Hidden Markov Models and Connectionist Temporal Classification

Jianbo Ma

July 2022

1 Introduction

This document is a part of series for sequential modeling techniques. It provides a view that categorize different branches of algorithms into types of sequence to sequence modeling. Focus on Hidden Markov Models (HMM) and Connectionist Temporal Classification (CTC) and how to merge them into the same framework.

Overall, HMM first defines the number of states. The states are discrete. The HMM defines a graphical structure, which contains the number of discrete states and all parameters. One way to train the HMM parameters is the Expectation-Maximization (EM) algorithm that maximizes the likelihood function $p(\mathrm{X}|\theta)$, where X is the collection of observed data and θ denotes the parameters of HMM.

The closed form solution of maximizing $p(\mathrm{X}|\theta)$ is not trivial. Instead, EM algorithm can be used. The underlying idea is that, by specifying the probabilistic graph structure and its parameters, how likely the observed data are generated by this graph with its parameters. By fixing the structure, but updating the parameters, we expect the likelihood should increase, which indicates the observed data is more likely to be generated from the graph with the updated parameters.

We can see that HMM is generative. But the underlying idea of CTC is discriminative. The HMM graph and its parameters is to generate observed data, while CTC focuses on the discrepancy between ground truth labels and predicted labels.

2 HMM

HMM is a special type of Markov Chain. We can develop the HMM from identical, independent distributed (i.i.d) data.

Suppose there is a collection of observations $\{x_n\} \quad n \in \{1, 2, 3, ..., N\}$. The joint probability of the observations can be expressed as:

$$p(x_1, x_2, ..., x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)...p(x_N|x_{N-1}, x_{N-2}, ..., x_1) = \prod_{n=1}^{N} p(x_N|x_{N-1}, ..., x_1) \tag{1}$$

If the assumption of the observations are i.i.d, since the observations are independent, the conditional probability $p(x_N|x_{N-1}, ..., x_1)$ is shrunk to $p(x_N)$, which makes the 1 become

$$P(x_1, x_2, ..., x_N) = \prod_{n=1}^{N} p(x_N) \tag{2}$$

But for some situations, the i.i.d is not a good assumption, such as speech signal, where only within a short amount of time (typically 10 to 100 milliseconds) the i.i.d is weakly met.

The first-order Markov Chain is then one of different assumptions to tight the i.i.d assumption that typically useful for sequential data. Instead of being totally independent with all the observations, the conditional probability $p(x_N|x_{N-1}, ..., x_1)$ is given by $p(x_N|x_{N-1})$, which means that the current observation is independent of all previous observations except the most recent one.

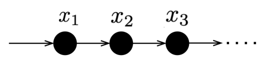

Figure 1: illustration of first-order Markov Chain of observations x_n.

Then it is straightforward to rewrite the joint probability to

$$P(x_1, x_2, ..., x_N) = \prod_{n=1}^{N} p(x_N|x_{N-1}). \tag{3}$$

2.1 From Markov Chain To HMM

Figure 1 illustrate the dependencies of observation in a Markov Chain property, where relationship between observations are modeled directly. This is very helpful to model observations directly.

In many situations, it may be more inspiring and powerful to illustrate the observations in a generative way. This happens in the model such as Gaussian or Gaussian Mixture Models (GMM). This is very powerful in the machine learning and pattern recognition area. In this generative view, we regard the observations are drawn from a particular distribution, such as GMM. Those observations are sampled from a distribution with some probability density function (pdf). By collecting enough samples, we expect to infer the underlying pdf with assumption that the distribution follows a particular type.

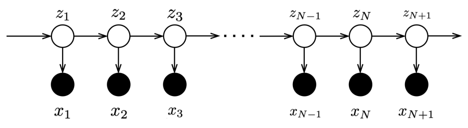

Figure 2: illustration of Hidden Markov Model.

In contrast to figure 1, we explain how the observations are derived using probabilistic graphical model as presented in figure 2.

In this representation, the shaded circles are observations. The empty circles are those variables that are hidden, which we call latent variables. For each observation x_n, there is a particular latent variable realization z_n associated with it. If this variable only takes limited and discrete possibilities, we then can call this variable as state. In a particular state, there is a particular pdf associated with it. And the observation is drawn from that distribution.

At this point, we have covered most of the components of HMM. Now we can use more rigorous mathematical way to develop the model.

2.1.1 HMM mathematical expressions

The HMM is the case when the latent variable in figure 2 is discrete. We can conveniently use a one-hot vector to represent the variable, which is the same form used in [1]. Suppose there are K different states. For example, there are K different phones we want to model in a speech recognizer. The k^{th} state of n^{th} sequential index is represented as,

$$z_n^k = 1 \quad and \quad \sum_{k=1}^{K} z_n^k = 1. \tag{4}$$

We then use the bold lower character $\mathbf{z_n}$ to represent the latent variable as vector.

From the probabilistic graphical model figure 2, there is an implicit property of this graph. That is the conditional independent property. We see that $\mathbf{z_n}$ is a *head-to-tail* node in this graph, which means that nodes linked to it are conditional independent. To express it, we have

$$\mathbf{z_{n+1}} \perp \mathbf{z_{n-1}} | \mathbf{z_n}. \tag{5}$$

We call this property as $\mathbf{z_{n+1}}$ and $\mathbf{z_{n-1}}$ are independent conditioned on $\mathbf{z_n}$. The joint probability of figure 2 is expressed as,

$$P(x_1, x_2, ..., x_N, \mathbf{z_1}, \mathbf{z_2}, ...\mathbf{z_N}) = p(\mathbf{z_1})p(x_1|\mathbf{z_1}) \prod_{n=2}^{N} p(\mathbf{z_n}|\mathbf{z_{n-1}})p(x_n|\mathbf{z_n}). \tag{6}$$

In a more realistic model, we restrict the probability given a state and state transition is fixed over different time or sequential index. This is then called homogeneous model. The component $p(\mathbf{z_n}|\mathbf{z_{n-1}})$ is then can be expressed as a $K \times K$ matrix A, which is usually called transition matrix and its element $a_{ij} = p(z_{n,i} = 1|z_{n-1,j} = 1)$.

In Eq. 6, the choice of $p(\mathbf{z_1})$ will usually modeled by a parameter, which is called initial state probability, and denoted as π. The emission probability density function are also parameterized and denoted using ϕ. Then we ended with the whole set of parameters to describe the HMM using $\theta = \{\pi, A, \phi\}$.

Expectation-Maximization (EM) algorithm usually is used to find parameters in maximum likelihood estimation with latent variables. Here we follow [1] to develop the EM algorithm. First define the objective function. Then define the posterior probability. Later on, we will find a efficient way to calculate the posterior probability (forward-backward algorithm).

Different algorithms are then used to infer the parameters using observed data. Followings will show the corresponding procedures. For this, the Rabiner's tutorial paper [2] is a good one to read.

2.2 Notation from Rabiner's paper

Here we will have many sentences are directly copied from Rabiner's paper. Note that in order to be consistent with the Rabiner's paper, we use the same notation in this tutorial paper, but they are sometime is different with those in previous section. These are $S = \{S_1, S_2, ..., S_N\}$ are used to denote the states and they are represented as $z = \{z_1, z_2, ..., z_N\}$ in previous sections. The observed data is denoted as $O_1O_2...O_T$ and is represented as $x_1, x_2, ..., x_T$ in previous sections.

- N number of states in the model. Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to sets of states of the model. For example, the speech recognition, a phoneme may be modeled as three states. We denote the individual states as $S = \{S_1, S_2, ..., S_N\}$, (**note that the state here is denoted as z_t in previous sections.**) and the state at time t as q_t.
- M the number of distinct observation symbols per state, i.e., the discrete alphabet size. The observation symbols correspond to the physical output of the system being modeled. We denote the individual symbols as $V = \{v_1, v_2, ..., v_M\}$.
- The state transition probability distribution $A = \{a_{ij}\}$, where $a_{ij} = P[q_{t+1} = S_j|q_t = S_i], 1 \leqslant i, j \leqslant N$.
- the observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where $b_j(k) = P[v_k\ at\ t|q_t = S_j]$, $1 \leqslant j \leqslant N$, $1 \leqslant k \leqslant M$.
- The initial state distribution $\pi = \pi_i$ where $\pi_i = P[q_1 = S_i]$, $1 \leqslant i \leqslant N$.

Given appropriate values of N, M, A, B, and π, the HMM can be used as a generator to give an observation sequence $O_1O_2...O_T$ (**note that the state**

here is denoted as x_t in previous sections.) , where each observation O_t is one of the symbols from V and T is the number of observations in the sequence as follow:

1. Choose an initial state $q = S_i$ according to the initial state distribution π.
2. Set $t = 1$.
3. Choose $O_t = v_k$ according to the symbol probability distribution in state S_i, i.e., $b_i(k)$.
4. Transit to a new state $q_{t+1} = S_j$ according to the state transition probability distribution for state S_j, i.e., a_{ij}.
5. Set $t = t+1$; return to step 3) if $t \leq T$; otherwise terminate the procedure.

The above procedure can be used as both a generator of observations, and as a model for how a given observation sequence was generated by an appropriate HMM.

It can be seen from the above discussion that a complete specification of an HMM requires specification of two model parameters (N and M), specification of observation symbols, and the specification of the three probability measures A, B, and π. For convenience, we use the compact notation $\lambda = \{A, B, \pi\}$ to indicate the complete parameter set of the model.

2.3 Three basic questions

Three basic questions of HMM

1. given observations $O = O_1, ..., O_T$, and model λ, how do we efficiently compute $P(O|\lambda)$, the probability of the observation sequence,given the mode?
2. given the observation sequence $O = O_1, ..., O_T$, and model λ, how do we choose a corresponding state sequence $Q = q_1 q_2 ... q_T$ which is optimal in some meaningful sense (i.e. best 'explains' the observations)?
3. How do we adjust the model parameters $\lambda = \{A, B, \pi\}$ to maximize $P(O|\lambda)$

Problem 1 is the evaluation problem, namely given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model. We can also view the problem as one of scoring how well a given model matches a given observation sequence. The latter viewpoint is extremely useful. For example, if we consider the case in which we are trying to choose among several competing models, the solution to Problem 1 allows us to choose the model which best matches the observations. (force alignment).

Problem 2 is the one in which we attempt to uncover the hidden part of the model, i.e., to find the “correct” state sequence. It should be clear that for all but the case of degenerate models, there is no "correct" state sequence to be found. Hence for practical situations, we usually use an optimality criterion to solve this problem as best as possible. Unfortunately, as we will see, there are several reasonable optimality criteria that can be imposed, and hence the choice of criterion is a strong function of the intended use for the uncovered state sequence. Typical uses might be to learn about the structure of the model, to find optimal state sequences for continuous speech recognition, or to get average statistics of individual states, etc. (decoding)

Problem 3 is the one in which we attempt to optimize the model parameters so as to best describe how a given observation sequence comes about. The observation sequence used to adjust the model parameters is called a training sequence since it is used to "train" the HMM. The training problem is the crucial one for most applications of HMMs, since it allows us to optimally adapt model parameters to observed training data, i.e., to create best models for real phenomena. (training the model)

2.4 Some thoughts

A very important question is how to connect symbol sequence (O) with state sequence Q.

There are several components in HMM. Observation, symbol, state, how those are linked?

For speech recognition, feature vector extracted from acoustic signals are observations. Corresponding uttered words are symbols. Then what are the states, and then how to link them?

It seems like the states are hidden, and we don't know. A specific state has different density distribution for each symbol given observation.

2.5 Problem one

Problem one is to find a proper way to value the probability of the observation sequence $O = O_1, ..., O_T$ given the model parameters λ. We can first simplify this by thinking a specific state path $Q = q_1 q_2 ... q_T$ for this observation sequence, which will give the conditional probability $P(O|Q, \lambda)$. If we can find the probability of state paths given the model (which can be modeled by model parameters), then we can marginalize this probability over all different paths (suppose we can enumerate all different paths) to give the probability we want, that is $P(O|\lambda)$. The probability of the observation sequence O for the specific state sequence is

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t | q_t, \lambda). \tag{7}$$

Note here we have an assumption, that observations are statistically independent conditioned on state. The probability of state paths given the model is

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} ... a_{q_{T-1} q_T}. \tag{8}$$

Then we can formulate the joint probability of O and Q given λ as

$$P(O, Q|\lambda) = P(O|Q, \lambda)P(Q, \lambda). \tag{9}$$

where $P(Q, \lambda)$ can be further factorized as $P(Q|\lambda)P(\lambda)$ and $P(\lambda)$ can be seen as a constant.

The last step is to marginalize the joint probability over all the sequences. This can be achieved by summing all the probabilities of each path by

$$P(O|\lambda) = \sum_{all\ Q} P(O, Q|\lambda) = \sum_{all\ Q} P(O|Q, \lambda)P(Q|\lambda). \tag{10}$$

This idea is simple, but it is hard to obtained as the computational burden is unfeasible [2]. Fortunately, it can be solved by the forward calculation of the forward-backward procedure (dynamic programming idea). This is formulated as follow. The idea of dynamic programming (DP) is that recursively solve a problem and using memory to reduce extra calculation. We can use DP because for each time step we only use N different states, and next step we will enter the same N element-set states.

First, we need to define a forward variable as

$$\alpha_t(i) = P(O_1O_2...O_t, q_t = S_i|\lambda) \tag{11}$$

i.e. the probability of the partial observation sequence $O_1O_2...O_t$ and state S_i at time t, given the model. It can be solved inductively as

1. Initialization: the first element is calculated as
   $\alpha_1(i) = \pi_i b_i(O_1), \ \ 1 \leqslant i \leqslant N.$
2. Induction: increment t to $t + 1$,
   $\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i)a_{ij}]b_j(O_{t+1}, \ \ 1 \leqslant t \leqslant T - 1, \ \ 1 \leqslant i \leqslant N.$
3. Termination:
   $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$

The backward calculation is similar as forward calculation by first defining a backward variable as

$$\beta_t(i) = P(O_TO_{T-1}...O_t, q_t = S_i|\lambda) \tag{12}$$

i.e. the probability of the partial observation sequence from $t + 1$ to the end and state S_i at time t, given the model. It can be solved inductively as

1. Initialization: the first element is calculated as
   $\beta_T(i) = 1, \ \ 1 \leqslant i \leqslant N.$
2. Induction: increment t to $t + 1$,
   $\beta_t(j) = \sum_{i=1}^{N} a_{ij}b_j(O_{t+1})\beta_{t+1}(j), \ \ t = T - 1, T - 2, ..., 1, \ \ 1 \leqslant i \leqslant N.$

The computation required by the forward-backward is on the order of N^2T, which is much less than enumating method.

2.6 Problem two

For problem two, we can also start with a naive method. This problem is to find an 'optimal' path to an observation sequence and model. We can define the 'optimal' path as the best state for each time step. This idea is the same with greedy decoding.

First, we define the variable $\gamma_t(i) = P(q_t = S_i|O, \lambda)$ - the probability of being in state S_i at time t, given the observation sequence and model. This can be calculated by forward and backward variables as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{13}$$

The denominator is to make $\gamma_t(i)$ a probability measure. Using $\gamma_t(i)$, we can solve for the greedy decoding as following:

$$q_t = \underset{1\leqslant i\leqslant N}{\text{argmax}}[\gamma_t(i)], \quad 1 \leqslant t \leqslant T \tag{14}$$

But the solution above may not be validate since it does not consider the whole path as an entire chain. Sometime, the transition probability from one state to another is zero, but we can find it may occur in this naive method. Then a better method is needed. The Viterbi algorithm is here to be applied.

Find the single best state sequence (path) i.e., to maximize $P(Q|O, \lambda)$ which is equivalent to maximizing $P(Q, O|\lambda)$, as $P(O, \lambda)$ can be regarded as constant if model parameters are fix.n
Viterbi Algorithm [3]: to find the single best state sequence for the given observation sequence, first we define the quantity:

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P[q_1, q_2, ..., q_{t-1}, q_t = S_i, O_1, O_2, ..., O_t|\lambda] \tag{15}$$

i.e., $\delta_t(i)$ is the best score (highest probability) along a single path, at time t, which accounts for the first t observations and ends in state S_i. By induction we have

$$\delta_{t+1}(j) = [\max_i \delta_t(i)a_{ij}] \cdot b_j(O_{t+1}) \tag{16}$$

The algorithm is expressed as

- Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{17a}$$

$$\phi_1(i) = 0. \tag{17b}$$

- Recursion:

$$\delta_t(j) = \max_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}]b_j(O_t), \quad 2 \leq t \leq T,\ 1 \leq j \leq N \tag{18a}$$

$$\phi_t(j) = \underset{1\leq i\leq N}{\text{argmax}}[\delta_{t-1}(i)a_{ij}], \quad 2 \leq t \leq T,\ 1 \leq j \leq N \tag{18b}$$

- Termination:

$$P^* = \max_{1\leq i\leq N}[\delta_T(i)], 1 \leq i \leq N \tag{19a}$$

$$q_T^* = \underset{1\leq i\leq N}{\text{argmax}}[\delta_T(i)],\ 1 \leq i \leq N \tag{19b}$$

- Path (state sequence) backtracking: $q_t^* = \phi_{t+1}(q_{t+1}^*),\ t = T-1, T-2, ..., 1$

2.7 Problem three

The third problem is by far the most difficult one. The EM or Baum-Welch algorithm is used to estimate the local optimal parameters, as there is no optimal way of estimating the model parameters.

This is accomplished by first define $\xi_t(i,j)$-the probability of being in state $S(i)$ at time t, and state $S(j)$ at time $t+1$ given the model and the observation sequence, which can be denoted as

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O,\lambda). \tag{20}$$

This can be written as

$$\xi_t(i,j) = \frac{\alpha_t(i)b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)b_j(O_{t+1}\beta_{t+1}(j))}{\sum_{i=1}^N\sum_{j=1}^N \alpha_t(i)b_j(O_{t+1}\beta_{t+1}(j))} \tag{21}$$

Again, the denominator is used to make it a probability measure. It can also be easily confirmed that $\gamma_t(i) = \sum_{j=1}^N \xi_t(i,j)$.

Then, if we sum $\gamma_t(i)$ over the time index t, we get a quantity which can be interpreted as the expected number of times that state S_i is visited. Similarly, summation of $\xi_t(i,j)$ over time can be interpreted as the expected number of transitions from state $S(i)$ to $S(j)$. By constraining the observed symbol as v_k, we can also get the $b_i(k)$.

The illustration of the third problem is not so readable yet. The combined optimization of HMM and acoustic model (e.g. GMM) is not explained here. Basically, we can see speech signal (extracted features) will be softly aligned to different states (e.g. phoneme states), and then each acoustic model for each state can be updated.

2.8 Implementation

There are many HMM implementation. A python implementation of HMM can be found in this url.

3 CTC

This section will also have many sentences that are directly borrowed from [4]. As mentioned in Section 1 underlying idea of CTC is discriminative. The power of CTC is that it does not need alignment to train a sequence-to-sequence model. By using a large amount of data, the algorithm itself consider all the possible alignments with the constrain that the sequence is left-to-right forward and gradually converge to a most likely alignment path. The posterior probability from the network $p(l|x, \theta)$ where θ is the parameter of neural network, is used inside the objective function and is directly optimized by algorithms like gradient descent algorithms.

3.1 Defining terminology

First we need to define some terminologies.

Let S be a set of training examples drawn from a fixed distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$. The input space $\mathcal{X} = (\mathbb{R}^m)^*$ is the set of all sequences of m dimensional real valued vectors. The target space $\mathcal{Z} = L^*$ is the set of all sequences over the (finite) alphabet L of labels. In general, we refer to elements of L^* as label sequences or labellings. Each example in S consists of a pair of sequences $(\mathbf{x}, \mathbf{z})$. The target sequence $z = (z_1, z_2, ..., z_U)$ is at most as long as the input sequence $x = (x_1, x_2, ..., x_T)$, i.e. $U \leqslant T$. Since the input and target sequences are not generally the same length, there is no a *priori* way of aligning them.

The aim is to use S to train a temporal classifier $h : \mathcal{X} \longmapsto \mathcal{Z}$ to classify previously unseen input sequences in a way that minimizes some task specific error measure.

For an input sequence x of length T, define a recurrent neural network with m inputs, n outputs and weight vector ω as a continuous map $\mathcal{N}_\omega$: $(\mathbb{R}^m)^T \longmapsto (\mathbb{R}^n)^T$.

Then y^t_k is interpreted as the probability of observing label k at time t, which defines a distribution over the set $L^{'T}$ of length T sequences over the alphabet $L^{'} = L \cup \{blank\}$:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t},\ \forall \pi \in L^{'T}. \tag{22}$$

3.2 Probability given a certain paths

As the length of output label sequence is not larger than the length of input, there may have several elements of input will be aligned to the same label. In order to produce repeated labels, we need to define a $< blank >$ symbol. If the previous input is aligned to a symbol z_i, we need to have a blank symbol after this input in order to generate a second symbol that is the same with z_i, i.e. to generate $(...z_i, z_i)...$, we need to have $(...z_i, < blank >, z_i...)$ in the alignment.

The next step is to define a many-to-one map $\mathcal{B} : L^{'T} \longmapsto L^{\leqslant T}$, where $L \leqslant T$ is the set of possible labellings (i.e. the set of sequences of length less than or equal to T over the original label alphabet L). We do this by simply removing all blanks and repeated labels from the paths (e.g. $\mathcal{B}(a - ab-) = \mathcal{B}(-aa - abb) = aab$). Intuitively, this corresponds to outputting a new label when the network switches from predicting no label to predicting a label, or from predicting one label to another (c.f. the CTC outputs in figure 1). Finally, we use $\mathcal{B}$ to define the conditional probability of a given labelling $l \in L^{\leqslant T}$ as the sum of the probabilities of all the paths corresponding to it:

$$p(l|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|\mathbf{x}). \tag{23}$$

3.3 Objective function and back-propagation

The objective function is the log probabilities of all the correct alignment paths. It is represented as

$$L_{ctc} = -\sum_{(x,z)\in S} log(p(\mathbf{z}|\mathbf{x})). \tag{24}$$

The Fig. 3 gives an example of the forward pass of CTC. It shows the all the different possible paths for the ground truth 'CAT'. The probability $p(\mathbf{z}|\mathbf{x})$ then model one of the particular path. Apparently, the probability of single path is modeled by the Eq. 22. This fact that Eq. 22 factorizes the joint distribution into the product of single node means that the underlying assumption is conditional independent: given the path, each node in Fig. 3 is independently distributed. This apparently not a good assumption, but it is an efficient assumption to apply. The later on proposed RNN-T [5] relaxed this assumption but with more memory consumption.

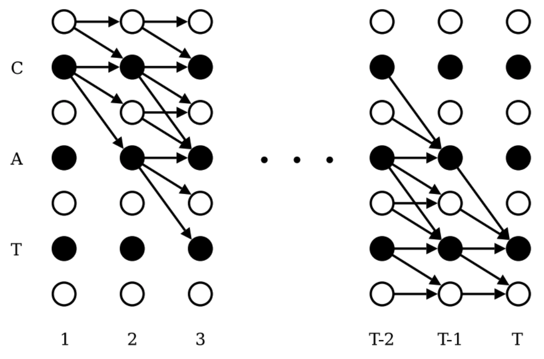

Figure 3. **illustration of the forward backward algorithm applied to the labelling 'CAT'.** Black circles represent labels, and white circles represent blanks. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated against them.

Figure 3: illustration of CTC.

The Fig. 4 intuitively illustrates the back-propogation algorithm in training the CTC loss function. The algorithm in [4] is inspired by the forward-backward algorithm used in HMM. In high level description, the forward-backward algorithm is a dynamic programming algorithm. In addition, the observation in Fig. 4 is that all the path pass over the state) $s('T', T-1)$ (the probability of observing label T at time $T-1$ can be divided into two parts. The first part is the forward path, which consist of three components: all probabilities of paths to $s('A', T-2)$ multiply the transition and emission of the red node, all probabilities of paths to $s(< blank >, T-2)$ multiply the transition and emission of the red node and all probabilities of paths to $s('T', T-2)$ multiply the transition and emission of the red node. The second part is the backward path, which consists of two components: all probabilities of paths to $s(< blank >, T)$ multiply the transition and emission of the red node and all probabilities of paths to $s('T', T)$ multiply the transition and emission of the red node.

Two further important observations are:

- the forward and backward probabilities can be calculated recursively,
- the derivative of the objective function to emission of the red node $s('T', T-1)$ is calculated by the probabilities of all paths pass over the red node. As can be seen in the previous reasoning, the multiplication of the emission of the red node makes the derivative easier. It means that the derivative is the forward probabilities and multiply the backward probabilities and divided by the emission probability. **This is illustrated by the Eq.(15) in [4]. By differently define the backward probability in Grave's thesis (Eq. 7.25), the denominator can be emitted.**

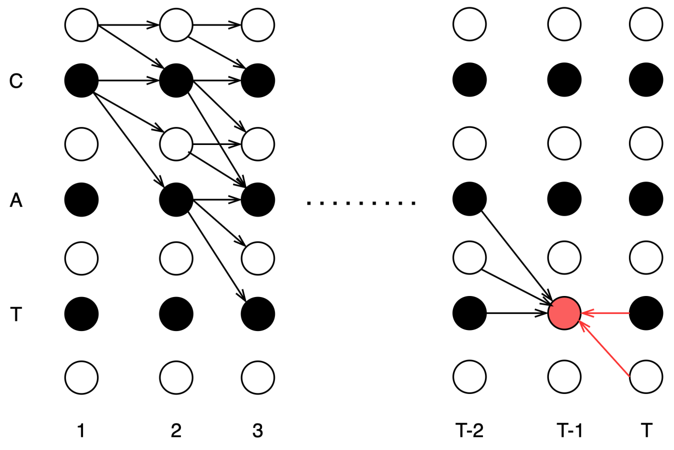

Figure 4: illustration of CTC backpropogation.

3.4 Implementation of CTC

The warp-ctc is a very good implementation. Pytorch also has a own implementation.

To further improve the algorithm (e.g. remove the blank in order to make the probabilities less spiky) and for education purpose, it is good to implement the algorithm by ourselves. I implemented the algorithm repository. It gives a python implementation and illustrate it step by step. We also have version in CUDA enabled.

4 Maximum Mutual Information criteria

Maximum mutual information (MMI) is a discriminative objective function used in automatic speech recognition. It has been investigated in [6][7]. The one with neural network version is also implemented in kaldi [8]. It also compared with CTC loss function [9]. CTC training maximizes the conditional log-likelihood of the correct transcript. The difference between CTC and MMI is that in CTC the probabilities are locally normalized but in MMI they are globally normalized. The objective function of MMI is,

$$L_{MMI} = \sum_{u=1}^{U} \log \frac{p_\lambda(x^{(u)}|M_{w^{(u)}})}{p_\lambda(x^{(u)})}. \tag{25}$$

The objective function of CTC is,

$$L_{CTC} = \sum_{u=1}^{U} \log \sum_{\pi \in \mathcal{B}^{-1}(w^{(u)})} \prod_{t=0}^{T_u-1} p(\pi_t|x^{(u)}). \tag{26}$$

It can be regarded that in eq. 26 the production and summation term is a realization of numerator in eq. 25.

5 application in ASR

This paper [10] gives very comprehensive introduction for HMM, CTC and attention encoder-decoder model.

6 use in KWS

For lattice in kaldi https://kaldi-asr.org/doc/lattices.html Kaldi use comile-train-graphs to compile text etc. to a graph then use gmm-align-compiled (or others) to get alignment.

Then we have two questions:

- how do we compile a graph dynamically during decoding with given text?

- how do we get the probability of the best path?

The second question is more or less easy. Just get the best_cost in faster-decoder. The first question is much more complicated. Above is the static graph method. gc=TrainingGraphCompiler gc.CompileGraphFromText(transcript, &decode_fst)

References

[1] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[2] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[3] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[4] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[5] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[6] L. Bahl, P. Brown, P. De Souza, and R. Mercer, "Maximum mutual information estimation of hidden markov model parameters for speech recognition," in *ICASSP'86. IEEE international conference on acoustics, speech, and signal processing*, vol. 11. IEEE, 1986, pp. 49–52.

[7] D. Povey, "Discriminative training for large vocabulary speech recognition," Ph.D. dissertation, University of Cambridge, 2005.

[8] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi." in *Interspeech*, 2016, pp. 2751–2755.

[9] H. Hadian, H. Sameti, D. Povey, and S. Khudanpur, "End-to-end speech recognition using lattice-free mmi." in *Interspeech*, 2018, pp. 12–16.

[10] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi, "Hybrid ctc/attention architecture for end-to-end speech recognition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1240–1253, 2017.