STAT 4830: Numerical optimization for data science and ML

Lecture 0: Introduction

Professor Damek Davis

Course overview

- **Focus**: Numerical optimization for data science and ML
- **Tools**: PyTorch, Python, occasional use of other libraries
- **Format**: Lecture-based
- **Final Project**: Incrementally developed throughout semester

Prerequisites

- Basic calculus and linear algebra (Math 2400)
- Basic probability (Stat 4300)
- Python programming experience
- No advanced optimization/ML background needed

Why PyTorch?

- Modern auto-differentiation frameworks drive deep learning success
- Enables rapid experimentation with:
    - New model architectures and
    - Novel optimization algorithms
- More flexible than traditional solver-based tools

Preview: spam classification

Let's start with a practical example:

- How do we automatically filter spam emails?
- Demonstrates core optimization concepts
- Shows PyTorch in action

How computers read email

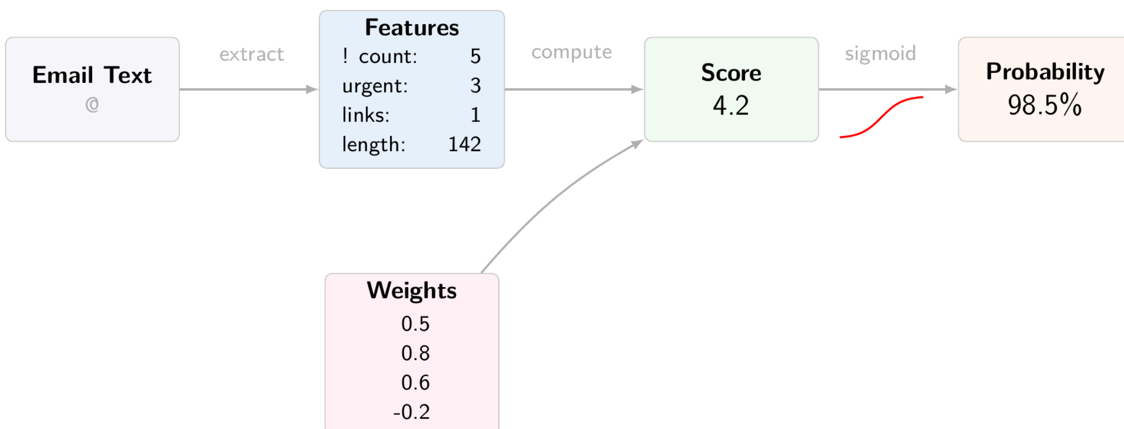
```
email1 = """
Subject: URGENT! You've won $1,000,000!!!
Dear Friend! Act NOW to claim your PRIZE money!!!
"""

email2 = """
Subject: Team meeting tomorrow
Hi everyone, Just a reminder about our 2pm sync.
"""
```

Feature extraction

Convert text to numbers:

```python
def extract_features(email):
    features = {
        'exclamation_count': email.count('!'),
        'urgent_words': len(['urgent', 'act now', 'prize']
                          & set(email.lower().split())),
        'suspicious_links': len([link for link in email.split()
                                if 'www' in link]),
        'time_sent': email.timestamp.hour,
        'length': len(email)
    }
    return features
```

Classification process

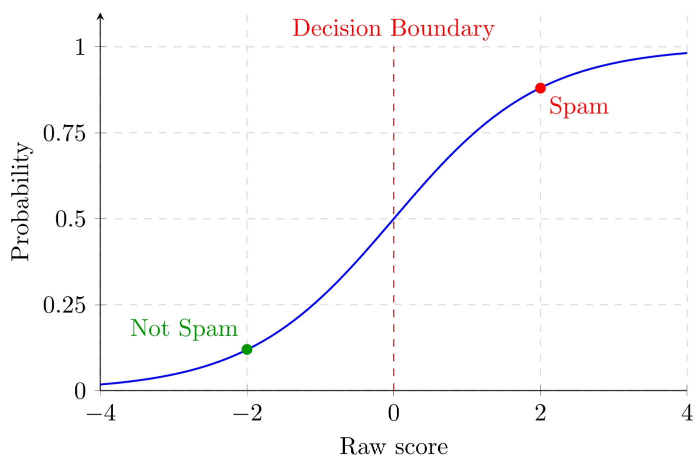


1. Extract numeric features
2. Multiply by weights
3. Sum weighted features
4. Convert to probability

The sigmoid function

Converts any number into a probability (0-1):

```
def sigmoid(x):
    return 1 / (1 + torch.exp(-x))
```



Mathematical formulation

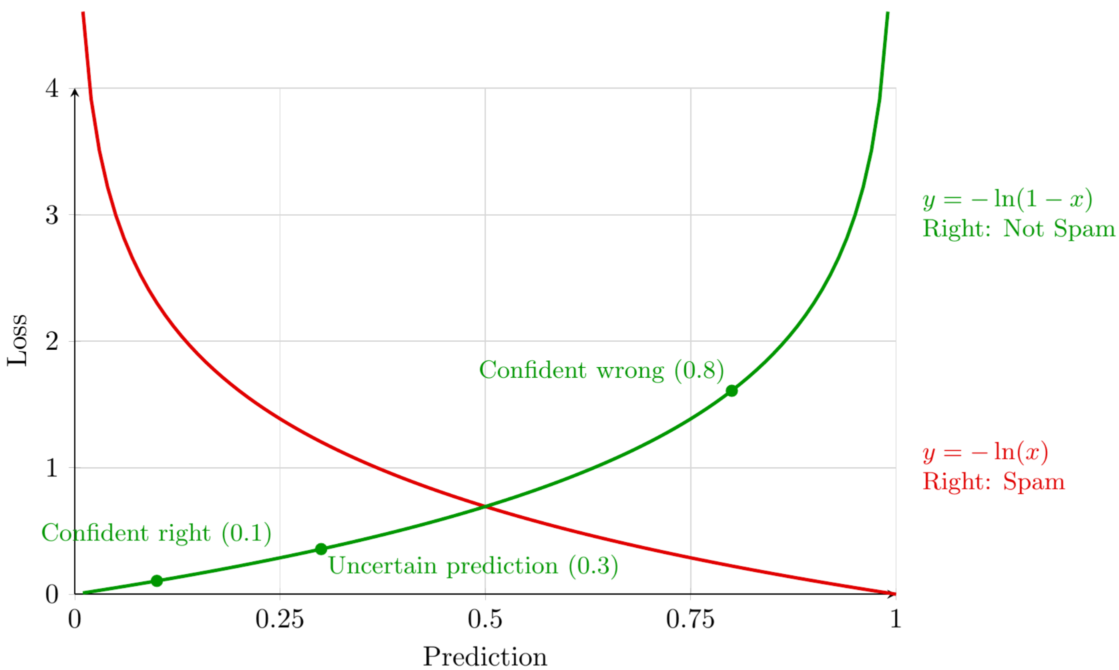
Our optimization problem:

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \left[-y_i \log(\sigma(x_i^\top w)) - (1 - y_i) \log(1 - \sigma(x_i^\top w)) \right]$$

Where:

- w = weights vector
- x_i = feature vector
- y_i = true label (0/1)
- σ = sigmoid function

Cross-entropy loss

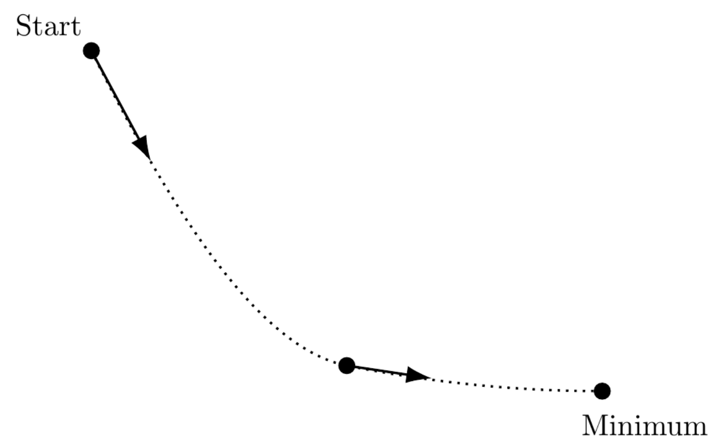


- Penalizes wrong predictions

How Gradient Descent Works

The optimization process works like hiking:

1. Look around you (measure gradient)
2. Take a step downhill
3. Repeat until you reach the bottom



The optimization loop

Each iteration:

1. **Measure** how well current weights classify emails
2. **Calculate** gradient (direction of steepest error reduction)
3. **Update** weights by stepping in this direction
4. **Repeat** until convergence

The learning rate controls step size:

- Too small → slow progress
- Too large → overshooting

PyTorch: What, how, and why

What: Modern framework for optimization and deep learning

How:

- Tracks operations in a computational graph
- Automatically computes gradients
- Enables parallel computation (CPU/GPU)

Why:

- Automates the hardest part (gradients)
- Makes experimentation fast
- Scales from simple to complex models

Inside PyTorch: Tensors and autograd

```
# Tensors: The building blocks
x = torch.tensor([1.0, 2.0], requires_grad=True)
y = x * 2
z = y.sum()

# Automatic differentiation
z.backward()  # Computes gradients
print(x.grad)  # Shows ∂z/∂x
```

PyTorch builds a graph of operations, enabling automatic gradient computation.

Implementation in PyTorch

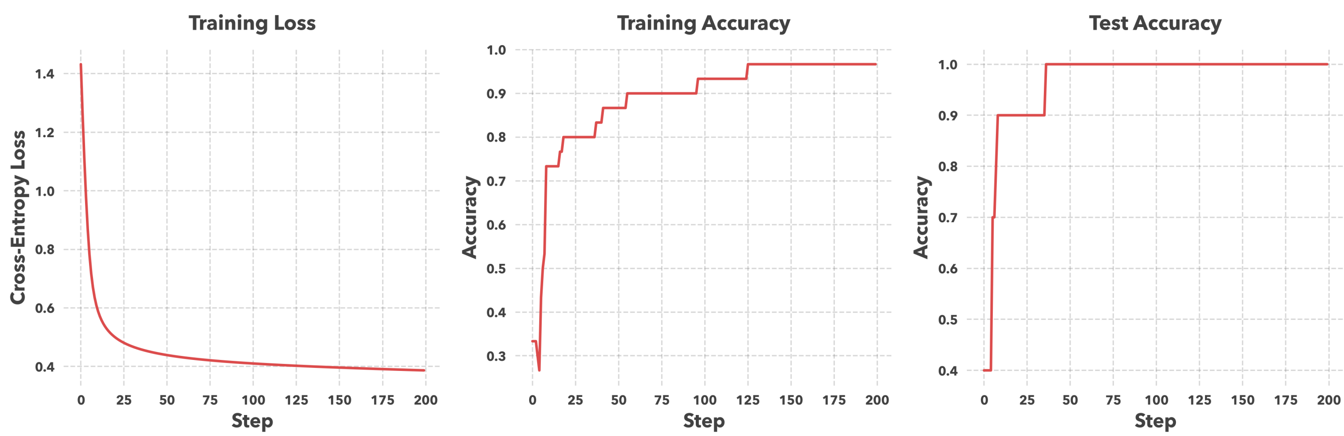
```python
# Initialize
weights = torch.randn(5, requires_grad=True)
learning_rate = 0.01

for _ in range(1000):
    # Forward pass
    predictions = spam_score(features, weights)
    loss = cross_entropy_loss(predictions, true_labels)

    # Backward pass
    loss.backward()

    # Update weights
    with torch.no_grad():
        weights -= learning_rate * weights.grad
        weights.grad.zero_()
```

Try it yourself!

Open in Colab

- Complete implementation in the notebook
- Experiment with different learning rates
- See how the loss changes during training
- Test the model on new emails

Training results



Three key metrics:

- **Loss** and **Training accuracy**: Performance on known data.
- **Test accuracy**: Performance on new emails

Course structure

1. Linear algebra & direct methods
2. Problem formulations & classical software
3. Calculus for optimization
4. Automatic differentiation & PyTorch
5. First-order methods
6. Second-order methods
7. Advanced topics
8. Modern deep learning practice

Learning outcomes

By course end, you'll be able to:

1. Model real problems as optimization problems
2. Select appropriate algorithms
3. Implement solutions in PyTorch
4. Apply optimization to practical problems
5. Conduct optimization research

Getting started

- Review the syllabus
- Set up Python environment
- Try the Colab notebook
- Start thinking about project ideas

Questions?

- Course website: https://damek.github.io/STAT-4830/
- Office hours: Listed on the course website
- Email: damek@wharton.upenn.edu
- Discord: Check email for invite.