Lecture 11:

Sequence Models

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

Part 1/5:

Sequence Modeling

1. What is sequence data?
2. Why Special Modeling Is Needed?
3. Features of Sequences
4. Feature Engineering for Sequences

What is Sequence Data?

Data points ordered by time, position, or logical sequence

> 🔑 **Key Property: Order matters — shuffling changes meaning**

Dependencies

Temporal or spatial dependencies between elements

Feature Types

Univariate (single) or multivariate (multiple features)

Length

Fixed or variable across samples

Structure

Sequential ordering preserves critical information

Examples

📈 Stock Prices

`$100` → `$105` → `$103` → `$108` → `$110`

🧬 DNA Sequences

`A` → `T` → `G` → `C` → `T` → `A`

Time series of financial data with temporal dependencies

Nucleotide order determines genetic information

💬 Sentences

I → love → machine → learning

Word order conveys meaning and grammar

🎬 Video Frames

Frame_1 → Frame_2 → Frame_3 → Frame_4

Sequential visual data capturing motion over time

Why Special Modeling Is Needed?

✗ Traditional ML

- Assumes **independence** between samples
- **No memory** of past events
- Feedforward networks **cannot capture** sequential patterns
- Ignores **temporal/spatial dependencies**
- Context from previous elements is **not utilized**

✓ Sequence Models

- Recognize **dependencies** between data points
- **Maintain memory** of historical information
- Designed to **capture sequential patterns**
- Model **temporal/spatial relationships**
- Use **context** to influence predictions

↓

Key Insight

Memory of past events is crucial for accurate forecasting.
We need models that maintain and utilize historical information.

Features of Sequences

Temporal Ordering

1

Position in sequence is meaningful and carries information

Variable Length

2

Sequences can have different durations or sizes

Local Patterns

3

Nearby elements often correlate with each other

Long-Range Dependencies

4

Distant elements may influence each other across time

Trend & Seasonality

In time series data

Hierarchical Structure

In language and biology

Feature Engineering for Sequences

Raw Sequence Data

↓

Windowing

Creating fixed-size segments from sequences

Lag Features

Using previous timesteps as predictors

Rolling Statistics

Moving averages, standard deviations

Differencing

Removing trends to achieve stationarity

Fourier Transforms

Extracting frequency domain features

Embeddings

Representations for categorical sequential data

↓

Engineered Features

Ready for sequence modeling and prediction

Real-World Application Examples

Windowing - Stock Price Prediction

Original Data
Day 1: $100
Day 2: $102
Day 3: $101
Day 4: $105
Day 5: $107

3-Day Window Applied
Window 1: [100, 102, 101] → 105
Window 2: [102, 101, 105] → 107
Predict next day from past 3 days

Lag Features - Website Traffic

Hourly Visitors
10:00 - 150 visitors
11:00 - 180 visitors
12:00 - 220 visitors
13:00 - 200 visitors

Lag Features Created
Current: 220, 1hr ago: 180
Current: 200, 1hr ago: 220
Utilize previous hour information

Rolling Statistics - Temperature Data

Daily High Temperature (°C)
Mon: 22°C

3-Day Moving Average
Wed: 23.0°C (22+24+23)/3

Tue: 24°C

Wed: 23°C

Thu: 25°C

Fri: 26°C

Thu: 24.0°C (24+23+25)/3

Fri: 24.7°C (23+25+26)/3

Reduce volatility, identify trends

📉 Differencing - Sales Data

Monthly Cumulative Sales ($M)

Jan: $100M

Feb: $130M

Mar: $145M

Apr: $170M

Monthly Growth (Difference)

Feb: +$30M

Mar: +$15M

Apr: +$25M

Remove trend, analyze changes

🌊 Fourier Transforms - ECG Signal

Time Domain Signal

t=0ms: 0.1V

t=10ms: 0.8V

t=20ms: 0.2V

Voltage changes over time

Frequency Domain Features

Heart Rate: 75 BPM (1.25Hz)

High Frequency: 0.02

Low Frequency: 0.15

Extract periodic patterns

Embeddings - User Behavior Sequence

Click Event Sequence

Homepage → Search

Product Page → Cart

Checkout → Complete

Categorical sequential data

Embedding Vectors

Homepage: [0.2, 0.8, 0.1]

Search: [0.3, 0.6, 0.4]

Product: [0.7, 0.2, 0.5]

Transform to semantic representation

Part 2/5:

Types of Sequence Data

5. Time Series Data
6. Text Data
7. Speech/Audio Data & Other Sequences

Time Series Data

Financial

Stock prices, trading volumes, market indices

IoT Sensors

Temperature, humidity, energy consumption

Medical

Heart rate, blood pressure, EEG signals

Weather

Temperature, precipitation, wind speed over time

Business Metrics

Sales, revenue, customer traffic

...

And More

Many other time-dependent applications

Key Characteristics

 Continuous values

 Regular/Irregular sampling

Text Data

🔤 Sequential Tokens

Words | Characters | Subwords

Key Characteristics

- 📝 Natural language structure
- ↔️ Variable length sequences
- 🏗️ Grammatical structure
- 🔄 Context-dependent meaning

✓ Applications

- Machine Translation
- Text Summarization
- Sentiment Analysis

⚠ Challenges

- Large vocabulary size
- Rare words handling
- Ambiguity (polysemy, homonyms)

💡 Real-World Examples

Machine Translation

Google Translate: "안녕하세요" → "Hello"

Text Summarization

News aggregators condensing long articles into key bullet points

Sentiment Analysis

Amazon reviews: "This product is amazing!" → Positive (★★★★★)

Speech/Audio Data & Other Sequences

Audio Waveforms

Speech recognition, music generation

Video Sequences

Action recognition, video prediction

Biological Sequences

DNA, RNA, protein structures

User Behavior

Clickstreams, purchase history

Network Traffic

Packet sequences, anomaly detection

... Many More

Various other sequential domains

Each domain has unique characteristics and challenges

Part 3/5:

Statistical Approaches

8. Moving Average (MA)
9. Autoregressive (AR)
10. ARMA & ARIMA
11. Regression-Based Extensions
12. Limitations of Traditional Methods

Moving Average (MA)

Types of Moving Averages

Simple MA

Equal weights for recent observations

Weighted MA

Different weights by recency

Exponential Smoothing

Exponentially decreasing weights

✓ Benefits

- Smoothing noisy data
- Trend identification
- No look-ahead bias for forecasting

⚠ Limitations

- Cannot capture complex patterns
- Cannot model seasonality

Trend Calculation Example

Time Period	t=1	t=2	t=3	t=4	t=5
Observed Value (Y_t)	10	12	15	14	18

3-Period Simple Moving Average

t=3: $MA_3 = (10 + 12 + 15) / 3 = \mathbf{12.33}$

t=4: $MA_3 = (12 + 15 + 14) / 3 = \mathbf{13.67}$

t=5: $MA_3 = (15 + 14 + 18) / 3 = \mathbf{15.67}$

✅ Trend: 12.33 → 13.67 → 15.67 (Upward trend)

🔮 Forecasting with Moving Average

Method: Use the most recent MA value as the forecast for the next period

Forecast for t=6: $\hat{Y}_6 = MA_3$ at t=5 = **15.67**

Alternative (if $Y_6 = 20$): MA_3 at t=6 = (14 + 18 + 20) / 3 = **17.33**

Then forecast for t=7: $\hat{Y}_7 = 17.33$

💡 The MA forecast assumes the trend continues at the current smoothed level

Autoregressive (AR)

Core Concept: AR(p)

Predicts current value from previous **p** values
Linear combination of past observations

AR Model Formula

$$X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2} + \ldots + \varphi_p X_{t-p} + \varepsilon_t$$

Order Selection

Via AIC/BIC criteria

Stationarity

Assumes stationary time series

Dependencies

Good for linear temporal dependencies

Trend Calculation Example

📌 Sample Data: Monthly Sales

t	t	t	t	t	t
1	2	3	4	5	6
100	110	105	115	120	125

📐 Method 1: Trend using Moving Average

- **3-period Moving Average** calculation: $MA_3 = (Y_{t-1} + Y_t + Y_{t+1}) / 3$
- t=2: **MA = (100+110+105)/3 = 105.0**
- t=3: MA = (110+105+115)/3 = 110.0
- t=4: MA = (105+115+120)/3 = 113.3
- t=5: MA = (115+120+125)/3 = 120.0

📐 Method 2: Trend using Linear Regression

- **Regression equation**: $Y_t = a + b \cdot t$
- Slope $b = \Sigma[(t - \bar{t})(Y_t - \bar{Y})] / \Sigma(t - \bar{t})^2$
- Calculation: $b \approx 5.14$ → **Average increase of 5.14 per month**
- Intercept $a = \bar{Y} - b \cdot \bar{t} \approx 94.86$

- Trend equation: $Y_t = 94.86 + 5.14 \cdot t$

Forecasting Using Trend

Trend-Based Forecasting

Use the estimated trend equation to predict future values

📈 Method 1: Forecasting with Moving Average

- **Last calculated MA** at t=5: MA = 120.0
- **Trend rate**: Calculate average change between consecutive MAs
- (110.0 - 105.0) + (113.3 - 110.0) + (120.0 - 113.3) = 15.0
- Average change ≈ 15.0 / 3 = **5.0 per period**
- **Forecast for t=7**: 125 + 5.0 = **130.0**
- **Forecast for t=8**: 130.0 + 5.0 = **135.0**

📈 Method 2: Forecasting with Linear Regression (Recommended)

- **Use trend equation**: $Y_t = 94.86 + 5.14 \cdot t$
- **Forecast for t=7**: $Y_7 = 94.86 + 5.14 \times 7 =$ **130.84**
- **Forecast for t=8**: $Y_8 = 94.86 + 5.14 \times 8 =$ **135.98**
- **Forecast for t=9**: $Y_9 = 94.86 + 5.14 \times 9 =$ **141.12**
- **Forecast for t=10**: $Y_{10} = 94.86 + 5.14 \times 10 =$ **146.26**

Summary: Forecasted Values

Period	t=7	t=8	t=9	t=10
Forecast	130.84	135.98	141.12	146.26

ARMA & ARIMA

ARMA

Combines AR and MA components for stationary series

```
ARMA(p, q)
```

ARIMA

Adds differencing for non-stationary series

```
ARIMA(p, d, q)
```

Seasonal ARIMA (SARIMA)

Extended version for modeling periodic patterns and seasonality

Methodology

Box-Jenkins for model identification

Applications

Widely used in econometrics & forecasting

Trend Calculation Example

선형 트렌드 추정 (Linear Trend Estimation)

Time (t)	1	2	3	4	5
Observed (Y_t)	10	12	15	17	20

1 Trend Model

$$Y_t = \beta_0 + \beta_1 \cdot t + \varepsilon_t$$

2 Coefficient Estimation using OLS

$$\beta_1 = \Sigma[(t - \bar{t})(Y_t - \bar{Y})] \;/\; \Sigma(t - \bar{t})^2$$

$$\beta_0 = \bar{Y} - \beta_1 \cdot \bar{t}$$

$\beta_1 = 2.5$ (Slope) | $\beta_0 = 7.5$ (Intercept)

3 Final Trend Equation

$$\hat{Y}_t = 7.5 + 2.5 \cdot t$$

Linear Trend

Constant rate of increase/decrease

Nonlinear Trend

Polynomial of degree 2 or higher

Exponential Trend

Exponential growth/decay

Differencing Method

Differencing to remove trend

4 Forecasting with Trend Model

Using the estimated equation: $\hat{Y}_t = 7.5 + 2.5 \cdot t$

Forecast for t = 6:
$\hat{Y}_6 = 7.5 + 2.5 \times 6 = 22.5$

Forecast for t = 7:
$\hat{Y}_7 = 7.5 + 2.5 \times 7 = 25.0$

Forecast for t = 8:
$\hat{Y}_8 = 7.5 + 2.5 \times 8 = 27.5$

Forecast Limitation

Long-term forecasts become less reliable

Best Practice

Check model assumptions and residuals

Regression-Based Extensions

ARIMAX

ARIMA with exogenous variables for external predictors

Vector Autoregression (VAR)

Handles multivariate time series simultaneously

State Space Models

Uses Kalman filtering for latent state estimation

Structural Time Series

Models underlying components explicitly

External Predictors

Incorporates exogenous variables

Multiple Series

Handles multivariate relationships

⚠ Limitations of Traditional Methods

Traditional statistical methods face several fundamental constraints

Linear Assumptions

Assume linear relationships between observations

Manual Engineering

Require manual feature engineering

Complex Patterns

Struggle with high-dimensional or complex patterns

Long-Term Dependencies

Limited capacity for long-term dependencies

Unstructured Data

Cannot handle raw data (images, text)

Representation Learning

Lack automatic representation learning

Part 4/5:

Deep Learning for Sequences

13. High-Dimensional Sequences Challenge
14. CNN for Sequences
15. RNN (Recurrent Neural Network)
16. LSTM (Long Short-Term Memory)
17. GRU (Gated Recurrent Unit)
18. Bidirectional RNNs

High-Dimensional Sequences Challenge

⚠ Challenges

- **Curse of dimensionality** in sequence modeling
- **Exponential growth** in parameter space
- Need for **automatic feature extraction**
- Raw data (pixels, waveforms) vs **handcrafted features**

✓ Deep Learning Solution

- **Hierarchical** representation learning
- **End-to-end training** from raw sequences
- **Automatic feature learning** at multiple levels
- Direct processing of **raw data**

Deep Learning Enables

Automatic feature extraction and hierarchical representation learning
from high-dimensional raw sequences

CNN for Sequences

1D Convolutions scan across temporal dimension for sequence processing

Local Pattern Detection
Weight sharing across positions

Growing Receptive Field
Expands with network depth

Parallel Computation
Efficient processing

Hierarchical Features
Captures multi-scale patterns

✓ Applications
- Text classification
- Audio processing
- Time series analysis

⚠ Limitations
- Fixed receptive field
- No explicit memory mechanism

Trend Calculation Example with CNN

Time Series Data: Stock Price Trend Detection

Input Sequence (5-day prices):

100 → 102 → 105 → 103 → 107

Step 1: 1D Convolution

Filter size: 3, Stride: 1

[100, 102, 105] × [0.5, 0.3, 0.2] = **101.1**

[102, 105, 103] × [0.5, 0.3, 0.2] = **103.1**

[105, 103, 107] × [0.5, 0.3, 0.2] = **104.3**

Step 2: Activation Function (ReLU)

101.1 → 101.1

103.1 → 103.1

104.3 → **104.3 ↗**

🎯 Trend Analysis Result

Upward Trend Detected Confidence: 85%

💡 *Through consecutive convolutional layers, short-term, mid-term, and long-term trends can be captured simultaneously*

RNN (Recurrent Neural Network)

Core Architecture

$x_t \rightarrow h_t \circlearrowright h_{t+1}$

Hidden State Update Formula

$$h_t = \tanh(W_{hh} \times h_{t-1} + W_{xh} \times x_t)$$

- **Hidden state** maintains sequence memory
- **Weight sharing:** same weights at each timestep
- **Bidirectional RNN:** forward & backward processing
- **Sequential processing:** models temporal dependencies

⚠ Training Challenges

- Vanishing/exploding gradients
- Difficult to train for long sequences

📊 Detailed Trend Calculation Example

Setup: Hidden dim = 2, Input dim = 3

W_{xh} = [[0.5, 0.3], [0.2, 0.4], [0.1, 0.6]] (3×2), W_{hh} = [[0.7, 0.2], [0.3, 0.8]] (2×2)

Input sequence: x_0 = [1.0, 0.5, 0.8], x_1 = [1.2, 0.6, 0.9]

Step 1 (t=0):

- h_0 = [0, 0] (initial hidden state)
- $W_{hh} \times h_0$ = [[0.7, 0.2], [0.3, 0.8]] × [0, 0] = **[0, 0]**
- $W_{xh} \times x_0$ = [[0.5, 0.3], [0.2, 0.4], [0.1, 0.6]] × [1.0, 0.5, 0.8] = **[0.68, 0.88]**
- h_1 = tanh([0, 0] + [0.68, 0.88]) = tanh([0.68, 0.88]) = **[0.59, 0.71]**

Step 2 (t=1):

- h_1 = [0.59, 0.71]
- $W_{hh} \times h_1$ = [[0.7, 0.2], [0.3, 0.8]] × [0.59, 0.71] = **[0.55, 0.75]**
- $W_{xh} \times x_1$ = [[0.5, 0.3], [0.2, 0.4], [0.1, 0.6]] × [1.2, 0.6, 0.9] = **[0.78, 1.02]**
- h_2 = tanh([0.55, 0.75] + [0.78, 1.02]) = tanh([1.33, 1.77]) = **[0.87, 0.95]**

> **Trend detected:** Hidden states evolve [0, 0] → [0.59, 0.71] → [0.87, 0.95], capturing increasing trend through accumulated memory in both dimensions

LSTM (Long Short-Term Memory)

✓ Solves vanishing gradient problem with gating mechanism

Forget Gate

Decides what to remove from cell state

Input Gate

Controls new information addition

Output Gate

Determines hidden state output

Cell State

Long-term memory pathway

Hidden State

Short-term working memory

🎯 Success

Widely successful for various sequence tasks across domains

📊 RNN vs LSTM Comparison

🔴 RNN Issues

- Vanishing gradient problem
- Cannot capture long-term dependencies

✅ LSTM Solutions

- Gating mechanism prevents gradient vanishing
- Maintains long-term memory via cell state

- Simple architecture with single state
- Gradient diminishes exponentially over time

- Complex architecture with gates
- Controlled information flow preserves gradients

💡 Gradient Flow Example

Consider a sequence of length T with gradient backpropagation:

$$\text{RNN: } \partial L/\partial h_0 = \partial L/\partial h_t \times \left(\prod_{i=1}^{T} \partial h_i/\partial h_{i-1}\right) \rightarrow \text{vanishes when T is large}$$

LSTM gates control gradient flow, preventing exponential decay:
Forget gate + Input gate + Cell state = Stable gradient pathway

GRU (Gated Recurrent Unit)

Simplified LSTM with fewer parameters

Two-Gate Mechanism

Update Gate
Combines forget and input gates from LSTM

Reset Gate
Controls past information usage

Faster Training
Than LSTM

Similar Performance
On many tasks

Easier to Implement
And tune

LSTM vs GRU: Key Differences

LSTM

3 gates : Forget, Input, Output

GRU

2 gates : Update, Reset

2 states : Cell state (C) + Hidden state (h)

More parameters to learn

Better for complex, long sequences

Slower training time

More hyperparameters to tune

1 state : Hidden state (h) only

Fewer parameters (~25-30% less)

Better for smaller datasets

Faster training time

Simpler to implement and tune

LSTM Calculation (3D Vector Example)

Given Input:

```
x_t = [0.5, -0.3, 0.8] (current input)
h_{t-1} = [0.2, 0.4, -0.1] (previous hidden state)
C_{t-1} = [0.6, -0.2, 0.3] (previous cell state)
```

Step 1: Forget Gate (f_t)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Decides what to forget from cell state

```
Result: f_t = [0.65, 0.42, 0.78]
```

Step 2: Input Gate (i_t) + Candidate ($\tilde{C}_t$)

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Decides what new information to add

Result: $i_t = [0.53, 0.71, 0.48]$, $\tilde{C}_t = [0.32, -0.54, 0.67]$

Step 3: Update Cell State (C_t)

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Result: $C_t = [0.56, -0.47, 0.55]$

Step 4: Output Gate (o_t) + Hidden State (h_t)

GRU Calculation (3D Vector Example)

Given Input:

$x_t = [0.5, -0.3, 0.8]$ (current input)

$h_{t-1} = [0.2, 0.4, -0.1]$ (previous hidden state)

Note: GRU has NO separate cell state - only hidden state!

Step 1: Reset Gate (r_t)

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Controls how much past info to use for candidate

```
Result: r_t = [0.58, 0.73, 0.45]
```

Step 2: Update Gate (z_t)

```
z_t = σ(W_z · [h_{t-1}, x_t] + b_z)
```


Combines forget & input: decides mix of old vs new

Practice Materials

Interactive RNN Visualization Tool

RNN Explainer

Interactive simulation tool to visually understand how LSTM and GRU work

Go to Practice Site →

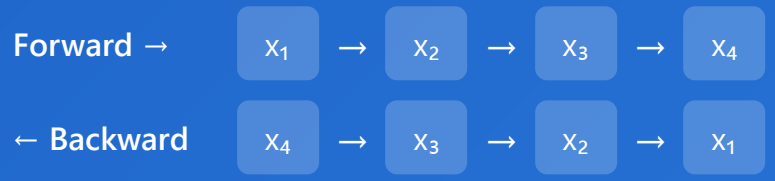
Gate Operation Visualization

Interactive Parameter Control

Real-time Results

Bidirectional RNNs

Dual-Direction Architecture



⊕ Concatenate or Average

Captures future context for current prediction

Full sequence required before processing

✓ Best For

Tasks with full sequence available:

- Named Entity Recognition
- POS Tagging
- Sentiment Analysis

⚠ Cannot Be Used For

Real-time streaming applications where future context is unavailable

Part 5/5:

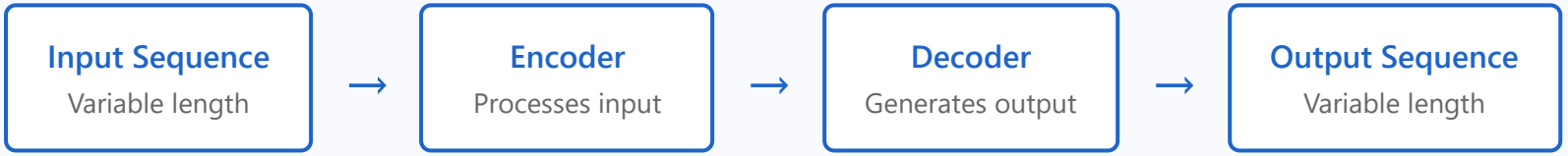
Advanced Topics

19. Sequence-to-Sequence Models
20. Encoder-Decoder Architecture
21. Teacher Forcing
22. Beam Search
23. CTC Loss
24. Practical Implementation Tips
25. Applications and Next Steps

Sequence-to-Sequence Models

Maps input sequence to output sequence

Different lengths allowed (input ≠ output)



🎯 Key Feature

- Handles variable-length sequences
- End-to-end learning

💼 Applications

- Machine Translation
- Text Summarization
- Conversation Systems

Bottleneck Challenge

Entire input compressed to fixed vector → Attention mechanism addresses this

🔄 Step-by-Step Process

1. **Input Encoding**

Encoder RNN processes input sequence token by token, updating hidden state

2 **Context Vector**

Final encoder hidden state becomes context vector (fixed representation)

3 **Decoder Initialization**

Context vector initializes decoder RNN's hidden state

4 **Sequential Generation**

Decoder generates output tokens one at a time, using previous output as next input

5 **Termination**

Process continues until special end-of-sequence token is generated

🔢 Numerical Example with 3D Vectors

Weight Matrices (Encoder)

W (input weight, 3×3): `[[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]]`

U (hidden weight, 3×3): `[[0.4, 0.3, 0.2], [0.2, 0.5, 0.3], [0.3, 0.2, 0.6]]`

Input Sequence

x_1 = `[0.5, 0.3, 0.8]` → x_2 = `[0.2, 0.7, 0.4]`

⚠

Encoder Step 1

h_0 = `[0, 0, 0]` (initial hidden state)

$\mathbf{W} \cdot \mathbf{x_1}$ = [[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]] · [0.5, 0.3, 0.8] = `[0.55, 0.47, 0.77]`

$\mathbf{U} \cdot \mathbf{h_0}$ = `[0, 0, 0]`

h_1 = tanh([0.55, 0.47, 0.77]) ≈ `[0.50, 0.44, 0.65]`

Encoder Step 2

$\mathbf{W \cdot x_2}$ = [[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]] · [0.2, 0.7, 0.4] = `[0.38, 0.58, 0.48]`

$\mathbf{U \cdot h_1}$ = [[0.4, 0.3, 0.2], [0.2, 0.5, 0.3], [0.3, 0.2, 0.6]] · [0.50, 0.44, 0.65] = `[0.46, 0.42, 0.61]`

h_2 = tanh([0.38 + 0.46, 0.58 + 0.42, 0.48 + 0.61]) = tanh([0.84, 1.00, 1.09]) ≈ `[0.69, 0.76, 0.80]`

Context Vector c = h_2 = [0.69, 0.76, 0.80]

Weight Matrices (Decoder)

W' (input weight, 3×3): `[[0.6, 0.3, 0.2], [0.2, 0.5, 0.4], [0.4, 0.3, 0.5]]`

U' (hidden weight, 3×3): `[[0.5, 0.2, 0.3], [0.3, 0.6, 0.2], [0.2, 0.3, 0.7]]`

V (output weight, vocab_size×3): maps hidden to output vocabulary

Decoder Step 1

s_0 = `[0.69, 0.76, 0.80]` (initialized with context)

$\mathbf{V \cdot s_0}$ → softmax → y_1 (output token, e.g., word embedding)

Assume y_1 = `[1, 0, 0]` (one-hot encoded)

s_1 = tanh(W'·y_1 + U'·s_0) = tanh([0.6, 0.2, 0.4] + [0.82, 0.84, 0.91]) ≈ `[0.77, 0.72, 0.80]`

Decoder Step 2

$\mathbf{V \cdot s_1}$ → softmax → y_2 (next output token)

s_2 = tanh(W'·y_2 + U'·s_1) → continue...

Process continues until **<END>** token is generated

Encoder-Decoder Architecture

Encoder

Converts input sequence to context vector

(Typically LSTM/GRU)

→

Context Vector

Fixed-size representation of entire input

→

Decoder

Generates output conditioned on context

(Typically LSTM/GRU)

Training

Teacher forcing for faster convergence

Inference

Autoregressive generation

Both encoder and decoder typically use **LSTM** or **GRU** cells

Teacher Forcing

Training technique for sequence generation

👨‍🏫 Teacher Forcing

Uses **ground truth** as decoder input (not predictions)

🎯 Normal Inference

Uses **model's own predictions** as next input

⚙️ How It Works

1. **During Training:** Model ignores its own predictions and always uses ground truth as next input
2. **Example:** Translating "I love cats" → Even if model incorrectly predicts "나는" → Still provide correct "나는" as next input
3. **During Inference:** No ground truth available, so model uses its own predictions as next input → Initial errors can accumulate

🔢 Calculation Example: Sequence Generation (3D Input → 1D Output)

Scenario: Input vector $[x_1, x_2, x_3]$ → Generate number sequence (e.g., [2, 5, 3])

👨‍🏫 Teacher Forcing (Training)

t=1: Input: [1.2, 0.8, 2.1] → Model predicts: **4** → ✋ **Ignore!** → Use ground truth **2** as next input

t=2: Input: [2, 0.8, 2.1] → Model predicts: **6** → ✋ **Ignore!**

🎯 Normal Inference (Testing)

t=1: Input: [1.2, 0.8, 2.1] → Model predicts: **4** → ⚠️ **Use it!** → Use **4** as next input

t=2: Input: [4, 0.8, 2.1] → Model predicts: **7** → ⚠️ **Use it!**

→ Use ground truth **5** as next input

t=3: Input: [5, 0.8, 2.1] → Model predicts: **4** → ✋ **Ignore!**
→ Use ground truth **3** as next input

✅ Always learns with correct context → Fast convergence

→ Use **7** as next input

t=3: Input: [7, 0.8, 2.1] → Model predicts: **9** → ⚠️ **Use it!**
→ Output complete

⚠️ Initial errors accumulate → Diverges from ground truth [2,5,3] (Exposure Bias)

✓ Benefits

- Accelerates training convergence
- Faster learning

⚠ Problem

- **Exposure bias:** train/test mismatch
- Lower inference quality

Solution: Scheduled Sampling

Gradually use predictions → Trade-off between training speed and inference quality

Scheduled Sampling Implementation

- **Early stage:** 100% ground truth (Pure Teacher Forcing)
- **Gradual transition:** Use ground truth with probability p, model predictions with (1-p) (p decreases over time)
- **Late stage:** Mostly use model predictions → Similar to actual inference environment

Beam Search

Decoding Algorithm for Sequence Generation

Maintains k best candidates (beam width)

Balances **quality** and **computational cost** by keeping multiple candidate sequences

⚡

k = 1

Greedy Search

Fastest, Lower Quality

⚖️

k = 5-10

Typical Beam

Balanced Trade-off

🎯

k = Large

Wide Beam

Better Quality, Slower

📊 **Trade-off**

Larger beam width → Better quality but higher computation

💼 **Applications**

- Machine Translation
- Image Captioning
- Text Generation

How Beam Search Works

Algorithm Process

1 Initialization

Select k most probable candidates from the start token. Each candidate maintains its own probability score.

2 Expansion

Consider all possible next words for each candidate. This generates k candidates × vocabulary size possible sequences.

3 Scoring

Calculate cumulative probability for each expanded sequence. Typically uses sum of log probabilities for numerical stability.

4 Selection

Select only k candidates with highest cumulative scores from all expanded candidates to proceed to next step.

5 Iteration & Termination

Repeat steps 2-4 until end token is generated or maximum length is reached. Finally select the sequence with highest score.

Beam Search Loss Calculation Example

📊 3D Vector Input → 1D Output

Setup: Vocabulary size = 3 (words A, B, C), Beam width k = 2
At each time step, the model outputs a probability distribution over next words

🔢 Step 1: Initialization (t=0)

Model Output (Probability Vector): [0.5, 0.3, 0.2] → words [A, B, C]

Log Probabilities:
• log P(A) = log(0.5) = -0.693
• log P(B) = log(0.3) = -1.204
• log P(C) = log(0.2) = -1.609

Select k=2: A (-0.693), B (-1.204)
Current Candidates: ["A"], ["B"]

🔢 Step 2: First Expansion (t=1)

Sequence "A" next word probabilities: [0.4, 0.4, 0.2] → [A, B, C]
• A→A: -0.693 + log(0.4) = -0.693 + (-0.916) = **-1.609**
• A→B: -0.693 + log(0.4) = -0.693 + (-0.916) = **-1.609**
• A→C: -0.693 + log(0.2) = -0.693 + (-1.609) = -2.302

Sequence "B" next word probabilities: [0.6, 0.2, 0.2] → [A, B, C]

• B→A: -1.204 + log(0.6) = -1.204 + (-0.511) = -1.715
• B→B: -1.204 + log(0.2) = -1.204 + (-1.609) = -2.813
• B→C: -1.204 + log(0.2) = -1.204 + (-1.609) = -2.813

Select k=2 from 6 total candidates:
1st: "AA" (cumulative loss: -1.609)
2nd: "AB" (cumulative loss: -1.609)

🔢 Step 3: Second Expansion (t=2)

Sequence "AA" next word probabilities: [0.3, 0.5, 0.2] → [A, B, C]
• AA→A: -1.609 + log(0.3) = -1.609 + (-1.204) = -2.813
• AA→B: -1.609 + log(0.5) = -1.609 + (-0.693) = -2.302
• AA→C: -1.609 + log(0.2) = -1.609 + (-1.609) = -3.218

Sequence "AB" next word probabilities: [0.2, 0.6, 0.2] → [A, B, C]
• AB→A: -1.609 + log(0.2) = -1.609 + (-1.609) = -3.218
• AB→B: -1.609 + log(0.6) = -1.609 + (-0.511) = -2.120 ✨
• AB→C: -1.609 + log(0.2) = -1.609 + (-1.609) = -3.218

Final k=2 Selection:
🏆 1st: "ABB" (cumulative loss: -2.120) ← Final Choice!
2nd: "AAB" (cumulative loss: -2.302)

🎯 Key Points

- **Loss = Negative Log-Likelihood:** Lower is better (higher probability = lower loss)
- **Cumulative Calculation:** Add current log probability to previous cumulative value at each step

- **Beam Maintenance:** Select only top k from all possible expansions at each step
- **Final Result:** Sequence with lowest cumulative loss becomes the final output

Beam Search Concrete Example

📝 Translation Example: "I love AI" → Korean

Step 1: Start - Set k=3
Initial candidates: ["나는", "저는", "내가"] (Top 3 by probability)

Step 2: First Expansion
• "나는" → ["나는 사랑해", "나는 좋아해", "나는 즐겨"]
• "저는" → ["저는 사랑해", "저는 좋아합니다", "저는 즐깁니다"]
• "내가" → ["내가 사랑하는", "내가 좋아하는", "내가 즐기는"]
Select top 3 from 9 candidates

Step 3: Second Expansion
Selected 3: ["나는 사랑해", "저는 좋아합니다", "나는 좋아해"]
Add next word to each → Select top 3 again

Final: "나는 AI를 사랑해" (Highest cumulative probability)

$$\text{Score(sequence)} = \log P(w_1) + \log P(w_2|w_1) + \log P(w_3|w_1,w_2) + \ldots$$

🔑 Key Point

By maintaining only **k candidates** at each step, achieves a balance between exhaustive search (exponential) and greedy search.

Beam Search Advantages & Improvements

✅ Advantages

- Better quality results than Greedy
- More efficient than exhaustive search
- Parallelizable structure
- Applicable to various tasks
- Adjustable performance via beam width

⚠️ Limitations

- No guarantee of optimal solution
- Can be biased toward shorter sequences
- Memory usage proportional to k
- May generate repetitive phrases
- Diversity can be limited

🔧 Improvement Techniques

Length Normalization

Normalizes scores by sequence length to prevent bias toward shorter sequences. Divides score by length raised to a power.

Coverage Penalty

Penalizes already generated words to reduce repetition. Particularly effective in machine translation.

Diverse Beam Search

Divides beam into multiple groups, each exploring different candidates to increase diversity.

Beam Search Practical Application Guide

⚙️ Beam Width Selection Guide

k = 1: Real-time chatbots, quick prototyping
k = 3-5: General machine translation, balanced quality and speed
k = 10-20: High-quality translation, image captioning
k > 50: Research, benchmark testing (low practicality)

🎯 Task-specific Settings

- **Machine Translation:** k=5-10, length norm
- **Summarization:** k=3-5, coverage penalty
- **Image Captioning:** k=3-7
- **Dialogue Generation:** k=1-3, diversity focus
- **Code Generation:** k=5-10

💡 Performance Optimization Tips

- Parallelize with batch processing
- Set early stopping conditions
- Consider GPU memory efficiency
- Eliminate redundant calculations with caching
- Adjust beam width dynamically

🚀 Practical Checklist

1. **Identify Requirements:** Speed vs quality - which is more important?
2. **Initial Setup:** Start with small k and gradually increase
3. **Evaluation:** Measure performance with metrics like BLEU, ROUGE
4. **Tuning:** Adjust length normalization, temperature
5. **Validation:** Check result consistency across various inputs

⚖️ Core Trade-offs

Finding the balance between Quality ↔ Speed ↔ Diversity ↔ Memory Usage is key!

CTC Loss

Connectionist Temporal Classification

Alignment-Free Sequence Labeling

No need for frame-level annotations

Alignment-Free

Handles sequences without explicit alignment

No Frame Annotations

Only sequence-level labels needed

Blank Token

Introduces blank for alignment flexibility

Variable Length

Input/output can have different lengths

How It Works

Probability Distribution Generation

The neural network outputs a probability distribution over all possible characters (alphabet) at each time step, including a special 'blank' token.

Alignment Paths

Multiple alignment paths can produce the same output sequence. For example, "CAT" can be represented as "C-A-T", "CC-AAT", "-C-A-T-", and many other variations.

Key Idea

CTC sums the probabilities of all possible alignment paths that can be converted to the target sequence.

Computation Methodology

Step 1: Forward Algorithm

Progressing from left to right, at each time step, cumulatively calculate the probabilities of all paths that can reach each position in the target sequence.

Step 2: Backward Algorithm

Progressing from right to left, calculate the probabilities of all paths that reach the end of the sequence from each position.

Step 3: Marginalization

Combine forward and backward probabilities to calculate the total probability of all possible alignments that generate the target sequence. This is efficiently performed using dynamic programming.

Loss Function Calculation

CTC Loss = -log(Sum of probabilities of all correct alignment paths)

The training objective is to minimize this loss, thereby maximizing the probability of the correct output sequence.

Practical Calculation Example

Problem Setup

Target Output: "A" (1D sequence)

Input: 3 time steps (T=3)

Possible Characters: {A, blank(-)}

📊 Neural Network Output (Probability Distribution)

Time Step	P(A)	P(blank)
t=1	0.4	0.6
t=2	0.7	0.3
t=3	0.5	0.5

All Possible Alignment Paths

Calculate all possible paths and their probabilities that can produce the output "A":

Path 1: A - -

→ P(A at t=1) × P(- at t=2) × P(- at t=3)

→ 0.4 × 0.3 × 0.5 = **0.060**

Path 2: - A -

→ P(- at t=1) × P(A at t=2) × P(- at t=3)

→ 0.6 × 0.7 × 0.5 = **0.210**

Path 3: - - A

→ P(- at t=1) × P(- at t=2) × P(A at t=3)

→ 0.6 × 0.3 × 0.5 = **0.090**

Path 4: A A -

→ P(A at t=1) × P(A at t=2) × P(- at t=3)

→ 0.4 × 0.7 × 0.5 = **0.140**

Path 5: A - A

→ P(A at t=1) × P(- at t=2) × P(A at t=3)

→ 0.4 × 0.3 × 0.5 = **0.060**

Path 6: - A A

→ P(- at t=1) × P(A at t=2) × P(A at t=3)

→ 0.6 × 0.7 × 0.5 = **0.210**

Path 7: A A A

→ P(A at t=1) × P(A at t=2) × P(A at t=3)

→ 0.4 × 0.7 × 0.5 = **0.140**

✅ Final CTC Loss Calculation

Step 1: Sum probabilities of all paths

P(A) = 0.060 + 0.210 + 0.090 + 0.140 + 0.060 + 0.210 + 0.140 = **0.910**

Step 2: Calculate negative log probability

CTC Loss = -log(0.910) = **0.094**

💡 Key Insights

- High probability paths (0.210, 0.140) significantly influence the overall loss
- By considering all possible alignments, explicit alignment information is not required
- Through training, the total probability of correct outputs approaches 1 (Loss → 0)

🎯 Decoding Strategies

🏃 Greedy Decoding

Select the character with the highest probability at each time step, then remove consecutive duplicate characters and blanks. Fast but not always optimal.

🔍 Beam Search

Explore multiple candidate paths simultaneously to find the sequence with the highest overall probability. More accurate but computationally expensive.

🎓 Training Approach

Marginalizes over all possible alignments using dynamic programming

💼 Primary Applications

 Speech Recognition  Handwriting Recognition

Practical Implementation Tips

STABILITY

Gradient Clipping

Prevent exploding gradients during training

NORMALIZATION

Batch/Layer Normalization

Stabilize training and improve convergence

REGULARIZATION

Dropout

Apply to non-recurrent connections only

TRAINING

Learning Rate Scheduling

Use warmup + decay strategy

AUGMENTATION

Data Augmentation

Time warping, noise injection

MONITORING

Validation Metrics

Track to prevent overfitting

Applications and Next Steps

Current Applications

NLP
Transformers, BERT, GPT for language understanding

Time Series
Forecasting with attention mechanisms

Speech
End-to-end ASR with CTC and attention

Video
Action recognition, video captioning

Reinforcement Learning
Sequential decision making in dynamic environments

Future Directions

Transformers replacing RNNs

Temporal Graph Networks

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com