ISBN: 978-93-7020-738-7

DEEP LEARNING FOR DATA SCIENCE: ARCHITECTURES, ALGORITHMS, AND REAL-WORLD APPLICATIONS

Abstract

Deep learning, a specialized subset of machine learning, employs multi-layered neural networks to autonomously learn hierarchical data representations, eliminating the need for manual feature engineering required in traditional machine learning. Key innovations like convolutional neural networks (CNNs) rev olutionized computer vision through spatial hierarchy learning, while recurrent neural networks (RNNs) enabled sequential data processing for time-series and NLP tasks. The backpropagation algorithm remains central to training these models, optimizing weights via gradient descent while leveraging activation functions (e.g., ReLU, Softmax) to introduce non-linearity. Modern frame- works such as TensorFlow and PyTorch democratize implementation through automatic differentiation and GPU acceleration, supporting architectures like Transformers and GANs that dominate 2025's AI landscape. These advancements power applications ranging from medical image analysis to real-time language translation, with CNNs achieving >98% accuracy in image classification benchmarks and Transformers enabling context-aware chatbots. As deep learning evolves, techniques like mixed-precision training and neuro-symbolic integration address computational and interpretability challenges, solidifying its role in next-generation AI systems [1, 2].

Keywords: Deep learning, neural networks, convolutional networks, recurrent networks, natural language processing

Authors

Shubneet
Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.
jeetshubneet27@gmail.com

Anushka Raj Yadav
Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.
ay462744@gmail.com

Paras Mahajan
Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.
mahajanparas115@gmail.com

Partha Chanda
Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.

Atahar Shihab
Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.
ataharshihab5112@gmail.com

I. INTRODUCTION

The evolution of neural networks represents one of the most transformative journeys in artificial intelligence, transitioning from rudimentary mathematical models to architectures capable of human-like reasoning. Beginning with Frank Rosenblatt's perceptron in 1958-a single-layer network simulating basic biological neurons-the field languished for decades until the 1980s revival of backpropagation. This algorithm enabled multi-layer perceptrons (MLPs) to learn hierarchical representations, laying the groundwork for modern deep learning. The 21st century witnessed exponential growth, with convolutional neural networks (CNNs) revolutionizing computer vision and transformers redefining natural language processing (NLP). Today, architectures like ResNet and GPT-4 demonstrate superhuman performance in specialized tasks, powered by innovations in parallel computation and self-attention mechanisms [3, 4].

From Perceptrons to Parallel Processing

The perceptron's inability to solve non-linear problems limited early progress until backpropagation emerged. By iteratively adjusting weights through gradient descent, networks could now learn complex patterns. The 2010s saw CNNs dominate image recognition, with ResNet's skip connections (2015) enabling unprecedented 1,000-layer networks that achieved 96% accuracy on ImageNet. Simultaneously, recurrent neural networks (RNNs) and long short-term memory (LSTM) networks advanced sequential data processing, though their sequential computation remained inefficient.

The 2017 transformer architecture marked a paradigm shift. By replacing recurrence with self-attention, models like BERT and GPT could process entire text sequences in parallel while capturing long-range dependencies. This innovation fueled NLP breakthroughs: GPT-3 generates human-like text with 175 billion parameters, while vision transformers (ViTs) now rival CNNs in image classification. These advancements stem from three core drivers: (1) exponential growth in computational power, (2) availability of massive labeled datasets, and (3) theoretical innovations in network design.

Landmark Architectural Innovations

- **ResNet (2015):** Introduced residual learning with skip connections, solving vanishing gradients in deep CNNs. Enabled networks exceeding 1,000 layers while improving accuracy.
- **Transformers (2017):** Scaled self-attention mechanisms for parallel processing, achieving state-of-the-art results in translation (BLEU score >40) and text generation.
- **GPT Series (2018–2024):** Autoregressive transformers pretrained on web-scale text data, demonstrating few-shot learning capabilities.
- **Vision Transformers (2020):** Applied transformer principles to image patches, matching CNN performance on ImageNet with 85% fewer parameters

Chapter Outline

This chapter systematically explores deep learning through:

- **Foundational Concepts:** Perceptrons, activation functions, and backpropagation
- **Neural Network Architectures:** CNNs, RNNs, and transformer blocks

- **Training Techniques:** Optimization, regularization, and transfer learning
- **Framework Ecosystems:** TensorFlow, PyTorch, and Keras
- **Applications:** Medical imaging diagnostics, real-time speech translation, and autonomous systems
- Ethical considerations in deploying deep learning systems

As neural networks evolve toward multimodal reasoning and embodied AI, their ability to synthesize vision, language, and sensory data heralds a new era of general- purpose intelligence. The following sections provide both theoretical frameworks and practical tools to harness these revolutionary technologies.

II. NEURAL NETWORK FUNDAMENTALS

Perceptron Mathematics: The fundamental building block of neural networks is the perceptron, which computes a weighted sum of inputs with a bias term:

$$z = \mathbf{w}^T \mathbf{x} + b$$

where:

- $\mathbf{w}$ = weight vector (w_1, w_2, ..., w_n)
- $\mathbf{x}$ = input vector (x_1, x_2, ..., x_n)
- b = bias term

Layer Types

- **Dense Layers:** Fully connected layers where each neuron connects to all inputs
- **Convolutional Layers:** Use kernel filters for spatial pattern recognition in images/video
- **Recurrent Layers:** Process sequential data through memory cells (LSTM/-
- GRU)

Activation Functions

$$\text{ReLU}(z) = \max(0, z) \quad \text{(Hidden layers)}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{(Binary classification)}$$

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad \text{(Multi-class output)}$$

3-Layer MLP in Keras

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

model = Sequential ([
        Dense (128 , activation =' relu ', input_shape =(784 ,)),
```

```
        Dense (64 , activation =' relu '),
        Dense (10 ,  activation =' softmax ')
])
model. compile ( optimizer=' adam ',
        loss=' categorical_crossentropy ',
        metrics =[' accuracy '])
```

Feedforward Network Architecture

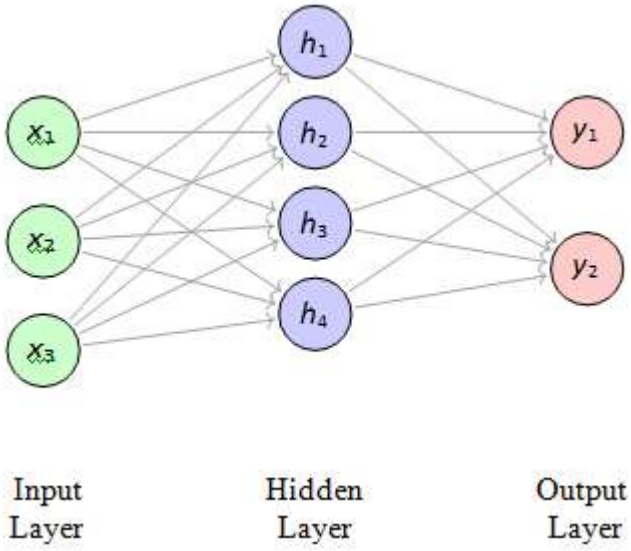

Figure 1: Feedforward neural network with 3 inputs, 4 hidden neurons, and 2 outputs

Function	Range	Derivative	Use Case
ReLU	$[0, \infty)$	0 if $z < 0$ 1 if $z \geq 0$	Hidden layers, CNN backbone
Sigmoid	(0, 1)	$\sigma(z)(1 - \sigma(z))$	Binary classification output
Softmax	(0, 1)	Complex matrix form	Multi-class classifica- tion
Tanh	$(-1, 1)$	$1 - \tanh^2(z)$	RNN hidden states, normalization

III. BACKPROPAGATION AND OPTIMIZATION

Gradient Descent and Chain Rule: The backpropagation algorithm calculates error gradients through neural networks using the chain rule from calculus. For a loss function L, the gradient of weight w(l) in layer l is:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{\partial L}{\partial a_i^{(l+1)}} \cdot \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}}$$

where z represents pre-activations and a denotes layer outputs. Batch gradient descent updates weights as:

$$\frac{\partial L}{\partial w^{(l)}} = \delta^{(l)} \cdot (a^{(l-1)})^T$$

with η as learning rate.

Common Optimizers

- **Adam:** Combines momentum and adaptive learning rates:

$$m_t = \beta_1 m_t - 1 + (1 - \beta_1) g_t \quad ; \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_2$$

- **RMSProp:** Adjusts learning rates per parameter using moving average of squared gradients:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g^2$$

Vanishing/Exploding Gradients

Deep networks suffer from unstable gradients due to:

- **Vanishing:** Small derivatives from activation functions (e.g., sigmoid) compound in deep layers

- **Exploding:** Large weight matrices amplify gradients exponentially Solutions include ReLU activations, batch normalization, and gradient clipping.

CNN Training on MNIST

```
from tensorflow . keras import layers , models

model = models. Sequential ([
        layers. Conv2D (32 , (3 ,3), activation =' relu ', input_shape
                =(28 ,28 ,1)),
        layers. Max Pooling 2 D ((2 ,2)),
        layers. Flatten (),
        layers. Dense (128 , activation =' relu '),
        layers. Dense (10 , activation =' softmax ')
])

model. compile ( optimizer=' adam ',
        loss=' sparse_categorical_crossentropy ',
        metrics =[' accuracy ']) 
model. fit( train_images , train_labels , epochs =10 , validation_split =0.2)
```

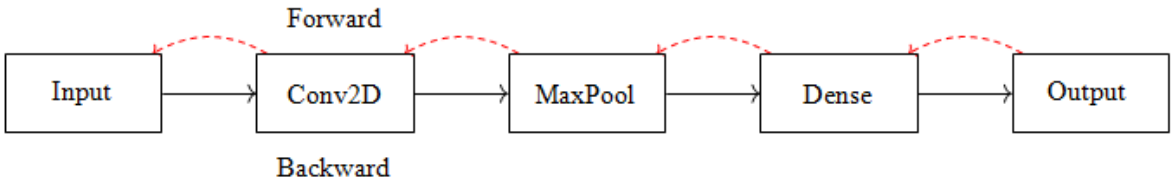

Figure 2: Backpropagation flow through CNN layers (red dashed lines)

Optimizer Performance Comparison

Table 1: Optimizer Characteristics on MNIST

Optimizer	Train Acc	Val Acc	Time/Epoch
SGD	92.4%	91.1%	45s
Adam	98.2%	97.8%	52s
RMSProp	97.9%	97.5%	55s

IV. CNNS FOR COMPUTER VISION

Convolutional Neural Networks (CNNs) have fundamentally transformed computer vision by enabling machines to automatically learn hierarchical feature representations from raw image data. Unlike traditional approaches that rely on handcrafted features, CNNs leverage convolutional layers to detect spatial patterns such as edges, textures, and complex shapes, making them highly effective for tasks like image classification, object detection, and segmentation.

Convolutional Layers and Pooling

CNNs are built from two primary types of layers: convolutional and pooling. The convolutional layers apply a set of learnable filters (kernels) that slide over the input image to produce feature maps. Mathematically, the convolution operation for an input image I and kernel K is defined as:

$$(I * K)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n) \cdot K(m,n)$$

where i, j denote spatial positions, and m, n index the kernel dimensions. Each filter is trained to activate in response to specific visual features, such as vertical or horizontal edges.

Pooling layers reduce the spatial dimensions of feature maps, providing translation invariance and reducing computational complexity. Max pooling, the most common type, selects the maximum value within a local window:

$$P_{max}(x, y) = \max_{(i,j)\in R} I(x + i, y + j)$$

where R is the pooling region. This process helps retain the most salient features while discarding redundant information.

CNN Architectures

Over the past decades, several influential CNN architectures have been introduced:

- **LeNet-5 (1998):** One of the earliest CNNs, designed for handwritten digit recognition. It consists of two convolutional layers followed by average pooling and fully connected layers. LeNet-5 achieved remarkable accuracy on the MNIST dataset and inspired future research.
- **AlexNet (2012):** Marked a breakthrough in large-scale image classification, winning the ImageNet competition by a large margin and popularizing the use of ReLU **activations and dropout regularization.**
- **VGG-16 (2014):** Demonstrated the effectiveness of deeper networks with small 3× 3 filters, leading to improved accuracy at the cost of increased parameters.
- **ResNet-50 (2015):** Introduced residual connections to mitigate the vanishing gradient problem, enabling training of very deep networks with over 50 layers.

ResNet-50 achieved state-of-the-art performance on ImageNet and is widely used in transfer learning.

TensorFlow Implementation Example

```
import tensorflow as tf
from tensorflow . keras import layers , models

model = models. Sequential ([
        layers. Conv2D (32 , (3 ,3), activation =' relu ', input_shape
                =(32 ,32 ,3)),
        layers.  Max Pooling 2 D  ((2  ,2)),
        layers. Conv2D (64 , (3 ,3), activation =' relu '),
        layers.  Max Pooling 2 D  ((2  ,2)),
        layers. Flatten (),
        layers. Dense (128 , activation =' relu '),
        layers. Dense (10 , activation =' softmax ')
])

model. compile ( optimizer=' adam ',
                loss=' sparse_categorical_crossentropy ',
                metrics =[' accuracy '])
```

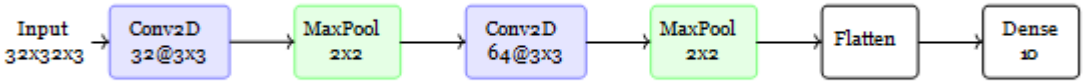

Figure 3: Feature extraction and classification flow in a typical CNN.

CNN Architecture Comparison

Table 2: Comparison of Popular CNN Architectures

Model	Year	Layers	Parameters	Top-1 Acc.
LeNet-5	1998	7	0.06M	99.2%
AlexNet	2012	8	60M	63.3%
VGG-16	2014	16	138M	71.5%
ResNet-50	2015	50	25.6M	76.0%

CNNs have become the backbone of many computer vision applications, from facial recognition to autonomous driving, due to their ability to learn complex features and generalize well to new data [5] [6].

V. RNNS FOR SEQUENTIAL DATA

Recurrent Neural Networks (RNNs) specialize in processing sequential data by maintaining hidden states that capture temporal dependencies. Unlike feedforward networks, RNNs reuse parameters across time steps, making them ideal for time-series analysis, NLP, and speech recognition.

LSTM and GRU Gates

Long Short-Term Memory (LSTM) networks address vanishing gradients in vanilla RNNs through gated memory cells:

- **Forget Gate:** Decides what information to discard

$$f_t = \sigma(W_f \bullet [h_{t-1}, x_t] + b_f)$$

- **Input Gate:** Updates cell state with new information

$$i_t = \sigma(W_i \bullet [h_{t-1}, x_t] + b_i)$$

- **Cell State Update:**

$$\tilde{C}_t = \tanh(W_C \bullet [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- **Output Gate:** Controls hidden state exposure

$$o_t = \sigma(W_o \bullet [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

Gated Recurrent Units (GRUs) simplify LSTMs by combining forget/input gates into an update gate z_t and merging cell/hidden states.

Applications in Time-Series Forecasting

- Stock price prediction using historical OHLC data
- Energy load forecasting for smart grids
- Weather pattern modeling with sensor data
- Anomaly detection in IoT device streams

Text Generation with PyTorch

```
import torch
import torch . nn as nn
class  CharLSTM ( nn. Module ):
        def _init_( self , vocab_size , hidden_size ):
                super (). _init_()
                self. lstm = nn. LSTMCell( vocab_size , hidden_size )
                self. fc = nn. Linear( hidden_size , vocab_size )

def forward ( self , x, hc):
         h, c = self. lstm (x, hc)
        return self. fc( h), (h, c)

# Sample usage
model = CharLSTM ( vocab_size =128 , hidden_size =256)
input_seq = torch . randn (32 , 128)       # Batch of 32 sequences
h, c = torch . zeros (32 , 256) , torch . zeros (32 , 256)
output , (h, c) = model( input_seq , (h, c))
```

LSTM Architecture Diagram

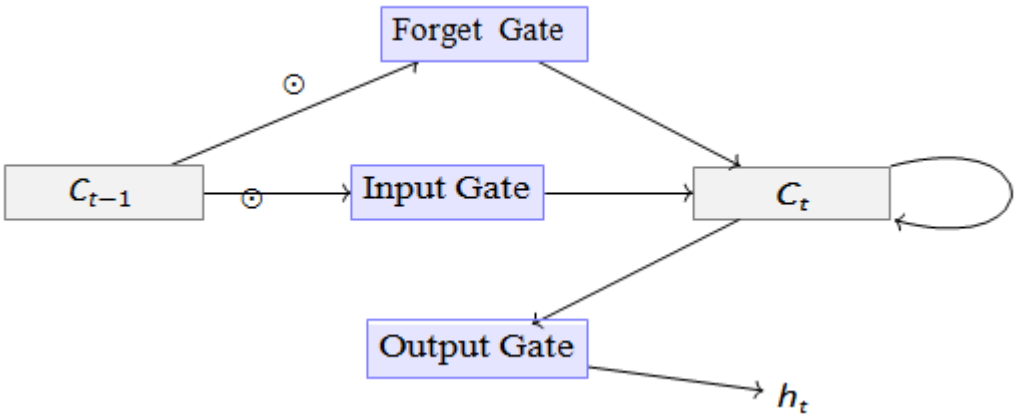

Figure 4: LSTM cell architecture with gating mechanisms

RNN Variants Comparison

Table 3: Comparison of RNN Architectures

Type	Gates	Parameters	Training Stability
Vanilla RNN	0	Low	Poor
LSTM	3	High	Excellent
GRU	2	Medium	Good

RNNs power applications requiring temporal awareness, from real-time translation to predictive maintenance [7, 8].

VI. DEEP LEARNING FRAMEWORKS

The rapid growth of deep learning has been fueled by powerful frameworks that sim- plify model development, training, and deployment. Among these, TensorFlow and PyTorch are the most widely adopted, each with distinct philosophies and strengths. Keras, as a high-level API, further streamlines deep learning workflows, particularly for beginners and rapid prototyping.

TensorFlow vs. PyTorch: Static vs. Dynamic Graphs

TensorFlow is built around a static computation graph paradigm, where the entire model architecture is defined before execution. This approach enables advanced graph optimizations, efficient parallelization, and is well-suited for large-scale, production- grade deployments. TensorFlow's ecosystem includes tools for distributed training, visualization (TensorBoard), and robust deployment options such as TensorFlow Serv- ing and TensorFlow Lite. However, static graphs can slow experimentation and make debugging less intuitive for newcomers [9].

PyTorch, in contrast, uses dynamic computation graphs (define-by-run). Models are constructed and modified on the fly, making PyTorch exceptionally flexible and "pythonic." This dynamic nature is ideal for research, rapid prototyping, and tasks involving variable-length inputs or custom architectures. PyTorch's eager execution model allows seamless integration with native Python control flow, making debugging and experimentation more straightforward. While deployment tools like TorchServe have improved, TensorFlow remains stronger for production at scale.

Keras: High-Level API for Productivity

Keras is a high-level, user-friendly API that runs on top of TensorFlow (and previously supported Theano and CNTK). It offers three ways to build models: the Sequential API (for simple, linear stacks of layers), the Functional API (for complex, multi- input/output architectures), and Model Subclassing (for full customization). Keras is designed for fast experimentation, code readability, and accessibility, making it an excellent choice for both beginners and professionals. With TensorFlow 2.x, Keras is tightly integrated as tf.keras, combining ease of use with TensorFlow's scalability and deployment capabilities [10].

Code Example: Identical MLP in Keras vs. PyTorch

TensorFlow/Keras

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

model = Sequential ([
        Dense (64 , activation =' relu ', input_shape =(100 ,)),
        Dense (10 , activation =' softmax ')
])
model. compile ( optimizer=' adam ',  loss=' categorical_crossentropy ')
```

PyTorch:

```
import torch
import torch . nn as nn

class MLP ( nn. Module ):
        def _init_( self):
                super (). _init_()
                self. fc1 = nn. Linear (100 , 64)
                self. relu  =  nn. ReLU ()
                self. fc2 = nn. Linear (64 , 10)
                self. softmax = nn. Softmax ( dim =1)
        def forward ( self , x):
                x = self. relu ( self. fc1 ( x))
                x = self. softmax ( self. fc2 ( x))
```

```
        return x
model = MLP ()
```

Framework Comparison Table

Aspect	TensorFlow	PyTorch	Keras
Computation Graph	Static(define-and-run)	Dynamic (define-by-run)	High-level API (on TensorFlow)
Ease of Use	Moderate,improved with Keras	Very Pythonic intuitive,	Extremely user-friendly
Deployment	Excellent (Serving, Lite, JS)	Good (TorchServe)	Excellent via Tensor-Flow
Best For	Production, scalability	Research, prototyping	Rapid prototyping, education
Visualization	TensorBoard	Visdom, TensorBoard support	TensorBoard via tf.keras
Community	Large, mature	Fast-growing, strong in academia	Large, especially for beginners

In summary, PyTorch is favored in research for its flexibility and native Python feel, while TensorFlow dominates production with scalability and deployment tools. Keras bridges both worlds, offering a productive interface for building and deploying deep learning models.

VII. DEEP LEARNING IN PRACTICE

Deep learning architectures have become foundational across industries, enabling breakthroughs in healthcare, speech technology, and natural language processing.

Their ability to learn complex, hierarchical representations from raw data has driven adoption in real-world applications.

CNNs for Medical Imaging Diagnostics

Convolutional Neural Networks (CNNs) have transformed medical image understanding. By learning to detect patterns in X-rays, CT scans, and MRIs, CNNs now assist clinicians in diagnosing diseases such as lung cancer, breast cancer, and heart anomalies. For example, CNN models have achieved diagnostic accuracies rivaling or surpassing human experts in tasks like tumor detection, COVID-19 identification from chest X-rays, and segmentation of brain lesions. These models are routinely used for image classification, localization, and segmentation, helping radiologists make faster and more accurate decisions [11, 12].

RNNs for Speech Recognition

Recurrent Neural Networks (RNNs), particularly when augmented with attention or implemented as encoder-decoder architectures, excel at modeling sequential data such as speech. OpenAI's Whisper, for instance, leverages a transformer-based encoder- decoder

(with RNN-like sequence modeling) to perform robust, multilingual speech recognition. Whisper is trained on hundreds of thousands of hours of diverse audio, enabling it to transcribe speech with near-human accuracy, even in noisy environments or across different languages. This has enabled new levels of accessibility, voice- driven interfaces, and real-time transcription in applications from virtual assistants to automated subtitling [13].

Transformers for NLP

Transformers, and especially models like BERT, have revolutionized natural language processing. Using self-attention mechanisms, transformers capture long-range dependencies and context, enabling state-of-the-art performance on tasks such as text classification, sentiment analysis, question answering, and named entity recognition. BERT, in particular, can be fine-tuned for a wide range of NLP tasks, making it a cornerstone model for chatbots, document search, and language understanding in enterprise and consumer applications [14, 15].

Mapping Architectures to Use Cases

Architecture	Industry	Use Cases
CNN	Healthcare	Tumor detection, organ seg- mentation, COVID-19 diagnosis from X-rays
RNN/Transformer	Speech/Media	Voice assistants, speech-to-text (Whisper), automated captioning
Transformer (BERT)	NLP/Enterprise	Text classification, chatbots, search, sentiment analysis

Deep learning's versatility and accuracy have made it indispensable for extracting insights and automating decision-making across sectors.

Exercises

Python Tasks

1. MNIST CNN Classification

```
# Train a CNN on MNIST dataset
import tensorflow as tf
from tensorflow . keras import layers , models

( x_train , y_train ), ( x_test , y_test) = tf. keras. datasets. mnist. load_data ()
x_train = x_train . reshape (-1 , 28 , 28 , 1). astype (' float32 ') / 255.0
x_test = x_test. reshape (-1 , 28 , 28 , 1). astype (' float32 ') / 255.0

model = models. Sequential ([
        layers. Conv2 D (32 , (3 ,3), activation =' relu ', input_shape
        =(28 ,28 ,1)),
        layers.  Max Pooling 2 D  ((2  ,2)),
        layers. Conv2 D (64 , (3 ,3), activation =' relu '),
```

```
        layers.  Max Pooling 2 D  ((2  ,2)), layers. Flatten (),
        layers. Dense (128 , activation =' relu '), layers. Dense (10 , activation =' softmax ')
])

model. compile ( optimizer=' adam ',
        loss=' sparse_categorical_crossentropy ',
        metrics =[' accuracy '])
model. fit( x_train , y_train , epochs =5 , validation_split =0.2)
```

2. LSTM Text Prediction

```
# Character - level LSTM text generator
import torch
import torch . nn as nn

class  CharLSTM ( nn. Module ):
        def _init_( self , vocab_size , hidden_size ):
                super (). _init_()
                self. embed = nn. Embedding ( vocab_size , 64)
                self. lstm = nn. LSTM (64 , hidden_size , batch_first= True
                        )
                self. fc = nn. Linear( hidden_size , vocab_size )

def forward ( self , x, hidden = None ):
        x = self. embed ( x)
        out , hidden = self. lstm (x, hidden )
        return self. fc( out), hidden

# Example usage
model = CharLSTM ( vocab_size =128 , hidden_size =256)
input_seq = torch . randint (0 , 128 , (16 , 100))          # ( batch_size , seq_length )
output , hidden = model( input_seq )
```

Framework Comparison Task

Implement the same neural network architecture in both TensorFlow/Keras and PyTorch:

- **Input Layer:** 784 dimensions (MNIST flattened)
- **Hidden Layer:** 128 units with ReLU activation
- **Output Layer:** 10 units with softmax
- **Compare:** Model definition syntax, training loops, debugging tools

Mini-Project: Sentiment Analysis

Build an RNN-based sentiment classifier:

- **Dataset:** IMDB movie reviews

- **Preprocessing:** Tokenization, padding/truncating sequences
- **Model:** Embedding → LSTM → Dense layers
- **Evaluation:** Accuracy, precision, recall, ROC-AUC
- **Deployment:** Export as TensorFlow SavedModel or TorchScript

REFERENCES

[1] Lee, S.: Applying Backpropagation in Deep Learning for Enhanced Model Accuracy. https://www.numberanalytics.com/blog/ backpropagation-deep-learning-model-accuracy
[2] iQuanta: Top 5 Deep Learning Algorithms in 2025. https://www.iquanta.in/ blog/top-5-deep-learning-algorithms-in-2025/
[3] Ikomia: Mastering ResNet: Deep Learning Breakthrough in Image Recognition. https://www.ikomia.ai/blog/mastering-resnet-deep-learning-image-recognition
[4] IndustryWired: 10 AI Breakthroughs in Natural Language Processing. https: //industrywired.com/10-ai-breakthroughs-in-natural-language-processing/
[5] Topics, S.: LeNet-5 Architecture Explained. https://www.scaler.com/topics/ lenet/
[6] IBM: Convolutional Neural Networks Guide. https://www.ibm.com/think/ topics/convolutional-neural-networks
[7] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997) https://doi.org/10.1162/neco.1997.9.8.1735
[8] Team, P.: LSTM Documentation. https://pytorch.org/docs/stable/generated/ torch.nn.LSTM.html
[9] OpenCV: PyTorch Vs TensorFlow: Comparative Guide of AI Frameworks 2025. https://opencv.org/blog/pytorch-vs-tensorflow/
[10] Team, K.: Keras Activation Functions. https://keras.io/api/layers/activations/
[11] Shankar, K., Perumal, E., et al.: Convolutional neural networks in medical image understanding. Computers in Biology and Medicine 137, 104835 (2021)
[12] Al-Tarawneh, M., et al.: Diagnosis of medical images using convolutional neural networks. Journal of Engineering Science and Technology Review (2023)
[13] OpenAI: Introducing Whisper. https://openai.com/index/whisper/
[14] Cloud, G.: Transformer Models and BERT Model | Google Cloud Skills Boost. https://cloudskillsboost.google/course_templates/538
[15] Kumar, S.: Overview of Transformer and BERT. https://www.linkedin.com/ pulse/overview-transformer-bert-sanjay-kumar-mba-ms-phd
[16] Labs, V.: Activation Functions in Neural Networks. https://www.v7labs.com/ blog/neural-networks-activation-functions
[17] Schmidt, R.M., Schneider, F., Hennig, P.: Benchmarking deep learning optimiz- ers. arXiv preprint arXiv:1910.05446 (2021)
[18] Brownlee, J.: How to Develop a CNN for MNIST Handwrit- ten Digit Classification. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
[19] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015) https://doi.org/10.1038/nature14539
[20] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
[21] Team, T.: Text Classification with TensorFlow. https://www.tensorflow.org/ tutorials/keras/ text_classification