Deep Learning

Recurrent Networks: Part 3
Fall 2020
Story so far

- **Iterated structures** are good for analyzing time series data with short-time dependence on the past
  - These are “Time delay” neural nets, AKA *convnets*
• Iterated structures are good for analyzing time series data with short-time dependence on the past
  – These are “Time delay” neural nets, AKA convnets

• **Recurrent structures** are good for analyzing time series data with *long-term* dependence on the past
  – These are *recurrent* neural networks
Recap: Recurrent networks can be incredibly effective at modeling long-term dependencies

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT)
        /*
         * The kernel blank will coedl it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SE(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++)
        {
            seq = buf[i++];
            bpf = bd->bd.next + i * search;
            if (fd)
                current = blocked;
        }
    rw->name = "Getjbbregs";
    bprm_self_clear1(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac) | PFMR_CLOBATHINC_SECOND]
    return segtable;
}
```
Recurrent structures can do what static structures cannot

- The addition problem: Add two N-bit numbers to produce a N+1-bit number
  - Input is binary
  - Will require large number of training instances
    - Output must be specified for every pair of inputs
    - Weights that generalize will make errors
  - Network trained for N-bit numbers will not work for N+1 bit numbers

- An RNN learns to do this very quickly
  - With very little training data!
• 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
• 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
Story so far: stability

- Recurrent networks can be unstable
  - And not very good at remembering at other times
Recap: Vanishing gradient examples..

• Learning is difficult: gradients tend to vanish..
The long-term dependency problem

PATTERN1 [...............................] PATTERN 2

Jane had a quick lunch in the bistro. Then she..

• Long-term dependencies are hard to learn in a network where memory behavior is an untriggered function of the network
  – Need it to be a triggered response to input
The LSTM addresses the problem of input-dependent memory behavior.
Recap: LSTM-based architecture

- LSTM based architectures are identical to RNN-based architectures
Recap: Bidirectional LSTM

• Bidirectional version..
Key Issue

- How do we define the divergence
- Also: how do we compute the outputs...

Primary topic for today
What follows in this series on recurrent nets

• Architectures: How to train recurrent networks of different architectures

• Synchrony: How to train recurrent networks when
  – The target output is time-synchronous with the input
  – The target output is order-synchronous, but not time synchronous
  – Applies to only some types of nets

• How to make predictions/inference with such networks
Variants of recurrent nets

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

Images from Karpathy
Variants of recurrent nets

- Sequence classification: Classifying a full input sequence
  - E.g. isolated word/phrase recognition
- Order synchronous, time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori
More variants

• A posteriori sequence to sequence: Generate output sequence after processing input
  – E.g. language translation
• Single-input a posteriori sequence generation
  – E.g. captioning an image

Images from Karpathy
Variants of recurrent nets

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

Images from Karpathy
Regular MLP for processing sequences

- No recurrence in model
  - Exactly as many outputs as inputs
  - Every input produces a unique output
  - The output at time $t$ is unrelated to the output at $t' \neq t$
Learning in a Regular MLP

- **No recurrence**
  - Exactly as many outputs as inputs
    - **One to one correspondence between desired output and actual output**
  - The output at time $t$ is unrelated to the output at $t' \neq t$. 
Regular MLP

- Gradient backpropagated at each time
  \[ \nabla_{Y(t)} Div(Y_{target}(1 ... T), Y(1 ... T)) \]

- Common assumption:
  \[
  Div(Y_{target}(1 ... T), Y(1 ... T)) = \sum_{t} w_t Div(Y_{target}(t), Y(t)) \\
  \nabla_{Y(t)} Div(Y_{target}(1 ... T), Y(1 ... T)) = w_t \nabla_{Y(t)} Div(Y_{target}(t), Y(t))
  \]

  - \( w_t \) is typically set to 1.0
  - This is further backpropagated to update weights etc
Regular MLP

- Gradient backpropagated at each time
  \[ \nabla_{Y(t)} Div(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) \]

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

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

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

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

With a brief detour into modelling language

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
Time-synchronous networks: Inference

- One sided network: Process input left to right and produce output after each input
Time-synchronous networks: 
Inference

- For bidirectional networks:
  - Process input left to right using forward net
  - Process it right to left using backward net
  - The combined outputs are time-synchronous, one per input time, and are passed up to the next layer

- Rest of the lecture(s) will not specifically consider bidirectional nets, but the discussion generalizes
How do we *train* the network

- Back propagation through time (BPTT)

- Given a collection of *sequence* training instances comprising input sequences and output sequences of equal length, with one-to-one correspondence
  - \((X_i, D_i)\), where
  - \(X_i = X_{i,0}, ..., X_{i,T}\)
  - \(D_i = D_{i,0}, ..., D_{i,T}\)
Training: Forward pass

- For each training input:
- Forward pass: pass the entire data sequence through the network, generate outputs
Training: Computing gradients

- For each training input:
- Backward pass: Compute divergence gradients via backpropagation
  - Back Propagation Through Time
• The divergence computed is between the sequence of outputs by the network and the desired sequence of outputs

• This is not just the sum of the divergences at individual times
  ▪ Unless we explicitly define it that way
Back Propagation Through Time

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

The rest of backprop continues from there
First step of backprop: Compute $\nabla_{Y(t)} DIV$ for all $t$

$\nabla_{Z^{(1)}(t)} DIV = \nabla_{Y(t)} DIV \nabla_{Z(t)} Y(t)$

And so on!
First step of backprop: Compute $\nabla_{Y(t)} DIV$ for all $t$

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

• Usual assumption: \textit{Sequence divergence is the sum of the divergence at individual instants}

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

\[
\nabla_{Y(t)} \text{Div}(Y_{\text{target}}(1 \ldots T), Y(1 \ldots T)) = \nabla_{Y(t)} \text{Div}(Y_{\text{target}}(t), Y(t))
\]
**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: \( \text{Div}(Y_{\text{target}}(t), Y(t)) = KL(Y_{\text{target}}(t), Y(t)) \)
Simple recurrence example: Text Modelling

- Learn a model that can predict the next character given a sequence of characters
  - **LINCOLN?**
  - Or, at a higher level, words
    - **TO BE OR NOT TO ???**

- After observing inputs $w_0 \ldots w_k$ it predicts $w_{k+1}$
Simple recurrence example: Text Modelling

Figure from Andrej Karpathy.

Input: Sequence of characters (presented as one-hot vectors).

Target output after observing “h e l l” is “o”

- Input presented as one-hot vectors – Actually “embeddings” of one-hot vectors
- Output: probability distribution over characters – Must ideally peak at the target character
Training

- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the “vocabulary”
- Output: Probability distribution over symbols
  \[ Y(t, i) = P(V_i|w_0 \ldots w_{t-1}) \]
  - \( V_i \) is the i-th symbol in the vocabulary
- Divergence
  \[ \text{Div}(Y_{target}(1 \ldots T), Y(1 \ldots T)) = \sum_t KL(Y_{target}(t), Y(t)) = -\sum_t \log Y(t, w_{t+1}) \]
Brief detour: Language models

• Modelling language using time-synchronous nets

• More generally language models and embeddings..
Language modelling using RNNs

Four score and seven years ???

ABRAHAM LINCOLN??

• Problem: Given a sequence of words (or characters) predict the next one
Language modelling: Representing words

• Represent words as one-hot vectors
  – Pre-specify a vocabulary of N words in fixed (e.g. lexical) order
    • E.g. [ A AARDVARK AARON ABACK ABACUS... ZZYP]
  – Represent each word by an N-dimensional vector with N-1 zeros and a single 1 (in the position of the word in the ordered list of words)
    • E.g. “AARDVARK” → [0 1 0 0 0 ...]
    • E.g. “AARON” → [0 0 1 0 0 0 ...]

• Characters can be similarly represented
  – English will require about 100 characters, to include both cases, special characters such as commas, hyphens, apostrophes, etc., and the space character
Predicting words

Four score and seven years ???

\[ W_n = f(W_0, \ldots, W_{n-1}) \]

\textbf{Nx1 one-hot vectors}

- Given one-hot representations of \( W_0 \ldots W_{n-1} \), predict \( W_n \)
Predicting words

Four score and seven years ???

\[ W_n = f(W_0, ..., W_{n-1}) \]

- Given one-hot representations of \( W_0...W_{n-1} \), predict \( W_n \)
- **Dimensionality problem**: All inputs \( W_0...W_{n-1} \) are both very high-dimensional and very sparse
The one-hot representation

- The one hot representation uses only $N$ corners of the $2^N$ corners of a unit cube
  - Actual volume of space used = 0
    - $(1, \varepsilon, \delta)$ has no meaning except for $\varepsilon = \delta = 0$
  - Density of points: $O\left(\frac{N}{\sqrt[N]{N}}\right)$
- This is a tremendously inefficient use of dimensions
Why one-hot representation

- The one-hot representation makes no assumptions about the relative importance of words
  - All word vectors are the same length
- It makes no assumptions about the relationships between words
  - The distance between every pair of words is the same
Solution to dimensionality problem

- Project the points onto a lower-dimensional subspace
  - Or more generally, a linear transform into a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $\mathcal{O}(\frac{N}{\sqrt{M}})$
Solution to dimensionality problem

• Project the points onto a lower-dimensional subspace
  – Or more generally, a linear transform into a lower-dimensional subspace
  – The volume used is still 0, but density can go up by many orders of magnitude
    • Density of points: $O\left(\frac{N}{\sqrt{M}}\right)$
  – If properly learned, the distances between projected points will capture semantic relations between the words
The **Projected** word vectors

Four score and seven years ???

\[ W_n = f(PW_0, PW_1, \ldots, PW_{n-1}) \]

- **Project** the N-dimensional one-hot word vectors into a lower-dimensional space
  - Replace every one-hot vector \( W_i \) by \( PW_i \)
  - \( P \) is an \( M \times N \) matrix
  - \( PW_i \) is now an \( M \)-dimensional vector
  - **Learn** \( P \) using an appropriate objective
    - Distances in the projected space will reflect relationships imposed by the objective
“Projection”

\[ W_n = f(PW_1, PW_2, \ldots, PW_{n-1}) \]

- \( P \) is a simple linear transform
- A single transform can be implemented as a layer of \( M \) neurons with linear activation
- The transforms that apply to the individual inputs are all \( M \)-neuron linear-activation subnets with tied weights
Predicting words: The TDNN model

- Predict each word based on the past N words
  - “A neural probabilistic language model”, Bengio et al. 2003
  - Hidden layer has Tanh() activation, output is softmax

- One of the outcomes of learning this model is that we also learn low-dimensional representations $PW$ of words
Alternative models to learn projections

• Soft bag of words: Predict word based on words in immediate context
  – Without considering specific position
• Skip-grams: Predict adjacent words based on current word
• More on these in a future recitation?
Embeddings: Examples

Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

- From Mikolov et al., 2013, “Distributed Representations of Words and Phrases and their Compositionality”
• The hidden units are (one or more layers of) LSTM units
• Trained via backpropagation from a lot of text
  – No explicit labels in the training data: at each time the next word is the label.

Modelling language
Generating Language: Synthesis

- On trained model: Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
• On trained model: Provide the first few words
  – One-hot vectors

• After the last input word, the network generates a probability distribution over words
  – Outputs an N-valued probability distribution rather than a one-hot vector

• Draw a word from the distribution
  – And set it as the next word in the series
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
- Continue this process until we terminate generation
  - In some cases, e.g. generating programs, there may be a natural termination
Which open source project?

Trained on linux source code

Actually uses a character-level model (predicts character sequences)
Composing music with RNN

Returning to our problem

• Divergences are harder to define in other scenarios..
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
Example..

- 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
- $DIV(Y_{target}, Y) = KL(Y(T), Phoneme)$

This divergence must propagate through the net to update all parameters
• 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
• Exploiting the untagged inputs: assume the same output for the entire input
• Define the divergence everywhere

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

Fix: Use these outputs too.

These too must ideally point to the correct phoneme

• Define the divergence everywhere

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

• 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:

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

- 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

<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>

![Diagram](image)

- Find most likely symbol sequence given inputs

$$S_0 \ldots S_{K-1} = \arg\max_{S'_0 \ldots S'_{K-1}} \text{prob}(S'_0 \ldots S'_{K-1} | X_0 \ldots X_{N-1})$$

$$S'_0 \ldots S'_{K-1}$$
Finding the best output

<table>
<thead>
<tr>
<th>/AH/</th>
<th>$\gamma_0^1$</th>
<th>$\gamma_1^1$</th>
<th>$\gamma_2^1$</th>
<th>$\gamma_3^1$</th>
<th>$\gamma_4^1$</th>
<th>$\gamma_5^1$</th>
<th>$\gamma_6^1$</th>
<th>$\gamma_7^1$</th>
<th>$\gamma_8^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B/</td>
<td>$\gamma_0^2$</td>
<td>$\gamma_1^2$</td>
<td>$\gamma_2^2$</td>
<td>$\gamma_3^2$</td>
<td>$\gamma_4^2$</td>
<td>$\gamma_5^2$</td>
<td>$\gamma_6^2$</td>
<td>$\gamma_7^2$</td>
<td>$\gamma_8^2$</td>
</tr>
<tr>
<td>/D/</td>
<td>$\gamma_0^3$</td>
<td>$\gamma_1^3$</td>
<td>$\gamma_2^3$</td>
<td>$\gamma_3^3$</td>
<td>$\gamma_4^3$</td>
<td>$\gamma_5^3$</td>
<td>$\gamma_6^3$</td>
<td>$\gamma_7^3$</td>
<td>$\gamma_8^3$</td>
</tr>
<tr>
<td>/EH/</td>
<td>$\gamma_0^4$</td>
<td>$\gamma_1^4$</td>
<td>$\gamma_2^4$</td>
<td>$\gamma_3^4$</td>
<td>$\gamma_4^4$</td>
<td>$\gamma_5^4$</td>
<td>$\gamma_6^4$</td>
<td>$\gamma_7^4$</td>
<td>$\gamma_8^4$</td>
</tr>
<tr>
<td>/IY/</td>
<td>$\gamma_0^5$</td>
<td>$\gamma_1^5$</td>
<td>$\gamma_2^5$</td>
<td>$\gamma_3^5$</td>
<td>$\gamma_4^5$</td>
<td>$\gamma_5^5$</td>
<td>$\gamma_6^5$</td>
<td>$\gamma_7^5$</td>
<td>$\gamma_8^5$</td>
</tr>
<tr>
<td>/F/</td>
<td>$\gamma_0^6$</td>
<td>$\gamma_1^6$</td>
<td>$\gamma_2^6$</td>
<td>$\gamma_3^6$</td>
<td>$\gamma_4^6$</td>
<td>$\gamma_5^6$</td>
<td>$\gamma_6^6$</td>
<td>$\gamma_7^6$</td>
<td>$\gamma_8^6$</td>
</tr>
<tr>
<td>/G/</td>
<td>$\gamma_0^7$</td>
<td>$\gamma_1^7$</td>
<td>$\gamma_2^7$</td>
<td>$\gamma_3^7$</td>
<td>$\gamma_4^7$</td>
<td>$\gamma_5^7$</td>
<td>$\gamma_6^7$</td>
<td>$\gamma_7^7$</td>
<td>$\gamma_8^7$</td>
</tr>
</tbody>
</table>

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

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

\[
\begin{align*}
n &= 1 \\
\text{best}(1) &= \arg\max_i (y(1,i)) \\
\text{for } t = 1:T \\
\quad \text{best}(t) &= \arg\max_i (y(t,i)) \\
\quad \text{if } (\text{best}(t) \neq \text{best}(t-1)) \\
\quad \quad \text{out}(n) &= \text{best}(t-1) \\
\quad \quad \text{time}(n) &= t-1 \\
\quad n &= n+1
\end{align*}
\]
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
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**

| /AH/ | \( y_0^1 \) | \( y_1^1 \) | \( y_2^1 \) | \( y_3^1 \) | \( y_4^1 \) | \( y_5^1 \) | \( y_6^1 \) | \( y_7^1 \) | \( y_8^1 \) |
| /B/  | \( y_0^2 \) | \( y_1^2 \) | \( y_2^2 \) | \( y_3^2 \) | \( y_4^2 \) | \( y_5^2 \) | \( y_6^2 \) | \( y_7^2 \) | \( y_8^2 \) |
| /D/  | \( y_0^3 \) | \( y_1^3 \) | \( y_2^3 \) | \( y_3^3 \) | \( y_4^3 \) | \( y_5^3 \) | \( y_6^3 \) | \( y_7^3 \) | \( y_8^3 \) |
| /EH/ | \( y_0^4 \) | \( y_1^4 \) | \( y_2^4 \) | \( y_3^4 \) | \( y_4^4 \) | \( y_5^4 \) | \( y_6^4 \) | \( y_7^4 \) | \( y_8^4 \) |
| /IY/ | \( y_0^5 \) | \( y_1^5 \) | \( y_2^5 \) | \( y_3^5 \) | \( y_4^5 \) | \( y_5^5 \) | \( y_6^5 \) | \( y_7^5 \) | \( y_8^5 \) |
| /F/  | \( y_0^6 \) | \( y_1^6 \) | \( y_2^6 \) | \( y_3^6 \) | \( y_4^6 \) | \( y_5^6 \) | \( y_6^6 \) | \( y_7^6 \) | \( y_8^6 \) |
| /G/  | \( y_0^7 \) | \( y_1^7 \) | \( y_2^7 \) | \( y_3^7 \) | \( y_4^7 \) | \( y_5^7 \) | \( y_6^7 \) | \( y_7^7 \) | \( y_8^7 \) |

- **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

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

- 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
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

- 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:

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

• Or repeat the symbols over their duration

\[ DIV = \sum_t KL(Y_t, symbol_t) = - \sum_t \log Y(t, symbol_t) \]
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. $S_0 S_1 S_1 S_2 S_3 S_3 \ldots S_{K-1}$
      – i.e. $s_0 = S_0, s_2 = S_1, s_3 = S_1, s_4 = S_2, s_5 = S_3, \ldots s_{N-1} = S_{K-1}$
    • 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

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

- 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

- 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/
### Blocking out unnecessary outputs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</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>/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>

### Computations

1. Compute the entire output (for all symbols)
2. Copy the output values for the target symbols into the secondary reduced structure
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>

- 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
Arrangement of the constructed table

**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>/IY/</th>
<th>/F/</th>
<th>/IY/</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_0^B$</td>
<td>$y_0^{IY}$</td>
<td>$y_0^F$</td>
<td>$y_0^{IY}$</td>
</tr>
<tr>
<td>$y_1^B$</td>
<td>$y_1^{IY}$</td>
<td>$y_1^F$</td>
<td>$y_1^{IY}$</td>
</tr>
<tr>
<td>$y_2^B$</td>
<td>$y_2^{IY}$</td>
<td>$y_2^F$</td>
<td>$y_2^{IY}$</td>
</tr>
<tr>
<td>$y_3^B$</td>
<td>$y_3^{IY}$</td>
<td>$y_3^F$</td>
<td>$y_3^{IY}$</td>
</tr>
<tr>
<td>$y_4^B$</td>
<td>$y_4^{IY}$</td>
<td>$y_4^F$</td>
<td>$y_4^{IY}$</td>
</tr>
<tr>
<td>$y_5^B$</td>
<td>$y_5^{IY}$</td>
<td>$y_5^F$</td>
<td>$y_5^{IY}$</td>
</tr>
<tr>
<td>$y_6^B$</td>
<td>$y_6^{IY}$</td>
<td>$y_6^F$</td>
<td>$y_6^{IY}$</td>
</tr>
<tr>
<td>$y_7^B$</td>
<td>$y_7^{IY}$</td>
<td>$y_7^F$</td>
<td>$y_7^{IY}$</td>
</tr>
<tr>
<td>$y_8^B$</td>
<td>$y_8^{IY}$</td>
<td>$y_8^F$</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 2nd and 4th 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

\[
Scr(Path) = y_0^B y_1^B y_2^{IY} y_3^{IY} y_4^F
\]
• 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 \( \text{BestParent} \)

\[ = \arg \max_{\text{Parent} \in (y_2^{\text{IY}}, y_2^F)} (\text{Score}(\text{BestPath}(y_0^B \rightarrow \text{Parent}))) \]
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
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 \( \text{(given inputs } X_0 \ldots X_t) \)
  - 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:

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

  \[ Bscr(0, 0) = \gamma_0^{s(0)}, \ Bscr(0, i) = -\infty, \ i = 1 \ldots K - 1 \]
Viterbi algorithm

- Initialization:

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

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

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

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

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

• for \( t = 1 \ldots T - 1 \)

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

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

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

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

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

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

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

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

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

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

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

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

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

- **Initialization:**

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

  \[ Bscr(0,0) = \gamma_0^{S(0)}, \quad Bscr(0, i) = -\infty, \quad i = 1 \ldots K - 1 \]

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

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

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

  - \( BP(t, l) = \left( \text{if } (Bscr(t - 1, l - 1) > Bscr(t - 1, l)) \ldots l - 1; \text{ else } l \right) \)
  - \( Bscr(t, l) = Bscr(BP(t, l)) \times \gamma(t)^{S(l)} \)
Viterbi algorithm

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

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

• Initialization:

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

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

• for \( t = 1 \ldots T - 1 \)

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

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

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

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

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

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

  \[
  BP(t, 0) = 0; \quad Bsc r(t, 0) = Bsc r(t - 1,0) \times y_t^{S(0)} \\
  \text{for } l = 1 \ldots K - 1 
  \]

  - \( BP(t,l) = \begin{cases} 
    l - 1 : & \text{if } (Bsc r(t - 1,l - 1) > Bsc r(t - 1,l)) \ l - 1; \\
    l : & \text{else}
  \end{cases} \)

  - \( Bsc r(t,l) = Bsc r(BP(t,l)) \times y_t^{S(l)} \)
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 \( t = T - 1 \) down to 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) = -infty
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) = -infty
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))

Do not need explicit construction of output table
Information about order already in symbol sequence S(i), so we can use y(t,S(i)) instead of composing s(t,i) = y(t,S(i)) and using s(t,i)

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) = -infty
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

\[
\begin{array}{cccccccccc}
/B/ & /B/ & /IY/ & /F/ & /F/ & /IY//IY/ & /IY/ & /IY/ \\
\hline
y_0^B & y_1^B & y_2^B & y_3^B & y_4^B & y_5^B & y_6^B & y_7^B & y_8^B \\
y_0^{IY} & y_1^{IY} & y_2^{IY} & y_3^{IY} & y_4^{IY} & y_5^{IY} & y_6^{IY} & y_7^{IY} & y_8^{IY} \\
y_0^F & y_1^F & y_2^F & y_3^F & y_4^F & y_5^F & y_6^F & y_7^F & y_8^F \\
y_0^{IY} & y_1^{IY} & y_2^{IY} & y_3^{IY} & y_4^{IY} & y_5^{IY} & y_6^{IY} & y_7^{IY} & y_8^{IY} \\
\end{array}
\]
Gradients from the alignment

$$\begin{array}{ccccccc}
/B/ & /B/ & /IY/ & /F/ & /F/ & /IY/ & /IY/ \\
y_0^B & y_1^B & y_2^B & y_3^B & y_4^B & y_5^B & y_6^B \\
y_0^{IY} & y_1^{IY} & y_2^{IY} & y_3^{IY} & y_4^{IY} & y_5^{IY} & y_6^{IY} \\
y_0^F & y_1^F & y_2^F & y_3^F & y_4^F & y_5^F & y_6^F \\
y_0^{IY} & y_1^{IY} & y_2^{IY} & y_3^{IY} & y_4^{IY} & y_5^{IY} & y_6^{IY} \\
\end{array}$$

$$DIV = \sum_t KL(Y_t, symbol_{t\text{bestpath}}) = - \sum_t \log Y(t, symbol_{t\text{bestpath}})$$

- The gradient w.r.t the $t$-th output vector $Y_t$

$$\nabla_{Y_t} DIV = \begin{bmatrix}
0 & 0 & \ldots & \frac{-1}{Y(t, symbol_{t\text{bestpath}})} & 0 & \ldots & 0
\end{bmatrix}$$

- Zeros except at the component corresponding to the target in the estimated alignment
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..