ML Algorithm Study Guide

Deep Learning & Classical ML for Staff/Principal Interviews

Overview

This study guide accompanies the *ML Algorithm Templates* cheatsheet. Use this document to:

- Master deep learning architectures (CNNs, RNNs, Transformers, GANs)
- Solidify classical ML fundamentals (trees, boosting, SVMs)
- Track progress with spaced repetition for ML concepts
- Prepare systematically for Staff/Principal ML Engineer interviews at Google/Meta/FAANG

Key Difference from SWE Interviews: ML Engineer interviews test both:

1. **ML Fundamentals** - Algorithm implementations, math intuition, model selection
2. **ML System Design** - Production ML infrastructure (covered in separate guide)

1 Study Strategy: Depth-First on Deep Learning

1.1 Why Deep Learning First?

In 2025, **deep learning dominates** Staff/Principal ML interviews:

- **60% of questions** involve neural networks, transformers, or deep learning concepts
- **Foundation for everything**: Understanding backprop helps you reason about gradient boosting
- **Differentiation factor**: Classical ML is table stakes; deep learning expertise sets you apart
- **Transfer learning era**: Most production ML uses pretrained models (BERT, ResNet, GPT)

1.2 The Optimal Study Path

Phase 1: Deep Learning Foundations (Days 1-7)
Focus on fundamentals that underpin all architectures:

1. **Neural Network Mechanics** (Days 1-2)
   - Forward propagation, backpropagation (write from scratch)
   - Activation functions (ReLU, Sigmoid, Tanh, Softmax)
   - Loss functions (MSE, Cross-Entropy, Hinge)
   - Initialization (Xavier, He), optimization (SGD, Adam)
   - **Practice**: Implement 3-layer NN from scratch in NumPy
2. **Convolutional Networks** (Days 3-4)
   - Conv layers, pooling, receptive fields
   - Classic architectures (LeNet, VGG, ResNet)
   - Batch normalization, residual connections

- **Practice**: Build CNN for CIFAR-10, understand ResNet blocks

3. **Recurrent Networks** (Day 5)
   - RNN, LSTM, GRU architectures
   - Vanishing/exploding gradients
   - Bidirectional RNNs, encoder-decoder
   - **Practice**: Implement LSTM for sentiment analysis
4. **Transformers & Attention** (Days 6-7)
   - Self-attention mechanism (critical!)
   - Multi-head attention, positional encoding
   - BERT vs GPT architectures
   - Vision Transformers (ViT)
   - **Practice**: Implement scaled dot-product attention from scratch

Phase 2: Advanced Deep Learning (Days 8-10)

1. **Generative Models** (Day 8)
   - Autoencoders (AE, VAE, DAE)
   - GANs (generator, discriminator, training dynamics)
   - Diffusion models (conceptual understanding)
   - **Practice**: Implement VAE for MNIST
2. **Optimization & Regularization** (Day 9)
   - Optimizers (SGD, Momentum, Adam, AdamW)
   - Regularization (L1/L2, dropout, batch norm, early stopping)
   - Learning rate schedules (step decay, cosine annealing)
   - Gradient clipping, weight initialization
   - **Practice**: Compare Adam vs SGD+Momentum empirically
3. **Transfer Learning & Fine-tuning** (Day 10)
   - Pretrained models (BERT, ResNet, GPT)
   - Feature extraction vs fine-tuning
   - Domain adaptation techniques
   - **Practice**: Fine-tune BERT on custom classification task

Phase 3: Classical ML (Days 11-14)
Now that you understand deep learning, classical ML is easier:

1. **Tree-Based Methods** (Days 11-12)
   - Decision trees (Gini, entropy, information gain)
   - Random forests (bagging, feature sampling)
   - Gradient boosting (XGBoost, LightGBM, CatBoost)
   - **Key insight**: Boosting = gradient descent in function space
   - **Practice**: Kaggle tabular dataset with XGBoost
2. **Core Classical Algorithms** (Day 13)
   - Linear/Logistic regression (closed form, gradient descent)
   - SVMs (margin, kernel trick)

- K-Means, PCA (dimensionality reduction)
- Naive Bayes (for text classification)
- **Practice**: Implement logistic regression with L1/L2 regularization

3. **Evaluation & Metrics** (Day 14)
   - Classification metrics (precision, recall, F1, ROC-AUC)
   - Regression metrics (MSE, MAE, R^2)
   - Cross-validation strategies
   - Bias-variance tradeoff
   - **Practice**: Debug a model with high bias vs high variance

1.3 Spaced Repetition Schedule

For critical deep learning concepts (Priority 1), use this schedule:

- **Day 1**: Deep study + implement from scratch (2-3 hours)
- **Day 2**: Review notes + re-implement key parts (1 hour)
- **Day 4**: Quick review + whiteboard explanation (30 min)
- **Day 7**: Explain to someone else or write blog post (30 min)
- **Day 14**: Final review + practice problem (30 min)

This schedule is neurologically optimal for long-term retention.

2 Common Weak Spots for ML Engineers

Most candidates struggle with these (prioritize if time-limited):

2.1 Deep Learning Weak Spots

1. **Backpropagation Math**
   - Can't derive gradients for custom layers
   - Confused about chain rule application
   - **Fix**: Implement backprop for sigmoid and ReLU from scratch
2. **Attention Mechanism**
   - Don't understand scaled dot-product attention formula
   - Confused about Q, K, V matrices
   - **Fix**: Implement attention step-by-step, visualize attention weights
3. **Normalization Techniques**
   - Confused about Batch Norm vs Layer Norm vs Instance Norm
   - Don't know when to use each
   - **Fix**: Understand why Layer Norm works better for Transformers
4. **Vanishing/Exploding Gradients**
   - Can't explain why RNNs suffer from this
   - Don't know solutions (LSTM gates, gradient clipping, residual connections)
   - **Fix**: Derive gradient flow through 10-layer tanh network
5. **GAN Training Dynamics**
   - Don't understand Nash equilibrium concept
   - Confused about mode collapse
   - **Fix**: Implement simple GAN, observe training instability

2.2 Classical ML Weak Spots

1. **Bias-Variance Tradeoff**
   - Can't diagnose whether model has high bias or high variance
   - Don't know how to fix each problem
   - **Fix**: Practice identifying from learning curves
2. **Regularization Intuition**
   - Don't understand L1 vs L2 differences
   - Can't explain why L1 induces sparsity
   - **Fix**: Visualize L1/L2 penalty contours geometrically
3. **Kernel Trick in SVM**
   - Can't explain how it makes data linearly separable
   - Don't know when to use RBF vs polynomial kernel
   - **Fix**: Work through kernel SVM example on XOR problem
4. **Tree Pruning**
   - Don't know how to prevent overfitting in trees
   - Confused about pre-pruning vs post-pruning
   - **Fix**: Implement decision tree with max_depth and min_samples_split

3 Staff/Principal Level Expectations

3.1 Beyond Implementation

At Staff/Principal level, you're expected to:

1. **Justify Architecture Choices**
   - "Why CNN over RNN for this problem?"
   - "When would you use LSTM vs Transformer?"
   - "Why XGBoost instead of neural network for tabular data?"
   - **Prepare**: Know trade-offs for every algorithm (see Quick Reference)
2. **Quantitative Analysis**
   - Calculate model parameters (e.g., BERT has 110M parameters - why?)
   - Estimate training time and memory requirements
   - Compare computational complexity (Transformer is $O(n^2)$ in sequence length)
   - **Practice**: Calculate FLOPs for ResNet-50 forward pass
3. **Debug ML Systems**
   - "Model accuracy dropped from 95% to 70% - what's wrong?"
   - "Training loss decreases but val loss increases - what to do?"
   - "GAN not converging - how to fix?"
   - **Prepare**: Common failure modes checklist (next section)
4. **Understand Recent Advances**
   - Vision Transformers (ViT) vs CNNs
   - GPT vs BERT (decoder-only vs encoder-only)
   - Diffusion models vs GANs for generation
   - **Prepare**: Read 5-10 key papers (see Reading List)

3.2 Common Failure Modes & Fixes

Training Issues:

- Loss is NaN → Gradient exploding (use gradient clipping, lower LR)
- Loss not decreasing → Learning rate too small, bad initialization, or gradient vanishing
- Training loss drops but val loss increases → Overfitting (add dropout, regularization, early stopping)
- Training very slow → Bottleneck in data loading, use smaller batches, gradient accumulation

Model Performance Issues:

- High bias (underfitting) → Use larger model, more features, less regularization
- High variance (overfitting) → More data, regularization, simpler model, data augmentation
- Poor generalization → Data leakage, distribution shift, need domain adaptation
- Class imbalance → Weighted loss, oversampling minority class, focal loss

Deep Learning Specific:

- RNN not learning long-term dependencies → Use LSTM/GRU or Transformer
- CNN not working on small dataset → Use transfer learning (pretrained ResNet)
- Transformer running out of memory → Reduce sequence length, use gradient checkpointing
- GAN mode collapse → Use different loss (WGAN), add noise, batch diversity

4 Progress Tracking Table

Use this table to track your study progress. Fill in dates and check boxes as you complete each phase.

Priority Legend:

- **Priority 1 (P1)**: Critical - Master with spaced repetition
- **Priority 2 (P2)**: Important - Solid understanding needed
- **Priority 3 (P3)**: Good to know - Quick review sufficient

Note: On B&W printers, priorities show as: P1 (darkest gray), P2 (medium gray), P3 (lightest gray)

Algorithm/Concept	Pri	D1	D2	D4	D7	D14	Practice Notes
DEEP LEARNING FUNDAMENTALS							
Neural Network (MLP)	P1	☐	☐	☐	☐	☐	
Backpropagation	P1	☐	☐	☐	☐	☐	
Activation Functions	P1	☐	☐	☐	☐	☐	
Loss Functions	P1	☐	☐	☐	☐	☐	
CONVOLUTIONAL NEURAL NETWORKS							
CNN Basics	P1	☐	☐	☐	☐	☐	
Conv/Pooling Layers	P1	☐	☐	☐	☐	☐	
ResNet (Skip Connections)	P1	☐	☐	☐	☐	☐	
Batch Normalization	P2	☐	☐	☐	☐	☐	
RECURRENT NEURAL NETWORKS							
RNN Basics	P2	☐	☐	☐	☐	☐	
LSTM Architecture	P1	☐	☐	☐	☐	☐	
GRU Architecture	P2	☐	☐	☐	☐	☐	
Bidirectional RNN	P2	☐	☐	☐	☐	☐	
Seq2Seq + Attention	P2	☐	☐	☐	☐	☐	
TRANSFORMERS & ATTENTION							
Self-Attention Mechanism	P1	☐	☐	☐	☐	☐	
Multi-Head Attention	P1	☐	☐	☐	☐	☐	
Positional Encoding	P1	☐	☐	☐	☐	☐	
Transformer Encoder	P1	☐	☐	☐	☐	☐	
BERT Architecture	P2	☐	☐	☐	☐	☐	
GPT Architecture	P2	☐	☐	☐	☐	☐	
Vision Transformer (ViT)	P2	☐	☐	☐	☐	☐	
GENERATIVE MODELS							
Autoencoder	P2	☐	☐	☐	☐	☐	
VAE (Variational AE)	P1	☐	☐	☐	☐	☐	
GAN (Generator/Discriminator)	P1	☐	☐	☐	☐	☐	
Conditional GAN	P2	☐	☐	☐	☐	☐	
Diffusion Models (Conceptual)	P3	☐	☐	☐	☐	☐	
OPTIMIZATION & TRAINING							
SGD + Momentum	P1	☐	☐	☐	☐	☐	
Adam Optimizer	P1	☐	☐	☐	☐	☐	
Learning Rate Schedules	P2	☐	☐	☐	☐	☐	
Gradient Clipping	P2	☐	☐	☐	☐	☐	
REGULARIZATION							
L1/L2 Regularization	P1	☐	☐	☐	☐	☐	
Dropout	P1	☐	☐	☐	☐	☐	
Batch/Layer Normalization	P1	☐	☐	☐	☐	☐	

Algorithm/Concept	Pri	D1	D2	D4	D7	D14	Practice Notes
Early Stopping	P2	☐	☐	☐	☐	☐	
Data Augmentation	P2	☐	☐	☐	☐	☐	
CLASSICAL MACHINE LEARNING							
Linear Regression	P2	☐	☐	☐	☐	☐	
Logistic Regression	P1	☐	☐	☐	☐	☐	
Decision Trees	P1	☐	☐	☐	☐	☐	
Random Forest	P2	☐	☐	☐	☐	☐	
XGBoost/LightGBM	P1	☐	☐	☐	☐	☐	
SVM (Support Vector Machine)	P2	☐	☐	☐	☐	☐	
K-Nearest Neighbors	P3	☐	☐	☐	☐	☐	
Naive Bayes	P2	☐	☐	☐	☐	☐	
K-Means Clustering	P2	☐	☐	☐	☐	☐	
PCA (Dimensionality Reduction)	P2	☐	☐	☐	☐	☐	
EVALUATION & METRICS							
Precision/Recall/F1	P1	☐	☐	☐	☐	☐	
ROC-AUC	P1	☐	☐	☐	☐	☐	
Confusion Matrix	P2	☐	☐	☐	☐	☐	
MSE/MAE/R^2 (Regression)	P2	☐	☐	☐	☐	☐	
Cross-Validation	P2	☐	☐	☐	☐	☐	
Bias-Variance Tradeoff	P1	☐	☐	☐	☐	☐	

5 Algorithm Selection Quick Reference

5.1 When to Use Each Algorithm

Deep Learning:

- **CNN**: Images, spatial data, computer vision tasks
- **RNN/LSTM**: Sequential data, time series, short text (¡ 512 tokens)
- **Transformer**: NLP, long sequences, any task where attention matters
- **VAE**: Generate similar samples, anomaly detection, compression
- **GAN**: Generate realistic images, data augmentation, style transfer
- **ResNet**: Image classification (transfer learning baseline)
- **BERT**: Text classification, NER, question answering (encoder tasks)
- **GPT**: Text generation, completion, few-shot learning (decoder tasks)

Classical ML:

- **XGBoost/LightGBM**: Tabular data, structured data, Kaggle competitions
- **Random Forest**: Baseline for tabular data, feature importance, robust to outliers
- **Logistic Regression**: Baseline classification, need interpretability
- **Linear Regression**: Continuous prediction, feature relationships
- **SVM**: Small/medium datasets, high-dimensional data, clear margin
- **Naive Bayes**: Text classification, spam detection, fast training
- **K-Means**: Customer segmentation, data exploration, clustering
- **PCA**: Dimensionality reduction, visualization, noise reduction

5.2 Dataset Size Guidelines

- **¡ 1K samples**: Classical ML (Random Forest, SVM) or transfer learning
- **1K - 10K**: Classical ML or small neural networks with regularization
- **10K - 100K**: Neural networks or gradient boosting
- **100K - 1M**: Deep learning with data augmentation
- **¿ 1M**: Deep learning, large models, distributed training

5.3 Trade-off Cheatsheet

Algorithm	Pros	Cons	When to Use
CNN	Great for images, translation invariant	Needs lots of data	Computer vision
LSTM	Handles sequences, memory	Slow, vanishing gradient	Short sequences
Transformer	Parallel, long-range deps	$O(n^2)$ memory	NLP, long sequences
XGBoost	Fast, tabular SOTA	No spatial/sequential	Tabular data
Random Forest	Robust, no tuning needed	Less accurate than XGBoost	Quick baseline
SVM	Works in high-dim	Slow on large data	Small datasets

6 Essential Math Review

6.1 Calculus (for Backpropagation)

Chain Rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w}$

Common Derivatives:

- $\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$ (Sigmoid)
- $\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$
- $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$
- $\frac{d}{dx}(Ax + b) = A$ (Linear layer)

6.2 Linear Algebra (for Neural Networks)

Matrix Multiplication:

- $(m \times n) \cdot (n \times p) = (m \times p)$
- Not commutative: $AB \neq BA$
- Associative: $(AB)C = A(BC)$

Dot Product Attention:

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

6.3 Probability (for VAE, Naive Bayes)

Bayes' Theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

KL Divergence (VAE loss):

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

7 Recommended Reading List

7.1 Must-Read Papers (Priority Order)

1. **Attention is All You Need** (Vaswani et al., 2017) - Transformer architecture
2. **BERT: Pre-training of Deep Bidirectional Transformers** (Devlin et al., 2019)
3. **Deep Residual Learning for Image Recognition** (He et al., 2015) - ResNet
4. **Adam: A Method for Stochastic Optimization** (Kingma & Ba, 2014)
5. **Batch Normalization** (Ioffe & Szegedy, 2015)
6. **Auto-Encoding Variational Bayes** (Kingma & Welling, 2013) - VAE
7. **Generative Adversarial Networks** (Goodfellow et al., 2014) - GAN
8. **An Image is Worth 16x16 Words** (Dosovitskiy et al., 2020) - Vision Transformer
9. **XGBoost: A Scalable Tree Boosting System** (Chen & Guestrin, 2016)
10. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting** (Srivastava et al., 2014)

7.2 Online Resources

- **Stanford CS231n** (CNNs for Visual Recognition) - Free lecture videos
- **Stanford CS224n** (NLP with Deep Learning) - Covers Transformers well
- **Fast.ai Courses** - Practical deep learning
- **The Illustrated Transformer** (Jay Alammar's blog) - Best visual explanation
- **Distill.pub** - Interactive ML explanations

8 Mock Interview Practice

8.1 Sample ML Algorithm Questions

Implementation Questions (60 min):

1. Implement backpropagation for a 3-layer neural network from scratch
2. Implement multi-head attention mechanism in NumPy
3. Build a decision tree classifier with Gini impurity
4. Implement Adam optimizer from scratch
5. Code a VAE for MNIST digit generation

Conceptual Questions (30-45 min):

1. Explain how LSTM solves the vanishing gradient problem
2. Why does Batch Norm help training? What are alternatives?
3. Compare BERT and GPT architectures - when to use each?
4. Explain the reparameterization trick in VAE
5. How does gradient boosting differ from random forest?
6. Why is Transformer $O(n^2)$ in sequence length? How to reduce?

Debugging Questions (30 min):

1. Training loss stuck at 0.69 for binary classification - what's wrong?
2. Model works on training set but fails on validation - diagnose and fix
3. GAN generator producing garbage - how to debug?
4. RNN not learning dependencies beyond 10 timesteps - why?

8.2 Study Group Tips

If practicing with others:

- **Whiteboard sessions**: Explain algorithms without looking at notes
- **Code reviews**: Review each other's implementations
- **Paper reading**: Discuss one paper per week from reading list
- **Mock interviews**: Take turns being interviewer/candidate (45 min each)

9 Staff/Principal Level: Beyond Prediction

At Staff/Principal level, interviewers expect you to go beyond "what model gives best accuracy" to "what is the causal effect of our intervention" and "how do we learn and adapt online."

9.1 Causal Inference & Uplift Modeling

The Critical Distinction: Correlation vs Causation

Prediction (Correlation):

- **Question**: Who will click on this ad?
- **Math**: $P(Y = 1|X)$ where X is user features
- **Example**: High-income users are more likely to buy luxury products
- **Problem**: They might buy anyway, even without seeing the ad!

Causal Inference (Causation):

- **Question**: Who will click *because* they saw this ad?
- **Math**: $P(Y = 1|do(X = 1)) - P(Y = 1|do(X = 0))$
- **Example**: Which users are *persuadable* by the ad?
- **Goal**: Maximize incremental impact, not just raw conversions

Why This Matters:

Scenario: Ad targeting for e-commerce

- **Prediction model says**: Show ads to high-income users (they have 10% conversion rate)
- **Problem**: High-income users have 9% baseline conversion (without ads)
- **Incremental lift**: Only 1% due to ads
- **Causal model says**: Show ads to mid-income users (5% with ads, 1% without = 4% lift!)
- **Result**: 4× better ROI by targeting persuadable users

Uplift Modeling Approaches:

1. Two-Model Approach (T-Learner)

Train two separate models:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier

class UpliftTwoModel:
    """Two-model approach for uplift modeling"""
    def __init__(self):
        self.model_treatment = RandomForestClassifier(n_estimators=100)
        self.model_control = RandomForestClassifier(n_estimators=100)

    def fit(self, X, y, treatment):
        """
        X: Features (n_samples, n_features)
        y: Outcome (1 = conversion, 0 = no conversion)
        treatment: Treatment indicator (1 = saw ad, 0 = no ad)
        """
        # Split data by treatment
        X_treatment = X[treatment == 1]
        y_treatment = y[treatment == 1]

        X_control = X[treatment == 0]
```

```
        y_control = y[treatment == 0]

        # Train separate models
        self.model_treatment.fit(X_treatment, y_treatment)
        self.model_control.fit(X_control, y_control)

    def predict_uplift(self, X):
        """
        Predict uplift: E[Y | T=1, X] - E[Y | T=0, X]
        """
        # Predict conversion probability in each scenario
        p_treatment = self.model_treatment.predict_proba(X)[:, 1]
        p_control = self.model_control.predict_proba(X)[:, 1]

        # Uplift = difference
        uplift = p_treatment - p_control

        return uplift

# Usage
X_train = features  # User features
y_train = conversions  # Did they convert?
treatment = ad_shown  # Did they see the ad?

model = UpliftTwoModel()
model.fit(X_train, y_train, treatment)

# Predict uplift for new users
X_new = new_user_features
uplift_scores = model.predict_uplift(X_new)

# Target users with highest uplift
high_uplift_users = X_new[uplift_scores > 0.05]  # > 5% incremental lift
```

2. Single-Model Approach (S-Learner)
Train one model with treatment as a feature:

```
class UpliftSingleModel:
    """Single-model approach with treatment as feature"""
    def __init__(self):
        self.model = RandomForestClassifier(n_estimators=100)

    def fit(self, X, y, treatment):
        # Add treatment as a feature
        X_with_treatment = np.column_stack([X, treatment])
        self.model.fit(X_with_treatment, y)

    def predict_uplift(self, X):
        # Predict with treatment = 1
        X_treatment = np.column_stack([X, np.ones(len(X))])
        p_treatment = self.model.predict_proba(X_treatment)[:, 1]

        # Predict with treatment = 0
        X_control = np.column_stack([X, np.zeros(len(X))])
        p_control = self.model.predict_proba(X_control)[:, 1]

        # Uplift
        uplift = p_treatment - p_control
        return uplift
```

3. Class Transformation Approach (Pessimistic Uplift)
Reframe as 4-class problem:

- **Persuadables**: $Y = 0$ in control, $Y = 1$ in treatment (want to target these!)

- **Sure Things**: $Y = 1$ in both (waste of budget)
- **Lost Causes**: $Y = 0$ in both (waste of budget)
- **Sleeping Dogs**: $Y = 1$ in control, $Y = 0$ in treatment (avoid these!)

Evaluation Metrics for Uplift:
Uplift Curve:

```
def uplift_curve(uplift_scores, y, treatment, num_bins=10):
    """
    Plot cumulative uplift vs % population targeted
    """
    import matplotlib.pyplot as plt

    # Sort by uplift score (descending)
    sorted_indices = np.argsort(uplift_scores)[::-1]
    y_sorted = y[sorted_indices]
    treatment_sorted = treatment[sorted_indices]

    # Calculate cumulative uplift
    cumulative_uplift = []
    percentiles = np.linspace(0, 1, num_bins)

    for p in percentiles:
        n = int(p * len(y_sorted))
        if n == 0:
            cumulative_uplift.append(0)
            continue

        # Uplift = (conversion_rate_treatment - conversion_rate_control)
        treated_mask = treatment_sorted[:n] == 1
        control_mask = treatment_sorted[:n] == 0

        if treated_mask.sum() == 0 or control_mask.sum() == 0:
            cumulative_uplift.append(0)
            continue

        cr_treatment = y_sorted[:n][treated_mask].mean()
        cr_control = y_sorted[:n][control_mask].mean()

        uplift = cr_treatment - cr_control
        cumulative_uplift.append(uplift)

    # Plot
    plt.plot(percentiles * 100, cumulative_uplift)
    plt.xlabel('% Population Targeted')
    plt.ylabel('Cumulative Uplift')
    plt.title('Uplift Curve')
    plt.grid(True)
    plt.show()

    return cumulative_uplift
```

Qini Coefficient:

- Analogous to AUC for uplift
- Area between uplift curve and random targeting
- Higher = better uplift model

Interview Discussion Points:

- **When to use uplift**: Ad targeting, promotional campaigns, content recommendations

- **Trade-offs**: Requires randomized experiment data (treatment + control groups)
- **A/B test design**: Need to log who was shown treatment vs control
- **Business impact**: "We increased ROI by 40% by targeting persuadable users instead of high-propensity users"

9.2 Advanced Exploration: Bandits & Reinforcement Learning

Problem with Static A/B Testing:

- **Slow**: Wait 2-4 weeks for statistical significance
- **Wasteful**: Send 50% traffic to inferior variant
- **Rigid**: Can't adapt to changing user preferences

Solution: Online Learning with Multi-Armed Bandits
The Explore-Exploit Dilemma:

- **Exploit**: Show the item you think is best (maximize short-term reward)
- **Explore**: Try new items to learn (maximize long-term reward)
- **Goal**: Minimize regret (cumulative difference from optimal choice)

1. Epsilon-Greedy
Algorithm:

```
class EpsilonGreedy:
    """Simplest bandit algorithm"""
    def __init__(self, n_arms, epsilon=0.1):
        self.n_arms = n_arms
        self.epsilon = epsilon

        # Track statistics
        self.counts = np.zeros(n_arms)  # Times each arm was pulled
        self.values = np.zeros(n_arms)  # Average reward for each arm

    def select_arm(self):
        """Select arm to pull"""
        if np.random.random() < self.epsilon:
            # Explore: random arm
            return np.random.randint(self.n_arms)
        else:
            # Exploit: best arm so far
            return np.argmax(self.values)

    def update(self, arm, reward):
        """Update statistics after observing reward"""
        self.counts[arm] += 1

        # Incremental average update
        n = self.counts[arm]
        old_value = self.values[arm]
        self.values[arm] = old_value + (reward - old_value) / n

# Example: Recommend one of 5 articles
bandit = EpsilonGreedy(n_arms=5, epsilon=0.1)

for user in users:
    # Select article
    article = bandit.select_arm()
```

```
    # Show article, observe click (1) or no click (0)
    reward = show_article_and_get_feedback(user, article)

    # Update model
    bandit.update(article, reward)
```

Pros: Simple, easy to implement
Cons: Fixed exploration rate (doesn't reduce over time)

2. Upper Confidence Bound (UCB)

Idea: Be optimistic about uncertain arms

Formula:

$$\text{UCB}_i = \bar{x}_i + \sqrt{\frac{2 \log t}{n_i}}$$

where:

- $\bar{x}_i$ = average reward for arm i
- t = total number of pulls
- n_i = number of pulls for arm i

```
class UCB:
    """Upper Confidence Bound algorithm"""
    def __init__(self, n_arms):
        self.n_arms = n_arms
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)
        self.total_pulls = 0

    def select_arm(self):
        # Initialize: pull each arm once
        for arm in range(self.n_arms):
            if self.counts[arm] == 0:
                return arm

        # UCB score for each arm
        ucb_scores = []
        for arm in range(self.n_arms):
            avg_reward = self.values[arm]
            exploration_bonus = np.sqrt(2 * np.log(self.total_pulls) / self.counts[arm])
            ucb = avg_reward + exploration_bonus
            ucb_scores.append(ucb)

        # Select arm with highest UCB
        return np.argmax(ucb_scores)

    def update(self, arm, reward):
        self.counts[arm] += 1
        self.total_pulls += 1

        n = self.counts[arm]
        old_value = self.values[arm]
        self.values[arm] = old_value + (reward - old_value) / n
```

Pros: Theoretical regret bounds, adaptive exploration
Cons: Assumes rewards are bounded in [0, 1]

3. Thompson Sampling (Bayesian Approach)

Idea: Sample from posterior distribution of arm quality

```
class ThompsonSampling:
    """Bayesian bandit with Beta-Bernoulli"""
    def __init__(self, n_arms):
        self.n_arms = n_arms
```

```
        # Beta distribution parameters (prior: Beta(1, 1) = Uniform)
        self.alpha = np.ones(n_arms)  # Successes
        self.beta = np.ones(n_arms)   # Failures

    def select_arm(self):
        # Sample from Beta distribution for each arm
        samples = [
            np.random.beta(self.alpha[arm], self.beta[arm])
            for arm in range(self.n_arms)
        ]

        # Select arm with highest sample
        return np.argmax(samples)

    def update(self, arm, reward):
        """Update Beta distribution"""
        if reward > 0:
            self.alpha[arm] += 1
        else:
            self.beta[arm] += 1

# Example: Netflix thumbnail selection
bandit = ThompsonSampling(n_arms=3)  # 3 thumbnail variants

for user in users:
    # Sample which thumbnail to show
    thumbnail = bandit.select_arm()

    # Show thumbnail, observe click
    clicked = show_thumbnail_and_get_feedback(user, thumbnail)

    # Update posterior
    bandit.update(thumbnail, reward=clicked)
```

Pros: Best empirical performance, naturally balances exploration/exploitation
Cons: Requires choosing prior distribution

4. Contextual Bandits

Problem: User preferences vary (one-size-fits-all doesn't work)

Solution: Use user features to personalize

```
class LinUCB:
    """Contextual bandit with linear model"""
    def __init__(self, n_arms, n_features, alpha=1.0):
        self.n_arms = n_arms
        self.n_features = n_features
        self.alpha = alpha  # Exploration parameter

        # For each arm, maintain linear regression
        self.A = [np.identity(n_features) for _ in range(n_arms)]  # X^T X
        self.b = [np.zeros(n_features) for _ in range(n_arms)]     # X^T y

    def select_arm(self, context):
        """
        context: User features (n_features,)
        """
        ucb_scores = []

        for arm in range(self.n_arms):
            # Estimate: theta = A^-1 b
            A_inv = np.linalg.inv(self.A[arm])
            theta = A_inv.dot(self.b[arm])
```

```
            # Predicted reward
            predicted_reward = theta.dot(context)

            # Confidence interval
            uncertainty = np.sqrt(context.dot(A_inv).dot(context))

            # UCB
            ucb = predicted_reward + self.alpha * uncertainty
            ucb_scores.append(ucb)

        return np.argmax(ucb_scores)

    def update(self, arm, context, reward):
        """Update model for selected arm"""
        self.A[arm] += np.outer(context, context)  # X^T X
        self.b[arm] += reward * context             # X^T y

# Usage: Personalized article recommendation
linucb = LinUCB(n_arms=10, n_features=50, alpha=0.5)

for user in users:
    # Get user features
    context = get_user_features(user)  # (50,)

    # Select article
    article = linucb.select_arm(context)

    # Observe reward
    reward = show_article(user, article)

    # Update
    linucb.update(article, context, reward)
```

5. Introduction to Reinforcement Learning for Recommendations
Framing Recommendations as RL:
MDP (Markov Decision Process):

- **State** s_t: User's history (last 10 items viewed, embeddings)
- **Action** a_t: Recommend item i
- **Reward** r_t: Immediate engagement (click, watch time)
- **Transition** s_{t+1}: User's updated state after seeing recommendation

Goal: Learn policy $\pi(a|s)$ that maximizes cumulative reward

$$J = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

where $\gamma \in [0, 1]$ is discount factor
Why RL vs Supervised Learning?

- **Sequential decisions**: Today's recommendation affects tomorrow's user state
- **Long-term reward**: Maximize lifetime value, not just immediate click
- **Exploration**: Discover new user preferences

Simple RL Algorithm: Q-Learning

```
class QLearning:
    """Tabular Q-learning for recommendations"""
    def __init__(self, n_states, n_actions, lr=0.1, gamma=0.9, epsilon=0.1):
```

```
        self.n_states = n_states
        self.n_actions = n_actions
        self.lr = lr          # Learning rate
        self.gamma = gamma    # Discount factor
        self.epsilon = epsilon

        # Q-table: Q[s, a] = expected return from state s, taking action a
        self.Q = np.zeros((n_states, n_actions))

    def select_action(self, state):
        """Epsilon-greedy action selection"""
        if np.random.random() < self.epsilon:
            return np.random.randint(self.n_actions)  # Explore
        else:
            return np.argmax(self.Q[state, :])  # Exploit

    def update(self, state, action, reward, next_state):
        """Q-learning update"""
        # Current Q-value
        current_q = self.Q[state, action]

        # Target: r + gamma * max_a' Q(s', a')
        max_next_q = np.max(self.Q[next_state, :])
        target = reward + self.gamma * max_next_q

        # Update Q-value
        self.Q[state, action] = current_q + self.lr * (target - current_q)

# Example (simplified)
q_learning = QLearning(n_states=100, n_actions=50)

state = get_user_state(user)  # Discretized user state
action = q_learning.select_action(state)  # Select item
reward = show_item_and_get_feedback(user, action)
next_state = get_updated_state(user)

q_learning.update(state, action, reward, next_state)
```

Deep RL for Recommendations:

- **DQN**: Deep Q-Network (replace Q-table with neural network)
- **Policy Gradient**: Directly learn policy $\pi_\theta(a|s)$
- **Actor-Critic**: Combine value function and policy

Used by: YouTube (REINFORCE), Alibaba (deep Q-learning)
Interview Discussion Points:
Bandits:

- **When to use**: Real-time personalization, A/B test optimization, content selection
- **Trade-offs**: UCB (theoretical guarantees) vs Thompson Sampling (best empirical)
- **Contextual**: When user features matter (personalization)
- **Example**: "At Netflix, we use Thompson Sampling for thumbnail selection, reducing time-to-first-watch by 15%"

RL:

- **When to use**: Long-term user engagement, sequential recommendations
- **Challenges**: Delayed rewards, exploration cost, off-policy learning
- **Example**: "YouTube uses RL to optimize for long-term watch time, not just immediate clicks"

9.3 Connecting the Dots: When to Use Uplift vs. Bandits vs. RL

The Principal-Level Skill: Knowing the tools is good. Knowing *which tool to use for which business problem* is what separates Staff from Principal.

This decision guide helps you quickly map business problems to the right technique in interviews.

Quick Decision Tree:

```
Business Problem
    |
    +-- Is the intervention costly/limited? (e.g., sending coupons)
    |   +-- YES -> Uplift Modeling
    |       * Identify WHO to treat (persuadables)
    |       * Optimize ROI of expensive interventions
    |
    +-- Fixed set of choices, need to pick best option in real-time?
    |   +-- YES -> Contextual Bandits
    |       * E.g., Which of 5 headlines to show?
    |       * Minimize regret, maximize immediate reward
    |
    +-- Sequential decisions where today's action affects future state?
        +-- YES -> Reinforcement Learning
            * E.g., Recommend series of videos to maximize session time
            * Optimize long-term cumulative reward
```

Detailed Comparison Table:

Characteristic	Uplift Modeling	Contextual Bandits	Reinforcement Learning
Goal	Identify users who respond to treatment	Select best action to maximize immediate reward	Maximize long-term cumulative reward
Data	Historical A/B test data (treatment/control)	Online interaction data (click/no-click)	Sequential interaction data
Decision timing	Offline (batch targeting)	Online (real-time per request)	Online (sequential decisions)
Action space	Binary (treat vs. don't treat)	Fixed set (e.g., 5-10 options)	Large/continuous (many possible actions)
Feedback	Post-hoc analysis of RCT	Immediate (click/no-click)	Delayed (session reward)
Complexity	Medium	Low-Medium	High
When to use	Costly interventions, segment users	Real-time personalization, fixed choices	Sequential decisions, long-term optimization

Use Case 1: Marketing Campaign

Problem: "We have 10M users. Sending discount coupons costs $5 each. How do we maximize ROI?"

Wrong approach: Send to everyone → Waste money on users who'd convert anyway

Right approach: Uplift Modeling

- **Why**: Intervention is costly ($5), need to identify persuadables
- **Method**: Run A/B test on 100K users, train T-Learner
- **Outcome**: Target top 20% by uplift → 4x ROI vs random targeting
- **Key metric**: Incremental conversions per dollar spent

Interview answer: "I'd use uplift modeling to identify the persuadable segment. Train on historical coupon experiment data, predict uplift score for each user, target top 20% by predicted incremental conversion probability. This maximizes ROI by avoiding 'sure things' who'd buy anyway and 'lost causes' who won't buy even with discount."

Use Case 2: Homepage Personalization

Problem: "Which of 5 hero images should we show each user on homepage?"
Wrong approach: Static A/B test takes 4 weeks to find best image
Right approach: Contextual Bandit (Thompson Sampling)

- **Why**: Fixed choices (5 images), real-time decision, want to minimize regret
- **Method**: Thompson Sampling with user features (location, device, time)
- **Outcome**: Converge to best image in 3 days instead of 4 weeks, adapt to user context
- **Key metric**: Cumulative regret (clicks lost vs. optimal policy)

Interview answer: "I'd use LinUCB or Thompson Sampling bandit. Initialize with uniform exploration, then adaptively allocate traffic to winning images. Include user context (new vs. returning, mobile vs. desktop) to personalize. This finds the optimal policy 10x faster than A/B testing and minimizes opportunity cost."

Use Case 3: Video Recommendation Sequence
Problem: "Recommend next video to maximize user's entire session watch time"
Wrong approach: Maximize immediate click probability → Clickbait videos, short sessions
Right approach: Reinforcement Learning (DQN or Policy Gradient)

- **Why**: Sequential decisions, long-term reward (session time), state transitions matter
- **Method**: Model as MDP, state = user history, action = next video, reward = watch time
- **Outcome**: Optimize for 30-min session, not 3-min video
- **Key metric**: Total session watch time, retention rate

Interview answer: "I'd frame this as an RL problem with state = [last 10 videos watched, time on platform], action = next video to recommend, reward = total session watch time. Use off-policy learning (importance sampling) to train from logged data, then deploy with epsilon-greedy for online exploration. This optimizes for long-term engagement, not just immediate clicks."

Hybrid Approaches (Advanced):
In practice, you often **combine** these techniques:
Example: E-commerce Personalization

1. **Uplift model**: Decide WHETHER to show discount banner (costly intervention)
2. **Bandit**: Select WHICH product to feature in banner (5 options)
3. **RL**: Optimize product browsing sequence for long-term purchase intent

Implementation:

```
# Step 1: Uplift - Should we show discount?
uplift_score = uplift_model.predict_uplift(user_features)
if uplift_score > threshold:
    show_discount = True
else:
    show_discount = False

# Step 2: Bandit - Which product to feature?
if show_discount:
    # Use contextual bandit to select product
    product = bandit.select_arm(user_context)

# Step 3: RL - Optimize browsing sequence
# State: user's current session history
# Action: which category to recommend next
# Reward: eventual purchase value
next_category = rl_agent.select_action(session_state)
```

Interview Gold - The Framework:
When asked "Design a personalization system," use this framework:

1. **Identify the intervention**:
    - Is it costly? → Uplift might be needed
    - Is it free? → Bandits or RL can explore aggressively
2. **Understand the action space**:
    - Fixed small set (¡ 20 options)? → Bandits work well
    - Large/continuous? → RL or approximate methods
3. **Consider time horizon**:
    - Immediate reward? → Bandits optimize this
    - Long-term reward? → RL required
4. **Check data availability**:
    - Have historical RCT data? → Uplift modeling ready to go
    - Need to learn online? → Bandits/RL
5. **Evaluate complexity vs. value**:
    - Simple problem → Start with bandits
    - Complex sequential problem + high value → Justify RL investment

Common Pitfalls to Avoid:

- **Using RL when bandits suffice**: "Recommend next video" → If reward is immediate click, bandits work fine. Only use RL if optimizing session-level metrics.
- **Using bandits for costly interventions**: "Which users to send $50 gift card?" → Use uplift, not bandits. Exploration is too expensive.
- **Using uplift for real-time decisions**: "Which ad to show?" → Uplift is for batch targeting, use bandits for real-time.
- **Ignoring offline evaluation**: Always validate on historical data before deploying RL/bandits online.

Summary - Quick Reference:

Technique	Best For	Interview Keywords
Uplift Modeling	Costly interventions, segment targeting	"Persuadables," "incremental," "ROI optimization," "T-Learner"
Contextual Bandits	Real-time choices, fixed action space	"Regret minimization," "Thompson Sampling," "UCB"
Reinforcement Learning	Sequential decisions, state transitions	"Long-term reward," "MDP," "Q-Learning," "policy gradient"

Final Interview Tip:

When you mention these techniques, immediately clarify *why* you chose them:

"I'd use uplift modeling here because we're dealing with a costly intervention—sending physical mailers at $2 each. The goal is to identify the persuadable segment, not just predict who will respond. Traditional classification would target 'sure things' who'd convert anyway, wasting budget."

This shows you understand the **business problem**, not just the technical solution.

10 Final Checklist Before Interviews

10.1 Day Before Interview

☐ Review all Priority 1 algorithms (15 min each)
☐ Practice whiteboarding one architecture from each category:

- Neural network (MLP with backprop)
- CNN (ResNet block)
- Transformer (attention mechanism)
- Classical ML (decision tree or XGBoost concept)

☐ Review common failure modes and debugging strategies
☐ Prepare 3-5 questions about the team's ML stack

10.2 Common Interview Mistakes to Avoid

- **Jumping to implementation too fast** - Always clarify the problem first
- **Not stating assumptions** - "Assuming we have labeled data..."
- **Ignoring trade-offs** - Every algorithm has pros/cons, discuss them
- **Not explaining your reasoning** - Think out loud during implementation
- **Forgetting to validate** - Always mention train/val/test split and metrics
- **Over-engineering** - Start simple, then optimize if asked

11 Conclusion

This 14-day intensive plan focuses on **depth-first mastery of deep learning** followed by classical ML fundamentals. The key to success:

1. **Implement from scratch** - Don't just read, code it yourself
2. **Spaced repetition** - Review on Days 1, 2, 4, 7, 14
3. **Understand trade-offs** - Every algorithm has strengths and weaknesses
4. **Practice debugging** - Most interviews test this more than implementation

Good luck with your Staff/Principal ML Engineer interviews!