Lecture 25: Self-Supervised Learning for Speech and Language

Mark Hasegawa-Johnson

ECE 537, Fall 2022

Outline

Unsupervised Learning

- Manifold learning: e.g., current sample, or current spectrum, is predictable from previous samples/spectra.
- Clustering: If differences between classes are big enough, clustering can find natural classes.
- Self-supervised learning: if differences from noise tend in one direction, the self-supervised learner can learn that direction.

Outline

The N-Gram Language Model

Claude Shannon ("A Mathematical Theory of Communication," 1948) proposed representing the probability of a sentence, $w = [w_1, \ldots, w_L]$ as the product of its N-gram probabilities:

$$P(w) = \prod_{i=1}^{L} P(w_i | w_{i-1}, \ldots, w_1) \approx \prod_{i=1}^{L} P(w_i | w_{i-1}, \ldots, w_{i-N+1})$$

He proposed storing these probabilities in a lookup table, and using them to verify a transmitted message.

The N-Gram Language Model

Shannon proposed that pseudo-sentences can be generated by sampling from these N-gram probability tables:

- Sentence generated by sampling from unigram (1-gram) probability tables:

  REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

- Sentence generated by sampling from unigram (2-gram) probability tables:

  THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Autoregressive Neural Language Model

An autoregressive neural language model expresses each word as a learned vector. The vector corresponding to word w is $x(w) = [x_1(w), \ldots, x_d(w)]$, whose elements $x_i(w)$ are learned parameters. The parameters $x(w)$, ϕ and θ are then learned in order to minimize the cross-entropy loss:

$$\mathcal{L} = -\sum_{i=1}^{L} \ln P(w_i | w_{i-1}, \ldots, w_1)$$

which is computed using an LSTM or Transformer as:

$$P(w_i | w_{i-1}, \ldots, w_1) = \frac{\exp\left(\text{Score}(w_i, h_i; \phi)\right)}{\sum_w \exp\left(\text{Score}(w, h_i; \phi)\right)}$$

$$h_i = f\left(x(w_{i-1}), \ldots, x(w_1); \theta\right)$$

Autoregressive Neural Language Model

Autoregressive language modeling performance varies dramatically depending on the number of parameters of the neural net.

- GPT-2 (1.5 billion params): sometimes "In practice, he was a very conservative figure, but he was also a very conservative figure in the sense that he was a very conservative figure in the sense that he was a very conservative figure. . . "
- GPT-3 (175 billion params): `https://maraoz.com/2020/07/18/openai-gpt3/`: "So there are lots of posts for GPT-3 to study and learn from. The forum also has many people I don't like. I expect them to be disproportionately excited by the possibility of having a new poster that appears to be intelligent and relevant. I've been following the forum for years. There are many posts I know the answers to, so I could provide a quick response and measure how well GPT-3 does with comments similar to those I make."

Autoregressive Predictive Coding (Chung and Glass, 2021)

- In an autoregressive language model, the word representations, $x(w)$, and the LSTM or Transformer state vectors, h_i, are summaries of the context-independent and context-dependent meaning of a word, respectively.
- Chung and Glass (2020) proposed "Autoregressive Predictive Coding" to compute similar representations for speech.

Autoregressive Predictive Coding (Chung and Glass, 2021)

APC trains a transformer so that its last layer, h_L, can be used to predict the input vector n steps later:

$$
\begin{aligned}
h_0 &= W_{in}x + P(x) \\
h_l &= \text{Transformer}(h_{l-1}) \\
y &= W_{out}h_L \\
\mathcal{L} &= \sum_{i=1}^{N-n} |x_{i+n} - y_i|
\end{aligned}
$$

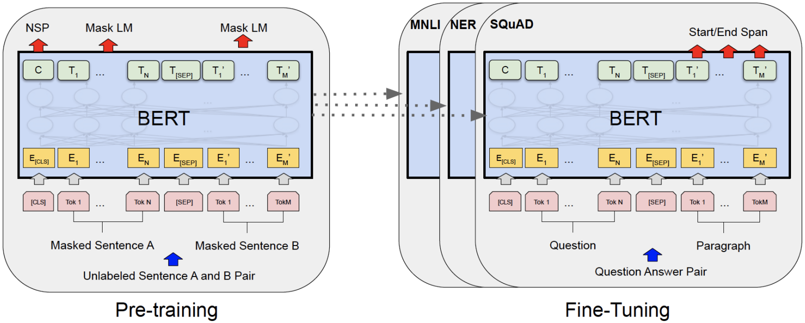
How is it used?

- The neural net is first **pre-trained** using the autoregressive criterion on the previous page, using only the inputs (the audio).
- The system is then connected to a few extra layers of neural net, in order to perform some downstream task. Either:
  - The pre-trained network may be **frozen**, in which case only the extra layers are trained using downstream labels, or
  - The pre-trained network is **fine-tuned** (trained in order to optimize performance on the downstream task.
- These ideas (frozen vs. fine-tuned) were laid out in BERT, so let's look at BERT.

Outline

The BERT ecosystem: Pre-training and Fine-tuning

BERT stands for Bidirectional Encoder Representations from Transformers. The idea is to pre-train an encoder, which can then be fine-tuned for downstream tasks:



Devlin et al., 2018

Pre-Training: Masked Language Modeling

The pre-training criterion is masked language modeling.

$$m_i = \begin{cases} 1 & \text{word } i \text{ is visible (probability: 85\%)} \\ 0 & \text{word } i \text{ is masked (probability: 15\%)} \end{cases}$$

Then, from input vectors $e_1, \ldots, e_n$ and masked-word replacement noises $v_1, \ldots, v_n$, the transformer computes outputs

$$t_i = \text{Transformer}\left(m_1 e_1 + (1 - m_1)v_1, \ldots, m_n e_n + (1 - m_n)v_n\right)$$

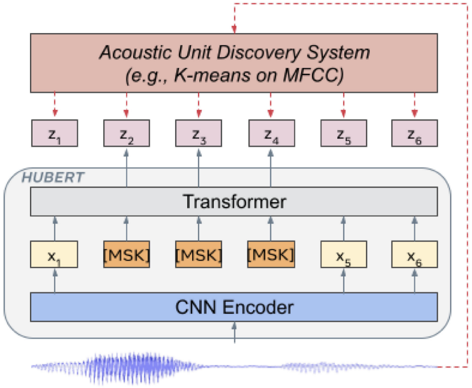
The loss function measures the ability of the model to predict the masked words only.

$$\mathcal{L} = -\sum_{i=1}^{n} (1 - m_i) \ln \left(\frac{\exp\left(\text{Score}(At_i, e_i)\right)}{\sum_E \exp\left(\text{Score}(At_i, e)\right)} \right),$$

where A is the matrix of softmax weights.

Hidden Unit BERT (HuBERT)

HuBERT uses the masked language modeling idea, but instead of words, it predict units such as K-means clusters of mel frequency cepstral coefficients (MFCC).



Hsu et al., 2021, Fig. 1

HuBERT notation

- $X = [x_1, \ldots, x_T]$ are the frames of a speech utterance, $x_t \in \Re^d$,
- $h(X) = Z = [z_1, \ldots, z_T]$ are the hidden units computed from X, $z_t \in \{1, \ldots, C\}$ where C is the number of categories.
- $M \subset \{1, \ldots, T\}$ are the frames to be masked.
- $\tilde{X}$ is the masked sequence, i.e., masked frames are replaced by a "MASK" token.

HuBERT: Masked language modeling loss function

The loss function is

$$L(f; X, M, Z) = -\sum_{t \in M} \ln p(z_t | \tilde{X}, t),$$

where the probability is computed using a softmax:

$$p(c|\tilde{X}, t) = \frac{\exp\left(\text{Score}(Ao_t, e_c)\right)}{\sum_{c'=1}^{C} \exp\left(\text{Score}(Ao_t, e_{c'})\right)},$$

where A is the softmax weight matrix, o_t is the transformer output, and e_c is an embedding learned for hidden unit c.

HuBERT: Fine tuning

- The transformer is trained so that its outputs, o_t, best predict the hidden units, z_t.
- The hidden units are then discarded; the softmax weights A and the transformer o_t are trained to minimize

$$L_{\text{CTC}}(Y|X) + w_1 L_{\text{LM}}(Y) + w_2 |Y|$$

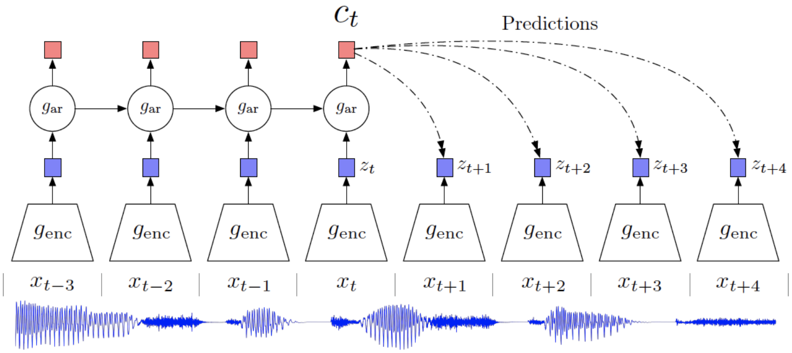
Why it works

- The transformer is trained so that it accumulates information from context (o_t) in order to optimally predict a quantized unit.
- The quantized units are kind of like phonemes, so a Transformer that predicts them well is also good at predicting phonemes.
- The more phoneme-like the hidden units are, the more effective the pre-training works. The best procedure is:
  1. K-means cluster the MFCC; pre-train HuBERT to predict that.
  2. K-means cluster the first-round HuBERT vectors; pre-train another HuBERT.
  3. K-means cluster the second-round HuBERT vectors; pre-train another HuBERT.
  4. Fine tune as an ASR.

Outline

Contrastive Predictive Coding (CPC)

Contrastive predictive coding was originally a form of autoregressive prediction:



Oord et al., 2018, Fig. 1

The "Contrastive" in CPC

The key innovation in CPC is the loss term. A normal autoregressive language model trains c_t in order to minimize

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

where V is the entire vocabulary (all possible words). By contrast, CPC trains c_t in order to minimize

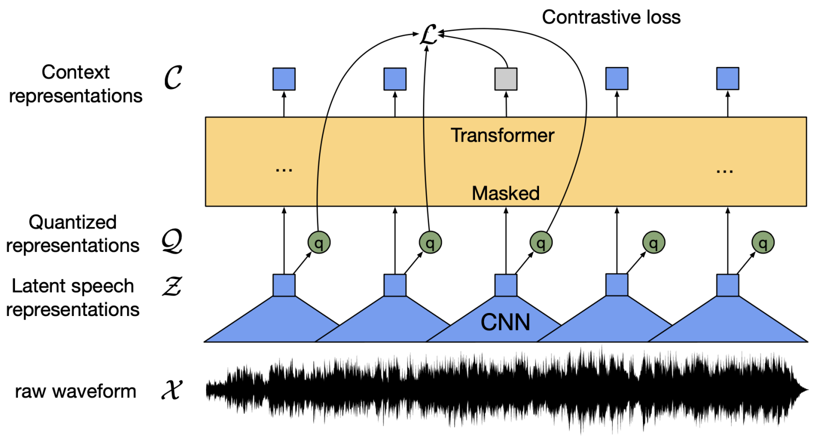
$$\mathcal{L}_{\text{APC}} = -\sum_t \ln \frac{\exp\left(\text{Score}(x_{t+k}, c_t)\right)}{\sum_{x\in X} \exp\left(\text{Score}(x, c_t)\right)}$$

where $X = \{x_1, \ldots, x_N\}$ is a set of N randomly chosen negative examples.

CPC can predict either continuous or discrete targets

- The language modeling criteria (autoregressive and MLM) require a discrete set of words. CPC, by contrast, can predict continuous vectors. The set of negative examples, $X = \{x_1, \ldots, x_N\}$, can be either continuous-valued or discrete; the only difference is the way $\text{Score}(x_{t+k}, c_t)$ is computed.
- The original CPC paper (Oord, 2018) used continuous vectors for speech and images, and discrete words for NLP applications.
- Wav2vec 2.0 tested both continuous and discrete speech targets, and found that discrete targets worked better, possibly because they force the transformer to learn to predict something phoneme-like.

wav2vec 2.0 (Baevski et al., 2020)


Contrastive loss
$\mathcal{L}$
Context representations $\mathcal{C}$
Transformer
...
...
Masked
Quantized representations $\mathcal{Q}$
q
Latent speech representations $\mathcal{Z}$
CNN
raw waveform $\mathcal{X}$

Baevski et al., 2020

wav2vec 2.0: Loss Function

$$\mathcal{L} = -\ln \frac{\exp\left(\text{Score}(c_t, q_t)\right)}{\sum_{q \in Q_t} \exp\left(\text{Score}(c_t, q_t)\right)},$$

where q_t is quantized (codebook index), but Q_t is a set of negative examples.

wav2vec 2.0: Other Details

Unlike HuBERT, the quantizer functions $q_t = Q(z_t)$ is learned along with the other parameters (instead of being a K-means learned offline). It's therefore necessary to prevent the trivial failure mode in which all inputs are mapped to the same codevector. To prevent this, wav2vec 2.0 adds a **diversity loss**, equal to the negative entropy of the codevector indices $q_1, \ldots, q_T$. If they are all the same, the entropy is zero; if they are all different, the entropy is maximized (and loss is minimized):

$$\mathcal{L}_d = -\frac{1}{V} H(p) = \frac{1}{V} \sum_{v=1}^{V} p_v \log p_v$$

$$p_v = \frac{1}{T} \sum_{t=1}^{T} (1 \text{ iff } q_t = v)$$

Outline

A Brief History of Self-Supervised Learning for NLP

- **Autoregressive Neural Language Modeling:** Many papers, e.g., "LSTM neural networks for language modeling," Sundermeyer, Schlüter & Ney, 2012

$$\mathcal{L} = -\sum_{i=1}^{L} \ln P(w_i | w_{i-1}, \ldots, w_1)$$

- **Masked Language Modeling (BERT):** "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Devlin, Chang, Lee & Toutanova, 2018

$$\mathcal{L} = -\sum_{i=1}^{n} (1 - m_i) \ln \left(\frac{\exp\left(\text{Score}(At_i, e_i)\right)}{\sum_E \exp\left(\text{Score}(At_i, e)\right)} \right),$$

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

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

A Brief History of Self-Supervised Learning for NLP

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

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

- **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

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