38

NATURAL LANGUAGE PROCESSING I: FOUNDATION

38.1 Introduction

38.1.1 NLP tasks

Human language is a communication tool that consists of complex yet organic combinations of different levels of linguistic building blocks such as characters, words, sentences, etc. Natural language processing (NLP) is a collection of methodology that aims to process and understand human language via machines. NLP is a field at the interface of computer science, artificial intelligence, and linguistics.

Recently, increasing availability of text data from web and computational resources has accelerated drastic changes in NLP methodologies and its tasks and application areas [Figure 38.1.1].

There are a broad range NLP tasks, either arising from real-world application needs or are designed by human experts to test model's capabilities of understanding human language. In the following, we list major NLP tasks and denote their difficulty levels ($\star$ easy; $\star\star$ medium; $\star\star\star$ difficult). Successful accomplishment of these tasks require different levels of natural language understanding, from word meaning, sentence structure to high-level logical reasoning and application of common knowledge.

Topic modeling ($\star$): This is the task of uncovering the topical structure of a large collection of documents. Topic modeling is a common text-mining tool and is used in a wide range of domains, from literature to bioinformatics.

Text classification ($\star$): This is the task of bucketing the text into a known set of categories (e.g., news, sports, politics, etc.) based on its content. Text classification is one of the most popular tasks in NLP. One example is spam email identification.

Sentiment analysis ($\star$): This is the task of understanding social sentiment of an event, a product, a movie, or a service. Sentiment analysis is also considered as a subarea of text classification.

Language modeling ($\star\star$): This is the task of predicting what the next word in a sentence will be based on the history of previous words. The goal of this task is to learn the probability of a sequence of words appearing in a given language. Language modeling is useful for building solutions for a wide variety of problems, such as speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction.

Information extraction ($\star\star$): This is the task of extracting relevant information from text, such as calendar events, address, names from emails or web post. In Amazon product review, NLP can be used to extract relevant information from product descriptions to understanding user reviews.

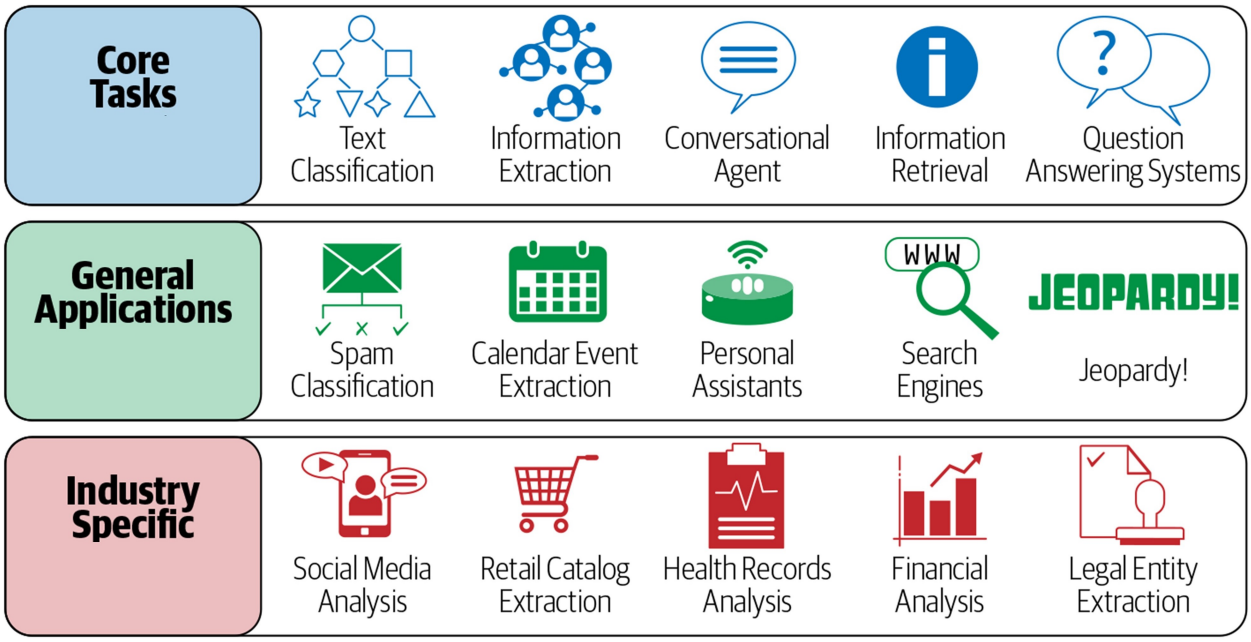


Figure 38.1.1: Modern NLP tasks and their application areas. Source from [1].

Information retrieval ($\star\star$): This is the task of finding documents relevant to a user query from a large collection. Applications like Google Search are well-known use cases of information retrieval.

Conversational agent ($\star\star\star$): This is the task of building dialogue systems that can converse in human languages. A concrete example is that voice-based assistants such as Google Assistant and Amazon Alexa are capable of interacting with the user, understanding user commands, and follow instructions accordingly.

Text summarization ($\star\star\star$): This task aims to create short summaries of longer documents while retaining the core content and preserving the overall meaning of the text.

Question answering $\star\star\star$: This is the task of building a system that can automatically answer questions posed in natural language.

Machine translation $\star\star\star$: This is the task of converting a piece of text from one language to another. Tools like Google Translate are common applications of this task.

38.1.2 Challenges

Natural languages, either in spoken or in written forms, are long-established communication tools between human beings. Unlike programming languages which have strict syntactic rules on human-to-computer instructions, natural languages emerge from

formal or casual communications between human beings and only have a limited set of formal rules to follow.

Natural languages can be extremely flexible and sometimes ambiguous, yet still be understood by human beings. Consider the following examples[1].

- Sentences consisting of completely different tokens can express the same meaning. For example, *What is your age?* vs. *How old are you?*
- Sentences consisting of similarly tokens can have vastly different meanings. For example *You know him better than I.* vs. *You know him better than me.*
- There could be ambiguities. For example, *I hit the man with a stick* (Used a stick to hit the man) vs. *I hit the man with a stick* (I hit the man who was holding a stick).
- Punctuation can change the meaning significantly. For example, *Woman, without her man, is helpless.* vs. *Woman! Without her, man is helpless!*

Besides the **ambiguity** challenge, other challenges include **Common knowledge**: A key aspect of any human language is **common knowledge**. In vast amount of textual content, it is assumed that these common knowledge and facts are known, hence they're directly used to convey meaning without being explicitly mentioned. For example, consider two grammatically valid sentences: *a blue car* and *a blue horse*. We all know that the first sentence is unlikely to happen, while the second one is very possible.

Creativity: Language is not just rule driven; there is also a creative aspect to it. Various styles, dialects, genres, and variations are used in any language. Therefore it is necessary to make machine adaptable to new changes in the language.

Generalization across languages: For most languages in the world, there is no direct mapping between the vocabularies of any two languages. This makes porting an NLP solution from one language to another hard.

38.1.3 Approaches

Similar to other early AI systems, early attempts at designing NLP systems were based on building rules for the task at hand. This required that the developers had some expertise in the domain to formulate rules that could be incorporated into a program. Such systems also required resources like dictionaries and thesauruses. An example of designing rules to solve an NLP problem using such resources is lexicon-based sentiment analysis. It uses counts of positive and negative words in the text to deduce the sentiment of the text.

Machine learning techniques are applied to textual data just as they're used on other forms of data, such as images, speech, and structured data. Supervised machine learn-

1 http://www.city-data.com/forum/writing/1115620-two-sentences-have-same-words-but-2.html

ing techniques such as classification and regression methods are heavily used for various NLP tasks. As an example, an NLP classification task would be to classify news articles into a set of news topics like sports or politics. On the other hand, regression techniques, which give a numeric prediction, can be used to estimate the price of a stock based on processing the social media discussion about that stock. Similarly, unsupervised clustering algorithms can be used to club together text documents. Any machine learning approach for NLP, supervised or unsupervised, can be described as consisting of three common steps: extracting features from text, using the feature representation to learn a model, and evaluating and improving the model.

In the last few years, we have seen a huge surge in using neural networks to deal with complex, unstructured data. Language is inherently complex and unstructured. Therefore, we need models with better representation and learning capability to understand and solve language tasks.

However, deep learning methods have also met practical challenges such as data scarcity, domain mismatch even with pretrained models, lack of interpretibility, prohibitive cost for training large models. Also, deep learning methods have not addressed the fundamental limitation of common knowledge and logical reasoning like human beings.

In this chapter and the next chapter, we will survey fundamental principles and methodology in NLP and discuss different modeling approach specific to major NLP tasks.

38.2 Word embeddings

38.2.1 Overview

Human language is structured through a complex combinations of different levels of linguistic building blocks such as characters, words, sentences, etc. Among different levels of these building blocks, words and its subunits (i.e., morphemes[2]) are the most basic ones.

Many machine learning and deep learning approaches in natural language processing (NLP) requires explicit or implicit construction of word-level or subword level representationss. These word-level representations are used to construct representations of larger linguistic units (e.g., sentences, context, and knowledge), which are used to solve NLP tasks, ranging from simple ones such as sentiment analysis and search completion to complex ones such as text summerization, writing, question-answering, etc. Modern NLP tasks heavily hinge on the quality of word embedding and pre-trained language models that produce context-dependent or task dependent word representations.

NLP tasks are faced with text data consisting of tokens from a large vocabulary ($> 10^5 - 10^6$). In sentiment analysis, we need to represent text data by numeric values such that computers can understand. One naive way to represent the feature of a word is the **one-hot word vector**, whose length of the typical size of the vocabulary.

Example 38.2.1. Consider a vocabulary of size V, the one hot encodings for selected of words are represented as follows.

$$\begin{aligned} \text{Rome} &= \underbrace{[1,0,0,0,0,0,\ldots,0]}_{length\ V} \\ \text{Paris} &= [0,1,0,0,0,0,\ldots,0] \\ \text{America} &= [0,0,1,0,0,0,\ldots,0] \\ \text{Canada} &= [0,0,0,1,0,0,\ldots,0] \end{aligned}$$

One-hot sparse representation treats each word as an independent atomic unit that has equal distance to all other words. Such encoding does not capture the relations among words (i.e., meanings, lexical semantic) and lose its meaning inside a sentence. For example, consider three words *run, horse, and cloth*. Although *run* and *horse* tend to be more relevant to each other than *horse* and *ship*, they have same Euclidean distance.

2 A morpheme is the smallest unit of language that has a meaning. Not all morphemes are words, but all prefixes and suffixes are morphemes. For example, in the word *multimedia*, *multi-* is not a word but a prefix that changes the meaning when put together with *media*. *Multi-* is a morpheme.

Additional disadvantage include its poor scalability, that is, its representation size grows with the size of vocabulary. As such one hot encodings are thus not considered as good features for advanced natural language processing tasks that draw on interactions and semantics among words, such as language modeling, machine learning. But there are exceptions when the vocabulary associated with a task is indeed quite small and words in the vocabulary are largely irrelevant to each other.

A much better alternative is to represent each word vector by a dense vector, whose dimensionality D typically ranges from 25 to 1,000.

Example 38.2.2. Dense vector representation for some words could be

$$\begin{aligned} \text{Rome} &= \underbrace{[0.1, 0.3, -0.2, \ldots, 0]}_{\textit{length } D} \\ \text{Paris} &= [-0.6, 0.5, 0.2, \ldots, 0.3] \\ \text{America} &= [0.3, 0.2, -0.3, \ldots, 0.2] \\ \text{Canada} &= [0.15, 0.2, 0.4, \ldots, 0.1]. \end{aligned}$$

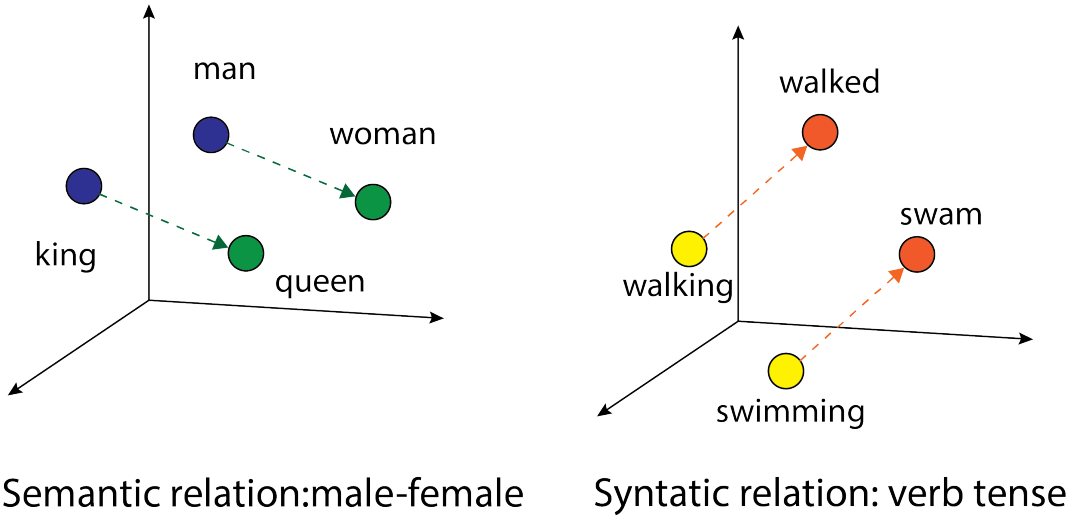


Figure 38.2.1: (a) Embedding layer maps large, sparse one-hot vectors to short, dense vectors. (b) Example of low dimensional embeddings that capture semantic meanings.

In a dense vector representation, every component in the vector can contribute to enrich the concept and semantic meaning associated with the word. A linguistic

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter

(a) Examples of five types of semantic relationships.

Type of relationship	Word Pair 1		Word Pair 2	
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

(b) Examples of nine types of syntactic relationships.

phenomenon is that words that occur in similar contexts have similar meanings. Now the similarity or dissimilarity among words can be captured via distance in the vector space. A basic test on the ability to capture semantic and syntactic information is to be able to answer questions like

- Semantic questions like "Being is to China as Berlin is to [·]".
- Syntactic questions like "dance is to dancing as run is to [·]".

Ideally, we would like the word embeddings distributed in the vector space in certain way that capture semantic and syntactic relations and facilitates answering these questions [Figure 38.2.1].

There are different ways to obtain word embeddings. In the following sections, we will discuss methods that utilize classical singular value decomposition as well as modern neural network.

38.2.2 SVD based word embeddings

Here we introduce a way to obtain low-dimensional representation of a word vector that capture the semantic and syntactic relation between words by performing SVD on a matrix constructed on a large corpus. The matrix used to perform SVD can be a **co-occurrence matrix** or it can be a **document-term** matrix, which describes the occurrences

count	I	love	like	NLP	math
I	0	1	1	0	0
love	1	0	0	1	0
like	1	0	0	0	1
NLP	0	0	1	0	0
math	0	1	1	0	0

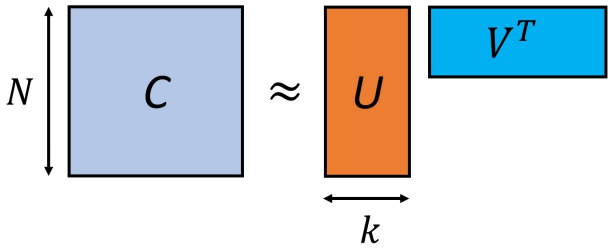


Figure 38.2.2: (left) Example of co-occurrence matrix constructed from corpus "I love math" and "I like NLP". The context window size of 2. (right) We can obtain lower-dimensional word embeddings from SVD truncated factorization of the co-occurrence matrix. Such low-dimensional embeddings captures important semantic and syntactic information in the co-occurrence matrix.

of terms in documents. When the matrix is the document-term matrix, this method is also known as **latent semantic analysis** (**LSA**)[2].

Co-occurrence matrix is a big matrix whose entry encode the frequency of a pair of words occurring together within a fixed length context window. More formally, let M be a co-occurrence matrix, and we have

$$M_{ij} = \frac{\#(w_i, w_j)/n_{pair}}{\#(w_i)/n_{words} \cdot \#(w_j)/n_{words}}$$

where $\#(w_i, w_j)$ is the number of co-occurrence of words w_i and w_j within a context window, n_{pair} is the total number pairs, n_{words} is the total number of words.

Another popular matrix to capture the co-occurrence information is the the **pointwise mutual information (PMI)** [3]. PMI entry for a word pair is defined as the probability of their co-occurrence divided by the probabilities of them appearing individually,

$$M_{ij}^{PMI} = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)} \approx \log M_{ij}.$$

The co-occurrence information captures to some extent both semantic and syntactic information. For example, terms tend to appear together either because they have related meanings (a semantic relationship, e.g., *write* and *book*) or because the grammar rule specifies so (a syntactic relation, e.g., verbs and *to*).

By using truncated SVD to decompose the co-occurrence matrix, we obtain the low-dimensional word vectors that preserve the co-occurrence information, or the semantic and syntactic relation implied by the co-occurrence information. For example, in the low-dimensional representation, apple and pear are expected to be closer (in terms of Euclidean distance of the embedding vector) than apple and dog.

More formally, via truncated SVD, we have factorization

$$M \approx UV^T$$

where $M \in R^{N \times N}$, N is the size of the one-hot vector, $U, V \in \mathbb{R}^{N \times k}, k << N$. Columns of U are the basis vector in latent word space. Each row in V is the low dimensional representation of a word in the latent word space.

The word embeddings derived from the co-occurrence matrix preserves semantic information within a relative local context window. For words that do not appear frequently within a context window but actually share semantic links, the word embeddings might miss the link. This shortcoming can be overcome by performing a SVD on a document-term matrix. The document-term matrix is a sparse matrix whose rows correspond to terms and whose columns correspond to documents. The typical entry is the tf-idf (term frequency–inverse document frequency), whose value is proportional to frequency of the terms appear in each document, where common terms are downweighted to de-emphasize their relative importance.

A truncated SVD produces **document vectors** and **term vectors** (i.e., word embeddings). In constructing the document-term matrix, documents are just cohesive paragraphs covering one or multiple closely related topics. Words appear in a document therefore share certain semantic links. Overall, the decomposition results can be used to measure word-word, word-document and document-document relations. For example, document vector can also be used to measure similarities between documents.

38.2.3 Word2Vec

38.2.3.1 *The model*

In subsection 38.2.2, we introduce a SVD based matrix decomposition method to map one-hot word vector to semantic meaning preserving dense word vector. This section, we introduce a neural networ based method. The two classical methods, called continuous bags of words (**CBOW**)[4] and **Skip-gram**[5]. Both methods employ a three-layer neural networks [Figure 38.2.3], taking a one-hot vector as input and predict the probability of its nearby words. In CBOW, the inputs are surrounding words within a context window of size c and the goal is to predict the central word (same as multi-class classification problems); in Skip-gram, the input is the single central word and the goal is to predict its surrounding words within a context window.

Denote a sequence of words $w_1, w_2, ..., w_T$ (represented as integer indices) in a text, the objective of a Skip-gram model is to maximize the likelihood of observing the occurrence of its surrounding words within a context window of size c, given by

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(w_{t+j}|w_t\right)$$

where we have assumed conditional independence given word w_t.

In the neural networks of Skip-gram and CBOW, we use Softmax function after the output layer to produce classification probability for V classes, where V is the size of the vocabulary. Note that the input layer has a weight matrix $W \in \mathbb{R}^{V\times D}$ that performs look-up and converts a word integer to a dense vector of D dimension; the output layer has a weight matrix $W' \in \mathbb{R}^{D\times V}$. The classification probability is given by

$$p(w_k|w_j) = \frac{\exp(v'_k \cdot v_j)}{\sum_{i=1}^{V}\exp(v'_i \cdot v_j)},$$

where v_i is the column i of the input matrix W, and v'_i is the row i in the output matrix W'.

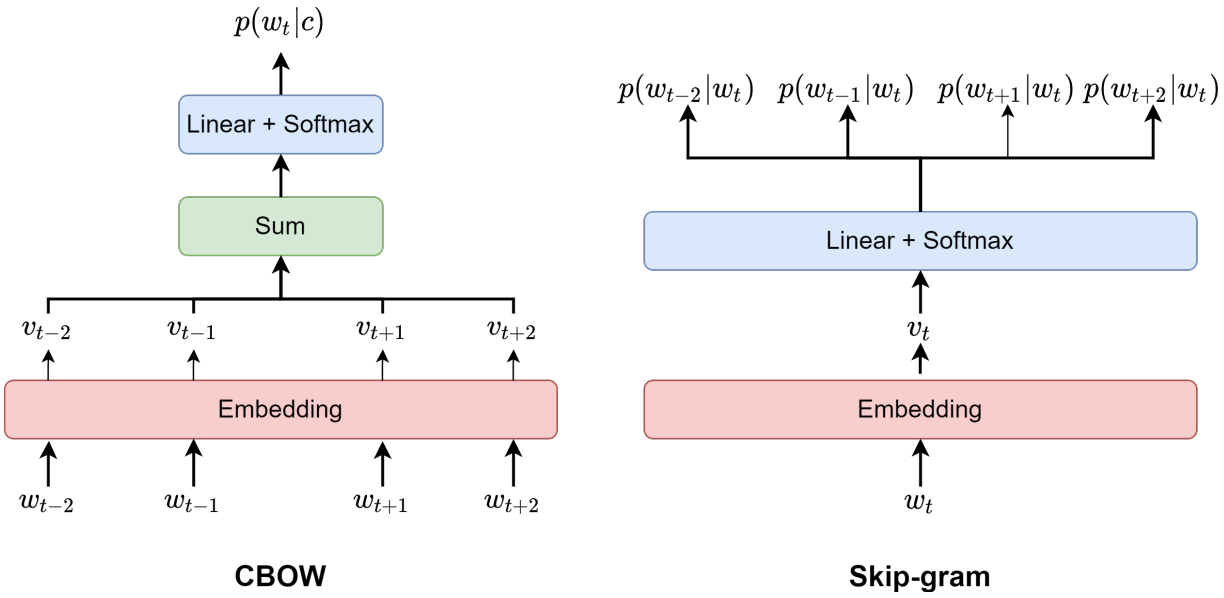


Figure 38.2.3: (a) The CBOW architecture that predicts the central word given its surrounding context words. (b) The Skip-gram architecture that predicts surrounding words given the central word. The embedding layer is represented by a $V \times D$ weight matrix that performs look-up for each word token integer index, where V is the vocabulary size and D is the dimensionality of the dense vector. The linear output layer is also represented by a $V \times D$ weight matrix that is used to compute the logit for each token label as sort of classification over the vocabulary.

Definition 38.2.1 (Skip-gram and CBOW optimization problem). *The neural network weights $\{v_i, v_i'\}$ of the Skip-gram model are optimized to maximize the observation of a text consisting of words $w_1, w_2, ..., w_T$, which can then be written by*

$$\begin{aligned}
&\max_{v,v'} \sum_{t=1}^{T} \sum_{-c\le j\le c, j\neq 0} \ln p\left(w_{t+j}|w_t\right) \\
=&\max_{v,v'} \sum_{t=1}^{T} \sum_{-c\le j\le c, j\neq 0} \ln \frac{\exp\left(v'_{t+j}\cdot v_{t)}\right)}{\sum_{w\in V}\exp\left(v'_w\cdot v_t\right)} \\
=&\max_{v,v'} \sum_{t=1}^{T} \sum_{-c\le j\le c, j\neq 0} \left[v'_{t+j}\cdot v_t - \ln \sum_{w\in V}\exp\left(v'_w\cdot v_t\right)\right]
\end{aligned}$$

In the CBOW model, the optimization problem becomes

$$\begin{aligned}
&\max_{v,v'} \sum_{t=1}^{T} \ln p\left(w_t|w_{t-c},...,w_{t+c}\right) \\
=&\max_{v,v'} \sum_{t=1}^{T} \ln \frac{\exp(v'_t\cdot \sum_{-c\le j\le c, j\neq 0} v_j)}{\sum_{i=1}^{V}\exp(v'_i\cdot \sum_{-c\le j\le c, j\neq 0} v_j)}
\end{aligned}$$

In the original Skip-gram and the CBOW model, each word will have two embeddings, v_i and v_i' in the input matrix W and the output matrix W', respectively. Notable, the embedding v_i' corresponds to the dense word vector that produces one-hot probability vector in the output. For these two embeddings, we can use one of them, a mixed version of them, and a concatenated one. It is only found that tying the two weight matrices together can lead to performance [6, 7].

With the trained embeddings for each word, we can assemble then into a matrix of size $D \times V$, which is also called an Embedding layer. In applications, the one-hot word vector is fed into the embedding layer and produce the corresponding dense word vectors. From the computational perspective, we do not need to perform matrix multiplication; instead, we can view the Embedding layer as a dictionary that maps integer indices of the word to dense vectors.

In Skip-gram, the weight associated with each word receives adjustment signal (via gradient descent) from its surrounding context words. In CBOW, a central word provides signal to optimize the weights of its multiple surrounding words. Skip-gram is more computational expensive than CBOW as the Skip-gram model has to make predictions of size $O(cV)$ while CBOW makes prediction on the scale of $O(V)$. Further, because of the averaging effect from input layer to hidden layer in CBOW, CBOW is less competent in calculating effective word embedding for rare words than Skip-gram.

38.2.3.2 *Optimization I: negative sampling*

Solving Skip-gram optimization [Definition 38.2.1] requires summing over the probabilities of every incorrect vocabulary word in the denominator ($\sum_{w \in V} \exp\left(v'_w \cdot v_t\right)$). In a practical scenario, the dimensionality of the word embedding D could be ~ 500 and the size of the vocabulary $|V|$ could be $\sim 10,000$. Naively running gradient descent on the optimization would lead to update millions of network weight parameters ($O(D|V|)$). Computing the summation is therefore costly. One idea to reduce the cost is: just summing over probabilities of a few ($k = 5 - 20$) high-frequent incorrect words, rather than summing over the probabilities of every incorrect word. These chosen non-target words are called **negative samples**. Note that negative sampling will result in incorrect normalization since we are not summing over the vast majority of the vocabulary. In practice, this approximation that turns out to work well. Further, the computational cost to update weight parameters goes from $O(D \cdot |V|)$ to $O(D \cdot k)$.

In the optimization, gradient descent steps tend to pull embeddings of frequently co-occurring words closer (i.e., to make $v_i \cdot v_j$ have a larger value) while push embeddings of rarely co-occurring words away (i.e., to make $v_i \cdot v_j$ have a smaller value). It is justified to pick more commonly seen words with larger probability as negative samples. In the study, the negative samples w are empirically sampled from

$$P_n(w) \approx f(w)^{3/4},$$

where $f(w)$ is the frequency of word w in the training corpus. This distribution is found to significantly outperform uni-gram or uniform distribution [5].

Finally, we have the modified optimization for Skip-gram model given by

$$\arg\max_{v,v'} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \neq 0} \left[v'_{t+j} \cdot v_t - \log \sum_{m=1}^{k} \exp\left(v'_{w_m} \cdot v_t\right) \right]$$

where $w_m \sim P_n(w), m = 1, ..., k$ are negative samples.

38.2.3.3 *Optimization II: subsampling of frequent words*

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., *in*, *the*, and *a*). These words usually provide less information value than the rare words. For example, while the Skip-gram model gains more information from observing the co-occurrences of *China* and *Beijing*, it gains much less information from observing the frequent co-occurrences of *France* and *the*, as nearly every word co-occurs frequently within a sentence with *the*.

We like to assign lower weights to these frequent words. This is achieved via a simple subsampling approach: each word w_i in the training set is discarded with probability computed by the formula

$$P\left(w_i\right) \approx 1 - \sqrt{\frac{t}{f\left(w_i\right)}}$$

where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold, typically around 10^{-5}.

38.2.3.4 *Visualization*

We can visualize the word embedding space by projecting onto a 2D plane using two leading principal components [Figure 38.2.4]. The neighboring words of *apple* include *macintosh, microsoft, ibm, Windows, mac, intel, computers* as well as *wine, juice*, which capture to some extent the two common meanings in the word *apple*. This example also reveals the drawback of the word2vec approach, where we associate each token with a fixed/static embedding irrespective of context. For example, *apple* in *I like to eat an apple* vs *Apple is great company* means two different things and have the same embedding.

Another example is the word *bank*, which has two contrasting meanings in the following two sentences:

- *We went to the river bank.*
- *I need to go to bank to make a deposit.*

The nearest words of *bank* in the Word2Vec model are *banks, monetary, banking, imf, fund, currency*, etc. , which does not capture the second meaning. More formally, we say static word embeddings from Word2Vec model cannot address polysemy.[3] On the other hand, the mean of a word can usually be inferred from its left and right context. Therefore it is also essential to develop context-dependent embeddings, which will be discussed in section 38.4.

3 the coexistence of many possible meanings for a word or phrase.

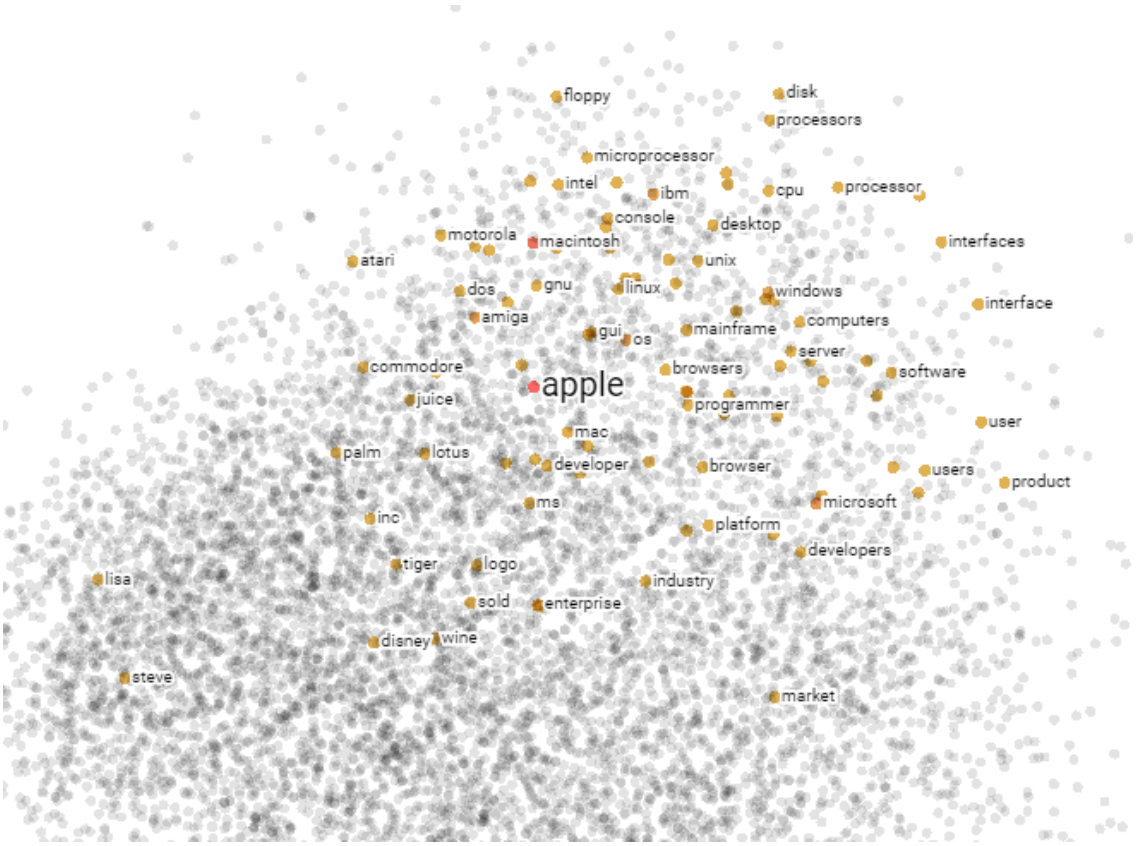


Figure 38.2.4: Visualization of neighboring words of *apple* in a 2D low-dimensional space (first two components via PCA). Image from Tensorflow projector.

38.2.4 GloVe

So far we have largely seen two major approaches to obtaining word embeddings. One is the LSA algorithm based on SVD on the document-term matrix. Since entries in document-term reflects global statistical feature of term, LSA algorithm obtains word embeddings that preserves global information. Another approach is the word2vec algorithm (skip-gram and CBOW), which obtain word embeddings that facilitates prediction of local context words. Word embedding from word2vec algorithm therefore tend to preserve local information.

GloVe, which stands fro Global Vectors for Word Representation, is proposed in [8] to combines ideas from two approaches together. GloVe uses both overall statistics feature of the corpus as well as the local context statistics.

The first step is to construct the co-occurrence probability matrix X, whose entry X_{ij} is the number of times word j occurs in the context of word i. Let $X_i = \sum_k X_{ik}$ be the

number of times any word appears in the context of word i. Finally, let $P_{ij} = P(j|i) = X_{ij}/X_i$ be the probability that word j appear in the context of word i. Following table shows some example entries in the co-occurrence probability matrix. For words i, j that are relevant will have a P_{ij} larger than words that are less irrelevant.

Probability and Ratio	$k =$ solid	$k =$ gas	$k =$ water	$k =$ fashion
$P(k \mid \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \mid \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \mid \text{ice})/P(k \mid \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

We like to use the co-occurrence matrix to guide the search of word embedding vectors, which is achieved by minimizing an objective function J, which evaluates the sum of all squared errors based on the above equation, weighted with a function f :

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Where V is the size of the vocabulary and $f(x)$ is a weighting function that cap the value of x. A empirical choice of f is given by

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

Intuitively, the optimization problem aims to optimize weights w and bias b to approximate $\log X_{ij}$. Bias serves a global offset that encodes global information and product of weights capture the co-occurrence in the local context window. Finally, the model generates two sets of word vectors, W and W', either of which can be used as word embeddings.

38.2.5 Subword model

The word embedding models we discussed so far are typically trained on a large corpus. On the runtime inference stage, there is no guarantee that the words we see during the runtime are in the vocabulary of the training corpus. Those words are known of out-of-vocabulary words, OOV words. Another issue with previous word embedding models is that some text normalization techniques [4] are performed to standardize texts.

4 Some typical text normalization include contraction expansion, stemming, lemmatization, etc. For example, in contraction expansion, we have *ain't* $\rightarrow$ *are not*. Lemmatization is to reduce words to their base forms.

While text normalization allows statistics and parameter sharing across words of the same root (e.g., *bag* and *bags*) and save computational memory and cost, it also ignores meanings that could be encoded in these morphological variations.

Facebook AI research proposed [9] a key idea that one can derive word embeddings by aggregating sub-word level embeddings. It has several advantages: First it addresses the OOV problem by breaking down uncommon words into subword units that are in the training corpus. For example, for the *gregarious* that's not found in the embedding's word vocabulary, we can break it into following character 3-grams, *gre, reg, ega, gar,rio, iou, ous* and combine embeddings of these n-grams to arrive at the embedding of *gregarious*. Second, this approach enables modeling morphological structures (e.g., prefixes, suffixes, word endings, etc.) across words. For example, *dog*, *dogs* and *dogcatcher* have the same root *dog*, but different suffixes to modify the meaning of the word. By allowing parameter sharing across subword units, the eventual word vectors will be enriched with subword level information.

Such subword level modeling is posing an inductive bias that words with similar subword components tend to share similar meaning. For example, the similarity between *dog* and *dogs* are directly expressed in the model. On the other hand, in CBOW or Skip-gram, they are either treated as two different vectors and the same vector, depending on the text normalization applied in the pre-processing step.

The subword model follows the same optimization framework of Skip-gram. Denote a sequence of words $w_1, w_2, ..., w_T$ (represented as integer indices) in a text, the objective of a Skip-gram model is to maximize the likelihood of observing the occurrence of its surrounding words within a context window of size c, given by

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(w_{t+j}|w_t\right)$$

where we have assumed conditional independence given word w_t. Further applying the negative sampling technique, we arrive at the approximate loss function given by

$$\sum_{t=1}^{T}\log\left(1+e^{-s(w_t,w_c)}\right)+\sum_{n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,n)}\right).$$

Here the score function is computed via $s(w_t, w_c) = v_t' \cdot v_c$, where v_t' is the word vector in the input layer and v_c is the word vector in the output layer.

In the subword model, Each word w is represented as a bag of character n-gram. Word boundary symbols $<$ and $>$ are dded at the beginning and end of words to distinguish prefixes and suffixes from other character sequences. For example, the word *where* will be presented as by 3-grams of <*wh, whe, her, ere, re*>

In the subword model, we have a vocabulary V of regular words as well as a vocabulary of n-grams of size G. Given a word w, whose n-gram decomposition is $\mathcal{G}_w \subset \{1, \ldots, G\}$, we let the embedding of w be the sum of the vector representations of its n -grams. That is

$$v_w = \sum_{g \in \mathcal{G}_w} z_g^\top$$

where z_g is the vector representation of n-gram g.

We goal in the training phase is to learn z_g, which can be realized by using the skip-gram optimization except that the scoring function is now

$$s(w, c) = \sum_{g \in \mathcal{G}_w} z_g^\top \mathbf{v}_c.$$

where v_c is the column vector in the output layer matrix associated with word c.

After the n-gram embeddings are trained, we can compute word embedding of each word by aggregating its constituent n-gram embeddings.

Note that the vocabulary size of G can be huge for large n. Below is the maximum number of n -grams as a function of n.

n-grams	maximum number of subwords
3	17576
4	$26^4 \approx 4.6 \times 10^5$
5	$26^5 \approx 1.2 \times 10^7$
6	$26^6 \approx 3.1 \times 10^8$

In order to bound the model memory requirements, we can use a hashing function that maps n -grams to K (e.g., $K \approx 10^6$) buckets. Note that when collison occurs, two n-grams will share the same embedding.

One direct application of subword representation is the Fasttext text classifier**??**, which we discuss in subsubsection 39.1.4.2

38.3 Language modeling

38.3.1 Motivation

Natural languages emerge from formal or casual communications between human beings and only have a limited set of formal rules to follow. Linguists have been directing decades' efforts to modeling languages via grammars, rules, and structure of natural language. In NLP, language modeling [10] tasks involve the the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words forming a sentence that make sense [Figure 38.3.1]. A closed related language modeling task is predict the next word following a sequence of words.

Predicting the next word in a language system can be much more challenging than predicting the next observation in the many physical system. One perspective is that the evolution of many physical systems can governed by physical laws that are well established; on the other hand, the generation of next word, sentence, and even logically coherent paragraph is closely related to human reasoning and intelligence, which remain poorly understood. In fact, it is widely believed that being able to write up logical and coherent paragraphs is an indication of human-level intelligence[11].

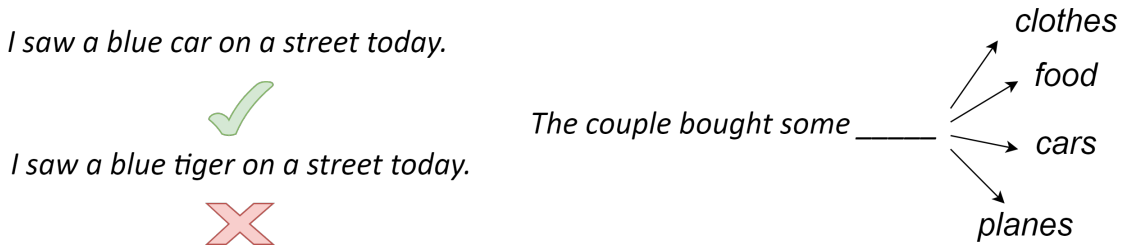


Figure 38.3.1: Illustrating basic language modeling tasks: assigning probability to a sentence (left); and predict the next word based on preceding context (right).

Language modeling has an indispensable components in real-world applications. A language model can be used directly generate new sequences of text that appear to have come from the corpus. For example, an AI-writer can generate an article with given key words. Moreover, language models are an integral part of many text-related applications, where they ensure output word sequences in these tasks to have a sensible meaning or to appear from human beings. These tasks include

- Optical character Recognition, a task converting images to sentences.
- Machine translation, where one sentence from one language is translated to a sentence in another language.

- Image captioning, where a image is used as the input and the output is a descriptive sentence.
- Text summarization, where a large paragraph or a full article is converted several summarizing sentences.
- Speech recognition, where audio signal is converted to meaningful words, phases, and sentences.

In this section, we first consider a classical n-gram statistical language approach [**??**], where we count the co-occurrence of words and model the language generation using probabilistic models (e.g., Markov chains). n-gram models focus on predicting next word based on the superficial co-occurrence counts instead of high-level semantic links between the context and the unknown next word, It further suffers from the curse of dimensionality since sentences are normally long and language vocabulary size is huge (e.g., 10^6). Over the past two decades, the neural network based language models have attracted considerable attention because these models are able to reveal deep connections between words as well as to alleviate the dimensionality challenge. Our focus in this section is on early development of neural language models; recent developments of pre-trained language models (e.g., GPT 1-3 [**??**]) will be covered in next section.

38.3.2 n-gram statistical language model

38.3.2.1 *The baseline counting model*

Given a sequence of N words by $w_1 \dots w_N$. The language modeling task involves the estimation $P(w_1, ..., w_N)$. The task is equivalent to estimate the conditional probability of predicting w_N based on $w_1, ..., w_{N-1}$ thanks to

$$p(w_N | w_1, ..., w_{N-1}) = \frac{w_1, ..., w_N}{w_1, ..., w_{N-1}}.$$

An n-gram statistical language model is one fundamental model that estimates probabilities by counting the co-occurrence of n consecutive words. More precisely, an n-gram is a sequence consisting of n words. For example, given a sentence *The homework is due tomorrow*, it has:

- 2-grams (or bigrams) like *the homework, homework is, is due, due tomorrow*;
- 3-grams (or trigrams) like *the homework is, homework is due, is due tomorrow*;
- 4-grams like *the homework is due, homework is due tomorrow*.

Formally, let's denote a sequence of n words by $w_1 \dots w_n$ or w_1^n, where we use short-hand notation w_1^n to represent n words starting from w_1, i.e., $w_1^n = (w_1, w_2, \dots, w_n)$. The

probability of sequence $w_1, ..., w_N$ have the following decomposition based on conditional probability **chain rule**:

$$\begin{aligned} P\left(w_1^N\right) &= P\left(w_1\right) P\left(w_2|w_1\right) P\left(w_3|w_1^2\right) \ldots P\left(w_N|w_1^{N-1}\right) \\ &= \prod_{k=1}^{N} P\left(w_k|w_1^{k-1}\right) \end{aligned}$$

Computing accurate estimations of $P\left(w_k|w_1^{k-1}\right)$ for arbitrary k is impractical for when k is large. Instead, we can make a *nth-order Markov assumption* and use n-gram statistics to estimate truncated conditional probabilities. For example, based on bigram statistics, we can estimate $P(w_i|w_{i-1})$ and then further approximate

$$P\left(w_N|w_1^{N-1}\right) \approx P\left(w_N|w_{N-1}\right)$$

and therefore

$$P\left(w_1^N\right) \approx \prod_{k=1}^{N} P\left(w_k|w_{k-1}\right).$$

In practice, we usually go beyond the simple bigram approximation. In general, the n-gram approximation for the conditional probability is given by

$$P\left(w_N|w_1^{N-1}\right) \approx P\left(w_N|w_{N-n+1}^{n-1}\right).$$

As such, the full probability of $(w_1, ..., w_N)$ is approximated by and

$$P\left(w_1^N\right) \approx \prod_{k=1}^{N} P\left(w_k|w_{k-n+1}^{n-1}\right).$$

In the training stage with a large corpus, the n-gram conditional probability used above can be simply obtained by counting. For example, the bigram probability of a word $w_n = y$ given a previous word $w_{n-1} = x$, is given by

$$P\left(w_n|w_{n-1}\right) = \frac{\#\left(w_{n-1}w_n\right)}{\sum_w \#\left(w_{n-1}w\right)} = \frac{\#\left(w_{n-1}w_n\right)}{\#\left(w_{n-1}\right)}$$

where $\#(xy)$ is the total count of bigrams xy in the corpus and $w_{n-1}w$ enumerates all distinct bigrams.

Note that the storage requirement for n-grams scales like $O(|V|^n)$, where $|V|$ is the vocabulary size. A language model with a large n can overfit to small-sized training corpus, as it tends to memorize specific long sequence patterns in the training corpus and to give low probabilities to those unseen. Using a bigram model can underfit since bigram can hardly capture the sequential, long distance relationship among words.

38.3.2.2 *Evaluation*

Given a training corpus, we can construct different language models by varying different n-grams as well as other model hyperparameters we will introduce in subsequent sections. Similar to supervised machine learning problems, n-gram language models face bias and variance trade-off given a finite-sized corpus. How can we know one is better than the other?

One principled way is to evaluate the model on a test set with unseen sentences and then compare the probability or its variant computed from different candidate models on the test set. A language model that assigns a higher probability to the test set is considered a better one. We define **perplexity** of a language model on a test set as the inverse probability of the test set, normalized by the number of words. For a test set $W = (w_1 w_2 \dots w_N)$ (N can be millions),

$$\begin{aligned} \text{perplexity}(W) &= e^{-\frac{1}{N}\sum_{i=1}^{N} \ln P(w_1 w_2 \dots w_N)} \\ &= \sqrt[N]{\frac{1}{P\left(w_1 w_2 \dots w_N\right)}}, \end{aligned}$$

where $P\left(w_1 w_2 \dots w_N\right)$ can be estimated in an n-gram model via

$$P\left(w_1^n\right) \approx \prod_{i=1}^{N} P\left(w_i | w_{i-n+1}^{n-1}\right).$$

Intuitively, perplexity is roughly the inverse probability of the test set. Therefore, a good larnguage model will assign high probability to the test set and produce a low perplexity value. The perplexity is always greater than or equal to 1.0.

38.3.2.3 *Smoothing technique*

One fundamental limit of the baseline counting method is the zero probability assigned to n-grams that do not appear in the training corpus. In the evaluation process, one zero probability of any unseen n-gram will cause the probability of the test set is 0 or infinite large perplexity. A simply remedy is to assume each n-gram occurred at least k times in addition to its observations in the corpus. The resulting estimation is given by

$$P^*_{\text{Add-}k}\left(w_n | w_{n-1}\right) = \frac{\#\left(w_{n-1} w_n\right) + k}{\#\left(w_{n-1}\right) + kV}.$$

Another popular approach is **back-off**. If the n-gram is not observed, use the statistics of $n-1$ grams, and so on. The latest development of n-gram language modeling is to use modified Kneser Ney smoothing[12].

Poor statistics can also be cause of the existence of many out-of-vocabulary rare words in the corpus. One can choose a vocabulary (word list) that is fixed in advance and convert any OOV word o the special unknown word token $<$ UNK $>$ in a text normalization step. Also see [13] for a discussion of different techniques.

38.3.3 Feed-forward neural language model

The n-gram statistical language model aims to learn the joint distribution of word sequences. This approach suffers from curse of dimensionality when n becomes large. For example, consider a language with a vocabulary of size $V = 10^6$, a 10-gram model would require model parameters of V^{10}. On the other hand, natural languages tend to have long-range dependency, i.e., the occurrence probability of a word may depend on the existence of an another word multiple steps before. Therefore, a high-quality language model must require large n and, unfortunately, the curse of dimensionality becomes one inherent limitation of n-gram models.

n-gram models also fails to capture the semantic meaning of words[14]. The core idea of n-gram model is nothing but a mechanical counter of co-occurrence of words. In natural language, there are many words that are similar in their meaning as wells as their grammar rules. For example, *A cat is walking in the living room* vs. *a dog is running in the bedroom* have similar word pairs (cat, dog), (walking, running), (living room, bedroom) and use similar patterns. These similarities or word semantic meaning can be exploited to construct model with smaller parameters.

To overcome the limitation of n-gram models, Bengio et al [14] proposed neural language models, which predict and generate next word based on its context and operating at a low dimensional dense vector space (i.e., word embedding). In the early development of neural language model, there are two important attempts. One is feed-forward neural network [14] and one is recurrent neural network[15] [**??**].

The core idea is that by projecting words into low dimensional space (via learning and gradient descent), similar words are automatically clustered together. The curse of dimensionality is naturally addressed because of the efficient parameterization in neural networks.

In feed-forward network model [Figure 38.3.2], each word, together with its preceding $N-1$ words as context are projected into low-dimensional space and further predict the next word probability. Note that the context has a fixed length of n, which it is limited in the same way as in n-gram models.

Formally, the model is optimized to maximize

$$P\left(w_t \mid w_{t-1}, \cdots w_{t-n+1}\right) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

The y_i is the logic for word i, given by

$$y = b + We + U \tanh(d + He)$$

where W, H are matrices and b, d are biases. Here W can optionally zero, meaning no direct connections. $e = (e_1, ..., e_{n-1})$ is the concatenations of word embeddings of each preceding token.

Feed-forward neural language model brings several improvements over the traditional n-gram language model: it provides a compact and efficient parameterization to capture word dependencies among text data. Recall that n-gram model would have to store all observed n-grams. However, feed-forward neural language model still meet challenges to capture long-distance dependencies in natural language. In the feed-forward neural language model, capturing long-distance dependencies will require an increase of the context window size, which linearly scales with model parameters W. Another drawback is that a word that appears at different locations will be multiplied by different weights to get its embedding, which is inconsistent.

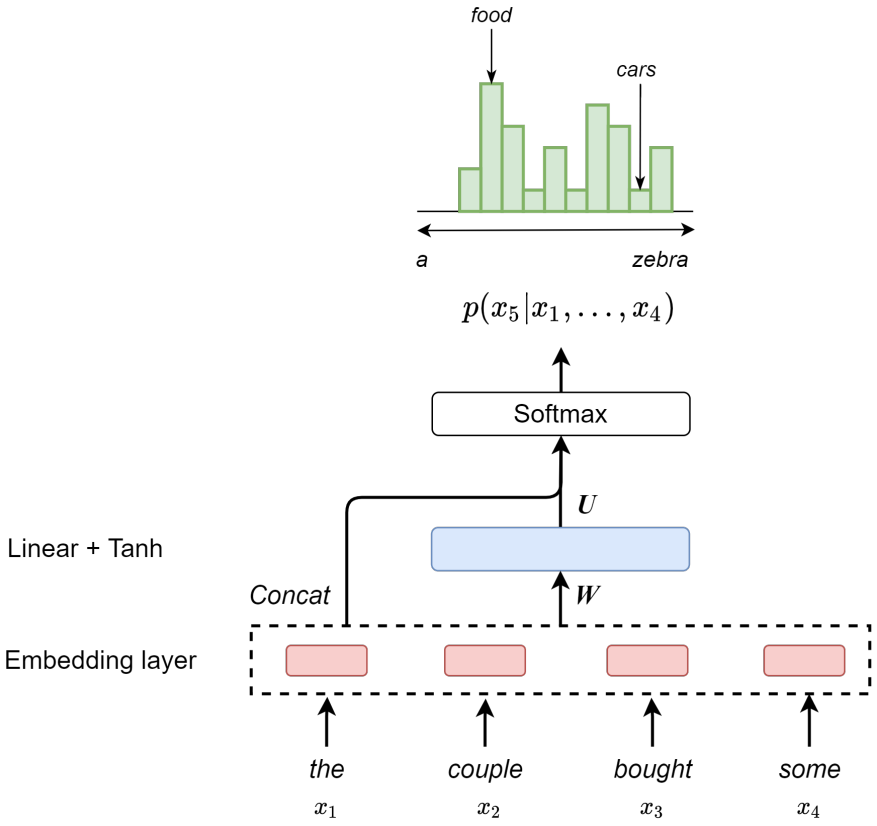


Figure 38.3.2: Feedforward neural netowk based language model.

38.3.4 Recurrent neural language model

In recurrent network model [Figure 38.3.3], context is extended via recurrent connections and context length is theoretically unlimited. Specially, let the recurrent network input be x, hidden layer output be h and output probabilities be y. Input vector x_t is a concatenation of a word vector w_t and the previous hidden layer output s_{t-1}, which represents the context. To summarize, we have recurrent computation given by

$$x_t = \text{Concat}(e_t, h_{t-1})$$
$$h_t = \text{Sigmoid}\left(W_x x_t + b_x\right)$$

The prediction probability at each t is given by

$$p(y_k(t) = \text{Softmax}\left(W_y s(t) + b_y\right)$$

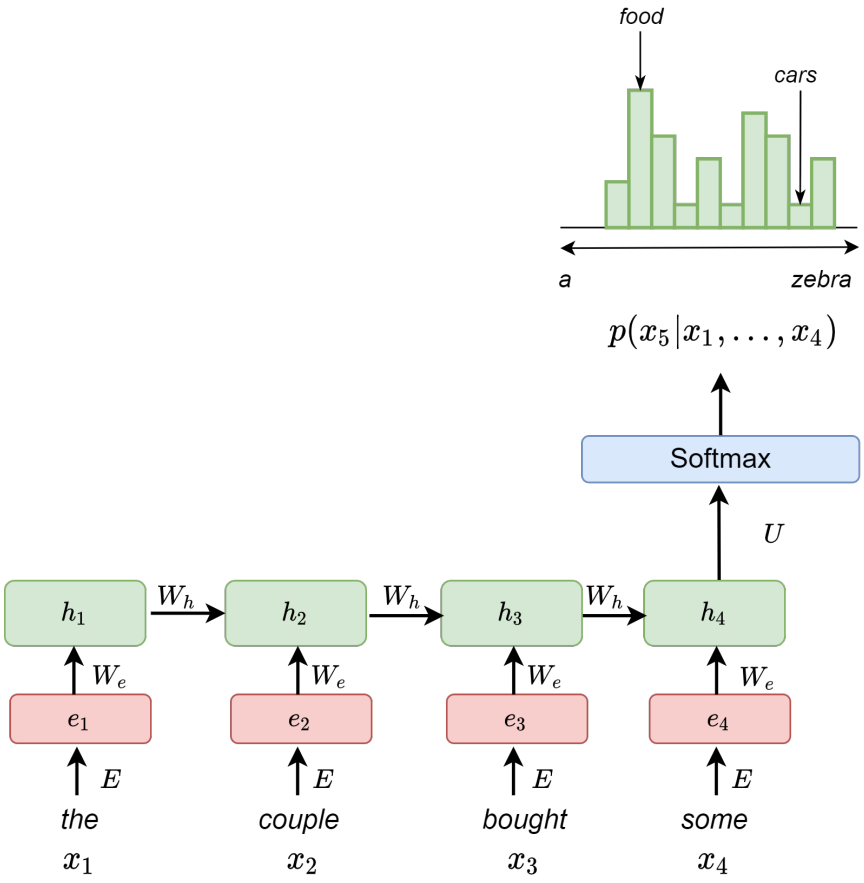


Figure 38.3.3: Recurrent neural network based language model.

Compared with n-gram language model and feed-forward neural language model, RNN language model can in principle process input of any length without increasing the model size. RNN language model also has several drawbacks: Computing a conditional probability $p(w_t|w_{1:t-1})$ is expensive. One mitigating strategy is to cache and re-use previous computed results or to pre-compute conditional probabilities for frequent n-grams. some In practical applications, RNN language models from different domains are not easy to merge;

38.4 Contextualized word embeddings

38.4.1 Introduction

In natural language processing, although we have seen many successful end-to-end systems , they usually require large scale training examples and the systems require a complete retrain for a different task. Alternatively, we can first learn a good representation of word sequences that not task-specific but would be likely to facilitate downstream specific tasks. Learning a good representation in prior from broadly available unlabeled data also resembles how human perform various intelligent tasks. In the context of natural languages, a good representation should capture the implicit linguistic rules, semantic meaning, syntactic structures, and even basic knowledge implied by the text data.

With a good representation, the downstream tasks can be significantly sped up by fining training the system on top of the representation. Therefore, the process of a learning a good representation from unlabeled data is also known as pre-training a language model.

Pre-training language models have several advantages: first, it enable the learning of universal language representations that suit different downstream tasks; second, it usually gives better performance in the downstream tasks after fine-tuning on a target task; finally, we can also interpret pre-training as a way of regularization to avoid overfitting on small data set.

In section 38.2, we discussed different approaches (e.g., Word2Vec, GloVe) to learning a low-dimensional dense vector representation of word tokens. One significant drawback of these representations is context-independent or context-free static embedding, meaning the embedding of a word token is fixed no matter the context it is in. By contrast, in natural language, the meaning of a word is usually context-dependent. For example, in sentences *I like to have an apple since I am thirty* vs. *I like to have an Apple to watch fun movies*, the word *apple* mean the fruit apple and the electronic device, respectively.

There has been significant efforts directed to learning contextual embedding of word sequences[16]. A contextual embedding encoder usually operates at the sequence level. As shown in Figure 38.4.2, given a non-contextual word embedding sequence $x_1, x_2, \cdots, x_T$, the contextual embeddings of the whole sequence are obtained simultaneously via

$$[h_1, h_2, \cdots, h_T] = \text{ContextualEncoder}\left(x_1, x_2, \cdots, x_T\right).$$

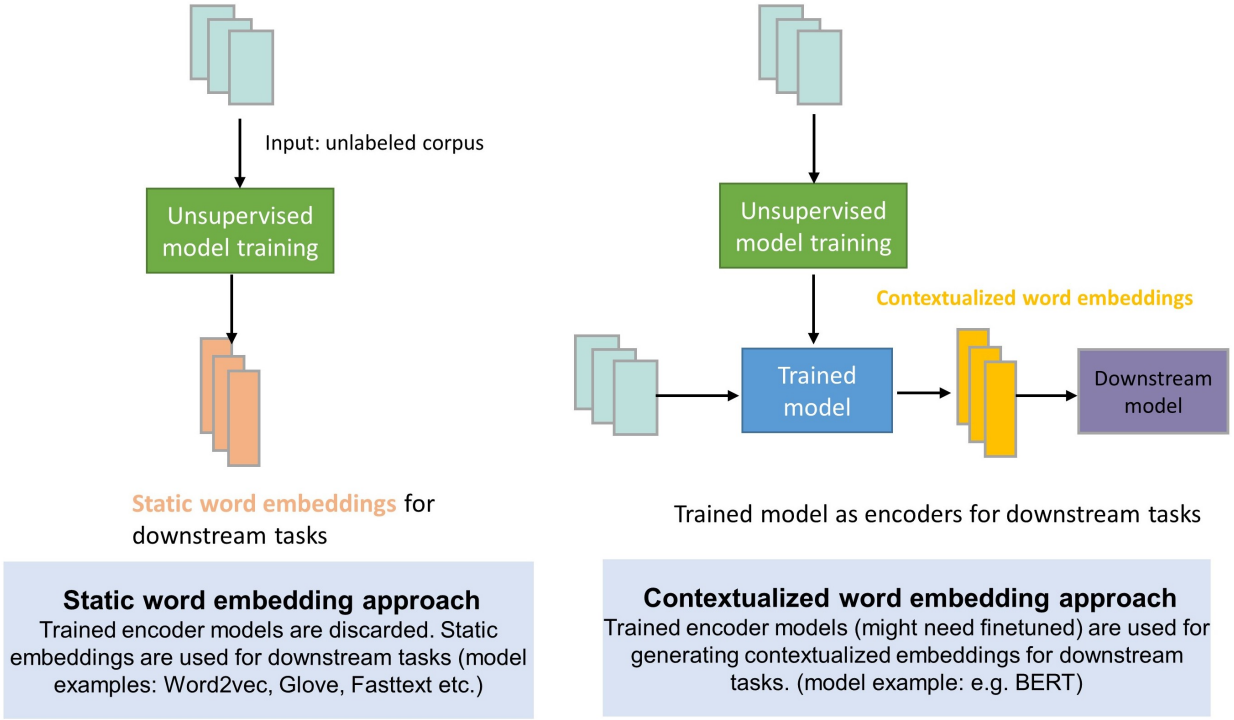


Figure 38.4.1: Static word embedding approach vs. contextualized word embedding approach.

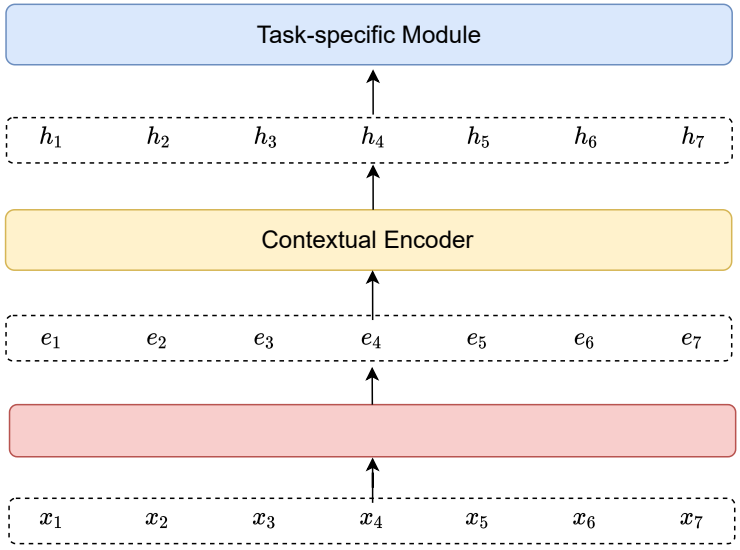


Figure 38.4.2: A generic neural contextual embedding encoder.

Given large-scale unlabeled data, the most common way to learn a good representation is via self-Supervised learning. The key idea of self-Supervised learning is

to predict part of the input from other parts in some form (e.g., add distortion). By minimizing prediction task loss or other auxiliary task losses, the neural network learns good presentations that can be used to speed up downstream tasks.

Since the advent of the most successful pretrained language model BERT[17], many follow-up research found that the performance of the pretrained model in downstream tasks highly depend on the self-supervised tasks in the pretraining stage. If the downstream tasks are closely related to self-supervised tasks, a pretrained model can offer significant performance boost. And the fine tuning process can be understood as a process that further improves features relevant to downstream tasks and discards irrelevant features.

38.4.2 BERT

BERT, Bidirectional Encoder Representations from Transformers[17], is one of the most successful pre-trained language model. BERT relies on a Transformer (the attention mechanism that learns contextual relationships between words in a text). BERT model heavily utilizes stacked self-attention modules to contextualize word embeddings[Figure 38.4.3].

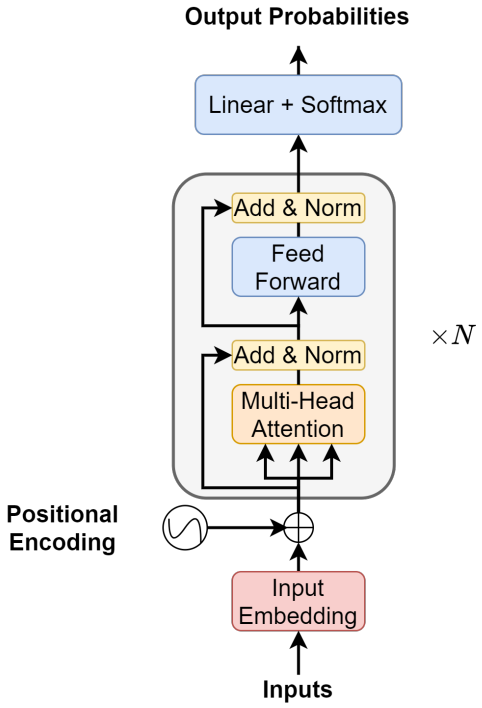


Figure 38.4.3: BERT model architecture

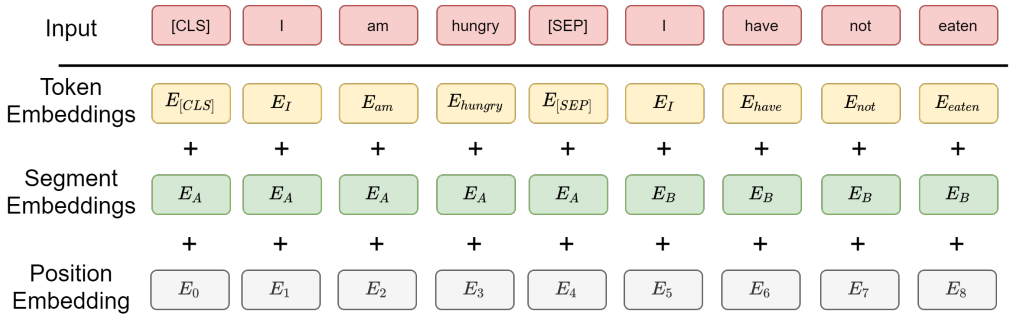


Figure 38.4.4: Input embedding in BERT, which consists of token embedding, segmenet embedding and positional embedding.

In the following, we will go through detailed structure of BERT.

38.4.2.1 *Input embeddings*

The input to the BERT is a sequence of token representations. The representation of each token is a dense vector of dimensionality d_{model}, which is the summation of following components:

- **Token embedding** of dimensionality d_{model}, which is the ordinary dense word embedding. Specifically, a sub-word type of embedding, called wordPiece embedding [18], with a 30,000 token vocabulary is used. Note that A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- **Segment embedding** of dimensionality d_{model}, which is a marker 0 or 1 indicating if sentence A precedes sentence B. Segment embedding is typically learned from the training.
- **Positional embedding** of dimensionality d_{model}, which encodes information of position in the sentence. Positions matters in word and sentence level meanings. For example, *I love you* and *you love me* are different. Position embedding is a vector depending on where the token is located in the segment. It is a constant vector throughout the training.

The token, segment, and position embeddings are implemented through a look-up matrix. The token embedding look-up matrix has a size of $(vocab\ size, d_{model})$; the position embedding look-up matrix has a size of $(max\ len, d_{model})$; the segment embedding look-up matrix has a size of $(2, d_{model})$.

38.4.2.2 *Position encodings*

Consider an input sequence represented by integer sequence $s = (i_1, ..., i_p, ..., i_n), i_p \in \mathbb{N}$, *e.g.*, $s = (3, 5, 30, 2, ..., 21)$. For the input sequence s, the token embedding and segment embedding do not encode positional information in the sequence, so we can utilize a position encoding PE maps an integer index representing the position of the token in the sequence, $s^p = (1, 2, ..., n)$ to n dense vector with same dimension d_{model}, i.e., $\text{PE}(s^p) \in \mathbb{R}^{n \times d_{model}}$.

Notably, given the token position $i \in \{1, ..., n\}$ $\text{PE}(i) \in \mathbb{R}^{d_{model}}$ is given by

$$[\text{PE}(i)]_j = \begin{cases} \sin\left(\frac{i}{10000^{2^j/d_{model}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{2^{j-1}/d_{model}}}\right) & \text{if } j \text{ is odd} \end{cases}$$

where $j = \{1, ..., d_{model}\}$. Note that the position encodings have the same dimension d_{model} as the word embeddings such that they can be summed. Intuitively, each dimension of the positional encoding corresponds to a sine wave of different wavelengths ranging from 2π to $10000 \cdot 2\pi$. An example position encodings of dimensionality 256 for position index from 1 to 256 is shown in Figure 38.4.5.

The position encoding can take other function forms as well. In theory, the purpose of the position encoding is to preserve position information by mapping different position to different values. The position encoding mapping can also be learned from data.

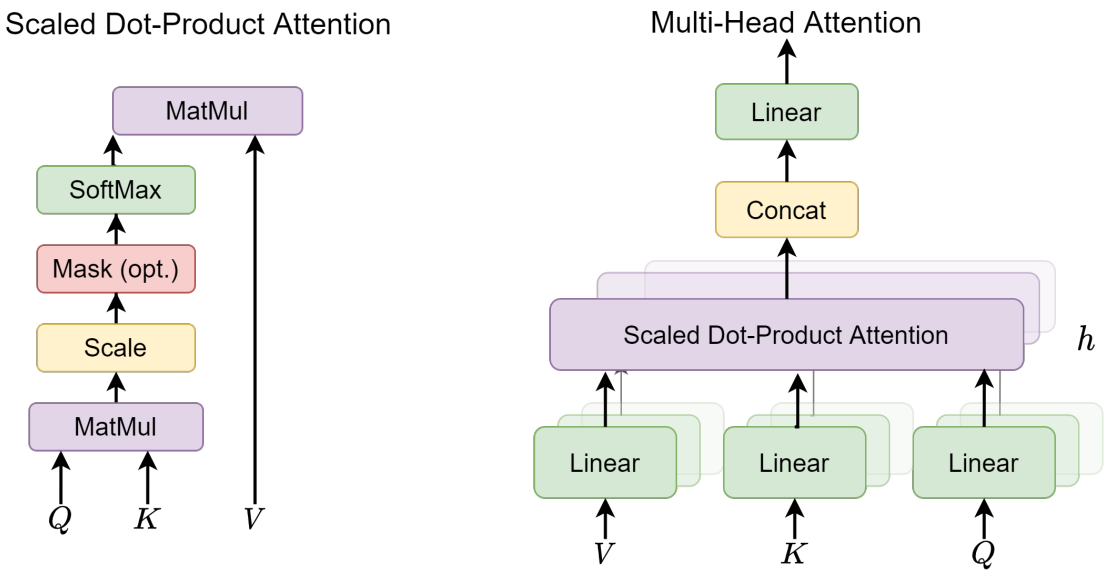


Figure 38.4.6: The multi-head attention architecture

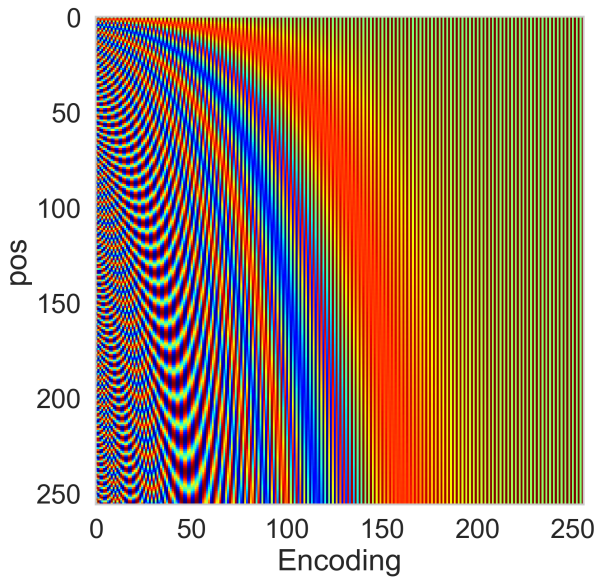


Figure 38.4.5: Example position encodings of dimensionality 256 for position index from 1 to 256.

38.4.2.3 *Multihead attention with marks*

Multi-head attention mechanism plays a central role in both encoder and decoder side; Self-attention itself enables embedding contextualization for each token that depends on its context (i.e., self-attention of the input sequence). Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions[19].

Given a query matrix $Q \in \mathbb{R}^{n \times d_{model}}$, a key matrix $K \in \mathbb{R}^{m \times d_{model}}$, and a value matrix $V \in \mathbb{R}^{m \times d_{model}}$, the multi-head ($H$ heads) attention associated with (Q, K, V) is given by

$$\text{MultiHeadAttention}\left(Q, K, V\right) = Concat\left(head_1, \cdots, head_h\right) W^O$$

where $head_i \in \mathbb{R}^{n \times d_v}$

$$head_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

with $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{h \times d_v \times d_{model}}$. In general, we require $d_k = d_v = d_{model}/h$ such that the output of $\text{MultiHeadAttention}\left(Q, K, V\right)$ has the dimensionality of $n \times d_{model}$.

The attention among (Q, K, V) is given by

$$\text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) = \text{softmax}\left(\frac{QW_i^Q\left(KW_i^K\right)^T}{\sqrt{d_k}}\right) VW_i^V.$$

Note that the Softmax applies to each row such that the $\text{softmax}(\cdot)$ produces a weight matrix $w^{att} \in \mathbb{R}^{n \times m}$, with each row summing up to unit 1.

Explicitly, we have

$$\begin{bmatrix} w_{11}^{att} & w_{12}^{att} & \cdots & w_{1m}^{att} \\ w_{21}^{att} & w_{22}^{att} & \cdots & w_{2m}^{att} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{att} & w_{n2}^{att} & \cdots & w_{nm}^{att} \end{bmatrix} \begin{bmatrix} [VW^V]_1 \\ [VW^V]_2 \\ \vdots \\ [VW^V]_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^m w_{1j}^{att}[VW^V]_j \\ \sum_{j=1}^m w_{2j}^{att}[VW^V]_j \\ \vdots \\ \sum_{j=1}^m w_{nj}^{att}[VW^V]_j \end{bmatrix}$$

where $[VW^V]_i$ is the ith row vector of value matrix VW^V.

Sometimes, for each query associated with a symbol, we like to only allow a subset of keys and values to be queried via a binary **mask** $mask \in \{0, 1\}^m$, where 1 indicates exclusion of the key. We can set the values in the intermediate matrix $QW_i^Q(KW_i^K)^T$ to be negative infinity if this column index is associated with mask value 1.

Remark 38.4.1. On the scalability of Self-Attention At this point, it is apparent that the memory and computational complexity required to compute the attention matrix is quadratic in the input sequence length, i.e., $N \times N$. In particular, the $QK^\top$ matrix multiplication operation alone consumes N^2 time and memory. This restricts the overall utility of self-attentive models in applications which demand the processing of long sequences. In subsequent sections, we discuss methods that reduce the cost of self-attention.

38.4.2.4 *Point-wise feed-forward network*

After all inputs go through multi-head self-attention layer, it is still a linear mapping with respect to the input. To enhance the modeling capacity with non-linearity and to consider interactions between different latent dimensions, we apply a point-wise two-layer feed-forward network with ReLU activation to input $x \in \mathbb{R}^{d_{model}}$ at each position separately and identically (i.e., sharing parameters across positions):

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2.$$

There are additional dropouts, residual connections, and layer-wise normalization after the point-wise feed-forward layer. All together, we can define the following encoder layer [20].

Definition 38.4.1 (Encoder layer). *The BERT encoder with n sequential inputs can be decomposed into following calculation procedures*

$$\begin{aligned} e_{mid} &= \text{LayerNorm}(e_{in} + \text{MultiHeadAttention}(e_{in}, e_{in}, e_{in}, padMask)) \\ e_{out} &= \text{LayerNorm}(e_{mid} + \text{FFN}(e_{mid})) \end{aligned}$$

where $e_{mid}, e_{out} \in \mathbb{R}^{n \times d_{model}}$,

$$\text{FFN}(e_{mid}) = \max(0, e_{mid}W_1 + b_1)W_2 + b_2,$$

with $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}, b_1 \in \mathbb{R}^{d_{ff}}, b_2 \in \mathbb{R}^{d_{model}}$, *and the padMask excludes padding symbols in the sequence.*

In a typical setting, we may have $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$. Also note that there is active research on where to optimally add the layer normalization in the encoder.

38.4.2.5 *Put it together*

The whole computation in the encoder module can be summarized in the following.

Definition 38.4.2 (computation in encoder module). *Given an input sequence represented by integer sequence $s = (i_1, ..., i_p, ..., i_n)$ and its position $s^p = (1, ..., p, ..., n)$. The encoder module takes s, s^p as inputs and produce $e_N \in \mathbb{R}^{n \times d_{model}}$.*

$$\begin{aligned} e_0 &= \text{WE}(s) + \text{PE}(s^p) \\ e_1 &= \text{EncoderLayer}(e_0) \\ e_2 &= \text{EncoderLayer}(e_1) \\ &\dots \\ e_L &= \text{EncoderLayer}(e_{L-1}) \end{aligned}$$

where $e_i \in \mathbb{R}^{n \times d_{model}}$, $\text{EncoderLalyer} : \mathbb{R}^{n \times d_{model}} \to \mathbb{R}^{n \times d_{model}}$ is an encoder sub-unit, N is the number of encoder layers. Specifically, this encoder layer is given by Definition 38.4.1.

Note that Dropout operations are not shown above. Dropouts are applied after initial embeddings e_0, every self-attention output, and every point-wise feed-forward network output.

Commonly used BERT models take the following configurations:

- BERT-BASE, $L = 12, d_{model} = 768, H = 12$, total Parameters 110M.
- BERT-LARGE, $L = 24, d_{model} = 1024, H = 16$, total parameters 340M.

38.4.2.6 *Compared with ELMO*

An influential contextualized word embedding model via deep learning is ELMO (Embeddings from Language Models)[21], in which word vectors are learned functions of the internal states of a deep bidirectional LSTM language model [Figure 38.4.7]. Given a sequence of N tokens (character-based), $(t_1, t_2, \dots, t_N)$. Their static word embeddings $(e_1, ..., e_N)$ are contextualized by stacked bidirectional LSTM as $(h_1, ..., h_N)$.

The LSTMs are pre-trained to perform language modeling task, i.e., predicting the next token given preceding tokens in forward and backward directions, respectively. Specifically, an linear plus Softmax layer take LSTM hidden states (two directions) as the input and computes the distribution. The following log-likelihood of the forward and backward directions is maximized:

$$\sum_{k=1}^{N} \left(\log p\left(t_k \mid t_1, \ldots, t_{k-1}; \Theta\right) + \log p\left(t_k \mid t_{k+1}, \ldots, t_N; \Theta\right) \right)$$

After pretraining, the top layer LSTM hidden states are used as contextualized embeddings.

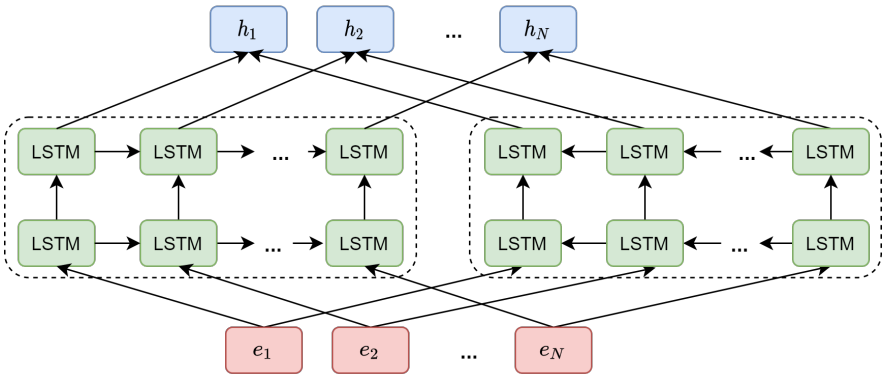


Figure 38.4.7: In ELMO, static word embeddings $(e_1, ..., e_N)$ are contextualized by stacked bidirectional LSTM as $(h_1, ..., h_N)$.

Compared to ELMO, BERT is deeply bidirectional due to its novel masked language modeling technique. Having bidirectional context is expected to theory, generate more accurate word representations.

Another improve of BERT over EMLO is the tokenization strategy. BERT tokenizes words into sub-words (using WordPiece), while ELMO uses character based input. It's often observed that character level language models don't perform as well as word based or sub-word based models.

38.4.2.7 *Computation consideration*

The computational complexity in the encoder structure is from the self-attention layer and the feed-forward layer. The total computational complexity gives $O\left(n^2 d + nd^2\right)$, where $O(n^2 d)$ is from self-attention layer and $O(nd^2)$ is from feed-forward layer. The dominance of either term depends on the length of the sequence and model dimensionality. If we use RNN architecture for context modeling, each layer will have complexity $O(nd^2)$. Note that, a convenient property of the Transformer encoder is that the computation in each self-attention layer is fully parallelizable, which is amenable to GPU acceleration. On the other hand, RNN-based methods is sequential. During testing, for each user, after calculating the embedding $\mathbf{F}_n^{(b)}$, the process is the same as standard MF methods. $(O(d)$ for evaluating the preference toward an item).

Transformer encoders does not scale well to long sequences. Strategies to alleviate this issue includes 1) using restricted self-attention which only attends on local r inputs; 2) splitting long sequences into short segments and fuse segment level embeddings via additional networks.

To summarize, we have following table.

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O\left(n^2 \cdot d\right)$	$O(1)$
Recurrent	$O\left(n \cdot d^2\right)$	$O(n)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

38.4.3 Pre-training

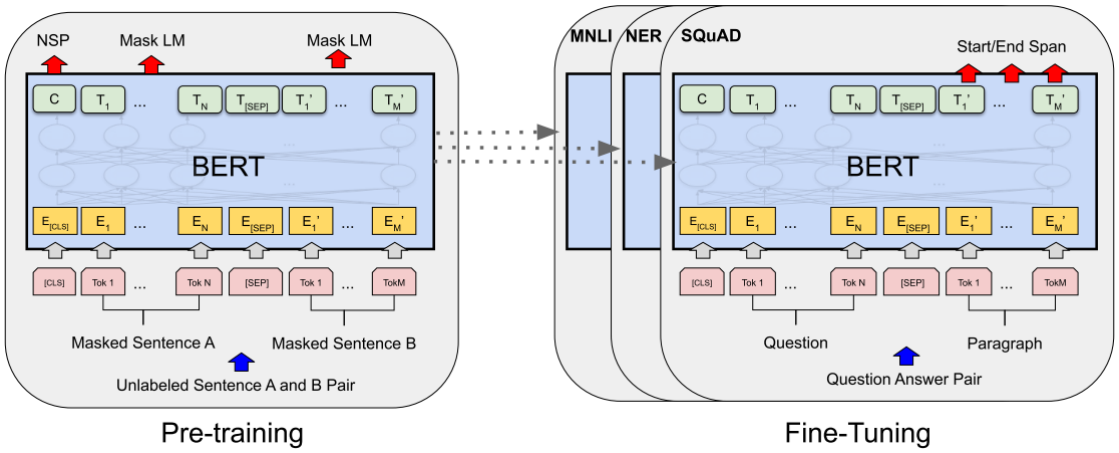


Figure 38.4.8: BERT pre-training and downstream task fine tuning framework. Image from [17].

There are two tasks to pretrain the network: masked language modeling (Marked LM) and next sentence prediction (NSP). In the Masked LM, some percentage of randomly sampled words in a sequence are masked, i.e., being replaced by a [MASK] token. The task is to predict (via Softmax) only the masked words, based on the context provided by the other non-masked words in the sequence.

Masked LM task has this drawback of introducing Mismatch between pretraining and fine-tuning stage: in the fine-tuning stage, training sentences do not contain masked tokens. To reduce the mismatch between pre-training and fine-tuning, as the [MASK] symbol never appears during the fine-tuning stage, different masking strategies are explored. It is found one effective strategy is to select 15% of tokens for the following possible replacement. For each token,

- 80% probability is replaced by [MASK].
- 10% probability is replaced by a random token.
- 10% probability is left unchanged.

In additional, Mask LM task also suffers from training inefficiency and slow convergence since each batch only 15% of masked tokens are predicted.

In the NSP, the network is trained to understand relationship between two sentences. A pre-trained model with this kind of understanding is relevant for tasks like question answering and natural language Inference. The input for NSP is a pair of segments, which can each contain multiple natural sentences, but the total combined length must be less than 512 tokens. Notably, it is found that using individual natural sentence pairs hurts performance on downstream tasks[22].

The model is trained to predict if the second sentence is the next sentence in the original text. In choosing the sentences pair for each pretraining example, 50% of the time, the second sentence is the actual next sentence of the first one, and 50% of the time, it is a random sentence from the corpus.

The next sentence prediction task can be illustrated in the following examples.
Input: [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
Label: IsNext
Input: [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight1ess birds [SEP]
Label: NotNext

The pre-training is performed in a joint manner with loss function given by

$$L\left(\theta, \theta_1, \theta_2\right) = \underbrace{L_1\left(\theta, \theta_1\right)}_{\text{Mask LM}} + \underbrace{L_2\left(\theta, \theta_2\right)}_{\text{NSP}},$$

where θ are the BERT encoder parameters BERT, θ_1 are the parameters of the output layer (linear layer with Softmax) associated with the Mask-LM task and θ_2 are the parameters of the output layer associated with NSP task.

More specifically, for each batch of sentence pair with masks, we have

$$L_1\left(\theta, \theta_1\right) = -\sum_{i=1}^{M} \log p\left(m = m_i \mid \theta, \theta_1\right), m_i \in [1, 2, \ldots, |V|]$$

where $m_1, .., m_M$ are masked tokens and $|V|$ is the vocabulary size. In the NSP task, we have sentence level classification loss

$$L_2\left(\theta, \theta_2\right) = -\sum_{j=1}^{N} \log p\left(n = n_i \mid \theta, \theta_2\right), n_i \in [\text{ IsNext, NotNext }]$$

The joint loss for the two pretaining task is then written by

$$L\left(\theta, \theta_1, \theta_2\right) = -\sum_{i=1}^{M} \log p\left(m = m_i \mid \theta, \theta_1\right) - \sum_{j=1}^{N} \log p\left(n = n_i \mid \theta, \theta_2\right).$$

Pre-training data include the BooksCorpus (800M words) and English Wikipedia (2,500M words). The two corpus in total have a size of 16 GB.

38.4.4 Downstream tasks: GLUE

Contextualized word embeddings from pre-trained language model have greatly advanced many NLP tasks, attaining human level performance for relative straightforward tasks such as text classification. Since long term goal of NLP [23].

38.4.4.1 *Single-sentence tasks*

CoLA The Corpus of Linguistic Acceptability consists of English acceptability judgments drawn from books and journal articles on linguistic theory. Each example is a sequence of words annotated with whether it is a grammatical English sentence. A model will be evaluated on unbalanced binary classification and ranges from -1 to 1 , with 0 being the performance of uninformed guessing.

SST-2 The Stanford Sentiment Treebank consists of sentences from movie reviews and human annotations of their sentiment. The task is to predict the sentiment (positive or negative) of a given sentence.

38.4.4.2 *Similarity and paraphrase tasks*

MRPC The Microsoft Research Paraphrase Corpus is a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent.

QQP The Quora Question Pairs [2] dataset is a collection of question pairs from the community question-answering website Quora. The task is to determine whether a pair of questions are semantically equivalent.

STS-B The Semantic Textual Similarity Benchmark (Cer et al., 2017) is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data.

38.4.4.3 *Inference tasks*

MNLI The Multi-Genre Natural Language Inference Corpus [24] is a crowdsourced collection of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral).

QNLI The Stanford Question Answering Dataset[25] is a question-answering dataset consisting of question-paragraph pairs, where one of the sentences in the paragraph (drawn from Wikipedia) contains the answer (specified by a span in the paragram) to the corresponding question. The task is to determine whether the context sentence contains the answer to the question.

RTE The Recognizing Textual Entailment (RTE) datasets come from a series of annual textual entailment challenges.

WNLI The Winograd Schema Challenge is a reading comprehension task in which a system must read a sentence with a pronoun and select the referent of that pronoun from a list of choices.

38.4.5 Fine-tuning and evaluation

A pretrained BERT model can be fine-tuned to a wide range of downstream tasks, as we introduced in the previous section [subsection 38.4.4]. Depending on the task type, different architecture configurations will be adopted Figure 38.4.9:

- For single-sentence tasks, such as sentiment analysis, tokens of a single sentence will be fed into BERT. The embedding output corresponding to the [CLS] token will be used in a linear classifier to predict the class label.
- For sequence-labeling tasks, where named-entity-recognition, tokens of a single sentence will be fed into BERT. The token embedding outputs will be used in a linear classifier to predict the class label of a token.
- For sentence-pair tasks, where the relationship between two sentences will be predicted, tokens from two sentences will be fed into BERT. The embedding output corresponding to the [CLS] token will be used in a linear classifier to predict the class label.
- For questioning-answering tasks, where the start and end span needs to be determined in the context paragraph, question sentence and context paragraph sentence will be fed into BERT. The token embedding outputs on paragraph side will be used in a linear classifier to predict the start and end span.

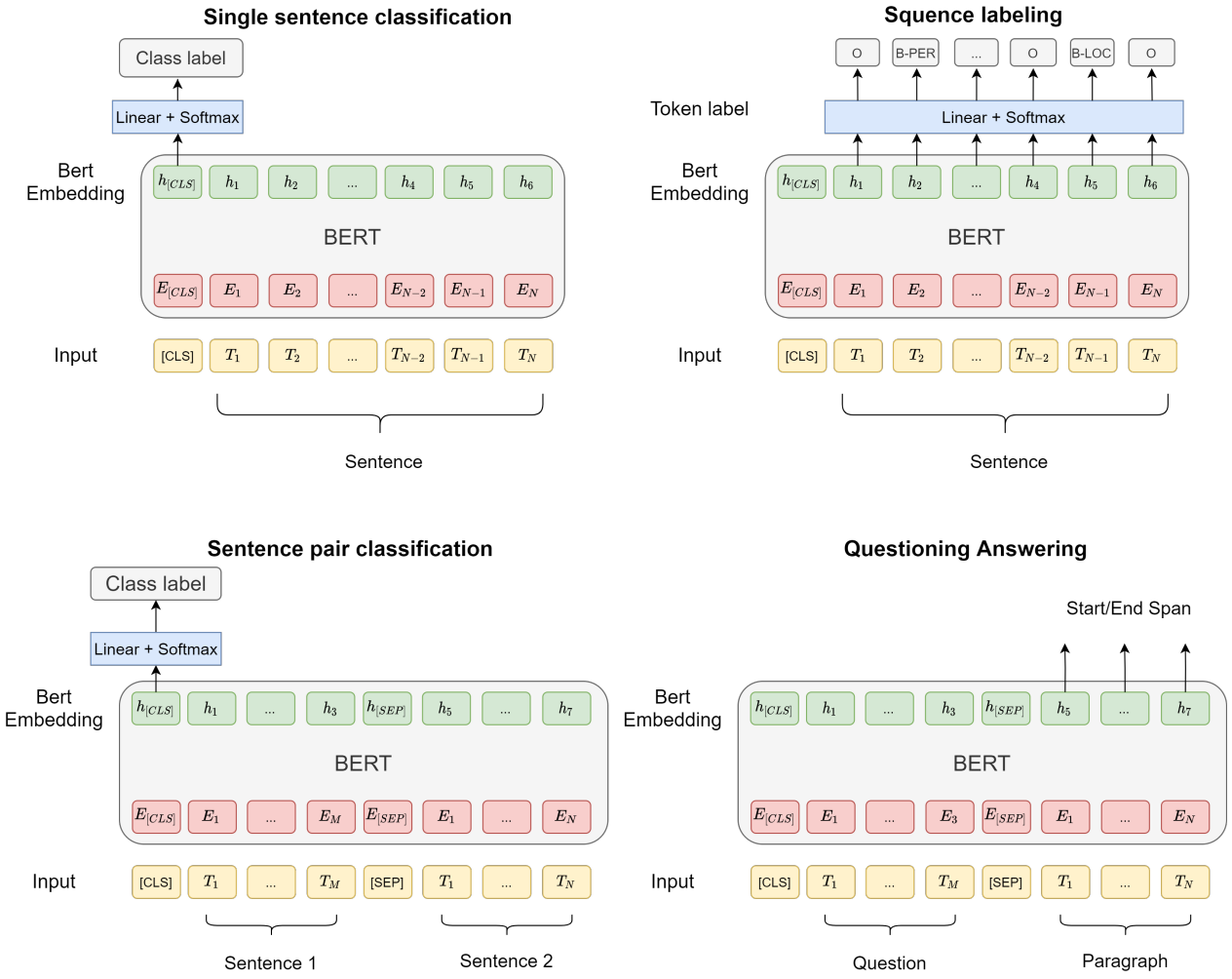


Figure 38.4.9: BERT architecture configuration for different downstream tasks.

38.4.6 Other BERT family members

38.4.6.1 *RoBERTa*

RoBERTa[22], standing for Robustly Optimized BERT, explored Bert model's performance limit by improving following training settings.

Dynamic masking. The original BERT implementation performs masking once during data preprocessing, resulting in a single static mask. To avoid using the same mask for each training instance in every batch, the masking pattern is generated *dynamically* every time a sequence is fed into the model.

Large Batching. Large batch size often leads to faster convergence and better final result. Roberta enlarges the batch size to 2,000 from the original Bert batch size of 256. Increasing the batch size will allow the the model to converge several time faster.

Large training corpus and time. Large training corpus, more than 10 times larger than the training corpus for BERT.

Input format and removing next sentence prediction loss. In the original BERT, each input has a pair of segments, which can each contain multiple natural sentences, but the total combined length must be less than 512 tokens. In RoBERTa, each input is packed with full sentences sampled contiguously from one or more documents, such that the total length is at most 512 tokens. Inputs may cross document boundaries, with an extra separator token between documents. The NSP loss is removed.

It is found that with these training settings, additional performance gain was achieved.

38.4.6.2 *ALBERT*

BERT has achieved marked success in tackling challenging NLP tasks such as natural language inference, machine reading comprehension, and question answering. In an end-to-end system, BERT can used as an encoder, which connects back-end task-specific modules (e.g., classifiers). By fine-tuning the BERT, the system can usually achieve satisfactory results even under limited resources setting (e.g., small training set). However, BERT is huge model, the base version has 108M parameters, which prohibits its application in devices with limited memory and computation power. There have been constant efforts to develop smaller versions of BERT without significant compromise on its performance.

ALBERT[26], standing for A Lite BERT, is one of the recent achievement that reduces the model parameter number considerably and at the same time improves performance. In ALBERT , there are three major improvements on the model architecture and the pretraining process. The first two improvements on the model architecture are

- **Factorized embedding parameterization**. In the original BERT, tokens (represented as one-hot vectors) are directly projected the hidden space with dimensionality $H = 768$ or 1024. For large vocabulary size V at the scale of 10,000, the projection matrix W has parameters HV. One way to reduce parameter size is to factorize W into two lower rank matrices. This is equivalent to two-step projection:
  - First project one-hot vector to embedding space of dimensionality E, say $E = 128$;
  - Second project the embedding space to the hidden space of dimensionality H.

  The two-step projection only requires parameters $VE + EH$ and the reduction is notable when $E \ll H$.
- **Cross-layer parameter sharing**. In the original BERT, each encoder sub-unit (which has an attention module and a feed-forward module) has different parameters. In ALBERT, these parameters are shared in different sub-units to reduce model size and improve parameter efficiency. The authors also mentioned that there are alternative parameter sharing strategies, for example, only sharing feed-forward network parameters or only sharing attention module parameters. could also be

The effect of embedding factorization and parameter sharing can be illustrated in the comparison [Table 38.4.1].

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

Table 38.4.1: Comparison of model parameter size for different configurations of BERT and ALBERT models.

The third improvement in ALBERT over BERT is a new loss in next sentence prediction task. In ALBERT, a new sentence-order prediction (SOP) loss to model inter- sentence coherence, which demonstrated better performance in multi-sentence encoding tasks.

38.5 Notes on Bibliography

38.5.1 Books and references

General books on Natural language processing include [27][28][10]. Practical books include [27][10].

For a comprehensive review on deep learning methods for text classification, see [29].

38.5.2 Software

NLTK and **spaCy** are two popular packages widely used for pre-processing tasks such as tokenization, stemming,lemmatization, stopwords removal, parts of speech tagging.

BIBLIOGRAPHY

1. Vajjala, S., Majumder, B., Gupta, A. & Surana, H. *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems* ISBN: 9781492054009 (O'Reilly Media, 2020).
2. Dumais, S. T. Latent semantic analysis. *Annual review of information science and technology* **38,** 188–230 (2004).
3. Arora, S., Li, Y., Liang, Y., Ma, T. & Risteski, A. A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics* **4,** 385–399 (2016).
4. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
5. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. *Distributed representations of words and phrases and their compositionality* in *Advances in neural information processing systems* (2013), 3111–3119.
6. Press, O. & Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859* (2016).
7. Inan, H., Khosravi, K. & Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* (2016).
8. Pennington, J., Socher, R. & Manning, C. D. *Glove: Global vectors for word representation* in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), 1532–1543.
9. Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* **5,** 135–146 (2017).
10. Goldberg, Y. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies* **10,** 1–309 (2017).
11. Radford, A. *et al.* Language models are unsupervised multitask learners. *OpenAI Blog* **1,** 9 (2019).
12. Chen, S. F. & Goodman, J. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* **13,** 359–394 (1999).
13. Jurafsky, D & Martin, J. *Speech & Language Processing, an Introduction to NL Processing, Computational Linguistics & Speech Recognition* 2000.

14. Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. A neural probabilistic language model. *Journal of machine learning research* **3,** 1137–1155 (2003).

15. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J. & Khudanpur, S. *Recurrent neural network based language model.* in *INTERSPEECH* (eds Kobayashi, T., Hirose, K. & Nakamura, S.) (ISCA, 2010), 1045–1048.

16. Qiu, X. *et al.* Pre-trained Models for Natural Language Processing: A Survey. arXiv: 2003.08271 (2020).

17. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

18. Wu, Y. *et al.* Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

19. Li, J., Tu, Z., Yang, B., Lyu, M. R. & Zhang, T. Multi-head attention with disagreement regularization. *arXiv preprint arXiv:1810.10183* (2018).

20. Xiong, R. *et al. On layer normalization in the transformer architecture* in *International Conference on Machine Learning* (2020), 10524–10533.

21. Peters, M. E. *et al.* Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* (2018).

22. Liu, Y. *et al.* Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

23. Wang, A. *et al.* GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

24. Williams, A., Nangia, N. & Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426* (2017).

25. Rajpurkar, P., Zhang, J., Lopyrev, K. & Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

26. Lan, Z. *et al.* ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, 1–17. arXiv: 1909.11942 (2019).

27. Manning, C. D. & Schütze, H. *Foundations of statistical natural language processing* (MIT press, 1999).

28. Jurafsky, D., Martin, J., Norvig, P. & Russell, S. *Speech and Language Processing* ISBN: 9780133252934 (Pearson Education, 2014).

29. Minaee, S. *et al.* Deep Learning Based Text Classification: A Comprehensive Review. *arXiv preprint arXiv:2004.03705* (2020).

39

NATURAL LANGUAGE PROCESSING II: TASKS

39.1 Text classification and sentiment analysis

39.1.1 Introduction

Text classification tasks are one of the most fundamental tasks in NLP. Sentiment analysis or emotion classification, news classification, topic classification, question and answer matching, intent recognition all use the relevant knowledge or technology of text classification.

The textual content to be classified can come from a wide range of document types, including web-pages, emails, books, news stories, scholarly papers, text messages, Powerpoint presentations legal proceedings, medical records, etc.

According to the classification label, classification tasks can be divided into binary-classification, multi-classification, and multi-label classification; Depending on the length of text or document to be classified, classification tasks can be divided into sentence classification and long document classification. Based on the degree of class imbalance of the data, it can also be divided into ordinary classification problems and imbalanced classification problems(e.g., anomaly detection).

Text data is unstructured and needs to go through a couple of pre-processing steps or feature engineering before being fed into a model classifier for both training and inference Figure 39.1.1.

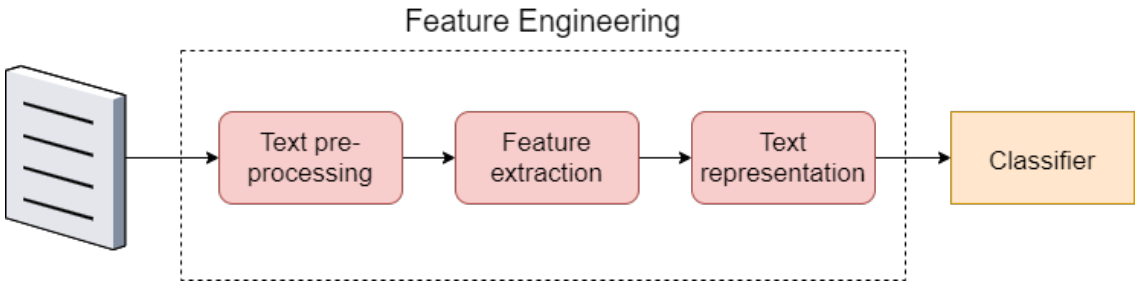


Figure 39.1.1: A typical workflow for text classification tasks.

39.1.2 Example tasks

39.1.2.1 *Sentiment analysis*

We now consider a sentiment analysis task where we are given a movie review text and we need determine if the review's sentiment is positive or negative.

Some negative example reviews are:

- this was an impulse pick up for me from the local video store . don t make the same mistake i did . this movie is **tedious unconvincingly acted** and generally **boring**. the dialogue between the young ...
- yes there are great performances here . unfortunately they happen in the context of a movie that doesn t seem to have a clue what it s doing . during the first minutes of this all the music ta...
- i saw this movie with some indian friends on christmas day . the quick summary of this **movie is must avoid** . jp dutta wrote directed produced and edited this movie and did none of these jobs well.
- this movie is **extremely boring** it tells a story of a female gas station owner and her life.**nothing exciting** ever happens . the director has really kept it real and it feels just like a camera fol...

And some positive example reviews are

- his scary and rather gory adaptation of stephen king s **great** novel features **outstandin**g central performances by dale midkiff fred gwynne who sadly died few years ago and denise crosby and some ...
- okay i ll confess . this is the movie that made me love what michael keaton could do . he does a beautiful parody of someone doing a parody of james cagney with **charm** to spare.
- directors had the same task tell stories of love set in paris . naturally some of them turned out better than others but the whole mosaic is **pretty charming** besides wouldn t it be boring if

39.1.2.2 *Document category*

Besides classifying sentiment of a review or comment, another common text classification is to classify document category, such as new's tag. Following shows the example news abstracts.

- (financial) Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers, Wall Street's dwindling band of ultra-cynics, are seeing green again.
- (financial) Carlyle Looks Toward Commercial Aerospace (Reuters) Reuters - Private investment firm Carlyle Group, which has a reputation for making well-timed and occasionally controversial plays in the defense industry, has quietly placed its bets on another part of the market.
- (financial) Oil and Economy Cloud Stocks' Outlook (Reuters) Reuters - Soaring crude prices plus worries about the economy and the outlook for earnings are expected to hang over the stock market next week during the depth of the summer doldrums.
- (science&technology) T. Rex Had Teen Growth Spurt, Scientists Say (Reuters) Reuters - Tyrannosaurus Rex grew incredibly fast during a teenaged growth spurt that saw

the dinosaur expand its bulk by six times, but the fearsome beasts "lived fast and died young," researchers said on Wednesday.
- (science&technology) Gene Blocker Turns Monkeys Into Workaholics - Study (Reuters) Reuters - Procrastinating monkeys were turned into workaholics using a gene treatment to block a key brain compound, U.S. researchers reported on Wednesday.
- (science&technology) Dolphins Too Have Born Socialites (Reuters) Reuters - Some people are born to be the life and soul of the party – and so it seems are some dolphins.

39.1.3 Text preprocessing

39.1.3.1 *Basic steps*

Text contents can come in various format depending on the document type (e.g., web-pages, emails). Text pre-processing protocols often highly depend on the document type as well as the domain.

The first step is to think about the paragraph size we need to handle. Textual content can be as long as a novel and can also be as short as a sentence. The paragraph size is ultimately determined by the task itself. Paragraphs can be obtained by segmenting the textual content by punctuation, by conversational sessions, or by paragraphs under a heading.

After segmenting documents into chunks, the typical next step is **tokenization**, which splits the sentence into words. According to the document type, we also need to remove special symbols, hyperlinks, etc. that do not convey information relevant to the task. We might also need to **lower case** all words if capitalization is irrelevant to the task. Finally, we often **remove punctuation**.

For tasks that requires handling large quantities of word phrases (e.g., locations, people's name) we might also need to convert phrases to single words. For example, *New York* will be preprocessed to a single token *New_York*.

39.1.3.2 *Spell correction and rare word replacement*

Textual content like product remark, messages often contains spelling mistakes. Articles may also contain rare words and acronyms. To improve model robustness and reduce vocabulary size, we can perform spell correction and rare word replacement. We can simply replace all rare words by an UNK token, or in a finer-grained, replace them by an UNK token plus its part-of-speech tag, such as UNK-NN, UNK-VERB, etc.

39.1.3.3 *Stop words removal*

When we work with words as feature input to an NLP task, we sometimes need to exclude some very frequent words that do not bring any substantial meaning to a sentence, words such as *but, can, we, the, a, an*, and so on. For example, some search engines may be programmed to ignore stop words such as *the, a, an, in*, both during the stage of indexing entries for searching and the stage of retrieving them as the result of a search query.

Following are stopwords in English language from `nltk`:

i me my myself we our ours ourselves you you're you've you'll you'd your yours yourself yourselves he him his himself she she's her hers herself it it's its itself they them their theirs themselves what which who whom this that that'll these those am is are was were be been being have has had having do does did doing a an the and but if or because as until while of at by for with about against between into through during before after above below to from up down in out on off over under again further then once here there when where why how all any both each few more most other some such no nor not only own same so than too very s t can will just don don't should should've now d ll m o re ve y ain aren aren't couldn couldn't didn didn't doesn doesn't hadn hadn't hasn hasn't haven haven't isn isn't ma mightn mightn't mustn mustn't needn needn't shan shan't shouldn shouldn't wasn wasn't weren weren't won won't wouldn wouldn't

Note that according to the specific NLP tasks, we often need to curate a list of stopwords that are least important to task to prevent removing important words from the text.

39.1.3.4 *Lemmatization and stemming*

Stemming and Lemmatization are two commonly used pre-processing steps to generate the root form of the inflected words. They are widely applied in NLP tasks, such as text clustering, entity extraction, sentiment analysis, document summarization, and entity relation modeling, etc.

Stemming is to extract the stem or root form of a word by chops off some suffix or sometimes prefix of a word based on some crude heuristics. The resulting word might not be a valid word in the vocabulary and stemming is often not able to express the complete semantics of the original one. For example, using Porter's stemming algorithm[1],

- *trouble, troubling, troubled* all reduce to *troubl*.
- *connect, connections, connected, connection* all reduce to *connect*.

Lemmatization aims to reduce variants of a word to its base form. Lemmatization usually involves the use of a vocabulary and morphological as well as part-of-speech (e.g., verb or noun) analysis of words. As a result, lemmatization is typically a more sophiciated

process than stemming and preserves better semantics of the original word. For example, *connect*, *connected*, *connection*, *connections* reduce to *connect*, *connected*, *connection*, *connection*, respectively.

In practical applications, choosing one over the other highly depends on the application requirement. if speed is focused then stemming should be used since lemmatizers scan a corpus which consumed time and processing. Also, stemming is more used in the field of information retrieval, particularly for the first-stage coarse retrieval. Lemmatization is more often used in NLP tasks required more fine-grained and more accurate text analysis.

39.1.4 Text classification models

39.1.4.1 *Bag of words baselines*

We will first introduce a baseline model - bag-of-words, which is actually a very good fit for long documents despite its simplicity. When the document is long, we would expect the characteristic words associated with its document type to overwhelm the others.

To understand this intuition, we look at some sentiment analysis examples [subsubsection 39.1.2.1].

Clearly, the existence of words like **tedious, boring** will likely be found in a negative review; similarly, the existence of words like **outstanding, charm** will likely be found in a positive review. Our goal is to define a neural network that can automatically establish such relation.

An initial analysis is to identify these words that are hallmarks of positive and negative reviews. Our method is to first count the word frequency for positive reviews and negative reviews, respectively. The most frequent words in both positive and negative review are actually neural stop words like *the*, *I*, etc..

Let w_i^P and w_i^N be the frequency of word i shown in positive and negative reviews respectively. We instead of show the words i with highest ratio of w_i^P / w_i^N and lowest ratio of w_i^P / w_i^N with at least 100 counts. They are expected to be most distinguishing words that determine the sentiment of review.

The result in the following table agrees with our expectation that words like *superb* and *worst* are the key words to determine the sentiment.

Positive	Negative
superb	worst
wonderful	awful
fantastic	horrible
excellent	crap
amazing	worse
powerful	terrible
favorite	mess
perfect	stupid
brilliant	dull

To build a bag-of-words model, we first need text preprocessing steps like stopword removal, lemmatization, etc. After that, we can simply count the word frequency of each document with respect to a vocabulary. The vocabulary typically only keeps words that have certain number of count to avoid overfitting issue.

The word count of a document can be viewed as a feature vector with dimensionality of the vocabulary size. The feature vector can be fed into a linear classifier (multinomial logistic regression, SVM, etc.) or a Naive Bayes model.

Bag-of-words may fall short in tasks where logically complex sentences are more frequent. For example,

- *The movie is not very good, but i still like it.*
- *The movie is very good, but i still do not like it.*
- *I do not like it, but the movie is still very good.*

More complex ones:

- *This is not the worst thing that can happen.*
- *It would be nice if you acted like you understood.*
- *There is no reason at all to believe that the polluters are suddenly going to become reasonable.*

Instead of only counting words, we can also enhance the model via a bag-of-bigrams, where features of adjacent words are consider. For example, (that's, not), (not,*bad*), (*bad*, for), . . . Bigrams can handle relatively straightforward cases, such as when an adjective is immediately negated; trigrams would be required to extend to larger contexts (e.g., not the worst).

A further improvement is to decrease the weight for words that are frequent across all documents and increase the weight for words that are particularly unique to a document. This leads to the well-known tf-idf feature, which is defined as

tf stands for term frequency, $\text{tf}(t, d)$,

$$\text{tf}(t, d) = \frac{c_{t,d}}{\sum_{t' \in d} c_{t',d}}$$

where $f_{t,d}$ is the count of a term in a document, i.e., the number of times that term t occurs in document d.

idf stands for inverse document frequency of a word. idf measures how much information a word provides. The most frequent the words appears across multiple documents, the less informative it is. It is defined by

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where D is the corpus containing all the documents. and $|\{d \in D : t \in d\}|$ is number of documents where the term t appears in the document.

The tf-idf feature assumes that the most meaningful words to distinguish documents are words that appear frequently in the document, but appear less frequently in other documents in the entire document collection. In extreme cases, rare, unique words could be the most important hallmark of a document. This assumption does not completely hold for many cases.

In addition, the position information of words is not reflected in the tf-idf algorithm. For Web documents, structural characteristics of HTML (i.e., headings) also conveys extra information, but a simple tf-idf feature does not take it into account.

39.1.4.2 *Fasttext text classification*

Bag-of-words model relies on a vocabulary to construct the word count so it naturally suffers from OOV issue. In addition, word count does not consider semantic meanings and relations among similar word, as such words of similar meanings cannot share their statistical strength. For example, words like *excellent, terrific, wonderful* are treating as distinctive feature in the bag-of-words model.

One way to address the two shortcomings is to use some sort of bag-of-subword-embeddings model, where subword can address the OVV issue and embeddings can capture semantics. One efficient off-the-shelf of such text classification model is from Facebook AI research Fasttext [2]. In short, fastText text classification architecture [Figure 39.1.2] is to turn all words in the document into dense vectors through the lookup table, take average over all word vectors, and then feed average vector into a linear classifier to get the classification result. The subword decomposition is directly from the procedure described in subsection 38.2.5. The embedding representation for each word is a sum of embeddings of its constituent character n-grams, i.e., embeddings of its constituent subwords. The approach can handle the OOV issue and the implementation is memory efficient and extremely fast and on even very large corpora.

Such drawback can be alleviated by feeding embeddings into the recurrent neural network.

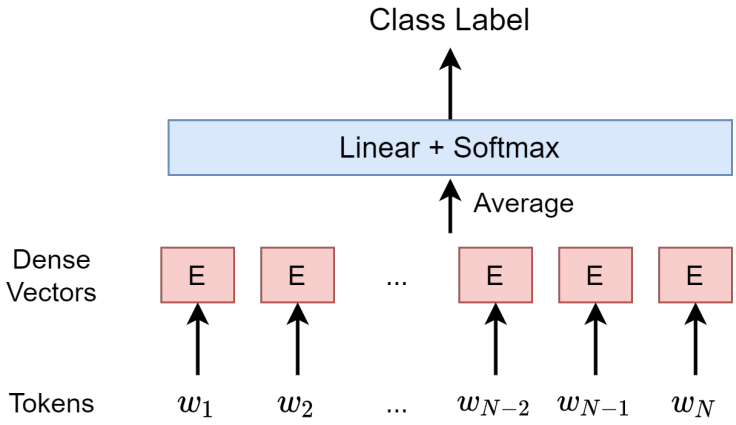


Figure 39.1.2: Fasttext classification architecture.

39.1.4.3 *Word level CNN*

The primary playgrounds of CNNs are image analysis and understanding. The local connectivity in CNN suits spatial locality in images and the multi-layer of filters enable progressive extraction of high-levels features from low-level raw pixel data [subsection 34.4.1]. The ability of CNN to extract high-feature from local low-level features also sparks novel application in language related application, such as sentence classification[3].

We first look at how CNN filters can act as feature extractor for sentences. Consider a sentence consisting of N words, with each word represented by a D-dimensional vector $x_i \in \mathbb{R}^D$. In general x_i are low-dimensional dense word embeddings [section 38.2]. If we view a sentence as a stack of row vectors of its constituent words, then the representation of a sentence is similar to an image with size $N \times D$ of a single channel. If we apply a convolutional filter with kernel size $k \times D$, then the output will be a one-dimensional feature vector of length $N - k + 1$ [Figure 39.1.3].

Mathematically, the CNN filter extracts a new feature vector from a word sequence of window h given by

$$c_i = f\left(\sum_{j=1}^{h} W x_{i+j-1} + b\right),$$

where W is the kernel matrix, b is the bias, and f is a non-linear activation function. In models proposed in [3], CNN filters have different window sizes $h = 3, 4, 5$, with 100 for each of them. Intuitively, these CNN filters are extracting features from 3-grams, 4-grams, and 5-grams. All these CNN extracted features are then pooled and combined to feed into feed-forward layers for sentence classification [**??**].

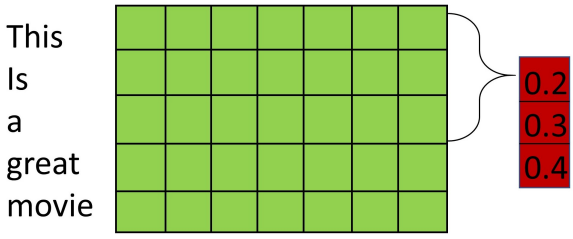


Figure 39.1.3: Demonstration of a CNN filter applying to a sentence to produce a one-dimensional feature vector.

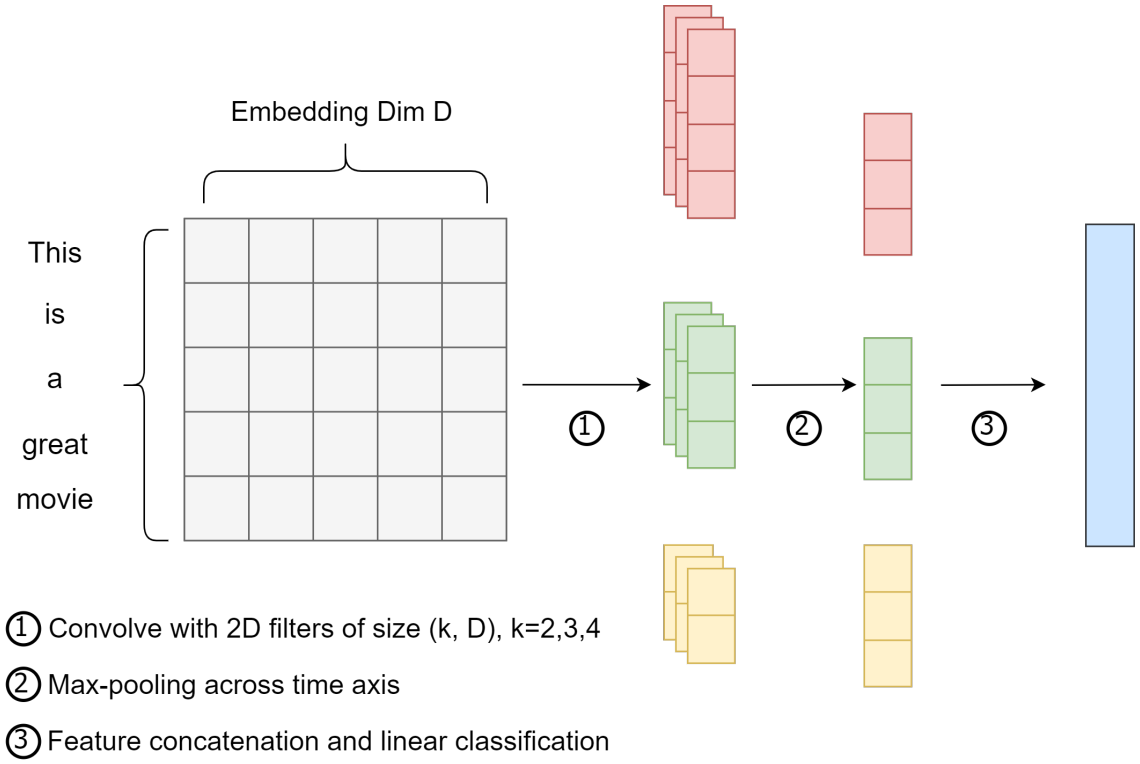


Figure 39.1.4: 2D CNN for sentence classification.

Also, we can treat the input sequence feature $[x_1^T; ...; x_N^T] \in \mathbb{R}^{N \times D}$ as a multi-channel signal with the number of channels being D. To mimic the 2D convolution feature extraction process, we can perform 1D convolution with different kernel sizes on the time axis. The architecture is summarized in Figure 39.1.5.

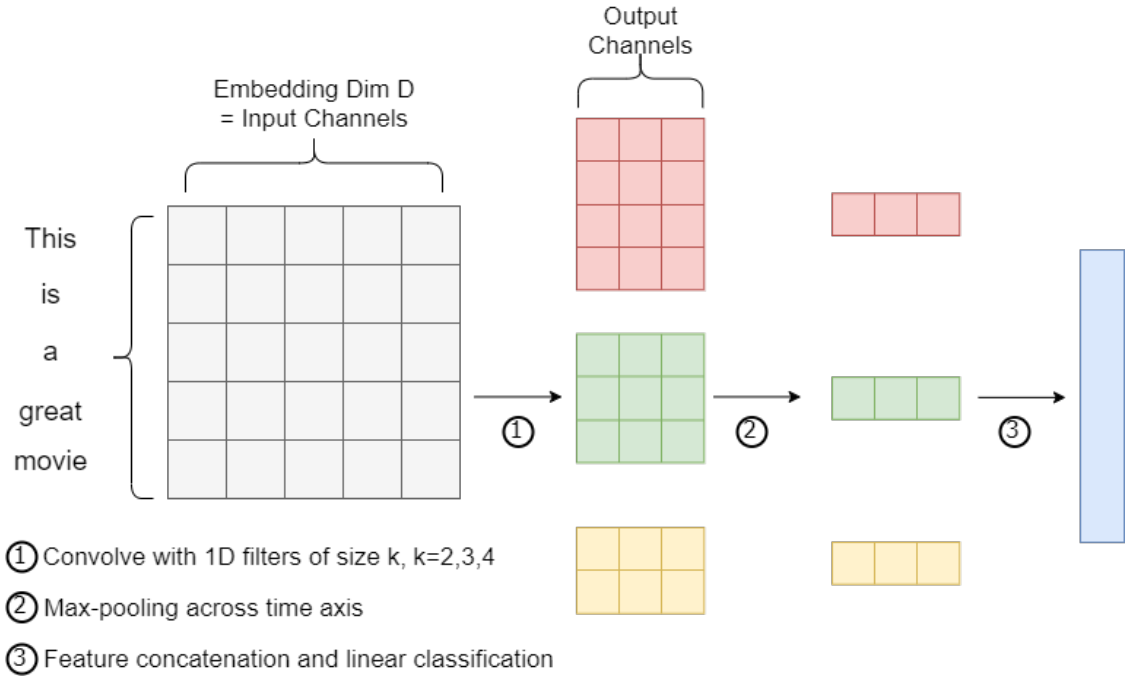


Figure 39.1.5: 1D CNN for sentence classification.

39.1.4.4 *Character level CNN*

Word-level CNN achieves excellent performance on text classification task. However, since it takes sentence features from the word level, it naturally suffers from the out-of-vocabulary (OOV) problem. Besides OOV, typical text normalization pre-processing steps also removes morphological variations that conveys sentiment information. For example, *tooooo bad* signals a strong negative sentiment. The character level CNN model is proposed to address the limitation.

Textual content is converted to character level representation according to an uncased alphabet. The alphabet used in all of our models consists of 70 characters, including 26 english letters, 10 digits, 33 other characters and the new line character. The non-space characters are:

```
abcdefghijklmnopqrstuvwxyz0123456789-,;.!?:'\"/\\|_@#$%^&*~`+ =<>()[]{}
```

Different from word level CNN, where each word has a dense vector representation, each character here takes a one-hot vector representation. We can view a sequence of characters as a multi-channel signal with the number of channels being the alphabet size (or the size of the one-hot vector). The CNN architecture to extract feature is summarized in Figure 39.1.6. Compared to word-level CNN, character CNN has to be much deeper in order to capture high-level semantic information. Typically, around 10 CNN and pooling

layers are used to extract high-level textual features. After that, multi-layer feed-forward neural network is used to perform classification.

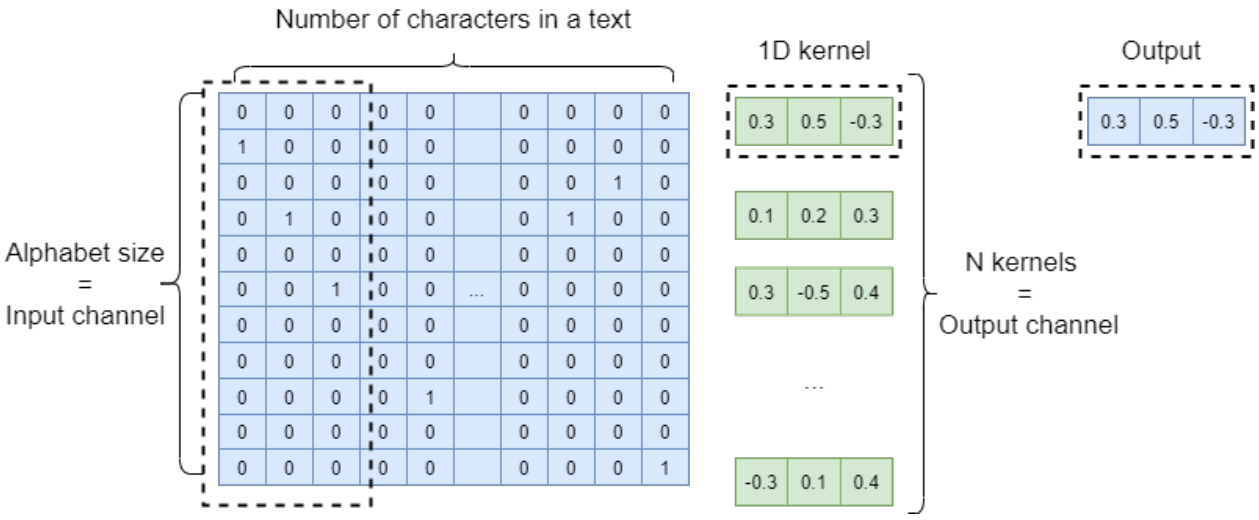


Figure 39.1.6: Character level CNN for text classification tasks.

39.1.4.5 *RNN*

Sentiment analysis via simple bagging solutions [subsubsection 39.1.2.1] does not take into the semantic meaning conveyed by the order of words. For example, sentences like 'do not miss this good movie' and 'this is not a good movie' have opposite sentiments even they have similar word counts.

Using RNN for sentiment analysis, on the other hand, can capture complex semantic meaning in the sequence of words in the classifier. The overall architecture [Figure 39.1.7] contains the following components:

- An embedding layer that convert one-hot vectors of words to efficient low dimensional representations. The embedding layer can have initialized weight from some pretrained models.
- LSTM layers that process sequences of embedded word vectors and capture dependence between words.
- The last output from the LSTM will be fed into a Sigmoid output layer. Sigmoid output is used as the probability of positive or negative labels.

Before feeding the data to the network, we also need some basic data cleaning. Unlike traditional machine learning methods, we here only need minimal feature engineering.

- We need to get rid of punctuation such as comma, periods, etc.

- To facilitate batch processing, we also like to make all input sequence have the same size. For longer sequences, we will truncate the size; for shorter sequences, we will left pad dummy one-hot encoding.

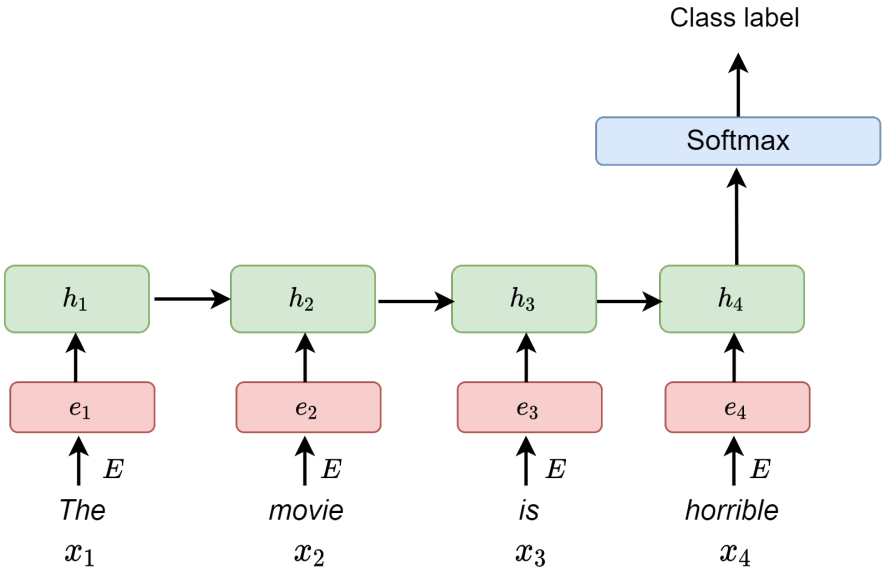


Figure 39.1.7: RNN architecture for textual classification.

39.2 Sequence labeling

39.2.1 Sequence labeling tasks

39.2.1.1 *Named entity recognition*

Named-entity recognition (NER) is the task identifying named entities from a text paragraph or a document. A named entity is typical a person, a location, an organization with proper names that we can refer to. For example,

NER is also the basis for many NLP tasks such as word segmentation, relationship extraction, event extraction, knowledge graphs, machine translation, and question answering systems. For example, by tracking specific entities in the text, relationship between entities can be extracted and analyzed. Further more, knowledge graph through the association between named entities can be constructed for provide better answers in the question answering system.

In real-world applications, a named entity also includes date, time, product name, domain-specific terms, etc. Typically, date and time can be generally identified by format matching. NER systems extract the entities from unstructured input text, including product names, models, prices, etc., according to business needs. Therefore, the concept of entity can be very broad, as long as it is a special text fragment required by the business, it can be called an entity. In interacting with conversational agents, like Amazon Alexa and Google assistant, NER is also used to identify intent and relevant information (such as date, singer's name, music name) in the user's command.

In English, a named entity can consist of multiple words, such as *World Health Organization*. In the practical tagging task, we attach a tag to each word, where tags come from the following categories.

An example of a tagged sentence is as follows.

$$\underbrace{\text{The}}_{\text{O}}\ \underbrace{\text{World}}_{\text{B-ORG}}\ \underbrace{\text{Health}}_{\text{I-ORG}}\ \underbrace{\text{Organization}}_{\text{I-ORG}}\ \underbrace{\text{a specialized agency of the}}_{\text{O}}\ \underbrace{\text{United}}_{\text{B-ORG}}\ \underbrace{\text{Nations}}_{\text{I-ORG}}.$$

One of the challenges of NER includes the difficulty to distinguish the boundaries of the named entity. For example, consider *First National Bank has donated 1 Billion to*

Type	Tag	Example sentences
People	PER	**Turing** is a giant of computer science.
Organization	ORG	The IPCC warned about the cyclone.
Location	LOC	Mt. Sanitas is in Sunshine Canyon.
Geo-Political Entity	GPE	Palo Alto is raising the fees for parking.

Tag	Meaning
O	Not part of a named entity
B-PER	First word of a person name
I-PER	Continuation of a person name
B-LOC	First word of a location name
I-LOC	Continuation of a location name
B-ORG	First word of an organization name
I-ORG	Continuation of an organization name
B-MISC	First word of another kind of named entity
I-MISC	Continuation of another kind of named entity

Table 39.2.1: Common used entity category in NER tasks.

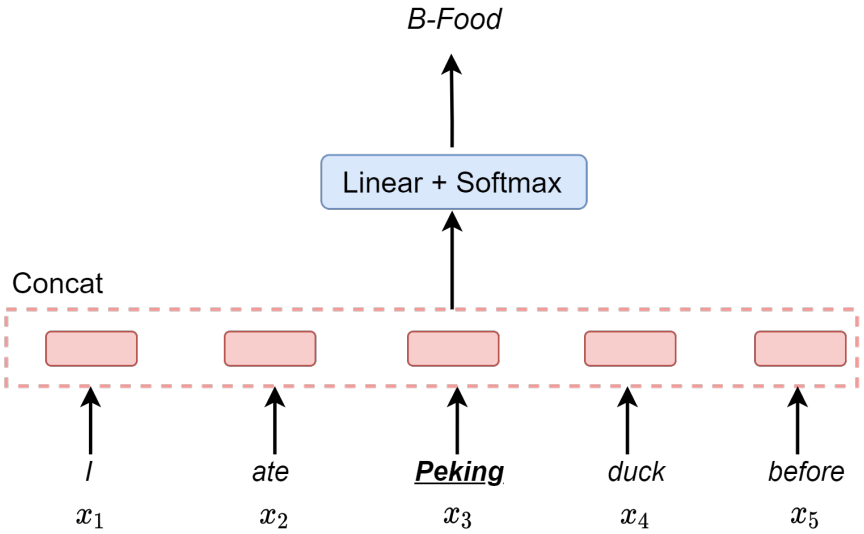


Figure 39.2.1: A basic window classification model for the NER task.

community education this year. Here both *First National Bank* and *National Bank* are named-entities. Another challenge is that named entity depends on the context. The same noun may be an organization name in some contexts. In other contexts, it is the name of the person. For example, the word *Milan* can be a person's name or the football club.

A very basic model, known as **window classifier**, to perform word tagging with word context is shown in Figure 39.2.1. Each word and its nearby tokens are first encoded by their word embeddings. Embeddings are then concatenated and fed into a linear classifier. Because of the context, Peking is tagged as an food entity, instead of a location entity.

One issue with the window classifier is that the process of labeling each token is carried out independently, and the previously predicted label cannot be directly used to

predict the next label. As a result, the resulting sequence can be invalid. For example, the label I-ORG follows B-PER. In our following sections, we will discuss models such as hidden Markov model and conditional random fields to Leverage nearby label information to enhance prediction accuracy.

39.2.1.2 *Part-of-speech tagging*

In plain language, Part-of-Speech (POS) tagging is about identifying in a sentence, what words act as nouns, pronouns, verbs, adverbs, and so on.

In general, POS tagging in itself may not be the direct solution to a particular NLP problem. But POS tagging can usually provide additional information or simplification to facilitate various NLP tasks. POS tagging finds wide applications in Named entity recognition (NER), sentiment analysis, question answering, and word sense disambiguation. Take word sense disambiguation as an example. In the sentences *I left the room* and *Left of the room*, the word *left* conveys different meanings. A POS tagger would help to differentiate between the two meanings of the word left.

What makes POS tagging complicated is its context dependency. It is quite possible for a single word to have a different POS tags in different sentences based on different contexts. For example, the word *love* in *I love dogs* and *feelings of love* have different tags.

Consider another example *John saw the saw*. The first *saw* is more likely to be a verb rather than a noun as it follows a *noun*. However, the second *saw* is a noun because a noun is more likely to follow a determiner. POS tagging cannot be solved without considering its context.

39.2.2 HMM for POS tagging

39.2.2.1 *The model*

In the POS tagging, we are given a sequence of token $x = (x_1, ..., x_T)$ as observations and we aim to tag each token and get the tag sequence $y = (y_1, ..., y_T)$. Each y_t is selected from the tag space including ADV, NOUN, JJ, etc. [Table 39.2.2]. In HMM, we make *Markov assumption* and *independent observation assumption* to model the joint distribution $P(x_{1:T}, y_{1:T})$. Specifically,

- Markov assumption: $P(y_t|y_{t-1}, y_{t-2}, ...) = P(y_t|y_{t-1})$;
- Independent observation assumption: $P(x_t|x_{1:T}, y_{1:T}) = P(x_t|y_t)$.

Tag	Description	Example
ADJ	Adjective: noun modifiers describing properties	*red, young, awesome*
ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
VERB	words for actions and processes	*draw, provide, go*
PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
ADP	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by under*
AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
DET	Determiner: marks noun phrase properties	*a, an, the, this*
NUM	Numeral	*one, two, first, second*
PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause	*that, which*

Table 39.2.2: Universal Dependencies part-of-speech tags

With these two assumptions we have

$$P(x_{1:T}, y_{1:T}) = \prod_{t=1}^{T} P(x_t|y_t)P(y_t|y_{t-1}),$$

where for simplification we use $P(y_1|y_0) = \pi(y_1)$ and π is the prior distribution of y.

The probabilities of $P(y_t|y_{t-1})$ are known as transition probability model, which specifies the transitions among different tags. One way of estimating the transition probability is through counting in the training annotated training corpus. Specifically,

$$P\left(y_i|y_{i-1}\right) = \frac{\#\left(y_{i-1}, y_i\right)}{\#\left(y_{i-1}\right)}$$

The probabilities of $P(x_t|y_t)$ are known as emission probability model, which can be similarly estimated via counting, given by

$$P\left(x_i|y_i\right) = \frac{\#\left(x_i\right)}{\#\left(y_i\right)}.$$

39.2.2.2 *Decoding*

Bayes' Theorem gives the condition probability

$$P(y_{1:T}|x_{1:T}) = \frac{P(x_{1:T}|y_{1:T})}{P(x_{1:T})}$$

which is used for decoding $y_{1:T}$ given observation $x_{1:T}$.

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence $y_1 \dots y_T$ that is most probable given the observation sequence $x_1 \dots x_n$

$$\hat{y}_{1:T} = \underset{y_1,\dots,y_T}{\arg\max}\, P\left(y_{1:T}|x_{1:T}\right)$$

Using Bayes' Theorem gives the condition probability We have

$$\begin{aligned}\hat{y}_{1:T} &= \underset{y_1,\dots,y_T}{\arg\max}\, P\left(y_{1:T}|x_{1:T}\right) \\ &= \underset{y_1,\dots,y_T}{\arg\max}\, \frac{P\left(y_{1:T}, x_{1:T}\right)}{P\left(x_{1:T}\right)} \\ &= \underset{y_1,\dots,y_T}{\arg\max}\, P\left(y_{1:T}, x_{1:T}\right)\end{aligned}$$

where we remove the $P(x_{1:T})$ since they do not depend on y.

We use dynamic programming (also known as Viterbi algorithm) to solve the optimal sequence. Let $V_t(y_t)$ be a function given by.

$$V_t(y_t = j) = \max_{y_{1:t-1}} P(x_1, ..., x_t, y_1, ..., y_{t-1}, y_t = j).$$

We have a recursive relationship

$$V_t(y_t = j) = \max_{y_{t-1}} V_{t-1}(y_{t-1}) P(y_t = j|y_{t-1}) P(x_t|y_t = j), \forall j \in Y$$

with initialization

$$V_1(y_1 = j) = \pi(y_1 = j) P(x_1|y_1 = j),$$

where π is the prior distribution of y_1.

With V_t solved, we can further solve optimal y_t^* via

$$y_t^* = \underset{y_t \in Y}{\arg\max}\, V_t(y_t).$$

The time complexity of the Viterbi algorithm is $O(|Y|^2 T)$. Beam search is a truncated version of Viterbi algorithm where in each V_t function, only top B best hypotheses are kept. The parameter B is the beam width. Beam search has a time complexity of $O(B^2 T)$. However, beam search might not find the global optimal label sequence.

39.2.3 Conditional random field

39.2.3.1 *Formulation*

Given sequences $x = (x_1, ..., x_T)$ and $y = (y_1, ..., y_T)$, the conditional random field (CRF) is a discriminative model that directly models the conditional probability $P(y|x)$, given by

$$p(y|x) \approx \exp\left(\sum_{k=1}^{K} \lambda_k F_k(x, y)\right)$$

where $F_k, k = 1, ..., K$ are K feature functions that take x, y as input and produces a scalar and $\lambda_k, k = 1, ...K$ are feature multipliers that control the contribution of each feature to $p(y|x)$ in terms of their directions (i.e., positive impact or negative impact) and strengths. A normalized version $p(y|x)$ is given by

$$p(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{k=1}^{K} w_k F_k(x, y)\right)$$
$$Z(X) = \sum_{y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k\left(x, y'\right)\right)$$

CRF can be view as the (multinomial) logistic regression in sequence-labeling domain, where the log conditional probability of candidate output is modeled as proportional to linear combinations of features; that is,

$$\log p(y|x) \approx \sum_{k=1}^{K} w_k F_k(x, y).$$

The major difference is that CRF is operating over the sequence space, whose total candidate output class for y is $\sim O(TD)$, where T is the sequence length and D is number of choices on each y_t.

Remark 39.2.1. CRF also offers modeling flexibility with respect HMM framework 2 Because in HMMs all computation is based on the two probabilities $P(\text{tag} \mid \text{tag})$ and $P(\text{word} \mid \text{tag})$, if we want to include some source of knowledge into the tagging process, we must find a way to encode the knowledge into one of these two probabilities. Each time we add a feature we have to do a lot of complicated conditioning which gets harder and harder as we have more and more such features.

How feature functions $F_k(x, y)$ are specified is not only of practical importance to the problems we are solving but also affect training and inference stage. Typically, we have following categories

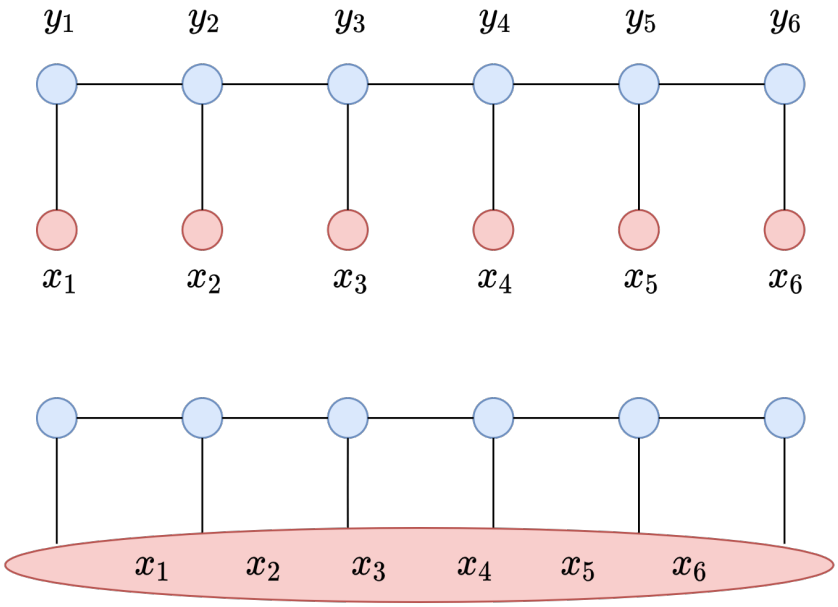


Figure 39.2.2: Linear conditional random field architectures. (upper) Each output label only connects with the input token at the same step and its immediate left and right label. (lower) Each output label connects with all input tokens.

- Standalone features $F_k(x, y) = f_k(x_t, y_t)$.
- Local label features $F_k(x, y) = f_k(x_t, y_t, y_{t-1})$.
- Global features $F_k(x, y) = f_k(x, y_t)$.
- Non-local label features $F_k(x, y) = f_k(x_t, y_t, y_{t-1}, y_{t-2}, ...)$.

Notably, when feature functions only include standalone features, local label features, and global features, the CRF is known as linear CRF.

We typically avoid non-local label features with extensive dependencies on the label history, e.g., $f(y_{i-5}, ..., y_i)$, which would increase the difficulty of implementing dynamic programming techniques to speed up training and inference.

For streaming applications, the model receives observations $x_1, ..., x_t$ and produces labels $y_1, ..., y_t$ sequentially, we would also avoid features that has a dependency on future observations.

39.2.3.2 *Feature function choices*

In principle, feature function $F(x, y)$ can be arbitrary functions. For convenience, a feature function is specified as indicator functions, which gives value 1 when certain conditions is met, 0 otherwise.

In the POS tagging, example features include

- $f_{k,t}\left(y_t, x_t\right)$ if y_t is ADVERB and the word x_t ends with suffix *ly* and o otherwise. This is a standalone morphological feature that express the linguistic phenomenon that an adverb word has a higher chances ends with suffix *ly*. The contribution of this feature to conditional probability is determined by its feature multiplier.
- $f_{k,t}\left(y_t, x_t\right) = 1$ if y_t is DET and the word x_t is *the* or *a* or *an*; o otherwise. This is also a standalone feature that express the grammar phenomenon that *a, an, the* are determiners.
- $f_{k,t}\left(y_t, y_{t-1}\right) = 1$ if y_{t-1} is ADJ and y_t is NOUN; o otherwise. This is also a local label feature that express the linguistic phenomenon that a NOUN typically follows an ADJ.

For NER problems, we can engineer similar features based on common linguistic phenomenon. There is one brute feature, which simply checks if a word exists in the pre-complied dictionary (also known as gazetteer) that maps a word to an entity category.

39.2.3.3 *Training*

Here we consider the training of a linear CRF. Recall that the log conditional probability of y given x is specified by

$$\log p(y|x) = \left(\sum_{k=1}^{K}\sum_{t=1}^{T}\lambda_k f_{k,t}(x, y_t, y_{t-1})\right) - \log Z(X)$$

where $f_{k,t}$ are feature functions, $\lambda_k, k = 1, ..., K$ are learnable feature multipliers, and $Z(x) = \sum_{y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k\left(x, y'\right)\right)$.

The goal of training is to determine λ_k and one way to realize it is through gradient descent. Take gradient with respect to λ_k, we have

$$\frac{\partial}{\partial \lambda_k} \log p(y|x) = \sum_{t=1}^{T} f_k\left(s, y_j, y_{j-1}\right) - \sum_{y'} p\left(y'|x\right) \sum_{j=1}^{m} f_k\left(xy'_t, y'_{t-1}\right).$$

Intuitively, the first term promote the probability of observing sequence y and the second term reduces the probability of observing other sequence y. Note that the summation in the term can be computed via dynamic programming to reduce redundant re-computing.

39.2.3.4 *Inference on CRF*

How do we find the best tag sequence $\hat{y}$ for a given input x? We start with

$$\begin{aligned}\hat{y} &= \underset{y \in \mathcal{Y}}{\operatorname{argmax}} P(y|x) \\ &= \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} \lambda_k F_k(x, y)\right) \\ &= \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \exp\left(\sum_{k=1}^{K} \lambda_k \sum_{t=1}^{T} f_k\left(y_{t-1}, y_t, x_{1:t}\right)\right) \\ &= \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{k=1}^{K} \lambda_k \sum_{t=1}^{T} f_k\left(y_{t-1}, y_t, x_{1:t}\right) \\ &= \underset{Y \in y}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{k=1}^{K} w_k f_k\left(y_{t-1}, y_t, x_{1:t}\right)\end{aligned}$$

where in the above, we ignore exp and the constant denominator $Z(X)$ ($Z(X)$ is fixed for a given observation) as they won't change how we select y.

We use dynamic programming (i.e., Viterbi algorithm) to solve the optimal sequence. Let $V_t(y_t)$ be a function stores the optimal accumulated score (i.e., feature values) when the step t ends with label y_t. We have recursive relationship

$$V_t(y_t) = \max_{y_{t-1}} \sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, x_{1:t}) + V_{t-1}(y_{t-1}), \forall y_t \in Y.$$

Then the optimal label at time t is

$$y_t^* = \arg\max_{y_t} V_t(y_t).$$

To probably initialize $V_1(y_1)$, we use a dummy start of sequence label $< S >$ and set $V_1(y_1) = \sum_{k=1}^{K} f_k(y_0 =< S >, y_1)$. Often we set $V_1(y_1) = \pi(y_1)$, where π is the prior distribution of y

The time complexity of the Viterbi algorithm is $O(|Y|^2 T)$. We can also use Beam search with beam width B, which can reduce the time complexity to $O(B^2 T)$.

39.2.4 Neural sequence labeling

39.2.4.1 *BiLSTM tagging*

For sequence labeling tasks like NER, it is necessary for a neural model to consider both left and right context. If the model encounters an entity like *Johns Hopkins University*

and only attends to the left context, it will likely tag the Hopkins token as a name, because the model cannot see to the *university* token that appears on the right. One way to resolve this challenge is to introduce a bidirectional LSTM (BiLSTM) network [Figure 39.2.3] taking all words in a sequence as the input. The bidirectional LSTM consists of two LSTM networks - one takes the input in a forward direction, and a second one taking the input in a backward direction.

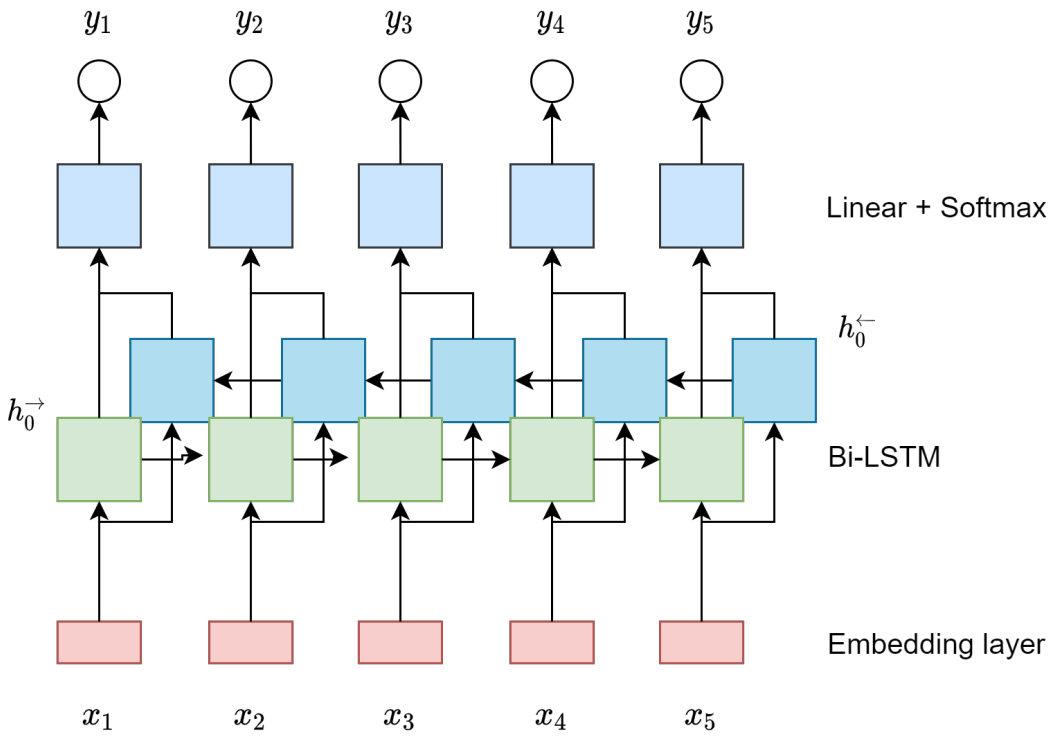


Figure 39.2.3: BiLSTM for sequence labeling tasks.

The model takes in a sequence of tokens, $X = \{x_1, x_2, \ldots, x_T\}$, passes them through an embedding layer, e, to get the token embeddings, $e(X) = \{e(x_1), e(x_2), \ldots, e(x_T)\}$ These embeddings are processed - one per time-step - by the forward and backward LSTMs. The forward LSTM processes the sequence from left-to-right, whilst the backward LSTM processes the sequence right-to-left, i.e. the first input to the forward LSTM is x_1 and the first input to the backward LSTM is x_T. The LSTMs also take in the the hidden, h, and cell, c, states from the previous time-step $h_t^{\leftarrow} = \text{LSTM}^{\leftarrow}\left(e\left(x_t^{\leftarrow}\right), h_{t-1}^{\leftarrow}, c_{t-1}^{\leftarrow}\right)$ After the whole sequence has been processed, the hidden and cell states are then passed to the next layer of the LSTM.

Both the forward and backward hidden states from the final layer of the LSTM are concatenated, and passed through a linear layer to predict the final label.

39.2.4.2 *BiLSTM-CRF tagging*

One issue with the BiLSTM classifier is that the prediction of a label is solely based on the token input and does not take into account other predicted labels. As a result, the resulting predicted label sequence can be invalid. For example, the label I-ORG follows B-PER. In our following sections, we will discuss models such as hidden Markov model and conditional random fields to Leverage nearby label information to enhance prediction accuracy.

The BiLSTM-CRF model[4] utilizes a linear CRF layer to model the relationship among predicted labels [Figure 39.2.4]. Experimental results show that BiLSTM-CRF has reached or surpassed the traditional CRF model and BiLSTM.

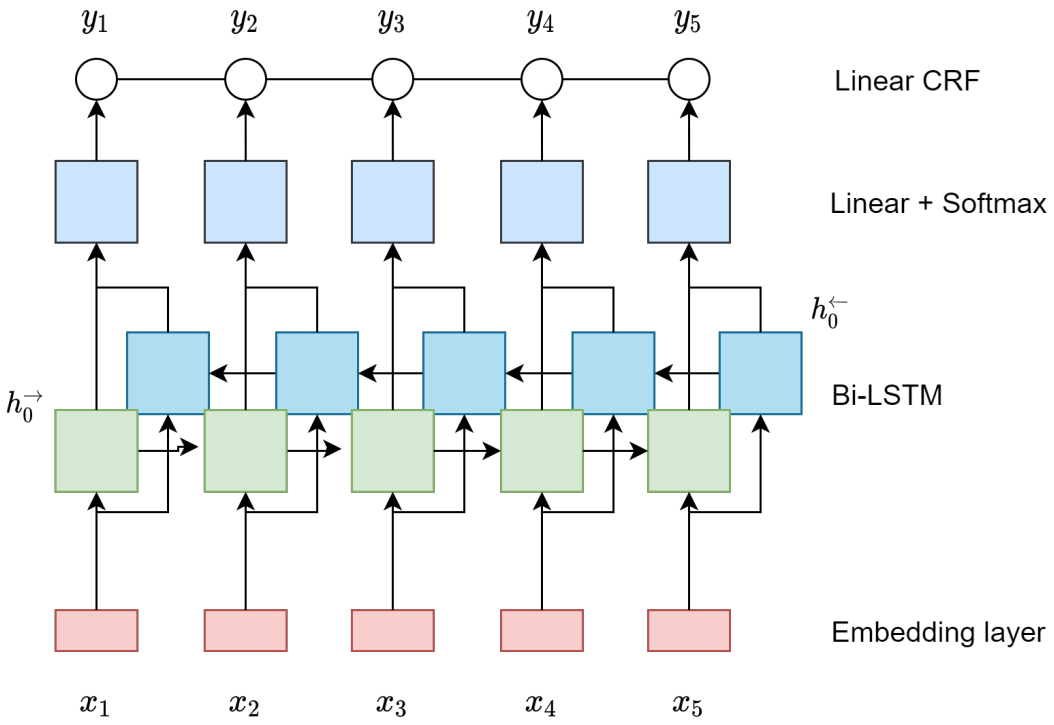


Figure 39.2.4: BiLSTM with an additional linear CRF for sequence labeling tasks.

The linear CRF layer can be parameterized by a transition matrix $A \in \mathbb{R}^{|Y| \times |Y|}$ for consecutive tags. Given a training sequence with tokens $(x_1, ..., x_T)$ and tags $(y_1, ...y_T)$, we maximize the likelihood score of this sequence, given by:

$$s\left(x_{1:T}, y_{1:T}\right) = \sum_{t=1}^{T} \left(A(y_{t-1}, y_t) + f_\theta(y_t | x_{1:T})\right),$$

where $f_\theta(y_t|x_{1:T})$ is the logit from the network output at tag y_t and the model parameters are $\theta \cup A$. Note that A can also be separately optimized to maximize the score, which is the MLE for the transition matrix.

39.2.4.3 *BERT labeling*

With wide adoption of BERT model in a broad range of NLP tasks, BERT and its variant are often the models that achieve the state-of-the-art performance in sequence labeling tasks. Typically, we only need to fine-tune a pretrained BERT model for a couple of epochs. BERT model relies on the self-attention mechanism to capture the context dependence among words. The BERT model architecture for sequence labeling has a token classification head on top (a linear layer on top of the hidden-states output) [Figure 39.2.5]. The classification head takes the token embedding as input and predict the token class label.

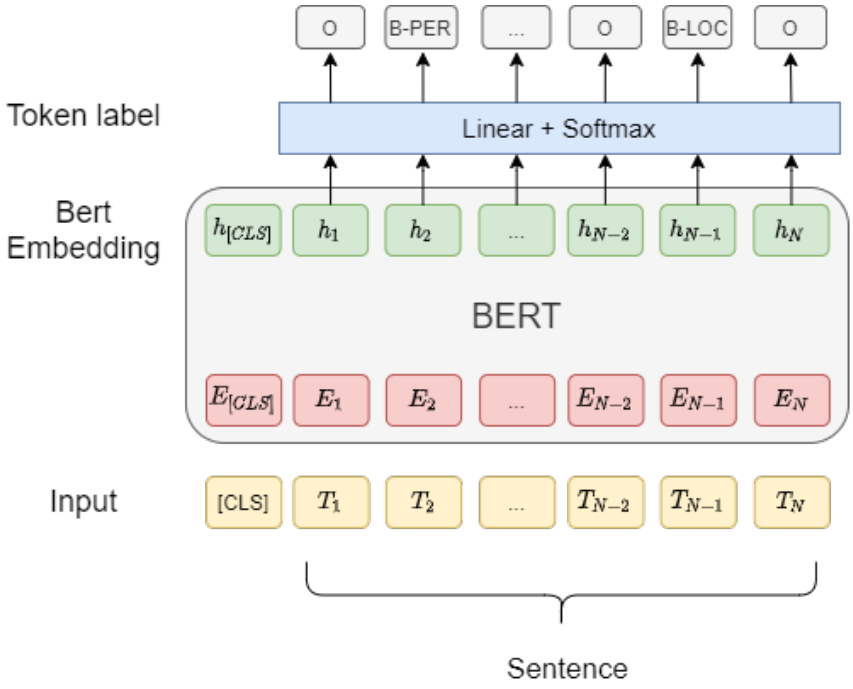


Figure 39.2.5: The BERT model architecture for sequence labeling has a token classification head on top (a linear layer on top of the hidden-states output).

39.3 Sentence embeddings and its applications

39.3.1 Introduction

There are many NLP tasks involves determining the relationship of two sentences, including semantic similarity, semantic relation reasoning, questioning answering etc. For example, Quora needs to determine if a question asked by a user has a semantically similar duplicate. The GLUE benchmark as an example [subsection 38.4.4], 6 of them are tasks that require learning sentences Inter-relationship. Specifically,

MRPC: The Microsoft Research Paraphrase Corpus [5] is a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent.

QQP: The Quora Question Pairs[1] dataset is a collection of question pairs from the community question-answering website Quora. The task is to determine whether a pair of questions are semantically equivalent. As in MRPC, the class distribution in QQP is unbalanced (63% negative), so we report both accuracy and F1 score. We use the standard test set, for which we obtained private labels from the authors. We observe that the test set has a different label distribution than the training set.

STS-B: The Semantic Textual Similarity Benchmark [6] is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data.

Natural language inference: Understanding entailment and contradiction is fundamental to understanding natural language, and inference about entailment and contradiction is a valuable testing ground for the development of semantic representations. The semantic concepts of entailment and contradiction are central to all aspects of natural language meaning, from the lexicon to the content of entire texts. Natural language inference is the task of determining whether a *hypothesis* is true (entailment), false (contradiction), or undetermined (neutral) given a *premise*. [7]

Although BERT model and its variant have achieved new state-of-the-art among many sentence-pair classification and regression tasks. It has many practical challenges in tasks like large-scale semantic similarity comparison, clustering, and information retrieval via semantic search, etc. These tasks require that both sentences are fed into the network, which causes a massive computational overhead for large BERT model. Considering the

1 https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

Premise	Label	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	Contraction	The man is sleeping.
An older and younger man smiling.	Neutral	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	Entailment	Some men are playing a sport.

Table 39.3.1: *Entailment*, *contradiction*, and *neural* examples in a natural language inference task.

task of finding the most similar pair among N sentences, then it requires $N^2/2$ forward pass computation of BERT.

An alternative approach is to derive a semantically meaningful sentence embeddings for each sentence. A sentence embedding is a dense vector representation of a sentence. Sentences with similar semantic meanings are close and sentences with different meanings are apart. With sentence embeddings, similarity search can be realized simply via a distance or similarity metrics, such as cosine similarity. Besides performing similarity comparison between two sentences, the sentence embedding can also be used as generic sentence feature for different NLP tasks.

Early on, there have been efforts directed to derive sentence embedding by aggregating word embeddings. For example, one can average the BERT token embedding output as the sentence embedding. Another common practice is to use the output of the first token (the [CLS] token) as the sentence, which is found to be worse than averaging GloVe embeddings[8]. Recently, contrastive learning and Siamese type of learning have been successfully applied in deriving sentence embeddings, which will be the focus of our following sections.

39.3.2 InferSent

InferSent is a sentence embeddings training method invented by Facebook AI[9]. Sentence encoder is trained on natural language inference data[7] and can provide semantic sentence representations generalizing well to many different tasks.

InferSent uses the Siamese type of learning scheme [Figure 39.3.1]. A sentence pair are encoded by sentence encoders into separate sentence representations. Sentence representations are then concatenated and combined before being fed into a linear Softmax classifier with three labels: entailment, contradiction, and neutral.

In the inference stage, we can compute and store sentence embedding offline for large scale semantic search tasks.

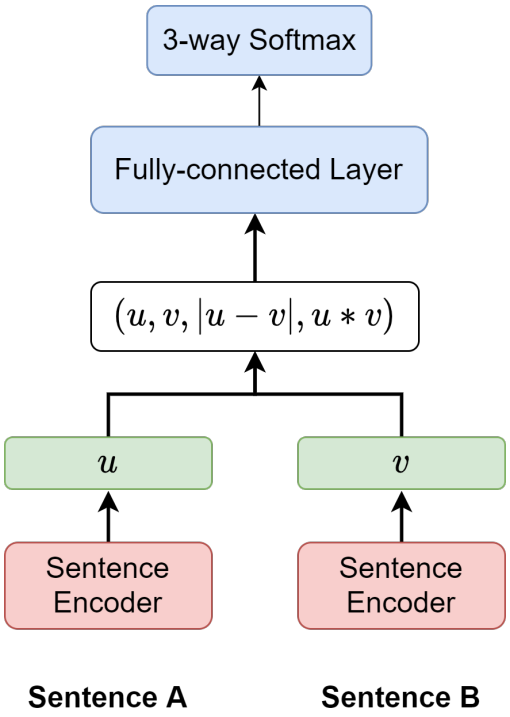


Figure 39.3.1: Siamese type of training scheme to learning sentence embedding from natural language inference task. The sentence encoder is bi-directional LSTM architecture with max pooling and it takes 300 dimension of Glove word embeddings of constituents as the input.

InferSent represents a new paradigm of solving sentence-pair related tasks: each sentence has an representation derived separately, and the representation is used to solve downstream tasks. This contrasts BERT approach, which is a joint method that uses cross-features or attention from one sentence to the other.

39.3.3 Sentence-BERT

While BERT has demonstrated remarkable performance across NLP tasks, a large disadvantage of the BERT approach is that no independent sentence embeddings are computed, which makes it difficult to derive sentence embeddings from BERT.

Sentence-BERT enables sentence embedding extraction via Siamese type of learning like InferSent. The major difference between the InferSent and Sentence-BERT is that Sentence-BERT utilizes BERT to encode the sentence. Specifically, three strategies can be used to derived sentence embedding

- the output of the CLS-token
- the mean of all output vectors
- max-over-time of the output vectors

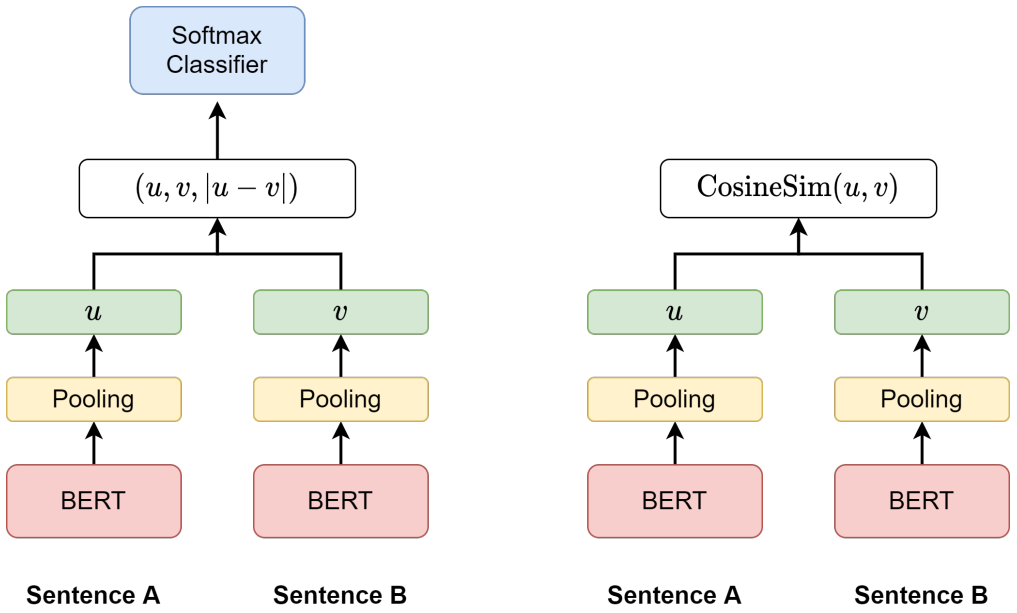


Figure 39.3.2: (**left**) Sentence-BERT architecture with classification objective function for training. The two BERT networks are pre-trained have tied weights (Siamese network structure). (**right**) Sentence-BERT architecture at inference. The similarity between two input sentences can be computed as the similarity score between two sentence embeddings.

39.4 Sequence-to-sequence modeling

39.4.1 Encoder decoder model

In Sequence-to-sequence (seq2seq) modeling, we are seeking a mapping that takes a sequence $X_1, X_2, ..., X_N$ as the input and outputs another sequence $Y_1, Y_2, ..., Y_M$ that meet certain requirements. In general, X_i and Y_j can be a character, a words, or a real-valued vector, an image, etc, and generally $N \neq M$.

Many fascinating AI real-world applications fall into the seq2seq category. Examples include

- machine translation, where a word sequence in one language is mapped to a word sequence in another language;
- speech recognization, where a time series of audio signal is mapped to a word sequence;
- video captioning, where a video, or a sequence of images, is mapped to a word sequence;
- text summarization, where a long sequence of words is mapped to a short sequence of words (i.e., a summary).

One challenge in a seq2seq modeling problem is that usually it cannot be decomposed to a smaller-scale mapping problem between one sequence unit to another sequence unit. For example, in translation task, one word in one language could be translated to one word, or multiple words, or even not translated depending on the context. Further, usually no synchrony exist between sequence X and sequence Y. Take machine translation for another example. Suppose in a word sequence X_i precedes word X_j, and in the translated word sequence, they also have clear correspondence to Y_k and Y_l, respectively. By no synchrony, we mean Y_k might come after Y_l due to language grammar reasons. Therefore, in the seq2seq modeling, sequences must be processed as a whole unit. Note that such architecture can include multiple hidden layers as well. A example of synchrony issue in machine translation is showed in Figure 39.4.1.

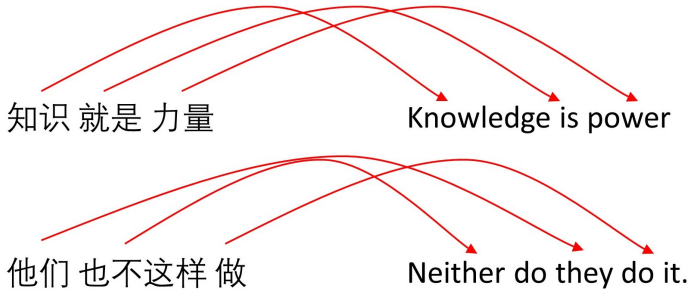


Figure 39.4.1: Seq2seq modeling for language transformation. The input and output sequences might have different lengths, and not in synchrony.

The most basic implementation of a sequence-to-sequence model is via a encoder-decoder network consisting of two RNNs, called encoder and decoder, respectively [Figure 39.4.2]. The encoder-decoder framework stems from efforts in natural language model[10]. The encoder reads a sequence of inputs and outputs the hidden vector at the end of the sequence, known as **the context vector**; the decoder takes the context vector as the initial hidden state input and then sequentially generates a sequence. After training, the context vector is expected to store all relevant, critical information of the input sequence such that the decoder can produce sequences meet certain requirements.

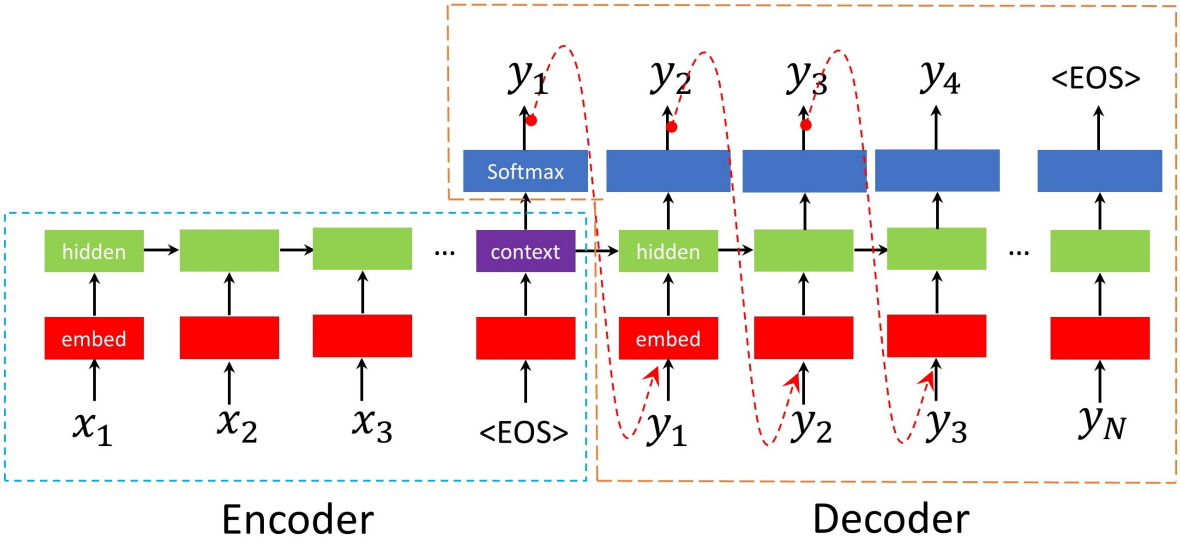


Figure 39.4.2: The Encoder-decoder architecture for seq2seq language modeling. The input sequence is fed into the encoder RNN and terminated by an explicit <EOS> symbol. Then the decoder RNN starts with the context vector and the final prediction of the encoder to generate an output sequence until an explicit <EOS> symbol is produced.

In a video captioning application, this encoder-decoder model will take image sequence as the input, with the usage of convolutional layers replacing embedding layers to extract image feature, and will produce a sequence of words to generate words as captions.

39.4.1.1 *work overflow*

The above image shows an example translation. The input/source sentence, "guten morgen", is passed through the embedding layer (yellow) and then input into the encoder (green). We also append a start of sequence ($\langle \text{sos} \rangle$) and end of sequence eos token to the start and end of sentence, respectively. At each time-step, the input to the encoder RNN is both the embedding, e, of the current word, $e(x_t)$, as well as the hidden state from the previous time-step, h_{t-1}, and the encoder RNN outputs a new hidden state h_t. We can think of the hidden state as a vector representation of the sentence so far. The RNN can be represented as a function of both of $e(x_t)$ and h_{t-1}

$$h_t = \text{EncoderRNN}\left(e(x_t), h_{t-1}\right)$$

We're using the term RNN generally here, it could be any recurrent architecture, such as an LSTM (Long Short-Term Memory) or a GRU (Gated Recurrent Unit).

Here, we have $X = \{x_1, x_2, \ldots, x_T\}$, where $x_1 = \langle \text{sos} >, x_2 = \text{guten}$, etc. The initial hidden state, h_0, is usually either initialized to zeros or a learned parameter. Once the final word, x_T, has been passed into the RNN via the embedding layer, we use the final hidden state, h_T, as the context vector, i.e. $h_T = z$. This is a vector representation of the entire source sentence. Now we have our context vector, z, we can start decoding it to get the output/target sentence, "good morning". Again, we append start and end of sequence tokens to the target sentence. At each time-step, the input to the decoder RNN (blue) is the embedding, d, of current word, $d\left(y_t\right)$, as well as the hidden state from the previous time-step, s_{t-1}, where the initial decoder hidden state, s_0, is the context vector, $s_0 = z = h_T$, i.e. the initial decoder hidden state is the final encoder hidden state. Thus, similar to the encoder, we can represent the decoder as:

$$s_t = \text{ DecoderRNN } \left(d\left(y_t\right), s_{t-1}\right)$$

Although the input/source embedding layer, e, and the output/target embedding layer, d, are both shown in yellow in the diagram they are two different embedding layers with their own parameters. In the decoder, we need to go from the hidden state to an actual word, therefore at each time-step we use s_t to predict (by passing it through a Linear layer, shown in purple) what we think is the next word in the sequence, $\hat{y}_t$

$$\hat{y}_t = f\left(s_t\right)$$

The words in the decoder are always generated one after another, with one per time-step. We always use $\langle$ sos $\rangle$ for the first input to the decoder, y_1, but for subsequent inputs, $y_{t>1}$, we will sometimes use the actual, ground truth next word in the sequence, y_t and sometimes use the word predicted by our decoder, $\hat{y}_{t-1}$. This is called teacher forcing, see a bit more info about it here. When training/testing our model, we always know how many words are in our target sentence, so we stop generating words once we hit that many. During inference it is common to keep generating words until the model outputs an token or after a certain amount of words have been generated.

Once we have our predicted target sentence, $\hat{Y} =\mid \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T \Big\}$, we compare it against our actual target sentence, $Y = \left\{y_1, y_2, \ldots, y_T\right\}$, to calculate our loss. We then use this loss to update all of the parameters in our model.

In the paper we are implementing, they find it beneficial to reverse the order of the input which they believe "introduces many short term dependencies in the data that make the optimization problem much easier". We copy this by reversing the German sentence after it has been transformed into a list of tokens.

39.4.2 Attention mechanism

The performance of a basic encoder-decoder model [Figure 39.4.2] relies on the **context vector** to store all relevant, critical information of the input sequence. It is not hard to notice that this encoder-decoder network suffers when the input sequences are very long: difficulties arise in both in the encoding part - difficult to encode long sequences in a single vector without losing information, and in the decoding part - difficult to decode a single information vector to a long sequence.

Attention mechanism is inspired how human beings focus on relevant information when performing visual and language tasks[11, 12]. Consider a human translates a sentence. When he translates each word in the sentence, he tends to focus on its related context rather than focusing on the entire sentence. Similarly, when a human is watching closely some part of a object, he tends to focus on the part of interest.

In the context of encoder-decoder, all hidden states carry information and some information would get diluted in the subsequent context vector, especially when the sequence is long. A encoder-decoder architecture with attention mechanism have following features to resolve the difficulties of encoding and decoding [Figure 39.4.3]:

- During the encoding phase, all hidden states, rather than the final one, are saved to construct different context vectors via linear combination for the decoding stage.
- During the decoding phase, relevant context vectors are constructed and fed into the each hidden states in the decoder, alleviating the difficulty to decode all information from one starting context vector.

In summary, the encoder encodes a library of hidden vectors as the building blocks of different context vectors for each output in the decoding stage. The attention mechanism resembles a information lookup system that constructs and feeds the most important and relevant context vector to help the decoder part to do their job. From information flow point of view, the primary purpose attention is to calculate a weight vector to amplify signals that are most relevant in the input sequence and to de-emphasize the part that is irrelevant. The higher weights will enable more relevant hidden state information fed into the the decoder hidden layer.

Now we consider a more concrete implementation of attention mechanism from [11]. Let $(x_1, x_2, ..., x_m)$ and $(y_1, y_2, ..., y_n)$ denote the input and output sequences. The encoder uses bidirectional RNN to encode input sequences to a library of hidden states. Particularly, let $\overrightarrow{h}_i$ denote the hidden state in the forward pass and $\overleftarrow{h}_i$ the hidden state in the backward pass. A hidden state is the concatenation $h_i = cat(\overrightarrow{h}_i, \overleftarrow{h}_i)$. In the decoder part, let s_t be the hidden state information that just precedes the output y_t. s_t is given by

$$s_t = f(s_{t-1}, y_{t-1}, c_i),$$

where c_i is the context vector constructed as a weighted combination of hidden information $\sum_{i=1}^{M} w_i(t)h_i$. Note that the weights is index-dependent, derived from

$$e_i(t) = g(h_i, s_{t-1})$$
$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}.$$

The primary purpose of g is to capture the relevance or correlation between the decoder hidden state and the encoder hidden state. Typical choices for g functions include

$$g\left(h_i, s_{t-1}\right) = h_i^T s_{t-1}$$
$$g\left(h_i, s_{t-1}\right) = h_i^T W_g s_{t-1}$$
$$g\left(h_i, s_{t-1}\right) = v_g^T \tanh\left(W_g[h_i; s_{t-1}]\right)$$

where W_g and v_g are learnable parameters.

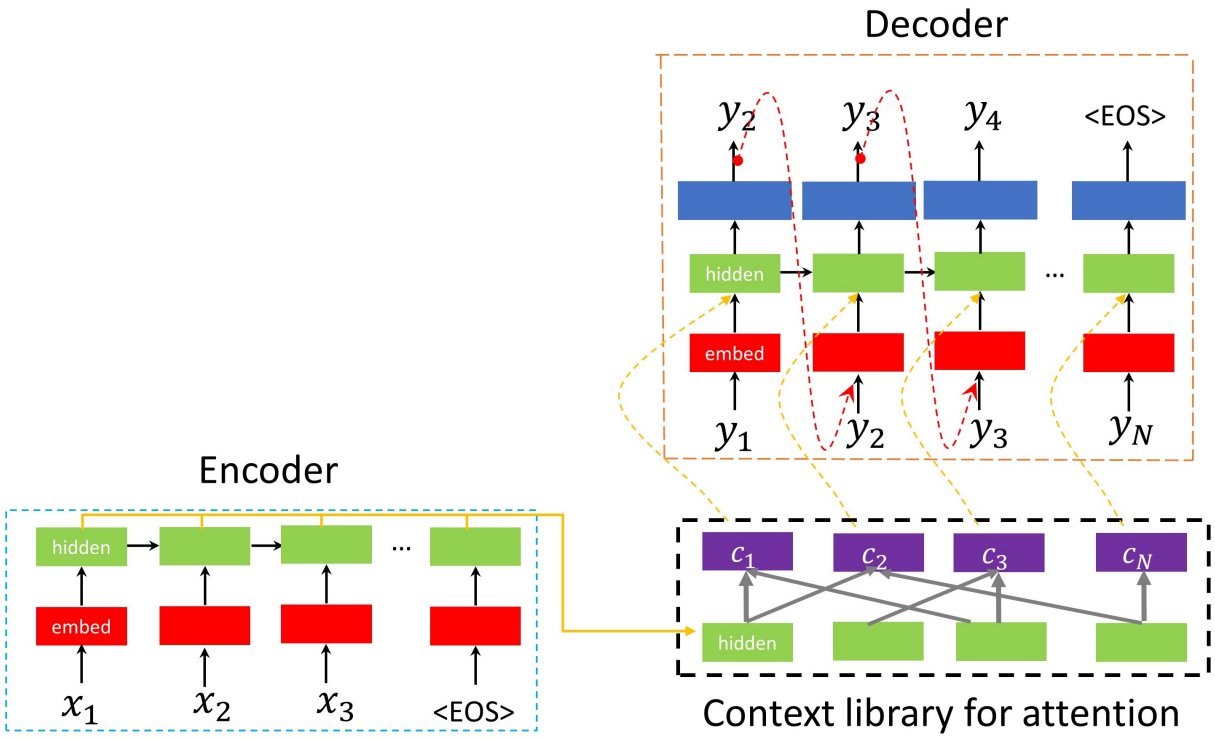


Figure 39.4.3: The Encoder-decoder architecture with attention mechanism for seq2seq modeling. During the encoding phase, all hidden states, rather than the final one, are saved to construct different context vectors via linear combination for the decoding stage. During the decoding phase, relevant context vectors are constructed and fed into the each hidden states in the decoder.

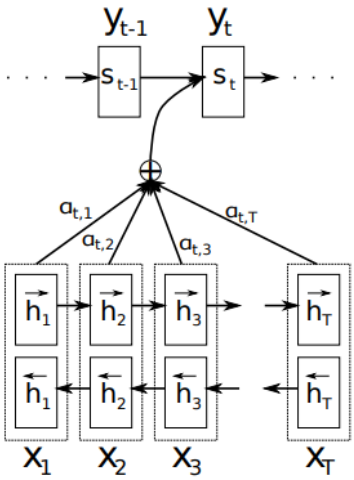


Figure 39.4.4: A bidirectional RNN encoder system with attention mechanism. [11]

39.4.3 Google's Neural Machine Translation System

A production level application of the attention-based seq2seq framework is the Google's Neural Machine Translation System [13].

Our model consists of a deep LSTM network with 8 encoder and 8 decoder layers using residual connections as well as attention connections from the decoder network to the encoder. Simply stacking more LSTM layers will lead to performance degradation, probably due to exploding and vanishing gradient in the vertical direction. Residual connection can greatly improve the gradient flow in the backpropogation process. As translation usually requires information across the whole source side, bi-directional connection is employed. However, to enable maximum possible parallelization, bi-directional connections are only used in the first layer in the encoder.

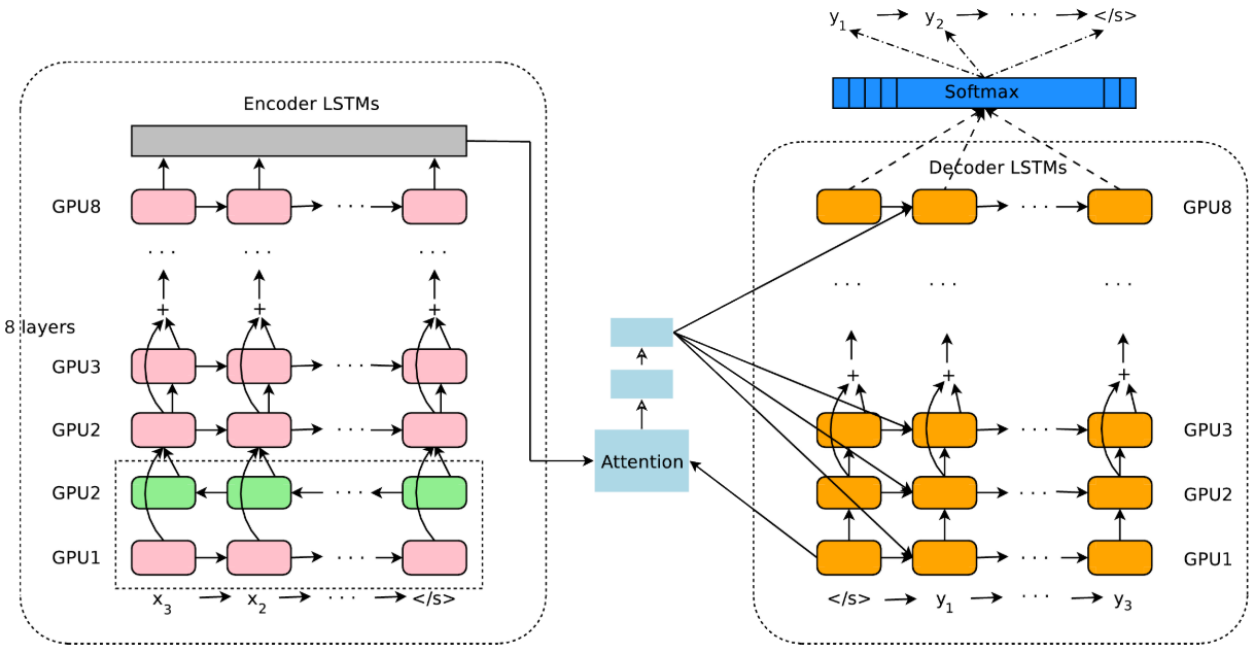
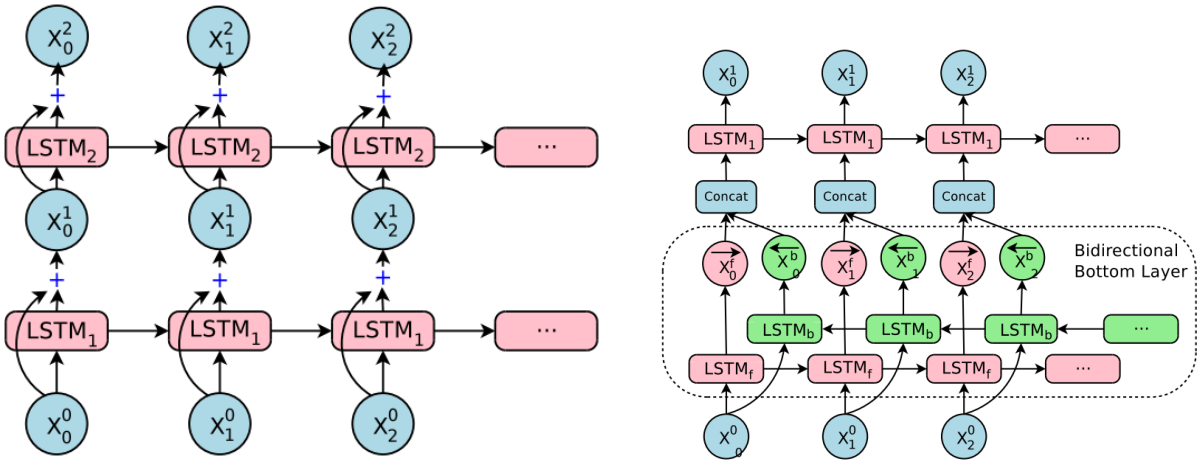


Figure 39.4.5: The model architecture of Google's Neural Machine Translation system, which consists of an encoder module (left), an attention module (middle), a decoder module (right). The bottom encoder layer is bi-directional and other layers are uni-directional. Residual connections are used from the third layer in the encoder and in all layers in the decoder. There are total 8 LSTM layers in both encoder and decoder and the model is partitioned into multiple GPUs to speed up training. Image from [13].



(a) stacked LSTM layers with residual connections.

(b) The structure of bi-directional connections in the first layer of the encoder.

Figure 39.4.6: The residual connection and bi-directional LSTM structure in the model.Image from [13].

Now we look into the mathematical detail. Let (X, Y) be a source and target sentence pair. Let $X = (x_1, x_2, x_3, \ldots, x_M)$ be the sequence of M tokens in the source sentence and let $Y = y_1, y_2, y_3, \ldots, y_N$ be the sequence of N tokens in the target sentence. The encoder module transforms the M input tokens into M hidden states $(\boldsymbol{x_1}, ..., \boldsymbol{x_M})$:

$$x_1, x_2, \ldots, x_M = \text{Encoder}\left(x_1, x_2, x_3, \ldots, x_M\right).$$

With the residual connection, the update of hidden states $\boldsymbol{h}$ and cell state $\boldsymbol{c}$ have the following update procedures throughout the stacked LSTM encoder (from layer i to layer $i+1$).

$$\begin{aligned} c_t^i, h_t^i &= \text{LSTM}_i(c_{t-1}^i, h_{t-1}^i, x_t^{i-1}) \\ x_t^i &= h_t^i + x_t^{i-1} \\ c_t^{i+1}, h_t^{i+1} &= \text{LSTM}_{i+1}(c_{t-1}^{i+1}, h_{t-1}^{i+1}, x_t^i) \end{aligned}$$

The computation of the context vector $\boldsymbol{a}_i$ for ith step in the decoder is given by the following step

$$\begin{aligned} s_t &= \text{Attention}\left(y_{i-1}, x_t\right) \quad \forall t, \quad 1 \leq t \leq M \\ w_t &= \frac{\exp(s_t)}{\sum_{t=1}^{M} \exp(s_t)} \quad \forall t, \quad 1 \leq t \leq M \\ a_i &= \sum_{t=1}^{M} w_t \cdot \mathbf{x}_t \end{aligned}$$

where the Attention function is implemented as a feed-forward neural network with one hidden layer.

As the softmax layer in the decoder output the token probability based on previous inputs, the whole model approximates the conditional probability in the following way

$$\begin{aligned} P(Y|X) &= P\left(Y|x_1, x_2, x_3, \ldots, x_{\mathrm{M}}\right) \\ &= \prod_{i=1}^{N} P\left(y_i|y_0, y_1, y_2, \ldots, y_{i-1}; x_1, x_2, x_3, \ldots, x_M\right) \end{aligned}$$

The optimization on model parameter θ is achieved by performing gradient descent to maximizes a mixed likelihood, one is ordinary likelihood and one is reward-weighted likelihood

$$L_{\text{Mixed}}\left(\theta\right) = \alpha L_{\text{ML}}(\theta) + L_{\text{RL}}(\theta)$$

where α is a scalar, Y^* is the ground truth output sequence and

$$L_{\text{ML}}(\theta) = \sum_{i=1}^{N} \log P_\theta\left(Y^{*(i)}|X^{(i)}\right)$$

and

$$L_{\text{RL}}(\theta) = \sum_{i=1}^{N} \sum_{Y \in \mathcal{Y}} P_\theta\left(Y|X^{(i)}\right) r\left(Y, Y^{*(i)}\right).$$

The reward is related to evaluation score in the translation task.

39.5 Notes on Bibliography

39.5.1 Books and references

General books on Natural language processing include [14][15][16]. Practical books include [14][16].

For a comprehensive review on deep learning methods for text classification, see [17].

39.5.2 Software

NLTK and **spaCy** are two popular packages widely used for pre-processing tasks such as tokenization, stemming,lemmatization, stopwords removal, parts of speech tagging.

BIBLIOGRAPHY

1. Porter, M. F. An algorithm for suffix stripping. *Program* (1980).
2. Joulin, A., Grave, E., Bojanowski, P. & Mikolov, T. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).
3. Kim, Y. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).
4. Huang, Z., Xu, W. & Yu, K. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991* (2015).
5. Dolan, W. B. & Brockett, C. *Automatically constructing a corpus of sentential paraphrases* in *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)* (2005).
6. Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I. & Specia, L. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055* (2017).
7. Bowman, S. R., Angeli, G., Potts, C. & Manning, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326* (2015).
8. Reimers, N. & Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
9. Conneau, A., Kiela, D., Schwenk, H., Barrault, L. & Bordes, A. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364* (2017).
10. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to sequence learning with neural networks* in *Advances in neural information processing systems* (2014), 3104–3112.
11. Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
12. Luong, M.-T., Pham, H. & Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).
13. Wu, Y. *et al.* Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).
14. Manning, C. D. & Schütze, H. *Foundations of statistical natural language processing* (MIT press, 1999).
15. Jurafsky, D., Martin, J., Norvig, P. & Russell, S. *Speech and Language Processing* ISBN: 9780133252934 (Pearson Education, 2014).

16. Goldberg, Y. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies* **10,** 1–309 (2017).

17. Minaee, S. *et al.* Deep Learning Based Text Classification: A Comprehensive Review. *arXiv preprint arXiv:2004.03705* (2020).

40

DEEP LEARNING FOR AUTOMATIC SPEECH RECOGNITION

40.1 Speech signal characterization

40.1.1 Speech signal representation

A speech signal is a sequence of observations of the amplitude of the speech spoken by the speaker. Physically, when an audio signal is recorded, the signal values represent variations in the air pressure around a central average value (as a result of the physics of sound generation). Typically, observations are made at a sampling rate as high as 16kHz, i.e., 16,000 samples per second.

In terms of speech or speaker recognition, it is widely believed that the most useful information is contained in the frequency domain of the signals. For example, pronunciation of different vowels will produce formants (position of peaks in amplitude) at different frequency bands.

As we can see Figure 40.1.1, audio signals are normally *non-stationary*, which prevents us from directly performing Fourier transform on the whole sequence. Instead, we usually first decompose a recorded audio signal (typical 3 s to 10 s), which is also referred to as an utterance, into shorter signal sequences,of 20 to 30 milliseconds, which usually consists of about 320 to 480 observations (assuming 16kHz sampling rate). Then we assume these short sequence signals are stationary and then we perform **short-term Fourier transform (STFT)** on these short sequences. This stationarity assumption is justified by the fact that our vocal track physically remains still within such short time frame. Each short sequence is called a **frame**, and the process of dividing an utterance into multiple short sequences is called **framing**.

When you use the FFT to measure the frequency component of a signal, you are basing the analysis on a finite set of data. The actual FFT transform assumes that it is a finite data set, a continuous spectrum that is one period of a periodic signal. For the FFT, both the time domain and the frequency domain are circular topologies, so the two endpoints of the time waveform are interpreted as though they were connected together. When the measured signal is periodic and an integer number of periods fill the acquisition time interval, the FFT turns out fine as it matches this assumption. However, many times, the measured signal is not an integer number of periods. Therefore, the finiteness of the measured signal may result in a truncated waveform with different characteristics from the original continuous-time signal, and the finiteness can introduce sharp transition changes into the measured signal. The sharp transitions are discontinuities.

When the number of periods in the acquisition is not an integer, the endpoints are discontinuous. These artificial discontinuities show up in the FFT as high-frequency components not present in the original signal. These frequencies can be much higher than the Nyquist frequency and are aliased between 0 and half of your sampling rate. The

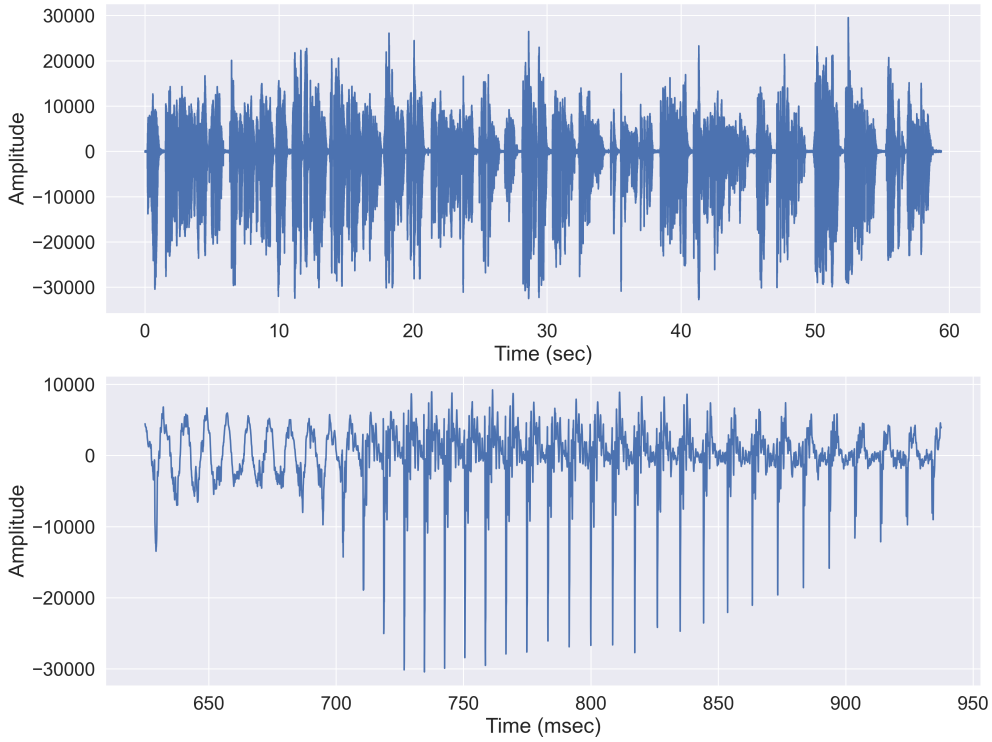


Figure 40.1.1: (a) 100 second audio signal (sampling rate 16kHz), with a sampling rate 16kHz and total samples of 950353. (b) The magnified view of the audio signal in (a) within 1 s.

spectrum you get by using a FFT, therefore, is not the actual spectrum of the original signal, but a smeared version. It appears as if energy at one frequency leaks into other frequencies. This phenomenon is known as spectral leakage, which causes the fine spectral lines to spread into wider signals.

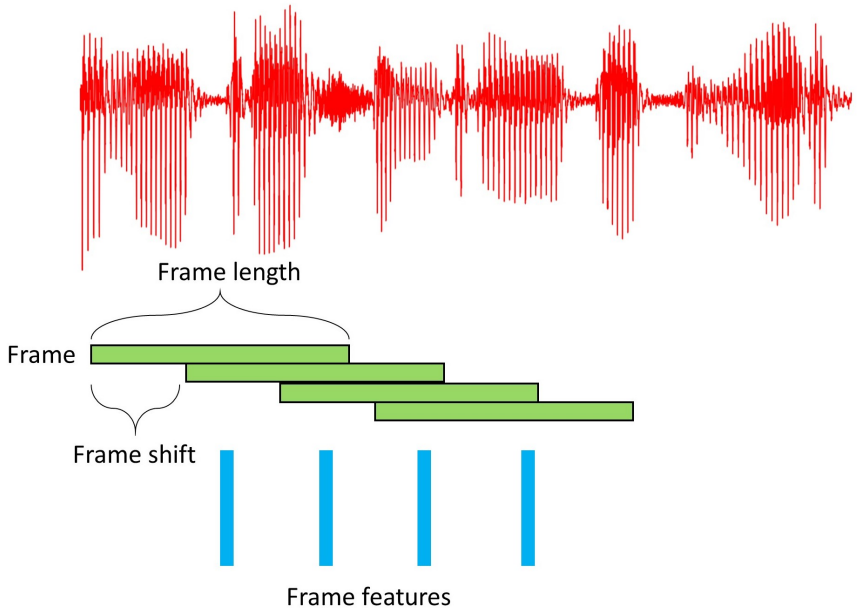


Figure 40.1.2: Speech audio signals are decomposed into overlapping frames, which are further processed into features for machine learning applications.

The purpose the window function is to minimize the end-point discontinuities in the frame. Commonly used window functions include [1, p. 89]

- Hamming window function, which has the form

$$h[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$$

  where $n = 0, ..., N-1$, and N is the window length.
- Hanning window function, which has the form

$$h[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right)$$

  where $n = 0, ..., N-1$, and N is the window length.
- Triangular window function, which has the form

$$h[n] = \begin{cases} \frac{n}{N/2}, & \text{for } n = 0, 1, 2, \ldots, N/2 \\ 2 - \frac{n}{N/2}, & \text{for } n = N/2+1, N/2+2, \ldots, N-1 \end{cases}$$

where $n = 0, ..., N-1$, and N is the window length.

Figure 40.1.3 demonstrates the application of the hamming window function to a frame of audio signal. Due to windowing, we are actually losing the samples at the beginning and the end of the frame; this too will lead to an incorrect frequency representation. To compensate for this loss, we can take overlapping frames rather than disjoint frames. For example, the frame length is usually 25ms and the frame shift step size is usually 10ms. Therefore there is an overlap between frames.

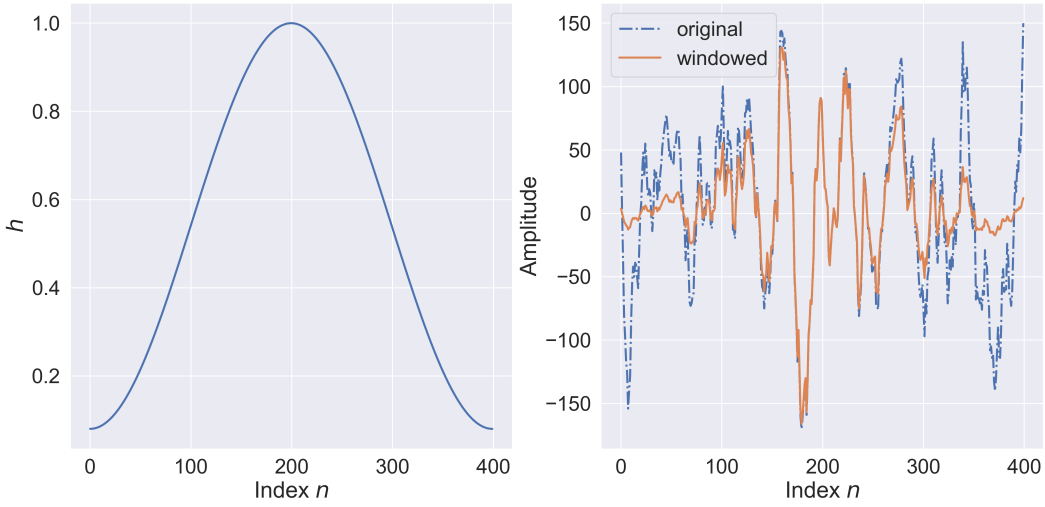


Figure 40.1.3: (a) Hamming window function. (b) A signal frame before and after hamming windowing.

40.1.2 Measurement of signal

40.1.2.1 *Amptitude and power*

Because a signal is time series, usually we refer to the amplitude of a signal as the root mean square (rms) amplitude of the signal. More formally, let $S_1, ..., S_T$ be the signal observation values. The rms amplitude is given by

$$A_{rms} = \sqrt{\frac{1}{T}\sum_{i=1}^{T} S_i^2}.$$

Similarly, we usually refer to the power of a signal of the root mean square (rms) power of the signal, which is given by

$$P_{rms} = \frac{1}{T}\sum_{i=1}^{T} S_i^2.$$

40.1.2.2 *Decibel*

The decibel (symbol: dB) is a relative unit of measurement corresponding to one tenth of a bel (B). It is used to express the ratio of one value of a power or root-power quantity to another, on a logarithmic scale.

Sound is measured in units called decibels (dB). The higher the decibel level, the louder the noise. On the decibel scale, the level increase of 10 means that a sound is actually 10 times more intense, or powerful.

Two principal types of scaling of the decibel are in common use. When expressing a power ratio, it is defined as ten times the logarithm in base 10.[5] That is, a change in power by a factor of 10 corresponds to a 10 dB change in level. When expressing root-power quantities, a change in amplitude by a factor of 10 corresponds to a 20 dB change in level. The decibel scales differ by a factor of two, so that the related power and root-power levels change by the same value in linear systems, where power is proportional to the square of amplitude.

Given a signal, the conversion from amplitude to Decibel is given by

$$A_{dB} = 20\log_{10}(\frac{A}{A_ref}),$$

where A_{ref} is the reference level (e.g., $A_{ref} = 1$). Similarly, the conversion from power to Decibel is given by

$$P_{dB} = 10\log_{10}(\frac{P}{P_ref}),$$

where P_{ref} is the reference level (e.g., $P_{ref} = 1$).

40.1.2.3 *Signal to noise ratio*

Signal-to-noise ratio (SNR) is defined as the ratio of the power of a signal to the power of background noise:

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}}$$

where P is average power.

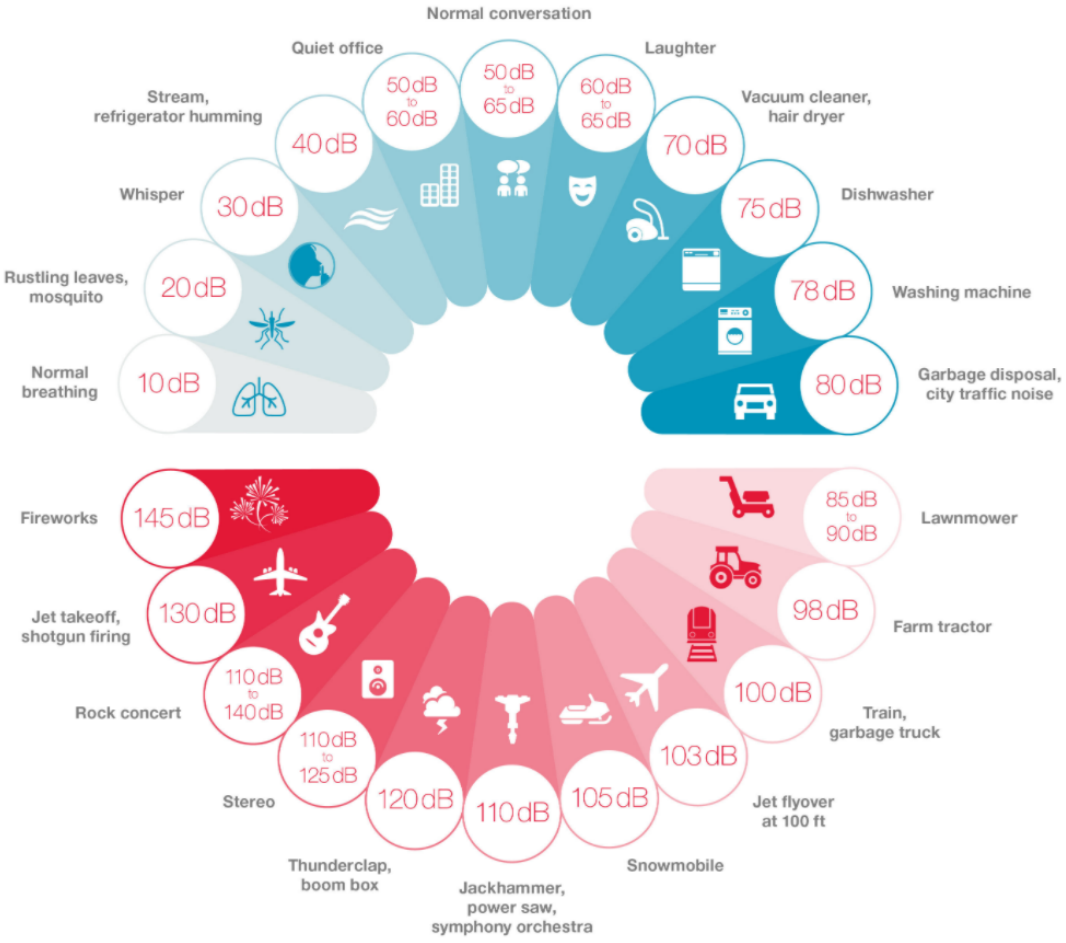


Figure 40.1.4: Image from source.

If the signal is represented by an iid sequence of random variable S and the noise is represented by an iid sequence of random variable N, then we can write SNR by , i.e. in this case the mean square of N, or

$$\text{SNR} = \frac{\text{E}\left[S^2\right]}{\text{E}\left[N^2\right]}$$

where E refers to the expected value.

Because many signals have a very wide dynamic range, signals are often expressed using the logarithmic decibel scale. Based upon the definition of decibel, signal and noise may be expressed in decibels (dB) as

$$P_{\text{signal ,dB}} = 10\log_{10}\left(P_{\text{signal}}\right)$$

and

$$P_{\text{noise ,dB}} = 10\log_{10}\left(P_{\text{noise}}\right)$$

In a similar manner, SNR may be expressed in decibels as

$$\text{SNR}_{\text{dB}} = 10\log_{10}(\text{SNR})$$

Using the definition of SNR

$$\text{SNR}_{\text{dB}} = 10\log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right)$$

Using the quotient rule for logarithms

$$10\log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right) = 10\log_{10}\left(P_{\text{signal}}\right) - 10\log_{10}\left(P_{\text{noise}}\right)$$

40.1.3 Signal transformation and representations

40.1.3.1 *Discrete Fourier Transform (DFT)*

Discrete Fourier Transform (DTF) aims to reveal frequency-domain characteristics of audio signals. Denote a time-domain signal (an utterance) by S and its constituent frames by $s_1, ..., s_M$, where M is the total number of frames. We also denote each observation at frame i by $s_i(n), n = 0, ..., N-1$ ($N = 400$ if assuming 16kHz sampling rate and 0.025 s frame length, then each frame has 400 samples).

The DFT on windowed s_i is given by [Figure 40.1.5]

$$S_i(k) = \sum_{n=0}^{N-1} s_i(n)h(n)e^{-j2\pi kn/N} \quad 0 \leq k \leq K$$

where j is the imaginary number, $h(n)$ is an N sample long analysis window (e.g. hamming window), and $K = N/2$ is the length of the DFT. Here we note the following points.

- $S_i(k)$ is a complex number, which can also be written as magnitude and phase. For speech applications, it is usually the magnitude that is the most interesting and important.
- We usually zero-pad a frame, typically containing $N = 400$ data points, to have $N_{FFT} = 512$ sample points such that we can perform computational efficient FFT (Fast Fourier Transform).
- We could in principle compute $S_i(k)$ for any k from $-\infty$ to $+\infty$, but with only N_{FFT} data points, only $N_{FFT} + 1$ final outputs $k = -\frac{N_{FFT}}{2}, ..., \frac{N_{FFT}}{2}$ in the frequency domain are significant (due to Nyquist frequency). Because s_i is a real-valued signal sequence, we have $S_i(k) = \text{Conj}(S_i(-k))$. Therefore, we normally only keep frequency for $k = 0, ..., N_{FFT}/2$ ($N_{FFT}/2$ in total).

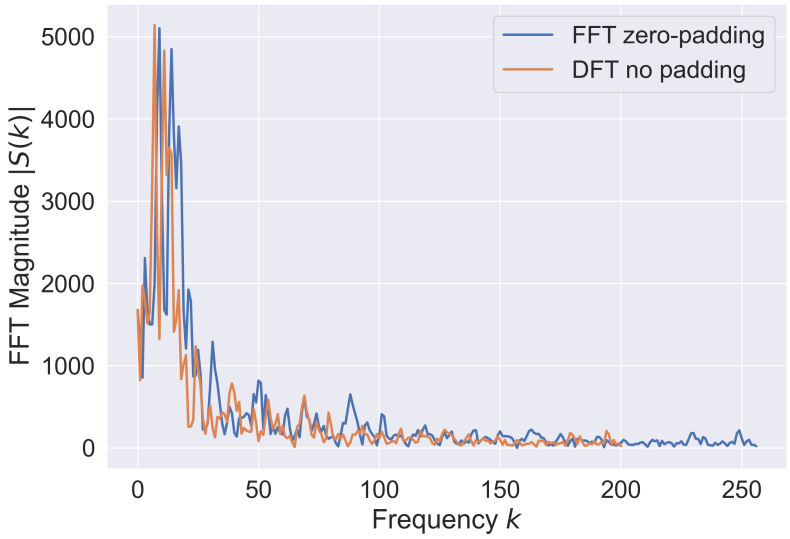


Figure 40.1.5: Demonstration of DFT/FFT on the windowed frame in Figure 40.1.3. This example frame has 400 data points. For FFT, we zero padded the frame to have 512 sample points. The frequency k runs to 200 and 256 in DFT and FFT, respectively.

Further note that since there are only a finite number of data points, the DFT treats the data as if it were infinitely long and periodic. Therefore, we can interpret k in the DFT equation as the fundamental frequency and its multipliers. The frequency $k = 0$ is the DC component of the signal, i.e., the average. The frequency $\frac{1}{N}$ is the fundamental frequency, and all frequencies we have include

$$k = 0, \frac{1}{N}, \frac{2}{N}, \dots \frac{N/2}{N}.$$

The corresponding **physical frequencies** are

$$0, \frac{SR}{N}, \frac{2SR}{N}, \dots \frac{SR}{2},$$

where SR is the sample rate. It is clear that increasing N will not increase the maximum frequency that can be probed, which is limited by the Nyquist frequency $SR/2$. Nevertheless, increasing N can allow higher resolution in the frequency range $[0, SR/2]$.

Given an utterance consisting of hundreds of frames, we can compute the FFT magnitude and stack them together to get the **periodogram-based FFT magnitude** [Figure 40.1.6]. Another import quantity is speech analysis is the **power spectrum** $P_i(k)$ of each frame. The **periodogram-based power spectral** estimate for the speech frame $s_i(n)$ is given by:

$$P_i(k) = \frac{1}{N} \left| S_i(k) \right|^2.$$

40.1.3.2 *Mel Filter bank features or Mel-spectrogram*

Mel Filter bank energies or Mel-spectrogram is one feature widely used in automatic speech recognition. Filter bank energies is obtained by applying triangular filters, typically 40 filters on a Mel-scale to the *power* spectrum to extract frequency bands. The resulting energy units are often expressed in Decibel, which is roughly equivalent to taking a log and multiplying a constant. The Mel scale connects perceived frequency, or pitch, of a signal to its actual measured physical frequency. Humans are much better at distinguishing small changes in pitch at low frequencies than they are at high frequencies. [1] Incorporating this scale makes our features match more closely what humans hear. The formula for converting from frequency to Mel scale is [Figure 40.1.7]

$$M(f) = 1125 \ln(1 + f/700);$$

in reverse, to go from Mels back to frequency we have

$$M^{-1}(m) = 700(\exp(m/1125) - 1).$$

1 In **??**, we will see that high pitch can also be useful speaker features.

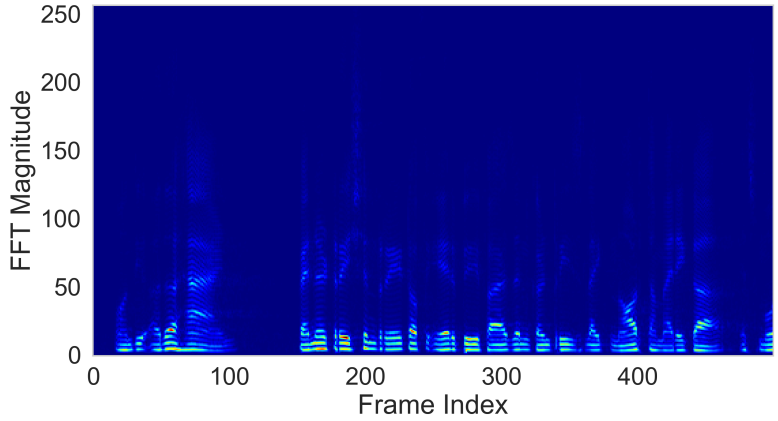


Figure 40.1.6: FFT magnitudes corresponding to 256 frequencies for each frame (horizontal axis, 500 frames in total) in the signal.

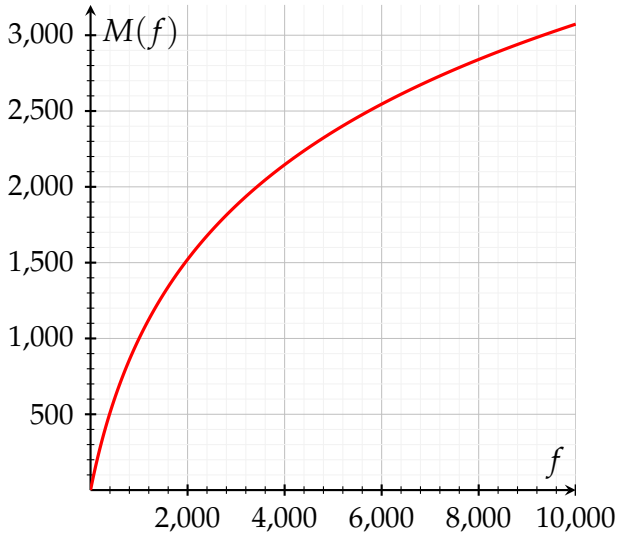


Figure 40.1.7: Mel scale transformation for physically measure frequency.

To construct filter bank energies, the first step is to construct filter banks on the Mel scale. Typically, we are interested in frequencies between 300Hz and 8000Hz (the Nyquist frequency of 16kHz), which corresponds to Mel frequency of 401.25Mels and 2834.99Mels. Suppose we like to construct 10 filters in the Mel scale, we need 10 intervals in the Mel

scale or we need cutting points (12 points including starting and ending points) in Mel scale given by

$$m(i) = 401, 623, 844, 1065, 1286, 1508, 1729, 1950, 2171, 2392, 2614, 2835.$$

These Mel scale cutting points have Hertz scale given by

$$h(i) = 300, 517, 782, 1103, 1496, 1973, 2554, 3261, 4123, 5171, 6447, 8000.$$

Note that intervals will be further converted to bin number in FFT spectrum via

$$f(i) = \left((N)^* h(i) / \text{ sample rate } \right).$$

Note that when $h(i) = 0.5$sample rate, we have $f = \frac{N}{2}$, which is the last frequency bin in FFT spectrum.

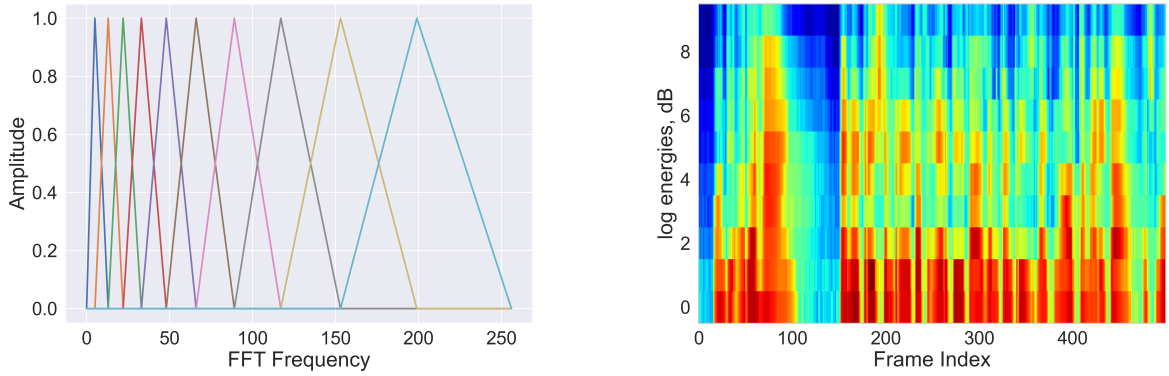
The filters have triangular shape and are given by [Figure 41.1.4]

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

Finally, we convert energy to dB unit, or log form, by

$$Y_{dB} = 10 \log_1 0(Y).$$

The complete implemented can be found in Librosa.



(a) Mel filter banks.

(b) Mel spectrogram.

Figure 40.1.8: Computation of filter bank features.

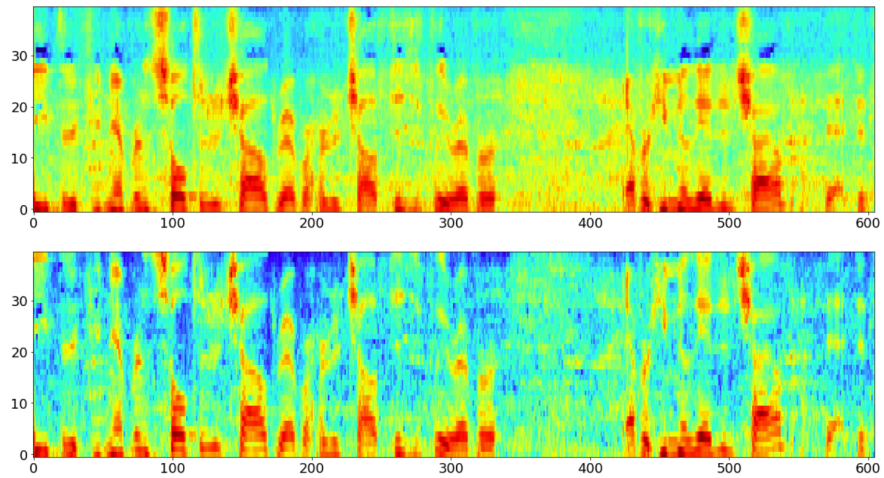


Figure 40.1.9: Mel spectrum at 16k sampling rate and 8k sampling rate.

40.1.3.3 *Mel Frequency Cepstral Coefficient (MFCC)*

Mel filter bank coefficients computed in the previous section are highly correlated, which could be problematic in some machine learning algorithms. Therefore, we can apply Discrete Cosine Transform (DCT) to decorrelate the filter bank coefficients, known as Mel-Frequency Cepstral Coeffificients (MFCC). For typical speech applications, the resulting cepstral coefficients 2-13 are retained and the rest are discarded; The reasons for discarding the other coefficients is that they represent fast changes in the filter bank coefficients and these fine details don't contribute to too much to the model performance.

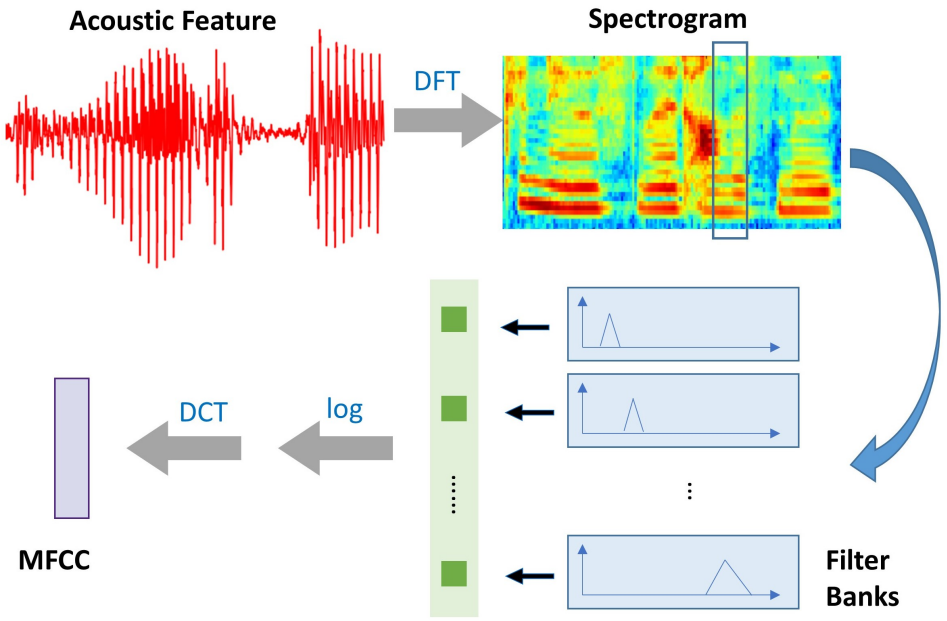


Figure 40.1.10: The computational flow for MFCC features.

We can now compute the Mel spectrum of the magnitude spectrum:

$$s(m) = \sum_{k=0}^{N-1} \left[|X(k)|^2 H_m(k) \right]$$

Where $H_m(k)$ is the weight given to the k^{th} energy spectrum bin contributing to the m^{th} output band. Discrete Cosine Transform (DCT) Finally, we take the log of the spectrum and a Discrete cosine Transform is applied. We obtain the Mel-Frequency Cepstral Coefficients (MFCC), and since most of the information is gathered in the first few coefficients, we can only select the first few ones (usually 12 or 20).

$$c_n = \sum_{m=0}^{M-1} \log_{10}(s(m)) \cos\left(\frac{\pi}{M} n \left(m - \frac{1}{2} \right) \right)$$

There we are, we obtained the MFCC coefficients describing the input signal window.

Finally, and espacially in Speaker Verification tasks, the cepstral mean vector is substracted from each vector. This step is called Cepstral Mean Substraction (CMS) and removes slowly varying convolutive noises. Cepstral mean variance normalization (CMVN) minimizes distortion by noise contamination for robust feature extraction by linearly transforming the cepstral coefficients to have the same segmental statistics (mean 0, variance 1).

Further, we also include first-order and second-order differences of MFCC as additional dynamic features. In total, the feature vector in a speech recognition system is usually a combination given by $X_i = (C_i, \Delta C_i, \Delta\Delta C_i)$.

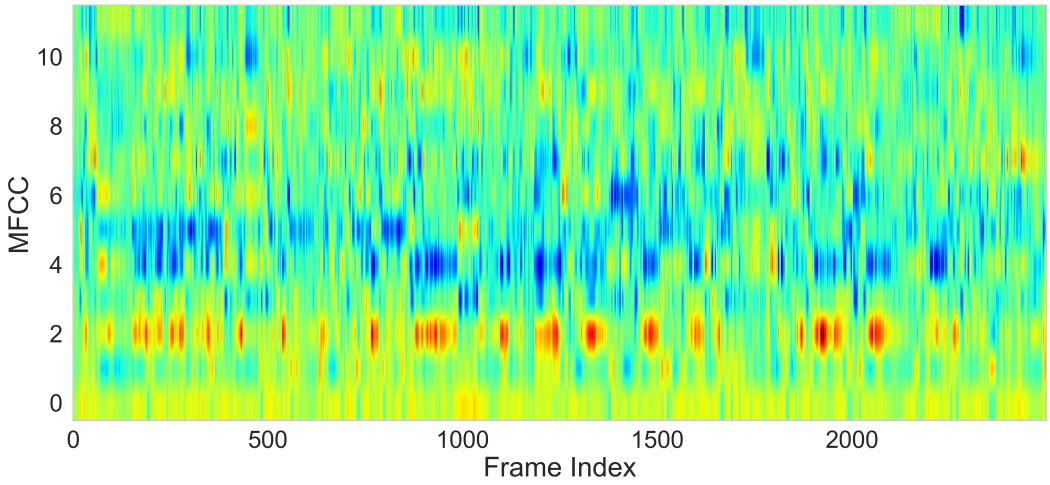


Figure 40.1.11: An example MFCC feature.

40.2 Classical speech recognition model

40.2.1 Introduction

Automatic speech recognition (ASR), also known as speech-to-text, refers to a collection of methods that convert an audio to a sequence of words. The essential objective of ASR is to convert speech into text. From this perspective, an ASR system is also called a TTS (Text to Speech) system. ASR plays a fundamental role to enable human beings to interact with computers or other smart devices (e.g., smart phones) via natural communication. Providing convenience and efficiency to our daily life and work, ASR have found broad applications ranging from earlier voice command and control, interactive voice response, call and meeting transcription, to more recent voice search and interactions with mobile and edge devices such as Siri on iPhone and Alexa smart speakers.

The complexity of an ASR system and its technical development routes depend on multiple factors, including but not limiting to

- the size of vocabulary
- robustness to acoustic environments (noise and reverberation) and channel effects.
- speaker-specific vs speaker independent
- speaking style such as spontaneous, continuous speaking vs. isolated-word speaking.
- Language (e.g., English, Chinese, etc.) and accent.
- Computation and data resources.

Simple ASR systems include one that can only recognize isolated words from a small vocabulary (e.g., 0-9), whereas sophisticated ones include speaker independent large-vocabulary continuous speech recognition that aims to fully empower flexible, intelligent human-machine interactions.

For non-specific-speaker, large vocabulary continuous speech recognition tasks, HMM-based technology and deep learning technology are the key to its breakthrough. In the classical HMM-GMM system, we use HMMs to model the temporal evolution of unobserved speech states and Gaussian mixture models (GMMs) to model the observed acoustic inputs, typically a short window of frames of coefficients that characterize the acoustic inputs.

40.2.2 Mathematical formulation

40.2.2.1 *The Big picture*

Suppose the input is an audio sequence $O = o_1, ..., o_T$ and output is a text sequence $W = \{w_1, ..., w_m\}$. The goal is to seek a probabilistic model (with proper model parameters) that maximizes the likelihood of N sample pairs $\{(O_i, W_i)\}$, given by

$$\prod_{n=1}^{N} P(W_n|O_n)$$

. ASR deals with a sequence mapping from a T -length speech feature sequence, $X = \left\{\mathbf{x}_t \in \mathbb{R}^D \mid t = 1, \ldots, T\right\}$, to an N length word sequence, $W = \left\{w_n \in \mathcal{V} \mid n = 1, \ldots, N\right\}$. Here, $\mathbf{x}_t$ is a D -dimensional speech feature vector (e.g., log Mel filterbanks) at frame t, and w_n is a word at position n in the vocabulary, $\mathcal{V}$

Based on Bayes' Theorem, the probability for a single sample pair (O, W), we have inference probability

$$P(W|X) = \frac{P(O|W)P(W)}{P(X)}.$$

In the decoding phase, we seek an optimal text sequence W^* such that

$$W^* = \arg\max_W P(X|W)P(W)$$

where $P(X|W)$ is given by an acoustic model and $P(W)$ is given by a language model that specifies the probability of word sequence W.

Since $P(W)$ is typically obtained from an external language model trained on other text data, the main challenge of ASR is the likelihood function $P(X|W)$. Directly modeling $P(X|W)$ can be intractable due to following reasons:

- The large size of the vocabulary of V and new words not in the vocabulary cannot be recognized.
- There is a huge variation of pronunciation space associated even with a single word. The acoustic observation depends on how each phoneme in a word is pronounced. A phoneme is a unit of sound.

Instead, a more manageable option is to exploit the fact that the pronunciation of each word's pronunciation comprises different phonemes. Let the phoneme sequence of $Q = \{q_1, ..., q_T\}$, the probabilistic model is rewritten as

$$P(X|W) = \sum_Q P(X, Q|W) = \sum_Q P(O|Q, W)P(Q|W) \approx \sum_Q P(X|Q)P(Q|W),$$

where $P(X|Q)$ is the acoustic-phoneme model, and $P(Q|W)$ is the phoneme model, which specifies the decomposition of a word's pronunciation into a combination of different phonemes. We also make the conditional independence assumption that $P(X|Q,W) = P(X|Q)$. The summation over Q is because a word can have different pronunciations depending on its context, although it is not that common.

Usually, a phoneme model is usually given by a dictionary mapping from a word to phonemes. For example,

$$cat \rightarrow KAET$$

$$good \rightarrow GUHD$$

$$man \rightarrow MAEN$$

$$one \rightarrow WAHN$$

$$punch \rightarrow PAHNCH$$

where $K, AE, T, G, UH, N, W, P, AH, CH$ are symbols of phonemes.

Modeling at the phoneme level might sufficient for speaker-specific small vocabulary recognition system. For non-specific speaker recognition system, the phoneme level is still not fine-grained enough such that the distribution $P(X_t|Q_t)$ cannot be directly fit by a statistical model like GMM (Gaussian mixture model), because phoneme pronunciation also has a large variation depending on where the phoneme is located, the speaking speech and style.

In practice, we use HMM-GMM to model the probability $P(X|S)$. Here the HMM bfstate sequence, $S = \{s_t \in \{1, \ldots, J\} \mid t = 1, \ldots, T\}$. And typically one phone is decomposed into three states, representing the start, the middle, and the end of the phone pronunciation. Since hidden HMM states self-transition to itself, HMM addresses the speaking speed issue.

With the granularity at the HMM state level, we can factorize $p(W|X)$ into the following three distributions:

$$\begin{aligned} & p(W|X) \\ =& p(X|W)p(W) \\ =& \sum_S p(X|S,W)p(S|W)p(W) \\ \approx& \sum_S p(X|S)p(S|W)p(W) \end{aligned}$$

The three factors, $p(X|S)$, $p(S|W)$, and $p(W)$, are the acoustic, lexicon, and language models, respectively. We have applied a conditional independence assumption (i.e., $p(X \mid S, W) \approx p(X \mid S)$) to simplify the dependency of the acoustic model.

The summation over S means that given a word sequence W, there could be multiple HMM state sequence S associated with W, with each state sequence has different probability.

A traditional speech recognition system consisting of several components [Figure 40.2.1].

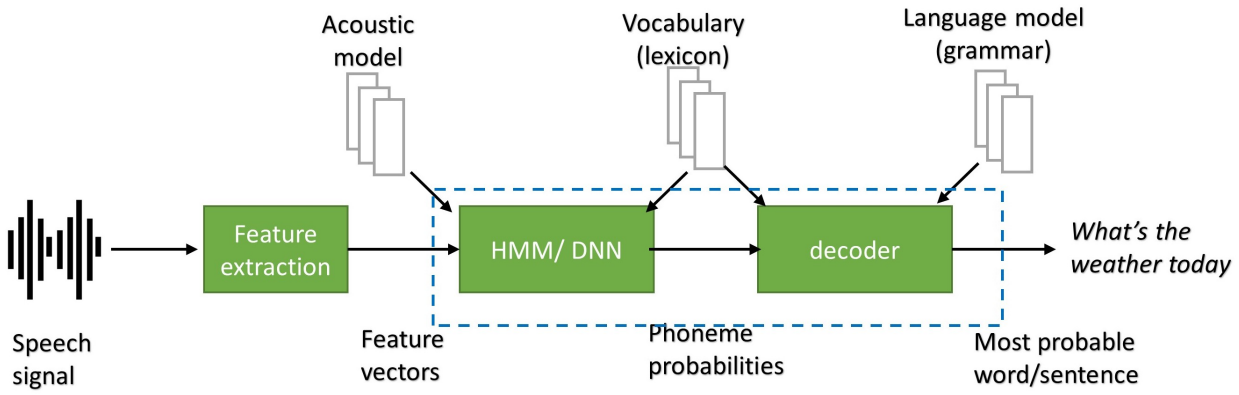


Figure 40.2.1: A traditional end-to-end speech recognition system

40.2.2.2 *Acoustic model component*

The acoustic model $p(X|S)$ is further factorized by using a probabilistic chain rule and conditional independence assumption as follows:

$$\begin{aligned} p(X \mid S) &= \prod_{t=1}^{T} p\left(x_t | x_1, \ldots, x_{t-1}, S\right) \\ &\approx \prod_{t=1}^{T} p\left(x_t | s_t\right) \end{aligned}$$

where the framewise likelihood function $p\left(x_t|s_t\right)$ is given the HMM observation model. Note that to calibrate the framewise observation model, we also need framewise state alignment (i.e., the ground truth of underlying state $s(t)$ for frame t), which is however given by iterative force alignment.

40.2.2.3 *Lexicon model*

The lexicon model $p(S|W)$ is also factorized by using a probabilistic chain rule and conditional independence assumption (1st-order Markov assumption) as follows:

$$\begin{aligned} p(S \mid W) &= \prod_{t=1}^{T} p\left(s_t \mid s_1, \ldots, s_{t-1}, W\right) \\ &\approx \prod_{t=1}^{T} p\left(s_t \mid s_{t-1}, W\right) \end{aligned}$$

This probability is represented by an HMM state transition given W. The conversion from W to HMM states is deterministically performed by using a pronunciation dictionary through a phoneme representation. In the decoding stage, we map connected HMM states to phenomes, then map connected phonemes to words.

Remark 40.2.1. There could be different realization of S, for example

$$\begin{gathered} s_1^{(1)}, s_2^{(1)}, s_3^{(1)}, \ldots, s_T^{(1)} \\ s_1^{(2)}, s_2^{(2)}, s_3^{(2)}, \ldots, s_T^{(2)} \\ s_1^{(3)}, s_2^{(3)}, s_3^{(3)}, \ldots, s_T^{(3)} \end{gathered}$$

each has different realization probabilities. For simplicity, we usually assume that the conversion from word sequence to HMM states deterministic through the usage a pronunciation dictionary. In this way, $P(S|W)$ actually becomes a deterministic mapping.

Using the lexion dictionary also bring the out-of-vocabulary (OOV) issue. The words that are not in the dictionary cannot be recognized.

40.2.2.4 *Language models*

The language Model $p(W)$ is factorized by using a probabilistic chain rule and conditional independence assumption ($(m-1)$ th-order Markov assumption) as an m gram model, i.e.,

$$\begin{aligned} p(W) &= \prod_{n=1}^{N} p\left(w_n \mid w_1, \ldots, w_{n-1}\right) \\ &\approx \prod_{n=1}^{N} p\left(w_n \mid w_{n-m-1}, \ldots, w_{n-1}\right) \end{aligned}$$

Phoneme	Example	Phoneme	Example	Phoneme	Example	Phoneme	Example
AA	*odd*	AE	*at*	AH	*hut*	AO	*ought*
AW	*cow*	AY	*hide*	B	*be*	CH	*cheese*
D	*dee*	DH	*thee*	EH	*Ed*	ER	*hurt*
EY	*ate*	F	*fee*	G	*green*	HH	*he*
IH	*it*	IY	*eat*	JH	*gee*	K	*key*
L	*lee*	M	*me*	N	*knee*	NG	*ping*
OW	*oat*	OY	*toy*	P	*pee*	R	*read*
S	*sea*	SH	*she*	T	*tea*	TH	*theta*
UH	*hood*	UW	*two*	V	*vee*	W	*we*
Y	*yield*	Z	*zee*	ZH	*seizure*	SIL	

Word	Prounciation
Good	G UH D
Hello	HH AH L OW
World	W ER L D

Table 40.2.1: The CMU Pronouncing Dictionary

40.2.3 Acoustic modeling framework

40.2.3.1 *Monophone system*

In the monophonetic system, the pronunciation dictionary maps each word to a sequence of phones that encode their pronunciations. One of most common monophonetic pronunciation dictionary for North American English is the CMU Pronouncing Dictionary, which contains over 134,000 words and their pronunciations. The phoneme set has 39 phonemes. The pronunciation dictionary is constructed based on the ARPAbet symbol set for speech recognition purpose. The phoneme set (plus the silence symbol SIL) and their examples are given by the following table [Table 40.2.1]. To address OOV issue when using the CMU dictionary, we can typically train a grapheme-to-phoneme (g2p) model[2] on these lexicons to predict pronunciations for previously unseen words.

To capture the dynamics of pronouncing each phone, we define three HMM states to associate with the three pronunciation stages (beginning, middle, ending) of the phone [CITE]. Each HMM state is a sub-phonetic unit and comes with a GMM to characterize its emission probability distribution. In the HMM framework, we usually use a 3-state left-to-right phone HMM with self-loops to model the acoustics of each phoneme pronunciation. For example, the phone AA has three states: AA1, AA2, AA3 [Figure 40.2.2].

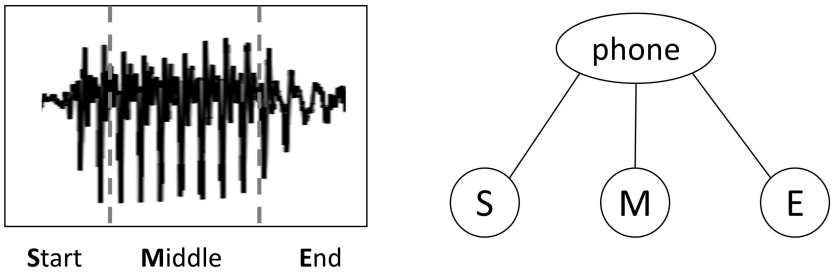


Figure 40.2.2: A monophone acoustic generation is modeled by three phases, and each phase is an HMM state.

40.2.3.2 *Context dependent triphone system*

However, a phoneme is typically pronounced differently in different contexts. For example, the *a* in *cat* and *pat* are pronounced differently because of their different left contexts. This is related to how vocal muscles move together to continuously pronounce different sound units, which is called co-articulation. One approach to address this limitation is to establish a context-sensitive phone model. Consider a triphone model, which enrich the monophones by considering its left and right phones. We denote each triphone by *l*-*x*+*r*, indicating that the center sound is x, the sound on the left is l and the right is r.

For example, the word *world* has the following pronunciation in a monophone system

$$hello \rightarrow HH,\ AH,\ L,\ OW.$$

Its pronunciation in a triphone system is given by

$$hello \rightarrow HH+AH,\ HH-AH+L,\ AH-L+OW,\ L-OW.$$

Depending on if a triphone crosses the word boundary, triphones can further be classified into word-internal triphones and cross-word triphones.

Despite the benefit of a triphone system in capturing finer acoustic features, it also pose a modeling challenge because of the large number of triphones. Consider a monophone system with 40 phones, the number of triphones will be $40^3 = 64000$. Since each triphone is associated with 3 HMM states and 3 GMM emission models. If each GMM has 10 Gaussian distributions, there will be close to 2 Million Gaussian distributions. If acoustic feature is the 39 MFCC and each Gaussian is characterized by 39 means and 39 variances, there will be close to 150 Million model parameters. Too many models to train!

To deal with insufficient data to train HMMs, we can cluster HMM states with similar emission probability together to share the GMM, which will enable better estimation of

the GMMs. These tied HMM triphone states is also known as **senone** [3]. The clustering is usually achieved using decision tree[4].

40.2.4 Decoding

After training the model, we can use the decoder to search state spaces under pronunciation dictionary constraints. The goal is to find the most probable state sequence, where the sequence probability is the sum of acoustic and language model probability) is the largest. The search process can use the Viterbi algorithm. To achieve efficient search in the huge search space, we usually need to prune unlikely path in advance or only keep top B candidate paths (known as Beam Search). In real-world applications, efficient search is realized via the weighted finite-state transducer method[**mohri2002weighted**], which compiles a static graph based on the pronunciation dictionary and performs search on it.

Decoding process might also have two passes. In the first pass, a small language model is used and the decoder output N-best results, where N is usually 100-1000. In the second pass, a larger, more complex model is used to rescore the N-best result from the pass and output the best results.

40.3 Speech data augmentation

40.3.1 Realistic noise and reverberation augmentation

To enhance the robustness of speech and speaker recognition models, we can add realistic noise and reverberation to training dataset[5]. The resulting augmentation increases the amount and diversity of the existing training data.

The reverberation effect can be achieved by convoluting room impulse responses (RIR) with audio. The most common used impulse signal is the simulated RIRs described in [6]. For a rich set of noises, we can use the Musan dataset, which is a large corpus of music, speech, and noise recordings.

- babble: Three to seven speakers are randomly picked from MUSAN speech, summed together, then added to the original signal (13-20dB SNR).
- music: A single music file is randomly selected from MUSAN, trimmed or repeated as necessary to match duration, and added to the original signal (5-15dB SNR).
- noise: MUSAN noises are added at one second intervals throughout the recording $(0-15$ dB SNR). - reverb: The training recording is artificially reverberated via convolution with simulated RIRs.

If we are given a noise signal N and like to add it to a signal S at a specified target SNR, we have

$$S_{aug} = S + \alpha \cdot N,$$

where

$$\alpha = \frac{1}{10^{SNR/20}} \frac{A_{signal}}{A_{noise}}.$$

Note that we can get α from

$$SNR = 20 \log_{10} \frac{A_{signal}}{\alpha \cdot A_{noise}}.$$

40.3.2 SpecAugment

The methods mentioned above operates on the raw waveform data. SpecAugment [7] method is a simple data augmentation technique that directly operates on the spectrograms, which are usually used as the input features to many speech and speaker recognition systems.

Two most effective component in SpecAugment are

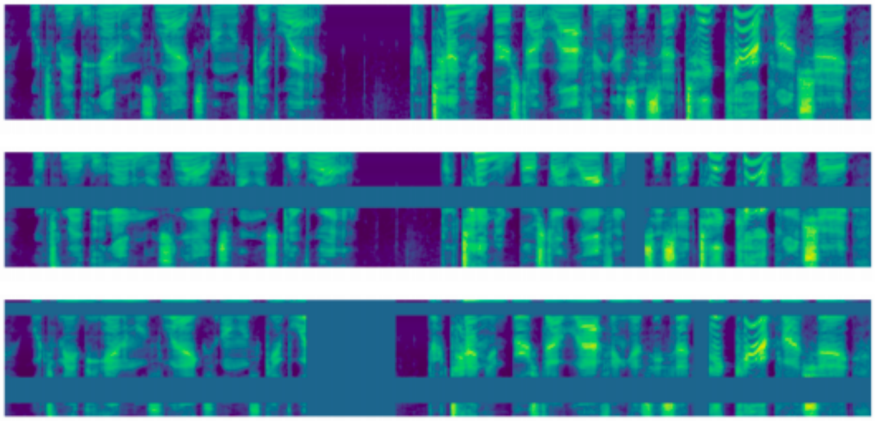


Figure 40.3.1: Demonstration of SpecAugment technique. Upper is the original spectrogram, middle and bottom are two augmented spectrogram with frquency masking (rows) and time masking (columns)

- **Time masking**: Randomly selected t consecutive time steps $[t_0, t_0 + t)$ are masked, where t is chosen from a uniform distribution from 0 to the time mask parameter T, and t_0 is chosen from $[0, T - t)$ where T is the total number of frames.

The resulting spectrogram after SpecAugment is shown in Figure 40.3.1.

40.4 Deep learning models

40.4.1 Motivation and overview

Classical ASR systems are sophisticated pipelines that are comprised of separate acoustic, pronunciation, and language modeling components that are optimized independently. The acoustic model is responsible to recognize phonemes from acoustic features, via either HMM model or neural network. The pronunciation model is responsible for mapping the sequences of phonemes produced by the acoustic model into word sequences. Finally, word sequences are used to guide the decoding process by scoring of hypotheses of word sequences. After decades of optimization on individual components, classical ASR systems have achieved state-of-the-art on standard testing benchmarks within clean or noisy environments. Still, classical ASR systems have various drawbacks that we aim to eliminate.

- Optimization: The above modules are optimized separately with different objectives, which may result in incoherence in optimization, where each module is not trained to match the other modules. Improvement on classical ASR systems are typically achieved via module-specific optimizations (e.g., acoustic modeling is improved by replacing GMM by HMM).
- Feature engineering and data preparation: Extensive efforts in classic ASR models are directed towards acoustic modeling units (e.g., phonemes) and build the pronunciation model to bridge the connection from acoustic units to words. Feature engineering efforts from linguistic experts are needed to choose appropriate acoustic units. Additional feature selection and clustering are also quite involved. To train classical ASR, time-aligned training data is required. Time-aligned training data is obtained by using an existing model for alignment.
- Complicated decoding: Due to the large search space constructed to link sound units to words, efficient decoding in classical ASR requires more than Viterbi algorithm and beam search. Instead, finite state transducers are often used and its construction and implementation are very complicated.
- Generalization to new application and new languages: Classical ASR require laborious feature engineering and optimization of each module. Therefore, it is quite difficult to adapt an existing ASR systems for new applications, especially for new languages.

Recently, there has been growing interest on end-to-end speech recognition models, the benefit of end-to-end model includes

- Simplified feature engineering and data preparation. Unlike traditional ASR model, end-to-end model can be trained on character-level acoustic unit and will not rely on time-aligned data.

- Large learning capacity of nerual networks enables the learning of robust acoustic model. Typically, all components inside a model are jointly optimized together.

However, training a deep learning ASR model typically requires significantly larger training data as well as computation power.

We will discuss three major types of end-to-end architectures for ASR:

- Attention-based method where attention mechanism to perform alignment between acoustic frames and recognized symbols;
- Acoustic encoder with connectionist temporal classification (CTC) on output symbols;
- RNN-transducer with joint modeling on acoustics and previous output symbols.

40.4.2 Seq2Seq attention model

40.4.2.1 *Framework*

The model encodes the input $\mathbf{x}$ into a sequence of feature vectors, then computes the probability of the next output y_u as a function of the encoded input and previous outputs. The attention mechanism allows the decoder to look at different parts of the input sequence when predicting each output.

Attention models can be applied to any problem, but they are not always the best choice for certain problems, like speech recognition, for a few reasons? - The attention operation is expensive for long input sequences. The complexity of attending to the entire input for every output is $O(TU)$ -and for audio, T and U are big. - Attention models cannot be run online (in real time), since the entire input sequence needs to be available before the decoder can attend to it. Attention models also don't take advantage of the fact that, for speech recognition, the alignment between inputs and outputs is monotonic: that is, if word A comes after word B in the transcript, word A must come after word B in the audio signal (see image below, from Chan et al., for an example of a monotonic alignment). The fact that attention models lack this inductive bias seems to make them harder to train for speech recognition; it's common to add auxiliary loss terms to stabilize training.

40.4.2.2 *Listen, Attend, and Spell*

Listen, Attend, and Spell (LAS) is a model architecture [Figure 40.4.1] based on seq2seq framework with attention mechanism[8]. The listener is an encoder consisting of three pyramidal Bidirectional LSTM layers. The pyramidal structure is required to reduce the length of encoder output h, which is shown to be critical for input speech signals that

usually comprise of hundreds and thousands of frames. Without pyramidal structure, the encoder output would have the same large length of the input, which often leads to costly computation and the difficulty of finding relevant information in the attention module.

As a comparison, in a typical deep bidirectional LSTM architecture, the output at the ith time step, from the jth layer is given by

$$h_i^{(j)} = \text{BLSTM}\left(h_{i-1}^{(j)}, h_i^{(j-1)}\right);$$

while in the pyramidal bidirectional LSTM model, we first concatenate the outputs at consecutive steps of each layer and then feeding then to the next layer, we have

$$h_i^{(j)} = \text{pBLSTM}\left(h_{i-1}^{(j)}, \text{Concat}[h_{2i}^{(j-1)}, h_{2i+1}^{(j-1)}]\right).$$

By stacking three pyramidal layers, we actually reduce the output length by a factor of $2^3 = 8$.

In the decoder module, at every output step, an input token y^{i-1} is first projected to a state s_{i-1}, which is later combined with an attention-based context vector c_{i-1} and are fed into an RNN together. The final output is a probability distribution over the next character $y_i$conditioned on all the characters seen previously. Note that the decoder state s_i is a function of the previous state s_{i-1}, the previous input y_{i-1} and previous context c_{i-1}. Specifically,

$$\begin{aligned} c_i &= \text{Attention}\left(s_i, h\right) \\ s_i &= \text{RNN}\left(s_{i-1}, y_{i-1}, c_{i-1}\right) \\ P\left(y_i | x_{1:T}, y_{1:i-1}\right) &= \text{CharacterDistribution}\left(s_i, c_i\right) \end{aligned}$$

where CharacterDistribution is a feed-forward network with Softmax outputs over characters, and RNN is a 2-layer LSTM.

At each time step, i, the Attention module generates a context vector, c_i, which extract most relevant information from $h = (h_1, ..., h_U)$ needed to generate the next character. We have following mathematical summary.

$$\begin{aligned} e_{i,u} &= \left\langle \phi\left(s_i\right), \psi\left(h_u\right) \right\rangle \\ w_{i,u} &= \frac{\exp\left(e_{i,u}\right)}{\sum_u \exp\left(e_{i,u}\right)} \\ c_i &= \sum_{u=1}^{U} w_{i,u} h_u \end{aligned}$$

where ϕ and ψ are nonlinear transformation represented feed-forward neural networks. After convergence, the attention weight wi vector distribution is typically very sharp and localized, and focused on only a few frames of h [Figure 40.4.2]

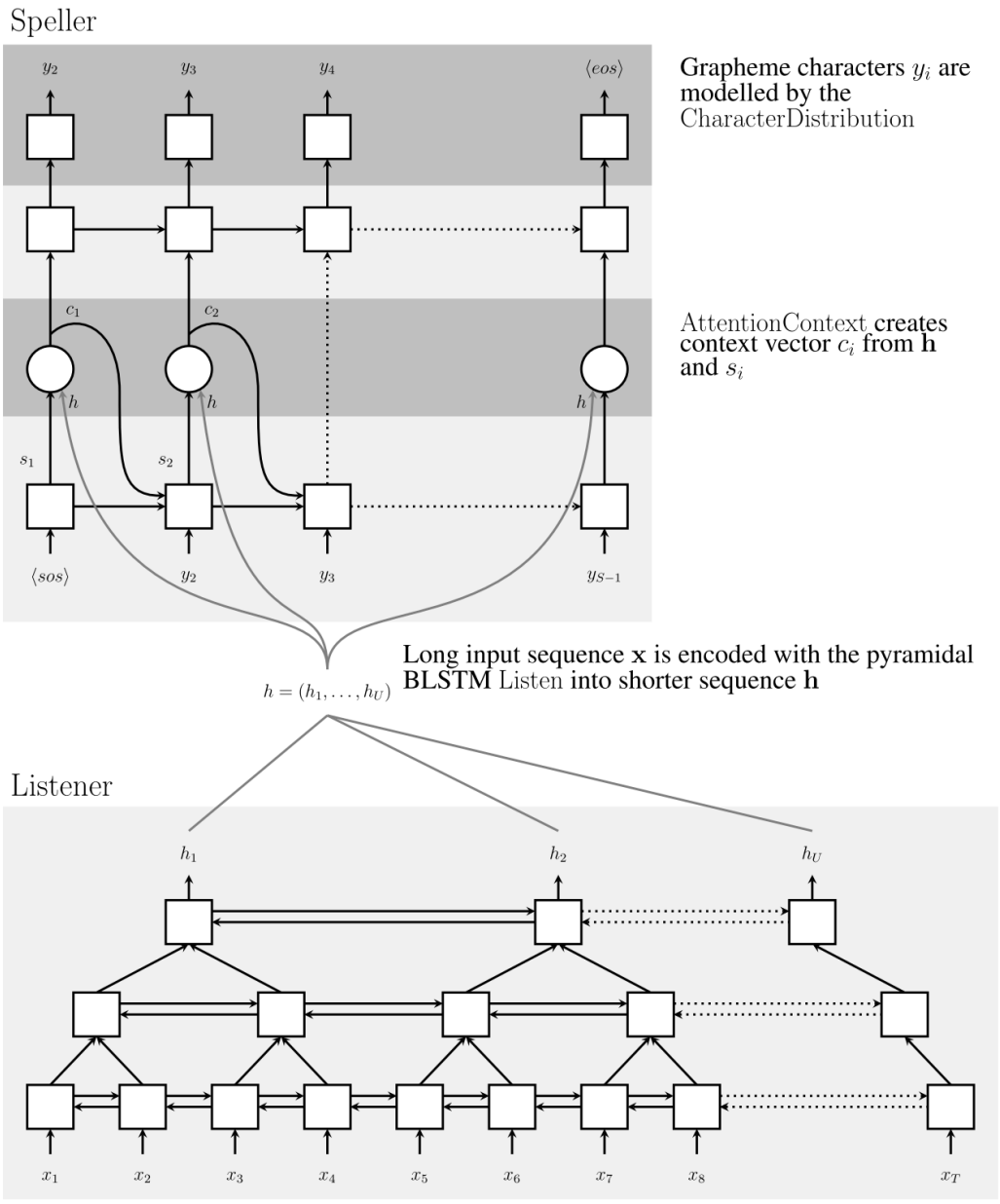


Figure 40.4.1: Listen, Attend and Spell (LAS) architecture. The listener is a pyramidal bidirectional LSTM encoding input sequence $(x_1, ..., x_T)$ into high level features $h = (h_1, ..., h_U)$. The speller is an attention-based decoder generating the y characters from h. Image from [8].

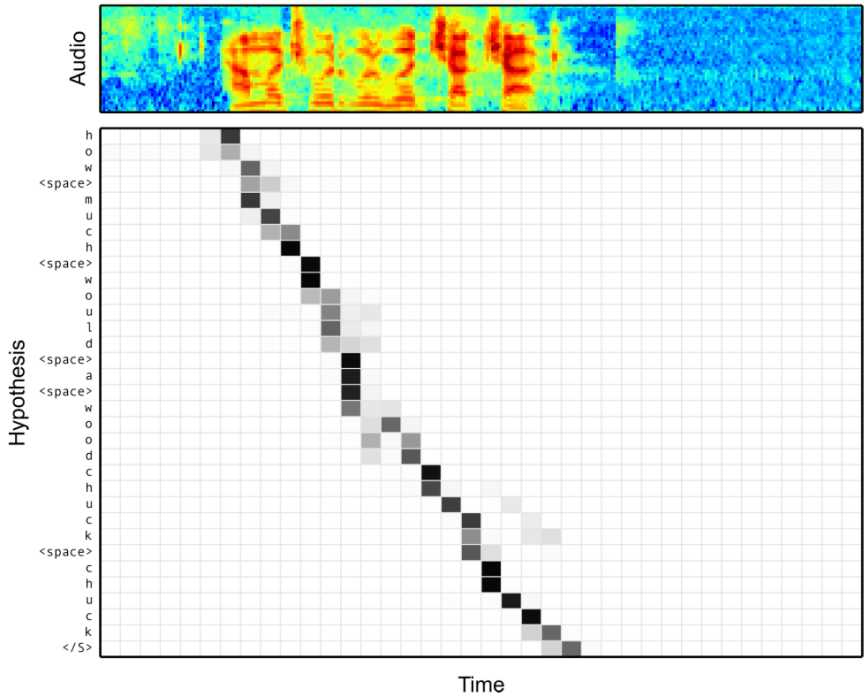


Figure 40.4.2: Alignment, as indicated by attention, between character outputs and audio signal produced by the Listen, Attend and Spell (LAS) model for the utterance "how much would a woodchuck chuck". Image from [8].

The whole model can be trained in an end-to-end fashion. The goal is to maximizes the log probability of target sequence $y_1, ..., y_N$ with teacher forcing, that is

$$\max_{\theta} \sum_i \log P\left(y_i | x_1, ..., x_T, y_1, ..., y_{i-1}; \theta\right)$$

where $y_1, ..., y_{i-1}$ are previous ground truth characters.

Note that during inference stage, the ground truth is missing. To enhance robustness when the previous prediction is bad, we sometimes feed previous predicted character as inputs in the next step predictions, that is

$$\begin{aligned} \tilde{y}_{i-1} &\sim \text{ CharacterDistribution } \left(s_{i-1}, c_{i-1}\right) \\ \max_{\theta} &\sum_i \log P\left(y_i | x_{1:T}, \tilde{y}_{1:i-1}; \theta\right). \end{aligned}$$

In the decoding stage, we typically leverage a separately trained language model to boost performance and solve the following optimization instead,

$$y^* = \arg\max_y P(y|x)P(y),$$

where $P(y|x)$ is from the LAS model and $P(y)$ is from a separate language model.

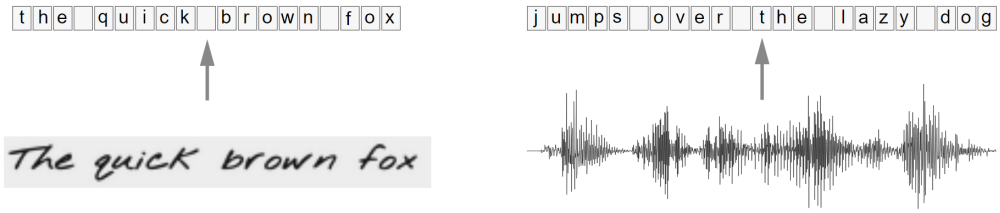


Figure 40.4.3: (left)Handwriting recognition: The input can be (x,y)(x,y) coordinates of a pen stroke or pixels in an image. (right) Speech recognition: The input can be a spectrogram or some other frequency based feature extractor. Image from [9].

40.4.3 Acoustic encoder with CTC loss

40.4.3.1 *Fundamentals*

A deep learning based ASR model aims to learn the mapping from sequence of acoustic feature inputs to a sequence of symbols (e.g., characters) that can be easily converted to words. Let's consider mapping input sequences $X = [x_1, x_2, \ldots, x_T]$, such as audio, to corresponding output sequences $Y = [y_1, y_2, \ldots, y_U]$, such as transcripts. Unfortunately, we don't know how the characters in the transcript align to the audio. This alignment issue makes specifying supervised loss hard as we don't know the ground truth for each input time-step [Figure 40.4.3].

Attention-based method use attention to perform alignment, which however does not take into account the monotonic alignment structure, i.e., if x_{t_1}, x_{t_2} aligns with $y_{t'_1}, y_{t'_2}$, it is always $t_1 < t_2$ if and only if $t'_1 < t'_2$.

The CTC (Connectionist Temporal classification) loss, proposed by Graves [10], is a way of training end-to-end models without requiring a frame-level alignment of the target labels for a training utterance. CTC augments the set of target labels with an additional "blank" symbol, denoted $\langle b \rangle$.

Given a reference label sequence y, we say hypothesis sequence y' is a **valid alignment** of y if we can transform y' to y by first collapsing consecutive repeated labels and then removing any blank symbols in y'. Note that to avoid undesired collapsing of same label, we need to have $\langle b \rangle$ between them. In this way, we can differentiate between alignments which collapse to *hello* and those which collapse to *helo*.

Consider the label sequence $y = \{c, a, t\}\}$ corresponding to the word *cat*, valid alignments include but not limit to

- $\langle b \rangle, c, c, \langle b \rangle, a, t$
- c, c, a, a, t, t

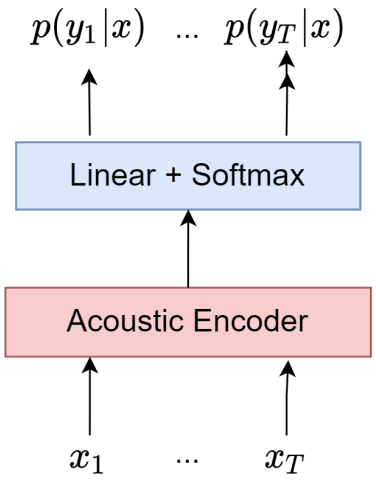


Figure 40.4.4: Typical acoustic encoder with CTC loss.

- $c, c, \langle b \rangle, a, \langle b \rangle, t$

Any alignments that are not valid are called **invalid alignments**, which include

- $c, \langle b \rangle, c, \langle b \rangle, a, t$ (it reduces to *ccat*)
- $c, \langle b \rangle, \langle b \rangle, \langle b \rangle, t, t$ (it reduces to *ct*)

The CTC alignments have a few notable properties. First, the allowed alignments between X and Y are monotonic. If we advance to the next input, we can keep the corresponding output the same or advance to the next one. A second property is that the alignment of X to Y is many-to-one. One or more input elements can align to a single output element but not viceversa. This implies a third property: the length of Y cannot be greater than the length of X.

A typical acoustic encoder with CTC loss is shown in Figure 40.4.4. Given input sequences $x = [x_1, x_2, \ldots, x_T]$, we model the conditional probability of hypothesis y'_t by

$$h_1, ...h_T = \text{AoucsticEncoder}(x_1, ..., x_T)$$
$$p(y'_t|x) = \text{Softmax}(W_h h_t + b_h), t = 1, ..., T$$

CTC loss function defines the probability of the actual output sequence conditioned on the acoustics as:

$$P(y|x) = \sum_{y' \in \mathcal{A}} P\left(y'|x\right) = \sum_{y' \in \mathcal{A}} \prod_{t=1}^{T} P\left({y'}_t|x\right)$$

where we sum over all possible valid alignments and we assume that labels at each time step are conditional independent, given the acoustics.

We often use deep recurrent neural network as the acoustic encoder and the encoder is trained to maximize $P(y|x)$ by gradient descent. We also need the forward-backward algorithm to efficiently compute the summation in $P(y|x)$ [9, 10].

CTC model is a simple framework to enable end-to-end training. But CTC models have a couple problems:

- Condition independence assumption on the output sequence. The outputs are assumed to be independent of each other given observed acoustic observation. The result is that CTC models often produce outputs that are obviously wrong, like "I eight food" instead of "I ate food". Getting good results with CTC usually requires a search algorithm that incorporates an additional language model.
- The output sequence length U has to be smaller than the input sequence length T. Although typically T is much larger than U, but this requirement prevents us from using a model architecture that does a lot of pooling in feature extraction.

40.4.3.2 *Path summation*

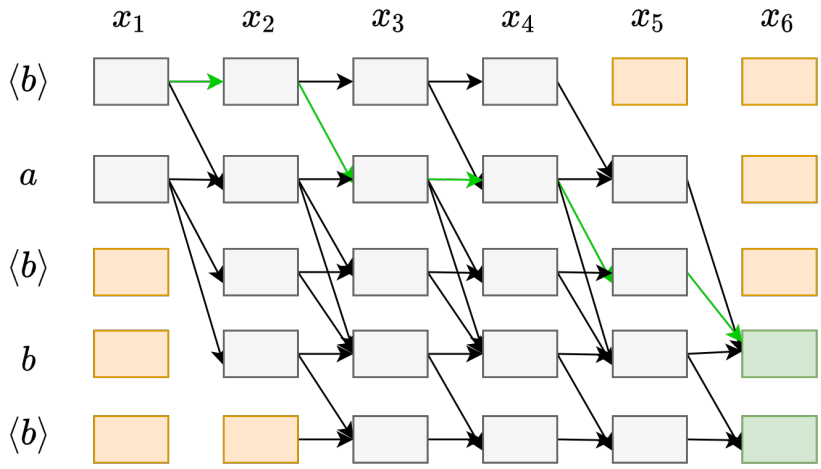


Figure 40.4.5: Illustrating valid alignments and the summation of path probability.

Here we would like to illustrate how we can compute $P(y|x)$ efficiently using dynamic programming, also known as forward algorithm. Let the ground truth label sequence be ab, we can augment $\langle b \rangle$ symbols and then the valid alignments are just different paths in the diagram [Figure 40.4.5].

There are two valid starting nodes and two valid final nodes since the $\langle b \rangle$ at the beginning and end of the sequence is optional. The probability $P(y|x)$ is the sum of the two final nodes.

We can use recursive relationship to compute probabilities of paths accumulated at the certain node $\alpha_{s,t}$.

$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t\left(z_s \mid x\right)$$

where $p_t(z_s|x)$ is the current character subsequences after $t-1$ input steps.

40.4.3.3 *Decoding*

After we've trained the model, we'd like to use it to find a likely output for a given input. More precisely, we need to solve:

$$Y^* = \underset{Y}{\text{argmax}} p(Y \mid X)$$

One heuristic is to take the most likely output at each time-step. This gives us the alignment with the highest probability:

$$A^* = \underset{A}{\text{argmax}} \prod_{t=1}^{T} p_t\left(a_t \mid X\right)$$

We can then collapse repeats and remove $\langle b \rangle$ tokens to get Y.

40.4.3.4 *Deep Speech*

One early successful end-to-end speech recognition system based on CTC loss is deep speech[11]. The core component of Deep Speech the recurrent neural network (RNN) with CTC loss. RNN is trained to take frame-level acoustic features $x = (x_1, ..., x_T)$ as the input and generate characters $y = (y_1, ..., y_T$ as the output. Specifically, the output is a sequence of character probabilities, with $\hat{y}_t = P\left(c_t \mid x\right)$, where $c_t \in \{\text{a}, \text{b}, \text{c}, \ldots, \text{z}$, space, apostrophe, blank $\}$. The output probabilities, together with ground-truth labels, can be used to compute CTC loss.

The RNN model is composed of 5 layers of hidden units[CITE]. For an input x, the hidden units at layer l are denoted $h^{(l)}$ with the convention that $h^{(0)}$ is the input. The first three layers are not recurrent. For the first layer, at each time t, the output depends on the spectrogram frame x_t along with a context of C frames on each side. [1] The remaining non-recurrent layers operate on independent data for each time step. Thus, for each time t, the first 3 layers are computed by:

$$h_t^{(l)} = \text{ClipReLU}\left(W^{(l)} h_t^{(l-1)} + b^{(l)}\right), l = 1, 2, 3$$

where $\text{ClipReLU}(z) = \min\{\max\{0, z\}, 20\}$ is the clipped rectified-linear (ReLu) activation function and $W^{(l)}, b^{(l)}$ are the weight matrix and bias parameters for layer l The fourth

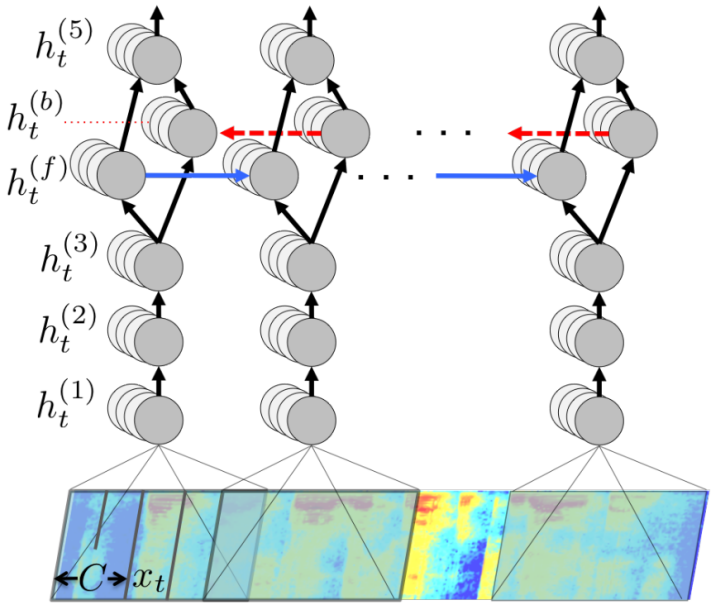


Figure 40.4.6: Multilayer RNN as the feature extractor of Deep Speech architecture.[11]

layer is a bi-directional recurrent layer. The hidden units at the fifth layer will be fed into a linear layer with Softmax to produce distribution over label space. The model will be trained with CTC loss.

In the decoding stage, we utilize the probability output of $P(y|x)$ of the RNN and an externally trained language[2] to guide the decoding process. Particularly, we are seeking the sequence of characters $c_1, c_2, \ldots$ that is most probable based on maximizing the following combined objective:

$$Q(y) = \log(P(y \mid x)) + \alpha \log\left(P_{\text{lm}}(y)\right) + \beta \,\text{WordCount}(y)$$

where α and β are hyperparameters that control the trade-off between the RNN, the language model constraint and the length of the sentence.[3] The term $\mathbb{P}_{\text{lm}}$ denotes the probability of the sequence y according to the language model.

The following table shows the impact of incoporating external language model. Decoding with the RNN output alone often times gives transcripts that sound similar but not linguistic correct.

2 CTC loss does not consider the language model constraint, therefore we need to incorporate one during the decoding stage to improve the performance.

3 Because a language model generally assigns a smaller probability to longer sentence, the term β word count (c) is used to penalize the tendency of choosing short texts.

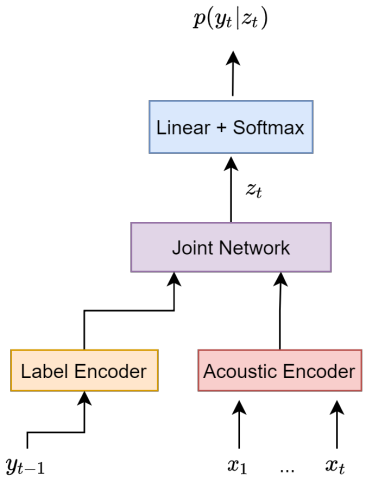


Figure 40.4.7: A typical transducer architecture.

RNN output	Decoded Transcription
what is the weather like in bostin right now	*what is the weather like in boston right now*
prime miniter nerenr modi	*prime minister narendra modi*
arther n tickets for the game	*are there any tickets for the game*

Deep Speech model was trained on massive 5000+ hours training data, which far exceeds hundreds of hours of recording used in traditional ASR system. On the standard task of Switchboard, DeepSpeech's word error rate (WER) is 12.6, while the best solution is 10.4; and on the difficult task of Switchboard, DeepSpeech got the best result of 19.3 at that time.

40.4.4 Transducer architecture

The CTC model is similar to an acoustic model in a traditional ASR system. The **RNN transducer**, or **RNN-T** [12] augments the encoder network from the CTC model architecture with a separate recurrent prediction network over the output symbols, as illustrated in Figure 40.4.7. Intuitively, the encoder can be thought of as an acoustic model, while the prediction network is analogous to a language model.

Given an acoustic feature sequence $x = \{x_1, \cdots, x_T\}$ and its corresponding label sequence $y = \{y_1, \cdots, y_U\}$, where T is the length of the acoustic sequence, and U is the length of the label sequence, RNN-T directly computes the sequence-level conditional

probability of the label sequence given the acoustic features by marginalizing all the possible alignments

$$P(y|x) = \sum_{y' \in \mathcal{A}(y)} P(y'|x)$$

where y' is a path that contains the blank token $\langle b \rangle$, and the function $\mathcal{A}$ denotes all possible label sequence equivalent to y after removing the blank tokens. Like CTC loss, the summation can be efficiently computed by the forward-backward algorithm.

Let's now work through how the network works? At time step t, we have acoustic encoding and label encoding given by h_t^f and h_u^g. That is,

$$\begin{aligned} h_t^f &= \text{AudioEncoder}\left(x_{1:t}\right), \\ h_u^g &= \text{LabelEncoder}\left(y_{0:u-1}\right) \end{aligned}$$

where the first time step, y_0 has the special label $\langle \text{sos} \rangle$ as input.

h_t^f and h_u^g are passed to a joint network, which computes output logits $z_{t,u}$ over the label space, t, in the encoder sequence and label, u, from the prediction network, as follows:

$$\begin{aligned} h_{t,u}^{\text{joint}} &= \tanh\left(Ah_t^f + Bh_u^g + b\right) \\ z_{t,u} &= Dh_{t,u}^{\text{joint}} + d \end{aligned}$$

The conditional probability of the next token is obtained by a Softmax function, i.e.

$$P\left(y_u|z_{t,u}\right) = \text{Softmax}\left(Wz_{t,u} + b\right)$$

where W and b are the weight matrix and bias. Given $P\left(y_u|\ z_{t,u}\right)$, the sequence-level conditional probability can be computed by dynamic programming, and the RNN-T loss can be defined as the negative log-likelihood as:

$$\mathcal{L}_{\text{rnnt}}\left(y, x\right) = -\log P(y \mid x).$$

BIBLIOGRAPHY

1. Lyons, R. *Understanding Digital Signal Processing: Unders Digita Signal Proces_3* ISBN: 9780137028528 (Pearson Education, 2010).
2. Bisani, M. & Ney, H. Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication* **50,** 434–451 (2008).
3. Hwang, M.-Y. & Huang, X. *Subphonetic modeling with Markov states-Senone* in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing* **1** (1992), 33–36.
4. Young, S. J., Odell, J. J. & Woodland, P. C. *Tree-based state tying for high accuracy modelling* in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994* (1994).
5. Snyder, D., Garcia-Romero, D., Sell, G., Povey, D. & Khudanpur, S. *X-vectors: Robust dnn embeddings for speaker recognition* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 5329–5333.
6. Ko, T., Peddinti, V., Povey, D., Seltzer, M. L. & Khudanpur, S. *A study on data augmentation of reverberant speech for robust speech recognition* in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), 5220–5224.
7. Park, D. *et al. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition* in *INTERSPEECH* (2019).
8. Chan, W., Jaitly, N., Le, Q. V. & Vinyals, O. *Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition* in *ICASSP* (2016).
9. Hannun, A. Sequence modeling with ctc. *Distill* **2,** e8 (2017).
10. Graves, A., Fernández, S., Gomez, F. & Schmidhuber, J. *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks* in *Proceedings of the 23rd international conference on Machine learning* (2006), 369–376.
11. Hannun, A. *et al.* Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).
12. Graves, A. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711* (2012).

41

DEEP LEARNING FOR SPEAKER RECOGNITION

41.5.2 Dataset 1778

41.1 Introduction

41.1.1 Background

The usage of fingerprint is widely adopted in fields like authentication and forensics. For example, one can use fingerprints to unlock his or her smartphone or PC for software installation and financial transaction authorization. The uniqueness of fingerprint justifies its usage as an identity proof.

Similarly, in human perception, different people's voice typically sound differently when they speak even the same content. The uniqueness of everyone's voice stems from the variations in the sound producing organs, e.g., vocal cord. Human beings can easily distinguish who is speaking in our household based on the characteristics like pitch. Generally speaking, men's voices are lower in pitch and women's or kids' voices are higher. As a result, we can typically tell the speaker's gender without even knowing the speaker's identity. Factors like gender, age, accent, linguistic background all contribute to the uniqueness of voices.

Like fingerprint, we can define the term **voiceprint** as the characteristics of a speech signal that help identify who is speaking. Voiceprints are usually more abstract than visual fingerprints since voiceprints usually involve characteristics like frequencies derived from a speech signal.

Speaker recognition systems aim to recognize a speaker's identity or verify a person's claimed identity from his or her voice. With the proliferation of smart home/vehicles and mobile application, speaker recognition systems are playing increasingly important roles in enabling natureal interaction between machines/devices and humans. There are three major tasks relevant to speaker recognition:

- Speaker identification, which decides who is speaking among a group of people.
- Speaker verification, which decides if a speaker is who he claims to be.
- Speaker diarization, which partitions an audio signal with multiple speakers into homogeneous segments associated with each individual speaker.

A major challenge for speaker recognition systems operating in real-world conditions is to effectively overcome variability in the speech signal. There are many factors affecting speech presented to a speaker recognition system; these can be loosely grouped into factors independent of the speaker (extrinsic factors), and factors dependent on the speaker (intrinsic factors). Extrinsic factors include environmental noise, room acoustics, and the effects of the microphone and transmission channel. Intrinsic factors include the speaking style and conversational context, the emotional state of the speaker, changes to the speaker's health, and the longer-term impact of aging.

Speaker recognition systems have found many important applications [Figure 41.1.1] such as authentication, access control, personalization, and law enforcement.

Speaker recognition is also broadly relevant to applications from speaker emotion analysis (happy or sad) to music genre prediction, further to audio instrument classification, where we determine which type of instruments (e.g., guitar, piano, violin) that produce a sound sample.

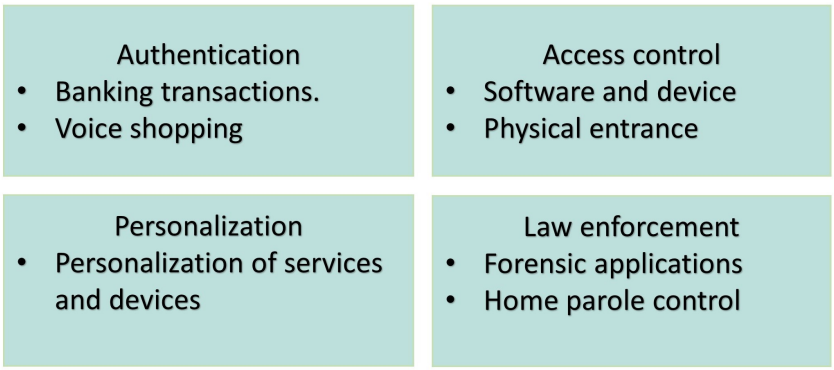
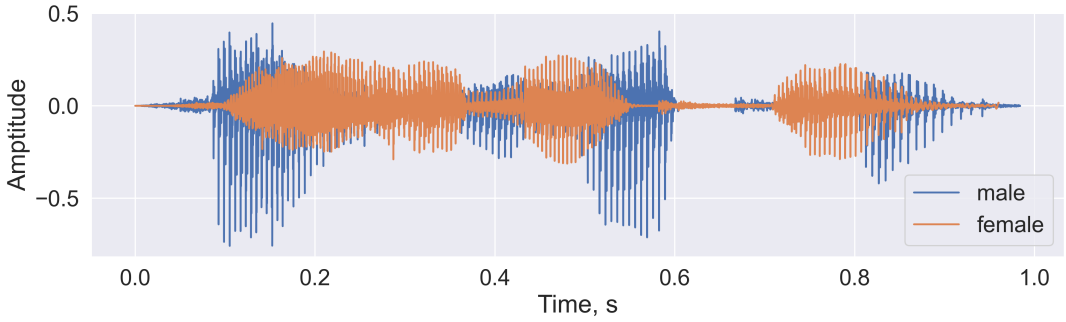
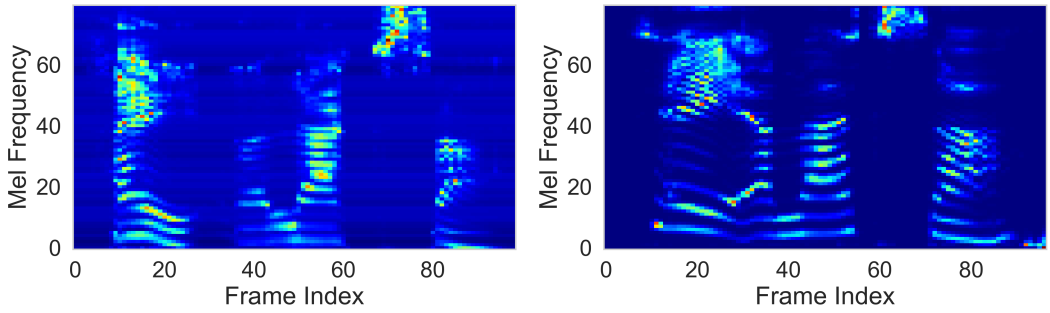


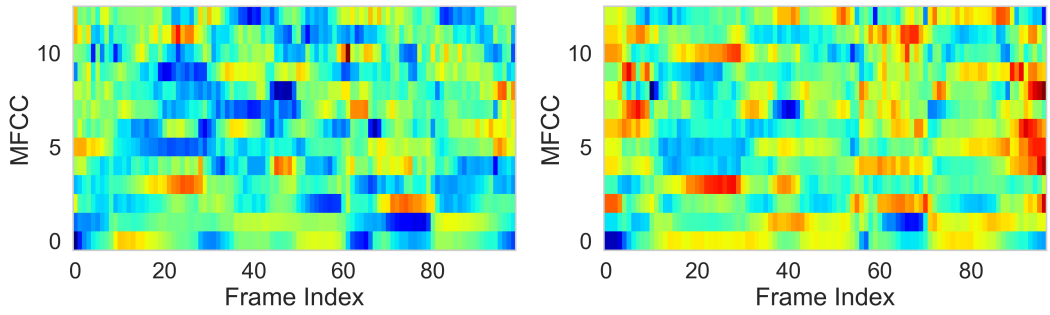
Figure 41.1.1: Speaker recognition system applications.



(a) Wave form of audio signals The audio signals have a sampling rate of 16kHz and last around 1 s.



(b) Mel spectrogram of the male audio signal (left) and the female audio signal (right). There are 80 Mel filter banks in total and around 100 frames.



(c) MFCC (first 13 coefficients) of the male audio signal (left) and the female audio signal (right).

Figure 41.1.2: Audio features of utterances from a male and a female speaking *Hey, Alexa*. Code

41.1.2 Speaker identification and verification

The goal of **speaker identification** is to decide the identity of an unknown speaker from a group of candidates based on his or her voice sample. As a comparison, the goal of **speaker verification** is to verify the claimed identity of a speaker based on his or her voice samples. Speaker identification can be applied in household settings where a voice assistant (such as Google Home or Amazon Alexa) to identify the speaker's identity to enable personalization services like playing one's favorite music and checking one's calendar. Speaker verification is more applicable to identity authorization settings for security applications, such as authorizing financial transactions.

Despite their different application scenarios, speaker identification and verification share rather similar working mechanism, as we will illustrate shortly. Therefore, we sometimes use speaker recognition to refer to both of them.

A speaker recognition process typically has two stages: **enrollment** stage and **recognition** stage. In the enrollment stage, a speaker first enrolls his or her voice in the system. In the second recognition stage, we given an utterance from a test speaker, and the goal is to identify *who is speaking* from a candidate group of speakers and verify the claimed identity of the test speaker.

Because the voice sample is only compared against the model of the claimed speaker, the computational cost is independent of the speaker population. Similar to speaker identification, the verification process can also be categorized into text-dependent verification and text-independent verification.

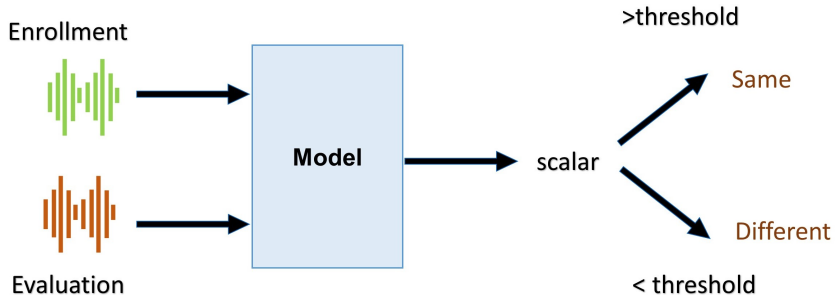


Figure 41.1.3: A generic computational flow for speaker verification

Usually, the voice sample is transformed to first feature vector , and then the feature vector is compared against features from a candidate speaker set. We distinguish **open-set identification** and **closed-set identification**. In open-set identification, the unknown speaker might not lie in the candidate speaker set. On the other hand, in closed-set identi-

fication, the unknown speaker is in the candidate speaker set. Clearly, the computational cost of a speaker identification increases as the candidate set increases.

Further, based on the content of the audio signal, we also distinguish **text-dependent recognition** and **text-independent recognition**. In text-dependent recognition, recognition system knows the text the unknown speaker is going to speak and assumes the speaker is cooperative. This knowledgment can improve performance. In text-independent recognition, the system does not known what the unknown speaker is going to speak.Speech recognition system can be used to determine the text spoken by the person.

Text-dependent recognition is common for security applications where speakers are asked to speak specific words and sentences. Text-independent recognition is common for applications like voice assistant device, customer services.

41.1.3 Evaluation metrics

41.1.3.1 *Error types*

Given a speaker verification system, we desire following property: A speaker will not be falsely rejected and an impostor will not be falsely accepted. As a detection task, we can characterize the system performance in terms of two types of errors, namely **misses** (in which the true speaker is falsely rejected) and **false alarms** (in which an impostor speaker is falsely accepted). To this end, three commonly used metrics for a speaker verification system are listed in the following.

Definition 41.1.1 (metrics in a speaker recognition system).

- ***False Rejection Rate (FRR)****: The speaker false rejection rate (FRR) is the rate that a given speaker is incorrectly rejected. In other words, the prediction is false but the ground truth is true.*
- ***False Acceptance Rate (FAR)****: The speaker false acceptance rate (FAR) is the rate that utterances not belonging to an enrolled speaker are incorrectly accepted as belonging to the enrolled speaker.*
- ***Equal Error Rate (EER)****: This is the rate when its FAR and FRR are equal. EER can serve as a summary of system performance*

Here FRR measures how often a speaker will be falsely rejected and FAR measures how often an impostor will be falsely accepted. Clearly, an adequate system requires both FAR and FRR are low, or collectively ERR is low.

The metrics FRR, FAR, EER can be estimated by evaluating the system on a large number of trials. Particularly, each trial is classified as a positive trial, which consists of a pair of utterances from the same speaker, or as a negative trial, which consists of a pair of utterances from two different speakers.

The system should produce a similarity score, where a higher score indicates a higher likelihood that the trial is positive and vice versa.

At each decision threshold, we can obtain the FRR and FAR.

Definition 41.1.2 (calculation of metrics).

- $$FRR = \frac{FN}{FN + TP},$$
  where the sum of FN and TP is the number of positive trials.
- $$FAR = \frac{FP}{FP + TN},$$
  where the sum of FP and TP is the number of negative trials.

		prediction	
		speaker	*impostor*
ground truth	*speaker*	*TP*	*FN*
	impostor	*FP*	*TN*

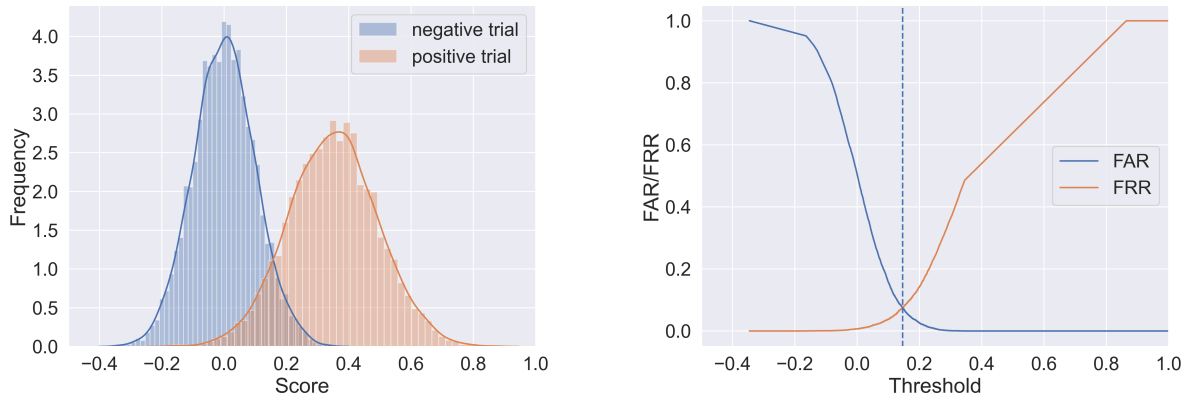
Commonly, a speaker verification computes an embedding vector (usually around 200 to 1000 dimensions) for the utterance from the speaker. The similarity score can be obtained via cosine similarity. In this case, by varying the decision threshold between -1 and 1, we can obtain a FRR-FAR curve. Note that

- FRR is an increasing function of the threshold. As the threshold increases from -1 and 1, FRR increases from 0 to 1. Particularly, when threshold is at -1, all speakers will be accepted and thus FN = 0; when threshold is at 1, all speakers will be rejected and thus TP = 0.
- FAR is an decreasing function of the threshold. As the threshold decreases from 1 and -1, FAR decreases from 1 to 0. Particularly, when threshold is at -1, all impostors will be accepted as speakers and thus TN = 0; when threshold is at 1, all impostors will be rejected and thus FP = 0.

Based on the need of application, we select different threshold of the operating point to achieve the desired FRR or FAR. For example, some applications (e.g., control a smart

device to turn on the TV, or to play music) hope to achieve lower FRR, then we can select a lower threshold such that a true speaker will not be rejected often at the expense that an impostor might get accepted incorrectly.

On the other hand, for applications requires some level of safety and privacy (e.g., control the device to place an shopping order or to open email inbox), we can select a high threshold such that an impostor will be highly likely rejected at the expense that a true speaker might get rejected.



(a) Mel filter banks.

(b) Mel spectrogram.

Figure 41.1.4: Computation of filter bank features.

41.1.3.2 *Detection cost function*

This is, in fact, a family of performance measures introduced by NIST over the years. As mentioned before, the EER does not differentiate between the two errors, which sometimes is not a realistic performance measure. The detection cost function (DCF), thus, introduces numerical costs/penalties for the two types of errors (FA and miss). The a priori probability of encountering a target speaker is also provided. The DCF is computed over the full range of decision threshold values as

$$\text{DCF}(\tau) = C_{\text{MISS}} P(\tau) P_{\text{Target}} + C_{\text{FA}} P_{\text{FA}}(\tau) \left(1 - P_{\text{Target}}\right)$$

Here,

$$C_{\text{Miss}} = \text{Cost of a miss/FR error}$$
$$C_{\text{FA}} = \text{Cost of an FA error}$$

P_{Target} = Prior probability of target speaker. $P_{\text{Miss}}(\tau)$ = Probability of (Miss | Target, Threshold $= \tau$)

$$P_{\text{FA}}(\tau) = \text{Probability of (FA} \mid \text{Nontarget, Threshold} = \tau)$$

Usually, the DCF is normalized by dividing it by a constant [77]. The probability values here can be computed using the distribution of true and impostor scores and computing the areas under the curve as shown in Figure 10 . The first three quantities above $\left(C_{Miss}, C_{FA}, \text{ and } P_{Target}\right)$ are predefined. Generally, the goal of the system designer is to find the optimal threshold value that minimizes the DCF.

41.2 Classical methods: from GMM to i-vector

41.2.1 Overview

41.2.2 A simple GMM inference model

Let's now consider a simple GMM that helps us identify the speaker identity of an utterance.

The GMM specifies a data generation model for a D-dimensional feature vector x. It specifies the density as the sum of Gaussian mixtures given by:

$$P(x|\lambda) = \sum_{k=1}^{M} w_k \times g\left(x|\mu_k, \Sigma_k\right)$$

where x is a D-dimensional feature vector $w_k, k = 1, 2, \ldots, M$ are the mixture weights that sum to 1, $\mu_k, \Sigma_k k = 1, 2, \ldots, M$ are the mean and the covariance of each Gaussian. Explicitly, $g\left(x|\mu_k, \Sigma_k\right)$ are the Gaussian densities given by

$$g\left(x|\mu_k, \Sigma_k\right) = \frac{1}{(2\pi)^{\frac{D}{2}} \left|\Sigma_k\right|^{\frac{1}{2}}} \exp^{-\frac{1}{2}\left(x-\mu_k\right)^T \Sigma_k^{-1}\left(x-\mu_k\right)}$$

The speaker identity recognition process has two stages: training stage and testing stage.

In the training stage, we collected utterances examples (number of examples can range from 10 to 100) of different speakers that we like to identify, and train a GMM for each speaker. More formally, let θ_i denote the model parameters $\theta = \left(w_k, \mu_k, \Sigma_k\right), k = 1, 2, 3, \ldots, M$ associated with the generative model of speaker i.

In the testing stage, given an utterance audio signal from a testing speaker, we extract acoustic features from overlapping frames $X = (x_1, ..., x_T)$ [1] and under independence assumption, compute the log-likelihood of each speaker model $i, i = 1, ..., K$, given by

$$\log P(X|\theta_i) = \frac{1}{T} \sum_{t=1}^{T} \log p(x_t|\theta_i).$$

The testing speaker is identified as speaker

$$j^* = \arg\max_i \log P(X|\theta_i).$$

1 Suppose that we have used voice activity detection (VAD) to remove silence frames

41.2.3 GMM-UBM

The most successful early speaker recognition system is the GMM-UBM system[1][Figure 41.2.1]. There are two components in the model, one is the universal background model (UBM), which is a high-order Gaussian Mixture Model (usually 512 to 2048 mixtures with 24 dimensions) trained on a large quantity of speech samples, from a wide population. UBM learns the speaker-independent distribution of features. Another component is individual Gaussian mixture models used to learn the feature of individual speakers. In the enrollment step, we train and adapt one GMM on the extracted features for each speaker. The speaker model is adapted from the UBM GMM using Maximum a Posteriori (MAP) Adaptation .

In MAP adaption, we simply start the expectation maximum (EM) algorithm [subsubsection 30.4.5.1] with the parameters learned by the UBM. Normally, in practice, we only adapt the mean, and not the covariance, since updating the covariance does not improve the performance.

The mean update based on MAP is given by

$$\mu_k^{MAP} = \alpha_k \mu_k + (1 - \alpha_k)\, \mu_k^{UBM}$$

where k is iteration number, $\alpha_k = \frac{n_k}{n_k + \tau_k}$ is the mean adaptation coefficient n_k is the count for the adaptation data, τ_k is the relevance factor, between 8 and 32.

The speaker verification decision can be achieved by the likelihood ratio. Let the speech sample with speaker S be Y. The verification task can be formulated as following hypothesis testing: $H_0 : Y$ is from speaker $SH_1 : Y$ is not from speaker S.

The decision to accept H_0 or not is the Likelihood Ratio (LR):

$$LR = \frac{p\left(Y|H_0\right)}{p\left(Y|H_1\right)}$$

where H_0 would use the speaker GMM and H_1 would use the UBM. If the Likelihood ratio is greater than the threshold θ, we accept H_0, otherwise we accept H_1.

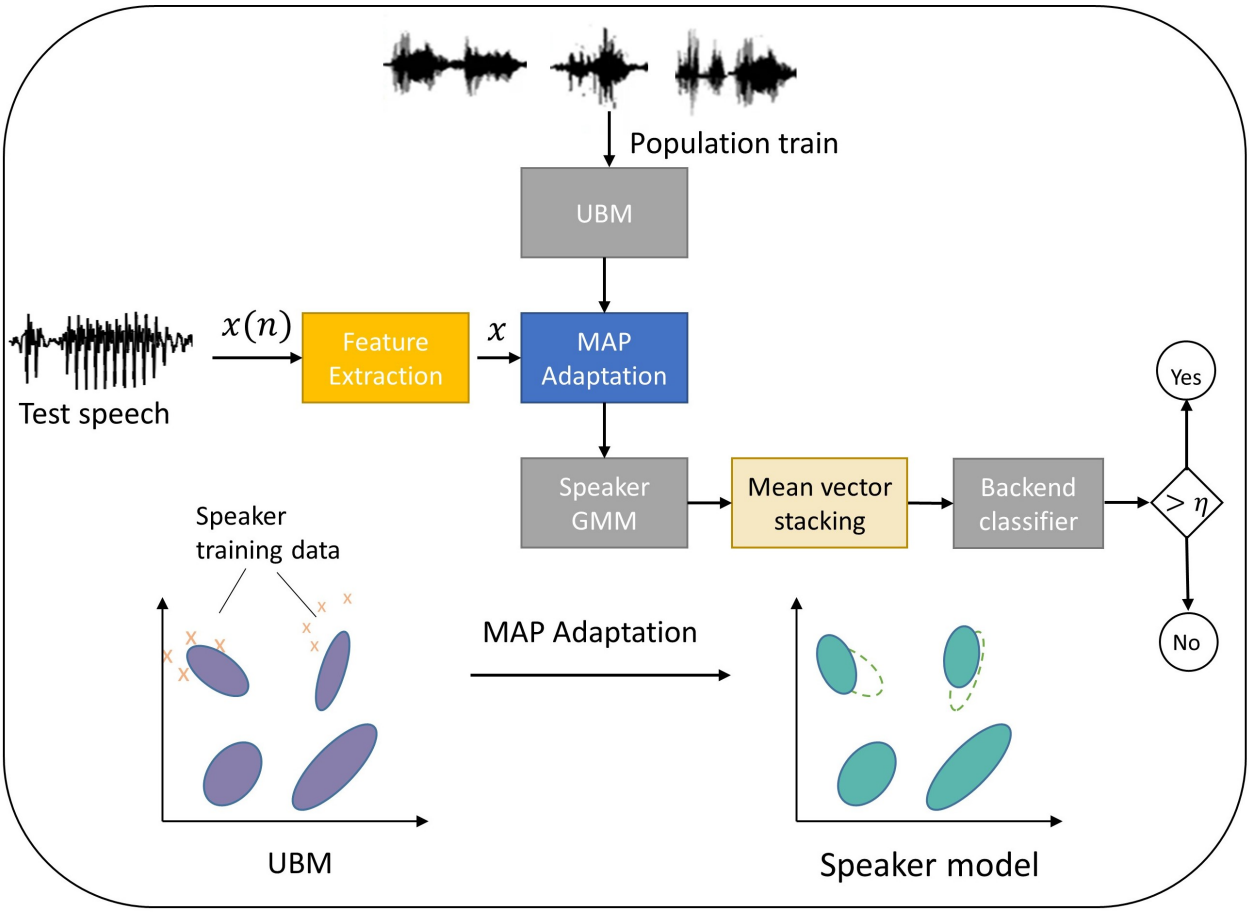


Figure 41.2.1: GMM-UBM scheme for speaker identification.

We typically use a diagonal covariance-matrix rather than a full covariance one since it is more computationally efficient and empirically works better.

41.2.4 GMM-SVM

Suppose in the MAP adaption process we only adapt mean vectors to individual speakers. Then we can construct a supervector s_a by stacking each mean vector in the GMM of speaker a. That is

$$s_a = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_M \end{bmatrix},$$

where $\mu_i \in \mathbb{R}^D$ is the Gaussian component mean vector, D is the feature dimensionality (e.g., $D = 13$ for MFCC) and M is the number of components (M ranges from 512 to 2948). The GMM supervector can be thought of as a mapping from an utterance to a high-dimensional feature vector.

These supervectors can be used as features to distinguish different speakers via more flexible machine learning methods. GMM-SVM [2] is one approach that applies support vector machine (SVM) to supervector and treat speaker recognition and verification as classification problems. Common kernels used to measure the similarity between two supervectors s_a and s_b include

- generalized linear kernel

$$\begin{aligned} K(s_a, s_b) &= \sum_{i=1}^{M} \lambda_i s_i^a \Sigma^{-1} s_i^b \\ &= \sum_{i=1}^{M} \left(\sqrt{\lambda_i}\Sigma^{-\frac{1}{2}} s_{a,i}\right)^t \left(\sqrt{\lambda_i}\Sigma^{-\frac{1}{2}} s_{b,i}\right) \end{aligned}$$

  where λ_i is weight for Gaussian mixture component.
- Gaussian kernel

$$K(s_a, s_b) = \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j MN\left(s_i^a - s_j^b; \mathbf{0}, \Sigma_i + \Sigma_j\right)$$

  A convenient approximation to above is to assume that means from different mixture components are far apart, which makes the terms $\lambda_i\lambda_j, i \neq j$ small. Then we have

$$K(s_a, s_b) \approx \sum_{i=1}^{N} \lambda_i MN\left(s_i^a - s_i^b; \mathbf{0}, \Sigma_i + \Sigma_i\right).$$

41.2.5 Joint Factor Analysis

The supervectors constructed from GMM-UBM across different speakers can contain variations beyond speaker variability, for example

- channel and session variability, which is caused by recording using different devices at different environment.
- additional noises.

The undesired variability causes mis-identification of same person in two different sessions as two different persons.Further, supervectors can have super-high dimensionality (e.g., upto 40, 000) that is computational prohibitive.

Joint factor analysis (JFA) [3–5] aims to accomplish two goals. First, by decomposing a supervector into different components (speaker component, channel component, etc), we can isolate out the speaker component for further analysis. Second, JFA also achieves dimensionality reduction and alleviate the computational burden.

Specifically, consider the supervector $s \in \mathbb{R}^{F\times C}$ where F is the feature dimension, and C is number of mixtures in the UBM. We have decomposition in the joint factor analysis given by

$$s = m + Vy + Ux + Dz$$

where V, U, D are low rank factor loading matrices for speaker-dependent, channel-dependent, and residual subspaces, respectively, and m is the constant mean component, y is the speaker factor assumed to following standard normal distribution components. Similarly, x, z are factors following standard normal distribution. The columns in V are also called **eigenvoices**. Within JFA framework, a typical low-dimensional speaker y (usually at 300 to 599) can be extracted. Different backend analysis, like SVM, cosine similarity, or PLDA [**??**], can be carried out on the vector y for speaker verification tasks.

Note that this decomposition also implicitly assume the independence between speaker factor and channel factor and x would not contain any speaker related information. Later we will show that this assumption is flawed in empirical studies and could be remedied by i-vector method[6].

41.2.6 i-vector method

Empirical studies [7] show that channel vector x in joint factor analysis (JFA) method, which is supposed to model only channel effects, also contains information to distinguish speakers. [6] proposed the i-vector method, whose key idea is to combine the channel and speaker spaces into a total variability space, but reduce intersession variability by using backend scoring procedures, such as linear discriminant analysis and within-class covariance normalization, and other scoring methods [**??**]. i-vector system has been the dominating speaker verification technology before the advent of deep learning methods.

Let s be a supervector consisting of stacked mean vectors from a GMM speaker model, m be the GMM-UBM supervector. In the i-vector method [Figure 41.2.2], we assume the factorization

$$s = m + Tw,$$

where T is the total variability loading matrix (with number of columns around 100), and w is a low-dimensional total (variability) factors, m is usually the Universal Background Model mean supervector, the vector irrelevant to speaker and channel variability. w known as the **i-vector**, standing for the identity vector

The i-vector w is a hidden variable and requires factor analysis to estimate. T and w can be estimated through expectation-maximization algorithm [3]. The estimated w can be used as a feature vector and fed into backend classifiers (e.g., SVM) for classification.

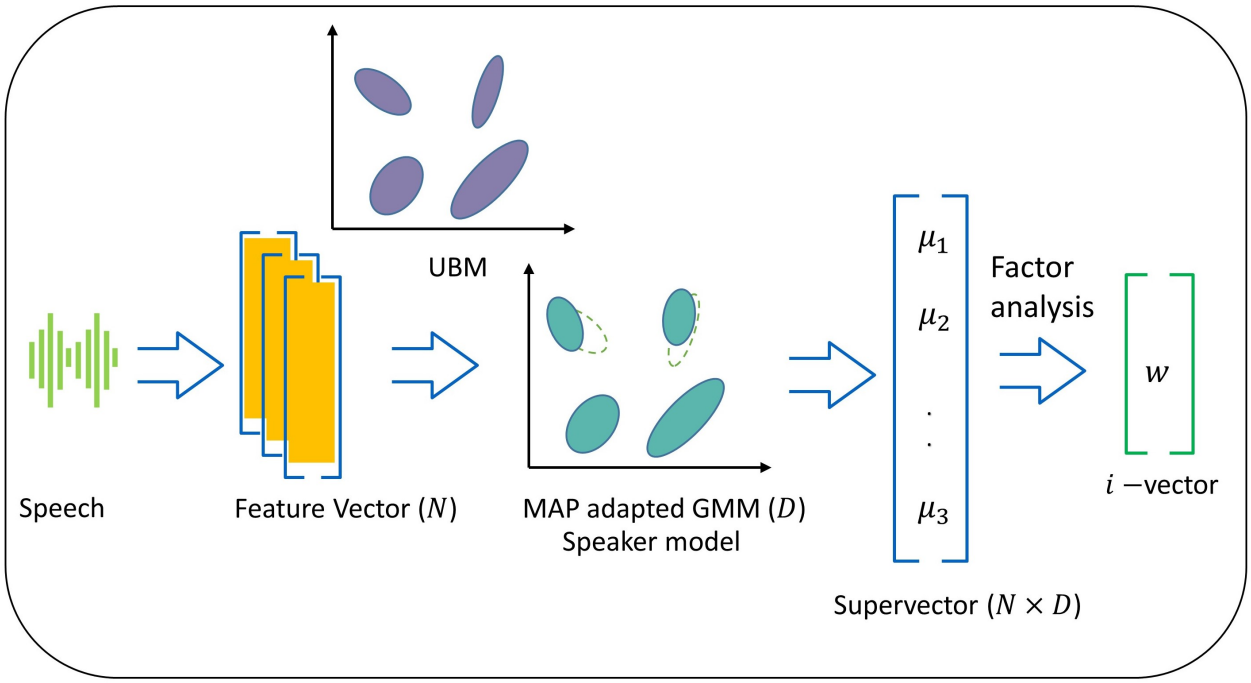


Figure 41.2.2: i-vector computational flow.

41.3 Deep learning for Speaker Recognition

41.3.1 Introduction

Speaker recognition application is further categorized into text-dependent and text-independent. In text-dependent speaker recognition,the utterance content is prescribed. Examples include wakewords likes *Hey Siri*, *Hey Google*, or *Alexa* for voice assistant devices. On the other hand, in text-independent speaker recognition, utterance content is not constrained. As free choice of speaking contents introduces additional variability in feature representation, text-independent speaker recognition is usually more challenging than text-dependent one.

A typical speaker verification system operates at two phases [Figure 41.3.1]: **enrollment phase** and **verification phase**. In the enrollment phase, we use embedding extraction model to convert the enrollment speech of each speaker to speaker embeddings, as some sort of voiceprints In the verification phase, the embedding extraction model extract the embedding from a speech and compared against a voiceprint of the claimed identity. Based on the similarity score, the speaker verification system accepts or rejects the person.

A speaker embedding is a dense vector representation of an utterance speech. Speaker embeddings extraction models are trained via metric learning framework such that embeddings of same speakers' utterances are closer and embeddings from different speakers' utterances are far apart.

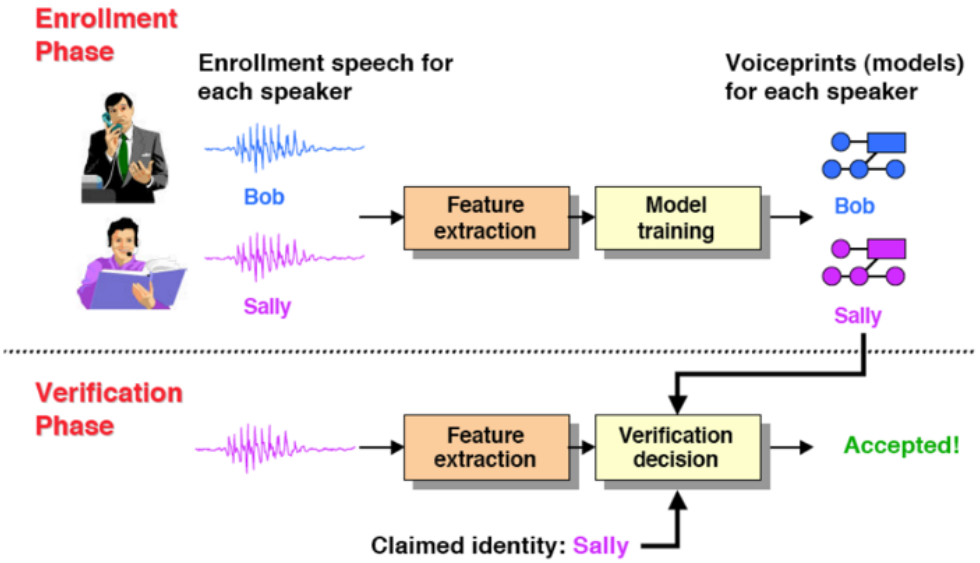


Figure 41.3.1: Two phases (enrollment and verification) of using deep learning methods for speaker verification.

In the metric learning framework, there are two major paradigms to train a deep embedding extractor [Figure 41.3.2]. Both paradiagms aim to obtain an embedding space where similar items are close together and dissimilar items are far apart, as measured by some distance metric. The embedding space could be a high-dimension Euclidean space where we can measure object distance via Euclidean distance; or the embedding space is on a hyper-spherical surface where we can measure object distance via cosine similarity.

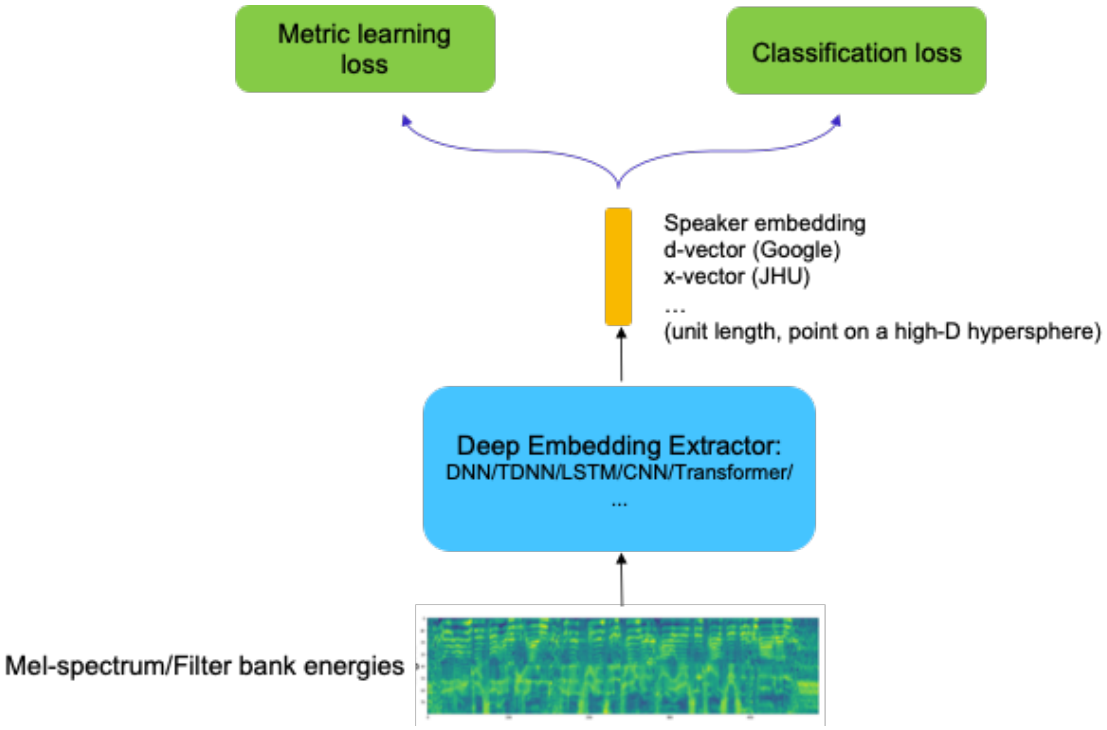


Figure 41.3.2: Mel-spectrums or filter bank energies of a speech is fed into the neural network as the embedding extractor. The embedding is refined by minimizing either classification loss or metric learning loss, which corresponding to two different paradigms. The speaker embedding extracted are named d-vector by Google or x vector by Hopkins

In the learning paradigm that involves metric loss, loss functions include Siamese binary contrastive loss, triplet loss, generalized end-to-end loss, etc. The learning process is a direct optimization on the embeddings on the embedding space. The learning mechanism is to pull utterance embeddings from same speakers closer and push utterance embeddings from different speakers apart.

In the learning paradigm that involves classification loss, loss functions include the typical softmax loss and its variants (e.g., Angular-margin Softmax).Usually, the number of classes or speakers involved in the classification is more than $10k$. The mechanism is to learn discriminative embeddings by predicting a class/speaker in a large scale classification task. Compared to metric loss type of learning, it is an indirect optimization on the embeddings on the embedding space. One drawback of using classification loss is that the last Softmax loss type can be rather memory consuming for large scale classification.

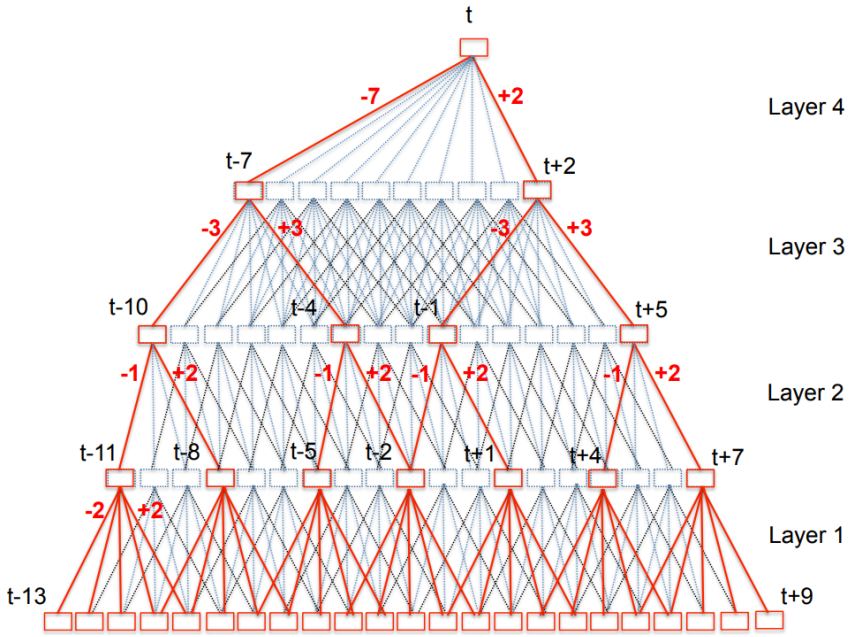


Figure 41.3.3: Computation in TDNN with sub-sampling (red) and without sub-sampling (blue+red)

41.3.2 Architecture for embedding extractors

41.3.2.1 *Time-delayed neural network and x-vector*

The x-vector speaker verification system [8] utilize time-delayed neural network [9] to perform acoustic feature extraction [Figure 41.3.3]. The salient feature of TNDD compared to fully-connected neural network is that affine transformation from one layer to the next layer is not applied globally to all frames but smaller windows with local context. The extracted framewise features are aggregated to pooled features, which are further fed into a feed-forward network to get utterance level embeddings.

As shown below, x-vectors are extracted at layer segment6, before the nonlinearity. The network is trained to classify the N speakers in the training data.

Layer	Layer context	Total context	Input x output
frame 1	$[t-2, t+2]$	5	120×512
frame 2	$\{t-2, t, t+2\}$	9	1536×512
frame3	$\{t-3, t, t+3\}$	15	1536×512
frame4	$\{t\}$	15	512×512
frame5	$\{t\}$	15	512×1500
stats pooling	$[0, T)$	T	$1500T \times 3000$
segment6	$\{0\}$	T	3000×512
segment7	$\{0\}$	T	512×512
softmax	$\{0\}$	T	$512 \times N$

41.3.2.2 *LSTM and d-vector*

In [10], audio signals are first transformed into frames of width 25ms and step 10ms. Then we extract 40-dimension log-mel-filterbank energies features for each frame (frames of width 25ms and step 10ms). For both text-independent and text-dependent speaker verification systems, 3-layer LSTM with projection is used[11, 12]. And the embedding vector (d-vector) size is the same as the LSTM projection size. For the text-dependent speaker verification system, LSTM has 128 hidden nodes with projection size 64. For the text-independent speaker verification system, LSTM has 768 hidden nodes with projection size 256.

Using LSTM as feature extractor might have computational disadvantage because of its sequential computation order prohibiting parallel computing. Yet LSTM is more suitable for real-time speaker verification application. For example, LSTM can emit embeddings and their scores based on audio received so far. No need to wait for the completion of the whose utterance.

For text-dependent application, to facilitate the attention module to learn the context-dependent weight vector w_t, one can augment the framewise extracted feature vector $x_t \in \mathbb{R}^D$ with additional positional information. A simple strategy is

$$x'_t = \begin{bmatrix} x \\ \frac{t}{T} - \frac{1}{2} \end{bmatrix} \in \mathbb{R}^{D+1},$$

where T is the length in the time axis.

41.3.2.3 *CNN*

By treating Mel-spectrum as a two-dimensional image (one is the temporal axis and one is the frequency axis), we can also use CNN as the embedding extractor. Recent studies on Voxceleb and SRE datasets show that ResNet [13, 14] usually out-performs vanilla CNN in terms of final speaker verification error rate.

Layer name	–	Output shape $T \times F \times C$
Input	–	$T \times 80 \times 1$
Conv2D	3×3, stride $= 1$	$T \times 80 \times 128$
ResBlock-1	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, stride $= 1$	$T \times 80 \times 128$
ResBlock-2a	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 128 \end{bmatrix} \times 1$, stride $= 2$	$T/2 \times 40 \times 128$
ResBlock-2b	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, stride $= 1$	$T/2 \times 40 \times 128$
ResBlock-3a	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 1$, stride $= 2$	$T/4 \times 20 \times 256$
ResBlock-3b	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 5$, stride $= 1$	$T/4 \times 20 \times 256$
ResBlock-4a	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 1$, stride $= 2$	$T/8 \times 10 \times 256$
ResBlock-4b	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$, stride $= 1$	$T/8 \times 10 \times 256$
Flatten (C, F)	–	$T/8 \times 2560$
Statistical pooling	–	5120
Linear embedding	–	D

41.3.3 Pooling strategies

The above neural networks are usually used to obtain framewise feature representations of the input audio. To get the embeddings for each utterance, we need to aggregate the framewise features to utterance level features.

The aggregated result will ultimately pass through a linear embedding dense layer to get the final embedding vector.

Statistical pooling Statistical pooling extracts statistical quantities (e.g., mean and variance) as utterance level features by reducing through the time axis[8, 13]. More specifically, given $x \in \mathbb{R}^{T \times p}$ produced by the framewise feature extractor, a vanilla statistical pooling is given by

$$\mu_j = \frac{1}{T} \sum_{i=1}^{T} x_{ij}$$
$$\sigma_j = \sqrt{\frac{1}{T} \sum_{i=1}^{T} (x_{ij} - \mu_j)^2}$$

and the concatenated output $y = (\mu, \sigma) \in \mathbb{R}^{2p}$ is used as the input to the linear embedding layer to produce the utterance embedding.

Self-attentive pooling Self-attentive pooling calculates the utterance-level embedding via a weighted sum of framewise features[15]. Given framewise features $\{x_1, x_2, \cdots, x_L\}, x_i \in \mathbb{R}^D$, we first feed them into a multi-layer perceptron (MLP) to get $\{h_1, h_2, \cdots, h_L\}, h_i \in \mathbb{R}^H$ as their hidden representations. For example, in a one-layer perceptron, we have

$$h_t = \tanh(Wx_t + b)$$

where $W \in \mathbb{R}^{H \times D}, b \in \mathbb{R}^D$.

The weight w_t of each frame is related to the similarity of h_t with a learnable context vector $u \in \mathbb{R}^H$ via.

$$w_t = \frac{\exp\left(h_t^T u\right)}{\sum_{\tau=1}^{T} \exp\left(h_\tau^T u\right)}.$$

The context vector u is randomly initialized and jointly learned during the training process. From the key-value-query perspective, it can be interpreted as a constnat query that aims to place more weight on more informative frames. Finally, the utterance level representation e is given by

$$e = \sum_{t=1}^{T} w_t x_t.$$

Attentive statistical pooling

Attentive statistical pooling aims to enhance statistical pooling by assigning more weights/attention to specific frames, since it is often the case that some frames are more unique and important for discriminating speakers than others in a given utterance [16].

The attention module first calculates a scalar score e_t for each frame-level feature

$$h_t = f\left(Wh_t + b\right) + k$$

where $f(\cdot)$ is a non-linear activation function (e.g., tanh or ReLU function). Then we measure the importance of each frame as the similarity of h_t with a learnable context vector $u \in \mathbb{R}^H$ and get a normalized importance weight w_t through a Softmax function.

$$w_t = \frac{\exp\left(h_t^T u\right)}{\sum_{\tau=1}^{T} \exp\left(h_\tau^T u\right)}$$

where $\sum_{t=1}^{L} w_t = 1$.

The normalized score w_t is then used as the weight in the pooling layer to calculate the weighted mean vector and weighted standard deviation,

$$\tilde{\mu} = \sum_{t}^{T} w_t h_t,$$

$$\tilde{\sigma} = \sqrt{\sum_{t}^{T} w_t h_t \odot h_t - \tilde{\mu} \odot \tilde{\mu}}$$

Finally, the weighted mean $\tilde{\mu}$ and weighted standard deviation $\tilde{\sigma}$ are used input to the linear embedding layer to produce the utterance embedding.

41.3.4 Contrastive learning

41.3.4.1 *Pairwise contrastive loss*

In this training setup, we construct positive and negative training pairs. Positive pairs are composed by utterances from the same speaker, denoted by an anchor sample x_A and a positive sample x_P. Negative pairs are composed by utterances from two different speakers, denoted by an anchor sample x_A and a negative sample x_N. The objective is to learn utterance representations that encourages a small distance between positive paris and a distance larger than margin m for negative pairs [Figure 41.3.4]. Let x_A, x_P and x_N be the sample representations and d be a distance function, we can write the loss function to minimize to be

$$L = \begin{cases} d\left(x_A, x_P\right) & \text{if} \quad \text{PositivePair} \\ \max\left(0, m - d\left(x_A, x_N\right)\right) & \text{if} \quad \text{NegativePair} \end{cases}$$

For positive pairs, the loss is given by the distance; For negative pairs, the loss is non-zero when the distance is smaller than the margin m.

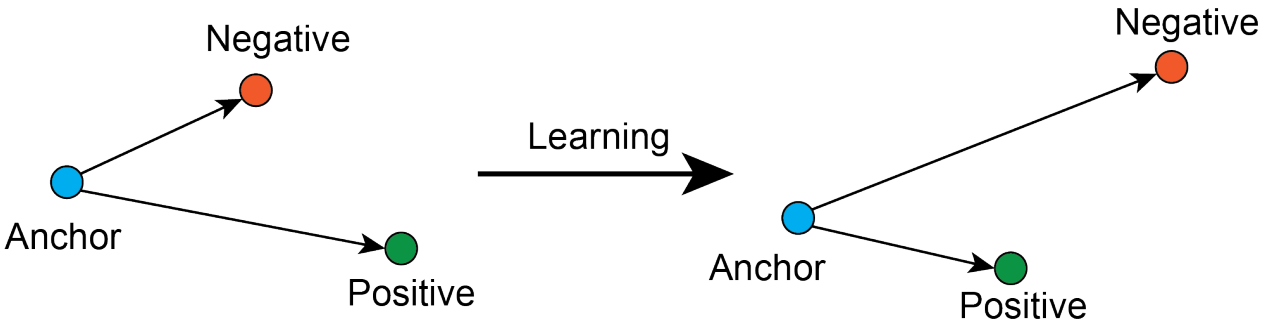


Figure 41.3.4: Pairwise contrastive loss encourages the learning of representations that positive pairs are close and negative pairs are far apart.

We can also define similarity measures between a pair of features x_i and x_j using cosine similarity $\cos(x_i, x_j)$ and construct the logit as

$$l_{ij} = w \cos(x_i, x_j) + b,$$

where $w > 0, b \in \mathbb{R}$ are learnable scalars.

Then we can optimize the embeddings x_i and x_j via a binary cross entropy in a minibatch given by

$$\begin{aligned} L &= -\frac{1}{N_{pos}} \log \sigma(l_{ij}) - \frac{1}{N_{neg}} \log(1 - \sigma(l_{ij})) \\ &= \frac{1}{N_{pos}} \log(1 + e^{-l_{ij}})) + \frac{1}{N_{neg}} \log(1 + e^{(}l_{ij})) \end{aligned}$$

where N_{pos} and N_{neg} are positive and negative pairs in a minibatch, and $\sigma(\cdot)$ is the Sigmoid function.

Via minimizing the binary cross entropy, embeddings of positive pairs are brought closer to increase l_{ij} and embeddings of negative pairs are brought apart to minimize l_{ij}.

41.3.4.2 *Prototypical loss and angular prototypical loss*

When we train the embedding extractor network via prototypical loss [17], we generate a mini-batch which contains a support set S and a query set Q. For simplicity, we denote the M-th utterance from every speaker as the query and the first $M-1$ utterances

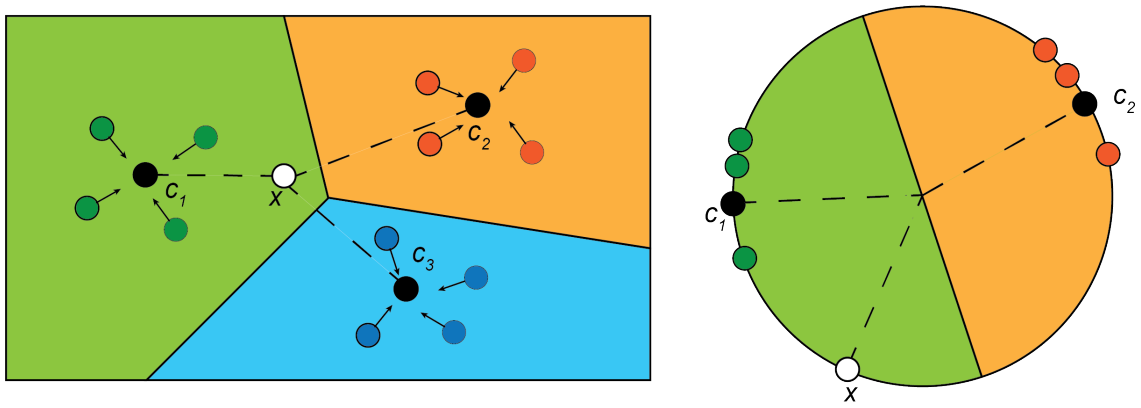


Figure 41.3.5: Prototypical loss (left) and angular prototypical loss. Utterances in the support set are used to compute class centroids. Losses are measured between class centroids and query examples.

from every speaker as the support set. Then the prototype (or centroid) for each speaker j is constructed from the associated support set, given by:

$$c_j = \frac{1}{M-1} \sum_{m=1}^{M-1} x_{j,m}.$$

For each query $x_{j,M}$, its positive example pair is c_j and its negative examples pairs are all c_k, where $k \neq j$. Typically, the value of M is typically chosen to match the expected situation at test-time, e.g. $M = 5 + 1$, such that the prototype is composed of five different utterances.

We then use squared Euclidean distance to measure distances between positive and negative example pairs, given by:

$$S_{j,k} = \left\| x_{j,M} - c_k \right\|_2^2$$

The angular prototypical loss constructs the same positive and negative examples as the prototypical loss but computes distance based on angular cosine distance. Specifically, we use a cosine-based similarity metric with learnable scale and bias

$$S_{j,k} = w \cdot \cos\left(x_{j,M}, c_k\right) + b.$$

Compared to Euclidean distance, using the angular distance function achieves scale invariance, thus likely improving the robustness to scale and demonstrating more stable convergence.

During training, each query example is classified against N speakers based on a Softmax over distances to each speaker prototype:

$$L_{\mathrm{P}} = -\frac{1}{N}\sum_{j=1}^{N}\log\frac{e^{S_{j,j}}}{\sum_{k=1}^{N}e^{S_{j,k}}}$$

By minimizing the loss function, the query embedding and the centroid embedding of the same class are brought closer to maximize $S_{j,j}$; the query embedding and the centroid embedding of different classes are brought apart to minimize $S_{j,k}$.

It is shown empirically that prototypical loss led to better performance compared to Triplet loss[18].

41.3.4.3 *Generalized end-to-end (GE2E)*

Generalized end-to-end (GE2E) [10] employs a slightly different strategy to define centroids for each speaker and compute similarility between positive and negative examples.

Suppose in a minibatch each speaker has M utterances. We define two versions of centroids, given by

$$c_j = \frac{1}{M}\sum_{m=1}^{M}x_{j,m}$$

$$c_j^{(-i)} = \frac{1}{M-1}\sum_{\substack{m=1\\m\neq i}}^{M}x_{j,m}$$

Note that for each j, there are M different way to construct $c_j^{(-i)}$.

The scaled cosine similarity between the embeddings and all centroids can be summarized by

$$S_{j,i,k} = \begin{cases} w\cdot\cos\left(x_{j,i}, c_j^{(-i)}\right) + b & \text{if} \quad k = j \\ w\cdot\cos\left(x_{j,i}, c_k\right) + b & \text{otherwise} \end{cases}$$

where j and k is over all speakers in the minibatch, i is over all utterances belonging to a speaker, $w > 0$ and b are learnable scale and bias. The final GE2E loss is defined as:

$$L_{\mathrm{G}} = -\frac{1}{N}\sum_{j,i}\log\frac{e^{\mathbf{S}_{j,i,j}}}{\sum_{k=1}^{N}e^{\mathbf{S}_{j,i,k}}}$$

Note that in angular prototypical, only the first $M-1$ examples are used to construct centroids, while in generalized E2E, all combinations of size of $M-1$ are used to construct centroids. Therefore, generalized E2E utilizes each example in a minibatch in a more economical way.

41.3.5 Metric learning via classification

41.3.5.1 *Motivations*

For open-set speaker verification, we desire to extract discriminative voice characteristics that maximize inter-class distance and minimize intra-class distance. Performing large scale (>1,000) speaker classification task simultaneously offer advantages in learning discriminative features.

First, predicting a training sample into one of many classes is a considerable challenging task that can make full use of the large learning capacity of neural networks to extract effective features. Second, by predicting a speaker from thousands speakers helps the network to learn shared representations that can classify all the identities well. Therefore, the learned high-level features have good generalization ability to unseen examples.

In the following, we will go over different classification loss function choices, starting from the basic Softmax loss to the state-of-the-art CosFace loss that encourages the learning of large margin features.

41.3.5.2 *Softmax loss*

[2]

Suppose there are N utterances in a minibatch, whose embeddings and labels are $(x_1, y_1), ..., (x_N, y_N)$. Softmax loss is implemented via a Softmax loss layer, which is the combination of the last fully connected layer without activation and the Softmax function.

Definition 41.3.1 (Softmax loss layer). *A Softmax loss layer consists of a linear layer with the Softmax function, a multi-class cross-entropy loss. It is formulated as:*

$$L_S = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{C} e^{W_j^T x_i + b_j}}$$

where $x_i \in \mathbb{R}^D$ is the embedding representation for utterance i, $y_i \in \{1, 2, ..., C\}$ is the class label for utterance i, $W = [W_1, ..., W_C], W_i \in \mathbb{R}^{C\times D}$ and $b \in \mathbb{R}^C$ are the weights and bias of the linear layer, respectively.

2 Here we define softmax loss layer as the combination of the last fully connected layer and the softmax function.

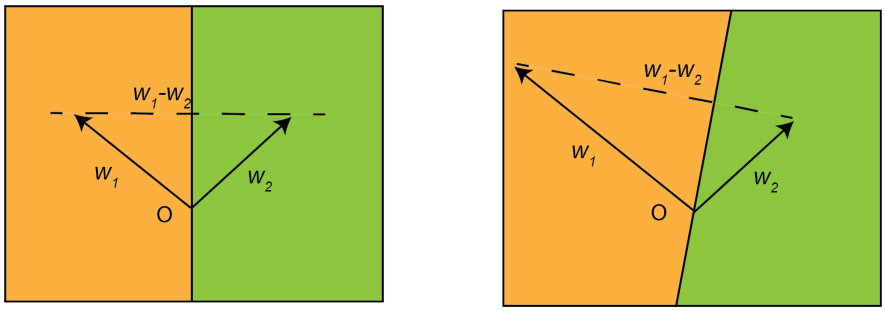


Figure 41.3.6: Decision boundaries in a binary classification problem using Softmax. For simplicity, we assume bias terms are zero. Decision boundaries are perpendicular to $w_1 - w_2$. When x falls into orange region, it belongs to class 1; when x falls into green region, it belongs to class 2.

Example 41.3.1 (softmax for binary classification). In a binary class case, the posterior probabilities obtained by Softmax loss layer are

$$p_1 = \frac{\exp\left(W_1^T x + b_1\right)}{\exp\left(W_1^T x + b_1\right) + \exp\left(W_2^T x + b_2\right)}$$
$$p_2 = \frac{\exp\left(W_2^T x + b_2\right)}{\exp\left(W_1^T x + b_1\right) + \exp\left(W_2^T x + b_2\right)}$$

where x is the learned feature vector, W_i and b_i are weights and bias of fully connected layer corresponding to class 1 and 2. The binary classification decision is obtained by requiring $p_1 = p_2$, which gives a **linear** decision boundary [Figure 41.3.6]

$$(W_1 - W_2)x + (b_1 - b_2) = 0.$$

Therefore, in an end-to-end neural network training framework, the Softmax loss encourages the learning of linearly separable features.

Now we explain how Softmax loss helps training embedding extractors to produce embeddings that aims to maximize the intra-speaker distance and minimize inter-speaker distance.

The weight vector W_j associated with speaker j is shared with all embeddings x from speaker j. Intuitively, Softmax loss is minimized when all embeddings x align with W_j such that the numerator term $W_j x$ produces larger values. Separately, Softmax loss is minimized when all embeddings x of speaker j deviate from $W_k, k \neq j$ such that the denominator term $W_k x$ produces smaller values. Therefore, even when weight vectors are fixed, training embedding extractors with Softmax loss would result in aggregations of embeddings of same speakers and separation of embeddings of different speakers. Besides, the training process also involves the adaption of weight vectors to minimize

Softmax loss, that is, W_j moves towards of embeddings associated with speaker j and moves away from $W_k, k \neq j$.

The embeddings trained from Softmax loss are often combined with additional scoring backends, such as PLDA to generate scoring functions.

41.3.5.3 *Normalized Softmax loss*

Softmax loss can also be written by

$$L_S = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\left\|W_{y_i}\right\|\|x_i\|\cos(\theta_{y_i,i})+b_{y_i}}}{\sum_{j=1}^{C}e^{\left\|W_j\right\|\|x_i\|\cos(\theta_{j,i}+b_j}}$$

where $\theta_{i,j}$ is the angle between vector W_j and x_i and

$$\cos(\theta_{y_i,i}) = \frac{W_j}{\left\|W_j\right\|}\cdot\frac{x_i}{\|x_i\|}.$$

Therefore, if we enforce weight vectors W_j in the last Softmax loss layer and x_i to have unit length, i.e., $\|W_i\| = \|x_i\| = 1$ and discard bias terms, then we have the **normalized Softmax loss** given by the following.

Definition 41.3.2 (normalized Softmax loss).

$$L_{NS} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\cos\left(\theta_{y_i,i}\right)}}{\sum_j e^{\cos\left(\theta_{j,i}\right)}}$$

where $\cos\left(\theta_{j,i}\right)$ *is the dot product of* ***normalized*** *vector* W_j *and normalized* x_i.

Normalization on embedding vectors and weight vectors has several impact on training. First, it enforces scale invariance on embeddings and weight vectors such that embeddings and weight vectors can be viewed as points on a high-dimensional hypersphere. Second, to maximize inter-speaker distance and minimize intra-speaker variance, the neural network is forced to optimize on the angles between vectors. Third, in the testing stage, the speaker verification score of a testing pair can be simplified to cosine similarity between the two embedding vectors. This improves the simplicity and scalability of the scoring step.

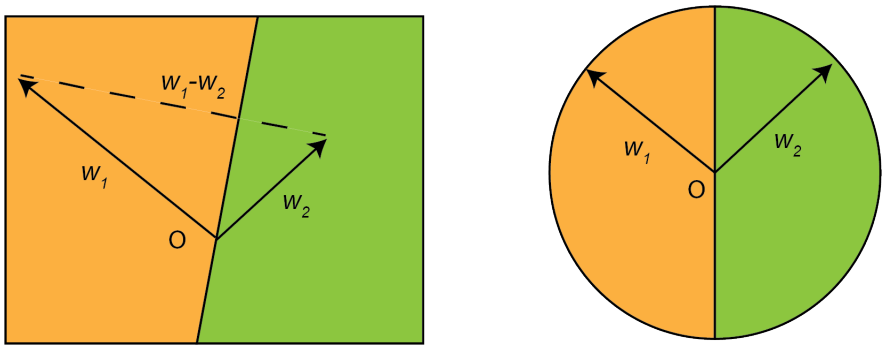


Figure 41.3.7: Visualization of 2D decision boundary resulted from Softmax loss and normalized Softmax loss for binary classification. When x falls into orange region, it belongs to class 1; when x falls into green region, it belongs to class 2.

To prevent gradients getting too small in the training process, one can also use a globally-scaled normalized Softmax loss, which is given by

$$L_{NS} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos\left(\theta_{y_i,i}\right)}}{\sum_j e^{s\cos\left(\theta_{j,i}\right)}},$$

where $s \in \mathbb{R}$ is a learnable scalar (typically $s \approx 30$). Geometrically, multiplying s is equivalent to mapping unit embedding vectors onto a hypersphere with larger radius $s > 1$. Also note that the scalar s would not affect the scoring results in the testing stage since ROC curve and EER are invariant to affine transformations of scores.

Now we consider a binary classification example to understand the mechanism of normalized Softmax loss in partitioning the embedding space. For an embedding vector x, the decision boundary becomes

$$\cos(\theta_1) - \cos(\theta_2) = 0,$$

where θ_1 is the angle between x and W_1 and θ_2 is the angle between x and W_2.

Particularly, when $\cos(\theta_1) > \cos(\theta_2)$ or $\theta_1 < \theta_2$, it belongs to class 1; when $\cos(\theta_1) < \cos(\theta_2)$ or $\theta_1 > \theta_2$, it belongs to class 2. The resulting decision boundary in 2D and 3D embedding space for binary classification can be visualized by **??**.

41.3.5.4 *AM-Softmax (CosFace)*

Similar to the motivation of support vector machine (SVM), we also like to incorporate margins with decision boundaries to enhance the discrimination power, which is

however not directly available in Softmax loss or normalized Softmax loss. This is because Softmax loss only penalizes classification error and not explicitly promote margins. Following normalized Softmax loss where we use angles to compute distance and set decision boundaries, we like to incorporate angular margin in the loss function.

To this end, additive margin Softmax (AM-Softmax) loss functions was first proposed in face recognition [19] (known as CosFace) and then successfully applied to speaker recognition[13, 20, 21].

The AM-Softmax is given by the following definition.

Definition 41.3.3 (AM-Softmax).

$$L_{\text{AM}} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\left(\cos\left(\theta_{y_i,i}\right)-m\right)}}{e^{s\left(\cos\left(\theta_{y_i,i}\right)-m\right)}+\sum_{j\neq y_i}e^{s\left(\cos\left(\theta_{j,i}\right)\right)}}$$

where $\cos\left(\theta_{j,i}\right)$ *is the dot product of* ***normalized*** *vector* W_j *and normalized* x_i*,* m *is a scalar margin (typically* $m = 0.1 \sim 0.3$*) and* s *is a fixed scalar factor.*

Again we consider a binary classification example to understand the mechanism of AM-Softmax loss in enhancing discriminative power of embedding extractors [Figure 41.3.8]. For an embedding vector x, the decision boundary becomes

- If $\cos(\theta_1) - m > \cos(\theta_2)$, x belongs to class 1;
- If $\cos(\theta_2) - m > \cos(\theta_1)$, x belongs to class 2;

where θ_1 is the angle between x and W_1 and θ_2 is the angle between x and W_2. Note that there are two decision boundaries and the margin between the boundaries is given by $2m$ in terms of cosine values. Suppose x belongs to class 1, x has to get closer to W_1 other it would get heavier penalty from AM-Softmax than what it would get from normalized Softmax.

41.3.5.5 *Other variants*

So far we have discussed the Softmax loss and its popular variants in speaker verification community. Other Softmax loss variants include **SphereFace**[22] and **ArcFace**[23], which are all invented by the face recognition community.

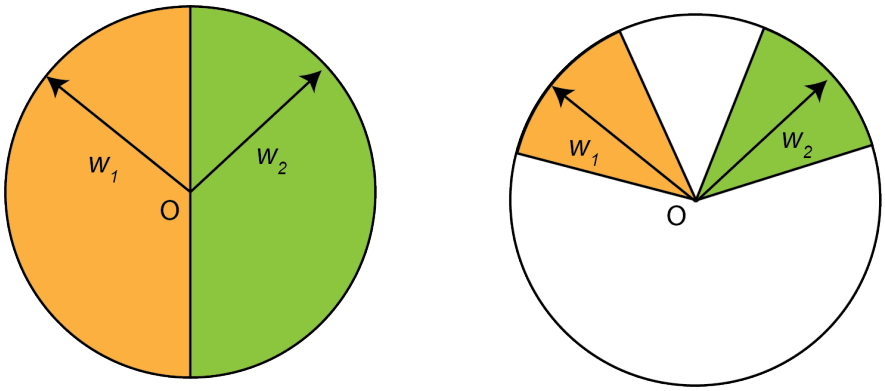


Figure 41.3.8: Visualization of 2D decision boundary resulted from normalized Softmax loss and AM-Softmax loss for binary classification. When x falls into orange region, it belongs to class 1; when x falls into green region, it belongs to class 2.

The loss function of SphereFace is given by

$$L_{\text{SF}} = \frac{1}{N}\sum_i -\log\left(\frac{e^{s\cos\left(m\theta_{y_i,i}\right)}}{e^{s\cos\left(m\theta_{y_i,i}\right)} + \sum_{j\neq y_i} e^{s\cos\left(\theta_{j,i}\right)}}\right)$$

where $m \geq 2$ is an integer greater than 1.

Intuitively, when $m > 1$, $\theta_{y_i,i}$ must remain smaller to minimize loss compared to when $m = 1$. In the binary classification example, the decision boundaries for x would be like

- $\cos(m\theta_1) > \cos(\theta_2)$ or $\theta_1 < \frac{\theta_2}{m}$ for x to be classified as class 1;
- $\cos(m\theta_2) > \cos(\theta_1)$ or $\theta_2 < \frac{\theta_1}{m}$ for x to be classified as class 2;

where θ_1 is the angle between x and W_1 and θ_2 is the angle between x and W_2. Note that there are two decision boundaries and the margin between the boundaries is given by $(1 - \frac{2}{m+1})\theta_{12}$ in terms of cosine values, where θ_{12} is the angle between W_1 and W_2.[3]

ArcFace has loss function given by

$$L_{\text{AF}} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\left(\cos\left(\theta_{y_i,i}+m\right)\right)}}{e^{s\left(\cos\left(\theta_{y_i,i}+m\right)\right)} + \sum_{j=1,j\neq y_i}^{n} e^{s\cos\theta_{j,i}}}.$$

ArcFace is a variant closely related to CosFace, it has a better geometric interpretation since the margin is added to the angle and not to the cosine.

3 If we evenly devided the arc from W_1 to W_2 into $m+1$ pieces. Then the distance or margin between two decision boundaries are $\frac{m-1}{m+1}\theta_{12}$.

41.4 Speaker Diarization

41.4.1 Introduction

41.4.1.1 *What is speaker diarization?*

Speaker recognition and verification address the question of *who is speaking* in an utterance that contains one speaker. In daily life speaking scenarios, multiple speakers appearing in an recorded audio is not uncommon. Speaker diarization aims to address the question of *who spoke when*. More formally, given an audio recording consisting multiple speakers, speaker diarization is the process of partitioning it into homogeneous segments and associate segments with speakers identities.

As many speech processing technologies, such as in automatic speech recognition or speaker recognition only applies to one speaker scenario, speaker diarization is an critical front end module in multi-speaker applications. It is widely anticipated that there is great potential in speaker diarization technologies in the broad areas, including but not limit to

- Transcription for medical conversion between doctors and patients.
- Automatic legal proceedings in courts.
- Video indexing based on speaker diarization results.

41.4.1.2 *Diarization evaluation*

One standard metric for evaluating and comparing speaker diarization systems is **Diarization Error Rate (DER)**, which the ratio of incorrect segmentation over the total duration. Suppose there is *no overlapping speakers* (i.e., at most one speaker speaks at any time), we have DER definition given by the following.

Definition 41.4.1 (diarization error rate). *Consider an utterance without overlapping speakers, the **Diarization Error Rate (DER)** is as follows [Figure 41.4.2]:*

$$\text{DER} = \frac{FA \ + \ MS \ + \ SPE}{Total}$$

where

- ***False Alarm (FA)** is the duration of non-speech incorrectly classified as speech,*
- ***Missed Speaker Error (MS)** is the duration of speech incorrectly classified as non-speech,*
- ***Speaker Error** (SPE) is the duration of speaker confusion,*

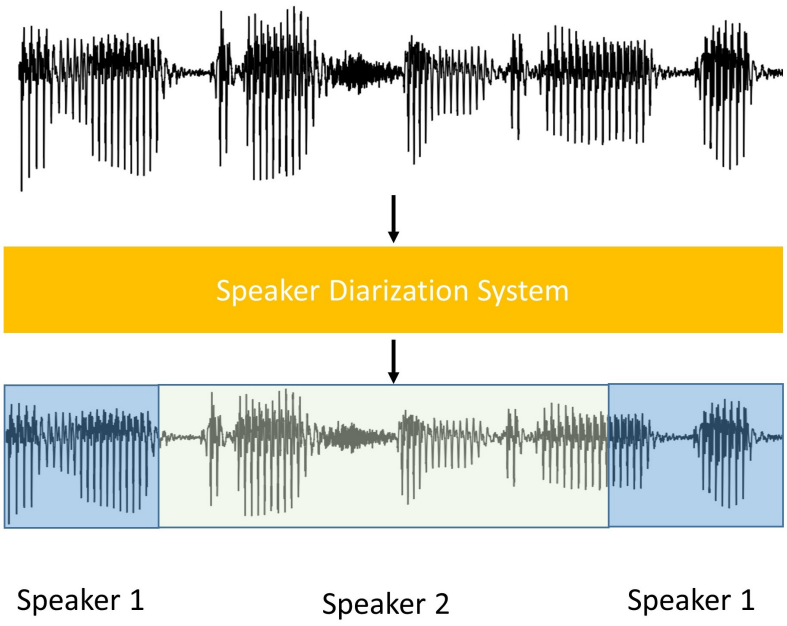


Figure 41.4.1: A speaker diarization system aims to partition an audio into homogeneous segments corresponding to different speakers.

- ***Total*** *is the total duration of speech of the utterance.*

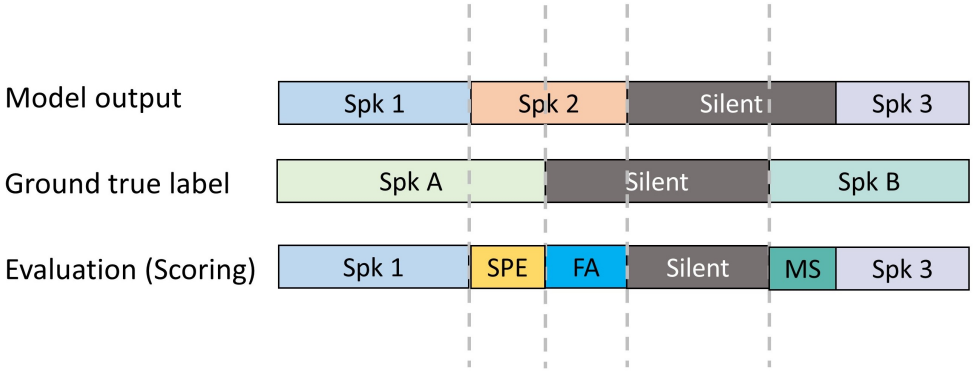


Figure 41.4.2: Schematics for calculating diarization error.

Note that since this metric does not take **overlapping speakers** into account, it can lead to increased missed speakers when overlapping speakers are present at ground truth labels. To cope with a more general setting where there are overlapping speakers in an utterance, we can tweak above definitions in the following way.

Definition 41.4.2 (generalized diarization error). *When there are overlapping speakers in an utterance, we define following generalized diarization error.*

- *False alarm is given by*
$$\max(n_{detect} - n_{true}, 0),$$
where n_{detect} is the number of speakers being detected and n_{true} is the number of speakers in the ground truth labels.
- *Missed Speaker error is given by*
$$\max(n_{true} - n_{detect}, 0).$$
- *Speaker error is given by*
$$\min(n_{true}, n_{detect}) - n_{correct},$$
where $n_{correct}$ is the number of correctly identified speakers.
- *Diarization error is the sum of the three errors.*

Who speaks when annotations should be provided using the RTTM file format. Each line in this file must follow the following convention:

SPEAKER *uri* 1 *start duration* <NA> <NA> *identifier* <NA> <NA>

where

- *uri* stands for "unique resource identifier" (think of it as the filename).
- *start* is the start time (elapsed time since the beginning of the file, in seconds) of the speech turn.
- *duration* is its duration (in seconds).
- *identifier* is the unique speaker identifier.
-

An example RTTM content is given below.

```
SPEAKER ES2002b.Mix-Headset 1 14.4270 1.3310 <NA> <NA> FEE005 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 16.9420 1.0360 <NA> <NA> FEE005 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 19.4630 1.0600 <NA> <NA> FEE005 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 23.0420 41.1690 <NA> <NA> FEE005 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 27.3530 1.2570 <NA> <NA> MEE008 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 30.4780 0.4810 <NA> <NA> MEE007 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 30.4970 3.6660 <NA> <NA> MEE008 <NA> <NA>
SPEAKER ES2002b.Mix-Headset 1 36.4310 2.5000 <NA> <NA> MEE008 <NA> <NA>
```

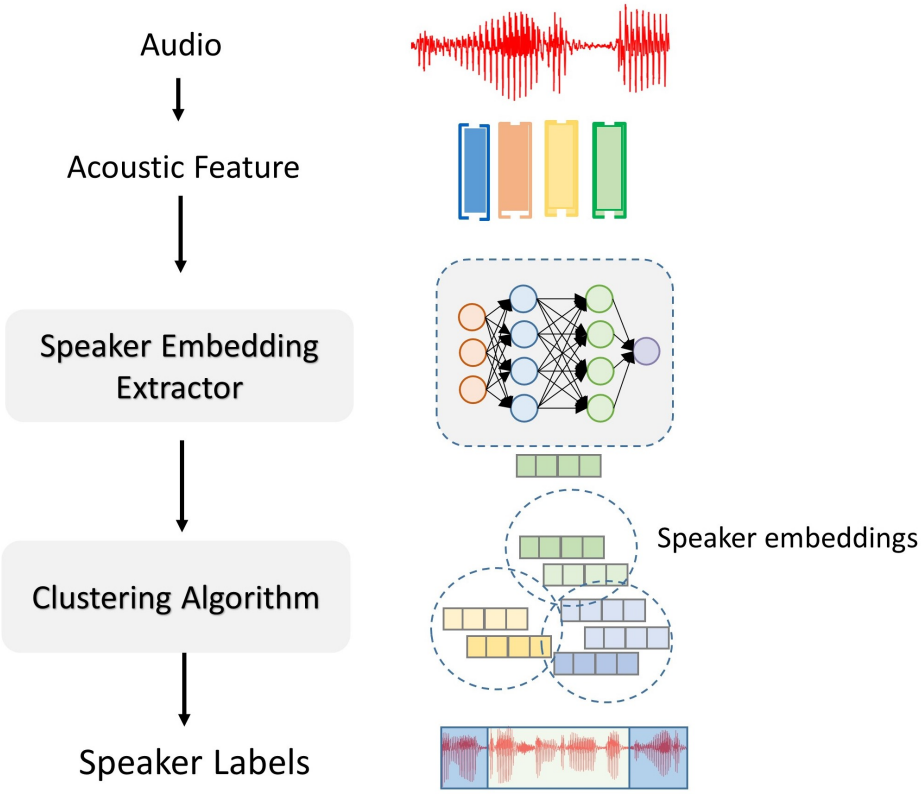


Figure 41.4.3: Conventional clustering approach pipeline for speaker diarization.

41.4.1.3 *Conventional clustering pipeline*

Conventional speaker diarization takes a modular approach [Figure 41.4.3] and is comprised of two steps: embedding generation and embedding clustering.

Techniques to generate embeddings include i-vector [6] based on traditional Gaussian mixture model and neural network based d-vector [24] and x-vector [8, 9, 25].

Clustering methods commonly used for speaker diarization are K-means [26], agglomerative hierarchical clustering [27], Gaussian mixture models[28], spectral clustering [29], etc.

Those generated embeddings are fed into a clustering module and produce clusters corresponding to different speakers. The clustering results can be used to produce speaker labels in an utterance.

Most clustering based approaches for speaker diarization are inherently unsupervised and there are two shortcomings: first, it is not optimized to directly minimize diarization errors because of un-supervision; second, ambiguity can appear when multiple clusters are closed to each other [Figure 41.4.4]. As a result of ambiguity, clustering

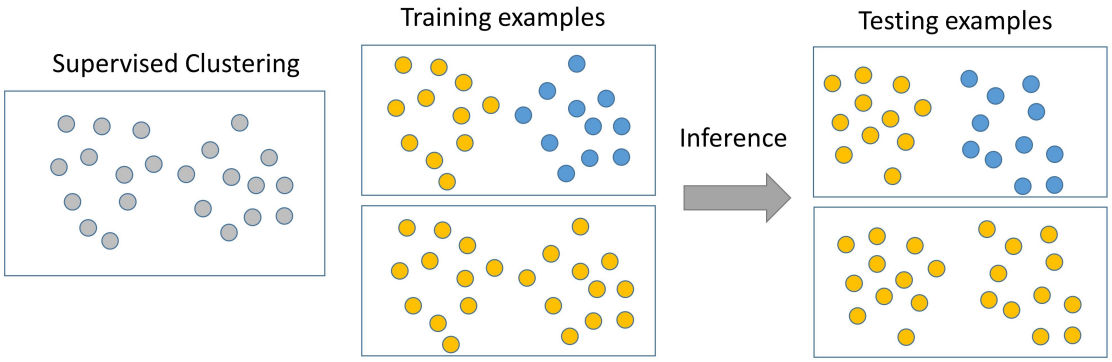


Figure 41.4.4: Supervised clustering method can help reduce ambuity in unsupervised clustering. The blobs in left figure can be one cluster or two clusters, depending on the algorithm parameter and model assumptions. In a supervised learning setting, training data examples (middle) provide additional clues on how to cluster data when ambiguity occurs (right).

result can also be highly sensitivity to the algorithm chosen, algorithm hyperparameters, and model assumptions. In supervised learning, patterns exists in the training data examples can provide additional clues on how to cluster data when ambiguity occurs.

41.4.1.4 *End-to-end neural diarization method*

As the deep learning methods and end-to-end systems achieve remarkable success in multiple domains, end-to-end neural speaker diarization system [Figure 41.4.5] gain popularity as part of the effort to reduce the system complexity of modular system. End-to-end system can also be trained in a supervised manner and achieve a smaller diarization error compared to clustering methods [30–32]. In an end-to-end neural diarization system, a neural network module that accepts multi-speaker audio and produces utterance embeddings. Usually, it comes with multiple outputs captures the joint speech speech activity of multiple speakers. To enable end-to-end supervised learning, some permutation-free training scheme is employed. Various techniques and architecture in natural language domain like Bidirectional BSTM and self-attentive Transformer can be incorporated in the framework.

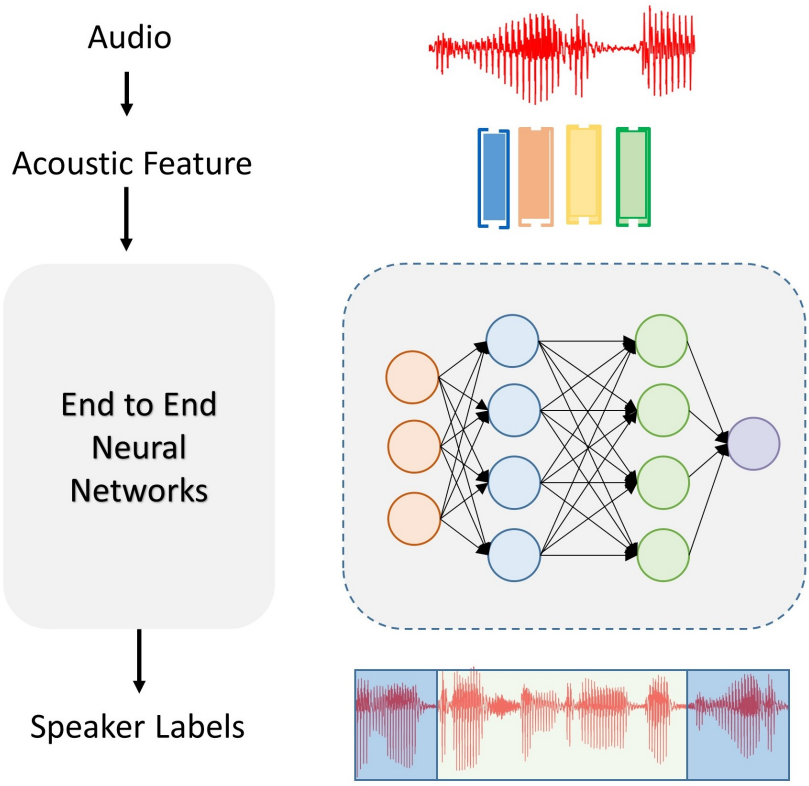


Figure 41.4.5: A generic end-to-end neural diarization pipeline for speaker diarization.

41.4.2 Clustering approach

41.4.2.1 *Hierarchical clustering system*

Clustering methods achieve initial marked success after combining i-vector representation and hierarchical clustering methods[9, 33]. In a typical clustering based speaker diarization system, diarization operates at a segment level consisting of around 20-100 frames. We have made following summary on how to construct segments.

Methodology 41.4.1 (speech segment construction). *We use following procedures to construct segments from an utterance for speaker diarization tasks.*

- *First we use a Voice Activity Detector (VAD) to determine voiced and unvoiced speech segments.*
- *Voiced segments are further divided into smaller non-overlapping segments using a maximal segment-length limit (e.g. 400 ms). The maximal segment-length determines the temporal resolution of the diarization results.*

Pair similarities or distances are computed using segment-wise embeddings. Then se apply a threshold on the similarities between embeddings of segments to merge different segments into clusters. **Hierarchical clustering** is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. One of most common Hierarchical clustering method is **agglomerative hierarchical clustering** performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together[Figure 41.4.6]. The method can be summarize as the following.

Methodology 41.4.2 (agglomerative hierarchical clustering). *The input data consists of N data points $x_i, i = 1, ..., N$.*

- *Initialization. Start with N clusters, whose cluster $C_i^{(0)} = \{x_i\}$ (i.e., only has one member).*
- *Merging. Suppose in $(k-1)$ iteration, we have $N+1-k$ clusters*

$$C_1^{(k-1)}, ..., C_{N+1-k}^{(k-1)}.$$

Compute pair-wise distance based a given distance metric (or similarity based on a given similarity metric) and merge two clusters with minimal distance. Now we have $N-k$ clusters given by

$$C_1^{(k)}, ..., C_{N-k}^{(k)}.$$

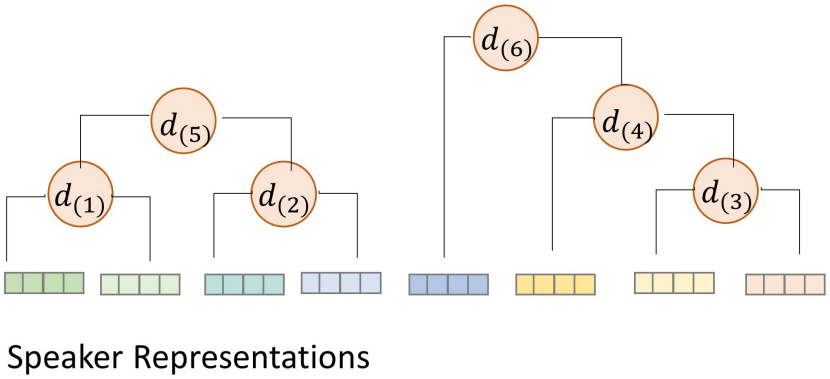


Figure 41.4.6: Bottom up scheme of agglomerative hierarchical clustering

- *Continuation or stop: Continue the merging step until stop criterion is met (e.g., all pair distances are greater than a threshold or number of clusters is smaller than a number).*

After clustering analysis, we assign each segment to a specific speaker. The initial segmentation result might be too coarse and a **resegmentation** needs to carry out to refine initial segmentation boundaries. Because resegmentation is performed after initial segmentation, it cannot be applied in online speaker diarization applications.

41.4.2.2 *LSTM based spectral clustering system*

In this system, audio signals are first transformed into frames of width 25ms and step 10ms, and log-mel-filterbank energies of dimension 40 are extracted from each frame as the network input. We build sliding windows of a fixed length on these frames, and run the LSTM network on each window. The last-frame output of the LSTM is then used as the d-vector representation of this sliding window.

Again, the first step is to construct segment via Methodology 41.4.1. For each segment, the segment-level d-vector embedding is obtained from the average of L2 normalized frame-level d-vectors. The segment-level embeddings can be fed into a clustering algorithm module like k-means, spectral clustering, to carry out speaker diarization tasks, including determining the number of unique speakers, and assign each part of the audio to a specific speaker.

Among different clustering algorithms, a modified spectral clustering algorithm yielded the best performance, as showed in the following.

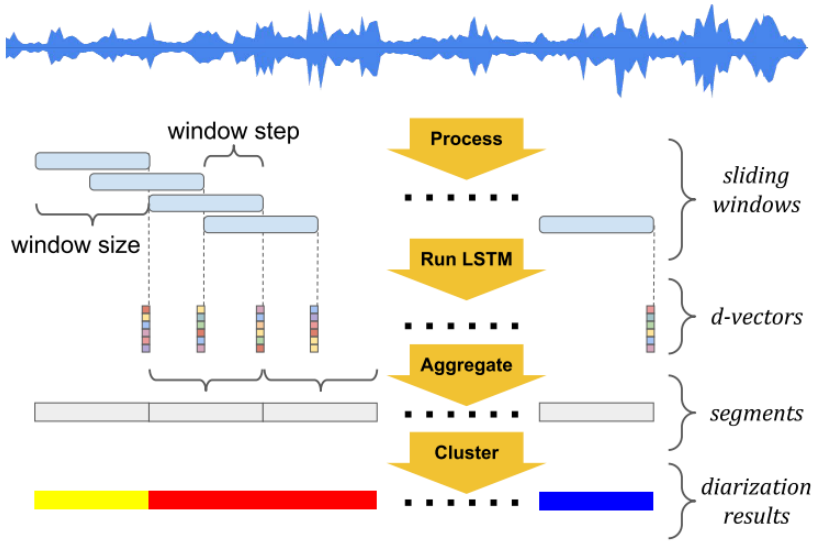


Figure 41.4.7: A LSTM speaker diarization system. Image from [29].

Methodology 41.4.3 (spectral clustering for speaker diarization). *Given N segments of x-vector representations, the spectral clustering consists of the following steps.*

- *Construct the affinity matrix $A \in \mathbb{R}^{N \times N}$, where A_{ij} is the cosine similarity between i th and j th segment embedding when $i \neq j$, and the diagonal elements are set to the maximal value in each row: $A_{ii} = \max_{j \neq i} A_{ij}$*
- *Refine the affinity matrix A, including Gaussian blurring, thresholding, symmetrization, etc.*
- *Perform eigen-decomposition on the refined affinity matrix. Let the eigen-vectors corresponding to the largest K eigenvalues be $v_1, v_2, \cdots, v_K, v_i \in \mathbb{R}^N$. For each segment i, we obtain its new representation*

$$e_i = \left[v_{1,i}, v_{2,i}, \cdots, v_{k,i}\right].$$

- *We then use K-means algorithm to perform clustering on these new embedding $e_1, ..., e_N$ and produce corresponding speaker labels.*

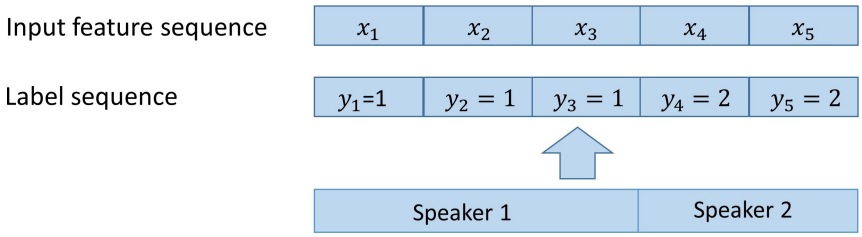


Figure 41.4.8: Training examples consisting of input feature sequence and the ground truth label sequence.

41.4.3 Generative approach: unbounded interleaved-state RNN (UIS-RNN)

41.4.3.1 *Model overview*

Contrasting with previous clustering based methods, unbounded interleaved-state RNN (UIS-RNN)[34] took a generative modeling approach and introduced a supervised learning scheme. Given an utterance with a real-valued vector sequence representation

$$X = (x_1, x_2, ..., x_T),$$

where $x_t \in \mathbb{R}^D$ is the embedding for the segment t in the utterance [segment construction is described in Methodology 41.4.1]. The *goal* of speaker diarization is to assign a label to each segment. Accordingly, the expected output for an utterance from a diarization system is

$$\hat{Y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_T), \hat{y}_i \in \mathbb{Z}_+.$$

In a supervised speaker diarization task, in the training dataset we are also given a sequence of ground-truth label for each segment, i.e., $Y = (y_1, y_2, ..., y_T), y_i \in \mathbb{Z}_+$ and y_i is the index of the speaker at segment t . For example, $Y = (1, 1, 1, 2, 2)$ means this utterance has five segments, from three different speakers, where $y_t = k$ means segment t belongs to speaker k[Figure 41.4.8].

In a high-level, UIS-RNN (Unbounded interleaved-state RNN)[34] specifies an online generative process of an entire utterance (X, Y), given by

$$p(X, Y|\theta) = p\left(x_1, y_1\right) \cdot \prod_{t=2}^{T} p\left(x_t, y_t | x_{[t-1]}, y_{[t-1]}\right),$$

Here θ are model parameters, $[t]$ denotes an ordered set $(1, 2, ..., t)$, and the equality is the result of joint distribution factorization.

In the **training** stage, where we are given N training data pair $\{X_i, Y_i\}_N$, we estimate model parameters θ via maximizing log-likelihood of observing the training data, i.e.,

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{N} p(X_i, Y_i|\theta).$$

In the **testing** or **inference** stage, where we are given a test utterance X^{test}, the decoding procedure is to look for a sequence of speaker labels $\hat{Y}$ that maximizes the joint probability:

$$\hat{Y} = \arg\max_{Y} P(X^{\text{test}}, Y|\hat{\theta}).$$

41.4.3.2 *Model structure*

In order to facilitate the model parameter estimation, UIS-RNN model introduces structure into the generative process; in particular, an augmented process is used:

$$p(X, Y, Z) = p\left(x_1, y_1\right) \cdot \prod_{t=2}^{T} p\left(x_t, y_t, z_t | x_{[t-1]}, y_{[t-1]}, z_{[t-1]}\right)$$

where $Z = (z_2, \ldots, z_T)$, and $z_t = \mathbb{1}\left(y_t \neq y_{t-1}\right) \in \{0, 1\}$ is a binary indicator for speaker changes at time t. For example, if $Y = (1, 1, 2, 3, 2, 2)$, then $Z = (0, 1, 1, 1, 0)$. Note that Z is uniquely determined by Y, but Y cannot be uniquely determined by a given Z, since we don't know which speaker we are changing to.

The conditional distribution on x_t, y_t, z_t given their history $(x_{[t-1]}, y_{[t-1]}, z_{[t-1]})$ is rewritten as:

$$\begin{aligned} & p\left(x_t, y_t,\ z_t | x_{[t-1]}, y_{[t-1]}, z_{[t-1]}\right) \\ & \quad = \underbrace{p\left(x_t | x_{[t-1]}, y_t\right)}_{\text{sequence generation}} \cdot \underbrace{p\left(y_t | y_{[t-1]}, z_t\right)}_{\text{assignment}} \cdot \underbrace{p\left(z_t | z_{[t-1]}\right)}_{\text{speaker change}} \end{aligned}$$

Now we can see the generative model has three components:

- speaker change model that models the change of z_t given its history $z_{[t-1]}$.
- speaker assignment model that models y_t given indicator z_t and its history $y_{[t-1]}$.
- sequence generation model the change of segment embedding x_t given history $x_{[t-1]}$ and $y_{[t]}$.

Now we discuss additional simplification and parameterization of its components.
Speaker change model

The speaker change indicator random variable z_t is assumed to take **Bernoulli distribution** with parameter λ, and z_t and z_{t-1} is assumed to be independent as well. That is,

$$z_t = \begin{cases} 1, \text{ with probability } \lambda \\ 0, \text{ with probability } 1-\lambda \end{cases}$$

Accordingly, we have probabilistic characterization on y_t given by

$$\begin{aligned} p(y_t = y_{t-1} | z_t = 0, y_{[t-1]}) &= 1-\lambda \\ p(y_t \neq y_{t-1} | z_t = 1, y_{[t-1]}) &= \lambda \end{aligned}$$

The specification on y_t values in $p(y_t \neq y_{t-1} | z_t = 1, y_{[t-1]})$ will be discussed in the following speaker assignment model. *Speaker assignment model*

The speaker assignment term is implemented as a **distance dependent Chinese Restaurant Processes** [35], which gives probabilistic characterization on the next restaurant to be visited, given the visiting history:

- Visit a new restaurant with some probability (i.e., model parameter α)
- Revisit a previously visited restaurant, with probability proportional to number of previous visit.

In our generative model, when $z_t = 0$, the speaker label $y_t = y_{t-1}$ (i.e., unchanged); when $z_t = 1$, we assume following change rule on $y_t = k, k \neq y_{t-1}$:

$$\begin{aligned} p\left(y_t = k \mid z_t = 1, y_{[t-1]}\right) &\propto N_{k,t-1} \\ p\left(y_t = K_{t-1} + 1 \mid z_t = 1, y_{[t-1]}\right) &\propto \alpha \end{aligned}$$

Here K_{t-1} is the total number of unique speakers up to time $(t-1)$. since $z_t = 1$ indicates a speaker change, we have $k \in \{1, ..., K_{t-1}\}, k \neq y_{t-1}$. In addition, we let $N_{k,t-1}$ be the number of continuous segments for speaker k up to time $t-1$. For example, if $y_{[6]} = (1,1,2,3,2,2)$, then there are four continuous segments $(1,1)|(2)|(3)|(2,2)$ separated by the vertical bar, with $N_{1,6} = 1, N_{2,6} = 2, N_{3,6} = 1$. The probability of switching back to a previously appeared speaker is proportional to the number of continuous speeches she/he has spoken. There is also a chance to switch to a new speaker, with a probability proportional to a constant α. *Sequence generation step*

The conditional distribution $p(x_t | x_{[t-1]}, y_t)$ is modeled by a group of parameter-sharing recurrent neural network instances - GRU. Particularly, for speaker $1, 2, ..., N, ...$, their GRU instances are denoted by $GRU^{(1)}, GRU^{(2)}, ..., GRU^{(N)}, ...$ At each t, we have y_t decides which GRU instance exclusively generates x_t based on historical $x_{[t-1]}$ that belongs to the

y_t speaker. These GRUs share the same set o parameters θ_{GRU}. In this way, the network $GRU^{(k)}$ is devoted to model $p(x_t|x_{[t-1]})$ and by using multiple instances, we remove the dependence on the label y_t in $p(x_t|x_{[t-1]}, y_t)$.

Consider a specific speaker y_t, its hidden states $h_t^{(y_t)}$ in its GRU instance will not be shared across different speakers, and the hidden state update can be written by

$$h_t = GRU^{(y_t)}(x_{t'}, h_{t'}|\theta_{GRU}),$$

where $t' = \max\{0, s < t : y_s = y_t\}$ be the last time we saw speaker y_t before t.

Denote all outputs from different GRU instances by $(m_1, ..., m_t)$, where m_s is the GRU instance y_s output at time s. Finally We assume the speaker embeddings are modeled by multivariate Gaussian distribution whose mean and variance is specified by GRU outputs. That is

$$x_t|x_{[t-1]}, y_t \sim N\left(\mu_t, \sigma^2 I\right)$$

where

$$\mu_t = \frac{\left(\sum_{s=1}^{t} \mathbf{1}\left(y_s = y_t\right) m_s\right)}{\left(\sum_{s=1}^{t} \mathbf{1}\left(y_s = y_t\right)\right)}$$

is the averaged GRU output associated with speaker y_t.[4]

41.4.3.3 *Maximum likelihood estimation*

The UIS-RNN generative model developed above include following parameters: the λ in the Bernoulli distribution characterizing z_t, the α in speaker assignment model, and the GRU network parameter θ_{GRU} and the variance σ in the sequence generation model. Given a training set $(X_1, X_2, \ldots, X_N)$ containing N utterances together with their labels $(Y_1, Y_2, \ldots, Y_N)$, we maximize the following log joint likelihood.

$$\max_{\theta_{GRU}, \alpha, \sigma^2, \lambda} \sum_{n=1}^{N} \ln p\left(X_n, Y_n, Z_n|\theta_{GRU}, \alpha, \sigma^2, \lambda\right)$$

Note that we have a closed-form solution for λ estimation

$$\lambda^* = \frac{\sum_{n=1}^{N} \sum_{t=2}^{T_n} \mathbf{1}\left(y_{n,t} = y_{n,t-1}\right)}{\sum_{n=1}^{N} T_n - N}$$

where T_n denotes the sequence length of the n th utterance.

The other parameters can be estimated by maximizing the log-likelihood function $\ln P(X|Y, Z)$. Note that Z can be directly derived from Y.

4 we can assume $x_0 = \mathbf{0}$ and $h_0 = \mathbf{0}$, meaning all GRU instances are initialized with the same zero state.

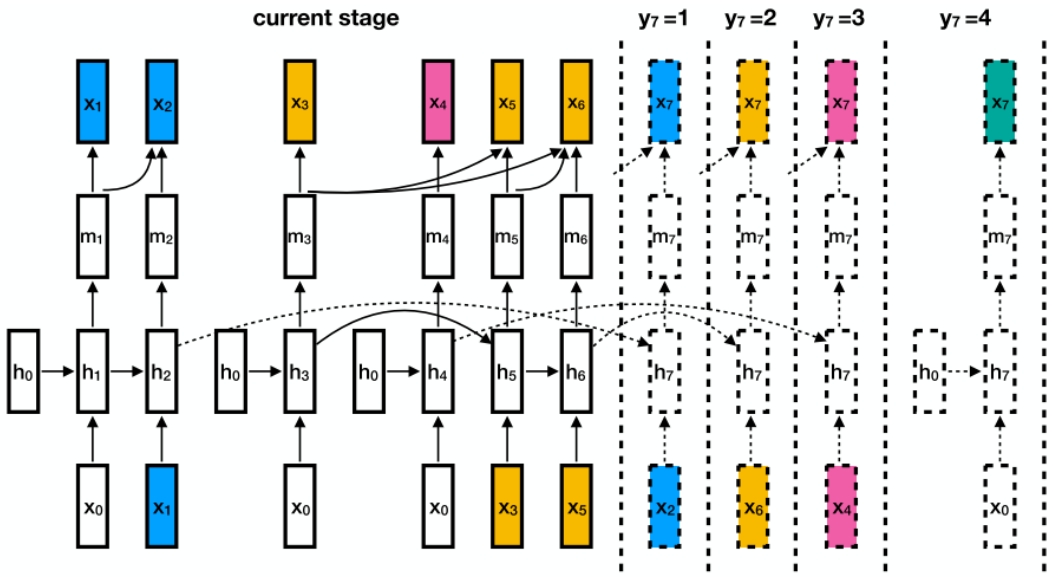


Figure 41.4.9: Generative process of UIS-RNN. Colors indicate labels for speaker segments. There are four options for y_7 given $x_{[6]}, y_{[6]}$. Image from [34].

41.4.3.4 *Decoding method*

Given a calibrated generative model, we can decode a testing utterance, where we assign each segment to one specific speaker[5]. Let $X^{\text{test}} = (x_1, x_2 \ldots, x_T)$ be the testing utterance, the decoding process aims to find speaker label Y^* that maximizes the joint probability, i.e.,

$$Y^* = \arg\max_Y p\left(X^{\text{test}}, Y\right).$$

A brute force search for global optimal solution would require an exhaustive search over the entire combinatorial speaker label space, which is at the prohibitive complexity $\mathcal{O}(T!)$. In practice, global optimal solution might not be necessary and we often adopt an online greedy decoding approach [algorithm 91].

The online decoding approach runs from $t = 1$ to T in a greedy manner, where at t, the decoded y_t^*, z_t^* maximize the joint probability up to t. That is,

$$\left(y_t^*, z_t^*\right) = \arg\max_{(y_t, z_t)} p(x_{[t]}, y_t).$$

5 Some algorithms we discuss later can handle the overlapping speaker situation

Decomposing $p(x_{[t]}, y_t)$ and using log scale, we have

$$(y_t^*, z_t^*) = \arg\max_{(y_t, z_t)} \left(\ln p\left(z_t\right) + \ln p\left(y_t | z_t, y_{[t-1]}\right) + \ln p\left(x_t | x_{[t-1]}, y_{[t-1]}^*, y_t\right) \right)$$

This greedy strategy will significantly reduce computational complexity to $\mathcal{O}\left(TC\right)$, where C is the maximum number of speakers in the utterance. In practice, we can apply a beam search to limit number of speakers considered in each step and adjust the number of look-ahead steps to achieve better decoding results.

Algorithm 91: Online greedy MAP decoding for UIS-RNN.

Input: Geneartive model and $X^{\text{test}} = (x_1, x_2 \ldots, x_T)$.

Initialize $x_0 = h_0 = 0$.

for $t = 1, 2, ..., T$ **do**

$$(y_t^*, z_t^*) = \arg\max_{(y_t, z_t)} \left(\ln p\left(z_t\right) + \ln p\left(y_t | z_t, y_{[t-1]}\right) + \ln p\left(x_t | x_{[t-1]}, y_{[t-1]}^*, y_t\right) \right).$$

Update $N_{k,t-1}$ (used in compute $p(y_t | z_t, y_{[t-1]}^*)$) and GRU hidden states.

end

Output: $Y^* = \left(y_1^*, y_2^*, \ldots, y_T^*\right)$.

41.4.4 Discriminative end-to-end neural speaker diarization

41.4.4.1 *Methodology overview*

Given a T-frame feature vector sequence $X = (x_1, ..., x_T), x_t \in \mathbb{R}^F$ from an audio signal, the speaker diarization problem tries to estimate the corresponding speaker label sequence $\hat{Y} = (\hat{y}_1, ..., \hat{y}_T)$, where y_t has C binary components, i.e., $y_{t,c} \in \{0, 1\}, c = 1, \cdots, C$, denoting a joint activity for $C, C > 1$, speakers at time index t. There could be overlapping situations where more than 1 speakers speaking at the same time. This can be represented by $y_{t,c} = 1$ and $y_{t,c'} = 1\left(c \neq c'\right)$. Thus, determining Y is a sufficient condition to determine the speaker diarization information.

In a discriminative modeling approach, the most probable speaker label sequence $\hat{Y}$ is selected from among all possible speaker label sequences $\mathcal{Y}$, as follows:

$$\hat{Y} = \arg\max_{Y \in \mathcal{Y}} P(Y \mid X)$$

$P(Y \mid X)$ can be factorized using the conditional independence assumption as follows:

$$\begin{aligned} P(Y \mid X) &= \prod_{t=1}^{T} P\left(y_t | y_1, \cdots y_{t-1}, X\right) \\ &\approx \prod_{t=1}^{T} P\left(y_t | X\right) \approx \prod_{t=1}^{T} \prod_{c=1}^{C} P\left(y_{t,c} \mid X\right) \end{aligned}$$

Here, we assume the frame-wise posterior is conditioned on *all previous inputs*, and each speaker is *present independently*. This contrast the UIS-RNN model, where the presence of each speaker is exclusive [subsection 41.4.3].

The difficulty on training of the model described above is that the model have to deal with the speaker permutations: changing an order of speakers within a correct label sequence is also regarded as correct. A critical contribution in [30] is the proposal of permutation-invariant training loss, which enables end-to-end training.

More specifically, we utilize the utterance-level permutation invariant training (PIT) criterion in the proposed method. We apply the PIT criterion on time sequence of speaker labels instead of time-frequency mask used in.

Definition 41.4.3 (permutation invariant training loss). *Consider an utterance with T frames and C speakers. Let $Y = (y_1, ...y_T), y_t \in \{0,1\}^C$ [a]. Let $p = (p_1, ..., p_T), p_t \mathbb{R}^C, p_{t,c} = P(y_{t,c} = 1|X)$. The PIT loss function is defined by*

$$L_{PIT} = \frac{1}{TC} \min_{\phi \in \text{perm}(C)} \sum_t \text{BCE}\left(y_t^{\phi}, p_t\right)$$

where $\text{perm}(C)$ is a set of all the possible permutation of $(1, \ldots, C)$, and y_t^{ϕ} is the ϕ permutated of the ground-truth speaker label, $\text{BCE}(\cdot, \cdot)$ is the binary cross entropy function between the label and the output. That is

$$BCE(y_t, p_t) = -\sum_{c=1}^{C} \left[y_{t,c} \ln p_{t,c} - (1 - y_{t,c}) \ln(1 - p_{t,c}\right].$$

[a] For example, let $C = 2$, y_t can take $(0,1), (0,0), (1,0), (1,1)$.

Equipped with permutation invariant training loss, a general end-to-end speaker diarization system is shown in Figure 41.4.10.

In the following, we start with different encoder strategies, including a bidirectional LSTM system [30] and a self-attentive Transformer encoder system [32]. There are two major limitations:

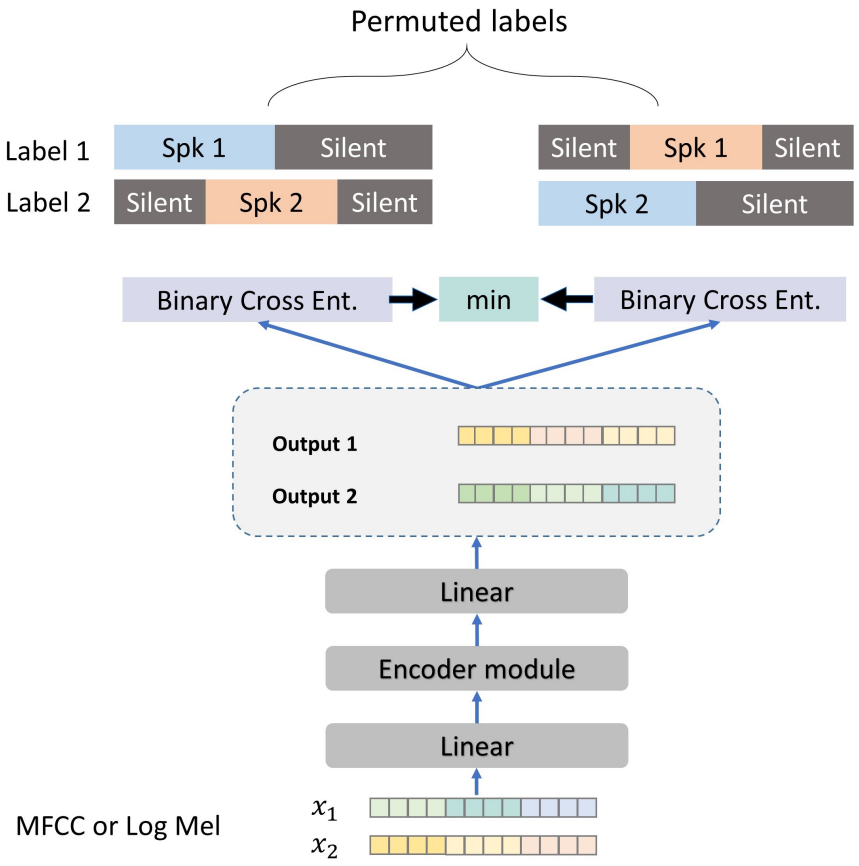


Figure 41.4.10: A general end-to-end neural speaker diarization system with permutation invariant training loss.

- The number of speakers is tied to neural architecture. Therefore same number of speakers are required in both training and inference stage and in all training examples, which impose significant inflexibility in real-world applications.
- Encoding process requires the whose utterance as input, which limits the method to offline application.

To enable flexible speaker numbers, [32] introduce encoder-decoder attractor structure to dynamically generate number of speakers based on a given utterance.

41.4.4.2 *Stacked BLSTM system*

In [30], the frame-wise posterior $P\left(y_{t,c}|X\right), c = 1, ..., C$ is modeled with P layer bidirectional long short-term memory (BLSTM), as follows [Figure 41.4.11]:

$$
\begin{aligned}
h_t^{(1)} &= \text{BLSTM}_t\left(x_1, \cdots, x_T\right), h_t^{(1)} \in R^{2H} \\
h_t^{(2)} &= \text{BLSTM}_t\left(h_1^{(1)}, \cdots, h_T^{(p-1)}\right) \\
&\cdot = \cdots \\
h_t^{(P)} &= \text{BLSTM}_t\left(h_1^{(P-1)}, \cdots, h_T^{(P-1)}\right) \\
z_t &= \text{Sigmoid}\left(W_h h_t^{(P)} + b_h\right), W_h \in \mathbb{R}^{2H \times C}, b_h \in \mathbb{R}^C
\end{aligned}
$$

where $\text{BLSTM}_t(\cdot)$ is a BLSTM layer which accepts an input sequence and outputs $2H$-dimensional hidden activations $h_t^{(p)}$ at time index t. Note that the output from BLSTM is denoted by $h_t^{(p)}$, which consists of hidden states from two directions and therefore has a dimensionality of $2H$.

The final Sigmoid function converts activations to a quantity $z_t \in (0, 1)$ between 0 and 1, which approximates $P\left(y_{t,c}|X\right)$.

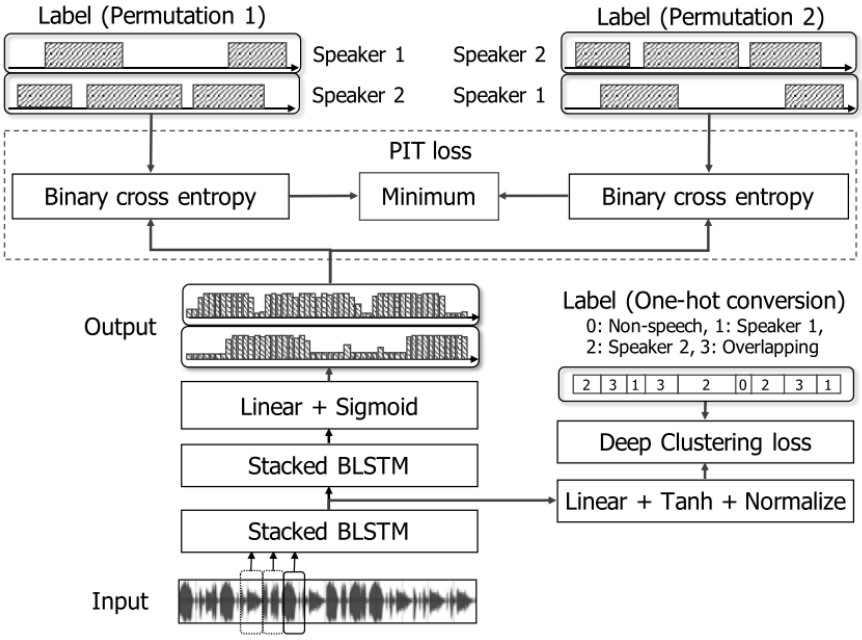


Figure 41.4.11: Two-speaker end-to-end neural speaker diarization (EEND) model trained with the permutation invariant loss and the deep clustering loss. Image from [30].

41.4.4.3 *Self-attentive Transformer system*

Another encoding strategy is to use self-attentive Transformer encoder [36], which can potentially benefit from the ability to capture long-term dependence via attention modules. Given a sequence of acoustic feature vectors $(x_1, ..., x_T), x_t \in \mathbb{R}^F$, the Transformer encoder convert them to embedding vectors $(e_1, ..., e_T), e_t \in \mathbb{R}^D$.

The architecture of the encoder block is depicted in Figure 41.4.12. First, each input feature x_t of dimensionality F is projected into a D dimensional space. Then P encoder blocks are used to produce encoding $(e_1^{(P)}, ..., e_T^{(P)})$ in the last encoder block. Finally, output layer consisting of linear sigmoid operation produces frame-wise posteriors $p(y_t|X)$. More formally,

$$
\begin{aligned}
e_t^{(0)} &= W_X x_t + b_0, t = 1, ..., T \\
e_t^{(1)} &= \text{Encoder}_t^{(1)}\left(e_1^{(0)}, \cdots, e_T^{(0)}\right) \\
\cdots &= \cdots \\
e_t^{(P)} &= \text{Encoder}_t^{(P)}\left(e_1^{(P-1)}, \cdots, e_T^{(P-1)}\right)
\end{aligned}
$$

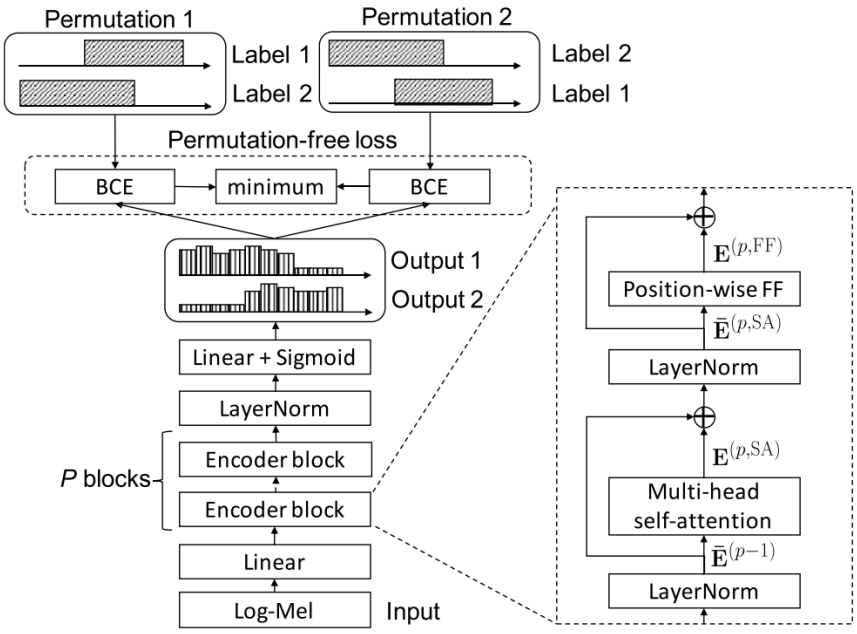


Figure 41.4.12: Two-speaker self-attentive end-to-end neural speaker diarization model trained with permutation-free loss. Image from [31].

where $W_X \in \mathbb{R}^{D\times F}$ and $b_0 \in \mathbb{R}^D$ project an input feature x_t of dimensionality F into a D dimensional space. Note the encoder blocks used here have similar structure as in [36] except that no positional encoding is used. In the following, we first discuss in detail the two sub-structures in each encoder block, a multi-head self-attention layer and the second is a position-wise feed-forward layer. Then we discuss output layer.

Multi-head self-attention layer

The multi-head self-attention layer transforms a sequence of input vectors as follows.

- The sequence of vectors $\left(e_t^{(p-1)} \mid t = 1, \cdots, T\right)$ is converted into a $\mathrm{R}^{T\times D}$ matrix, followed by layer normalization:

$$\overline{E}^{(p-1)} = \text{LayerNorm}\left(\left[e_i^{(p-1)}, \cdots e_T^{(p-1)}\right]^\top\right) \in \mathbb{R}^{T\times D}$$

- Compute **query vector** $\overline{E}^{(p-1)} Q_h^{(p)} \in \mathbb{R}^{T\times d}$, where $Q_h^{(p)}$ is the query projection matrix for the h-th bead, $d = D/H$ is the dimensionality of each head, and H is the number of heads.
- Compute **key vector** $\overline{E}^{(p-1)} K_h^{(p)} \in \mathbb{R}^{T\times d}$, $K_h^{(p)} \in \mathbb{R}^{D\times d}$ is the key projection matrix
- Pairwise similarity matrix

$$A_b^{(p)} = \overline{E}^{(p-1)} Q_h^{(p)} \left(\overline{E}^{(p-1)} K_h^{(p)}\right)^T \in R^{T\times T} (1 \le h \le H)$$

- Attention weight matrix

$$\hat{A}_h^{(p)} = \text{Softmax}\left(\frac{A_h^{(p)}}{\sqrt{d}}\right) \in \mathbb{R}^{T\times T}.$$

Then, using the attention weight matrix, context vectors $C_h^{(p)}$ are computed as a weighted sum of the value vectors $\overline{E}^{(j-1)}V_h^{(p)} \in \mathbb{R}^{T\times d}$

$$C_h^{(p)} = \hat{A}_h^{(p)}\left(E^{(p-1)}V_h^{(p)}\right) \in \mathbb{R}^{T\times d}$$

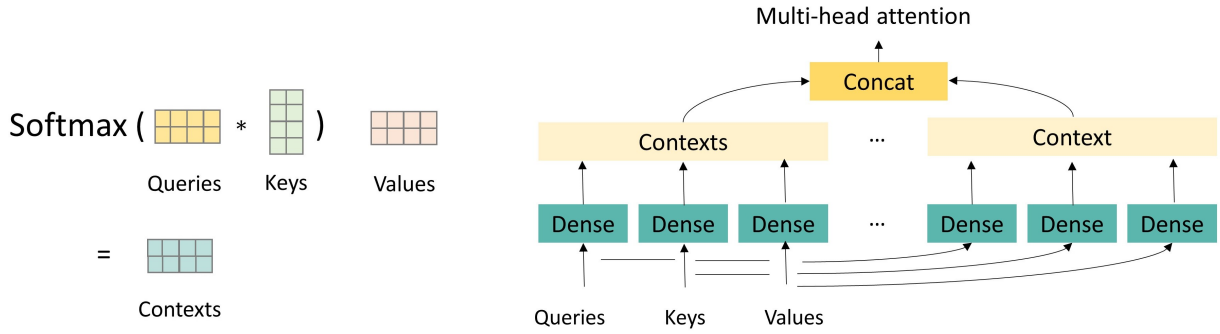
where $V_h \in \mathbb{R}^{D\times d}$ is the value projection matrix. Finally, the **context vectors** for all heads are concatenated and projected using the output projection matrix $O^{(p)} \in \mathbb{R}^{D\times D}$:

$$C^{(p)} = \left[C_i^{(p)} \dots C_M^{(p)}\right]$$
$$E^{(p,SA)} = C^{(p)}O^{(p)} \in \mathbb{R}^{T\times D}$$

The computation of context vectors in each attention head and the concatenation of multiple context vectors are summarized in

Following the self-attention layer, a residual connection and layer normalization is applied:

$$\overline{E}^{(p,\text{SA})} = \text{LayerNorm}\left(\overline{E}^{(p-1)} + E^{(p,SA)}\right) \in \mathbb{R}^{T\times D}$$



(a) Computation of context vector based on self-attention mechanism. **(b)** Multihead self-attention by concatenating context vectors.

Figure 41.4.13: Multihead self-attention calculation scheme.

Position-wise feed-forward layer

After the attention module, embeddings $\overline{E}^{(p,SA)}$ are further fed into a position-wise feed-forward layer transforms [36] given by

$$E^{(p,\text{FF})} = \text{ReLU}\left(\overline{E}^{(p,SA)} W_1^{(p)} + b_1^{(p)}\mathbf{1}\right) W_2^{(p)} + b_2^{(p)} 1$$

where $E^{(p,FF)} \in \mathbb{R}^{T\times D}, W_1^{(p)} \in \mathbb{R}^{D\times d_{\text{ff}}}$ and $\mathbf{b}_1^{(p)} \in \mathbb{R}^{d_{ff}}$ are the first linear projection matrix and bias, respectively, $\mathbf{1} \in \mathbb{R}^{1\times T}$ is an all-one row vector. d_{ff} is the number of internal units in this layer. $W_2^{(p)} \in \mathbb{R}^{d_{ff}\times D}$ and $b_2^{(p)} \in \mathbb{R}^D$ are the second linear projection matrix and bias, respectively

For simplicity, we can also write

$$E^{(p,\text{FF})} = \text{FFN}(\overline{E}^{(p,SA)}),$$

where $\text{FFN}(x) = ReLU(xW_1 + b_1)W_2 + b_2$ and the linear transformations have the same parameters across different positions (i.e., time index).

Finally, the output of the encoder block $e_t^{(p)}$ for each time frame is computed by applying a residual connection as follows:

$$\left[e_1^{(p)} \cdots e_T^{(p)}\right] = \left(\overline{E}^{(p,SA)} + E^{(p,FF)}\right)^T.$$

Output layer for frame-wise posteriors
The frame-wise posteriors $(z_1, ..., z_T), z_t \in \mathbb{R}^C$ are calculated from $e_1^{(P)}, ..., e_T^{(P)}, e_1^{(P)} \in \mathbb{R}^D$ using layer normalization and a fully-connected layer as follows:

$$\overline{E}^{(P)} = \text{LayerNorm}\left(\left[e_1^{(P)} \ldots e_T^{\langle P\rangle}\right]^\top\right) \in \mathbb{R}^{T\times D}$$
$$[z_1 \cdot \cdots \cdot z_T] = \text{Sigmoid}\left(\overline{E}^{(P)} W_E + b_E\mathbf{1}\right)^\top$$

where $W_E \in \mathbb{R}^{D\times C}$ and $b_E \in \mathbb{R}^C$ are the linear projection matrix and bias.

41.5 Notes on Bibliography

41.5.1 Literature

General books on speech recognition include [37][38][39][40].

Relevant software include PyTorch-Kaldi [41], Kaldi, S3PRL[42].

For an excellent overview of speaker recognition, see [43].

41.5.2 Dataset

TIMIT: TIMIT is a small data set consisting of 6000 utterances from 600 speakers. Each speaker has 10 utterances. Closely related datasets are NTIMIT and CTIMIT, which add artificial noise to TIMIT to mimic the real-world noisy environment.

VoxCeleb and VoxCeleb2: VoxCeleb/VoxCeleb2 is a large-scale speaker identification dataset. It contains more than 1 million utterances from 7,000 speakers, lasting more than 2,000 hours in total. The audio data are extracted from YouTube videos, spanning a diverse range of accents, professions and age. More details are in [44] and [14].

LibriSpeech: LibriSpeech consists of approximately 1,000 hours of 16kHz read English speech. The data is derived from read audiobooks from the LibriVox project. The LibriVox project is a volunteer effort responsible for the creation of approximately 8,000 public domain audio books, the majority of which are in English. Most of the recordings are based on texts from Project Gutenberg, also in the public domain.

A summary of the LibriSpeech corpus is given by the following table [45]:

subset	hours	per-spk minutes	female spkrs	male spkrs	total spkrs
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Other existing datasets are reviewed in [44].

Speaker diarization dataset: ChiME-6 Challenge [46], LibriCSS [47], Librimix [48], Second DIHARD Diarization Challenge [49]

Musan dataset is a large corpus of music, speech, and noise recordings, which can used to train VAD and perform data augmentation (by adding noise). [50] describes how to augment audio with room reverberation. has the downloadable resources.

Chime-6 challenge: In this paper, we describe a multi-microphone multispeaker ASR system developed using many of these methods for the CHiME-6 challenge [27]. The challenge aims to improve speech recognition and speaker diarization for far-field conversational speech in challenging environments in a multimicrophone setting. The CHiME-6 data [5] contains a total of 20 4-speaker dinner party recordings. Each dinner party is two to three hours long and is recorded simultaneously on the participants' ear-worn microphone and six microphone arrays placed in the kitchen, dining room, and the living room.

DIHARD is a new annual challenge focusing on "hard" diarization; that is, speech diarization for challenging corpora where there is an expectation that the current state-of-the-art will fare poorly, including, but not limited to:

clinical interviews extended child language acquisition recordings YouTube videos "speech in the wild" (e.g., recordings in restaurants)

BIBLIOGRAPHY

1. Reynolds, D. A., Quatieri, T. F. & Dunn, R. B. Speaker verification using adapted Gaussian mixture models. *Digital signal processing* **10,** 19–41 (2000).
2. Campbell, W. M., Sturim, D. E. & Reynolds, D. A. Support vector machines using GMM supervectors for speaker verification. *IEEE signal processing letters* **13,** 308–311 (2006).
3. Kenny, P., Ouellet, P., Dehak, N., Gupta, V. & Dumouchel, P. A study of interspeaker variability in speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing* **16,** 980–988 (2008).
4. Kenny, P., Boulianne, G., Ouellet, P. & Dumouchel, P. Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing* **15,** 1435–1447 (2007).
5. Kenny, P. Joint factor analysis of speaker and session variability: Theory and algorithms. *CRIM, Montreal,(Report) CRIM-06/08-13* **14,** 28–29 (2005).
6. Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P. & Ouellet, P. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing* **19,** 788–798 (2010).
7. Burget, L. *et al. Discriminatively trained probabilistic linear discriminant analysis for speaker verification* in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (2011), 4832–4835.
8. Snyder, D., Garcia-Romero, D., Sell, G., Povey, D. & Khudanpur, S. *X-vectors: Robust dnn embeddings for speaker recognition* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 5329–5333.
9. Peddinti, V., Povey, D. & Khudanpur, S. *A time delay neural network architecture for efficient modeling of long temporal contexts* in *Sixteenth Annual Conference of the International Speech Communication Association* (2015).
10. Wan, L., Wang, Q., Papir, A. & Moreno, I. L. *Generalized end-to-end loss for speaker verification* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 4879–4883.
11. Jia, Y., Wu, Z., Xu, Y., Ke, D. & Su, K. Long short-term memory projection recurrent neural network architectures for piano's continuous note recognition. *Journal of Robotics* **2017** (2017).

12. Sak, H., Senior, A. W. & Beaufays, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling (2014).

13. Garcia-Romero, D., Sell, G. & McCree, A. *Magneto: X-vector magnitude estimation network plus offset for improved speaker recognition* in ().

14. Chung, J. S., Nagrani, A. & Zisserman, A. Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622* (2018).

15. Cai, W., Chen, J. & Li, M. *Exploring the Encoding Layer and Loss Function in End-to-End Speaker and Language Recognition System* in *Odyssey* (2018).

16. Okabe, K., Koshinaka, T. & Shinoda, K. Attentive statistics pooling for deep speaker embedding. *arXiv preprint arXiv:1803.10963* (2018).

17. Snell, J., Swersky, K. & Zemel, R. S. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175* (2017).

18. Wang, J., Wang, K.-C., Law, M. T., Rudzicz, F. & Brudno, M. *Centroid-based deep metric learning for speaker recognition* in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), 3652–3656.

19. Wang, H. *et al.* CosFace: Large Margin Cosine Loss for Deep Face Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5265–5274. ISSN: 10636919. arXiv: 1801.09414 (2018).

20. Xiang, X., Wang, S., Huang, H., Qian, Y. & Yu, K. Margin matters: Towards more discriminative deep neural network embeddings for speaker recognition. *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA ASC 2019*, 1652–1656. arXiv: 1906.07317 (2019).

21. Chung, J. S. *et al.* In defence of metric learning for speaker recognition. arXiv: 2003.11982 (2020).

22. Liu, W. *et al.* SphereFace: Deep Hypersphere Embedding for Face Recognition. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6738–6746 (2017).

23. Deng, J., Guo, J., Xue, N. & Zafeiriou, S. *Arcface: Additive angular margin loss for deep face recognition* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 4690–4699.

24. Heigold, G., Moreno, I., Bengio, S. & Shazeer, N. *End-to-end text-dependent speaker verification* in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), 5115–5119.

25. Snyder, D., Garcia-Romero, D., Povey, D. & Khudanpur, S. *Deep Neural Network Embeddings for Text-Independent Speaker Verification.* in *Interspeech* (2017), 999–1003.

26. Shum, S., Dehak, N., Chuangsuwanich, E., Reynolds, D. & Glass, J. *Exploiting intra-conversation variability for speaker diarization* in *Twelfth Annual Conference of the International Speech Communication Association* (2011).

27. Sell, G., Garcia-Romero, D. & McCree, A. Speaker diarization with i-vectors from DNN senone posteriors. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* **2015-January,** 3096–3099. ISSN: 19909772 (2015).

28. Shum, S., Dehak, N. & Glass, J. Unsupervised Methods for Speaker Diarization : An Integrated and Iterative Approach. **21** (2012).

29. Wang, Q., Downey, C., Wan, L., Mansfield, P. A. & Moreno, I. L. *Speaker diarization with lstm* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 5239–5243.

30. Fujita, Y., Kanda, N., Horiguchi, S., Nagamatsu, K. & Watanabe, S. End-to-end neural speaker diarization with permutation-free objectives. *arXiv preprint arXiv:1909.05952* (2019).

31. Fujita, Y. *et al. End-to-end neural speaker diarization with self-attention* in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)* (2019), 296–303.

32. Horiguchi, S., Fujita, Y., Watanabe, S., Xue, Y. & Nagamatsu, K. End-to-End Speaker Diarization for an Unknown Number of Speakers with Encoder-Decoder Based Attractors. *arXiv preprint arXiv:2005.09921* (2020).

33. Garcia-Romero, D., Snyder, D., Sell, G., Povey, D. & McCree, A. *Speaker diarization using deep neural network embeddings* in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), 4930–4934.

34. Zhang, A., Wang, Q., Zhu, Z., Paisley, J. & Wang, C. *Fully supervised speaker diarization* in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), 6301–6305.

35. Blei, D. M. & Frazier, P. I. Distance Dependent Chinese Restaurant Processes. *Journal of Machine Learning Research* **12** (2011).

36. Vaswani, A. *et al. Attention is all you need* in *Advances in neural information processing systems* (2017), 5998–6008.

37. Rabiner, L., Rabiner, L. & Juang, B. *Fundamentals of Speech Recognition* ISBN: 9780130151575 (PTR Prentice Hall, 1993).

38. Huang, X., Acero, A., Hon, H.-W. & Reddy, R. *Spoken language processing: A guide to theory, algorithm, and system development* (Prentice hall PTR, 2001).

39. Yu, D. & Deng, L. *Automatic Speech Recognition: A Deep Learning Approach* ISBN: 9781447157793 (Springer London, 2014).

40. Li, J., Deng, L., Haeb-Umbach, R. & Gong, Y. *Robust Automatic Speech Recognition: A Bridge to Practical Applications* ISBN: 9780128026168 (Elsevier Science, 2015).

41. Ravanelli, M., Parcollet, T. & Bengio, Y. *The pytorch-kaldi speech recognition toolkit* in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), 6465–6469.

42. Liu, A. T. & Shu-wen, Y. *S3PRL: The Self-Supervised Speech Pre-training and Representation Learning Toolkit* 2020.

43. Hansen, J. H. & Hasan, T. Speaker recognition by machines and humans: A tutorial review. *IEEE Signal processing magazine* **32,** 74–99 (2015).

44. Nagrani, A., Chung, J. S. & Zisserman, A. Voxceleb: a large-scale speaker identification dataset. *arXiv preprint arXiv:1706.08612* (2017).

45. Panayotov, V., Chen, G., Povey, D. & Khudanpur, S. *Librispeech: an asr corpus based on public domain audio books* in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), 5206–5210.

46. Watanabe, S., Mandel, M., Barker, J. & Vincent, E. CHiME-6 Challenge: Tackling multispeaker speech recognition for unsegmented recordings. *arXiv preprint arXiv:2004.09249* (2020).

47. Chen, Z. *et al. Continuous speech separation: Dataset and analysis* in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), 7284–7288.

48. Cosentino, J., Pariente, M., Cornell, S., Deleforge, A. & Vincent, E. LibriMix: An Open-Source Dataset for Generalizable Speech Separation. *arXiv preprint arXiv:2005.11262* (2020).

49. Ryant, N. *et al.* The second DIHARD diarization challenge: Dataset, task, and baselines. *arXiv preprint arXiv:1906.07839* (2019).

50. Ko, T., Peddinti, V., Povey, D., Seltzer, M. L. & Khudanpur, S. *A study on data augmentation of reverberant speech for robust speech recognition* in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), 5220–5224.