End-to-end systems 1: CTC

(Connectionist Temporal Classification)

Peter Bell

Automatic Speech Recognition – ASR Lecture 15
9 March 2020

End-to-end systems

- End-to-end systems are systems which learn to directly map from an input sequence X to an output sequence Y, estimating $P(Y|X)$
  - Y can be a sequence of words or subwords
- ML trained HMMs are kind of end-to-end system – the HMM estimates $P(X|Y)$, and when combined with a language model gives an estimate of $P(Y|X)$
- Sequence discriminative training of HMMs (using GMMs or DNNs) can be regarded as end-to-end
  - But training is quite complicated – need to estimate the denominator (total likelihood) using lattices, first train conventionally (ML for GMMs, CE for NNs) then finetune using sequence discriminative training
  - Lattice-free MMI is one way to address these issues

Fully differentiable end-to-end systems

Approaches based purely on recurrent networks which directly map input to output sequences

- CTC – Connectionist Temporal Classification
- Encoder-decoder approaches

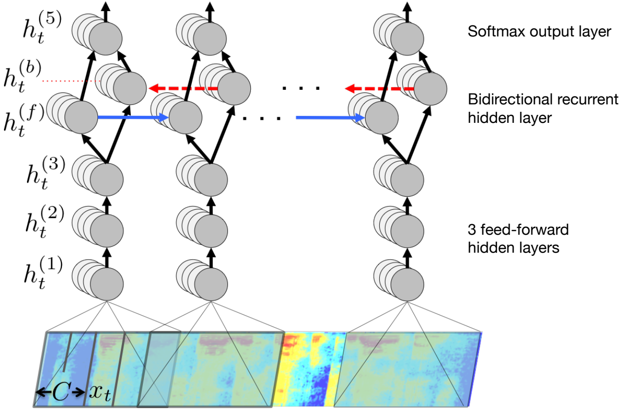
Fully differentiable end-to-end systems

Approaches based purely on recurrent networks which directly map input to output sequences

- **CTC – Connectionist Temporal Classification**
- Encoder-decoder approaches (*next lecture*)

Example: Deep Speech

Output: character probabilities (a-z, <apostrophe>, <space>, <blank>)
Trained using CTC



Input: Filter bank features (spectrogram)

Hannun et al (2014), “Deep Speech: Scaling up end-to-end speech recognition”,

Deep Speech: Results

Model	SWB	CH	Full
Vesely et al. (GMM-HMM BMMI) [44]	18.6	33.0	25.8
Vesely et al. (DNN-HMM sMBR) [44]	12.6	24.1	18.4
Maas et al. (DNN-HMM SWB) [28]	14.6	26.3	20.5
Maas et al. (DNN-HMM FSH) [28]	16.0	23.7	19.9
Seide et al. (CD-DNN) [39]	16.1	n/a	n/a
Kingsbury et al. (DNN-HMM sMBR HF) [22]	13.3	n/a	n/a
Sainath et al. (CNN-HMM) [36]	11.5	n/a	n/a
Soltau et al. (MLP/CNN+I-Vector) [40]	**10.4**	n/a	n/a
Deep Speech SWB	20.0	31.8	25.9
Deep Speech SWB + FSH	12.6	**19.3**	**16.0**

Table 3: Published error rates (%WER) on Switchboard dataset splits. The columns labeled "SWB" and "CH" are respectively the easy and hard subsets of Hub5'00.

Deep Speech Training

- Maps from acoustic frames $\boldsymbol{X}$ to subword sequences $\boldsymbol{S}$, where $\boldsymbol{S}$ is a sequence of characters (in some other CTC approaches, $\boldsymbol{S}$ can be a sequence of phones)
- CTC loss function
- Makes good use of large training data
    - Synthetic additional training data by jittering the signal and adding noise
- Many computational optimisations
- n-gram language model to impose word-level constraints
- Competitive results on standard tasks

Deep Speech Training

- Maps from acoustic frames $\boldsymbol{X}$ to subword sequences $\boldsymbol{S}$, where $\boldsymbol{S}$ is a sequence of characters (in some other CTC approaches, $\boldsymbol{S}$ can be a sequence of phones)
- **CTC loss function**
- Makes good use of large training data
  - Synthetic additional training data by jittering the signal and adding noise
- Many computational optimisations
- n-gram language model to impose word-level constraints
- Competitive results on standard tasks

Connectionist Temporal Classification (CTC)

- Train a recurrent network to map from input sequence $\boldsymbol{X}$ to output sequence $\boldsymbol{S}$
  - sequences can be different lengths – for speech, input sequence X (acoustic frames) is much longer than output sequence S (characters or phonemes)
  - CTC does not require frame-level alignment (matching each input frame to an output token)
- CTC sums over all possible alignments (similar to forward-backward algorithm) – "alignment free"
- Possible to back-propagate gradients through CTC loss function

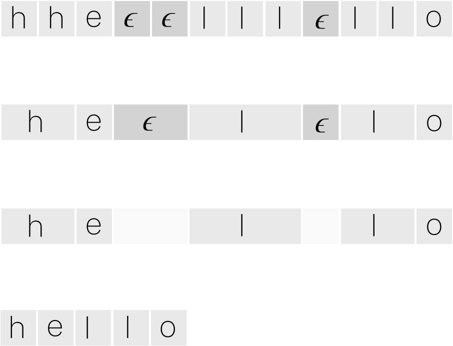
Gopod overview of CTC: Awni Hannun, "Sequence Modeling with CTC", *Distill*. https://distill.pub/2017/ctc

CTC: Alignment

- Imagine mapping $(x_1, x_2, x_3, x_4, x_5, x_6)$ to $[a, b, c]$
  - Possible alignments: $aaabbc$, $aabbcc$, $abbbbc$, ...
- However
  - Don't always want to map every input frame to an output symbol (e.g. if there is "inter-symbol silence")
  - Want to be able to have two identical symbols adjacent to each other – keep the difference between
- Solve this using an additional *blank* symbol (ϵ)
- CTC output compression
  1. Merge repeating characters
  2. Remove blanks

  Thus to model the same character successively, separate with a blank
- Some possible alignments for $[h, e, l, l, o]$ and $[h, e, l, o]$ given a 10-element input sequence
  - $[h, e, l, l, o]$: $h\epsilon\epsilon e\epsilon ll\epsilon lo$; $he\epsilon ll\epsilon l\epsilon oo$
  - $[h, e, l, o]$: $h\epsilon\epsilon e\epsilon llllo$; $hh\epsilon e\epsilon l\epsilon\epsilon o\epsilon$

CTC: Alignment example



First, merge repeat characters.

Then, remove any ϵ tokens.

The remaining characters are the output.

CTC: Valid and invalid alignments

Consider an output [c, a, t] with an input of length six

Valid Alignments

ϵ	c	c	ϵ	a	t
c	c	a	a	t	t
c	a	ϵ	ϵ	ϵ	t

Invalid Alignments

c	ϵ	c	ϵ	a	t		corresponds to $Y =$ [c, c, a, t]
c	c	a	a	t			has length 5
c	ϵ	ϵ	ϵ	t	t		missing the 'a'

CTC: Alignment properties

- Monotonic – Alignments are monotonic (left-to-right model); no re-ordering (unlike neural machine translation)
- Many-to-one – Alignments are many-to-one; many inputs can map to the same output
- But a single input cannot map to many outputs – could be a problem for sounds like “th”
- CTC doesn’t find a single alignment: it sums over all possible alignments

CTC: Loss function (1)

- Let $\boldsymbol{C}$ be an output label sequence, including blanks and repetitions – same length as input sequence $\boldsymbol{X}$
- Posterior probability of output labels $\boldsymbol{C} = (c_1, \ldots c_t, \ldots c_T)$ given the input sequence $\boldsymbol{X} = (x_1, \ldots x_t, \ldots x_T)$:

$$P(\boldsymbol{C}|\boldsymbol{X}) = \prod_{t=1}^{T} P_t(c_t|\boldsymbol{X})$$

  where $y(c_t, t)$ is the output for label c_t at time t
- This is the probability of a single alignment – we need to sum over all alignments consistent with $\boldsymbol{S}$

CTC: Loss function (2)

- Let $\boldsymbol{S}$ be the compressed target output sequence
- Compute the posterior probability of the target sequence $\boldsymbol{S} = (s_1, \ldots s_m, \ldots s_M)$ $(M \leq T)$ given $\boldsymbol{X}$ by summing over the possible CTC alignments:

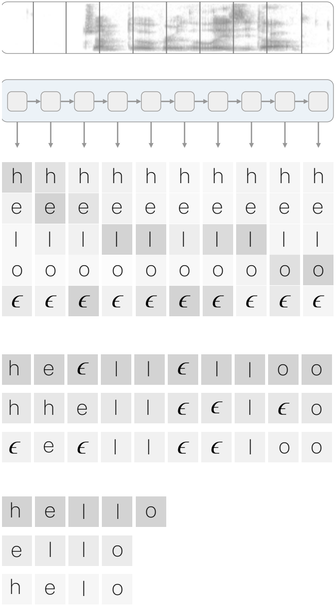
$$P(\boldsymbol{S}|\boldsymbol{X}) = \sum_{\boldsymbol{c} \in A(\boldsymbol{S})} P(\boldsymbol{C}|\boldsymbol{X})$$

  where A is the set of possible output label sequences $\boldsymbol{c}$ that can be mapped to $\boldsymbol{S}$ using the CTC compression rules (merge repeated labels, then remove blanks)
- The CTC loss function $\mathcal{L}_{CTC}$ is given by the negative log likelihood of the sum of CTC alignments:

$$\mathcal{L}_{CTC} = -\log P(\boldsymbol{S}|\boldsymbol{X})$$

- Various NN architectures can be used for CTC – usually use a deep bidirectional LSTM RNN

CTC: Distribution over alignments



We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives $p_t(a \mid X)$, a distribution over the outputs $\{h, e, l, o, \epsilon\}$ for each input step.

With the per time-step output distribution, we compute the probability of different sequences

By marginalizing over alignments, we get a distribution over outputs.

CTC: Dynamic programming

Perform the sum over alignments, $A(\boldsymbol{S})$, using dynamic programming – very similar to the forward algorithm for classic HMMs.

We first define the expanded symbol sequence,
$Z = (z_1, \ldots, z_i, \ldots, z_J) = (\epsilon, s_1, \epsilon, s_2, \epsilon, \ldots, \epsilon, s_M, \epsilon)$
(where $J = 2M + 1$)

The forward probability is:

$$\begin{aligned}\alpha_j(t) &= P(z_1, \ldots z_j | X) \\ &= \sum_{(c_1,\ldots,c_t) \in A(z_1,\ldots,z_j)} P(c_1, \ldots, c_t | \boldsymbol{X})\end{aligned}$$

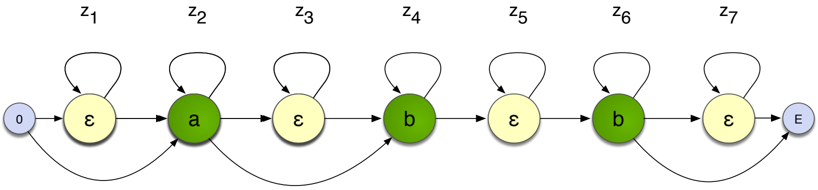
This computes the probability over all label sequences up to time t that are consistent with $(z_1, \ldots z_j)$.

CTC: HMM topology

We can encode the valid transitions of Z over time using an HMM.

This is a standard left-to-right HMM topogy, with the addition of a skip $z_{i-2} \rightarrow z_i$ if $z_i \neq \epsilon$ and $z_i \neq z_{i-2}$

Example for original sequence $\boldsymbol{S} = [a, b, b]$:



CTC: Forward recursion

- **Initialisation:**

$$\alpha_i(0) = 1 \qquad i = 1$$
$$= 0 \qquad \textit{otherwise}$$

- **Recursion:**
  If $z_i = \epsilon$ or $z_i = z_{i-2}$:

$$\alpha_i(t) = \left[\alpha_{i-1}(t-1) + \alpha_i(t-1)\right] p_t(z_i|X)$$

  Otherwise:

$$\alpha_i(t) = \left[\alpha_{i-2}(t-1) + \alpha_{i-1}(t-1) + \alpha_i(t-1)\right] p_t(z_i|X)$$

- **Termination:**

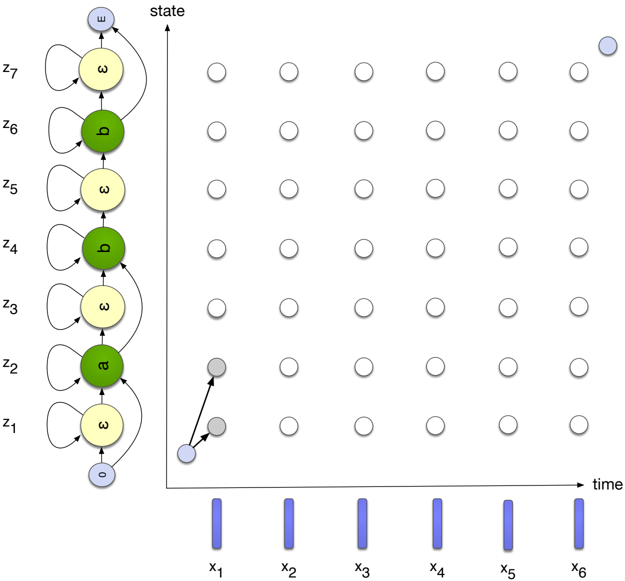
$$P(Z|X) = \alpha_{J-1}(t) + \alpha_J(t)$$

Example

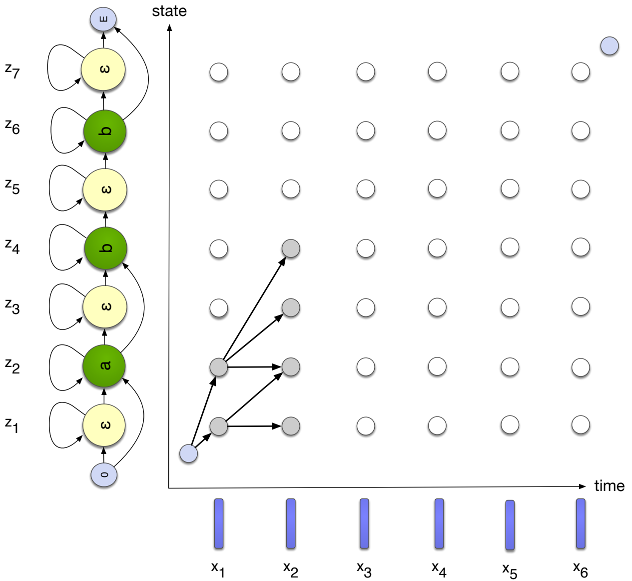
Example alignments for [a, b, b] to an utterance of six frames:

a	ε	b	ε	b	b
a	a	ε	b	ε	b
ε	a	a	b	ε	b
ε	a	b	ε	b	ε
a	b	ε	ε	b	b
⋮	⋮	⋮			

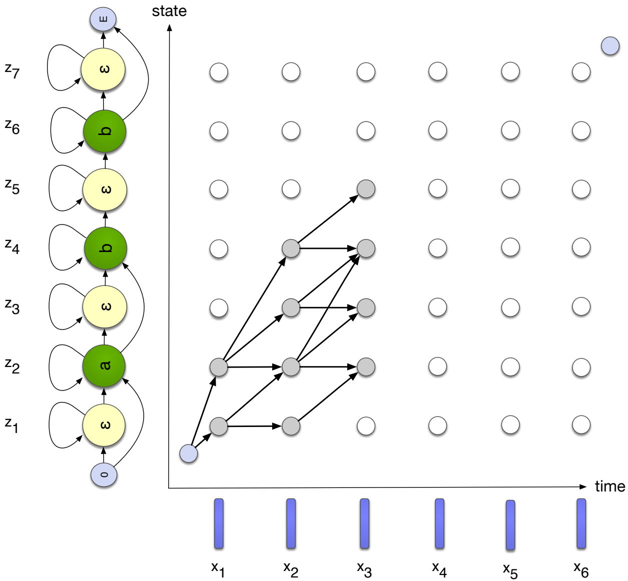
Forward recursion



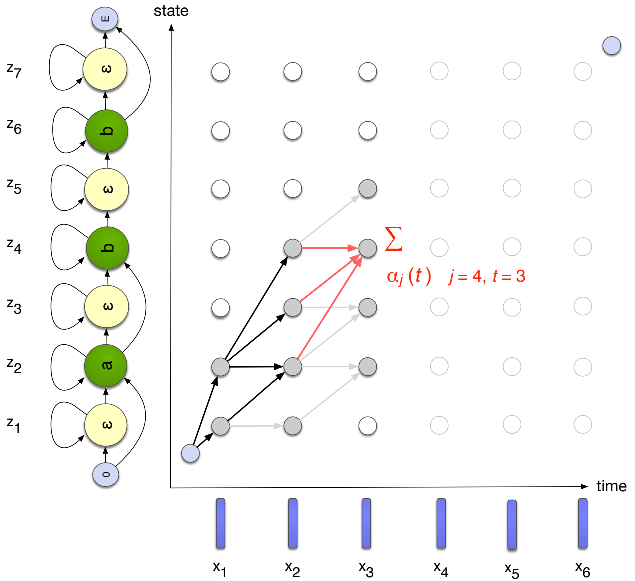
Forward recursion



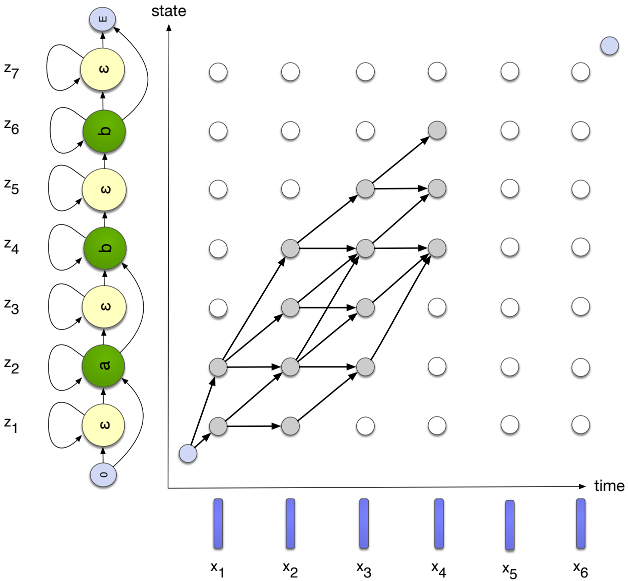
Forward recursion



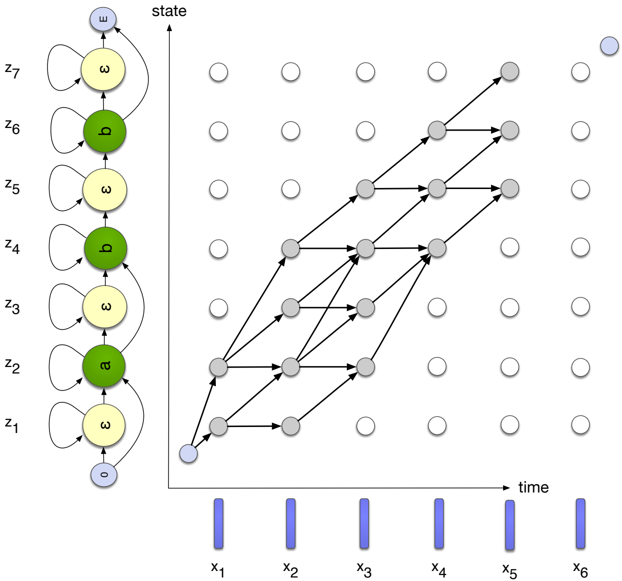
Forward recursion



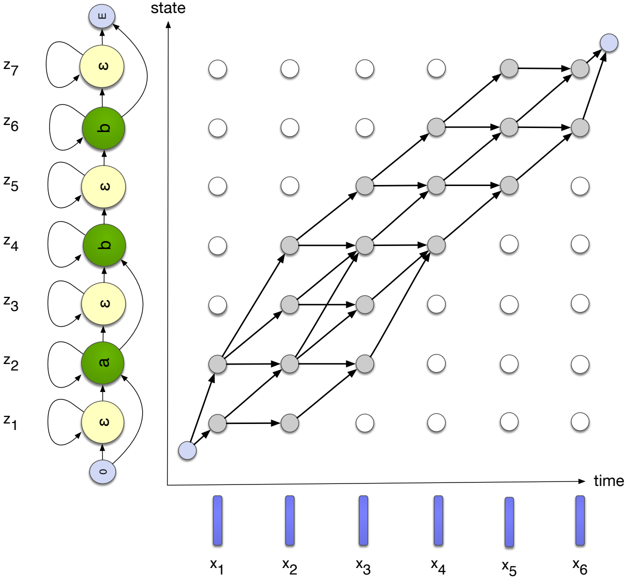
Forward recursion



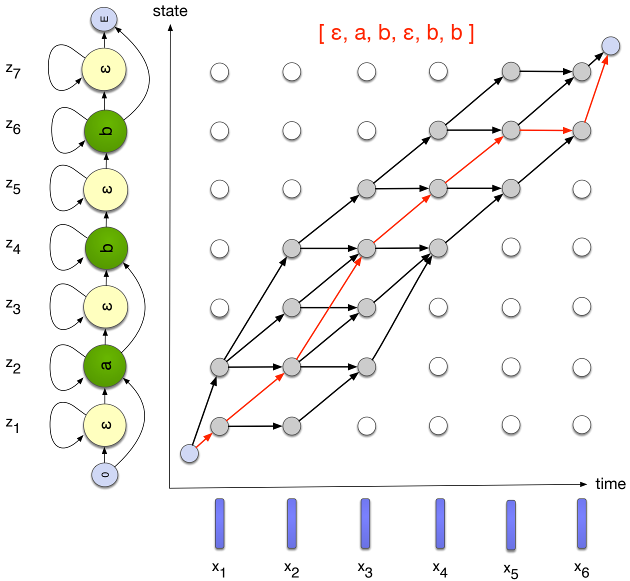
Forward recursion



Forward recursion



One alignment...



CTC: Decoding

Need to solve

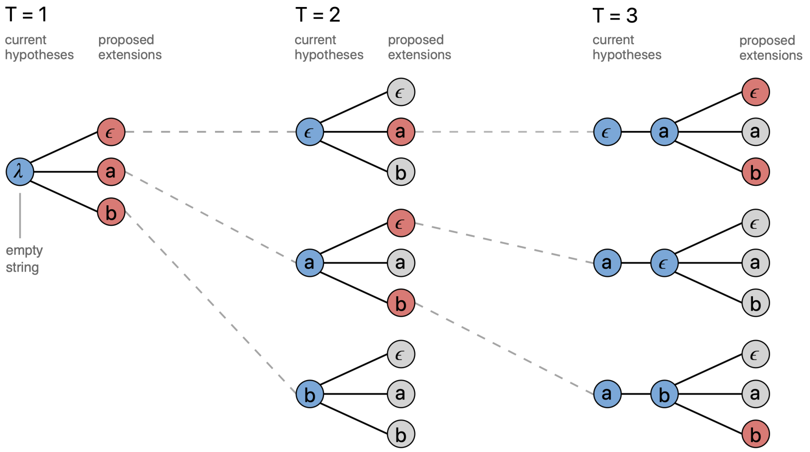
$$\boldsymbol{S}^* = \arg\max_{\boldsymbol{S}} P(\boldsymbol{S}|\boldsymbol{X})$$

Find best alignment:

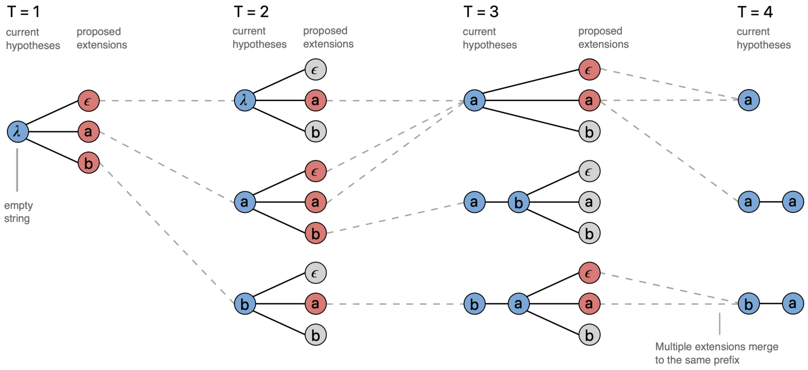
$$\boldsymbol{C}^* = \arg\max_{\boldsymbol{C}} \prod_{t}^{T} P(c_t|X)$$

Solve using beam search

CTC: Decoding with beam search

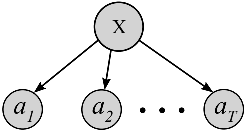


Merge hypotheses



Understanding CTC: Conditional independence assumption

- Each output is dependent on the entire input sequence (in Deep Speech this is achieved using a bidirectional recurrent layer)
- Given the inputs, each output is independent of the other outputs (conditional independence)
- CTC does not learn a language model over the outputs, although a language model can be applied later
- Graphical model showing dependences in CTC:



Applying language models to CTC

- Direct interpolation of a language model with the CTC acoustic model:

$$\hat{\boldsymbol{W}} = \arg\max_{W} (\log P(\boldsymbol{S}|\boldsymbol{X}) + \lambda \log P(W)) + \eta\beta L(W))$$

  Only consider word sequences W which correspond to the subword sequence $\boldsymbol{S}$ (using a lexicon)
- λ,η are empirically determined scaling factor/insertion bonus
- Lexicon-free CTC: use a "subword language model" $P(\boldsymbol{S})$ (Maas et al, 2015)
- WFST implementation: create an FST T which transforms a framewise label sequence $\boldsymbol{c}$ into the subword sequence $\boldsymbol{S}$, then compose with L and G: $T \circ \min(\det(L \circ G))$ (Miao et al, 2015)

Mozilla Deep Speech

- Mozilla have released an Open Source TensorFlow implementation of the Deep Speech architecture:
- https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/
- https://github.com/mozilla/DeepSpeech
- Close to state-of-the-art results on librispeech
- Mozilla Common Voice project: https://voice.mozilla.org/en

Summary and reading

- CTC is an alternative approach to sequence discriminative training, typically applied to RNN systems
- Used in "Deep Speech" architecture for end-to-end speech recognition
- Reading
  - A Hannun et al (2014), "Deep Speech: Scaling up end-to-end speech recognition", ArXiV:1412.5567. https://arxiv.org/abs/1412.5567
  - A Hannun (2017), "Sequence Modeling with CTC", *Distill*. https://distill.pub/2017/ctc
- Background reading
  - Y Miao et al (2015), "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding", ASRU-2105. https://ieeexplore.ieee.org/abstract/document/7404790
  - A Maas et al (2015). "Lexicon-free conversational speech recognition with neural networks", NAACL HLT 2015, http://www.aclweb.org/anthology/N15-1038