Deep Learning
Recurrent Networks: Part 4
Spring 2021
• Recurrent structures can be trained by minimizing the divergence between the *sequence* of outputs and the *sequence* of desired outputs
  – Through gradient descent and backpropagation

• The challenge: Defining this divergence
  – Inputs and outputs may not be time aligned or even synchronous
Variants of recurrent nets

• Conventional MLP
• Time-synchronous outputs
  – E.g. part of speech tagging
This is a regular MLP

- No recurrence
  - Exactly as many outputs as inputs
  - The output at time $t$ is unrelated to the output at $t' \neq t$. 
Learning in a regular MLP for series

- In the context of analyzing time series, the divergence to minimize is still the divergence between two series
  - Must be differentiable w.r.t every $Y(t)$

- In this setting: One-to-one correspondence between actual and target outputs

- Common assumption: Total divergence is the sum of local divergences at individual times
  - Simplifies model and maths
“Series MLP” as a regular MLP

- Gradient backpropagated at each time
  \[ \nabla_{Y(t)} \text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) \]
- Common assumption: One-to-one correspondence
  \[ \text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) = \sum_t \text{Div}(Y_{\text{target}}(t), Y(t)) \]
  \[ \nabla_{Y(t)} \text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) = \nabla_{Y(t)} \text{Div}(Y_{\text{target}}(t), Y(t)) \]
  
  - This is further backpropagated to update weights etc

Typical Divergence for classification: \[ \text{Div}(Y_{\text{target}}(t), Y(t)) = KL(Y_{\text{target}}(t), Y(t)) \]
Variants of recurrent nets

- Conventional MLP
- Time-synchronous outputs
  - E.g. part of speech tagging

Images from Karpathy
Time synchronous network

- Network produces one output for each input
  - With one-to-one correspondence
  - E.g. Assigning grammar tags to words
    - May require a bidirectional network to consider both past and future words in the sentence

```
h_{-1} → CD → NNS → VBD → IN → DT → JJ → NN
  two → roads → diverged → in → a → yellow → wood
```
Time-synchronous networks: Inference

- One sided network: Process input left to right and produce output after each input
- Bi-directional network: Process input in both directions
- In all cases, there is an output for every input with exact one-to-one time-synchronous correspondence
  - Will continue to assume unidirectional models for explanations
Back Propagation Through Time

- Train given a set of input-target output pairs that are time synchronous
  - \((\mathbf{X}_i, \mathbf{D}_i)\), where \(\mathbf{X}_i = X_{i,0}, \ldots, X_{i,T}\), \(\mathbf{D}_i = D_{i,0}, \ldots, D_{i,T}\)

- The divergence computed is between the sequence of outputs by the network and the desired sequence of outputs
  \(\text{Div}(Y_{target}(1 \ldots T), Y(1 \ldots T))\)
Back Propagation Through Time

First step of backprop: Compute $\nabla_{Y(t)} \text{DIV}$ for all $t$

- The key component is the computation of this derivative!!
- This depends on the definition of “DIV”
BPTT: Time-synchronous recurrence

• Usual assumption: *Sequence divergence is the sum of the divergence at individual instants*

\[
\text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) = \sum_t \text{Div}(Y_{\text{target}}(t), Y(t))
\]

\[
\nabla_{Y(t)} \text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) = \nabla_{Y(t)} \text{Div}(Y_{\text{target}}(t), Y(t))
\]

Typical Divergence for classification: \(\text{Div}(Y_{\text{target}}(t), Y(t)) = KL(Y_{\text{target}}(t), Y(t))\)
Variants of recurrent nets

- Sequence classification: Classifying a full input sequence
  - E.g. phoneme recognition
- Order synchronous, time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori
• Question answering
• Input: Sequence of words
• Output: Answer at the end of the question
• Speech recognition
• Input: Sequence of feature vectors (e.g. Mel spectra)
• Output: Phoneme ID at the end of the sequence
  – Represented as an N-dimensional output probability vector, where N is the number of phonemes
Inference: Forward pass

• Exact input sequence provided
  – Output generated when the last vector is processed
    • Output is a probability distribution over phonemes

• But what about at *intermediate stages*?
Forward pass

- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes

- Outputs are actually produced for every input
  - We only *read* it at the end of the sequence
The Divergence is only defined at the final input

\[ DIV(Y_{\text{target}}, Y) = KL(Y(T), \text{Phoneme}) \]

This divergence must propagate through the net to update all parameters
Training

Shortcoming: Pretends there’s no useful information in these

• The Divergence is only defined at the final input
  \[ DIV(Y_{target}, Y) = KL(Y(T), Phoneme) \]
• This divergence must propagate through the net to update all parameters
• Exploiting the untagged inputs: assume the same output for the entire input

• Define the divergence everywhere

\[
DIV(Y_{target}, Y) = \sum_{t} w_t KL(Y(t), Phoneme)
\]
Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme

- Define the divergence everywhere

\[
DIV(Y_{target}, Y) = \sum_t w_t KL(Y(t), \text{Phoneme})
\]

- 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
Variants on recurrent nets

- Sequence classification: Classifying a full input sequence
  - E.g phoneme recognition
- Order synchronous, time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori
A 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..
The *sequence-to-sequence* problem

- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs
    - Outputs that represent the definitive occurrence of a symbol
The actual output of the network

At each time the network outputs a probability for each output symbol given all inputs until that time

- E.g. $y_{4}^{D} = \text{prob}(s_4 = D | X_0 \ldots X_4)$
Recap: The output of a network

• Any neural network with a softmax (or logistic) output is actually outputting an estimate of the *a posteriori* probability of the classes given the output

\[ P(c_1|X), P(c_2|X), \ldots, P(c_K|X) \]

• Selecting the class with the highest probability results in *maximum a posteriori probability* classification

\[ \text{Class} = \arg\max_i P(Y_i|X) \]

• We use the same principle here
Overall objective

- Find most likely symbol sequence given inputs
  \[ S_0 \ldots S_{K-1} = \arg\max_{S_0' \ldots S_{K-1}'} \text{prob}(S_0' \ldots S_{K-1}' | X_0 \ldots X_{N-1}) \]
Finding the best output

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

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$y_0^{AH}$</th>
<th>$y_1^{AH}$</th>
<th>$y_2^{AH}$</th>
<th>$y_3^{AH}$</th>
<th>$y_4^{AH}$</th>
<th>$y_5^{AH}$</th>
<th>$y_6^{AH}$</th>
<th>$y_7^{AH}$</th>
<th>$y_8^{AH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$y_0^{B}$</td>
<td>$y_1^{B}$</td>
<td>$y_2^{B}$</td>
<td>$y_3^{B}$</td>
<td>$y_4^{B}$</td>
<td>$y_5^{B}$</td>
<td>$y_6^{B}$</td>
<td>$y_7^{B}$</td>
<td>$y_8^{B}$</td>
</tr>
<tr>
<td>/D/</td>
<td>$y_0^{D}$</td>
<td>$y_1^{D}$</td>
<td>$y_2^{D}$</td>
<td>$y_3^{D}$</td>
<td>$y_4^{D}$</td>
<td>$y_5^{D}$</td>
<td>$y_6^{D}$</td>
<td>$y_7^{D}$</td>
<td>$y_8^{D}$</td>
</tr>
<tr>
<td>/EH/</td>
<td>$y_0^{EH}$</td>
<td>$y_1^{EH}$</td>
<td>$y_2^{EH}$</td>
<td>$y_3^{EH}$</td>
<td>$y_4^{EH}$</td>
<td>$y_5^{EH}$</td>
<td>$y_6^{EH}$</td>
<td>$y_7^{EH}$</td>
<td>$y_8^{EH}$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^{F}$</td>
<td>$y_1^{F}$</td>
<td>$y_2^{F}$</td>
<td>$y_3^{F}$</td>
<td>$y_4^{F}$</td>
<td>$y_5^{F}$</td>
<td>$y_6^{F}$</td>
<td>$y_7^{F}$</td>
<td>$y_8^{F}$</td>
</tr>
<tr>
<td>/G/</td>
<td>$y_0^{G}$</td>
<td>$y_1^{G}$</td>
<td>$y_2^{G}$</td>
<td>$y_3^{G}$</td>
<td>$y_4^{G}$</td>
<td>$y_5^{G}$</td>
<td>$y_6^{G}$</td>
<td>$y_7^{G}$</td>
<td>$y_8^{G}$</td>
</tr>
</tbody>
</table>

- 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
Simple pseudocode

- Assuming $y(t, i), t = 1 ... T, i = 1 ... N$ is already computed using the underlying RNN

```plaintext
n = 1
best(1) = argmax_i (y(1,i))
for t = 1:T
    best(t) = argmax_i (y(t,i))
    if (best(t) != best(t-1))
        out(n) = best(t-1)
        time(n) = t-1
    n = n+1
```
Finding the best output

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$y_0^{AH}$</th>
<th>$y_1^{AH}$</th>
<th>$y_2^{AH}$</th>
<th>$y_3^{AH}$</th>
<th>$y_4^{AH}$</th>
<th>$y_5^{AH}$</th>
<th>$y_6^{AH}$</th>
<th>$y_7^{AH}$</th>
<th>$y_8^{AH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$y_0^B$</td>
<td>$y_1^B$</td>
<td>$y_2^B$</td>
<td>$y_3^B$</td>
<td>$y_4^B$</td>
<td>$y_5^B$</td>
<td>$y_6^B$</td>
<td>$y_7^B$</td>
<td>$y_8^B$</td>
</tr>
<tr>
<td>/D/</td>
<td>$y_0^D$</td>
<td>$y_1^D$</td>
<td>$y_2^D$</td>
<td>$y_3^D$</td>
<td>$y_4^D$</td>
<td>$y_5^D$</td>
<td>$y_6^D$</td>
<td>$y_7^D$</td>
<td>$y_8^D$</td>
</tr>
<tr>
<td>/EH/</td>
<td>Cannot distinguish between an extended symbol and repetitions of the symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
<td>/IY/</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
<tr>
<td>/G/</td>
<td>$y_0^G$</td>
<td>$y_1^G$</td>
<td>$y_2^G$</td>
<td>$y_3^G$</td>
<td>$y_4^G$</td>
<td>$y_5^G$</td>
<td>$y_6^G$</td>
<td>$y_7^G$</td>
<td>$y_8^G$</td>
</tr>
</tbody>
</table>

- 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
Finding the best output

- 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

*Resulting sequence may be meaningless (what word is “GFIYD”?)*

*Cannot distinguish between an extended symbol and repetitions of the symbol*
Finding the best output

- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g.* Sub-symbol units (like in HW1 – what were they?)
  - *E.g.* using special “separating” symbols to separate repetitions
### Finding the best output

- **Option 2:** Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g.* *Sub-symbol* units (like in HW1 – what were they?)
  - *E.g.* using special “separating” symbols to separate repetitions

We will refer to the process of obtaining an output from the network as **decoding**

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$y_0^1$</th>
<th>$y_1^1$</th>
<th>$y_2^1$</th>
<th>$y_3^1$</th>
<th>$y_4^1$</th>
<th>$y_5^1$</th>
<th>$y_6^1$</th>
<th>$y_7^1$</th>
<th>$y_8^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$y_0^2$</td>
<td>$y_1^2$</td>
<td>$y_2^2$</td>
<td>$y_3^2$</td>
<td>$y_4^2$</td>
<td>$y_5^2$</td>
<td>$y_6^2$</td>
<td>$y_7^2$</td>
<td>$y_8^2$</td>
</tr>
<tr>
<td>/D/</td>
<td>$y_0^3$</td>
<td>$y_1^3$</td>
<td>$y_2^3$</td>
<td>$y_3^3$</td>
<td>$y_4^3$</td>
<td>$y_5^3$</td>
<td>$y_6^3$</td>
<td>$y_7^3$</td>
<td>$y_8^3$</td>
</tr>
<tr>
<td>/EH/</td>
<td>$y_0^4$</td>
<td>$y_1^4$</td>
<td>$y_2^4$</td>
<td>$y_3^4$</td>
<td>$y_4^4$</td>
<td>$y_5^4$</td>
<td>$y_6^4$</td>
<td>$y_7^4$</td>
<td>$y_8^4$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^5$</td>
<td>$y_1^5$</td>
<td>$y_2^5$</td>
<td>$y_3^5$</td>
<td>$y_4^5$</td>
<td>$y_5^5$</td>
<td>$y_6^5$</td>
<td>$y_7^5$</td>
<td>$y_8^5$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^6$</td>
<td>$y_1^6$</td>
<td>$y_2^6$</td>
<td>$y_3^6$</td>
<td>$y_4^6$</td>
<td>$y_5^6$</td>
<td>$y_6^6$</td>
<td>$y_7^6$</td>
<td>$y_8^6$</td>
</tr>
<tr>
<td>/G/</td>
<td>$y_0^7$</td>
<td>$y_1^7$</td>
<td>$y_2^7$</td>
<td>$y_3^7$</td>
<td>$y_4^7$</td>
<td>$y_5^7$</td>
<td>$y_6^7$</td>
<td>$y_7^7$</td>
<td>$y_8^7$</td>
</tr>
</tbody>
</table>

---

We will refer to the process of obtaining an output from the network as **decoding**.
Decoding

This is in fact a *suboptimal* decode that actually finds the most likely *time-synchronous* output sequence

- Which is not necessarily the most likely *order-synchronous* sequence
  - The “merging” heuristics do not guarantee optimal order-synchronous sequences
- We will return to this topic later
The sequence-to-sequence problem

• How do we know when to output symbols
  – In fact, the network produces outputs at every time
  – Which of these are the real outputs

• How do we train these models?
Training data: input sequence + output sequence
- Output sequence length <= input sequence length

Given output symbols at the right locations
- The phoneme /B/ ends at $X_2$, /AH/ at $X_6$, /T/ at $X_9$
The “alignment” of labels

- The time-stamps of the output symbols give us the “alignment” of the output sequence to the input sequence
  - Which portion of the input aligns to what symbol

- Simply knowing the output sequence does not provide us the alignment
  - This is extra information
Training with alignment

• Training data: input sequence + output sequence
  – Output sequence length <= input sequence length

• Given the alignment of the output to the input
  – The phoneme /B/ ends at $X_2$, /AH/ at $X_6$, /T/ at $X_9$
• Either just define Divergence as:

\[ DIV = KL(Y_2, B) + KL(Y_6, AH) + KL(Y_9, T) \]

• Or..
• Either just define Divergence as:
  \[ \text{DIV} = KL(Y_2, B) + KL(Y_6, AH) + KL(Y_9, T) \]
• Or repeat the symbols over their duration
  \[ \text{DIV} = \sum_t KL(Y_t, \text{symbol}_t) = -\sum_t \log Y(t, \text{symbol}_t) \]
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. \( Y_t \)
Training *without* alignment

- We know how to train if the alignment is provided
- Problem: Alignment is *not* provided

- Solution:
  1. *Guess* the alignment
  2. Consider *all possible* alignments
Solution 1: Guess the alignment

- Guess an initial alignment and iteratively refine it as the model improves
- 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
Solution 1: Guess the alignment

- Guess an initial alignment and iteratively refine it as the model improves
- 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
Characterizing the alignment

- An alignment can be represented as a repetition of symbols
  - Examples show different alignments of /B/ /AH/ /T/ to $X_0 \ldots X_9$
Estimating an alignment

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

• Find:
  – An $N$-length expansion $s_0 \ldots s_{N-1}$ comprising the symbols in $S$ in strict order
    • e.g. $S_0 S_1 S_1 S_2 \ldots S_{K-1}$
      – i.e. $s_0 = S_0, s_1 = S_0, s_2 = S_1, s_3 = S_1, s_4 = S_1, \ldots s_{N-1} = S_{K-1}$
    • E.g. /B/ /B/ /IY/ /IY/ /IY/ /F/ /F/ /F/ /IY/ ..

• Outcome: an alignment of the target symbol sequence $S_0 \ldots S_{K-1}$ to the input $X_0 \ldots X_{N-1}$
Estimating an alignment

• Alignment problem:

• Find

\[
\arg\max P(s_0, s_1, \ldots, s_{N-1} | S_0, S_1, \ldots, S_K, X_0, X_1, \ldots, X_{N-1})
\]
  – Such that

\[
\text{compress}(s_0, s_1, \ldots, s_{N-1}) \equiv S_0, S_1, \ldots, S_K
\]

• \text{compress()} is the operation of compressing repetitions into one
### Recall: The actual output of the network

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$y_0^{AH}$</th>
<th>$y_1^{AH}$</th>
<th>$y_2^{AH}$</th>
<th>$y_3^{AH}$</th>
<th>$y_4^{AH}$</th>
<th>$y_5^{AH}$</th>
<th>$y_6^{AH}$</th>
<th>$y_7^{AH}$</th>
<th>$y_8^{AH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$y_0^B$</td>
<td>$y_1^B$</td>
<td>$y_2^B$</td>
<td>$y_3^B$</td>
<td>$y_4^B$</td>
<td>$y_5^B$</td>
<td>$y_6^B$</td>
<td>$y_7^B$</td>
<td>$y_8^B$</td>
</tr>
<tr>
<td>/D/</td>
<td>$y_0^D$</td>
<td>$y_1^D$</td>
<td>$y_2^D$</td>
<td>$y_3^D$</td>
<td>$y_4^D$</td>
<td>$y_5^D$</td>
<td>$y_6^D$</td>
<td>$y_7^D$</td>
<td>$y_8^D$</td>
</tr>
<tr>
<td>/EH/</td>
<td>$y_0^{EH}$</td>
<td>$y_1^{EH}$</td>
<td>$y_2^{EH}$</td>
<td>$y_3^{EH}$</td>
<td>$y_4^{EH}$</td>
<td>$y_5^{EH}$</td>
<td>$y_6^{EH}$</td>
<td>$y_7^{EH}$</td>
<td>$y_8^{EH}$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
<tr>
<td>/G/</td>
<td>$y_0^G$</td>
<td>$y_1^G$</td>
<td>$y_2^G$</td>
<td>$y_3^G$</td>
<td>$y_4^G$</td>
<td>$y_5^G$</td>
<td>$y_6^G$</td>
<td>$y_7^G$</td>
<td>$y_8^G$</td>
</tr>
</tbody>
</table>

- At each time the network outputs a probability for *each* output symbol
Recall: unconstrained decoding

<table>
<thead>
<tr>
<th>/AH/</th>
<th>y_0^{AH}</th>
<th>y_1^{AH}</th>
<th>y_2^{AH}</th>
<th>y_3^{AH}</th>
<th>y_4^{AH}</th>
<th>y_5^{AH}</th>
<th>y_6^{AH}</th>
<th>y_7^{AH}</th>
<th>y_8^{AH}</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>y_0^{B}</td>
<td>y_1^{B}</td>
<td>y_2^{B}</td>
<td>y_3^{B}</td>
<td>y_4^{B}</td>
<td>y_5^{B}</td>
<td>y_6^{B}</td>
<td>y_7^{B}</td>
<td>y_8^{B}</td>
</tr>
<tr>
<td>/D/</td>
<td>y_0^{D}</td>
<td>y_1^{D}</td>
<td>y_2^{D}</td>
<td>y_3^{D}</td>
<td>y_4^{D}</td>
<td>y_5^{D}</td>
<td>y_6^{D}</td>
<td>y_7^{D}</td>
<td>y_8^{D}</td>
</tr>
<tr>
<td>/EH/</td>
<td>y_0^{EH}</td>
<td>y_1^{EH}</td>
<td>y_2^{EH}</td>
<td>y_3^{EH}</td>
<td>y_4^{EH}</td>
<td>y_5^{EH}</td>
<td>y_6^{EH}</td>
<td>y_7^{EH}</td>
<td>y_8^{EH}</td>
</tr>
<tr>
<td>/IY/</td>
<td>y_0^{IY}</td>
<td>y_1^{IY}</td>
<td>y_2^{IY}</td>
<td>y_3^{IY}</td>
<td>y_4^{IY}</td>
<td>y_5^{IY}</td>
<td>y_6^{IY}</td>
<td>y_7^{IY}</td>
<td>y_8^{IY}</td>
</tr>
<tr>
<td>/F/</td>
<td>y_0^{F}</td>
<td>y_1^{F}</td>
<td>y_2^{F}</td>
<td>y_3^{F}</td>
<td>y_4^{F}</td>
<td>y_5^{F}</td>
<td>y_6^{F}</td>
<td>y_7^{F}</td>
<td>y_8^{F}</td>
</tr>
<tr>
<td>/G/</td>
<td>y_0^{G}</td>
<td>y_1^{G}</td>
<td>y_2^{G}</td>
<td>y_3^{G}</td>
<td>y_4^{G}</td>
<td>y_5^{G}</td>
<td>y_6^{G}</td>
<td>y_7^{G}</td>
<td>y_8^{G}</td>
</tr>
</tbody>
</table>

- We find the most likely sequence of symbols
  - (Conditioned on input $X_0 \ldots X_{N-1}$)

- This may not correspond to an expansion of the desired symbol sequence
  - E.g. the unconstrained decode may be
    /AH///AH///AH///D///D///AH///F///IY///IY/  
    - Contracts to /AH/ /D/ /AH/ /F/ /IY/  
  - Whereas we want an expansion of /B///IY///F///IY/
**Constraining the alignment: Try 1**

<table>
<thead>
<tr>
<th>/B/</th>
<th>$y_0^B$</th>
<th>$y_1^B$</th>
<th>$y_2^B$</th>
<th>$y_3^B$</th>
<th>$y_4^B$</th>
<th>$y_5^B$</th>
<th>$y_6^B$</th>
<th>$y_7^B$</th>
<th>$y_8^B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
</tbody>
</table>

- Block out all rows that do not include symbols from the target sequence
  - E.g. Block out rows that are not /B/ /IY/ or /F/

51
Blocking out unnecessary outputs

Compute the entire output (for all symbols)
Copy the output values for the target symbols into the secondary reduced structure
Constraining the alignment: Try 1

- Only decode on reduced grid
  - We are now assured that only the appropriate symbols will be hypothesized
Constraining the alignment: Try 1

- **Only decode on reduced grid**
  - We are now assured that only the appropriate symbols will be hypothesized

- **Problem:** This still doesn’t assure that the decode sequence correctly expands the target symbol sequence
  - E.g. the above decode is not an expansion of /B//IY//F//IY/

- **Still needs additional constraints**
Try 2: Explicitly arrange the constructed table

<table>
<thead>
<tr>
<th>/B/</th>
<th>$y_0^B$</th>
<th>$y_1^B$</th>
<th>$y_2^B$</th>
<th>$y_3^B$</th>
<th>$y_4^B$</th>
<th>$y_5^B$</th>
<th>$y_6^B$</th>
<th>$y_7^B$</th>
<th>$y_8^B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$y_0^{AH}$</th>
<th>$y_1^{AH}$</th>
<th>$y_2^{AH}$</th>
<th>$y_3^{AH}$</th>
<th>$y_4^{AH}$</th>
<th>$y_5^{AH}$</th>
<th>$y_6^{AH}$</th>
<th>$y_7^{AH}$</th>
<th>$y_8^{AH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$y_0^B$</td>
<td>$y_1^B$</td>
<td>$y_2^B$</td>
<td>$y_3^B$</td>
<td>$y_4^B$</td>
<td>$y_5^B$</td>
<td>$y_6^B$</td>
<td>$y_7^B$</td>
<td>$y_8^B$</td>
</tr>
<tr>
<td>/D/</td>
<td>$y_0^D$</td>
<td>$y_1^D$</td>
<td>$y_2^D$</td>
<td>$y_3^D$</td>
<td>$y_4^D$</td>
<td>$y_5^D$</td>
<td>$y_6^D$</td>
<td>$y_7^D$</td>
<td>$y_8^D$</td>
</tr>
<tr>
<td>/EH/</td>
<td>$y_0^{EH}$</td>
<td>$y_1^{EH}$</td>
<td>$y_2^{EH}$</td>
<td>$y_3^{EH}$</td>
<td>$y_4^{EH}$</td>
<td>$y_5^{EH}$</td>
<td>$y_6^{EH}$</td>
<td>$y_7^{EH}$</td>
<td>$y_8^{EH}$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
<tr>
<td>/G/</td>
<td>$y_0^G$</td>
<td>$y_1^G$</td>
<td>$y_2^G$</td>
<td>$y_3^G$</td>
<td>$y_4^G$</td>
<td>$y_5^G$</td>
<td>$y_6^G$</td>
<td>$y_7^G$</td>
<td>$y_8^G$</td>
</tr>
</tbody>
</table>

Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required.
Try 2: Explicitly arrange the constructed table

<table>
<thead>
<tr>
<th>/B/</th>
<th>$y_0^B$</th>
<th>$y_1^B$</th>
<th>$y_2^B$</th>
<th>$y_3^B$</th>
<th>$y_4^B$</th>
<th>$y_5^B$</th>
<th>$y_6^B$</th>
<th>$y_7^B$</th>
<th>$y_8^B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
<tr>
<td>/F/</td>
<td>$y_0^F$</td>
<td>$y_1^F$</td>
<td>$y_2^F$</td>
<td>$y_3^F$</td>
<td>$y_4^F$</td>
<td>$y_5^F$</td>
<td>$y_6^F$</td>
<td>$y_7^F$</td>
<td>$y_8^F$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$y_0^{IY}$</td>
<td>$y_1^{IY}$</td>
<td>$y_2^{IY}$</td>
<td>$y_3^{IY}$</td>
<td>$y_4^{IY}$</td>
<td>$y_5^{IY}$</td>
<td>$y_6^{IY}$</td>
<td>$y_7^{IY}$</td>
<td>$y_8^{IY}$</td>
</tr>
</tbody>
</table>

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 2\textsuperscript{nd} and 4\textsuperscript{th} positions.

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

#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input

#First create output table
For i = 1:N
\[ s(1:T,i) = y(1:T, S(i)) \]

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation
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. symbol chosen at any time is at the same level or at the next level to the symbol at the previous time
- This guarantees that the sequence *is* an expansion of the target sequence
  - /B/ /IY/ /F/ /IY/ in this case
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//IY//F//IY/)

- Edge scores are 1
- Node scores are the probabilities assigned to the symbols by the neural network
• 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//IY//F//IY/)
• 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
• E.g. the probability of the marked path is

\[
Scr(Path) = y_0^B y_1^B y_2^{IY} y_3^{IY} y_4^F
\]
Path Score (probability)

- 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//IY//F//IY/)
- 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

Figure shows a typical end-to-end path. There are an exponential number of such paths. Challenge: Find the path with the highest score (probability)
Explicitly constrain alignment

- Find the **most probable path** from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm
Viterbi algorithm: Basic idea

- The best path to any node must be an extension of the best path to one of its parent nodes
  - Any other path would necessarily have a lower probability
- The best parent is simply the parent with the best-scoring best path
Viterbi algorithm: Basic idea

\[ \text{BestPath}(y_0^B \rightarrow y_3^F) = \text{BestPath}(y_0^B \rightarrow y_2^{\text{IY}})y_3^F \]
\[ \text{or} \quad \text{BestPath}(y_0^B \rightarrow y_2^F)y_3^F \]
\[ \text{BestPath}(y_0^B \rightarrow y_3^F) = \text{BestPath}(y_0^B \rightarrow \text{BestParent})y_3^F \]

• The best parent is simply the parent with the best-scoring best path
\[ \text{BestParent} \]
\[ = \arg\max_{\text{Parent} \in (y_2^\text{IY}, y_2^F)} \left( \text{Score}(\text{BestPath}(y_0^B \rightarrow \text{Parent})) \right) \]
Viterbi algorithm

- Dynamically track the best path (and the score of the best path) from the source node to every node in the graph
  - At each node, keep track of
    - The best incoming parent edge
    - The score of the best path from the source to the node through this best parent edge
- Eventually compute the best path from source to sink
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 $X_0 \ldots X_t$)
  - E.g., $S(0) = /B/$
    - The scores in the 0-th row have the form $y_t^B$
  - E.g. $S(1) = S(3) = /IY/$
    - The scores in the 1-st and 3-rd rows have the form $y_t^{IY}$
  - E.g. $S(2) = /F/$
    - The scores in the 2-nd row have the form $y_t^F$
Viterbi algorithm

- Initialization:

\[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
\[ Bscr(0,0) = y_0^{S(0)}, \ Bscr(0, i) = 0 \ \text{for} \ i = 1 \ldots K - 1 \]
Viterbi algorithm

- Initialization:

\[ BP(0, i) = \text{null}, \quad i = 0 \ldots K - 1 \]

\[ Bscr(0,0) = y_0^{S(0)}, \quad Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]

- for \( t = 1 \ldots T - 1 \)

  for \( l = 0 \ldots K - 1 \)

  - \( BP(t, l) = \arg\max_{p \in \text{parents}(l)} Bscr(t - 1, p) \)

  - \( Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \)
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
  \[ Bscr(0,0) = y_0^{S(0)}, \ Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \ Bscr(t, 0) = Bscr(t - 1,0) \times y_t^{S(0)} \]
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
  \[ B\text{scr}(0,0) = \gamma_0^{S(0)}, \ B\text{scr}(0, i) = 0 \ for \ i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \ B\text{scr}(t, 0) = B\text{scr}(t - 1, 0) \times \gamma_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
  - \[ BP(t, l) = \left( l - 1: \ if \ (B\text{scr}(t - 1, l - 1) > B\text{scr}(t - 1, l)) \right) \]
  - \[ B\text{scr}(t, l) = B\text{scr}(BP(t, l)) \times \gamma_t^{S(l)} \]
Viterbi algorithm

- Initialization:
  
  \[ BP(0, i) = \text{null}, \quad i = 0 \ldots K - 1 \]
  
  \[ Bscr(0,0) = y_0^{S(0)}, \quad Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]

- for \( t = 1 \ldots T - 1 \)

  \[ BP(t, 0) = 0; \quad Bscr(t, 0) = Bscr(t - 1,0) \times y_t^{S(0)} \]

  for \( l = 1 \ldots K - 1 \)

  - \[ BP(t, l) = \begin{cases} 
  l - 1 : & \text{if } (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \quad l - 1; \\
  l : & \text{else}
  \end{cases} \]

  - \[ Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \]
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
  \[ B_{scr}(0, 0) = y_0^{S(0)}, \ B_{scr}(0, i) = 0 \ for \ i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; B_{scr}(t, 0) = B_{scr}(t - 1, 0) \times y_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
    - \( BP(t, l) = \begin{cases} \ l - 1 : & \text{if } (B_{scr}(t - 1, l - 1) > B_{scr}(t - 1, l)) \ l - 1; \\ \ l : & \text{else} \end{cases} \)
    - \( B_{scr}(t, l) = B_{scr}(BP(t, l)) \times y_t^{S(l)} \)
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \quad i = 0 \ldots K - 1 \]
  \[ Bscr(0, 0) = y_0^{S(0)}, \quad Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \quad Bscr(t, 0) = Bscr(t - 1, 0) \times y_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
  - \( BP(t, l) = (\text{if } (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \text{ \(l - 1\);} \text{ else } l) \)
  - \( Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \)
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
  \[ Bscr(0,0) = y_0^{S(0)}, \ Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \ Bscr(t, 0) = Bscr(t - 1,0) \times y_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
    - \( BP(t, l) = \begin{cases} \text{if} \ (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \ l - 1; \ \text{else} \ l \end{cases} \)
  - \( Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)} \)
Viterbi algorithm

• Initialization:
  \[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]
  \[ Bscr(0,0) = y_0^{S(0)}, \ Bscr(0, i) = 0 \text{ for } i = 1 \ldots K - 1 \]
• for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \ Bscr(t, 0) = Bscr(t - 1,0) \times y_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
    • \( BP(t, l) = \begin{cases} 
    l - 1 : & \text{if } (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \\
    l : & \text{else} 
  \end{cases} \)
    • \( Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \)
Viterbi algorithm

- Initialization:
  \[ BP(0, i) = \text{null}, \quad i = 0 \ldots K - 1 \]
  \[ Bscr(0, 0) = y_0^{S(0)}, \quad Bscr(0, i) = 0 \quad \text{for} \quad i = 1 \ldots K - 1 \]
- for \( t = 1 \ldots T - 1 \)
  \[ BP(t, 0) = 0; \quad Bscr(t, 0) = Bscr(t - 1, 0) \times y_t^{S(0)} \]
  for \( l = 1 \ldots K - 1 \)
    - \( BP(t, l) = \begin{cases} l - 1 : & (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \quad l - 1; \\ l : & \text{else} \end{cases} \)
    - \( Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \)
• Initialization:

\[ BP(0, i) = \text{null}, \ i = 0 \ldots K - 1 \]

\[ Bscr(0,0) = y_0^{S(0)}, \ Bscr(0, i) = 0 \ for \ i = 1 \ldots K - 1 \]

• for \( t = 1 \ldots T - 1 \)

\[ BP(t, 0) = 0; \ Bscr(t, 0) = Bscr(t - 1, 0) \times y_t^{S(0)} \]

\[ \text{for } l = 1 \ldots K - 1 \]

- \[ BP(t, l) = \left( \begin{array}{l}
  l - 1: \ \text{if} \ (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \ l - 1; \\
  l: \ \text{else}
\end{array} \right) \]

- \[ Bscr(t, l) = Bscr(BP(t, l)) \times y_t^{S(l)} \]
Viterbi algorithm

- $s(T - 1) = S(K - 1)$
Viterbi algorithm

- $s(T - 1) = S(K - 1)$
- for $t = T - 1$ downto 1
  
  $s(t - 1) = BP(s(t))$
Viterbi algorithm

- \( s(T - 1) = S(K - 1) \)
- for \( t = T - 1 \) downto 1
  \( s(t - 1) = BP(s(t)) \)
VITERBI

#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input

#First create output table
For i = 1:N
    s(1:T,i) = y(1:T, S(i))

#Now run the Viterbi algorithm
# First, at t = 1
BP(1,1) = -1
Bscr(1,1) = s(1,1)
Bscr(1,2:N) = 0
for t = 2:T
    BP(t,1) = 1;
    Bscr(t,1) = Bscr(t-1,1)*s(t,1)
    for i = 1:min(t,N)
        BP(t,i) = Bscr(t-1,i) > Bscr(t-1,i-1) ? i : i-1
        Bscr(t,i) = Bscr(t-1,BP(t,i))*s(t,i)

# Backtrace
AlignedSymbol(T) = N
for t = T downto 2
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation
VITERBI

#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input

#First create output table
For i = 1:N
  s(1:T,i) = y(1:T, S(i))

#Now run the Viterbi algorithm
# First, at t = 1
BP(1,1) = -1
Bscr(1,1) = s(1,1)
Bscr(1,2:N) = 0
for t = 2:T
  BP(t,1) = 1;
  Bscr(t,1) = Bscr(t-1,1)*s(t,1)
  for i = 2:min(t,N)
    BP(t,i) = Bscr(t-1,i) > Bscr(t-1,i-1) ? i : i-1
    Bscr(t,i) = Bscr(t-1,BP(t,i))*s(t,i)

# Backtrace
AlignedSymbol(T) = N
for t = T downto 2
  AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation
VITERBI

#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input

# First, at t = 1
BP(1,1) = -1
Bscr(1,1) = y(1,S(1))
Bscr(1,2:N) = 0
for t = 2:T
    BP(t,1) = 1;
    Bscr(t,1) = Bscr(t-1,1)*y(t,S(1))
    for i = 2:min(t,N)
        BP(t,i) = Bscr(t-1,i) > Bscr(t-1,i-1) ? i : i-1
        Bscr(t,i) = Bscr(t-1,BP(t,i))*y(t,S(i))

# Backtrace
AlignedSymbol(T) = N
for t = T downto 2
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation
Assumed targets for training with the Viterbi algorithm
**Gradients from the alignment**

\[
\begin{array}{cccccccc}
/B/ & y^B_0 & y^B_1 & y^B_2 & y^B_3 & y^B_4 & y^B_5 & y^B_6 & y^B_7 & y^B_8 \\
/IY/ & y^IY_0 & y^IY_1 & y^IY_2 & y^IY_3 & y^IY_4 & y^IY_5 & y^IY_6 & y^IY_7 & y^IY_8 \\
/F/ & y^F_0 & y^IY_1 & y^IY_2 & y^IY_3 & y^IY_4 & y^IY_5 & y^IY_6 & y^IY_7 & y^IY_8 \\
/IY/ & y^IY_0 & y^IY_1 & y^IY_2 & y^IY_3 & y^IY_4 & y^IY_5 & y^IY_6 & y^IY_7 & y^IY_8 \\
\end{array}
\]

\[DIV = \sum_t KL(Y_t, symbol_t^{bestpath}) = -\sum_t \log Y(t, symbol_t^{bestpath})\]

- The gradient w.r.t the \( t \)-th output vector \( Y_t \)

\[
\nabla_{Y_t} DIV = \begin{bmatrix} 0 & 0 & \cdots & -1 \left[ \frac{1}{Y(t, symbol_t^{bestpath})} \right] & \cdots & 0 \end{bmatrix}
\]

- Zeros except at the component corresponding to the target in the estimated alignment
Iterative Estimate and Training

The “decode” and “train” steps may be combined into a single “decode, find alignment compute derivatives” step for SGD and mini-batch updates.
Iterative update

• Option 1:
  – Determine alignments for every training instance
  – Train model (using SGD or your favorite approach) on the entire training set
  – Iterate

• Option 2:
  – During SGD, for each training instance, find the alignment during the forward pass
  – Use in backward pass
Iterative update: Problem

• Approach heavily dependent on initial alignment

• Prone to poor local optima

• Alternate solution: Do not commit to an alignment during any pass.
Next Class

• Training without explicit alignment..
  – Connectionist Temporal Classification
  – Separating repeated symbols

• The CTC decoder..