alpha_\tau(\mathbf{z'}_{1:s})\beta_\tau(\mathbf{z'}_{s:(2U+1)}) \qquad (1)$$

- $\beta_t(\mathbf{z'}_{s:2U+1}) = p(\mathbf{z'}_{s:(2U+1)}|\mathbf{x}_{t:T})$
- Notice that $\alpha_\tau(\mathbf{z'}_{1:s})$ and $\beta_\tau(\mathbf{z'}_{s:(2U+1)})$ both include the fact that the network is producing $z'_s = k$ at time τ. To compensate for that duplication, Eq. (1) has a $\frac{1}{y_k^\tau}$ factor.

Outline

The Cauchy-Schwartz Inequality

The Cauchy-Schwart inequality says that, for any two vectors $\vec{x} = [x_1, \ldots, x_N]^T$ and $\vec{y} = [y_1, \ldots, y_N]^T$,

$$|\vec{x}^T \vec{y}| \leq \|\vec{x}\|^2 \|\vec{y}\|^2$$

If we define the unit vectors as follows,

$$\hat{x} = \frac{\vec{x}}{\|\vec{x}\|}, \quad \hat{y} = \frac{\vec{y}}{\|\vec{y}\|},$$

then the Cauchy-Schwartz inequality says that

$$-1 \leq \hat{x}^T \hat{y} \leq 1$$

Attention

$$\text{Context Vector:} \quad c(q^i) = \sum_{j=1}^{n} \alpha_{i,j} v^j$$

$$\text{Attention:} \quad \alpha_{i,j} = \frac{\exp\left(\text{Similarity}(q^i, k^j)\right)}{\sum_{j=1}^{n} \exp\left(\text{Similarity}(q^i, k^j)\right)}$$

- The query, q (sometimes q^i), is the vector whose context we want
- The key, k (sometimes k^j), tells us whether or not v^j is useful context
- The value, v (sometimes v^j), provides the actual context

Scaled Dot-Product Attention

We assume that q and k have been transformed by some preceding neural net, so qk^T is large if and only if they should be considered similar. Therefore the similarity score is

$$e_{i,j} = \frac{1}{\sqrt{d_k}} q^i k^{j,T},$$

and the corresponding attention weight is

$$\alpha_{i,j} = \text{softmax}(e_{i,j}) = \frac{\exp(e_{i,j})}{\sum_{j=1}^{n} \exp(e_{i,j})}$$

$$\begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,n} \end{bmatrix} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Masking

If q, k and v are decoder vectors being produced autoregressively (e.g., decoder self-attention), then $c(q^i)$ can only depend on values of v^j for $j < i$:

$$c(q^i) = \sum_{j=1}^{i-1} \alpha_{i,j} v^j$$

This can be done by setting $\alpha_{i,j} = 0$ for $j \geq i$. In turn, this can be done by masking the similarity scores as follows:

$$e_{i,j} = \frac{1}{\sqrt{d_k}} q^i k^{j,T} + m_i^j,$$

where

$$m_i^j = \begin{cases} 0 & j < i \\ -\infty & j \geq i \end{cases}$$

Multi-Head Attention

$$\begin{aligned}\text{head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\\ &= \text{softmax}\left(\frac{QW_i^Q W_i^{K,T} K^T}{\sqrt{d_k}}\right) VW_i^V,\end{aligned}$$

where the weight matrices W_i^Q, W_i^K, and W_i^V, for $1 \leq i \leq h$, are learned matrices summarizing the type of context accumulated in each head. Then

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

where W^O is a final transformation that can, e.g., combine information from different heads in a learned manner.

Outline

Unsupervised Learning Algorithms

- Manifold learning, e.g., Autoencoder:

$$\mathcal{L}_{\text{MAE}} = E\left[|x - x'|\right]$$
$$\mathcal{L}_{\text{MSE}} = E\left[\|x - x'\|^2\right]$$

- Clustering
  - Classify each token to its nearest mean
  - Recompute each mean as the average of its tokens
- Self-supervised learning
  - The hyperplane is $H_n^T Z_n =$ threshold.
  - H_n is the average of all of the Z vectors on the right side of the hyperplane.

Recent Self-Supervised Learning for Speech

- **Contrastive Predictive Coding:** "Representation Learning with Contrastive Predictive Coding," Oord, Li & Vinyals, 2018

$$\mathcal{L}_{\text{CPC}} = -\sum_t \ln \frac{\exp\left(\text{Score}(x_{t+k}, c_t)\right)}{\sum_{x\in X} \exp\left(\text{Score}(x, c_t)\right)}$$

- **Autoregressive Predictive Coding:** "Generative Pre-training for Speech with Autoregressive Predictive Coding," Chung & Glass, 2020

$$\mathcal{L} = \sum_{i=1}^{N-n} |x_{i+n} - y_i|$$

- **Masked Language Modeling (HuBERT):** "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units," Hsu et al., 2021

$$\mathcal{L} = -\sum_{t\in M} \ln \frac{\exp\left(\text{Score}(Ao_t, e_c)\right)}{\sum_{c'=1}^{C} \exp\left(\text{Score}(Ao_t, e_{c'})\right)}$$

Summary: Discrete Disentangled Self-Supervised Representations

- Content is encoded using CPC, HuBERT, or VQ-VAE.
- Pitch is encoded using a VQ-VAE:

$$z_q(x) = \operatorname*{argmin}_j \|z_e(x) - e_j\|_2$$

$$\nabla_{z_e(x)}\mathcal{L} \approx \nabla_{z_q(x)}\mathcal{L}$$

- Speaker ID is encoded using a neural net trained to perform speaker discrimination.

Summary: Speech Resynthesis

- Speech resynthesis uses transposed convolution:

$$[h_1, \ldots, h_{2L+1}] = [0, z_1, 0, z_2, 0, \ldots, 0, z_L, 0]$$

$$y_i = \sum_{j=-D}^{D} K_j h_{i+j}$$

- Reconstruction minimizes absolute error of the mel-frequency spectrogram:

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^{L} \|\phi(\mathbf{x}) - \phi(\hat{\mathbf{x}})\|_1,$$

- GAN is used to avoid "regression to the mean," to make sure speech sounds good:

$$\mathcal{L}_G(D, G) = \sum_{j=1}^{J} [\mathcal{L}_{\text{adv}}(G, D_j) + \lambda_{fm}\mathcal{L}_{fm}(G, D_j)] + \lambda_r \mathcal{L}_{\text{recon}}(G)$$

Outline

Topics to be covered

- Material from exam 1: loudness, voder, pitch, speech production
- Material from exam 2: DTW, LPC, HMM
- New material:
  - Voice conversion: formant synthesis, neural nets
  - CTC: RNNs, CTC labeling, CTC forward-backward, loss gradient
  - Transformer: Dot-product similarity, Attention, Masking, Positional encoding
  - Self-supervised learning: CPC, HuBERT, VQ-VAE, Transposed Convolution