Deep Speech[1] 논문 정리

엮은이: LEE, SANGHEON

1. **들어가며**
   본 논문이 나오기 전, CTC loss라는 것이 상용화되기 전 음성인식을 하는 방식을 상상해보자. 본디 음성인식이란 잡음제거의 전처리과정을 거쳐, 각 음소단위(phoneme)당의 발음을 지속적으로 트레이닝 시키고, 말이 빠르거나 느리거나, 톤이 높거나 낮거나 한 입력 데이터를 같은 음절과 같은 속도로(normalize)만드는 과정이 지난 후에 인식을 하였다. 따라서 음성인식에는 음성 그 자체의 수준높은 지식을 요했고, 필연적인 한계가 크게 존재했다.
   이후 음소단위의 인식이 아닌 글자 단위의 인식이나(영어), 단어 단위의 인식이 시도되어져 왔지만 이러한 접근은 여전히 수많은 학습용 데이터를 전처리하는 과정을 요했다.
   굳이 이런식으로 접근하지말고, 길이가 정해져있지 않은 문장 그 자체를 트레이닝 시키자, 음소의 단위로 데이터를 해체시키지 말고 매 데이터 자체를 통해 RNN 모델을 만들자, 하고 나온 것이 baidu에서 나온 Deep speech, 이를 가능케 한 것이 CTC loss이다.
   본 정리글에서는 이것이 어떻게 가능한 지, 그 아이디어와 논리를 알기쉽게 해석해보고자 한다.

2. **RNN Training step**

   Input data 정의:
   $X = \{(x^{(1)}, y^{(1)}), \dots (x^{(n)}, y^{(n)})\}$
   x : speech data. 파 형태의 음성 데이터가 아닌 음성인식에서의 전통적인 방식으로 Audio spectogram을 그 데이터로 받는다.
   y : 해당 speech data의 문장(transcript)

   Audio spectogram을 사용하고, Deep speech의 특성 상 sequence level training이기 때문에 추가적인 몇 가지의 정의가 필요하다.
   (1)각 speech data의 음성 길이와,
   (2) 해당 speech data에서 spectogram으로 나눠진 주파수 중, 해당 주파수의 특정 시점 t에서의 값(power)의 notation을 정의한다.

   (1) $T^{(i)}$ =각 데이터의 길이
   (2) $x_t^{(i)}$ = 해당 time series[2] t에서의 p 번째 frequency의 값(power).

   가장 첫 번째로, 해당 time series에서 character가 나올 확률을 예측해야 한다.
   $\hat{y}_t = P(c_t|x)$, where $c_t \in \{a, b, c, ..., z, space, apostrophe, blank\}$

   Model 정의:

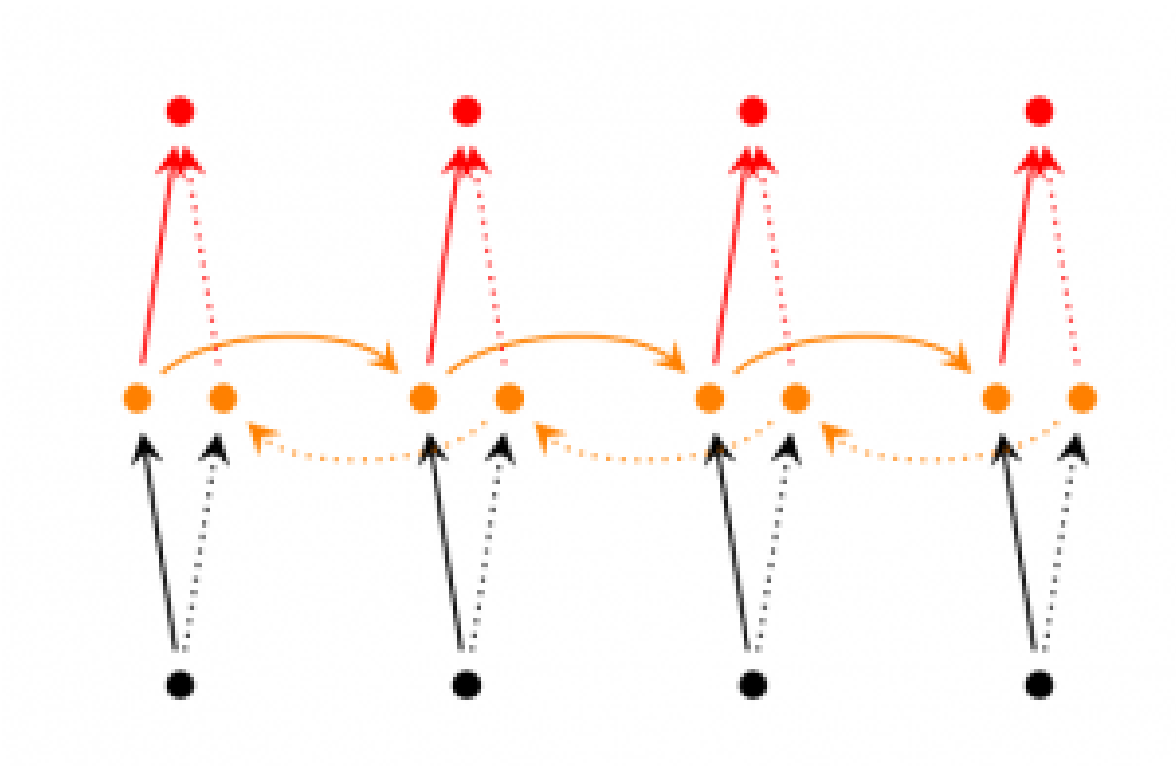
[1] https://arxiv.org/abs/1412.5567
[2] time series 는 통상적으로 10ms이다.

- 5개의 hidden layers.

- 첫 3개 layer는 FC. not recurrent.
  $\Rightarrow h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})$
  $h^{(l)} = l$ 번째 레이어의 hidden unit을 얘기한다.
  cf) 여기서 중요한 부분은, 논문에서는 $h_t^{(l-1)}$ 만이 output을 정의한다는 식의 수식을 사용했지만,
  실험 설계에서 context of C frames where $C \in \{5, 7, 9\}$ 이므로 좀 더 수식을 명확하게 작성하자면
  $\Rightarrow h_t^{(l)} = g(W^{(l)}[h_{t-2}^{(l-1)}, h_{t-1}^{(l-1)}, h_t^{(l-1)}, h_{t+1}^{(l-1)}, h_{t+2}^{(l-1)}] + b^{(l)})$

- ReLu function을 사용하는데, 값이 20 이상인 부분은 clipping 한다. $\Rightarrow g(z) = min\{max\{0, z\}, 20\}$

- 네 번째 layer는 **bi-directional recurrent layer (BRNN)**.



  hidden layer의 왼쪽 점 $\Rightarrow h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)})$
  hidden layer의 오른쪽 점 $\Rightarrow h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})$

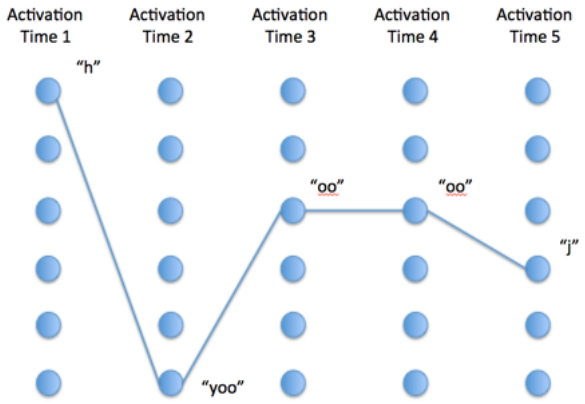
- 다섯 째 layer는 또한 FC.
  $\Rightarrow h_t^{(5)} = g(W^{(5)} h_t^{(4)} + b^{(5)})$ where $W_t^{(4)} = h_t^{(f)} + {}^{(b)}$

- 마지막 layer는 softmax function.
  $\Rightarrow h_{t,k}^{(6)} = \hat{h_{t,k}} = P(c_t = k|x) = \dfrac{exp(W_k^{(6)} h_t^{(5)} + b_k^{(6)})}{\Sigma_j exp(W_j^{(6)} h_t^{(5)} + b_j^{(6)})}$

- Loss function은 CTC loss $L(\hat{y}, y)$ 를 다룬다. CTC loss는 너무나도 거대한 한 축이기 때문에 본 페이지에서는 그 개념만 다루고, Implementation 단이나 추가적으로 따로 다루겠다.
  Gradient descent method는 Nesterov's Accelerated gradient method를 사용한다. 본 method는 굳이 다룰 필요가 없기에 스킵한다.

3. **CTC loss**[3]
해당 time series에 알맞는 음소를 몰라도 loss를 계산하는 획기적인 방식이다. 이에 따라 sequential training이 가능한 것이다.
CTC loss 를 설명하기 전, 잊지 않아야 할 것과 사전에 알아야 할 notation에 대해 언급하고 가겠다.
- 잊지 않아야 할 것 :
  - softmax function에서 나온 T * K dimension의 matrix y_k^t가 input으로 들어간다.
  - 결국 back propagation을 위한 하나의 loss 값으로 이것을 압축시키는 방식다.

- notation :
  - π = path



이 node와 edge 전체가 path이다.
π_t 는 해당 time series의 node이고, edge를 따로 지칭하는 notation은 없지만 이 전체의 sequence가 pi라고 보면 된다.
  - l = label
label은 말 그대로 해당 x, 즉 speech data에 대한 글자이다.
예를 들자면, path에서 node의 진행이 bbbbbbbbbbeeeeeeeeeeeeeee가 된다면 해당 label은 ⇒ be가 될 수 있도록 하는 것이다.

loss란 voice data x가 주어졌을 때 해당 레이블y(여기서는 l. 왜 l이라고 하는지는 결국 sentence와 label의 철학적인 차이 때문인 것 같다)이 나올 확률이다. ⇒ $p(l|x)$
여기서 간단히 함수 B 를 정의하고 가자. $l = B(\pi)$ 인데, 여기서 이 함수는 엄청나게 간단하다.
그냥 모든 공란과 반복되는 label을 지워주기만 하면 되는 것!
$\therefore p(l|x) = \Sigma_{\pi \in B^{-1}(1)} p(\pi|x)$
그러면 $B(\pi) = l$ 인 모든 π를 고려해서 gradient를 구해주면 된다.
근데 여기서 이상한 점이 명확하다. $B(\pi) = l$ 인 π가 너무나 많다는 것.

[3] Alex Graves. Supervised sequence labelling with recurrent neural networks , volume 385. Springer, 2012

그렇다면 실제로 어떻게 이 모든 고려를 하고, 또 하나하나 back propagate하느냐, 너무 많으니까 불가능할 것 같고 모든 label에 대해 general 하지도 않을 것 같다는 물음이 자연스레 든다.
여기에서 **Dynamic programming algorithm** 이 등장한다.
말이 어렵지 dynamic programming이란 본 label의 데이터적 특성에 의해 고려할만한걸 줄이는 것에 불과하다.

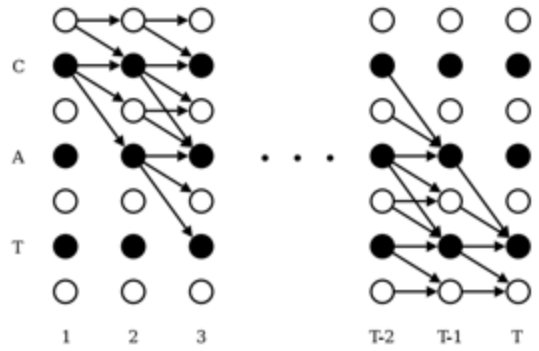
1 본 data는 sequential data이다. 즉, 순서가 있다는 것이다.

2 따라서 만약 label이 SANGHEON이라면 가장 첫 time series에서 S를 예측해야만 하고, 그 다음은 A 또는 S 또는 공백을 예측하는 것만 고려하면 된다. 다른걸 예측하는건 애초에 틀렸다고 보면 되는 것이다.

이런 제약조건 덕분에, α 와 β라는 함수를 통해 손쉽게 $p(l|x)$를 계산하고(재귀적인 방식을 통하여), 이를 또 back propagate할 수 있다.

- **dynamic programming algorithm**
  (1) loss를 계산하는 수식과, (2) 해당 loss에서 back prop을 하는 방식을 나눠 설명하겠다(Maximum likelihood training).

(1) How to calculate loss($p(l|x)$)



본 그림을 이해하는 것이 loss 계산의 전부이다.
x 축은 time series, y 축은 재정의된 레이블 l'이다.
재정의된 label $l' =$ 각 label의 사이와 맨 처음과 맨 끝 blank symbol을 넣어놓은 label이다.
$\Rightarrow |l'| = 2|l| + 1$
먼저 간단히 그림을 해석해보자. cat이라는 음성을 총 T 만큼의 time series로 얘기했다고 생각하자. 첫 time segment에서는 C 라는 말이 나올 수도, 아니면 아무런 말이 안 나왔을 수 있다. 그 다음 time segment에서는 앞서 아무말이 나오지 않았다면 C혹은 아무말을 안할 수 있고(A가 나올 수 없음에 주목하자), 앞서 C라는 발음이 나왔다면 C, C 다음의 blank, 아니면 A라는 발음이 나올 수 있다. 이렇게 재귀적으로(recursive) 진행하여 마지막 time segment T에서 cat의 t가 나오거나 t 이후 공란이 나오게 된다면, 이 time series는 cat이라는 발음을 하게되는 것이다. 이 확률을 구한다면, 그

자체가 loss 즉 $p(l|x)$가 될 것이다.
본 내용을 수식적으로 표현하기 위해 하나의 변수(forward variable)을 정의한다.
$\alpha_t(s) = \sum_{\substack{\pi \in N^T \\ B(\pi_{1:t})=1'_{1:s}}} \prod_{t'=1}^{t} y^{t'}_{\pi'_t}$
forward variable $\alpha_t(s)$ 는 t라는 time series 동안 s 만큼의 label이 읽혔을 총 확률이다.
수식의 한 부분이 논문과는 다른데, 보다 직관적인 이해가 쉽도록 약간 개조했으니 넘어가자.
그래서 $p(l|x)$ 는 T라는 시간동안 l 이라는 label이 읽혔을 총 확률이니까
$\Rightarrow p(l|x) = \alpha_T(|l'|) + \alpha_T(|l'|-1)$ 임은 자명하다.
이제 $\alpha_t(s)$ 의 계산방법만 알면 되는데, 이는 재귀함수로 간단하게 구할 수 있다.
재귀함수 시작값 :
$\alpha_1(1) = y^1_b$
$\alpha_1(2) = y^1_{1_1}$
$\alpha_1(s) = 0, \forall s > 2$
$y^1_b, y^1_{1_1}$는 loss 를 구하기 위한 마지만 softmax에 있다는 것을 잊지말자.
재귀함수 :

$$\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1))y^t_{l'_s} & l'_s = b, l'_{s-2} = l'_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2))y^t_{l'_s} & o.w \end{cases} \quad (1)$$

어려워보여도, 위 그림에서 blank 다음 그 다음 blank로는 못가지만, character(label) 다음 다음 character(label)로는 갈 수 있다는 점을 생각하면 직관적이다.
(사실 본 function은 label이 길어짐에 따라 underflow가 남이 자명하다. 따라서 본 논문에서는 augment된 수식을 제공하는데, 이는 실습 때 자세히 다루기로 한다.)

(2) How to do back propagation(Maximum Likelihood Training)
Gradient descent method는 일단 생각하지 말고, back prop에서 알아야 할 것은 softmax matrix의 각 원소 : y^t_k 마다의 1차 미분값을 알아야 함은 자명하다.
이를 위하여 $\alpha_t(s)$ 와 상반되는 backward variable을 정의한다.

- $\beta_t(s) = \sum_{\substack{\pi \in N^T \\ B(\pi_{1:t})=1'_{1:s}}} \prod_{t'=t}^{t} y^{t'}_{\pi'_t}$
  α variable이 앞에서 부터 계산한 재귀함수라면, β variable은 뒤에서 부터 계산한 variable 이다.
  본 수식의 계산은 그저 alpha와 beta를 뒤집은 것에 불과하기 때문에 재귀함수를 작성하지는 않겠다.

다만 하나를 명확히 하고 넘어가자면 :
$\alpha_t(s) = 0, \forall s < |l'| - 2(T-t) - 1$
$\beta_t(s) = 0, \forall s > 2t$
(1)의 그림을 보자면, cat이라는 음소가 남은 segment상 나올 수 없는 경우 0이고,
하나의 하나의 음소를 뛰어넘을 경우도 0인데 이는 그림을 보면 쉽게 이해할 수 있다.

이제 구해야 할 것은 $\frac{\partial p(l|x)}{\partial y_k^t}$ 이다.
이를 위하여 $\alpha_t(s)\beta_t(s)$ 를 정의하는데, 이는 수식적으로는 복잡할지몰라도
t time segment 에서 s라는 label이 나오는 모든 path의 경우의 확률이다.
$\alpha_t(s)\beta_t(s) = \sum_{\substack{\pi \in \mathrm{B}^{-1}(1): \\ \pi_t = l'_s}} y_{l^s}^t \prod_{t=1}^T y_{\pi_t}^t$

본 수식을 약간만 바꿔보면 : $\frac{\alpha_t(s)\beta_t(s)}{y_{l^s}^t} = \sum_{\substack{\pi \in \mathrm{B}^{-1}(1): \\ \pi_t = l'_s}} \prod_{t=1}^T y_{\pi_t}^t$

위 식의 우항은 time series t 에서 label l'_s가 나오는 모든 경우의 수의 확률의 합이니까,
time series t 에서 나올 수 있는 모든 label의 총 합 :
$\sum_{s=1}^{|l'|} \sum_{\substack{\pi \in \mathrm{B}^{-1}(1): \\ \pi_t = l'_s}} \prod_{t=1}^T y_{\pi_t}^t = p(l|x) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y_{l^s}^t}$ 이다.
α, β variable은 구할 수 있고,
$\sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y_{l^s}^t}$ 본 수식 안에서 y_k^t로 미분하는 것은 매우 trivial하지 않는가! 그래서
이제 각 softmax 원소의 미분 값을 구할 수 있는 것이다!
$\therefore \frac{\partial p(l|x)}{\partial y_k^t} = \frac{1}{y_k^{t^2}} \sum_{s \in lab(l',k)} \alpha_t(s)\beta_t(s)$
여기서 lab 함수의 정의는 결국 time series t 에 s 가 가지는 label이 k 일 수는 없다는건데, 이는 미분할 때 독립성을 유지시키기 위한 것이지만 실제 적용단에서는 매우매우 작은 값이기 때문에 무시해도 전혀 상관이 없다.

4. **Regularization**
보통 이미지 프로세싱의 경우 이미지를 jittering(좌우반전, 움직이기 등) 하여 training time과 test time에 loss를 평균을 내어 적용시키는게 일반적이다.
Audio의 경우는 이러한 방식이 일반적이지 않지만, Baidu의 결과 앞 뒤 5ms정도 audio signal을 당기거나 밀거나 하여 그 결과값을 평균냈을 때 더 좋은 결과가 나옴을 입증하였다.

5. **언어모델**
결국 본 모델은 audio를 character level로 생성하는데, 생성 후 원래 전통적인 언어모델의 통과 과정에 대해 deep speech는 일단 '데이터의 한계'에 대하여 언급한다. 즉, 트레이닝 데이터 셋에서 언급이 적거나, 거의 없는 것들이 인식이 안되는 것이기 때문에 이론상 모든 언어를 충분히 트레이닝 시키면 정확해질 것이라는 말이다. 하지만 실질적인 적용방안으로는 비효율적임이 자명하다.
따라서 본 논문에서는 **N-gram language model**을 사용, 본 딥러닝 모델과 결합한다.

5.1 **N-gram language model**
본 언어모델은 다른 페이지를 참조한다.

5.2 결합방식

결국 음성인식이 구하는 것은 Input 음성(x)에 해당하는 문장(c)중 가장 확률이 높은 문장$P(c|x)$ 을 구하는 것이다.
그렇다면 N-gram을 통하여 $P_{lm}(c)$ 를 구할 수 있으니, Deep speech 둘을 결합한 방식은 다음과 같다:

$$Q(c) = log(P(c|x)) + \alpha log(P_{lm}(c)) + \beta word_count(c)$$

여기서 word count는 글자 수를 이야기하는데, 글자 수를 굳이 넣는 이유는 deep speech에서는 굳이 설명하지 않지만, N-gram 언어모델과 음성인식 모델을 결합하는 과정에서 0으로 수렴하는 것을 막기 위한 통상적인 bias의 역할을 해 주는 것 같다.
이제 새로운 loss($Q(c)$)가 나왔으니, α, β 또한 back propagation으로 구해주면 되는 것이다.

5.3 Beam search algorithm

본 알고리즘에 대한 설명은 다른 페이지를 참조한다.
통상적인 beam size는 1000-8000이고, 이 hyperparameter의 경우 본 모델에서 몇을 사용하였는 지는 정확히 밝히지 않고 있다.

두 모델을 합치는 방식 하고 HMM이랑 비터비 탐색 해야