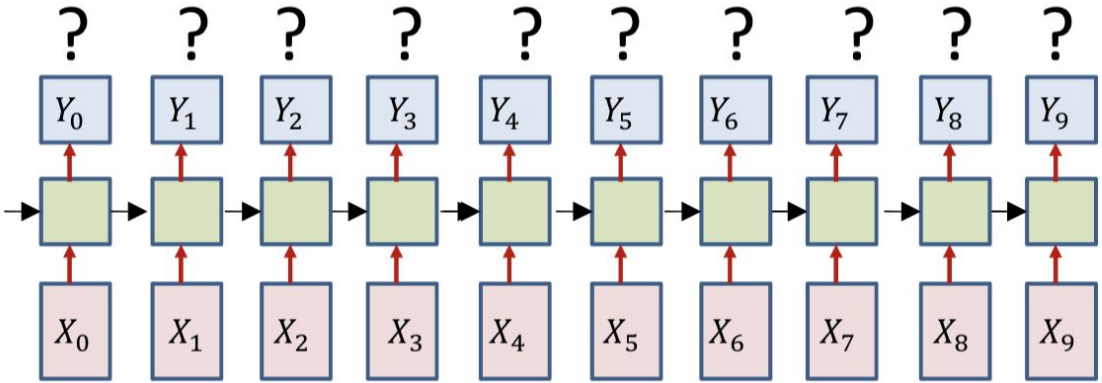
Recitation 8

CTC Decoding & Beam Search

Sequence to Sequence Modeling

Order-Synchronous, Not Time-Synchronous Output

1. Training -> we already know how to do that
2. Testing -> “Decoding” or “obtaining an output from a sequence-to-sequence network”



A key decoding problem

- Consider a problem where the output symbols are characters
- We have a decode: RRROOOOOD
- Is this the merged symbol sequence ROD or ROOD?

How to distinguish between an extended symbol and repetitions of a symbol?

A key decoding problem

Solution: Introduce an explicit extra symbol which serves to separate discrete versions of a symbol

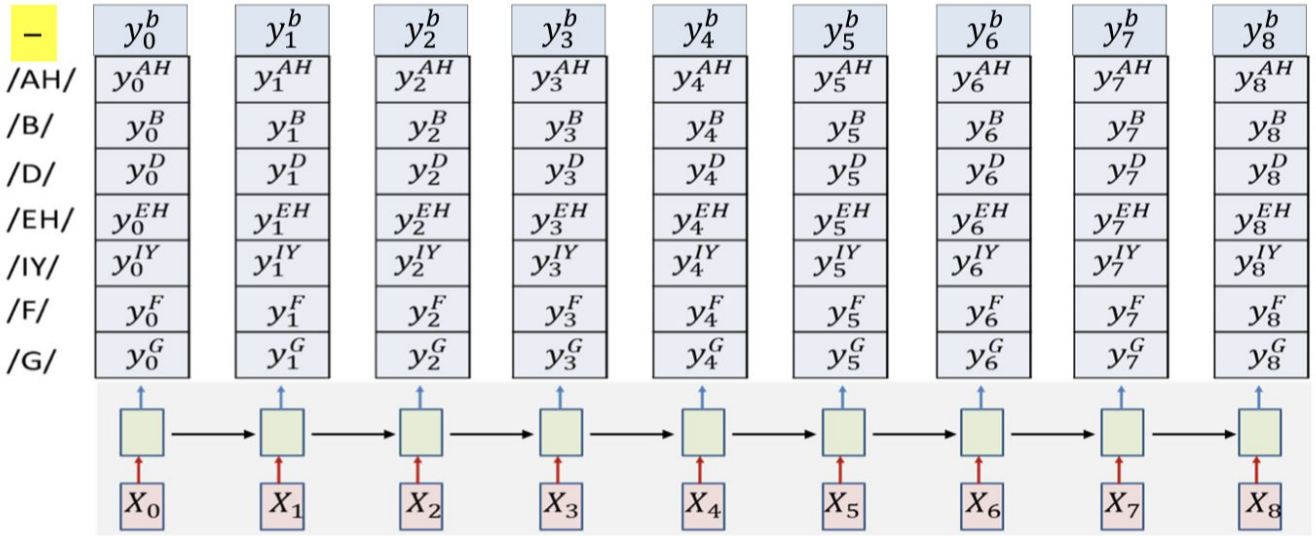
- A “blank” (represented by “-”)
- RRR---OO---DDD = ROD
- RR-R---OO---D-DD = RRODD
- R-R-R---O-ODD-DDDD-D = RRROODDD

The symbol set recognized by the network must now include the extra blank symbol

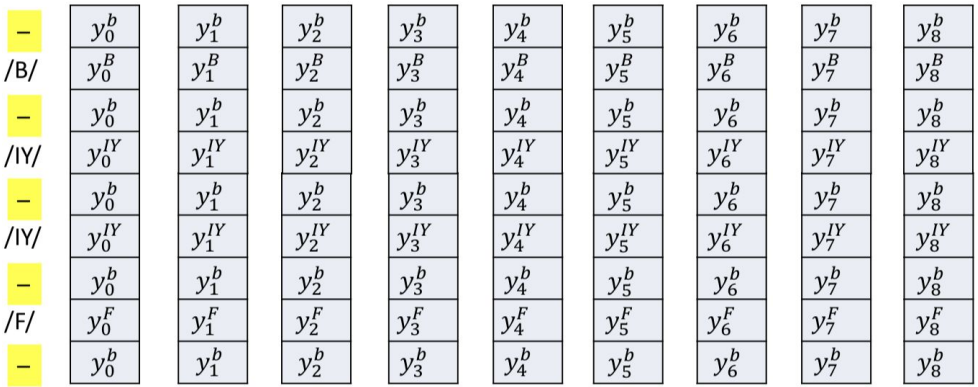
- Which too must be trained

The modified forward output

Note the extra “blank”at the output



Composing graph for training



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

Train as before!

CTC: Connectionist Temporal Classification

- The overall framework we saw is referred to as CTC

  – Applies when “duplicating” labels at the output is considered acceptable, and when output sequence length < input sequence length

Returning to the decoding problem

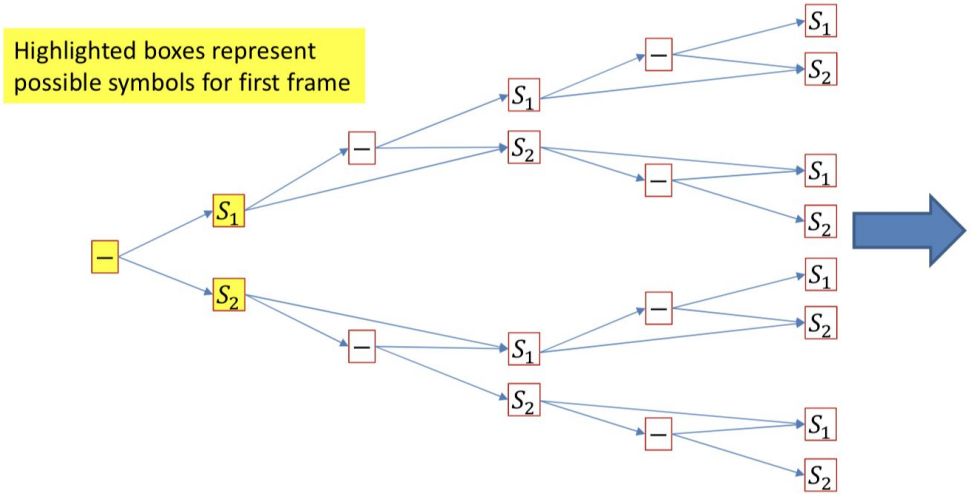
How to decode at test time?

- Greedy decode -> choose symbol with highest probability at each time step and merge
  - Sub-optimal decode which finds most likely synchronous output sequence

- Objective of decoding -> Most likely asynchronous symbol sequence
  - Find all decodings and pick the most likely decode!
  - Unfortunately, explicit computation of this will require evaluate of an exponential number of symbol sequences
  - Solution: Organize all possible symbol sequences as a (semi)tree

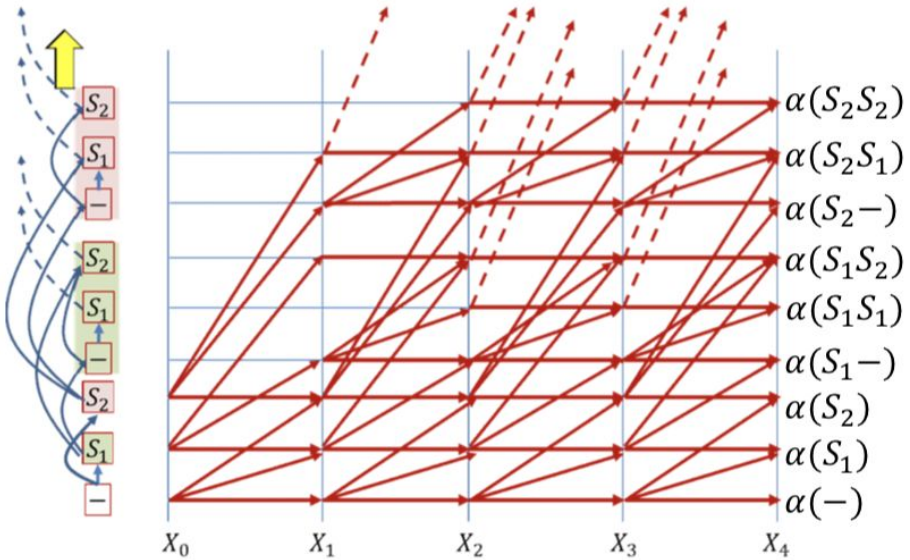
Hypothesis semi-tree

- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
- It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique symbol sequence!



Decoding graph for the tree

- The figure to the left is the tree, drawn in a vertical line
- The graph is just the tree unrolled over time
- The alpha at final time represents the full forward score for a unique symbol sequence
- Select the symbol sequence with the largest alpha



Pruning

- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable

BEAM SEARCH

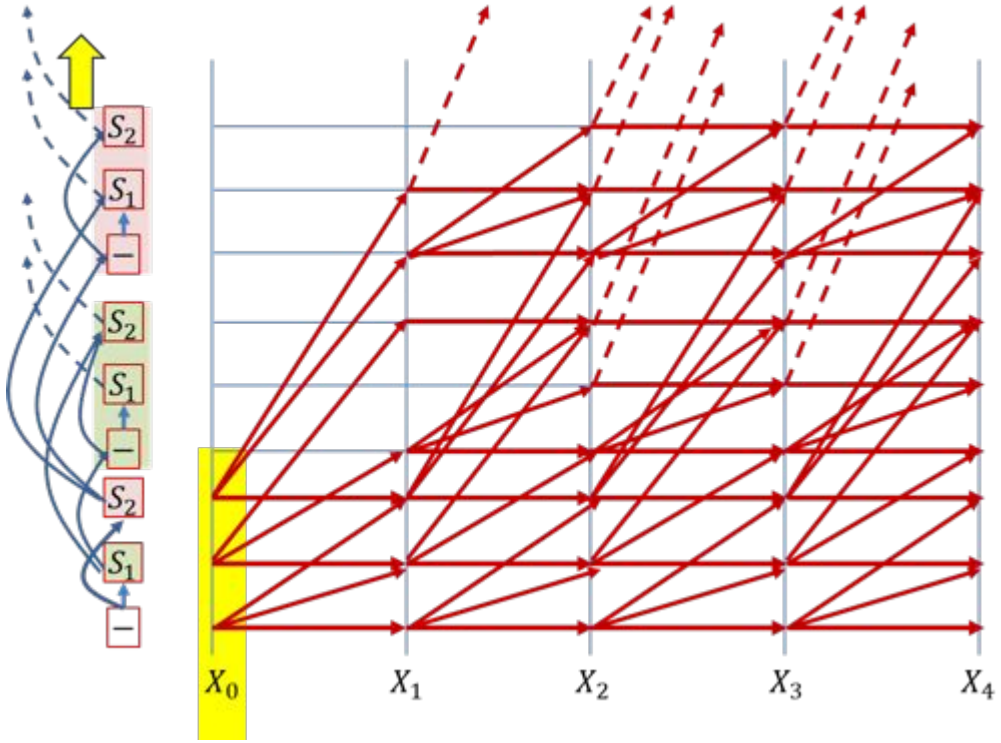
- **PathScore** : array of scores for paths ending with symbols
- **BlankPathScore** : array of scores for paths ending with blanks
- **SymbolSet** : A list of symbols *not* including the blank

BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                        InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                           PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                       PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                        UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

BEAM SEARCH

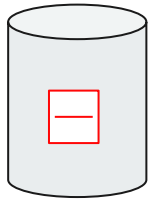
```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                        InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                           PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                       PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                  Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                                  UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```


S_2
S_1
–
S_2
S_1
–
S_2
S_1
–
X_0
X_1
X_2
X_3
X_4

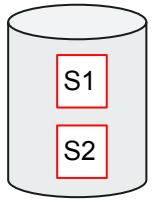
BEAM SEARCH InitializePaths: FIRST TIME INSTANT

```
Global PathScore, BlankPathScore

function InitializePaths(SymbolSet, y, BeamWidth)

# First push the blank into a path-ending-with-blank stack. No symbol has been invoked yet
path = null
BlankPathScore[path] = y[blank]  # Score of blank at t=1
InitialPathsWithFinalBlank = {path}

# Push rest of the symbols into a path-ending-with-symbol stack
InitialPathsWithFinalSymbol = {}
for c in SymbolSet  # This is the entire symbol set, without the blank
    path = c
    PathScore[path] = y[c]  # Score of symbol c at t=1
    InitialPathsWithFinalSymbol += path # Set addition
end

# Prune poor paths and return
return Prune(InitialPathsWithFinalBlank, InitialPathsWithFinalSymbol, BlankPathScore, PathScore, BeamWidth)
```

InitialPathWithFinalBlank

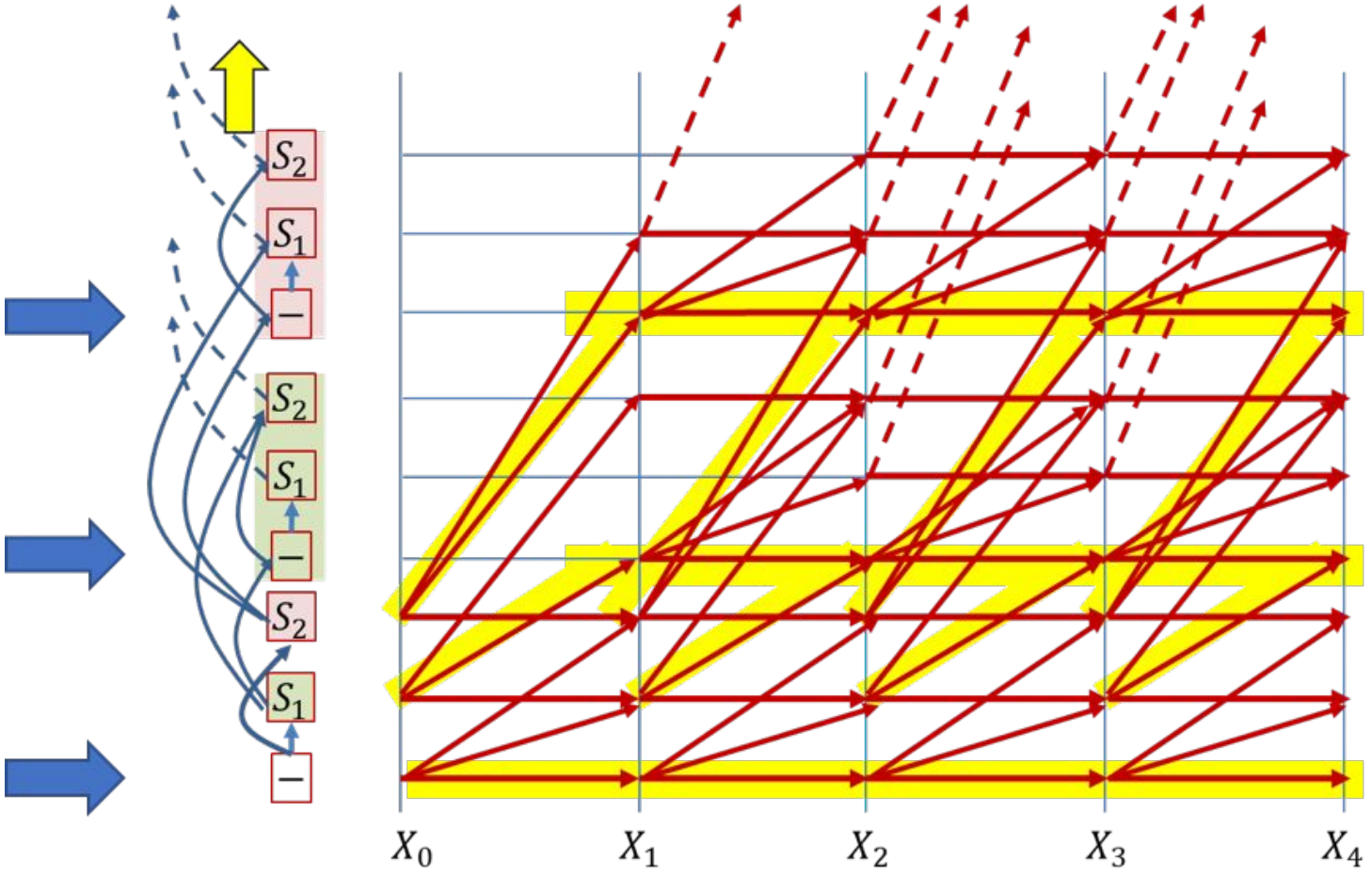


InitialPathWithFinalSymbols



BEAM SEARCH

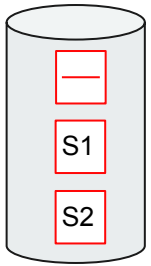
```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                        InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                           PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                       PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                                     UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```


S_2
S_1
$-$
S_2
S_1
$-$
S_2
S_1
$-$
X_0
X_1
X_2
X_3
X_4

BEAM SEARCH: Extending with blanks

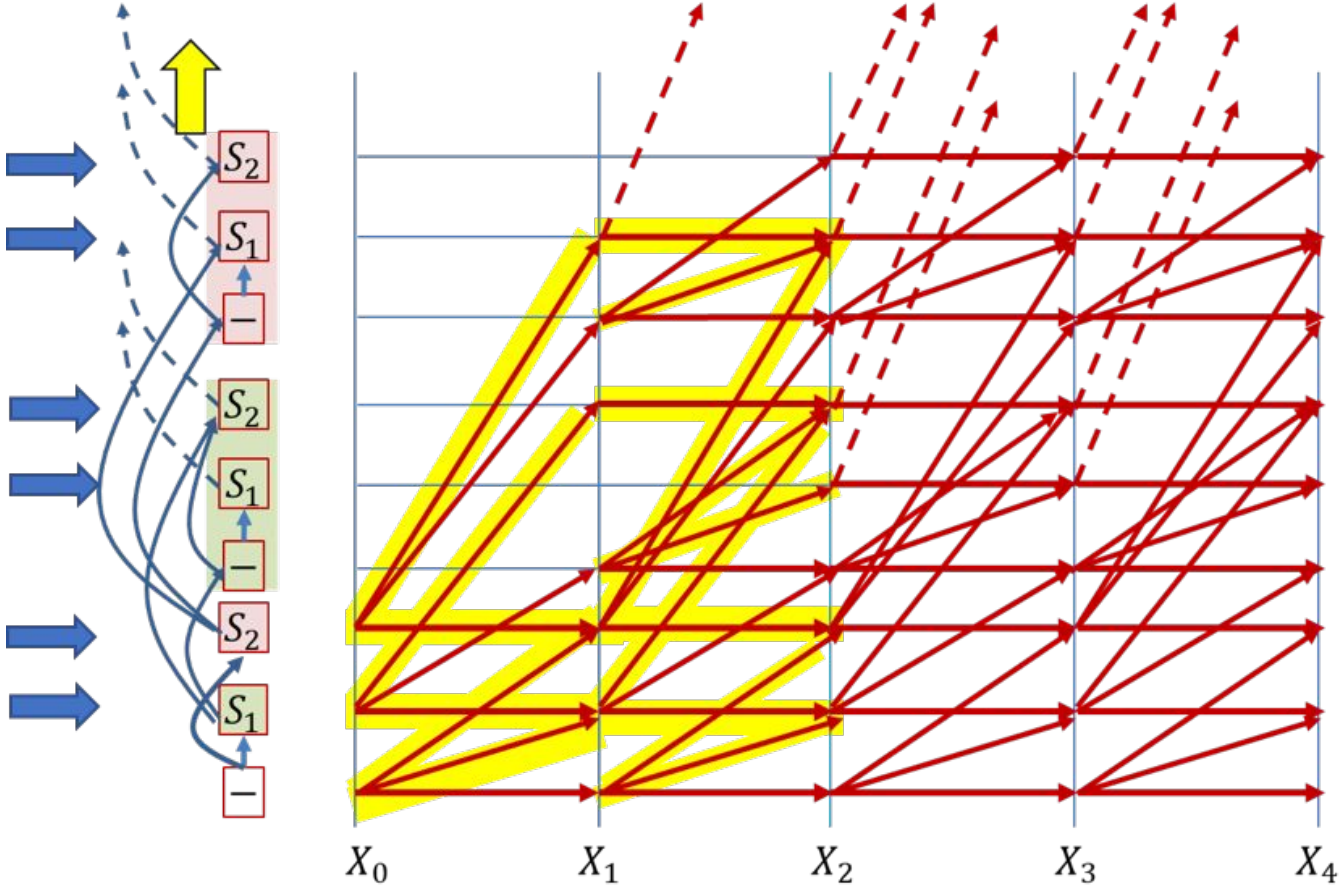
```
Global PathScore, BlankPathScore

function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # Repeating a blank doesn't change the symbol sequence
        UpdatedPathsWithTerminalBlank += path  # Set addition
        UpdatedBlankPathScore[path] = BlankPathScore[path]*y[blank]
    end

    # Then extend paths with terminal symbols by blanks
    for path in PathsWithTerminalSymbol:
        # If there is already an equivalent string in UpdatesPathsWithTerminalBlank
        # simply add the score. If not create a new entry
        if path in UpdatedPathsWithTerminalBlank
            UpdatedBlankPathScore[path] += Pathscore[path]* y[blank]
        else
            UpdatedPathsWithTerminalBlank += path # Set addition
            UpdatedBlankPathScore[path] = PathScore[path] * y[blank]
        end
    end

    return UpdatedPathsWithTerminalBlank,
           UpdatedBlankPathScore
```

(only at t=1)
UpdatedPathsWIthTerminalBlank



BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                          InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                        PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                  Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                              UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

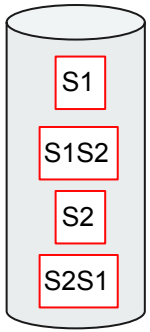


(figure shows path extensions for only 2 time steps)

BEAM SEARCH: Extending with symbols

```
Global PathScore, BlankPathScore

function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalSymbol = {}
    UpdatedPathScore = []
    # First work on paths with terminal symbols
    for path in PathsWithTerminalSymbol:
        # Extend the path with every symbol other than blank
        for c in SymbolSet:  # SymbolSet does not include blanks
            if c == path[end]  # The final symbol is repeated, so this is the same symbol sequence
                newpath = path
            else:
                newpath = path + c  # Concatenate new symbol to the path
            UpdatedPathsWithTerminalSymbol += newpath  # Set addition
            UpdatedPathScore[newpath] = PathScore[path] * y[c]
        end
    end

    # Then add in extensions of paths terminating in blanks
    for path in PathsWithTerminalBlank:
        for c in SymbolSet:  # SymbolSet does not include blanks
            newpath = path + c  # Concatenation
            # If there is already an equivalent string in UpdatesPathsWithTerminalSymbol
            # simply add the score. If not create a new entry
            if newpath in UpdatedPathsWithTerminalSymbol
                UpdatedPathScore[newpath] += BlankPathScore[path] * y(c)
            else
                UpdatedPathsWithTerminalSymbol += newpath # Set addition
                PathScore[newpath] = BlankPathScore[path] * y(c)
            end
        end
    end

    return UpdatedPathsWithTerminalSymbol,
           UpdatedPathScore
```

(only at t=1)
UpdatedPathsWIthTerminalSymbol



BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                          InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                         PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                  Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                              UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

BEAM SEARCH: Pruning low-scoring entries

```
Global PathScore, BlankPathScore

function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
    PrunedBlankPathScore = []
    PrunedPathScore = []
    # First gather all the relevant scores
    i = 0
    for p in PathsWithTerminalBlank
        scorelist[i] = BlankPathScore[p]
        i++
    end
    for p in PathsWithTerminalSymbol
        scorelist[i] = PathScore[p]
        i++
    end

    # Sort and find cutoff score that retains exactly BeamWidth paths
    sort(scorelist)  # In decreasing order
    cutoff = scorelist[BeamWidth]

    PrunedPathsWithTerminalBlank = {}
    for p in PathsWithTerminalBlank
        if BlankPathScore[p] > cutoff
            PrunedPathsWithTerminalBlank += p # Set addition
            PrunedBlankPathScore[p] = BlankPathScore[p]
        end
    end

    PrunedPathsWithTerminalSymbol = {}
    for p in PathsWithTerminalSymbol
        if PathScore[p] > cutoff
            PrunedPathsWithTerminalSymbol += p # Set addition
            PrunedPathScore[p] = PathScore[p]
        end
    end

    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
```

BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                          InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                         PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                  Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                              UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

BEAM SEARCH: Merging final paths

```
Global PathScore, BlankPathScore

function MergeIdenticalPaths(PathsWithTerminalBlank, PathsWithTerminalSymbol)

    # All paths with terminal symbols will remain
    MergedPaths = PathsWithTerminalSymbol
    for p in MergedPaths
        FinalPathScore[p] = PathScore[p]
    end

    # Paths with terminal blanks will contribute scores to existing identical paths from
    # PathsWithTerminalSymbol if present, or be included by themselves in the final set, otherwise
    for p in PathsWithTerminalBlank
        if p in MergedPaths
            FinalPathScore[p] += BlankPathScore[p]
        else
            MergedPaths += p # Set addition
            FinalPathScore[p] = BlankPathScore[p]
        end
    end




    return MergedPaths, FinalPathScore
```

BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=0
PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                        InitializePaths(SymbolSet, y[:,0], BeamWidth)

# Subsequent time steps
for t = 1:T
    # First extend paths by a blank
    UpdatedPathsWithTerminalBlank, UpdatedBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                           PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    UpdatedPathsWithTerminalSymbol, UpdatedPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                       PathsWithTerminalSymbol, SymbolSet, y[:,t])

    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
                  Prune(UpdatedPathsWithTerminalBlank, UpdatedPathsWithTerminalSymbol,
                                                       UpdatedBlankPathScore, UpdatedPathScore, BeamWidth)
end
# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(PathsWithTerminalBlank,
                                                  PathsWithTerminalSymbol)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```