

End-to-end Automatic Speech Recognition Part1 (CTC and Decoding)

Wonyong Sung
Signal Processing Systems Lab @ SNU

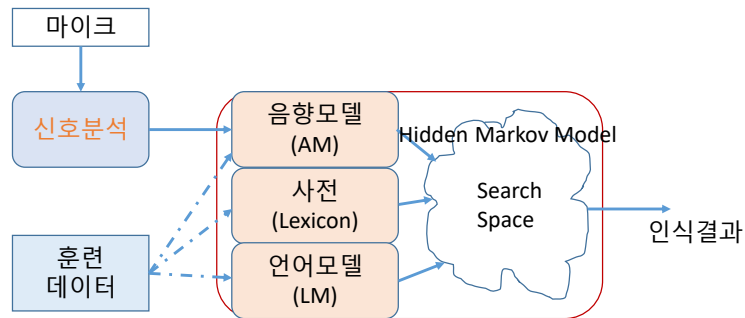


Contents

- End-to-end speech recognition 개요
- CTC algorithm
 - Overview
 - Training
 - Tensorflow Implementation
 - Decoding
- CTC Implementation Examples
 - DeepSpeech2
 - CTC Gram
 - CTC Word
 - CTC on FPGA



기존 음성인식 – Many Components Integration (확률모델)



신호분석 – 음성을 주파수 성분으로 나눔 (time to frequency domain)

음향모델 – 음성을 음소 (phoneme) 단위로 분석

사전 – 어떤 단어의 발음 (음소의 연결)

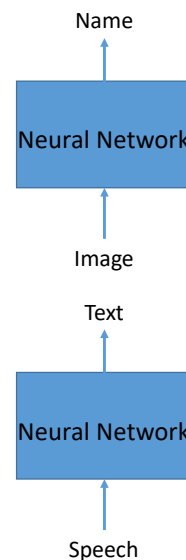
언어모델 – 다음에 나올 단어를 예측 (perplexity를 줄임)

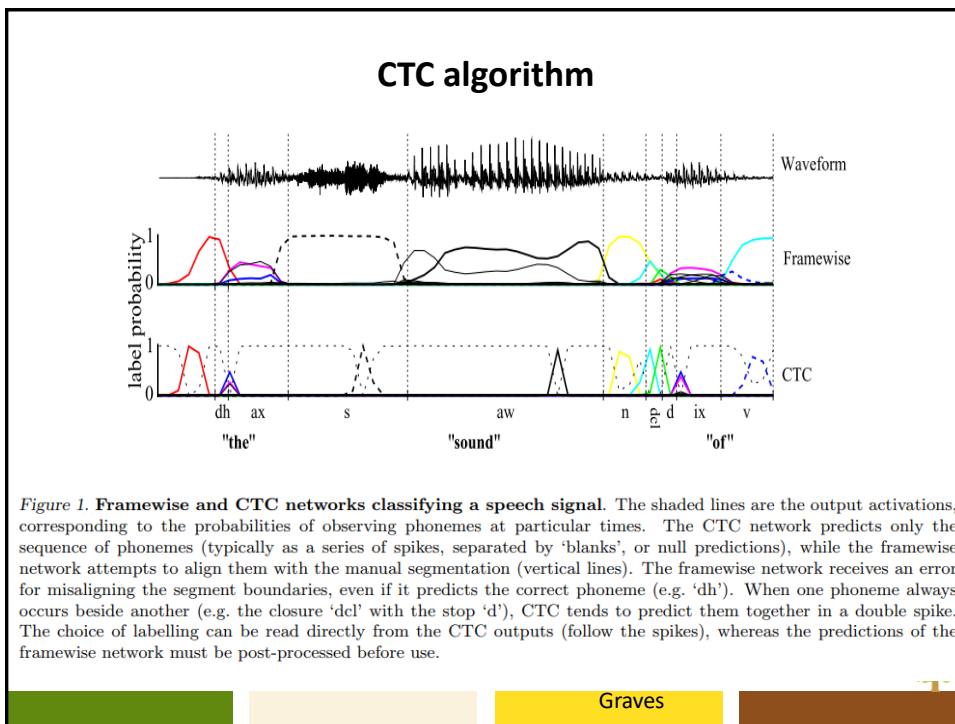
탐색 – 들어오는 발음을 세가지 모델에 맞추어서 가장 **확률이 높은** 연결을 찾음



End-to-end Speech Recognition

- 음성을 인공신경망 모델에 넣으면 (중간과정 없이) 텍스트 (단어 나 character)가 나옴
- 개념은 Image recognition 과 비슷
- 왜 speech 는 어려운가?
 - 음성은 내부에 sub-structure 가 있다. (phoneme 의 복잡한 연결이 글자, 글자의 연결이 단어, 단어의 연결이 문장 등)
 - Segmentation problem 이 있다.
 - 내부 structure 가 있는 문제 들: 번역, handwriting text recognition





Acoustic modeling with CTC RNN

- 음성을 듣고서 발음으로 표시함. 이 때 표현하는 발음의 단위에 따라
 - ~~Triphone states (매우 작은 시간 단위로 표현)~~
 - 매우 안정된 주파수 특징을 보이기 때문에 인식에 쉽다.
 - 이를 꺾어 맞추어서 단어와 문장으로 만들기가 복잡하다. 더 복잡한 hidden Markov model.
 - 전체적으로 정확도 좋다.
 - Monophone (40~70개의 phoneme 으로 표현)
 - Grapheme (알파벳 글자로 표현) – 수십개, 1초에 몇 번 나옴.
 - Wordpiece – 수백 ~ 수만
 - ~~Word – 수십만 단어, 1초 정도의 시간~~



Why end-to-end?

- Potentially better accuracy
- Simpler architecture
- Easy to collect the training data (speech file, text label)
 - No need to frame wise phoneme information

```
1 3227 15907 don't
2 15907 19990 ask
3 21200 23120 me
4 23120 24680 to
5 24680 31248 carry
6 31248 33149 an
7 33149 39080 oily
8 39080 44850 rag
9 44850 49040 like
0 49040 55000 that
```

```
0 13227 h#
13227 13419 d
13419 15093 ow
15093 15907 n
15907 18250 ae
18250 19990 s
19990 21200 epi
21200 21600 m
21600 23120 iy
23120 23728 tcl
23728 23920 t
```

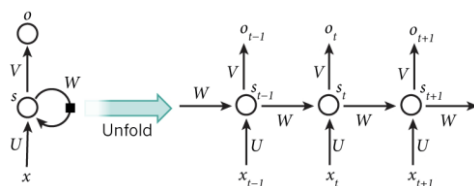


Part 1: CTC algorithm





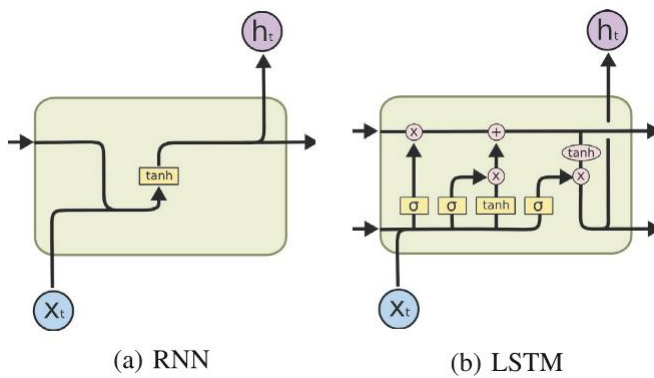
RNN for sequence learning



Parameters to be learned:
 U, V, W



Simple RNN vs LSTM (Long Short Term Memory) RNN



LSTM

▪ Long Short-Term Memory

- *Long Short-Term Memory, 1997. (Sepp Hochreiter, Jürgen Schmidhuber)*
- *Learning to Forget: Continual prediction with LSTM, 2000.*
- Leaky Unit과 비슷하게, state가 덜 지워질 수 있는 방법을 찾음.
- Gate를 도입!
- Gate는 state만 leaky 하게 만드는 것이 아니라 input과 output도 제어.

▪ Gate: [0,1] 사이의 값으로 나와 결과와 element-wise 곱셈.

- 결과의 각 element가 어느 정도 반영될 지를 결정.
- Sigmoid $\sigma \rightarrow [0,1]$

$$\begin{aligned} f_t &= \sigma(U_f x_t + W_f h_{t-1} + b_f) \\ i_t &= \sigma(U_i x_t + W_i h_{t-1} + b_i) \\ o_t &= \sigma(U_o x_t + W_o h_{t-1} + b_o) \\ C_{cand} &= \tanh(U_c x_t + W_c h_{t-1} + b_c) \end{aligned}$$

Forget Gate

Input Gate

Output Gate

Cell Candidate

$$\begin{aligned} C_t &= C_{t-1} \odot f_t + C_{cand} \odot i_t \\ h_t &= \tanh(C_t) \odot o_t \end{aligned}$$

Updated Cell

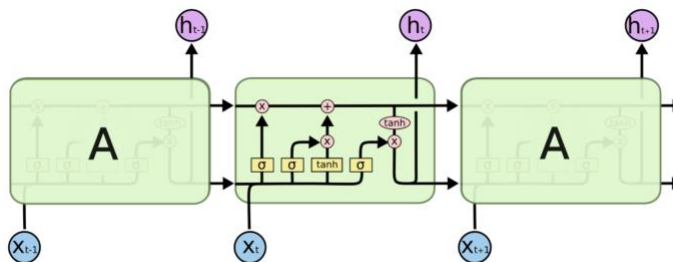
Output



LSTM

▪ LSTM

- State는 2개 (cell, output)
- Gate와 candidate는 h_{t-1}, x_t 에만 영향을 받음.
- 핵심 메모리인 Cell은 leaky unit 처럼 관리.
- $C_t = C_{t-1} \odot f_t + C_{cand} \odot i_t$
- 출력은 업데이트 된 Cell에만 영향을 받음.
- $h_t = \tanh(C_t) \odot o_t$

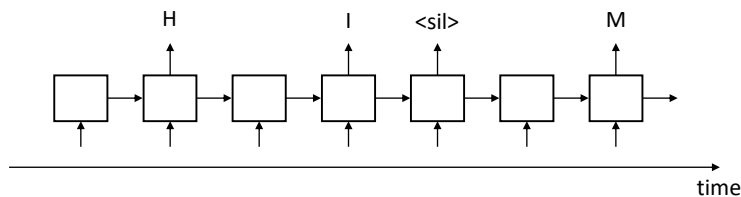


The repeating module in an LSTM contains four interacting layers.



Speech recognition with RNN

- RNN: Sequence data를 처리하기 적합
 - Text data -> language model
 - Speech data -> acoustic model
- Acoustic model : T 개의 입력 -> N 개의 출력
 - T : # of sampled speech feature
 - N : # of text sequence(# of character, # of phone, ...)
 - 일반적으로, $T \gg N$
 - 몇 번째 step에서 출력을 얻을 것인지 알아야 한다.



CTC algorithm

- Connectionist Temporal Classification
 - Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks, 2006 ICML.
 - 시간 축에서의 classification을 neural network로 해결
 - Feature : frame sequence
 - Label : text sequence: {a, b, c, : : : , z, space, apostrophe, blank}
 - Input과 output의 sequence 길이가 다른 문제점을 blank label 을 넣어서 해결!
 - Data labeling이 훨씬 간단해진다.

CTC algorithm

▪ How?

- Output 의 sequence 길이를 늘려서 frame sequence 길이와 같게 만든다.
- Temporal classification을 위해 **blank** 라벨을 추가
- Example
 - Label : ABCD (4)
 - Expanded CTC path : __A__BBBB__CD__ (13)
- Blank는 실제 label과는 관련이 없다.
- 연속된 expanded label은 하나의 label로 처리된다
 - 실제 label이 연속된 같은 값을 가지는 경우 blank로 구분
 - ABBC -> __A__BBB_BB__C



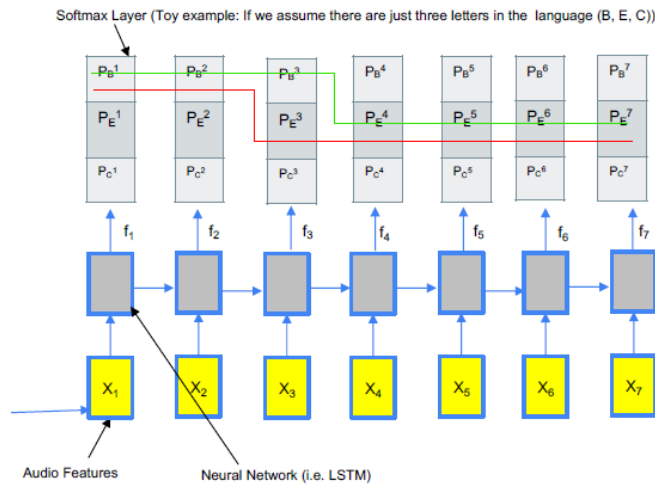
CTC algorithm

- Output sequence 의 길이를 늘리는 경우의 수가 아주 많다.
- Label : ABC
 - Expanded path(= 정답인 CTC output)
 - => {ABC__, A_BC__, ..., __ABC}
- Acoustic model Recall
 - $L^* = \operatorname{argmax}_L P(L|X)$
 - Inference : 주어진 Speech feature를 가장 잘 표현하는 text sequence를 찾는 것
 - Training : speech feature를 처리해 해당 label이 나올 확률을 높이는 것
 - $P(L|X)$ 의 계산이 필요



Sequence learning problem

- 어느 한 순간이 아닌 전체 경로에서의 확률값이 가장 높아야 한다.



CTC algorithm

Notation

- Input sequence : $X = \{x_1, x_2, \dots, x_T\}$
- target label sequence : L (text sequence)
 - $L = \{l_1, l_2, \dots, l_N\}$
- Path : π (CTC output sequence)
 - $\pi = \{\pi_1, \pi_2, \dots, \pi_T\}$
- $\pi \rightarrow L$: mapping from path to label (___A_B_C -> ABC)

CTC acoustic model

- $P(L|X) = \sum_{\pi \rightarrow L} P(\pi|X)$
- $P(\pi|X) = P(\pi_1, \dots, \pi_T | x_1, \dots, x_T)$

$$= P(\pi_1 | x_1) P(\pi_2 | \pi_1, x_1, x_2) \dots P(\pi_T | \pi_1, \dots, \pi_{T-1}, X)$$

CTC algorithm

- CTC acoustic model

- $$P(L|X) = \sum_{\pi \rightarrow L} P(\pi|X)$$
- $$P(\pi|X) = P(\pi_1, \dots, \pi_T | x_1, \dots, x_T)$$

$$= P(\pi_1 | x_1) P(\pi_2 | \pi_1, x_1, x_2) \dots P(\pi_T | \pi_1, \dots, \pi_{T-1}, X)$$

$$\approx P(\pi_1 | x_1) P(\pi_2 | x_2) \dots P(\pi_T | x_T)$$

$$= \prod_{t=1}^T P(\pi_t | x_t)$$

- Conditional Independence 가정하여 $P(\pi|X)$ 계산



CTC algorithm

- $P(L|X) = \sum_{\pi \rightarrow L} P(\pi|X)$
 - $P(\pi|X)$ 를 가능한 path에 대해 모두 계산 ($\pi \rightarrow L$)
- L 을 CTC blank 가 포함된 label sequence L_c 로 확장
 - $L = ABC$ (길이 N)
 - $L_c = _A_B_C_$ (길이 2N+1)
- State transition matrix (2N+1, T) 을 만들고 가능한 path 연결
(N은 label의 길이인데 글자 사이에 blank 넣고 처음 시작에 blank 넣으면 길이가 2N+1)
- Example
 - $L = DOOR$
 - $L_c = _D_O_O_R_$



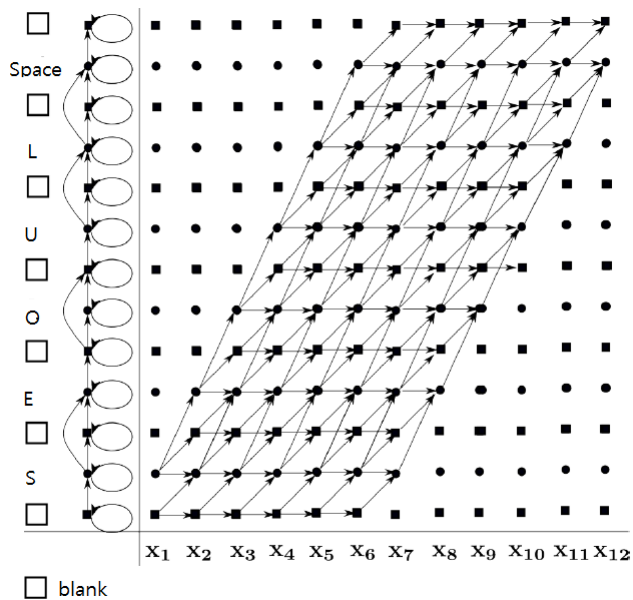
Sequence output 의 제약조건

- State는 blank 를 포함한 character (a, b, ..., z, space, apostrophe, blank 등)
- $t \rightarrow t+1$ 로 갈 때
 - 같은 state 에 머무를 수 있다.
 - Non-blank 에서 다음의 blank 로 갈 수 있다.
 - Blank 에서 다음의 non-blank 로 갈 수 있다.

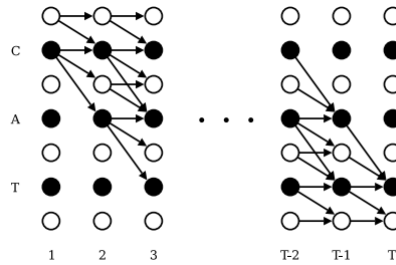


$2N+1$ state

CTC state transition 의 예



CTC algorithm



State transition
확률을 동일하게
가정

$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\pi \in N^T; \mathcal{B}(\pi_{1:t})=1_{1:s}} \prod_{t'=1}^t y_{\pi_{t'}}^{t'}$$

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{l'_s}^t & \text{if } l'_s = b \text{ or } l'_{s-2} = l'_s \\ (\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)) y_{l'_s}^t & \text{otherwise} \end{cases}$$

$$\bar{\alpha}_t(s) \stackrel{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1).$$

$$\beta_t(s) \stackrel{\text{def}}{=} \sum_{\pi \in N^T; \mathcal{B}(\pi_{t:T})=1_{s:|1|}} \prod_{t'=t}^T y_{\pi_{t'}}^{t'}$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y_{l'_s}^t & \text{if } l'_s = b \text{ or } l'_{s+2} = l'_s \\ (\bar{\beta}_t(s) + \beta_{t+1}(s+2)) y_{l'_s}^t & \text{otherwise} \end{cases}$$

$$\bar{\beta}_t(s) \stackrel{\text{def}}{=} \beta_{t+1}(s) + \beta_{t+1}(s+1).$$

Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with RNNs, 2006 ICML.



Connectionist Temporal Classification

- **Posterior** for sentence $W = w_1^M$ and features $X = x_1^T$:

$$p(W|X) = \sum_{s_1^T \in \mathcal{B}^{-1}(W)} p(s_1^T|X)$$

$$:= \sum_{s_1^T \in \mathcal{B}^{-1}(W)} \prod_{t=1}^T p(s_t|x_t)$$

- **Training criterion** for training samples (X_n, W_n) , $n = 1, \dots, N$:

$$\mathcal{F}_{\text{CTC}}(\Lambda) = -\frac{1}{N} \sum_{n=1}^N \log p_{\Lambda}(W_n|X_n)$$

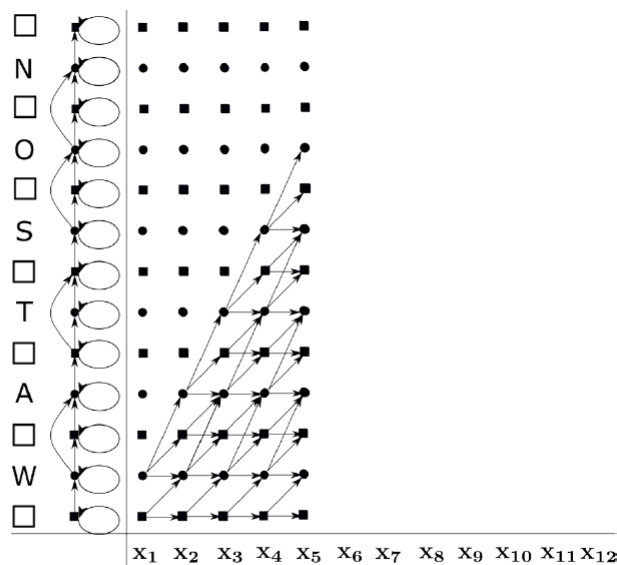


Forward-Backward Decomposition

- ▶ $\alpha(t, m, v)$: Sum over $s_1^t \in B(w_1^m)$ for given x_1^t ending in v .
- ▶ $\beta(t, m, v)$: Sum over $s_t^T \in B(w_m^M)$ for given x_t^T starting in v .
- ▶ Choose $t \in 1, \dots, T$:

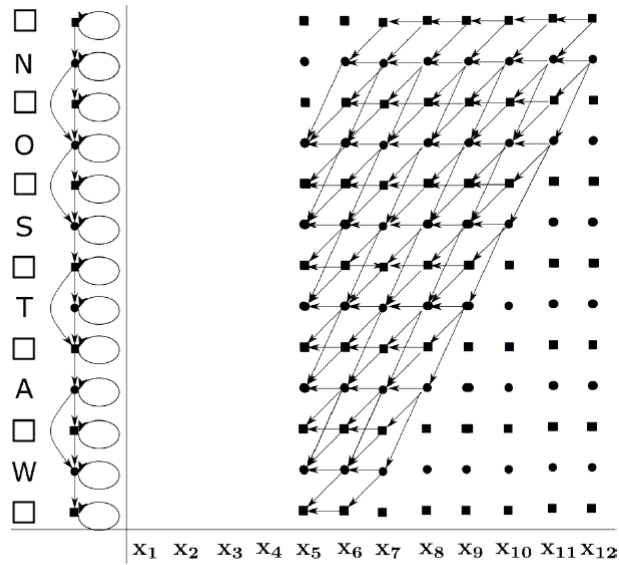
$$\begin{aligned}
 p(w_1^M | x_1^T) &= \sum_{s_1^T \in B^{-1}(w_1^M)} p(s_1^T | x_1^T) \\
 &= \dots \\
 &= \sum_{m=1}^M \sum_{v \in \{w_m, \square\}} \frac{\alpha(t, m, v)}{p(v | x_t)} \cdot p(v | x_t) \cdot \frac{\beta(t, m, v)}{p(v | x_t)}
 \end{aligned}$$

Forward Algorithm



- ▶ Compute $\alpha(5, m, v)$.

Backward Algorithm



► Compute $\beta(5, m, v)$.

CTC Derivatives

► Derivative **posterior**:

$$\begin{aligned}
 & \nabla_{p(s|x_t)} P(W|X) \\
 &= \nabla_{p(s|x_t)} \sum_{m=1}^M \sum_{v \in \{w_m, \square\}} \frac{\alpha(t, m, v)}{p(v|x_t)} \cdot p(v|x_t) \cdot \frac{\beta(t, m, v)}{p(v|x_t)} \\
 &= \sum_{m=1}^M \sum_{v \in \{w_m, \square\}} \delta(v, s) \frac{\alpha(t, m, s) \cdot \beta(t, m, s)}{p^2(s|x_t)}
 \end{aligned}$$

$$\frac{\partial O^{ML}(\{(\mathbf{x}, \mathbf{z})\}, \mathcal{N}_w)}{\partial y_k^t} = -\frac{\partial \ln(p(\mathbf{z}|\mathbf{x}))}{\partial y_k^t}$$

$$p(\mathbf{l}|\mathbf{x}) = \sum_{t=1}^T \sum_{s=1}^{|\mathbf{l}|} \frac{\alpha_t(s) \beta_t(s)}{y_{l_s}^t}.$$

$$\frac{\partial \ln(p(\mathbf{l}|\mathbf{x}))}{\partial y_k^t} = \frac{1}{p(\mathbf{l}|\mathbf{x})} \frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t}$$

$$\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t} = -\frac{1}{y_k^{t,2}} \sum_{s \in \text{lab}(\mathbf{l}, k)} \alpha_t(s) \beta_t(s).$$



CTC Training of Long Speech Input

- 일반적 CTC 훈련용 input 의 길이는 짧다 (10초 이내). 실제로 소리가 긴 상황에서 잘 동작?
- 긴 소리로 훈련을 시킴. Partial windowing
- 짧은 소리로 훈련시켜도 긴 소리 잘 인식 (Encoder-Decoder 모델에서는 이 문제 때문에 LAS가 나옴)
- Hwang, Kyu Yeon, and Wonyong Sung. "Sequence to sequence training of ctc-rnns with partial windowing." International Conference on Machine Learning. 2016.

Sequence to Sequence Training of CTC-RNNs with Partial Windowing

Kyu Yeon Hwang
Wonyong Sung
Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul, 08826 Korea

KYUYEON.HWANG@GMAIL.COM
WYSUNG@SNU.AC.KR



CTC algorithm

- Training CTC-RNN
 - $p(L|X) = \sum_u \alpha(t, u) \beta(t, u)$ 를 높이는 방향으로 훈련
 - $\text{Loss} = -\ln(\sum_u \alpha(t, u) \beta(t, u))$
 - Target label의 NLL을 줄이는 방향으로 훈련 (NLL: negative log likelihood)
- Inference CTC-RNN
 - $L^* = \operatorname{argmax}_L P(L|X)$
 - 가능한 모든 label sequence에 대해 $P(L|X)$ 를 계산
 - **Computationally Expensive!**
 - Search space를 줄이기 위한 다양한 decoding 방법
 - Greedy Decoding
 - Beam search Decoding
 - WSFT (EESN)



Tensorflow CTC-RNN

- Training
 - Tensorflow에는 CTC_Loss라는 함수가 존재
 - Dynamic programming이 미리 되어있다.
 - $\text{Loss} = \text{Tf.ctc_loss}(\text{logit}, y)$ 한 줄로 처리 가능!
 - Logit shape : (N_batch, N_seq, n_class + 1)
 - Y(label) shape : sparse tensor
 - Sparse tensor
 - (Sparse indices, sparse values, sparse shape)로 구성

0	2	0
0	0	-1
0	0	0

Sparse indices : [[0,1],[1,2]]
 Sparse values : [2, -1]
 Sparse shape : [3,3]



Tensorflow CTC-RNN

■ Training

- 원래 label shape : character sequence -> (n_batch, sentence length)
- 원하는 label shape : (n_batch, n_seq)
- 보통 $n_seq > \text{sentence length}$ (recall $T \gg N$)
- Sparse shape : (n_batch, n_seq) 인 sparse tensor type으로 원래 label을 바꿔줘야 한다.
- Training 시 tf.sparse_placeholder() 를 활용
- Learning rate에 따라 초반에 Nan이 뜰 수 있다.
 - 훈련이 잘 안된 초기에는 확률 값이 매우 낮음
 - NLL -> Inf 가 나와 문제가 생길 수 있음
 - 자주 발생한다면 hyper parameter tuning이 필요
 - 가끔 발생한다면 roll back

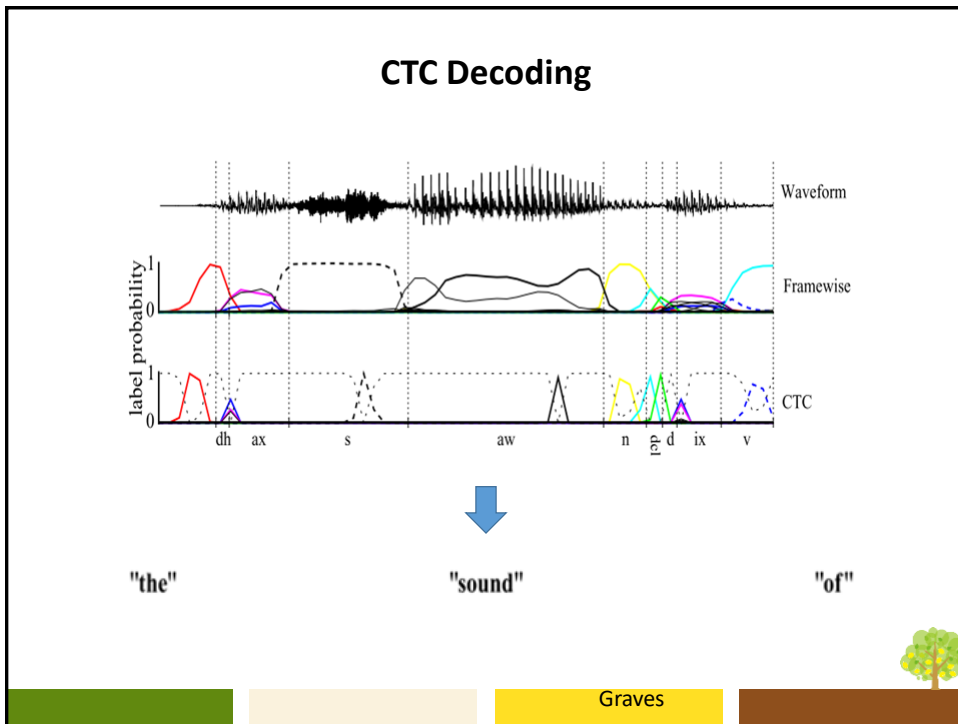


Tensorflow CTC-RNN

■ Training

- Curriculum learning
 - Sequence가 짧은 speech가 상대적으로 쉽다.
 - 짧은 data로 먼저 훈련을 시킨 뒤 점차 긴 data를 학습한다.
- Early-stopping & learning rate decay
 - Training data overfitting을 방지
 - Validation 결과를 보고 overfitting이 예상되면 learning rate를 낮춰 더 작은 범위를 탐색한다.
 - Best model의 parameter들을 매번 저장하며 진행





CTC Decoding

- Decoding CTC output
 - CTC-RNN의 output dimension : # label + 1(blank)
 - 가장 높은 확률의 label sequence L 을 찾는 것이 목적
- Greedy decoding (best path decoding)
 - 가장 쉬운 방법
 - 매 step 가장 높은 확률의 output만 선택해 output path π 결정
 - $\pi \rightarrow L$ mapping을 통해 label predict
 - $\pi = _A_BB_C_ \rightarrow L = ABC$

Greedy Decoding

h h e € € l l l € l l o

h e € l € l o


h e l l o

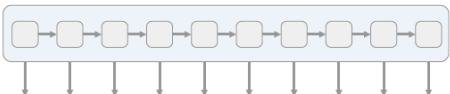
h e l l o

First, merge repeat characters.

Then, remove any € tokens.

The remaining characters are the output.





The input is fed into an RNN, for example.

h	h	h	h	h	h	h	h	h	h
e	e	e	e	e	e	e	e	e	e
l	l	l	l	l	l	l	l	l	l
o	o	o	o	o	o	o	o	o	o
€	€	€	€	€	€	€	€	€	€


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

h	e	€	l	l	€	l	l	o	o
h	h	e	l	l	€	€	l	€	o
€	e	€	l	l	€	€	l	o	o

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

h	e	l	l	o
e	l	l	o	
h	e	l	o	

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



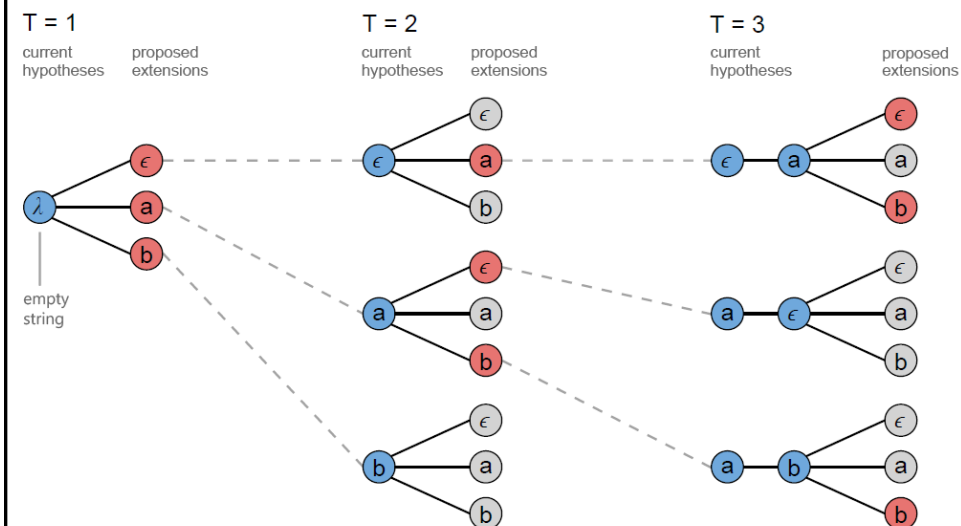
19

CTC algorithm

- Beam search decoding
 - 매 step마다 K-best context를 보존
 - K : beam-width
 - Context : Prefix
 - Label이 N개라면
 - 매 step $K \times N$ 개 path의 확률을 계산
 - 확률의 크기 기반으로 K개만 보존



Beam Search Decoding



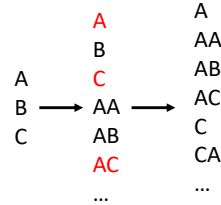
A standard beam search algorithm with an alphabet of $\{\epsilon, a, b\}$ and a beam size of three.

Algorithm 1 Prefix Beam Search: The algorithm initializes the previous set of prefixes A_{prev} to the empty string. For each time step and every prefix ℓ currently in A_{prev} , we propose adding a character from the alphabet Σ to the prefix. If the character is a blank, we do not extend the prefix. If the character is a space, we incorporate the language model constraint. Otherwise we extend the prefix and incorporate the output of the network. All new active prefixes are added to A_{next} . We then set A_{prev} to include only the k most probable prefixes of A_{next} . The output is the 1 most probable transcript, although this can easily be extended to return an n -best list.

```

 $p_b(\emptyset; x_{1:0}) \leftarrow 1, p_{nb}(\emptyset; x_{1:0}) \leftarrow 0$ 
 $A_{\text{prev}} \leftarrow \{\emptyset\}$ 
for  $t = 1, \dots, T$  do
   $A_{\text{next}} \leftarrow \{\}$ 
  for  $\ell$  in  $A_{\text{prev}}$  do
    for  $c$  in  $\Sigma$  do
      if  $c = \text{blank}$  then
         $p_b(\ell; x_{1:t}) \leftarrow p(\text{blank}; x_t)(p_b(\ell; x_{1:t-1}) + p_{nb}(\ell; x_{1:t-1}))$ 
        add  $\ell$  to  $A_{\text{next}}$ 
      else
         $\ell^+ \leftarrow \text{concatenate } \ell \text{ and } c$ 
        if  $c = \text{end}$  then
           $p_{nb}(\ell^+; x_{1:t}) \leftarrow p(c; x_t)p_b(\ell; x_{1:t-1})$  //ab+b case
           $p_{nb}(\ell; x_{1:t}) \leftarrow p(c; x_t)p_b(\ell; x_{1:t-1})$  //ab_+b case
        else if  $c = \text{space}$  then
           $p_{nb}(\ell^+; x_{1:t}) \leftarrow p(W(\ell^+)|W(\ell))^\alpha p(c; x_t)(p_b(\ell; x_{1:t-1}) + p_{nb}(\ell; x_{1:t-1}))$  //ab+' case
        else
           $p_{nb}(\ell^+; x_{1:t}) \leftarrow p(c; x_t)(p_b(\ell; x_{1:t-1}) + p_{nb}(\ell; x_{1:t-1}))$  //ab+c case
        end if
        if  $\ell^+$  not in  $A_{\text{prev}}$  then
           $p_b(\ell^+; x_{1:t}) \leftarrow p(\text{blank}; x_t)(p_b(\ell^+; x_{1:t-1}) + p_{nb}(\ell^+; x_{1:t-1}))$ 
           $p_{nb}(\ell^+; x_{1:t}) \leftarrow p(c; x_t)p_{nb}(\ell^+; x_{1:t-1})$ 
        end if
        add  $\ell^+$  to  $A_{\text{next}}$ 
      end if
    end for
  end for
   $A_{\text{prev}} \leftarrow k \text{ most probable prefixes in } A_{\text{next}}$ 
end for
return 1 most probable prefix in  $A_{\text{prev}}$ 

```



CTC Error 특성

- Output label의 단위 : character (or phone)
- Character error rate은 낮은 편 (6~7%)
- Word error rate은 매우 높음 (26~30%)
 - Beam search를 해도 효과가 낮다.
- Word에 대한 prior probability가 필요
 - Language model rescoring
 - Decoding 과정에서 LM을 이용해 점수 보정
 - $L^* = \text{argmax}_L P_{AM}(L|X)P_{LM}(L)^\alpha (\#words(L)^\beta)$
 - Beam search에 WLM, CLM 등 다양한 LM 적용 가능

CTC decoding with character language model(CLM)

- Maas, Andrew L., et al. "Lexicon-Free Conversational Speech Recognition with Neural Networks." *HLT-NAACL*. 2015.
- Hwang, Kyuyeon, and Wonyong Sung. "Character-level incremental speech recognition with recurrent neural networks." *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016.
- CTC output is the probability distribution of characters (although there is word CTC output), thus it is more logical to incorporate CLM to filter out noisy CTC output
- The decoding procedure is simpler than the WLM constrained CTC decoding (first-pass...)
- CLM does not need a highly demanding softmax function
- The CLM is applied every time step, while the WLM based approach only be applied when we consider adding a space or by computing the likelihood of a sequence being the prefix of a word in the lexicon<- (demand more computation!)
- Can better handle OOV (the result does not care whether the word in in the lexicon or not)



Inputs CTC likelihoods $p_{\text{ctc}}(c|x_t)$, character language model $p_{\text{clm}}(c|s)$

Parameters language model weight α , insertion bonus β , beam width k

Initialize $Z_0 \leftarrow \{\emptyset\}$, $p_b(\emptyset|x_{1:0}) \leftarrow 1$, $p_{\text{nb}}(\emptyset|x_{1:0}) \leftarrow 0$

for $t = 1, \dots, T$ **do**

$Z_t \leftarrow \{\}$

for s in Z_{t-1} **do**

$p_b(s|x_{1:t}) \leftarrow p_{\text{ctc}}(-|x_t)p_{\text{tot}}(s|x_{1:t-1})$ ▷ Handle blanks

$p_{\text{nb}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{nb}}(s|x_{1:t-1})$ ▷ Handle repeat character collapsing

Add s to Z_t

for c in ζ' **do** ζ' : character set excluding “-”,

$s^+ \leftarrow s + c$

if $c \neq s_{t-1}$ **then**

$p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{tot}}(c|x_{1:t-1})$

else

$p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_b(c|x_{1:t-1})$ ▷ Repeat character

end if

Add s^+ to Z_t ▷ Repeat characters have “-” between

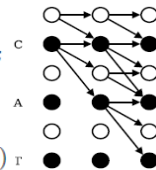
end for

end for

$Z_t \leftarrow k$ most probable s by $p_{\text{tot}}(s|x_{1:t})|s|^\beta$ in Z_t

end for

Return $\arg \max_{s \in Z_t} p_{\text{tot}}(s|x_{1:T})|s|^\beta$



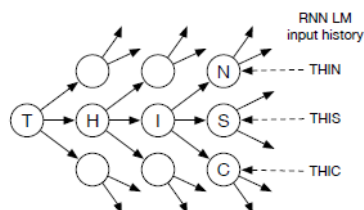


Fig. 2. Beam search tree consisting of label nodes. The CTC blank label is not included.

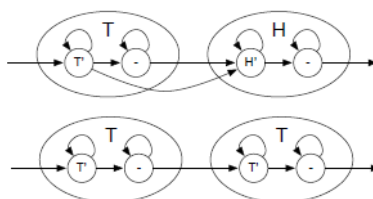
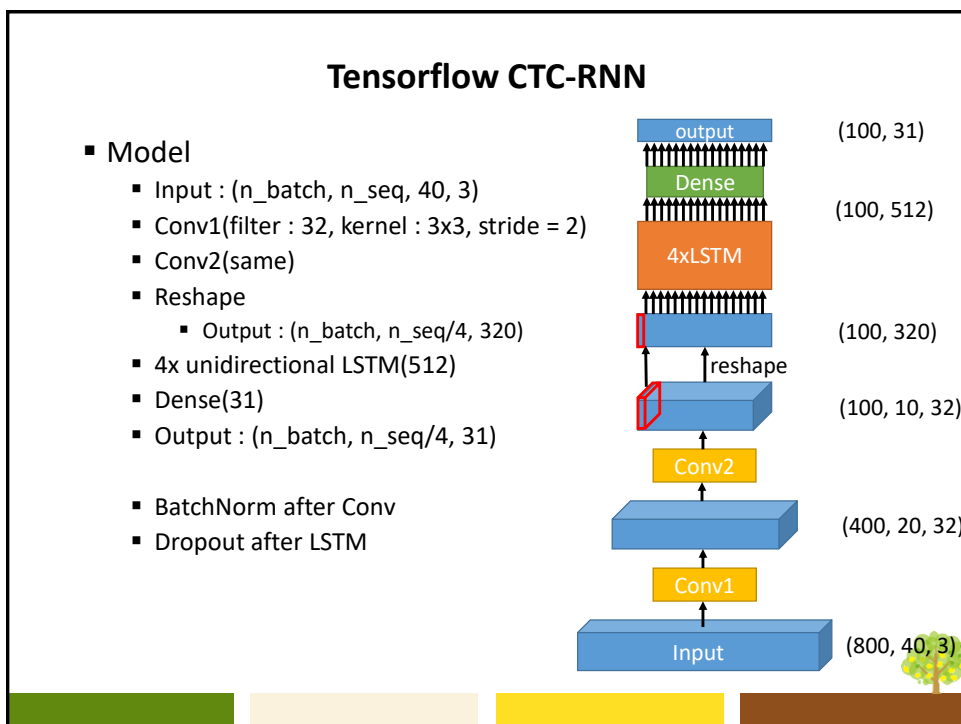


Fig. 3. CTC state transition between two label nodes. If the two nodes have the same label, then a transition between the same CTC state is not allowed.

Implementation Complexity

- Acoustic RNN runs only once.
- The character language model inference is needed for each beam and more frequently (compared to word-model).
- The decoding complexity is proportional to the beam width.
- Memory size: character language model

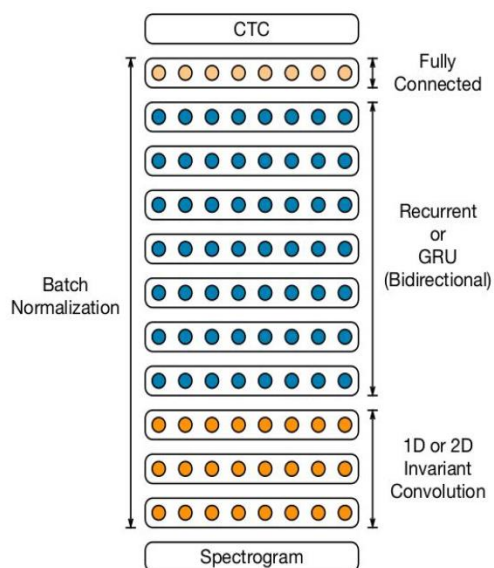


Part 2: CTC Implementation Examples

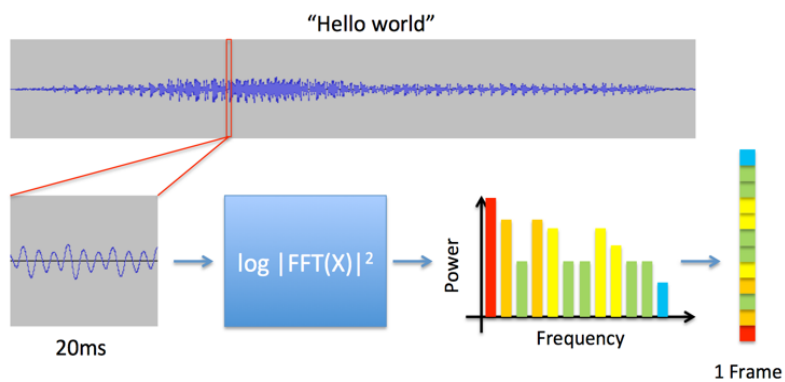
Deepspeech2

▪ Baidu, ICML 2016

1. Spectrogram
2. CNN
3. RNN
4. Fully connected
5. CTC cost function



Spectrogram



Batch Normalization

- To efficiently scale the model as the training set is scaled
- Increasing the depth of the network leads to optimization issues
- BatchNorm accelerates the training for DNN
- Basic formulation for RNN

$$\mathcal{B}(x) = \gamma \frac{x - \mathbb{E}[x]}{(\text{Var}[x] + \epsilon)^{1/2}} + \beta$$

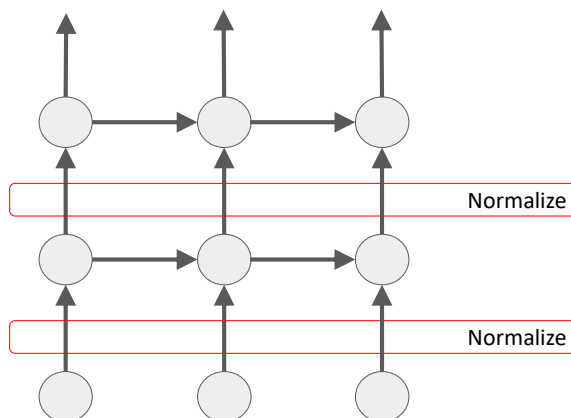
$$h^l = \vec{h}^l + \overleftarrow{h}^l$$

$$\vec{h}_t^l = f(W^l h_t^{l-1} + \vec{U}^l \vec{h}_{t-1}^l + b^l)$$

$$\overleftarrow{h}_t^l = f(\mathcal{B}(W^l h_t^{l-1} + \vec{U}^l \vec{h}_{t-1}^l))$$



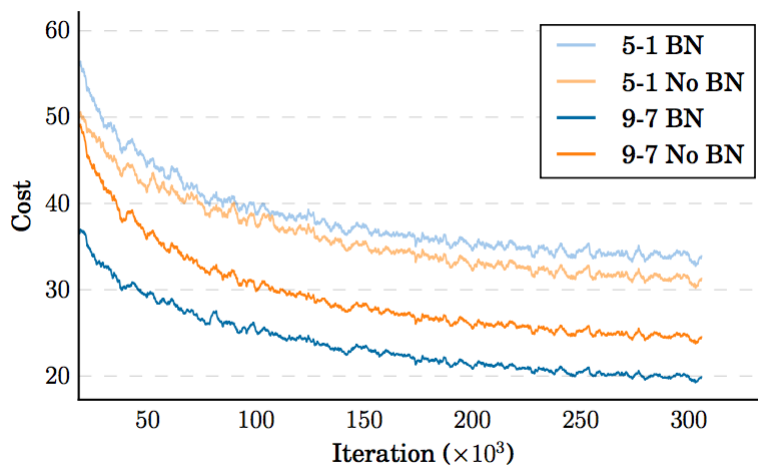
Deep Speech – Batch Norm for RNNs



$$\vec{h}_t^l = f(\mathcal{B}(W^l h_t^{l-1}) + \vec{U}^l \vec{h}_{t-1}^l)$$



Deep Speech – Batch Norm for RNNs



Slides from Awni Hannun

Deep Speech - Hours of speech data

Language	Hours
English	12,000
Mandarin	10,000

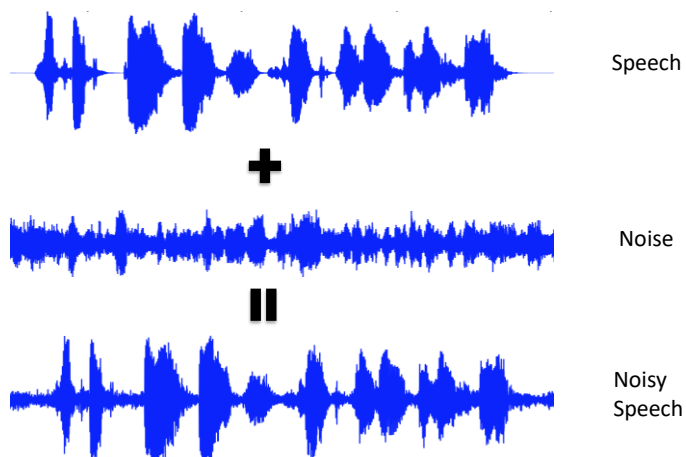
Where does the data come from?

- Public benchmarks (English)
- Internal manually labelled data (English and Mandarin)
- Captioned videos (English and Mandarin)

Slides from Awni Hannun

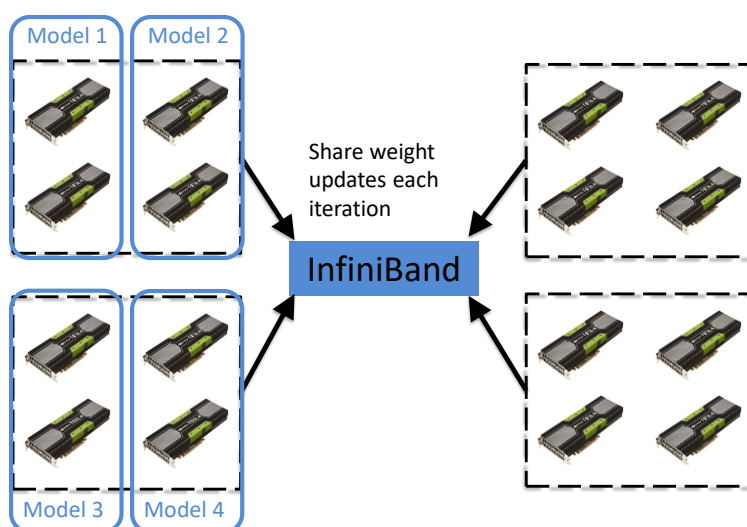
Deep Speech - Even more data!

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



Slides from Awni Hannun

Deep Speech – Data Parallel GPU Scaling



Slides from Awni Hannun

Deep Speech – **Some results**

Architecture	English (WER)	Mandarin (WER)
5-layer 1-RNN	13.55	15.41
5-layer 3-RNN	11.61	11.85
5-layer 3-RNN + BatchNorm	10.56	9.39
9-layer 7-RNN + BatchNorm + Frequency Convolution	9.52	7.93

Mandarin: 6000 output labels

Slides from Awni Hannun



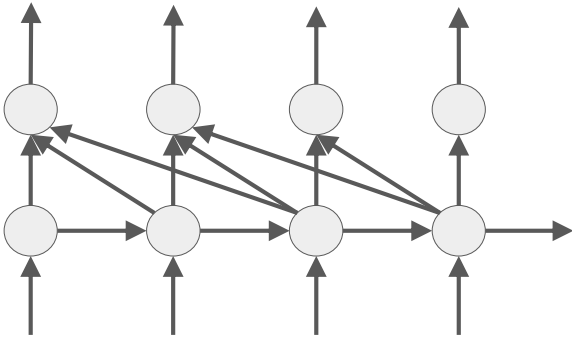
Deep Speech – **Deployment**

- Bi-directional models give almost 10% relative boost ... but we can't deploy them.
- ASR latencies for voice search <50ms
- For 3 second audio would need to decode 60x faster than realtime!

Slides from Awni Hannun



Deep Speech – **Lookahead convolution**



$$\mathbf{h}_t = \sum_{j=1}^{\tau+1} \mathbf{w}_j \odot \mathbf{x}_{t+j-1}$$

Slides from Awni Hannun



Deep Speech – **Lookahead convolution**

For a lookahead of 20 time-steps (about 800ms in the future)

Model	English (WER)	Chinese (WER)
Forward only	18.8	15.7
Forward + Lookahead (+50k params)	16.8	13.5
Bidirectional (+12M params)	15.4	12.8

Slides from Awni Hannun



SortaGrad

- Training on examples of varying length pose some algorithmic challenges
 - ✂ Think of how a child learns
 - ✂ Longer examples tend to be more challenging
- 1st epoch : iterate through the training set in increasing order of the length of the longest utterance in the minibatch
- Other epochs : the training reverts back to a random order over minibatches

	Train		Dev	
	Baseline	BatchNorm	Baseline	BatchNorm
Not Sorted	10.71	8.04	11.96	9.78
Sorted	8.76	7.68	10.83	9.52



LM based Decoding

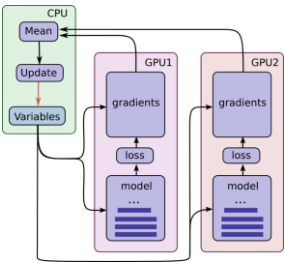
- Beam search decoding with 5-gram LM
- Parameters tuned to EV set

$$Q(y) = \log(p_{\text{ctc}}(y|x)) + \alpha \log(p_{\text{lm}}(y)) + \beta \text{word_count}(y)$$



GPU Implementations

- Memory allocation issue
 - Most of it is for activations through each layer for use by back propagation
 - 70M parameters: “only” 280 MB of memory for the **weights** but for the **activations** with a batch of 64 and seven-second utterances it is 1.5 GB...
- Leads to parallel SGD
- Also allocate to CPU memory accessible by the GPU



Effects of Data Size

Fraction of Data	Hours	Regular Dev	Noisy Dev
1%	120	29.23	50.97
10%	1200	13.80	22.99
20%	2400	11.65	20.41
50%	6000	9.51	15.90
100%	12000	8.46	13.59

Scaling Up

LibriSpeech - clean 100h

- Big variance of WER depending on batch size
- batch size = 75 (max): WER = 58%
- batch size = 40 : WER = 52%
- batch size = 12 : WER = 42%
- Tradeoff: faster training vs lower WER
- Explanation: noisy gradients

LibriSpeech - clean ~1000h (full training dataset)

- Audio files: 60 GB - almost 300k utterances
- Input data to network: 212 GB
- Batch size 64
- One epoch took ~8h
- Whole training over a week
- WER ~12%
- Baidu got 5.33%



Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary Speech Recognition

Hagen Soltau, Hank Liao, Hasim Sak
Google, Inc.
{soltau,hankliao,hasim}@google.com

- Soltau, Hagen, Hank Liao, and Hasim Sak. "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition." *arXiv preprint arXiv:1610.09975* (2016).



- 100,000 words
- 125,000 hours of semi-supervised acoustic training data.

We present results that show it is possible to build a competitive, greatly simplified, large vocabulary continuous speech recognition system with whole words as acoustic units. We model the output vocabulary of about 100,000 words directly using deep bi-directional LSTM RNNs with CTC loss. The model is trained on 125,000 hours of semi-supervised acoustic training data, which enables us to alleviate the data sparsity problem for word models. We show that the CTC word models work very well as an end-to-end all-neural speech recognition model without the use of traditional context-dependent sub-word phone units that require a pronunciation lexicon, and without any language model removing the need to decode. We demonstrate that the CTC word models perform better than a strong, more complex, state-of-the-art baseline with sub-word units.



order possibly with repetitions and with blank labels permitted between labels. The CTC loss can be efficiently and easily computed using finite state transducers (FSTs) as described in Sak et al. [15]

$$\mathcal{L}_{CTC} = - \sum_{(x,l)} \ln p(z^l|x) = - \sum_{(x,l)} \mathcal{L}(x, z^l) \quad (1)$$

where x is the input sequence of acoustic frames, l is the input label sequence (e.g. a sequence of words for the NSR model), z^l is the lattice encoding all possible alignments of x with l which allows label repetitions possibly interleaved with *blank* labels. The probability for correct labelings $p(z^l|x)$ can be computed using the forward-backward algorithm. The gradient of the loss function w.r.t. input activations a_i^t of the softmax output layer for a training example can be computed as follows:

$$\frac{\partial \mathcal{L}(x, z^l)}{\partial a_i^t} = y_i^t - \frac{1}{p(z^l|x)} \sum_{u \in \{u: z_u^l = l\}} \alpha_{x, z^l}(t, u) \beta_{x, z^l}(t, u) \quad (2)$$

where y_i^t is the softmax activation for a label l at time step t , and u represents the lattice states aligned with label l at time t , $\alpha_{x, z^l}(t, u)$ is the forward variable representing the summed probability of all paths in the lattice z^l starting in the initial state at time 0 and ending in state u at time t , $\beta(t, u)$ is the backward variable starting in state u of the lattice at time t and going to a final state.

The NSR model has a final softmax predicting word posteriors with the number of outputs equaling the vocabulary size. Modeling words directly can be problematic due to data sparsity, but we use a large amount of acoustic training data to alleviate it. We experiment with both written and spoken vocabulary. The vocabulary obtained from the training data transcripts is mapped to the spoken forms to reduce the data sparsity further and limit label ambiguity for the spoken vocabulary experiments. For written-to-spoken domain mapping a FST verbalization model is used [16]. For example, "104" is converted to "one hundred four" and "one oh four". Given all possible verbalizations for an entity, the one that aligns best with acoustic training data is chosen.

The NSR model is essentially an all-neural network speech recognizer that does not require any beam search type of decoding. The network takes as input mel-spaced log filterbank features. The



We train the models in a distributed manner using asynchronous stochastic gradient descent (ASGD) with a large number of machines [17, 18]. We found the word acoustic models performed better when initialized using the parameters from hidden states of phone models—the output layer weights are randomly initialized and the weights in the initial networks are randomly initialized with a uniform $(-0.04, 0.04)$ distribution. For training stability, we clip the activations of memory cells to $[-50, 50]$, and the gradients to $[-1, 1]$ range. We implemented an optimized native TensorFlow CPU kernel (`multi_lstm_op`) for multi-layer LSTM RNN forward pass and gradient calculations. The `multi_lstm_op` allows us to parallelize computations across LSTM layers using pipelining and the resulting speed-up decreases the parameter staleness in asynchronous updates and improves accuracy.



Table 1: Bidirectional-LSTM acoustic models trained on data sets of varying sizes.

Model	Training Criterion	Size	Data (hrs)	WER(%)
CD states	CE	5x600	650	29.0
	CE	5x600	5000	21.2
CD phones	CE	5x600	5000	20.3
	CE	5x600	50000	17.7
	CE	5x600	125000	16.7
	CTC	5x600	125000	16.5
	CTC, multi_lstm_op ¹	5x600	125000	15.5
	CTC, multi_lstm_op ¹	7x1000	125000	14.2



Table 2: CTC CD phone models compared with CTC word models.

Model	Layers	Outputs	Params	Vocab	OOV(%)	WER(%)	
						w/ LM	w/o LM
CTC CD phone	5x600	6400	14m	500000	0.24	15.5	—
	7x1000	6400	43m	500000	0.24	14.2	—
	7x1000	35326	75m	500000	0.24	14.5	—
	7x1000	6400	43m	82473	0.63	14.7	—
CTC spoken words	5x600	82473	57m	82473	0.63	14.5	15.8
	7x1000	82473	116m	82473	0.63	13.5	14.8
CTC written words	7x1000	97827	137m	97827	0.70	13.4	13.9

Table 3: Comparison of CD phone with spoken word models in spoken domain.

Model	Layers	Outputs	Params	Vocab	OOV(%)	Spoken WER(%)	
						w/ LM	w/o LM
CTC CD phone	7x1000	6400	43m	500000	0.24	12.3	—
CTC spoken words	7x1000	82473	116m	82473	0.63	11.6	12.0

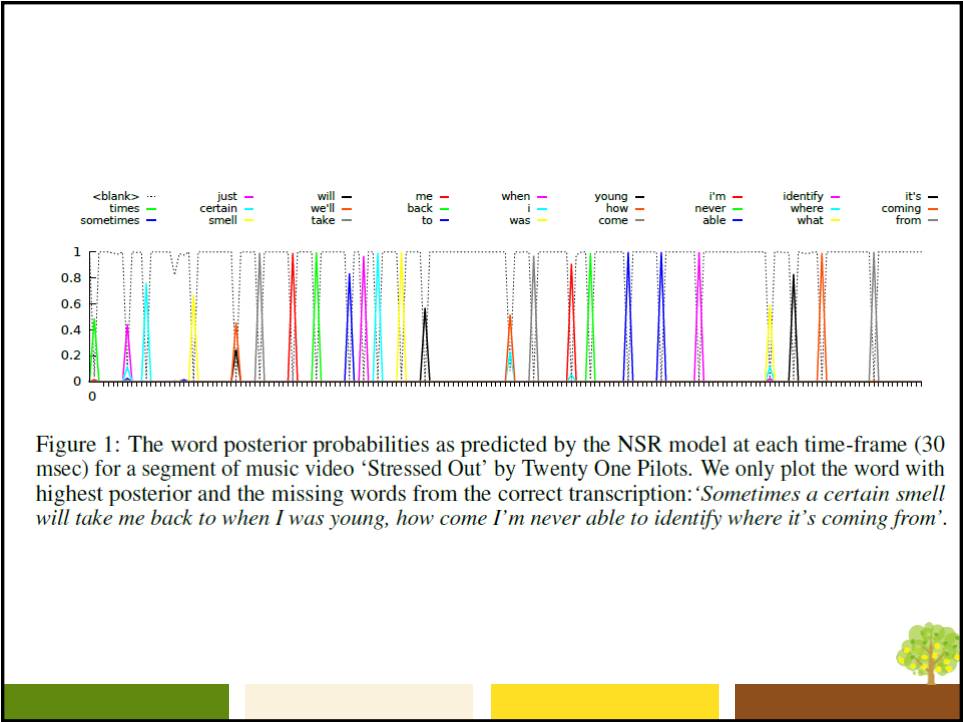
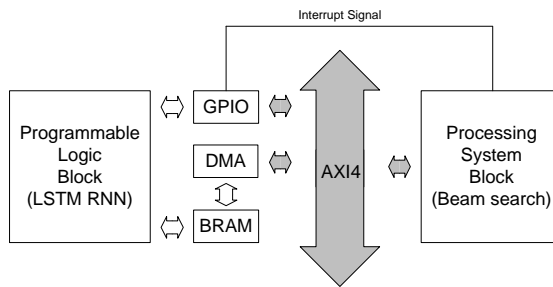


Figure 1: The word posterior probabilities as predicted by the NSR model at each time-frame (30 msec) for a segment of music video ‘Stressed Out’ by Twenty One Pilots. We only plot the word with highest posterior and the missing words from the correct transcription: ‘Sometimes a certain smell will take me back to when I was young, how come I’m never able to identify where it’s coming from’.



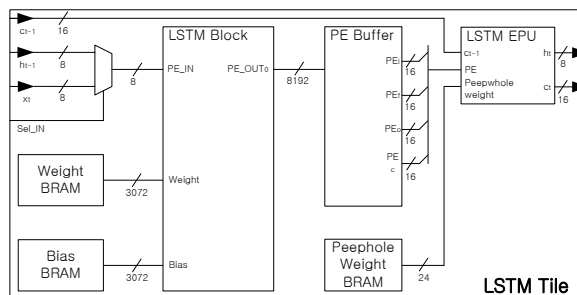
FPGA implementation

- LSTM RNN operations occupy most of the computation.
 - We assign system resource at best by implementing them on the PL block.
 - Hypothesis search is conducted on the PS block (ARM CPU).



FPGA implementation

- Fixed-point optimization (retraining-based) applied
 - Bit-width for weights: 4 ~ 6 bits
 - Bit-width for LSTM memory: 16 bits
 - The LSTM block computes the matrix-vector multiplication in parallel.
 - The results are stored in PE buffer for later use.
 - Other parts of the LSTM equation are implemented on the LSTM EPU.



FPGA implementation results

- RNN for AM + RNN for CLM
 - Small RNN: 3x256 RNN AM and 2x256 C-LM RNN (implemented on Xilinx XC7Z045)
 - Large RNN: 4x512 RNN AM and 4x512 C-LM RNN (implemented on Xilinx XCVU190)
- Small RNN needs about 7MHz for real-time, consuming about 5.5 W (less than 1W for each person). Large RNN consumes about 11.8W



- Low-power, on-device, speech recognition is possible with HMM-free, neural network based algorithms
 - Neural networks can be implemented efficiently by employing many PU's (processing units)
 - Fixed-point quantization and on-chip memory based operations
- RNN based speech recognition algorithm.
 - RNN based AM and hierarchical LM are employed.
 - Can deal with both character- and word-level characteristics.
 - Low complexity input and output (alphabet size, not word size)
 - The parameter size is very small (2.3 M ~ 15.1 M weights), thus it can be implemented without using DRAM accesses.
- FPGA implementation of the algorithm.
 - About 7MHz clock is needed for real-time operation with 128 beam size (more than 10 times speed-up with 100MHz clock)



End-to-end 연구 개발의 문제

- Accuracy (WER): 많은 데이터를 동원하면 기존 (HMM 기반) 과 필적, 또는 더 나아진다. Data!
- 훈련을 위한 efficient computing: multi-GPU, pretraining 등 Acoustic model training 매우 느리다 (weeks)
- Decoding: 많은 경우 외부 LM등을 이용하여 decoding 을 해야 한다. Beam-search 시 width 에 비례해서 LM 계산이 증가한다. 이를 어떻게 줄이나.
- On-line service 문제: bi-directional RNN 또는 LAS (Listen Attend Spell)이 uni-directional RNN 기반보다 성능이 좋다. 그러나 latency 문제
- 희망: 위의 모든 문제가 2~3년 내에 풀릴 것이다. 기존의 HMM 기반보다 더 효율적인 ASR 가능.



Some slides sources:

- End to end automatic speech recognition
http://llcao.net/cu-deeplearning17/lecture/lecture6_Markus.pdf
- (CTC algorithm illustration) <https://distill.pub/2017/ctc/>
- (Deepspeech2) End-to-end speech recognition system using RNNs and the CTC loss function:
<http://www.zhoutengyu.com/ppt/DeepSpeech.pptx?WebShieldSessionVerify=YNWYPZrjOFbUvSXISgvN>



중요논문

- Sequence Transduction with Recurrent Neural Networks A Graves, 2012
- Towards End-to-End Speech Recognition with Recurrent Neural Networks
Graves Jaitly 2014
- EESEN: END-TO-END SPEECH RECOGNITION USING DEEP RNN MODELS AND
Y Miao 2015
- Neural Speech Recognizer: Acoustic-to-Word LSTM Model for Large Vocabulary
Speech Recognition H Soltau, Hasim Sak 2016
- Comparison of decoding strategies for CTC acoustic models
CMU, Karlsruhe
- Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in english and
mandarin." International Conference on Machine Learning. 2016.
- Lee, Minjae, et al. "FPGA-based low-power speech recognition with recurrent neural
networks." Signal Processing Systems (SiPS), 2016 IEEE International Workshop on.
IEEE, 2016.
- Hwang, Kyuyeon, and Wonyong Sung. "Sequence to sequence training of ctc-rnns
with partial windowing." International Conference on Machine Learning. 2016.
- Hwang, Kyuyeon, and Wonyong Sung. "Character-level incremental speech
recognition with recurrent neural networks." Acoustics, Speech and Signal Processing
(ICASSP), 2016 IEEE International Conference on. IEEE, 2016.

