Speech Recognition and Processing

Lecture 8

Yossi Keshet
The CTC loss function
\( Y = \) jumps over the lazy dog also denoted as \( \bar{p} \)

\( X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & \cdots & x_{n-1} & x_T \end{pmatrix} \) also denoted as \( \bar{o} \)

**Speech recognition:** The input can be a spectrogram or some other frequency based feature extractor.
alignment

\[ A = \begin{bmatrix} j & j & j & u & u & m & \ldots & g & g \end{bmatrix} \]

also denoted as \( \tilde{q} \)

\[ Y = \text{jumps over the lazy dog} \]

also denoted as \( \tilde{p} \)

\[ X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & \ldots & x_{T-1} & x_T \end{bmatrix} \]

also denoted as \( \tilde{d} \)

\[ \text{Speech recognition: The input can be a spectrogram or some other frequency based feature extractor.} \]
Speech recognition: The input can be a spectrogram or some other frequency based feature extractor.
Take $X = (x_1, \ldots, x_T)$, $Y = (y_1, \ldots, y_L) \rightarrow$ alignment algorithms $\rightarrow A = (a_1, \ldots, a_T)$

Train a network that works for each frame: gets $x_t$ and predicts $a_t$

CTC works differently. For a given $X$ it gives us an output distribution over all possible alignments of $Y$: $p(Y|X)$

We can use this distribution either to infer a likely output or to assess the probability of a given output.

This is also a loss function. Who using this loss we would like to minimize

$$\sum_{i=1}^{m} - \log p(Y_i | X_i)$$
The same concepts can be used for OCR

The quick brown fox

Handwriting recognition: The input can be \((x, y)\) coordinates of a pen stroke or pixels in an image.
Connectionist Temporal Classification (CTC)

The CTC algorithm is alignment-free: it doesn’t require an alignment between the input and the output. However, to get the probability of an output given an input, CTC works by summing over the probability of all possible alignments between the two.

\[
\begin{array}{cccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\
  \text{alignment} & & & & & \\
  c & c & a & a & a & t \\
  \text{input (X)} & & & & & \\
  c & a & a & t \\
  \text{output (Y)} & & & & & 
\end{array}
\]
Connectionist Temporal Classification (CTC)

CTC introduces a new token to the set of allowed outputs. This new token is sometimes called the blank token. We’ll refer to it here as $\epsilon$.

The $\epsilon$ token doesn’t correspond to anything and is simply removed from the output.

The alignments allowed by CTC are the same length as the input.

We allow any alignment which maps to $Y$ after merging repeats and removing $\epsilon$ tokens:

First, merge repeat characters.

Then, remove any $\epsilon$ tokens.

The remaining characters are the output.
Connectionist Temporal Classification (CTC)

We start with an input sequence, like a spectrogram of audio.
Connectionist Temporal Classification (CTC)

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

The input is fed into an RNN, for example.
Connectionist Temporal Classification (CTC)

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.
Connectionist Temporal Classification (CTC)

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

The input is fed into an RNN, for example.
Connectionist Temporal Classification (CTC)

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

The input is fed into an RNN, for example.

With the per time-step output distribution, we compute the probability of different sequences.
We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.
Connectionist Temporal Classification (CTC)

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

The input is fed into an RNN, for example.

By marginalizing over alignments, we get a distribution over outputs.
Connectionist Temporal Classification (CTC)

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

The input is fed into an RNN, for example.

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

\[ p(Y | X) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T} p_t(a_t | X) \]

The CTC conditional probability marginalizes over the set of valid alignments computing the probability for a single alignment step-by-step.
Connectionist Temporal Classification (CTC)

Below is an example of the computation performed by the dynamic programming algorithm. Every valid alignment has a path in this graph.

- **Input:** $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
- **Output:** $Y = \{a, b, c\}$

Node $(s, t)$ in the diagram represents $\alpha_{s,t}$ – the CTC score of the subsequence $Z_{1:s}$ after $t$ input steps.
Connectionist Temporal Classification (CTC)

Below is an example of the computation performed by the dynamic programming algorithm. **Every valid alignment has a path in this graph.**

Node \((s, t)\) in the diagram represents \(\alpha_{s,t}\) – the CTC score of the subsequence \(Z_{1:s}\) after \(t\) input steps.
Connectionist Temporal Classification (CTC)
Connectionist Temporal Classification (CTC)

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:

\[ Y^* = \arg \max_Y p(Y | X) \]

One heuristic is to take the most likely output at each time-step. This gives us the alignment with the highest probability:
Connectionist Temporal Classification (CTC)

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:

\[ Y^* = \underset{Y}{\text{arg max}} \ p(Y \mid X) \]

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

\[ A^* = \underset{A}{\text{arg max}} \ \prod_{t=1}^{T} p_t(a_t \mid X) \]
Connectionist Temporal Classification (CTC)

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:

\[ Y^* = \arg \max_Y p(Y | X) \]

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

\[ A^* = \arg \max_A \prod_{t=1}^{T} p_t(a_t | X) \]

We can then collapse repeats and remove \( \epsilon \) tokens to get \( Y \).
Connectionist Temporal Classification (CTC)

However, this approach can sometimes miss easy to find outputs with much higher probability.

The problem is, it doesn’t take into account the fact that a single output can have many alignments.

For example:

\[ [b,b,b] \rightarrow 0.3 \]
\[ [a,a,a] \rightarrow 0.15 \]
\[ [a,a,\epsilon] \rightarrow 0.25 \]

\( Y=[b] \) is the most likely hypothesis (0.3)
It should have chosen \( Y=[a] \) which has overall probability of 0.15+0.25=0.4
A regular beam search computes a new set of hypotheses at each input step. The new set of hypotheses is generated from the previous set by extending each hypothesis with all possible output characters and keeping only the top candidates.

A standard beam search algorithm with an alphabet of \( \{ \epsilon, a, b \} \) and a beam size of three.
We can modify the vanilla beam search to handle multiple alignments mapping to the same output.

Instead of keeping a list of alignments in the beam, we store the output prefixes after collapsing repeats and removing $\epsilon$ characters.

At each step of the search we accumulate scores for a given prefix based on all the alignments which map to it.

The CTC beam search algorithm with an output alphabet $\{\epsilon, a, b\}$ and a beam size of three.
Connectionist Temporal Classification (CTC)

We can modify the vanilla beam search to handle multiple alignments mapping to the same output.

Instead of keeping a list of alignments in the beam, we store the output prefixes after collapsing repeats and removing $\epsilon$ characters.

At each step of the search we accumulate scores for a given prefix based on all the alignments which map to it.

The CTC beam search algorithm with an output alphabet $\{\epsilon, a, b\}$ and a beam size of three.
Remember, the $\epsilon$ is required between repeat characters.

We have to keep track of two probabilities for each prefix in the beam. The probability of all alignments which end in $\epsilon$ and the probability of all alignments which don’t end in $\epsilon$.

When we rank the hypotheses at each step before pruning the beam, we’ll use their combined scores.
Deep Speech 2
Word Error Rate (WER)

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]
Word Error Rate (WER)

- We can not use standard multi-class evaluation techniques

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]
• We can not use standard multi-class evaluation techniques

• We define the Word Error Rate (WER) metric:

\[
Wer = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]
• We cannot use standard multi-class evaluation techniques

• We define the Word Error Rate (WER) metric:

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]

• Substitutions (S)
Word Error Rate (WER)

- We can not use standard multi-class evaluation techniques

- We define the Word Error Rate (WER) metric:

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]

- Substitutions (S)

- Deletions (D)
Word Error Rate (WER)

- We can not use standard multi-class evaluation techniques

- We define the Word Error Rate (WER) metric:

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]

- Substitutions (S)
- Deletions (D)
- Insertions (I)
We cannot use standard multi-class evaluation techniques.

We define the Word Error Rate (WER) metric:

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]

- Substitutions (S)
- Deletions (D)
- Insertions (I)
- Corrects (C)
We can not use standard multi-class evaluation techniques

We define the Word Error Rate (WER) metric:

\[
WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}
\]

- Substitutions (S)
- Deletions (D)
- Insertions (I)
- Corrects (C)
- The number of words in the reference (N)
## Word Error Rate (WER)

<table>
<thead>
<tr>
<th>Type</th>
<th>REF: What a <strong>bright</strong> day</th>
<th>HYP: What a <strong>day</strong></th>
<th>WER = 1/4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deletion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insertion</strong></td>
<td>REF: What a <strong>day</strong></td>
<td>HYP: What a <strong>bright</strong> day</td>
<td>WER = 1/3</td>
</tr>
<tr>
<td><strong>substitution</strong></td>
<td>REF: What a <strong>bright</strong> day</td>
<td>HYP: What a <strong>light</strong> day</td>
<td>WER = 1/4</td>
</tr>
</tbody>
</table>

Note that WER can be greater than 1.
The network is designed to minimize the following loss:

$$\mathcal{L}(x, y; \theta) = -\log \sum_{\ell \in \text{Align}(x, y)} \prod_{t} p_{\text{ctc}}(\ell_t|x; \theta)$$

where \(\text{Align}(x, y)\) is the set of all possible alignments of the characters of the transcription \(y\) to frames of input \(x\) under the CTC operator. \(\ell_t\) is either a character in the alphabet or the blank symbol. In English we have \(\ell_t \in \{a, b, c, \ldots, z, \text{space}, \text{apostrophe}, \epsilon\}\), where space and apostrophe denote word boundaries.
**Deep Speech 2 - training**

**Data**: English (3,600 hours) and Mandarin (1,400 hours) datasets.

They were created from raw data captured as long audio clips with noisy transcriptions.

Duration — from several minutes to more than hour —→ impractical to unroll them in time in the RNN during training.

To solve this problem, the authors developed an alignment, segmentation and filtering pipeline that can generate a training set with shorter utterances and few erroneous transcriptions.
Deep Speech 2 - training

I. Alignment

Use an existing bidirectional RNN model trained with CTC to align the transcription to the frames of audio. For each pair \((x, y)\) find the alignment that maximizes:

\[
\ell^* = \arg \max_{\ell \in \text{Align}(x, y)} \prod_{t} p_{\text{ctc}}(\ell_t | x; \theta)
\]

II. Segmentation

Splice the audio whenever it encounters a long series of consecutive blank labels occurs, since this usually denotes a stretch of silence.

III. Filtering

Remove erroneous examples that arise from a failed alignment, if WER between the ground truth and the aligned transcription is greater than 5%.
Deep Speech 2 - data augmentation

Augmentation: noise synthesis, reverb, time-stretching, pitch-shifting.

The noise is added to 40% of the utterances that are chosen at random.
Deep Speech 2 - performance

<table>
<thead>
<tr>
<th>Read Speech</th>
<th>DS1</th>
<th>DS2</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ eval’92</td>
<td>4.94</td>
<td>3.60</td>
<td>5.03</td>
</tr>
<tr>
<td>WSJ eval’93</td>
<td>6.94</td>
<td>4.98</td>
<td>8.08</td>
</tr>
<tr>
<td>LibriSpeech test-clean</td>
<td>7.89</td>
<td>5.33</td>
<td>5.83</td>
</tr>
<tr>
<td>LibriSpeech test-other</td>
<td>21.74</td>
<td>13.25</td>
<td>12.69</td>
</tr>
</tbody>
</table>

Table 13: Comparison of WER for two speech systems and human level performance on read speech.

<table>
<thead>
<tr>
<th>Accented Speech</th>
<th>DS1</th>
<th>DS2</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoxForge American-Canadian</td>
<td>15.01</td>
<td>7.55</td>
<td>4.85</td>
</tr>
<tr>
<td>VoxForge Commonwealth</td>
<td>28.46</td>
<td>13.56</td>
<td>8.15</td>
</tr>
<tr>
<td>VoxForge European</td>
<td>31.20</td>
<td>17.55</td>
<td>12.76</td>
</tr>
<tr>
<td>VoxForge Indian</td>
<td>45.35</td>
<td>22.44</td>
<td>22.15</td>
</tr>
</tbody>
</table>

Table 14: Comparing WER of the DS1 system to the DS2 system on accented speech.
Wav2Letter
The last two layers are equivalent to fully connected layers.

The first two layers are used when the input is the raw wave. Power spectrum and MFCC based networks do not have the first layer.
Wav2Letter - extension to CTC

Graph which represents all the acceptable sequences of letters (black denoted as $\emptyset$)

\[ \mathcal{L}(x, y; \theta) = - \log \sum_{\ell \in \text{Align}(x, y)} \prod_{t=1}^{T} p_{\text{cln}}(\ell_t | x; \theta) \]

The same graph unfolded over 5 frames. There are no transitions scores. At each time step, nodes are assigned a conditional probability output by the neural network acoustic model.
Wav2Letter - extension to CTC

• Blank complicates the search graph

• Output is computationally independent

• Idea:
  • Remove the black note, and use numbers for repetitions
  • Add another graph to learn transition probability
Wav2Letter - extension to CTC

All acceptable sequences of letters for the transcription “cat”

Shows the corresponding fully connected graph, which describes all possible sequences of letter; this graph is used for normalization purposes.

\[ \mathcal{L}(x, y; \theta) = -\log \sum_{\pi_t \in \mathcal{G}_{\text{ASG}}} \prod_{t} p(\pi_t | x; \theta) + p(\pi_t, \pi_{t-1} | x; \theta) + \log \sum_{\pi \in \mathcal{G}_{\text{full}}} \prod_{t} p(\pi_t | x; \theta) + p(\pi_t, \pi_{t-1} | x; \theta) \]
Wav2Letter - extension to CTC

All acceptable sequences of letters for the transcription “cat”

Shows the corresponding fully connected graph, which describe all possible sequences of letter; this graph is used for normalization purposes.

Shows the same graph unfolded over 5 frames

\[
L(x, y; \theta) = -\log \sum_{p_i \in \mathcal{G}_{ASG}} \prod_t p(\pi_t | x; \theta) + p(\pi_t, \pi_{t-1} | x; \theta) + \log \sum_{\pi \in \mathcal{G}_{full}} \prod_t p(\pi_t | x; \theta) + p(\pi_t, \pi_{t-1} | x; \theta)
\]

Used in training for normalization
Wav2Letter - performance

<table>
<thead>
<tr>
<th></th>
<th>ASG</th>
<th>CTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev-clean</td>
<td>10.4</td>
<td>10.7</td>
</tr>
<tr>
<td>test-clean</td>
<td>10.1</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>batch size</th>
<th>CTC</th>
<th>ASG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>1</td>
<td>1.9</td>
<td>5.9</td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
<td>6.0</td>
</tr>
<tr>
<td>8</td>
<td>2.0</td>
<td>6.1</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>batch size</th>
<th>CTC</th>
<th>ASG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>1</td>
<td>40.9</td>
<td>97.9</td>
</tr>
<tr>
<td>4</td>
<td>41.6</td>
<td>99.6</td>
</tr>
<tr>
<td>8</td>
<td>41.7</td>
<td>100.3</td>
</tr>
</tbody>
</table>

(c)

Table 1: CTC vs ASG. CTC is Baidu’s implementation. ASG is implemented on CPU (core in C, threading in Lua). (a) reports performance in LER. Timings (in ms) for small sequences (input frames: 150, letter vocabulary size: 28, transcription size: 40) and long sequences (input frames: 700, letter vocabulary size: 28, transcription size: 200) are reported in (b) and (c) respectively. Timings include both forward and backward passes. CPU implementations use 8 threads.
Figure 4: Valid LER (a) and WER (b) v.s. training set size (10h, 100h, 200h, 1000h). This compares MFCC-based and power spectrum-based (POW) architectures. AUG experiments include data augmentation. In (b) we provide Baidu Deep Speech 1 and 2 numbers on LibriSpeech, as a comparison [8, I].
Wav2Letter - performance

Table 2: LER/WER of the best sets of hyper-parameters for each feature types.

<table>
<thead>
<tr>
<th></th>
<th>MFCC LER</th>
<th>MFCC WER</th>
<th>PS LER</th>
<th>PS WER</th>
<th>Raw LER</th>
<th>Raw WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev-clean</td>
<td>6.9</td>
<td>7.2</td>
<td>9.3</td>
<td>9.1</td>
<td>10.3</td>
<td>10.1</td>
</tr>
<tr>
<td>test-clean</td>
<td>6.9</td>
<td>7.2</td>
<td>9.1</td>
<td>9.4</td>
<td>10.6</td>
<td>10.1</td>
</tr>
</tbody>
</table>
Questions?