Lecture Information

Course: CSCI E-89B: Natural Language Processing
Lecture: Lecture 11
Topic: Sequence Models: HMMs, CRFs, and Generative Models
Date: Fall 2024

Contents

1 Introduction to Sequence Models

Overview

This lecture covers probabilistic sequence models used in NLP: Markov Chains, Hidden Markov Models (HMMs), and Conditional Random Fields (CRFs). We also explore how to combine these with neural networks (BiLSTM-CRF) and introduce generative models like VAEs and GANs.

2 Markov Chains

Overview

Markov chains model sequences where the probability of each element depends only on the previous element. This "memoryless" property, called the Markov property, simplifies modeling but limits expressiveness.

2.1 Definition and Structure

Markov Chain

A **Markov chain** is a sequence of random variables $X_1, X_2, \ldots, X_T$ satisfying the **Markov property**:

$$P(X_{t+1}|X_t, X_{t-1}, \ldots, X_1) = P(X_{t+1}|X_t)$$

The future depends only on the present, not on the past.

Weather as Markov Chain

States: {Sunny, Rainy, Cloudy}
Transition probabilities:

From/To	Sunny	Rainy	Cloudy
Sunny	0.7	0.1	0.2
Rainy	0.2	0.5	0.3
Cloudy	0.3	0.3	0.4

Each row sums to 1 (must transition somewhere).

2.2 Transition Probabilities

Transition Matrix

For states $\{1, 2, \ldots, N\}$, the **transition probability** from state i to state j is:

$$P_{ij} = P(X_{t+1} = j|X_t = i)$$

Constraints:

- $P_{ij} \geq 0$ (non-negative)
- $\sum_{j=1}^{N} P_{ij} = 1$ (rows sum to 1)

2.3 NLP Application: Language Modeling

Bigram Language Model

Treat words as states. Transition probabilities model word sequences:

$$P(\text{“sat”}|\text{“cat”}) = 0.15$$

A sentence's probability:

$$P(\text{“the cat sat”}) = P(\text{the}) \cdot P(\text{cat}|\text{the}) \cdot P(\text{sat}|\text{cat})$$

2.4 Limitations of Markov Chains

The Markov Assumption is Limiting

- Only previous word matters—ignores long-range dependencies
- “The cat that I saw yesterday sat” vs “The cats that I saw yesterday sat”
- Solution: Use n-grams (state = last n-1 words) or neural models

3 Hidden Markov Models (HMMs)

Overview

HMMs extend Markov chains by introducing **hidden states** that generate **observations**. We observe the outputs but not the underlying state sequence.

3.1 Motivation

Why Hidden States?

In language, we observe words but not their underlying structure:

- **Observed**: “The cat sat on the mat”
- **Hidden**: DET NOUN VERB PREP DET NOUN (part-of-speech tags)

The hidden states (POS tags) follow Markov dynamics, and each state “emits” an observed word.

3.2 HMM Structure

Hidden Markov Model Components

An HMM consists of:

1. **Hidden states**: $Y_1, Y_2, \ldots, Y_T$ (e.g., POS tags)
2. **Observations**: $X_1, X_2, \ldots, X_T$ (e.g., words)
3. **Transition probabilities**: $P(Y_{t+1} = j|Y_t = i) = A_{ij}$
4. **Emission probabilities**: $P(X_t = x|Y_t = j) = B_{jx}$

5. **Initial state distribution**: $\pi_i = P(Y_1 = i)$

POS Tagging as HMM

Hidden states: {NOUN, VERB, DET, ADJ, PREP, ...}
Observations: {the, cat, sat, on, mat, ...}
Transitions (e.g.):

- $P(\text{VERB}|\text{NOUN}) = 0.35$ (nouns often followed by verbs)
- $P(\text{NOUN}|\text{DET}) = 0.60$ (determiners often followed by nouns)

Emissions (e.g.):

- $P(\text{“cat”}|\text{NOUN}) = 0.002$
- $P(\text{“sat”}|\text{VERB}) = 0.001$

3.3 HMM Parameters

HMM Parameter Summary

$$\lambda = (A, B, \pi)$$

where:

- A: Transition matrix ($N \times N$ for N hidden states)
- B: Emission matrix ($N \times V$ for vocabulary size V)
- π: Initial state distribution (length N)

3.4 Training HMMs

Important

If hidden states are observed (supervised training):

- Count transitions and emissions directly
- Estimate probabilities via maximum likelihood

If hidden states are NOT observed (unsupervised):

- Use **Baum-Welch algorithm** (a form of EM)
- E-step: Estimate expected counts of transitions/emissions
- M-step: Update parameters from expected counts

3.5 Decoding: Finding Hidden States

Viterbi Algorithm

Given observations $X_1, \ldots, X_T$, find the most likely hidden state sequence:

$$\hat{Y}_{1:T} = \arg\max_{Y_{1:T}} P(Y_{1:T}|X_{1:T})$$

The **Viterbi algorithm** uses dynamic programming to find this efficiently in $O(T \cdot N^2)$ time.

3.6 HMM Limitations

HMM Limitations

1. **Markov assumption**: Hidden state depends only on previous state
2. **Independence assumption**: Observation depends only on current hidden state
3. **No future context**: Can't use future words to help label current word

4 Conditional Random Fields (CRFs)

Overview

CRFs address HMM limitations by modeling $P(Y|X)$ directly (discriminative) rather than the joint $P(X, Y)$ (generative). They allow arbitrary features and bidirectional dependencies.

4.1 From HMMs to CRFs

Key Differences: HMM vs CRF

Aspect	HMM	CRF
Model type	Generative	Discriminative
Models	$P(X, Y)$	$P(Y\|X)$
Direction	Forward only	Bidirectional
Features	Emissions only	Arbitrary features
Independence	Strong assumptions	Flexible

4.2 CRF Model

Linear-Chain CRF

$$P(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{t=1}^{T}\sum_{k} \lambda_k f_k(y_{t-1}, y_t, X, t)\right)$$

where:

- f_k: Feature functions (manually designed)
- λ_k: Feature weights (learned)

- $Z(X)$: Normalization constant (partition function)

4.3 Feature Functions

CRF Feature Functions

Feature functions $f_k(y_{t-1}, y_t, X, t)$ encode patterns. They typically return 0 or 1:
Example features for NER:

- $f_1 = 1$ if $y_t =$ PERSON and $x_{t-1} =$ "Mr."
- $f_2 = 1$ if $y_t =$ PERSON and x_t starts with capital letter
- $f_3 = 1$ if $y_t =$ ORG and x_t ends with "Inc."
- $f_4 = 1$ if $y_{t-1} =$ B-PERSON and $y_t =$ I-PERSON

Concrete Feature Example

For the input "Mr. Smith works at Apple Inc.":
Feature: "If previous word is 'Mr.' and current word is capitalized, likely PERSON"

$$f_1(y_{t-1}, y_t, X, t) = \mathbf{1}[x_{t-1} = \text{“Mr.”} \wedge x_t[0] \in \text{A-Z} \wedge y_t = \text{PERSON}]$$

The weight λ_1 is learned from data—higher weight means this pattern is more predictive.

4.4 Advantages of CRFs

Important

CRF Advantages:

1. **Arbitrary features**: Include any information about entire input
2. **Global normalization**: Avoids label bias problem
3. **Bidirectional context**: Feature can look at future words
4. **No independence assumptions**: Features can overlap

4.5 Disadvantages of CRFs

CRF Disadvantages

1. **Manual feature engineering**: Must design features by hand
2. **Computational cost**: Training can be expensive
3. **Feature explosion**: Many features needed for good performance

5 BiLSTM-CRF

Overview

BiLSTM-CRF combines the automatic feature learning of neural networks with the sequence modeling of CRFs. The BiLSTM replaces hand-crafted features; the CRF layer captures label dependencies.

5.1 Architecture

BiLSTM-CRF Architecture

1. **Embedding Layer**: Words → dense vectors
2. **BiLSTM Layer**: Captures bidirectional context
3. **Linear Layer**: Maps hidden states to "emission scores"
4. **CRF Layer**: Models label transitions, outputs final labels

5.2 How It Works

Important

Step 1: Embeddings
Each word x_t is mapped to an embedding vector e_t.
Step 2: BiLSTM
Forward and backward LSTMs process the sequence:

$$\overrightarrow{h_t} = \text{LSTM}_{\rightarrow}(e_t, \overrightarrow{h_{t-1}})$$

$$\overleftarrow{h_t} = \text{LSTM}_{\leftarrow}(e_t, \overleftarrow{h_{t+1}})$$

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

Step 3: Emission Scores
Linear transformation produces scores for each label:

$$E_t = W \cdot h_t + b$$

E_t has dimension equal to number of labels (e.g., B-PER, I-PER, O, ...).
Step 4: CRF Layer
CRF uses emission scores E and learns transition matrix T (label-to-label scores). Final prediction maximizes:

$$\text{score}(X, Y) = \sum_{t=1}^{T} E_{t,y_t} + \sum_{t=1}^{T-1} T_{y_t,y_{t+1}}$$

5.3 Why CRF on Top of BiLSTM?

CRF Layer Benefits

Without CRF, BiLSTM predicts each label independently. This can produce invalid sequences like:

- I-PER following O (invalid—can't continue without begin)
- B-LOC immediately after B-PER (missing I-PER)

The CRF layer learns that certain transitions are unlikely (e.g., $T_{\text{O,I-PER}} \ll 0$), enforcing valid sequences.

5.4 Implementation Sketch

```python
import torch
import torch.nn as nn
from torchcrf import CRF

class BiLSTM_CRF(nn.Module):
    def __init__(self, vocab_size, tag_size, embed_dim, hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_dim // 2,
                            bidirectional=True, batch_first=True)
        self.linear = nn.Linear(hidden_dim, tag_size)
        self.crf = CRF(tag_size, batch_first=True)

    def forward(self, x):
        embeds = self.embedding(x)
        lstm_out, _ = self.lstm(embeds)
        emissions = self.linear(lstm_out)
        return emissions

    def loss(self, x, tags):
        emissions = self.forward(x)
        return -self.crf(emissions, tags)  # Negative log-likelihood

    def predict(self, x):
        emissions = self.forward(x)
        return self.crf.decode(emissions)  # Viterbi decoding
```

Listing 1: BiLSTM-CRF in PyTorch (Simplified)

6 Variational Autoencoders (VAEs)

Overview

VAEs are generative models that learn a structured latent space. Unlike regular autoencoders, VAEs can generate new, realistic samples by sampling from the latent space.

6.1 Regular Autoencoder Problem

Unstructured Latent Space

In a regular autoencoder:

- Encoder compresses input to latent code z
- Decoder reconstructs input from z
- Problem: Latent space is **unstructured**

If you move slightly away from a learned encoding, the decoder produces garbage—no smooth interpolation between points.

6.2 VAE Solution

VAE Key Idea

Instead of encoding to a point, encode to a **distribution**:

1. Encoder outputs μ (mean) and σ (standard deviation)
2. Sample $z \sim \mathcal{N}(\mu, \sigma^2)$
3. Decoder reconstructs from sampled z

This forces nearby points in latent space to also decode to realistic outputs.

6.3 VAE Loss Function

VAE Loss

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|^2}_{\text{Reconstruction loss}} + \underbrace{D_{KL}(\mathcal{N}(\mu, \sigma^2) \| \mathcal{N}(0, 1))}_{\text{KL divergence (regularization)}}$$

The KL term forces distributions to stay close to standard normal $\mathcal{N}(0, 1)$.

6.4 Why KL Divergence?

Important

Without KL regularization:

- Network minimizes reconstruction by shrinking $\sigma \to 0$
- Returns to point encoding—loses structure

With KL regularization:

- Forces $\mu \to 0$ and $\sigma \to 1$
- Distributions overlap, creating smooth latent space
- Can interpolate between encodings

7 Generative Adversarial Networks (GANs)

Overview

GANs learn to generate realistic data through adversarial training: a generator tries to fool a discriminator, while the discriminator tries to distinguish real from fake data.

7.1 The Chess Analogy

Learning Without a Teacher

Problem: Train children to play chess without an expert teacher.
Solution: Have them play against each other! Both improve through competition.
GANs work similarly: two networks compete, both improving without labeled data.

7.2 GAN Architecture

GAN Components

Generator G:

- Input: Random noise $z \sim \mathcal{N}(0, 1)$
- Output: Fake sample $G(z)$
- Goal: Generate samples indistinguishable from real data

Discriminator D:

- Input: Real sample x or fake sample $G(z)$
- Output: Probability that input is real
- Goal: Correctly classify real vs fake

7.3 GAN Training

GAN Objective (Minimax Game)

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

- D maximizes: classify real as real, fake as fake
- G minimizes: fool D into thinking fake is real

7.4 Training Procedure

Important

Alternating optimization:

1. **Train D**: Fix G, update D to better distinguish real/fake
2. **Train G**: Fix D, update G to better fool D

3. Repeat until equilibrium

At equilibrium: D outputs 0.5 for everything (can't tell real from fake), G generates perfect samples.

7.5 Mode Collapse Problem

Mode Collapse

GANs can suffer from **mode collapse**:

- Generator finds one output that fools discriminator
- Keeps producing only that output (e.g., only shoes)
- Discriminator adapts, generator switches to another mode
- Cycle continues without learning diversity

Solutions: Experience replay, progressive growing, StyleGAN architecture

8 One-Page Summary

Summary

Markov Chains: Sequence model where $P(X_{t+1}|X_t, \ldots) = P(X_{t+1}|X_t)$. Simple but limited—no long-range dependencies.
Hidden Markov Models (HMMs):

- Hidden states Y generate observations X
- Parameters: transitions A, emissions B, initial π
- Training: Baum-Welch (EM) if hidden states unknown
- Decoding: Viterbi algorithm
- Limitation: Only uses past context

Conditional Random Fields (CRFs):

- Discriminative: models $P(Y|X)$ directly
- Feature functions $f_k(y_{t-1}, y_t, X, t)$ encode patterns
- Bidirectional context, no independence assumptions
- Disadvantage: Manual feature engineering

BiLSTM-CRF:

- BiLSTM provides automatic feature learning
- CRF layer captures label dependencies
- State-of-the-art for NER, POS tagging before transformers

- No manual features needed

Variational Autoencoders (VAEs):

- Encode to distribution (μ, σ), sample, decode
- KL divergence regularization prevents collapse
- Creates structured, interpolatable latent space

GANs:

- Generator vs Discriminator adversarial game
- No labeled data needed—self-supervised
- Mode collapse is common challenge
- Can generate highly realistic images/text

9 Glossary

Key Terms

- **Markov Property**: Future depends only on present, not past
- **HMM**: Hidden Markov Model—hidden states emit observations
- **Transition Probabilities**: $P(Y_{t+1}|Y_t)$
- **Emission Probabilities**: $P(X_t|Y_t)$
- **Viterbi Algorithm**: Dynamic programming for most likely path
- **Baum-Welch**: EM algorithm for HMM parameter estimation
- **CRF**: Conditional Random Field—discriminative sequence model
- **Feature Function**: Pattern indicator in CRF
- **Partition Function**: Normalization constant $Z(X)$
- **BiLSTM**: Bidirectional LSTM—forward and backward context
- **Emission Scores**: BiLSTM output before CRF layer
- **VAE**: Variational Autoencoder—generative with structured latent space
- **KL Divergence**: Measures distribution similarity
- **Latent Space**: Compressed representation space
- **GAN**: Generative Adversarial Network
- **Generator**: Creates fake samples from noise
- **Discriminator**: Classifies real vs fake
- **Mode Collapse**: GAN failure mode—limited diversity

- **Discriminative Model**: Models $P(Y|X)$
- **Generative Model**: Models $P(X, Y)$ or $P(X)$