Deep Learning Fundamentals I

Course:
INFO-6146 Tensorflow & Keras with Python

Developed by:
Mohammad Noorchenarboo

May 5, 2025

Current Section

Motivating Problem: Can We Make Machines Think?

Imagine this: You're building a medical assistant that helps doctors diagnose diseases faster and more accurately.

Challenge: How can a machine learn to make predictions, recognize images, understand language, or make decisions?

Why This Is Hard

Humans learn from experience. Machines need a different approach – they need data and algorithms.

Let's explore the foundational terms and how they relate: AI, ML, and DL.

What Is Artificial Intelligence (AI)?

Artificial Intelligence (AI) is the broad science of mimicking human abilities.

Definition

AI refers to systems or machines that mimic human intelligence to perform tasks and can iteratively improve themselves based on the information they collect.

Examples of AI:

- Voice assistants (e.g., Siri, Alexa)
- Fraud detection systems
- Self-driving cars

What Is Machine Learning (ML)?

Machine Learning (ML) is a subset of AI that enables machines to learn from data without being explicitly programmed.

Example

If you want to predict housing prices, an ML model can learn from features like location, size, and year built to make predictions.

Key Idea

ML uses algorithms to identify patterns in data and make decisions or predictions.

What Is Deep Learning (DL)?

Deep Learning (DL) is a subset of ML that uses neural networks with multiple layers to model complex patterns.

Example

DL powers applications like facial recognition, speech-to-text systems, and advanced language models (like ChatGPT).

Key Insight

The “deep” in deep learning refers to the number of layers in a neural network.

Example: A simple neural network with Keras

```
model = Sequential()
model.add(Dense(64, activation='relu'))
model.add(Dense(1))  # output layer
```

AI vs ML vs DL: A Hierarchical View

Relationship:

- AI is the broadest field: mimics human behavior
- ML is a subset of AI: learns from data
- DL is a subset of ML: learns using deep neural networks

Visual Analogy

AI $\supset$ ML $\supset$ DL

Real-World Mapping

- **AI:** A chatbot that mimics conversation
- **ML:** A spam filter that learns from email data
- **DL:** An image classifier that recognizes cats and dogs

Summary Table: AI, ML, and DL

Aspect	AI	ML	DL
Definition	Broad science	Learning from data	Neural networks with layers
Example	Chess-playing bot	Spam filter	Image recognition
Data Requirement	Low to moderate	Moderate	High
Hardware Needs	Varies	Moderate	High (GPUs)

Tip

Start with ML for simpler tasks. Use DL when the data is large and patterns are complex.

When to Use What?

AI: When you need systems to behave intelligently (rule-based or data-driven).
ML: When you have structured data and want to learn patterns.
DL: When you have large datasets and unstructured data (e.g., images, audio).

Warning

DL is powerful but requires more data, compute, and tuning. It's not always the right choice.

Current Section

What Makes Machine Learning Hard?

Building ML systems is not just about coding. It's about making the right decisions.

Problem

Many ML models fail to generalize well to unseen data. Why?

Challenges arise from:

- Poor data quality or insufficient data
- Overfitting or underfitting the model
- Choosing the wrong algorithm
- Lack of interpretability or explainability

Overfitting vs. Underfitting

- **Overfitting:** The model memorizes the training data but fails on new data.
- **Underfitting:** The model is too simple and fails to capture patterns in training data.

Overfitting Danger

A highly complex model can achieve near-perfect accuracy on training data and still perform poorly on real-world data.

Good Practice

Use validation data, dropout, and regularization to avoid overfitting.

Bias-Variance Tradeoff

Bias: Error due to overly simplistic assumptions.
Variance: Error due to sensitivity to small fluctuations in the training data.

Tradeoff Dilemma

You must balance bias and variance to achieve low generalization error.

Visual Example

High bias = underfitting,
High variance = overfitting

Challenge: Data Quality and Quantity

“Garbage in, garbage out.”

Warning

No ML model can fix bad data. Incomplete, noisy, or biased data will lead to poor results.

Solution

Use data cleaning, augmentation, and proper labeling to ensure quality input.

Challenge: Interpretability

“Why did the model make this prediction?”

Problem

Many deep learning models act as black boxes.

Solution

Use tools like SHAP, LIME, or attention visualization to improve transparency.
For Transformers: Attention heatmaps can highlight which input tokens influenced the output.

Example: A Real Transformer Mistake

Example

A sentiment analysis model based on BERT predicted “This movie was sick!” as negative sentiment.
Why? The model was trained on formal text where “sick” had only negative usage.

This shows:

- Importance of domain-specific training data
- Need for context-aware models

How BERT might be used

```
from transformers import pipeline
classifier = pipeline('sentiment-analysis')
print(classifier("This movie was sick!"))
```

Summary: ML Challenges at a Glance

Challenge	Description
Overfitting	Model learns training data too well, fails on new data
Underfitting	Model too simple, fails to learn even training data
Bias–Variance Tradeoff	Need to find the sweet spot between simplicity and flexibility
Poor Data Quality	Noisy or unbalanced data reduces model effectiveness
Interpretability	Understanding why a model made a decision is often difficult

Current Section

Motivating Problem: How Do Machines Learn?

Suppose we want a machine to recognize handwritten digits.

- Do we need to label thousands of examples?
- Can the model learn patterns by itself?
- What if the model must improve by trial-and-error?

Key Question

What kinds of learning are available, and when should we use each?

Supervised Learning

Supervised learning involves training a model on labeled data – where the correct output is known.

Key Point

The model learns to map inputs to outputs by minimizing error on labeled examples.

Common tasks:

- Classification (e.g., spam detection, disease diagnosis)
- Regression (e.g., predicting house prices)

Simple classification with Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(10,)))
model.add(Dense(1, activation='sigmoid'))
```

Unsupervised Learning

Unsupervised learning uses input data **without labels** to discover hidden patterns.

Example

Clustering similar customer profiles for marketing strategies.

Common tasks:

- Clustering (e.g., K-Means, DBSCAN)
- Dimensionality Reduction (e.g., PCA, t-SNE, autoencoders)

K-Means clustering with scikit-learn

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(X)
```

Reinforcement Learning

Reinforcement learning (RL) involves agents learning to make decisions by interacting with an environment.

Key Idea

The agent learns through rewards and penalties over time.

Applications:

- Game playing (e.g., AlphaGo, Dota 2)
- Robotics
- Recommender systems

Example RL environment

```
import gym
env = gym.make('CartPole-v1')
state = env.reset()
action = env.action_space.sample()
```

Types of Learning Algorithms: Comparison Table

Aspect	Supervised Learning	Unsupervised Learning
Input	Labeled data (input + target)	Only input data
Goal	Learn mapping from input to output	Discover structure or groupings
Common Algorithms	Linear regression, decision trees, CNNs	K-Means, PCA, autoencoders
Output	Predictions	Clusters, embeddings, patterns

Where Does Reinforcement Learning Fit?

Reinforcement Learning	Agent interacts with environment, receives feedback as rewards, and updates its strategy.
Key Terms	State, action, reward, policy, value function
Example Tools	OpenAI Gym, RLlib, Stable Baselines

Fun Fact

RL beat world champions in Go, StarCraft II, and Dota 2 using deep neural networks.

Current Section

What Is Feature Engineering?

Feature engineering is the process of selecting, transforming, or creating new input variables to improve model performance.

Common techniques:

- Normalization, encoding, binning
- Creating interaction terms
- Dimensionality reduction

Example

In a dataset of house sales, you might create a new feature: “price per square foot” or “age of house”.

Why Feature Engineering Is Hard

Limitations

- Requires deep domain knowledge
- Time-consuming and labor-intensive
- Often involves trial-and-error
- Risk of introducing data leakage

Also: Good features for one model (e.g., linear regression) may not help another (e.g., random forest).

Manual vs. Automated Feature Learning

Classical ML: Heavy reliance on handcrafted features.
Deep Learning: Learns hierarchical representations from raw data.

Advantage of DL

Neural networks can learn complex, non-linear features automatically – especially from unstructured data.

Example

Image classification with CNNs does not require edge detectors or pixel histograms – these are learned during training.

Code Snippet: Manual Feature Engineering (Classical ML)

Using Scikit-learn for basic preprocessing

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer([
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(), categorical_features)
])
X_transformed = ct.fit_transform(X)
```

Code Snippet: Deep Learning Learns Features Automatically

Keras CNN for raw image classification

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
        28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Feature Engineering vs. Representation Learning

Aspect	Classical ML (Manual)	Deep Learning (Automatic)
Feature Creation	Manual, requires domain knowledge	Learned from raw data
Flexibility	Low – features may need redesign per task	High – reuses same architectures across tasks
Data Suitability	Structured/tabular data	Unstructured data (images, audio, text)
Scalability	Moderate – hand-engineered features can be a bottleneck	High – leverages GPUs, large datasets
Example	One-hot encoding, normalization	CNNs, RNNs extract spatial/temporal features

Current Section

Why Tensors?

Tensors are the fundamental data structures used in deep learning frameworks like TensorFlow and PyTorch.

Definition

A tensor is a generalization of scalars (0D), vectors (1D), and matrices (2D) to N-dimensional arrays.

Why use tensors?

- Efficient computation on GPUs
- Unified representation of diverse data types: text, images, sequences
- Flexible batch processing and automatic differentiation

Tensor Rank and Shape

Rank: Number of dimensions (axes)
Shape: Size of each dimension

Examples

- Scalar (0D): ‘x = 42‘
- Vector (1D): ‘x = [1.0, 2.0, 3.0]‘
- Matrix (2D): ‘x = [[1, 2], [3, 4]]‘
- Tensor (3D+): Batch of images with shape ‘(32, 28, 28, 1)‘

Checking tensor rank and shape

```
import tensorflow as tf

x = tf.constant([[1.0, 2.0], [3.0, 4.0]])
print('Rank:', tf.rank(x))          # 2
print('Shape:', x.shape)            # (2, 2)
```

Real-World Data as Tensors

How different data types are represented:

Data Type	Tensor Representation
Scalar	Single number (e.g., temperature: ‘x = 36.6‘)
Vector	1D tensor (e.g., age vector for 10 people: ‘shape = (10,)‘)
Matrix	2D tensor (e.g., grayscale image: ‘shape = (28, 28)‘)
Image	3D tensor: height & width & channels (e.g., ‘shape = (28, 28, 1)‘)
Batch of Images	4D tensor (e.g., 32 images: ‘shape = (32, 28, 28, 1)‘)
Text (Tokenized)	2D tensor (e.g., sentences with fixed word length: ‘shape = (batch, sequence)‘)

Tensor Data Types and Devices

Tensor Attributes:

- **dtype:** data type (e.g., float32, int64)
- **device:** location in memory (CPU, GPU, TPU)

Creating tensors on GPU

```
with tf.device('/GPU:0'):
    a = tf.constant([[1.0, 2.0], [3.0, 4.0]])
    print(a.device)
```

Good Practice

Always check tensor shapes when debugging model errors – many bugs are due to shape mismatches.

Current Section

What Is Optimization in ML?

Optimization is the process of adjusting a model's parameters to minimize a loss function.
Objective:

$$\theta^* = \arg\min_{\theta} L(\theta)$$

Goal

Find the set of weights θ that make the model predictions as close as possible to the true values.

Gradient Descent: The Core Idea

Gradient descent is an iterative optimization algorithm that updates parameters in the opposite direction of the gradient.

Update Rule

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta L(\theta)$$

Where:

- θ – model parameters (e.g., weights and biases)
- η – learning rate (step size)
- $\nabla_\theta L(\theta)$ – gradient of the loss w.r.t. parameters

Caution

If η is too large, you may overshoot the minimum. If it's too small, learning will be slow.

Gradient Descent in Python

Manual gradient descent update

```
w = tf.Variable(2.0)
learning_rate = 0.1

for step in range(100):
    with tf.GradientTape() as tape:
        loss = (w - 5) ** 2  # simple loss

    grad = tape.gradient(loss, w)
    w.assign_sub(learning_rate * grad)
```

Observation

After enough steps, ‘w‘ will converge close to 5 (the minimum of the loss).

Using Optimizers in Keras

Using SGD optimizer in model training

```
model.compile(optimizer='sgd',
              loss='mse',
              metrics=['mae'])

model.fit(X_train, y_train, epochs=10)
```

Other available optimizers: ‘Adam‘, ‘RMSprop‘, ‘Adagrad‘, ‘Nadam‘, etc.

Types of Gradient Descent

Type	Description
Batch Gradient Descent	Computes gradient using the entire training dataset. Stable but memory-heavy.
Stochastic Gradient Descent (SGD)	Updates parameters for each training example. Noisy but faster.
Mini-Batch Gradient Descent	Updates parameters using small batches. Best of both worlds.
Adaptive Methods (Adam, etc.)	Automatically adjust learning rate per parameter. Popular in deep learning.

Common Pitfalls in Gradient Descent

Problems You May Encounter

- Poor learning rate selection
- Getting stuck in local minima or saddle points
- Vanishing/exploding gradients (especially in deep networks)

Good Practice

Use learning rate schedules, gradient clipping, and advanced optimizers to stabilize training.

Current Section

What Is Overfitting?

Overfitting occurs when a model performs well on training data but poorly on unseen data.

Symptoms of Overfitting

- Training loss is low, but validation loss increases
- Model memorizes noise and outliers
- Poor generalization

Goal

Build models that generalize well, not just memorize.

Underfitting vs. Overfitting

Underfitting: Model is too simple, performs poorly on training data.
Overfitting: Model is too complex, performs poorly on validation data.

Condition	Model Behavior
Underfitting	High training loss and high validation loss. Model cannot capture underlying pattern.
Good Fit	Low training and validation loss. Generalizes well.
Overfitting	Low training loss but high validation loss. Model memorizes training data.

What Is Regularization?

Regularization refers to techniques that discourage overfitting by limiting model complexity.

Why It's Needed

Neural networks have many parameters – without constraints, they can easily overfit.

Common methods:

- L1/L2 regularization
- Dropout
- Early stopping
- Data augmentation

L1 and L2 Regularization

L1 (Lasso): Adds absolute value of weights to loss **L2 (Ridge):** Adds squared value of weights to loss

Regularized Loss

$$L_{\text{reg}} = L_{\text{original}} + \lambda \cdot \sum_i w_i^2$$

Adding L2 regularization in Keras

```
from tensorflow.keras import regularizers

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.001)))
```

Dropout Regularization

Dropout: Randomly deactivates neurons during training to prevent co-dependence on specific features.

Key Benefit

Improves robustness and generalization by making the network less reliant on any single neuron.

Adding Dropout to a Keras model

```
from tensorflow.keras.layers import Dropout

model = Sequential([
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

Early Stopping

Early stopping: Stop training when validation performance stops improving.

Keras Implementation

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(patience=3, restore_best_weights=True)
model.fit(X_train, y_train, callbacks=[early_stop])
```

Advantage

Saves time and avoids over-training the model.

Summary: Regularization Techniques

Technique	Description
L1/L2 Regularization	Adds penalty to the loss to constrain weights. Helps shrink unimportant parameters.
Dropout	Randomly drops neurons during training to prevent reliance on specific pathways.
Early Stopping	Stops training once validation loss stops improving. Reduces overfitting risk.
Data Augmentation	Generates additional data samples to reduce overfitting in small datasets.