Machine Learning Course Notes

Deep Learning Specialization - Course 2, Week 3

Andrew Ng - Complete Study Guide

---

1. Advice for Applying Machine Learning

Learning Objectives

- Understand the systematic approach to machine learning development
- Learn how to diagnose and fix common ML problems
- Develop intuition for when different techniques should be applied

Theoretical Concepts

Machine learning development is an iterative process that requires systematic debugging and improvement. Rather than randomly trying different approaches, successful ML practitioners follow a structured methodology to identify and solve problems.

Key Principles:

- **Systematic Debugging**: Use diagnostics to understand what's wrong before attempting fixes
- **Data-Driven Decisions**: Let error analysis and performance metrics guide your choices
- **Iterative Improvement**: Continuously refine models based on evidence

Practical Tips

- Always start with a simple baseline model
- Use diagnostics (bias/variance analysis) before adding complexity
- Focus on one improvement at a time to understand its impact
- Document your experiments and results

Quick Summary/Insights

- Machine learning success requires systematic debugging, not random experimentation
- Use diagnostics to identify problems before implementing solutions
- Follow an iterative process: build → measure → learn → improve

---

2. Evaluating a Model

Learning Objectives

- Understand how to properly evaluate ML model performance
- Learn the importance of train/test splits
- Master different evaluation metrics for regression and classification

Theoretical Concepts

Model evaluation is crucial for understanding how well your algorithm will perform on new, unseen data. The fundamental principle is to test your model on data it has never seen during training.

Data Splitting:

- **Training Set**: Used to learn model parameters
- **Test Set**: Used to evaluate final model performance (typically 20-30% of data)

Evaluation Metrics:

- **Regression**: Mean Squared Error, Mean Absolute Error
- **Classification**: Accuracy, Precision, Recall, F1-Score

Mathematical Formulas and Intuition

Mean Squared Error (Regression):

```
J_test = (1/2m_test) * Σ(h_θ(x^(i)) - y^(i))²
```

Misclassification Rate (Classification):

```
Test Error = (1/m_test) * Σ(h_θ(x^(i)) ≠ y^(i))
```

Where:

- m_test = number of test examples
- h_θ(x) = model prediction
- y = actual label

Code Implementation

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train model
model.fit(X_train, y_train)

# Evaluate
predictions = model.predict(X_test)
test_error = mean_squared_error(y_test, predictions)  # For regression
# accuracy = accuracy_score(y_test, predictions)      # For classification
```

Practical Tips

- Never use test data for model selection or hyperparameter tuning
- Ensure test set is representative of real-world data
- Consider stratified sampling for imbalanced datasets

Quick Summary/Insights

- Proper evaluation requires separate test data never seen during training
- Choose appropriate metrics based on your problem type (regression vs classification)
- Test error provides an estimate of how well your model will generalize

---

3. Model Selection and Training/Cross Validation/Test Sets

Learning Objectives

- Understand the three-way data split methodology
- Learn how to use cross-validation for model selection
- Master hyperparameter tuning without overfitting to test data

Theoretical Concepts

When selecting between different models or tuning hyperparameters, using only training and test sets can lead to overfitting to the test set. The solution is a three-way split:

Three-Way Data Split:

- **Training Set (60%)**: Learn model parameters
- **Cross-Validation/Dev Set (20%)**: Select hyperparameters and model architecture
- **Test Set (20%)**: Final unbiased performance estimate

Model Selection Process:

1. Train multiple models on training set
2. Evaluate each on cross-validation set
3. Select best performing model
4. Report final performance on test set

Mathematical Formulas and Intuition

Cross-Validation Error:

```
J_cv = (1/2m_cv) * Σ(h_θ(x_cv^(i)) - y_cv^(i))²
```

Model Selection Criterion:

```
θ* = argmin_θ J_cv(θ)
```

Code Implementation

```python
from sklearn.model_selection import train_test_split
from sklearn.polynomial_features import PolynomialFeatures

# Three-way split
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4)
X_cv, X_test, y_cv, y_test = train_test_split(X_temp, y_temp, test_size=0.5)

# Model selection
best_degree, best_error = None, float('inf')
for degree in range(1, 11):
    poly_features = PolynomialFeatures(degree)
    X_train_poly = poly_features.fit_transform(X_train)
    X_cv_poly = poly_features.transform(X_cv)

    model.fit(X_train_poly, y_train)
    cv_error = mean_squared_error(y_cv, model.predict(X_cv_poly))

    if cv_error < best_error:
        best_error, best_degree = cv_error, degree
```

Practical Tips

- Never make decisions based on test set performance
- Use cross-validation set for all hyperparameter tuning
- Keep test set truly held out until final evaluation

Quick Summary/Insights

- Three-way split prevents overfitting to test data during model selection
- Cross-validation set is used for all model selection decisions
- Test set provides unbiased final performance estimate

---

4. Bias and Variance

Learning Objectives

- Understand the fundamental bias-variance tradeoff
- Learn to identify high bias vs high variance problems
- Develop intuition for model complexity effects

Theoretical Concepts

Bias and variance are two fundamental sources of error in machine learning models:

Bias (Underfitting):

- Model is too simple to capture underlying patterns
- High error on both training and test data
- Model assumptions are too strong

Variance (Overfitting):

- Model is too complex and memorizes training data
- Low training error, high test error
- Model is too sensitive to training data variations

The Tradeoff:

- Simple models: High bias, low variance
- Complex models: Low bias, high variance
- Goal: Find optimal balance

Mathematical Formulas and Intuition

Expected Error Decomposition:

$$E[Error] = Bias^2 + Variance + Irreducible\ Error$$

Bias:

$$Bias = E[\hat{f}(x)] - f(x)$$

Variance:

$$Variance = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

Where $\hat{f}(x)$ is our model and $f(x)$ is the true function.

Practical Tips

- Start with simple models and gradually increase complexity

- Use learning curves to visualize bias-variance tradeoff
- Regularization helps control variance
- More data generally reduces variance

Quick Summary/Insights

- Bias-variance tradeoff is fundamental to ML model performance
- High bias = underfitting, high variance = overfitting
- Optimal model complexity balances both sources of error

---

5. Diagnosing Bias and Variance

Learning Objectives

- Learn to quantitatively diagnose bias and variance issues
- Understand how training and CV errors indicate different problems
- Master the decision framework for model improvement

Theoretical Concepts

Systematic diagnosis of bias and variance involves comparing training error (J_train) and cross-validation error (J_cv):

High Bias (Underfitting) Indicators:

- J_train is high
- J_cv ≈ J_train (both high)

High Variance (Overfitting) Indicators:

- J_train is low
- J_cv >> J_train (large gap)

Just Right:

- Both J_train and J_cv are low
- Small gap between them

Mathematical Formulas and Intuition

Diagnostic Criteria:

```
High Bias: J_train > threshold AND J_cv ≈ J_train
High Variance: J_train < threshold AND J_cv >> J_train
```

Gap Analysis:

```
Variance Gap = J_cv - J_train
```

Large gap indicates overfitting; small gap with high errors indicates underfitting.

Code Implementation

python

```python
def diagnose_bias_variance(train_errors, cv_errors, threshold=0.1):
    train_err = train_errors[-1]  # Final training error
    cv_err = cv_errors[-1]        # Final CV error
    gap = cv_err - train_err

    if train_err > threshold and gap < 0.05:
        return "High Bias (Underfitting)"
    elif train_err < threshold and gap > 0.1:
        return "High Variance (Overfitting)"
    else:
        return "Good Balance"
```

Practical Tips

- Compare errors to human-level performance when possible
- Consider the gap between training and CV error
- Use multiple metrics for comprehensive diagnosis

Quick Summary/Insights

- Compare J_train and J_cv to diagnose bias vs variance issues
- High bias: both errors high; High variance: large gap between errors
- Quantitative diagnosis guides improvement strategy

---

6. Regularization and Bias/Variance

Learning Objectives

- Understand how regularization affects bias-variance tradeoff
- Learn to tune regularization parameter λ systematically
- Master the relationship between λ and model complexity

Theoretical Concepts

Regularization adds a penalty term to the cost function to control model complexity:

Regularization Effects:

- **Small λ**: Less regularization → Lower bias, higher variance
- **Large λ**: More regularization → Higher bias, lower variance
- **λ = 0**: No regularization (may overfit)
- **λ too large**: Under-regularization (may underfit)

L2 Regularization (Ridge): Adds penalty proportional to sum of squared weights.

Mathematical Formulas and Intuition

Regularized Cost Function:

```
J(θ) = (1/2m)[Σ(h_θ(x^(i)) - y^(i))² + λΣθ_j²]
```

λ Selection Process:

```
For λ in [0, 0.01, 0.02, 0.04, 0.08, ..., 10]:
    Train model with λ
    Compute J_cv(λ)
Choose λ* = argmin_λ J_cv(λ)
```

Code Implementation

```python
from sklearn.linear_model import Ridge
import numpy as np

# λ selection
lambdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
cv_errors = []

for lam in lambdas:
    model = Ridge(alpha=lam)
    model.fit(X_train, y_train)
    cv_pred = model.predict(X_cv)
    cv_errors.append(mean_squared_error(y_cv, cv_pred))

best_lambda = lambdas[np.argmin(cv_errors)]
```

Practical Tips

- Try λ values over several orders of magnitude
- Use cross-validation to select optimal λ
- Plot training/CV error vs λ to visualize tradeoff

Quick Summary/Insights

- Regularization parameter λ controls bias-variance tradeoff
- Use systematic search over λ values with cross-validation
- Optimal λ minimizes cross-validation error

---

7. Establishing a Baseline Level of Performance

Learning Objectives

- Understand the importance of baseline performance metrics
- Learn to use human-level performance as a reference
- Master quantitative bias-variance assessment

Theoretical Concepts

Baseline performance provides context for interpreting training and cross-validation errors. Common baselines include:

Types of Baselines:

- **Human-level performance**: For tasks humans can do well
- **Competing algorithms**: Published results or existing systems
- **Simple heuristics**: Basic rules or statistical methods

Avoidable vs Unavoidable Bias:

- **Bayes Error**: Theoretical minimum achievable error
- **Avoidable Bias**: J_train - Bayes_error
- **Variance**: J_cv - J_train

Mathematical Formulas and Intuition

Error Analysis Framework:

```
Bayes Error ≤ Human Error ≤ Training Error ≤ CV Error
```

Quantitative Assessment:

```
Avoidable Bias = J_train - Baseline
Variance = J_cv - J_train
```

If Avoidable Bias > Variance: Focus on reducing bias
If Variance > Avoidable Bias: Focus on reducing variance

Practical Tips

- Establish realistic baselines early in project
- Use domain expertise to set appropriate baselines
- Consider multiple baseline metrics for comprehensive assessment

Quick Summary/Insights

- Baseline performance provides context for error interpretation
- Compare avoidable bias vs variance to prioritize improvements
- Human-level performance often serves as a practical baseline

---

8. Learning Curves

Learning Objectives

- Understand how to create and interpret learning curves
- Learn to diagnose bias vs variance using learning curves
- Master the relationship between data size and model performance

Theoretical Concepts

Learning curves plot training and cross-validation error against training set size, revealing important insights about model behavior:

High Bias (Underfitting) Pattern:

- Both curves converge to a high error value
- Large gap between curves closes quickly
- More data doesn't help much

High Variance (Overfitting) Pattern:

- Large gap between training and CV error persists
- Training error increases with more data
- CV error decreases with more data
- More data likely to help

Mathematical Formulas and Intuition

Learning Curve Generation:

```
For m in [100, 200, 400, 800, 1600, ...]:
    Train on first m examples
    Compute J_train(m) and J_cv(m)
    Plot both against m
```

Code Implementation

```python
import matplotlib.pyplot as plt

def plot_learning_curves(X_train, y_train, X_cv, y_cv, model):
    train_sizes = [50*i for i in range(1, len(X_train)//50)]
    train_errors, cv_errors = [], []

    for size in train_sizes:
        model.fit(X_train[:size], y_train[:size])

        train_pred = model.predict(X_train[:size])
        cv_pred = model.predict(X_cv)

        train_errors.append(mean_squared_error(y_train[:size], train_pred))
        cv_errors.append(mean_squared_error(y_cv, cv_pred))

    plt.plot(train_sizes, train_errors, 'r-', label='Training Error')
    plt.plot(train_sizes, cv_errors, 'b-', label='CV Error')
    plt.legend()
    plt.xlabel('Training Set Size')
    plt.ylabel('Error')
```

Practical Tips

- Use learning curves to decide if more data will help
- Look for convergence patterns to diagnose bias vs variance
- Consider computational cost when collecting more data

Quick Summary/Insights

- Learning curves reveal whether more data will improve performance
- High bias: curves converge high (more data won't help much)
- High variance: large persistent gap (more data likely helps)

---

9. Deciding What to Try Next Revisited

Learning Objectives

- Connect bias-variance diagnosis to specific improvement strategies
- Learn systematic approach to model improvement

- Master the decision tree for ML debugging

Theoretical Concepts

Based on bias-variance diagnosis, different strategies are appropriate:

High Bias (Underfitting) Solutions:

- Add more features
- Add polynomial features
- Decrease regularization (λ)
- Try more complex model architecture

High Variance (Overfitting) Solutions:

- Get more training examples
- Try smaller sets of features
- Increase regularization (λ)
- Simplify model architecture

Practical Tips

- Always diagnose before implementing solutions
- Try one solution at a time to measure impact
- Keep track of what works and what doesn't

Quick Summary/Insights

- Systematic diagnosis leads to targeted solutions
- High bias → increase model complexity or features
- High variance → get more data or reduce complexity

---

10. Bias/Variance and Neural Networks

Learning Objectives

- Apply bias-variance concepts to neural network training
- Understand how network architecture affects bias-variance tradeoff
- Learn neural network-specific diagnostic techniques

Theoretical Concepts

Neural networks offer unique flexibility in addressing bias-variance tradeoff:

Neural Network Advantages:

- Can reduce bias and variance simultaneously (with enough data)
- Architecture choices affect complexity
- Regularization techniques (dropout, early stopping)

Key Considerations:

- Larger networks generally have lower bias
- More data helps reduce variance
- Regularization prevents overfitting

Practical Tips

- Start with simpler architectures and gradually increase complexity
- Use dropout and other regularization techniques
- Monitor both training and validation metrics during training

Quick Summary/Insights

- Neural networks can address both bias and variance with proper tuning
- Larger networks typically reduce bias (if not overfitting)
- Use regularization to control variance in complex networks

---

11. Iterative Loop of ML Development

Learning Objectives

- Understand the systematic ML development process
- Learn how to prioritize improvements effectively
- Master the feedback loop for continuous improvement

Theoretical Concepts

ML development follows an iterative cycle:

1. **Choose Architecture**: Model type, features, hyperparameters
2. **Train Model**: Fit on training data
3. **Diagnostics**: Bias-variance analysis, error analysis

4. **Improve**: Based on diagnostic insights
5. **Repeat**: Continue until satisfactory performance

Example - Email Spam Classification:

- Feature selection (word presence, email metadata)
- Model choice (logistic regression, neural network)
- Performance evaluation and iteration

Practical Tips

- Set clear success criteria upfront
- Use diagnostics to guide each iteration
- Track experiments and results systematically
- Focus on one improvement at a time

Quick Summary/Insights

- ML development is inherently iterative and diagnostic-driven
- Use systematic feedback loops rather than random experimentation
- Each iteration should be informed by previous results

---

12. Error Analysis

Learning Objectives

- Learn systematic error analysis methodology
- Understand how to identify improvement opportunities
- Master manual inspection techniques for misclassified examples

Theoretical Concepts

Error analysis involves manually examining misclassified examples to identify patterns and prioritize improvements:

Error Analysis Process:

1. **Collect Misclassified Examples**: From CV set (never test set)
2. **Categorize Errors**: Group by common characteristics
3. **Quantify Impact**: Count errors in each category

4. **Prioritize**: Focus on categories with highest potential impact

Example Categories (Spam Classification):

- Pharmaceutical spam
- Phishing attempts
- Deliberate misspellings
- Unusual email routing

Code Implementation

python

```python
def error_analysis(predictions, y_true, X_text, categories):
    misclassified_idx = np.where(predictions != y_true)[0]

    category_counts = {cat: 0 for cat in categories}

    for idx in misclassified_idx:
        email_text = X_text[idx]
        # Manual categorization logic here
        for category in categories:
            if category_condition(email_text, category):
                category_counts[category] += 1

    total_errors = len(misclassified_idx)
    for cat, count in category_counts.items():
        print(f"{cat}: {count}/{total_errors} ({100*count/total_errors:.1f}%)")
```

Practical Tips

- Focus on CV set errors, not training set
- Look for patterns, not individual examples
- Quantify potential impact before implementing fixes
- Consider multiple annotators for subjective categorization

Quick Summary/Insights

- Manual error analysis reveals patterns missed by automated metrics
- Prioritize improvements based on error frequency and fixability
- Focus on CV set to avoid overfitting to specific examples

---

13. Adding Data

Learning Objectives

- Learn strategic approaches to data collection
- Understand data augmentation techniques
- Master data synthesis methods

Theoretical Concepts

Adding data is most effective when done strategically based on error analysis:

Data Addition Strategies:

1. **Targeted Collection**: Focus on error-prone categories
2. **Data Augmentation**: Transform existing examples
3. **Data Synthesis**: Generate new examples programmatically

Data Augmentation Examples:

- **Images**: Rotation, scaling, cropping, color changes
- **Audio**: Speed changes, noise addition, pitch shifting
- **Text**: Synonym replacement, back-translation

Code Implementation

```python
from torchvision import transforms

# Image data augmentation
augmentation = transforms.Compose([
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(0.5),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0))
])

# Apply augmentation
augmented_images = [augmentation(img) for img in original_images]
```

Practical Tips

- Use error analysis to guide data collection priorities
- Ensure augmented data maintains label correctness
- Balance augmentation to avoid introducing bias
- Validate that synthetic data improves real performance

Quick Summary/Insights

- Strategic data addition is more effective than random collection
- Data augmentation can multiply training data effectively
- Focus on categories identified through error analysis

---

14. Transfer Learning: Using Data from a Different Task

Learning Objectives

- Understand transfer learning principles and benefits
- Learn when and how to apply transfer learning
- Master fine-tuning techniques for pre-trained models

Theoretical Concepts

Transfer learning leverages knowledge from a pre-trained model on a large dataset to improve performance on a related task with limited data:

Transfer Learning Process:

1. **Pre-training**: Large model trained on massive dataset (e.g., ImageNet)
2. **Feature Extraction**: Use pre-trained features as input to new classifier
3. **Fine-tuning**: Adjust pre-trained weights for new task

When Transfer Learning Helps:

- Limited data for new task
- Tasks are related (similar input types)
- Pre-trained model learned relevant features

Code Implementation

```python
import torch
import torchvision.models as models

# Load pre-trained model
pretrained_model = models.resnet50(pretrained=True)

# Freeze early layers (feature extraction)
for param in pretrained_model.parameters():
    param.requires_grad = False

# Replace final layer for new task
num_classes = 10  # For new task
pretrained_model.fc = torch.nn.Linear(pretrained_model.fc.in_features, num_classes)

# Fine-tune (unfreeze some layers if needed)
for param in pretrained_model.layer4.parameters():
    param.requires_grad = True
```

Practical Tips

- Start with feature extraction, then try fine-tuning
- Use lower learning rates for fine-tuning
- Consider which layers to freeze/unfreeze based on task similarity
- Validate that transfer learning improves over training from scratch

Quick Summary/Insights

- Transfer learning leverages large-scale pre-training for data-limited tasks
- Most effective when tasks share similar low-level features
- Can dramatically reduce training time and data requirements

---

15. Full Cycle of a Machine Learning Project

Learning Objectives

- Understand the complete ML project lifecycle
- Learn about deployment and production considerations
- Master MLOps principles and practices

Theoretical Concepts

A complete ML project encompasses much more than model training:

Project Phases:

1. **Scoping**: Define problem, success metrics, resources
2. **Data**: Collection, labeling, organization
3. **Modeling**: Training, evaluation, iteration
4. **Deployment**: Production systems, APIs, scaling
5. **Monitoring**: Performance tracking, maintenance

MLOps Considerations:

- **Deployment**: Inference servers, edge deployment
- **Monitoring**: Data drift, model degradation
- **Maintenance**: Retraining, versioning, rollback

Practical Tips

- Plan for the full lifecycle from project start
- Invest in monitoring and maintenance infrastructure
- Consider deployment constraints during model development
- Build reproducible pipelines for model updates

Quick Summary/Insights

- ML projects require much more than just model training
- Production deployment introduces new challenges and requirements
- MLOps practices are essential for long-term project success

---

16. Fairness, Bias, and Ethics

Learning Objectives

- Understand ethical considerations in ML systems
- Learn about different types of bias and their impacts
- Master approaches for building fair and responsible AI

Theoretical Concepts

ML systems can perpetuate or amplify societal biases, making ethical considerations crucial:

Types of Bias:

- **Historical Bias**: Training data reflects past discrimination
- **Representation Bias**: Some groups underrepresented in data
- **Measurement Bias**: Different quality data for different groups
- **Evaluation Bias**: Inappropriate benchmarks or metrics

Fairness Approaches:

- **Individual Fairness**: Similar individuals treated similarly
- **Group Fairness**: Equal outcomes across demographic groups
- **Demographic Parity**: Equal positive prediction rates
- **Equalized Odds**: Equal true/false positive rates across groups

Practical Tips

- Assemble diverse teams for broader perspective
- Audit data and models for bias regularly
- Develop mitigation strategies proactively
- Monitor deployed systems for fairness issues
- Engage with affected communities

Quick Summary/Insights

- Ethical considerations are paramount in ML system development
- Bias can enter at multiple stages: data, modeling, deployment
- Proactive measures and diverse teams help build fairer systems
- Continuous monitoring and mitigation are essential

Conclusion

This comprehensive guide covers the essential concepts from Andrew Ng's Deep Learning Specialization Course 2, Week 3. The systematic approach to machine learning development - from initial model evaluation through production deployment - provides a robust framework for building successful ML systems.

Key Takeaways:

- Use systematic diagnostics (bias-variance analysis) to guide improvements
- Follow iterative development cycles with clear feedback loops
- Consider the full project lifecycle, not just model accuracy
- Prioritize ethical considerations throughout development

Remember: successful machine learning is about systematic problem-solving, not random experimentation. Use these concepts and techniques to build robust, fair, and effective ML systems.