# End-to-end models for Automatic Speech Recognition

CTC, DeepSpeech and beyond

Shreekantha Nadig

February 15, 2019

International Institute of Information Technology - Bangalore

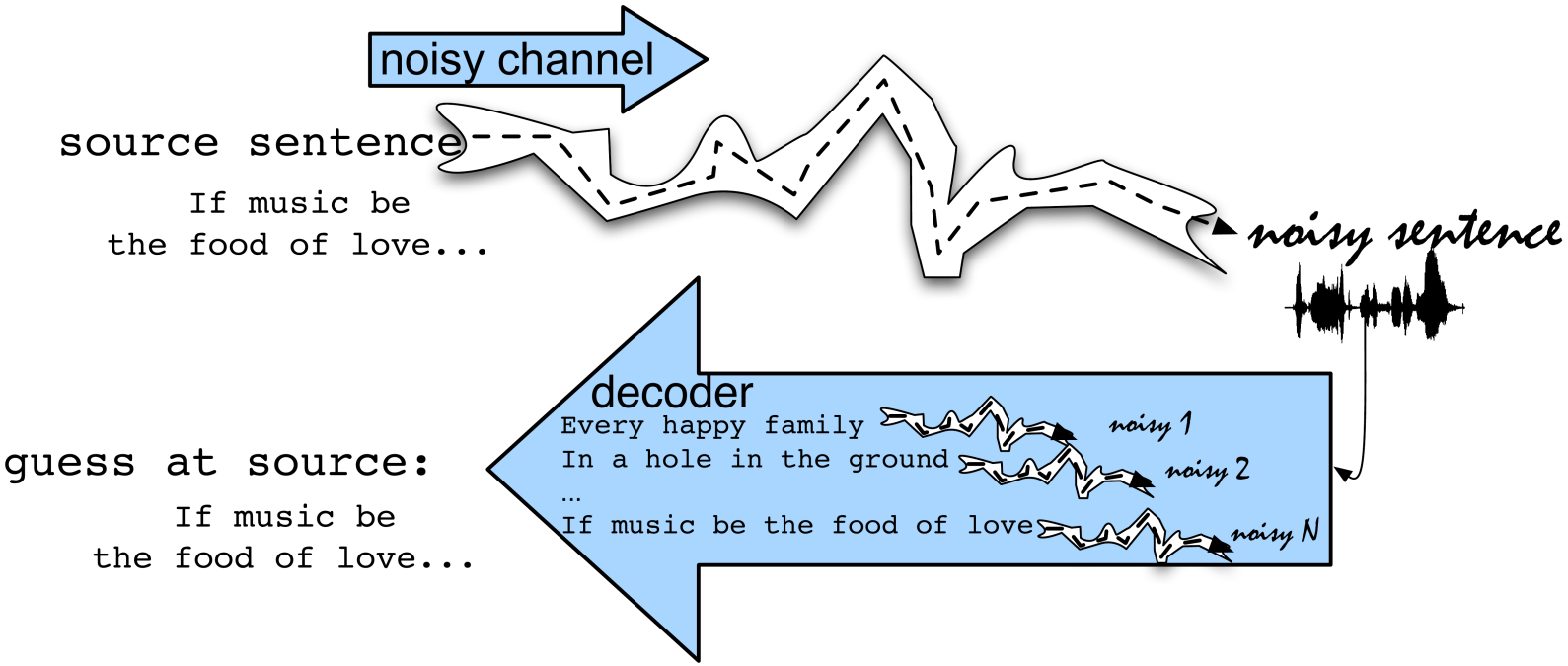
# Table of contents

1. Introduction
2. ASR architecture
3. WER
4. Decoding
5. RNN Acoustic Model
6. Motivation for CTC
7. CTC - Connectionist Temporal Classification
8. Deepspeech architecture
9. Beyond DeepSpeech
10. Summary

# Introduction

# The Noisy Channel Model

- Search through space of all possible sentences.
- Pick the one that is most probable given the waveform.



## The Noisy Channel Model (II)

- What is the most likely sentence out of all sentences in the language L given some acoustic input O?
- Treat acoustic input O as sequence of individual observations
  - $O = o_1, o_2, o_3, \ldots, o_t$
- Define a sentence as a sequence of words:
  - $W = w_1, w_2, w_3, \ldots, w_n$

## Noisy Channel Model (III)

- Probabilistic implication: Pick the highest prob S:

$$\hat{W} = \underset{W \in L}{\arg\max}\, P(W \mid O)$$

- We can use Bayes rule to rewrite this:

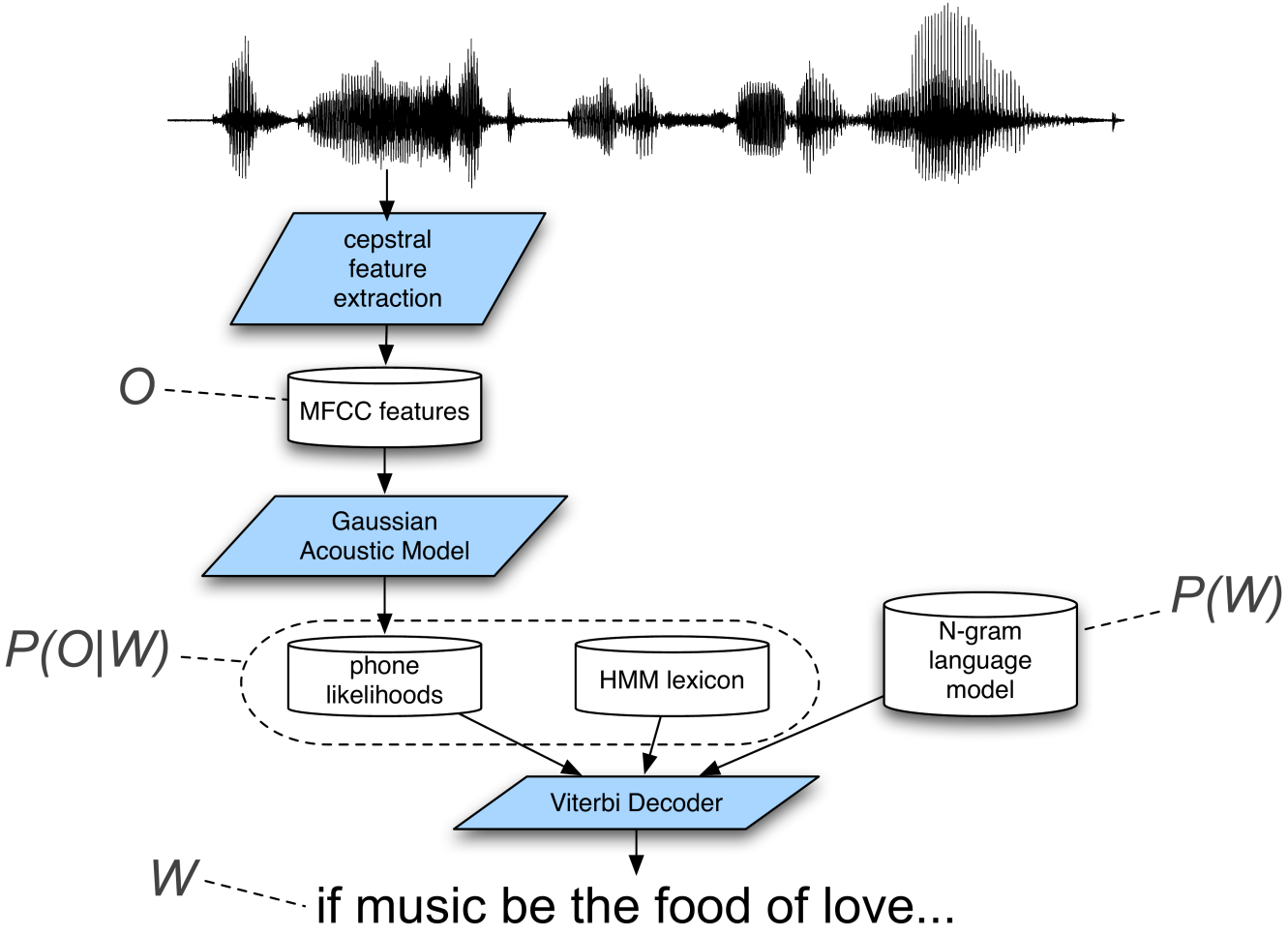
$$\hat{W} = \underset{W \in L}{\arg\max} \frac{P(O \mid W)P(W)}{P(O)}$$

- Since denominator is the same for each candidate sentence W, we can ignore it for the argmax:

$$\hat{W} = \underset{W \in L}{\arg\max}\, P(O \mid W)P(W)$$

# ASR architecture

# Speech Recognition Architecture



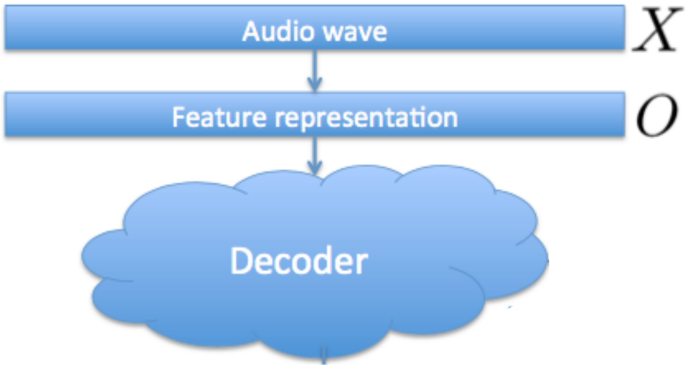
## Noisy channel model

likelihood prior

$$\hat{W} = \arg\max_{W \in L} P(O \mid W) P(W)$$

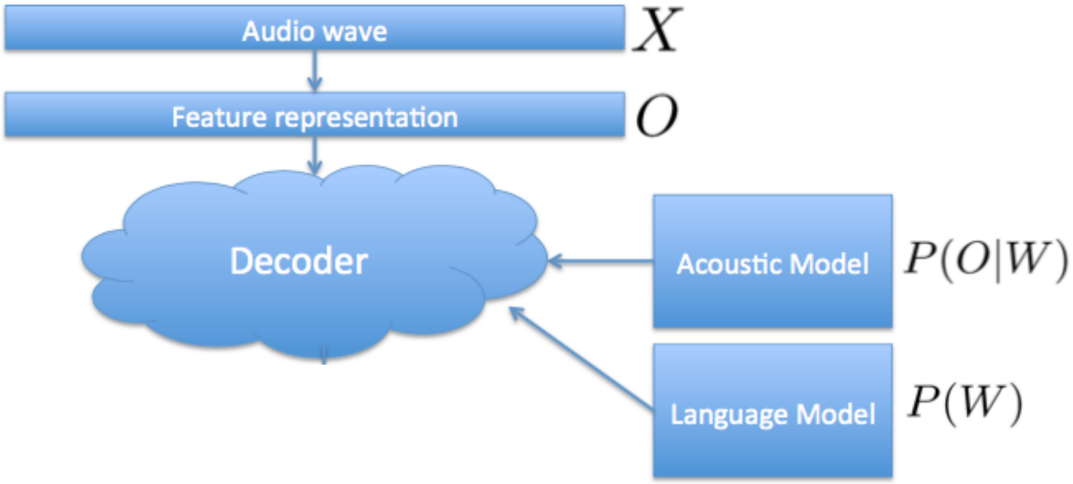
# ASR Workchain

- Split up the task into different components:



# ASR Workchain

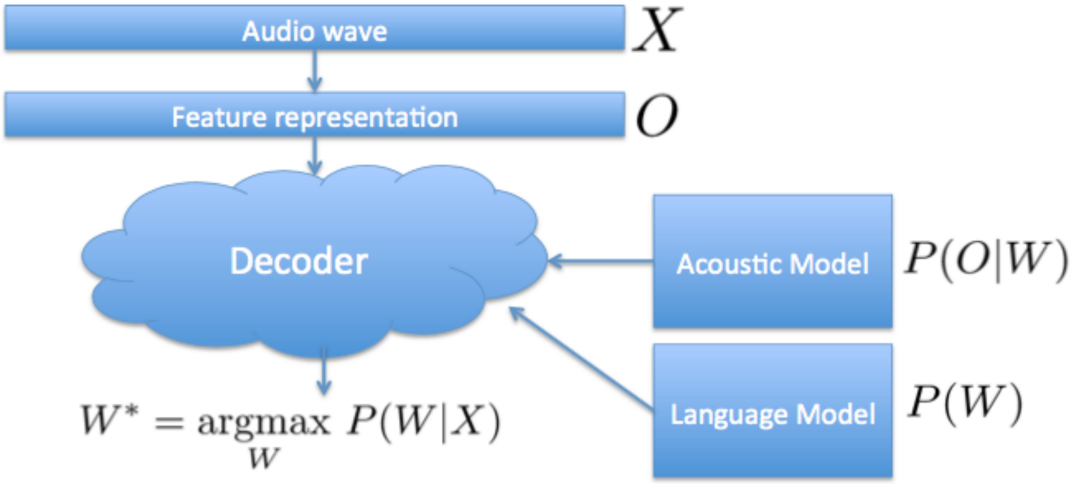
- Split up the task into different components:




Gales & Young, 2008
Jurafsky & Martin, 2000

# ASR Workchain

- Split up the task into different components:

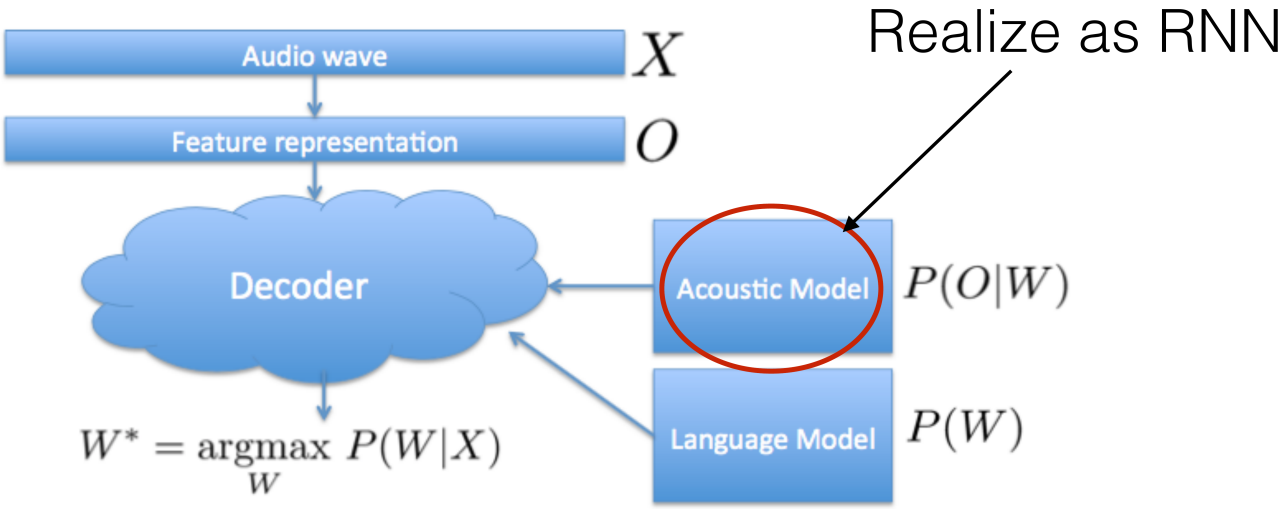



$$W^* = \underset{W}{\operatorname{argmax}}\ P(W|X)$$
$$= \underset{W}{\operatorname{argmax}}\ P(O|W)P(W)$$

Gales & Young, 2008
Jurafsky & Martin, 2000

# ASR Workchain

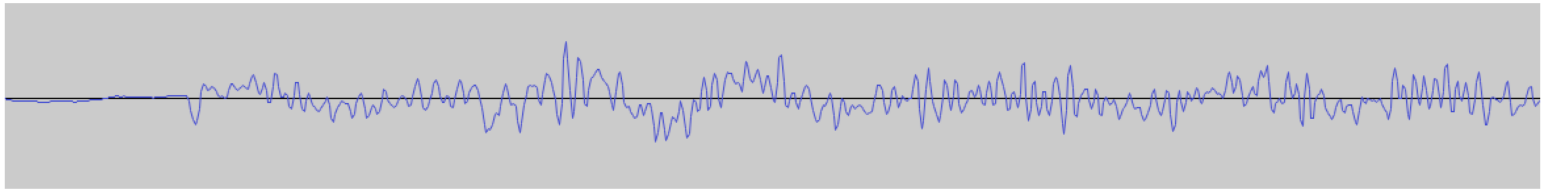
- Split up the task into different components:




Gales & Young, 2008
Jurafsky & Martin, 2000

# Raw audio

- Simple 1D signal:



Typical sample rates for speech: 8KHz, 16KHz.
Each sample typically 8-bit or 16-bit.

- 1D vector: $X = [x_1 x_2 \ldots]$

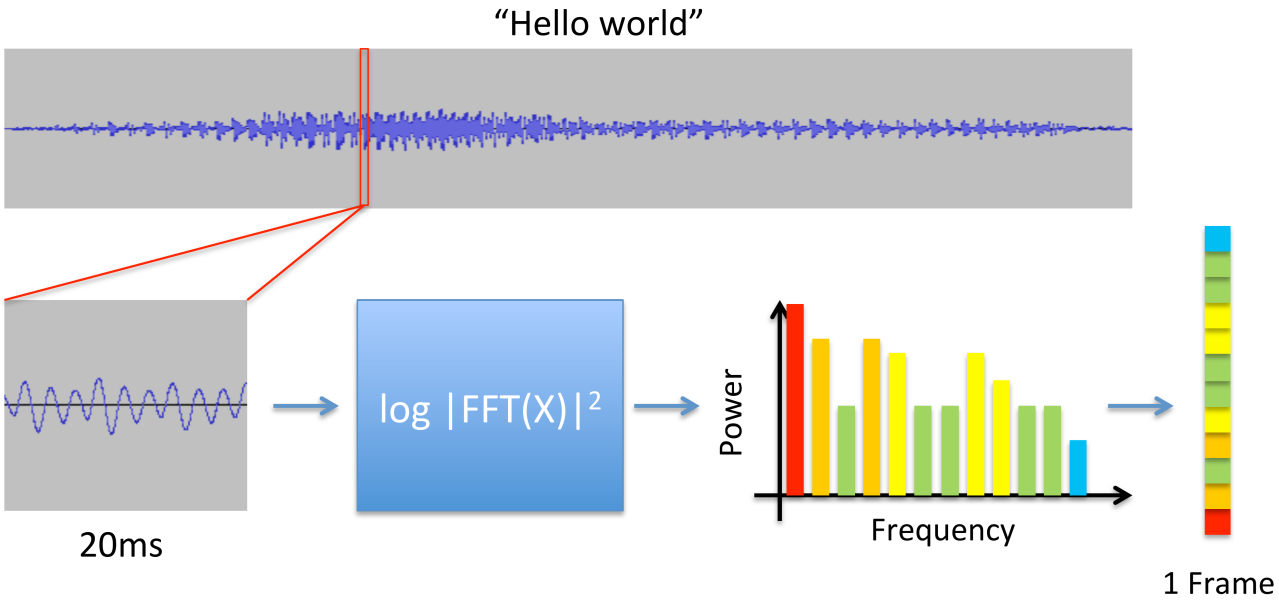
# Pre-processing

- Two ways to start:
  - Minimally pre-process (e.g., simple spectrogram).
    - We'll use this.
  - Train model from raw audio wave.
    - It works!

    See, e.g., Sainath et al., Interspeech 2015

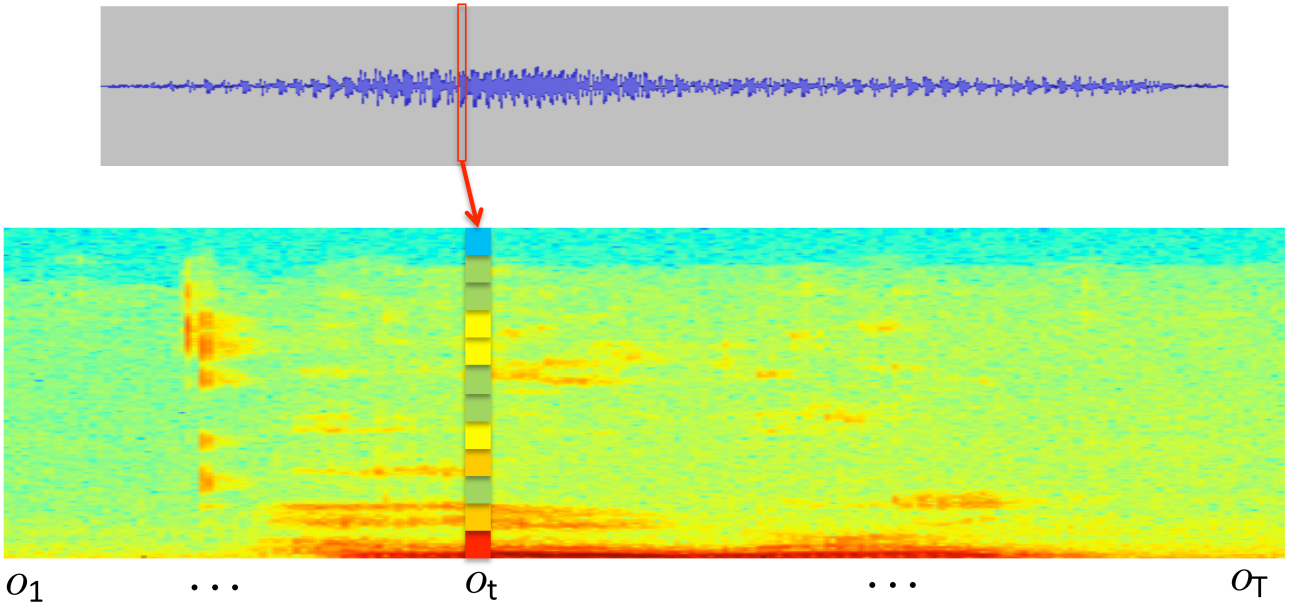
# Spectrogram

- Take a small window (e.g., 20ms) of waveform.
  - Compute FFT and take magnitude. (i.e., power)
  - Describes frequency content in local window.



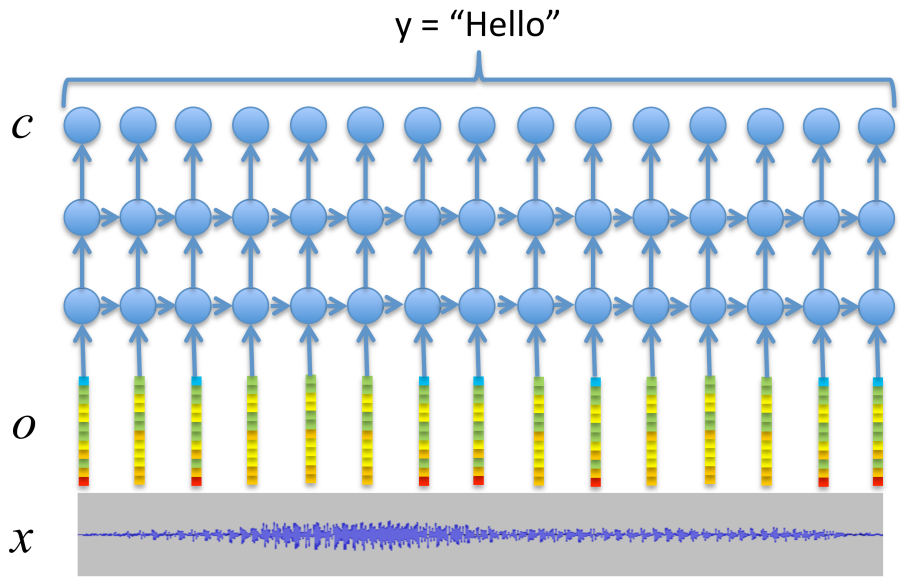
# Spectrogram

- Concatenate frames from adjacent windows to form “spectrogram”.



# Acoustic Model

- Goal: create a neural network (DNN/RNN) from which we can extract transcription, $y$.
  - Train from labeled pairs ($x$,$y$*)



# Acoustic model

- Main issue: length(x) != length(y)
  - Don't know how symbols in y map to frames of audio.
  - Traditionally, try to bootstrap alignment – painful!

- Multiple ways to resolve:
  - Use attention, sequence to sequence models, etc. [Chan et al., 2015; Bahdanau et al., 2015]

  - Connectionist Temporal Classification [Graves et al., 2006]

# WER

## Word Error Rate

- Word Error Rate =

$$\frac{100\ (\text{Insertions+Substitutions + Deletions})}{\text{Total Word in Correct Transcript}}$$

- Aligment example:

```
REF:  portable ****  PHONE UPSTAIRS last night so
HYP:  portable FORM  OF    STORES   last night so
Eval           I     S     S
```

- WER = 100 (1+2+0)/6 = 50%

# Decoding

# What we are searching for

- Given Acoustic Model (AM) and Language Model (LM):

AM (likelihood) LM (prior)

$$(1) \quad \hat{W} = \arg\max_{W \in L} P(O \mid W) P(W)$$

# Combining Acoustic and Language Models

- We don't actually use equation (1)

$$(1)\quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O|W)P(W)$$

  - AM underestimates acoustic probability
    - Why? Bad independence assumptions
    - Intuition: we compute (independent) AM probability estimates; but if we could look at context, we would assign a much higher probability. So we are underestimating
    - We do this every 10 ms, but LM only every word.
    - Besides: AM isn't a true probability
  - AM and LM have vastly different dynamic ranges

# Language Model Scaling Factor

- Solution: add a language model weight (also called language weight LW or language model scaling factor LMSF

$$(2)\quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O|W)P(W)^{LMSF}$$

$$= \text{argmax } \log P(O|W) + LMSF * \log P(W)$$

- Value determined empirically, is positive (why?)
- Often in the range 10 +/- 5.
- Kaldi uses an acoustic model scaling factor instead, but it achieves the same efffect

## Language Model Scaling Factor

- As LMSF is increased:
  - More deletion errors (since increase penalty for transitioning between words)
  - Fewer insertion errors
  - Need wider search beam (since path scores larger)
  - Less influence of acoustic model observation probabilities

Slide from Bryan Pellom

# Word Insertion Penalty

- But LM prob P(W) also functions as penalty for inserting words
  - Intuition: when a uniform language model (every word has an equal probability) is used, LM prob is a 1/V penalty multiplier taken for each word
  - Each sentence of N words has penalty N/V
  - If penalty is large (smaller LM prob), decoder will prefer fewer longer words
  - If penalty is small (larger LM prob), decoder will prefer more shorter words
- When tuning LM for balancing AM, side effect of modifying penalty
- So we add a separate word insertion penalty to offset

$$(3)\quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O|W)P(W)^{LMSF}\, WIP^{N(W)}$$

## Word Insertion Penalty

- Controls trade-off between insertion and deletion errors
  - As penalty becomes larger (more negative)
  - More deletion errors
  - Fewer insertion errors
- Acts as a model of effect of length on probability
  - But probably not a good model (geometric assumption probably bad for short sentences)

## Log domain

- We do everything in log domain
- So final equation:

$$(4)\quad \hat{W} = \underset{W \in L}{\operatorname{argmax}} \log P(O|W) + LMSF \log P(W) + N \log WIP$$

# RNN Acoustic Model

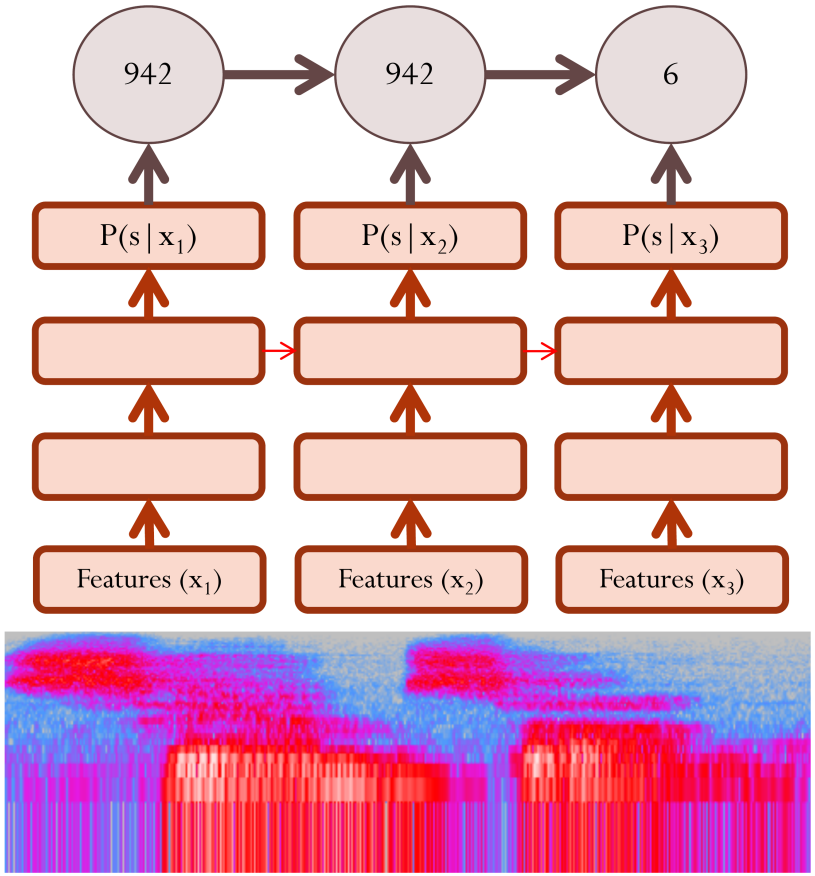
# *Recurrent* DNN Hybrid Acoustic Models

**Transcription:** Samson
**Pronunciation:** S – AE – M – S –AH – N
**Sub-phones :** 942 – 6 – 37 – 8006 – 4422 …

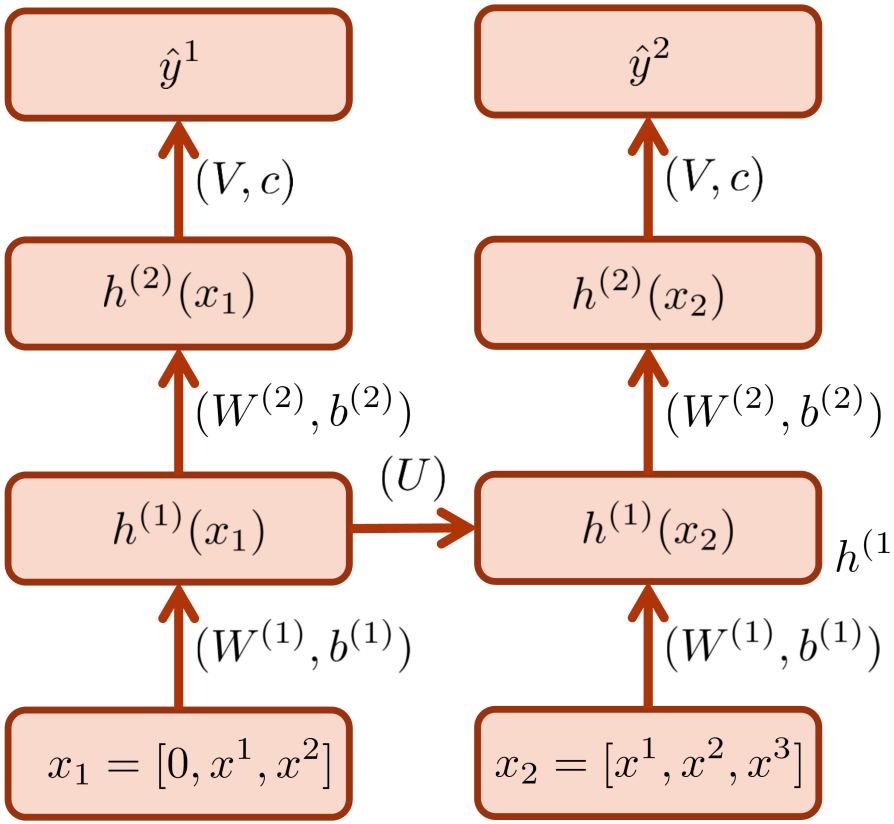
**Hidden Markov Model (HMM):**

**Acoustic Model:**

**Audio Input:**



# *Deep* Recurrent Network




Output Layer

$$\hat{y}^2 = Vh^{(1)}(x_2) + c$$

Hidden Layer

$$h^{(2)}(x_2) = \sigma(W^{(2)}h^{(1)}(x_2) + b^{(2)})$$

Hidden Layer

$$h^{(1)}(x_2) = \sigma(W^{(1)}x_2 + b^{(1)} + Uh^{(1)}(x_1))$$

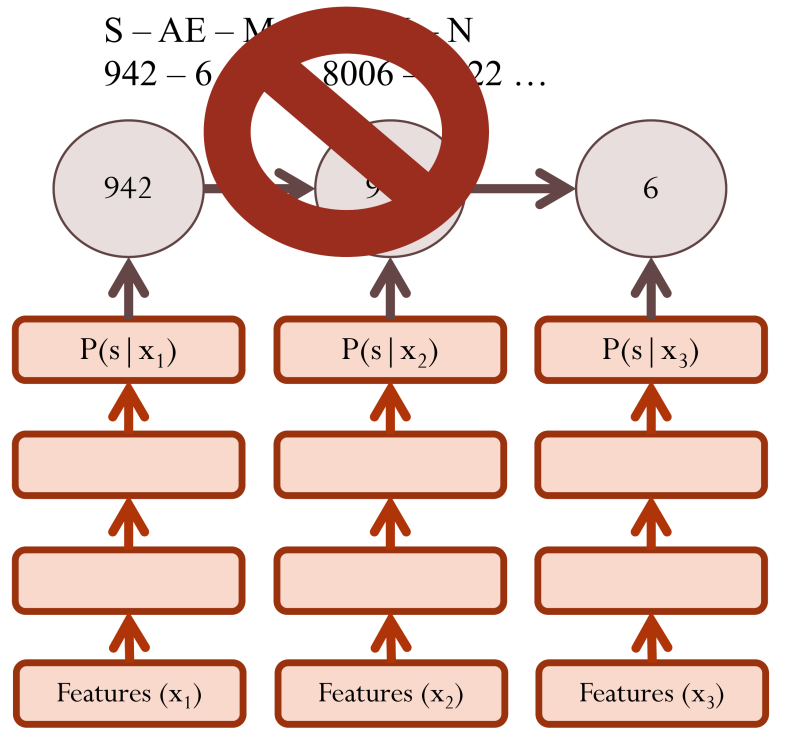
Input

$$\mathbf{X}$$

## HMM-Free Recognition

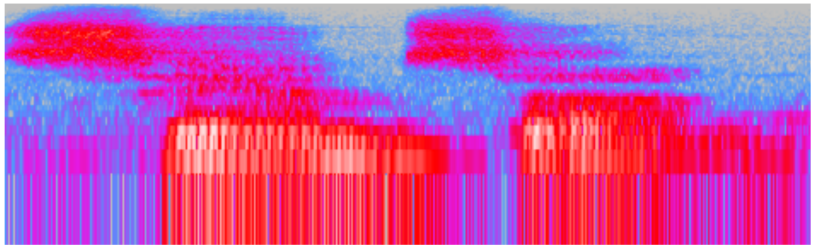
**Transcription:** Samson
**Pronunciation:** S – AE – M[illegible] N
**Sub-phones :** 942 – 6[illegible] 8006 – [illegible]22 …

**Hidden Markov Model (HMM):**




**Acoustic Model:**

**Audio Input:**



(Graves & Jaitly. 2014)

# HMM-Free Recognition

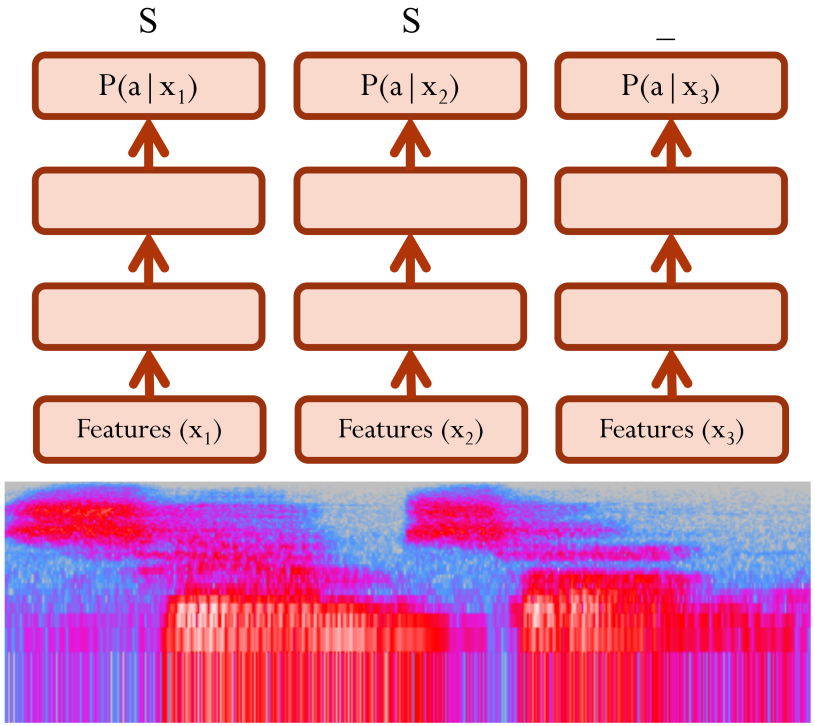
**Transcription:** Samson

**Characters:** SAMSON

**Collapsing function:** SS___AA_M_S___O___NNNN

**Acoustic Model:**

**Audio Input:**




Use a DNN to approximate: P(a|x)

The distribution over *characters*

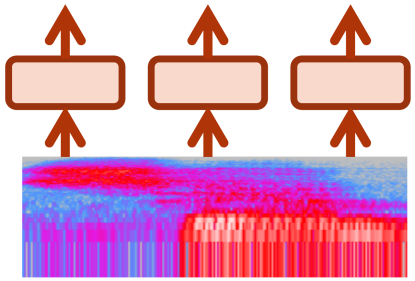
(Graves & Jaitly. 2014)

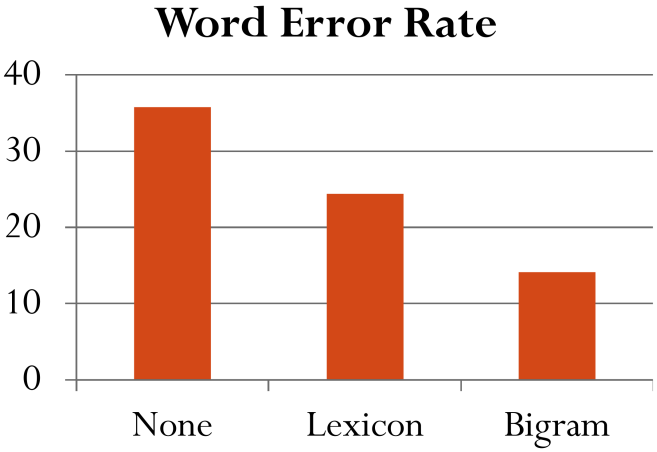
# Decoding with a Language Model

**Lexicon** [a, …, zebra]

**Language Model** p(“yeah” | “oh”)

**Character Probabilities** __oo_h__y_e_aa_h






(Hannun, Maas, Jurafsky, & Ng. 2014)

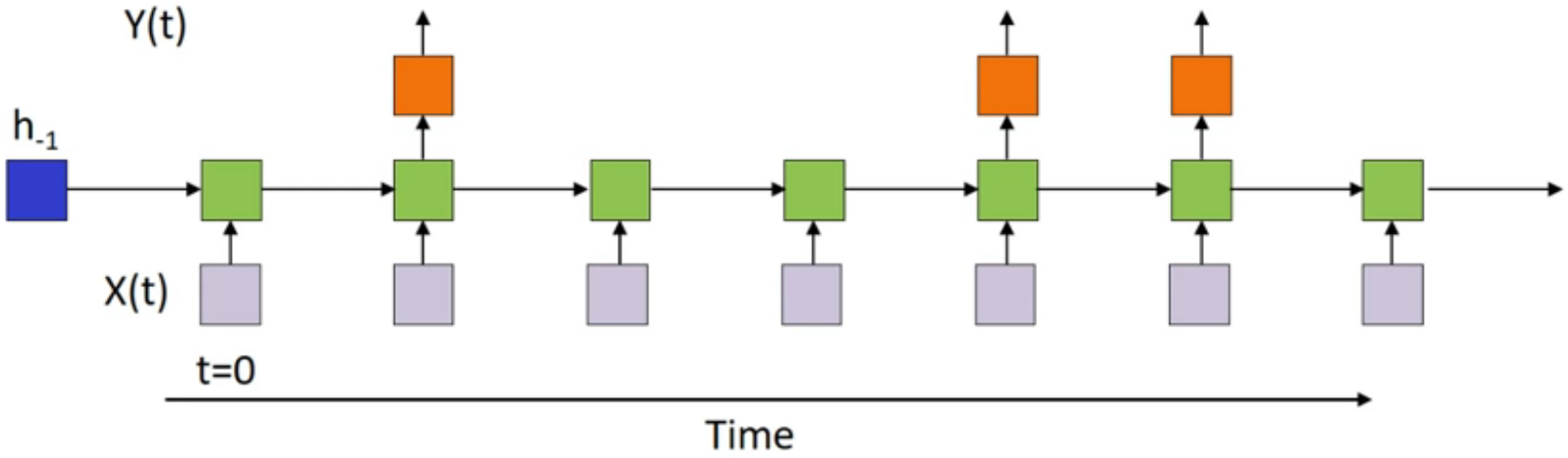
# Motivation for CTC

# Motivation

- Popular loss functions (MSE, XE) assume a one-to-one correspondence between the network's output and target labels
- But what if there is no one-to-one correspondence? (Speech Recognition, Machine Translation, handwriting recognition)
- Need a way to find alignment between network outputs and target labels.

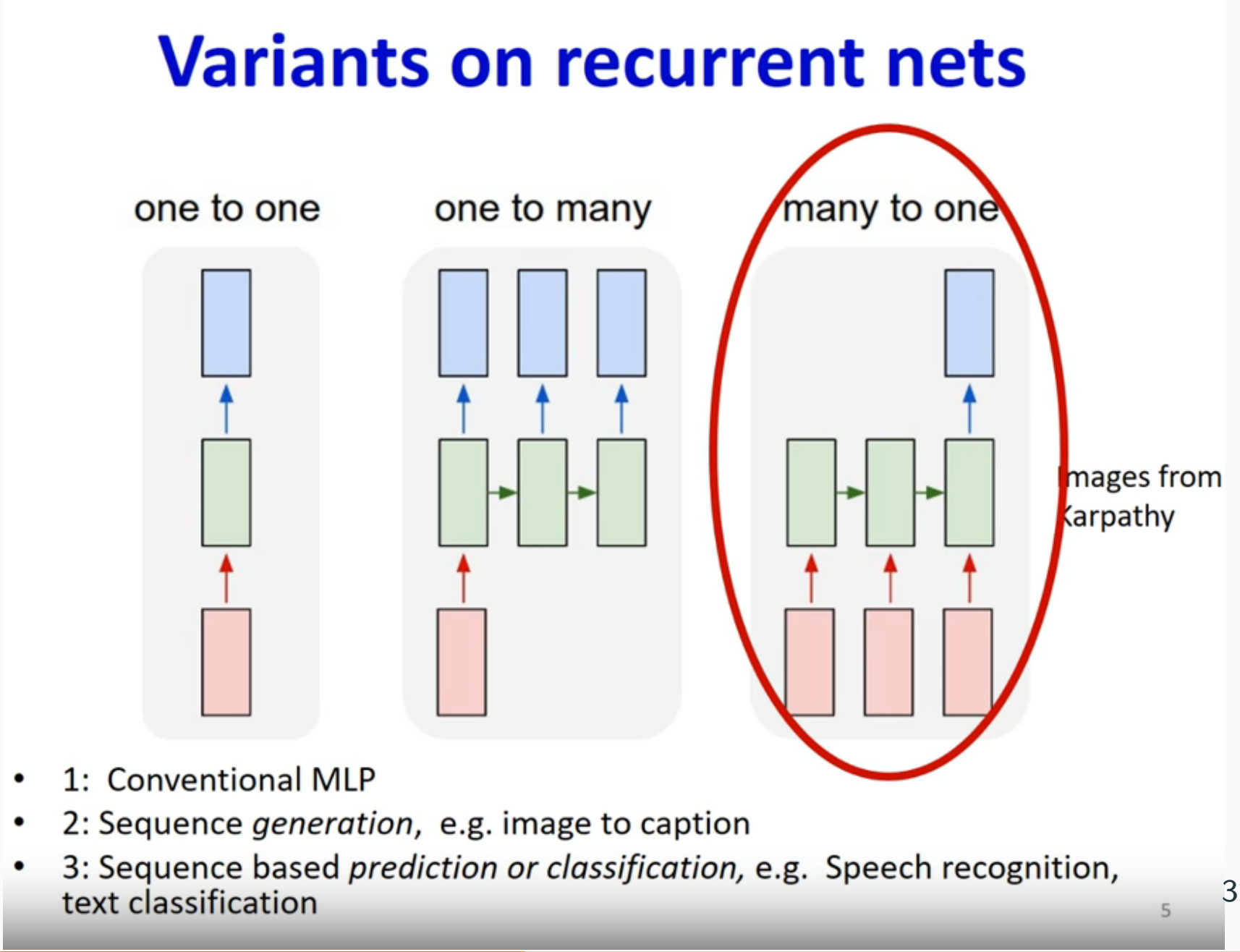
CTC - Connectionist Temporal Classification

# Case 1: With alignment

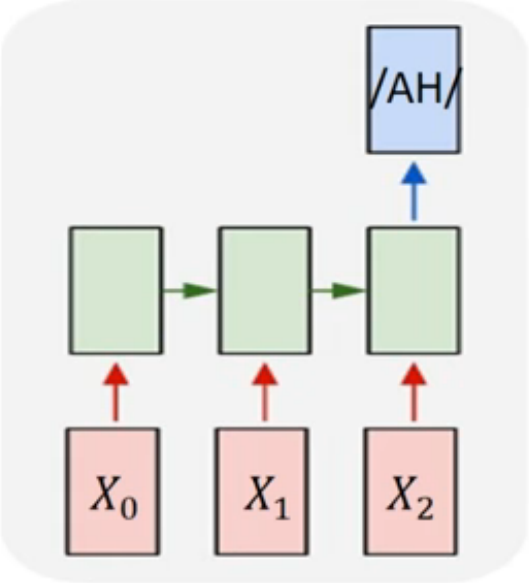



- The input and output sequences happen in the same order
  - Although they may be asynchronous
  - E.g. Speech recognition
    - The input speech corresponds to the phoneme sequence output

# Motivation for CTC

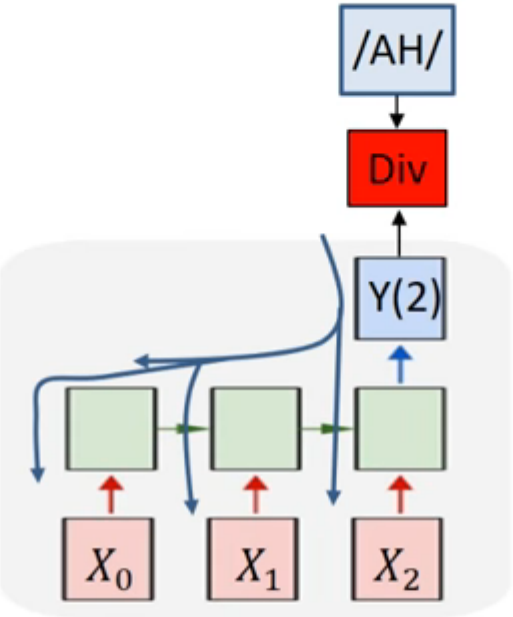


# Basic model




- Sequence of inputs produces a single output

# Training

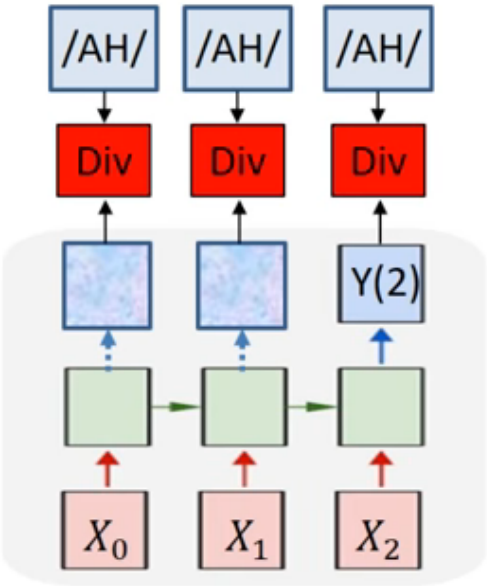



- The Divergence is only defined at the final input
  - $DIV(Y_{target}, Y) = Xent(Y(T), Phoneme)$
- This divergence must propagate through the net to update all parameters
- Ignores outputs at intermediate steps

# Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme




- Exploiting the untagged inputs: assume the same output for the entire input
- Define the divergence everywhere

$$DIV(Y_{target}, Y) = \sum_t w_t Xent(Y(t), Phoneme)$$

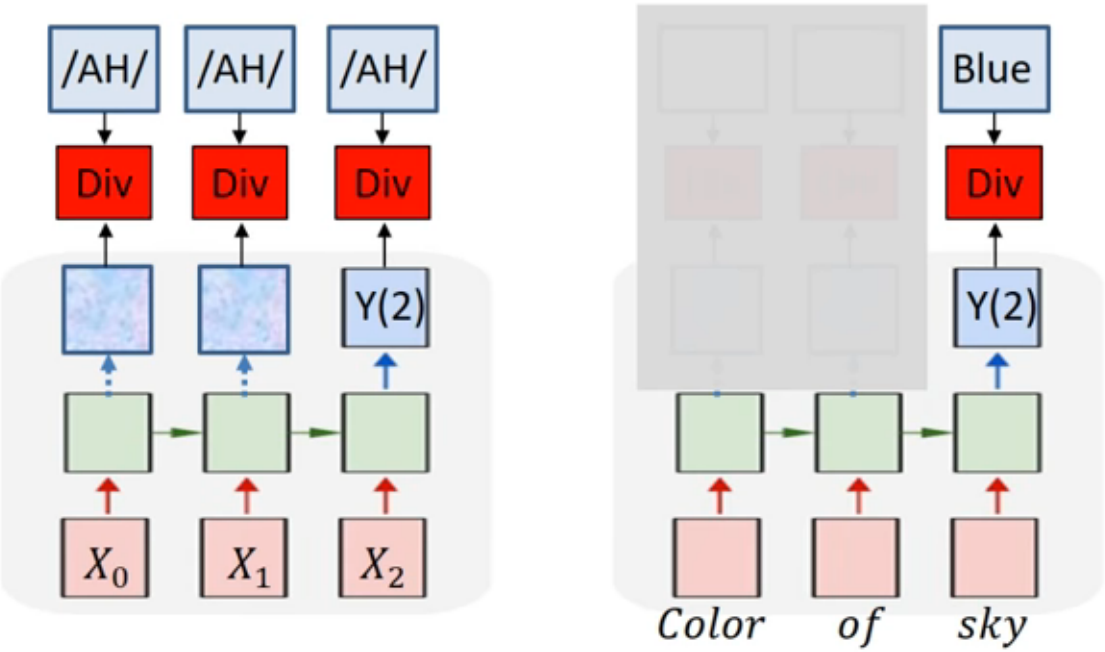
8

# Motivation for CTC

## Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



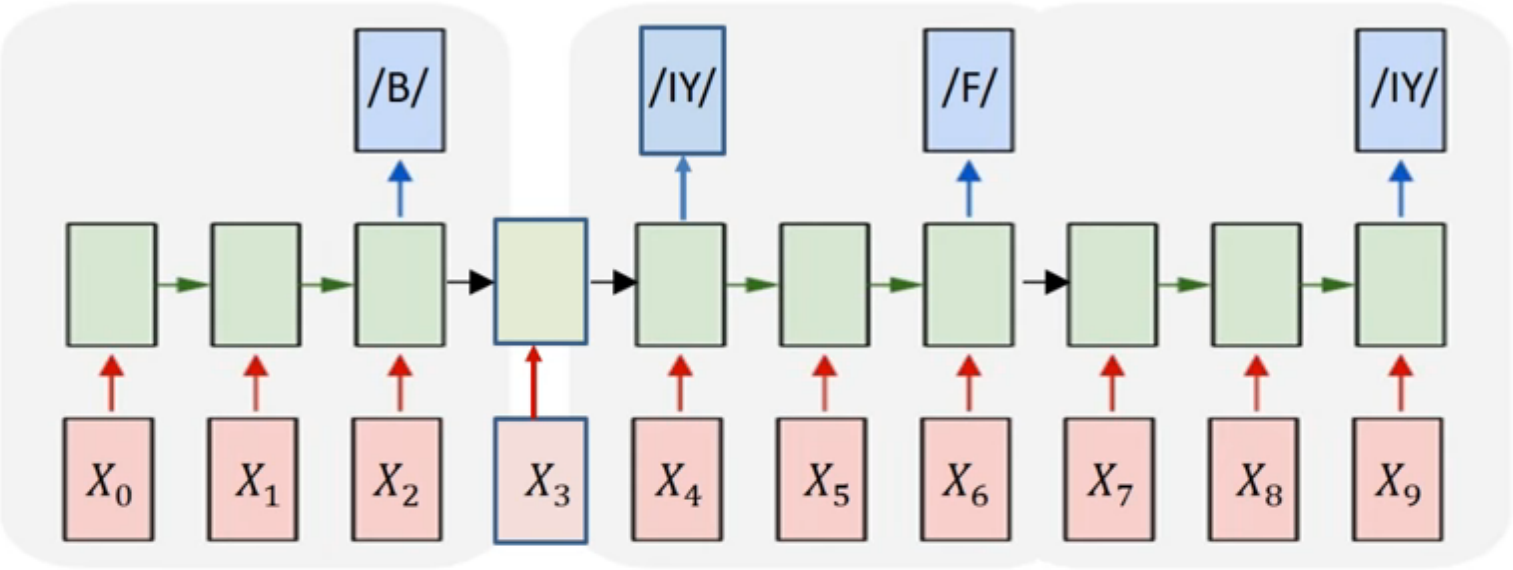

- Define the divergence everywhere

$$DIV(Y_{target}, Y) = \sum_t w_t Xent(Y(t), Phoneme)$$

- Typical weighting scheme for speech: all are equally important
- Problem like question answering: answer only expected after the question ends
  - Only $w_T$ is high, other weights are 0 or low

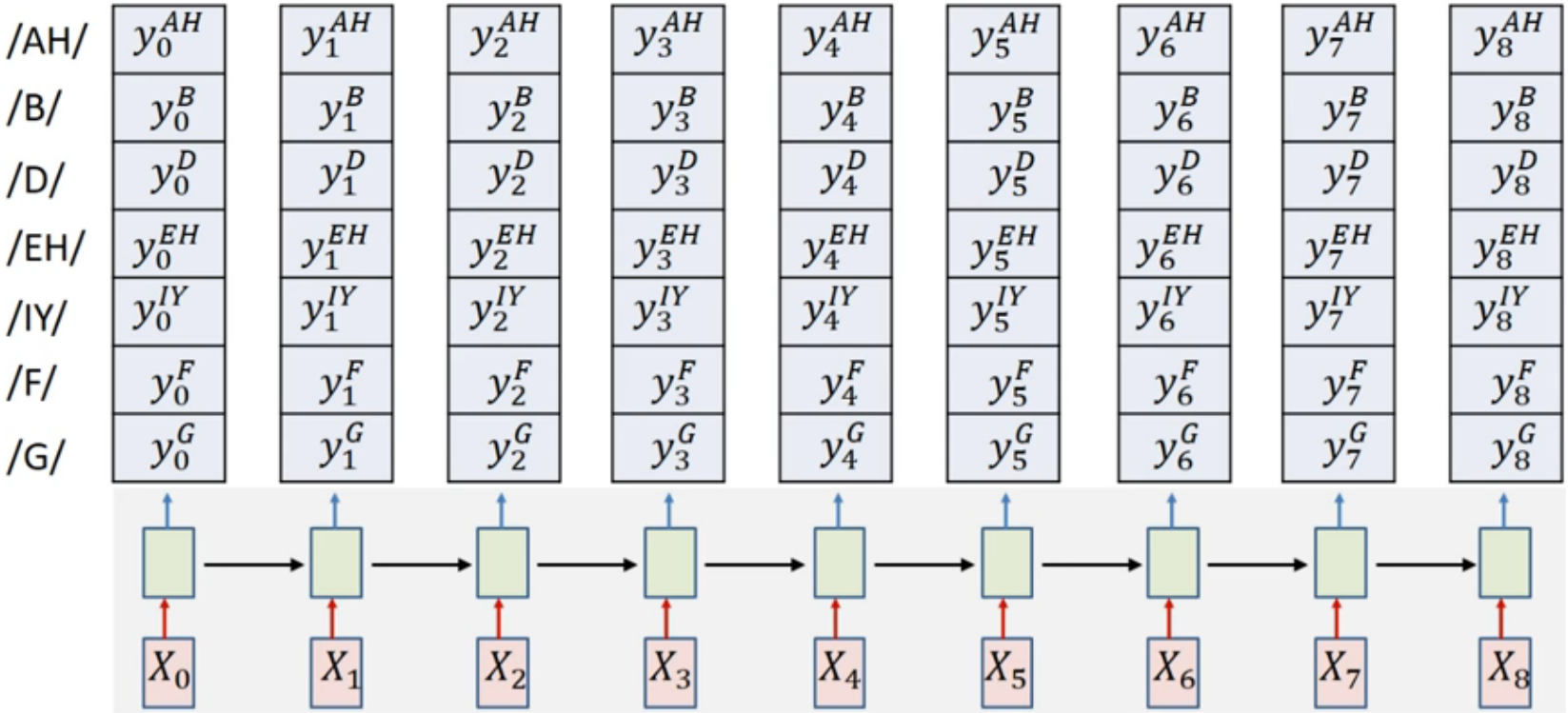
9

# The more complex problem




- Objective: Given a sequence of inputs, asynchronously output a sequence of symbols
  - This is just a simple concatenation of many copies of the simple "output at the end of the input sequence" model we just saw
- But this simple extension complicates matters..

# Overall objective

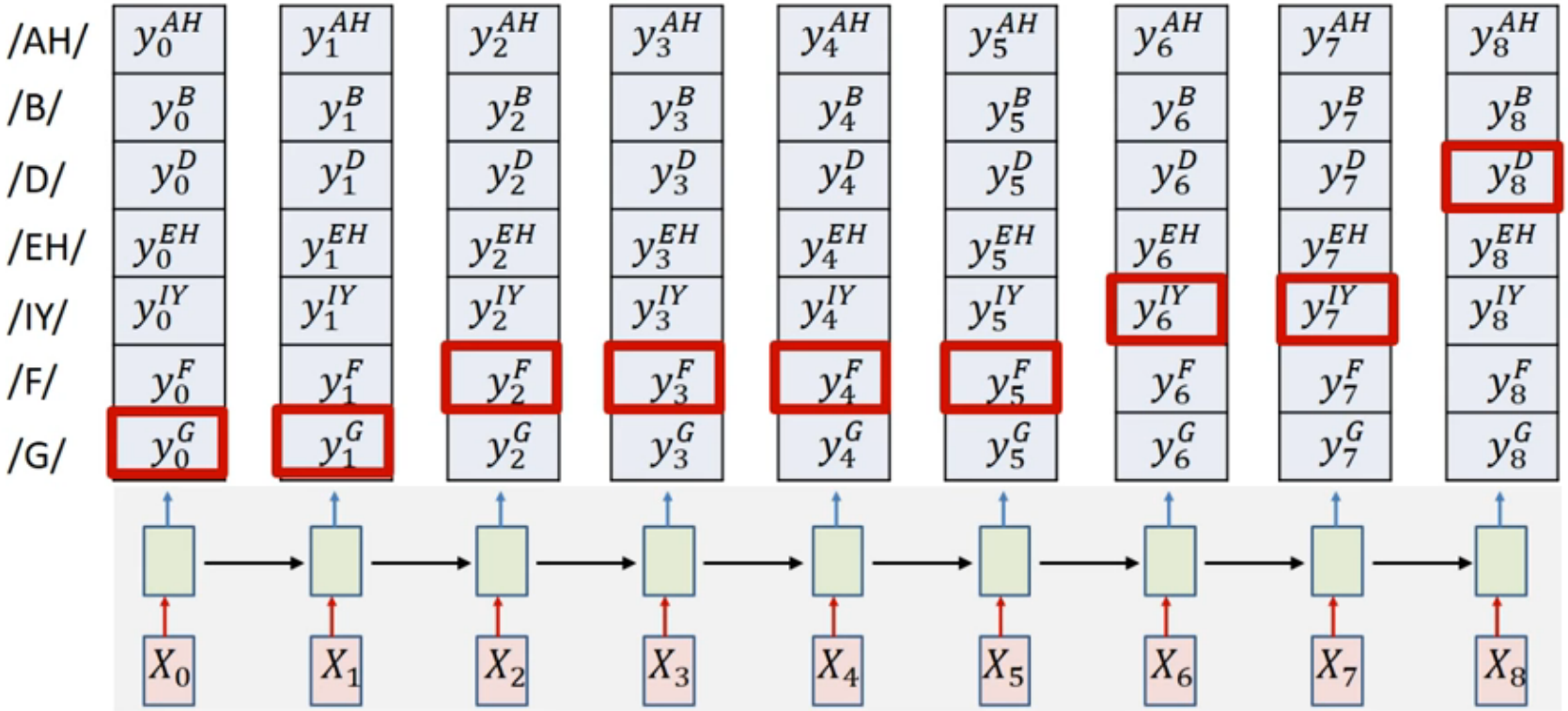



- Find most likely symbol sequence given inputs

$$S_0 \dots S_{K-1} = \underset{S'_0 \dots S'_{K-1}}{\operatorname{argmax}} \; prob(S'_0 \dots S'_{K-1} | X_0 \dots X_{N-1})$$

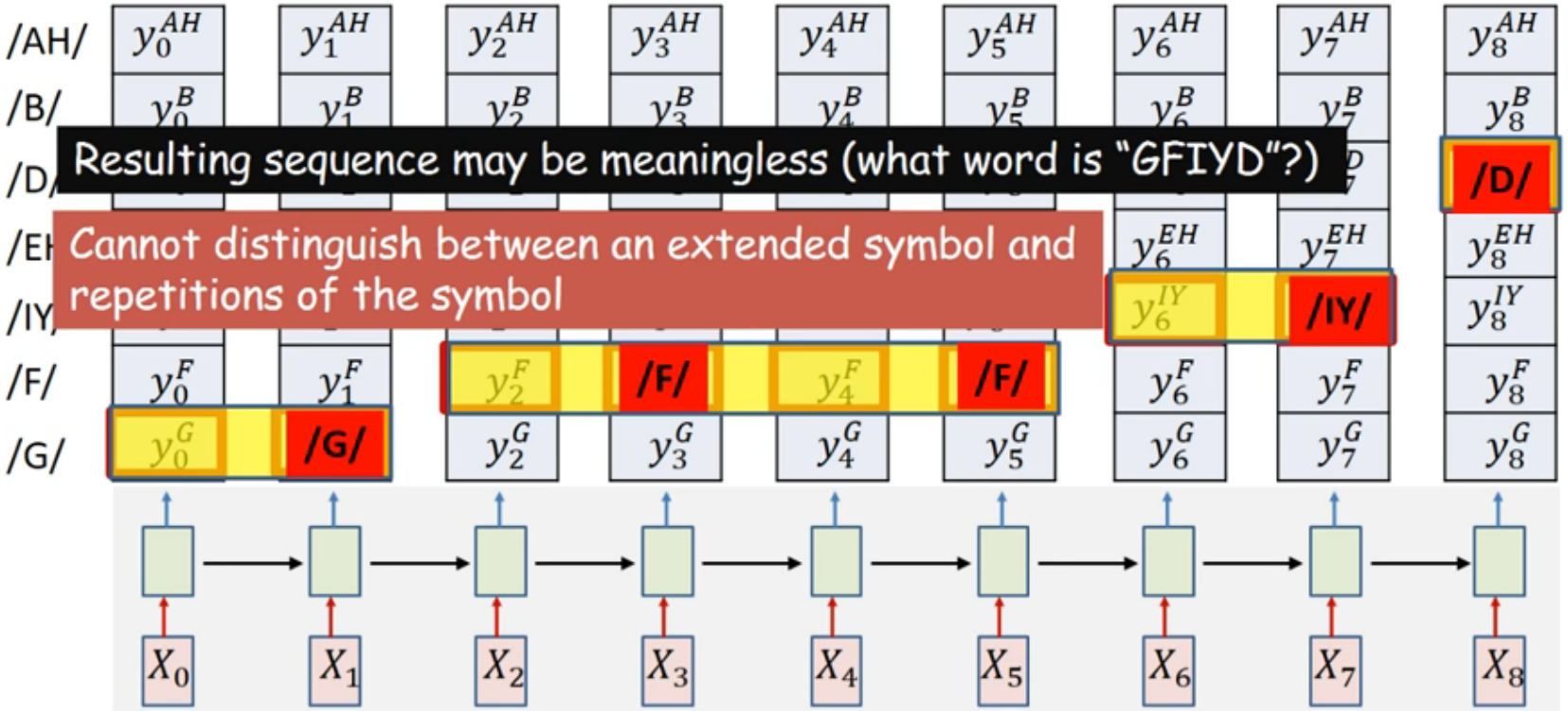
14

# Finding the best output




- Option 1: Simply select the most probable symbol at each time

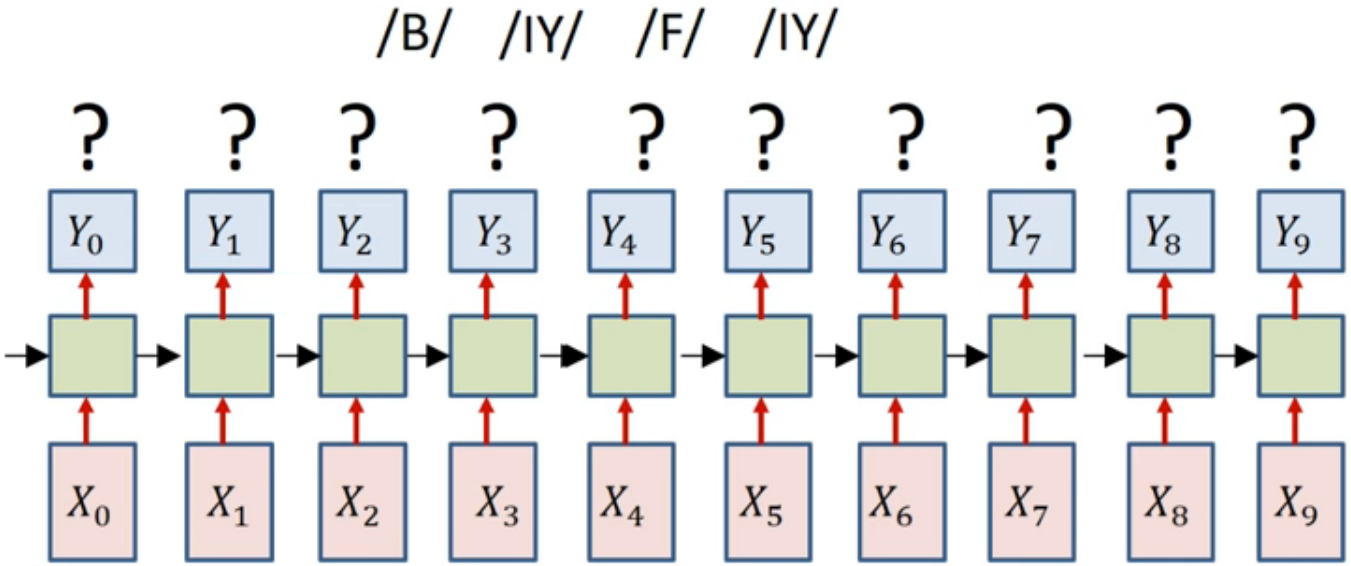
# The actual output of the network




- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

18

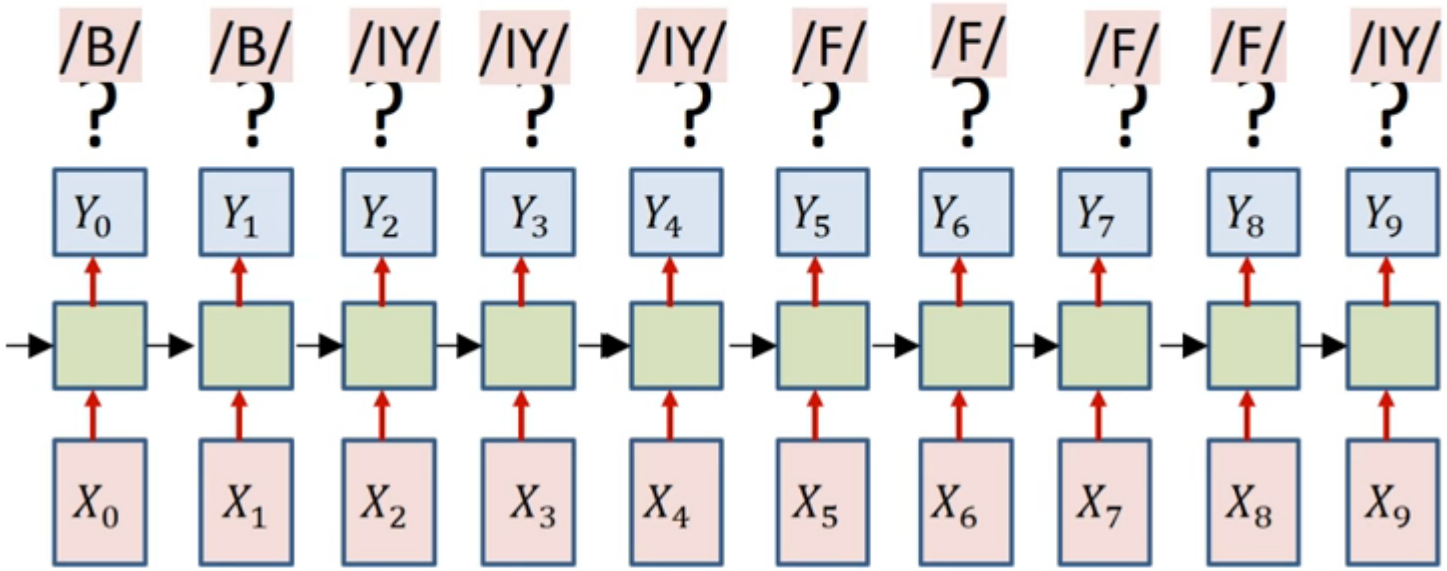
# Problem: No timing information provided




- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where
- How do we compute the divergence?
  - And how do we compute its gradient w.r.t. $Y_t$

26

# Solution 1: *Guess the alignment*




- Initialize: Assign an initial alignment
  - Either randomly, based on some heuristic, or any other rationale
- Iterate:
  - Train the network using the current alignment
  - *Reestimate* the alignment for each training instance
    - Using the decoding methods already discussed

27

# Estimating an alignment

- Given:
  - The unaligned $K$-length symbol sequence $S = S_0 \dots S_{K-1}$ (e.g. /B/ /IY/ /F/ /IY/)
  - An $N$-length input ($N \geq K$)
  - And a (trained) recurrent network

- Find:
  - An $N$-length expansion $s_0 \dots s_{N-1}$ comprising the symbols in S in strict order
    - e.g. $S_0 S_1 S_1 S_2 S_3 S_3 \dots S_{K-1}$
      - i.e. $s_0 = S_0, s_2 = S_1, S_3 = S_1, s_4 = S_2, s_5 = S_3, \dots s_{N-1} = S_{K-1}$
    - E.g. /B/ /B/ /IY/ /IY/ /IY/ /F/ /F/ /F/ /F/ /IY/ ..
  - $s_i = S_k \Rightarrow i \leq k$
  - $s_i = S_k,\ s_j = S_l, \quad i < j \Rightarrow k \leq l$
- Outcome: an *alignment* of the target symbol sequence $S_0 \dots S_{K-1}$ to the input $X_0 \dots X_{N-1}$

29

# Try 2: Explicitly arrange the constructed table

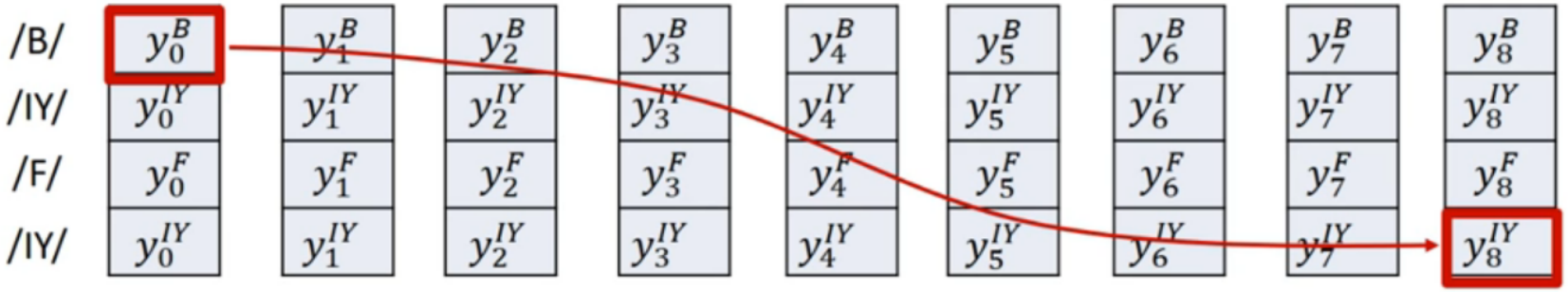
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| /B/ | $y_0^B$ | $y_1^B$ | $y_2^B$ | $y_3^B$ | $y_4^B$ | $y_5^B$ | $y_6^B$ | $y_7^B$ | $y_8^B$ |
| /IY/ | $y_0^{IY}$ | $y_1^{IY}$ | $y_2^{IY}$ | $y_3^{IY}$ | $y_4^{IY}$ | $y_5^{IY}$ | $y_6^{IY}$ | $y_7^{IY}$ | $y_8^{IY}$ |
| /F/ | $y_0^F$ | $y_1^F$ | $y_2^F$ | $y_3^F$ | $y_4^F$ | $y_5^F$ | $y_6^F$ | $y_7^F$ | $y_8^F$ |
| /IY/ | $y_0^{IY}$ | $y_1^{IY}$ | $y_2^{IY}$ | $y_3^{IY}$ | $y_4^{IY}$ | $y_5^{IY}$ | $y_6^{IY}$ | $y_7^{IY}$ | $y_8^{IY}$ |

Note: If a symbol occurs multiple times, we repeat the row in the appropriate location.
E.g. the row for /IY/ occurs twice, in the 2nd and 4th positions

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| /B/ | $y_0^B$ | $y_1^B$ | $y_2^B$ | $y_3^B$ | $y_4^B$ | $y_5^B$ | $y_6^B$ | $y_7^B$ | $y_8^B$ |
| /D/ | $y_0^D$ | $y_1^D$ | $y_2^D$ | $y_3^D$ | $y_4^D$ | $y_5^D$ | $y_6^D$ | $y_7^D$ | $y_8^D$ |
| /EH/ | $y_0^{EH}$ | $y_1^{EH}$ | $y_2^{EH}$ | $y_3^{EH}$ | $y_4^{EH}$ | $y_5^{EH}$ | $y_6^{EH}$ | $y_7^{EH}$ | $y_8^{EH}$ |
| /IY/ | $y_0^{IY}$ | $y_1^{IY}$ | $y_2^{IY}$ | $y_3^{IY}$ | $y_4^{IY}$ | $y_5^{IY}$ | $y_6^{IY}$ | $y_7^{IY}$ | $y_8^{IY}$ |
| /F/ | $y_0^F$ | $y_1^F$ | $y_2^F$ | $y_3^F$ | $y_4^F$ | $y_5^F$ | $y_6^F$ | $y_7^F$ | $y_8^F$ |
| /G/ | $y_0^G$ | $y_1^G$ | $y_2^G$ | $y_3^G$ | $y_4^G$ | $y_5^G$ | $y_6^G$ | $y_7^G$ | $y_8^G$ |

Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required

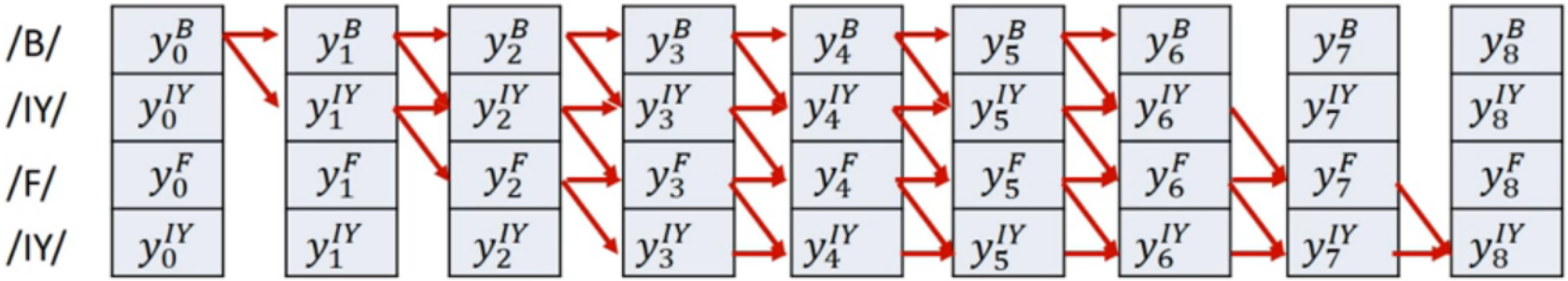
# Explicitly constrain alignment




- Constrain that the first symbol in the decode *must* be the top left block
- The last symbol *must* be the bottom right
- The rest of the symbols must follow a sequence that *monotonically* travels down from top left to bottom right
  - I.e. never goes up
- This guarantees that the sequence *is* an expansion of the target sequence
  - /B/ /IY/ /F/ /IY/ in this case

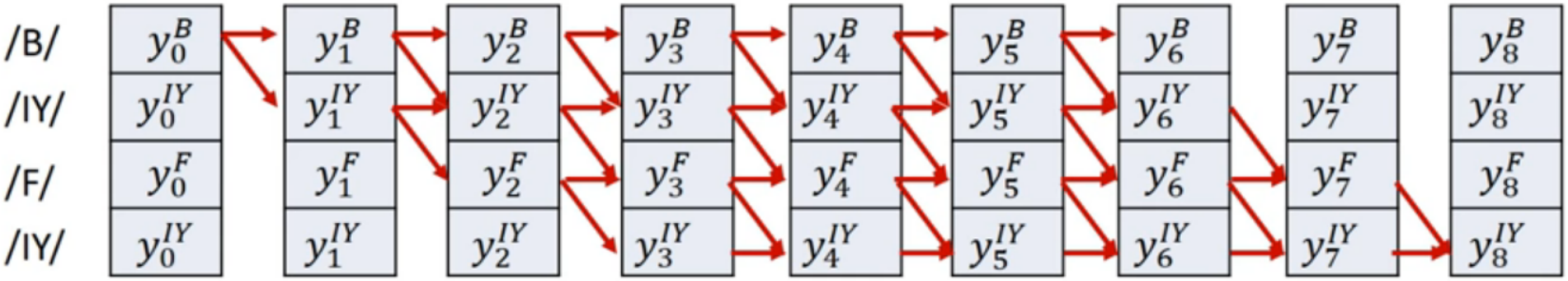
38

# Explicitly constrain alignment




- Compose a graph such that every path in the graph from source to sink represents a valid alignment
  - Which maps on to the target symbol sequence (/B//AH//T/)
- Edge scores are 1
- Node scores are the probabilities assigned to the symbols by the neural network
- The "score" of a path is the product of the probabilities of all nodes along the path
- Find the most probable path from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm

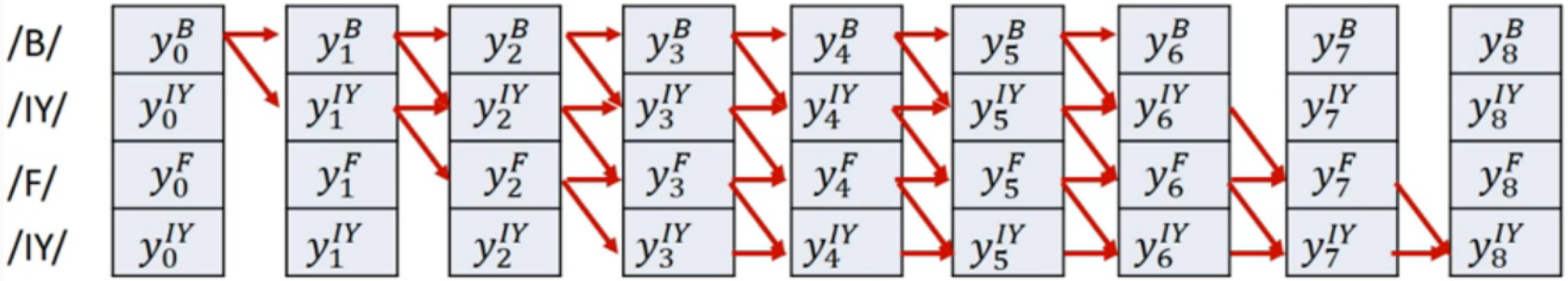
# Viterbi algorithm




- At each node, keep track of
  - The best incoming edge
  - The score of the best path from the source to the node
- Dynamically compute the best path from source to sink

40

# Viterbi algorithm



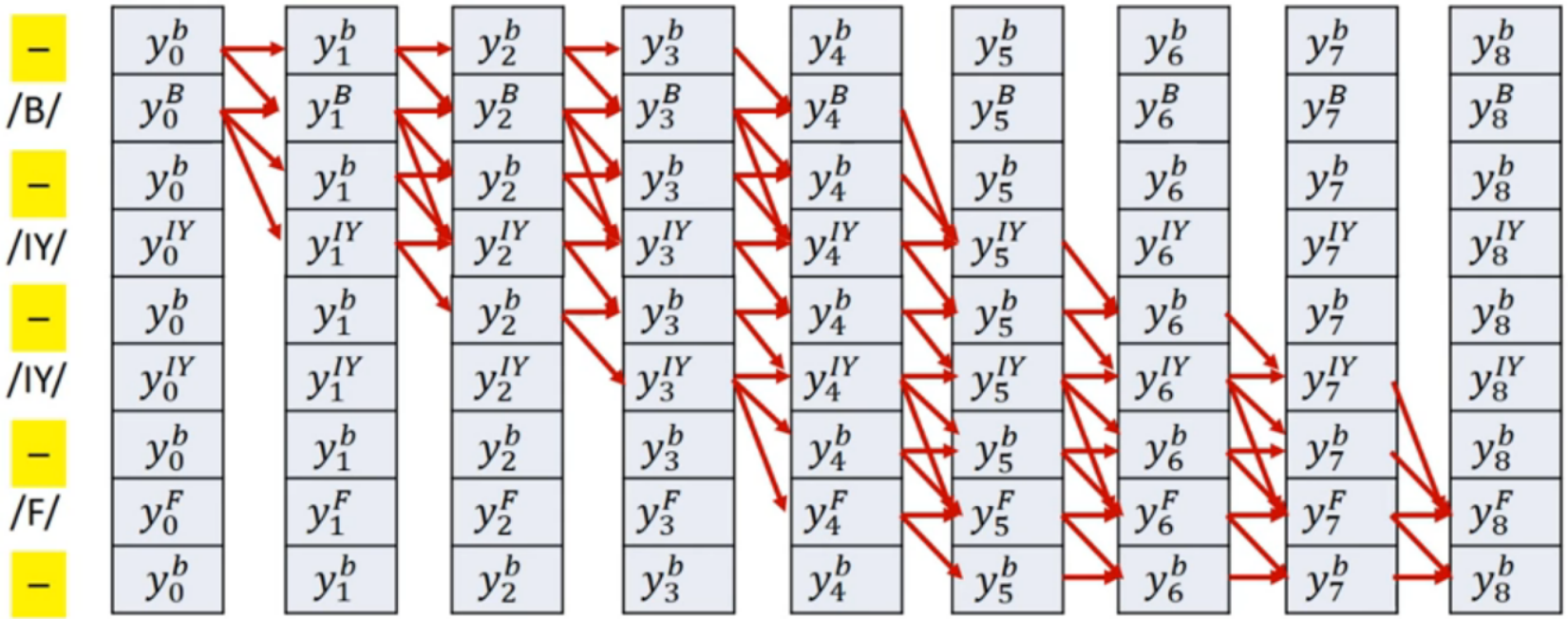

- First, some notation:
- $y_t^{S(r)}$ is the probability of the target symbol assigned to the $r$-th row in the $t$-th time (given inputs $X_1 \dots X_t$)
  - E.g., S(0) = /B/
    - The scores in the 0th row have the form $y_t^B$
  - E.g. S(1) = S(3) = /IY/
    - The scores in the 1st and 3rd rows have the form $y_t^{IY}$
  - E.g. S(2) = /F/
    - The scores in the 2nd row have the form $y_t^F$

# Composing the graph for training

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| – | $y_0^b$ | $y_1^b$ | $y_2^b$ | $y_3^b$ | $y_4^b$ | $y_5^b$ | $y_6^b$ | $y_7^b$ | $y_8^b$ |
| /B/ | $y_0^B$ | $y_1^B$ | $y_2^B$ | $y_3^B$ | $y_4^B$ | $y_5^B$ | $y_6^B$ | $y_7^B$ | $y_8^B$ |
| – | $y_0^b$ | $y_1^b$ | $y_2^b$ | $y_3^b$ | $y_4^b$ | $y_5^b$ | $y_6^b$ | $y_7^b$ | $y_8^b$ |
| /IY/ | $y_0^{IY}$ | $y_1^{IY}$ | $y_2^{IY}$ | $y_3^{IY}$ | $y_4^{IY}$ | $y_5^{IY}$ | $y_6^{IY}$ | $y_7^{IY}$ | $y_8^{IY}$ |
| – | $y_0^b$ | $y_1^b$ | $y_2^b$ | $y_3^b$ | $y_4^b$ | $y_5^b$ | $y_6^b$ | $y_7^b$ | $y_8^b$ |
| /IY/ | $y_0^{IY}$ | $y_1^{IY}$ | $y_2^{IY}$ | $y_3^{IY}$ | $y_4^{IY}$ | $y_5^{IY}$ | $y_6^{IY}$ | $y_7^{IY}$ | $y_8^{IY}$ |
| – | $y_0^b$ | $y_1^b$ | $y_2^b$ | $y_3^b$ | $y_4^b$ | $y_5^b$ | $y_6^b$ | $y_7^b$ | $y_8^b$ |
| /F/ | $y_0^F$ | $y_1^F$ | $y_2^F$ | $y_3^F$ | $y_4^F$ | $y_5^F$ | $y_6^F$ | $y_7^F$ | $y_8^F$ |
| – | $y_0^b$ | $y_1^b$ | $y_2^b$ | $y_3^b$ | $y_4^b$ | $y_5^b$ | $y_6^b$ | $y_7^b$ | $y_8^b$ |

- With blanks
- Note: a row of blanks between any two symbols
- Also blanks at the very beginning and the very end

# Composing the graph for training




- Add edges such that all paths from initial node(s) to final node(s) unambiguously represent the target symbol sequence

# CTC - Connectionist Temporal Classification

## CTC Loss Function

- Maximum log likelihood training of transcript
- Intuition: Alignments are unknown so integrate over all possible time-character alignments

$$\mathcal{L}_{\text{CTC}}(X, W) = \sum_{C:\ \kappa(C)=W} p(C|X)$$

$$= \sum_{C:\ \kappa(C)=W} \prod_{t=1}^{T} p(c_t|X).$$

- Example: W = “hi”, T = 3
  possible C such that K(C) = W:
  `hhi`, `hii`, `_hi`, `h_i`, `hi_`

(Graves & Jaitly. 2014)

# CTC Objective Function

Labels at each time index are conditionally independent (like HMMs)

$$\Pr(\mathrm{a}|x) = \prod_{t=1}^{T} \Pr(a_t, t|x)$$

Sum over all time-level labelings consistent with the output label.

$$\Pr(y|x) = \sum_{a \in \mathcal{B}^{-1}(y)} \Pr(a|x)$$

Output label: AB

Time-level labelings: AB, _AB, A_B, … _A_B_

Final objective maximizes probability of true labels:

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

(Graves & Jaitly, ICML 2014)

# Collapsing Example

**Per-frame argmax:**

_____
yy__ee_____tt_ _____a_____
_rr__e_____hh_____b___ii_____lll__i_____tt_____aa_____tt_____iio__n___
___cc_____rrr_u_____ _____ii___ss
_____o_____nn_____hhh_a_____nnddd _____i__n___
__thh_e_____ _____bb_uuii_____llldd____ii____nng_____
_____l___o___o_g__g___ii____nng_____
____b___rr_ii_____ck__s_____p___ll__a_____sstt_____eerr__
_____a___nnd_ ___b___lll_uu____ee__pp___r___i_____nnss_
_____f_____oou_____rrr_____ _____f_____oo__rrr__tt_y____
_____t____www_oo_____ ____nn___ew_____
_____b___e_____t_____i____n___
____e_____pp_____aa___rr___tt____mm_ee___nnntss
_____
_____

**After collapsing:**
yet a rehbilitation cru is onhand in the building loogging bricks plaster and blueprins four forty two new betin epartments

**Reference:**
yet a rehabilitation crew is on hand in the building lugging bricks plaster and blueprints for forty two new bedroom apartments

(Hannun, Maas, Jurafsky, & Ng. 2014)

# Beam Search Decoding

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

## Connectionist Temporal Classification (CTC)

- Basic idea:
  1. RNN output neurons $c$ encode distribution over symbols. Note length(c) == length(x).

     For phoneme-based model: $c \in \{AA, AE, AX, ..., ER1, \text{blank}\}$

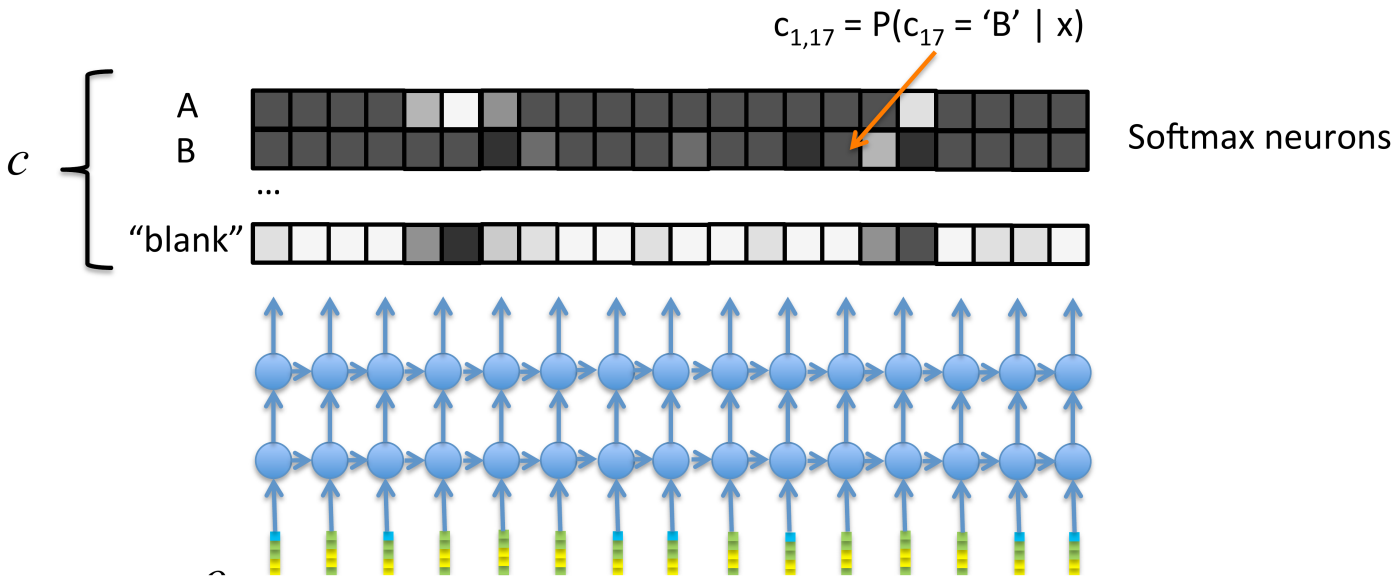
     For grapheme-based model: $c \in \{A, B, C, D, ..., Z, \text{blank}, \text{space}\}$

  2. Define a mapping $\beta(c) \rightarrow y$.
  3. Maximize likelihood of $y^*$ under this model.

## Connectionist Temporal Classification (CTC)

1. RNN output neurons $c$ encode distribution over symbols. Note length(c) == length(x).

For grapheme-based model: $c \in \{A, B, C, D, ..., Z, \text{blank}, \text{space}\}$



## Connectionist Temporal Classification (CTC)

1. RNN output neurons $c$ encode distribution over symbols. Note length(c) == length(x).

   For grapheme-based model: $c \in \{A, B, C, D, ..., Z, \text{blank}, \text{space}\}$

- Output neurons define distribution over whole character sequences $c$ assuming independence:

$$P(c|x) \equiv \prod_{i=1}^{N} P(c_i|x)$$

$$P(c = \text{HHH_E__LL_LO___}|x) = P(c_1 = \text{H}|x)P(c_2 = \text{H}|x) \cdots P(c_{15} = \text{blank}|x)$$

Connectionist Temporal Classification (CTC)

2. Define a mapping $\beta(c) \rightarrow y$.
   – Given a specific character sequence $c$, squeeze out duplicates + blanks to yield transcription:

$$y = \beta(c) = \beta(\text{HHH_E__LL_LO___}) = \text{“HELLO”}$$

## Connectionist Temporal Classification (CTC)

- Mapping implies a distribution over possible *transcriptions* $y$:

$P(c|x) = \{$

| | | |
|---|---|---|
| 0.1 | HHH_E__LL_LO___ | "HELLO" v.1 |
| 0.02 | HH__E__LL_LO___ | "HELLO" v.2 |
| 0.01 | HHH_E__L_L_OH__ | "HELL OH" |
| 0.01 | HHH_EE_LL_L_O__ | "HELLO" v.3 |
| … | YY__E__LL_LO_W_ | "YELLOW" |

$$P(y|x) = \sum_{c:\beta(c)=y} P(c|x)$$

$P($"HELLO"$) =$ 0.1 + 0.02 + 0.01 + …

# CTC - Backprop

Connectionist Temporal Classification (CTC)

3. Update network parameters $\theta$ to maximize likelihood of correct label $y$*:

$$\theta^\star = \arg\max_\theta \sum_i \log P(y^{\star(i)}|x^{(i)})$$
$$= \arg\max_\theta \sum_i \log \sum_{c:\beta(c)=y^{\star(i)}} P(c|x^{(i)})$$

- [Graves et al., 2006] provides an efficient dynamic programming algorithm to compute the inner summation and its gradient.

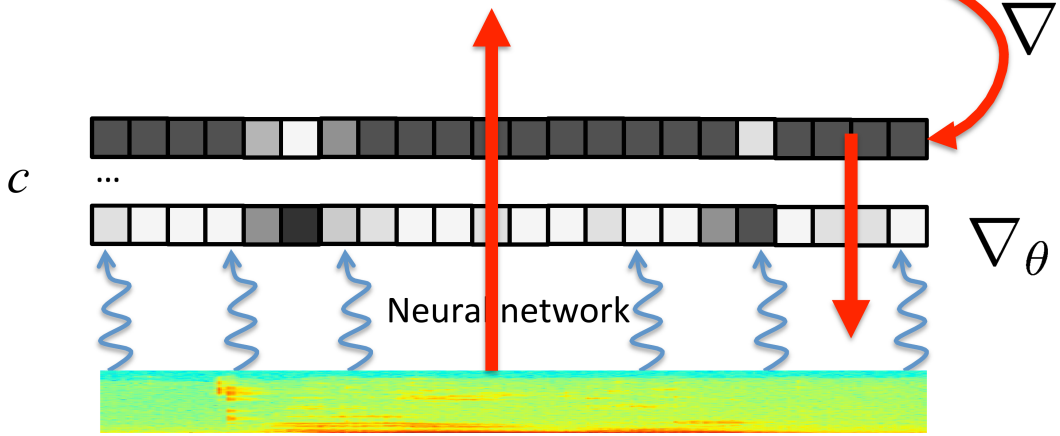
## Connectionist Temporal Classification (CTC)

- Use usual gradient descent methods to optimize.
  Tune entire network with backpropagation.
  - Given network outputs, many off-the-shelf packages to compute CTC loss (likelihood) from $c$ and $y$*, and gradient w.r.t. $c$.
    - Warp CTC: github.com/baidu-research/warp-ctc
    - Stanford CTC: github.com/amaas/stanford-ctc
    - Tensorflow: tf.nn.ctc_loss

Efficient OTS software

$$L(\theta) = \log P(y^{*(i)} \mid x^{(i)}) = \textbf{CTC}(c^{(i)}, y^{*(i)})$$

$$\nabla_c \textbf{CTC}(c^{(i)}, y^{*(i)})$$

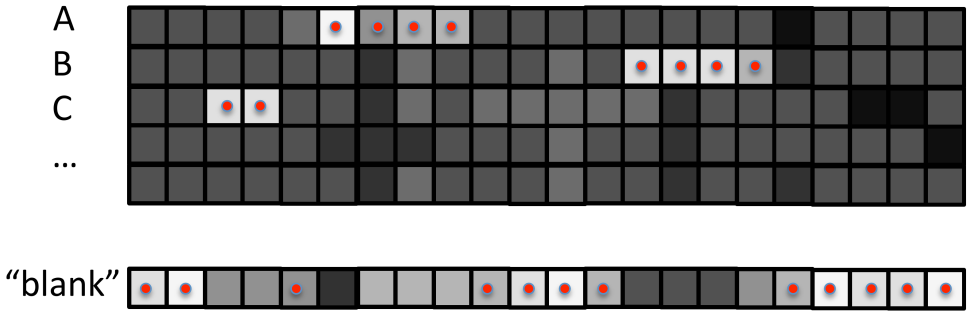



$$\nabla_\theta$$

# Decoding

- Network outputs P(c|x). How do we find most likely transcription from P(y|x)?
- Simple (approximate) solution:

$$\beta\left(\arg\max_{c} \; P(c|x)\right)$$

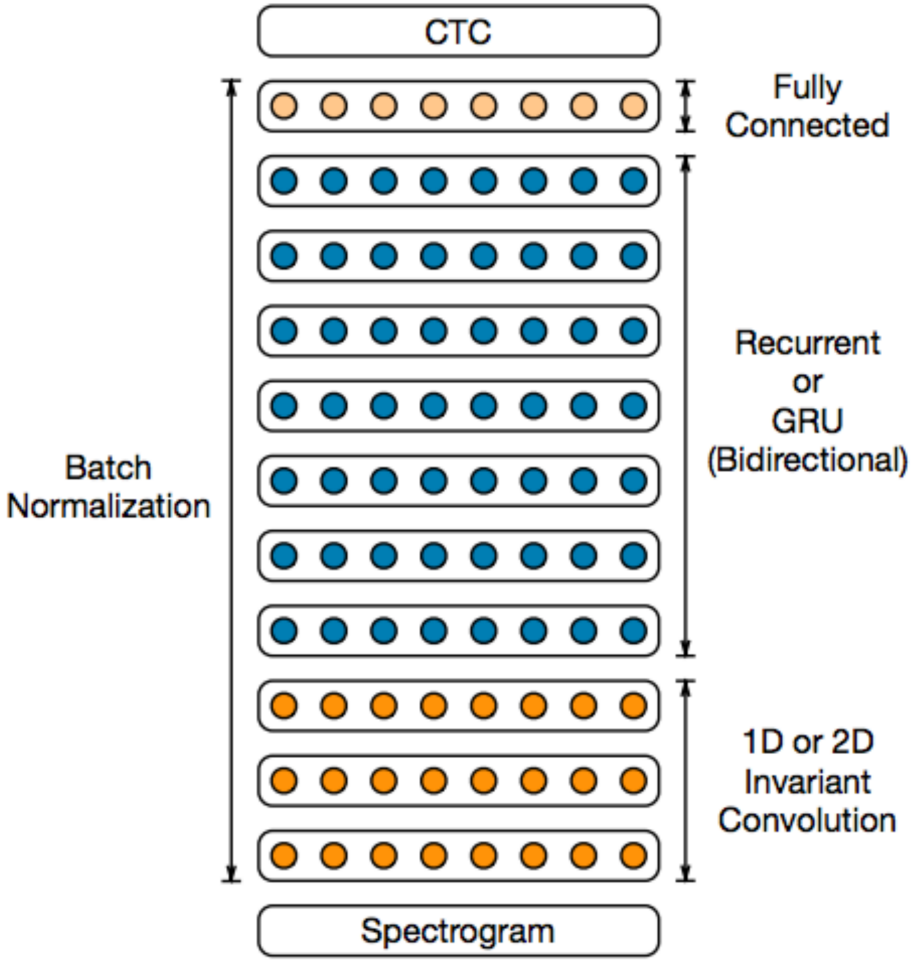



"CAB"

- Often terrible, but a useful diagnostic to "eyeball" models.

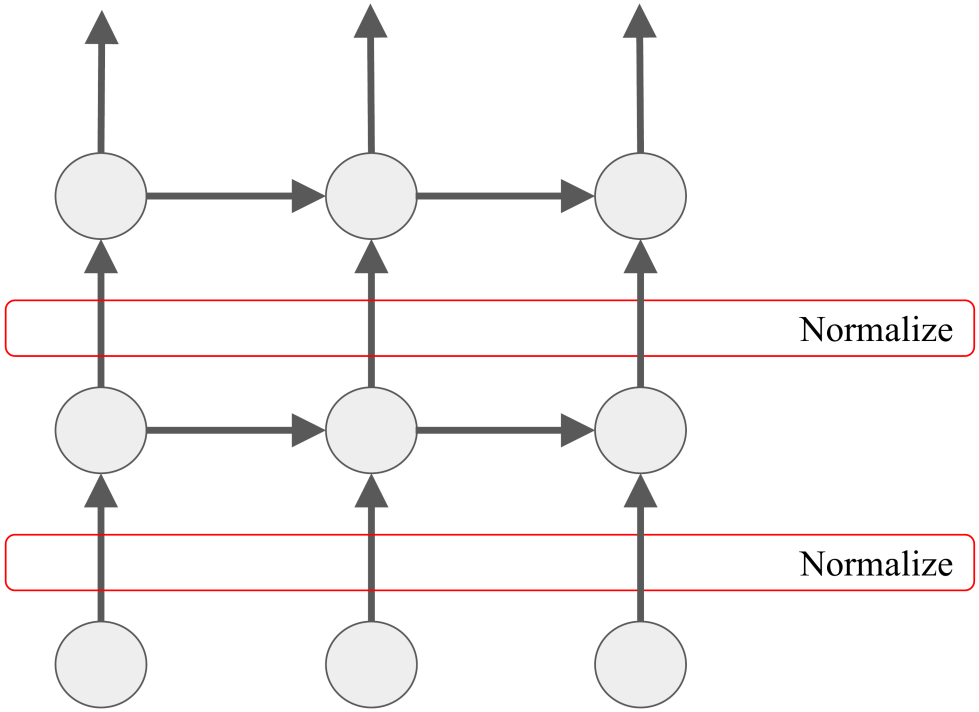
# Deepspeech architecture

# Deep Speech – Deep RNN

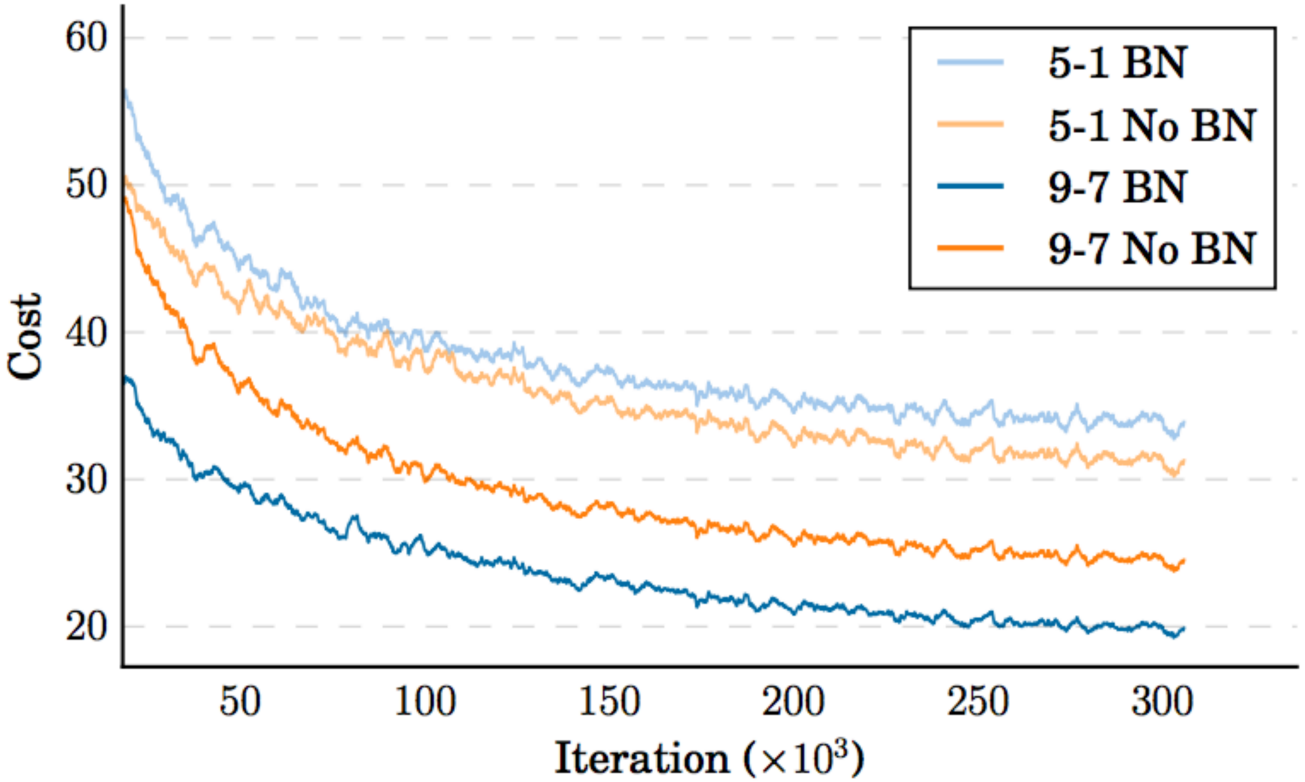



Slides from Awni Hannun

# Deep Speech – Batch Norm for RNNs



# Deep Speech – Batch Norm for RNNs



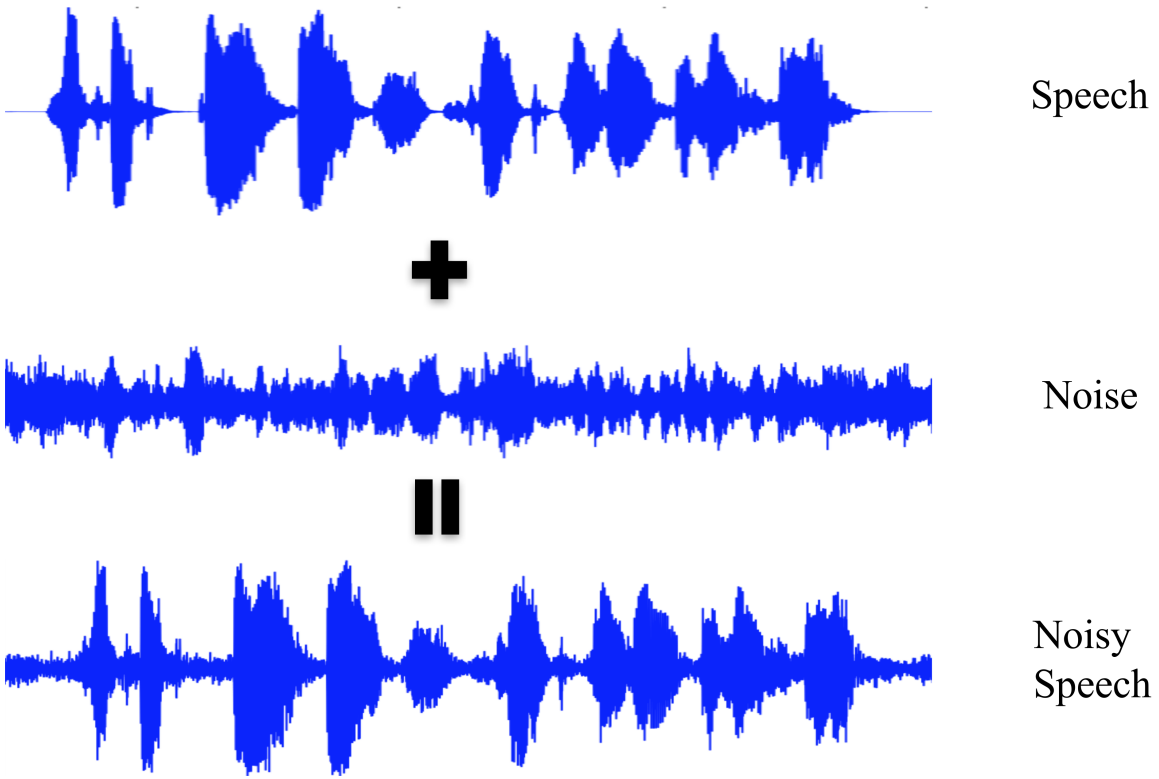

Slides from Awni Hannun

## Deep Speech - Captioned Video Data Pipeline

1. Download publicly available video + captions.
1. Align caption to video with CTC Model
1. Segment at regions of silence
1. Use simple classifier to throw out very noisy samples.

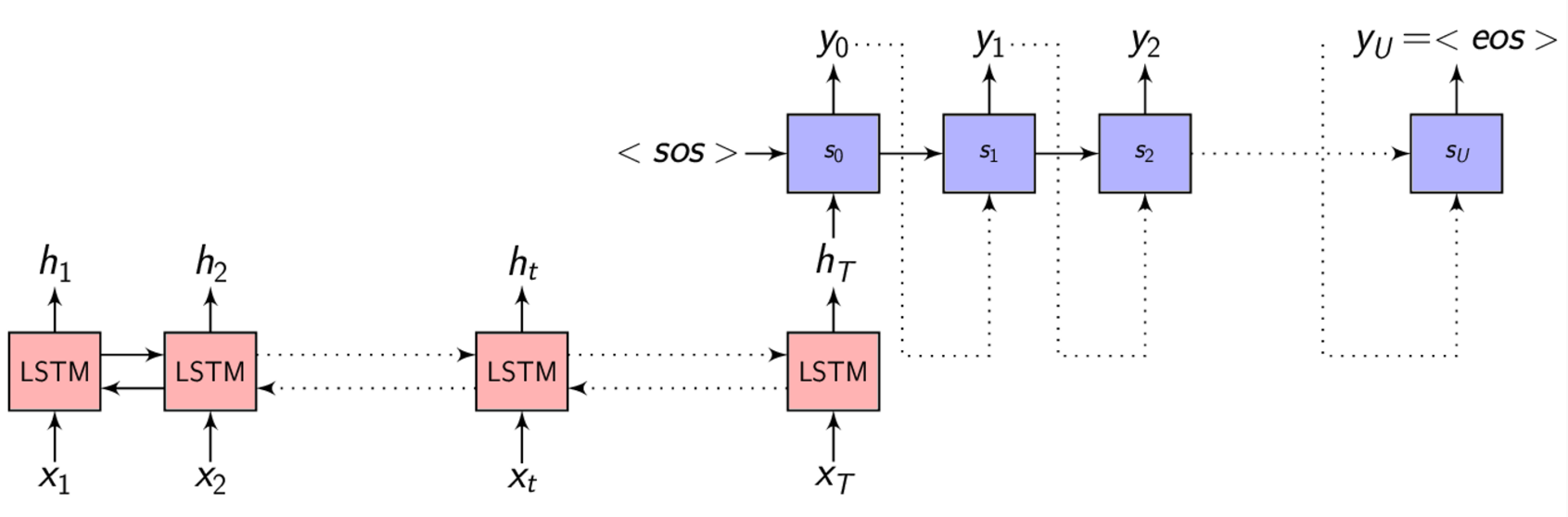
# Deep Speech - Even more data!

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



# Beyond DeepSpeech

# Encoder-Decoder



# Listen, Attend, and Spell

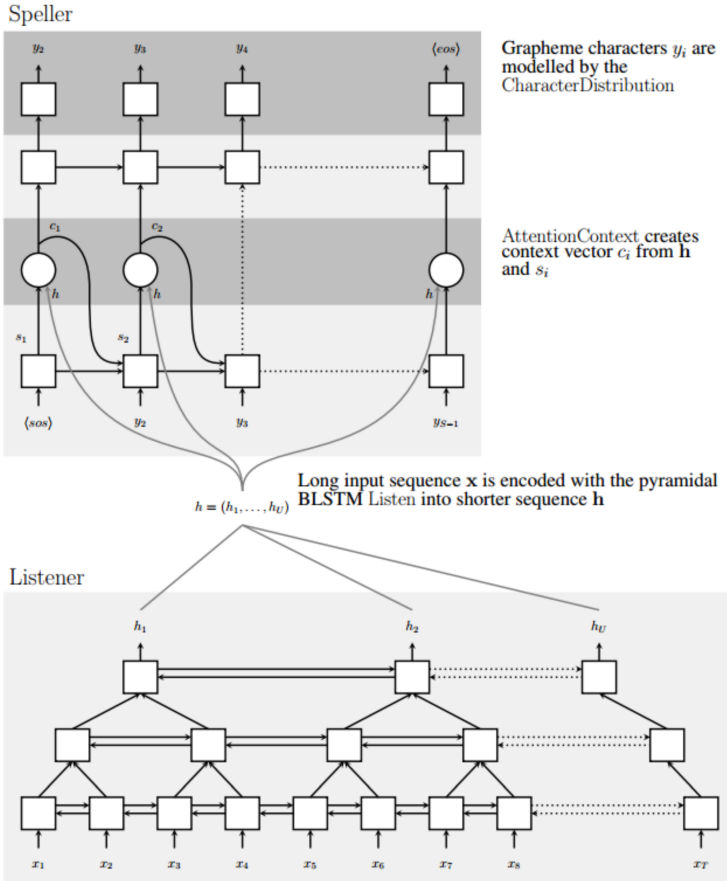
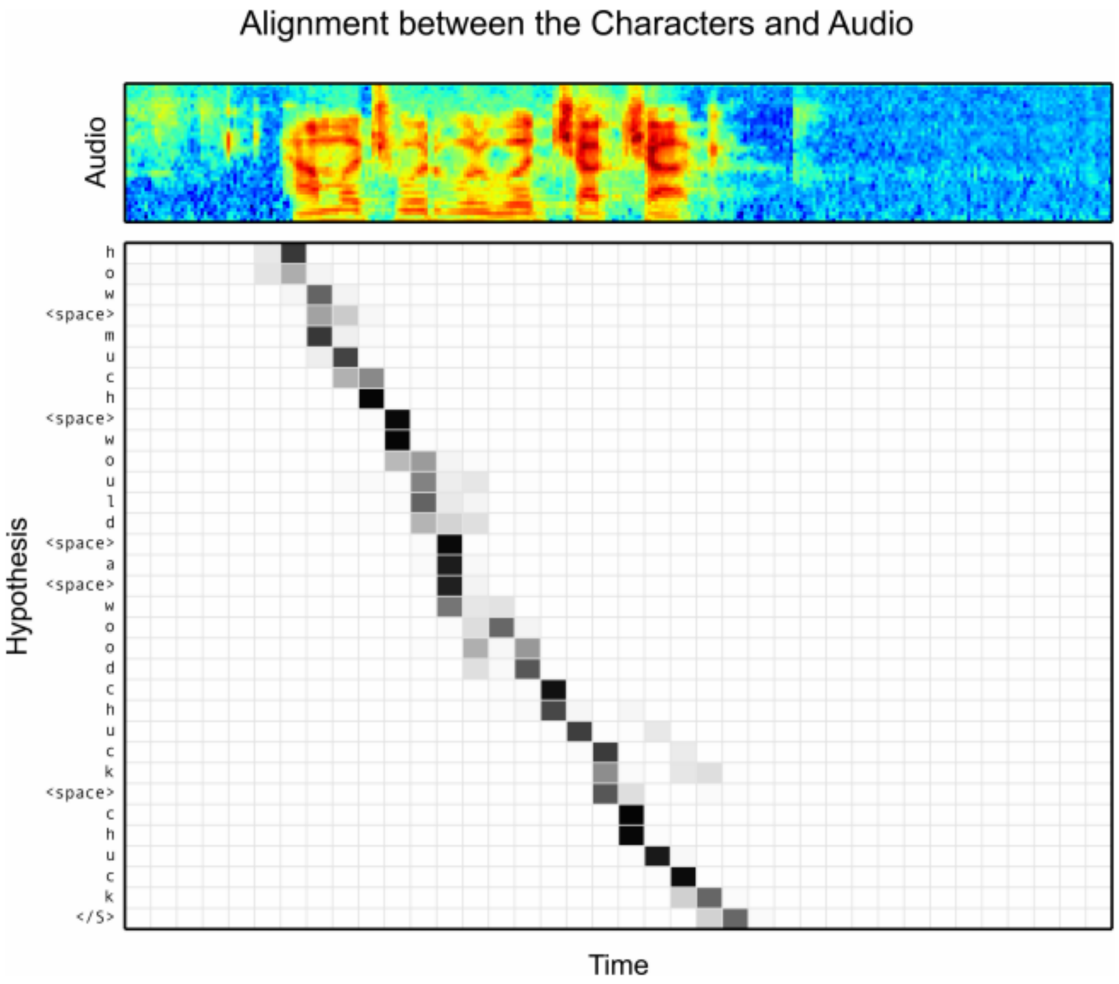



Figure 1: Listen, Attend and Spell (LAS) model: the listener is a pyramidal BLSTM encoding our input sequence $\mathbf{x}$ into high level features $\mathbf{h}$, the speller is an attention-based decoder generating the $\mathbf{y}$ characters from $\mathbf{h}$.

(Chan, Jaitly, Le, & Vinyals. 2015)

# Listen, Attend, and Spell




(Chan, Jaitly, Le, & Vinyals. 2015)

# Listen, Attend, and Spell

Table 1: WER comparison on the clean and noisy Google voice search task. The CLDNN-HMM system is the state-of-the-art system, the Listen, Attend and Spell (LAS) models are decoded with a beam size of 32. Language Model (LM) rescoring was applied to our beams, and a sampling trick was applied to bridge the gap between training and inference.

| Model | Clean WER | Noisy WER |
|---|---|---|
| CLDNN-HMM [20] | 8.0 | 8.9 |
| LAS | 16.2 | 19.0 |
| LAS + LM Rescoring | 12.6 | 14.7 |
| LAS + Sampling | 14.1 | 16.5 |
| LAS + Sampling + LM Rescoring | 10.3 | 12.0 |

(Chan, Jaitly, Le, & Vinyals. 2015)

# Attention-based sequence generation

- Maximum likelihood conditional language model given the aud $p(y_1)\prod_{t=2}^{T} p(y_t|y_1,\cdots,y_{t-1})$

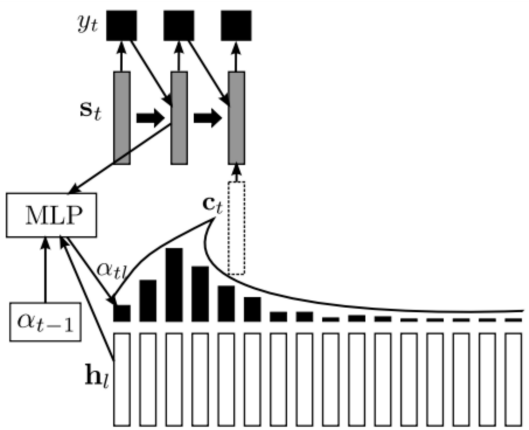



**Fig. 3**. Schematic representation of the Attention-based Recurrent Sequence Generator. At each time step $t$, an MLF combines the hidden state $\mathbf{s}_{t-1}$ with all the input vectors $\mathbf{h}_l$ to compute the attention weights $\alpha_{tl}$. Subsequently, the new hidden state $\mathbf{s}_t$ and prediction for output label $y_t$ can be computed.

(Bahdanau *et al*. 2016)

# Summary

# Summary

- Input speech $X = x_1, x_2, x_3, ....., x_T$
- Feature vector sequence $O = o_1, o_2, o_3, ....., o_T$
- Expected output sequence $Y = y_1, y_2, ....., y_U$
- Bi-RNN/Bi-LSTM encodes the feature sequence
- Use CTC to compute loss between $BiLSTM(O)$ and $Y$
- CTC Beam Search Decoding to decode the test audio
- Language model to re-score the output.
- Encoder-Decoder and Attention to improve modeling