Speech Recognition and Graph Transformer Networks

Awni Hannun, awni@fb.com
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

“The quick brown fox jumps over the lazy dog”
Automatic Speech Recognition

Improved significantly in the past 8 years

Word Error Rate on Switchboard Conversational Phone Speech
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!

Quantifying Bias in Automatic Speech Recognition

Siyuan Feng, Olya Kudina, Bence Mark Halpern, Odette Scharenborg
Automatic Speech Recognition

But not yet solved!

"…state-of-the-art (SotA) ASRs struggle with the large variation in speech due to e.g., gender, age, speech impairment, race, and accents"
Question: Why has ASR gotten so much better?

Word Error Rate on Switchboard Conversational Phone Speech

![Graph showing the decline of word error rate from 2012 to 2017, reaching human level by 2017.](image-url)
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?

- Better models (end-to-end deep learning)
- More data
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:

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

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:

$$\log P(c | x_1) + \log P(a | x_2) + \log P(t | x_3)$$
The CTC Loss

Example:

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

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

Compute:

$$\log P(c | x_1) + \log P(a | x_2) + \log P(t | x_3)$$
The CTC Loss

Example:

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

Compute:

$$\log P(c \mid x_1) + \log P(a \mid x_2) + \log P(t \mid x_3) + \log P(?? \mid x_4)$$
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.

```
<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>a</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>a</td>
<td>t</td>
</tr>
<tr>
<td>Or A2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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 $\log P(Y \mid X)$?

- $A_1$: $x_1 \rightarrow c$, $x_2 \rightarrow a$, $x_3 \rightarrow t$, $x_4 \rightarrow t$
- Or $A_2$: $x_1 \rightarrow c$, $x_2 \rightarrow a$, $x_3 \rightarrow a$, $x_4 \rightarrow t$
- Or $A_3$: $x_1 \rightarrow c$, $x_2 \rightarrow c$, $x_3 \rightarrow a$, $x_4 \rightarrow t$
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!

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

\(\approx\) 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: $\alpha_t^u$ 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: \( X = [x_1, x_2, x_3, x_4], \ Y = [c, a, t] \)

\[
\alpha^c_2 = \log P(c \mid x_1) + \log P(c \mid x_2)
\]
The CTC Loss

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

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

$$\alpha^a_2 = \log P(c \mid x_1) + \log P(a \mid x_2)$$
The CTC Loss

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

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

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

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

$\log P(A_2) = \log P(c | x_1) + \log P(a | x_2) + \log P(a | x_3)$
The CTC Loss

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

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

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

\[
\log P(A_1) = \log P(c | x_1) + \log P(c | x_2) + \log P(a | x_3)
\]

\[
\log P(A_2) = \log P(c | x_1) + \log P(a | x_2) + \log P(a | x_3)
\]
The CTC Loss

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

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

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

\[
\log P(A_1) = \alpha_2^c + \log P(a | x_3)
\]

\[
\log P(A_2) = \alpha_2^a + \log P(a | x_3)
\]
The CTC Loss

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

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

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

\[
\log P(A_1) = \alpha_2^c + \log P(a | x_3)
\]

\[
\log P(A_2) = \alpha_2^a + \log P(a | x_3)
\]

Exercise: prove this equality!
The CTC Loss

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

General recursion:

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

\[ \alpha^u_t = \text{actual-softmax}[\alpha^u_{t-1}, \alpha^{u-1}_{t-1}] + \log P(y_u | x_t) \]
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)

<table>
<thead>
<tr>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
<th>x₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>α₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

\[
\begin{align*}
\alpha_1^c &= \log P(Y | X) \\
\alpha_2^a &= \alpha_2^c \\
\alpha_3^a &= \alpha_3^c \\
\alpha_4^a &= \alpha_4^d \\
\alpha_5^t &= \alpha_5^t
\end{align*}
\]
The CTC Loss

**Problem:** Not every input corresponds to “speech”

Can be silence, noise, laughter, ...
The CTC Loss

**Solution:** Use a “garbage” or *blank* token: \(<\triangleright/>\)

\[ x_1 \rightarrow c \]
\[ x_2 \rightarrow a \]
\[ x_3 \rightarrow \text{Can be silence, noise, laughter, ...} \]
\[ x_4 \rightarrow t \]
The CTC Loss

Solution: Use a “garbage” or blank token: \( \langle b \rangle \)

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

$$x_1 \downarrow c \quad x_2 \downarrow a \quad x_3 \downarrow t \quad x_4 \quad x_5 \downarrow t$$

No!

Corresponds to “catt”.
The CTC Loss

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

Blank token is optional …

except between repeats in $Y$

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

\[
\begin{align*}
\alpha_t^a & \quad \alpha_{t+1}^a \\
\langle b \rangle & \quad \alpha_t^{<b>} \\
\end{align*}
\]
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: $\log P(Y | X)$
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: \( \log P(Y | X) \)
2. Language model: \( \log P(Y) \)

Find:

\[
Y^* = \arg\max_Y \log P(Y | X) + \log P(Y)
\]
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

Better path!
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

$N = 3$

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

$$Y^* \approx \arg\max_Y \log P(Y \mid X) + \log P(Y)$$
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

For deep learning: see pages 1-16
For GTNs: see pages 16-42
Core data structure

Neural Networks
- Tensor
- Matrix multiplication
- Reduction operations (sum, prod, ...)
- Unary and binary operations (negate, add, subtract, ...)

GTNs
- Graph (WFSA)
- Compose
  - Shortest distance ops (forward, viterbi)
  - Unary and binary operations (closure, union, concatenate, ...)

Core operations

Unary and binary operations (closure, union, concatenate, ...)
Example: WFSTs in Speech Recognition

- Acoustic Model
- Language Model
- Lexicon
- Decoder
Example: WFSTs in Speech Recognition

Acoustic Model

Language Model

Lexicon

WFST

WFST

WFST

Only combined when decoding!
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:

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp-CTC</td>
<td>9,742</td>
</tr>
<tr>
<td>wav2letter</td>
<td>2,859</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1,161</td>
</tr>
<tr>
<td>GTN</td>
<td>30</td>
</tr>
</tbody>
</table>
## Sequence Criteria with WFSTs

### Lines of code for CTC: Custom vs GTN

<table>
<thead>
<tr>
<th></th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp-CTC</td>
<td>9,742</td>
</tr>
<tr>
<td>wav2letter</td>
<td>2,859</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1,161</td>
</tr>
<tr>
<td>GTN</td>
<td>30</td>
</tr>
</tbody>
</table>

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

$\epsilon$ transitions are allowed in WFSAs
More WFSAs and WFSTs

\( \epsilon \) 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*\}

\[\text{union}\{g_1, g_2, g_3\} \rightarrow\]
Operations: Kleene Closure

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

Recognizes \{aba\}

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:

Dead end!
Operations: Intersect

Intersected graph:

Dead end!
Operations: Intersect

Intersected graph:

No arcs to explore!
Operations: Intersect

Intersected graph:
Operations: Intersect

Graph g1

Graph g2

\[ \text{intersect}(g1, g2) \]
Operations: Compose

1. If \(x \rightarrow y\) in the first graph and \(y \rightarrow z\) in the second graph then \(x \rightarrow z\) 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**
- Node 0: a:x/0, b:y/0
- Node 1: c:z/0

**Graph g2**
- Node 0: x:a/0, x:b/0, y:c/0
- Node 1: x:a/0, y:b/0, z:c/0
- Node 2: y:a/0, z:b/0, z:c/0
- Node 3: 

**compose(g1, g2)**

- Node 0
  - a:a/0, a:b/0, b:c/0
- Node 1
  - a:a/0, b:b/0
- Node 2
  - b:a/0, c:b/0, c:c/0
- Node 3
  - b:a/0, c:b/0
- Node 4
  - c:c/0
- Node 5
Operations: Forward Score

Accumulate the scores of all possible paths:

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

\[ y_i = \text{actual-softmax}_i(y_i + x_i) \]
The graph accepts three paths:

- \textit{aca} with score=$1.1+1.4+2.1$
- \textit{ba} with score=$3.2+2.1$
- \textit{ca} with score=$1.4+2.1$

\textit{forwardScore}(g)$ is the actual-softmax of the path scores.
Sequence Criteria with WFSTs

Simple ASG (AutoSegCriterion) with WFSTs

Target graph $\mathcal{Y}$

Emissions graph $\mathcal{E}$

$$\text{intersect}(\mathcal{Y}, \mathcal{E})$$

Target constrained graph $\mathcal{A}$
**Sequence Criteria with WFSTs**

**Simple ASG with WFSTs**

Target constrained graph $A$

Normalization graph $Z = E$

\[
\text{loss} = -(\text{forwardScore}(A) - \text{forwardScore}(E))
\]
Sequence Criteria with WFSTs

Make the target graph

```python
import gtn

target = gtn.Graph(calc_grad=False)

target.add_node(start=True)
target.add_node(accept=True)

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)

label_map = {0: 'a', 1: 'b'}
gtn.draw(target, "target.pdf", label_map)
```
Sequence Criteria with WFSTs

Make the emissions graph

```python
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)
```
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

Step 1: Compute the graphs

Step 2: Compute the loss

```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()
```
```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

Step 1: Compute the graphs

Step 2: Compute the loss

Step 3: Automatic gradients!
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()
```
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()
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()
Thanks!

References and Further Reading:

CTC

• Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks, Graves, et al. 2006, ICML

GTNs

• More references: https://leon.bottou.org/talks/gtn

Modern GTNs

• Code: https://github.com/facebookresearch/gtn, pip install gtn