End to End Speech Recognition System

Seminar Report

Bachelors of Technology
in
Computer Science and Engineering

by

SAURABH GARG
(140070003)

under the guidance of

PROF. PREETHI JYOTHI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

March, 2017

Abstract

Speech Recognition is the task of transcribing the speech signal into equivalent text. While automatic speech recognition has greatly benefited from the introduction of neural networks, the networks are at present only a single component in a complex pipeline in state of the art systems.

In existing systems, the first stage of the pipeline is input feature extraction: standard techniques include melscale filterbanks. Neural networks are then trained to classify individual frames of acoustic data, and their output distributions are reformulated as emission probabilities for a hidden Markov model. The objective function used to train the networks is therefore substantially different from the true performance measure which is sequence-level transcription accuracy. This is precisely the sort of inconsistency that end-to-end learning seeks to avoid. In practice thus even is there is large gain in frame accuracy it translates to a negligible improvement, or even deterioration in transcription accuracy. An additional problem is that the frame-level training targets must be inferred from the alignments determined by the HMM. This leads to an awkward iterative procedure, where network retraining is alternated with HMM realignments to generate more accurate targets.

In this report, we will describe various models for sequence labelling task and the problem of labelling unsegmented sequence data. We will explain a end to end speech recognition system that directly transcribes the audio data with text/phonemes. The system tries to replace the conventional speech recognition pipeline by a single recurrent neural network (RNN) architecture. We have chosen the spectrograms as a minimal preprocessing scheme. The system is based on the combination of a deep bidirectional LSTM recurrent neural network architecture and the Connectionist Temporal Classification objective function.

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 30th March, 2017 Saurabh Garg

Place: IIT Bombay, Mumbai Roll No: 140070003

Acknowledgements

This report is an outcome of my work under the guidance of Prof. Preethi Jyothi on End to End Speech Recognition System. I am thankful to the people who have been instrumental in helping me out throughout this seminar project. First and foremost, I would like to express my sincere gratitude towards my supervisor Prof. Preethi Jyothi for her guidance. My research in End to End Speech Recognition System is being driven by her vision and support. It was immensely helpful in gaining the understanding required for writing this report.

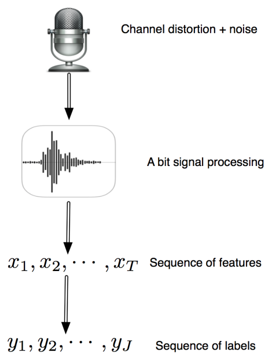
I would also like to thank Diptesh Kanojia for providing GPU access and helping me out to run the code on server.

Contents

Chapter 1

Introduction

According to Wikipedia, Speech recognition (SR) is the inter-disciplinary sub-field of computational linguistics that develops methodologies and technologies that enables the recognition and translation of spoken language into text by computers. Speech Recognition is the task of transcribing the speech signal into text.


Channel distortion + noise
A bit signal processing
$x_1, x_2, \cdots, x_T$ Sequence of features
$y_1, y_2, \cdots, y_J$ Sequence of labels

It is a typical sequence to sequence transduction problem. Given, $\mathbf{y} = \{y_1, y_2, ..., y_n\}$ the aim is to output $\mathbf{x} = \{x_1, x_2, ..., x_T\}$ which maximizes the $P(x|y)$.
In this chapter, we will explain the basics behind recurrent neural network and convolutional neural network, and how to train these networks.

1.1 Recurrent Neural Networks

While reading humans don't start thinking from scratch every time they read a word, instead they use the understanding from previous words.Normal neural networks cannot handle this, but recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to the next network. Thus in this way recurrent neural networks aren't at all different that a normal neural network.

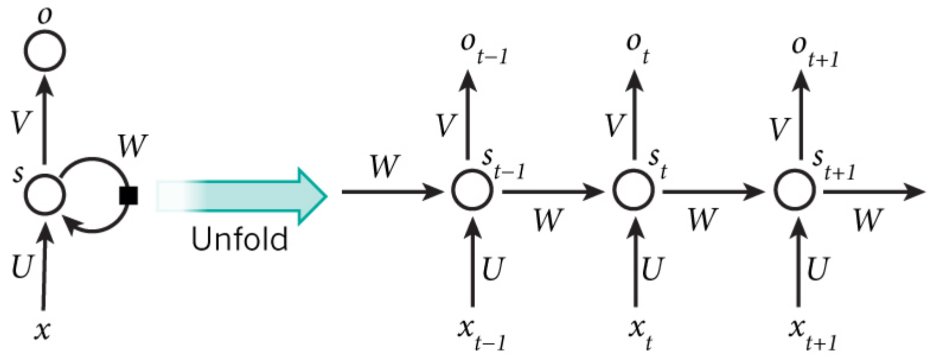


Figure 1.1: *Unrolled Recurrent Neural Network (Source: WildML)*

1.1.1 Problem of Long-Term Dependencies

There are two possible kinds of dependencies : long term and short term. Sometimes we only need to look at the recent information to perform a specific task. For example, if we are trying to predict the last word in "the sky is *blue*", we don't need further context and RNNs can learn to use the past information. But there are cases where we need more information. Consider trying to predict the last word in "I grew up in Mumbai..... I can speak fluent *Marathi*", recent information suggest that the next word is probably name of some language, but if we want to narrow down to which language, we need the context of Mumbai. Thus, it is always possible that the gap between the relevant information and point where it is needed become very large.
Unfortunately, as that gap grows, RNNs are less able to connect the information [1]. In thoery, RNNs are able to handle such "Long Term Dependencies", but in practice RNNs don't seem to be able to learn such information. The reason being the gradients explode or vanishes very quickly which doesn't allow inputs which are far apart to influence each other.

1.1.2 LSTM Networks

LSTMs are designed to avoid the long-term dependency problem [12]. Recurrent neural networks have repeating modules of simple structures such as a single tanh layer, whereas the repeating module in LSTM has a bit different structure, there are four neural network layers, interacting in a very special way as shown in [1.2].

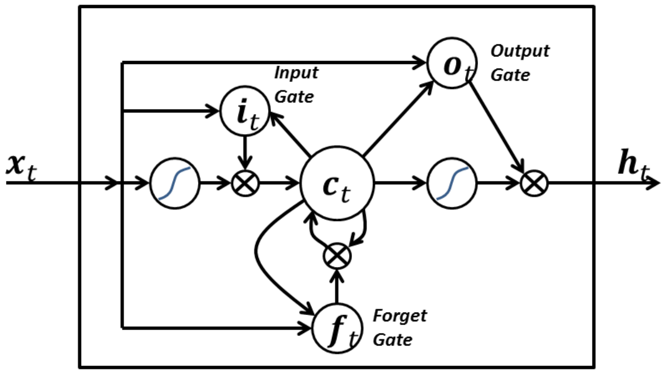
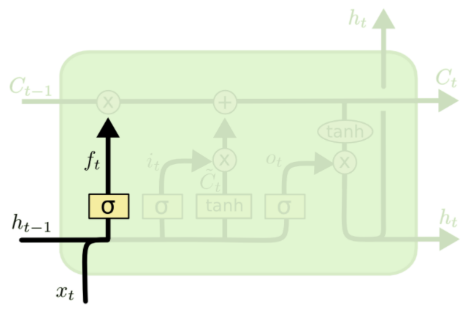


Figure 1.2: *A peehole Long Short Term Memory Block (Source: Wikipedia)*

In LSTMs, the cell state is like a conveyor belt. It is very easy for information to just flow along it unchanged, with some minor linear interactions. The added ability is to remove and add information to the cell state, which is regulated by gates.
Gates are a way to optionally let through the information. Generally, they are composed of neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, which describes how much of each component should be let through, where value zeros denotes "nothing let through" and a value of one means "let everything through".
A LSTM basically has four gates: Control gate, Forget Gate, Input Gate and Output Gate.

1.1.3 LSTM step-by-step walk through

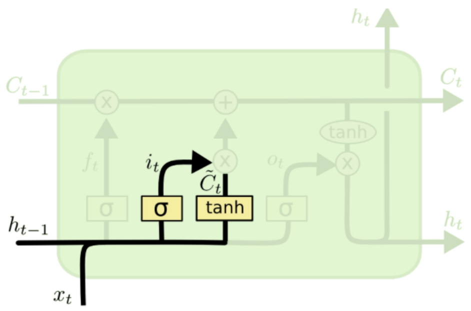
The first step in LSTM is to decide what information to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer".



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 1.3: *LSTM Forget Gate (Source: [12])*

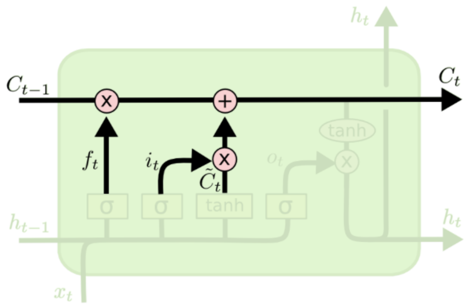
The next step is to decide what new information to store in the cell state. A sigmoid layer called the "input gate layer" decides which values to update. Next, a tanh layer creates a vector of new



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 1.4: *LSTM Input Gate (Source: [12])*

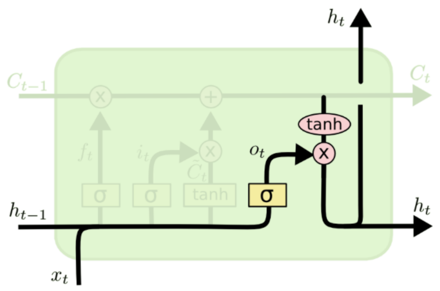
candidate values, $\hat{C}_t$ that could be added to the state.
Now, we have update the old cell state, C_{t1}, into the new cell state C_t. In language modelling, this is the crucial step where old information is dropped and new information is added.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 1.5: *LSTM Control Gate (Source: [12])*

Finally, output is based on the cell state, but a filtered version as explained above. A sigmoid layer decides what parts are going to the outputs. Then, cell state is passed through tanh and it is multiplied by the output of the sigmoid layer.



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 1.6: *LSTM Output Gate (Source: [12])*

1.2 Convolutional Neural Network

Convolution is a specialized kind of linear operation. Convolutional network are neural networks that use convolution in place of general matrix multiplication. Convolutional neural networks

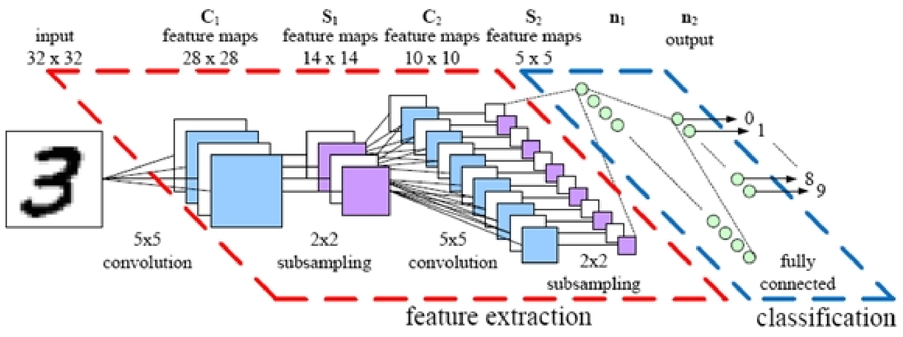


Figure 1.7: *Convolutional Neural Network architecture for digit recognition (Source: Parse)*

leverages three important ideas : sparse interactions, parameter sharing and equivalent representation.
Sparse interactions is accomplished by making kernel(matrix) smaller than the input. For example, the input image might have thousands of pixels but we can detect small, meaning full features such as edges with kernels that occupy only a fraction of number of input pixels.
Parameter sharing refers to using the same parameters for more that one function of the model. In CNN, as compared to neural nets where each element of the weight matrix is used exactly once when computing the output of a layer, each member of kernel is used at every position of the input. This means that instead of learning a different set of parameters for every location, it learns only one set of parameters.
Convolution has the property of equivariance to the translation i.e. due to the particular form of parameter sharing, if the input changes the output changes the same way.
After, getting the output of a CNN, in general a fully connected Neural Network is used to get the final output.

1.2.1 Pooling

Convolution Neural Network in general consist of three stages. In first stage, the layer performs several convolution to produce a set of linear activations. In the second stage, each linear activation is run through a non linear activation function, such as rectified linear activation function, this stage is known as detector stage. In the third stage, pooling function is used to modify and subsample the output of the layer. Some pooling functions that are rapidly used include max pooling which report the maximum within a rectangular box, L^2 norm of a rectangular box and average over a rectangular neighbourhood.

1.3 Training Neural Networks

In general, training a neural network requires propagation of gradient of the cost function.
To train a RNN, we need to back propagate each output sequence loss/cost in time which is known as backpropagation through time. Thus, as compared to general neural network when sequence length is long this may need to back propagate though many layers. In practice, we sometimes truncate the backpropagation in few steps.
To train a CNN, we have to backpropagate through basic layers namely softmax, fully connected network, pooling, ReLU and convolutional layer.

1.4 Organization of the report

In this chapter, we explained basics behind LSTMs, their need and the problem of long term dependency. We also discussed basics behind CNN. In later chapters, we will introduce various models for the task of sequence labelling which includes few attention and augmented memory models. We will discuss the problem of labelling unsegmented sequence data and how we can train a end to end system using segmental RNN and connectionist temporal classification. We will explain one end to end architecture for speech recognition task which uses connectionist temporal classification and some improvements over it. In the end, we will present a baseline implementation of a end to end system.

Chapter 2

Various Models for Sequence labelling

Sequence labelling is a pattern recognition task that involves the assignment of a categorical label to each member of a sequence of observed values. Most of the sequence labelling tasks are probabilistic in nature and finding the best sequence problem relies on statistical model. Most statistic models use the Markovian assumption i.e. the choice of labels for a particular word is directly dependent only on the adjacent labels and they form a Markov Chain.

In this chapter, we discussed various sequence labelling models based on Hidden Markov Model which are pipelined, RNN based end to end models which uses Attention and Augmented Memory.

2.1 Hidden Markov Model

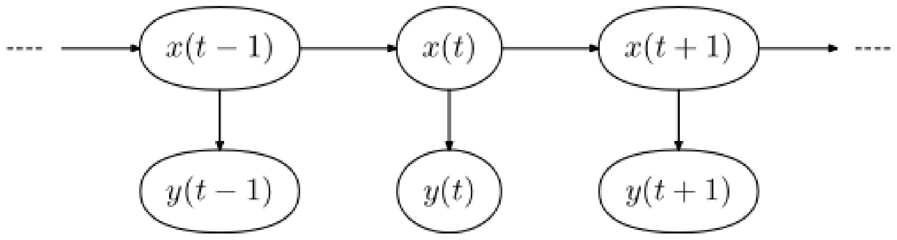


Figure 2.1: *Hidden Markov Model (Source: Wikipedia))*

A hidden markov model is a statistical markov model in which system modeled is assumed to be a marko process with hidden states. The task is to compute the probability of the output

sequence, given the parameters of the model. [13]
The probability of observing a sequence, $Y = y(0), y(1), ..., y(T)$ of length T is $P(Y) = \sum_x P(Y|X)P(X)$ where the sum is over all possible hidden node sequences $X = x(0), x(1), ..., x(T)$.

2.1.1 Three basic problems for HMMs

Problem 1: Evaluation
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, how to compute efficiently the probability of observation given the model i.e. $P(Y|\lambda)$. This evaluation problem is to compute the probability that observed sequence was produced from the model.

Problem 2: Decoding
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, how to chose the corresponding state sequence which maximises the probability of observed sequence. It uncovers the hidden part of the problem.

Problem 3: Learning
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, how to adjust model parameters so that it maximises probability of the observed sequence. It allows us to adjust model parameters so as to create best model for the given training sequence.

2.1.2 Solution to three problems

Forward Algorithm for Evaluation Problem
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, we want to compute efficiently the probability of observation given the model i.e. $P(Y|\lambda)$. The most obvious way is to find the solution by enumerating every possible state sequence of length T. However, with N hidden states and T observations the complexity of this solution is of the order $N^T T$ as there are N^T possible sequences. Instead we can use forward algorithm which uses dynamic programming to reduce complexity to $N^2 T$.

Forward Algorithm:
It uses the forward variable $\alpha_t(i)$ defined as,

$$\alpha_t(i) = P(y(0), y(1), ..., y(t), q_t = i|\lambda)$$

i.e. the probability of the partial observed sequence till $y(t)$ and in state q_i at time t given the model λ
Initialization:

$$\alpha_1(i) = \pi_i b_i(y(1)) \qquad 1 \leq i \leq N$$

Induction:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(y(t+1)) \qquad 1 \le t \le T-1, 1 \le j \le N$$

where a_{ij} is the transition probability from previous state q_i to state q_j and $b_j(y(t))$ is the state observation likelihood of the observation in $y(t)$ given state q_j. Finally, the required probability is the sum of the terminal forward variables $\alpha_T(i)$

$$P(Y|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Viterbi Algorithm for Decoding Problem
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, we want to chose the corresponding state sequence which maximises the probability of observed sequence i.e. $\text{argmax}_Q\, P(Y, Q|\lambda)$

Viterbi Algorithm:
It uses the probability of HMM being in the state q_j after seeing the subsequence of the observation and passing through the most probable state sequence $q_1, q_2, ...q_{t-1}$, $\delta_t(i)$ defined as,

$$\delta_t(i) = \max_{q_1, q_2, ..., q_{t-1}} P(y(0), y(1), ..., y(t), q_1, q_2, ..., q_t = i|\lambda)$$

Initialization:

$$\delta_1(i) = \pi_i b_i(y(1)) \qquad 1 \le i \le N$$

Induction:

$$\delta_{t+1}(j) = \max_i \delta_t(i) \alpha_{ij} b_j(y(t+1)) \qquad 1 \le t \le T-1, 1 \le j \le N$$

where a_{ij} is the transition probability from previous state q_i to state q_j and $b_j(y(t))$ is the state observation likelihood of the observation in $y(t)$ given state q_j. Finally, once the final state is reached corresponding state sequence can be found using back tracking.

Baum-Welch Algorithm for Learning Problem
Given the observed sequence $Y = y(0), y(1), ..., y(T)$ and model λ, we want to adjust model parameters so that it maximises probability of the observed sequence. There is no optimal way of finding the solution but there are some heuristic which try to find local optimal solution over global optimal solution. It uses EM algorithm to find the maximum likelihood estimate of the parameter of a hidden markov model.

As above, we define a backward variable $\beta_t i$ i.e. the probability of the partial observed sequence $y(t), y(t+1), ..., y(T)$ and in state q_i at time t given the model and it is calculated by similar recursions. Now, we define the probability of being in state q_i at time t and in state q_j at time t+1 given the model and observed sequence, is $\zeta_t(i,j)$

$$\zeta_t(i,j) = P(q_t = i, q_{t+1} = j|Y, \lambda)$$

$$\zeta_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(y(t+1))\beta_{t+1}(j)}{P(Y|\lambda)}$$

Also, we define the probability of being in the state q_i at time t, given the observation sequence and model, is $\gamma_t(i)$

$$\gamma_t(i) = \sum_{j=1}^{N} \zeta_t(i,j)$$

Using this we get a method to re-estimate parameters of HMM.

$$\bar{\pi} = \gamma_1(i)\text{which is expected number of times in the state } q_i \text{ at time t=1}$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } q_i \text{ to } q_j}{\text{expected number of transitions from state } q_i} = \frac{\sum\limits_{t=1}^{T} \zeta_t(i,j)}{\sum\limits_{t=1}^{T} \gamma_t(i)}$$

$$\bar{b}_i(l) = \frac{\text{expected number of times in state } q_i \text{ and observing symbol } v_l}{\text{expected number of transitions from state } q_i} = \frac{\sum\limits_{t=1\&y(t)=v_k}^{T} \gamma_t(i)}{\sum\limits_{t=1}^{T} \gamma_t(i)}$$

These steps are repeated till convergence upto desired level is achieved.

2.1.3 Speech Recognition and HMM

In training mode large amount of data is given to HMM model and HMM learns transition matrix and its probability distribution. For Recognition, unknown predicted word is fed to HMM and its output probability is calculated.

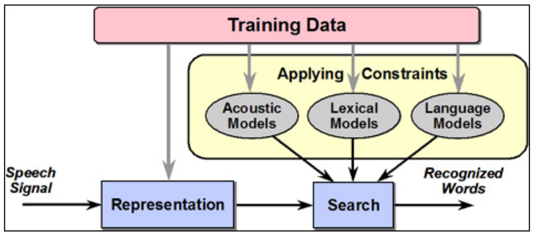


Figure 2.2: *Block Diagram for Speech Recognition (Source: MIT OCW))*

2.2 Attention and Augmented Recurrent Neural Network

As simple RNN struggles with long term depencies [11], LSTMs tend to work a lot better. Now, there are growing attempts to augment RNNs with new properties.

2.2.1 Attention Interfaces

Humans generally tend to be more attentive to things they are presently doing, keeping memories in the past. Similar behaviour can be achieved from a neural network using attention, focusing on a subset of the information they are given. We would like the attention to be differentiable, so that we can learn where to focus and train it end-to-end efficiently with gradient descent. The attention distribution is generally generated with content-based attention. The idea is attending RNN generates a query describing the focus. Items are dot product with the query which gives a similarity measure i.e. how well it matches with the query, to generate attention distribution a softmax is taken over these scores.

Attention based recurrent sequence generator
Attention based RNN [3] have been applied to wide variety of tasks such as handwriting synthesis, machine translation and image caption generation.
In image caption generation [15], they used a RNN + CNN based model. This allows RNN to look at different positions of an image at every time step. CNN extracts high level features. and RNN generates description by focusing on relevant parts of CNN output.
An attention based recurrent sequence generator(ARSG) is a recurrent neural network that stochastically generates an output sequence $(y_1, y_2, ...y_T)$ from an input x. Input x is processed by an encoder to generate sequence $(h_1, h_2, ...h_L)$ which is more suitable for attention to work with.

At i^{th} step, an ARSG generates an output y_i as,

$$\alpha_i = Attend(s_{i-1}, \alpha_{i-1}, h)$$

$$g_i = \sum_{j=1}^{L} \alpha_{i,j} h_j$$

$$y_i \approx Generate(s_{i-1}, g_i)$$

where s_{i-1} is $(i-1)^t h$ state of the RNN which we refer to as generator. $\alpha_i \in R^L$ is a vector of attention weights called alignment. g_i is a glimpse. The step is completed by computing a new state,

$$s_i = Recurrency(s_{i-1}, g_i, y_i)$$

LSTMs or GRUs are used as recurrent activation, which is refered as recurrency.
Context Based Attention

The above equation of α_i is very generic. If in the attention, we drop the term α_{i-1} from Attend arguments then it is content based attention mechanism.

$$e_{i,j} = Score(s_{i-1}, h_j)$$
$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum\limits_{i=1}^{L} exp(e_{i,j}}$$

Location Based Attention
Instead, if in the attention, we drop the term h_i from Attend arguments then it is location based attention mechanism.

$$e_{i,j} = Score(s_{i-1}, h_j)$$
$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum\limits_{i=1}^{L} exp(e_{i,j}}$$

Location based attention mechanism is used for handwriting synthesis.

In speech recognition, hybrid mechanism is used. The idea is, we would like attention model that uses the previous alignment to select a short list of elements from h and then content based attention will select the relevant ones.

2.2.2 Listen, Spell and Attend Model

Listen, Attend and Spell (LAS) [2] system has two components a Listener and a Speller. The listener is a pyramid recurrent neural network encoder that accepts filter bank spectra as input. The speller is an attention based recurrent neural network decoder that emits character as outputs. The network produces character sequences without making any independent assumption between characters.

Encoder
Encoder RNN maps the sequential variable length input into fixed vector.

Decoder
Decoder RNN uses the fixed vector to produce variable length output sequence, one token at a time.

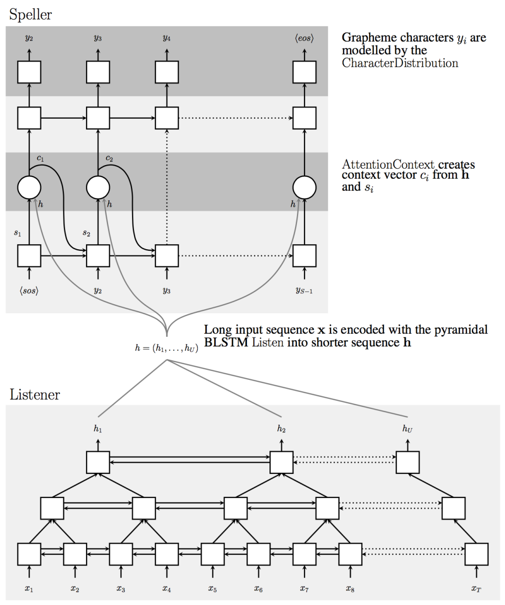
Attention vector
These are skip connections that allow information and gradients to flow effectively. At each output step, the last decoder RNN state is used to generate an attention vector over input of encoder. The attention vector is used to propagate information from encoder to decoder at every time step.
Let $x = (x_1, x_2, ...x_T)$ be input sequence of filter bank features and let $y = (y_1, y_2, ...y_T)$ where each $y_i \in$ english alphabates. We want to model

$$P(y|x) = \prod_i P(y_i|x, y_{<i})$$

Listener is an acoustic model whose operation is to listen and speller is an attention based character decoder whose key operation is AttendAndSpell.

$$h = Listen(x)$$

$$P(y|x) = AttendAndSpell(h, y)$$



Speller
Grapheme characters y_i are modelled by the CharacterDistribution
AttentionContext creates context vector c_i from $\mathbf{h}$ and s_i
Long input sequence $\mathbf{x}$ is encoded with the pyramidal BLSTM Listen into shorter sequence $\mathbf{h}$
$h = (h_1, \ldots, h_U)$
Listener

Figure 2.3: *Listen Attend and Spell Model (Source: LAS [2]))*

Listener
It is a BLSTM with pyramid structure. Requirement of pyramid structure is, linear BLSTM RNN generates lots of outputs and speller finds it hard to converge because then it has to extract relevant information from large number of input time steps. In deep BLSTM architecture, the output at the i^{th} time step, from j^{th} layer is,

$$h_i^j = BLSTM(h_{i-1}^j, h_i^{j-1})$$

In deep pyramidal BLSTM, we concatenate the outputs before feeding it to next layer,

$$h_i^j = pBLSTM(h_{i-1}^j, [h_{2i}^{j-1}, h_{2i+1}^{j-1}])$$

This helps in three ways: i) Reducing the time complexity, ii) Helps in learning the non-linear features, and iii) Extract relevant features from smaller number of time steps.

Attend & Spell
Uses an attention based LSTM transducer. At every output step, the transducer produces the

probability distribution over the next character given previously seen characters. The distribution is dependent on the decoder state s_i and context c_i.

$$c_i = AttentionContext(s_i, h)$$

$$s_i = RNN(s_{i-1}, c_{i-1}, y_{i-1})$$

$$P(y_i|x, y_{<i}) = CharacterDistribution(s_i, c_i)$$

Learning
The Listener and Speller can be trained jointly end to end for speech recognition. It maximises the log probability:

$$\max_\theta \sum_i \log P(y_i|x, y^*_{<i}, \theta)$$

where y^* is the ground truth. But during the inference ground truth is missing and the predictions can suffer because the model is not resilient to feeding the bad predictions at some time step. To make it better, during training, instead of always feeding the ground truth transcriptions we sample from previous character distribution and use that as input in the next step predictions

$$\tilde{y}_i \sim CharacterDistribution(s_i, c_i)$$

$$\max_\theta \sum_i \log P(y_i|x, \tilde{y}_{<i}, \theta)$$

Decoding and Re-scoring
During inference, we want to find the most likely character sequence given the input sequence

$$\hat{y} = \operatorname*{argmax}_y \log P(y|x)$$

Decoding is performed simply using beam search algorithm.

2.2.3 Adaptive Computation Time

RNNs in general do same amount of computation each time step. Adaptive computation time [5], is an algorithm that allows RNN to learn how many computation steps to take between receiving an input and emitting an output. The idea is at each time step allow RNN to do multiple number of steps which is variable and learnt. Again, as above models we want to learn number steps and hence, it has to be differentiable. We achieve this using the same trick, instead of running for a discrete number of steps, we have a attention distribution over number of steps to run.
Adaptive computation time works by modifying the conventional setup by allowing a recurrent neural network to perform variable number od transitions and compute variable number of

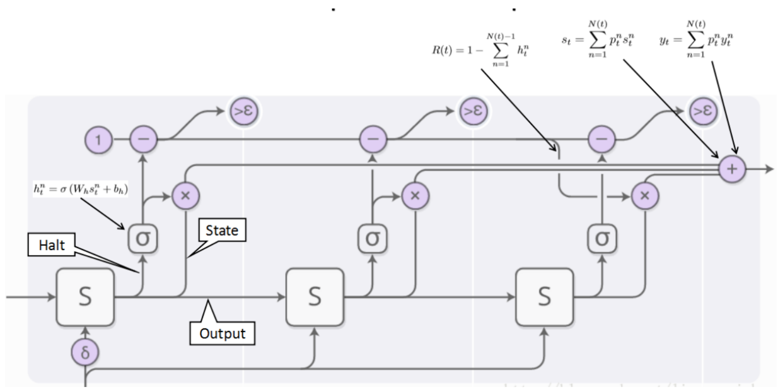

$R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$
$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n$
$y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n$
$h_t^n = \sigma(W_h s_t^n + b_h)$
Halt
State
Output
S
σ
δ
>ε
1

Figure 2.4: *Adaptive Computation Time Architecture (Source: Distill [11]))*

outputs, let it be N(t), at each time step, t.
The intermediate state and output sequence is defined as,

$$s_t^n = \begin{cases} S(s_{t-1}, x_t^1) & \text{if n = 1} \\ S(s_t^{n-1}, x_t^n) & \text{otherwise} \end{cases}$$

$$y_t^n = W_y s_t^n + b_y$$

where $x_t^n = x_t + \delta_{n,1}$ is the input at time t. It is augmented with a delta binary function which helps network to distinguish between repeated inputs and repeated computations of same input. An extra halting unit h is added to the network output to determine number of updates performed by the RNN,

$$h_t^n = \sigma(W_h s_t^n + b_h)$$

The halting probability p_t^n of the intermediate steps is,

$$p_t^n = \begin{cases} R(t) & \text{if n = N(t)} \\ h_t^n & \text{otherwise} \end{cases}$$

$$N(t) = \min\{n_1 : \sum_{n=1}^{n_1} h_t^n \geq 1 - \epsilon\}$$

and the reminder R(t) is deifned as

$$R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$$

and the ϵ is to allow the computation to stop even after one update otherwise minimum of two updates would have always been required. The s_t and y_t are then simply computed as below,

$$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n$$

$$y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n$$

Now, we will discuss two interesting models viz. Neural Turing Machines and Neural Programmers which are not much explored for sequence labelling task but these models have great potential to perform nicely on the sequence labelling task.

2.2.4 Neural Turing Machine

Neural Turing Machines combine a RNN with an external memory bank [8]. They extend the capabilities of neural network by coupling them to external memory, which they can interact with by attention process. The combined system is analogous to Turing Machine but is differentiable end-to-end with respect to the location we read from and write to, allowing it to be trained efficiently with gradient descent.

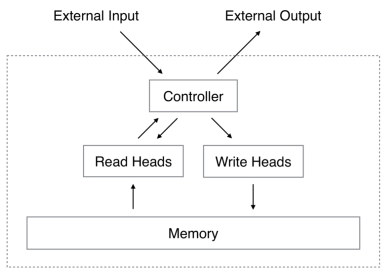


Figure 2.5: *Neural Turing Machine Architecture (Source: NTM [8]))*

The solution to make it differentiable is to read and write everywhere everytime just to different extents.

Reading
Instead of specifying a single location to read from, it gives a *attention distribution*. The differentiability is achieved by defining "blurry read" operation that interact to a greater or lesser degree with all elements in the memory. The degree of blurriness is determined by attention. Thus, each read operation is a weighted sum.

$$r_t \leftarrow \sum_i w_t(i) M_t(i)$$

given all weightings are initially normalized i.e. $\sum_i w_t(i) = 1$ and $0 \leq w_t(i) \leq 1 \quad \forall i$. Clearly r_t is differentiable with respect to memory and weights.

Writing
Similar to reading we write everywhere but to different extents and attention distribution governs

the amount to be written. Write is decomposed into an erase followed by an add. In summary, the new value of a position in memory be a convex combination of old memory content and the write value.

$$M_t(i) \leftarrow M_{t-1}(i)(1 - w_t(i)e_t) + w_t(i)a_t$$

These weighting arises by combining two techniques which provide complementary facilities:
Content based Attention
This mechanism focuses attention based on locations based on the similarity of the current value and values emitted by the controller.
Location based Attention
This mechanism is based on the relative movement in the memory which enables NTM to loop.

Content based attention is strictly more general than location based attention as the content of memory location can include location information inside it.
This capabilities allows NTMs to do things which were beyond the capabilities of a simple Neural Network like they can learn to mimic a lookup table, or even sort numbers.

2.2.5 Neural Programmer

Neural Programmer learns to create programs in order to solve a specific task. It learns to generate such programs without needing the examples of correct programs. The supervision that is required is only the correct solution that it should produce. One operation is predicted at each time step by a controller RNN i.e. the controller RNN outputs a probability distribution over next operation. Now to make it differentiable, the usual trick is used as above i.e instead of running a single operation at each time step, the attention trick of running all of the operations is incorporated and the output is averaged out outputs together with the probability of that operation.

Chapter 3

Labelling Unsegmented Sequence Data

Sequence learning tasks require the predictions of sequence of labels from unsegmented noisy data. RNNs are powerful sequence learners but they require pre-segmented data and post-processing to transform into label sequence. Segmented RNN [9] and CTC [4] gives us novel ways for RNN to train end to end for the unsegmented data. The idea of segmental RNN is to automatically construct segment level features and the idea of CTC is to interpret the network outputs as probability distribution over all possible labelling given the input sequence.

3.1 Segmental Recurrent Neural Network

Conditional random fields is a graphical model whose nodes can be divided in to exactly two disjoint sets, one a set of observations $(x_1, x_2, ..., x_T)$ and second a set of predicted labels $(y_1, y_2, ..., y_T)$. CRF models,

$$P(y_{1:T}|x_{1:T}) = \frac{1}{Z(x_{1:T})} \prod_j \exp(w^T \phi(y_j, x_{1:T}))$$

where Z is the normalization constant and w is a weight vector.
Semi Markov or Segmental conditional random fields on an input sequence $(x_1, x_2, ..., x_T)$, outputs a segmentation of x, in which labels are assigned to each segments of x rather than to individual element x_i. Semi-CRFs measure properties of segments and thus transitions within a segment need not be markovian. Segmental CRF models,

$$P(y_{1:J}, E|x_{1:T}) = \frac{1}{Z(x_{1:T})} \prod_j \exp(w^T \phi(y_j, e_j, x_{1:T}))$$

where $e_j =< s_j, n_j >$ denotes the beginning(s_j) and end time (n_j) of y_j and $E = \{e_{1:L}\}$ is the latent segment label

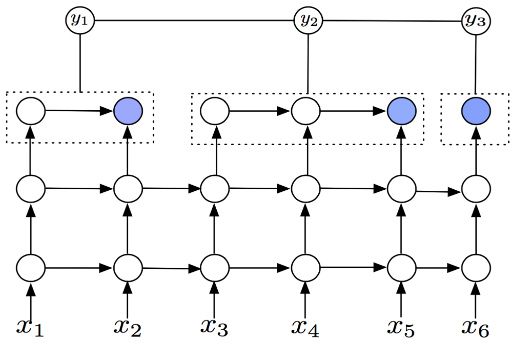


Figure 3.1: *Segmental RNN using CRF (Source: Segmental RNN [9]))*

3.1.1 Segmental CRF and RNN

Segmental RNN model connects the segmental conditional random field with recurrent neural network which is used for features extraction. The model marginalises out all the possible segmentations and features are extracted from RNN trained together with segmental CRF. Thus, it does not depend on external systems to provide segment boundaries and features and hence can be trained end to end [9].
Given a sequence of acoustic frames $X = \{x_1, x_2, ..., x_T\}$ and corresponding sequence of output labels $y = \{y_1, y_2, ...y_J\}$ where $T \geq J$ segmental CRF as above defines seqeunce level conditional probability with auxiliary segment labels,

$$P(y_{1:J}, E|x_{1:T}) = \frac{1}{Z(x_{1:T})} \prod_j \exp(w^T \phi(y_j, e_j, x_{1:T}))$$

where $\phi(.)$ denotes the feature function and others are same as defined above. There this ϕ is defined using neural network, which otherwise used to be heuristically handcrafted.
Feature Representation
Neural Network defines the feature function ϕ which maps acoustic segments to its label into a joint feature space.
Given the segment label e_j, RNN maps the acoustic segment to a fixed dimensional vector,

$$h_1^j = r(h_0, x_{s_j})$$

$$h_2^j = r(h_1^j, x_{s_{j+1}})$$

$$...$$
$$h^j_{d_j} = r(h^j_{d_j-1}, x_{s_{n_j}})$$

where $d_j = n_j - s_j$ duration of the segment and h_0 denotes the initial hidden state and r(.) is a non linear function. $\phi(.)$ is,

$$\phi(y_j, r_j, X) = g(h^j_{d_j}, Mv_j)$$

where M is a linear embedding matrix and v_j is one hot representation of y_j and g(.) can correspond to linear or non linear transformation.
Conditional Maximum Likelihood Training
The segmentation labels are unknown and thus, maximising conditional probability won't help. The loss function is defined as the negative marginal log-likelihood,

$$\begin{aligned}\mathcal{L}(\theta) &= -\log P(y|X) \\ &= -\log \sum_E P(y, E|X)\end{aligned}$$

where θ denotes model parameters. As the number of segmentations are exponential, the problem is solved using dynamic programming.
Decoding
Decoding problem is to search the target label seqeunce y that gives the highest posterior probability given X by marginalising all possible segmentations

$$y^* = \underset{y}{\operatorname{argmax}} \log \sum_E P(y, E|X)$$

This is solved using modified viterbi algorithm.
The computational cost is further reduced by using hierarchical subsampling RNN to shorten the input sequence. This can be done with three variants i) *concatenate*: the hidden states in the subsampling window are concatenated together before they are passed to higher layers. ii) *add*: the hidden states are added into one vector. iii) *skip*: only the last hidden state in a subsample window is kept.

3.2 Connectionist Temporal Classification

The idea of CTC as mentioned above is to interpret the network outputs as probability distribution over all possible labelling given the input sequence. CTC achieves this by allowing network to predict the label at any point of time in input sequence so long as the overall labelling is correct. [6]

3.2.1 Labellings from outputs

Given the input sequence $\{x_1, x_2, ...x_T\}$ and the sequence of labels drawn from the alphabet of size $|L| + 1$, where $|L|$ is the size of alphabet and 1 is added to include the extra blank label.

Let, y_k^t be the probability of observing the label k, where $k \in L'$ and $L' = L \cup \{blank\}$ at time t and π be the T length sequence of these labels such that $\pi \in L'^T$, then assuming that the probability of the labels at each time step are conditional independent given x the probability of observing π is given by,

$$p(\pi|x) = \prod_{k=1}^{T} y_{\pi_k}^k$$

Now, consider a many-to-many mapping $B : L'^T \rightarrow L^{\leq T}$ which maps set of paths onto possible mappings of length $\leq T$. This is done by first removing the repeated labelling and then removing blanks from the paths. Intuitively, this corresponds to outputting a label from no label or outputting a different label from the previous one. Probability of some labelling l is given by,

$$p(l|x) = \sum_{\pi \in B^{-1}(l)} p(\pi|x)$$

This in theory makes CTC unsuitable for tasks where location of labels must be determined, as it collapses different paths onto same labelling. Inclusion of blank labels is important because without blank labels, same labels can not appear consecutively.

3.2.2 Forward Backward Algorithm

To calculate conditional probability $P(l|x)$ directly using the above equation is exponential and this problem is solved using dynamic programming algorithm which is similar to the forward-backward algorithm of HMMs.
Now consider a modified label sequence l' with blanks added between every pair of consecutive labels and in the end and start. Thus length of l' is $2|l| + 1$. For a labelling l, the forward variable $\alpha_t(s)$ is defined as sum of all paths whose length t prefixes are mapped by B on length $s/2$ prefixes of l, i.e.

$$\alpha_t(s) = P(\pi_{1:t} : B(\pi_{1:t}) = l_{1:s/2}, \pi_t = l'_s|x) = \sum_{\pi:B(\pi)=l_{1:s/2}} \prod_{k=1}^{t} y_{\pi_k}^k$$

Using this formulation we can define probability of l given x as sum of forward variables with and without the final blank symbol, i.e.

$$P(l|x) = \alpha_T(|l'|) + \alpha_T(|l'| - 1)$$

Initialization:

$$\begin{aligned} \alpha_1(b) &= y_b^1 \\ \alpha_1(s) &= y_{nb}^1 \quad \text{s.t nb} = l_1 \text{ and s=(b,nb)} \\ \alpha_1(s) &= 0 \qquad\qquad\qquad \forall\ |s| \geq 2 \end{aligned} \tag{3.1}$$

Recursion:

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

The backward variables $\beta_t(s)$ which is defined as sum of all paths whose length t suffixes are mapped by B on length $s/2$ suffixes of l are defined in similar way.

3.2.3 Decoding

After training, we want the most probable labelling l^* given some unknown input x, i.e.

$$l^* = \underset{l}{\operatorname{argmax}}\, p(l|x)$$

Best Path Decoding
It is based on the assumption that most probable path also corresponds to most probable label sequence i.e.

$$l^* \approx B(\pi^*)$$

$$\text{where } \pi^* = \underset{\pi}{\operatorname{argmax}}\, P(\pi|x)$$

Best path decoding is very trivial to compute, as it is just concatenation of the most probable outputs at every time step. However, it is very clear that this can lead to errors as many paths yielding same label sequence can contribute together to give more probability label sequence, which can be better than this most probable path sequence labelling.

Prefix Search Decoding
Prefix search is best first search through the tree of labellings, where the children of a given labelling are those that share it as a prefix. At each step the search extends the labelling whose children have maximum cumulative probability which is the greedy approach.
Let $\gamma_t(p_n)$ and $\gamma_t(p_b)$ be the probability of the network outputting prefix p by time t such that the last letter is non blank and blank resp.

$$\gamma_t(p_n) = P(\pi_{1:t} : B(\pi_{1:t}) = p, \pi_t = p_{|p|}|x)$$

$$\gamma_t(p_b) = P(\pi_{1:t} : B(\pi_{1:t}) = p, \pi_t = blank|x)$$

The pseudo code for the prefix search algorithm is given in 3.2. The algorithm relies on the fact that by modifying forward backward algorithm state above, probabilities of successive extensions of prefix labellings can be calculated.

Constraint Decoding
Usually the output labellings are constraint based on some predefined grammar. These constraints can be expressed by altering the label sequence probability as,

$$l^* = \underset{l}{\operatorname{argmax}}\, p(l|x, G)$$

$$p(l|x, G) = \frac{p(l|x)p(l|G)p(x)}{p(x|G)p(l)}$$

Assuming the fact that the input sequence x is independent of grammar G(though it is not completely correct) and all input sequence are equiprobable,

$$p(l|x,G) = \frac{p(l|x)p(l|G)}{p(l)}$$

$$l^* = \underset{l}{\operatorname{argmax}}\, P(l|x)p(l|G)$$

```
Initialisation:
$1 \le t \le T \begin{cases} \gamma_t(\emptyset_n) = 0 \\ \gamma_t(\emptyset_b) = \prod_{t'=1}^{t} y_b^{t'} \end{cases}$
$p(\emptyset|\mathbf{x}) = \gamma_T(\emptyset_b)$
$p(\emptyset\ldots|\mathbf{x}) = 1 - p(\emptyset|\mathbf{x})$
$\mathbf{l}^* = \mathbf{p}^* = \emptyset$
$P = \{\emptyset\}$

Algorithm:
while $p(\mathbf{p}^*\ldots|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ do
  $probRemaining = p(\mathbf{p}^*\ldots|\mathbf{x})$
  for all labels $k \in L$ do
    $\mathbf{p} = \mathbf{p}^* + k$
    $\gamma_1(\mathbf{p}_n) = \begin{cases} y_k^1 \text{ if } \mathbf{p}^* = \emptyset \\ 0 \text{ otherwise} \end{cases}$
    $\gamma_1(\mathbf{p}_b) = 0$
    $prefixProb = \gamma_1(\mathbf{p}_n)$
    for $t = 2$ to $T$ do
      $newLabelProb = \gamma_{t-1}(\mathbf{p}_b^*) + \begin{cases} 0 \text{ if } \mathbf{p}^* \text{ ends in } k \\ \gamma_{t-1}(\mathbf{p}_n^*) \text{ otherwise} \end{cases}$
      $\gamma_t(\mathbf{p}_n) = y_k^t\,(newLabelProb + \gamma_{t-1}(\mathbf{p}_n))$
      $\gamma_t(\mathbf{p}_b) = y_b^t\,(\gamma_{t-1}(\mathbf{p}_b) + \gamma_{t-1}(\mathbf{p}_n))$
      $prefixProb \mathrel{+}= y_k^t\; newLabelProb$
    $p(\mathbf{p}|\mathbf{x}) = \gamma_T(\mathbf{p}_n) + \gamma_T(\mathbf{p}_b)$
    $p(\mathbf{p}\ldots|\mathbf{x}) = prefixProb - p(\mathbf{p}|\mathbf{x})$
    $probRemaining \mathrel{-}= p(\mathbf{p}\ldots|\mathbf{x})$
    if $p(\mathbf{p}|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ then
      $\mathbf{l}^* = \mathbf{p}$
    if $p(\mathbf{p}\ldots|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ then
      add $\mathbf{p}$ to $P$
    if $probRemaining \le p(\mathbf{l}^*|\mathbf{x})$ then
      break
  remove $\mathbf{p}^*$ from $P$
  set $\mathbf{p}^* = \mathbf{p} \in P$ that maximises $p(\mathbf{p}\ldots|\mathbf{x})$

Termination:
output $\mathbf{l}^*$
```

Figure 3.2: *Prefix Search Decoding Algorithm (Source: CTC [4]))*

Chapter 4

End to End Speech Recognition System

In this chapter, we will present a speech recognition system that directly transcribes audio data with text without the requirement of an intermediate phonetic representation [7]. Objective functions used to train networks which are not end to end learning are a lot different from true performance measure. Here the target is phonetic and output is lexical and the objective function used allows direct optimization of the word error rate. For training, in general, we need aligned data. But aligning is difficult and we can easily get unaligned transcription. We can use approaches based on segmental RNN as described in section 3.1 or use CTC based approach as described in section 3.2. We will discuss a system based on the combination of deep bidirectional LSTM RNN architecture and connectionist temporal classification.

4.1 Network Architecture

Given an input sequence $x = (x_1, x_2, ...x_T)$, recurrent neural network computes hidden vector sequence $h = (h_1, h_2, ...h_T)$ and output vector sequence $y = (y_1, y_2, ..., y_T)$, using

$$h_t = H(W_{ih}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{ho}h_t + b_o$$

where H is the hidden layer activation function(usually sigmoid function), W is weight matrix and b is the bias. For LSTM based network, H is implemented by composite functions as,

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

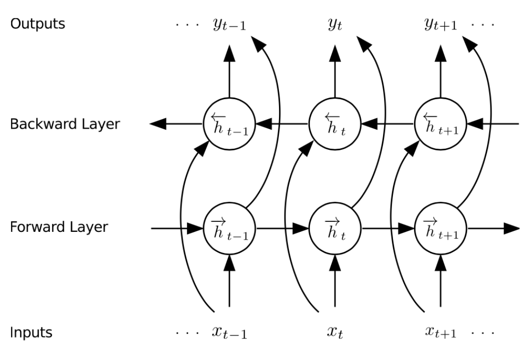


Figure 4.1: *Bidirectional Recurrent Neural Network (Source: [7])*

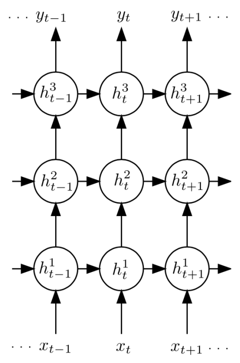


Figure 4.2: *Deep Recurrent Neural Network (Source: [7])*

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$
$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$
$$h_t = o_t tanh(c_t)$$

where σ is the logistic sigmoid function, and i,f,o and c are input gate, forget gate, output gate and control gate activation vectors. Here, they are only able to use the previous context, but bidirectional RNNs can be used so as to exploit the future context as well. BRNNs uses two separate hidden layers to process data in both directions. It computes the output vector sequence as,

$$\overleftarrow{h}_t = H(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t-1} + b_{\overleftarrow{h}})$$
$$\overrightarrow{h}_t = H(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}})$$
$$y_t = W_{\overleftarrow{h}y}\overleftarrow{h}_t + W_{\overrightarrow{h}y}\overrightarrow{h}_t + b_o$$

Here combining the BRNNs with LSTM by using the composite function for H as described above, we get a network which can access long range context and in both input directions. Using deep architectures on top of this helps to build up deeper lever representations of acoustic data. Deep RNNs are created by stacking multiple RNN layers on top of each other where the output of one layer becomes input of the next layer. Assuming the same hidden layer is used for all N layers,

$$h^n_t = H(W_{h^{n-1}h^n}h^{n-1}_t + W_{h^n h^n}h^n_{t-1} + b^n_h)$$
$$y_t = W_{h^N y}h^N_t + b_o$$

where h^0 is the input x. Using all of these together gives Deep Bidirectional Long short term memory based recurrent neural network.

4.2 Expected Transcription Loss

Given a target transcription y^*, the network can be trained to minimise CTC objective function,

$$CTC(x) = -\log P(y^*|x)$$

But, here relative probabilities of incorrect transcription are ignored as it only tries to maximise the probability of correct transcription. Also, in speech recognition the standard measure is Word Error Rate (WER), defined as the edit distance between the correct transcription and the predicted transcription. Given input sequence x and the distribution $P(y|x)$ as defined by CTC, the expected transcription loss is defined as,

$$\mathcal{L}(x) = \sum_{y} P(y|x)\mathcal{L}(x, y)$$

where $\mathcal{L}$ is a real values transcription loss function. Monte-Carlo sampling is used to approximate $\mathcal{L}$ because, in general, calculating this expectation is computationally expensive. In randomly initialised network, vast majority of alignments gives wrong transcription and for the sampling procedure to be effective, expected loss transcription is used on the network already trained with CTC.

4.3 Decoding with a language model

To decode with the help of a character language model, beam search can be modified as in [10]. The pseudo code is shown in 4.3. The idea is, given the likelihoods from the DBRNN and the character language model, for each step t and string s in the previous hypothesis set Z_{t-1}, extend s with a new character. The decoding procedure without a language model is significantly difficult, and computationally costly than the decoding with a language model because without a lexicon they have to obey various constraints during character level decoding procedure.

```
**Inputs** CTC likelihoods $p_{\text{ctc}}(c|x_t)$, character language model $p_{\text{clm}}(c|s)$
**Parameters** language model weight $\alpha$, insertion bonus $\beta$, beam width $k$
**Initialize** $Z_0 \leftarrow \{\varnothing\}$, $p_{\text{b}}(\varnothing|x_{1:0}) \leftarrow 1$, $p_{\text{nb}}(\varnothing|x_{1:0}) \leftarrow 0$
**for** $t = 1, \ldots, T$ **do**
    $Z_t \leftarrow \{\}$
    **for** $s$ **in** $Z_{t-1}$ **do**
        $p_{\text{b}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(\_|x_t)p_{\text{tot}}(s|x_{1:t-1})$            ▷ 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** $\zeta'$ **do**
            $s^+ \leftarrow s + c$
            **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**
                $p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{b}}(c|x_{1:t-1})$    ▷ Repeat characters have "_" between
            **end if**
            Add $s^+$ to $Z_t$
        **end for**
    **end for**
    $Z_t \leftarrow k$ most probable $s$ by $p_{\text{tot}}(s|x_{1:t})|s|^\beta$ in $Z_t$            ▷ Apply beam
**end for**
**Return** $\arg\max_{s \in Z_t} p_{\text{tot}}(s|x_{1:T})|s|^\beta$
```

Figure 4.3: *Beam Search Decoding using a language Model (Source: [10])).*
Blanks and repeat characters with no separating blank are handled separately. For all other character extensions, character language model is applied when computing the probability of s. Z_0 is initialized with the empty string ϕ.
Notation: $p_{ctc}(a|x)$ is the probability of character "a" in the language model. p_b and p_{nb} is the probability of seeing a blank and a non blank symbol at the last position.

Chapter 5

Baseline Implementation

We have implemented system as described in [6] as a baseline implementation in tensorflow. We used TIMIT speech corpus for the training purpose. The task is to annotate the utterances in the TIMIT test set with the phoneme sequences and character sequence that gave the lowest possible label error rate. We chose BLSTM because our experiments for the reasons explained in previous chapter.

5.1 Data

TIMIT contain recordings of prompted English speech, accompanied by manually segmented phonetic transcripts. It has a lexicon of 61 distinct phonemes, and comes divided into training and test sets containing 4620 and 1680 utterances respectively. We used kaldi to pre-process the audio frames and to extract Mel-Frequency Cepstrum Coefficients (MFCCs). The input to the BLSTM is a seqeunce of feature vectors each sized 23.

5.2 Experiment setup

The CTC network used an extended BLSTM architecture with peepholes and forget gates, 2 blocks in each of the forward and backward hidden layers, hyperbolic tangent for the input and output cell activation functions and a logistic sigmoid in the range [0, 1] for the gates.

The hidden layers were fully connected to themselves and the output layer, and fully connected from the input layer. The input layer was size 23, the softmax output layer size 63 (1 extra to include blank symbol emitted by CTC).

Training was carried out with back propagation through time and online gradient descent (weight updates after every training example), using a learning rate of 104 and a momentum of 0.9. We used Dropouts with the probability 0.6 for each of the input and output layer to avoid over-fitting. Dropout is a technique for addressing over fitting problem [14]. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces over-fitting and gives major improvements over other regularization methods. We used glorot initialization to improve initialization of the BLSTM.

5.3 Experimental Results

We run the implementation on GPUs to train the network, but the phonemes error rate and word error rate were not too low possibly because of lack training data and number of hidden layers. In 5.1 and 5.2 we show the results of phoneme error rate and word error rate on TIMIT dataset. The

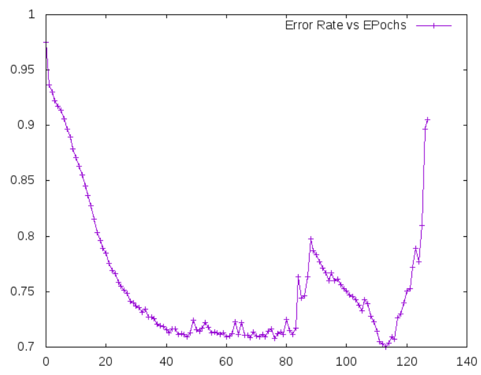


Figure 5.1: *Graph of phoneme error rate vs epochs*

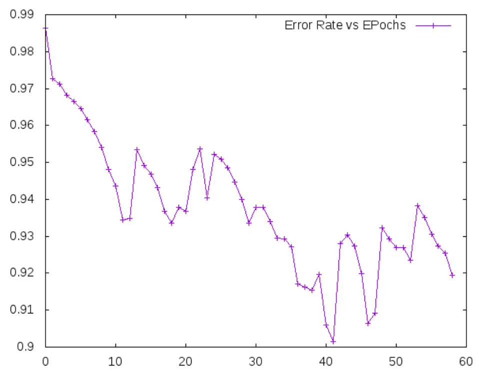


Figure 5.2: *Graph of word error rate vs epochs*

graph is not smooth because, in each iteration(epoch) it tries to decrease the loss but, error rate fluctuates.

5.4 Future Work

Currently, we are trying to incorporate a character language model as described in section 4.3, which will surely improve on the phoneme and word error rate. Also, we would like to see the error rate when trained on large data set.

Chapter 6

Conclusion and Open Problems

In this report, we discussed the basics behind recurrent and convolutional neural networks. We discussed few models for sequence labelling task, where I explained Hidden Markov Model, its three basic problems and their solutions, followed by various attention and augmented memory models (like Neural Turing machine and Neural Programmer) and segmented recurrent neural network. We introduced connectionist temporal classification which gives a novel way for RNN to train end to end. Lastly, we discussed a end to end system which uses CTC and a character language model for decoding.

But, still there are many problems which are not much explored. Consider the problem of training a end to end system for a bilingual system i.e. the network being trained on the input utterances in one language and the transcription in some other language. How it will perform on high or low mutually intelligible languages? Also, consider the problem where the utterances are not solely from one language (the problem of code switching) i.e. they come from two languages. Again, how it will perform on high and low mutually intelligible languages? We would also, like to know how the end to end network learns the language model and language constraints. And given a input utterance, can it confidently predict whether the utterance is from the same language on which it was trained. Also, attention and augmented memory based models are not explored for speech recognition task. Consider the problem of getting semantics from the input utterance, can we get the mood of the speaker?

Bibliography

[1] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks 5.2 (1994)*, 1994.

[2] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2016.

[3] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in Neural Information Processing Systems*, 2015.

[4] Alex Graves. Supervised sequence labelling with recurrent neural networks. *Springer Berlin Heidelberg*, 2012.

[5] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv*, 2016.

[6] Alex Graves, Santiago FernÌĄand, and Faustino Gomez JuÌĹrgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *Proceedings of the 23rd international conference on Machine learning*, 2006.

[7] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. *Proceedings of the 31rd international conference on Machine learning*, 2014.

[8] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv*, 2014.

[9] Kong, Lingpeng, Chris Dyer, , and Noah A. Smith. Segmental recurrent neural networks. *arXiv preprint arXiv*, 2015.

[10] Andrew L. Maas, Ziang Xie, Dan Jurafsky, and Andrew Y. Ng. Lexicon-free conversational speech recognition with neural networks. *Conference of the North American Chapter of the Association for Computational Linguistics âĂŞ Human Language Technologies*, 2015.

[11] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 2016.

[12] Christopher Olah. Understanding lstm networks. 2015.

[13] L.R. Rabiner. A tutorial on hidden markov model and selected applications in speech recongition. *Proceedings of the IEEE*, 77, 1989.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning (ICML)*, 2015.