End-to-end systems: Deep Speech and CTC

Steve Renals

Automatic Speech Recognition – ASR Lecture 15
5 April 2018
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 $P(Y|X)$.

- **ML trained HMMs** are kind of end-to-end system – the HMM estimates $P(X|Y)$ but 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.

- **Other approaches** based on recurrent networks which directly map input to output sequences:
  - **CTC** – Connectionist Temporal Classification
  - **Encoder-decoder approaches**
Deep Speech

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

Input: Filter bank features (spectrogram)

Hannun et al, “Deep Speech: Scaling up end-to-end speech recognition”,
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 to character sequences
- 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 to character sequences
- **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 $X$ to output sequence $Y$
  - sequences can be different lengths
  - frame-level alignment (matching each input frame to an output token) not required
- CTC sums over all possible alignments (similar to forward-backward algorithm) – ”alignment free”
- Possible to back-propagate gradients through CTC

This presentation of CTC based on 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: \(aaabbc, aabbcc, abbbbc, \ldots\)
- However
  - Don’t always want to map every input frame to an output symbol (e.g. silence)
  - Want to be able to have two identical symbols adjacent to each other e.g. \([h, e, l, l, o]\)
- Solve this with an additional blank symbol (\(\epsilon\))
  - Blanks removed from the output sequence
  - To model the same character in a row, separate with a blank
CTC: Alignment example

First, merge repeat characters.

Then, remove any $\epsilon$ tokens.

The remaining characters are the output.

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

**Valid Alignments**

- $\epsilon \ c \ c \ \epsilon \ a \ t$
- $c \ c \ a \ a \ a \ t \ t$
- $c \ a \ a \ \epsilon \ \epsilon \ \epsilon \ t$

**Invalid Alignments**

- $c \ c \ \epsilon \ c \ \epsilon \ a \ t$
- $c \ c \ a \ a \ a \ a \ t$
- $c \ c \ \epsilon \ \epsilon \ \epsilon \ \epsilon \ t \ t$

- corresponds to $Y = [c, c, a, t]$
- has length 5
- 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 (however many outputs cannot map to a single input)
- CTC doesn’t find a single alignment, sums over all possible alignments
CTC: Loss function

\[ P(Y|X) = \sum_A \prod_t p(a_t|X) \]

Estimate using an RNN
Sum over alignments using dynamic programming – similar structure as used in forward-backward algorithm and Viterbi
We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives \( p_t(a | X) \), 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: Valid paths

Two final nodes

End-to-end systems: Deep Speech and CTC
CTC: Allowed transitions

No skip transition allowed
Previous token in output seq
OR blank between repeat symbols

Skip transition allowed
Previous token is a blank between different symbols
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:
CTC can be interpreted as an HMM with additional (skippable) blank states, trained discriminatively.
Mozilla have released an Open Source TensorFlow implementation of the Deep Speech architecture:

- https://github.com/mozilla/DeepSpeech

Close to state-of-the-art results on librispeech

Mozilla Common Voice project: https://voice.mozilla.org/en
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

https://arxiv.org/abs/1412.5567

https://distill.pub/2017/ctc