End Module C Exam Handbook

Autoregressive Models

Understanding Autoregressive Models

Definition: Autoregressive (AR) models predict future values in a time series based on a linear function of previous observations. They are widely used in time-series forecasting, econometrics, and natural language processing.

Equation:

$$X_t = b + \sum_{i=1}^{p} w_i X_{t-i} + \epsilon_t \tag{1}$$

$$X_t = b + w_1 X_{t-1} \quad \text{(AR(1))} \tag{2}$$

$$X_t = b + w_1 X_{t-1} + w_2 X_{t-2} \quad \text{(AR(2))} \tag{3}$$

State Transition Matrix:

$$\begin{bmatrix} X_t \\ X_{t-1} \\ \vdots \\ X_{t-p+1} \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & \dots & w_p \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{t-1} \\ X_{t-2} \\ \vdots \\ X_{t-p} \end{bmatrix} + \begin{bmatrix} b \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4}$$

Sequence Modelling Auto-Regressive Models

Definition: A time-series model where current values depend on past values.

Equation:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{5}$$

Special Case: AR(1):

$$X_t = \phi X_{t-1} + \epsilon_t \tag{6}$$

Markov Chain Connection:

- Transition probability: $P(X_{t+1}|X_t)$
- Can be represented as a transition matrix.

Binary Cross-Entropy Loss (BCELoss)

Definition

Binary Cross-Entropy (BCE) is a widely used loss function for binary classification tasks. It measures the difference between two probability distributions and is commonly used in logistic regression and neural networks.

Mathematical Formulation:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y_i} + (1 - y_i)\log(1 - \hat{y_i})\right] \tag{7}$$

Properties of BCE Loss

- Penalizes incorrect predictions heavily.
- Ensures stable training with a small ϵ to avoid log(0) errors.
- Works well for probabilistic models in classification tasks.

NLP Pipeline

Natural Language Processing (NLP) involves preprocessing textual data before feeding it into machine learning models. The key steps include:

Tokenization

- Splitting text into words or subwords.
- Helps in feature extraction for models.
- Example: Hello, world!" → [Hello", ," , world", "!"]

Part-of-Speech (POS) Tagging

- Assigns grammatical roles to words (noun, verb, adjective, etc.).
- Used in syntactic parsing and named entity recognition.
- Example: The cat sleeps" → [(The", DET), (cat", NOUN), (sleeps", VERB)]

Text Normalization

- Converts words into a standard format to improve model performance.
- Includes lowercasing, stemming, and lemmatization.

Lowercasing

- Converts text to lowercase to avoid treating Hello" and hello" as different words.

Stemming

- Reduces words to their root form by removing suffixes.
- Example: running" $\rightarrow$ run"

Lemmatization

- Converts words to their dictionary form.
- Example: better" $\rightarrow$ good"

Stopword and Punctuation Removal

- Removes commonly occurring words (e.g., the", is", "and") that do not add significant meaning.
- Helps reduce dimensionality and improve model performance.

Understanding N-Grams

Definition: A sequence of N items from a given text.
Types:

- Unigram: $\{word_1, word_2, ...\}$
- Bigram: $\{word_1 word_2, word_2 word_3, ...\}$
- Trigram: $\{word_1 word_2 word_3, ...\}$

Probability Calculation:

$$P(w_n|w_{n-1}) = \frac{count(w_{n-1}, w_n)}{count(w_{n-1})} \tag{8}$$

$$P(w_n|w_{n-2}, w_{n-1}) = \frac{count(w_{n-2}, w_{n-1}, w_n)}{count(w_{n-2}, w_{n-1})} \tag{9}$$

Recurrent Neural Networks (RNNs)

Definition

A Recurrent Neural Network (RNN) is a type of artificial neural network designed to process sequential data. Unlike traditional feedforward networks, RNNs use internal memory (hidden states) to capture dependencies in sequential inputs.

Basic Structure:

- Input Layer: Accepts sequential data.
- Hidden Layer with Loops: Stores past information.
- Output Layer: Produces predictions.

Mathematical Formulation:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \tag{10}$$

$$o_t = W_o h_t \tag{11}$$

Types of RNNs:

- One-to-One: Standard feedforward network.
- One-to-Many: Single input, multiple outputs.
- Many-to-One: Multiple inputs, single output.
- Many-to-Many: Sequence-to-sequence tasks.

Backpropagation Through Time (BPTT):

- Used to train RNNs by unrolling through time and applying backpropagation.

Challenges of RNNs

- **Vanishing Gradient Problem**: Gradients shrink exponentially over time, making it difficult for the network to learn long-term dependencies.
- **Exploding Gradient Problem**: Gradients grow exponentially, leading to unstable training.
- **Limited Memory**: Standard RNNs struggle with capturing long-range dependencies in sequential data.

Long Short-Term Memory (LSTM)

Definition

LSTMs are an advanced type of RNN designed to address the vanishing gradient problem by incorporating memory cells that selectively retain information over long sequences.

LSTM Architecture

LSTM networks consist of three gates:

- **Forget Gate**: Controls what portion of the past information should be discarded.
- **Input Gate**: Regulates what new information should be added to memory.
- **Output Gate**: Determines the final output based on the cell state.

Mathematical Formulation: Forget Gate:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{12}$$

Input Gate:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{13}$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{14}$$

Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{15}$$

Output Gate:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{16}$$

$$h_t = o_t * \tanh(C_t) \tag{17}$$

Gated Recurrent Units (GRUs)

Definition

GRUs are a variant of LSTMs that use gating mechanisms to control information flow but require fewer parameters than LSTMs.

Mathematical Formulation

Reset Gate:

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{18}$$

Update Gate:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{19}$$

Candidate Hidden State:

$$\tilde{h}_t = \tanh(W_h[r_t * h_{t-1}, x_t] + b_h) \tag{20}$$

Final Hidden State:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{21}$$

Bidirectional RNNs

Definition

A Bidirectional RNN (BiRNN) processes input sequences in both forward and backward directions to improve context understanding.

Mathematical Formulation:

$$h_t^{forward} = f(W_x x_t + U h_{t-1}^{forward}) \tag{22}$$

$$h_t^{backward} = f(W_x x_t + U h_{t+1}^{backward}) \tag{23}$$

$$h_t = [h_t^{forward}, h_t^{backward}] \tag{24}$$

Applications of RNNs, LSTMs, and GRUs

- **Natural Language Processing (NLP)**: Machine translation, text generation, and sentiment analysis.
- **Speech Recognition**: Transcribing spoken language into text.
- **Time-Series Forecasting**: Predicting financial market trends and weather patterns.
- **Bioinformatics**: Analyzing DNA sequences and protein structures.

Attention Mechanism for NLP

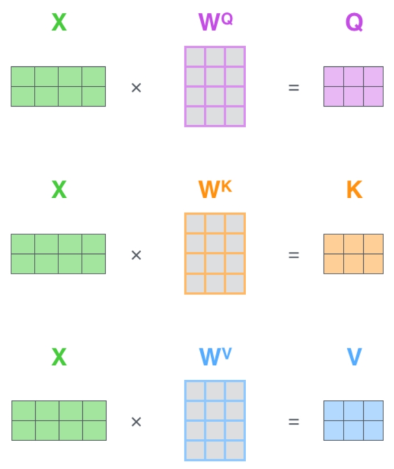

FIGURE 8.8: Input transformation used in a Transformer, $Q = \text{Query}$, $K = \text{Key}$, $V = \text{Value}$. Image Source: (Alammar 2018)

Figure 1: Input transformation used in a Transformer. The input matrix X is multiplied by weight matrices W^Q, W^K, and W^V to produce Query (Q), Key (K), and Value (V) matrices.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

and projections are the parameter matrices:

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \quad W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

The attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Sentiment Classification

Definition:
Sentiment classification is the task of identifying the emotional tone behind a body of text, typically categorized as **positive**, **negative**, or **neutral**.

Goal:
To automatically determine the sentiment expressed in text using computational methods.

Applications:

- Product reviews
- Social media monitoring
- Customer feedback analysis

Approaches:

1. **Rule-based:** Uses manually defined rules and sentiment lexicons (e.g., SentiWordNet).
2. **Machine Learning:** Trains models (e.g., Naive Bayes, SVM) on labeled data.
3. **Deep Learning:** Uses neural networks (e.g., RNNs, LSTMs, Transformers) for better context understanding.

Steps Involved:

- **Text Preprocessing:** Tokenization, stopword removal, stemming/lemmatization
- **Feature Extraction:** Bag of Words, TF-IDF, word embeddings
- **Model Training:** Using labeled data to train classifiers
- **Prediction:** Classify unseen text into sentiment categories

Popular Tools/Libraries:

- NLTK
- TextBlob
- Scikit-learn
- Hugging Face Transformers

Word Embeddings

Definition:
Word embeddings are dense vector representations of words in a continuous vector space where semantically similar words are mapped close together.

Purpose:
To capture syntactic and semantic meaning of words for use in machine learning models.

Characteristics:

- Low-dimensional (e.g., 100–300 dimensions)
- Real-valued vectors
- Context-based representations

Popular Techniques:

- **Word2Vec:** Uses two architectures:
  - **CBOW (Continuous Bag of Words):** Predicts the current word from surrounding context.
  - **Skip-gram:** Predicts surrounding context words given the current (center) word.
- **GloVe (Global Vectors):** Combines global word co-occurrence statistics.
- **FastText:** Improves Word2Vec by using subword information.
- **BERT Embeddings:** Contextual embeddings generated using transformer models.

Advantages:

- Captures semantic similarity (e.g., `king - man + woman = queen`)
- Improves performance in NLP tasks

Applications:

- Text classification
- Machine translation
- Question answering
- Named entity recognition (NER)

BLEU Score (Bilingual Evaluation Understudy)

Definition:
BLEU is an automatic evaluation metric for comparing machine-generated text (e.g., translations) against one or more reference texts using n-gram precision.

Purpose:
To measure the quality of machine translation by checking how many n-grams in the candidate sentence appear in the reference sentences.

Formula:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where:

- p_n = modified precision for n-grams (usually $n = 1$ to 4)
- w_n = weight for n-gram (commonly $w_n = \frac{1}{N}$)
- BP = brevity penalty to penalize short candidates

Brevity Penalty:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

- c = length of candidate translation
- r = effective reference length

Machine Translation (MT)

Definition:
Machine Translation is the task of automatically translating text from one language to another using computational models.

Evolution:

- **Statistical MT:** Early systems based on word/phrase probabilities.
- **Seq2Seq Models:** Encoder-decoder neural networks using RNNs.
- **Attention Mechanism:** Solves long-range dependency issues in Seq2Seq.
- **Transformers:** Fully attention-based models, state-of-the-art in MT.

Preprocessing Pipeline:

- **Tokenization:** Splits text into meaningful units using language-specific tokenizers.
- **Vocabulary Building:** Maps words to indices; includes special tokens like `<sos>`, `<eos>`, `<pad>`, and `<unk>`.
- **Numericalization:** Converts token sequences into index sequences for model input.

Introduction to Transformers

Transformers revolutionized machine translation by overcoming the limitations of RNNs:

- **RNN Issues:** Struggles with long-range dependencies, poor parallelization, and fading context.
- **Transformer Strengths:** Self-attention allows direct modeling of dependencies, facilitates parallel computation, and maintains context across sentences.

Positional Encoding

Transformers are permutation-invariant. Positional encodings are added to token embeddings to provide order information.

Encoder

The encoder processes input sequences and creates contextualized embeddings. It consists of the following components:

- Input embedding

- Positional encoding
- Multi-head self-attention
- Feed-forward network
- Layer normalization + residual connections

Decoder

The decoder generates the output sequence by attending to both previous tokens and the encoder's output. It consists of the following components:

- Output embedding
- Positional encoding
- Masked multi-head self-attention
- Encoder-decoder attention
- Feed-forward network
- Linear + softmax layer

Complete Transformer Model

The Transformer model consists of:

- **Encoder:** Processes the source sequence in parallel.
- **Decoder:** Generates the target sequence, attending to the encoder's output.
- **Training:** Uses teacher forcing with a shifted version of the target sequence.

Semi-Supervised Learning (SSL)

Introduction

Semi-Supervised Learning (SSL) combines a small amount of labeled data with a large pool of unlabeled data. It strikes a balance between supervised learning (using labeled data) and unsupervised learning (using only unlabeled data), leveraging both to improve model performance.

Motivation

Labeled data is costly to acquire, while unlabeled data is abundant. SSL utilizes unlabeled data to enhance learning, reducing reliance on labeled data.

Mathematical Formulation

Let:

- $D_L = \{(x_i, y_i)\}_{i=1}^{l}$: Labeled data
- $D_U = \{x_i\}_{i=l+1}^{l+u}$: Unlabeled data

The objective is to minimize:

$$\mathcal{L}(f) = \mathcal{L}_{\text{supervised}}(f; D_L) + \mathcal{L}_{\text{unsupervised}}(f; D_U)$$

Importance

SSL reduces the need for large labeled datasets, making it particularly useful in domains with limited labeled data but abundant unlabeled data.

Key Assumptions in SSL

- **Self-Training**: Confident predictions on unlabeled data are added to the training set.
- **Co-Training**: Two classifiers, trained on different views, teach each other.
- **Cluster Assumption**: Points in the same cluster likely share the same label.
- **Manifold Assumption**: Data points lie on a low-dimensional manifold.

Inductive vs Transductive SSL

- **Inductive SSL**: Learns a general classifier applicable to unseen data.
- **Transductive SSL**: Focuses on labeling a fixed set of unlabeled data.

Ladder Networks and -Models (Pi-Models)

Ladder Networks

A Ladder Network combines supervised and unsupervised learning by denoising internal representations:

- **Encoder**: Adds noise to the input and processes it.
- **Decoder**: Recovers clean activations for each layer.
- **Skip Connections**: Form a "ladder" structure between encoder and decoder.

Loss function:

$$\mathcal{L} = \mathcal{L}_{\text{supervised}} + \sum_l \lambda_l \mathcal{L}_{\text{reconstruction}}(l)$$

Where λ_l is a weight for each layer's reconstruction loss.

Pi-Model

The Pi-Model enforces consistency regularization by ensuring stable predictions under input perturbations:

- Input x is passed through the network twice with different augmentations/noise.
- Two outputs: $f_1(x + \epsilon_1)$ and $f_2(x + \epsilon_2)$.

Loss function:

$$\mathcal{L}_{\text{unsupervised}} = \|f_1(x) - f_2(x)\|^2$$

This encourages predictions to remain consistent under small input changes.

Variational Autoencoders (VAEs)

Introduction

VAEs model latent representations as probability distributions, enabling data generation by sampling from the learned distribution.

Intuition

The encoder compresses an input into a latent variable z, treated as a random variable. This enables diverse outputs from the same input.

VAE Loss

$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - \mathrm{KL}(q(z|x) \parallel p(z))$$

Consists of:

- **Reconstruction Loss**
- **KL Divergence**

Why Use a Distribution?

Modeling z as a distribution allows infinite variants of data generation.

Diffusion Models

Diffusion models reverse noise addition across steps for sharper outputs, unlike VAEs' single-step approach.

VAEs in SSL

Extended with a classifier, using reconstruction, KL divergence, and classification losses for labeled data, and pseudo-labels for unlabeled data.

Applications

- **Medical Imaging**: Small labeled datasets, synthetic augmentation.
- **Autonomous Driving**: Unlabeled video frames.

Conditional VAEs

CVAEs guide generation by conditioning on a label y, e.g., generating a digit "4" by conditioning on $y = 4$.

Introduction to Reinforcement Learning

Definition and Basic Concepts

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions through interaction with an environment to maximize rewards.

Comparison with Other Learning Paradigms

Key Components of Reinforcement Learning

Agent

Learns and makes decisions to maximize cumulative reward.

Environment

Responds to agent's actions, provides feedback, and can be deterministic or stochastic.

State/Observation

Represents the environment's current situation. States can be complete or partial.

Action

Decisions made by the agent that affect the environment.

Reward

Scalar feedback indicating the quality of actions taken by the agent.

Policy

Strategy that defines how actions are selected.

Exploration vs. Exploitation Dilemma

Balancing between exploring new actions and exploiting known good actions.

Common Approaches

- ϵ-greedy
- Decaying ϵ-greedy
- Softmax exploration
- Upper Confidence Bound (UCB)
- Thompson Sampling
- Intrinsic Motivation/Curiosity

Types of Reinforcement Learning Algorithms

- Model-based vs. Model-free
- Value-based vs. Policy-based
- On-policy vs. Off-policy
- Deterministic vs. Stochastic Policies

Concepts in Reinforcement Learning

Episodes

An episode is a sequence from an initial state to a terminal state:

- Finite in length, with a defined start and endpoint.

State Spaces

The state space is the set of all possible states:

- **Discrete**: Finite number of states
- **Continuous**: Infinite number of states.
- **Fully Observable**: Complete state visible to the agent.
- **Partially Observable**: Only partial state visible to the agent.

Markov Property

The Markov property states that the future depends only on the current state:

- Forms the foundation of Markov Decision Processes (MDPs).
- Simplifies learning by discarding past history.

Challenges in Reinforcement Learning

Delayed Rewards

Challenges with delayed rewards:

- **Credit Assignment Problem**: Determining which actions led to the final reward.
- Approaches: Temporal Difference Learning, Eligibility Traces, Reward Shaping, Hierarchical RL.

Continuous vs. Discrete Action Spaces

- **Discrete Actions**: Finite set (e.g., Q-learning).
- **Continuous Actions**: Infinite set (e.g., DDPG, SAC, PPO).

Multi-Armed Bandit Problem (MAB)

Problem Overview

The goal in the multi-armed bandit (MAB) problem is to maximize the expected total reward by selecting actions (arms) over time, while facing uncertainty in reward distributions. We aim to learn which actions yield better rewards.

Reward Estimation

Each arm corresponds to an unknown reward distribution. We estimate the expected reward using the sample mean:

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{N_t(a)} R_i$$

where $Q_t(a)$ is the estimated value of arm a at time t, and $N_t(a)$ is the number of times arm a has been selected up to time t.

Incremental Update Rule

To avoid storing all past rewards, we update the estimated value incrementally:

$$Q_n = Q_{n-1} + \frac{1}{n}(R_n - Q_{n-1})$$

where Q_n is the updated estimate after observing reward R_n at step n.

Greedy Algorithm

The greedy algorithm selects the arm with the highest estimated reward:

$$\text{Action} = \arg\max_a Q_t(a)$$

However, it may converge prematurely to suboptimal arms and avoid exploring.

Epsilon-Greedy Algorithm

The Epsilon-Greedy algorithm balances exploration and exploitation:

- With probability ϵ, select an arm randomly (exploration).
- With probability $1-\epsilon$, select the arm with the highest $Q_t(a)$ (exploitation).

The exploration rate ϵ controls the trade-off.

Upper Confidence Bound (UCB)

The UCB algorithm explores uncertain arms more frequently. The UCB1 formula is:

$$\text{UCB}_t(a) = Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}}$$

where c is the exploration coefficient. Arms with higher uncertainty are explored earlier.

Markov Decision Processes (MDPs)

State-value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Expected return starting from state s under policy π.

Action-value function:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

Expected return from state s, taking action a, then following π.

Solving MDPs with Dynamic Programming

Bellman Expectation Equation (Policy Evaluation):

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R + \gamma V^{\pi}(s')]$$

Iteratively update value estimates using current policy.

Value Iteration:

$$V(s) = \max_{a} \sum_{s'} P(s'|s,a)[R + \gamma V(s')]$$

Optimal value found by repeatedly applying Bellman optimality.

Monte Carlo Methods

MC Value Estimation:

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i$$

Average return over episodes starting from s.

Learn from complete episodes without model of environment.

Temporal Difference (TD) and Deep Reinforcement Learning

TD(0) Update:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

Update value using immediate reward and next state's estimate.

SARSA (On-policy):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

Learn action values while following the same policy.

Q-Learning (Off-policy):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Learn optimal policy regardless of the behavior policy.

DQN Loss Function:

$$L(\theta) = \Big(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta)\Big)^2$$

Neural network predicts Q-values; loss guides learning.