음성 인식 오류 수정을 위한 Trie 기반 사전을 이용한 Guided Sequence Generation

최준휘, 류성한, 유환조, 이근배
포항공과대학교
(chasunee, ryush, hwanjoyu, gblee)@postech.ac.kr

Guided Sequence Generation using Trie-based Dictionary for ASR Error Correction

Junhwi Choi, Seonghan Ryu, Hwanjo Yu, Gary Geunbae Lee
Pohang University of Science and Technology

요 약

현재 나오는 많은 음성 인식기가 대체로 높은 정확도를 가지고 있더라도, 음성 인식 오류는 여전히 빈번하게 발생한다. 음성 인식 오류는 관련 어플리케이션에 있어 많은 오동작의 원인이 되므로, 음성 인식 오류는 고쳐져야 한다. 본 논문에서는 Trie 기반 사전을 이용한 Guided Sequence Generation을 제안한다. 제안하는 모델은 목표 단어와 그 단어의 문맥을 Encoding하고, 그로부터 단어를 Character 단위로 Decoding하며 단어를 Generation한다. 올바른 단어를 생성하기 위하여, Generation 시에 Trie 기반 사전을 통해 유도한다. 실험을 위해 모델은 영어 TV 가이드 도메인의 말뭉치의 음성 인식 오류를 단순히 Simulation하여 만들어진 말뭉치로부터 훈련되고, 같은 도메인의 음성 인식 문장과 결과로 이루어진 병렬 말뭉치에서 성능을 평가하였다. Guided Generation은 Unguided Generation에 비해 14.9% 정도의 오류를 줄였다.

주제어: 음성 인식 오류 수정, 딥러닝, RNN, LSTM

1. 서론

현재의 많은 음성 인식기가 대체로 높은 정확도를 가지고 있으나, 음성 인식 오류는 여전히 빈번히 발생한다. 음성 인식기를 이용한 많은 응용 프로그램들이 개발되어 나왔으나, 음성 인식 오류로 인해 발생하는 수많은 오동작은 여전하다. 많은 오류 수정 방법론이 제안되었으나 대체적으로 음성 인식 결과 문장과 그의 정답 문장으로 이루어진 말뭉치를 바탕으로 훈련된 모델들이 제안되었다.[1][2] 해당 병렬 말뭉치가 필요한 것은 기존 방법론들의 큰 단점 중의 하나인데 여러 문제점을 가지고 있다. 우선적으로 병렬 말뭉치는 얻기 힘들며, 해당 말뭉치는 그 음성 인식기가 사용된 환경과 그 음성 인식기에 종속되어 있다. 따라서 해당 말뭉치로 훈련된 모델은 환경이 다르거나 음성 인식기가 바뀌었을 때 적용하기 힘들다는 단점이 있다. 그러므로 새로운 음성 인식기를 이용할 때모델을 다시 훈련해야 하나, 다시 훈련하기 위

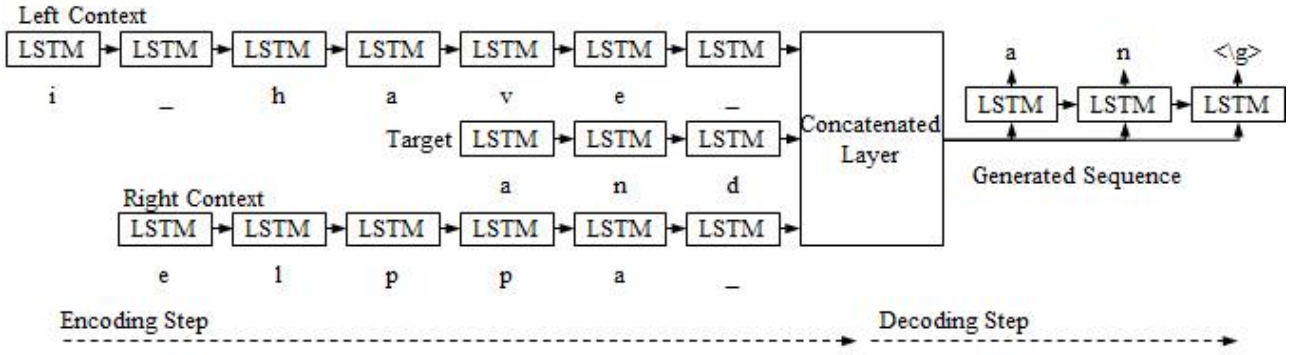

그림 1 제안한 Multiple Sequence to Sequence Generation 구조의 예

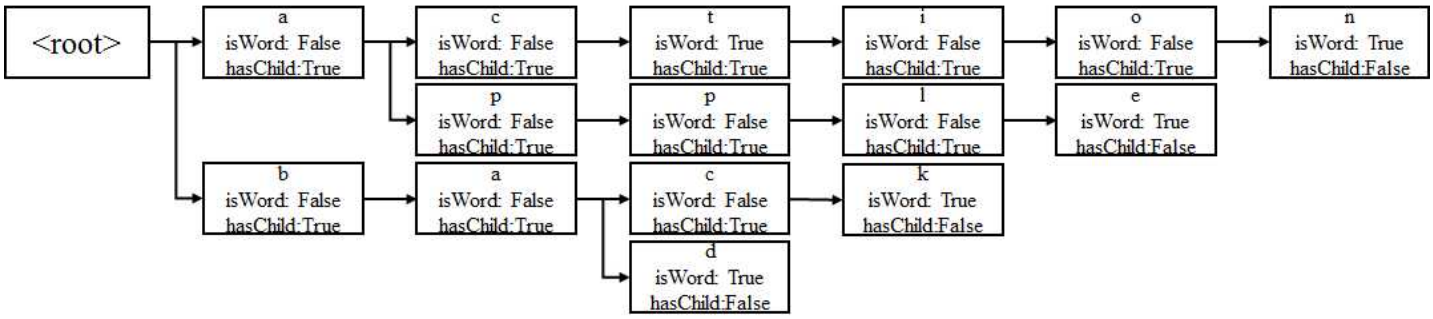

그림 2. Trie 기반 사전의 예

해 말뭉치를 재생성할 때 음성 말뭉치가 아니면, 그 또한 어렵다. 따라서 본 방법론은 병렬 말뭉치 필요 없이, 간단한 simulation된 말뭉치로도 효과적인 방법론을 제안코자 한다.

많은 자연어 처리 분야에서 딥러닝 기반의 방법론이 적용되어 state-of-the-art의 성능을 내고 있다. 자연어 데이터는 일반적으로 sequential 데이터이기 때문에 sequential 데이터에 효과적인 recurrent neural network (RNN) 기반 방법론들이 적용되고 있다. 그러나 단순한 RNN은 sequence가 길어짐에 따라 그 효과가 약하며 훈련시 굉장히 오랜 시간을 요구한다. 그에 따라 최근에는 long short-term memory (LSTM) [3] 이나 gated recurrent unit (GRU) [4]의 방법론이 적용되곤 한다. 그러나 여전히 문제는 남아있는데, 이는 자연어 처리를 위한 자질들이 word level로 접근하고 있음에 발생하는 문제이다. Word level로 접근하기 때문에 입력 vector의 dimension은 vocabulary의 size가 되고 이는 모델 훈련에 큰 문제 중의 하나이다. 이를 해결하기 위해 다양한 word embedding 방법론 [5][6]이 개발되었지만, 여전히 입력 vector의 차원은 word level이다. 따라서 최근에는 word level이 아닌 character level의 접근이 늘어나는 추세이다. 본 논문은 word level의 접근이 아닌 character level로 접근하여 RNN 기반의 방법론을 적용한 sequence generation framework를 적용하여 word를 character 단위로 생성하여 음성 인식 문제를 해결하고자 제안한다. 그러나 마지막 문제가 남아 있는데, 이는 invalid sequence의 생성이다. 음성 인식 오류의 수정이라고 함은 수정된 결과는 적어도 존재하는 단어여야 한다. 그러나 RNN 기반의 방법론을 적용한 sequence generation은 빈번히 invalid sequence를 생성하며, 이는 곧 non-word이다. 따라서 음성 인식기 응용프로그램에 따라서는 도리어 음성 인식 오류 수정 전보다 못한 결과를 내기도 한다. 이러한 문제를 해결하기 위해 본 논문에서는 RNN 기반의 방법론을 적용한 sequence generation을 이용하면서도 invalid sequence를 생성하지 않게 하기 위한 guide를 두는 방법론을 제안한다.

2. 방법론

2.1. LSTM Network를 이용한 Multi Sequence to Sequence Generation

제안한 방법론은 encoding 단계와 decoding 단계의 두 단계로 이루어져 있다. encoding 단계에서 모델은 목표 (음성 인식 오류) 단어와 그 단어의 왼쪽 문맥과 오른쪽 문맥이 하나의 vector로 encoding 된다. 예를 들어 'I_have_and_apple' [1]이라는 문장이 있을 때, 검출된 오류, 즉 목표 단어는 'and' 이고, 왼쪽 문맥은 'I_have_', 오른쪽 문맥은 '_apple' 이다. 목표 단어와 왼쪽 문맥은 정방향으로 encoding되나, 오른쪽 문맥은 목표 단어와 가까운 character일수록 더 중요하다고 판단하여, 역방향으로 encoding되었다. 각 sequence의 마지막 character가 encoding되면, 생성된 각각의 vector를 concatenation하여, 하나의 vector로 만들고, 그 vector를 입력으로 하여 decoding 단계를 진행한다. decoding 단계에서는 encoding 단계에서 만들어진 vector와 직전의 출력 character를 바탕으로 새로운 character를 generation하며, generation end symbol인 '</g>' 가 나올 때까지 반복한다. 첫 번째 character를 generation 할 때에는 입력으로 들어가게 되는 직전의 출력 character vector는 zero vector를 이용한다. Generation end symbol이 나오면, 출력 결과를 목표 단어와 바꾸어 수정을 완료한다. (그림 1)

제안한 방법론을 보다 자세히 설명하면, encoding 단계에서는 encoding 단계의 t번 째 출력인 h^t를 생성하기 위해 character의 one-hot vector인 w_t^T로부터 아래와 같이 진행된다.

$$x_t = Ww_t^T + b_x, \quad (1)$$

$$i_t = \sigma(V_i x_t + U_i h_{t-1} + b_i), \quad (2)$$

$$f_t = \sigma(V_f x_t + U_f h_{t-1} + b_f), \quad (3)$$

$$o_t = \sigma(V_o x_t + U_o h_{t-1} + b_i), \quad (4)$$

$$c_t = i_t \tau(V_c x_t + U_c h_{t-1} + b_c) + f_t c_{t-1}, \quad (5)$$

1) '_' 문자는 공백을 표시하기 위해 사용하였다.

알고리즘 1. Trie 기반 사전을 이용한 guided generation

procedure TreeDictionaryBasedGuidedGeneration(T,h^c)
  y_0^{symbol}←'<g>', y←"", n←1
  WordAccomplished←**False**
  WordNone←**False**
  PositionDown←**False**
  while not WordAccomplished **do**
    L←∅
    y_{n-1}^{onehot}←$Onehot(y_{n-1}^{symbol})$
    x_n^d←$[h_c y_{n-1}^{onehot}]$
    generate y_n with w_n^d
    generate L with y_n
    $y_n^{1st\ Symbol}$← highest scored c at n in L
    y_n^{symbol}←$y_n^{1st\ Symbol}$
    while not WordNone **do**
      $threshold$←$Score(y_n^{1st\ Symbol}) \times thresholdingFactor$
      if not PositionDown **then**
        if y_n^{symbol} ='<\g>' **then**
          if $y \in T$ **then**
            WordAccomplished←**True**
            break
        else if $y \oplus y_n^{symbol}$ hasChild in T **then**
          y←$y \oplus y_n^{symbol}$, n←$n+1$
          break
      y_n^{symbol}←next scored c at n in L
      PositionDown←**False**
      if $Score(y_n^{symbol}) < threshold$ **then**
        n←$n-1$
        delete last character of y
        PositionDown←**True**
        if $n=0$ **then**
          WordAccomplished←**True**
          WordNone←**True**
    if *WordNone* **then**
      y←None
    return y

$$h_t = o_t \cdot \tau(c_t) \quad (6)$$

x_t는 character의 one-hot vector인 w_t^T에 weight matrix W를 곱하여 나오는 character embedding이며, i_t는 LSTM의 input gate, f_t는 forget gate, o_t는 output gate, c_t는 LSTM의 memory cell state이다. V와 U들은 각각의 gate와 state의 weight matrix들이며, b들은 bias들이다. 내부 활성 함수 σ는 hard sigmoid 함수를 사용하였고, 활성 함수 τ는 hyperbolic tangent 함수를 사용하였다.

Decoding 단계에서는 n번 째 입력 vector인 x_n^d는 encoding 단계에서의 세 개의 출력 vector를 concatenation한 h_c와 $n-1$번 째 generation의 출력 character의 one-hot vector인 y_{n-1}^{onehot}를 concatenation 하여 구성한다. 식으로 표현하자면,

$$h_c = [h^{left\ context}\ \ h^{target\ error}\ \ h^{right\ context}], \quad (7)$$

$$x_n^d = [h^c\ \ y_{n-1}^{onehot}] \quad (8)$$

으로 표현할 수 있다. 그 후 decoding LSTM의 n번째 generation의 출력 vector h_n^d은 아래와 같이 계산된다.

$$i_n^d = \sigma(V_{i^d}x_n^d + U_{i^d}h_{n-1}^d + b_{i^d}), \quad (9)$$

$$f_n^d = \sigma(V_{f^d}x_n^d + U_{f^d}h_{n-1}^d + b_{f^d}), \quad (10)$$

$$o_n^d = \sigma(V_{o^d}x_n^d + U_{o^d}h_{n-1}^d + b_{o^d}), \quad (11)$$

$$c_n^d = i_n^d\tau(V_{c^d}x_n^d + U_{c^d}h_{n-1}^d + b_{c^d}) + f_n^d c_{n-1}^d, \quad (12)$$

$$h_n^d = o_n^d \bullet \tau(c_n^d) \quad (13)$$

i_n^d는 LSTM의 input gate, f_n^d는 forget gate, o_n^d는 output gate, c_n^d는 decoding LSTM의 memory cell state 이다. 이로부터 최종적으로 n번 째 출력인 y_n을 다음과 같이 계산할 수 있다.

$$y_n = \varphi(V_y h_n^d + b_y) \quad (14)$$

활성 함수 φ는 softmax 함수이다. 결과적으로 y_n는 각각의 character의 확률 분포를 나타낸다고 할 수 있다. 모든 weight matrix들은 일반적인 역전파 알고리즘으로 학습된다.

2.1. LSTM Network를 이용한 Multi Sequence to Sequence Generation

LSTM을 이용한 multi sequence to sequence generation의 한계점은 invalid sequence, 즉, non-word가 생성된다는 것이다. 학계에서는 좀 더 정확한 generation을 위해 generation을 guide하는 방법론이 여럿 제안되어 왔다 [7][8][9]. 그러나 해당 방법론들은 여전히 invalid sequence가 generation되는 것을 막을 수 없기 때문에 좀 더 명확한 guide가 필요하다. 따라서 trie 기반 사전을 이용한 guided generation을 제안한다.

제안하는 방법의 trie 기반 사전은 단어를 character level로 저장한다. 해당 사전은 character node가 연결된 방식으로 구성되어 있고, 각각의 character node는 'hasChild' 변수와 'isWord' 변수로 구성되어 있다. 만약 어떤 node가 child node를 가지고 있을 때 'hasChild' 변수는 'True' 이다. 또한 만약 해당 node가 존재하는 어떤 단어의 마지막 character일 경우 'isWord' 변수는 'True' 이다. 예로 그림 2는 단어 'act', 'action', 'apple', 'back', 'bad'를 저장하고 있는 trie 기반 사전을 보여준다.

위와 같이 만들어진 사전 T를 바탕으로 제안하는 guided generation 방법론은 알고리즘 1으로 설명할 수 있다. $Onehot$ 함수는 character로부터 해당character의 one-hot vector를 생성해주는 함수이며, $Score$ 함수는 위치 n에서의 어떤 character의 확률을 출력해주는 함수이다. L은 생성된 character들과 그 character들의 확률로 이루어진 집합이다. 알고리즘 1에 따르면 rejection threshold가 sequence y에 n번째 위치에 삽입될지 안 될지를 결정하는데 필요한데, 그 이유는 근거 있는 생성이 보장되어야하기 때문이다. 해당 threshold는 위치 n에 따라 동적으로 변해야 되는데, 이는 최종 출력을 위한 활성함수가 softmax 함수이기 때문이다. 수식은 다음과 같다.

$$threshold = Score(y_n^{1st\,Symbol}) \times tf \quad (15)$$

여기서 $y_n^{1st\,Symbol}$는 위치 n에서의 가장 확률이 높은 character이며, tf는 threshold factor이다. threshold factor에 따라 생성되는 결과는 다를 수 있다. 마지막으로 출력 단어가 **None**일 경우에는 생성에 실패했다는 의미이며, 이는 마땅한 단어를 찾을 수 없다는 의미로 목표 오류를 수정하지 않는다. 이는 삽입 오류에 대한 처리와는 다르며 삽입 오류는 아무 character가 generation되지 않은 채 바로 <\g>가 generation이 되었을 때 빈 단어로 교체되며, 처리되는 것이다.

3. 실험

3.1. 실험 상세

3.1.1. 훈련 말뭉치

훈련 말뭉치로 정제된 영어 TV 가이드 대화 말뭉치를 이용하였다. Encoding 단계의 입력 layer의 dimension을 줄이기 위하여, 모든 단어는 소문자로 normalize되었으며, 숫자는 모든 숫자가 0으로 치환되었고, 알파벳과 숫자가 아닌 모든 기호는 공백 문자로 치환되었다. 결과적으로 해당 dimension은 알파벳 문자 26개, 숫자 1개, 공백 1개, none symbol, start symbol, end symbol으로 이루어진 31 dimension으로 구성되었다. Decoding 단계의 출력 dimension 또한 이와 같다.

올바른 훈련 말뭉치 생성을 위해, 해당 말뭉치에서 음성 인식 오류를 simulation하였다. 각각의 문장의 첫 단어에서부터 끝 단어까지 숫자를 포함한 단어가 아닌 단어에 한해 해당 단어는 다른 임의의 단어로 랜덤하게 치환시켜 구성했다. 삽입 오류와 삭제 오류의 simulation을 위해 임의의 공백 위치에 임의 단어를 삽입하거나, 단어를 삭제하기도 하였다. 음성 인식 오류의 특성을 고려하면 오류는 해당 오류의 정답과 phonetically 유사하다. 따라서 좀 더 음성 인식 오류에 가까운 simulation을 위해 CMUDICT[10]를 이용하여, 발음열 유사도가 0.7 이상인 단어로 랜덤하게 치환하도록 하였다. 이러한 방법으로 약 29,000 문장에서 약 155,000 개의 데이터를 생성하였다.

3.1.2. 실험 말뭉치

실험을 위해서는 음성 인식 결과 문장과 그 문장의 정답 문장으로 이루어진 쌍으로 구성된 병렬 말뭉치가 필

표 1. Threshold factor tf에 따른 제안한 방법론의 WER

tf	Training epoch (%)		
	50	100	150
0.1	7.47	7.42	7.30
0.2	7.47	7.42	7.30
0.3	7.47	7.42	7.30
0.4	7.47	7.43	7.32
0.5	7.50	7.39	7.24
0.6	7.49	7.36	7.20
0.7	7.35	7.28	7.10
0.8	**7.25**	**7.23**	**7.05**
0.9	7.31	7.25	7.15

요하다. 훈련 말뭉치와 같은 영어 TV 가이드 대화 말뭉치를 준비했으며, 약 4,700 문장 쌍으로 구성되었다. 음성 인식 문장은 30만 단어급 언어 모델을 가진 음성 인식기로부터 생성되었으며, Word Error Rate (WER)은 약 8.29%로 전체 오류의 89.3%는 대체 오류, 7.7%는 삭제 오류, 4.0%는 삽입 오류이다.

3.1.3. 구현 환경

모든 구조는 Python 기반 딥러닝 라이브러리인 KERAS [11]를 통해 구현되었으며, 해당 라이브러리는 Google Tensorflow [12]와 Theano [13] 기반으로 작동된다. (어떤 라이브러리를 back-end로 사용할지는 옵션으로 선택 가능하다.) 모델은 nVidia GeForce GTX Titan X를 이용하여 훈련되었다.

3.1.4. 파라메터 Setting

최적의 파라메터를 찾기 위해 grid search를 이용하였다. 그에 따라 input embedding의 dimension은 128이며, output embedding과 hidden layer의 dimension들 또한 같다. Optimizer로는 Adadelta[14]가 사용되었고, ρ는 0.95, ϵ는 $1e^{-8}$으로 결정하였다. Learning rate는 1이며 batch size는 256이다.

3.2. Threshold Factor에 따른 실험 결과

제안된 알고리즘에 따르면, guided generation 시에 rejection threshold가 필요하며 이는 성능에 영향을 미친다. 따라서 threshold factor에 따른 성능 변화를 측정하였다. (표 1)

결과적으로 threshold factor가 0.8 일 때 전체 오류의 14.9%를 줄이며 가장 좋은 성능을 확보했다. Threshold factor가 0.3 이하 일 때는 모두 같은 성능을 보였는데, 이는 0.3이하의 threshold factor는 character가 generation될 때에 해당 character의 적절성은 고려치 않고 단순히 non-word가 되는 것만을 피하는 형태로 적용되었음을 의미한다. 이러한 결과에 따르면 적절한 threshold factor는 음성 인식 오류 수정 성능에 큰 영향을 미치는 것으로 보인다.

표 2. 각 항목의 WER 성능

			WER (%)
No correction			8.29
Generation w/o guidance	Epoch	50	7.59
		100	7.58
		150	7.52
Generation with guidance	Epoch	50	7.25
		100	7.23
		150	7.05

3.3. 일반적 성능

위 실험을 통해 threshold factor를 0.8로 정하고, 수정을 하지 않은 상태 (음성 인식기의 WER), guide를 하지 않은 generation, guided generation의 세 가지 결과를 비교하였다. (표 2)

결과에 따르면 guided generation이 가장 높은 성능을 기록하며 guide 하지 않았을 때의 결과보다 전체 오류의 14.9%를 줄이며 우수한 성능을 보였다. 이는 epoch가 150 일 때 guide 하지 않은 것의 출력 중의 74.8%가 non-word였기 때문이다. 한 가지 한계점은 guided generation 방법론이 guide 하지 않았을 때의 방법론과 비교했을 때 character level의 정확도는 더 떨어진다. 제안한 방법론은 정방향으로 진행하여 최적의 path를 찾는 것과 유사한 방법론이지만 이는 사실 sequence 전체로 보았을 때의 최적의 path는 아니라고 할 수 있다. 따라서 추후 연구로 먼저 decoding의 출력으로 character의 network를 구성하고 Viterbi search를 통해 단어를 구성할 때 단어 사전을 이용하여 non-word가 생성되지 않도록 하는 방법을 고려할 수 있을 것이다.

4. 결론 및 후속 연구

본 논문에서 우리는 음성 인식 오류 수정을 위한 trie 기반 사전을 이용한 guided sequence generation을 제안하였다. 이는 목표 오류와 그 주위의 문맥을 하나의 vector로 embedding하여 해당 embedding vector로부터 단어를 character 단위로 generation하는 방법론이다. 음성 인식 오류 수정이기 때문에 적어도 non-word가 발생치 않도록 하기 위하여 generation 시 guide하는 방법이 적용되었다. 본 방법론에 따라 단순하게 simulation된 훈련 말뭉치로 실제 발생한 오류 전체의 14.9%를 줄이는 성능을 확인할 수 있었다. 이 방법론을 바탕으로 전체 이는 단순히 음성 인식 오류뿐만 아니라, RNN을 응용한 generation의 고질적인 문제인 invalid sequence generation 문제를 해결하는 방법론을 제안하였고 등록된 sequence를 generation하는 많은 분야에서 응용될 수 있을 것이라 기대된다.

Acknowledgement

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0101-16-0307, 인간 수준의 평생 기계학습 SW 기초연구 (기계학습연구센터))

참고문헌

[1] Minwoo Jeong, Sangkeun Jung, and Gary Geunbae Lee, “Speech recognition error correction using maximum entropy language model”, In Proceedings of INTERSPEECH, pages 2137-2140, 2004.

[2] E Byambakhishig, K Tanaka, Ryo Aihara, Toru Nakashika, Tetsuya Takiguchi, and Yasuo Ariki, “Error correction of automatic speech recognition based on normalized web distance”, In Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[3] Felix A Gers, J¨urgen Schmidhuber, and Fred Cummins, “Learning to forget: Continual prediction with lstm”, Neural computation, 12(10):2451-2471, 2000.

[4] Kyunghyun Cho, Bart van Merri¨enboer, Dzmitry Bahdanau, and Yoshua Bengio, “On the properties of neural machine translation: Encoder-decoder approaches” arXiv preprint arXiv:1409.1259, 2014.

[5] Tomas Mikolov, Martin Karafi´at, Lukas Burget, Jan Cernock`y, and Sanjeev Khudanpur, “Recurrent neural network based language model”, In INTERSPEECH, pages 1045-1048, 2010.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, “Efficient estimation of word representations in vector space”, arXiv preprint arXiv:1301.3781, 2013.

[7] Xu Jia, Efstratios Gavves, Basura Fernando, and Tinne Tuytelaars, “Guiding the long-short term memory model for image caption generation”, In Proceedings of the IEEE International Conference on Computer Vision, pp 2407-2415, 2015.

[8] Philip Bachman and Doina Precup, “Data generation as sequential decision making”, In Advances in Neural Information Processing Systems, pages 3231- 3239, 2015.

[9] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia, “Abccnn: An attention based convolutional neural network for visual question answering” arXiv preprint arXiv:1511.05960, 2015.

[10] R Weide, The carnegie mellon pronouncing dictionary [cmudict. 0.7], 2014.

[11] Franc¸ois Chollet, Keras: Theano-based deep learning library, Code: https://github.com/fchollet, Documentation: http://keras. io, 2015.

[12] Mart ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”, arXiv preprint arXiv:1603.04467, 2016.

[13] Fr´ed´eric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio, “Theano: new features and speed improvements”, arXiv preprint arXiv:1211.5590, 2012.

[14] Matthew D Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701, 2012.