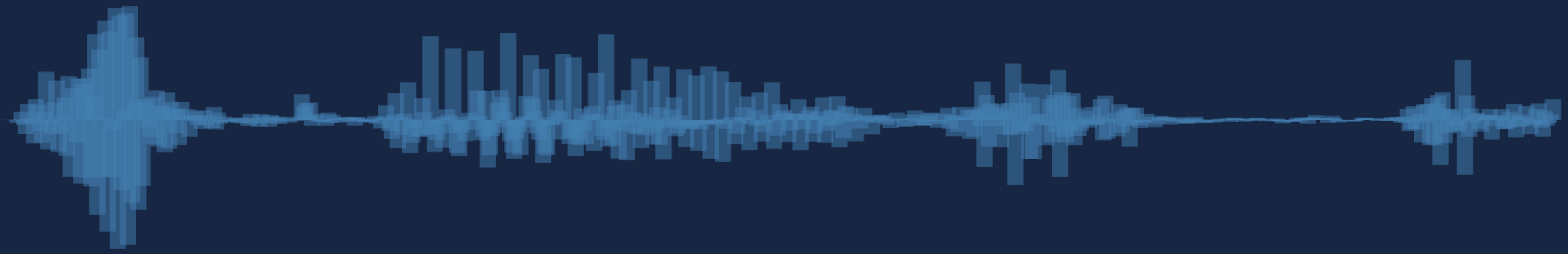
Speech Recognition and Graph Transformer Networks

Awni Hannun, awni@fb.com



facebook AI Research

Outline

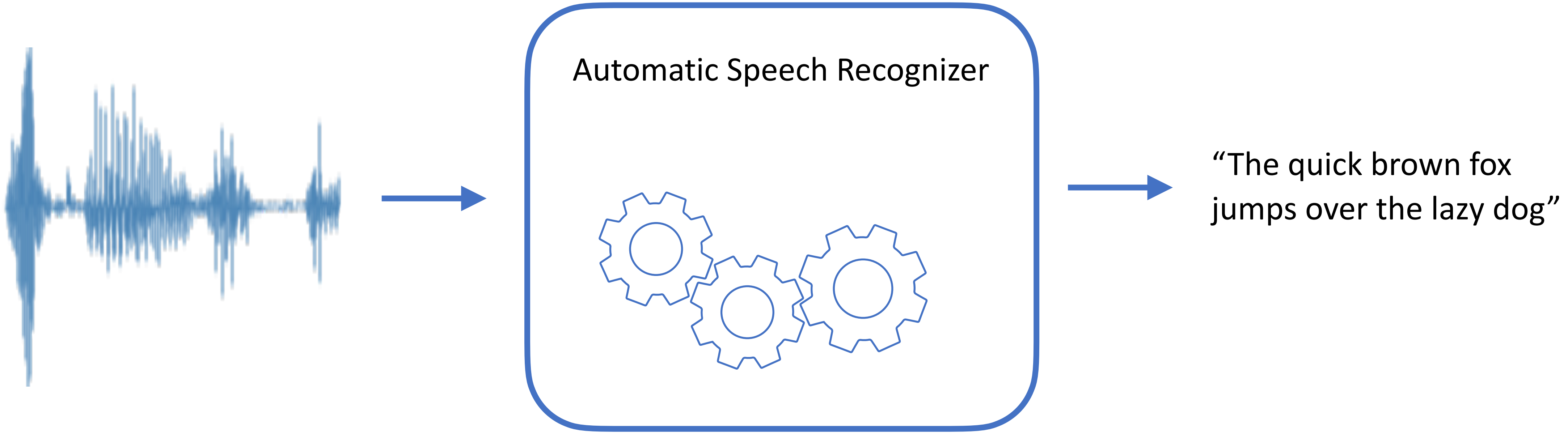
- Modern Speech Recognition
- Deep Dive: The CTC Loss
- Deep Dive: Decoding with Beam Search
- Graph Transformer Networks

Outline

- Modern Speech Recognition
- Deep Dive: The CTC Loss
- Deep Dive: Decoding with Beam Search
- Graph Transformer Networks

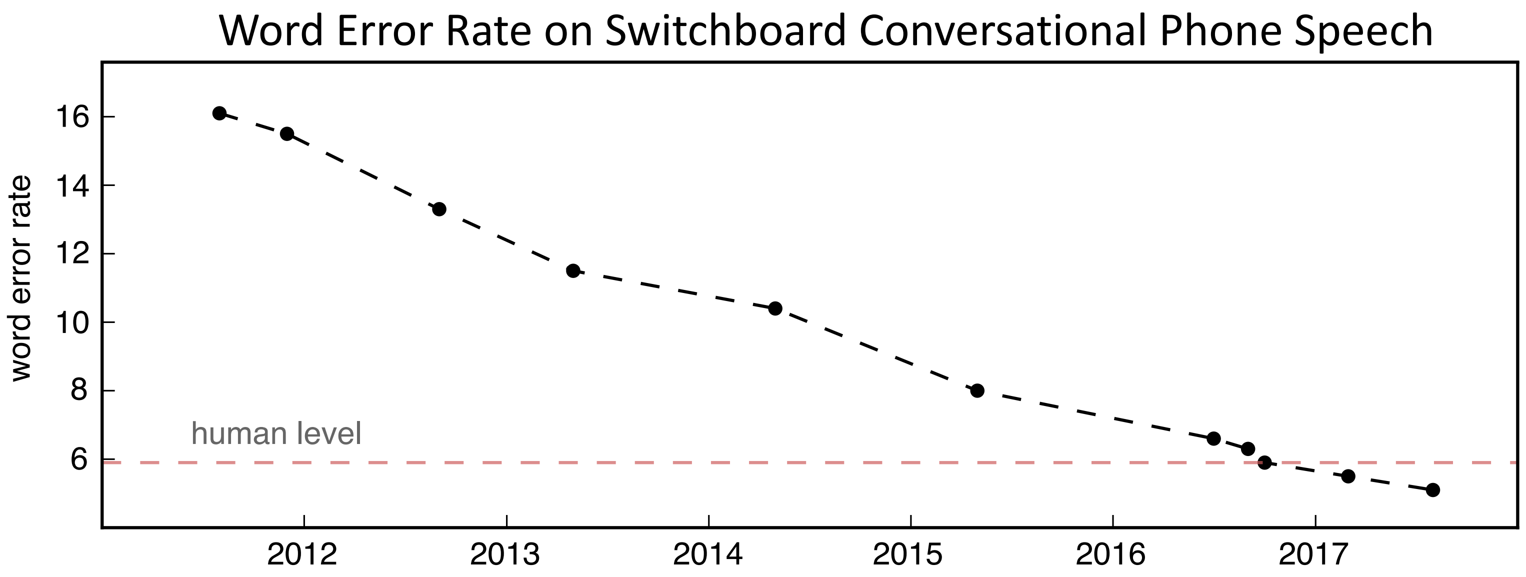
Automatic Speech Recognition

Goal: Input speech → output transcription



Automatic Speech Recognition

Improved significantly in the past 8 years



Automatic Speech Recognition

But not yet solved!

- **Conversation:** Fully conversational speech with multiple speakers
- **Noise:** Lot’s of background noise
- **Bias:** Substantially worse performance for underrepresented groups

Automatic Speech Recognition

But not yet solved!

[Submitted on 28 Mar 2021 (v1), last revised 1 Apr 2021 (this version, v2)]

Quantifying Bias in Automatic Speech Recognition

Siyuan Feng, Olya Kudina, Bence Mark Halpern, Odette Scharenborg

Automatic Speech Recognition

But not yet solved!

[Submitted on 28 Mar 2021 (v1), last revised 1 Apr 2021 (this version, v2)]

Quantifying Bias in Automatic Speech Recognition

Siyuan Feng, Olya Kudina, Bence Mark Halpern, Odette Scharenborg

Automatic Speech Recognition

But not yet solved!

[Submitted on 28 Mar 2021 (v1), last revised 1 Apr 2021 (this version, v2)]

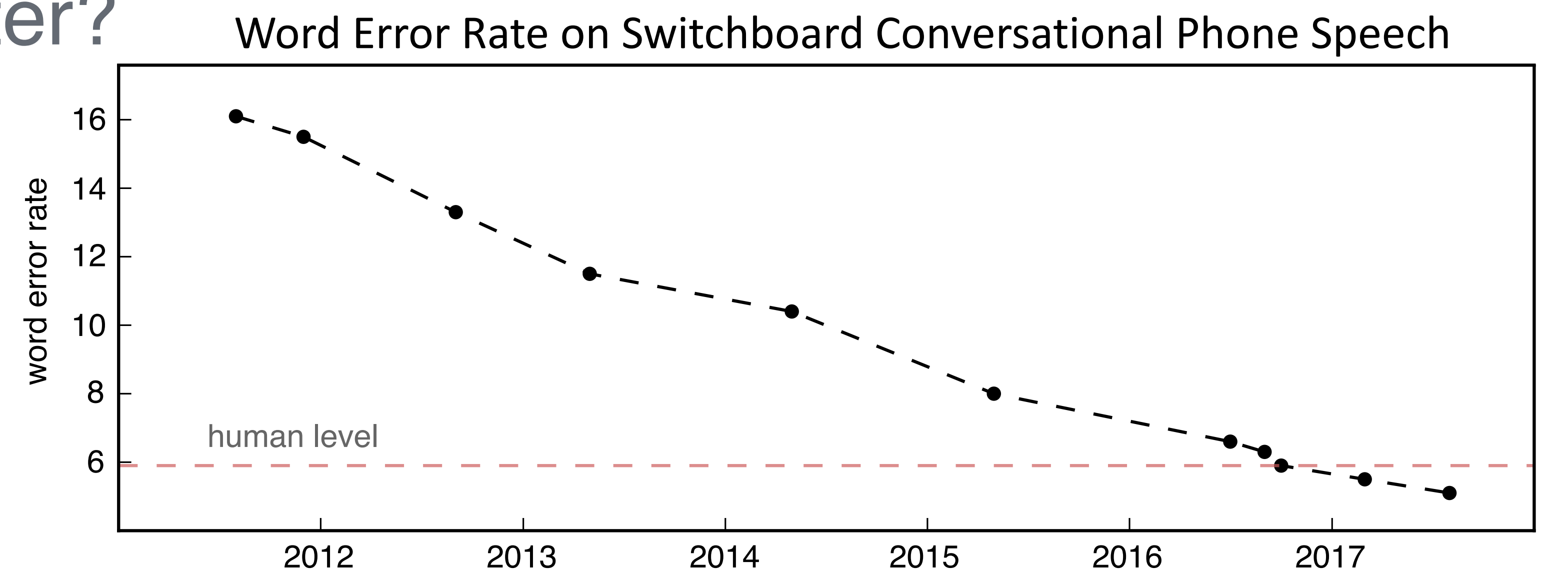
Quantifying Bias in Automatic Speech Recognition

Siyuan Feng, Olya Kudina, Bence Mark Halpern, Odette Scharenborg

"…state-of-the-art (SotA) ASRs **struggle** with the large variation in speech due to e.g., **gender, age, speech impairment, race, and accents**"

Automatic Speech Recognition

Question: Why has ASR gotten so much better?



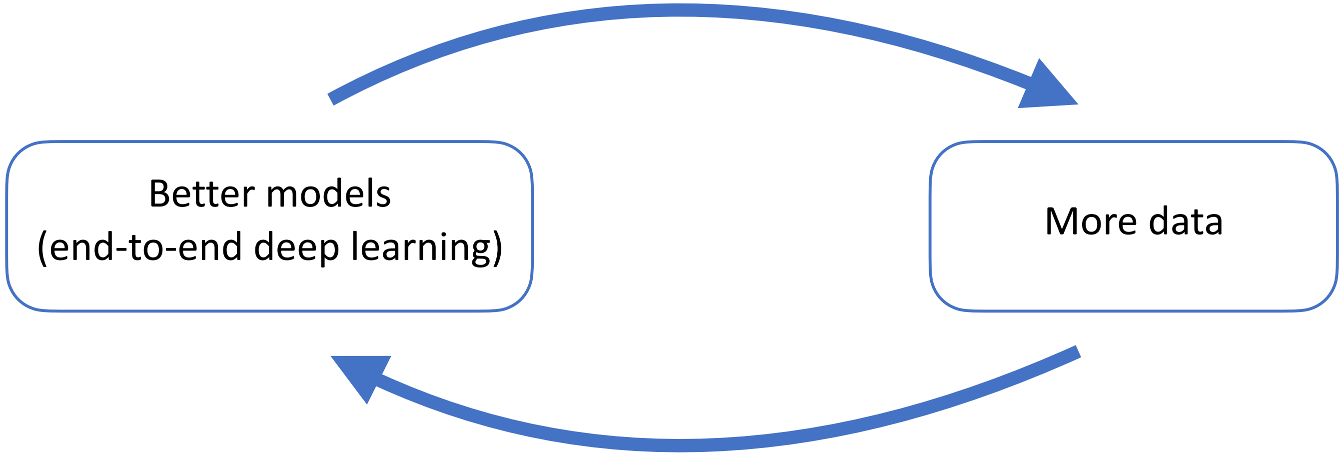
Automatic Speech Recognition

Pre 2012 ASR system:

- **Alphabet soup:** Too many hand-engineered components
- **Data:** Small and not useful
- **Cascading errors:** Combine modules only at the inference
- **Complex:** Difficult to do research

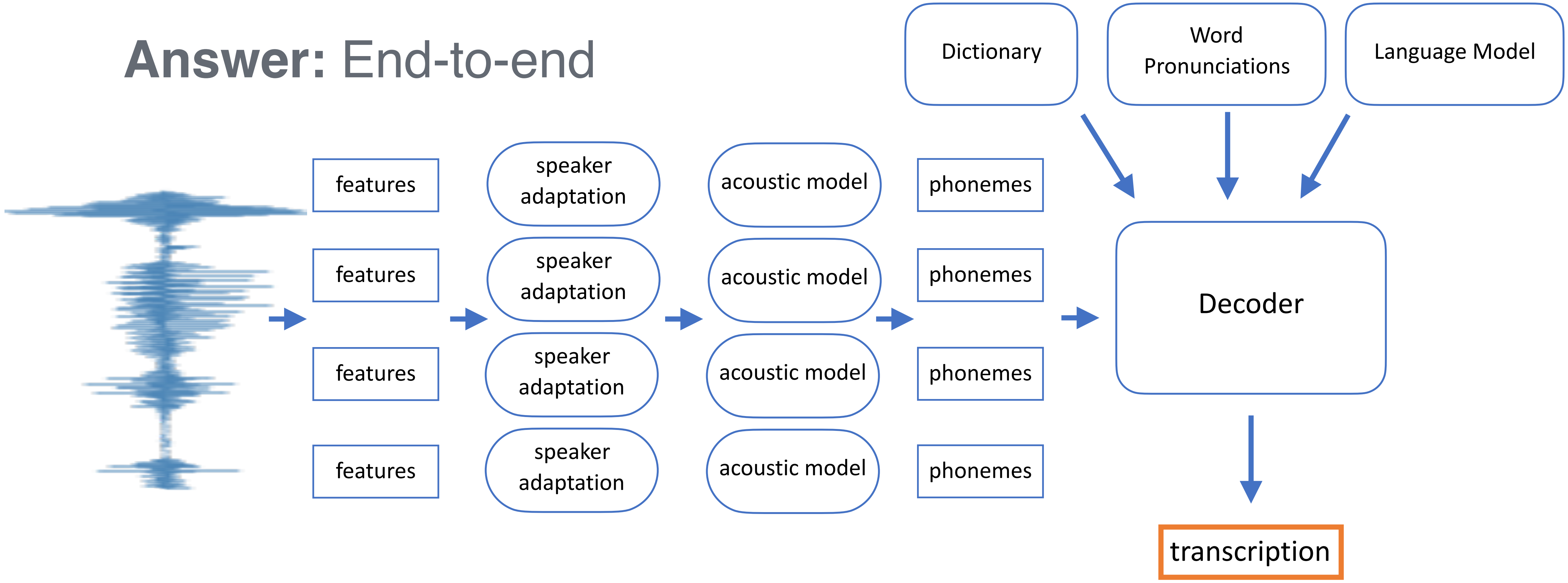
Automatic Speech Recognition

Question: Why has ASR gotten so much better?



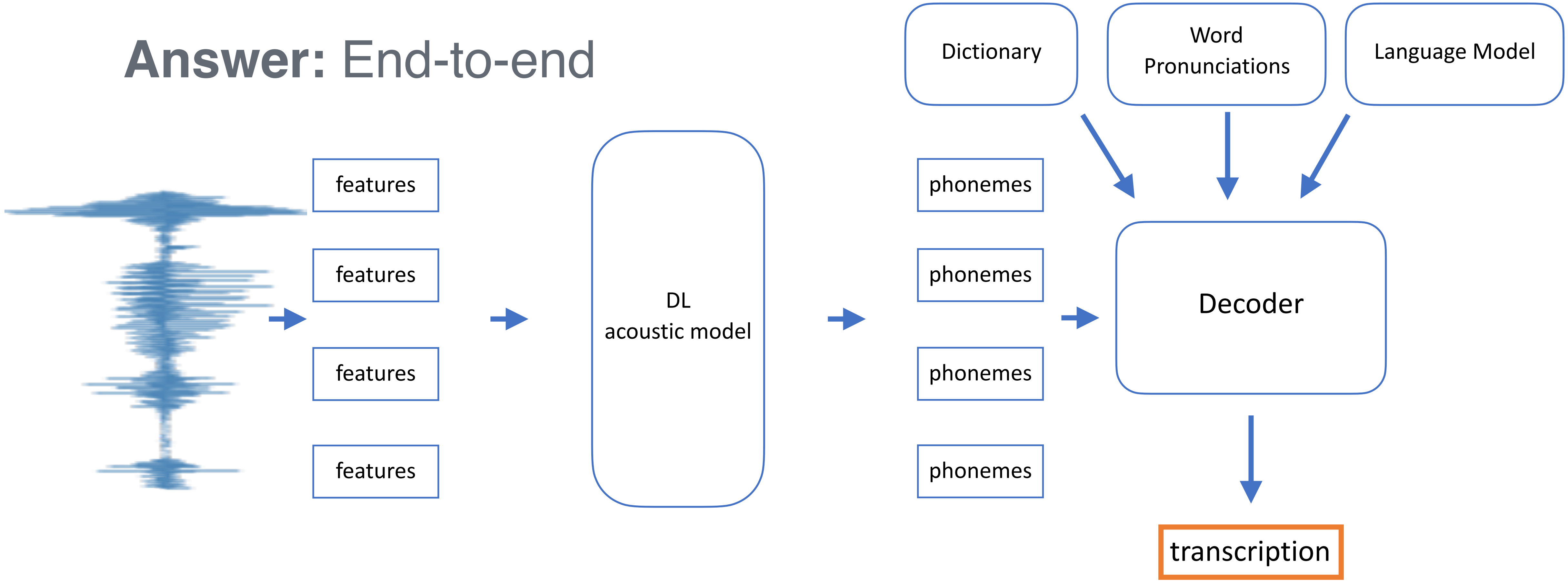
Automatic Speech Recognition

Answer: End-to-end



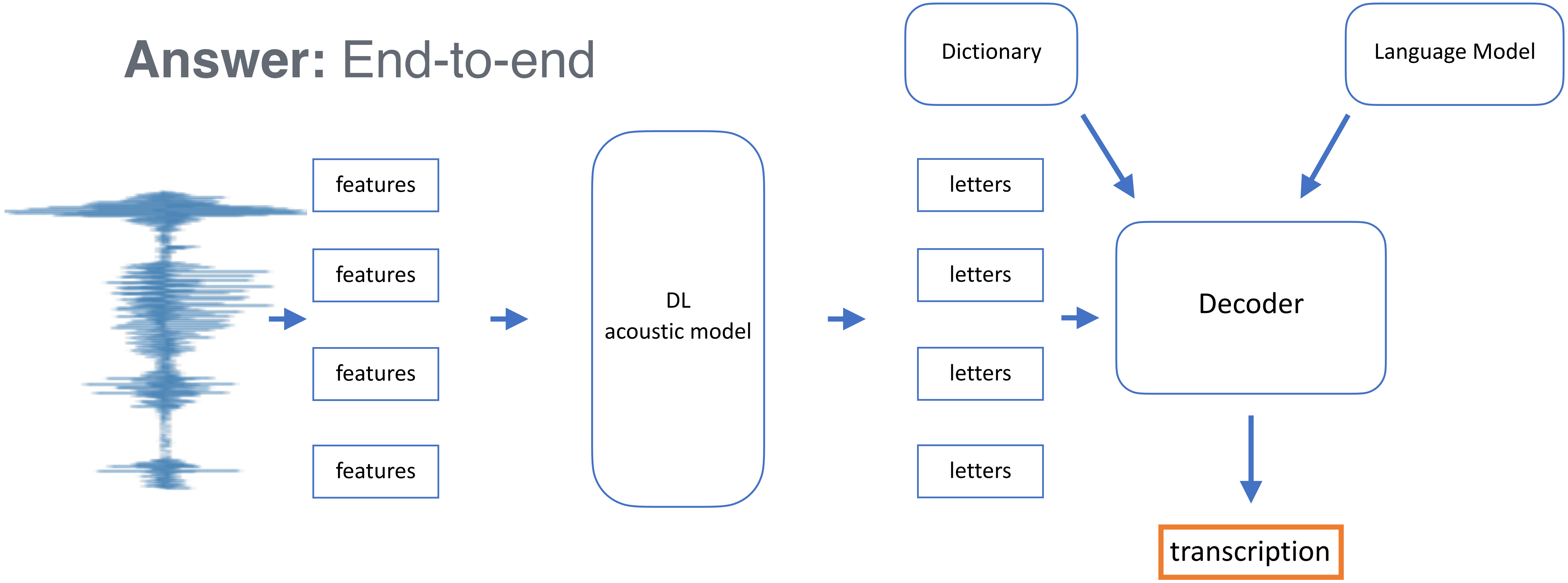
Automatic Speech Recognition

Answer: End-to-end



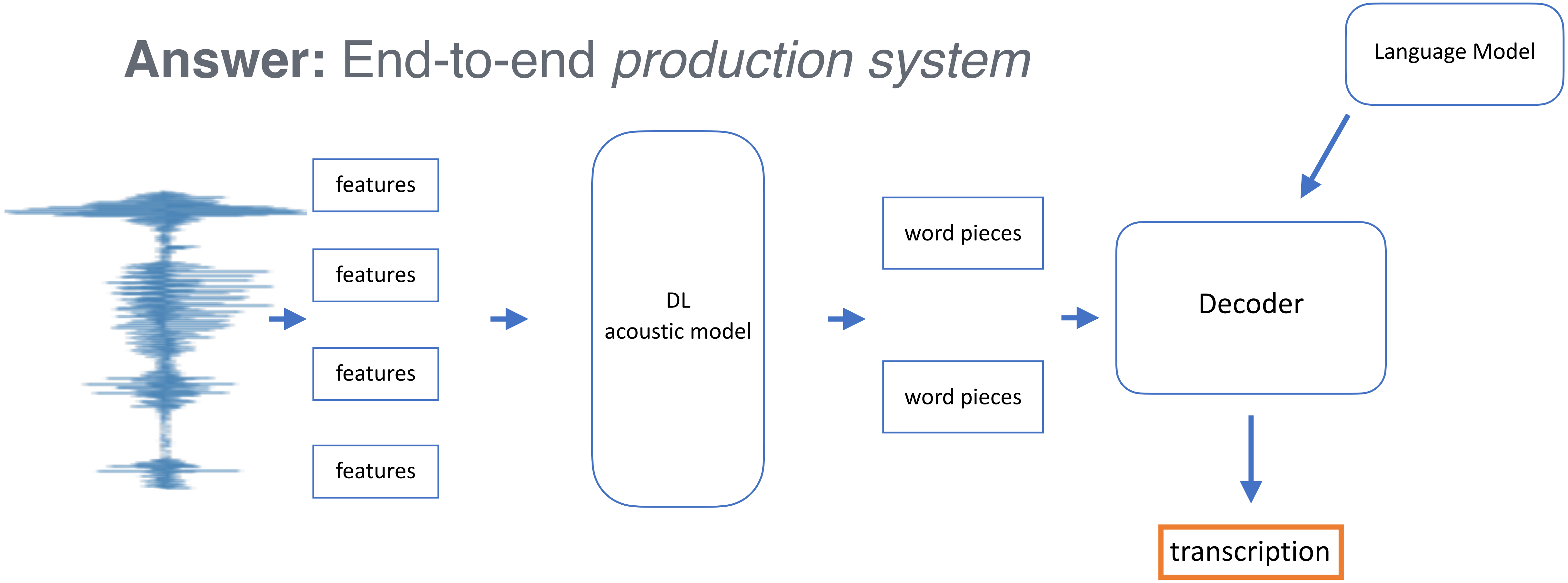
Automatic Speech Recognition

Answer: End-to-end



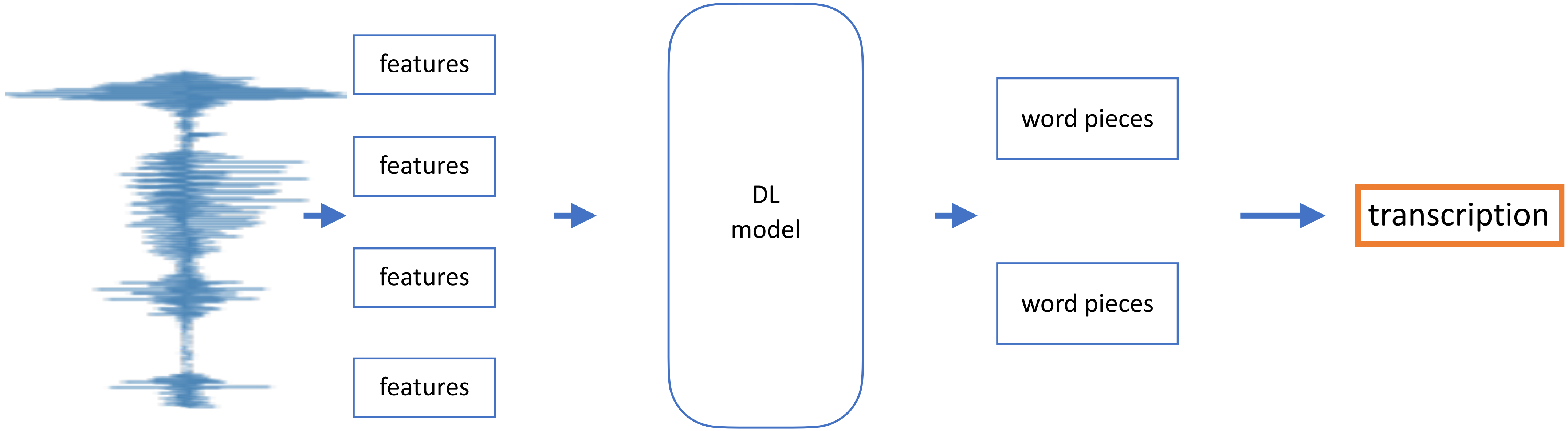
Automatic Speech Recognition

Answer: End-to-end *production system*



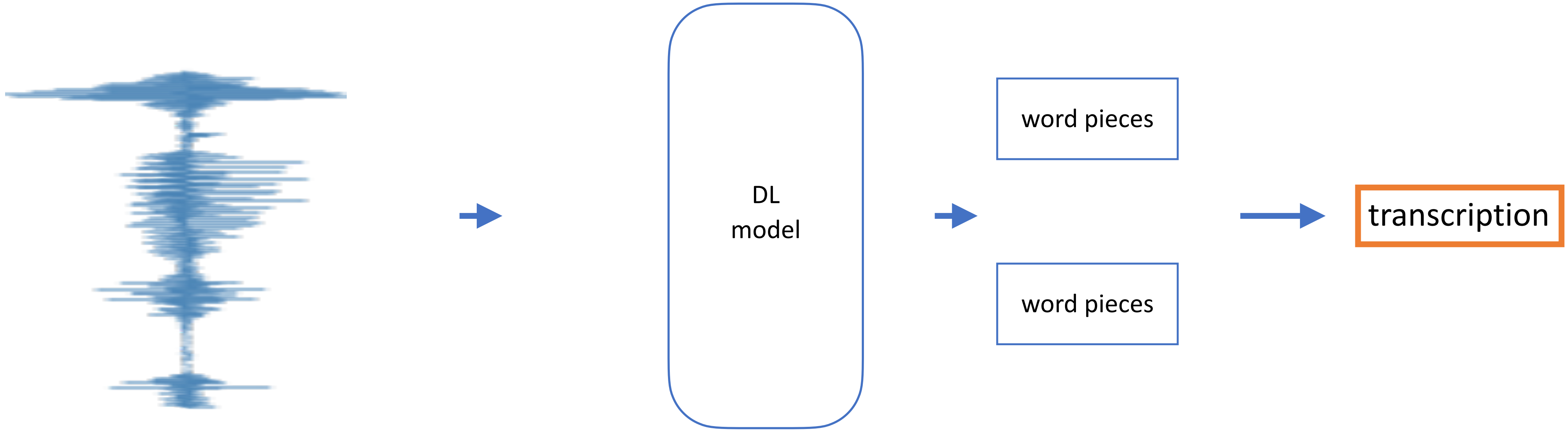
Automatic Speech Recognition

Answer: End-to-end *in research*



Automatic Speech Recognition

Answer: End-to-end *in research*



Outline

- Modern Speech Recognition
- Deep Dive: The CTC Loss
- Deep Dive: Decoding with Beam Search
- Graph Transformer Networks

The CTC Loss

Goal: Given

1. Input speech $X = [x_1, \ldots, x_T]$
2. Output transcription $Y = [y_1, \ldots, y_U]$

Compute:

$$\log P(Y \mid X; \theta)$$

The CTC Loss

Goal: Given

1. Input speech $X = [x_1, \ldots, x_T]$

2. Output transcription $Y = [y_1, \ldots, y_U]$

Compute:

$$\log P(Y \mid X; \boxed{\theta})$$

Ideally differentiable w.r.t. model parameters

The CTC Loss

Example:

1. Input speech $X = [x_1, x_2, x_3]$
2. Output transcription $Y = [c, a, t]$

Compute:

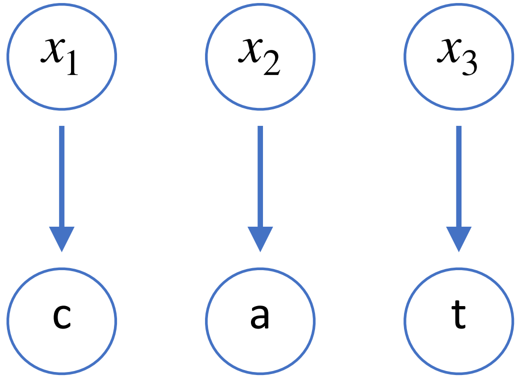
$$\log P(c \mid x_1) + \log P(a \mid x_2) + \log P(t \mid x_3)$$

The CTC Loss

Example:

1. Input speech $X = [x_1, x_2, x_3]$

2. Output transcription $Y = [c, a, t]$



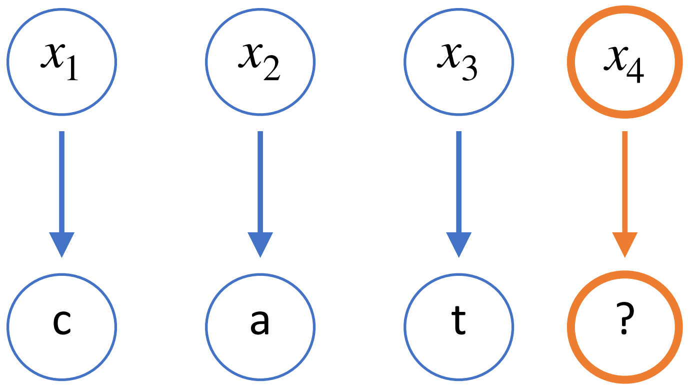
Compute:

$$\log P(c \mid x_1) + \log P(a \mid x_2) + \log P(t \mid x_3)$$

The CTC Loss

Example:

1. Input speech $X = [x_1, x_2, \boxed{x_3}, x_4]$
2. Output transcription $Y = [c, a, t]$

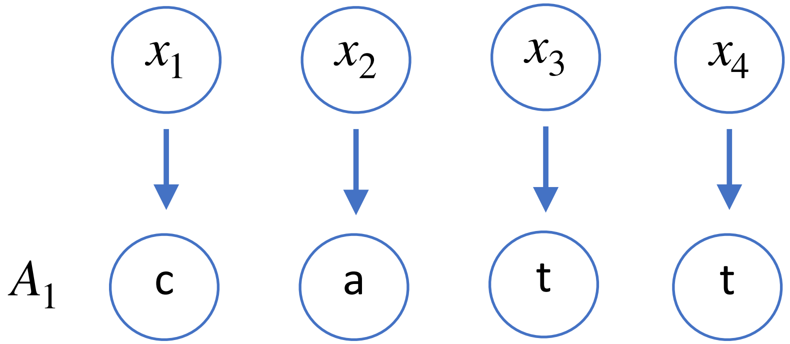


Compute:

$$\log P(c\,|\,x_1) + \log P(a\,|\,x_2) + \log P(t\,|\,x_3) + \log \boxed{P(??\,|\,x_4)}$$

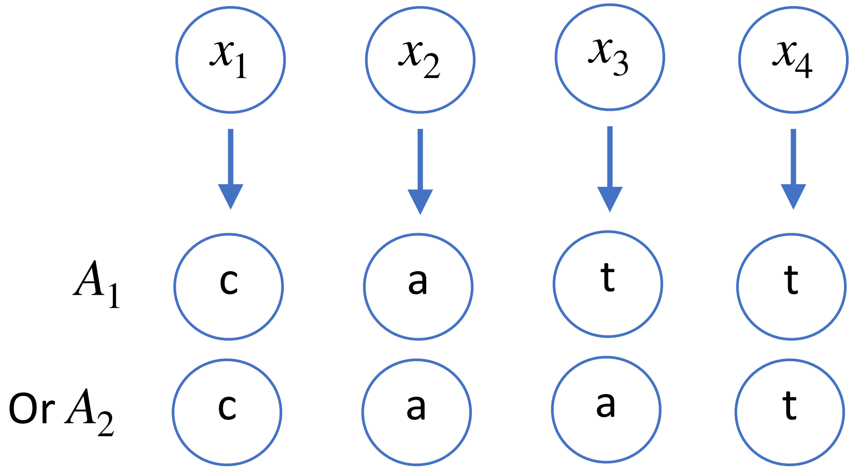
The CTC Loss

Alignment: One or more of each input maps to an output.



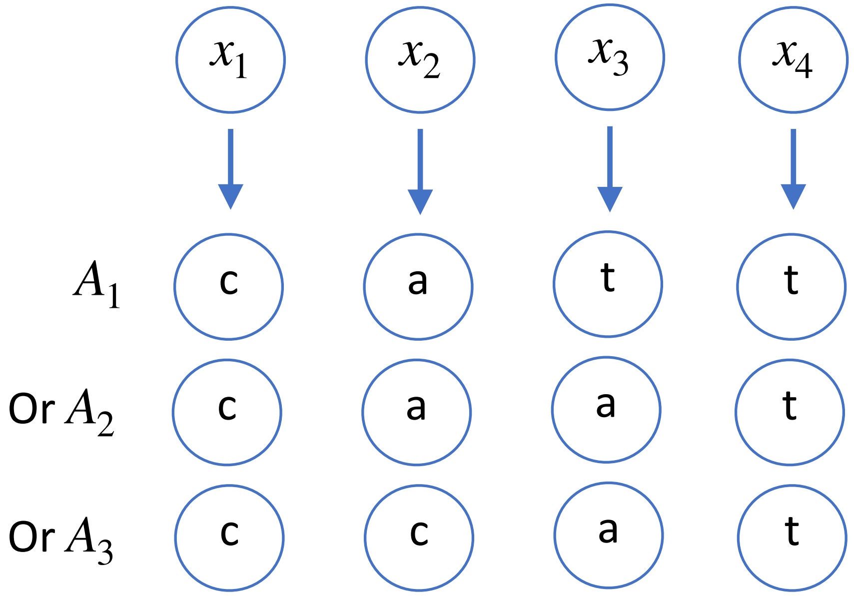
The CTC Loss

Alignment: One or more of each input maps to an output.



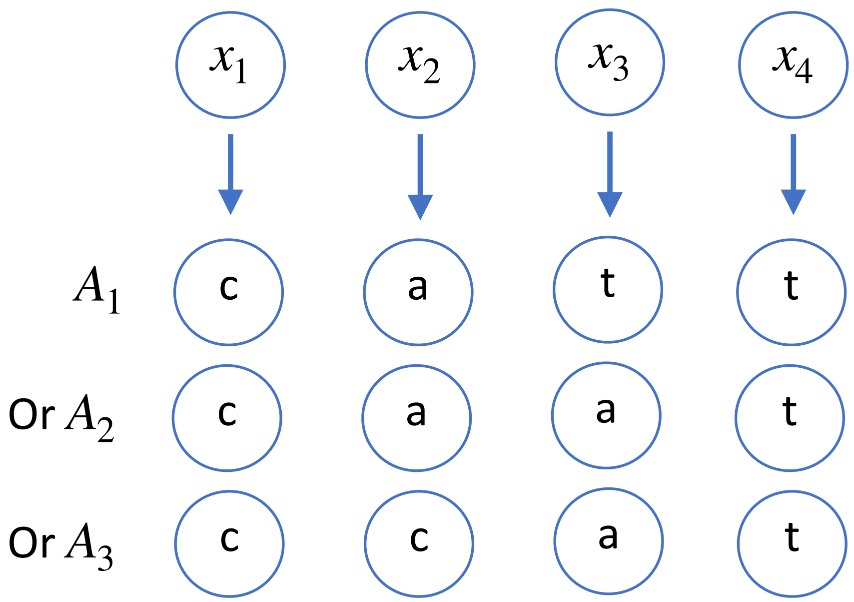
The CTC Loss

Alignment: One or more of each input maps to an output.



The CTC Loss

Q: Which alignment should we use to compute $\log P(Y \mid X)$?



The CTC Loss

Q: Which alignment should we use to compute $\log P(Y \mid X)$?

A: All of them!

$$\log P(Y \mid X) = \log \left[P(A_1 \mid X) + P(A_2 \mid X) + P(A_3 \mid X) \right]$$

The CTC Loss

Reminder: Use actual-softmax to sum log probabilities

Want $\log(P_1 + P_2)$ from $\log P_1$ and $\log P_2$

$$\text{actual-softmax}(\log P_1, \log P_2) = \log(P_1 + P_2)$$
$$= \log(e^{\log P_1} + e^{\log P_2})$$

The CTC Loss

Q: Which alignment should we use to compute $\log P(Y \mid X)$?

A: All of them!

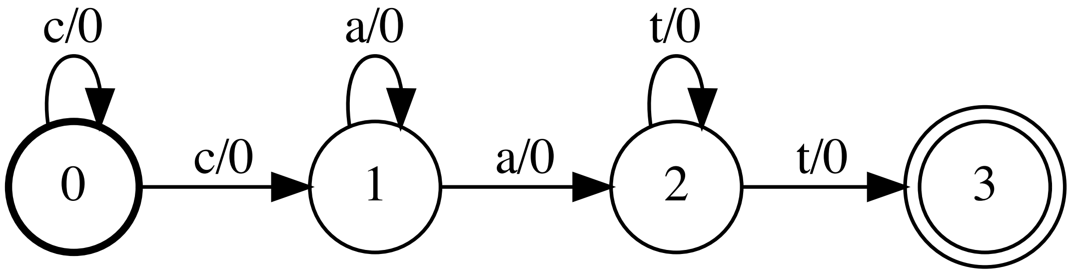
$$\log P(Y \mid X)$$

$$= \log[P(A_1 \mid X) + P(A_2 \mid X) + P(A_3 \mid X)]$$

$$= \text{actual-softmax}[\log P(A_1 \mid X), \log P(A_2 \mid X), \log P(A_3 \mid X)]$$

The CTC Loss

Aside: Alignment graph for $Y = [c, a, t]$



The CTC Loss

Problem: X has T frames and Y has U frames

If $T = 1000$ and $U = 100$ there are $\approx 6.4 \times 10^{139}$ alignments!

(For a fun combinatorics exercise show the exact number is $\binom{T-1}{U-1}$, Hint: "Stars and Bars.")

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

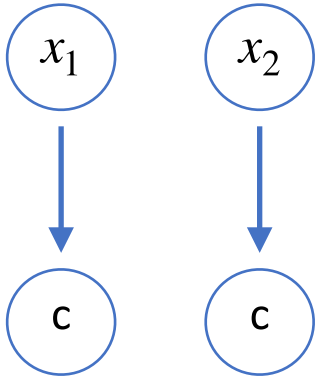
Forward variable: α_t^u the score for all alignments of length t which end in y_u.

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$

$$\alpha_2^c = \log P(c \mid x_1) + \log P(c \mid x_2)$$

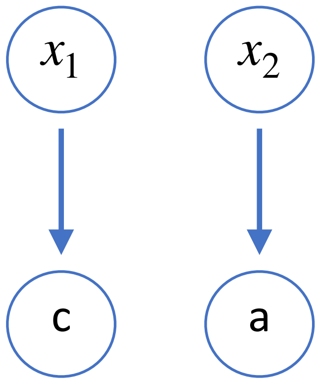


The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$

$$\alpha_2^a = \log P(c \mid x_1) + \log P(a \mid x_2)$$



The CTC Loss

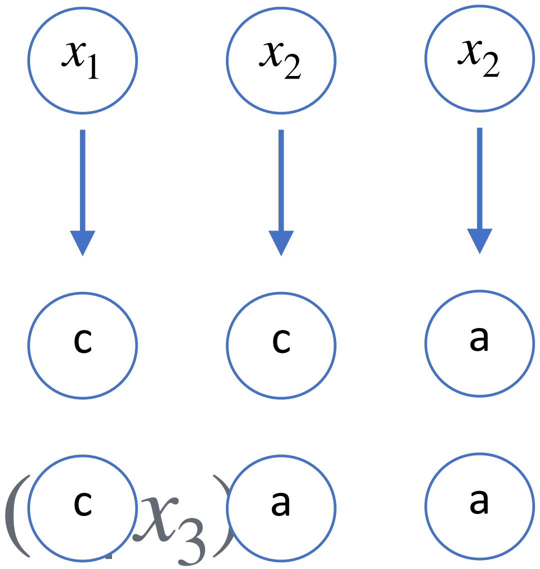
Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$

$\alpha_3^a =$ actual-softmax$[\log P(A_1), \log P(A_2)]$

$\log P(A_1) = \log P(c \mid x_1) + \log P(c \mid x_2) + \log P(c \mid x_3)$

$\log P(A_2) = \log P(c \mid x_1) + \log P(a \mid x_2) + \log P(a \mid x_3)$



The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$ α_2^c

$$\alpha_3^a = \text{actual-softmax}[\log P(A_1), \log P(A_2)]$$

$$\log P(A_1) = \log P(c \mid x_1) + \log P(c \mid x_2) + \log P(a \mid x_3)$$

$$\log P(A_2) = \log P(c \mid x_1) + \log P(a \mid x_2) + \log P(a \mid x_3)$$ α_2^a

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$

$\alpha_3^a =$ actual-softmax$[\log P(A_1), \log P(A_2)]$

$$\log P(A_1) = \alpha_2^c + \log P(a \mid x_3)$$

$$\log P(A_2) = \alpha_2^a + \log P(a \mid x_3)$$

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

Example: $X = [x_1, x_2, x_3, x_4], \quad Y = [c, a, t]$

$\alpha_3^a = \text{actual-softmax}[\log P(A_1), \log P(A_2)] = \text{actual-softmax}[\alpha_2^c, \alpha_2^a] + \log P(a \mid x_3)$

Exercise: prove this equality!

$$\log P(A_1) = \alpha_2^c + \log P(a \mid x_3)$$

$$\log P(A_2) = \alpha_2^a + \log P(a \mid x_3)$$

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

General recursion:

$$X = [x_1, x_2, x_3, \ldots, x_T], \quad Y = [y_1, y_2, \ldots, y_U]$$

$$\alpha_t^u = \text{actual-softmax}[\alpha_{t-1}^u, \alpha_{t-1}^{u-1}] + \log P(y_u \mid x_t)$$

The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)

General recursion:

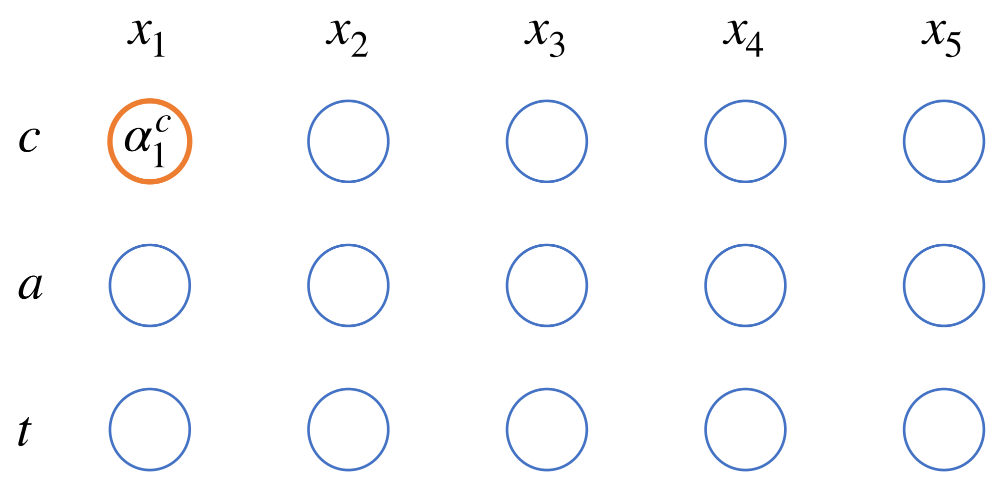
$$X = [x_1, x_2, x_3, \ldots, x_T], \quad Y = [y_1, y_2, \ldots, y_U]$$

$$\alpha_t^u = \text{actual-softmax}[\alpha_{t-1}^u, \alpha_{t-1}^{u-1}] + \log P(y_u \,|\, x_t)$$

Final score: $\log P(Y \,|\, X) = \alpha_T^U$

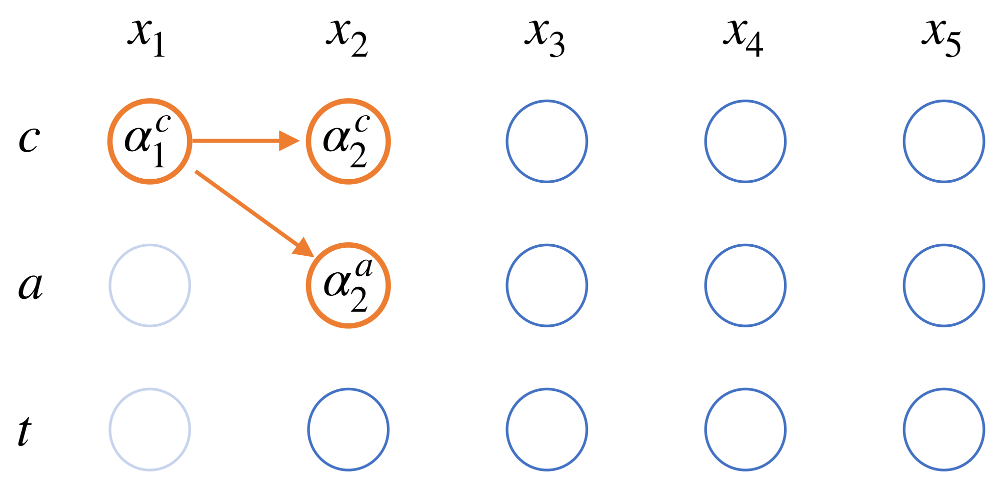
The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)



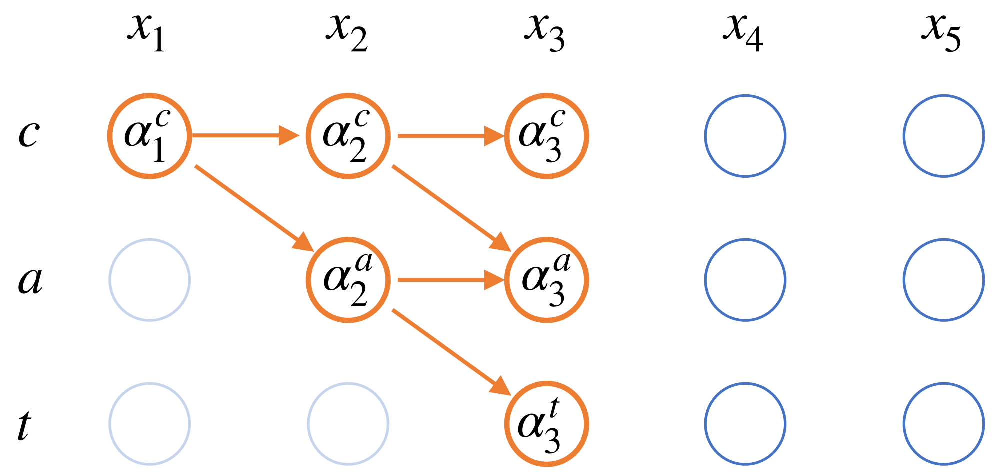
The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)



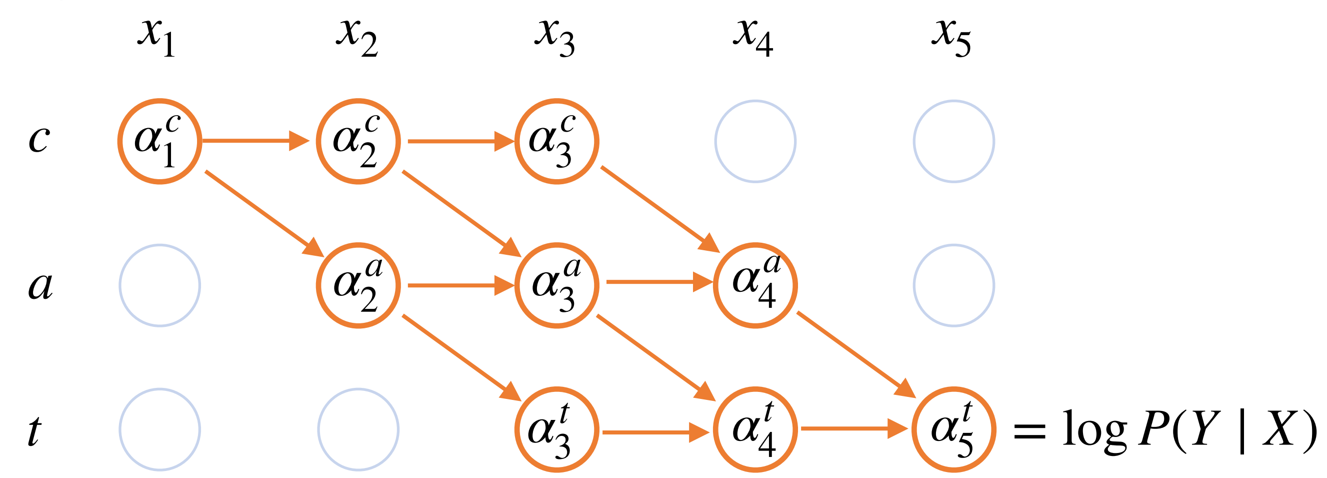
The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)



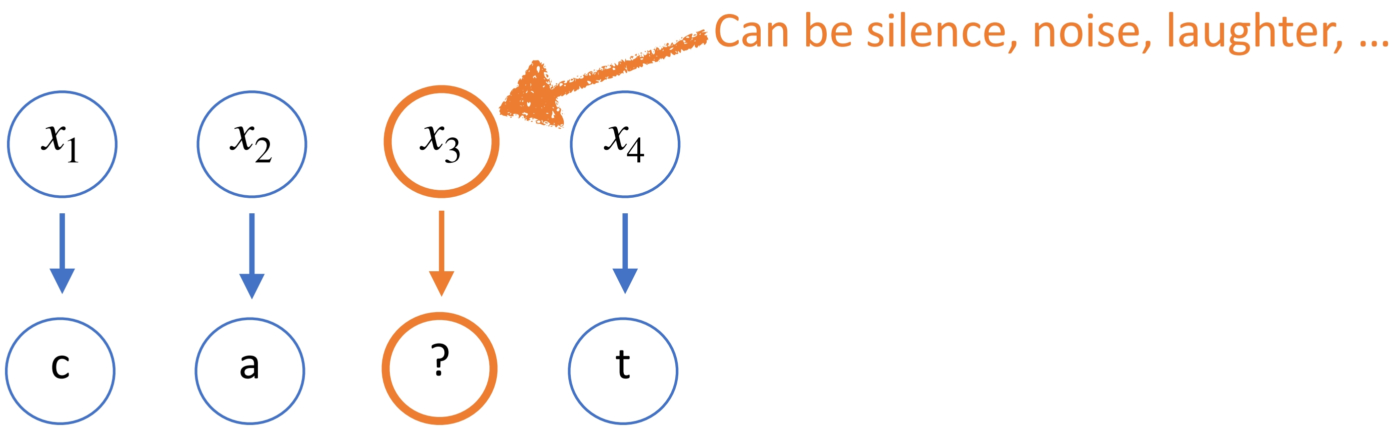
The CTC Loss

Solution: The Forward algorithm (A.K.A. dynamic programming)



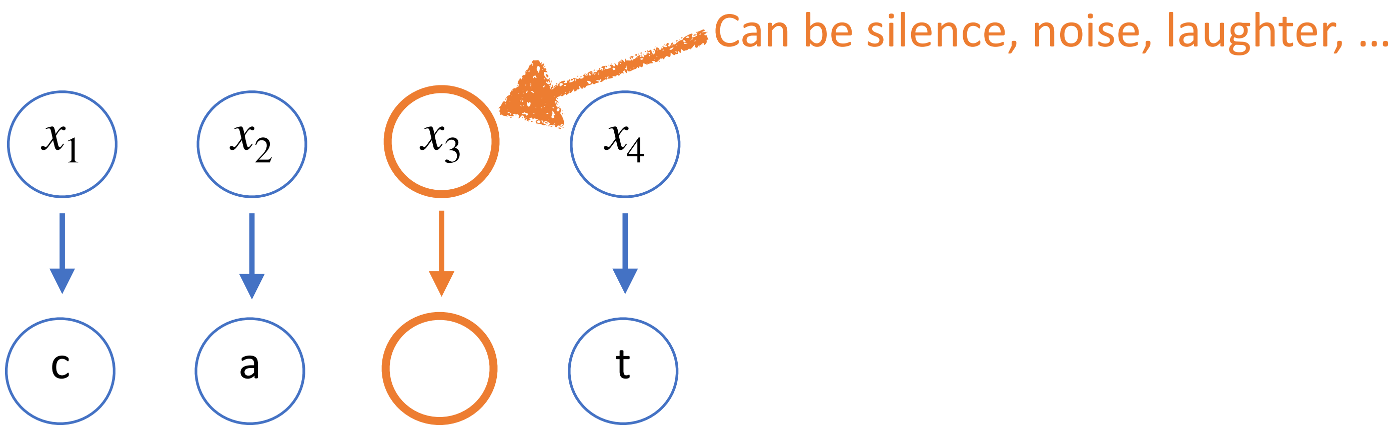
The CTC Loss

Problem: Not every input corresponds to "speech"



The CTC Loss

Solution: Use a “garbage” or *blank* token: <b>

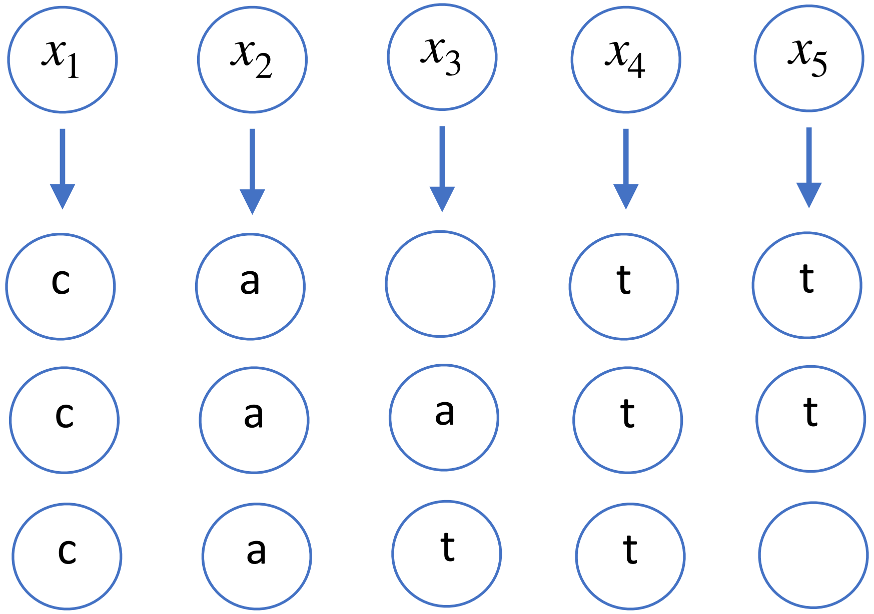


The CTC Loss

Solution: Use a “garbage” or *blank* token: <b>

Blank token is optional

Some allowed alignments:

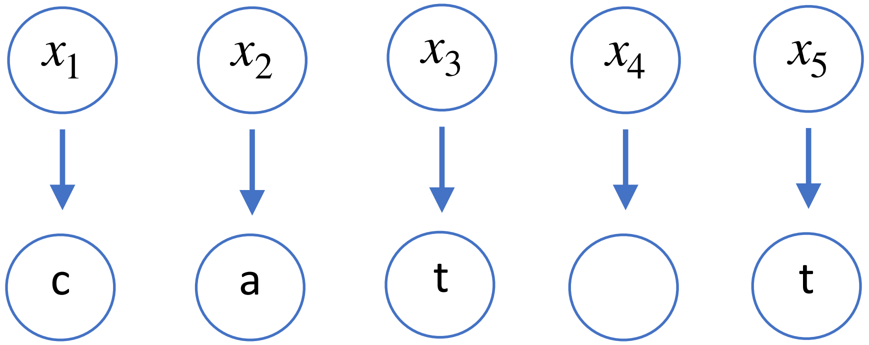


The CTC Loss

Solution: Use a "garbage" or *blank* token: <b>

Blank token is optional

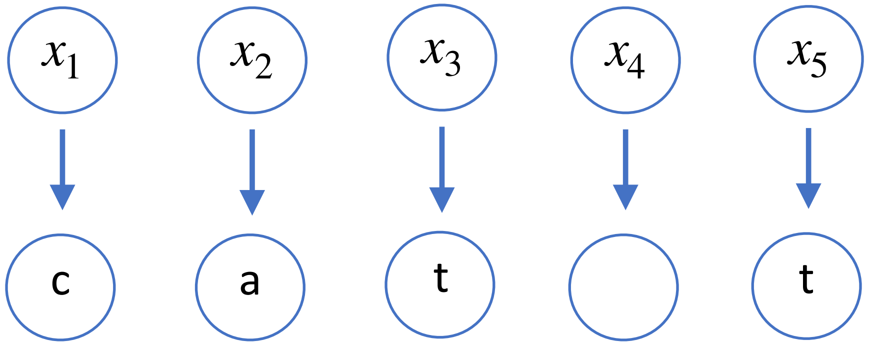
Allowed?



The CTC Loss

Solution: Use a “garbage” or *blank* token: <b>

Blank token is optional



No!

Corresponds to “catt”.

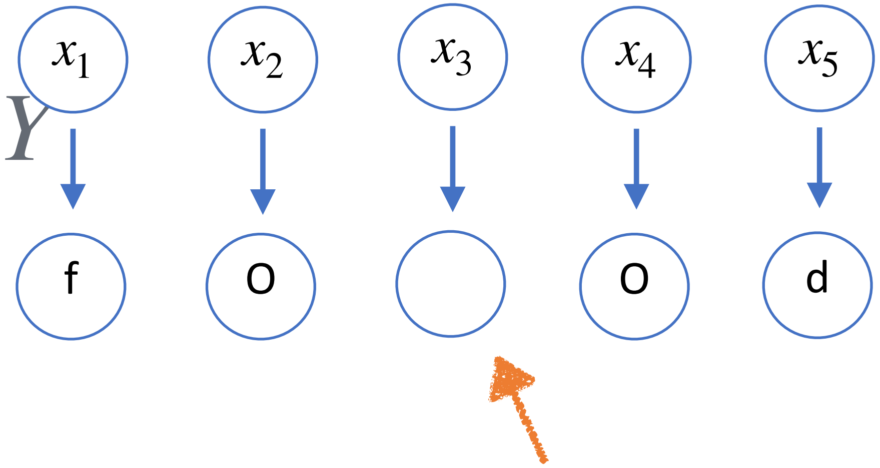
The CTC Loss

Solution: Use a “garbage” or *blank* token: <b>

Blank token is optional …

except between repeats in Y

$$Y = [f, o, o, d]$$

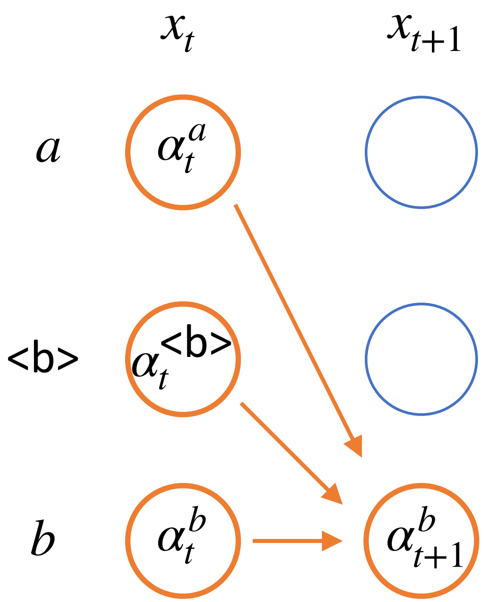


Not optional!

The CTC Loss

CTC Recursion: Three cases

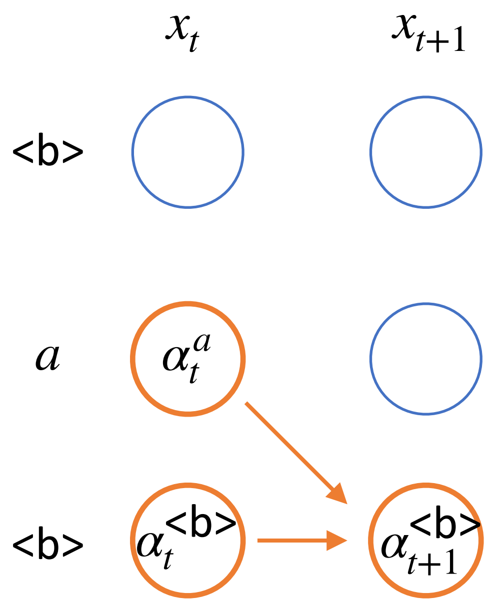
Case 1: Blank is optional



The CTC Loss

CTC Recursion: Three cases

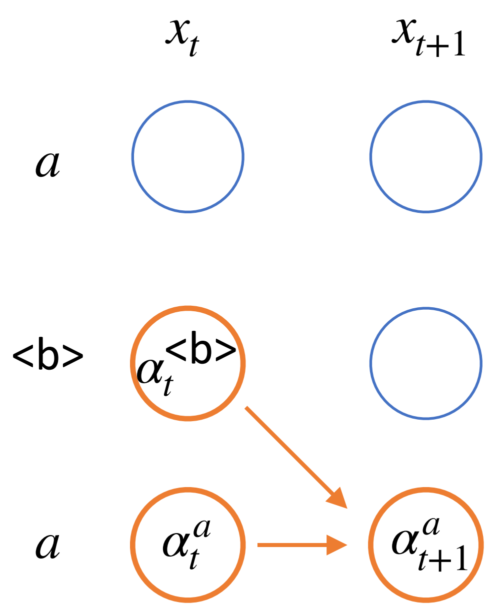
Case 2: Output is not optional



The CTC Loss

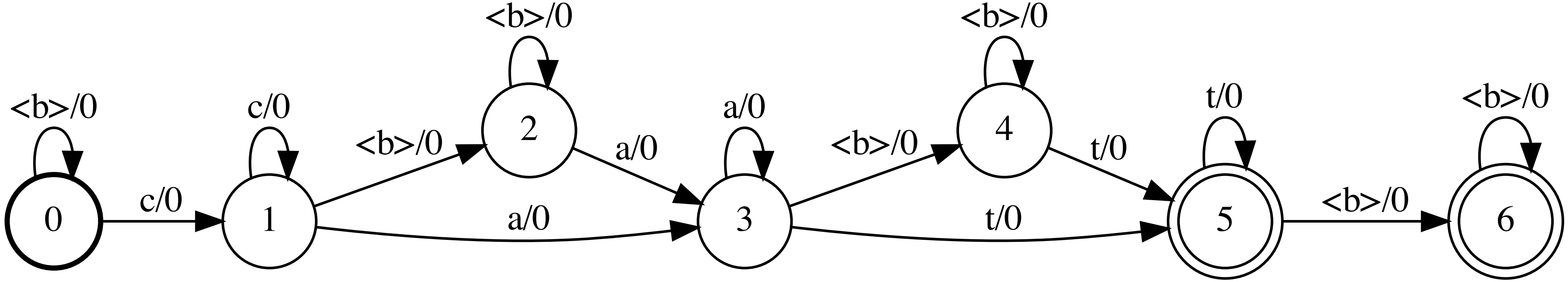
CTC Recursion: Three cases

Case 3: Repeats,
blank is not optional



The CTC Loss

Aside: The CTC graph



Outline

- Modern Speech Recognition
- Deep Dive: The CTC Loss
- Deep Dive: Decoding with Beam Search
- Graph Transformer Networks

Inference

Goal: Find the best Y (transcription) given an X (speech)

We have two models:

1. Acoustic model: $\log P(Y \mid X)$

2. Language model: $\log P(Y)$

Inference

Language Model: $\log P(Y)$

1. Trained on much larger text corpus
2. Fine-tuned for given application (or even user!)
3. Typically word-level *n*-gram with *n* between three and five

Inference

Goal: Find the best Y (transcription) given an X (speech)

We have two models:

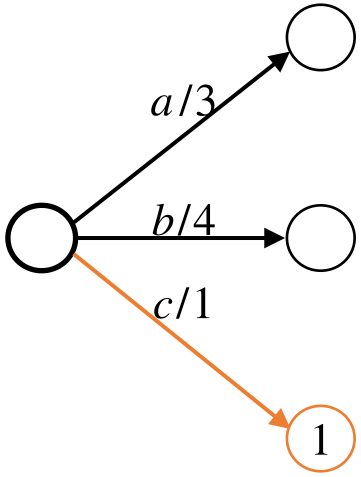
1. Acoustic model: $\log P(Y \mid X)$
2. Language model: $\log P(Y)$

Find:

$$Y^* = \text{argmax}_Y \quad \log P(Y \mid X) + \log P(Y)$$

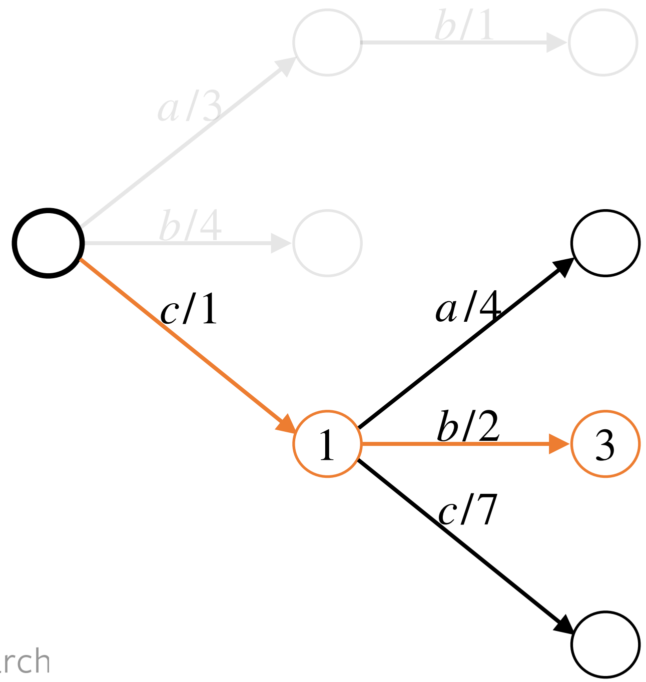
Graph Shortest Path: Greedy

Goal: Find the best (lowest scoring) path in the graph



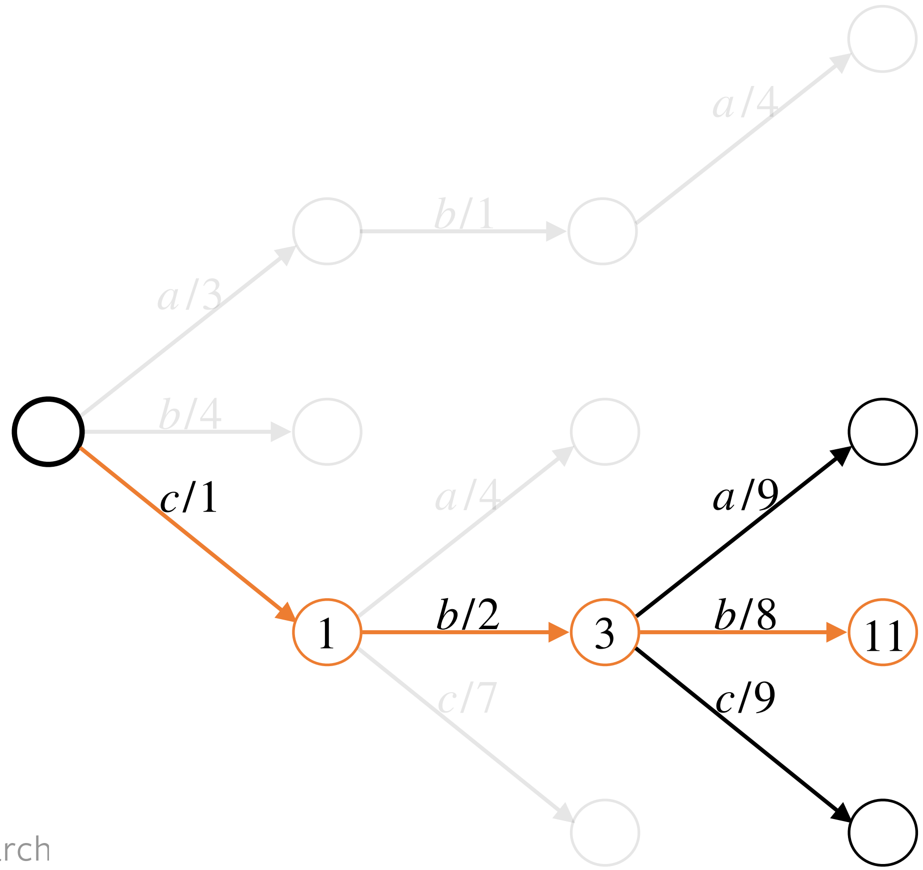
Graph Shortest Path: Greedy

Goal: Find the best (lowest scoring) path in the graph



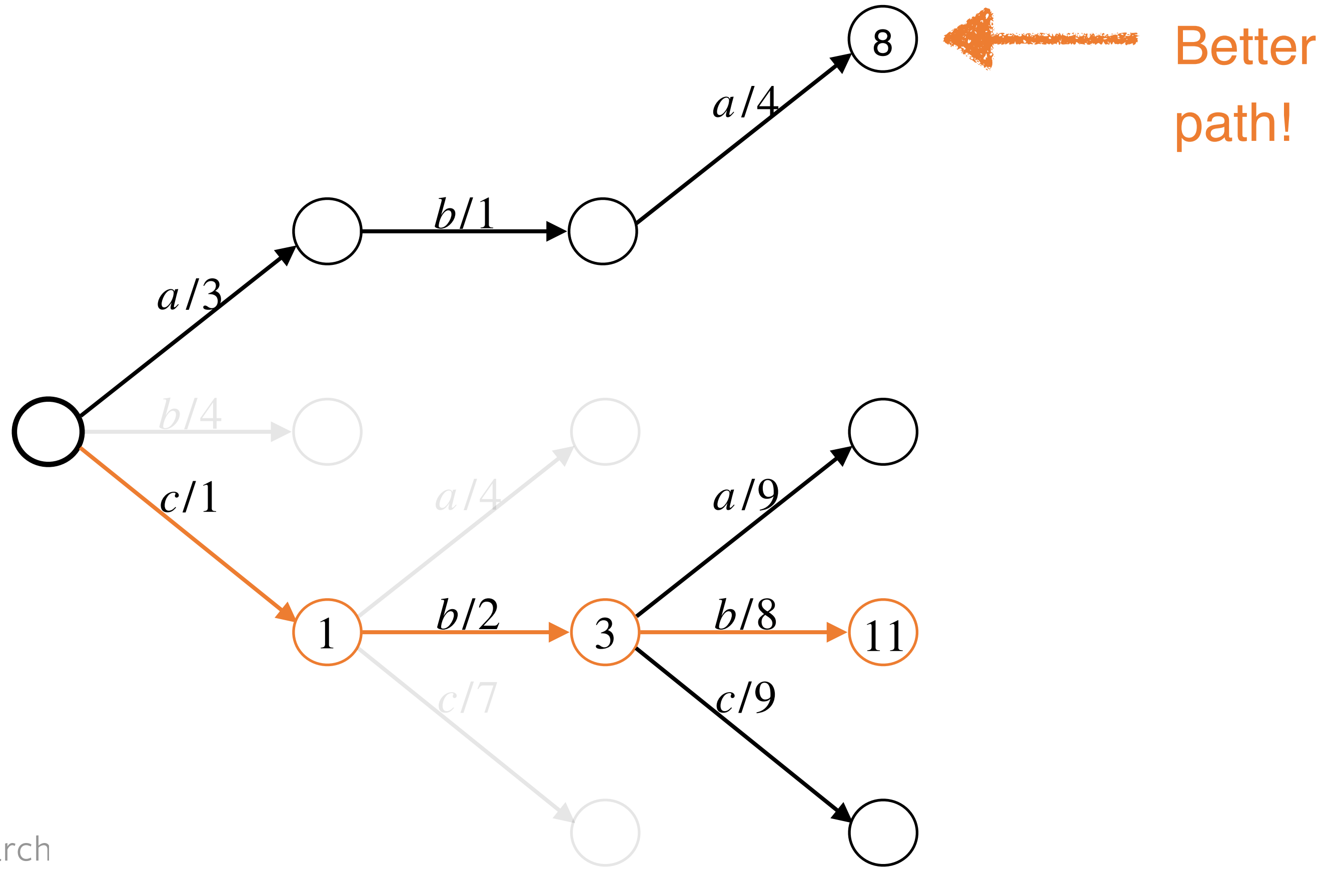
Graph Shortest Path: Greedy

Goal: Find the best (lowest scoring) path in the graph



Graph Shortest Path: Greedy

Goal: Find the best (lowest scoring) path in the graph



Graph Shortest Path: Beam Search

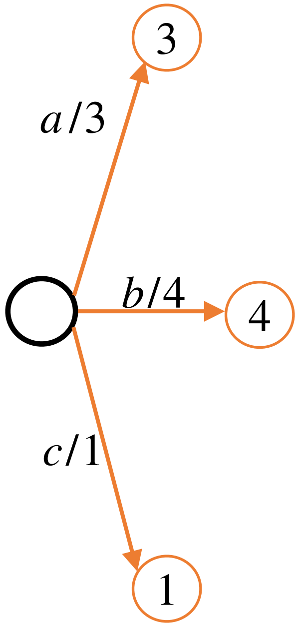
Algorithm:

Repeat:

1. Extend current candidates by all possibilities
2. Sort by score and keep N best

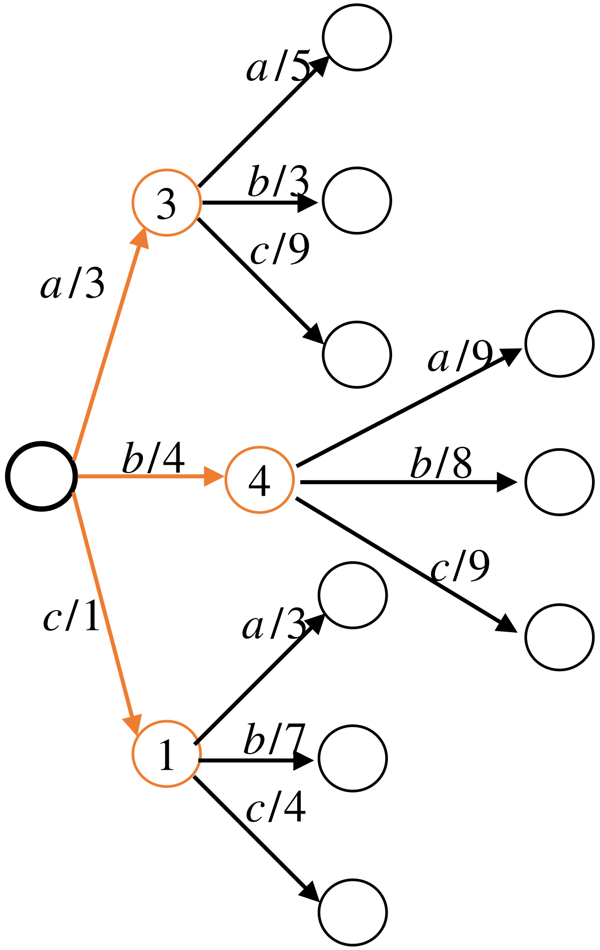
Graph Shortest Path: Beam Search

N = 3



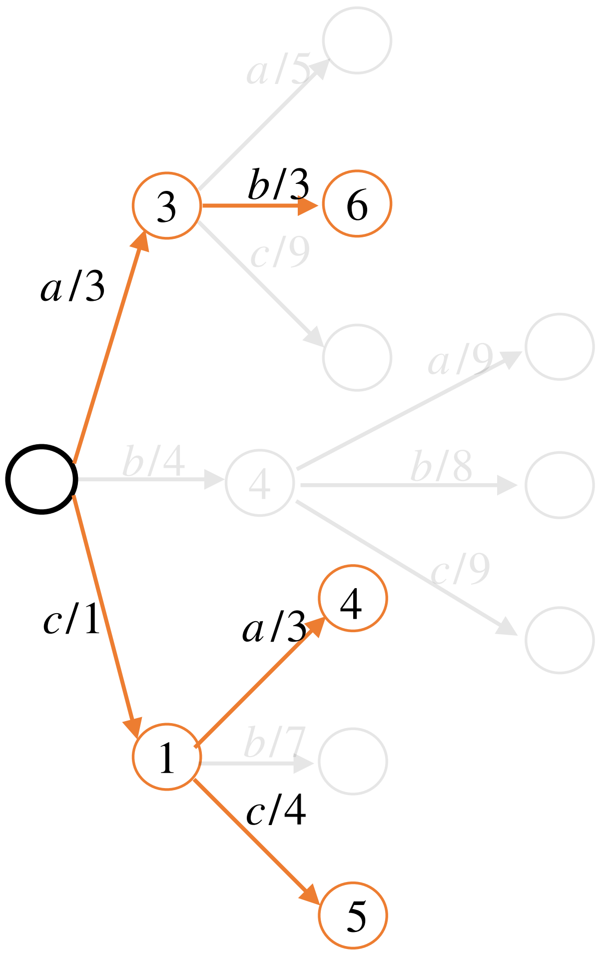
Graph Shortest Path: Beam Search

N = 3



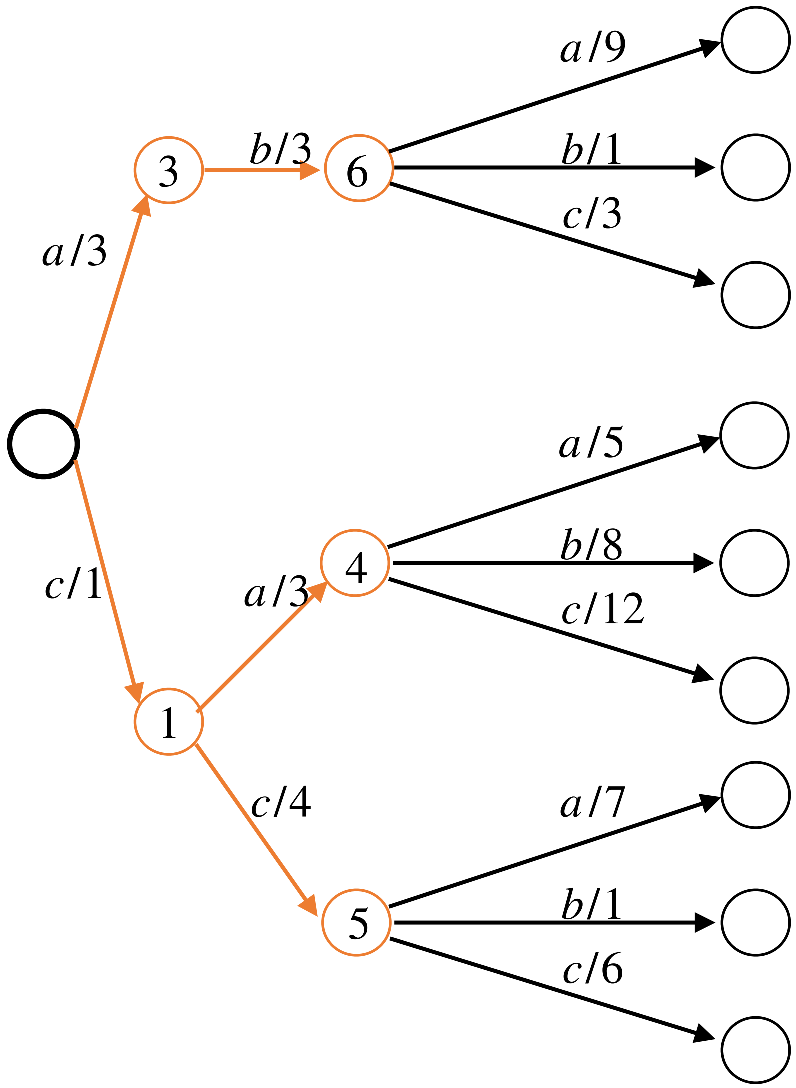
Graph Shortest Path: Beam Search

N = 3



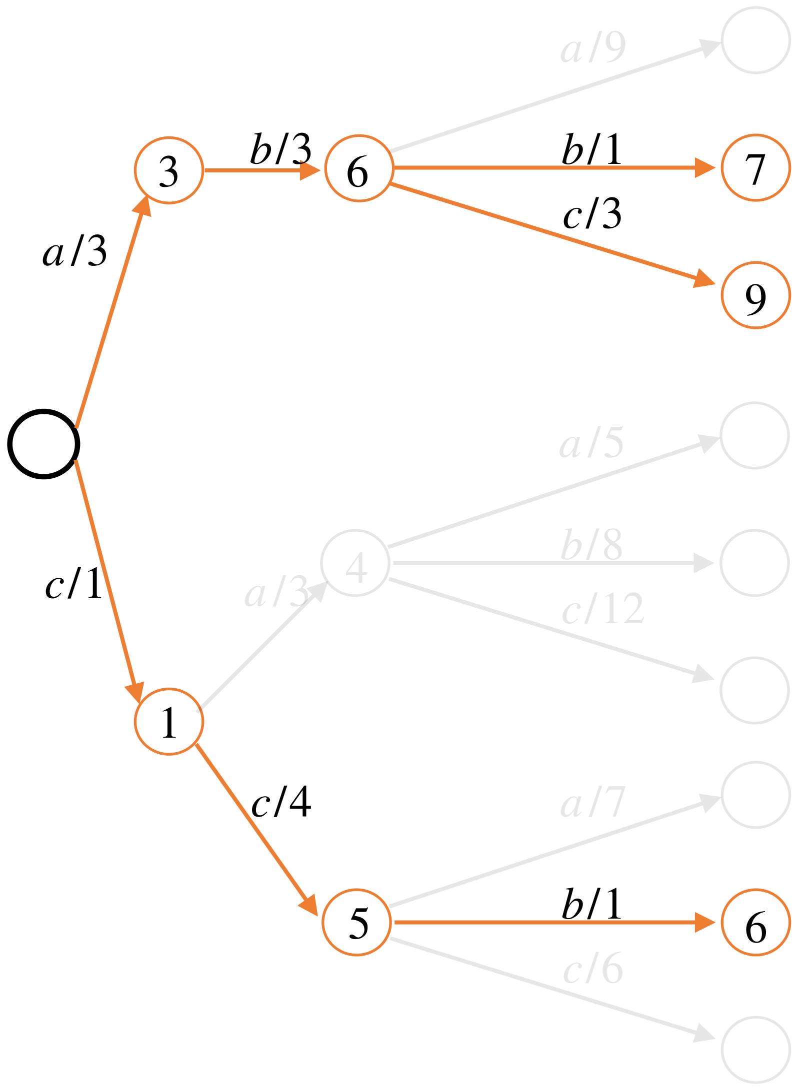
Graph Shortest Path: Beam Search

N = 3



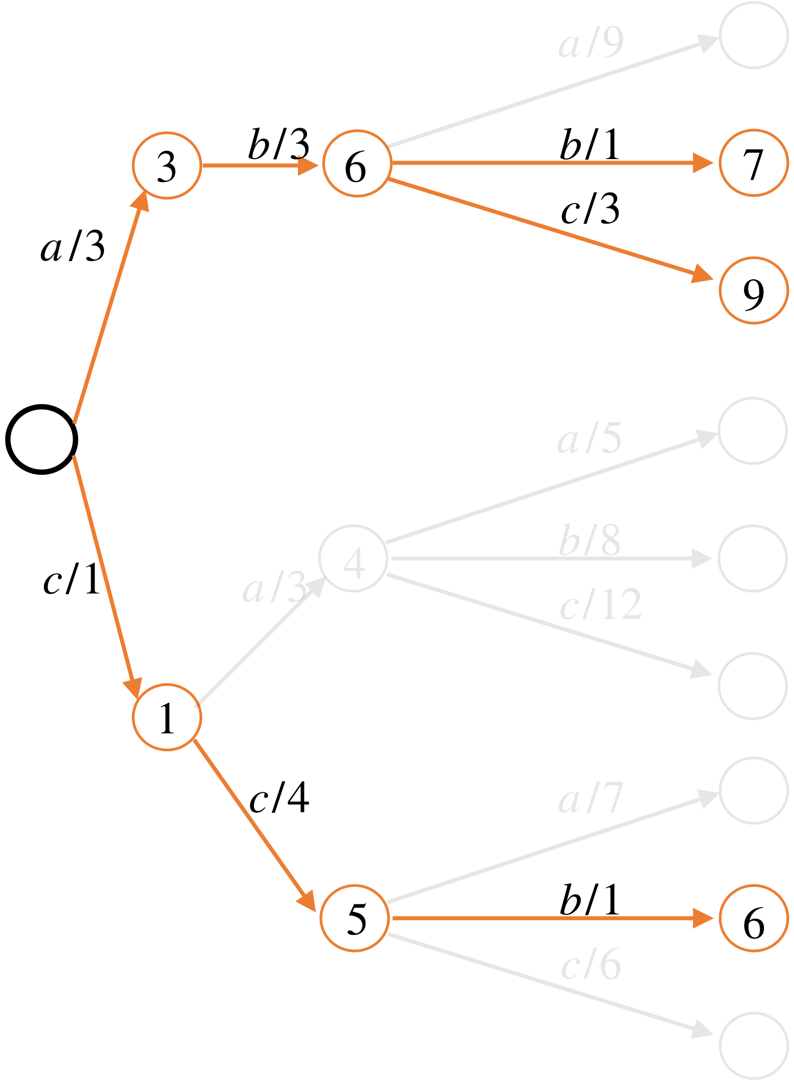
Graph Shortest Path: Beam Search

N = 3



Graph Shortest Path: Beam Search

N = 3



Return N-best list:

```
[c, c, b], score=6
[a, b, b], score=7
[a, b, c], score=9
```

Inference

Goal: Find the best Y (transcription) given an X (speech)

Use beam search to find

$$Y^* \approx \text{argmax}_Y \quad \log P(Y \mid X) + \log P(Y)$$

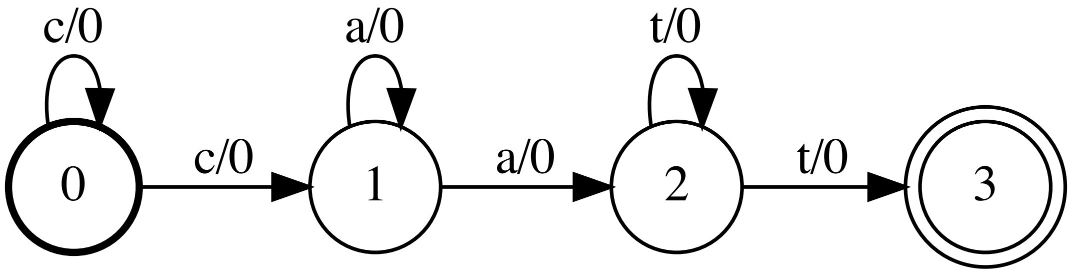
Outline

- Modern Speech Recognition
- Deep Dive: The CTC Loss
- Deep Dive: Decoding with Beam Search
- Graph Transformer Networks

Weighted Finite State Automata (WFSA)

Remember: Alignment graph for $Y = [c, a, t]$

GTN: WFSAs with automatic differentiation.



Graph Transformer Networks (GTNs): History

- Developed by Bottou, Le Cun, et al. at AT&T in the early 90s
- First used in a state-of-the-art automatic check-reading system

Graph Transformer Networks (GTNs): History

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

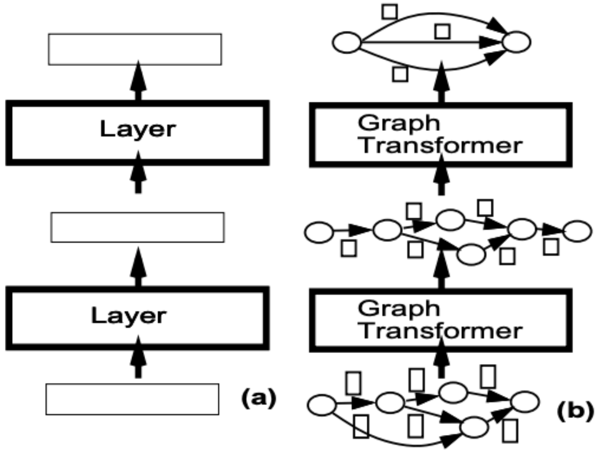


Fig. 15. Traditional neural networks, and multi-module systems communicate fixed-size vectors between layer. Multi-Layer Graph Transformer Networks are composed of trainable modules that operate on and produce graphs whose arcs carry numerical information.

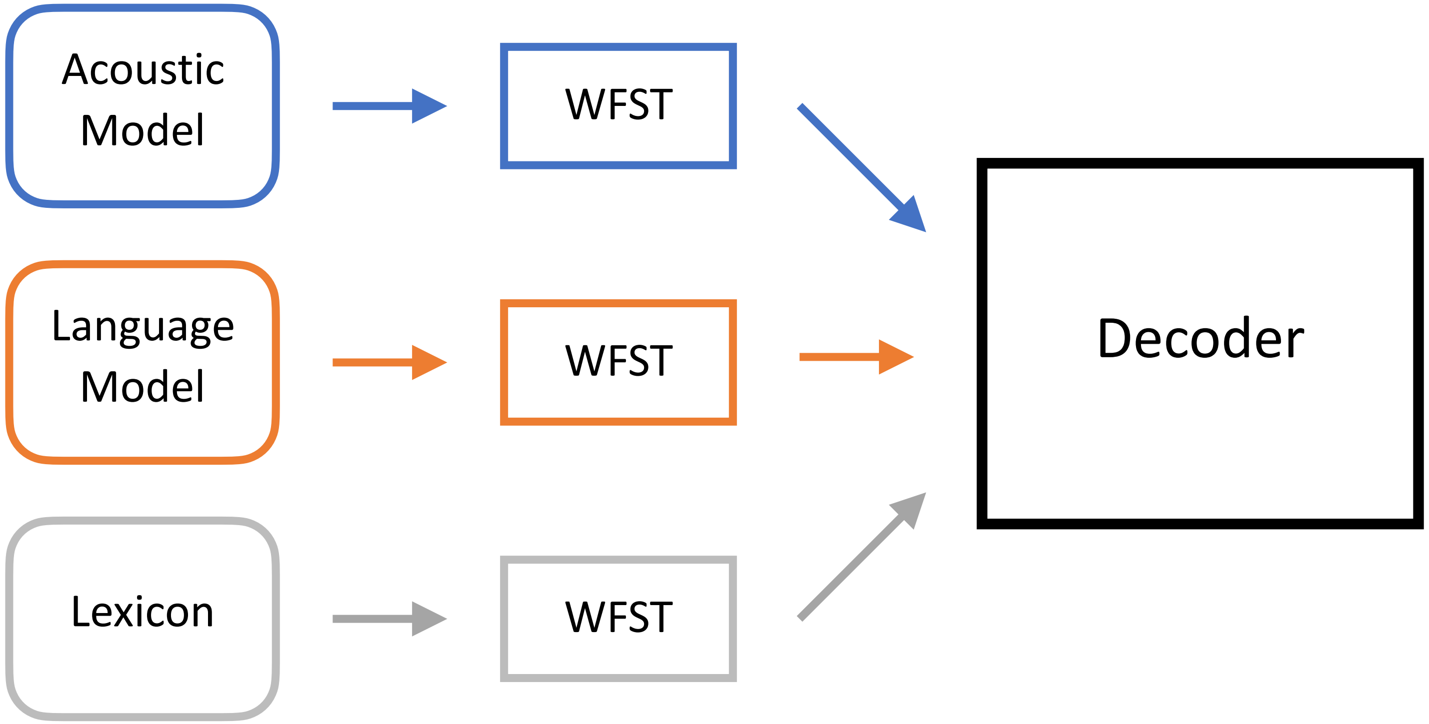
For deep learning:
see pages 1-16

For GTNs:
see pages 16-42

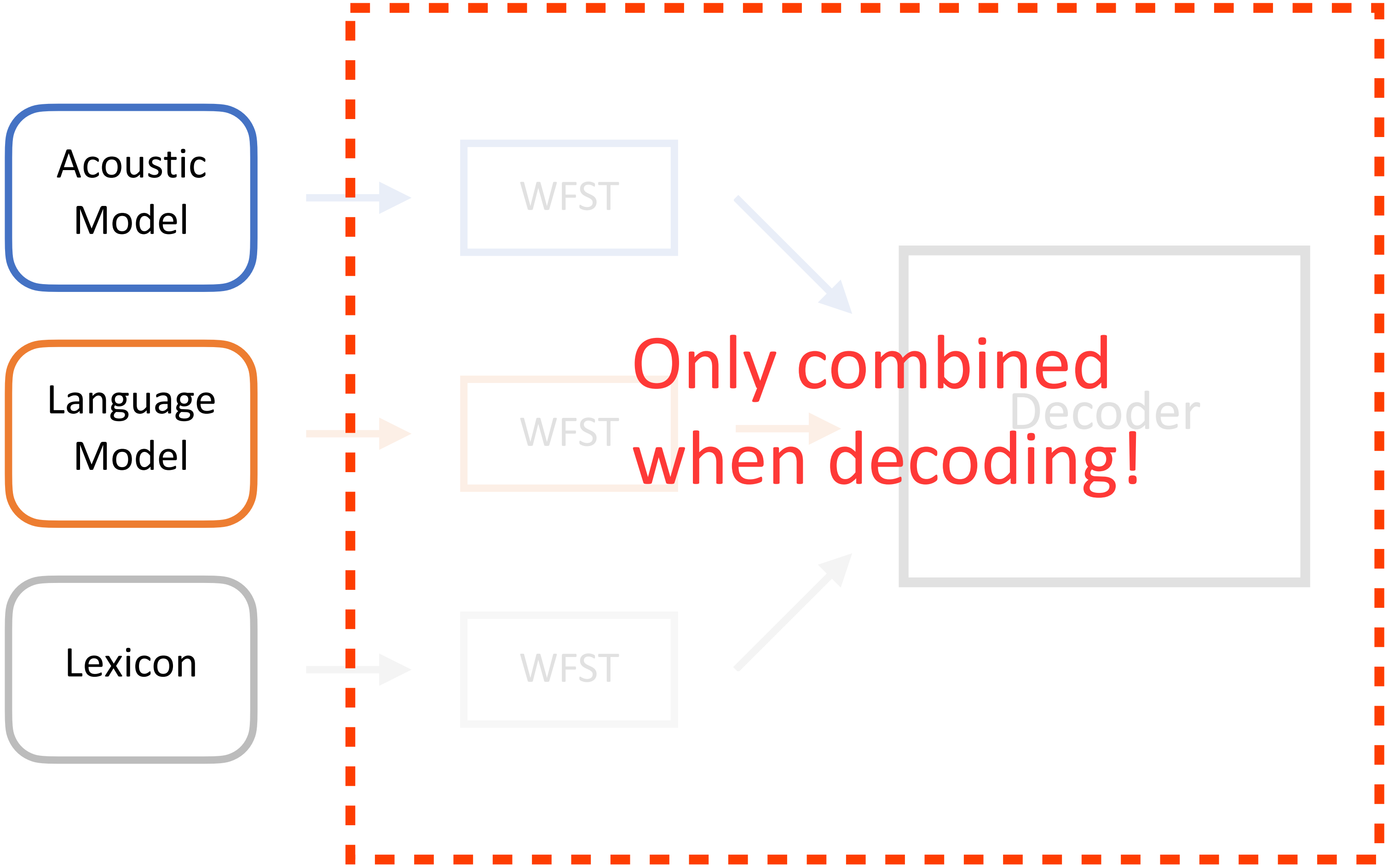
W

	Neural Networks	GTNs
Core data structure	Tensor	Graph (WFSA)
Core operations	Matrix multiplication Reduction operations (sum, prod, …) Unary and binary operations (negate, add, subtract, …)	Compose Shortest distance ops (forward, viterbi) Unary and binary operations (closure, union, concatenate, …)

Example: WFSTs in Speech Recognition



Example: WFSTs in Speech Recognition



Why Differentiable WFSAs?

- **Encode Priors:** Conveniently encode prior knowledge into a WFST
- **End-to-end:** Use at training time avoids issues such as label bias, exposure bias
- **Facilitate Research:** Separate data (graph) from code (operations on graphs)!

Sequence Criteria with WFSAs

Many loss functions are the difference of two WFSTs

The graph `A` is a function of the input X (e.g. speech) and target Y (e.g. transcription)

The graph `Z` is a function of only the input X

The loss is given by:

$$\log P(Y \mid X) = \texttt{forwardScore}(A_{X,Y}) - \texttt{forwardScore}(Z_X)$$

Sequence Criteria with WFSTs

Many criteria are the difference of two WFSTs

Includes common loss functions in ASR such:

- Automatic Segmentation Criterion (ASG)
- Connectionist Temporal Classification (CTC)
- Lattice Free MMI (LF-MMI)

Sequence Criteria with WFSTs

Lines of code for CTC: Custom vs GTN

	Lines of Code
Warp-CTC	9,742
wav2letter	2,859
PyTorch	1,161
GTN	30

Sequence Criteria with WFSTs

Lines of code for CTC: Custom vs GTN

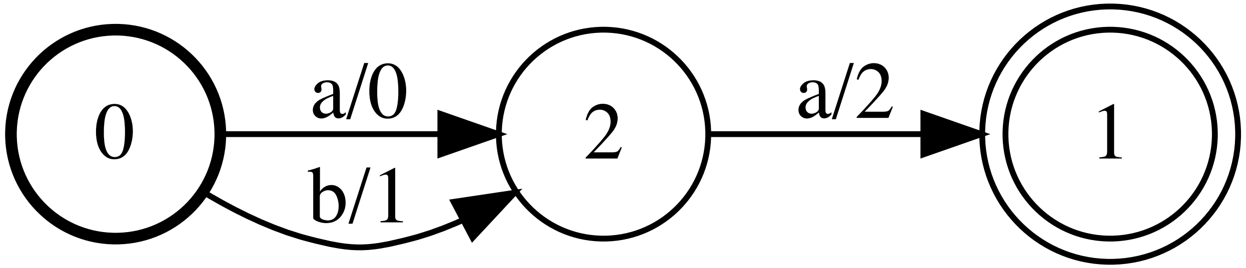
	Lines of Code
Warp-CTC	9,742
wav2letter	2,859
PyTorch	1,161
GTN	30

Same graphs work for decoding!

Weighted Finite-State Acceptor (WFSA)

A simple WFSA which recognizes `aa` or `ba`

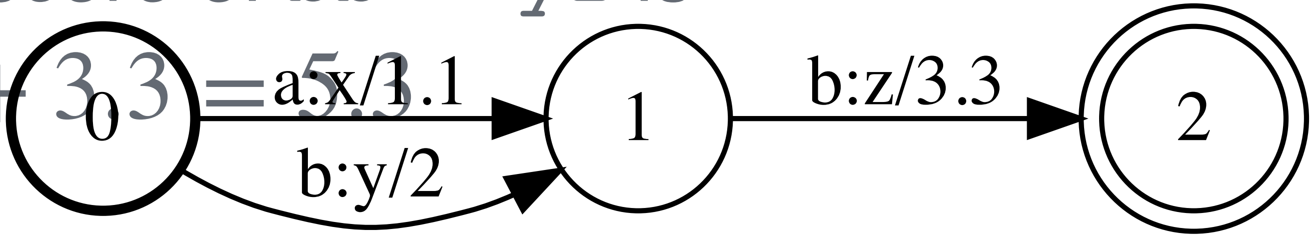
- The score of `aa` is $0 + 2 = 2$
- The score of `ba` is $1 + 2 = 3$



Weighted Finite-State Transducer (WFST)

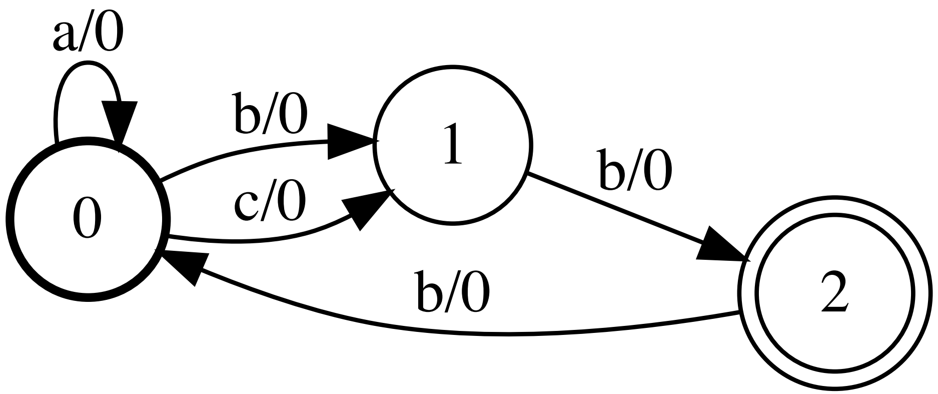
A simple WFST which transduces `ab` to `xz` and `bb` to `yz`.

- The score of `ab` $\rightarrow$ `xz` is $1.1 + 3.3 = 4.4$
- The score of `bb` $\rightarrow$ `yz` is $2.0 + 3.3 = 5.3$



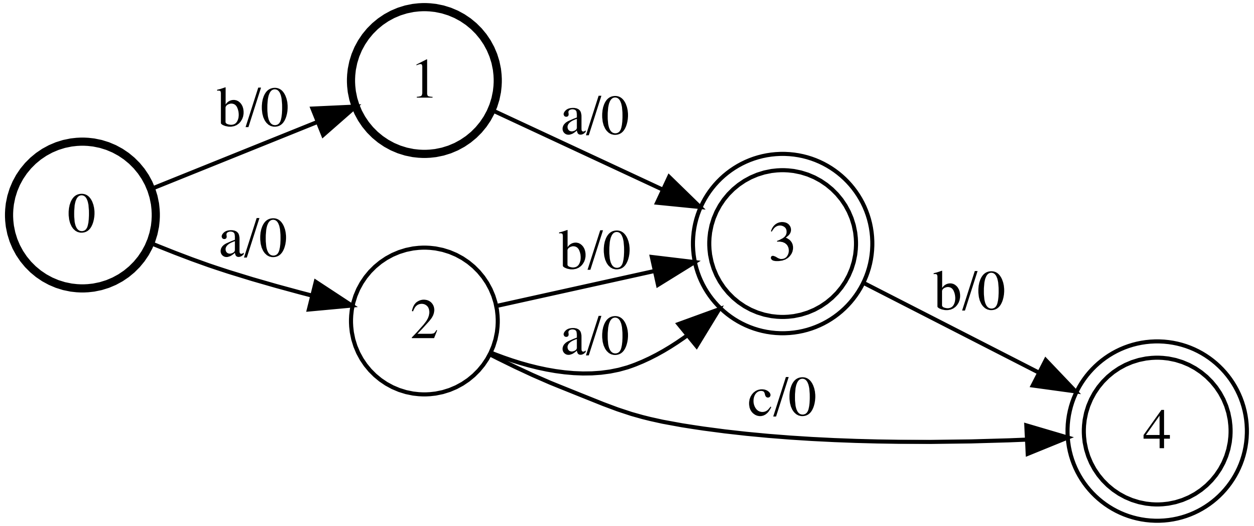
More WFSAs and WFSTs

Cycles and self-loops are allowed



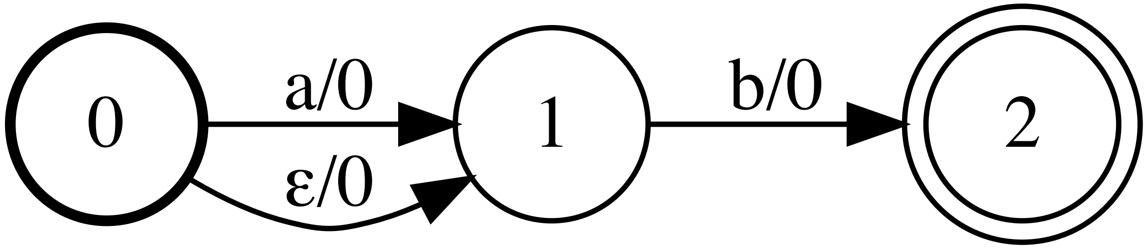
More WFSAs and WFSTs

Multiple start and accept nodes are allowed



More WFSAs and WFSTs

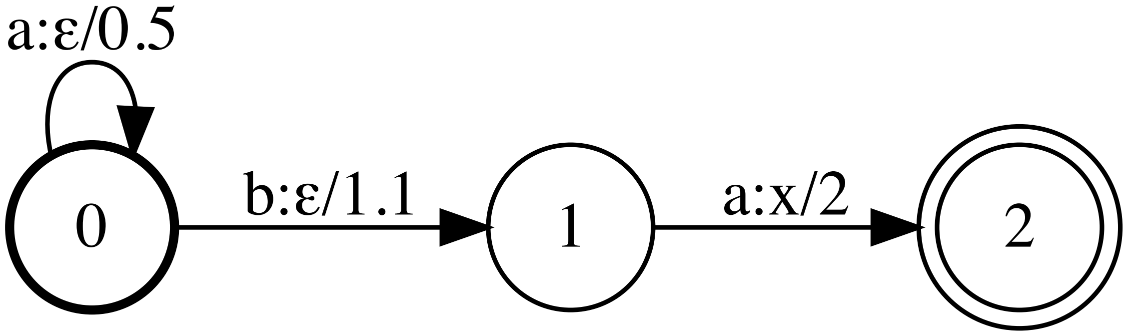
ϵ transitions are allowed in WFSAs



More WFSAs and WFSTs

ϵ transitions are allowed in WFSTs

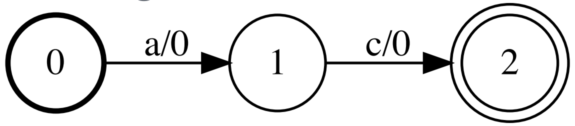
- The score of `aba`$\rightarrow$`x` is 3.6



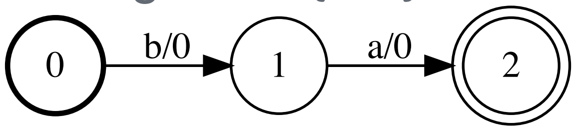
Operations: Union

The union accepts a sequence if it is accepted by any of the input graphs.

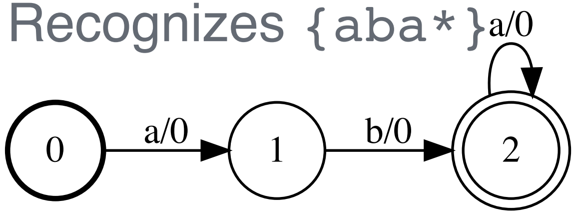
Recognizes `{ac}`



Recognizes `{ba}`

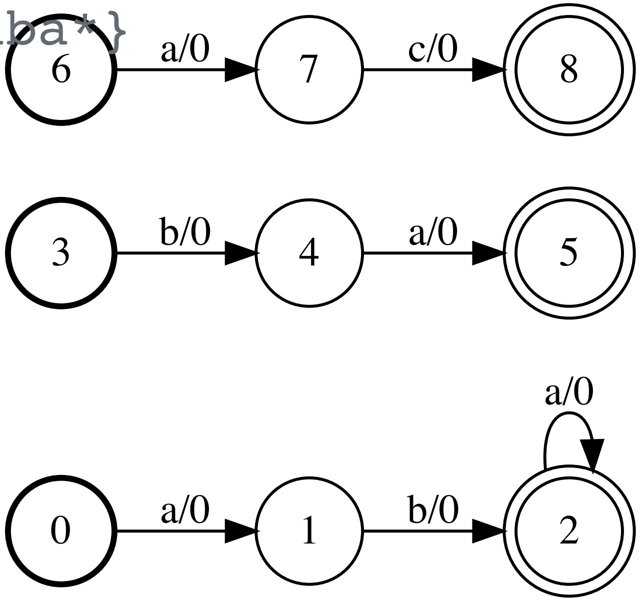


Recognizes `{aba*}`



`union({g1, g2, g3})`→

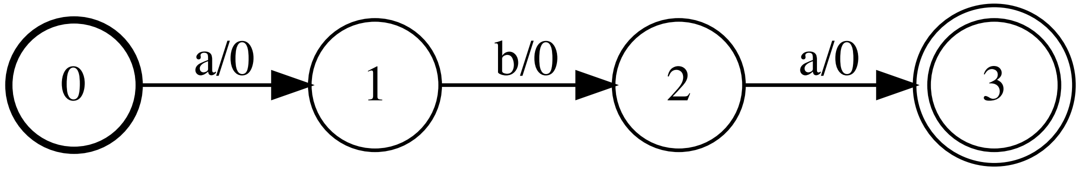
Recognizes `{ac, ba, aba*}`



Operations: Kleene Closure

Accepts any sequence accepted by the input graph repeated 0 or more times.

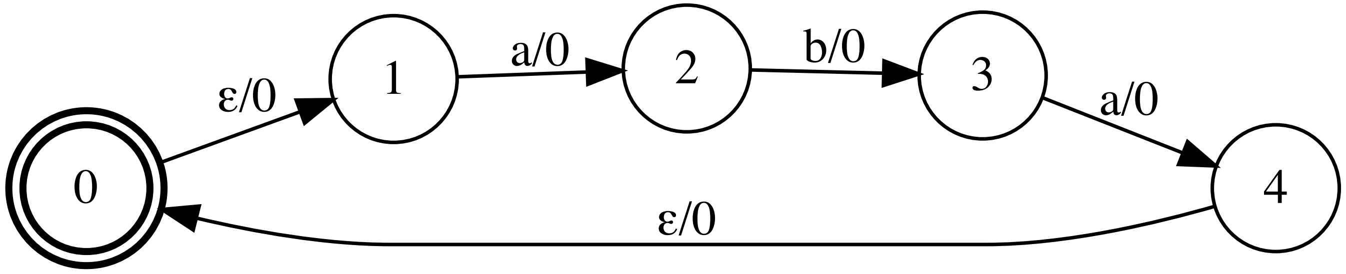
Recognizes `{aba}`



`closure(g)`

↓

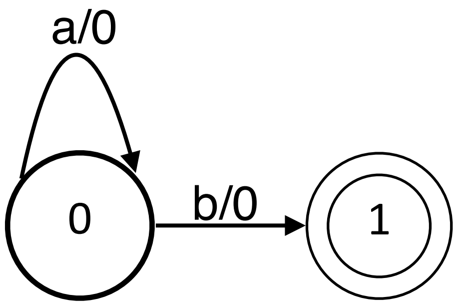
Recognizes $\{\epsilon$, `aba, abaaba, ...}`

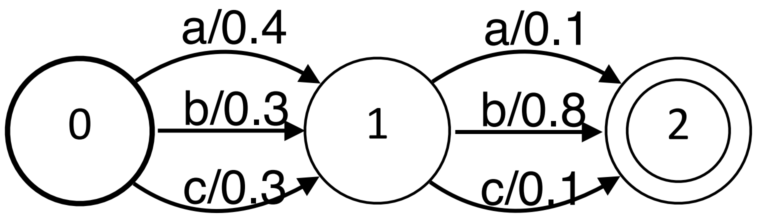


Operations: Intersect

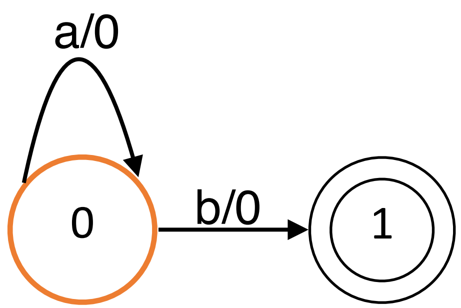
1. Any path accepted by both WFSAs is accepted by the intersection.
2. The score of the path in the intersected graph is the sum of the scores of the paths in the input graphs.

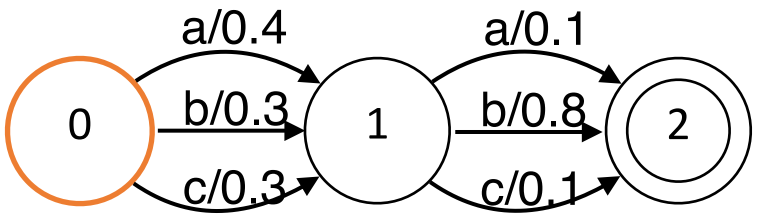
Operations: Intersect





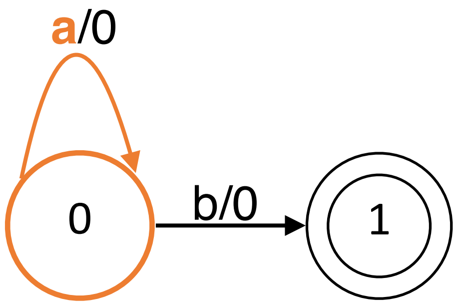
Operations: Intersect

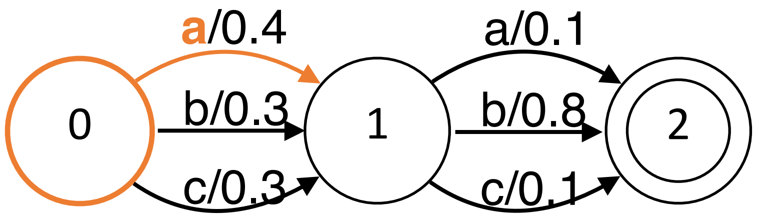




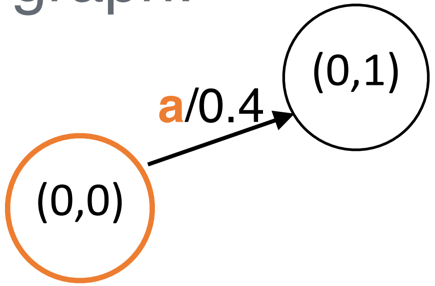
Intersected graph:

Operations: Intersect

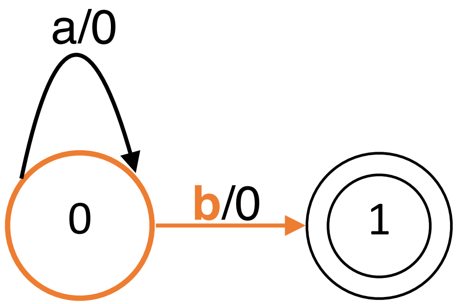


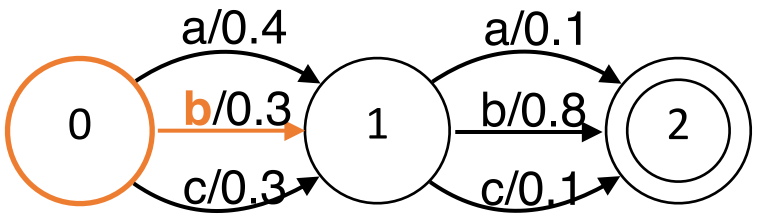


Intersected graph:

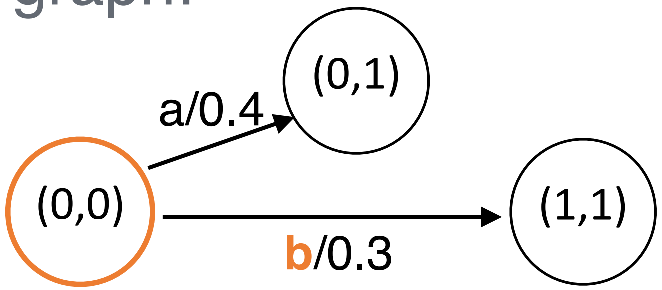


Operations: Intersect

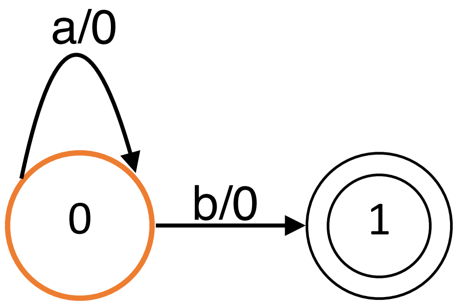


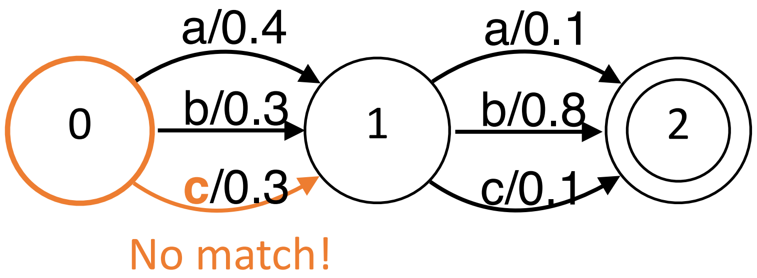


Intersected graph:

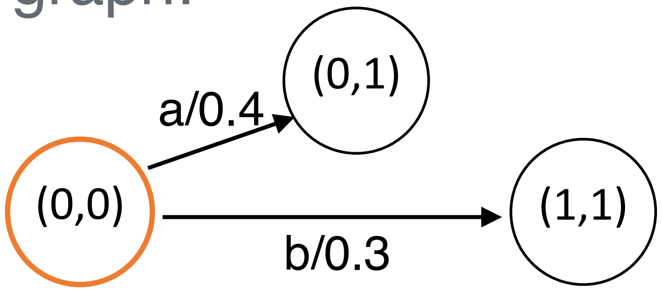


Operations: Intersect

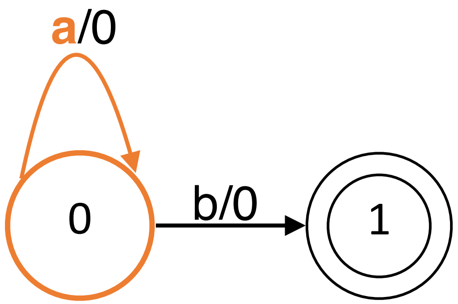


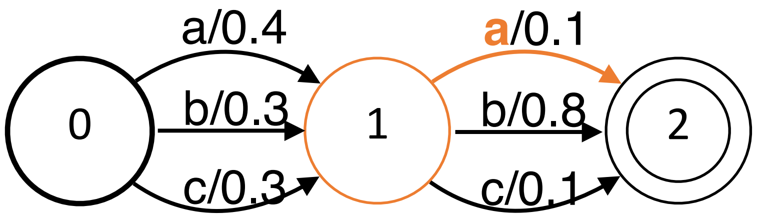


Intersected graph:

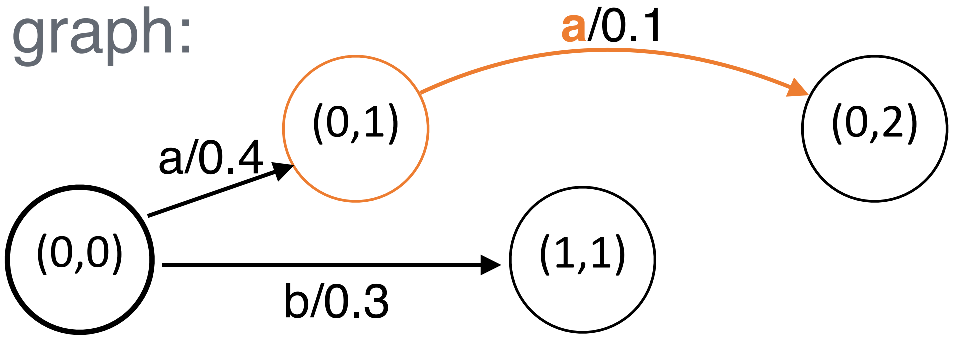


Operations: Intersect

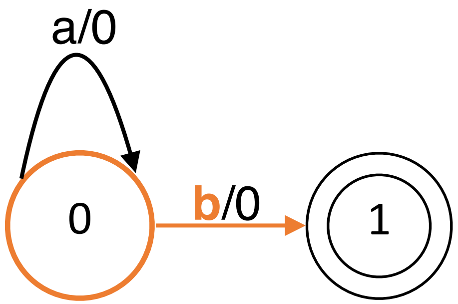


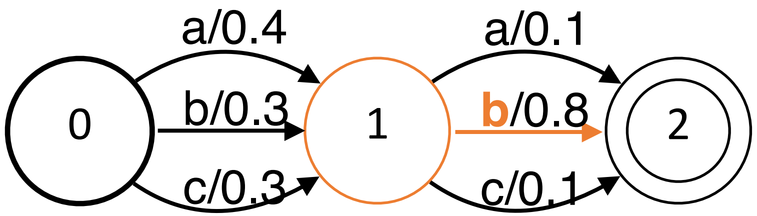


Intersected graph:

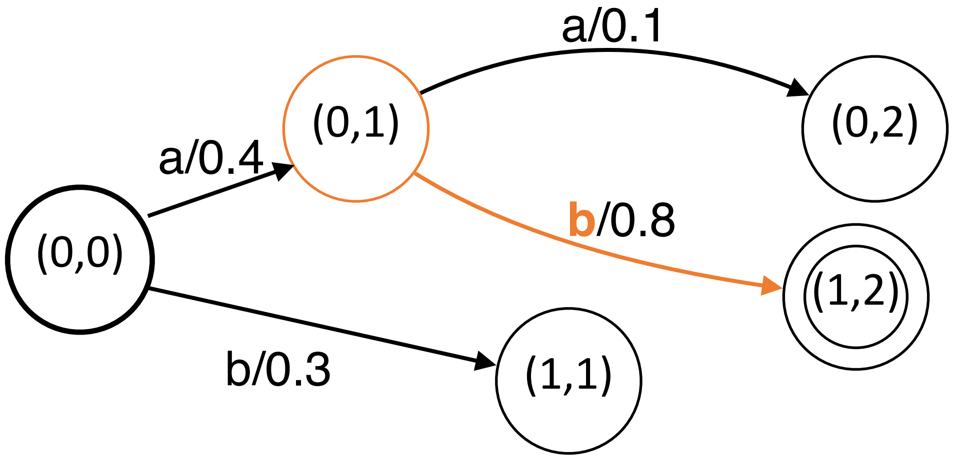


Operations: Intersect

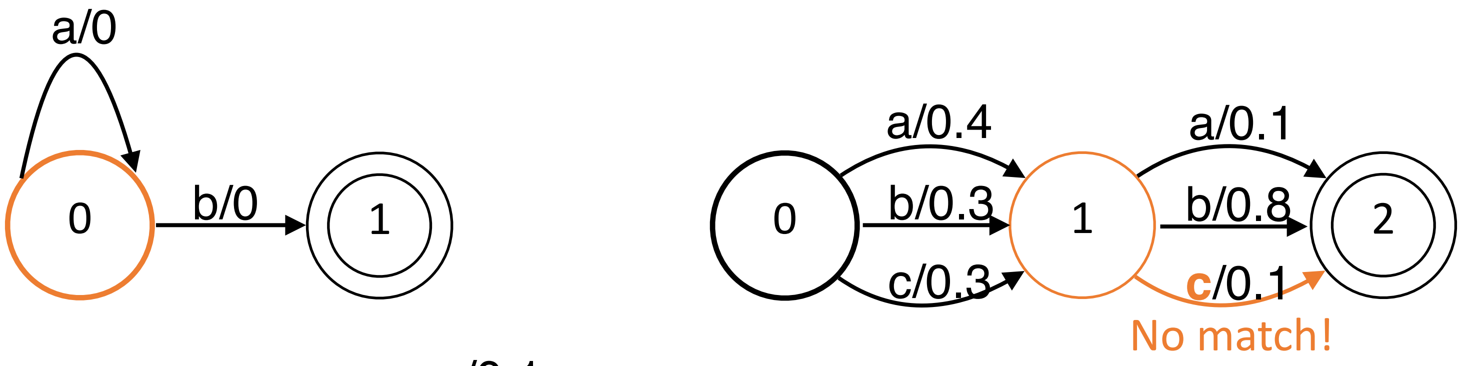




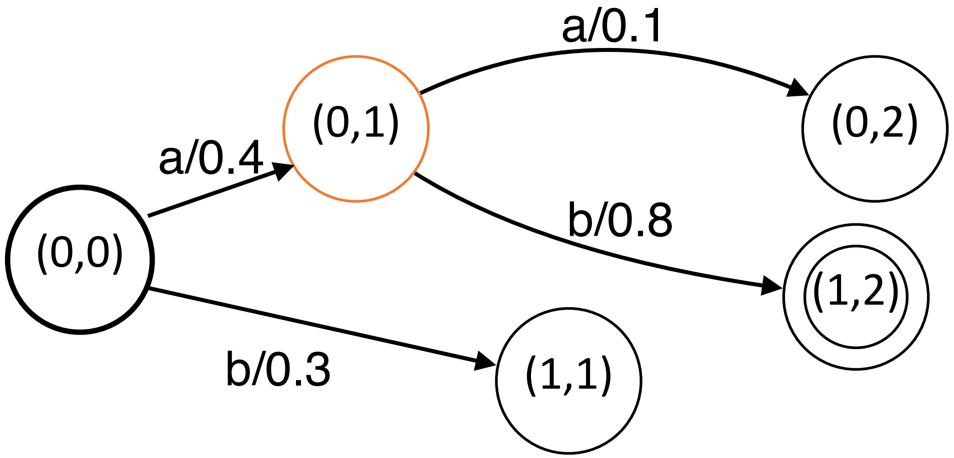
Intersected graph:



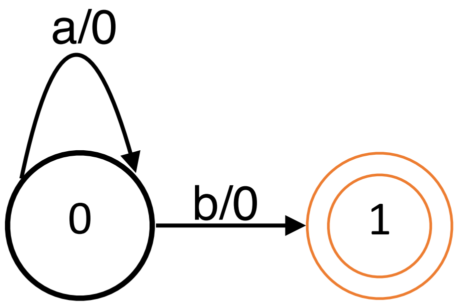
Operations: Intersect

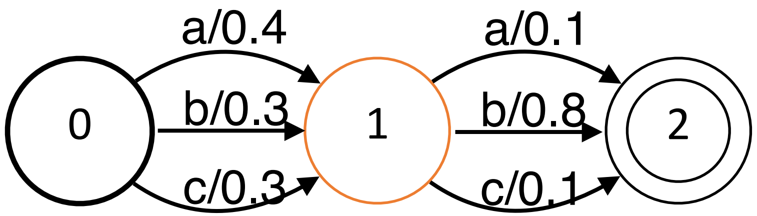


Intersected graph:

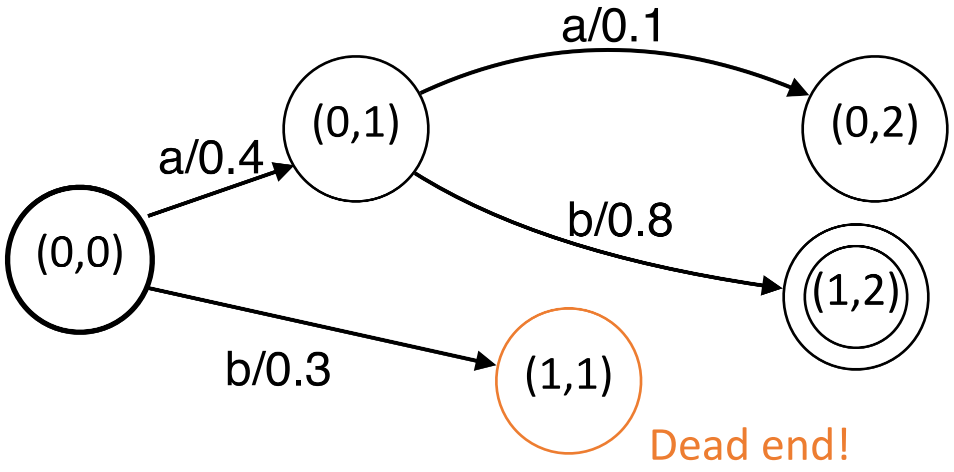


Operations: Intersect

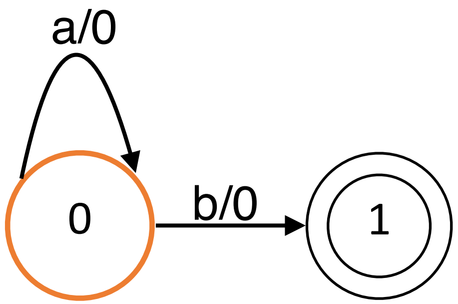


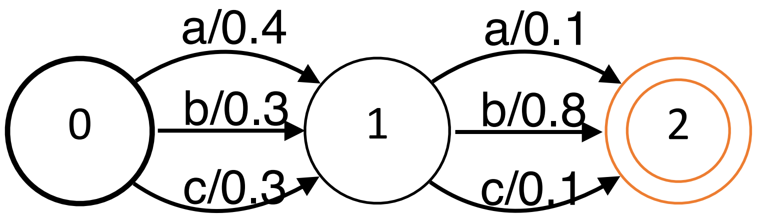


Intersected graph:

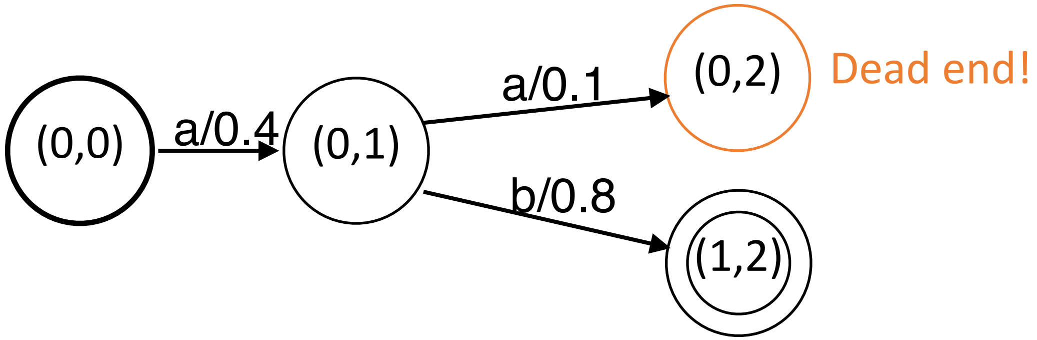


Operations: Intersect

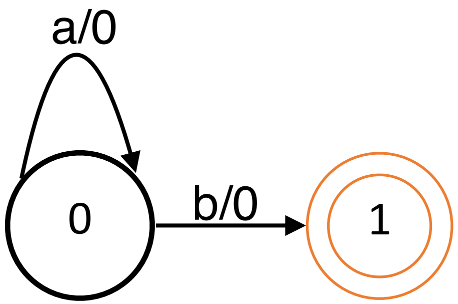


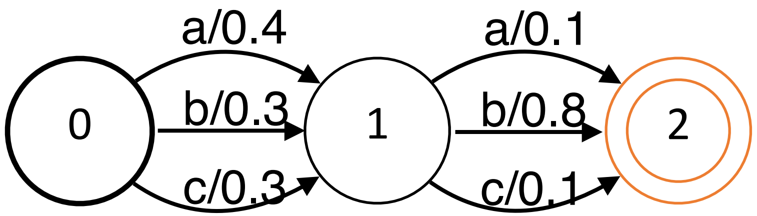


Intersected graph:

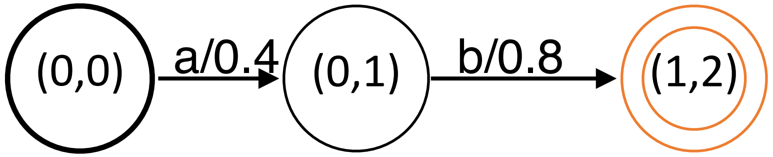


Operations: Intersect



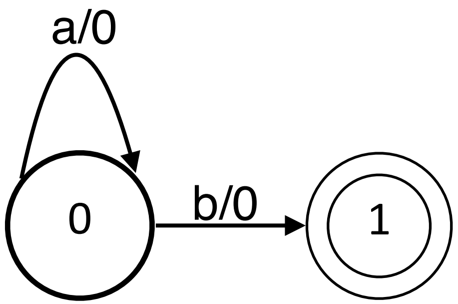


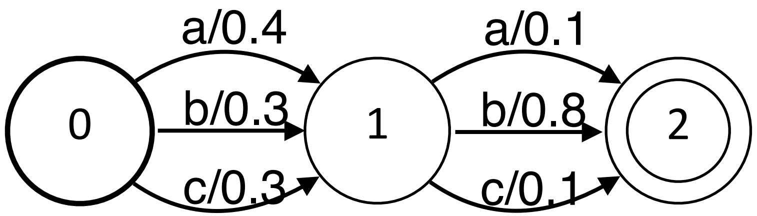
Intersected graph:



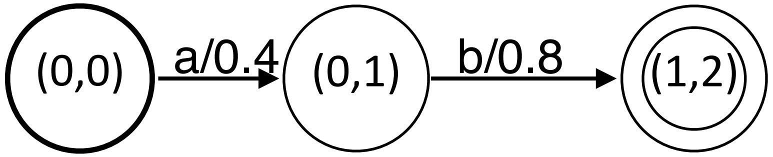
No arcs to explore!

Operations: Intersect



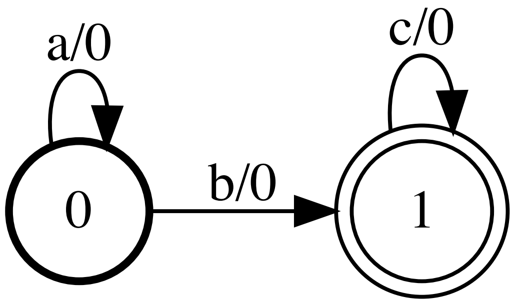


Intersected graph:

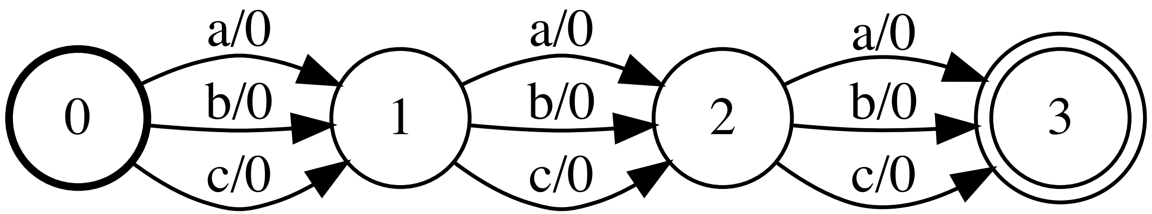


Operations: Intersect

`Graph g1`

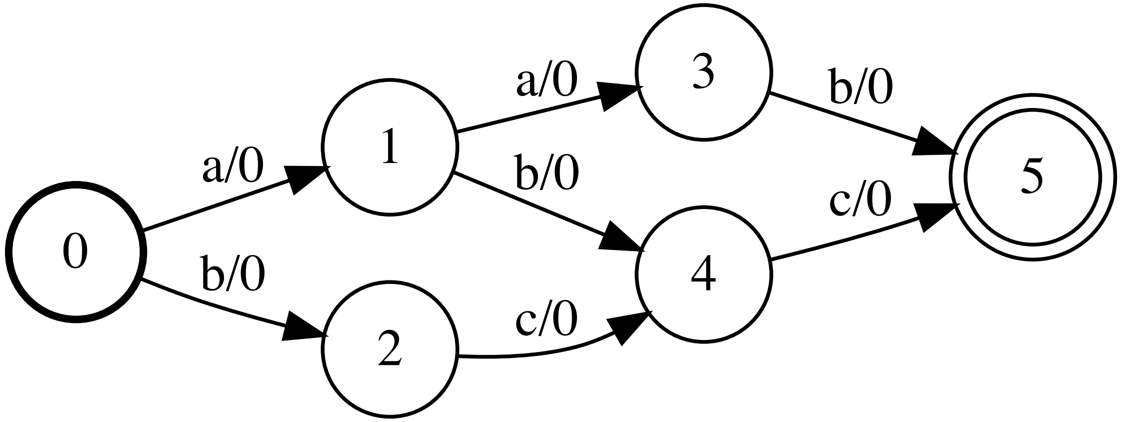


`Graph g2`



`intersect(g1, g2)`

↓

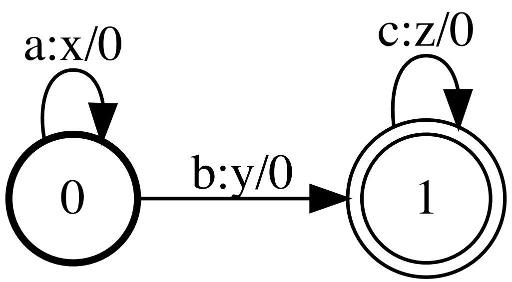


Operations: Compose

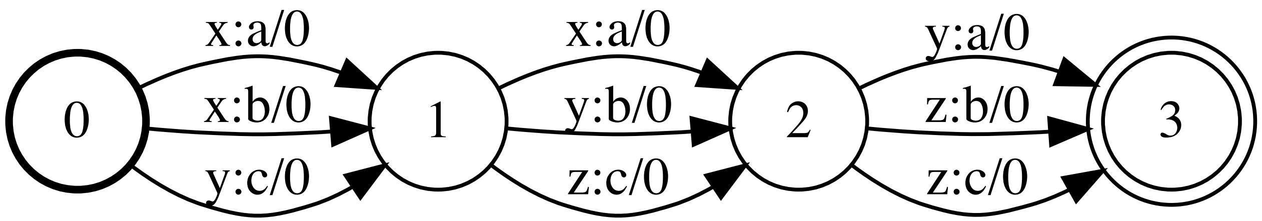
1. If $\mathtt{x} \rightarrow \mathtt{y}$ in the first graph and $\mathtt{y} \rightarrow \mathtt{z}$ in the second graph then $\mathtt{x} \rightarrow \mathtt{z}$ in the composed graph.
2. The score of the composed path is the sum of the scores of the paths in the input graphs.

Operations: Compose

`Graph g1`

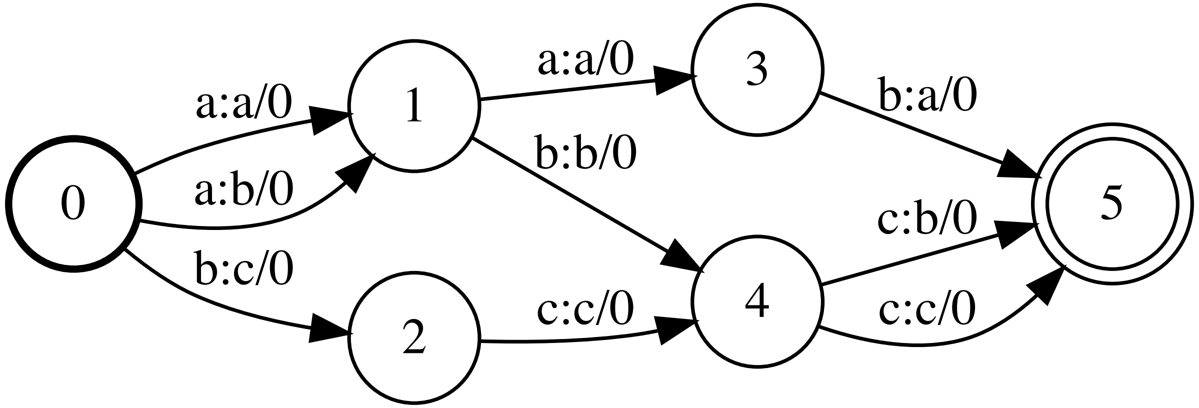


`Graph g2`



`compose(g1, g2)`

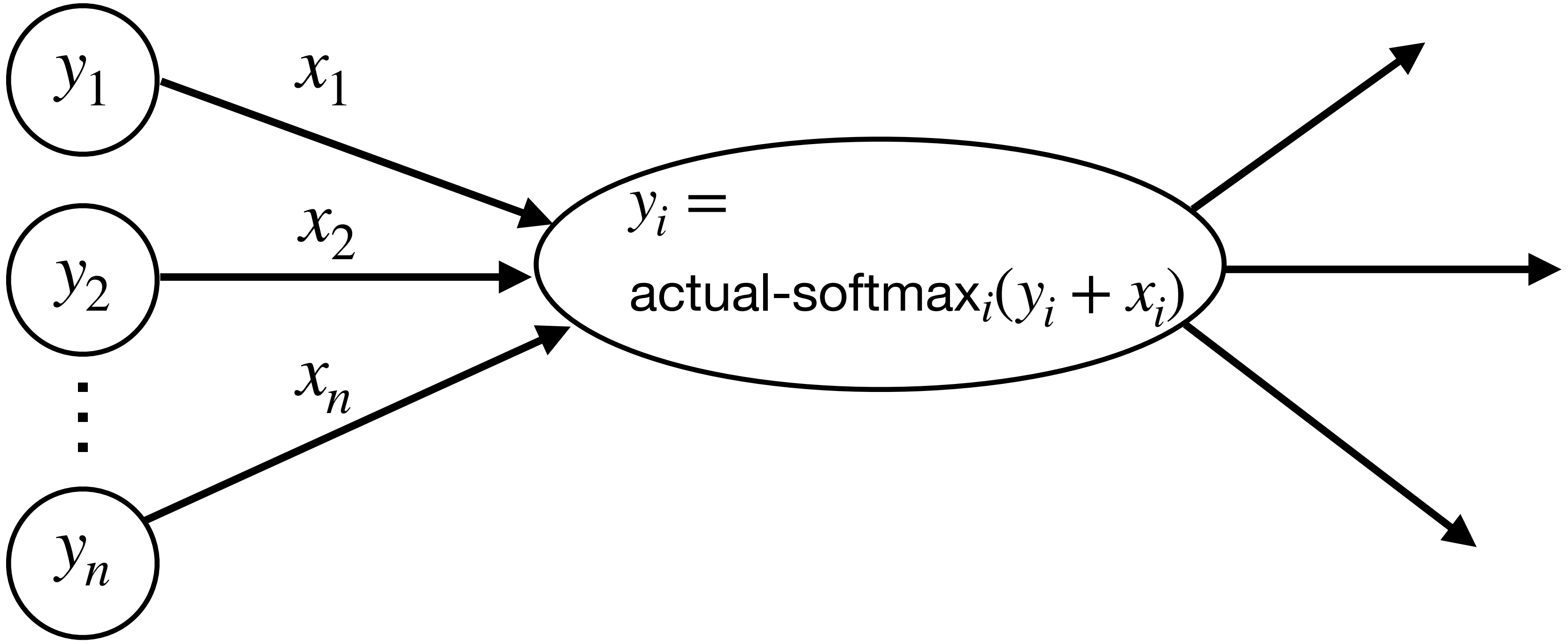
↓



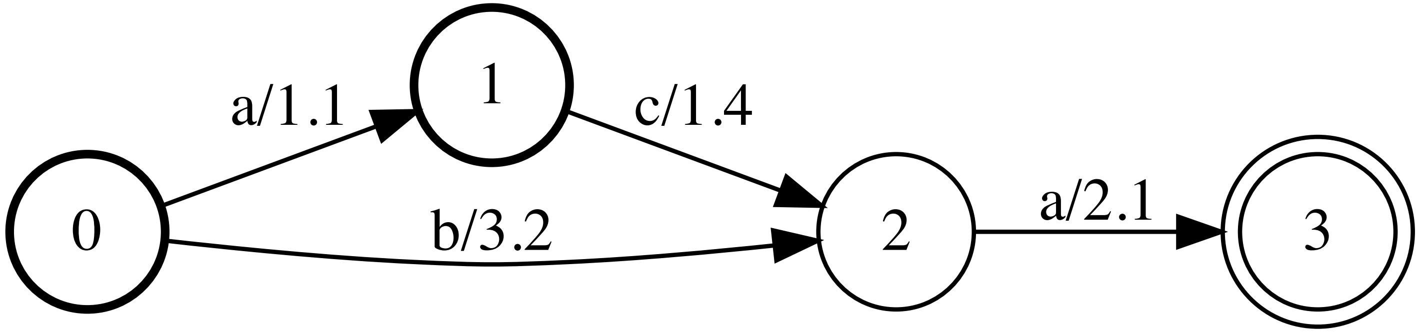
Operations: Forward Score

Accumulate the scores of all possible paths:

1. Assumes the graph is a DAG
2. Efficient dynamic programming algorithm



Operations: Forward Score



The graph accepts three paths:

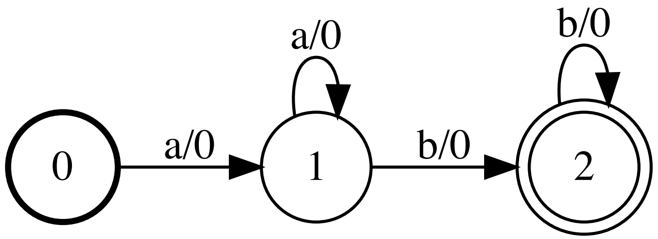
- `aca` with score=1.1+1.4+2.1
- `ba` with score=3.2+2.1
- `ca` with score=1.4+2.1

`forwardScore(g)` is the actual-softmax of the path scores.

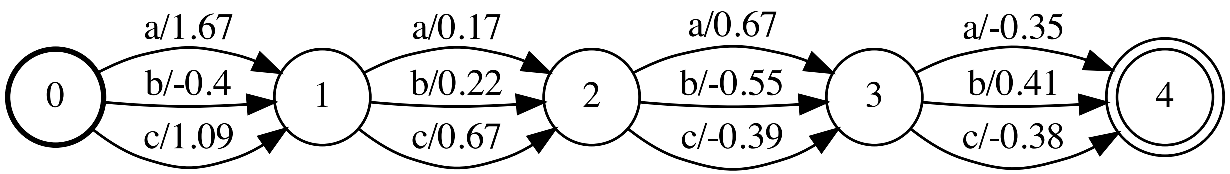
Sequence Criteria with WFSTs

Simple ASG (AutoSegCriterion) with WFSTs

Target graph `Y`



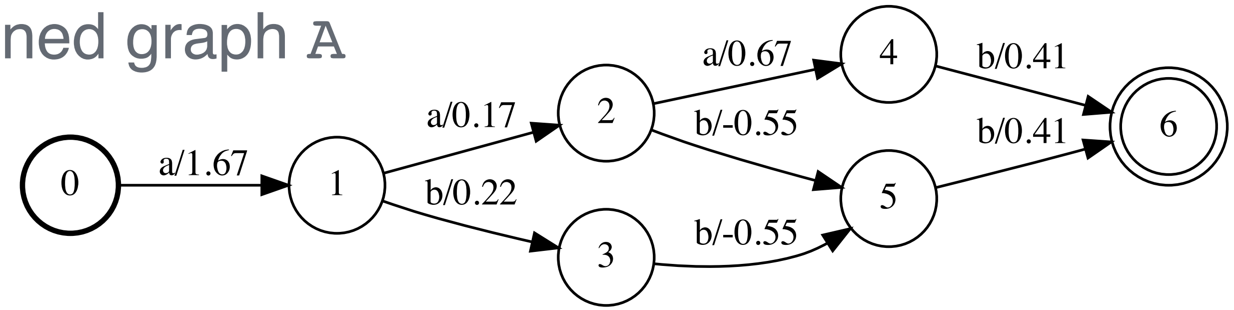
Emissions graph `E`



`intersect(Y, E)`

↓

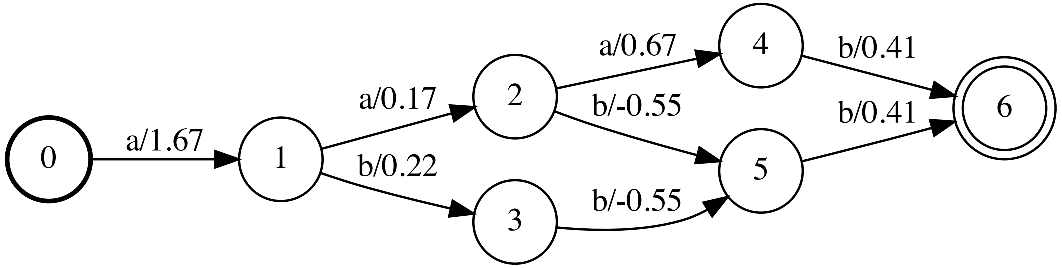
Target constrained graph `A`



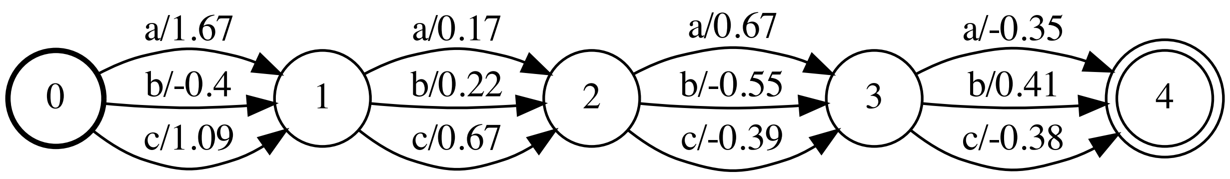
Sequence Criteria with WFSTs

Simple ASG with WFSTs

Target constrained graph `A`



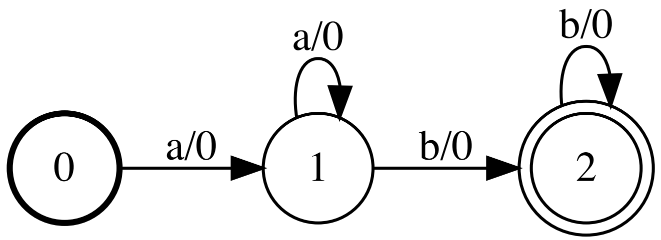
Normalization graph `Z=E`




```
loss = -(forwardScore(A) - forwardScore(E))
```

Sequence Criteria with WFSTs

Make the target graph




```python
import gtn

# Make the graph:
target = gtn.Graph(calc_grad=False)

# Add nodes:
target.add_node(start=True)
target.add_node()
target.add_node(accept=True)

# Add arcs:
target.add_arc(src_node=0, dst_node=1, label=0)
target.add_arc(src_node=1, dst_node=1, label=0)
target.add_arc(src_node=1, dst_node=2, label=1)
target.add_arc(src_node=2, dst_node=2, label=1)

# Draw the graph:
label_map = {0: 'a', 1: 'b'}
gtn.draw(target, "target.pdf", label_map)
```

Sequence Criteria with WFSTs

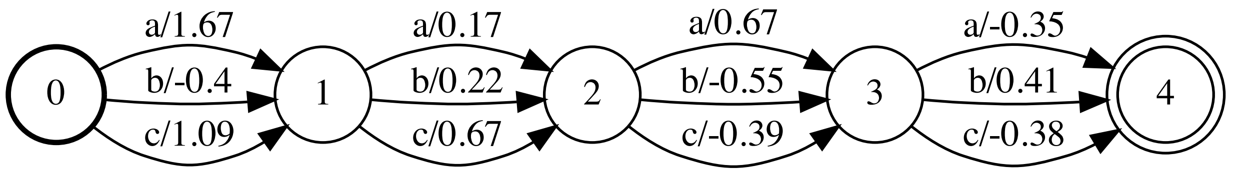
Make the emissions graph

```
import gtn

# Emissions array (logits)
emissions_array = np.random.randn(4, 3)

# Make the graph:
emissions = gtn.linear_graph(4, 3, calc_grad=True)

# Set the weights:
emissions.set_weights(emissions_array)
```



Example: ASG in GTN

ASG in GTN

Step 1:
Compute the graphs

```
from gtn import *

def ASG(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Example: ASG in GTN

ASG in GTN

Step 1:
Compute the graphs

Step 2:
Compute the loss

```
from gtn import *

def ASG(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Example: ASG in GTN

ASG in GTN

Step 1: Compute the graphs

Step 2: Compute the loss

Step 3: Automatic gradients!

```python
from gtn import *

def ASG(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Example: ASG in GTN

ASG in GTN

Step 1: Compute the graphs

Step 2: Compute the loss

Step 3: Automatic gradients!

```
from gtn import *

def ASG(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Example: CTC in GTN

CTC in GTN

```python
from gtn import *

def CTC(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Example: CTC in GTN

CTC in GTN

```python
from gtn import *

def CTC(emissions, target):
  # Compute constrained and normalization graphs:
  A = intersect(target, emissions)
  Z = emissions

  # Forward both graphs:
  A_score = forward_score(A)
  Z_score = forward_score(Z)

  # Compute loss:
  loss = negate(subtract(A_score, Z_score))

  # Clear previous gradients:
  emissions.zero_grad()

  # Compute gradients:
  backward(loss, retain_graph=False)
  return loss.item(), emissions.grad()
```

Only difference!

Thanks!

References and Further Reading:

CTC

- Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks , Graves, et al. 2006, ICML
- Sequence Modeling with CTC, Hannun. 2017, Distill, https://distill.pub/2017/ctc/

GTNs

- Gradient-based learning applied to document recognition, LeCun, et al. 1998, Proc. IEEE
- Global Training of Document Processing Systems using Graph Transformer Networks, Bottou, et al. 1997, CVPR
- More references: https://leon.bottou.org/talks/gtn

Modern GTNs

- Code: https://github.com/facebookresearch/gtn, `pip install gtn`
- Differentiable Weighted Finite-State Transducers, Hannun, et al. 2020, https://arxiv.org/abs/2010.01003