

PART 1 OF 3

Mastering LLMs

Part 1: AI & Machine Learning Foundations

A 3-Day Intensive Course

Day 1: Python for Data Science & ML Fundamentals

Day 2: Neural Networks & Deep Learning

Day 3: From Deep Learning to LLMs

← Previous

Next →

The Complete Learning Path

Part 1: Foundations

(This Course)

- Python for ML
- ML Fundamentals
- Neural Networks
- Intro to LLMs

Part 2: LLM Development

(Coming Next)

- Transformers Deep Dive
- RAG Pipelines
- AI Agents
- Fine-tuning

Part 3: Production

(Advanced)

- Optimization
- Deployment
- Ethics & Safety
- Enterprise Scale

What You Already Know

Your Background

- Basic Python programming
- Angular / .NET development
- General programming concepts
- Working with APIs

What You'll Learn

- Data science libraries
- Machine learning concepts
- Neural network fundamentals
- How LLMs actually work

No prior ML/AI experience required! We start from the fundamentals and build up.

DAY 1

Python for Data Science & ML Fundamentals

Module 0: Python Refresher

Module 1: Python Data Science Stack

Module 2: Introduction to Machine Learning

Module 3: Supervised Learning Basics

Module 0: Python Refresher

Quick review of Python fundamentals for ML

Data Types

```
# Numbers
x = 42          # int
y = 3.14        # float

# Strings
name = "AI"
msg = f"Hello {name}"

# Boolean
is_ready = True

# None (null equivalent)
result = None
```

Collections

```
# List (mutable, ordered)
nums = [1, 2, 3, 4, 5]
nums.append(6)

# Dictionary (key-value)
person = {"name": "Alice", "age": 30}

# Tuple (immutable)
point = (10, 20)

# Set (unique values)
unique = {1, 2, 3}
```

Control Flow & Functions

Conditionals & Loops

```
# If-else
if score > 90:
    grade = "A"
elif score > 80:
    grade = "B"
else:
    grade = "C"

# For loop
for i in range(5):
    print(i)

# While loop
while count < 10:
    count += 1

# List comprehension
squares = [x**2 for x in range(10)]
```

Functions

```
# Basic function
def greet(name):
    return f"Hello, {name}!"

# Default arguments
def power(x, n=2):
    return x ** n

# Lambda (anonymous function)
double = lambda x: x * 2

# Multiple return values
def stats(nums):
    return min(nums), max(nums)

low, high = stats([1, 5, 3])
```

Classes & Object-Oriented Python

Essential for understanding PyTorch models

Python: Classes

```
class Model:
    def __init__(self, name, learning_rate=0.01):
        self.name = name                  # Instance attribute
        self.lr = learning_rate
        self.weights = []

    def train(self, data):
        """Train the model on data."""
        print(f"Training {self.name}...")
        # Training logic here

    def predict(self, x):
        return x * 2 # Simplified prediction

# Create instance
my_model = Model("LinearRegression", learning_rate=0.001)
my_model.train(data)
result = my_model.predict(5) # Returns 10
```

Why OOP matters: PyTorch neural networks inherit from `nn.Module` - understanding classes is key!

Useful Python Patterns for ML

Unpacking

```
# Tuple unpacking
x, y = (10, 20)

# Extended unpacking
first, *rest = [1,2,3,4]
# first=1, rest=[2,3,4]

# Dict unpacking
config = {"lr": 0.01}
train(**config)
```

Iteration

```
# enumerate
for i, val in enumerate(lst):
    print(f"{i}: {val}")

# zip
for a, b in zip(x, y):
    print(a, b)

# dict iteration
for k, v in d.items():
    print(k, v)
```

Common Ops

```
# Type conversion
int("42")      # 42
float("3.14")  # 3.14
str(100)       # "100"

# Slicing
lst[1:4]       # items 1-3
lst[::-2]      # every 2nd
lst[::-1]      # reverse
```

Module 1: NumPy - The Foundation

NumPy provides efficient array operations essential for ML

Python: NumPy Basics

```
import numpy as np

# Create arrays
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2, 3], [4, 5, 6]])

# Array operations (vectorized - very fast!)
doubled = arr * 2          # [2, 4, 6, 8, 10]
squared = arr ** 2         # [1, 4, 9, 16, 25]

# Statistical operations
mean = np.mean(arr)        # 3.0
std = np.std(arr)          # 1.414...
```

```
# Matrix multiplication (crucial for ML!)
```

```
result = np.dot(matrix, arr[:3]) # [14, 32]
```

Why NumPy Matters for ML

Speed

NumPy operations are 10-100x faster than Python loops because they're implemented in C.

```
# Slow Python loop
```

```
result = []
for x in data:
    result.append(x * 2)
```

ML Operations

- Matrix multiplication
- Broadcasting
- Random number generation
- Linear algebra

All neural networks are fundamentally matrix operations!

```
# Fast NumPy  
result = data * 2
```

Pandas - Data Manipulation

Pandas makes working with structured data intuitive

Python: Pandas DataFrames

```
import pandas as pd  
  
# Create a DataFrame  
df = pd.DataFrame({  
    'name': ['Alice', 'Bob', 'Charlie'],  
    'age': [25, 30, 35],  
    'salary': [50000, 60000, 70000]  
})  
  
# Common operations  
df['age'].mean()          # Average age: 30  
df[df['salary'] > 55000]   # Filter rows  
df.groupby('age').mean()    # Group and aggregate
```

```
# Load data from files
df = pd.read_csv('data.csv')
df = pd.read_json('data.json')
```

Matplotlib - Visualization

Visualizing data is crucial for understanding and debugging ML models

Python: Basic Plotting

```
import matplotlib.pyplot as plt

# Line plot
plt.plot(x, y, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Training Progress')
plt.legend()
plt.show()

# Scatter plot (for seeing data relationships)
plt.scatter(features, targets, alpha=0.5)
```

```
# Histogram (for data distribution)  
plt.hist(data, bins=30)
```

Module 2: What is Machine Learning?

Machine Learning: Teaching computers to learn patterns from data, rather than explicitly programming rules.

Traditional Programming

Rules + Data → Program → Output

Machine Learning

Data + Output → ML → Rules

You write the rules explicitly

The algorithm discovers the rules

Types of Machine Learning

Supervised Learning

Learn from labeled examples

- Classification (spam/not spam)
- Regression (price prediction)

You provide: inputs + correct answers

Unsupervised Learning

Find patterns in unlabeled data

- Clustering (customer segments)
- Dimensionality reduction

You provide: inputs only

Reinforcement Learning

Learn through trial and error

- Game playing
- Robotics

Agent learns from rewards

The ML Workflow

1. Collect Data → 2. Prepare Data → 3. Choose Model → 4. Train → 5. Evaluate
→ 6. Deploy

Data Preparation (80% of work!)

- Clean missing values
- Handle outliers

Model Training

- Feed training data to model
- Model learns patterns

- Normalize/scale features
- Split into train/test sets
- Adjust parameters to minimize error
- Validate on held-out data

Module 3: Linear Regression

The simplest ML algorithm - find the best-fit line through data

$$y = mx + b \quad \text{or} \quad y = w_1x_1 + w_2x_2 + \dots + b$$

Python: Linear Regression with scikit-learn

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

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

```
# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Check performance
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, predictions)
```

Classification

Predict discrete categories instead of continuous values

Binary Classification

- Spam / Not Spam
- Fraud / Legitimate
- Cat / Dog

Multi-class Classification

- Digit recognition (0-9)
- Sentiment (pos/neg/neutral)
- Image categories

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
probabilities = model.predict_proba(X_test) # Confidence scores
```

Evaluating Models

Regression Metrics

- **MSE:** Mean Squared Error
- **RMSE:** Root MSE
- **MAE:** Mean Absolute Error
- **R²:** Coefficient of determination

Classification Metrics

- **Accuracy:** % correct predictions
- **Precision:** True positives / predicted positives
- **Recall:** True positives / actual positives

- **F1 Score:** Harmonic mean of precision & recall

Key Concept: Always evaluate on data the model hasn't seen (test set)!

The Overfitting Problem

Underfitting

Model is too simple

- High training error
- High test error

Overfitting

Model is too complex

- Low training error
- High test error

- Misses patterns in data

- Memorizes noise

Goal: Find the sweet spot - complex enough to capture patterns, simple enough to generalize

DAY 2

Neural Networks & Deep Learning

Module 4: Neural Network Fundamentals

Module 5: Training Neural Networks

Module 6: Deep Learning with PyTorch

Module 4: What is a Neural Network?

Inspired by biological neurons, but really just math!

Input → [Weights × Inputs + Bias] → Activation Function → Output

```
import numpy as np

def neuron(inputs, weights, bias):
    # Weighted sum
    z = np.dot(inputs, weights) + bias

    # Activation function (sigmoid)
    output = 1 / (1 + np.exp(-z))
    return output

# Example
inputs = np.array([0.5, 0.3, 0.2])
weights = np.array([0.4, 0.6, 0.8])
bias = 0.1

result = neuron(inputs, weights, bias) # 0.64...
```

Activation Functions

Non-linear functions that allow neural networks to learn complex patterns

Sigmoid

ReLU

Tanh

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Output: 0 to 1

Good for: probabilities

$$f(x) = \max(0, x)$$

Output: 0 to ∞

Most common today

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Output: -1 to 1

Zero-centered

Why non-linear? Without activation functions, stacking layers would just be linear algebra. Non-linearity enables learning complex patterns!

Network Architecture

Input Layer \rightarrow Hidden Layer(s) \rightarrow Output Layer

Components

- **Layers:** Groups of neurons
- **Weights:** Connection strengths
- **Biases:** Offset terms
- **Activations:** Non-linearities

Why "Deep" Learning?

- Multiple hidden layers
- Each layer learns higher-level features
- Layer 1: edges → Layer 2: shapes → Layer 3: objects

Module 5: Training - Loss Functions

Loss measures how wrong our predictions are

Regression: Mean Squared Error

Classification: Cross-Entropy

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Penalizes large errors more

$$\text{CE} = - \sum_i y_i \log(\hat{y}_i)$$

Measures probability difference

Python: Loss Functions

```
import torch.nn as nn

# For regression
mse_loss = nn.MSELoss()

# For classification
ce_loss = nn.CrossEntropyLoss()

# Calculate loss
loss = mse_loss(predictions, targets)
```

Gradient Descent

How we minimize loss by adjusting weights

Intuition: Imagine you're on a mountain in fog. To get down, feel which way is steepest and step that direction. Repeat until you reach the bottom.

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \nabla L$$

- **Gradient:** Direction of steepest increase in loss
- **Learning Rate:** Size of steps (hyperparameter)
- **Goal:** Find weights that minimize loss

Backpropagation

Algorithm to compute gradients efficiently through the network

Forward Pass: Input → Predictions → Loss

Backward Pass: Loss → Gradients → Update Weights

Python: Training Loop

```
# Training loop
for epoch in range(num_epochs):
    # Forward pass
    predictions = model(inputs)
    loss = loss_function(predictions, targets)

    # Backward pass
    optimizer.zero_grad()    # Clear old gradients
    loss.backward()           # Compute gradients
    optimizer.step()          # Update weights
```

Module 6: PyTorch Basics

PyTorch is the leading deep learning framework (used by OpenAI, Meta, Tesla...)

Python: PyTorch Tensors

```
import torch

# Create tensors (like NumPy arrays, but GPU-capable)
x = torch.tensor([1.0, 2.0, 3.0])
y = torch.randn(3, 3) # Random 3x3 matrix

# Operations
z = x + y[0]
product = torch.matmul(y, x)

# Move to GPU (if available)
if torch.cuda.is_available():
    x = x.cuda()

# Automatic differentiation!
x = torch.tensor([2.0], requires_grad=True)
y = x ** 2
y.backward()
print(x.grad) # dy/dx = 2x = 4.0
```

Building Networks in PyTorch

Python: Neural Network Class

```
import torch.nn as nn

class SimpleNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.layer2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

# Create model
model = SimpleNetwork(input_size=10, hidden_size=64, output_size=2)

# Create optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Complete Training Example

Python: Full Training Loop

```
model = SimpleNetwork(10, 64, 2)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training
for epoch in range(100):
    for batch_x, batch_y in dataloader:
        # Forward
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if epoch % 10 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
```

DAY 3

From Deep Learning to LLMs

Module 7: Natural Language Processing Basics

Module 8: Attention & Transformers

Module 9: Introduction to LLMs

Module 7: NLP Fundamentals

How do we convert text to numbers that neural networks can process?

Text Processing Pipeline

1. Tokenization (split into words/pieces)
2. Vocabulary mapping (word → number)
3. Embedding (number → vector)

Example

"Hello world" →
["Hello", "world"] →
[42, 156] →
[[0.2, 0.8, ...], [0.5, 0.3, ...]]

Word Embeddings

Dense vector representations that capture meaning

Key Insight: Similar words have similar vectors!

King - Man + Woman ≈ Queen

Python: Using Embeddings

```
import torch.nn as nn

# Create embedding layer
# 10000 words in vocabulary, 256-dimensional vectors
embedding = nn.Embedding(num_embeddings=10000, embedding_dim=256)

# Convert word indices to vectors
word_indices = torch.tensor([42, 156, 789]) # "Hello world !"
word_vectors = embedding(word_indices) # Shape: (3, 256)
```

Module 8: The Attention Mechanism

The breakthrough that enabled modern LLMs

Problem: In the sentence "The cat sat on the mat because it was tired", what does "it" refer to?

Solution: Attention lets the model look at all words and decide which are relevant.

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

- **Query:** What am I looking for?
- **Key:** What do I contain?

- **Value:** What information do I provide?

The Transformer Architecture

"Attention Is All You Need" (2017) - The paper that changed everything

Key Innovations

- Self-attention (each word attends to all others)
- Parallel processing (not sequential)
- Positional encoding (knows word order)
- Multi-head attention (multiple perspectives)

Why It Matters

- Handles long sequences
- Trains much faster
- Captures complex relationships
- Foundation of ALL modern LLMs

Module 9: What are LLMs?

Large Language Models are transformers trained on massive text data

The Scale

- GPT-3: 175 billion parameters
- GPT-4: ~1.7 trillion parameters
- Training data: trillions of words
- Training cost: millions of dollars

How They Work

- Predict the next token
- Learn patterns from text
- Emergent capabilities at scale
- In-context learning

Using LLM APIs

Python: OpenAI API

```
from openai import OpenAI

client = OpenAI(api_key="your-key")

response = client.chat.completions.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Explain neural networks simply."}
    ],
    temperature=0.7,  # Creativity (0=deterministic, 1=creative)
    max_tokens=150     # Response length limit
)

print(response.choices[0].message.content)
```

Prompt Engineering Basics

How you ask matters as much as what you ask

Bad Prompt

"Summarize this"

Vague, no context, unclear format

Good Prompt

"Summarize the following article in 3 bullet points, focusing on key findings for a technical audience:"

Specific format, clear audience, defined scope

Prompt Engineering Tips

- Be specific about format and length
- Provide context and examples
- Use system prompts to set behavior
- Break complex tasks into steps

What's Next? Part 2 Preview

Now that you understand the foundations, Part 2 will cover:

Advanced LLM Techniques

- Transformer deep dive
- RAG (Retrieval-Augmented Generation)
- AI Agents & Tools
- Chain-of-Thought prompting

Customization

- Fine-tuning with LoRA
- Building custom agents
- Vector databases
- Production deployment

Course Summary

Day 1

Python Data Science

- NumPy, Pandas, Matplotlib
- ML fundamentals
- Supervised learning

Day 2

Deep Learning

- Neural networks
- Training & backprop
- PyTorch basics

Day 3

NLP & LLMs

- Text processing
- Attention & Transformers
- Using LLM APIs

You now have the foundation to understand and work with LLMs!

